CAN-CBX-AI814

8 A/D-Converter-Inputs, 14 Bit

Manual

to Product C.3020.xx

esd electronic system design gmbh
Vahrenwalder Str. 207 • 30165 Hannover • Germany
www.esd-electronics.com • Fax: 0511/37 29 8-68
Phone: 0511/37 29 80 • International: +49-5 11-37 29 80

N O T E

The information in this document has been carefully checked and is believed to be entirely reliable. **esd** makes no warranty of any kind with regard to the material in this document, and assumes no responsibility for any errors that may appear in this document. **esd** reserves the right to make changes without notice to this, or any of its products, to improve reliability, performance or design.

esd assumes no responsibility for the use of any circuitry other than circuitry which is part of a product of **esd** gmbh.

esd does not convey to the purchaser of the product described herein any license under the patent rights of **esd** gmbh nor the rights of others.

esd electronic system design gmbh
Vahrenwalder Str. 207
30165 Hannover
Germany

Phone:	+49-511-372 98-0
Fax:	+49-511-372 98-68
E-mail:	info@esd-electronics.com
Internet:	www.esd-electronics.com

USA / Canada:
esd electronics Inc.
525 Bernardston Road
Suite 1
Greenfield, MA 01301
USA

Phone:	+1-800-732-8006
Fax:	+1-800-732-8093
E-mail:	us-sales@esd-electronics.com
Internet:	www.esd-electronics.us

Document-File:	I:\texte\Doku\MANUALS\CAN\CBX\AI814\Englisch\CAN-CBX-AI814_13.en9
Date of print:	05.07.2006

PCB version:	from CAN-CBX-AI814 Rev. 1.0
Firmware version:	from Rev. 1.8

Changes in the chapters

The changes in the document listed below affect changes in the hardware and firmware as well as changes in the description of facts only.

Chapter	Changes versus previous version
-	First English version

Technical details are subject to change without further notice.

This page is intentionally left blank.

Contents

1. Overview

1.1 Description of the Module

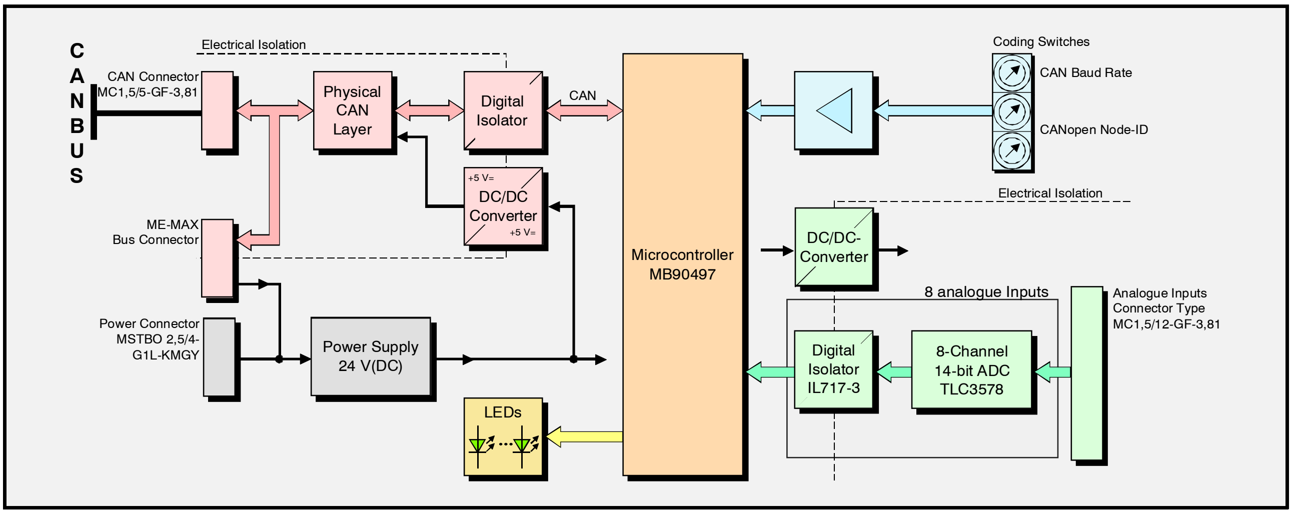


Fig. 1: Block circuit diagram of the CAN-CBX-AI814 module

The CAN-CBX-AI814 module is equipped with a MB90F497 microcontroller, which buffers CAN data into a local SRAM. The firmware is held in the flash. Parameters are stored in a serial EEPROM.

The 14-bit A/D-converter, a TLC3578 converts the eight analogue inputs.
The inputs are connected via a 12-pole screw/plug connector. The analogue inputs are electrically isolated by digital isolators for the protection of the other components.

The power supply voltage and the CAN-bus connention can be fed via the In-Rail bus connector, integrated in the top hat rail or via separate plugs.

The ISO 11898 compliant CAN interface allows a maximum data transfer rate of 1 Mbit/s. The CAN interface is electrically isolated by a dual digital isolator and a DC/DC converter.

The CANopen node number and the CAN bit rate can be configured via three coding switches.

2. Technical Data

2.1 General technical Data

Power supply voltage	nominal voltage: 24 V/DC input voltage range: 24 V ±20% current consumption (24 V, 20 ˚C): ca. 42 mA
Connectors	X100 (4-pin COMBICON connector with spring-cage connection) - 24V-power supply voltage X101 (5-pin ME-MAX-TBUS-connector, Phoenix Contact) - CAN interface and power supply voltage via In-Rail bus X500 (12-pin Mini-COMBICON connector) - analogue inputs X400 (5-pin Mini-COMBICON connector) - CAN interface Only for test and programming purposes: X200 (6-pin COMBICON connector) - the connector is placed inside the case
Temperature range	-20 ˚C ... +70 ˚C ambient temperature
Humidity	max. 90%, non-condensing
Dimensions	width: 2.2 cm, height: 11.2 cm, depth: 11.3 cm (including mounting rail fitting and connector projection)
Weight	approx. 125 g

Table 1: General technical data

2.2 CPU-Unit

CPU	16 bit µC MB90F497
RAM	2 Kbyte integrated
Flash	64 Kbyte integrated
EEPROM	minimum 256 byte

Table 2: Microcontroller

2.3 CAN Interface

Number	1
Connection	5-pin COMBICON with spring-cage connection or via Phoenix Contact TBUS-connector (In-Rail-Bus)
CAN Controller	MB90F497, CAN 2.0A/B, (CANopen software supports only 11-bit CAN identifier)
Electrical isolation of CAN interfaces against other units	via dual digital isolator (ADUM120BR) and DC/DC-converters
Physical layer CAN	physical layer according to ISO 11898, transfer rate programmable from 10 Kbit/s up to 1 Mbit/s
Bus termiantion	has to be set externally

Table 3: Data of the CAN interface

2.4 Analogue Inputs

Number	8 A/D-converter channels
Converter-Type	TLC3578
Resolution	14 bit
Input voltage rage	±10.24 V
Conversion time	max. 200 ksps
Electrical isolation against other units	via dual digital isolator

Table 4: Data of analogue inputs

2.5 Software Support

The firmware of the module supports the CANopen standards DS-301 and DS-401.

2.6 Order Information

Type	Features	Order No.
CAN-CBX-AI814	CAN-CBX-AI814 8 analogue inputs, 14 bit, including 1x CAN-CBX-TBUS (C.3000.01)	C.3020.02
Manuals		
CAN-CBX-AI814-ME	Manual in English 1*)	C.3020.21
CAN-CBX-AI814-ENG	Engineering manual in Englisch 2*) Content: Circuit diagrams, PCB top overlay drawing, datesheets of significant components	C.3020.25
Accessories:		
CAN-CBX-TBUS	Mounting-rail bus connector of the CBX-In-Rail bus for CAN-CBX-modules, (a bus connector is included in delivery of the CAN-CBX module)	C.3000.01
CAN-CBX-TBUS-connector	Terminal plug of the CBX-In-Rail bus for the connection of the +24V power supply voltage and the CAN interface	C.3000.02

1*) If module and manual are ordered together, the manual is free of charge.
2*) This manual is liable for costs, please contact our support.

Table 5: Order information

3. Hardware Installation

3.1 Connecting Diagram

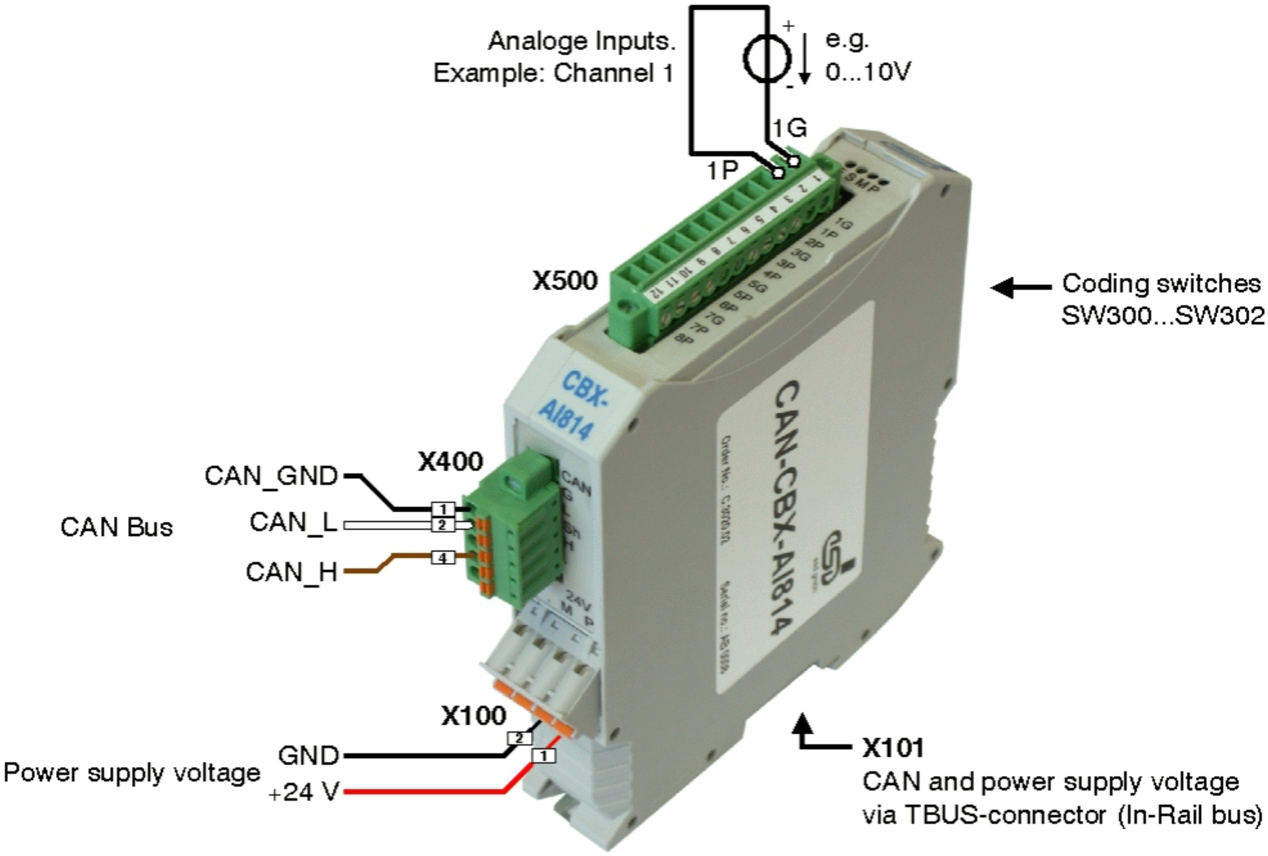


Fig. 2: Connections of the CAN-CBX-AI814 module

The connector pin assignment can be found on page 19 and following.

3.2 LED Display

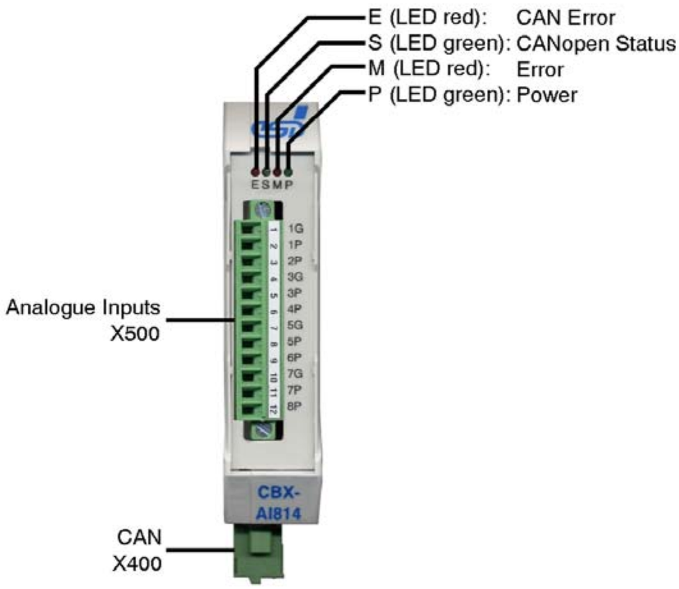


Fig. 3: Position of the LEDs in the front panel

The CAN-CBX-AI814 module is equipped with 4 status LEDs.
The terms of the indicator states of the LEDs are chosen in accordance with the CANopen Standard DS 303-3, V 1.2 (chapter 3.1). They are described in the following chapters.

3.2.1 Indicator States

In principle there are 8 indicator states distinguished:

Indicator state	Display
on	LED constantly on
off	LED constantly off
blinking	LED blinking with a frequency of approx. 2.5 Hz
flickering	LED flickering with a frequency of approx. 10 Hz
1 flash	LED 200 ms on, 1400 ms off
2 flashes	LED 200 ms on, 200 ms off, 200 ms on 1000 ms off
3 flashes	LED 2x (200 ms on, 200 ms off) + 1x (200 ms on, 1000 ms off)
4 flashes	LED 3x (200 ms on, 200 ms off) + 1x (200 ms on, 1000 ms off)

Table 6: Indicator states

The red LED flashes opposite in phase with the green LED.

3.2.2 Operation of the CAN-Error LED

LED indication				Display function	
Label	Name	Colour	Component No.	Indicator state	Description
E	CAN Error	red	200A	off	no error
				1 flash	CAN controller is in *Error Active* state
				on	CAN controller state is *Bus Off*
				2 flashes	Heartbeat or Nodeguard error occurred. The LED automatically turns off, if Nodeguard/Heartbeat-messages are received again.

Table 7: Indicator states of the red CAN Error-LED

3.2.3 Operation of the CANopen-Status LED

LED indication				Display function	
Label	Name	Colour	Component No.	Indicator state	Description
S	CANopen Status	green	200B	blinking	*Pre-operational*
				on	*Operational*
				1 flash	*Stopped*
				3 flashes	Module is in bootloader mode

Table 8: Indicator states of the CANopen Status-LED

3.2.4 Operation of the Error-LED

LED indication				Display function	
Label	Name	Colour	Component No.	Indicator state	Description
M	Error	red	200C	off	no error
				on	CAN Overrun Error The sample rate is set so high, that the firmware is not able to transmit all data on the CAN bus.
				2 flashes	Internal software error e.g.: - stored data have an invalid checksum therefore default values are loaded - internal watchdog has triggered - indicator state is continued until the module resets or an error occurs at the outputs.

