CAN-CBM-SIO1
CAN-CBM-SIO4

CAN - RS-232, RS-422, RS-485 oder TTY-Interface

CAN-CBM-PLC/331-1

Automationsrechner mit CAN-Schnittstelle

Hardware - Handbuch

Der Inhalt dieses Handbuches wurde mit größter Sorgfalt erarbeitet und geprüft. **esd** übernimmt jedoch keine Verantwortung für Schäden, die aus Fehlern in der Dokumentation resultieren könnten. Insbesondere Beschreibungen und technische Daten sind keine zugesicherten Eigenschaften im rechtlichen Sinne.

esd hat das Recht, Änderungen am beschriebenen Produkt oder an der Dokumentation ohne vorherige Ankündigung vorzunehmen, wenn sie aus Gründen der Zuverlässigkeit oder Qualitätssicherung vorgenommen werden oder dem technischen Fortschritt dienen.

Sämtliche Rechte an der Dokumentation liegen bei **esd**. Die Weitergabe an Dritte und Vervielfältigung jeder Art, auch auszugsweise, sind nur mit schriftlicher Genehmigung durch **esd** gestattet.

esd electronic system design gmbh
Vahrenwalder Str. 205
30165 Hannover

Tel.: 0511/372 98-0
FAX : 0511/372 98-68
E-Mail: info@esd.electronics.com
Internet: www.esd-electronics.com

Dokument-Datei:	I:\texte\Doku\MANUALS\CAN\Cbm\SIO-331\Deutsch\CSIO-22H.ma6
Datum des Ausdrucks:	18.07.2000

Platinenversion:	CPU331 Rev. 1.1 SIO3311 Rev. 1.1 SIO4 Rev. 1.0

Änderungen in den Kapiteln

Die hier aufgeführten Änderungen im Anwenderhandbuch betreffen sowohl Änderungen in der *Hardware* als auch reine Änderungen in der *Beschreibung* der Sachverhalte.

Kapitel	Änderungen gegenüber Vorversion
2	Eintrag in Tabelle 2.1 für CAN-CBM-PLC/331-1 geändert.
-	-

Weitere technische Änderungen vorbehalten.

Inhalt

Seite

1. Übersicht 3
1.1 Beschreibung des Moduls 3
1.2 Frontansicht mit Steckern und Kodierschaltern 5
1.2.1 CAN-CBM-SIO1 und CAN-CBM-PLC/331-1 5
1.2.2 CAN-CBM-SIO4 5
1.3 Zusammenfassung der technischen Daten 6
1.3.1 Allgemeine technische Daten 6
1.3.2 Microcontroller-Baugruppe 7
1.3.3 CAN/DeviceNet-Schnittstelle 7
1.3.4 Serielle Schnittstellen 8
1.4 Software-Unterstützung 8
1.4.1 CAN-CBM-SIO1 und CAN-CBM-SIO4 8
1.4.2 CAN-CBM-PLC/331-1 8
1.5 Bestellhinweise 9
1.5.1 CAN-CBM-SIO1 und CAN-CBM -SIO4 9
1.5.2 CAN-CBM-PLC/331-1 10

2. CAN-Identifier 11

3. Beschreibung der Baugruppen 15
3.1 CAN/DeviceNet-Baugruppe 15
3.1.1 Interface-Schaltung 15
3.2 Serielle Schnittstelle X100 (9-pol. DSUB Stift) 17
3.2.1 Konfiguration 17
3.2.2 Anschluß der verschiedenen seriellen Schnittstellen am DSUB9-Stecker . . . 19
3.2.2.1 Die RS-232-Schnittstelle 19
3.2.2.2 Die RS-422-Schnittstelle 20
3.2.2.3 Die RS-485-Schnittstelle 20
3.2.2.4 Die TTY(20 mA)-Schnittstelle 21
3.2.3 Anschluß der verschiedenen seriellen Schnittstellen an RJ45-Buchsen 22
3.2.3.1 Die RS-232-Schnittstelle 22
3.2.3.2 Die RS-422-Schnittstelle 23
3.2.3.3 Die RS-485-Schnittstelle 23
3.2.3.4 Die TTY(20 mA)-Schnittstelle 24

4. Steckerbelegungen 25
4.1 CAN-Bus (X400, 5-pol. Combicon-Style) 25
4.2 DeviceNet (X400, 5-pol. Combicon-Style) 25
4.3 Belegung der seriellen Schnittstelle auf DSUB9 26
4.3.1 RS-232-Interface (X100, 9-pol. DSUB Stift) 26
4.3.2 RS-422-Interface (X100, 9-pol. DSUB Stift) 27
4.3.3 RS-485-Interface (X100, 9-pol. DSUB Stift) 28
4.3.4 TTY-passiv-Interface (X100, 9-pol. DSUB Stift) 29
4.3.5 TTY-aktiv-Interface (X100, 9-pol. DSUB Stift) 30

Inhalt

Seite

4.4 Belegung der seriellen Schnittstellen auf RJ45-Buchsen 31
4.4.1 Serielle Schnittstellen 2...4 (P200/P230, 8-pol. RJ45-Buchse) 31
4.4.2 Belegung der RJ45-Buchsen (P200/230, 8-polig) 32
4.4.3 Anschluß des Adapters RJ45-DSUB9/Buchse 33
4.4.4 Anschluß des Adapters RJ45-DSUB25/Buchse 35
4.5 Spannungszuführung (X101, UEGM) 37

5. Konfiguration des CAN-CBM-PLC/331-1/-2 Moduls 39

6. Korrekte Verdrahtung galvanisch getrennter CAN-Netze 49

7. Stromlaufpläne 53

1. Übersicht

1.1 Beschreibung des Moduls

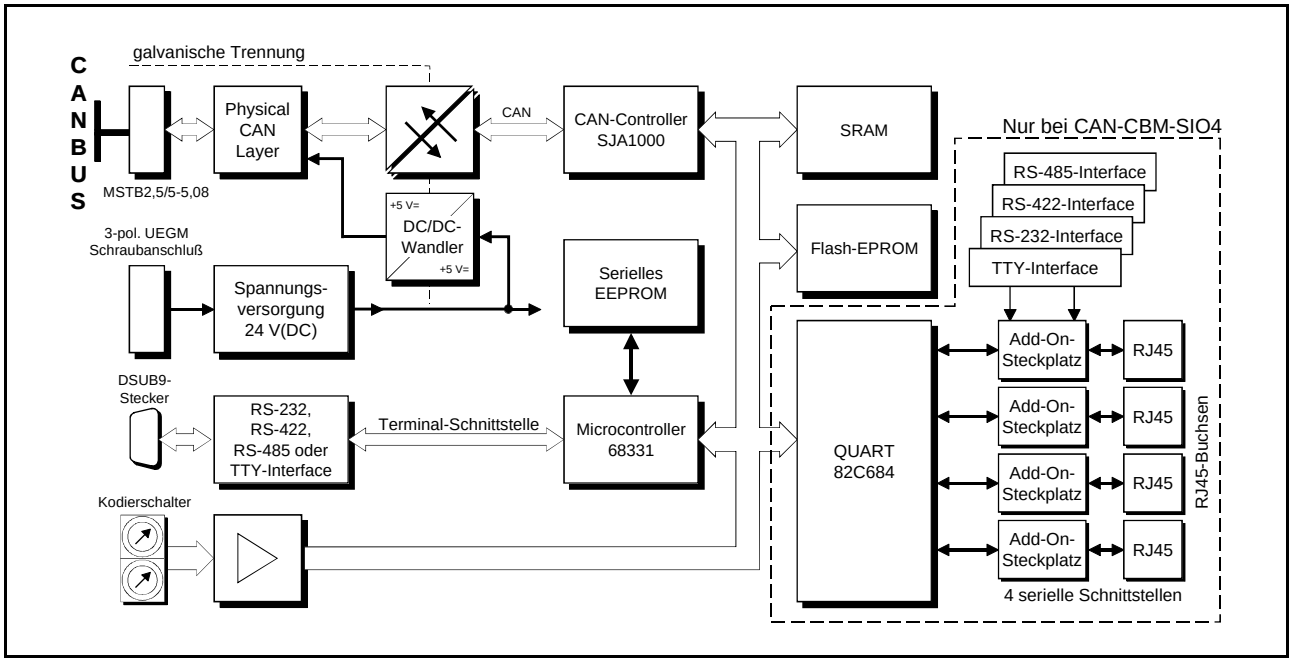


Abb. 1.1.1: Blockschaltbild der CAN-Module

Die Module CAN-CBM-SIO1 und CAN-CBM-PLC/331-1 bieten die Kopplung zwischen einer seriellen Schnittstelle und dem CAN-Netz. Das CAN-CBM-PLC/331-1-Modul ist darüber hinaus als SPS-Controller mit dem Software-Tool CoDeSys konfiguriert. Das Modul CAN-CBM-SIO4 ist mit insgesamt fünf seriellen Schnittstellen ausgestattet. Das physikalische Interface der seriellen Schnittstellen läßt sich, wie bei dem CAN-CBM-SIO1-Modul über Add-On-Platinen (Piggybacks) konfigurieren.

Die hier beschriebenen Module arbeiteten mit einem 68331 Microcontroller, der die CAN-Daten in einem lokalen SRAM zwischenspeichert. Datensicherheit und -konsistenz bis zu 1 MBit/s auf dem CAN-Netz werden garantiert. Die Firmware - auch optionale Protokolle - wird im Flash gehalten.

Die zu ISO 11898 kompatible CAN-Schnittstelle gestattet eine maximale Datenübertragungsrate von 1 MBit/s. Das CAN-Interface ist durch Optokoppler und DC/DC-Wandler galvanisch getrennt. Der Anschluß erfolgt über einen 5-poligen Steckverbinder mit Schraubkontakten. Das Modul ist außerdem mit DeviceNet-Interface lieferbar.

Die Parameter der seriellen Schnittstelle(n) sind via CAN konfigurierbar - die maximale Baudrate beträgt 500 kBaud. Die Parameter werden - wie auch die CAN-Einstellungen - in einem EEPROM abgespeichert.

Übersicht

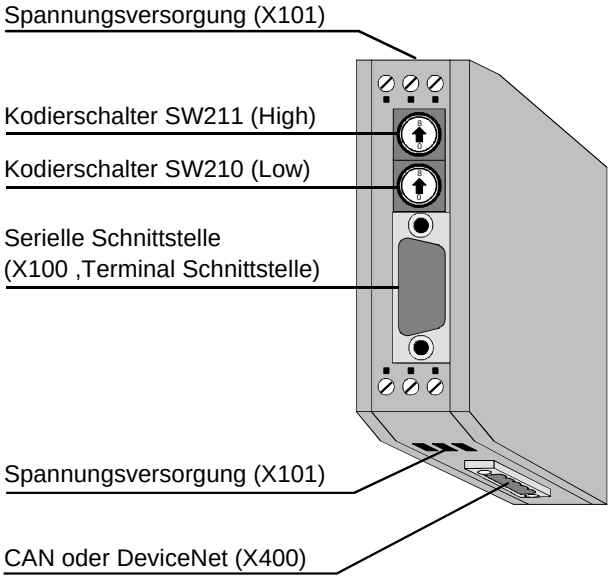
Alternativ sind für CAN-CBM-SIO1 und CAN-CBM-SIO4 als Option gängige Protokolle wie 3964R, Modbus oder auch FreePort zum Anschluß einer S7-200 verfügbar. Kundenspezifische Protokolle können auf Anfrage erstellt oder mit Hilfe einer GNU-C-Umgebung selbst entwickelt werden.

Durch Nutzung der RS-232-Schnittstelle als Modem-Anschluß kann auch eine Fernwartung des CAN-Netzes im Remote-Betrieb erfolgen. Als physikalische Schnittstelle stehen neben RS-232 wahlweise auch RS-422, RS-485 oder TTY-20mA zur Verfügung. Der Anschluß erfolgt über einen DSUB9-Stecker sowie vier zusätzlichen RJ45-Buchsen beim CBM-SIO4-Modul.

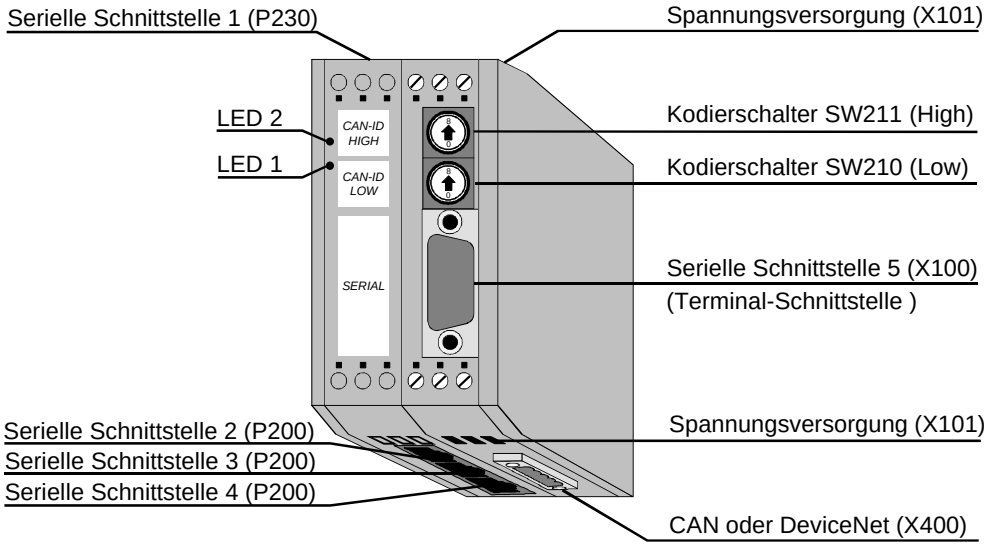
Als Schicht-7-Protokolle auf der CAN-Seite werden auf Wunsch CANopen und DeviceNet unterstützt.

1.2 Frontansicht mit Steckern und Kodierschaltern

1.2.1 CAN-CBM-SIO1 und CAN-CBM-PLC/331-1



1.2.2 CAN-CBM-SIO4



Übersicht

1.3 Zusammenfassung der technischen Daten

1.3.1 Allgemeine technische Daten

Versorgungsspannung	Nennspannung: 24 V/DC ±10%, Stromaufnahme (bei 20°C): max. 70 mA (+20 mA bei TTY-Betrieb)
Steckverbinder	X100 (DSUB9, Stifte) - CAN-CBM-SIO1: serielle Schnittstelle 1 CAN-CBM-SIO4: serielle Schnittstelle 5 CAN-CBM-PLC/331-1: serielle Schnittstelle 1 X101 (6-pol. Schraubverbinder UEGM) - 24V-Spannungsversorgung X400 (Combicon-Bauform, 5-pol. MSTB2.5/5-5.08) - CAN oder DeviceNet *Nur bei CAN-CBM-SIO4 bestückt:* P200 (RJ45-Buchse, einfach) - serielle Schnittstelle 1 P230 (RJ45-Buchse, dreifach) - serielle Schnittstellen 2, 3, 4
Temperaturbereich	0...50 /C Umgebungstemperatur
Luftfeuchtigkeit	max. 90%, nicht kondensierend
Maße der Gehäuses (B x H x T)	Breite: 25 mm (CAN-CBM-SIO1, CAN-CBM-PLC/331-1), 40 mm (CAN-CBM-SIO4), Höhe: 85 mm, Tiefe: 83 mm (Maße einschließlich Hutschienenhalterung und Steckerüberstand DSUB9, ohne CAN/DeviceNet-Stecker)
Gewicht	CAN-CBM-SIO1, CAN-CBM-PLC/331-1: ca. 150 g

Tabelle 1.3.1: Allgemeine technische Daten

1.3.2 Microcontroller-Baugruppe

Microcontroller	68331
Speicher	SRAM: 128 k x 16 Bit Flash-EPROM: 128 k x 8 Bit EEPROM: serielles I²C-EEPROM
Debug-Interface	für Service und Programmierung

Tabelle 1.3.2: Microcontroller-Baugruppe

1.3.3 CAN/DeviceNet-Schnittstelle

Anzahl der CAN-Interfaces	1x CAN Option: 1x DeviceNet
CAN-Controller	SJA1000, CAN 2.0A/B
Galvanische Trennung des CAN-Interfaces gegenüber den anderen Baugruppen	über Optokoppler und DC/DC-Wandler
Physical Layer CAN	Physical Layer gemäß ISO 11898, Übertragungsrate programmierbar von 10 kBit/s bis 1 MBit/s
Physical Layer DeviceNet (Option)	Physical Layer gemäß DeviceNet-Spezifikation Rev. 2.0, Bitrate: 125 kBit/s, 250 kBit/s, 500 kBit/s

Tabelle 1.3.3: Daten der CAN-Schnittstelle

Übersicht

1.3.4 Serielle Schnittstellen

	Schnittstelle auf DSUB9-Stecker	**Schnittstellen auf RJ45-Buchsen** (nur bei CAN-CBM-SIO4 bestückt)
Kanal-Bezeichung bei CAN-CBM-SIO1	Kanal 1	nicht vorhanden
Kanal-Bezeichung bei CAN-CBM-PLC/331-1	Kanal 1	nicht vorhanden
Kanal-Bezeichung bei CAN-CBM-SIO4	Kanal 5	Kanal 1, 2, 3, 4
Controller	68331	82C684
Interface	Standard: RS-232 Optionen: RS-422, RS-485, TTY aktiv/passiv	
Anschluß	9-poliger DSUB-Stecker	8-polige RJ45-Buchsen

Tabelle 1.3.4: Daten der seriellen Schnittstellen

1.4 Software-Unterstützung

Die komplette EPROM-residente Kommunikations-Firmware zum Betrieb der CAN-CBM-Module ist im Lieferumfang enthalten.

1.4.1 CAN-CBM-SIO1 und CAN-CBM-SIO4

Im Standardmode ohne Protokoll sendet die Baugruppe bei Empfang von 8 ASCII-Zeichen oder nach Empfang eines konfigurierbaren Endezeichens (z.B. CR, LF oder EOT) sowie nach Ablauf eines einstellbaren Time-Outs (wenn keine Zeichen mehr empfangen wurden) einen CAN-Frame auf dem vorher eingestellten CAN-Identifier.

1.4.2 CAN-CBM-PLC/331-1

Das CAN-CBM-PLC/331-1-Modul wird mit $\text{CoDeSys}_{\text{RTOS-UH}}$ betrieben. $\text{CoDeSys}_{\text{RTOS-UH}}$ ist in Programmiersystem unter Windows für Steuerungsanwendungen nach IEC1131-3 mit einem Laufzeitsystem unter RTOS-UH. Die Konfiguration des CAN-CBM-PLC/331-1-Moduls ist in Kapitel 5 beschrieben. Die CoDeSys Software wird mit einer Online Hilfe und einem Handbuch geliefert, die das CoDeSys-Programmiersystem beschreiben. Weitergehende Informationen zu den höheren Protokollschichten enthält die CAL/CANopen Spezifikation ‘CiA-Draft Standard 301’

1.5 Bestellhinweise

1.5.1 CAN-CBM-SIO1 und CAN-CBM -SIO4

Typ	Eigenschaften	Bestell-Nr.
CAN-CBM-SIO1	1x CAN 2.0A/B auf RS-232	C.2840.03
CAN-CBM-SIO4	1x CAN 2.0A/B auf (4+1) x RS-232	C.2843.03
Statt RS-232 ergänzt mit (bei Bestellung anzugeben):	RS-422 Adapter RS-485 Adapter TTY-20mA passiv TTY-20mA aktiv	X.1930.02 X.1930.04 X.1930.06 X.1930.08
CAN-CBM-SIO	Freeport-Protokoll	C.2840.42
CAN-CBM-SIO-DvN	DeviceNet-Slave	C.2840.13
CAN-CBM-SIO-DvN-M	DeviceNet-Master (Scanner)	C.2840.19
CAN-CBM-SIO-Co	CANopen (Slave)	C.2840.18
-	Anschlußkabel 8-pol. RJ45 auf 8-pol. RJ45, Länge: 2 m	C.2401.30
-	Adapter 8-pol. RJ45 auf 25-pol. DSUB/Stifte - Pinbelegung ist ohne Werkzeug frei konfigurierbar.	C.2401.34
-	Adapter 8-pol. RJ45 auf 25-pol. DSUB/Buchse - Pinbelegung ist ohne Werkzeug frei konfigurierbar.	C.2401.36
-	Adapter 8-pol. RJ45 auf 9-pol. DSUB/Buchse - Pinbelegung ist ohne Werkzeug frei konfigurierbar.	C.2401.38
-	Adapter 8-pol. RJ45 auf 9-pol. DSUB/Stifte - Pinbelegung ist ohne Werkzeug frei konfigurierbar.	C.2401.40
CAN-CBM-SIO-MD	Anwenderhandbuch in deutsch [1*)]	C.2840.20

[1*)]... Wird das Handbuch gemeinsam mit dem Modul bestellt, so wird es kostenlos mitgeliefert.

Tabelle 1.5.1: Bestellhinweise für Modul CAN-CBM-SIO1 und CAN-CBM-SIO4

Übersicht

1.5.2 CAN-CBM-PLC/331-1

Typ	Eigenschaften	Bestell-Nr.
CAN-CBM-PLC/331-1	1x CAN 2.0A/B auf RS-232	C.2845.03
Statt RS-232 ergänzt mit (bei Bestellung anzugeben):	RS-422 Adapter RS-485 Adapter TTY-20mA passiv TTY-20mA aktiv	X.1930.02 X.1930.04 X.1930.06 X.1930.08
$CoDeSys_{RTOS-UH}$	IEC1131-3 SPS-Entwicklungssystem mit 5 Sprachen; für RTOS-UH; PC-Host	P.4071.02
CAN-CBM-PLC/331-MD	Anwenderhandbuch in deutsch [1*)]	C.2845.20

[1*)]... Wird das Handbuch gemeinsam mit dem Modul bestellt, so wird es kostenlos mitgeliefert.

Tabelle 1.5.1: Bestellhinweise für Modul CAN-CBM-PLC/331-1

2. CAN-Identifier

Das CAN-CBM-SIO4-Modul besitzt für jeden der fünf Kanäle einen Rx- und einen Tx-Identifier. Die Module CAN-CBM-SIO1 und CAN-CBM-PLC/331-1-Modul besitzen nur ein Identifier-Paar, da sie nur über einen seriellen Kanal verfügen.

Modul	Physikalischer Kanal	CAN-Daten empfangen	CAN-Daten-Senden
CAN-CBM-SIO1	Terminal-Schnittstelle auf DSUB9 Kanal 1	RxID1	TxID1
CAN-CBM-SIO4	Kanal 1 Kanal 2 Kanal 3 Kanal 4 Terminal-Schnittstelle auf DSUB9 Kanal 5	RxID1 RxID2 RxID3 RxID4 RxID5	TxID1 TxID2 TxID3 TxID4 TxID5
CAN-CBM-PLC/331-1	Über die Kodierschalter wird für das CAN-CBM-PLC/331-1 Modul derzeit keine Einstellung genutzt. Der CAN-Identifier wird über CoDeSys eingestellt.		

Achtung: Der Terminal-Schnittstelle (auf DSUB9) sind auf dem CAN-CBM-SIO4-Modul der Rx-Identifier **RxID5** und der Tx-Identifier **TxID5** zugeordnet. Auf dem CAN-CBM-SIO1 und dem CAN-CBM-PLC/331-1 -Modul mit nur einer seriellen Schnittstelle sind der Terminal-Schnittstelle der Rx-Identifier **RxID1** und der Tx-Identifier **TxID1** zugeordnet!

Tabelle 2.1: Zuordnung der seriellen Kanäle zu dem Identifier des Moduls

Die Identifier des CAN-CBM-SIO1- und des CAN-CBM-SIO4-Moduls errechnen sich in der Default-Konfiguration aus einem Basiswert, der über die Kodierschalter eingestellt wird, und einem festen Offset dazu.
Der Identifier des CAN-CBM-PLC/331-1-Moduls wird über CoDeSys eingestellt.

CAN-Identifier

CAN-CBM-SIO4	
Identifier	**Offset (HEX)**
TxID1	0
TxID2	1
TxID3	2
TxID4	3
TxID5	4
RxID1	5
RxID2	6
RxID3	7
RxID4	8
RxID5	9

CAN-CBM-SIO1	
Identifier	**Offset (HEX)**
TxID1	0
RxID1	1

Tabelle 2.2: Offset der Identifier in der Default-Einstellung

Berechnung des Basiswertes und des Identifier:

Basiswert	= 10 x Kodierschalter-Wert
Identifier	= Basiswert + Offset (HEX)

Beispiel:

Die Kodierschalter sind jeweils auf ‘1’ eingestellt. Der Kodierschalter-Wert beträgt also $11. Damit ergibt sich der Basiswert:

$A x $11 = $AA

Die Identifier-Werte ergeben sich wie folgt:

$AA + Offset (HEX) = Identifier

CAN-CBM-SIO4	
Identifier	**Wert (HEX)**
TxID1	AA
TxID2	AB
TxID3	AC
TxID4	AD
TxID5	AE
RxID1	AF
RxID2	B0
RxID3	B1
RxID4	B2
RxID5	B3

CAN-CBM-SIO1	
Identifier	**Wert (HEX)**
TxID1	AA
RxID1	AB

Tabelle 2.3: Beispiel der Identifier-Einstellung

3. Beschreibung der Baugruppen

3.1 CAN/DeviceNet-Baugruppe

3.1.1 Interface-Schaltung

Die CAN-CBM-Module sind mit einem CAN-Interface gemäß ISO11898 oder mit einem DeviceNet-Interface lieferbar. Für beide Interfaces wird der gleiche Steckverbinder verwendet. Die Belegung des Steckers ist jedoch unterschiedlich. Die folgenden Abbildungen zeigen die beiden Interfaces.

