Systemhandbuch

Platinensteuerung

CS0015

Systemhandbuch Platinensteuerung CS0015, Stand Oktober 1999

Hinweis zur Gewährleistung

Dieses Handbuch wurde unter Beachtung der größtmöglichen Sorgfalt erstellt. Gleichwohl kann keine Garantie für die Richtigkeit des Inhalts übernommen werden.

Da sich Fehler trotz intensiver Bemühungen nie vollständig vermeiden lassen, sind wir für Hinweise jederzeit dankbar.

Im übrigen behalten wir uns technische Änderungen der Produkte vor, so daß sich auch soweit Abweichungen vom Inhalt des Handbuches ergeben können.

1. Allgemeines

1.1. Sicherheitshinweise

Befolgen Sie die Angaben der Beschreibung. Nichtbeachten der Hinweise, Betrieb außerhalb der nachstehend bestimmungsgemässen Verwendung, falsche Installation oder fehlerhafte Handhabung können schwerwiegende Beeinträchtigungen der Sicherheit von Menschen und Anlagen zur Folge haben.

Die Anleitung richtet sich an Personen, die im Sinne der EMV- und der Niederspannungs-Richtlinie als “fachkundig” angesehen werden können. Die Steuerungen sind von einer Elektrofachkraft **(**Programmierer bzw. Servicetechniker) einzubauen und in Betrieb zu setzen.

Diese Beschreibung ist Bestandteil des Gerätes. Sie enthält Texte und Abbildungen zum korrekten Umgang mit der Steuerung und muß vor einer Installation oder dem Einsatz gelesen werden.

Es ist darauf zu achten, daß die externe Spannung gemäß den Kriterien für sichere Kleinspannung (SELV) erzeugt und zugeführt wird, da diese ohne weitere Maßnahmen zur Versorgung der angeschlossenen Steuerung, der Sensorik und der Aktorik zur Verfügung gestellt wird.

Die Verdrahtung aller in Zusammenhang mit dem SELV-Kreis des Geräts stehenden Signale muß ebenfalls den SELV-Kriterien entsprechen (sichere Schutzkleinspannung, galvanisch sicher getrennt von anderen Stromkreisen).

Wird die zugeführte SELV-Spannung extern geerdet (SELV wird zu PELV), so geschieht dies in der Verantwortung des Betreibers und im Rahmen der dort geltenden nationalen Installations-Vorschriften. Alle Aussagen in dieser Bedienungsanleitung beziehen sich auf das bezügl. der SELV-Spannung nicht geerdete Gerät.

An den Anschlußklemmen dürfen nur die in den technischen Daten, bzw. auf dem Geräteaufdruck angegebenen Signale eingespeist bzw. die zugelassenen Zubehörkomponenten der ifm electronic gmbh angeschlossen werden.

Die beim Betrieb der Steuerung erzeugte Abwärme muß frei entweichen können, d.h. beim Einbau ist auf hinreichende Konvektion und Berührschutz von heißen Bauteilen zu achten.

Bei Fehlfunktionen oder Unklarheiten setzen Sie sich bitte mit dem Hersteller in Verbindung. Eingriffe in das Gerät können schwerwiegende Beeinträchtigungen der Sicherheit von Menschen und Anlagen zur Folge haben. Sie sind nicht zulässig und führen zu Haftungs- und Gewährleistungsausschluß.

1.2. Bestimmungsgemäße Verwendung

Die Platinensteuerung ecomat 100 Typ CS0015 (im folgenden Text CS0015) ist für den Industrieeinsatz vorgesehen. Sie muß gemäß den gültigen Vorschriften in ein Gehäuse oder einen Schaltschrank eingebaut und betrieben werden.

Die Steuerung ist mit integrierten CMOS-Bauteilen bestückt. Diese können durch elektrostatische Entladungen zerstört werden. Solche Entladungen können bereits bei der Berührung mit der Hand auftreten. Es sind entsprechende Maßnahmen zur Verhinderung bzw. Ableitung der elektrostatischen Entladungen bei Transport, Montage, Programmierung, Einstellung an Schaltern und Betrieb der Steuerung vorzunehmen.

Die Steuerung CS0015 ist nicht für sicherheitsrelevante Aufgaben im Sinne des Personenschutzes zugelassen.

Die Applikationssoftware kann vom Anwender komfortabel mit dem Programmiersystem ecolog 100*plus* selbst erstellt werden.

Alle in dieser Dokumentation beschriebenen Software-Funktionen und Programmierverfahren beziehen sich auf die Programmiersoftware ecolog 100*plus* dessen Kenntnis in dieser Beschreibung vorausgesetzt wird.

Der Anwender muß außerdem beachten, welcher Softwarestand (speziell beim Betriebssystem und den Funktionsbibliotheken) zum Einsatz kommt. Softwarestände werden durch nachgestellte Buchstaben in alphabetischer Reihenfolge in den Dateinamen (z.B. CS0015_G.M66 oder TDM_C.LIB) gekennzeichnet. Bei Überarbeitung bestehender Applikationsprojekte muß man sich gegebenenfalls über Inkompatibilitäten zwischen den alten und neuen Versionsständen informieren.

Für die sichere Funktion der Applikationsprogramme, die vom Anwender erstellt werden, ist dieser selbst verantwortlich. Bei Bedarf muß er zusätzlich entsprechend der nationalen Vorschriften eine Abnahme durch entsprechende Prüf- und Überwachungsorganisationen durchführen lassen.

1.3 Technische Daten

Bauform:	offene Platine
Maße:	162 x 126 x 75 mm (BxHxT)
Geräteanschluß:	Spannungsversorgung und Ein-/Ausgänge: Phoenix Contact CMBICON RM5,08 CAN-/Serielle-Schnittstelle: Phoenix Contact MINI-CMBICON RM3,81
Betriebstemperatur:	0°C ... +40°C
Schutzart:	IP00 , Verschmutzungsgrad 2
Anschlußspannung:	U_B nominal 24 V DC (-15% ... +25%)
Stromaufnahme:	≤ 150 mA, ohne externe Last
Prozessor:	CMOS-Microcontroller C 167C
Geräteanzeige:	2 LEDs (rot und grün) zur Status- und Fehleranzeige
Geräteüberwachung:	Watchdog (200ms)
Speicher:	256 kByte Programmspeicher 256 kByte Datenspeicher (flüchtig) davon 1 kByte Datenspeicher spannungsausfallsicher (256 Byte autosave)
Schnittstellen:	CAN, Version 2.0 (ISO/DIS-11898), 10 ... 1000 kBaud Protokoll: CANopen oder freies Kommunikationsprofil Geräteklasse: CANopen Master/Slave; CAN: FullCAN Serielle Schnittstelle RS 232 C, 9,6 kBaud Teilnehmer-Anzahl: 2 (Master/Slave)

Binärer Eingänge IN0 ... IN15

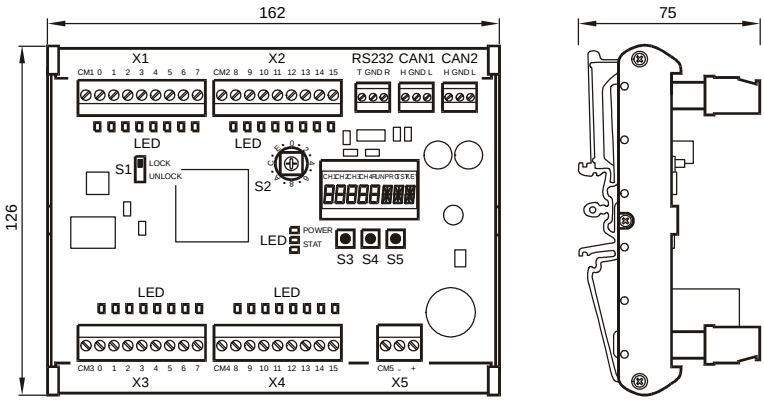
Eingänge IX0.0 ... IX0.15:	gemeinsamer Bezugspunkt GND	
	optische Anzeige	LED gelb
	Eingangsspannung	24 C DC (nominal)
	Gleichzeitigkeitsfaktor	100 % bei 24 V DC 50 % bei 30 V DC
	Einschaltpegel	+ 15 ... + 30 V DC
	Ausschaltpegel	0 ... + 5,5 V DC (bzw. Eingangsstrom < 1,5 mA)
	Eingangsfrequenz	500 Hz (IN0 ... IN3) 25 Hz (IN4 ... IN15)

Binärer Ausgänge OUT0 ... OUT15

Ausgänge QX0.0 ... QX0.15:	gemeinsame Versorgungsspannung für je 8 Ausgänge (X3/X4) +24 V DC.	
	optische Anzeige	LED rot
	Schaltspannung	12 ... 34 V DC, nominal 24 V DC
	Schaltstrom	1,1 A
	Gleichzeitigkeitsfaktor	100%
	Kurzschlußschutz	> 6 A (elektronisch)
	Ausgangsfrequenz	max. 200 Hz

Bei Beachtung genauer Spezifikationen können auch größere Ströme (max. 1,9 A) geschaltet werden.

Maßskizze:



1.4. Montage der Steuerung

Die Platinensteuerung wird in einen Trägergehäuse zur Tragschinenmontage ausgeliefert. Die Montage kann auf Tragschinen vom Typ TS 32 bzw. TS 35 erfolgen.

Es ist darauf zu achten, daß die beim Betrieb der Steuerung erzeugte Abwärme frei entweichen kann.

1.5. Elektrischer Anschluß

Vor der Inbetriebnahme ist zu beachten, daß folgende Anschlüsse mit den zugehörigen Potentialen belegt werden.

Bezeichnung	Pin-Nr.	Potential
Versorgungsspannung	X5 +	+ 24 V DC
Masse	X5 -	GND
Störschutz GND	X5 CM5	GND
Versorgungsspannung Ausgänge 0 ... 7 (High-Side)	X3 CM3	+ 24 V DC
Versorgungsspannung Ausgänge 8 ... 15 (High-Side)	X4 CM4	+ 24 V DC
Versorgungsspannung Ausgäne Low-Side ohne Überwachungsrelais	15 (GNDo)	GND
Störschutz GND Eingänge	X1 CM1	GND
Störschutz GND Eingänge	X2 CM2	GND
Programmierschnittstelle RS 232	(RxD)	Pin 03, PC 9pol. SUB-D
	(TxD)	Pin 02, PC 9pol. SUB-D
	(CM_5)	Pin 05, PC 9pol. SUB-D
CAN-Interface	(CAN_H)	CAN_H weiterer Teilnehmer
	(CAN_L)	CAN_L weiterer Teilnehmer
	(CAN_{GND})	GND weiterer Teilnehmer

Um bei besonderen Umgebungsbedingungen einen verbesserten elektrischen Störschutz der Steuerung sicherzustellen, können die Störschutz GND X1, X2 (Eingänge) und X5 (Spannungsversorgung) mit Erde (GND) verbunden werden.
Über diese Anschlüsse besteht keine Verbindung zum GND der Spannungsversorgung der Steuerung.

1.5. Absicherung der Steuerungsmodule

Die Ausgangskanäle sind pro Kanal elektronisch gegen Überlast und Kurzschluß (> 6 A) gesichert. Dennoch empfielt es sich zum Schutz des gesamten Systems (Verkabelung und Steuerung) die einzelnen Stromkreise entsprechend getrennt abzusichern. Dabei ist auch der Summenstrom von 10 A der einzelnen Ausgangsgruppen (max. 8 Ausgänge - z.B. OUT0 ... OUT7) zu beachten.

2. LCD-Anzeige und Bedienelemente

Die CS0015 ist mit einer LCD-Anzeige, drei frei programmierbaren Drucktastern, einem Drehschalter und dem Schalter für die Programmierfreigabe ausgerüstet.

Diese können zum Beispiel zur Parametrierung bei der Maschineneinrichtung eingesetzt werden. Bedingt durch ihre Lage direkt auf der Leiterplatte, sollte die Bedienung nur durch Fachpersonal erfolgen.

Zur dauerhalten Bedienung von Maschinenfunktionen sind diese Elemente nicht geeignet. In solchen Fällen sollte dann über den CAN-Bus eines der Dialoggeräte der ifm electronic (z.B. vollgrafisches Display CR1000 oder Datendisplay CS0014) angeschlossen werden. Diese sind durch ihren mechanischen Aufbau den harten Anforderungen des Industrieeinsatzes gewachsen.

2.1. Schalter S1 Programmierfreigabe

Mittels dieses Schiebeschalters kann die Steuerung in den Programmier- bzw. Betriebsmodus versetzt werden.

In der Schalterstellung ***LOCK*** ist der Betriebsmodus aktiviert und der Programmspeicher ist gegen Datenverlust geschützt.

Soll in die Steuerung ein neues Programm geladen bzw. eine Kommunikationsverbindung zwischen Steuerung und dem Programmiersystem ecolog 100^{plus} hergestellt werden, muß der Schalter in der Position ***UNLOCK*** stehen.

Es ist darauf zu achten, daß im normalen Betrieb, der Schalter auf der Position *LOCK* steht, da Datenverlust auch durch Störimpulse, d.h. ohne eine Verbindung zum Programmiersystem, auftreten können.

2.2. Drehcodierschalter S2

Der Drehcodierschalter kann zur Auswahl im Programm fest abgelegter Parameter (z.B. Zeiten und Zählerstände) oder bestimmter Programmabläufe (z.B. Einrichtbetrieb) eingesetzt werden. Der Schalter kann mit einem kleinen Schlitzschraubendreher verstellt werden. Er ist als Endlosschalter ausgeführt , d.h. er hat keinen mechanischen Anschlag.

Die Schalterstellung 0 ... 15 (0 ... F Hex) kann über die Systemvariable S2 (IEC-Adresse %IB2) abgefragt und im Anwenderprogramm weiter verarbeitet werden.

2.3. Drucktaster S3 ... S5

Die in der Nähe der LCD-Anzeige angeordneten Drucktaster können z.B. zur Steuerung von Anzeigefunktionen genutzt werden. Die Taster sind mit je einem Schließerkontakt ausgerüstet.

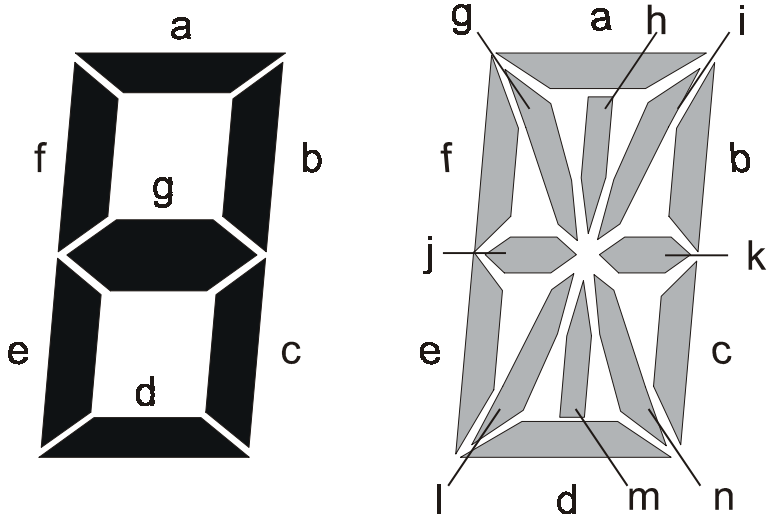
Die Tasterbetätigung kann über die Systemvariablen S3 ... S5 (IEC-Adresse %IX1.8, %IX2.0, %IX2.8) als Bitinformation (TRUE) abgefragt und im Anwenderprogramm weiter verarbeitet werden.

2.4. LCD-Anzeige

Die Steuerung CS0015 ist mit einer LCD-Anzeige ausgerüstet. Diese Anzeige kann z.B. zur Anzeige von Betriebszuständen eingesetzt werden. Alle Anzeigeelemente können über das Anwenderprogramm frei programmiert werden. Dazu können über die Funktionsaufrufe LCD_SEGMENTS und LCD_TEXT einzelne Anzeigesegmente bzw. direkt Zahlen und Buchstaben dargestellt werden.

Die Anzeige hat folgende Elemente:

5 x 7-Segment
3 x 14-Segment
8 Festtexte



Die einzelnen Anzeigesegmente werden mittels Buchstaben gekennzeichnet. Das einzelne Segment wird gesetzt, wenn das entsprechende Bit im ARRAY gesetzt ist.

Durch die Funktion STR, kann ein Zahlenwert in einen String (Zeichenkette) umgewandelt werden. Das Ergebnis dieser Funktion kann direkt als Eingangswert für die Funktion LCD_TEXT genutzt werden.

3. Betriebszustände und Betriebssystem

3.1. Betriebszustände

Nach Anlegen der Versorgungsspannung kann sich die Steuerungsmodul in einem von 5 möglichen Betriebszuständen befinden:

Reset

Dieser Zustand wird nach jedem Power-On-Reset durchlaufen. Das Betriebssystem wird initialisiert. Verschiedene Checks werden durchgeführt. Dieser nur temporäre Zustand wird vom Run-Zustand abgelöst.

⇨ Die LEDs STAT leuchten kurzzeitig rot und grün

Run

Dieser Zustand wird erreicht:

- aus dem Reset-Zustand (Autostart)
- aus dem Stop-Zustand durch das Run-Kommando
  Voraussetzung: Test-Betrieb
- Durch den CANopen NMT-Master durch die Funktion PREOPERATIONAL oder OPERATIONAL

⇨ Die LED STAT blinkt grün oder blinkt rot (RUN mit Fehler)

Stop

Dieser Zustand wird erreicht:

- aus dem Reset-Zustand, wenn kein Programm geladen ist
- aus dem Run-Zustand, indem über die Schnittstelle das Stop-Kommando gegeben wird.

Voraussetzung: Test-Betrieb

- Durch den CANopen NMT-Master durch die Funktion PREPARED.

⇨ Die LED STAT leuchtet konstant grün

Fatal Error

In diesen Zustand fällt die Steuerung, wenn ein nicht tolerierbarer Fehler festgestellt wurde. Dieser Zustand kann nur durch einen Reset verlassen werden.

⇨ Die LED STAT leuchtet konstant rot

Kein Betriebssystem

Es wurde kein Betriebssystem geladen, die Steuerung befindet sich im Bootlader. Vor dem Laden der Applikationssoftware muß ein Betriebssystemdownload durchgeführt werden.

⇨ Die LED STAT blinkt grün (schnell).

3.2. Status-LEDs

Diese Betriebszustände werden durch zwei Status-LEDs (LED STAT) in den Farben rot und grün angezeigt.

LED-Farbe	Blinkfrequenz	Beschreibung
grün/rot	konstant	Reset-Checks
Grün	5 Hz	Kein Betriebssystem geladen
Grün	0,5 Hz	Run, CANopen: PREOPERATIONAL
	2,0 Hz	Run, CANopen: OPERATIONAL
	konstant ein	Stop, CANopen: PRERPARED
Rot	0,5 Hz	Run m. Fehler (CANopen: PREOPERATIONAL)
	2,0 Hz	Run m. Fehler (CANopen: OPERATIONAL)
	konstant ein	Fatal Error

Die Betriebszustände STOP (PREPARED) und RUN (PRE-OPERATIONAL / OPERATIONAL) können vom Programmier-system oder vom NMT-Master geändert werden.

Im Zustand RUN wird das Anwenderprogramm abgearbeitet. An der CANopen-Kommunikation (PDO-Verarbeitung, siehe Kap. 5.ff) nimmt die Steuerung aber nur teil, wenn sie in den Zustand OPERATIONAL gesetzt wird. Um den aktuellen Betriebszustand im Applikationsprogramm zu erkennen, kann der Anwender den Merker COP_PREOPERATIONAL auswerten. Der Merker ist TRUE im Zustand PREOPERATIONAL andernfalls ist er FALSE.

3.3. Betriebssystem laden

Im Auslieferungszustand ist im Normalfall kein Betriebssystem in der Steuerung geladen (LED STAT blinkt grün mit 5 Hz). In diesem Betriebszustand ist nur der Boot-Lader aktiv. Dieser stellt die minimalen Funktionen für den Betriebssystem-Ladevorgang zur Verfügung (u.a. die Unterstützung der seriellen und der CAN Schnittstelle).

Der Betriebssystemdownload muß im Normalfall nur einmal durchgeführt werden. Das Applikationsprogramm kann anschließend (auch mehrfach) in die Steuerung geladen werden, ohne das Betriebssystem zu beeinflussen. Der Vorteil dieses Verfahrens liegt darin, daß bei einem Betriebssystem-Update kein EPROM getauscht werden muß und für bestimmte Applikationen kundenspezifische Betriebssysteme realisiert werden können.

Das Betriebssystem wird zusammen mit dieser Dokumentation auf einem separaten Datenträger zur Verfügung gestellt.

Der Programmierer muß beachten, das immer der gleiche Softwarestand des Betriebssystems (CS..._x.H86), der Steuerungskonfiguration (CS..._x.M66) und der Gerätebibliothek (CS..._x.LIB) zum Einsatz kommt. Andernfalls wird bei Download der Applikationsoftware eine Fehlermeldung erzeugt. Softwarestände werden durch nachgestellte Buchstaben in alphabetischer Reihenfolge in dem Dateinamen (z.B. CS0015_G.H86) gekennzeichnet. Der Basisdateiname muß immer gleich sein.

Betriebssystem Download

Das Betriebssystem wird, wie auch die Applikationssoftware, direkt aus dem Programmiersystem geladen. Der Download kann über die serielle und über die CAN-Schnittstelle erfolgen. Folgende Punkte sind dabei zu beachten:

Neue Steuerung

Im Auslieferungszustand des Steuerungsmoduls ist noch kein Betriebssystem geladen. Beim Anlegen der Versorgungsspannung geht dieses daher in den Zustand ´Kein Betriebssystem geladen´. Es ist nur der Bootlader aktiv.

⇨ Zum Download muß man über den Button oder den Menüpunkt *Fenster / Steuerungskonfiguration* das Steuerungskonfigurationsfenster aktivieren.

⇨ Über den Menüpunkt *Einfügen / Firmware* wird die gewünschte Steuerungskonfiguration (CS..._x.M66) aufgerufen.

⇨ Anschließend kann durch *Online / Einloggen* die Verbindung zwischen Steuerung und PC aufgebaut werden. Über welche Schnittstelle die Verbindung hergestellt wird, hängt von der Einstellung in *Extras / HW-Konfig* (Seriell oder CAN) und der anschließenden Parametrierung der PC-Schnittstelle unter *Online / Kommunikatiosparameter...* ab.

Es wird nur eine Kommunikationsverbindung zur Steuerung aufgebaut, wenn ein Projekt geladen ist und dieses fehlerfrei übersetzt wird.

⇨ Der Downloadvorgang wird durch Anwahl des Menüpunktes *Extras / Hexfile laden* im Fenster *Steuerungskonfiguration* gestartet.

Für alle Applikationsprogramme, die nun in die Steuerung geladen werden sollen, muß nun die neue Steuerungskonfigurationsdatei genutzt werden.

Betriebssystem-Update

Grundsätzlich kann auch zu einem späteren Zeitpunkt eine neue Betriebssystemsoftware in die Steuerung geladen werden. Dieser Vorgang entspricht in den wesentlichen Teilen dem oben beschriebenen.

Im Unterschied zum Auslieferungszustand der Steuerung ist jetzt schon ein Betriebssystem geladen, d.h. die Steuerung befindet sich im STOP- oder RUN-Modus.

⇨ Damit das Programmiersystem die Online-Verbindung zwischen Steuerung und PC herstellen kann, muß zunächst die Steuerungskonfiguration des zum aktuellen Zeitpunkt geladenen Bertriebssystems aktiviert werden.

⇨ Dazu wird über den Button oder den Menüpunkt *Fenster / Steuerungskonfiguration* das Steuerungskonfigurationsfenster aktiviert.

⇨ Über den Menüpunkt *Einfügen / Firmware* wird die gewünschte Steuerungskonfiguration (CS..._x.M66) aufgerufen.

⇨ Anschließend kann durch *Online / Einloggen* die Verbindung zwischen Steuerung und PC aufgebaut werden. Über welche Schnittstelle die Verbindung hergestellt wird, hängt von der Einstellung in *Extras / HW-Konfig* (Seriell oder CAN) und der anschließenden Parametrierung der PC-Schnittstelle unter *Online / Kommunikatiosparameter...* ab.

Dabei ist es unerheblich welche Projektdatei (es muß aber ein lauffähiges Projekt mit der Routine PLC_PRG sein) geladen ist. Der Übersetzungsvorgang, der beim Einloggen ausggelöst wird, kann ignoriert werden. Die Systemmeldung:

Programm wurde geändert! Soll das neue Programm geladen werden?

kann mit *Nein* beantwortet werden.

⇨ Durch Anwahl des Menüpunktes *Extras / Hexfile laden* im Fenster *Steuerungskonfiguration* wird das aktuelle Betriebssystem in der Steuerung gelöscht. Die LED des Steuerungsmoduls blinkt danach schnell (5 Hz).

⇨ Anschließend muß ein Reset der Steuerung durchgeführt werden, da mit Löschen des Betriebssystems keine Online-Verbindung zwischen PC und Steuerung mehr besteht.

⇨ Nach dem Reset kann das neue Betriebssystem geladen werden. Der Ablauf entspricht dem bei „neue Steuerung“ beschriebenen Vorgang.

Für alle Applikationsprogramme, die ab diesem Zeitpunkt in die Steuerung geladen werden sollen, muß nun die neue Steuerungskonfigurationsdatei genutzt werden

3.3. Betriebsmodi

Unabhängig von den Betriebszuständen kann die Steuerung in verschiedenen Betriebsmodi betrieben werden. Die entsprechenden Steuerungs-Bits können über die Applikationssoftware oder im Programmierbetrieb mit der Programmiersoftware ecolog 100^{plus} (Fenster: Globale Variablen) gesetzt und rückgesetzt werden.

Programmierung

Dieser Betriebsmodus wird aktiviert wenn sich der Schiebeschalter S1 in der Position ***UNLOCK*** befindet. Jetzt kann die Steuerung im RUN- oder STOP-Zustand Kommandos über eine der Schnittstellen entgegennehmen. Über den Merker UNLOCK kann der Zustand vom Anwenderprogramm abgefragt werden.

Serial Mode

Die serielle Schnittstelle steht für den Datenaustausch in der Applikation zur Verfügung. Ein Debugging der Applikations-software ist nur noch über die CAN-Schnittstelle möglich.

Diese Funktion ist standardmäßig abgeschaltet (FALSE). Über den Merker SERIAL_MODE kann der Zustand über das Anwenderprogramm oder das Programmiersystem gesteuert und abgefragt werden.

4. Fehlercodes und Fehlerklassen

Um eine möglichst hohe Betriebssicherheit zu gewährleisten, wird vom Betriebssystem aus die Steuerung in der Startphase (Reset-Phase) und während der Programmausführung durch interne Fehlerchecks überprüft.

Folgende Fehlermerker werden im Fehlerfall gesetzt:

Fehler	Fehlerbeschreibung
CAN_INIT_ERROR	CAN-Baust. kann nicht initialisiert werden
CAN_DATA_ERROR	CAN inkonsistente Daten
CAN_RX_OVERRUN_ERRO R	CAN overrun, Empfangsdaten
CAN_TX_OVERRUN_ERROR	CAN overrun, Sendedaten
CAN_BUS_OFF_ERROR	CAN nicht am Bus
CAN_ERROR	CAN-Bus Sammelfehlerbit
ERROR	Sammelfehlerbit (allgemein)
ERROR_MEMORY	Speicherfehler
COP_SYNCFAIL_ERROR	SYNC-Objekt wurde nicht übertragen
COP_GUARDFAIL_ERROR	Guarding-Objekt fehlt (nur im Slave)
COP_GUARDFAIL_NODEID	Nummer des fehlenden Slaves (nur im Master)

4.1. Reaktion auf System-Fehler

Es liegt grundsätzlich in der Verantwortung des Programmierers auf die Fehlermerker zu reagieren.

Die spezifischen Fehlerbits sollten im Anwenderprogramm verarbeitet werden und müssen anschließend zurückgesetzt werden. Über das Fehlerbit erhält man eine Fehlerbeschreibung. Diese kann bei Bedarf noch weiter verarbeitet werden.

Bei schweren Fehlern kann zusätzlich das ERROR-Bit gesetzt werden. Das bewirkt gleichzeitig, daß die LED STAT rot leuchtet undl die Ausgänge abgeschaltet werden.

Je nach Applikation muß nun entschieden werden, ob durch Rücksetzen des ERROR-Bit die Ausgänge wieder eingeschaltet werden dürfen.

Bei Einsatz von CAN zur Kommunikation sollte auf jeden Fall die Funktion CAN_ERRORHANDLER eingesetzt werden. Dadurch werden zumindest alle CAN-Fehler als Sammelstörung erkannt, gezählt und CAN erneut gestartet.

Beispiel

Ein CAN-BUS-OFF-Fehler tritt auf.

Das Betriebssystem setzt das CAN-BUS-OFF-ERROR-Bit.

Das Anwendungsprogramm erkennt diesen Zustand durch Abfrage der betreffenden Bits.

Bei Bedarf kann das ERROR-Bit gesetzt werden:
Als Folge blinkt die Betriebsanzeige-LED rot. Damit werden alle Ausgänge abgeschaltet.

Der Fehler wird behoben, indem CAN über den Funktionsaufruf CAN_RESTART neu gestartet wird. Dabei wird das CAN-BUS-OFF-ERROR-Bit automatisch gelöscht.

Anschließend muß ggf. noch das ERROR-Bit per Anwenderprogramm gelöscht werden. Die LED blinkt wieder grün.

5. CAN in der CS0015

5.1. Technische Spezifikation

Bustyp:	FULL-CAN
Physikalische Schicht:	ISO/DIS 11898
Baudrate:	10 kBit/s ... 1 MBit/s
Protokoll:	CANopen freies Protokoll

2048 Datenobjekte im System (CAN-Spezifikation 2.0 B)

Identifier-Verwendung

1 ... 2048 Identifier frei verfügbar für den Datentransfer

Folgende Identifier davon sind reserviert:

220 ... 221	reserviert für das Display tdm R 360
223 ... 252	Geräte-Identifier der Teilnehmer
254	Geräte-Identifier eines noch nicht konfigurierten Moduls
255	Identifier des Download-Systems (z.B. PC)

System-Konfiguration

Die CS0015 wird mit dem Geräte-Identifier 254 (ID 32) als Teilnehmer 0 ausgeliefert. Das Download-System benutzt diesen Identifier für die erste Kommunikation mit einem nicht konfigurierten Modul.

Es darf jeweils nur **ein** nicht konfiguriertes Modul mit dem Netz verbunden werden. Nachdem die neue Teilnehmernummer 1 ... 30 (entspricht dem Node-Identifier 1 ... 30) über die Programmiersoftware zugewiesen wurde, kann ein Download bzw. ein Debugging stattfinden und ein weiteres Gerät ins System eingebunden werden (siehe auch 5.5.).

5.2. CAN-Datenaustausch

Wenn man über den Datenaustausch über CAN spricht, wird das in der ISO 11898 international genormte CAN-Protokoll der Verbindungsschicht (Ebene 2) des siebenschichtigen ISO/OSI-Referenzmodell zu Grunde gelegt.

Jeder BUS-Teilnehmer kann Nachrichten senden (Multimaster-Fähigkeit). Der Datenaustausch arbeitet ähnlich dem Rundfunk. Daten werden ohne Absender bzw. Adresse auf den Bus gegeben. Die Daten sind lediglich durch ihren Identifier gekennzeichnet. Es ist Aufgabe jedes Teilnehmers, die gesendeten Daten zu empfangen und an Hand des Identifiers zu prüfen, ob die Daten für diesen Teilnehmer relevant sind. Dieser Vorgang wird vom CAN-Controller in Verbindung mit dem Betriebssystem automatisch durchgeführt. Damit nicht jede CAN-Meldung bearbeitet werden muß, kann durch Angabe

einer sogenannten Akzeptanzmaske (CAN_ACCEPTANCE) nur ein bestimmter Teil der Bus-Daten zum CAN-Controller durchgelassen werden. Der Einsatz dieser Spezialfunktion ist nur dann sinnvoll, wenn Daten für bestimmte Busteilnehmer unrelevant sind und eine Zeitoptimierung in einem Steuerungsmodul bei der CAN-Verarbeitung unbedingt benötigt wird. Um diese Funktion anzuwenden sind Hardware-Kenntnisse des CAN-Controllers notwendig. Diese können der Herstellerdokumentation entnommen werden oder beim technischen Support der ifm electronic gmbh angefragt werden.

Für den normalen CAN-Datenaustausch muß der Programmierer lediglich bei der Softwareerstellung mit den Funktionen CAN_RECEIVE (CAN-Daten empfangen) und CAN_TRANSMIT (CAN-Daten senden) die Datenobjekte mit ihren Identifiern dem System bekannt machen. Über diese Funktionen werden die RAM-Adresse der Arbeitsdaten, der Datentyp und der gewählte Identifier zu einem Datenobjekt verknüpft. Diese nehmen dann am Datenaustausch über den CAN-Bus teil. Die Sende- und Empfangsobjekte können aus allen gültigen IEC-Datentypen (z.B. BOOL, WORD, INT, ARRAY) definiert werden.

Die CAN-Message besteht aus einem Identifier und maximal 8 Datenbytes. Der Identifier ist zwischen 1 und 2048 frei wählbar. Er repräsentiert, wie schon erwähnt nicht das Absender- oder Empfängermodul, sondern kennzeichnet die Message (Meldung). Um Daten zu übertragen, ist es notwendig, daß im Sendemodul ein Sendeobjekt, und in **mindestens einem** anderen Modul ein Empfangs-Objekt deklariert ist. Beide Deklarationen müssen dem gleichen Identifier zugeordnet sein.

Daten empfangen

Grundsätzlich werden die empfangenen Datenobjekte automatisch (d.h. ohne Einfluß durch den Anwender) in einem Zwischenspeicher abgelegt.

Pro Identifier steht ein solcher Zwischenspeicher (Warteschlange) zur Verfügung. Dieser wird in Abhängigkeit von der Anwendersoftware nach dem FIFO-Prinzip (First In, First Out) über die Funktion CAN_RECEIVE entleert. In der Warteschlage werden **maximal 30** Datensendungen zwischengespeichert. Weitere Datensendungen können erst dann gespeichert werden, wenn der Puffer entleert wurde. Ein Eingang einer neuen CAN-Message führt zum Überlauf der Warteschlange. Dieses wird dem Anwender durch das Bit OVERFLOW angezeigt.

Daten senden

Durch den Aufruf der Funktion CAN_TRANSMIT übergibt das Anwenderprogramm genau eine CAN-Message an den CAN-Controller. Als Rückgabe erhält man die Information ob die Message erfolgreich an den CAN-Controller übergeben wurde. Dieser führt dann die eigendliche Übergabe der Daten auf den CAN-Bus selbständig aus.

Der Sendeauftrag wird abgewiesen, wenn der Controller nicht bereit ist weil er aktuell ein Datenobjekt überträgt. Der Sendeauftrag muß dann durch das Anwenderprogramm wiederholt werden. Der Anwender bekommt diese Information durch ein Bit angezeigt.

5.3. CAN Fehler und Fehlerbehandlung

Die im folgenden beschriebenen Fehlermechanismen werden von dem in der Steuerung integrierten CAN-Controller automatisch abgearbeitet. Der Anwender hat darauf keinen Einfuß. Er muß/sollte lediglich auf gemeldete Fehler in der Anwendersoftware reagieren.

Ziel der CAN-Fehler-Mechanismen ist es:

- Sicherstellung einheitlicher Datenobjekte im gesamten CAN-Netz.
- Dauerhafte Funktionsfähigkeit des Netzes auch im Falle eines defekten CAN-Teilnehmers.
- Unterscheidung zwischen zeitweiliger und dauerhafter Störung eines CAN-Teilnehmers.
- Lokalisierung und Selbstabschaltung eines defekten Teilnehmers in 2 Stufen (Error-passiv, Bus-off). Diese ermöglicht einem zeitweilig gestörtem Teilnehmer eine „Erholungspause“.

