SIEMENS

SIMOTION

SIMOTION SCOUT Basisfunktionen

Funktionshandbuch

11/2010

Rechtliche Hinweise

Warnhinweiskonzept

Dieses Handbuch enthält Hinweise, die Sie zu Ihrer persönlichen Sicherheit sowie zur Vermeidung von Sachschäden beachten müssen. Die Hinweise zu Ihrer persönlichen Sicherheit sind durch ein Warndreieck hervorgehoben, Hinweise zu alleinigen Sachschäden stehen ohne Warndreieck. Je nach Gefährdungsstufe werden die Warnhinweise in abnehmender Reihenfolge wie folgt dargestellt.

⚠ GEFAHR
bedeutet, dass Tod oder schwere Körperverletzung eintreten **wird**, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

⚠ WARNUNG
bedeutet, dass Tod oder schwere Körperverletzung eintreten **kann**, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

⚠ VORSICHT
mit Warndreieck bedeutet, dass eine leichte Körperverletzung eintreten kann, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

VORSICHT
ohne Warndreieck bedeutet, dass Sachschaden eintreten kann, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

ACHTUNG
bedeutet, dass ein unerwünschtes Ergebnis oder Zustand eintreten kann, wenn der entsprechende Hinweis nicht beachtet wird.

Beim Auftreten mehrerer Gefährdungsstufen wird immer der Warnhinweis zur jeweils höchsten Stufe verwendet. Wenn in einem Warnhinweis mit dem Warndreieck vor Personenschäden gewarnt wird, dann kann im selben Warnhinweis zusätzlich eine Warnung vor Sachschäden angefügt sein.

Qualifiziertes Personal

Das zu dieser Dokumentation zugehörige Produkt/System darf nur von für die jeweilige Aufgabenstellung **qualifiziertem Personal** gehandhabt werden unter Beachtung der für die jeweilige Aufgabenstellung zugehörigen Dokumentation, insbesondere der darin enthaltenen Sicherheits- und Warnhinweise. Qualifiziertes Personal ist auf Grund seiner Ausbildung und Erfahrung befähigt, im Umgang mit diesen Produkten/Systemen Risiken zu erkennen und mögliche Gefährdungen zu vermeiden.

Bestimmungsgemäßer Gebrauch von Siemens-Produkten

Beachten Sie Folgendes:

⚠ WARNUNG
Siemens-Produkte dürfen nur für die im Katalog und in der zugehörigen technischen Dokumentation vorgesehenen Einsatzfälle verwendet werden. Falls Fremdprodukte und -komponenten zum Einsatz kommen, müssen diese von Siemens empfohlen bzw. zugelassen sein. Der einwandfreie und sichere Betrieb der Produkte setzt sachgemäßen Transport, sachgemäße Lagerung, Aufstellung, Montage, Installation, Inbetriebnahme, Bedienung und Instandhaltung voraus. Die zulässigen Umgebungsbedingungen müssen eingehalten werden. Hinweise in den zugehörigen Dokumentationen müssen beachtet werden.

Marken

Alle mit dem Schutzrechtsvermerk ® gekennzeichneten Bezeichnungen sind eingetragene Marken der Siemens AG. Die übrigen Bezeichnungen in dieser Schrift können Marken sein, deren Benutzung durch Dritte für deren Zwecke die Rechte der Inhaber verletzen kann.

Haftungsausschluss

Wir haben den Inhalt der Druckschrift auf Übereinstimmung mit der beschriebenen Hard- und Software geprüft. Dennoch können Abweichungen nicht ausgeschlossen werden, so dass wir für die vollständige Übereinstimmung keine Gewähr übernehmen. Die Angaben in dieser Druckschrift werden regelmäßig überprüft, notwendige Korrekturen sind in den nachfolgenden Auflagen enthalten.

Siemens AG
Industry Sector
Postfach 48 48
90026 NÜRNBERG
DEUTSCHLAND

Vorwort

Inhalt

Das vorliegende Dokument ist Bestandteil des **Dokumentationspaketes System- und Funktionsbeschreibungen**.

Gültigkeitsbereich

Dieses Handbuch ist gültig für SIMOTION SCOUT Produktstufe V4.2:

- SIMOTION SCOUT V4.2 (Engineering System der Produktfamilie SIMOTION),
- SIMOTION Kernel V4.2, V4.1, V4.0
- SIMOTION Technologiepakete CAM, CAM_ext und TControl in der zum jeweiligen Kernel passenden Version.

Informationsblöcke des Handbuches

Das vorliegende Handbuch beschreibt allgemein gültige Funktionen von SIMOTION und von Technologieobjekten.

- **Systemübersicht**

  Allgemeine Informationen zu SIMOTION

- **Technologiepakete und Technologieobjekte**

  Grundlegende Informationen zu den Technologiepaketen und zu Technologieobjekten

- **Symbolische Zuordnung**

  Informationen zur symbolischen Zuordnung von SINAMICS-Objekten an SIMOTION.

- **Programmieren von Technologieobjekten**

  Informationen wie und mit welchen Systemfunktionen Technologieobjekte programmiert werden können.

- **Fehlerbehandlung bei Technologieobjekten**

  Allgemeine Informationen zu technologischen Alarmen und Rückgabewerten von Befehlen

- **Ablaufsystem/Tasks/Systemtakte**

  Informationen zum Ablaufsystem und über Tasks

- **Programmieren Ablaufsystem/Tasks/Systemtakte**

  Informationen zur Programmierung des Ablaufsystems, von Tasks und von Systemtakten

- **Programmierung allgemeiner Systemfunktionen**

  Informationen welche allgemeinen Systemfunktionen zur Verfügung stehen und wie sie verwendet werden.

- **Programmierung allgemeiner Systemfunktionsbausteine**

  Informationen welche allgemeinen Systemfunktionsbausteine zur Verfügung stehen und wie sie verwendet werden.
- **SIMOTION Speicherkonzept (im Zielgerät)**

  Informationen zum Speicherkonzept im Zielgerät
- **Daten in das Zielgerät laden**

  Informationen über den Download von Daten in das Zielgerät bzw. Zielsystem
- **Anhang**

  Informationen zu Symbolischen Konstanten und Bezeichnern
- **Index**

  Stichwortverzeichnis zum Finden der Informationen

SIMOTION Dokumentation

Einen Überblick zur SIMOTION Dokumentation erhalten Sie in einem separaten Literaturverzeichnis.

Diese Dokumentation ist als elektronische Dokumentation im Lieferumfang von SIMOTION SCOUT enthalten und besteht aus 10 Dokumentationspaketen.

Zur SIMOTION Produktstufe V4.2 stehen folgende Dokumentationspakete zur Verfügung:

- SIMOTION Engineering System Handhabung
- SIMOTION System- und Funktionsbeschreibungen
- SIMOTION Service und Diagnose
- SIMOTION IT
- SIMOTION Programmieren
- SIMOTION Programmieren - Referenzen
- SIMOTION C
- SIMOTION P
- SIMOTION D
- SIMOTION Ergänzende Dokumentation

Hotline und Internetadressen

Weiterführende Informationen

Unter folgendem Link finden Sie Informationen zu den Themen:

- Dokumentation bestellen / Druckschriftenübersicht
- Weiterführende Links für den Download von Dokumenten
- Dokumentation online nutzen (Handbücher/Informationen finden und durchsuchen)

http://www.siemens.com/motioncontrol/docu

Bei Fragen zur technischen Dokumentation (z. B. Anregungen, Korrekturen) senden Sie bitte eine E-Mail an folgende Adresse:
docu.motioncontrol@siemens.com

My Documentation Manager

Unter folgendem Link finden Sie Informationen, wie Sie Dokumentation auf Basis der Siemens Inhalte individuell zusammenstellen und für die eigene Maschinendokumentation anpassen:

http://www.siemens.com/mdm

Training

Unter folgendem Link finden Sie Informationen zu SITRAIN - dem Training von Siemens für Produkte, Systeme und Lösungen der Automatisierungstechnik:

http://www.siemens.com/sitrain

FAQs

Frequently Asked Questions finden Sie in den Service&Support-Seiten unter **Produkt Support**:

http://support.automation.siemens.com

Technical Support

Landesspezifische Telefonnummern für technische Beratung finden Sie im Internet unter **Kontakt**:

http://www.siemens.com/automation/service&support

Inhaltsverzeichnis

Systemübersicht

1

1.1 Systemarchitektur

1.1.1 SIMOTION Systemarchitektur

Die SIMOTION Systemarchitektur ermöglicht Dezentralisierung, unterschiedliche Zielsysteme und verteilte Intelligenz.

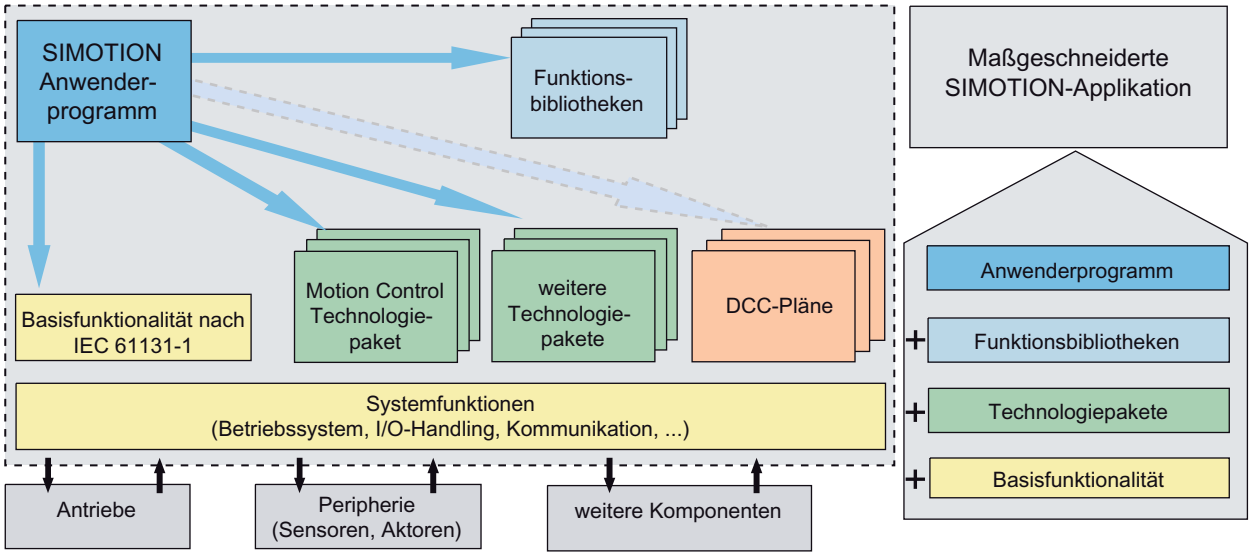


Bild 1-1 SIMOTION Systemarchitektur

Die Basisfunktionalität (SIMOTION Kernel) des Gerätes beinhaltet Funktionen für Steuern und Regeln sowie Logik und Arithmetik. Die Programmbearbeitung kann zyklisch-, Zeit- oder Interrupt-gesteuert erfolgen. Der SIMOTION Kernel enthält somit die Funktionalität, die bei nahezu allen Anwendungen erforderlich ist und entspricht im Wesentlichen einer PLC mit dem Befehlsvorrat nach IEC 61131-3 sowie Systemfunktionen zur Steuerung verschiedener Komponenten, z. B. Ein- /Ausgänge.

Den SIMOTION Kernel des Gerätes können Sie durch Laden von Technologiepaketen erweitern. Auf die Technologiepakete greifen Sie aus dem Anwenderprogramm über zusätzliche Sprachbefehle in gleicher Weise zu wie auf den SIMOTION Kernel.

Bei besonderen Aufgaben können Sie auf bereits erstellte Applikationen zurückgreifen oder diese selbst programmieren und einbinden. Die Applikationen werden IEC 61131-3 konform programmiert und können an Ihre spezifische Aufgabe angepasst werden.

Darüber hinaus stellt SIMOTION Funktionsbibliotheken bereit, die Systemfunktionen und Bewegungsfunktionen abdecken. Sie enthalten Funktionen und Zugriffe auf Systemvariablen eines Technologieobjekts und werden im SIMOTION SCOUT mit dem zugehörigen Gerät und dem Technologiepaket verknüpft.

Für besondere Aufgaben, z. B. Regelungsaufgaben, können Sie Verschaltungspläne nutzen und die mit einem grafischen Tool verschalteten Bausteine in den entsprechenden DCC-Ablauftasks zum Ablauf bringen.

1.1.2 Engineering System SIMOTION SCOUT

Im Rahmen einer Maschinenautomatisierung müssen verschiedene Tätigkeiten durchgeführt werden:

- Auswahl/Projektierung der Komponenten in Hard- und Software, Konfiguration der Hard- und Software, einschließlich der Kommunikationsnetzwerke mit **HW Konfig**
- Anlegen und Konfiguration der Technologieobjekte
- Erstellung des Anwenderprogramms
- Test und die Inbetriebnahme der Antriebsgeräte
- Einbindung der Maschinenbedienung (HMI)
- abschließende Arbeiten, wie beispielsweise die Generierung der Maschinendokumentation.

Das **SIMOTION SCOUT** Engineering System bietet eine einheitliche Anwendersicht und eine flexible Funktionalität.

Die Formulierung von individuellen Automatisierungsaufgaben für Produktionsmaschinen erfolgt in einer einheitlichen und durchgängigen Oberfläche.

