SO-1059

Erweiterungspaket CANopen Source Code für CiA 304 Safety Framework

Software Manual

Auflage September 2015 L-1077d_08

SYS TEC electronic GmbH
Am Windrad 2 08468 Heinsdorfergrund Deutschland
Telefon: +49 (0) 3765 38600-0 Fax: +49 (0) 3765 38600-4100
Web: www.systec-electronic.com Mail: info@systec-electronic.com

Systemhaus für verteilte Automatisierung

Status/ Änderungen

Status: Freigegeben

Datum/ Version	Abschnitt	Änderung	Autor/Bearbeiter
04.08.2015 V8	4	Abkürzungsverzeichnis angefügt	D. Krüger
	1.6	„Zulassungsstelle“ in „Zertifizierungsstelle“ geändert	
	1	DIN EN 50325-5:2009 in EN 50325-5:2010 geändert	
	alle	„Safety relevant“ in „Safety related“ gemäß EN 50325-5:2010 geändert	
	2.8.1.1 ➔ 2.8.2	um eine Gliederungsebene hochgestuft	
	2.10.2	Hinweis zu SRDO_MAX_SRDO_IN_OBD eingefügt	N. Hehlke
28.09.2015 V8	2.4	Beschreibung für Define SRDO_GRANULARITY überarbeitet.	D. Krüger
	1.1	Beschreibung für „sicherer Zustand“ überarbeitet	D. Krüger
	2.5.3	Beschreibung für „sicherer Zustand“ überarbeitet und korrigiert	D. Krüger
	-	Abschnitt „Referenzen“ eingefügt	D. Krüger

Im Buch verwendete Bezeichnungen für Erzeugnisse, die zugleich ein eingetragenes Warenzeichen darstellen, wurden nicht besonders gekennzeichnet. Das Fehlen der © Markierung ist demzufolge nicht gleichbedeutend mit der Tatsache, dass die Bezeichnung als freier Warenname gilt. Ebenso wenig kann anhand der verwendeten Bezeichnung auf eventuell vorliegende Patente oder einen Gebrauchsmusterschutz geschlossen werden.

Die Informationen in diesem Handbuch wurden sorgfältig überprüft und können als zutreffend angenommen werden. Dennoch sei ausdrücklich darauf verwiesen, dass die Firma SYS TEC electronic GmbH weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgeschäden übernimmt, die auf den Gebrauch oder den Inhalt dieses Handbuches zurückzuführen sind. Die in diesem Handbuch enthaltenen Angaben können ohne vorherige Ankündigung geändert werden. Die Firma SYS TEC electronic GmbH geht damit keinerlei Verpflichtungen ein.

Ferner sei ausdrücklich darauf verwiesen, dass SYS TEC electronic GmbH weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgeschäden übernimmt, die auf falschen Gebrauch oder falschen Einsatz der Hard- bzw. Software zurückzuführen sind. Ebenso können ohne vorherige Ankündigung Layout oder Design der Hardware geändert werden. SYS TEC electronic GmbH geht damit keinerlei Verpflichtungen ein.

SYS TEC electronic GmbH - Kontakte	
Adresse:	SYS TEC electronic GmbH Am Windrad 2 D-08468 Heinsdorfergrund GERMANY
Angebots-Hotline:	+49 (0) 3765 / 38600-2110 info@systec-electronic.com
Technische Hotline:	+49 (0) 3765 / 38600-2140 support@systec-electronic.com
Fax:	+49 (0) 3765 / 38600-4100
Webseite:	http://www.systec-electronic.com

8. Auflage September 2015

Referenzen

/1/ EN 50325-5:2010: Industrial communications subsystem based on ISO 11898 (CAN) for controller-device interfaces – Part 5: Functional safety communication based on EN 50325-4
/2/ CANopen User Manual, Software Manual, L-1020, SYS TEC electronic GmbH
/3/ CANopen Objektverzeichnis, Software Manual, L-1024, SYS TEC electronic GmbH
/4/ CAN Treiber, Software Manual, L-1023, SYS TEC electronic GmbH

Einleitung

Dieses Handbuch beschreibt als Ergänzung zum „CANopen User Manual L-1020“ die Anwenderschicht des SRDO Moduls.

Das Kapitel 1 vermittelt einige grundlegende Begriffe des Safety Framework.

Kapitel 2 erläutert die konkrete Umsetzung und beschreibt die Anwenderfunktionen, -schnittstellen und Datenstrukturen.

1 Grundlagen “CANopen Safety”

Das CiA Draft Standard Proposal 304 “CANopen Framework for Safety-Relevant Communication” definiert die CANopen Protokollerweiterungen für die Integration von sicherheitsrelevanten Geräten in CANopen Netze. Das Protokoll ermöglicht es, sicherheitsgerichtete Geräte neben nicht-sicherheitsgerichteten Geräten in einem CANopen Netz zu betreiben. Die Sicherheitsfunktionen werden über spezielle Kommunikationsobjekte, den SRDO’s (safety related data objects) realisiert.

Das CANopen Safety Protokoll erlaubt es, sicherheitsgerichtete Sensoren und Aktoren direkt miteinander zu verbinden. Eine sicherheitsgerichtete Steuerung (z.B.: SPS, Sicherheitsmonitor) wird nicht benötigt. Somit lassen sich logisch vergleichbare Sicherheitsketten wie in herkömmlich verdrahteter Technik realisieren (z.B. Not-Aus-Taster wirkt direkt auf das Sicherheitsrelais).

Der Standard CiA-304 wurde in die Norm EN 50325-5:2010 überführt.

1.1 SRDO – Safety-Related Data Object

Die SRDO Kommunikation folgt dem Producer/Consumer Prinzip. D.h. es gibt einen SRDO Producer und einen oder mehrere SRDO Consumer.
Ein SRDO besteht aus zwei CAN-Telegrammen. Für die Bildung eines SRDO gelten folgende Regeln:

1. Die CAN-Identifier der zwei CAN-Telegramme unterscheiden sich in mindestens zwei Bitpositionen. Der CAN-Identifier des CAN-Telegramms mit den normalen Daten ist immer ungerade. Der CAN-Identifier des CAN-Telegramms mit den invertierten Daten ist immer der darauf folgende gerade Wert.
2. Die Daten der zwei CAN-Telegramme sind redundant. Jedoch werden die Daten des zweiten CAN-Telegramms bitweise invertiert.
3. Ein SRDO wird periodisch übertragen, wobei der Abstand zwischen zwei SRDO’s durch die SCT (safeguard cycle time) bestimmt ist.
4. Der Abstand der zwei CAN-Telegramme eines SRDO darf die SRVT (safety related object validation time) nicht überschreiten.
5. Die Reihenfolge der zwei CAN-Telegramme eines SRDO muss eingehalten werden. Zuerst erfolgt die Übertragung der realen Daten und anschließend folgt die Übertragung der invertierten Daten.

Der Empfänger prüft die Gültigkeit eines SRDO. Die zeitliche und logische Abfolge der CAN-Telegramme eines SRDO wird mit einem Erwartungswert verglichen. Anschließend erfolgt eine Verifizierung der Nutzdaten. Erkannte Fehler führen zum Wechsel in den sicheren Zustand der zugeordneten Sicherheitsfunktion (z.B. Aktor). Der sichere Zustand ist in Abhängigkeit der Applikation durch den Gerätehersteller und/oder Anwender zu definieren.
Die Eigenschaften der SRDO’s (CAN-Identifier, SCT, SRVT, Mapping) sind im Objektverzeichnis hinterlegt und über eine CRC (16- Bit cyclic redundant check) auf Gültigkeit geprüft.

1.1.1 Kommunikations-Parameter eines SRDO

Die Kommunikations-Parameter eines SRDO definieren die Übertragungs-eigenschaften und die COB-IDs eines SRDO.

Die Kommunikations-Parameter eines SRDO sind Einträge im Objektverzeichnis (Index 0x1301 – 0x1340) und können daher über den CAN-Bus mit Hilfe von Servicedatenobjekten (SDO) gelesen und, wenn erlaubt, verändert werden.

Index	Sub-index	Objektdaten	Bedeutung
0x1301	0	Anzahl der folgenden Einträge	
	1	Information Direction	Definition, ob das SRDO Ausgeschaltet (0), eine TSRDO (1) oder ein RSRDO (2) ist
	2	Refresh Time / SCT	Abstand zwischen zwei Übertragungen eines SRDO
	3	SRVT	Abstand zwischen den zwei CAN-Nachrichten eines SRDO
	4	Transmission Type	Übertragungsart des SRDO (fix 254)
	5	COB-ID 1	CAN-Identifier normale Daten
	6	COB-ID 2	CAN-Identifier inverse Daten

Tabelle 1: Kommunikations-Parameter für das erste SRDO

Information Direction (Subindex 1)
Die *Information Direction* dient zur Festlegung, ob das SRDO ausgeschaltet ist oder ob es als Sende- oder Empfangs-SRDO verwendet wird. Folgende Werte sind möglich:

Wert	Bedeutung
0x00	das SRDO ist ausgeschaltet
0x01	das SRDO ist eingeschaltet als Sende-SRDO
0x02	das SRDO ist eingeschaltet als Empfangs-SRDO
0x03 – 0xFF	reserviert

Tabelle 2: Information Direction eines SRDO

Refresh Time / SCT (Subindex 2)
Die *Refresh Time / SCT* legt den Abstand zwischen zwei Übertragungen eines SRDO fest, d.h. der Abstand zwischen den jeweils ersten CAN-Nachrichten eines SRDO. Für Sende-SRDO bedeutet der Parameter den Abstand zwischen zwei Sendungen des SRDO. Bei Empfangs-SRDO ist es die maximale Zeit, die zwischen zwei Übertragungen des SRDO liegen darf, damit das SRDO als gültig erkannt wird. Die Angabe erfolgt in Millisekunden.

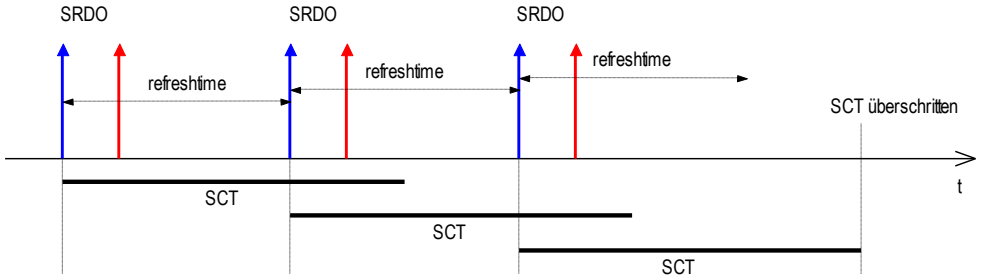


Abbildung 1: Prinzip SCT

SRVT (Subindex 3)
Die *SRVT* legt den maximalen Abstand zwischen den zwei CAN-Nachrichten eines Empfangs-SRDO fest, d.h. die Zeit zwischen der Nachricht mit den normalen Daten und der Nachricht mit den inversen Daten. Sende-SRDO werden direkt hintereinander gesendet.
Die Angabe erfolgt in Millisekunden.

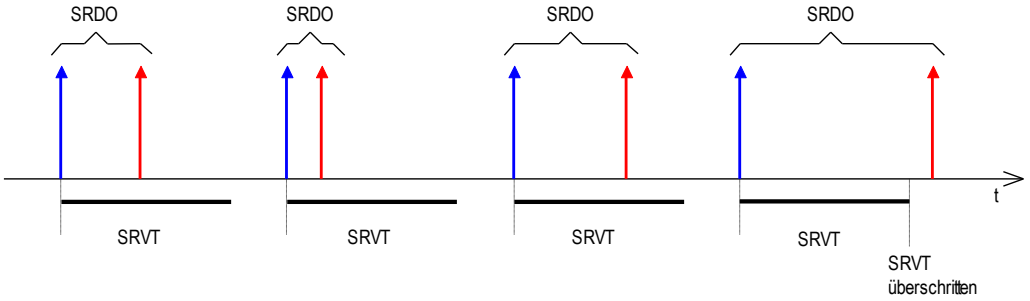


Abbildung 2: Prinzip SRVT

Transmission Type (Subindex 4)
Der *Transmission Type* legt den Charakter einer SRDO-Übertragung fest. Als Wert ist nur 254 zulässig. Dies bedeutet asynchrone Übertragung (siehe CiA DS301).

COB-IDs (CAN-Identifier, Subindex 5 und 6)
Die *COB-IDs 1 und 2* dienen zur Identifizierung und zur Definition der Priorität eines SRDO beim Buszugriff. Für jede CAN-Nachricht darf es nur einen Sender (Producer) geben. Es können jedoch mehrere Empfänger (Consumer) existieren. Es sind Werte im Bereich von 0x101 – 0x180 zulässig. Ein SRDO besteht immer aus zwei aufeinanderfolgenden COB-Ids, wobei COB-ID 1 ungerade und COB-ID 2 die darauffolgende ID ist. Eine Veränderung der Werte ist nur möglich, wenn das SRDO ausgeschaltet ist, d.h. Subindex 1 *Information Direction* ist auf 0 gesetzt.

Bit 31 – 11	***Bit 10 – 0***
reserved (0)	CAN Identifier

Tabelle 3: Aufbau einer COB-ID für ein SRDO

1.1.2 Mapping-Parameter eines SRDO

Der Dateninhalt eines SRDO wird durch die Mapping-Parameter beschrieben. Die Mapping-Parameter sind Einträge im Objektverzeichnis (Index 0x1381 – 0x13C0). Ein Mapping-Eintrag ist wie beim Mapping eines PDO aufgebaut *(siehe dazu L-1020)*. Allerdings wird beim SRDO-Mapping immer ein Eintrag für die normalen Daten gefolgt von dem dazugehörigen Eintrag für die inversen Daten angelegt.

Index	Sub-index	Objektdaten	Bedeutung
0x1381	0	8	Anzahl der gemappten Einträge
	1	0x20000310	UNSIGEND16 auf Index 0x2000, Subindex3 (normale Daten)
	2	0x21000310	UNSIGEND16 auf Index 0x2100, Subindex3 (inverse Daten)
	3	0x20010108	UNSIGEND8 auf Index 0x2001, Subindex1 (normale Daten)
	4	0x21010108	UNSIGEND8 auf Index 0x2101, Subindex1 (inverse Daten)
	5	0x20010208	UNSIGEND8 auf Index 0x2001, Subindex2 (normale Daten)
	6	0x21010208	UNSIGEND8 auf Index 0x2101, Subindex2 (inverse Daten)
	7	0x20020620	REAL32 auf Index 0x2002, Subindex6 (normale Daten)
	8	0x21020620	REAL32 auf Index 0x2102, Subindex6 (inverse Daten)

Tabelle 4: Beispiel Mapping-Tabelle für das erste SRDO

1.1.3 CRC eines SRDO

Um die Parameter eines SRDO auf Ihre Gültigkeit zu prüfen, wird über die sicherheitsrelevanten Daten eines jeden SRDOs eine CRC berechnet. Diese wird im Objektverzeichnis auf Index 0x13FF abgelegt. Dabei entspricht die Nummer des Subindex der Nummer des SRDO. Folgende Parameter gehen in die CRC ein:

Kommunikations-Parameter:
a) 1 Byte Information Direction
b) 2 Byte Refresh Time / SCT
c) 1 Byte SRVT
d) 4 Byte COB-ID 1
e) 4 Byte COB-ID 2

Mapping-Parameter:
f) 1 Byte Subindex 0
g1) 1 Byte Subindex
h1) 4 Byte Mapping Daten
...
g128) 1 Byte Subindex
h128) 4 Byte Mapping Daten

Das verwendete Polynom ist: $G(x) = X^{16} + X^{12} + X^5 + 1$.
Startwert für die CRC ist 0x0000.

1.2 Configuration Valid

Um eine komplette SRDO-Konfiguration als gültig zu setzen, wird im Objektverzeichnis auf Index 0x13FE dafür ein Flag angelegt. Bei jedem Schreibzugriff auf einen sicherheitsrelevanten SRDO-Parameter wird dieses Flag automatisch auf eine ungültige Konfiguration gesetzt. Dieses Flag muss vom Anwender nach Beendigung der Konfiguration auf eine gültige Konfiguration gesetzt werden.

Wert	Bedeutung
0xA5	die Konfiguration ist gültig
andere Werte	die Konfiguration ist ungültig

Tabelle 5: Configuration Valid

Allgemeiner Ablauf einer Konfiguration:

1.) Schreiben aller sicherheitsrelevanten Parameter sowie der Checksummen

2.) Zurücklesen aller sicherheitsrelevanten Parameter sowie der Checksummen und Vergleichen mit den geschriebenen Parametern

3.) Konfiguration als gültig setzen

Dieses Flag muss von der Applikation innerhalb der Sicherheitszykluszeit zyklisch geprüft werden. Solange dieses Flag nicht valid ist, darf der sichere Zustand nicht verlassen werden.