Table 9: Indicator state of the Error-LED

3.2.5 Operation of the Power-LED

LED indication				Display function	
Label	Name	Colour	Component No.	Indicator state	Description
P	Power	green	200D	off	no power supply voltage
				on	power supply voltage is on

Table 10: Indicator state of the Power-LED

3.2.6 Special Indicator States

The indicator states described in the following table are indicated by the four status LEDs together:

LED indication	Description
- red CAN-Error-LED is on - all other LEDs are off	Invalid Node-ID: The coding switches for the Node-ID are set to an invalid ID-value, the module is stopped

Table 11: Special Indicator States

3.3 Coding Switches

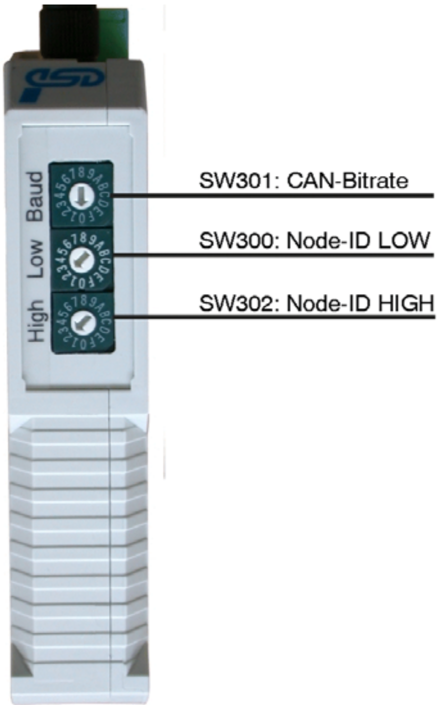


Abb. 4: Position of the coding switches

> **Attention!**
> At the moment the module is switched 'on', the state of the coding switches is determined. Changes of the settings therefore have to be made **before switching on** the module, because changes of the settings are not determined during operation.

After a reset (e.g. NMT reset) the settings are read again.

3.3.1 Setting the Node-ID via Coding Switch

The address range of the CAN-CBX-module can be set *decimal* from 1 to 127 or *hexadecimal* from 01_h to $7F_h$.

The four higher-order bits (higher-order nibble) can be set with coding switch **HIGH**, the four lower-order bits can be set with coding switch **LOW**.

3.3.2 Setting the Baud Rate

The baud rate can be set with the coding switch **Baud**.

Values from 0_h to F_h can be set via the coding switch. The values of the baud rate can be taken from the following table:

Setting [Hex]	Bit rate [Kbit/s]
0	1000
1	666.6
2	500
3	333.3
4	250
5	166
6	125
7	100
8	66.6
9	50
A	33.3
B	20
C	12.5
D	10
E	reserved
F	reserved

Table 12: Index of the baud rate

3.4 Installation of the Module Using Optional In-Rail Bus Connector

If the CAN bus signals and the power supply voltage shall be fed via the In-Rail bus, please proceed as follows:

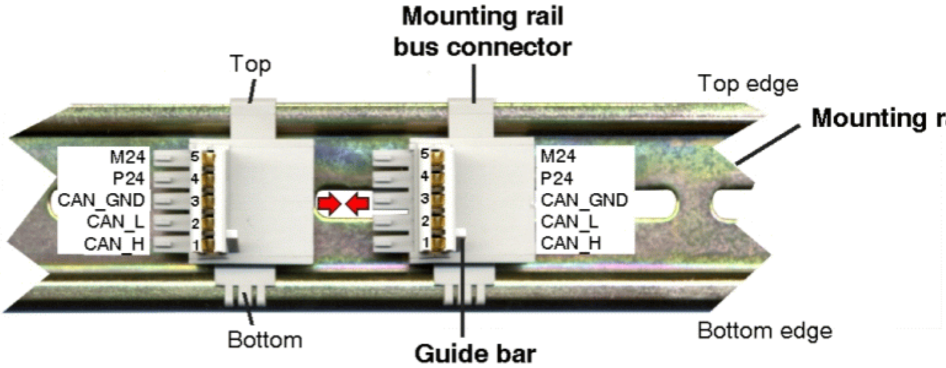


Figure 5: Mounting rail with bus connector

1. Position the In-Rail bus connector on the mounting rail and snap it onto the mounting rail using slight pressure. Plug the bus connectors together to contact the communication and power signals (in parallel with one). The bus connectors can be plugged together before or after mounting the CAN-CBX modules.

2. Place the CAN-CBX module with the DIN rail guideway on the top edge of the mounting rail.

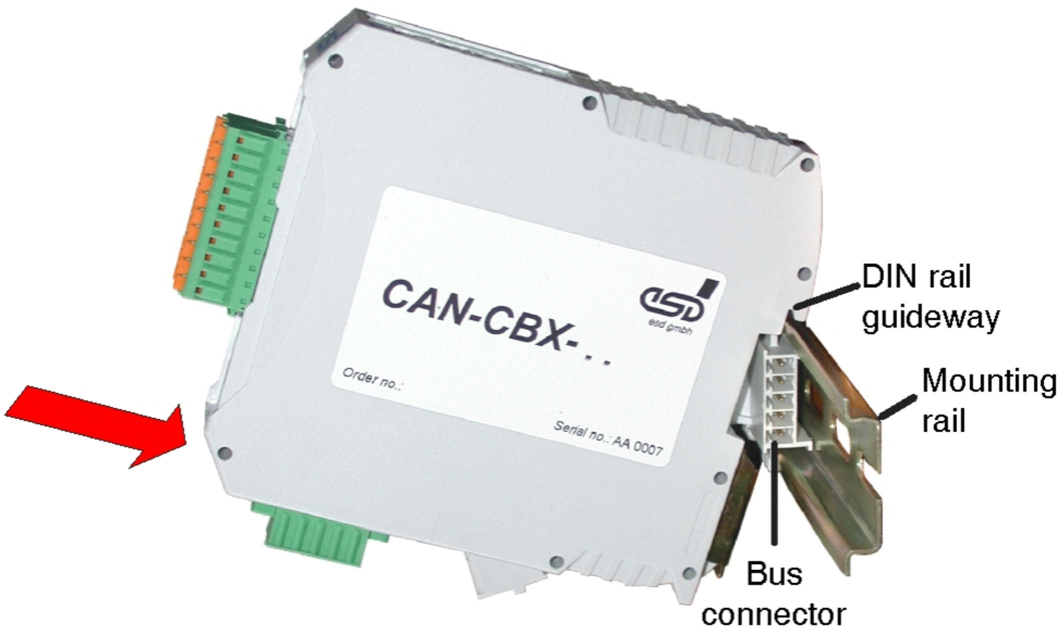


Figure 6 : Mounting CAN-CBX modules

3. Swivel the CAN-CBX module onto the mounting rail in pressing the module downwards according to the arrow as shown in figure 6. The housing is mechanically guided by the DIN rail bus connector.

4. When mounting the CAN-CBX module the metal foot catch snaps on the bottom edge of the mounting rail. Now the module is mounted on the mounting rail and connected to the In-Rail bus via the bus connector. Connect the bus connectors and the In-Rail bus if not already done.

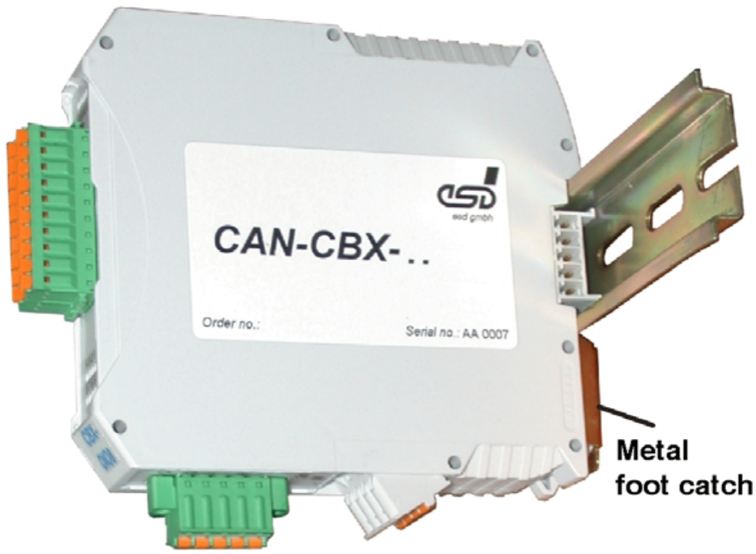


Figure 7: Mounted CAN-CBX module

3.4.1 Connecting Power Supply and CAN-Signals to CBX-In-Rail-Bus

To connect the power supply and the CAN-signals via the In-Rail bus, a terminal plug (order no.:C.3000.02) is needed. The terminal plug is not included in delivery and must be ordered separately (see order information).

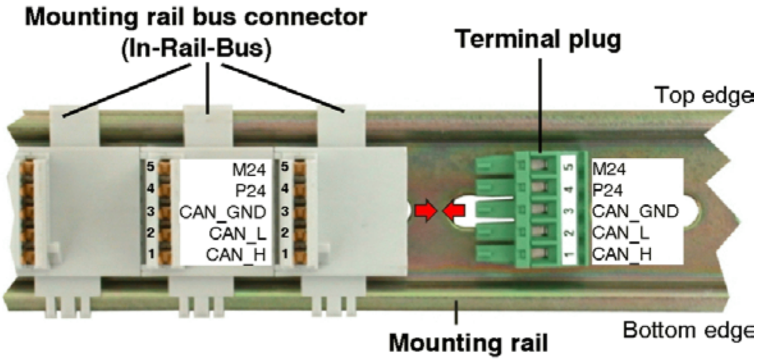


Fig. 8: Mounting rail with In-Rail bus and terminal plug

Plug the terminal plug into the socket on the right of the mounting-rail bus connector of the In-Rail bus, as described in Fig. 8. Then connect the CAN interface and the power supply voltage via the terminal plug.

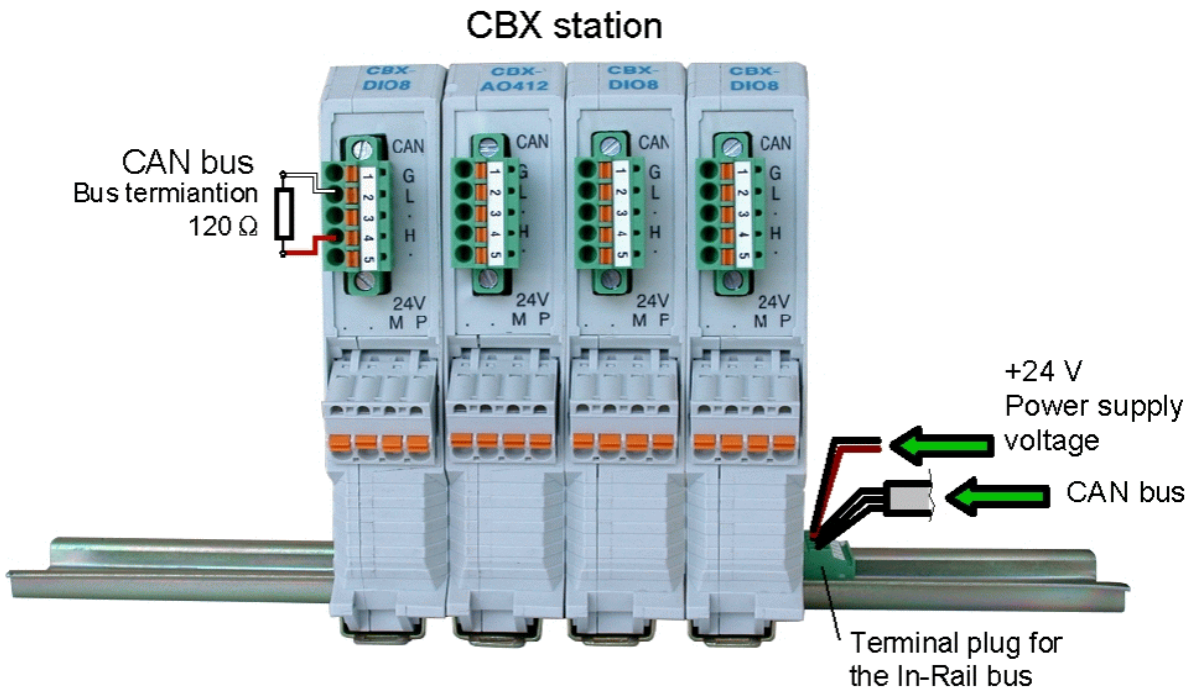


Fig. 9: CAN-CBX station with terminal plug

A bus termination must be connected to the CAN connector of the CAN-CBX module at the end of the CBX station (see Fig. 9), if the CAN bus ends there.

Hardware-Installation

Generally the CAN signals can be fed via the CAN connector of the first CAN-CBX module of the CBX station. The signals are then connected through the CAN-CBX station via the In-Rail bus. To lead through the CAN-signals the CAN bus connector of the last CAN-CBX module of the CAN-CBX station (see Fig. 10) has to be used. The CAN connectors of the CAN-CBX modules which are not at the ends of the CAN-CBX station must not be connected to the CAN bus, because this would lead to incorrect branching.

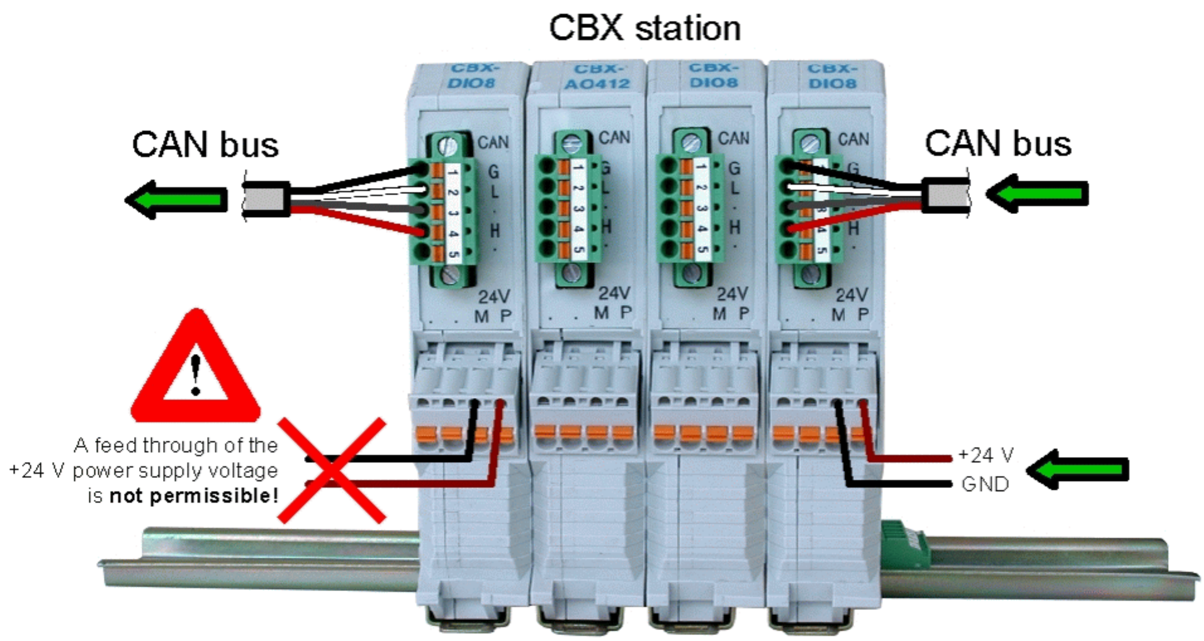


Fig. 10: Connecting the CAN signals through the CAN-CBX station

Attention!
A feed through of the power supply voltage is not permissible! A feed through of the +24 V power supply voltage can cause damage on the CBX modules.

