DIN EN ISO 9001 certified

ADDI-DATA®

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

Technical description

ADDICOUNT PA 1700-2

High-speed counter board

Edition: 05.01 - 1/2006

Copyright

Guarantee and responsibility

Basically are effective our "general terms of delivery and payment". The manager receives them at the latest with the invoice. Claims for guarantee and responsibility in case of injuries and material damages are excluded, if they are due to one or some of the following causes:
- if the board has not been used for the intended purpose
- improper installation, operation and maintenance of the board
- if the board has been operated with defective safety devices or with not appropriate or non-functioning safety equipment
- non-observance of the instructions concerning: transport, storage, inserting the board, use, limit values, maintenance, device drivers
- altering the board at the user's own initiative
- altering the source files at the user's own initiative
- not checking properly the parts which are subject to wear
- disasters caused by the intrusion of foreign bodies and by influence beyond the user's control.

Licence for ADDI-DATA software products

Read carefully this licence before using the standard software. The right for using this software is given to the customer, if he/she agrees to the conditions of this licence.
- this software can only be used for configuring ADDI-DATA boards.
- copying the software is forbidden (except for archiving/ saving data and for replacing defective data carriers).
- deassembling, decompiling, decoding and reverse engineering of the software are forbidden.
- this licence and the software can be transferred to a third party, so far as this party has purchased a board, declares to agree to all the clauses of this licence contract and the preceding owner has not kept copies of the software.

Trademarks

Borland C and Turbo Pascal are registered trademarks of Borland International, INC.
Burr-Brown is a registered trademark of Burr-Brown Corporation
Intel is a registered trademark of Intel Corporation
AT, IBM, ISA and XT are registered trademarks of International Business Machines Corporation
Microsoft, MS-DOS, Visual Basic and Windows are registered trademarks of
Microsoft Corporation

The original version of this manual is in German. You can obtain it on request.

Declaration of Conformity

Document-Number/Month-Year: B-25808 / 02.1999

Manufacturer/Importer: ADDI-DATA GmbH
Dieselstraße 3
D-77833 OTTERSWEIER

Type: **PA 1700-2**

Product description:
Board to be inserted in an ISA slot of a PC
Counting board for 3 to 6 incremental encoders
Frequency measurement and pulse measurement
24 TTL I/Os

The above named product complies with the following European directives:

Directive 72/23/EEC of 19 February 1973 on the harmonization of the laws of Member States relating to electrical equipment designed for use within certain voltage limits.

Directive 89/336/EEC of 3 May 1989 on the approximation of the laws of the Member States relating to electromagnetic compatibility.

The following norms have been applied:

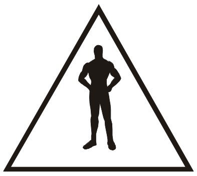
IEC 61010-1 2002-08
IEC 61326-2 2004

2004/11/10
Date

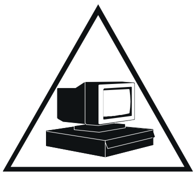
Legally valid signature of the manufacturer

WARNING

In case of improper use and if the board is not used for the intended purpose:



people may be injured



the board, PC and peripheral devices may be destroyed

the environment may be polluted

★★★ Protect yourself, other people and the environment★★★

- **Read the yellow safety leaflet carefully!**
  If this leaflet is not with the documentation, please contact us.
- **Observe the instructions in the manual!**
  Make sure that you do not forget or skip any step. We are not liable for damage resulting from a wrong use of the board.
- **Symbols used**

WARNING!
It designates a possibly dangerous situation.
If the instructions are ignored **the board, PC and/or peripheral devices may be damaged**.

i

IMPORTANT!
designates hints and other useful information.

- **Do you have any question?**

Our technical support is at your disposal

Figures

Tables

1 INTENDED PURPOSE OF THE BOARD

The board **PA 1700-2** is the interface between an industrial process and a personal computer (PC).

It is to be used in a free ISA slot. The PC is to comply with the EU directive 89/336/EEC and the specifications for EMC protection.

Products complying with these specifications bear the normed CE mark.

Data exchange between the board **PA 1700-2** and the peripheral is to occur through a shielded cable. It has to be connected to the 37-pin SUB-D male connector of the board **PA 1700-2**.

The board has **3 or 6 input channels** for 5 V signal acquisition.
The use of the board **PA 1700-2** in combination with external terminal or relay boards is to occur in a closed switch cabinet; the installation is to be effected competently.

Check the shielding capacity of the PC housing and of the cable prior to putting the device into operation.

The connection with our standard cable ST010 complies with the specifications:
- metallized plastic hoods
- shielded cable
- cable shield folded back and firmly screwed to the connector housing.

Uses beyond these specifications are not allowed. The manufacturer is not liable for any damages which would result from the non-observance of this clause.

The use of the board according to its intended purpose includes observing all advice given in this manual and in the safety leaflet.

1.1 Limits of use

Our boards are not to be used for securing emergency stop functions.

The emergency stop functions are to be secured separately.
This securing must not be influenced by the board or the PC.

The use of the board in a PC could change the PC features regarding noise emission and immunity. Increased noise emission or decreased noise immunity could result in the system not being conform anymore.

The board must not operate in a PC which does not provide for an adequate protection against environmental conditions (e.g.: liquids, dust, ...) and which is not equipped with an appropriate aeration system.

Make sure that your PC is equipped with a safety system and a shielded metal housing.
The installation of the board in sites lying under risk of explosion is excluded.
The board should not be submitted to vibrations without additional fixation.

WARNING!
The EMC tests have been carried out in a specific appliance configuration. We guarantee these limit values **only** in this configuration.

The tested appliance configuration is at your disposal on request.

Please control that the PC-housing and the cable are effectively shielded before putting the device into operation.

Make sure that the board remains in its protective blister pack **until it is used**.

Do not remove or alter the identification numbers of the board.
If you do, the guarantee expires.

Consider the professional regulations of your sector.

2 USER

2.1 Qualification

Only persons trained in electronics are entitled to perform the following tasks:

- installation,
- use,
- maintenance.

The basic knowledge of a high-level programming language is sufficient for programming the board.
The knowledge of conversion between the different numerative systems (binary, decimal, hexadecimal) is necessary.

The basic knowledge of EMC is necessary to operate under the CE conformity.

2.2 Personal protection

Consider the country-specific regulations about

- the prevention of accidents
- electrical and mechanical installations
- radio interference suppression

3 HANDLING THE BOARD

Fig. 3-1: Wrong handling

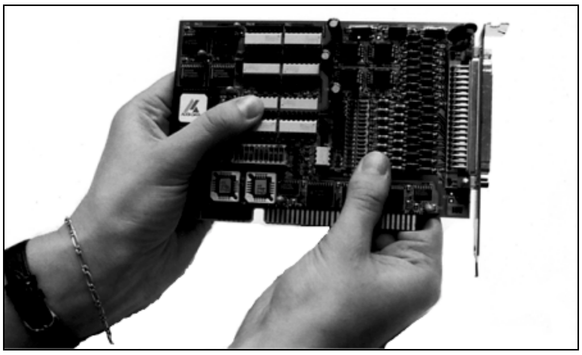
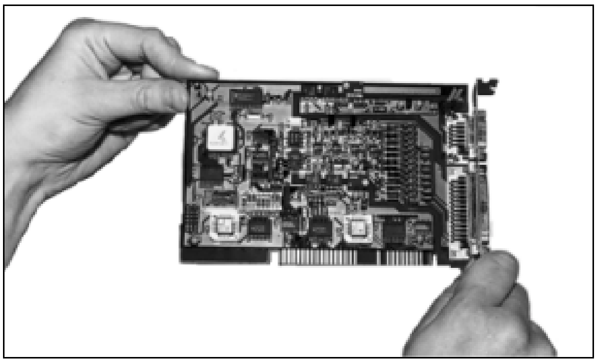


Fig. 3-2: Correct handling



4 TECHNICAL DATA

4.1 Electromagnetic compatibility (EMC)

The board has been subjected to EMC tests in an accredited laboratory in accordance with the norms EN50082-2, EN55011, EN55022.
The board complies as follows with the limit values set by the norm EN50082-2:

	True value	Set value
ESD	4 kV	4 kV
Fields	10 V/m	10 V/m
Burst	4 kV	2 kV
Conducted radio interferences	10 V	10 V

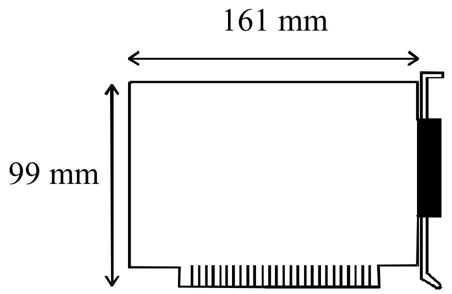
WARNING!
The EMC tests have been carried out in a specific appliance configuration. We guarantee these limit values **only** in this configuration.

Consider the following aspects:
- your test program must be able to detect operation errors.
- your system must be set up so that you can find out what caused errors.

4.2 Physical set-up of the board

The board is assembled on a 6-layer printed circuit card.

External board dimensions:



Weight:	160 g
Installation in:	AT slot
Connection to the peripheral:	37-pin SUB-D male connector
Accessories:	Screw terminal board: PX 901-ZG Cables: ST010/ST011, Ribbon cable FB1700 See connection examples

4.3 Limit values

Operating temperature: .... 0 to 60°C
Storage temperature: .... -25° to +70°C
Relative humidity: .... 30% to 95% non condensing

Minimum PC requirements:
Energy requirements
- Operating voltage of the PC: .... 5V ± 5%
- Current consumption in mA (without load): . typ. See table ± 10 mA

	PA 1700-2
+ 5 V of the PC	150 mA

Differential inputs:
Complies with the EIA standards RS422A
Common mode range: .... ± 7 V
Input sensitivity: .... ± 200 mV
Inputs hysteresis: .... 50 mV (typ.)
Input impedance: .... 12 kΩ (min)
Terminator: .... 100 Ω (typ.)

Cable break recognition at differential signals:
Current consumption for each differential receiver: .... ± 5 mA (with DV = ± 5V)
Minimum differential voltage for the circuit functioning correctly: .... ± 2.5 V
Reaction time of the recognition circuit: .... 1 ms (typ.)

Signal specifications:
Limiting frequency of the counter signals (±UAx; ±UBx): .... 1.25 MHz
Minimum phase shift: .... 45° (at 1.25 MHz)
Minimum INDEX pulse duration: .... 200 ns
Minimum EXTSTBxy pulse duration (low): ... 200 ns

TTL input channels (REFx, EXTSTBxy)
Pull up resistor: .... 10 kΩ
Series resistor: .... 100 Ω
Input logic "0": .... 0.8 V max.
Input logic "1": .... 2.4 V min.
TTL compatible
TTL output channels (TIMER-OUT), pin 37 of the SUB-D connector**:**
TTL compatible
Series resistor: .... 10 Ω
Max. output current: .... ±4 mA

Protection circuitry:
All differential inputs are protected with transil diodes of type BZW06-10B against transients. Their reference point is the PC housing (derivation through the board bracket).

Pin 20 on the front connector:
Voltage supply from PC: .... +5 through microfuse
Max. reliable current: .... 500 mA

Breakdown voltage: .... ± 11.4 V (typ.)
Clamping voltage: .... ± 21.7 V (typ.)
at Ipp1) = 106 A

All the TTL inputs on the SUB-D connector are provided with double Schottky protection diodes.
Voltage reversal protection
Overvoltage protection up to: .... + 15 V (at POWER On)

Digital I/O (82C55PPI)
Compatibility: .... TTL compatible
Configuration: .... Three 8-bit ports
(use 82C55PPI)
Input at logic "0": .... 0.8 V maximum
Input at logic "1": .... 2.0 V minimum
Input load current $0 \leq V_{in} \leq 5$ V .... ± 10 µA maximum
Output at logic "0"
for output current = 1.7 mA .... 0.45 V maximum
Output at logic "1"
for output current = -200 µA .... 2.4 V minimum
Darlington drive current (ports B and C only)
R_{EXT} = 750 Ω; V_{EXT} = 1.5 V .... 1.0 mA minimum,
4.0 mA maximum

---

[1] Surge current

Timer I/O (82C54)

Compatibility:	Three 16-bit counters/timers 82C54
Configuration:	TTL compatible inputs and outputs; Gate circuit (counter gate) is provided with a pull up resistor of 100 kΩ.
Input at logic "0":	0.8 V maximum
Input at logic "1":	2.2 V minimum
Output at logic "0" for output current = 1.6 mA:	0.45 V maximum
Output at logic "1" for output current = -150 µA:	2.4 V minimum
Input capacitance at 1 MHz:	10 pF maximum
Available input frequencies:	892.857 kHz 27.901 kHz

4.4 Conventions

Designation	Value range	Meaning
Counting module X	X =1...3	Integrated circuit which allows to acquire phase shifted signals. Each module contains a 32-bit counter or two 16-bit counters
Counter channels X	X =1...6	Complete electronics for the acquisition of encoders. Up to 6 channels are available for the user

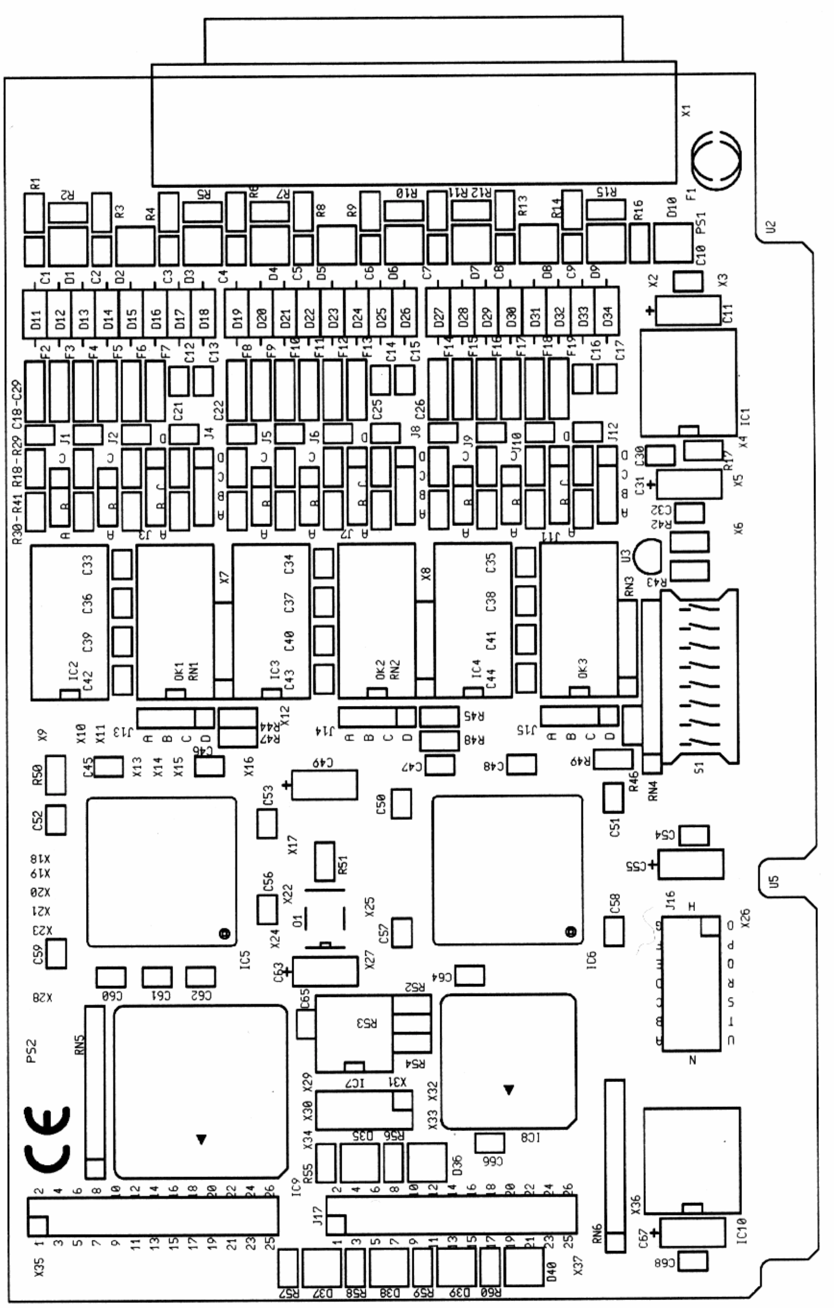
EXTSTBxy	x =1...3
	y = 1..2
		Strobe signal which allows to store the counter content of counting module x in the data latch register y (TTL signal, low active)

INDEXx	x =1...3	Diff. or TTL input which allows to determine a reference point in a counting module
TIMER COMPONENT		Designates the integrated circuit 82C54. It contains three 16-bit timers
TIMERx	x = 0...2	Designates timer x of the timer component

5 SETTINGS

5.1 Component scheme

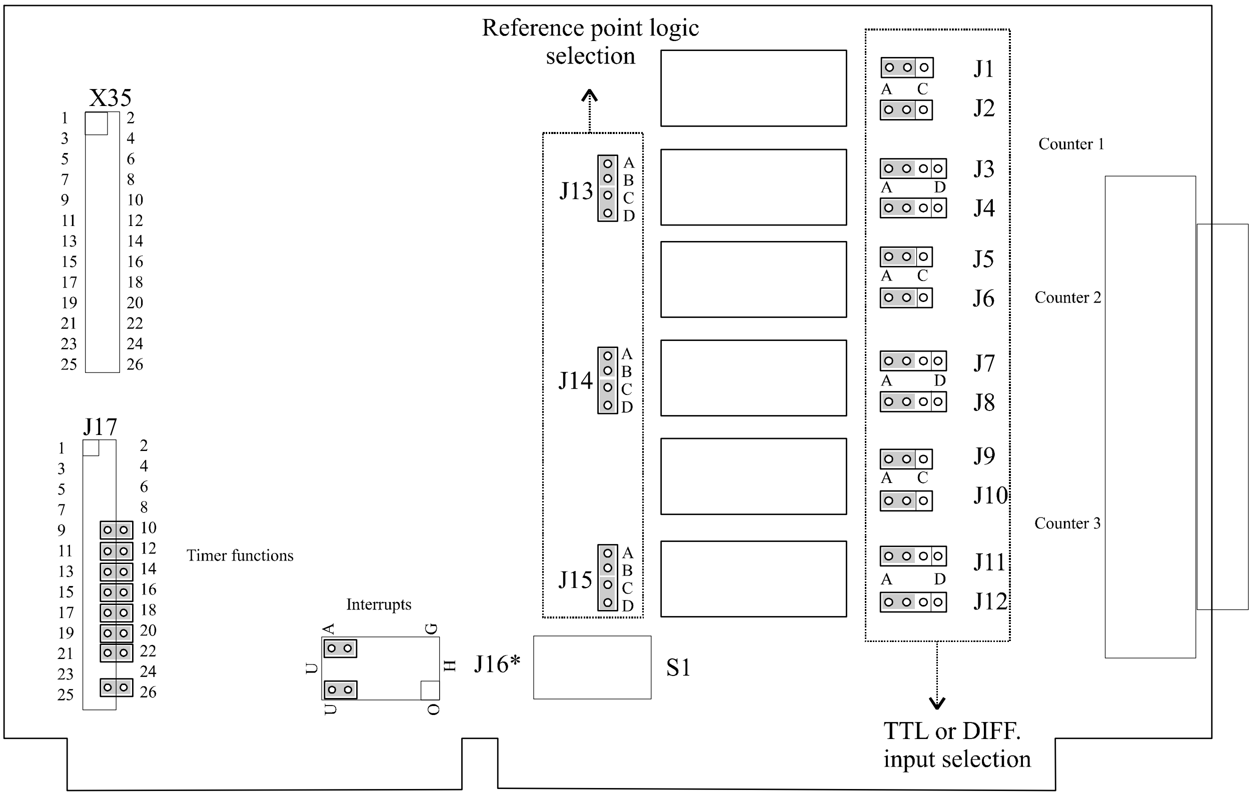
Fig. 5-1: Component scheme of the board PA 1700-2



5.2 Jumper location and settings at delivery

5.2.1 Jumper location on the PA 1700-2 board

Fig. 5-2: Jumper location



* This setting has no function

5.2.2 Jumper settings

Table 5-1: Jumper settings: TTL or differential signals

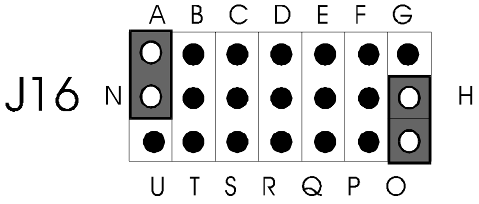
Jumper	Functions	Settings	Settings at delivery
J1, J5, J9	TTL or differential input selection, UA signal	**A-B:** UAx signal in differential mode connected to ± UAx **B-C:** UAx signal in TTL mode, connected to -UAx.	A-B: differential input channels
J2, J6, J10	TTL or differential input selection, UB signal	**A-B:** UBx signal in differential mode connected to ± UBx **B-C:** UBx signal in TTL mode, connected to -UBx	A-B: differential input channels
J3, J7, J11	TTL or differential input selection, INDEX signal	**A-B:** INDEXx signal in differential mode connected to ± INDEXx **B-C:** INDEXx signal in TTL mode, connected to - INDEXx (negative pulse signal) **C-D:** INDEXx signal in TTL mode, connected to + INDEXx (positive pulse signal)	A-B: differential input channels
J4, J8, J12	TTL or differential input selection, UAS signal	**A-B:** UASx signal in differential mode connected to ± UASx **B-C:** UASx signal in TTL mode, connected to UASx (Bit UASx read in STATUS_REG1 and negated reflection of the input) **C-D:** UASx signal in TTL mode, connected to + UASx (Bit UASx read in STATUS_REG1 and reflection of the input)	A-B: differential input channels

Table 5-2: Jumper settings: Reference point logic selection

Jumper	Functions	Settings	Settings at delivery
J13	4-pin wire-wrap for counter 1. Determines the source of the CLEAR or EXTSTBx1 signals	**A-B:** EXTSTBx1 signal of the 37-pin front connector or of the timer component is led to input EXTSTBx1 of the counter module x	A-B and C-D
J14	4-pin wire-wrap for counter 2. Determines the source of the CLEAR or EXTSTBx1 signals	**B-C:** The counter content is stored in data latch register 1 when the INDEXx input becomes active (Reference point logic)	
J15	4-pin wire-wrap for counter 3. Determines the source of the CLEAR or EXTSTBx1 signals	**C-D:** The counter content is cleared when the when the INDEXx input becomes active (Reference point logic)	

Remark: 2 jumpers can be set (positions: A-B and C-D)

Table 5-3: Jumpers settings: Interrupt lines with jumper J16



	TIMER-INTERRUPT	STROBE-INTERRUPT
IRQ3	H-G	H-O
IRQ5	F-I	I-P
IRQ10	E-J	J-Q
IRQ11	D-K	K-R
IRQ12	C-L	L-S
IRQ14	B-M	M-T
IRQ15	A-N	N-U

Table 5-4: Jumper settings: Timer functions with J17

Positions	Timer function	Settings	Remark
21-22	Timer2 as a time interrupt generator	1 MHz clock pulse at input CLK2	Select a free interrupt line through J16
25-26		Output OUT2 is connected to J16 and to the pin 37 of the front connector	
9-10	Timer0 as a time generator for strobe control	OUT0 on external strobe input 1, counter module 1	J13 in position A-B
11-12		OUT0 on external strobe input 1, counter module 2	J14 in position A-B
13-14		OUT0 on external strobe input 1, counter module 3	J15 in position A-B
15-16	Timer1as a time generator for strobe control	OUT0 on external strobe input 2, counter module 1	
17-18		OUT0 on external strobe input 2, counter module 2	
19-20		OUT0 on external strobe input 2, counter module 3	

6 INSTALLATION

i

IMPORTANT!
If you want to install simultaneously **several** ADDI-DATA boards, consider the following procedure.

- **Install and configure** the boards one after the other.
  You will thus avoid configuration errors.

1. Switch off the PC
2. Install the **first** board
3. Start the PC
4. Install the software (once is enough)
5. Configure the board

6. Switch off the PC
7. Install the **second** board
8. Start the PC
9. Configure the board

etc

You will find additional information to these different steps
in the sections 6.1 to 6.5.

i

IMPORTANT!
You have installed already **one or more** ADDI-DATA boards in your PC, and you wish to install **an additional** board?

Proceed as if you wished to install one single board.

6.1 Base address

WARNING!
If the base address set is wrong, the board and/or the PC may be damaged.

Before installing the board

The base address is set at delivery on the address 0300H.

- **Check**, that
  - the base address is free
  - the address range (32 I/O addresses) required by the board is not already used by the PC or by boards already installed in the PC.

The base address and/or the address range **are wrong**?

- **Select** another base address with the 8-pole block of DIP switches S1

The address 0300H is decoded in the following table.

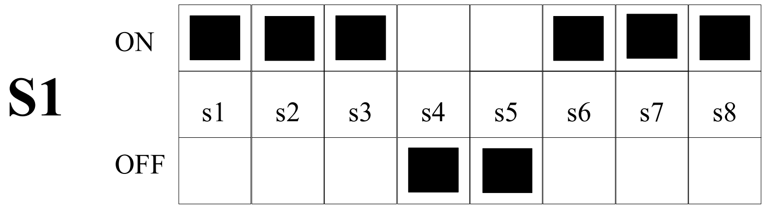
Table 6-1: Decoding table

MSB LSB

Decoded address bus	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Wished base address Hex	0				3				0				0			
Wished base address binary	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
DIP switch S1 Logic "0"= ON Logic "1" = OFF	*	*	*	s8 ON	s7 ON	s6 ON	s5 OFF	s4 OFF	s3 ON	s2 ON	s1 ON	X	X	X	X	X

X: decoded address range of the board
* : permanently decoded on logic "0"

Fig. 6-1: DIP switches



IMPORTANT!
You will find the switch s1 on the left of the DIP switches!

6.2 Inserting the board

IMPORTANT!
Please observe the *safety instructions.*

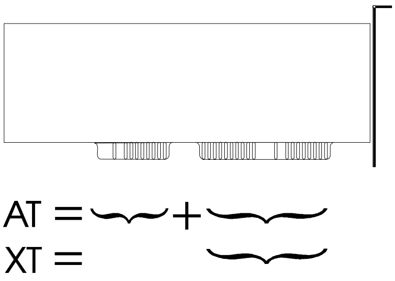
6.2.1 Opening the PC

- Switch off your PC and all the units connected to the PC.
- Pull the PC mains plug from the socket.
- Open your PC as described in the manual of the PC manufacturer.

6.2.2 Selecting a free slot

1. Select a free AT slot

Fig. 6-2: Slot types



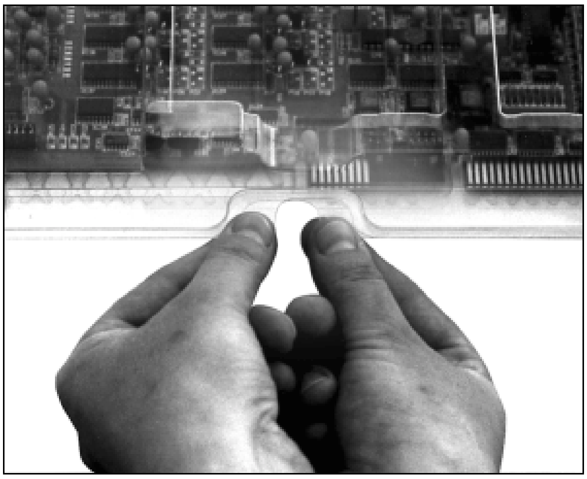
The board can be used in EISA slots under certain conditions.

2. **Remove the back cover of the selected slot**
   according to the instructions of the PC manufacturer.
   Keep the back cover. You will need it if you remove the board.

3. **Discharge yourself from electrostatic charges.**

4. **Take the board from its protective blister pack .**

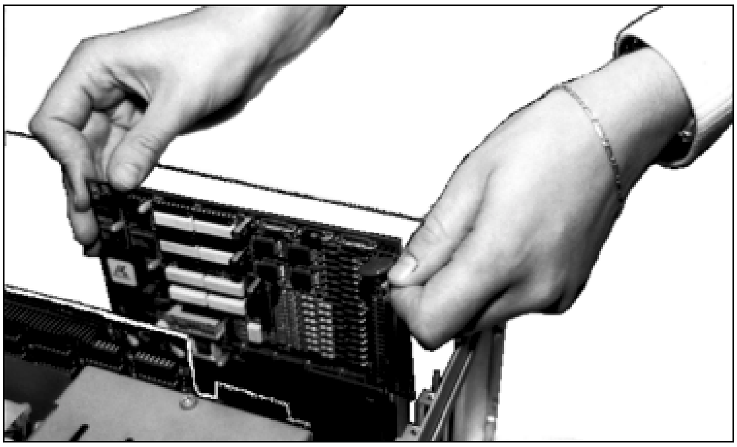
Fig. 6-3: Opening the protective blister pack



6.2.3 Inserting the board

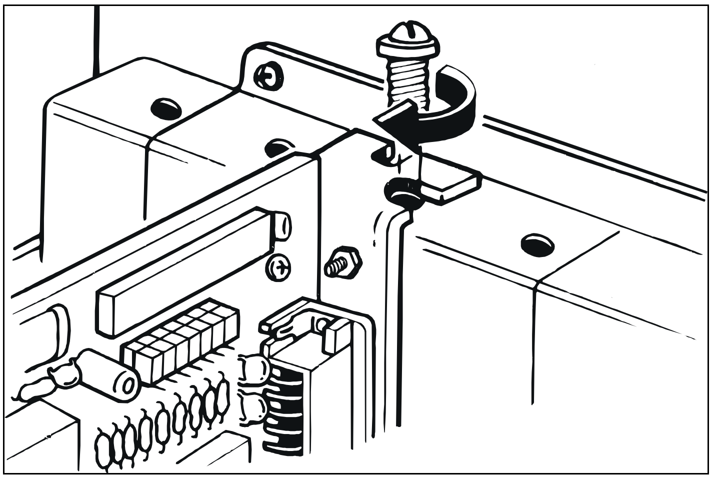
- **Discharge yourself from electrostatic charges**
- Insert the board **vertically into the chosen slot.**

Fig. 6-4: Inserting the board



- **Secure the board** to the rear of the PC housing with the screw which was fixed on the back cover.