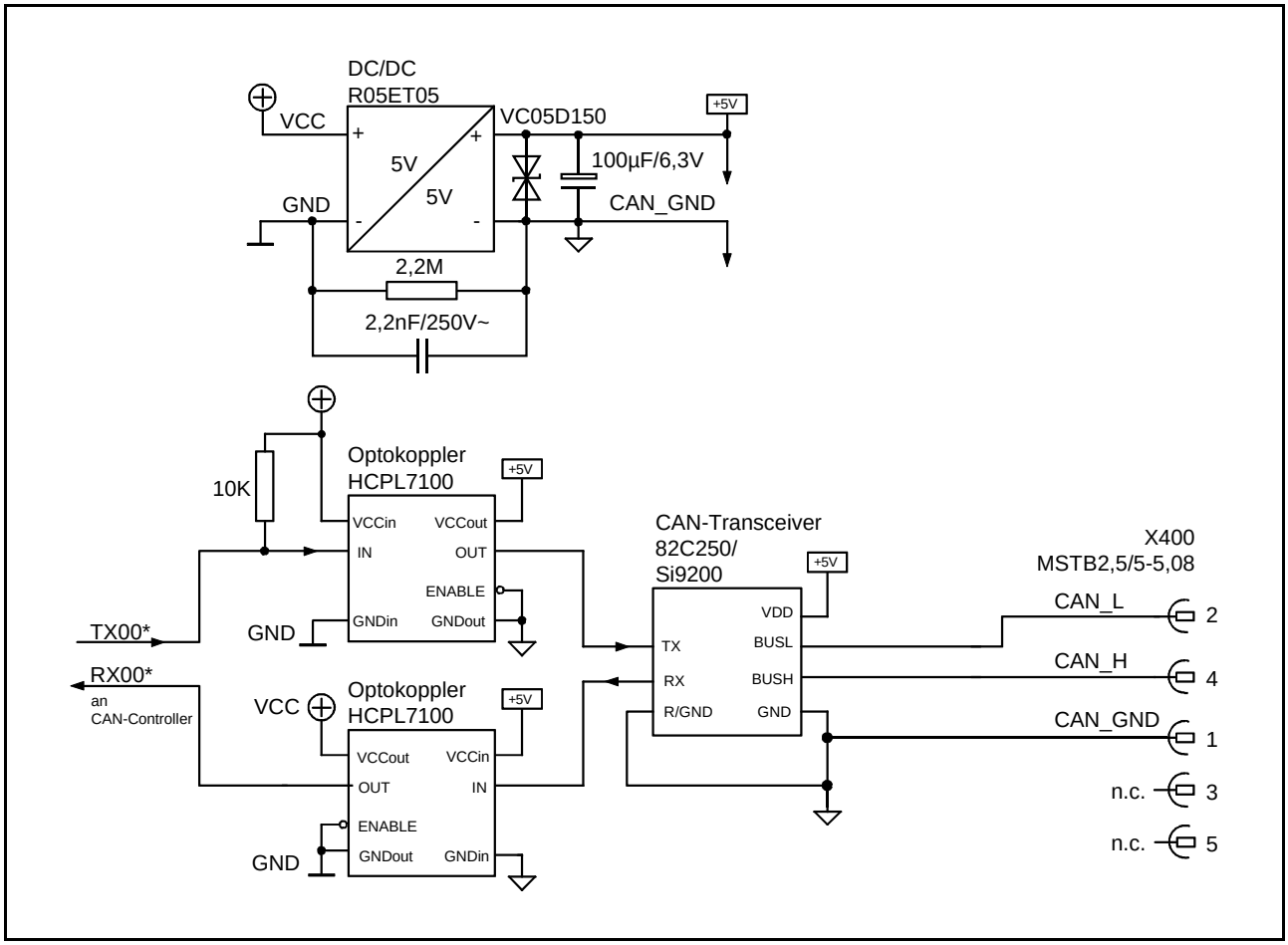


Abb. 3.1.1: Schaltung des CAN-Interfaces

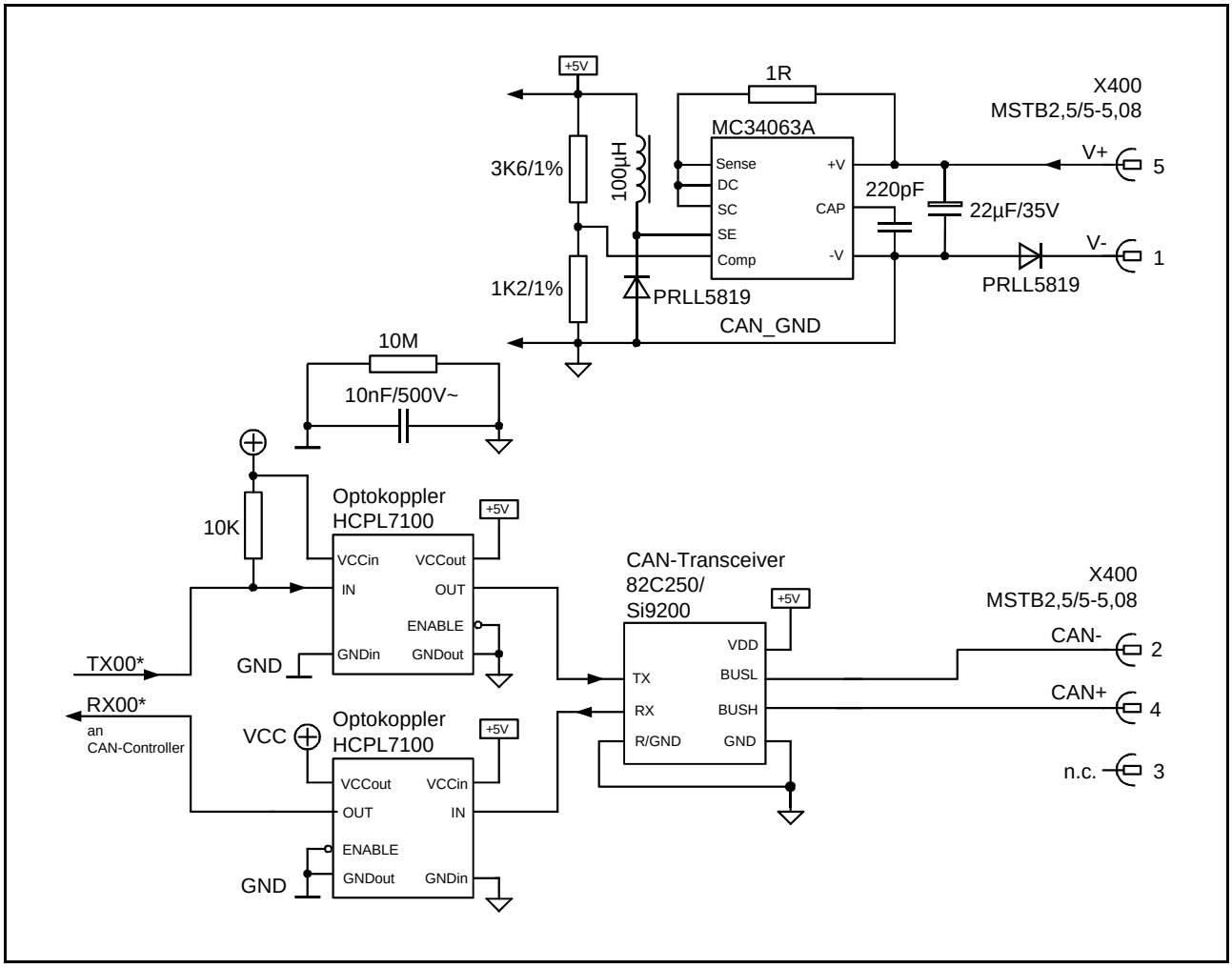


Abb. 3.1.2: Schaltung des DeviceNet-Interfaces

3.2 Serielle Schnittstelle X100 (9-pol. DSUB Stift)

3.2.1 Konfiguration

Das physikalische Interface der seriellen Schnittstelle kann als RS-232-, RS-422-, RS-485-, TTY-aktiv- oder TTY-passiv-Interface konfiguriert werden. Für den RS-232-Betrieb wird dabei ein RS-232A-Treiberbaustein eingesetzt und für die anderen Interfaces Aufsteckplatinen (Piggybacks).

Die seriellen Schnittstellen werden vom 68331 Controller und vom QUART 82C684 gesteuert. Die Bitrate der Schnittstelle ist parametrierbar.

Die maximale Bitrate des Controllers QUART 82C684 beträgt 230 kBit/s. Bei gleichzeitigem Betrieb von 4 Schnittstellen sind jedoch nur 38,4 kBit/s zu erreichen.

Der im 68331 integrierte serielle Controller unterstützt in dieser Applikation Bitraten bis zu 500 kBit/s.

Bitraten über 38,4 kBit/s sind nur mit RS-422- und RS-485-Interfaces erreichbar. Mit den eingesetzten RS-232-Treibern sind maximal 38,4 kBit/s möglich.

Baugruppe	maximal Bitrate
Controller: - 68331 - QUART 82C684	 500 kBit/s 230 kBit/s (38,4 kBit/s)
RS-422-Interface	500 kBit/s
RS-485-Interface	500 kBit/s
RS-232-Interface	38,4 kBit/s
TTY-Interface	38,4 kBit/s

Tabelle 3.2.1: Erreichbare Bitraten bei den verschiedenen physikalischen Interfaces

Die folgenden Bitraten sind mit Hilfe der Software einstellbar. Die Werte der zweiten Spalte zeigen die tatsächlichen Bitraten, die sich speziell beim Controller 68331 ergeben.

Bitrate (Sollwert) [Bit/s]	**Bitrate (Istwert, 68331) [Bit/s]**
500.000 (nur 68331)	500.000
38.400	38.462
19.200	19.231
9.600	9.615
4.800	4.808
2.400	2.404
1.200	1.199
600	600,2
300	299,9

Tabelle 3.2.2: Einstellbare Bitraten

3.2.2 Anschluß der verschiedenen seriellen Schnittstellen am DSUB9-Stecker

Im Folgenden wird die Verdrahtung der seriellen Schnittstellen für Kanal 1 (CAN-CBM-SIO1 und CAN-CBM-PLC/331-1) bzw. Kanal 5 (CAN-CBM-SIO4) dargestellt. Die Abbildungen sollen die im Anhang (Steckerbelegung) verwendeten Kurzbezeichnungen der Signale erläutern. Im Anhang (Stromlaufpläne) sind außerdem die Schaltpläne der verschiedenen lieferbaren Piggybacks zu finden.
Die Signalbezeichnung ist beispielhaft für den Anschluß der CAN-CBM-Module als Sender (Terminal DEE) angegeben.

3.2.2.1 Die RS-232-Schnittstelle

Die Signale CTS, DSR und DCD werden von den CAN-CBM-Modulen nicht ausgewertet!

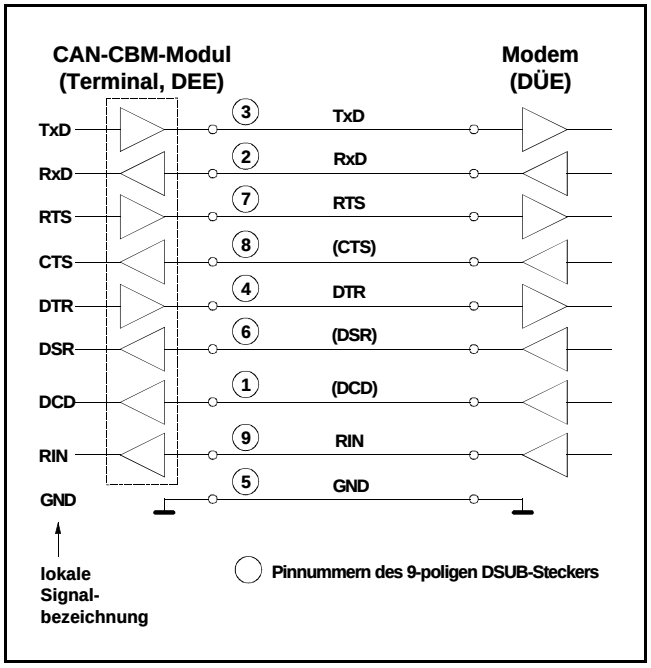


Abb. 3.2.3: Anschlußschema für RS-232-Betrieb

3.2.2.2 Die RS-422-Schnittstelle

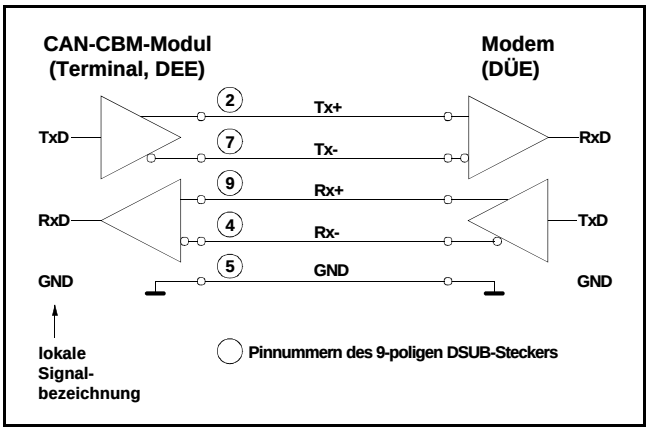


Abb. 3.2.4: Anschlußschema für RS-422-Betrieb

3.2.2.3 Die RS-485-Schnittstelle

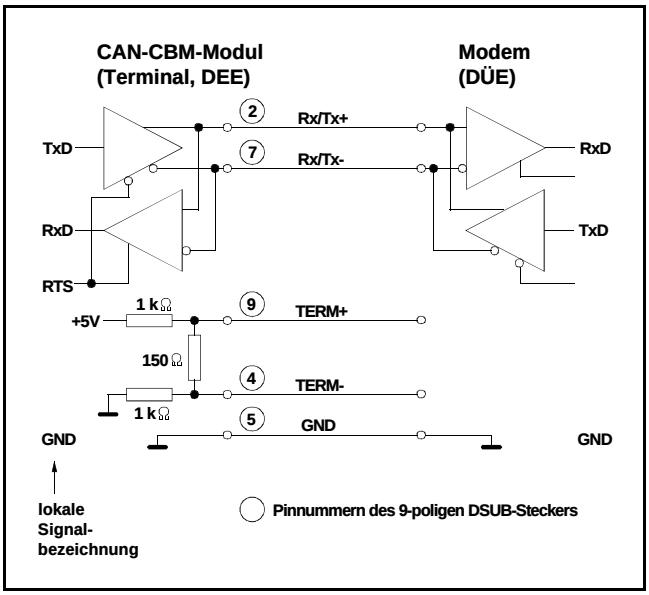


Abb. 3.2.5: Anschlußschema für RS-485-Betrieb

Um das Abschlußwiederstandsnetzwerk, das sich auf dem Piggyback befindet, zu aktivieren, sind z.B. im DSUB-Stecker die Pins 9 und 2, sowie die Pins 4 und 7 zu verbinden.

3.2.2.4 Die TTY(20 mA)-Schnittstelle

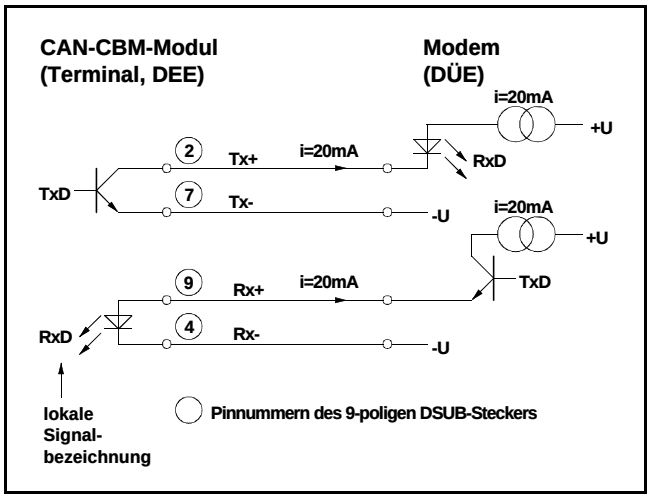


Abb. 3.2.6: Anschlußschema für TTY-Betrieb (passiv)

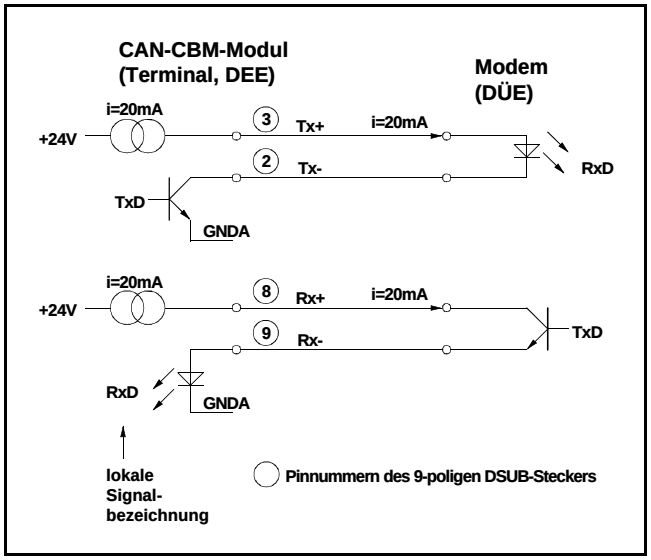


Abb. 3.2.7: Anschlußschema für TTY-Betrieb (aktiv)

3.2.3 Anschluß der verschiedenen seriellen Schnittstellen an RJ45-Buchsen

Im Folgenden wird die Verdrahtung der seriellen Schnittstellen in Bezug auf die Datenrichtung dargestellt. Die Abbildungen sollen die im Kapitel *Steckerbelegung* verwendeten Kurzbezeichnungen der Signale erläutern. Im Kapitel *Stromlaufpläne* sind außerdem die Schaltpläne der verschiedenen lieferbaren Piggybacks zu finden.

Als Beispiel für das Anschlußkabel ist hier das Adapterkabel RJ45-DSUB9/Buchse gewählt worden, das für den RS-232-Modem-Betrieb (DÜE) der CAN-CBM-SIO4 ausgelegt ist.

Die mit RTS bezeichnete Leitung kann im Controller 82C684 als RTS- oder DTR-Signal programmiert werden. Die Modul-Software programmiert das Signal als RTS-Eingang. Wird ein DTR-Signal von dem anderen angeschlossenen Gerät benötigt, kann die RTS-Leitung auf den DTR-Pin gebrückt werden.

3.2.3.1 Die RS-232-Schnittstelle

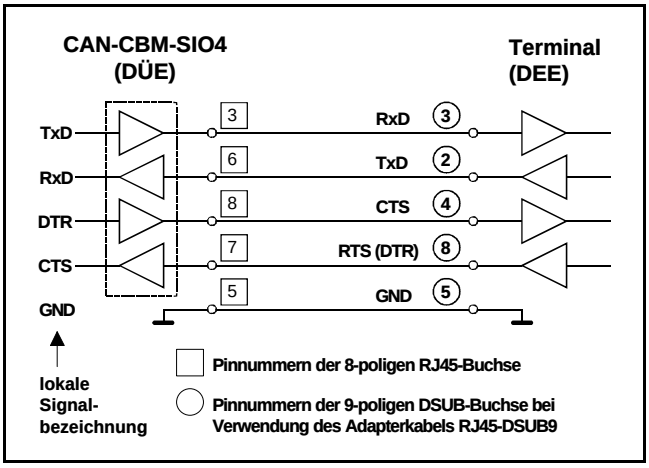


Abb. 3.2.8: Anschlußschema für RS-232-Betrieb

3.2.3.2 Die RS-422-Schnittstelle

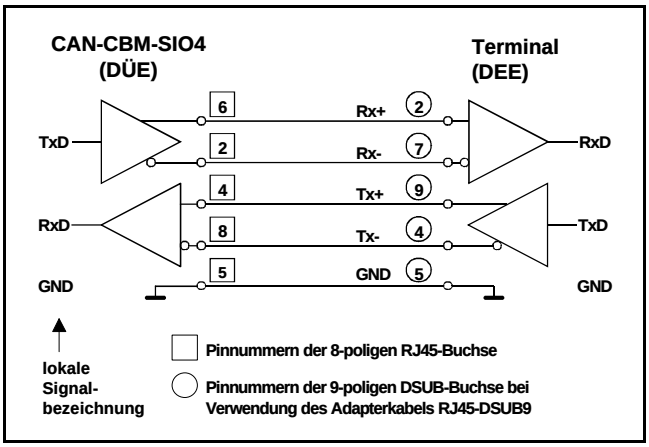


Abb. 3.2.9: Anschlußschema für RS-422-Betrieb

3.2.3.3 Die RS-485-Schnittstelle

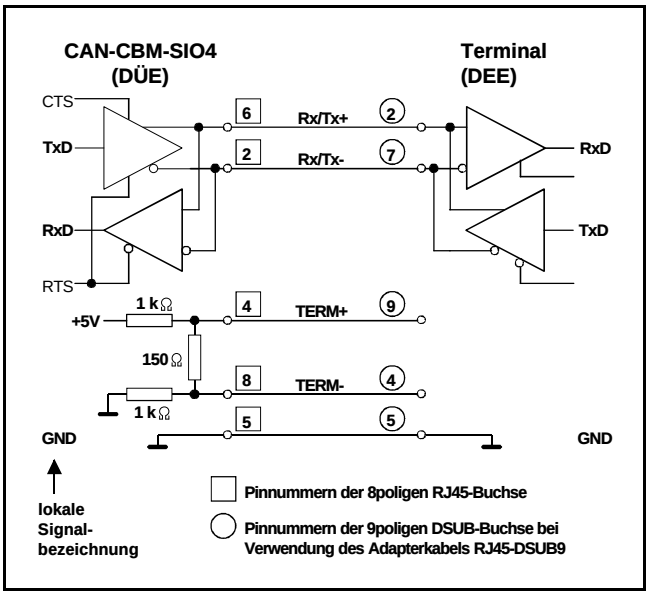


Abb. 3.2.10: Anschlußschema für RS-485-Betrieb

Die Pins 4 und 8 der RJ45-Buchsen führen beim RS-485-Betrieb auf ein Abschlußwiderstandsnetzwerk, das sich auf dem Piggyback befindet. Um den Abschluß zu aktivieren, ist das Signal Rx/Tx+ mit TERM+ und das Signal Rx/Tx- mit TERM- zu verbinden.

3.2.3.4 Die TTY(20 mA)-Schnittstelle

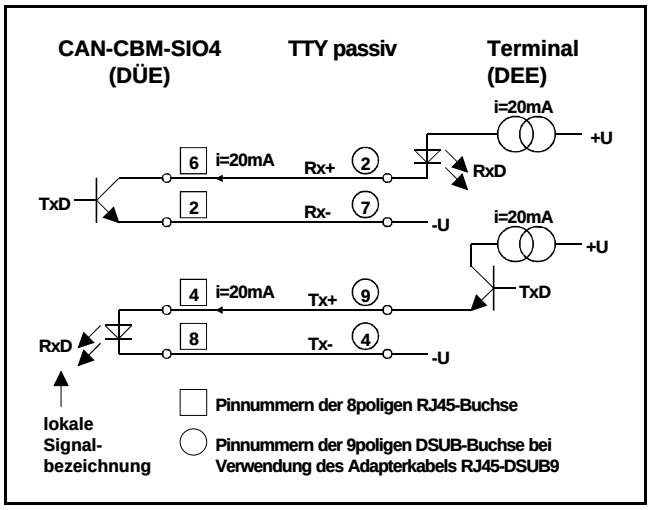


Abb. 3.2.11: Anschlußschema für TTY-Betrieb (passiv)

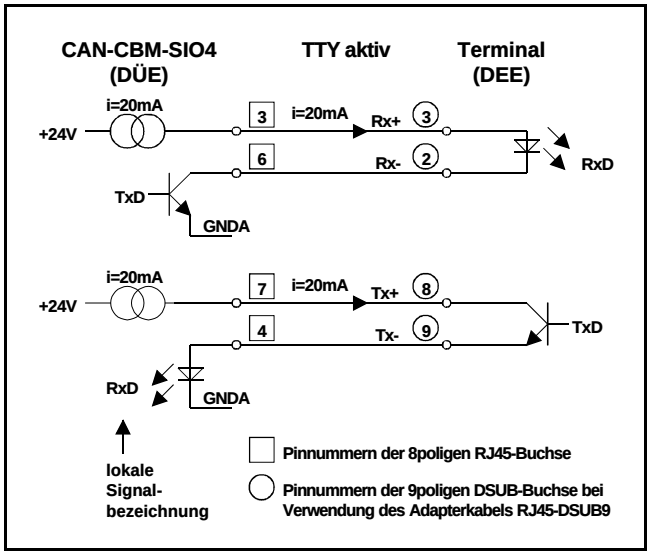
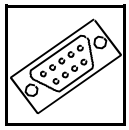


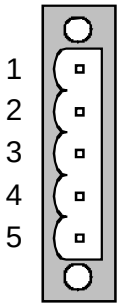
Abb. 3.2.12: Anschlußschema für TTY-Betrieb (aktiv)



4. Steckerbelegungen

4.1 CAN-Bus (X400, 5-pol. Combicon-Style)

Pin-Zuordnung:



Pin-Belegung:

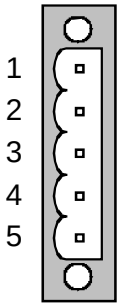
Pin	Signal
1	CAN_GND
2	CAN_L
3	n.c.
4	CAN_H
5	n.c.

Signalbeschreibung:

CAN_L,
CAN_H... CAN-Signalleitungen
CAN_GND ... Bezugspotential des lokalen CAN-Physical Layers

4.2 DeviceNet (X400, 5-pol. Combicon-Style)

Pin-Zuordnung:

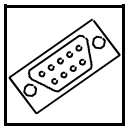


Pin-Belegung:

Pin	Signal
1	V-
2	CAN-
3	n.c.
4	CAN+
5	V+

Signalbeschreibung:

V+... Zuführung der Spannungsversorgung (U_{VCC} = 24 V ± 4%)
V-... Bezugspotential zu V+ und zu CAN+/CAN-
CAN+, CAN-... CAN-Signalleitungen



Steckerbelegung

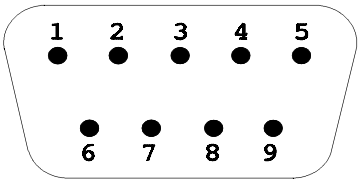
4.3 Belegung der seriellen Schnittstelle auf DSUB9

Hinweise zum Anschluß der seriellen Schnittstellen auf DSUB9 können auch dem vorangegangenen Kapitel '*Anschluß der verschiedenen seriellen Schnittstellen am DSUB9-Stecker*' entnommen werden. Aus den dort aufgeführten Prinzipschaltbildern läßt sich im Zweifelsfall die Signalrichtung (Rx<->Tx) eindeutig bestimmen.

4.3.1 RS-232-Interface (X100, 9-pol. DSUB Stift)

Die Signale CTS, DSR und DCD werden von der CAN-CBM-Modulen nicht ausgewertet!