3.5 Remove the CAN-CBX Module from the Optional In-Rail Bus

If the CAN-CBX module is connected to the In-Rail bus please proceed as follows:

Release the module from the mounting rail in moving the foot catch (see figure 7) downwards (e.g. with a screwdriver). Now the module is detached from the bottom edge of the mounting rail and can be removed.

Note:
It is possible to remove entire individual devices from the whole without interrupting the In-Rail bus connection, because the contact chain will not be interrupted.

4. Description of the Units

4.1 CAN Interface

An 82C251 is used as driver unit. The differential CAN bus signals are electrically isolated from the other signals via a dual digital converter (ADUM120BR) and a DC/DC converter.

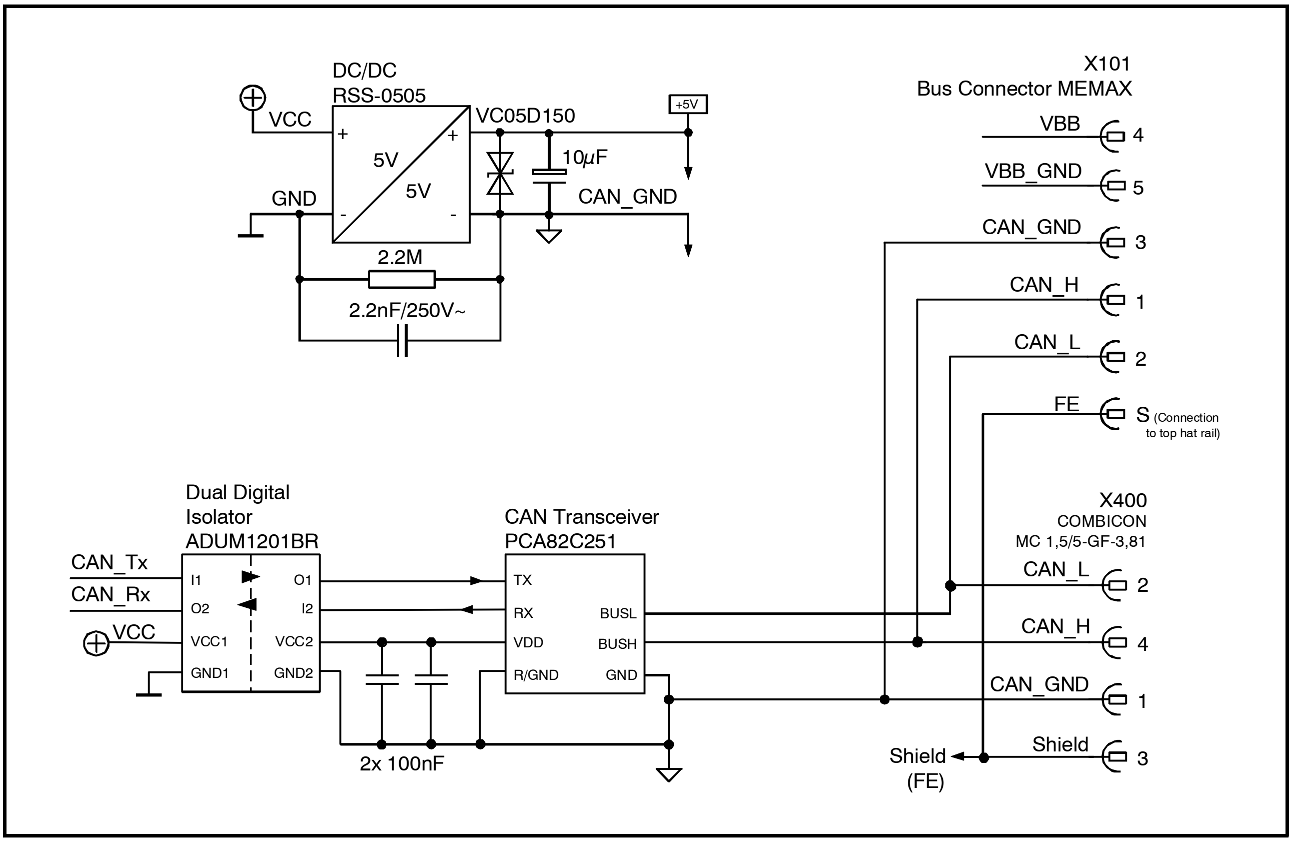


Fig. 11: CAN Interface

4.2 Analogue Inputs

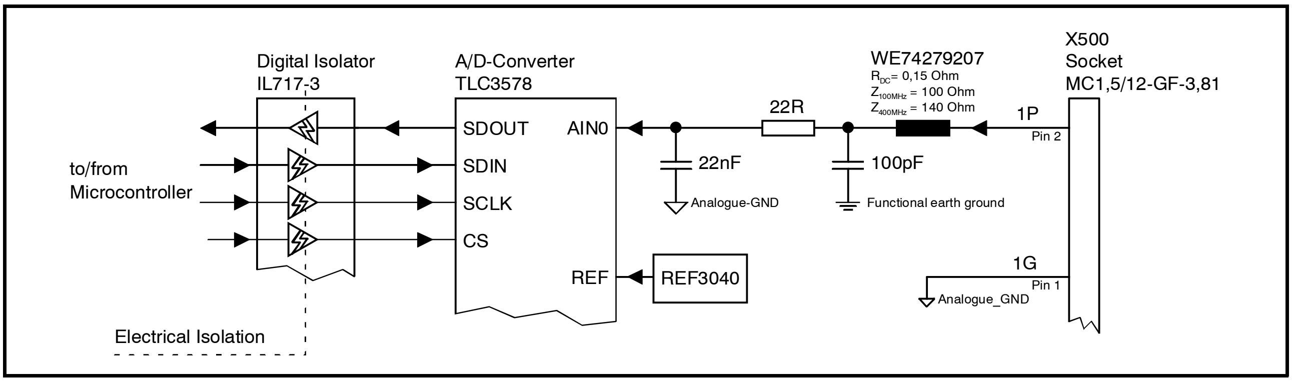


Fig. 12: Analogue input circuit (example: channel 1)

5. Connector Assignment

5.1 Power Supply Voltage X100

Device connector: COMBICON MSTBO 2,5/4-G1L-KMGY
Line connector: COMBICON FKCT 2,5/4-ST, 5.0 mm pitch, spring-cage connection,
PHOENIX-CONTACT order No.: 19 21 90 0 (included in delivery)

Pin Position:

Pin Assignment:

Pin	4	3	2	1
Signal	P24 (+ 24 V)	M24 (GND)	M24 (GND)	P24 (+ 24 V)

Please refer also to the connecting diagram on page 8.

Note: The pins 1 and 4 are connected to each other at the PCB.
The pins 2 and 3 are connected to each other at the PCB.

Signal Description:

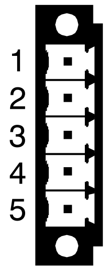
P24... power supply voltage +24 V
M24... reference potential

5.2 CAN-Bus X400

Device Connector: COMBICON MC 1,5/5-GF-3,81
Line Connector: COMBICON FK-MCP 1,5/5-STF-3,81 (spring-cage connection, (included in delivery)

Pin Position:

(Illustration of device connector)



Pin-Assignment:

Pin	Signal
1	CAN_GND
2	CAN_L
3	Shield
4	CAN_H
5	-

Signal description:

CAN_L, CAN_H ... CAN signals
CAN_GND ... reference potential of the local CAN physical layer
Shield ... pin for line shield connection (using hat rail mounting direct contact to the mounting rail potential)
- ... not connected

Recommendation of an adapter cable from 5-pin Combicon (here line connector FK-MCP1,5/5-STF-3,81 with spring-cage-connection) to 9-pin DSUB:

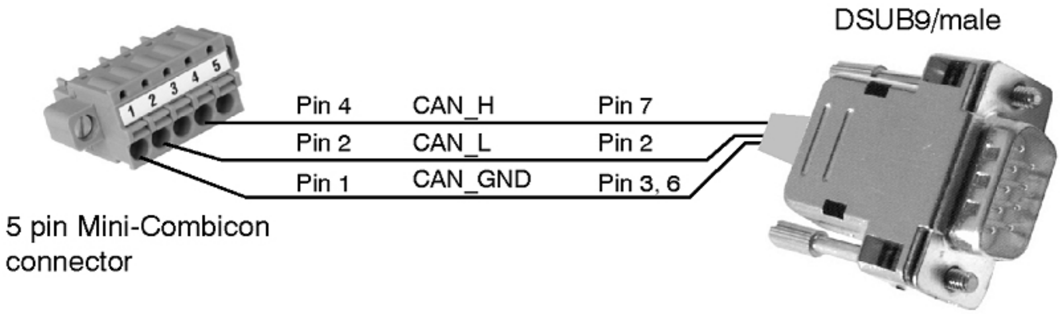
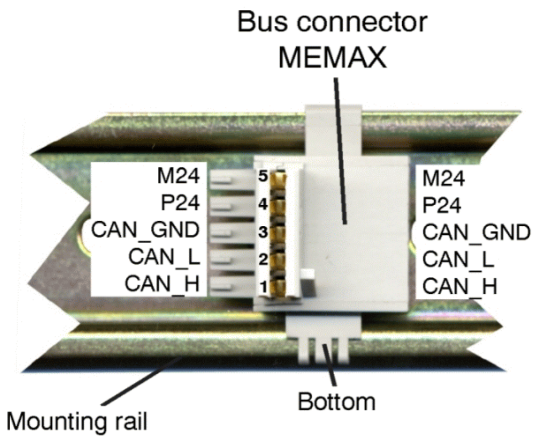


Figure 13: Assignment of the 9-pin DSUB-connector according to CiA DS 102.

5.3 CAN and Power Supply Voltage via In-Rail Bus Connector X101

Connector type: Bus connector MEMAX
ME 22,5 TBUS 1,5/5-ST-3,81 KMGY

Pin Position:



Pin Assignment:

Pin	Signal
5	M24 (GND)
4	P24 (+24 V)
3	CAN_GND
2	CAN_L
1	CAN_H
S	FE (PE_GND)

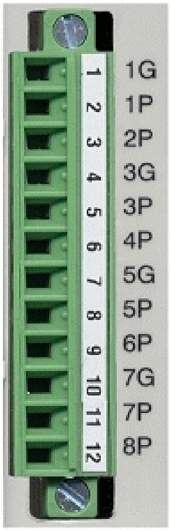
Signal Description:

CAN_L,
CAN_H ... CAN signals
CAN_GND ... reference potential of the local CAN-Physical layers
P24... power supply voltage +24 V
M24... reference potential
FE... functional earth contact (EMC)(connected to mounting rail potential)

5.4 Analogue Inputs X500

Device's socket: COMBICON MC 1,5/12-GF-3,81
Line connector: COMBICON MC 1,5/12-STF-3,81 (screw-connection)
(included in delivery)

Pin Position:



Signal description:

*x*P... Positive input pin of the analogue input
*x*G ... Reference potential analogue ground (connected to functional earth contact of the module)

(x = 1, 2, ..., 8)

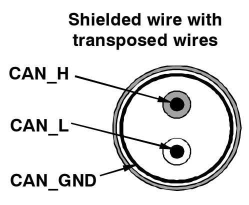
6. Correctly Wiring Electrically Isolated CAN Networks

Generally all instructions applying for wiring regarding an electromagnetic compatible installation, wiring, cross sections of wires, material to be used, minimum distances, lightning protection, etc. have to be followed.

The following **general rules** for the CAN wiring must be followed:

1.	A CAN net must not branch (exception: short dead-end feeders) and has to be terminated by the wave impedance of the wire (generally 120 W ±10%) at both ends (between the signals CAN_L and CAN_H and **not** at GND)!
2.	A CAN data wire requires **two twisted** wires and a wire to conduct the reference potential (CAN_GND)! For this the shield of the wire should be used!
3.	The reference potential CAN_GND has to be connected to the earth potential (PE) at **one** point. Exactly **one** connection to earth has to be established!
4.	The bit rate has to be adapted to the wire length.
5.	Dead-end feeders have to kept as short as possible (l < 0.3 m)!
6.	When using double shielded wires the external shield has to be connected to the earth potential (PE) at **one** point. There must be not more than **one** connection to earth.
7.	A suitable type of wire (wave impedance ca. 120 Ω ±10%) has to be used and the voltage loss in the wire has to be considered!
8.	CAN wires should not be laid directly next to disturbing sources. If this cannot be avoided, double shielded wires are preferable.

