ADDI-DATA®

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

Attention!
Product discontinuation due to EC RoHS directive
More info: www.addi-data.com

CE

Technical documentation

ADDICOM PA 7200

CAN bus and serial interface

3rd edition 12/1997

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 nonfunctioning safety equipment
- nonobservance 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 software ADDISET and ADDIMON. 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.

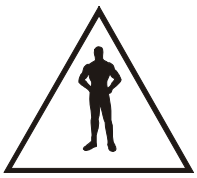
Trademarks

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

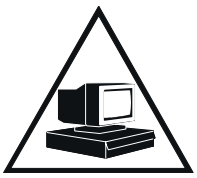
The original of this manual is in German. You can obtain it on request.

WARNING

In case of wrong uses and if the board is not used for the purpose it is intended:



people may be injured



the board, PC and peripheral may be destroyed

the environment may be polluted

Protect yourself, the others and the environment !

- **Read carefully the safety leaflet (yellow)!**
  If this leaflet is not with the documentation , please contact us and ask for it.

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

- **Used symbols**

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

IMPORTANT!
designates hints and other useful information.

- **Any question?**

  Our technical support is at your disposal

Figures

Tables

1 INTENDED PURPOSE OF THE BOARD

The board **PA 7200** is the interface between the processing electronics of the PC and an industrial process. The CAN bus interface is particularly appropriate for the connection with a CAN bus network.

The board is to be used in a free PC 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 CE mark.

The 9-pole SUB-D pin connector of the board **PA 7200** allows the exchange of CAN bus data with external communications devices.
The 25-pole SUB-D pin connector of the board **PA 7200** allows the exchange of serial data in a predetermined interface standard (RS422, RS485 or 20mA Current Loop) with external communications devices.

The connection with a shielded cable should comply with the following specifications:
- metallized plastic cap
- shielded cable
- cable shield folded back and firmly screwed to the connector housing.

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.

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

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.

Using the board in the intended purpose also includes respecting all instructions contained in this manual.

2 USER

2.1 Qualification

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

- installation,
- putting into operation,
- use,
- maintenance.

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

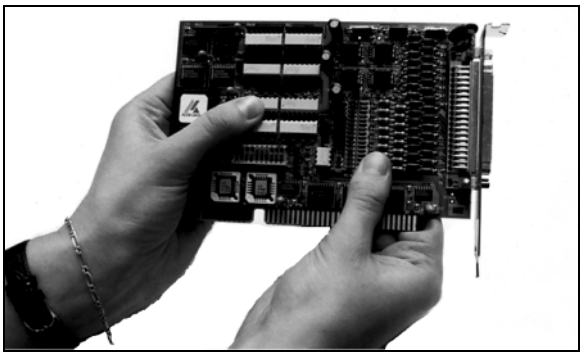
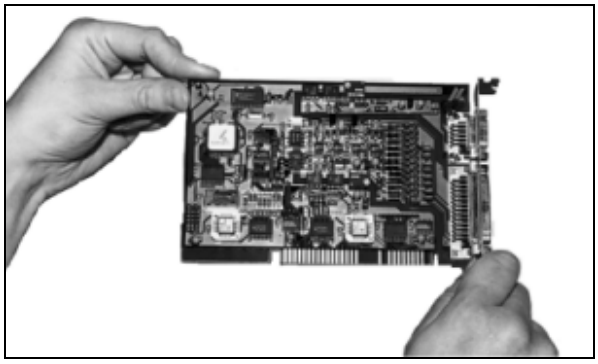


Fig. 3-2: Correct handling



4 TECHNICAL DATA

4.1 Electromagnetic compatibility (EMC)

The board has been subjected to EMC tests (89/336/EWG) in an accredited laboratory. The board complies as follows with the limit values set by the norms EN50082-2, EN55011, EN55022:

	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 4-layer printed circuit card.

Approximate card dimensions with components	Lenght:	172 mm
	Max. height:	127 mm (XT)
	Breadth:	19.2 mm
Weight:	140 g	
Installation in:	XT / AT slot	
Connection to the peripheral		
• for serial interface:	25-pole SUB-D pin connector	
• for CAN bus interface:	9-pole SUB-D pin connector	

4.3 Options

For the serial interface

Option U: FIFO memory for transmitted or received data of 16 bytes for the serial interface.
Option S: A DIP switches for setting the base address of the serial interface.

Option RC: Modem Control signals RTS, CTS as RS422 signals

4.4 Versions

The board **PA 7200** is available in the following versions:

- **PA 7200**
  - 1-port serial interface
  - 1 x RS 422, 1 x RS 485 with galvanic separation
  - 1 x 20 mA Current Loop, active, passive, with galvanic separation
  - 1 CAN bus interface with galvanic separation
- **PA 7200CAN**
  - 1 CAN bus interface with galvanic separation

4.5 Limit values

Operating temperature: ..... 0 to 60°C
Storage temperature: ..... -25 to 70°C
Relative humidity: ..... 30% to 99% noncondensing

Energy requirements:
- Operating voltage of the PC: ..... 5V ± 5%
  12V ± 5%
- Current consumption in mA
  (without load): typ. ..... See table ± 10%

	PA 7200	PA 7200 CAN
+ 5 V vom PC	130 ± 10mA	50 ± 10mA
+ 12 V vom PC	90 ± 10mA	30 ± 5mA

Serial interface:
Transfer rate:

- RS 422/RS 485: ..... max. 112 kBaud (at 1,8432 MHz quarz oscillator frequency)
- 20 mA Current Loop: ..... max 19,2 kBaud

Load resistance:
20mA Current Loop:..... ≤ 300 Ω

Overvoltage protection:

• RS 422/RS 485:	breakdown voltage= ± 6,5V VCL[1)] = ± 11,3V; at Ipp[2)] = 35,4A in the 1ms test Ppp[3)] = SURGE 300W/1ms All the lines are protected against short-circuit through PTC resistors.
• 20 mA Current Loop:	breakdown voltage = ± 26V VCL = ± 41,5V; at Ipp = 9,6A in the 1ms test Ppp = SURGE 400W/1ms
Insulation voltage:	1kV (RS 422/RS 485 and 20 mA Current Loop)

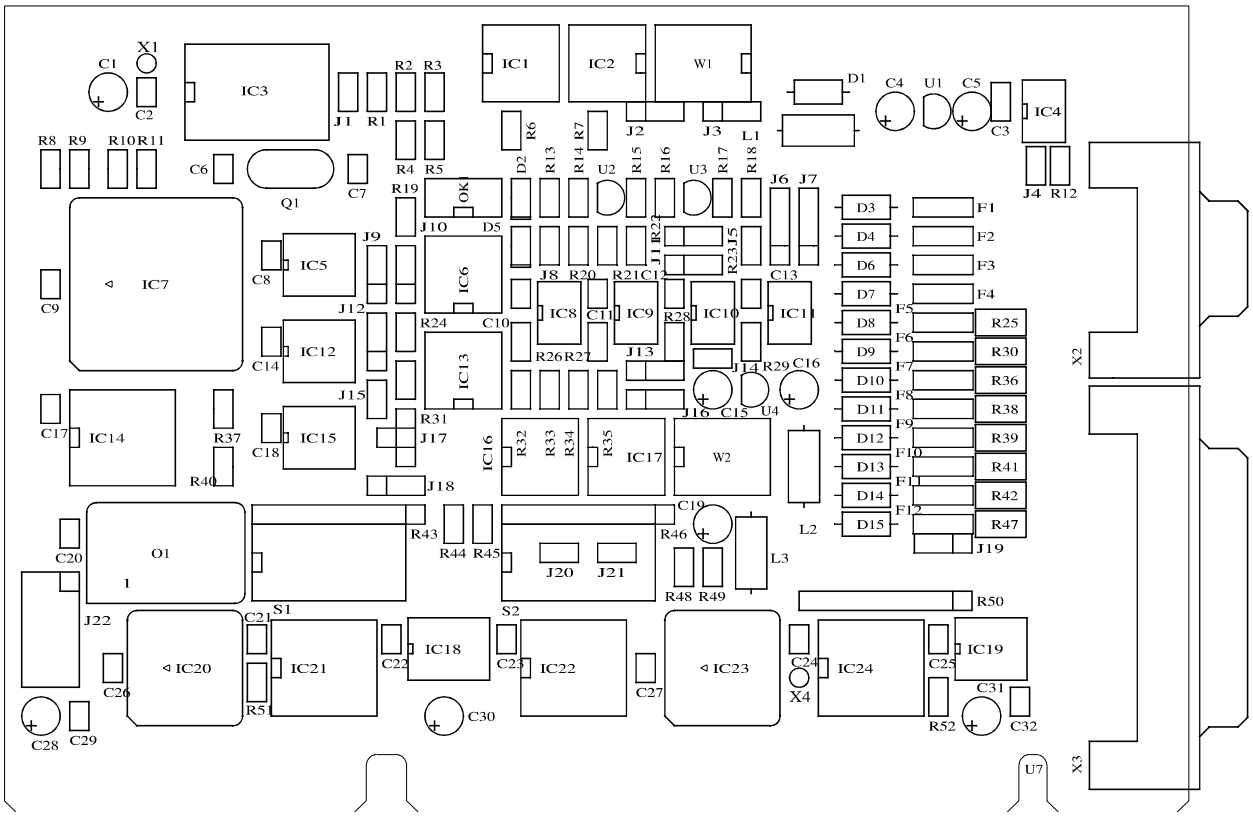
CAN bus interface:

Transfer rate:	max. 1 MBaud
Insulation voltage:	1kV

1 Clamping voltage
2 Surge non repetitive reverse current
3 Peak Pulse Power

4.6 Component scheme

Fig. 4-1: component scheme



5 INSTALLATION

5.1 Delivered software

The board is supplied with a CD-ROM.

The CD contains:
- ADDIREG for Windows NT 4.0 and Windows 95,
  You can also download the ADDIREG program from Internet.
- Standard software for the ADDI-DATA boards:
  - 16-bit for MS-DOS and Windows 3.11
  - 32-bit for Windows NT/95.

5.1.1 Installation under MS-DOS and Windows 3.11

- Copy the contents of PA7200\Dos\German\Disk1 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 directories DRIVER and MONITOR are created.

5.1.2 Installation under Windows NT / 95

- Select the directory PA7200\Win311\Driver\Disk1 or PA7200\WinNT-9x\Driver\Disk1.
- Start the setup 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.

5.3 Configuring the board with ADDIMON

- In the directory MONITOR enter <MON7200>.
- The Monitor program of the board **PA 7200** is loaded.

5.3.1 Help for configuration

- Load ADDIMON
- Select the wished configuration
- Visualize the jumper settings
  (and the DIP switches settings for the base address)
- Configure the board as shown on the screen

5.3.2 Testing

- Insert the board in your PC
- Load ADDIMON
- The basic functions of the board are available
- You can build up and test a simple communication.

5.3.3 Hotline protocol, quick technical support

- Insert the board in your PC
- Load ADDIMON
- If a problem arises

  • Fill in the hotline protocol
  • Print it and send it to our technical support
  • We can thus answer your questions quickly and more precisely.