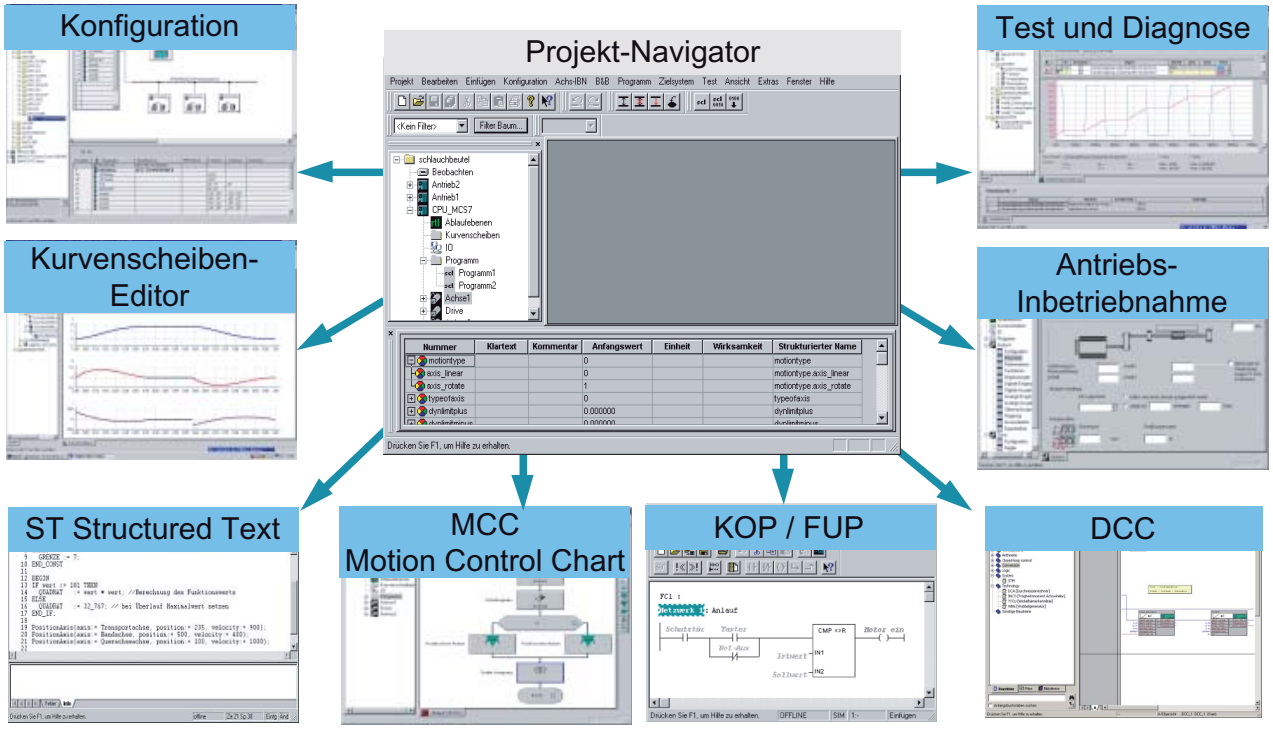


Bild 1-2 SIMOTION SCOUT Engineering System

Das **SIMOTION SCOUT** Engineering System ist als PC-Entwicklungsumgebung ein leistungsfähiges Werkzeug, welches auf einfache Art und Weise die notwendigen Engineeringsschritte optimal unterstützt. Es ist integriert in die SIMATIC- Landschaft, wobei es als Optionspaket zu **STEP7** betrieben wird. Dabei wurde besonderer Wert auf beste Usability und eine ganzheitliche, funktionsorientierte Sicht auf die Automatisierungsaufgabe gelegt.

1.1.3 SIMOTION Projekt

Das **Projekt** beinhaltet alle SIMOTION Geräte, Antriebe usw. Den jeweiligen Geräten sind hierarchisch die Technologieobjekte und Programme zugeordnet. Diese hierarchische Struktur wird im **Projektnavigator** dargestellt.

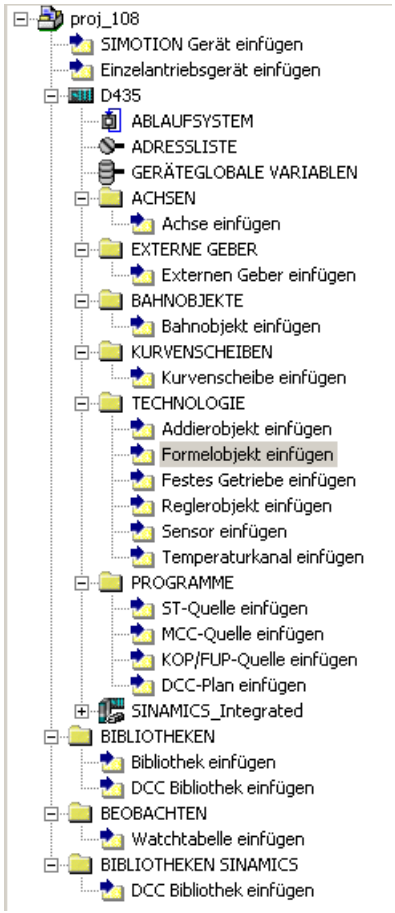


Bild 1-3 SIMOTION SCOUT Projektnavigator

Das Projekt ist die oberste Hierarchiestufe der Datenverwaltung. Alle Daten, die z. B. zu einer Produktionsmaschine gehören, legt SIMOTION SCOUT in dem zum Projekt gehörenden Verzeichnis ab.

1.1.4 Offline-/Online-Modus

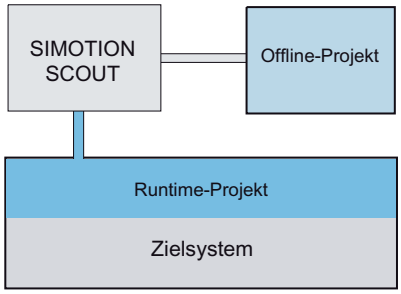


Bild 1-4 Zusammenhang zwischen Offline- und Runtime-Projekt mit SIMOTION SCOUT

SIMOTION SCOUT kann in zwei Modi betrieben werden:

- Im **Offline-Modus** können Sie Projekte anlegen und Anwenderprogramme erstellen.

  Im Offline-Modus arbeiten Sie mit dem Engineering System SCOUT ohne Verbindung zum Runtime-System. Die Daten und die Technologieobjekte werden im Engineering System gehalten.

  Der Offline-Modus ist der *führende* Arbeitsmodus, d. h. Änderungen an Systemvariablen und Konfigurationsdaten sollten in der Regel offline erfolgen.

- Im **Online-Modus** können Sie Projekte und Daten ins Zielsystem laden, die Applikation steuern und überwachen.

  Die Daten im Engineering System SCOUT bleiben als Offline-Projekt erhalten.

  Online geänderte Konfigurationsdaten können ins Offline-Projekt zurückgeladen werden, jedoch keine online geänderten Systemvariablen.

1.1.5 Programmierung

Die Steuerungs- und Motion-Control-Aufgaben werden im Anwenderprogramm vorgegeben.

Folgende Programmiersprachen stehen Ihnen zur Erstellung Ihrer Anwenderprogramme zur Verfügung:

- **MCC - Motion Control Chart**

  ist eine grafische Programmiersprache zur Programmierung von Logik- und Motion-Control-Funktionen. Dabei werden MCC-Charts erstellt.

- **ST - Structured Text**

  ist eine textuelle Programmiersprache gemäß IEC 61131-3 mit der Sie eine ST-Quelle erstellen, die mehrere Programme umfassen kann.

  Das Bearbeiten einer ST-Quelle ist auch mit einem externen ST-Editor möglich.

  Mit MCC erstellte Programme können nach ST konvertiert werden, in *umgekehrter* Richtung ist dies *nicht* möglich.

- **KOP/FUP - Kontaktplan/Funktionsplan**

  sind grafische Programmiersprachen gemäß IEC 61131-3 zur Programmierung von Logik als auch Motion Control über PLCopen-Bausteine

- **DCC - Drive Control Chart**

  Ist eine grafische Programmiersprache zur Programmierung von Antriebsfunktionalität.

  Das Bearbeiten eines Drive Control Charts ist mit einem externen DCC-Editor möglich.

Detaillierte Informationen zu den Programmiersprachen erhalten Sie in den Programmierhandbüchern **SIMOTION MCC**, **SIMOTION ST** und **SIMOTION KOP/FUP**

Die Programmierung in SIMOTION bietet folgende Vorteile:

- In einem Projekt sind Programme verschiedener Programmiersprachen einsetzbar (**MCC**, **ST** und **KOP/FUP**).
- Die Programmierung ist unabhängig von der Hardware-Plattform.
- PLC, Motion Control und Technologie können in einem Programm realisiert werden.
- Funktionen für direkten Zugriff auf Antriebsparameter über PROFIDRIVE sind verfügbar.

Modulare Maschinen werden unterstützt durch:

- Modulare Softwareentwicklung mit Bibliotheken
- Aufteilung in einzelne Maschinenmodule
- Aktivieren/Deaktivieren von DP-Slaves und Technologieobjekten
- Befehle für die Synchronisation

1.1.6 Ablaufsystem

SIMOTION bietet ein Ablaufsystem mit einer zyklischen Task, sequentiellen, zeitgesteuerten, taktsynchronen und eventgetriggerten Tasks. Anwenderprogramme können in jede Task "eingehängt" werden, auch mehrere Programme in eine Task.

Jeder Baustein kann die komplette SIMOTION-Funktionalität, d.h. Logik-, Motion- und Technologie-Funktionen nutzen.

Aufruf, Koordination / Synchronisation und Überwachung der Programme müssen nicht programmiert werden, sondern können komplett an das System delegiert werden!

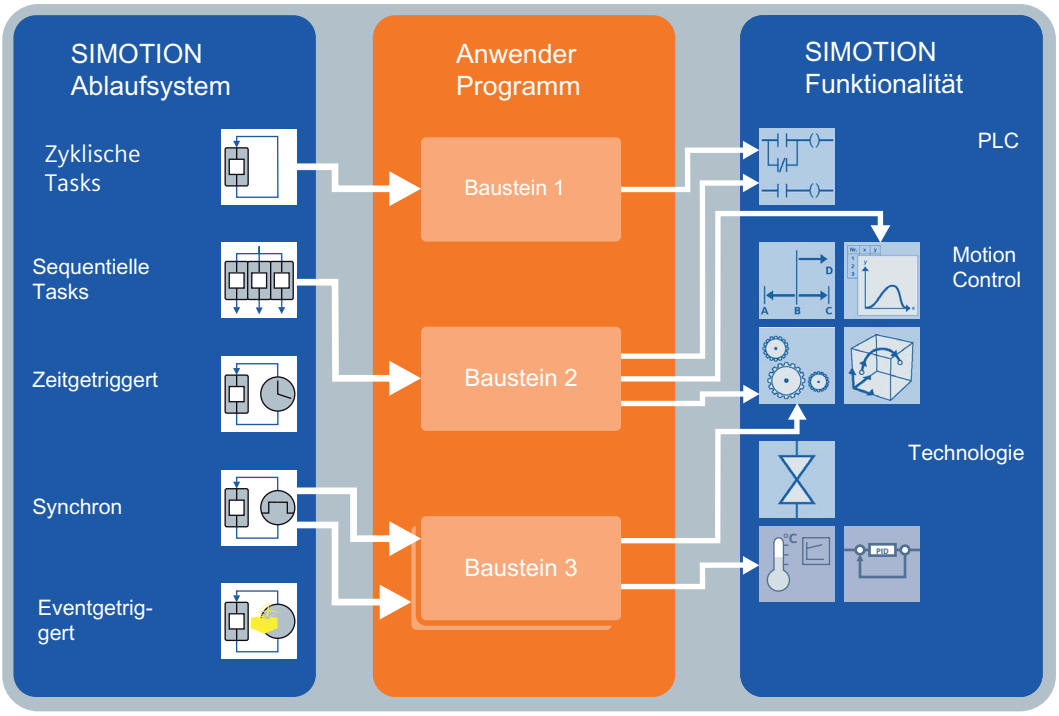


Bild 1-5 Ablaufsystem SIMOTION

1.2 SIMOTION Motion Control

SIMOTION bietet eine optimierte Systemplattform für Automatisierungs- und Antriebslösungen, bei denen im Vordergrund Motion Control - Applikationen und Technologieaufgaben stehen.

SIMOTION ist die Antwort auf die neuesten Trends bei Produktionsmaschinen:

- Mechatronik

  Mechatronik bedeutet die gesamtheitliche Betrachtung von Mechanik und Elektrotechnik in einer Maschine. Ein Teilaspekt von Mechatronik ist es, mechanische und damit relativ unflexible Komponenten (Kurvenscheiben, Getriebe, Kupplungen, Königswellen, usw.) durch intelligente Softwarelösungen zu ersetzen.

- Zusammenführung PLC-Funktionalität und Motion-Control-Technologie

  Die historische Trennung zwischen reinen Automatisierungsfunktionen und Bewegungsfunktionen wird aufgehoben. Die Zusammenführung erfolgt sowohl auf der Hardware- als auch auf der Softwareseite.

- Usability

  Ein einheitliches Engineering System (SIMOTION SCOUT) gewährleistet ein durchgängiges Konfigurieren, Parametrieren und Programmieren. Sowohl Automatisierungs- als auch Motion-Control-Aufgaben werden in einer gemeinsamen Sprache programmiert.

- Standards

  Standards der PC-Welt (z. B. Windows, Ethernet) prägen mehr und mehr auch die Automation in der Industrie. Weltweit genormte Programmiersprachen vereinfachen den Kunden die Handhabung.

- Modulare Maschinenkonzepte

  Die Standardisierung macht auch vor den Maschinenkonzepten nicht Halt. So wird zunehmend versucht, das Maschinenkonzept in verschiedene Teilkomponenten aufzuteilen. Durch dieses Konzept lassen sich dann auch einzelne Teilkomponenten standardisieren und in verschiedenen Maschinentypen *serienmäßig* einsetzen.

1.3 Einsatzgebiete

Entscheidend für ein leistungsfähiges Steuerungssystem ist, dass alle heutigen Aufgaben und zukünftige Trends im Bereich Produktionsmaschinen durch offene Steuerungskonzepte ermöglicht werden.

SIMOTION ist ein durchgängiges **Motion Control System** mit dem Schwerpunkt Automatisierung von Produktionsmaschinen. Die Durchgängigkeit bezieht sich dabei auf Engineering, Programmierung, Kommunikation, Datenhaltung, Bedienen & Beobachten (B&B) und besteht somit für das gesamte System - auch auf verschiedenen Hardware-Plattformen.

Die beschriebenen Trends stehen in erster Linie im Zusammenhang mit den nachfolgend aufgezeigten Maschinenbausegmenten. Diese Branchen bilden somit das Einsatzgebiet für unser Motion-Control-System:

- Verpackungsmaschinen
- Kunststoffmaschinen
- Umformtechnik
- Textilmaschinen
- Druckmaschinen
- Holz, Glas, Keramik
- und weitere

Die Automatisierungslösungen in diesen Branchen erfordern neben den üblichen Logik- und Antriebsaufgaben in erhöhtem Maße auch die integrierte und durchgängige Einbeziehung von Motion Control und Technologieaufgaben.