Um dem interessierten Anwender einen Überblick über das Verhalten des CAN-Controllers im Fehlerfall zu geben, soll an dieser Stelle vereinfacht die Fehlerbehandlung beschrieben werden. Nach der Fehlererkennung werden die Informationen automatisch aufbereitet und stehen in der Anwendersoftware dem Programmierer als CAN-Fehler-Bits zur Verfügung.

Fehlertelegramm

Erkennt ein Busteilnehmer eine Fehlerbedingung, so sendet er sofort ein Fehlerflag und veranlaßt damit den Abbruch der Übertragung bzw. das Verwerfen der von anderen Teilnehmern schon empfangenen fehlerfreien Nachrichten. Dadurch wird sichergestellt, daß allen Teilnehmern fehlerfreie und einheitliche Daten zur Verfügung stehen. Da das Fehlerflag unmittelbar überragen wird, kann im Gegensatz zu anderen Feldbussystemen (diese warten eine festgelegte Quittierungs-zeit ab) sofort mit der Wiederholung der gestörten Nachricht durch den Absender begonnen werden. Dies ist eines der wichtigsten Merkmale von CAN.

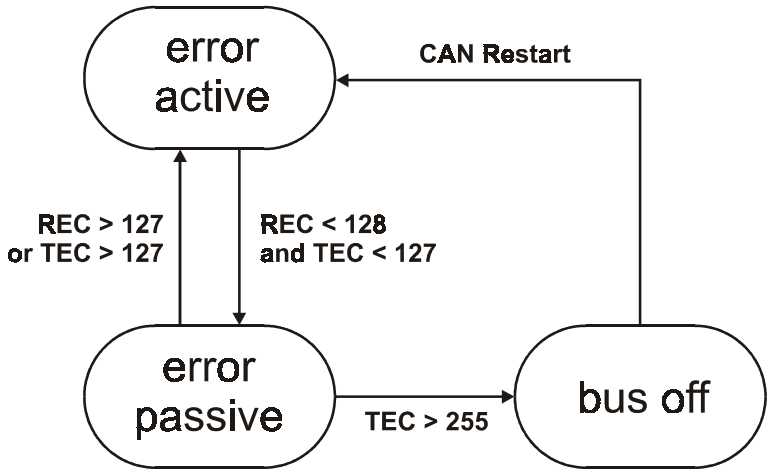
Eine der grundsätzlichen Problematiken der seriellen Datenübertragung ist, daß ein dauerhaft gestörter oder defekter Busteilnehmer das gesamte System blockieren kann. Gerade die Fehlerbehandlung bei CAN würde diese Gefahr beinhalten. Um diesen Fall auszuschließen, ist ein Mechanismus erforderlich, welcher den Defekt eines Teilnehmers erkennt und diesen gegebenenfalls vom Bus abschaltet.

Fehlerzähler

Dazu sind im CAN-Controller ein Sende- und ein Empfangsfehlerzähler enthalten. Diese werden bei jedem fehlerhaften Sende- oder Empfangsvorgangs heraufgezählt (inkrementiert). War eine Übertragung fehlerfrei, werden diese Zähler wieder heruntergezählt (dekrementiert).

Die Fehlerzähler werden jedoch im Fehlerfall stärker inkrementiert, als sie im Erfolgsfalle dekrementiert werden. Über eine bestimmte Zeitspanne kann dies zu einem merklichen Anstieg der Zählerstände führen, selbst wenn die Anzahl der ungestörten Nachrichten größer ist, als die Anzahl der gestörten Nachrichten. Längere fehlerfreie Zeitspannen bauen die Zählerstände jedoch wieder ab. Die Zählerstände sind somit ein Maß für die relative Häufigkeit von Störungen.

Werden Fehler durch einen Teilnehmer als erster erkannt (selbstverschuldete Fehler), wird bei diesem der Fehler stärker ´bestraft´ als bei den anderen Busteilnehmern. Dazu wird der Zähler um eine höheren Betrag inkrementiert. Übersteigt nun der Zählerstand einen bestimmten Wert, kann davon ausgegangen werden, daß dieser Teilnehmer defekt ist. Damit dieser Teilnehmer den folgenden Busverkehr nicht weiter durch **aktive Fehlermeldungen** (error active) stört, wird er **fehlerpassiv** (error passiv).



REC = Receive error counter
TEC = Transmit error counter

Teilnehmer, fehleraktiv

Ein fehleraktiver Teilnehmer nimmt voll am Busverkehr teil und darf erkannte Fehler durch senden des aktiven Fehlerflags signalisieren. Wie schon beschrieben wird dadurch die übertragene Nachricht zerstört.

Teilnehmer, fehlerpassiv

Ein fehlerpassiver Teilnehmer ist auch noch voll kommunikationsfähig. Er darf allerdings einen von ihm erkannten Fehler nur durch ein den Busverkehr nicht störendes passives Fehlerflag kenntlich machen. Ein fehlerpassiver Teilnehmer wird beim Unterschreiten eines festgelegten Zählerwertes wieder fehleraktiv.

Teilnehmer, bus-off

Wird der Fehlerzählerwert weiter inkrementiert, wird nach Überschreiten eines Maximalzählerwertes der Teilnehmer vom Bus abgeschaltet (bus-off).

Der Bus-off-Zustand kann nur durch einen Reset (CAN_RESTART) des CAN-Controllers behoben werden.

Deshalb sollte die Funktion CAN_ERRORHANDLER genutzt werden. Durch diese werden alle CAN-Fehlerzustände registriert und der CAN-Controller gegebenenfalls zurückgesetzt. Gleichzeitig steht dem Anwenderprogramm ein Fehlerzähler zur Verfügung. Dieser könnte z.B. dazu genutzt werden um in Abhängigkeit vom Zählerstand weitere Maßnahmen zu ergreifen (z.B. Fehler-LED).

Eine detailierte Fehleranalyse ist aber nur über eine genaue Auswertung der Fehlerbits möglich.

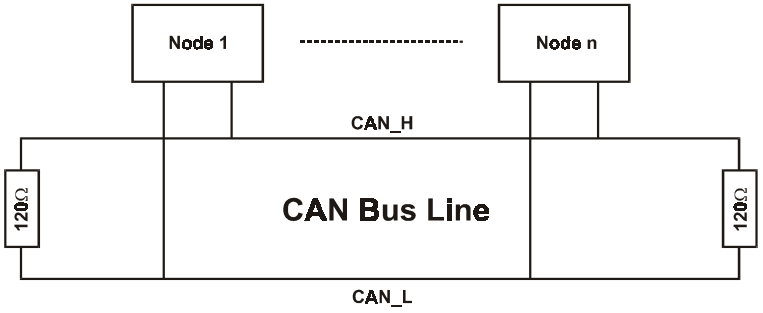
5.4. Die physikalische Anbindung des CAN

Die in den Kapiteln 5.2. und 5.3. beschriebenen Mechanismen der Datenübertragung und der Fehlerbehandlung sind direkt im CAN-Controller implementiert. Die physikalische Verbindung der einzelnen CAN-Teilnehmer wird in der ISO 11898 in der Schicht 1 beschrieben.

Der Netzaufbau

Die Norm ISO 11898 setzt einen Aufbau des CAN-Netzes mit einer Linienstruktur voraus.

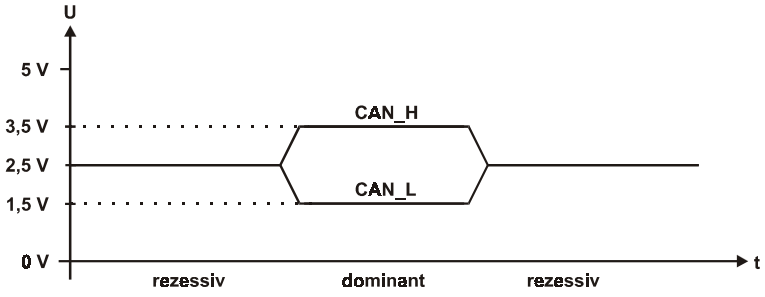
Zusätzlich muß die Linie an ihren beiden Enden jeweils mit einen Abschlußwiderstand von der Größe 120 Ω versehen werden. Die Geräte der ifm electronic, die mit einem CAN-Interface ausgestattet sind, haben grundsätzlich keine Abschlußwiderstände.



Idealerweise sollte zu den Busteilnehmern (Node 1 ... Node n) keine Stichleitung führen, da in Abhängigkeit von der Gesamtleitungslänge und den zeitlichen Abläufen auf dem Bus Reflektionen auftreten. Damit diese nicht zu Systemfehlern führern, sollten die Stichleitungen zu einem Busteilnehmer (z.B. einem E/A-Modul) eine gewisse Länge nicht überschreiten. Stichleitungen mit einer Länge von 2 m werden als unkritisch angesehen. Die Summe aller Stichleitungen im Gesamtsystem sollte 30 m nicht übersteigen. In besonderen Fällen müssen die Leitungslängen der Linie und der Stiche genau berechnet werden.

Der Buspegel

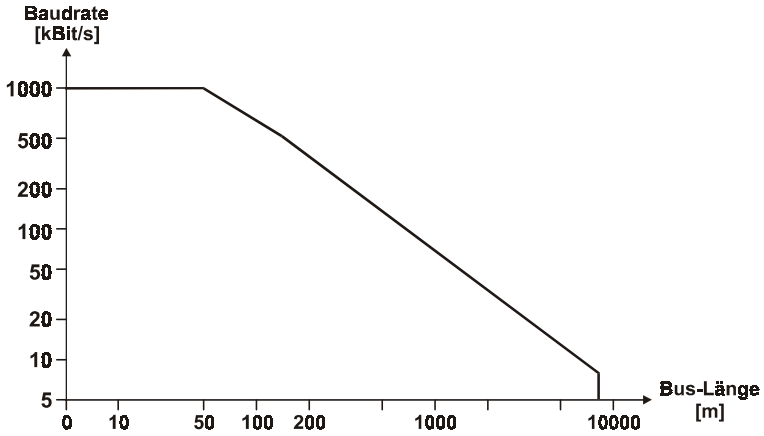
Der CAN-Bus befindet sich im inaktiven (rezessiven) Zustand, wenn die Ausgangstransistorpaare in allen Busteilnehmern ausgeschaltet sind. Wird mindestens ein Transistorpaar eingeschaltet, wird ein Bit auf den Bus gegeben. Dieser wird dadurch aktiv (dominant). Dadurch fließt ein Strom durch die Abschlußwiderstände und erzeugt eine Differenzspannung zwischen den beiden Busleitungen. Die rezessiven und dominanten Zustände werden in den Busknoten in entsprechende Spannungen umgewandelt und von den Empfängerschaltkreisen erkannt.



Durch diese differentielle Übertragung mit gemeinsamen Rückleiter wird die Übertragungssicherheit entscheidend verbessert. Störspannungen, die von außen auf das System einwirken, oder Massepotentialverschiebungen beeinflussen beide Signalleitungen mit gleichen Störgrößen. Dadurch fallen diese bei der Differenzbildung wieder heraus.

Die Busleitungslänge

Die Länge der Busleitung ist abhängig von der Beschaffenheit der Busverbindung (Kabel, Steckverbinder), dem Leitungswiderstand und der benötigten Übertragungsrate (Baudrate). Zusätzlich muß noch wie oben beschrieben die Länge der Stichleitungen bei der Netzauslegung berücksichtigt werden. Vereinfachend kann man von folgender Abhängigkeit zwischen Buslänge und Baudrate ausgehen.



Die Leitungsquerschnitte

Für die Auslegung des CAN-Netzes ist auch der Leitungsquerschnitt der eingesetzen Busleitung zu beachten. Die nachfolgende Tabelle beschreibt bezogen auf eine Übertragungsrate von 1 Mbit/s und einer maximalen Leitungslänge von 40 m (Leitungswiderstand r = 70 mΩ/m) die Abhängigkeit der Leitungsquerschnitte von der Anzahl der Busteilnehmer.

Leitungslänge	32 Busknoten	64 Busknoten	100 Busknoten
100 m	0,25 mm^2	0,25 mm^2	0,25 mm^2
250 m	0,34 mm^2	0,50 mm^2	0,50 mm^2
500 m	0,75 mm^2	0,75 mm^2	1,00 mm^2

In Abhängigkeit von den EMV-Anforderungen können die Busleitungen parallel, als Twisted-Pair und/oder abgeschirmt ausgeführt werden.

5.5. Allgemeine Hinweise zur Nutzung von CAN

Wird im Zusammenhang mit der Steuerung CS0015 CAN bzw. CANopen benutzt sind einige Punkte zu beachten. Diese betreffen sowohl die physikalische Auslegung des CAN-Netzwerkes als auch den richtigen Umgang mit der Software.

Physikalische Netzauslegung

Zusammenfassend gilt bei der Auslegung des CAN-Netzes gilt:

- Die Datenübertragungsrate nur so hoch wählen wie sie benötigt wird. Eine niedrige Übertragungsrate erhöht die Betriebssicherheit.
- Leitungslänge entsprechend der Datenübertragungsrate beachten. Bei der CS0015 beträgt sie bei 125 kBaud typisch 400 m.
- Busleitung als Linie verlegen und Stichleitungen vermeiden. Klemmstellen sauber und fest ausführen um unnötige Übergangswiderstände zu vermeiden. Wenn notwendig die Leitungen als Twisted-Pair und/oder abgeschirmt verlegen.
- Die Busleitung an beiden Enden mit einem Abschlußwiderstand von 120 Ω versehen.
- Je höher die Anzahl der vernetzten Teilnehmer ist, um so sogfältiger muß das Netzwerk dimensioniert werden (Leitungsausführung, Leitungslänge usw.).

Software für CAN und CANopen

Grundsätzlich kann die CS0015 durch Nutzung der Funktionen CAN_TRANSMIT und CAN_RECEIVE direkt an der CAN-Kommunikation teilnehmen (Schicht 2). In der Betriebsart CANopen, werden dem Programmierer die festgelegten Dienste zur Verfügung gestellt.
Folgende Punkte sind zu beachten:

- In der Betriebsart CAN-Direkt auf Schicht 2 ist der Programmierer für alle Dienste selbst verantwortlich. Die Steuerung befindet sich in diesem Zustand, nach einem Programm-Download oder einem Reset-Kommando durch das Programmiersystem.
- Bei CAN-Direkt wird die zyklische Einbindung des Funktionsbausteines CAN_ERRORHANDLER empfohlen. Anderfalls, muß durch das Anwenderprogramm bei BUS_OFF ein CAN_RESTART durchgeführt werden.
- Die Steuerung ist nach einem Programm-Download oder einem Reset-Kommando durch das Programmiersystem zunachächst noch kein CANopen-Gerät.

Zum Umschalten in die Betriebsart CANOpen, muß der Merker CAN_OPEN zu Beginn des Programms gesetzt werden. Die CS0015 läuft dann als CANopen-Slave.

Wird ein CS0015-Slave über die Programmiersoftware gestoppt, wird ein nachfolgendes Node-Start Kommando des CANopen-Masters ignoriert. Ein Stop-Kommando des Masters (NMM_SET_PREPARED) wird aber immer ausgeführt.

- Bei fehlender Guarding-Antwort des CS0015-Slaves sendet der Master fortlaufend Node-Resets. Dadurch kann es zu Problemen beim Einloggen des Programmiersystems über die CAN-Schnittstelle kommen. In diesem Fall muß der Master ausgeschaltet werden.
- Soll die CS0015 auch als CANopen-Master arbeiten, muß dieser mit der Funktion NMM_SET_NMT_MASTER initialisiert werden.

Wird die Steuerung gestoppt (durch PC), behält sie die CANopen-Funktionalität, die Masterfunktionalität ist jedoch unterbrochen (z.B. keine SYNC-Meldung).

- Allen Teilnehmern im CAN-Netzwerk muß ein eindeutiger Modul-ID zugewiesen werden.

Geräte-IDs in der CS0015

Um mit den Teilnehmer im CAN-Netzwerk zu kommunizieren, muß jeder einen eindeutigen Geräte-Identifier besitzen. Dabei ist es unerheblich ob die Steuerung als NMT-Master als CANopen-Slave oder für die direkte CAN-Kommunikation eingesetzt wird. Außerdem ist zu beachten, daß die Geräte-Identifier sich nicht mit den IDs der E/A-Module überschneiden. Die CS0015 wird mit dem Default-ID 32 (unter CANopen) ausgeliefert. In der Programmiersoftware ecolog 100*plus* wird der Node-ID 32 als Modul-ID Nr. 0 bezeichnet.

Modul-ID ecolog 100*plus*	**Node-ID CANopen**	**Device-ID Debugger**
(default) 0	(n. konf.) 32	0xFE
1	1	0xDF
2	2	0xE0
3	3	0xE1
:	:	:
29	29	0xFB
30	30	0cFC

Die Zuweisung des Geräte-IDs kann online über ecolog 100*plus* erfolgen.

5.6. Beschreibung der CAN Funktionsbausteine

Im folgenden werden die CAN-Funktionsbausteine zur Nutzung im Anwenderprogramm beschrieben.

Um die volle Leistungsfähigkeit von CAN zu nutzen ist es unbedingt erforderlich, daß sich der Programmierer vor Beginn seiner Arbeit ein genaues Buskonzept aufbaut. Dabei muß die Anzahl der Datenobjekte mit ihren Identifieren genauso festgelegt werden, wie eine Reaktion auf mögliche CAN-Fehler. Außerdem muß beachtet werden in welcher Häufigkeit Daten übertragen werden müssen. Dem entsprechend oft müssen dann die Funktionen CAN_TRANSMIT und CAN_RECEICE aufgerufen werden. Der Programmierer muß dabei zusätzlich überwachen, ob seine Sendeaufträge erfolgreich an CAN_TRANSMIT (Bit RESULT) übergeben wurden, bzw. dafür sorgen, daß die empfangenen Daten mit CAN_RECEIVE aus dem Datenpuffer der Warteschlage ausgelesen und entsprechend sofort im übrigen Programm verarbeitet werden.

Damit eine Kommunikationsverbindung aufgebaut werden kann, muß zuvor bei allen Telnehmern des CAN-Netzwerkes die gleiche Übertragungsrate (Baudrate) eingestellt werden. Bei der CS0015 wird diese mit der Funktion CAN_BAUDRATE vorgenommen.

Beispielprogramm

Ein Beispielprogramm in Funktionsplan (FUP) ist auf der Programmdiskette ecolog 100^{plus} gespeichert. In diesem werden über die Identifier 1 und 2 Datenobjekte mit einem weiteren CAN-Teilnehmer ausgetauscht. Dazu muß im anderen Teilnehmer zum Sende-Identifier ein Empfangs-Identifier (bzw. umgekehrt) existieren.

Die Funktion CAN_ACCEPTANCE wird an dieser Stelle nicht weiter dokumentiert, da die Anwendung genaue Hardware-Kenntnisse des CAN-Controllers erfordert. Anwender die diese speziellen Möglichkeiten benötigen, werden an den technischen Support verwiesen.

Funktion **CAN_BAUDRATE**

Library **CSxxxx.LIB**

Funktionssymbol

Zweck Stellt die Übertragungsrate für den Busteilnehmer ein.

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet
BAUDRATE	WORD	Wert der einzustellen Baudrate in kBit/s (10, 20, 50, 100, 125, 250, 500, 1000)

Funktionsausgänge, keine

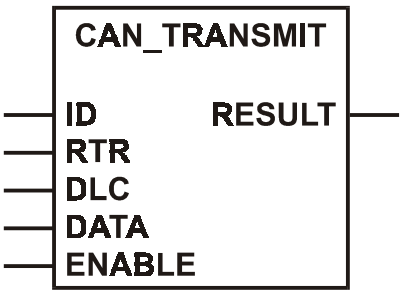
Beschreibung Mit der Funktion CAN_BAUDRATE wird für das Steuerungsmodul die Übertragungsrate eingestellt. Dazu wird am Funktioneingang BAUDRATE der entsprechende Wert in kBit/s angegeben. Nach Ausführung der Funktion, wird dieser neue Wert im Gerät gespeichert, und steht auch nach einem Spannungsausfall wieder zur Verfügung. Die Default-Baudrate bei Auslieferung der Module beträgt 125 kBit/s.

Die Funktion sollte nur einmal bei der Initialisierung im ersten Programmzyklus ausgeführt werden. Danach wird sie über den Eingang ENABLE gesperrt.
Die Baudrate wird nach Aufruf der Funktion sofort gültig.

Funktion **CAN_TRANSMIT**

Library **CSxxxx.LIB**

Funktionssymbol



Zweck Übergibt ein CAN-Datenobjekt (Message) an den CAN-Controller zur Übertragung

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
ID	WORD	Enthält die Nummer des Datenobjekt-Identifierer 0 ... 2048.
RTR	BYTE	wird nicht genutzt, daher Wert 0
DLC	BYTE	Anzahl der zu übertragenden Bytes aus dem Array DATA. (Zulässige Werte 0 ... 8)
DATA	ARRAY	Das Array enthält maximal 8 Datenbytes
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet

Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BOOL	TRUE: Die Funktion hat den Sendeauftrag angenommen

Beschreibung

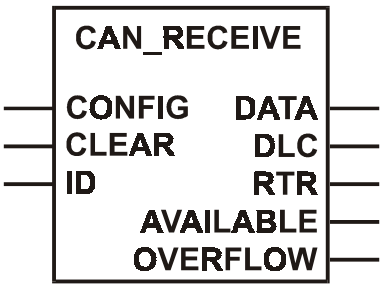
CAN_TRANSMIT wird für jedes Datenobjekt im Programmzyklus aufgerufen. Bei langen Programmzyklen auch mehrfach. Der Programmierer muß durch Auswertung des Bits RESULT dafür Sorge tragen, daß sein Sendeauftrag auch angenommen wurde. Vereinfacht gilt bei 125 kBit/s, daß pro 1 ms ein Sendeauftrag ausgeführt werden kann.

Über den Bit-Eingang ENABLE kann die Ausführung der Funktion zeitweilig gesperrt werden. Damit kann z.B. eine Busüberlastung verhindert werden. Außerdem können so mehrere Datenobjekte quasi gleichzeitig verschickt werden, wenn jedem Datenobjekt ein Merkerflag zugeordnet wird und mit diesem die Ausführung der Funktion über den ENABLE-Eingang gesteuert wird.

Funktion **CAN_RECEIVE**

Library **CSxxxx.LIB**

Funktionssymbol



Zweck Konfiguriert ein Datenempfangsobjekt und liest den Empfangspuffer des Datenobjektes aus.

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
CONFIG	BOOL	Bit muß einmalig bei der Konfiguration des Datenobjektes gesetzt (TRUE) werden. Im weiteren Programmablauf ist das Bit FALSE.
CLEAR	BOOL	löscht den Datenpuffer (Warteschlange)
ID	WORD	Enthält die Nummer des Datenobjekt-Identifierer 0 ... 2048.

Funktionsausgänge

Name	Datentyp	Beschreibung
DATA	ARRAY	Das Array enthält maximal 8 Datenbytes
DLC	BYTE	Anzahl der übertragenen Bytes im Array DATA. Mögliche Werte 0 ... 8.
RTR	BYTE	wird nicht genutzt
AVAILABLE	BYTE	Anzahl der eingegangenen Meldungen
OVERFLOW	BOOL	TRUE: Überlauf des Datenpuffers. Datenverlust! FALSE: Puffer noch nicht gefüllt

Beschreibung

CAN_RECEIVE muß für jedes Datenobjekt in der Initialisierungsphase einmalig aufgerufen werden um dem CAN-Controller die Identifier der Datenobjekte bekannt zu machen.

Im weiteren Programmzyklus wird CAN_RECEIVE zum Auslesen des jeweiligen Empfangspuffers aufgerufen. Bei langen Programmzyklen auch mehrfach. Der Programmierer muß durch Auswertung des Bytes AVAILABLE dafür Sorge tragen, daß neu eingegangene Datenobjekte aus dem Puffer abgerufen und weiterverarbeitet werden. Jeder Aufruf der Funktion dekrementiert das Byte AVAILABLE um 1. Ist der Wert von AVAILABLE gleich 0, sind keine Daten im Puffer.

Durch Auswertung des Bits OVERFLOW kann ein Überlauf des Datenpuffers erkannt werden. Wird das Bit OVERFLOW gesetzt, ist mindestens 1 Datenobjekt **verloren** gegangen.

Funktion

CAN_RESTART

Library

CSxxxx.LIB

Funktionssymbol

Zweck

Neustart des CAN-Teilnehmers nach „massiven“ Übertragungsfehlern (bus-off-Zustand)

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet

Funktionsausgänge, keine

Beschreibung

CAN ermöglicht eine Unterscheidung zwischen zeitweiligen und dauerhaften Störungen eines Busteilnehmers. Wie unter Kapitel 5.3. beschrieben können drei Funktionszustände vorliegen.

Ist ein Teilnehmer **error-activ** handelt es sich um den Normalzustand.

Tritt eine bestimmt Anzahl von Übertragungsfehlern auf, so wird der Teilnehmer **error-passiv**. Verringert sich die Fehlerhäufigkeit wird der Teilnehmer wieder error-activ.

Ist ein Teilnehmer schon error-passiv und es treten weiterhin Übertragungsfehler auf, wird er vom Bus abgeschaltet (**bus-off**) und das Fehlerflag CAN_BUS_OFF_ERROR wird gesetzt. Die Rückkehr an den Bus ist nur mit der Funktion CAN_RESTART möglich. Das Fehlerflag wird nach erfolgreicher Rückkehr zurückgesetzt.

Der Eingang ENABLE unterdrückt die Ausführung der Funktion.

Funktion **CAN_ERRORHANDLER**

Library **CSxxxx.LIB**

Funktionssymbol

Zweck Minimale Fehlerroutine zur Überwachung von CAN

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
RESET	BOOL	Löscht den Fehlerzähler

Funktionsausgänge

Name	Datentyp	Beschreibung
ERROR-COUNT	WORD	Fehlerzähler, enthält die Anzahl der aufgetretenen Fehler

Beschreibung

CAN_ERRORCOUNT wertet alle möglichen CAN-Fehler aus und summiert die Anzahl der Fehler im Zähler ERRORCOUNT auf. Im Falle eines bus-off-Fehlers versucht die Funktion die Rückkehr des Teilnehmers an den Bus. Dazu ist die Funktion CAN_RESTART integriert.

Der Programmierer hat nun die Aufgabe durch Auswertung des Fehlerzählers und der vom System zur Verfügung gestellten Fehlerbits die genaue Fehlerursache zu lokalisieren. Über den Funktionseingang RESET kann anschließend der Zähler wieder auf 0 gesetzt werden.

In jeder Applikationssoftware, in der die CAN-Kommunikation genutzt wird (auch bei der Kommunikation mit einem CAN-Display), sollte mindestens diese Funktion eingesetzt und zyklisch abgearbeitet werden.

5.7. CANopen in der CS0015

Die am Anfang des Kapitel 5. beschriebenen Schichten 1 und 2 von CAN regeln die physikalische Anbindung und die Übertragung der Daten zwischen den Busteilnehmern. Beim praktischen Einsatz von CAN in einer Applikation bedeutet dieses, daß der Programmierer das Datenprotokoll für die spezielle Anwendung selbst festlegen muß.

Um eine einheitliche Protokollschicht für die Vernetzung der unterschiedlichen Teilnehmer zu bekommen die die Bedeutung der übertragenen Daten beschreibt, wurde die Anwendungsschicht CAL (CAN Application Layer) als Schicht 7 festgelegt. CANopen wiederum baut auf CAL auf und definiert, welche Daten mit welchen CAL-Diensten übertragen werden sollen. Außerdem ist festgelegt was die Daten für den jeweiligen Gerätetyp (E/A-Modul, Antriebe, Drehgeber usw.) bedeuten. Diese Festlegungen ermöglichen es dem Applikationsprogrammierer ohne großen Protokollaufwand herstellerübergreifend auf alle CANopen-fähigen Komponenten zuzugreifen. CANopen-Teilnehmer, die der gleichen Gerätefamilie angehören haben ihre Daten auf die gleiche Art und Weise organisiert. Die Eigenschaften dieser Geräteklassen werden in den „Device Profiles“ (DS-40x) zusammengefaßt.

Trotz dieser Festlegung bleibt die Grundstruktur von CAN erhalten, die es jedem Busteilnehmer ermöglicht Nachrichten (Daten) auf das Netzwerk zu geben. Lediglich der Netzwerkmaster (NMT-Master) ist einmalig vorhanden und dient hauptsächlich zum hochfahren und zur Überwachung des Systems.

Die im folgenden Text beschriebenen Mechanismen sollen einen groben Überblick über die Funktion von CANopen geben. Um das CANopen-Protokoll in seiner gesamten Leistungsfähigkeit optimal für die jeweilige Applikation zu nutzen, wird auf weiterführende Literatur verwiesen (Informationen über CAN in Automation e.V. (CiA), Erlangen).

Allgemeines zu CANopen

Grundsätzlich besitzt jeder CANopen Knoten ein Objektverzeichnis, das über „Service Data Objects“ (SDOs) angesprochen werden kann. Zusätzlich stehen mindestens zwei „Process Data Objects“ (PDOs) zum Senden und Empfangen von Prozeßdaten, ein „Nodeguarding Object“ zur Realisierung einer Netzwerküberwachung, sowie ein „Emergency Object“ zur Anzeige von Fehlerzuständen zur Verfügung.

Die objektorientierten Identifier (11 Bit) werden unter CANopen als „CAN Object Ids“ (COB-IDs) bezeichnet. Über die 4 höchstwertigen Bits (MSBs) werden diese in 16 Gruppen eingeteilt. Die verbleibenden 7 Bits werden zur Unterscheidung von 127 CANopen Knoten genutzt. Hierdurch ist eine eindeutige Zuordnung der einzelnen Objekttypen zu den Knoten gegeben. Bei dieser Festlegung handelt es sich um eine Default-Zuordnung.

Diese wird im „predefined connection set“ festgelegt. Ob man von dieser Vorgabe abweicht hängt von der jeweiligen Applikation ab. Um eine hohe Flexibilität bei der Auswahl von CANopen-Geräten unterschiedlicher Hersteller zu haben, sollte man sich diesen Schritt genau überlegen.

Objekt	Code (Binär)	COB-IDs (Dezimal)	Default-Funktion
NMT	0000 0000000	0	Netzwerk-Manag.
SYNC	0001 0000000	128	Synchronisation
EMCY	0001 xxxxxxx	129 - 255	Fehlerzustände
TIME STAMP	0010 0000000	256	Netzwerk-Zeit
PDO1(tx)	0011 xxxxxxx	385 - 511	Synchrones PDO
PDO1(rx)	0100 xxxxxxx	513 - 639	Synchrones PDO
PDO2(tx)	0101 xxxxxxx	641 - 767	Asynchrones PDO
PDO2(rx)	0110 xxxxxxx	769 - 895	Asynchrones PDO
SDO(tx)	1011 xxxxxxx	1409 - 1535	Master->Slave SDO
SDO(rx)	1100 xxxxxxx	1537 - 1663	Slave->Master SDO
Nodeguarding	1110 xxxxxxx	1793 - 1919	Node/Life Guarding

Das Objektverzeichnis

Alle Knotenparameter werden im Objektverzeichnis des jeweiligen CANopen-Knoten gespeichert. Zur eindeutigen Identifizierung wird ein Verzeichniseintrag durch einen Index (IDX, Länge 16 Bit) und einen Subindex (SUBIDX, Länge 8 Bit) gekennzeichnet. Je nach Art der Parameter werden diese in den einzelnen Indexbereichen abgelegt. Die Bedeutung der einzelnen Indizes für die Kommunikations- und die Standard-parameter sind in der CANopen-Norm für die einzelnen Gerätetypen festgelegt. Zusätzlich steht noch ein Bereich für herstellerspezifische Daten zur Verfügung. In diesem Bereich werden z.B. die Konfigurationsparameter für die E/A-Module der ifm electronic gmbh abgelegt.

Index (Hex)	Objekt
0000	nicht belegt
0001 - 009F	Daten Typen
00A0 - 0FFF	reserviert
1000 - 1FFF	Bereich für das Kommunikationsprofil
2000 - 5FFF	Bereich für herstellerspezifische Daten
6000 - 9FFF	Bereich für Standardgeräteparameter
A000 - FFFF	Bereich für allg. IEC1131-Netzvariablen

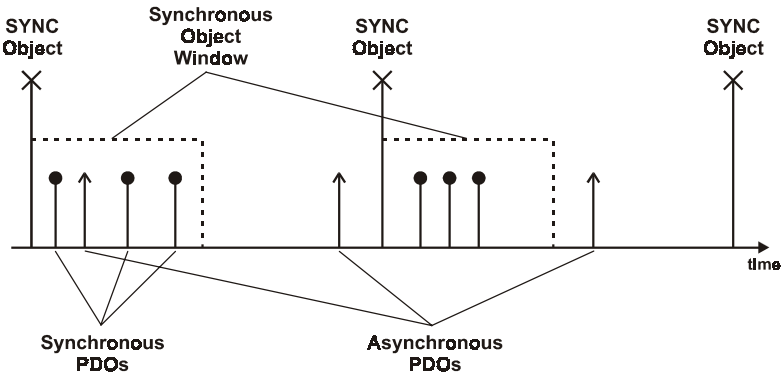
Service Data Object (SDO)

Ein lesender und schreibender Zugriff auf das Objektverzeichnis erfogt über die „Service Data Objects“ (SDOs).

Die SDOs werden für alle zeitunkritischen Daten in CANopen eingesetzt. Sie werden grundsätzlich nur von Punkt zu Punkt (Netzwerkmaster / Slave) übertragen. Hauptsächlich werden die SDOs zur Übertragung der Konfigurationsdaten des CAN-Teilnehmers in der Boot-Up Phase genutzt.

Process Data Object (PDO)

Die zeitkritischen Prozeßdaten werden mit Hilfe der „Process Data Objects“ (PDOs) übertragen. Die PDOs können beliebig zwischen den einzelnen Knoten ausgetauscht werden (PDO-Linking). Zusätzlich wird noch festgelegt, ob der Datenaustausch ereignisgesteuert (asynchron) oder synchronisiert erfolgen soll. Je nach der Art der zu übertragenden Daten kann die richtige Wahl der Übertragungsart zu einer erheblichen Entlastung des CAN-Bus führen. In den E/A-Modulen der ifm electronic gmbh werden in der Default-Einstellung analoge Eingangsdaten und alle Ausgangs-Daten synchronisiert und digitale Eingangsdaten ereignisgesteuert übertragen.



Nodeguarding Object

Zur Erkennung von Kommunikationfehlern im Netzwerk wird das Nodeguarding (Knotenüberwachung) eingesetzt. Damit wird jeder Busknoten periodisch durch den Netzwerk-Master über den festgelegten Nodeguarding COB-ID angesprochen. Bleibt eine Antwort in der festgelegten Guard Time (Überwachungszeit) aus, signalisiert der Master einen Fehler. Über die Life Time (Life Time Factor * Guard Time) kann zusätzlich festgelegt werden, nach wieviel erfolglosen Versuchen die Fehlermeldung erzeugt werden soll.

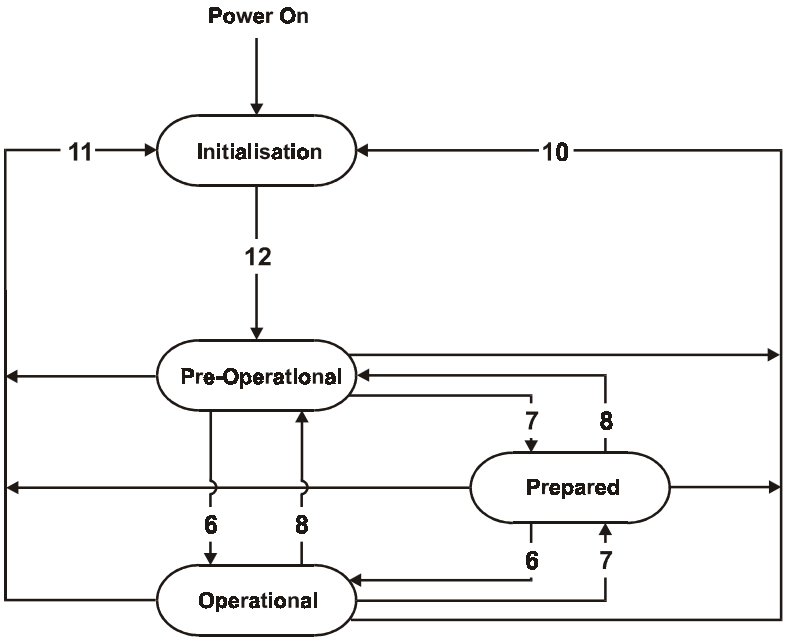
Emergency Object

Tritt ein interner Fehler in einem Busteilnehmer auf (z.B. falscher Konfigurationsparameter, Kurzschluß am Ausgang), wird eine EMCY Object erzeugt. Dieses EMCY Object hat ein standardisiertes Aussehen und wird einmalig gesendet wenn der Fehler auftritt und einmalig wenn der Fehlerzustand verschwunden ist.