Wire structure



Signal assignment of wire and connection of earthing and terminator

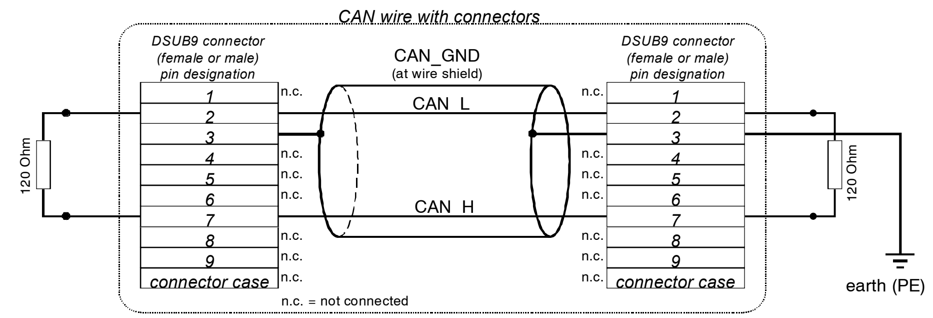


Figure: Structure and connection of wire

Wiring

Cabling

❍ for devices which have only one CAN connector per net use T-connector and dead-end feeder (shorter than 0.3 m) (available as accessory)

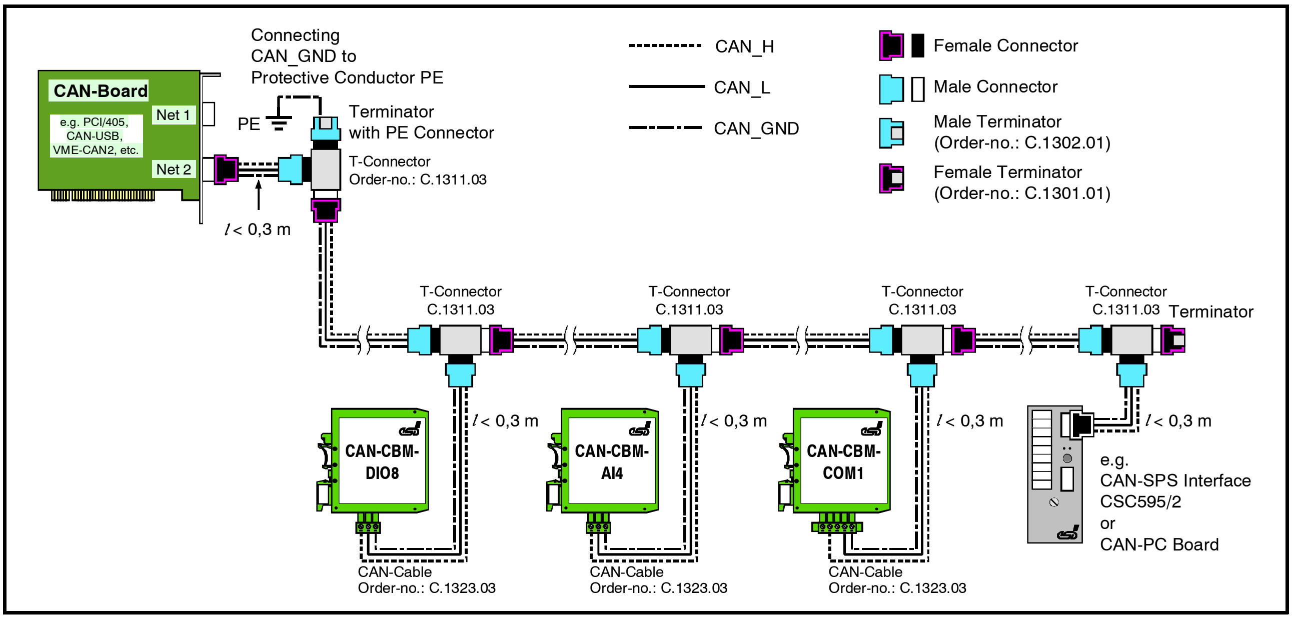


Figure: Example for correct wiring (when using single shielded wires)

Terminal Resistance

❍ use **external** terminator, because this can later be found again more easily!

❍ 9-pin DSUB-terminator with male and female contacts and earth terminal are available as accessories

Earthing

❍ CAN_GND has to be conducted in the CAN wire, because the individual esd modules are electrically isolated from each other!

❍ CAN_GND has to be connected to the earth potential (PE) at **exactly one** point in the net!

❍ each CAN user without electrically isolated interface works as an earthing, therefore: do not connect more than one user without potential separation!

❍ Earthing CAN e.g. be made at a connector

Wire Length

❍ Optical couplers are delaying the CAN signals. By using fast optical couplers and testing each board at 1 Mbit/s, however, esd CAN guarantee a reachable length of 37 m at 1 Mbit/s for most esd CAN modules within a closed net without impedance disturbances like e.g. longer dead-end feeders. (Exception: CAN-CBM-DIO8, -AI4 and AO4 (these modules work only up to 10 m with 1 Mbit/s))

Bit rate [Kbit/s]	Typical values of reachable wire length **with esd interface** l_{max} [m]	**CiA recommendations** (07/95) for reachable wire lengths l_{min} [m]
1000	37	25
800	59	50
666.6	80	-
500	130	100
333.3	180	-
250	270	250
166	420	-
125	570	500
100	710	650
66.6	1000	-
50	1400	1000
33.3	2000	-
20	3600	2500
12.5	5400	-
10	7300	5000

Table: Reachable wire lengths depending on the bit rate when using esd-CAN interfaces

Examples for CAN Wires

Manufacturer	Type of wire
U.I. LAPP GmbH Schulze-Delitzsch-Straße 25 70565 Stuttgart Germany www.lappkabel.de	e.g. UNITRONIC ®-BUS CAN UL/CSA (UL/CSA approved) UNITRONIC ®-BUS-FD P CAN UL/CSA (UL/CSA approved)
ConCab GmbH Äußerer Eichwald 74535 Mainhardt Germany www.concab.de	e.g. BUS-PVC-C (1 x 2 x 0.22 mm²) Order No.: 93 022 016 (UL appr.) BUS-Schleppflex-PUR-C (1 x 2 x 0.25 mm²) Order No.: 94 025 016 (UL appr.)
SAB Bröckskes GmbH&Co. KG Grefrather Straße 204-212b 41749 Viersen Germany www.sab-brockskes.de	e.g. SABIX® CB 620 (1 x 2 x 0.25 mm²) Order No.: 56202251 CB 627 (1 x 2 x 0.25 mm²) Order No.: 06272251 (UL appr.)

Note: Completely configured CAN wires can be ordered from **esd**.

7. CAN-Bus Troubleshooting Guide

The CAN-Bus Troubleshooting Guide is a guide to find and eliminate the most frequent hardware-error causes in the wiring of CAN-networks.

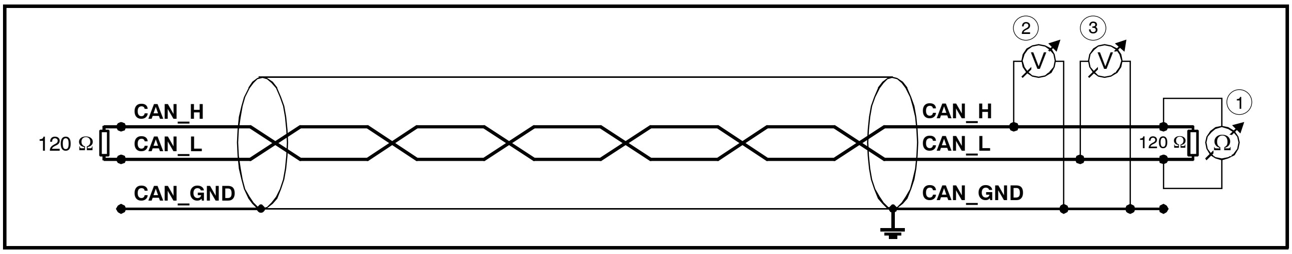


Figure: Simplified diagram of a CAN network

7.1 Termination

The termination is used to match impedance of a node to the impedance of the transmission line being used. When impedance is mismatched, the transmitted signal is not completely absorbed by the load and a portion is reflected back into the transmission line. If the source, transmission line and load impedance are equal these reflections are eliminated. This test measures the series resistance of the CAN data pair conductors and the attached terminating resistors.

To test it, please

1. Turn off all power supplies of the attached CAN nodes.
2. Measure the DC resistance between CAN_H and CAN_L at the middle and ends of the network 1 (see figure above).

The measured value should be between 50 and 70 Ω. The measured value should be nearly the same at each point of the network.

If the value is below 50 Ω, please make sure that:
- there is no short circuit between CAN_H and CAN_L wiring
- there are not more than two terminating resistors
- the nodes do not have faulty transceivers.

If the value is higher than 70 Ω, please make sure that:
- there are no open circuits in CAN_H or CAN_L wiring
- your bus system has two terminating resistors (one at each end) and that they are 120 Ω each.

7.2 CAN_H/CAN_L Voltage

Each node contains a CAN transceiver that outputs differential signals. When the network communication is idle the CAN_H and CAN_L voltages are approximately 2.5 volts. Faulty transceivers can cause the idle voltages to vary and disrupt network communication.

To test for faulty transceivers, please

1. Turn on all supplies.
2. Stop all network communication.
3. Measure the DC voltage between CAN_H and GND 2 (see figure above).
4. Measure the DC voltage between CAN_L and GND 3 (see figure above).

   Normally the voltage should be between 2.0 V and 4.0 V.
   If it is lower than 2.0 V or higher than 4.0 V, it is possible that one or more nodes have faulty transceivers. For a voltage lower than 2.0 V please check CAN_H and CAN_L conductors for continuity. For a voltage higher than 4.0 V, please check for excessive voltage.

To find the node with a faulty transceiver please test the CAN transceiver resistance (see next page).

7.3 Ground

The shield of the CAN network has to be grounded at only one location. This test will indicate if the shielding is grounded in several places.

To test it, please

1. Disconnect the shield wire from the ground.
2. Measure the DC resistance between Shield and ground.
3. Connect Shield wire to ground.

The resistance should be higher than 1 M Ω. If it is lower, please search for additional grounding of the shield wires.

7.4 CAN Transceiver Resistance Test

CAN transceivers have one circuit that controls CAN_H and another circuit that controls CAN_L. Experience has shown that electrical damage to one or both of the circuits may increase the leakage current in these circuits.

To measure the current leakage through the CAN circuits, please use an resistance measuring device and:

1. Disconnect the node from the network. Leave the node unpowered (4) (see figure below).

2. Measure the DC resistance between CAN_H and CAN_GND (5) (see figure below).

3. Measure the DC resistance between CAN_L and CAN_GND (6) (see figure below).

Normally the resistance should be between 1 M Ω and 4 M Ω or higher. If it is lower than this range, the CAN transceiver is probably faulty.

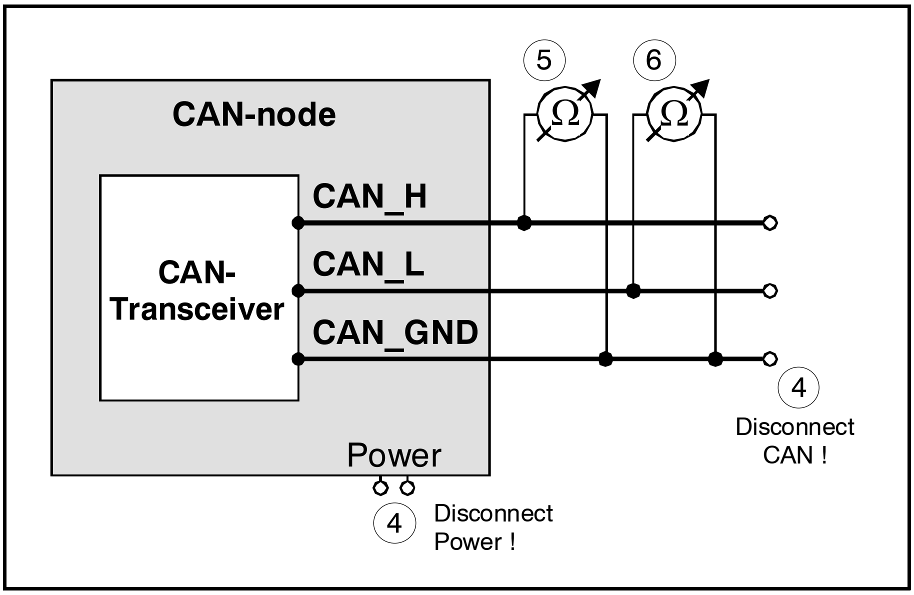


Figure: Simplified diagram of a CAN node

8. Software

Apart from basic descriptions of the CANopen, this chapter contains the most significant information about the implemented functions.

A complete CANopen description is too extensive for the purpose of this manual.
Further information can therefore be taken from the CAL/CANopen documentation ‘CiA Draft Standard 301, V 4.02’ and ‘CiA Draft Standard Proposal 401, V 2.1’.

8.1 Definition of Terms

COB ...	Communication Object
Emergency-Id...	Emergency Data Object
NMT...	Network Management (Master)
Rx...	receive
SDO...	Service Data Object
Sync...	Sync(frame) Telegram
Tx...	transmit

PDOs (Process Data Objects)

PDOs are used to transmit process data.
In the ‘Transmit’-PDO (TxPDO) the CAN-CBX-AI814 module transmits data to the CANopen network.

SDOs (Service Data Objects)

SDOs are used to transmit module internal configuration- and parameter data. In opposition to the PDOs SDO-messages are confirmed. A write or read request on a data object is always answered by a response telegram with an error index.

8.2 NMT-Boot-up

The CAN-CBX-AI814 module can be initialized with the ‘Minimum Capability Device’ boot-up as described in CiA-Draft Standard 301 in chapter 9.4.

Usually a telegram to switch from *Pre-Operational* status to *Operational* status after boot-up is sufficient. For this the 2-byte telegram ‘01_h’, ‘00_h’, for example, has to be transmitted to CAN-identifier ‘0000_h’ (= Start Remote Node all Devices).

8.3 The CANopen-Object Directory

The object directory is basically a (sorted) group of objects which can be accessed via the CAN network. Each object in this directory is addressed with a 16-bit index. The index in the object directories is represented in hexadecimal format.

The index can be a 16-bit parameter in accordance with the CANopen specification (CiA-Draft DS 301) or a manufacturer-specific code. By means of the MSBs of the index the object class of the parameter is defined.

Part of the object directory are among others:

Index [Hex]	Object	Example
0001 ... 009F	definition of data types	-
1000 ... 1FFF	Communication Profile Area	1001_h : Error register
2000 ... 5FFF	Manufacturer Specific Profile Area	2310_h : Sample rate
6000 ... 9FFF	Standardized Device Profile Area	according to Device Profile DS 40x
A000 ... FFFF	reserved	-

8.3.1 Access on the Object Directory via SDOs

The SDOs (Service Data Objects) are used to get access to the object directory of a device.
An SDO therefore represents a ‘channel’ to access the parameter of the device. The access via this channel can be made in CAN-CBX-AI814 module state *operational* and *pre-operational*.

The SDOs are transmitted on ID ‘**600_h + NodeID**’ (request). The receiver acknowledges the parameter on ID ‘**580_h + NodeID**’ (response).

Software

An SDO is structured as follows:

Identifier	Command code	Index		Sub-index	LSB	Data field		MSB
		(low)	(high)					

Example:

600_h+ Node-ID	23_h	00	14_h	01	$7F_h$	04_h	00	00
	(write)	(Index=1400_h) (Receive-PDO-Comm-Para)		(COB-def.)	COB = $047F_h$			

Description of the SDOs:

Identifier . . . . The parameters are transmitted on ID '600_h + NodeID' (request).
The receiver acknowledges the parameters on ID '580_h + NodeID' (response).