1.3 Global Fail-Safe Command GFC

Um die Reaktionszeit in sicherheitsgerichteten Systemen zu erhöhen, ist in das GFC definiert. Es besteht aus einem hochprioren CAN- Telegramm (CAN-Identifier 1). Das GFC enthält keine Daten und kann somit von allen Teilnehmern gesendet werden. Der auslösende Teilnehmer muss jedoch im Anschluss ein zugehöriges SRDO verschicken.

Die Verwendung des GFC ist optional. Es ist ereignisgesteuert und nicht sicherheitsrelevant, da es keine Zeitüberwachung gibt.

Im Objektverzeichnis existiert für GFC der Eintrag Global Fail-Safe Command Parameter auf Index 0x1300. Folgende Werte sind möglich:

Wert	Bedeutung
0x00	GFC wird nicht unterstützt
0x01	GFC wird unterstützt
andere Werte	reserviert

Tabelle 6: Global Fail-Safe Command GFC

1.4 Predefined Connection Set

Das Predefined Connection Set aus CiA DS-301 wird für SRDO wie folgt erweitert:

Broadcast-Objekte:

Objekt	Funktions-code	COB-ID	Index im Objektverzeichnis
GFC	0000	0x001	0x1300

Tabelle 7: Erweiterung Broadcast Predefined Connection Set

Peer-to-Peer Objekte:

Objekt	Funktions-code	COB-ID normale Daten	COB-ID inverse Daten	Index im Objekt-verzeichnis
SRDO-Nachrichten				
SRDO (Node-ID 1 – 32)	0010	0x101 – 0x13F	0x102 – 0x140	0x1301 – 0x1340 tx
SRDO (Node-ID 33 – 64)	0010	0x141 – 0x17F	0x142 – 0x180	0x1301 – 0x1340 rx

Tabelle 8: Erweiterung Peer-to-Peer Predefined Connection Set

1.5 Übersicht sicherheitsgerichteter Einträge im Objektverzeichnis

Index	Name	Objekt-typ	Daten-typ	Attribute
0x1300	GFC Parameter	var	u8	rw
0x1301	1. SRDO Kommunikations-Parameter	record	SRDO Parameter	rw
...	...	...	...	...
0x1340	64. SRDO Kommunikations-Parameter	record	SRDO Parameter	rw
0x1341	reserved			
...	...			
0x1380	reserved			
0x1381	1. SRDO Mapping Parameter	array	u32	rw
...	...	...	...	...
0x13C0	64. SRDO Mapping Parameter	array	u32	rw
0x13C1	reserved			
...	...			
0x13FD	reserved			
0x13FE	Configuration Valid	var	u8	rw
0x13FF	Safety Configuration Checksum	array	u16	rw

Tabelle 9: SRDO Einträge im Objektverzeichnis

1.6 Zertifizierung

Bei dem Softwarepaket SO-1059 handelt es sich um ein Erweiterungspaket für den CANopen Source Code SO-877. Dieses lässt sich nicht als Einheit zertifizieren. Die Zertifizierung setzt ein in sich abgeschlossenes Gerät mit allen dazugehörigen Softwarekomponenten voraus. Daher ist immer der Hersteller des Gerätes für die Zertifizierung verantwortlich.

Was für die Zertifizierung nötig ist, hängt davon ab, welche Sicherheitsstufe erreicht werden soll. Für SIL[1]3 sind z.B. höhere Anforderungen notwendig als für SIL2.

Für eine SIL3 Zertifizierung muss die Hardware zweikanalig aufgebaut werden (*siehe Abbildung 3 und Abbildung 4*). Für geringere Anforderungen kann die Hardware einkanalig aufgebaut werden (*siehe Abbildung 5*). Dazu empfiehlt sich der Einsatz einer Safety-CPU (z.B. TMS570LS von Texas Instruments).

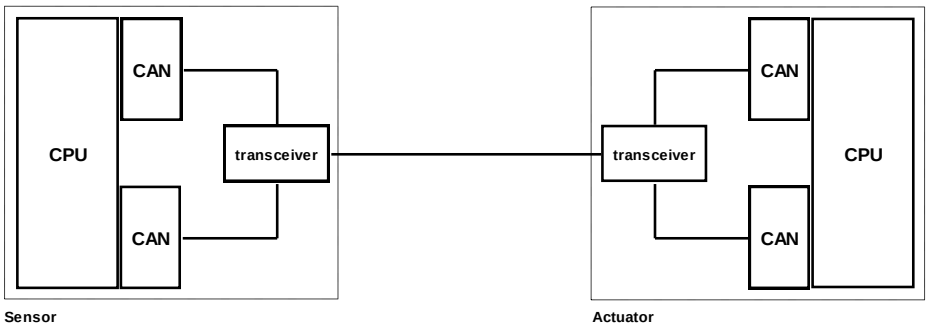


Abbildung 3: zweikanalige Hardware mit einer CPU

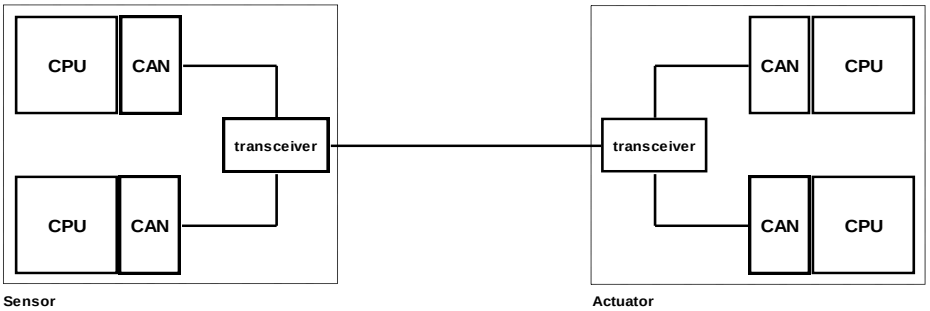


Abbildung 4: zweikanalige Hardware mit zwei CPUs

Abbildung 5: einkanalige Hardware mit einer Safety-CPU

[1] SIL Sicherheits-Integritätslevel (engl. Safety Integrity Level), nach DIN EN 61508

Weiterhin empfiehlt sich die Implementierung weiterer Sicherheitsüberprüfungen in der Software. Diese werden im Folgenden aufgelistet:

- Wiederholte Berechnung einer CRC über den Programmspeicher.
- Wiederholte Prüfung des verwendeten RAM.
- Einsatz eines Watchdog.
- Auswertung von Exceptions, die durch Programmierfehler auftreten können (z.B. Zugriffe auf geschützten Speicher, Zugriffe auf unaligned Adressen usw.)

Das Erweiterungspaket SO-1059 bietet bereits folgende Möglichkeiten zur Sicherheitsprüfung über die Software:

- Berechnung der CRC über die SRDO-Konfiguration.
- Senden der SRDOs über zwei CAN-Nachrichten mit den normalen und invertierten Daten.
- Überwachung der Safety Zykluszeit SCT und Validation-Zeit SRVT sowie der invertierten Daten für Empfangs-SRDOs.

Im Fehlerfall muss die Software immer einen sicheren Zustand für die zu schaltenden Ausgänge einnehmen, so dass keine Lebewesen verletzt oder andere Maschinen zerstört werden.

Es wird empfohlen, vor Beginn der Implementierung, die Struktur der Hardware mit der Zulassungsstelle abzustimmen.

2 Erweiterung der CANopen Anwenderschicht

Dieses Kapitel beschreibt Erweiterungen der in L-1020 beschriebenen Anwenderschicht des SYS TEC CANopen Stack, sowie die Datenstrukturen und API-Funktionen der SYS TEC electronic GmbH spezifischen Umsetzung des CANopen-Standards CiA DS-304, im weiteren SRDO-Modul genannt.

Die Beschreibung umfasst die Syntax der Funktionen, die Parameter, die Rückgabewerte und Erläuterungen zur Anwendung.

Die Bedeutung der Returncodes und die unterstützten Abortcodes
sind im Kapitel 2.13 erläutert.

2.1 Einschränkungen der Hardware

Die Verwendung des SRDO Moduls setzt einen CAN-Controller voraus, bei dem sich die zeitliche Reihenfolge der CAN-Nachrichten auf dem CAN-Bus bestimmen lässt.

Momentan ist das SRDO Modul für den CAN-Controller SJA1000 der Firma NXP (ehemals Phillips) und auf dem internen CAN-Controller des TMS570LS der Firma Texas Instruments portiert und getestet worden. Weitere CAN-Controller werden folgen.

2.2 Einschränkungen der Software

Das SRDO Modul kann nur mit einer bestimmten Konfiguration des CAN-Treibers betrieben werden. Dazu setzen Sie bitte in der Datei copcfg.h folgende Defines auf folgende Werte:

```
CDRV_USE_HIGHBUFF     TRUE
CDRV_USE_BASIC_CAN    TRUE
CDRV_USE_IDVALID      TRUE
```

Die Anzahl der hochpriorisierten Puffereinträge des CAN-Controllers in der Datei obdcfg.h ist mindestens auf die Anzahl der Empfangs- bzw. Send-SRDOs zu setzen. Setzen Sie ihn zur Sicherheit etwas höher.

Der OD-Builder (bis Version V1.19 zum Zeitpunkt dieses Hinweises) kann für die Erstellung des Objektverzeichnisses mit SRDOs nicht verwendet werden. Das liegt daran, dass für die Objekte zwischen Index 0x1300 und 0x13FF spezielle Makros verwendet werden müssen, die diese Version des OD-Builders noch nicht unterstützt. Muss die Anzahl der SRDOs für Ihre Applikation erhöht werden, dann kopieren Sie die entsprechenden Objekte in der Datei obdict.h und passen Sie den Objektindex bzw. -subindex an. Lesen Sie dazu auch die Kapitel 2.10.1 und 2.10.2. Müssen andere Objekte erweitert oder hinzugefügt werden, dann können Sie diese in einem temporären Verzeichnis mit dem OD-Builder erzeugen und durch Copy&Paste in die eigentliche Datei objdict.h übertragen werden.

2.3 Softwarestruktur

Das SRDO Modul gliedert sich parallel zu vorhandenen Modulen wie PDO oder SDO in den Stack ein.

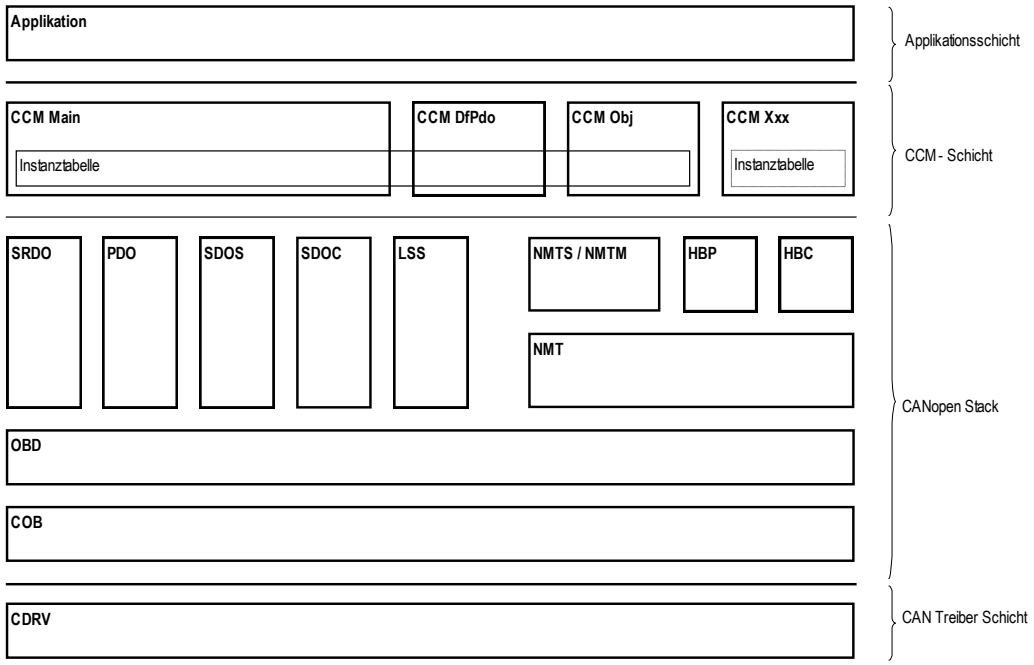


Abbildung 6: Allgemeine Softwarestruktur

Es sind zwei verschiedene SRDO Module implementiert:

SRDO.C	Dieses Modul enthält die Dienste zum Definieren und Übertragen von SRDO.
SRDOSTC.C	Mit diesem Modul steht der gleiche Dienst wie bei SRDO.C zur Verfügung, jedoch handelt es sich hier um die Realisierung des statischen SRDO Mapping.
CCMSRDO.C	Anwenderschnittstelle zum SRDO Modul

2.4 Konfiguration der Software

Die Konfiguration der Software erfolgt wie im Standard-CANopen Stack auch über die Datei copcfg.h. Für SRDOs gibt es wenige Defines, die im Nachfolgenden erläutert werden. Fehlen diese Defines in der Datei copcfg.h, dann wirken ihre Default-Einstellungen.

SRDO_USE_STATIC_MAPPING:
Wertebereich: FALSE, TRUE
Default: FALSE
Bedeutung: Wenn TRUE, dann wird statt dynamisches Mapping der SRDOs statisches Mapping verwendet. Das Mapping lässt sich dann weder per SDO noch von der Applikation aus zur Laufzeit ändern. Statt der Datei SRDO.C muss dann SRDOSTC.C verwendet werden.

SRDO_USE_DUMMY_MAPPING:
Wertebereich: FALSE, TRUE
Default: FALSE
Bedeutung: Bei Verwendung des dynamischen SRDO Mapping können Dummy-Objekte gemappt werden, wenn dieses Define auf TRUE gesetzt ist. Das ermöglicht es für Empfangs-SRDOs, nicht jede Variable im OD implementieren zu müssen, falls diese Variablen für diesen CANopen Knoten nicht wichtig sind.

SRDO_GRANULARITY:
Wertebereich: 8, 16, 32, 64
Default: 8
Bedeutung: Dieses Define bestimmt die kleinste Auflösung in Bits der in ein SRDO gemappten Applikationsobjekte. Steht dieses Define auf 8, ist die kleinste Größe von Applikationsobjekten 8 Bit. Damit können bis zu 8 normale und 8 inverse Applikationsobjekte in ein SRDO gemappt werden. Für den Wert 16 (kleinste Größe von Applikationsobjekten ist 16 Bit) halbiert sich die Anzahl auf 4 normale und 4 inverse Applikationsobjekte usw.

SRDO_ALLOW_GAPS_IN_OD:
Wertebereich: FALSE, TRUE
Default: FALSE
Bedeutung: Dieses Define dient zur Optimierung des Code-Bedarfs im SRDO Modul. Werden die SRDOs im Objektverzeichnis der Reihe nach ohne Lücken implementiert, dann kann dieses Define auf FALSE belassen werden. In diesem Fall werden die SRDOs für die Überprüfungen schneller referenziert. Fehlen aber im Objektverzeichnis einige SRDOs (z.B. es wird nur das SRDO2 mit dem Kommunikationsindex 0x1302 implementiert, aber SRDO1 mit dem Index 0x1301 fehlt – oder SRDO1 und SRDO3 wird implementiert, aber SRDO2 fehlt), dann muss dieses Define auf TRUE gesetzt werden. In diesem Fall werden die SRDOs durch ein Suchalgorithmus referenziert, aus dem eine höhere Laufzeit der Software resultiert. Siehe dazu auch das Kapitel 2.10.2.

SRDO_USE_GFC:
Wertebereich: FALSE, TRUE
Default: TRUE
Bedeutung: Wird die GFC-Nachricht in einem Projekt nicht benötigt, dann können die API-Funktionen CcmSendGfc() und SrdoSendGfc() sowie das Objekt 0x1300 aus Optimierungsgründen entfallen. In diesem Fall muss das Define SRDO_USE_GFC auf FALSE gesetzt werden.

SRDO_USE_PROGMONITOR:
Wertebereich: FALSE, TRUE
Default: TRUE
Bedeutung: Wird in einem Projekt der Programm-Monitor nicht benötigt, dann kann mit Setzen dieses Defines auf FALSE der entsprechende Programmcode aus Optimierungsgründen entfernt werden. Die Callback-Funktion AppProgMonEvent() wird in diesem Fall nicht aufgerufen.

SRDO_CHECK_SRVT_BEFORE_1STRX
Wertebereich: FALSE, TRUE
Default: FALSE
Bedeutung: Soll die SRVT auch wie die SCT zyklisch mit dem Aufruf der Funktion SrdoProcess() überwacht werden, wenn nur eine der beiden CAN-Nachrichten eines SRDOs empfangen wurde, dann muss diese Konstante auf TRUE gesetzt werden. Steht diese Konstante auf FALSE; dann wird frühestens ein Fehler erkannt, wenn die zweite CAN-Nachricht nach Ablauf der SRVT empfangen wurde, oder nachdem die SCT abgelaufen ist. Mit TRUE wird sofort nach Ablauf der SRVT ein Fehler erkannt.