5.4 Setting the jumpers

5.4.1 Jumper location and factory setting at delivery

Fig. 5-1: Jumper location on the board PA 7200

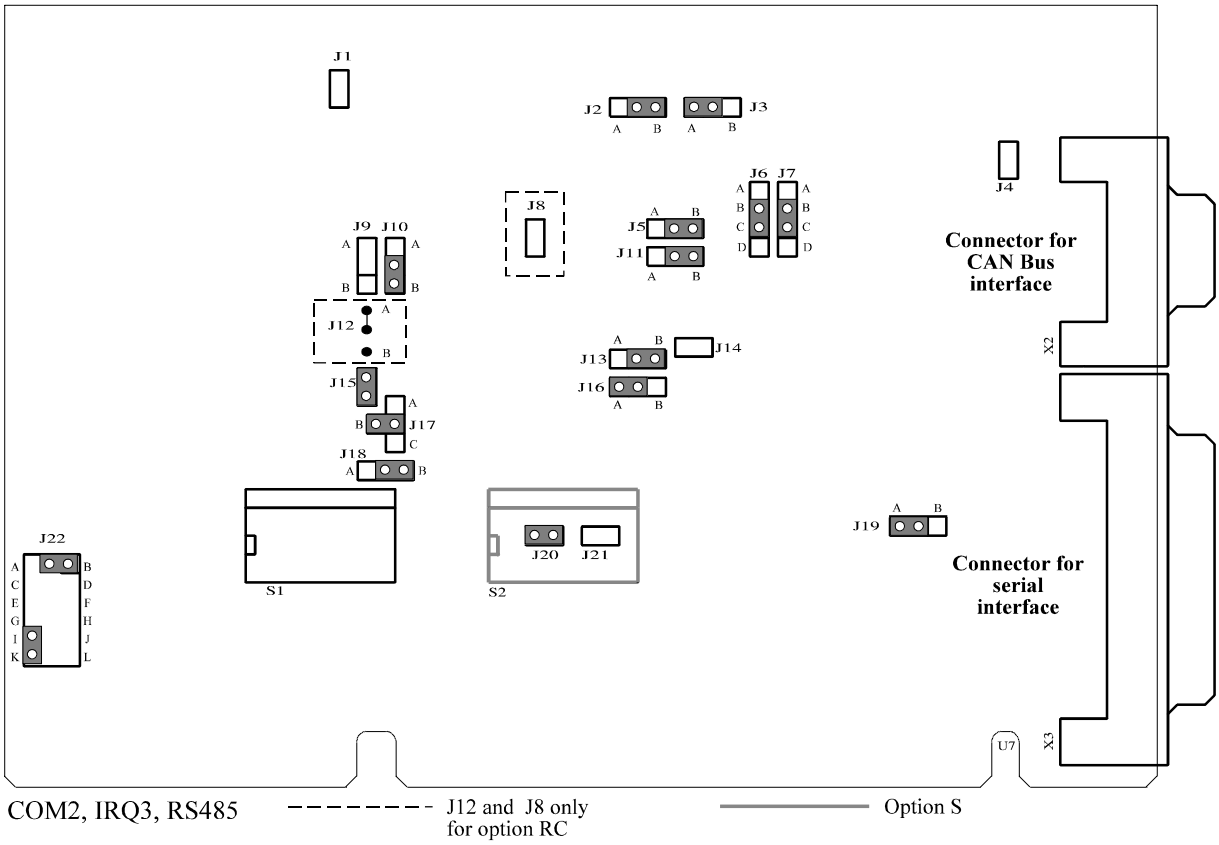


Table 5-1: Jumper settings at delivery

J1	No function	J12	CTS with RTS signal connected
J2	Internal voltage supply of the	J13	Transmission through RTS, DTR, DATA-DIR
J3	CAN bus interface	J14	Not connected
J4	No terminal resistor for CAN	J15	RS 485: UART receives data
J5	RS 485: 120 Ω terminator	J16	Open circuit potential reception = 5 V (high) Rx+
J6	Current Loop: passive transmission	J17	transmission through DTR-Bit
J7	Current Loop: passive reception	J18	RS485 with Echo
J8	Only available by option RC	J19	Voltage potential: mass DC/DC converter
J9	No reception of Current Loop	J20	COM2
J10	Current flows in the rest state	J21	Base address of the serial interface
J11	Open circuit potential reception = 0 V (low) Rx-	J22	IRQ3 serial interface

5.4.2 Jumper assignment

Table 5-2: Jumper assignment

Jumper	Functions for the CAN bus interface
J1	No function
J2	Selection of the positive voltage supply (12 V)
J3	Selection of the negative voltage supply (GND)
J4	Selection of the terminator for CAN bus network
Jumper	**Functions for the serial interface**
J5	Terminator RS422/485-Network
J6	Selection of active/passive transmission for 20mA Current Loop
J7	Selection of active/passive reception for 20mA Current Loop
J8	Option RC: Terminator RS422 CTS signal
J9	Selection of the receiver open circuit potential for 20mA Current Loop. Connects the receiver line of the Current Loop to the receiver line of the UART. **By RS422/RS485 receiving data no jumper shall be set in this field.**
J10	Selection of the open circuit potential for 20mA Current Loop
J11	Selection of the open circuit potential RS422/485 receiver line Rx-
J12	Selection of the CTS signal, internally connected with RTS or externally through option RC, CTS as a RS422 Signal
J13	Selection of the transmitter RS485
J14	Selection of the transmitter RS422
J15	Selection of the receiver RS422/485
J16	Selection of the open circuit potential RS422/485 receiver line Rx+
J17	Selection of the command bits for RS485 transmission
J18	Selection of RS485 with/without echo
J19	Selection of voltage for the protective circuit
J20	Selection of base address for the serial interface
J21	Selection of base address for the serial interface
J22	Selection of the interrupt line for serial and CAN bus interface

5.4.3 RS422

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

Table 5-3: RS422

Jumper	Position	Function	Warning / Remarks
J15	connected	**Data receiveing**	**Warning:** By wrong settings of the Jumper J9 and J13 the board can be damaged
J9	Not connected		
J13	A		
J14	Not connected	**Transmitter permanently ON**	If the board is connected to a RS422 network, you can disable the receiver after transmisson. Make sure that the value of the data bits D1-D7 remains when writing in the MODEM CONTROL REGISTER on **Base +4**.
J14 J17	connected C	**Transmission programmable through DTR-Bit** The **DTR** Bit is the Bit D0 of the MODEM CONTROL REGISTER on Addresse **Base +4** of the UART component. **DTR** = 0: Transmitter disabled (Reset value) **DTR** = 1: Transmitter enabled The **DTR** Bit also controls the hardware pin DTR of the UART.	
J18	B	**Reception** through RD-EN-Bit (Bit D1 on Address Base +7 of the UART component.	RD-EN = 0: Receiver enabled (Resetwert) RD-EN = 1: Receiver disabled

Table 5-3: RS422 (continued)

J12	Permanent bridge standard version	**Modem Control Signals** (PA7200 without Option RC)	The Modem Control Signals are internally connected as follows: DTR → DCD → DSR → RI The Modem Control Signals RTS and CTS are internally connected.
J12	A (Option RC)	**Modem Control Signals** (PA7200 with Option RC) CTS internnally connected with RTS.	The Modem Control Signals RTS and CTS are available as RS422 differential signals on the peripheral connector.
J12	B (Option RC)	**Modem Control Signals** (PA7200 with Option RC) CTS signal from peripheral connector	

5.4.4 RS485

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

Table 5-4: RS485

Jumper	Position	Function	Warning / Remarks
J15	connected	**Data receiving**	**Warning:** By wrong settings of the Jumper J9 the board can be damaged
J9	Not connected		
J18	A	**Alternately** **Receiver/transmitter** Reception through **RTS**, **DTR** or **DATA-DIR** Bit.	Transmitter enabled = receiver disabled transmitter disabled = receiver enabled **DTR, RTS**: Base + 4 **DATA-DIR**: Base + 7
J18	B	Reception through **RD-EN** Bit (Bit D1 on address **Base +7)**	0 = Reception enabled (Resetwert) 1 = Reception disabled After a Reset of the system, **RD-EN** = 0. The system is in reception mode.
		transmission	**Make sure that** the value of the data bit D2-D7 stays on **Base + 4** by writing on the MODEM CONTROL REGISTER.

Table 5-4: RS485 (Continued)

J13	B	Control through Signal sent to J17	
J13	A	Transmitter by hardware disabled.	
J17	A	Control through **DATA-DIR** Bit (Bit D0 on Address **Base +7**) The **DATA- DIR** Bit can be read on address **Base +7**	0 = Transmitter disabled (Reset value) 1 = Receiver enabled
J17	B	Control through **DTR** Bit (Bit D0 on Address **Base +4**).	0 = Receiver disabled (Reset value) 1 = Receiver enabled The **DTR** Bit also controls the Hardware pin DTR of the UART.
J17	C	Control through **RTS** Bit (Bit D1 on Address **Base +4**)	0 = Receiver disabled (Reset value) 1 = Receiver enabled The **RTS** Bit also controls the Hardware pin DTR of the UART.
J12	Permanent bridge	**Modem Control Signals** (PA7200 without Option RC)	The Modem Control Signals are internally connected as follows: DTR → DSR → DCD → RI RTS → CTS are internally connected.
J12	B	**Modem Control Signals** (PA7200 without Option RC)	**On the board with option RC, jumper 12 must be set on pos. B.**

5.4.5 20 mA Current Loop

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

Table 5-5: 20 mA Current Loop

Jumper	Position	Function	Warning / Remarks
J9	A	**Data receiving:** current does not flow in the rest state	**Warning:** By wrong settings of the Jumper J15 the board can be damaged
J9	B	**Data receiving:** current flows in the rest state	
J15	Not connected	With the function Data receiving the jumper J15 should not be set.	
J7	A-B and C-D	**Current source for reception: Active reception:** the board supplies Current Loop	
J7	B-C	**Current source for reception: Passive reception:** the peripheral supplies Current Loop	**Make sure that** the settings correspond to the Current Loop of the peripheral
J10	A	**Data transmitting:** current does not flow in the rest state	
J10	B	**Data transmitting:** current flows in the rest state	
J6	A-B and C-D	**Current source for transmission: Aktive transmission:** the board supplies Current Loop	
J6	B-C	**Current source for transmission: Passive transmission:** The peripheral supplies Current Loop	
J12	Permanent bridge	**Modem Control Signals** (PA7200 without option RC)	The Modem Control Signals are internally connected as follows: DTR → DSR → DCD → RI RTS → CTS are internally connected.
J12	B	**Modem Control Signals** (PA7200 without option RC)	**With the option RC, jumper J12 should be set on pos. B!**

5.4.6 Terminator and open-circuit potential

Table 5-6: Terminator and open-circuit potential

Jumper	Position	Function	Warning / Remarks
J5	A	RS422: 100 Ω terminator	Is the board located at the end of the network?
J5	B	RS485: 120 Ω terminator	Insert a terminator.
J11	B	The receiver lines Rx+ und Rx- are in **RS 422** and **RS 485** mode	
J16	A	Connected to open-circuit potenrial through 1 kΩ resistors.	
J11	A	Open-circuit potential inverted	**IMPORTANT!** The board does not receive any data? Detect if a line interruption has
J16	B		occurred (damaged line, no device connected).

5.4.7 Selecting the reference point of the protection circuitry for serial interfaces

Table 5-7: Reference point of the protection cicuitry for the serial interface

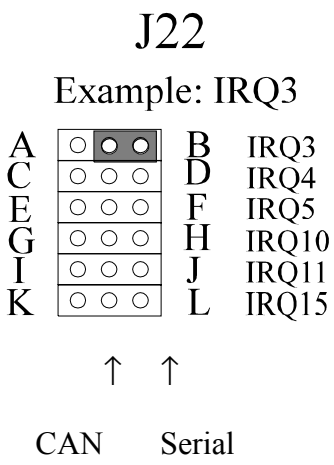
Jumper	Position	Function	Warning / Remarks
J19	A	Derivation of noise voltage through the secondary ground GND of the DC/DC converter	Settings at delivery.
J19	B	Derivation of noise voltage through pin of the 25 pole SUB-D pin connector	**Important!** **Make sure that** pin 1 of the 25 pole SUB-D pin connector is externally earthed.

