ifm electronic

CE

ecomat mobile

Systemhandbuch
SmartController

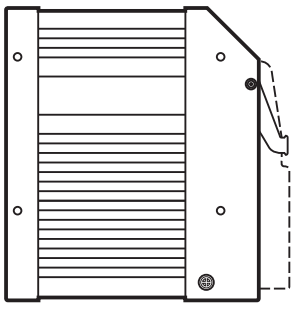
ecomat100

CR2500

CoDeSys® V2.3

Target V05

7390675 / 00 10 / 2009



Inhalt

Inhalt

1 Über diese Anleitung

Inhalt:

Im ergänzenden "Programmierhandbuch CoDeSys® V2.3" erhalten Sie weitergehende Informationen über die Nutzung des Programmiersystems "CoDeSys for Automation Alliance™ ". Dieses Handbuch steht auf der **ifm**-Homepage als kostenloser Download zur Verfügung:
→ www.ifm.com > Land/Sprache wählen > [Service] > [Download] > [Steuerungssysteme]
→ **ifm**-CD "Software, tools and documentation"

Niemand ist vollkommen. Wenn Sie uns Verbesserungsvorschläge zu dieser Anleitung melden, erhalten Sie von uns ein kleines Geschenk als Dankeschön.

1.1 Was bedeuten die Symbole und Formatierungen?

Folgende Symbole oder Piktogramme verdeutlichen Ihnen unsere Hinweise in unseren Anleitungen:

⚠ GEFAHR

Tod oder schwere irreversible Verletzungen sind zu erwarten.

⚠ WARNUNG

Tod oder schwere irreversible Verletzungen sind möglich.

⚠ VORSICHT

Leichte reversible Verletzungen sind möglich.

ACHTUNG

Sachschaden ist zu erwarten oder möglich.

HINWEIS

Wichtige Hinweise auf Fehlfunktionen oder Störungen.

Info

Weitere Hinweise.

► ...	Handlungsaufforderung
> ...	Reaktion, Ergebnis
→ ...	"siehe"
abc	Querverweis
[...]	Bezeichnung von Tasten, Schaltflächen oder Anzeigen

1.2 Wie ist diese Anleitung aufgebaut?

Diese Anleitung ist eine Kombination aus verschiedenen Anleitungstypen. Sie ist eine Lernanleitung für den Einsteiger, aber gleichzeitig auch eine Nachschlageanleitung für den versierten Anwender.

Und so finden Sie sich zurecht:

- Um gezielt zu einem bestimmten Thema zu gelangen, benutzen Sie bitte das Inhaltsverzeichnis.
- Am Anfang eines Kapitels geben wir Ihnen eine kurze Übersicht über dessen Inhalt.
- Abkürzungen und Fachbegriffe stehen im Glossary.
- Die Druckversion der Anleitung enthält im Anhang einen Suchindex.

Im Übrigen behalten wir uns Änderungen vor, so dass sich Abweichungen vom Inhalt der vorliegenden Anleitung ergeben können. Die aktuelle Version finden Sie auf der **ifm**-Homepage:
→ www.ifm.com > Land/Sprache wählen > [Service] > [Download] > [Steuerungssysteme]

Bei Fehlfunktionen oder Unklarheiten setzen Sie sich bitte mit dem Hersteller in Verbindung:
→ www.ifm.com > Land/Sprache wählen > [Kontakt].

2 Sicherheitshinweise

Inhalt:

2.1 Allgemein

Mit den in dieser Anleitung gegebenen Informationen, Hinweisen und Beispielen werden keine Eigenschaften zugesichert. Die abgebildeten Zeichnungen, Darstellungen und Beispiele enthalten weder Systemverantwortung noch applikationsspezifische Besonderheiten.

Die Sicherheit der Maschine/Anlage muss auf jeden Fall eigenverantwortlich durch den Hersteller der Maschine/Anlage gewährleistet werden.

⚠ WARNUNG

Sach- oder Körperschäden möglich bei Nichtbeachten der Hinweise in dieser Anleitung!
Die **ifm electronic gmbh** übernimmt hierfür keine Haftung.

- ► Die handelnde Person muss vor allen Arbeiten an und mit diesem Gerät die Sicherheitshinweise und die betreffenden Kapitel dieser Anleitung gelesen und verstanden haben.
- ► Die handelnde Person muss zu Arbeiten an der Maschine/Anlage autorisiert sein.
- ► Beachten Sie die Technischen Daten der betroffenen Geräte!
  Das aktuelle Datenblatt finden Sie auf der **ifm**-Homepage:
  → www.ifm.com > Land/Sprache wählen > [Datenblatt-Suche] > (Artikel-Nr.) > [Technische Daten im PDF-Format]
- ► Beachten Sie die Montage- und Anschlussbedingungen sowie die bestimmungsgemäße Verwendung der betroffenen Geräte!
  → mitgelieferte Montageanleitung oder auf der **ifm**-Homepage:
  → www.ifm.com > Land/Sprache wählen > [Datenblatt-Suche] > (Artikel-Nr.) > [Betriebsanleitungen]

ACHTUNG

Der Treiberbaustein der seriellen Schnittstelle kann beschädigt werden!

Beim Trennen der seriellen Schnittstelle unter Spannung kann es zu undefinierten Zuständen kommen, die zu einer Schädigung des Treiberbausteins führen.

- ► Die serielle Schnittstelle nur im spannungslosen Zustand trennen!

Anlaufverhalten der Steuerung

Der Hersteller der Maschine/Anlage muss mit seinem Applikations-Programm gewährleisten, dass beim Anlauf oder Wiederanlauf der Steuerung keine gefahrbringenden Bewegungen gestartet werden können.

Ein Wiederanlauf kann z.B. verursacht werden durch:

- Spannungswiederkehr nach Spannungsausfall
- Reset nach Watchdog-Ansprechen wegen zu langer Zykluszeit

2.2 Welche Vorkenntnisse sind notwendig?

Das Dokument richtet sich an Personen, die über Kenntnisse der Steuerungstechnik und SPS-Programmierkenntnisse mit IEC 61131-3 sowie der Software CoDeSys® verfügen.

Das Dokument richtet sich an Fachkräfte. Dabei handelt es sich um Personen, die aufgrund ihrer einschlägigen Ausbildung und ihrer Erfahrung befähigt sind, Risiken zu erkennen und mögliche Gefährdungen zu vermeiden, die der Betrieb oder die Instandhaltung eines Produkts verursachen kann. Das Dokument enthält Angaben zum korrekten Umgang mit dem Produkt.

Lesen Sie dieses Dokument vor dem Einsatz, damit Sie mit Einsatzbedingungen, Installation und Betrieb vertraut werden. Bewahren Sie das Dokument während der gesamten Einsatzdauer des Gerätes auf.

Befolgen Sie die Sicherheitshinweise.

3 Systembeschreibung

Inhalt:

3.1 Angaben zum Gerät

Diese Anleitung beschreibt die Controller-Gerätefamilie **ecomat*mobil*** der **ifm electronic gmbh** mit 16 Bit Mikrocontroller für den mobilen Einsatz:

- SmartController: CR2500

3.2 Angaben zur Software

Der Controller arbeitet mit CoDeSys® ab Version 2.3.9.1.

Im "Programmierhandbuch CoDeSys® 2.3" erhalten Sie weitergehende Informationen über die Nutzung des Programmiersystems "CoDeSys for Automation Alliance". Dieses Handbuch steht auf der **ifm**-Internetseite als kostenloser Download zur Verfügung:
→ www.ifm.com > Land/Sprache wählen > [Service] > [Download] > [Steuerungssysteme]
→ **ifm**-CD "Software, tools and documentation"

Die Applikationssoftware kann vom Anwender komfortabel mit dem Programmiersystem CoDeSys® selbst erstellt werden.

Der Anwender muss außerdem beachten, welcher Softwarestand (speziell beim R360-Betriebssystem und den Funktionsbibliotheken) zum Einsatz kommt.

HINWEIS

Es müssen immer die zum gewählten Target passenden Software-Stände zum Einsatz kommen:

- des Betriebssystems (`CRnnnn_Vxxyyyzz.H86`),
- der Steuerungskonfiguration (`CRnnnn_Vxx.CFG`),
- der Gerätebibliothek (`CRnnnn_Vxxyyyzz.LIB`)
- und der weiteren Dateien
  (→ Kapitel Übersicht der verwendeten Dateien und Bibliotheken, Seite 288).

CRnnnn	Geräte-Artikelnummer
Vxx: 00...99	Target-Versionsnummer
yy: 00...99	Release-Nummer
zz: 00...99	Patch-Nummer

Dabei müssen der Basisdateiname (z.B. "`CR0032`") und die Software-Versionsnummer "xx" (z.B. "`02`") überall den gleichen Wert haben! Andernfalls geht der Controller in den STOPP-Zustand

Die Werte für "yy" (Release-Nummer) und "zz" (Patch-Nummer) müssen **nicht** übereinstimmen.

Außerdem beachten: Folgende Dateien müssen ebenfalls geladen sein:

- die zum Projekt erforderlichen internen Bibliotheken (in IEC1131 erstellt),
- die Konfigurationsdateien (`*.CFG`)
- und die Target-Dateien (`*.TRG`).

Außerdem beachten:

Das Target für CRnn32 muss ≥ V02 sein, für alle übrigen Geräte ≥ V05.

Für die sichere Funktion der Applikations-Programme, 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.

3.3 Steuerungskonfiguration

Bei dem Steuerungssystem ecomat*mobil* handelt es sich um ein Gerätekonzept für den Serieneinsatz. Das bedeutet, dass die Controller optimal auf den jeweiligen Einsatzfall konfiguriert werden können. Wenn notwendig, können auch Sonderfunktionen und spezielle Hardwarelösungen realisiert werden. Zusätzlich kann auch die aktuelle Version der ecomat*mobil*-Software über www.ifm.com aus dem Internet geladen werden.

Ob bestimmte in der Dokumentation beschriebene Funktionen, Hardwareoptionen, Ein- und Ausgänge in der betreffenden Hardware verfügbar sind, muss in jedem Fall vor Einsatz der Controller überprüft werden.

4 Konfigurationen

Inhalt:

Die in den jeweiligen Montage- und Installationsanweisungen oder dem Anhang (→ Seite 285) dieser Dokumentation beschriebenen Gerätekonfigurationen stehen als Standardgeräte (Lagerware) zur Verfügung. Diese decken bei den meisten Applikationen die geforderten Spezifikationen ab.

Entsprechend den Kundenanforderungen bei Serieneinsatz ist es aber auch möglich, dass andere Gerätekonfigurationen z.B. hinsichtlich der Zusammenstellung der Ein- und Ausgänge und der Ausführung der Analogkanäle eingesetzt werden.

⚠ WARNUNG

Sach- oder Körperschäden möglich durch Fehlfunktionen!

Die in dieser Dokumentation beschriebenen Softwarefunktionen gelten nur für die Standardkonfigurationen. Bei Einsatz von kundenspezifischen Geräten:

► die besonderen Hardwareausführungen und zusätzlichen Hinweise (Zusatzdokumentation) zum Einsatz der Software beachten.

Installieren der Dateien und Bibliotheken im Gerät:

Werkseinstellung: Das Gerät enthält nur den Bootloader.

► Betriebssystem (*.H86) laden.

► Projekt (*.PRO) im PC anlegen: Target (*.TRG) eintragen.

► (zusätzlich bei Targets vor V05:) Steuerungskonfiguration (*.CFG) festlegen.

> CoDeSys® bindet die zum Target zugehörenden Dateien in das Projekt ein: *.TRG, *.CFG, *.CHM, *.INI, *.LIB.

► Bei Bedarf das Projekt mit weiteren Bibliotheken (*.LIB) ergänzen.

Bestimmte Bibliotheken binden automatisch weitere Bibliotheken in das Projekt ein:
z.B. basieren einige Funktionen in **ifm**-Bibliotheken (ifm_*.LIB) auf Funktionen in CoDeSys®-Bibliotheken (3S_*.LIB).

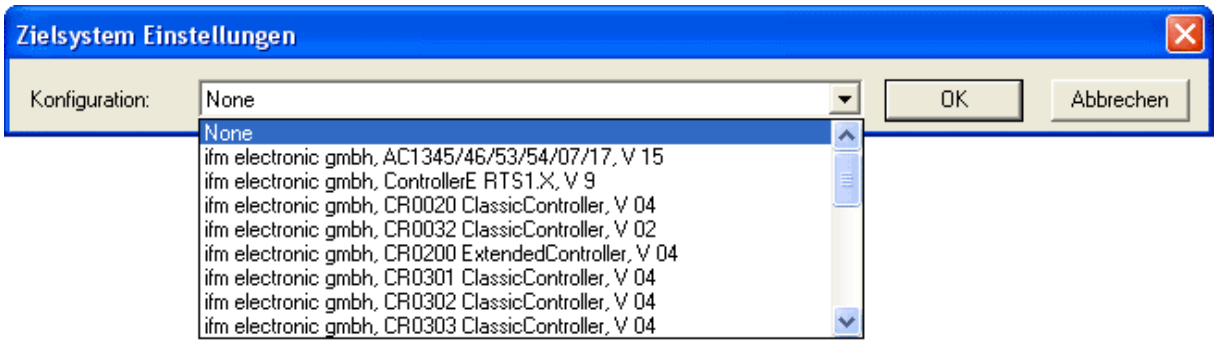
4.1 Programmiersystem einrichten

Inhalt:

4.1.1 Programmiersystem manuell einrichten

Target einrichten

Beim Erstellen eines neuen Projektes in CoDeSys® muss die dem Controller entsprechende Target-Datei geladen werden. Sie wird im Dialogfenster für jede Hardware gewählt und stellt für das Programmiersystem die Schnittstelle zur Hardware her.



Grafik: Zielsystem Einstellungen

Gleichzeitig werden mit Auswahl des Targets alle wichtigen Bibliotheken und die Steuerungskonfiguration geladen. Diese können vom Programmierer bei Bedarf wieder entfernt oder durch weitere Bibliotheken ergänzt werden.

HINWEIS

Es müssen immer die zum gewählten Target passenden Software-Stände zum Einsatz kommen:

- des Betriebssystems (`CRnnnn_Vxxyyyzz.H86`),
- der Steuerungskonfiguration (`CRnnnn_Vxx.CFG`),
- der Gerätebibliothek (`CRnnnn_Vxxyyyzz.LIB`)
- und der weiteren Dateien
  (→ Kapitel Übersicht der verwendeten Dateien und Bibliotheken, Seite 288).

CRnnnn	Geräte-Artikelnummer
Vxx: 00...99	Target-Versionsnummer
yy: 00...99	Release-Nummer
zz: 00...99	Patch-Nummer

Dabei müssen der Basisdateiname (z.B. "`CR0032`") und die Software-Versionsnummer "xx" (z.B. "`02`") überall den gleichen Wert haben! Andernfalls geht der Controller in den STOPP-Zustand

Die Werte für "yy" (Release-Nummer) und "zz" (Patch-Nummer) müssen **nicht** übereinstimmen.

Außerdem beachten: Folgende Dateien müssen ebenfalls geladen sein:

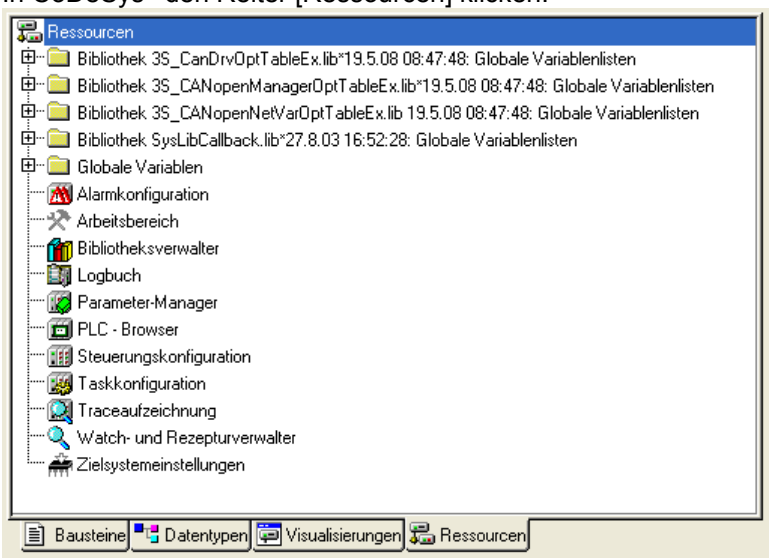
- die zum Projekt erforderlichen internen Bibliotheken (in IEC1131 erstellt),
- die Konfigurationsdateien (`*.CFG`)
- und die Target-Dateien (`*.TRG`).

Steuerungskonfiguration aktivieren

Bei der Konfiguration des Programmiersystems (→ vorheriger Abschnitt) erfolgte automatisch auch die Steuerungskonfiguration.

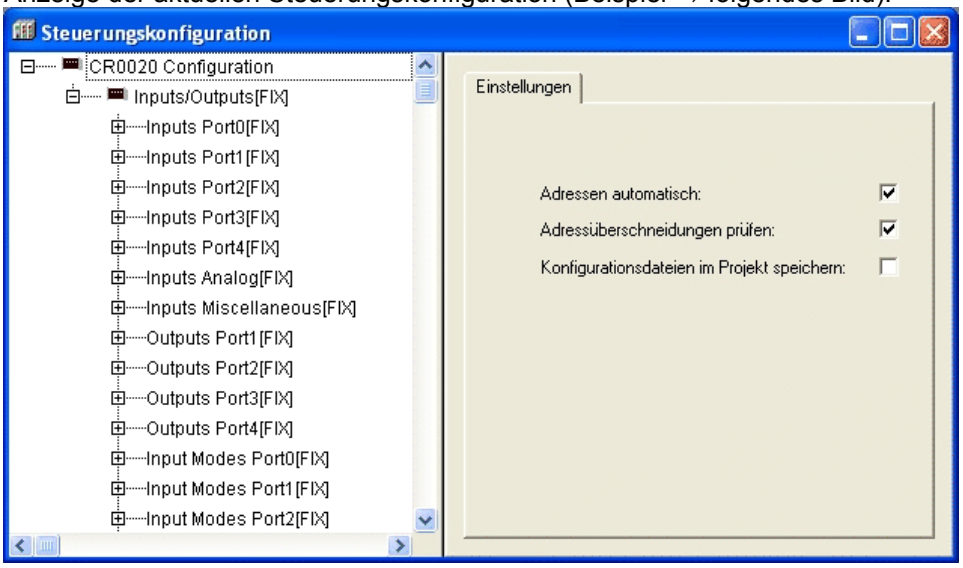
Den Punkt [Steuerungskonfiguration] erreicht man über den Reiter [Ressourcen]. Über einen Doppelklick auf den Punkt [Steuerungskonfiguration] öffnet sich das entsprechende Fenster.

► In CoDeSys® den Reiter [Ressourcen] klicken:



► In der linken Spalte Doppelklick auf [Steuerungskonfiguration]

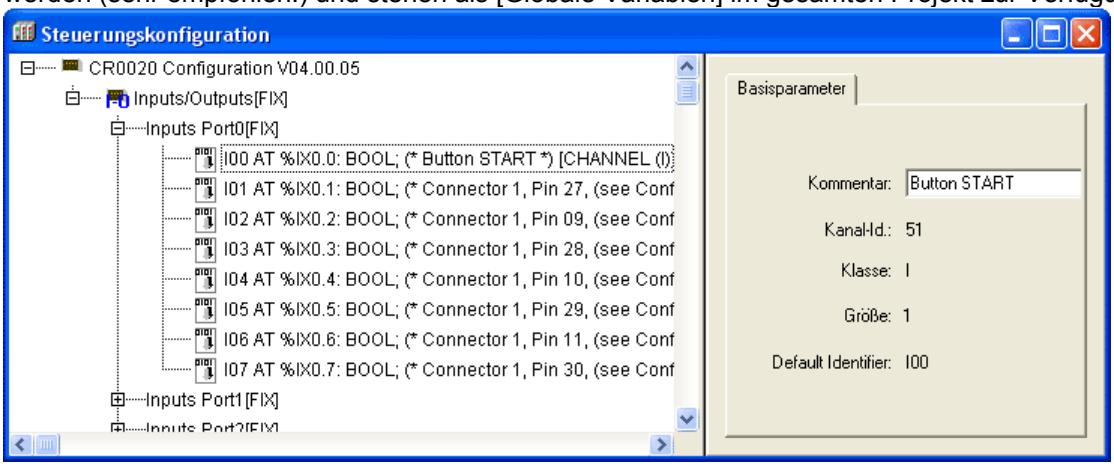
> Anzeige der aktuellen Steuerungskonfiguration (Beispiel → folgendes Bild):



Durch die Konfiguration erhält der Anwender in der Programmumgebung Folgendes verfügbar:

- alle wichtigen System- und Fehlermerker
  Je nach Anwendung und Applikations-Programm müssen diese Merker bearbeitet und ausgewertet werden. Der Zugriff erfolgt über deren symbolischen Namen.
- die Struktur der Ein- und Ausgänge
  Diese können im Fenster [Steuerungskonfiguration] (→ Bild unten) direkt symbolisch bezeichnet

werden (sehr empfohlen!) und stehen als [Globale Variablen] im gesamten Projekt zur Verfügung.



4.1.2 Programmiersystem über Templates einrichten

Inhalt:

ifm bietet vorgefertigte Templates (Programm-Vorlagen), womit Sie das Programmiersystem schnell, einfach und vollständig einrichten können.

HINWEIS

Beim Installieren der ecomat*mobil*-CD "Software, Tools and Documentation" wurden auch Projekte mit Vorlagen auf Ihrem Computer im Programmverzeichnis abgelegt:
`…\ifm electronic\CoDeSys V…\Projects\Template_CDV…`

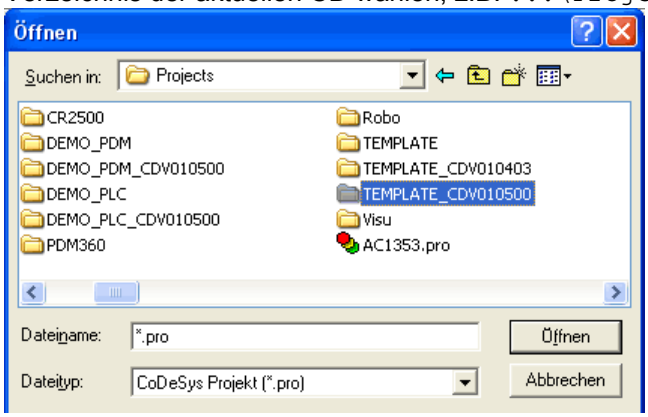
► Die gewünschte dort gespeicherte Vorlage in CoDeSys® öffnen mit:
[Datei] > [Neu aus Vorlage…]

> CoDeSys® legt ein neues Projekt an, dem der prinzipielle Programmaufbau entnommen werden kann. Es wird dringend empfohlen, dem gezeigten Schema zu folgen.
→ Kapitel Programmiersystem über Templates einrichten, Seite 16

Wie richten Sie das Programmiersystem schnell und einfach ein?

► Im CoDeSys-Menü wählen: [Datei] > [Neu aus Vorlage...].

► Verzeichnis der aktuellen CD wählen, z.B. `...\Projects\TEMPLATE_CDV010500`:

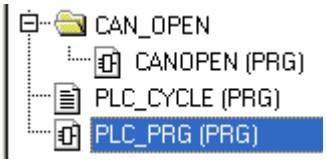
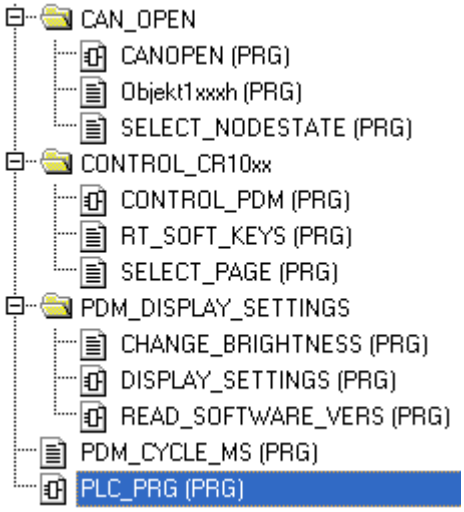


► Artikelnummer des Geräts in der LIste suchen, z.B. CR2500 als CANopen-Master:

► Wie ist das CAN-Netzwerk organisiert?
Soll auf Layer2-Basis gearbeitet werden oder gibt es (mit CANopen) einen Master mit mehreren Slaves?
(Hier im Beispiel: CANopen-Slave, → Bild oben)

► Wahl mit [Öffnen] bestätigen.

> Neues CoDeSys-Projekt wird angelegt mit zunächst folgender Ordnerstruktur (links):

Beispiel für CR2500 als CANopen-Master:	Anderes Beispiel für CR1051 als CANopen-Slave:
	

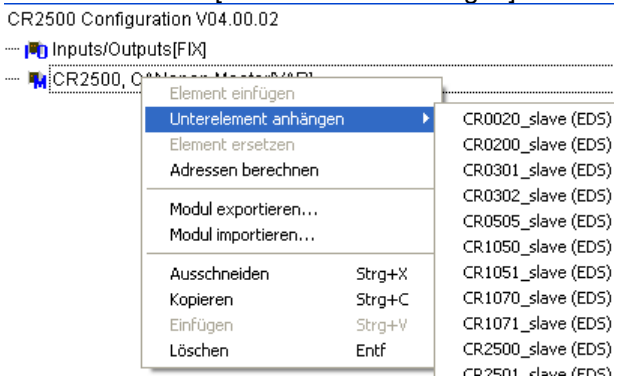
> (Über die Ordnerstrukturen in Templates → Kapitel Über die ifm-Templates, Seite 18).

► Das neue Projekt speichern mit [Datei] > [Speichern unter...], dabei geeignetes Verzeichnis und Projektnamen festlegen.

► Das CAN-Netzwerk im Projekt konfigurieren:
Im CoDeSys-Projekt über dem Tabulator []Ressourcen] das Element [Steuerungskonfiguration] doppelklicken.

► Mit **rechter** Maustaste in den Eintrag [CR2500, CANopen Master] klicken.

► Im Kontext-Menü [Unterelement anhängen] klicken:



> Im ergänzten Kontextmenü erscheint eine Liste aller verfügbaren EDS-Dateien.

► Gewünschtes Element wählen, z.B. "System R360: I/O CompactModule CR2011 (EDS)".
Die EDS-Dateien liegen im Verzeichnis `C:\...\CoDeSys V...\Library\PLCConf\`.

> Das Fenster [Steuerungskonfiguration] ändert sich wie folgt:

► Für den eingetragenen Slave den Erfordernissen entsprechend die CAN-Parameter, das PDO-Mapping und die SDOs einstellen. Hinweis: [alle SDOs erzeugen] besser abwählen.

► Mit weiteren Slaves sinngemäß wie vorstehend verfahren.

► Projekt speichern!

Damit ist das Netzwerk Ihres Projekts hinreichend beschrieben. Sie wollen dieses Projekt mit weiteren Elementen und Funktionen ergänzen?
→ Kapitel Projekt mit weiteren Funktionen ergänzen, Seite 23

Über die ifm-Templates

In der Regel werden für jedes Gerät folgende Templates angeboten:

- `ifm_template_CRnnnnLayer2_Vxxyyzz.pro` für den Betrieb des Geräts mit CAN Layer 2
- `ifm_template_CRnnnnMaster_Vxxyyzz.pro` für den Betrieb des Geräts als CAN-Master
- `ifm_template_CRnnnnSlave_Vxxyyzz.pro` für den Betrieb des Geräts als CAN-Slave

Die hier beschriebenen Templates gelten für:
- **CoDeSys** ab Version 2.3.9.6
- auf der ecomat*mobil*-CD ab Version 010500

Die Templates enthalten alle die gleichen Strukturen.

Mit dieser Auswahl der Programm-Vorlage für den CAN-Betrieb ist bereits eine wichtige Grundlage für ein funktionsfähiges Programm geschaffen.

Ordner-Struktur, allgemein

Die Bausteine sind sortiert in die folgenden Ordner:

Ordner	Beschreibung
CAN_OPEN	für Controller und PDM, CAN-Betrieb als Master oder Slave: Enthält die Funktionen für CANopen.
I_O_CONFIGURATION	für Controller, CAN-Betrieb mit Layer 2 oder als Master oder als Slave: Funktionen zum Parametrieren der Betriebsarten der Ein- und Ausgänge.
PDM_COM_LAYER2	für Controller, CAN-Betrieb als Layer 2 oder Slave: Funktionen zur Basiskomunikation über Layer2 zwischen PLC und PDM.
CONTROL_CR10nn	für PDM, CAN-Betrieb mit Layer 2 oder als Master oder als Slave: Enthält Funktionen zur Bild- und Tastensteuerung im laufenden Betrieb.
PDM_DISPLAY_SETTINGS	für PDM, CAN-Betrieb mit Layer 2 oder als Master oder als Slave: Enthält Funktionen zum Einstellen des Monitors.

Programme und Funktionen in den Ordnern der Templates

Die vorgenannten Ordner enthalten die folgenden Programme und Funktionen (=Bausteine):

Bausteine im Ordner CAN_OPEN	Beschreibung
CANOPEN	für Controller und PDM, CAN-Betrieb als Master: Enthält folgende parametrierte Bausteine: - CAN1_MASTER_EMCY_HANDLER (→ Funktion CANx_MASTER_EMCY_HANDLER, Seite 120), - CAN1_MASTER_STATUS (→ Funktion CANx_MASTER_STATUS, Seite 125), - SELECT_NODESTATE (→ unten).
CANOPEN	für Controller und PDM, CAN-Betrieb als Slave: Enthält folgende parametrierte Bausteine: - CAN1_SLAVE_EMCY_HANDLER (→ Funktion CANx_SLAVE_EMCY_HANDLER, Seite 134), - CAN1_SLAVE_STATUS (→ Funktion CANx_SLAVE_STATUS, Seite 139), - SELECT_NODESTATE (→ unten).
Objekt1xxxh	für Controller und PDM, CAN-Betrieb als Slave: Enthält die Werte [STRING] zu folgenden Parametern: - ManufacturerDeviceName, z.B.: 'CR1051' - ManufacturerHardwareVersion, z.B.: 'HW_Ver 1.0' - ManufacturerSoftwareVersion, z.B.: 'SW_Ver 1.0'

Bausteine im Ordner CAN_OPEN	Beschreibung
SELECT_NODESTATE	für PDM, CAN-Betrieb als Master oder als Slave: Wandelt den Wert des Knoten-Status [BYTE] in den zugehörigen Text [STRING]: 4 → 'STOPPED' 5 → 'OPERATIONAL' 127 → 'PRE-OPERATIONAL'

Bausteine im Ordner I_O_CONFIGURATION	Beschreibung
CONF_IO_CRnnnn	für Controller, CAN-Betrieb mit Layer 2 oder als Master oder als Slave: Parametriert die Betriebsarten der Ein- und Ausgänge.

Bausteine im Ordner PDM_COM_LAYER2	Beschreibung
PLC_TO_PDM	für Controller, CAN-Betrieb mit Layer 2 oder als Slave: Organisiert die Kommunikation vom Controller zum PDM: - überwacht die Übertragungszeit, - überträgt Steuerdaten für Bildwechsel, LEDs, Eingabewerte usw.
TO_PDM	für Controller, CAN-Betrieb mit Layer 2 oder als Slave: Organisiert die Signale für LEDs und Tasten zwischen Controller und PDM. Enthält folgende parametrierte Bausteine: - PACK (→ 3S), - PLC_TO_PDM (→ oben), - UNPACK (→ 3S).

Bausteine im Ordner CONTROL_CR10nn	Beschreibung
CONTROL_PDM	für PDM, CAN-Betrieb mit Layer 2 oder als Master oder als Slave: Organisiert die Bildsteuerung im PDM. Enthält folgende parametrierte Bausteine: - PACK (→ 3S), - PDM_MAIN_MAPPER, - PDM_PAGECONTROL, - PDM_TO_PLC (→ unten), - SELECT_PAGE (→ unten).

Bausteine im Ordner CONTROL_CR10nn	Beschreibung
PDM_TO_PLC	für PDM, CAN-Betrieb mit Layer 2: Organisiert die Kommunikation vom PDM zum Controller: - überwacht die Übertragungszeit, - überträgt Steuerdaten für Bildwechsel, LEDs, Eingabewerte usw. Enthält folgende parametrierte Bausteine: - CAN_1_TRANSMIT, - CAN_1_RECEIVE.
RT_SOFT_KEYS	für PDM, CAN-Betrieb mit Layer 2 oder als Master oder als Slave: Liefert von den (virtuellen) Tasten-Signalen im PDM die steigenden Flanken. Es können beliebige Variablen (als virtuelle Tasten) auf die globalen Variablen SoftKeyGlobal gemappt werden, wenn z.B. ein Programmteil von einem CR1050 in ein CR1055 kopiert werden soll. Dort gibt es nur die Tasten F1...F3: → Für die virtuellen Tasten F4...F6 Variablen erzeugen. Diese selbst erzeugten Variablen hier auf die globalen Softkeys mappen. Im Programm nur mit den globalen Softkeys arbeiten. Vorteil: Anpassungsarbeiten sind nur an **einer** Stelle erforderlich.
SELECT_PAGE	für PDM, CAN-Betrieb mit Layer 2 oder als Master oder als Slave: Organisiert die Wahl der Visualisierungen. Enthält folgende parametrierte Bausteine: - RT_SOFT_KEYS (→ oben).

Bausteine im Ordner PDM_DISPLAY_SETTINGS	Beschreibung
CHANGE_BRIGHTNESS	für PDM, CAN-Betrieb mit Layer 2 oder als Master oder als Slave: Organisiert Helligkeit / Kontrast des Monitors.
DISPLAY_SETTINGS	für PDM, CAN-Betrieb mit Layer 2 oder als Master oder als Slave: Stellt die Echtzeituhr, steuert Helligkeit / Kontrast des Monitors, zeigt die Software-Version. Enthält folgende parametrierte Bausteine: - CHANGE_BRIGHTNESS (→ oben), - CurTimeEx (→ 3S), - PDM_SET_RTC, - READ_SOFTWARE_VERS (→ unten), - TP (→ 3S).
READ_SOFTWARE_VERS	für PDM, CAN-Betrieb mit Layer 2 oder als Master oder als Slave: Zeigt die Software-Version. Enthält folgende parametrierte Bausteine: - DEVICE_KERNEL_VERSION1, - DEVICE_RUNTIME_VERSION, - LEFT (→ 3S).

Bausteine im Wurzel-Verzeichnis	Beschreibung
PLC_CYCLE	für Controller, CAN-Betrieb mit Layer 2 oder als Master oder als Slave: Ermittelt die Zykluszeit der SPS im Gerät.
PDM_CYCLE_MS	für PDM, CAN-Betrieb mit Layer 2 oder als Master oder als Slave: Ermittelt die Zykluszeit der SPS im Gerät.
PLC_PRG	für Controller und PDM, CAN-Betrieb mit Layer 2 oder als Master oder als Slave: Hauptprogramm; hier werden die weiteren Programm-Elemente eingebunden.

Struktur der Visualisierungen in den Templates

(Nur für PDM)

Die Visualisierungen sind wie folgt in Ordnern strukturiert:

Ordner	Bild-Nr.	Beschreibung Inhalt
START_PAGE	P00001	Einstellung / Anzeige von... - Node-ID - CAN-Baudrate - Status - GuardErrorNode - SPS-Zykluszeit
__MAIN_MENUES	P00010	Menübild: - Display-Setup
____MAIN_MENUE_1		
______DISPLAY_SETUP		
________1_DISPLAY_SETUP1	P65000	Menübild: - Software-Version - Helligkeit / Kontrast - Echtzeituhr anzeigen / setzen
__________1_SOFTWARE_VERSION	P65010	Anzeige der Software-Version
__________2_BRIGHTNESS	P65020	Einstellen von Helligkeit / Kontrast
__________3_SET_RTC	P65030	Echtzeituhr anzeigen / setzen

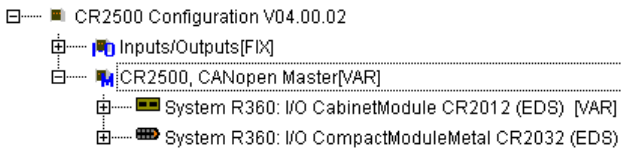
In den Templates haben wir die Bildnummern in 10er-Schritten organisiert. So können Sie mit Hilfe eines Bildnummer-Offsets in verschiedene Sprachversionen der Visualisierungen schalten.

Projekt mit weiteren Funktionen ergänzen

Sie haben ein Projekt mittels eines ifm-Templates angelegt und das CAN-Netzwerk definiert. Nun wollen Sie diesem Projekt weitere Funktionen hinzufügen.

Für das Beispiel nehmen wir einen CabinetController CR2500 als CANopen-Master an, an den ein I/O-CabinetModul CR2011 und ein I/O-Compact-Modul CR2032 als Slaves angeschlossen sind:

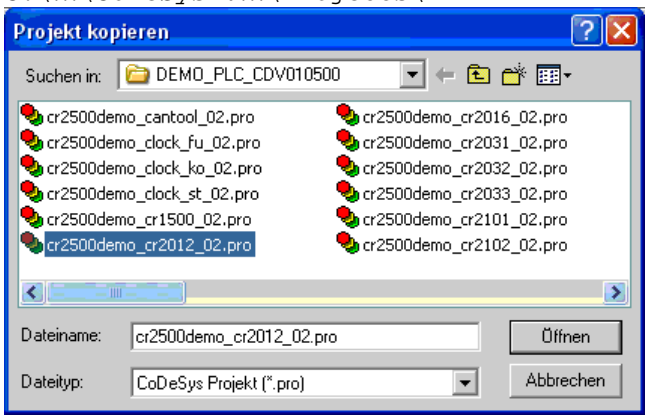
Steuerungskonfiguration:



Am CR2012 sei ein Joystick angeschlossen, der am CR2032 einen PWM-Ausgang ansteuern soll. Wie geht das schnell und einfach?

► CoDeSys-Projekt speichern!

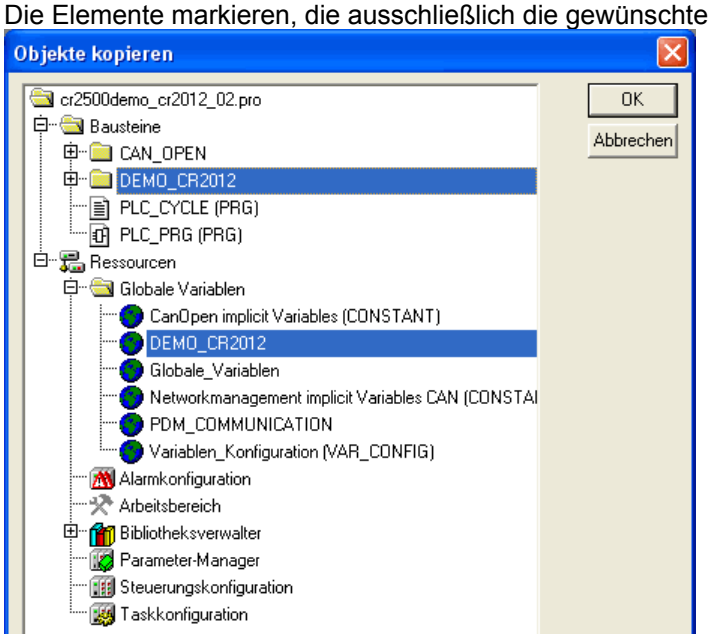
► In CoDeSys mit [Projekt] > [kopieren...] das Projekt öffnen, das die gewünschte Funktion enthält: z.B. `CR2500Demo_CR2012_02.pro` aus dem Verzeichnis `DEMO_PLC_CDV...` unter `C:\...\CoDeSys V...\Projects\`:



► Wahl mit [Öffnen] bestätigen.

> Fenster [Objekte kopieren] erscheint:

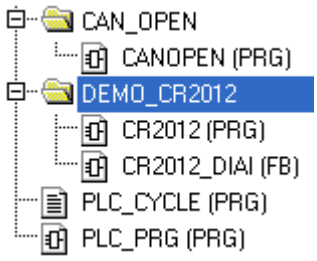
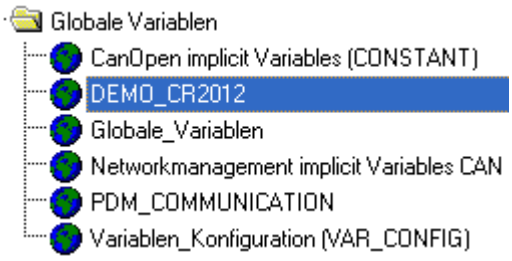
► Die Elemente markieren, die ausschließlich die gewünschte Funktion enthalten, hier z.B.:



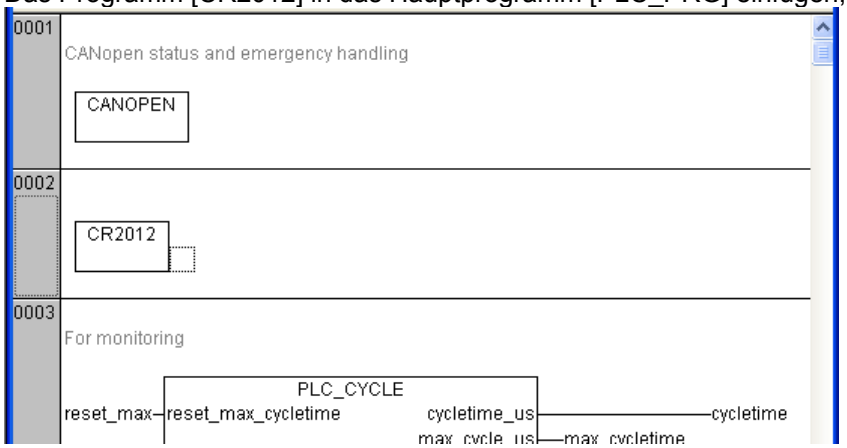
HINWEIS: In anderen Fällen können auch Bibliotheken und / oder Visualisierungen erforderlich sein.

► Wahl mit [OK] bestätigen.

> In unserem Beispiel-Projekt sind die im Demo-Projekt gewählten Elemente hinzugekommen:

Bausteine:	Ressourcen:
	

► Das Programm [CR2012] in das Hauptprogramm [PLC_PRG] einfügen, z.B.:



► In den Kommentaren der Bausteine und Globalen Variablen stehen meist Hinweise, wie bei Bedarf einzelne Elemente daraus konfiguriert, eingeschlossen oder ausgeschlossen werden müssen. Diesen Hinweisen Folge leisten.

► Ein- und Ausgangsvariable sowie CAN-Parameter und ggf. Visualisierungen den eigenen Bedingungen anpassen.

► [Projekt] > [speichern] und
[Projekt] > [Alles übersetzen].

► Nach eventuell erforderlichen Korrekturen und Ergänzen von fehlenden Bibliotheken (→ Fehlermeldungen nach dem Übersetzen) das Projekt nochmals speichern.

► Nach diesem Prinzip schrittweise (!) mit weiteren Funktionen aus anderen Projekten ergänzen und jeweils die Ergebnisse prüfen.

► [Projekt] > [speichern] und
[Projekt] > [Alles übersetzen].

4.1.3 ifm-Demo-Programme

Im Verzeichnis `DEMO_PLC_CDV…` (für Controller) oder `DEMO_PDM_CDV…` (für PDMs) unter `C:\...\CoDeSys V...\Projects\` erklären wir bestimmte Funktionen in getesteten Demo-Programmen. Bei Bedarf können diese Funktionen in eigene Projekte übernommen werden. Die Strukturen und Variablen der **ifm**-Demos passen zu denen in den **ifm**-Templates.

In jedem Demo-Programm wird nur genau **ein** Thema gezeigt. Auch für Controller werden dazu einige Visualisierungen gezeigt, die auf dem PC-Monitor die getestete Funktion anschaulich machen sollen.

Kommentare in den Bausteinen und in den Variablenlisten helfen beim Anpassen der Demos an Ihr Projekt.

Wenn nicht anders angegeben, gelten die Demo-Programme jeweils für alle Controller oder für alle PDMs.

Die hier beschriebenen Demo-Programme gelten für:
- **CoDeSys** ab Version 2.3.9.6
- auf der ecomat*mobil*-CD ab Version 010500

Demo-Programme für Controller

Demo-Programm	Funktion
`CR2500Demo_CanTool_xx.pro`	getrennt für PDM360, PDM360 compact, PDM360 smart und Controller: Enthält Funktionen zum Einstellen und Analysieren der CAN-Schnittstelle.

`CR2500Demo_ClockFu_xx.pro`
`CR2500Demo_ClockKo_xx.pro`
`CR2500Demo_ClockSt_xx.pro`
	Taktgenerator für Controller als Funktion eines Wertes an einem Analog-Eingang:
	Fu = in Funktionsplan
	Ko = in Kontaktplan
	St = in Strukturiertem Text

`CR2500Demo_CR1500_xx.pro`	Anschluss eines Tastatur-Moduls CR1500 als Slave eines Controllers (CANopen-Master).
`CR2500Demo_CR2012_xx.pro`	I/O-Cabinet-Modul CR2012 als Slave eines Controllers (CANopen-Master), Anschluss eines Joysticks mit Richtungsschalter und Referenz-Mittelspannung.

Demo-Programm	Funktion
CR2500Demo_CR2016_xx.pro	I/O-Cabinet-Modul CR2016 als Slave eines Controllers (CANopen-Master), 4x Frequenz-Eingang, 4x Digital-Eingang Highside, 4x Digital-Eingang Lowside, 4x Analog-Eingang ratiometrisch, 4x PWM1000-Ausgang und 12x Digitalausgang.
CR2500Demo_CR2031_xx.pro	I/O-Compact-Modul CR2031 als Slave eines Controllers (CANopen-Master), Strommessung an den PWM-Ausgängen.
CR2500Demo_CR2032_xx.pro	I/O-Compact-Modul CR2032 als Slave eines Controllers (CANopen-Master), 4x Digital-Eingang, 4x Digital-Eingang analog ausgewertet, 4x Digital-Ausgang, 4x PWM-Ausgang.
CR2500Demo_CR2033_xx.pro	I/O-Compact-Modul CR2033 als Slave eines Controllers (CANopen-Master), 4x Digital-Eingang, 4x Digital-Eingang analog ausgewertet, 4x Digital-Ausgang.
CR2500Demo_CR2101_xx.pro	Neigungssensor CR2101 als Slave eines Controllers (CANopen-Master).
CR2500Demo_CR2102_xx.pro	Neigungssensor CR2102 als Slave eines Controllers (CANopen-Master).
CR2500Demo_CR2511_xx.pro	I/O-Smart-Modul CR2511 als Slave eines Controllers (CANopen-Master), 8x PWM-Ausgang stromgeregelt.
CR2500Demo_CR2512_xx.pro	I/O-Smart-Modul CR2512 als Slave eines Controllers (CANopen-Master), 8x PWM-Ausgang. Anzeige des aktuellen Stroms für jedes Kanalpaar.
CR2500Demo_CR2513_xx.pro	I/O-Smart-Modul CR2513 als Slave eines Controllers (CANopen-Master), 4x Digital-Eingang, 4x Digital-Ausgang, 4x Analogeingang 0...10 V.
CR2500Demo_Interrupt_xx.pro	Beispiel mit der Funktion SET_INTERRUPT_XMS (→ Seite 245).
CR2500Demo_Operating_hours_xx.pro	Beispiel für einen Betriebsstundenzähler mit Schnittstelle zu einem PDM.

Demo-Programm	Funktion
`CR2500Demo_PWM_xx.pro`	Wandelt einen Potentiometer-Wert an einem Eingang in einen normierten PWM-Wert an einem Ausgang mit folgenden Bausteinen: - Funktion INPUT_VOLTAGE (→ Seite 265), - Funktion NORM (→ Seite 268), - Funktion PWM100 (→ Seite 172).
`CR2500Demo_RS232_xx.pro`	Beispiel für den Empfang von Daten auf der seriellen Schnittstelle mit Hilfe des Windows-Hyperterminal.

`StartersetDemo.pro`
`StartersetDemo2.pro`
`StartersetDemo2_fertig.pro`
	Verschiedene Übungen zum E-Learning mit dem Starterset EC2074.

_xx = Angabe der Demo-Version

Demo-Programme für PDM

Demo-Programm	Funktion

`CR1051Demo_CanTool_xx.pro`
`CR1053Demo_CanTool_xx.pro`
`CR1071Demo_CanTool_xx.pro`
	getrennt für PDM360, PDM360 compact, PDM360 smart und Controller:
	Enthält Funktionen zum Einstellen und Analysieren der CAN-Schnittstelle.

`CR1051Demo_Input_Character_xx.pro`	Ermöglicht beliebige Zeicheneingabe in eine Zeichenkette: - Großbuchstaben, - Kleinbuchstaben, - Sonderzeichen, - Ziffern. Auswahl der Zeichen mit dem Drehgeber. Beispiel ist auch z.B. für eine Passworteingabe geeignet. Bild P01000: Auswahl und Übernahme von Zeichen
`CR1051Demo_Input_Lib_xx.pro`	Demo der Funktion INPUT_INT aus der Bibliothek `ifm_pdm_input_Vxxyyzz` (mögliche Alternative zum 3S-Standard). Werte wählen und einstellen mittels Drehgeber. Bild P10000: 6 Werte INT Bild P10010: 2 Werte INT Bild P10020: 1 Wert REAL

Demo-Programm	Funktion
`CR1051Demo_Linear_logging_on_flash_intern_xx.pro`	Schreibt einen CSV-Datensatz mit dem Inhalt einer CAN-Nachricht in den internen Flash-Speicher (`/home/project/daten.csv`), wenn [F3] gedrückt wird oder eine CAN- Nachricht auf dem ID 100 empfangen wurde. Wenn der definierte Speicherbereich gefüllt ist, wird die Aufzeichnung der Daten beendet. Verwendete Bausteine: - Funktion WRITE_CSV_8BYTE, - Funktion SYNC. Bild P35010: Anzeige Datei-Informationen Bild P35020: Anzeige aktueller Datensatz Bild P35030: Anzeige Liste von 10 Datensätzen
`CR1051Demo_O2M_1Cam_xx.pro`	Anschluss von 1 Kamera O2M100 am Monitor mit der Funktion CAM_O2M. Umschalten zwischen Teil- und Vollbild. Bild 39000: Auswahlmenü Bild 39010: Kamerabild + Textbox Bild 39020: Kamerabild als Vollbild Bild 39030: nur Visualisierung
`CR1051Demo_O2M_2Cam_xx.pro`	Anschluss von 2 Kameras O2M100 am Monitor mit der Funktion CAM_O2M. Umschalten zwischen den Kameras und zwischen Teil- und Vollbild. Bild 39000: Auswahlmenü Bild 39010: Kamerabild + Textbox Bild 39020: Kamerabild als Vollbild Bild 39030: nur Visualisierung
`CR1051Demo_Powerdown_Retain_bin_xx.pro`	Beispiel mit der Funktion PDM_POWER_DOWN aus der Bibliothek `ifm_CR1051_Vxxyyzz.Lib`, um Retain-Variable in die Datei `Retain.bin` zu speichern. Simulation des ShutDown mit [F3].
`CR1051Demo_Powerdown_Retain_bin2_xx.pro`	Beispiel mit der Funktion PDM_POWER_DOWN aus der Bibliothek `ifm_CR1051_Vxxyyzz.Lib`, um Retain-Variable in die Datei `Retain.bin` zu speichern. Simulation des ShutDown mit [F3].
`CR1051Demo_Powerdown_Retain_cust_xx.pro`	Beispiel mit der Funktion PDM_POWER_DOWN und der Funktion PDM_READ_RETAIN aus der Bibliothek `ifm_CR1051_Vxxyyzz.Lib`, um Retain-Variable in die Datei `/home/project/myretain.bin` zu speichern. Simulation des ShutDown mit [F3].
`CR1051Demo_Read_Textline_xx.pro`	Das Beispiel-Programm liest jeweils 7 Textzeilen aus dem PDM-Dateisystem mit Hilfe der Funktion READ_TEXTLINE. Bild P01000: Anzeige gelesener Text
`CR1051Demo_Real_in_xx.pro`	Einfaches Beispiel für die Eingabe eines REAL-Werts in das PDM. Bild P01000: Eingabe und Anzeige des REAL-Werts

Demo-Programm	Funktion
CR1051Demo_Ringlogging_on_flash_intern_xx.pro	Schreibt einen CSV-Datensatz in den internen Flash-Speicher, wenn [F3] gedrückt wird oder eine CAN-Nachricht auf dem ID 100 empfangen wurde. Die Dateinamen sind frei definierbar. Wenn der definierte Speicherbereich gefüllt ist, beginnt die Aufzeichnung der Daten von vorn. Verwendete Bausteine: - Funktion WRITE_CSV_8BYTE, - Funktion SYNC. Bild P35010: Anzeige Datei-Informationen Bild P35020: Anzeige aktueller Datensatz Bild P35030: Anzeige Liste von 8 Datensätzen
CR1051Demo_Ringlogging_on_flash_pcmcia_xx.pro	Schreibt einen CSV-Datensatz auf die PCMCIA-Karte, wenn [F3] gedrückt wird oder eine CAN-Nachricht auf dem ID 100 empfangen wurde. Die Dateinamen sind frei definierbar. Wenn der definierte Speicherbereich gefüllt ist, beginnt die Aufzeichnung der Daten von vorn. Verwendete Bausteine: - Funktion WRITE_CSV_8BYTE, - Funktion OPEN_PCMCIA, - Funktion SYNC. Bild P35010: Anzeige Datei-Informationen Bild P35020: Anzeige aktueller Datensatz Bild P35030: Anzeige Liste von 8 Datensätzen
CR1051Demo_RW-Parameter_xx.pro	In einer Liste können Parameter gewählt und geändert werden. Beispiel mit folgenden Bausteinen: - Funktion READ_PARAMETER_WORD, - Funktion WRITE_PARAMETER_WORD. Bild P35010: Liste von 20 Parametern

_xx = Angabe der Demo-Version

4.2 Funktionskonfiguration der Ein- und Ausgänge

Bei einigen Geräten der Controller-Familie ecomat*mobil* sind bei den Ein- und Ausgängen zusätzliche Diagnosefunktionen aktivierbar. Damit kann das jeweilige Ein- und Ausgangssignal überwacht werden und im Fehlerfall kann das Applikations-Programm darauf reagieren.

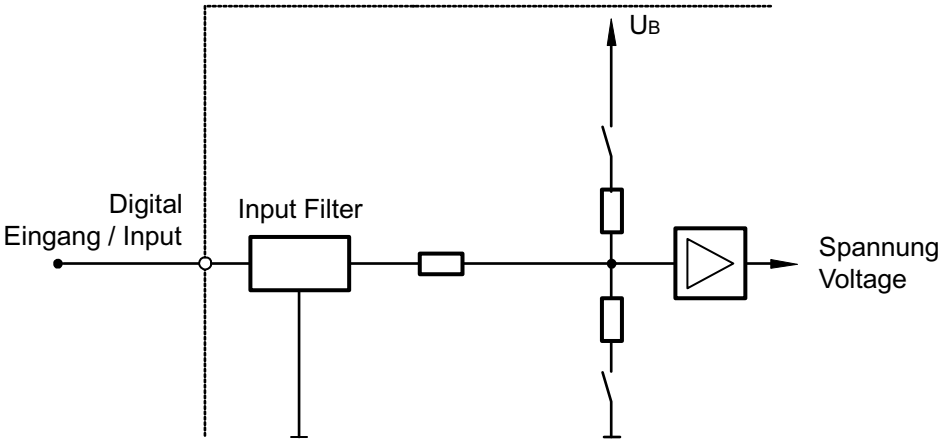
Je nach Ein- und Ausgang müssen bei der Nutzung der Diagnose bestimmte Randbedingungen beachtet werden:

- Anhand des Datenblattes muss überprüft werden, ob das eingesetzte Gerät die beschriebenen Ein- und Ausgangsgruppen zur Verfügung stellt.
- Zur Konfiguration der Ein- und Ausgänge sind in den Gerätebibliotheken (z.B. `ifm_CR0020_Vx.LIB`) Konstanten vordefiniert (z.B. IN_DIGITAL_H).
  Ausführliche Angaben → Anhang, Seite 285.

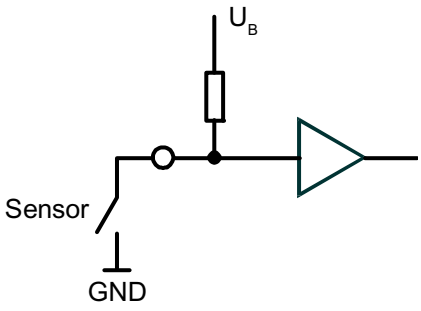
4.2.1 Eingänge konfigurieren

Digitaleingänge

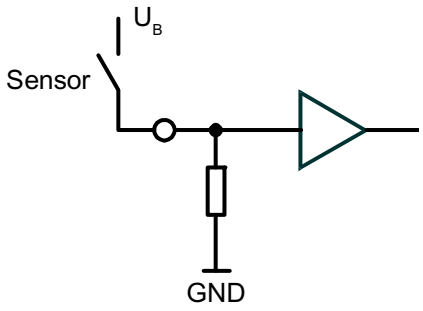
Je nach Controller können auch die Digitaleingänge unterschiedlich konfiguriert werden. Neben den Schutzmechanismen gegen Störungen werden die Digitaleingänge intern über eine Analogstufe ausgewertet. Das ermöglicht die Diagnose der Eingangssignale. In der Applikations-Software steht das Schaltsignal aber direkt als Bit-Information zur Verfügung. Bei einem Teil dieser Eingänge kann auch das Potential gewählt werden, gegen das geschaltet wird.



Grafik: Prinzipschaltung High-/Lowside Eingang für negative und positive Gebersignale



Highside Eingang für negatives Gebersignal



Lowside Eingang für positives Gebersignal

Schnelle Eingänge

Zusätzlich verfügen die Controller über bis zu 16 schnelle Zähl-/Impulseingänge für eine Eingangsfrequenz bis 50 kHz (→ Datenblatt). Werden z.B. mechanische Schalter an diesen Eingängen angeschlossen, kann es durch Kontaktprellen zu Fehlsignalen in der Steuerung kommen. Über die Applikations-Software müssen bei Bedarf diese "Fehlsignale" ausgefiltert werden.

Ferner muss beachtet werden, ob die Impulseingänge für Frequenzmessung (FRQx) und/oder Periodendauermessung (CYLx) ausgelegt sind (→ Datenblatt).

Z.B. folgende Funktionsblöcke können Sie hier sinnvoll einsetzen:

an FRQx-Eingängen:

- Frequenzmessung mit Funktion FREQUENCY (→ Seite 210)
- Schneller Zähler mit Funktion FAST_COUNT (→ Seite 221)

an CYLx-Eingängen:

- Periodendauermessung mit Funktion PERIOD (→ Seite 212) oder mit Funktion PERIOD_RATIO (→ Seite 214)
- Phasenlage von 2 schnellen Eingängen miteinander vergleichen mit Funktion PHASE (→ Seite 216)

Info

Bei Einsatz dieser Funktion werden automatisch die dort parametrierten Ein-/Ausgänge konfiguriert. Der Programmierer der Applikation ist hiervon entlastet.

Analogeingänge

Die Analogeingänge können über das Applikationsprogramm konfiguriert werden. Der Messbereich kann zwischen folgenden Bereichen umgeschaltet werden:

- Stromeingang 0...20 mA
- Spannungseingang 0...10 V
- Spannungseingang 0...30 / 32 V

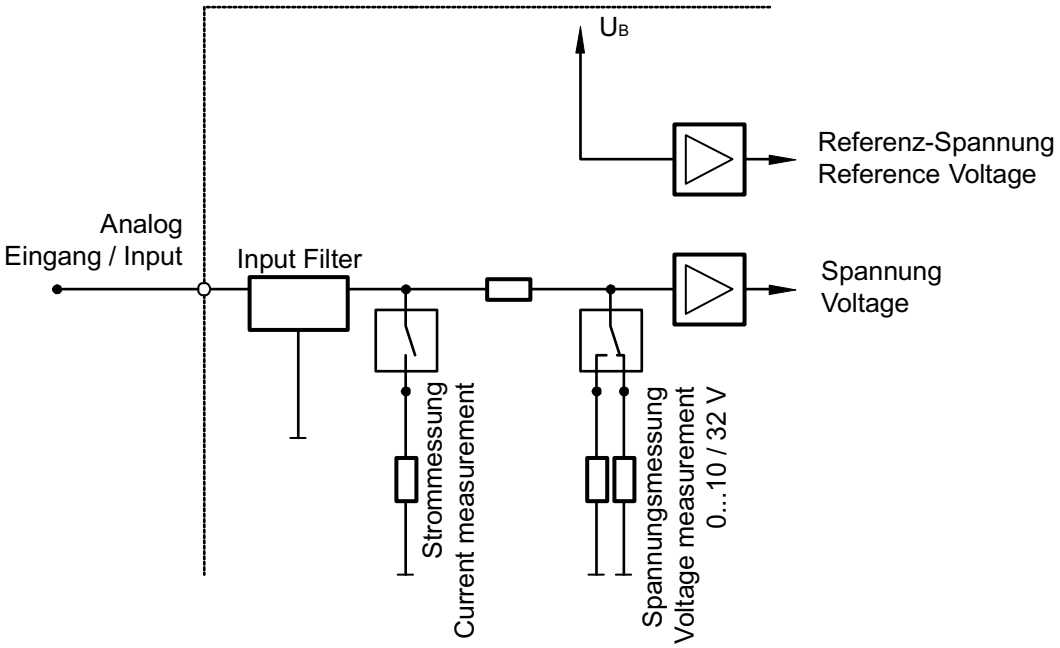
Wird in der Betriebsart "0...30 / 32 V" die Versorgungsspannung zurückgelesen, kann die Messung auch ratiometrisch erfolgen. Das bedeutet, ohne zusätzliche Referenzspannung können Potentiometer oder Joysticks ausgewertet werden. Ein Schwanken der Versorgungsspannung hat auf diesen Messwert dann keinen Einfluss.

Alternativ kann ein Analogkanal auch digital ausgewertet werden.

HINWEIS

Bei ratiometrischer Messung sollten die angeschlossenen Sensoren mit der gleichen Spannungsquelle wie der Controller versorgt werden. Dadurch werden Fehlmessungen durch Spannungsverschiebungen vermieden.

Bei digitaler Auswertung sind die höheren Eingangswiderstände zu berücksichtigen.



Grafik: Prinzipschaltung der Analogeingänge

Analogeingänge ANALOG4...7 (%IW6...%IW9)

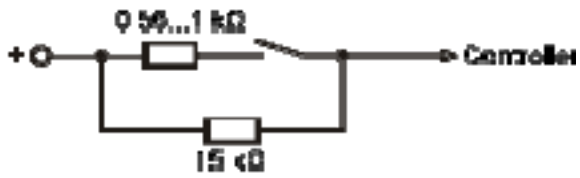
Bei diesen Eingängen handelt es sich um eine Gruppe von Analogkanälen, die auch digital ausgewertet werden können.

Die Konfiguration kann über die Systemvariablen I4_MODE...I7_MODE oder vorzugsweise über die Funktion INPUT_ANALOG (→ Seite 263) (Eingang MODE) erfolgen.

Werden die Analogeingänge auf Strommessung konfiguriert, wird bei Überschreiten des Endwertes (> 23 mA) in den sicheren Spannungsmessbereich (0...32V DC) geschaltet und das jeweilige Fehlerbit im Merkerbyte ERROR_Ix gesetzt. Wird der Grenzwert wieder unterschritten, schaltet der Eingang selbsttätig auf den Strommessbereich zurück.

Digitaleingangsgruppe I0...I3 (%IX0.0...%IX1.8)

Bei diesen Eingängen handelt es sich um Digitaleingänge, die für die Diagnose intern analog ausgewertet werden. Die Konfiguration der Diagnosefunktion erfolgt über die Systemvariablen Ix_MODE. Die Diagnoseinformation wird über das System-Merkerbit ERROR_Ix angezeigt. Das System-Merkerbit DIAGNOSE zeigt den Leiterbruch oder den Kurzschluss des Eingangssignals als Sammelfehler an.



Grafik: Nichtelektronische Schalter

Um die Eingangssignale von nichtelektronischen Schaltern zu überwachen, müssen diese mit einer zusätzlichen Widerstandsbeschaltung versehen werden.

Info

An allen Eingängen können diagnosefähige Sensoren nach NAMUR verwendet werden. Eine zusätzliche Widerstandsbeschaltung ist dann nicht erforderlich.

Ist die Diagnosefunktion aktiv, steht zusätzlich für jeden Eingangskanal die Systemvariable ANALOG_0...ANALOG_3 mit den Spannungswerten am Eingang zur Verfügung.

4.2.2 Ausgänge konfigurieren

Digital- und PWM-Ausgänge

Bei den Controller-Ausgängen können drei Typen unterschieden werden:

- Highside-Digitalausgänge mit und ohne Diagnosefunktion
- Highside-Digitalausgänge mit und ohne Diagnosefunktion und zusätzlichem PWM-Modus
- PWM-Ausgängen, die mit und ohne Stromregelfunktion betrieben werden können. Stromgeregelte PWM-Ausgänge werden überwiegend zur Ansteuerung von proportionalen Hydraulikfunktionen genutzt.

⚠ WARNUNG

Sach- oder Körperschäden möglich durch Fehlfunktionen!

Ausgänge, die im PWM-Modus betrieben werden, unterstützen keine Diagnosefunktionen und es werden keine ERROR-Merker gesetzt. Dies ist bedingt durch den Aufbau der Ausgänge.

Die Funktion OUT_OVERLOAD_PROTECTION ist in diesem Modus nicht aktiv!

HINWEIS

Wird ein Ausgang im Fehlerfall (z.B. Kurzschluss) hardwaremäßig (mittels Sicherung) abgeschaltet, ändert sich der durch das Applikations-Programm erzeugte logische Zustand dadurch nicht.

► Um die Ausgänge nach Beseitigung des Peripheriefehlers erneut zu setzen, müssen die Ausgänge zunächst logisch im Applikations-Programm zurückgesetzt und ggf. erneut gesetzt werden.

Ausgangsgruppe Q0...Q4 (%QX0.0...%QX1.8)

Wenn Q0...Q4 als PWM-Ausgänge eingesetzt werden, wird die Diagnose über die integrierten Strommesskanäle realisiert, die auch für die stromgeregelten Ausgangsfunktionen genutzt werden. Über die Funktion OUTPUT_CURRENT (→ Seite 181) können Lastströme ≥ 100 mA angezeigt werden.

Dieser Funktionsblock kann auch für die Diagnose genutzt werden, wenn die Ausgänge als Digitalkanal genutzt werden (nur für Lastströme ≥ 100 mA).

4.3 Hinweise zur Anschlussbelegung

Die Anschlussbelegungen (→ Montageanleitungen der Controller, Kapitel "Anschlussbelegung") beschreiben die Standard-Gerätekonfigurationen. Die Anschlussbelegung dient der Zuordnung der Ein- und Ausgangskanäle zu den IEC-Adressen und den Geräteanschlussklemmen.

Beispiele:

12 GND_A

12	Klemmennummer
GND_A	Klemmenbezeichnung

30 %IX0.7 BL

30	Klemmennummer
%IX0.7	IEC-Adresse für einen binären Eingang
BL	hardwaremäßige Ausführung des Eingangs, hier: **B**inär **L**ow-Side

47 %QX0.3 BH/PH

47	Klemmennummer
%QX0.3	IEC-Adresse für einen binären Ausgang
BH/PH	Hardwaremäßige Ausführung des Ausgangs, hier: **B**inär-**H**igh-Side oder **P**WM-**H**igh-Side

Die einzelnen Kürzel haben folgende Bedeutung:

A	Analog-Eingang
BH	Binärer Eingang/Ausgang, High-Side
BL	Binärer Eingang/Ausgang, Low-Side
CYL	Eingang Periodendauermessung
ENC	Eingang Drehgebersignale
FRQ	Frequenzeingang
H-Bridge	Ausgang mit H-Brücken-Funktion
PWM	**P**uls**w**eiten-**m**oduliertes Signal
PWM_I	PWM-Ausgang mit Strommessung
IH	Impuls-/Zählereingang, High-Side
IL	Impuls-/Zählereingang, Low-Side
R	Rücklesekanal für einen Ausgang

Zuordnung der Ein-/Ausgangskanäle:

Je nach Gerätekonfiguration steht an einer Geräteklemme ein Eingang und/oder ein Ausgang zur Verfügung (→ Katalog, Montageanleitung oder Datenblatt des jeweiligen Gerätes).

5 Betriebszustände und Betriebssystem

Inhalt:

5.1 Betriebszustände

Nach Anlegen der Versorgungsspannung kann sich der R360-Controller in einem von fünf möglichen Betriebszuständen befinden:

5.1.1 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- oder Stopp-Zustand abgelöst.

> Die LED leuchtet kurzzeitig orange.

5.1.2 Run-Zustand

Dieser Zustand wird in folgenden Fällen erreicht:

- Aus dem Reset-Zustand (Autostart)
- Aus dem Stopp-Zustand durch das Run-Kommando
  - nur bei Betriebsmodus = Test (→ Kapitel TEST-Betrieb, Seite 39)

5.1.3 Stopp-Zustand

Dieser Zustand wird in folgenden Fällen erreicht:

- Aus dem Reset-Zustand, wenn kein Programm geladen ist
- Aus dem Run-Zustand, wenn:
  - Stopp-Kommando kommt über die Schnittstelle
  - UND: Betriebsmodus = Test (→ Kapitel TEST-Betrieb, Seite 39)

5.1.4 Fatal Error

In diesen Zustand fällt der R360-Controller, wenn ein nicht tolerierbarer Fehler festgestellt wurde. Dieser Zustand kann nur durch einen Reset verlassen werden.

> Die LED leuchtet rot.

5.1.5 Kein Betriebssystem

Es wurde kein Betriebssystem geladen, der R360-Controller befindet sich im Bootlader-Zustand. Vor dem Laden der Applikationssoftware muss ein Betriebssystem-Download durchgeführt werden.

> Die LED blinkt grün (schnell).

5.2 Status-LED

Die Betriebszustände werden durch die integrierte Status-LED (Default-Einstellung) angezeigt.

LED-Farbe	Blinkfrequenz	Beschreibung
aus	konstant aus	keine Betriebsspannung
Grün	5 Hz	kein Betriebssystem geladen
Grün	2 Hz	RUN-Zustand
Grün	konstant ein	STOPP-Zustand
Rot	2 Hz	RUN-Zustand mit Fehler
Rot	konstant ein	Fatal Error oder STOPP-Zustand mit Fehler
Gelb/Orange	kurzzeitig ein	Initialisierung oder Reset Checks

Die Betriebszustände STOPP und RUN können vom Programmiersystem geändert werden.