Fig. 6-5: Securing the board at the back cover



- **Tighten all loosen screws**

6.2.4 Closing the PC

- Close **your PC as described in the manual of the PC manufacturer.**

6.3 Installing the software

The CD contains:
- ADDIREG for Windows NT 4.0 and Windows 95/98.

You can download the latest version of the ADDIREG program from the Internet.

The CD also contains standard software for the ADDI-DATA boards:
  - 16-bit for MS-DOS and Windows 3.11
  - 32-bit for Windows NT/95/98.

6.3.1 Software installation under MS-DOS and Windows 3.11

- Copy the contents of PA1700\16bit on a diskette.
  If several diskettes are to be used, the directory content is stored in several sub-directories (Disk1, Disk2, Disk3...).
- Insert the (first) diskette into a driver and change to this drive.
- Enter <INSTALL>.

The installation program gives you further instructions.

6.3.2 Software installation under Windows NT/95/98

- Select the directory PA1700\32bit\Disk1.
- Start the set-up program "setup.exe" (double click)
- Select one of the 3 parameters
  1- typical
  2- compact
  3- custom

Proceed as indicated on the screen and read attentively the "Software License" and "Readme".
In "custom", you can select your operating system.

The installation program gives you further instructions.

6.4 Board configuration with ADDIREG

The ADDIREG registration program is a 32-bit program for Windows NT 4.0 and Windows 95/98. The user can register all hardware information necessary to operate the ADDI-DATA PC boards.

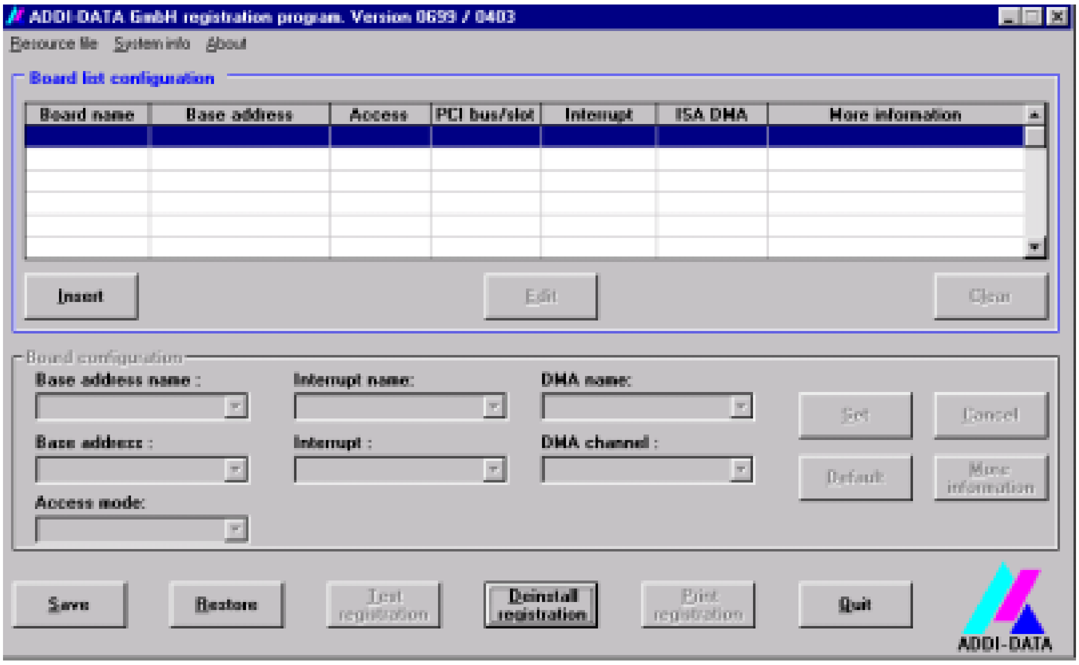
IMPORTANT!
If you use one or several resources of the board, you cannot start the ADDIREG program.

6.4.1 Program description

IMPORTANT!
Insert the ADDI-DATA boards to be registered before starting the ADDIREG program.

If the board is not inserted, the user cannot test the registration.
Once the program is called up, the following box appears.

Fig. 6-6: ADDIREG registration program



The table in the middle lists the registered boards and their respective parameters.

Board name:
Names of the different registered boards (eg.: APCI-7800).
When you start the program for the first time, no board is registered in this table.

Base address:
Selected base address of the board.

IMPORTANT!
The base address selected with the ADDIREG program must correspond to the one set through DIP-switches.

Access:
Selection of the access mode for the ADDI-DATA digital boards. Access in 8-bit or 16-bit.

PCI bus / slot:
Used PCI slot. If the board is no PCI board, the message „NO“ is displayed.

Interrupt:
Used interrupt of the board. If the board uses no interrupt, the message „Not available“ is displayed.

IMPORTANT!
The interrupt selected with the ADDIREG program must correspond to the one set through DIP-switches or jumper.

ISA DMA:
Indicates the selected DMA channel or „Not available“ if the board uses no DMA.

More information:
Additional information like the identifier string (e.g.: PCI1500-50) or the installed COM interfaces.

Text boxes:

Under the table you will find 6 text boxes in which you can change the parameters of the board.

Base address name:
When the board operates with several base addresses (One for port 1, one for port 2, etc.) you can select which base address is to be changed.

Base address:
In this box you can select the base addresses of your PC board. The free base addresses are listed. The used base addresses do not appear in this box.

Interrupt name:
When the board must support different interrupt lines (common or single interrupts), you can select them in this box.

Interrupt:
Selection of the interrupt number which the board has to use.

DMA name:
When the board supports 2 DMA channels, you can select which DMA channel is to be changed.

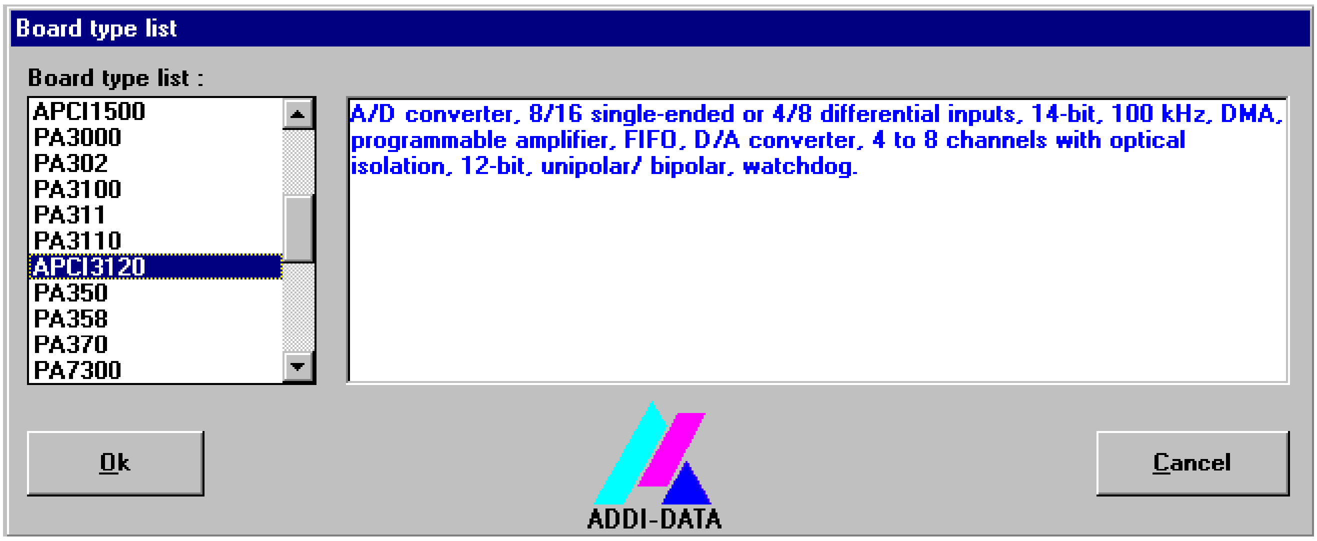
DMA channel:
Selection of the used DMA channel.

Buttons:

Edit[1]:
Selection of the highlighted board with the different parameters set in the text boxes. Click on Edit to activate the data or click twice on the selected board.

Insert:
When you want to insert a new board, click on „Insert". The following dialog window appears:

Fig. 6-7: Configuring a new board



Board type list

Board type list :

APCI1500
PA3000
PA302
PA3100
PA311
PA3110
APCI3120
PA350
PA358
PA370
PA7300

A/D converter, 8/16 single-ended or 4/8 differential inputs, 14-bit, 100 kHz, DMA, programmable amplifier, FIFO, D/A converter, 4 to 8 channels with optical isolation, 12-bit, unipolar/ bipolar, watchdog.

All boards you can register are listed on the left. Select the board. (The corresponding line is highlighted). You can read technical information about the board(s) on the right. Activate with „OK"; You come back to the former screen.

Clear:
You can delete the registration of a board. Select the board to be deleted and click on „Clear".

Set:
Sets the parametered board configuration. The configuration should be set before you save it.

Cancel:
Reactivates the former parameters of the saved configuration.

Default:
Sets the standard parameters of the board.

More information:
You can change the board specific parameters like the Identifier string, the COM number, the operating mode of a communication board, etc...
If your board does not support this information, you cannot activate this button.

---

[1] "x": keyboard shortcuts; e.g.: "Alt + e" for Edit

Save:
Saves the parameters and registers the board.

Restore:
Reactivates the last saved parameters and registration.

Test registration:
Controls if there is a conflict between the board and other devices.

A message indicates the parameter which has generated the conflict. If there is no conflict, „OK“ is displayed.

Deinstall registration:
Deinstalls the registrations of all board listed in the table.

Print registration:
Prints the registration parameter on your standard printer.

Quit:
Quits the ADDIREG program.

6.4.2 Registering a new board

IMPORTANT!
To register a new board, you must have administrator rights.
Only an administrator is allowed to register a new board or change a registration.

- Call up the ADDIREG program. The figure 6-6 is displayed on the screen. Click on „Insert“. Select the wished board.

- Click on „OK“. The default address, interrupt, and the other parameters are automatically set and listed in the lower fields.
  If the parameters are not automatically set by the BIOS, you can change them. Click on the wished scroll function(s) and select a new value.
  Activate your selection with a click.

- Once the wished configuration is set, click on „Set“.

- Save the configuration with „Save“.

- You can test if the registration is „OK“.
  This test controls if the registration is right and if the board is present.
  If the test has been successfully completed you can quit the ADDIREG program. The board is initialised with the set parameters and can now be operated.

  In case the registration data is to be modified, it is necessary to boot your PC again. A message asks you to do so. When it is not necessary you can quit the ADDIREG program and directly begin with your application.

6.4.3 Changing the registration of a board

IMPORTANT!
To register a new board, you must have administrator rights.
Only an administrator is allowed to register a new board or change a registration.

- Call up the ADDIREG program. Select the board to be changed.
  The board parameters (Base address, DMA channel,..) are listed in the lower fields.

- Click on the parameter(s) you want to set and open the scroll function(s).

- Select a new value. Activate it with a click.
  Repeat the operation for each parameter to be modified.

- Once the wished configuration is set, click on „Set“.

- Save the configuration with „Save“.

- You can test if the registration is „OK“.
  This test controls if the registration is right and if the board is present.
  If the test has been successfully completed you can quit the ADDIREG program.
  The board is initialised with the set parameters and can now be operated.

  In case the registration data is to be modified, it is necessary to boot your PC again. A message asks you to do so. When it is not necessary you can quit the ADDIREG program and directly begin with your application.

6.4.4 Removing the ADDIREG program

The ADDI_UNINSTALL program is delivered on the CD-ROM.

- Install the ADDI_UNINSTALL program on your computer.
- Start the ADDIREG program and click on "Deinstall registration"
- Quit ADDIREG
- Start the ADDI_UNINSTAlL program
- Proceed as indicated until the complete removing of ADDIREG.

You can also download the programm from Internet.

6.5 Board configuration with ADDIMON for DOS

- In the directory MONITOR enter <MON1700>.
-The monitoring program of the board **PA 1700-2** is loaded.

6.5.1 Help for configuration

Set the required configuration:
- Base address settings with the block of DIP switches
The ADDIMON program tests the basic hardware functions of the **PA 1700-2.**

6.5.2 Hotline protocol, quick technical support

Should a problem arise:
- fill in the hotline protocol.
- Print it and send it to our technical support

You will then receive a quick solution to your problem.

6.6 Software downloads from the Internet

You can download the latest version of the device driver for the PA 1700-2 board from

http://www.addi-data.de. or
http://www.addi-data.com

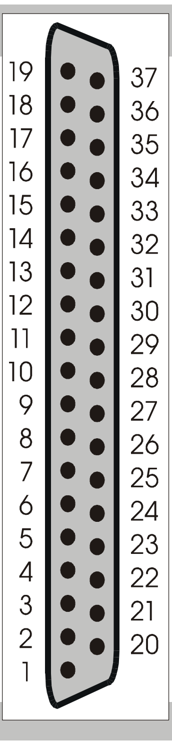
If you have any questions, do not hesitate to send us an e-mail to

info@addi-data.de or
hotline@addi-data.com

7 CONNECTION TO THE PERIPHERAL

7.1 Connector pin assignment

Fig. 7-1: 37-pin SUB-D male connector

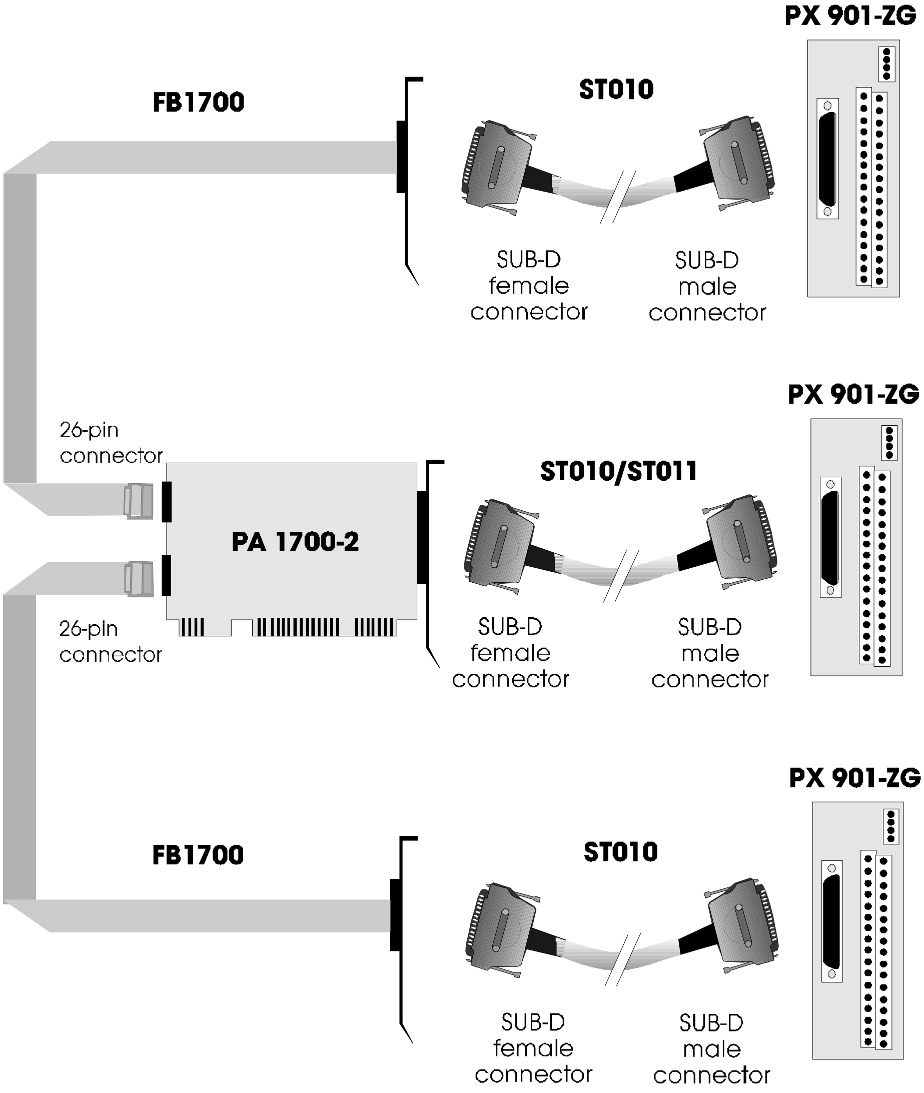


Level	Signal			Signal	Level
	GND	19	37	Timer2 out	TTL
Diff.	+B1	18	36	B1	Diff./TTL
Diff.	+A1	17	35	A1	Diff./TTL
Diff.	+B2	16	34	B2	Diff./TTL
Diff.	+A2	15	33	A2	Diff./TTL
TTL	Ext. strobe 1/1	14	32	Ext. strobe 1/2	TTL
TTL	Ext. strobe 2/1	13	31	Ext. strobe 2/2	TTL
Diff.	+B3	12	30	-B3	Diff./TTL
Diff.	+A3	11	29	-A3	Diff./TTL
Diff./TTL	+Index2	10	28	-Index2	Diff./TTL
Diff./TTL	+Index1	9	27	-Index1	Diff./TTL
TTL	Ext. Strobe 3/1	8	26	Ext. strobe 3/2	TTL
TTL	Ref1	7	25	Ref2	TTL
Diff./TTL	+Index3	6	24	-Index3	Diff./TTL
Diff./TTL	-AS2	5	23	Ref3	TTL
Diff./TTL	+AS2	4	22	+AS3	Diff./TTL
Diff./TTL	+AS1	3	21	-AS3	Diff./TTL
Diff./TTL	-AS1	2	20	+ 5 V from PC *	
	GND	1			

Fuse
Max. reliable current : 500 mA

7.2 Connection to the screw terminal board PX 901-ZG

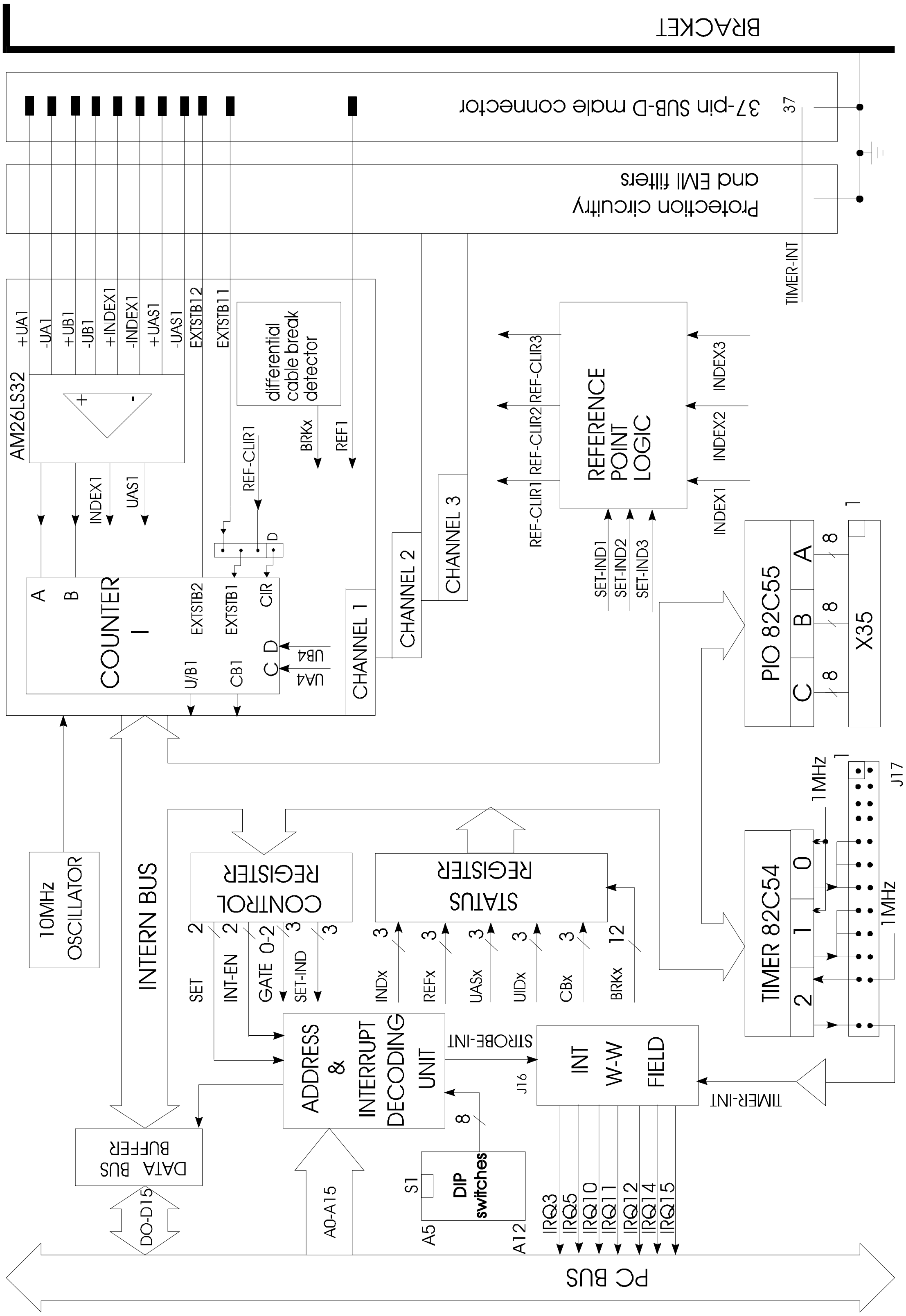
Fig. 7-2: connection to the screw terminal board PX 901-ZG



8 FUNCTIONS OF THE BOARD

8.1 Block diagram

Fig. 8-1: Block diagram



8.2 Description of the board

- Typical applications:

- Acquisition of incremental displacement measurement systems
- X,Y,Z controls
- Pulse width and frequency measurement
- Analysis of incremental encoders
- Tolerance measurements
- Electronic "mouse"

- Connection of 3 up to 6 incremental encoders

- 3 "INDEX" inputs for reference point logic (TTL / diff.)
- 3 "UAS" inputs: can be used as error inputs (TTL / diff.)
- 3 "REF" inputs (TTL)
- 6 "EXTSTB" inputs (TTL, low active). 2 inputs for each counter channel
- High counting speed

- Functions of the counter modules:

- quadruple / double / single analysis of 2 phase-shifted clock pulses. Further processing in a 32-bit loadable upward and downward counter (or a 16-bit counter)
- Direction recognition for upwards and downwards counting
- Hysteresis circuit for the suppression of the first pulse after change of rotation, switchable
- Two 32-bit data latches, can be programmed independently for internal/external strobe, latch strobe synchronised with the internal clock pulse
- Operating mode is defined by the internal mode register, loadable/ readable through the data bus
- Clear and strobe inputs, can be triggered either through 2 external pins or by writing in a register

- Interrupt indication, triggered by the external strobe inputs
- Integrated test mode
- Pulse width measurement through direct inputs, programmable counter direction
- +5V on the front connector through microfuse for supplying the incremental encoders
- Timer component 82C54 can be used internally or externally
- PIO component 82C55, 24 TTL I/O lines through the pin header
- AT board with 16-bit access
- Base address set with DIP switches
- Recognition of line interruption in the differential mode
- All front connector inputs are protected
- Interrupt possibility through timer component or external strobe inputs

The board ADDICOUNT **PA 1700-2** is a fast counter board for 90° phase-shifted signals (displacement measuring systems) for AT or compatible computers. It is particularly suited for tough industrial applications.

The board is provided with numerous acquisition channels (up to 6) and with functions (timer, TTL I/O) requiring minimum space. Furthermore the board has been developed very carefully with circuit design and layout in accordance with electromagnetic compatibility.

All inputs of the incremental encoders are protected with Transil diodes (11V, 400W, 1ms). Additional TTL input are protected through a series resistor and double Schottky diodes against voltage reversal and overvoltage.

All inputs of the front connector are filtered with EMI filters or condensators.

The board allows to connect directly the signals of 3 incremental encoders (A, B signals, index signal, error signal, reference signal). 3 others incremental encoders can be connected through a pin header (A, B signals) (TTL level).

The board is in quadruple, double, single analysis mode or can be programmed in the direct mode for measuring frequencies or pulse widths.

The modes are programmed through the registers.

After PC reset the board is ready to be operated. The counters can be read at any time or loaded with an initial value. The integrated direction recognition logic increments or decrements the count according to the edge change of the incremental encoder 90° phase-shifted A & B signals.

Each counter module is provided with a double set of data latches which allows to store two different counting values whereby the counting operation continues running in the background.

This storage occurs by software with a writing of a bit (according to the data latch set) or with two external signals (EXTSTB).

Thereby one of these signals is responsible for a data latch set. If a count is latched through an external strobe line, this can generate an interrupt to the PC bus.

The **PA 1700-2** board can be configured in a system of three counter channels with a counting depth of 16 or 32 bits. All necessary input signals are then situated on the 37-pin SUB-D male connector. Incremental encoders can be connected with TTL or differential outputs (selectable through jumper).

If a 16-bit counting depth is needed, the user has up to 3 counter channels for incremental encoders with TTL outputs (A & B signals). These 3 other counter channels are available on the pin header.

A reference point logic has been integrated. It allows to clear or to store (via jumper) the count, as soon as an index pulse appears and is controlled by software.

When the counter content was stored, an interrupt request can occur.

On the board there is a timer component of the type **82C54** which can be used for internal or external purposes:

- internally as a time interrupt generator: it can act as a synchronous signal for storing one or several counter modules.
- externally as a time reference signal (on the SUB-D front connector): it can be used for frequency or pulse width measurement.

On the board there is a PIO component of type **82C55**. The 24 TTL inputs and outputs are available for the user through the pin header.

8.3 I/O map

The board **PA 1700-2** requires an address range of 28 I/O addresses within the I/O address space of the PC.

The functionality of the board is responded with a write or read operation on this address range.

- **counter channels 1 to 6:** the word addresses 0 to 14, as well as 24 and 26 are used. You determine which **counter module** is selected through the word addresses with the bits **SEL0** and **SEL1** of **CONTROL-REG on Base + 24**.

- The Timer (**82C54**) is programmed with the byte addresses 16 to 19.

- The PIO (**82C55**) is programmed with the byte addresses 20 to 23.

8.3.1 16-bit I/O map

Table 8-1: 16-bit I/O map

	IORD						**IOWR**					
	D15	**----------**	**D8**	**D7**	**----------**	**D0**	**D15**	**----------**	**D8**	**D7**	**----------**	**D0**
Base+0	X			INT-REG			-			STB-REG		
Base+2	LATCH 1.1			LATCH 1.0			COUNT.1			COUNT.0		
Base+4	LATCH 1.3			LATCH 1.2			COUNT.3			COUNT.2		
Base+6	LATCH 2.1			LATCH 2.0			-			-		
Base+8	LATCH 2.3			LATCH 2.2			-			-		
Base+10	MOD-REG2			MOD-REG1			MOD-REG2			MOD-REG1		
Base+12	X			X			-			TEST-REG		
Base+14	X			X			-			-		
Base+16	**8-bit**											
Base+20	**I/O map**											
Base+24	STATUS-REG1						CONTROL-REG1					
Base+26	STATUS-REG2						CONTROL-REG2					

X: any data -: no function

10 word addresses are reserved in the I/O map for the counter channels. Write and read operations can access this I/O map part only in the 16-bit mode (word access).

The word addresses 0 to 14 are the internal registers of the respective **counter module.** Since each **counter module** would occupy 16 addresses of the I/O map, the addresses of the three counter modules have been placed in parallel. The **counter module** is selected through the bits **SEL0** and **SEL1** of **CONTROL-REG.**

The word addresses 24 and 26 are external registers and are used for example to get digital information through the input states or to select the counter module.

CONTROL-REG1

The register **CONTROL-REG1** is located on **Base + 24**. You define the following parameters through a write on this register:

- **INT-EN1, INT-EN2**: Enable the strobe interrupt.
- **GATE0, GATE1, GATE2**: Enable the gate inputs of the timers.
- **SEL0, SEL1**: select the counter module.

This register can not be read. We advise to store the last written value in a shadow register. After the PC reset this register is set to "0".

Table 8-2: CONTROL-REG1 (Base+24) (Write)

D15		D6						D0
-	...	INT-EN 2	INT-EN 1	GATE 2	GATE 1	GATE 0	SEL 1	SEL 0

1) INT-EN1-2

Each **counter module** can generate two interrupts. These interrupts are generated as soon as a negative pulse is produced on the **EXTSTBxy** inputs.

Each **counter module** has 2 strobe inputs which allow to store the respective count in the appropriate latch.

An interrupt is generated when the storage operation is ended.

These two bits are used for dividing the **Strobe-Int** (common interrupt of all counter modules) into two groups.

After a PC reset these two bits are set to logic "0".

- **INT-EN1** = "0"
  The interrupt requests of the three **counter modules** are suppressed through a negative pulse at the **EXTSTBx1** inputs.
- **INT-EN1** = "1"
  The interrupt requests of the three **counter modules** are enabled through a negative pulse at the **EXTSTBx1** inputs.
- **INT-EN2** = "0"
  The interrupt requests of the three **counter modules** are suppressed through a negative pulse at the **EXTSTBx2** inputs.
- **INT-EN2** = "1"
  The interrupt requests of the three **counter modules** are enabled through a negative pulse at the **EXTSTBx2** inputs.

2) GATE0-2

These 3 bits are used for controlling the three GATE inputs of timer 82C54. These three digital signals are directly led to the timer. After a PC reset these three bits are set to logic "0".

- **GATEx** = "0" TIMERx = disabled
- **GATEx** = "1" TIMERx = enabled

3) SEL0-1

These three bits are used for selecting the **counter module**. After a PC reset these two bits are set to logic "0".

SEL1	SEL0	Selected counter module
0	0	1
0	1	2
1	0	3
1	1	-

CONTROL-REG2

The register **CONTROL-REG2** is located on **Base + 26**. You define the following parameters through a write on this register:

- **SET-IND1, SET-IND2, SET-IND3**: Enables clearing or storing the counter content when an **INDEX** input indicates that a reference point has been exceeded.

This register can not be read. We advise to store the last written value in a shadow register. After a PC reset this register is set to "0".

Table 8-3: CONTROL-REG2 (Base+26) (Write)

D15		D2	D1	D0
-	...	SET-IND3	SET-IND2	SET-IND1

1) SET-IND1-3

These three bits allow to clear or to store the counter content when an **INDEX** input indicates that the reference point has been exceeded (reference point logic). After a PC reset these three bits are set to logic "0".