5.4.8 Interrupt

Table 5-8: Base address and corresponding interrupt lines for the serial interface

Designation	Address	Interrupt
COM1	3F8H	IRQ4 (XT)
COM2	2F8H	IRQ3 (XT)
COM3	3E8H	IRQ10 (AT)
COM4	2E8H	IRQ11 (AT)

Fig. 5-2: Select the interrupt line for the serial interface and the CAN bus interface



5.5 Base address

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

Configurating the base address with ADDIMON

The delivered disk contains the monitoring program ADDIMON that helps the user to configurate the base address.

Before installing the board

At delivery, the base address of the CAN bus interface is set to 0300H, the base address of the serial interface to 2F8H (COM2).

- **Check**, whether
  - the base addresses are free.
  - the address range required by the board is not already used by the PC or by boards already installed in the PC.

Is the base address or the address range **already used**?

- **Set** the base address for the CAN bus interface through the 8 pole DIP switches S1**.**
- **Set** the base address for the serial interface through J20 and J21.

Possible settings of the base address:

• COM1-COM4	serial interface
• COM1-COM4 or any address	serial interface (option S)
• Any address	CAN bus interface

IMPORTANT!
Find out which serial and parallele interfaces are already used by your PC.
Make sure that the address range of the serial interface and the address range of the CAN bus interface do not overlap.

5.5.1 Base address of the serial interface

Table 5-9: Base address of the serial interface

Designation	Address	Jumper	Jumper
COM1	3F8H	without J21	without J20
COM2	2F8H	without J21	with J20
COM3	3E8H	with J21	without J20
COM4	2E8H	with J21	with J20

DIP switches
Option S: the serial interface is delivered with DIP switches S2.

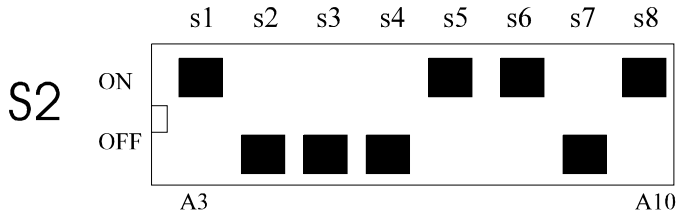
The base address of the serial interface is decoded in steps of 8 bytes.

The following figure is an example of the address 0390H
decoded for the serial interface

	MSB															LSB
Decoded address bus	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Selected base address (Hexadecimal)	0				3				9				0			
Selected base address (binary)	0	0	0	0	0	0	1	1	1	0	0	1	0	0	0	0
DIP Switch S1 Logic "0"= ON Logic "1" = OFF						s8* ON	s7 OFF	s6 OFF	s5 OFF	s4 ON	s3 ON	s2 OFF	s1 ON	X	X	X

X:decoded address range
* : Permanently decoded on logic "0"

IMPORTANT!
The switch **s1** is located on the **left side** of the DIP switches!



5.5.2 Base address of the CAN bus interface

Fig. 5-3: Base address of the CAN bus interface

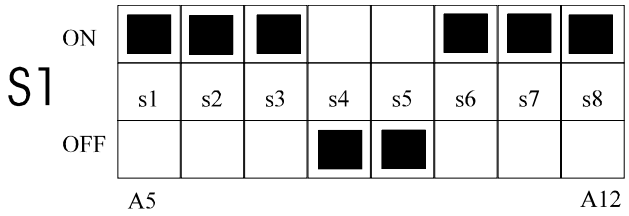
The base address of the CAN bus interface is decoded in steps of 32 Bytes through the DIP switches S1.

The following figure is an example of the base address 0300H decoded for the CAN bus interface.

	MSB															LSB
Decoded address bus	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Selected base address (hexadecimal)	0				3				0				0			
Selected 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

IMPORTANT!
The switch S1 is located on the left side of the DIP switches!



5.6 Inserting the board

IMPORTANT!
Do observe the *safety instructions.*

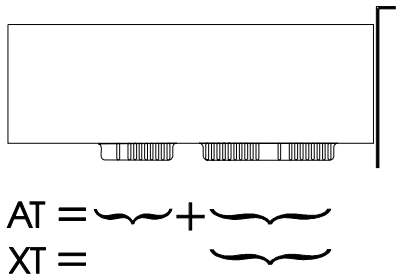
5.6.1 Opening the PC

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

5.6.2 Selecting a free slot

Two types of ISA slots are available: XT and AT.

Fig. 5-4: Types of slots



If necessary, the board can also be used in EISA slots under certain conditions. See in the PC manual which types of slots are free.

1. Decide
in which type of slot to insert the board.

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.

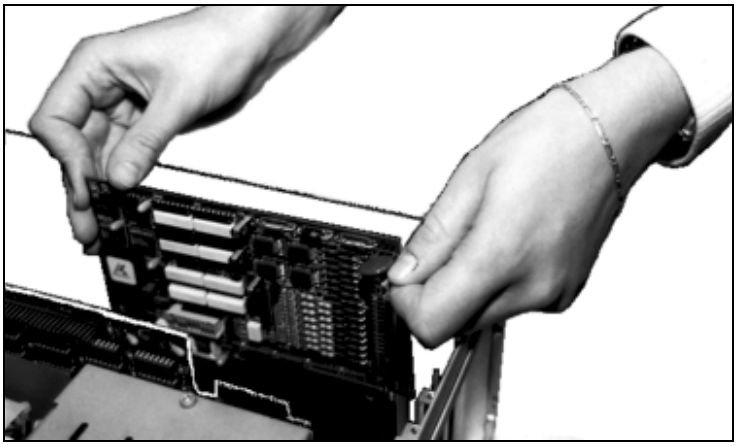
3. Discharge yourself from electrostatic charges

4. Take the board out of its protective bag
and put the board with its opposite side on the protective bag

5.6.3 Plugging the board into the slot

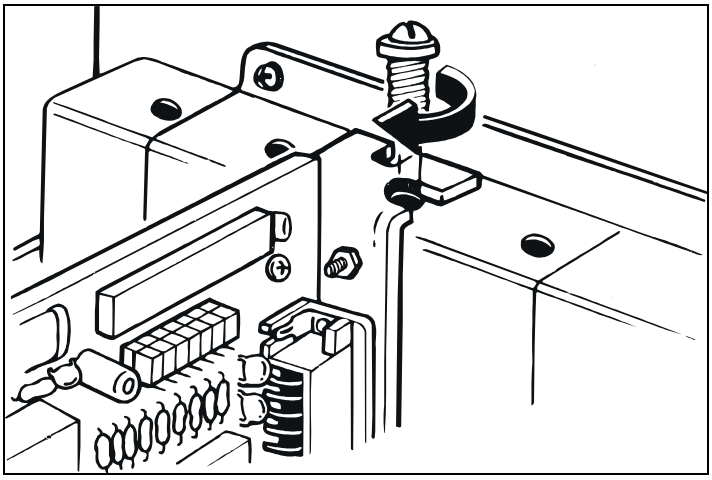
- **Discharge yourself from electrostatic charges**
- Insert the board **vertically into the chosen slot.**

Fig. 5-5: Inserting the board



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

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



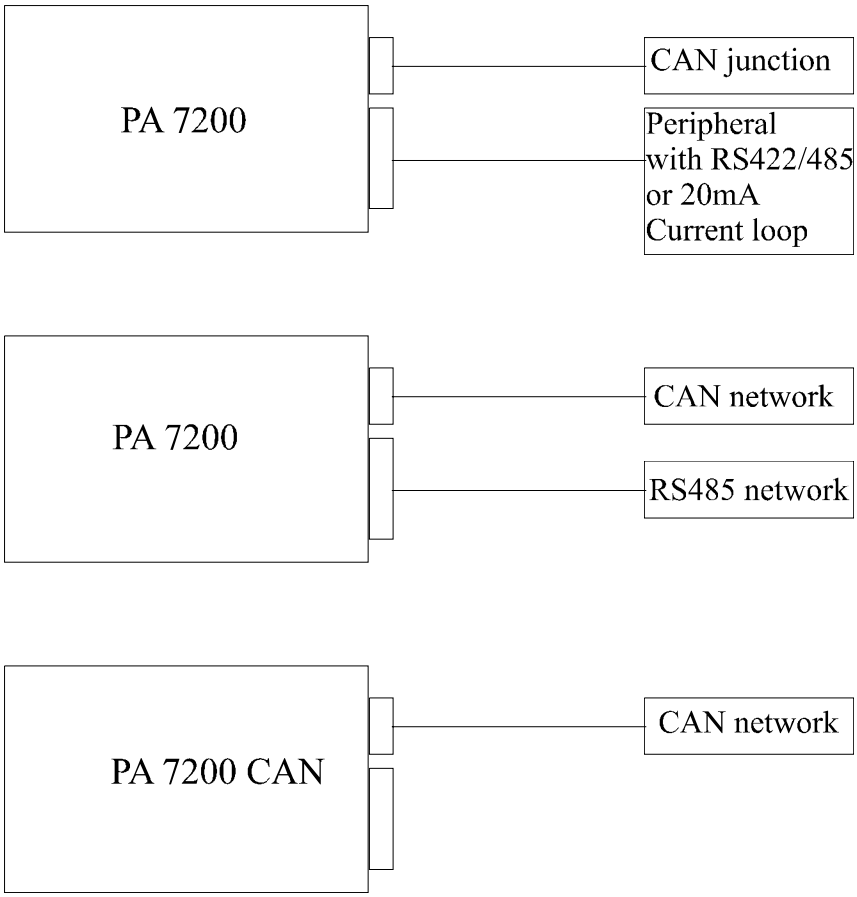
- **Tighten all the loosed screws.**

5.6.4 Closing the PC

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

6 CONNECTION TO THE PERIPHERAL

Fig. 6-1: Connection principle



6.2 Connection pin assignment

6.1.2 Serial interface

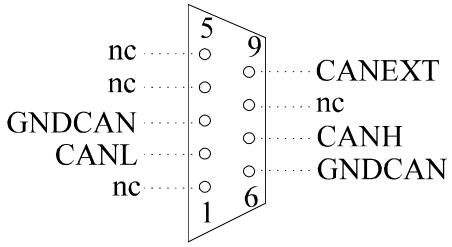
Fig. 6-2: 25 pole pin connector

USER DESIGNATION	NAME	PINS	NAME	USER DESIGNATION
	RTS-	13		
	-Tx-CL-DATA	12	25 -RCV-CL-DATA	
	+Tx-CL-DATA	11	24 Tx+	
	GNDA	10	23 Tx-	
	Current Source Out 1	9	22 nc	
	nc	8	21 CTS+	
	GNDA	7	20 nc	
	nc	6	19 CTS-	
	nc	5	18 +RCV-CL-DATA	
	nc	4	17 Rx+	
	nc	3	16 Rx-	
	nc	2	15 Current Source Out 2	
	PG1	1	14 RTS+	