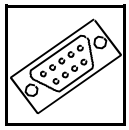
Pin-Zuordnung:



Pin-Belegung:

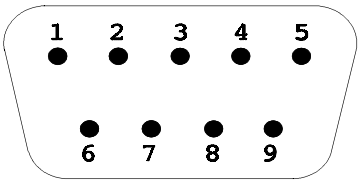
Signal		Pin		Signal	
(DSR)	(Eingang)	6	1	(DCD)	(Eingang)
RTS	(Ausgang)	7	2	RxD	(Eingang)
(CTS)	(Eingang)	8	3	TxD	(Ausgang)
RIN	(Eingang)	9	4	DTR	(Ausgang)
			5	GND	

9-poliger DSUB-Stecker



4.3.2 RS-422-Interface (X100, 9-pol. DSUB Stift)

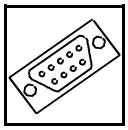
Pin-Zuordnung:



Pin-Belegung:

Signal	Pin		Signal
-	6	1	-
Tx- (Ausgang)	7	2	Tx+ (Ausgang)
-	8	3	-
Rx+ (Eingang)	9	4	Rx- (Eingang)
		5	GND

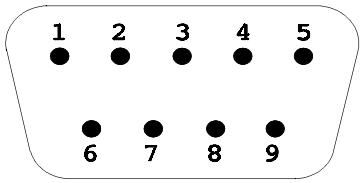
9-poliger DSUB-Stecker



Steckerbelegung

4.3.3 RS-485-Interface (X100, 9-pol. DSUB Stift)

Pin-Zuordnung:

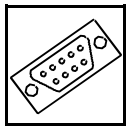


Pin-Belegung:

Signal	Pin		Signal
-	6	1	-
		2	Rx/Tx+
Rx/Tx-	7	3	-
-	8	4	Term-(für Rx/Tx-)
Term+ (für Rx/Tx+)	9	5	GND

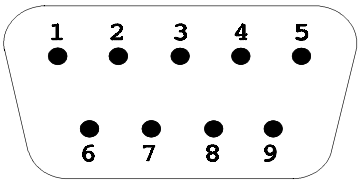
9-poliger DSUB-Stecker

Die Signale Term+ und Term- sind mit einem Abschlußwiderstandsnetzwerk auf der Platine verbunden. Um den Aschluß zu aktivieren, ist Term+ mit dem Rx/Tx+-Signal zu verbinden und Term- mit dem Rx/Tx- -Signal.



4.3.4 TTY-passiv-Interface (X100, 9-pol. DSUB Stift)

Pin-Zuordnung:

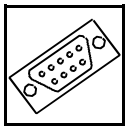


Pin-Belegung:

Signal	Pin		Signal
-	6	1	-
Tx- (Sender)	7	2	Tx+ (Sender)
(I2+)	8	3	(I1+)
Rx+ (Empfänger)	9	4	Rx- (Empfänger)
		5	GND

9-poliger DSUB-Stecker

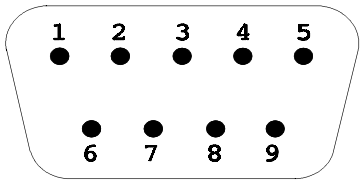
( )... Die in Klamern angegebenen Signale sind zwar belegt, werden aber für den Betrieb dieses physikalischen Interfaces nicht benötigt.



Steckerbelegung

4.3.5 TTY-aktiv-Interface (X100, 9-pol. DSUB Stift)

Pin-Zuordnung:

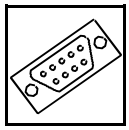


Pin-Belegung:

Signal	Pin		Signal
-	6	1	-
		2	Tx- (Sender)
(GNDA)	7	3	Tx+ (Sender)
Rx+ (Empfänger)	8	4	(GNDA)
Rx- (Empfänger)	9	5	GND

9-poliger DSUB-Stecker

( )... Die in Klamern angegebenen Signale sind zwar belegt, werden aber für den Betrieb dieses physikalischen Interfaces nicht benötigt.



4.4 Belegung der seriellen Schnittstellen auf RJ45-Buchsen

Diese Schnittstellen sind nur auf der CAN-CBM-SIO4 bestückt!

Hinweise zum Anschluß der seriellen Schnittstellen können auch dem vorangegangenen Kapitel '*Anschluß der verschiedenen seriellen Schnittstellen an den RJ45-Buchsen*' entnommen werden. Aus den dort aufgeführten Prinzipschaltbildern läßt sich im Zweifelsfall die Signalrichtung (Rx<->Tx) eindeutig bestimmen.

4.4.1 Serielle Schnittstellen 2...4 (P200/P230, 8-pol. RJ45-Buchse)

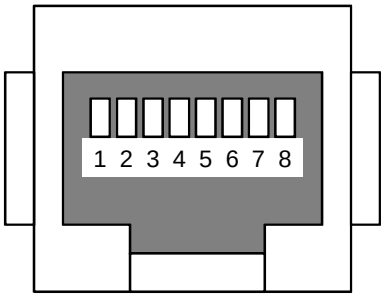
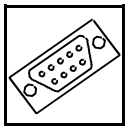


Abb. 4.4.1: Kontaktbezeichnungen RJ45-Buchse



Steckerbelegung

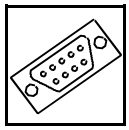
4.4.2 Belegung der RJ45-Buchsen (P200/230, 8-polig)

Die in der unten aufgeführten Tabelle verwendeten Signalnamen entsprechen den physikalischen Datenrichtungen von der CAN-CBM-SIO4 aus betrachtet, d.h. das TxD-Signal ist ein Ausgang und muß mit der RxD-Leitung des anderen Gerätes verbunden werden.

Stecker-Pin RJ45	**Signalbelegung**				
	RS-232	**RS-422**	**RS-485**	**TTY-passiv**	**TTY-aktiv**
1	-	-	-	-	-
2	-	Tx-	Rx/Tx-	Tx-	[GNDA]
3	TxD Data Output	-	-	[I1+]	Tx+
4	-	Rx+	TERM+ *1)	Rx+	Rx-
5	GND	GND	GND	GND	GND
6	RxD Data Input	Tx+	Rx/Tx+	Tx+	Tx-
7	CTS Handshake Input	GND	GND	[I2+]	Rx+
8	RTS Handshake Output	Rx-	TERM- *1)	Rx-	[GNDA]

*1)... Die Pins 4 und 8 der RJ45-Buchsen (P200/230) führen beim RS-485-Betrieb auf ein Abschlußwiderstandsnetzwerk, das sich auf dem Piggyback befindet. Um den Abschluß zu aktivieren, ist das Signal Rx/Tx+ mit TERM+ und das Signal Rx/Tx- mit TERM- zu verbinden.

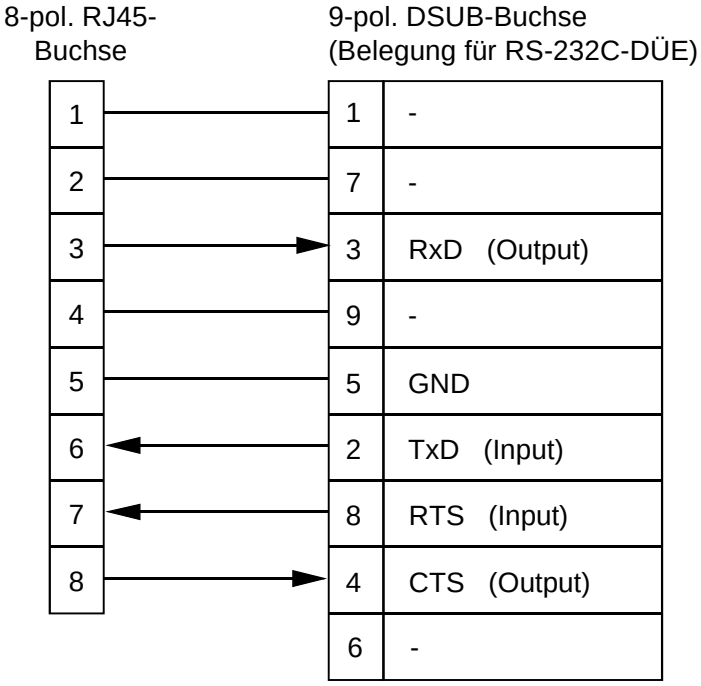
[ ]... Die in eckigen Klammern angegebenen Signale sind zwar belegt, werden aber für den Betrieb der Schnittstelle nicht benötigt.

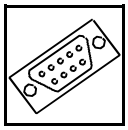


4.4.3 Anschluß des Adapters RJ45-DSUB9/Buchse

Die Adapter RJ45-DSUB9/Stift (Bestell-Nr. C.2401.40) und RJ45-DSUB9/Buchse (Bestell-Nr. C.2401.38) können ohne Werkzeug einmalig frei konfiguriert werden. Die Verbindung zwischen Adapter und CAN-CBM-SIO4 erfolgt über das Anschlußkabel RJ45-RJ45 (Bestell-Nr. C.2401.30).

Die folgende Tabelle gibt die Steckerbelegung der DSUB9-Buchse für den Fall an, daß die CAN-CBM-SIO4 im RS-232-Betrieb als Modem (DÜE), also Empfänger arbeiten soll (Signalnamen vom Terminal/DEE aus betrachtet angegeben).





Steckerbelegung

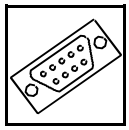
Wird der Adapter RJ45-DSUB9/Buchse für den Anschluß der anderen Schnittstellen genutzt, ergibt sich daraus folgende Belegung (Signalnamen vom Terminal/DEE aus betrachtet angegeben):

Beim Anschluß der TTY-Leitungen ist zu beachten:
Die Bezeichnungen (out) und (in) geben nur die Richtung der Datenübertragung und *nicht* die Richtung des Stromes wieder. Für den Anschluß der TTY-Signale ist das Schaltbild im Kapitel '*Anschluß der verschiedenen seriellen Schnittstellen an den RJ45-Buchsen*' hilfreich.

RJ45-Buchse	DSUB 9-Buchse	Signalbelegung			
		RS-422	RS-485	TTY-passiv	TTY-aktiv
1	1	-	-	-	-
6	2	Rx+ (out)	Rx/Tx+	Rx+ (out)	Rx- (out)
3	3	-	-	[I1+]	Rx+ (out)
8	4	Tx- (in)	TERM- *1)	Tx- (in)	[GNDA]
5	5	GND	GND	GND	GND
-	6	-	-	-	-
2	7	Rx- (out)	Rx/Tx-	Rx- (out)	[GNDA]
7	8	GND	GND	[I2+]	Tx+ (in)
4	9	Tx+ (in)	TERM+ *1)	Tx+ (in)	Tx- (in)

*1)... Die Pins 4 und 8 der RJ45-Buchsen (P200/P230) führen beim RS-485-Betrieb auf ein Abschlußwiderstandsnetzwerk, das sich auf dem Piggyback befindet. Um den Abschluß zu aktivieren, ist das Signal Rx/Tx+ mit TERM+ und das Signal Rx/Tx- mit TERM- zu verbinden.

[ ]... Die in eckigen Klammern angegebenen Signale sind zwar belegt, werden aber für den Betrieb der Schnittstelle nicht benötigt.



4.4.4 Anschluß des Adapters RJ45-DSUB25/Buchse

Die Adapter RJ45-DSUB25/Stift (Bestell-Nr. C.2401.34) und RJ45-DSUB25/Buchse (Bestell-Nr. C.2401.36) können ohne Werkzeug einmalig frei konfiguriert werden. Die Verbindung zwischen Adapter und CAN-CBM-SIO4 erfolgt über das Anschlußkabel RJ45-RJ45 (Bestell-Nr. C.2401.30).

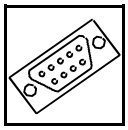
Die folgende Tabelle gibt die Steckerbelegung der DSUB25-Buchse für den Fall an, daß die CAN-CBM-SIO4 im RS-232-Betrieb als Modem (DÜE), also Empfänger arbeiten soll (Signalnamen vom Terminal/DEE aus betrachtet angegeben).
Die Belegung der anderen seriellen Schnittstellen (RS-422, RS-485, TTY) ergibt sich hieraus.

Beim Anschluß der TTY-Leitungen ist zu beachten:
Die Bezeichnungen (out) und (in) geben nur die Richtung der Datenübertragung und *nicht* die Richtung des Stromes wieder. Für den Anschluß der TTY-Signale ist das Schaltbild im Kapitel '*Anschluß der verschiedenen seriellen Schnittstellen an den RJ45-Buchsen*' hilfreich.

Stecker-Pin		**Signalbelegung**				
RJ45-Buchse	**DSUB25-Buchse**	**RS-232**	**RS-422**	**RS-485**	**TTY-passiv**	**TTY-aktiv**
1	1	-	-	-	-	-
2	14	-	Rx- (out)	Rx/Tx-	Rx- (out)	[GNDA]
3	3	RxD Data Output	-	-	[I1+]	Rx+ (out)
4	16	-	Tx+ (in)	TERM+ *1)	Tx+ (in)	Tx- (in)
5	7	GND	GND	GND	GND	GND
6	2	TxD Data Input	Rx+ (out)	Rx/Tx+	Rx+ (out)	Rx- (out)
7	4	RTS Handshake Input	GND	GND	[I2+]	Tx+ (in)
8	5	CTS Handshake Output	Tx- (in)	TERM- *1)	Tx- (in)	[GNDA]

*1)... Die Pins 4 und 8 der RJ45-Buchsen (P200/P230) führen beim RS-485-Betrieb auf ein Abschlußwiderstandsnetzwerk, das sich auf dem Piggyback befindet. Um den Abschluß zu aktivieren, ist das Signal Rx/Tx+ mit TERM+ und das Signal Rx/Tx- mit TERM- zu verbinden.

[ ]... Die in eckigen Klammern angegebenen Signale sind zwar belegt, werden aber für den Betrieb der Schnittstelle nicht benötigt.



Steckerbelegung

Die folgende Tabelle gibt die Steckerbelegung des 25-poligen DSUB-Steckers (Stifte) an, daß die CBM-SIO4 im RS-232-Betrieb als Terminal (DEE), also Sender arbeiten soll. Die Belegung der anderen seriellen Schnittstellen (RS-422, RS-485, TTY) ergibt sich aus der RS-232-Zuordnung.

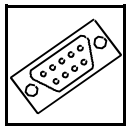
Achtung: Diese Belegung ist *nicht kompatibel* zur Steckerbelegung der entsprechenden Schnittstellen aus der vorangegangenen Tabelle! Die Steckerbelegung ist *nur* für die *RS-232*-Signale kompatibel zur vorangegangenen Belegung (DEE-DÜE-Verbindung)!

Stecker-Pin		**Signalbelegung**				
RJ45-Buchse	**DSUB25-Stifte**	**RS-232**	**RS-422**	**RS-485**	**TTY-passiv**	**TTY-aktiv**
1	1	-	-	-	-	-
2	14	-	Tx- (out)	Rx/Tx-	Tx- (out)	[GNDA]
3	2	TxD Data Output	-	-	[I1+]	Tx+ (out)
4	16	-	Rx+ (in)	TERM+ *1)	Rx+ (in)	Rx- (in)
5	7	GND	GND	GND	GND	GND
6	3	RxD Data Input	Tx+ (out)	Rx/Tx+	Tx+ (out)	Tx- (out)
7	5	CTS Handshake Input	-	-	[I2+]	Rx+ (in)
8	4	RTS *2) Handshake Output	Rx- (in)	TERM- *1)	Rx- (in)	[GNDA]

*1)... Die Pins 4 und 8 der RJ45-Buchsen (P200/P230) führen beim RS-485-Betrieb auf ein Abschlußwiderstandsnetzwerk, das sich auf dem Piggyback befindet. Um den Abschluß zu aktivieren, ist das Signal Rx/Tx+ mit TERM+ und das Signal Rx/Tx- mit TERM- zu verbinden.

[ ]... Die in eckigen Klammern angegebenen Signale sind zwar belegt, werden aber für den Betrieb der Schnittstelle nicht benötigt.

*2)... Von einigen Modems (Daten von CBM-SIO4 -> Terminal) wird ein DTR-Signal benötigt. Ist dieses der Fall, so kann das DTR-Signal erzeugt werden, indem im Stecker das RTS-Signal auf den DTR-Pin gebrückt wird. Bei einem 25-poligem DSUB-Stecker ist in diesem Fall Pin 4 auf Pin 20 zu brücken.



4.5 Spannungszuführung (X101, UEGM)

Die Spannungszuführung erfolgt über die im Gehäuse integrierten UEGM-Schraubverbinder. Sie können Leitungen mit einem Querschnitt von bis zu 2,5 mm² aufnehmen.

Die Schraubverbinder sind an beiden Seiten des Gehäuses gleich belegt. Sie können alternativ genutzt werden. Der mittlere Kontakt ist jeweils für +24V vorgesehen, die beiden äußeren für GND.

Anmerkung: Es ist **nicht zulässig**, die 24V-Versorgungsspannung ‘durchzuschleifen’, d.h. die eine Seite als 24V-Eingang und die andere Seite als 24V-Ausgang zu verwenden, um z.B. weitere Geräte zu versorgen!

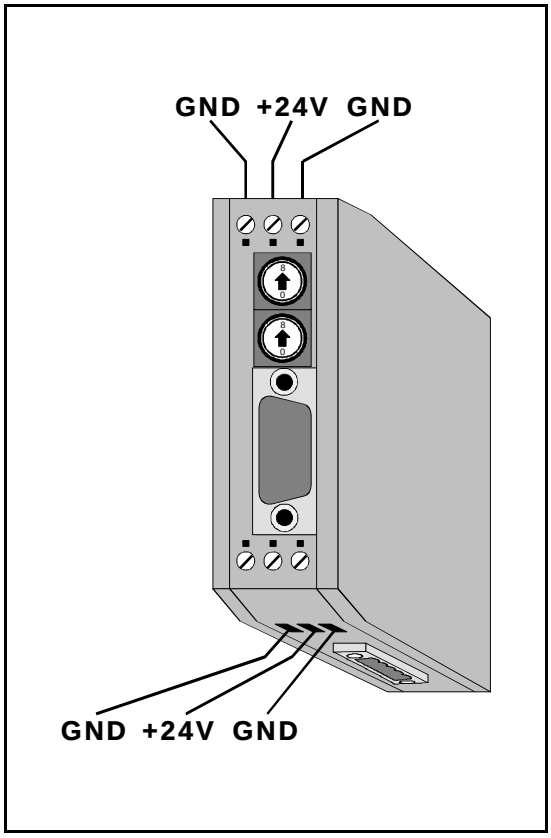


Abb 4.5.1: Spannungszuführung CAN-CBM-SIO1-Modul und CAN-CBM-PLC/331-1-Modul

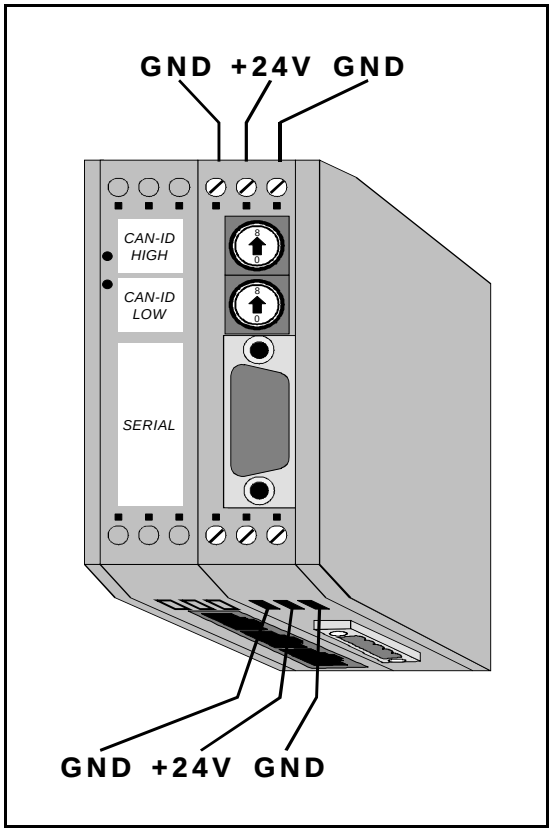


Abb. 4.5.2: Spannungszuführung CAN-CBM-SIO4-Modul

5. Konfiguration des CAN-CBM-PLC/331-1/-2 Moduls

Das Kapitel beschreibt die Vorgehensweise für die Inbetriebnahme und Konfiguration des CAN-CBM-PLC/331-1/-2 Moduls mit der CoDeSys Programmierumgebung.
Die CoDeSys-Software wird mit einer Online-Hilfe geliefert, die die vielfältigen Möglichkeiten von CoDeSys beschreibt.
Weitere Informationen über CANopen enthält die Spezifikation CANopen CiA Draft Standard 301.

Um das CAN-CBM-PLC/331-1/-2-Modul zu konfigurieren, ist wie folgt vorzugehen:

1. Importieren der verschiedenen Dateien: Installieren des Target Support Packages mit dem Installationsprogramm ***Install Target.exe***. Kontrollieren Sie, ob die EDS-Dateien der gewünschten Module im Unterverzeichnis: `%CoDeSys%\Targets\ESD\ESD_CAN-Module\` des Bibliotheksverzeichnisses liegen. Importieren Sie benötigte EDS-Dateien gegebenenfalls.

2. Starten der CoDeSys Entwicklungsumgebung.

3. Konfiguration: Im Menü ***Datei*** den Menüpunkt ***Neu*** wählen. Es erscheint die in der folgenden Abbildung dargestellte Dialogbox ***Zielsystem Einstellungen***.

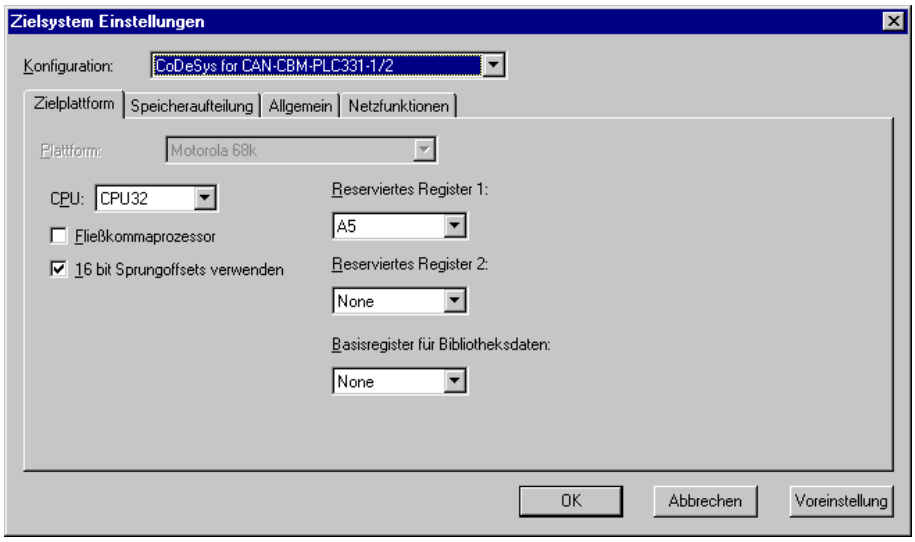


Abb.1: Einstellungen des Zielsystems

Die ***Konfiguration*** muß auf 'CoDeSys for CAN-CBM-PLC/331-1/2' eingestellt sein. Durch die Auswahl dieses 'Targets' wird die plattformspezifische Grundkonfiguration geladen.

CPU auf 'CPU32' einstellen. Bestätigen Sie mit ***OK***.

Kontrollieren Sie das aktuelle Projektverzeichnis und das Library-Verzeichnis. Wählen Sie dafür unter dem Menü ***Projekt***, Menüpunkt ***Optionen*** und weiter Menüpunkt ***Verzeichnisse*** und kontrollieren Sie die Pfadangaben unter ***Bibliotheken*** (z.B.: C:\codesys\library) und ***Übersetzungsdateien*** (z.B.: C:\CoDeSys).

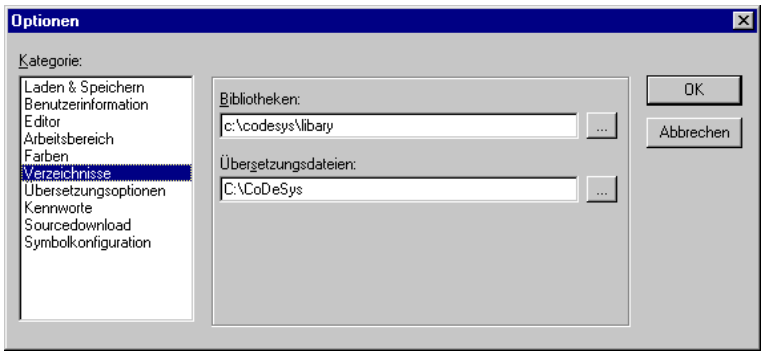


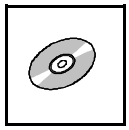
Abb.2: Bibliotheken und Übersetzungsdateien kontrollieren

4. Neuer Baustein: Bestätigen Sie Ihre Auswahl im ***Zielsystem Einstellungen*** mit ***OK***, öffnet sich die Dialogbox ***Neuer Baustein:***

Abb.3: Dialogbox ***Neuer Baustein***

Der Baustein PLC_PRG ist speziell vordefiniert und wird automatisch für jedes neue Projekt angelegt. Er darf nicht gelöscht oder umbenannt werden (gilt nicht bei Verwendung von Taskkonfiguration). Weitere Informationen dazu finden Sie in der CoDeSys Online-Hilfe.

Bestätigen Sie die Einstellungen ohne weitere Änderung mit ***OK***.



5. Auswahl des CAN-Master:

Wechseln Sie in das Register ***Resourcen*** (Register, linke untere Bildschirmecke).

Wählen Sie den Menüpunkt ***Steuerungskonfiguration*** aus.

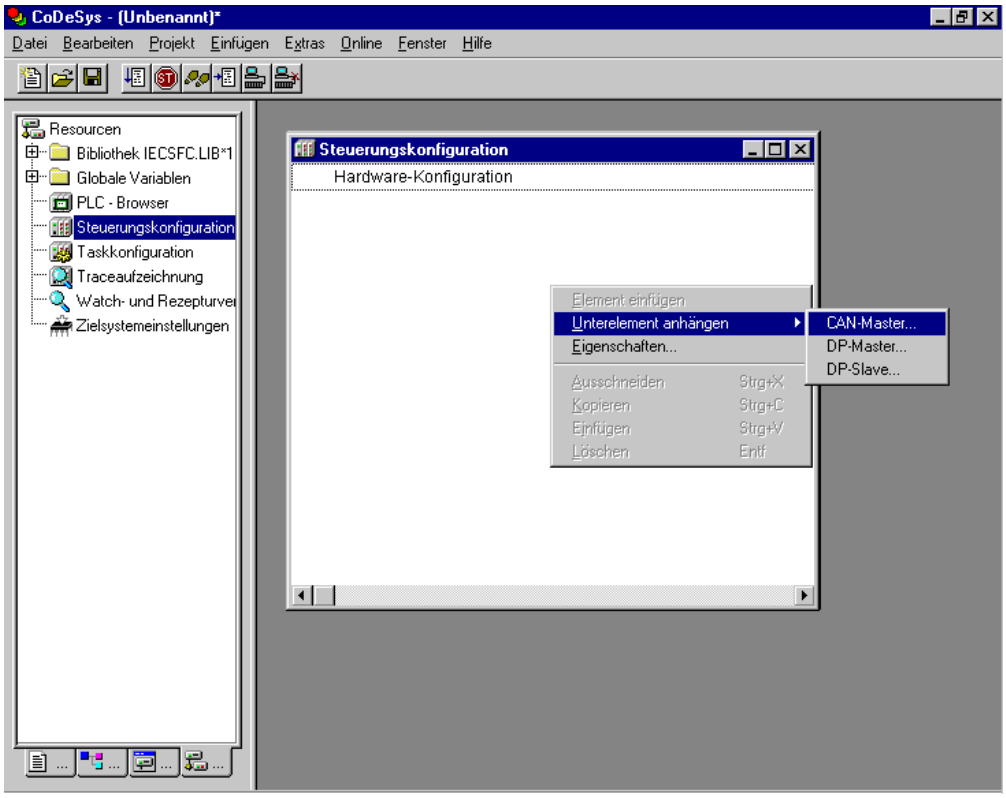


Abb.4: CAN-Master auswählen

Es erscheint das Feld ***Steuerungskonfiguration*** auf dem Bildschirm. Klicken Sie mit der rechten Maustaste auf das Feld und wählen Sie den Menüpunkt ***Unterelement anhängen*** und dann ***CAN-Master***.