Bei diesem Controller kann die Status-LED auch durch das Applikations-Programm gesetzt werden. Dazu dient folgende Systemvariable:

LED_MODE	Blinkfrequenz aus der Datenstruktur "LED_MODES" zulässig: LED_2HZ, LED_1HZ, LED_05HZ, LED_0HZ (konstant)

HINWEIS

Wird der Blinkmodus durch das Applikations-Programm geändert, gilt die obige Tabelle (Default-Einstellung) nicht mehr.

5.3 Betriebssystem laden

Im Auslieferungszustand ist im Normalfall kein Betriebssystem im Controller geladen (LED 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 Schnittstellen (z.B. RS232, CAN).

Der Betriebssystem-Download muss im Normalfall nur einmalig durchgeführt werden. Das Applikations-Programm kann anschließend (auch mehrfach) in den Controller geladen werden, ohne das Betriebssystem zu beeinflussen. Vorteil:

- Bei einem Betriebssystem-Update muss kein EPROM getauscht werden.

Das Betriebssystem wird zusammen mit dieser Dokumentation auf einem separaten Datenträger zur Verfügung gestellt. Zusätzlich kann auch die aktuelle Version von der Homepage der **ifm electronic gmbh** heruntergeladen werden:
→ www.ifm.com > Land/Sprache wählen > [Service] > [Download] > [Steuerungssysteme]

HINWEIS

Es müssen immer die zum gewählten Target passenden Software-Stände zum Einsatz kommen:

- des Betriebssystems (`CRnnnn_Vxxyyyzz.H86`),
- der Steuerungskonfiguration (`CRnnnn_Vxx.CFG`),
- der Gerätebibliothek (`CRnnnn_Vxxyyyzz.LIB`)
- und der weiteren Dateien
  (→ Kapitel Übersicht der verwendeten Dateien und Bibliotheken, Seite 288).

CRnnnn	Geräte-Artikelnummer
Vxx: 00...99	Target-Versionsnummer
yy: 00...99	Release-Nummer
zz: 00...99	Patch-Nummer

Dabei müssen der Basisdateiname (z.B. "`CR0032`") und die Software-Versionsnummer "xx" (z.B. "`02`") überall den gleichen Wert haben! Andernfalls geht der Controller in den STOPP-Zustand

Die Werte für "yy" (Release-Nummer) und "zz" (Patch-Nummer) müssen **nicht** übereinstimmen.

Außerdem beachten: Folgende Dateien müssen ebenfalls geladen sein:

- die zum Projekt erforderlichen internen Bibliotheken (in IEC1131 erstellt),
- die Konfigurationsdateien (`*.CFG`)
- und die Target-Dateien (`*.TRG`).

Das Betriebssystem wird mit dem eigenständigen Programm "Downloader" in den Controller übertragen. (Der Downloader befindet sich auf der ecomat*mobil*-CD "Software, Tools and Documentation" oder kann bei Bedarf von der ifm-Homepage heruntergeladen werden)

Das Applikations-Programm wird im Normalfall über das Programmiersystem in den Controller geladen. Es kann aber ebenfalls mit dem Downloader geladen werden, wenn es zuvor aus dem Controller ausgelesen wurde (→ Upload).

5.4 Betriebsmodi

Unabhängig von den Betriebszuständen kann der Controller in verschiedenen Betriebsmodi betrieben werden. Die entsprechenden Steuerungs-Bits können über die Applikations-Software oder im Testbetrieb (→ Kapitel TEST-Betrieb, Seite 39) mit der Programmiersoftware CoDeSys (Fenster: Globale Variablen) gesetzt und rückgesetzt werden.

5.4.1 TEST-Betrieb

Dieser Betriebsmodus wird durch Anlegen eines High-Pegels (Versorgungsspannung) am Test-Eingang erreicht (→ Montageanleitung, Kapitel "Anschlussbelegung"). Jetzt kann der Controller im RUN- oder STOPP-Zustand Kommandos über eine der Schnittstellen entgegennehmen und z.B. mit dem Programmiersystem kommunizieren. Außerdem ist nur in diesem Betriebszustand ein Software-Download im Controller möglich.

Über den Merker TEST kann der Zustand vom Applikations-Programm abgefragt werden.

ACHTUNG

Verlust der gespeicherten Software möglich!

Im Test-Betrieb kein Schutz der gespeicherten Betriebssystem- und Applikations-Software.

Nur für folgende Controller beachten:

- SmartController: CR250n
- CabinetController: CR0301, CR0302
- Platinensteuerung: CS0015

ACHTUNG

Zerstörung des EEPROMs möglich!

Der Test-Eingang darf nicht permanent aktiviert werden, weil sonst die zulässigen Schreibzyklen im EEPROM überschritten werden.

5.4.2 SERIAL_MODE

Die serielle Schnittstelle steht für den Datenaustausch in der Applikation zur Verfügung. Ein Debugging der Applikations-Software ist dann nur noch über die CAN-Schnittstelle möglich.
Für CRnn32 gilt: Ein Debugging der Applikations-Software ist dann nur noch über alle 4 CAN-Schnittstellen oder über USB möglich.

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

(→ Kapitel Nutzung der seriellen Schnittstelle, Seite 253)

5.4.3 DEBUG-Modus

Wird der Eingang DEBUG der Funktion SET_DEBUG (→ Seite 235) auf TRUE gesetzt, kann z.B. das Programmiersystem oder der Downloader mit dem Controller kommunizieren und Systemkommandos ausführen (z.B. für Servicefunktionen über das GSM-Modem CANremote).

Ein Softwaredownload ist in dieser Betriebsart nicht möglich, da der Test-Eingang (→ Kapitel TEST-Betrieb, Seite 39) nicht mit Versorgungsspannung verbunden wird.

6 Fehlercodes und Diagnoseinformationen (Übersicht)

Inhalt:

Um eine möglichst hohe Betriebssicherheit zu gewährleisten, wird vom Betriebssystem der Controller in der Startphase (Reset-Phase) und während der Programmausführung durch interne Fehler-Checks überprüft.

Folgende Fehlermerker werden im Fehlerfall gesetzt:

Fehler	Beschreibung
CANx_BUSOFF	CAN-Schnittstelle x: nicht am Bus
CANx_LASTERROR ¹)	CAN-Schnittstelle x: Fehlernummer der letzten CAN-Übertragung: 0= kein Fehler ≠0 → CAN-Spezifikation → LEC
CANx_WARNING	CAN-Schnittstelle x: Warnschwelle erreicht (≥ 96)
ERROR	Bit ERROR setzen ³) / Relais ausschalten *)
ERROR_MEMORY	Speicher-Fehler
ERROR_POWER	Unter-/Überspannungs-Fehler
ERROR_TEMPERATURE ²)	Übertemperatur-Fehler (> 85 °C)
ERROR_VBBR	Versorgungsspannungs-Fehler VBB_R

CANx steht für die Nummer der CAN-Schnittstelle (CAN 1...x, abhängig vom Gerät)

¹) der Zugriff auf diese Merker erfordert genaue Kenntnisse der CAN-Schnittstelle und wird im Normalfall nicht benötigt.

²) Merker nicht für CR250n, CR0301, CR0302 verfügbar

³) Mit Setzen des Systemmerkers ERROR wird der Ausgang ERROR (Klemme 13) auf FALSE gesetzt. Im "fehlerlosen Zustand" ist der ERROR Ausgang TRUE (negative Logik).

*) Relais für CR250n und CR030x NICHT verfügbar.

Folgende Diagnosemeldungen sind nur für Geräte mit Peripherie-Anschlüssen verfügbar:

Dignosemeldung	Art	Beschreibung
ERROR_BREAK_Qx *)	BYTE	Leiterbruch-Fehler an der Ausgangsgruppe x
ERROR_Ix	BYTE	Peripherie-Fehler an der Eingangsgruppe x
ERROR_SHORT_Qx *)	BYTE	Kurzschluss-Fehler an der Ausgangsgruppe x

x steht für die Nummer der Ein- oder Ausgangsgruppe (Wort 0...x, abhängig vom Gerät).

*) Merker nur für ClassicController, ExtendedController, SafetyController verfügbar

HINWEIS

In ungünstigen Fällen kann der Ausgangstransitstor einen gestörten Ausgang bereits abschalten, bevor das Betriebsystem die Störung erkennen konnte. Dann wird der entsprechende Fehlermerker NICHT gesetzt.

Wir empfehlen, dass der Applikations-Programmierer den Fehler (zusätzlich) durch Rücklesen der Ausgänge auswertet.

Vollständige Aufstellung der gerätespezifischen Fehlercodes und Diagnosemeldungen
→ Kapitel Systemmerker, Seite 287

6.1 Reaktion auf System-Fehler

Es liegt grundsätzlich in der Verantwortung des Programmierers, auf die Fehlermerker (Systemmerker) im Applikations-Programm zu reagieren.

Die spezifischen Fehlerbits und -bytes sollten im Applikations-Programm verarbeitet werden. Über den Fehlermerker erhält man eine Fehlerbeschreibung. Diese Fehlerbits/-bytes können bei Bedarf weiter verarbeitet werden.

Grundsätzlich müssen alle Fehlermerker durch das Applikations-Programm zurückgesetzt werden. Ohne ausdrückliches Rücksetzen der Fehlermerker bleiben die Merker gesetzt mit entsprechender Auswirkung im Applikations-Programm.

Bei schweren Fehlern kann zusätzlich das System-Merkerbit ERROR gesetzt werden. Das bewirkt gleichzeitig, dass die Betriebs-LED (sofern vorhanden) rot leuchtet, der ERROR-Ausgang auf FALSE gesetzt wird und dass die Überwachungsrelais (sofern vorhanden) abgeschaltet werden. Damit fallen die darüber gesicherten Ausgänge ab.

6.1.1 Beispielablauf für Reaktion auf System-Fehler

System stellt überhöhte Temperatur im Controller fest.

Betriebssystem setzt das Fehler-Bit ERROR_TEMPERATURE.

Das Applikations-Programm erkennt diesen Zustand durch Abfrage der betreffenden Bits.

> Applikations-Programm schaltet Ausgänge ab.

Bei Bedarf kann per Applikations-Programm zusätzlich das Fehler-Bit ERROR gesetzt werden.

> Folgen:
- Betriebsanzeige-LED blinkt rot
- Sicherheitsrelais fällt ab
- die Versorgungsspannung aller Ausgänge wird abgeschaltet
- Pegel des Ausgangs ERROR*) wird LOW.

► Ursache des Fehlers beheben.

> Betriebssystem setzt Fehler-Bit ERROR_TEMPERATURE zurück.

► Wenn gesetzt, muss das Fehler-Bit ERROR per Applikations-Programm gelöscht werden.

> Das Relais zieht wieder an und die LED blinkt wieder grün.

*) Ausgang bei CR0301, CR0302, CS0015 nicht vorhanden

7 Programmierung und Systemressourcen

Inhalt:

Bei den frei programmierbaren Geräten aus der Controller-Familie ecomatmobil stehen in einem großen Umfang Funktionen zur Verfügung, die den Einsatz der Geräte in den unterschiedlichsten Applikationen ermöglichen.

Da diese Funktionen je nach Komplexität mehr oder weniger Systemressourcen belegen, können nicht immer alle Funktionen gleichzeitig und mehrfach eingesetzt werden.

ACHTUNG

Gefahr von zu trägem Verhalten des Controllers! Zykluszeit darf nicht zu lang werden!

► Beim Erstellen des Applikations-Programms müssen die oben aufgeführten Empfehlungen beachtet und durch Austesten überprüft werden. Bei Bedarf muss durch Neustrukturieren der Software und des Systemaufbaus die Zykluszeit vermindert werden.

Ferner muss beachtet werden, welche CPU in dem eingesetzten Gerät verwendet wird.

Controller-Familie	Artikel-Nr.	CPU-Frequenz [MHz]
ClassicController (16 Bit)	CR0020 CR0505	40
ExtendedController (16 Bit)	CR0220	40
CabinetController	CR0301 CR0302	20
CabinetController	CR0303	40
SmartController	CR250n	20
ClassicController (32 Bit)	CR0032	150
ExtendedController (32 Bit)	CR0232	150

Je höher die CPU-Frequenz, desto größer ist die Leistungsfähigkeit für den gleichzeitigen Einsatz von komplexen Funktionen.

7.1 Überdurchschnittliche Belastungen

Folgende Funktionen z.B. belasten die Systemressourcen überdurchschnittlich:

Funktion	Überdurchschnittliche Belastung
CYCLE, PERIOD, PERIOD_RATIO, PHASE	Einsatz mehrerer Messkanäle mit einer hohen Eingangsfrequenz
OUTPUT_CURRENT_CONTROL, OCC_TASK	Einsatz mehrerer Stromregler gleichzeitig

Funktion	Überdurchschnittliche Belastung
CAN-Schnittstelle	Hohe Baud-Rate (> 250 kBit) mit einer hohen Buslast
PWM, PWM1000	Viele PWM-Kanäle gleichzeitig. Es sind besonders die Kanäle ab 4 deutlich zeitkritischer
INC_ENCODER	Viele Encoder-Kanäle gleichzeitig

Die oben exemplarisch aufgeführten Funktionen lösen System-Interrupts aus. Das bedeutet: Jeder Aufruf verlängert die Zykluszeit des Applikations-Programms.

Als Richtwerte sollten folgende Angaben beachtet werden:

7.2 Grenzen bei SmartController

Stromregler	max. 1	Möglichst keine weiteren belastenden Funktionen einsetzen
CYCLE, PERIOD, PERIOD_RATIO, PHASE	1 Kanal	Eingangsfrequenz ≤ 5 kHz
	4 Kanäle	Eingangsfrequenz ≤ 1 kHz
INC_ENCODER	max. 2	Möglichst keine weiteren belastenden Funktionen einsetzen!

ACHTUNG

Gefahr von zu trägem Verhalten des Controllers! Zykluszeit darf nicht zu lang werden!

► Bei der Erstellung des Applikations-Programms müssen die oben aufgeführten Empfehlungen beachtet und durch Austesten überprüft werden. Bei Bedarf muss durch Neustrukturierung der Software und des Systemaufbaus die Zykluszeit optimiert werden.

7.3 Verhalten des Watchdog

Bei allen ecomat*mobil*-Controllern wird die Programmlaufzeit über einen Watchdog überwacht. Wird die maximale Watchdog-Zeit überschritten, führt der Controller einen Reset durch und startet neu (SafetyController: Controller bleibt im Reset; LED erlischt).

Je nach Hardware haben die einzelnen Controller ein unterschiedliches Zeitverhalten:

Controller	Watchdog [ms]
ClassicController	100
ExtendedController	100
SmartController	100…200
SafetyController	100
CabinetController	100…200
Platinensteuerung	100…200
PDM360 smart	100…200
PDM360 compact	kein Watchdog
PDM360	kein Watchdog

7.4 Verfügbarer Speicher

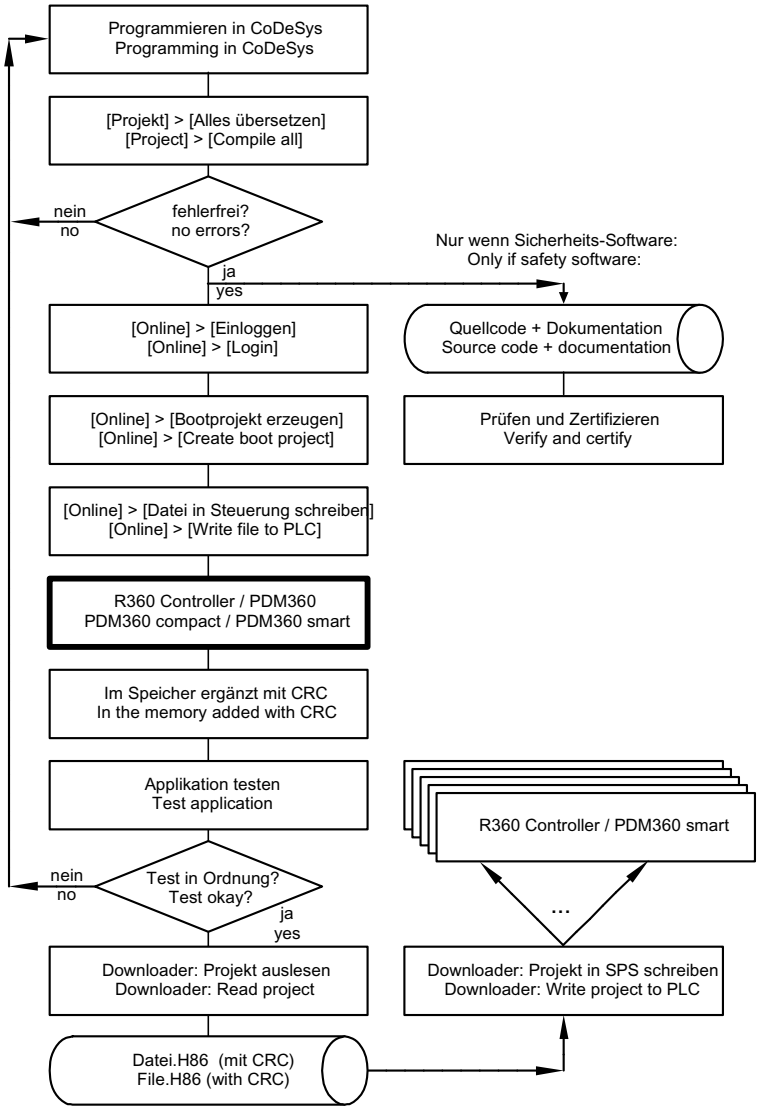
Physikalischer Speicher	Physikalisch vorhandener FLASH-Speicher (nichtflüchtiger, langsamer Speicher)	512 kByte
	Physikalisch vorhandener RAM (flüchtiger, schneller Speicher)	256 kByte
	Physikalisch vorhandener EEPROM (nichtflüchtiger, langsamer Speicher)	4 kByte
	Physikalisch vorhandener FRAM (nichtflüchtiger, schneller Speicher)	---
Nutzung des FLASH-Speichers	Für die IEC-Applikation reservierter und vom PC maximal geladener Code im FLASH-Speicher	192 kByte
	Speicher für Daten außerhalb der IEC-Applikation, die vom Anwender beschrieben werden können, wie z.B. Files, Bitmaps, Fonts	48 kByte
	Speicher für Daten außerhalb der IEC-Applikation, die vom Anwender mit Funktionen wie FLASHREAD, FLASHWRITE bearbeitet werden	16 kByte
RAM	Speicher für die von der IEC-Applikation reservierten und vom PC maximal geladenen Daten im RAM	48 kByte
Remanenter Speicher	Speicher für in der IEC-Applikation als VAR_RETAIN deklarierten Daten	256 Byte
	Speicher für in der IEC-Applikation als RETAIN vereinbarten Merker	512 kByte
	Vom Anwender frei verfügbarer remanenter Speicher. Der Zugriff erfolgt über die Funktionen FRAMREAD, FRAMWRITE, E2READ, E2WRITE.	256 kByte
	Vom Anwender frei verfügbarer FRAM. Der Zugriff erfolgt über Adressoperator.	4 kByte

7.5 Programm-Erstellung und Download in die Steuerung

Das Applikations-Programm wird mit dem Programmiersystem CoDeSys erstellt und während der Programmentwicklung mehrfach zum Testen in die Steuerung geladen:
In CoDeSys: [Online] > [Datei in Steuerung schreiben].

Für jeden derartigen Download via CoDeSys wird dazu der Quellcode neu übersetzt. Daraus resultiert, dass auch jedes Mal im Speicher der Steuerung eine neue Prüfsumme gebildet wird. Auch für Sicherheitssteuerungen ist dieses Verfahren bis zur Freigabe der Software zulässig.

Zumindest für sicherheitsrelevante Applikationen muss aber für die Serienproduktion der Maschine eine Einheitlichkeit der Software und der zugehörigen Prüfsumme gewährleistet sein.



Grafik: Erstellen und Verteilen der (zertifizierten) Software

ifm-Downloader

Der ifm-Downloader dient dem einfachen Übertragen des Programmcodes vom Programmierplatz in die Steuerung. Grundsätzlich kann jede Applikations-Software mit dem ifm-Downloader auf die Steuerungen kopiert werden. Vorteil: Dazu ist kein Programmiersystem mit einer CoDeSys-Lizenz erforderlich.

Sicherheitsrelevante Applikations-Software MUSS mit dem ifm-Downloader auf die Steuerungen kopiert werden, um die Prüfsumme CRC, mit der die Software zertifiziert wurde, nicht zu verfälschen.

> **HINWEIS**
>
> Der ifm-Downloader kann nicht eingesetzt werden für folgende Geräte:
> - PDM360: CR1050, CR1051, CR1060,
> - PDM360 compact: CR1052, CR1053, CR1055, CR1056.

Zertifizieren und Verteilen der sicherheitsrelevanten Software

Nur sicherheitsrelevante Applikations-Software muss zertifiziert sein, bevor sie auf Serienmaschinen kopiert wird und zum Einsatz kommt:

- Sichern der freigegebenen Software
  Nach Abschluss der Programmentwicklung und Freigabe des Gesamtsystems durch die entsprechende Zertifizierungsstelle (z.B. TÜV, BiA), muss die letzte Version des in die Steuerung geladenen Applikations-Programms mit dem ifm-Downloader zunächst aus der Steuerung ausgelesen und auf einem Datenträger unter dem Namen `name_der_projektdatei.H86` gespeichert werden. Nur dieses Verfahren gewährleistet, dass die Applikations-Software mit den entsprechenden Prüfsummen gesichert ist.
- Download der freigegebenen Software
  Um in der Serienproduktion alle Maschinen mit einer einheitlichen Software auszurüsten, darf nur diese Datei mit dem ifm-Downloader in die Steuerungen geladen werden.
- Ein Fehler in den Daten dieser Datei wird durch die integrierte Prüfsumme beim erneuten Laden durch den ifm-Downloader automatisch erkannt.

Ändern der sicherheitsrelevanten Software nach der Zertifizierung

Veränderungen in der Applikations-Software mit dem Programmiersystem CoDeSys erzeugen automatisch wieder eine neue Applikations-Datei. Diese darf nur nach erneuter Zertifizierung in die sicherheitsrelevanten Steuerungen kopiert werden. Dabei das oben beschriebene Verfahren erneut anwenden!

Auf die Neu-Zertifizierung kann gegebenenfalls unter folgenden Bedingungen verzichtet werden:

- für die Änderung erfolgte eine neue Risikobewertung,
- es wurden KEINE sicherheitsgerichteten Elemente verändert, hinzugefügt oder entfernt,
- die Änderung wurde sauber dokumentiert.

8 CAN im ecomatmobil-Controller

Inhalt:

8.1 Allgemeines zu CAN

Der CAN-Bus (**C**ontroller **A**rea **N**etwork) gehört zu den Feldbussen.

Es handelt sich dabei um ein asynchrones, serielles Bussystem, das 1983 von Bosch für die Vernetzung von Steuergeräten in Automobilen entwickelt und 1985 zusammen mit Intel vorgestellt wurde, um die Kabelbäume (bis zu 2 km pro Fahrzeug) zu reduzieren und dadurch Gewicht zu sparen.

8.1.1 Topologie

Das CAN-Netzwerk wird als Linienstruktur aufgebaut. Stichleitungen sind in eingeschränktem Umfang zulässig. Des Weiteren sind auch ein ringförmiger Bus (Infotainment Bereich) sowie ein sternförmiger Bus (Zentralverrieglung) möglich. Beide Varianten haben im Vergleich zum linienförmigen Bus jeweils einen Nachteil:

Im ringförmigen Bus sind alle Steuergeräte in Reihe geschaltet, so dass bei einem Ausfall eines Steuergeräts der gesamte Bus ausfällt.

Der sternförmige Bus wird meist von einem Zentralrechner gesteuert, da diesen alle Informationen passieren müssen, mit der Folge, dass bei einem Ausfall des Zentralrechners keine Informationen weitergeleitet werden können. Bei einem Ausfall eines einzelnen Steuergeräts funktioniert der Bus weiter.

Der lineare Bus hat den Vorteil, dass alle Steuergeräte parallel zu einer zentralen Leitung gehen. Nur wenn diese ausfällt, funktioniert der Bus nicht mehr.

Stichleitungen und sternförmiger Bus haben den Nachteil, dass der Wellenwiderstand schwer zu bestimmen ist. Im schlimmsten Fall funktioniert der Bus nicht mehr.

Bei einem Highspeed-Bus (> 125 kBit/s) müssen zusätzlich 2 Abschlusswiderstände von je 120 Ohm (zwischen CAN_HIGH und CAN_LOW) an dem jeweiligen Ende verwendet werden.

8.1.2 CAN-Schnittstellen

Die Controller werden je nach Aufbau der Hardware mit mehreren CAN-Schnittstellen ausgerüstet. Grundsätzlich können alle Schnittstellen unabhängig voneinander mit folgenden Funktionen genutzt werden:

- Layer 2: CAN auf Ebene 2
- CANopen (→ Seite 85), ein Protokoll nach CiA 301/401 für Master/Slave-Betrieb (via CoDeSys®)
- CAN-Netzwerkvariablen (→ Seite 110) (via CoDeSys®)
- Protokoll SAE J1939 (→ Seite 148) (für Antriebs-Management)
- Buslast-Erkennung
- Errorframe-Zähler
- Download-Schnittstelle
- 100 % Buslast ohne Paketverlust

Welche CAN-Schnittstelle des Geräts welche konkreten Möglichkeiten bietet, → Datenblatt des Geräts.

Informativ: Weitere interessante CAN-Protokolle sind:

- "Truck & Trailer Interface" nach ISO 11992
- ISOBUS nach ISO 11783 für Landmaschinen
- NMEA 2000 für den maritimen Einsatz
- CANopen Truck Gateway nach CiA 413 (Umsetzung zwischen ISO 11992 und SAE J1939)

8.1.3 System-Konfiguration

Die Controller werden mit dem Download-Identifier 127 ausgeliefert. Das Download-System benutzt diesen Identifier (= ID) für die erste Kommunikation mit einem nicht konfigurierten Modul über CAN. Der Download-ID kann über den PLC-Browser des Programmiersystems, den Downloader oder das Applikations-Programm eingestellt werden.

Da der Download-Mechanismus auf Basis des CANopen-SDO-Dienstes arbeitet (auch wenn der Controller nicht im CANopen-Modus betrieben wird), müssen alle Steuerungen im Netzwerk einen eindeutigen Identifier besitzen. Die eigentlichen COB-IDs werden nach dem "predefined connection set" aus den Modulnummern abgeleitet. Es darf jeweils nur ein nicht konfiguriertes Modul mit dem Netz verbunden werden. Nachdem die neue Teilnehmernummer 1...126 zugewiesen wurde, kann ein Download oder ein Debugging stattfinden und danach ein weiteres Gerät ins System eingebunden werden.

Der Download-ID wird unabhängig von dem CANopen-Identifier eingestellt. Es muss beachtet werden, dass sich diese IDs nicht mit den Download-IDs und den CANopen-Knotennummern der anderen Controller oder Netzwerkteilnehmer überschneiden.

Controller Programm-Download		**CANopen**	
ID	**COB-ID SDO**	**Node-ID**	**COB-ID SDO**
1…127	TX: 0x580 + Download-ID	1…127	TX: 0x580 + Node-ID
	RX: 0x600 + Download-ID		RX: 0x600 + Node-ID

HINWEIS

Der CAN-Download-ID des Geräts muss mit dem in CoDeSys eingestellten CAN-Download-ID übereinstimmen!

Im CAN-Netzwerk müssen die CAN-Download-IDs einmalig sein!

8.2 CAN-Datenaustausch

Der CAN-Datenaustausch erfolgt über das in der ISO 11898 international genormte CAN-Protokoll der Verbindungsschicht (Ebene 2) des siebenschichtigen ISO/OSI-Referenzmodells.

Jeder Bus-Teilnehmer kann Nachrichten senden (Multimaster-Fähigkeit). Der Datenaustausch arbeitet ähnlich dem Rundfunk. Daten werden ohne Absender und Adresse auf den Bus gesendet. 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.

Für den normalen CAN-Datenaustausch muss der Programmierer lediglich bei der Softwareerstellung die Datenobjekte mit ihren Identifiern dem System bekannt machen. Dies erfolgt über folgende Funktionen:

- Funktion CANx_RECEIVE (→ Seite 75) (CAN-Daten empfangen) und
- Funktion CANx_TRANSMIT (→ Seite 73) (CAN-Daten senden).

Über diese Funktionen werden folgende Einheiten zu einem Datenobjekt verknüpft:

- die RAM-Adresse der Arbeitsdaten,
- der Datentyp,
- der gewählte Identifier (ID).

Diese Datenobjekte nehmen 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-Nachricht besteht aus einem CAN-Identifier (→ Seite 51) und maximal 8 Datenbytes. Der ID repräsentiert nicht das Absender- oder Empfängermodul, sondern kennzeichnet die Nachricht. 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.

8.2.1 CAN-ID

Je nach CAN-ID sind folgende CAN-Identifier frei verfügbar für den Datentransfer:

CAN-ID base	CAN-ID extended
11 Bit	29 Bit
2 047 CAN-Identifier	536 870 912 CAN-Identifier
Standard-Applikationen	Motor-Management (SAE J1939), Truck & Trailer Interface (ISO 11992)

HINWEIS

Der 29-Bit-CAN-ID steht bei einigen Geräten nicht für alle CAN-Schnittstellen zur Verfügung,
→ Datenblatt.

Beispiel 11-Bit CAN-ID (base):

SOF	CAN-ID base Bit 28 ... Bit 18										RTR	IDE
0	0	0	0	0	1	1	1	1	1	1	0	0
	0			7				F				

Beispiel 29-Bit CAN-ID (extended):

SOF	CAN-ID base Bit 28 ... Bit 18										SRR	IDE	CAN-ID extended Bit 17 ... Bit 0																RTR
0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0		1			F			C				0			0			0			0							

Legende:
SOF = **S**tart **o**f **f**rame
Flanke von rezessiv zu dominant

RTR = **R**emote **t**ransmission **r**equest
dominant: Diese Nachricht liefert Daten
rezessiv: Diese Nachricht fordert Daten an

IDE = **Id**entifier **e**xtension flag
dominant: Hiernach folgen Steuerungs-Bits
rezessiv: Hiernach folgt der zweite Teil des 29-Bit-Identifier

SRR = **S**ubstitute **r**emote **r**equest
rezessiv: Extended CAN-ID: Ersetzt das RTR-Bit an dieser Stelle

8.2.2 Daten empfangen

Grundsätzlich werden die empfangenen Datenobjekte automatisch (also ohne Einfluss durch den Anwender) in einem Zwischenspeicher abgelegt.

Pro Identifier steht ein solcher Zwischenspeicher (Warteschlange) zur Verfügung. Dieser Zwischenspeicher wird in Abhängigkeit von der Anwendersoftware nach dem FIFO-Prinzip (**F**irst **I**n, **F**irst **O**ut) über die Funktion CANx_RECEIVE (→ Seite 75) entleert.

8.2.3 Daten senden

Durch den Aufruf der Funktion CANx_TRANSMIT (→ Seite 73) übergibt das Applikations-Programm genau eine CAN-Nachricht an den CAN-Controller. Als Rückgabe erhält man die Information, ob die Nachricht erfolgreich an den CAN-Controller übergeben wurde. Dieser führt dann selbständig die eigentliche Übergabe der Daten auf den CAN-Bus aus.

Der Sendeauftrag wird abgewiesen, wenn der Controller nicht bereit ist, weil er bereits ein Datenobjekt überträgt. Der Sendeauftrag muss dann durch das Applikations-Programm wiederholt werden. Der Anwender bekommt diese Information durch ein Bit angezeigt.

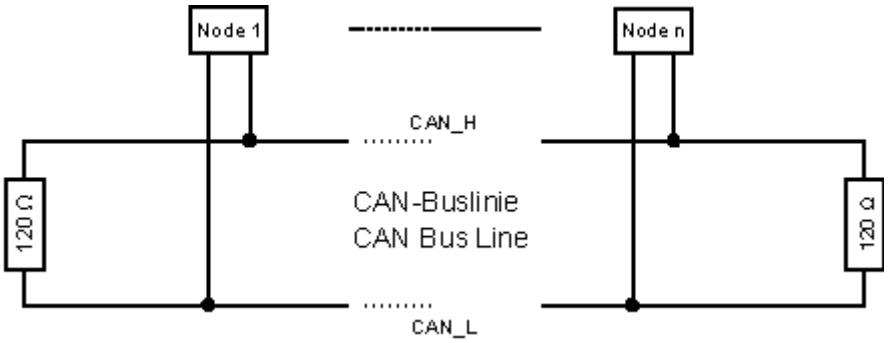
Bei mehreren zeitgleich zum Senden bereiten CAN-Nachrichten wird die Nachricht mit dem niedrigsten ID vorrangig gesendet. Der Programmierer muss daher den CAN-ID (→ Seite 51) sehr umsichtig vergeben.

8.3 Physikalische Anbindung des CAN

Die in den Kapiteln CAN-Datenaustausch (→ Seite 51) und CAN-Fehler (→ Seite 57) beschriebenen Mechanismen der Datenübertragung und der Fehlerbehandlung sind direkt im CAN-Controller implementiert. Die physikalische Verbindung der einzelnen CAN-Teilnehmer wird von der ISO 11898 in der Schicht 1 beschrieben.

8.3.1 Netzaufbau

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



Grafik: Netzaufbau

HINWEIS

Die Linie muss an ihren beiden Enden jeweils mit einem Abschlusswiderstand von der Größe 120 Ω abgeschlossen werden, um ein Verfälschen der Signalqualität zu verhindern.

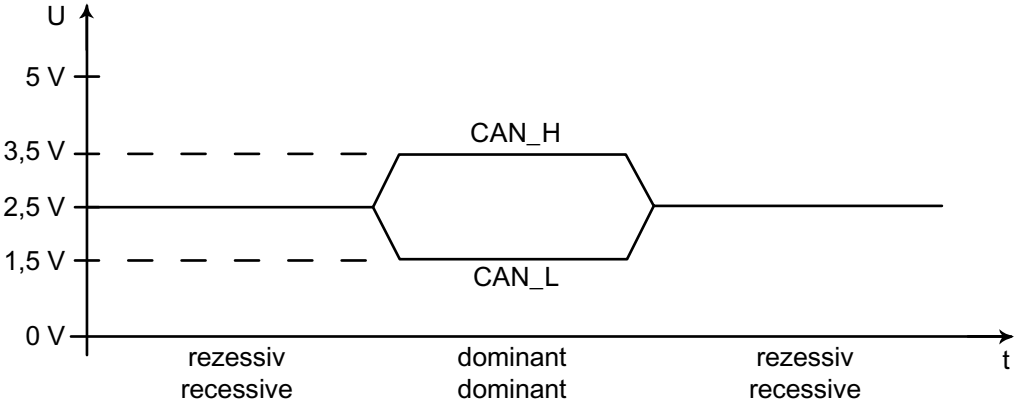
Die Geräte der **ifm electronic gmbh**, die mit einem CAN-Interface ausgestattet sind, haben grundsätzlich keine Abschlusswiderstände.

Stichleitungen

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 führen, sollten die Stichleitungen zu einem Busteilnehmer (z.B. einem E/A-Modul) eine gewisse Länge nicht überschreiten. Stichleitungen mit einer Länge von 2 m (bezogen auf 125 kBit/s) 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.

8.3.2 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. Der Bus wird dadurch aktiv (dominant). Es 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.



Grafik: Buspegel

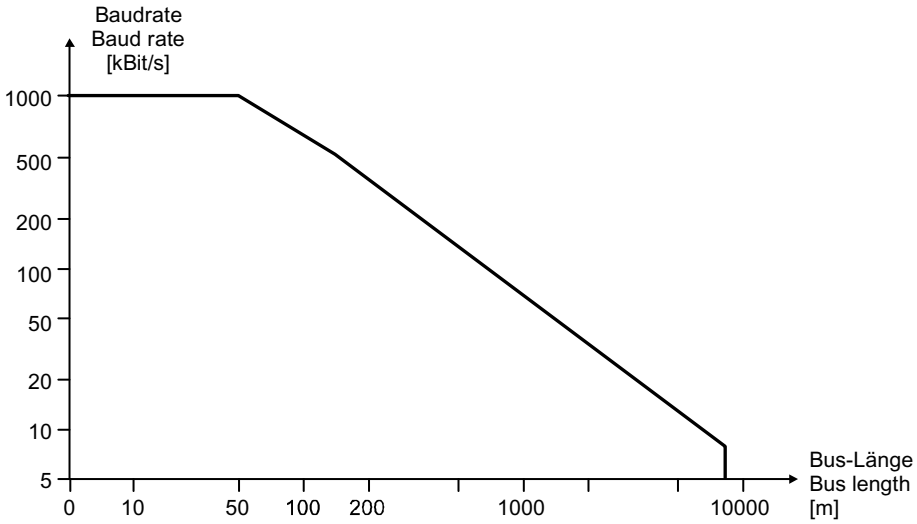
Durch diese differentielle Übertragung mit gemeinsamem Rückleiter wird die Übertragungssicherheit entscheidend verbessert. Störspannungen, die von außen auf das System einwirken, oder Massepotential-Verschiebungen beeinflussen beide Signalleitungen mit gleichen Störgrößen. Dadurch fallen die Störungen bei der Differenzbildung im Empfänger wieder heraus.

8.3.3 Busleitungslänge

Die Länge der Busleitung ist abhängig von:

- Beschaffenheit der Busverbindung (Kabel, Steckverbinder),
- Leitungswiderstand,
- benötigte Übertragungsrate (Baud-Rate),
- Länge der Stichleitungen.

Vereinfachend kann man von folgender Abhängigkeit zwischen Buslänge und Baud-Rate ausgehen:



Grafik: Busleitungslänge

Baud-Rate [kBit/s]	Buslänge [m]	nominelle Bit-Länge [µs]
1 000	30	1
800	50	1,25
500	100	2
250	250	4
125	500	8
62,5	1 000	20
20	2 500	50
10	5 000	100

Tabelle: Abhängigkeiten Buslänge / Baudrate / Bitzeit

HINWEIS

Diese Angaben gelten für CAN Layer 2.

Andere CAN-Protokolle (z.B. SAE J1939 oder ISO 11992) haben andere Vorgaben!

8.3.4 Leitungsquerschnitte

Für die Auslegung des CAN-Netzes ist auch der Leitungsquerschnitt der eingesetzten Busleitung zu beachten. Die folgende Tabelle beschreibt die Abhängigkeit des Leiterquerschnitts bezogen auf die Leitungslänge und der Anzahl der daran angeschlossenen Teilnehmer (Knoten).

Leitungslänge [m]	**Leiterquerschnitt bei 32 Knoten** [mm^2]	**Leiterquerschnitt bei 64 Knoten** [mm^2]	**Leiterquerschnitt bei 100 Knoten** [mm^2]
≤ 100	0,25	0,25	0,25
≤ 250	0,34	0,50	0,50
≤ 500	0,75	0,75	1,00

Abhängig von den EMV-Anforderungen können Sie die Busleitungen wie folgt ausführen:

- parallel,
- als Twisted-Pair
- und/oder abgeschirmt.

8.4 Software für CAN und CANopen

Grundsätzlich können Controller durch Nutzung der Funktionen CANx_TRANSMIT und CANx_RECEIVE direkt an der CAN-Kommunikation teilnehmen (Schicht 2). In der Betriebsart CANopen werden dem Programmierer die festgelegten Dienste aus dem Programmiersystem CoDeSys® zur Verfügung gestellt.

Folgende Punkte sind zu beachten:

- In der Betriebsart "CAN-Direkt" auf Schicht 2 ist der Programmierer für alle Dienste selbst verantwortlich. In diesem Zustand befindet sich der Controller nach folgenden Ereignissen:
  - nach einem Programm-Download oder
  - nach einem Reset-Kommando durch das Programmiersystem.
- Durch Einbinden der CoDeSys®-CANopen-Systembibliotheken (Funktionen in den Zielsystemeinstellungen aktivieren) wird die Betriebsart CANopen aktiviert. Je nach gewählter Funktion läuft der Controller als CANopen-Master oder -Slave (→ Kapitel ifm CANopen-Bibliothek, Seite 85).

8.5 CAN-Fehler und Fehlerbehandlung

Die hier beschriebenen Fehlermechanismen werden von dem im Controller integrierten CAN-Controller automatisch abgearbeitet. Der Anwender hat darauf keinen Einfluss. Der Anwender sollte (je nach Applikation) 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:
  - Fehlerpassiv (Error-passiv)
  - Trennen vom Bus (Bus-off)
  Dies ermöglicht einem zeitweilig gestörten 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.

8.5.1 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 übertragen wird, kann im Gegensatz zu anderen Feldbussystemen (diese warten eine festgelegte Quittierungszeit 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 solche Gefahr fördern. Um diesen Fall auszuschließen, ist ein Mechanismus erforderlich, welcher den Defekt eines Teilnehmers erkennt und diesen Teilnehmer gegebenenfalls vom Bus abschaltet.

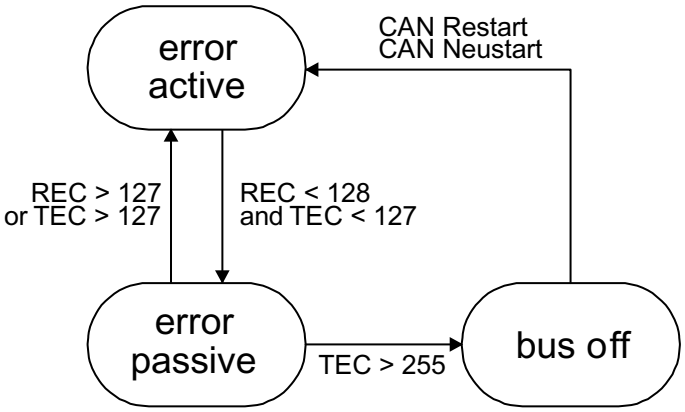
8.5.2 Fehlerzähler

Dazu sind im CAN-Controller ein Sende- und ein Empfangsfehlerzähler enthalten. Diese werden bei jedem fehlerhaften Sende- oder Empfangsvorgang 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 langsam wieder ab. Die Zählerstände sind somit ein Maß für die relative Häufigkeit von gestörten Nachrichten.

Werden Fehler von einem Teilnehmer selbst als erster erkannt (= selbstverschuldete Fehler), wird bei diesem Teilnehmer der Fehler stärker "bestraft" als bei den anderen Busteilnehmern. Dazu wird der Zähler um einen höheren Betrag inkrementiert.

Übersteigt nun der Zählerstand eines Teilnehmers 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" geschaltet (error passiv).



error active
→ Teilnehmer fehleraktiv, Seite 58

error passive
→ Teilnehmer fehlerpassiv, Seite 58

bus off
→ Teilnehmer bus-off, Seite 58

CAN Restart
→ Teilnehmer bus-off, Seite 58

Grafik: Mechanismus des Fehlerzählers

8.5.3 Teilnehmer fehleraktiv

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

8.5.4 Teilnehmer fehlerpassiv

Ein fehlerpassiver Teilnehmer ist ebenfalls 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.

Um den Anwender über das Ansteigen des Fehlerzählers zu informieren, wird bei einem Wert des Fehlerzählers ≥ 96 die Systemvariable CANx_WARNING gesetzt. Der Teilnehmer ist in diesem Zustand noch fehleraktiv.

8.5.5 Teilnehmer bus-off

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

Um diesen Zustand anzuzeigen, wird im Applikations-Programm der Merker CANx_BUSOFF gesetzt.

HINWEIS

Der Fehler CANx_BUSOFF wird vom Betriebssystem automatisch behandelt und zurückgesetzt. Soll eine genauere Fehlerbehandlung und Auswertung über das Applikations-Programm erfolgen, muss die Funktion CANx_ERRORHANDLER (→ Seite 82) eingesetzt werden. Der Fehler CANx_BUSOFF muss dann explizit durch das Applikations-Programm zurückgesetzt werden.

8.6 Beschreibung der CAN-Funktionsblöcke

Inhalt:

Hier werden die CAN-Funktionsblöcke zur Nutzung im Applikationsprogramm beschrieben.

HINWEIS

Um die volle Leistungsfähigkeit von CAN zu nutzen, ist es unbedingt erforderlich, dass sich der Programmierer vor Beginn seiner Arbeit ein genaues **Buskonzept** aufbaut:

- Wie viele Datenobjekte mit welchen Identifiern werden benötigt?
- Wie soll der Controller auf mögliche CAN-Fehler reagieren?
- Wie oft müssen Daten übertragen werden? Dem entsprechend oft müssen die Funktion CANx_TRANSMIT (→ Seite 73) und die Funktion CANx_RECEIVE (→ Seite 75) aufgerufen werden.

► Dabei überwachen, ob die Sendeaufträge erfolgreich an CANx_TRANSMIT übergeben wurden (FB-Ausgang RESULT) oder dafür sorgen, dass die empfangenen Daten mit CANx_RECEIVE aus dem Datenpuffer der Warteschlage ausgelesen und sofort im übrigen Programm entsprechend verarbeitet werden.

Damit eine Kommunikationsverbindung aufgebaut werden kann, muss zuvor bei allen Teilnehmern des CAN-Netzwerkes die gleiche Übertragungsrate (Baud-Rate) eingestellt werden. Beim Controller wird diese mit der Funktion CAN1_BAUDRATE (→ Seite 60) (für die 1. CAN-Schnittstelle) oder über die Funktion CAN2 (→ Seite 71) (für die 2. CAN-Schnittstelle) vorgenommen.

Unabhängig davon, ob die Geräte eine oder mehrere CAN-Schnittstellen unterstützen, werden die der Schnittstelle zugehörigen Funktionen durch Nummerierung in der CAN-Funktion gekennzeichnet (z.B. CAN1_TRANSMIT oder CAN2_RECEIVE). In der Dokumentation wird aus Vereinfachungsgründen die Bezeichnung (z.B. CANx_TRANSMIT) für alle Varianten verwendet.

HINWEIS

Beim Installieren der ecomat*mobil*-CD "Software, Tools and Documentation" wurden auch Projekte mit Vorlagen auf Ihrem Computer im Programmverzeichnis abgelegt:
`…\ifm electronic\CoDeSys V…\Projects\Template_CDV…`

► Die gewünschte dort gespeicherte Vorlage in CoDeSys® öffnen mit:
[Datei] > [Neu aus Vorlage…]

> CoDeSys® legt ein neues Projekt an, dem der prinzipielle Programmaufbau entnommen werden kann. Es wird dringend empfohlen, dem gezeigten Schema zu folgen.
→ Kapitel Programmiersystem über Templates einrichten, Seite 16

In diesem Beispiel 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 (oder umgekehrt) existieren.

8.6.1 Funktion CAN1_BAUDRATE

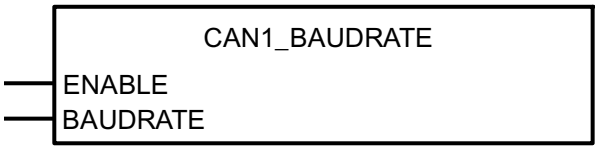
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015

Funktionssymbol:



Beschreibung

CAN1_BAUDRATE stellt die Übertragungsrate für den Busteilnehmer ein.

Mit der Funktion wird für den Controller die Übertragungsrate eingestellt. Dazu wird am Funktionseingang BAUDRATE der entsprechende Wert in kBit/s angegeben. Nach Ausführen der Funktion wird der neue Wert im Gerät gespeichert und steht auch nach einem Spannungsausfall wieder zur Verfügung.

ACHTUNG

Für CR250n, CR0301, CR0302, CS0015 beachten:

Das EEPROM-Speichermodul kann bei Dauerbetrieb dieser Funktion zerstört werden!

► Die Funktion nur **einmalig** bei der Initialisierung im ersten Programmzyklus ausführen! Anschließend die Funktion wieder sperren (ENABLE = "FALSE")!

HINWEIS

Die neue Baud-Rate wird erst nach einem RESET gültig (Spannung Aus/Ein oder Soft-Reset).

ExtendedController: Im Slave-Modul wird die neue Baud-Rate erst nach Spannung Aus/Ein übernommen.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE (nur 1 Zyklus lang): Funktion wird abgearbeitet FALSE: Funktion wird nicht ausgeführt
BAUDRATE	WORD	Baud-Rate [kBit/s] Zulässige Werte: 50, 100, 125, 250, 500, 1000 Voreinstellung = 125 kBit/s

8.6.2 Funktion CAN1_DOWNLOADID

Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015

Funktionssymbol:

Beschreibung

CAN1_DOWNLOADID stellt den Download-Identifier für die erste CAN-Schnittstelle ein.

Mit der Funktion kann der Kommunikations-Identifier für den Programmdownload und das Debuggen eingestellt werden. Der neue Wert wird eingetragen, wenn der Funktionseingang ENABLE auf TRUE gesetzt wird. Der neue Download-ID wird gültig nach Spannung Aus/Ein oder nach einem Softreset.

ACHTUNG

Für CR250n, CR0301, CR0302, CS0015 beachten:

Das EEPROM-Speichermodul kann bei Dauerbetrieb dieser Funktion zerstört werden!

► Die Funktion nur **einmalig** bei der Initialisierung im ersten Programmzyklus ausführen! Anschließend die Funktion wieder sperren (ENABLE = "FALSE")!

HINWEIS

Achten Sie darauf, dass bei jeder Steuerung im selben Netzwerk ein anderer Download-ID eingestellt ist!

Wird der Controller im CANopen-Netzwerk betrieben, darf sich der Download-ID auch mit keinem Modul-ID (Knotennummer) der anderen Teilnehmer überschneiden!

ExtendedController: Im Slave-Modul wird der Download-ID erst nach Spannung Aus/Ein gültig.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE (nur 1 Zyklus lang): Der ID wird gesetzt FALSE: Funktion wird nicht ausgeführt
ID	BYTE	Download-Identifier Zulässige Werte: 1…127

8.6.3 Funktion CAN1_EXT

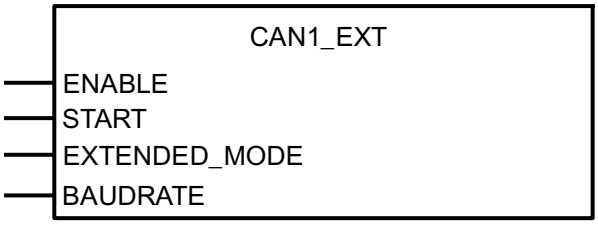
Enthalten in Bibliothek:

`ifm_CAN1_EXT_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

Die Funktion CAN1_EXT initialisiert die 1. CAN-Schnittstelle für den erweiterten Identifier (29 Bits).

Die Funktion muss aufgerufen werden, wenn die 1. CAN-Schnittstelle z.B. mit den Funktionsbibliotheken für SAE J1939 (→ Seite 148) benutzt werden soll.

Eine Änderung der Baud-Rate wird erst gültig nach Spannung Aus/Ein. Die Baud-Raten von CAN 1 und CAN 2 können unterschiedlich eingestellt werden.

Der Eingang START wird nur für einen Zyklus bei Neustart oder Restart der Schnittstelle gesetzt.

HINWEIS

Die Funktion muss **vor** den Funktionen CAN1_EXT_... ausgeführt werden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht ausgeführt
START	BOOL	TRUE (im 1. Zyklus): Schnittstelle wird initialisiert FALSE: Initialisierungszyklus ist beendet
EXTENDED_MODE	BOOL	TRUE: Identifier der 1. CAN-Schnittstelle arbeitet mit 29 Bits FALSE: Identifier der 1. CAN-Schnittstelle arbeitet mit 11 Bits
BAUDRATE	WORD	Baud-Rate [kBit/s] Zulässige Werte: 50, 100,125, 250, 500, 1000 Voreinstellung = 125 kBit/s

8.6.4 Funktion CAN1_EXT_TRANSMIT

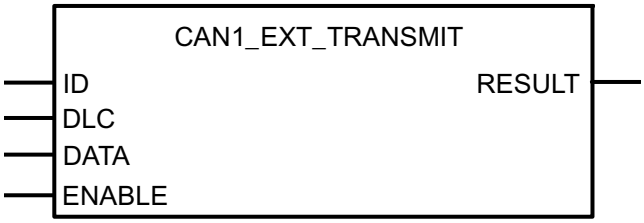
Enthalten in Bibliothek:

`ifm_CAN1_EXT_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

CAN1_EXT_TRANSMIT übergibt ein CAN-Datenobjekt (Message) an den CAN-Controller zur Übertragung.

Die Funktion wird für jedes Datenobjekt im Programmzyklus aufgerufen, bei langen Programmzyklen auch mehrfach. Der Programmierer muss durch Auswertung des FB-Ausgangs 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 Eingang ENABLE kann die Ausführung der Funktion zeitweilig gesperrt werden (ENABLE = FALSE). Damit kann z.B. eine Busüberlastung verhindert werden.

Mehrere Datenobjekte können quasi gleichzeitig verschickt werden, wenn jedem Datenobjekt ein Merkerflag zugeordnet wird und mit diesem die Ausführung der Funktion über den ENABLE-Eingang gesteuert wird.

HINWEIS

Soll diese Funktion verwendet werden, muss zuvor mit der Funktion CAN1_EXT (→ Seite 65) die 1. CAN-Schnittstelle für den erweiterten ID initialisiert werden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ID	DWORD	Nummer des Datenobjekt-Identifier Zulässige Werte: 11-Bit-ID: 0...2 047, 29-Bit-ID: 0...536 870 911
DLC	BYTE	Anzahl der zu übertragenden Bytes aus dem Array DATA Zulässige Werte: 0...8
DATA	ARRAY[0...7] OF BYTE	Das Array enthält maximal 8 Datenbytes
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht ausgeführt

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BOOL	TRUE (nur 1 Zyklus lang): Die Funktion hat den Sendeauftrag angenommen.

8.6.5 Funktion CAN1_EXT_RECEIVE

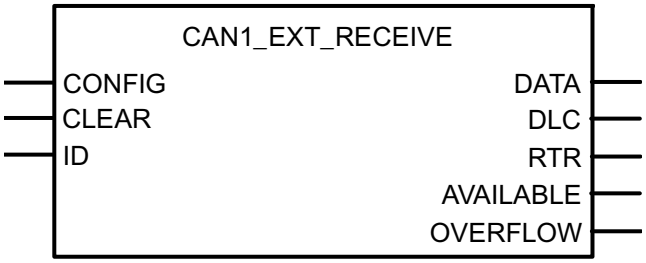
Enthalten in Bibliothek:

`ifm_CAN1_EXT_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

CAN1_EXT_RECEIVE konfiguriert ein Datenempfangsobjekt und liest den Empfangspuffer des Datenobjektes aus.

Die Funktion 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 CAN1_EXT_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 Auswerten des Ausgangs OVERFLOW kann ein Überlauf des Datenpuffers erkannt werden. Wenn OVERFLOW = TRUE, dann ist mindestens 1 Datenobjekt verloren gegangen.

HINWEIS

Soll diese Funktion verwendet werden, muss zuvor mit der Funktion CAN1_EXT (→ Seite 65) die 1. CAN-Schnittstelle für den erweiterten ID initialisiert werden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
CONFIG	BOOL	TRUE (nur 1 Zyklus lang): Datenobjekt konfigurieren FALSE: Funktion wird nicht ausgeführt
CLEAR	BOOL	TRUE: löscht den Datenpuffer (Warteschlange)
ID	DWORD	Nummer des Datenobjekt-Identifier Zulässige Werte Normal Frame: 0...2.047 (2^{11}) Zulässige Werte Extended Frame: 0...536.870.912 (2^{29})

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
DATA	ARRAY[0...7] OF BYTE	Das Array enthält maximal 8 Datenbytes
DLC	BYTE	Anzahl der übertragenen Bytes im Array DATA Mögliche Werte: 0...8.
RTR	BOOL	Wird nicht unterstützt
AVAILABLE	BYTE	Anzahl der eingegangenen Meldungen
OVERFLOW	BOOL	TRUE: Überlauf des Datenpuffers → Datenverlust! FALSE: Puffer noch nicht gefüllt

8.6.6 Funktion CAN1_EXT_ERRORHANDLER

Enthalten in Bibliothek:

`ifm_CAN1_EXT_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1070, CR1071

Funktionssymbol:

Beschreibung

Fehlerroutine zur Überwachung der 1. CAN-Schnittstelle.

Die Funktion CAN1_EXT_ERRORHANDLER überwacht die 1. CAN-Schnittstelle und wertet die CAN-Fehler aus. Tritt eine bestimmte Anzahl von Übertragungsfehlern auf, so wird der CAN-Teilnehmer error-passiv. Verringert sich die Fehlerhäufigkeit, wird der Teilnehmer wieder error-activ (= Normalzustand).

Ist ein Teilnehmer schon error-passiv und es treten weiterhin Übertragungsfehler auf, wird er vom Bus abgeschaltet (= bus-off) und das Fehlerbit CANx_BUSOFF gesetzt. Die Rückkehr an den Bus ist nur möglich, wenn der Bus-off-Zustand behoben wird (Signal BUSOFF_RECOVER).

Das Fehlerbit CANx_BUSOFF muss anschließend im Applikations-Programm zurückgesetzt werden.

HINWEIS

Wenn die automatische Bus-Recover-Funktion genutzt werden soll (Default-Einstellung), darf die Funktion CAN1_EXT_ERRORHANDLER **nicht** in das Programm eingebunden und instanziert werden!

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
BUSOFF_RECOVER	BOOL	TRUE (nur 1 Zyklus lang): > Neustart der CAN-Schnittstelle x > Bus-off-Zustand beheben FALSE: Funktion wird nicht ausgeführt

8.6.7 Funktion CAN2

(nur einsetzbar bei Geräten mit 2. CAN-Schnittstelle)

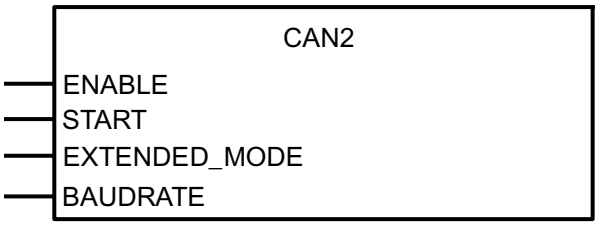
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201

Funktionssymbol:



Beschreibung

Die Funktion CAN2 initialisiert die 2. CAN-Schnittstelle.

Die Funktion muss aufgerufen werden, wenn die 2. CAN-Schnittstelle benutzt werden soll.

Eine Änderung der Baud-Rate wird erst gültig nach Spannung Aus/Ein. Die Baud-Raten von CAN 1 und CAN 2 können unterschiedlich eingestellt werden.

Der Eingang START wird nur für einen Zyklus bei Neustart oder Restart der Schnittstelle gesetzt.

Für die 2. CAN-Schnittstelle stehen u.a. Funktionsbibliotheken für SAE J1939 (→ Seite 148) zur Verfügung.

HINWEIS

Die Funktion muss **vor** den Funktionen CAN2_... ausgeführt werden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht ausgeführt
START	BOOL	TRUE (im 1. Zyklus): Schnittstelle wird initialisiert FALSE: Initialisierungszyklus ist beendet
EXTENDED_MODE	BOOL	TRUE: Identifier der 2. CAN-Schnittstelle arbeitet mit 29 Bits FALSE: Identifier der 2. CAN-Schnittstelle arbeitet mit 11 Bits
BAUDRATE	WORD	Baud-Rate [kBit/s] Zulässige Werte: 50, 100,125, 250, 500, 800, 1000 Voreinstellung = 125 kBit/s

8.6.8 Funktion CANx_TRANSMIT

x = Nr. 1...n der CAN-Schnittstelle (je nach Gerät, → Datenblatt)

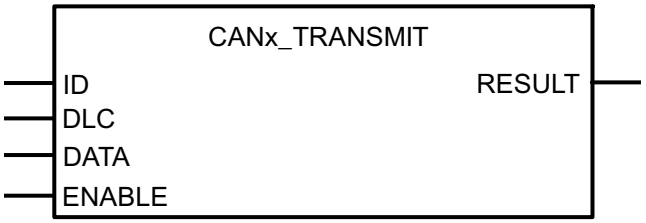
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
  Funktion NICHT für Sicherheitssignale!
  (Für Sicherheitssignale → Funktion CAN_SAFETY_TRANSMIT)
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015

Funktionssymbol:



Beschreibung

CANx_TRANSMIT übergibt ein CAN-Datenobjekt (Message) an den CAN-Controller zur Übertragung.

Die Funktion wird für jedes Datenobjekt im Programmzyklus aufgerufen, bei langen Programmzyklen auch mehrfach. Der Programmierer muss durch Auswertung des FB-Ausgangs 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 Eingang ENABLE kann die Ausführung der Funktion zeitweilig gesperrt werden (ENABLE = FALSE). Damit kann z.B. eine Busüberlastung verhindert werden.

Mehrere Datenobjekte können quasi gleichzeitig verschickt werden, wenn jedem Datenobjekt ein Merkerflag zugeordnet wird und mit diesem die Ausführung der Funktion über den ENABLE-Eingang gesteuert wird.

HINWEIS

Soll die Funktion CAN2_TRANSMIT verwendet werden, muss zuvor mit der Funktion CAN2 (→ Seite 71) die zweite CAN-Schnittstelle initialisiert werden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ID	WORD	Nummer des Datenobjekt-Identifier Zulässige Werte: 0...2047
DLC	BYTE	Anzahl der zu übertragenden Bytes aus dem Array DATA Zulässige Werte: 0...8
DATA	ARRAY[0...7] OF BYTE	Das Array enthält maximal 8 Datenbytes
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht ausgeführt

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BOOL	TRUE (nur 1 Zyklus lang): Die Funktion hat den Sendeauftrag angenommen.

8.6.9 Funktion CANx_RECEIVE

x = Nr. 1...n der CAN-Schnittstelle (je nach Gerät, → Datenblatt)

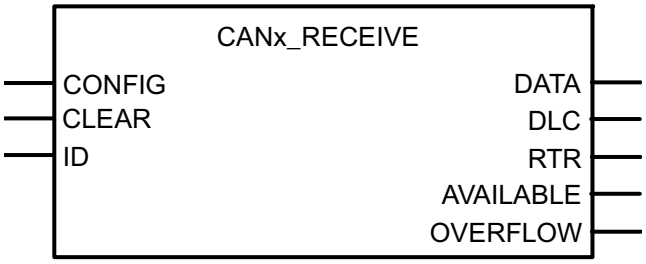
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
  Funktion NICHT für Sicherheitssignale!
  (Für Sicherheitssignale → Funktion CAN_SAFETY_RECEIVE)
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015

Funktionssymbol:



Beschreibung

CANx_RECEIVE konfiguriert ein Datenempfangsobjekt und liest den Empfangspuffer des Datenobjektes aus.

Die Funktion 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 CANx_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 Auswerten des Ausgangs OVERFLOW kann ein Überlauf des Datenpuffers erkannt werden. Wenn OVERFLOW = TRUE, dann ist mindestens 1 Datenobjekt verloren gegangen.

HINWEIS

Soll die Funktion CAN2_RECEIVE verwendet werden, muss zuvor mit der Funktion CAN2 (→ Seite 71) die zweite CAN-Schnittstelle initialisiert werden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
CONFIG	BOOL	TRUE (nur 1 Zyklus lang): Datenobjekt konfigurieren FALSE: Funktion wird nicht ausgeführt
CLEAR	BOOL	TRUE: löscht den Datenpuffer (Warteschlange)
ID	WORD	Nummer des Datenobjekt-Identifier Zulässige Werte: 0...2047

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
DATA	ARRAY[0...7] OF BYTE	Das Array enthält maximal 8 Datenbytes
DLC	BYTE	Anzahl der übertragenen Bytes im Array DATA Mögliche Werte: 0...8.
RTR	BOOL	Wird nicht unterstützt
AVAILABLE	BYTE	Anzahl der eingegangenen Meldungen
OVERFLOW	BOOL	TRUE: Überlauf des Datenpuffers → Datenverlust! FALSE: Puffer noch nicht gefüllt

8.6.10 Funktion CANx_RECEIVE_RANGE

x = Nr. 1...n der CAN-Schnittstelle (je nach Gerät, → Datenblatt)

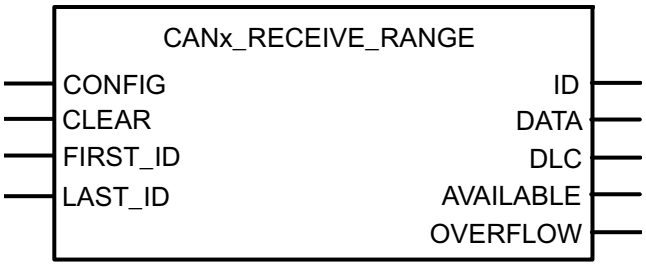
Enthalten in Bibliothek:

`ab ifm_CRnnnn_V05yyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
  Funktion NICHT für Sicherheitssignale!
  (Für Sicherheitssignale → Funktion CAN_SAFETY_RECEIVE)
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

CANx_RECEIVE_RANGE konfiguriert eine Folge von Datenempfangsobjekten und liest den Empfangspuffer der Datenobjekte aus.

Für die 1. CAN-Schnittstelle sind max. 2048 IDs je 11 Bits möglich.
Für die 2. CAN-Schnittstelle sind max. 256 IDs je 11 ODER 29 Bits möglich.
Die 2. CAN-Schnittstelle benötigt eine lange Initialisierungszeit. Damit der Watchdog nicht anspricht, sollte bei größeren Bereichen der Vorgang auf mehrere Zyklen verteilt werden (→ Beispiel Seite 80).

Die Funktion muss für jede Folge von Datenobjekten in der Initialisierungsphase einmalig aufgerufen werden, um dem CAN-Controller die Identifier der Datenobjekte bekannt zu machen.

Die Funktion darf für die selben IDs an den selben CAN-Schnittstellen NICHT gemischt eingesetzt werden mit der Funktion CANx_RECEIVE (→ Seite 75) oder der Funktion CANx_RECEIVE_RANGE.

Im weiteren Programmzyklus wird CANx_RECEIVE_RANGE 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 SOFORT abgerufen und weiterverarbeitet werden, da die Daten nur einen Zyklus lang bereitstehen.

Jeder Aufruf der Funktion dekrementiert das Byte AVAILABLE um 1. Ist der Wert von AVAILABLE gleich 0, sind keine Daten im Puffer.

Durch Auswerten des Ausgangs OVERFLOW kann ein Überlauf des Datenpuffers erkannt werden. Wenn OVERFLOW = TRUE, dann ist mindestens 1 Datenobjekt verloren gegangen.

Receive-Puffer: max. 16 Software-Puffer pro Identifier.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
CONFIG	BOOL	TRUE (nur 1 Zyklus lang): Datenobjekt konfigurieren FALSE: Funktion wird nicht ausgeführt
CLEAR	BOOL	TRUE: löscht den Datenpuffer (Warteschlange)
FIRST_ID	CAN1: WORD CAN2: DWORD	Nummer des ersten Datenobjekt-Identifiers der Folge. Zulässige Werte Normal Frame: 0...2.047 (2^{11}) Zulässige Werte Extended Frame: 0...536.870.912 (2^{29})
LAST_ID	CAN1: WORD CAN2: DWORD	Nummer des letzten Datenobjekt-Identifiers der Folge. Zulässige Werte Normal Frame: 0...2.047 (2^{11}) Zulässige Werte Extended Frame: 0...536.870.912 (2^{29}) LAST_ID muss größer sein als FIRST_ID.

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
ID	CAN1: WORD CAN2: DWORD	ID des ausgegebenen Datenobjekts
DATA	ARRAY[0...7] OF BYTE	Das Array enthält maximal 8 Datenbytes
DLC	BYTE	Anzahl der übertragenen Bytes im Array DATA Mögliche Werte: 0...8.
AVAILABLE	BYTE	Anzahl der Meldungen im Puffer
OVERFLOW	BOOL	TRUE: Überlauf des Datenpuffers → Datenverlust! FALSE: Puffer noch nicht gefüllt

Beispiel Initialisieren von CANx_RECEIVE_RANGE in 4 Zyklen

PLC_PRG (PRG-ST) (-1/181/-1/88)

```
0001 PROGRAM PLC_PRG
0002 VAR
0003     init : BOOL := FALSE;
0004     initstep : WORD := 1;
0005     can20 : CAN2;
0006     cr2 : CAN2_RECEIVE_RANGE;
0007     cnt : WORD;
0008 END_VAR
```

```
0001 (* CAN2 init. *)
0002 can20(ENABLE:= TRUE , START:= init, EXTENDED_MODE:= FALSE, BAUDRATE:= 125);
0003
0004 (* CAN2_RECEIVE_RANGE in mehreren Steps initialisieren *)
0005 CASE initstep OF
0006     1:
0007         cr2(CONFIG:= TRUE,CLEAR:= FALSE,FIRST_ID:= 16#100,LAST_ID:= 16#10F,ID=> ,DATA=> ,DLC=> ,AVAILABLE=> ,OVERFLOW=> );
0008         initstep := initstep + 1;
0009     2:
0010         cr2(CONFIG:= TRUE,CLEAR:= FALSE,FIRST_ID:= 16#110,LAST_ID:= 16#11F,ID=> ,DATA=> ,DLC=> ,AVAILABLE=> ,OVERFLOW=> );
0011         initstep := initstep + 1;
0012     3:
0013         cr2(CONFIG:= TRUE,CLEAR:= FALSE,FIRST_ID:= 16#120,LAST_ID:= 16#12F,ID=> ,DATA=> ,DLC=> ,AVAILABLE=> ,OVERFLOW=> );
0014         initstep := initstep + 1;
0015     4:
0016         cr2(CONFIG:= TRUE,CLEAR:= FALSE,FIRST_ID:= 16#130,LAST_ID:= 16#13F,ID=> ,DATA=> ,DLC=> ,AVAILABLE=> ,OVERFLOW=> );
0017         initstep := initstep + 1;
0018     ELSE
0019         cr2(CONFIG:=FALSE,CLEAR:= FALSE,FIRST_ID:= 16#100,LAST_ID:= 16#100,ID=> ,DATA=> ,DLC=> ,AVAILABLE=> ,OVERFLOW=> );
0020 END_CASE
0021
0022 init := FALSE;
0023
0024 (* Test *)
0025 IF cr2.available > 0 THEN
0026     cnt := cnt + 1;
0027 END_IF
```

8.6.11 Funktion CANx_EXT_RECEIVE_ALL

x = Nr. 1...n der CAN-Schnittstelle (je nach Gerät, → Datenblatt)

Enthalten in Bibliothek:

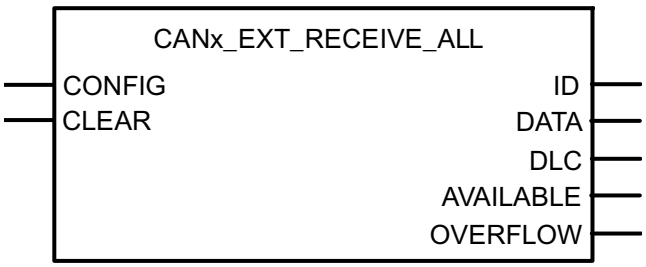
Für CAN-Schnittstelle 1: `ifm_CAN1_EXT_Vxxyyzz.LIB`

Für CAN-Schnittstelle 2...n: `ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
  Funktion NICHT für Sicherheitssignale!
  (Für Sicherheitssignale → Funktion CAN_SAFETY_RECEIVE)
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

CANx_EXT_RECEIVE_ALL konfiguriert alle Datenempfangsobjekte und liest den Empfangspuffer der Datenobjekte aus.