Der Systembaukasten SIMOTION besteht aus dem Engineering System SCOUT und einem gemeinsamen Runtime-System für verschiedene Hardware-Plattformen. Das Engineering System SCOUT ist dabei für alle Hardwareplattformen gleich, wobei das Konfigurieren, Parametrieren und Programmieren grafisch oder textuell erfolgt. Dies bildet gleichzeitig die Basis für branchenspezifische Lösungen.

Welche Runtime-Software (Positionieren, Gleichlauf, ...) zusätzlich zur Basisfunktionalität geladen wird, bestimmen Sie aufgrund Ihrer Applikation selbst. Durch die Ablauffähigkeit auf den verschiedenen Hardware-Plattformen bietet SIMOTION flexible Lösungen - für jede Anforderung!

1.4 Zusammenführung von PLC und Motion Control

SIMOTION führt Steuerung, Technologie und Motion Control zusammen. Damit entfallen Hardware- und Software-Schnittstellen zwischen den Steuerungen.

Bild 1-6 Zusammenführung von PLC und Motion Control

Auf der Hardwareseite bedeutet dies, dass das Automatisierungsgerät in der Lage ist, Bewegungsfunktionen zu verarbeiten. Die Hardwareplattform kann individuell ausgewählt und Hardwarekomponenten reduziert werden.

Auf der Softwareseite bedeutet die Zusammenführung von Automatisierungs- und Bewegungsfunktionen eine wesentliche Vereinfachung für das Engineering. Dies beginnt bereits bei der Projektierung und findet seine Fortsetzung in der Parametrierung und Programmierung.

Durch die Verschmelzung entfallen auch Verzögerungszeiten, was zu schnellen Reaktionen und höheren Taktzahlen im Vergleich zu Systemen mit getrennten Tasks/Prozessoren für Motion Control und PLC führt. Hinzu kommen genau bestimmbare Reaktionszeiten, die eine hohe Reproduzierbarkeit des Maschinenverhaltens und damit der Produkte gewährleisten, was sich unmittelbar auf die Produktqualität auswirkt.

Wesentlich ist ferner die Durchgängigkeit zur SIMATIC, da in einer Anlage beide Systeme genutzt werden können.

1.5 Totally Integrated Automation

Die dreifache Durchgängigkeit in Programmierung und Projektierung, in der Datenhaltung und in der Kommunikation ist der Kern von Totally Integrated Automation. Damit kommen bei jeder mit SIMATIC gelösten Automatisierungsaufgabe alle Vorteile von Totally Integrated Automation voll zum Tragen:

- Deutliche Reduzierung der Engineeringkosten
- Keine Systembrüche innerhalb der Automatisierungslandschaft
- Eine Softwarebasis für alle Komponenten

Dabei ist es unerheblich, welche Anlagen Sie automatisieren, ob Sie individuelle Lösungen im Maschinen- oder Anlagenbau realisieren oder kleine Automatisierungsaufgaben realisieren müssen. Totally Integrated Automation mit SIMATIC enthält alle Techniken, also z. B. SPS, PC-based Control, Automatisierungsrechner, dezentrale Peripherie, Bedien- und Beobachtungssysteme, Kommunikationsnetze oder Prozessleitsysteme, die Sie in Ihrer Automatisierungslandschaft benötigen. Ihre Modularität ermöglicht es, aus einem vollständigen und durchgängigen Systembaukasten genau die Lösung zu realisieren, die technisch notwendig und wirtschaftlich sinnvoll ist.

Die Durchgängigkeit von SIMOTION zur SIMATIC im Sinne von Totally Integrated Automation ist gewährleistet. Dies wird einerseits ermöglicht durch die Integration des SIMOTION SCOUT im SIMATIC-Manager und andererseits durch dieselbe Engineeringphilosophie bei vergleichbaren Tätigkeiten. Für Engineeringprozesse, die entweder in SIMATIC nicht vorhanden (Motion Control, Nockenschaltwerk, etc.) oder bei SIMOTION aufgrund der Anforderungen an ein verteiltes Systems notwendig sind, wird ein durch optimale Usability geprägter Weg gewählt. Außerdem ist SIMOTION in PROFINET integriert und enthält in diesem Sinne auch die dafür notwendigen Systemmerkmale.

1.6 Hardwareplattformen

SIMOTION unterstützt verschiedene Hardwareplattformen. Die Entscheidung für die eingesetzte Hardware-Komponente ist im Wesentlichen abhängig von den Anforderungen. Dabei können Sie Ihre Automatisierungsaufgabe auch auf unterschiedliche Zielsysteme aufteilen.

Bild 1-7 SIMOTION Hardwareplattformen

Hierfür stehen Ihnen folgende Plattformen zur Verfügung:

- PC-based (**SIMOTION P**)

  Als PC-basiertes Motion Control System arbeitet die SIMOTION P mit dem Betriebssystem Windows XP, ausgestattet mit einer Echtzeiterweiterung für SIMOTION. Neben SIMOTION-Applikationen können auch andere PC- Anwendungen gestartet werden.

  SIMOTION P eignet sich für:

  – Anwendungen, die eine Offenheit zur PC-Welt erfordern
  – Anwendungen, die Steuerung und Visualisierung auf einer Hardware verlangen
  – Umfangreiche Datenhaltung, -auswertung und -protokollierung

- Controller-based (**SIMOTION C**)

  Dieser Controller in SIMATIC S7-300-Aufbautechnik verfügt über integrierte analoge Antriebsschnittstellen und mehreren Digital-Ein- und Ausgänge.

  Zudem lässt er sich mit Peripheriebaugruppen aus dem SIMATIC S7-300- Spektrum erweitern. Zwei PROFIBUS-Anschlüsse mit PROFIdrive-Schnittstelle und eine Industrial-Ethernet-Verbindung ermöglichen die Kommunikation mit anderen Maschinenteilen.

  Der PROFIBUS kann auch für die Kommunikation mit Bediengeräten - etwa aus dem SIMATIC HMI Spektrum - oder mit übergeordneten Steuerungen wie SIMATIC S7 genutzt werden.

  Als Bediensysteme eignen sich sowohl die Panels von SIMATIC HMI als auch PCs mit WinCC flexible oder OPC-Schnittstellen.

  SIMOTION C ermöglicht:

  – Freiheit bei der Wahl der Antriebe
  – breites Spektrum an Prozesssignalen
  – Retrofit-Anwendungen durch integrierte Analog-Schnittstellen
  – Direkte Ankopplung von Analog- und Schrittantrieben

- Drive-based (**SIMOTION D**)

  Bei SIMOTION D ist die SIMOTION Funktionalität direkt in die Regelungsbaugruppe des Antriebssystems SINAMICS S120 integriert. Dadurch wird das Gesamtsystem aus Steuerung und Antrieb sehr kompakt und besonders reaktionsschnell. SIMOTION D ist in zwei Aufbauformen als Einachssystem SIMOTION D410 und als Mehrachssystem SIMOTION D4x5/D4x5-2 in vier Performancevarianten erhältlich, womit ein Höchstmaß an Skalierbarkeit und Flexibilität gewährleistet ist. Der Anwendungsbereich erstreckt sich von einzelnen Achsen bis hin zu hochperformanten Vielachsmaschinen.

Technologiepakete und Technologieobjekte 2

2.1 Einführung

SIMOTION Basisfunktionalität

Ein SIMOTION-System ohne Technologiepaket, ohne Technologieobjekte, stellt das Ablaufsystem und die Basisfunktionalität eines Steuerungssystems nach IEC 61131-3 bereit.

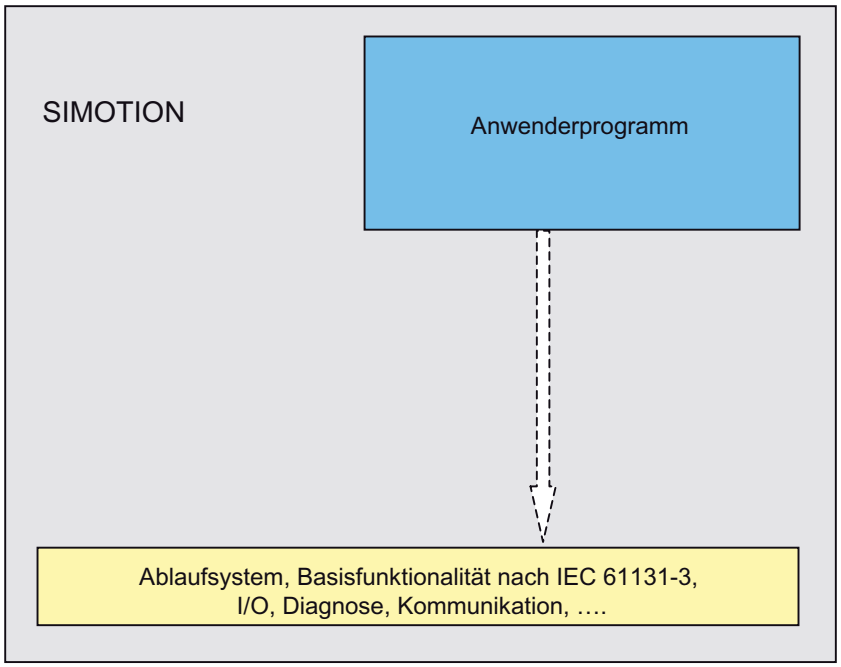


Bild 2-1 SIMOTION Basisfunktionalität

Die Anwenderprogramme können wahlfrei den verschiedenen Tasks zugeordnet werden.

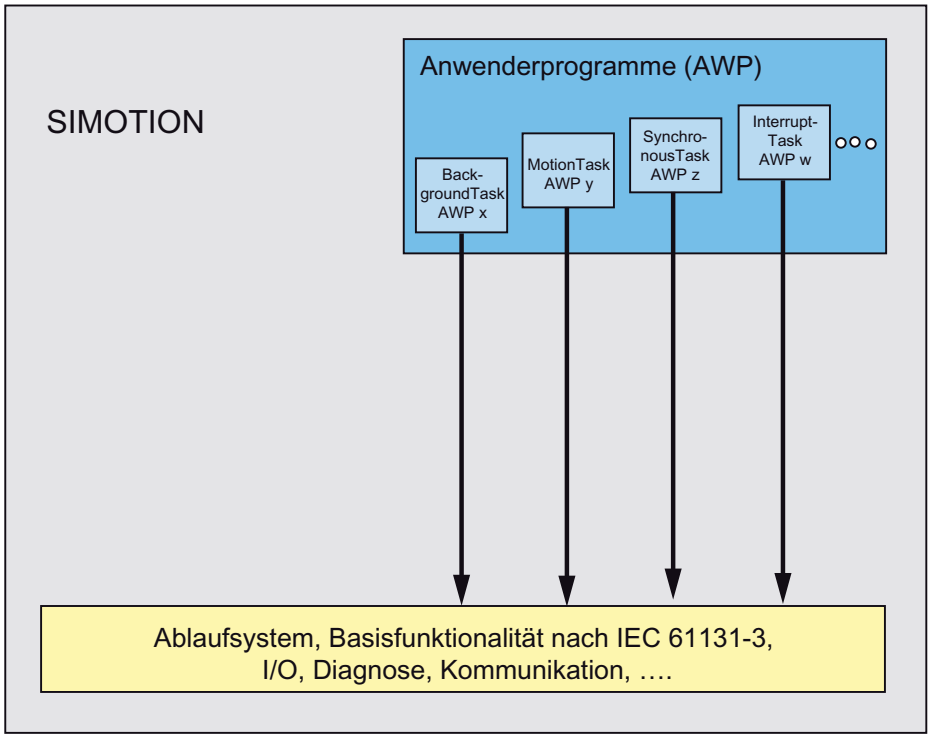


Bild 2-2 Zuordnung der Anwenderprogramme zu Tasks

Technologieobjekte

Das Runtime-System von SIMOTION ist nicht funktionsorientiert, sondern objektorientiert realisiert, d.h. es gibt selbständige, sogenannte Technologieobjekte (TO). Diese Technologieobjekte stehen für Technologie und Motion Control zur Verfügung. Technologieobjekte haben eine hohe Funktionalität integriert, z. B. enthält ein TO Achse die Kommunikation zum Antrieb, die Messwertverarbeitung, Lageregelung und Positionierfunktionalität.

Die TOs werden per Konfiguration angelegt und parametriert und laufen dann automatisch im Kern des Systems. Im Anwenderprogramm wird diese Funktionalität über entsprechende Befehle aufgerufen. Die TOs arbeiten die Aufträge dann selbständig ab und stellen entsprechende Statusmeldungen zur Verfügung (vergleichbar mit einem Treiber).