- **SET-INDx** = "0"
  The counter content is not cleared nor stored when **INDEXx** indicates that the reference point has been exceeded.
- **SET-INDx** = "1"
  The counter content is cleared or stored when **INDEXx** indicates that the reference point has been exceeded. The bit **SET-INDx** is reset automatically if **INDEXx** indicates that the reference point has been reached.

STATUS-REG1, STATUS-REG2

STATUS-REG1, 2 are located on **Base + 24** and **Base + 26**. With one read on these registers you can receive the following digital information:

- **U/D1, U/D2, U/D3**: Counting direction of the respective counter module
- **UAS1, UAS2, UAS3**: if error has occurred
- **REF1, REF2, REF3**: if a reference is present
- **IND1, IND2, IND3**: if an index pulse has been stored
- **BRK0, ..., BRK11**: line interruption recognition in the differential mode

Table 8-4: STATUS-REG1 (Base+24) (Read)

D15							D8	D7							D0
-	-	-	-	U/D 3	U/D 2	U/D 1	UAS 3	UAS 2	UAS 1	REF 3	REF 2	REF 1	IND 3	IND 2	IND 1

1) U/D1-3

These three bits inform about the counting direction of **counter module x**, when used as a 32-bit counter.

- **U/Dx** = "0"
  Counter module x is counting upwards
- **U/Dx** = "1"
  Counter module x is counting downwards

2) UAS1-3

These three bits inform about the state of the **diff.-/ TTL** inputs **UAS1-3**.

- **UASx** = "0"
  Input **UASx** on logic "0"
- **UASx** = "1"
  Input **UASx** on logic "1"

3) REF1-3

These three bits inform about the state of the **TTL** inputs **REF1-3**.

- **REFx** = "0"
  Input **REFx** on logic "0"
- **REFx** = "1"
  Input **REFx** on logic "1" (state when input open)

4) IND1-3

These three bits indicate if an index pulse has been generated on **INDEX (**input) when the reference point was exceeded. These three bits are cleared when **STATUS-REG1** is read.

- **INDx** = "0"
  No index pulse was stored on **counter channel x**
- **INDx** = "1"
  An index pulse was detected on **counter channel x**

Table 8-5: STATUS-REG2 (Base+26) (Read)

D15							D8	D7							D0
-	-	BRK 11	BRK 10	BRK 9	BRK 8	BRK 7	BRK 6	BRK 5	BRK 4	BRK 3	BRK 2	BRK 1	BRK 0	SEL 1	SEL 0

5) BRK0-11

These bits inform about the state of the lines when used in the differential mode. You will find in the next table the lines and their corresponding bit:

- **BRKx** = "0" No line interruption has been detected
- **BRKx** = "1" Line interruption detected on **line x**

	BRK bit	**Differential line monitoring**
Counter channel 1	BRK0	UA1
	BRK1	UB1
	BRK2	INDEX1
	BRK3	UAS1
Counter channel 2	BRK4	UA2
	BRK5	UB2
	BRK6	INDEX2
	BRK7	UAS2
Counter channel 3	BRK8	UA3
	BRK9	UB3
	BRK10	INDEX3
	BRK11	UAS3

6) SEL0-1

These two bits are from register **CONTROL-REG1**.

8.3.2 8-bit I/O map

Table 8-6: 8-bit I/O map

	IORD								**IOWR**							
	D7	D6	D5	D4	D3	D2	D1	D0	D7	D6	D5	D4	D3	D2	D1	D0
Base +16	TIMER 0															
Base +17	TIMER 1															
Base +18	TIMER 2															
Base +19	STATUS TIMER								CONTROL TIMER							
Base +20	PORT A															
Base +21	PORT B															
Base +22	PORT C															
Base +23	STATUS - PIO								CONTROL - PIO							

8.4 Functions

WARNING!
Do not operate the board simultaneously in several modes.
Otherwise you may destroy the board, PC and/or the peripheral.

- **Make sure** to set only the jumpers required for the respective functions.
  See Fig. 8-1: Block diagram

8.4.1 Counter module

The board main functions and programming are characterised by the features of the counter module.

The counter module contains:
- two independent 16-bit counters, internally cascadable
- quadruple/double/single edge analysis circuits for two phase-shifted clock pulses,
- Direction discriminators,
- Hysteresis circuits,
- two 32-bit latches and a function and control logic.

The operating mode of the counter module is written through a 16-bit data bus in register **MOD-REG1** and **MOD-REG2.**

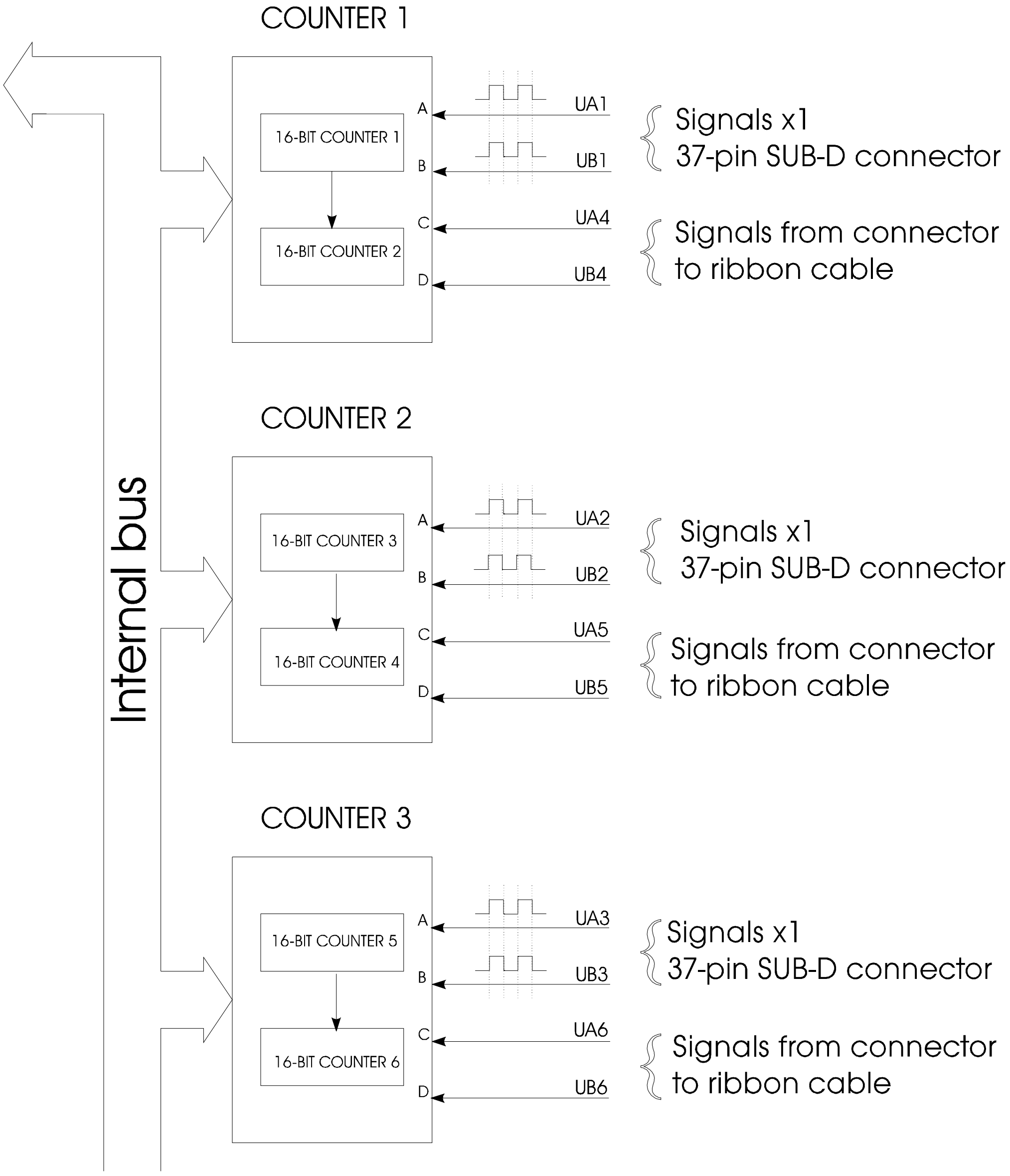
A 10 MHz quartz oscillator produces the clock pulses for the counter modules.

Each of the two 16-bit counters can be connected to two-phase shifted clock pulses through two input pins (A, B for the first, C, D for the second counter).

If both counters are cascaded internally by software (32-bit counters), only a phase-shifted signal of an incremental encoder can be connected.

From 3 up to 6 incremental encoders can thus be connected in the following configurations:
- 3 counter channels with a counting depth of 16 or 32 bits, for TTL or differential incremental encoders through the SUB-D front connector, in each case the A, B inputs of the three counter modules.
- 3 other counter channels with a 16-bit counting depth, for TTL incremental encoders through the pin header (only if the 3 first counter channels are operated in the 16-bit mode), in each case the C, D inputs of the three counter modules.
- Other possibilities are available for the user:
  - two 32-bit counters (for each counter module) are connected to the incremental encoder through the front connector
  - two other 16-bit counters (third counter module) are connected to the first incremental encoder through the front connector. The second incremental encoder is connected through the pin header.

Fig. 8-2: Assignment of counter channels and counter modules



The counter module is provided with:
- a "Clear" input, which clears the content of the 32-bit counter, without affecting the programmed mode
- and two external strobe inputs.

To each of these strobe signals is assigned a set of data latch registers:

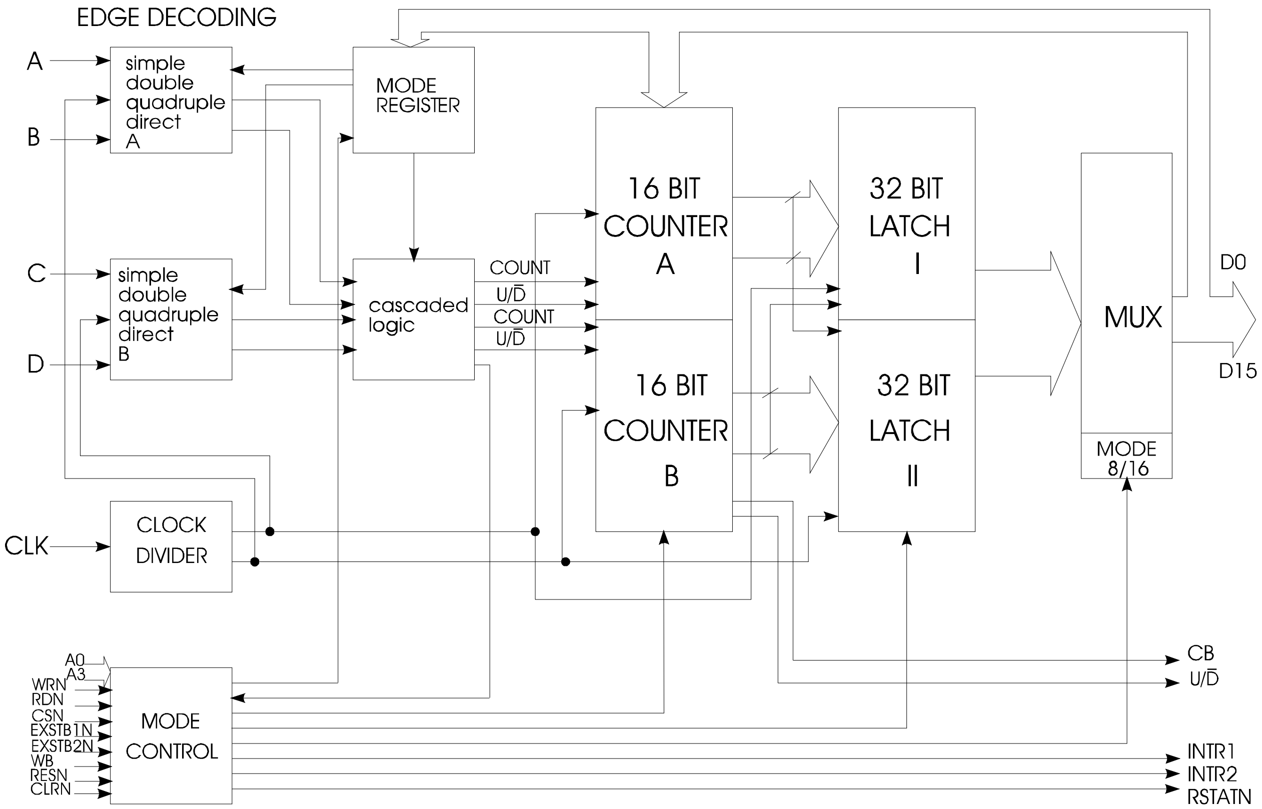
EXTSTBx1 is assigned to LATCH1.0 → LATCH1.3,
EXTSTBx2 is assigned to LATCH2.0 → LATCH2.3

If an external strobe signal becomes active (low), the content of the 32-bit counter is written in the corresponding data latch register.

Afterwards an interrupt request is given to the interrupt logic.

Block diagram of the counter module

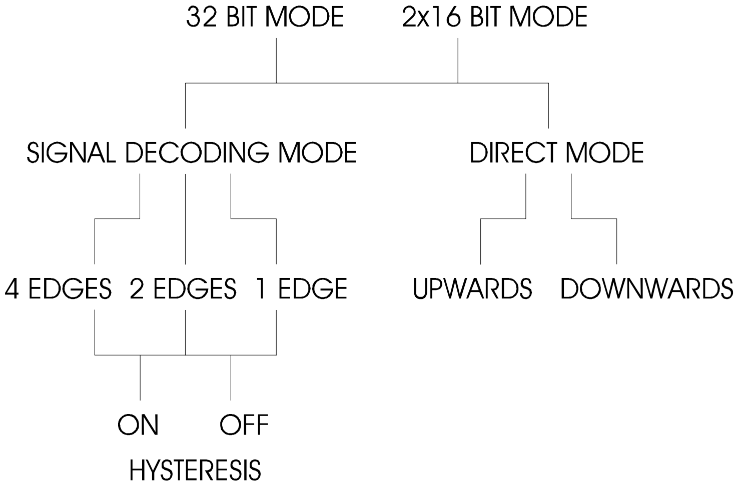
Fig. 8-3: Block diagram of the counter module



Function description

The function of the counter module is determined by **MOD-REG1** (**base + 10**) that can be written and read.

The following operating modes are possible:

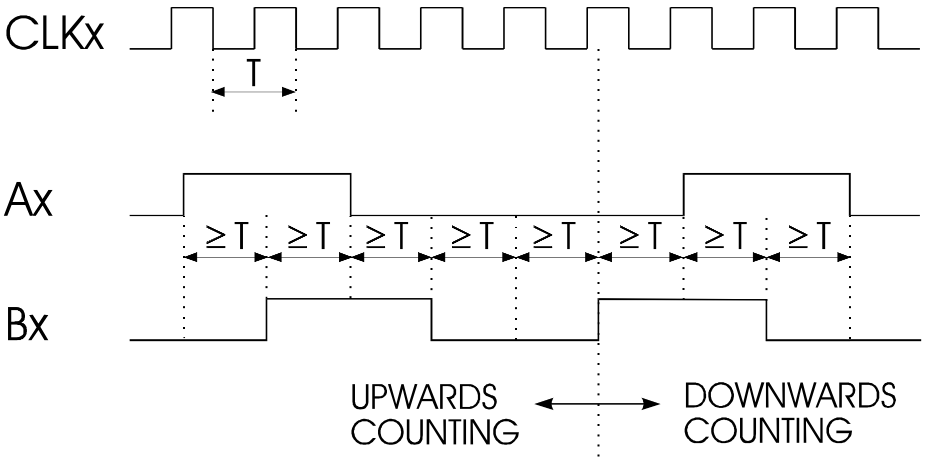


Inputs for Edge analysis circuit

- A, B
  Inputs for phase shifted signals with a quadruple/double/single MODE 32-bit counter or a 16-bit counter A.
  A change of signal A before a change of signal B means counting upwards.
  A change of signal B before a change of signal A means counting downwards.
- C, D
  Like A, B; however inputs for phase shifted signal with the 16-bit counter B.

The edge analysis circuit requires 3 clock pulse periods for analysing the edges or direction recognition. Counting occurs with the 4th clock pulse (concerns all operating modes related to the external negative clock pulse edge).

Fig. 8-4: Edge analysis circuit

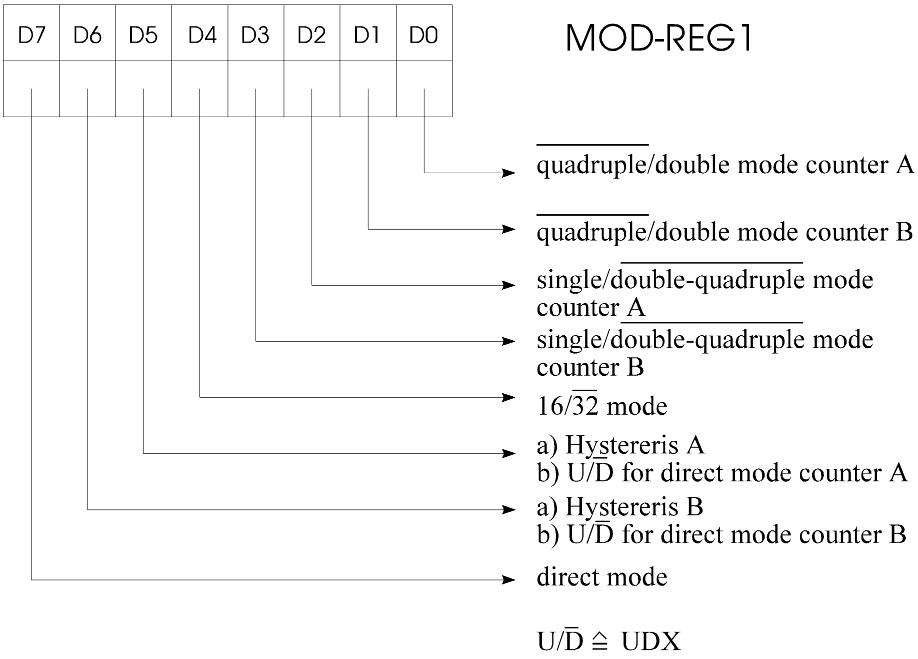


This figure shows that no definite phase shifts between the signals A, B and C, D are necessary. But minimum values which are temporarily related to the system clock pulse have to be respected (minimal edge distance).

A ≅ AX
B ≅ BX
C ≅ CX
D ≅ DX

MODE-Register 1 (Base + 10)

Fig. 8-5: MODE-Register 1


D7
D6
D5
D4
D3
D2
D1
D0
MOD-REG1
$\overline{\text{quadruple}}$/double mode counter A
$\overline{\text{quadruple}}$/double mode counter B
single/$\overline{\text{double-quadruple}}$ mode counter A
single/$\overline{\text{double-quadruple}}$ mode counter B
16/$\overline{32}$ mode
a) Hystereris A
b) U/$\overline{D}$ for direct mode counter A
a) Hystereris B
b) U/$\overline{D}$ for direct mode counter B
direct mode
U/$\overline{D}$ ≙ UDX

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 deactivated internally.
Independently from the analysis inputs no counting is processing any longer, and the last count is maintained.

1) 16/32-bit MODE is determined with 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 is either used the quadruple/double/single edge analysis circuit A or the DIRECT MODE.

On U/$\overline{D}$ is available the upwards/downwards signal recognised in the edge analysis circuit A.

In the 16-bit MODE is either used:
- 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).

On U/$\overline{D}$ is available the upwards/downwards signal recognised in the edge analysis circuit.

2) quadruple/double/single MODE is determined with data bits D0-D3 of MODE-Register 1

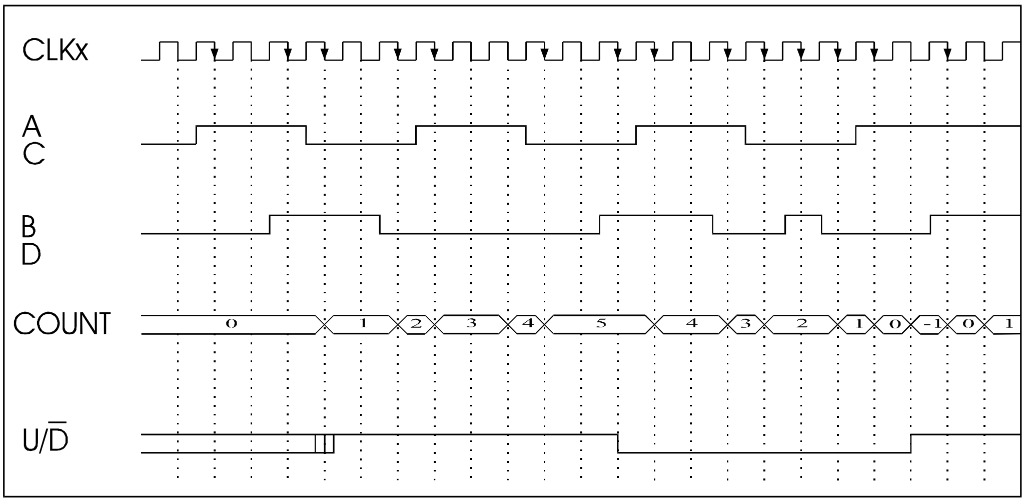
L at D0	quadruple MODE for edge analysis circuit A
L at D1	quadruple MODE for edge analysis circuit B
H at D0	double MODE for edge analysis circuit A
H at D1	double MODE for edge analysis circuit B
L at D2, D3	quadruple or double MODE for edge analysis circuit A, B, independent from D0, D1
H at D0 and D2, D1 and D3	single MODE for edge analysis circuit A,B
L at D0 and D1, H at D2 and D3	edge analysis circuit A, B inactive

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 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. 8-6: Edge analysis circuit (quadruple MODE)



Sampling (A, B, C, D) with CLKX↓
Counting (COUNT) with CLKX↓

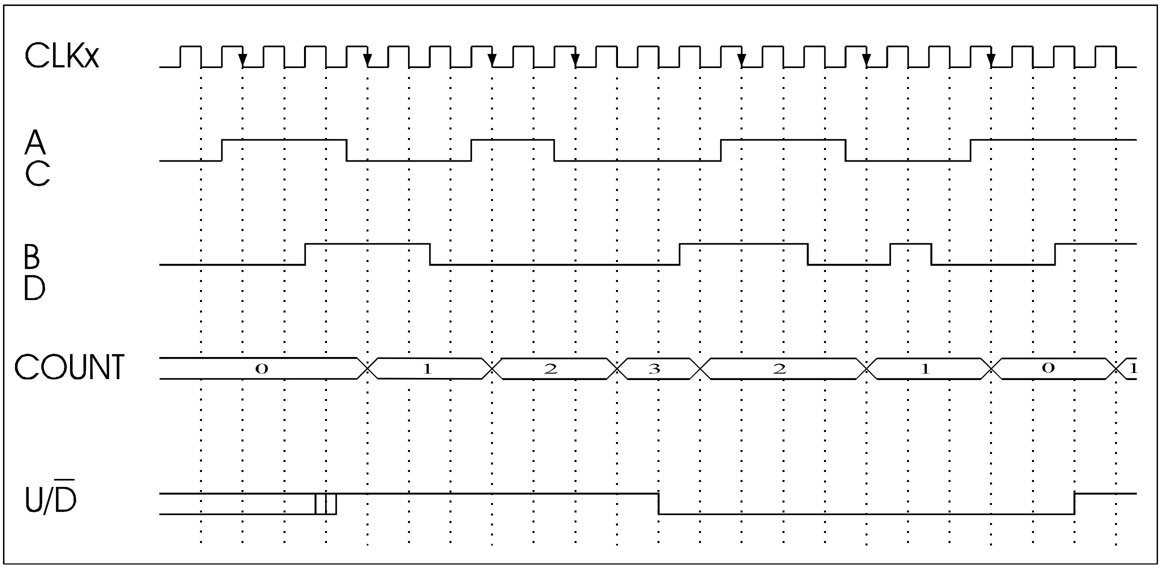
MODE Register 1	Bit	7	6	5	4	3	2	1	0
32-bit MODE		0	X	0	0	X	0	X	0
16-bit MODE		0	0	0	1	0	0	0	0

Hysteresis is switched off in this case

double MODE

Functions in the same way as the quadruple MODE, except that only two of the four edges are analysed per period.

Fig. 8-7: Edge analysis circuit (double MODE)



Sampling (A, B, C, D) with CLKX↓
Counting (COUNT) with CLKX↓

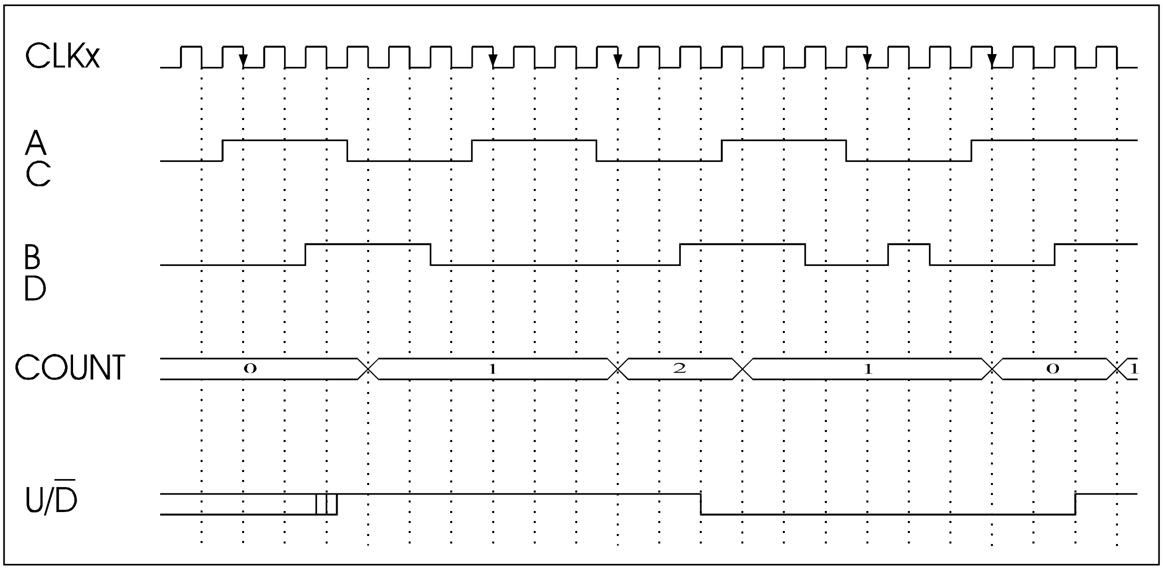
MODE Register 1	**Bit**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
32-bit MODE		0	X	0	0	X	0	X	1
16-bit MODE		0	0	0	1	0	0	1	1

Hysteresis is switched off in this case

single MODE

Functions in the same way as the quadruple MODE, except that only one of the four edges is analysed per period.

Fig. 8-8: Edge analysis circuit (single MODE)



Sampling (A, B, C, D) with CLKX↓
Counting (COUNT) with CLKX↓

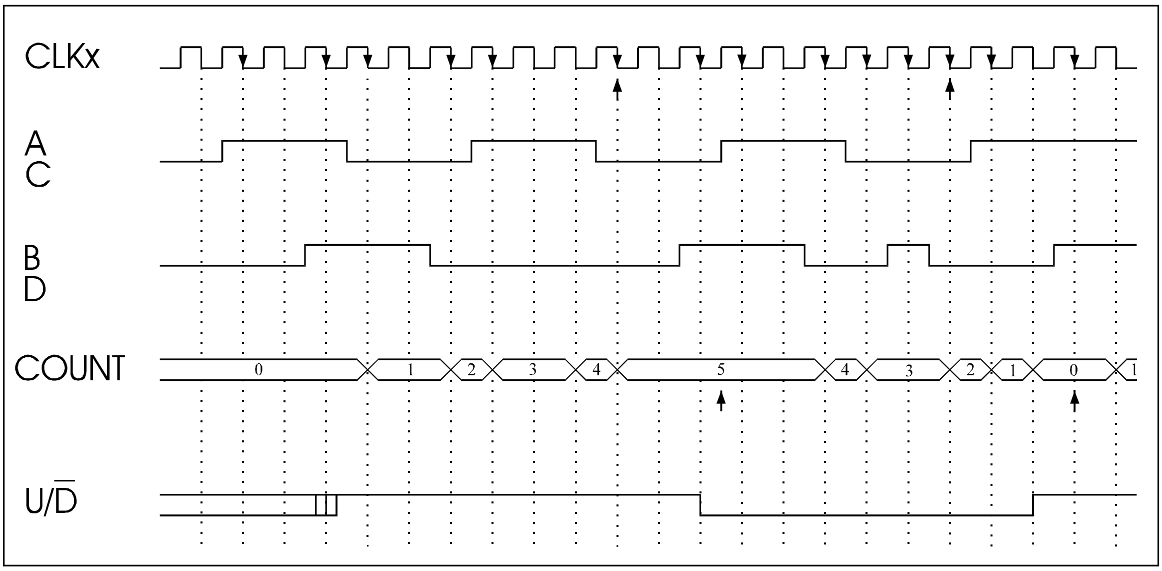
MODE Register 1	Bit	7	6	5	4	3	2	1	0
32-bit MODE		0	X	0	0	X	1	X	1
16-bit MODE		0	0	0	1	1	1	1	1

Hysteresis is switched off in this case

Hysteresis

In both edge analysis circuits is available one hysteresis circuit. It suppresses each time the first counting pulse after a change of rotation.