Die Funktion muss in der Initialisierungsphase einmalig aufgerufen werden, um dem CAN-Controller die Identifier der Datenobjekte bekannt zu machen.

Im weiteren Programmzyklus wird CANx_EXT_RECEIVE_ALL 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 Auswerten des Ausgangs OVERFLOW kann ein Überlauf des Datenpuffers erkannt werden. Wenn OVERFLOW = TRUE, dann ist mindestens 1 Datenobjekt verloren gegangen.

Receive-Puffer: max. 16 Software-Puffer pro Identifier.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
CONFIG	BOOL	TRUE (nur 1 Zyklus lang): Datenobjekt konfigurieren FALSE: Funktion wird nicht ausgeführt
CLEAR	BOOL	TRUE: löscht den Datenpuffer (Warteschlange)

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
ID	DWORD	ID des ausgegebenen Datenobjekts
DATA	ARRAY[0...7] OF BYTE	Das Array enthält maximal 8 Datenbytes
DLC	BYTE	Anzahl der übertragenen Bytes im Array DATA Mögliche Werte: 0...8.
AVAILABLE	BYTE	Anzahl der Meldungen im Puffer
OVERFLOW	BOOL	TRUE: Überlauf des Datenpuffers → Datenverlust! FALSE: Puffer noch nicht gefüllt

8.6.12 Funktion CANx_ERRORHANDLER

x = Nr. 1...n der CAN-Schnittstelle (je nach Gerät, → Datenblatt)

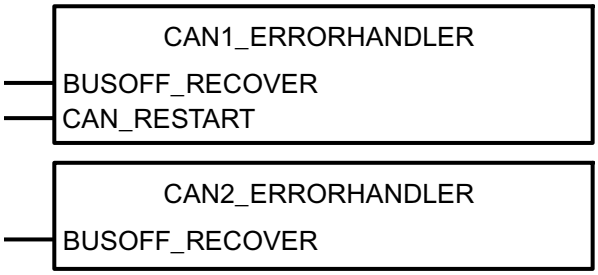
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015

Funktionssymbol:



Beschreibung

Fehlerroutine zur Überwachung der CAN-Schnittstellen

Die Funktion CANx_ERRORHANDLER überwacht die CAN-Schnittstellen und wertet die CAN-Fehler aus. Tritt eine bestimmte Anzahl von Übertragungsfehlern auf, so wird der CAN-Teilnehmer error-passiv. Verringert sich die Fehlerhäufigkeit, wird der Teilnehmer wieder error-activ (= Normalzustand).

Ist ein Teilnehmer schon error-passiv und es treten weiterhin Übertragungsfehler auf, wird er vom Bus abgeschaltet (= bus-off) und das Fehlerbit CANx_BUSOFF gesetzt. Die Rückkehr an den Bus ist nur möglich, wenn der Bus-off-Zustand behoben wird (Signal BUSOFF_RECOVER).

Der Funktionseingang CAN_RESTART dient zur Behebung anders gearteter CAN-Fehler. Die CAN-Schnittstelle wird dadurch neu initialisiert.

Das Fehlerbit muss anschließend im Applikations-Programm zurückgesetzt werden.

Das Vorgehen für den Neustart der Schnittstellen unterscheidet sich:

- für CAN-Schnittstelle 1 oder Geräte mit nur einer CAN-Schnittstelle:
  den Eingang CAN_RESTART = TRUE (nur 1 Zyklus) setzen
- für CAN-Schnittstelle 2:
  in der Funktion CAN2 (→ Seite 71) den Eingang START = TRUE (nur 1 Zyklus) setzen

HINWEIS

Die Funktion CAN2 (→ Seite 71) muss grundsätzlich zum Initialisieren der zweiten CAN-Schnittstelle ausgeführt werden, bevor Funktionen für diese genutzt werden können.

Wenn die automatische Bus-Recover-Funktion genutzt werden soll (Default-Einstellung), darf die Funktion CANx_ERRORHANDLER **nicht** in das Programm eingebunden und instanziert werden!

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
BUSOFF_RECOVER	BOOL	TRUE (nur 1 Zyklus lang): Bus-off-Zustand beheben FALSE: Funktion wird nicht ausgeführt
CAN_RESTART	BOOL	TRUE (nur 1 Zyklus lang): CAN-Schnittstelle 1 komplett neu initialisieren FALSE: Funktion wird nicht ausgeführt

8.7 ifm-CANopen-Bibliothek

CANopen Netzwerk-Konfiguration, Status- und Fehlerbehandlung

Bei allen programmierbaren Geräten wird die CANopen-Schnittstelle von CoDeSys® eingesetzt. Während Sie die Netzwerkkonfiguration und die Parametrierung der angeschlossenen Geräte direkt über die Programmiersoftware vornehmen, können die Fehlermeldungen nur über verschachtelte Variablenstrukturen im CANopen-Stack erreicht werden. Die nachfolgende Dokumentation zeigt Ihnen den Aufbau und die Anwendung der Netzwerkkonfiguration und beschreibt die Funktionen der ifm CANopen-Gerätebibliotheken.

Die Kapitel CANopen-Unterstützung durch CoDeSys (→ Seite 85), CANopen-Master (→ Seite 87), CAN-Device (→ Seite 102) und CAN-Netzwerkvariablen (→ Seite 110) beschreiben die internen Funktionen des CoDeSys®-CANopen-Stacks und ihre Anwendung. Außerdem bekommen Sie einen Einblick über die Anwendung des Netzwerkkonfigurators.

Die Kapitel über die Bibliotheken `ifm_CRnnnn_CANopenMaster_Vxxyyzz.lib` und `ifm_CRnnnn_CANopenSlave_Vxxyyzz.lib` beschreiben alle Funktionen zur Fehlerverarbeitung und zur Abfrage des Gerätestatus beim Einsatz als Master oder Slave (CAN-Device).

HINWEIS

Unabhängig vom eingesetzten Gerät haben alle Bibliotheken den gleichen Aufbau der Funktionsschnittstellen. Die geringfügigen Unterschiede (z.B. CANOPEN_LED_STATUS) werden direkt in den jeweiligen Funktionen beschrieben.

Es ist zwingend notwendig, dass Sie nur die jeweilige gerätespezifische Bibliothek einsetzen. Den Zusammenhang können Sie an der integrierten Geräte-Artikelnummer erkennen, z.B.:

CR0020: → `ifm_CR0020_CANopenMaster_V040003.lib`

→ Kapitel Target einrichten, Seite 14

Bei Verwendung anderer Bibliotheken kann das Gerät nicht mehr richtig funktionieren.

8.7.1 CANopen-Unterstützung durch CoDeSys

Allgemeines zu CANopen mit CoDeSys

CoDeSys® ist eines der führenden Systeme für die Programmierung von Steuerungssystemen nach dem internationalem Standard IEC 61131. Um CoDeSys® für den Anwender interessanter zu gestalten, wurden viele wichtige Funktionen in das Programmiersystem integriert, darunter auch ein Konfigurator für CANopen. Mit diesem CANopen-Konfigurator können Sie CANopen-Netzwerke (in einigen Punkten eingeschränkt) unter CoDeSys® konfigurieren.

CANopen ist als CoDeSys®-Bibliothek in IEC 61131-3 implementiert. Die Bibliothek stützt sich auf sehr einfache Basis-CAN-Funktionen ab, die als CAN-Treiber bezeichnet werden.

Durch die Realisierung der CANopen-Funktionen als CoDeSys®-Bibliothek ist eine einfache Skalierung des Zielsystems möglich. So verbraucht die CANopen-Funktion nur dann Zielsystem-Ressourcen, wenn die Funktion auch wirklich genutzt wird. Zur weiteren Schonung von Zielsystem-Ressourcen wird durch CoDeSys® automatisch eine genau der Konfiguration entsprechende Datenbasis für die CANopen-Master-Funktion generiert.

Ab der Programmiersystemversion CoDeSys® Version 2.3.6.0 kann ein ecomat*mobil*-Controller als CANopen-Master und als -Slave (CAN-Device) genutzt werden.

HINWEIS

Für alle ecomat*mobil*-Controller und das PDM360 smart müssen Sie die CANopen-Bibliotheken mit folgendem Zusatz einsetzen:

- Für CR0032 Target-Version bis V01, alle anderen Geräte bis V04.00.05: "**OptTable**"
- Für CR0032 Target-Version ab V02, alle anderen Geräte ab V05: "**OptTableEx**"

Wenn Sie ein Projekt neu anlegen, werden diese Bibliotheken im allgemeinen automatisch geladen. Sollten Sie selbst die Bibliotheken über die Bibliotheksverwaltung einfügen, müssen Sie auf die korrekte Auswahl achten.

Die CANopen-Bibliotheken ohne diesen Zusatz werden für alle anderen programmierbaren Geräte genutzt (z.B. PDM360 compact).

CANopen Begriffe und Implementation

Nach der CANopen-Spezifikation gibt es keine Master und Slaves in einem CAN-Netz. Statt dessen gibt es nach CANopen einen NMT-Master (NMT = Netzwerk-Management), einen Konfigurationsmaster usw., immer mit der Vorstellung, dass alle Teilnehmer eines CAN-Netzes gleichberechtigt sind.

Die Implementierung geht davon aus, dass ein CAN-Netz als Peripherie einer CoDeSys®-programmierbaren Steuerung dient. Demzufolge wird eine ecomat*mobil*-Steuerung oder ein PDM360-Display im CAN-Konfigurator von CoDeSys® als CAN-Master bezeichnet. Dieser Master ist NMT-Master und Konfigurationsmaster. Im Normalfall wird der Master dafür sorgen, dass das Netz in Betrieb genommen werden kann. Er übernimmt die Initiative, die einzelnen Nodes (= Netzwerk-Knoten) zu starten, die ihm per Konfiguration bekannt sind. Diese Nodes werden als Slaves bezeichnet.

Um den Master ebenfalls dem Status eines CANopen-Nodes näherzubringen, wurde ein Objektverzeichnis für den Master eingeführt. Auch kann der Master als SDO-Server (SDO = Service Data Object) auftreten und nicht nur in der Konfigurationsphase der Slaves als SDO-Client.

"Adressen" in CANopen

In CANopen werden diverse Arten von Adressen (IDs) unterschieden:

- **COB-ID**
  Der **C**AN-**Ob**ject-**Id**entifier adressiert die Nachricht (= das CAN-Objekt) im Geräteverzeichnis. Gleiche Nachrichten haben den selben COB-ID. Die COB-ID-Einträge im Objektverzeichnis enthalten u.a. den CAN-Identifier (CAN-ID).
- CAN-ID
  Der **CAN-Id**entifier identifiziert netzwerkweit CAN-Nachrichten. Der CAN-ID ist Bestandteil des COB-ID im Objektverzeichnis.
- Node-ID
  Der **Node-Id**entifier identifiziert netzwerkweit die CANopen-Geräte (Devices). Der Node-ID ist Bestandteil einiger vordefinierter CAN-IDs (untere 7 Bits).

8.7.2 CANopen-Master

Inhalt:

Abgrenzung zu anderen CANopen-Bibliotheken

Die von 3S (Smart Software Solutions) realisierte CANopen-Bibliothek grenzt sich in verschiedenen Punkten von auf dem Markt befindlichen Systemen ab. Sie wurde nicht entwickelt, um andere Bibliotheken namhafter Hersteller überflüssig zu machen, sondern ist bewusst für den Einsatz mit dem CoDeSys®-Programmier- und Laufzeitsystem optimiert.

Die Bibliotheken wurden nach der Spezifikation der CiA DS301, V402 erstellt.

Für Sie als Anwender der CoDeSys®-CANopen-Bibliothek ergeben sich folgende Vorteile:

- Die Implementierung ist unabhängig vom Zielsystem und damit praktisch auf jeder mit CoDeSys®-programmierbaren Steuerung direkt verwendbar.
- Das komplette System beinhaltet den CANopen-Konfigurator und die Einbindung in das Entwicklungssystem.
- Die CANopen-Funktionalität ist nachladbar. Das bedeutet, dass die CANopen-Funktionen ohne Änderung des Betriebssystems geladen und aktualisiert werden können.
- Die Ressourcen des Zielsystems werden geschont, da nicht die Ressourcen für eine Maximalkonfiguration vorgehalten werden.
- Automatisches Aktualisieren der Ein- und Ausgänge ohne zusätzliche Maßnahmen.

Folgende in CANopen definierten Funktionen werden zur Zeit von der ifm-CANopen-Bibliothek unterstützt:

- **PDOs Senden:** Master sendet zu den Slaves (Slave = Knoten, Device)
  Senden ereignisgesteuert (d.h. bei Änderung), zeitgesteuert (RepeatTimer) oder als synchrone PDOs, d.h. immer wenn ein SYNC vom Master gesendet wurde. Auch eine externe SYNC-Quelle kann benutzt werden, um das Senden von synchronen PDOs zu initiieren.
- **PDOs Empfangen:** Master empfängt vom Slave
  Je nach Slave: ereignisgesteuert, abfragegesteuert, azyklisch und zyklisch.
- **PDO-Mapping**
  Zuordnung zwischen lokalem Objektverzeichnis und PDOs vom/zum CAN-Device (wenn vom Slave unterstützt).
- **SDO Senden und Empfangen** (unsegmentiert, d.h. 4 Bytes pro Objektverzeichnis-Eintrag)
  Automatische Konfiguration aller Slaves über SDOs beim Systemstart.
  Applikationsgesteuertes Senden und Empfangen von SDOs zu konfigurierten Slaves.
- **Synchronisation**
  Automatisches Senden von SYNC-Nachrichten durch den CANopen-Master.
- **Nodeguarding**
  Automatisches Senden von Guarding-Nachrichten und Überwachung der Lifetime für jeden entsprechend konfigurierten Slave.
  Wir empfehlen: Für aktuelle Geräte besser mit Heartbeat arbeiten, weil dann die Buslast niedriger ist.

- **Heartbeat**
  Automatisches Senden und Überwachen von Heartbeat-Nachrichten.
- **Emergency**
  Empfangen und Speichern von Emergency-Nachrichten von den konfigurierten Slaves.
- **Node-ID** und **Baudrate** in den Slaves setzen
  Durch Aufruf einer einfachen Funktion können Node-ID und Baudrate eines Slaves zur Laufzeit der Applikation gesetzt werden.

Folgende in CANopen definierten Funktionen werden von der 3S (Smart Software Solutions) CANopen-Bibliothek derzeit **nicht** unterstützt:

- Dynamische Identifier-Zuordnung
- Dynamische SDO-Verbindungen
- Blockweiser SDO-Transfer, segmentierter SDO-Transfer (die Funktionalität kann über die Funktion CANx_SDO_READ (→ Seite 142) und die Funktion CANx_SDO_WRITE (→ Seite 144) in der jeweiligen ifm-Gerätebibliothek realisiert werden).
- Alle oben nicht genannten Möglichkeiten des CANopen Protokolls.

Ein CANopen-Projekt erstellen

Die Erstellung eines neuen Projektes mit einem CANopen-Master wird nachfolgend schrittweise beschrieben. Dabei gehen wir davon aus, dass Sie CoDeSys® auf dem Rechner bereits fertig installiert haben und die Target- und EDS-Dateien ebenfalls richtig installiert oder kopiert wurden.

Eine weitergehende detaillierte Beschreibung zur Einstellung und Anwendung des Dialogs Steuerungs- und CANopen-Konfiguration → CoDeSys®-Handbuch unter [Ressourcen] > [Steuerungskonfiguration] und in der Online-Hilfe.

► Nach der Neuanlage eines Projektes (→ Kapitel Target einrichten, Seite 14) in der Steuerungskonfiguration über [Einfügen] > [Unterelement anhängen] den CANopen-Master einfügen.

> Bei Steuerungen mit 2 oder mehr CAN-Schnittstellen wird automatisch Schnittstelle 1 für den Master konfiguriert.

> Die folgenden Bibliotheken und Software-Module werden automatisch eingebunden:

- die `STANDARD.LIB`, welche die in der IEC-61131 definierten Standardfunktionen für die Steuerung zu Verfügung stellt,
- die `3S_CanOpenManager.LIB`, welche die CANopen-Basisfunktionalitäten zur Verfügung stellt
  (ggf. `3S_CanOpenManagerOptTable.LIB` für C167-Controller),
- eine oder mehrere der Bibliotheken `3S_CANopenNetVar.LIB`, `3S_CANopenDevice.LIB` und `3S_CANopenMaster.LIB` (ggf. `3S_...OptTable.LIB` für C167-Controller), je nach gewünschter Funktionalität,
- die Systembibliotheken `SysLibSem.LIB` und `SysLibCallback.LIB`.

► Um die vorbereiteten Netzwerkdiagnose-, Status- und EMCY-Funktion zu nutzen, die Bibliothek `ifm_CRnnnn_CANopenMaster_Vxxyyzz.LIB` manuell im Bibliotheksverwalter einfügen. Ohne diese Bibliothek müssen Sie die Netzwerkinformationen direkt aus den verschachtelten Strukturen der CoDeSys®-CANopen-Bibliotheken auslesen.

► Zusätzlich die folgenden Bibliotheken und Software-Module einbinden:

- die Gerätebibliothek für die jeweilige Hardware, z.B. `ifm_CR0020_Vxxyyzz.LIB`. Diese Bibliothek stellt alle gerätespezifischen Funktionen zur Verfügung.
- EDS-Dateien für alle Slaves, die am Netzwerk betrieben werden sollen. Die EDS-Dateien für alle ifm-CANopen-Slaves stellt die **ifm electronic gmbh** zur Verfügung
  (→ Kapitel Programmiersystem über Templates einrichten, Seite 16).
  Für die EDS-Dateien von Fremd-Knoten ist der jeweilige Hersteller verantwortlich.

HINWEIS

Die CANopen-Unterstützung durch CoDeSys® kann bei den ecomatmobil-Controllern und beim PDM360-smart nur für die 1. CAN-Schnittstelle aktiviert werden.

Wurde schon der CAN-Master eingefügt, kann die Steuerung nicht mehr als CAN-Device über CoDeSys® genutzt werden.

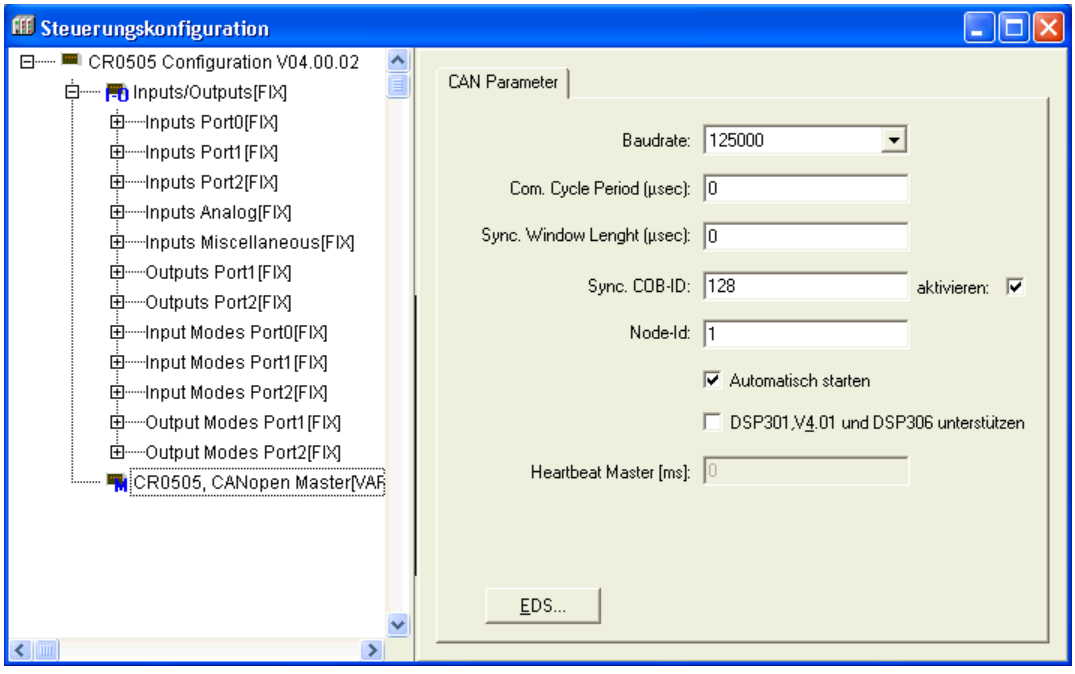
Die Implementierung eines eigenen Protokolls auf Schnittstelle 2 oder Nutzung des Protokolls nach SAE J1939 oder ISO11992 ist aber jederzeit möglich.

Beim PDM360 und beim PDM360-compact können beide CAN-Schnittstellen als CANopen-Master oder CAN-Device genutzt werden.

Geräte der Serie CRnn32 können auf allen CAN-Schnittstellen (→ Seite 49) mit allen Protokollen genutzt werden.

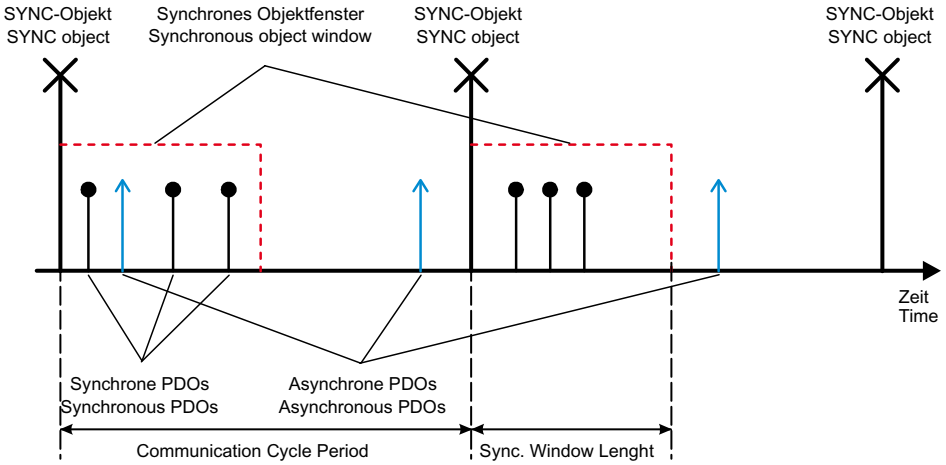
Register [CAN-Parameter]

In diesem Dialogfenster können für den Master die wichtigsten Parameter eingestellt werden. Bei Bedarf kann über die Schaltfläche [EDS...] der Inhalt der Master-EDS-Datei angesehen werden. Dieser Button wird nur angezeigt, wenn die EDS-Datei (z.B. `CR0020MasterODEntry.EDS`) im Verzeichnis `...\CoDeSys V2.3\Library\PLCConf` vorhanden ist.
Aus dieser EDS-Datei wird bei der Übersetzung des Applikations-Programms automatisch das Objektverzeichnis des Masters erzeugt.



Baudrate
Wählen Sie an dieser Stelle bitte die Baudrate für den Master aus. Sie muss der Übertragungsgeschwindigkeit der anderen Netzwerkteilnehmer entsprechen.

Communication Cycle Period/Sync. Window Length
Nach Ablauf der [Communication Cycle Period] wird eine SYNC-Nachricht vom Master verschickt. Die [Sync. Window Length] gibt die Zeit an, in der synchrone PDOs von den anderen Netzwerkteilnehmern verschickt und vom Master empfangen werden müssen.
Da in den meisten Applikationen keine besonderen Anforderungen an das SYNC-Objekt gestellt werden, können Sie für die [Communication Cycle Period] und die [Sync. Window Length] die gleiche Zeit einstellen. Bitte beachten Sie, dass die Zeit in [µsec] eingegeben wird (der Wert 50000 entspricht 50 ms).



Sync. COB-ID
In diesem Feld kann der Identifier für die SYNC-Nachricht einstellt werden. Diese wird immer nach Ablauf der Communication Cycle Period verschickt. Der Defaultwert ist 128 und sollte im Normalfall nicht geändert werden. Um das Versenden der SYNC-Nachricht zu aktivieren, muss das Kontrollfeld [aktivieren] gesetzt sein.

HINWEIS

Die SYNC-Nachricht wird immer am Anfang eines Programmzyklus erzeugt. Danach werden die Eingänge gelesen, das Programm abgearbeitet, die Ausgänge geschrieben und zuletzt alle synchronen PDOs gesendet.

Bitte beachten Sie, dass sich die SYNC-Zeit verlängert, wenn die eingestellte SYNC-Zeit kürzer als die Programmzykluszeit ist.

Beispiel: Communication Cycle Period = 10 ms und Programmzykluszeit = 30 ms.
Die SYNC-Nachricht wird erst nach 30 ms versendet.

Node-Id
Setzen Sie in diesem Feld die Knotennummer (nicht den Download-ID!) des Masters ein. Die Knotennummer darf im Netzwerk nur einmal vorkommen, andernfalls kommt es zu Kommunikationsstörungen.

Automatisch starten
Das Netzwerk und die angeschlossenen Knoten werden nach einer erfolgreichen Konfiguration in den Zustand "operational" gesetzt und damit gestartet.
Ist das Optionsfeld nicht angewählt, muss das Netzwerk manuell gestartet werden.

Heartbeat
Wenn die anderen Teilnehmer im Netzwerk Heartbeat unterstützen, kann die Option [DSP301, V4.01... unterstützen] selektiert werden. Bei Bedarf kann der Master noch ein eigenes Heartbeat-Signal nach Ablauf der eingestellten Zeit erzeugen.

CANopen-Slaves einfügen und konfigurieren

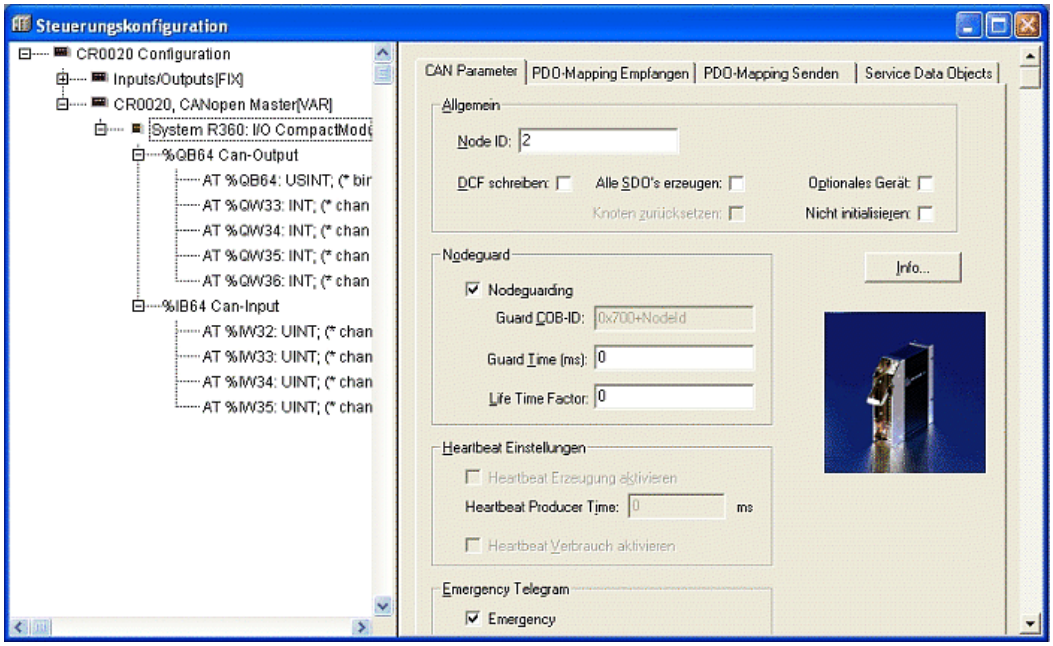
Als nächstes können Sie nun die CAN-Slaves einfügen. Dazu müssen Sie erneut den Dialog in der Steuerungskonfiguration [Einfügen] > [Unterelement anhängen] aufrufen. Es steht Ihnen eine Liste der im Verzeichnis PLC_CONF gespeicherten CANopen-Gerätebeschreibungen (EDS-Dateien) zur Verfügung. Durch Auswahl des entsprechenden Gerätes wird dieses direkt in den Baum der Steuerungskonfiguration eingefügt.

HINWEIS

Wird ein Slave über den Konfigurationsdialog in CoDeSys® hinzugefügt, wird für jeden Knoten dynamisch Quellcode in das Applikations-Programm integriert. Gleichzeitig verlängert jeder zusätzlich hinzugefügte Slave die Zykluszeit des Applikations-Programms. Das bedeutet: in einem Netzwerk mit vielen Slaves kann der Master keine weiteren zeitkritischen Aufgaben (z.B. den FB OCC_TASK) abarbeiten.

Ein Netzwerk mit 27 Slaves hat eine Grund-Zykluszeit von 30 ms.

Bitte beachten Sie, dass die maximale Zeit für einen SPS-Zyklus von ca. 50 ms nicht überschritten werden sollte (Watchdog-Zeit: 100 ms).



Register [CAN Parameter]

Node-ID
Die Node-ID dient zur eindeutigen Identifizierung des CAN-Moduls und entspricht der am Modul eingestellten Nummer zwischen 1 und 127. Der ID wird dezimal eingegeben und wird automatisch um eins erhöht, wenn Sie ein neues Modul hinzufügen.

DCF schreiben
Ist [DCF schreiben] aktiviert, wird nach dem Einfügen einer EDS-Datei im eingestellten Verzeichnis für Übersetzungsdateien eine DCF-Datei erstellt, deren Namen sich zusammensetzt aus dem Namen der EDS-Datei und der angehängten Node-ID.

Alle SDOs erzeugen
Ist diese Option aktiviert, werden für alle Kommunikationsobjekte SDOs erzeugt. (Default-Werte werden nicht erneut geschrieben!)

Knoten zurücksetzen
Der Slave wird zurückgesetzt ("load"), sobald die Konfiguration in die Steuerung geladen wird.

Optionales Gerät
Ist die Option [Optionales Gerät] aktiviert, versucht der Master nur einmal, von diesem Knoten zu lesen. Bei fehlender Antwort wird der Knoten ignoriert und der Master geht in den normalen Betriebszustand über.
Wird der Slave zu einem späteren Zeitpunkt an das Netzwerk angeschlossen und erkannt, wird er automatisch gestartet.
Dazu müssen Sie die Option [Automatisch starten] in den CAN-Parametern des Masters angewählt haben.

Nicht initialisieren
Wird diese Option aktiviert, nimmt der Master den Knoten sofort in Betrieb, ohne ihm Konfigurations-SDOs zu schicken. (Die SDO-Daten werden aber dennoch erzeugt und auf der Steuerung gespeichert.)

Nodeguarding- / Heartbeat-Einstellungen
Je nach Gerät müssen Sie [Nodeguarding] und [Life Time Factor] oder [Heartbeat] einstellen.
Wir empfehlen: Für aktuelle Geräte besser mit Heartbeat arbeiten, weil dann die Buslast niedriger ist..

Emergency Telegram
Die Option ist im Normalfall angewählt. Die EMCY-Nachrichten werden mit dem angegebenen Identifier übertragen.

Communication Cycle
In ganz speziellen Anwendungsfällen können Sie an dieser Stelle eine Überwachungszeit für die vom Master erzeugten SYNC-Nachrichten einstellen. Bitte beachten Sie, dass diese Zeit länger als die SYNC-Zeit des Masters sein muss. Der optimale Wert muss ggf. experimentell ermittelt werden. Nodeguarding und Heartbeat reicht in den meisten Fällen zur Knotenüberwachung aus.

Register [PDO-Mapping empfangen] und [PDO-Mapping senden]

Die Registerkarten [PDO-Mapping empfangen] und [PDO-Mapping senden] im Konfigurationsdialog eines CAN-Moduls ermöglichen es, dass in der EDS-Datei beschriebene "Mapping" (Zuordnung zwischen lokalem Objektverzeichnis und PDOs vom/zum CAN-Device) des Moduls zu verändern (wenn es vom CAN-Modul unterstützt wird).
Auf der linken Seite stehen alle "mapbaren" Objekte der EDS-Datei zur Verfügung und können zu den PDOs (Process Data Objects) der rechten Seite hinzugefügt oder wieder entfernt werden. Die [StandardDataTypes] können eingefügt werden, um im PDO leere Zwischenräume zu erzeugen.

Einfügen
Mit der Schaltfläche [Einfügen] können Sie weitere PDOs erzeugen und mit entsprechenden Objekten belegen. Über die eingefügten PDOs erfolgt die Zuordnung der Ein- und Ausgänge zu den IEC-Adressen. In der Steuerungskonfiguration werden die vorgenommenen Einstellungen nach Verlassen des Dialoges sichtbar. Die einzelnen Objekte können dort mit symbolischen Namen belegt werden.

Eigenschaften
Über Eigenschaften lassen sich die in der Norm definierten Eigenschaften der PDOs in einem Dialog editieren:

COB-ID	Jede PDO-Nachricht benötigt einen eindeutigen COB-ID (Communication Object Identifier). Wird eine Option von dem Modul nicht unterstützt oder darf der Wert nicht verändert werden, so erscheint das Feld grau und kann nicht editiert werden.
Inhibit Time	Die Inhibit Time (100 µs) ist die minimale Zeit zwischen zwei Nachrichten dieses PDOs, damit die Nachrichten, die bei Änderung des Wertes übertragen werden, nicht zu häufig versendet werden. Die Einheit ist 100 µs.
Transmission Type	Bei Transmission Type erhalten Sie eine Auswahl von möglichen Übertragungmodi für dieses Modul: **acyclic – synchronous** Das PDO wird nach einer Änderung mit dem nächsten SYNC übertragen. **cyclic – synchronous** Das PDO wird synchron übertragen, wobei [Number of SYNCs] die Anzahl der Synchronisationsnachrichten angibt, die zwischen zwei Übertragungen dieses PDOs liegen. **asynchronous – device specific** Das PDO wird ereignisgesteuert, d.h. wenn sich der Wert ändert, übertragen. Welche Daten auf diese Weise übertragen werden können, ist im Geräteprofil festgelegt. **asynchronous – manufacturer specific** Das PDO wird ereignisgesteuert, d.h. wenn sich der Wert ändert, übertragen. Welche Daten auf diese Weise übertragen werden, wird vom Gerätehersteller festgelegt. **(a)synchronous – RTR only** Diese Dienste sind nicht implementiert. **Number of SYNCs** Abhängig vom Transmission Type ist dieses Feld editierbar zur Eingabe der Anzahl der Synchronisationsnachrichten (Definition in [CAN-Parameter-Dialog], [Com. Cycle Period], [Sync Window Length], [Sync. COB-Id]), nach denen das PDO wieder versendet werden soll. **Event-Time** Abhängig vom Transmission Type wird hier die Zeitspanne in Millisekunden [ms] angegeben, die zwischen zwei Übertragungen des PDOs liegen soll.

Register [Service Data Objects]

Index, Name, Wert, Typ und Default
Hier werden alle Objekte der EDS- oder DCF-Datei aufgelistet, die im Bereich von Index 0x2000 bis 0x9FFF liegen und als beschreibbar definiert sind. Zu jedem Objekt werden Index, Name, Wert, Typ und Default angegeben. Der Wert kann verändert werden. Markieren Sie den Wert und drücken Sie die [Leertaste]. Nach Änderung können Sie den Wert durch die Taste [Eingabe] bestätigen oder mit [ESC] verwerfen.

Bei der Initialisierung des CAN-Buses werden die eingestellten Werte in Form von SDOs (Service Data Object) an die CAN-Module übertragen und haben damit direkten Einfluss auf das Objektverzeichnis des CAN-Slaves. Sie werden im Normalfall bei jedem Start des Applikations-Programms neu geschrieben – unabhängig davon, ob sie im CAN-Device dauerhaft gespeichert werden.

Der Master zur Laufzeit

Hier lesen Sie über Funktionalität der CANopen-Master-Bibliotheken zur Laufzeit.

Die CANopen-Master-Bibliothek stellt der CoDeSys®-Applikation implizite Dienste zur Verfügung, die für die meisten Applikationen ausreichend sind. Diese Dienste werden für den Anwender transparent integriert und stehen in der Applikation ohne zusätzliche Aufrufe zur Verfügung. In der nachfolgenden Beschreibung wird davon ausgegangen, dass Sie zur Nutzung der Netzwerkdiagnose-, Status- und EMCY-Funktionen die Bibliothek `ifm_CRnnnn_CANopenMaster_Vxxyyzz.LIB` manuell im Bibliotheksverwalter eingefügt haben.

Zu den Diensten der CANopen-Master-Bibliothek zählen:

Reset aller konfigurierten Slaves am Bus beim Systemstart
Um die Slaves zurückzusetzen, wird standardmäßig das NMT-Kommando "Reset Remote Node" benutzt, explizit für jeden Slave einzeln. (NMT steht nach CANopen für **N**etwork **M**anagmen**t**. Die einzelnen Kommandos sind im CAN-Dokument DSP301 beschrieben.) Um Slaves mit weniger leistungsstarken CAN-Controllern nicht zu überlasten, ist es sinnvoll, die Slaves mit einem Kommando "All Remote Nodes" zurückzusetzen.
Der Dienst wird für **alle** konfigurierten Slaves ausgeführt mit der Funktion CANx_MASTER_STATUS (→ Seite 125) mit GLOBAL_START=TRUE. Sollen die Slaves **einzeln** zurückgesetzt werden, muss dieser Eingang auf FALSE gesetzt werden.

Abfrage des Slave-Gerätetyps mittels SDO (Abfrage des Objekts 0x1000) und Vergleich mit der konfigurierten Slave-ID
Ausgabe eines Fehlerstatus' für die Slaves, von denen ein falscher Gerätetyp empfangen wurde. Die Anfrage wird nach 0,5 s wiederholt, wenn:
kein Gerätetyp wurde empfangen
UND Slave wurde in der Konfiguration **nicht** als optional markiert
UND Timeout ist **nicht** abgelaufen.

Konfiguration aller fehlerfrei detektierten Geräte mittels SDO
Jedes SDO wird auf Antwort überwacht und wiederholt, wenn sich innerhalb der Überwachungszeit der Slave nicht meldet.

Automatische Konfiguration von Slaves mittels SDOs bei laufendem Busbetrieb
Voraussetzung: Der Slave hat sich mittels Bootup-Message beim Master anmeldet.

Start aller fehlerfrei konfigurierten Slaves nach dem Ende der Konfiguration des betreffenden Slaves
Zum Starten der Slaves wird normalerweise das NMT-Kommando "Start remote node" benutzt. Wie beim "Reset" kann dieses Kommando durch "Start All Remote Nodes" ersetzt werden. Der Dienst ist über die Funktion CANx_Master_STATUS mit GLOBAL_START=TRUE aufrufbar.

Zyklisches Senden der SYNC-Message
Dieser Wert ist nur bei der Konfiguration einstellbar.

Nodeguarding mit Lifetime-Überwachung für jeden Slave einstellbar
Der Fehlerstatus kann für max. 8 Slaves über die Funktion CANx_MASTER_STATUS (→ Seite 125) mit ERROR_CONTROL=TRUE überwacht werden.
Wir empfehlen: Für aktuelle Geräte besser mit Heartbeat arbeiten, weil dann die Buslast niedriger ist.

Heartbeat vom Master an die Slaves und überwachen der Heartbeats der Slaves
Der Fehlerstatus kann für max. 8 Slaves über die Funktion CANx_MASTER_STATUS mit ERROR_CONTROL=TRUE überwacht werden.

Empfangen von Emergency-Messages für jeden Slave mit Speicherung der zuletzt empfangenen Emergency-Messages für jeden Slave getrennt
Die Fehlernachrichten können über die Funktion CANx_MASTER_STATUS mit EMERGENCY_OBJECT_SLAVES=TRUE ausgelesen werden. Zusätzlich liefert diese Funktion die zuletzt erzeugte EMCY-Message am Ausgang GET_EMERGENCY.

Netzwerk starten

Hier lesen Sie über das Starten des CANopen-Netzwerks.

Nach einem Download des Projekts auf die Steuerung oder einem Reset der Applikation wird das CAN-Netz vom Master neu hochgefahren. Das geschieht immer in der gleichen Reihenfolge von Aktionen:

- Alle Slaves werden zurückgesetzt, außer wenn sie als [nicht initialisieren] im Konfigurator markiert sind. Das Zurücksetzen geschieht einzeln mit dem NMT-Kommando "Reset Node" (0x81), jeweils mit der Node-ID des Slaves. Wurde über die Funktion CANx_MASTER_STATUS (→ Seite 125) das Flag GLOBAL_START gesetzt, wird zum Hochfahren des Netzes das Kommando einmal mit Node-ID 0 benutzt.
- Alle Slaves werden konfiguriert. Dazu wird zunächst das Objekt 0x1000 des Slaves abgefragt.
  - Wenn der Slave innerhalb der Überwachungszeit von 0,5 Sekunden antwortet, wird das jeweils nächste Konfigurations-SDO gesendet.
  - Ist ein Slave als [optional] markiert und antwortet nicht innerhalb der Überwachungszeit auf die Abfrage des Objekts 0x1000, wird er als nicht vorhanden markiert und keine weiteren SDOs werden an ihn geschickt.
  - Wenn ein Slave auf die Abfrage des Objekts 0x1000 mit einem anderen Typ als dem konfigurierten (in den unteren 16 Bit) antwortet, wird er zwar konfiguriert, aber als falscher Typ markiert.
- Alle SDOs werden jeweils solange wiederholt, bis innerhalb einer Überwachungszeit eine Antwort des Slaves gesehen wurde. Hier kann die Applikation den Hochlauf der einzelnen Slaves überwachen und ggf. durch Setzen des Flags SET_TIMEOUT_STATE im NODE_STATE_SLAVE-Array der Funktion CANx_MASTER_STATUS (→ Seite 125) reagieren.
- Wenn der Master eine Heartbeat-Zeit ungleich 0 konfiguriert hat, beginnt die Erzeugung des Hearbeats sofort nach dem Starten der Mastersteuerung.
- Nachdem alle Slaves ihre Konfigurations-SDOs erhalten haben, beginnt für Slaves mit konfiguriertem Nodeguarding das Guarding.
- Wenn der Master auf [automatisch starten] konfiguriert wurde, werden jetzt alle Slaves einzeln vom Master gestartet. Dazu wird das NMT-Kommando "Start Remote Node" (0x01) benutzt. Wurde über die Funktion CANx_MASTER_STATUS (→ Seite 125) das Flag GLOBAL_START gesetzt, dann wird das Kommando mit Node-ID 0 genutzt und somit alle Slaves mit einem "Start all Nodes" gestartet.
- Es werden mindestens einmal alle konfigurierten TX-PDOs gesendet (für die Slaves sind das RX-PDOs).
- Wenn [automatisch starten] deaktiviert wurde, müssen die Slaves einzeln über das Flag START_NODE im NODE_STATE_SLAVE-Array oder über den Funktionseingang GLOBAL_START der Funktion CANx_MASTER_STATUS gestartet werden.

Netzwerkzustände

Hier lesen Sie, wie Sie die Zustände des CANopen-Netzwerks interpretieren und darauf reagieren können.

Beim Hochlauf (→ Netzwerk starten, Seite 95) des CANopen Netzwerks und während des Betriebs durchlaufen die einzelnen Funktionsblöcke der Bibliothek verschiedene Zustände.

HINWEIS

Im Monitorbetrieb (Online-Modus) von CoDeSys® können Sie die Zustände des CAN-Netzwerkes in der globalen Variablenliste "Can Open implicit variables" einsehen. Dazu sind genaue Kenntnisse von CANopen und der Struktur der CoDeSys®-CANopen-Bibliotheken notwendig.

Um den Zugriff zu erleichtern, steht Ihnen die Funktion CANx_MASTER_STATUS (→ Seite 125) aus der Bibliothek `ifm_CRnnnn_CANopenMaster_Vxxyyzz.LIB` zur Verfügung.

Hochlauf des CANopen-Masters

Während des Hochlaufs des CAN-Netzwerks durchläuft der Master verschiedene Zustände, die Sie über den Ausgang NODE_STATE der Funktion CANx_MASTER_STATUS ablesen können.

Status	Beschreibung
0, 1, 2	Die werden vom Master automatisch und in den ersten Zyklen nach einem SPS-Start durchlaufen.
3	Der Status 3 des Masters wird für einige Zeit beibehalten. Im Status 3 konfiguriert der Master seine Slaves. Dazu werden den Slaves der Reihe nach alle vom Konfigurator erzeugten SDOs gesendet.
5	Nachdem an die Slaves alle SDOs übertragen wurden, geht der Master in den Status 5 und bleibt in diesem Status. Status 5 ist für den Master der normale Betriebszustand.

Immer, wenn ein Slave auf eine SDO-Anfrage (Upload oder Download) nicht antwortet, dann wird die Anfrage wiederholt. Der Master verlässt den Status 3, wie oben beschrieben, aber erst, wenn alle SDOs erfolgreich übertragen wurden. So kann also erkannt werden, ob ein Slave fehlt oder ob der Master nicht alle SDOs richtig empfangen kann. Dabei ist es für den Master unerheblich, ob ein Slave mit einer Bestätigung oder einem Abort antwortet. Für den Master ist nur von Interesse, ob er überhaupt eine Antwort empfangen hat.

Eine Ausnahme stellt ein als [optional] markierter Slave dar. Optionale Slaves werden nur einmal nach ihrem 0x1000er-Objekt gefragt. Wenn sie nicht innerhalb von 0,5 s antworten, wird der Slave vom Master zunächst ignoriert und der Master geht auch ohne weitere Reaktion dieses Slaves in Status 5.

Hochlauf der CANopen-Slaves

Die Stati eines Slaves können Sie über das Array NODE_STATE_SLAVE der Funktion CANx_MASTER_STATUS (→ Seite 125) auslesen. Während des Hochlaufs des CAN-Netzwerks durchläuft der Slave die Stati -1, 1 und 2 automatisch. Dabei sind diese Stati wie folgt zu interpretieren:

Status	Beschreibung
-1	Der Slave wird durch die NMT-Nachricht [Reset Node] zurückgesetzt und wechselt selbständig in den Status 1.
1	Der Slave wechselt nach einer maximalen Zeit von 2 s oder sofort nach Empfang seiner Bootup-Message in den Status 2.
2	Der Slave wechselt nach einer Verzögerungszeit von 0,5 s automatisch in den Status 3. Diese Zeit entspricht der Erfahrung, dass viele CANopen-Geräte nicht sofort bereit sind, ihre Konfigurations-SDOs zu empfangen, nachdem sie Ihre Bootup-Message verschickt haben.
3	Im Status 3 wird der Slave konfiguriert. Der Slave bleibt solange im Status 3, bis er alle vom Konfigurator erzeugten SDOs erhalten hat. Dabei spielt es keine Rolle, ob während der Konfiguration vom Slave SDO-Transfers mit Abort (Fehler) oder ob alle fehlerfrei beantwortet wurden. Nur die vom Slave erhaltene Antwort als solche ist wichtig – nicht ihr Inhalt. Wenn im Konfigurator die Option [Knoten zurücksetzen] aktiviert wurde, wird nach dem Senden des Objekts 0x1011 Subindex 1, der dann den Wert "load" enthält, ein erneuter Reset des Nodes durchgeführt. Der Slave wird dann wieder mit dem Upload des Objekts 0x1000 angefragt. Slaves, bei denen während der Konfigurationsphase ein Problem auftritt, bleiben im Status = 3 oder wechseln nach der Konfigurationsphase direkt in einen Fehlerstatus (Status > 5).

Nachdem der Slave die Konfigurationsphase durchlaufen hat, kann er in folgende Stati übergehen:

Status	Beschreibung
4	Ein Knoten wechselt immer in den Status 4, außer es handelt sich um einen "optionalen" Slave und er wurde als nicht am Bus verfügbar detektiert (Abfrage Objekt 0x1000), oder der Slave ist zwar vorhanden, aber hat auf die Abfrage des Objekts 0x1000 mit einem anderen Typ in den unteren 16 Bits reagiert, als der Konfigurator erwartet hat.
5	Status 5 ist der normale Betriebszustand des Slaves. Wenn der Master auf [Automatisch starten] konfiguriert wurde, wird der Slave im Status 4 gestartet (d.h. es wird eine "Start Node"-NMT-Nachricht erzeugt) und der Slave wechselt automatisch nach Status 5. Wurde von der Applikation das Flag GLOBAL_START der Funktion CANx_MASTER_STATUS (→ Seite 125) gesetzt, dann wird gewartet, bis sich alle Slaves im Zustand 4 befinden. Anschließend werden alle Slaves mit dem NMT-Kommando [Start All Nodes] gestartet.
97	Ein Knoten wechselt in den Status 97, wenn er optional ist (Optionales Gerät in der CAN Konfiguration) und nicht auf die SDO-Anfrage nach dem Objekt 0x1000 reagiert hat. Wird der Slave zu einem späteren Zeitpunkt an das Netzwerk angeschlossen und erkannt, wird er automatisch gestartet. Dazu müssen Sie aber die Option [Automatisch starten] in den CAN-Parametern des Masters angewählt haben.
98	Ein Knoten wechselt in den Status 98, wenn der Gerätetyp (Objekt 0x1000) nicht dem konfiguriertem Typ entspricht.

Befindet sich der Slave im Status 4 oder höher, werden Nodeguard-Nachrichten an den Slave gesendet, wenn Nodeguarding konfiguriert wurde.

Nodeguarding-/Heartbeatfehler

Status	Beschreibung
99	Im Falle eines Nodeguarding-Timeouts wird die Variable NODE_STATE im Array NODE_STATE_SLAVE der Funktion CANx_MASTER_STATUS (→ Seite 125) auf 99 gesetzt. Sobald der Knoten wieder auf NodeGuard-Anfragen reagiert und die Option [Automatisch starten] eingeschaltet ist, wird er automatisch vom Master gestartet. Dabei wird der Knoten abhängig von seinem Status, der in der Antwort auf die Nodeguard-Anfragen enthalten ist, neu konfiguriert oder nur gestartet. Um den Slave manuell zu starten, genügt es, die Methode [NodeStart] zu benutzen.

Für Heartbeat-Fehler gilt das selbe Vorgehen.

Der aktuelle CANopen-Status eines Knotens kann über das Strukturelement LAST_STATE aus dem Array NODE_STATE_SLAVE der Funktion CANx_MASTER_STATUS (→ Seite 125) abgerufen werden.

Status	Beschreibung
0	Der Knoten befindet sich im Bootup-Zustand.
4	Der Knoten befindet sich im Status PREPARED.
5	Der Knoten befindet sich im Status OPERATIONAL.
127	Der Knoten befindet sich im Status PREOPERATIONAL.

8.7.3 Hochlauf des Netzwerks ohne [Automatisch starten]

Manchmal ist es notwendig, dass die Applikation den Zeitpunkt bestimmt, wann die CANopen-Slaves gestartet werden. Dazu müssen Sie die Option [Automatisch starten] des CAN-Masters in der Konfiguration deaktivieren. Dann ist die Applikation für das Starten der Slaves zuständig.

Um einen Slave über die Applikation zu starten, müssen Sie das Strukturelement START_NODE im Array NODE_STATE_SLAVES setzen.
Das Array wird per ADR-Operator an die Funktion CANx_MASTER_STATUS (→ Seite 125) übergeben.

Starten des Netzwerks mit GLOBAL_START

In einem CAN-Netz mit vielen Teilnehmern (meist mehr als 8) kommt es häufig dazu, dass schnell aufeinanderfolgende NMT-Nachrichten nicht von allen (meist langsamen) IO-Knoten (z.B. CompactModule CR2013) erkannt werden. Das liegt daran, dass diese Knoten alle Nachrichten mit dem ID 0 mithören müssen. In zu schneller Folge gesendete NMT-Nachrichten überlasten den Empfangspuffer solcher Knoten.

Eine Abhilfe können Sie schaffen, wenn die Anzahl schnell aufeinanderfolgender NMT-Nachrichten reduziert wird.

► Dazu von der Funktion CANx_MASTER_STATUS (→ Seite 125) den Eingang GLOBAL_START auf TRUE setzen (mit [Automatisch starten]).

> Die CANopen-Master-Bibliothek benutzt den Befehl "Start All Nodes", anstatt alle Knoten einzeln mit dem Kommando "Start Node" zu starten.

> GLOBAL_START wird nur einmalig bei der Netzwerk-Initialisierung ausgeführt.

> Wenn dieser Eingang gesetzt wird, startet die Steuerung auch Knoten mit dem Status 98 (siehe oben). Die PDOs für diese Nodes bleiben jedoch weiterhin deaktiviert.

Starten des Netzwerks mit START_ALL_NODES

Wird das Netzwerk nicht automatisch mit GLOBAL_START der Funktion CANx_MASTER_STATUS (→ Seite 125) gestartet, kann es jederzeit gestartet werden, d.h. jeder Knoten einzeln nacheinander. Ist das nicht gewünscht, besteht folgende Möglichkeit:

► Von der Funktion CANx_MASTER_STATUS den Funktionseingang START_ALL_NODES auf TRUE setzen.
START_ALL_NODES wird typisch zur Laufzeit durch das Applikations-Programm gesetzt.

> Wenn dieser Eingang gesetzt wird, werden auch Knoten mit dem Status 98 (siehe oben) gestartet. Die PDOs für diese Nodes bleiben jedoch weiterhin deaktiviert.

Initialisieren des Netzwerks mit RESET_ALL_NODES

Aus den selben Gründen, die für den Befehl START_ALL_NODES sprechen, gibt es Fälle, in denen Sie besser das NMT-Kommando RESET_ALL_NODES (anstelle RESET_NODES für jeden einzelnen Knoten) einsetzen.

► Dazu müssen Sie von der Funktion CANx_MASTER_STATUS (→ Seite 125) den Eingang RESET_ALL_NODES auf TRUE setzen.

> Dadurch werden einmalig alle Knoten gleichzeitig zurückgesetzt.

Zugriff auf den Status des CANopen-Masters

Damit der Applikations-Code erst abgearbeitet wird, wenn das IO-Netzwerk bereit ist, sollten Sie den Status des Masters abfragen. Das folgende Code-Fragment-Beispiel zeigt eine Möglichkeit:

Variablendeklaration

```
VAR
    FB_MasterStatus:= CR0020_MASTER_STATUS;
    :
END_VAR
```

Programmcode

```
If   FB_MasterStatus. NODE_STATE = 5 then
     <Applikationscode>
End_if
```

Durch Setzen des Flags TIME_OUT_STATE im Array NODE_STATE_SLAVE der Funktion CANx_MASTER_STATUS (→ Seite 125) kann die Applikation reagieren und zum Beispiel den nicht konfigurierbaren Knoten überspringen.

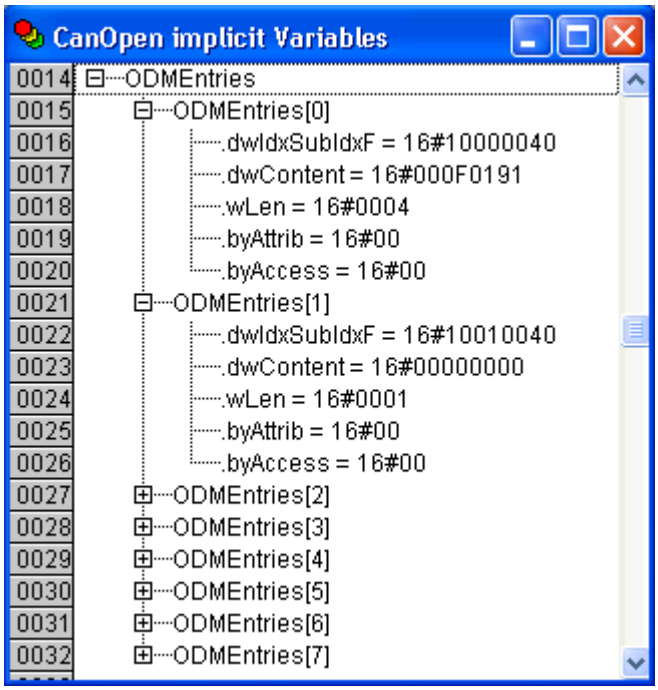
Das Objektverzeichnis des CANopen Masters

In manchen Fällen ist es hilfreich, wenn der CAN-Master über ein eigenes Objektverzeichnis verfügt. Das ermöglicht z.B. den Datenaustausch der Applikation mit anderen CAN-Knoten.

Das Objektverzeichnis des Masters wird über eine EDS-Datei mit dem Namen `CRnnnnMasterODEntry.EDS` während der Übersetzungszeit erstellt und mit Werten vorbelegt. Diese EDS-Datei ist im Verzeichnis `CoDeSys Vn\Library\PLCconf` abgelegt. Der Inhalt der EDS-Datei kann über die Schaltfläche [EDS...] im Konfigurations-Fenster [CAN-Parameter] angesehen werden.

Auch, wenn das Objektverzeichnis nicht vorhanden ist, kann der Master ohne Einschränkungen genutzt werden.

Der Zugriff auf das Objektverzeichnis durch die Applikation erfolgt über ein Array, das die folgende Struktur hat:



Strukturelement	Beschreibung
dwIdxSubIdxF	Die Struktur der Komponente 16#iiiissff ist: iiii - Index (2 Byte, Bit 16-31), Idx ss - Subindex (1 Byte, Bit 8-15), SubIdx ff - Flags (1 Byte, Bit 0-7), F Die Flag-Bits haben folgende Bedeutung: Bit 0 = Schreiben (Write) Bit 1 = Inhalt ist ein Zeiger auf eine Adresse (Content is pointer) Bit 2 = mapbar (mappable) Bit 3 = swap Bit 4 = Vorzeichen behafteter Wert (signed) Bit 5 = Fließkomma (float) Bit 6 = Weitere Subindizes enthalten (has more elements)
dwContent	Inhalt des Eintrags
wLen	Länge der Daten

Strukturelement	Beschreibung
byAttrib	Ursprünglich als Zugriffsberechtigung gedacht. Kann von der Applikation des Masters beliebig genutzt werden.
byAccess	Früher Zugriffsberechtigung. Kann von der Applikation des Masters beliebig genutzt werden.

An der Oberfläche verfügt CoDeSys® über keinen Editor für dieses Objektverzeichnis.

Die EDS-Datei gibt nur vor, mit welchen Objekten das Objektverzeichnis angelegt wird. Dabei werden die Einträge immer mit der Länge 4 erzeugt und die Flags (niederwertigstes Byte der Komponente eines Objektverzeichniseintrags dwIdxSubIdxF) immer mit 1 belegt. D.h. beide Bytes werden mit 16#41 belegt.

Wenn ein Objektverzeichnis im Master vorhanden ist, kann der Master als SDO-Server im Netz auftreten. Immer wenn ein Client auf einen Objektverzeichnis-Eintrag schreibend zugreift, wird das der Applikation über das Flag OD_CHANGED in der Funktion CANx_MASTER_STATUS (→ Seite 125) angezeigt. Nach der Auswertung müssen Sie dieses Flag wieder zurücksetzen.

Die Applikation kann das Objektverzeichnis nutzen, indem die Einträge direkt beschrieben oder gelesen werden, oder indem die Einträge auf IEC-Variablen zeigen. D.h.: beim Lesen/Schreiben eines anderen Knotens wird direkt auf diese IEC-Variablen zugegriffen.

Wenn Index und Subindex des Objektverzeichnisses bekannt sind, kann ein Eintrag wie folgt angesprochen werden:

```
I := GetODMEntryValue(16#iiiiss00, pCanOpenMaster[0].wODMFirstIdx,
pCanOpenMaster[0].wODMFirstIdx + pCanOpenMaster[0]. wODMCount;
```

Wobei für "iiii" der Index und für "ss" der Subindex (als Hex-Werte) eingesetzt werden müssen.

Damit steht die Nummer des Array-Eintrags in I zur Verfügung. Nun können Sie direkt auf die Komponenten des Eintrags zugreifen.
Damit Sie diesen Eintrag direkt auf einer IEC-Variable ausgeben können, genügt es, Adresse, Länge und Flags einzutragen:

```
ODMEntries[I].dwContent := ADR(<Variablenname>);
ODMEntries[I].wLen := sizeof(<Variablenname>);
ODMEntries[I]. dwIdxSubIdxF := ODMEntries[I]. dwIdxSubIdxF OR
OD_ENTRYFLG_WRITE OR OD_ENTRYFLG_ISPOINTER;
```

Um nur den Inhalt des Eintrags zu ändern, genügt es, den Inhalt von "dwContent" zu ändern.

8.7.4 CAN-Device

Inhalt:

CAN-Device ist ein anderer Name für CANopen-Slave oder CANopen-Node.

Eine CoDeSys®-programmierbare Steuerung kann in einem CAN-Netzwerk auch als CANopen-Slave erscheinen.

Funktionalität

Die CAN-Device-Bibliothek zusammen mit dem CANopen-Konfigurator stellt dem Anwender folgende Möglichkeiten zur Verfügung:

- In CoDeSys® Konfiguration der Eigenschaften NodeGuarding/Heartbeat, Emergency, Node-ID und Baudrate, auf der das Device arbeiten soll.
- Zusammen mit dem Parametermanager in CoDeSys® kann ein Default-PDO-Mapping erstellt werden, das zur Laufzeit vom Master geändert werden kann. Die Änderung des PDO-Mappings erfolgt während der Konfigurationsphase durch den Master. Durch das Mapping können IEC-Variablen der Applikation in PDOs gemappt werden. D.h. den PDOs werden IEC-Variable zugeordnet, um sie im Applikations-Programm einfach auswerten zu können.
- Die CAN-Device-Bibliothek stellt ein Objektverzeichnis zur Verfügung. Die Größe dieses Objektverzeichnisses wird zur Übersetzungszeit von CoDeSys® festgelegt. In diesem Verzeichnis befinden sich alle Objekte, die das CAN-Device beschreiben und zusätzlich die, die vom Parametermanager definiert sind. Im Parametermanager können zusammen mit dem CAN-Device nur die Listenarten Parameter und Variablen verwendet werden.
- Die Bibliothek verwaltet die Zugriffe auf das Objektverzeichnis, tritt also am Bus als SDO-Server auf.
- Die Bibliothek überwacht das Nodeguarding und die Heartbeat-Consumer-Zeit (immer nur von einem Producer) und setzt entsprechende Fehlerflags für die Applikation.
- Es kann eine EDS-Datei erzeugt werden, die die konfigurierten Eigenschaften des CAN-Device so beschreibt, dass das Device als Slave unter einem CAN-Master eingebunden und konfiguriert werden kann.

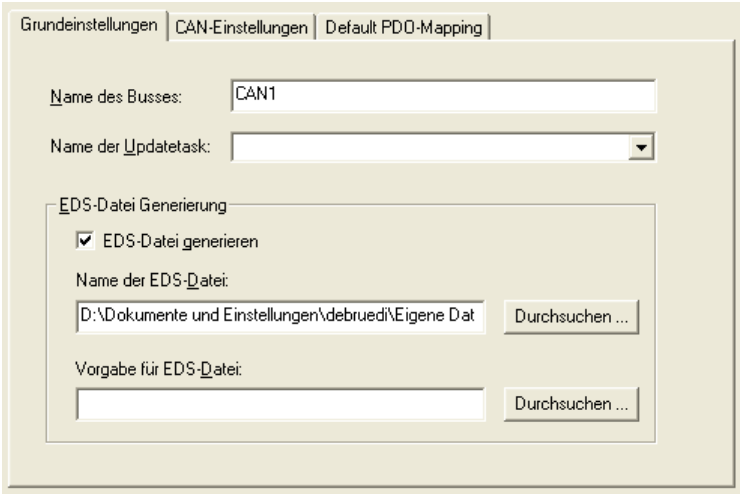
Die CAN-Device Bibliothek stellt ausdrücklich folgende, in CANopen beschriebene, Funktionalitäten nicht zur Verfügung (alle hier und im obigen Abschnitt nicht genannten Möglichkeiten des CANopen-Protokolls sind ebenfalls nicht implementiert):

- Dynamische SDO- und PDO-Identifier
- SDO Block-Transfer
- Automatische Erzeugung von Emergency-Nachrichten. Emergency-Nachrichten müssen immer durch die Funktion CANx_SLAVE_EMCY_HANDLER (→ Seite 134) und die Funktion CANx_SLAVE_SEND_EMERGENCY (→ Seite 136) von der Applikation erzeugt werden. Die Bibliothek `ifm_CRnnnn_CANopenSlave_Vxxyyzz.LIB` stellt Ihnen dazu diese Funktionen zur Verfügung.
- Dynamische Änderungen der PDO-Eigenschaften werden z.Z. immer nur beim Eintreffen einer StartNode NMT-Nachricht übernommen, nicht mit den in CANopen definierten Mechanismen.

CAN-Device konfigurieren

Um die Steuerung als CANopen-Slave (Device) zu nutzen, muss zunächst in der Steuerungskonfiguration über [Einfügen] > [Unterelement anhängen] der CANopen-Slave eingefügt werden. Bei Steuerungen mit 2 oder mehr CAN-Schnittstellen wird automatisch CAN-Schnittstelle 1 als Slave konfiguriert. Alle notwendigen Bibliotheken werden automatisch in den Bibliotheksverwalter eingefügt.

Register [Grundeinstellungen]



Name des Busses
wird im Moment nicht benutzt.

Name der Updatetask
Name der Task, in der der Aufruf des CAN-Device erfolgt.

EDS-Datei generieren
Soll aus den Einstellungen hier eine EDS-Datei erzeugt werden, um das CAN-Device in eine beliebigen Masterkonfiguration einfügen zu können, muss hier die Option [EDS-Datei generieren] aktiviert werden und der Name einer Datei angegeben werden. Optional kann auch noch eine Vorlagendatei angeben werden, deren Einträge zum EDS-File des CAN-Device hinzugefügt werden. Bei Überschneidungen werden Vorgaben der Vorlage nicht überschrieben.

Beispiel für ein Objektverzeichnis

Folgende Einträge könnten zum Beispiel im Objektverzeichnis stehen:

```
[FileInfo]
FileName=D:\CoDeSys\lib2\plcconf\MyTest.eds
FileVersion=1
FileRevision=1
Description=EDS for CoDeSys-Project:
D:\CoDeSys\CANopenTestprojekte\TestHeartbeatODsettings_Device.pro
CreationTime=13:59
CreationDate=09-07-2005
CreatedBy=CoDeSys
ModificationTime=13:59
ModificationDate=09-07-2005
ModifiedBy=CoDeSys

[DeviceInfo]
VendorName=3S Smart Software Solutions GmbH
```

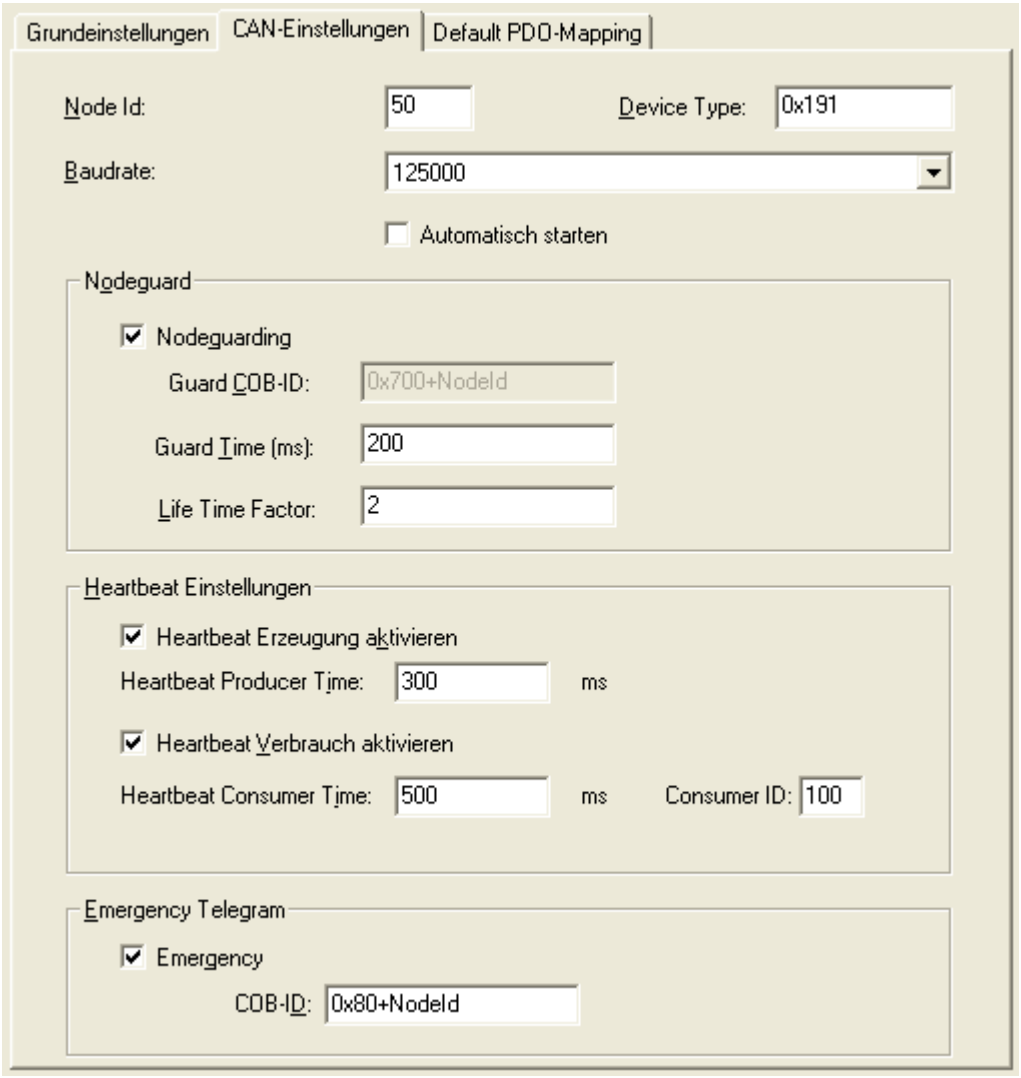
```
ProductName=TestHeartbeatODsettings_Device
ProductNumber=0x33535F44
ProductVersion=1
ProductRevision=1
OrderCode=xxxx.yyyy.zzzz
LMT_ManufacturerName=3S GmbH
LMT_ProductName=3S_Dev
BaudRate_10=1
BaudRate_20=1
BaudRate_50=1
BaudRate_100=1
BaudRate_125=1
BaudRate_250=1
BaudRate_500=1
BaudRate_800=1
BaudRate_1000=1
SimpleBootUpMaster=1
SimpleBootUpSlave=0
ExtendedBootUpMaster=1
ExtendedBootUpSlave=0

...

[1018sub0]
ParameterName=Number of entries
ObjectType=0x7
DataType=0x5
AccessType=ro
DefaultValue=2
PDOMapping=0

[1018sub1]
ParameterName=VendorID
ObjectType=0x7
DataType=0x7
AccessType=ro
DefaultValue=0x0
PDOMapping=0

[1018sub2]
ParameterName=Product Code
ObjectType=0x7
DataType=0x7
AccessType=ro
DefaultValue=0x0
PDOMapping=0
```

Bedeutung der einzelnen Objekte entnehmen Sie bitte der CANopen-Spezifikation DS301.