6. CAN-Eigenschaften auswählen:

In der erscheinenden Dialogbox können Sie jetzt die gewünschten CAN-Eigenschaften eingeben:

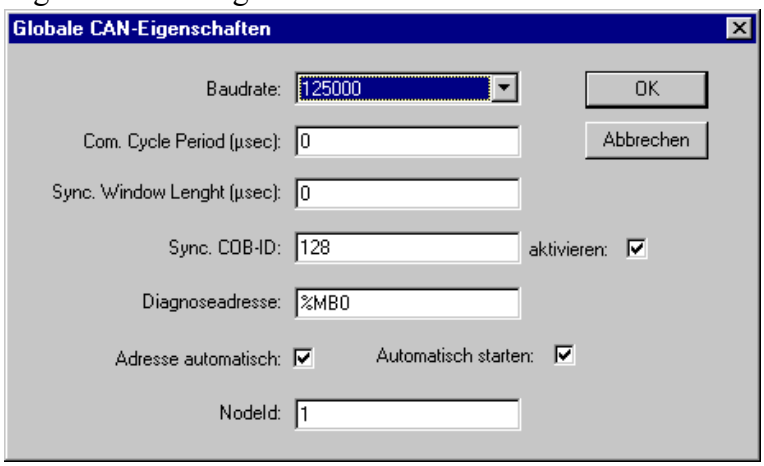


Abb.5: Globale CAN-Eigenschaften einstellen

Die Einstellungen der aufgelisteten Parameter sind abhängig von der jeweiligen Anwendung. Weitere Informationen finden Sie in der CoDeSys-Online Hilfe.

***Baudrate*:** Stellen Sie die für die Übertragung gewünschte Baudrate ein (hier 125 kBaud)

***Com. Cycle Period*:** Zykluszeit für Sync. Telegramm, d.h. Zeit, die zwischen dem Absenden von zwei SYNC Telegrammen des SYNC-Masters vergeht.
Die Zykluszeit ist abhängig von Slaves, Bus-geschwindigkeit und interner Datenverarbeitungsrate.

Achtung! Die ***Com.Cycle Periode*** muß größer als die ***Sync. Window Length*** sein um sicherzustellen, dass alle Busteilnehmer die synchronen PDO's (Process Data Objects) empfangen haben. Siehe auch Abb.6

Sync.Windows Length*:** Gibt die Zeit an, die vom Absenden einer SYNC vergeht, bis alle synchronen PDO's gesendet worden sind. Da sie kleiner ist als die ***Com.Cycle Periode wird so sichergestellt, dass alle geforderten Daten übertragen werden, bevor ein neues SYNC Telegramm gestartet werden kann. Siehe Abb.6

Achtung! Sind die Felder ***Com. Cycle Period*** und ***Sync.Windows Length*** mit '0' belegt, werden keine SYNC Telegramme verschickt.

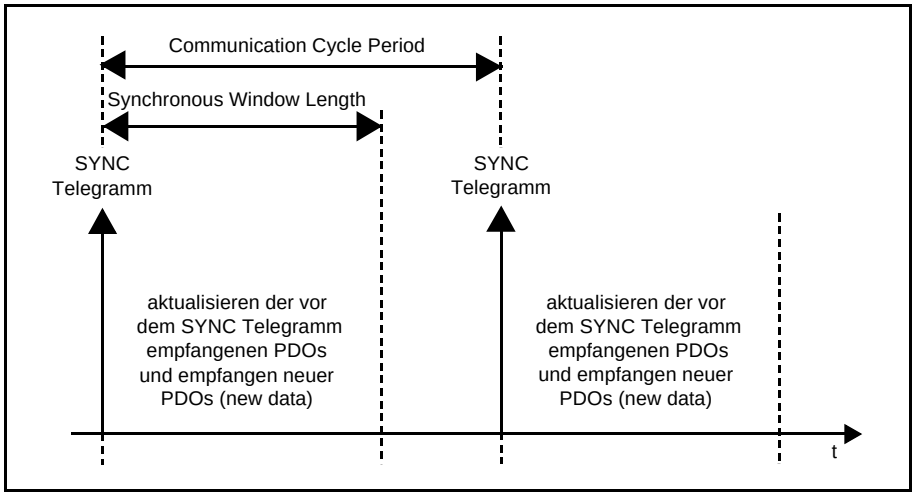


Abb.6: Bus Synchronisation

***Sync. COB-ID*:** Identifier unter dem SYNC Telegramme gesendet oder empfangen werden.

***Diagnoseadresse*:** Hier muß eine Merkadresse angegeben werden, unter der die Diagnosedaten abgelegt werden.

***Node ID*:** Identifier der CAN-CBM-PLC/331-1/2 (zwischen 1 und 127, dezimale Eingabe).

Bestätigen Sie ihre Auswahl mit ***OK***, wird der CAN-Master im Feld ***Steuerungskonfiguration*** in das Konfigurationsschema unter Hardwarekonfiguration aufgenommen (Siehe Abb.7. 'Unterelemente anhängen').

Weitere Informationen und Erläuterungen finden Sie in der CANopen Spezifikation 'CANopen CiA Draft Standard 301' Kap. 9.3.1

7. Unterelement einfügen:

Nach der Konfiguration des Masters wird nun das restliche CAN-Netzwerk zusammengestellt und konfiguriert. Um weitere Elemente einzufügen, klicken Sie mit der rechten Maustaste auf den eingefügten CAN-Master und gelangen über den Menüpunkt ***Unterelemente anhängen*** zu einer Auswahl der Module.

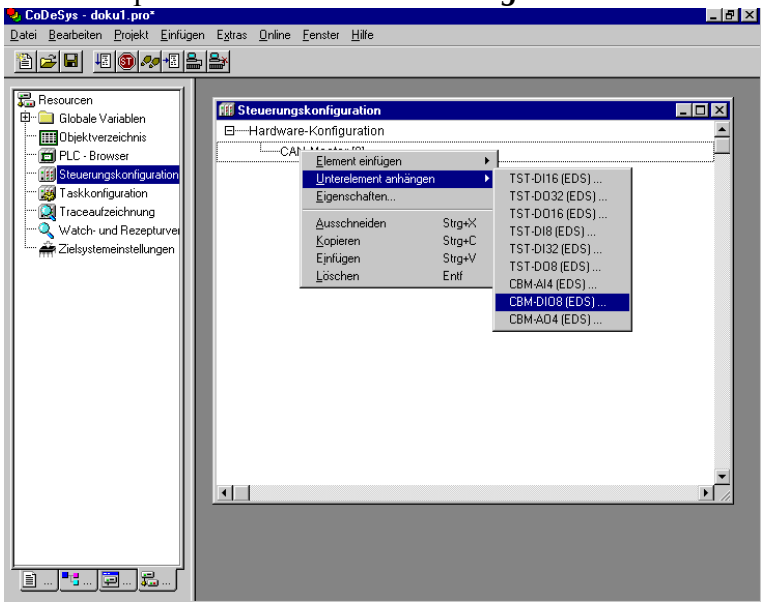


Abb.7: Unterelemente anhängen

Klicken Sie das gewünschte Modul (hier z.B. CBM-DIO8) mit der linken Maustaste an, erscheint eine Dialogbox (siehe Abb. 8) in der Sie die Eigenschaften des ausgewählten CAN-Teilnehmers festlegen können.

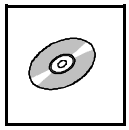
Ist für den gewünschten CAN-Teilnehmer kein Eintrag bzw. keine EDS-Datei vorhanden, kann durch die TST-Dateien die nicht verfügbare EDS-Datei ersetzt werden. Die TST-Dateien sind für einfache Anwendungen konfigurierte EDS-Dateien. Die Endung des Namens der Test-Dateien (TST-Dateien) erklärt dabei die jeweilige Funktion der Datei:

TST-*xxyyzz*

Stamm des Namens:	**TST-**	
folgender Buchstabe:	***xx***...	D, A (Digital, Analog)
folgende Buchstaben:	***yy***...	I, O, IO (Input, Output, In/Output)
folgende Zahlen:	***zz***...	gibt die Anzahl der Kanäle z.B.:8, 16, 32 oder 64

Beispiel:

TST-DI8 (EDS)	Digitaler Input, 8 Bits
TST-DO32 (EDS)	Digitaler Output, 32 Bits



8. Basisparameter:

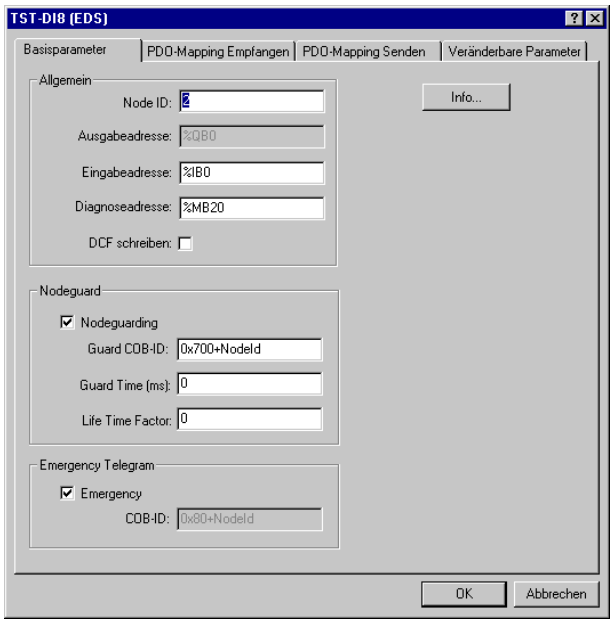


Abb.8: Basisparameter der Unterelemente einstellen

Geben Sie die folgenden Parameter entsprechend Ihrer Applikation ein.
Eine ausführliche Beschreibung der Parameter entnehmen Sie bitte dem Kapitel ‘Basisparameter eines CAN-Moduls’ der CoDeSys-Online Hilfe.

Node ID: Identifier des CAN Slaves

Eingabeadresse: Adresse unter der das Modul vom Anwenderprogramm angesprochen wird.

Diagnoseadresse: Adresse unter der die Diagnosedaten abgelegt werden.

DCF schreiben: Erstellen einer DCF-Datei nach dem Einfügen einer EDS-Datei, wenn aktiviert.

Alle Prozeßdaten im CAN-Netzwerk werden über den Eingangs- und Ausgangs-Adress-Bereich des CAN-CBM-PLC/331-1/2-Moduls gelesen oder geschrieben. (Über z.B.: %IB4 ... Eingangsbyte 4, %QB6 ... Ausgangsbyte 6).
CAN-spezifische Daten z.B.: Identifier, RTR... werden im Anwendungs-programm selber nicht verwendet.

Werden für die Überwachung der Teilnehmer die Option ***Nodeguarding*** und das ***Emergency-Telegramm*** gewünscht, aktivieren Sie diese. Weitere Informationen dazu sowie zu den Menüpunkten ***PDO-Mapping empfangen***, ***PDO-Mapping senden*** und ***Veränderbare Parameter*** entnehmen Sie bitte ebenfalls der CoDeSys Online-Hilfe.
Bestätigen Sie Ihre Auswahl mit ***OK***

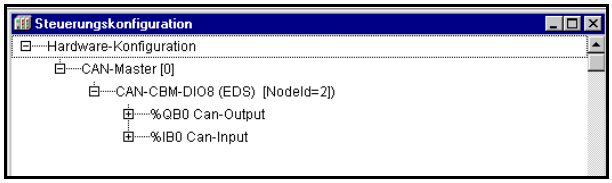


Abb.9: Steuerungskonfiguration

Das ausgewählte Modul (hier CAN-CBM-DIO8) wird nun im Fenster ***Steuerungskonfiguration*** in dem Konfigurationsschema als Unterelement angezeigt. Weitere Informationen über das jeweilige Element, wie die Eingabe- und Ausgabeadresse, erhalten Sie, wenn Sie mit der linken Maustaste auf das vorangestellte Pluszeichen klicken.

9. Weitere Module einfügen:

Um weitere Module einzubinden, wiederholen Sie die unter Punkt 7. und 8. beschriebenen Schritte.

Beispiel: Als weiteres Modul wird das Modul CBM-AO4 (EDS) wie unter Punkt **7.'Unterelemente anhängen'** beschrieben, ausgewählt.
Es öffnet sich die Dialogbox Eigenschaften ***CAN-CBM-AO4***:

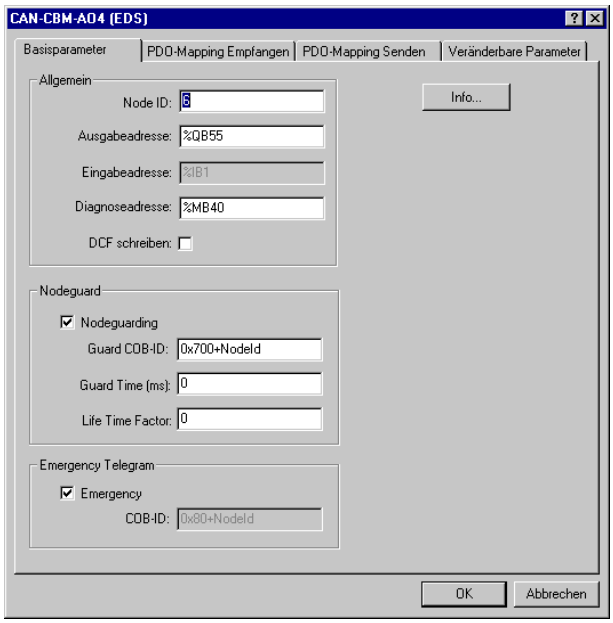
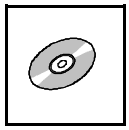


Abb.10: Beispiel CAN-CBM-AO4

Der Identifier des gewünschten Moduls hat die Node ID = 6,
die Ausgabeadresse: %QB55 (das Ausgangsbyte 55),
die Diagnoseadresse: %MB40 (das Byte an der Speicherstelle im Merker 40),
die Guard COB-ID ergibt sich aus 0x700+Node ID (hier NodeID= 6)



Bestätigen Sie mit OK, erscheint das folgende Fenster:

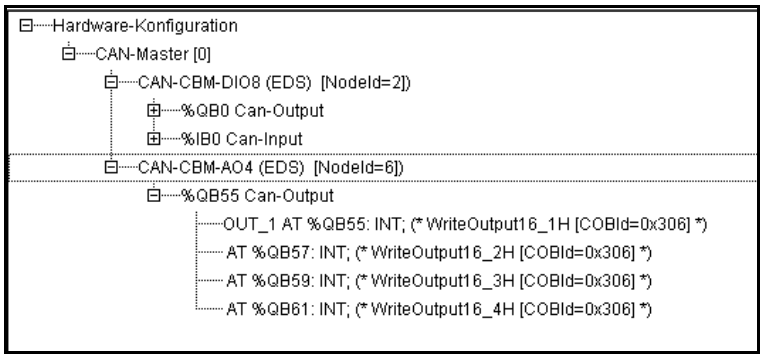


Abb.11: Konfigurationsbeispiel

Das Modul CAN-CBM-AO4 mit der Node ID = 6 ist nun aufgenommen. Durch Mausklick auf die Pluszeichen erhalten Sie weitere Informationen über die Konfiguration. Die Ausgabeadresse des ersten Kanals beträgt:%QB55
Zu jedem Kanal des gewählten Moduls wird folgende Funktionsbeschreibung ausgegeben:

Funktionsbe-schreibung	*name* AT*%adresse*:Datentyp;(*Kommentar*)
Beispiel	OUT_1 AT%QB55:INT;(*WriteOutput16_1H [COBId0x306]*)

name: Wird ein weiteres Ausgangsmodul vor dem CAN-CBM-AO4-Modul eingefügt, erhöht sich die Ausgabeadresse entsprechend der Anzahl der Ausgänge des neu eingefügten Moduls automatisch. Um zu verhindern, dass nun alle Programme, in denen diese Adresse erscheint, ebenfalls geändert werden müssen, kann der globalen Variablen ein Name zugewiesen werden. Durch Mausklick auf AT erscheint ein kleines Eingabefenster. Hier läßt sich ein Name für die globale Variable, die auf der Adresse %QB55 liegt, eingeben. Dieser Name kann nun, unabhängig von anderen Modulen, für alle Programmierungen verwendet werden.

AT%*adresse*: ausgewählte Ausgabeadresse, %QB55

Datentyp: hier vom Typ Integer

(*Kommentar*): WriteOutput16_1H: Ausgabe = 16 Bit auf Kanal 1(1H), unter der COB-ID = 0x306 werden die Prozessdaten im CAN-Netzwerk gesendet.

6. Korrekte Verdrahtung galvanisch getrennter CAN-Netze

Generell sind bei der Verdrahtung sämtliche gültigen Richtlinien (DIN, VDE) bzgl. EMV-gerechtem Aufbau, Leitungsführung, Leiterquerschnitte, zu verwendende Materialien, Mindestabstände, Blitzschutz etc. zu beachten.

Die folgenden **Grundregeln** für die CAN-Bus Verdrahtung sollten unbedingt beachtet werden:

1	Ein CAN-Netz darf sich nicht verzweigen (Ausnahme: kurze Stichleitungen) und muß an beiden Enden mit dem Wellenwiderstand der Leitung (in der Regel 120 S ±10%) abgeschlossen werden (zwischen den Signalen CAN_L und CAN_H und **nicht** gegen GND)!
2	Eine CAN-Datenleitung benötigt **zwei verdrillte** Adern (Twisted Pair) und eine Leitung zur Mitführung des Bezugspotentials (CAN_GND)! Hierzu sollte die Abschirmung des Kabels verwendet werden!
3	Das mitgeführte Bezugspotential CAN_GND muß an **einem** Punkt mit dem Erdpotential (PE) verbunden werden. Es muß genau **eine** Verbindung mit Erde hergestellt werden!
4	Die Baudrate muß an die Leitungslänge angepaßt werden.
5	Stichleitungen sind so kurz wie möglich zu halten (l < 0,3 m)!
6	Bei doppelt abgeschirmten Leitungen muß der äußere Schirm an **einem** Punkt mit dem Erdpotential (PE) verbunden werden. Es darf nicht mehr als **einen** Anschluß an Erde geben.
7	Es ist ein geeigneter Leitungstyp (Wellenwiderstand ca. 120 S ±10%) zu verwenden und der Spannungsabfall auf der Leitung ist zu beachten!
8	Die CAN-Leitungen sollten nicht in unmittelbarer Nähe von Störquellen verlegt werden. Läßt sich dies nicht vermeiden, so sind doppelt abgeschirmte Leitungen vorzuziehen.

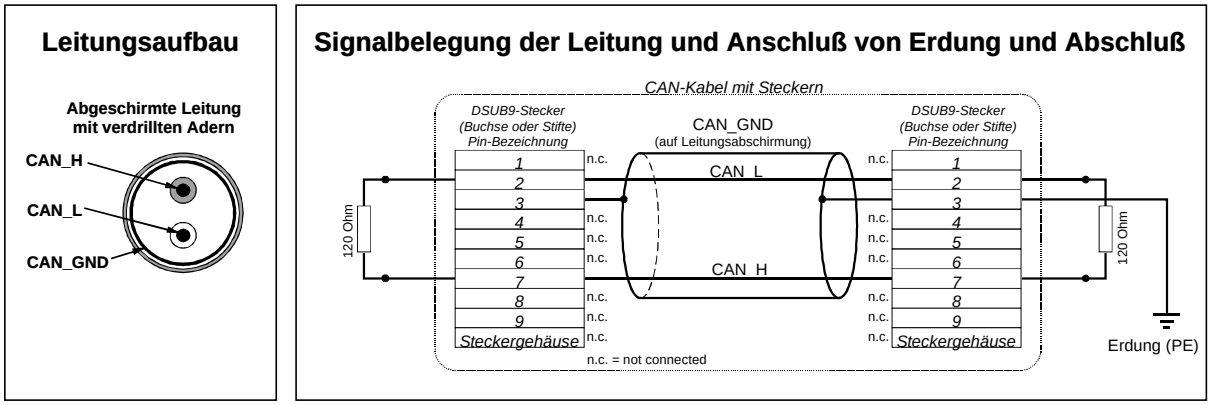


Abb.: Aufbau und Anschluß der Leitung

Verkabelung

- bei Geräten, die nur einem CAN-Stecker besitzen, T-Stück und Stichleitung (kürzer als 0,3 m) verwenden (als Zubehör lieferbar)

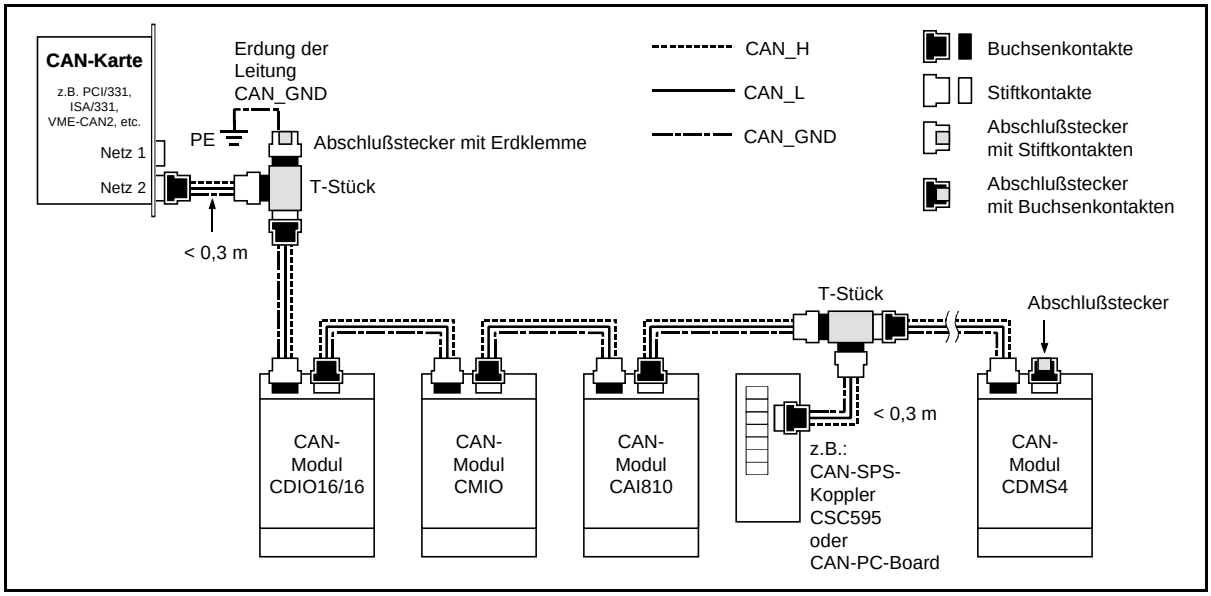


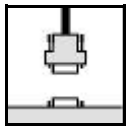
Abb.: Beispiel für korrekte Verdrahtung (bei Verwendung einfach abgeschirmter Leitungen)

Abschlußwiderstand

- **externen** Abschlußstecker verwenden, weil dieser später leichter auffindbar ist!
- 9-polige DSUB-Abschlußstecker mit Stift- oder Buchsenkontakten und Erdungsklemme sind als Zubehör erhältlich

Erdung

- CAN_GND muß in der CAN-Leitung mitgeführt werden, weil die einzelnen esd-Module galvanisch voneinander getrennt sind!
- CAN_GND muß an **exakt einem** Punkt im Netz mit dem Erdpotential (PE) verbunden werden!
- jeder CAN-Teilnehmer ohne galvanisch getrenntes Interface wirkt wie eine Erdung, darum: maximal einen Teilnehmer ohne Potentialtrennung anschließen!
- Erdung kann z.B. an einem Abschlußstecker vorgenommen werden



Leitungslänge

- Optokoppler verzögern die CAN-Signale. Durch den Einsatz schneller Optokoppler und den Test jedes Boards bei 1 MBit/s kann esd jedoch eine erreichbare Länge von 37 m bei 1 MBit/s garantieren. Voraussetzung hierfür ist ein abgeschlossenes Netz ohne Impedanzstörungen, wie z.B. längere Stichleitungen. (Ausnahme: CAN-CBM-DIO8, -AI4, und -AO4 hier nur 10 m bei 1 MBit/s.)

Bit-Rate [kBit/s]	typische Werte der erreichbaren Leitungslänge **mit esd-Interface** l_{max} [m]	**CiA-Empfehlungen** (07/95) für erreichbare Leitungslängen 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

Tabelle: Erreichbare Leitungslängen in Abhängigkeit von der Bitrate beim Einsatz von esd-CAN-Interfaces

Beispiele für geeignete Leitungstypen

Hersteller	Leitungstyp
U.I. LAPP GmbH & Co. KG Schulze-Delitzsch-Straße 25 70565 Stuttgart	UNITRONIC ®-BUS LD, UNITRONIC ®-BUS FD P LD
metrofunk KABEL-UNION GmbH Postfach 410109 12111 Berlin	LiYCY 2 x 0,38 mm², LiYCY 2 x 0,5 mm², LiYCY 2 x 0,75 mm², LiYCY 2 x 1,0 mm², 1P x AWG 22 C, 1P x AWG 20 C

Hersteller	Leitungstyp
Alcatel Kabelmetal Kabelkamp 20 30179 Hannover	DUE 4401, DUE 4001, DUE 4402
ConCab Kabel GmbH Äußerer Eichwald 74535 Mainhardt	1 x 2 x 0,22 mm² Best-Nr. 93022016 (UL approved)

7. Stromlaufpläne

CAN-CBM-SIO
CAN-CBM-SIO4

CAN - RS-232,
RS-422, RS-485
oder TTY-Interface

Handbuch der
modulspezifischen Software

Der Inhalt dieses Handbuches wurde mit größter Sorgfalt erarbeitet und geprüft. esd übernimmt jedoch keine Verantwortung für Schäden, die aus Fehlern in der Dokumentation resultieren könnten. Insbesondere Beschreibungen und technische Daten sind keine zugesicherten Eigenschaften im rechtlichen Sinne.

esd hat das Recht, Änderungen am beschriebenen Produkt oder an der Dokumentation ohne vorherige Ankündigung vorzunehmen, wenn sie aus Gründen der Zuverlässigkeit oder Qualitätssicherung vorgenommen werden oder dem technischen Fortschritt dienen.