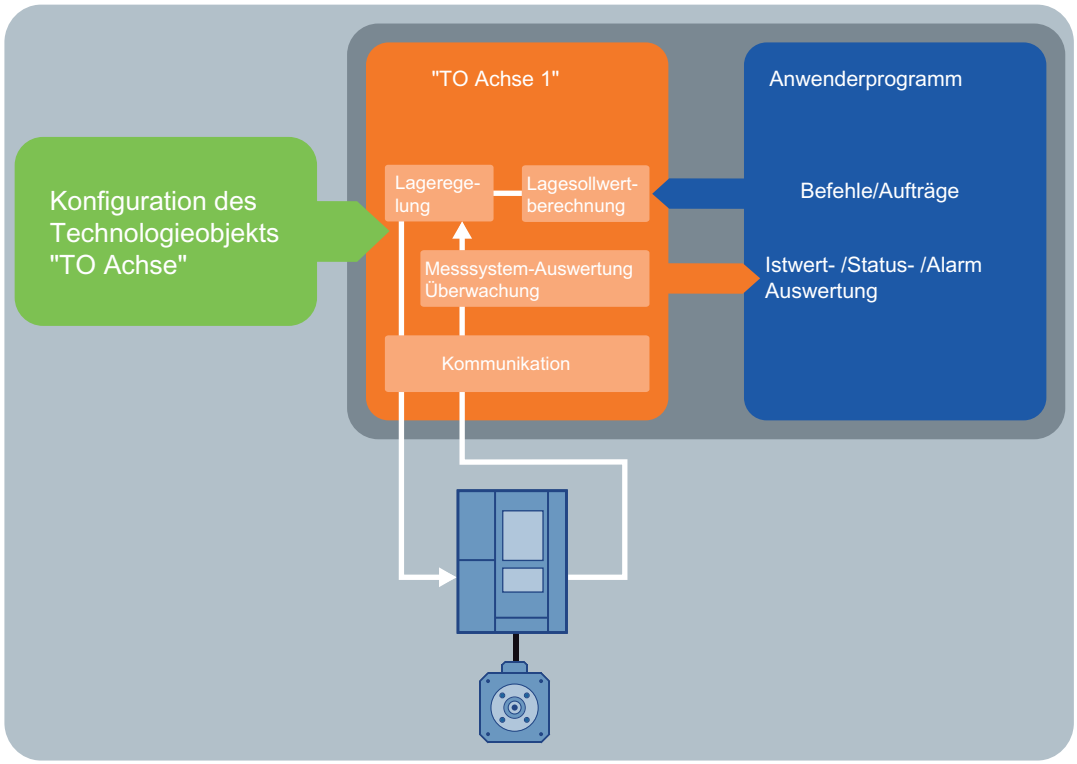


Bild 2-3 Technologieobjekte SIMOTION

Die Anzahl der Objekte ist prinzipiell unbegrenzt, d.h. es sind beliebig viele Achsen, Gleichlaufbeziehungen, Nocken/-spuren an einer Achse, Externe Geber als Leitwerte, etc. möglich. Die TOs können auch aus dem Programm aktiviert oder deaktiviert werden. Dies ermöglicht eine einfache Anpassung an verschiedene Maschinenkonfigurationen (z.B. wenn eine Achse in einer bestimmten Konfiguration nicht vorhanden ist)

Die Technologieobjekte bieten dem Anwender eine technologische Sicht auf Aktoren und Sensoren, und stellen technologische Funktionen für diese zur Verfügung, z. B.:

- das Technologieobjekt Achse auf Antrieb und Geber
- das Technologieobjekt Externer Geber nur auf einen Geber
- das Technologieobjekt Nocken/Nockenspur auf einen definiert zu schaltenden Ausgang
- das Technologieobjekt Messtaster auf einen Messeingang

Daneben sind Technologieobjekte zur Aufbereitung von technologischen Daten auf Systemebene verfügbar, z. B.

- das Technologieobjekt Gleichlauf für den Gleichlauf zwischen zwei Achsen oder einer Achse auf einen Geberwert
- das Technologieobjekt Bahn zum Verfahren von Bahnachsen entlang einer Bahn und einer Positionierachse synchron zur Bahn
- das Technologieobjekt Kurvenscheibe zur Repräsentation von komplexen, programmierbaren Funktionen
- die Technologieobjekte Addierobjekt, Formelobjekt zur systemseitigen Bearbeitung von Bewegungsdaten und Technologiedaten

Die Technologieobjekte werden vom System in Systemtasks ausgeführt, z. B. der IPO-Task, der IPO_2-Task, oder der Servo-Task.

Instanziierung

Technologieobjekte werden vom System als **Technologieobjekttypen** bereitgestellt, die über Instanziierung an den konkreten Einsatzfall angepasst wird.

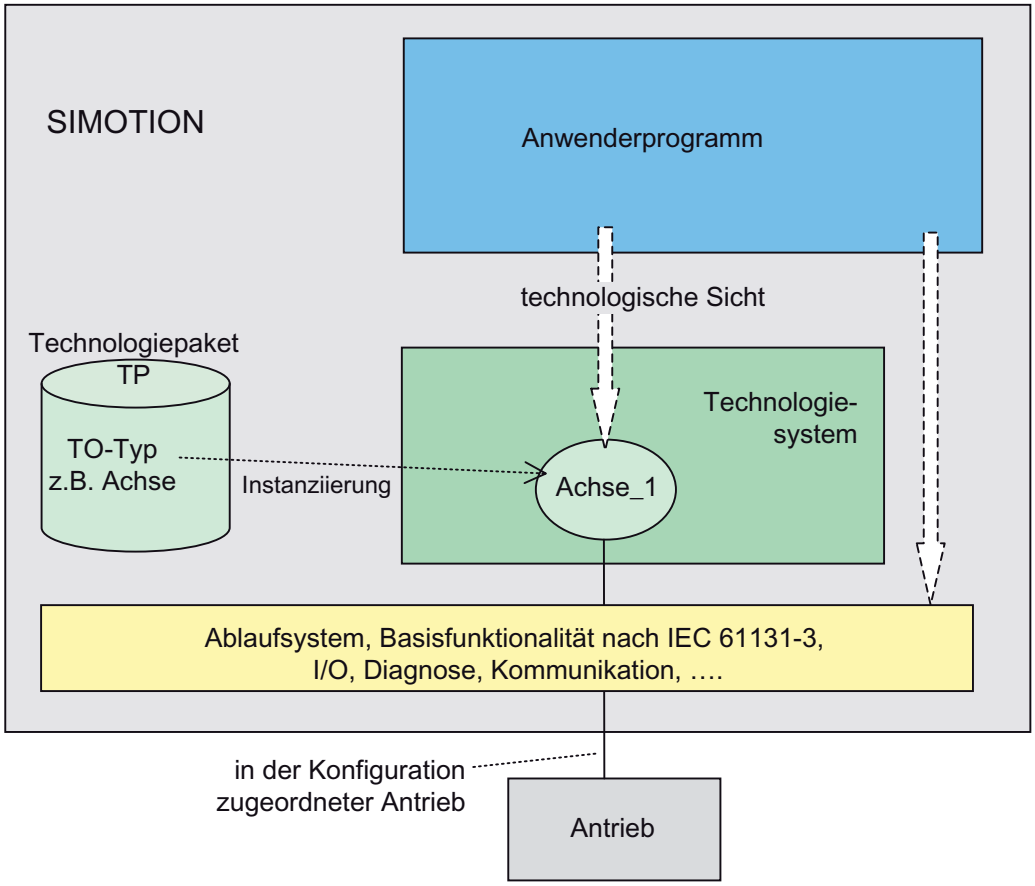


Bild 2-4 Programmiermodell auf Technologieobjekt-Instanzen, z. B. auf TO Achse

Technologieobjekttypen werden in einem Technologiepaket zusammengefasst und auf das Runtime-System geladen.

2.2 Technologiepakete

Die Technologieobjekttypen werden bei SIMOTION über **Technologiepakete** bereit gestellt. Sie werden mit dem Projekt auf das SIMOTION Runtime-System geladen.

Folgende Technologiepakete sind verfügbar:

- **TP CAM** enthält die grundlegenden Technologien für Motion Control, wie z. B. Drehzahlachse, Positionierachse, Gleichlaufachse, Gleichlaufobjekt, Kurvenscheibe, Nocken, Nockenspur, Messtaster
- **TP Path** enthält zusätzlich die Technologie Bahn
- **TP CAM_ext** enthält zusätzliche Objekte zur Aufbereitung technologischer Daten auf Systemebene, z. B. Addierobjekt, Formelobjekt
- **DCC** enthält verschaltbare Bausteine für antriebsnahe Regelfunktionen.
- **TControl** enthält die Technologie Temperaturregler

Schalenmodell der Technologiepakete

Der Funktionsumfang der Technologiepakete lässt sich mit einem Schalenmodell grafisch darstellen.

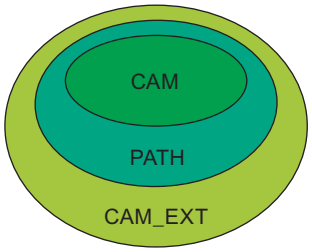


Bild 2-5 Schalenmodell der Technologiepakete

Weitere Technologiepakete

Zusätzlich stehen weitere branchenspezifische Technologiepakete als separate Produkte zur Verfügung.

2.3 Technologieobjekte (TO)

Technologieobjekte / Technologieobjekttypen stellen Funktionalität für Technologie und Motion Control bereit, beinhalten damit technologische Systemfunktionen und verdecken die konkrete Hardware-Anschaltung.

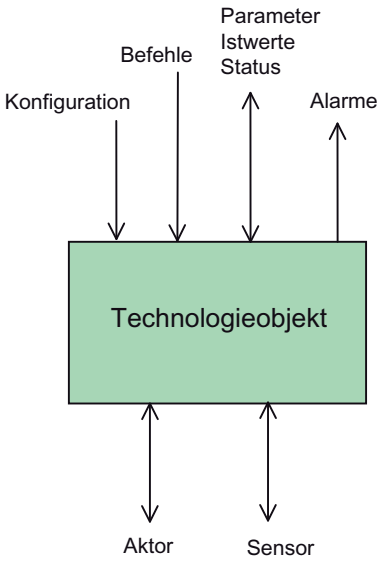


Bild 2-6 Mögliche Schnittstellen zu Technologieobjekten

Für die TO-übergreifende Verarbeitung von technologischen Daten auf Systemebene stellen die Technologieobjekte definierte Eingangs- und Ausgangsinterfaces bereit.

Siehe auch

Verfügbare Technologieobjekte (Seite 43)

Verschaltungen (Seite 39)

2.3.1 Instanziierung und Konfiguration

Über **<Technologieobjekt> einfügen** erzeugen Sie eine Instanz des entsprechenden TO-Typs in **SIMOTION SCOUT**. (Daten, Parameter, Alarmlisten etc. werden im System angelegt.)

Gegebenenfalls muss die Ausprägung des Technologieobjekts angegeben werden, z. B. bei der Achse: Drehzahlachse, Positionierachse, Gleichlaufachse.

Die Hardware/Software-Schnittstellen zum Aktor/Sensor werden festgelegt, z. B. Hardware-Adressen, Datenbreite, Telegramm-Typ bei PROFIdrive-Telegrammen.

Grundlegende Einstellungen werden festgelegt, wie z. B. Bearbeitung der technologischen Systemfunktionen im IPO- oder IPO_2-Takt, siehe Tasklaufzeiten (Seite 251) .

Für die **Konfiguration** der Technologieobjekte stehen Assistenten und Dialoge in SIMOTION SCOUT zur Verfügung. Dabei legen Sie die grundsätzliche Funktionalität eines Technologieobjekts über die Konfigurationsdaten fest.

Alle Eingaben in den Konfigurations- und Parametriermasken sind in Konfigurationsdaten und Systemvariablen abgebildet, die in den Masken als Tooltipps angezeigt werden.

Eine detaillierte Beschreibung der Konfigurationsdaten und Systemvariablen finden Sie in den SIMOTION Referenzlisten.

Die Beschreibung zum Vorgehen bei Anlegen und Konfiguration der einzelnen Technologieobjekte finden Sie in den betreffenden **Funktionshandbüchern**.

Ein Großteil der Konfigurationsdaten kann zur Laufzeit über das Anwenderprogramm oder über die Expertenliste geändert werden.

Siehe auch

Programmierung allgemeiner Standardfunktionen - Überblick (Seite 361)

Systemvariablen (Seite 135)

Konfigurationsdaten (Seite 139)

2.3.2 Parametrierung

Systemvariablen beinhalten technologische Parameter und Anzeigewerte der Technologieobjekte.

Für die Einstellung von technologischen Parametern und Standardwerten/Vorbelegung sind in der Regel Masken in SIMOTION SCOUT verfügbar.

Systemvariablen sind Einzelwerte oder Strukturen, die konsistent ausgelesen werden.

Weitere Informationen zu Systemvariablen finden Sie in Systemvariablen (Seite 135).

2.3.3 Programmierung

Die Technologiefunktionalitäten werden über spezifische Befehle aktiviert/deaktiviert.

Synchrone/asynchrone Befehlsbearbeitung

Die Befehle können *synchron* oder *asynchron* eingestellt werden.

Bei synchroner Einstellung von Bewegungsbefehlen kann am Befehl auf einen bestimmten Bewegungsstatus oder bis auf das Ende der Bewegung, z. B. einer Positionierung gewartet werden. Diese Einstellung ist besonders vorteilhaft bei der Programmierung von Bewegungsabläufen in sequentiellen Tasks (MotionTasks).

Rückgabewert

Der Rückgabewert liefert das Ergebnis des Funktionsaufrufs, z. B. Funktion wurde/wird wie geplant ausgeführt, oder es liegt ein Fehler vor.

CommandId

Über die **CommandId** ist ein TO-Befehl eindeutig identifizierbar und nachverfolgbar.

Informationen über Parameter, Weiterschaltbedingungen, Diagnose etc. finden Sie unter:

- Funktionen für CommandID (Seite 438) und diversen Beispielen in diesem Handbuch.
- **SIMOTION MCC Motion Control Chart**, MCC-Chart programmieren

Eine detaillierte Beschreibung der TO-Befehle finden Sie in den SIMOTION Referenzlisten.

Programmiermodell

Die Befehle werden Tasks zugeordnet, welche wiederum den Ablaufebenen des Tasksystems zugeordnet werden.

Befehle können aus allen Anwenderprogramm-Tasks des Systems abgesetzt werden.