Die EDS-Datei enthält, neben den vorgeschriebenen Einträgen, die Definitionen für SYNC, Guarding, Emergency und Heartbeat. Wenn diese Objekte nicht benutzt werden, sind die Werte auf 0 gesetzt (voreingestellt). Da die Objekte aber im Objektverzeichnis des Slaves zur Laufzeit vorhanden sind, werden sie in der EDS-Datei auch beschrieben.

Das Gleiche gilt für die Einträge für die Kommunikations- und Mapping-Parameter. Es sind immer alle 8 möglichen Subindizes der Mapping-Objekte 0x16xx oder 0x1Axx vorhanden, aber u.U. im Subindex 0 nicht berücksichtigt. **HINWEIS:** Bit-Mapping wird von der Bibliothek nicht unterstützt!

Register [CAN-Einstellungen]

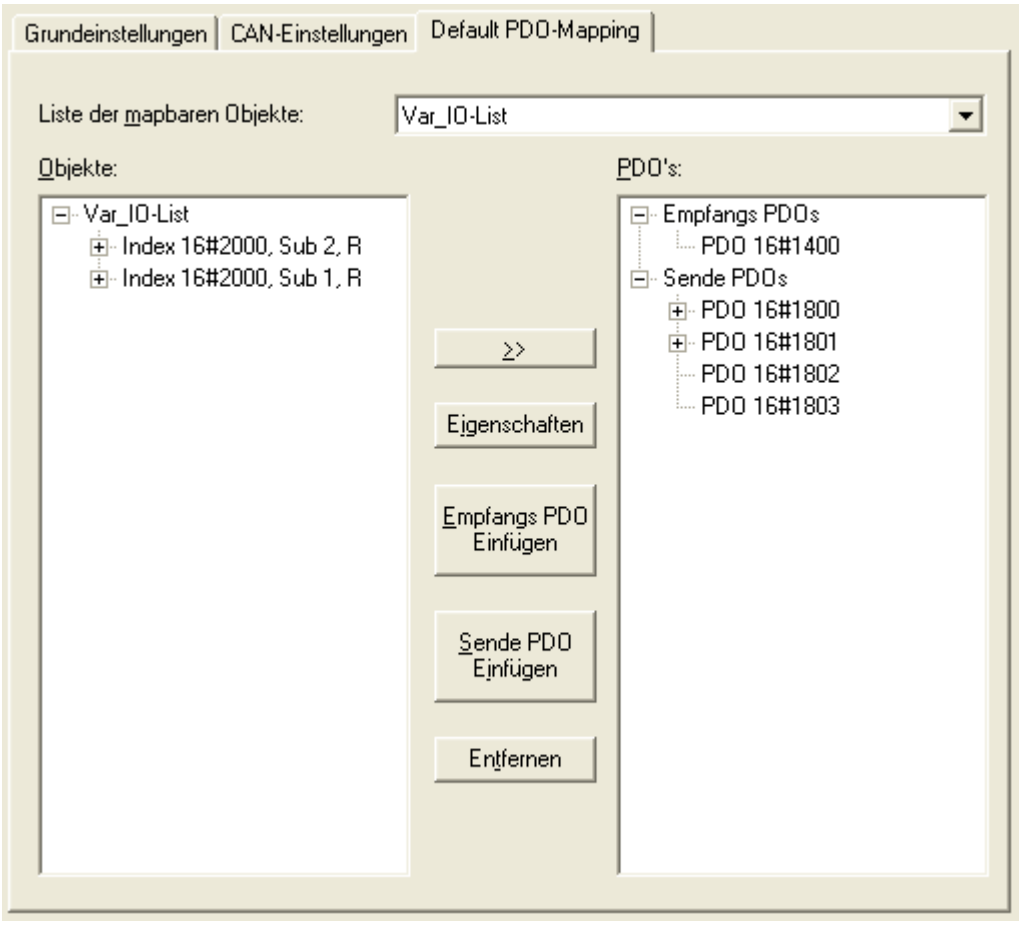


Hier können Sie die **Node-ID** und die **Baudrate** einstellen.

Device Type
(das ist der Default-Wert des Objekts 0x1000, der im EDS eingetragen wird) wird mit 0x191 (Standard-IO-Device) vorbelegt und kann von Ihnen beliebig geändert werden.
Der Index des CAN-Controllers ergibt sich aus der Position des CAN-Device in der Steuerungskonfiguration.

Die **Nodeguarding**-Parameter, die **Heartbeat**-Parameter und die Emergency-COB-ID können Sie ebenfalls auf diesem Register festlegen. Das CAN-Device kann nur für die Überwachung eines Heartbeats konfiguriert werden.
Wir empfehlen: Für aktuelle Geräte besser mit Heartbeat arbeiten, weil dann die Buslast niedriger ist.

Register [Default PDO-Mapping]

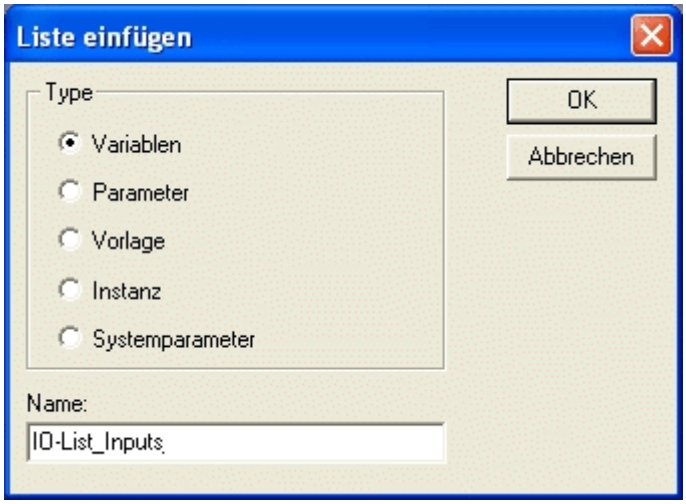


In diesem Register können Sie die Zuordnung zwischen lokalem Objektverzeichnis (OD-Editor) und den PDOs festlegen, die vom CAN-Device gesendet/empfangen werden. Eine solche Zuordnung wird als "Mapping" bezeichnet.

In den verwendeten Objektverzeichniseinträgen (Variablen OD) wird zwischen Objektindex/Subindex die Verbindung zu Variablen der Applikation hergestellt. Dabei müssen Sie nur darauf achten, dass der Subindex 0 eines Indexes, der mehr als einen Subindex enthält, die Information über die Anzahl der Subindizes enthält.

Beispiel Variablenliste

Auf dem ersten Empfangs-PDO (COB-ID = 512 + Node-ID) des CAN-Device sollen die Daten für die Variable PLC_PRG.a empfangen werden.



Info

Als Listentyp kann [Variablen] oder [Parameter] gewählt werden.

Zum Datenaustausch (z.B. über PDOs oder sonstige Einträge im Objektverzeichnis) wird eine Variablenliste angelegt.

Die Parameterliste sollten Sie einsetzen, wenn Sie Objektverzeichniseinträge nicht mit Applikations-Variablen verknüpfen wollen. Für die Parameterliste ist zur Zeit nur der Index 16#1006 / SubIdx 0 vordefiniert. In diesen Eintrag kann vom Master der Wert für die [Com. Cycle Period] eingetragen werden. Damit wird das Ausbleiben der SYNC-Nachricht gemeldet.

Also müssen Sie im Objektverzeichnis (Parametermanager) eine Variablenliste anlegen und einen Index/SubIndex mit der Variablen PLC_PRG.a verknüpfen:

- ► Dazu fügen Sie in der Variablenliste eine Zeile hinzu (rechte Maustaste öffnet das Kontextmenü) und tragen einen Variablen-Namen (beliebig) sowie den Index und den Subindex ein.
- ► Als Zugriffsrichtung ist für ein Empfangs-PDO nur [write only] (schreiben) zugelassen.
- ► In die Spalte [Variable] tragen Sie dann "`PLC_PRG.a`" ein, oder drücken [F2] und wählen die Variable aus.

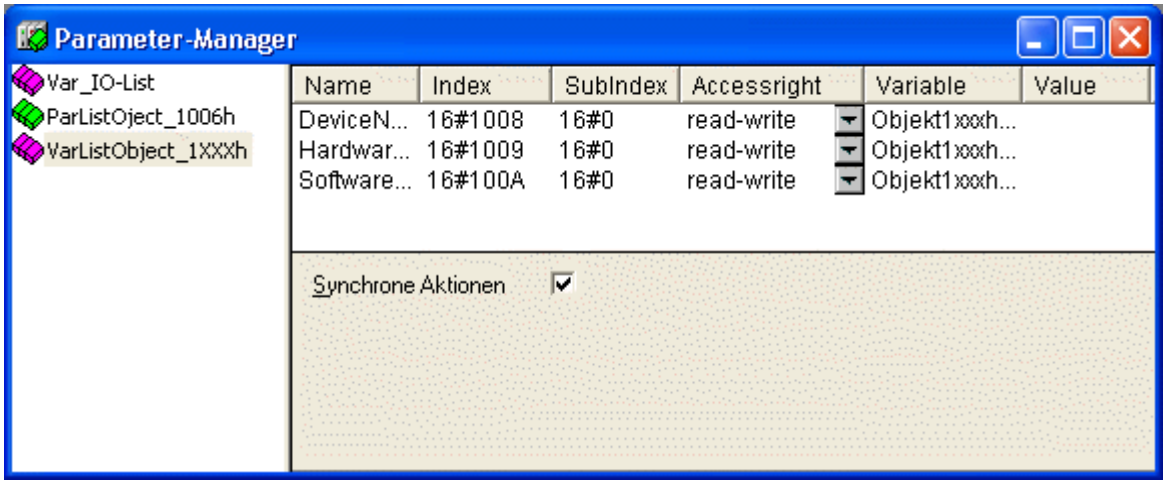
HINWEIS

Daten, die vom CAN-Master gelesen werden sollen (z.B. Eingänge, Systemvariablem), müssen die Zugriffsrichtung (Accessright) [read only] (lesen) haben.

Daten, die vom CAN-Master geschrieben werden sollen (z.B. Ausgänge im Slave), müssen die Zugriffsrichtung (Accessright) [write only] (schreiben) haben.

SDO-Parameter, die vom CAN-Master geschrieben und gleichzeitig aus der Slave-Applikation gelesen und geschrieben werden sollen, müssen die Zugriffsrichtung (Accessright) [read-write] (lesen+schreiben) haben.

Damit Sie den Parametermanager öffnen können, muss in den Zielsystemeinstellungen unter [Netzfunktionen] der Parametermanager aktiviert sein. Die Bereiche für Index/Subindex sind bereits mit sinnvollen Werten vorbelegt und sollten nicht geändert werden.



Im Default PDO-Mapping des CAN-Device wird anschließend der Index-/Subindex-Eintrag als Mapping-Eintrag einem Empfangs-PDO zugewiesen. Die Eigenschaften des PDOs lassen sich über den Dialog festlegen, der aus CANopen-Slaves einfügen und konfigurieren (→ Seite 91) bekannt ist.

Nur Objekte aus dem Parametermanager, die mit dem Attribut [read only] (lesen) oder [write only] (schreiben) versehen sind, werden in der evtl. erzeugten EDS-Datei als mapbar (= zuordnungsfähig) markiert und tauchen in der Liste der mapbaren Objekte auf. Alle anderen Objekte werden in der EDS-Datei als nicht mapbar markiert.

HINWEIS

Werden mehr als 8 Datenbytes in ein PDO gemappt, werden automatisch die nächsten freien Identifier dafür genutzt, bis alle Datenbytes übertragen werden können.

Um eine klare Struktur der verwendeten Identifier zu erhalten, sollten Sie die richtige Zahl der Empfangs- und Sende-PDOs einfügen und diesen die Variablen-Bytes aus der Liste zuordnen.

Verändern des Standard-Mappings durch Master-Konfiguration

Sie können das vorgegebene PDO-Mapping (in der CAN-Device-Konfiguration) in bestimmten Grenzen durch den Master verändern.

Dabei gilt die Regel, dass das CAN-Device nicht in der Lage ist, Objektverzeichniseinträge neu anzulegen, die nicht bereits im Standard-Mapping (Default PDO-Mapping in der CAN-Device-Konfiguration) vorhanden sind. Also kann z.B. für ein PDO, das im Default PDO-Mapping ein gemapptes Objekt enthält, in der Masterkonfiguration kein zweites Objekt gemappt werden.

Das durch die Masterkonfiguration veränderte Mapping kann also höchstens die im Standard-Mapping vorhandenen PDOs enthalten. Innerhalb dieser PDOs sind 8 Mapping-Einträge (Subindizes) vorhanden.

Eventuelle Fehler, die hierbei auftreten können, werden Ihnen nicht angezeigt, d.h. die überzähligen PDO-Definitionen / die überzähligen Mapping-Einträge werden so behandelt, als seien sie nicht vorhanden.

Die PDOs müssen im Master immer von 16#1400 (Empfangs-PDO-Kommunikationsparameter) oder 16#1800 (Sende-PDO-Kommunikationsparameter) beginnend angelegt sein und lückenlos aufeinander folgen.

Zugriff auf das CAN-Device zur Laufzeit

Einstellen der Knotennummer und der Baud-Rate eines CAN-Device

Beim CAN Device kann zur Laufzeit des Applikations-Programms die Knotennummer und die Baudrate eingestellt werden.

- Zum Einstellen der **Knotennummer** wird die Funktion CANx_SLAVE_NODEID (→ Seite 133) der Bibliothek `ifm_CRnnnn_CANopenSlave_Vxxyyzz.lib` genutzt.
- Zum Einstellen der **Baud-Rate** wird bei den Controllern und beim PDM360 smart die Funktion CAN1_BAUDRATE (→ Seite 60) oder die Funktion CAN1_EXT (→ Seite 65) oder die Funktion CANx der jeweiligen Gerätebibliothek benutzt. Beim PDM360 oder PDM360 compact steht hierfür die Funktion CANx_SLAVE_BAUDRATE über die Bibliothek `ifm_CRnnnn_CANopenSlave_Vxxyyzz.lib` zur Verfügung.

Zugriff auf die OD-Einträge vom Applikations-Programm

Standardmäßig gibt es Objektverzeichniseinträge, die auf Variablen gemappt sind (Parametermanager).

Es gibt jedoch auch die automatisch erzeugten Einträge des CAN-Device, auf die Sie nicht über den Parametermanager in einen Variableninhalt mappen können. Diese Einträge stehen über die Funktion CANx_SLAVE_STATUS (→ Seite 139) in der Bibliothek `ifm_CRnnnn_CANopenSlave_Vxxyyzz.LIB` zur Verfügung.

Ändern der PDO-Eigenschaften zur Laufzeit

Sollen die Eigenschaften eines PDOs zur Laufzeit verändert werden, so funktioniert das durch einen anderen Knoten über SDO-Schreibzugriffe, wie dies von CANopen beschrieben wird.

Alternativ kann man auch direkt eine neue Eigenschaft, wie z.B. die "Event time" eines Sende-PDOs schreiben und anschließend einen Befehl "StartNode-NMT" an den Knoten schicken, obwohl er bereits gestartet ist. Das führt dazu, dass das Device die Werte im Objektverzeichnis neu interpretiert.

Emergency-Messages durch das Applikations-Programm senden

Um eine Emergency-Message durch das Applikations-Programm zu versenden, können Sie die Funktion CANx_SLAVE_EMCY_HANDLER (→ Seite 134) und die Funktion CANx_SLAVE_SEND_EMERGENCY (→ Seite 136) einsetzen. Die Bibliothek `ifm_CRnnnn_CANopenSlave_Vxxyyzz.LIB` stellt ihnen dazu diese Funktionen zur Verfügung.

8.7.5 CAN-Netzwerkvariablen

Allgemeine Informationen

Netzwerkvariablen
CAN Netzwerkvariablen sind eine Möglichkeit, Daten zwischen zwei oder mehreren Steuerungen auszutauschen. Der Mechanismus sollte dabei für den Anwender möglichst einfach zu handhaben sein. Derzeit sind Netzwerkvariablen auf Basis von CAN und UDP implementiert. Die Variablenwerte werden dabei auf der Basis von Broadcast-Nachrichten automatisch ausgetauscht. In UDP sind diese als Broadcast-Telegramme realisiert, in CAN als PDOs. Diese Dienste sind vom Protokoll her nicht bestätigte Dienste, d.h. es gibt keine Kontrolle, ob die Nachricht auch beim Empfänger ankommt. Netzwerkvariablen-Austausch entspricht einer "1-zu-n-Verbindung" (1 Sender zu n Empfängern).

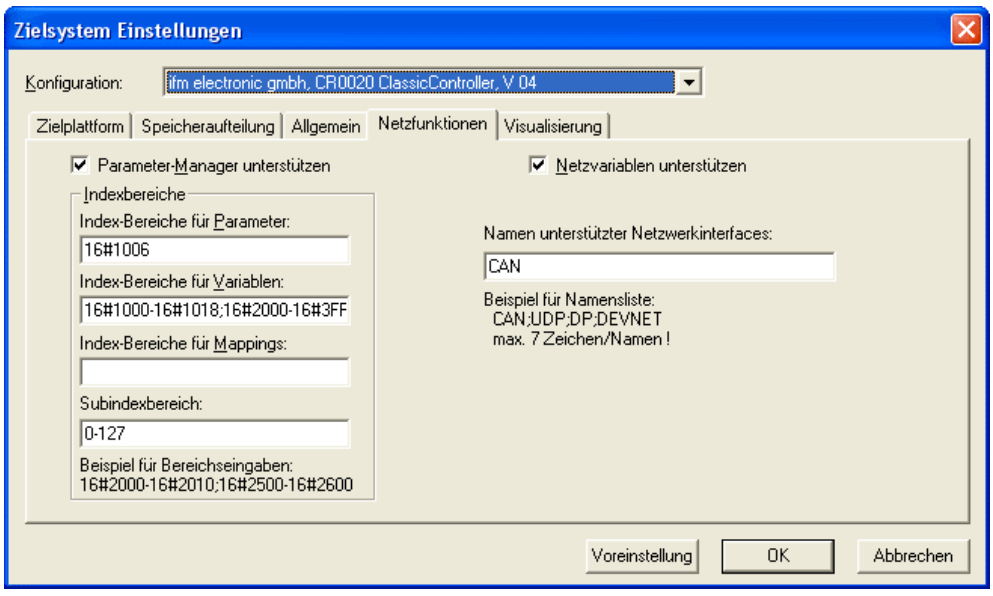
Objektverzeichnis
Das Objektverzeichnis ist eine weitere Möglichkeit, Variablen auszutauschen. Dabei handelt es sich um eine 1-zu-1-Verbindung, die ein bestätigtes Protokoll verwendet. Hier kann der Anwender also kontrollieren, ob die Nachricht den Empfänger erreichte. Der Austausch erfolgt nicht automatisch, sondern über den Aufruf von Funktionsblöcken aus dem Applikations-Programm.
→ Kapitel Das Objektverzeichnis des CANopen-Masters, Seite 100

CAN-Netzwerkvariablen konfigurieren

Um die Netzwerkvariablen mit CoDeSys® zu nutzen, benötigen Sie die Bibliotheken `3s_CanDrv.lib`, `3S_CANopenManager.lib` und `3S_CANopenNetVar.lib`. Außerdem benötigen Sie die Bibliothek `SysLibCallback.lib`.

CoDeSys® erzeugt automatisch den nötigen Initialisierungscode sowie den Aufruf der Netzwerk-Bausteine am Zyklusanfang und -ende.

Einstellungen in den Zielsystemeinstellungen



- ► Dialogbox [Zielsystemeinstellungen] wählen
- ► Register [Netzfunktionen] wählen
- ► Aktivieren Sie das Kontrollkästchen [Netzvariablen unterstützen].
- ► Bei [Namen unterstützter Netzwerkinterfaces] geben Sie den Namen des gewünschten Netzwerks an, hier CAN.

► Um Netzwerkvariablen zu nutzen, müssen Sie außerdem einen CAN-Master oder CAN-Slave (Device) in der Steuerungskonfiguration einfügen.

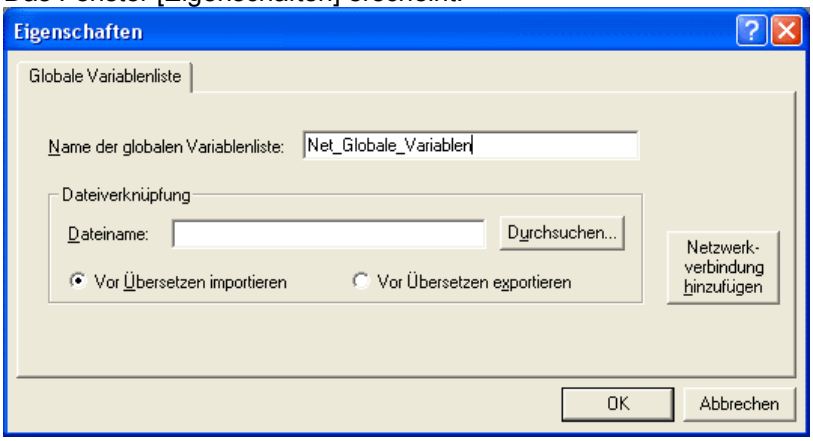
► Bitte beachten Sie die Besonderheiten bei der Anwendung von Netzwerkvariablen für die jeweiligen Gerätetypen
→ Kapitel Besonderheiten bei Netzwerkvariablen, Seite 114

Einstellungen in den globalen Variablenlisten

► Legen Sie eine neue globale Variablenliste an. Hier definieren Sie die Variablen, die sie mit anderen Steuerungen austauschen wollen.

► Öffnen Sie den Dialog mit dem Kontextmenü [Objekt Eigenschaften...].

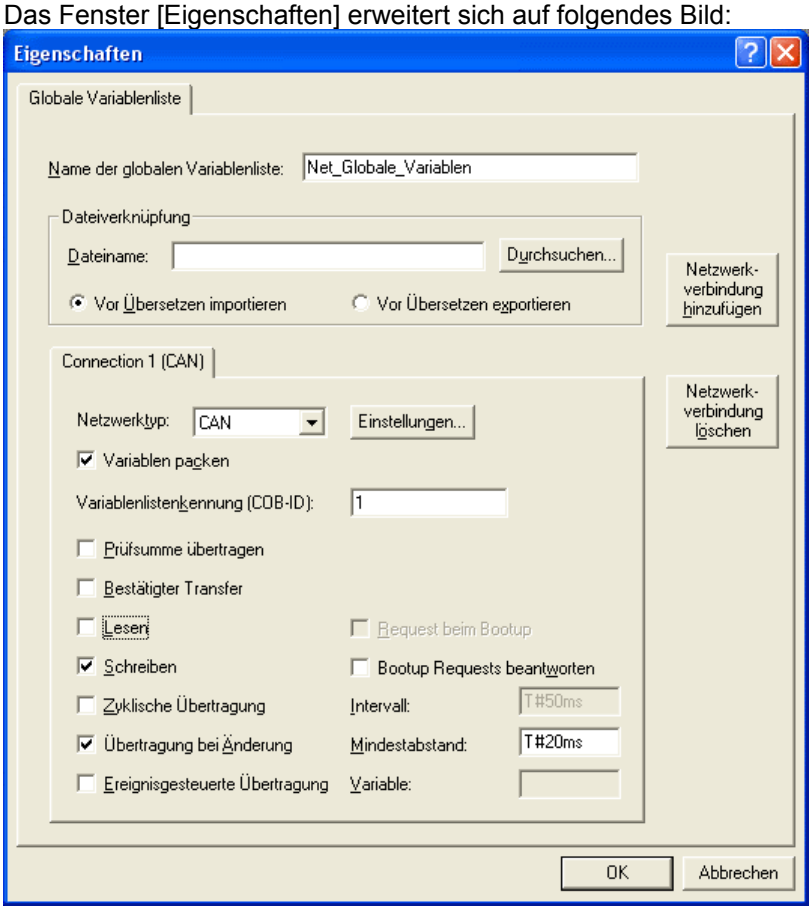
> Das Fenster [Eigenschaften] erscheint:



Wenn Sie die Netzwerkeigenschaften dieser Variablenliste definieren wollen:

► Schaltfläche [Netzwerkverbindung hinzufügen] klicken.
Wenn Sie mehrere Netzwerkverbindungen konfiguriert haben, können Sie hier auch pro Variablenliste mehrere Verbindungen konfigurieren.

> Das Fenster [Eigenschaften] erweitert sich auf folgendes Bild:



Die Optionen haben dabei folgende Bedeutungen:

Netzwerktyp
Als Netzwerktyp können Sie einen der bei den Zielsystemeinstellungen angegebenen Netzwerknamen angeben.
Wenn Sie daneben auf die Schaltfläche [Einstellungen] klicken, können Sie die CAN-Schnittstelle wählen:

1. CAN-Schnittstelle: Wert = 0
2. CAN-Schnittstelle: Wert = 1

usw.

Variablen packen
Wenn diese Option mit [**v**] aktiviert ist, werden die Variablen nach Möglichkeit in einer Übertragungseinheit zusammengefasst. Bei CAN ist eine Übertragungseinheit 8 Bytes groß. Passen nicht alle Variablen der Liste in eine Übertragungseinheit, dann werden für diese Liste automatisch mehrere Übertragungseinheiten gebildet.

Ist die Option nicht aktiviert, kommt jede Variable in eine eigene Übertragungseinheit.

Wenn [Übertragung bei Änderung] konfiguriert ist, wird für jede Übertragungseinheit getrennt geprüft, ob sie geändert ist und gesendet werden muss.

Variablenlistenkennung (COB-ID)
Der Basis-Identifier wird als eindeutige Kennung benutzt, um Variablenlisten verschiedener Projekte auszutauschen. Variablenlisten mit gleichem Basis-Identifier werden ausgetauscht. Es ist darauf zu achten, dass die Definitionen der Variablenlisten mit gleichem Basis-Identifier in den verschiedenen Projekten übereinstimmen.

HINWEIS

Der Basis-Identifier wird in CAN-Netzwerken direkt als COB-ID der CAN-Nachrichten benutzt. Es gibt keine Überprüfung, ob der Identifier auch in der übrigen CAN-Konfiguration benutzt wird.

Damit die Daten korrekt zwischen zwei Steuerungen ausgetauscht werden, müssen die globalen Variablenlisten in den beiden Projekten übereinstimmen. Sie können das Feature [Dateiverknüpfung] benutzen, um dies sicherzustellen. Ein Projekt kann die Variablenlisten-Datei vor dem Übersetzen exportieren. Die anderen Projekte sollten diese Datei vor dem Übersetzen importieren.

Neben einfachen Datentypen kann eine Variablenliste auch Strukturen und Arrays enthalten. Die Elemente dieser zusammengesetzten Datentypen werden einzeln versendet.

Es dürfen keine Strings über Netzwerkvariablen verschickt werden, da es sonst zu einem Laufzeitfehler kommt und der Watchdog aktiviert wird.

Wenn eine Variablenliste größer ist als ein PDO des entsprechenden Netzwerks, dann werden die Daten auf mehrere PDOs aufgeteilt. Es kann darum nicht zugesichert werden, dass alle Daten der Variablenliste in einem Zyklus empfangen werden. Teile der Variablenliste können in verschiedenen Zyklen empfangen werden. Dies ist auch für Variablen mit Struktur- und Array-Typen möglich.

Prüfsumme übertragen
Diese Option wird nicht unterstützt.

Bestätigter Transfer
Diese Option wird nicht unterstützt.

Lesen
Es werden die Variablenwerte von einer (oder mehreren) Steuerungen gelesen.

Schreiben
Die Variablen dieser Liste werden zu anderen Steuerungen gesendet.

HINWEIS

Sie sollten für jede Variablenliste nur eine dieser Möglichkeiten auswählen, also entweder nur lesen oder nur schreiben.

Wollen Sie verschiedene Variablen eines Projekts lesen und schreiben, so verwenden Sie bitte mehrere Variablenlisten (eine zum Lesen, eine zum Schreiben).

Für die Kommunikation zwischen 2 Teilnehmern sollten Sie die Variablenliste von einer Steuerung auf die andere kopieren, um die gleiche Datenstruktur zu erhalten.

Zwecks besserer Übersichtlichkeit sollten Ihre Variablenlisten jeweils nur für ein Teilnehmerpaar gelten. Es ist nicht sinnvoll, die selbe Liste für alle Teilnehmer zu verwenden.

Zyklische Übertragung
Nur gültig, wenn [Schreiben] aktiviert. Die Werte werden in angegebenen [Intervall] gesendet, unabhängig davon, ob sie sich geändert haben.

Übertragung bei Änderung
Die Variablenwerte werden nur gesendet, wenn sich einer der Werte geändert hat. Mit [Mindestabstand] (Wert > 0) kann eine Mindestzeit zwischen den Nachrichtenpaketen festgelegt werden.

Ereignisgesteuerte Übertragung
Wenn diese Option gewählt ist, wird die CAN-Nachricht nur dann übertragen, wenn die angegebene binäre [Variable] auf TRUE gesetzt wird. Diese Variable kann nicht über die Eingabehilfe aus der Liste der definierten Variablen gewählt werden.

Besonderheiten bei Netzwerkvariablen

Gerät	Beschreibung
ClassicController CR0020, CR0505, CR0200 SafetyController CR7020, CR7021, CR7505, CR7506, CR7200, CR7201	Netzwerkvariablen werden nur auf CAN-Schnittstelle 1 (Wert = 0 eintragen) unterstützt. **CAN-Master** Sende- und Empfangslisten werden direkt verarbeitet. Sie brauchen nur die oben beschriebenen Einstellungen vornehmen. **CAN-Device** Sendelisten werden direkt verarbeitet. Für Empfangslisten müssen Sie zusätzlich noch den Bereich der Identifier im Objektverzeichnis auf Empfangs-PDOs mappen. Es ist ausreichend, wenn Sie nur zwei Empfangs-PDOs anlegen und dem ersten Objekt den ersten Identifier und dem zweiten Objekt den letzen Identifier zuweisen. Werden die Netzwerkvariablen nur auf einem Identifier übertragen, müssen Sie nur ein Empfangs-PDO mit diesem Identifier anlegen. **Wichtig!** Bitte beachten Sie, dass die Identifier der Netzwerkvariablen und der Empfangs-PDOs als **dezimale** Werte eingegeben werden müssen.
ClassicController CRnn32	Netzwerkvariablen werden auf allen CAN-Schnittstellen unterstützt. (Alle anderen Angaben wie oben)
PDM360 smart CR107x	Es steht nur eine CAN-Schnittstelle zur Verfügung (Wert = 0 eintragen). **CAN-Master** Sende- und Empfangslisten werden direkt verarbeitet. Sie brauchen nur die oben beschriebenen Einstellungen vornehmen. **CAN-Device** Sendelisten werden direkt verarbeitet. Für Empfangslisten müssen Sie zusätzlich noch den Bereich der Identifier im Objektverzeichnis auf Empfangs-PDOs mappen. Es ist ausreichend, wenn Sie nur zwei Empfangs-PDOs anlegen und dem ersten Objekt den ersten Identifier und dem zweiten Objekt den letzen Identifier zuweisen. Werden die Netzwerkvariablen nur auf einem Identifier übertragen, müssen Sie nur ein Empfangs-PDO mit diesem Identifier anlegen. **Wichtig!** Bitte beachten Sie, dass die Identifier der Netzwerkvariablen und der Empfangs-PDOs als **dezimale** Werte eingegeben werden müssen.

Gerät	Beschreibung
PDM360, PDM360 compact CR105x, CR106x	Netzwerkvariablen werden auf den CAN-Schnittstelle 1 (Wert = 0) und 2 (Wert = 1) unterstützt. **CAN-Master** Sende- und Empfangslisten werden direkt verarbeitet. Sie brauchen nur die oben beschriebenen Einstellungen vornehmen. **CAN-Device** Sende- und Empfangslisten werden direkt verarbeitet. Sie brauchen nur die oben beschriebenen Einstellungen vornehmen. **Wichtig!** Wird [Netzvariablen unterstützen] im PDM360 oder PDM360 compact angewählt, müssen Sie mindestens eine Variable in der Globalen Varableniste anlegen und diese einmalig im Applikations-Programm aufgerufen. Andernfalls wird die folgende Fehlermeldung bei der Programmübersetzung generiert: Fehler 4601: Netzwerkvariablen 'CAN' : Es ist keine zyklische oder freilaufende Task zum Netwerkvariablenaustausch vorhanden.

8.7.6 Informationen zur EMCY- und Error-Codes

Aufbau einer EMCY-Nachricht

Die Signalisierung von Fehlerzuständen erfolgt unter CANopen über einen sehr einfachen, standardisierten Mechanismus. Jedes Auftreten eines Fehlers bei einem CANopen-Gerät wird über eine spezielle Nachricht signalisiert, die den Fehler genauer beschreibt.

Verschwindet ein Fehler oder seine Ursache nach einer bestimmten Zeit wieder, wird dieses Ereignis ebenfalls über die EMCY-Nachricht signalisiert. Die zuletzt aufgetretenen Fehler werden im Objektverzeichnis (Objekt 1003h) abgelegt und können über einen SDO-Zugriff ausgelesen werden (→ Funktion CANx_SDO_READ, Seite 142). Zusätzlich spiegelt sich die aktuelle Fehlersituation im Error-Register (Objekt 1001 h) wider.

Man unterscheidet folgende Fehler:

a) Kommunikationsfehler

- Der CAN-Controller signalisiert CAN-Fehler.
  (Das gehäufte Auftreten ist ein Indiz für physikalische Probleme. Diese Fehler können einen erheblichen Einfluss auf das Übertragungsverhalten und damit auf den Datendurchsatz eines Netzwerks haben.)
- Life-Guarding- oder Heartbeat-Fehler

b) Anwendungsfehler

- Kurzschluss oder Leiterbruch
- Temperatur zu hoch

Aufbau einer Fehlernachricht

Eine Fehlernachricht (EMCY Message) hat folgenden Aufbau:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
EMCY-Fehlercode, wie im Objekt 1003h eingetragen		Objekt 1001h	Herstellerspezifische Informationen				

Identifier

Der Identifier für die Fehlernachricht besteht aus der Summe folgender Elemente:

EMCY-Default-Identifier 128 (80h)
+
Node-ID

EMCY-Fehlercode

Er gibt detailliert Auskunft darüber, welcher Fehler aufgetreten ist. Eine Liste möglicher Fehlercodes ist bereits im Kommunikationsprofil definiert. Fehlercodes, die nur für eine bestimmte Geräteklasse gültig sind, werden im jeweiligen Geräteprofil dieser Geräteklasse festgelegt.

Objekt 1003h (Error Field)

Das Objekt 1003h stellt den Fehlerspeicher eines Gerätes dar. Die Subindizes enthalten die zuletzt aufgetretenen Fehler, die ein Fehler-Telegramm ausgelöst haben.

Tritt ein neuer Fehler auf, dann wird sein EMCY-Fehlercode immer im Subindex 1h gespeichert. Alle anderen, älteren Fehler werden im Fehlerspeicher einen Platz nach hinten geschoben, also der Subindex um 1 erhöht. Falls alle unterstützten Subindizes belegt sind, wird der älteste Fehler gelöscht. Der Subindex 0h wird auf die Anzahl der gespeicherten Fehler erhöht. Nachdem alle Fehler behoben sind, wird in das Fehlerfeld des Subindex 1h der Wert "0" geschrieben.

Um den Fehlerspeicher zu löschen, kann der Subindex 0h mit dem Wert "0" beschrieben werden. Andere Werte dürfen nicht eingetragen werden.

Gerätefehler signalisieren

Wie beschrieben, werden EMCY-Nachrichten versendet, wenn Fehler in einem Gerät auftreten. Im Unterschied zu frei programmierbaren Geräten, werden beispielsweise von dezentralen Ein-/Ausgangsmodulen (z.B. CompactModule CR2033) Fehlermeldungen automatisch verschickt. Entsprechende Fehler-Codes → jeweiliges Gerätehandbuch.

Die programmierbaren Geräte erzeugen nur dann automatisch eine EMCY-Nachricht (z.B. Kurzschluss an einem Ausgang), wenn die Funktion CANx_MASTER_EMCY_HANDLER (→ Seite 120) oder die Funktion CANx_SLAVE_EMCY_HANDLER (→ Seite 134) in das Applikations-Programm eingebunden wird.

Übersicht der automatisch verschickten EMCY-Fehlercodes für alle mit CoDeSys® programmierbaren ifm-Geräte → Kapitel Übersicht der CANopen-Error-Codes, Seite 117.

Sollen zusätzlich noch applikations-spezifische Fehler durch das Applikations-Programm verschickt werden, werden die Funktion CANx_MASTER_SEND_EMERGENCY (→ Seite 122) oder die Funktion CANx_SLAVE_SEND_EMERGENCY (→ Seite 136) eingesetzt.

Übersicht CANopen Error-Codes

Error Code (hex)	Meaning / Bedeutung
00xx	Reset or no Error (Fehler rücksetzen/kein Fehler)
10xx	Generic Error (allgemeiner Fehler)
20xx	Current (Stromfehler)
21xx	Current, device input side (Stromfehler, eingangsseitig)
22xx	Current inside the device (Stromfehler im Geräteinnern)
23xx	Current, device output side (Stromfehler, ausgangsseitig)
30xx	Voltage (Spannungsfehler)
31xx	Mains Voltage
32xx	Voltage inside the device (Spannungsfehler im Geräteinnern)
33xx	Output Voltage (Spannungsfehler, ausgangsseitig)
40xx	Temperature (Temperaturfehler)
41xx	Ambient Temperature (Umgebungstemperaturfehler)
42xx	Device Temperature (Gerätetemperaturfehler)
50xx	Device Hardware (Geräte-Hardware-Fehler)
60xx	Device Software (Geräte-Software-Fehler)
61xx	Internal Software (Firmware-Fehler)
62xx	User Software (Applications-Software)
63xx	Data Set (Daten-/Parameterfehler)
70xx	Additional Modules (zusätzliche Module)
80xx	Monitoring (Überwachung)
81xx	Communication (Kommunikation)
8110	CAN Overrun-objects lost (CAN Überlauf-Datenverlust)
8120	CAN in Error Passiv Mode (CAN im Modus "fehlerpassiv")
8130	Life Guard Error or Heartbeat Error (Guarding-Fehler oder Heartbeat-Fehler)
8140	Recovered from Bus off (Bus-Off zurückgesetzt)
8150	Transmit COB-ID collision (Senden "Kollision der COB-ID")
82xx	Protocol Error (Protokollfehler)
8210	PDO not procedded due to length error (PDO nicht verarbeitet, fehlerhafte Längenangabe)
8220	PDO length exceeded (PDO Längenfehler, ausgangsseitig)
90xx	External Error (Externer Fehler)
F0xx	Additional Functions (zusätzliche Funktionen)
FFxx	Device specific (gerätespezifisch)

Objekt 1001h (Error Register)

Dieses Objekt spiegelt den allgemeinen Fehlerzustand eines CANopen-Gerätes wider. Das Gerät ist dann als fehlerfrei anzusehen, wenn das Objekt 1001h keinen Fehler mehr signalisiert.

Bit	Meaning / Bedeutung
0	Generic Error (allgemeiner Fehler)
1	Current (Stromfehler)
2	Voltage (Spannungsfehler)
3	Temperature (Temperaturfehler)
4	Communication Error (Kommunikationsfehler)
5	Device Profile specific (Geräteprofil spezifisch)
6	Reserved – always 0 (reserviert – immer 0)
7	manufacturer specific (herstellerspezifisch)

Herstellerspezifische Informationen

Hier kann ein Gerätehersteller zusätzliche Fehlerinformationen mitteilen. Das Format ist dabei frei wählbar.

Beispiel:

In einem Gerät treten zwei Fehler auf und werden über den Bus gemeldet:

- Kurzschluss der Ausgänge:
  Fehlercode 2300h,
  im Objekt 1001h wird der Wert 03h (0000 0011b) eingetragen
  (allg. Fehler und Stromfehler)

- CAN-Überlauf:
  Fehlercode 8110h,
  im Objekt 1001h wird der Wert 13h (0001 0011b) eingetragen
  (allg. Fehler, Stromfehler und Kommunikationsfehler)

>> CAN-Überlauf bearbeitet:
Fehlercode 0000h,
im Objekt 1001h wird der Wert 03h (0000 0011b) eingetragen
(allg. Fehler, Stromfehler, Kommunikationsfehler zurückgesetzt.)
Nur aus dieser Information kann man entnehmen, dass der Kommunikationsfehler nicht mehr anliegt.

Übersicht CANopen ecomatmobil EMCY-Codes

alle Angaben für 1. CAN-Schnittstelle

EMCY-Code Objekt 1003h		Objekt 1001h	Hersteller-spezifische Informationen					
Byte 0	1	2	3	4	5	6	7	Beschreibung
00h	21h	03h	I0					Diagnose Eingänge (Bit I0...I7)
00h	31h	05h						Versorgungsspannung VBBo/VBBs
00h	61h	11h						Speicherfehler
00h	80h	11h						CAN1 Monitoring SYNC-Error (nur Slave)
00h	81h	11h						CAN1 Warngrenze (≥ 96)
10h	81h	11h						CAN1 Empfangspuffer Überlauf
11h	81h	11h						CAN1 Sendepuffer Überlauf
30h	81h	11h						CAN1 Guard-/Heartbeat-Error (nur Slave)

8.7.7 Bibliothek für den CANopen-Master

Inhalt:

Für den CANopen-Master stellt die Bibliothek `ifm_CRxxxx_CANopenMaster_Vxxyyzz.LIB` eine Reihe von Funktionen zur Verfügung, die im Folgenden erklärt werden.

Funktion CANx_MASTER_EMCY_HANDLER

x = Nr. 1...n der CAN-Schnittstelle (je nach Gerät)

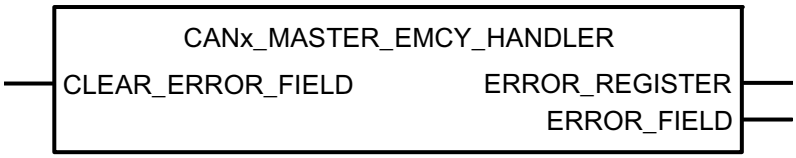
Enthalten in Bibliothek:

`ifm_CRnnnn_CANopenMaster_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360: CR1050, CR1051, CR1060
- PDM360 compact: CR1052, CR1053, CR1055, CR1056
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

Geräteeigenen Fehlerstatus überwachen

Die Funktion CANx_MASTER_EMCY_HANDLER überwacht den geräteeigenen Fehlerstatus des Masters. Die Funktion muss in folgenden Fällen aufgerufen werden:

- der Fehlerstatus soll ins Netzwerk übertragen werden und
- die Fehlermeldungen der Applikation sollen im Objektverzeichnis gespeichert werden.

> **HINWEIS**
>
> Sollen applikations-spezifische Fehlernachrichten im Objektverzeichnis gespeichert werden, muss die Funktion CANx_MASTER_EMCY_HANDLER **nach** dem (mehrfachen) Bearbeiten der Funktion CANx_MASTER_SEND_EMERGENCY (→ Seite 122) aufgerufen werden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
CLEAR_ERROR_FIELD	BOOL	TRUE: Löscht den Inhalt des Arrays ERROR_FIELD FALSE: Funktion wird nicht ausgeführt

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
ERROR_REGISTER	BYTE	Zeigt den Inhalt des OBV Index 1001h (Error Register)
ERROR_FIELD	ARRAY[0...5] OF WORD	Das Array[0...5] zeigt den Inhalt des OBV Index 1003h (Error Field). ERROR_FIELD[0]: Anzahl der gespeicherten Fehler ERROR_FIELD[1...5]: gespeicherte Fehler, der jüngste Fehler steht im Index [1]

Funktion CANx_MASTER_SEND_EMERGENCY

x = Nr. 1...n der CAN-Schnittstelle (je nach Gerät)

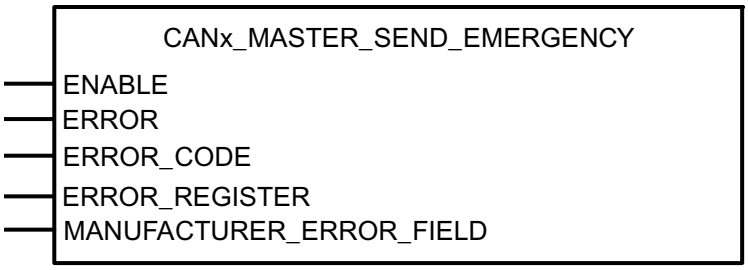
Enthalten in Bibliothek:

`ifm_CRnnnn_CANopenMaster_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360: CR1050, CR1051, CR1060
- PDM360 compact: CR1052, CR1053, CR1055, CR1056
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

Versenden von applikations-spezifischen Fehlerstati.

Die Funktion CANx_MASTER_SEND_EMERGENCY versendet applikations-spezifische Fehlerstati. Funktion wird aufgerufen, wenn der Fehlerstatus an andere Geräte im Netzwerkverbund übertragen werden soll.

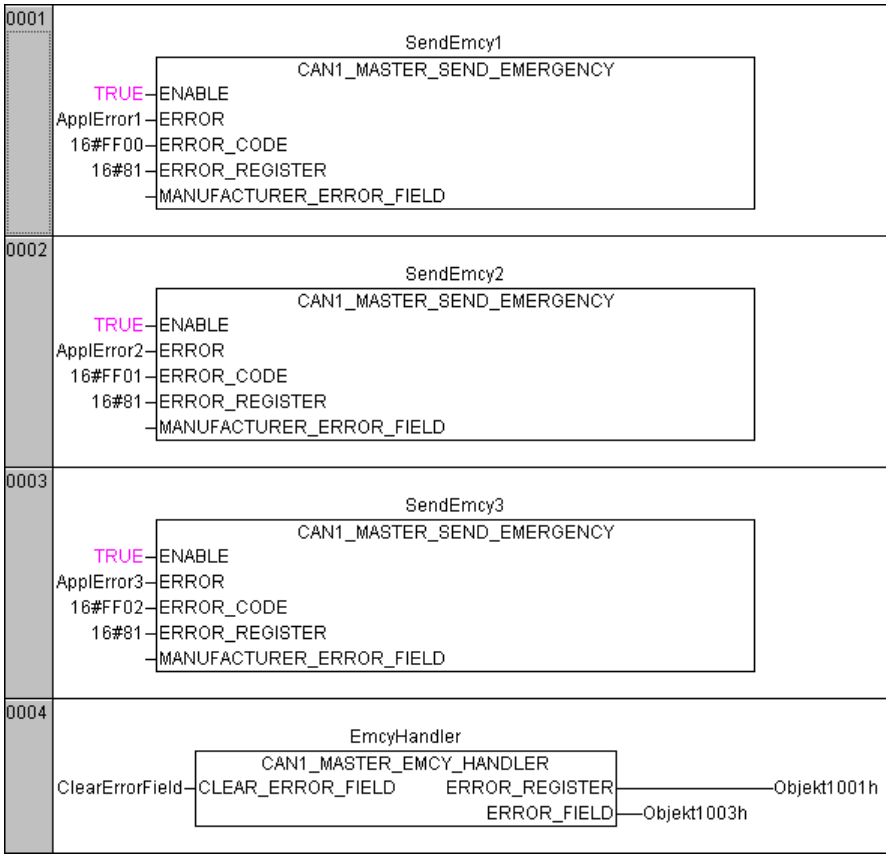
HINWEIS

Sollen applikations-spezifische Fehlernachrichten im Objektverzeichnis gespeichert werden, muss die Funktion CANx_MASTER_EMCY_HANDLER (→ Seite 120) **nach** dem (mehrfachen) Bearbeiten der Funktion CANx_MASTER_SEND_EMERGENCY aufgerufen werden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet
ERROR	BOOL	FALSE → TRUE (Flanke): sendet den anstehenden Fehlercode TRUE → FALSE (Flanke) UND Fehler steht nicht mehr an: nach Verzögerung von ca. 1 s wird Null-Fehlermeldung gesendet
ERROR_CODE	WORD	Der Error-Code gibt detailliert Auskunft über den erkannten Fehler. Die Werte sollten gemäß der CANopen-Spezifikation eingetragen werden. → Kapitel Übersicht CANopen Error-Codes, Seite 117
ERROR_REGISTER	BYTE	Dieses Objekt spiegelt den allgemeinen Fehlerzustand des CANopen-Netzwerkteilnehmers wider. Die Werte sollten gemäß der CANopen-Spezifikation eingetragen werden.
MANUFACTURER_ERROR_FIELD	ARRAY[0...4] OF BYTE	Hier können bis zu 5 Bytes applikations-spezifische Fehlerinformationen eingetragen werden. Das Format ist dabei frei wählbar.

Beispiel mit Funktion CANx_MASTER_SEND_EMERGENCY



In diesem Beispiel werden nacheinander 3 Fehlermeldungen generiert:

1. ApplError1, Code = 16#FF00 im Fehlerregister 16#81
2. ApplError2, Code = 16#FF01 im Fehlerregister 16#81
3. ApplError3, Code = 16#FF02 im Fehlerregister 16#81

Der FB CAN1_MASTER_EMCY_HANDLER sendet die Fehlermeldungen an das Fehler-Register "Objekt1001h" im Fehler-Array "Objekt1003h".

Funktion CANx_MASTER_STATUS

x = Nr. 1...n der CAN-Schnittstelle (je nach Gerät)

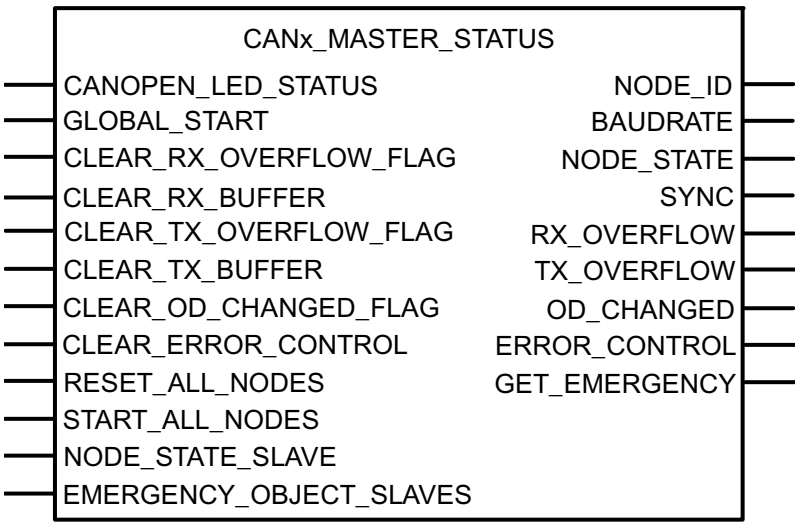
Enthalten in Bibliothek:

`ifm_CRnnnn_CANopenMaster_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360: CR1050, CR1051, CR1060
- PDM360 compact: CR1052, CR1053, CR1055, CR1056
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

Status-Anzeige des als CANopen-Master eingesetzten Gerätes

Die Funktion zeigt den Status des als CANopen-Master eingesetzten Gerätes an. Außerdem kann der Status des Netzwerks und der angeschlossenen Slaves überwacht werden.

Die Funktion vereinfacht die Anwendung der CoDeSys®-CANopen-Master-Bibliotheken. Wir empfehlen dringend, die Auswertung des Netzwerkstatus und der Fehlermeldungen über diese Funktion vorzunehmen.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
CANOPEN_LED_STATUS	BOOL	(Eingang ist nicht für PDM-Geräte verfügbar) TRUE: Die Status-LED der Steuerung wird in den Modus "CANopen" geschaltet: Blinkfrequenz 0,5 Hz = Preoperational Blinkfrequenz 2,0 Hz = Operational Die sonstigen LED-Diagnoseanzeigen werden durch diese Betriebsart nicht verändert.
GLOBAL_START	BOOL	TRUE: Alle angeschlossenen Netzwerkteilnehmer (Slaves) werden gleichzeitig bei der Netzwerkinitialisierung gestartet. FALSE: Die angeschlossenen Netzwerkteilnehmer werden einzeln nacheinander gestartet. Weitere Informationen → Kapitel Starten des Netzwerks mit GLOBAL_START, Seite 98
CLEAR_RX_OVERFLOW_FLAG	BOOL	FALSE → TRUE (Flanke): Fehlerflag "Empfangspuffer-Überlauf" löschen FALSE: Funktion wird nicht ausgeführt
CLEAR_RX_BUFFER	BOOL	FALSE → TRUE (Flanke): Daten im Empfangspuffer löschen FALSE: Funktion wird nicht ausgeführt
CLEAR_TX_OVERFLOW_FLAG	BOOL	FALSE → TRUE (Flanke): Fehlerflag "Sendepuffer-Überlauf" löschen FALSE: Funktion wird nicht ausgeführt
CLEAR_TX_BUFFER	BOOL	FALSE → TRUE (Flanke): Daten im Sendepuffer löschen FALSE: Funktion wird nicht ausgeführt
CLEAR_OD_CHANGED_FLAG	BOOL	FALSE → TRUE (Flanke): Flag "Daten im Objektverzeichnis geändert" löschen FALSE: Funktion wird nicht ausgeführt
CLEAR_ERROR_CONTROL	BOOL	FALSE → TRUE (Flanke): Die Guard-Fehlerliste (ERROR_CONTROL) löschen FALSE: Funktion wird nicht ausgeführt
RESET_ALL_NODES	BOOL	FALSE → TRUE (Flanke): Alle Knoten zurücksetzen FALSE: Funktion wird nicht ausgeführt
START_ALL_NODES	BOOL	TRUE: Alle angeschlossenen Netzwerkteilnehmer (Slaves) werden gleichzeitig zur Laufzeit des Applikations-Programms gestartet FALSE: Die angeschlossenen Netzwerkteilnehmer müssen einzeln nacheinander gestartet werden Weitere Informationen → Kapitel Starten des Netzwerks mit START_ALL_NODES, Seite 99)

Name	Datentyp	Beschreibung
NODE_STATE_SLAVE	ARRAY [0...MAX_NODEINDEX] STRUCT NODE_STATE	Um den Status eines einzelnen Netzwerkknotens zu ermitteln, kann das globale Array "NodeStateList" verwendet werden. Das Array enthält dann folgende Elemente: • NodeStateList[n].NODE_ID: Knotennummer des Slaves • NodeStateList[n].NODE_STATE: aktueller Status aus Sicht des Masters • NodeStateList[n].LAST_STATE: der CANopen-Status des Knotens • NodeStateList[n].RESET_NODE: TRUE: Slave zurücksetzen • NodeStateList[n].START_NODE: TRUE: Slave starten • NodeStateList[n].PREOP_NODE: TRUE: Slave in den Modus "Preoperation" setzen • NodeStateList[n].SET_TIMEOUT_STATE: TRUE: Timeout für Konfigurationabbruch setzen • NodeStateList[n].SET_NODE_STATE: TRUE: neuen Knotenstatus setzen Beispiel-Code → Kapitel Beispiel mit Funktion CANx_MASTER_STATUS, Seite 130 Weitere Informationen → Kapitel Der Master zur Laufzeit, Seite 94
EMERGENCY_OBJECT_SLAVES	ARRAY [0...MAX_NODEINDEX] STRUCT EMERGENCY_MESSAGE	Um eine Auflistung der zuletzt aufgetretenen Fehlermeldungen aller Netzwerkknoten zu erhalten, kann das globale Array "NodeEmergencyList" verwendet werden. Das Array enthält dann folgende Elemente: • NodeEmergencyList[n].NODE_ID: Knotennummer des Slaves • NodeEmergencyList[n].ERROR_CODE: Error-Code • NodeEmergencyList[n].ERROR_REGISTER: Error-Register • NodeEmergencyList[n].MANUFACTURER_ERROR_FIELD[0..4]: herstellerspezifische Error-Field Weitere Informationen → Kapitel Zugriff auf die Strukturen zur Laufzeit der Applikation, Seite 131

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
NODE_ID	BYTE	Node-ID des Masters
BAUDRATE	WORD	Baudrate des Masters
NODE_STATE	INT	aktueller Status des Masters.
SYNC	BOOL	SYNC-Signal des Masters. Dieses wird in Abhängigkeit der eingestellten Zeit Com. Cycle Period im Register [CAN-Parameter] (→ Seite 89) des Masters eingestellt.
RX_OVERFLOW	BOOL	Fehlerflag "Empfangspuffer-Überlauf"
TX_OVERFLOW	BOOL	Fehlerflag "Sendepuffer-Überlauf"
OD_CHANGED	BOOL	Flag "Objektverzeichnis Master wurde geändert"
ERROR_CONTROL	ARRAY [0..7] OF BYTE	Das Array enthält die Liste (max. 8) der fehlenden Netzwerkknoten (Guard- oder Heartbeat-Fehler) Weitere Informationen → Kapitel Zugriff auf die Strukturen zur Laufzeit der Applikation, Seite 131
GET_EMERGENCY	STRUCT EMERGENY_ MESSAGE	Am Ausgang stehen die Daten für die Struktur EMERGENY_MESSAGE zur Verfügung. Es wird immer die letzte Fehlermeldung eines Netzwerkknotens angezeigt. Um eine Liste aller aufgetretenen Fehler zu erhalten, muss das Array "EMERGENCY_OBJECT_SLAVES" ausgewertet werden.

Parameter interne Strukturen

Hier sehen Sie die Strukturen der in dieser Funktion genutzten Arrays.

Name	Datentyp	Beschreibung
CANx_EMERGENY_MESSAGE	STRUCT	NODE_ID: BYTE ERROR_CODE: WORD ERROR_REGISTER: BYTE MANUFACTURER_ERROR_FIELD: ARRAY[0..4] OF BYTE Die Struktur ist in den globalen Variablen der Bibliothek `ifm_CRnnnn_CANopenMaster_Vxxyyzz.LIB` angelegt.
CANx_NODE_STATE	STRUCT	NODE_ID: BYTE NODE_STATE: BYTE LAST_STATE: BYTE RESET_NODE: BOOL START_NODE: BOOL PREOP_NODE: BOOL SET_TIMEOUT_STATE: BOOL SET_NODE_STATE: BOOL Die Struktur ist in den globalen Variablen der Bibliothek `ifm_CRnnnn_CANopenMaster_Vxxyyzz.LIB` angelegt.

Ausführliche Beschreibung der Funktionalitäten des CANopen-Masters und der Mechanismen
→ Kapitel CANopen-Master, Seite 87.

Die folgenden Code-Fragmente zeigen Ihnen am Beispiel des Controllers CR0020 die Anwendung der Funktion CANx_MASTER_STATUS (→ Seite 125).

Beispiel mit Funktion CANx_MASTER_STATUS

Slave-Informationen

Damit Sie auf die Informationen der einzelnen CANopen-Knoten zugreifen können, müssen Sie ein Array über die jeweilige Struktur bilden. Die Strukturen sind in der Bibliothek enthalten. Sie können Sie im Bibliotheksverwalter unter [Datentypen] sehen.

Die Anzahl der Array-Elemente wird bestimmt durch die Globale Variable MAX_NODEINDEX, die automatisch vom CANopen-Stack angelegt wird. Sie enthält die Anzahl der im Netzwerkkonfigurator angegebenen Slaves minus 1.

HINWEIS

Die Nummern der Array-Elemente entsprechen **nicht** dem Node-ID. Der Identifier kann aus der jeweiligen Struktur unter NODE_ID ausgelesen werden.

```
0001 PROGRAM MasterStatus
0002 VAR
0003     Status: CR0020_MASTER_STATUS;
0004     LedStatus: BOOL:= TRUE;
0005     GlobalStartNodes: BOOL:= TRUE;
0006     ClearRxOverflowFlag: BOOL;
0007     ClearRxBuffer: BOOL;
0008     ClearTxOverflowFlag: BOOL;
0009     ClearTxBuffer: BOOL;
0010     ClearOdChanged: BOOL;
0011     ClearErrorControl: BOOL;
0012     ResetAllNodes: BOOL;
0013     StartAllNodes: BOOL;
0014     NodeId: BYTE;
0015     Baudrate: WORD;
0016     NodeState: INT;
0017     Sync: BOOL;
0018     RxOverflow: BOOL;
0019     TxOverflow: BOOL;
0020     OdChanged: BOOL;
0021     GuardHeartbeatErrorArray: ARRAY[0..7] OF BYTE;
0022     GetEmergency: EMERGENCY_MESSAGE;
0023 END_VAR
```

Struktur Knoten-Status

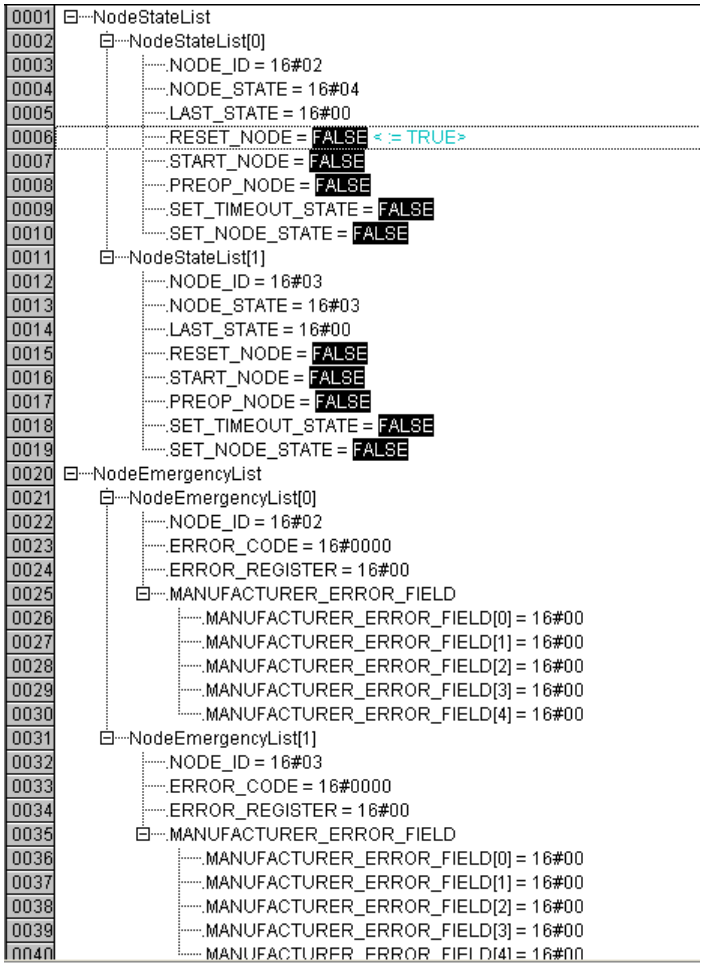
```
TYPE CAN1_NODE_STATE :
STRUCT
    NODE_ID: BYTE;
    NODE_STATE: BYTE;
    LAST_STATE: BYTE;
    RESET_NODE: BOOL;
    START_NODE: BOOL;
    PREOP_NODE: BOOL;
    SET_TIMEOUT_STATE: BOOL;
    SET_NODE_STATE: BOOL;
END_STRUCT
END_TYPE
```

Struktur Emergency_Message

```
TYPE CAN1_EMERGENCY_MESSAGE :
STRUCT
    NODE_ID: BYTE;
    ERROR_CODE: WORD;
    ERROR_REGISTER: BYTE;
    MANUFACTURER_ERROR_FIELD: ARRAY[0..4] OF BYTE;
END_STRUCT
END_TYPE
```

Zugriff auf die Strukturen zur Laufzeit der Applikation

Zur Laufzeit können Sie auf das jeweilige Array-Element über die globalen Variablen der Bibliothek zugreifen und so den Status oder die EMCY-Nachrichten auslesen oder den Knoten zurücksetzen.



Setzen Sie im obigen Beispiel ResetSingleNodeArray[0].RESET_NODE kurzzeitig auf TRUE, wird der erste Knoten im Konfigurationsbaum zurückgesetzt.

Weitere Informationen zu den möglichen Fehler-Codes → Kapitel Informationen zur EMCY- und Error-Codes, Seite 115.

8.7.8 Bibliothek für den CANopen-Slave

Inhalt:

Für den CANopen-Slave (= CANopen-Device = CANopen-Node) stellt die Bibliothek `ifm_CRxxxx_CANopenSlave_Vn.LIB` eine Reihe von Funktionen zur Verfügung, die im Folgenden erklärt werden.

Funktion CANx_SLAVE_NODEID

x = Nr. 1...n der CAN-Schnittstelle (je nach Gerät)

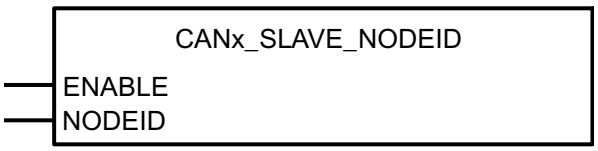
Enthalten in Bibliothek:

`ifm_CRnnnn_CANopenSlave_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360: CR1050, CR1051, CR1060
- PDM360 compact: CR1052, CR1053, CR1055, CR1056
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

Die Funktion CANx_SLAVE_NODEID ermöglicht das Einstellen des Node-ID eines CAN-Device (Slave) zur Laufzeit des Applikations-Programms.

Die Funktion wird im Normalfall bei der Initalisierung der Steuerung einmalig, im ersten Zyklus, aufgerufen. Anschließend wird der Eingang ENABLE wieder auf FALSE gesetzt.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	FALSE → TRUE (Flanke): NodeID setzen FALSE: Funktion wird nicht ausgeführt
NODEID	BYTE	Wert der neuen Knotennummer

Funktion CANx_SLAVE_EMCY_HANDLER

x = Nr. 1...n der CAN-Schnittstelle (je nach Gerät)

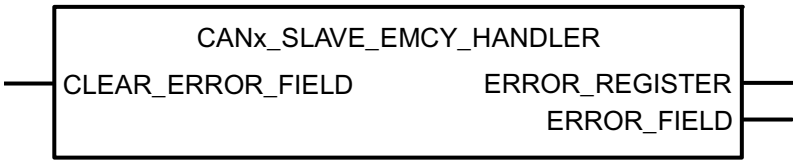
Enthalten in Bibliothek:

`ifm_CRnnnn_CANopenSlave_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360: CR1050, CR1051, CR1060
- PDM360 compact: CR1052, CR1053, CR1055, CR1056
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

Die Funktion CANx_SLAVE_EMCY_HANDLER überwacht den geräteeigenen Fehlerstatus (Gerät wird als Slave betrieben).

Die Funktion muss in folgenden Fällen aufgerufen werden:

- der Fehlerstatus soll ins CAN-Netzwerk übertragen werden und
- die Fehlermeldungen der Applikation sollen im Objektverzeichnis gespeichert werden.

> **HINWEIS**
>
> Sollen applikations-spezifische Fehlernachrichten im Objektverzeichnis gespeichert werden, muss die Funktion CANx_SLAVE_EMCY_HANDLER (→ Seite 134) **nach** dem (mehrfachen) Bearbeiten der Funktion CANx_SLAVE_SEND_EMERGENCY aufgerufen werden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
CLEAR_ERROR_FIELD	BOOL	FALSE → TRUE (Flanke): ERROR-FIELD löschen FALSE: Funktion wird nicht ausgeführt

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
ERROR_REGISTER	BYTE	Zeigt den Inhalt des OBV Index 1001h (Error Register).
ERROR_FIELD	ARRAY[0...5] OF WORD	Das Array[0...5] zeigt den Inhalt des OBV Index 1003h (Error Field): ERROR_FIELD[0]: Anzahl der gespeicherten Fehler ERROR_FIELD[1...5]: gespeicherte Fehler, der jüngste Fehler steht im Index [1].

Funktion CANx_SLAVE_SEND_EMERGENCY

x = Nr. 1...n der CAN-Schnittstelle (je nach Gerät)

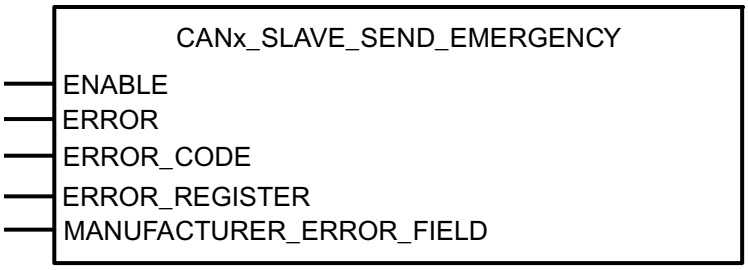
Enthalten in Bibliothek:

`ifm_CRnnnn_CANopenSlave_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360: CR1050, CR1051, CR1060
- PDM360 compact: CR1052, CR1053, CR1055, CR1056
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

Durch die Funktion CANx_SLAVE_SEND_EMERGENCY werden applikations-spezifische Fehlerstati versendet. Das sind Fehlernachrichten, die zusätzlich zu den geräteinternen Fehlernachrichten (z.B. Kurzschluss am Ausgang) gesendet werden sollen.

Die Funktion wird aufgerufen, wenn der Fehlerstatus an andere Geräte im Netzwerkverbund übertragen werden soll.