2.5 Funktion des SRDO Moduls

Das SRDO-Modul übernimmt die SRDO-Verarbeitung für dynamisches SRDO Mapping (d.h. das Mapping kann von der Applikation oder per SDO zur Laufzeit verändert werden). Für das statische SRDO Mapping existiert das Modul SRDOSTC.

Zur Beschleunigung der SRDO-Verarbeitung wird für jedes SRDO eine Struktur mit allen relevanten Daten angelegt. Diese Strukturen werden in Tabellen zusammengefasst. Diese SRDO-Tabellen sind Bestandteil des Objektverzeichnisses.

Jedes SRDO verwendet Variablen, die vorher von der Applikation angelegt werden müssen. Beim Mapping werden die Adressen im SRDO auf die entsprechenden Variablen ausgerichtet. Das heißt, für jedes mappbare Objekt muss eine Variable existieren. Dazu muss bei der Definition des Objektverzeichnisses in der Datei **objdict.h** das Makro OBD_SUBINDEX_RAM_USERDEF bzw. OBD_SUBINDEX_RAM_USERDEF_RG für das entsprechende Objekt verwendet werden. Das SRDO Modul überprüft bei jeder Änderung des Mappings die gewählten Parameter. Existiert das Objekt nicht oder besitzt es keine Variable aus der Applikation, dann wird ein Fehler gemeldet.

2.5.1 Senden von SRDO

Die Sendung von SRDO erfolgt direkt aus der Applikation heraus. Dazu wird die Funktion **CcmSrdoSend()** verwendet. Die Überwachung der Refresh Time erfolgt in der Applikation, da nur die Applikation sicherstellen kann, dass die normalen und invertierten Daten untereinander konsistent sind, bevor die CAN-Nachrichten eines SRDO gesendet werden.

Wichtig ist, dass die erste Sendung nach dem Wechsel in den Knotenzustand OPERATIONAL um 0,5ms * Node-ID verzögert werden muss. Den Wechsel des Knotenstatus bekommt die Applikation in der Funktion **AppCbNmtEvent()** mitgeteilt.

2.5.2 Empfang von SRDO

Den Empfang von SRDO übernimmt die Funktion **SrdoProcess()**. Diese Funktion muss zyklisch gerufen werden. Bei Verwendung der Funktion **CcmProcess()** ist dies realisiert.

2.5.3 Sende- und Empfangssignalisierung von SRDO

Die Sendung bzw. der Empfang wird über zwei verschiedene Wege der Applikation signalisiert.

Einerseits über die Callback-Funktionen der Applikation **AppSrdoEvent()** und **AppSrdoError()** und andererseits über den von der Applikation zu pollenden Status eines SRDO. Dieser wird mit **CcmSrdoGetState()** gelesen und mit **CcmSrdoSetState()** geschrieben.

Der Status eines SRDO ist bitcodiert in folgender Form:

```
TX-SRDO:
xx00 xxxxb   Sendung war ok
xx01 xxxxb   Sendung war fehlerhaft
xx11 xxxxb   SRDO wurde bearbeitet

RX-SRDO:
xx00 xxxxb   Empfang war ok
xx01 xxxxb   Empfang war fehlerhaft
xx11 xxxxb   SRDO wurde bearbeitet

SRDO-ERROR:
00xx xxxxb   Resetwert
01xx xxxxb   Wert vor Aufruf von AppSrdoError()
10xx xxxxb   AppSrdoError() muss diesen Wert setzen
```

Die Applikation muss beide Wege bearbeiten.

Beispiel Empfang eines SRDO:
Das SRDO Modul setzt den Status entsprechend auf “Empfang war OK”. Danach ruft das SRDO Modul die Funktion **AppSrdoEvent()** auf. Diese Funktion prüft den Status, ob dieser auf “Empfang war OK” steht. Ist dies nicht der Fall, so ist dies sicherheitskritisch. Die Applikation muss dann entsprechend reagieren. Bei fehlerfreiem Status wird dann der Status auf “SRDO wurde bearbeitet” gesetzt.

In der Applikation muss in der Hauptschleife ebenfalls der Status geprüft werden. Dieser darf nie ungleich “SRDO wurde bearbeitet” sein, da sonst das SRDO nicht in der Funktion **AppSrdoEvent()** bearbeitet wurde. Dies ist dann wiederum sicherheitskritisch.

Mit der Implementierung des SRDO Moduls verfolgen wir die Philosophie, dass der Wechsel vom sicheren Zustand in den Betriebszustand erst mit dem erfolgreichen Empfang des SRDOs erfolgt. Tritt während der Laufzeit ein Fehler auf, ist es Aufgabe der Applikation den Wechsel der zugehörigen Sicherheitsfunktion in den sicheren Zustand zu veranlassen.

2.5.4 logische Programmlaufüberwachung

Im SRDO Modul ist eine logische Programmlaufüberwachung integriert. Es wird bei verschiedenen Programmschritten die Funktion **AppProgMonEvent()** mit dem entsprechenden Event aufgerufen. Die eigentliche Realisierung des Programmlaufmonitors wird in der gerufenen Applikationsfunktion realisiert.

2.6 Funktion des SRDOSTC Moduls

Das Modul SRDOSTC ersetzt das SRDO Modul für statisches SRDO Mapping. Mit dem statischen SRDO Mapping werden die SRDOs bereits im OD fertig gemappt. Das Mapping selbst kann nicht von der Applikation bzw. per SDO geändert werden. Dadurch wird weniger CODE Speicher benötigt.

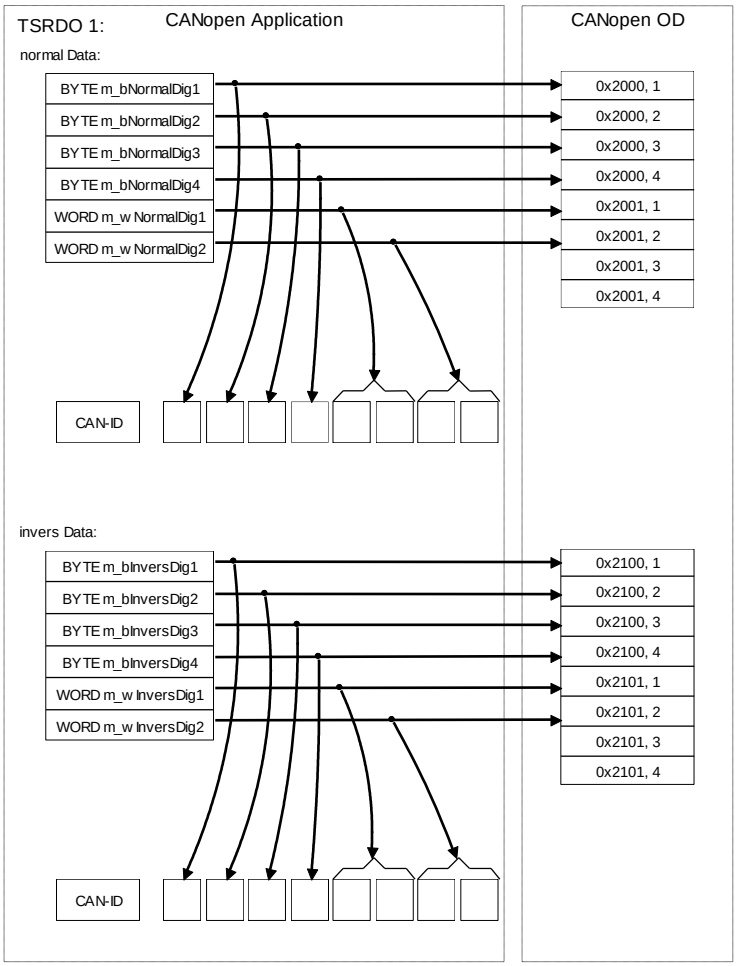


Abbildung 7: Abbildung der Variablenfelder

Der Bezug der SRDO Variablen in der Applikation zu den Daten im OD bzw. zu den Daten in der CAN Nachricht eines SRDOs wird mit der Funktion **CcmStaticDefineSrdoVarField()** hergestellt. Die Applikation muss für jedes SRDO 2 Mal maximal 8 zusammenhängende Datenbytes (das heißt ohne Füllbytes → Struct Alignment 1) zur Verfügung stellen. Diese Datenpakete werden in diesem Handbuch als Variablenfelder eines SRDO bezeichnet. Die Abbildung der Variablenfelder in das OD erfolgt in der Applikation weiterhin durch den Aufruf der Funktion **CcmDefineVarTab()** oder durch das Makro OBD_SUBINDEX_RAM_EXTVAR (*siehe L-1024*) im OD.

Um das statische SRDO Mapping nutzen zu können, muss statt der Datei SRDO.C die Datei SRDOSTC.C eingebunden werden. Weiterhin muss das Define SRDO_USE_STATIC_MAPPING innerhalb der Datei CopCfg.h auf TRUE gesetzt werden.

Einschränkung:
Bei CPUs, die keine ungeraden Zugriffe auf Datentypen größer BYTE unterstützen ist kein gemischtes Mapping von beispielsweise BYTE und WORD in der folgenden Form möglich:
BYTE – WORD – BYTE

Jedoch ist das folgende Mapping möglich:
BYTE – BYTE – WORD

2.7 Prinzipieller Programmablauf

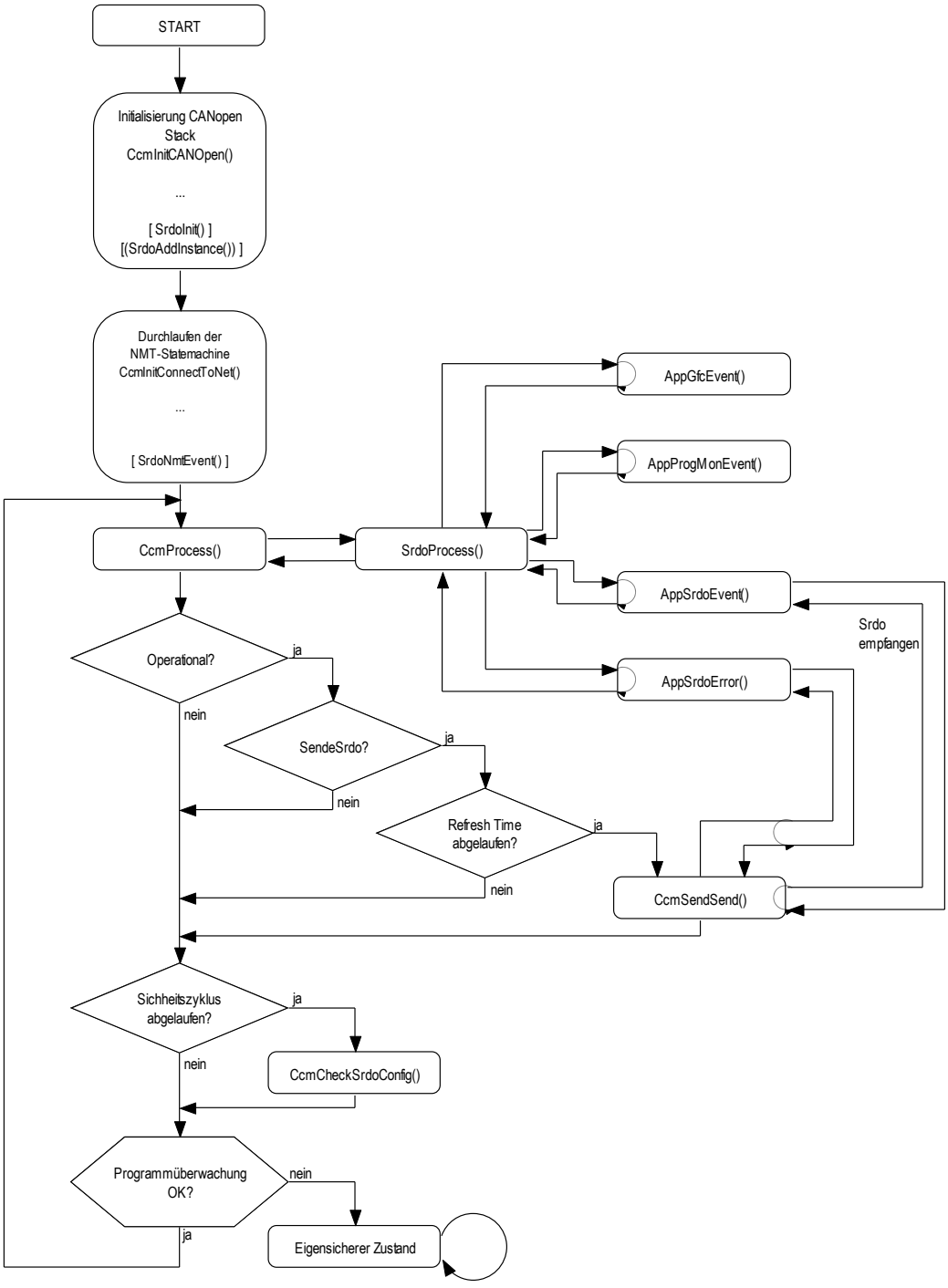


Abbildung 8: Prinzip Programmablauf

2.8 Erweiterung der CCM Schicht

Die Datei CCMMAIN.C ist für die Einbindung des SRDO Moduls erweitert.
Das SRDO Modul muss in der Datei COPCFG.H über das Define `CCM_MODULE_INTEGRATION` aktiviert werden. Dazu ist die Konstante `CCM_MODULE_SRDO` hinzuzufügen.

Ist das SRDO Modul aktiviert, dann führt die Funktion **CcmInitCANOpen()** die Initialisierung des SRDO-Moduls mit durch. Ebenso wird in der Funktion **CcmProcess()** die entsprechende SRDO Funktion gerufen.

Im Weiteren werden die Anwenderfunktionen des SDRO Moduls beschrieben.

2.8.1 Funktion CcmSendSrdo

Syntax:
#include <cop.h>
tCopKernel PUBLIC CcmSendSrdo (CCM_DECL_INSTANCE_HDL_
WORD wSrdoCommuIndex_p);

Parameter:
CCM_DECL_INSTANCE_HDL_: Instanz-Handle

wSrdoCommuIndex_p: Objektindex der Kommunikations-Parameter des SRDO im Objektverzeichnis

Return:
kCopSuccessful Die Funktion wurde ohne Fehler ausgeführt.

Weitere Returncodes siehe Kapitel 2.11.5 - unktion SrdoSend().

Beschreibung:
Die Funktion sendet ein über den Kommunikationsindex angegebenes SRDO oder alle SRDO bei Angabe von 0x0000 als Kommunikationsindex. Vor dem Senden der CAN-Nachrichten eines SRDOs werden alle Bits der Daten auf korrekte Invertierung geprüft. Ist mindestens ein Bit nicht korrekt invertiert, werden die CAN-Nachrichten eines SRDOs nicht gesendet und die Callback Funktion AppSrdoError() aufgerufen.