Pin	Signal	Meaning	Operating mode
1	PG1	Protection Ground	
2	Not connected		
3	Not connected		
4	Not connected		
5	Not connected		
6	Not connected		
7	GNDA	Isolated Ground A	
8	Not connected		
20	Not connected		
22	Not connected		
17	Rx+	RS 422 Receive/ RS 485 Transmit	→RS 422/RS 485
16	Rx-	RS 422 Receive/ RS 485 Transmit	→RS 422/RS 485
24	Tx+	RS 422 Transmit	RS 422
23	Tx-	RS 422 Transmit	
14	RTS +	Request to Send +	RS 422 (OPTION RC)
13	RTS -	Request to Send -	
21	CTS +	Clear to Send +	
19	CTS -	Clear to Send -	
10	GNDA	Isolated Ground	20mA Current Loop
11	+Tx-CL-DATA	TTY Transmitter	
12	-Tx-CL-DATA	TTY Transmitter	
18	+RCV-CL-DATA	TTY Receiver	
25	-RCV-CL-DATA	TTY Receiver	
9	Current Source Out 1	Transmitter Current Source	
15	Current Source Out 2	Receiver Current Source	

6.2.2 CAN bus interface

Fig. 6-3: 9 pole pin connector



Pin	Signal	Meaning
1	Not connected	
2	CANL	CAN Transceiver -
7	CANH	CAN Transceiver +
3	GNDCAN	Extern Ground for CAN Transceiver
6	GNDCAN	Extern Ground for CAN Transceiver
9	VCCEXT	Extern Power for CAN Transceiver
4	Not connected	
5	Not connected	
8	Not connected	

6.3 Cabling RS422

Fig. 6-4: Cabling RS422 for the serial interface

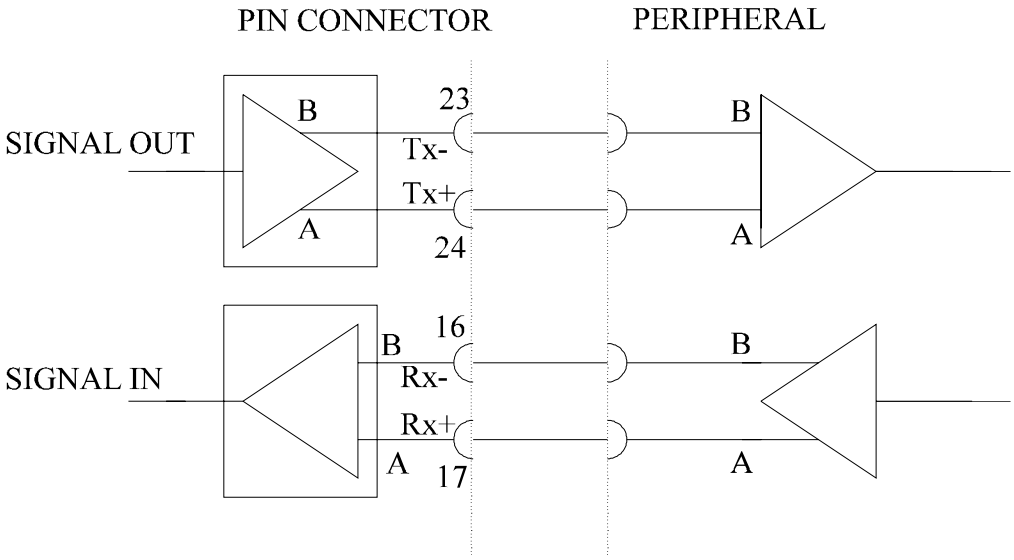
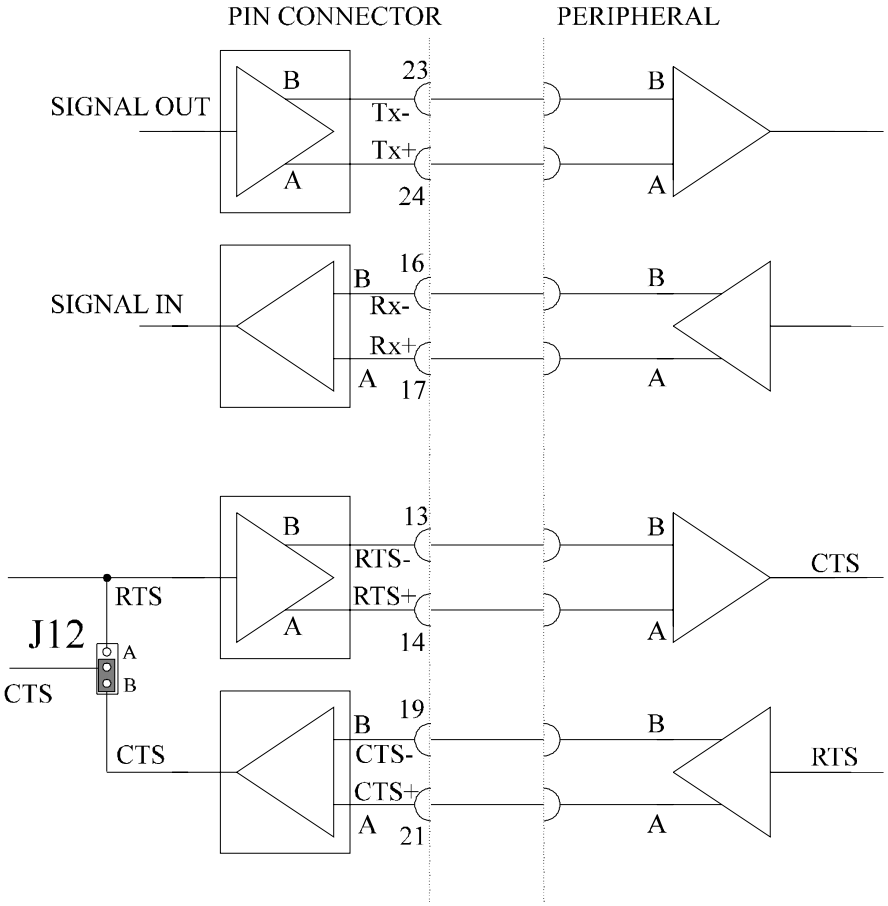
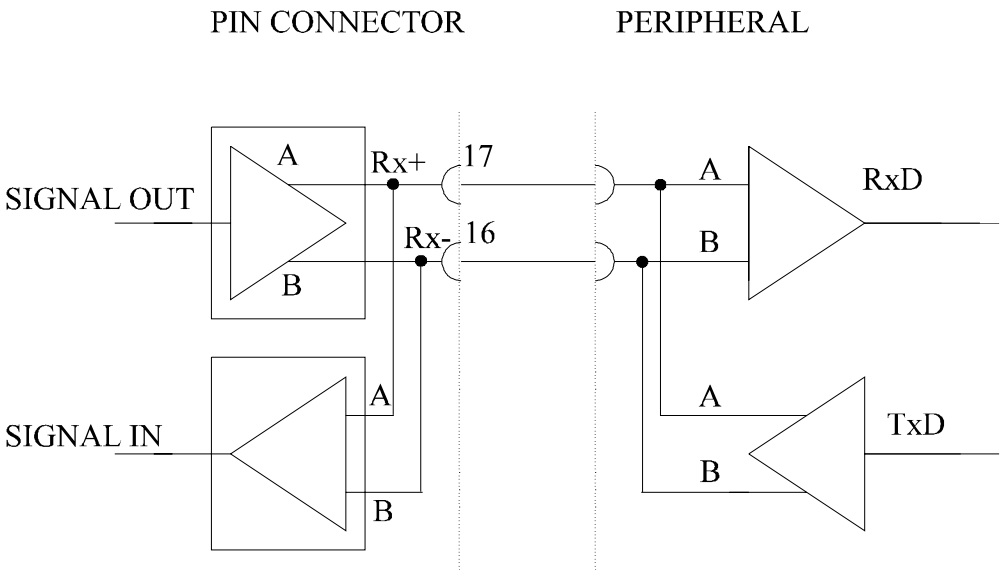