Sämtliche Rechte an der Dokumentation liegen bei esd. Die Weitergabe an Dritte und Vervielfältigung jeder Art, auch auszugsweise, sind nur mit schriftlicher Genehmigung durch esd gestattet.

esd electronic system design gmbh
Vahrenwalder Str. 205
D-30165 Hannover

Tel.: 0511/37298-0
FAX : 0511/37298-198
Email: info@esd-electronics.com
Internet: http://www.esd-electronics.com

Handbuch-Datei:	I:\TEXTE\DOKU\MANUALS\CAN\CBM\SIO-331\CSIO-11S.MA6
Datum der Druckvorlagenerstellung:	23.12.1999

Software-Version:	sio4_V1.0aE0

Änderungen in den Kapiteln

Die hier aufgeführten Änderungen im Anwenderhandbuch betreffen sowohl Änderungen in der Firmware als auch reine Änderungen in der Beschreibung der Sachverhalte.

Handbuch-Rev.	Kapitel	Änderungen gegenüber Vorversion
1.1	4.1.2	Beispiel korrigiert.
	-	-

Weitere technische Änderungen vorbehalten.

Inhalt

Seite

1. Übersicht . . . . . 1 - 1
1.1 Was steht wo? . . . . . 1 - 1
1.2 Default-Einstellungen . . . . . 1 - 3

2. Beschreibung des Datentransfers . . . . . 2 - 1
2.1 Serielle Schnittstellen senden Daten . . . . . 2 - 1
2.2 Serielle Schnittstellen empfangen Daten . . . . . 2 - 3

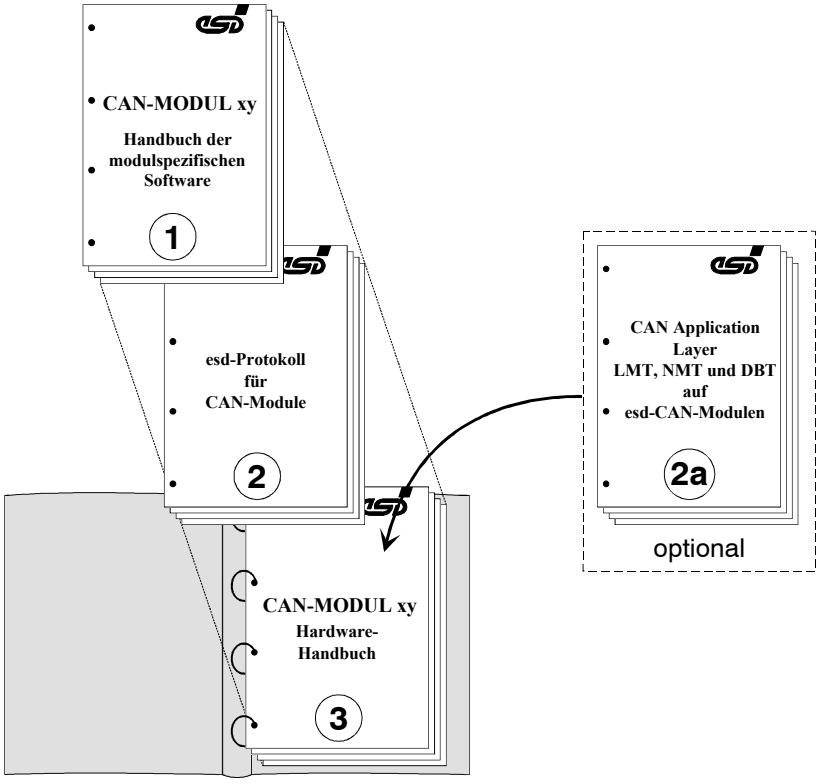
3. User-Parameter des CAN-CBM-SIO und CAN-CBM-SIO4-Moduls . . . . . 3 - 1
3.1 First Tx-activate Delay (Parameter 0) . . . . . 3 - 2
3.2 CAN-Tx-Mode (Parameter 1, 9, 11, 19, 21) . . . . . 3 - 2
3.3 Serial-Mode (Parameter 2, A, 12, 1A, 22) . . . . . 3 - 5

4. Beispiele . . . . . 4 - 1
4.1 Betrieb mit den Default-Parametern . . . . . 4 - 1
4.1.1 Rahmenbedingungen, Zielsetzung . . . . . 4 - 1
4.1.2 Vorgehensweise . . . . . 4 - 1
4.1.2.1 Identifier einstellen . . . . . 4 - 1
4.1.2.2 Senden der Daten auf die serielle Schnittstelle . . . . . 4 - 2
4.1.2.3 Empfangen der Daten von der seriellen Schnittstelle . . . . . 4 - 2
4.2 Ändern der Baudrate . . . . . 4 - 4

1. Übersicht

1.1 Was steht wo?

Die Beschreibung der esd-CAN-Module ist in drei Handbücher aufgeteilt, die in einem gemeinsamen Ringordner ausgeliefert werden.



Das erste Handbuch befaßt sich mit Software-Eigenschaften und Parametern, die nur für dieses Modul von Bedeutung sind und ist unabhängig vom gewählten CAN-Protokoll einsetzbar:

CAN-CBM-SIO CAN-CBM-SIO4 **serielle Schnittstellen** **Handbuch der modulspezifischen Software**

Hier werden z.B. die Funktionen der typ-spezifischen Firmware, die Belegung der Identifier und die Belegung der User-Parameter beschrieben.

In diesem Handbuch werden die esd-CAN-Module CAN-CBM-SIO4 und CAN-CBM-SIO beschrieben. Die beiden Module sind im wesentlichen gleich aufgebaut. Das CAN-CBM-SIO Modul besitzt im Gegensatz zum CAN-CBM-SIO4-Modul jedoch nur eine serielle Schnittstelle, die auf Kanal 1 bestückt ist. Da es nur eine Schnittstelle besitzt, verfügt es nur über zwei CAN-Identifier, bzw. über zwei COB-IDs.
Im folgenden wird generell das CAN-CBM-SIO4-Modul beschrieben. Treten Differenzen zum CAN-CBM-SIO-Modul auf, so werden diese an den entsprechenden Stellen beschrieben.

Übersicht

Das zweite Handbuch beinhaltet allgemeine Software-Beschreibungen, die für alle esd-CAN-Module, die mit dem gleichen Protokoll betrieben werden, gültig sind.

Für die Module sind zwei verschiedene Protokolle lieferbar: Das esd-CAN-Protokoll und das CMS-Protokoll. Die Protokolle sind unabhängig voneinander und werden alternativ eingesetzt. In Abhängigkeit vom implementierten Protokoll ist daher eines der folgenden beiden Handbücher für das Modul zut1reffend:

Das esd-CAN-Protokoll ist in dem Handbuch

esd-Protokoll für CAN-Module

beschrieben. Es bietet dem Anwender die Möglichkeit, die esd-CAN-Module über einen Initialisierungs-Identifier ($700) zu parametrieren. Über dieses Protokoll können den Modulen Identifier zugeordnet werden, User-Parameter gesetzt und Watchdog-Funktionen aktiviert werden.

Alternativ hierzu können die Module über das CMS-Protokoll gesteuert werden. Ist dieses Protokoll implementiert, so ist das zur CMS-Option lieferbare Handbuch

CAN Application Layer LMT, NMT und DBT auf esd-Modulen

hinzuzuziehen. Hier wird die Umsetzung der CMS-Dienste
des Layer Managements (LMT),
des Network Managements (NMT) und
des Identifier Distributors (DBT)
auf esd-CAN-Modulen erläutert.

Das dritte Handbuch enthält die Hardware-Beschreibung des Moduls. Es beinhaltet sowohl allgemeine als auch modulspezifische Erläuterungen zur Hardware. Hier finden sich z.B. Installationshinweise und Steckerbelegungen.

CAN-CBM-SIO CAN-CBM-SIO4
CAN - RS-232, RS-422, RS-485 oder TTY-Interface
Hardware-Handbuch

1.2 Default-Einstellungen

Die Default-Einstellungen des Moduls sind aktiv, wenn eine oder mehrere der folgenden Bedingungen zutrifft:

- Auf dem Modul wurde über das esd-CAN-Protokoll ein Default-RESET ausgelöst.
- Die Daten des I²C-EEPROMs sind nicht OK (z.B. EEPROM nicht bestückt).
- Die Stellung der Kodierschalter nach einem RESET oder Power-On war auf '00' gesetzt und wurde danach auf einen anderen Wert eingestellt.

Einzelne Parameter können verändert werden, ohne die Default-Einstellung anderer Parameter zu beeinflussen. Änderungen der Parameter bleiben nach einem RESET nur aktiv, wenn sie im EEPROM gespeichert wurden.

Default-Werte beim Betrieb des Moduls mit dem esd-CAN-Protokoll	
INIT-Id.	in allen Betriebsarten $700
Rx-Identifier, Tx-Identifier	RxId1, TxId1 -þ serieller Kanal 1 RxId2, TxId2 -þ serieller Kanal 2 RxId3, TxId3 -þ serieller Kanal 3 RxId4, TxId4 -þ serieller Kanal 4 RxId5, TxId5 -þ serieller Kanal 5 Die Default-Werte der Identifier entsprechen den Einstellungen über die Kodierschalter. Eine detaillierte Beschreibung der Einstellungen ist dem Hardware-Handbuch zu entnehmen. (*)
Module-No.	= Einstellung der Kodierschalter
CAN-Bitrate	= 125 KBit/s

(*) Da das CAN-CBM-SIO4-Modul fünf Tx- und fünf Rx-Identifier belegt, müssen die Identifier der nachfolgenden Module mindestens mit einem Offset von +10 gewählt werden, da sonst Überschneidungen der Identifier auftreten würden!

Tabelle 1.2.1: Default-Einstellungen des Moduls beim Betrieb mit dem esd-Protokoll

Achtung: Der Terminal-Schnittstelle (auf DSUB9) sind auf dem CAN-CBM-SIO4-Modul der Rx-Identifier RxId**5** und der Tx-Identifier TxId**5** zugeordnet. Auf dem CAN-CBM-SIO-Modul mit nur einer seriellen Schnittstelle sind der Terminal-Schnittstelle jedoch der Rx-Identifier RxId**1** und der Tx-Identifier TxId**1** zugeordnet.

Default-Werte beim Betrieb des Moduls mit dem CAL-Protokoll	
Manufacturer Name	ASCII 'esd_han'
Product Name	noch nicht definiert
Module-ID	= Einstellung der Kodierschalter
Module Name	noch nicht definiert
CAN-Bitrate	125 KBit/s
Nach einem Default-RESET ist zwingend ein *Configuration Download* zum Modul über das NMT-Protokoll erforderlich!	

Tabelle 1.2.2: Default-Einstellungen des Moduls beim Betrieb mit CAL

Default-Werte der User-Parameter (unabhängig vom eingesetzten Protokoll)		
First Tx-activate Delay	10.000 msec	
CAN-Tx-Mode (alle Kanäle)	$1411, d.h.	'MinChar', 'MaxChar' = 1, 'Inhibit-Time' = 20 ms
Serial-Mode (alle Kanäle)	$2273, d.h.	CTS* aktiv (Terminal-Schnittstelle: no CTS), 9600 Baud, 2 Stop-Bit (Terminal-Schnittstelle: 1 Stop-Bit) no Parity, 8 Bit/Character

Tabelle 1.2.3: Default-Einstellungen der User-Parameter des Moduls

Erläuterungen zu den Begriffen der User-Parameter in Tabelle 1.2.3:

First Tx-activate Delay... Verzögerungszeit nach einem RESET, bevor das Modul mit dem Senden von Nachrichten auf den CAN-Bus oder den seriellen Schnittstellen beginnt.

CAN-Tx-Mode... 'MinChar' und 'MaxChar' legen die minimale und die maximale Anzahl der Daten-Bytes fest, die innerhalb eines CAN-Frames auf dem CAN-Bus gesendet werden sollen. 'Inhibit-Time' gibt die Wartezeit des CAN-Controllers zwischen der letzten erfolgreichen Sendung auf dem CAN-Bus und dem Start der nächsten Sendungen an.

Serial-Mode... Mit Serial-Mode werden die seriellen Schnittstellen parametriert.

2. Beschreibung des Datentransfers

Die Pufferung der seriellen Daten zwischen dem CAN-Bus und den seriellen Schnittstellen erfolgt in beiden Datenrichtungen und für jeden seriellen Kanal über einen 256 Byte großen Ringspeicher. Die Daten, die zuerst eintreffen, werden auch zuerst wieder gesendet.

Ist der Ringspeicher gefüllt und treffen noch weitere Daten ein, gehen Daten verloren. Aus diesem Grund müssen die Übertragungsraten des CAN-Bus und der seriellen Schnittstellen aufeinander abgestimmt werden.

2.1 Serielle Schnittstellen senden Daten

Die Menge der Daten des CAN-Bus, die pro Zeit im Ringspeicher abgelegt werden, muß kleiner sein als die über die serielle Schnittstelle gesendete Datenmenge.

Die Anzahl der eintreffenden Daten pro Zeit hängt von der Anzahl der übertragenen Daten-Bytes, der Häufigkeit der Sendungen, der Bitrate des CAN-Bus und der zugeteilten Busfreigabe des CAN-Bus ab.

Treten während des Betriebs Datenverluste auf, dann sollten die Pausen zwischen den Sendungen vergrößert, oder/und die Anzahl der übertragenen Bytes verringert werden.

Die Daten werden vom CAN-Bus über die Rx-Identifier, bzw. über die COBs 1 bis 4 zum Modul übertragen. Die Anzahl der übertragenen Bytes kann frei gewählt werden.

esd-Prot.	CAL	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
RxId	COB								
RxId 1	COB 1	Daten für serielle Schnittstelle Kanal 1							
RxId 2	COB 2	Daten für serielle Schnittstelle Kanal 2							
RxId 3	COB 3	Daten für serielle Schnittstelle Kanal 3							
RxId 4	COB 4	Daten für serielle Schnittstelle Kanal 4							
RxId 5	COB 5	Daten für serielle Schnittstelle Kanal 5							

Tabelle 2.1.1: Empfangen der zu sendenden Daten über Rx-Identifier, bzw. COB 1 bis 5 (CAN-CBM-SIO4)

Achtung:
Der Terminal-Schnittstelle (auf DSUB9) sind auf dem CAN-CBM-SIO4-Modul der Rx-Identifier RxId**5** (COB5) und der Tx-Identifier TxId**5** (COB5) zugeordnet.
Auf dem CAN-CBM-SIO-Modul mit nur einer seriellen Schnittstelle sind der Terminal-Schnittstelle jedoch der Rx-Identifier RxId**1** und der Tx-Identifier TxId**1** zugeordnet !

Datentransfer

Beim Betrieb mit dem esd-Protokoll kann die Datenausgabe auf den seriellen Schnittstellen mit dem Supervisor-Command 'Suspend Module' gestoppt werden. Es werden alle vier Kanäle gleichzeitig angehalten. Mit dem Kommando 'Continue' werden die Kanäle gleichzeitig wieder gestartet.

Befindet sich das Module im Zustand 'Suspended' werden eintreffende Daten sowohl vom CAN-Bus als auch von den seriellen Schnittstellen im Ringspeicher abgelegt.

Der Datentransfer vom CAN-Bus zu den seriellen Schnittstellen hat prinzipiell folgenden zeitlichen Ablauf:

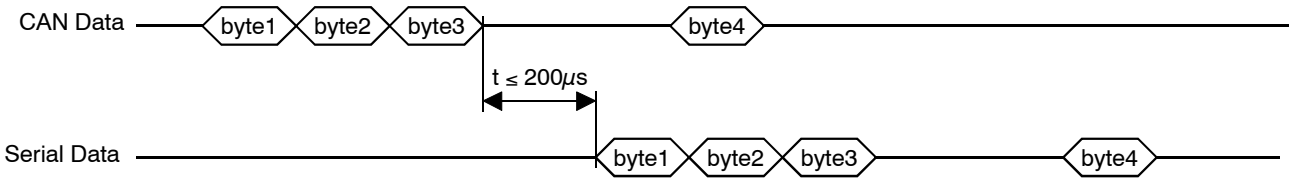


Abb. 2.1.1: Datentransfer CAN-Bus -> serielle Schnittstelle

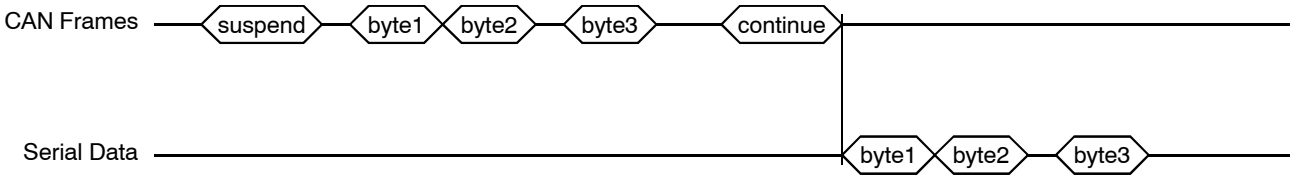


Abb. 2.1.2: Datentransfer CAN-Bus -> serielle Schnittstelle über 'Suspend' und 'Continue' gesteuert

2.2 Serielle Schnittstellen empfangen Daten

Die Menge der Daten der seriellen Schnittstelle, die pro Zeit im Ringspeicher abgelegt werden, muß kleiner sein als die über den CAN-Bus abführbare Datenmenge.

Der CAN-Bus schränkt den Datenfluß über die Belastung des Busses (Priorität), die CAN-Bus-Bitrate, die Häufigkeit der Sendungen und die Anzahl der übertragenen Bytes ein.

Das Modul bietet über die Parameter 'Inhibit-Time', 'MaxChar' und 'MinChar' Möglichkeiten, die letzten beiden Faktoren zu beeinflussen. Diese Parameter sind in dem User-Parameter 'CAN-Tx-Mode' zusammengefaßt.

Mit 'Inhibit-Time' wird die Wartezeit zwischen zwei Sendungen auf dem CAN-Bus festgelegt.

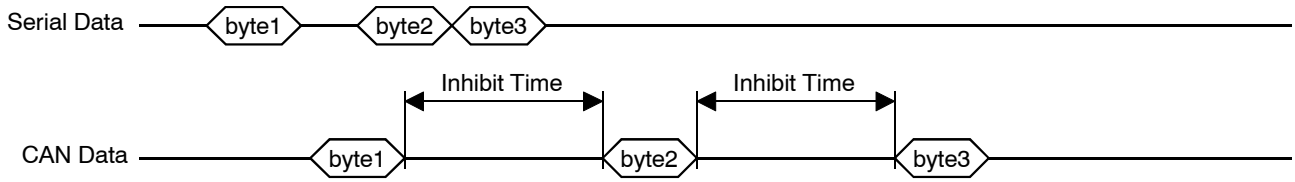


Abb. 2.2.1: Funktion der 'Inhibit-Time'

'MinChar' und 'MaxChar' legen die minimale und die maximale Anzahl der Daten-Bytes fest, die innerhalb eines CAN-Frames auf dem CAN-Bus gesendet werden sollen.

Die Daten werden vom Modul über die Tx-Identifier, bzw. über die COBs 5 bis 8 auf dem CAN-Bus gesendet.

esd-Prot.	CAL	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
TxId	COB								
TxId 1	COB 6	Daten von serieller Schnittstelle Kanal 1							
TxId 2	COB 7	Daten von serieller Schnittstelle Kanal 2							
TxId 3	COB 8	Daten von serieller Schnittstelle Kanal 3							
TxId 4	COB 9	Daten von serieller Schnittstelle Kanal 4							
TxId 5	COB 10	Daten von serieller Schnittstelle Kanal 5							

Tabelle 2.2.1: Übertragung der auf den seriellen Schnittstellen empfangenen Daten über Tx-Identifier, bzw. COBs (CAN-CBM-SIO4)

Das CAN-CBM-SIO-Modul besitzt nur den Tx-Identifier TxId1, bzw. das COB 1.

3. User-Parameter des CAN-CBM-SIO und CAN-CBM-SIO4-Moduls

Über die User-Parameter werden dem Modul die Parameter für die seriellen Schnittstellen und die Parameter 'Inhibit-Time', 'MaxChar' und 'MinChar' (CAN-Tx-Mode) zur Steuerung der CAN-Bus-Übertragungsrate übergeben.

Wird das Modul mit dem esd-Protokoll betrieben, so werden die User-Parameter mit dem Kommando 'Setzen User-Parameter' ($86) auf Byte 5 und 6 des INIT-Id ($700) übergeben.
Alle User-Parameter müssen immer als 16-Bit-Wert mit Byte 5 als höherwertigem Byte übergeben werden!

Ist das CMS-Protokoll implementiert, werden die User-Parameter über einen Configuration Download (NMT) gesetzt.

Die anschließende Tabelle gibt eine Übersicht über die User-Parameter des CAN-CBM-SIO4-Moduls. Beim CAN-CBM-SIO-Modul werden nur die Parameter $00...$02 benötigt.

User-Parameter No.	Parameter	Wertebereich	Default-Einstellung
$00	First Tx-activate Delay	$0000...$FFFF (0...65535 msec)	10.000 msec
$01	CAN-Tx-Mode Kanal 1	$0011...$FF88	$1411
$02	Serial-Mode Kanal 1	$0000...$88FF	$2273
$03	reserviert	-	-
$04...$08	nicht belegt	-	-
$09	CAN-Tx-Mode Kanal 2	$0011...$FF88	$1411
$0A	Serial-Mode Kanal 2	$0000...$88FF	$2273
$0B	reserviert	-	-
$0C...$10	nicht belegt	-	-
$11	CAN-Tx-Mode Kanal 3	$0011...$FF88	$1411
$12	Serial-Mode Kanal 3	$0000...$88FF	$2273
$13	reserviert	-	-
$14...$18	nicht belegt	-	-
$19	CAN-Tx-Mode Kanal 4	$0011...$FF88	$1411
$1A	Serial-Mode Kanal 4	$0000...$88FF	$2273
$1B	reserviert	-	-
$14...$20	nicht belegt	-	-
$21	CAN-Tx-Mode Kanal 5	$0011...$FF88	$1411
$22	Serial-Mode Kanal 5	$0000...$88FF	$2273
$23	reserviert	-	-

Tabelle 3.1.1: User-Parameter des CAN-CBM-SIO/CBM-SIO4-Moduls

3.1 First Tx-activate Delay (Parameter 0)

Parameter 0 übergibt die Verzögerungszeit die verstreichen soll, bevor das Modul nach einem RESET mit dem Senden von Daten auf dem CAN-Bus und den seriellen Schnittstellen beginnt.

Diese Verzögerungszeit soll sicherstellen, daß alle Module stabil auf dem CAN-Bus arbeiten, bevor das Modul mit den Übertragungen beginnt.

User-Parameter No. (= Sub-Command-No.)	Parameter	Wertebereich	Default-Einstellung
$00	First Tx-activate Delay	$0000...$FFFF (0...65535 msec)	10.000 msec

Tabelle 3.1.2: User-Parameter 0

3.2 CAN-Tx-Mode (Parameter 1, 9, 11, 19, 21)

Mit diesen User-Parametern werden die 'Inhibit-Time' und Parameter 'MaxChar' und 'MinChar' übergeben.

Mit 'Inhibit-Time' wird die Wartezeit zwischen zwei Sendungen auf dem CAN-Bus festgelegt.

'MinChar' und 'MaxChar' legen die minimale und die maximale Anzahl der Daten-Bytes fest, die innerhalb eines CAN-Frames auf dem CAN-Bus gesendet werden sollen. Ab Software-Rev. '17.7e' ist die Funktion des Parameters 'MinChar' erweitert worden. Hier kann jetzt alternativ eine Abbruchbedingung mit übergeben werden.

User-Parameter-No. (Sub-Command-No.)	Parameter	Wertebereich	Default-Einstellung
$01	CAN-Tx-Mode Kanal 1	$0011..$FF88	$1411
$09	CAN-Tx-Mode Kanal 2		
$11	CAN-Tx-Mode Kanal 3		
$19	CAN-Tx-Mode Kanal 4		
$21	CAN-Tx-Mode Kanal 5		

Tabelle 3.2.1: User-Parameter CAN-Tx-Mode

Die zwei Byte des Parameters CAN-Tx-Mode setzen sich wie folgt zusammen:

	CAN-Tx-Mode		
	Byte 5 des INIT-Id $700	Byte 6 des INIT-Id $700	
Parameter	Inhibit-Time	MaxChar	MinChar
Wertebereich	$00...$FF	$1...$8	$1...$8

Tabelle 3.2.2: Zusammensetzung des Parameters CAN-Tx-Mode

Inhibit-Time..... Der Parameter Inhibit-Time gibt die Zeit an, die der CAN-Controller nach der letzten erfolgreichen Sendung von CAN-Bus Daten wartet, bevor er eine neue Sendung initiiert. Die Eingabe erfolgt in [ms].
Die Default-Einstellung beträgt $14 (= 20 ms).

MinChar, MaxChar... In diesen beiden Parametern wird die Anzahl der Bytes angegeben, ab der eine Sendung auf den CAN-Bus gestartet werden soll und wieviele Daten-Bytes maximal in einem CAN-Frame gesendet werden sollen.
Die Default-Einstellung beträgt $11, d.h. es wird jedes eintreffende Byte einzeln in einem Frame gesendet.

MinChar wird nur dann wie oben beschrieben ausgewertet, wenn Werte zwischen $1...$8 eingegeben werden.
Bei Werten zwischen $9 und $F ändert sich die Funktionalität dieses Parameters: In diesem Fall wertet die lokale Software den Eintrag der Steuerzeichen 'Carriage Return' ($0D) oder 'Line Feed' ($0A) als Sendebedingung aus. Wird eines oder beide Zeichen erkannt, so werden die bisher von der seriellen Schnittstelle empfangenen Daten auf den CAN-Bus gesendet. Sind 8 Bytes empfangen worden, bevor die Steuerzeichen eingetroffen sind, so erfolgt die Sendung automatisch.
Für die Sendung kann außerdem noch ausgewählt werden, ob die Steuerzeichen auf dem CAN-Bus zusammen mit den Daten übertragen werden sollen oder nicht. Die folgende Tabelle zeigt die verschiedenen Optionen:

MinChar [HEX]	Sendebe-dingung	Steuerzeichen-Übertragung auf CAN	Kommentar
1...8	-	-	Es werden immer mindestens 1 bis 8 Bytes gesendet. Der Parameter MaxChar wird ebenfalls ausgewertet.
9	< Cr >	mit < Cr >	Daten werden gesendet, wenn < Cr > empfangen wurde. < Cr > wird mit übertragen.
A	< Lf >	mit < Lf >	Wie oben, aber mit < Lf >.
B	< Cr >	ohne < Cr >	Wie oben, aber < Cr > wird nicht mit übertragen.
C	< Lf >	ohne < Lf >	Wie oben, aber ohne < Lf >.
D	< Cr >	ohne < Cr > und ohne < Lf >	Am Ende der Nachricht kann < Cr > und < Lf > stehen. Die Daten werden gesendet, wenn < Cr > empfangen wurde. Weder < Cr > noch < Lf > werden mit übertragen.
E	< Lf >	ohne < Cr > und ohne < Lf >	Wie oben, aber mit < Lf >.
F	-	-	reserviert

Tabelle 3.2.3: Auswahl der Sendebedingung über den Parameter MinChar

Beispiele:

1. Für MinChar wurde der Wert $B ausgewählt. Über die serielle Schnittstelle treffen die Daten 'abcd<Cr>' ein. Auf dem CAN-Bus würden nach dem Empfang von <Cr> die Daten 'abcd' gesendet werden.