Fig. 8-9: Hysteresis
Edge analysis circuit in the quadruple MODE



Sampling (A, B, C, D) with CLKX↓
Counting (COUNT) with CLKX↓

Change of rotation upwards/downwards/upwards
Hysteresis switched on

quadruple MODE with hysteresis:

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

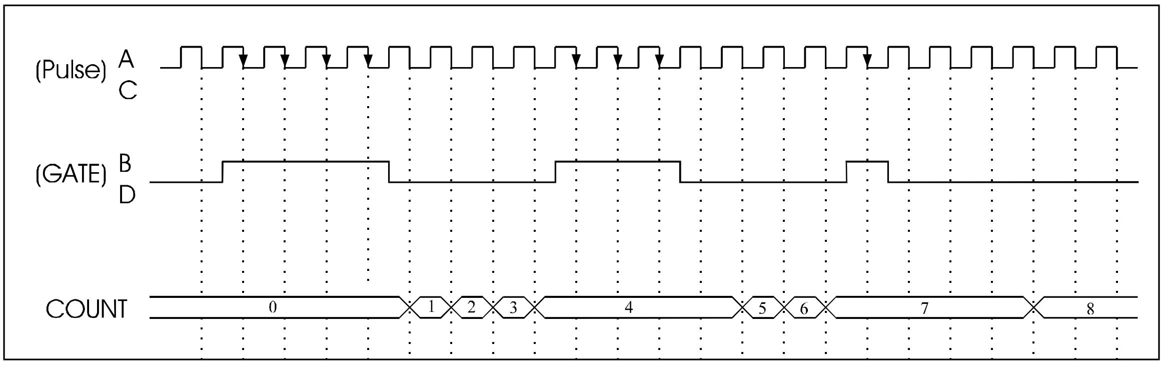
Direct MODE

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. Set the CLK input on low. The 32-bit counter or both 16-bit counters can be independently set to upward or downward counting through bit D5 or D6 of the MODE register.
The gate input is synchronised with the falling edge of A or C.

In the 16-bit MODE, the CLK input can be used as a clock pulse input. Set therefore inputs A and C to low.

Fig. 8-10: Pulse diagram of the time gate measurement



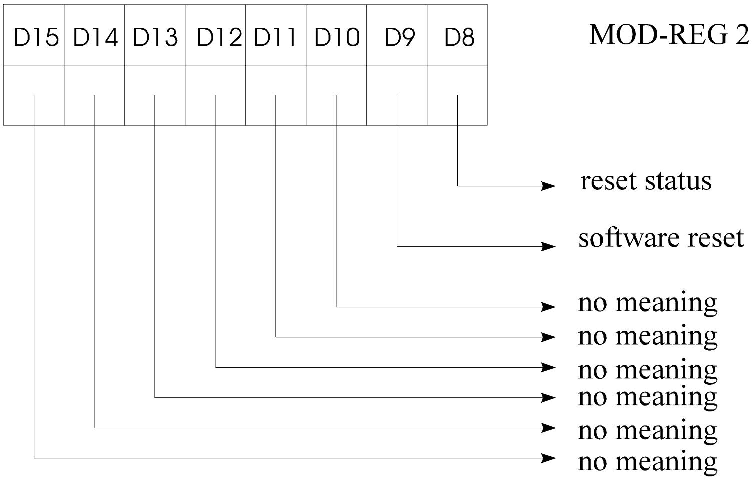
Sampling (B, D) with A↓, B↓ (CLK = 0)
CLK ≅ CLKX

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).

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

MODE Register 2 (Base + 10)



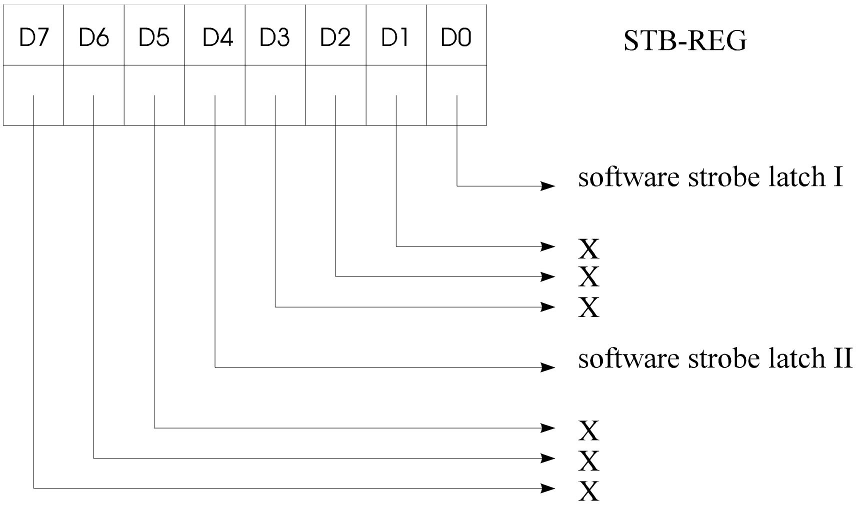
The MODE register 2 is intended for generating and displaying the RESET state of the component (see also reset logic).

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 the reset has run)

The bit positions D10-D15 have internally no meaning.
They can be written in and read out. They are neither influenced by an internal nor by an external reset operation. Your values are random until a write access occurs for the first time on these bit positions.

Strobe register (Base + 0)



A software strobe is produced by writing on the corresponding data bits of the strobe register. The strobe cycle is released with value 1 of the corresponding data bit. Value 0 has no meaning.

A hardware strobe is released through the both pins EXTSTBx1 (low active for Latch I) and EXTSTBx2 (low active for Latch II).

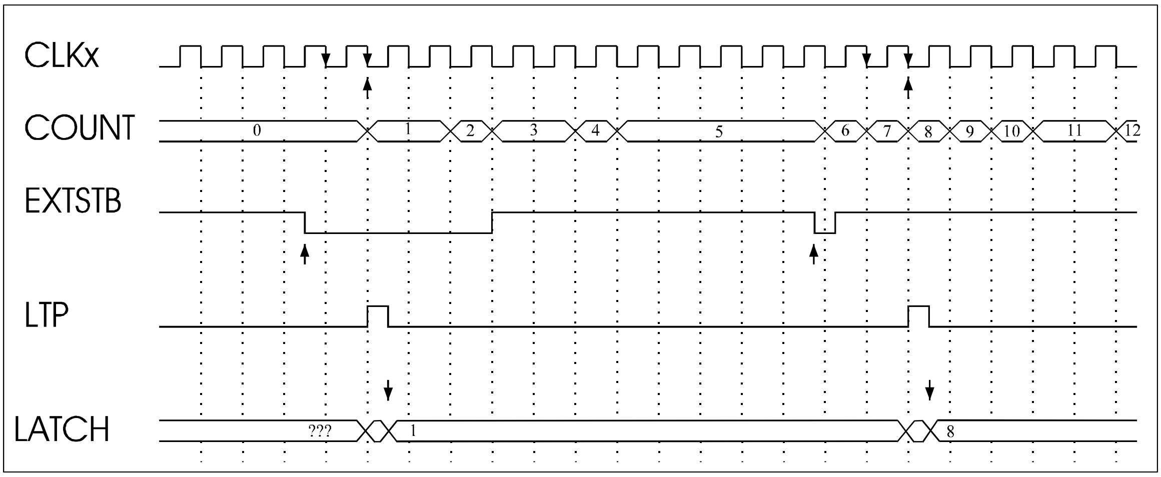
The strobe register can not be written in.

Latch logic

It consists of 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 (Pins EXSTBx1, EXSTBx2).

All strobe signals are synchronised with eventually simultaneous count pulses occurring internally. Intermediate states are thus not stored. Through synchronising the count is stored after a shift of two clock pulses.

Fig. 8-11: Pulse diagram of the latch logic



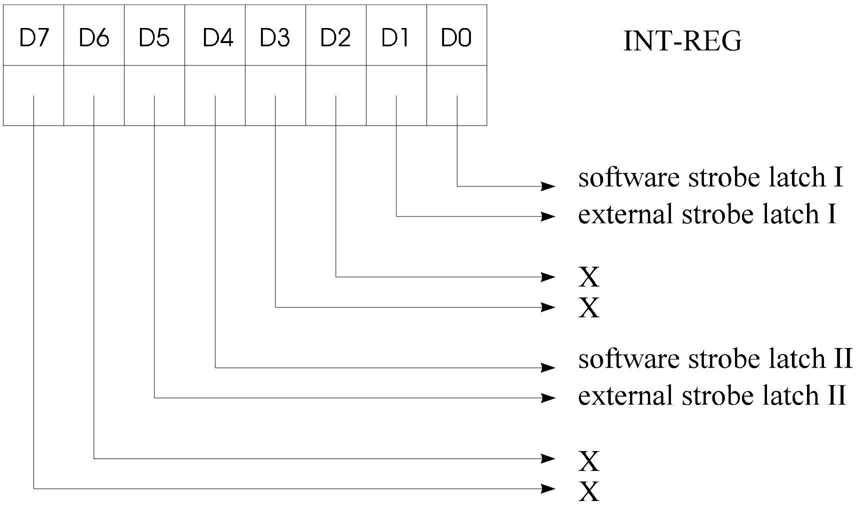
LTP: internal latch pulse
Counting (COUNT) with CLKX↓

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

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

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

Interrupt register (Base + 0)



The interrupt register contains the interrupt state and can only be read. The initial values are random and are defined through an external reset (RESNX = low). They are set on logic "0".

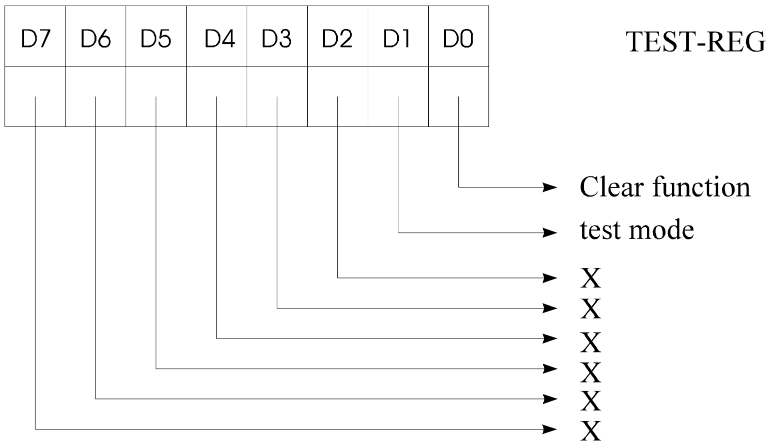
When a value has been latched, the value 1 indicates an active state. Thereby are differentiated the latch group, the internal and external latch triggering.

The bits D1 and D5 have the same meaning as output pins INTRx1 and INTRx2. A display occurs only after triggering at inputs EXSTBx1 or EXSTBx2 (= low) and after the latch operation has ended. Also the bits D0 and D4 are active after a software strobe, only if the latch operation has ended internally.

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

Caution!
If a new strobe operation is triggered during the read operation, a new latch pulse is generated. But it is neither displayed in the interrupt register nor at the interrupt output signals INTRx1 and INTRx2. In this special case the data read may be wrong.

Test register (Base + 12)



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

A test mode is intended for testing the component and the connected peripheral. All 8-bit counter chains are operated internally as down 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.

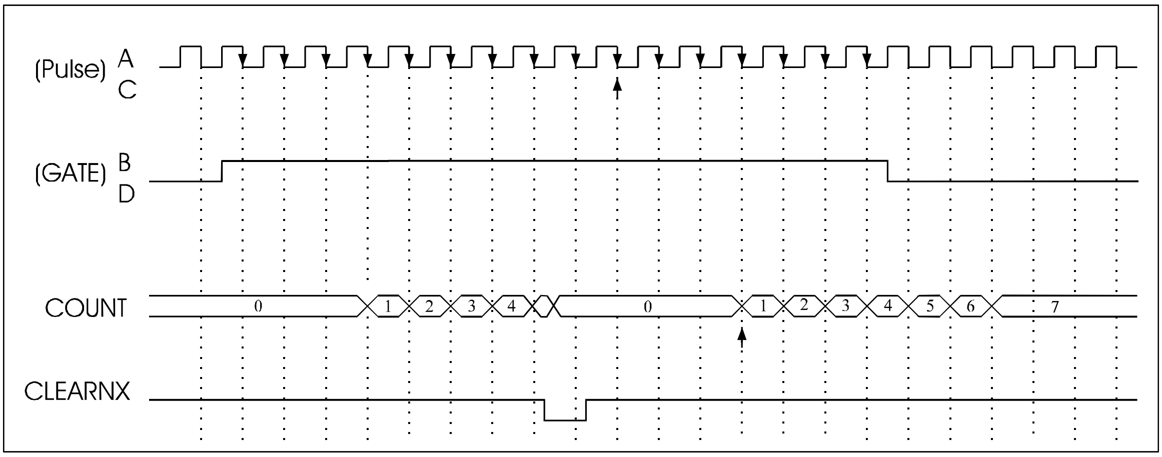
The test mode is left, either when a RESET operation has been triggered or when the value "0" is written in bit 1 of the test register.

The clear function resets automatically the bit 0 of the test register after execution. On this bit position only a write access is meaningful.

Clear logic

An external clear signal at input CLRNX (= low) resets immediately all counter chains. Removing this signal is synchronised in operation.

Fig. 8-12: 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 soft clear (see test register) begins the internal clear operation after the write operation has been ended (WRNX = High). It is lasting 5 negative clock pulse edges.
It means that an internal counting operation can occur again at the earliest with the 6th negative edge.

Reset logic

By activating the input signal RESNX (= low) the control logic of the circuit is set on the basic MODE.

- 32-bit counter
- Edge analysis circuit A in quadruple-MODE
- Hysteresis off
- Control unit B inactive
- Output signal RSTATNX and bit 8 of MODE register 2 on L
- No influence of the counter chains

The internal reset state is immediately left after the input signal RESNX (= high) has become inactive.
The output signal RSTATNX and bit 8 of the MODE-Register 2 remain active until this bit is set on "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).
It means that data can be processed again by the control unit with the 5th negative clock pulse edge.

Loading the counter chains

The 16/32-bit-counter can be loaded through the 16-bit data bus by words.

The corresponding memory locations are selected through the address bus A0X..A3X in accordance with the 16-bit I/O map

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

When loading the counter chains, interrupt the counting operation to avoid counting errors.

See also Chapter 5 "Settings" for the corresponding jumpers.

8.4.2 Programming examples

With the board **PA 1700-2** are delivered drivers, which are written in the current languages like Turbo Pascal, C, PDS Basic.
You can link these drivers in your application. They allow to respond the functionality of the board.

- Storing the counter content in the data latch register 1 or 2
  Example: i_PA1700_LatchCounter
- Reading the counter content in the 16-bit or 32-bit mode.
  Example: i_PA1700_Read16BitCounterValue
  i_PA1700_Read32BitCounterValue
- Writing the counter content in the 16-bit or 32-bit mode.
  Example: i_PA1700_Write16BitCounterValue
  i_PA1700_Write32BitCounterValue
- Clearing the counter content in the 32-bit mode (clear function).
  Example: i_PA1700_ClearCounterValue
  i_PA1700_ClearAllCounterValue

8.5 Interface to the incremental encoders

Up to three incremental encoders can be connected through the front connector:

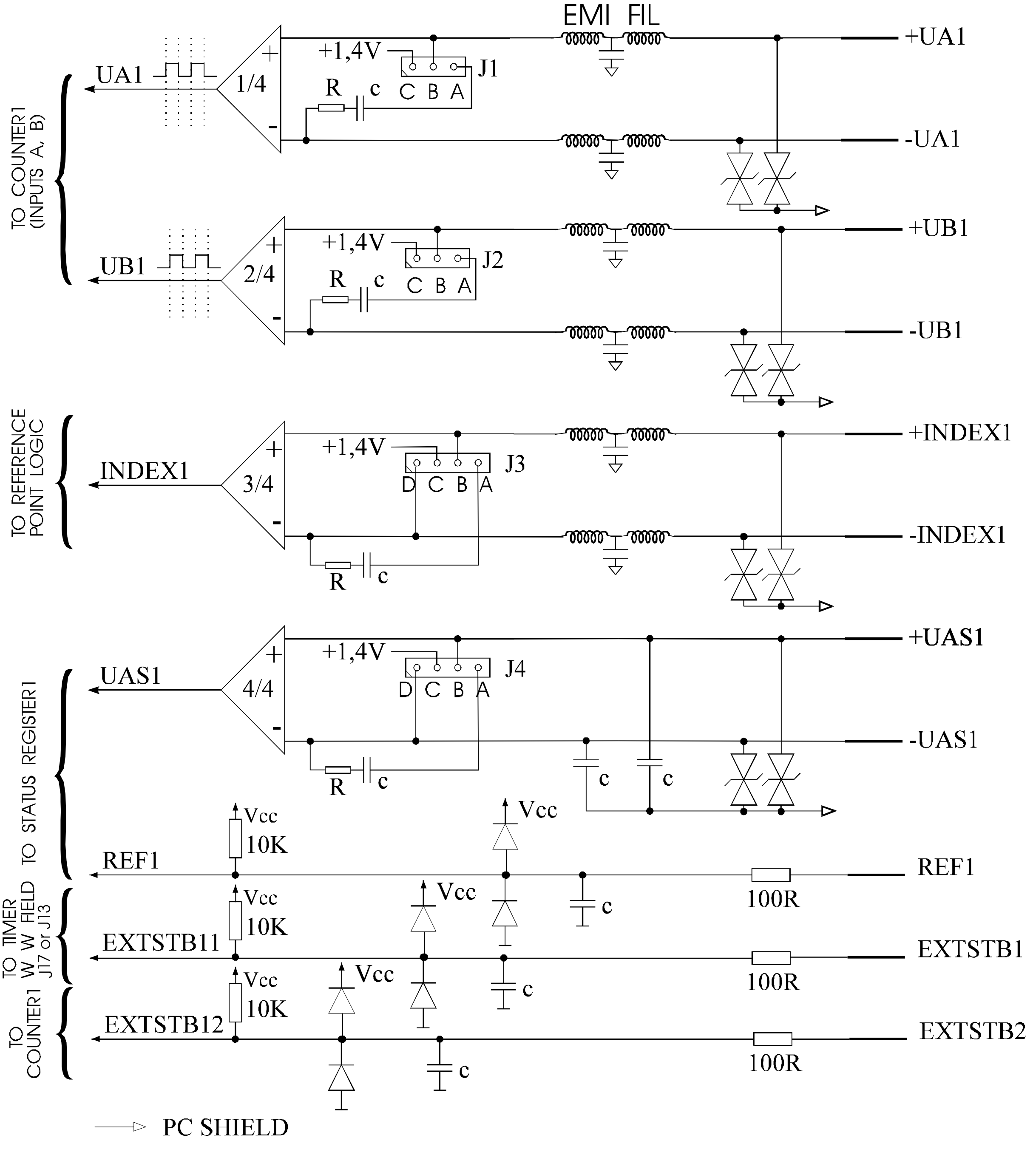
- Incremental encoder 1 is led to counter module 1 (16-bit counter channel1 or 32-bit counter channel1).
- Incremental encoder 2 is led to counter module 2 (16-bit counter channel3 or 32-bit counter channel2).
- Incremental encoder 3 is led to counter module 3 (16-bit counter channel5 or 32-bit counter channel3).

The signals available on each counter channel are:

- UAx, /UAx: A and not A signals of the incremental encoder (Diff. / TTL).
- UBx, /UBx: B and not B signals of the incremental encoder (Diff./TTL).
- UINDEXx, /UINDEXx: INDEX and not INDEX signals of the incremental encoder (Diff./TTL).
- UASx, /UASx: ex. error signals of the incremental encoder (Diff./TTL).
- UREFx: TTL Input.
- EXTSTBx1: TTL Input, stores counter module x in data latch register 1.
- EXTSTBx2: TTL Input, stores counter module x in data latch register 2.
  $x \in [1..3]$

Up to three other incremental encoders can be connected through wire wrap field J17. A and B signals are available to connect two phase-shifted signals.

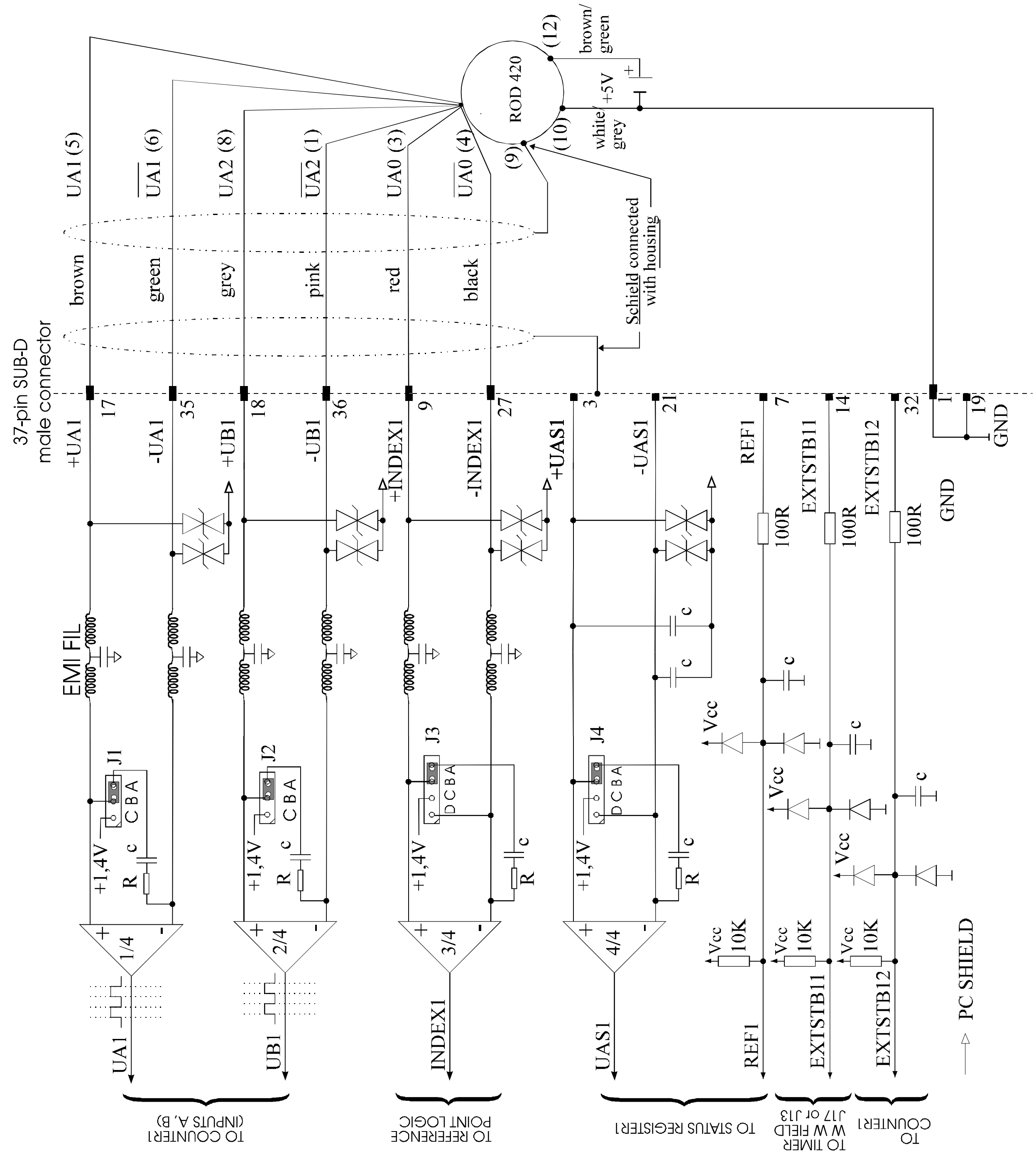
Fig. 8-13: Set up figure of a counter channel



8.5.1 Connection to a shift encoder ROD 420

The encoder ROD 420 of HEIDENHAIN is connected to the board channel 1. Differential signals for incremental information and reference signals.

Fig. 8-14: Connection to HEIDENHAIN encoder ROD 420



Jumper settings

The ±UA signals are either TTL or Diff., configurable with field J1, J5, J9.
The ±UB signals are either TTL or Diff., configurable with field J2, J6, J10.
The ±INDEX signals are either TTL or Diff., configurable with field J3, J7, J11.
If the input signal is used as TTL, the signal can also be inverted.
The ±UAS signals are either TTL or Diff., configurable with field J4, J8, J12.
If the input signal is used as TTL, the signal can also be inverted.

See Chapter 5 "Settings" for the settings of the corresponding jumpers

Programming examples

With the board **PA 1700-2** are delivered drivers, which are written in the current languages like Turbo Pascal, C, PDS Basic.

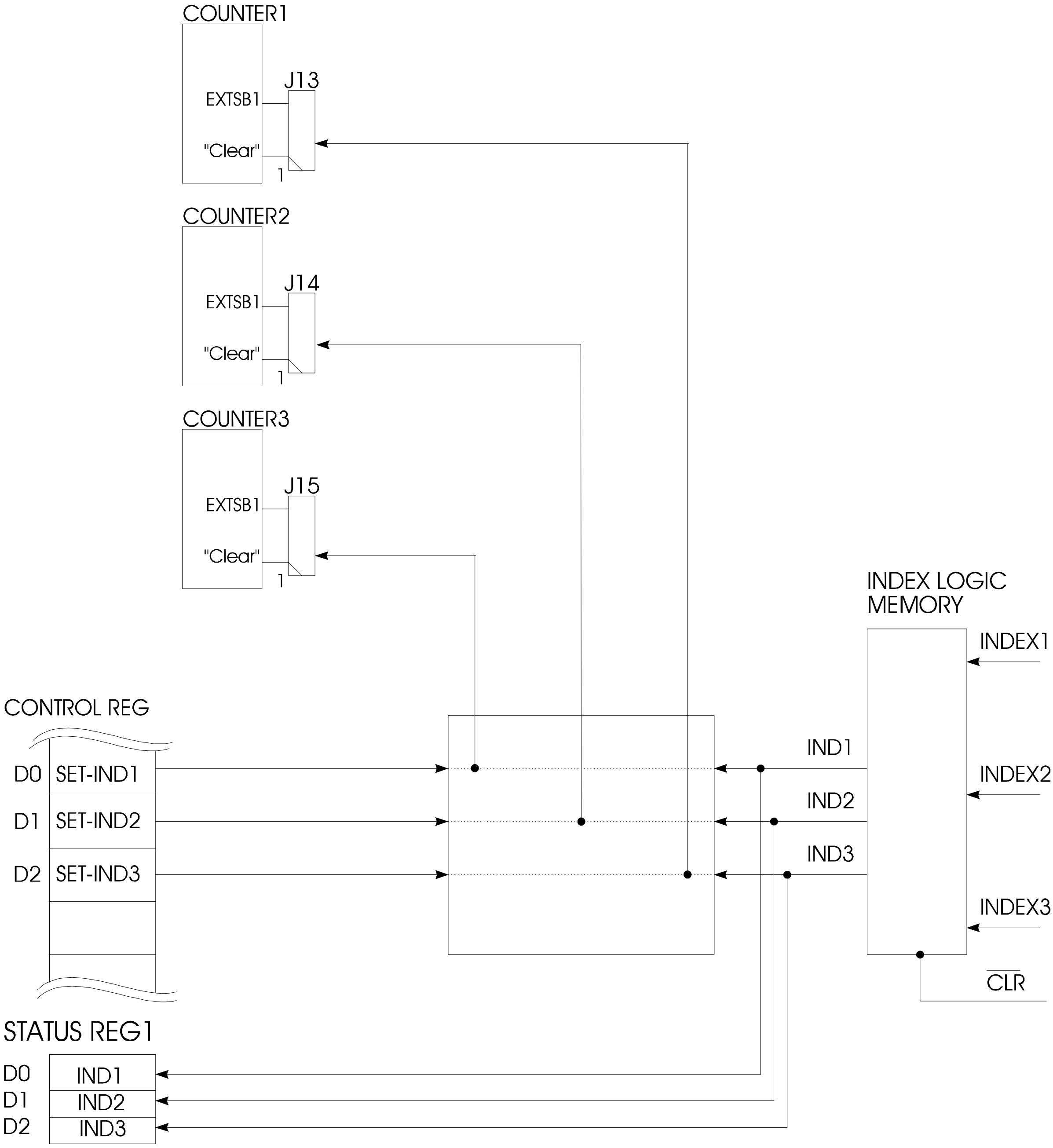
You can link these drivers in your application. They allow to respond the functionality of the board.

- Reading UASx error inputs
  Example: i_PA1700_GetUASStatus
- Reading REFx inputs
  Example: i_PA1700_GetReferenceStatus

8.6 Reference point logic

There is for each counter module of board **PA 1700-2** a reference point logic. It allows clearing or storing the counter content according to the jumper setting.

Fig. 8-15: Reference point logic



With jumpers J13, J14, J15 you determine for each counter module if the counter content is cleared or stored.

Each counter module x is assigned an input called INDEX-x at the SUB-D front connector.

This input can be used as a TTL or differential input. It can act as:
- the connection to a reference mark,
- an index signal of a ruler,
- an encoder to perform a reference displacement of the system.

If the reference displacement of the counter module-x has been enabled with the SET-INDx bits of the control register (**Base + 24**) and according to the jumper setting, the counter module is cleared or stored if the pulse INDEXx is available.

The INDEX-x input is stored in an intermediate latch, independently of the fact if the reference displacement has been enabled or not. A read of STATUS-REG1 (**Base + 24**) clears the store content.

A reference displacement consists of the following steps:
- Setting with jumper J13, J14, J15: storing or clearing
- Reading STATUS-REG1 (**Base + 24**) for clearing references which have been eventually stored previously
- Setting the bits SET-IND for defining with which counter module the reference displacement will be performed.
- Moving the axis(axes) until one or all INDEX-x signals become active
- Clearing again the IND-x bits by reading STATUS-REG.1 (**Base + 24**).

Remark:
If the counter content has been stored, an interrupt is generated to the PC bus through the interrupt logic (if enabled by bit INT-EN1).

This interrupt is intended for reading the stored value in the PC main memory for being processed.

Special case:
More than three counter channels are used (three via the front connector, others via WW field J17). A reference displacement must be performed with each counter channel. In this case it is only possible to store the counter contents with the EXTSTBxy signals.

- Counter channel1 (16 bit) → EXTSTB11
- Counter channel2 (16 bit) → EXTSTB21
- Counter channel3 (16 bit) → EXTSTB31
- Counter channel4 (16 bit) → EXTSTB12
- Counter channel5 (16 bit) → EXTSTB22
- Counter channel6 (16 bit) → EXTSTB32

The counter contents are stored in the corresponding data latch register (1 or 2) (**base + 2, 4, 6, 8**).
The memory process is represented by logic "1"in bits D1 and D5 of INT-REG (**base + 0**) of the corresponding counter module. These bits produce an interrupt request to the board interrupt logic. They are reset with a read on the data latches (**base + 2, +4, +6, +8**).