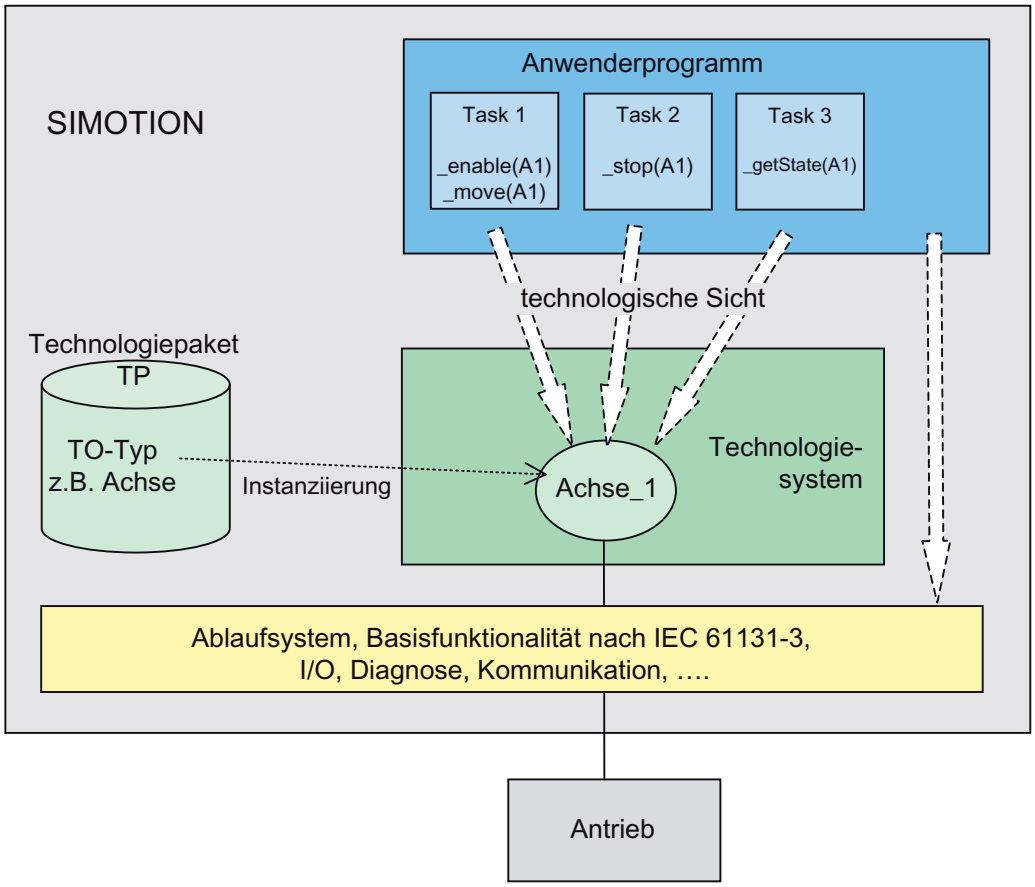


Bild 2-7 Programmiermodell auf Technologieobjekt-Instanzen, z. B. auf TO Achse

Für die Wirksamkeit eines Befehls an einem TO ist nur der Zeitpunkt entscheidend, zu dem er am TO bearbeitet wird.

Werden Befehle aus mehreren Tasks abgesetzt, muss die konsistente Programmierung durch das Anwenderprogramm sichergestellt werden.

Befehlsausführung am TO / Wirksamkeit von TO-Befehlen

Die aus dem Anwenderprogramm, den Anwenderprogramm-Tasks auf die TO abgesetzten Befehle können prinzipiell eingeteilt werden in:

- Befehle die unmittelbar im Kontext/Ablauf der Anwenderprogramm-Tasks ausgeführt werden

  Diese werden wie eine Funktion innerhalb der Anwenderprogramm-Tasks abgehandelt.

  Diese Befehle sind synchron, da das Anwenderprogramm erst mit der Rückgabe des Funktionsergebnisses fortgeführt wird.

  Beispiel: TO-Werte lesen
- Befehle, die in einen Befehlspuffer eingetragen werden und sich gegenseitig überschreiben
- Befehle, die in einen Befehlspuffer eingetragen werden und bei belegtem Befehlspuffer abgewiesen werden

An den Befehlen der TO geben Sie an, wie sich die Befehle am TO verhalten, wenn zwischen zwei Bearbeitungstakten mehr als ein Befehl auf einer Befehlsgruppe abgesetzt wird, z. B. Ersetzen/Überschreiben oder Befehlsabweisung.

Die Befehlspuffer und das Wirksamwerden der Befehle finden Sie in den **Funktionshandbüchern** der einzelnen Technologieobjekte.

Ablaufeigenschaften

Die Ablaufeigenschaften von Technologieobjekten sind objektspezifisch.

Für die Bearbeitung der TO sind die synchronen Ablaufebenen DP-Takt, Servo-Takt und IPO-Takt bzw. IPO_2-Takt vorgesehen (mit Ausnahme von Temperaturreglern).

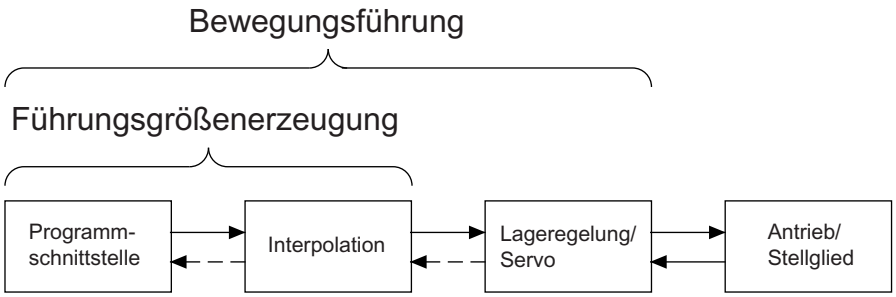


Bild 2-8 Funktionsschema

Die Instanzen der Technologieobjekte werden in verschiedenen Ablaufebenen bearbeitet.

- Die Befehlsauswertung und Bewegungsführung erfolgt im IPO/IPO_2 -Takt.
- Die Lageregelung bzw. Sollwertführung erfolgt im Servo-Takt .
- Die Kommunikation mit dem Antrieb erfolgt über PROFIBUS DP im DP-Takt bzw. über PROFINET IO mit IRT im PN-Takt.

Die Bearbeitung kann durch die System- und Objektkonfiguration beeinflusst werden.

- Die Einstellung der Systemtakte
  Durch die Einstellung der Systemtakte wird die Abtastzeit für die Bewegungsführung von Achsen (IPO, IPO_2), die Lagerregelung (Servo) sowie die Kommunikation über PROFIBUS DP bzw. PROFINET IO mit IRT eingestellt.
- Bei der Konfiguration der Objekte können Sie festlegen, ob die Bewegungsführung im IPO-Takt oder im IPO_2-Takt erfolgen soll. Hierdurch können Sie zwischen zeitkritischen und unkritischeren Vorgängen unterscheiden.
- Die Steuerung der Technologieobjekte erfolgt aus dem Anwenderprogramm durch den Aufruf von Systemfunktionen.

Ab V4.2 kann optional in einem zweiten Applikationstakt auch der Servo_fast und IPO_fast verwendet werden.

Alarme

Ein Technologieobjekt überwacht die Ausführung und Ausführbarkeit von technologischen Funktionen, wie auch die für das TO benötigte Peripherie, und erzeugt gegebenenfalls einen **Technologischen Alarm**.

Ein **Technologischer Alarm** besitzt eine TO-lokale Alarmreaktion, z. B. Bewegung anhalten, und eine globale Reaktion, z.B. System anhalten, oder Alarmtask aufrufen, in der dann weitere Reaktionen angegeben werden können. Die lokalen und globalen Reaktionen sind alarmspezifisch einstellbar.

Eine detaillierte Beschreibung der TO-Alarme finden Sie in den SIMOTION Referenzlisten.

Siehe auch

Das Ablaufsystem (Seite 189)

Unterschiede zwischen zyklischer und sequentieller Programmierung (Seite 117)

Technologische Alarme (Seite 169)

2.3.4 Verschaltungen

Die Technologieobjekte können flexibel verschaltet werden (z.B. Gleichlaufachsen können umgeschaltet oder verkettet werden), und auch auf mehrere SIMOTION-Geräte verteilt werden. Die Funktionalität steckt hierfür im Objekt und muss nicht programmiert werden. Dadurch ergibt sich eine höchste Flexibilität bzgl. Funktionalität und Verteilbarkeit.

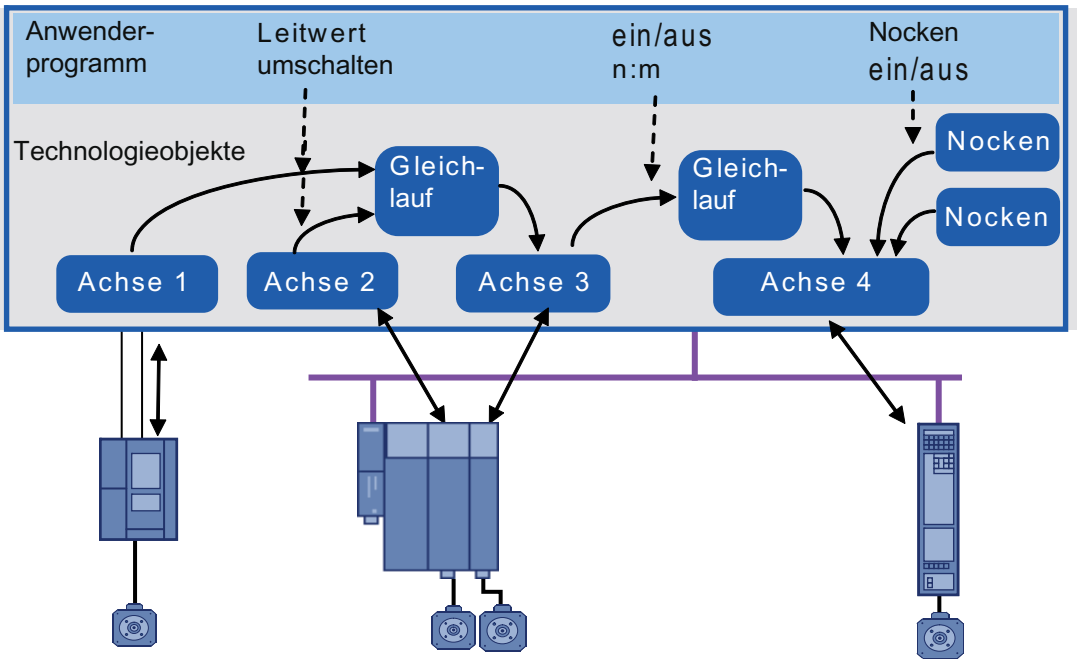


Bild 2-9 Verschaltungsbeispiel

Für den Austausch von Daten/Informationen zwischen den Technologieobjekten auf Systemebene sind Verschaltungsinterfaces an den TO definiert, die in der Regel einen bidirektionalen Datenaustausch zwischen einzelnen Technologieobjekten ermöglichen.

Die Ausprägung der Außenschnittstellen kann an jedem TO-Typ unterschiedlich sein. Verschaltungen oder Aktoren / Sensoren müssen nicht vorhanden sein.

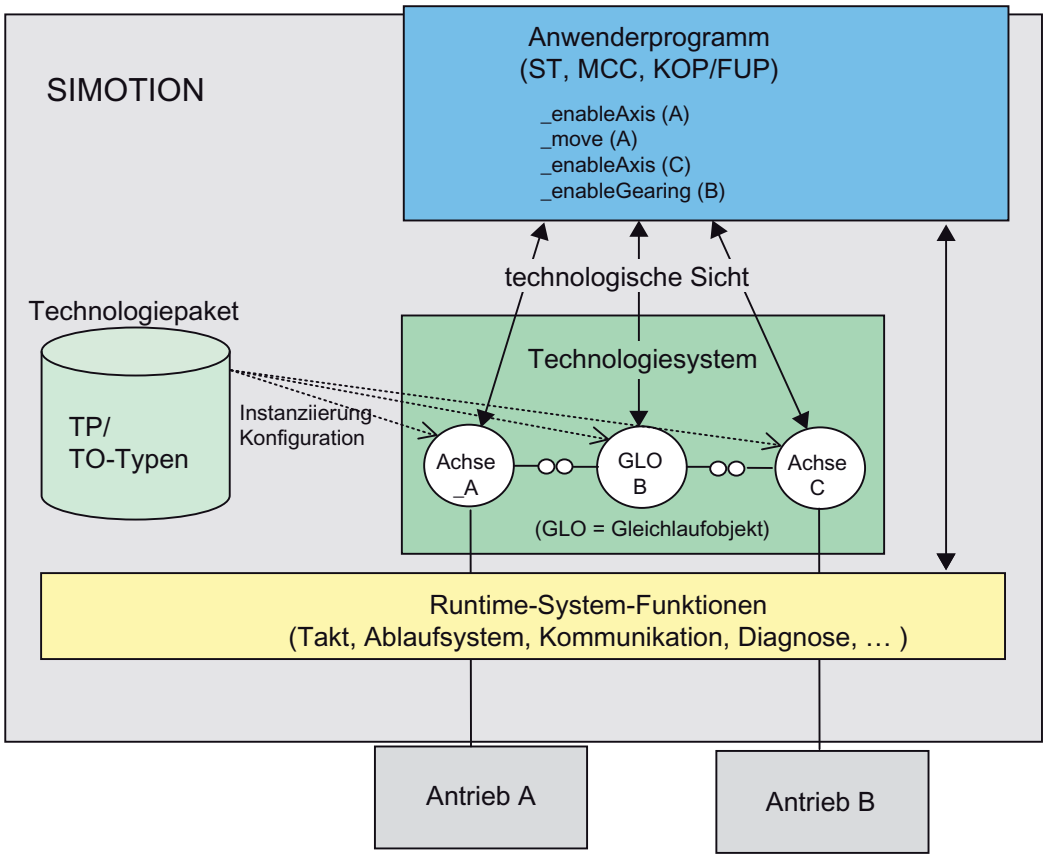


Bild 2-10 Verschaltung zwischen den Technologieobjekten
Beispiel: Leitachse - Gleichlaufobjekt - Folgeachse

Verschaltungsinterfaces können abhängig vom Interfacetyp im zyklischen Betrieb (zyklische Bewegungsdaten, z. B. s,v,a beim Bewegungsvektor vom Typ Motion) und bei der Initialisierung (z. B. Modulo-Information, Einheiten) unterschiedliche Daten austauschen.

Implizite Verschaltung

Ist eine Verschaltung zwingend und eindeutig, wird diese vom Engineering System SIMOTION SCOUT implizit mit angelegt, z. B. Messtaster mit Achse, Nocken mit Achse, Gleichlauf mit Achse. Die entsprechenden TO-Typen werden unter der Achse mit angeboten.