Command code . . The command code transmitted consists among other things of the command specifier and the length. Frequently required combinations are, for instance:

$40_h = 64_{dec}$: Read Request, i.e. a parameter is to be read
$23_h = 35_{dec}$: Write Request with 32-bit data, i.e. a parameter is to be set

The CAN-CBX-AI814 module responds to every received telegram with a response telegram. This can contain the following command codes:

$43_h = 67_{dec}$: Read Response with 32 bit data, this telegram contains the parameter requested
$60_h = 96_{dec}$: Write Response, i.e. a parameter has been set successfully
$80_h = 128_{dec}$: Error Response, i.e. the CAN-CBX-AI814 module reports a communication error

Frequently Used Command Codes
The following table summarizes frequently used command codes. The command frames must always contain 8 data bytes. Notes on the syntax and further command codes can be found in CiA DS 301, chapter “Service Data Object”.

Command	Number of data bytes	Command code [Hex]
Write Request (Initiate Domain Download)	1 2 3 4	2F 2B 27 23
Write Response (Initiate Domain Download)	-	60
Read Request (Initiate Domain Upload)	-	40
Read Response (Initiate Domain Upload)	1 2 3 4	4F 4B 47 43
Error Response (Abort Domain Transfer)	-	80

Index, Sub-Index . Index and sub-index will be described in the chapters “Device Profile Area” and “Manufacturer Specific Objects” of this manual.

Data Field . . . . . . The data field has got a size of a maximum of 4 bytes and is always structured ‘LSB first, MSB last’. The least significant byte is always in ‘Data 1’. With 16-bit values the most significant byte (bits 8...15) is always in ‘Data 2’, and with 32-bit values the MSB (bits 24...31) is always in ‘Data 4’.

Error Codes of the SDO Domain Transfer

The following error codes might occur (according to CiA DS 301, chapter “Abort SDO Transfer Protocol”):

Abort code [Hex]	Description
0x05040001	wrong command specifier
0x06010002	wrong write access
0x06020000	wrong index
0x06040041	object can not be mapped to PDO
0x06060000	access failed due to an hardware error
0x06070010	wrong number of data bytes
0x06070012	service parameter too long
0x06070013	service parameter too small
0x06090011	wrong sub-index
0x06090030	transmitted parameter is outside the accepted value range
0x08000000	undefined cause of error
0x08000020	data cannot be transferred or stored in the application
0x08000022	data cannot be transferred or stored in the application because of the present device state
0x08000024	access to flash failed

8.4 Overview of used CANopen-Identifiers

Function	Identifier [Hex]	Description
Network management	0	NMT
SYNC	80	Sync to all, (configurable via object 1005_h)
Emergency Message	80 + *NodeID*	configurable via object 1014_h
Tx-PDO2	280 + *NodeID*	PDO2 from CAN-CBX-AI814 (Tx) (object 1801_h)
Tx-PDO3	380 + *NodeID*	PDO3 from CAN-CBX-AI814 (Tx) (object 1802_h)
Tx-SDO	580 + *Node-ID*	SDO from CAN-CBX-AI814 (Tx)
Rx-SDO	600 + *Node-ID*	SDO from CAN-CBX-AI814 (Rx)
Node Guarding	700 + *NodeID*	configurable via object $100E_h$

NodeID: CANopen address [1_h...$7F_h$]

8.4.1 Setting the COB-ID

The COB-IDs which can be set (except the one of SYNC), are deduced initially from the setting of the Node-ID via the coding switches (see page 11). If the COB-IDs are set via SDO, this setting is valid even if the coding switches are set to another Node-ID after that.
To accept the Node-ID from the coding switches again, the *Comm defaults* or all defaults have to be restored (object 1011_h)

8.5 Default PDO-Assignment

PDOs (Process Data Objects) are used to transmit process data. The PDO mapping can be changed. The following tables show the default mapping at delivery of the module:

PDO	CAN Identifier	Length	Transmission direction	Assignment
TxPDO1	n.a.	n.a.	n.a.	TxPDO1 is not used
TxPDO2	280_h + Node-ID	8 byte	from CAN-CBX-AI814 (Tx/Transmit PDO)	A/D values channel 1 to 4 as 16 bit-values
TxPDO3	380_h + Node-ID	8 byte	from CAN-CBX-AI814 (Tx/Transmit PDO)	A/D values channel 5 to 8 as 16 bit values

Tx-PDO2 (CAN-CBX-AI814 ->)

CAN Identifier: 280_h + Node-ID

Byte	0	1	2	3	4	5	6	7
Parameter	*Analogue_Input_16-Bit_1*		*Analogue_Input_16-Bit_2*		*Analogue_Input_16-Bit_3*		*Analogue_Input_16-Bit_4*	

Tx-PDO3 (CAN-CBX-AI814 ->)

CAN-Identifier: 380_h + Node-ID

Byte	0	1	2	3	4	5	6	7
Parameter	*Analogue_Input_16-Bit_5*		*Analogue_Input_16-Bit_6*		*Analogue_Input_16-Bit_7*		*Analogue_Input_16-Bit_8*	

8.6 Reading the Analogue Values

8.6.1 Messages of the Analogue Inputs

The transmission types for the analogue inputs are described in the following:

- *acyclic, synchronous*: The transmission is initiated if a SYNC-message has been received (PDO-transmission type 0) and data has changed.
- *cyclic, synchronous*: The transmission is initiated if a defined number of SYNC-messages have been received (PDO-transmission type 1...240).
- *synchronous, remote request*: The state of the inputs is latched with each SYNC-message and is transmitted after the reception of a RTR-frame (PDO-transmission type 252).
- *asynchronous, remote request*: After the reception of a RTR-frame the last latched state of the inputs is transmitted (PDO-transmission type 253).
- *event controlled, asynchronous*: The transmission is initiated if the state of selected inputs has changed (PDO-transmission type 254, 255).

8.6.2 Supported Transmission Types Based on DS-301

Transmission Type	PDO-Transmission					supported by CAN-CBX-AI814
	cyclic	acyclic	synchro-nous	asynchro-nous	RTR	
0		X	X			x
1...240	X		X			x
241...251	reserved					-
252			X		X	x
253				X	X	x
254				X	X	x
255				X	X	x

8.7 Implemented CANopen-Objects

A detailed description of the objects can be taken from CiA DS-301.

Index [Hex]	Sub-index (max.) [Dec]	Description	Data type	R/W	Default value
1000	-	Device Type	unsigned 32	ro	00040191_h
1001	-	Error Register	unsigned 8	ro	00_h
1003	10	Pre-Defined-Error-Field	unsigned32	rw	00_h
1005	-	COB-ID-Sync	unsigned32	rw	80_h
1008	-	Manufacturer Device Name	visible string	ro	"CAN-CBX-AI814"
1009	-	Manufacturer Hardware Version	visible string	ro	x.yy (depending on version)
100A	-	Manufacturer Software Version	visible string	ro	x.yy (depending on version)
100C	-	Guard Time	unsigned 16	rw	0000_h
100D	-	Life Time Factor	unsigned 8	rw	00_h
100E	-	Node Guarding Identifier	unsigned 32	rw	Node-ID + 700_h
1010	3	Store Parameter	unsigned 32	rw	
1011	3	Restore Parameter	unsigned 32	rw	
1014	-	COB-ID Emergency Object	unsigned 32	rw	80_h + Node-ID
1015	-	Inhibit Time EMCY	unsigned 16	rw	00_h
1016	1	Consumer Heartbeat Time	unsigned 32	rw	00_h
1017	-	Producer Heartbeat Time	unsigned 16	rw	00_h
1018	4	Identity Object	unsigned 32	ro	Vendor Id: 00000017_h Prod. Code: 23020002_h
1029	3	Error Behaviour	unsigned 8	ro	00_h

Index [Hex]	**Sub-index** [Hex]	**Description**	**Data type**	**R/W**
1801	5	2. Transmit PDO-Parameter	PDO CommPar (20_h)	rw
1802	5	3. Transmit PDO-Parameter	PDO CommPar (20_h)	rw
1A01	2	2. Transmit PDO-Mapping	PDO Mappping (21_h)	rw
1A02	2	3. Transmit PDO-Mapping	PDO Mappping (21_h)	rw

8.7.1 Device Type (1000_h)

INDEX	**1000_h**
Name	*device type*
Data Type	unsigned 32
Access Type	ro
Default Value	0004 0191

The value of the *device type* is: 0004.0191_h (Analogue input: 0004_h digital profile number: 0191_h)

Example: Auslesen des Device Type

The CANopen master transmits the read request on identifier '603_h' (600_h + Node-ID) to the CAN-CBX-AI814 module with the module no. 3 (Node-ID=3_h):

ID	RTR	LEN	DATA							
			1	2	3	4	5	6	7	8
603_h	0_h	8_h	40_h	00_h	10_h	00_h	00_h	00_h	00_h	00_h
			Read Request	Index=1000_h		Sub Index				

The CAN-CBX-AI814 module no. 3 responds to the master by means of read response on identifier '583_h' (580_h + Node-ID) with the value of the device type:

ID	RTR	LEN	DATA							
			1	2	3	4	5	6	7	8
583_h	0_h	8_h	43_h	00_h	10_h	00_h	91_h	01_h	04_h	00_h
			Read Response	Index=1000_h		Sub Index	dig. Profile Nr.191		Analogue Input	

value of device type: 0004.0191_h

The data field is always structured following the rule 'LSB first, MSB last' (see page 33, data field).

8.7.2 Error Register (1001_h)

The CAN-CBX-AI814 module uses the error register to indicate error messages.

INDEX	**1001_h**
Name	*error register*
Data type	unsigned 8
Access type	ro
Default value	0

The following bits of the error register are being supported at present:

Bit	Meaning
0	*generic*
1	-
2	*voltage*
3	-
4	*communication error* (overrun, error state)
5	-
6	-
7	-

Bits which are not supported (-) are always returned as ‘0’. If an error is active, the according bit is set to ‘1’.

The following messages are possible:

00_h no errors
01_h generic error
04_h voltage error
10_h communication error

8.7.3 Pre-defined Error Field (1003_h)

INDEX	**1003_h**
Name	*pre-defined error field*
Data Type	unsigned 32
Access Type	ro
Default Value	No

The *pre-defined error field* provides an error history of the errors that have occurred on the device and have been signalled via the Emergency Object.
Sub-index 0 contains the current number of errors stored in the list.
Under sub-index 1 the last error which occurred is stored. If a new error occurs, the previous error is stored under sub-index 2 and the new error under sub-index 1, etc. In this way a list of the error history is created.
The error buffer is structured like a ring buffer. If it is full, the oldest entry is deleted for the latest entry.

This module supports a maximum of 10 error entries. When the 11th error occurs the oldest error entry is deleted. In order to delete the entire error list, sub-index '0' has to be set to '0'. This is the only permissible write access to the object.

With every new entry to the list the module transmits an **Emergency Frame** to report the error.

Index [Hex]	Subindex [Dec]	Description	Value range [Hex]	Default	Data type	R/W
1003	0	*no_of_errors_in_list*	0, 1...10	-	unsigned 8	rw
	1	*error-code n*	0...FFFFFFFF	-	unsigned 32	ro
	2	*error-code (n-1)*	0...FFFFFFFF	-	unsigned 32	ro
	:	*:*	:	:	:	ro
	10	*error-code (n-9)*	0...FFFFFFFF	-	unsigned 32	ro

Meaning of the variables:

no_of_errors_in_list - contains the number of error codes currently on the list
n = number of error which occurred last
- in order to delete the error list this variable has to be set to '0'
- if *no_of_errors_in_list* $\neq 0$, the error register (Object 1001_h) is set

error-code x The 32-bit long error code consists of the CANopen-emergency error code described in DS 301, Table 21 and the error code defined by esd (manufacturer-specific error field).

Bit:	31 16	15 0
Contents:	*manufacturer-specific error field*	*emergency-error-code*

manufacturer-specific error field: for CAN-CBX-AI814 always ‘00’, unless *emergency-error-code* = 2300_h (see below)

emergency-error-code: The following error-codes are supported:

8110_h - CAN overrun error
- Sample rate is set to high, thus the firmware is not able to transmit all data to the CAN bus.

8120_h - CAN in error passive mode
8130_h - Lifeguard error / heartbeat error
8140_h - Recovered from “Bus Off”
6000_h - Software error:
-EEPROM checksum error (no transmission of this error message as emergency message)
6110_h - Internal software error
e.g.:
- checksum of saved data is invalid and default values are loaded
- internal watchdog has triggered

$FF10_h$ - Data lost (A/D-data overflow)
5000_h - Hardware error (e.g. A/D-converter defective)

Emergency Message

The data otf the emergency frame transmitted by the CAN-CBX-AI814 have the following structure:

Byte:	0	1	2	3	4	5	6	7
Inhalt:	*emergency-error-code* (siehe oben)		*error-register* 1001_h	*no_of_errors_in_list* $1003,00_h$	-			

An emergency message is transmitted, if an error occurs. If this error occurs again, no further emergency message is generated.
If the last error message disappears, again an emergency message is transmitted to indicate the error disappearance.

8.7.4 COB-ID of SYNC-Message (1005_h)

INDEX	**1005_h**
Name	*COB-ID SYNC message*
Data Type	unsigned 32
Access Type	rw
Default Value	80_h

Structure of the parameter:

Bit-No.	Value	Meaning
31 (MSB)	-	do not care
30	0/1	0: Device does not generate SYNC message 1: Device generates SYNC message
29	0	always 0 (11-bit ID)
28...11	0	always 0 (29-bit IDs are not supported)
10...0 (LSB)	x	Bit 0...10 of the SYNC-COB-ID

The identifier can take values between 0...$7FF_h$.

8.7.5 Manufacturer's Device Name (1008_h)

INDEX	**1008_h**
Name	*manufacturer's device name*
Data Type	visible string
Default Value	string: 'CAN-CBX-AI814'

For detailed description of the Domain Uploads, please refer to CiA DS 202-2 (CMS-Protocol Specification).

8.7.6 Manufacturer's Hardware Version (1009$_h$)

INDEX	1009$_h$
Name	*manufacturer's hardware version*
Data Type	visible string
Default Value	string: z.B. '1.0'

The hardware version is read similarly to reading the manufacturer's device name via the domain upload protocol. Please refer to CiA DS 202-2 (CMS-Protocol Specification) for a detailed description of the upload.

8.7.7 Manufacturer's Software Version 100A$_h$

INDEX	100A$_h$
Name	*manufacturer's software version*
Data Type	visible string
Default Value	string: z.B.: '1.2'

Reading the software version is similar to reading the manufacturer's device name via the domain upload protocol. Please refer to CiA DS 202-2 (CMS-Protocol Specification) for a detailed description of the upload.

8.7.8 Guard Time ($100C_h$) und Life Time Factor ($100D_h$)

The CAN-CBX-AI814 module supports the node guarding or alternatively the heartbeat function (see page 55).

Note:
On the recommendation of the CiA, the heartbeat-function shall be used preferentially. Use the node-guarding only for exisiting systems and not for new developments!

Guard time and life time factors are evaluated together. Multiplying both values will give you the life time. The guard time is represented in milliseconds.