Im Objektverzeichnis des Knotens werden diese Fehler noch zusätzlich gespeichert. Dazu steht das „Error Register“, „herstellerspezifische Status Register“ und die „error history“ zur Verfügung.

Boot-Up-Vorgang

Beim Boot-Up-Vorgang wird durch den Netzwerkmaster das Hochlaufen des Netzes ermöglicht. Dabei werden die wichtigsten Kommunikationsparameter dem Master bekannt gemacht und ggf. das Guarding aktiviert. Im Boot-Up sollten auch die Konfigurationsparameter übertragen werden. Dabei sollte sich der Knoten im Zustand „Pre-Operational“ befinden.



Status	Beschreibung
6	Start Remote Node indication
7	Stop Remote Node indication
8	Enter Pre-Operational State indication
10	Reset Node indication
11	Reset_Communication indication
12	Initialisation finished - enter Pre-Operational automatically

Damit der Boot-Up erfolgreich durchgeführt werden kann muß mindestens die Knotennummer und die Baudrate des CAN-Teilnehmers eingestellt sein. Die Baudrate des Masters muß mit dieser übereinstimmen. Diese Einstellung erfolgt über DIP-Schalter im Knoten oder eine zusätzliche Parametriersoftware. Da die Steuerung CS0015 zusätzlich die Möglichkeit bietet über die SDOs das Objektverzeichnis zu beschreiben, kann eine Einstellung auch über die Steuerung erfolgen.

Damit die CS0015 in CANopen Modus arbeitet, muß der Merker CAN_OPEN zu Beginn des Programms (in der Initialisierung) auf TRUE gesetzt werden.

5.8. Die CS0015 als CANopen-Slave

Die CS0015 kann unter CANopen auch als freiprogrammierbares Ein-/Ausgangsmodul eingesetzt werden. Er verhält sich dabei wie ein CANopen-Slave. Als CANopen-Slave wird die CS0015 als „Programmable Device“ entsprechend CiA DS 405 eingeordnet.

Um die CS0015 als CANopen-Slave zu nutzen muß das Systembit CAN_OPEN gesetzt werden.

Objektverzeichnis

Über das Objektverzeichnis kann auf die Geräteparameter zugegriffen werden. Wenn diese als Read/Write gekennzeichnet sind, ist eine Veränderung über SDO_WRITE und vom NMT_Master oder von einem externen Parametriersystem aus möglich.

Das Objektverzeichnis in der CS0015 hat drei wesentliche Bereiche. Ab Index 1000 Hex stehen die CANopen-Kommunikationsparameter.
Ab Index 2000 Hex werden die herstellerspezifischen Daten Baudrate und Knotennummer abgelegt.
Ab Index A000 Hex beginnt der Bereich für die allgemeinen IEC1131-Netzwerkvariablen. Diese werden über die PDOs übertragen. Die Identifier und die Transmissiontypes der PDOs werden in diesem Bereich eingetragen.
Die genaue Struktur des Objektverzeichnisses kann dem Anhang unter 1.6. entnommen werden

Baudrate und Knotennummer

Baudrate und Knotennummer werden im herstellerspezifischen Bereich des Objektverzeichnisses ab Index 20F0 / 20F1 Hex und 20F2 / 20F3 Hex eingetragen. Eine Änderung der Baudrate bzw. der Knotennummer kann über ein SDO vom Master, einen Funktionsaufruf oder das Programmiersystem erfolgen. Wird die Änderung über SDO_WRITE durchgeführt, müssen beide Einträge im Objektverzeichnis, den gleichen Inhalt haben. Die Änderung der Baudrate wird erst nach einem Reset gültig, die des Node-ID sofort.

Index	Sub-Index	Name	Default-Wert
20F0	0	Node-ID	32
20F1	0	Node-ID	32
20F2	0	Baudrate	3
20F3	0	Baudrate	3

Es dürfen nie zwei Teilnehmer mit der gleichen Knotennummer im Netz enthalten sein.

Zur Einstellung der Baudrate sind folgende Parameter zulässig:

Nummer	Baudrate (kBit/s)
0	1000
1	500
2	250
3	125
4	100
5	50
6	20
7	10

Remanente Daten

Über den herstellerspezifischen Bereich des Objektverzeichnis, besteht die Möglichkeit per SDO_WRITE einen maximal 256 Byte großen Datenblock in den CS0015-Slave zu übertragen. Diese Daten werden spannungsausfallsicher im Flash-Speicher abgelegt und können im Anwenderprogramm über die Retain-Adressen %MB0 ... %MB255 (%MW0 ... %MW127) weiter verarbeitet werden. Damit steht dieser Datenbereich als freidefinierter Parameter-Satz zur Verfügung.

PDOs

Im „predefined connection set“ gemäß CiA DS 401 werden die ersten zwei RX und TX PDOs in Abhängigkeit von der Knotennummer festgelegt. Mit diesen PDOs können jeweils 16 Datenbytes gesendet und empfangen werden. Werden weitere PDOs benötigt, müssen diese „von Hand“ über die Funktionen PDO_RX_CONFIG und PDO_TX_CONFIG im Anwenderprogramm definiert werden. Die Identifier müssen dann ab 380 Hex aufwärts vergeben werden. Wird nicht das „predefined connection set“ genutzt, müssen auch die COB-IDs für PDO 1 und PDO 2 ab 380 Hex beginnen. Insgesamt können 2 x 8 PDOs eingerichtet werden.

Da die COB-IDs für die PDOs nicht gespeichert werden (Ausnahme PDO 1 und 2 im „predefined connection set“), müssen diese für alle PDOs in der Initialisierungsroutine **einmalig** nach jeden Start der Steuerung erneut initialisiert werden. Grundsätzlich sind die PDO-IDs, die nicht im „predefined connection set“ enthalten sind, in allen Geräten identisch vorbelegt (RX-PDOs ab 380 Hex, TX-PDOs ab 388 Hex). Deshalb müssen diese, wenn mehrere CS0015-Slaves eingesetzt werden, neu mit PDO_TX/RX_CONFIG konfiguriert werden. Andernfalls würde es zu Konflikten bei den IDs kommen.

RX-PDO	ID	TX-PDO	ID
RX-PDO 1	pred. C.-Set	TX-PDO 1	pred. C.-Set
RX-PDO 2	pred. C.-Set	TX-PDO 2	pred. C.-Set
RX-PDO 3	382 Hex	TX-PDO 3	38A Hex
RX-PDO 4	383 Hex	TX-PDO 4	38B Hex
RX-PDO 5	384 Hex	TX-PDO 5	38C Hex
RX-PDO 6	385 Hex	TX-PDO 6	38D Hex
RX-PDO 7	386 Hex	TX-PDO	38E Hex
RX-PDO 8	387 Hex	TX-PDO 8	38F Hex

PDO-Mapping

Ein PDO-Mapping im herkömmlichen Sinn ist in der CS0015 nicht möglich, da dieses bei einer freiprogrammierbaren Steuerung nicht notwendig ist.
Über das Anwenderprogramm können die für das CANopen-Netzwerk relevanten Daten direkt in die entsprechen PDOs geschrieben bzw. aus diesen gelesen werden. Netzwerkvariablen im Bereich ab %MW 2000 für die Empfangsdaten und ab %MW 2032 für die Sendedaten, können sofort durch das Anwenderprogramm bearbeitet werden (siehe Anhang 1.5.). Damit stehen dem Anwender 8 x 4 Sendeworte (TX-PDOs) und 8 x 4 Empfangsworte (RX-PDOs) zur Verfügung.

Überwachung des PDO-Empfangs

CANopen unterstützt nicht die Erkennung ob neue Daten übertragen wurden. Wird diese Funktion benötigt, muß diese durch dem Programmierer erstellt werden. Mögliche Realisierungsansätze sind:

- Signatur in Empfangsobjekt schreiben
- PDO enthält Togglebit oder fortlaufende Nummer
- Funktionsbaustein CAN_RECEIVE verwenden

Transmission-Types

Es werden die Transmissiontypes SYNC, d.h. synchrones Übertragen nach einem PDO-SYNC-Objekt, bzw. ASYNC, Übertragung nach einer Änderung der Netzwerkvariablen (Event durch Änderung) unterstützt. Der COB-ID des Sync-Objektes ist konfigurierbar.

Durch die Angabe einer Inhibit-Time kann das Senden von ASYNC-Objekten verzögert werden. So können z.B. stark schwankende Prozeßwerte bei einer eventgesteuerten Auswertung zu einer übermäßig starken Bus-Belastung führen. Wird die Inhibit-Time angegeben, kann das nächste PDO frühestens nach Ablauf der Zeit auf den Bus gegeben werden.
Sollen strategisch wichtige Werte im ASYNC-Modus übertragen werden, kann möglicherweise eine einmalige Übertragung zu unsicher sein. Über den Funktionsbaustein PDO_TX_-REFRESH kann von Zeit zu Zeit der wichtige PDO wiederholt werden.

Als Default-Einstellung werden alle PDOs nach einer Datenänderung (ASYNC-Modus) übertragen.

Nodeguarding

Wird eine CS0015 vom NMT-Master einmalig durch ein Guarding-Objekt angesprochen, steht er durch das zyklische Nodeguarding unter voller Kontrolle des NMT-Masters. Ist die CAN-Kommunikation gestört, wird im NMT-Master eine Guarding-Fehlermeldung erzeugt. Außerdem wird in CS0015-CANopen-Slave der Merker COP_EVENT_GUARDFAIL gesetzt.

Der Programmierer muß diese Fehlermeldungen speziell bei kritischen Applikationen in seiner Software auswerten.

ResetNode

Wird vom CANopen-Master ein ResetNode ausgelöst, müßte im Normalfall ein kompletter Neustart des CS0015-Slave durchgeführt werden (wie z.B. bei einem Watchdog-Reset). Um eine höhere Flexibilität zu erreichen, unterliegt dies bei CANopen der Kontrolle des Anwenderprogramms.
Der Anwender erkennt am Merker COP_EVENT_RESETNODE = TRUE, ob ein Reset ausgelöst wurde. Er kann dann bei Bedarf den Funktionsbaustein SOFTRESET aufrufen. Der Merker muß anschließend zurückgesetzt werden.
Im CS0015-Master muß eine lange Guarding-Zeit bzw. eine hohe Lifetime eingestellt werden, um die lange Resetphase des Slave zu überbrücken.

Emergency-Objekte

Tritt im CS0015-CANopen-Slave ein Fehler auf, wird dieser in einem Emergency-Objekt an den Master übertragen. Der COB-ID des EMCY-Objektes ist konfigurierbar.

Die Emergency-Objekte (bestehend aus 8 Datenbytes) werden gemäß CANopen in drei Teile eingeteilt.

1. Emergency-Code (Error-Code, EMCY), Byte 0 und Byte 1
2. Error-Register (Error Reg.), Byte 2
3. Daten (ergänzende Informationen), Byte 3 ... Byte 7

Folgende Fehler werden übertragen:

EMCY Code	Error Reg.	Beschreibung
0x1000	Bit 0	Fehler (generell), Ausgang ERROR gesetzt, LED - rot
0x2100	Bit 1	Leiterbruch
0x2300	Bit 1	Kurzschluß, Überlast, Übertemperatur
0x3200	Bit 2	Fehler Unter-/Überspannung
0x4000	Bit 3	Fehler Gerätetemperatur (> 85°C)
0x8100	Bit 4	Guarding-Fehler, kein Guard-Objekt empfangen
0x8200	Bit 4	SYNC-Fehler, kein Sync-Objekt empfangen

EMCY Code	Datenbyte	Beschreibung
0x2100	Byte 3	Leiterbruch-Bit QX0.0 ... QX0.7
	Byte 4	Leiterbruch-Bit QX0.8 ... QX0.15
	Byte 5	Leiterbruch-Bit QX0.16 ... QX0.23
0x2300	Byte 3	Kurzschluß-Bit QX0.0 ... QX0.7
	Byte 4	Kurzschluß-Bit QX0.8 ... QX0.15
	Byte 5	Kurzschluß-Bit QX0.16 ... QX0.23
0x8200	Byte 3	Bit 0, CAN-Fehler
	Byte 3	Bit 1, SYNC-Fehler

Funktion **NMS_SET_NODEID**

Library **COB.LIB**

Funktionssymbol

Zweck Es wird der Node-ID des CANopen-Slave eingestellt.

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet
NODEID	BYTE	Nummer des Identifiers (1 ... 30)

Funktionsausgänge, keine

Beschreibung Über die Funktion NMS_SET_NODEID kann in der Initialisierung die Knotennummer des CANopen-Slaves eingestellt werden. Dazu wird die Funktion einmalig aufgerufen. Über den Funktionseingang ENABLE wird die Ausführung der Funktion gesteuert.

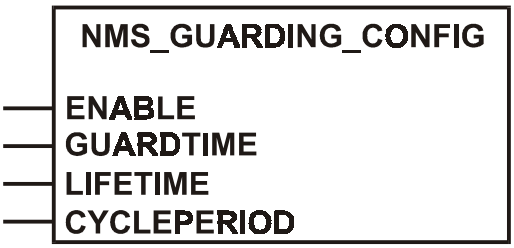
Als NODEID kann eine Nummer zwischen 1 und 30 angegeben werden.

Mit Ausführung der Funktion wird der Node-ID sofort gültig. Damit werden auch die vom Node-ID abhängigen TX- und RX-PDO des „predefined connection set“ sofort umgestellt. Der Node-ID bleibt solange gültig bis er erneut über den Funktionsaufruf oder über das Programmiersystem eingestellt wird.

Funktion **NMS_GUARDING_CONFIG**

Library **COB.LIB**

Funktionssymbol



Zweck Es wird die Guardingzeit für einen CS0015-CANopen-Slave eingestellt.

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet
GUARDTIME	TIME	Zeit zwischen zwei Überwachungsaufrufen 0 ms = keine Überwachung 1ms .. 65535ms = Überwachungszeit
LIFETIME	BYTE	Anzahl der zulässigen fehlerhaften Überwachungsaufrufen
CYCLEPERIOD	TIME	Zeit zwischen zwei SYNC-Objekten 0 ms = keine Überwachung 1ms ... 65535ms = Überwachungszeit

Funktionsausgänge, keine

Beschreibung Über die Funktion NMS_GUARDING_CONFIG kann in der Initialisierung die zulässigen Zeiten für das Node-Guarding und die SYNC-Objekte im CS0015-CANopen-Slave, eingestellt werden. Dazu wird die Funktion einmalig aufgerufen. Über den Funktionseingang ENABLE wird die Ausführung der Funktion gesteuert.

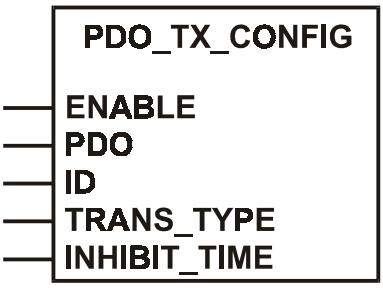
Werden in den angegebenen Zeiten nicht die jeweiligen Objekte (beim Node-Guarding ggf. mal Anzahl der LIFETIME-Zyklen) vom CS0015-Slave empfangen, werden die entsprechenden Fehlerbits (COP_GUARDFAIL_ERROR und COP_SYNCFAIL_ERROR) gesetzt. Diese müssen dann über das Anwenderprogramm ausgewertet werden. Zusätzlich besteht auch noch die Möglichkeit den Merker COP_SYNC auszuwerten. Er ist immer genau einen Zyklus TRUE.

Die angegeben Zeiten müssen etwas größer als die eingestellten Zeiten im Master sein.

Funktion **PDO_TX_CONFIG**

Library **COB.LIB**

Funktionssymbol



Zweck Initialisiert ein Sende-PDO im CS0015-CANopen-Slave.

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet
PDO	BYTE	Nummer des TX-PDO (1 ... 8)
ID	WORD	Identifier des TX-PDO (ab 380 Hex)
TRANS_TYPE	BYTE	Art der PDO-Übertragung Es werden die Typen SYNC (0,1 ... 240) und ASYNC (255) unterstützt
INHIBIT_TIME	TIME	Verzögerungszeiten für asynchronen Übertragungsmodus (0 ... 65535ms)

Funktionsausgänge, keine

Beschreibung

PDO_TX_CONFIG initialisiert ein Sende-PDO für den CANopen-Slave. Diese Funktion muß einmalig in der Initialisierung mit ENABLE = TRUE ausgeführt werden. Im Anschluß daran, wird ENABLE = FALSE gesetzt.

Am Funktionseingang PDO wird die entsprechende Nummer von 1 ... 8 angegeben.

Bei PDOs, die nicht über das „predefined connection set“ genutzt werden sollen, müssen mit einem Identifier ab 380 Hex beginnen. Anderfalls kann es zu Überschneidungen mit anderen Systemidentifieren kommen. Als Transmissiontypes (TRANS_TYPE) stehen die Betriebsarten SYNC (1) und ASYNC (255) zur Verfügung. Soll nicht bei jedem SYNC-Objekt eine Übertragung erfolgen, kann kann ein Wert zwischen 1 und 240 (Anzahl der SYNC-Objekte zwischen zwei Zugriffen) eingetragen werden.

Damit die Daten im SYNC-Modus übertragen werden, muß der SYNC-ID des Master und des Slave übereinstimmen. Als Voreinstellung ist beim Slave kein SYNC-ID eingetragen.

Im ASYNC-Modus muß gegebenenfalls noch die INHIBIT_TIME (Wartezeit) angegeben werden. Andernfalls, kann es bei stark schwankenden Werten zu einer übermäßigen Busbelastung kommen.

Sollen strategisch wichtige Werte im ASNC-Modus übertragen werden, kann möglicherweise eine einmalige Übertragung zu unsicher sein. Über den Funktionsbaustein PDO_TX_-REFRESH kann von Zeit zu Zeit der wichtige PDO wiederholt werden.

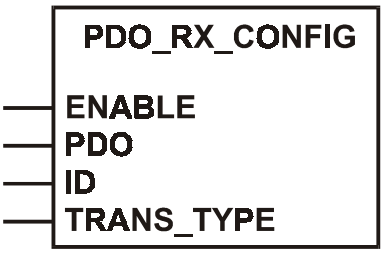
Als Voreinstellung werden alle TX-PDOs asynchron übertragen.

Wird die Funktion PDO_TX_CONFIG in einem CANopen-Master eingesetzt, muß diese vor der Ausführung der Funktion NMM_SET_NMT_MASTER abgearbeitet werden da diese einen internen CANopen-Reset auslöst. Dadurch geht die Masterfunktionalität verloren. Deshalb muß die Initialisierung zweistufig erfolgen (Master-Bootup erst einen Zyklus später starten - siehe Beispielprogramm).

Funktion **PDO_RX_CONFIG**

Library **COB.LIB**

Funktionssymbol



Zweck Initialisiert ein Empfangs-PDO im CS0015 CANopen-Slave.

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet
PDO	BYTE	Nummer des RX-PDO (1 ... 8)
ID	WORD	Identifier des RX-PDO (ab 380 Hex)
TRANS_TYPE	BYTE	Art der PDO-Übertragung Es werden die Typen SYNC (0, 1 ... 240) und ASYNC (255) unterstützt

Funktionsausgänge, keine

Beschreibung

PDO_RX_CONFIG initialisiert ein Empfangs-PDO für den CANopen-Slave. Diese Funktion muß einmalig in der Initialisierung mit ENABLE = TRUE ausgeführt werden. Im Anschluß daran, wird ENABLE = FALSE gesetzt.

Am Funktionseingang PDO wird die entsprechende Nummer von 1 ... 8 angegeben.

Bei PDOs, die nicht über das „predefined connection set“ genutzt werden sollen, müssen mit einem Identifier ab 380 Hex beginnen. Anderfalls kann es zu Überschneidungen mit anderen Systemidentifieren kommen. Als Transmissiontypes (TRANS_TYPE) stehen die Betriebsarten SYNC (0) und ASYNC (255) zur Verfügung. Soll nicht bei jedem SYNC-Objekt eine Übertragung erfolgen, kann kann ein Wert zwischen 1 und 240 (Anzahl der SYNC-Objekte zwischen zwei Zugriffen) eingetragen werden.

Damit die Daten im SYNC-Modus übertragen werden können, muß der SYNC-ID des Master und des Slave übereinstimmen. Als Voreinstellung ist beim Slave kein SYNC-ID eingetragen.

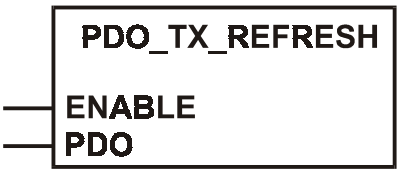
Als Voreinstellung werden alle RX-PDOs asynchron übertragen.

Wird die Funktion PDO_RX_CONFIG in einem CANopen-Master eingesetzt, muß diese vor der Ausführung der Funktion NMM_SET_NMT_MASTER abgearbeitet werden da diese einen internen CANopen-Reset auslöst. Dadurch geht die Masterfunktionalität verloren. Deshalb muß die Initialisierung zweistufig erfolgen (Master-Bootup erst einen Zyklus später starten - siehe Beispielprogramm).

Funktion **PDO_TX_REFRESH**

Library **COB.LIB**

Funktionssymbol



Zweck Ein gesendetes TX-PDO wird noch einmal übertragen.

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet
PDO	BYTE	Nummer des TX-PDO (1 ... 8)

Funktionsausgänge, keine

Beschreibung

Gerade wenn strategisch wichtige Werte im ASNC-Modus übertragen werden sollen, kann möglicherweise eine einmalige Übertragung zu unsicher sein. Über den Funktionsbaustein PDO_TX_REFRESH kann von Zeit zu Zeit der wichtige PDO wiederholt werden.

Die Funktion darf nicht in jedem Zyklus ausgeführt werden, da es sonst zu einer Überlastung des CAN kommen würde. Die Ausführung kann daher mit dem Funktionseingang ENABLE gesteuert werden

Am Funktionseingang PDO wird die entsprechende Nummer von 1 ... 8 angegeben.

5.9. Die CS0015 als CANopen-Master

In einem typischen CANopen-Netzwerk ist ein Netzwerk-Master vorhanden. Die im folgenden Text beschriebenen Funktionen stellen alle Basisdienste zum Aufbau einer Master-Software für die Steuerung CS0015 zur Verfügung. Durch den Einsatz der Funktionen können die Slave-Knoten ins CAN-Netzwerk eingebundenen, konfiguriert und überwacht werden. Um einen einfachen Einstieg in CANopen zu erreichen (speziell in Applikationen in denen „nur“ eine dezentrale Erweiterung der Ein-/Ausgangsebene benötigt wird) wurden zwei Funktionen COP_MSTR_BOOTUP und COP_MSTR_MAIN in der Programmiersprache ST realisiert. Diese nutzen die nachfolgend beschriebenen Funktionen. Eine genaue Beschreibung erfolgt im Kapitel 5.10.

Damit die CS0015 als CANopen-Master arbeitet, muß der Merker CAN_OPEN zu Beginn des Programms (in der Initialisierung) auf TRUE gesetzt werden und die Funktion NMM_SET_NMT_MASTER einmalig aufgerufen werden.

Funktion **NMM_SET_NMT_MASTER**

Library **COB.LIB**

Funktionssymbol

Zweck Initialisiert das Steuerungsmodul als Master.

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet

Funktionsausgänge, keine

Beschreibung

NMM_SET_NMT_MASTER initialisiert die Steuerung als CANopen-Master. Wird diese Funktion nicht aufgerufen, arbeitet die Steuerung nur als „normaler“ CANopen-Teilnehmer (Slave) im Netzwerk.

Der Netzwerk-Master ist für die Konfiguration und die Überwachung des Netzwerkes zuständig. In einem CANopen-Netzwerk darf nur ein NMT-Master, d.h. ein Master mit Management-Funktion, vorhanden sein.

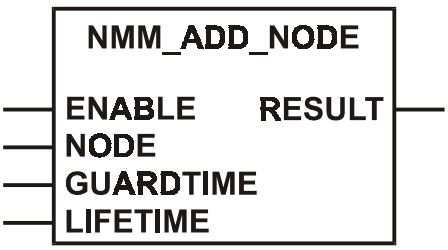
Der Programmierer hat die Aufgabe alle vom NMT-Master zur Verfügung gestellten Statusinformationen auszuwerten um ein sicheres Netz zu betreiben.

Werden die Funktionen PDO_RX_CONFIG und PDO_TX_CONFIG in einem CANopen-Master eingesetzt, müssen diese vor der Ausführung der Funktion NMM_SET_NMT_MASTER abgearbeitet werden da diese einen internen CANopen-Reset auslösen. Dadurch geht die Masterfunktionalität verloren. Deshalb muß die Initialisierung zweistufig erfolgen (Master-Bootup erst einen Zyklus später starten - siehe Beispielprogramm).

Funktion **NMM_ADD_NODE**

Library **COB.LIB**

Funktionssymbol



Zweck Initialisiert ein Überwachungsobjekt für den angegebenen Knoten.

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet
NODE	BYTE	Knotennummer von 1 ... 127
GUARDTIME	TIME	Zeit zwischen zwei Überwachungsaufrufen
LIFETIME	BYTE	Anzahl der zulässigen fehlerhaften Überwachungsaufrufen

Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BYTE	Ergebnis: 0 = erfolgreich 1 = nicht erfolgreich 2 = ungültige Parameter

Beschreibung NMM_ADD_NODE initialisiert den CANopen-Knoten und ein Guardingobjekt im NMT_Master. Der Lifetime-Faktor legt fest, wie oft ein fehlerhafter Aufruf erfolgen darf. Die Funktion muß einmalig in der Initialisierung für jeden Knoten aufgerufen werden. Ein Beispiel ist in der Datei NMT_MSTR.PRO abgelegt.

Das Nodeguarding wird erst ausgeführt, wenn es über die Funktion NMM_START_GUARDING gestartet wurde.

Der Programmierer hat die Aufgabe durch Auswertung des Guardings und der übrigen vom System zur Verfügung gestellten Fehlerbits die genaue Fehlerursache zu lokalisieren und in Abhängigkeit von der Applikation zu reagieren.

Wird ein Knoten nicht mit NMM_ADD_NODE initialisiert, kann er - unabhängig vom fehlenden Nodeguarding - auch nicht über andere Masterfunktionen (z.B. SDO_WRITE) angesprochen werden.

Funktion **NMM_START_GUARDING**

Library **COB.LIB**

Funktionssymbol

Zweck Startet das Nodeguarding für einen oder alle initialisierten Knoten.

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet
NODE	BYTE	alle initialisierten Knoten: 0 initialisierter Knoten: 1 ... 127

Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BYTE	Ergebnis: 0 = erfolgreich 2 = ungültige Parameter

Beschreibung NMM_START_GUARDING startet das Nodeguarding für einen einzelnen Knoten oder alle angeschlossenen Knoten (gesamtes Netzwerk). Dazu muß zuvor ein Guardingobjekt für den angegebenen CANopen-Knoten mit NMM_ADD_NODE initialisiert werden.

Der Programmierer hat die Aufgabe durch Auswertung des Guardings und der übrigen vom System zur Verfügung gestellten Fehlerbits die genaue Fehlerursache zu lokalisieren und in Abhängigkeit von der Applikation zu reagieren.

Funktion **NMM_STOP_GUARDING**

Library **COB.LIB**

Funktionssymbol

Zweck Stoppt das Nodeguarding für einen oder alle initialisierten Knoten.

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet
NODE	BYTE	alle initialisierten Knoten: 0 initialisierter Knoten: 1 ... 127

Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BYTE	Ergebnis: 0 = erfolgreich 2 = ungültige Parameter

Beschreibung NMM_STOP_GUARDING stoppt das Nodeguarding für einen einzelnen Knoten oder alle angeschlossenen Knoten (gesamtes Netzwerk).

Wird das Nodeguarding abgeschaltet, wird von der Steuerung das Fehlen eines Knotens nicht mehr erkannt.

Der Programmierer hat die Aufgabe durch Auswertung des Guardings und der übrigen vom System zur Verfügung gestellten Fehlerbits die genaue Fehlerursache zu lokalisieren und in Abhängigkeit von der Applikation zu reagieren.

Funktion **NMM_NODE_GUARDING**

Library **COB.LIB**

Funktionssymbol

Zweck Die Funktion ruft die Überwachung aller initialisierten CANopen-Knoten auf.

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet
AUTO_RE-START	BOOL	TRUE: Überwachter Knoten wird nach einem Guarding-Fehler automatisch Operational gesetzt. FALSE: Knoten bleibt im Zustand Pre-Operational

Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BYTE	Ergebnis: 0 = erfolgreich > 0 = fehlende Knoten 0xFF = fehlerhafter Aufruf

Beschreibung NMM_NODE_GUARDING organisiert das Nodeguarding für alle initialisierten Knoten im gesamten Netzwerk. Die Funktion muß zyklisch aufgerufen werden. Fehlen mehrere Knoten, werden diese nacheinander angezeigt. Das Nodeguarding wird nur ausgeführt, wenn mit der Funktion NMM_START_GUARDING die Netzwerküberwachung gestartet wurde. Der Funktionseingang AUTO_RESTART ermöglicht das automatische Starten eines Knotens durch den Master, nach einem Guarding-Fehler. Ist AUTO_RESTART auf TRUE gesetzt, wird nach dem einem NODE_RESET der Knoten automatisch wieder „Operational“ gesetzt. Ist der Eingang auf FALSE gesetzt, verbleibt der Knoten im Zustand „Pre-Operational“.

Es wird empfohlen mit AUTO_RESTART = TRUE zu arbeiten.

Wird das Nodeguarding abgeschaltet, wird von der Steuerung das Fehlen eines Knotens nicht mehr erkannt.

Funktion **NMM_SET_PREOPERATIONAL**

Library **COB.LIB**

Funktionssymbol

Zweck

Versetzt einen einzelnen Knoten oder das gesamte Netzwerk in den Zustand „Pre-Operational“.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet
NODE	BYTE	alle initialisierten Knoten: 0 initialisierter Knoten: 1 ... 127

Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BYTE	Ergebnis: 0 = erfolgreich 1 = Sendefehler 2 = ungültige Parameter 255 = NMT-Master nicht aktiv

Beschreibung

NMM_SET_PREOPERATIONAL setzt den angegebenen Knoten oder das gesamte Netzwerk in den Zustand „Pre-Operational“ (siehe auch Kapitel 5.7.). Nach der Initialisierung eines (oder aller) Netzwerkknoten, wird dieser im Normalfall in den Zustand „Pre-Operational“ versetzt. In diesem Zustand kann der Knoten (oder die Knoten) nur über die SDOs mit dem für das Netzwerkmanagement zuständigen NMT-Master kommunizieren.

Funktion **NMM_SET_OPERATIONAL**

Library **COB.LIB**

Funktionssymbol

Zweck Versetzt einen einzelnen Knoten oder das gesamte Netzwerk in den Zustand „Operational“.

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet
NODE	BYTE	alle initialisierten Knoten: 0 initialisierter Knoten: 1 ... 127

Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BYTE	Ergebnis: 0 = erfolgreich 1 = Sendefehler 2 = ungültige Parameter 255 = NMT-Master nicht aktiv

Beschreibung NMM_SET_OPERATIONAL setzt den angegebenen Konten oder das gesamte Netzwerk in den Zustand „Operational“ (siehe auch Kapitel 5.7.). Nach der Initialisierung eines oder aller Netzwerkknoten, wird nach dem Zustand „Pre-Operational“ der Zustand „Operational“ erreicht. In diesem Zustand kann der Knoten (oder die Knoten) über alle Kommunikationsdienste (SDOs und PDOs) mit dem für das Netzwerkmanagement zuständigen NMT-Master und allen anderern Netzwerk-Teilnehmern kommunizieren.

Auch der Netzwerkmaster muß einmalig in den Zustand „Operational“ gesetzt werden, damit eine ordnungsgemäße Kommunikation aufgenommen werden kann.

Funktion **NMM_SET_PREPARED**

Library **COB.LIB**

Funktionssymbol

Zweck

Versetzt einen einzelnen Knoten oder das gesamte Netzwerk in den Zustand „Prepared“.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet
NODE	BYTE	alle initialisierten Knoten: 0 initialisierter Knoten: 1 ... 127

Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BYTE	Ergebnis: 0 = erfolgreich 1 = Sendefehler 2 = ungültige Parameter 255 = NMT-Master nicht aktiv

Beschreibung

NMM_SET_PREPARED setzt den angegebenen Knoten oder das gesamte Netzwerk in den Zustand „Prepared“ (siehe auch Kapitel 5.7.). In diesem Zustand nimmt der Knoten (oder die Knoten) nicht mehr an der PDO-Kommunikation teil. Außerdem kann auch nicht mehr über die SDOs kommuniziert werden.

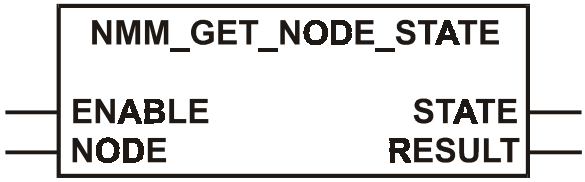
.

Dieser Zustand wird oft für anwenderspezifische Belange genutzt um z.B. einen oder alle Teilnehmer zeitweise vom Bus abzutrennen. Der Zustand „Prepared“, kann nur durch NMM_SET_PREOPERATIONAL / NMM_SET_OPERATIONAL aufgehoben werden.

Funktion **NMM_GET_NODE_STATE**

Library **COB.LIB**

Funktionssymbol



Zweck Gibt den Netzwerkstatus eines CANopen-Knotens zurück.

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet
NODE	BYTE	alle initialisierten Knoten: 0 initialisierter Knoten: 1 ... 127

Funktionsausgänge

Name	Datentyp	Beschreibung
STATE	BYTE	Status gemäß CANopen Spezifikation
RESULT	BYTE	Ergebnis: 0 = erfolgreich 2 = ungültige Parameter 255 = NMT-Master nicht aktiv

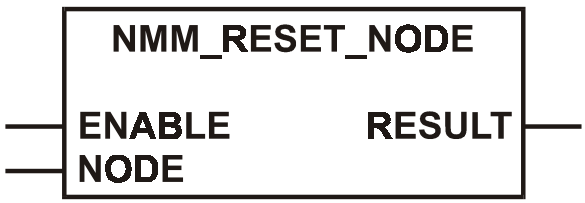
Beschreibung NMM_GET_NODE_STATE gibt den aktuellen Netzwerkstatus (Preoperational, Operational, Prepared) eines oder aller Knoten zurück. Der Wert ergibt sich aus der CANopen Spezifikation.

127	State Pre-Operational
5	State Operational
4	State Prepared

Funktion **NMM_RESET_NODE**

Library **COB.LIB**

Funktionssymbol



Zweck Setzt die Applikations- und Kommunikationsparameter für einen oder alle Knoten auf die Default-Werte zurück.

.

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet
NODE	BYTE	alle initialisierten Knoten: 0 initialisierter Knoten: 1 ... 127

Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BYTE	Ergebnis: 0 = erfolgreich 1 = Sendefehler 2 = ungültige Parameter 255 = NMT-Master nicht aktiv

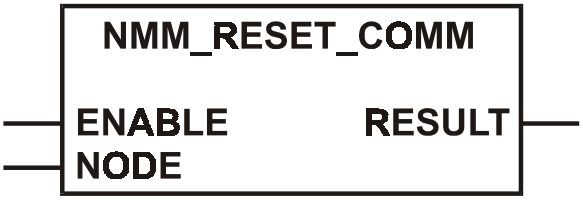
Beschreibung NMM_RESET_NODE führt für den aufgerufenen Knoten (oder alle Knoten im Netz) ein Reset durch. Dabei bleiben alle spannungsausfallsicheren Daten im Knoten gespeichert. Nach dem Reset durchläuft der Knoten die nomale Initialisierungsroutine.