2. Für MinChar wurde der Wert $E ausgewählt. Über die serielle Schnittstelle treffen die Daten 'abcd<Cr><Lf>' ein. Auf dem CAN-Bus würden nach dem Empfang von <Lf> die Daten 'abcd' gesendet werden.

3.3 Serial-Mode (Parameter 2, A, 12, 1A, 22)

Die 'Serial-Mode'-User-Parameter setzen die Baudrate, die Stop-Bits, die Anzahl der Bits/ Character und die CTS*-Verriegelung und legen die Parity-Auswertung der seriellen Kanäle fest.

User-Parameter-No. (Sub-Command-No.)	Parameter	Wertebereich	Default-Einstellung
$02	Serial-Mode Kanal 1	$0000..$88FF	$2273
$0A	Serial-Mode Kanal 2		
$12	Serial-Mode Kanal 3		
$1A	Serial-Mode Kanal 4		
$22	Serial-Mode Kanal 5		

Tabelle 3.3.1: User-Parameter Serial-Mode

Die zwei Byte des Parameters Serial-Mode setzen sich wie folgt zusammen:

	Serial-Mode		
	Byte 5 des INIT-Id $700		Byte 6 des INIT-Id $700
Parameter	Rx-Baudrate	Tx-Baudrate	Mode
Wertebereich	$0...$8	$0...$8	$00...$7F

Tabelle 3.3.2: Zusammensetzung des Parameters Serial-Mode

Rx-Baudrate,
Tx-Baudrate...

In 4 Bit wird jeweils festgelegt, mit welcher Baudrate auf den seriellen Schnittstellen gesendet (Tx), bzw. empfangen (Rx) werden soll. Die Standardeinstellung ist für Rx- und Tx-Baudrate 9600 KBit/s.
Die physikalisch erreichbare Baudrate wird durch die Hardware auf maximal 38,4 kBit/s, bei gleichzeitiger Nutzung aller vier Kanäle, beschränkt.

Parameter Rx-(Tx) Baudrate [HEX]	Baudrate [Bit/s]
0	38400
1	19200
2	**9600**
3	4800
4	2400
5	1200
6	600
7	300
8	150
9	7200
A	14400
B	28800
C	(57600)
D	(115200)
E	(2304000)
F	(76800)

Tabelle 3.3.3: Einstellung der Baudrate der seriellen Schnittstellen

Mode..... Die Bits des Parameters Mode sind mit folgenden Funktionen belegt:

Bit	7	6	5	4	3	2	1	0
Belegung	RTS-Mode	CTS*-enable	Stop-Bit	Parity-Mode		Parity-Type	Bits per Character	
Default	0	1	1	1	0	0	1	1

Tabelle 3.3.4: Belegung des Parameters Mode

Erläuterung der Bits des Parameters Mode

RTS-Mode..... Mit diesem Bit kann der RTS-Modem-Mode für RS485-Schnittstellen gewählt werden.

RTS-Mode	Auswertung
0	RTS on Rx (Default-Einstellung)
1	RTS-Modem (RS485)

Tabelle 3.3.5: Auswertung des RTS-Mode-Bits

CTS* enable... Über dieses Bit wird die CTS*-Funktion der seriellen Controller freigegeben.
Ist das Bit '0', so wird das CTS*-Signal nicht ausgewertet und der Controller sendet die vorhandenen Daten sofort (sofern sich das Modul nicht im Zustand 'Suspended' befindet).
Die Auswertung des Bits kann der folgenden Tabelle entnommen werden.

CTS* enable Bit	CTS* -Auswertung
0	CTS* -Eingang wird ignoriert (immer bei Terminal-Schnittstelle)
1	CTS* -Eingang wird ausgewertet: Hardware-Handshake (Default-Einstellung für Kanal 1...4)

Tabelle 3.3.6: Auswertung des 'CTS* enable'-Bits

Die eingesetzten RS232-Treiber aktivieren das CTS*-Signal selbstätig, wenn die CTS*-Leitung der seriellen Schnittstelle nicht angeschlossen ist.

Bei der Terminal-Schnittstelle (DSUB9) wird der CTS-Eingang nicht ausgewertet. Das CTS* enable-Bit ist für diese Schnittstelle daher ohne Bedeutung.

Stop Bit..... Hier wird die Anzahl der Stop-Bits der seriellen Schnittstelle festgelegt:

Stop Bit	Anzahl der Stop Bits
0	1 Stop Bit (immer bei Terminal-Schnittstelle)
1	2 Stop Bits (Default-Einstellung für Kanal 1...4)

Tabelle 3.3.7: Anzahl der Stop-Bits

Die Terminal-Schnittstelle (DSUB9) arbeitet immer mit nur einem Stop-Bit. Die Einstellung ‘1’ ist für diese Schnittstelle daher ohne Bedeutung.

Parity Mode... Mit diesen beiden Bits wird die Auswertung des Parity-Bits festgelegt:

Parity Mode		Auswertung
Bit 4	Bit 3	
0	0	Parity-Auswertung beim Empfangen von Daten und Senden des Parity-Bits
0	1	reserviert
1	0	keine Parity-Auswertung, keine Parity-Sendung (Default-Einstellung)
1	1	reserviert

Tabelle 3.3.8: Parity-Auswertung

Parity Type... Die Polarität des Parity-Bits wird mit dem Parameter-Bit 'Parity Type' festgelegt:

Parity Type	Polarität
0	'gerade' (even) (Default-Einstellung)
1	'ungerade' (odd)

Tabelle 3.3.9: Einstellung der Polarität

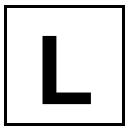
Bits per Character..... Mit diesen beiden Bits wird die Anzahl der Bits/Character ausgewählt:

Bits per Character		Anzahl der übertragenen Bits
Bit 1	Bit 0	
0	0	5 (nicht bei Terminal-Schnittstelle)
0	1	6 (nicht bei Terminal-Schnittstelle)
1	0	7
1	1	8 (Default-Einstellung)

Tabelle 3.3.10: Anzahl der Bits/Character

Für die Terminal-Schnittstelle (DSUB9) sind nur folgende Kombinationen von ‘Bits per Character’ und ‘Parity Mode’ zulässig:

7 Daten-Bits +	1 Parity-Bit	
8 Daten-Bits	no Parity	(Default-Einstellung)
8 Daten-Bits +	1 Parity-Bit	



4. Beispiele

In diesem Kapitel soll anhand einiger Beispiele der Betrieb und die Initialisierung eines Moduls, das mit dem ***esd-CAN-Protokoll*** betrieben wird, verdeutlicht werden.

4.1 Betrieb mit den Default-Parametern

4.1.1 Rahmenbedingungen, Zielsetzung

An Kanal 1 der CAN-CBM-SIO4 soll ein Gerät angeschlossen werden, das in seinen Parametern der Default-Einstellung der CAN-CBM-SIO4 entspricht:

- 9600 Baud
- 8 Bit/Character
- 2 Stop-Bits
- no Parity

Das Modul ist bisher nicht initialisiert worden.

Der gewünschte Tx-Identifier, auf dem die von der seriellen Schnittstelle empfangenen Daten auf dem CAN-Bus gesendet werden sollen, soll hier $15E betragen. Der Rx-Identifier soll $15F betragen.

4.1.2 Vorgehensweise

4.1.2.1 Identifier einstellen

Der Identifier wird z.B. über die Kodierschaltern festgelegt. Er entspricht dem 10fachen des eingestellten Kodierschalterwertes zuzüglich dem Identifier-Offset. (Anmerkung: Die Auswahl eines beliebigen Wertes ist daher über die Kodierschalter nicht möglich. Ist dies gewünscht, so muß die Einstellung über das esd-CAN-Protokoll erfolgen.)
Mit der oben angegebenen Vorgabe für den Identifier, müssen die Kodierschalter auf einen Wert eingestellt werden, der $15E geteilt durch $A entspricht.

hexadezimal:	$15E	/	$A	=	$23
(dezimal:	350	/	10	=	35)

An dem Kodierschalter ‘HIGH’ wird jetzt der Wert ‘2’ und an Kodierschalter ‘LOW’ der Wert ‘3’ eingestellt.
Der Offset ergibt sich aus dem gewünschten seriellen Kanal: TxId-Offset = 0, RxId-Offset=1

Die Lage und Funktion der Kodierschalter sind im Hardware-Handbuch des Moduls ausführlich beschrieben.

4.1.2.2 Senden der Daten auf die serielle Schnittstelle

Über den Rx-Identifier RxId1 werden die Daten an das Modul übertragen. Die Anzahl der übertragenen Daten-Bytes kann zwischen 0 und 8 liegen. In diesem Beispiel sollen die folgenden 5 Byte übertragen werden:

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
$11	$22	$33	$44	$55

Auf dem CAN-Bus werden die Bytes auf dem Rx-Identifier RxId1 wie folgt gesendet:

RxId1	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
$15F	$11	$22	$33	$44	$55	-	-	-

Tabelle 4.1.2: Übertragung der zu sendenden Daten über Rx-Identifier RxId1

Die Bytes 6...8 werden nicht benötigt.

4.1.2.3 Empfangen der Daten von der seriellen Schnittstelle

Über den Tx-Identifier TxId1 werden die von der seriellen Schnittstelle empfangenen Daten vom Modul auf den CAN-Bus gesendet. Da das Modul mit den Default-Parametern betrieben wird, wird jedes eintreffende Byte sofort gesendet. Die Wartezeit zwischen dem Start der einzelnen Sendungen durch den CAN-Controller beträgt 20 ms.

In diesem Beispiel soll das an Kanal 1 angeschlossene Gerät einen Block von 8 Bytes mit folgendem Inhalt senden:

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
$FF	$EE	$DD	$CC	$BB	$AA	$99	$88

Das Modul sendet jetzt die einzelnen Bytes dieses Blockes auf dem Tx-Identifier mit Pausen von ca. 20 ms. Ist der CAN-Bus mit Nachrichten höherer Priorität belegt, können sich die Pausen zwischen den Sendungen verlängern.

TxId1	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
$15E	$FF	-	-	-	-	-	-	-

:
t þ 20ms
:

TxId1	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
$15E	$EE	-	-	-	-	-	-	-

:
t þ 20ms
:

TxId1	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
$15E	$DD	-	-	-	-	-	-	-

:
...
:
t þ 20ms
:

TxId1	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
$15E	$88	-	-	-	-	-	-	-

Tabelle 4.1.3: Senden der empfangenen seriellen Daten auf dem CAN-Bus

Die Bytes 2...8 der einzelnen Tx-Transfers werden nicht mit übertragen, da in der Default-Einstellung für 'MaxChar' der Wert '1' eingestellt ist.

4.2 Ändern der Baudrate

Die Baudraten von Kanal 1 sollen von 9600 Baud (Default-Einstellung) für Empfangs- und Sendedaten auf 19200 Baud erhöht werden.

Die Module-No. der CAN-CBM-SIO4 entspricht in der Default-Einstellung der Kodierschalterstellung und beträgt somit $23.

Die Änderung der Baudrate wird in dem Zellen 'Rx-Baudrate' und 'Tx-Baudrate' des User-Parameters 'Serial-Mode' vorgenommen. Diese beiden Zellen belegen das erste Byte des User-Parameters (bei der Übertragung = Byte 5). Für Kanal 1 wird dieser Parameter über das Sub-Command $02 ausgewählt.

Die Zuordnung der Baudrate 19200 ergibt für jedes Nibble von Byte 5 des User-Parameters den Wert '$1'.

Byte 6 des Parameters (Serial-Mode) soll nicht verändert werden, muß aber, wie bei allen User-Parametern, ebenfalls mit übertragen werden! Daher wird hier der Default-Wert $73 eingetragen.

CAN-Id	Byte 1 (Command)	Byte 2 (Sub-Command)	Byte 3 (immer $00)	Byte 4 (Module-No.)	Byte 5 (Baudrate)	Byte 6 (Serial Mode)
$700 =INIT-Id	$86	$02	$00	$23	$11	$73

Tabelle 4.2.1: Ändern der Baudraten des seriellen Kanals 1

Byte 7 und 8 werden für dieses Kommando nicht benötigt.

Die geänderte Baudrate ist sofort nach dem Empfang des User-Parameters aktiv.

esd CAN Protocol
for
CAN-CBM-SIO1/4

Manual File:	I:\texte\Doku\MANUALS\CAN\Cbm\SIO-331\Englisch\ESD_SIO1.EN6
Date of Print:	11.04.2001

Described Software Revision:	
CAN kernel :	from revision '1.f' (HEX)
esd protocol :	from revision '0' (HEX)
Module specific implementation:	refer to manual of the module specific software

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. 205
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
7667 W. Sample Road
Suite 127
Coral Springs, FL 33065
USA

Phone: +1-800-504-9856
Fax: +1-800-288-8235
E-mail: sales@esd-electronics.com

Content

Page

1. Introduction . . . . . 1 - 1
1.1 Notes to this Manual . . . . . 1 - 1
1.2 Specification of the *esd* Protocol . . . . . 1 - 1
1.3 General Notes on Data Transmission . . . . . 1 - 1
1.4 General Hardware Functions . . . . . 1 - 2
1.5 Parameters after a RESET . . . . . 1 - 2
1.6 Advanced Configuration . . . . . 1 - 6
1.6.1 Setting Instruction . . . . . 1 - 6
1.6.2 Parameter Numbers and Values . . . . . 1 - 7
1.7 Summary of LED States . . . . . 1 - 8
1.8 Calling the Commands and Setting the Parameters . . . . . 1 - 9

2. Overview of the Implemented Commands and Parameters . . . . . 2 - 1
2.1 Overview of the Commands . . . . . 2 - 1
2.2 Overview of the Returned Parameter Values . . . . . 2 - 4

3. Description of the Commands and Parameters . . . . . 3 - 1
3.1 Configuration Reply . . . . . 3 - 1
3.2 System Parameters . . . . . 3 - 7
3.3 Process TxIds . . . . . 3 - 19
3.4 Process RxIds . . . . . 3 - 22
3.5 Cyclic Tx Transfers (Tx Activate Time) . . . . . 3 - 25
3.6 Process User Parameters . . . . . 3 - 28
3.7 Service Request . . . . . 3 - 30
3.8 Supervisor Commands . . . . . 3 - 32

4. Examples for Parameterization . . . . . 4 - 1
4.1 Setting the Tx Identifier TxId1 . . . . . 4 - 1
4.2 Restoring the Default Parameters . . . . . 4 - 3
4.2.1 ...if the actual module no. is unknown: . . . . . 4 - 3
4.2.2 ...if the module no. is known: . . . . . 4 - 4

1. Introduction

1.1 Notes to this Manual

This manual describes the '*esd*-CAN protocol' for *esd*-CAN modules, here specially for the CAN-CBM-SIO modules. By this protocol it is possible to set the CAN parameters of the modules, as for example, the Rx and Tx identifiers or the baudrate. Apart from the CAN parameters it is also possible to set and play back module-specific parameters (user parameters) with the help of the protocol.

In the chapter 'Overview of the implemented commands and parameters' all commands supported at the moment are shown in a tabular summary.

You can find detailed information about the module-specific User-Parameters in the description of the module-specific software.

1.2 Specification of the *esd* Protocol

On the basis of its problem-oriented structure the *esd* protocol cannot be categorized clearly into a layer of the ISO layer model: It offers services which range from fundamental functions, as e.g., the inquiry of the status of the CAN hardware controller up to application-specific adjustments, as, e.g., the setting of so-called 'user parameters'.

The *esd* protocol offers functions which are comparable, e.g., with the LMT (layer management) in the CAL (CAN application layer). Because the identifier allocation occurs by the *esd* protocol, too, its functionalities are partly similar to those of the DBT (identifier distributor) in the CAL, as well.

But the *esd* protocol is operated totally independently and has got no interface to the CAL!

1.3 General Notes on Data Transmission

In the following descriptions the transmission direction of data is looked at, if not other wisely stated, from the module. The module receives data on the 'Rx identifier' and transmits data on the 'Tx identifier'.

The data bytes are counted from 1 to 8 and are always transmitted in the order byte 1...byte 8. The number of transmitted bytes can vary from 0 to 8. The data transmission has to start with byte 1 and progress in continuous order (e.g., byte 1, byte 2, byte 3 - not possible, e.g., byte 1, byte 7, byte 8).

Generally only those bytes are overwritten which are received by the module. The data of not recorded bytes remain unchanged.

1.4 General Hardware Functions

To use the esd CAN protocol at the CAN modules at each module the following hardware circuits are necessary:

CAN controller SJA1000 (or compatible)

The controller has a internal RAM, that is used as a working memory. In this RAM the dynamical parameters are stored. The RAM is deleted with each RESET.

I²C EEPROM

The I²C EEPROM is used for storing the configuration parameters. The data will remain stored in power off condition or after a RESET.

Coding switch

Via the coding switch, e.g. the default value of the module no. is set. Valid module no's are from $00 up to $FE. The value $FF is reserved for the 'Advanced Configuration'.

Further descriptions can be read in the hardware manual of the module.

1.5 Parameters after a RESET

The module offers various possibilities to trigger a RESET:

A power-on RESET and a RESET by the general command 'RESET module' reset the local components.
Furthermore the module is able to trigger a RESET independently when the hardware watchdog has expired. This RESET also resets the local components without changing the stored parameters.

The listed RESETs only change the parameters of the module filed in the I²C EEPROM, if the conditions apply which are listed in the table below.

The module also supports the command 'default RESET'. By this command a local RESET is triggered and the parameters of the module are <u>always</u> overwritten with the default parameters.

The used parameters with which the module operates after a power-on RESET or a RESET by the general command 'RESET module', depend mainly on three factors:

The switch position of the coding switches, the availability of the I²C EEPROM data and the module number (parameter 'module no.') stored in the I²C EEPROM.

The following table should give an overview of the resulting parameters. It does not contain the 'default RESET', because this does always lead to the activation of the default parameters.

	Actual position of the coding switches after RESET	I²C EEPROM status	I²C EEPROM module no.	CAN identifier (CAN Id.), parameter, module no.
1a	$00	x	x	CAN Id. = f{Coding switches} Parameter = default I²C EEPROM mod no. = in this case the previous module no. is deleted at power-on active mod no. = $00
1b	≠ $00	ERROR	x	CAN Id. = f{Coding switches} Parameter = default I²C EEPROM mod no. =$00 active mod no. = Coding switch no.
2	≠ $00	OK	$00	CAN Id. = f{Coding switches} Parameter = default I²C EEPROM mod no. = $00 active mod no. = Coding switch no.
3	≠ $00	OK	≠ $00	CAN Id. = f{I²C EEPROM} Parameter = f{I²C EEPROM} active mod no. = I²C EEPROM mod no.

(x) This value or status is of no importance in this case.

Table 1.5.1: Parameter after a RESET

Explanations to Table 1.5.1:

Some of the terms from the table above will be described in detail in following sections of this manual. For a general understanding of the table, a short explanation of the terms:

Module no. ...	Serial number (1...254) which can be allocated to the module by the user independently from the module type.
active mod no. ...	The module no. with which the module is selected by the initialisation identifier (INIT Id) during the parameter interchange.
I²C EEPROM mod no. ...	The module no. which was stored in the local I²C EEPROM of the module. If no change in this number had been made after the storing, the actual mod no. is identical with the I²C EEPROM module no..

Below the combinations of factors, shown in table 1.4.1, which are decisive for the selection of the default parameters will be explained:

Combination 1a

If the positions of the coding switches are $00 when the module starts after a RESET, the I²C EEPROM data, previously stored, will be overwritten at the moment in which the adjustment is changed from $00 to any other value. The firmware needs about 10 seconds for this. Afterwards a local RESET is triggered by the firmware. During the RESET both LEDs are flashing fast.

The CAN identifier corresponds to the value adjusted at the coding switches.

The module operates with the default parameters.

Combination 1b

The second listed combination occurs if no I²C EEPROM is available or the I²C EEPROM data are faulty. If this is the case, the local software would use the default parameters.

The CAN identifier corresponds to the value adjusted at the coding switches. The actual module no. corresponds to the value which is adjusted on the coding switches.

Combination 2

In the third combination the module no. $00 is stored. The modules works with the default parameters.

Combination 3

In this combination of the factors listed above, the module operates after a RESET with the previously changed and in the I²C EEPROM stored parameters.

Requirements for this are I²C EEPROM status=OK, a coding switch position unequal $00 and a module no., stored in the I²C EEPROM, which has got a value unequal $00.

The CAN identifier with which the module operates and all used parameters are taken from the I²C EEPROM.

The actual module no. with which the module is selected in the initialisation phase corresponds to the module no. stored in the I²C EEPROM.

1.6 Advanced Configuration

With the *Advanced Configuration* you can set up to 15 additional parameters with the coding switches. To set the value of the parameters you have to execute the below described step-by-step instruction.

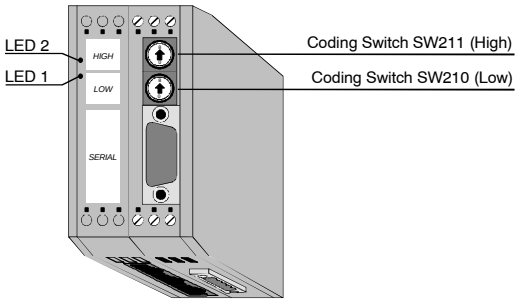


Fig. 1.6.1: Names of LEDs and coding switches

1.6.1 Setting Instruction

Step	State of LEDs		Action/Setting of Coding Switches
	LED 1	LED 2	
1	OFF	OFF	- set both coding switches to 'F' - switch on the power supply of the module
2	flashes fast	flashes slow	- now you have 10 seconds to set the switch HIGH to the parameter number and the switch LOW to the parameter value (parameters see table below) Only, if previous step was step 4: - if do not want to change more parameters, set both coding switches to 'F' to write the changed parameter(s) to the EEPROM
3	lights continuously	flashes slow	- the local firmware verifies your settings of step 2 in 2 seconds
4	flashes fast	lights continuously	- if this light state appears, the changed parameter value is accepted - after 2 seconds the firmware continues with step 2, except you have set both coding switches to 'F', than the firmware continuous with step 5
5	flashes fast	flashes fast	- the LEDs are flashing for approx. 10 seconds, then the module is automatically reset; within this time you have to set the module no. at the coding switches - after the reset the module starts with the changed parameters and the set module no.

1.6.2 Parameter Numbers and Values

Coding Switch		Meaning
HIGH (parameter number)	LOW	
0	baud (0...F)	setting the CAN baudrate index (values see table at page 11)
1-E	reserved	-
F	F	write data to EEPROM

1.7 Summary of LED States

State of LEDs		Meaning
LED 1	LED 2	
OFF	OFF	- power supply off
lights continuously	OFF	- module is working with stored parameters
OFF	lights continuously	- module is working with default parameters
lights continuously	flash slow	- local firmware verifies parameter settings of coding switches (see page 6)
flash fast	flash slow	- time to set parameter value by coding switches (see page 6)
flash fast	lights continuously	- the changed parameter value is accepted (see page 6)
flash fast	flash fast	- microcontroller writes parameter value to EEPROM (see page 6)

1.8 Calling the Commands and Setting the Parameters

If the module is in original condition (default condition at delivery), it operates only by the CAN identifiers on the CAN (see hardware manual) adjusted by hardware.

To report the initialisation parameters to the module nevertheless, a special CAN identifier (INIT Id) has been reserved which is the same for all *esd*-CAN modules.

In spite of the identifier adjusted by the coding switches, the module receives and processes every CAN frame transmitted on the INIT Id.

The INIT Id determined by *esd* has got the value:

$700

(valid since software version V0.8, subject to alterations)

The identifier $700 is reserved for the initialisation, i.e., if this identifier should wrongly be allocated to other functions, the transmitted data will be interpreted as initialisation parameter, nevertheless!

Generally not all modules should be initialized with the same parameters. To distinguish the modules the fourth byte of the six INIT Id bytes has to have the 'actual Module no.' of the wanted module at the initialisation.

The module no. is a characteristic number in the area of $00...$FE which can be freely defined by the user. It is possible, e.g., to number all existing modules (max. 254 modules), independent from the type, continuously.

In a CAN net the same module no. should only be used once. The module no. $00 should not be used, because it is used during the initialisation sequence with global commands, which are valid for all modules.

The actual module no. is identical with the number adjusted at the coding switches, when the module is operated with the default parameters.

But it is also possible to program the module no. freely during the initialisation.

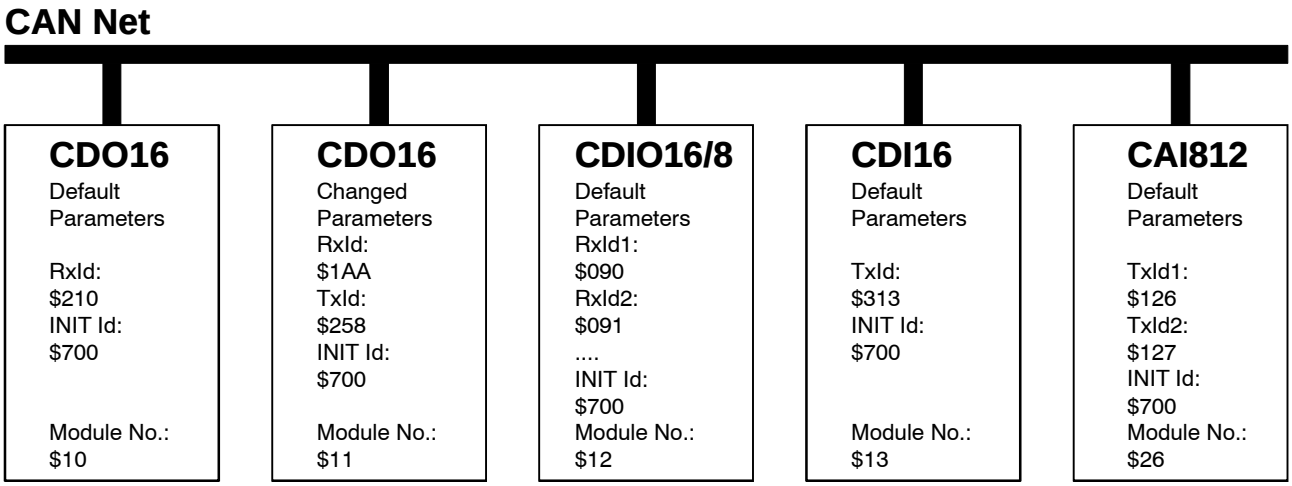


Fig. 1.6.1: Examples for the module no. and the CAN identifier

6 bytes are necessary for the initialisation of a module. The transmitted structure has to have the correct length and the functions corresponding to the module type of board. If this is not the case, the module would not react to the initialisation.