INDEX	**$100C_h$**
Name	*guard time*
Data Type	unsigned 16
Access Type	rw
Default Value	0 [ms]
Minimum Value	0
Maximum Value	$FFFF_h$ (65.535 s)

INDEX	**$100D_h$**
Name	*life time factor*
Data Type	unsigned 8
Access Type	rw
Default Value	0
Minimum Value	0
Maximum Value	FF_h

8.7.9 Node Guarding Identifier ($100E_h$)

The module only supports 11-bit identifiers.

INDEX	**$100E_h$**
Name	*node guarding identifier*
Data Type	unsigned 32
Access Type	rw
Default Value	700_h + Node-ID

Structure of the parameter *node guarding identifier* :

Bit-No.	Meaning
31 (MSB) 30	reserved
29...11	always 0, because 29-bit-IDs are not supported
10...0 (LSB)	bit 0...10 of the node guarding identifier

The identifier can take values between 1...$7FF_h$.

8.7.10 Store Parameters (1010_h)

This object supports saving of parameters to the EEPROM.

In order to avoid storage of parameters by mistake, storage is only executed when the specific signature as shown below is transmitted.
Reading the index returns information about the implemented storage functionalities (refer to CiA DS 301 for more information).

INDEX	**1010_h**
Name	*store parameters*
Data Type	unsigned 32

Index [Hex]	Sub-index	Description	Value range	Data type	R/W
1010	0	*number_of_entries*	4_h	unsigned 8	ro
	1	*save_all_parameters* (objects 1000_h ... $9FFF_h$)	no default, write: 65 76 61 73_h (= ASCII: ‘e’ ‘v’ ‘a’ ‘s’)	unsigned 32	rw
	2	*save_communication_parameter* (objects 1000_h ... $1FFF_h$)		unsigned 32	rw
	3	*save_application_parameter* (objects 6000_h ... $9FFF_h$)		unsigned 32	rw
	4	*save_manufacturer_parameter* (objects 2000_h ... $5FFF_h$)		unsigned 32	rw

Parameters which can be saved or loaded:

Communication parameter of the objects 1005_h ... 1029_h

Application parameter of the objects 6421_h ... 6426_h

Manufacturer specific parameter of the objects 2310_h, 2311_h, 2401_h ... 2405_h

8.7.11 Restore Default Parameters (1011_h)

Via this command the default parameters, valid when leaving the manufacturer, are activated again. Every individual setting stored in the EEPROM will be lost. Only command ‘Restore all Parameters’ is being supported.

In order to avoid restoring of default parameters by mistake, restoring is only executed when a specific signature as shown below is transmitted.
Reading the index provides information about its parameter restoring capability (refer to CiA DS 301 for more information).

INDEX	**1011_h**
Name	*restore default parameters*
Data Type	unsigned 32

Index [Hex]	Subindex	Description	Value range	Data type	R/W
1011	0	*number_of_entries*	4	unsigned 8	ro
	1	*load_all_default_parameters* (objects 1000_h ... $9FFF_h$)	no default, write: 64 61 6F $6C_h$ (= ASCII: ‘d’ ‘a’ ‘o’ ‘l’)	unsigned 32	rw
	2	*load_communication_parameter* (objects 1000_h ... $1FFF_h$)		unsigned 32	rw
	3	*load_application_parameter* (objects 6000_h ... $9FFF_h$)		unsigned 32	rw
	4	*load_manufacturer_parameter* (objects 2000_h ... $5FFF_h$)		unsigned 32	rw

Parameters which can be saved or loaded:

Communication parameter of the objects 1005_h ... 1029_h

Application parameter of the objects 6421_h ... 6426_h

Manufacturer specific parameter of the objects 2310_h, 2311_h, 2401_h ... 2405_h

8.7.12 COB_ID Emergency Message (1014_h)

INDEX	**1014_h**
Name	*COB-ID emergency object*
Data Type	unsigned 32
Default Value	80_h + Node-ID

This object defines the COB-ID of the emergency object (EMCY).

The structure of this object is shown in the following table:

Bit-No.	Value	Meaning
31 (MSB)	0/1	0: EMCY exists / is valid 1: EMCY does not exist / EMCY is not valid
30	0	reserved (always 0)
29	0	always 0 (11-bit ID)
28...11	0	always 0 (29-bit IDs are not supported)
10...0 (LSB)	x	bits 0...10 of COB-ID

The identifier can take values between 0...$7FF_h$.

8.7.13 Inhibit Time EMCY (1015_h)

INDEX	**1015_h**
Name	*inhibit_time_emergency*
Data Type	unsigned 16
Access Type	rw
Value Range	0-$FFFF_h$
Default Value	0

The *Inhibit Time* for the EMCY message can be adjusted via this entry. The time is determined as a multiple of 100 µs.

8.7.14 Consumer Heartbeat Time (1016_h)

INDEX	1016_h
Name	*consumer heartbeat time*
Data Type	unsigned 32
Default Value	No

The heartbeat function can be used for mutual monitoring of the CANopen modules (especially to detect connection failures). Unlike node guarding/life guarding the heartbeat function does not require RTR-Frames.

Function:
A module, the so-called heartbeat producer, cyclically transmits a heartbeat message on the CAN-bus on the node-guarding identifier (see object $100E_h$). One or more heartbeat consumer receive the message. It has to be received within the heartbeat time stored on the heartbeat consumer, otherwise a heartbeat event is triggered on the heartbeat-consumer module. A heartbeat event generates a heartbeat error on the CAN-CBX-AI814 module.

Each module can act as a heartbeat producer and a heartbeat consumer. The CAN-CBX-AI814 module supports maximum one heartbeat producer per CAN net.

Index [Hex]	Subindex	Description	Value range [Hex]	Default	Data type	Index [Hex]
1016	0	*number_of_entries*	1	1	unsigned 8	ro
	1	*consumer-heartbeat_time*	0...007FFFFF	0	unsigned 32	rw

Meaning of the variable *consumer-heartbeat_time_x*:

	consumer-heartbeat_time_x		
Bit	31 24	23 16	15 0
Assignment	reserved (always ‘0’)	*Node-ID* (unsigned 8)	*heartbeat_time* (unsigned 16)

Node-ID Node-Id of the heartbeat producer to be monitored.

heartbeat_time Cycle time of heartbeat producer to transmit the heartbeat on the node-guarding ID (see object $100E_h$).
The consumer-heartbeat time of the monitoring module must always be higher than the producer-heartbeat time of the heartbeat-transmitting module.

Example:

consumer-heartbeat_time = $0031\ 03E8_h$

=> *Node-ID* = 31_h = 49_d
=> *heartbeat time* = $3E8_h$ = 1000_d => 1 s

8.7.15 Producer Heartbeat Time (1017_h)

INDEX	**1017_h**
Name	*producer heartbeat time*
Data Type	unsigned 16
Default Value	0 ms

The producer heartbeat time defines the cycle time with which the CAN-CBX-AI814 module transmits a heartbeat-frame to the node-guarding ID.

If the value of the producer heartbeat time is higher than ‘0’, it is active and stops the node-/ life-guarding (see page 47).
If the value of the producer-heartbeat-time is set to ‘0’, transmitting heartbeats by this module is stopped.

Index [Hex]	Sub-index	Description	Value range [Hex]	Default	Data type	R/W
1017	0	*producer-heartbeat_time*	0...FFFF	0 ms	unsigned 16	rw

producer-heartbeat_time Cycle time of heartbeat producer to transmit the heartbeat on the node-guarding ID (see object $100E_h$).
The consumer-heartbeat time of the monitoring module must always be higher than the producer-heartbeat time of the heartbeat-transmitting module.

8.7.16 Identity Object (1018_h)

INDEX	**1018_h**
Name	*identity object*
Data Type	unsigned 32
Default Value	No

This object contains general information to the CAN module.

Index [Hex]	Subindex	Description	Value range [Hex]	Default	Data type	R/W
1018	0	*no_of_entries*	4	4	unsigned 8	ro
	1	*vendor_id*	0...FFFFFFFF	0000 0017_h	unsigned 32	ro
	2	*product_code*	0...FFFFFFFF	2302 0002_h	unsigned 32	ro
	3	*revision_number*	0...FFFFFFFF	0	unsigned 32	ro
	4	*serial_number*	0...FFFFFFFF	-	unsigned 32	ro

Description of the variables:

vendor_id This variable contains the esd-vendor-ID. This is always 00000017_h.

product_code Here the esd-article number of the product is stored.
Example:
Value '2302 0002_h' corresponds to article number 'C.3020.02'.

revision_number Here the software version is stored. In accordance with DS 301 the two MSB represent the revision numbers of the major changes and the two LSB show the revision number of minor corrections or changes.

revision_no	
major_revision_no 31 16	*minor_revision_no* 15 0

MSB LSB

serial_number Here the serial number of the hardware is read. The first two characters of the serial number are letters which designate the manufacturing lot. The following characters represent the actual serial number.

In the two MSB of *serial_no* the letters of the manufacturing lot are coded. They each contain the ASCII-code of the letter with the MSB set ‘1’ in order to be able to differentiate between letters and numbers:
(ASCII-Code) + 80_h = read_byte

The two last significant bytes contain the number of the module as BCD-value.

Example:
If the value ‘C1C2 0105_h’ is being read, this corresponds to the hardware-serial number code ‘AB 0105’. This value has to correspond to the serial number of the module.

8.7.17 Verify Configuration (1020_h)

INDEX	**1020_h**
Name	*verify configuration*
Data Type	unsigned 32
Default Value	No

In this object the date and the time of the last configuration can be stored to check whether the configuration complies with the expected configuration or not in the future.

Index [Hex]	Sub-index	Description	Value range [Hex]	Default	Data type	R/W
1020	0	*no_of_entries*	2	2	unsigned 8	ro
	1	*configuration_date*	0...FFFFFFFF	0	unsigned 32	rw
	2	*configuration_time*	0...FFFFFFFF	0	unsigned 32	rw

Parameter Description:

configuration_date Date of the last configuration of the module. The value is defined in number of days since the 01.01.1984.

configuration_time Time in ms since midnight at the day of the last configuration.

8.7.18 Error Behaviour Object (1029$_h$)

INDEX	**1029$_h$**
Name	*error behaviour object*
Data Type	unsigned 8
Default Value	No

If an error event occurs (such as heartbeat error), the module changes into the status which has been defined in variable *communication_error* or *output_error*.

Index [Hex]	Subindex	Description	Value range [Hex]	Default	Data type	R/W
1029	0	*no_of_error_classes*	2	2	unsigned 8	ro
	1	*communication_error*	0...2	0	unsigned 8	rw

Meaning of the variables:

Variable	Meaning
no_of_error_classes	number of error-classes (here always ‘2’)
communication_error	heartbeat/lifeguard error and *Bus off*

The module can enter the following states if an error occurs.

Variable	Module state
0	pre-operational (only if the current state is operational)
1	no state change
2	stopped

8.7.19 Object Transmit PDO Communication Parameter 1801_h, 1802_h

This objects define the parameters of the transmit-PDOs.

INDEX	**1801_h 1802_h**
Name	*transmit PDO parameter*
Data Type	PDOCommPar

Index [Hex]	Sub-index	Description	Value range [Hex]	Default	Data type	R/W
1801	0	*number_of_entries*	0...FF	5	unsigned8	ro
	1	*COB-ID used by PDO*	1...800007FF	280_h +Node-ID	unsigned32	rw
	2	*transmission type*	0...FF	255	unsigned8	rw
	3	*inhibit time*	0...FFFF	0	unsigned16	rw
	4	*reserved*	0..FF	0	unsigned8	const
	5	*event timer*	0...FFFF	0	unsigned16	rw
1802	0	*number_of_entries*	0...FF	5	unsigned8	ro
	1	*COB-ID used by PDO*	1...800007FF	380_h +Node-ID	unsigned32	rw
	2	*transmission type*	0...FF	255	unsigned8	rw
	3	*inhibit time*	0...FFFF	0	unsigned16	rw
	4	*reserved*	0..FF	0	unsigned8	const
	5	*event timer*	0...FFFF	0	unsigned16	rw

The *transmission types* 0, 1...240 and 252...255 are supported.

8.7.20 Transmit PDO Mapping Parameter 1A01$_h$, 1A02$_h$

This objects define the assignment of the transmit data to the Tx-PDOs.

INDEX	1A01$_h$, 1A02$_h$
Name	*transmit PDO mapping*
Data Type	PDO Mapping

The following table shows the assignment of the transmit PDO mapping parameters:

Index [Hex]	Subindex	Description	Value range [Hex]	Default	Data type	R/W
1A01	0	*number of entries*	0...FF	4$_h$	unsigned 8	rw
	1	*Read_Analogue_Input_16_1*	0...FFFF	6401 0110$_h$	unsigned 16	rw
	2	*Read_Analogue_Input_16_2*	0...FFFF	6401 0210$_h$	unsigned 16	rw
	3	*Read_Analogue_Input_16_3*	0...FFFF	6401 0310$_h$	unsigned 16	rw
	4	*Read_Analogue_Input_16_4*	0...FFFF	6401 0410$_h$	unsigned 16	rw
1A02	0	*number of entries*	0...FF	4$_h$	unsigned 8	rw
	1	*Read_Analogue_Input_16_5*	0...FFFF	6401 0510$_h$	unsigned 16	rw
	2	*Read_Analogue_Input_16_6*	0...FFFF	6401 0610$_h$	unsigned 16	rw
	3	*Read_Analogue_Input_16_7*	0...FFFF	6401 0710$_h$	unsigned 16	rw
	4	*Read_Analogue_Input_16_8*	0...FFFF	6401 0810$_h$	unsigned 16	rw

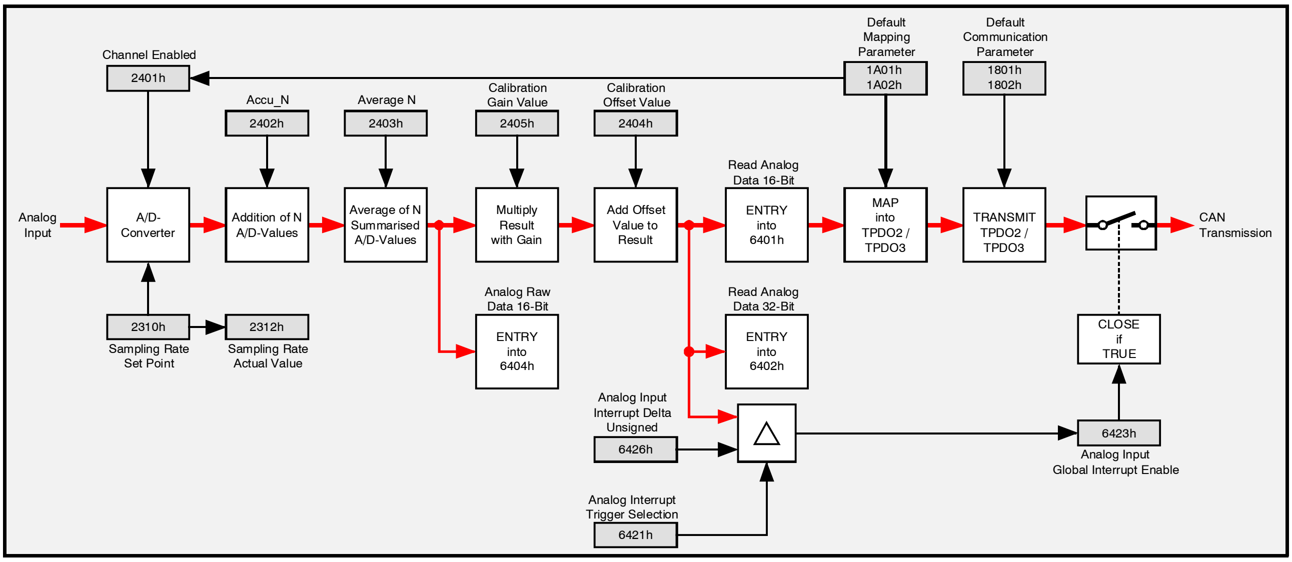
Note: The local firmware allows every TxPDO-mapping, i.e. combinations of 16-bit- and 32-bit-A/D-values in one frame are also possible.