Das genaue Verhalten nach einem Reset muß den gerätespezifischen Unterlagen des Knotenbausteins entnommen werden.

Funktion **NMM_RESET_COMM**

Library **COB.LIB**

Funktionssymbol



Zweck

Setzt die Kommunikationsparameter für einen oder alle Knoten auf die Default-Werte zurück.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet
NODE	BYTE	alle initialisierten Knoten: 0 initialisierter Knoten: 1 .. 127

Funktionsausgänge

Name	Datentyp	Beschreibung
STATE	BYTE	Status gemäß CANopen Spezifikation
RESULT	BYTE	Ergebnis: 0 = erfolgreich 1 = Sendefehler 2 = ungültige Parameter 255 = NMT-Master nicht aktiv

Beschreibung

NMM_RESET_COMM führt für den aufgerufenen Knoten (oder alle Knoten im Netz) ein Reset für die CAN-Schnittstelle durch. Dabei bleiben alle spannungsausfallsicheren Daten im Knoten gespeichert.

Das genaue Verhalten nach einem Reset muß den geräte-spezifischen Unterlagen entnommen werden.

Funktion **PDO_INI_SEND_SYNC_OBJ**

Library **COB.LIB**

Funktionssymbol

Zweck Initialisiert das PDO-Sync-Objekt zur synchronen Abtastung von E/A-Daten.

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet

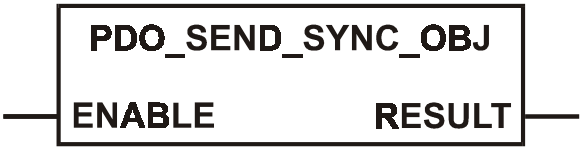
Funktionsausgänge, keine

Beschreibung PDO_INI_SEND_SYNC_OBJ initialisiert das SYNC-Objekt zur synchronen Abtastung von Daten im CANopen-Netzwerk (siehe auch 5.7. Process Data Objects). Die Funktion muß einmalig in der Initialisierung aufgerufen werden. Über die Funktion PDO_SEND_SYNC_OBJ wird dann das SYNC-Objekt übertragen.

Funktion **PDO_SEND_SYNC_OBJ**

Library **COB.LIB**

Funktionssymbol



Zweck Sendet das Synchronisations-Objekt

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet

Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BOOL	TRUE: Funktion wurde erfolgreich abgearbeitet

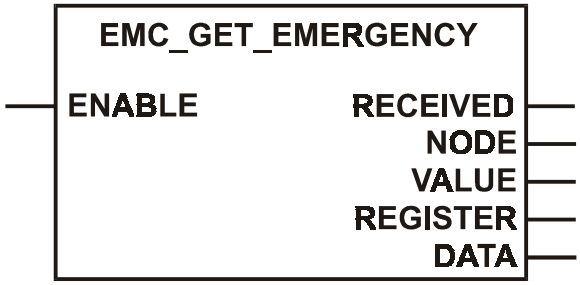
Beschreibung PDO_SEND_SYNC_OBJ setzt ein SYNC-Objekt in das CANopen-Netzwerk ab. SYNC-Objekte werden zur synchronen Abtastung von Daten (siehe auch 5.7. Process Data Objects) genutzt. Diese Funktion muß zyklisch aufgerufen werden. Die Zeitsteuerung erfolgt wie im Beispiel mit Hilfe der beiden Systemmerker COP_PRESYNC bzw. COB_SYNC.

```
0012    send_sync_obj : PDO_SEND_SYNC_OBJ;
0013    node_guarding : NMM_NODE_GUARDING;
0014
0015END_VAR
0001IF ENABLE THEN
0002    timer1 (IN := NOT m1, PT := T#500ms);
0003    m1 := timer1.Q;
0004    IF m1 THEN
0005        COP_PRESYNC := TRUE;
0006    END_IF
0007
0008    timer2 (IN := COP_PRESYNC, PT := T#50ms);
0009    COP_SYNC := timer2.Q;
0010
0011    IF COP_SYNC THEN
0012        COP_PRESYNC := FALSE;
0013        send_sync_obj (ENABLE := TRUE);
0014    END_IF
0015
```

Funktion **EMC_GET_EMERGENCY**

Library **COB.LIB**

Funktionssymbol



Zweck Auslesen des CANopen Emergency-Objektes.

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet

Funktionsausgänge

Name	Datentyp	Beschreibung
RECEIVED	BOOL	TRUE: neue Fehlerdaten stehen an
NODE	BYTE	Knotennummer
VALUE	WORD	Fehlercodes des Emergency Objekts
REGISTER	BYTE	Fehlerregister gemäß Index 0x1001
DATA	ARRAY	herstellerspezifische Fehlerinformation

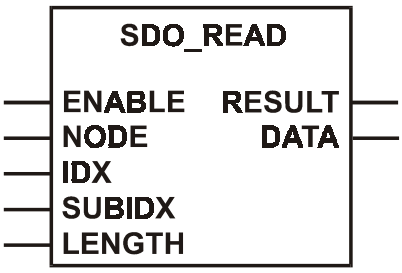
Beschreibung

Die Funktion EMC_GET_EMERGENCY fragt die Fehlerdaten der angeschlossenen Netzwerk-Knoten ab. Sobald neue Daten anstehen, wird der Ausgang RECEIVED für einen Zyklus auf TRUE gesetzt. Durch anschließendes Abfragen der Knotennummer (NODE), des Fehlercodes (VALUE) und des Fehlerregisters (REGISTER) kann dann der aufgetretene Fehler analysiert werden. Am Ausgang DATA stehen zusätzlich noch die herstellerspezifischen Informationen des Knotens zur Verfügung. Bei den E-/A-Modulen der ifm electronic erhält man so z.B. Informationen über einen Leiterbruch oder Kurzschluß an den Ausgängen.

Funktion **SDO_READ**

Library **COP.LIB**

Funktionssymbol



Zweck

Liest das SDO mit den angegebenen Indizes aus dem Knoten aus.

Parameter

Funktionseingänge

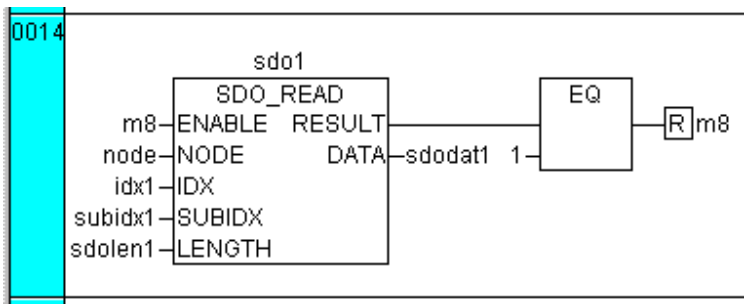
Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet
NODE	BYTE	Nummer das Knotens
IDX	WORD	Index im Objektverzeichnis
SUBIDX	WORD	Subindex bezogen auf den Index im Objektverzeichnis.
LENGTH	WORD	Länge des Eintrags in Anzahl der Bytes

Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BYTE	0 Funktion inaktiv 1 Funktionsausführung beendet 2 Funktion ist aktiv
DATA	ARRAY	Ausgelesene Daten (Array, Länge 0 ... 255)

Beschreibung

Über die Funktion SDO_READ können die Einträge im Objektverzeichnis gelesen werden. Dadurch ist es möglich die Knotenparameter gezielt zu lesen. Damit diese Funktion genutzt werden kann muß sich der Knoten im Zustand „Pre-Operational“ oder „Operational“ befinden.



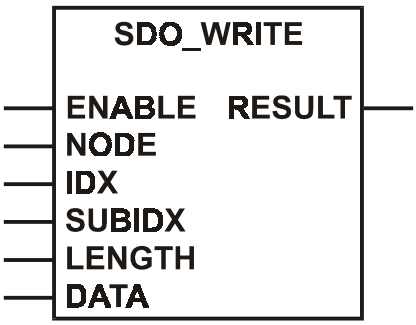
Der Eingang ENABLE steuert die Ausführung der Funktion. Da aber mit jedem Aufruf der Funktion das Datenarray übergeben wird, belastet auch bei ENABLE=FALSE die Funktion den Steuerungszyklus. Daher sollte SDO_READ, wenn die Funktion nicht genutzt wird, übersprungen werden.

Der Wert für LENGTH sollte der Länge des erwarteten Daten-Objektes entsprechen.

Funktion **SDO_WRITE**

Library **COP.LIB**

Funktionssymbol



Zweck Schreibt das SDO mit den angegebenen Indizes in den Knoten.

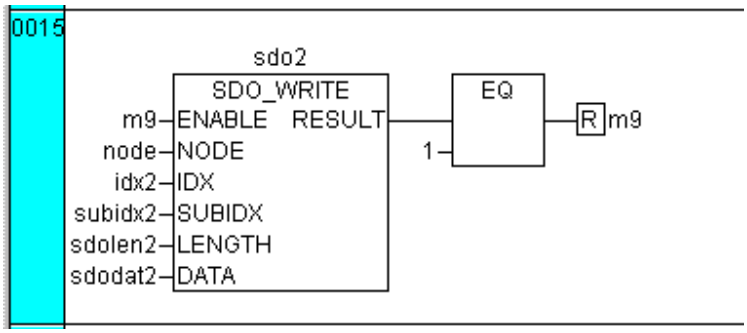
Parameter Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet
NODE	BYTE	Nummer das Knotens
IDX	WORD	Index im Objektverzeichnis
SUBIDX	WORD	Subindex bezogen auf den Index im Objektverzeichnis.
LENGTH	WORD	Länge des Eintrags in „Anzahl der Bytes“
DATA	ARRAY	Sendedaten (Array, Länge 0 ... 255)

Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BYTE	0 Funktion inaktiv 1 Funktionsausführung beendet 2 Funktion ist aktiv

Beschreibung Über die Funktion SDO_WRITE können die Einträge im Objektverzeichnis geschrieben werden. Dadurch ist es möglich die Knotenparameter gezielt zu setzen. Damit diese Funktion genutzt werden kann, muß sich der Knoten im Zustand „Pre-Operational“ oder „Operational“ befinden.



Der Eingang ENABLE steuert die Ausführung der Funktion. Da aber mit jedem Aufruf der Funktion das Datenarray übergeben wird, belastet auch bei ENABLE=FALSE die Funktion den Steuerungszyklus. Daher sollte SDO_WRITE, wenn die Funktion nicht genutzt wird, übersprungen werden.

Der Wert für LENGTH muß mit der Länge des Sendearray übereinstimmen. Anderfalls treten Störungen bei der SDO-Kommunikation auf.

5.10. Funktionen für CANopen-E/A-Module der ifm electronic

Sehr oft setzt sich eine Steuerungslösung in einer Applikation aus einer Zentralsteuerung und einem bzw. mehreren dezentralen Ein-/Ausgangs-Modulen zusammen. In diesen Applikationen bildet die Zentralsteuerung gleichzeitig den Netzwerkmaster (siehe Kapitel 5.9.). Um bei solchen Anwendungen einen einfachen Einstieg zu bekommem können die nachfolgend beschriebenen Funktionen genutzt werden. **Wenn der Anwender die volle CANopen-Funktionalität nutzen möchte, muß er die in den vorangehenden Kapiteln beschriebenen Funktionen einsetzen. In diesem Fall werden die folgenden Funktionen nicht benötigt.** COP_MSTR_BOOTUP und COP_MSTR_MAIN wurden bewußt in der Sprache ST geschrieben. Dadurch können diese, wenn gewünscht, noch erweitert bzw. verändert werden (Quellcode NMT_MSTR.PRO)

Speziell zur Konfiguration und zur Auswertung der E/A-Module der ifm electronic, dienen die anderen Funktionen. Die Funktionen SLAVE_CRxxxx_CONFIG ermöglicht dem Programmierer die Knotenkonfiguration der Ein- und Ausgänge direkt aus der Applikationssoftware einzustellen bzw. diese aus einem angewählten Knoten zu lesen.

Über die Funktionen SLAVE_CRxxxx_WORK werden durch den zyklischen Aufruf die Ein- und Ausgangsdaten (digital und analog) über einem festgegelegten Merkerbereich ausgetauscht (lesen und schreiben). Über diese Merkeradressen kann dann direkt in der Applikation auf die Prozeßdaten zugegriffen werden. Die Merkeradressen sind nach folgenden Schema organisiert:

Adresse	Bitadresse	Bedeutung
%MW1010		1. Slave, 1. Anschluß, analoge E/A-Daten
	%MX1010.0	1. Slave, 1. Anschluß, digitale E/A-Daten
%MW1011		1. Slave, 2. Anschluß, analoge E/A-Daten
	%MX1011.0	1. Slave, 2. Anschluß, digitale E/A-Daten
%MW1012		1. Slave, 3. Anschluß, analoge E/A-Daten
	%MX1012.0	1. Slave, 3. Anschluß, digitale E/A-Daten
:	:	:
%MW1017		1. Slave, 8. Anschluß, analoge E/A-Daten
	%MX1017.0	1. Slave, 8. Anschluß, digitale E/A-Daten
%MW1020		2. Slave, 1. Anschluß, analoge E/A-Daten
	%MX1020.0	2. Slave, 1. Anschluß, digitale E/A-Daten
%MW1021		2. Slave, 2. Anschluß, analoge E/A-Daten
	%MX1021.0	2. Slave, 2. Anschluß, digitale E/A-Daten
:	:	:
%MW1327		32. Slave, 8. Anschluß, analoge E/A-Daten
	%MX1327.0	32. Slave, 8. Anschluß, digitale E/A-Daten

Dabei beschreibt die letzte Stelle der Wortadresse den Anschluß des Knotens Nr 1 - 8 (0 - 7) und die zweite und dritte Stelle die Knotennummer 1 - 32 (1 - 20 Hex). Standardmäßig ist der vordefinierte Adressbereich für 32 E/A-Module ausgelegt.

Grundsätzlicher Programmaufbau

Um in einer Steuerungsapplikation die E/A-Module zu nutzen, kann folgender Programmaufbau eingesetzt werden. Dieser unterstützt in einer Standardapplikation den Einsatz von bis zu 31 E/A-Modulen. Als 32. Teilnehmer wird die als Netzwerk-Master (NMT-Master) konfigurierte Steuerung CS0015 angeschlossen. Ein Knoten mit der Adresse 0 ist nicht zulässig, da diese Adresse für die systemweite Steuerung aller Knoten genutzt wird (siehe auch NMM_NMT-Funktionen Kap. 5.9.).

Programmschritt 1

COP_MSTR_BOOTUP
Die Funktion initialisiert die Steuerung als Master und die angeschlossenen Knoten. Sie wird nur in der Bootup-Phase ausgeführt. In dieser Funktion wird auch der Merker CAN_OPEN auf TRUE gesetzt.

Programmschritt 2

COP_MSTR_MAIN
Die Funktion erzeugt durch ihren zyklischen Aufruf das Sync-Objekt zur synchronen Übertragung der E/A-Daten.

Programmschritt 3

SLAVE_CRxxxx_CONFIG
Slavekonfiguration für jeden angeschlossenen E/A-Knoten.
Nach erfolgreicher Konfiguration wird diese Funktion wieder deaktiviert.

Programmschritt 4

NMM_SET_OPERATIONAL
Durch Aufruf mit dem Parameter NODE = 0 wird das gesamte Netzwerk (auch der NMT-Master) in den Operational-Mode versetzt. Die Funktion darf nur einmalig ausgeführt werden.

Programmschritt 5

SLAVE_CRxxxx_WORK
Durch zyklischen Aufruf der Funktion, werden die E/A-Daten der Slavemodule in den festgelegten Merkerbereich der CS0015 geschrieben bzw. aus diesem gelesen.

Programmschritt 6

EMC_GET_EMERGENCY
Die Funktion stellt die Emergency- (Fehler-) Daten der angeschlossenen Knoten zur Verfügung**.**

Das Beispielprogramm EA_SLAVE.PRO im Verzeichnis DEMO\CS0015 zeigt den Aufbau der Software für zwei Knoten. Es kann als Basis für den Ausbau einer Applikationssoftware dienen. Wird nur ein Slave-Knoten an die CS0015 angeschlossen müssen alle Funktionsaufrufe für den zweiten Knoten entfernt werden. Zusätzlich sind in diesem Programm noch einige weitere Masterfunktionen enthalten (z.B. SDO_READ, SDO_WRITE). Über diese Funktionen kann online über das Programmiersystem mit den angeschlossenen Slaves kommuniziert werden.

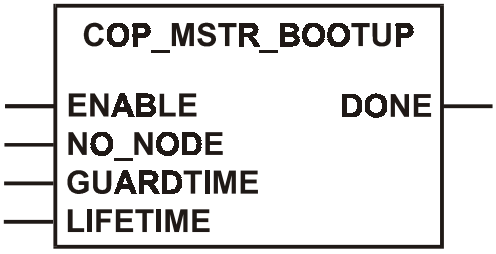
Funktion

COP_MSTR_BOOTUP

Library

NMT_MSTR.LIB

Funktionssymbol



Zweck

Initialisiert das Steuerungsmodul als CANopen NMT-Master und alle angeschlossenen E/A-Knoten.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet
NO_NODE	BYTE	Anzahl der angeschlossenen Knoten ohne NMT-Master
GUARDTIME	TIME	Guardzeit zur Knotenüberwachung
LIFETIME	BYTE	Lifetimefaktor für Knotenüberwachung

Funktionsausgänge

Name	Datentyp	Beschreibung
DONE	BOOL	FALSE: BOOTUP ist noch aktiv TRUE: BOOTUP ist beendet

Beschreibung

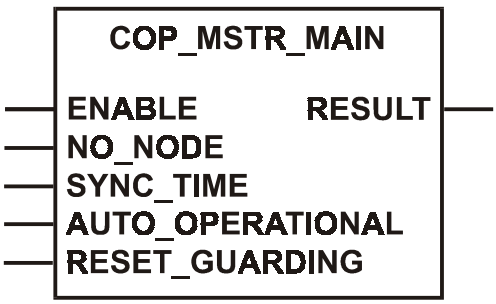
COP_MSTR_BOOTUP versetzt die CS0015 in den CANopen-Modus und initialisiert die Steuerung als NMT-Master. Gleichzeitig werden dem Master die Anzahl der angeschlossenen Knoten (NO_NODE) mit der für sie festgelegten Überwachungszeit (GUARDTIME und LIFETIME-Faktor) bekanntgemacht. Wenn der Bootup-Vorgang beendet ist (> 2 sek.), wird der Funktionsausgang DONE auf TRUE gesetzt.

Nach erfolgreichem Bootup muß die Ausführung der Funktion über den Eingang ENABLE gesperrt werden.

Funktion **COP_MSTR_MAIN**

Library **NMT_MSTR.LIB**

Funktionssymbol



Zweck Erzeugt zyklisch das Sync-Objekt und überwacht die angeschlossenen Knoten

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet
NO_NODE	BYTE	Anzahl der angeschlossenen Knoten ohne NMT-Master
SYNC_TIME	TIME	Zeit zwischen zwei SYNC-Objekten zur synchronen Abtastung von Daten
AUTO_ OPERA-TIONAL	BOOL	TRUE: Überwachter Knoten wird nach einem Guarding-Fehler automatisch „Operational“ gesetzt. FALSE: Knoten bleibt im Zustand „Pre-Operational“
RESET_ GUARDING	BOOL	TRUE: Gurading-Fehlerregister löschen

Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	ARRAY	Das Fehlerregister kann maximal 8 nicht erkannte Knoten speichern

Beschreibung COP_MSTR_MAIN muß im Programm zyklisch ausgeführt werden. Dadurch wird das SYNC-Objekt für die angeschlossenen Slave-Module erzeugt. Außerdem wird gleichzeitig das Netzwerk überwacht. Fällt ein Slave aus, wird die Nummer des Knotens in das Array RESULT eingetragen. Es können so maximal 8 Fehler gespeichert werden. Diese werden in der Reihenfolge ihres Auftretens eingetragen. Der Fehlerspeicher kann über den Funktionseingang RESET_GUARDING wieder gelöscht werden.

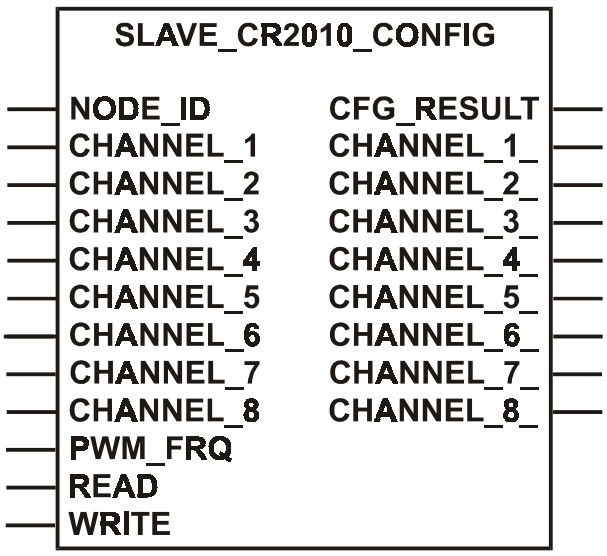
Über den Funktionseingang AUTO_OPERATIONAL kann der automatische Neustart eines Knotens nach einem Guarding-Fehler angewählt werden. Ist AUTO_OPERATIONAL auf TRUE gesetzt, wird nach Beseitigung der Störung, der entsprechende Knoten wieder in den Modus OPERATIONAL gesetzt. Dadurch nimmt er direkt wieder am PDO-Austausch teil (E/A-Daten werden gelesen und geschrieben). Ist AUTO_OPERATIONAL = FALSE, verharrt der Knoten nach der Störungsbeseitigung im Zustand „Pre-Operational“. Er muß dann über den NMT-Master gezielt in den Zustand „Operational“ gesetzt werden (Funktion NMM_SET_OPERATIONAL).

Sollen schnelle Vorgänge bearbeitet werden, müssen die SYNC-Zeiten angepaßt werden. Diese können nur im Quell-code verändert werden (NMT_MSTR.PRO).

Funktion **SLAVE_CR2010_CONFIG**

Library **CSxxxx.LIB**

Funktionssymbol



Zweck Parametriert bzw. liest die Konfiguration eines E/A-Moduls.

Parameter Funktionseingänge

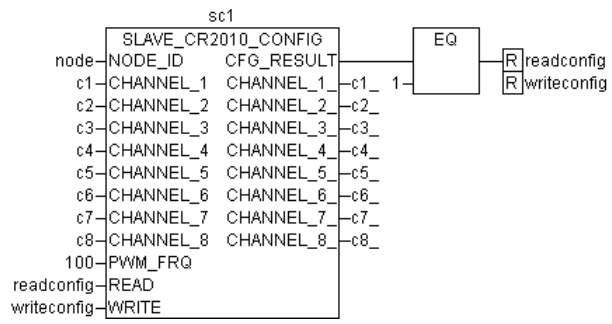
Name	Datentyp	Beschreibung
NODE_ID	BYTE	Knotennummer
CHANNEL_1	BYTE	Konfigurationsparameter für Kanal 1 0 = AUS, 1 = binär Eingang
CHANNEL_2	BYTE	Konfigurationsparameter für Kanal 2 0 = AUS, 2 = binär Ausgang 3 = analog Eing., 4 = analog Ausg.
CHANNEL_3	BYTE	Konfigurationsparameter für Kanal 3 0 = AUS, 1 = binär Eingang
CHANNEL_4	BYTE	Konfigurationsparameter für Kanal 4 0 = AUS, 2 = binär Ausgang 3 = analog Eing., 4 = analog Ausg.
CHANNEL_5	BYTE	Konfigurationsparameter für Kanal 5 0 = AUS, 1 = binär Eingang
CHANNEL_6	BYTE	Konfigurationsparameter für Kanal 6 0 = AUS, 2 = binär Ausgang 3 = analog Eing., 4 = analog Ausg.
CHANNEL_7	BYTE	Konfigurationsparameter für Kanal 7 0 = AUS, 1 = binär Eingang
CHANNEL_8	BYTE	Konfigurationsparameter für Kanal 8 0 = AUS, 2 = binär Ausgang 3 = analog Eing., 4 = analog Ausg.
PWM_FRQ	BYTE	PWM-Frequenz in Hz (20 ... 200 Hz)
READ	BOOL	aktuelle Modulkonf. lesen
WRITE	BOOL	aktuelle Modulkonf. schreiben

Funktionsausgänge

Name	Datentyp	Beschreibung
CFG_RESULT	BYTE	1 = Konfiguration wurde erfolgreich gelesen oder geschrieben 2 = Konfiguration noch nicht gelesen oder geschrieben
CHANNEL_1_	BYTE	aktuelle Konfigurationsparameter für Kanal 1
CHANNEL_2_	BYTE	aktuelle Konfigurationsparameter für Kanal 2
CHANNEL_3_	BYTE	aktuelle Konfigurationsparameter für Kanal 3
CHANNEL_4_	BYTE	aktuelle Konfigurationsparameter für Kanal 4
CHANNEL_5_	BYTE	aktuelle Konfigurationsparameter für Kanal 5
CHANNEL_6_	BYTE	aktuelle Konfigurationsparameter für Kanal 6
CHANNEL_7_	BYTE	aktuelle Konfigurationsparameter für Kanal 7
CHANNEL_8_	BYTE	aktuelle Konfigurationsparameter für Kanal 8

Beschreibung

Die Funktion SLAVE_CR2010_CONFIG setzt oder liest die E/A-Konfigurationsparameter der ifm-eigenen 8-kanaligen Module. Über die Parameter wird die gewünschte Konfiguration eingestellt. Zur Kontrolle des Funktionsablaufes sollten die Eingänge READ bzw. WRITE solange gesetzt bleiben, bis am Funktionsausgang CFG_RESULT der Wert 1 ansteht. Sind die Daten noch nicht aktuell bzw. aktualisiert worden, oder können diese nicht gelesen oder geschrieben werden, steht am Funktionsausgang CFG_RESULT der Wert 0 an.



Die Parameter können der Funktion zur Laufzeit des Applikationsprogramms zugewiesen werden.

Die Konfigurationsdaten für das E/A-Modul werden nur im Zustand „Pre-Operational“ übernommen. Erfolgt die Konfiguration im Zustand „Operational“, werden die neuen Einstellungen erst nach dem Umschalten in den Modus „Pre-Operational“ -> „Operational“ gültig.

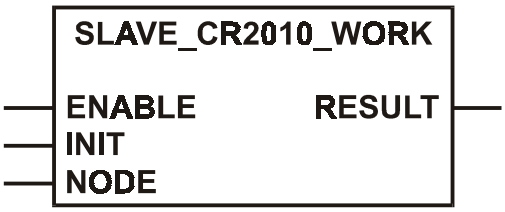
Die Funktion muß einmalig zur Initialisierung der Steuerung mit READ = TRUE aufgerufen werden. Ohne diesen Aufruf kann die Steuerung die E/A-Daten des Moduls nicht verarbeiten.

Die Funktion SLAVE_CR2010_CONFIG entspricht genau der Funktion SLAVE_8_CONFIG. Für neue Applikationen sollte nur noch die Funktion mit der Artikelnummerangabe genutzt werden.

Funktion **SLAVE_CR2010_WORK**

Library **CSxxxx.LIB**

Funktionssymbol



Zweck Schreibt bzw. liest die E/A-Daten eines Moduls.

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet
INIT	BOOL	TRUE: Funktion wird initialisiert FALSE: Daten werden aktualisiert
NODE	BYTE	Knotennummer

Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BOOL	Die Funktion wurde erfolgreich durchgeführt

Beschreibung

Über die Funktion SLAVE_CR2010_WORK werden die E/A-Daten für die ifm-eigenen 8-kanaligen Module aktualisiert. Dazu muß diese Funktion für jeden Knoten einmal im Programmzyklus aufgerufen werden. Im ersten Programmzyklus muß zusätzlich noch der Eingang INIT einmalig auf TRUE gesetzt werden. Damit werden dem Betriebssystem der Steuerung die konfigurierten Module bekannt gemacht.

Die Funktion SLAVE_CR2010_WORK entspricht genau der Funktion SLAVE_8_WORK. Für neue Applikationen sollte nur noch die Funktion mit der Artikelnummerangabe genutzt werden.

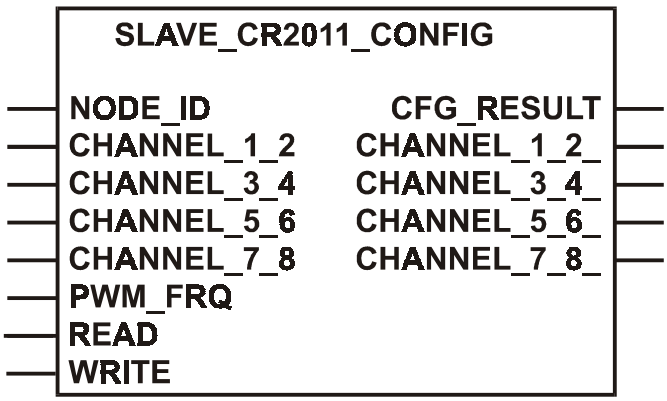
Funktion

SLAVE_CR2011_CONFIG

Library

CSxxxx.LIB

Funktionssymbol



Zweck

Parametriert bzw. liest die Konfiguration eines Ausgangs-Moduls.

Parameter

Funktionseingänge

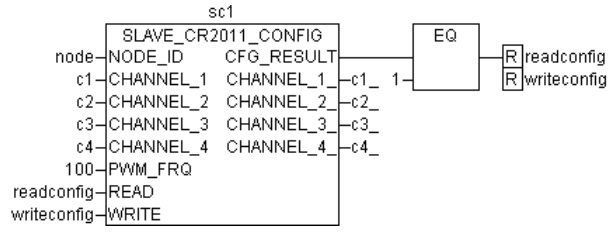
Name	Datentyp	Beschreibung
NODE_ID	BYTE	Knotennummer
CHANNEL_1_2	BYTE	Konfigurationsparameter für Kanal 3/4 0 = AUS, 2 = binär Ausgang 4 = analog Ausg. (PWM)., 5 = analog Ausg. (stromgeregelt)
CHANNEL_3_4	BYTE	Konfigurationsparameter für Kanal 3/4 0 = AUS, 2 = binär Ausgang 4 = analog Ausg. (PWM)., 5 = analog Ausg. (stromgeregelt)
CHANNEL_5_6	BYTE	Konfigurationsparameter für Kanal 3/4 0 = AUS, 2 = binär Ausgang 4 = analog Ausg. (PWM)., 5 = analog Ausg. (stromgeregelt)
CHANNEL_7_8	BYTE	Konfigurationsparameter für Kanal 3/4 0 = AUS, 2 = binär Ausgang 4 = analog Ausg. (PWM)., 5 = analog Ausg. (stromgeregelt)
PWM_FRQ	BYTE	PWM-Frequenz in Hz (20 ... 200 Hz)
READ	BOOL	aktuelle Modulkonf. lesen
WRITE	BOOL	aktuelle Modulkonf. schreiben

Funktionsausgänge

Name	Datentyp	Beschreibung
CFG_RESULT	BYTE	1 = Konfiguration wurde erfolgreich gelesen oder geschrieben 2 = Konfiguration noch nicht gelesen oder geschrieben
CHANNEL_1_2_	BYTE	aktuelle Konfigurationsparameter für Kanal 1/2
CHANNEL_3_4_	BYTE	aktuelle Konfigurationsparameter für Kanal 3/4
CHANNEL_5_6_	BYTE	aktuelle Konfigurationsparameter für Kanal 5/6
CHANNEL_7_8_	BYTE	aktuelle Konfigurationsparameter für Kanal 7/8

Beschreibung

Die Funktion SLAVE_CR2011_CONFIG setzt oder liest die Konfigurationsparameter der ifm-eigenen 8-kanaligen Ausgangsmodule. Über die Parameter wird die gewünschte Konfiguration eingestellt. Zur Kontrolle des Funktionsablaufes sollten die Eingänge READ bzw. WRITE solange gesetzt bleiben, bis am Funktionsausgang CFG_RESULT der Wert 1 ansteht. Sind die Daten noch nicht aktuell bzw. aktualisiert worden, oder können diese nicht gelesen oder geschrieben werden, steht am Funktionsausgang CFG_RESULT der Wert 0 an.



Die Parameter können der Funktion zur Laufzeit des Applikationsprogramms zugewiesen werden.

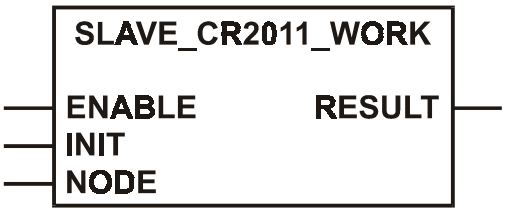
Die Konfigurationsdaten für das E/A-Modul werden nur im Zustand „Pre-Operational“ übernommen. Erfolgt die Konfiguration im Zustand „Operational“, werden die neuen Einstellungen erst nach dem Umschalten in den Modus „Pre-Operational“ -> „Operational“ gültig.

Die Funktion muß einmalig zur Initialisierung der Steuerung mit READ = TRUE aufgerufen werden. Ohne diesen Aufruf kann die Steuerung die E/A-Daten des Moduls nicht verarbeiten.

Funktion **SLAVE_CR2011_WORK**

Library **CSxxxx.LIB**

Funktionssymbol



Zweck Schreibt bzw. liest die E/A-Daten eines Moduls

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet
INIT	BOOL	TRUE: Funktion wird initialisiert FALSE: Daten werden aktualisiert
NODE	BYTE	Knotennummer

Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BOOL	Die Funktion wurde erfolgreich durchgeführt

Beschreibung

Über die Funktion SLAVE_CR2011_WORK werden die Daten für die ifm-eigenen 8-kanaligen Ausgangsmodule aktualisiert. Dazu muß diese Funktion für jeden Knoten einmal im Programmzyklus aufgerufen werden. Im ersten Programmzyklus muß zusätzlich noch der Eingang INIT einmalig auf TRUE gesetzt werden. Damit werden dem Betriebssystem der Steuerung die konfigurierten Module bekannt gemacht.

Die Ein-/Ausgangsdaten werden wie oben beschrieben über die festgelegten Merkerbereiche übergeben. Bei den Modulen vom Typ CR2011 ist zu beachten, das die analogen Sollwerte (PWM-Wert bzw. Strom) als vorzeichenbehaftete Werte eingegeben werden. Entsprechend dem Vorzeichen wird die linke bzw. rechte Buchse des Ausgangspaares (1/2, 3/4, 5/6, 7/8) angesteuert.