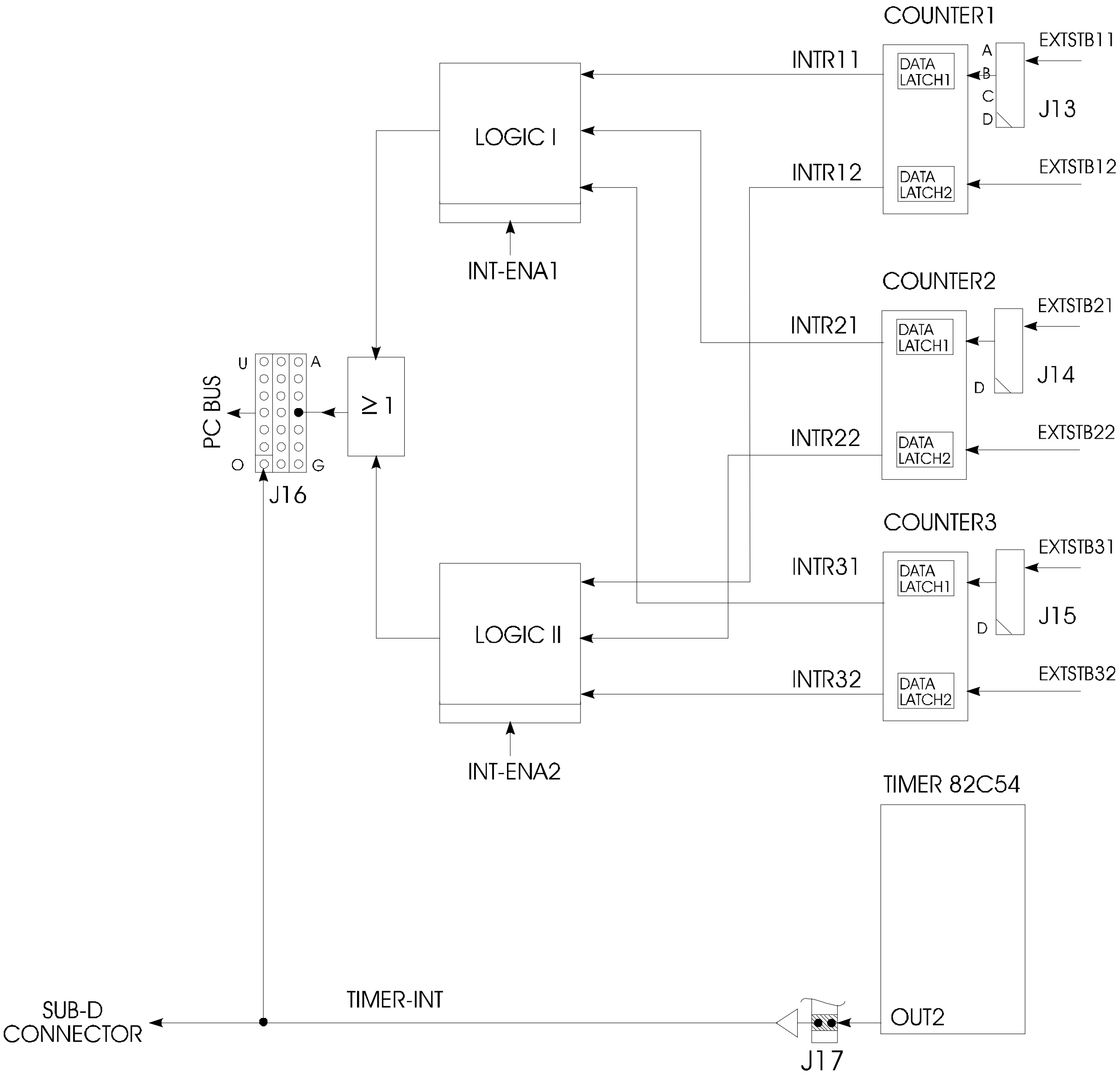
Programming examples

With the board **PA 1700-2** are delivered drivers, which are written in the current languages like Turbo Pascal, C, PDS Basic.
You can link these drivers in your application. They allow to respond the functionality of the board.

- Reading the INDEXx inputs and thus clearing the index store
  Example: i_PA1700_GetIndexStatus
- Enabling the Set-Index bits for the reference displacement
  Example: i_PA1700_InitIndex

8.7 Interrupt logic

Fig. 8-16: Interrupt logic



The board **PA 1700-2** can transmit two interrupt requests to the PC:

- TIMER-INT

- STROBE-INT

These two interrupt sources can be connected through WW field J16 to the interrupt lines IRQ3, 5, 10, 11, 12, 14, 15 with the AT bus.

8.7.1 Timer interrupt

The board **PA 1700-2** can transmit cyclically an interrupt to the PC bus. The time interval between two interrupts can be programmed through the timer component 82C54.
The GATE2 bit of CONTROL-REG (**base + 24**) can be used for synchronising the interrupt by software.

8.7.2 Strobe interrupt

The strobe interrupt is a common interrupt for all interrupt lines of the three counter modules. These interrupt signals are divided into two groups. Each group can be enabled through one bit of CONTROL-REG (**base + 24**).

Group 1 is enabled with bit INT-EN1.

An interrupt is transmitted to the PC, when the counter content has been stored by an EXTSTBx1 signal (low level).
The interrupt request is stored by a bit in INT-REG (**base + 0**) of the corresponding counter module.
Resetting occurs when the data latch register 1 (**base +2, +4**) of the corresponding counter module is read.

Group 2 is enabled by bit INT-EN2.

An interrupt is transmitted to the PC, when the counter content has been stored by an EXTSTBx2 signal (low level).
The interrupt request is stored by a bit in INT-REG (**base + 0**) of the corresponding counter module.
Resetting occurs when the data latch register 1 (**base +6, +8**) of the corresponding counter module is read.
Since the interrupt request is concerning a common interrupt, you must at first detect in the interrupt service routine the origin of this interrupt.
It occurs by reading the bits D1, D5 of INT-REG (**base + 0**) of the counter modules.
If one of these bits is on logic "1", the corresponding counter module has transmitted an interrupt request.
Before a new interrupt can be produced, all counter modules must be tested on an external strobe and the corresponding data latch register (**base +2, +4, +6, +8**) must be read.

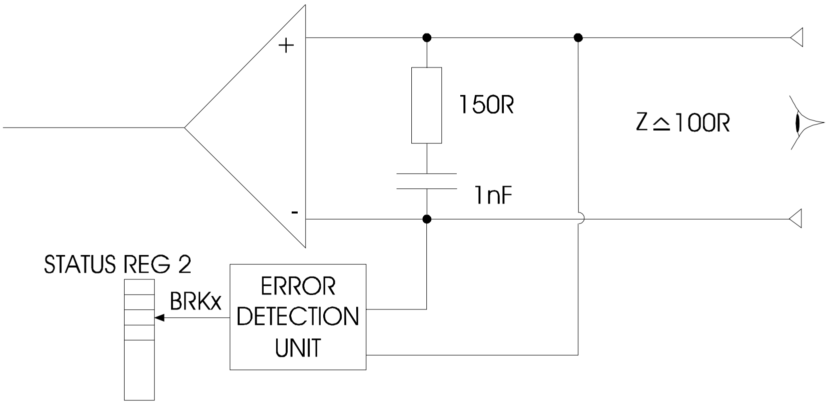
Programming examples

With the board **PA 1700-2** are delivered drivers, which are written in the current languages like Turbo Pascal, C, PDS Basic.
You can link these drivers in your application.
They allow to respond the functionality of the board.

- Enabling the strobe interrupt
  Example: i_PA1700_EnableLatchInterrupt
- Disabling the strobe interrupt
  Example: i_PA1700_DisableLatchInterrupt

8.8 Line interruption recognition

Fig. 8-18: Line interruption recognition



The logic for line interruption recognition is only active if the inputs are operated in differential mode.

Each pair of differential lines has a line interruption supervision and a bit BRKx in STATUS-REG2 (**base + 26**).

These bits indicate if a line interruption has been recognised.
The logic for line interruption recognition does not store states but functions statically. Only statically available states are displayed.

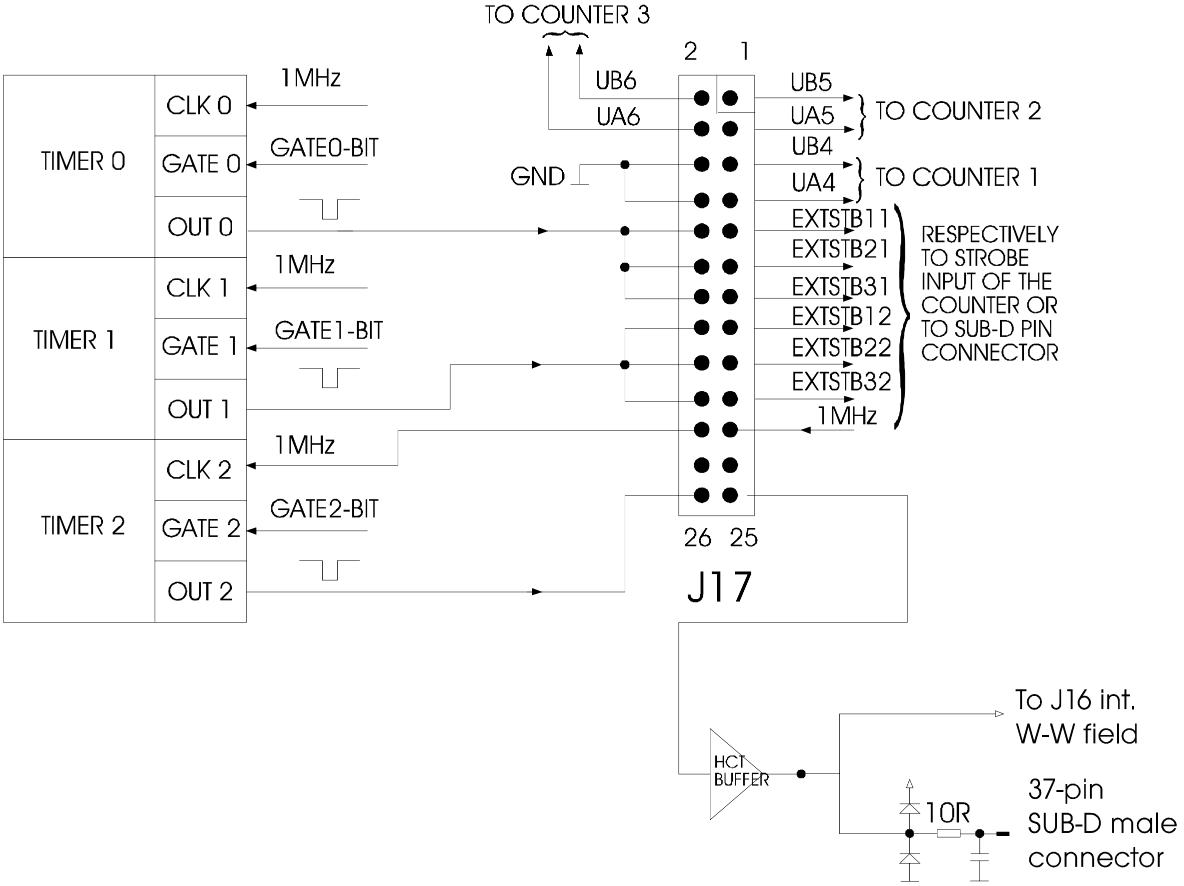
You will find in the following table the input lines and the corresponding BRKx bit:

Signal name	Bit name	Bit in STATUS-REG2
±UA1	BRK0	D2
±UB1	BRK1	D3
±UIND1	BRK2	D4
±UAS1	BRK3	D5
±UA2	BRK4	D6
±UB2	BRK5	D7
±UIND2	BRK6	D8
±UAS2	BRK7	D9
±UA3	BRK8	D10
±UB3	BRK9	D11
±UIND3	BRK10	D12
±UAS3	BRK11	D13

This supervision logic functions only if a differential voltage > 2.5V is available at the differential input. This logic needs a current of approximately 5 mA at a differential voltage of 5 V.

8.9 Timer component

Fig. 8-19: Timer component



This function is realised by the timer component 82C54.
This component consists of three programmable 16-bit timers.
Two timers (Timer0 and Timer1) can be used on board **PA 1700-2** for producing a strobe signal controlled by the timer.
Thus differential measures can be performed directly.

The third timer can function as a time interrupt generator.
Each of the three timers can be programmed independently from the others.

The component needs to be initialised by software for choosing the wished function.

The component is ready for being programmed immediately after applying the working voltage. You program through the addresses **Base + 16** to **Base + 19** in the 8-bit mode.

With the bits GATE0-2 of CONTROL-REG (Base + 24) you can control the gate inputs of the corresponding timer.

8.9.1 Timer2 as a time interrupt generator

Interrupts can be generated with this timer function.
Previously set the jumpers according to chapter 5 Jumper settings.

8.9.2 Timer0, Timer1 as a time generator for controlling strobe

Strobes can be produced with this timer function.
According to the jumper settings of J17, J13, J14 and J15, timer0 allows to generate cyclically a strobe pulse at the external strobe input 1 of the corresponding counter module.

This pulse causes that the content of a counter is stored in the data latch register 1. The interrupt request resulting from this can be used for reading the stored value(s) if enabled with bit INT-EN1-Bit (**base + 24**).

According to the jumper setting of J17, timer1 allows to generate cyclically a pulse at the external strobe input 2 of the corresponding counter module.
This pulse causes that the content of a counter is stored in the data latch register 2.

The interrupt request resulting from this can be used for collecting the stored value(s) if enabled with bit INT-EN1-Bit (**base + 24**).

Remark:
If the timer component works as a strobe signal generator, the strobe signals are then also available at pins EXTSTBxx of the SUB-D front connector. It means that the EXTSTBxx pins used can be simultaneously also timer outputs.

CAUTION!
The counter modules can only be stored through the inputs EXTSTBxx or, exclusively, through the timer(s).

Programming examples

With the board **PA 1700-2** are delivered drivers, which are written in the current languages like Turbo Pascal, C, PDS Basic.
You can link these drivers in your application. They allow to respond the functionality of the board.

- Initialising the timers
  Example: i_PA1700_InitTimer,
  i_PA1700_GetIndexStatus
- Reading a timer value
  Example: i_PA1700_ReadTimerValue;
  i_PA1700_ReadAllTimerValue
- Writing a value into a timer
  Example: i_PA1700_WriteTimerValue
- Storing the timer content
  Example: i_PA1700_ReadTimerValue;
  i_PA1700_ReadAllTimerValue

Example in Pascal

Task: a cyclic interrupt signal of 100Hz is needed. Calculate the divider factor:

T = fin / fout ; T = 1MHz / 100Hz = 10.000

```
Procedure Timer-Start (divider : word);
CONST      Timer-Mode = $B4;            (* Timer 2 in mode 2     *)
VAR        Lowbyte , Highbyte : Byte;
Begin
   Lowbyte : = divider Mod 256;
   Highbyte:= divider Div 256;
   Port [ base + 19 ] : = Timer_Mode;       (* TIMER 2 control byte *)
   Delay(1);
   Port [ base + 18 ] := Lowbyte;          (* TIMER 2 initial value *)
                                          (* low byte            *)
   Delay(1);
   Port [ base + 18 ] := Highbyte;         (* TIMER 2 initial value *)
                                          (* high byte           *)
End;

--------------------------------------------------------------------
                                    MAIN

--------------------------------------------------------------------
BEGIN
    clrscr;
    value  := 0;
    Install_New_Irq_3;                    (* Install the NEW IRQ *)
    Timer_Start ( 28 );                   (* Timer conversion every
10 mS *)
    Portw [base+24]:= shadow_Reg OR $010; (* GATE2 = "1", Start of
TIMER2 *)
                                          (*  User application   *)
    Desinstall_Irq_3;
END.
```

8.10 Parallel input and output (TTL)

8.10.1 Input and output channels, TTL

The board **PA 1700-2** allows to process simultaneously 24 TTL input and output signals.

There is no optical isolation. The direction of the signal inputs and outputs can be programmed freely.

The interface consists of the parallel port component 82C55A. Because of its programmability this component offers a wide range of applications.

The parallel port is divided into three groups: port A, B and C. (8-bit width).

Each of these three port groups can be configured independently from the others: as input, output or port A and bi-directional. The port C functions as a control port in several modes.

The component needs to be initialised by software for choosing the wished function.

The component is ready for being programmed immediately after applying the working voltage and a successful Power On/Reset.

All port signals are available in jumper field X35. You will find the location of the base address in the I/O map.

Programming examples

The port functions of board **PA 1700-2** are responded through the following addresses. Programming occurs per bytes in the 8-bit MODE:

Base + 20	Port A
Base + 21	Port B
Base + 22	Port C
Base + 23	Port Status

"Base" designates the board's base address set with the DIP switch.
The base address is at 0300H at delivery.

The above mentioned Port Status is responded in this case with address 0317H.

Task: All port groups = TTL outputs

Solution 1: AS7M86

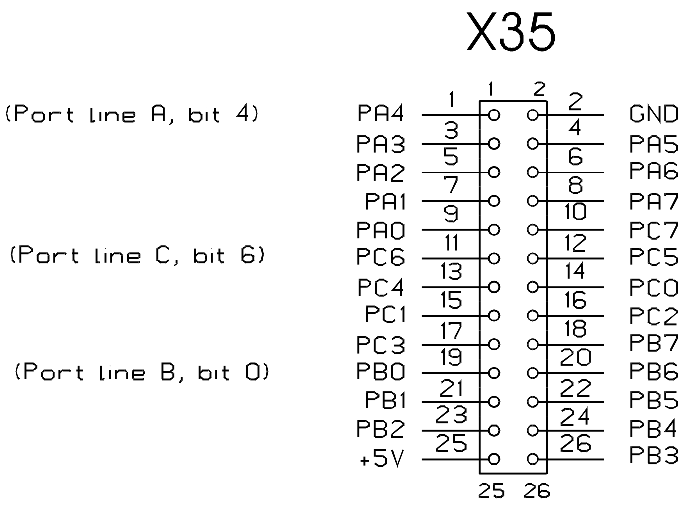
```
.
.
.
MOV    DX, 030BH          ; Port status address
MOV    AL, 80H            ; Three ports as TTL outputs (mode 0)
OUT    DX, AL             ; Programmation
MOV    DX, 0308H          ; Load address port A
MOV    AL, DATEA          ; Output data for port A
OUT    DX, AL             ; Set the outputs of port A
INC    DX                 ; Set the address of port B
MOV    AL, DATEB          ; Output data for port B
OUT    DX, AL             ; Set the outputs of port B
INC    DX                 ; Set the address of port C
MOV    AL, DATEC          ; Output data for port C
OUT    DX, AL             ; Set the outputs of port C
.
.
.
```

Solution 2: BASIC

```
OUT&H30B,&H80         ' Program mode
OUT&H308,&Hxx         ' xx = output data for port A
OUT&H309,&Hxx         ' xx = output data for port B
OUT&H30A,&Hxx         ' xx = output data for port C
...
```

8.10.2 Interface

Fig. 8-20: Jumper field X35 / pin assignment



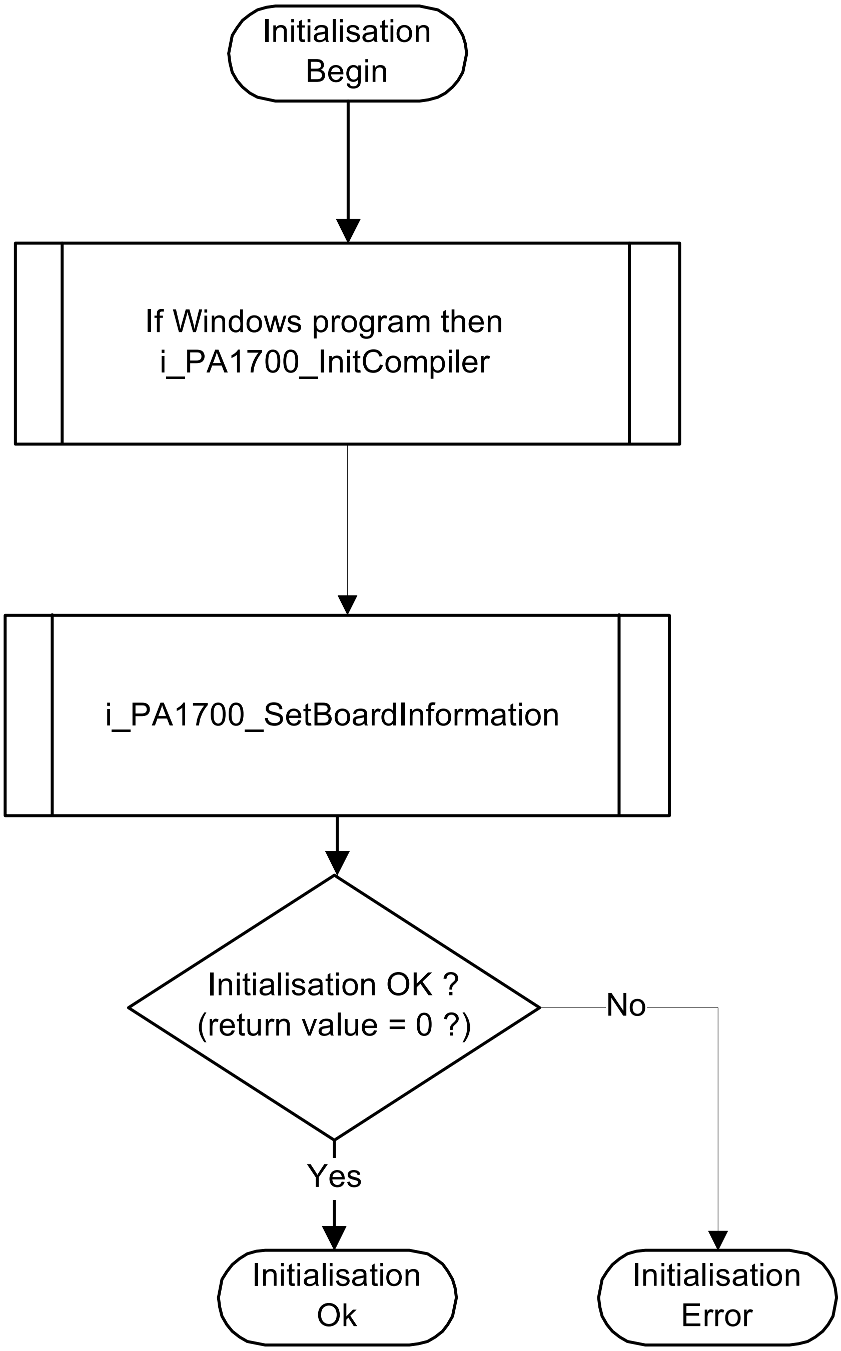
The connector designations of the TTL signals correspond to the port designation.

9 SOFTWARE EXAMPLES

9.1 Initialisation

9.1.1 Initialisation of the PA 1700 under DOS and Windows 3.11

a) Flow chart

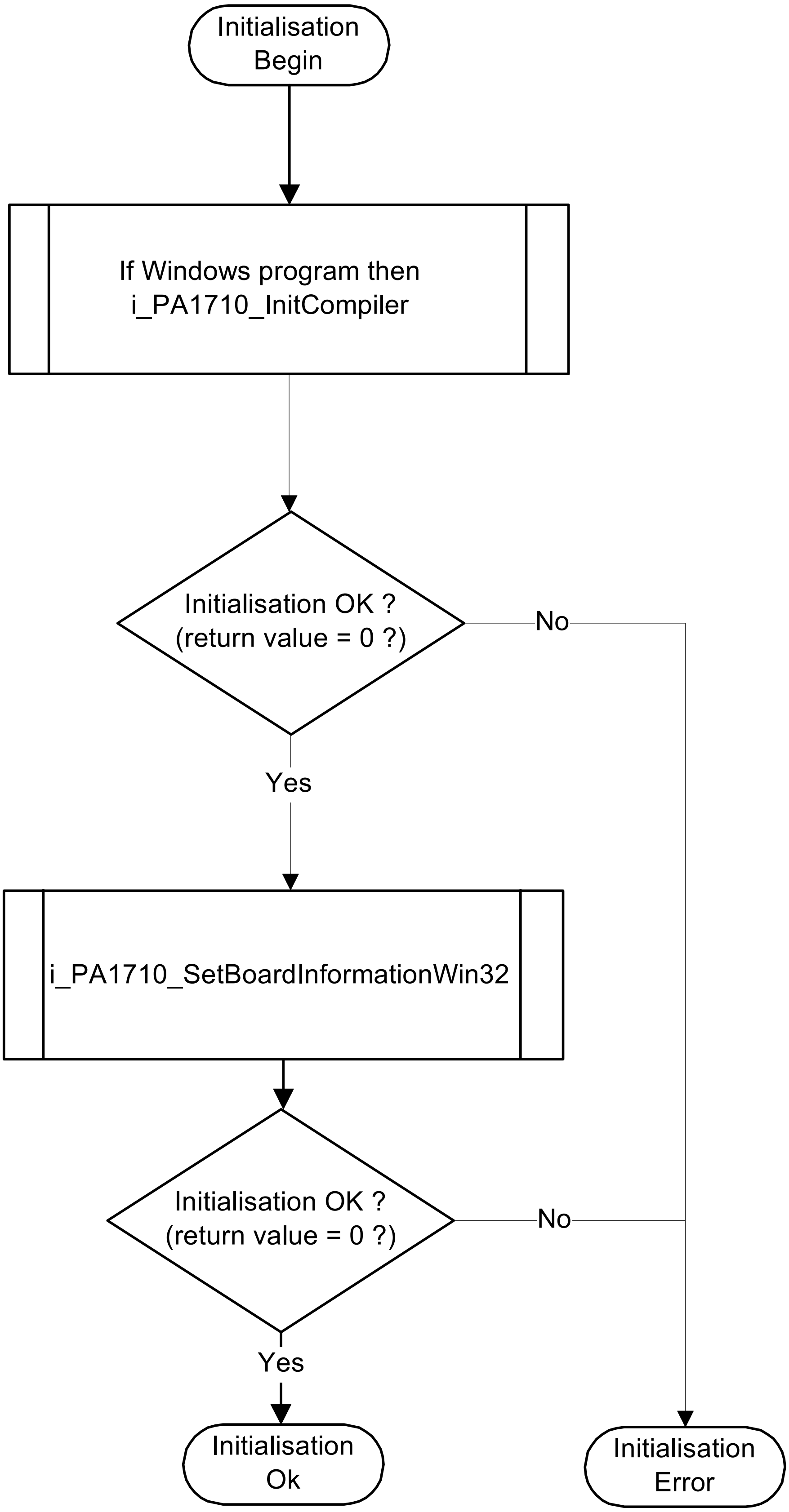

Initialisation Begin
If Windows program then i_PA1700_InitCompiler
i_PA1700_SetBoardInformation
Initialisation OK ? (return value = 0 ?)
No
Yes
Initialisation Ok
Initialisation Error

b) Example in C for DOS and Windows 3.11

```
int Initialisation(unsigned char *pb_BoardHandle)
    {
    #ifdef _Windows
       i_PA1700_InitCompiler (DLL_COMPILER_C);
    #endif

    if(i_PA1700_SetBoardInformation (0x390,
                  3,
                  5) == 0)
       {
       return (0);                  /* OK */
       }
    else
       {
       return (-1);                 /* ERROR */
       }
    }
```

9.1.2 Initialisation of the PA 1700 under Windows NT/95/98

a) Flow chart



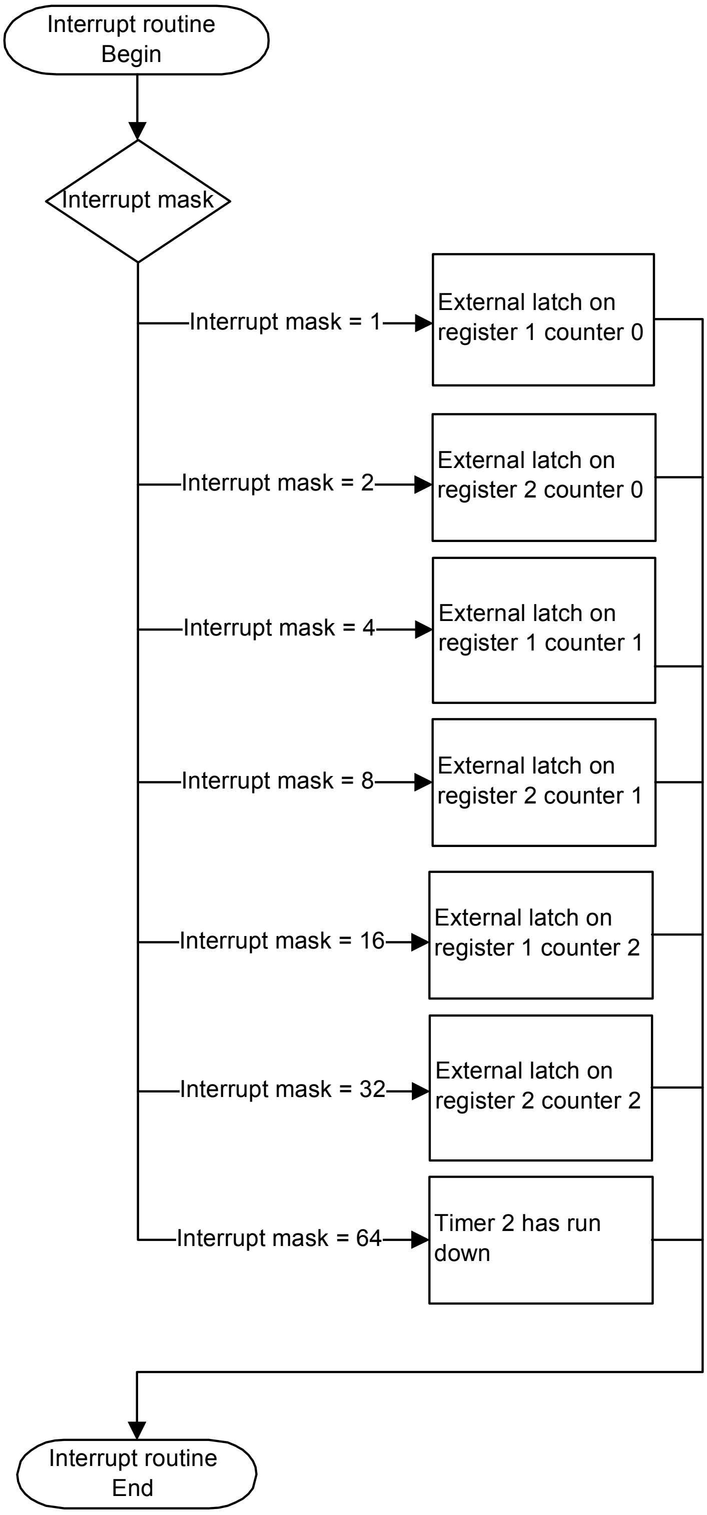
b) Example in C for Windows NT / 95/98

```
int Initialisation(unsigned char *pb_BoardHandle)
    {
    if (i_PA1700_InitCompiler (DLL_COMPILER_C) == 0)
       {
       if(i_PA1700_SetBoardInformationWin32 (“PA1700-00“,
               pb_BoardHandle) == 0)
          {
          return (0);                           /* OK */
          }
       else
          {
          return (-1);                          /* ERROR */
          }
       }
    else
       {
       return (-1);                             /* ERROR */
       }
    }
```