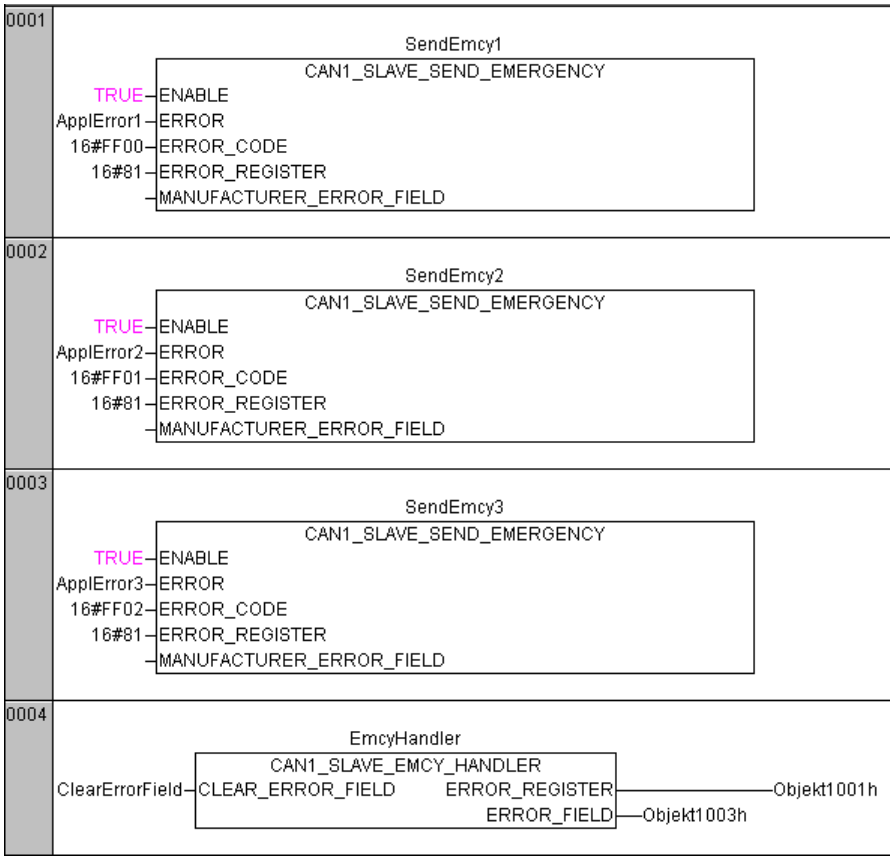
HINWEIS

Sollen applikations-spezifische Fehlermachrichten im Objektverzeichnis gespeichert werden, muss die Funktion CANx_SLAVE_EMCY_HANDLER (→ Seite 134) **nach** dem (mehrfachen) Bearbeiten der Funktion CANx_SLAVE_SEND_EMERGENCY aufgerufen werden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet
ERROR	BOOL	FALSE → TRUE (Flanke): sendet den anstehenden Fehlercode TRUE → FALSE (Flanke) UND Fehler steht nicht mehr an: nach Verzögerung von ca. 1 s wird Null-Fehlermeldung gesendet
ERROR_CODE	WORD	Der Error-Code gibt detailliert Auskunft über den erkannten Fehler. Die Werte sollten gemäß der CANopen-Spezifikation eingetragen werden. → Kapitel Übersicht CANopen Error Codes, Seite 117
ERROR_REGISTER	BYTE	Dieses Objekt spiegelt den allgemeinen Fehlerzustand des CANopen-Netzwerkteilnehmers wider. Die Werte sollten gemäß der CANopen-Spezifikation eingetragen werden.
MANUFACTURER_ERROR_FIELD	ARRAY[0...4] OF BYTE	Hier können bis zu 5 Bytes applikations-spezifische Fehlerinformationen eingetragen werden. Das Format ist dabei frei wählbar.

Beispiel mit Funktion CANx_SLAVE_SEND_EMERGENCY



In diesem Beispiel werden nacheinander 3 Fehlermeldungen generiert:

1. ApplError1, Code = 16#FF00 im Fehlerregister 16#81
2. ApplError2, Code = 16#FF01 im Fehlerregister 16#81
3. ApplError3, Code = 16#FF02 im Fehlerregister 16#81

Der FB CAN1_SLAVE_EMCY_HANDLER sendet die Fehlermeldungen an das Fehler-Register "Objekt1001h" im Fehler-Array "Objekt1003h".

Funktion CANx_SLAVE_STATUS

x = Nr. 1...n der CAN-Schnittstelle (je nach Gerät)

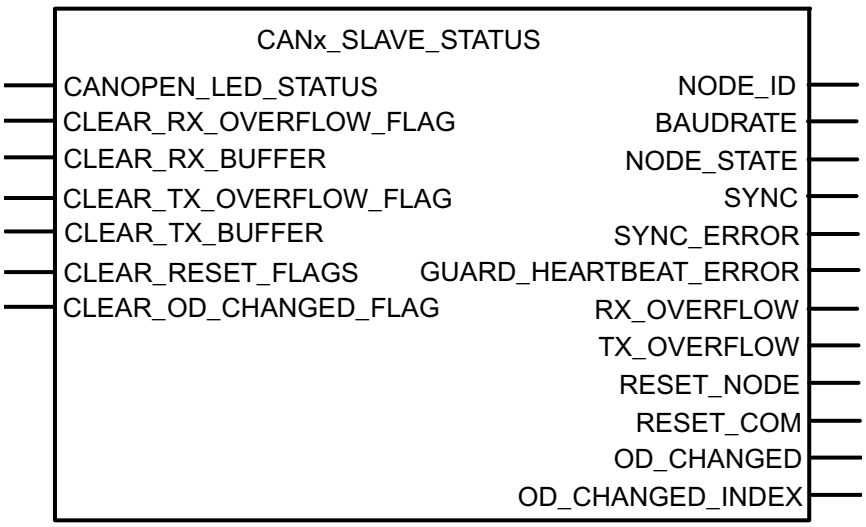
Enthalten in Bibliothek:

`ifm_CRnnnn_CANopenSlave_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015

Funktionssymbol:



Beschreibung

Die Funktion CANx_SLAVE_STATUS zeigt den Status des als CANopen-Slave eingesetzten Gerätes an. Die Funktion vereinfacht die Anwendung der CoDeSys®-CAN-Device-Bibliotheken. Wir empfehlen dringend, die Auswertung des Netzwerkstatus über diese Funktion vorzunehmen.

Info

Eine ausführliche Beschreibung der Funktionalitäten des CANopen-Slaves und der Mechanismen
→ Kapitel CANopen-Device, Seite 102.

Zur Laufzeit können Sie dann auf die einzelnen Funktionsausgänge des Bausteins zugreifen, um eine Statusübersicht zu erhalten.

Beispiel:

```
0001 PROGRAM SlaveStatus
0002 VAR
0003     SlaveStatus: CR0505_SLAVE_STATUS;
0004     LedStatus: BOOL := TRUE;
0005     ClearRxOverflowFlag: BOOL;
0006     ClearRxBuffer: BOOL;
0007     ClearTxOverflowFlag: BOOL;
0008     ClearTxBuffer: BOOL;
0009     ClearResetFlags: BOOL;
0010     ClearOdChanged: BOOL;
0011     NodeId: BYTE;
0012     Baudrate: WORD;
0013     NodeState: BYTE;
0014     Sync: BOOL;
0015     SyncError: BOOL;
0016     GuardHeartbeatError: BOOL;
0017     RxOverflow: BOOL;
0018     TxOverflow: BOOL;
0019     ResetNode: BOOL;
0020     ResetCom: BOOL;
0021     OdChanged: BOOL;
0022     OdChangedIndex: INT;
0023 END_VAR
```

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
GLOBAL_START	BOOL	TRUE: Alle angeschlossenen Netzwerkteilnehmer (Slaves) werden gleichzeitig bei der Netzwerk-Initialisierung gestartet. FALSE: Die angeschlossenen Netzwerkteilnehmer werden einzeln nacheinander gestartet. Weitere Informationen → Kapitel Starten des Netzwerkes mit GLOBAL_START, Seite 98
CLEAR_RX_OVERFLOW_FLAG	BOOL	FALSE → TRUE (Flanke): Fehlerflag "Empfangspuffer-Überlauf" löschen FALSE: Funktion wird nicht ausgeführt
CLEAR_RX_BUFFER	BOOL	FALSE → TRUE (Flanke): Daten im Empfangspuffer löschen FALSE: Funktion wird nicht ausgeführt
CLEAR_TX_OVERFLOW_FLAG	BOOL	FALSE → TRUE (Flanke): Fehlerflag "Sendepuffer-Überlauf" löschen FALSE: Funktion wird nicht ausgeführt
CLEAR_TX_BUFFER	BOOL	FALSE → TRUE (Flanke): Daten im Sendepuffer löschen FALSE: Funktion wird nicht ausgeführt

Name	Datentyp	Beschreibung
CLEAR_RESET_FLAG	BOOL	FALSE → TRUE (Flanke): Die Flags "Knoten zurückgesetzt" und "Kommunikationsschnittstelle zurückgesetzt" löschen. FALSE: Funktion wird nicht ausgeführt
CLEAR_OD_CHANGED_FLAG	BOOL	FALSE → TRUE (Flanke): Flags "Daten im Objektverzeichnis geändert" und "Index-Position" löschen FALSE: Funktion wird nicht ausgeführt

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
NODE_ID	BYTE	Node-ID des Slaves
BAUDRATE	WORD	Baudrate des Slaves
NODE_STATE	BYTE	aktueller Status des Slaves
SYNC	BOOL	Empfangenes SYNC-Signal des Masters
SYNC_ERROR	BOOL	Es wurde kein SYNC-Signal des Masters empfangen ODER: die eingestellte SYNC-Zeit (ComCyclePeriod im Master) wurde überschritten.
GUARD_HEARTBEAT_ERROR	BOOL	Es wurde kein Guard- oder Heartbeat-Signal des Masters empfangen ODER: die eingestellten Zeiten wurden überschritten
RX_OVERFLOW	BOOL	Fehlerflag "Empfangspuffer-Überlauf"
TX_OVERFLOW	BOOL	Fehlerflag "Sendepuffer-Überlauf"
RESET_NODE	BOOL	Der CAN-Stack des Slaves wurde vom Master zurückgesetzt. Dieses Flag kann von der Applikation ausgewertet und ggf. für weitere Reaktionen genutzt werden.
RESET_COM	BOOL	Das Kommunikationsinterface des CAN-Stack wurde vom Master zuückgesetzt. Dieses Flag kann von der Applikation ausgewertet und ggf. für weitere Reaktionen genutzt werden.
OD_CHANGED	BOOL	Flag "Objektverzeichnis Master wurde geändert"
OD_CAHNGED_INDEX	INT	Ausgang zeigt den geänderten Index des Objektverzeichnisses

8.7.9 Weitere ifm-Bibliotheken zu CANopen

Inhalt:

Hier stellen wir Ihnen weitere **ifm**-Funktionen vor, die für CANopen sinnvolle Ergänzungen darstellen.

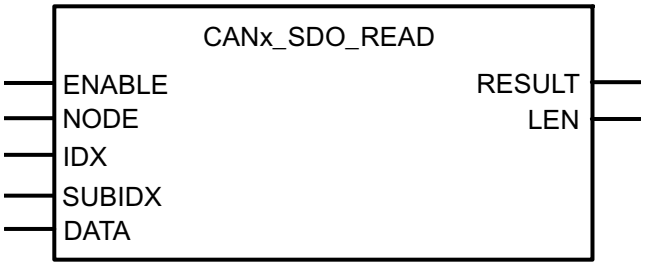
Funktion CANx_SDO_READ

x = Nr. 1...n der CAN-Schnittstelle (je nach Gerät)

Enthalten in Bibliothek:

ifm_CRnnnn_Vxxyyzz.LIB	ifm_CANx_SDO_Vxxyyzz.LIB
verfügbar für: ClassicController: CR0020, CR0032, CR0505 ExtendedController: CR0200, CR0232 SmartController: CR2500 SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201 CabinetController: CR0301, CR0302, CR0303 Platinensteuerung: CS0015 PDM360 smart: CR1070, CR1071	verfügbar für: PDM360: CR1050, CR1051, CR1060 PDM360 compact: CR1052, CR1053, CR1055, CR1056

Funktionssymbol:



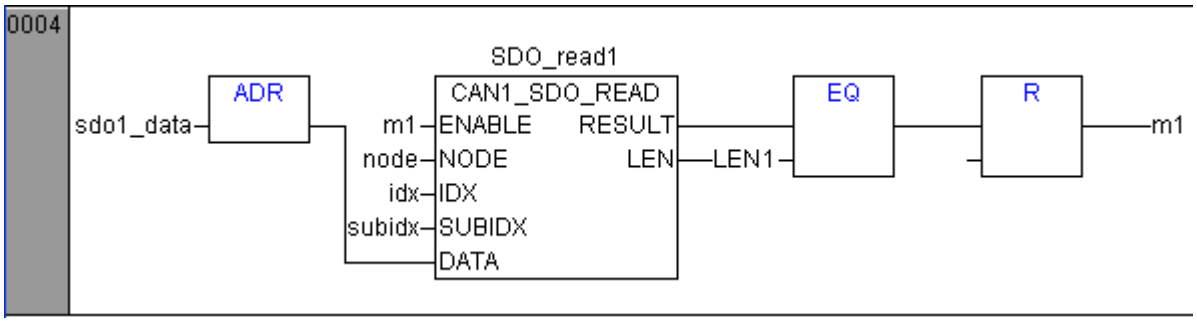
Beschreibung

CANx_SDO_READ liest das SDO (→ Kapitel Register [Service Data Objects], Seite 93) mit den angegebenen Indizes aus dem Knoten aus.

Über diese können die Einträge im Objektverzeichnis gelesen werden. Dadurch ist es möglich, die Knotenparameter gezielt zu lesen.

Controller Platinensteuerung PDM360 smart	PDM360 compact PDM360
aus Gerätebibliothek `ifm_CRnnnn_Vxxyyzz.LIB`	aus Gerätebibliothek `ifm_CANx_SDO_Vxxyyzz.LIB`
Voraussetzung: Knoten muss sich im Zustand "Pre-Operational" oder "Operational" befinden.	Voraussetzung: Knoten muss sich im Modus "CANopen-Master" oder "CAN-Device" befinden.

Beispiel:



Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht ausgeführt
NODE	BYTE	Nummer des Knotens
IDX	WORD	Index im Objektverzeichnis
SUBIDX	BYTE	Subindex bezogen auf den Index im Objektverzeichnis
DATA	DWORD	Adresse des Empfangsdaten-Arrays zulässige Länge = 0...255 Übergabe mit ADR-Operator

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BYTE	0 = Funktion inaktiv 1 = Funktionsausführung beendet 2 = Funktion ist aktiv 3 = Funktion wurde nicht ausgeführt
LEN	WORD	Länge des Eintrags in "Anzahl der Bytes" Der Wert für LEN muss mit der Länge des Empfangs-Arrays übereinstimmen. Andernfalls treten Störungen bei der SDO-Kommunikation auf.

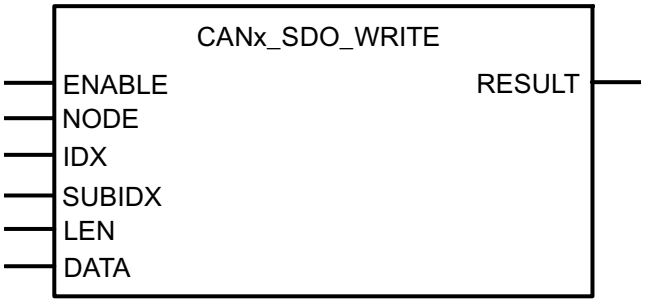
Funktion CANx_SDO_WRITE

x = Nr. 1...n der CAN-Schnittstelle (je nach Gerät)

Enthalten in Bibliothek:

ifm_CRnnnn_Vxxyyzz.LIB	ifm_CANx_SDO_Vxxyyzz.LIB
verfügbar für: ClassicController: CR0020, CR0032, CR0505 ExtendedController: CR0200, CR0232 SmartController: CR2500 SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201 CabinetController: CR0301, CR0302, CR0303 Platinensteuerung: CS0015 PDM360 smart: CR1070, CR1071	verfügbar für: PDM360: CR1050, CR1051, CR1060 PDM360 compact: CR1052, CR1053, CR1055, CR1056

Funktionssymbol:



Beschreibung

CANx_SDO_WRITE schreibt das SDO (→ Kapitel Register [Service Data Objects], Seite 93) mit den angegebenen Indizes in den Knoten.

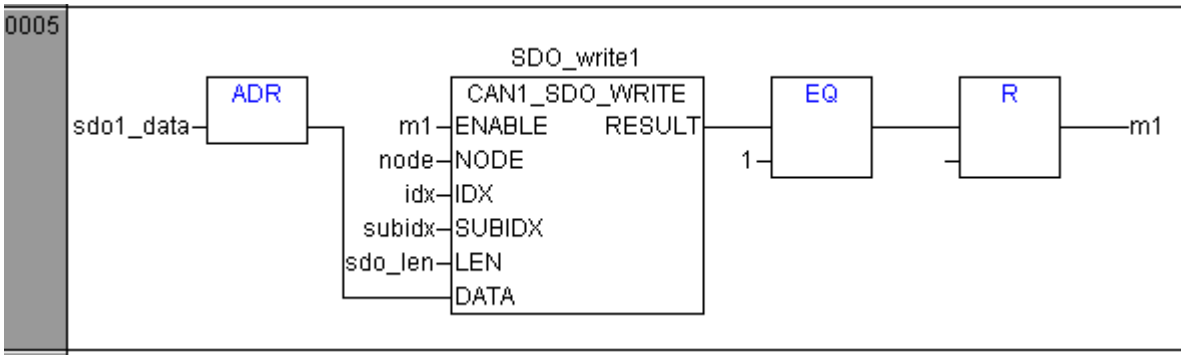
Über diese Funktion können die Einträge im Objektverzeichnis geschrieben werden. Dadurch ist es möglich, die Knotenparameter gezielt zu setzen.

Controller Platinensteuerung PDM360 smart	**PDM360 compact PDM360**
aus Gerätebibliothek `ifm_CRnnnn_Vxxyyzz.LIB`	aus Gerätebibliothek `ifm_CANx_SDO_Vxxyyzz.LIB`
Voraussetzung: Knoten muss sich im Zustand "Pre-Operational" oder "Operational" befinden und im Modus "CANopen-Master".	Voraussetzung: Knoten muss sich im Modus "CANopen-Master" oder "CAN-Device" befinden.

HINWEIS

Der Wert für LEN muss mit der Länge des Sendearrays übereinstimmen. Andernfalls treten Störungen bei der SDO-Kommunikation auf.

Beispiel:



Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht ausgeführt
NODE	BYTE	Nummer des Knotens
IDX	WORD	Index im Objektverzeichnis
SUBIDX	BYTE	Subindex bezogen auf den Index im Objektverzeichnis.
LEN	WORD	Länge des Eintrags in "Anzahl der Bytes" Der Wert für LEN muss mit der Länge des Sende-Arrays übereinstimmen. Andernfalls treten Störungen bei der SDO-Kommunikation auf.
DATA	DWORD	Adresse des Sendedaten-Arrays zulässige Länge = 0...255 Übergabe mit ADR-Operator

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BYTE	0 = Funktion inaktiv 1 = Funktionsausführung beendet 2 = Funktion ist aktiv 3 = Funktion wurde nicht ausgeführt

8.8 Zusammenfassung CAN / CANopen

- Die COB-ID der Netzwerkvariablen muss sich unterscheiden von der CANopen Device-ID in der Steuerungskonfiguration und von den IDs der Funktionen CANx_TRANSMIT und CANx_RECEIVE!
- Wenn mehr als 8 Bytes von Netzwerkvariablen in eine COB-ID gepackt werden, erweitert CANopen das Datenpaket automatisch auf mehrere aufeinander folgende COB-IDs. Dies kann zu Konflikten mit manuell definierten COB-IDs führen!
- Netzwerkvariable können keine String-Variablen transportieren.
- Netzwerkvariable können transportiert werden...
  - wenn eine Variable TRUE wird (Event),
  - bei Datenänderung in der Netzwerkvariablen oder
  - zyklisch nach Zeitablauf.
- Die Intervall-Zeit beschreibt die Periode zwischen Übertragungen bei zyklischer Übertragung. Der Mindestabstand beschreibt die Wartezeit zwischen zwei Übertragungen, wenn die Variable sich zu oft ändert.
- Um die Buslast zu mindern, die Nachrichten via Netzwerkvariablen oder CANx_TRANSMIT mit Hilfe von verschiedenen Events auf mehrere SPS-Zyklen verteilen.
- Jeder Aufruf von CANx_TRANSMIT oder CANx_RECEIVE erzeugt ein Nachrichtenpaket von 8 Bytes.
- In der Steuerungskonfiguration sollten die Werte für [Com Cycle Period] und [Sync. Window Length] gleich groß sein. Diese Werte müssen größer sein als die SPS-Zykluzeit.
- Wenn die [Com Cycle Period] für einen Slave eingestellt ist, sucht der Slave in genau dieser Zeit nach einem Sync-Objekt des Masters. Deshalb muss der Wert für [Com Cycle Period] größer sein als die [Master Synch Time].
- Wir empfehlen, Slaves als "optional startup" und das Netzwerk als "automatic startup" zu setzen. Dies reduziert unnötige Buslast und ermöglicht einem kurzzeitig verlorenen Slave, sich wieder in das Netzwerk zu integrieren.
- Weil wir keinen Inhibit-Timer haben, empfehlen wir, Analog-Eingänge auf "synchrone Übertragung" zu setzen, um Busüberlastung zu vermeiden.
- Binäre Eingänge, insbesondere die unregelmäßig schaltenden, sollten am besten auf "asynchrone Übertragung" mittels Event-Timer gesetzt werden.
- Beim Überwachen des Slave-Status beachten:
  - Nach dem Starten von Slaves dauert es etwas, bis die Slaves "operational" sind.
  - Beim Abschalten des Systems können Slaves wegen vorzeitigem Spannungsverlust eine scheinbare Status-Änderung anzeigen.

8.9 Nutzung der CAN-Schnittstellen nach SAE J1939

Inhalt:

Die CAN-Schnittstellen in den Controllern kann auch zur Kommunikation mit speziellen Busprotokollen für die Antriebstechnik und aus dem Nutzfahrzeugbereich eingesetzt werden. Bei diesen Protokollen wird der CAN-Controller der 2. Schnittstelle in den sogenannten "Extended Mode" geschaltet. Das bedeutet, dass die CAN-Nachrichten mit einem 29 Bit-Identifier übertragen werden. Durch den längeren Identifier kann eine große Anzahl von Nachrichten direkt dem Identifier zugeordnet werden.

Bei der Protokollerstellung hat man sich diesen Vorteil zu Nutze gemacht und gruppiert bestimmte Nachrichten in ID-Gruppen. Die Zuordnung der IDs ist in den Normen SAE J1939 und ISO 11992 festgeschrieben. Das Protokoll der ISO 11992 baut auf dem Protokoll der SAE J1939 auf.

Norm	Einsatzbereich
SAE J1939	Antriebs-Management (ist mit diesem Controller möglich)
ISO 11992	Truck & Trailer Interface (benötigt andere Hardware wegen höherer Spannungspegel)

Der 29 Bit-Identifier setzt sich aus zwei Teilen zusammen:
- einem 11 Bit-ID und
- einem 18 Bit-ID.

Vom Software-Protokoll unterscheiden sich die beiden Normen nicht, da die ISO 11992 auf der SAE J1939 aufbaut. Bezüglich der Hardwareschnittstelle besteht aber ein Unterschied: höhere Spannungspegel bei der ISO 11992. Für die Kommunikation von Aggregaten mit der Schnittstelle ISO 11992 werden aus diesem Grunde Controller mit einer modifizierten CAN-Schnittstelle benötigt.

HINWEIS

Zur Nutzung der Funktionen nach SAE J1939 benötigt man auf jeden Fall die Protokollbeschreibung des Aggregat-Herstellers (z.B. für Motor, Getriebe). Aus dieser müssen die in das Aggregat-Steuergerät implementierten Nachrichten entnommen werden, da nicht jeder Hersteller alle Nachrichten implementiert oder die Implementierung nicht für alle Aggregate sinnvoll ist.

Tabelle: Aufbau des Identifiers

Priority			Reserviert	Data Page	PDU format							PDU specific								Source / Destination address							
29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Folgende Informationen und Hilfsmittel sollten zur Entwicklung von Programmen für Funktionen nach SAE J1939 vorhanden sein:

- Aufstellung, welche Daten von den Aggregaten genutzt werden sollen
- Übersichtsliste des Aggregatherstellers mit allen relevanten Daten
- CAN-Monitor mit 29 Bit-Unterstützung
- Wenn benötigt, die Norm SAE J1939

Beispiel für eine ausführliche Nachrichten-Dokumentation:

ETC1: Electronic Transmission Controller #1 (3.3.5) 0CF00203

Transmission repetition rate	10 ms
Data length:	8 Bytes
PDU format	240
PDU specific	2
Default priority	3
Data Page	0
Source Address	3
Parameter group number	$00F002_{16}$
Identifier	$0CF00203_{16}$
Data Field	Die Bedeutung der Datenbytes 1...8 wird an dieser Stelle nicht weiter behandelt. Sie ist der Herstellerdokumentation zu entnehmen.

Da im Beispiel vom Hersteller alle relevanten Daten bereits aufbereitet wurden, können diese direkt an die Funktionsblöcke übertragen werden.

Dabei bedeuten:

Bezeichung in der Herstellerdokumentation	**Funktionseingang Bibliotheksfunktion**	**Beispielwert**
Transmission repetition rate	RPT	T#10ms
Data length	LEN	8
PDU format	PF	240
PDU specific	PS	2
Default priority	PRIO	3
Data Page	PG	0
Source Address / Destination Address	SA / DA	3
Data Field	SRC / DST	Array-Adresse

Je nach benötigter Funktion werden die entsprechenden Werte eingesetzt. Bei den Feldern SA / DA oder SRC / DST ändert sich die Bedeutung (aber nicht der Wert), entsprechend der Empfangs- oder der Sendefunktion.

Die einzelnen Datenbytes müssen aus dem Array ausgelesen und entsprechend ihrer Bedeutung weiterverarbeitet werden.

Beispiel für eine kurze Nachrichten-Dokumentation:

Aber auch wenn vom Aggregathersteller nur eine Kurzdokumentation zur Verfügung steht, kann man sich die Funktionsparameter aus dem Identifier herleiten. Neben dem ID werden zusätzlich in jedem Fall die "Transmission repetition rate" und die Bedeutung der Datenfelder benötigt.

Wenn es sich nicht um herstellerspezifische Protokollnachrichten handelt, kann auch die Norm SAE J1939 oder ISO 11992 als Infomationsquelle dienen.

Der Identifier 0x0CF00203 setzt sich wie folgt zusammen:

PRIO, reserv., PG		PF + PS				SA / DA	
0	C	F	0	0	2	0	3

Da es sich bei diesen Werten um hexadezimale Zahlen handelt, von denen man teilweise einzelne Bits benötigt, müssen die Zahlen weiter zerlegt werden:

SA / DA		Source / Destination Address (hexadezimal)		Source / Destination Address (dezimal)	
0	3	00	03	0	3

PF		PDU format (PF) (hexadezimal)		PDU format (PF) (dezimal)	
F	0	0F	00	16	0

PS		PDU specific (PS) (hexadezimal)		PDU specific (PS) (dezimal)	
0	2	00	02	0	2

PRIO, reserv., PG		PRIO, reserv., PG (binär)	
0	C	0000	1100

Von den 8 Bit ($0C_{16}$) werden nur die 5 niederwertigen Bits benötigt:

nicht benötigt			Priority			res.	PG
x	x	x	0_2	1_2	1_2	0_2	0_2
			03_{10}			0_{10}	0_{10}

Weitere typische Kombinationen für "PRIO, reserv., PG "

18_{16}:

nicht benötigt			Priority			res.	PG
x	x	x	1_2	1_2	0_2	0_2	0_2
			6_{10}			0_{10}	0_{10}

$1C_{16}$:

nicht benötigt			Priority			res.	PG
x	x	x	1_2	1_2	1_2	0_2	0_2
			7_{10}			0_{10}	0_{10}

8.9.1 Funktion J1939_x

x = Nr. 1...n der CAN-Schnittstelle (je nach Gerät, → Datenblatt)

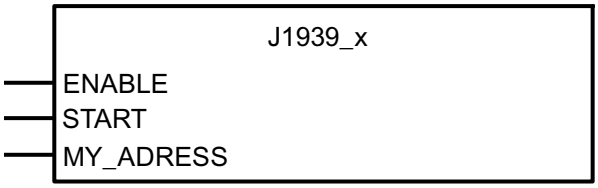
Enthalten in Bibliothek:

`ifm_J1939_x_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0303
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

J1939_x dient als Protokoll-Handler für das Kommunikationsprofil SAE J1939.

Zur Abwicklung der Kommunikation muss der Protokoll-Handler in jedem Programmzyklus aufgerufen werden. Dazu wird der Eingang ENABLE auf TRUE gesetzt.

Der Protokoll-Handler wird gestartet, wenn der Eingang START für einen Zyklus auf TRUE gesetzt wird.

Über MY_ADRESS wird dem Controller eine Geräteadresse übergeben. Sie muss sich von Adressen der anderen J1939-Busteilnehmer unterscheiden. Sie kann dann von anderen Busteilnehmern ausgelesen werden.

HINWEIS

J1939-Kommunikation über 1. CAN-Schnittstelle:	J1939-Kommunikation über 2. CAN-Schnittstelle:
► Schnittstelle zuvor über die Funktion CAN1_EXT (→ Seite 65) initialisieren!	► Schnittstelle zuvor über die Funktion CAN2 (→ Seite 71) initialisieren!

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht ausgeführt
START	BOOL	TRUE (nur 1 Zyklus): Protokoll-Handler wird gestartet FALSE: im zyklischen Programmablauf
MY_ADRESS	BYTE	Geräteadresse des Controllers

8.9.2 Funktion J1939_x_RECEIVE

x = Nr. 1...n der CAN-Schnittstelle (je nach Gerät, → Datenblatt)

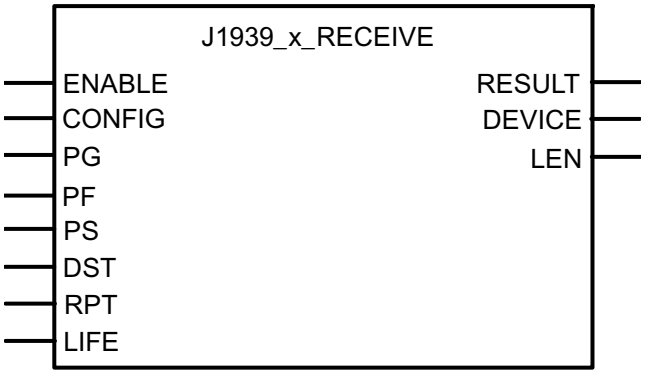
Enthalten in Bibliothek:

`ifm_J1939_x_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0303
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

J1939_x_RECEIVE dient dem Empfang einer einzelnen Nachricht oder eines Nachrichtenblocks.

Dazu muss die Funktion über den Eingang CONFIG für einen Zyklus initialisiert werden. Bei der Initialisierung werden die Parameter PG, PF, PS, RPT, LIFE und die Speicheradresse des Datenarrays DST übergeben. Die Adresse muss über die Funktion ADR ermittelt werden.

Der Datenempfang muss über das RESULT-Byte ausgewertet werden. Wird RESULT = 1, können die Daten von der über DST übergebenen Speicheradresse ausgelesen und weiter verarbeitet werden. Der Empfang einer neuen Nachricht überschreibt die Daten auf der Speicheradresse DST.

Die Anzahl der empfangenen Nachrichten-Bytes wird über den Funktionsausgang LEN angezeigt.

Wird RESULT = 3, wurden im angegebenen Zeitfenster (LIFE * RPT) keine gültigen Nachrichten empfangen.

HINWEIS

Dieser Baustein muss auch eingesetzt werden, wenn die Nachrichten mit den Funktionen J1939_..._REQUEST angefordert werden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet. FALSE: Funktion wird nicht ausgeführt
CONFIG	BOOL	TRUE (nur 1 Zyklus): zur Konfiguration des Datenobjektes FALSE: im weiteren Programmablauf
PG	BYTE	Page address (normalerweise = 0)
PF	BYTE	PDU Format Byte
PS	BYTE	PDU Specific Byte
DST	DWORD	Zieladresse des Arrays, unter der die Empfangsdaten abgelegt werden
RPT	TIME	Überwachungszeit Innerhalb dieses angegebenen Zeitfensters müssen die Telegramme wiederholt empfangen werden. Andernfalls erfolgt eine Fehlersignalisierung. Wird keine Überwachung gewünscht, muss RPT auf T#0s gesetzt werden.
LIFE	BYTE	Anzahl der zulässigen fehlerhaften Überwachungsaufrufe

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BYTE	0 = nicht aktiv 1 = Daten wurden empfangen 3 = Fehler-Signalisierung: Innerhalb des Zeitfensters (LIFE * RPT) wurde nichts empfangen
DEVICE	BYTE	Geräteadresse des Absenders
LEN	WORD	Anzahl der empfangenen Bytes

8.9.3 Funktion J1939_x_TRANSMIT

x = Nr. 1...n der CAN-Schnittstelle (je nach Gerät, → Datenblatt)

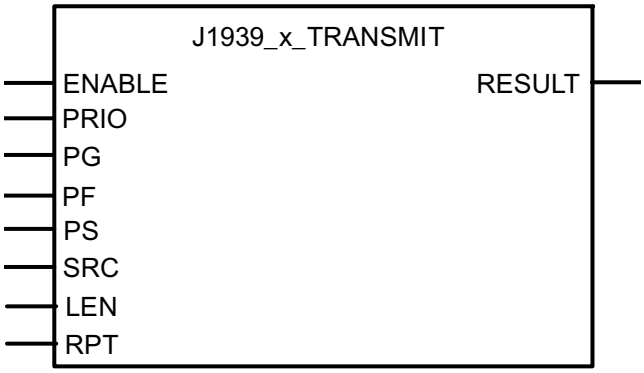
Enthalten in Bibliothek:

`ifm_J1939_x_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0303
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

J1939_x_TRANSMIT dient dem Senden von Nachrichten.

Die Funktion J1939_x_TRANSMIT ist für das Versenden einzelner Nachrichten oder Nachrichtenblocks verantwortlich. Dazu werden der Funktion die Parameter PG, PF, PS, RPT und die Adresse des Datenarrays SRC übergeben. Die Adresse muss über die Funktion ADR ermittelt werden. Zusätzlich muss die Anzahl der zu übertragenen Datenbytes und die Priorität (typisch 3, 6 oder 7) übergeben werden.

Da das Versenden der Daten über mehrerer Steuerungszyklen abgewickelt wird, muss der Vorgang über das RESULT-Byte ausgewertet werden. Wird RESULT = 1, wurden alle Daten übertragen.

Info

Wenn mehr als 8 Bytes gesendet werden sollen, wird ein "multi package transfer" durchgeführt.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet. FALSE: Funktion wird nicht ausgeführt
PRIO	BYTE	Nachrichtenpriorität (0…7)
PG	BYTE	Page address (normalerweise = 0)
PF	BYTE	PDU Format Byte
PS	BYTE	PDU Specific Byte
SRC	DWORD	Speicheradresse des Datenarrays, dessen Inhalt übertragen werden soll
LEN	WORD	Anzahl der zu sendenden Bytes
RPT	TIME	Wiederholzeit, innerhalb der die Daten-Telegramme zyklisch versendet werden

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BYTE	0 = nicht aktiv 1 = Datenübertragung beendet 2 = Funktion aktiv (Datenübertragung) 3 = Fehler, Daten können nicht gesendet werden

8.9.4 Funktion J1939_x_RESPONSE

x = Nr. 1...n der CAN-Schnittstelle (je nach Gerät, → Datenblatt)

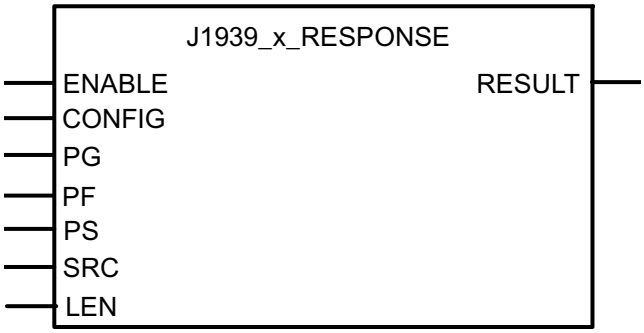
Enthalten in Bibliothek:

`ifm_J1939_x_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0303
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

J1939_x_RESPONSE organisiert die automatische Antwort auf ein Request-Telegramm (Anforderungstelegramm).

Diese Funktion ist für das automatische Versenden von Nachrichten auf "Global Requests" und "Specific Requests" verantwortlich. Dazu muss die Funktion über den Eingang CONFIG für einen Zyklus initialisiert werden.

Der Funktion werden die Parameter PG, PF, PS, RPT und die Adresse des Datenarrays SRC übergeben. Die Adresse muss über die Funktion ADR ermittelt werden. Zusätzlich wird die Anzahl der zu übertragenen Datenbytes übergeben.

Der Ausgang LEN zeigt an, wie viele Datenbytes empfangen wurden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet. FALSE: Funktion wird nicht ausgeführt
CONFIG	BOOL	TRUE (nur 1 Zyklus lang): zur Konfiguration des Datenobjektes FALSE: im weiteren Programmablauf
PG	BYTE	Page address (normalerweise = 0)
PF	BYTE	PDU Format Byte
PS	BYTE	PDU Specific Byte
SRC	DWORD	Speicheradresse des Datenarrays, dessen Inhalt übertragen werden soll
LEN	WORD	Anzahl der zu sendenden Bytes

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BYTE	0 = nicht aktiv 1 = Datenübertragung beendet 2 = Funktion aktiv (Datenübertragung) 3 = Fehler, Daten können nicht gesendet werden

8.9.5 Funktion J1939_x_SPECIFIC_REQUEST

x = Nr. 1...n der CAN-Schnittstelle (je nach Gerät, → Datenblatt)

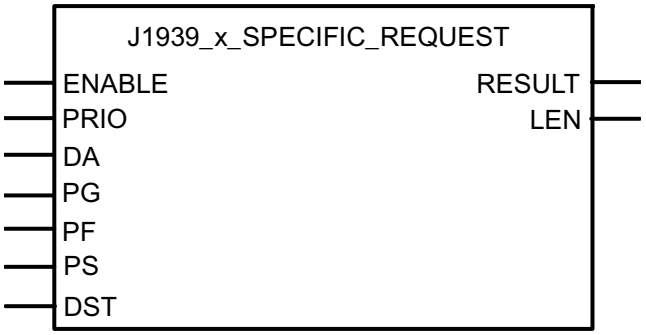
Enthalten in Bibliothek:

`ifm_J1939_x_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0303
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

J1939_x_SPECIFIC_REQUEST organisiert das Anfordern und Empfangen von Daten eines bestimmten Netzwerkteilnehmers.

Der Funktionsblock ist für das automatische Anfordern einzelner Nachrichten von einem bestimmten (specific) J1939-Netzwerkteilnehmer verantwortlich. Dazu werden der Funktion die logische Geräteadresse DA, die Parameter PG, PF, PS und die Adresse des Arrays DST übergeben, in dem die empfangenen Daten abgelegt werden. Die Adresse muss über die Funktion ADR ermittelt werden. Zusätzlich muss die Priorität (typisch 3, 6 oder 7) übergeben werden.

Da das Anfordern der Daten über mehrere Steuerungszyklen abgewickelt werden kann, muss dieser Vorgang über das RESULT-Byte ausgewertet werden. Wird RESULT = 1, wurden alle Daten empfangen.

Der Ausgang LEN zeigt an, wie viele Datenbytes empfangen wurden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet. FALSE: Funktion wird nicht ausgeführt
PRIO	BYTE	Priorität (0…7)
DA	BYTE	Logische Adresse (Zieladresse) des angeforderten Gerätes
PG	BYTE	Page address (normalerweise = 0)
PF	BYTE	PDU Format Byte
PS	BYTE	PDU Specific Byte
DST	DWORD	Zieladresse des Arrays, unter der die Empfangsdaten abgelegt werden

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BYTE	0 = nicht aktiv 1 = Datenübertragung beendet 2 = Funktion aktiv (Datenübertragung) 3 = Fehler, Daten können nicht gesendet werden
LEN	WORD	Anzahl der empfangenen Datenbytes

8.9.6 Funktion J1939_x_GLOBAL_REQUEST

x = Nr. 1...n der CAN-Schnittstelle (je nach Gerät, → Datenblatt)

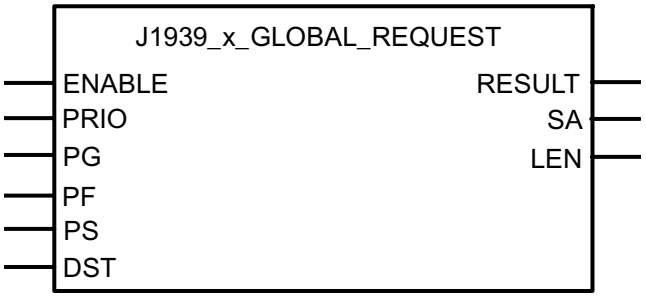
Enthalten in Bibliothek:

`ifm_J1939_x_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0303
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

J1939_x_GLOBAL_REQUEST organisiert globales Anfordern und Empfangen von Daten der Netzwerkteilnehmer.

Der Funktionsblock ist für das automatische Anfordern einzelner Nachrichten von allen (global) aktiven J1939-Netzwerkteilnehmern verantwortlich. Dazu werden der Funktion die logische Geräteadresse DA, die Parameter PG, PF, PS und die Adresse des Arrays DST übergeben, in dem die empfangenen Daten abgelegt werden. Die Adresse muss über die Funktion ADR ermittelt werden. Zusätzlich muss die Priorität (typisch 3, 6 oder 7) übergeben werden.

Da das Anfordern der Daten über mehrere Steuerungszyklen abgewickelt werden kann, muss dieser Vorgang über das RESULT-Byte ausgewertet werden. Wird RESULT = 1, wurden alle Daten empfangen.

Der Ausgang LEN zeigt an, wie viele Datenbytes empfangen wurden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet. FALSE: Funktion wird nicht ausgeführt
PRIO	BYTE	Priorität (0…7)
PG	BYTE	Page address (normalerweise = 0)
PF	BYTE	PDU Format Byte
PS	BYTE	PDU Specific Byte
DST	DWORD	Zieladresse des Arrays, unter der die Empfangsdaten abgelegt werden

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BYTE	0 = nicht aktiv 1 = Datenübertragung beendet 2 = Funktion aktiv (Datenübertragung) 3 = Fehler, Daten können nicht gesendet werden
SA	BYTE	Logische Geräteadresse (Sendeadresse) des angeforderten Gerätes
LEN	WORD	Anzahl der empfangenen Datenbytes

9 PWM im ecomatmobil-Controller

Inhalt:

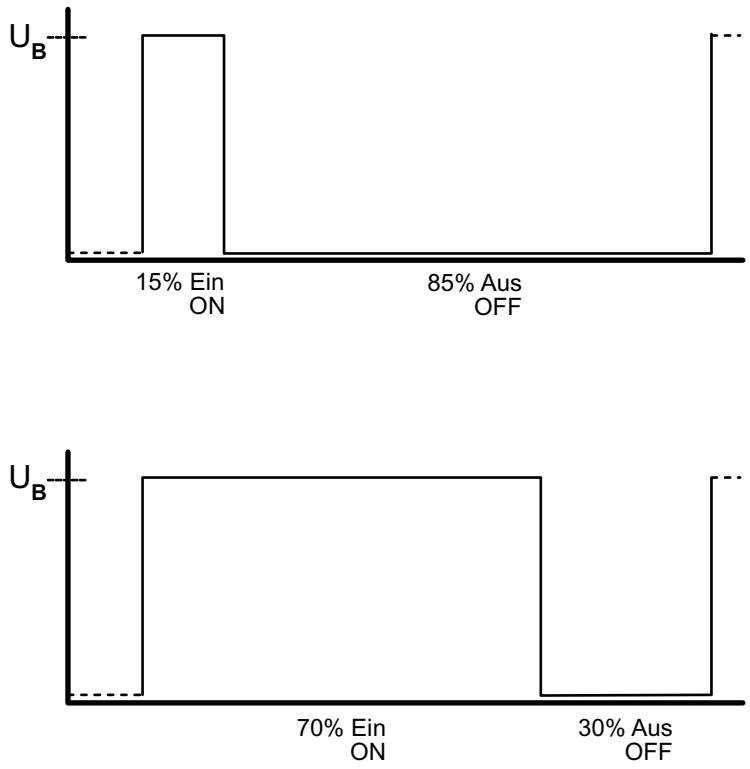
In diesem Kapitel erfahren Sie mehr über die Pulsweitenmodulation im Controller.

PWM ist in folgenden Controllern verfügbar:

	Anzahl verfügbare PWM-Ausgänge	davon Anzahl stromgeregelte PWM-Ausgänge	PWM-Frequenz [Hz]
ClassicController: CR0032	16	16	2...2000
ClassicController: CR0020 / CR0505	12 / 8	8 / 8	25...250
ExtendedController: CR0232	32	32	2...2000
ExtendedController: CR0200	24	16	25...250
SmartController: CR2500	4	4	25...250
SafetyController CR7020 / CR7505 / CR7200 CR7021 / CR7506 / CR7201	12 / 8 / 24	8 / 8 / 16	25...250
CabinetController: CR0301 / CR0302 / CR0303	4 / 8 / 8	0 / 0 / 0	25...250
Platinensteuerung: CS0015	8	0	25...250
PDM360 smart: CR1071	4	0	25...250

9.1 PWM-Signalverarbeitung

PWM steht als Abkürzung für die **P**uls-**W**eiten-**M**odulation, zuweilen auch "Puls-Pausen-Modulation" (PPM) genannt. Sie wird im Bereich der Steuerungen für den mobilen und robusten Einsatz hauptsächlich zur Ansteuerung von proportionalen Ventilen (PWM-Ventilen) genutzt. Ferner kann durch eine entsprechende Zusatzbeschaltung eines PWM-Ausganges (Zubehör) aus dem pulsweitenmodulierten Ausgangssignal eine analoge Ausgangsspannung erzeugt werden.



Grafik: Prinzip PWM

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

Die PWM-Funktion der Controller ist eine Hardware-Funktion, die vom Prozessor zur Verfügung gestellt wird. Um die integrierten PWM-Ausgänge des Controllers zu nutzen, müssen diese im Applikations-Programm initialisiert und entsprechend dem gewünschten Ausgangssignal parametriert werden.

9.1.1 PWM-Funktionen und deren Parameter (allgemein)

PWM/PWM1000

Je nach Einsatzfall und gewünschter Auflösung kann bei der Applikations-Programmierung zwischen den Funktionen PWM und PWM1000 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 soll keine hohen Anforderungen an die Genauigkeit gestellt werden, kann die Funktion PWM1000 (→ Seite 174) eingesetzt werden. Bei dieser Funktion können 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 PWM1000) übergeben. Je nach R360-Controller unterscheiden sich die PWM-Ausgänge in ihrer Arbeits-, aber nicht in ihrer Wirkungsweise.

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 und/oder 4...7 oder 4...11) zählt dieser dann von $FFFF_{16}$ rückwärts bzw. von 0000_{16} aufwärts. Bei Erreichen eines übergebenen Vergleichswertes (VALUE) wird der Ausgang gesetzt. Mit Überlauf des Zählers (Zählerstandwechsel von 0000_{16} nach $FFFF_{16}$ oder von $FFFF_{16}$ nach 0000_{16}) wird der Ausgang wieder zurückgesetzt und der Vorgang neu gestartet.

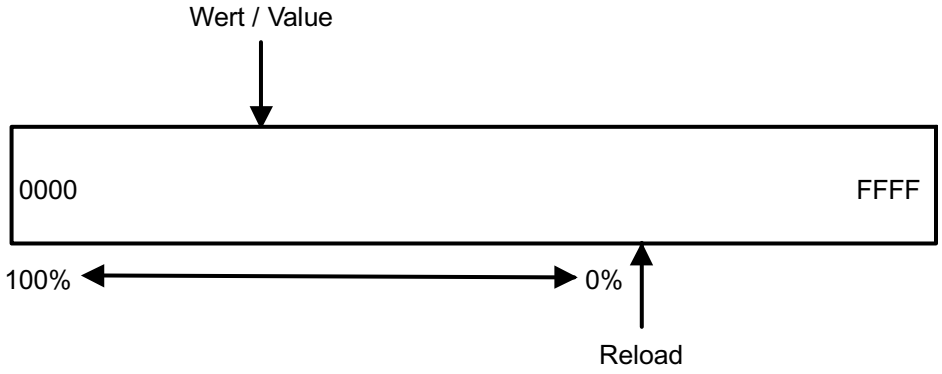
Soll dieser interne Zähler nicht zwischen 0000_{16} und $FFFF_{16}$ laufen, kann ein anderer Preset-Wert (RELOAD) für den internen Zähler übergeben werden. Dadurch steigt die PWM-Frequenz. Der Vergleichswert muss innerhalb des nun festgelegten Bereiches liegen.

PWM-Kanäle 0...3

Diese 4 PWM-Kanäle bieten die größte Flexibilität bei der Parametrierung. Die PWM-Kanäle 0...3 sind in allen Controller-Varianten vorhanden, je nach Geräteausführung mit oder ohne Stromregelung.

Für jeden Kanal kann eine eigene PWM-Frequenz (RELOAD-Wert) eingestellt werden. Zwischen der Funktion PWM (→ Seite 170) und der Funktion PWM1000 (→ Seite 174) kann frei gewählt werden.

Berechnung des RELOAD-Wertes



Grafik: RELOAD-Wert für PWM-Kanäle 0...3

Der RELOAD-Wert des internen PWM-Zählers berechnet sich in Abhängigkeit des Parameters DIV64 und der CPU-Frequenz wie folgt:

	ClassicController ExtendedController SafetyController CabinetController (CR0303)	**SmartController CabinetController (CR0301/CR0302) Platinensteuerung**
DIV64 = 0	RELOAD = 20 MHz / f_{PWM}	RELOAD = 10 MHz / f_{PWM}
DIV64 = 1	RELOAD = 312,5 kHz / f_{PWM}	RELOAD = 156,25 kHz / f_{PWM}

Je nachdem, ob eine hohe oder niedrige PWM-Frequenz benötigt wird, muss der Eingang DIV64 auf FALSE (0) oder TRUE (1) gesetzt werden. Bei PWM-Frequenzen unter 305 Hz oder 152 Hz (je nach Controller) muss DIV64 auf "1" gesetzt werden, damit der Reload-Wert nicht größer als $FFFF_{16}$ wird.

Berechnungsbeispiele RELOAD-Wert

ClassicController ExtendedController SafetyController CabinetController (CR0303)		**SmartController CabinetController (CR0301/CR0302) Platinensteuerung**
Die PWM-Frequenz soll 400 Hz betragen. 20 MHz _____ = 50000_{10} = $C350_{16}$ = RELOAD 400 Hz Der zulässige Bereich des PWM-Wertes ist damit der Bereich von 0000_{16} bis $C350_{16}$. Der Vergleichswert,bei dem der Ausgang durchschaltet, muss dann zwischen 0000_{16} und $C350_{16}$ liegen.		Die PWM-Frequenz soll 200 Hz betragen. 10 MHz _____ = 50000_{10} = $C350_{16}$ = RELOAD 200 Hz Der zulässige Bereich des PWM-Wertes ist damit der Bereich von 0000_{16} bis $C350_{16}$. Der Vergleichswert,bei dem der Ausgang durchschaltet, muss dann zwischen 0000_{16} und $C350_{16}$ liegen.

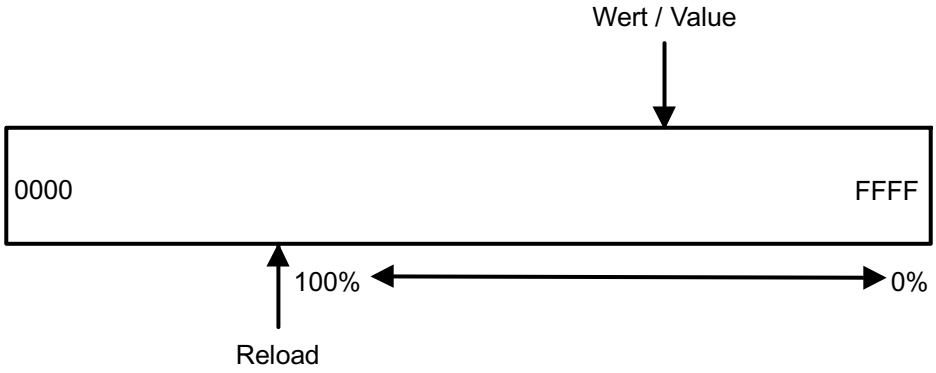
Daraus ergeben sich folgende Puls-Pausen-Verhältnisse:

Puls-Pausen-Verhältnis	Einschaltdauer	Wert für Puls-Pausen-Verhältnis
Minimal	0 %	$C350_{16}$
Maximal	100 %	0000_{16}

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

PWM-Kanäle 4...7 / 8...11

Diese 4/8 PWM-Kanäle können nur auf eine gemeinsame PWM-Frequenz eingestellt werden. Bei der Programmierung dürfen die Funktionen PWM und PWM1000 nicht gemischt eingesetzt werden.



Grafik: RELOAD-Wert für PWM-Kanäle 4...7 / 8...11

Der RELOAD-Wert des internen PWM-Zählers berechnet sich (für alle Controller) in Abhängigkeit des Parameters DIV64 und der CPU-Frequenz wie folgt:

DIV64 = 0	RELOAD = 10000_{16} – (2,5 MHz / f_{PWM})
DIV64 = 1	RELOAD = 10000_{16} – (312,5 kHz / f_{PWM})

Je nachdem, ob eine hohe oder niedrige PWM-Frequenz benötigt wird, muss der Eingang DIV64 auf FALSE (0) oder TRUE (1) gesetzt werden. Bei PWM-Frequenzen unter 39 Hz muss DIV64 auf "1" gesetzt werden, damit der RELOAD-Wert nicht kleiner als 0000_{16} wird.

Beispiel:

Die PWM-Frequenz soll 200 Hz betragen.

$$\frac{2{,}5\ \text{MHz}}{200\ \text{Hz}} = 12500_{10} = 30D4_{16}$$

RELOAD-Wert = 10000_{16} - $30D4_{16}$ = $CF2C_{16}$

Der zulässige Bereich des PWM-Wertes ist damit der Bereich von $CF2C_{16}$ bis $FFFF_{16}$

Der Vergleichswert, bei dem der Ausgang durchschaltet, muss dann zwischen $CF2C_{16}$ und $FFFF_{16}$ liegen.

HINWEIS

Die PWM-Frequenz ist für alle PWM-Ausgänge (4...7 oder 4...11) gleich.

Die Funktionen PWM und PWM1000 dürfen nicht gemischt werden.

Daraus ergeben sich folgende Puls-Pausen-Verhältnisse:

Puls-Pausen-Verhältnis	Einschaltdauer	Wert für Puls-Pausen-Verhältnis
Minimal	0 %	$FFFF_{16}$
Maximal	100 %	$CF2C_{16}$

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

HINWEIS

für ClassicController und ExtendedController gilt:

Werden die PWM-Ausgänge 4...7 eingesetzt (unabhängig ob stromgeregelt oder über einen der PWM-Funktionsblöcke), muss auch bei den Ausgängen 8...11 die gleiche Frequenz und der entsprechende Reload-Wert eingestellt werden. Daraus folgt: es müssen bei diesen Ausgängen die gleichen Funktionsblöcke eingesetzt werden.

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 festsetzen.

Um dieses Blockieren zu verhindern, wird der PWM-Wert in Abhängigkeit von der Dither-Frequenz um einen festgelegten Wert (DITHER_VALUE) vergrößert oder 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 * 2) der PWM-Frequenz angegeben.

Rampenfunktion

Soll der Wechsel von einem PWM-Wert zum nächsten nicht hart erfolgen, z.B. von 15 % Ein auf 70 % Ein (→ Grafik in PWM-Signalverarbeitung, Seite 164), kann z.B. durch Nutzung der Funktion PT1 ein verzögerter Anstieg realisiert werden. Die für PWM genutzte Rampenfunktion basiert auf der CoDeSys®-Bibliothek `UTIL.LIB`. Auf diese Weise können dann z.B. Hydrauliksysteme im Sanftanlauf betrieben werden.

HINWEIS

Beim Installieren der ecomat*mobil*-CD "Software, Tools and Documentation" wurden auch Projekte mit Beispielen auf Ihrem Computer im Programmverzeichnis abgelegt:
`…\ifm electronic\CoDeSys V…\Projects\DEMO_PLC_CDV…` (für Controller) oder
`…\ifm electronic\CoDeSys V…\Projects\DEMO_PDM_CDV…` (für PDMs)

Dort finden Sie auch Projekte mit Beispielen zu diesem Thema. Es wird dringend empfohlen, dem gezeigten Schema zu folgen.
→ Kapitel ifm-Demo-Programme, Seite 25

HINWEIS

Die PWM-Funktion der Controller ist eine vom Prozessor zur Verfügung gestellte Hardware-Funktion. Die PWM-Funktion bleibt solange gesetzt, bis am Controller ein Hardware-Reset (Aus- und Einschalten der Versorgungsspannung) durchgeführt wurde.

9.1.2 Funktion PWM

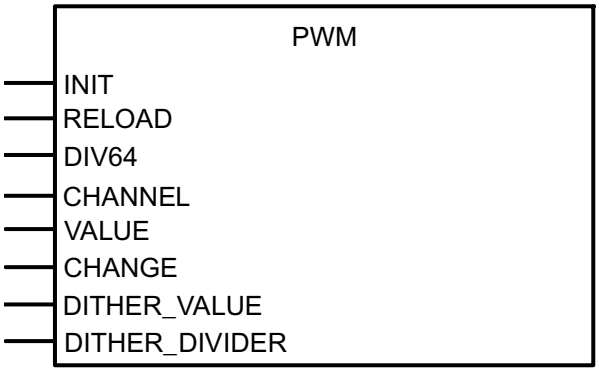
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015

Funktionssymbol:



Beschreibung

PWM wird zum Initialisieren und Parametrieren der PWM-Ausgänge genutzt.

Die Funktion 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 wird einmalig für jeden Kanal in der Initialisierung des Applikations-Programms aufgerufen. Dabei muss der Eingang INIT auf TRUE gesetzt sein. Bei der Initialisierung wird auch der Parameter RELOAD übergeben.

> **HINWEIS**
>
> Der Wert RELOAD muss für die Kanäle 4...7 (beim ClassicController oder ExtendedController: 4...11) gleich sein.
>
> Bei diesen Kanälen dürfen die Funktion PWM und die Funktion PWM1000 (→ Seite 174) nicht gemischt werden.
>
> Die PWM-Frequenz (und damit der RELAOD-Wert) ist intern auf 5 kHz begrenzt.

Je nachdem, ob eine hohe oder niedrige PWM-Frequenz benötigt wird, muss der Eingang DIV64 auf FALSE (0) oder TRUE (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.

Über die Funktion OUTPUT_CURRENT (→ Seite 181) kann eine Strommessung für den initialisierten PWM-Kanal realisiert werden

PWM_DITHER wird einmalig für jeden Kanal in der Initialisierung des Applikations-Programms 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.

Info

Die Parameter DITHER_FREQUENCY und DITHER_VALUE können für jeden Kanal individuell eingestellt werden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
INIT	BOOL	TRUE (im 1. Zyklus): Funktion PWM wird initialisiert FALSE: im zyklischen Programmablauf
RELOAD	WORD	Wert zur Festlegung der PWM-Frequenz (→ Kapitel Berechnung des RELOAD-Wertes, Seite 166)
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(→ Kapitel PWM-Dither, Seite 169)
DITHER_DIVIDER	WORD	Dither-Frequenz = PWM-Frequenz / DIVIDER * 2

9.1.3 Funktion PWM100

WICHTIG: Neue ecomat*mobil*-Controller unterstützen nur noch die Funktion PWM1000 (→ Seite 174).

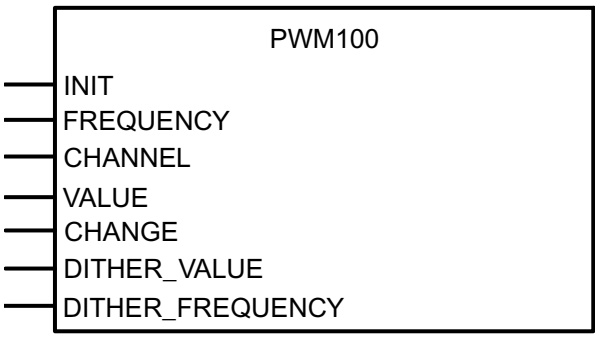
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7505, CR7200
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015

Funktionssymbol:



Beschreibung

PWM100 organisiert die Initialisierung und Parametrierung der PWM-Ausgänge.

Die Funktion ermöglicht eine einfache Anwendung der PWM-Funktion im R360-Controller. 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 relativ grobe Abstufung **nicht** geeignet.

Die Funktion wird einmalig für jeden Kanal in der Initialisierung des Applikations-Programms aufgerufen. Dabei muss der Eingang INIT auf TRUE gesetzt sein. Bei der Initialisierung wird auch der Parameter FREQUENCY übergeben.

HINWEIS

Der Wert FREQUENCY muss für die Kanäle 4...7 (beim ClassicController oder ExtendedController: 4...11) gleich sein.

Bei diesen Kanälen dürfen die Funktion PWM (→ Seite 170) und die Funktion PWM100 nicht gemischt werden.

Die PWM-Frequenz ist intern auf 5 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.

Eine Strommessung für den initialisierten PWM-Kanal kann realisiert werden:

- über die Funktion OUTPUT_CURRENT (→ Seite 181)
- oder z.B. mit ifm-Gerät EC 2049 (Vorschaltgerät zur Strommessung).

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

Info

Die Parameter DITHER_FREQUENCY und DITHER_VALUE können für jeden Kanal individuell eingestellt werden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
INIT	BOOL	TRUE (im 1. Zyklus): 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]

9.1.4 Funktion PWM1000

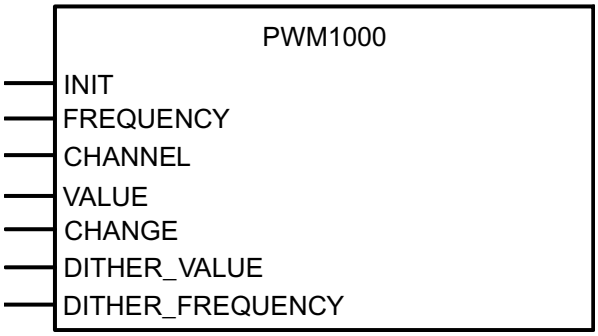
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015

Funktionssymbol:



Beschreibung

PWM1000 organisiert die Initialisierung und Parametrierung der PWM-Ausgänge.

Die Funktion ermöglicht eine einfache Anwendung der PWM-Funktion im R360-Controller. Die PWM-Frequenz kann direkt in [Hz] und das Puls-Pausen-Verhältnis in 1 ‰-Schritten angegeben werden.

Die Funktion wird einmalig für jeden Kanal in der Initialisierung des Applikations-Programms aufgerufen. Dabei muss der Eingang INIT auf TRUE gesetzt sein. Bei der Initialisierung wird auch der Parameter FREQUENCY übergeben.

HINWEIS

Der Wert FREQUENCY muss für die Kanäle 4...7 (beim ClassicController oder ExtendedController: 4...11) gleich sein.

Bei diesen Kanälen dürfen die Funktion PWM (→ Seite 170) und die Funktion PWM1000 nicht gemischt werden.

Die PWM-Frequenz ist intern auf 5 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.

Eine Strommessung für den initialisierten PWM-Kanal kann realisiert werden:

- über die Funktion OUTPUT_CURRENT (→ Seite 181)
- oder z.B. mit ifm-Gerät EC 2049 (Vorschaltgerät zur Strommessung).

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

Info

Die Parameter DITHER_FREQUENCY und DITHER_VALUE können für jeden Kanal individuell eingestellt werden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
INIT	BOOL	TRUE (im 1. Zyklus): PWM1000 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 [%]
DITHER_FREQUENCY	WORD	Dither-Frequenz in [Hz]

9.2 Stromregelung mit PWM

Inhalt:

Dieses Gerät der R360-Controllerfamilie kann den tatsächlich fließenden Strom an bestimmten Ausgängen messen und das Signal zur Weiterverarbeitung nutzen. **ifm electronic** stellt dem Anwender einige Funktionen zu diesem Zweck bereit.

9.2.1 Strommessung bei PWM-Kanälen

Über die im Controller integrierten Strommesskanäle kann eine Strommessung des Spulenstroms durchgeführt werden. Dadurch kann zum Beispiel der Strom bei einer Spulenerwärmung nachgeregelt werden. Damit bleiben die Hydraulikverhältnisse im System gleich.

ACHTUNG

Überlastschutz bei ClassicController und ExtendedController:

Grundsätzlich sind die stromgeregelten Ausgänge gegen Kurzschluss geschützt. Im Überlastfall, bei dem die Ströme z.B. zwischen 8 A und 20 A durch Leitungslängen und -querschnitte begrenzt sind, werden die Messwiderstände (Shunts) thermisch überlastet.

► Da der maximal zulässige Strom nicht jeweils vorgegeben werden kann, sollte im Applikations-Programm der Betriebsmodus OUT_OVERLOAD_PROTECTION für die Ausgänge gewählt werden. Bei Strömen > 4,1 A wird der betroffene Ausgang automatisch abgeschaltet.

> Ist der Ausgang nicht mehr überlastet, wird der Ausgang selbsttätig wieder eingeschaltet.

Die Funktion OUT_OVERLOAD_PROTECTION ist im reinen PWM-Modus (ohne Stromregelung) nicht aktiv!

HINWEIS

für ClassicController und ExtendedController gilt:

Die Stromregelblöcke (Funktion OCC_TASK (→ Seite 179) und Funktion OUTPUT_CURRENT_CONTROL (→ Seite 177)) basieren auf der Funktion PWM (→ Seite 170). Werden die Stromregelblöcke eingesetzt, darf bei den Kanälen 8...11 nur der PWM-Funktionsblock genutzt werden. Es ist der zur Frequenz entsprechende RELOAD-Wert zu berechnen.

9.2.2 Funktion OUTPUT_CURRENT_CONTROL

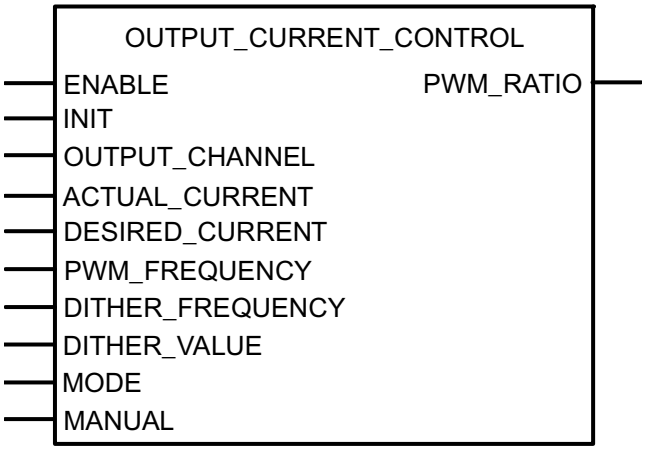
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201

Funktionssymbol:



Beschreibung

OUTPUT_CURRENT_CONTROL arbeitet als Stromregler für die PWM-Ausgänge.

Der Regler ist als adaptiver Regler konzipiert, so dass dieser selbstoptimierend arbeitet. Ist das selbstoptimierende Verhalten nicht gewünscht, kann über den Eingang MANUAL ein Wert > 0 übergeben werden; damit wird das selbstoptimierende Verhalten deaktiviert. Der Zahlenwert repräsentiert einen Korrekturwert, der u.a eine Auswirkung auf den I- und D-Anteil des Reglers hat. Zur Ermittlung der besten Einstellung des Reglers im MANUAL-Modus, bietet sich der Wert 50 an. Je nach gewünschtem Reglerverhalten kann der Wert dann schrittweise vergrößert (Regler wird schärfer / schneller) oder verkleinert (Regler wird schwächer / langsamer) werden.

Ist der Funktionseingang MANUAL auf "0" gesetzt, arbeitet der Regler immer selbstoptimierend. Das Verhalten der Regelstrecke wird ständig überwacht und die aktualisierten Korrekturwerte werden automatisch in jedem Zyklus dauerhaft gespeichert. Veränderungen in der Regelstrecke werden somit sofort erkannt und korrigiert.

HINWEIS

Um einen stabilen Ausgangswert zu bekommen, sollte die Funktion OUTPUT_CURRENT_CONTROL zyklisch in gleichmäßigen Zeitabständen aufgerufen werden.
Wird eine genaue Zykluszeit (5 ms) benötigt: → Funktion OCC_TASK (Seite 179) einsetzen.

OUTPUT_CURRENT_CONTROL basiert auf der Funktion PWM (→ Seite 170).

Wird OUTPUT_CURRENT_CONTROL für die Ausgänge 4...7 genutzt, darf bei gleichzeitiger Verwendung der PWM-Ausgänge 8...11 auch dort nur der PWM-Funktionsblock eingesetzt werden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht ausgeführt
INIT	BOOL	TRUE (nur im 1. Zyklus): Funktion wird initialisiert FALSE: Im Programmablauf
OUTPUT_CHANNEL	BYTE	PWM-Ausgangskanal (0...x: Werte abhängig vom Gerät)
ACTUAL_CURRENT	WORD	Aktueller Strom des PWM-Ausgangs in [mA], hierzu muss die Funktion OUTPUT_CURRENT (→ Seite 181) aufgerufen werden. Der Ausgangswert von OUTPUT_CURRENT wird hierzu dem Eingang von ACTUAL CURRENT zugeführt.
DESIRED_CURRENT	WORD	Stromsollwert in [mA]
PWM_FREQUENCY	WORD	Zulässige PWM-Frequenz für die am Ausgang angeschlossene Last
DITHER_FREQUENCY	WORD	Dither-Frequenz in [Hz]
DITHER_VALUE	BYTE	Amplitude des Dither-Wertes in [%]
MODE	BYTE	Reglercharakteristik: 0 = sehr langsamer Anstieg, kein Überschwingen 1 = langsamer Anstieg, kein Überschwingen 2 = minimales Überschwingen 3 = mäßiges Überschwingen zulässig
MANUAL	BYTE	Wenn Wert > 0, dann wird das selbstoptimierende Verhalten des Reglers überschrieben (typ. Wert: 50)

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
PWM_RATIO	BYTE	Zu Kontrollzwecken: Anzeige PWM-Tastverhältnis 0...100%

9.2.3 Funktion OCC_TASK

(Für SafetyController NICHT verfügbar)

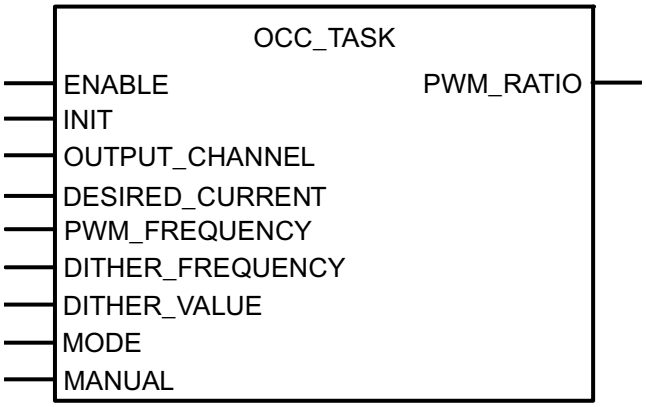
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500

Funktionssymbol:



Beschreibung

OCC_TASK arbeitet als Stromregler für die PWM-Ausgänge.