Fig. 6-5: Cabling RS422 with option RC



6.4 Cabling RS485

Fig. 6-6: Cabling RS485



6.5 Cabling 20mA Current Loop

Fig. 6-7: Active transmission / active reception

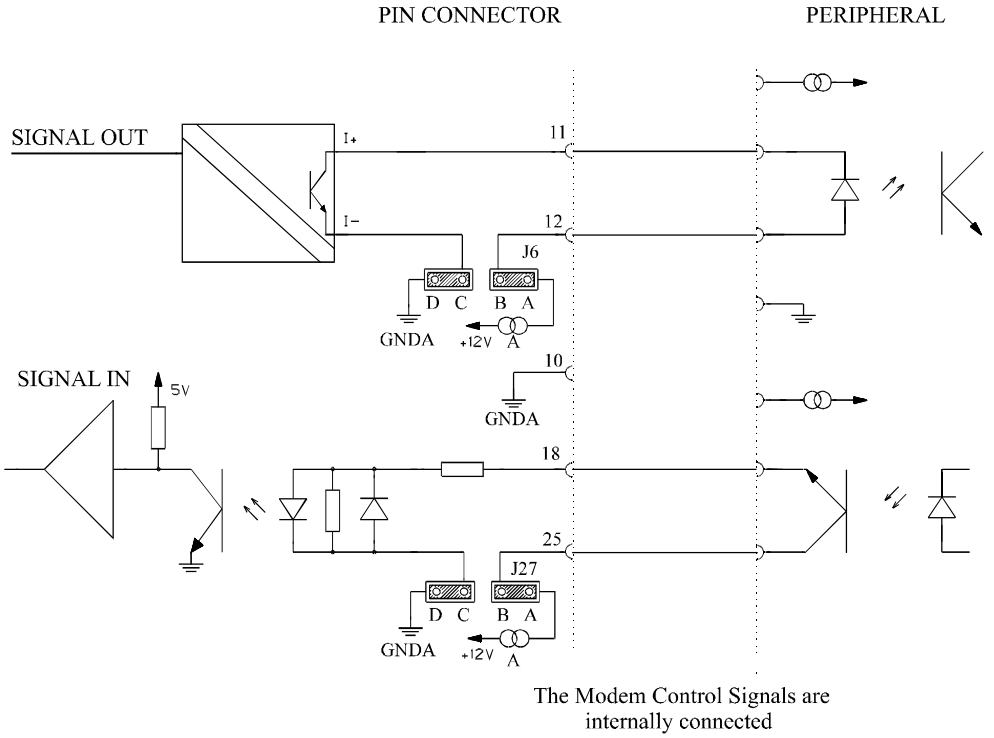


Fig. 6-8: Active transmission / passive reception

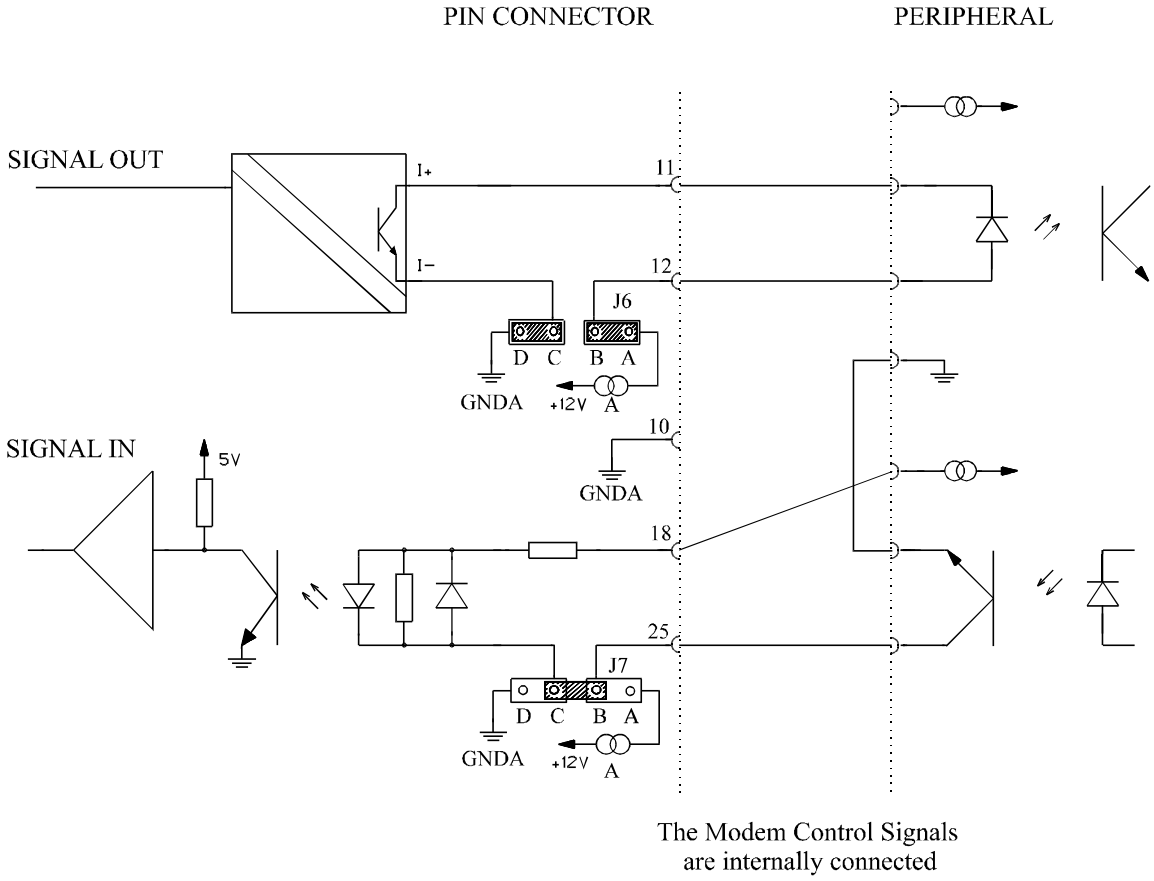


Fig. 6-9: Passive transmission / active reception

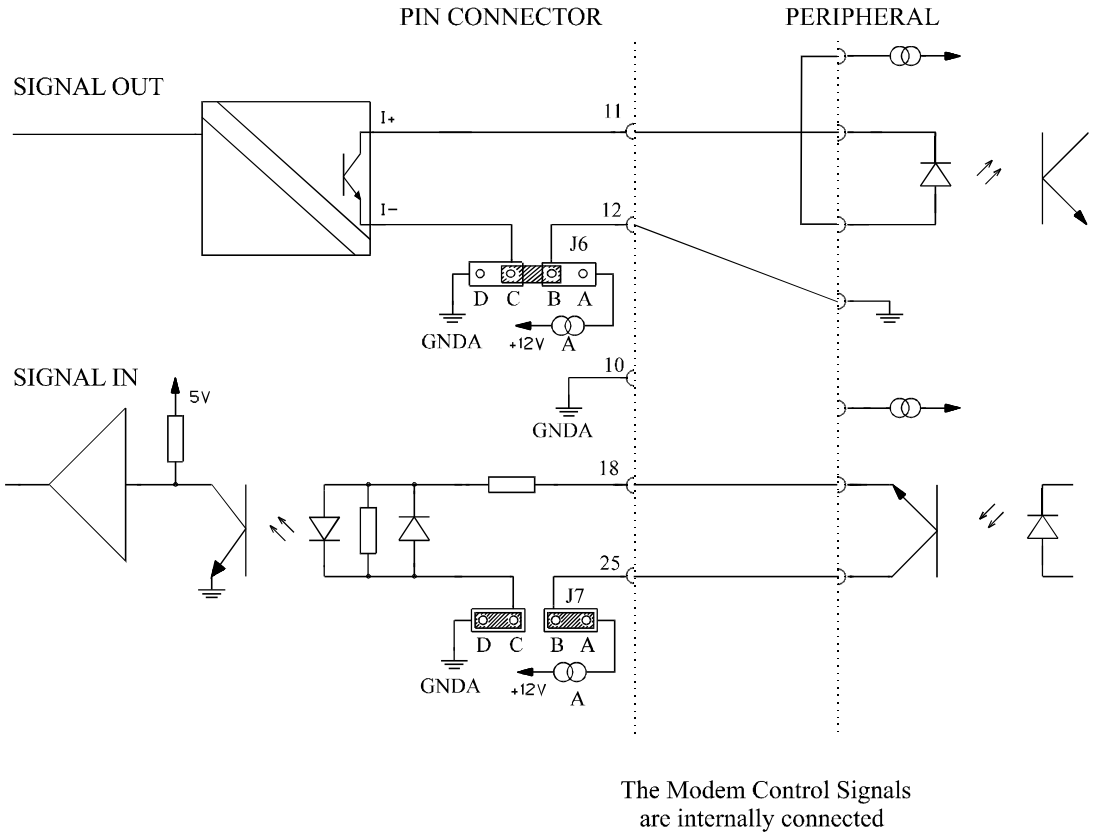
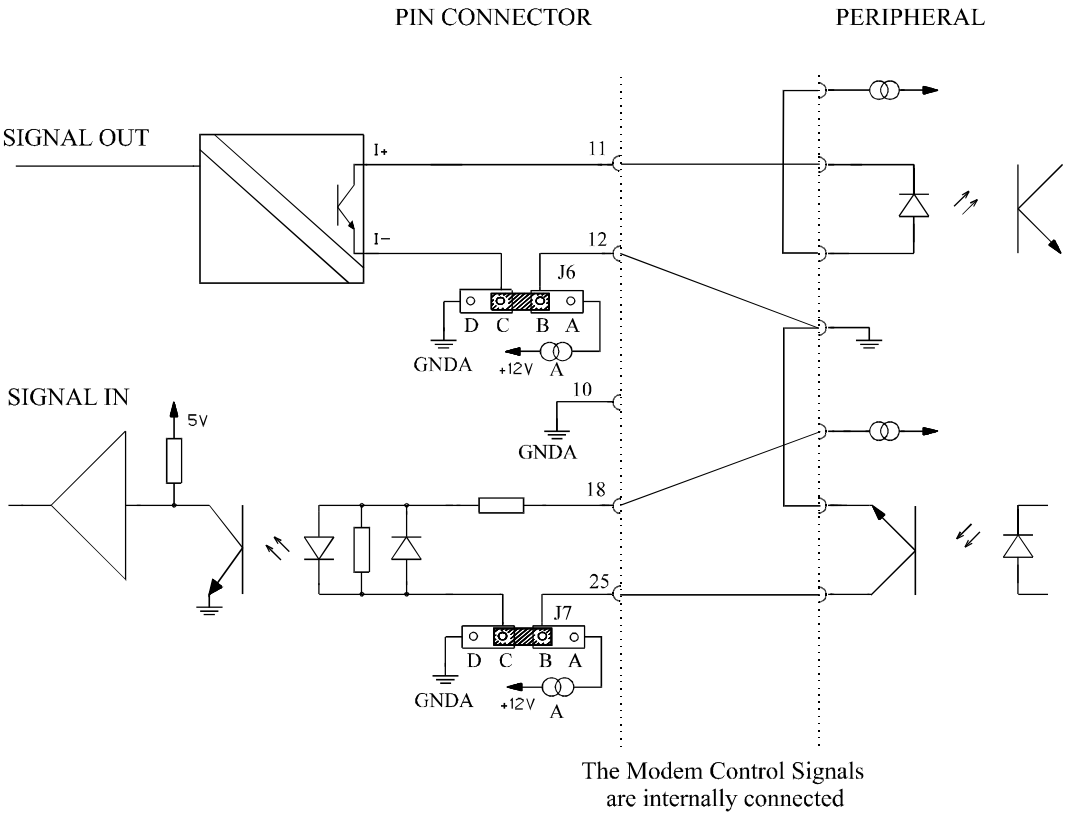


Fig. 6-10 Passive transmission / passive reception



6.6 Cabling the CAN bus interface

Fig. 6-11: Internal voltage supply of the CAN bus interface

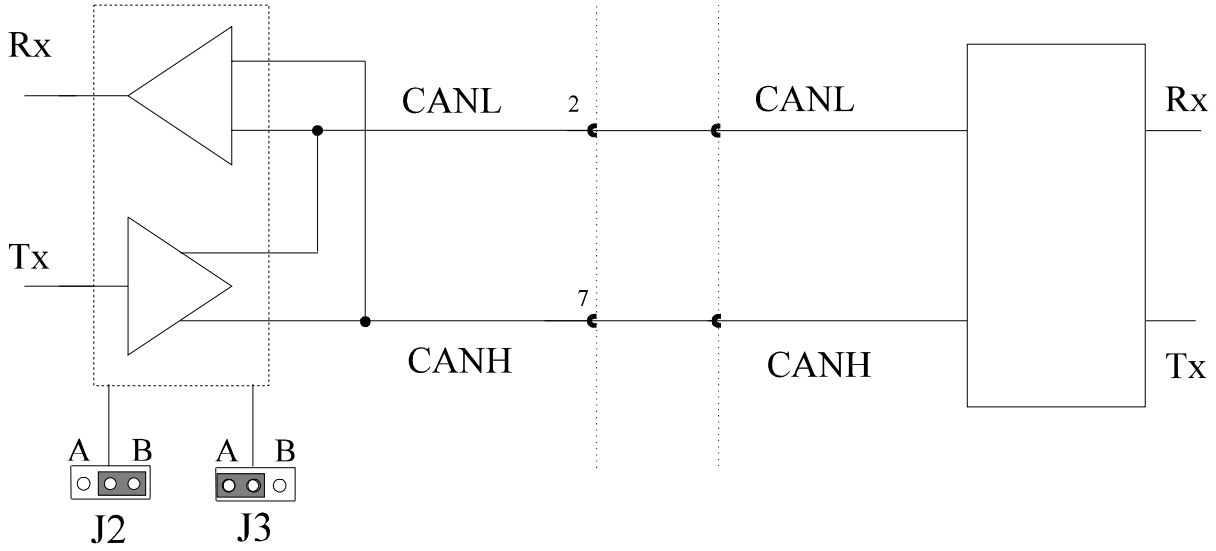
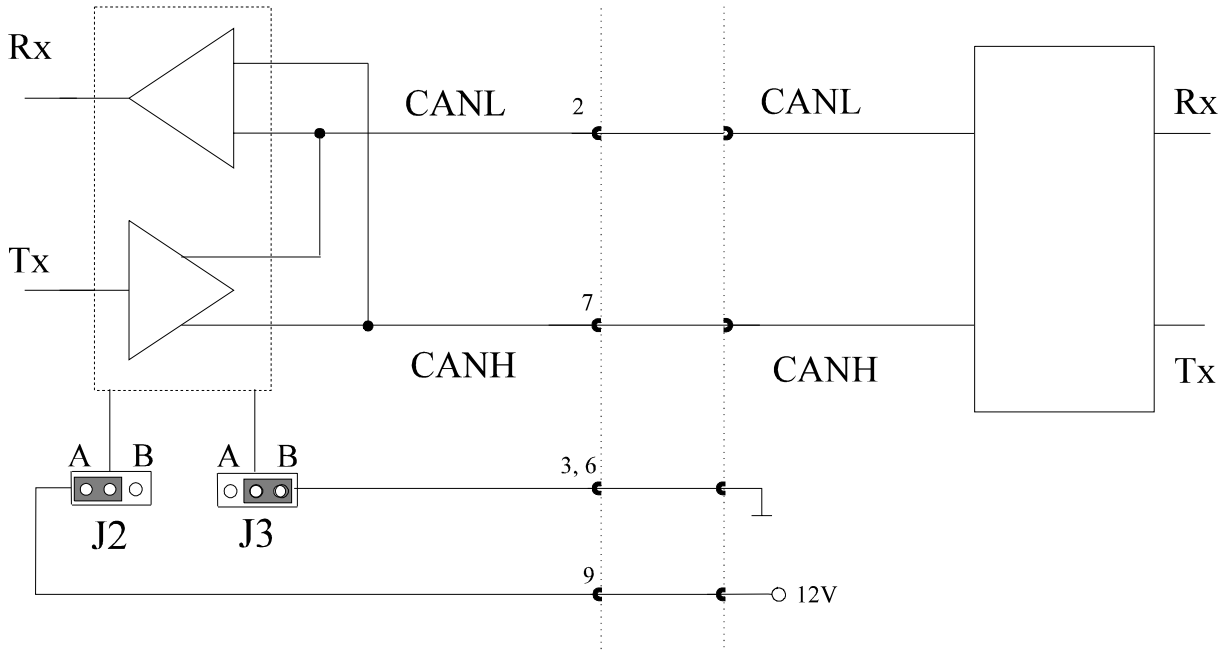
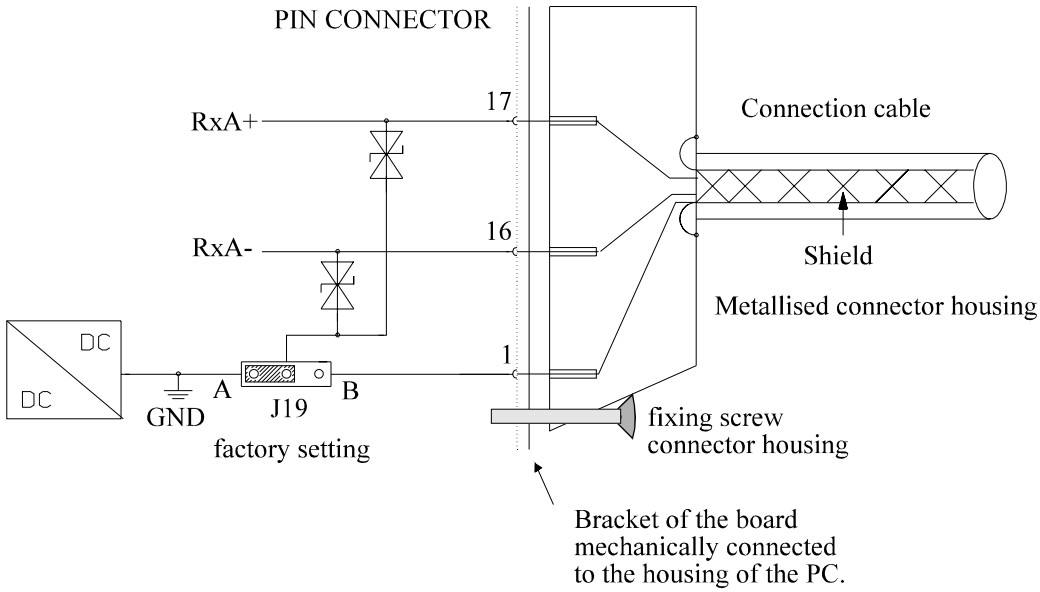