Verschaltung über technologische Verschaltungsmasken

Für weitere Verschaltungen werden spezifische Masken angeboten z. B. für:

- Gleichlaufachse mit Leitachse/Leitwert
- Gleichlaufobjekt mit Kurvenscheiben
- Profileingänge mit Kurvenscheiben

Verschaltung über allgemeine Verschaltungsmaske (nur für Experten)

Für besondere Verschaltungen wird eine allgemeine Verschaltungsmaske angeboten, z. B. für

- Verschaltung MotionIn-Interface (für Bewegungsvektoren)
- Verschaltung Momenteneingangs-Interface

Siehe auch

Verschaltungsinterfacetyp 'Motion' (für Experten) (Seite 63)

Verschaltungsinterfacetyp 'LREAL' (für Experten) (Seite 65)

Verschaltung von Technologieobjekten (Seite 55)

2.3.5 Technologieobjekte und DCC

Beschreibung

Der Austausch der Daten zwischen DCC und TO erfolgt seitens der DCC-Pläne

- über die direkte Verschaltung von Bausteineingängen und Bausteinausgängen mit Systemvariablen der TO;

  bei TO Achse besteht die Möglichkeit, in Systemvariablen vorgegebene Daten, wie Override und Sollwerte zyklisch zu übernehmen, ohne dass hierfür jedes Mal Befehle im Anwenderprogramm abgesetzt werden müssen.

- bzw. über die Weitergabe von in DCC berechneten Werten über Befehle und Variablen in den Anwenderprogrammen an die TO. Die TO besitzen spezifische funktionsbezogene Verschaltungsinterfaces.

Die TO selbst sind in den DCC-Plänen nicht als Bausteine verfügbar.

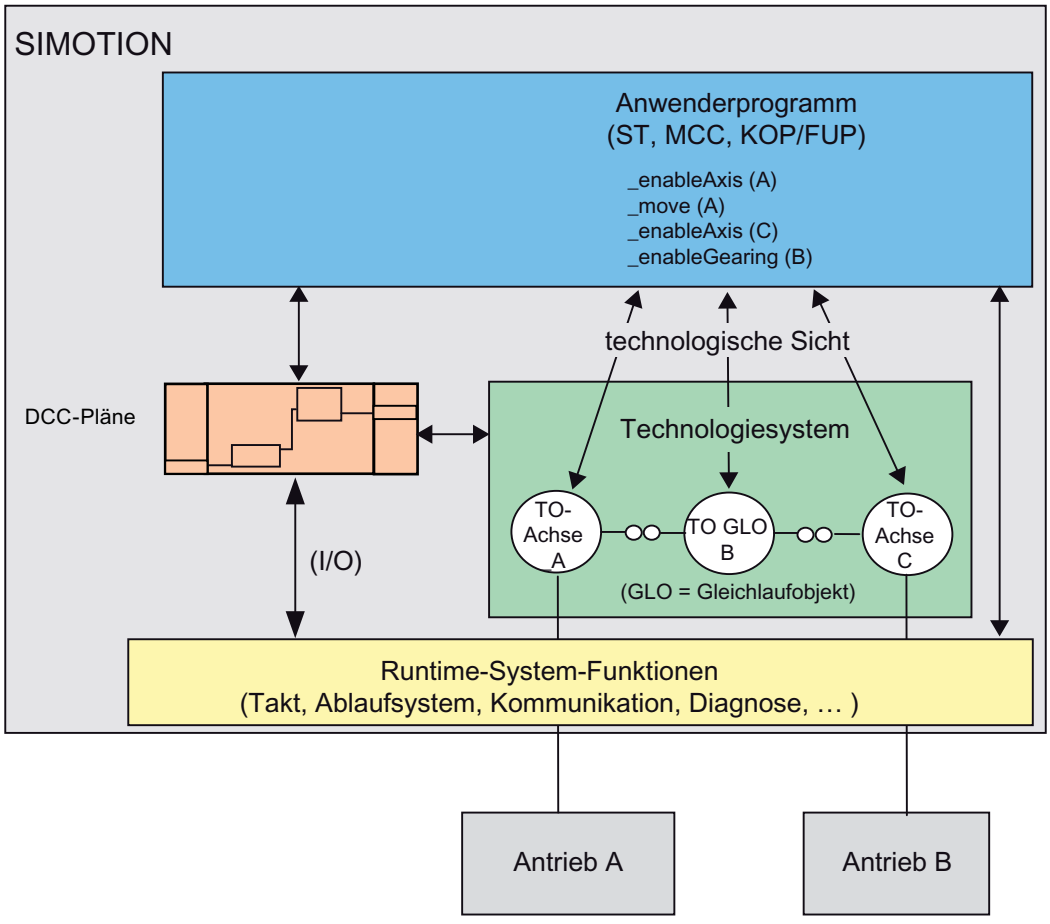


Bild 2-11 Technologie und DCC

Siehe auch

Ablaufmodell für DCC-Bausteine (DCB) (Seite 290)

2.3.6 Verfügbare Technologieobjekte

Tabelle 2- 1 Übersicht der in SIMOTION verfügbaren Technologieobjekte

Technologieobjekt	Symbol	Kurzbeschreibung
Achse		Achsen gibt es in verschiedenen Ausprägungen. Es gibt: • reale oder virtuelle Achsen • Positionierachsen oder Drehzahlachsen • elektrische Achsen oder Hydraulikachsen • Standardachsen oder kraft-/druckgeregelt • Moduloachsen • Gleichlaufachsen • Bahnachsen Ausführliche Informationen siehe Funktionshandbuch **Motion Control Technologieobjekte Achse elektrisch/hydraulisch, Externer Geber**
Gleichlaufobjekt		Wird eine Achse mit der Technologie Gleichlauf angelegt, dann wird unterhalb der Achse das Gleichlaufobjekt erzeugt. Im Gleichlaufobjekt sind die Einstellungen für den Gleichlauf hinterlegt. Ausführliche Informationen siehe Funktionshandbuch **Motion Control Technologieobjekte Gleichlauf, Kurvenscheibe**
Bahnobjekt (ab V4.1)		Mit dem Technologieobjekt Bahnobjekt können Sie Bahninterpolation projektieren. Ausführliche Informationen siehe Funktionshandbuch **Motion Control Bahninterpolation**
Messtaster		Messtaster dienen zur schnellen und genauen Erfassung von Istpositionen. Ausführliche Informationen siehe Funktionshandbuch **Motion Control Technologieobjekte für Nocken und Messtaster**
Nocken		Das Technologieobjekt Nocken erzeugt positionsabhängige Schaltsignale und kann Positionier-, Gleichlaufachsen oder externen Gebern zugeordnet werden. Ausführliche Informationen siehe Funktionshandbuch **Motion Control Technologieobjekte für Nocken und Messtaster**
Nockenspur		Nockenspuren fassen mehrere Nocken zu einem Technologieobjekt zusammen. Ausführliche Informationen siehe Funktionshandbuch **Motion Control Technologieobjekte für Nocken und Messtaster**
Externer Geber		Das Technologieobjekt Externer Geber stellt im System die Funktionalität für die Anschaltung eines externen Gebers ohne Achse bereit, z. B. Winkelgeber an einer Presse. Ausführliche Informationen siehe Funktionshandbuch **Motion Control Technologieobjekte Achse elektrisch/hydraulisch, Externer Geber**
Kurvenscheibe		Mit dem Technologieobjekt Kurvenscheibe können Sie eine Übertragungsfunktion definieren und diese mit anderen Technologieobjekten anwenden. Ausführliche Informationen siehe Funktionshandbuch **Motion Control Technologieobjekte Gleichlauf, Kurvenscheibe**
Temperaturkanal		Mit dem Technologieobjekt Temperaturkanal sind in SIMOTION Temperaturregelungen projektierbar. Ausführliche Informationen siehe Funktionshandbuch **Motion Control Ergänzende Technologieobjekte**

Technologieobjekt	Symbol	Kurzbeschreibung
Ergänzende Technologieobjekte (für Experten): Der Vorteil dieser Technologieobjekte besteht darin, dass sie, wie andere TOs auch, auf Systemebene bearbeitet werden. Die verschalteten Signale werden direkt innerhalb eines Systemtaktes in den Technologieobjekten bearbeitet. Applikative Lösungen in Structured Text bringen dagegen einen häufigen Wechsel zwischen System und Applikation und damit Totzeiten mit sich. Die ergänzenden TOs werden z. B. bei der Realisierung von Wickler-Applikationen angewendet.		
Festes Getriebe		Mit dem Technologieobjekt Festes Getriebe können Sie einen festen Gleichlauf (ohne Auf-/Absynchronisieren) auf Basis eines vorgegebenen Getriebefaktors realisieren. Ausführliche Informationen siehe Funktionshandbuch **Motion Control Ergänzende Technologieobjekte**
Addierobjekt		Mit Addierobjekten können Sie bis zu vier Eingangsvektoren zu einem Ausgangsvektor aufaddieren. Ausführliche Informationen siehe Funktionshandbuch **Motion Control Ergänzende Technologieobjekte**
Formelobjekt		Mit Formelobjekten können Sie mathematische Operationen auf Skalare (LREAL, DINT) und auf Bewegungsvektoren vom Typ Motion anwenden. Ausführliche Informationen siehe **Funktionshandbuch Ergänzende Technologieobjekte**
Sensor		Mit dem Technologieobjekt Sensor können Sie skalare Messwerte erfassen. Ausführliche Informationen siehe Funktionshandbuch **Motion Control Ergänzende Technologieobjekte**
Reglerobjekt		Mit dem Reglerobjekt können Sie skalare Größen aufbereiten und regeln. Ausführliche Informationen siehe Funktionshandbuch **Motion Control Ergänzende Technologieobjekte**

2.4 Expertenliste

Neben dem Zugang zu den Konfigurationsdaten und Systemvariablen über die Assistenten und Parametriermasken steht Ihnen auch der direkte Zugriff über die **Expertenliste** zur Verfügung.

Da alle wichtigen Konfigurationsdaten und Systemvariablen über Dialoge parametriert werden können, ist die Expertenliste jedoch nur in Ausnahmefällen erforderlich (wie z. B. Online-Änderung von Konfigurationsdaten).

Hierüber sind für Achsen, Gleichlaufobjekt und Externer Geber die wichtigsten Konfigurationsdaten und Systemvariablen für die Programmierung und Diagnose ersichtlich.

Ab V4.1 enthält die Expertenliste eine eigene Sicht auf eine Auswahl wichtiger Konfigurationsdaten und Systemvariablen (**Ausgewählte Parameter**). Hierüber haben, Sie spezifisch für die einzelnen Technologieobjekte, einen einfachen Zugang auf wichtige Konfigurationsdaten und Systemvariablen für die Programmierung und Diagnose

Ab V4.2 können keine anwenderdefinierten Werte-/Parameterlisten mehr erzeugt werden. Bereits vorhandene anwenderdefinierte Werte-/Parameterlisten können problemlos durch Öffnen in einer Watchtabelle konvertiert werden, die exakt denselben Funktionsumfang bieten.

Hinweis

Die Änderung von Parametern in der Expertenliste setzt besondere Systemkenntnisse voraus.

- Eingaben über die Expertenliste können durch den Aufruf der Assistenten und Parametriermasken überschrieben werden.
- Es erfolgt keine Überprüfung von Abhängigkeiten zu anderen Parametern.

Expertenliste aufrufen

1. Markieren Sie das gewünschte Technologieobjekt im Projektnavigator.
2. Rufen Sie im Kontextmenü des Objekts **Experte > Expertenliste** auf oder drücken Sie die Tasten **Strg+E** auf der Tastatur (wenn z.B. ein Achsdialog geöffnet ist).

   Oder
3. Doppelklicken Sie auf den Eintrag **Expertenliste** im Projektnavigator unter einem TO.

2.4.1 Expertenliste für Konfigurationsdaten und Systemvariablen

In der **Expertenliste** werden in ersten beiden Registern die **Systemvariablen** und die **Konfigurationsdaten** für das Technologieobjekt angezeigt.

Sie können die Werte von Konfigurationsdaten im OFFLINE-Modus und im ONLINE-Modus ändern. Schreibbare Parameter werden in der Farbe grün dargestellt, nicht schreibbare in der Farbe gelb. Abgeblendet angezeigte Werte können Sie nicht ändern.

Ab V4.2 können keine anwenderdefinierten Werte-/Parameterlisten mehr erzeugt werden. Bereits vorhandene anwenderdefinierte Werte-/Parameterlisten können problemlos nur durch Öffnen in Watchtabellen konvertiert werden, die exakt denselben Funktionsumfang bieten.

Tabelle 2- 2 Folgende Informationen werden in der Expertenliste angezeigt:

Schaltfläche/Tabellenspalte	Bedeutung/Hinweis
Konfigurationsdaten	Auf dem Reiter **Konfigurationsdaten** werden die Konfigurationsdaten des TO in alphabetischer Reihenfolge aufgelistet. Alle Konfigurationsdaten können grundsätzlich vom Anwender verändert werden (grüne Darstellung).
Systemvariablen	Auf dem Reiter **Systemvariablen** werden die Systemvariablen des TO in alphabetischer Reihenfolge aufgelistet. Es gibt sowohl schreibbare (grüne Darstellung), wie nicht schreibbare (gelbe Darstellung) Systemvariablen.