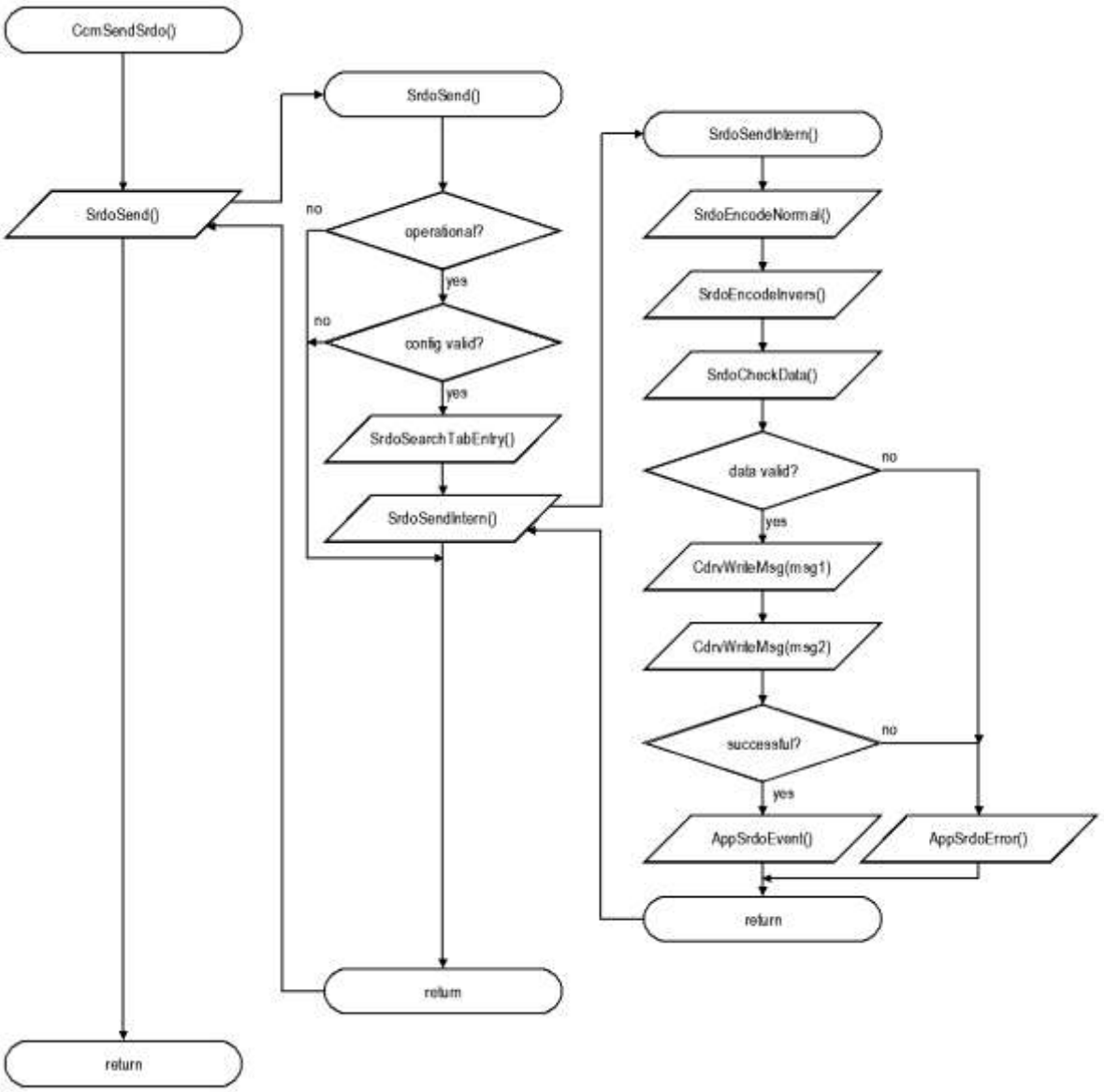


Abbildung 9: Prinzip für das Senden von SRDOs

2.8.2 Funktion CcmCheckSrdoConfig

Syntax:
#include <cop.h>
tCopKernel PUBLIC CcmCheckSrdoConfig (
CCM_DECL_INSTANCE_HDL_
WORD * pwCommuIndex_p);

Parameter:
CCM_DECL_INSTANCE_HDL_: Instanz-Handle

pwCommuIndex_p: Pointer auf eine Variable, in die die Funktion bei fehlerhafter Konfiguration den Kommunikations-index des fehlerhaften SRDO hinterlegt

Return:
kCopSuccessful Die Funktion wurde ohne Fehler ausgeführt.

Weitere Returncodes siehe Kapitel 2.11.7 - Funktion SrdoCheckConfig().

Beschreibung:
Die Funktion berechnet die Checksumme (CRC) über alle SRDO (auch die deaktivierten SRDOs mit Direction = 0) und vergleicht diese mit der im OD konfigurierten. Stellt sie einen Fehler fest, so gibt sie einen Fehler und den entsprechenden Kommunikationsindex des fehlerhaften SRDO zurück. Diese Funktion stellt die API-Funktion für die Funktion SrdoCheckConfig() dar und ruft diese auf. Es ist erforderlich, diese Funktion im Rahmen der Diagnose zyklisch im Diagnosetestintervall aufzurufen. Wird ein Fehler erkannt und der Eintrag *Configuration Valid* (Index 0x13FE) ist valid (0xA5) dann muss in den sicheren Zustand gewechselt werden.

Hinweis: Die Funktion SrdoCheckConfig() wird vom SRDO Modul beim Schreiben des Eintrags *Configuration Valid* (Index 0x13FE im Objektverzeichnis) mit dem Wert 0xA5 gerufen.

2.8.3 Funktion CcmSendGfc

Syntax:
#include <cop.h>
tCopKernel PUBLIC CcmSendGfc (CCM_DECL_INSTANCE_HDL)

Parameter:
CCM_DECL_INSTANCE_HDL: Instanz-Handle

Return:
kCopSuccessful Die Funktion wurde ohne Fehler ausgeführt.

Weitere Returncodes siehe Kapitel 2.11.8 - Funktion SrdoSendGfc().

Beschreibung:
Die Funktion sendet eine GFC-Nachricht.

Sie ist nicht vorhanden, wenn die Konfiguration SRDO_USE_GFC auf FALSE gesetzt ist.
Diese Funktion stellt die API-Funktion für Funktion SrdoSendGfc() dar und ruft diese auf.
Das folgende SRDO ist von der Applikation über die Funktion **CcmSendSrdo()** zu übertragen.

2.8.4 Funktion CcmGetSrdoState

Syntax:
#include <cop.h>
tCopKernel PUBLIC CcmGetSrdoState (CCM_DECL_INSTANCE_HDL_
BYTE * pSrdoState_p,
WORD wSrdoCommuIndex_p);

Parameter:
CCM_DECL_INSTANCE_HDL_: Instanz-Handle

pSrdoState_p: Pointer auf den die Funktion den Status kopiert

wSrdoCommuIndex_p: Objektindex der Kommunikations-Parameter des SRDO im Objektverzeichnis enthält

Return:
kCopSuccessful Die Funktion wurde ohne Fehler ausgeführt.

Weitere Returncodes siehe Kapitel 2.11.9 - Funktion SrdoGetState().

Beschreibung:
Die Funktion liest den Status eines SRDO. Aufbau und Verwendung des Status siehe Kapitel 2.5.3.

2.8.5 Funktion CcmSetSrdoState

Syntax:
#include <cop.h>
tCopKernel PUBLIC CcmSetSrdoState (CCM_DECL_INSTANCE_HDL_
BYTE SrdoState_p,
WORD wSrdoCommuIndex_p);

Parameter:
CCM_DECL_INSTANCE_HDL_: Instanz-Handle

SrdoState_p: zu setzender Status

wSrdoCommuIndex_p: Objektindex der Kommunikations-Parameter des SRDO im Objektverzeichnis enthält

Return:
kCopSuccessful Die Funktion wurde ohne Fehler ausgeführt.

Weitere Returncodes siehe Kapitel 2.11.10 - Funktion SrdoSetState().

Beschreibung:
Die Funktion schreibt den Status eines SRDO. Aufbau und Verwendung des Status siehe Kapitel 2.5.3.

2.8.6 Funktion CcmGetSrdoParam

Syntax:
#include <cop.h>
tCopKernel PUBLIC CcmGetSrdoParam (CCM_DECL_INSTANCE_HDL_
WORD wSrdoCommuIndex_p,
tSrdoCommuParam * pSrdoCommuParam_p,
tSrdoMappParam * pSrdoMappParam_p);

Parameter:
CCM_DECL_INSTANCE_HDL_: Instanz-Handle

wSrdoCommuIndex_p: Objektindex der Kommunikations-Parameter des SRDO im Objektverzeichnis enthält

pSrdoCommuParam_p: Pointer auf die Struktur, in die die Funktion die Werte für *Information Direction* und *SCT* kopiert

pSrdoMappParam_p: Pointer auf die Struktur, in die die Funktion die Werte für *Number of Mapped Objects* und die Zeiger auf die gemappten Variablen kopiert

Return:
kCopSuccessful Die Funktion wurde ohne Fehler ausgeführt.

Weitere Returncodes siehe Kapitel 2.11.11 - Funktion SrdoGetCommuParam() und Kapitel 2.11.12 - Funktion SrdoGetMappParam().

Beschreibung:
Die Funktion liest die in der Applikation benötigten Parameter eines SRDO. Dies sind die Kommunikationsparameter *Information Direction* und *SCT*, sowie die Mappingparameter *Number of Mapped Objects* und die Pointer auf die gemappten Variablen. Die Funktion füllt die Strukturen nur aus, wenn der übergebene Pointer nicht der Nullpointer ist. Die Struktur tSrdoMappParam existiert nur bei dynamischam Mapping.
Diese Funktion stellt die API-Funktion für die Funktion SrdoGetCommuParam() und die Funktion SrdoGetMappParam() dar und ruft diese auf.

Die Struktur tSrdoCommuParam hat den folgenden Aufbau:

```
typedef struct
{
    BYTE m_bDirection;  // direction of SRDO
                        // (0-invalid; 1-Tx; 2-Rx)
    WORD m_wSct;        // refresh time / SCT
}
tSrdoCommuParam;
```

Die Struktur tSrdoMappParam hat den folgenden Aufbau:

```
typedef struct
{
    // Number of mapped objects
    BYTE       m_bNoOfMappedObjects;

    // array of pointers to the mapped variables
    void MEM* m_apMappedVariable[SRDO_MAX_MAPPENTRIES];
}
tSrdoMappParam;
```

2.8.7 Funktion CcmStaticDefineSrdoVarFields

Die Funktion ist nur bei statischem SRDO Mapping verfügbar.

Syntax:
#include <cop.h>
tCopKernel PUBLIC CcmStaticDefineSrdoVarFields(
CCM_DECL_INSTANCE_HDL_
WORD wCommuIndex_p,
void MEM* pNormalData_p,
void MEM* pInversData_p);

Parameter:
CCM_DECL_INSTANCE_HDL_: Instanz-Handle

wCommuIndex_p:	Kommunikations-Index des SRDO im OD, dessen Variablen definiert werden sollen.
pNormalData_p:	Pointer auf ein zusammenhängendes Variablenfeld, das mit den normalen Daten des SRDO verknüpft (bzw. gemappt) werden soll.
pInversData_p:	Pointer auf ein zusammenhängendes Variablenfeld, das mit den inversen Daten des SRDO verknüpft (bzw. gemappt) werden soll.

Return:

kCopSuccessful	Die Funktion wurde ohne Fehler ausgeführt.

Weitere Returncodes siehe Kapitel 2.12.1 - Funktion SrdoStaticDefineVarField().

Beschreibung:
Diese Funktion definiert für ein SRDO die Variablenfelder. Die Applikation ändert die Variablen nur über diese Variablenfelder. Beim Senden eines SRDO werden genau diese Datenbytes aus dem Variablenfeld in die beiden CAN Nachrichten kopiert. Beim Empfang eines SRDO werden die Datenbytes der CAN Nachrichten direkt in die Variablenfelder kopiert.

Die Funktion prüft, ob die angegebenen Variablenfelder auch mit den Variablen übereinstimmen, auf die das Mapping im OD zeigt.

Diese Funktion stellt die API-Funktion für die Funktion SrdoStaticDefineVarFields() dar und ruft diese auf.

2.8.8 Funktion CcmCalcSrdoCrc

Syntax:
#include <srdo.h>
tCopKernel PUBLIC CcmCalcSrdoCrc (MCO_DECL_INSTANCE_PTR_
WORD wCommuIndex_p,
WORD* pwCrc_p);

Parameter:
MCO_DECL_INSTANCE_PTR_: Pointer auf die Instanz

wCommuIndex_p:	Objektindex der Kommunikations-Parameter des SRDO im Objektverzeichnis enthält
pwCrc_p:	Pointer auf eine WORD-Variable für die Rückgabe der 16 Bit CRC innerhalb der aufrufenden Funktion.

Return:

kCopSuccessful	Die Funktion wurde ohne Fehler ausgeführt.
kCopSrdoNotExist	Das ausgewählte SRDO existiert nicht.

Beschreibung:
Die Funktion berechnet die Checksumme (CRC) über ein SRDO und gibt diese der aufrufenden Funktion zurück. Die Berechnung erfolgt auch, wenn das SRDO ausgeschaltet ist. Es erfolgt kein Vergleich auf die Richtigkeit der CRC. Die Applikation kann diese Funktion verwenden, um die CRC eines SRDOs zu aktualisieren, wenn die Konfiguration eines SRDOs über die Applikation dynamisch neu gesetzt werden muss (z.B. Ändern der COB-ID in Abhängigkeit der Node-ID). Diese Funktion stellt die API-Funktion für die Funktion SrdoCalcSrdoCrc() dar und ruft diese auf.

Hinweis: Die Gültigkeitsprüfung der CRC, d.h. die Berechnung der CRC über die Konfigurationsdaten eines SRDO und Vergleich mit der zugehörigen CRC im Index 0x13FF, erfolgt in der Funktion **CcmCheckSrdoConfig**().

Beispiel:

```
WORD  wTestCrc;
Ret = CcmCalcSrdoCrc (0x1301, &wTestCrc);
if (Ret != kCopSuccessful)
{
    goto Exit;
}
PRINTF1 ("Calculated CRC of SRDO1 = 0x%04X\n", wTestCrc);
```

2.9 Funktionen in der Applikation

Die Funktionen der Applikation, die vom SRDO Modul als Callback-Funktion gerufen werden, werden direkt aufgerufen, nicht wie im restlichen CANopen über Funktionspointer. Diese Funktionen müssen also in der Applikation vorhanden sein und dürfen nicht umbenannt werden.

2.9.1 Funktion AppSrdoEvent

Syntax:
#include <cop.h>
tCopKernel PUBLIC AppSrdoEvent (CCM_DECL_INSTANCE_HDL_
WORD wSrdoCommuIndex_p)

Parameter:
CCM_DECL_INSTANCE_HDL_: Instanz-Handle

wCommuIndex_p: Kommunikations-Index des betreffenden SRDO im OD

Return:
kCopSuccessful Die Funktion wurde ohne Fehler ausgeführt.

Alle anderen Returncodes sind reserviert.

Beschreibung:
Die Funktion wird vom SRDO Modul bei einer fehlerfreien Übertragung (Empfang oder Sendung) eines SRDO gerufen. Der Status des SRDO ist in der Funktion entsprechend Kapitel 2.5.3 zu behandeln.

```
tCopKernel PUBLIC AppSrdoEvent (CCM_DECL_INSTANCE_HDL_
    WORD wSrdoCommuIndex_p)
{
BYTE bSrdoState;
tCopKernel Ret;

    Ret = CcmGetSrdoState (&bSrdoState,
                           wSrdoCommuIndex_p);
    if (Ret != kCopSuccessful)
    {
        ...
    }

    if ((bSrdoState & 0x30) != 0x00)
    {
        // Sicherheitskritischer Fehler !!!
        ...
    }

    // je nach Anwendung Information verarbeiten
    // beispielsweise Ausgänge des SRDO einschalten

    Ret = CcmSetSrdoState ((bSrdoState | 0x30),
                           wSrdoCommuIndex_p);
    if (Ret != kCopSuccessful)
    {
        ...
    }

    return kCopSuccessful;
}
```

2.9.2 Funktion AppSrdoError

Syntax:
#include <cop.h>
tCopKernel PUBLIC AppSrdoError (CCM_DECL_INSTANCE_HDL_
WORD wSrdoCommuIndex_p,
tCopKernel ErrorCode_p)

Parameter:
CCM_DECL_INSTANCE_HDL_: Instanz-Handle

wCommuIndex_p:	Kommunikations-Index des betreffenden SRDO im OD
ErrorCode_p:	Fehlercode des entsprechenden SRDO:
kCopSrdoSctTimeout	Die SCT eines Empfangs-SRDO wurde überschritten.
kCopSrdoSrvtTimeout	Die SRVT eines Empfangs-SRDO wurde überschritten.
kCopSrdoNotInOrder	Die zwei CAN Nachrichten eines SRDO wurden in der falschen Reihenfolge empfangen.
kCopSrdoDataError	Die Daten der CAN Nachrichten eines SRDO sind nicht invers.

Weitere Fehlercodes aus dem CDRV Modul sind möglich.

Return:

kCopSuccessful	Die Funktion wurde ohne Fehler ausgeführt.

Alle anderen Returncodes sind reserviert.

Beschreibung:
Die Funktion wird vom SRDO Modul bei einer fehlerhaften Übertragung (Empfang oder Sendung) eines SRDO gerufen. Der Status des SRDO ist in der Funktion entsprechend Kapitel 2.5.3 zu behandeln.

```
tCopKernel PUBLIC AppSrdoError (CCM_DECL_INSTANCE_HDL_
    WORD wSrdoCommuIndex_p,
    tCopKernel ErrorCode_p)
{
BYTE bSrdoState;
tCopKernel Ret;

    Ret = CcmGetSrdoState (&bSrdoState,
                           wSrdoCommuIndex_p);
    if (Ret != kCopSuccessful)
    {
        ...
    }

    if ((bSrdoState & 0x30) == 0x10)
    {

        // je nach Anwendung Information verarbeiten
        // beispielsweise Ausgänge des SRDO abschalten

        // Status auf "SRDO bearbeitet" setzen
        bSrdoState |=  0x30;

        // toggle Bit 6 and 7
        bSrdoState ^= 0xC0;
    }
    else
    {
        // Sicherheitskritischer Fehler !!!
        ...
    }

    Ret = CcmSetSrdoState ((bSrdoState),
                            wSrdoCommuIndex_p);
    if (Ret != kCopSuccessful)
    {
        ...
    }

    return kCopSuccessful;
}
```

2.9.3 Funktion AppGfcEvent

Syntax:
#include <cop.h>
tCopKernel PUBLIC AppGfcEvent (CCM_DECL_INSTANCE_HDL)

Parameter:
CCM_DECL_INSTANCE_HDL: Instanz-Handle

Return:
kCopSuccessful Die Funktion wurde ohne Fehler ausgeführt.