Fig. 6-12: External voltage supply of the CAN bus interface



6.7 Selection of the reference point of the protection circuitry

Fig. 6-13: Selection of the reference point of the protection circuitry



Is pin 1 of the 25 pole SUB D pin connector used as derivation line?

• **Connect** pin 1 of the connection cable (see fig. 6-12)
- with the shield of the cable
as well as
- with the housing of the female connector connected to the cable.

7 DEVICE DRIVER - CAN LAYER 2 INTERFACE

7.1 Introduction

IMPORTANT!
We have used the following notation in the text:

Function: "i_PA7200_SetBoardInformation"
Variable: *ui_Address*

Type Declaration

	Borland C	Microsoft C	Borland Pascal	Microsoft Visual Basic Dos	Microsoft Visual Basic Windows
VOID_	void	void	pointer		any
BYTE_	unsigned char	unsigned char	byte	integer	integer
INT_	int	int	integer	integer	integer
UINT_	unsigned int	unsigned int	word	long	long
LONG_	long	long	longint	long	long
PBYTE_	unsigned char *	unsigned char *	var byte	integer	integer
PINT_	int *	int *	var integer	integer	integer
PUINT	unsigned int *	unsigned int *	var word	long	long
PCHAR	char *	char *	var string	string	string

The device driver for the CAN bus interface is made of three parts:

- The first part is a runtime: "RUN7200.EXE". This runtime is only used for DOS.
- The second part is a configuration file in which you define the messages.
- The third part is a function library used to control the CAN bus interface. When working with DOS, the control is made with the runtime, when working with Windows, it is made through a DLL.

Remark for VB DOS: The VB DOS examples use the C functions.

7.1.1 The notion of "object"

Messages to be transmitted or received are named "objects". Define them in an object configuration file (OCF). These objects not only constist of the message, but also of other information such as for ex. the **Identifier**. An object consists of 10 bytes and is structured as follows:

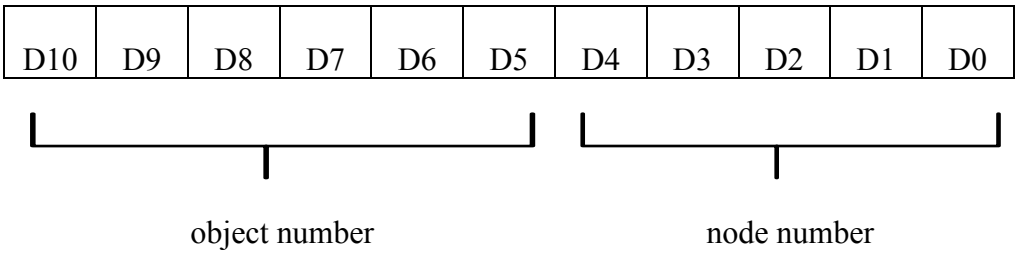
Table 7-1: Object structure

D7	D6	D5	D4	D3	D2	D1	D0
IDENTIFIER							
			RTR BIT	DATA LENGTH			
BYTE 1							
BYTE 2							
BYTE 3							
BYTE 4							
BYTE 5							
BYTE 6							
BYTE 7							
BYTE 8							

Identifier

An Identifier is sent everytime a message is transmitted or received.
This Identifier consists of 11 bits and is structured as follows:

Fig. 7-1: Structure of the Identifier - example



Object number

Number of an object within a CAN node

Node number

This number corresponds to the device in the network which transmits/receives the message. It cannot be >199, because a CAN network can run max.200 interfaces.

IMPORTANT!
Some software functions need the Identifier. To convert the object Handle into an Identifier, the Parameter 5 of the OCF is read by the function i_PA7200_GetIdentifier.

RTR bit

This bit defines/establishes whether a message or a remote[1] is to be/has been sent/received
1: Remote
0: Message

IMPORTANT!
The received objects are filtered through the Parameter5 of the object configuration file (OCF).

Data length code

Amount of transmitted or received data.

Message

Bytes 1 to 8: Transmitted or received message.

7.1.2 Receiving an object

When an object is received, an interrupt is triggered.

The object is read and its Identifer is checked for acceptance through the OCF. If the Identifier is correct, the received message and its object Handle are stored or passed on to the user function "v_User_ReceiveFunction".

Store: in FIFO or array

The received message and object Handle can be stored in a FIFO or in an array.

FIFO: The received message and object Handle are automatically stored in a FIFO (First In First Out Buffer). You can read the received message and object Handle with the function "i_PA7200_ReadDataAndHandleFromFIFO".

Array: The received message and object Handle are automatically stored in the index of an array. The index is defined through the OCD.Read the message with the function i_PA7200_ReadDataAndHandleFromArray".

[1] A remote is a commando sent by a master

User routine

With the function "v_User_ReceiveFunction", you receive the object Handle and the received message. This allows you to process the message immediately.

7.1.3 Transmitting objects

The objects are written directly on the CAN interface.

In order to transmit messages, the CAN interface needs all the information described in chapter 7.1.1 (Identifier, RTR bit, Data length code).

There are two ways to transmit objects and to provide the CAN interface with this information.

User-defined Identifier

You can generate the Identifier and the Data length code yourself and transmit the objects with the function "i_PA7200_WriteDataPerObjectIdentifier".

OCF-defined Identifier

You can transmit the object with the function "i_PA7200_WriteDataPerobjectHandle".
This function reads the Data length code and object identifier from the OCF.

7.1.4 Remote

Three modes are available:

Automatic mode

The remote objects are directly processed, without beeing passed on to the user. The user cyclically calls up the function "i_PA7200_TestAndWriteRemote" to transmit the remote objects.
This function extracts the index from the remote array through the OCF and writes the remote object in the remote array.

FIFO mode

The remote Handle is stored in a FIFO and read by the function "i_PA7200_ReadRemoteHandleFromFIFO".

User mode

The object is passed on to the user through the function "v_User_RemoteFunction".

7.2 Runtime "RUN7200.EXE"

7.2.1 Introduction

The program "RUN7200.EXE" is a resident memory program accessed by a software interrupt. This programm is only used when working with DOS. When working under windows, the DLL is used.

7.2.2 Install the runtime

To install the runtime, enter an interrupt number when calling up the program "RUN7200.EXE" as follows:
RUN7200.EXE XX /I .
XX corresponds to the interrupt number.
If you program with Visual Basic for DOS, enter the following parameter:
RUN7200.EXE XX /I /B.
The runtime has only to be installed once.

7.2.3 Desinstall the runtime

To desinstall the runtime, enter the following parameter:
RUN7200.EXE XX /D.
XX corresponds to the interrupt number.

7.3 Object configuration file (OCF)

7.3.1 Introduction

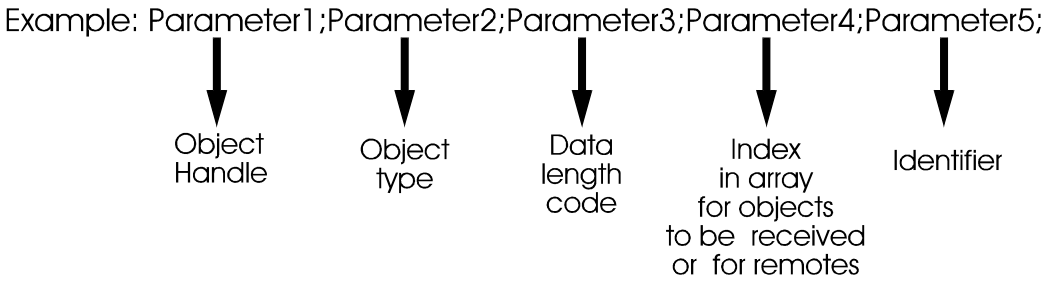
The object configuration file (OCF) links identifiers with objects. An alteration of the object identifier is achieved by altering this file without changes in the program.
The node number is automatically defined according to the defined object identifier.
Also defined in this file:

- Objects to be received and remote objects
- Some objects to be transmitted
- Data length code and index in the receiving array

7.3.2 Line structure

The object configuration file (OCF) is a text file which can be generated with any editor. Every line must contain 5 parameters, separated with ´;´.

Fig. 7-2: OCF line structure



These five parameters are described in the following sections.

Parameter1: object handle

Define this value between 0 and 2047. Give every object a handle.
This handle cannot be identical for several objects.