Der Regler ist als adaptiver Regler konzipiert, so dass dieser selbstoptimierend arbeitet. Ist das selbstoptimierende Verhalten nicht gewünscht, kann über den Eingang MANUAL ein Wert > 0 (selbstoptimierende Verhalten wird deaktiviert) übergeben werden. Der Zahlenwert repräsentiert einen Korrekturwert, der u.a eine Auswirkung auf den I- und D-Anteil des Reglers hat. Zur Ermittlung der besten Einstellung des Reglers im MANUAL-Modus, bietet sich der Wert 50 an. Je nach gewünschtem Reglerverhalten kann der Wert dann schrittweise vergrößert (Regler wird schärfer / schneller) oder verkleinert (Regler wird schwächer / langsamer) werden.

Ist der Funktionseingang MANUAL auf "0" gesetzt, arbeitet der Regler immer selbstoptimierend. Das Verhalten der Regelstrecke wird ständig überwacht und die aktualisierten Korrekturwerte werden automatisch in jedem Zyklus dauerhaft gespeichert. Veränderungen in der Regelstrecke werden somit sofort erkannt und korrigiert.

HINWEIS

OCC_TASK arbeitet mit einer festen Zykluszeit von 5 ms. Es müssen auch keine Istwerte zugeführt werden, da diese schon funktionsintern erfasst werden.

OCC_TASK basiert auf der Funktion PWM (→ Seite 170).

Wird OUTPUT_CURRENT_CONTROL für die Ausgänge 4...7 genutzt, darf bei gleichzeitiger Verwendung der PWM-Ausgänge 8...11 auch dort nur der PWM-Funktionsblock eingesetzt werden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht ausgeführt
INIT	BOOL	TRUE (im 1. Zyklus): Funktion wird initialisiert FALSE: im Programmablauf
OUTPUT_CHANNEL	BYTE	PWM-Ausgangskanal (0...x: Werte abhängig vom Gerät)
DESIRED_CURRENT	WORD	Stromsollwert in [mA]
PWM_FREQUENCY	WORD	Zulässige PWM-Frequenz für die am Ausgang angeschlossene Last
DITHER_FREQUENCY	WORD	Dither-Frequenz in [Hz]
DITHER_VALUE	BYTE	Amplitude des Dither-Wertes in [%]
MODE	BYTE	Reglercharakteristik: 0 = sehr langsamer Anstieg, kein Überschwingen 1 = langsamer Anstieg, kein Überschwingen 2 = minimales Überschwingen 3 = mäßiges Überschwingen zulässig
MANUAL	BYTE	Wenn Wert > 0, dann wird das selbstoptimierende Verhalten des Reglers überschrieben (typ. Wert: 50)

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
PWM_RATIO	BYTE	Zu Kontrollzwecken: Anzeige PWM-Tastverhältnis 0...100%

9.2.4 Funktion OUTPUT_CURRENT

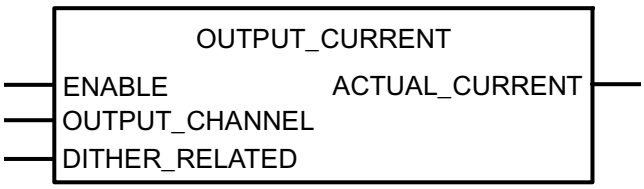
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201

Funktionssymbol:



Beschreibung

OUTPUT_CURRENT dient dem stabilen Messen des Stroms (Mittelung über Dither-Periode) an einem Ausgangskanal.

Die Funktion liefert den aktuellen Ausgangsstrom, wenn die Ausgänge als PWM-Ausgänge benutzt werden. Die Strommessung erfolgt innerhalb des Gerätes, es werden also keine externen Messwiderstände benötigt.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet. FALSE: Funktion wird nicht ausgeführt
OUTPUT_CHANNEL	BYTE	PWM-Ausgangskanal (0...x: Werte abhängig vom Gerät)
DITHER_RELATED	BOOL	Strom wird ermittelt als Mittelwert über... TRUE: eine Dither-Periode. FALSE: eine PWM-Periode.

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
ACTUAL_CURRENT	WORD	Ausgangsstrom in [mA].

9.3 Hydraulikregelung mit PWM

Inhalt:

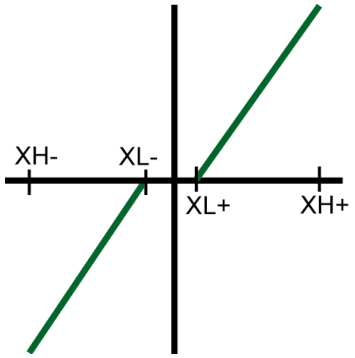
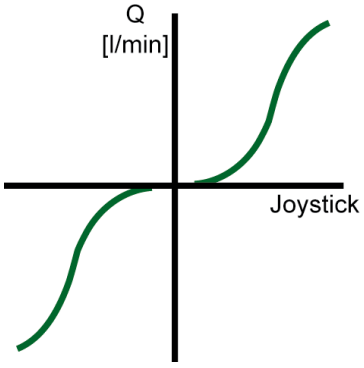
Als Spezialgebiet der Stromregelung mit PWM bietet **ifm electronic** dem Anwender spezielle Funktionen zur Regelung von Hydrauliksystemen.

9.3.1 Wozu diese Bibliothek? – Eine Einführung

Mit den Funktionen dieser Bibliothek können Sie folgende Aufgaben erfüllen:

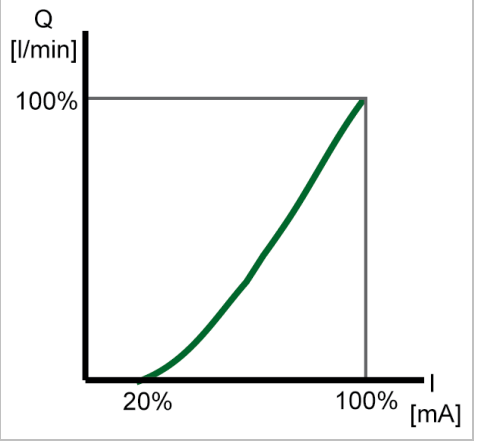
Ausgangssignale von Joysticks normieren

Nicht immer will man, dass sich der volle Bewegungsbereich des Joysticks auf die Maschinenbewegung auswirkt.

Oft soll der Bereich um die Neutralstellung des Joysticks herum ausgespart werden, weil der Joystick in der Neutralstellung nicht sicher 0 V liefert. Hier im Bild soll der Bereich zwischen XL- und XL+ ausgespart bleiben.	
Die FBs dieser Bibliothek ermöglichen Ihnen, die Kennlinie Ihres Joysticks nach Ihrem Bedarf anzupassen – auf Wunsch sogar frei parametrierbar:	

Hydraulikventile mit stromgeregelten Ausgängen ansteuern

Hydraulikventile haben in der Regel keine völlig lineare Kennlinie:

Typischer Kennlinienverlauf eines Hydraulikventils: Erst bei ca. 20 % des Spulenstroms beginnt der Ölfluss. Der Ölfluss ist anfänglich nicht linear. Dies muss bei der Berechnung der Sollwerte für den Spulenstrom berücksichtigt werden. Die FBs aus dieser Bibliothek unterstützen Sie hierbei.	

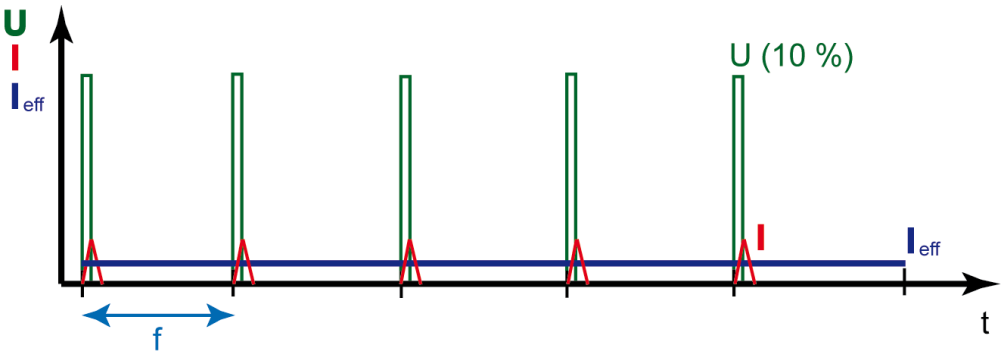
9.3.2 Was macht ein PWM-Ausgang?

PWM steht für "**P**uls-**W**eiten-**M**odulation" und meint folgendes Prinzip:

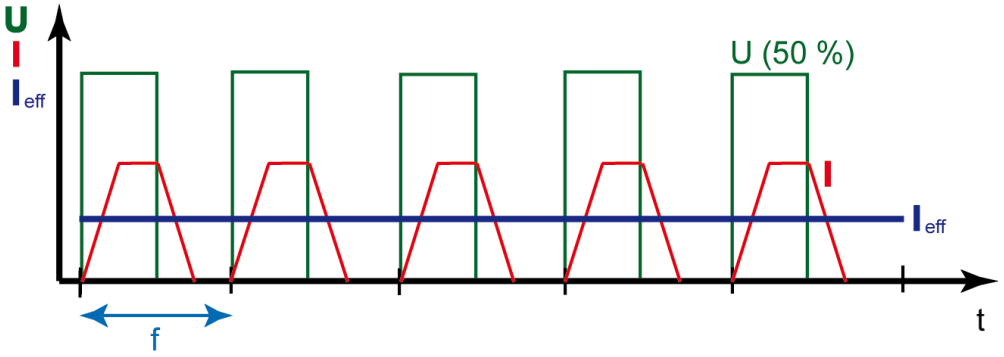
Digitale Ausgänge liefern in der Regel eine feste Ausgangsspannung, sobald sie eingeschaltet sind. Der Wert der Ausgangsspannung lässt sich hier **nicht** verändern. PWM-Ausgänge dagegen zerlegen die Spannung in eine schnelle Folge von vielen Rechteck-Impulsen. Das Verhältnis der Impulsdauer "eingeschaltet" zur Impulsdauer "ausgeschaltet" bestimmt den Effektivwert der gewünschten Ausgangsspannung. Man spricht dann von der Einschaltdauer (in Prozent).

Info

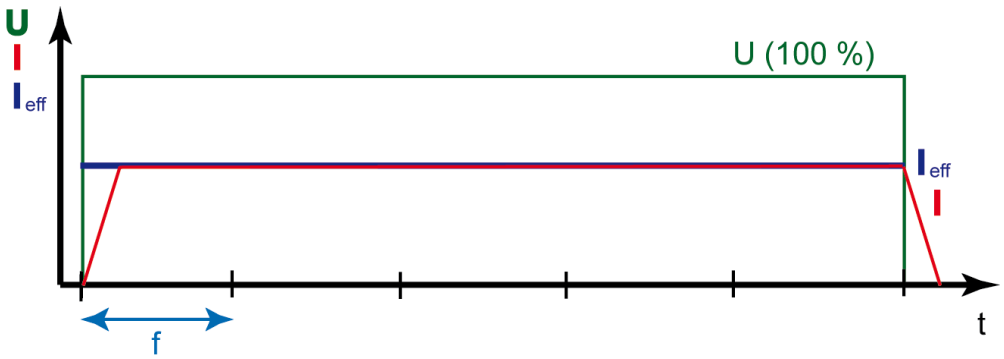
In den folgenden Skizzen sind die Strom-Verläufe nur stilisiert als Gerade dargestellt. Tatsächlich verläuft der Strom nach einer e-Funktion.



Grafik: Verlauf von PWM-Spannung U und Spulenstrom I bei 10 % Einschaltdauer:
Der effektive Spulenstrom I_{eff} beträgt ebenfalls 10 %



Grafik: Verlauf von PWM-Spannung U und Spulenstrom I bei 50 % Einschaltdauer:
Der effektive Spulenstrom I_{eff} beträgt ebenfalls 50 %



Grafik: Verlauf von PWM-Spannung U und Spulenstrom I bei 100 % Einschaltdauer:
Der effektive Spulenstrom I_{eff} beträgt ebenfalls 100 %

9.3.3 Was ist der Dither?

Wenn ein Proportional-Hydraulikventil angesteuert wird, bewegt sich sein Kolben nicht sofort los und anfangs auch nicht proportional zum Spulenstrom. Durch diesen "Slip-Stick-Effekt" – eine Art "Losbrechmoment" – benötigt das Ventil zu Anfang einen etwas höheren Strom, um die Kraft aufzubringen, den Kolben aus der Ruhelage zu bewegen. Das gleiche geschieht auch bei jeder anderen Positionsänderung des Ventilkolbens. Gerade bei sehr geringen Stellgeschwindigkeiten zeigt sich dieser Effekt in Form einer ruckenden Bewegung.

Diesem Problem begegnet die Technik, indem der Ventilkolben ständig einer kleinen Hin- und Herbewegung (dem Dither) unterworfen wird. Dabei vibriert der Kolben ständig hin und her und kann nicht "festkleben". Eine auch kleine Positionsänderung erfolgt nun ohne Verzögerung quasi als "fliegender Start".

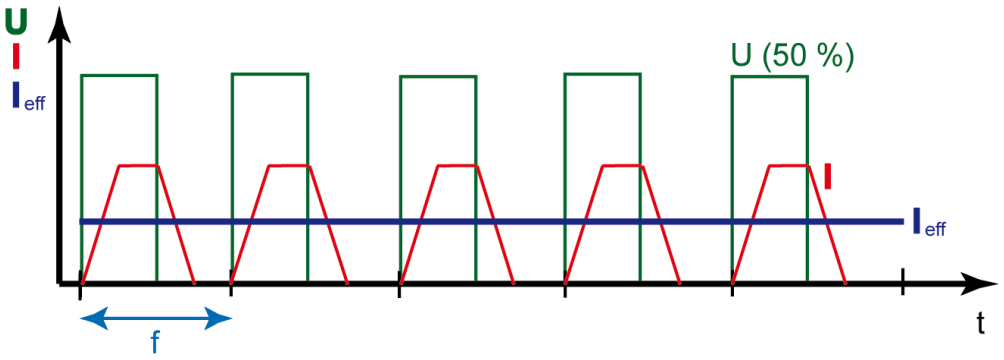
Vorteil: Der so angesteuerte Hydraulikzylinder kann wesentlich feinfühliger bewegt werden.

Nachteil: Das Ventil wird mit Dither messbar heißer als ohne, weil die Ventilspule nun dauernd arbeitet.

Es gilt also, eine "goldene Mitte" zu finden.

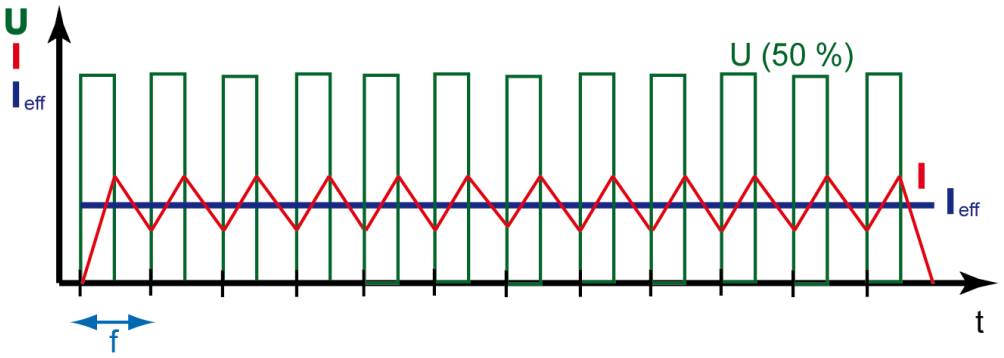
Wann ist ein Dither sinnvoll?

Wenn der PWM-Ausgang eine Puls-Frequenz ausgibt, die klein genug ist (Richtwert: bis 250 Hz), dass sich der Ventilkolben ständig mit einem Mindesthub bewegt, dann ist kein zusätzlicher Dither erforderlich (→ nächstes Bild):

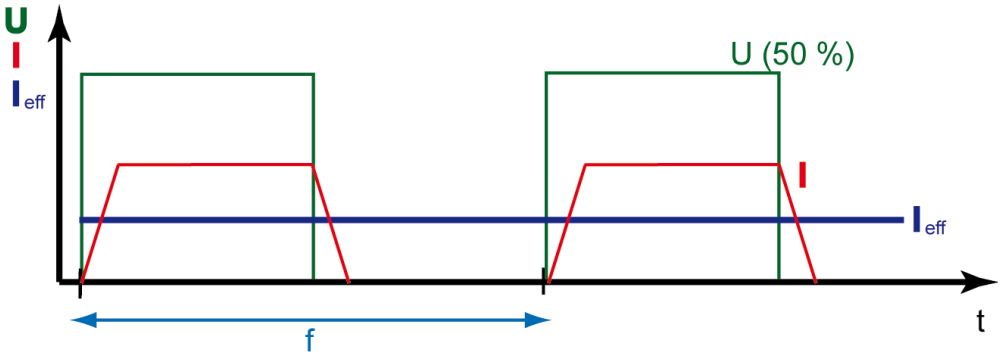


Grafik: Ausgewogenes PWM-Signal; kein Dither erforderlich.

Bei einer höheren PWM-Frequenz (Richtwert: 250 Hz bis 1 kHz) ist die Restbewegung des Ventilkolbens so kurz oder so langsam, dass dies effektiv als Stillstand resultiert, der Ventilkolben also wieder in der momentanen Position festkleben kann (und auch wird!) (→ nächste Grafiken):



Grafik: Hohe Frequenz des PWM-Signals führt annähernd zu einem resultierenden Gleichstrom in der Spule. Der Ventilkolben bewegt sich nicht mehr genug. Bei jeder Signaländerung muss der Ventilkolben erneut das Losbrechmoment überwinden.



Grafik: Zu niedrige Frequenz des PWM-Signals lässt nur seltene, ruckende Bewegungen des Ventilkolbens entstehen. Jeder Impuls bewegt den Ventilkolben erneut aus seiner Ruhelage; jedes Mal muss der Ventilkolben erneut das Losbrechmoment überwinden.

> **HINWEIS**
>
> Bei einer Einschaltdauer unter 10 % und größer 90 % hat der Dither keine messbare Auswirkung mehr. In solchen Fällen ist es sinnvoll und notwendig, dem PWM-Signal ein Dither-Signal zu überlagern.

Dither-Frequenz und -Amplitude

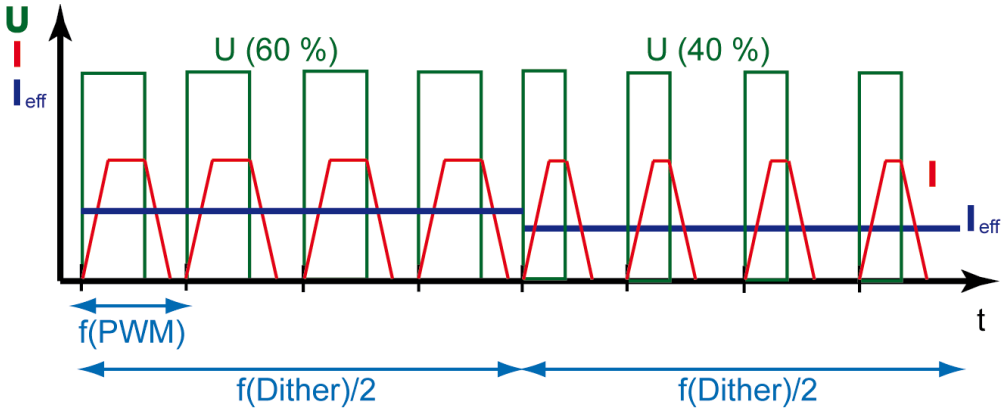
Das Puls-/Pausenverhältnis (die Einschaltdauer) des PWM-Ausgangssignals wird mit der Dither-Frequenz umgeschaltet. Die Dither-Amplitude bestimmt, wie groß der Unterschied der Einschaltdauer in den beiden Dither-Halbwellen ist.

> **HINWEIS**
>
> Die Dither-Frequenz muss ein ganzzahliger Teil der PWM-Frequenz sein. Andernfalls wird das hydraulische System nicht gleichförmig arbeiten, sondern schwingen.

Beispiel Dither

Die Dither-Frequenz beträgt den 8-ten Teil der PWM-Frequenz.
Die Dither-Amplitude beträgt 10 %.

Bei der im Bild anstehenden Einschaltdauer von 50 % wird die tatsächliche Einschaltdauer für 4 Impulse 60 % betragen und für die nächsten 4 Impulse 40 %, was im Mittel wieder 50 % Einschaltdauer ausmacht. Der resultierende effektive Spulenstrom wird 50 % des maximalen Spulenstroms betragen.



Im Ergebnis wird der Ventilkolben wieder um seine jeweilige Ruhelage schwingen, um bei der nächsten Signaländerung sofort seine neue Position annehmen zu können. ohne zuvor das Losbrechmoment überwinden zu müssen.

9.3.4 Bausteine der Bibliothek "ifm_HYDRAULIC_Vxxyyzz.Lib"

Die Bibliothek `ifm_HYDRAULIC_Vxxyyzz.Lib` enthält folgende Bausteine:

- Funktion CONTROL_OCC (→ Seite 188) *)
  Dieser FB nutzt die Funktion OUTPUT_CURRENT_CONTROL (→ Seite 177) und die Funktion OUTPUT_CURRENT (→ Seite 181) aus der Bibliothek `ifm_CRnnnn_Vxxyyzz.Lib`.
- Funktion CONTROL_OCC_TASK (→ Seite 191) *)
  Dieser FB nutzt die Funktion OCC_TASK (→ Seite 179) aus der Bibliothek `ifm_CRnnnn_Vxxyyzz.Lib`.
- Funktion JOYSTICK_0 (→ Seite 194)
- Funktion JOYSTICK_1 (→ Seite 197)
- Funktion JOYSTICK_2 (→ Seite 201)
- Funktion NORM_HYDRAULIC (→ Seite 205)

* OCC steht für **O**utput **C**urrent **C**ontrol (= stromgeregelter Ausgang)

Aus der Bibliothek `UTIL.Lib` (im CoDeSys®-Paket) werden folgende Bausteine benötigt:

- Funktion RAMP_INT
- Funktion CHARCURVE

Diese Bausteine werden von den FBs der `ifm_HYDRAULIC_Vxxyyzz.Lib` automatisch aufgerufen und parametriert.

Aus der Bibliothek `ifm_CRnnnn_Vxxyyzz.Lib` werden folgende Bausteine benötigt:

- Funktion OUTPUT_CURRENT (→ Seite 181)
- Funktion OUTPUT_CURRENT_CONTROL (→ Seite 177)
- Funktion OCC_TASK (→ Seite 179)

Diese Bausteine (→ Kapitel PWM-Signalverarbeitung, Seite 164) werden von den FBs der `ifm_HYDRAULIC_Vxxyyzz.Lib` automatisch aufgerufen und parametriert.

9.3.5 Funktion CONTROL_OCC

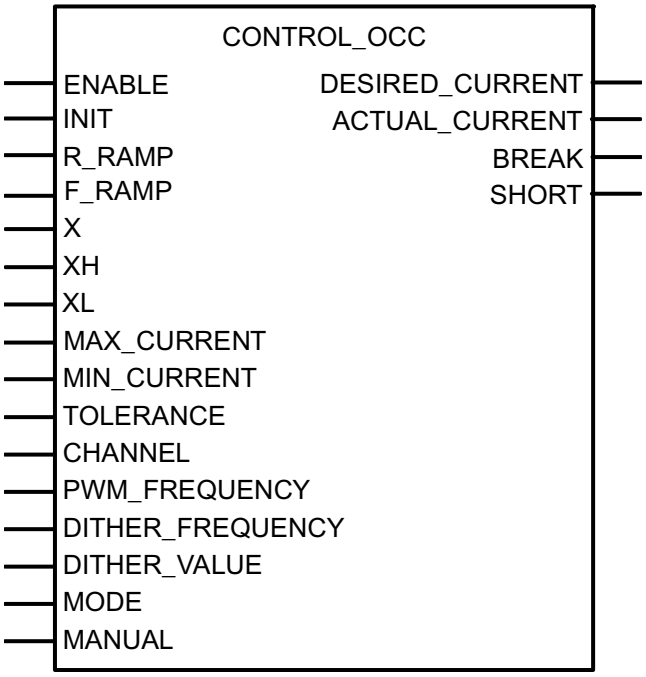
Enthalten in Bibliothek:

`ifm_HYDRAULIC_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201

Funktionssymbol:



Beschreibung

CONTROL_OCC skaliert den Eingangswert X auf einen angegebenen Strombereich.

Jede Instanz der Funktion wird in jedem SPS-Zyklus einmalig aufgerufen. Die Funktion nutzt die Funktion OUTPUT_CURRENT_CONTROL (→ Seite 177) und die Funktion OUTPUT_CURRENT (→ Seite 181) aus der Bibliothek `ifm_CRnnnn_Vxxyyzz.Lib`. Der Regler ist als adaptiver Regler konzipiert, so dass dieser selbstoptimierend arbeitet.

Ist das selbstoptimierende Verhalten nicht gewünscht, kann über den Eingang MANUAL ein Wert > 0 übergeben werden → das selbstoptimierende Verhalten wird deaktiviert.

Der Zahlenwert in MANUAL repräsentiert einen Korrekturwert, der u. a. eine Auswirkung auf den I- und den D-Anteil des Reglers hat. Zur Ermittlung der besten Einstellung des Reglers im MANUAL-Modus bietet sich der Wert 50 an.

Wert MANUAL vergrößern: → Regler wird schärfer / schneller
Wert MANUAL verkleinern: → Regler wird schwächer / langsamer

Ist der Funktionseingang MANUAL auf "0" gesetzt, arbeitet der Regler immer selbstoptimierend. Das Verhalten der Regelstrecke wird ständig überwacht und die aktualisierten Korrekturwerte werden automatisch in jedem Zyklus dauerhaft gespeichert. Veränderungen in der Regelstrecke werden somit sofort erkannt und korrigiert.

> **Info**
>
> Der Eingang X der Funktion CONTROL_OCC sollte von einem Ausgang der JOYSTICK-Funktionen gespeist werden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet. FALSE: Funktion wird nicht abgearbeitet
INIT	BOOL	TRUE: Funktion wird initialisiert, 1. Zyklus. FALSE: Im Programmablauf
R_RAMP	INT	Steigende Flanke der Rampe in [Inkremente/SPS-Zyklus] 0 = keine Rampe
F_RAMP	INT	Fallende Flanke der Rampe in [Inkremente/SPS-Zyklus] 0 = keine Rampe
X	WORD	Eingangswert in [Inkremente] normiert durch FB NORM_HYDRAULIC
XH	WORD	Max. Eingangswert in [Inkremente]
XL	WORD	Min. Eingangswert in [Inkremente]
MAX_CURRENT	WORD	Max. Ventilstrom in [mA]
MIN_CURRENT	WORD	Min. Ventilstrom in [mA]
TOLERANCE	BYTE	Toleranz für min. Ventilstrom in [mA] Bei Überschreiten der Toleranz erfolgt Sprung auf MIN_CURRENT
CHANNEL	BYTE	0...x PWM-Ausgangskanal (Werte abhängig vom Gerät)
PWM_FREQUENCY	WORD	PWM-Frequenz für das angeschlossene Ventil in [Hz]
DITHER_FREQUENCY	WORD	Dither-Frequenz in [Hz]
DITHER_VALUE	BYTE	Amplitude des Dither-Wertes in Prozent von MAX_CURRENT
MODE	BYTE	Reglercharakteristik: 0 = sehr langsamer Anstieg, kein Überschwingen 1 = langsamer Anstieg, kein Überschwingen 2 = minimales Überschwingen 3 = mäßiges Überschwingen zulässig
MANUAL	BYTE	Wert = 0: Regler arbeitet selbstoptimierend Wert > 0: Das selbstoptimierende Verhalten des Reglers wird überschrieben (typisch: 50)

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
DESIRED_CURRENT	WORD	Stromsollwert in [mA] für OCC (zu Kontrollzwecken)
ACTUAL_CURRENT	WORD	Aktueller Strom des PWM-Ausgangs in [mA] (zu Kontrollzwecken)
BREAK	BOOL	Fehler: Leitung zum Ventil unterbrochen
SHORT	BOOL	Fehler: Kurzschluss in Leitung zum Ventil

9.3.6 Funktion CONTROL_OCC_TASK

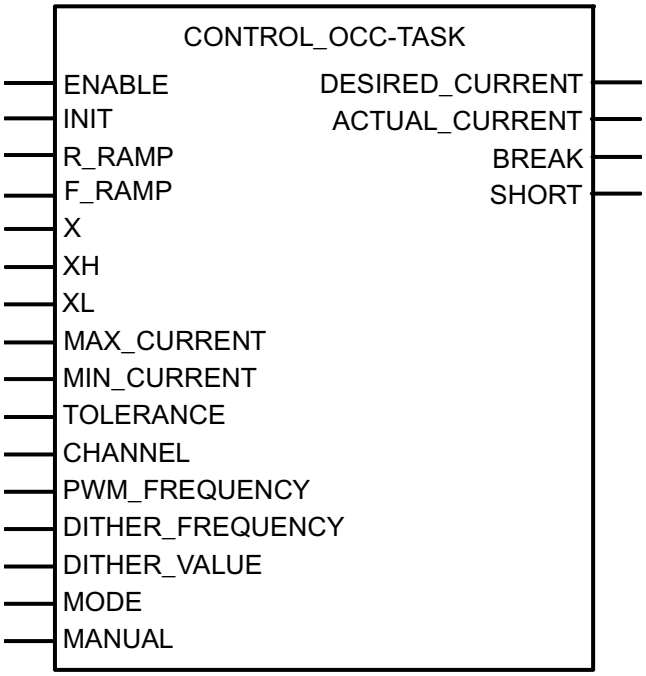
Enthalten in Bibliothek:

`ifm_HYDRAULIC_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201

Funktionssymbol:



Beschreibung

CONTROL_OCC_TASK skaliert den Eingangswert X auf einen angegebenen Strombereich.

HINWEIS

Jede Instanz der Funktion wird im Zyklus von 5 ms aufgerufen.

CONTROL_OCC_TASK soll daher sparsam verwendet werden: In CR0020 sind max. 6 Instanzen des FB sinnvoll (weitere Einschränkung beim Einsatz weiterer FBs mit hoher CPU-Last). Ansonsten ist die CPU-Last zu hoch. Folge: Die Zykluszeit wird zu lang.

Die Funktion nutzt die Funktion OCC_TASK (→ Seite 179) aus der Bibliothek `ifm_CRnnnn_Vxxyyzz.Lib`.

Der Regler ist als adaptiver Regler konzipiert, so dass dieser selbstoptimierend arbeitet. Ist das selbstoptimierende Verhalten nicht gewünscht, kann über den Eingang MANUAL ein Wert > 0 übergeben werden → das selbstoptimierende Verhalten wird deaktiviert.

Der Zahlenwert in MANUAL repräsentiert einen Korrekturwert, der u. a. eine Auswirkung auf den I- und den D-Anteil des Reglers hat. Zur Ermittlung der besten Einstellung des Reglers im MANUAL-Modus, bietet sich der Wert 50 an.

Wert MANUAL vergrößern: → Regler wird schärfer / schneller
Wert MANUAL verkleinern: → Regler wird schwächer / langsamer

Ist der Funktionseingang MANUAL auf "0" gesetzt, arbeitet der Regler immer selbstoptimierend. Das Verhalten der Regelstrecke wird ständig überwacht und die aktualisierten Korrekturwerte werden automatisch in jedem Zyklus dauerhaft gespeichert. Veränderungen in der Regelstrecke werden somit sofort erkannt und korrigiert.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet. FALSE: Funktion wird nicht abgearbeitet
INIT	BOOL	TRUE: Funktion wird initialisiert, 1. Zyklus. FALSE: Im Programmablauf
R_RAMP	INT	Steigende Flanke der Rampe in [Inkremente/SPS-Zyklus] 0 = keine Rampe
F_RAMP	INT	Fallende Flanke der Rampe in [Inkremente/SPS-Zyklus] 0 = keine Rampe
X	WORD	Eingangswert in [Inkremente] normiert durch FB NORM_HYDRAULIC
XH	WORD	Max. Eingangswert in [Inkremente]
XL	WORD	Min. Eingangswert in [Inkremente]
MAX_CURRENT	WORD	Max. Ventilstrom in [mA]
MIN_CURRENT	WORD	Min. Ventilstrom in [mA]
TOLERANCE	BYTE	Toleranz für min. Ventilstrom in [mA] Bei Überschreiten der Toleranz erfolgt Sprung auf MIN_CURRENT
CHANNEL	BYTE	0...x PWM-Ausgangskanal (Werte abhängig vom Gerät)
PWM_FREQUENCY	WORD	PWM-Frequenz für das angeschlossene Ventil in [Hz]
DITHER_FREQUENCY	WORD	Dither-Frequenz in [Hz]
DITHER_VALUE	BYTE	Amplitude des Dither-Wertes in Prozent von MAX_CURRENT
MODE	BYTE	Reglercharakteristik: 0 = sehr langsamer Anstieg, kein Überschwingen 1 = langsamer Anstieg, kein Überschwingen 2 = minimales Überschwingen 3 = mäßiges Überschwingen zulässig
MANUAL	BYTE	Wert = 0: Regler arbeitet selbstoptimierend Wert > 0: Das selbstoptimierende Verhalten des Reglers wird überschrieben (typisch: 50)

Parameter der Funktionsausgänge

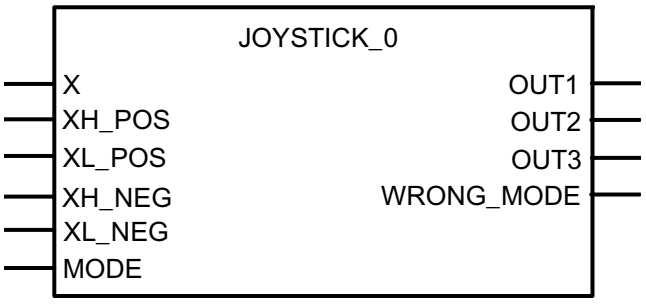
Name	Datentyp	Beschreibung
DESIRED_CURRENT	WORD	Stromsollwert in [mA] für OCC (zu Kontrollzwecken)
ACTUAL_CURRENT	WORD	Aktueller Strom des PWM-Ausgangs in [mA] (zu Kontrollzwecken)
BREAK	BOOL	Fehler: Leitung zum Ventil unterbrochen
SHORT	BOOL	Fehler: Kurzschluss in Leitung zum Ventil

9.3.7 Funktion JOYSTICK_0

Enthalten in Bibliothek:

ifm_HYDRAULIC_Vxxyyzz.LIB	ifm_HYDRAULIC32_Vxxyyzz.LIB
verfügbar für: • ClassicController: CR0020, CR0505 • ExtendedController: CR0200 • SmartController: CR2500 • SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201	verfügbar für: • ClassicController: CR0032 • ExtendedController: CR0232

Funktionssymbol:

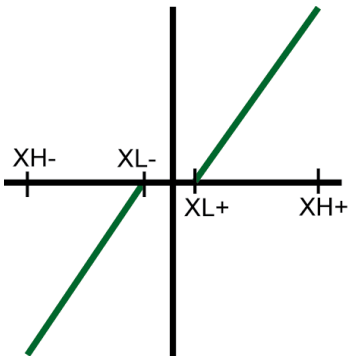


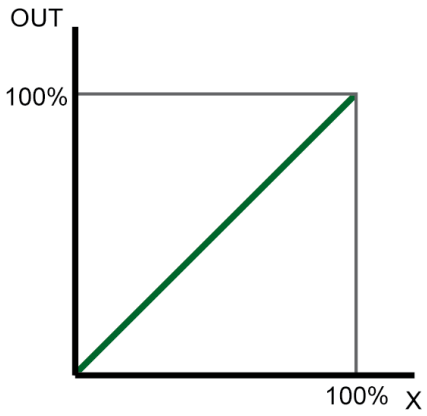
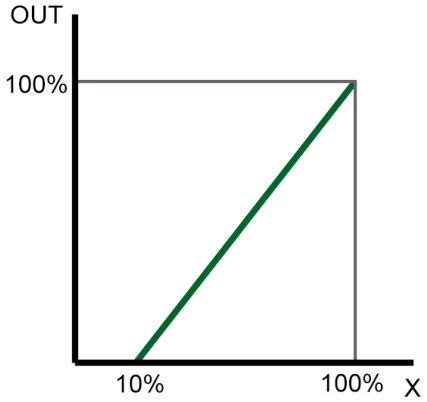
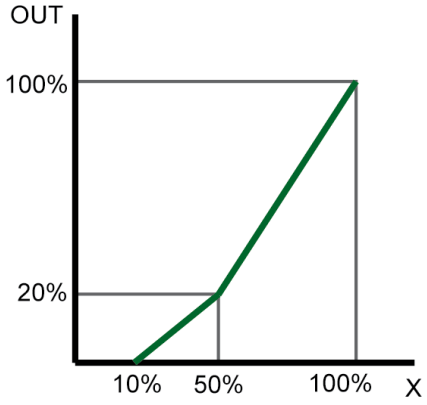
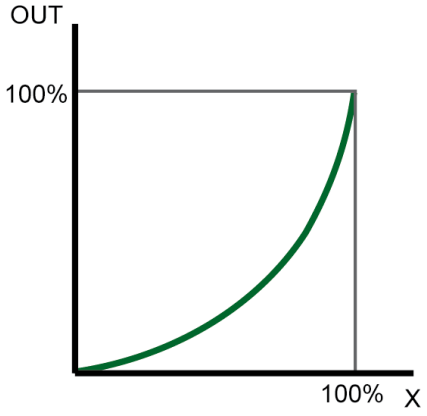
Beschreibung

JOYSTICK_0 skaliert Signale aus einem Joystick auf fest definierte Kennlinien, normiert auf 0…1000.

Bei dieser Funktion sind die Kennlinien-Werte fest vorgegeben (→ Grafiken):

- Steigende Flanke der Rampe = 5 Inkremente/SPS-Zyklus
- Fallende Flanke der Rampe = keine Flanke

Die Parameter XL_POS (XL+), XH_POS (XH+), XL_NEG (XL-) und XH_NEG (XH-) dienen dazu, die Joystickbewegung nur im erwünschten Bewegungsbereich auszuwerten. Die Werte für den positiven und den negativen Bereich dürfen sich unterscheiden. Die Werte für XL_NEG und XH_NEG sind hier negativ.	

Modus 0: Kennlinie linear für den Bereich XL bis XH	
Modus 1: Kennlinie linear mit Totbereich Werte fest eingestellt auf: Totbereich: 0…10% von 1000 Inkrementen	
Modus 2: Kennlinie 2-stufig linear mit Totbereich Werte fest eingestellt auf: Totbereich: 0…10% von 1000 Inkrementen Stufe: X = 50 % von 1000 Inkrementen Y = 20 % von 1000 Inkrementen	
Kennlinie Modus 3: Kurve ansteigend (Verlauf ist fest eingestellt)	

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
X	INT	Sollwert-Eingang in [Inkremente]
XH_POS	INT	Max. Sollwert positive Richtung in [Inkremente] (auch negative Werte zulässig)
XL_POS	INT	Min. Sollwert positive Richtung in [Inkremente] (auch negative Werte zulässig)
XH_NEG	INT	Max. Sollwert negative Richtung in [Inkremente] (auch negative Werte zulässig)
XL_NEG	INT	Min. Sollwert negative Richtung in [Inkremente] (auch negative Werte zulässig)
MODE	BYTE	Modus Auswahl Kennlinie: 0 = linear (0|0 – 1000|1000) 1 = linear mit Totbereich (0|0 – 100|0 – 1000|1000) 2 = 2-stufig linear mit Totbereich (0|0 – 100|0 – 500|200 – 1000|1000) 3 = Kurve ansteigend

Parameter der Funktionsausgänge

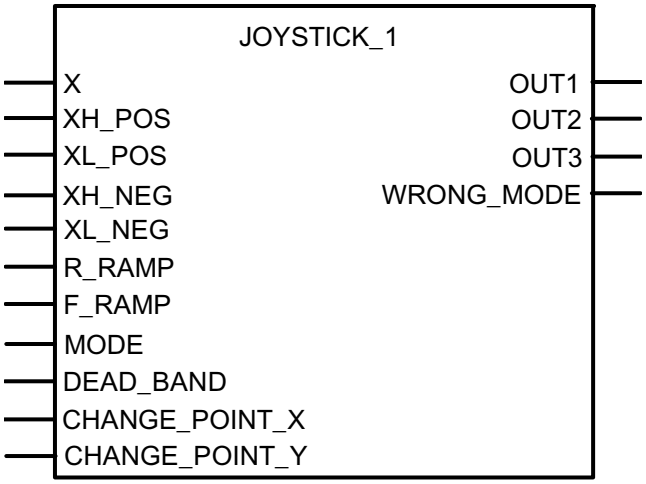
Name	Datentyp	Beschreibung
OUT1	WORD	normierter Ausgangswert: 0…1000 Inkremente z.B. für Ventil links
OUT2	WORD	normierter Ausgangswert: 0…1000 Inkremente z.B. für Ventil rechts
OUT3	INT	normierter Ausgangswert: -1000…0…1000 Inkremente z.B. für Ventil an Ausgangsmodul (z.B. CR2011 oder CR2031)
WRONG_MODE	BOOL	Fehler: Ungültiger Modus

9.3.8 Funktion JOYSTICK_1

Enthalten in Bibliothek:

ifm_HYDRAULIC_Vxxyyzz.LIB	ifm_HYDRAULIC32_Vxxyyzz.LIB
verfügbar für: • ClassicController: CR0020, CR0505 • ExtendedController: CR0200 • SmartController: CR2500 • SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201	verfügbar für: • ClassicController: CR0032 • ExtendedController: CR0232

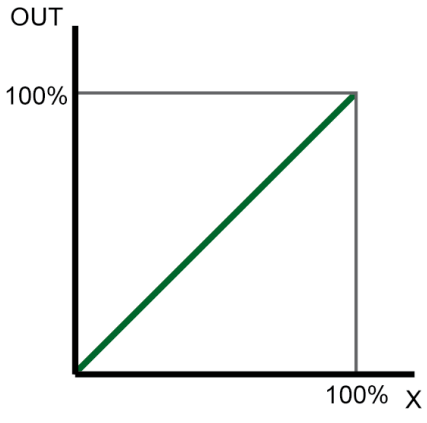
Funktionssymbol:

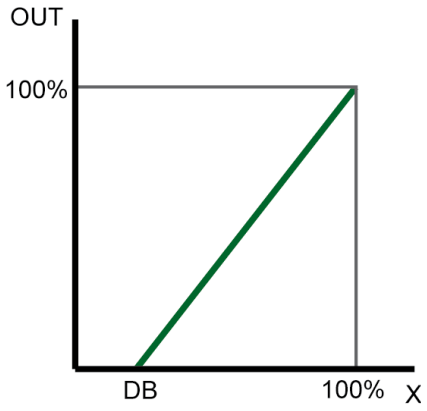
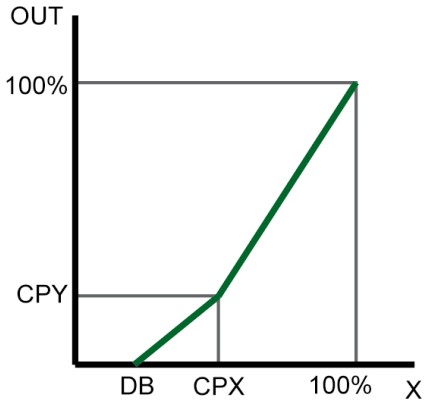
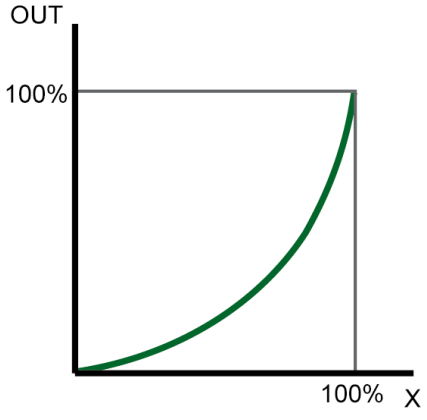


Beschreibung

JOYSTICK_1 skaliert Signale aus einem Joystick auf parametrierbare Kennlinien, normiert auf 0…1000.

Bei dieser Funktion sind die Kennlinien-Werte parametrierbar (→ Grafiken):

Modus 0: Kennlinie linear 100 % = 1000 Inkremente	

Modus 1: Kennlinie linear mit Totbereich Wert für Totbereich (DB) einstellbar in % von 1000 Inkrementen 100 % = 1000 Inkremente DB = Dead_Band	
Modus 2: Kennlinie 2-stufig linear mit Totbereich Werte parametrierbar auf: Totbereich: 0…DB in % von 1000 Inkrementen Stufe: X = CPX in % von 1000 Inkrementen Y= CPY in % von 1000 Inkrementen 100 % = 1000 Inkremente DB = Dead_Band CPX = Change_Point_X CPY = Change_Point_Y	
Kennlinie Modus 3: Kurve ansteigend (Verlauf ist fest eingestellt)	

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
X	INT	Sollwert-Eingang in [Inkremente]
XH_POS	INT	Max. Sollwert positive Richtung in [Inkremente] (auch negative Werte zulässig)
XL_POS	INT	Min. Sollwert positive Richtung in [Inkremente] (auch negative Werte zulässig)
XH_NEG	INT	Max. Sollwert negative Richtung in [Inkremente] (auch negative Werte zulässig)

Name	Datentyp	Beschreibung
XL_NEG	INT	Min. Sollwert negative Richtung in [Inkremente] (auch negative Werte zulässig)
R_RAMP	INT	Steigende Flanke der Rampe in [Inkremente/SPS-Zyklus] 0 = ohne Rampe
F_RAMP	INT	Fallende Flanke der Rampe in [Inkremente/SPS-Zyklus] 0 = ohne Rampe
MODE	BYTE	Modus Auswahl Kennlinie: 0 = linear (0|0 – 1000|1000) 1 = linear mit Totbereich (0|0 – DB|0 – 1000|1000) 2 = 2-stufig linear mit Totbereich (0|0 – DB|0 – CPX|CPY – 1000|1000) 3 = Kurve ansteigend
DEAD_BAND	BYTE	Einstellbarer Totbereich in [% von 1000 Inkrementen]
CHANGE_POINT_X	BYTE	Für Modus 2: Rampenstufe, Wert für X in [% von 1000 Inkrementen]
CHANGE_POINT_Y	BYTE	Für Modus 2: Rampenstufe, Wert für Y in [% von 1000 Inkrementen]

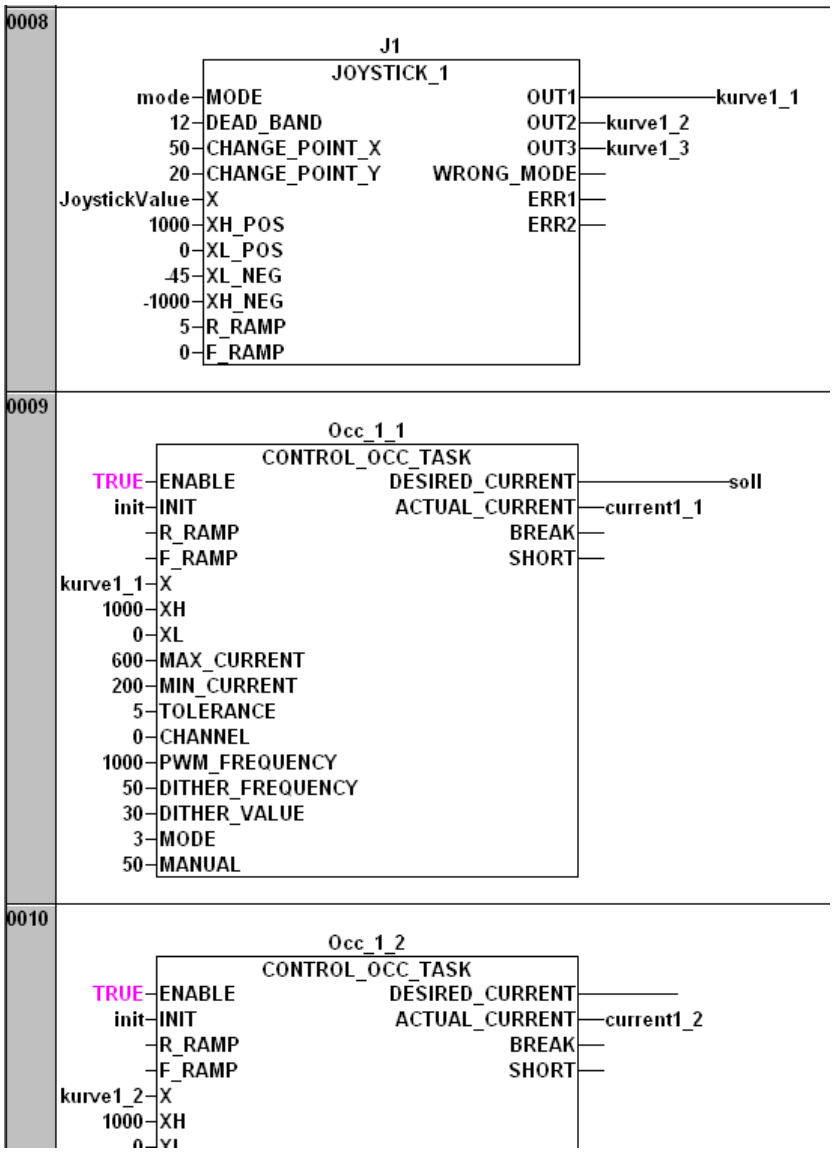
Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
OUT1	WORD	normierter Ausgangswert: 0…1000 Inkremente z.B. für Ventil links
OUT2	WORD	normierter Ausgangswert: 0…1000 Inkremente z.B. für Ventil rechts
OUT3	INT	normierter Ausgangswert: -1000…0…1000 Inkremente z.B. für Ventil an Ausgangsmodul (z.B. CR2011 oder CR2031)
WRONG_MODE	BOOL	Fehler: Ungültiger Modus

Beispiel JOYSTICK_1

Ein Joystick liefert -1000…0…1000 Inkremente. Der Bereich -45…0 ist als Nullpunkt ausgeblendet. Der Totbereich geht bis X = 120, der Umschaltpunkt liegt bei 50|20 % von 1000 Inkrementen.

Das Ausgangssignal wird getrennt für zwei Ventilspulen über jeweils einen Aufruf von Control_OCC_Task ausgegeben. Die Rampen sind in der Funktion Joystick_1 parametriert.



Die Beschaltungen der beiden Instanzen des FB CONTROL_OCC_TASK sind – bis auf das Eingangs- und das Ausgangssignal – identisch.

HINWEIS

Jede Instanz des FBs wird im Zyklus von 5 ms aufgerufen.

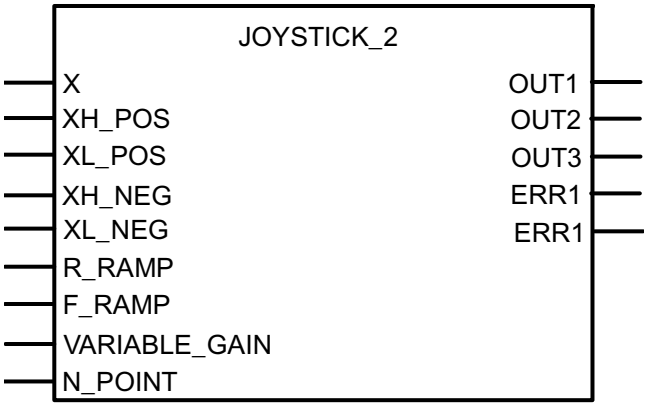
CONTROL_OCC_TASK soll daher sparsam verwendet werden: In CR0020 sind max. 6 Instanzen des FB sinnvoll (weitere Einschränkung beim Einsatz weiterer FBs mit hoher CPU-Last). Ansonsten ist die CPU-Last zu hoch. Folge: Die Zykluszeit wird zu lang.

9.3.9 Funktion JOYSTICK_2

Enthalten in Bibliothek:

ifm_HYDRAULIC_Vxxyyzz.LIB	ifm_HYDRAULIC32_Vxxyyzz.LIB
verfügbar für: • ClassicController: CR0020, CR0505 • ExtendedController: CR0200 • SmartController: CR2500 • SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201	verfügbar für: • ClassicController: CR0032 • ExtendedController: CR0232

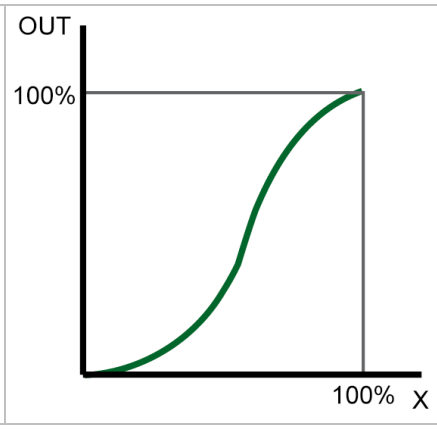
Funktionssymbol:



Beschreibung

JOYSTICK_2 skaliert Signale aus einem Joystick auf einen parametrierbaren Kennlinien-Verlauf. Die Normierung ist frei bestimmbar.

Bei dieser Funktion ist der Kennlinien-Verlauf frei parametrierbar (→ Grafik):

Kennlinie frei parametrierbar	

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
X	INT	Sollwert-Eingang in [Inkremente]
XH_POS	INT	Max. Sollwert positive Richtung in [Inkremente] (auch negative Werte zulässig)
XL_POS	INT	Min. Sollwert positive Richtung in [Inkremente] (auch negative Werte zulässig)
XH_NEG	INT	Max. Sollwert negative Richtung in [Inkremente] (auch negative Werte zulässig)
XL_NEG	INT	Min. Sollwert negative Richtung in [Inkremente] (auch negative Werte zulässig)
R_RAMP	INT	Steigende Flanke der Rampe in [Inkremente/SPS-Zyklus] 0 = ohne Rampe
F_RAMP	INT	Fallende Flanke der Rampe in [Inkremente/SPS-Zyklus] 0 = ohne Rampe
VARIABLE_GAIN	ARRAY [0..10] OF POINT	Wertepaare, die den Kurven-Verlauf beschreiben Es werden die ersten in N_POINT angegebenen Wertepaare verwertet. n = 2…11 Beispiel: 9 Wertepaare als Variable VALUES deklariert: VALUES : ARRAY[0..10] OF POINT := (X:=0,Y:=0),(X:=200,Y:=0), (X:=300,Y:=50), (X:=400,Y:=100), (X:=700,Y:=500), (X:=1000,Y:=900), (X:=1100,Y:=950), (X:=1200,Y:=1000), (X:=1400,Y:=1050); Zwischen den Werten dürfen auch Leerzeichen stehen.
N_POINT	BYTE	Anzahl der Punkte (Wertepaare in VARIABLE_GAIN), womit die Kurven-Charakteristik definiert ist: n = 2…11

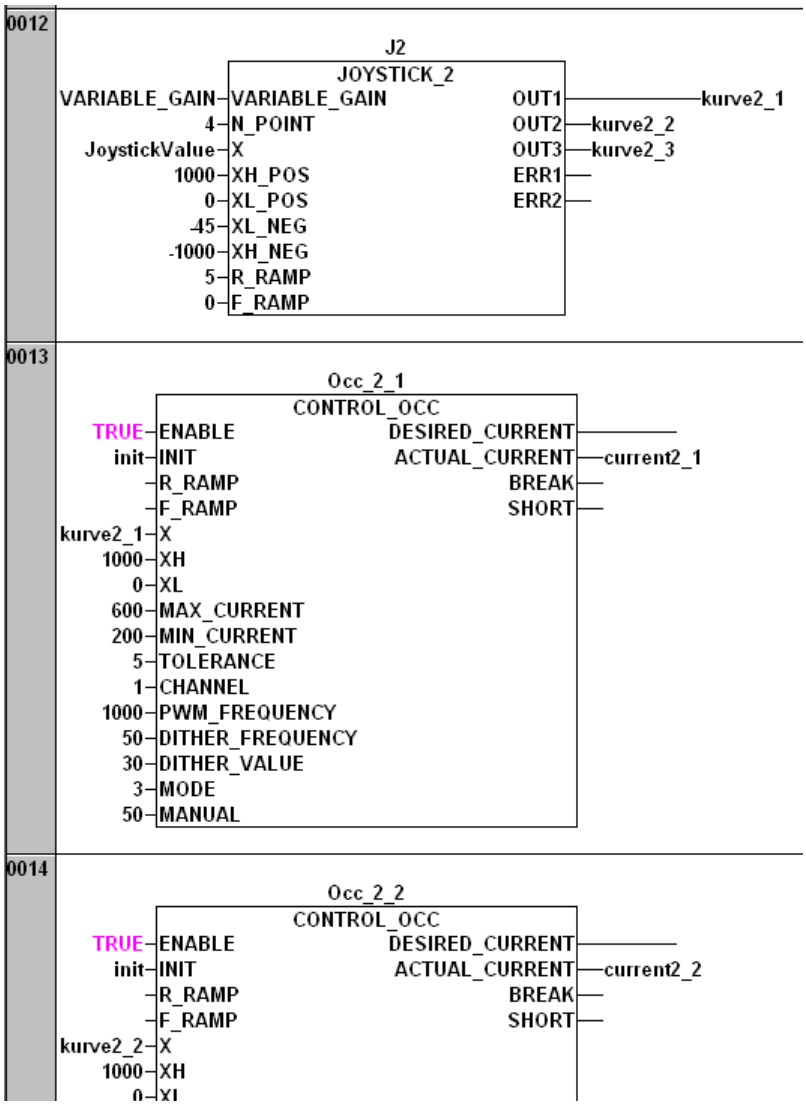
Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
OUT1	WORD	normierter Ausgangswert: Wertepaar 0 bis 10 [Inkremente] z.B. für Ventil links
OUT2	WORD	normierter Ausgangswert: Wertepaar 0 bis 10 [Inkremente] z.B. für Ventil rechts
OUT3	INT	normierter Ausgangswert: Wertepaar 0 bis 10 [Inkremente] z.B. für Ventil an Ausgangsmodul (z.B. CR2011 oder CR2031)
ERR1	BOOL	Fehlercode für OUT_1 / Ventil links: 0 = kein Fehler 1 = Fehler in Zahlenreihe: Falsche Reihenfolge 2 = Eingangswert IN nicht im Wertebereich der Zahlenreihe P enthalten 3 = Ungültige Anzahl N für Zahlenreihe P
ERR2	BOOL	Fehlercode für OUT_2 / Ventil rechts: 0 = kein Fehler 1 = Fehler in Zahlenreihe: Falsche Reihenfolge 2 = Eingangswert IN nicht im Wertebereich der Zahlenreihe P enthalten 3 = Ungültige Anzahl N für Zahlenreihe P

Beispiel JOYSTICK_2

Die Wertepaare für die Definition des Kennlinienverlaufes sind als Variable VARIABLE_GAIN deklariert (hier: 4 Wertepaare):

VARIABLE_GAIN:ARRAY[0..10] OF POINT := (X:=0,Y:=0),(X:=200,Y:=0),(X:=700,Y:=200), (X:=1000,Y:=1000);



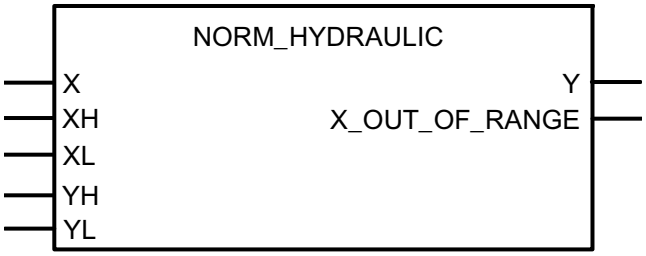
Die Beschaltungen der beiden Instanzen des FB CONTROL_OCC sind – bis auf das Eingangs- und das Ausgangssignal – identisch.

9.3.10 Funktion NORM_HYDRAULIC

Enthalten in Bibliothek:

ifm_HYDRAULIC_Vxxyyzz.LIB	ifm_HYDRAULIC32_Vxxyyzz.LIB
verfügbar für: • ClassicController: CR0020, CR0505 • ExtendedController: CR0200 • SmartController: CR2500 • SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201	verfügbar für: • ClassicController: CR0032 • ExtendedController: CR0232

Funktionssymbol:



Beschreibung

NORM_HYDRAULIC normiert Eingangswerte innerhalb festgesetzter Grenzen auf Werte mit neuen Grenzen.

Hinweis: Diese Funktion entspricht der 3S-Funktion NORM_DINT aus der CoDeSys®-Bibliothek `UTIL.Lib`.

Die Funktion normiert einen Wert vom Typ DINT, der innerhalb der Grenzen zwischen XH und XL liegt, auf einen Ausgangswert innerhalb der Grenzen zwischen YH und YL.

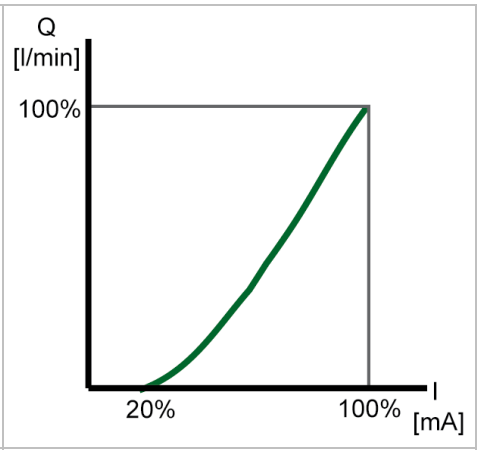
Bedingt durch Rundungsfehler können Abweichungen beim normierten Wert um 1 auftreten. Werden die Grenzen (XH/XL oder YH/YL) invertiert angegeben, erfolgt auch die Normierung invertiert.

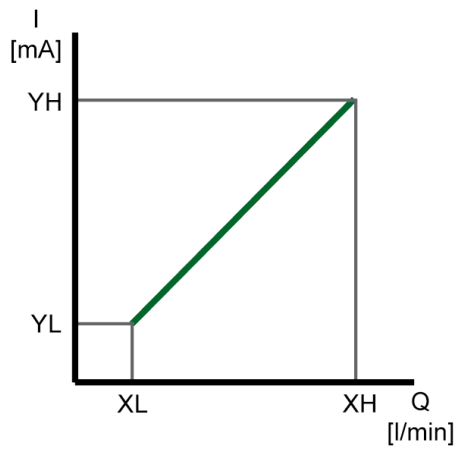
Wenn X außerhalb der Grenzen XL…XH liegt, wird die Fehlermeldung X_OUT_OF_RANGE = TRUE.

Typischer Kennlinienverlauf eines Hydraulikventils:

Erst bei ca. 20 % des Spulenstroms beginnt der Ölfluss.

Der Ölfluss ist anfänglich nicht linear.



Charakteristik der Funktion	

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
X	DINT	Sollwert-Eingang
XH	DINT	Max. Eingangswert [Inkremente]
XL	DINT	Min. Eingangswert [Inkremente]
YH	DINT	Max. Ausgangswert [Inkremente], z.B.: Ventilstrom [mA] / Durchfluss [l/min]
YL	DINT	Min. Ausgangswert [Inkremente], z.B.: Ventilstrom [mA] / Durchfluss [l/min]

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
Y	DINT	normierter Ausgangswert
X_OUT_OF_RANGE	BOOL	Fehler: X liegt außerhalb der Grenzen von XH und XL

Beispiele NORM_HYDRAULIC

Parameter	Fall 1	Fall 2	Fall 3
oberer Grenzwert Eingang XH	100	100	2000
unterer Grenzwert Eingang XL	0	0	0
oberer Grenzwert Ausgang YH	2000	0	100
unterer Grenzwert Ausgang YL	0	2000	0
nicht normierter Wert X	20	20	20
normierter Wert Y	400	1600	1

Fall 1: Eingang mit relativ grober Auflösung. Ausgang mit hoher Auflösung.
1 X-Inkrement ergibt 20 Y-Inkremente.

Fall 2: Eingang mit relativ grober Auflösung. Ausgang mit hoher Auflösung.
1 X-Inkrement ergibt 20 Y-Inkremente.
Ausgangssignal ist gegenüber dem Eingangssignal invertiert.

Fall 3: Eingang mit hoher Auflösung. Ausgang mit relativ grober Auflösung.
20 X-Inkremente ergeben 1 Y-Inkrement.

10 Weitere Funktionen im Controller

Inhalt:

In diesem Kapitel lernen Sie weitere Funktionen kennen, die Sie im Controller nutzen können.

10.1 Zählerfunktionen zur Frequenz- und Periodendauermessung

Inhalt:

Je nach Controller werden bis zu 16 schnelle Eingänge unterstützt, die Eingangsfrequenzen bis zu 30 kHz verarbeiten können. Neben der reinen Frequenzmessung an den Eingängen FRQ können die Eingänge ENC auch zur Auswertung von inkrementellen Drehgebern (Zählerfunktion) mit einer maximalen Frequenz von 10 kHz eingesetzt werden. Die Eingänge CYL werden zur Periodendauermessung von langsamen Signalen eingesetzt.

Eingang	Frequenz [kHz]	Erklärung
FRQ 0 / ENC 0	30 / 10	Frequenzmessung / Drehgeber 1, Kanal A
FRQ 1 / ENC 0	30 / 10	Frequenzmessung / Drehgeber 1, Kanal B
FRQ 2 / ENC 1	30 / 10	Frequenzmessung / Drehgeber 2, Kanal A
FRQ 3 / ENC 1	30 / 10	Frequenzmessung / Drehgeber 2, Kanal B
CYL 0 / ENC 2	10	Periodendauermessung / Drehgeber 3, Kanal A
CYL 1 / ENC 2	10	Periodendauermessung / Drehgeber 3, Kanal B
CYL 2 / ENC 3	10	Periodendauermessung / Drehgeber 4, Kanal A
CYL 3 / ENC 3	10	Periodendauermessung / Drehgeber 4, Kanal B

Zur einfachen Auswertung stehen folgende Funktionen zur Verfügung:

10.1.1 Einsatzfälle

Es ist zu beachten, dass – bedingt durch die unterschiedlichen Messmethoden – Fehler bei der Frequenzermittlung auftreten.

Die Funktion FREQUENCY (→ Seite 210) eignet sich für Frequenzen zwischen 100 Hz und 30 kHz, wobei der Fehler sich bei hohen Frequenzen verringert.

Die Funktion PERIOD (→ Seite 212) führt eine Periodendauermessung durch. Sie ist damit für Frequenzen kleiner 1000 Hz geeignet. Generell kann sie auch höhere Frequenzen messen. Dadurch wird aber die Zykluszeit stark belastet. Bei der Auslegung der Applikations-Software ist dies zu berücksichtigen.

10.1.2 Einsatz als Digitaleingänge

Werden die schnellen Eingänge (FRQx / CYLx) 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 50 Hz. Das Eingangssignal muss ggf. softwaretechnisch entprellt werden.

10.1.3 Funktion FREQUENCY

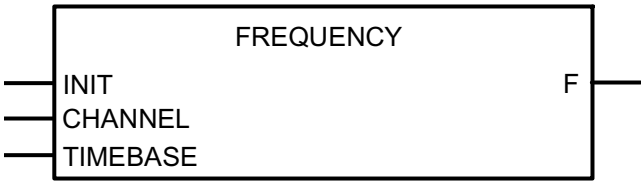
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
  (Für Sicherheitssignale zusätzlich Funktion SAFE_FREQUENCY_OK zusammen mit Funktion PERIOD (→ Seite 212) einsetzen!)
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1071

Funktionssymbol:



Beschreibung

FREQUENCY misst die anstehende Signalfrequenz am angegebenen Kanal. Maximale Eingangsfrequenz → Datenblatt.

Die Funktion misst die Frequenz des am gewählten Kanal (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.

HINWEIS

Für die Funktion FREQUENCY können nur die Eingänge FRQ0...FRQ3 genutzt werden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
INIT	BOOL	TRUE (nur 1 Zyklus lang): Funktion wird initialisiert FALSE: im zyklischen Programmablauf
CHANNEL	BYTE	Nummer des Eingangs (0...x Wert abhängig vom Gerät)
TIMEBASE	TIME	Zeitbasis

HINWEIS

Vor dem Initialisieren kann die Funktion falsche Werte ausgeben.

► Ausgang erst auswerten, wenn Funktion initialisiert wurde!

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
F	REAL	Frequenz in [Hz]

10.1.4 Funktion PERIOD

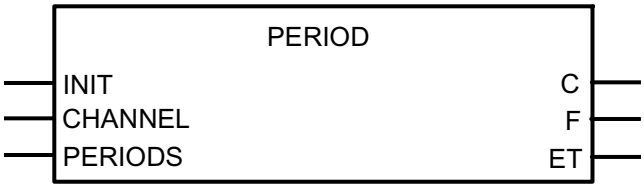
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
  (Für Sicherheitssignale zusätzlich Funktion SAFE_FREQUENCY_OK zusammen mit Funktion FREQUENCY (→ Seite 210) einsetzen!)
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1071

Funktionssymbol:



Beschreibung

PERIOD misst die Frequenz und die Periodendauer (Zykluszeit) in [µs] am angegebenen Kanal. Maximale Eingangsfrequenz → Datenblatt.

Die Funktion misst die Frequenz und die Zykluszeit des am gewählten Kanal (CHANNEL) anstehenden Signals. Zur Berechnung werden alle positiven Flanken ausgewertet und der Mittelwert über die Anzahl der angegebenen Perioden (PERIODS) gebildet.

Bei niedrigen Frequenzen kommt es mit der Funktion FREQUENCY zu Ungenauigkeiten. Um dieses zu umgehen, kann die Funktion PERIOD genutzt werden. Die Zykluszeit wird direkt in [µs] ausgegeben.

Der maximale Messbereich beträgt ca. 71 min.

HINWEIS

Für die Funktion PERIOD können nur die Eingänge CYL0...CYL3 genutzt werden.

Frequenzen < 0,5 Hz werden nicht mehr eindeutig angezeigt!