The table below shows the construction of the INIT Id. In the command byte and the sub command bytes the function of the respective initialisation level is determined. The module no. selects the wanted module. In the cells parameter 1 and 2 the wanted parameters are transmitted.

Byte no.	Function	Value range
1	Command	$00...$FF
2	Sub command 1	$00...$FF
3	Sub command 2 1*)	$00...$FF 1*)
4	Module no.	$00, $01...$FF
5	Parameter 1	$00...$FF
6	Parameter 2	$00...$FF
7	Not used	-
8		

(*) Byte 3 (sub command 2) is not needed for the majority of commands and parameters. In this case it should always be allocated with $00.

Table 1.6.1: Data bytes of the INIT Id ($700)

The parameter interchange can lead to the setting of parameters, to the reply of already adjusted parameters or to the execution of a command. At the command interchange and the setting of parameters the highest bit of the command byte is always '1', at the request of parameters it is always '0'.

The requested reply of the module is only sent once by the module. The identifier (CTxId) which had been allocated to the module for this transmission is not stored on the module. If another transmission is desired an INIT Id with the corresponding parameters has to be transmitted to the module again.

If the module processes faultlessly and the CAN is free, it transmits the reply within 200µs (max. 10 ms).

Normally, at the reply of the actual parameter condition of the modules, in the first byte the contents of the received command byte and in the second byte the contents of the sub command byte is given.

This is not the case if the second byte is used for the display of other parameters or an unknown command or sub command byte had been sent to the module. In the second case a message containing only one byte with the contents $FF is given back to the CAN.

The given value ranges of the parameters have to be kept, because otherwise the faultless execution of commands is not guaranteed! No error message occurs after wrong entry of the parameter values.

Below the parameters and commands will be specified. The examples added subsequently to the specification should clarify the function course further.

The description of the bytes of the INIT Id restricts to those which are relevant for the corresponding mode.

Byte 3 (sub command 2) should, if it is not needed, always be recorded with $00. In byte 4 the actual module no. has to be entered, as already mentioned above.

2. Overview of the Implemented Commands and Parameters

The two following tables give a complete summary of all bytes implemented until now and the parameters given back by the module. The individual designations of the commands and parameters will not be explained in detail in the tables in favour of the clarity. The descriptions of the used expressions can be taken from the following chapters in which the individual commands will be described.

The value ranges of the parameters cover all possible entries which are evaluated correctly by the software. Here it has to be noted that partly special functions (e.g. 'function switched off') have been allocated to individual values of the parameters (e.g. $00). The allocation of values which are outside of the given limits is not permissible, because otherwise the perfect function of the addressed module will not be guaranteed anymore.

2.1 Overview of the Commands

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Command	Sub-Command	don't care	Module No.	Parameter 1	Parameter 2	not used	not used
Request Configuration $00	$00 - Module Type $01 - Active Switch $02 - ASCII Id $03 - Software Rev. $04 - reserved $05 - Serial No.	Please write always $00.	selected Module No. $01...$FF	CTxId $0000...$07FF		These Bytes are not used.	

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Command	Sub-Command 1	Sub-Command 2	Module No.	Parameter 1	Parameter 2	not used	not used
System Parameter: $81	$00 - Save Parameter			selected Module No. (= Byte 4)	-		
$81 - set	$01 - Module No			new Module No. $00...$FF	-	▲	
$01 - request	$01 - Saved Module No			CTxId $0000...$07FF			
$81 - set	$02 - CAN-Status-Byte	▲ Please write always $00.	▲ selected Module No. $01...$FF	cstat $00	-	These Bytes are not used.	
$01 - request				CTxId $0000...$07FF			
$81 - set	$03 - Bitrate	▼	▼	bust 0 (=BTR0) (see Controller-Manual)	bust 1 (=BTR1) (see Controller-Manual)	▼	
$01 - request				CTxId $0000...$07FF			
$81 - set	$04 - Watchdog Tx Identifier			WTxId $0000...$07FF			
$01 - request				CTxId $0000...$07FF			
$81 - set	$05 - Watchdog Time	WDLife TimeFactor		WDtime $0000...$FFFF [ms]			
$01 - request		-		CTxId $0000...$07FF			

Overview

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Command	Sub-Command	don't care	Module No.	Parameter 1	Parameter 2	not used	not used
TxIds $82 - set	$00 - TxId1 $01 - TxId2 $02 - TxId3 : n - TxId(n+1)	Please write always $00.	selected Module No. $01...$FF	TxId $0000...$07FF		These Bytes are not used.	
$02 - request				CTxId $0000...$07FF			

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Command	Sub-Command	don't care	Module No.	Parameter 1	Parameter 2	not used	not used
RxIds $83 - set	$00 - RxId1 $01 - RxId2 $02 - RxId3 : n - RxId(n+1)	Please write always $00.	selected Module No. $01...$FF	RxId $0000...$07FF		These Bytes are not used.	
$03 - request				CTxId $0000...$07FF			

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Command	Sub-Command	don't care	Module No.	Parameter 1	Parameter 2	not used	not used
Tx-Activate-Time $84 - set	$00 - act.-T. TxId1 $01 - act.-T. TxId2 $02 - act.-T. TxId3 : n - act.-T. TxId(n+1)	Please write always $00.	selected Module No. $01...$FF	tx_act $0000...$FFFF [ms]		These Bytes are not used.	
$04 - request				CTxId $0000...$07FF			

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Command	Sub-Command	don't care	Module No.	Parameter 1	Parameter 2	not used	not used
User Parameter $86 - set	User Parameter No. $00...$7F	Please write always $00.	selected Module No. $01...$FF	Para $0000...$FFFF		These Bytes are not used.	
$06 - request				CTxId $0000...$07FF			

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Command	Sub-Command 1	Sub-Command 2	Module No.	Parameter 1	Parameter 2	not used	not used
Service Request $7F	Module No_LOW $00...$FF	Module No_HIGH $00...$FF	Please write always $00.	CTxId $0000...$07FF		These Bytes are not used.	

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Command	Sub-Command	don't care	Module No.	Parameter 1	Parameter 2	not used	not used
Supervisor Commands $FF	$00 - RESET Module	Please write always $00.	$00 - all Modules $01...$FF - selected Module	$AAAA - RESET		These Bytes are not used.	
	$01 - reserved			-			
	$02 - Supervisor Watchdog			-			
	$03 - Default RESET			$AAAA - Default-Reset			
	$04 - Suspend/ Continue Module			$5A5A - suspend $A5A5 - continue			
	$05 - RESET CAN-Status			-			

2.2 Overview of the Returned Parameter Values

Command: Request...	Sub-Command	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Configuration $00	$00-Module Type	$00	$00	type	iomode	-	-	-	-
	$01-Active Switch	$00	$01	switch	-	-	-	-	-
	$02- ASCII Id	$00	a (ASCII)	b (ASCII)	c (ASCII)	d (ASCII)	e (ASCII)	Module No.-H (ASCII)	Module No.-L (ASCII)
	$03- Software Revision	$00	'V' in ASCII	level H	'.' in ASCII	level L	revision	esd/cms	protocol-revision
	$04- reserved	$FF	-	-		-			
	$05- Serial No.	$00	$05	u (ASCII)	v (ASCII)	w (ASCII)	x (ASCII)	y (ASCII)	z (ASCII)

Command: Request...	Sub-Command	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
System Parameter: $01	$00-Saved Module No.	$01	$00	saved Module No.	-	-	-	-	-
	$01-Active Module No.	$01	$01	active Module No.	-	-	-	-	-
	$02 - CAN-Status Byte	$01	$02	cstat	-	-	-	-	-
	$03 - Saved Bitrate	$01	$03	bust 0	bust 1	-	-	-	-
	$04 - WD-Tx-Id	$01	$04	WTxId		-	-	-	-
	$05 - WD-Time	$01	$05	WDtime [ms]		WDLifeTime Factor	-	-	-

Command: Request...	Sub-Command	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
TxIds $02	$00-TxId1	$02	$00	TxId1		-	-	-	-
	$01-TxId2	$02	$01	TxId2		-	-	-	-
	$02-TxId3	$02	$02	TxId3		-	-	-	-
	:	$02	:	:		-	-	-	-
	n -TxId(n+1)	$02	n	TxId(n+1)		-	-	-	-

Command: Request...	Sub-Command	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
RxIds $03	$00-RxId1	$03	$00	RxId1S		RxId1E		-	-
	$01-RxId2	$03	$01	RxId2S		RxId2E		-	-
	$02-RxId3	$03	$02	RxId3S		RxId3E		-	-
	:	$03	:	:		:		-	-
	n -RxId(n+1)	$03	n	RxId(n+1)S		RxId(n+1)E		-	-

Command: Request...	Sub-Command	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Tx Activate Time $04	$00- Time TxId1	$04	$00	tx-act1 [ms]		-	-	-	-
	$01- Time TxId2	$04	$01	tx-act2 [ms]		-	-	-	-
	$02- Time TxId3	$04	$02	tx-act3 [ms]		-	-	-	-
	:	$04	:	:		-	-	-	-
	n-Time TxId(n+1)	$04	n	tx-act(n+1)		-	-	-	-

Command: Request...	Sub-Command	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
User Parameter $06	$00-Para 0	$06	$00	Para0		-	-	-	-
	$01-Para 1	$06	$01	Para1		-	-	-	-
	...	$06	...	...		-	-	-	-
	$7F-Para127	$06	$7F	Para7F		-	-	-	-

(-) This Byte is not transmitted.

3. Description of the Commands and Parameters

3.1 Configuration Reply

The transmission is called by transmitting a 'request configuration' command. The identifier on which the module should transmit the information on the CAN is reported to it by byte 5 and 6.

Contrary to the other commands only parameters are called with this command.

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Function	Command	Sub command	Not used	Module no.	Parameter 1	Parameter 2
Value	$00	$00...$05	Always write $00	Selected module no. $01...$FF	CTxId $0000...$07FF	

Table 3.1.1: Bytes of the command 'request configuration'

Explanation of the Bytes Transmitted to the Module:

Command... The command 'request configuration' requests the transmission of the actual parameters of the module.

Sub command... The sub command determines the configuration bytes which should reply:

Sub command	Reply of the parameters
$00	Module type
$01	Active switch
$02	ASCII Id
$03	Software rev.
$04	reserved
$05	Serial number

Table 3.1.2: Selection of the configuration reply by sub command

Parameter 1 and 2... With these parameters it is reported to the module to which CAN Id. it should transmit the requested reply.
The module does only transmit once on this identifier. The identifier is not stored on the module.
If another transmission is desired another INIT Id with the corresponding parameters has to be transmitted to the module.

Transmission of the adjusted configuration by the module:

Decisive for the selection of the message to be transmitted is the value of the sub command received by the module. The following table shows the information which is transmitted to the CAN by the module.

Sub command	Reply of the parameter	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
$00	Module type	$00	$00	type	iomode	-	-	-	-
$01	Active switch	$00	$01	switch	-	-	-	-	-
$02	ASCII Id	$00	a	b	c	d	e	Mod No ASCIIH	Mod No ASCIIL
$03	Software rev.	$00	'V' ASCII	level H	'.' ASCII	level L	rev.	esd/cms	prot-rev
$04	reserved	$00	not defined						
$05	Serial number	$00	$05	u	v	w	x	y	z

(-) Byte is not transmitted.

Table 3.1.3: Transmitting the adjusted configuration

Explanation of the bytes transmitted by the module:

Sub command $00 --▸ module type

Byte 1... The first byte replies the contents of the received command byte (here always $00).

Byte 2... The second byte replies the contents of the received sub command (here always $00).

Byte 3:
type... In this cell the type of the *esd*-CAN module is coded. The following table shows examples of the existing type designations:

type	Modules (examples)
$00	reserved
$01	CDO16
$02	CDI16
$03	CAI810
$04	CAO812
$05	CDIO16/16
$06	CAI812
$07	reserved
$08	CREL8
$09	CSC595
$0A	CPIO16/8
$0B	CCOM4
$0C	CCOM1
$0D	PTIDAC
$0E	SPS16
$0F	CMIO
$10	reserved
$11	CTERM
$12	CBIP
$13	CANSAT
$14	AIS16
$15	CI488
$16	CAN-PT100/DMS4
$17	CDMS4I
$18	CAN-PCC
$19	CCOM1
$1A	XMIO4
$1B	reserved
:	:
$1F	reserved
$20	LasCon I/O
:	:
$30	SIO
$31	SIO4
:	:
$FE	reserved
$FF	reserved

Table 3.1.4 Examples for module type designations

Byte 4:
iomode...

The byte 'iomode' contains, broken down into 6 bits, information about the operating mode of the addressed module. A '1' of the respective bit signalizes the operating mode possible for this module:

Bit	Function	Example: CAN-CMIO
0	Output	1
1	Input	1
2	Digital	1
3	Analog	1
4	Controller	0
5	Serial	0
6	Reserved 1*)	0
7	Reserved 1*)	0

1*) These bits are read as '0'.

Table 3.1.5: Bits in parameter 'iomode'

Example: At a CAN-CMIO module the value $0F would be replied for 'iomode'. The value for a CAN-CBM-SIO is not defined at the moment.

Sub command $01 --▸ active switch

Byte1,
Byte2...

See sub command $00.

Byte 3:
switch...

The byte 'switch' replies the number adjusted on the coding switches.

If the module no. filed in the EEPROM has got the value $00 or if the EEPROM data are not OK, the actual module no. by which the module is addressed during the initialisation corresponds to this coding switch number.

Sub command $02 --▸ ASCII Id

Byte 1... See sub command $00.

Byte 2 - Byte 6
a, b, c, d, e... These bytes describe the module type in ASCII code.

Byte 7, Byte 8:
mod no. ASCII... These two bytes describe the module name and the active module number in ASCII code. The following table gives an example for the display of these bytes in ASCII code. It is a CBM-SIO4 module with the module no. $99.

Byte	2	3	4	5	6	7	8
Parameter	a	b	c	d	e	Module no. H	Module no. L
ASCII Id SIO4	S	I	O	4	1	9	9

Table 3.1.6: Example for an ASCII Id (the exact value for the SIO4 is not defined at the moment)

Sub command $03 --▸ software revision

Byte 1... See sub command $00.

Byte 2 - Byte 5
'V', level,
'.', rev.... In ASCII code these bytes describe the revision number of the firmware of the CAN core used on the module.
Byte 2 and byte 4 are permanently allocated with the ASCII symbols 'V'($56) or '.'($2E).
In byte 3 and byte 5 the actual revision number is described.
Byte 6 contains a letter which stands for the revision of the module-specific firmware.
In byte 7 is returned, which protocol is implemented (esd CAN protocol or CMS protocol). An 'E' means, that the esd CAN protocol is implemented and a 'C' means, that the CMS protocol is implemented.
Byte 8 returns the actual revision number of the used protocol (e.g. of the esd CAN protocol).

Byte	2	3	4	5	6	7	8
Parameter	'V'	level H	'.'	level L	rev.	esd/cms	prot-rev
ASCII display	V	1	.	0	a	E	0

Table 3.1.7: Example for the ASCII software rev no. 'V1.0aE0'

Sub command $04 --▸ reserved

Sub-Command $05 --▸ serial-number

Byte 1, Byte 2 ... refer to sub command $00.

Byte 3...8 ... Hardware serial number of the CAN module.

Byte	3	4	5	6	7	8
Parameter	u	v	w	x	y	z
ASCII display	A	A	0	0	1	2

Table 3.1.7: Example for the ASCII serial no. 'AA0012'

3.2 System Parameters

With the command $81 the parameters described below are set. By command $01 and the according sub command the module is lead to reply the actual parameters.

If the command 'store parameter' (sub command $00) is transmitted to the module, all previously interchanged parameters are stored in the local I²C EEPROM.
If the module is reset (RESET) or the supply voltage is switched off, the entered parameters are lost if this command has not been entered before.

After a RESET command or a power-on RESET the module operates with the stored parameters (identifiers etc.). If the programming has been unsuccessful (e.g. no I²C EEPROM or defect) the module uses the standard parameters (e.g. identifier =▸ coding switches).

The module no. with which the module is selected at the parameter interchange corresponds to the adjustment of the coding switches (provided no modification had been made so far). By the sub command $01 it is possible to allocate another module no. to the module. The new number is active immediately after the setting and the number adjusted by the coding switches is ignored.

The CAN-status byte offers various information about the condition of the module: It is shown if the module had previously not been connected to the CAN, if the default parameters had been activated after a RESET, if the last RESET had been caused by a power-on cycle, etc.

By the sub command $03 it is possible to change the CAN bitrate of the module which was adjusted by the configuration jumper of the module. The new bitrate is only activated after the parameters have been stored by sub command $00 and a RESET has been triggered.

Between the modules and a supervisor (master) a mutual function control by a watchdog protocol similar to the CMS specification can occur. By the sub commands $04 and $05 it is possible to interchange a watchdog identifier and a watchdog time.

<table>
<tr><th></th><th>Byte 1</th><th>Byte 2</th><th>Byte 3</th><th>Byte 4</th><th>Byte 5</th><th>Byte 6</th></tr>
<tr><th>Function</th><th>Command</th><th>Sub command</th><th>Not used</th><th>Module no.</th><th>Parameter 1</th><th>Parameter 2</th></tr>
<tr><td rowspan="8">Value</td><td>Request: $01</td><td>$00...$03</td><td rowspan="6">Always allocate $00</td><td rowspan="8">Selected module no. $01..$FF

= active module no.</td><td colspan="2">CTxId
$0000...$07FF</td></tr>
<tr><td rowspan="7">Set: $81</td><td>$00 store parameter</td><td>active mod no.</td><td>-</td></tr>
<tr><td>$01 module no.</td><td>new mod no.</td><td>-</td></tr>
<tr><td>$02 status</td><td>cstat</td><td>-</td></tr>
<tr><td>$03 bitrate</td><td>bust 0</td><td>bust 1</td></tr>
<tr><td>$04 watchdog Id</td><td colspan="2">WTxId
$0000...$07FF</td></tr>
<tr><td>$05 watchdog time</td><td>WDLife TimeFactor
$00...$FF</td><td colspan="2">WDtime
$0000...$FFFF</td></tr>
</table>

Table 3.2.1: Bytes of the commands 'store parameter, module no., bitrate and watchdog'

<u>Setting the Commands and Parameters (Command $81):</u>

Command... The command $81 leads to the setting of parameters or to the activation of the commands.
The command $01 leads to the reply of the actual parameters.

Sub command... The sub command selects the desired parameter interchange or the command to be executed.

Parameter1, Parameter2 Following sub commands are implemented:

Sub command	Function
$00	Storing the actual parameter
$01	Setting a new module no.
$02	Setting the CAN-status byte
$03	Setting a new bitrate
$04	Setting the watchdog Tx identifier
$05	Setting the watchdog time

Table 3.2.2: Function of the sub commands

The kind of the parameters interchanged to the module depends on the selected sub command:

Sub command $00 --▸ store parameter

active
mod. no... To store all interchanged parameters (also those of other commands) the actual module no. has to be entered into byte 5 when calling this sub command.
The actual module no. is either the number adjusted by the coding switches (default parameter active) or the number changed by sub command 'set module no.'.

Sub command $01 --▸ set module no.

new. mod. no.. Here the desired new module no. is entered. The module is addressed immediately after this command by the new module no. The module no. adjusted by the coding switches is ignored.
If the new module no. should remain active after a RESET, the parameters have to be stored by the sub command 'store parameters' before a power down or a RESET.

Sub command $02 --▸ set CAN-status byte

cstat... A 'set' access onto the CAN-status byte sets the bits 2 and 7 of the byte to '0'. All other bits of the status byte remain unchanged, because they serve as read only information (see also 'requesting the actual parameters').

Bit 2 shows that a CAN error has been detected by the module. The bit serves the documentation of errors which do not cause a 'standstill' for the CAN, but remain only for a short period. Error arising for a short time can easily be overlooked, because they make themselves visible only by a temporal limited flashing of the status LED.
The bit is set to '1' when an error has been detected. A 'set' access with any data (recommended: byte 5 = $00) onto the status byte or each RESET reset the bit back to '0'.

The CAN-error bit can also be reset by the supervisor command 'reset CAN error' (sub command $05). The other bits remain unchanged by this command.

Notes on the internal management of the CAN-error bit:

The CAN-error bit is set by the local software if the status bit of the CAN controller 'error status' is activated. (see constat).
After the first recognition of a CAN error the controller at first tries to transmit or receive data repeatedly. If it recognizes after several attempts that the error does not occur anymore, it does not take back its error bit at once. First further successful transmissions have to take place to count back the internal error counter again. Supervised Tx transfers which are addressed to other modules are also counted as successful transmissions. But if only one module and one CAN master are installed on the bus, the master possibly has to transmit some messages first to reset the error counter and therefore reset the controller status bit.
Therefore it is possible that the CAN-error bit of the CAN-status byte is still active after only one reset, because the error bit of the controller is still active.

Bit 7 of the byte has got the designation 'new on bus' and shows if the module processes for the first time on the CAN:
A CAN master is able to evaluate and set the bit to zero to document on the module that it noted the presence of the module on the CAN. If the bit has got the value '1' at the reading, in this application of the bit the module had not been found by a master so far after the last RESET.

A 'set' access with any data (recommended: byte 5 = $00) onto the status byte sets the bit onto '0'.

Sub command $03 --▸ set bitrate

bust0, bust1.. These two bytes set the contents of the registers BTR0 and BTR1 of the CAN controller SJA1000 which determine the bitrate of the CAN interface.

An allocation of the register contents to the bit rates can be taken from the table below.

Contrary to the other commands this parameters only become active after the actual parameter set was stored in the EEPROM (store parameters) and a RESET was triggered on the module.

If no communication is possible with the module, this is often due to a wrong adjustment of the bitrate.
If the local software discovers a malfunction on the CAN, the bitrate is adjusted again to the default value (adjustment at the configuration jumper of the module), but without changing the other parameters.

The specified typical line lengths base on experimental values from experience. The minimum reachable line lengths result from the 'worst case' delay times of the used components.

baud [hex]	CAN controller register		Bit rate [kBit/s]	Typical values of the reachable line length l_{max} [m]	Minimum values of the reachable line length l_{min} [m]
	BTR0 [HEX]	**BTR1 [HEX]**			
0	**00**	**14**	1000	37	20
1	**00**	**16**	800	59	42
2	**00**	**18**	666.6	80	65
3	**00**	**1C**	500	130	110
4	**01**	**18**	333.3	180	160
5	**01**	**1C**	250	270	250
6	**02**	**1C**	166	420	400
7	**03**	**1C**	125	570	550
8	**04**	**1C**	100	710	700
9	**45**	**2F**	66.6	1000	980
A	**09**	**1C**	50	1400	1400
B	**4B**	**2F**	33.3	2000	2000
C	**18**	**1C**	20	3600	3600
D	**5F**	**2F**	12.5	5400	5400
E	**31**	**1C**	10	7300	7300

The specifications in the table base on the limit values of the bit timing of the CAN protocol, the runtime of the local CAN interface and the runtime of the cable. The runtime of the cable is assumed with about 5.5 ns/m. Further influences, e.g., by the terminal resistances, the specific resistance, the geometry of the cable of outer interference effects at the transmission have not been included in the specifications!

Table 3.2.3: Allocation of the bitrate to the registers of the controller SJA1000

Sub command $04 --► set watchdog Tx identifier Sub command $05 --► set watchdog time (guard time)

The watchdog protocol functions with following scheme:

1. After a RESET, the watchdog is inactive. The 'master' interchanges with these sub commands a Tx identifier, a watchdog time (Guard Time) and a life time factor to the module. Setting the watchdog time to values > '0' and setting the life time factor to values > '0' enables the local watchdog on the module. The watchdog is not activated at that moment! The watchdog time (WDtime) and the watchdog life time factor (WDLifeTimeFactor) are ‘0’.

2. The watchdog Tx identifier has to be saved in the EEPROM. The watchdog time (WDtime) and the watchdog life time factor (WDLifeTimeFactor) are ‘0’ after RESET.

3. Now a remote request has to be received for this Tx identifier or the command 'Supervisor Watchdog' ($FF) has to be received, before the watchdog time is counted down for the first time. After this the master has to send a RTR or a supervisor command at the given Tx identifier within the time (WDtime x WDLifeTimeFactor), otherwise a RESET is triggered at the module. Receiving the RTR or the command shows the module that the master is still active.

4. If the remote request or the supervisor command arrives within the given time, the module transmits a one byte containing message back on the Tx identifier. The byte is constructed as follows:

Bit	7	6	5	4	3	2	1	0
Contents	Toggle bit	x	x	x	x	CMS State		

Table 3.2.4: Watchdog reply of the modules

The toggle bit changes its condition with each transmission. In normal position, i.e. before the first transmission, it has got the value '0'. The bits 6 to 3 are always transmitted as '1', if the module is in the state ‘preoperational’ and are always transmittet as ‘0’, if the module is in the state ‘operational’. The module turns into the state ‘operational’ after the ‘First Tx-activate Delay’. The bits 2 to 0 contain a CMS status message which is permanently programmed on '101' with all modules.
Possible replies therefore are in preoperational state: **$7D** and **$FD**.
Possible replies therefore are in operational state: **$05** and **$85**.

5. The master can recognize from the reply that the module is still active. After a RESET that is generated by the watchdog the module will not answer to receiving RTR frames of the master (first the watchdog time and the life time factor has to be set).

The interchanged parameters of the sub command $04 and $05 have the following meaning:

WTxId ... In this word the Tx identifier is interchanged on which the master transmits the remote request and on which the module transmits the response.

WDtime... The watchdog time after that a local RESET is generated results from the time set in WDtime multiplied by the life time factor, that is set in WDLifeTimeFactor. **The product has to be greater than '0' to enable the watchdog!**

WDtime [HEX]	Watchdog time in [ms]
0000	Watchdog disable (default adjustment)
0001 0002 0003 .. FFFF	1 2 3 ... 65.535

Table 3.2.5: Allocation of the parameters to the watchdog time

WDLifeTime Factor... Multiplier for the watchdog time (function is described at WDtime).
value: $00...$FF

Requesting the actual parameters by the module (command $01):

Command... The command $01 leads to the reply of the actual parameters.