9.2 Interrupt

9.2.1 Interrupt routine

a) Flow chart for DOS, Windows 3.11 and Windows NT/95/98 (asynchronous mode)


Interrupt routine Begin
Interrupt mask
Interrupt mask = 1
External latch on register 1 counter 0
Interrupt mask = 2
External latch on register 2 counter 0
Interrupt mask = 4
External latch on register 1 counter 1
Interrupt mask = 8
External latch on register 2 counter 1
Interrupt mask = 16
External latch on register 1 counter 2
Interrupt mask = 32
External latch on register 2 counter 2
Interrupt mask = 64
Timer 2 has run down
Interrupt routine End

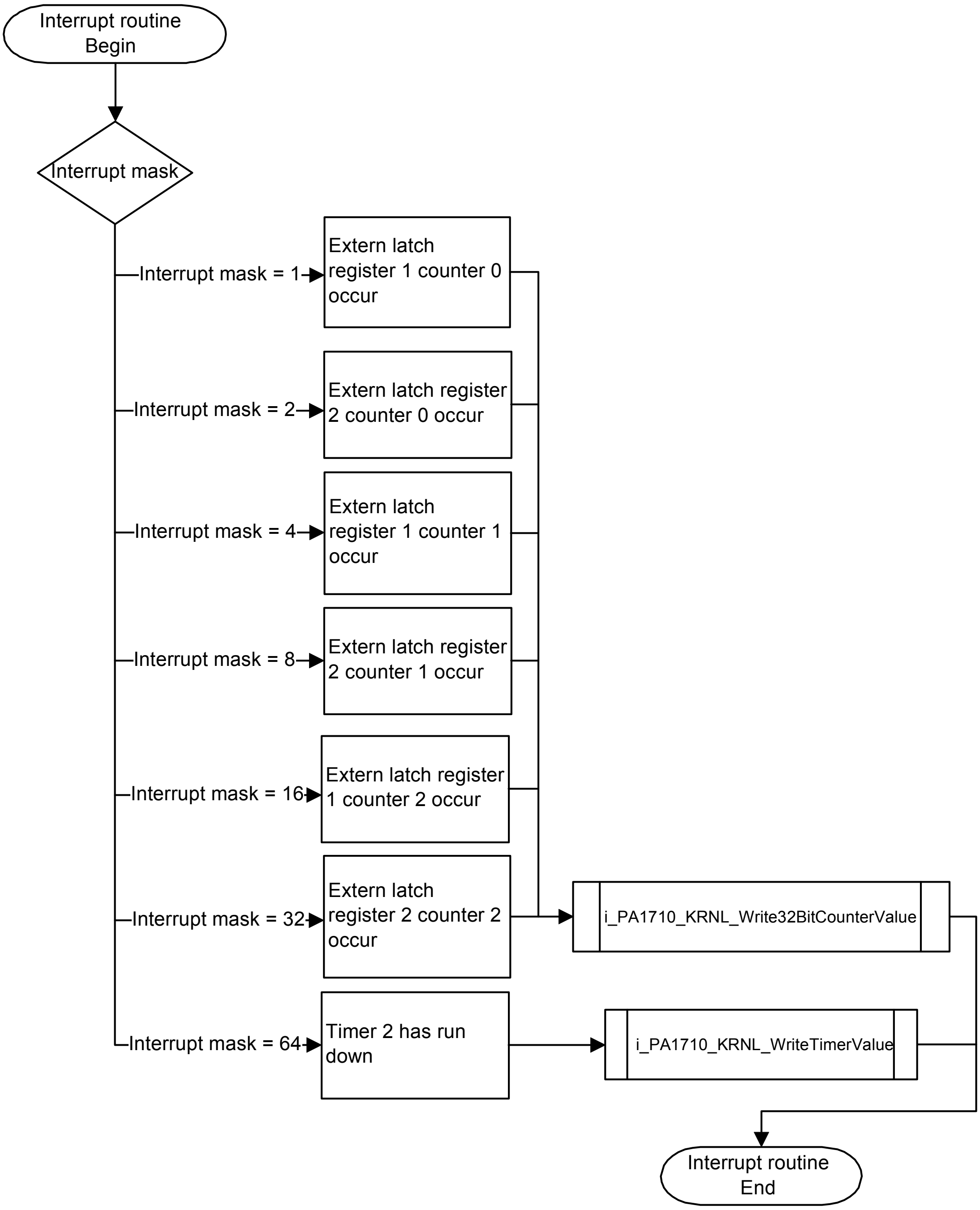
b) Example in C for DOS and Windows 3.11

```
unsigned long ul_CounterValue [3] [2];                     /* Global buffer          */
unsigned char  b_IntHardCpt         = 0;                   /* Hardware counter */
unsigned char  b_IntLatchReg        = 0;                   /* Latch register         */
unsigned char  b_ReceiveInterrupt   = 0;                   /* Interrupt flag         */
unsigned long ul_TimerInterruptCpt  = 0;                   /* Timer 2 interrupt      */

_VOID_ v_InterruptRoutine       (BYTE_ b_BoardHandle,
                                 BYTE_ b_InterruptMask,
                                 ULONG_ ul_CounterLatchValue)
   {
   switch (b_InterruptMask)
      {
      case 0x1:
           b_IntHardCpt  = 0;
           b_IntLatchReg = 1;
           ul_CounterValue [0][0] = ul_CounterLatchValue;
           break;

      case 0x2:
           b_IntHardCpt  = 0;
           b_IntLatchReg = 2;
           ul_CounterValue [0][1] = ul_CounterLatchValue;
           break;

      case 0x4:
           b_IntHardCpt  = 1;
           b_IntLatchReg = 1;
           ul_CounterValue [1][0] = ul_CounterLatchValue;
           break;

      case 0x8:
           b_IntHardCpt  = 1;
           b_IntLatchReg = 2;
           ul_CounterValue [1][1] = ul_CounterLatchValue;
           break;

      case 0x10:
           b_IntHardCpt  = 2;
           b_IntLatchReg = 1;
           ul_CounterValue [2][0] = ul_CounterLatchValue;
           break;

      case 0x20:
           b_IntHardCpt  = 2;
           b_IntLatchReg = 2;
           ul_CounterValue [2][1] = ul_CounterLatchValue;
           break;

      case 0x40:
           ul_TimerInterruptCpt = ul_TimerInterruptCpt + 1;
           break;
      }

   b_ReceiveInterrupt = 1;
   }
```

c) Example in C for Windows NT/95/98 (asynchronous mode)

```
unsigned long  ul_CounterValue [3] [2];                    /* Global buffer          */
unsigned char  b_IntHardCpt         = 0;                   /* Hardware counter */
unsigned char  b_IntLatchReg        = 0;                   /* Latch register         */
unsigned char  b_ReceiveInterrupt   = 0;                   /* Interrupt flag         */
unsigned long ul_TimerInterruptCpt  = 0;                   /* Timer 2 interrupt      */

_VOID_ v_InterruptRoutine       (BYTE_  b_BoardHandle,
                                 BYTE_  b_InterruptMask,
                                 ULONG_ ul_CounterLatchValue
                                 BYTE_   b_UserCallingMode,
                                 VOID  *pv_UserSharedMemory)
   {
   switch (b_InterruptMask)
      {
      case 0x1:
           b_IntHardCpt  = 0;
           b_IntLatchReg = 1;
           ul_CounterValue [0][0] = ul_CounterLatchValue;
           break;

      case 0x2:
           b_IntHardCpt  = 0;
           b_IntLatchReg = 2;
           ul_CounterValue [0][1] = ul_CounterLatchValue;
           break;

      case 0x4:
           b_IntHardCpt  = 1;
           b_IntLatchReg = 1;
           ul_CounterValue [1][0] = ul_CounterLatchValue;
           break;

      case 0x8:
           b_IntHardCpt  = 1;
           b_IntLatchReg = 2;
           ul_CounterValue [1][1] = ul_CounterLatchValue;
           break;

      case 0x10:
           b_IntHardCpt  = 2;
           b_IntLatchReg = 1;
           ul_CounterValue [2][0] = ul_CounterLatchValue;
           break;

      case 0x20:
           b_IntHardCpt  = 2;
           b_IntLatchReg = 2;
           ul_CounterValue [2][1] = ul_CounterLatchValue;
           break;

      case 0x40:
           ul_TimerInterruptCpt = ul_TimerInterruptCpt + 1;
           break;
      }

   b_ReceiveInterrupt = 1;
   }
```

d) Flow chart for Windows NT/95/98 (synchronous mode)


Interrupt routine Begin
Interrupt mask
Interrupt mask = 1
Extern latch register 1 counter 0 occur
Interrupt mask = 2
Extern latch register 2 counter 0 occur
Interrupt mask = 4
Extern latch register 1 counter 1 occur
Interrupt mask = 8
Extern latch register 2 counter 1 occur
Interrupt mask = 16
Extern latch register 1 counter 2 occur
Interrupt mask = 32
Extern latch register 2 counter 2 occur
i_PA1710_KRNL_Write32BitCounterValue
Interrupt mask = 64
Timer 2 has run down
i_PA1710_KRNL_WriteTimerValue
Interrupt routine End

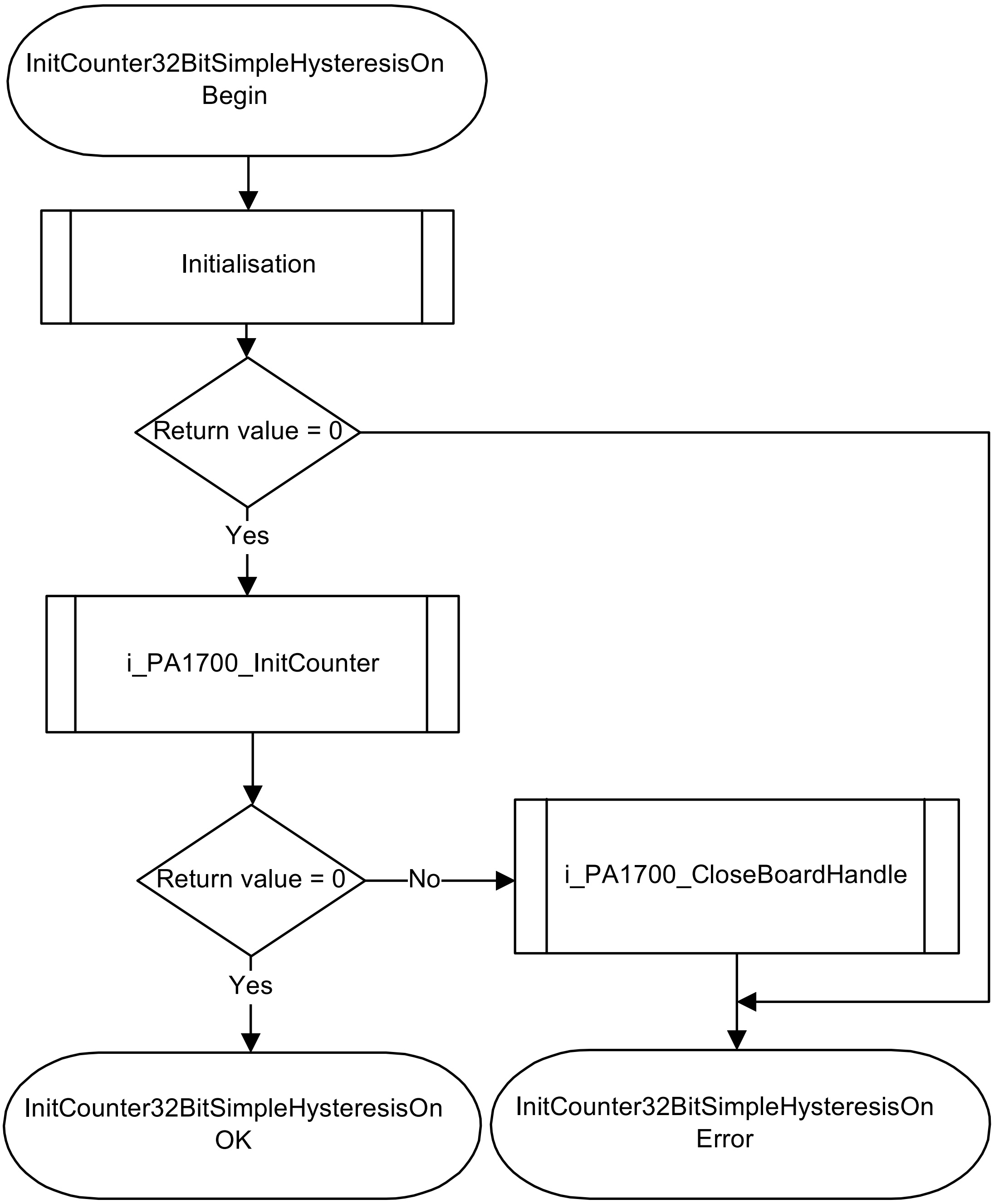
e) Example in C for Windows NT/95/98 (synchronous mode)

```
typedef struct
    {
    unsigned int   ui_BaseAddress;          /* PA1700 base address*/
    unsigned long  ul_CounterValue [3] [2];   /* Global buffer     */
    unsigned char b_IntHardCpt;             /* Hardware counter  */
    unsigned char b_IntLatchReg;            /* Latch register    */
    unsigned char b_ReceiveInterrupt;              /* Interrupt flag     */
    unsigned long ul_TimerInterruptCpt;     /* Timer 2 interrupt */
    unsigned char  b_CBCounter;             /* CB cpounter               */
    }str_UserStruct;
str_UserStruct *ps_GlobalUserStruct;

_VOID_ v_InterruptRoutine     (BYTE_  b_BoardHandle, BYTE_  b_InterruptMask,
                               ULONG_ ul_CounterLatchValue,       BYTE_ b_UserCallingMode,
                               VOID  *pv_UserSharedMemory)
    {
    str_UserStruct *ps_UserStruct = (str_UserStruct *) pv_UserSharedMemory;

    switch (b_InterruptMask)
       {
       case 0x1:
            ps_UserStruct->b_IntHardCpt  = 0; ps_UserStruct->b_IntLatchReg = 1;
            ps_UserStruct->ul_CounterValue [0][0] = ul_CounterLatchValue;
            i_PA1710_KRNL_Write32BitCounterValue(ps_UserStruct->ui_BaseAddress, 0,
                                                 ul_CounterLatchValue + 1)
            break;

       case 0x2:
            ps_UserStruct->b_IntHardCpt  = 0; ps_UserStruct->b_IntLatchReg = 2;
            ps_UserStruct->ul_CounterValue [0][1] = ul_CounterLatchValue;
            i_PA1710_KRNL_Write32BitCounterValue(ps_UserStruct->ui_BaseAddress, 0,
                                                 ul_CounterLatchValue + 1)
            break;

       case 0x4:
            ps_UserStruct->b_IntHardCpt  = 1; ps_UserStruct->b_IntLatchReg = 1;
            ps_UserStruct->ul_CounterValue [1][0] = ul_CounterLatchValue;
            i_PA1710_KRNL_Write32BitCounterValue(ps_UserStruct->ui_BaseAddress, 1,
                                                 ul_CounterLatchValue + 1)
            break;

       case 0x8:
            ps_UserStruct->b_IntHardCpt  = 1; ps_UserStruct->b_IntLatchReg = 2;
            ps_UserStruct->ul_CounterValue [1][1] = ul_CounterLatchValue;
            i_PA1710_KRNL_Write32BitCounterValue(ps_UserStruct->ui_BaseAddress, 1,
                                                 ul_CounterLatchValue + 1)
            break;

       case 0x10:
            ps_UserStruct->b_IntHardCpt  = 2; ps_UserStruct->b_IntLatchReg = 1;
            ps_UserStruct->ul_CounterValue [2][0] = ul_CounterLatchValue;
            i_PA1710_KRNL_Write32BitCounterValue(ps_UserStruct->ui_BaseAddress, 2,
                                                 ul_CounterLatchValue + 1)
            break;

       case 0x20:
            ps_UserStruct->b_IntHardCpt  = 2; ps_UserStruct->b_IntLatchReg = 2;
            ps_UserStruct->ul_CounterValue [2][1] = ul_CounterLatchValue;
            i_PA1710_KRNL_Write32BitCounterValue(ps_UserStruct->ui_BaseAddress, 2,
                                                 ul_CounterLatchValue + 1)
            break;

       case 0x40:
            ps_UserStruct->ul_TimerInterruptCpt =
            ps_UserStruct->ul_TimerInterruptCpt + 1;
            i_PA1710_KRNL_Write32BitCounterValue(ps_UserStruct->ui_BaseAddress, 2,
                                                 0x5555);
            break;
       case 0x81:
            ps_UserStruct->b_IntHardCpt  = 0; ps_UserStruct->b_CBCounter   = 1;
            break;

       case 0x84:
            ps_UserStruct->b_IntHardCpt  = 1; ps_UserStruct->b_CBCounter   = 1;
            break;

       case 0x90:
            ps_UserStruct->b_IntHardCpt  = 2; ps_UserStruct->b_CBCounter   = 1;
            break;
       }
    ps_UserStruct->b_ReceiveInterrupt = 1;
    }
```

9.3 Counter initialisation

9.3.1 Initialisation 1 x 32-bit /Simple mode/ Hysteresis ON

a) Flow chart

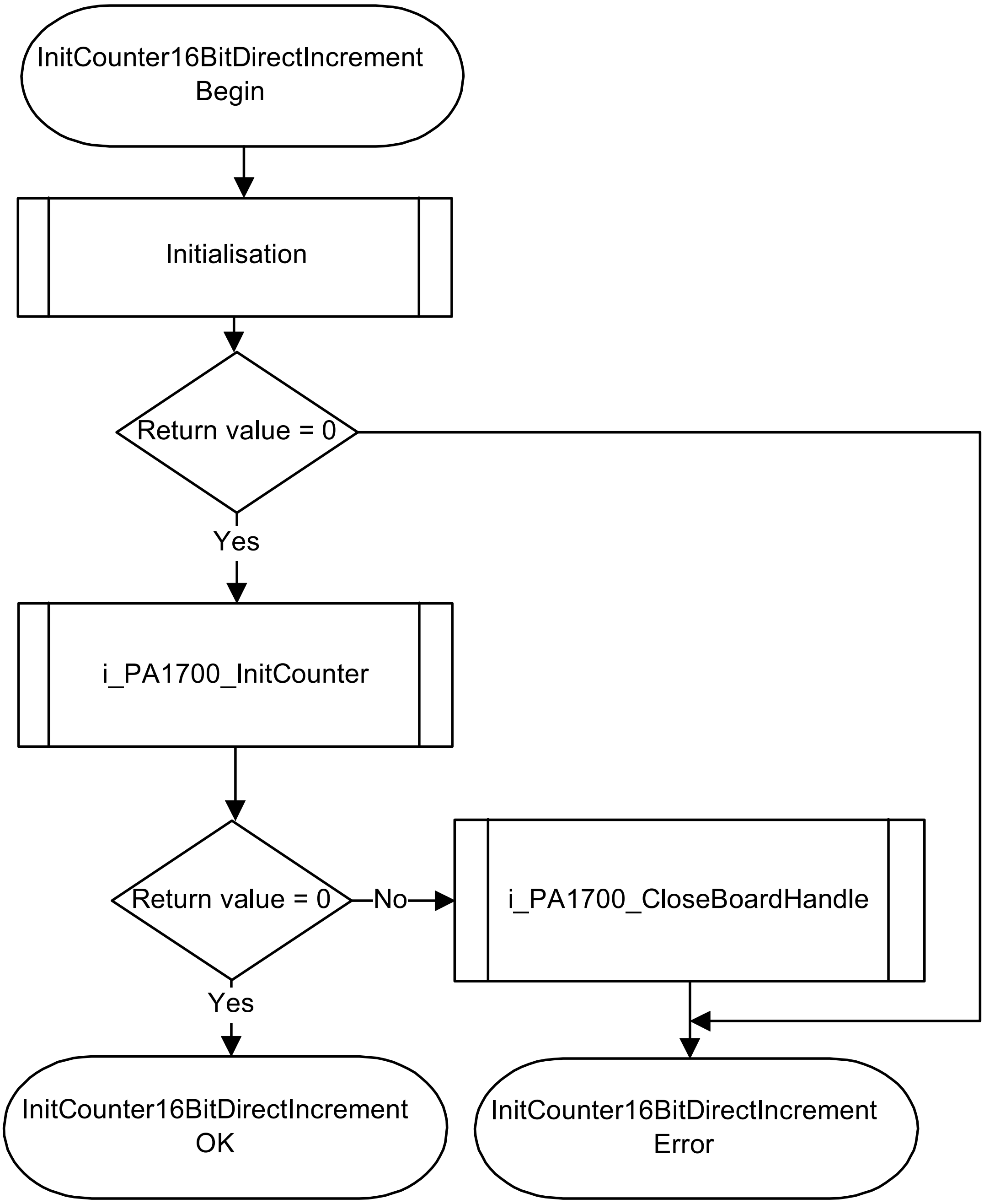

InitCounter32BitSimpleHysteresisOn Begin
Initialisation
Return value = 0
Yes
i_PA1700_InitCounter
Return value = 0
No
i_PA1700_CloseBoardHandle
Yes
InitCounter32BitSimpleHysteresisOn OK
InitCounter32BitSimpleHysteresisOn Error

b) Example in C

```
int  InitCounter32BitSimpleHysteresisOn (unsigned char * pb_BoardHandle)
     {
     if (Initialisation (pb_BoardHandle) == 0)
        {
        if (i_PA1700_InitCounter (*pb_BoardHandle,
                                   0,
                                   PA1700_32BIT_COUNTER,
                                   PA1700_SIMPLE_MODE,
                                   PA1700_HYSTERESIS_ON,
                                   0,
                                   0) == 0)
           {
           printf (“Initialisation OK“);
           return (0);
           }
        else
           {
           printf (“Initialisation error“);
      i_ReturnValue = i_PA1700_CloseBoardHandle (*pb_BoardHandle);
           return (-1);
           }
        }
     else
        {
        printf (“Initialisation error“);
        return (-1);
        }
     }
```

9.3.2 Initialisation 2 X 16-bit /direct mode/ increment

a) Flow chart


InitCounter16BitDirectIncrement
Begin
Initialisation
Return value = 0
Yes
i_PA1700_InitCounter
Return value = 0
No
i_PA1700_CloseBoardHandle
Yes
InitCounter16BitDirectIncrement
OK
InitCounter16BitDirectIncrement
Error

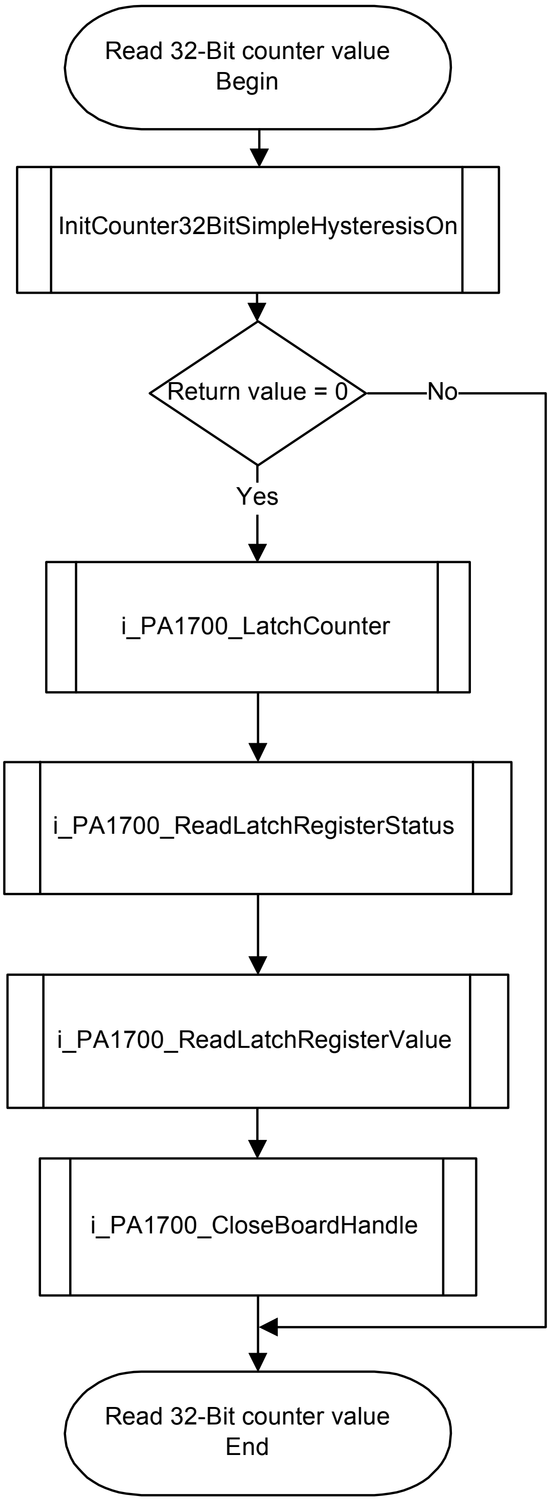
b) Example in C

```
int  InitCounter16BitDirectIncrement (unsigned char * pb_BoardHandle)
     {
     if (Initialisation (pb_BoardHandle) == 0)
        {
        if (i_PA1700_InitCounter (*pb_BoardHandle,
                                   0,
                                   PA1700_16BIT_COUNTER,
                                   PA1700_DIRECT_MODE,
                                   PA1700_INCREMENT,
                                   PA1700_DIRECT_MODE,
                                   PA1700_INCREMENT) == 0)
           {
           printf (“Initialisation OK“);
           return (0);
           }
        else
           {
           printf (“Initialisation error“);
           i_ReturnValue =  i_PA1700_CloseBoardHandle (*pb_BoardHandle);
           return (-1);
           }
        }
     else
        {
        printf (“Initialisation error“);
        return (-1);
        }
     }
```

9.4 Reading the counter

9.4.1 Reading the 32-bit counter

a) Flow chart

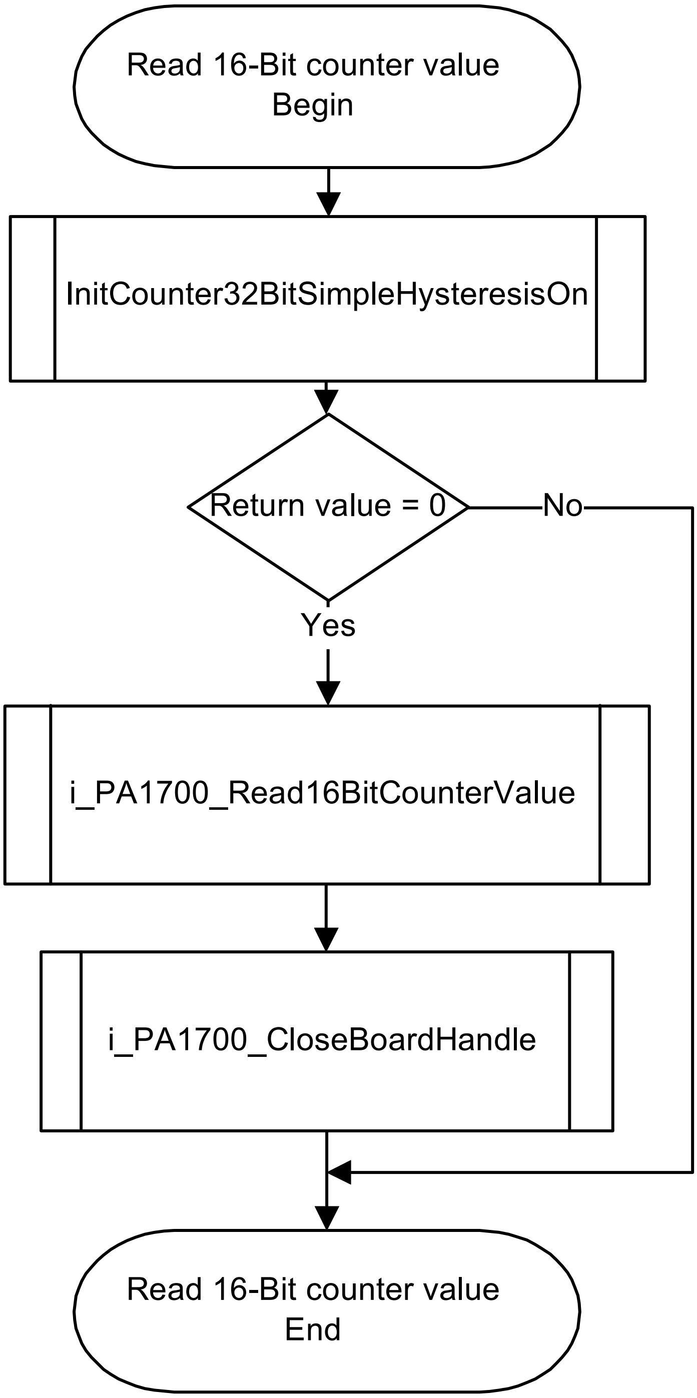

Read 32-Bit counter value
Begin
InitCounter32BitSimpleHysteresisOn
Return value = 0
No
Yes
i_PA1700_LatchCounter
i_PA1700_ReadLatchRegisterStatus
i_PA1700_ReadLatchRegisterValue
i_PA1700_CloseBoardHandle
Read 32-Bit counter value
End

b) Example in C

```
void   main (void)
   {
   int            i_ReturnValue;
   unsigned char  b_BoardHandle;
   unsigned char  b_LatchRegisterStatus;
   unsigned long  ul_LatchRegisterValue;


   if (InitCounter32BitSimpleHysteresisOn (&b_BoardHandle) == 0)
      {
      i_ReturnValue = i_PA1700_LatchCounter (b_BoardHandle, 0, 0);
      printf (“\nLatch counter return value = %d“, i_ReturnValue);

      i_ReturnValue = i_PA1700_ReadLatchRegisterStatus(b_BoardHandle,0,0,
&b_LatchRegisterStatus);
      printf (“\nRead latch register status return value = %d“, i_ReturnValue);
      printf (“\nLatch register status = %d“, b_LatchRegisterStatus);

      i_ReturnValue = i_PA1700_ReadLatchRegisterValue (b_BoardHandle,0,0,
&ul_LatchRegisterValue);
      printf (“\nRead latch register value return value = %d“, i_ReturnValue);
      printf (“\nLatch register value = %lu“, ul_LatchRegisterValue);

      i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
      printf (“\nClose board handle return value = %d“, i_ReturnValue);
      }
   else
      {
      printf (“Initialisation error“);
      }
   }
```