Alle anderen Returncodes sind reserviert.

Beschreibung:
Die Funktion wird vom SRDO Modul beim Empfang einer GFC-Nachricht gerufen.

```
tCopKernel PUBLIC AppGfcEvent (CCM_DECL_INSTANCE_HDL)
{

    // je nach Anwendung Information verarbeiten
    // beispielsweise in eigensicheren Zustand wechseln

    return kCopSuccessful;
}
```

Die Funktion wird nicht aufgerufen, wenn die Konfiguration SRDO_USE_GFC auf FALSE gesetzt ist.

2.9.4 Funktion AppProgMonEvent

Syntax:
#include <cop.h>
tCopKernel PUBLIC AppProgMonEvent (CCM_DECL_INSTANCE_HDL_
tProgMonEvent Event_p)

Parameter:
CCM_DECL_INSTANCE_HDL: Instanz-Handle

Event_p: Event des abgearbeiteten Programmcodes:

```
kSrdoPMEvSctChecked
    SCT eines SRDO wurde geprüft

kSrdoPMEvSctNotCheckedItIsTx
      SCT eines SRDO wurde nicht geprüft, da es ein Sende-SRDO ist

kSrdoPMEvSctNotCheckedItIsInvalid
      SCT eines SRDO wurde nicht geprüft, da es ausgeschaltet ist

kSrdoPMEvSctNotCheckedNotOperational
      SCT eines SRDO wurde nicht geprüft, da sich der Knoten nicht in
      OPERATIONAL befindet

kSrdoPMEvSrdoError
      Fehlerhaftes SRDO festgestellt (Sende- und Empfangs-SRDO)

kSrdoPMEvSrdoReceived
      ein SRDO wurde empfangen

kSrdoPMEvSrdoTransmitted
      ein SRDO wurde gesendet
```

Return:
kCopSuccessful Die Funktion wurde ohne Fehler ausgeführt.

Alle anderen Returncodes sind reserviert.

Beschreibung:
Die Funktion wird vom SRDO Modul bei der Abarbeitung bestimmter Programmschritte gerufen. Die Applikation kann daraus eine logische Programmlaufüberwachung aufbauen.

Diese Funktion wird nicht aufgerufen, wenn im Konfigurationsfile copcfg.h das Define SRDO_USE_PROGMONITOR auf FALSE gesetzt ist.

```
tCopKernel PUBLIC AppProgMonEvent (CCM_DECL_INSTANCE_HDL_
    tProgMonEvent Event_p)
{
    switch (Event_p)
    {
        case kSrdoPMEvSctChecked:
            // wird für jedes Rx SRDO gerufen
            wPMonValue_g += kPMonSctChecked;
            break;

        case kSrdoPMEvSctNotCheckedItIsTx:
            // wird für jedes Tx SRDO gerufen
            wPMonValue_g += kPMonSctNotCheckedItIsTx;
            break;

        case kSrdoPMEvSctNotCheckedItIsInvalid:
            // wird für jedes ausgeschaltete SRDO gerufen
            wPMonValue_g += kPMonSctNotCheckedItIsInvalid;
            break;

        case kSrdoPMEvSctNotCheckedNotOperational:
            // wird einmal für alle SRDO gerufen
            wPMonValue_g += kPMonSctNotCheckedNotOperational;
            break;

        case kSrdoPMEvSrdoError:
            // wird für fehlerhafte SRDO gerufen
            wPMonValue_g += kPMonSrdoError;
            break;

        case kSrdoPMEvSrdoReceived:
            // wird für jedes empfangene SRDO gerufen
            wPMonValue_g += kPMonSrdoReceived;
            break;

        case kSrdoPMEvSrdoTransmitted:
            // wird für jedes gesendete SRDO gerufen
            wPMonValue_g += kPMonSrdoTransmitted;
            break;

        default:
            break;
    }

    return kCopSuccessful;
}
```

2.9.5 Funktion AppCbNmtEvent

Diese Funktion, die bei Durchlaufen der NMT-Statemachine vom CANopen Stack gerufen wird muss bei verschiedenen Events Aufrufe des SRDO Moduls enthalten:

kNmtEvResetCommunication: Variablenfelder durch Aufruf von **CcmStaticDefineSrdoVarFields()** bekannt geben (bei statischem Mapping)
Initialisierung der SRDO Kommunikations-Parameter durch Aufruf von **CcmWriteObject()** mit den entsprechenden Parametern

```
// define all SRDOs in static SRDO modul
Ret = CcmStaticDefineSrdoVarFields (0x1301,
    &SrdoNormalData.m_abSrdoData[0],
    &SrdoInversData.m_abSrdoData[0]);
if (Ret != kCopSuccessful)
{
     ...
}

// write information direction into OD
Ret = CcmWriteObject (0x1301, 1, &bDirection, 1);
if (Ret != kCopSuccessful)
{
    ...
}

// set configuration valid
bTemp = 0xA5;
Ret = CcmWriteObject (0x13FE, 0, &bTemp, 1);
if (Ret != kCopSuccessful)
{
    ...
}
```

kNmtEvEnterPreOperational: SRDO dürfen nicht mehr bearbeitet werden (NMT-Status speichern, um dies in der Hauptschleife auszuwerten)

```
bSrdoState = kNotOperational;
```

kNmtEvEnterOperational: lesen der aktuellen SRDO-Parameter durch Aufruf von **CcmGetSrdoParameter()**
SRDO müssen verarbeitet werden (NMT-Status speichern, um dies in der Hauptschleife auszuwerten)

```
CcmGetSrdoParam (0x1301, &SrdoCommuParam);

bSrdoState = kEnterOperational;
```

2.10 Objektverzeichnis

Die verschiedenen sicherheitsrelevanten Einträge des Objektverzeichnisses sind im Kapitel 1 beschrieben.

2.10.1 Makros für die Safety Objekte

Für die Realisierung in der CANopen Software gibt es spezielle Makros für die verschiedenen SRDO Einträge. Diese sind in diesem Kapitel beschrieben.

Hinweis:
Der OD-Builder (zur Zeit dieses Hinweises Version V1.19) kann die speziellen Makros für die SRDOs nicht generieren. Deshalb sollten Sie dieses Tool für die Erstellung des Objektverzeichnisses nicht nutzen. Lesen Sie dazu bitte auch das Kapitel 2.2.

Die weiteren Informationen zu Objektverzeichnis sind im Dokument L-1024 “CANopen Objektverzeichnis Software Manual” beschrieben.

OBD_CREATE_SRDO_GFC_PARAM()

Das Makro OBD_CREATE_SRDO_GFC_PARAM wird zum Anlegen des Eintrages “Global Fail-Safe Command Parameter” (Index 0x1300) verwendet. Das Makro hat keine Parameter.

OBD_CREATE_SRDO_COMMU(ind,num,dir,sct,srvt,cob1,cob2)

und

OBD_BEGIN_SRDO_MAPP(ind,num,cnt)
OBD_SUBINDEX_SRDO_MAPP(ind,sub,num,name,val)
OBD_END_SRDO_MAPP(ind)

Das Makro OBD_CREATE_SRDO_COMMU dient zur Definition der Kommunikationsparameter eines SRDO.

Die Makros OBD_xxx_SRDO_MAPP dienen der Definition der Mappingparameter eines SRDO. Ein Eintrag eines SRDO beginnt immer mit dem Makro OBD_BEGIN_SRDO_MAPP. Die verschiedenen Subindexeinträge werden mit dem Makro OBD_SUBINDEX_SRDO_MAPP definiert. Der Eintrag endet mit OBD_END_SRDO_MAPP.

Da zu einem SRDO immer die Kommunikationparameter und die Mappingparameter gehören, ist es wichtig, dass bei beiden die fortlaufende Nummer des SRDO richtig gesetzt ist.

ind: Objektindex des zu definierenden SRDO (0x1301 bis 0x1340 und 0x1381 bis 0x13C0 für das Mapping)

num: laufende Nummer von 0 bis 63 für den entsprechenden Tabelleneintrag Das erste bekommt immer die laufende Nummer 0 zugewiesen. Folgende Einträge erhalten immer die nächst größere Nummer vom Vorgänger. Sind zum Beispiel die SRDO 0x1301, 0x1302 und 0x1305 anzulegen, dann erhält das SRDO 0x1301 die 0, 01302 die 1 und 0x1305 die 2.

dir: Information Direction des SRDO. Der Wert entspricht Index 0x1301 bis 0x1340 Subindex 1.

sct: Refresh-Time / SCT des SRDO. Der Wert entspricht Index 0x1301 bis 0x1340 Subindex 2.

srvt: SRVT des SRDO. Der Wert entspricht Index 0x1301 bis 0x1340 Subindex 3.

cob1: COB-ID 1 des SRDO, d.h. CAN-Identifier der Nachricht die die normalen Daten enthält. Der Wert entspricht Index 0x1301 bis 0x1340 Subindex 5.

cob2: COB-ID 2 des SRDO, d.h. CAN-Identifier der Nachricht die die inversen Daten enthält. Der Wert entspricht Index 0x1301 bis 0x1340 Subindex 6.

cnt: Anzahl der Mappingeinträge des SRDO. Der Wert entspricht dem Objekteintrag 0x1381 bis 0x13C0 Subindex 0.

sub: Subindex des zu definierenden Mappingeintrages

name: Objektname

val: Default Wert für die Mappingdaten, der nach einem Reset angenommen wird

OBD_CREATE_SRDO_CFG_VALID()

Das Makro OBD_CREATE_SRDO_CFG_VALID wird zum Anlegen des Eintrages “Configuration Valid” (Index 0x13FE) verwendet. Das Makro hat keine Parameter.

OBD_BEGIN_SRDO_CRC(cnt)
OBD_SUBINDEX_SRDO_CRC(sub,name)
OBD_END_SRDO_CRC()

Die Makros dienen zur Definition der Objekteinträge “Safety Configuration Checksum” (Index 0x13FF).

cnt: Anzahl der CRC-Tabellen-Einträge. Sind bei den Indizes 0x1301 bis 0x1340 Lücken enthalten, so müssen auch dafür CRC-Einträge definiert werden. Es entspricht dann also immer das n-te SRDO dem n-ten Subindex der CRC

sub: Subindex des zu definierenden CRC-Eintrages

name: Objektname

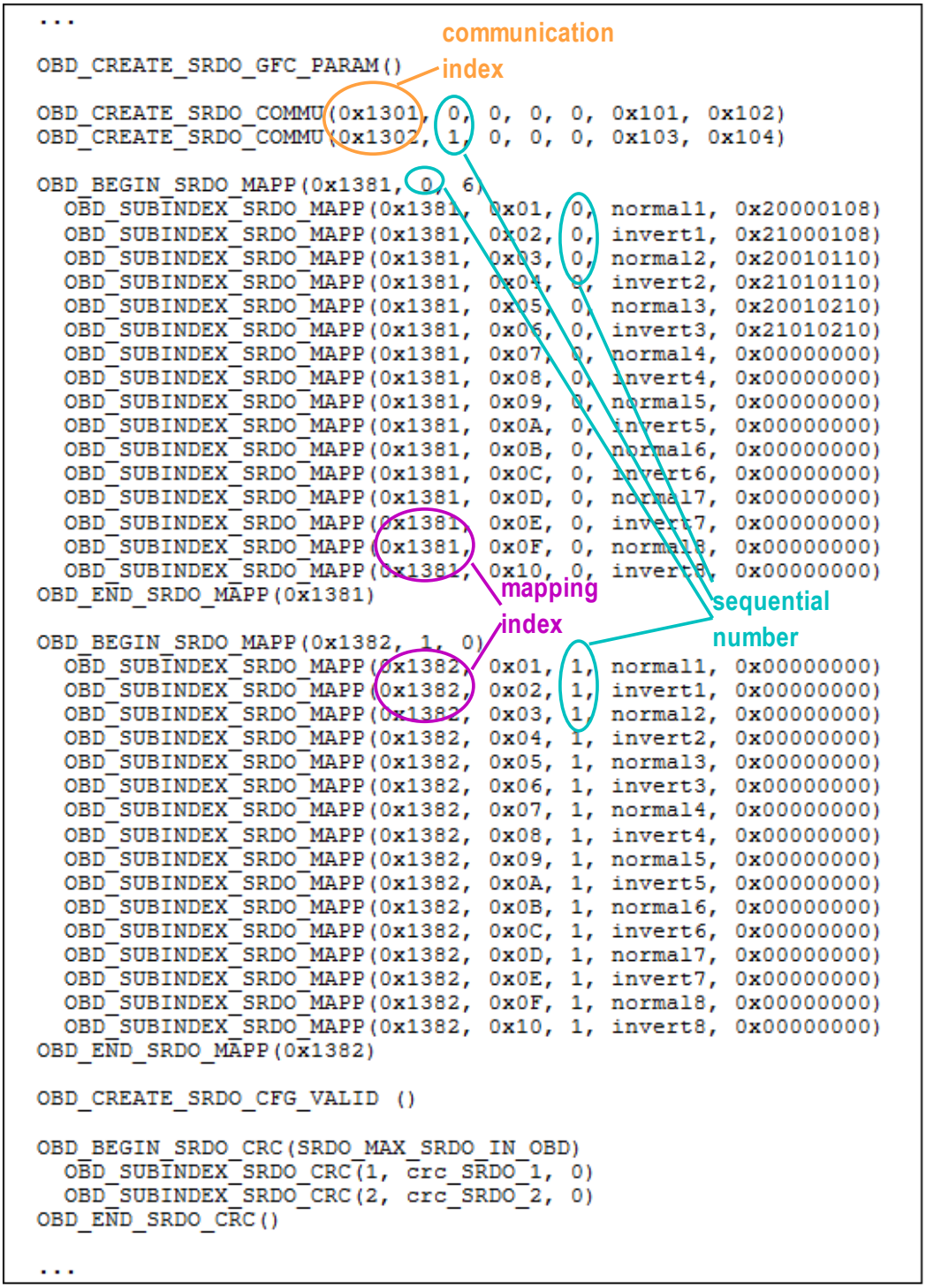
2.10.2 Hinweise für die Makros

Beachten Sie bitte, dass die Objekte im Objektverzeichnis immer in aufsteigender Reihenfolge anzulegen sind sonst kann der CANopen Stack die Objekte im OD nicht finden. Das bedeutet, dass die folgenden Makros immer in der unten angegebenen Reihenfolge angelegt werden müssen. Die Makros für die Kommunikations- und Mapping-Parameter können mehrfach auftreten, je nachdem wie viele SRDOs angelegt werden sollen.

```
OBD_CREATE_SRDO_GFC_PARAM()
OBD_CREATE_SRDO_COMMU(…)
OBD_BEGIN_SRDO_MAPP(…)
OBD_CREATE_SRDO_CFG_VALID()
OBD_BEGIN_SRDO_CRC(…)
```

Werden mehrere SRDOs in einem OD angelegt, dann muss darauf geachtet werden, dass jedes SRDO eine fortlaufende Nummer, beginnend mit 0, erhalten muss. Diese fortlaufende Nummer muss dem Makro OBD_CREATE_SRDO_COMMU() als zweiten Parameter, dem Makro OBD_BEGIN_SRDO_MAPP() ebenfalls als zweiten Parameter und dem Makro OBD_SUBINDEX_SRDO_MAPP() als dritten Parameter übergeben werden. Innerhalb eines SRDOs ist diese Nummer immer gleich. Das darauf folgende SRDO erhält immer die um 1 erhöhte Nummer. Die Nummer für die Kommunikations-Parameter eines SRDOs ist immer die gleiche Nummer wie die der dazugehörigen Mapping-Parameter. Es ist zu beachten, dass die absolute Anzahl der SRDOs mit dem Define SRDO_MAX_SRDO_IN_OBD in der Datei obdcfg.h übereinstimmen muss.