Sub command... The sub command determines the configuration bytes which should be returned:

Sub command	Reply of the parameter
$00	Saved module no.
$01	Active module no.
$02	CAN-status byte
$03	Saved bitrate
$04	Watchdog Tx identifier
$05	Watchdog time

Table 3.2.6: Definition of the reply by sub command

Parameter 1 and 2... With these parameters it is reported to the module onto which CAN Id it should transmit the requested reply.

Decisive for the selection of the requested message is the sub command value received by the module.

The following table shows the information which the module transmits onto the CAN if a request command has been received.

Sub command	Reply of the parameter	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
$00	Saved mod. no.	$01	$00	saved mod no	-	-	-	-	-
$01	Active mod. no.	$01	$01	active mod no	-	-	-	-	-
$02	CAN status	$01	$02	cstat	-	-	-	-	-
$03	Saved bitrate	$01	$03	bust0	bust1	-	-	-	-
$04	Watchdog TxId	$01	$04	WTxId		-	-	-	-
$05	Watchdog time	$01	$05	WDtime		WDLife TimeFactor	-	-	-

(-) Byte is not transmitted

Table 3.2.7: Transmitting the actual parameters module no. and bitrate

Explanation of the bytes transmitted by the module:

Sub command $00 --▸ stored module no.

Byte 1... The first byte returns the contents of the received command byte (here always $01).

Byte 2... The second byte returns the contents of the received sub command byte (here always $00).

Byte 3: saved mod no... In this cell the module no. is returned which is actually stored in the EEPROM. After a power-on RESET or a RESET command this module no. is the number with which the module is selected if no other module no. had been stored after a RESET.
If the module operates with the default parameters, then the value $00 is returned here always.

Sub command $01 --▸ actual module no.

Byte1, Byte2... See sub command $00.

Byte 3: active mod no... Here the module no. is returned which is active at the moment. This number is identical with the number of this INIT Id entered into byte 4.

Sub command $02 --▸ CAN-status byte

Byte1,
Byte2... See sub command $00.

cstat... Byte 3 describes status information about the condition of the CAN components of the selected module (see the following table).

Bit	Designation
0	Power-down RESET
1	Suspend bit
2	(Error on CAN)
3	(I²C error)
4	EEPROM size
5	(I²C busy)
6	Default wake up
7	New on bus

Table 3.2.8: Bits of the parameter 'cstat'

Explanation of the bits of parameter 'cstat':

Power-down RESET (bit 0)
Bit 0 shows if the last RESET on the module had been triggered by a power-down RESET. If the bit has got the value '1', the last RESET had been triggered by removing the supply voltage. For all other RESET causes (RESET by supervisor command, RESET by EMERGENCY STOP) the bit is '0'.

Suspend bit (bit 1)
In bit 1 is described if the module is in condition 'suspended' or in active condition:

By the supervisor command 'suspend module' it is possible to put individual or all modules of the CAN into a delay condition. In this operating mode the actual condition of the outputs of the selected module remains unchanged (if available) and cannot be changed anymore until the module is enabled again. Parameters and commands are received and filed by the module but not evaluated: It is possible, e.g., to set raw data on the module without changing the normal conditions at once.

In condition 'suspended' the module does not transmit data onto the CAN. This is valid for each kind of Tx transfer with the exception of this configuration reply.
If the module is activated again by the command 'continue', possibly existing outputs are set at once corresponding to the last received data and the module operates with the last received parameters.

If bit 1 has got the value '1', the module is in condition 'suspended'. In normal operating mode (continue) bit 1 has got the value '0'.

Error on CAN (bit 2)
This bit is always ‘0’ at the CAN-CBM-SIO module.

I²C error (bit 3)
This bit is always ‘0’ at the CAN-CBM-SIO module.

EEPROM size (bit 4)
If bit 4 has got the value '1', the module has got an EEPROM whose memory capacity is bigger than 128 bytes. An EEPROM of this size is prerequisite for the allocation of pin names for each individual connection pin, because the names are filed in the EEPROM (see also chapter 'Allocation of Pin Names').

I²C busy (bit 5)
This bit is always ‘0’ at the CAN-CBM-SIO module.

Default wake up (bit 6)
If the bit 'default start' has the value '1', the module 'awoke' with the default parameters after the last RESET.

New on bus (bit 7)
The bit 'new on bus' can be evaluated by the CAN master to recognize if it already found this module after the last RESET or if the module is new on the bus. The bit is set to the value '1' after each RESET which corresponds to condition 'new on bus'. The master can reset the bit to show that it found the module. Apart from 'error on CAN' it is the only bit of this byte which can be reset.

Sub command $03 --▸ stored bitrate

Byte1,
Byte2... See sub command $00.

Byte 3, 4:
bust 0,
bust1 ... These two bytes describe the contents of the registers BTR0 and BTR1 of the CAN controller SJA1000 which determine the bitrate of the CAN interface. The allocation of the register contents to the bit rates has already been explained in the description of setting these registers.

Sub command $04 --▸ watchdog Tx identifier

Byte1,
Byte2... See sub command $00.

Byte 3, 4:
WTxId-H,
WTxId-L ... These two bytes return Tx identifiers on which the module transmits the watchdog reply.

Sub command $05 --▸ watchdog time

Byte1,
Byte2... See sub command $00.

Byte 3, 4:
WDtime... Here the watchdog time is returned (value range see 'Setting the watchdog time').
The value is only returned, if the value of WTxId is valid.

Byte 5:
WDLifeTime
Factor... Return of the Life Time Factor ($00...$FF).
The value is only returned, if the value of WTxId is valid.

3.3 Process TxIds

By this command one or more Tx identifiers are allocated to the module, depending on the module type, by which it is able to transmit the data onto the CAN (set TxIds).

The new Tx identifier replaces immediately after he had been received by the module the default identifier adjusted by the coding switches. If the new Tx identifier should remain after a RESET or a power-down, it has to be filed into the local EEPROM by command 'store parameter'.

Apart from that the module can be forced to transmit a reply about the Tx identifiers actually used by it onto the CAN by the Tx identifier CTxId (request TxIds).

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Function	Command	Sub command	Not used	Module no.	Parameter 1	Parameter 2
Value	Request: $02	$00...n	Always allocate with $00	Selected module no.	CTxId $0000...$07FF	
	Set: $82			$01...$FF	TxId $0000...$07FF	

Table 3.3.1: Bytes of the command set/request TxIds

Command... By the command $02 the actual Tx identifiers are requested by the module. After this request the module transmits the information on the identifier 'CTxId', entered into byte 5 and 6.
By the command $82 new Tx identifiers (TxId) to transmit I/O data are allocated to the module corresponding to the sub command. The new Tx identifiers are interchanged by byte 5 and 6.

Sub command... By the sub command it is selected which Tx identifier should be interchanged. Depending on the module type a different number of Tx identifiers can be set.

Parameter 1 and 2... With these parameters it is reported to the module at a request command onto which CAN Id (CTxId) it should transmit the requested reply.
The module only transmits once on this identifier. The identifier is not stored on the module. If another transmission is wanted, another INIT Id with the corresponding parameters has to be transmitted to the module again.

At a set command the 11 bit wide Tx identifier on which it should transmit the data is interchanged to the module by the parameters.

By the sub command it is selected which one of the possible Tx identifiers should be set or reset:

Sub command	Tx identifiers on parameters 1 and 2
$00	TxId1
$01	TxId2
$02	TxId3
:	:
n	TxId(n+1)

Table 3.3.2: Selection of the identifiers by the sub command

The following table shows how the bits of Tx identifiers have to be allocated to the bytes 5 and 6:

	Byte 5								Byte 6							
	Parameter 1								Parameter 2							
Bits in byte 5 and 6	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
	TxId-H								TxId-L							
Bits of the TxId	-	-	-	-	-	id11	id10	id9	id8	id7	id6	id5	id4	id3	id2	id1

(-) These bits are not evaluated.

Table 3.3.3: Bits of the Tx identifiers in byte 5 and byte 6

Reply of the modules to the command 'request TxIds':

The reply of a Tx identifier depends on the value of the received sub command.

The following table shows the information which the module transmits onto the CAN.

Sub command	Reply of the parameter	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
$00	TxId 1	$02	$00	TxId1		-	-	-	-
$01	TxId 2	$02	$01	TxId2		-	-	-	-
$02	TxId 3	$02	$02	TxId3		-	-	-	-
:	:	$02	:	:		-	-	-	-
n	TxId (n+1)	$02	n	TxId(n+1)		-	-	-	-

(-) Byte is not transmitted

Table 3.3.4: Transmitting the actual Tx identifiers

Explanation of the bytes transmitted by the module:

Byte 1... The first byte returns the contents of the received command byte (here always $02).

Byte 2... The second byte returns the contents of the received sub command byte. The sub command designates the selected identifier.

Byte 3,
Byte 4... In these two cells the actual Tx identifier is returned. The bits of the identifier are described in byte 3 and 4 and returned in that form as they have to be entered at setting in byte 5 and 6. (See also table 'Bits of the Tx identifier in byte 5 and 6'.)

3.4 Process RxIds

Depending on the module type one or more Rx identifiers are allocated to the module on which it is able to receive data to the CAN (set RxIds).

The new Rx identifier replaces the default identifier adjusted by the coding switches immediately after it had been received by the module. If the new Rx identifier should remain after a RESET or a power-down, it has to be filed into the local EEPROM by the command 'store parameter'.

Apart from this it is possible to make the module transmit a reply onto the CAN by the Tx identifier CTxId (request RxIds) about the Rx identifiers actually used by the module.

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Function	Command	Sub command	Not used	Module no.	Parameter 1	Parameter 2
Value	Request: $03	$00...n	Always allocate with $00	Selected module no.	CTxId $0000...$07FF	
	Set: $83			$01...$FF	RxId $0000...$07FF	

Table 3.4.1: Bytes of the command set/request RxIds

Command... By the command $03 the actual Rx identifiers are requested by the module. The module transmits the information on the identifier 'CTxId' entered in byte 5 and byte 6 after this request.

By the command $83 new Rx identifiers for the transmission of I/O data are allocated to the module. The new Rx identifier are interchanged by byte 5 and byte 6.

Sub command... By the sub command it is selected which Rx identifier should be interchanged. Depending on the module type a different number of Rx identifiers can be set.

Parameter 1 and 2... With these parameters it is reported to the module at a request command onto which CAN Id (CTxId) it should transmit the requested reply.
The module only transmits once on this identifier. The identifier is not stored on the module. If another transmission is wanted, another INIT Id with the corresponding parameters has to be transmitted to the module again.

At a set command the Rx identifier is interchanged to the module by these parameters on which it should transmit the data.

By the sub command it is selected which one of the possible Rx identifiers should be set or reset:

Sub command	Rx identifiers on parameter 1 and 2
$00	RxId1
$01	RxId2
:	:
n	RxId(n+1)

Table 3.4.2: Selection of the Rx identifiers by the sub command

The following table shows how the bits of Rx identifiers have to be allocated to the bytes 5 and 6:

	Byte 5								Byte 6							
	Parameter 1								Parameter 2							
Bits in byte 5 and 6	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
	RxId-H								RxId-L							
Bits of the RxId	-	-	-	-	-	id11	id10	id9	id8	id7	id6	id5	id4	id3	id2	id1

(-) These bits are not evaluated.

Table 3.4.3: Bits of the Rx identifier in byte 5 and byte 6

Reply of the modules to the command 'request RxIds':

The reply of the Rx identifiers depends on the value of the received sub command.

The following table shows the information which the module transmits to the CAN.

Sub command	Reply of the parameter	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
$00	RxId 1	$03	$00	Rx1S		Rx1E		-	-
$01	RxId 2	$03	$01	Rx2S		Rx2E		-	-
$02	RxId 3	$03	$02	Rx3S		Rx3E		-	-
:	:	$03	:	:		:		-	-
n	RxId (n+1)	$03	n	Rx(n+1)S		Rx(n+1)E		-	-

(-) Byte is not transmitted

Table 3.4.4: Transmitting the actual Rx identifiers

Explanation of the bytes transmitted by the module:

Byte 1... The first byte returns the contents of the received command byte (here always $03).

Byte 2... The second byte returns the contents of the received sub command byte. The sub command designates the selected identifier.

Byte 3-
Byte 6... In these cells the actual Rx identifier is returned. The bits of the identifier are described in byte 3 and 4 (or byte 5 and 6) and returned in that form as they have to be entered at setting in byte 5 and 6. (See also table 'Bits of the Rx identifier in byte 5 and 6'.)

Apart from the allocation of Rx identifiers described above there is the possibility to allocate, depending on the module type, an own identifier to each output bit, byte or word (see chapter 'Rx Block Mode'). If this block mode is activated, the first (Rxm_S) and the last (Rxm_E) identifier of the block are returned at an RxId request command (m=1,2,3,4...).

If the module is not in Rx block mode, the returned identifier in byte 3 and byte 4 is identical with the one in byte 5 and byte 6.

3.5 Cyclic Tx Transfers (Tx Activate Time)

The interchange of the parameter 'Tx activate time' makes the module transmit I/O data onto the CAN in regular periods of time.

This parameter has no effort on the CAN-CBM-SIO module !

The data are transmitted on the actual Tx identifier (see also chapter 'Setting/Reading TxIds', and 'Parameters after RESET').

Apart from that the module can be forced to transmit a reply by the Tx identifier CTxId onto the CAN about the 'activate times' it uses at the moment.

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Function	Command	Sub command	Not used	Module no.	Parameter 1	Parameter 2
Value	Request: $04	$00...n	Always allocate with $00	Selected module no.	CTxId $0000...$07FF	
	Set: $84			$01...$FF	tx_act $0000...$FFFF	

Table 3.5.1: Bytes of the command set/request TxIds

Command... By the command $04 the actual Tx activate times are requested by the module. After this request the module transmits the information on the identifier 'CTxId' of byte 5 and 6.
By the command $84 new Tx activate times to transmit I/O data are allocated to the module. The new period time is interchanged by byte 5 and byte 6.

Sub command... By the parameter sub command it is selected for which Tx identifier the new Tx activate time should be interchanged. Depending on the module type it is possible to allocate an activate time for a different number of Tx identifiers.

Parameter 1 and 2... By these parameters it is reported to the module at a request command onto which CAN Id (CTxId) it should transmit the requested reply.
The module transmits only once on this identifier. The identifier is not stored on the module. If another transmission is wanted, another INIT Id with the corresponding parameters has to be transmitted to the module again.

At a set command the activate time is interchanged to the module by these parameters.

By the sub command it is selected to which one of the four possible Tx identifiers the transmitted activate time should be allocated.

Sub command	Tx activate time (parameters 1 and 2)
$00	tx_act 1
$01	tx_act 2
$02	tx_act 3
:	:
n	tx_act (n+1)

Table 3.5.2: Selection of the Tx activate time by the sub command

The parameters of the activate time show the period of time from the last data transmission, after which the module automatically starts a new transmission.
The last data transmission could have been the previous Tx transfer of this cycle, but also every other Tx transfer transmitted by the module (e.g. a transmission initiated by a 'remote request').

tx_act [HEX]	Activate time in [ms]
0000	Function switched off
0001	1
0002	2
0003	3
..	...
FFFF	65.535

Table 3.5.3: Allocation of the parameters to the Tx activate time

Reply of the modules to the command 'request Tx activate time':

The reply of the activate time depends on the value of the received sub command.

The following table shows the information which the module transmits onto the CAN.

Sub command	Activate time for TxId	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
$00	TxId 1	$04	$00	tx_act1		-	-	-	-
$01	TxId 2	$04	$01	tx_act2		-	-	-	-
$02	TxId 3	$04	$02	tx_act3		-	-	-	-
:	:	$04	:	:		-	-	-	-
n	TxId (n+1)	$04	n	tx_act(n+1)		-	-	-	-

(-) Byte is not transmitted

Table 3.5.4: Transmitting the actual Tx activate times

Explanation of the bytes transmitted by the module:

Byte 1... The first byte always returns the contents of the received command byte (here always $04).

Byte 2... The second byte returns the contents of the received sub command. The sub command designates the selected identifier.

Byte 3,
Byte 4... In these two cells the actual Tx activate time is returned (see also 'parameter 1 and 2').

3.6 Process User Parameters

The user parameters are designed for module-specific applications. By this sub command it is possible to process up to 127 parameters with 16 bit width.

The user parameters are described in detail in the manual CAN-CBM-SIO/SIO4, **Manual of the User-specific Software**'.

The assignment of the user parameters can be taken from the module-specific software.

By the command 'set user parameters' the parameters on the module are set. The command 'request user parameters' makes the module return the parameter contents on the Tx identifier CTxId which had been interchanged in the request.

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Function	Command	Sub command	Not used	Module no.	Parameter 1	Parameter 2
Value	Request: $06	$00...$7F	Always allocate with $00	Selected module no. $01...$FF	CTxId $0000...$07FF	
	Set: $86				Para $0000...$FFFF	

Table 3.6.1: Bytes of the command set/request user parameters

Command... By the command $06 the parameter data are requested by the module. After this request the module transmits the information on the identifier 'CTxId' of byte 5 and 6.
By the command $86 the parameters are set.

Sub command... By the sub command the desired parameter is selected.

Parameter 1 and 2... By these parameters it is reported to the module at a <u>request</u> <u>command</u> onto which CAN Id (CTxId) it should transmit the requested reply.
The module transmits <u>only once</u> on this identifier. The identifier is not stored on the module. If another transmission is desired, another INIT Id with the corresponding parameters has to be transmitted to the module again.

At a <u>set command</u> these two bytes contain the 16 bits of the parameter which should be interchanged. The possible parameters have got numbers which correspond to the value of the corresponding sub command. A maximum of 127 ($7F) parameters is possible.

Reply of the modules to the command 'request user parameters':

The parameter returned by the module is selected by the corresponding sub command.

The following table shows the information which is transmitted to the CAN by the module.

Sub command	Parameter	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
$00	1	$06	$00	Para0		-	-	-	-
$01	2	$06	$01	Para1		-	-	-	-
...	...	...	...	...	-	-	-	-	-
$7F	127	$06	$7F	Para7F		-	-	-	-

(-) Byte is not transmitted

Table 3.6.2: Module transmits parameter contents

Explanation of the bytes transmitted by the module:

Byte 1... The first byte returns the contents of the received command byte (here always $06).

Byte 2... The second byte returns the contents of the received sub command byte.

Byte 3,4:
Para... In these cells the actual parameter contents are returned:

Output	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Bits of the parameter	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Allocation of the parameter to the bytes 5 and 6	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
	Byte 5								Byte 6							

Table 3.6.3: Allocation of the parameter bits to the outputs

3.7 Service Request

By this command it is possible to inquire the error condition of the *esd*-CAN modules. It is possible to make various or individual modules return a transmission if they show the error status 'error on CAN' or the status bit 'new on bus' is active.
The message consists of a Tx frame which has got no data.

The parameters of the modules remain unchanged when this command is requested.

The command can be used to find out which modules operate faultless on the bus and which modules have got transmission problems.

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Function	Command	Sub command 1	Sub command 2	Not used	Parameter 1	Parameter 2
Value	$7F	Module no_LOW $00...$FF	Module no_HIGH $00...$FF	Always allocate with $00	CTxId $0000...$07FF	

Table 3.7.1: Bytes of the command service request

<u>Explanation of the bytes transmitted to the module:</u>

Command... This command has got the value $7F.

Sub command1,
Sub command2... By the sub commands the modules are selected which should be checked. The modules are addressed by their module numbers.
To select an area with various modules, in sub command 1 the lowest and in sub command 2 the highest module no. of the desired area has to be entered. The limit values are included. If only one module should transmit a status message, the same module no. has to be entered for both sub commands.

Byte 4... In contrast to most of the other commands, here byte 4 and not byte 3 should be allocated with '$00'.

Parameter 1,
Parameter 2... In byte 5 and byte 6 the Tx identifier is interchanged on which the module should transmit a reply.

Reply of the module to the command 'service request':

If a module receives this command, and if a short term CAN error existed (bit 'error on CAN' = 1) or/and if the bit 'new on bus' is active, the module transmits a message with the length 0 on the received Tx identifier. No bytes are transmitted.

TxId	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
CTxId	-	-	-	-	-	-	-	-

(-) Byte is not transmitted

Table 3.7.2: Reply by CTxId

3.8 Supervisor Commands

In contrast to the other commands, supervisor commands are able to address individual or all modules in the CAN net.
By the module no. $00 all modules are selected. Therefore it is not recommendable to adjust the module no. $00 on a module.
The supervisor commands are not able to request parameters from the module.

The command RESET has got the same consequences as a 'power-on RESET'. It resets all parameters which are not stored in the EEPROM. After the RESET the parameters filed in the EEPROM are active, or, if the EEPROM data are defective or the EEPROM is not available, the default parameters.

If individual or all modules should be reset in a way that they operate with the default parameters after the RESET, this can be achieved by the command default RESET.

During a local RESET the message outputs are activated.

The command Supervisor Watchdog can be called by the Supervisor, to activate the watchdog functionality (precondition is that the watchdog time and the life time factor are greater than ‘0’). After the first activation the command has to be repeated within the time 'WDtime x WDLifeTimeFactor' otherwise a local RESET is generated at the module (refer to chapter 'System Parameter').

The command suspend module causes one or all modules on the CAN to 'freeze' the actual condition of their outputs (if available) and not transmit any messages (Tx transfers) to the CAN (exception: configuration reply).

Parameters and commands are received and filed by the module, but not evaluated: It is possible therefore, e.g., to set output data on the module without changing the output conditions at once.

If the module is activated again by the command 'continue', possibly available outputs are set immediately according to the last received data and the module operates with the last received parameters.

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Function	Command	Sub command	Not used	selected module no.	Parameter 1	Parameter 2
Value	$FF	$00 Trigger RESET	Always allocate with $00	$00 All modules selected $01..$FF One module selected	$AAAA - RESET	
		$01 Reserved			-	
		$02 Supervisor Watchdog			-	
		$03 Default RESET			$AAAA - Default RESET	
		$04 Suspend/continue module			$5A5A - Suspend $A5A5 - Continue	
		$05 Reset CAN-error bit			-	

Table 3.8.1: Bytes of the supervisor commands

Explanation of the bytes transmitted to the module:

Command... The supervisor commands all have got the value $FF.

Sub command... The sub command selects the corresponding supervisor command:

Sub command	Function
$00	Reset module
$01	Reserved
$02	Supervisor Watchdog
$03	Reset module and activate the default parameters
$04	Suspend module
$05	Reset CAN error bit

Table 3.8.2: Selection of the supervisor commands by the sub command

Module no. ... Here the desired actual module no. is entered. The setting of individual modules occurs by the module no. $01 to $FF.

At this command all *esd* modules connected to this CAN net are accessed at the same time by the module no. $00!

Parameter 1,.. Depending on the sub command here are parameters interchanged, which are specific for the function:

The commands RESET, default RESET, suspend/continue modules and resetting the CAN error bits are only carried out, if exactly the specified parameters are interchanged.

4. Examples for Parameterization

In this chapter the operation and the initialisation of a module should be clarified with some examples.

4.1 Setting the Tx Identifier TxId1

In this example it will be shown how the Tx identifier TxId1 is programmed on a module. The Tx identifier should get the value $1AA. The Tx identifier had not been programmed at this module so far.

Before the new identifier is set, the module should transmit a reply by the actual Tx identifier on the Tx identifier $567. The reply is requested by command $02.
As actual module no. the value $12 will be presumed here.

CAN Id	Byte 1 (command)	Byte 2 (sub command)	Byte 3 (always $00)	Byte 4 (module no.)	Byte 5 (CTxId-H)	Byte 6 (CTxId-L)
$700 =INIT Id	$02	$00	$00	$12	$05	$67

Table 4.1.1: Request of the Tx identifier

Byte 7 and 8 are not needed for this command.

After this INIT Id had been transmitted to the module, the module responds on the Tx identifier $567:

CAN Id	Byte 1 (returns command)	Byte 2 (returns sub command)	Byte 3 (returns TxId1-H)	Byte 4 (returns TxId1-L)
$567	$02	$00	$00	$92

Table 4.1.2: Reply of the Tx identifier

Byte 5 to 8 are not transmitted.

As can be seen from byte 3 and 4 the module has actually got the Tx identifier TxId1 $092.

Examples

Now the Tx identifier TxId1 (=$1AA) is allocated to the module:

CAN Id	Byte 1 (command)	Byte 2 (sub command)	Byte 3 (always $00)	Byte 4 (module no.)	Byte 5 (TxId-H)	Byte 6 (TxId-L)
$700 =INIT Id	$82	$00	$00	$12	$01	$AA

Table 4.1.3: Setting the Tx identifier TxId1

Byte 7 and 8 are not needed for this command.

The Tx identifier TxId1 of the module is now assigned with the value $1AA. The new identifier is active immediately after it has been received by the module.

4.2 Restoring the Default Parameters

This example should show how a module whose default parameters have been changed during an initialisation is put back into the original condition.

There are various ways to reactivate the default parameters.

4.2.1 ...if the actual module no. is unknown:

If only on this *esd*-CAN module the default parameters should be restored, you have to proceed as follows:

The coding switches of the module have to be set to $00 and a RESET has to be triggered.

Because the module no. is unknown, the RESET is triggered by interrupting the supply voltage of the module or the voltage at the EMERGENCY-STOP outputs or by triggering a global module RESET on the CAN (RESET all modules).

This global RESET resets all modules. But the default parameters are only restored on those modules whose coding switches are adjusted to $00.

CAN Id	Byte 1 (command)	Byte 2 (sub command)	Byte 3 (always $00)	Byte 4 (RESET type)	Byte 5	Byte 6
$700 =INIT Id	$FF	$00	$00	$00 (global)	$AA	$AA

Table 4.2.1: RESET of all *esd* modules of the CAN

The bytes 7 and 8 are not needed for this command.

If on all *esd*-CAN modules on the CAN the default parameters should be reactivated, the global command 'default RESET' is given to the bus:

CAN Id	Byte 1 (command)	Byte 2 (sub command)	Byte 3 (always $00)	Byte 4 (RESET type)	Byte 5	Byte 6
$700 =INIT Id	$FF	$03	$00	$00 (global)	$AA	$AA

Table 4.2.2: Default RESET of all *esd* modules of the CAN

The bytes 7 and 8 are not needed for this command.
After this RESET the module starts again with the default parameters. By coding switches the desired identifier can be determined again.

4.2.2 ...if the module no. is known:

If the module no. is known it is possible to restore the default parameters by the command 'default RESET' at the desired module. The module no. of the module is presumed with $12 here, again:

CAN Id	Byte 1 (command)	Byte 2 (sub command)	Byte 3 (always $00)	Byte 4 (RESET type)	Byte 5	Byte 6
$700 =INIT Id	$FF	$03	$00	$12 (selective)	$AA	$AA

Table 4.2.3: Default RESET of an *esd* module on the CAN

The bytes 7 and 8 are not needed for this command.