Die Daten sind wie folgt organisiert:

Ein oder mehrere Ausgangspaare als Digitalausgang konfiguriert:

Adresse	Bitadresse	Bedeutung
%MW1010		1. Slave, 1. Anschluß
	%MX1010.0	1. Slave, 1. Anschluß, digitaler Ausgang
%MW1011		1. Slave, 2. Anschluß
	%MX1011.0	1. Slave, 2. Anschluß, digitaler Ausgang
%MW1012		1. Slave, 3. Anschluß
	%MX1012.0	1. Slave, 3. Anschluß, digitaler Ausgang
:	:	:
%MW1017		1. Slave, 8. Anschluß
	%MX1017.0	1. Slave, 8. Anschluß, digitaler Ausgang

Ein oder mehrere Ausgangspaare als Analogausgang (PWM) konfiguriert:

Adresse	Kanal		Bedeutung
%MW1010	1		1. Slave, Wert > 0 = Kanal 1; Wert < 0 = Kanal 2
%MW1011	2		kein Eintrag
%MW1012	3		1. Slave, Wert > 0 = Kanal 3; Wert < 0 = Kanal 4
%MW1013	4		kein Eintrag
%MW1014	5		1. Slave, Wert > 0 = Kanal 5; Wert < 0 = Kanal 6
%MW1015	6		kein Eintrag
%MW1016	7		1. Slave, Wert > 0 = Kanal 7; Wert < 0 = Kanal 8
%MW1017	8		kein Eintrag

Ein oder mehrere Ausgangspaare als Analogausgang (stromgeregelt) konfiguriert:

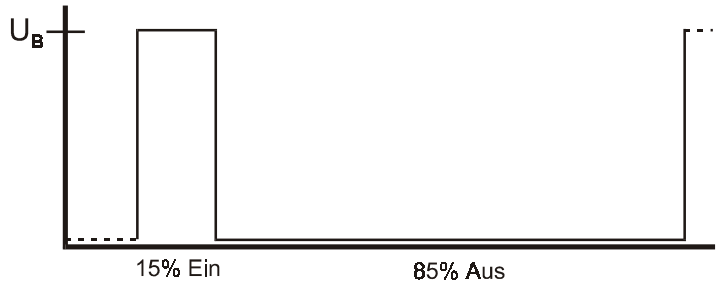
Adresse	Kanal	Bedeutung
%MW1010	1	1. Slave, Sollwert > 0 = Kanal 1; Sollwert < 0 = Kanal 2
%MW1011	2	Istwert des jeweiligen Kanals in mA
%MW1012	3	1. Slave, Sollwert > 0 = Kanal 3; Sollwert < 0 = Kanal 4
%MW1013	4	Istwert des jeweiligen Kanals in mA
%MW1014	5	1. Slave, Sollwert > 0 = Kanal 5; Sollwert < 0 = Kanal 6
%MW1015	6	Istwert des jeweiligen Kanals in mA
%MW1016	7	1. Slave, Sollwert > 0 = Kanal 7; Sollwert < 0 = Kanal 8
%MW1017	8	Istwert des jeweiligen Kanals in mA

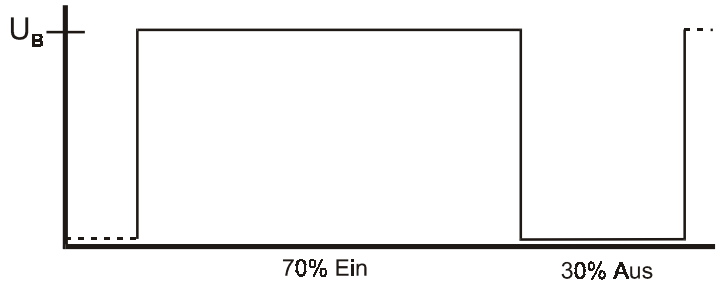
Es muß beachtet werden, daß die gewählte Konfiguration immer für zwei Ausgänge (1/2, 3/4, 5/6 bzw. 7/8) gültig ist.

6. PWM in der Steuerung CS0015

PWM steht als Abkürzung für die Puls-Weiten-Modulation. Sie wird hauptsächlich zur Ansteuerung von proportionalen Ventilen (PWM-Ventilen) und Antrieben genutzt. Ferner kann durch eine entsprechende Zusatzbeschaltung (Zubehör) eines PWM-Ausganges aus dem pulsweitenmodulierten Ausgangssignal eine analoge Ausgangsspannung erzeugt werden.

Bei dem PWM-Ausgangssignal handelt es sich um ein getaktetes Signal zwischen GND und Versorgungsspannung. Innerhalb einer festen Periode (PWM-Frequenz) wird dann das Puls-/Pausenverhältnis variiert. Durch die angeschlossene Last stellt sich je nach Puls-/Pausenverhältnis dann der entsprechende Strom ein.





Die PWM-Funktion der Steuerung CS0015 ist eine Hardware-Funktion, die vom µProzessor zur Verfügung gestellt wird. Um die 8 integrierten PWM-Ausgänge der Steuerung zu nutzen, müssen diese im Anwenderprogramm initialisiert und entsprechend dem gewünschten Ausgangssignal parametriert werden.

Als PWM-Kanal können in der Steuerung CS0015 die Ausgänge 0 ... 7 (Steckverbinder X3) genutzt werden.

PWM oder PWM100

Je nach Einsatzsatzfall und gewünschter Auflösung kann bei der Applikationsprogrammierung zwischen den Funktionen PWM und PWM100 gewählt werden. Bei Einsatz der Reglerfunktionen, wird eine hohe Genauigkeit und damit Auflösung benötigt. Daher wird in diesem Fall die mehr technische PWM-Funktion genutzt.

Soll der Aufwand bei der Implementierung gering gehalten, und werden keine hohen Anforderungen an die Genauigkeit gestellt, kann die Funktion PWM100 eingesetzt werden. Bei dieser Funktion kann die PWM-Frequenz direkt in Hz und das Puls-Pausen-Verhältnis in 1%-Schritten eingegeben werden.

PWM-Frequenz

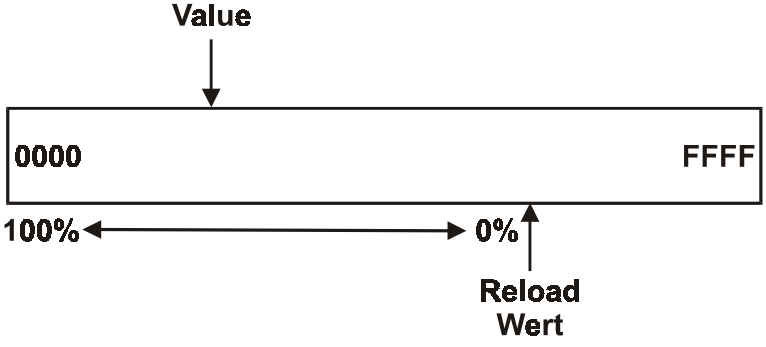
Abhängig vom Ventiltyp wird eine entsprechende PWM-Frequenz benötigt. Die PWM-Frequenz wird bei der PWM-Funktion über den Reload-Wert (Funktion PWM) oder direkt als Zahlenwert in Hz (Funktion PWM100) übergeben. Die Steuerung CS0015 hat 2 x 4 PWM-Ausgänge, die sich in ihrer Arbeits- aber nicht in ihrer Wirkungsweise unterscheiden.

Mittels eines intern ablaufenden Zählers, abgeleitet vom CPU-Takt, wird die PWM-Frequenz realisiert. Mit der Initialisierung der Funktion PWM wird dieser Zähler gestartet. Je nach PWM-Ausgangsgruppe (0 ... 3 oder 4 ... 7) zählt dieser dann von FFFF Hex rückwärts bzw von 0000 Hex aufwärts. Bei Erreichen eines übergebenen Vergleichswertes (VALUE), wird der Ausgang gesetzt. Mit Überlauf des Zähler (Zählerstandwechsel von 0000 Hex nach FFFF Hex oder von FFFF Hex nach 0000 Hex) wird der Ausgang wieder zurückgesetzt und der Vorgang neu gestartet.

Soll dieser interne Zähler nun nicht zwischen 0000 Hex und FFFF Hex laufen, kann ein anderer Presetwert (RELOAD-Wert) für den internen Zähler übergeben werden. Dadurch steigt die PWM-Frequenz. Der Vergleichswert muß dann innerhalb des nun festgelegten Bereiches liegen.

PWM-Kanäle 0 ... 3

Diese vier PWM-Kanäle bieten die größte Flexibilität bei der Parametrierung. Für jeden Kanal kann eine eigene PWM-Frequenz (RELOAD-Wert) eingestellt und zwischen den Funktionen PWM bzw. PWM100 kann frei gewählt werden.



Berechnung des RELOAD-Wertes

Der Reloadwert des internen PWM-Zählers berechnet sich in Abhängigkeit des Einganges DIV64 wie folgt:

DIV64 = 0: f_{PWM} = 10,00 MHz / Reload
DIV64 = 1: f_{PWM} = 156,25 kHz / Reload

Je nachdem ob eine hohe oder niedrige PWM-Frequenz benötigt wird, muß der Eingang DIV64 auf 0 bzw. 1 gesetzt werden. Bei PWM-Frquenzen < 152 Hz muß DIV64 auf 1 gesetzt werden, damit der Reload-Wert nicht größer als FFFF Hex wird.

Beispiel

Die PWM-Frequenz soll 200 Hz betragen.

$$\text{Reload-Wert} \Rightarrow \frac{10\ \text{MHz}}{200\ \text{Hz}} = 50000 \Rightarrow \text{C350 Hex}$$

Der zulässige Bereich des PWM-Wertes ist damit der Bereich

von 0000 Hex bis C350 Hex

Der Vergleichs-Wert bei dem der Ausgang durchschaltet, muß dann zwischen 0000 Hex und C350 Hex liegen.

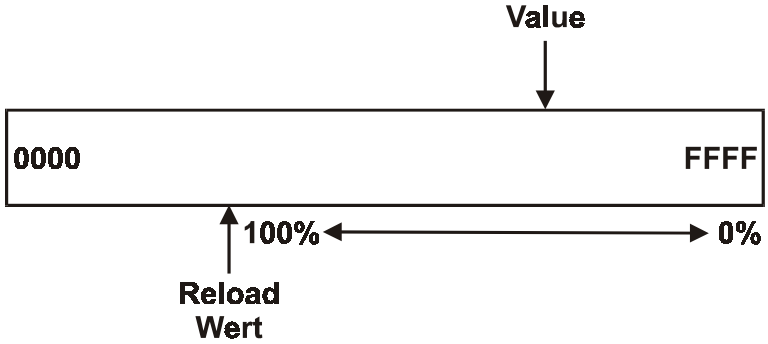
Daraus ergeben sich folgende Puls-Pausen-Verhältnisse:

Minimales Puls-Pausen-Verhältnis (0 % Ein): C350 Hex
Maximales Puls-Pausen-Verhältnis (100 % Ein): 0000 Hex

Zwischen minimaler und maximaler Ansteuerung sind 50000 Zwischenwerte (PWM-Werte) möglich.

PWM-Kanäle 4 ... 7

Diese vier PWM-Kanäle können nur auf eine gemeinsame PWM-Frequenz eingestellt werden. Bei der Programmierung dürfen die Funktionen PWM und PWM100 nicht gemischt eingesetzt werden.



Berechnung des RELOAD-Wertes

Der Reloadwert des internen PWM-Zählers berechnet sich in Abhängigkeit des Einganges DIV64 wie folgt:

DIV64 = 0: f_{PWM} = 2,50 MHz / (10000 Hex - Reload)
DIV64 = 1: f_{PWM} = 156,25 kHz / (10000 Hex - Reload)

Je nachdem ob eine hohe oder niedrige PWM-Frequenz benötigt wird, muß der Eingang DIV64 auf 0 bzw. 1 gesetzt werden. Bei PWM-Frquenzen < 39 Hz muß DIV64 auf 1 gesetzt werden, damit der Reload-Wert nicht kleiner als 0000 Hex wird.

Beispiel

Die PWM-Frequenz soll 200 Hz betragen.

$$\frac{2{,}5\ \text{MHz}}{200\ \text{Hz}} = 12500 \Rightarrow 30D4\ \text{Hex}$$

Reloadwert ⇒ 10000 Hex - 30D4 Hex = CF2C Hex

Der zulässige Bereich des PWM-Wertes ist damit der Bereich

von CF2C Hex bis FFFF Hex

Der Vergleichs-Wert bei dem der Ausgang durchschaltet, muß dann zwischen CF2C Hex und FFFF Hex liegen.

Die PWM-Frequenz ist für alle PWM-Ausgänge gleich. Die Funktionen PWM und PWM100 dürfen nicht gemischt werden.

Daraus ergeben sich folgende Puls-Pausen-Verhältnisse:

Minimales Puls-Pausen-Verhältnis (0 % Ein): FFFF Hex
Maximales Puls-Pausen-Verhältnis (100 % Ein): CF2C Hex

Zwischen minimaler und maximaler Ansteuerung sind 12500 Zwischenwerte (PWM-Werte) möglich.

PWM-Dither

Bei bestimmten Hydraulikventiltypen muß die PWM-Frequenz zusätzlich von einer sogenannten Dither-Frequenz (Zitter-Frequenz) überlagert werden. Würden diese Ventile über einen längeren Zeitraum mit einem konstanten PWM-Wert angesteuert, so könnten sie sich durch die hohen Systemtemperaturen festsetzten. Um dieses zu verhindern, wird der PWM-Wert in Abhängigkeit von der Dither-Frequenz um einen festgelegten Wert (DITHER_VALUE) vergrößert bzw. verkleinert. Die Folge ist, der konstante PWM-Wert wird von einer Schwebung mit der Dither-Frequenz und der Amplitude DITHER_VALUE überlagert. Die Dither-Frequenz wird als Verhältnis (Teiler, DITHER_DIVIDER) der PWM-Frequenz angegeben.

Die Funktion PWM_DITHER muß einmalig für jeden PWM-Ausgang initialisiert werden. Dabei kann das DELTA individuell gewählt werden. Die Dither-Frequenz kann bei den Kanälen 0 ... 3 unterschiedlich, bei den Kanälen 4 ...7 muß sie gleich sein.

Rampenfunktion

Soll der Wechsel von einem PWM-Wert zum nächsten nicht hart erfolgen (z.B. von 15 % Ein auf 70 % Ein, siehe Grafik in diesem Kapitel), kann z.B. durch Nutzung der Funktion PT1 (siehe Kapitel 9.) ein verzögerter Anstieg realisiert werden. Natürlich kann dieser Vorgang auch durch schrittweises Heraufzählen bis zum neuen Sollwert in der Applikationssoftware realisiert werden. Auf diese Wiese können dann z.B. Hydrauliksysteme im Sanftanlauf betrieben werden.

Beispielprogramm

Ein Beispielprogramm für eine Nutzung der PWM-Funktionalität der CS0015 ist auf der Programmdiskette ecolog 100*plus* gespeichert.

Die PWM-Funktion der Steuerung CS0015 ist eine vom Prozessor zur Verfügung gestellte Hardware-Funktion. Wird an einem der Ausgänge (0 ... 3 bzw. 4 ... 7) die PWM-Funktion initialisiert, bleibt diese Funktion solange gesetzt, bis an der Steuerung ein Hardware-Reset (Aus- und Einschalten der Versorgungsspannung) durchgeführt wurde.

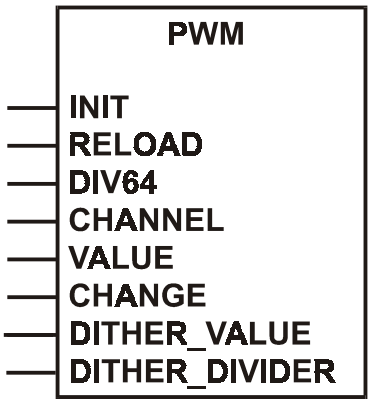
Wird bei einem der Ausgänge 0 ... 3 bzw. 4 ... 7 die PWM-Funktion aktiviert, so werden alle vier Ausgänge der Gruppe in den PWM-Modus geschaltet. D.h. diese Ausgänge stehen nicht mehr als Digitalausgänge zur Verfügung. Über die PWM-Funktion kann bei Bedarf das digitale Schaltverhalten mit PMW-maximal (100 %) und PWM-minimal (0%) nachgebildet werden.

Die maximale PWM-Frequenz ist abhängig von den eingesetzten Ausgangstransistoren. Bei der CS0015 beträgt diese 200 Hz. Bei höheren Frequenzen treten starke Ungenauigkeiten auf.

Funktion **PWM**

Library **CSxxxx.LIB**

Funktionssymbol



Zweck

Die Funktion wird zur Initialisierung und Parametrierung der PWM-Ausgänge genutzt.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
INIT	BOOL	TRUE: PWM-Ausgang wird initialisiert FALSE: PWM-Ausgang werden neue Werte zugewiesen
RELOAD	WORD	Wert zur Festlegung der PWM-Frequenz
DIV64	BOOL	CPU-Takt / 64
CHANNEL	BYTE	aktueller PWM-Kanal/Ausgang
VALUE	WORD	aktueller PWM-Wert
CHANGE	BOOL	TRUE: neuer PWM-Wert wird übernommen FALSE: geänderter PWM-Wert hat keinen Einfluß auf den Ausgang
DITHER_ VALUE	WORD	Amplitude des Dither-Wertes
DITHER_ DIVIDER	WORD	Dither-Frequenz = PWM-Frequenz/DIVIDER

Funktionsausgänge, keine

Beschreibung

Die Funktion PWM hat einen mehr technischen Hintergrund. Durch ihren Aufbau können die PWM-Werte sehr fein abgestuft ausgegeben werden. Damit eignet sich diese Funktion zum Aufbau von Reglern.

Die Funktion PWM wird einmalig für jeden Kanal in der Initialisierung des Anwenderprogramms aufgerufen. Dabei muß der Eingang INIT auf TRUE gesetzt sein. Bei der Initialisierung wird auch der Parameter RELOAD übergeben.

Der RELOAD-Wert muß für die Kanäle 4 ... 7 gleich sein. Außerdem dürfen bei diesen Kanälen die Funktionen PWM und PWM100 nicht gemischt werden.

Die PWM-Frequenz (und damit der RELAOD-Wert) ist intern auf 10 kHz begrenzt.

Je nachdem ob eine hohe oder niedrige PWM-Frequenz benötigt wird, muß der Eingang DIV64 auf 0 bzw. 1 gesetzt werden.

Während des zyklischen Programmablaufes ist INIT auf FALSE gesetzt. Die Funktion wird aufgerufen und dabei der neue PWM-Wert übergeben. Der Wert wird übernommen wenn, der Eingang CHANGE = TRUE ist.

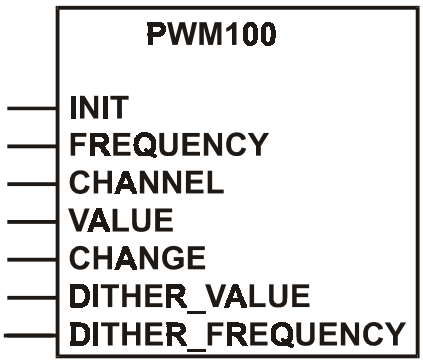
PWM_DITHER wird einmalig für jeden Kanal in der Initialisierung des Anwenderprogramms aufgerufen. Dabei muß der Eingang INIT auf TRUE gesetzt sein. Bei der Initialisierung werden der DIVIDER (Divisor) zur Bildung der Dither-Frequenz und der Wert (VALUE) übergeben.

Der DIVIDER-Wert muß für die Kanäle 4 ... 7 gleich sein. Der Wert (VALUE) kann für jeden Kanal individuell eingestellt werden.

Funktion **PWM100**

Library **CSxxxx.LIB**

Funktionssymbol



Zweck

Die Funktion wird zur Initialisierung und Parametrierung der PWM-Ausgänge genutzt.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
INIT	BOOL	TRUE: PWM100 wird initialisiert FALSE: im zyklischen Programmablauf
FREQUENCY	WORD	PWM-Frequenz in Hz
CHANNEL	BYTE	aktueller PWM-Kanal/Ausgang
VALUE	BYTE	aktueller PWM-Wert
CHANGE	BOOL	TRUE: neuer PWM-Wert wird übernommen FALSE: geänderter PWM-Wert hat keinen Einfluß auf den Ausgang
DITHER_ VALUE	BYTE	Amplitude des Dither-Wertes in Prozent
DITHER_ FREQUENCY	WORD	Dither-Frequenz in Hz

Funktionsausgänge, keine

Beschreibung

Die Funktion PWM100 ermöglicht eine einfache Anwendung der PWM-Funktionalität. Die PWM-Frequenz kann direkt in Hz und das Puls-Pausen-Verhältnis in 1%-Schritten angegeben werden. Zum Aufbau von Reglern ist diese Funktion durch die Abstufung nicht geeignet.

Die Funktion PWM100 wird einmalig für jeden Kanal in der Initialisierung des Anwenderprogramms aufgerufen. Dabei muß der Eingang INIT auf TRUE gesetzt sein. Bei der Initialisierung wird auch der Parameter FREQUENCY übergeben.

Der FREQUENCY-Wert muß für die Kanäle 4 ... 7 gleich sein. Außerdem dürfen bei diesen Kanälen die Funktionen PWM und PWM100 nicht gemischt werden.

Die PWM-Frquenz ist intern auf 10 kHz begrenzt.

Während des zyklischen Programmablaufes ist INIT auf FALSE gesetzt. Die Funktion wird aufgerufen und dabei der neue PWM-Wert übergeben. Der Wert wird übernommen wenn, der Eingang CHANGE = TRUE ist.

DITHER wird einmalig für jeden Kanal in der Initialisierung des Anwenderprogramms aufgerufen. Dabei muß der Eingang INIT auf TRUE gesetzt sein. Bei der Initialisierung wird der FREQUENCY-Wert zur Bildung der Dither-Frequenz und der Dither-Wert (VALUE) übergeben.

Der FREQUENCY-Wert muß für die Kanäle 4 ... 7 gleich sein. Der Dither-Wert (VALUE) kann für jeden Kanal individuell eingestellt werden.

7. Schnelle Eingänge

Die Steuerung CS0015 hat insgesamt 4 schnelle Eingänge, die Eingangsfrequenzen bis zu 500 Hz verarbeiten können. Neben der reinen Frequenzmessung an den Eingängen FRQ 0 ... FRQ 3, können die Eingänge ENC 0 und ENC 1 auch zur Auswertung von Drehgebern (Zählerfunktion) eingesetzt werden.

Eingang	Anschluß	Erklärung
FRQ 0/ENC 0	X1, In 0	Frequenzmessung / Drehg. 1, Kanal A
FRQ 1/ENC 0	X1, In 1	Frequenzmessung / Drehg. 1, Kanal B
FRQ 2/ENC 1	X1, In 2	Frequenzmessung / Drehg. 2, Kanal A
FRQ 3/ENC 1	X1, In 3	Frequenzmessung / Drehg. 2, Kanal B

Zur einfachen Auswertung stehen die Funktionen FREQUENCY, CYCLE und INC_ENCODER zur Verfügung.

Werden die schnellen Eingänge der Steuerung CS0015 als „normale“ Digitaleingänge eingesetzt, muß die erhöhte Empfindlichkeit gegen Störimpulse beachtet werden (z.B. Kontaktprellen bei mechanischen Kontakten). Der Standard-Digitaleingang hat eine Eingangsfrequenz von 25 Hz. Das Eingangssignal muß ggf. softwaretechnisch entprellt werden.

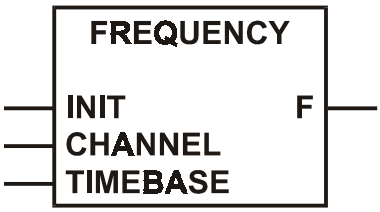
Funktion

FREQUENCY

Library

CSxxxx.LIB

Funktionssymbol



Zweck

Die Funktion FREQUENCY mißt die anstehende Signalfrequenz am angegeben Kanal.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
INIT	BOOL	TRUE: FREQUENCY wird initialisiert FALSE: im zyklischen Programm ablauf
CHANNEL	BYTE	Nummer des Eingangs (0 ... 3)
TIMEBASE	TIME	Zeitbasis

Funktionsausgang

Name	Datentyp	Beschreibung
F	WORD	Frequenz in Hz

Beschreibung

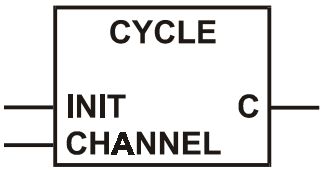
FREQUENCY mißt die Frequenz des am gewählten Kanals (CHANNEL) anstehenden Signals. Es wird dazu die positive Flanke ausgewertet. In Abhängigkeit von der Zeitbasis (TIMEBASE) können Frequenzmessungen in einem weiten Wertebereich durchgeführt werden. Hohe Frequenzen erfordern eine kurze Zeitbasis, niedrige eine entsprechend längere. Die Frequenz wird direkt in Hz ausgegeben. Bei niedrigen Frequenzen kommt es mit der Funktion FREQUENCY zu Ungenauigkeiten. Um dieses zu umgehen kann die Funktion CYCLE genutzt werden.

Für die Funktion FREQUENCY können nur die Eingänge FRQ 0 ... FRQ 3 genutzt werden.

Funktion **CYCLE**

Library **CSxxxx.LIB**

Funktionssymbol



Zweck

Die Funktion CYCLE mißt die Periodendauer (Zykluszeit) in ms am angegebenen Kanal.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
INIT	BOOL	TRUE: CYCLE wird initialisiert FALSE: im zyklischen Programm ablauf
CHANNEL	BYTE	Nummer des Eingangs (0 ... 3)

Funktionsausgang

Name	Datentyp	Beschreibung
C	WORD	Zykluszeit in ms

Beschreibung

CYCLE mißt die Zykluszeit des am ausgewählten Kanal (CHANNEL) anstehenden Signals. Es wird dazu die positive Flanke ausgewertet. Bei niedrigen Frequenzen kommt es mit der Funktion FREQUENCY zu Ungenauigkeiten. Um dieses zu umgehen kann die Funktion CYCLE genutzt werden. Die Zykluszeit wird direkt in ms ausgegeben.

Der maximale Meßbereich beträgt 65535 ms (= 15 Hz).

Für die Funktion CYCLE können nur die Eingänge FRQ 0 ... FRQ 3 genutzt werden.

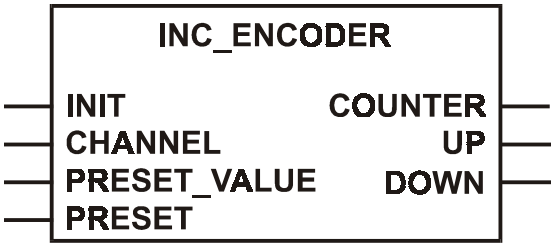
Funktion

INC_ENCODER

Library

CSxxxx.LIB

Funktionssymbol



Zweck

Vorwärts-/Rückwärts Zählerfunktion zur Auswertung von Drehgebern

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
INIT	BOOL	TRUE: INC_ENCODER wird initialisiert FALSE: im zyklischen Programmablauf
CHANNEL	BYTE	Nummer des Eingangspaares (0, 1)
PRESET_ VALUE	DINT	Voreinstellwert des Zähles
PRESET	BOOL	TRUE: Voreinstellwert wird übern. FALSE: Zähler aktiv

Funktionsausgänge

Name	Datentyp	Beschreibung
COUNTER	DINT	aktueller Zählerstand
UP	BOOL	TRUE: Zähler zählt aufwärts
DOWN	BOOL	TRUE: Zähler zählt abwärts

Beschreibung

Die Funktion INC_ENCODER ist als Vorwärts-/Rückwärtszähler ausgelegt. Immer zwei Frequenzeingänge bilden das Eingangspaar, das über die Funktion ausgewertet wird. Es können insgesamt 2 inkrementale Drehgeber angeschlossen werden.

Über den PRESET_VALUE kann der Zähler auf einen Voreinstellwert gesetzt werden. Der Wert wird übernommen wenn PRESET auf TRUE gesetzt wird. Anschließend muß PRESET wieder auf FALSE gesetzt werden, damit der Zähler wieder aktiv wird. Am Ausgang COUNTER steht der aktuelle Zählerstand an. Die Ausgänge UP und DOWN zeigen die aktuelle Zählrichtung des Zählers an. Die Ausgänge sind dann TRUE, wenn im vorangegangenen Programmzyklus der Zähler in die entsprechende Richtung gezählt hat. Bleibt der Zähler stehen, wird auch der Richtungsausgang im folgenden Programmzyklus zurückgesetzt.

8. Funktionen für das integrierte Display

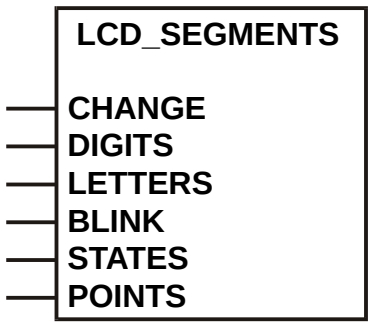
Funktion

LCD_SEGMENTS

Library

CSxxxx.LIB

Funktionssymbol



Zweck

Über die Funktion LCD_SEGMENTS können die Segmente im Display angesteuert werden.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
CHANGE	BOOL	TRUE: Die neuen Werte werden an das Display übergeben.
DIGITS	ARRAY	Entsprechend dem gesetzten Bit, wird das jeweilige Segment angesteuert. Für jede Ziffer steht ein Array-Element zur Verfügung (Array-Länge 0 ... 5 Byte).
LETTERS	ARRAY	Entsprechend dem gesetzten Bit, wird das jeweilige Segment angesteuert. Für jeden Buchstaben steht ein Array-Element zur Verfügung (Array-Länge 0 ... 2 Word).
BLINK	BYTE	Jeder Ziffer bzw. jedem Buchstaben ist genau ein Bit zugeordnent. Ist das Bit gesetzt blinkt die jeweilige Stelle.
STATES	BYTE	Ansteuerung der festen Displaysymbole. Ist das entsprechende Bit gesetzt wird das jeweilige Element angesteuert.
POINTS	BYTE	Ansteuerung der festen Dezimalpunkte. Jeder Ziffer ist ein Punkt zugeordnet. Ist das entsprechende Bit gesetzt wird der jeweilige Punkt angesteuert.

Funktionsausgänge, keine

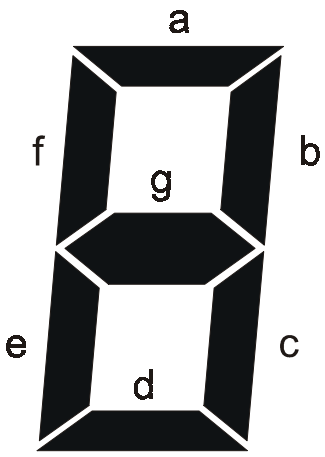
Beschreibung

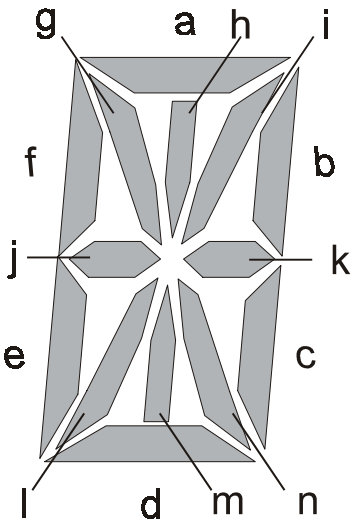
Über die Funktion LCD_SEGMENTS können die einzelnen Segmente auf dem integrierten Display angesteuert werden. Wenn das entsprechende Bit und der Eingang CHANGE auf TRUE gesetzt ist, werden die Segmente gesetzt. Ist der Eingang CHANGE auf FALSE gesetzt sind die Änderungen nicht wirksam.

Zuordnung der Segmente, Attribute, Symbole und Dezimalpunkte zu den einzelnen Bits (Bit 0 entspricht dem LSB):

Bit	Digit 1-5	Letter 1-3	Binken	Symbol	Punkt
0	e	d	Digit 1	CH1	Digit 1
1	f	e	Digit 2	CH2	Digit 2
2	d	f	Digit 3	CH3	Digit 3
3	g	l	Digit 4	CH4	Digit 4
4	a	j	Digit 5	RUN	Digit 5
5		g	Letter 1	PRG	
6	c	m	Letter 2	TST	
7	b	h	Letter 3	KEY	
8		a			
9		n			
10		k			
11		i			
12					
13		c			
14		b			
15					

Bit nicht belegt
ungültiger Bereich





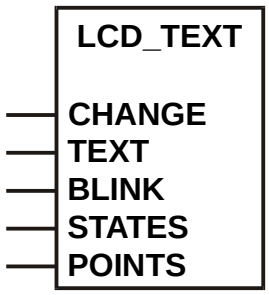
Funktion

LCD_TEXT

Library

CSxxxx.LIB

Funktionssymbol



Zweck

Über die Funktion LCD_TEXT kann direkt ein String mit maximal 8 Zeichen an das Display übergeben werden.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
CHANGE	BOOL	TRUE: Die neuen Werte werden an das Display übergeben.
TEXT	STRING	Es kann ein Text mit der maximalen Länge von 8 Zeichen ausgegen werden.
BLINK	BYTE	Jeder Ziffer bzw. jedem Buchstaben ist genau ein Bit zugeordnent. Ist das Bit gesetzt blinkt die jeweilige Stelle.
STATES	BYTE	Ansteuerung der festen Displaysymbole. Ist das entsprechende Bit gesetzt wird das jeweilige Element angesteuert.
POINTS	BYTE	Ansteuerung der festen Dezimalpunkte. Jeder Ziffer ist ein Punkt zugeordnet. Ist das entsprechende Bit gesetzt wird der jeweilige Punkt angesteuert.

Funktionsausgänge, keine

Beschreibung

Über die Funktion LCD_Text kann Text mit der maximalen Länge 8 Zeichen auf dem Display ausgegeben werden. Zahlenwerte (z.B. aus Berechnungen) müssen zuvor mit der Funktion STR in einen Text (Sting) umgewandelt werden. Entsprechend der Formatierung werden die String-Elemente auf den einzelnen Displayelementen angezeigt.

Bei der Formatierung ist zu beachten, daß prinzipbedingt Buchstaben nur unzureichend auf den 7-Segment-Elemeten angezeigt werden können.

Änderungen werden übernommen wenn der Eingang CHANGE auf TRUE gesetzt ist. Ist der Eingang CHANGE auf FALSE gesetzt sind die Änderungen nicht wirksam.

Zuordnung der Attribute, Symbole und Dezimalpunkte zu den einzelnen Bits (Bit 0 entspricht dem LSB):

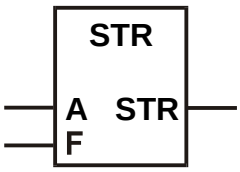
Bit	Binken	Symbol	Punkt
0	Digit 1	CH1	Digit 1
1	Digit 2	CH2	Digit 2
2	Digit 3	CH3	Digit 3
3	Digit 4	CH4	Digit 4
4	Digit 5	RUN	Digit 5
5	Letter 1	PRG	
6	Letter 2	TST	
7	Letter 3	KEY	

Bit nicht belegt

Funktion **STR**

Library **CSxxxx.LIB**

Funktionssymbol



Zweck

Die Funktion STR wandelt einen Zahlenwert in einen String um und formatiert in.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
A	DINT	Adresse der umzuwandelnden Variablen
F	STRING	Formatierungsanweisung

Funktionsausgänge

Name	Datentyp	Beschreibung
STR	STRING	formatierten String

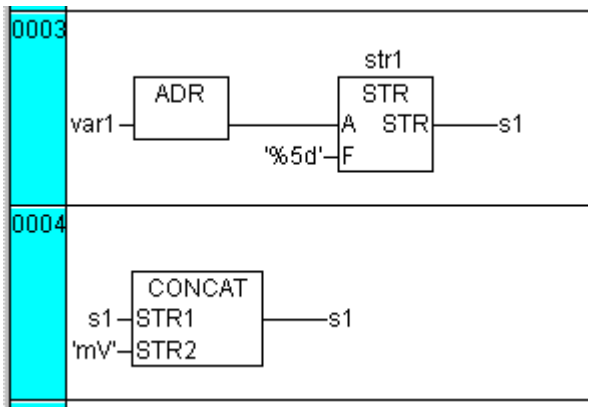
Beschreibung

STR wandelt einen Variablenwert in einen String um und formatiert ihn.

Dazu muß der Funktion die Adresse der Variablen übergeben werden. Diese wird mit dem ADR-Operator gebildet. Zusätzlich kann noch ein Fomatierungsstring entsprechend der Tabelle an die Funktion übergeben werden.

Als Ergebnis stellt die Funktion den gewandelten String zur Verfügung. Dieser kann dann z.B. direkt auf dem integrierten Display angezeigt werden.

Über Funktionen aus der Standard-Bibliothek ST167.LIB können weitere Stringbearbeitungen oder -verknüpfungen vorgenommen werden.