Werden im Objektverzeichnis die SRDOs mit Lücken angelegt, dann muss das Define SRDO_ALLOW_GAPS_IN_OD in der Datei copcfg.h auf TRUE gesetzt werden. Mit „Lücken“ ist gemeint, dass z.B. SRDO1 und SRDO3 im OD angelegt sind, jedoch SRDO2 nicht. In diesem Fall würde SRDO1 die laufende Nummer 0 und SRDO3 die laufende Nummer 1 erhalten. Eine eindeutige Zuordnung von Kommunikationsindex und laufender Nummer ist damit nicht mehr möglich.
Damit der CANopen Stack das entsprechende SRDO dennoch finden kann, muss der Stack einen anderen Suchalgorithmus implementieren, der zu einer höheren Laufzeit führen kann. Deshalb vermeiden Sie bitte solche Lücken im Objektverzeichnis.




```
...

OBD_CREATE_SRDO_GFC_PARAM()

OBD_CREATE_SRDO_COMMU(0x1301, 0, 0, 0, 0, 0x101, 0x102)
OBD_CREATE_SRDO_COMMU(0x1302, 1, 0, 0, 0, 0x103, 0x104)

OBD_BEGIN_SRDO_MAPP(0x1381, 0, 6)
  OBD_SUBINDEX_SRDO_MAPP(0x1381, 0x01, 0, normal1, 0x20000108)
  OBD_SUBINDEX_SRDO_MAPP(0x1381, 0x02, 0, invert1, 0x21000108)
  OBD_SUBINDEX_SRDO_MAPP(0x1381, 0x03, 0, normal2, 0x20010110)
  OBD_SUBINDEX_SRDO_MAPP(0x1381, 0x04, 0, invert2, 0x21010110)
  OBD_SUBINDEX_SRDO_MAPP(0x1381, 0x05, 0, normal3, 0x20010210)
  OBD_SUBINDEX_SRDO_MAPP(0x1381, 0x06, 0, invert3, 0x21010210)
  OBD_SUBINDEX_SRDO_MAPP(0x1381, 0x07, 0, normal4, 0x00000000)
  OBD_SUBINDEX_SRDO_MAPP(0x1381, 0x08, 0, invert4, 0x00000000)
  OBD_SUBINDEX_SRDO_MAPP(0x1381, 0x09, 0, normal5, 0x00000000)
  OBD_SUBINDEX_SRDO_MAPP(0x1381, 0x0A, 0, invert5, 0x00000000)
  OBD_SUBINDEX_SRDO_MAPP(0x1381, 0x0B, 0, normal6, 0x00000000)
  OBD_SUBINDEX_SRDO_MAPP(0x1381, 0x0C, 0, invert6, 0x00000000)
  OBD_SUBINDEX_SRDO_MAPP(0x1381, 0x0D, 0, normal7, 0x00000000)
  OBD_SUBINDEX_SRDO_MAPP(0x1381, 0x0E, 0, invert7, 0x00000000)
  OBD_SUBINDEX_SRDO_MAPP(0x1381, 0x0F, 0, normal8, 0x00000000)
  OBD_SUBINDEX_SRDO_MAPP(0x1381, 0x10, 0, invert8, 0x00000000)
OBD_END_SRDO_MAPP(0x1381)

OBD_BEGIN_SRDO_MAPP(0x1382, 1, 0)
  OBD_SUBINDEX_SRDO_MAPP(0x1382, 0x01, 1, normal1, 0x00000000)
  OBD_SUBINDEX_SRDO_MAPP(0x1382, 0x02, 1, invert1, 0x00000000)
  OBD_SUBINDEX_SRDO_MAPP(0x1382, 0x03, 1, normal2, 0x00000000)
  OBD_SUBINDEX_SRDO_MAPP(0x1382, 0x04, 1, invert2, 0x00000000)
  OBD_SUBINDEX_SRDO_MAPP(0x1382, 0x05, 1, normal3, 0x00000000)
  OBD_SUBINDEX_SRDO_MAPP(0x1382, 0x06, 1, invert3, 0x00000000)
  OBD_SUBINDEX_SRDO_MAPP(0x1382, 0x07, 1, normal4, 0x00000000)
  OBD_SUBINDEX_SRDO_MAPP(0x1382, 0x08, 1, invert4, 0x00000000)
  OBD_SUBINDEX_SRDO_MAPP(0x1382, 0x09, 1, normal5, 0x00000000)
  OBD_SUBINDEX_SRDO_MAPP(0x1382, 0x0A, 1, invert5, 0x00000000)
  OBD_SUBINDEX_SRDO_MAPP(0x1382, 0x0B, 1, normal6, 0x00000000)
  OBD_SUBINDEX_SRDO_MAPP(0x1382, 0x0C, 1, invert6, 0x00000000)
  OBD_SUBINDEX_SRDO_MAPP(0x1382, 0x0D, 1, normal7, 0x00000000)
  OBD_SUBINDEX_SRDO_MAPP(0x1382, 0x0E, 1, invert7, 0x00000000)
  OBD_SUBINDEX_SRDO_MAPP(0x1382, 0x0F, 1, normal8, 0x00000000)
  OBD_SUBINDEX_SRDO_MAPP(0x1382, 0x10, 1, invert8, 0x00000000)
OBD_END_SRDO_MAPP(0x1382)

OBD_CREATE_SRDO_CFG_VALID ()

OBD_BEGIN_SRDO_CRC(SRDO_MAX_SRDO_IN_OBD)
  OBD_SUBINDEX_SRDO_CRC(1, crc_SRDO_1, 0)
  OBD_SUBINDEX_SRDO_CRC(2, crc_SRDO_2, 0)
OBD_END_SRDO_CRC()

...
```

Abbildung 10: Beispiel eines OD mit 2 SRDOs

2.11 Funktionsbeschreibungen des SDRO Moduls

2.11.1 Funktion SrdoInit

Syntax:
#include <srdo.h>
tCopKernel PUBLIC SrdoInit (MCO_DECL_PTR_INSTANCE_PTR_
tSrdoInitParam MEM* pInitParam_p);

Parameter:
MCO_DECL_PTR_INSTANCE_PTR_: Pointer auf den Instanzpointer

pInitParam_p: Pointer auf die Parameterstruktur für das Initialisieren der SRDO Modul Instanz

Return:
kCopSuccessful Die Funktion wurde ohne Fehler ausgeführt.

kCopSrdoGranularityMismatch Die konfigurierte SRDO Granularität wird nicht unterstützt.

Weitere Returncodes des Standard CANopen sind möglich.

Beschreibung:
Die Funktion löscht die Instanztabelle und initialisiert mit Hilfe der Funktion **SrdoAddInstance()** die erste Instanz. Die Parameterstruktur **tSrdoInitParam** enthält die Parameter zum Initialisieren dieser Instanz und hat den folgenden Aufbau:

```
typedef struct
{
#if (COP_MAX_INSTANCES > 1)
    void MEM*           m_ObdInstance;
    void MEM*           m_CobInstance;
    void MEM*           m_CdrvInstance;
#endif
    tSrdoTabParam       m_SrdoTabParam;
    BYTE                m_bGranularity;
    BYTE MEM*           m_pbSrdoConfigValid;

} tSrdoInitParam;
```

2.11.2 Funktion SrdoAddInstance

Syntax:
#include <srdo.h>
tCopKernel PUBLIC SrdoAddInstance(
MCO_DECL_PTR_INSTANCE_PTR_
tSrdoInitParam MEM* pInitParam_p);

Parameter:
MCO_DECL_PTR_INSTANCE_PTR_: Pointer auf den Instanzpointer

pInitParam_p: Pointer auf die Parameterstruktur für das Initialisieren der SRDO Modul Instanz

Return:
kCopSuccessful Die Funktion wurde ohne Fehler ausgeführt.

kCopSrdoGranularityMismatch Die konfigurierte SRDO Granularität wird nicht unterstützt.

Weitere Returncodes des Standard CANopen sind möglich.

Beschreibung:
Diese Funktion fügt eine neue Instanz zum SRDO-Modul hinzu. Dafür muss das Define COP_MAX_INSTANCES größer als 1 sein. Ist kein freier Eintrag in der Instanztabelle verfügbar, gibt diese Funktion einen Fehler zurück. Die SRDO-Tabellen für diese Instanz werden initialisiert.

Der Aufbau der Parameterstruktur **tSrdoInitParam** ist in Kapitel 2.11.1. enthalten.

2.11.3 Funktion SrdoDeleteInstance

Syntax:
#include <srdo.h>
tCopKernel PUBLIC SrdoDeleteInstance (
MCO_DECL_INSTANCE_PTR);

Parameter:
MCO_DECL_INSTANCE_PTR: Pointer auf die Instanz

Return:
kCopSuccessful Die Funktion wurde ohne Fehler ausgeführt.

Weitere Returncodes des Standard CANopen sind möglich.

Beschreibung:
Die Funktion löscht alle angelegten Kommunikationsobjekte der angegebenen Instanz und markiert die Instanz als unbenutzt.

2.11.4 Funktion SrdoNmtEvent

Syntax:
#include <srdo.h>
tCopKernel PUBLIC SrdoNmtEvent (MCO_DECL_INSTANCE_PTR_
tNmtEvent NmtEvent_p);

Parameter:
MCO_DECL_INSTANCE_PTR_: Pointer auf die Instanz

NmtEvent_p: aufgetretenes NMT-Ereignis *(siehe L-1020)*

Return:
kCopSuccessful Die Funktion wurde ohne Fehler ausgeführt.

Weitere Returncodes des Standard CANopen sind möglich.

Beschreibung:
Die Funktion verarbeitet ein NMT Ereignis, das über die NMT State Machine ausgelöst wurde. Ein Ereignis führt zu einem Wechsel des NMT Knotenstatus. Je nach Knotenstatus wird die Ausführung des SRDO Moduls gesteuert.

2.11.5 Funktion SrdoSend

Syntax:
#include <srdo.h>
tCopKernel PUBLIC SrdoSend (MCO_DECL_INSTANCE_PTR_
WORD wSrdoCommuIndex_p);

Parameter:
MCO_DECL_INSTANCE_PTR_: Pointer auf die Instanz

wSrdoCommuIndex_p:	Objektindex der Kommunikations-Parameter des SRDO im Objektverzeichnis

Return:

kCopSuccessful	Die Funktion wurde ohne Fehler ausgeführt.
kCopSrdoNmtError	Die Aktion ist in diesem NMT-Zustand nicht erlaubt.
kCopSrdoInvalidCfg	Die Aktion wurde mit einer fehlerhaften SRDO Konfiguration versucht.
kCopSrdoNotExist	Das ausgewählte SRDO existiert nicht.
kCopSrdoRxTxConflict	Es wurde versucht, ein Empfangs-SRDO zu senden.
kCopSrdoInvalid	Die Aktion wurde mit einem ausgeschalteten SRDO versucht.

Weitere Returncodes des Standard CANopen sind möglich.

Beschreibung:

Die Funktion sendet ein über den Kommunikationsindex angegebenes SRDO oder alle SRDO bei Angabe von 0x0000 als Kommunikationsindex.
Siehe auch die zugehörige API-Funktion CcmSendSrdo().

2.11.6 Funktion SrdoProcess

Syntax:
#include <srdo.h>
tCopKernel PUBLIC SrdoProcess (MCO_DECL_INSTANCE_PTR)

Parameter:
MCO_DECL_INSTANCE_PTR: Pointer auf die Instanz

Return:

kCopSuccessful	Die Funktion wurde ohne Fehler ausgeführt.
kCopSrdoNotHandledInApp	Der an die Applikation gemeldete SRDO-Fehler wurde dort nicht korrekt bearbeitet.

Weitere Returncodes des Standard CANopen sind möglich.

Beschreibung:
Die Funktion wird statt der Funktion **CobProcessReceiveQueue()** gerufen. Sie behandelt den Empfang der CAN Nachrichten des CANopen Stacks. Empfangs-SRDO werden bevorzugt behandelt. Zusätzlich prüft diese Funktion die SCT von allen Empfangs-SRDOs. Ist die SCT abgelaufen, aber keine der beiden CAN-Nachrichten eines SRDOs empfangen worden, dann wird die Funktion AppSrdoError() mit dem Fehlercode kCopSrdoSctTimeout aufgerufen. Ist die Konstante SRDO_CHECK_SRVT_BEFORE_1STRX auf TRUE gesetzt, dann prüft diese Funktion auch die SRVT von allen SRDOs. Wurde nur eine der beiden CAN-Nachrichten empfangen und die SRVT ist abgelaufen, dann wird die Funktion AppSrdoError() mit dem Fehlercode kCopSrdoSrvtTimeout aufgerufen. Die Funktion SrdoProcess() muss zyklisch gerufen werden. Von ihr sind Schwankungen im Timing der SRDOs stark abhängig.

Die Funktion **SrdoProcess()** wird automatisch von **CcmProcess()** aus CcmMain.c gerufen, sobald das SRDO in CCM_MODUL_INTEGRATION aktiviert ist.

2.11.7 Funktion SrdoCheckConfig

Syntax:
#include <srdo.h>
tCopKernel PUBLIC SrdoCheckConfig (MCO_DECL_INSTANCE_PTR_
WORD * pwCommuIndex_p);

Parameter:
MCO_DECL_INSTANCE_PTR_: Pointer auf die Instanz

pwCommuIndex_p:	Pointer auf eine Variable, in die die Funktion bei fehlerhafter Konfiguration den Kommunikationsindex des fehlerhaften SRDO hinterlegt

Return:

kCopSuccessful	Die Funktion wurde ohne Fehler ausgeführt.
kCopSrdoCfgCrcError	Die SRDO Konfiguration ist fehlerhaft (CRC).

Beschreibung:
Die Funktion berechnet die Checksumme über alle SRDO (auch die deaktivierten SRDOs mit Direction = 0) und vergleicht diese mit der im OD konfigurierten. Stellt sie einen Fehler fest, so gibt sie einen Fehler und den entsprechenden Kommunikationsindex des fehlerhaften SRDO zurück. Die Funktion wird vom SRDO Modul beim Schreiben des Eintrags *Configuration Valid* (Index 0x13FE im Objektverzeichnis) mit dem Wert 0xA5 gerufen.

Siehe auch die zugehörige API-Funktion CcmCheckSrdoConfig().

2.11.8 Funktion SrdoSendGfc

Syntax:
#include <srdo.h>
tCopKernel PUBLIC SrdoSendGfc (MCO_DECL_INSTANCE_PTR)

Parameter:
MCO_DECL_INSTANCE_PTR: Pointer auf die Instanz

Return:

kCopSuccessful	Die Funktion wurde ohne Fehler ausgeführt.

Weitere Returncodes des Standard CANopen sind möglich.

Beschreibung:
Die Funktion sendet eine GFC-Nachricht.

Sie ist nicht vorhanden, wenn die Konfiguration SRDO_USE_GFC auf FALSE gesetzt ist.
Siehe zugehörige API-Funktion CcmSendGfc().

2.11.9 Funktion SrdoGetState

Syntax:
#include <srdo.h>
tCopKernel PUBLIC SrdoGetState (MCO_DECL_INSTANCE_PTR_
BYTE * pSrdoState_p,
WORD wSrdoCommuIndex_p);

Parameter:
MCO_DECL_INSTANCE_PTR_: Pointer auf die Instanz

pSrdoState_p:	Pointer auf den die Funktion den Status kopiert
wSrdoCommuIndex_p:	Objektindex der Kommunikations-Parameter des SRDO im Objektverzeichnis enthält

Return:

kCopSuccessful	Die Funktion wurde ohne Fehler ausgeführt.
kCopSrdoNotExist	Das ausgewählte SRDO existiert nicht.

Beschreibung:
Die Funktion liest den Status eines SRDO. Aufbau und Verwendung des Status siehe Kapitel 2.5.3.

Siehe zugehörige API-Funktion CcmGetSrdoState().

2.11.10 Funktion SrdoSetState

Syntax:
#include <srdo.h>
tCopKernel PUBLIC SrdoSetState (MCO_DECL_INSTANCE_PTR_
BYTE SrdoState_p,
WORD wSrdoCommuIndex_p);

Parameter:
MCO_DECL_INSTANCE_PTR_: Pointer auf die Instanz

SrdoState_p:	zu setzender Status
wSrdoCommuIndex_p:	Objektindex der Kommunikations-Parameter des SRDO im Objektverzeichnis enthält

Return:

kCopSuccessful	Die Funktion wurde ohne Fehler ausgeführt.
kCopSrdoNotExist	Das ausgewählte SRDO existiert nicht.

Beschreibung:
Die Funktion schreibt den Status eines SRDO. Aufbau und Verwendung des Status siehe Kapitel 2.5.3.

Siehe zugehörige API-Funktion CcmSetSrdoState().

2.11.11 Funktion SrdoGetCommuParam

Syntax:
#include <srdo.h>
tCopKernel PUBLIC SrdoGetCommuParam (
MCO_DECL_INSTANCE_PTR_
WORD wSrdoCommuIndex_p,
tSrdoCommuParam * pSrdoCommuParam_p);

Parameter:
MCO_DECL_INSTANCE_PTR_: Pointer auf die Instanz

wSrdoCommuIndex_p:	Objektindex der Kommunikations-Parameter des SRDO im Objektverzeichnis enthält
pSrdoCommuParam_p:	Pointer auf die Struktur, in die die Funktion die Werte für Information Direction und SCT kopiert

Return:

kCopSuccessful	Die Funktion wurde ohne Fehler ausgeführt.
kCopSrdoNotExist	Das ausgewählte SRDO existiert nicht.