Schaltfläche/Tabellenspalte		Bedeutung/Hinweis
Ausgewählte Parameter		Auf dem Reiter **Ausgewählte Parameter** werden die wichtigsten Systemvariablen und Konfigurationsdaten eines TO in alphabetischer Reihenfolge angezeigt, getrennt nach Systemvariablen und Konfigurationsdaten. Der Reiter kann geschlossen werden, wird aber beim nächsten Aufruf wieder angezeigt. Die Auswahl ist eine Werkseinstellung und kann nicht geändert werden.
Liste in Watchtabelle **konvertieren** 		Klicken Sie auf diese Schaltfläche, wenn Sie eine Anwenderdefinierte Liste, die Sie mit einer früheren Version erstellt haben, in eine Watchtabelle konvertieren wollen. Vor der Konvertierung bekommen Sie eine Meldung mit näheren Informationen. Nach dem Bestätigen der Meldung können Sie dann die Liste in einem Datei-Dialog auswählen. Nach der Auswahl wird die Liste konvertiert und als Watchtabelle geöffnet.
Schnellsuche 		In diesem Textfeld können Sie eine Schnellsuche innerhalb der Expertenliste durchführen. Es werden die Spalten Parameter, Parametertext und Wert durchsucht.
Änderungen aufsammeln 		Solange die Taste aktiviert ist, werden alle von Ihnen geänderten Konfigurationsdaten mit der Wirksamkeit "Sofort" und "Restart" gesammelt und die neuen Werte werden nicht im Zielsystem wirksam. Erst durch Klicken auf **Änderungen aktivieren** werden alle gesammelten Änderungen der Konfigurationsdaten im Zielsystem wirksam Siehe unten.
Änderungen aktivieren 		• Wenn alle gesammelten Änderungen von Konfigurationsdaten **nur** mit der Wirksamkeit **Sofort** versehen sind, dann erfolgen die Änderungen sofort nach Anklicken der Schaltfläche **Änderungen aktivieren**. • Wenn unter den gesammelten Änderungen in Konfigurationsdaten welche mit Wirksamkeit **Restart** sind, werden **alle** Änderungen erst nach einem **Restart** des TO wirksam.
Auswahlliste Konfiguration		Hier können Sie wählen, ob Sie in der Tabelle der Expertenliste die Parameter für eine Linearachse, Rundachse jeweils für Standard, Kraft oder Druck anzeigen bzw. ändern wollen.
Restart 		Für Konfigurationsdaten, die erst nach einem Restart des TOs wirksam werden, können Sie den Restart entweder bewirken, indem Sie unter Systemvariable "restartActivation" aktivieren oder indem Sie auf die Taste **Restart** klicken. Die Taste bewirkt einen Restart der Achse. Die Taste wird aktiv, wenn Sie nach einem Restart ein wirksames Datum verändert und noch nicht übernommen haben.
Parameter 		Hier wird der Name des Konfigurationsdatums bzw. der Systemvariable angezeigt. Klicken Sie auf das Pluszeichen um die gesamte Struktur zu öffnen.
Parametertext		Hier wird eine Kurzbeschreibung der Systemvariable bzw. des Konfigurationsdatums angezeigt.
Offlinewert		Hier wird der Wert des Konfigurationsdatum bzw. der Systemvariable angezeigt. Abhängig von der Art des Parameters können Sie den Wert direkt als Zahlenwert eintragen oder Sie wählen einen symbolischen Bezeichner in der Auswahlliste.
Projektierter Wert	(nur ONLINE, Konfigurationsdatum)	Hier wird der projektierte Wert des Konfigurationsdatums angezeigt. Dieser Wert ist im RAM des Zielgeräts gespeichert.

Schaltfläche/Tabellenspalte		Bedeutung/Hinweis
Aktualwert	(nur ONLINE)	Hier wird der Aktualwert der Systemvariablen bzw. der Konfigurationsdaten angezeigt. Den Wert von Systemvariablen können Sie direkt im ONLINE-Modus ändern. Änderungen bei den Konfigurationsdaten können Sie nur in der Spalte **Next-Wert** vornehmen. Aktualwerte werden im **Aktualspeicher** abgelegt. Um diese Werte in den RAM-Speicher zu übernehmen, müssen Sie erst im Menü **Zielsystem > Aktual nach RAM kopieren** wählen. **Hinweis:** ONLINE veränderte Systemvariablen können nicht in den RAM kopiert werden. Dadurch ist auch kein **Sichern auf Speicherkarte (Ram2Rom)** bzw. **Sichern im Engineeringprojekt (Laden Konfigurationsdaten in PG)** möglich. Siehe Speicherkonzept SIMOTION.
Next-Wert	(nur ONLINE, Konfigurationsdatum)	Hier können Sie den **Next-Wert** des Konfigurationsdatums im ONLINE-Modus eintragen. Dieser Wert wird dann als Aktualwert übernommen, abhängig davon wann das Konfigurationsdatum wirksam wird. Bei Wirksamkeit sofort wird der neue Wert umgehend nach Drücken der EINGABETASTE bzw. nach Auswahl eines Wertes im Zielsystem wirksam.
Einheit		Hier wird die Einheit der Systemvariable bzw. des Konfigurationsdatums angezeigt.
Wirksamkeit		Hier wird angezeigt, wann der Wert des geänderten Konfigurationsdatums wirksam wird z. B. bei **restart** wird der geänderte Wert erst nach einem Restart im Zielsystem wirksam. Bestimmte Konfigurationsdaten können Sie im ONLINE-Modus nicht ändern. Ab V4.1.2 wird diese Spalte auch OFFLINE angezeigt.
Datentyp		Hier wird der **Datentyp** der Systemvariable bzw. des Konfigurationsdatums angezeigt (z. B. INT für einen ganzzahligen Wert).
Minimum		Hier wird der minimale Wert der Systemvariable bzw. des Konfigurationsdatums angezeigt.
Maximum		Hier wird der maximale Wert der Systemvariable bzw. des Konfigurationsdatums angezeigt.
Filter		In dieser Zeile können Sie für die verschiedenen Spalteninhalte Kriterien eingeben, nach denen die Tabelle gefiltert werden soll oder aus bereits vordefinierten Filtern in der Auswahlbox wählen. Um die Filterung zurückzusetzen, wählen Sie in der Auswahlbox das Kriterium **Alle** aus oder klicken Sie auf das Symbol 

2.4.2 Expertenliste verwenden

So zeigen Sie Hilfe zu Systemvariablen und Konfigurationsdaten an

1. Wählen Sie **Hilfe > Hilfe zum Kontext** oder drücken Sie **SHIFT+F1**.

   Der Cursor verändert sich zum Fragezeichen.

2. Klicken Sie in der Expertenliste auf das Konfigurationsdatum bzw. die Systemvariable, zu dem Sie Hilfe benötigen.

   Die Hilfe dazu wird aufgeblendet.

So ändern Sie die Daten in der Expertenliste Offline

1. Wählen Sie in der Expertenliste das Register für Konfigurationsdaten bzw. Systemvariablen.

   Eine tabellarische Übersicht wird angezeigt.

2. Klicken Sie in der Spalte **Parameter** der Expertenliste auf das Pluszeichen vor einem Eintrag, um Untereinträge anzuzeigen.

   Gelb hinterlegte Einträge können Sie nicht ändern.

3. Klicken Sie in die Spalte **Offlinewert** des zu ändernden Eintrags.

4. Tragen Sie den Wert direkt ein, oder wählen Sie einen Wert aus, wenn eine Auswahlliste aufgeblendet wird.

So führen Sie eine Schnellsuche durch

Gibt man einen Suchbegriff vor, dann springt der Cursor in die erste Zeile, in der dieser Suchbegriff gefunden wird.

1. Geben Sie den Suchbegriff ein und drücken Sie die EINGABETASTE.

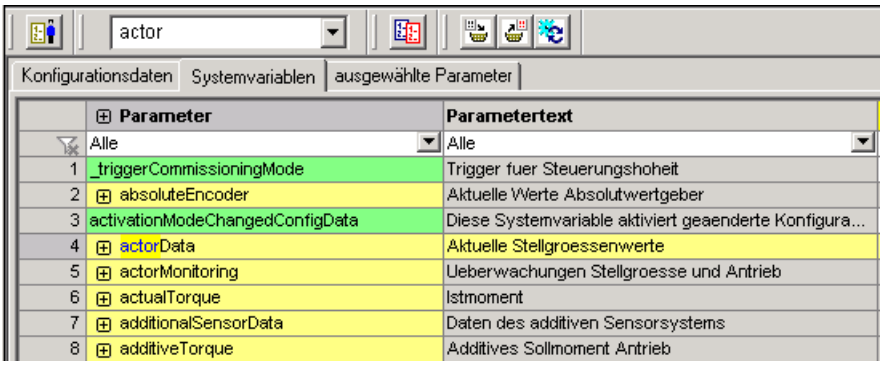


Bild 2-12 Beispiel für Schnellsuche

Wenn es sich um einen schreibbaren Parameter handelt, springt der Cursor direkt in das Wertefeld.

Wird eine Strukturhülse gefunden, die keinen eigenen Parameterwert hat, wird direkt in die Zelle gesprungen, in der der Begriff gefunden wurde.

Suche fortsetzen:

1. Drücken Sie die Taste **F3**.

   Die nächste Zelle wird gesucht, in der dieser Begriff vorkommt.

Wenn die Suchfunktion keinen weiteren Eintrag mehr in der Liste findet, wird eine Meldung ausgegeben.

So navigieren Sie mit den Pfeiltasten innerhalb der Expertenliste

Mit den Pfeiltasten kann jede Zelle innerhalb der Expertenliste angewählt werden.

1. Drücken Sie eine Pfeiltaste.

   Die nächstgelegene Zelle wird angewählt.

2. Mit der Tastenkombination **Strg+Pfeil** springt der Cursor an das Ende bzw. den Anfang der jeweiligen Zeile bzw. an das Ende bzw. an den Anfang der jeweiligen Spalte.

So selektieren Sie Zellen mit der Tastatur

Wenn Sie die gesamte Expertenliste, inklusive der nicht im sichtbaren Bereich gelegenen Zellen, selektieren wollen:

1. Drücken Sie die Tastenkombination **Strg+A**.

Im Online-Fall wird durch diese Tastenkombination auch ein Lesevorgang für sämtliche Parameterwerte angestoßen (bei Technologieobjekten ab V4.0).

Einzelne Zellbereiche können mittels Navigieren mit den Pfeiltasten oder mit der Maus selektiert werden:

1. Durch gleichzeitiges Drücken der **Shift**-Taste werden die Zellen im Zuge des Navigierens selektiert.

Zellbereiche können auch durch Ziehen mit der Maus selektiert werden:

1. Setzen Sie den Cursor in eine Zelle, drücken Sie die linke Maustaste und ziehen Sie den Mauszeiger über den zu selektierenden Zellbereich.

So kopieren Sie Zellinhalte in andere Windows-Anwendungen

Selektierte Zeilen können über die Windows-Zwischenablage in andere Windows-Anwendungen kopiert werden:

1. Drücken Sie die Tastenkombination **Strg+C** oder wählen Sie im Kontextmenü **Kopieren in Zwischenablage**.

Aus der Zwischenablage können die Zellinhalte z. B. in ein Excel- oder Word-Dokument eingefügt werden.

Über den Kontextmenü-Eintrag **Parameterpfad in Zwischenablage kopieren** kann der vollständige Strukturpfad in die Zwischenablage kopiert werden und von dort zurück in andere SCOUT-Anwendungen (ST-Programm, Watchtabelle , etc.).

Mit **Zellinhalt in Zwischenablage kopieren** können Sie den Inhalt von Zellen der Expertenliste in die Zwischenablage übernehmen.

So erstellen Sie eine Watchtabelle

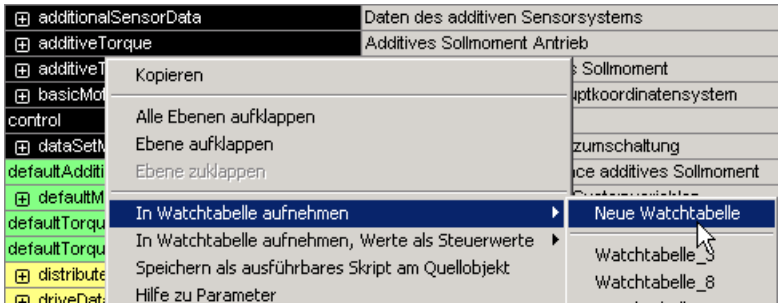


Bild 2-13 Kontextmenü in der Expertenliste (rechte Maustaste drücken)

Tabelle 2- 3 Kontextmenü in der Expertenliste

Menüpunkt		Bedeutung/Funktion
Kopieren		Kopiert markierte Parameter.
Alle Ebenen aufklappen		Klappt alle Ebenen auf.
Ebene aufklappen		Klappt alle markierten Ebenen auf.
Ebene zuklappen		Klappt Ebenen, die aufgeklappt sind, zu.
In Watchtabelle aufnehmen		
	Neue Watchtabelle	Erstellt eine neue Watchtabelle und öffnet sie in Register **<Watchtabellenname>** der Detailanzeige.
	Watchtabellenname	Fügt Parameter zu einer bereits vorhandenen Watchtabelle hinzu.
In Watchtabelle aufnehmen, Werte als Steuerwerte		
	Neue Watchtabelle	Erstellt eine neue Watchtabelle und öffnet sie in Register **<Watchtabellenname>** der Detailanzeige.
	Watchtabellenname	Fügt Parameter zu einer bereits vorhandenen Watchtabelle hinzu.
Speichern als ausführbares Skript am Quellobjekt		Speichert als ausführbares Script.
Hilfe zu Parameter		Ruft die Hilfe zu dem ausgewählten Parameter auf.

Vorgehensweise zum Erstellen einer Watchtabelle

1. Markieren Sie die Zeilen in der Expertenliste zum Übernehmen in die Watchtabelle.
2. Wählen Sie im Kontextmenü (rechte Maustaste drücken) **In Watchtabelle aufnehmen > Neue Watchtabelle**. Es öffnet sich ein Dialog **Watchtabelle einfügen**.

   Hinweis

   Zum Hinzufügen in eine bereits vorhandene Watchtabelle wählen Sie **In Watchtabelle aufnehmen > <Watchtabellenname>**. Es wird kein weiterer Dialog aufgerufen. Die Werte werden zu der bereits vorhandenen Watchtabelle hinzugefügt.

3. Im Dialog **Watchtabelle einfügen** können Sie Name, Autor, Version, und Kommentar eintragen. Name ist automatisch vorbelegt.

Hinweis

Sie können auch das Ergebnis eines Expertenlistenvergleichs als Watchtabelle oder als ausführbares Script speichern, siehe Expertenlisten vergleichen (Seite 52).

Hinweise zum Aufbau von Scripten:

- Jedes konvertiertes Script erhält eine Kopfzeile mit Datum und Herkunft, danach eine Leerzeile.
- In der nächsten Zeile wird der Logging-Mechanismus der Script-Engine eingeschaltet. Dadurch wird jeder Schreib- und Lesevorgang im Script-Ausgabefenster ausgegeben.
- Überschriften in der Watchtabelle werden als Kommentare mit einer führenden zusätzlichen Leerzeile in das Script eingetragen.
- Kommentare in der Watchtabelle werden als einfache Kommentare ohne zusätzliche Leerzeile in das Script eingetragen.
- Für jeden schreibbaren Parameter in der Watchtabelle wird ein Schreibauftrag in das Script eingetragen. Zahlen und Enums werden als Integerwerte geschrieben. Rechts neben dem Schreibauftrag wird als Kommentar der Parametertext des Parameters hinzugefügt.
- Nur lesbare Parameter in der Watchtabelle werden ab V4.2 in Scripten nicht mehr gespeichert. Beim Speichern als Watchtabelle wird nur der Parameter ohne Wert in die Watchtabelle aufgenommen.