Übersicht der möglichen Formatierungszeichen:

Zeichen	Type der Variablen	Ausgangsformat
%d	Integer	vorzeichenbehaftete Dezimal-Zahl
%u	Integer ohne Vorzeichen	nicht vorzeichenbehaftete Dezimal-Zahl
%o	Integer ohne Vorzeichen	nicht vorzeichenbehaftete Oktal-Zahl
%x	Integer ohne Vorzeichen	nicht vorzeichenbehaftete Hexa-dezimal-Zahl (0123456789abcdef)
%X	Integer ohne Vorzeichen	nicht vorzeichenbehaftete Hexa-dezimal-Zahl (0123456789ABCDEF)
%f	Fließkomma	Fließkommazahl [-]*dddd.dddd*
%e	Fließkomma	Fließkommazahl [-]*d.dddd*e[-]*dd*
%E	Fließkomma	Fließkommazahl [-]*d..dddd*E[-]*dd*
%g	Fließkomma	Fließkommazahl
%G	Fließkomma	Fließkommazahl
%C	Zeichen	einzelnes Zeichen

9. Sonstige Funktionen

9.1. Software-Reset

Funktion

SOFTRESET

Library

CSxxxx.LIB

Funktionssymbol

Zweck

Die Funktion SOFTRESET führt einen kompletten Neustart der Steuerung aus.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet

Funktionsausgänge, keine

Beschreibung

SOFTRESET führt einen kompletten Neustart der Steuerung aus. Die Funktion kann z.B. in Verbindung mit CANopen genutzt werden, wenn ein Node-Reset ausgeführt werden soll. Das Verhalten der Steuerung nach einem SOFTRESET entspricht dem nach Aus- und Einschalten der Versorgungsspannung.

Bei einer laufenden Kommunikation muß die lange Resetphase beachtet werden, da andernfalls Guarding-fehler gemeldet werden.

9.2. Daten im Speicher sichern und lesen

Automatische Datensicherung

Die Steuerung CS0015 bietet die Möglichkeit Daten (BOOL, BYTE, WORD, DWORD) remanent im Flash-Speicher zu sichern. Bei Abfall der Versorgungsspannung, wird der Sicherungsvorgang automatisch gestartet. Voraussetzung ist, daß die Daten im Merkerbereich MW0 ... MW127 (MB0 ... MB 255) abgelegt werden.

Der Vorteil des automatischen Speicherns ist, daß auch bei einem plötzlichen Spannungsabfall oder einer Unterbrechung der Versorgungsspannung der Speichervorgang ausgelöst wird und so die aktuellen Werte der Daten gesichert werden (z.B. Zählerstände).

Kehrt die Versorgungsspannung zurück werden die gesicherten Daten durch das Betriebssystem aus dem FLASH ausgelesen und wieder in den Merkerbereich geschrieben.

Auf diesen Datenbereich kann auch über das CANopen-Objektverzeichnis (Index ab 2000 Hex) zugegriffen werden.

Manuelle Datensicherung

Neben der Möglichkeit Daten, die im Merkerbereich bis MW127 (MB255) abgelegt sind automatisch zu sichern, kann der Datenbereich zwischen MB256 ... MB1024 über einen Funktionsaufruf in das integrierte serielle EEPROM gesichert werden. Um diese Daten wieder auszulesen, muß ebenfalls ein Funktionsaufruf ausgeführt werden. Die Daten werden als gesamter Block geschrieben bzw. gelesen.

Direkter Speicherzugriff

Grundsätzlich kann der Programmierer auch auf den nicht remanenten Merkerbereich durch die entsprechenden IEC-Adressen direkt lesend und schreibend zugreifen.

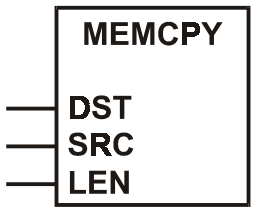
IEC Byte Adresse	IEC Word Adresse	Erklärung
%MB0 ... %MB255	%MW0 ... %MW127	remanente Daten, automatische Sicherung
%MB256 ... %MB1023	%MW128 ... %MW511	flüchtige Daten, können durch Aufruf von E2WRITE gesichert werden
%MB1025 ... %MB7935	%MW512 ... %MW3967	flüchtige Daten

Der Programmierer kann sich anhand der Speicheraufteilung (siehe Anhang 1.5.) darüber informieren, welcher Speicherbereich frei zur Verfügung steht.

Funktion **MEMCPY**

Library **CSxxxx.LIB**

Funktionssymbol



Zweck

Die Funktion MEMCPY ermöglicht das Schreiben und Lesen unterschiedlicher Datentypen direkt in den Speicher.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
DST	DWORD	Adresse der Zielvariablen
SRC	DWORD	Adresse der Quellvariablen
LEN	WORD	Anzahl der Datenbytes

Funktionsausgänge, keine

Beschreibung

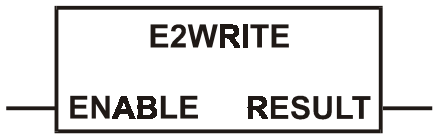
MEMCPY schreibt den Inhalt der Adresse von SRC an die Adresse DST. Dabei werden genau so viele Bytes übertragen, wie diese unter LEN angegeben wurden. Dadurch ist es auch möglich genau ein Byte einer Wortdate zu übertragen.

Die Adresse muß mit der Funktion ADR ermittelt und MEMCPY übergeben werden.

Funktion **E2WRITE**

Library **CSxxxx.LIB**

Funktionssymbol



Zweck

Die Funktion E2WRITE schreibt einen Datenblock in das serielle EEPROM.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird ausgeführt FALSE: Funktion wird nicht ausgeführt

Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BYTE	0 = Funktion ist inaktiv 1 = Funktion ist beendet 2 = Funktion arbeitet

Beschreibung

E2WRITE schreibt den Merkerbereich MW128 ... MW511 in das serielle EEPROM. Da die Abarbeitung der Funktion einige Zeit benötigt, muß die Ausführung über den Funktionsausgang RESULT überwacht werden. Wenn RESULT = 1 ist, muß der Eingang ENABLE wieder auf FALSE gesetzt werden.

Funktion **E2READ**

Library **CSxxxx.LIB**

Funktionssymbol

Zweck

Die Funktion E2READ liest einen Datenblock aus dem seriellen EEPROM aus.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird ausgeführt FALSE: Funktion wird nicht ausgeführt

Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BYTE	0 = Funktion ist inaktiv 1 = Funktion ist beendet 2 = Funktion arbeitet

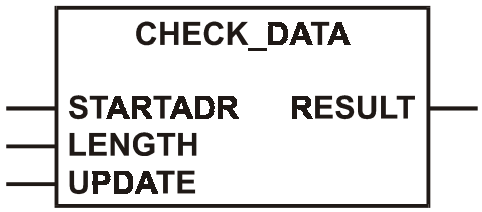
Beschreibung

E2READ liest einen Datenblock aus dem seriellen EEPROM aus und schreibt diesen in den Merkerbereich MW128 ... MW511. Da die Abarbeitung der Funktion einige Zeit benötigt, muß die Ausführung über den Funktionsausgang RESULT überwacht werden. Wenn RESULT = 1 ist, muß der Eingang ENABLE wieder auf FALSE gesetzt werden.

Funktion **CHECK_DATA**

Library **CSxxxx.LIB**

Funktionssymbol



Zweck Sichern der Daten im Anwenderdatenspeicher über einen CRC-Code.

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
STARTADR	DINT	Startadresse des überwachten Datenspeichers (Adresse ab %MW0...)
LENGTH	WORD	Länge des überwachten Datenspeichers in Byte
UPDATE	BOOL	TRUE: Datenänderungen zulässig FALSE: Datenänderungen nicht zulässig

Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BOOL	TRUE: Funktion wird ausgeführt FALSE: Funktion wird nicht ausgeführt

Beschreibung Die Funktion CHECK_DATA dient dazu in sicherheitskritischen Anwendungen einen Bereich des Datenspeichers (mögliche Adressen ab %MW0...) auf eine nicht gewollte Datenänderung zu überwachen. Die Funktion bildet dazu über den angegebenen Datenbereich eine CRC-Checksumme. Verändern sich Daten nun nicht gewollt, wird RESULT = FALSE. Das Ergebnis kann dann für weitere Aktionen (z.B. Abschalten der Ausgänge) genutzt werden.

Die Startadresse muß über den Adressoperator ADR an die Funktion übergeben werden. Zusätzlich muß noch die Anzahl der Datenbytes LENGTH (Länge ab der STARTADR) angegeben werden. Nur wenn UPDATE = TRUE ist, können Daten in dem Speicherbereich geändert werden (z.B. vom Anwenderprogramm oder tdm) und es wird keine Fehlermeldung RESULT = FALSE erzeugt.

Bei der Funktion handelt es sich um eine Sicherheitsfunktion. Dennoch wird durch Einsatz dieser Funktion die Steuerung nicht automatisch zur Sicherheitssteuerung. Als Sicherheitssteuerung kann nur eine geprüfte, zugelassene und mit einem speziellen Betriebssystem versehene Steuerung genutzt werden.

9.3. Nutzung der seriellen Schnittstelle

Funktion

SERIAL_TX

Library

CSxxxx.LIB

Funktionssymbol

Zweck

Überträgt eine Datenbyte über die serielle RS232-Schnittstelle.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Übertragung freigegeben FALSE: Übertragung gesperrt
DATA	BYTE	zu übertragende Byte-Date

Funktionsausgänge, keine

Beschreibung

SERIAL_TX überträgt die Datenbyte DATA über die serielle Schnittstelle. Mit dem Funktionseingang ENABLE kann die Übertragung freigegeben oder gesperrt werden.

Die SERIAL-Funktionen bilden die Grundlage für die Erstellung eines anwenderspezifischen Protokolls für die serielle Schnittstelle.

Grundsätzlich steht die serielle Schnittstelle dem Anwender nicht zur Verfügung, da sie für den Programmdownload und das Debugging genutzt wird. Wird das Systemmerkerbit SERIAL_MODE vom Anwender aber auf TRUE gesetzt, kann die Schnittstelle frei genutzt werden. Der Programmdownload und das Debugging ist dann **nur** noch über die **CAN-Schnittstelle** möglich.

Funktion **SERIAL_RX**

Library **CSxxxx.LIB**

Funktionssymbol

Zweck

Liest eine empfangene Datenbyte aus dem seriellen Empfangspuffer aus.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
CLEAR	BOOL	TRUE: Empfangspuffer wird gelöscht FALSE: Default-Zustand

Funktionsausgänge

Name	Datentyp	Beschreibung
RX	BYTE	empfangene Byte-Date aus dem Empfangspuffer
AVAILABLE	WORD	Anzahl der empfangenen Datenbytes
OVERFLOW	BOOL	Überlauf des Datenpuffers, Datenverlust!

Beschreibung

SERIAL_RX liest mit jedem Aufruf ein Datenbyte aus dem seriellen Empfangspuffer aus. Anschließend wird der Wert von AVAILABLE um 1 dekrementiert. Sind keine Daten mehr im Puffer ist AVAILABLE 0.

Gehen mehr als 1000 Datenbytes ein, läuft der Puffer über und es gehen Daten verloren. Dieses wird durch das Bit OVERFLOW angezeigt.

Die SERIAL-Funktionen bilden die Grundlage für die Erstellung eines anwenderspezifischen Protokolls für die serielle Schittstelle.

Grundsätzlich steht die serielle Schnittstelle dem Anwender nicht zur Verfügung, da sie für den Programmdownload und das Debugging genutzt wird. Wird das Systemmerkerbit SERIAL_MODE vom Anwender aber auf TRUE gesetzt, kann die Schnittstelle frei genutzt werden. Der Programmdownload und das Debugging ist dann **nur** noch über die **CAN-Schnittstelle** möglich.

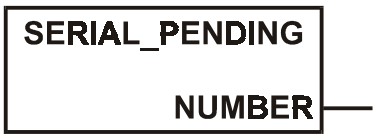
Funktion

SERIAL_PENDING

Library

CSxxxx.LIB

Funktionssymbol



Zweck

Die Funktion ermittelt die Anzahl der im seriellen Empfangspuffer gespeicherten Datenbytes.

Parameter

Funktionseingänge, keine

Funktionsausgang

Name	Datentyp	Beschreibung
NUMBER	WORD	Anzahl der empfangenen Datenbytes

Beschreibung

SERIAL_PENDING ermittelt die Anzahl der empfangenen Datenbytes im Empfangspuffer. Im Gegensatz zur Funktion SERIAL_RX bleibt der Inhalt des Puffers nach Aufruf dieser Funktion unverändert.

Die SERIAL-Funktionen bilden die Grundlage für die Erstellung eines anwenderspezifischen Protokolls für die serielle Schnittstelle.

Grundsätzlich steht die serielle Schnittstelle dem Anwender nicht zur Verfügung, da sie für den Programmdownload und das Debugging genutzt wird. Wird das Systemmerkerbit SERIAL_MODE vom Anwender aber auf TRUE gesetzt, kann die Schnittstelle frei genutzt werden. Der Programmdownload und das Debugging ist dann **nur** noch über die **CAN-Schnittstelle** möglich.

9.4. Auslesen der Systemzeit

Funktion **TIMER_READ**

Library **CSxxxx.LIB**

Funktionssymbol

Zweck Es wird die aktuelle Systemzeit in Sekunden ausgelesen.

Parameter Funktionseingänge, keine

Funktionsausgang

Name	Datentyp	Beschreibung
T	TIME	Aktuelle Systemzeit in Sekunden

Beschreibung Mit Anlegen der Versorgungsspannung wird geräteintern ein Zeittakt gebildet, und in einem Register aufwärts gezählt. Dieses Register kann mittels des Funktionsaufrufes ausgelesen werden und z.B. zur Zeitmessung genutzt werden.

Der Systemtimer läuft maximal bis 10 m 55 s 350 ms und startet anschließend wieder bei 0.

Funktion **TIMER_US_READ**

Library **CSxxxx.LIB**

Funktionssymbol

Zweck Es wird die aktuelle Systemzeit in µSekunden ausgelesen.

Parameter Funktionseingänge, keine

Funktionsausgang

Name	Datentyp	Beschreibung
TIME_US	DWORD	Aktuelle Systemzeit in µSekunden

Beschreibung Mit Anlegen der Versorgungsspannung wird geräteintern ein Zeittakt gebildet, und in einem Register aufwärts gezählt. Dieses Register kann mittels des Funktionsaufrufes ausgelesen werden und z.B. zur Zeitmessung genutzt werden.

Der Systemtimer läuft maximal bis zum Zählerwert 4294967295 (µs) und startet anschießend wieder bei 0.

9.5. Variablenbearbeitung

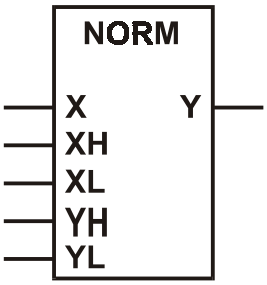
Funktion

NORM

Library

CSxxxx.LIB

Funktionssymbol



Zweck

Normiert einen Wert innerhalb festgelegter Grenzen auf einen Wert mit neuen Grenzen.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
X	WORD	Ausgangswert
XH	WORD	untere Grenze des Eingangswertebereich
XL	WORD	obere Grenze des Eingangswertebereich
YH	WORD	untere Grenze des Ausgangswertebereich
YL	WORD	obere Grenze des Ausgangswertebereich

Funktionsausgang

Name	Datentyp	Beschreibung
Y	WORD	normierte Date vom Typ BYTE

Beschreibung

Die Funktion NORM normiert einen Wert vom Typ WORD, der innerhalb der Grenzen XH und XL liegt, auf einen Ausgangswert innerhalb der Grenzen YH und YL. Diese Funktion wird z.B. bei der Erzeugung von PWM-Werten aus analogen Eingangsgrößen genutzt.

Beispiel

nicht normierter Wert:	**50**
unterer Grenzwert Eingang:	0
oberer Grenzwert Eingang:	100
unterer Grenzwert Ausgang:	0
oberer Grenzwert Ausgang:	2000
normierter Wert:	**1000**

Bedingt durch die Rundungsfehler bei hexadezimalen Zahlen können Abweichungen beim normierten Wert um 1 auftreten. Werden die Grenzen (XH/XL oderYH/YL) invertiert angegeben, erfolgt auch die Normierung invertiert.

9.6. Echtzeitverarbeitung

Funktion

SET_INTERRUPT_1MS

Library

CSxxxx.LIB

Funktionssymbol

Zweck

Echtzeitverarbeitung von Programmteilen im 1 ms Zyklus.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Datenänderungen zulässig FALSE: Datenänderungen nicht zulässig

Funktionsausgänge, keine

Beschreibung

In klassischen SPS ist die Zykluszeit das Maß der Dinge für Echtzeitbetrachtungen. Gegenüber kundenspezifischen Steuerungen ist die SPS damit im Nachteil. Auch ein „Echtzeit-Betriebssystem“ ändert nichts an dieser Tatsache, wenn das gesamte Applikationsprogramm in einer einzigen Task läuft.

Ein Lösungsansatz ist, die Zykluszeit kurzzuhalten. Dieser Weg führt oft dazu, die Applikation auf mehrere Steuerungszyklen zu verteilen. Die Programmierung wird dadurch unübersichtlich und schwierig.

Eine andere Möglichkeit besteht darin, einen bestimmten Programmteil in festen Zeitabständen (hier 1 ms) unabhängig vom Steuerungszyklus aufzurufen.

Der zeitkritische Teil der Applikation wird vom Anwender in einen Baustein vom Type PROGRAMM (PRG) zusammengefaßt. Dieser Baustein wird zur 1 ms-Interrupt-Routine deklariert, indem einmalig (zur Initialisierungzeit) der Funktionsbaustein SET_INTERRUPT_1MS aufgerufen wird. Das hat zur Folge, daß dieser Programmbaustein jede Millisekunde abgearbeitet wird. Damit er nicht auch noch zyklisch aufgerufen wird, sollte er (mit Ausnahme des Initialisierungsaufrufes) im Zyklus übersprungen werden. Werden Ein- und Ausgänge in diesem Programmteil genutzt, werden diese ebenfalls im 1 ms-Takt gelesen bzw. beschrieben. Innerhalb des Programmbausteins können also alle zeitkritischen Ereignisse bearbeitet werden, indem Eingänge oder globale Variablen verknüpft und Ausgänge beschrieben werden. So können auch Zeitglieder genauer überwacht werden, als dies in „normalen“ Zyklus möglich ist.

Zur gleichen Zeit darf nur ein Timer-Interrupt Baustein aktiv sein. Es kann aber je nach Programmzustand im laufenden Programm auch auf einen anderen Interrupt-Baustein gewechselt werden.

Der Zeitbedarf muß kurz gehalten werden! Aus diesem Grunde sollten auch keine Berechnungen, Gleitkomma-Arithmetik bzw. Regler-Funktionen in diesen Baustein eingesetzt werden.

Wichtig:

Die Eindeutigkeit der Ein- und Ausgänge im Zyklus wird durch die Interruptroutine aufgehoben. Deshalb kann nur ein Teil jede Millisekunde bedient werden.

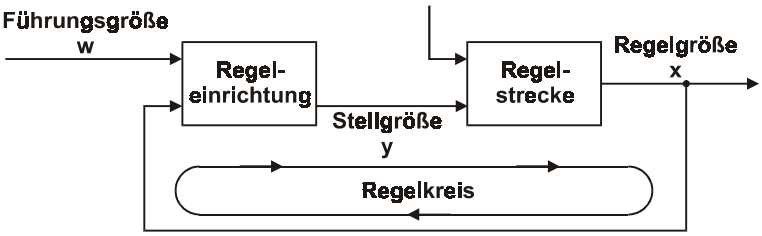
Eingänge: %IX0.00 ... %IX0.07
Ausgänge: %QX0.00 ... %QX0.07

Alle anderen Ein- und Ausgänge werden wie üblich einmalig im Zykus bearbeitet.

Auch globale Variablen verlieren ihre Eindeutigkeit, wenn auf sie quasi gleichzeitig im Zyklus und durch die Interruptroutine zugegriffen wird. Insbesondere größere Datentypen (z.B. DINT) sind von dieser Problematik betroffen.

10. Regler-Funktionen

Die Regelung ist ein Vorgang, bei dem die zu regelnde Größe (Regelgröße x) fortlaufend erfaßt und mit der Führungsgröße verglichen wird. In Abhängigkeit vom Ergebnis dieses Vergleiches wird zur Angleichung an die Führungsgröße die Regelgröße beeinflußt.

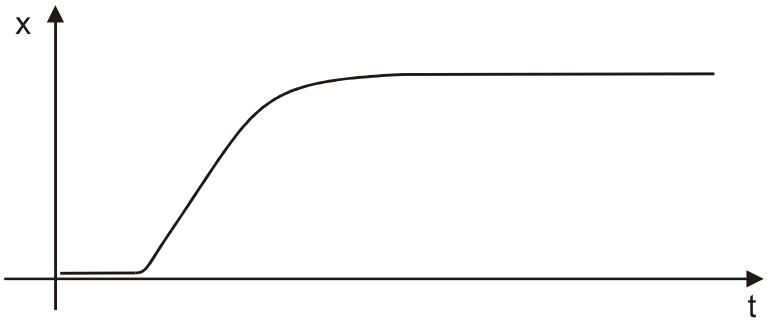


Die Auswahl einer geeigneten Regeleinrichtung und deren optimale Einstellung, setzt genaue Angaben über das Beharrungsverhalten und das dynamische Verhalten der Regelstrecke voraus. In den meisten Fällen können diese Kennwerte aber nur experimentell ermittelt werden und sind kaum beeinflußbar.

Man kann drei Typen von Regelstecken unterscheiden:

Regelstrecken mit Ausgleich

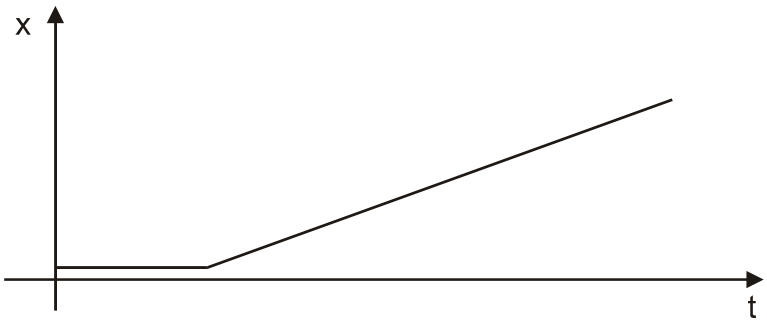
Bei einer Regelstrecke mit Ausgleich, strebt die Regelgröße x nach einer bestimmten Stellgrößenänderung einem neuen Endwert (Beharrungszustand) zu. Entscheidend ist bei diesen Regelstrecken die Verstärkung (Übertragungsbeiwert K_S). Je kleiner die Verstärkung ist, um so besser läßt sich die Strecke regeln.



Man bezeichnet diese Regelstrecken als P(roportionale)-Systeme.

Regelstrecken ohne Ausgleich

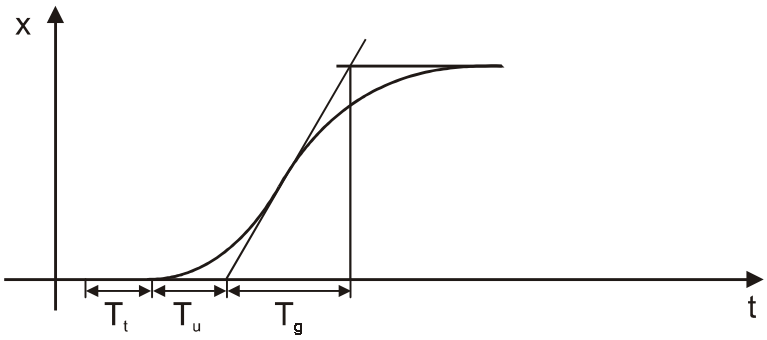
Regelstrecken mit einem Verstärkungsfaktor gegen unendlich, werden als Regelstrecken ohne Ausgleich bezeichnet. Dieses ist meistens auf ein integrierendes Verhalten zurückzuführen. Diese hat zur Folge, daß nach der Änderung der Stellgröße oder durch Einfluß einer Störgröße, die Regelgröße stetig wächst. Durch diese Verhalten erreicht sie nie einen Endwert.



Man bezeichnet diese Regelstrecken als I(ntegrale)-Systeme.

Regelstrecken mit Verzögerung

Die meisten Regelstrecken entsprechen der Reihenschaltung von P-Systemen (Strecken mit Ausgleich) und einem oder mehreren T_1-Systemen (Strecken mit Trägheit). Eine Regelstrecke 1. Ordnung entsteht z.B. durch die Reihenschaltung einer Drosselstelle und einem dahinter liegenden Speicher.



Bei Regelstrecken mit Totzeit reagiert die Regelgröße erst nach Ablauf der Totzeit T_t auf eine Veränderung der Stellgröße. Die Totzeit T_t bzw. die Summe aus T_t + T_u ist das Maß für die Regelbarkeit der Strecke. Die Regelbarkeit einer Strecke ist um so besser, je größer das Verhältnis T_g / T_u ist.

Die Regler, die in die Bibliothek integriert sind, stellen eine Zusammenfassung die vorgestellten Grundfunktionen dar. Welche Funktionen zum Einsatz kommen und wie sie kombiniert werden, hängt von der jeweiligen Regelstrecke ab.

10.1. Einstellregel für einen Regler

Für Regelstrecken, deren Zeitkonstanten nicht bekannt sind, ist das Einstellverfahren nach Ziegler und Nickols im geschlossenen Regelkreis vorteilhaft.

Einstellregel

Die Regeleinrichtung wird zunächst als eine reine P-Regeleinrichtung betrieben. Dazu wird die Vorhaltezeit TV auf 0 und die Nachstellzeit TN auf einen sehr großen Wert (ideal auf ∞) für eine träge Strecke eingestellt. Bei einer schnellen Regelstrecke sollte ein kleines TN gewählt werden. Der Proportionalbeiwert KP wird anschließend solange vergrößert, bis die Regel- und die Stellabweichung bei KP = $KP_{kritisch}$ Dauerschwingungen mit konstanter Amplitude ausführen. Es ist damit die Stabilitätsgrenze erreicht. Anschließend muß die Periodendauer $T_{kritisch}$ der Dauerschwingung ermittelt werden. Nur bei Bedarf einen D-Anteil hinzufügen. TV sollte ca. 2 - 10 mal kleiner als TN und KP = KD gewählt werden.

Idealisiert ist die Regelstrecke wie folgt einzustellen:

Regel-einrichtung	KP = KD	TN	TV
P	2,0 * $KP_{kritisch}$	-	-
PI	2,2 * $KP_{kritisch}$	0,83 * $T_{kritisch}$	-
PID	1,7 * $KP_{kritisch}$	0,50 * $T_{kritisch}$	0,125 * $T_{kritisch}$

Bei diesem Einstellverfahren ist zu beachten, daß die Regelstrecke durch die auftretenden Schwingungen keinen Schaden nimmt. **Bei empfindlichen Regelstrecken darf KP nur bis zu einem Wert erhöht werden, bei dem noch sicher keine Schwingungen auftreten.**

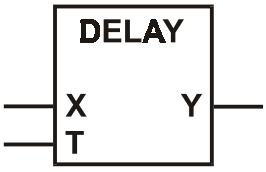
Dämpfung von Überschwingungen

Um Überschwingungen zu dämpfen, kann die PT1(Tiefpass)-Funktion eingesetzt werden. Dazu wird der Sollwert XS, bevor er der Reglerfunktion zugeführt wird, durch das PT1-Glied gedämpft. Als Einstellgröße für T1 gilt, daß dieser Wert ca. 4 - 5 mal größer als TN (des PID- oder GLR-Reglers) sein sollte.

Funktion **DELAY**

Library **CSxxxx.LIB**

Funktionssymbol



Zweck

Verzögert die Ausgabe des Eingangswertes um die Zeit T (Totzeit-Glied).

Parameter

Funktionseingang

Name	Datentyp	Beschreibung
X	WORD	Eingangswert
T	TIME	Verzögerungszeit (Totzeit)

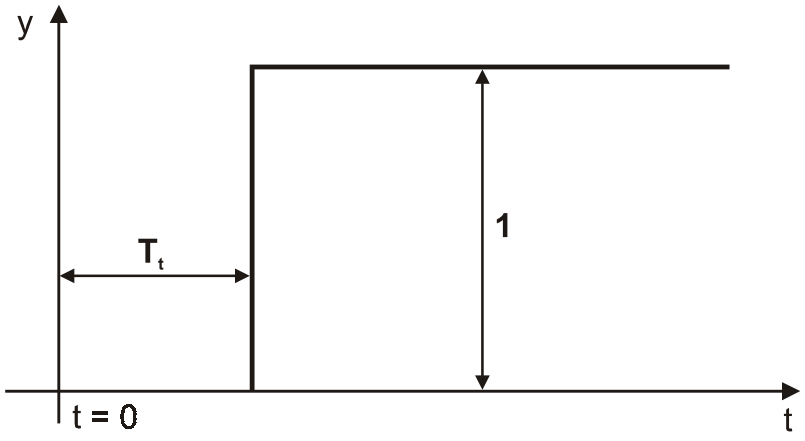
Funktionsausgänge

Name	Datentyp	Beschreibung
Y	WORD	Eingangswert verzögert um die Zeit T

Beschreibung

Die Funktion DELAY wird genutzt um einen Eingangswert um die Zeit T zu verzögern.

Die Ausgangsgröße y hat folgenden zeitlichen Verlauf.

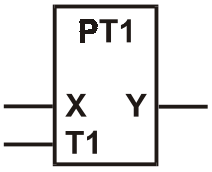


Damit die Funktion einwandfrei arbeitet muß sie in jedem Zyklus aufgerufen werden.

Funktion **PT1**

Library **CSxxxx.LIB**

Funktionssymbol



Zweck Regelstrecke mit Verzögerung 1. Ordnung.

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
X	WORD	Eingangswert
T1	TIME	Verzögerungszeit (Zeitkonstante)

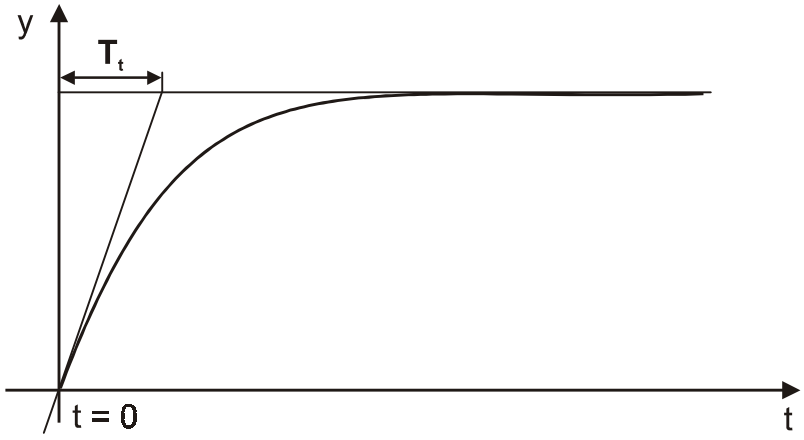
Funktionsausgang

Name	Datentyp	Beschreibung
Y	WORD	Ausgangsvariable

Beschreibung

Bei der Funktion PT1 handelt es sich um eine proportionale Regelstrecke mit Verzögerung. Sie wird z.B. zur Bildung von Rampen bei Einsatz der PWM-Funktionen genutzt.

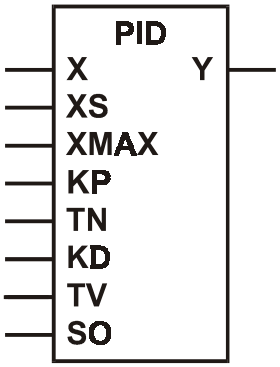
Die Ausgangsvariable y des Tiefpaßfilters hat folgenden zeitlichen Verlauf (Einheitssprungfunktion):



Funktion **PID**

Library **CSxxxx.LIB**

Funktionssymbol



Zweck PID-Regler

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
X	WORD	Istwert
XS	WORD	Sollwert
XMAX	WORD	Maximalwert des Sollwertes
KP	BYTE	Konstante des P-Anteils (/10)
TN	TIME	Nachstellzeit (I-Anteil)
KD	BYTE	Proportionalanteil des D-Anteils (/10)
TV	TIME	Vorhaltezeit (D-Anteil)
SO	BOOL	Selbstoptimierung

Funktionsausgang

Name	Datentyp	Beschreibung
Y	WORD	Stellgröße

Beschreibung

Die Änderung der Stellgröße eines PID-Regles setzt sich aus einem proportionalen, interalen und differentialen Anteil zusammen. Die Stellgröße ändert sich zunächst um einen von der Änderungsgeschwindigkeit der Eingangsgröße abhängigen Betrag (D-Anteil). Nach Ablauf der Vorhaltezeit TV geht die Stellgröße auf den dem Proportionalbereich entsprechenden Wert zurück und ändert sich dann entsprechend der Nachstellzeit TN.

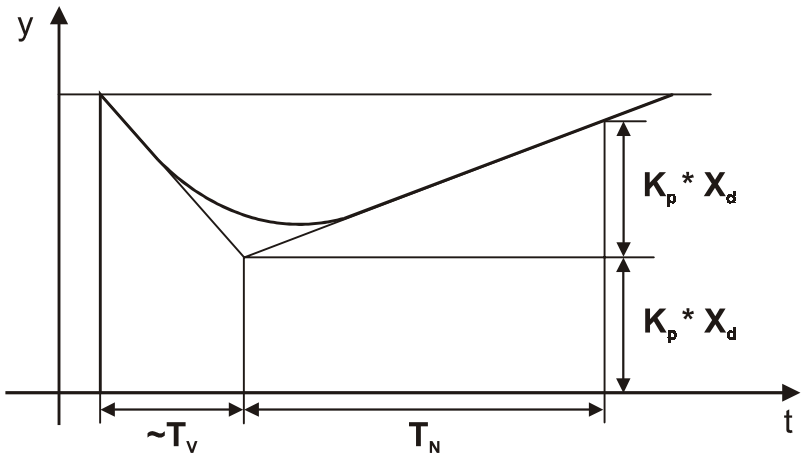
Die am Funktionseingang KP und KD eingegebenen Werte werden intern durch 10 geteilt. Damit kann eine feinere Abstufung erreicht werden (z.B: KP = 17 entspr. 1,7)

Der Stellwert Y ist bereits auf die PWM-Funktion normiert (RELOAD-Wert = 65535). Dabei ist die umgekehrte Logik zu beachten (65535 = minimaler Wert, 0 = maximaler Wert).

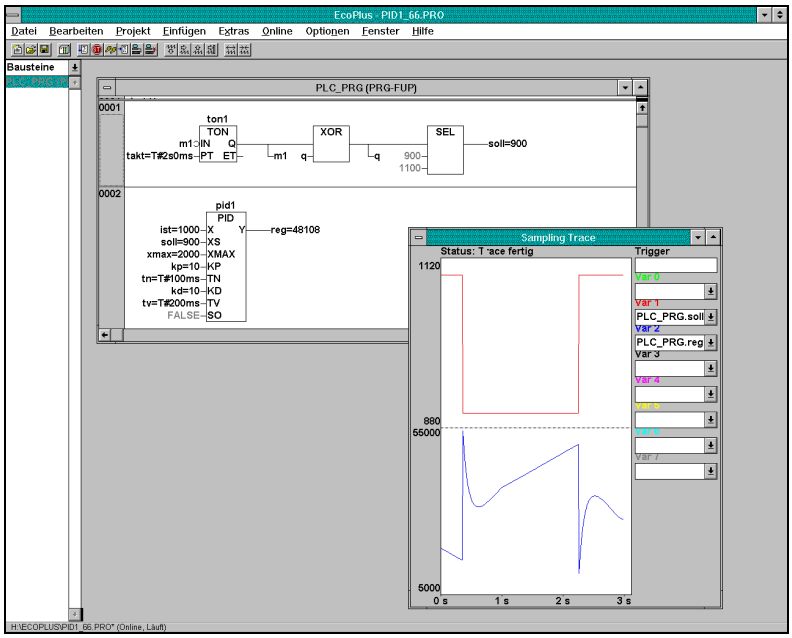
Wenn X > XS, dann wird der Stellwert erhöht.
Wenn X < XS, dann wird der Stellwert reduziert.

Eine Führungsgröße wird intern zum Stellwert hinzuaddiert:
Y = Y + 65536 - (XS / XMAX x 65536).

Die Stellgröße y hat folgenden zeitlichen Verlauf.



Typische Sprungantwort eines PID-Reglers



Einstellempfehlung:

- TN gemäss des Zeitverhaltens der Strecke wählen (schnelle Strecke = kleines TN, träge Strecke = großes TN)
- KP langsam, schrittweise erhöhen bis zu einem Wert, bei dem noch sicher kein Schwingen auftritt.
- TN evt. nachjustieren
- Nur bei Bedarf D-Anteil hinzufügen: TV ca. 2 - 10 mal kleiner als TN wählen. KD ungefähr gleich groß wie KP wählen.