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
INIT	BOOL	TRUE (nur 1 Zyklus lang): Funktion wird initialisiert FALSE: im zyklischen Programmablauf
CHANNEL	BYTE	Nummer des Eingangs (0...x Wert abhängig vom Gerät)
PERIODS	BYTE	Anzahl der zu vergleichenden Perioden

HINWEIS

Vor dem Initialisieren kann die Funktion falsche Werte ausgeben. Ausgang erst auswerten, wenn Funktion initialisiert wurde.

Wir empfehlen dringend, alle benötigten Instanzen dieser Funktion zeitgleich zu initialisieren. Andernfalls können falsche Werte ausgegeben werden.

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
C	DWORD	Zykluszeit der erfassten Perioden in [µs]
F	REAL	Frequenz der erfassten Perioden in [Hz]
ET	TIME	Verstrichene Zeit seit Beginn der Periodendauermessung (nutzbar bei sehr langsamen Signalen)

10.1.5 Funktion PERIOD_RATIO

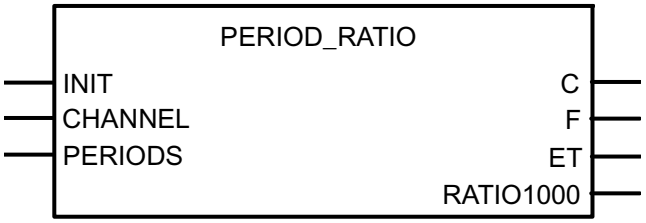
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1071

Funktionssymbol:



Beschreibung

PERIOD_RATIO misst die Frequenz und die Periodendauer (Zykluszeit) in µs über die angegebenen Perioden am angegebenen Kanal. Zusätzlich wird das Puls-/Pausenverhältnis in [‰] angegeben. Maximale Eingangsfrequenz → Datenblatt.

Die Funktion misst die Frequenz und die Zykluszeit des am gewählten Kanal (CHANNEL) anstehenden Signals. Zur Berechnung werden alle positiven Flanken ausgewertet und der Mittelwert über die Anzahl der angegebenen Perioden (PERIODS) gebildet. Zusätzlich wird das Puls-/Pausenverhältnis in [‰] angegeben.

Beispiel: Bei einem Signalverhältnis von 25 ms High-Pegel und 75 ms Low-Pegel, wird der Wert RATIO1000 von 250 ‰ ausgegeben.

Bei niedrigen Frequenzen kommt es mit der Funktion FREQUENCY zu Ungenauigkeiten. Um dieses zu umgehen, kann die Funktion PERIOD_RATIO genutzt werden. Die Zykluszeit wird direkt in [µs] ausgegeben.

Der maximale Messbereich beträgt ca. 71 min.

HINWEIS

Für die Funktion PERIOD_RATIO können nur die Eingänge CYL0...CYL3 genutzt werden.

Der Ausgang RATIO1000 liefert bei einen Puls/Pausenverhältnis von 100 % (Eingangssignal dauerhaft auf Versorgungsspannung) den Wert 0.

Frequenzen < 0,05 Hz werden nicht mehr eindeutig angezeigt!

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
INIT	BOOL	TRUE (nur 1 Zyklus lang): Funktion wird initialisiert FALSE: im zyklischen Programmablauf
CHANNEL	BYTE	Nummer des Eingangs (0...x Wert abhängig vom Gerät)
PERIODS	BYTE	Anzahl der zu vergleichenden Perioden

HINWEIS

Vor dem Initialisieren kann die Funktion falsche Werte ausgeben. Ausgang erst auswerten, wenn Funktion initialisiert wurde.

Wir empfehlen dringend, alle benötigten Instanzen dieser Funktion zeitgleich zu initialisieren. Andernfalls können falsche Werte ausgegeben werden.

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
C	DWORD	Zykluszeit der erfassten Perioden in [µs]
F	REAL	Frequenz der erfassten Perioden in [Hz]
ET	TIME	Verstrichene Zeit seit Beginn des letzten Zustandswechsels des Eingangssignals (nutzbar bei sehr langsamen Signalen)
RATIO1000	WORD	Puls-/Pause-Verhältnis in [‰]

10.1.6 Funktion PHASE

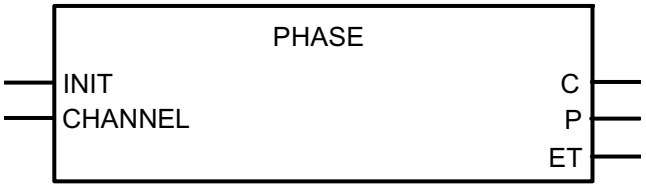
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1071

Funktionssymbol:



Beschreibung

PHASE liest ein Kanalpaar mit schnellen Eingängen ein und vergleicht die Phasenlage der Signale. Maximale Eingangsfrequenz → Datenblatt.

Diese Funktion fasst jeweils ein Kanalpaar mit schnellen Eingängen zusammen, so dass die Phasenlage zweier Signale zueinander ausgewertet werden kann. Es kann eine Periodendauer bis in den Sekundenbereich ausgewertet werden.

HINWEIS

Bei Frequenzen kleiner 15 Hz wird eine Periodendauer bzw. Phasenverschiebung von 0 angezeigt.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
INIT	BOOL	TRUE (nur 1 Zyklus lang): Funktion wird initialisiert FALSE: im Programmablauf
CHANNEL	BYTE	Kanalpaar 0 oder 1

HINWEIS

Vor dem Initialisieren kann die Funktion falsche Werte ausgeben. Ausgang erst auswerten, wenn Funktion initialisiert wurde.

Wir empfehlen dringend, für jeden Kanal, der ausgewertet werden soll, eine eigene Instanz dieser Funktion zu programmieren. Andernfalls können falsche Werte ausgegeben werden.

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
C	DWORD	Periodendauer in [µs]
P	INT	Winkel der Phasenverschiebung (0...360 °)
ET	TIME	Verstrichene Zeit seit Beginn der Periodendauermessung (nutzbar bei sehr langsamen Signalen)

10.1.7 Funktion INC_ENCODER

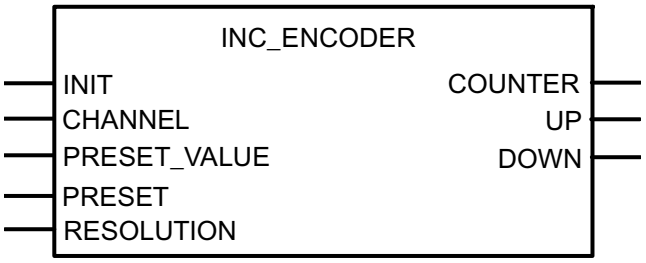
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015

Funktionssymbol:



Beschreibung

INC_ENCODER organisiert Vorwärts-/Rückwärts-Zählerfunktion zur Auswertung von Drehgebern.

Die Funktion ist als Vorwärts-/Rückwärtszähler ausgelegt. Immer zwei Frequenzeingänge bilden das Eingangspaar, das über die Funktion ausgewertet wird. In folgender Tabelle finden Sie die zulässigen Grenzfrequenzen und die max. anschließbaren inkrementalen Drehgeber:

Gerät	Grenzfrequenz	max. Anzahl Drehgeber
ClassicController: CR0020, CR0505	10 kHz	4
ClassicController: CR0032	30 kHz	8
ExtendedController: CR0200	10 kHz	8
ExtendedController: CR0232	30 kHz	16
SmartController: CR2500	10 kHz	2
SafetyController: CR7020, CR7505	10 kHz	4
ExtendedSafetyController: CR7200	10 kHz	8
SafetyController: CR7021, CR7506	10 kHz	4
ExtendedSafetyController: CR7201	10 kHz	8
CabinetController: CR0301, CR0302, CR0303	10 kHz	2
Platinensteuerung: CS0015	0,5 kHz	2
PDM360 smart: CR1071	1 kHz	2

HINWEIS

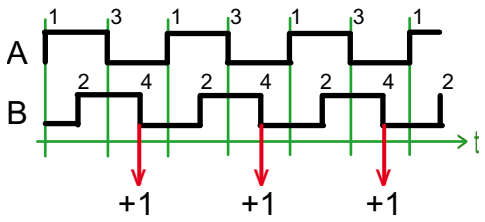
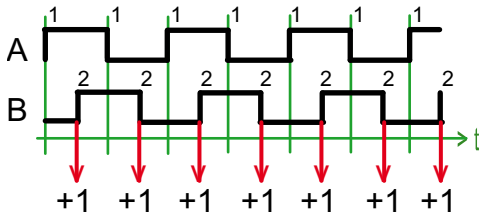
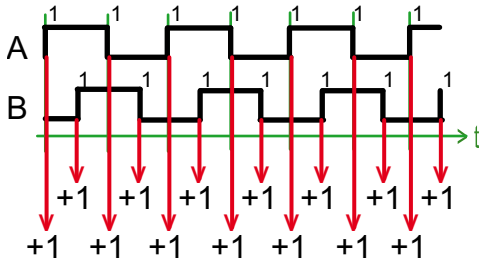
Je nach weiterer Belastung des Geräts kann die Grenzfrequenz sinken, wenn "viele" Drehgeber ausgewertet werden.

Bei zu hoher Belastung kann die Zykluszeit unzulässig lang werden (→ Systemressourcen, Seite 43).

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.

Am Eingang RESOLUTION kann die Auflösung des Drehgebers vervielfacht ausgewertet werden:
1 = normale Auflösung (identisch mit der Auflösung des Drehgebers),
2 = Auflösung doppelt auswerten,
4 = Auflösung 4-fach auswerten.
Alle anderen Werte an diesem Eingang bedeuten normale Auflösung.

	RESOLUTION = 1 Bei normaler Auflösung wird nur die fallende Flanke des B-Signals ausgewertet.
	RESOLUTION = 2 Bei doppelter Auflösung werden die fallenden und die steigenden Flanken des B-Signals ausgewertet.
	RESOLUTION = 4 Bei 4-facher Auflösung werden die fallenden und die steigenden Flanken sowohl des A-Signals wie auch des B-Signals ausgewertet.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
INIT	BOOL	TRUE (nur 1 Zyklus lang): Funktion wird initialisiert FALSE: im zyklischen Programmablauf
CHANNEL	BYTE	Nummer des Eingangskanal-Paares (0...3): 0 = Kanalpaar 0 = Eingänge 0 + 1 1 = Kanalpaar 1 = Eingänge 2 + 3 2 = Kanalpaar 2 = Eingänge 4 + 5 3 = Kanalpaar 3 = Eingänge 6 + 7
PRESET_VALUE	DINT	Voreinstellwert des Zählers
PRESET	BOOL	TRUE: (nur 1 Zyklus lang): Voreinstellwert wird übernommen FALSE: Zähler aktiv
RESOLUTION	BYTE	Faktor der Drehgeber-Auflösung (1, 2, 4): 1 = normale Auflösung 2 = doppelte Auflösung 4 = 4-fache Auflösung Alle anderen Werte zählen wie "1".

Parameter der Funktionsausgänge

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

10.1.8 Funktion FAST_COUNT

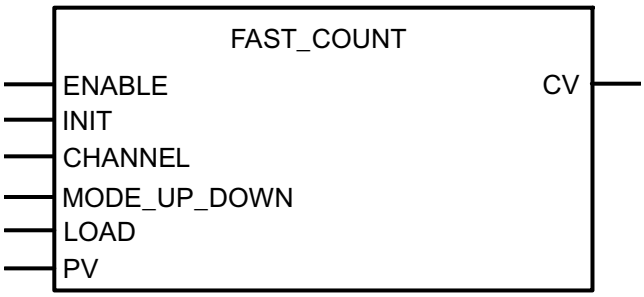
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1071

Funktionssymbol:



Beschreibung

FAST_COUNT arbeitet als Zählerbaustein für schnelle Eingangsimpulse.

Diese Funktion erfasst schnelle Impulse an den FRQ-Eingangskanälen 0...3. Mit dem FRQ-Eingangskanal 0 arbeitet FAST_COUNT wie der Baustein CTU. Maximale Eingangsfrequenz → Datenblatt.

HINWEIS

Bei den R360-Controllern kann der Kanal 0 nur als Aufwärtszähler eingesetzt werden. Die Kanäle 1...3 können als Auf- und Abwärtszähler genutzt werden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet, beginnend vom Startwert FALSE: Funktion wird nicht ausgeführt
INIT	BOOL	TRUE (nur 1 Zyklus lang): Funktion wird initialisiert FALSE: im zyklischen Programmablauf
CHANNEL	BYTE	Nummer des Eingangs (0...x Wert abhängig vom Gerät)
MODE_UP_DOWN	BOOL	TRUE: Zähler zählt abwärts FALSE: Zähler zählt aufwärts
LOAD	BOOL	TRUE: Startwert PV wird geladen FALSE: Startwert "0" wird geladen
PV	DWORD	Startwert (Preset value)

HINWEIS

Nach Rücksetzen des Parameters INIT zählt der Zähler vom angegebenen Startwert an.

Nach erneutem Setzen von ENABLE zählt der Zähler von dem Wert an weiter, der beim letzten Rücksetzen von ENABLE gültig war.

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
CV	DWORD	Ausgangswert des Zählers

10.2 Software-Reset

Inhalt:

Hiermit kann die Steuerung per Kommando im Applikations-Programm neu gestartet werden.

10.2.1 Funktion SOFTRESET

Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1070, CR1071

Funktionssymbol:

Beschreibung

SOFTRESET führt einen kompletten Neustart des Controllers aus.

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

HINWEIS

Bei einer laufenden Kommunikation muss die lange Reset-Phase beachtet werden, da andernfalls Guarding-Fehler gemeldet werden.

Parameter der Funktionseingänge

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

10.3 Daten im Speicher sichern, lesen und wandeln

Inhalt:

10.3.1 Automatische Datensicherung

Die R360-Controller bieten die Möglichkeit, Daten (BOOL, BYTE, WORD, DWORD) remanent (= spannungsausfallsicher) im Speicher zu sichern. Bei Abfall der Versorgungsspannung wird der Sicherungsvorgang automatisch gestartet. Voraussetzung ist, dass die Daten als RETAIN-Variablen angelegt 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 Speicher ausgelesen und wieder in den Merkerbereich zurückgeschrieben.

10.3.2 Manuelle Datensicherung

Neben der Möglichkeit, die Daten automatisch zu sichern, können über Funktionsaufrufe Anwenderdaten manuell in integrierte Speicher gesichert und von dort wieder gelesen werden.

Je nach Controller stehen folgende Speicher zur Verfügung:

- EEPROM-Speicher:
  Nur für SmartController, CabinetController CR0301 / CR0302, Platinensteuerung.
  Langsames Schreiben und Lesen.
  Begrenzte Schreib-/Lesehäufigkeit.
  Beliebige Speicherbereiche wählbar.
  Daten sichern mit Funktion E2WRITE (→ Seite 230).
  Daten lesen mit Funktion E2READ (→ Seite 232).
- FRAM-Speicher
  Nur für ClassicController, ExtendedController, SafetyController, CabinetController CR0303, PDM360 smart.
  Schnelles Schreiben und Lesen.
  Unbegrenzte Schreib-/Lesehäufigkeit.
  Beliebige Speicherbereiche wählbar.
  Daten sichern mit Funktion FRAMWRITE.
  Daten lesen mit Funktion FRAMREAD.
- Flash-Speicher
  Für alle o.g. Controller.
  Schnelles Schreiben und Lesen.
  Begrenzte Schreib-/Lesehäufigkeit.
  Nur zum Speichern großer Datenmengen sinnvoll einsetzbar.
  Vor dem erneuten Schreiben muss Speicherinhalt gelöscht werden.
  Daten sichern mit Funktion FLASHWRITE (→ Seite 227).
  Daten lesen mit Funktion FLASHREAD (→ Seite 229).

Info

Der Programmierer kann sich anhand der Speicheraufteilung (→ Datenblatt oder Betriebsanleitung) darüber informieren, welcher Speicherbereich frei zur Verfügung steht.

10.3.3 Funktion MEMCPY

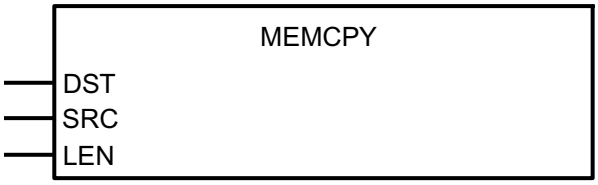
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

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

Die Funktion 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 Wortdatei zu übertragen.

HINWEIS

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

Parameter der Funktionseingänge

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

10.3.4 Funktion FLASHWRITE

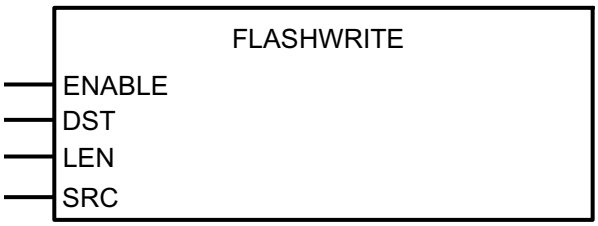
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

⚠ WARNUNG

Gefahr durch unkontrollierten Prozessablauf!

Der Zustand der Ein-/Ausgänge wird während der Ausführung von FLASHWRITE "eingefroren".

► Diese Funktion nicht bei laufender Maschine ausführen!

FLASHWRITE ermöglicht das Schreiben unterschiedlicher Datentypen direkt in den Flash-Speicher.

Die Funktion schreibt den Inhalt der Adresse SRC (muss mit der Funktion ADR ermittelt werden) in den Flash-Speicher. Dabei werden genau so viele Bytes übertragen, wie diese unter LEN angegeben sind.

Bevor der Speicher erneut beschrieben wird, muss vorher ein Löschvorgang durchgeführt werden. Dies geschieht mit dem Beschreiben der Adresse "0" mit beliebigem Inhalt.

Info

Mit dieser Funktion sollen während der Inbetriebnahme große Datenmengen gesichert werden, auf die im Prozess nur lesend zugegriffen wird.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird ausgeführt FALSE: Funktion wird nicht ausgeführt
DST	INT	Relative Anfangsadresse im Speicher. Speicherzugriff nur wortweise; zulässige Werte: 0, 2, 4, 6, 8, ...
LEN	INT	Anzahl der Datenbytes (max. 65.536 Bytes)
SRC	DWORD	Adresse der Quellvariablen

10.3.5 Funktion FLASHREAD

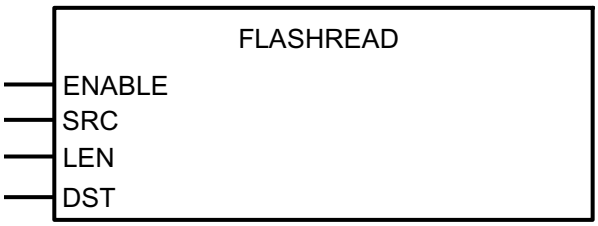
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

FLASHREAD ermöglicht das Lesen unterschiedlicher Datentypen direkt aus dem Flash-Speicher.

Die Funktion liest den Inhalt ab der Adresse von SRC aus dem Flash-Speicher. Dabei werden genau so viele Bytes übertragen, wie diese unter LEN angegeben sind.

HINWEIS

Die Adresse bei DST muss mit der Funktion ADR ermittelt und FLASHREAD übergeben werden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird ausgeführt FALSE: Funktion wird nicht ausgeführt
SRC	INT	Relative Anfangsadresse im Speicher
LEN	INT	Anzahl der Datenbytes (max. 65.536 Bytes)
DST	DWORD	Adresse der Zielvariablen

10.3.6 Funktion E2WRITE

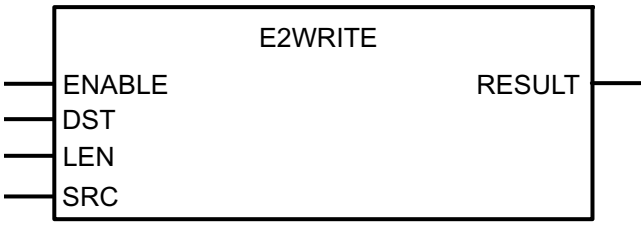
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- SmartController: CR2500
- CabinetController: CR0301, CR0302
- Platinensteuerung: CS0015

Funktionssymbol:



Beschreibung

E2WRITE ermöglicht das Schreiben unterschiedlicher Datentypen direkt in das serielle EEPROM.

Die Funktion schreibt den Inhalt ab der Adresse von SRC in das serielle EEPROM. Da das Abarbeiten 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.

HINWEIS

Die Adresse bei SRC muss mit der Funktion ADR ermittelt und E2WRITE übergeben werden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird ausgeführt FALSE: Funktion wird nicht ausgeführt
DST	INT	Anfangsadresse im Speicher (0...$2FF_{16}$ und 340_{16} bis EEPROM-Größe)
LEN	INT	Anzahl der zu übergebenden Datenbytes
SRC	DINT	Adresse der Quellvariablen

Parameter der Funktionsausgänge

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

10.3.7 Funktion E2READ

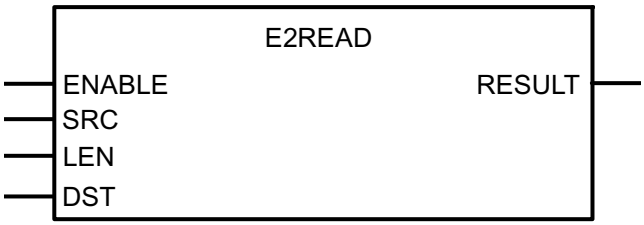
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- SmartController: CR2500
- CabinetController: CR0301, CR0302
- Platinensteuerung: CS0015

Funktionssymbol:



Beschreibung

E2READ ermöglicht das Lesen unterschiedlicher Daten aus dem seriellen EEPROM.

Die Funktion liest den Inhalt ab der Adresse von SRC aus dem seriellen EEPROM aus. 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.

HINWEIS

Die Adresse bei DST muss mit der Funktion ADR ermittelt und E2READ übergeben werden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird ausgeführt FALSE: Funktion wird nicht ausgeführt
SRC	INT	Anfangsadresse im Speicher ($0...2FF_{16}$ und 400_{16} bis EEPROM-Größe)
LEN	INT	Anzahl der zu übergebenden Datenbytes
DST	DINT	Adresse der Zielvariablen

Parameter der Funktionsausgänge

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

10.4 Datenzugriff und Datenprüfung

Inhalt:

Die Funktionen in diesem Kapitel steuern den Datenzugriff und ermöglichen ein Prüfen der Daten.

10.4.1 Funktion SET_DEBUG

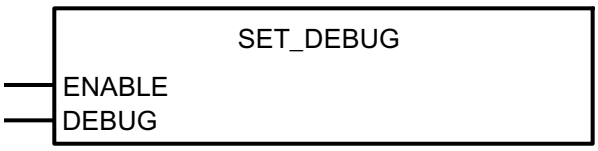
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015

Funktionssymbol:



Beschreibung

SET_DEBUG organisiert den DEBUG-Modus ohne aktiven Test-Eingang (→ Kapitel TEST-Betrieb, Seite 39).

Wird der Eingang DEBUG der Funktion auf TRUE gesetzt, kann z.B. das Programmiersystem oder der Downloader mit dem Controller kommunizieren und Systemkommandos ausführen (z.B. für Servicefunktionen über das GSM-Modem CANremote).

HINWEIS

Ein Softwaredownload ist in dieser Betriebsart nicht möglich, da der Test-Eingang nicht mit Versorgungsspannung verbunden wird.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird ausgeführt FALSE: Funktion wird nicht abgearbeitet
DEBUG	BOOL	TRUE: Debugging über die Schnittstellen möglich FALSE: Debugging über die Schnittstellen nicht möglich

10.4.2 Funktion SET_IDENTITY

Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1070, CR1071

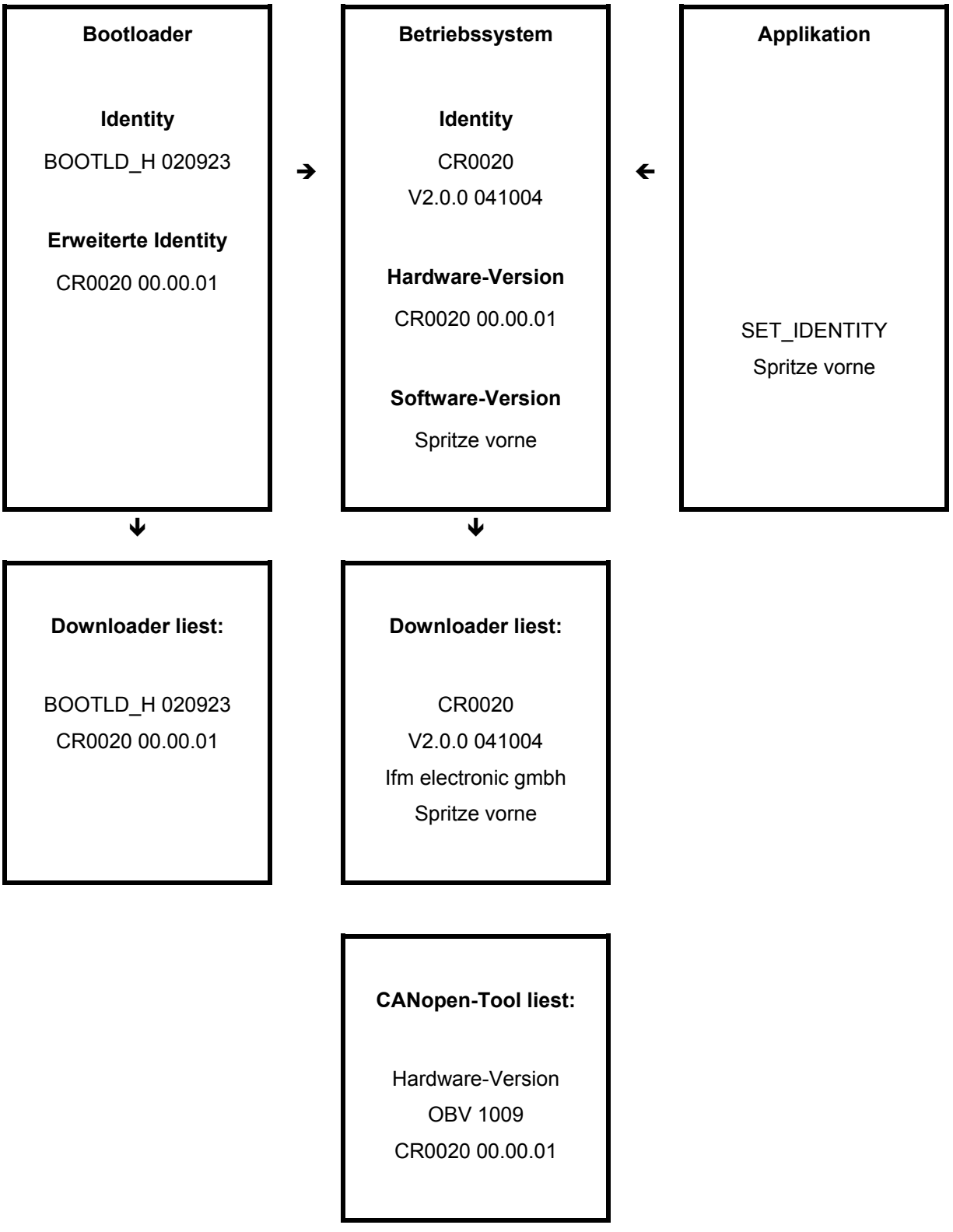
Funktionssymbol:

Beschreibung

SET_IDENTITY setzt eine applikationsspezifische Programmkennung.

Mit der Funktion kann durch das Applikationsprogramm eine Programmkennung erzeugt werden. Diese Kennung kann zur Identifizierung des geladenen Programms über das Software-Tool DOWNLOADER.EXE als Software-Version ausgelesen werden.

Die nachfolgende Grafik zeigt die Zusammenhänge der unterschiedlichen Kennungen, wie sie mit den unterschiedlichen Software-Tools angezeigt werden. (Beispiel: ClassicController CR0020):



Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ID	STRING(80)	Beliebiger String mit einer maximalen Länge von 80 Zeichen

10.4.3 Funktion GET_IDENTITY

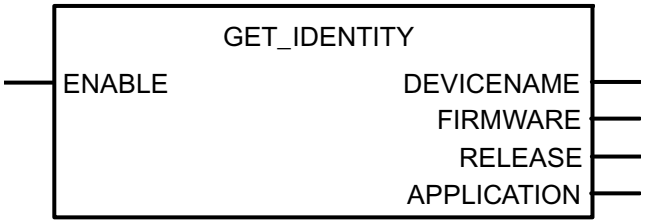
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

GET_IDENTITY liest die im Controller gespeicherte applikations-spezifische Programmkennung.

Mit der Funktion kann vom Applikations-Programm die gespeicherte Programmkennung gelesen werden. Folgende Angaben sind verfügbar:

- Hardware-Name und Version
  z.B.: "CR0032 00.00.01"
- Name des Laufzeitsystems
  z.B.: "CR0032"
- Version und Build des Laufzeitsystems
  z.B.: "V00.00.01 071128"
- Name der Applikation
  z.B.: "Crane1704"

Der Name der Applikation kann mit der Funktion SET_IDENTITY (→ Seite 236) verändert werden.

Parameter der Funktionseingänge

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

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
DEVICENAME	STRING(31)	Hardware-Name und Version als String von max. 31 Zeichen z.B.: "CR0032 00.00.01"
FIRMWARE	STRING(31)	Name des Laufzeitsystems als String von max. 31 Zeichen z.B.: "CR0032"
RELEASE	STRING(31)	Version und Build des Laufzeitsystems als String von max. 31 Zeichen z.B.: "V00.00.01 071128"
APPLICATION	STRING(79)	Name der Applikation als String von max. 79 Zeichen z.B.: "Crane1704"

10.4.4 Funktion SET_PASSWORD

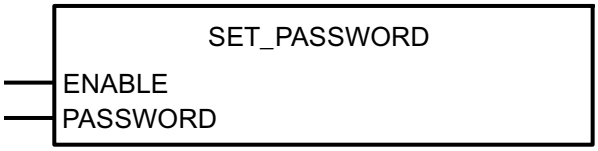
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

SET_PASSWORD setzt Benutzerkennung für Programm- und Speicher-Upload mit dem DOWNLOADER.

Ist die Benutzerkennung aktiv, kann durch das Software-Tool DOWNLOADER das Applikations-Programm oder der Datenspeicher nur ausgelesen werden, wenn das richtige Password eingegeben wurde.

Wird an den Eingang PASSWORD ein Leer-String (Default-Zustand) übergeben, ist ein Upload der Applikations-Software oder des Datenspeichers jederzeit möglich.

ACHTUNG

Für CR250n, CR0301, CR0302, CS0015 beachten:

Das EEPROM-Speichermodul kann bei Dauerbetrieb dieser Funktion zerstört werden!

► Die Funktion nur **einmalig** bei der Initialisierung im ersten Programmzyklus ausführen! Anschließend die Funktion wieder sperren (ENABLE = "FALSE")!

HINWEIS

Beim Laden eines neuen Applikations-Programms wird die Kennung wieder zurückgesetzt.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE (nur 1 Zyklus lang): Kennung wird gesetzt FALSE: Funktion wird nicht abgearbeitet
PASSWORD	STRING	Benutzerkennung (maximale String-Länge 16)

10.4.5 Funktion CHECK_DATA

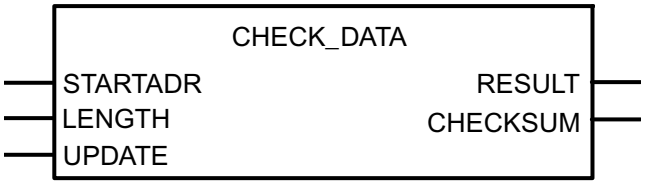
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

CHECK_DATA sichert die Daten im Applikations-Datenspeicher über einen CRC-Code.

Die Funktion dient dazu, in sicherheitsrelevanten Applikationen 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.

Wenn Eingang UPDATE = FALSE und Daten im Speicher sich ungewollt verändern, wird RESULT = FALSE. Das Ergebnis kann dann für weitere Aktionen (z.B. Abschalten der Ausgänge) genutzt werden.

Nur wenn der Eingang UPDATE auf TRUE gesetzt ist, sind Datenänderungen im Speicher (z.B. vom Applikations-Programm oder ecomat*mobil*-Gerät) zulässig. Der Wert der Prüfsumme wird dann neu berechnet. Der Ausgang RESULT ist wieder permanent TRUE.

Die Startadresse (z.B. %MW0) muss über den Adressoperator ADR an die Funktion übergeben werden. Zusätzlich muss die Anzahl der Datenbytes LENGTH (Länge ab der STARTADR) angegeben werden.

HINWEIS

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

Parameter der Funktionseingänge

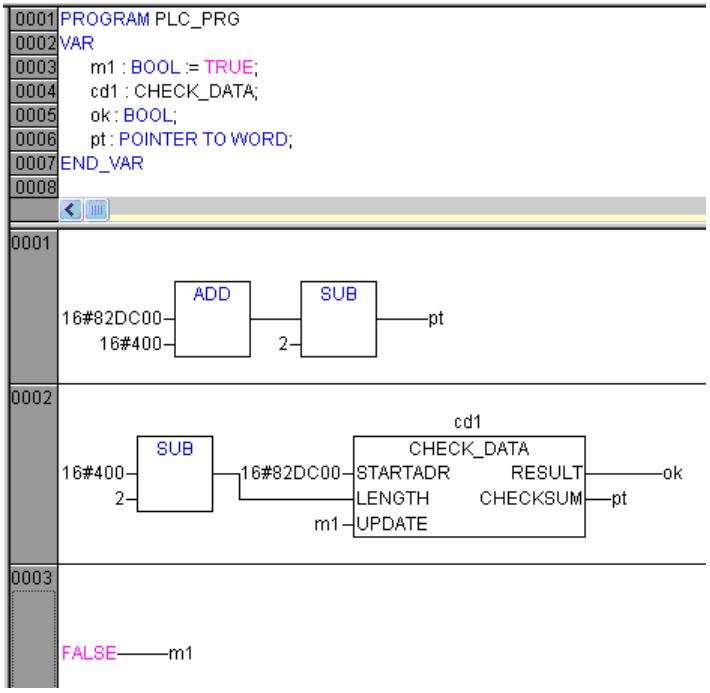
Name	Datentyp	Beschreibung
STARTADR	DINT	Startadresse des überwachten Datenspeichers (WORD-Adresse ab %MW0)
LENGTH	WORD	Länge des überwachten Datenspeichers in [Byte]
UPDATE	BOOL	TRUE: Datenänderungen zulässig FALSE: Datenänderungen nicht zulässig

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
RESULT	BOOL	TRUE: CRC-Checksumme in Ordnung FALSE: CRC-Checksumme fehlerhaft (Daten wurden geändert)
CHECKSUM	WORD	Ergebnis der CRC-Prüfsummenbildung

Beispiel zu CHECK_DATA

Im folgenden Beispiel ermittelt das Programm die Prüfsumme und legt sie über den Pointer pt im RAM ab:



HINWEIS: Das hier gezeigte Verfahren ist für den Flash-Speicher nicht geeignet.

10.5 Interrupts verarbeiten

Inhalt:

Die SPS arbeitet das gespeicherte Applikations-Programm zyklisch in voller Länge ab. Von z.B. äußeren Ereignissen abhängige Verzweigungen im Programm (= bedingte Sprünge) lassen die Zykluszeit variieren. Für bestimmte Funktionen kann dieses Verhalten nachteilig sein.

Mit Hilfe gezielter Unterbrechungen (= Interrupts) des zyklischen Programmablaufs können zeitkritische Abläufe unabhängig vom Zyklus in festen Zeitrastern oder bei bestimmten Ereignissen aufgerufen werden.

Für SafetyController sind Interrupt-Funktionen grundsätzlich nicht zulässig und deshalb nicht verfügbar.

10.5.1 Funktion SET_INTERRUPT_XMS

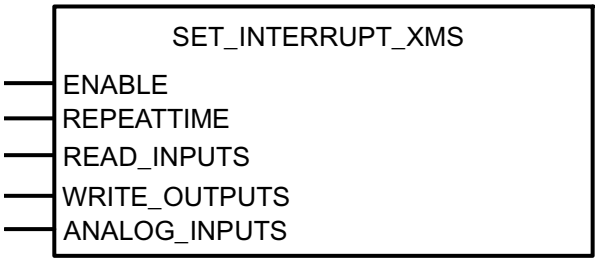
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1071

Funktionssymbol:



Beschreibung

SET_INTERRUPT_XMS organisiert das Ausführen eines Programmteils im Intervall von x ms.

In der 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 einem einzigen unveränderlichen Block abläuft.

Ein möglicher Lösungsansatz wäre, die Zykluszeit kurz zu halten. Dieser Weg führt oft dazu, die Applikation auf mehrere Steuerungszyklen zu verteilen. Die Programmierung wird dadurch jedoch unübersichtlich und schwierig.

Eine andere Möglichkeit besteht darin, einen bestimmten Programmteil in festen Zeitabständen (alle x 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 Interrupt-Routine deklariert, indem einmalig (zur Initialisierungzeit) die Funktion SET_INTERRUPT_XMS aufgerufen wird. Das hat zur Folge, dass dieser Programmteil immer nach Ablauf der REPEATTIME (alle x ms) abgearbeitet wird. Werden Ein- und Ausgänge in diesem Programmteil genutzt, werden diese ebenfalls im festgelegten Takt gelesen oder beschrieben. Über die Funktionseingänge READ_INPUTS, WRITE_OUTPUTS oder ANALOG_INPUTS kann das Lesen oder Schreiben unterbunden werden.

Innerhalb des Programmteils 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 es in einem "normalen" Zyklus möglich ist.

HINWEIS

Damit der per Interrupt aufgerufene Programmteil nicht zusätzlich zyklisch aufgerufen wird, sollte er (mit Ausnahme des Initialisierungsaufrufes) im Zyklus übersprungen werden.

Es können mehrere Timer-Interrupt-Blöcke aktiv sein. Der Zeitbedarf der Interrupt-Funktionen muss so berechnet werden, dass alle aufgerufenen Funktionen ausgeführt werden können. Das gilt besonders bei Berechnungen, Gleitkomma-Arithmetik und Regler-Funktionen.

Bitte beachten: Bei einer hohen CAN-Busaktivität kann die eingestellte REPEATTIME schwanken.

HINWEIS

Die Eindeutigkeit der Ein- und Ausgänge im Zyklus wird durch die Interrupt-Routine aufgehoben. Deshalb wird nur ein Teil der Ein- und Ausgänge bedient. Wurden sie im Interrupt-Programm initialisiert, werden folgende Ein- und Ausgänge gelesen oder geschrieben.

Eingänge, digital:

%IX0.0...%IX0.7 (CRnn32)

%IX0.12...%IX0.15, %IX1.4...%IX1.8 (übrige ClassicController, ExtendedController, SafetyController)

%IX0.0, %IX0.8 (SmartController)

IN08...IN11 (CabinetController)

IN0...IN3 (Platinensteuerung)

Eingänge, analog:

%IX0.0...%IX0.7 (CRnn32)

alle Kanäle (Auswahl bitcodiert) (alle übrigen Controller)

Ausgänge, digital:

%QX0.0...%QX0.7 (ClassicController, ExtendedController, SafetyController)

%QX0.0, %QX0.8 (SmartController)

OUT00...OUT03 (CabinetController)

OUT0...OUT7 (Platinensteuerung)

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.

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

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE (nur 1 Zyklus): Datenänderungen zulässig FALSE: Datenänderungen nicht zulässig (während des Programmablaufs)
REPEATTIME	TIME	Zeitfenster, in dem der Interrupt ausgelöst wird
READ_INPUTS	BOOL	TRUE: in die Routine eingebundene Eingänge werden gelesen (Eingänge ggf. auf IN_FAST setzen) FALSE: in die Routine eingebundene Eingänge werden nicht gelesen
WRITE_OUTPUTS	BOOL	TRUE: in die Routine eingebundene Ausgänge werden geschrieben FALSE: in die Routine eingebundene Ausgänge werden nicht geschrieben
ANALOG_INPUTS	BOOL	TRUE: in die Routine eingebundene Analog-Eingänge werden gelesen und der Rohwert der Spannung an die Systemmerker ANALOG_IRQxx ausgegeben FALSE: in die Routine eingebundene Analog-Eingänge werden nicht gelesen

10.5.2 Funktion SET_INTERRUPT_I

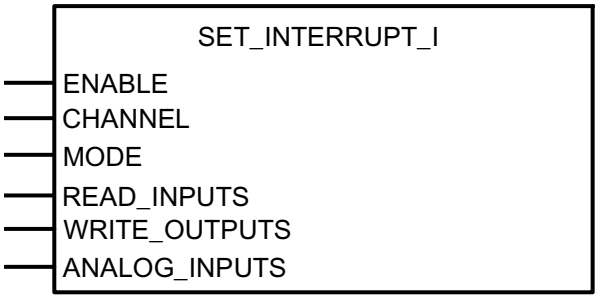
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

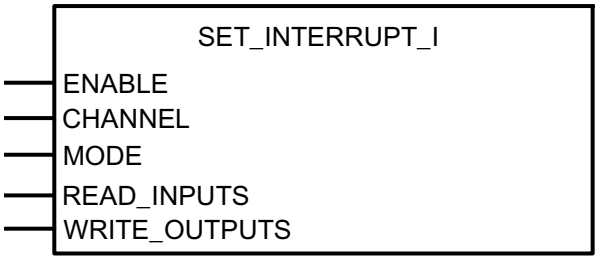
verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1071

Funktionssymbol:



(nur bei Geräten mit Analog-Kanälen)



(bei Geräten ohne Analog-Kanäle)

Beschreibung

SET_INTERRUPT_I organisiert das Ausführen eines Programmteils durch eine Interrupt-Anforderung über einen Eingangskanal.

In der 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 einem einzigen unveränderlichen Block abläuft.

Ein möglicher Lösungsansatz wäre, die Zykluszeit kurz zu halten. Dieser Weg führt oft dazu, die Applikation auf mehrere Steuerungszyklen zu verteilen. Die Programmierung wird dadurch jedoch unübersichtlich und schwierig.

Eine andere Möglichkeit besteht darin, einen bestimmten Programmteil nur auf Anforderung durch einen Eingangsimpuls unabhängig vom Steuerungszyklus aufzurufen.

Der zeitkritische Teil der Applikation wird vom Anwender in einen Baustein vom Type PROGRAMM (PRG) zusammengefasst. Dieser Baustein wird zur Interrupt-Routine deklariert, indem einmalig (zur Initialisierungzeit) die Funktion SET_INTERRUPT_I aufgerufen wird. Das hat zur Folge, dass dieser Programmteil immer dann ausgeführt wird, wenn eine Flanke am Eingang CHANNEL erkannt wird.

Werden Ein- und Ausgänge in diesem Programmteil genutzt, werden diese ebenfalls in der Interruptroutine, ausgelöst durch die Eingangs-Flanke, gelesen oder beschrieben. Über die Funktionseingänge READ_INPUTS, WRITE_OUTPUTS oder ANALOG_INPUTS kann das Lesen oder Schreiben unterbunden werden.

Innerhalb des Programmteils können also alle zeitkritischen Ereignisse bearbeitet werden, indem Eingänge oder globale Variablen verknüpft und Ausgänge beschrieben werden. So können auch Funktionen nur genau dann ausgeführt werden, wenn sie durch ein Eingangssignal angefordert werden.

HINWEIS

Damit der per Interrupt aufgerufene Programmteil nicht zusätzlich zyklisch aufgerufen wird, sollte er (mit Ausnahme des Initialisierungsaufrufes) im Zyklus übersprungen werden.

Der Eingang (CHANNEL), der zum Auslösen des Interrupt überwacht wird, kann in der Interruptroutine nicht initialisiert und weiter verarbeitet werden.

Die Eingänge müssen in der Betriebsart IN_FAST sein, sonst können die Interrupts nicht gelesen werden.

HINWEIS

Die Eindeutigkeit der Ein- und Ausgänge im Zyklus wird durch die Interrupt-Routine aufgehoben. Deshalb wird nur ein Teil der Ein- und Ausgänge bedient. Wurden sie im Interrupt-Programm initialisiert, werden folgende Ein- und Ausgänge gelesen oder geschrieben.

Eingänge, digital:

%IX0.0...%IX0.7 (CRnn32)

%IX0.12...%IX0.15, %IX1.4...%IX1.8 (übrige ClassicController, ExtendedController, SafetyController)

%IX0.0, %IX0.8 (SmartController)

IN08...IN11 (CabinetController)

IN0...IN3 (Platinensteuerung)

Eingänge, analog:

%IX0.0...%IX0.7 (CRnn32)

alle Kanäle (Auswahl bitcodiert) (alle übrigen Controller)

Ausgänge, digital:

%QX0.0...%QX0.7 (ClassicController, ExtendedController, SafetyController)

%QX0.0, %QX0.8 (SmartController)

OUT00...OUT03 (CabinetController)

OUT0...OUT7 (Platinensteuerung)

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.

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

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE (nur 1 Zyklus): Datenänderungen zulässig FALSE: Datenänderungen nicht zulässig (während des Programmablaufs)
CHANNEL	BYTE	Interrupt-Eingang Classic/ExtendedController: 0 = %IX1.4 1 = %IX1.5 2 = %IX1.6 3 = %IX1.7 SmartController: 0 = %IX0.0 1 = %IX0.8 CabinetController: 0 = IN08 (usw.) 3 = IN11 CS0015: 0 = IN0 (usw.) 3 = IN3
MODE	BYTE	Art der Flanke am Eingang CHANNEL, die den Interrupt auslöst 1 = steigende Flanke 2 = fallende Flanke 3 = steigende und fallende Flanke
READ_INPUTS	BOOL	TRUE: in die Routine eingebundene Eingänge werden gelesen (Eingänge ggf. auf IN_FAST setzen) FALSE: in die Routine eingebundene Eingänge werden nicht gelesen
WRITE_OUTPUTS	BOOL	TRUE: in die Routine eingebundene Ausgänge werden geschrieben FALSE: in die Routine eingebundene Ausgänge werden nicht geschrieben

Name	Datentyp	Beschreibung
ANALOG_INPUTS	BYTE	**(gilt nur bei Geräten mit Analogkanälen)** Auswahl der Eingänge bitcodiert: 0_{10} = kein Eingang gewählt 1_{10} = 1. Analogeingang gewählt ($0000\ 0001_2$) 2_{10} = 2. Analogeingang gewählt ($0000\ 0010_2$) ... 128_{10} = 8. Analogeingang gewählt ($1000\ 0000_2$) Eine Kombination der Eingänge entsteht durch ODER-Verknüpfung der Werte. Beispiel: 1. und 3. Analogeingang wählen: ($0000\ 0001_2$) ODER ($0000\ 0100_2$) = ($0000\ 0101_2$) = 5_{10}

10.6 Nutzung der seriellen Schnittstelle

Inhalt:

HINWEIS

Grundsätzlich steht die serielle Schnittstelle dem Anwender nicht zur Verfügung, da sie für den Programm-Download und das Debugging genutzt wird.

Setzt der Anwender das Systemmerker-Bit SERIAL_MODE auf TRUE, dann kann die Schnittstelle frei genutzt werden. Der Programm-Download und das Debugging sind dann jedoch nur noch über die CAN-Schnittstelle möglich.
Für CRxx32 gilt: Ein Debugging der Applikations-Software ist dann nur noch über alle 4 CAN-Schnittstellen oder über USB möglich.

Mit den folgend aufgeführten Funktionen kann die serielle Schnittstelle im Applikations-Programm genutzt werden.

10.6.1 Funktion SERIAL_SETUP

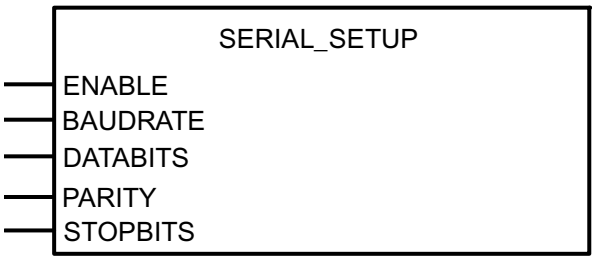
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

SERIAL_SETUP initialisiert die serielle RS232-Schnittstelle.

Die Funktion setzt die serielle Schnittstelle auf die angegebenen Parameter. Mit dem Funktionseingang ENABLE wird die Funktion für einen Zyklus aktiviert.

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

HINWEIS

Grundsätzlich steht die serielle Schnittstelle dem Anwender nicht zur Verfügung, da sie für den Programmdownload und das Debugging genutzt wird.

Setzt der Anwender das Systemmerkerbit SERIAL_MODE auf TRUE, dann kann die Schnittstelle frei genutzt werden. Der Programmdownload und das Debugging sind dann jedoch nur noch über die CAN-Schnittstelle möglich.
Für CRnn32 gilt: Ein Debugging der Applikations-Software ist dann nur noch über alle 4 CAN-Schnittstellen oder über USB möglich.

HINWEIS

Ein Teil der Ein- und Ausgänge des SafetyControllers ist für Applikationen bis zu PL d nach ISO 13849 zugelassen. Voraussetzung dafür ist, dass die Ein- und Ausgänge des SafetyController (wie in Kapitel Konfigurationen (→ Seite 13) beschrieben) verschaltet und durch das Applikations-Programm ausgewertet werden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE (nur 1 Zyklus lang): Schnittstelle wird initialisiert FALSE: laufender Betrieb
BAUDRATE	BYTE	Baud-Rate (zulässige Werte: 9600, 19200, 28800, (57600)) Voreinstellwert → Datenblatt
DATABITS	BYTE	Daten-Bits (zulässige Werte: 7 oder 8) Voreinstellwert = 8
PARITY	BYTE	Parität (zulässige Werte: 0=keine, 1=gerade, 2=ungerade) Voreinstellwert = 0
STOPBITS	BYTE	Stopp-Bits (zulässige Werte: 1 oder 2) Voreinstellwert = 1

10.6.2 Funktion SERIAL_TX

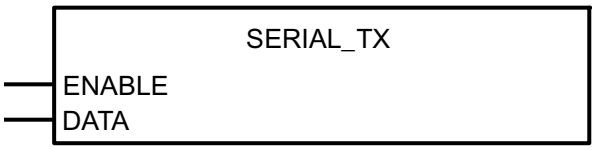
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

SERIAL_TX überträgt ein Datenbyte über die serielle RS232-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.

HINWEIS

Grundsätzlich steht die serielle Schnittstelle dem Anwender nicht zur Verfügung, da sie für den Programmdownload und das Debugging genutzt wird.

Setzt der Anwender das Systemmerkerbit SERIAL_MODE auf TRUE, dann kann die Schnittstelle frei genutzt werden. Der Programmdownload und das Debugging sind dann jedoch nur noch über die CAN-Schnittstelle möglich.
Für CRnn32 gilt: Ein Debugging der Applikations-Software ist dann nur noch über alle 4 CAN-Schnittstellen oder über USB möglich.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Übertragung freigegeben FALSE: Übertragung gesperrt
DATA	BYTE	zu übertragendes Byte

10.6.3 Funktion SERIAL_RX

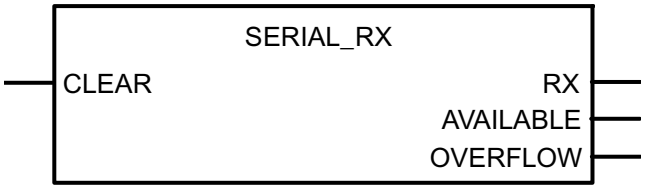
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

SERIAL_RX liest mit jedem Aufruf ein empfangenes Datenbyte aus dem seriellen Empfangspuffer aus.

Anschließend wird der Wert von AVAILABLE um 1 dekrementiert.

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 Schnittstelle.

HINWEIS

Grundsätzlich steht die serielle Schnittstelle dem Anwender nicht zur Verfügung, da sie für den Programmdownload und das Debugging genutzt wird.

Setzt der Anwender das Systemmerkerbit SERIAL_MODE auf TRUE, dann kann die Schnittstelle frei genutzt werden. Der Programmdownload und das Debugging sind dann jedoch nur noch über die CAN-Schnittstelle möglich.
Für CRnn32 gilt: Ein Debugging der Applikations-Software ist dann nur noch über alle 4 CAN-Schnittstellen oder über USB möglich.

Parameter der Funktionseingänge

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

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
RX	BYTE	empfangene Byte-Daten aus dem Empfangspuffer
AVAILABLE	WORD	Anzahl der empfangenen Datenbytes 0 = keine gültigen Daten vorhanden
OVERFLOW	BOOL	Überlauf des Datenpuffers, Datenverlust!

Beispiel:

Es werden 3 Bytes empfangen:

1. Aufruf von SERIAL_RX
   1 gültiger Wert am Ausgang RX
   → AVAILABLE = 3
2. Aufruf von SERIAL_RX
   1 gültiger Wert am Ausgang RX
   → AVAILABLE = 2
3. Aufruf von SERIAL_RX
   1 gültiger Wert am Ausgang RX
   → AVAILABLE = 1
4. Aufruf von SERIAL_RX
   ungültiger Wert am Ausgang RX
   → AVAILABLE = 0

Wenn AVAILABLE = 0 ist, kann die Funktion im Programmablauf übersprungen werden.

10.6.4 Funktion SERIAL_PENDING

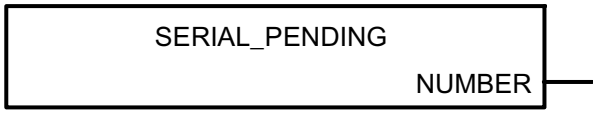
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

SERIAL_PENDING ermittelt die Anzahl der im seriellen Empfangspuffer gespeicherten Datenbytes.

Im Gegensatz zur Funktion SERIAL_RX (→ Seite 256) 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.

HINWEIS

Grundsätzlich steht die serielle Schnittstelle dem Anwender nicht zur Verfügung, da sie für den Programmdownload und das Debugging genutzt wird.

Setzt der Anwender das Systemmerkerbit SERIAL_MODE auf TRUE, dann kann die Schnittstelle frei genutzt werden. Der Programmdownload und das Debugging sind dann jedoch nur noch über die CAN-Schnittstelle möglich.
Für CRnn32 gilt: Ein Debugging der Applikations-Software ist dann nur noch über alle 4 CAN-Schnittstellen oder über USB möglich.

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
NUMBER	WORD	Anzahl der empfangenen Datenbytes

10.7 Systemzeit auslesen

Inhalt:

Mit folgenden Funktionen der **ifm electronic gmbh** können Sie die kontinuierlich laufende Systemzeit des Controllers lesen und im Applikations-Programm auswerten.

10.7.1 Funktion TIMER_READ

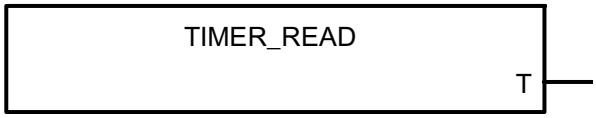
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1070, CR1071

Funktionssymbol:



Beschreibung

TIMER_READ liest die aktuelle Systemzeit aus.

Mit Anlegen der Versorgungsspannung bildet der Controller einen Zeittakt, der in einem Register aufwärts gezählt wird. Dieses Register kann mittels des Funktionsaufrufes ausgelesen und z.B. zur Zeitmessung genutzt werden.

HINWEIS

Der System-Timer läuft maximal bis FFFF $FFFF_{16}$ (entspricht ca. 49,7 Tage) und startet anschließend wieder bei 0.

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
T	TIME	Aktuelle Systemzeit (Auflösung [ms])

10.7.2 Funktion TIMER_READ_US

Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015
- PDM360 smart: CR1070, CR1071

Funktionssymbol:

Beschreibung

TIMER_READ_US liest die aktuelle Systemzeit in [µs] aus.

Mit Anlegen der Versorgungsspannung bildet der Controller einen Zeittakt, der in einem Register aufwärts gezählt wird. Dieses Register kann mittels des Funktionsaufrufes ausgelesen werden und z.B. zur Zeitmessung genutzt werden.

Info

Der System-Timer läuft maximal bis zum Zählerwert 4294967295 (µs) und startet anschließend wieder bei 0.

4294967295 µs = 71582,8 min = 1193 h = 49,7 d

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
TIME_US	DWORD	Aktuelle Systemzeit (Auflösung [µs])

10.8 Analoge Eingangswerte verarbeiten

Inhalt:

Hier stellen wir Ihnen Funktionen vor, mit denen Sie die Werte analoger Spannungen oder Ströme am Controller-Eingang lesen und verarbeiten können.

10.8.1 Funktion INPUT_ANALOG

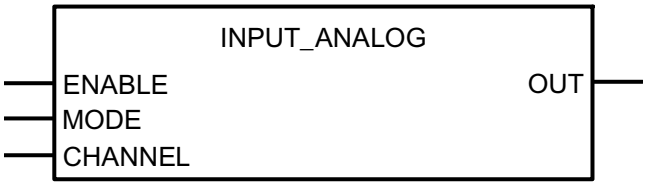
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
  (Für Sicherheitssignale zusätzlich Funktion SAFE_ANALOG_OK einsetzen!)
- CabinetController: CR0301, CR0302, CR0303

Funktionssymbol:



Beschreibung

INPUT_ANALOG ermöglicht Strom- und Spannungsmessung an den Analogkanälen.

Die Funktion liefert den aktuellen Analogwert am gewählten Analogkanal. Die Messung und der Ausgangswert resultiert aus der über MODE angegebenen Betriebsart (Digital-Eingang, 0...20 mA, 0...10 V, 0...30 V). Zur Parametrierung der Betriebsart sollten die angegebenen globalen Systemvariablen genutzt werden. Die Analogwerte werden normiert ausgegeben.

HINWEIS

Wird diese Funktion genutzt, muss unbedingt die Systemvariable RELAIS gesetzt werden, sonst fehlen die internen Referenzspannungen für die Strommessung.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung		
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet		
MODE	BYTE	IN_DIGITAL_H	Digitaleingang	
		IN_CURRENT	Stromeingang	0...20.000 µA
		IN_VOLTAGE10	Spannungseingang	0...10.000 mV
		IN_VOLTAGE30	Spannungseingang	0...30.000 mV
		IN_VOLTAGE32	Spannungseingang	0...32.000 mV
		IN_RATIO	ratiometrischer Analogeingang	
INPUT_CHANNEL	BYTE	Eingangskanal		

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
OUT	WORD	Ausgangswert

10.8.2 Funktion INPUT_VOLTAGE

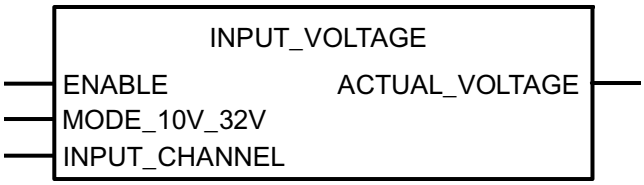
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303

Funktionssymbol:



Beschreibung

INPUT_VOLTAGE verarbeitet an den Analogkanälen gemessene analoge Spannungen.

Die Funktion liefert die aktuelle Eingangsspannung in mV an dem gewählten Analogkanal. Die Messung bezieht sich auf den über MODE_10V_32V angegebenen Spannungsbereich (10.000 mV oder 32.000 mV).

Info

INPUT_VOLTAGE ist eine Kompatibilitätsfunktion für ältere Programme. In neuen Programmen sollte die leistungsfähigere Funktion INPUT_ANALOG (→ Seite 263) eingesetzt werden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet
MODE_10V_32V	BOOL	TRUE: Spannungsbereich 0...32 V FALSE: Spannungsbereich 0...10 V
INPUT_CHANNEL	BYTE	Eingangskanal

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
ACTUAL_VOLTAGE	WORD	Ausgangsspannung in [mV]

10.8.3 Funktion INPUT_CURRENT

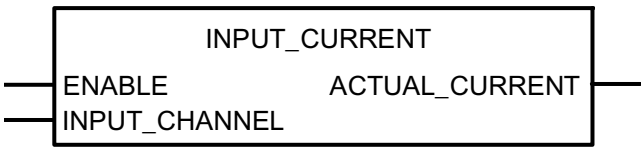
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303

Funktionssymbol:



Beschreibung

INPUT_CURRENT verarbeitet an den Analogkanälen gemessene analoge Ströme.

Die Funktion liefert den aktuellen Eingangsstrom in [µA] an den analogen Stromeingängen.

> **Info**
>
> INPUT_CURRENT ist eine Kompatibilitätsfunktion für ältere Programme. In neuen Programmen sollte die leistungsfähigere Funktion INPUT_ANALOG (→ Seite 263) eingesetzt werden.

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
ENABLE	BOOL	TRUE: Funktion wird abgearbeitet FALSE: Funktion wird nicht abgearbeitet
INPUT_CHANNEL	BYTE	Analoge Stromeingänge 4...7

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
ACTUAL_CURRENT	WORD	Eingangsstrom in [µA]

10.9 Analoge Werte anpassen

Inhalt:

Wenn die Werte analoger Eingänge oder die Ergebnisse von analogen Funktionen angepasst werden müssen, helfen Ihnen die folgenden Funktionen.

10.9.1 Funktion NORM

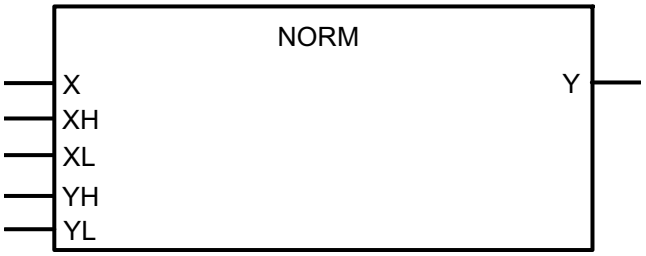
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015

Funktionssymbol:



Beschreibung

NORM normiert einen Wert innerhalb festgelegter Grenzen auf einen Wert mit neuen Grenzen.

Die Funktion 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.

HINWEIS

Der Wert für X muss sich im definierten Eingangsbereich zwischen XL und XH befinden (es findet keine interne Plausibilitätsprüfung des Wertes statt).

Bedingt durch die Rundungsfehler können Abweichungen beim normierten Wert um 1 auftreten.

Werden die Grenzen (XH/XL oderYH/YL) invertiert angegeben, erfolgt auch die Normierung invertiert.

Parameter der Funktionseingänge

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

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
Y	WORD	normierter Wert

Beispiel 1

unterer Grenzwert Eingang	0	XL
oberer Grenzwert Eingang	100	XH
unterer Grenzwert Ausgang	0	YL
oberer Grenzwert Ausgang	2000	YH

dann wandelt der Funktionsblock das Eingangssignal z.B. wie folgt um:

von X =	50	0	100	75
nach Y =	1000	0	2000	1500

Beispiel 2

unterer Grenzwert Eingang	2000	XL
oberer Grenzwert Eingang	0	XH
unterer Grenzwert Ausgang	0	YL
oberer Grenzwert Ausgang	100	YH

dann wandelt der Funktionsblock das Eingangssignal z.B. wie folgt um:

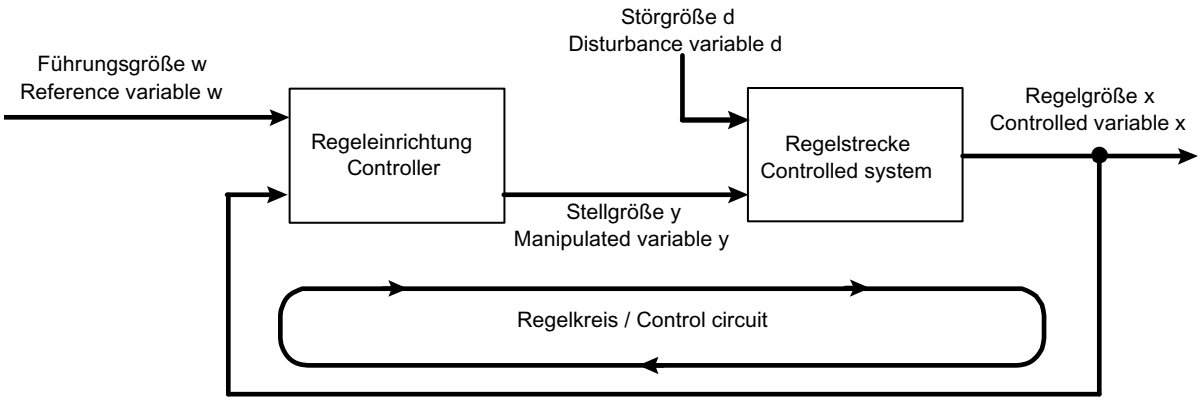
von X =	1000	0	2000	1500
nach Y =	50	100	0	25

11 Regler-Funktionen im ecomatmobil-Controller

Inhalt:

11.1 Allgemeines

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



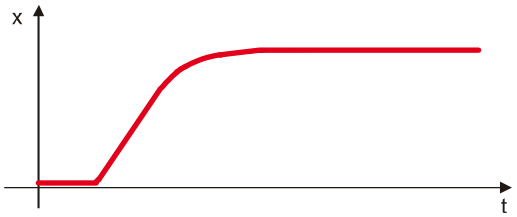
Grafik: Prinzip einer Regelung

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 Regelstrecken unterscheiden:

11.1.1 Regelstrecke mit Ausgleich

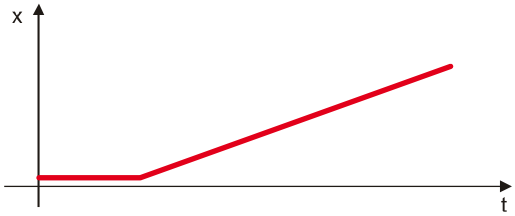
Bei einer Regelstrecke mit Ausgleich strebt die Regelgröße x nach einer bestimmten Stellgrößenänderung einem neuen Endwert (Beharrungszustand) zu. Entscheidend ist bei diesen Regelstrecken die Verstärkung (Übertragungsbeiwert KS). Je kleiner die Verstärkung ist, um so besser lässt sich die Strecke regeln. Man bezeichnet diese Regelstrecken als P-Systeme (P = proportional).



Grafik: P-Regler = Regelstrecke mit Ausgleich

11.1.2 Regelstrecke ohne Ausgleich

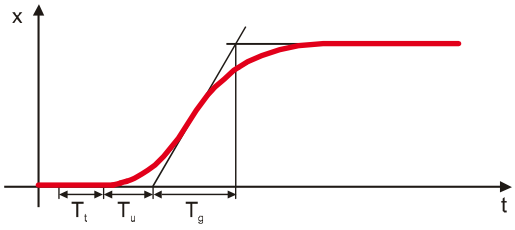
Regelstrecken mit einem Verstärkungsfaktor gegen unendlich werden als Regelstrecken ohne Ausgleich bezeichnet. Dieses ist meistens auf ein integrierendes Verhalten zurückzuführen. Diese hat zur Folge, dass nach der Änderung der Stellgröße oder durch Einfluss einer Störgröße die Regelgröße stetig wächst. Durch dieses Verhalten erreicht sie nie einen Endwert. Man bezeichnet diese Regelstrecken als I-Systeme (I = integral).



Grafik: I-Regler = Regelstrecke ohne Ausgleich

11.1.3 Regelstrecke mit Verzögerung

Die meisten Regelstrecken entsprechen der Reihenschaltung von P-Systemen (Strecken mit Ausgleich) und einem oder mehreren T1-Systemen (Strecken mit Trägheit). Eine Regelstrecke 1. Ordnung entsteht z.B. durch die Reihenschaltung einer Drosselstelle und einem dahinter liegenden Speicher.



Grafik: PT-System = Regelstrecke mit Verzögerung

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 der vorgestellten Grundfunktionen dar. Welche Funktionen zum Einsatz kommen und wie sie kombiniert werden, hängt von der jeweiligen Regelstrecke ab.

11.2 Einstellregel für einen Regler

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

11.2.1 Einstellregel

Die Regeleinrichtung wird zunächst als eine reine P-Regeleinrichtung betrieben. Dazu wird die Vorhaltezeit T_V auf 0 und die Nachstellzeit T_N auf einen sehr großen Wert (ideal auf ∞) für eine träge Strecke eingestellt. Bei einer schnellen Regelstrecke sollte ein kleines T_N 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.

T_V sollte ca. 2...10-mal kleiner sein als T_N

KP sollte gleich groß wie KD gewählt werden.

Idealisiert ist die Regelstrecke wie folgt einzustellen:

Regeleinrichtung	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 * $T_{kritisch}$

HINWEIS

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

11.2.2 Dämpfung von Überschwingungen

Um Überschwingungen zu dämpfen, kann die Funktion PT1 (→ Seite 276) (Tiefpass) eingesetzt werden. Dazu wird der Sollwert XS durch das PT1-Glied gedämpft, bevor er der Reglerfunktion zugeführt wird.

Die Einstellgröße T1 sollte ca. 4...5-mal größer sein als TN (des PID- oder GLR-Reglers).

11.3 Funktionsblöcke für Regler

Inhalt:

Der nachfolgende Abschnitt beschreibt im Detail die Funktionen, die zum Aufbau von Software-Reglern im R360-Controller bereitgestellt werden. Die Funktionen können auch als Basis für die Entwicklung von eigenen Regelungsfunktionen genutzt werden.

11.3.1 Funktion DELAY

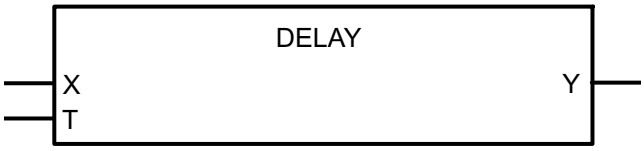
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015

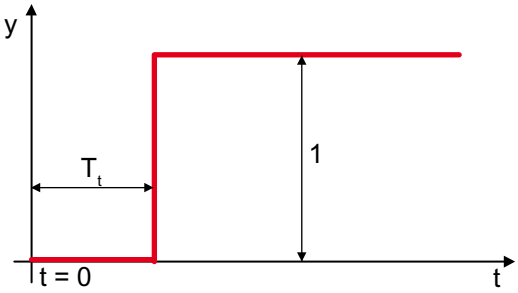
Funktionssymbol:



Beschreibung

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

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



Grafik: Zeitlicher Verlauf von DELAY

HINWEIS

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

Parameter der Funktionseingänge

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

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
Y	WORD	Eingangswert, verzögert um die Zeit T

11.3.2 Funktion PT1

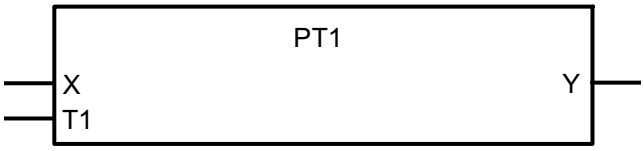
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015

Funktionssymbol:

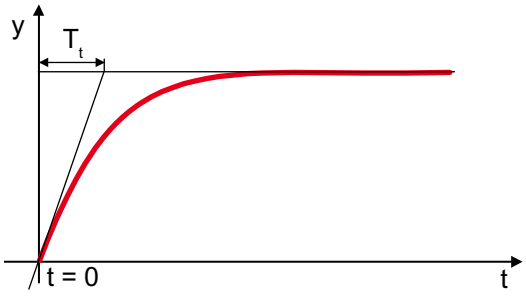


Beschreibung

PT1 organisiert eine Regelstrecke mit Verzögerung 1. Ordnung.