8.8 Device Profile Area

8.8.1 Overview of the Implemented Objects 6401_h ...6426_h

Index [HEX]	Name	Data Type
6401	*Read Analogue Inputs 16-Bit*	signed16
6402	*Read Analogue Inputs 32-Bit*	signed32
6404	*Read Analogue Inputs Raw Data 16-Bit*	signed16
6421	*Analogue Interrupt Trigger Selection*	unsigned 8
6423	*Analogue Input Global Interrupt Enable*	boolean
6426	*Analogue Input Interrupt Delta Unsigned*	unsigned32

8.8.2 Relationship between the implemented analogue Input Objects



Example here: Module is in ‘Operational’ state

8.8.3 Read Input 16-Bit (6401$_h$)

Index [Hex]	Subindex [Dec]	Description	Value range [Hex]	Default [Dec]	Data type	R/W
6401	0	*Number of entries*	8	8	unsigned 8	ro
	1	*Read_Analogue_Input_16_1*	8000...7FFF	-	signed 16	ro
	2	*Read_Analogue_Input_16_2*	8000...7FFF	-	signed 16	ro
	3	*Read_Analogue_Input_16_3*	8000...7FFF	-	signed 16	ro
	4	*Read_Analogue_Input_16_4*	8000...7FFF	-	signed 16	ro
	5	*Read_Analogue_Input_16_5*	8000...7FFF	-	signed 16	ro
	6	*Read_Analogue_Input_16_6*	8000...7FFF	-	signed 16	ro
	7	*Read_Analogue_Input_16_7*	8000...7FFF	-	signed 16	ro
	8	*Read_Analogue_Input_16_8*	8000...7FFF	-	signed 16	ro

Assignment of the variable *Read_Analogue_Input_16_x* (x = 1...8):
The last digit of the name of the variable is the number of the respective analogue input channel. The 14 data bits of the A/D-converter are corrected by addition, averaging, offset and gain and then shifted left-aligned in two's complement representation in the 16-bit variable.

Calculation of the measured voltage at default setting of the objects:
The 2 LSBs are without meaning in case that no addition or averaging of the measured values is done (see object 2402$_h$ and 2403$_h$).
In the default-setting of the objects however both an addition and an averaging are made before the object 6401$_h$ is updated. All 16 bits of the object 6401$_h$ can be evaluated in this case:

Value of the variable *Read_Analogue_Input_16_x*																	Measured voltage (object 2402$_h$ = 3, object 2403$_h$ = 4)
Binary (bit 15...0)																Hexadecimal	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8000	-10.24 V
:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	FFFF	-0.3125 mV
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0001	+0.3125 mV
:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7FFF	+10.24 V - (1 $LSB_{VARIABLE}$) = 10.2397 V

Resolution of the measured value (object 2402$_h$ = 3, object 2403$_h$ = 4):
1 LSB based on the variable *Read_Analogue_Input_16_x* (x = 1...8):
1 $LSB_{VARIABLE}$ => 10.24 V / 8000$_h$ => 0.3125 mV

Calculation of the measured voltage without addition and averaging of the measured values:
If no addition or averaging of the measured values (object $2402_h = 0$ and $2403_h = 0$) is made, only the A/D-converter resolution of 14 bits is relevant. The 2 LSBs can be ignored:

Value of the variable *Read_Analogue_Input_16_x*																	Measured voltage (object $2402_h = 0$, object $2403_h = 0$)
Binary (bit 15...0)																Hexadecimal	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
1	0	0	0	0	0	0	0	0	0	0	0	0	0	x	x	8000	-10.24 V
:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
1	1	1	1	1	1	1	1	1	1	1	1	1	1	x	x	FFFC	-1.2512 mV
0	0	0	0	0	0	0	0	0	0	0	0	0	0	x	x	0000	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	x	x	0004	+1.2512 mV
:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
0	1	1	1	1	1	1	1	1	1	1	1	1	1	x	x	7FFC	+10.24 V - (1 $LSB_{AD\text{-}CONVERTER}$) = 10.2387 V

Resolution of the measured value (object $2402_h = 0$, object $2403_h = 0$):
1 LSB based on the maximum A/D-converter resolution:
1 $LSB_{AD\text{-}CONVERTER}$ => 10.24 V / 2000_h => 1.2512 mV

Formula to calcualte the returned value:

$$Read\_Analogue\_Input\_16\_x = Offset\_x + (\text{AD-value} \cdot Gain\_x)$$

Provided that the objects 2402_h and 2403_h (see page 77 and 78) are set to default values, the *AD-value* is the value, which results from addition and averaging (see figure on page 62). It is equivalent to the "Read Input Raw Data 16-Bit"-value (object 6404_h).

Offset and gain are defined in the objects *Calibration Offset Value* (object 2404_h, see page 79) and *Calibration Gain Value* (object 2405_h, see page 80).

8.8.4 Read Input 32-Bit (6402_h)

Index [Hex]	Subindex [Dec]	Description	Value range [Hex]	Default [Dec]	Data type	R/W
6402	0	*Number of entries*	8	8	unsigned 8	ro
	1	*Read_Analogue_Input_32_1*	80000000 ... 7FFF0000	-	signed 32	ro
	2	*Read_Analogue_Input_32_2*	80000000 ... 7FFF0000	-	signed 32	ro
	3	*Read_Analogue_Input_32_3*	80000000 ... 7FFF0000	-	signed 32	ro
	4	*Read_Analogue_Input_32_4*	80000000 ... 7FFF0000	-	signed 32	ro
	5	*Read_Analogue_Input_32_5*	80000000 ... 7FFF0000	-	signed 32	ro
	6	*Read_Analogue_Input_32_6*	80000000 ... 7FFF0000	-	signed 32	ro
	7	*Read_Analogue_Input_32_7*	80000000 ... 7FFF0000	-	signed 32	ro
	8	*Read_Analogue_Input_32_8*	80000000 ... 7FFF0000	-	signed 32	ro

Assignment of the variable *Read_Analogue_Input_32_x* (x = 1...8):

The last digit of the name of the variable is the number of the respective analogue input channel.
The 14 data bits of the A/D-converter are corrected by addition, averaging, offset and gain and then shifted left-aligned in two's complement representation in the 32-bit variable.

Calculation of the measured voltage at default setting of the objects:
The 18 LSBs can be ignored, if no addition or averaging of the measured values is made (see object 2402_h and 2403_h). In the default-setting of the objects however both an addition and an averaging are made before the object 6402_h is updated. All 32 bits of the object 6401_h can therefore be evaluated in this case:

Note: Only the evaluation of the higher-order 16 bits is technically usable!

Value of the variable *Read_Analogue_Input_32_x*																																	Measured voltage (object2402$_h$=3 object 2403$_h$=4)
Binary (bit 31...0)																																Hexa-decimal	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	80000000	-10.24 V
:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	FFFFFFFF	-4.7684 nV
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00000000	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	00000001	+4.7684 nV
:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7FFFFFFF	+10.24 V -(1 $LSB_{VARIABLE}$)

Resolution of the measured value (object 2402_h = 3, object 2403_h = 4):
1 LSB based on the variable *Read_Analogue_Input_32_x* (x = 1...8):
1 $LSB_{VARIABLE}$ => 10.24 V / 8000 0000_h => 4.7684 nV

Calculation of the measured voltage without addition and averaging of the values:
If no addition or averaging of the measured values (object 2402_h = 0 and 2403_h = 0) is made, only the A/D-converter resolution of 14 bits is relevant. The 18 LSBs can be ignored:

Value of the variable *Read_Analogue_Input_32_x*									Measured voltage (object 2402_h=0 object 2403_h=0)
Binary (bit 31...0)								Hexa-decimal	
31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0		
1 0 0 0	0 0 0 0	0 0 0 0	0 0 x x	x x x x	x x x x	x x x x	x x x x	80000000	-10.24 V
: : : :	: : : :	: : : :	: : : :	: : : :	: : : :	: : : :	: : : :	:	:
1 1 1 1	1 1 1 1	1 1 1 1	1 1 x x	x x x x	x x x x	x x x x	x x x x	FFFC0000	-1.2512 mV
0 0 0 0	0 0 0 0	0 0 0 0	0 0 x x	x x x x	x x x x	x x x x	x x x x	00000000	0
0 0 0 0	0 0 0 0	0 0 0 0	0 1 x x	x x x x	x x x x	x x x x	x x x x	00040000	+1.2512 mV
: : : :	: : : :	: : : :	: : : :	: : : :	: : : :	: : : :	: : : :	:	:
0 1 1 1	1 1 1 1	1 1 1 1	1 1 x x	x x x x	x x x x	x x x x	x x x x	7FFC0000	+10.24 V -(1 $LSB_{AD-CONVERTER}$) = 10.2387 V

Resolution of the measured value (object 2402_h = 0, object 2403_h = 0):
1 LSB based on the maximum A/D-converter resolution:
1 $LSB_{AD-CONVERTER}$ => 10.24 V / 2000_h => 1.2512 mV

Formula to calcualte a return value:

Read_Analogue_Input_32_x = *Offset_x* + (AD-value · *Gain_x*)

Provided that the objects 2402_h and 2403_h (see page 77 and 78) are set to default values, the *AD-value* is the value, which results from addition and averaging (see figure on page 62).

Offset and gain are defined in the objects *Calibration Offset Value* (object 2404_h, see page 79) and *Calibration Gain Value* (object 2405_h, see page 80).

8.8.5 Read Input Raw Data 16-Bit (6404_h)

Index [Hex]	Subindex [Dec]	Description	Value range [Hex]	Default [Dec]	Data type	R/W
6404	0	*Number of entries*	8	8	unsigned 8	ro
	1	*Read_Analogue_In_Raw_16_1*	8000...7FFF	-	signed 16	ro
	2	*Read_Analogue_In_Raw_16_2*	8000...7FFF	-	signed 16	ro
	3	*Read_Analogue_In_Raw_16_3*	8000...7FFF	-	signed 16	ro
	4	*Read_Analogue_In_Raw_16_4*	8000...7FFF	-	signed 16	ro
	5	*Read_Analogue_In_Raw_16_5*	8000...7FFF	-	signed 16	ro
	6	*Read_Analogue_In_Raw_16_6*	8000...7FFF	-	signed 16	ro
	7	*Read_Analogue_In_Raw_16_7*	8000...7FFF	-	signed 16	ro
	8	*Read_Analogue_In_Raw_16_8*	8000...7FFF	-	signed 16	ro

Assignment of the variable *Read_Analogue_In_Raw_16_x* (x = 1...8):
The last digit of the name of the variable is the number of the respective analogue input channel.
The 14 data bits of the A/D-converter are shifted left-aligned in the 16-bit variable without correction with the offset and gain value.

Calculation of the measured voltage at default setting of the objects:
The 2 LSBs can be ignored, if no addition or averaging of the measured values is made (see object 2402_h and 2403_h). In the default-setting of the objects however both an addition and an averaging are made before the object 6402_h is updated. All 16 bits of the object 6404_h can be evaluated in this case:

Value of the variable *Read_Analogue_In_Raw_16_x*																	Measured voltage (object 2402_h = 3, object 2403_h = 4)
Binary (bit 15...0)																Hexadecimal	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8000	-10.24 V
:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	FFFF	-0.3125 mV
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0001	+0.3125 mV
:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7FFF	+10.24 V - (1 $LSB_{VARIABLE}$) = 10.2397 V

Resolution of the measured values (Object 2402_h = 3, Object 2403_h = 4):
1 LSB based on the variable *Read_Analogue_Input_16_x* (x = 1...8):
1 $LSB_{VARIABLE}$ => 10.24 V / 8000_h => 0.3125 mV

Calculation of the measured voltage without addition and averaging of the measured values:
If no addition or averaging of the measured values (object 2402_h = 0 and 2403_h = 0) is made, only the A/D-converter resolution of 14 bits is relevant. The 2 LSBs can be ignored:

Value of the variable *Read_Analogue_Input_16_x*																	Measured voltage (object 2402_h = 0, object 2403_h = 0)
Binary (bit 15...0)																Hexadecimal	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
1	0	0	0	0	0	0	0	0	0	0	0	0	0	x	x	8000	-10.24 V
:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
1	1	1	1	1	1	1	1	1	1	1	1	1	1	x	x	FFFC	-1.2512 mV
0	0	0	0	0	0	0	0	0	0	0	0	0	0	x	x	0000	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	x	x	0004	+1.2512 mV
:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
0	1	1	1	1	1	1	1	1	1	1	1	1	1	x	x	7FFC	+10.24 V - (1 $LSB_{AD\text{-}CONVERTER}$) = 10.2387 V

<u>Resolution of the measured value (object 2402_h = 0, object 2403_h = 0):</u>
1 LSB based on the maximum A/D-converter resolution:
1 $LSB_{AD\text{-}CONVERTER}$ => 10.24 V / 2000_h => 1.2512 mV

8.8.6 Analogue Interrupt Trigger Selection (6421_h)

Index [Hex]	Subindex [Dec]	Description	Value range [Hex]	Default [Hex]	Data type	R/W
6421	0	*Number of analogue inputs*	8	8	unsigned 8	ro
	1	*Analogue_Input_IRQ_Trigger_1*	00, 04	0	unsigned 8	rw
	2	*Analogue_Input_IRQ_Trigger_2*	00, 04	0	unsigned 8	rw
	3	*Analogue_Input_IRQ_Trigger_3*	00, 04	0	unsigned 8	rw
	4	*Analogue_Input_IRQ_Trigger_4*	00, 04	0	unsigned 8	rw
	5	*Analogue_Input_IRQ_Trigger_5*	00, 04	0	unsigned 8	rw
	6	*Analogue_Input_IRQ_Trigger_6*	00, 04	0	unsigned 8	rw
	7	*Analogue_Input_IRQ_Trigger_7*	00, 04	0	unsigned 8	rw
	8	*Analogue_Input_IRQ_Trigger_8*	00, 04	0	unsigned 8	rw

This object defines the interrupt trigger condition.