Man muß beachten, daß die maximale Regelabweichung +/- 127 beträgt. Für ein gutes Regelverhalten sollte dieser Bereich einerseits nicht überschritten, andererseits aber möglichst ausgenutzt werden.

Durch den Funktionseingang SO (Selbstoptimierung) werden die Regeleigenschaften deutlich verbessert. Voraussetzung, daß die gewünschten Eigenschaften erreicht werden ist:

- Der Regler wird mit I-Anteil betrieben (TN ≥ 50 ms)
- Die Parameter KP und insbesondere TN sind bereits gut an die reale Regelstrecke angepaßt.
- Der Regelbereich (X - XS) von +/- 127 wird ausgenutzt (bei Bedarf durch Multiplikation von X, XS und XMAX den Regelbereich vergrößern).

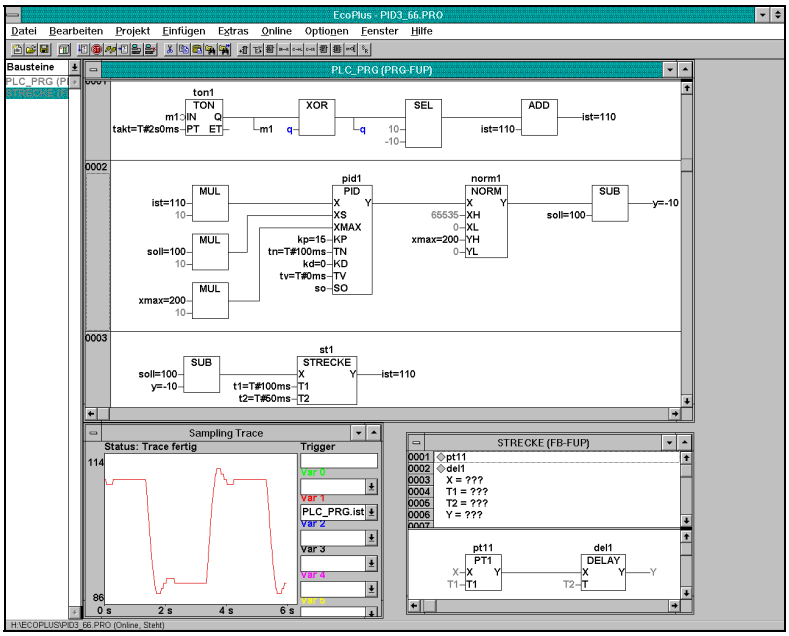
Nach Abschluss der Parametereinstellungen kann SO = TRUE gesetzt werden. Die Regeleigenschaften werden dann merklich verbessert. Insbesondere Überschwingungen werden reduziert.

Beispiel

PI-Regler an einer simulierten Strecke.

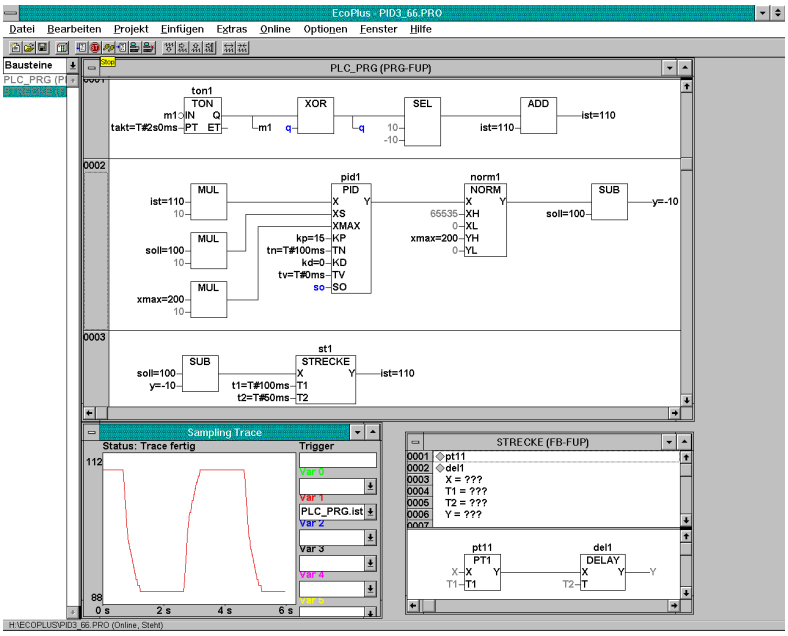
SO = FALSE.

Das Beispiel zeigt das Überschwinger auftreten, außerdem tritt auch eine Regelbereichsspreizung auf. Bedingt durch die kleine Regelgrösse ist das Signal 'stufig'.



SO = TRUE

Es treten keine Überschwinger mehr auf.



Beispiel

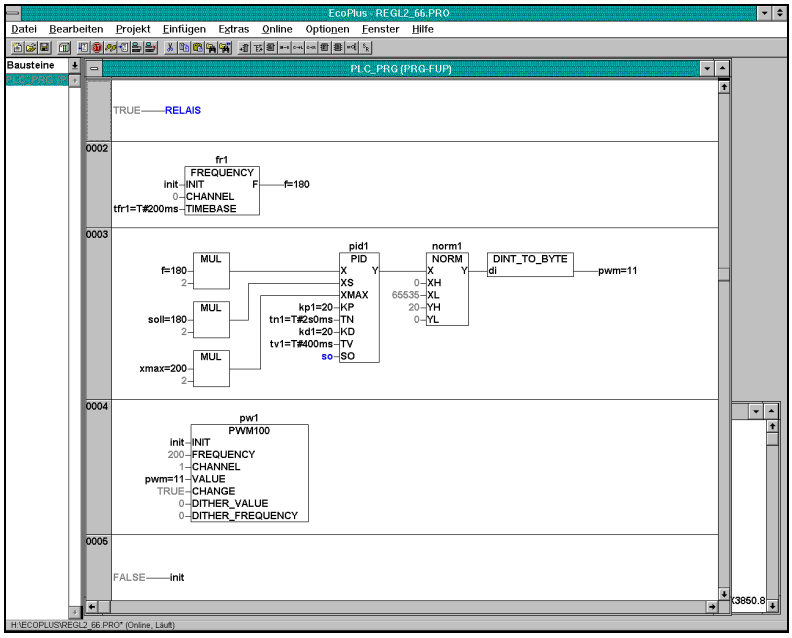
Drehzahlregelung mit PID-Regeler

Merkmale:

- 2-fache Regelbereichsspreizung
- Selbst-Optimierung
- Anpassung des Reglerausgangs Y an einen PWM-Funktionsbaustein.
- TN wurde an das relativ träge Verhalten der Strecke (Schwungmasse!) angepaßt.
- Trotz des D-Anteils ist das Überschwingen relativ gering.

Besonderheit

Der Motor des Beispiels erreicht bereits mit 20 % PWM seine maximale Drehzahl. Der Funktionsbaustein NORM berücksichtigt dies.



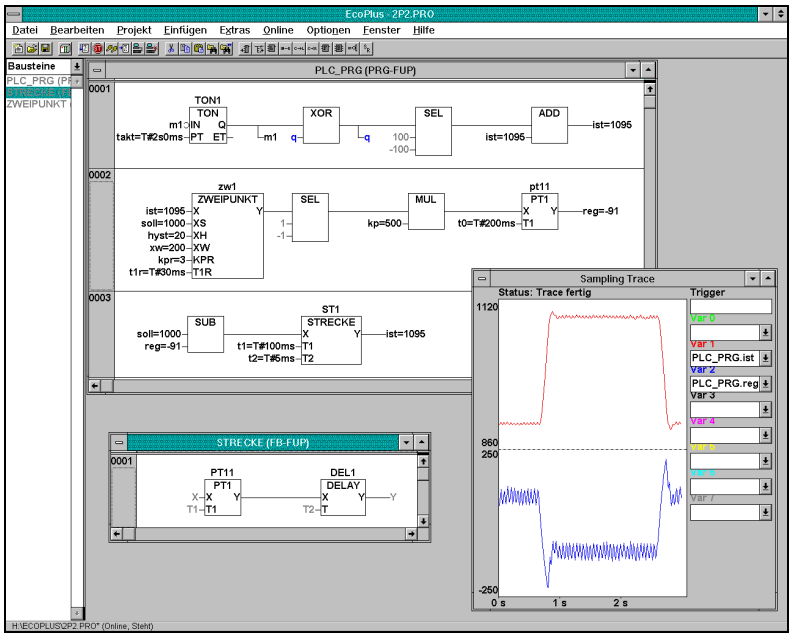
Beispiel

P-Regler

Dieser P-Regler wird aus einem 2-Punkt-Regler mit PT1-Rückführung und nachgeschaltetem PT1-Glied gebildet.
Die Regelstrecke wird simuliert.

Dieser Regler zeichnet sich durch besondere Robustheit aus und ist damit für schwierige Strecken geeignet.

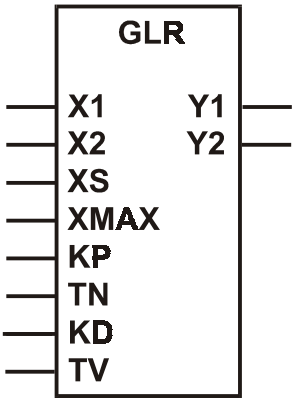
Man beachte die gewollte Eigenschwingung des Reglers, hervorgerufen durch seine interne Rückführung. Das zunächst grobe Schaltverhalten des 2-Punkt-Reglers wird verfeinert und die Schaltfrequenz nimmt zu.



Funktion **GLR**

Library **CSxxxx.LIB**

Funktionssymbol



Zweck Gleichlauf-Regler

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
X1	WORD	Istwert Kanal 1
X2	WORD	Istwert Kanal 2
XS	WORD	Sollwert = Führungsgröße
XMAX	WORD	Maximalwert des Sollwertes
KP	BYTE	Konstante des P-Anteils (/10)
TN	TIME	Nachstellzeit (I-Anteil)
KD	BYTE	Propottionalanteil des D-Anteils (/10)
TV	TIME	Vorhaltezeit (D-Anteil)

Funktionsausgänge

Name	Datentyp	Beschreibung
Y1	WORD	Stellgröße Kanal 1
Y2	WORD	Stellgröße Kanal 2

Beschreibung

Bei dem Gleichlaufregler handelt es sich um einen Regler mit PID-Verhalten.

Die am Funktionseingang KP und KD eingegebenen Werte werden intern durch 10 geteilt. Damit kann eine feinere Abstufung erreicht werden (z.B: KP = 17 entspr. 1,7).

Die Stellgrößen Y1 und Y2 sind bereits auf die PWM-Funktion normiert (RELOAD-Wert = 65535). Dabei ist die umgekerte Logik zu beachten (65535 = minimaler Wert, 0 = maximaler Wert). Die Stellgröße bezüglich des grösseren Istwerts wird jeweils erhöht, die Stellgröße bezüglich des kleineren Istwerts entspricht der Führungsgrösse.

Führungsgrösse = 65536 - (XS / XMAX x 65536).

11. Funktionen für das ecomat tdm R 360

Das ecomat tdm R 360 ist eine grafikfähige freiprogrammierbare Dialoggeräteserie zur Anzeige von Daten, Texten, Grafiken und Meldungen. Die im folgenden beschriebenen Funktionen behandeln nicht die Programmierung dieser Geräte, sondern nur die notwendigen Funktionen zum Datenaustausch mit dem Steuerungssystem CS0015. Die eigendliche Displayprogrammierung, z.B Aufbau der Grafik-Bildseiten und Festlegung der Kommunikationsparameter wird mit dem komfortablen Windows-Editor ecolog tdm R 360 durchgeführt.

Unabhängig davon ob die Programmierung der Steuerung oder des Displays über die serielle Schnittstelle statt findet, wird der Datenaustausch zwischen einem Steuerungsmodul und einem Displays über den CAN-Bus abgewickelt. Es empfielt sich aus diesem Grunde die Beschreibung im Kapitel 5. zu lesen.

Die Bibliothek TDM_x.LIB

Im Gegensatz zu den anderen Gerätebibliotheken, wurde die TDM_x.LIB nicht in einer Hochsprache (z.B. ´C´) oder in Assembler, sondern in der IEC-Sprache ´Strukturierter Text´ (ST) programmiert. Dieses hat den Vorteil, daß der kundige Anwender die Funktionen nach seinen eigenen Bedürfnissen anpassen und erweitern kann.

So sind z.B. in den Funktionen TDM_CONFIG und TDM_DATA_TRANSFER die Anzahl der Variablen, die mit dem tdm R 360 ausgetauscht werden können begrenzt.

Außerdem besteht in der Basisbibliothek nur die Möglichkeit zwischen einem Steuerungsmodul und einem Display zu kommunizieren. Die Identifier sind in den *globalen Variablen* voreingestellt. Diese Zahlenwerte müssen auch als Kommunikationsparameter bei der *Displayparametrierung, CAN-Einstellungen* angegeben werden.

Empfangs-Identifier	rxid : WORD := 220
Sende-Identifier	txid : WORD := 221

Um z.B. mit mehreren Steuerungsmodulen über den CAN-Bus auf ein Display zugreifen zu können, muß jedem Steuerungsmodul in den globalen Variablen ein eigener Sende-Identifier zugewiesen werden.

Die in der Bibliothek abgelegten Funktionen dienen dem Datenaustausch für Soll- und Istwerte, dem Aufruf von sogenannten SPS-Bildern, der Abfrage und Ansteuerung von Gerätefunktionen (Tastatur, LEDs, Geräteparameter).

Funktionsgruppen

Man kann die Funktionen in folgende Gruppen einteilen:

• Datenaustausch, Variablendefinition von Soll- und Istwerten	TDM_DATA_TRANSFER TDM_CONFIG TDM_READ_INTERN TDM_WRITE_INTERN
• Setzen und rücksetzen von SPS-Bildern und Meldungen	TDM_PICTURE TDM_MESSAGE TDM_REFRESH
• Abfrage und Auswertung des Gerätestatus und setzen und rücksetzen der LEDs	TDM_CONTROL_STATUS_REPORT TDM_REQUEST_STATUS TDM_REPORT_STATUS TDM_REPORT_KEYDATA TDM_LED TDM_SINGLE_LED_ON_OFF
• Gerätekontrolle	TDM_PARAM TDM_RESET

Beispielprogramm

Ein Beispielprogramm in Funktionsplan (FUP) ist auf der Programmdiskette ecolog 100*plus* gespeichert. In diesem einfachen Programm wird der generelle Programmaufbau und Datenaustausch zwischen der CS0015 und dem tdm R 360 gezeigt.

In den nachfolgenden Funktionsbeschreibungen wird nicht auf die Bedienung und Programmierung der Displayreihe ecomat tdm R 360 eingegangen. Diese Informationen sind dem entsprechenden Geräte- und Softwarehandbuch zu entnehmen.

11.1. Datenaustausch und Variablendefinition

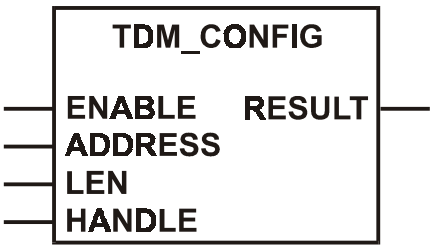
Funktion

TDM_CONFIG

Library

TDM.LIB

Funktionssymbol



Zweck

Die Funktion dient der Definition von Datenobjekten (Variablen) in der Initialisierung, die im tdm R 360 angezeigt werden sollen.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird ausgeführt FALSE: Funktion wird nicht ausgeführt
ADDRESS	DINT	Adresse der Variablen
LEN	BYTE	Anzahl der zu übertragenen Bytes
HANDLE	WORD	Bezeichnung (Nummer) der Variablen im tdm

Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BOOL	TRUE: Funktionsaufruf war erfolgreich

Beschreibung

TDM_CONFIG wird nur einmalig in der Initialisierungsroutine der Applikationssoftware aufgerufen. Über den Funktionseingang ENABLE kann anschließend die Ausführung gesperrt werden.

Dem Eingang ADDRESS muß die physikalische Adresse der Variablen zugewiesen werden. Dazu ermittelt man mit dem Adressoperator ADR die Hardwareadresse. Dieses Ergebnis muß an ADDRESS übergeben werden.

LEN legt die Anzahl der Bytes fest, die ab der Adresse übertragen werden (z.B. 2 = 2 Bytes (WORD), 4 = 4 Bytes (DWORD).

In Anhängigkeit vom Vorgabewert in der Bibliothek TDM_x.LIB können nur 50 Werte definiert werden.

HANDLE wird die festgelegte Variablennummer aus dem tdm R 360 zugewiesen. HANDLE ist die tdm-Adresse der Variablen. Eine Handle-Nummer darf sowohl im tdm als auch in der Applikationssoftware nur einmal vergeben werden.

RESULT zeigt an, ob der Aufruf der Funktion erfolgreich war. Ohne einen einmaligen, erfolgreichen Aufruf der Funktion für jede auszutauschende Variable, können keine Soll- und Istwerte aus der Steuerung im Display angezeigt werden.

Funktion

TDM_DATA_TRANSFER

Library

TDM.LIB

Funktionssymbol

Zweck

Diese Funktion wickelt den kompletten Datenaustausch zwischen dem tdm R 360 und dem Steuerungsmodul ab.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
INIT	BOOL	TRUE: Funktionsinitialisierung FALSE: zyklischer Funktionsaufruf

Funktionsausgänge, keine

Beschreibung

TDM_DATA_TRANSFER ist für die komplette Kommunikation zwischen dem Display und der Steuerung verantwortlich. Durch Einbindung dieser Funktion werden sowohl die Soll- und Istwerte, Setzen und Rücksetzen der Bilder und Meldungen, sowie der komplette Gerätestatus übertragen.

Zur Initialisierung der Funktion muß diese **einmalig** mit TRUE am INIT-Eingang aufgerufen werden. Im laufen Zyklus muß die Funktion mindestens einmal aufgerufen werden. Der INIT-Eingang wird dann FALSE gesetzt.

Wird CANopen zusammen mit den tdm-Funktionen genutzt, muß nach einem NMT_RESET_NODE/_COMM die Funktion erneut mit INIT = TRUE aufgerufen werden. Da die tdm-Funktionen direkte CAN-Objekte benutzen, gehen die Definitionen für diese verloren.

In langen Steuerungszyklen sollte die Funktion mehrfach aufgerufen werden, um den Datendurchsatz zwischen den Geräten zu vergrößern.

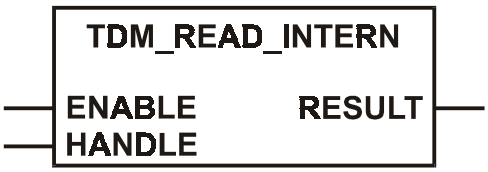
Funktion

TDM_READ_INTERN

Library

TDM.LIB

Funktionssymbol



Zweck

Die Funktion liest eine interne tdm R 360 Variable aus.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird ausgeführt FALSE: Funktion wird nicht ausgeführt
HANDLE	WORD	Bezeichnung (Nummer) der internen Variablen im tdm R 360

Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BOOL	TRUE: HANDLE wurde gefunden (war definiert)

Beschreibung

Im Gegensatz zu den Variablen, die als Soll- und Istwerte in der Steuerung verarbeitet, erzeugt und von den Betriebssystemen der Geräte automatisch immer auf dem aktuellen Stand gehalten werden, können die internen tdm-Variablen (z.B. Uhrzeit, oder im tdm R 360 gespeicherte Werte) nicht ´automatisch´ gelesen und geschrieben werden.

Wie jede andere Variable auch, muß eine interne Variable mit TDM_CONFIG der Applikationssoftware bekannt gemacht werden. Der anschließende Lesevorgang wird durch Aufruf der Funktion TDM_READ_INTERN einmalig ausgeführt. Über den Funktionseingang ENABLE kann die Ausführung gesperrt werden.

HANDLE wird die festgelegte Nummer der internen Variablen aus dem tdm R 360 zugewiesen. Eine Handle-Nummer darf sowohl im tdm R 360 als auch in der Applikationssoftware nur einmal vergeben werden.

RESULT = TRUE zeigt an, ob der beim Aufruf der Funktion angegebene Handle gefunden wurde.

Für die Verwaltung interner Daten in der Steuerung muß der Programmierer durch geeignete Softwareroutinen selber sorgen. Eine geänderte interne Variable wird nicht automatisch, sondern nur auf Anforderung mit TDM_READ_INTERN an die Steuerung übertragen.

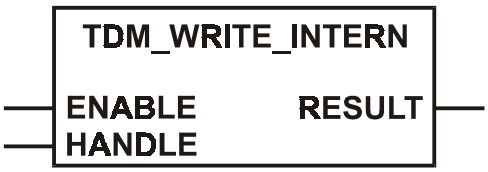
Funktion

TDM_WRITE_INTERN

Library

TDM.LIB

Funktionssymbol



Zweck

Die Funktion schreibt eine interne Variable an das tdm R 360.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird ausgeführt FALSE: Funktion wird nicht ausgeführt
HANDLE	WORD	Bezeichnung (Nummer) der internen Variablen im tdm

Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BOOL	TRUE: HANDLE wurde gefunden (war definiert)

Beschreibung

Im Gegensatz zu den Variablen, die als Soll- und Istwerte in der Steuerung verarbeitet, erzeugt und von den Betriebssystemen der Geräte automatisch immer auf dem aktuellen Stand gehalten werden, können die internen tdm-Variablen (z.B. Uhrzeit, oder im tdm R 360 gespeicherte Werte) nicht ´automatisch´ gelesen und geschrieben werden.

Wie jede andere Variable auch, muß eine interne Variable mit TDM_CONFIG der Applikationssoftware bekannt gemacht werden. Der anschließende Schreibvorgang wird durch Aufruf der Funktion TDM_WRITE_INTERN einmalig ausgeführt. Über den Funktionseingang ENABLE kann die Ausführung gesperrt werden.

HANDLE wird die festgelegte Nummer der internen Variablen aus dem tdm R 360 zugewiesen. Eine Handle-Nummer darf sowohl im tdm R 360 als auch in der Applikationssoftware nur einmal vergeben werden.

RESULT = TRUE zeigt an, ob der beim Aufruf der Funktion angegebene Handle gefunden wurde.

Für die Verwaltung interner Daten in der Steuerung muß der Anwenderprogrammierer durch geeignete Softwareroutinen selber sorgen. Eine in der Steuerung geänderte interne Variable wird nicht automatisch, sondern nur durch Aufruf der Funktion TDM_WRITE_INTERN an das Display übertragen.

11.2. Setzen und rücksetzen von Bildern und Meldungen

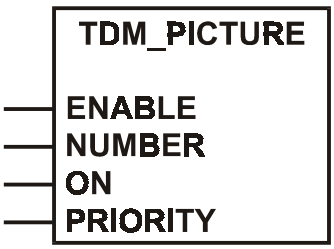
Funktion

TDM_PICTURE

Library

TDM.LIB

Funktionssymbol



Zweck

Die Funktion setzt bzw. setzt ein SPS-Bild zurück.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird ausgeführt FALSE: Funktion wird nicht ausgeführt
NUMBER	BYTE	tdm-Bildnummer
ON	BOOL	TRUE: Bild wird angezeigt (gesetzt) FALSE: Bild wird nicht angezeigt (zurückgesetzt)
PRIORITY	BOOL	TRUE: Bild wird als Prioritätsbild angezeigt (gesetzt)

Funktionsausgänge, keine

Beschreibung

Durch den Aufruf von TDM_PICTURE wird ein SPS-Bild gesetzt bzw. zurückgesetzt.

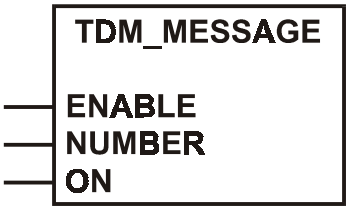
TDM_PICTURE kann, muß aber nicht zyklisch aufgerufen werden. Um Zykluszeit zu sparen, kann fallweise durch den Funktionseingang ENABLE die Ausführung der Funktion gesperrt werden. Ein einmaliger Aufruf der Funktion, wobei dem Funktionseingang ON der Wert TRUE zugewiesen wird, setzt das an Eingang NUMBER angegebene Bild. Ein weiterer Aufruf mit ON = TRUE hat keine Auswirkung (kostet aber durch die Überprüfung Zykluszeit). Wird ON = FALSE gesetzt und die Funktion mit der entsprechenden Bildnummer aufgerufen, wird das Bild zurückgesetzt.

Ein Aufruf der Funktion mit PRIORITY = TRUE kennzeichnet das SPS-Bild zusätzlich als Prioritätsbild. Prioritätsbilder werden unabhängig vom aktuellen Gerätestatus des Displays (z.B. Sollwerteingabe) sofort angezeigt. Alle anderen Displayaktivitäten werden unterdrückt.

Funktion **TDM_MESSAGE**

Library **TDM.LIB**

Funktionssymbol



Zweck Die Funktion setzt bzw. setzt eine Meldung zurück.

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird ausgeführt FALSE: Funktion wird nicht ausgeführt
NUMBER	BYTE	tdm-Bildnummer
ON	BOOL	TRUE: Meldung wird angezeigt (gesetzt) FALSE: Meldung wird nicht angezeigt (rückgesetzt)

Funktionsausgänge, keine

Beschreibung Durch den Aufruf von TDM_MESSAGE wird ein tdm-Meldung gesetzt bzw. zurückgesetzt.

Um Zykluszeit zu sparen, kann fallweise durch den Funktionseingang ENABLE die Ausführung der Funktion gesperrt werden.

TDM_MESSAGE kann, muß aber nicht zyklisch aufgerufen werden. Ein einmaliger Aufruf der Funktion, wobei dem Funktionseingang ON der Wert TRUE zugewiesen wird, setzt die an Eingang NUMBER angegebene Meldung. Ein weiterer Aufruf mit ON = TRUE hat keine Auswirkung (kostet aber durch die Überprüfung Zykluszeit). Wird ON = FALSE gesetzt und die Funktion mit der entsprechenden Meldungsnummer aufgerufen, wird diese zurückgesetzt.

Funktion **TDM_REFRESH**

Library **TDM.LIB**

Funktionssymbol

Zweck Die Funktion frischt den aktuellen Status der Bilder und Meldungen zwischen Steuerung und dem Display auf.

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird ausgeführt FALSE: Funktion wird nicht ausgeführt

Funktionsausgänge, keine

Beschreibung Z.B. nach einen Spannungsausfall am Display werden die aktuell gesetzten SPS-Bilder und Meldungen nicht mehr angezeigt. Durch den Aufruf von TDM_REFRESH wird der aktuelle Status unabhängig von TDM_PICTURE und TDM_MESSAGE an das Display übertragen.

Diese Funktion muß nicht, sollte aber in die Applikationssoftware eingebunden werden. Durch Einsatz der Funktion wird der Programmierer von der Aufgabe entlastet, selbst permanent den Gerätestatus zu überwachen und durch Aufruf der SPS-Bilder und Meldungen diese wieder zu aktivieren.

11.3. Der Gerätestatus und die LEDs

Funktion

TDM_CONTROL_STATUS_REPORT

Library

TDM.LIB

Funktionssymbol

Zweck

Die Funktion aktiviert bzw. deaktiviert den automatischen Status-Report des tdm R 360.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird ausgeführt FALSE: Funktion wird nicht ausgeführt
ON	BOOL	TRUE: automatischer Statusreport ist eingeschaltet FALSE: automatischer Statusreport ist ausgeschaltet

Funktionsausgänge, keine

Beschreibung

Der Statusreport stellt der Applikationssoftware alle relevanten Geräteinformationen zur Verfügung. Das sind:

- aktuell angezeigtes Bild
- aktuell angezeigte Meldung
- Status der Tastatur (welche Taste wurde gedrückt bzw. losgelassen)
- LED ist gesetzt bzw. nicht gesetzt
- Gerätestatus (z.B. Sollwerteingabe aktiv)

Mit TDM_CONTROL_STATUS_REPORT wird der automatische Statusreport aktiviert. Damit wird jede Änderung bei den oben angegebenen Punkten an die Steuerung übertragen.

Der aktuelle Status kann mit TDM_REPORT_STATUS und TDM_REPORT_KEYDATA ausgewertet werden.

Die Funktion muß jeweils nur einmalig zum Ein- und zum Ausschalten aufgerufen werden. Anschließend kann die Ausführung der Funktion über den Eingang ENABLE gesperrt werden. Dennoch ist es sinnvoll, wenn TDM_CONTROL_STATUS_REPORT aktiv ist, die Funktion in gewissen Zeitabständen (z.B. alle 500 ms) erneut aufzurufen. Damit wird verhindert, daß z.B. nach einem Ausfall der tdm-Spannungsversorgung, die Überwachung des Gerätestatus nicht mehr funktioniert.

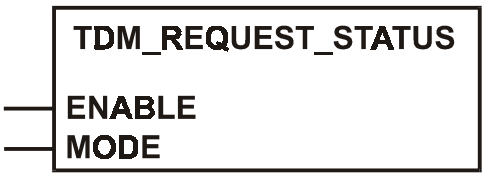
Funktion

TDM_REQUEST_STATUS

Library

TDM.LIB

Funktionssymbol



Zweck

Die Funktion fragt einmalig den aktuellen Status-Report des tdm R 360 ab.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird ausgeführt FALSE: Funktion wird nicht ausgeführt
MODE	BYTE	Wert zur Auswahl der Rückgabeparameter

Funktionsausgänge, keine

Beschreibung

Der Statusreport stellt der Applikationssoftware alle relevanten Geräteinformationen zur Verfügung. Das sind:

- aktuell angezeigtes Bild
- aktuell angezeigte Meldung
- Status der Tastatur (welche Taste wurde gedrückt bzw. losgelssen)
- LED ist gesetzt bzw. nicht gesetzt
- Gerätestatus (z.B. Sollwerteingabe aktiv)
- Status des Meldeausganges (nicht CR1000)

Mit TDM_REQUEST_STATUS wird einmalig ein Statusreport vom Display angefordert. Über MODE wird festgelegt welche Daten abgefragt werden sollen. Es wird, entsprechend einer Momentaufnahme, der Status der oben angegebenen Punkte an die Steuerung übertragen.

Wert MODE	Ausgabe über	Beschreibung
0	TDM_REPORT_STATUS	Gerätestatus
1	TDM_REPORT_KEYDAT	Tastatur-Status (Taste 1-32)
2	TDM_REPORT_KEYDAT	Tastatur-Status (Taste 33-64)
3	TDM_REPORT_KEYDAT	LED-Status (Taste 1-32)
4	TDM_REPORT_KEYDAT	LED-Status (Taste 33-64)
5	TDM_REPORT_OUTPUT	Status des Meldeausganges

Die Funktion muß für jeden Statusreport erneut aufgerufen werden. Um die Ausführung der Funktion zu sperren, kann der Eingang ENABLE auf FALSE gesetzt werden.

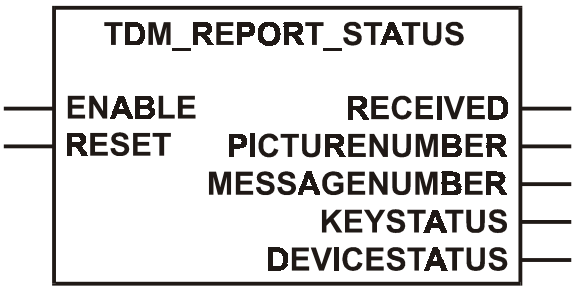
Funktion

TDM_REPORT_STATUS

Library

TDM.LIB

Funktionssymbol



Zweck

Die Funktion zeigt den aktuellen Status-Report des tdm R 360 an.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: neue Daten empfangen FALSE: Funktion wird nicht ausgeführt
RESET	BOOL	TRUE: setzt den Ausgang RECEIVED auf FALSE

Funktionsausgänge

Name	Datentyp	Beschreibung
RECEIVED	BOOL	TRUE: Funktion wurde ausgeführt FALSE: keine neuen Daten
PICTURE-NUMBER	WORD	Nummer des aktuellen Bildes
MESSAGE-NUMBER	WORD	Nummer der aktuellen Meldung
KEYSTATUS	BYTE	Tastaturstatus
DEVICE-STATUS	BYTE	Gerätestatus

Beschreibung

Die Funktion TDM_REPORT_STATUS wird zyklisch aufgerufen. Neben der aktuellen Bild- und Meldungsnummer wird auch der Status der Bedientasten und des Gerätes ausgegeben.

Die einzelnen Zahlenwerte sind dem tdm R 360 Handbuch zu entnehmen.

Die Tastaturbits werden im Normalfall in dezimale Zahlen umgerechnet.

Der Gerätestatus wird direkt als Dezimalwert ausgegeben.

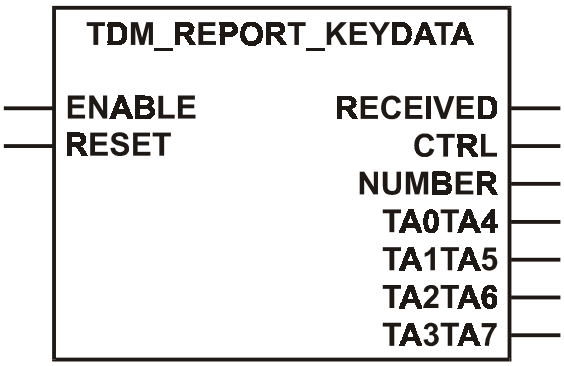
Funktion

TDM_REPORT_KEYDATA

Library

TDM.LIB

Funktionssymbol



Zweck

Die Funktion zeigt den aktuellen Tasten-Status des tdm R 360 an.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird ausgeführt FALSE: Funktion wird nicht ausgeführt
RESET	BOOL	TRUE: setzt den Ausgang RECEIVED auf FALSE

Funktionsausgänge

Name	Datentyp	Beschreibung
RECEIVED	BOOL	TRUE: Funktion wird ausgeführt FALSE: Funktion wird nicht ausgeführt
CTRL	BYTE	Steuerparameter für Tasten und LED Abfrage
NUMBER	BYTE	nur bei CTRL = 0, enthält die Tasten-nummer und deren Status
TA0TA4	BYTE	gültig für CTRL = 0 ... 4, Statusbytes der Tasten bzw. LEDs (bitorientiert)
TA1TA5	BYTE	gültig für CTRL = 0 ... 4, Statusbytes der Tasten bzw. LEDs (bitorientiert)
TA2TA6	BYTE	gültig für CTRL = 0 ... 4, Statusbytes der Tasten bzw. LEDs (bitorientiert)
TA3TA7	BYTE	gültig für CTRL = 0...4, Statusbytes der Tasten bzw. LEDs (bitorientiert)

Beschreibung

Die Funktion TDM_REPORT_KEYDATA kann zyklisch aufgerufen werden. Sie überträgt den aktuellen Tasten und LED-Status.

TA0 ... TA7 stellen den Tasten/LED-Status in Bytes (d.h. bitorientiert) zur Verfügung. Über CTRL wird ausgewählt welche Tasten/LED-Gruppe abgefragt werden soll.

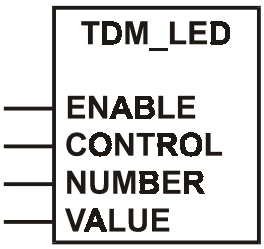
Die einzelnen Zahlenwerte sind dem tdm R 360 Handbuch zu entnehmen.

CTRL gibt immer den Wert 0 zurück wenn TDM_CONTROL_STATUS_REPORT eingeschaltet ist.

Funktion **TDM_LED**

Library **TDM.LIB**

Funktionssymbol



Zweck Die Funktion aktiviert bzw. deaktiviert die tdm-LEDs

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird ausgeführt FALSE: Funktion wird nicht ausgeführt
CONTROL	BYTE	Steuerparameter zum Setzen/ Rücksetzen der LEDs
NUMBER	BYTE	LED-Maskennummer oder einzelne LED-Nummer
VALUE	BYTE	Maskenwert wenn NUMBER = LED-Maskennummer

Funktionsausgänge, keine

Beschreibung Die Funktion TDM_LED kann zyklisch aufgerufen werden. Durch diese Funktion wird der Status der Tastatur-LEDs geändert (gesetzt/rückgesetzt). Je nach angegebenen Steuerparameter, können die LEDs einzeln oder bitweise über einen Maskenwert in 8er-Gruppen bearbeitet werden.