9.4.2 Reading the 16-bit counter

a) Flow chart


Read 16-Bit counter value
Begin
InitCounter32BitSimpleHysteresisOn
Return value = 0
No
Yes
i_PA1700_Read16BitCounterValue
i_PA1700_CloseBoardHandle
Read 16-Bit counter value
End

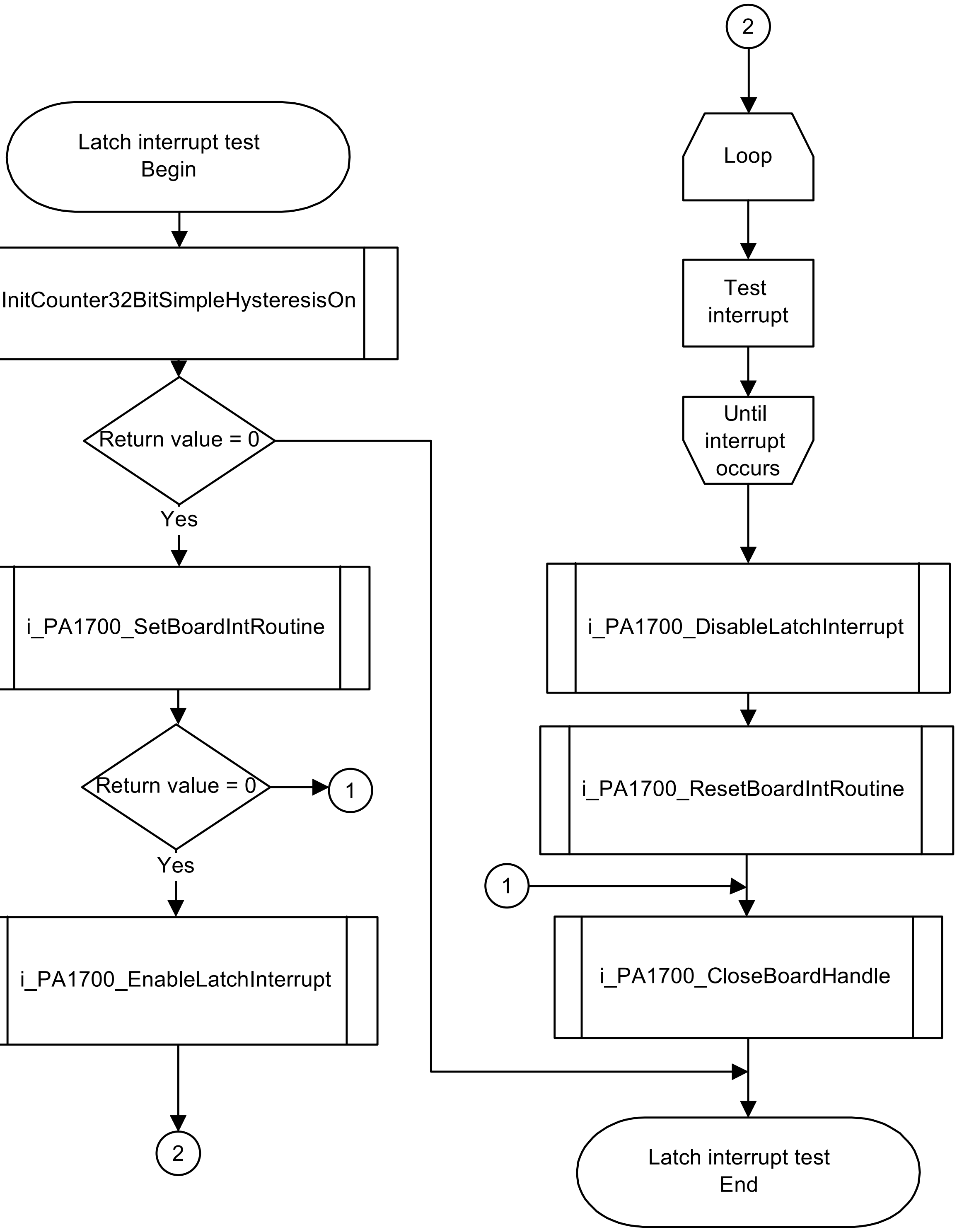
b) Example in C

```
void main (void)
     {
     int          i_ReturnValue;
     unsigned charb_BoardHandle;
     unsigned int ui_CounterValue;


     if (InitCounter32BitSimpleHysteresisOn (&b_BoardHandle) == 0)
        {
   i_ReturnValue = i_PA1700_Read16BitCounterValue (b_BoardHandle, 0, 0, &ui_CounterValue);
        printf (“\nRead 16-Bit counter return value = %d“, i_ReturnValue);
        printf (“\n16-Bit counter value = %u“, ui_CounterValue);

        i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
        printf (“\nClose board handle return value = %d“, i_ReturnValue);
        }
     else
        {
        printf (“Initialisation error“);
        }
     }
```

9.5 Latch interrupt

9.5.1 Clear / Write test

a) Flow chart


2
Loop
Latch interrupt test
Begin
Test
interrupt
InitCounter32BitSimpleHysteresisOn
Until
interrupt
occurs
Return value = 0
Yes
i_PA1700_SetBoardIntRoutine
i_PA1700_DisableLatchInterrupt
Return value = 0
1
i_PA1700_ResetBoardIntRoutine
Yes
1
i_PA1700_EnableLatchInterrupt
i_PA1700_CloseBoardHandle
2
Latch interrupt test
End

b) Example in C for DOS

```
void main (void)
     {
     int          i_ReturnValue;
     unsigned charb_BoardHandle;


     if (InitCounter32BitSimpleHysteresisOn (&b_BoardHandle) == 0)
        {
        if (i_PA1700_SetBoardIntRoutineDos (b_BoardHandle, v_InterruptRoutine) == 0)
           {
           b_ReceiveInterrupt = 0;
           i_ReturnValue = i_PA1700_EnableLatchInterrupt (b_BoardHandle, 0);
           printf (“\nEnable latch interrupt return value = %d“, i_ReturnValue);

           while (b_ReceiveInterrupt != 1);

           printf (“\nInterrupt occur. Counter value = %lu“,
                  ul_CounterValue [b_IntHardCpt] [b_IntLatchReg - 1]);

           i_ReturnValue = i_PA1700_DisableLatchInterrupt (b_BoardHandle, 0);
           printf (“\nDisable latch interrupt return value = %d“, i_ReturnValue);

           i_ReturnValue = i_PA1700_ResetBoardIntRoutine (b_BoardHandle);
           printf (“\nReset board interrupt routine return value = %d“, i_ReturnValue);

           i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
           printf (“\nClose board handle return value = %d“, i_ReturnValue);
           }
        else
           {
           printf (“\nInterrupt initialisation error“);

           i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
           printf (“\nClose board handle return value = %d“, i_ReturnValue);
           }
        }
     else
        {
        printf (“Initialisation error“);
        }
     }
```

c) Example in C for Windows 3.1X

```
void   main (void)
   {
   int            i_ReturnValue;
   unsigned char  b_BoardHandle;


   if (InitCounter32BitSimpleHysteresisOn (&b_BoardHandle) == 0)
      {
      if (i_PA1700_SetBoardIntRoutineWin16 (b_BoardHandle, v_InterruptRoutine) == 0)
         {
         b_ReceiveInterrupt = 0;
         i_ReturnValue = i_PA1700_EnableLatchInterrupt (b_BoardHandle, 0);
         printf (“\nEnable latch interrupt return value = %d“, i_ReturnValue);

         while (b_ReceiveInterrupt != 1);

         printf (“\nInterrupt occur. Counter value = %lu“,
                   ul_CounterValue [b_IntHardCpt] [b_IntLatchReg - 1]);

         i_ReturnValue = i_PA1700_DisableLatchInterrupt (b_BoardHandle, 0);
         printf (“\nDisable latch interrupt return value = %d“, i_ReturnValue);

         i_ReturnValue = i_PA1700_ResetBoardIntRoutine (b_BoardHandle);
         printf (“\nReset board interrupt routine return value = %d“, i_ReturnValue);

         i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
         printf (“\nClose board handle return value = %d“, i_ReturnValue);
         }
      else
         {
         printf (“\nInterrupt initialisation error“);

         i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
         printf (“\nClose board handle return value = %d“, i_ReturnValue);
         }
      }
   else
      {
      printf (“Initialisation error“);
      }
   }
```

d) Example in C for Windows NT and Windows 95/98 (asychronous mode)

```
void main (void)
     {
     int           i_ReturnValue;
     unsigned charb_BoardHandle;


     if (InitCounter32BitSimpleHysteresisOn (&b_BoardHandle) == 0)
        {
        if (i_PA1700_SetBoardIntRoutineWin32  (b_BoardHandle,
                                                PA1700_ASYNCHRONY_MODE,
                                                0
                                                NULL,
                                                v_InterruptRoutine) == 0)
           {
           b_ReceiveInterrupt = 0;
           i_ReturnValue = i_PA1700_EnableLatchInterrupt (b_BoardHandle, 0);
           printf (“\nEnable latch interrupt return value = %d“, i_ReturnValue);

           while (b_ReceiveInterrupt != 1);

           printf (“\nInterrupt occur. Counter value = %lu“,
                   ul_CounterValue [b_IntHardCpt] [b_IntLatchReg - 1]);

           i_ReturnValue = i_PA1700_DisableLatchInterrupt (b_BoardHandle, 0);
           printf (“\nDisable latch interrupt return value = %d“, i_ReturnValue);

           i_ReturnValue = i_PA1700_ResetBoardIntRoutine (b_BoardHandle);
           printf (“\nReset board interrupt routine return value = %d“, i_ReturnValue);

           i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
           printf (“\nClose board handle return value = %d“, i_ReturnValue);
           }
        else
           {
           printf (“\nInterrupt initialisation error“);

           i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
           printf (“\nClose board handle return value = %d“, i_ReturnValue);
           }
        }
     else
        {
        printf (“Initialisation error“);
        }
     }
```

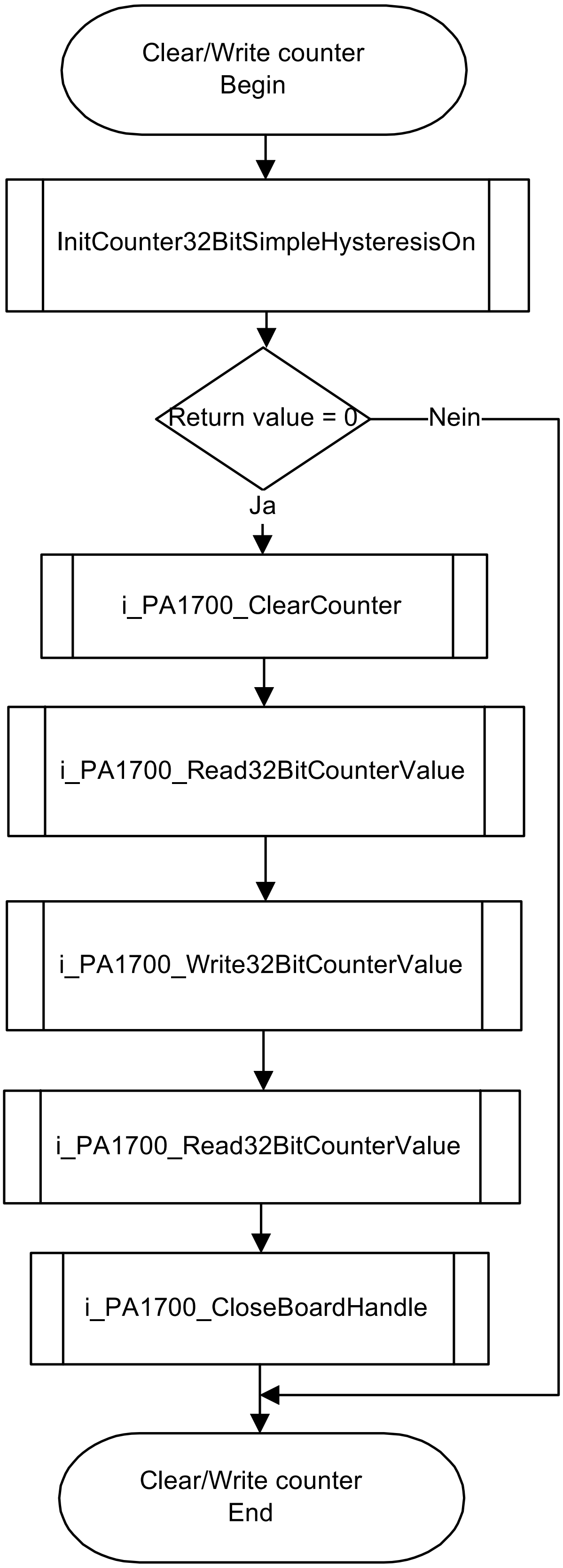
e) Example in C for Windows NT and Windows 95/98 (synchronous mode)

```
str_UserStruct * ps_GlobalUserStruct;

void   main (void)
   {
   int            i_ReturnValue;
   unsigned char  b_BoardHandle;


   if (InitCounter32BitSimpleHysteresisOn (&b_BoardHandle) == 0)
      {
      if (i_PA1700_SetBoardIntRoutineWin32    (b_BoardHandle,
                                               PCI1710_ASYNCHRONY_MODE,
                                               sizeof (str_UserStruct),
                                               (void **) & ps_GlobalUserStruct,
                                               v_InterruptRoutine) == 0)
         {
         ps_GlobalUserStruct->b_ReceiveInterrupt = 0;
         i_ReturnValue = i_PA1700_EnableLatchInterrupt (b_BoardHandle, 0);
         printf (“\nEnable latch interrupt return value = %d“, i_ReturnValue);

         while (ps_GlobalUserStruct->b_ReceiveInterrupt != 1);

         printf (“\nInterrupt occur. Counter value = %lu“,
         ps_GlobalUserStruct->ul_CounterValue [ps_GlobalUserStruct->b_IntHardCpt]
                                          [ps_GlobalUserStruct->b_IntLatchReg - 1]);

         i_ReturnValue = i_PA1700_DisableLatchInterrupt (b_BoardHandle, 0);
         printf (“\nDisable latch interrupt return value = %d“, i_ReturnValue);

         i_ReturnValue = i_PA1700_ResetBoardIntRoutine (b_BoardHandle);
         printf (“\nReset board interrupt routine return value = %d“, i_ReturnValue);

         i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
         printf (“\nClose board handle return value = %d“, i_ReturnValue);
         }
      else
         {
         printf (“\nInterrupt initialisation error“);

         i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
         printf (“\nClose board handle return value = %d“, i_ReturnValue);
         }
      }
   else
      {
      printf (“Initialisation error“);
      }
   }
```

9.6 Clear / Write

9.6.1 Clearing the counter

a) Flow chart


Clear/Write counter
Begin
InitCounter32BitSimpleHysteresisOn
Return value = 0
Nein
Ja
i_PA1700_ClearCounter
i_PA1700_Read32BitCounterValue
i_PA1700_Write32BitCounterValue
i_PA1700_Read32BitCounterValue
i_PA1700_CloseBoardHandle
Clear/Write counter
End

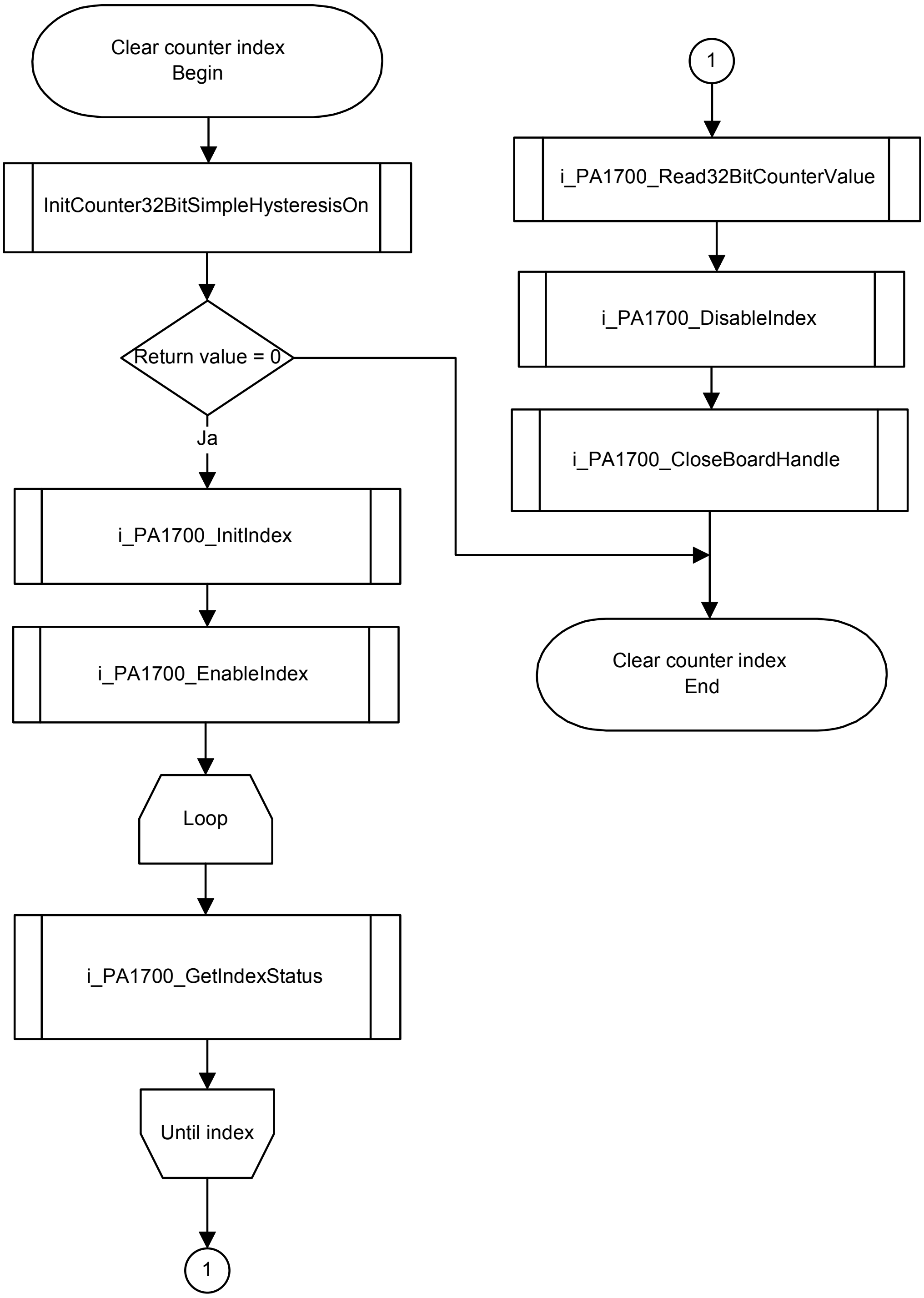
b) Example in C

```
void main (void)
     {
     int          i_ReturnValue;
     unsigned charb_BoardHandle;
     unsigned longul_CounterValue;


     if (InitCounter32BitSimpleHysteresisOn (&b_BoardHandle) == 0)
        {
        b_ReceiveInterrupt = 0;
        i_ReturnValue = i_PA1700_ClearCounter (b_BoardHandle, 0);
        printf (“\nClear counter return value = %d“, i_ReturnValue);

        i_ReturnValue = i_PA1700_Read32BitCounterValue (b_BoardHandle, 0, &ul_CounterValue);
        printf (“\nRead 32-bit counter return value = %d“, i_ReturnValue);
        printf (“\n32 bit counter value = %lu“, ul_CounterValue);

        i_ReturnValue = i_PA1700_Write32BitCounterValue (b_BoardHandle, 0, 200000);
        printf (“\nWrite 32-bit counter value return value = %d“, i_ReturnValue);

        i_ReturnValue = i_PA1700_Read32BitCounterValue (b_BoardHandle, 0, &ul_CounterValue);
        printf (“\nRead 32-bit counter return value = %d“, i_ReturnValue);
        printf (“\n32 bit counter value = %lu“, ul_CounterValue);

        i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
        printf (“\nClose board handle return value = %d“, i_ReturnValue);
        }
     else
        {
        printf (“Initialisation error“);
        }
     }
```

9.7 Index

9.7.1 Clearing the counter index

a) Flow chart



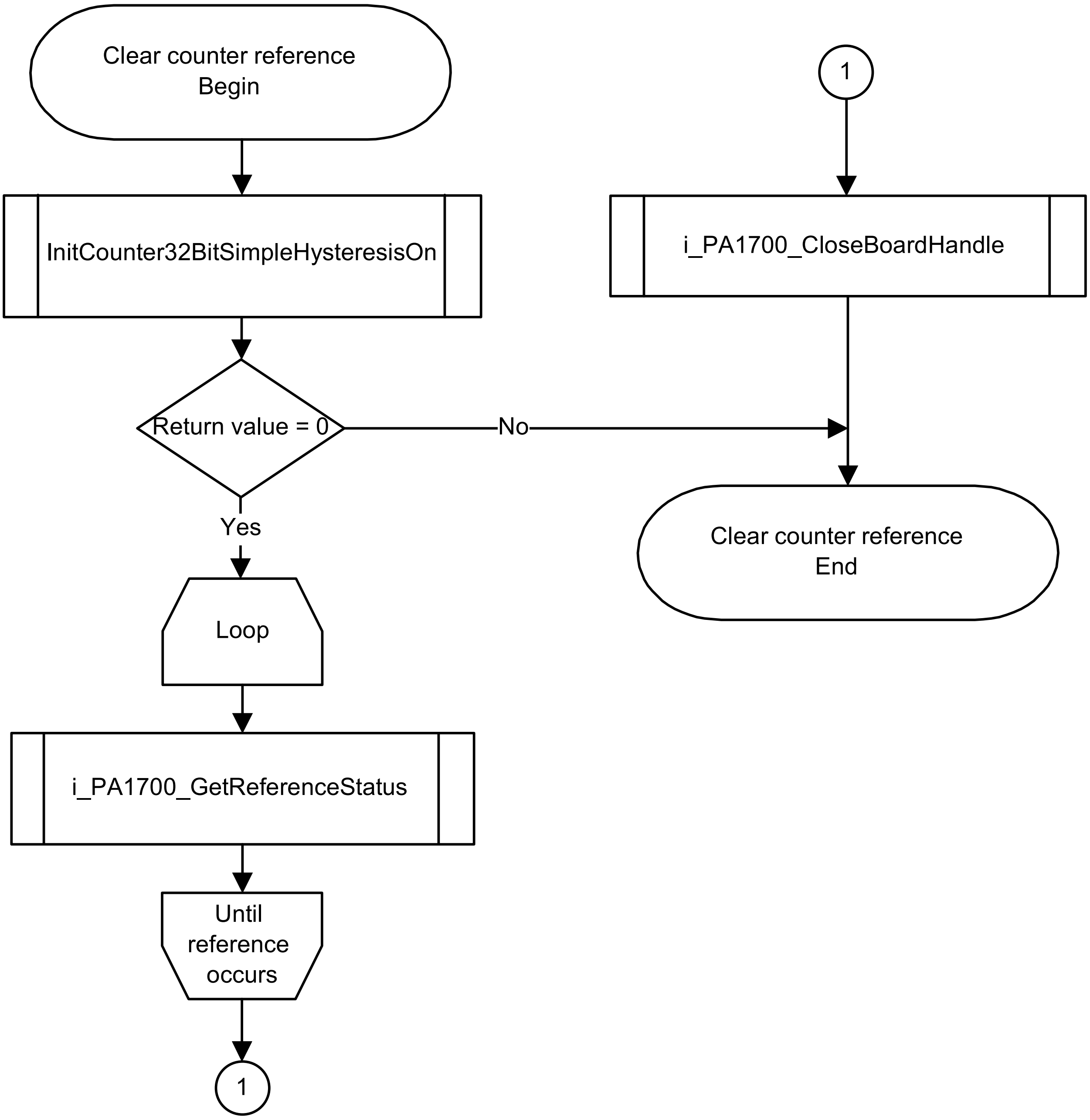
b) Example in C

```
void main (void)
     {
     int          i_ReturnValue;
     unsigned charb_BoardHandle;
     unsigned longul_CounterValue;
     unsigned charb_IndexStatus;


     if (InitCounter32BitSimpleHysteresisOn (&b_BoardHandle) == 0)
        {
        i_ReturnValue = i_PA1700_InitIndex    (b_BoardHandle,
                                               0,
                                               PA1700_ENABLE);
        printf (“\nIndex initialisation return value = %d“, i_ReturnValue);

        i_ReturnValue = i_PA1700_EnableIndex (b_BoardHandle, 0);
        printf (“\nEnable index return value = %d“, i_ReturnValue);

        do
           {
           i_ReturnValue = i_PA1700_GetIndexStatus (b_BoardHandle, 0, &b_IndexStatus);
           }
        while (b_IndexStatus != 1);

        i_ReturnValue = i_PA1700_Read32BitCounterValue (b_BoardHandle, 0, &ul_CounterValue);
        printf (“\nRead 32-bit counter return value = %d“, i_ReturnValue);
        printf (“\n32 bit counter value = %lu“, ul_CounterValue);

        i_ReturnValue = i_PA1700_DisableIndex (b_BoardHandle, 0);
        printf (“\nDisable index return value = %d“, i_ReturnValue);

        i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
        printf (“\nClose board handle return value = %d“, i_ReturnValue);
        }
     else
        {
        printf (“Initialisation error“);
        }
     }
```

9.8 Reference

9.8.1 Reference status

a) Flow chart



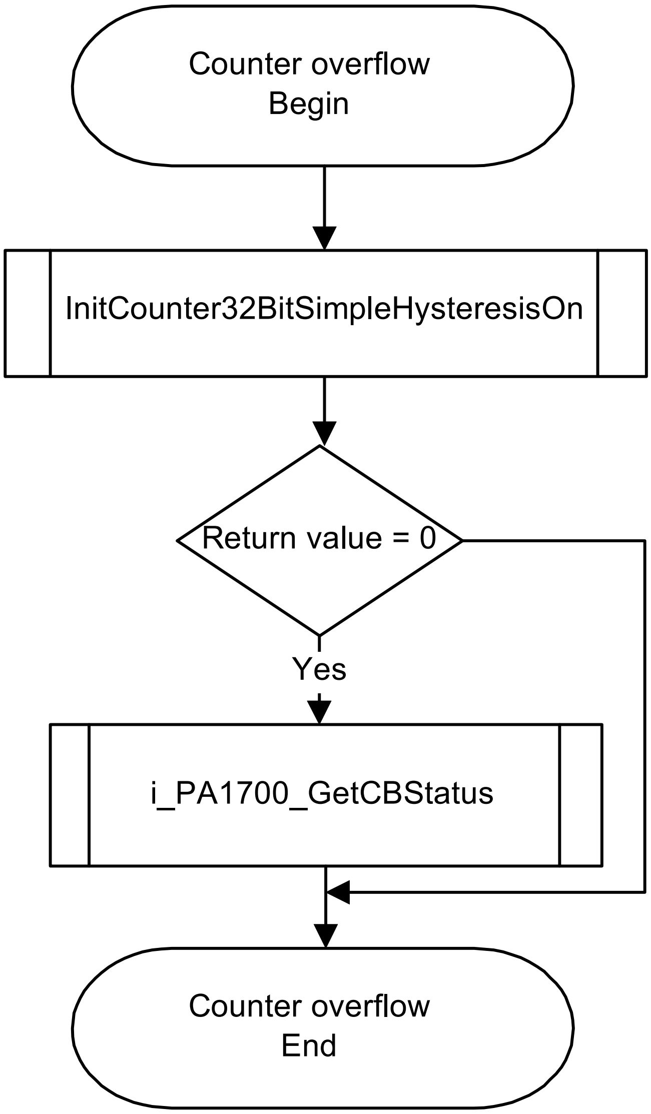
b) Example in C

```
void main (void)
     {
     int           i_ReturnValue;
     unsigned charb_BoardHandle;
     unsigned longul_LatchValue;
     unsigned char  b_ReferenceStatus;

     if (InitCounter32BitSimpleHysteresisOn (&b_BoardHandle) == 0)
        {
        do
           {
           i_ReturnValue = i_PA1700_GetReferenceStatus (b_BoardHandle, 0,
&b_ReferenceStatus);
           }
        while (b_ReferenceStatus != 1);

        i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
        printf (“\nClose board handle return value = %d“, i_ReturnValue);
        }
     else
        {
        printf (“Initialisation error“);
        }
     }
```