Assignment of the variable *Analogue_Input_IRQ_Trigger_x* (x = 1...8):

Bit:	7	6	5	4	3	2	1	0
Meaning:	reserved for future applications			not supported		**Change of value higher than Delta (6426_h)**	not supported	

Value range:

Analogue_Input_IRQ_Trigger_x = 00 => no interrupt trigger
Analogue_Input_IRQ_Trigger_x = 04 => interrupt trigger, if the change of the A/D-value is higher than the value (Delta) defined in object 6426_h.

Note: Changes of the voltage are evaluated in positive and negative direction, although the value in object 6426_h is always entered as a positive value.

8.8.7 Global Interrupt Enable (6423_h)

Index [Hex]	Subindex [Dec]	Description	Value range [Boolean]	Default [Boolean]	Data type	R/W
6423	0	*Analogue_Input_Global_Interrupt_Enable*	true, false	false	boolean	rw

With the object *Analogue Input Global Interrupt Enable* the interrupt-function of the module is enabled or disabled. The values in object 6421_h and 6426_h remain unaffected of this.

In the default-setting the interrupts are disabled.

Value range:

Analogue_Input_Global_Interrupt_Enable = true => Global Interrupt enabled
Analogue_Input_Global_Interrupt_Enable = false => Global Interrupt disabled (default setting)

8.8.8 Analogue Input Interrupt Delta (6426_h)

Index [Hex]	Subindex [Dec]	Description	Value range [Hex]	Default [Hex]	Data type	R/W
	0	*Number_Analogue_Inputs*	8	8	unsigned8	ro
	1	*Analogue_Input_IRQ_Delta_1*	0...FFFFFFFF	0	unsigned32	rw
	2	*Analogue_Input_IRQ_Delta_2*	0...FFFFFFFF	0	unsigned32	rw
	3	*Analogue_Input_IRQ_Delta_3*	0...FFFFFFFF	0	unsigned32	rw
6426	4	*Analogue_Input_IRQ_Delta_4*	0...FFFFFFFF	0	unsigned32	rw
	5	*Analogue_Input_IRQ_Delta_5*	0...FFFFFFFF	0	unsigned32	rw
	6	*Analogue_Input_IRQ_Delta_6*	0...FFFFFFFF	0	unsigned32	rw
	7	*Analogue_Input_IRQ_Delta_7*	0...FFFFFFFF	0	unsigned32	rw
	8	*Analogue_Input_IRQ_Delta_8*	0...FFFFFFFF	0	unsigned32	rw

In this object the difference value of the analogue input voltage, which can be evaluated for triggering the interrupt, is defined. Via object 6421_h the type of evaluation is selected. The interrupt function is enabled via object 6423_h.

Positive and negative changes of the voltage are evaluated, although the value contained in object 6426_h is always a positive value.

Value range:

Value of the variable *Analogue_Input_IRQ_Delta_x* [Hex]	Change of the voltage
0	0
0000 0001	9.5367 nV
:	:
FFFF FFFF	+20.48 V

Note: For the voltage difference calculation the objects 6401_h, or 6402_h are evaluated, which are calculated from gain, offset, Accu_N (object 2402_h) and Average_N (object 2403_h).

It can be expedient, to use the lower-order bits of the 32-bit variable *Analogue_Input_IRQ_Delta_x*, because resolutions higher than 14 bit can be achieved by the addition of the measured values. However, the technical limits of the hardware have to be taken into account at this.

8.9 Manufacturer Specific Profile Area

8.9.1 Overview of Manufacturer Specifc Objects 2310_h ... 2405_h

Index [HEX]	Name	Data Type
2310	*Sampling Rate Set Point*	unsigned16
2312	*Sampling Rate Actual Value*	unsigned16
2401	*Channel Enabled*	unsigned8
2402	*Accu_N*	unsigned8
2403	*Average_N*	unsigned8
2404	*Calibration Offset Value*	signed32
2405	*Calibration Gain Value*	signed 6

8.9.2 Sample Rate (2310_h)

Index [Hex]	Subindex [Dec]	Description	Value range [Hex]	Default [Dec]	Data type	R/W
2310	0	*Number of entries*	1	1	unsigned 8	ro
	1	*Sample_Rate*	C8...FFFF	625 => 312.5 µs *)	unsigned 16	rw

Via this object the sample rate can be defined. Always all eight A/D-channels are converted.

Assignment of the Variable *Sample_Rate*:

The sample rate is subdivided in 0.5 µs steps:

Value of the variable *Sample_Rate_x* [Hex]	Sample rate
C8	100 µs
C9	100.5 µs
:	:
271	312.5 µs (default)
:	:
FFFF	32768 µs

8.9.3 Sample Rate Actual Value (2312_h)

Index [Hex]	Subindex [Dec]	Description	Value range [Hex]	Default [Dec]	Data type	R/W
2312	0	*Number of entries*	8	8	unsigned 8	ro
	1	*Sample_Rate_Actual _1*	C8...FFFF	-	unsigned 16	ro
	2	*Sample_Rate_Actual _2*	C8...FFFF	-	unsigned 16	ro
	3	*Sample_Rate_Actual _3*	C8...FFFF	-	unsigned 16	ro
	4	*Sample_Rate_Actual _4*	C8...FFFF	-	unsigned 16	ro
	5	*Sample_Rate_Actual _5*	C8...FFFF	-	unsigned 16	ro
	6	*Sample_Rate_Actual _6*	C8...FFFF	-	unsigned 16	ro
	7	*Sample_Rate_Actual _7*	C8...FFFF	-	unsigned 16	ro
	8	*Sample_Rate_Actual _8*	C8...FFFF	-	unsigned 16	ro

Via this object the actual value of the sample rate preset can be read.

Assignment of the variable *Sample_Rate_Actual_x* (x = 1...8):

The value of the variable read multiplied by 0.5 equals the sample rate [µs] in the default setting of the objects:

Value of variables *Sample_Rate_Actual_x* [Hex]	Sample rate
C8	100 µs
C9	100.5 µs
:	:
FFFF	32768 µs

8.9.4 Channel Enabled (2401_h)

Index [Hex]	Subindex [Dec]	Description	Value range	Default [Dec]	Data type	R/W
2401	0	*Number of entries*	8	8	unsigned 8	ro
	1	*Channel_Enabled_1*	false, true	-	boolean	ro
	2	*Channel_Enabled_2*	false, true	-	boolean	ro
	3	*Channel_Enabled_3*	false, true	-	boolean	ro
	4	*Channel_Enabled_4*	false, true	-	boolean	ro
	5	*Channel_Enabled_5*	false, true	-	boolean	ro
	6	*Channel_Enabled_6*	false, true	-	boolean	ro
	7	*Channel_Enabled_7*	false, true	-	boolean	ro
	8	*Channel_Enabled_8*	false, true	-	boolean	ro

Module in *operational* state:
Depending on whether the channel has been assigned to a TxPDO in the PDO mapping, the firmware decides if an A/D-channel is converted or not. Channels which are not mapped, are not converted consequently. The data of the objects 6401_h and 6402_h are not updated if no new A/D-conversions are made.

Module in *pre-operational* state:
The objects can be read by means of SDO-transfers.

If not all channels are converted, the sampling rate of the remaining channels increases.

The object *Channel Enabled* returns, whether an A/D-channel is converted or not.

Value range:

Channel_Enabled_x = false => A/D-converter channel is not “mapped”
Channel_Enabled_x = true => A/D-converter channel is “mapped”

8.9.5 Accu N (2402_h)

Index [Hex]	Subindex [Dec]	Description	Value range [Hex]	Default [Dec]	Data type	R/W
2402	0	*Number of entries*	8	8	unsigned 8	ro
	1	*Accu_Count_1*	0...8	3	unsigned 8	rw
	2	*Accu_Count_2*	0...8	3	unsigned 8	rw
	3	*Accu_Count_3*	0...8	3	unsigned 8	rw
	4	*Accu_Count_4*	0...8	3	unsigned 8	rw
	5	*Accu_Count_5*	0...8	3	unsigned 8	rw
	6	*Accu_Count_6*	0...8	3	unsigned 8	rw
	7	*Accu_Count_7*	0...8	3	unsigned 8	rw
	8	*Accu_Count_8*	0...8	3	unsigned 8	rw

This object defines, how many analogue values are to be added up.

By the appropriate pre-processing of the output value of the A/D-converter the result of this addition is always a 16-bit value in the two's complement representation.

The number of additions is calculated as:

$$\boxed{\text{number of additions} = 2^{(Accu\_Count\_x)}}$$

Up to 256 values can be added up (default setting: 8 values are added up).

Advantage: Filter with decimation, improvement of the resolution, limitation of the data rate.
Disadvantage: The mesaured value is not updated until the addition is finished.

8.9.6 Average N (2403_h)

Index [Hex]	Subindex [Dec]	Description	Value range [Hex]	Default [Dec]	Data type	R/W
2403	0	*Number of entries*	8	8	unsigned 8	ro
	1	*Average_Count_1*	0...5	4	unsigned 8	rw
	2	*Average_Count_2*	0...5	4	unsigned 8	rw
	3	*Average_Count_3*	0...5	4	unsigned 8	rw
	4	*Average_Count_4*	0...5	4	unsigned 8	rw
	5	*Average_Count_5*	0...5	4	unsigned 8	rw
	6	*Average_Count_6*	0...5	4	unsigned 8	rw
	7	*Average_Count_7*	0...5	4	unsigned 8	rw
	8	*Average_Count_8*	0...5	4	unsigned 8	rw

This object defines, how many buffered analogue values are used to calculate the moving average. With a read access on the analogue values the average of the last 2^n samples is read.
After every conversion a new average is available, because the A/D-values are buffered in a ring buffer. The number of the averaged values is calculated as:

$$\text{number of averaged values} = 2^{(Average\_Count\_x)}$$

It can be averaged from the last 1, 2, 4, 8, 16 (default) or 32 values.

Note: For input signals with a frequency $\ll F_{sample}$ the filter improves the resolution by

Average_Count /2

bits .

Furthermore a Notchfilter characteristic can be obtained with zero points at

$F_{sample}/2^k$ with k = 1 ... *Average_Count_x*.

Example:
At a sample-frequency of 200 Hz and *Average_Count* = 2 a suppression of the frequency of 100 Hz and 50 Hz can be obtained.

Advantage: After every conversion a new average is available.
Disadvantge: Higher internal calculating effort as for the addition of the measured values (see object 2402_h)

8.9.7 Calibration Offset Value (2404_h)

Index [Hex]	Subindex [Dec]	Description	Value range [Hex]	Default [Hex]	Data type	R/W
2404	0	*Number of entries*	8	8	unsigned8	ro
	1	*Calibration_Offset_1*	80000000 ... 7FFFFFFF	0	signed 32	rw
	2	*Calibration_Offset_2*	80000000 ... 7FFFFFFF	0	signed 32	rw
	3	*Calibration_Offset_3*	80000000 ... 7FFFFFFF	0	signed 32	rw
	4	*Calibration_Offset_4*	80000000 ... 7FFFFFFF	0	signed 32	rw
	5	*Calibration_Offset_5*	80000000 ... 7FFFFFFF	0	signed 32	rw
	6	*Calibration_Offset_6*	80000000 ... 7FFFFFFF	0	signed 32	rw
	7	*Calibration_Offset_7*	80000000 ... 7FFFFFFF	0	signed 32	rw
	8	*Calibration_Offset_8*	80000000 ... 7FFFFFFF	0	signed 32	rw

In this object an offset value for the correction of the A/D-value can be specified. The offset affects the objects 6401_h and 6402_h (see also figure on page 62).

Value range:

Value of the variable *Calibration_Offset_x*									Offset voltage
Binary (bit 31...0)								Hexa-decimal	
31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0		
1 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	80000000	-10.24 V
: : : :	: : : :	: : : :	: : : :	: : : :	: : : :	: : : :	: : : :	:	:
1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	FFFFFFFF	-4.7684 nV
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	00000000	0
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1	00000001	+4.7684 nV
: : : :	: : : :	: : : :	: : : :	: : : :	: : : :	: : : :	: : : :	:	:
0 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	7FFFFFFF	+10.24 V -(1 $LSB_{VARIABLE}$)

<u>Resolution of the offset value:</u>
1 LSB based on the variable *Calibration_Offset_x* (x = 1...8):
1 $LSB_{VARIABLE}$ => 10.24 V / 8000 0000_h => 4.7684 nV

8.9.8 Calibration Gain Value (2405_h)

Index [Hex]	Subindex [Dec]	Description	Value range [Hex]	Default [Hex]	Data type	R/W
	0	*Number of entries*	8	8	unsigned8	ro
	1	*Calibration_Gain_1*	8000...7FFF	0	signed 16	rw
	2	*Calibration_Gain_2*	8000...7FFF	0	signed 16	rw
	3	*Calibration_Gain_3*	8000...7FFF	0	signed 16	rw
2404	4	*Calibration_Gain_4*	8000...7FFF	0	signed 16	rw
	5	*Calibration_Gain_5*	8000...7FFF	0	signed 16	rw
	6	*Calibration_Gain_6*	8000...7FFF	0	signed 16	rw
	7	*Calibration_Gain_7*	8000...7FFF	0	signed 16	rw
	8	*Calibration_Gain_8*	8000...7FFF	0	signed 16	rw

With this object the gain of the A/D-converter channels can be corrected. The gain value, with which the measured A/D-converter value is multiplied, is calculated as:

$$\boxed{Gain\_x = 1 + \frac{Calibration\_Gain\_x}{2^{18}}}$$

with x= 1, 2, 3, 4

The resulting value range for the gain factor is: 0.875 ... 1.125

8.10 Firmware Update via DS-302-Objects ($1F50_h$...$1F52_h$)

The objects described below are used for program updates via the object dictionary.

Attention:
The firmware update must be carried out only by qualified personnel!
Faulty program update can result in deleting of the memory and loss of the firmware. The module then can not be operated further!

In normal DS 301 mode the objects $1F50_h$ and $1F52_h$ can not be accessed.
The object $1F51_h$ is available in normal DS 301-mode, too.
For further information about the objects and the firmware-update please refer to the manual 'Firmware-Update via DS 302 Objects'.

Index [Hex]	Sub-index	Description	Data type	R/W
1F50	0	Boot-Loader: Firmware download	domain	rw
1F51	1	Boot-Loader: FLASH command	unsigned 8	rw
1F52	0,1,2	Boot-Loader: Firmware date	unsigned 32	ro

Firmware Update via DS-302-Objects

8.10.1 Download Control via Object $1F51_h$

INDEX	$1F51_h$
Name	Program Control
Data type	unsigned 8
Access type	rw
Value range	0...FE_h
Default value	0

Note:
The value range of this objects in the implementing of the CAN-CBX-AI814 differs from the value range specified in the DS 302.
For further information about object $1F51_h$ and the firmware-update please refer to the manual ‘Firmware Update via DS 302 Objects’