Beschreibung:
Die Funktion liest die in der Applikation benötigten Parameter eines SRDO. Dies sind *Information Direction* und *SCT*.

Siehe zugehörige API-Funktion CcmGetSrdoParam().

2.11.12 Funktion SrdoGetMappParam

Syntax:
#include <srdo.h>
tCopKernel PUBLIC SrdoGetMappParam (MCO_DECL_INSTANCE_PTR
_
WORD wSrdoCommuIndex_p,
tSrdoMappParam * pSrdoMappParam_p);

Parameter:
MCO_DECL_INSTANCE_PTR_: Pointer auf die Instanz

wSrdoCommuIndex_p:	Objektindex der Kommunikations-Parameter des SRDO im Objektverzeichnis enthält
pSrdoCommuParam_p:	Pointer auf die Struktur, in die die Funktion die Werte für Number Of Mapped Objects und und die Variablenpointer kopiert

Return:

kCopSuccessful	Die Funktion wurde ohne Fehler ausgeführt.
kCopSrdoNotExist	Das ausgewählte SRDO existiert nicht.

Beschreibung:
Die Funktion liest die in der Applikation benötigten Mappingparameter eines SRDO. Dies sind *Number of Mapped Objects* und die Pointer auf die gemappten Variablen.
Siehe zugehörige API-Funktion CcmGetSrdoParam().

2.11.13 Funktion SrdoCalcSrdoCrc

Syntax:
#include <srdo.h>
tCopKernel PUBLIC SrdoCalcSrdoCrc (MCO_DECL_INSTANCE_PTR_
WORD wCommuIndex_p,
tSrdoTabEntry MEM* pSrdoEntry_p,
WORD* pwCrc_p);

Parameter:
MCO_DECL_INSTANCE_PTR_: Pointer auf die Instanz

wCommuIndex_p:	Objektindex der Kommunikations-Parameter des SRDO im Objektverzeichnis enthält
pSrdoEntry_p:	Muss immer mit NULL übergeben werden.
pwCrc_p:	Pointer auf eine WORD-Variable für den Empfang der 16 Bit CRC innerhalb der aufrufenden Funktion.

Return:

kCopSuccessful	Die Funktion wurde ohne Fehler ausgeführt.
kCopSrdoNotExist	Das ausgewählte SRDO existiert nicht.

Beschreibung:
Die Funktion berechnet die Checksumme (CRC) über ein SRDO und gibt diese der aufrufenden Funktion zurück. Die Berechnung erfolgt auch, wenn das SRDO ausgeschaltet ist. Es erfolgt kein Vergleich auf die Richtigkeit der CRC.
Die Applikation kann diese Funktionalität über die API-Funktion CcmCalcSrdoCrc () verwenden, um die CRC eines SRDOs zu aktualisieren, wenn Konfiguration eines SRDOs über die Applikation dynamisch neu gesetzt werden muss (z.B. Ändern der COB-ID in Abhängigkeit der Node-ID).

2.12 Funktionsbeschreibungen des SRDOSTC Moduls

Folgende Funktionen aus dem SRDO Modul sind ebenfalls im SRDOSTC Modul implementiert. Deren Bedeutung und Syntax können aus Kapitel 2.11 entnommen werden:

SrdoInit(), SrdoAddInstance(), SrdoDeleteInstance(), SrdoNmtEvent(), SrdoSend(), SrdoProcess(), SrdoCheckConfig(), SrdoSendGfc(), SrdoGetState(), SrdoSetState(), SrdoGetCommuParam().

2.12.1 Funktion SrdoStaticDefineVarFields

Syntax:

```
#include <srdo.h>
tCopKernel PUBLIC SrdoStaticDefineVarFields(
                                    MCO_DECL_INSTANCE_PTR
                                    _
                                    WORD        wCommuIndex_p,
                                    void MEM* pNormalData_p,
                                    void MEM* pInversData_p);
```

Parameter:

MCO_DECL_INSTANCE_PTR_: Pointer auf die Instanz

wCommuIndex_p:	Kommunikations-Index des SRDO im OD, dessen Variablen definiert werden sollen.
pNormalData_p:	Pointer auf ein zusammenhängendes Variablenfeld, das mit den normalen Daten des SRDO verknüpft (bzw. gemappt) werden soll.
pInversData_p:	Pointer auf ein zusammenhängendes Variablenfeld, das mit den inversen Daten des SRDO verknüpft (bzw. gemappt) werden soll.

Return:

kCopSuccessful	Die Funktion wurde ohne Fehler ausgeführt.
kCopSrdoNotExist	Das ausgewählte SRDO existiert nicht.
kCopSrdoErrorMapp	Das Mapping eines SRDO ist fehlerhaft.
kCopSrdoLengtExceeded	Die Länge des gewählten SRDO Mapping überschreitet 64 Bit.

Beschreibung:

Diese Funktion definiert für ein SRDO die Variablenfelder. Die Applikation ändert die Variablen nur über diese Variablenfelder. Beim Senden eines SRDO werden genau diese Datenbytes aus dem Variablenfeld in die beiden CAN Nachrichten kopiert. Beim Empfang eines SRDO werden die Datenbytes der CAN Nachrichten direkt in die Variablenfelder kopiert.

Die Funktion prüft, ob die angegeben Variablenfelder auch mit den Variablen übereinstimmen, auf die das Mapping im OD zeigt.

Siehe zugehörige API-Funktion CcmStaticDefineSrdoVarFields()

2.13 erweiterte CANopen Returncodes

Die CANopen-Returncodes sind in der Datei **errordef.h** definiert.

Fehlercode	Bedeutung
kCopSuccessful	Die Funktion wurde ohne Fehler ausgeführt.
kCopSrdoNotExist	Das ausgewählte SRDO existiert nicht.
kCopSrdoGranularityMismatch	Die konfigurierte SRDO Granularität wird nicht unterstützt.
kCopSrdoCfgTimingError	Die SRDO Konfiguration ist fehlerhaft (Zeitkonfiguration SCT – SRVT).
kCopSrdoCfgIdError	Die SRDO Konfiguration ist fehlerhaft (COB-Ids).
kCopSrdoCfgCrcError	Die SRDO Konfiguration ist fehlerhaft (CRC).
kCopSrdoNmtError	Die Aktion ist in diesem NMT-Zustand nicht erlaubt.
kCopSrdoInvalidCfg	Die Aktion wurde mit einer fehlerhaften SRDO Konfiguration versucht.
kCopSrdoInvalid	Die Aktion wurde mit einem ausgeschalteten SRDO versucht.
kCopSrdoRxTxConflict	Es wurde versucht, ein Empfangs-SRDO zu senden.
kCopSrdoIllegalCanId	Der CAN Identifier ist ungültig.
kCopSrdoCanIdAlreadyInUse	Der CAN Identifier wird schon verwendet.
kCopSrdoNotInOrder	Die zwei CAN Nachrichten eines SRDO wurden in der falschen Reihenfolge empfangen.
kCopSrdoSctTimeout	Die SCT eines Empfangs-SRDO wurde überschritten.
kCopSrdoSrvtTimeout	Die SRVT eines Empfangs-SRDO wurde überschritten.
kCopSrdoCanIdNotValid	Mindestens eine der beiden empfangenen CAN Identifier eines SRDO ist fehlerhaft.
kCopSrdoDlcNotValid	Mindestens eine der beiden empfangenen CAN Nachrichtenlängen eines SRDO ist fehlerhaft.
kCopSrdoErrorMapp	Das Mapping eines SRDO ist fehlerhaft.
kCopSrdoDataError	Die Daten der CAN Nachrichten eines SRDO sind nicht invers.
kCopSrdoLengtExceeded	Die Länge des gewählten SRDO Mapping überschreitet 64 Bit.
kCopSrdoNotHandledInApp	Der an die Applikation gemeldete SRDO-Fehler wurde dort nicht korrekt bearbeitet.

3 Referenzumgebung TMDX570LS20SMDK

Das Entwicklungsboard TMDX570LS20SMDK wird von der Firma Texas Instruments bereitgestellt. Es dient als Referenzumgebung für unsere Safety-Erweitung.

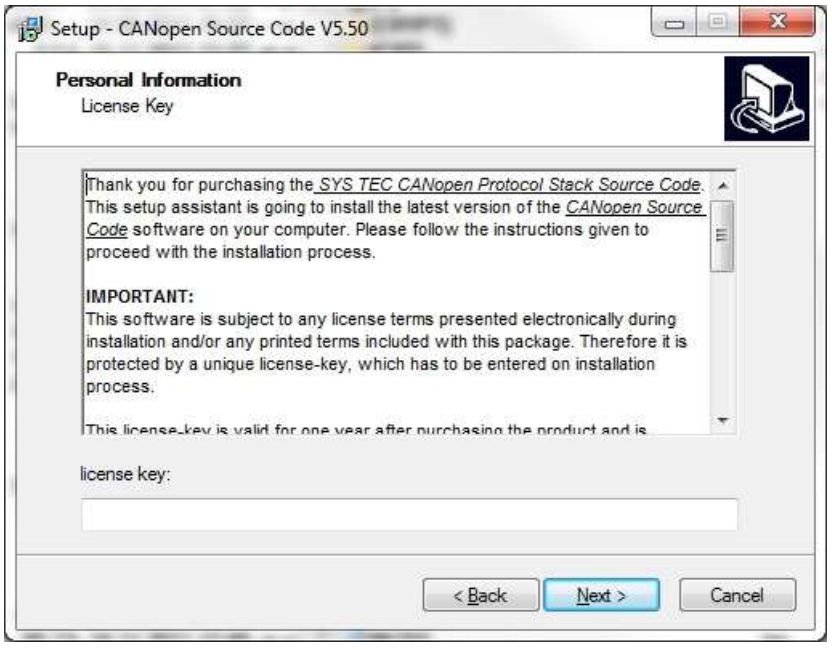
Für die Handhabung der Projekte in unserer Erweiterung gibt es einiges zu beachten. Dieses Kapitel beschreibt all diese Dinge, um Ihnen die ersten Schritte mit dem Projekt und der Hardware zu erleichtern.

3.1 Installation der Entwicklungsumgebung

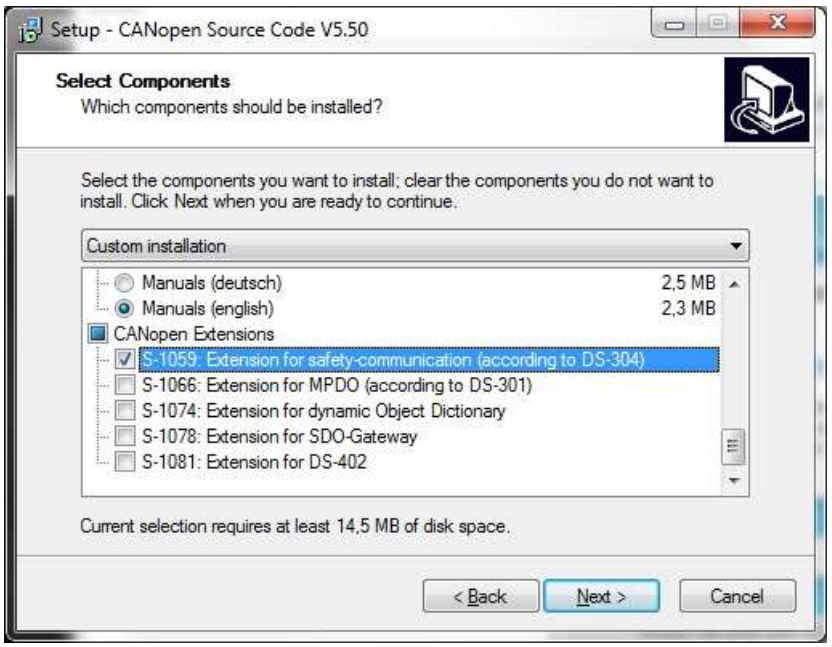
Mit dem Development Kit TMDX570LS20SMDK haben Sie eine CD erhalten, auf der sich die Entwicklungsumgebung Code Composer Studio befindet. Das Safety-Demo wurde mit der Version V4.2.3 erstellt und getestet. Installieren Sie die Entwicklunssoftware von dieser CD und fahren Sie danach mit der Installation der CANopen Software fort.

3.2 Installation der CANopen Software

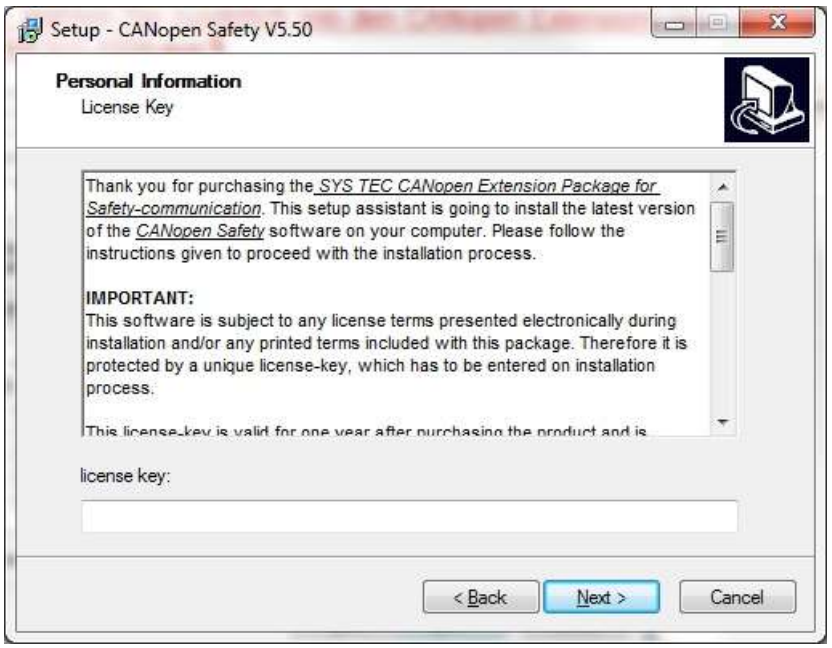
Zuerst muss der CANopen Stack SO-877 installiert werden. Starten Sie die Installation aus dem Autostart-Menü der SYS TEC electronic Produkt-CD. Die Version des CANopen Stack muss unbedingt größer oder gleich V5.51 sein. In einer früheren Version existiert das Projekt für den TMS570LS noch nicht. Nach dem Willkommensbildschirm, dem Akzeptieren der Lizenzbestimmungen und der Eingabe der Anwenderinformationen sehen Sie folgenden Dialog zur Eingabe des Lizenzschlüssels unseres CANopen Stacks:



Geben Sie hier den erworbenen Lizenzschlüssel ein und drücken Sie auf „Next“. Im Folgenden Dialog wählen Sie die zu installierenden Demo-Projekte aus. Wählen Sie zusätzlich von den CANopen Extensions das Software-Paket SO-1059 aus.



Folgen Sie allen weiteren Aufforderungen im Setup. Nach der Installation von SO-877 wird automatisch die Erweiterung SO-1059 installiert. Dabei müssen Sie einen weiteren Lizenzschlüssel für das Paket SO-1059 eingeben.