Parameter2: object type

Object type	Description
PA7200_SIMPLE_RECEIVE_OBJECT_ARRAY	The received object is stored in the receiving array without search option
PA7200_SEARCH_RECEIVE_OBJECT_ARRAY	The received object is stored in the receiving array with search option
PA7200_RECEIVE_FIFO_OBJECT	The received object is stored in the FIFO
PA7200_RECEIVE_USER_ROUTINE_OBJECT	The received object is passed on to the user by the function "v_User_ReceiveFunction"
PA7200_WRITE_OBJECT	Object to be transmitted
PA7200_REMOTE_AUTO_OBJECT	The remote object is automatically processed
PA7200_REMOTE_FIFO_OBJECT	The remote object is stored in the FIFO
PA7200_REMOTE_USER_ROUTINE_OBJECT	The received object is passed on to the user by the function "v_User_RemoteFunction"

PA7200_SIMPLE_RECEIVE_OBJECT_ARRAY:
The object is defined as an object to be received.
The message is stored in the index of the receiving array.

PA7200_SEARCH_RECEIVE_OBJECT_ARRAY:
This object type is the same as PA7200_SIMPLE_RECEIVE_OBJECT_ARRAY with the difference that you can read the last changed object in the array with the function "i_PA7200_SearchNextobjectFromArray".

PA7200_RECEIVE_FIFO_OBJECT:
The object is defined as an object to be received.
The messages is stored in the FIFO.

PA7200_RECEIVE_USER_ROUTINE_OBJECT:
The object is passed on to the user by "v_User_ReceiveFunction".

PA7200_WRITE_OBJECT:
The object is defined as an object to be transmitted.

PA7200_REMOTE_AUTO_OBJECT:
The object is defined as a remote.
The remote object is processed in the background.

PA7200_REMOTE_FIFO_OBJECT:
The object is defined as a remote.
The remote object is stored in the FIFO.

PA7200_REMOTE_USER_ROUTINE_OBJECT:
The object is defined as a remote.
The remote object is passed on to the user by the function "v_User_RemoteFunction".

Parameter3: Data length code

Object type	Data length code
PA7200_SIMPLE_RECEIVE_OBJECT_ARRAY	Defines the Data length code of the object to be received (0 to 8 bytes)
PA7200_SEARCH_RECEIVE_OBJECT_ARRAY	Defines the Data length code of the object to be received (0 to 8 bytes)
PA7200_RECEIVE_FIFO_OBJECT	Defines the Data length code of the object to be received (0 to 8 bytes)
PA7200_RECEIVE_USER_ROUTINE_OBJECT	No meaning
PA7200_WRITE_OBJECT	Defines the Data length code of the object to be transmitted (0 to 8 bytes)
PA7200_REMOTE_AUTO_OBJECT	Defines the Data length code of the message to be transmitted after the reception of a remote object (0 to 8 bytes).
PA7200_REMOTE_FIFO_OBJECT	No meaning
PA7200_REMOTE_USER_ROUTINE_OBJECT	No meaning

Parameter4: Array index

Object type	Index parameter
PA7200_SIMPLE_RECEIVE_OBJECT_ARRAY	Defines the index in the receiving array
PA7200_SEARCH_RECEIVE_OBJECT_ARRAY	Defines the index in the receiving array
PA7200_RECEIVE_FIFO_OBJECT	No meaning
PA7200_RECEIVE_USER_ROUTINE_OBJECT	No meaning
PA7200_WRITE_OBJECT	Defines the RTR bit
PA7200_REMOTE_AUTO_OBJECT	Defines the index in the transmitting remote array
PA7200_REMOTE_FIFO_OBJECT	No meaning
PA7200_REMOTE_USER_ROUTINE_OBJECT	No meaning

Parameter5: Identifier

Give a value between 0 and 2047.
This value cannot be identical for several objects.
It is used as a filter for the message to be received and the remote objects, and as transmittion identifier for the objects to be transmitted.

Example of an OCF

```
1;PA7200_SIMPLE_RECEIVE_OBJECT_ARRAY;8;0;500;
10;PA7200_SEARCH_RECEIVE_OBJECT_ARRAY;4;1;501;
100;PA7200_RECEIVE_FIFO_OBJECT;6;0;600;
20;PA7200_RECEIVE_USER_ROUTINE_OBJECT;1;0;100;
0;PA7200_WRITE_OBJECT;3;0;211;
12;PA7200_REMOTE_AUTO_OBJECT;7;8;50;
1000;PA7200_REMOTE_FIFO_OBJECT;0;0;51;
200;PA7200_REMOTE_USER_ROUTINE_OBJECT;0;0;52;
```

Explanation:

- The object 1 is defined as an object to be received. The object is stored at reception in index 0 of the receiving array.
  This object cannot be extracted by the function "i_PA7200_SearchNextobjectFromArray".
  The message is 8 bytes long.
  The identifier for this object is 500.
- The object 10 is defined as an object to be received. The object is stored at reception in index 1 of the receiving array.
  This object can be extracted by the function "i_PA7200_SearchNextobjectFromArray".
  The message is 4 bytes long.
  The identifier for this object is 501.
- The object 100 is defined as an object to be received.
  The object is stored at reception in the FIFO.
  This object can be extracted by the function "i_PA7200_ReadDataAndHandleFromFIFO".
  The message is 6 bytes long.
  The identifier for this object is 600
- The object 20 is defined as an object to be received.
  The object is passed on to the user by the function "v_User_ReceiveFunction".
  The message is 1 bytes long.
  The identifier for this object is 100.
- The object 0 is defined as an object to be transmitted.
  The message is 3 bytes long.
  The identifier for this object is 211.
- The object 12 is defined as a remote object in automatic mode.
  The message is 7 bytes long.
  The message to be transmitted is in index 8 in the remote transmitting array.
  The identifier for this object is 50.
- The object 1000 is defined as a remote object in FIFO mode.
  This object can be extracted by the function "i_PA7200_ReadRemoteFromFIFO"
  The identifier for this object is 51.
- The object 200 is defined as a remote object.
  The object The object is passed on to the user by the function "v_User_RemoteFunction".
  The identifier for this object is 52.

7.4 Software functions (API)

7.4.1 Address and interrupt

1) i_PA7200_SetBoardInformation (...)

Syntax:

```
<Return Value> = i_PA7200_SetBoardInformation (UINT   ui_Address,
                                               BYTE   b_CANInterruptNbr,
                                               BYTE   b_Baudrate,
                                               PCHAR  pc_ObjectDefineFile,
                                               PBYTE  pb_BoardHandle
                                               PUINT  pui_Zeile)
```

Parameter:

UINT	ui_Address	: Base address of the **PA 7200**
BYTE	b_CANInterrptNbr	: Interrupt line of the board for the CAN interface (IRQ3, 5, 10, 11, 12, 14 or 15).
BYTE	b_BaudRate	: Baud rate of the CAN interface: PA7200_BAUDRATE_50K: 50 kbits/s PA7200_BAUDRATE_100K : 100kbits/s PA7200_BAUDRATE_200K : 200kbits/s PA7200_BAUDRATE_500K : 500kbits/s PA7200_BAUDRATE_1M : 1Mbits/s PA7200_BAUDRATE_1M6: 1,6Mbits/s
PCHAR	pc_ObjectDefineFile	: Name of the object configuration file
PBYTE	pb_BoardHandle	: Handle[1] of the **PA 7200** for using the functions.
PUINT	pui_Zeile	: Number of the OCF line in which an error is found.

Task:

Checks if the **PA 7200** is installed and stores the base address and interrupt line number. The user receives a handle to use the next functions. Handles allow to run several boards.

Return value:

0: No error
-1: Base address used by another PA7200
-2: Error in the object configuration file (OCF)
-3: The same object handle is used several times in the OCF
-4: The same object identifier is used several times in the OCF
-5: Interrupt line not available
-6: Interrupt line used by another PA7200
-7: No handle available for the board (max. 5 handles)
-8: Base address cannot be set
-9: Board not installed
-10: Board cannot be configurated
-11: Buffer error by generating object
-12: Baud rate parameter error
-13: No runtime installed

[1] Identification number of the board

2) i_PA7200_ResetBoard (..)

Syntax:
<Return value> = i_PA7200_ResetBoard (BYTE_ b_BoardHandle)

Parameter:
BYTE b_BoardHandle : handle the **PA 7200**

Task:
Stops the operation of a **PA 7200.**
De-installs the interrupt routine, in case the interrupt management of all **PA7200** is stopped.

Return value:
0: No error
-1: Handle parameter of the board is wrong
-2: No runtime installed

3) i_PA7200_SetUserReceiveObjectFunction (...)

IMPORTANT!
This function can only be called up once.

Syntax:
<Return Value> = i_PA7200_SetUserReceiveObjectFunction
(VOID v_User_ReceiveFunction
(BYTE b_BoardHandle,
UINT ui_ObjectHandle,
BYTE b_ObjectDataLength,
PBYTE pb_ObjectData))

Parameter:
VOID v_UserReceiveFunction : The received object is passed on to this routine after reception.

Task:

This function installs the user routine for the object to be received of the type PA7200_RECEIVE_USER_ROUTINE_OBJECT.
The object is not stored.
This allows immediate data processing.

IMPORTANT!
This function is only available if you have defined the objects as PA7200_RECEIVE_USER_ROUTINE_OBJECT in the object configuration file.
No screen display possible, because this function is triggered through interrupt.

When several boards are run, this variable *b_BoardHandle* gives the handle of the board which has received the object

v_User_ReceiveFunction: Name the User routine
b_BoardHandle : Number of the **PA 7200** handle which has received the object.
b_objectHandle : Object handle
b_objectDataLength : Data length code (0 to 8)
pb_objectData : Message of the received objects

IMPORTANT!
This function can only be called up once.

Return value:
0: No error
-1: handle parameter of the board is wrong
-2: No received object is defined as PA7200_RECEIVE_USER_ROUTINE_OBJECT.
-3: User routine already installed.
-4: No runtime installed

4)i_PA7200_SetUserRemoteObjectFunction (...)

IMPORTANT!
This function can only be called up once.

Syntax:
<Return value> = i_PA7200_SetUserRemoteObjectFunction
(VOID v_User_RemoteFunction
(BYTE b_BoardHandle,
UINT ui_ObjectHandle,
BYTE b_ObjectDataLength,
PBYTE pb_ObjectData))

Parameter:
VOID v_UserRemoteFunction: The received object is passed on to this routine after reception.

Task:
This function installs the user routine for the object to be received of the type PA7200_REMOTE_USER_ROUTINE_OBJECT.
The object is not stored.
This allows immediate data processing.

IMPORTANT!
This function is only available if you have defined the received objects as PA7200_REMOTE_USER_ROUTINE_OBJECT in the object configuration file.
No screen display possible, because this function is triggered through interrupt.

When several boards are run, this variable *b_BoardHandle* gives the handle of the board which has received the object.

v_User_RemoteFunction	:	Name of the user routine
b_BoardHandle	:	Number of the **PA 7200** handle, which has received the object.
b_objectHandle	:	Handle of the objects
b_objectDataLength	:	Data length code (0 to 8)
pb_objectData	:	Messages of the received objects

IMPORTANT!
This function is not avalailable for Visual Basic Dos.

Return value:
0: No error
-1: Handle parameter of the board is wrong
-2: No received object is defined as PA7200_REMOTE_USER_ROUTINE_OBJECT.
-3: User routine already installed.
-4: No runtime installed

5)i_PA7200_SetUserErrorHandler (...)

IMPORTANT!
This function can only be called up once.

Syntax:
<Return value> = i_PA7200_SetUserErrorHandler
(VOID v_UserErrorHandler, BYTE b_ErrorNumber,
BYTE b_BoardHandle,
UINT ui_objectHandle))

Parameter:

VOID v_UserErrorHandler:	The received object is passed on to this routine after reception.