Die einzelnen Zahlenwerte sind dem tdm R 360 Handbuch zu entnehmen.

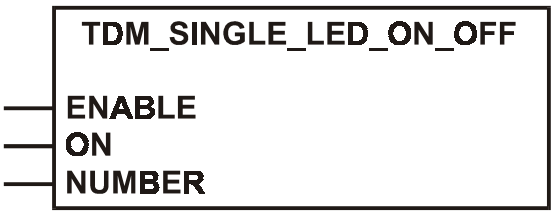
Funktion

TDM_SINGLE_LED_ON_OFF

Library

TDM.LIB

Funktionssymbol



Zweck

Die Funktion aktiviert bzw. deaktiviert eine einzelne tdm-LED

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird ausgeführt FALSE: Funktion wird nicht ausgeführt
ON	BOOL	TRUE: LED eingeschaltet FALSE: LED ausgeschaltet
NUMBER	BYTE	einzelne LED-Nummer

Funktionsausgänge, keine

Beschreibung

Mit TDM_SINGLE_LED_ON_OFF kann eine einzelne LED ein- (ON=TRUE) bzw. ausgeschaltet (ON=FALSE) werden.

Im Gegensatz zur Funktion TDM_LED kann TDM_SINGLE_LED_ON_OFF dauerhaft gesetzt werden. Das entsprechende CAN-Kommado wird innerhalb der Funktion aber nur einmalig übertragen. Dadurch wird eine Überlastung des CAN-Bus ausgeschlossen.

Die einzelnen Zahlenwerte sind dem tdm R 360 Handbuch zu entnehmen.

11.4. Gerätekontrolle

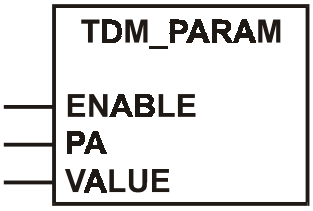
Funktion

TDM_PARAM

Library

TDM.LIB

Funktionssymbol



Zweck

Die Funktion überwacht und überträgt die Geräteparameter zur Grundeinstellung des Displays.

Parameter

Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird ausgeführt FALSE: Funktion wird nicht ausgeführt
PA	BYTE	Parametriercode
VALUE	BYTE	Zusatzwert, muß je nach Parametriercode noch angegeben werden.

Funktionsausgänge, keine

Beschreibung

Die Funktion TDM_PARAM sollte nur aufgerufen werden, wenn neue Geräteparameter gesetzt werden sollen. Mit TDM_PARAM können z.B. die Helligkeit, die Softkeymaske oder die Rollierzeit für die Meldungen verändert werden.

Die einzelnen Zahlenwerte sind dem tdm R 360 Handbuch zu entnehmen.

Funktion **TDM_RESET**

Library **TDM.LIB**

Funktionssysmbol

Zweck Die Funktion setzt alle Geräteparameter des tdm R 360, in die Grundstellung zurück.

Parameter Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird ausgeführt FALSE: Funktion wird nicht ausgeführt

Funktionsausgänge, keine

Beschreibung Das Display wird neu gestartet. Es verhält sich dabei so, als ob die Spannung aus- und wieder eingeschaltet wird.

Anhang 1. Adressbelegung CS0015

Anhang 1.1. Gesamtübersicht

Adresse	Symbol	Erläuterung	
%IX0.00-%IX0.07	-	Eingänge (Byte %IB0, Wort %IW0)	
%IX0.08-%IX0.15	-	Eingänge (Byte %IB1, Wort %IW0)	
%QX0.00-%QX0.07	-	Ausgänge (Byte %QB0, Wort %QW0)	
%QX0.08-%QX0.15	-	Ausgänge (Byte %QB1, Wort %QW0)	
%IW0	I0	Eingangs-Wort 0	
%QW0	Q0	Ausgangs-Wort 0	
%IB2	S2	16-stufiger Drehschalter	
%IX1.8	S3	Drucktaster S3	
%IX2.0	S4	Drucktaster S4	
%IX2.8	S5	Drucktaster S5	
Merkerbit*	ERROR	Error-Bit setzen	
Merkerbit*	ERROR_MEMORY	Speicher-Fehler	
Merkerbit*	UNLOCK	Programmiermodus freigeben	(FALSE)
Merkerbit*	SERIAL_MODE	serielle Kommunikation einschalten	(FALSE)
Merkerbit*	CAN_OPEN	CANopen-Modus einschalten	(FALSE)
Merkerbit*	ISO_DIRECTION	Daten senden oder empfangen	(FALSE)
Merkerbit*	CAN_ERROR	CAN-Bus Fehler (Sammelfehler-Bit)	
Merkerbit*	CAN_INIT_ERROR	CAN Initialisierungsfehler	
Merkerbit*	CAN_BUS_OFF_ERROR	CAN-Bus off Fehler	
Merkerbit*	CAN_DATA_ERROR	CAN-Data Fehler	
Merkerbit*	CAN_TX_OVERRUN_ERROR	CAN-TX-Overrun Fehler	
Merkerbit*	CAN_RX_OVERRUN_ERROR	CAN-RX-Overrun Fehler	
Merkerbit*	COP_SYNCFAIL_ERROR	SYNC-Objekt fehlt	
Merkerbit*	COP_GUARDFAIL_ERROR	Guarding-Objekt fehlt	
Merkerbyte*	COP_GUARDFAIL_NODEID	Nummer des fehlenden CANopen-Slaves	
Merkerbit*	COP_PREOPERATIONAL	CANopen-Modus Preoperational	
Merkerbit*	COP_PRESYNC	Presync-Merker	
Merkerbit*	COP_SYNC	Sync-Merker	
Merkerbit*	COP_EVENT_RESETCOM	Kommunik.-Reset wurde vom Master ausgelöst	
Merkerbit*	COP_EVENT_RESETNODE	Node-Reset wurde vom Master ausgelöst	
Merkerbit*	COP_GUARDING_AGAIN	Node-Guarding nach Reset-Node erneut starten	
Merkerword*	CANBAUDRATE	aktuell eingestellte CAN-Baudrate	
Merkerbyte*	NODEID	aktuell eingestellte Knotennummer	
Merkerbit*	TEXT_MODE	CAN-Kommunikation Kompaktdisplay(FALSE)	
Merkerbit*	TEXT_KEY_F1	Funktionstaste F1 Kompaktdisplay	
Merkerbit*	TEXT_KEY_F2	Funktionstaste F2 Kompaktdisplay	
Merkerbit*	TEXT_KEY_F3	Funktionstaste F3 Kompaktdisplay	
Merkerbit*	TEXT_KEY_ESC	Taste ESC Kompaktdisplay	
Merkerbit*	TEXT_KEY_LEFT	Taste Pfeil-LINKS Kompaktdisplay	
Merkerbit*	TEXT_KEY_RIGHT	Taste Pfeil-RECHTS Kompaktdisplay	
Merkerbit*	TEXT_KEY_DOWN	Taste Pfeil-UNTEN Kompaktdisplay	
Merkerbit*	TEXT_KEY_UP	Taste Pfeil-OBEN Kompaktdisplay	
Merkerbit*	TEXT_KEY_ENTER	Taste ENTER Kompaktdisplay	
Merkerbit*	TEXT_LED_F1	LED Funktionstaste F1 Kompaktdisplay	
Merkerbit*	TEXT_LED_F2	LED Funktionstaste F1 Kompaktdisplay	
Merkerbit*	TEXT_LED_F3	LED Funktionstaste F1 Kompaktdisplay	

Anhang 1.2. Eingänge und Ausgänge

Name	Bit-Adresse	Klemme	Bemerkung
I0	%IW0		Eingangswort 0
	%IX0.00	X1, In 0	Frequenz/Cycle 0, Drehgeber 0
	%IX0.01	X1, In 1	Frequenz/Cycle 1, Drehgeber 1
	%IX0.02	X1, In 2	Frequenz/Cycle 2, Drehgeber 2
	%IX0.03	X1, In 3	Frequenz/Cycle 3, Drehgeber 3
	%IX0.04	X1, In 4	
	%IX0.05	X1, In 5	
	%IX0.06	X1, In 6	
	%IX0.07	X1, In 7	
	%IX0.08	X2, In 8	
	%IX0.09	X2, In 9	
	%IX0.10	X2, In 10	
	%IX0.11	X2, In 11	
	%IX0.12	X2, In 12	
	%IX0.13	X2, In 13	
	%IX0.14	X2, In 14	
	%IX0.15	X2, In 15	
Q0	%QW0		Ausgangswort 0
	%QX0.00	X3, Out 0	PWM 0
	%QX0.01	X3, Out 1	PWM 1
	%QX0.02	X3, Out 2	PWM 2
	%QX0.03	X3, Out 3	PWM 3
	%QX0.04	X3, Out 4	PWM 4
	%QX0.05	X3, Out 5	PWM 5
	%QX0.06	X3, Out 6	PWM 6
	%QX0.07	X3, Out 7	PWM 7
	%QX0.08	X4, Out 8	
	%QX0.09	X4, Out 9	
	%QX0.10	X4, Out 10	
	%QX0.11	X4, Out 11	
	%QX0.12	X4, Out 12	
	%QX0.13	X4, Out 13	
	%QX0.14	X4, Out 14	
	%QX0.15	X4, Out 15	

Anhang 1.3. Der Merkerbereich

Inhalt	Merkeradresse	Bemerkung
	%MW 4096	
E/A- und System-Daten	%MW 3968	nicht beschreiben
TX - PDOs	%MW 2032	64 Bytes TX-PDOs
RX - PDOs	%MW 2000	64 Bytes RX-PDOs
Slave-Daten	%MW 1329 %MW 1010	Datenbereich für 32 CANopen E/A-Slaves
remanente Daten (Funktion)	%MW 511 %MW 128	768 Byte über Funktionsaufruf sicherbar
remanente Daten (Retains)	%MW 127 % MW 0	256 Bytes remanente Daten

Der gesamte Merkerbereich in der Steuerung CS0015 beträgt 8 kByte. Die hinterlegten Flächen sind direkt durch das Betriebssystem festbelegt und können nur für den angegebenen Zweck verwendet werden. Der übrige Speicherbereich kann vom Programmiersystem frei genutzt werden. Ob dieser für den Anwender zur Verfügung steht, muß daher im Einzelfall geprüft werden. Wenn möglich sollte daher von einer direkten Adressierung abgesehen werden.

Anhang 1.4. CANopen Geräteschnittstelle

- Das Gerät wird die Geräteklasse „Programmable Device“ entsprechend CiA DS 405 eingeordnet und gekennzeichnet.
- Es werden 1 Server SDO und die 4 Default PDOs gemäß CiA DS 401 eingerichtet. Die Default - Identifier sind ensprechend des „predefined connection set“ vergeben. Zusätzlich stehen 2 x 6 PDOs zur freien Verfügung.
- Die COB-IDs der PDOs sowie die Übertragungsart (synch / asynch) der einzelnen PDO sind konfigurierbar.
- Im Slave-Modus erwartet das E/A-Modul ein SYNC-Objekt. Der CAN Identifier des Synchobjektes ist konfigurierbar. Nach einer Änderung wird der ID automatisch spannungausfallsicher gespeichert.
- Im Master-Modus erzeugt das E/A-Modul ein SYNC-Objekt. Der CAN Identifier des Synchobjektes ist konfigurierbar. Nach einer Änderung wird der ID automatisch spannungausfallsicher gespeichert.
- Das E/A-Modul unterstützt „node guarding“. Die „guard time“, der „life time factor“ und der CAN Identifier des Guard Objektes sind konfigurierbar und werden spannungsausfallsicher gespeichert.
- Das E/A-Modul generiert ein Emergency Objekt . Der COB-ID des EMCY-Objektes ist konfigurierbar.

Parameterübersicht

Parameter	werkseitig eingestellter Defaultwert	Änderung automatisch gesichert	gültig nach
Node-ID	32	X	sofort
Baudrate	3 (125 kBit/s)	X	Reset
COB ID Synch Objekt	0x80	X	sofort
Communication Cycle	0 (Off)	X	sofort
Guard Time	0 (Off)	X	sofort
Life Time Factor	0 (Off)	X	sofort
COB ID Guarding	0x700 + Node ID	X	sofort
COB ID EMCY	0x80 + Node ID	X	sofort
Transmit Type Receive PDO1	asynchron	-	operational*
Transmit Type Receive PDO2	asynchron	-	operational*
Transmit Type Transmit PDO1	asynchron	-	operational*
Transmit Type Transmit PDO2	asynchron	-	operational*
COP ID Receive PDO1	0x200 + Node ID	-	sofort
COP ID Receive PDO2	0x300 + Node ID	-	sofort
COP ID Transmit PDO1	0x180 + Node ID	-	sofort
COP ID Transmit PDO2	0x280 + Node ID	-	sofort

* Die Änderung mit PDO_RX_CONFIG und PDO_TX_CONFIG bewirkt einen CANopen-Reset. Dadurch werden alle nicht gesicherten Einstellungen auf die Defaultwerte gesetzt. Aus diesem Grunde muß ein zweistufiger Bootup durchgeführt werden (siehe CS0015 als CANopen-Master).

Anhang 1.5. Objektverzeichnis

Anhang 1.5.1. Datenbereich Kommunikationsprofil, Index 1000 bis 1FFF

Index	S-Idx	Name	Typ	Default	Beschreibung
1000	0	device type	u32, ro	0x195	Prof. 405; Progammierbares Gerät
1001	0	error register	u8, ro	0x0	Bitcodiert gemäß Prof. 301; unterstützt wird: 0b0000 0000 kein Fehler 0b0000 000**1** generic error 0b000**1** 0000 communication error 0b**1**000 0000 manufacturer specific error
1004	0	number of PDOs	u32 ro	0x20002	2 Sende-PDOs 2 Empfangs-PDOs
	1	number of synch PDOs	u32 ro	0x20002	Alle PDOs können sychron oder asynchron übertragen werden
	2	number of asynch PDOs	u32 ro	0x20002	Alle PDOs können sychron oder asynchron übertragen werden
1005	0	COB ID SYNC-Object	u32 rw	0x80000080	CAN Identifier des SYNC Objektes
1006	0	Communic. Cycle	u32 rw	0x0	max. Zeit zwischen 2 SYNC-Objekten in µs. Nutzauflösung = 1 ms.
1007	0	synch window			wird nicht implementiert
1008	0	device name	str ro	ecomat 100	Gerätename: „ecomat 100“
1009	0	HW Version	str ro	CS*xxxx_x*	Betriebssystemversion
100A	0	SW Version	str ro	*jjmmtt*	Softwaredatum
100B	0	Node ID	u32 ro		nur zur Anfrage
100C	0	guard time	u16 rw	0x0	Zeit in ms. Das Gerät erwartet innerhalb dieser Zeit ein „node guarding“ des Netz-Masters. Wird hier der Wert 0 eingetragen, wird diese Funktion nicht unterstützt.
100D	0	life time factor	u8 rw	0x0	Wenn für „guard time“ x „life time“ kein „node guarding“ empfangen wird, meldet das Gerät den Fehler COP_GUARDFAIL_ERROR
100E	0	COB ID guarding	u32 rw	0x00000700 + Node ID	CAN Identifier des Node Guard Objektes
100F	0	number of SDOs			nicht implementiert (Es wird nur das Default SDO unterstützt)
1012	0	Time Stamp			nicht implementiert
1013	0	high res. Time Stamp			nicht implementiert

Index	S-Idx	Name	Typ	Default	Beschreibung
1014	0	COB ID Emergcy	u32 rw	0x40000080 +Node ID	• E/A-Modul reagiert nicht auf fremde EMCY Message (Bit 31 = 0) • E/A-Modul generiert EMCY Message (Bit 30 = 1) • 11 Bit ID (Bit 29 = 0) • ID = 0x80 + Node ID CAN-Identifier kann vom Benutzer geändert werden.
1200	0	Server SDOs	u8 ro	0x02	Anzahl der Einträge
	1	COB ID Rec SDO	u32 rw	0x600+ID	• SDO ist gültig (Bit 31 = 0) • CAN ID des Receive SDOs
	2	COB ID Trans SDO	u32 rw	0x580 + Node ID	• SDO ist gültig (Bit 31 = 0) • CAN ID des Transmit SDOs
1400	0	Receive PDO 1	u8 ro	0x02	Anzahl der Einträge RX PDO 1
	1	COB ID	u32 rw	0x200 + Node ID	• PDO ist gültig (Bit 31 = 0) • CAN ID des 1. RX PDOs
	2	Trans Type	u8 rw	0xFF	• 0x00 = synch acyclic • 0x01 ... 0xF0 = synch cyclic; Anzahl der Synchobjekte zwischen zwei Zugriffen • 0xFC nicht implementiert • 0xFD nicht implementiert • 0xFE = asynch manufac. specific event • 0xFF = asynch device profile event
1401	0	Receive PDO 2	u8 ro	0x02	Anzahl der Einträge RX PDO 2
	1	COB ID	u32 rw	0x300 + Node ID	• PDO ist gültig (Bit 31 = 0) • CAN ID des 2. RX PDOs
	2	Trans Type	u8 rw	0xFF	zulässige Werte wie bei RX PDO1
1402	0	Receive PDO 3	u8 ro	0x02	Anzahl der Einträge RX PDO 3
	1	COB ID	u32 rw	0x382	• PDO ist gültig (Bit 31 = 0) • CAN ID des 3. RX PDOs
	2	Trans Type	u8 rw	0xFF	zulässige Werte wie bei RX PDO1
1403	0	Receive PDO 4	u8 ro	0x02	Anzahl der Einträge RX PDO 4
	1	COB ID	u32 rw	0x383	• PDO ist gültig (Bit 31 = 0) • CAN ID des 4. RX PDOs
	2	Trans Type	u8 rw	0xFF	zulässige Werte wie bei RX PDO1
1404	0	Receive PDO 5	u8 ro	0x02	Anzahl der Einträge RX PDO 5
	1	COB ID	u32 rw	0x384	• PDO ist gültig (Bit 31 = 0) • CAN ID des 5. RX PDOs
	2	Trans Type	u8 rw	0xFF	zulässige Werte wie bei RX PDO1

Index	S-Idx	Name	Typ	Default	Beschreibung
1405	0	Receive PDO 6	u8 ro	0x02	Anzahl der Einträge RX PDO 6
	1	COB ID	u32 rw	0x385	• PDO ist gültig (Bit 31 = 0) • CAN ID des 6. RX PDOs
	2	Trans Type	u8 rw	0xFF	zulässige Werte wie bei RX PDO1
1406	0	Receive PDO 7	u8 ro	0x02	Anzahl der Einträge RX PDO 7
	1	COB ID	u32 rw	0x386	• PDO ist gültig (Bit 31 = 0) • CAN ID des 7. RX PDOs
	2	Trans Type	u8 rw	0xFF	zulässige Werte wie bei RX PDO1
1407	0	Receive PDO 8	u8 ro	0x02	Anzahl der Einträge RX PDO 8
	1	COB ID	u32 rw	0x387	• PDO ist gültig (Bit 31 = 0) • CAN ID des 8. RX PDOs
	2	Trans Type	u8 rw	0xFF	zulässige Werte wie bei RX PDO1
1600	0	Mapping Receive PDO 1	u32 ro	0x04	Anzahl der eingebundenen Appl. Objekte
	1	Index im Objekverzeichniss	u32 ro	0x5800 01	in 5800 SIdx 01 steht 1. Word Empfangsdaten RX PDO 1
	2	Index im Objekverzeichniss	u32 ro	0x5800 02	in 5800 SIdx 02 steht 2. Word Empfangsdaten RX PDO 1
	3	Index im Objekverzeichniss	u32 ro	0x5800 03	in 5800 SIdx 03 steht 3. Word Empfangsdaten RX PDO 1
	4	Index im Objekverzeichniss	u32 ro	0x5800 04	in 5800 SIdx 04 steht 4. Word Empfangsdaten RX PDO 1
1601	0	Mapping Receive PDO 2	u32 ro	0x04	Anzahl der eingebundenen Appl. Objekte
	1	Index im Objekverzeichniss	u32 ro	0x5810 01	in 5810 SIdx 01 steht 1. Word Empfangsdaten RX PDO 2
	2	Index im Objekverzeichniss	u32 ro	0x5810 02	in 5810 SIdx 02 steht 2. Word Empfangsdaten RX PDO 2
	3	Index im Objekverzeichniss	u32 ro	0x5810 03	in 5810 SIdx 03 steht 3. Word Empfangsdaten RX PDO 2
	4	Index im Objekverzeichniss	u32 ro	0x5810 04	in 5810 SIdx 04 steht 4. Word Empfangsdaten RX PDO 2
1602	0	Mapping Receive PDO 3	u32 ro	0x04	Anzahl der eingebundenen Appl. Objekte
	1	Index im Objekverzeichniss	u32 ro	0x5820 01	in 5820 SIdx 01 steht 1. Word Empfangsdaten RX PDO 3
	2	Index im Objekverzeichniss	u32 ro	0x5820 02	in 5820 SIdx 02 steht 2. Word Empfangsdaten RX PDO 3
	3	Index im Objekverzeichniss	u32 ro	0x5820 03	in 5820 SIdx 03 steht 3. Word Empfangsdaten RX PDO 3
	4	Index im Objekverzeichniss	u32 ro	0x5820 04	in 5820 SIdx 04 steht 4. Word Empfangsdaten RX PDO 3

Index	S-Idx	Name	Typ	Default	Beschreibung
1603	0	Mapping Receive PDO 4	u32 ro	0x04	Anzahl der eingebundenen Appl. Objekte
	1	Index im Objekverzeichniss	u32 ro	0x5830 01	in 5830 SIdx 01 steht 1. Word Empfangsdaten RX PDO 4
	2	Index im Objekverzeichniss	u32 ro	0x5830 02	in 5830 SIdx 02 steht 2. Word Empfangsdaten RX PDO 4
	3	Index im Objekverzeichniss	u32 ro	0x5830 03	in 5830 SIdx 03 steht 3. Word Empfangsdaten RX PDO 4
	4	Index im Objekverzeichniss	u32 ro	0x5830 04	in 5830 SIdx 04 steht 4. Word Empfangsdaten RX PDO 4
1604	0	Mapping Receive PDO 5	u32 ro	0x04	Anzahl der eingebundenen Appl. Objekte
	1	Index im Objekverzeichniss	u32 ro	0x5840 01	in 5840 SIdx 01 steht 1. Word Empfangsdaten RX PDO 5
	2	Index im Objekverzeichniss	u32 ro	0x5840 02	in 5840 SIdx 02 steht 2. Word Empfangsdaten RX PDO 5
	3	Index im Objekverzeichniss	u32 ro	0x5840 03	in 5840 SIdx 03 steht 3. Word Empfangsdaten RX PDO 5
	4	Index im Objekverzeichniss	u32 ro	0x5840 04	in 5840 SIdx 04 steht 4. Word Empfangsdaten RX PDO 5
1605	0	Mapping Receive PDO 6	u32 ro	0x04	Anzahl der eingebundenen Appl. Objekte
	1	Index im Objekverzeichniss	u32 ro	0x5850 01	in 5850 SIdx 01 steht 1. Word Empfangsdaten RX PDO 6
	2	Index im Objekverzeichniss	u32 ro	0x5850 02	in 5850 SIdx 02 steht 2. Word Empfangsdaten RX PDO 6
	3	Index im Objekverzeichniss	u32 ro	0x5850 03	in 5850 SIdx 03 steht 3. Word Empfangsdaten RX PDO 6
	4	Index im Objekverzeichniss	u32 ro	0x5850 04	in 5850 SIdx 04 steht 4. Word Empfangsdaten RX PDO 6
1606	0	Mapping Receive PDO 7	u32 ro	0x04	Anzahl der eingebundenen Appl. Objekte
	1	Index im Objekverzeichniss	u32 ro	0x5860 01	in 5860 SIdx 01 steht 1. Word Empfangsdaten RX PDO 7
	2	Index im Objekverzeichniss	u32 ro	0x5860 02	in 5860 SIdx 02 steht 2. Word Empfangsdaten RX PDO 7
	3	Index im Objekverzeichniss	u32 ro	0x5860 03	in 5860 SIdx 03 steht 3. Word Empfangsdaten RX PDO 7
	4	Index im Objekverzeichniss	u32 ro	0x5860 04	in 5860 SIdx 04 steht 4. Word Empfangsdaten RX PDO 7
1607	0	Mapping Receive PDO 8	u32 ro	0x04	Anzahl der eingebundenen Appl. Objekte
	1	Index im Objekverzeichniss	u32 ro	0x5870 01	in 5870 SIdx 01 steht 1. Word Empfangsdaten RX PDO 8
	2	Index im Objekverzeichniss	u32 ro	0x5870 02	in 5870 SIdx 02 steht 2. Word Empfangsdaten RX PDO 8
	3	Index im Objekverzeichniss	u32 ro	0x5870 03	in 5870 SIdx 03 steht 3. Word Empfangsdaten RX PDO 8
	4	Index im Objekverzeichniss	u32 ro	0x5870 04	in 5870 SIdx 04 steht 4. Word Empfangsdaten RX PDO 8

Index	S-Idx	Name	Typ		Default	Beschreibung
1800	0	Transmit PDO 1	u8	ro	0x02	Anzahl der Einträge TX PDO 1
	1	COB ID	u32	rw	0x180 + Node ID	• PDO ist gültig (Bit 31 = 0) • CAN ID des 1. TX PDOs
	2	Trans Type	u8	rw	0xFF	• 0x00 = synch acyclic • 0x01...0xF0 = synch cyclic; Anzahl der Synchobjekte zwischen zwei Zugriffen • 0xFC nicht implementiert • 0xFD nicht implementiert • 0xFE = asynch manufac. specific event • 0xFF = asynch device profile event
1801	0	Transmit PDO 2	u8	ro	0x02	Anzahl der Einträge TX PDO 2
	1	COB ID	u32	rw	0x280 + Node ID	• PDO ist gültig (Bit 31 = 0) • CAN ID des 2. TX PDOs
	2	Trans Type	u8	rw	0xFF	zulässige Werte wie bei TX PDO1
1802	0	Transmit PDO 3	u8	ro	0x02	Anzahl der Einträge TX PDO 3
	1	COB ID	u32	rw	0x38A	• PDO ist gültig (Bit 31 = 0) • CAN ID des 3. TX PDOs
	2	Trans Type	u8	rw	0xFF	zulässige Werte wie bei TX PDO1
1803	0	Transmit PDO 4	u8	ro	0x02	Anzahl der Einträge TX PDO 4
	1	COB ID	u32	rw	0x38B	• PDO ist gültig (Bit 31 = 0) • CAN ID des 4. TX PDOs
	2	Trans Type	u8	rw	0xFF	zulässige Werte wie bei TX PDO1
1804	0	Transmit PDO 5	u8	ro	0x02	Anzahl der Einträge TX PDO 5
	1	COB ID	u32	rw	0x38C	• PDO ist gültig (Bit 31 = 0) • CAN ID des 5. TX PDOs
	2	Trans Type	u8	rw	0xFF	zulässige Werte wie bei TX PDO1
1805	0	Transmit PDO 6	u8	ro	0x02	Anzahl der Einträge TX PDO 6
	1	COB ID	u32	rw	0x38D	• PDO ist gültig (Bit 31 = 0) • CAN ID des 6. TX PDOs
	2	Trans Type	u8	rw	0xFF	zulässige Werte wie bei TX PDO1
1806	0	Transmit PDO 7	u8	ro	0x02	Anzahl der Einträge TX PDO 7
	1	COB ID	u32	rw	0x38E	• PDO ist gültig (Bit 31 = 0) • CAN ID des 7. TX PDOs
	2	Trans Type	u8	rw	0xFF	zulässige Werte wie bei TX PDO1
1807	0	Transmit PDO 8	u8	ro	0x02	Anzahl der Einträge TX PDO 8
	1	COB ID	u32	rw	0x38F	• PDO ist gültig (Bit 31 = 0) • CAN ID des 8. TX PDOs
	2	Trans Type	u8	rw	0xFF	zulässige Werte wie bei TX PDO1

Index	S-Idx	Name	Typ	Default	Beschreibung
1A00	0	Mapping Transmit PDO 1	u32 ro	0x04	Anzahl der eingebundenen Appl. Objekte
	1	Index im Objekverzeichniss	u32 ro	0xA100 01	in A100 SIdx 01 steht 1. Word Sendedaten TX PDO 1
	2	Index im Objekverzeichniss	u32 ro	0xA100 02	in A100 SIdx 02 steht 2. Word Sendedaten TX PDO 1
	3	Index im Objekverzeichniss	u32 ro	0xA100 03	in A100 SIdx 03 steht 3. Word Sendedaten TX PDO 1
	4	Index im Objekverzeichniss	u32 ro	0xA100 04	in A100 SIdx 04 steht 4. Word Sendedaten TX PDO 1
1A01	0	Mapping Transmit PDO 2	u32 ro	0x04	Anzahl der eingebundenen Appl. Objekte
	1	Index im Objekverzeichniss	u32 ro	0xA101 01	in A101 SIdx 01 steht 1. Word Sendedaten TX PDO 2
	2	Index im Objekverzeichniss	u32 ro	0xA101 02	in A101 SIdx 02 steht 2. Word Sendedaten TX PDO 2
	3	Index im Objekverzeichniss	u32 ro	0xA101 03	in A101 SIdx 03 steht 3. Word Sendedaten TX PDO 2
	4	Index im Objekverzeichniss	u32 ro	0xA101 04	in A101 SIdx 04 steht 4. Word Sendedaten TX PDO 2
1A02	0	Mapping Transmit PDO 3	u32 ro	0x04	Anzahl der eingebundenen Appl. Objekte
	1	Index im Objekverzeichniss	u32 ro	0xA102 01	in A102 SIdx 01 steht 1. Word Sendedaten TX PDO 3
	2	Index im Objekverzeichniss	u32 ro	0xA102 02	in A102 SIdx 02 steht 2. Word Sendedaten TX PDO 3
	3	Index im Objekverzeichniss	u32 ro	0xA102 03	in A102 SIdx 03 steht 3. Word Sendedaten TX PDO 3
	4	Index im Objekverzeichniss	u32 ro	0xA102 04	in A102 SIdx 04 steht 4. Word Sendedaten TX PDO 3
1A03	0	Mapping Transmit PDO 4	u32 ro	0x04	Anzahl der eingebundenen Appl. Objekte
	1	Index im Objekverzeichniss	u32 ro	0xA103 01	in A103 SIdx 01 steht 1. Word Sendedaten TX PDO 4
	2	Index im Objekverzeichniss	u32 ro	0xA103 02	in A103 SIdx 02 steht 2. Word Sendedaten TX PDO 4
	3	Index im Objekverzeichniss	u32 ro	0xA103 03	in A103 SIdx 03 steht 3. Word Sendedaten TX PDO 4
	4	Index im Objekverzeichniss	u32 ro	0xA103 04	in A103 SIdx 04 steht 4. Word Sendedaten TX PDO 4
1A04	0	Mapping Transmit PDO 4	u32 ro	0x04	Anzahl der eingebundenen Appl. Objekte
	1	Index im Objekverzeichniss	u32 ro	0xA104 01	in A104 SIdx 01 steht 1. Word Sendedaten TX PDO 5
	2	Index im Objekverzeichniss	u32 ro	0xA104 02	in A104 SIdx 02 steht 2. Word Sendedaten TX PDO 5
	3	Index im Objekverzeichniss	u32 ro	0xA104 03	in A104 SIdx 03 steht 3. Word Sendedaten TX PDO 5
	4	Index im Objekverzeichniss	u32 ro	0xA104 04	in A104 SIdx 04 steht 4. Word Sendedaten TX PDO 5

Index	S-Idx	Name	Typ	Default	Beschreibung
1A05	0	Mapping Transmit PDO 6	u32 ro	0x04	Anzahl der eingebundenen Appl. Objekte
	1	Index im Objekverzeichniss	u32 ro	0xA105 01	in A105 SIdx 01 steht 1. Word Sendedaten TX PDO 6
	2	Index im Objekverzeichniss	u32 ro	0xA105 02	in A105 SIdx 02 steht 2. Word Sendedaten TX PDO 6
	3	Index im Objekverzeichniss	u32 ro	0xA105 03	in A105 SIdx 03 steht 3. Word Sendedaten TX PDO 6
	4	Index im Objekverzeichniss	u32 ro	0xA105 04	in A105 SIdx 04 steht 4. Word Sendedaten TX PDO 6
1A06	0	Mapping Transmit PDO 7	u32 ro	0x04	Anzahl der eingebundenen Appl. Objekte
	1	Index im Objekverzeichniss	u32 ro	0xA106 01	in A106 SIdx 01 steht 1. Word Sendedaten TX PDO 7
	2	Index im Objekverzeichniss	u32 ro	0xA106 02	in A106 SIdx 02 steht 2. Word Sendedaten TX PDO 7
	3	Index im Objekverzeichniss	u32 ro	0xA106 03	in A106 SIdx 03 steht 3. Word Sendedaten TX PDO 7
	4	Index im Objekverzeichniss	u32 ro	0xA106 04	in A106 SIdx 04 steht 4. Word Sendedaten TX PDO 7
1A07	0	Mapping Transmit PDO 8	u32 ro	0x04	Anzahl der eingebundenen Appl. Objekte
	1	Index im Objekverzeichniss	u32 ro	0xA107 01	in A107 SIdx 01 steht 1. Word Sendedaten TX PDO 8
	2	Index im Objekverzeichniss	u32 ro	0xA107 02	in A107 SIdx 02 steht 2. Word Sendedaten TX PDO 8
	3	Index im Objekverzeichniss	u32 ro	0xA107 03	in A107 SIdx 03 steht 3. Word Sendedaten TX PDO 8
	4	Index im Objekverzeichniss	u32 ro	0xA107 04	in A107 SIdx 04 steht 4. Word Sendedaten TX PDO 8

Anhang 1.5.2. Bereich herstellerspezifische Daten, Index 2000 bis 5FFF

Index	S-Idx	Name	Typ	Default	Beschreibung
2000	0	Retain Data	dom. rw	0x0	Maximal 256 Byte Daten, die im Retain-Merkerbereich zwischen %MW0 ... %MW127 abgelegt werden.
20F0	0	Einstellung Node ID	u8 rw	0x20	Node ID unter dem das E/A-Modul im CANopen Netz angesprochen wird.
20F1	0	Einstellung Node ID	u8 rw	0x20	Eine Änderung wird nur dann übernommen, wenn in den Einträgen 20F0 und 20F1 der gleiche geänderte Wert eingetragen ist.Die Änderung ist sofort gültig
20F2	0	Einstellung Baud Rate	u8 rw	0x3	Baud Rate des CAN-Netzes Eintrag 0 => 1000 kBaud Eintrag 1 => 500 kBaud Eintrag 2 => 250 kBaud Eintrag 3 => 125 kBaud Eintrag 4 => 100 kBaud Eintrag 5 => 50 kBaud Eintrag 6 => 20 kBaud Eintrag 7 => 10 kBaud
20F3	0	Einstellung Baud Rate	u8 rw	0x3	Eine Änderung wird nur dann übernommen, wenn in den Einträgen 20F2 und 20F3 der gleiche geänderte Wert eingetragen ist. Eine Änderung ist erst nach einem Reset gültig.

Anhang 1.5.3. Legende zum Objektverzeichnis

Abkürzung	Name	Erklärung
u8	unsigned 8	Datenlänge 1 Byte, ohne Vorzeichen
u16	unsigned 16	Datenlänge 2 Byte, ohne Vorzeichen
u32	unsigned 32	Datenlänge 4 Byte, ohne Vorzeichen
dom	domain	Datenlänge variabel
ro	read only	Werte können nur gelesen werden
rw	read write	Werte können gelesen und geschrieben werden
0x....	.... Hex	Zahlenwert hexadezimal dargestellt
0b....	.... binär	Zahlenwert als Dual-/Binärzahl dargestellt

Index und Sub-Indes (S-Idx) des Objektverzeichnis werden als Hex-Wert dargestellt.

Anhang 2. Anschlußbelegung