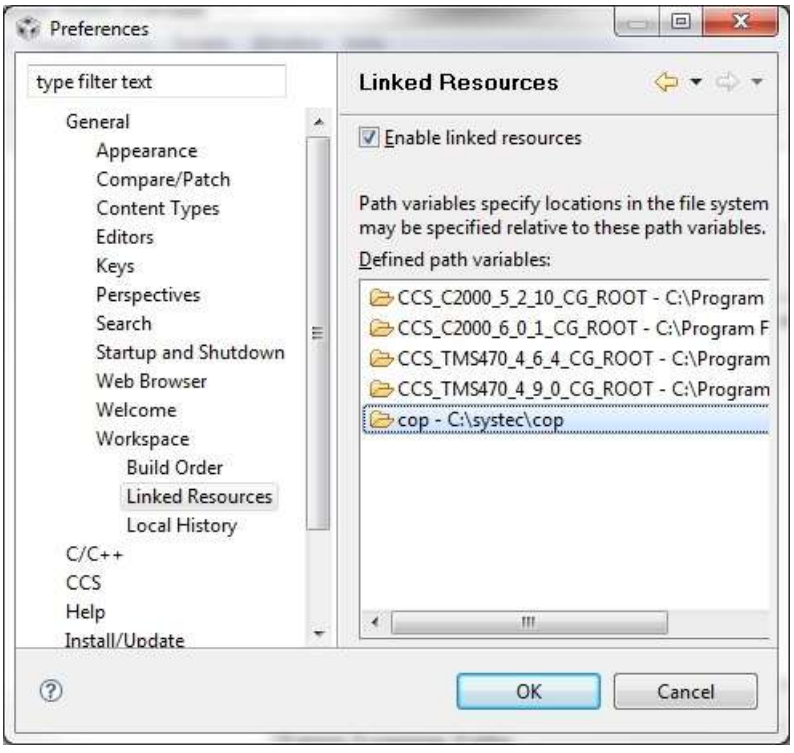


3.3 Importieren des Safety Demo im Code Composer Studio

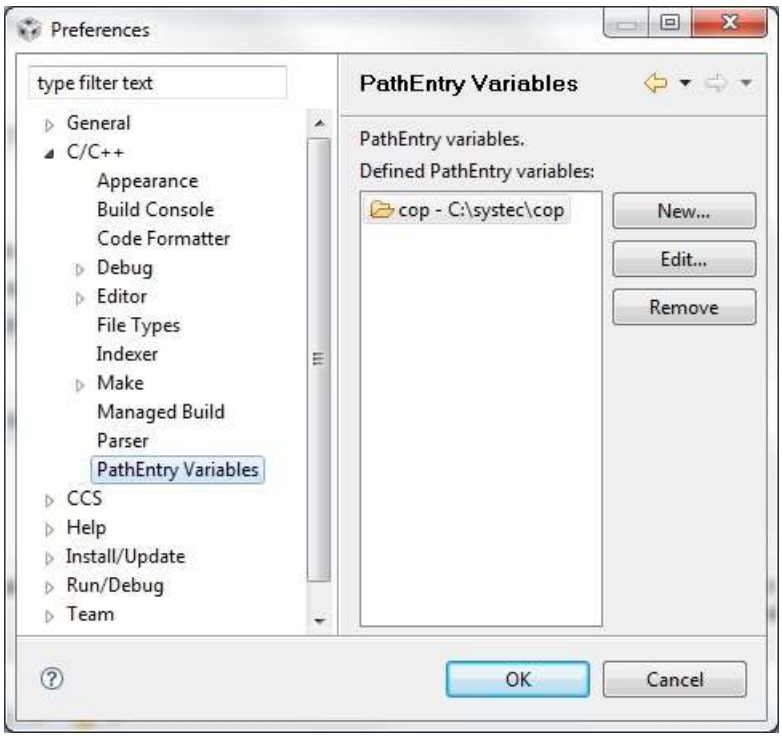
Wenn die Installation des CANopen Stack und der Safety-Erweiterung abgeschlossen ist, dann finden Sie im Pfad C:\systec\cop\target\TMDX570LS20SMDK\no_os\CodeComposerStudio\ demo_srdo_actor\ das Demo für den Aktuator auf dem TMS570LS Development Kit. Bitte achten Sie unbedingt darauf, dass die Dateien .ccsproject, .cdtbuild, .cdtproject und .project in diesem Verzeichnis nicht als „Versteckt“ gekennzeichnet sind. Andernfalls kann das Projekt nicht in den Code Composer Studio importiert werden. Entfernen Sie bitte gegebenenfalls das Datei-Attribut „Versteckt“, wenn es gesetzt sein sollte.

Nun starten Sie das Code Composer Studio. Sie werden aufgefordert, einen Workspace anzulegen. Schließen Sie dies durch Eingabe eines Verzeichnisses Ihrer Wahl ab.

Rufen Sie dann im Code Composer Studio das Menü ***Window → Preferences*** auf. Klappen Sie dann im linken Teil des Fensters das Menü ***General → Workspace → Linked Resources*** auf. Im rechten Fenster fügen Sie bitte mit dem Button ***New*** einen neuen Eintrag an: Name „cop“ und Location „C:\systec\cop“. Achten Sie bitte dabei auf die Groß- du Kleinschreibung. Am Ende sollte der Dialog bei Ihnen folgendermaßen aussehen:

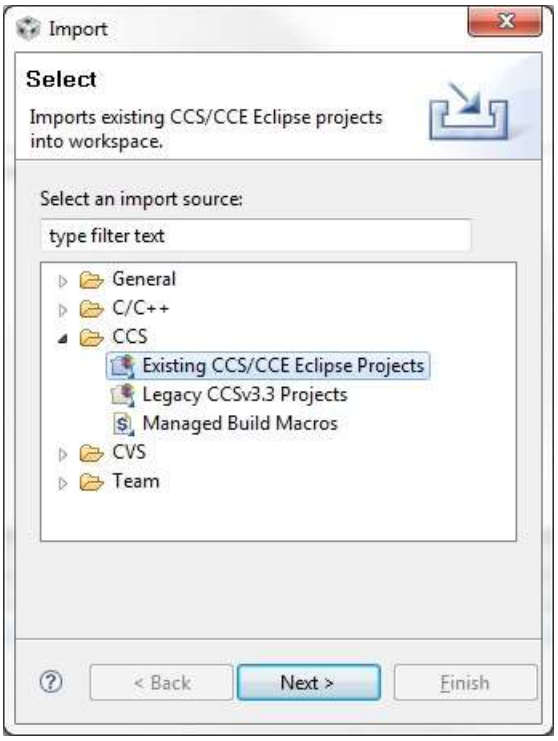


Wechseln Sie im linken Teil des Fensters auf ***C/C++ → PathEntry Variables***. Fügen Sie dort ebenfalls mit dem Button ***New…*** einen neuen Eintrag mit dem Namen ***cop*** und dem Verzeichnis ***C:\systec\cop*** hinzu.

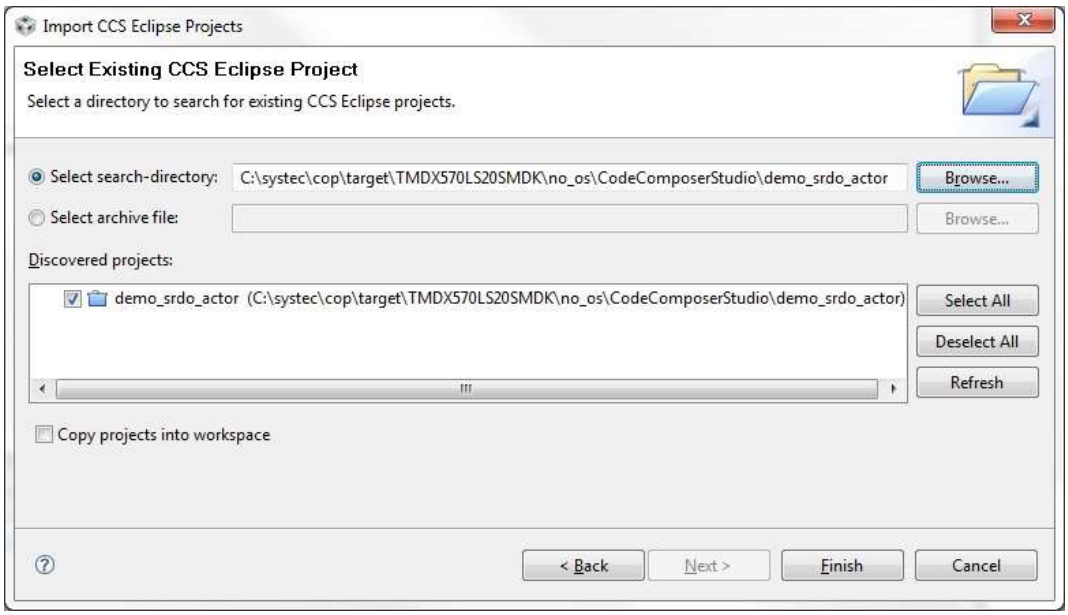


Bestätigen Sie die Eingabe mit OK.

Nun importieren Sie das Projekt über das Menü ***File → Import…*** . Wählen Sie dann im folgenden Dialog unter ***CCS*** die Zeile ***Existing CCS/CCE Eclipse Projects*** aus und bestätigen Sie mit ***Next***.



Im folgenden Dialog wählen Sie über den ***Browse***-Button den Pfad auf das Demo und klicken Sie danach auf den Button ***Finish***.



Wenn all diese Schritte Problemlos durchgeführt worden sind, kann das Projekt neu erstellt werden.

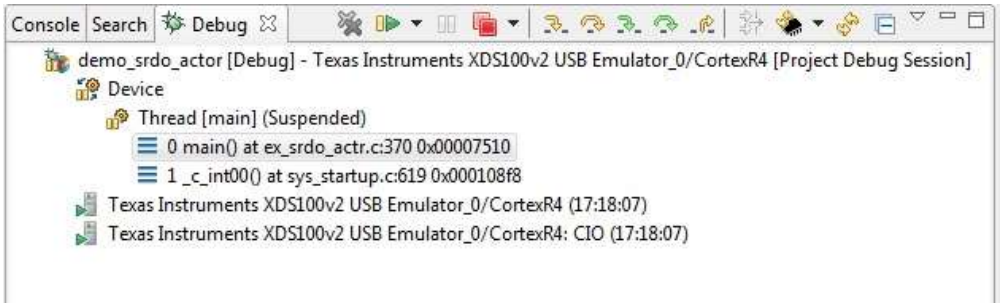
3.4 Debuggen des Demo auf der Hardware

Schließen Sie nun das Development Kit TMS570 an den PC an. Verwenden Sie dazu das mitgelieferte USB-Kabel und stecken Sie es auf der oberen Platine in die USB-Mini Buchse mit der Bezeichnung ***XDS100V2***. Nun sucht Windows nach den Gerätetreibern für das Development Kit. Diese Gerätetreiber wurden mit der Installations-CD von Texas Instruments installiert.

Nach der Installation der Gerätetreiber können Sie die Spannungsversorgung an das Development Kit anschließen. Mit dem Development Kit wurde dazu ein 12V Steckernetzteil mitgeliefert. Verbinden Sie es mit der Buchse auf der oberen Platine neben der USB-Mini-Buchse.

Nun klicken Sie im Code Composer Studio mit der rechten Maustaste auf das Projekt im linken Teil des Fensters und wählen dann im Context-Menü
Debug As → Debug Session. Beim aller ersten Mal müssen Sie den Typ der CPU auswählen. Wählen Sie dabei ***TMS570LS20216SZWT*** aus. Nach der Bestätigung programmiert der Code Composer das Demo in den Flash des Mikrocontrollers und hält danach in der main()-Funktion an.

Im Teilfenster ***Debug*** können Sie nun mit den Symbolen die Programmausführung steuern.

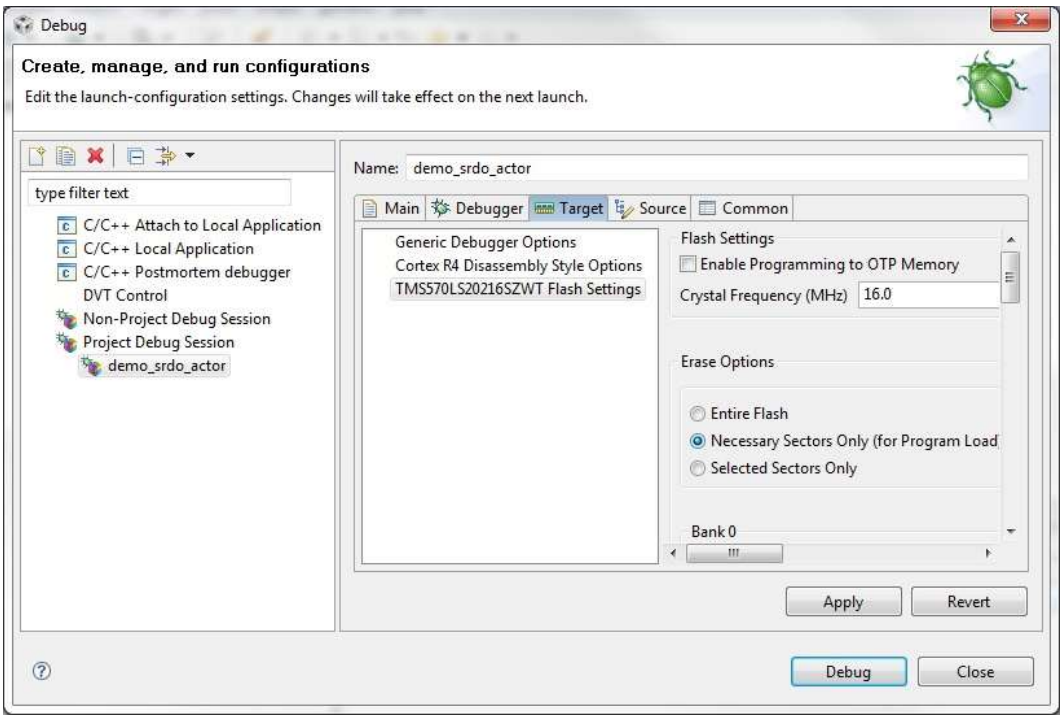


Wollen Sie das Debuggen beenden, dann wechseln Sie einfach die Perspektive zurück nach ***C/C++***. Klicken Sie dazu im oberen rechten Bereich des Code Composer Studios auf das Symbol links neben ***Debug***. Im folgenden Contex-Menü wird dann ***C/C++*** angeboten.

Nun befinden Sie sich wieder Projekt-Explorer des Code Composer Studios.

Das Programmieren der Firmware in den Flash dauert relativ lange, da zuerst der gesamte Flash gelöscht wird. Daher sollten Sie die Debug-Optionen so ändern, dass nur die Flash-Sektoren gelöscht werden sollen, die von der Applikation verwendet werden.

Klicken Sie dazu mit der rechten Maustaste auf das Projekt und wählen Sie im Context-Menü ***Debug As → Debug…*** . Wechseln Sie dann im folgenden Dialog auf der rechten Seite des Fensters auf den TabSheet ***Target***. Dann wählen Sie weiter unten die Zeile ***TMS570LS20216SZWT Flash Settings***. Nun finden Sie auf der rechten Seite die ***Erase Options***. Wählen Sie dort ***Necessary Sectors Only*** und bestätigen Sie mit ***Apply***.



4 Abkürzungsverzeichnis

BOM	Bill of Material (Stückliste)
CAN	Controller Area Network (gemäß ISO 11898-1:2003 und ISO 11898-2:2003)
CCM	CANopen Controlling Module
CiA	CAN in Automation e.V.
COB	Communication Object
CPU	Central Processing Unit
CRC	Cyclic redundancy check
DIN	Deutsches Institut für Normung e.V.
DLL	Data Link Layer (layer 2 according to OSI model)
EDS	Electronic Data Sheet
EEPROM	Electrically Erasable Programmable Read-Only Memory
EN	European Norm
EUC	Equipment under control
e.V.	eingetragener Verein
GFC	Global Fail Command, gemäß EN 50325-5:2010
GmbH	Gesellschaft mit beschränkter Haftung
GND	Ground (Masse)
HW	Hardware
ID	Identifier
IEC	International Electro technical Commission
IEEE	Institute of Electrical and Electronics Engineers
I/O	Input/Output
ISM	Industrial, Scientific and Medical
ISO	International Organization for Standardization
JTAG	Joint Test Action Group
kiB	Kilobyte
LSB	Least Significant Bit
MiB	Mega byte
ms	Millisekunden
MSB	Most Significant Bit
nc	not connected
NMT	Network Management
NSR	Non-safety-related
OD	Object Dictionary
OS	Operating System
OSI	Open Systems Interconnection model according to ISO 7498-1
PCB	Printed Circuit Board
PDF	Portable Document Format
PDO	Process Data Object
PhL	Physical Layer (layer 1 according to OSI model)
PHY	Physical layer in OSI model
RAM	Random-Access Memory
ro	read-only
ROM	Read-Only Memory
RPDO	Receive PDO
RSRDO	Receive safety-related data object, gemäß EN 50325-5:2010
RT	Real Time
RTC	Real Time Clock

rw	read-write
RX	Receive
SCL	Safety Communication Layer
SCT	Safeguard Cycle Time, gemäß EN 50325-5:2010
SDO	Service Data Object
sec	Seconds
SIL	Safety integrity level
SR	Safety-related
SRAM	Static RAM
SRDO	Safety-related data object, gemäß EN 50325-5:2010
SRVT	Safety-related validation time, gemäß EN 50325-5:2010
SW	Software
tbd	to be defined
TPDO	Transmit PDO
TSRDO	Transmit safety-related data object, gemäß EN 50325-5:2010
TTL	Transistor-Transistor-Logik
TX	Transmit
u.a.	unter anderem
UART	Universal Asynchronous Receiver Transmitter
UTC	Coordinated Universal Time

Index

Dokument:	CiA 304 Safety Framework
Dokumentnummer:	L-1077d_08, Auflage September 2015

Wie würden Sie dieses Handbuch verbessern?

Haben Sie in diesem Handbuch Fehler entdeckt? Seite

Eingesandt von:

Kundennummer:

Name:

Firma:

Adresse:

Einsenden an: Fax : +49 (0) 3765 / 38600-4100

SYS TEC electronic GmbH
Am Windrad 2
D-08468 Heinsdorfergrund
GERMANY

Veröffentlicht von

Best.-Nr. L-1077d_08
Printed in Germany