Task:
This function installs the user routine of the error handler. The error handler is called up when errors occur in the CAN layer 2 interface.

Error messages:
1: FIFO for received FIFO objects overflowed
2: FIFO for remote FIFO objects overflowed
3: FIFO for automatically-processed remote objects overflowed

When several boards are operated, the variable *b_BoardHandle* give the handle of the board which has received the message.

v_User_ErrorHandler	:	Name of the user routine
b_ErrorNumber	:	Number of the error message
b_BoardHandle	:	Number of the CAN interface having caused the error message.

IMPORTANT!
This function is not avalailable for Visual Basic Dos.

Return Value:

0: No error
-1: Handle parameter of the board is wrong
-3: User routine already installed
-4: No runtime installed

6) i_PA7200_GetIdentifier (...)

Syntax:

<Return value> = i_PA7200_GetIdentifier
(VOID v_UserErrorHandler, BYTE b_BoardHandle,
UINT ui_objectHandle
PUINT pui_Identifier))

Parameter:

BYTE	b_BoardHandle	: Handle of the CAN interface which caused the error message.
UINT	ui_ObjectHandle	: Object handle
PUINT	pui_Identifier	: Object identifier

Task:

This function supplies, on the base of an object handle (ui_objectHandle), the corresponding object identifier (pui_Identifier).

Return value:

-1 : Handle parameter of the board is wrong
-2 : No object with the given handle is available
-3 : No runtime installed

7.4.2 Receiving objects through FIFO

IMPORTANT!
This functions are only available if you have defined the objects to be received as PA7200_RECEIVE_FIFO_OBJECT in the object configuration file.
The FIFO can store 50 objects.

1)i_PA7200_TestReceiveFIFO (...)

Syntax:
<Return value> = i_PA7200_TestReceiveFIFO
(BYTE b_BoardHandle,
PINT pi_FIFOLength)

Parameter:
BYTE b_BoardHandle : Handle of the **PA 7200**
PINT pi_FIFOLength : Amount of the messages stored in the FIFO
0: No object in the FIFO
>0: Amount of objects in the FIFO.

Task:
Checks whether the receiving memory of the FIFO contains objects.

Return value:
0: No error
-1: handle parameter of the board is wrong
-2: No object to be received is defined as PA7200_RECEIVE_FIFO_OBJECT in the object configuration file.
-3: No runtime installed

2) i_PA7200_ReadDataAndHandleFromFIFO (...)

Syntax:
<Return value> = i_PA7200_ReadDataAndHandleFromFIFO
(BYTE b_BoardHandle,
PUINT ui_ObjectHandle
PBYTE pb_ObjectDataLength,
PBYTE pb_objectData)

Parameter:
BYTE b_BoardHandle : Handle of the **PA 7200**
PUINT pui_objectHandle : Handle of the objects to be received
PBYTE pb_objectDataLength : Data length code (0 to 8)
PBYTE pb_objectData: Message of objects to be received

Task:
Reads an object out of the FIFO memory. Supplies handle, message and Data length code of the objects out of the FIFO memory.

Return value:
-1: Handle parameter of the board is wrong
-2: No received object is defined as PA7200_RECEIVE_FIFO_OBJECT in the object configuration file.
-3: No objects in the FIFO
-4: No runtime installed
>0: Amount of available objects

7.4.3 Receiving objects through array

IMPORTANT!
This function is only available if you have defined the received objects as PA7200_SIMPLE_RECEIVE_OBJECT_ARRAY or as PA7200_SEARCH_RECEIVE_OBJECT_ARRAY in the object configuration file.

1) i_PA7200_ReadDataAndHandleFromArray (...)

Syntax:
```
<Return value> =  i_PA7200_ReadDataAndHandleFromArray
                              (BYTE    b_BoardHandle,
                               PUINT   pui_ObjecHandle
                               PBYTE   pb_ObjectDataLength,
                               BYTE    b_objectDataLength,
                               PBYTE   pb_objectData)
```

Parameter:

BYTE	b_BoardHandle	: Handle of the **PA 7200**
INT	i_ArrayIndex	: Index in the receiving array out of which the the objects are read
PUINT	i_ObjectHandle	: Handle of the objects to be received
PBYTE	b_objectDataLength	: Data length code (0 to 8)
PBYTE	pb_objectData	: Messages of the objects to be received

Task:
Reads a memory cell (object) of the receiving array which is selected through i_ArrayIndex. This memory cell contains the object handle, the object message and Data length code.

Return value:
-1: Handle parameter of the board is wrong.
-2: No object defined as
PA7200_SIMPLE_RECEIVE_OBJECT_ARRAY or as
PA7200_SEARCH_RECEIVE_OBJECT_ARRAY
in the OCF is contained in this index.
-3: Index number, wrong parameter.
-4: The messages can not be read properly because they are constantly actualized
-5: No runtime installed
0: New objects written in this memory cell since the last function call-up.
1: No new objects written in this memory cell since the last function call-up. The old objects have been read again.

2) i_PA7200_SearchLastChangeInArray (...)

Syntax:
<Return value> = i_PA7200_SearchLastChangeInArray
(BYTE b_BoardHandle,
PINT pi_ArrayIndex)

Parameter:
BYTE b_BoardHandle : Handle of the **PA 7200**
PINT pi_ArrayIndex : Index of the last alteration in the receiving array.
-1: No alteration
≥ 0: Index number

Task:
Supplies the index of the last alteration in the receiving array through the variable "pi_ArrayIndex". If no alteration has occurred since the last function call-up then "pi_ArrayIndex" is set to -1.

IMPORTANT!
This function is only available if you have defined the received objects as PA7200_SEARCH_RECEIVE_OBJECT_ARRAY in the object configuration file.

Return value:
0: No error
-1: handle parameter of the board is wrong.
-2: No object to be received is defined as
PA7200_SEARCH_RECEIVE_OBJECT_ARRAY.
-5: No runtime installed

7.4.4 Transmitting objects throught the object handle

IMPORTANT!
This function is only available if transmitted objects have been defined as PA7200_WRITE_OBJECT in the object configuration file (OCF).

1) i_PA7200_WriteDataPerObjectHandle (...)

Syntax:

```
<Return value> = i_PA7200_WriteDataPerObjectHandle
                                (BYTE    b_BoardHandle,
                                 BYTE    b_objectHandle,
                                 PBYTE   pb_WriteData)
```

Parameter:

BYTE b_BoardHandle : Handle of the **PA 7200**
BYTE b_objectHandle : Handle of the objects to be transmitted
PBYTE pb_WriteData : Messages to be transmitted

Task:
Writes an object on the CAN interface. The identifier, the Data length code and the RTR bit of the objects is read out of the OCF.

Return value:
0: No error
-1: handle parameter of the board is wrong
-2: The object is not defined as PA7200_WRITE_OBJECT in the object configuration file.
-3: CAN is not ready for transmission
-4: No runtime installed

7.4.5 Transmitting objects through the identifier

IMPORTANT!
This function is only available if you have defined the objects to be transmitted as PA7200_WRITE_OBJECT in the object configuration file (OCF).

1) i_PA7200_WriteDataPerIdentifier (...)

Syntax:
<Return value> = i_PA7200_WriteDataPerObjectHandle
(BYTEb_BoardHandle,
UINT ui_Identifier,
BYTE b_DataLength,
BYTE b_RTR,
PBYTE pb_WriteData)

Parameter:

BYTE	b_BoardHandle	: Handle of the **PA 7200**
UINT	ui_Identifier	: Identifier value. (See chapt. 7.1.1)
BYTE	b_DataLength	: Data length code
BYTE	b_RTR	: Defines the RTR bit
PBYTE	pb_WriteData	: Messages to be transmitted

Task:
Writes an object on the CAN interface. The number of the object is given by the parameter. The identifier is given by the parameter ui_Identifier.

Return value:
0: No error
-1: Handle parameter of the board is wrong
-2: CAN is not ready for transmission
-3: No runtime installed

7.4.6 Automatic processing of remote objects

IMPORTANT!
This functions are only available if you have defined the objects as PA7200_REMOTE_USER_AUTO_OBJECT in the object configuration file (OCF).

1) i_PA7200_TestAndWriteRemote (...)

Syntax:
<Return value> = i_PA7200_TestAndWriteRemote (BYTE b_BoardHandle)

Parameter:
BYTE b_BoardHandle : Handle of the **PA 7200**

Task:
Checks whether a remote object has been received.
If yes, the corresponding object is transmitted out of the remote array.
This object is stored by the function "i_PA7200_RemoteBufferUpdate"
in the remote array.

Return value:
-1: Handle parameter of the board is wrong
-2: No remote object is defined as PA7200_REMOTE_AUTO_OBJECT in the OCF.
-3: No remote object has been received.
-4: No runtime installed

2) i_PA7200_RemoteBufferUpdate (...)

Syntax:
<Return value> = i_PA7200_RemoteBufferUpdate (BYTE b_BoardHandle,
UINT ui_RemoteArrayIndex,
UINT ui_Identifier,
BYTE b_DataLenght,
PBYTE pb_SendBuffer)

Parameter:

BYTE	b_BoardHandle	: Handle of the **PA 7200**
UINT	ui_RemoteArrayIndex	: Memory index of the remote array, from which data is transmitted.
UINT	ui_Identifier	: Identifier value. (See chapter 7.1.1)
BYTE	b_DataLenght	: Data lenght
PBYTE	pb_SendBuffer	: Data to be transmitted

Task:
Updates the remote array in index "*ui_RemoteArrayIndex*"

Return value:
0: No error
-1: Handle parameter of the board is wrong
-2: No remote object is defined as PA7200_REMOTE_AUTO_OBJECT in the OCF.
-3: No runtime installed

7.4.7 Receiving remote objects through FIFO

IMPORTANT!
These functions are only available if you have defined objects as PA7200_REMOTE_FIFO_OBJECT in the object configuration file (OCF).

1) i_PA7200_TestRemoteFIFO (...)

Syntax:
<Return value> = i_PA7200_TestRemoteFIFO (BYTE b_BoardHandle,
PUINT pui_FIFOLength)

Parameter:
BYTE b_BoardHandle : handle of the **PA 7200**
PUINT pui_FIFOLength : Amount of objects contained in the FIFO
0: No objects in the FIFO
> 0: Amount of objects contained in the FIFO.

Task: Checks whether the FIFO remote memory contains objects.

Return value:
0: No error
-1: Handle parameter of the board is wrong
-2: No objects to be received is defined as PA7200_RECEIVE_FIFO_OBJECT in the OCF.
-3: No runtime installed

2) i_PA7200_ReadRemoteobjectFromFIFO (...)

Syntax:
<Return value> = i_PA7200_ReadRemoteFromFIFO
(BYTE b_BoardHandle,
PUINT i_ObjectHandle
PBYTE b_objectDataLength,
PBYTE pb_objectData)

Parameter:
BYTE b_BoardHandle : handle of the **PA 7200**
PUINT i_ObjekHandle : handle of the remote objects
PBYTE b_objectDataLength : Data length code (0 to 8)
PBYTE pb_objectData : Message of the remote objects

Task:

Reads a remote object out of the FIFO memory. Supplies handle, messages and Data length code out of the FIFO memory.

Return value:
-1: Handle parameter of the board is wrong
-2: No remote object defined as PA7200_REMOTE_FIFO_OBJECT in the OCF.
-3: No objects in the FIFO
-4: No runtime installed
> 0: Amount of objects contained in the FIFO.