Bei der Funktion 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):



Grafik: Zeitlicher Verlauf bei PT1

Parameter der Funktionseingänge

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

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
Y	INT	Ausgangsvariable

11.3.3 Funktion PID1

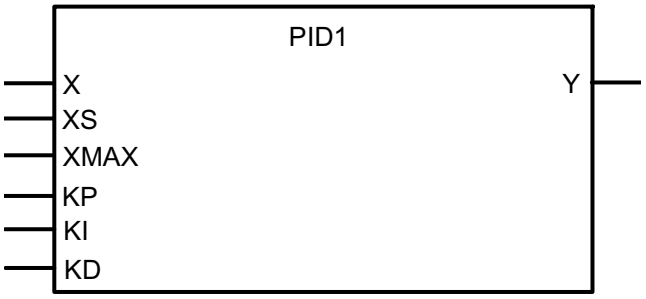
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015

Funktionssymbol:



Beschreibung

PID1 organisiert einen PID-Regler.

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

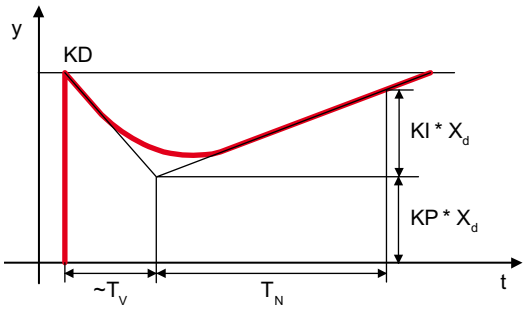
HINWEIS

Die Stellgröße Y ist bereits auf die PWM-Funktion normiert (RELOAD-Wert = 65.535). Beachten Sie dabei die umgekehrte Logik:
65.535 = minimaler Wert
0 = maximaler Wert.

Beachten Sie, dass die Eingangsgrößen KI und KD zykluszeitabhängig sind. Um ein stabiles, reproduzierbares Regelverhalten zu bekommen, sollte die Funktion zeitgesteuert aufgerufen werden.

Wenn X > XS, dann wird die Stellgröße erhöht.
Wenn X < XS, dann wird die Stellgröße reduziert.

Die Stellgröße Y hat folgenden zeitlichen Verlauf:



Grafik: Typische Sprungantwort eines PID-Reglers

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
X	WORD	Istwert
XS	WORD	Sollwert
XMAX	WORD	Maximalwert des Sollwertes
KP	BYTE	Konstante des P-Anteil
KI	BYTE	I-Anteil
KD	BYTE	Proportionalanteil des D-Anteils

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
Y	WORD	Stellgröße

Einstellempfehlung

KP = 50
KI = 30
KD = 5

Bei den oben angegebenen Werten arbeitet der Regler sehr schnell und stabil. Der Regler schwingt bei dieser Einstellung nicht.

► Um den Regler zu optimieren, können die Werte anschließend schrittweise verändert werden.

11.3.4 Funktion PID2

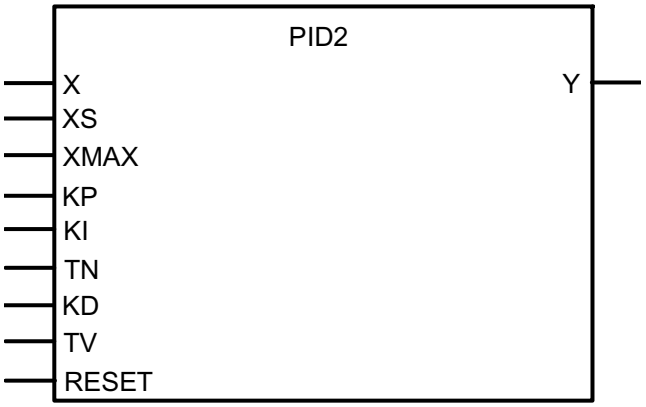
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0032, CR0505
- ExtendedController: CR0200, CR0232
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015

Funktionssymbol:



Beschreibung

PID2 organisiert einen PID-Regler mit Selbstoptimierung.

Die Änderung der Stellgröße eines PID-Reglers setzt sich aus einem proportionalen, integralen und differentialen Anteil zusammen. Die Stellgröße ändert sich zunächst um einen von der Änderungsgeschwindigkeit der Eingangsgröße abhängigen Betrag (**D**ifferential-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 an den Funktionseingängen KP, KI und KD eingegebenen Werte werden intern durch 10 geteilt. Damit kann eine feinere Abstufung erreicht werden (z.B: KP = 17, das entspricht 1,7).

HINWEIS

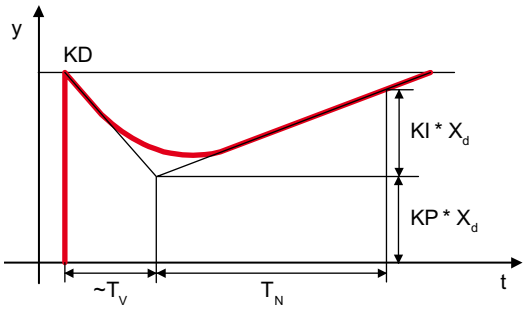
Die Stellgröße Y ist bereits auf die PWM-Funktion normiert (RELOAD-Wert = 65.535). Beachten Sie dabei die umgekehrte Logik:
65.535 = minimaler Wert
0 = maximaler Wert.

Beachten Sie, dass die Eingangsgröße KD zykluszeitabhängig ist. Um ein stabiles, reproduzierbares Regelverhalten zu bekommen, sollte die Funktion zeitgesteuert aufgerufen werden.

Wenn X > XS, dann wird die Stellgröße erhöht.
Wenn X < XS, dann wird die Stellgröße reduziert.

Eine Führungsgröße wird intern zur Stellgröße hinzuaddiert:
Y = Y + 65.536 - (XS / XMAX * 65.536).

Die Stellgröße Y hat folgenden zeitlichen Verlauf.



Grafik: Typische Sprungantwort eines PID-Reglers

Parameter der Funktionseingänge

Name	Datentyp	Beschreibung
X	WORD	Istwert
XS	WORD	Sollwert
XMAX	WORD	Maximalwert des Sollwertes
KP	BYTE	Konstante des **P**roprotional-Anteils (/10)
KI	BYTE	Konstante des **I**ntegral-Anteils (/10)
TN	TIME	Nachstellzeit (Integral-Anteil)
KD	BYTE	Proportionalanteil des **D**ifferential-Anteils (/10)
TV	TIME	Vorhaltezeit (Differential-Anteil)
RESET	BOOL	Funktion zurücksetzen

Parameter der Funktionsausgänge

Name	Datentyp	Beschreibung
Y	WORD	Stellgröße

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 sicher noch kein Schwingen auftritt.
- ► TN bei Bedarf nachjustieren
- ► Nur bei Bedarf D-Anteil hinzufügen: TV ca. 2...10- mal kleiner als TN wählen. KD etwa gleich groß wie KP wählen.

Beachten Sie, dass die maximale Regelabweichung + 127 beträgt. Für ein gutes Regelverhalten sollte dieser Bereich einerseits nicht überschritten, andererseits aber möglichst ausgenutzt werden.

11.3.5 Funktion GLR

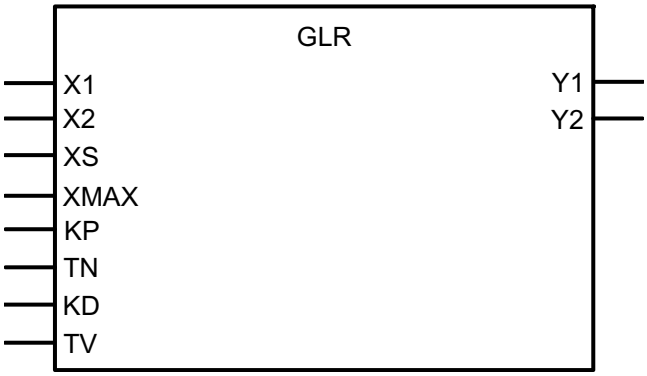
Enthalten in Bibliothek:

`ifm_CRnnnn_Vxxyyzz.LIB`

verfügbar für:

- ClassicController: CR0020, CR0505
- ExtendedController: CR0200
- SmartController: CR2500
- SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
- CabinetController: CR0301, CR0302, CR0303
- Platinensteuerung: CS0015

Funktionssymbol:



Beschreibung

GLR organisiert einen Gleichlauf-Regler.

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, das entspricht 1,7).

Die Stellgröße bezüglich des größeren Istwerts wird jeweils erhöht.
Die Stellgröße bezüglich des kleineren Istwerts entspricht der Führungsgröße.
Führungsgröße = 65536 - (XS / XMAX * 65536).

HINWEIS

Die Stellgrößen Y1 und Y2 sind bereits auf die PWM-Funktion normiert (RELOAD-Wert = 65535).
Beachten Sie dabei die umgekehrte Logik:
65535 = minimaler Wert
0 = maximaler Wert.

Beachten Sie, dass die Eingangsgröße KD zykluszeitabhängig ist. Um ein stabiles, reproduzierbares Regelverhalten zu bekommen, sollte die Funktion zeitgesteuert aufgerufen werden.

Parameter der 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	Proportionalanteil des D-Anteils (/10)
TV	TIME	Vorhaltezeit (D-Anteil)

Parameter der Funktionsausgänge

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

12 Anhang

Inhalt:

Hier stellen wir Ihnen – ergänzend zu den Angaben in den Datenblättern – zusammenfassende Tabellen zur Verfügung.

12.1 Adressbelegung und E/A-Betriebsarten

→ auch Datenblatt

12.1.1 Adressen / Variablen der E/As

Port	IEC-Adresse	E/A-Variable	Bemerkung
	%QB4	I0_MODE	Konfigurations-Byte für %IX0.0
	Merkerbit*)	ERROR_I0	Bit DIAGNOSE für %IX0.0
	%Q5	I1_MODE	Konfigurations-Byte für %IX0.8
	Merkerbit*)	ERROR_I1	Bit DIAGNOSE für %IX0.8
	%QB4	I2_MODE	Konfigurations-Byte für %IX1.0
	Merkerbit*)	ERROR_I2	Bit DIAGNOSE für %IX1.0
	%QB4	I3_MODE	Konfigurations-Byte für %IX1.8
	Merkerbit*)	ERROR_I3	Bit DIAGNOSE für %IX1.8
	%QB0	Q1Q2	Ausgangsbyte 0 (%QX0.00...%QX0.07)
	Merkerbyte*)	ERROR_SHORT_Q1Q2	Fehler-Byte Port 1+2 Kurzschluss (%QX0.00...%QX0.07)
	Merkerbyte*)	ERROR_BREAK_Q1Q2	Fehler-Byte Port 1+2 Unterbrechung (%QX0.00...%QX0.07)

*) IEC-Adressen können sich je nach Steuerungskonfiguration ändern.

12.1.2 Adressbelegung Ein-/Ausgänge

IEC-Adresse	Name EA-Variable	Konfiguration mit Variable	Default-Wert	mögliche Konfiguration
%IX0.0 / %IW2	I0 / ANALOG0	I0_MODE	0	L-digital / CYL0 / FRQ0
%IX0.8 / %IW3	I1 / ANALOG1	I1_MODE	0	L-digital / CYL1 / FRQ1
%IX1.0 / %IW4	I2 / ANALOG2	I2_MODE	0	nur L-digital
%IX1.8 / %IW5	I3 / ANALOG3	I3_MODE	0	nur L-digital
%IW6	ANALOG4	I4_MODE	3	analog U/I
%IW7	ANALOG5	I5_MODE	3	analog U/I
%IW8	ANALOG6	I6_MODE	3	analog U/I
%IW9	ANALOG7	I7_MODE	3	analog U/I
%QX0.0	Q0	-	-	H-digital / PWM / PWM_I
%QX0.8	Q1	-	-	H-digital / PWM / PWM_I
%QX1.0	Q2	-	-	H-digital / PWM / PWM_I
%QX1.8	Q3	-	-	H-digital / PWM / PWM_I

PWM Beschreibung → Kapitel PWM-Signalverarbeitung, Seite 164

PWM_I Beschreibung → Kapitel Stromregelung mit PWM, Seite 177

FRQ/CYL Beschreibung → Kapitel Zählerfunktionen zur Frequenz- und Periodendauermessung, Seite 209

12.1.3 Mögliche Betriebsarten Ein-/Ausgänge

Eingänge	Betriebsart	Konfig.-Wert	Ausgänge	Betriebsart	Konfig.-Wert
I0…I3	IN_DIGITAL	0 (Default)			
	IN_DIAGNOSIC	4			
I0…I1	IN_DIGITAL_FAST	5			
I4…I7	IN_CURRENT	1			
	IN_VOLTAGE10	2			
	IN_VOLTAGE30	3 (Default)			

12.2 Systemmerker

(→ Kapitel Fehlercodes und Diagnoseinformationen, Seite 41)

Systemmerker	Art	Erläuterung
CANx_BAUDRATE	WORD	CAN-Schnittstelle x: eingestellte Baud-Rate
CANx_BUSOFF	BOOL	CAN-Schnittstelle x: Fehler "CAN-Bus off"
CANx_LASTERROR [1])	BYTE	CAN-Schnittstelle x: Fehlernummer der letzten CAN-Übertragung: 0= kein Fehler ≠0 → CAN-Spezifikation → LEC
CANx_WARNING	BOOL	CAN-Schnittstelle x: Warnschwelle erreicht (≥ 96)
DOWNLOADID	WORD	Aktuell eingestellter Download-Identifier
ERROR	BOOL	Bit ERROR setzen
ERROR_BREAK_Qx	BYTE	Leiterbruch-Fehler an der Ausgangsgruppe x
ERROR_IO	BOOL	I/O-Fehler (Sammelbit)
ERROR_Ix	BYTE	Peripherie-Fehler an der Eingangsgruppe x
ERROR_MEMORY	BOOL	Speicher-Fehler
ERROR_POWER	BOOL	Unter-/Überspannungs-Fehler
ERROR_SHORT_Qx	BYTE	Kurzschluss-Fehler an der Ausgangsgruppe x
ERROR_VBBR	BOOL	Versorgungsspannungs-Fehler VBB_R
LED_MODE	WORD	Blinkfrequenz aus der Datenstruktur "LED_MODES"
SERIAL_MODE	BOOL	Serielle Kommunikation einschalten
SERIALBAUDRATE	WORD	Baud-Rate der RS232-Schnittstelle
SUPPLY_VOLTAGE	WORD	Versorgungsspannung
TEST	BOOL	Programmiermodus freigeben

CANx steht für die Nummer der CAN-Schnittstelle (CAN 1...x, abhängig vom Gerät).

Ix oder Qx steht für die Nummer der Ein- oder Ausgangsgruppe (Wort 0...x, abhängig vom Gerät).

[1]) Der Zugriff auf diese Merker erfordert genaue Kenntnisse des CAN-Controllers und wird im Normalfall nicht benötigt.

HINWEIS

Für die Programmierung sollten nur die Symbolnamen genutzt werden, da sich die zugehörigen Merkeradressen bei einer Erweiterung der Steuerungskonfiguration ändern können.

12.3 Übersicht der verwendeten Dateien und Bibliotheken

(Stand: 02.02.2009)

Je nach Gerät und gewünschter Funktion kommen verschiedene Bibliotheken und Dateien zum Einsatz. Teilweise werden sie automatisch geladen oder müssen vom Programmierer eingefügt oder geladen werden.

Installieren der Dateien und Bibliotheken im Gerät:

Werkseinstellung: Das Gerät enthält nur den Bootloader.

► Betriebssystem (*.H86) laden.

► Projekt (*.PRO) im PC anlegen: Target (*.TRG) eintragen.

► (zusätzlich bei Targets vor V05:) Steuerungskonfiguration (*.CFG) festlegen.

> CoDeSys® bindet die zum Target zugehörenden Dateien in das Projekt ein: *.TRG, *.CFG, *.CHM, *.INI, *.LIB.

► Bei Bedarf das Projekt mit weiteren Bibliotheken (*.LIB) ergänzen.

Bestimmte Bibliotheken binden automatisch weitere Bibliotheken in das Projekt ein:
z.B. basieren einige Funktionen in ifm-Bibliotheken (ifm_*.LIB) auf Funktionen in CoDeSys®-Bibliotheken (3S_*.LIB).

12.3.1 Allgemeine Übersicht

Dateiname	Beschreibung und Speicherort *)
ifm_CRnnnn_Vxxyyzz.CFG [1]) ifm_CRnnnn_Vxx.CFG [2])	Steuerungskonfiguration je Gerät nur 1 gerätespezifische Datei enthält: IEC- und symbolische Adressen der Ein- und Ausgänge, der Systemmerker sowie die Speicherverteilung …\CoDeSys V*\Targets\ifm\ifm_CRnnnncfg\Vxxyyzz
CAA-*.CHM	Online-Hilfe je Gerät nur 1 gerätespezifische Datei enthält: Online-Hilfe zu diesem Gerät …\CoDeSys V*\Targets\ifm\Help\… (Sprache)
ifm_CRnnnn_Vxxyyzz.H86	Betriebssystem / Laufzeitsystem (muss bei Erstbenutzung in den Controller / Monitor geladen werden) je Gerät nur 1 gerätespezifische Datei …\CoDeSys V*\Targets\ifm\Library\ifm_CRnnnn
ifm_Browser_CRnnnn.INI	CoDeSys-Bowser-Kommandos (CoDeSys® benötigt die Datei zum Projektstart) je Gerät nur 1 gerätespezifische Datei enthält: Kommandos für Browser in CoDeSys® …\CoDeSys V*\Targets\ifm
ifm_Errors_CRnnnn.INI	CoDeSys-Fehler-Datei (CoDeSys® benötigt die Datei zum Projektstart) je Gerät nur 1 gerätespezifische Datei enthält: gerätespezifische Fehlermeldungen aus CoDeSys® …\CoDeSys V*\Targets\ifm

Dateiname	Beschreibung und Speicherort *)
ifm_CRnnnn_Vxx.TRG	Target-Datei je Gerät nur 1 gerätespezifische Datei enthält: Hardware-Beschreibung für CoDeSys®, z.B.: Speicher, Dateiablageorte …\CoDeSys V*\Targets\ifm
ifm_*_Vxxyyzz.LIB	allgemeine Bibliotheken je Gerät mehrere Dateien möglich …\CoDeSys V*\Targets\ifm\Library
ifm_CRnnnn_Vxxyyzz.LIB	gerätespezifische Bibliothek je Gerät nur 1 gerätespezifische Datei enthält: Funktionen dieses Geräts …\CoDeSys V*\Targets\ifm\Library\ifm_CRnnnn
ifm_CRnnnn_*_Vxxyyzz.LIB	gerätespezifische Bibliotheken je Gerät mehrere Dateien möglich → folgende Tabellen …\CoDeSys V*\Targets\ifm\Library\ifm_CRnnnn

Legende:

*	beliebige Zeichen
CRnnnn	Artikelnummer des Controllers / Monitors
V*	CoDeSys®-Version
Vxx	Versionsnummer der ifm-Software
yy	Release-Nummer der ifm-Software
zz	Patch-Nummer der ifm-Software

¹) gültig für CRnn32 Target-Version bis V01, alle anderen Geräte bis V04
²) gültig für CRnn32 Target-Version ab V02, alle anderen Geräte ab V05

*) Speicherort der Dateien:
System-Laufwerk (C: / D:) \ Programme-Ordner \ ifm electronic

HINWEIS

Es müssen immer die zum gewählten Target passenden Software-Stände zum Einsatz kommen:

- des Betriebssystems (CRnnnn_Vxxyyyzz.H86),
- der Steuerungskonfiguration (CRnnnn_Vxx.CFG),
- der Gerätebibliothek (CRnnnn_Vxxyyyzz.LIB)
- und der weiteren Dateien
  (→ Kapitel Übersicht der verwendeten Dateien und Bibliotheken, Seite 288).

CRnnnn Geräte-Artikelnummer
Vxx: 00...99 Target-Versionsnummer
yy: 00...99 Release-Nummer
zz: 00...99 Patch-Nummer

Dabei müssen der Basisdateiname (z.B. "CR0032") und die Software-Versionsnummer "xx" (z.B. "02") überall den gleichen Wert haben! Andernfalls geht der Controller in den STOPP-Zustand

Die Werte für "yy" (Release-Nummer) und "zz" (Patch-Nummer) müssen **nicht** übereinstimmen.

Außerdem beachten: Folgende Dateien müssen ebenfalls geladen sein:

- die zum Projekt erforderlichen internen Bibliotheken (in IEC1131 erstellt),
- die Konfigurationsdateien (`*.CFG`)
- und die Target-Dateien (`*.TRG`).

12.3.2 Wozu dienen die einzelnen Dateien und Bibliotheken?

Die nachfolgende Übersicht zeigt, welche Dateien/Bibliotheken mit welchem Gerät eingesetzt werden können und dürfen. Dateien/Bibliotheken, die in dieser Liste nicht aufgeführt werden, können nur unter bestimmten Bedingungen eingesetzt werden oder die Funktionalität wurde noch nicht getestet.

Dateien für Betriebssystem / Laufzeitsystem

Dateiname	Funktion	verfügbar für:
ifm_CRnnnn_Vxxyyzz.H86	Betriebssystem / Laufzeitsystem	alle ecomat*mobil*-Controller alle PDM360 Monitore
ifm_Browser_CRnnnn.INI	CoDeSys-Browser-Kommandos	alle ecomat*mobil*-Controller alle PDM360 Monitore
ifm_Errors_CRnnnn.INI	CoDeSys-Fehler-Datei	alle ecomat*mobil*-Controller alle PDM360 Monitore

Target-Datei

Dateiname	Funktion	verfügbar für:
ifm_CRnnnn_Vxx.TRG	Target-Datei	alle ecomat*mobil*-Controller alle PDM360 Monitore

Steuerungskonfigurations-Datei

Dateiname	Funktion	verfügbar für:
ifm_CRnnnn_Vxxyyzz.CFG	Steuerungskonfiguration	alle ecomat*mobil*-Controller alle PDM360 Monitore

ifm-Gerätebibliotheken

Dateiname	Funktion	verfügbar für:
ifm_CRnnnn_Vxxyyzz.LIB	gerätespezifische Bibliothek	alle ecomat*mobil*-Controller alle PDM360 Monitore
ifm_CR0200_MSTR_Vxxyyzz.LIB	Bibliothek ohne Extended-Funktionen	ExtendedController: CR0200
ifm_CR0200_SMALL_Vxxyyzz.LIB	Bibliothek ohne Extended-Funktionen, reduzierter Funktionsumfang	ExtendedController: CR0200

ifm-CANopen-Hilfsbibliotheken Master/Slave

Diese Bibliotheken setzen auf CoDeSys®-Bibliotheken (3S-CANopen-Funktionen) auf und stellen sie dem Anwender übersichtlich zur Verfügung.

Dateiname	Funktion	verfügbar für:
ifm_CRnnnn_CANopenMaster_Vn.LIB	CANopen Master Emergency- und Status-Handler	alle ecomatmobil-Controller alle PDM360 Monitore
ifm_CRnnnn_CANopenSlave_Vn.LIB	CANopen Slave Emergency- und Status-Handler	alle ecomatmobil-Controller alle PDM360 Monitore
ifm_CANx_SDO_Vxxyyzz.LIB	CANopen SDO Read und SDO Write	PDM360: CR1050, CR1051, CR1060 PDM360 compact: CR1052, CR1053, CR1055, CR1056

CoDeSys®-CANopen-Bibliotheken

Dateiname	Funktion	verfügbar für:
3S_CanDrvOptTable.LIB [1] 3S_CanDrvOptTableEx.LIB [2]	CANopen Treiber	alle ecomatmobil Controller PDM360 smart: CR1070, CR1071
3S_CanDrv.LIB		PDM360: CR1050, CR1051, CR1060 PDM360 compact: CR1052, CR1053, CR1055, CR1056
3S_CANopenDeviceOptTable.LIB [1] 3S_CANopenDeviceOptTableEx.LIB [2]	CANopen Slave-Treiber	alle ecomatmobil Controller PDM360 smart: CR1070, CR1071
3S_CANopenDevice.LIB		PDM360: CR1050, CR1051, CR1060 PDM360 compact: CR1052, CR1053, CR1055, CR1056
3S_CANopenManagerOptTable.LIB [1] 3S_CANopenManagerOptTableEx.LIB [2]	CANopen Netzwerkmanager	alle ecomatmobil Controller PDM360 smart: CR1070, CR1071
3S_CANopenManager.LIB		PDM360: CR1050, CR1051, CR1060 PDM360 compact: CR1052, CR1053, CR1055, CR1056
3S_CANopenMasterOptTable.LIB [1] 3S_CANopenMasterOptTableEx.LIB [2]	CANopen Master	alle ecomatmobil Controller PDM360 smart: CR1070, CR1071
3S_CANopenMaster.LIB		PDM360: CR1050, CR1051, CR1060 PDM360 compact: CR1052, CR1053, CR1055, CR1056

Dateiname	Funktion	verfügbar für:
3S_CANopenNetVarOptTable.LIB [1] 3S_CANopenNetVarOptTableEx.LIB [2]	Treiber für Netzwerkvariablen	alle ecomat*mobil* Controller PDM360 smart: CR1070, CR1071
3S_CANopenNetVar.LIB		PDM360: CR1050, CR1051, CR1060 PDM360 compact: CR1052, CR1053, CR1055, CR1056

[1]) gültig für CRnn32 Target-Version bis V01, alle anderen Geräte bis V04
[2]) gültig für CRnn32 Target-Version ab V02, alle anderen Geräte ab V05

spezielle ifm-Bibliotheken

Dateiname	Funktion	verfügbar für:
ifm_J1939_Vxxyyzz.LIB	J1939 Kommunikationsfunktionen	Bis V04 für: ClassicController: CR0020, CR0505 ExtendedController: CR0200 SmartController: CR2500 SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201 CabinetController: CR0303
ifm_J1939_x_Vxxyyzz.LIB	J1939 Kommunikationsfunktionen	ClassicController: CR0032 ExtendedController: CR0232 ab V05 für: ClassicController: CR0020, CR0505 ExtendedController: CR0200 SmartController: CR2500 SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201 CabinetController: CR0303 PDM360 smart: CR1070, CR1071

Dateiname	Funktion	verfügbar für:
ifm_CANx_LAYER2_Vxxyyzz.LIB	CAN-Funktionen auf Basis Layer 2: CAN Transmit, CAN Receive	PDM360: CR1050, CR1051, CR1060 PDM360 compact: CR1052, CR1053, CR1055, CR1056
ifm_CAN1E_ Vxxyyzz.LIB	Stellt den CAN-Bus von 11 Bit auf 29 Bit um	PDM360 smart (bis V04): CR1070, CR1071
ifm_CAN1_EXT_ Vxxyyzz.LIB	Stellt den CAN-Bus von 11 Bit auf 29 Bit um	ab V05 für: ClassicController: CR0020, CR0505 ExtendedController: CR0200 SmartController: CR2500 SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201 CabinetController: CR0301, CR0302, CR0303 Platinensteuerung: CS0015 PDM360 smart: CR1070, CR1071
ifm_CAMERA_O2M_ Vxxyyzz.LIB	Kamera-Funktionen	PDM360: CR1051
CR2013AnalogConverter.LIB	Analogwertkonvertierung für E/A-Modul CR2013	alle ecomat*mobil*-Controller alle PDM360 Monitore
ifm_Hydraulic_Vxxyyzz.LIB	Hydraulikfunktionen für Controller	ClassicController: CR0020, CR0032, CR0505 ExtendedController: CR0200, CR0232 SmartController: CR2500 SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
ifm_SafetyIO_Vxxyyzz.LIB	Sicherheitsfunktionen	SafetyController: CR7020, CR7021, CR7505, CR7506, CR7200, CR7201
ifm_PDM_Util_Vxxyyzz.LIB	Hilfsfunktionen PDM	PDM360: CR1050, CR1051, CR1060 PDM360 compact: CR1052, CR1053, CR1055, CR1056
ifm_PDMsmart_Util_Vxxyyzz.LIB	Hilfsfunktionen PDM	PDM360 smart: CR1070, CR1071
ifm_PDM_Input_Vxxyyzz.LIB	alternative Eingabefunktionen PDM	alle PDM360 Monitore

Dateiname	Funktion	verfügbar für:
ifm_PDM_Init_Vxxyyzz.LIB	Initialisierungsfunktion PDM360 smart	PDM360 smart: CR1070, CR1071
ifm_PDM_File_Vxxyyzz.LIB	Dateifunktionen PDM360	PDM360: CR1050, CR1051, CR1060 PDM360 compact: CR1052, CR1053, CR1055, CR1056
Instrumente_x.LIB	vordefinierte Anzeige-Instrumente	alle PDM360 Monitore
Symbols_x.LIB	vordefinierte Symbole	PDM360: CR1050, CR1051, CR1060 PDM360 compact: CR1052, CR1053, CR1055, CR1056
Segment_x.LIB	vordefinierte 7-Segment-Anzeigen	PDM360: CR1050, CR1051, CR1060 PDM360 compact: CR1052, CR1053, CR1055, CR1056

Weitere Bibliotheken auf Anfrage.

13 Abkürzungen und Begriffe

A

Adresse

Das ist der „Name“ des Teilnehmers im Bus. Alle Teilnehmer benötigen eine unverwechselbare, eindeutige Adresse, damit der Austausch der Signale fehlerfrei funktioniert.

Anforderungsrate rd

Die Anforderungsrate r_d ist die Häufigkeit je Zeiteinheit von Anforderungen an eine sicherheitsgerichtete Reaktion eines SRP/CS.

Anleitung

Übergeordnetes Wort für einen der folgenden Begriffe:
Montageanleitung, Datenblatt, Benutzerinformation, Bedienungsanleitung, Gerätehandbuch, Installationsanleitung, Onlinehilfe, Systemhandbuch, Programmierhandbuch, usw.

Applikations-Software

Software, die speziell für die Applikation (Anwendung) vom Hersteller in die Maschine programmiert wird. Die Software enthält üblicherweise logische Sequenzen, Grenzwerte und Ausdrücke zum Steuern der entsprechenden Ein- und Ausgänge, Berechnungen und Entscheidungen.

Für sicherheitsrelevante Teile von Steuerungen (→SRP/CS) müssen spezielle Anforderungen erfüllt sein.
→ Programmiersprache, sicherheitsrelevant

Architektur

Spezifische Konfiguration von Hardware- und Software-Elementen in einem System.

Ausfall

Ausfall ist die Beendigung der Fähigkeit einer Einheit, eine geforderte Funktion zu erfüllen.

Nach einem Ausfall hat die Einheit einen Fehler. Der Ausfall ist ein Ereignis, der Fehler ein Zustand.

Der so definierte Begriff kann nicht auf Einheiten angewendet werden, die nur aus Softwrae bestehen.

Ausfall, gefahrbringend

Ein gefahrbringender Ausfall hat das Potential, das SRP/CS in einen gefährlichen Zustand oder eine Fehlfunktion zu bringen. Ob dieses Potential bemerkt werden kann oder nicht, hängt von der Architektur des Systems ab. In einem redundanten System wird ein gefährlicher Hardware-Ausfall weniger wahrscheinlich zu einem gefährlichen Ausfall des Gesamtsystems führen.

Ausfall, systematischer

Ein systematischer Ausfall ist ein Ausfall mit deterministischem (nicht zufälligem) Bezug zu einer bestimmten Ursache. Der systematische Ausfall kann nur beseitigt werden durch Änderung des Entwurfs oder des Herstellprozesses, Betriebsverfahren, Dokumentation oder zugehörigen Faktoren.

Eine Instandsetzung ohne Änderung des Systems wird den Grund des systematischen Ausfalls in der Regel nicht beseitigen.

B

Baud

Baud, Abk.: Bd = Maßeinheit für die Geschwindigkeit bei der Datenübertragung. Baud ist nicht zu verwechseln mit "bits per second" (bps, Bit/s). Baud gibt zwar die Anzahl von Zustandsänderungen (Schritte, Takte) pro Sekunde auf einer Übertragungsstrecke an. Aber es ist nicht festgelegt, wie viele Bits pro Schritt übertragen werden. Der Name Baud geht auf den französischen Erfinder J. M. Baudot zurück, dessen Code für Telexgeräte verwendet wurde.

1 MBd = 1024 x 1024 Bd = 1 048 576 Bd

Bestimmungsgemäße Verwendung

Das ist die Verwendung eines Produkts in Übereinstimmung mit den in der Anleitung bereitgestellten Informationen.

Betriebsdauer, mittlere

Mean **t**ime **b**etween **f**ailures (MTBF) = mittlere Betriebsdauer zwischen Ausfällen.
Ist der Erwartungswert der Betriebsdauer zwischen zwei aufeinanderfolgenden Ausfällen von Einheiten, die instand gesetzt werden.

HINWEIS: Für Einheiten, die NICHT instandgesetzt werden, ist der Erwartungswert (Mittelwert) der Verteilung von Lebensdauern die mittlere Lebensdauer →MTTF.

Betriebssystem

Grundprogramm im Gerät, stellt die Verbindung her zwischen der Hardware des Gerätes und der Anwender-Software.

Bus

Serielle Datenübertragung mehrerer Teilnehmer an derselben Leitung.

C

CAN

CAN = **C**ontroller **A**rea **N**etwork

CAN gilt als Feldbussystem für größere Datenmengen, das prioritätengesteuert arbeitet. Gibt es in verschiedenen Varianten z.B. als "CANopen" oder "CAN in Automation (CiA)."

CCF

Common **c**ause **f**ailure = Ausfall in Folge von gemeinsamer Ursache
Ausfälle verschiedener Einheiten aufgrund eines gemeinsamen Ereignisses, wobei diese Ausfälle nicht auf gegenseitige Ursachen beruhen.

CiA

CiA = CAN in Automation e.V.

Anwender- und Herstellerorganisation in Deutschland / Erlangen. Definitions- und Kontrollorgan für CAN und CAN-basierende Netzwerkprotokolle.

Homepage → http://www.can-cia.org

CiA DS 304

DS = **D**raft **S**tandard

CAN-Geräteprofil CANopen-Safety für sicherheitsgerichtete Kommunikation.

CiA DS 401

DS = **D**raft **S**tandard

CAN-Geräteprofil für digitale und analoge E/A-Baugruppen

CiA DS 402

DS = **D**raft **S**tandard

CAN-Geräteprofil für Antriebe

CiA DS 403

DS = **D**raft **S**tandard

CAN-Geräteprofil für Bediengeräte

CiA DS 404

DS = **D**raft **S**tandard

CAN-Geräteprofil für Messtechnik und Regler

CiA DS 405

DS = **D**raft **S**tandard

Spezifikation zur Schnittstelle zu programmierbaren Steuerungen (IEC 61131-3)

CiA DS 406

DS = **D**raft **S**tandard

CAN-Geräteprofil für Drehgeber / Encoder

CiA DS 407

DS = **D**raft **S**tandard

CAN-Applikations-Profil für den öffentlichen Nahverkehr

COB-ID

COB = **C**ommunication **Ob**ject = Kommunikationsobjekt
ID = **Id**entifier = Kennung

Über den COB-ID unterscheiden die Teilnehmer die verschiedenen auszutauschenden Nachrichten.

CoDeSys

CoDeSys® ist eingetragene Marke der 3S – Smart Software Solutions GmbH, Deutschland

"CoDeSys for Automation Alliance" vereinigt Firmen der Automatisierungsindustrie, deren Hardwaregeräte alle mit dem weit verbreiteten IEC 61131-3 Entwicklungswerkzeug CoDeSys® programmiert werden.

Homepage → http://www.3s-software.com/ (http://www.3s-software.com/)

D

DC

Direct **C**urrent = Gleichstrom

DC

Diagnostic **C**overage = Diagnose-Deckungsgrad

Der Diagnose-Deckungsgrad ist das Maß für die Wirksamkeit der Diagnose als Verhältnis der Ausfallrate der bemerkten gefahrbringenden Ausfälle und der Ausfallrate der gesamten gefahrbringenden Ausfälle:

Formel: DC = Ausfallrate bemerkte gefahrbringende Ausfälle / Ausfallrate gesamte gefahrbringende Ausfälle

Bezeichnung	Bereich
kein	DC < 60 %
niedrig	60 % ≤ DC < 90 %
mittel	90 % ≤ DC < 99 %
hoch	99 % ≤ DC

Tabelle: Diagnose-Deckungsgrad DC

Für die in der Tabelle gezeigten Grenzwerte wird eine Genauigkeit von 5 % angenommen.

Der Diagnose-Deckungsgrad kann für das gesamte sicherheitsgerichtete System ermittelt werden oder nur für Teile des sicherheitsgerichteten Systems.

Diagnose-Deckungsgrad

Diagnostic **C**overage = Diagnose-Deckungsgrad
Der Diagnose-Deckungsgrad ist das Maß für die Wirksamkeit der Diagnose als Verhältnis der Ausfallrate der bemerkten gefahrbringenden Ausfälle und der Ausfallrate der gesamten gefahrbringenden Ausfälle:

Formel: DC = Ausfallrate bemerkte gefahrbringende Ausfälle / Ausfallrate gesamte gefahrbringende Ausfälle

Bezeichnung	Bereich
kein	DC < 60 %
niedrig	60 % ≤ DC < 90 %
mittel	90 % ≤ DC < 99 %
hoch	99 % ≤ DC

Tabelle: Diagnose-Deckungsgrad DC

Für die in der Tabelle gezeigten Grenzwerte wird eine Genauigkeit von 5 % angenommen.

Der Diagnose-Deckungsgrad kann für das gesamte sicherheitsgerichtete System ermittelt werden oder nur für Teile des sicherheitsgerichteten Systems.

Dither

to dither (engl.) = schwanken / zittern

Dither ist ein Bestandteil der PWM-Signale zum Ansteuern von Hydraulik-Ventilen. Für die elektromagnetischen Antriebe von Hydraulik-Ventilen hat sich herausgestellt, dass sich die Ventile viel besser regeln lassen, wenn das Steuersignal (PWM-Impulse) mit einer bestimmten Frequenz der PWM-Frequenz überlagert wird. Diese Dither-Frequenz muss ein ganzzahliger Teil der PWM-Frequenz sein.
→ Kapitel Was ist der Dither?, Seite 185

diversitär

Unter Diversität (Vielfalt) versteht man in der Technik eine Strategie zur Erhöhung der Ausfallsicherheit.

Dabei werden Systeme redundant ausgelegt, allerdings werden bewusst verschiedene Realisierungen und keine baugleichen Einzelsysteme verwendet. Man geht davon aus, dass Systeme, die das Gleiche leisten, aber unterschiedlich realisiert sind, auch gegen unterschiedliche Störungen empfindlich oder unempfindlich sind und daher möglichst nicht alle gleichzeitig ausfallen.

Die konkrete Realisierung kann je nach Einsatzgebiet und geforderter Sicherheit unterschiedlich aussehen:

- Verwendung von Bauteilen verschiedener Hersteller,
- Nutzung unterschiedlicher Protokolle zur Steuerung von Geräten,
- Verwendung komplett unterschiedlicher Technologien, beispielsweise einer elektrischen und einer pneumatischen Steuerung,
- Verwendung unterschiedlicher Messmethoden (Strom, Spannung),
- zwei Kanäle mit gegenläufigen Werteverläufen:
  Kanal A: 0...100 %
  Kanal B: 100...0 %

E

EDS-Datei

EDS = **E**lectronic **D**ata **S**heet = elektronisch hinterlegtes Datenblatt, z.B. für:

- Datei für das Objektverzeichnis im Master
- CANopen-Gerätebeschreibungen

Via EDS können vereinfacht Geräte und Programme ihre Spezifikationen austauschen und gegenseitig berücksichtigen.

Embedded Software

System-Software, Grundprogramm im Gerät, praktisch das Betriebssystem.

Die Firmware stellt die Verbindung her zwischen der Hardware des Gerätes und der Anwender-Software. Diese Software wird vom Hersteller der Steuerung als Teil des Systems geliefert und kann vom Anwender nicht verändert werden.

EMCY

Abkürzung für Emergency (engl.) = Notfall

EMV

EMV = **E**lektro-**M**agnetische **V**erträglichkeit

Gemäß der EG-Richtlinie (2004/108/EG) zur elektromagnetischen Verträglichkeit (kurz EMV-Richtlinie) werden Anforderungen an die Fähigkeit von elektrischen und elektronischen Apparaten, Anlagen, Systemen oder Bauteilen gestellt, in der vorhandenen elektromagnetischen Umwelt zufriedenstellend zu arbeiten. Die Geräte dürfen ihre Umgebung nicht stören und dürfen sich von äußerlichen elektromagnetischen Störungen nicht ungünstig beeinflussen lassen.

Erstfehler-Eintrittszeit

Das ist die Zeit bis zum ersten Versagen eines Sicherheitselements.

Im Zeitraum von maximal 30 s wird durch die internen Überwachungs- und Testroutinen die Steuerung vom Betriebssystem überprüft.

Diese „Testzykluszeit“ muss kleiner sein als die statistische Erstfehler-Eintrittszeit für die Applikation.

Ethernet

Das Ethernet ist eine weit verbreitete, herstellerneutrale Technologie, mit der im Netzwerk Daten mit einer Geschwindigkeit von 10 oder 100 Millionen Bit pro Sekunde (Mbps) übertragen werden können. Das Ethernet gehört zu der Familie der sogenannten „bestmöglichen Datenübermittlung“ auf einem nicht exklusiven Übertragungsmedium. 1972 entwickelt, wurde das Konzept 1985 als IEEE 802.3 spezifiziert.

EUC

EUC = „equipment under control" (kontrollierte Einrichtung)

EUC ist eine Einrichtung, Maschine, Gerät oder Anlage, verwendet zur Fertigung, Stoffumformung, zum Transport, zu medizinischen oder anderen Tätigkeiten (→ IEC/EN 61508-4, Abschnitt 3.2.3). Das EUC umfasst also alle Einrichtungen, Maschinen, Geräte oder Anlagen, die Gefährdungen verursachen können und für die sicherheitsgerichtete Systeme erforderlich sind.

Falls eine vernünftigerweise vorhersehbare Aktivität oder Inaktivität zu durch das EUC verursachten Gefährdungen mit unvertretbarem Risiko führt, sind Sicherheitsfunktionen erforderlich, um einen sicheren Zustand für das EUC zu erreichen oder aufrecht zu erhalten. Diese Sicherheitsfunktionen werden durch ein oder mehrere sicherheitsgerichtete Systeme ausgeführt.

F

Fehlanwendung

Das ist die Verwendung eines Produkts in einer Weise, die vom Konstrukteur nicht vorgesehen ist. Eine Fehlanwendung führt meist zu einer Gefährdung von Personen oder Sachen.

Vor vernünftigerweise, vorhersehbaren Fehlanwendungen muss der Hersteller des Produkts in seinen Benutzerinformationen warnen.

Fehler

Ein Fehler ist die Unfähigkeit einer Einheit, eine geforderte Funktion auszuführen.

Kein Fehler ist diese Unfähigkeit während vorbeugender Wartung oder anderer geplanter Handlungen oder aufgrund des Fehlers externer Mittel.

Ein Fehler ist oft das Resultat eines Ausfalls der Einheit selbst, kann aber ohne vorherigen Ausfall bestehen.

In der ISO 13849-1 ist mit "Fehler" der "zufällige Fehler" gemeint.

Fehler-Toleranzzeit

Das ist die maximale Zeit, die zwischen dem Entstehen eines Fehlers und der Einnahme des sicheren Zustandes in der Applikation vergehen darf, ohne dass eine Gefahr für Personen zu befürchten ist.

Dabei ist die maximale Zykluszeit des Applikations-Programms (im ungünstigsten Fall 100 ms, → Watchdog, Seite 45) und die möglichen Verzögerungs- und Reaktionszeiten durch Abschaltglieder zu berücksichtigen.

Die sich daraus ergebende Gesamtzeit muss kleiner sein als die Fehler-Toleranzzeit der Applikation.

Firmware

System-Software, Grundprogramm im Gerät, praktisch das Betriebssystem.

Die Firmware stellt die Verbindung her zwischen der Hardware des Gerätes und der Anwender-Software. Diese Software wird vom Hersteller der Steuerung als Teil des Systems geliefert und kann vom Anwender nicht verändert werden.

Funktionale Sicherheit

Teil der Gesamtsicherheit, bezogen auf das →EUC und das EUC-Leit- oder Steuerungssystem, die von der korrekten Funktion des elektrischen oder elektronischen sicherheitsgerichteten Systems, sicherheitsgerichteten Systemen anderer Technologien und externer Einrichtungen zur Risikominderung abhängt.

G

Gebrauchsdauer Tm

Die Gebrauchsdauer T_M ist der Zeitraum, der die vorgegebene Verwendung der SRP/CS abdeckt.

Gefährdung

Mit Gefährdung bezeichnet man eine potentielle Schadensquelle.

Man unterscheidet den Ursprung der Gefährdung, z.B.:
- mechanische Gefährdung,
- elektrische Gefährdung,

oder die Art des zu erwartenden Schadens, z.B.:
- Gefährdung durch elektrischen Schlag,
- Gefährdung durch Schneiden,
- Gefährdung durch Vergiftung.

Die Gefährdung im Sinne dieser Definition ist bei der bestimmungsgemäßen Verwendung der Maschine entweder dauerhaft vorhanden, z.B.:
- Bewegung von gefährdenden beweglichen Teilen,
- Lichtbogen beim Schweißen,
- ungesunde Körperhaltung,
- Geräusch-Emission,
- hohe Temperatur,

oder die Gefährdung kann unerwartet auftreten, z.B.:
- Explosion,
- Gefährdung durch Quetschen als Folge eines unbeabsichtigten / unerwarteten Anlaufs,
- Herausschleudern als Folge eines Bruchs,
- Stürzen als Folge von Geschwindigkeitsänderung.

H

Heartbeat

Heartbeat (engl.) = Herzschlag

Die Teilnehmer senden regelmäßig kurze Signale. So können die anderen Teilnehmer prüfen, ob ein Teilnehmer ausgefallen ist. Dazu ist kein Master erforderlich.

I

ID

ID = **Id**entifier = Kennung

Name zur Unterscheidung der an einem System angeschlossenen Geräte / Teilnehmer oder der zwischen den Teilnehmern ausgetauschten Nachrichtenpakete.

IP-Adresse

IP = **I**nternet **P**rotocol = Internet-Protokoll

Die IP-Adresse ist eine Nummer, die zur eindeutigen Identifizierung eines Internet-Teilnehmers notwendig ist. Zur besseren Übersicht wird die Nummer in 4 dezimalen Werten geschrieben, z. B. 127.215.205.156.

K

Kategorie (CAT)

Einstufung der sicherheitsrelevante Teile einer Steuerung bezüglich ihres Widerstandes gegen Fehler und ihres nachfolgenden Verhaltens bei einem Fehler. Diese Sicherheit wird erreicht durch die Struktur der Anordnung der Teile, die Fehlererkennung und/oder ihre Zuverlässigkeit. (→ EN 954)

Klemme 15

Klemme 15 ist in Fahrzeugen die vom Zündschloss geschaltete Plusleitung.

L

Lebensdauer, mittlere

Mean **t**ime **t**o **f**ailure (MTTF) = mittlere Dauer bis zum Ausfall oder: mittlere Lebensdauer. Die $MTTF_d$ ist die erwartete mittlere Zeit bis zum gefahrbringenden Ausfall.

Bezeichnung	Bereich
niedrig	3 Jahre $\leq$ $MTTF_d$ < 10 Jahre
mittel	10 Jahre $\leq$ $MTTF_d$ < 30 Jahre
hoch	30 Jahre $\leq$ $MTTF_d$ $\leq$ 100 Jahre

Tabelle: Mittlere Zeit jedes Kanals bis zum gefahrbringenden Ausfall $MTTF_d$

LED

LED = **L**ight **E**mitting **D**iode = Licht aussendende Diode

Leuchtdiode, auch Luminiszenzdiode, ein elektronisches Element mit hoher, farbiger Leuchtkraft auf kleinem Volumen bei vernachlässigbarer Verlustleistung.

M

MAC-ID

MAC = **M**anufacturer‘s **A**ddress **C**ode = Hersteller-Seriennummer

→ID = **Id**entifier = Kennung

Jede Netzwerkkarte verfügt über eine so genannte MAC-Adresse, ein unverwechselbarer, auf der ganzen Welt einzigartiger Zahlencode – quasi eine Art Seriennummer. So eine MAC-Adresse ist eine Aneinanderreihung von 6 Hexadezimalzahlen, etwa „00-0C-6E-D0-02-3F“.

Master

Wickelt die komplette Organisation auf dem Bus ab. Der Master entscheidet über den zeitlichen Buszugriff und fragt die →Slaves zyklisch ab.

MTBF

Mean **t**ime **b**etween **f**ailures (MTBF) = mittlere Betriebsdauer zwischen Ausfällen.
Ist der Erwartungswert der Betriebsdauer zwischen zwei aufeinanderfolgenden Ausfällen von Einheiten, die instand gesetzt werden.

HINWEIS: Für Einheiten, die NICHT instandgesetzt werden, ist der Erwartungswert (Mittelwert) der Verteilung von Lebensdauern die mittlere Lebensdauer →MTTF.

MTTF

Mean **t**ime **t**o **f**ailure (MTTF) = mittlere Dauer bis zum Ausfall oder: mittlere Lebensdauer.

MTTFd

Mean **t**ime **t**o **f**ailure (MTTF) = mittlere Dauer bis zum Ausfall oder: mittlere Lebensdauer. Die $MTTF_d$ ist die erwartete mittlere Zeit bis zum gefahrbringenden Ausfall.

Bezeichnung	Bereich
niedrig	3 Jahre $\leq$ $MTTF_d$ < 10 Jahre
mittel	10 Jahre $\leq$ $MTTF_d$ < 30 Jahre
hoch	30 Jahre $\leq$ $MTTF_d$ $\leq$ 100 Jahre

Tabelle: Mittlere Zeit jedes Kanals bis zum gefahrbringenden Ausfall $MTTF_d$

Muting

Mit Muting bezeichnet man die vorübergehende und automatische Unterdrückung einer Sicherheitsfunktion durch das SRP/CS.

Beispiel: Der Sicherheits-Lichtvorhang ist überbrückt, wenn die schließenden Werkzeuge unter einen fingersicheren Abstand zueinander gelangt sind. Die bedienende Person kann nun gefahrlos an die Maschine herantreten und das Werkstück führen.

N

NMT

NMT = **N**etwork **M**anagemen**t** = Netzwerk-Verwaltung (hier: im CAN-Bus)

Der NMT-Master steuert die Betriebszustände der NMT-Slaves.

Node

Node (engl.) = Knoten. Damit ist ein Teilnehmer im Netzwerk gemeint.

Node Guarding

Node (engl.) = Knoten, hier: Netzwerkteilnehmer
Guarding (engl.) = Schutz

Parametrierbare, zyklische Überwachung von jedem entsprechend konfigurierten Slave. Der Master prüft, ob die Slaves rechtzeitig antworten. Die Slaves prüfen, ob der Master regelmäßig anfragt. Somit können ausgefallene Netzwerkteilnehmer schnell erkannt und gemeldet werden.

O

Obj / Objekt

Oberbegriff für austauschbare Daten / Botschaften innerhalb des CANopen-Netzwerks.

Objektverzeichnis

Das **Ob**jekt**v**erzeichnis OBV enthält alle CANopen-Kommunikationsparameter eines Gerätes, sowie gerätespezifische Parameter und Daten.

OBV

Das **Ob**jekt**v**erzeichnis OBV enthält alle CANopen-Kommunikationsparameter eines Gerätes, sowie gerätespezifische Parameter und Daten.

operational

Operational (engl.) = betriebsbereit

Betriebszustand eines CANopen-Teilnehmers. In diesem Modus können SDOs, NMT-Kommandos und PDOs übertragen werden.

P

PC-Karte

→ PCMCIA-Karte

PCMCIA-Karte

PCMCIA = Personal Computer Memory Card International Association, ein Standard für Erweiterungskarten mobiler Computer.
Seit der Einführung des Cardbus-Standards 1995 werden PCMCIA-Karten auch als PC-Karte (engl.: PC Card) bezeichnet.

PDO

PDO = **P**rocess **D**ata **O**bject = Nachrichten-Objekt mit Prozessdaten.

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 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.

Diese Dienste sind vom Protokoll her nicht bestätigte Dienste, d.h. es gibt keine Kontrolle, ob die Nachricht auch beim Empfänger ankommt. Netzwerkvariablen-Austausch entspricht einer "1-zu-n-Verbindung" (1 Sender zu n Empfängern).

Performance-Level

Performance-**L**evel
Ist nach ISO 13849-1 eine Einstufung (PL a...e) der Fähigkeit von sicherheitsrelevanten Teilen einer Steuerung, eine Sicherheitsfunktion unter vorhersehbaren Bedingungen auszuführen.

PES

Programable **e**lectronic **s**ystem = Programmierbares elektronisches System
Ein programmierbares elektronisches System ist ein System ...
- zur Steuerung, zum Schutz oder zur Überwachung,
- auf der Basis einer oder mehrerer programmierbarer Geräte,
- einschließlich aller Elemente dieses Systems, wie Ein- und Ausgabegeräte.

Piktogramm

Piktogramme sind bildhafte Symbole, die eine Information durch vereinfachte grafische Darstellung vermitteln.

→ Kapitel Was bedeuten die Symbole und Formatierungen?, Seite 7

PL

Performance-**L**evel
Ist nach ISO 13849-1 eine Einstufung (PL a...e) der Fähigkeit von sicherheitsrelevanten Teilen einer Steuerung, eine Sicherheitsfunktion unter vorhersehbaren Bedingungen auszuführen.

PLr

Mit dem "erforderlichen Performance-Level" PR_r wird nach ISO 13849 die erforderliche Risikominderung für jede Sicherheitsfunktion erreicht.

Für jede gewählte Sicherheitsfunktion, die durch ein SRP/CS ausgeführt wird, muss ein PR_r festgelegt und dokumentiert werden. Die Bestimmung des PR_r ist das Ergebnis der Risikobeurteilung, bezogen auf den Anteil der Risikominderung durch die sicherheitsrelevanten Teile der Steuerung.

Pre-Op

Pre-Op = Preoperational mode (engl.) = Zustand vor betriebsbereit

Betriebszustand eines CANopen-Teilnehmers. Nach dem Einschalten der Versorgungsspannung geht jeder Teilnehmer automatisch in diesem Zustand. Im CANopen-Netz können in diesem Modus nur SDOs und NMT-Kommandos übertragen werden, jedoch keine Prozessdaten.

prepared

prepared (engl.) = vorbereitet (auch: angehalten)

Betriebszustand eines CANopen-Teilnehmers. In diesem Modus werden nur NMT-Kommandos übertragen.

Programmiersprache, sicherheitsrelevant

Für sicherheitsrelevante Applikationen sollten nur folgende Programmiersprachen verwendet werden:

- Programmiersprache mit eingeschränktem Sprachumfang (LVL = **l**imited **v**ariability **l**anguage), kann vordefinierte, applikations-spezifische Bibliotheksfunktionen kombinieren.
  In CoDeSys sind das Kontaktplan KOP (Ladder Diagram LD) und Funktionsplan FUP (Function block diagram FBD).
- Programmiersprache mit nicht eingeschränktem Sprachumfang (FVL = **f**ull **v**ariability **l**anguage), kann einen großen Bereich von Funktionen kombinieren.
  Dazu gehören z.B. C, C++, Assembler. In CoDeSys ist das Strukturierter Text (ST).

► Strukturierter Text ist ausschließlich in gesonderten, zertifizierten Funktionen zu empfehlen, normalerweise in Embedded Software.

► Im "normalen" Applikations-Programm sollten nur KOP (LD) und FUP (FBD)

eingesetzt werden. Damit sollen die folgenden Mindestanforderungen erfüllt werden können.

Generell werden folgende Mindestanforderungen an sicherheitsrelevante Applikations-Software (SRASW) gestellt:

- ► Programm modular und klar strukturieren. Folge: einfache Testbarkeit.
- ► Funktionen verständlich darstellen:
  - für den Operator auf dem Bildschirm (Navigation),
  - Lesbarkeit des späteren Dokumentationsausdrucks.
- ► Symbolische Variablen verwenden (keine IEC-Adressen).
- ► Variablennamen und Kommentare aussagekräftig formulieren.
- ► Einfache Funktionen verwenden (keine indirekte Adressierung, keine Variablenfelder).
- ► Defensiv programmieren.
- ► Leichtes Erweitern oder Anpassen des Programms ermöglichen.

PWM

PWM = Puls-Weiten-Modulation

Via PWM kann ein (vom Gerät dazu befähigter) digitaler Ausgang mittels regelmäßiger, schneller Impulse eine beinahe analoge Spannung ausgeben. Bei dem PWM-Ausgangssignal handelt es sich um ein getaktetes Signal zwischen GND und Versorgungsspannung.

Innerhalb einer festen Periode (PWM-Frequenz) wird das Puls-/Pausenverhältnis variiert. Durch die angeschlossene Last stellt sich je nach Puls-/Pausenverhältnis der entsprechende Effektivstrom ein.
→ Kapitel PWM-Signalverarbeitung, Seite 164
→ Kapitel Was macht ein PWM-Ausgang?, Seite 184

R

Ratio

Ratio (lat.) = Verhältnis

Messungen können auch ratiometrisch erfolgen = Verhältnismessung. Das Eingangssignal erzeugt ein Ausgangssignal, das in einem bestimmten Verhältnis zu ihm liegt. Das bedeutet, ohne zusätzliche Referenzspannung können analoge Eingangssignale ausgewertet werden. Ein Schwanken der Versorgungsspannung hat auf diesen Messwert dann keinen Einfluss.
→ Kapitel "Zählerfunktionen"

redundant

Redundanz ist das Vorhandensein von mehr als den notwendigen Mitteln, damit eine Funktionseinheit eine geforderte Funktion ausführt oder damit Daten eine Information darstellen können.

Man unterscheidet verschiedene Arten der Redundanz:

- Die funktionelle Redundanz zielt darauf ab, sicherheitstechnische Systeme mehrfach parallel auszulegen, damit beim Ausfall einer Komponente die anderen den Dienst gewährleisten.
- Zusätzlich versucht man, die redundanten Systeme voneinander räumlich zu trennen. Dadurch minimiert man das Risiko, dass sie einer gemeinsamen Störung unterliegen.
- Schließlich verwendet man manchmal Bauteile unterschiedlicher Hersteller, um zu vermeiden, dass ein systematischer Fehler sämtliche redundanten Systeme ausfallen lässt (diversitäre Redundanz).

Die Software von redundanten Systemen sollte sich möglichst in den folgenden Aspekten unterscheiden:

- Spezifikation (verschiedene Teams),
- Spezifikationssprache,
- Programmierung (verschiedene Teams),
- Programmiersprache,
- Compiler.

remanent

Remanente Daten sind gegen Datenverlust bei Spannungsausfall geschützt.

Z.B. kopiert das Betriebssystem die remanenten Daten automatisch in einen Flash-Speicher, sobald die Spannungsversorgung unter einen kritischen Wert sinkt. Bei Wiederkehr der Spannungsversorgung lädt das Betriebssystem die remanenten Daten zurück in den Arbeitsspeicher.

Dagegen sind die Daten im Arbeitsspeicher einer Steuerung flüchtig und bei Unterbrechung der Spannungsversorgung normalerweise verloren.

Restrisiko

Das ist das verbleibende Risiko, nachdem Schutzmaßnahmen ergriffen wurden. Vor dem Restrisiko muss in Betriebsanleitungen und an der Maschine deutlich gewarnt werden.

Risiko

Als Risiko gilt die Kombination der Wahrscheinlichkeit des Eintritts eines Schadens und des Ausmaßes des Schadens.

Risikoanalyse

Kombination aus ...
- Festlegung der Grenzen der Maschine,
- Identifizierung der Gefährdung und
- der Risikoeinschätzung.

Risikobeurteilung

Das ist die Gesamtheit des Verfahrens, das die Risikoanalyse und die Risikobewertung umfasst.

Risikobewertung

Das ist die auf der Risikoanalyse beruhende Beurteilung, ob die Ziele zur Risikominderung erreicht wurden.

ro

ro = read only (engl.) = nur lesen

Unidirektionale Datenübertragung: Daten können nur gelesen werden, jedoch nicht verändert.

Rückstellung, manuell

Die manuelle Rückstellung ist eine interne Funktion des SRP/CS zum anuellen Wiederherstellen einer oder mehrerer Sicherheitsfunktionen. Wird vor dem Neustart einer Maschine verwendet.

rw

rw = read/write (engl.) = lesen und schreiben

Bidirektionale Datenübertragung: Daten können sowohl gelesen als auch verändert werden.

S

Schaden

Als Schaden bezeichnet man eine physische Verletzung oder Schädigung der Gesundheit.

Schutzmaßnahme

Maßnahme zur vorgesehenen Minderung des Risikos, z.B.:
- fehlerausschließender Entwurf,
- technische Schutzmaßnahme (trennende Schutzeinrichtung),
- ergänzende Schutzmaßnahme (Benutzerinformation),
- persönliche Schutzausrüstung (Helm, Schutzbrille).

SCT

Bei CANopen-Safety überprüft die Sicherheits-Zykluszeit SCT (**S**afeguard **c**ycle **t**ime) die korrekte Funktion der periodischen Übertragung (Daten-Refresh) der SRDOs. Die Daten müssen innerhalb der eingestellten Zeit wiederholt worden sein, um gültig zu sein. Andernfalls signalisiert die empfangene Steuerung einen Fehler und geht in den sicheren Zustand (= Ausgänge abgeschaltet).

SDO

SDO = **S**ervice **D**ata **O**bject = Nachrichten-Objekt mit Servicedaten.

SDO ist eine Spezifikation für eine herstellerunabhängige Datenstruktur zum einheitlichen Datenzugriff. Dabei fordern "Clients" die gewünschten Daten von "Servern" an. Die SDOs bestehen immer aus 8 Bytes. Längere Datenpakete werden auf mehrere Nachrichten verteilt.

Beispiele:

- Automatische Konfiguration aller Slaves über SDOs beim Systemstart.
- Auslesen der Fehlernachrichten aus dem Objektverzeichnis.

Jedes SDO wird auf Antwort überwacht und wiederholt, wenn sich innerhalb der Überwachungszeit der Slave nicht meldet.

Sicherheits-Normentypen

Sicherheitsnormen auf dem Gebiet der Maschinen sind wie folgt strukturiert:

Typ-A-Normen (Sicherheits-Grundnormen) behandeln Grundbegriffe, Entwurfsleitsätze und allemeine Aspekte, die auf Maschinen angewandt werden können.

Typ-B-Normen (Sicherheits-Fachgrundnormen)
behandeln einen Sicherheitsaspekt oder eine Art von Schutzeinrichtungen, die für eine Reihe von Maschinen verwendet werden können.

- Typ-B1-Normen für bestimmte Sicherheitsaspekte (Abstände, Temperaturen, Lärm, ...)

- Typ-B2-Normen für Schutzeinrichtrungen (Zweihandschaltungen, trennende Schutzeinrichtungen, ...)

Typ-C-Normen (Maschinensicherheitsnormen) behandeln detaillierte Sicherheitsanforderungen an eine bestimmte Maschine oder Maschinengruppen.

Sicherheitsfunktion

Der Ausfall einer Sicherheitsfunktion einer Maschine kann zum unmittelbar erhöhten Risiko führen. Der Konstrukteur einer solchen Maschine muss daher:
- einen Ausfall der Sicherheitsfunktion sicher verhindern,
- einen Ausfall der Sicherheitsfunktion rechtzeitig sicher erkennen,
- Maschine bei einem Ausfall der Sicherheitsfunktion rechtzeitig in einen sicheren Zustand bringen.

SIL

Der Sicherheits-Integritätslevel SIL ist nach IEC 62061 eine Einstufung (SIL 1...4) der Sicherheitsintegrität der Sicherheitsfunktionen. Er dient der Beurteilung elektrischer/elektronischer/programmierbar elektronischer (E/E/PE)-Systeme in Bezug auf die Zuverlässigkeit von Sicherheitsfunktionen. Aus dem angestrebten Level ergeben sich die sicherheitsgerichteten Konstruktionsprinzipien, die eingehalten werden müssen, damit das Risiko einer Fehlfunktion minimiert werden kann.

Slave

Passiver Teilnehmer am Bus, antwortet nur auf Anfrage des →Masters. Slaves haben im Bus eine eindeutige und einmalige →Adresse.

SRDO

Über SRDOs (**S**afety-**r**elevant **d**ata **o**bjects = Sicherheitsrelevante Datenobjekte) werden die sicheren Daten ausgetauscht. Ein SRDO besteht immer aus zwei CAN-Nachrichten mit unterschiedlichen Identifiern:

- Nachricht 1 enthält die Originalanwenderdaten,
- Nachricht 2 enthält die gleichen Daten, die aber bitweise invertiert werden.

SRP/CS

Safety-**r**elevant **p**art of **c**ontrol **s**ystems = Sicherheitsrelevanter Teil einer Steuerung

SRP/CS ist ein Teil einer Steuerung, das auf sicherheitsgerichtete Eingangssignale reagiert und sicherheitsgerichtete Ausgangssignale erzeugt. Die Kombination sicherheitsrelevanter Teile einer Steuerung beginnt an dem Punkt, an dem sicherheitsgerichtete Signale erzeugt werden (einschließlich Betätiger z.B. eines Positionsschalters) und endet an den Ausgängen der Leistungssteuerungselemente (einschließlich z.B. der Hauptkontakte eines Schützes).

SRVT

Die sicherheitsrelevante Objekt-Gültigkeitsdauer SRVT (**S**afety-**r**elevant **o**bject **v**alidation **t**ime) sorgt bei CANopen-Safety dafür, dass die Zeit zwischen den zusammengehörenden SRDO-Nachrichten eingehalten wird:

Nur wenn die redundante, invertierte Nachricht innerhalb der eingestellten Zeit SRVT nach der Original-Nachricht übertragen wurde, sind die damit übertragenen Daten gültig. Andernfalls signalisiert die empfangende Steuerung einen Fehler und geht in den sicheren Zustand (= Ausgänge abgeschaltet).

Symbole

Piktogramme sind bildhafte Symbole, die eine Information durch vereinfachte grafische Darstellung vermitteln.

→ Kapitel Was bedeuten die Symbole und Formatierungen?, Seite 7

Symbole und Formatierungen

Ein Link ist ein Querverweis zu einer anderen Stelle im Dokument oder auf ein externes Dokument.

T

Target

Das Target gibt das Zielsystem an, auf dem das SPS-Programm laufen soll. Im Target sind die Dateien (Treiber und ggf. spezifische Hilfedateien) enthalten, die zum Programmieren und Parametrieren erforderlich sind.

TCP

Das **T**ransmission **C**ontrol **P**rotocol ist Teil der Protokollfamilie TCP/IP.Jede TCP/IP-Datenverbindung hat einen Sender und einen Empfänger. Dieses Prinzip ist eine verbindungsorientierte Datenübertragung. In der TCP/IP-Protokollfamilie übernimmt TCP als verbindungsorientiertes Protokoll die Aufgabe der Datensicherheit, der Datenflusssteuerung und ergreift Maßnahmen bei einem Datenverlust.

(vgl.: →UDP)

Template

Template (englisch = Schablone)

Ist eine Vorlage, die mit Inhalten gefüllt werden kann.
Hier: Eine Struktur von passend vorkonfigurierten Software-Elementen als Basis für ein Applikations-Programm.

Testrate rt

Die Testrate r_t ist die Häufigkeit der automatischen Tests, um Fehler in einem SRP/CS rechtzeitig zu bemerken.

Ü

Überwachung

Die Überwachung ist eine Sicherheitsfunktion, die sicherstellt, dass eine Schutzmaßnahme eingeleitet wird, sobald Folgendes eintritt:

- Die Fähigkeit eines Bauteils oder eines Elements, seine Funktion auszuführen, wird vermindert.
- Die Betriebsbedingungen werden so verändert, dass das resultierende Risiko steigt.

U

UDP

UDP (**U**ser **D**atagram **P**rotocol) ist ein minimales, verbindungsloses Netzprotokoll, das zur Transportschicht der Internetprotokollfamilie gehört. Aufgabe von UDP ist es, Daten, die über das Internet übertragen werden, der richtigen Applikation zukommen zu lassen.

Derzeit sind Netzwerkvariablen auf Basis von CAN und UDP implementiert. Die Variablenwerte werden dabei auf der Basis von Broadcast-Nachrichten automatisch ausgetauscht. In UDP sind diese als Broadcast-Telegramme realisiert, in CAN als PDOs. Diese Dienste sind vom Protokoll her nicht bestätigte Dienste, d.h. es gibt keine Kontrolle, ob die Nachricht auch beim Empfänger ankommt. Netzwerkvariablen-Austausch entspricht einer "1-zu-n-Verbindung" (1 Sender zu n Empfängern).

V

Verwendung, bestimmungsgemäß

Das ist die Verwendung eines Produkts in Übereinstimmung mit den in der Anleitung bereitgestellten Informationen.

W

Watchdog

Der Begriff Watchdog (englisch; Wachhund) wird verallgemeinert für eine Komponente eines Systems verwendet, die die Funktion anderer Komponenten beobachtet. Wird dabei eine mögliche Fehlfunktionen erkannt, so wird dies entweder signalisiert oder geeignete Programm-Verzweigungen eingeleitet. Das Signal oder die Verzweigungen dienen als Auslöser für andere kooperierende Systemkomponenten, die das Problem lösen sollen.

wo

wo = write only (engl.) = nur schreiben

Unidirektionale Datenübertragung: Daten können nur verändert werden, jedoch nicht gelesen.

Z

Zustand, sicher

Der Zustand einer Maschine gilt als sicher, wenn von ihr keine Gefährdung mehr ausgeht. Dies ist meist der Fall, wenn alle gefahrbringenden Bewegungsmöglichkeiten abgeschaltet sind und nicht unerwartet wieder anlaufen können.

Zykluszeit

Das ist die Zeit für einen Zyklus. Das SPS-Programm läuft einmal komplett durch.

Je nach ereignisgesteuerten Verzweigungen im Programm kann dies unterschiedlich lange dauern.

14 Index

Index

Index