9.9 CB, U/D#

9.9.1 Counter overflow

a) Flow chart

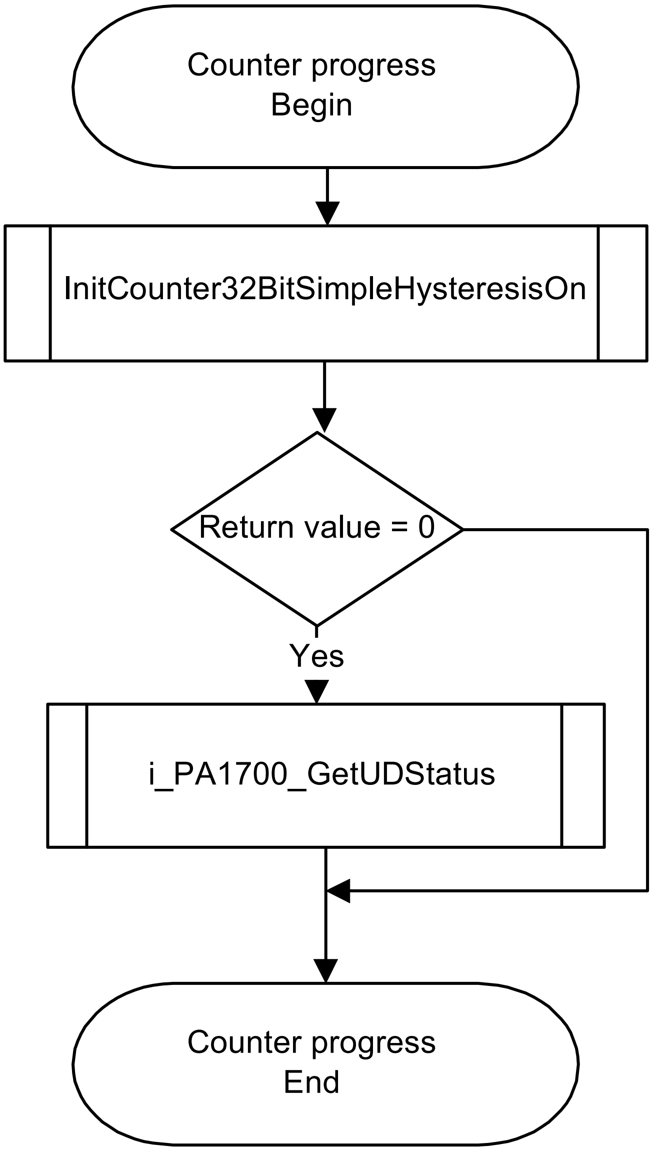


b) Example in C

```
void main (void)
     {
     int          i_ReturnValue;
     unsigned charb_BoardHandle;
     unsigned char  b_CBStatus;

     if (InitCounter32BitSimpleHysteresisOn (&b_BoardHandle) == 0)
        {
        i_ReturnValue = i_PA1700_GetCBStatus (b_BoardHandle, 0, &b_CBStatus);
        printf (“\nGet CB status return value = %d“, i_ReturnValue);

        if (b_CBStatus == 1)
           {
           printf (“\nCounter overflow“);
           }
        else
           {
           printf (“\nCounter no overflow“);
           }

        i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
        printf (“\nClose board handle return value = %d“, i_ReturnValue);
        }
     else
        {
        printf (“Initialisation error“);
        }
     }
```

9.9.2 Counter progress

a) Flow chart



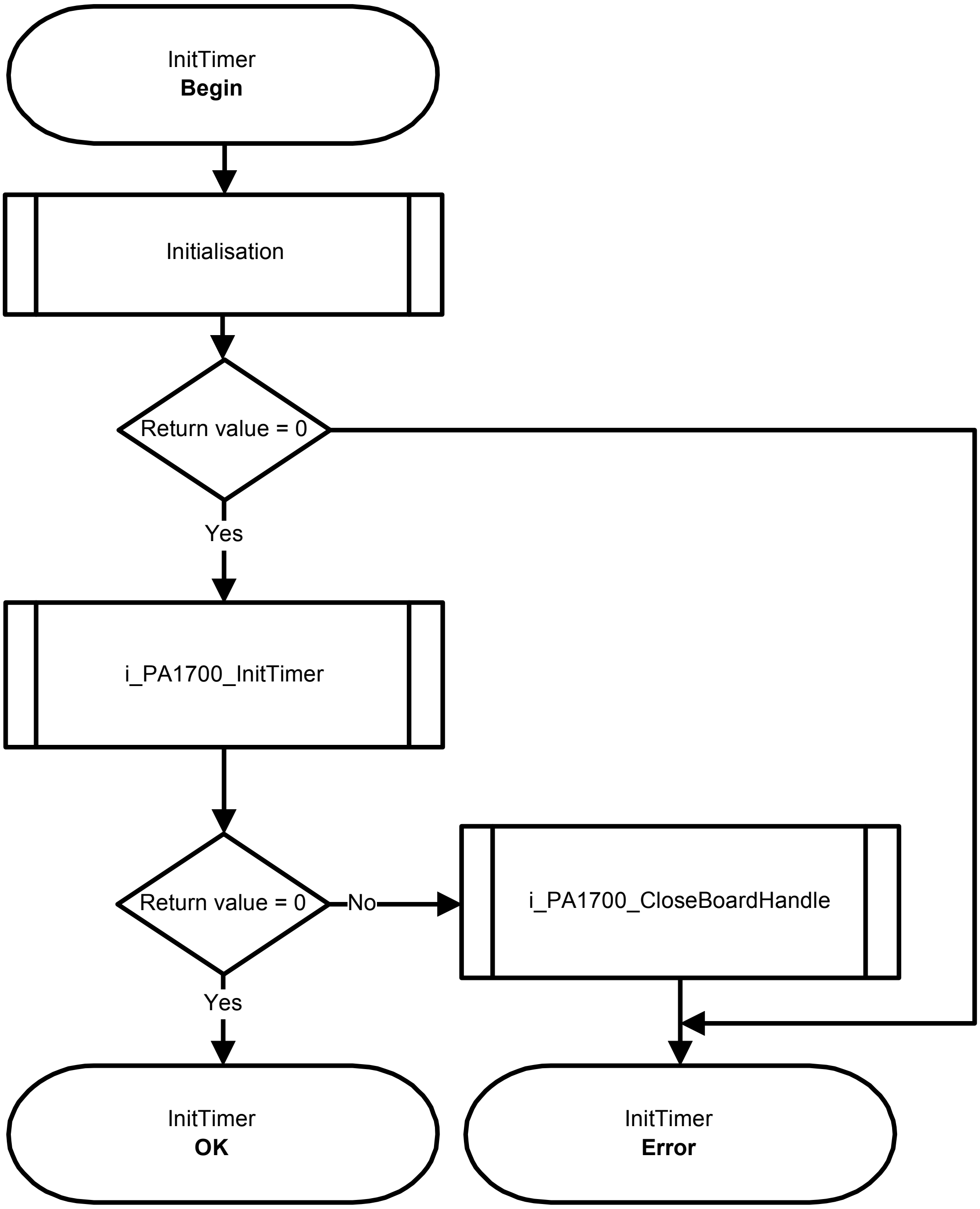
b) Example in C

```
void main (void)
     {
     int          i_ReturnValue;
     unsigned charb_BoardHandle;
     unsigned char  b_UDStatus;

     if (InitCounter32BitSimpleHysteresisOn (&b_BoardHandle) == 0)
        {
        i_ReturnValue = i_PA1700_GetUDStatus (b_BoardHandle, 0, &b_CBStatus);
        printf (“\nGet CB status return value = %d“, i_ReturnValue);

        if (b_CBStatus == 1)
           {
           printf (“\nCounter progress in the selected mode down“);
           }
        else
           {
           printf (“\nCounter progress in the selected mode up“);
           }

        i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
        printf (“\nClose board handle return value = %d“, i_ReturnValue);
        }
     else
        {
        printf (“Initialisation error“);
        }
     }
```

9.10 Timer

9.10.1 Timer initialisation

a) Flow chart

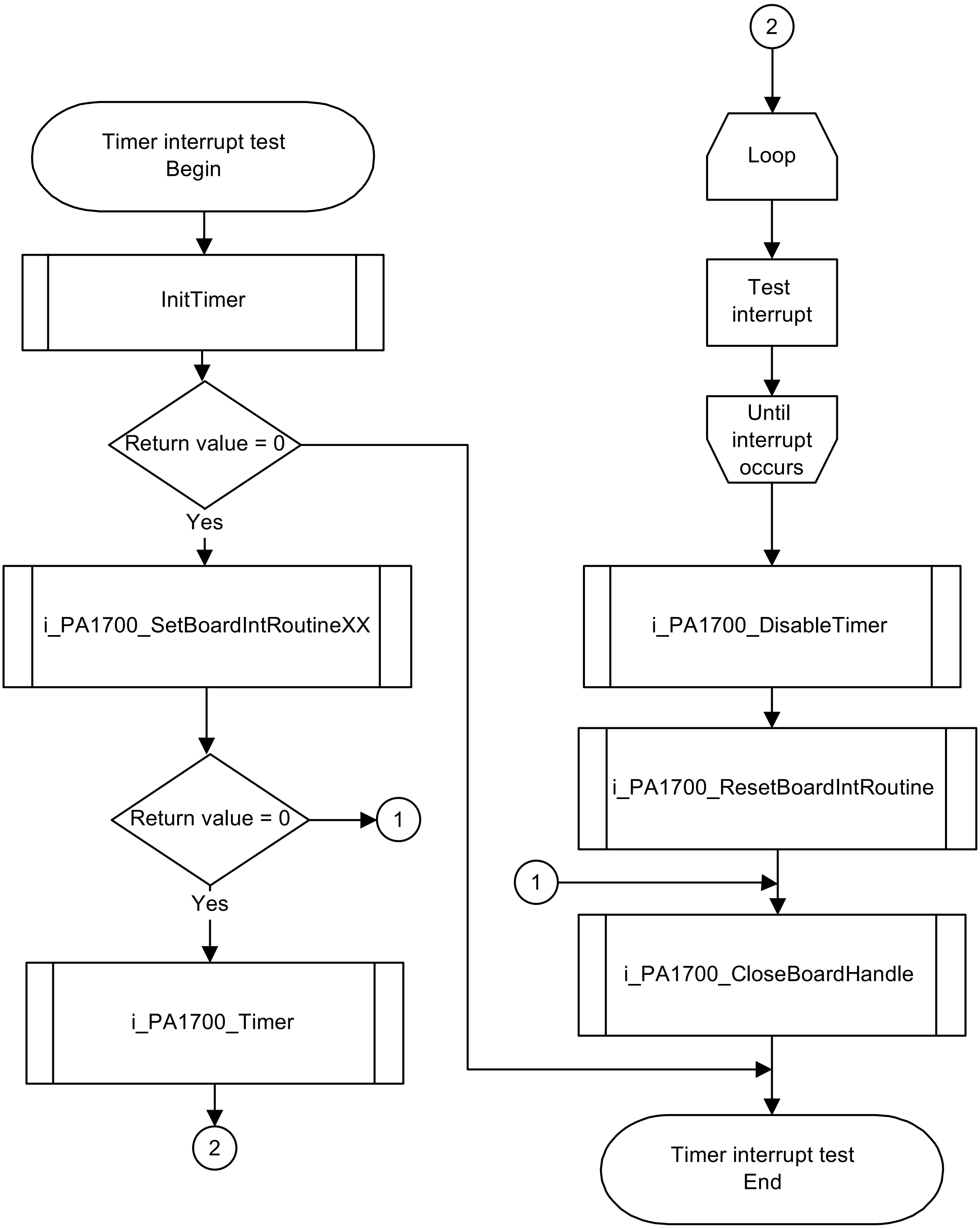


b) Example in C

```
int  InitTimer (unsigned char * pb_BoardHandle)
     {
     if (Initialisation (pb_BoardHandle) == 0)
        {
        if (i_PA1700_InitTimer    (*pb_BoardHandle,
                                   2,
                                   2,
                                   0xFF00) == 0)
           {
           printf (“Initialisation OK“);
           return (0);
           }
        else
           {
           printf (“Initialisation error“);
           i_ReturnValue =  i_PA1700_CloseBoardHandle (*pb_BoardHandle);
           return (-1);
           }
        }
     else
        {
        printf (“Initialisation error“);
        return (-1);
        }
     }
```

9.10.2 Timer interrupt

a) Flow chart



b) Example in C for DOS

```
void main (void)
     {
     int          i_ReturnValue;
     unsigned char  b_BoardHandle;


     if (InitTimer (&b_BoardHandle) == 0)
        {
        if (i_PA1700_SetBoardIntRoutineDos (b_BoardHandle, v_InterruptRoutine)==0)
           {
           b_ReceiveInterrupt = 0;
           i_ReturnValue = i_PA1700_EnableTimer(b_BoardHandle, 2,PA1700_ENABLE);
           printf (“\nEnable timer return value = %d“, i_ReturnValue);

           while (b_ReceiveInterrupt != 1);

           printf (“\nInterrupt occur. Timer %d“, 2, b_IntModule);

           i_ReturnValue = i_PA1700_DisableTimer (b_BoardHandle, 2);
           printf (“\nDisable timer return value = %d“, i_ReturnValue);

           i_ReturnValue = i_PA1700_ResetBoardIntRoutine (b_BoardHandle);
           printf (“\nReset board interrupt routine return value = %d“, i_ReturnValue);

           i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
           printf (“\nClose board handle return value = %d“, i_ReturnValue);
           }
        else
           {
           printf (“\nInterrupt initialisation error“);

           i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
           printf (“\nClose board handle return value = %d“, i_ReturnValue);
           }
        }
     else
        {
        printf (“Initialisation error“);
        }
     }
```

c) Example in C for Windows 3.11

```
void main (void)
     {
     int         i_ReturnValue;
     unsigned char  b_BoardHandle;


     if (InitTimer (&b_BoardHandle) == 0)
        {
        if (i_PA1700_SetBoardIntRoutineWin16(b_BoardHandle, v_InterruptRoutine)==0)
           {
           b_ReceiveInterrupt = 0;
           i_ReturnValue = i_PA1700_EnableTimer (b_BoardHandle, 2, PA1700_ENABLE);
           printf (“\nEnable timer return value = %d“, i_ReturnValue);

           while (b_ReceiveInterrupt != 1);

           printf (“\nInterrupt occur. Timer %d“, 2);

           i_ReturnValue = i_PA1700_DisableTimer (b_BoardHandle, 2);
           printf (“\nDisable timer return value = %d“, i_ReturnValue);

           i_ReturnValue = i_PA1700_ResetBoardIntRoutine (b_BoardHandle);
           printf (“\nReset board interrupt routine return value = %d“, i_ReturnValue);

           i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
           printf (“\nClose board handle return value = %d“, i_ReturnValue);
           }
        else
           {
           printf (“\nInterrupt initialisation error“);

           i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
           printf (“\nClose board handle return value = %d“, i_ReturnValue);
           }
        }
     else
        {
        printf (“Initialisation error“);
        }
     }
```

d) Example in C for Windows NT/95/98 (asynchronous mode)

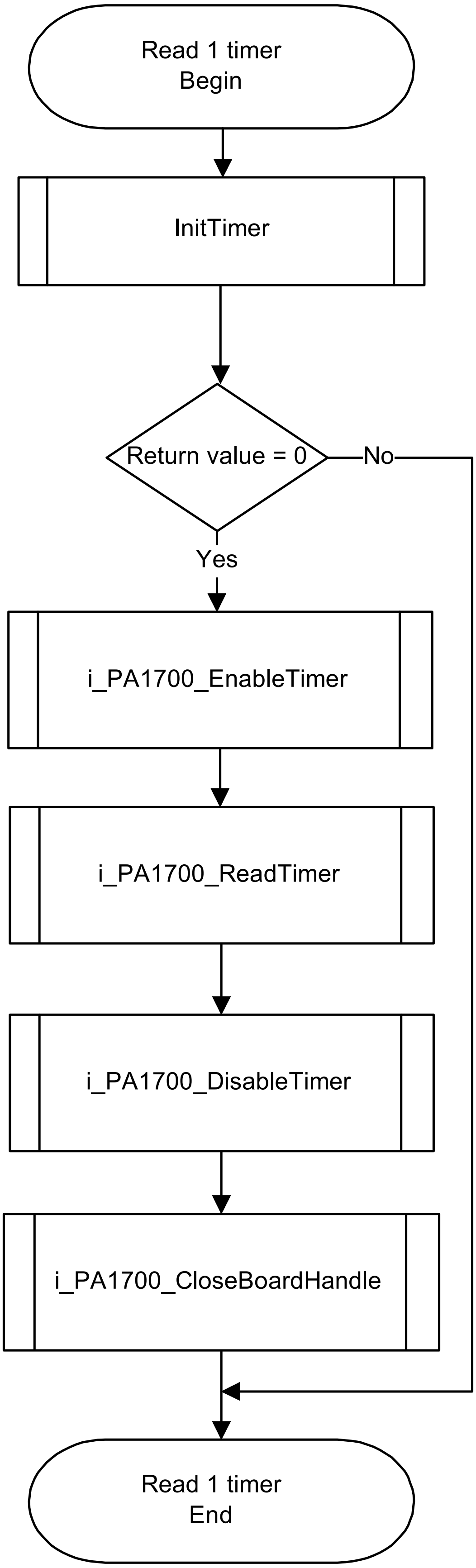
```
Void            main (void)
    {
    int         i_ReturnValue;
    unsigned char  b_BoardHandle;

    if (InitTimer (&b_BoardHandle) == 0)
       {
       if (i_PA1700_SetBoardIntRoutineWin32   (b_BoardHandle,
                                               PA1700_ASYNCHRONOUS_MODE,
                                               0
                                               NULL,
                                               v_InterruptRoutine) == 0)
          {
          b_ReceiveInterrupt = 0;
          i_ReturnValue = i_PA1700_EnableTimer (b_BoardHandle, 2 PA1700_ENABLE);
          printf (“\nEnable timer return value = %d“, i_ReturnValue);

          while (b_ReceiveInterrupt != 1);

          printf (“\nInterrupt occur. Timer %d“, 2);

          i_ReturnValue = i_PA1700_DisableTimer (b_BoardHandle, 2);
          printf (“\nDisable timer return value = %d“, i_ReturnValue);

          i_ReturnValue = i_PA1700_ResetBoardIntRoutine (b_BoardHandle);
          printf (“\nReset board interrupt routine return value = %d“, i_ReturnValue);

          i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
          printf (“\nClose board handle return value = %d“, i_ReturnValue);
          }
       else
          {
          printf (“\nInterrupt initialisation error“);

          i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
          printf (“\nClose board handle return value = %d“, i_ReturnValue);
          }
       }
    else
       {
       printf (“Initialisation error“);
       }
    }
```

e) Example in C for Windows NT/95/98 (synchronous mode)

```
str_UserStruct * ps_GlobalUserStruct;

void main (void)
     {
     int         i_ReturnValue;                      unsigned char  b_BoardHandle;
     unsigned char  b_TimerInterrupt;                unsigned char  b_StrobeInterrupt;

     if (InitTimer (&b_BoardHandle) == 0)
        {
        if (i_PA1700_SetBoardIntRoutineWin32   (b_BoardHandle,
                                               PCI1710_SYNCHRONOUS_MODE,
                                               sizeof (str_UserStruct),
                                               (void **) & ps_GlobalUserStruct,
                                               v_InterruptRoutine) == 0)
           {
           ps_GlobalUserStruct->b_ReceiveInterrupt = 0;
           i_ReturnValue = i_PA1700_GetHardwareInformation (b_BoardHandle,
                                                              &ps_GlobalUserStruct->
                                                              ui_BaseAddress,
                                                              &b_StrobeInterrupt,
                                                              &b_TimerInterrupt);
           printf (“\nGet hardware information return value = %d“, i_ReturnValue);

           i_ReturnValue = i_PA1700_EnableTimer (b_BoardHandle, 2, PA1700_ENABLE);
           printf (“\nEnable timer return value = %d“, i_ReturnValue);

           while (ps_GlobalUserStruct->b_ReceiveInterrupt != 1);

           printf (“\nInterrupt occur. Timer %d from module %d“,2);

           i_ReturnValue = i_PA1700_DisableTimer (b_BoardHandle, 2);
           printf (“\nDisable timer return value = %d“, i_ReturnValue);

           i_ReturnValue = i_PA1700_ResetBoardIntRoutine (b_BoardHandle);
           printf (“\nReset board interrupt routine return value = %d“, i_ReturnValue);

           i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
           printf (“\nClose board handle return value = %d“, i_ReturnValue);
           }
        else
           {
           printf (“\nInterrupt initialisation error“);
           i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
           printf (“\nClose board handle return value = %d“, i_ReturnValue);
           }
        }
     else
        {
        printf (“Initialisation error“);
        }
     }
```

9.10.3 Reading the timer

a) Flow chart

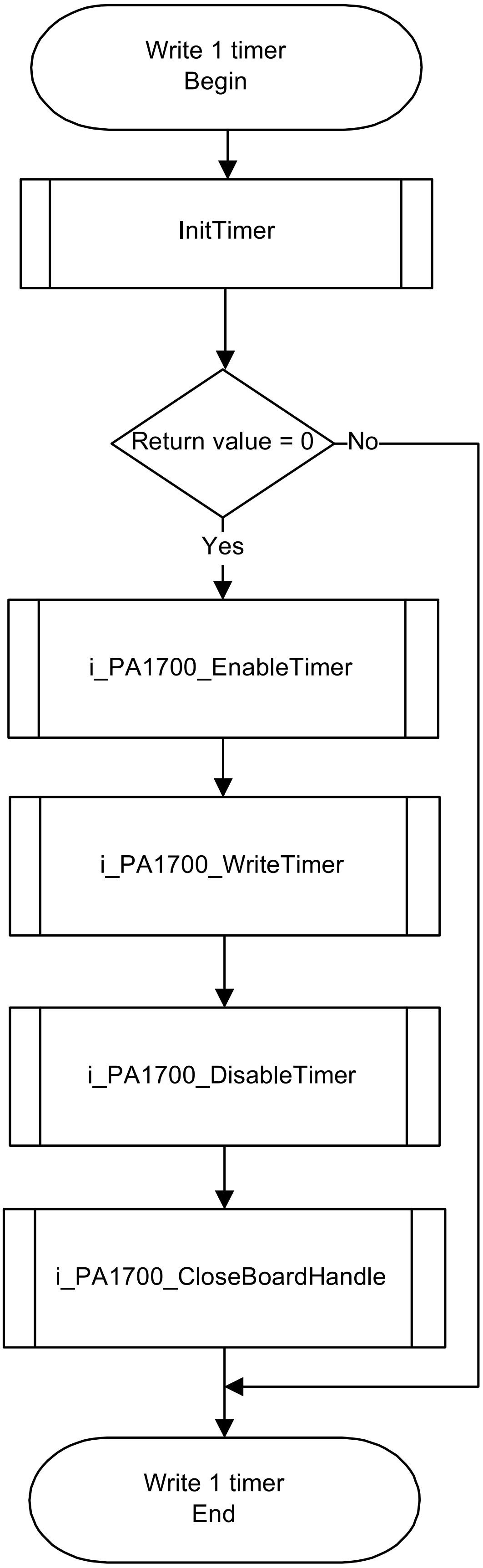


b) Example in C

```
void main (void)
     {
     int            i_ReturnValue;
     unsigned char  b_BoardHandle;
     unsigned int ui_TimerValue;

     if (InitTimer (&b_BoardHandle) == 0)
        {
        i_ReturnValue = i_PA1700_EnableTimer (b_BoardHandle, 2, PA1700_DISABLE);
        printf (“\nEnable timer return value = %d“, i_ReturnValue);

        i_ReturnValue = i_PA1700_ReadTimerValue (b_BoardHandle, 2, &ui_TimerValue);
        printf (“\nRead timer return value = %d“, i_ReturnValue);
        printf (“\nTimer value = %u“, ui_TimerValue);

        i_ReturnValue = i_PA1700_DisableTimer (b_BoardHandle, 2);
        printf (“\nDisable timer return value = %d“, i_ReturnValue);

        i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
        printf (“\nClose board handle return value = %d“, i_ReturnValue);
        }
     else
        {
        printf (“Initialisation error“);
        }
     }
```

9.10.4 Writing in the timer

a) Flow chart



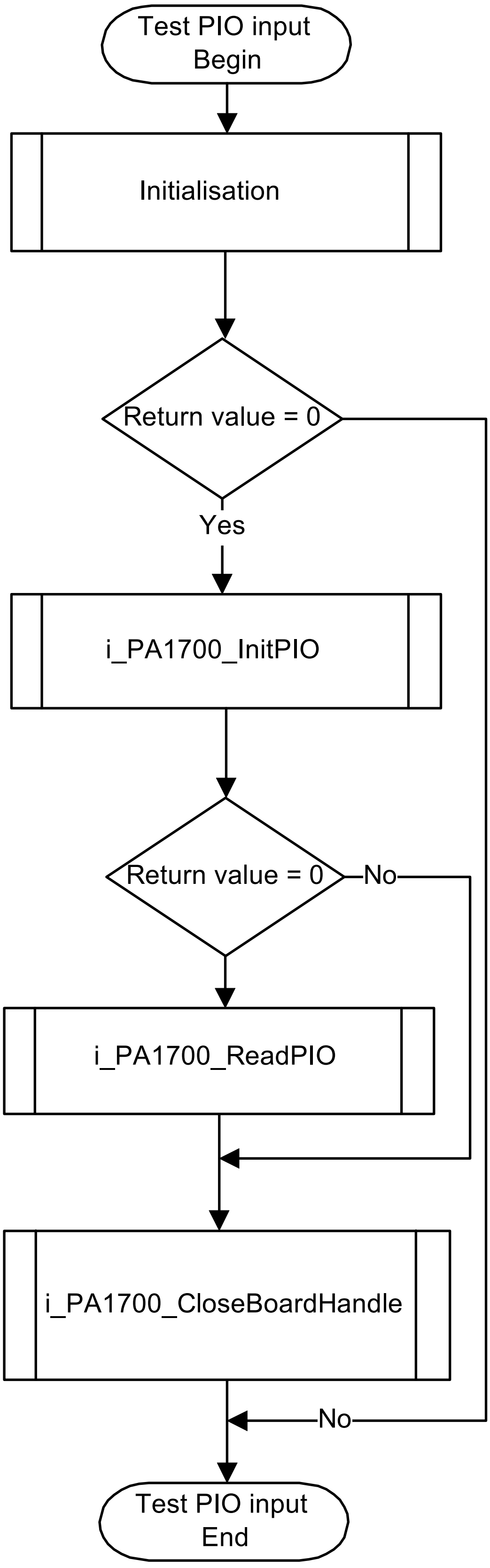
b) Example in C

```
Void            main (void)
    {
    int         i_ReturnValue;
    unsigned char    b_BoardHandle;

    if (InitTimer (&b_BoardHandle) == 0)
       {
       i_ReturnValue = i_PA1700_EnableTimer (b_BoardHandle, 2, PA1700_DISABLE);
       printf (“\nEnable timer return value = %d“, i_ReturnValue);

       i_ReturnValue = i_PA1700_WriteTimerValue (b_BoardHandle, 2, 0xFF55);
       printf (“\nWrite timer return value = %d“, i_ReturnValue);

       i_ReturnValue = i_PA1700_DisableTimer (b_BoardHandle, 0);
       printf (“\nDisable timer return value = %d“, i_ReturnValue);

       i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
       printf (“\nClose board handle return value = %d“, i_ReturnValue);
       }
    else
       {
       printf (“Initialisation error“);
       }
    }
```

9.11 PIO

9.11.1 PIO input

a) Flow chart

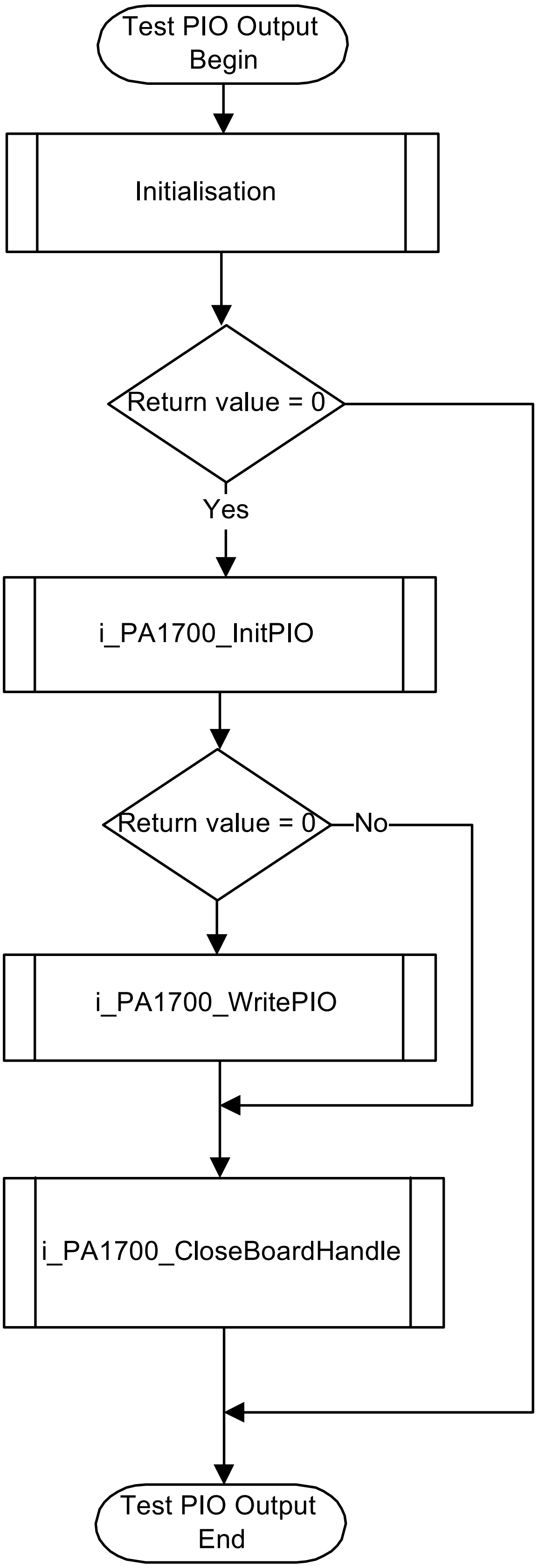


b) Example in C

```
Void   main (void)
       {
       unsigned char b_BoardHandle;
       unsigned char b_ReadValue;

       if (Initialisation (&b_BoardHandle) == 0)
          {
          if (i_PA1700_InitPIO (b_BoardHandle, 1, 1, 1, 1) == 0)
           {
             if (i_PA1700_ReadPio (b_BoardHandle, 0, &b_ReadValue) == 0)
                {
                printf (“Port A = %X Hex“, b_ReadValue);
                }
             else
                {
                printf (“Read PIO error“);
                }
             }
          else
             {
             printf (“Init PIO error“);
             }

          i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
          }
       else
          printf (“Initialisation error“);
       }
```

9.11.2 PIO output

a) Flow chart



b) Example in C

```
void main (void)
     {
     unsigned char b_BoardHandle;

     if (Initialisation (&b_BoardHandle) == 0)
        {
        if (i_PA1700_InitPIO (b_BoardHandle, 1, 1, 1, 1) == 0)
           {
           if (i_PA1700_WritePio (b_BoardHandle, 0, 0x55) == 0)
              {
              printf (“Write port A = 55 Hex“);
              }
           else
              {
              printf (“Read PIO error“);
              }
           }
        else
           {
           printf (“Init PIO error“);
           }

        i_ReturnValue =  i_PA1700_CloseBoardHandle (b_BoardHandle);
        }
     else
        printf (“Initialisation error“);
     }
```

INDEX