Hinweise zur Verwendung von Scripten:

- Veränderte Einheitensysteme werden beim Ausführen des Scriptes nicht berücksichtigt. Sie müssen selbst sicherstellen, dass das Übernehmen der Parameterwerte im aktuellen Kontext sinnvoll ist.
- Strukturverändernde Parameter (z. B. TypeOfAxis.typeOfAxis) werden nicht gesondert behandelt. Sie müssen selbst sicherstellen, dass diese Parameter in der Parameterauswahl ganz oben stehen.
- Es wird immer nur der Actualwert geschrieben, d. h., das Script darf Online nicht ausgeführt werden, wenn Konfigurationsdaten in ihm enthalten sind.

Anwendung: Sie können mit Scripten die Parametrierung von einer Achse auf eine andere übertragen.

2.4.3 Expertenlisten vergleichen

Beschreibung

In diesem Fenster können Sie die Unterschiede in den Einstellungen der Parameter von max. 5 Objekten (TOs) und die Unterschiede im ONLINE- und OFFLINE-Modus eines Objekts vergleichen.

Die dargestellten Daten sind immer die aktuell wirksamen Daten, für TOs im ONLINE-Modus sind das die Aktualwerte. Systemvariablen werden nicht ständig aktualisiert, sondern erst, wenn Sie mit **Neuer Vergleich** die Darstellung aktualisieren.

Es können sowohl die Werte der Vergleichsobjekte und der Referenzobjekte geändert werden. Nicht änderbare Objekte werden mit "???" angezeigt.

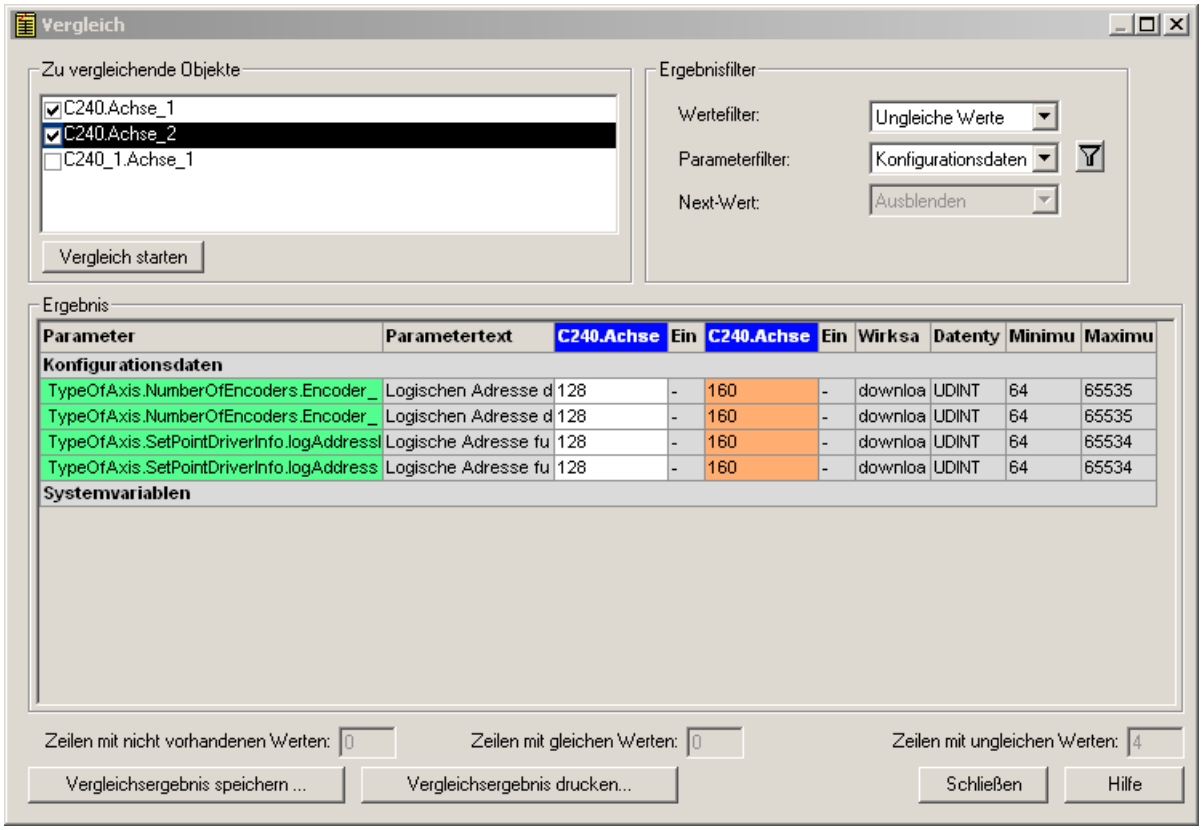


Bild 2-14 Expertenlistenvergleich

Es werden die Parameter mit unterschiedlichen Einstellungen bzw. bei einem Objekt nicht vorhandene Parameter in einer Liste dargestellt.

Hinweis

Ein Vergleich ist nur innerhalb von Gruppen kompatibler Objekte (TOs) möglich. Es werden nur die möglichen Objekte zum Vergleich angeboten.

Schaltfläche/Tabellenspalte	Bedeutung/Hinweis
zu vergleichende Objekte	Aktivieren Sie die Checkbox vor den Objekten, die Sie miteinander vergleichen wollen. Sie können max. bis zu fünf Objekte miteinander vergleichen und Objekte im OFF- und ONLINE-Modus. Im ONLINE-Modus ist jeweils eine Checkbox für die ONLINE- und OFFLINE-Parametrierung eines Objekts vorhanden. Als Referenzliste dient jeweils die oberste Liste, die gegraut dargestellt wird. Führend für die Anzeige sind immer die Struktur und damit die Parametermenge des Referenzobjektes. Damit kann bei den Vergleichsobjekten nur angezeigt werden, ob sie den Parameter des Referenzobjektes haben oder nicht. Nicht ausgewiesen werden Parameter, die nur die Vergleichsobjekte haben jedoch nicht das Referenzobjekt. Ein Wertunterschied einer Einstellung wird durch eine einheitliche farbliche Hinterlegung der betroffenen Zelle in Hell-Orange nur in den Spalten der betroffenen Vergleichsobjekte angezeigt – die betroffene Wertezelle des Referenzobjektes erhält keine Hinterlegung
Neuer Vergleich	Klicken Sie auf **neuer Vergleich**, um die Einstellungen der Parameter der gewählten Objekte zu vergleichen. Unter **Ergebnis** werden die verglichenen Einstellungen aufgelistet.
Wertefilter	Hier wählen Sie, welche Arten von Parametern nach dem Vergleich unter **Ergebnis** angezeigt werden sollen.
nicht vergleichbar	Es werden nur die nicht vergleichbaren Parameter unter **Ergebnis** angezeigt.
ungleiche Werte	Es werden nur die Parameter mit ungleichen Werten unter **Ergebnis** angezeigt.
gleiche Werte	Es werden nur die Parameter mit gleichen Werten unter **Ergebnis** angezeigt.
Anzeigefiltermodus	
Parameterfilter aktiv	Es werden nur die Parameter angezeigt, die unter **Anzeigefilter** im Register **Parameterfilter** festgelegt sind.
nur Spaltenauswahl	Es werden nur die Spalten angezeigt, die unter **Anzeigefilter** im Register **Spaltenauswahl** festgelegt sind.
Anzeigefilter	Klicken Sie auf den Button, um das Fenster **Anzeigefilter** zu öffnen. Dort können Sie die Anzeige der Parameter filtern bzw. Spalten ausblenden.
Next-Wert	Es werden zusätzlich die Next-Werte (Werte im Next-Speicher) angezeigt.
Ergebnis	Unter **Ergebnis** werden Ihnen die gefunden Parameter angezeigt, die beim Vergleich der Objekte unterschiedliche Einstellungen haben bzw. andere Parameter, die bei einem Objekt nicht vorhanden sind. Unterschiedliche Werte und nicht vergleichbare Werte von Parametern werden farblich hervorgehoben angezeigt.
Zeilen mit nicht vergleichbaren Werten	Hier wird die Anzahl der Parameter angezeigt, die nur in einem der vergleichenden Objekte vorhanden sind und deswegen kein Vergleich der Werte möglich sind. Die nicht vergleichbaren Parameter werden farblich hervorgehoben unter **Ergebnis** angezeigt. Die Parameter sind farbig markiert.
Zeilen mit gleichen Werten	Hier wird die Anzahl der Parameter angezeigt, die bei allen Objekten den gleichen Wert besitzen.

Schaltfläche/Tabellenspalte	Bedeutung/Hinweis
Zeilen mit ungleichen Werten	Hier wird die Anzahl der Parameter angezeigt, die bei den Objekten ungleiche Werte besitzen. Die ungleichen Werte der Parameter werden farblich hervorgehoben unter **Ergebnis** angezeigt.
Vergleichsergebnis speichern	Klicken Sie auf den Button, um die Vergleichsergebnisse als neue Watchtabelle oder als ausführbares Skript zu speichern. Der Button ist nur aktiv, wenn die Parameter eines Geräts unter **Ergebnis** angezeigt werden. Nähere Informationen dazu, siehe Expertenlistenvergleich speichern (Seite 54).
Vergleichsergebnis drucken	Klicken Sie auf den Button, um die Vergleichsergebnisse in einer tabellarischen Übersicht zu drucken. Es wird jeweils die Parameternummer, der Wert für das jeweilige Objekt und die Einheit ausgedruckt.

Siehe auch

Expertenlistenvergleich speichern (Seite 54)

2.4.4 Expertenlistenvergleich speichern

Beschreibung

In diesem Fenster können Sie das Ergebnis des Expertenlistenvergleichs speichern.

Der Button ist nur aktiv, wenn die Parameter eines Geräts unter Ergebnis angezeigt werden.

- Klicken Sie **als Watchtabelle (ohne Werte)** an um eine neue objektgranulare Watchtabelle anzulegen. Die linke Spalte ist immer die Referenzspalte, d. h. es werden nur die Parameter des Referenzobjekts verglichen und auch gespeichert.
- Klicken Sie **als Watchtabelle, werte als Steuerwerte** an und wählen Sie das Objekt aus, dem Sie der Watchtabelle zuordnen möchten. Sie können wählen, aus welcher Spalte die Werte gespeichert werden, von dieser Spalte werden dann auch die Parameter gespeichert. Bei der Auswahl der linken Spalte, ist das Verhalten OK, da sie die Referenzspalte ist. Bei der Auswahl der rechten Spalte wird nur die Schnittmenge der Parameter gespeichert. Enthält die rechte Spalte "nicht vorhandene" Parameter, werden die nicht mit gespeichert. Enthält sie mehr Parameter als das Referenzobjekt, erscheinen diese auch nicht in der Liste, weil die linke Spalte als Referenzobjekt führend ist.
- Klicken Sie **Ausführbares Script am Quellobjekt** an und wählen Sie das Objekt aus, dem Sie das Script zuordnen möchten. Unter dem Objekt wird dann ein SCRIPTE-Ordner angelegt.

Siehe auch Expertenlisten vergleichen (Seite 52).

Sie können nur die Vergleichsergebnisse eines Objekts speichern. Vor dem Abspeichern müssen Sie angeben, welches Objekt gespeichert werden soll.

2.5 Verschaltung von Technologieobjekten

Technologieobjekte werden beim Anlegen automatisch verschaltet (Gleichlaufachse, Nocke, Messtaster). Zusätzliche Verschaltungen können über eine allgemeine Verschaltungsmaske vorgenommen werden.

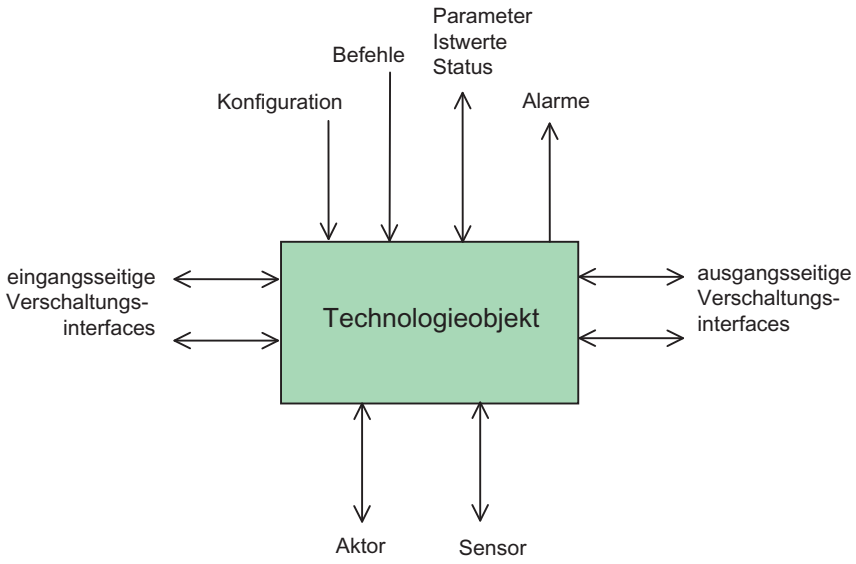


Bild 2-15 Verschaltungsinterfaces zu Technologieobjekten

2.5.1 Verschaltungsübersicht zu Technologieobjekten

Einleitung

Mit der Verschaltungsübersicht können Sie sich über einen Verschaltungsbaum alle Bewegungseingangs- bzw. ausgangsverschaltungen von Technologieobjekten im Projekt darstellen lassen. Die Baumdarstellung erlaubt es dabei, die Verschaltungen in Kaskaden darzustellen.

Eine Verschaltungstabelle zeigt, für das im Verschaltungsbaum ausgewählte TO, die ein- bzw. ausgangsseitig verschalteten TOs mit Interface-Bezeichnung.

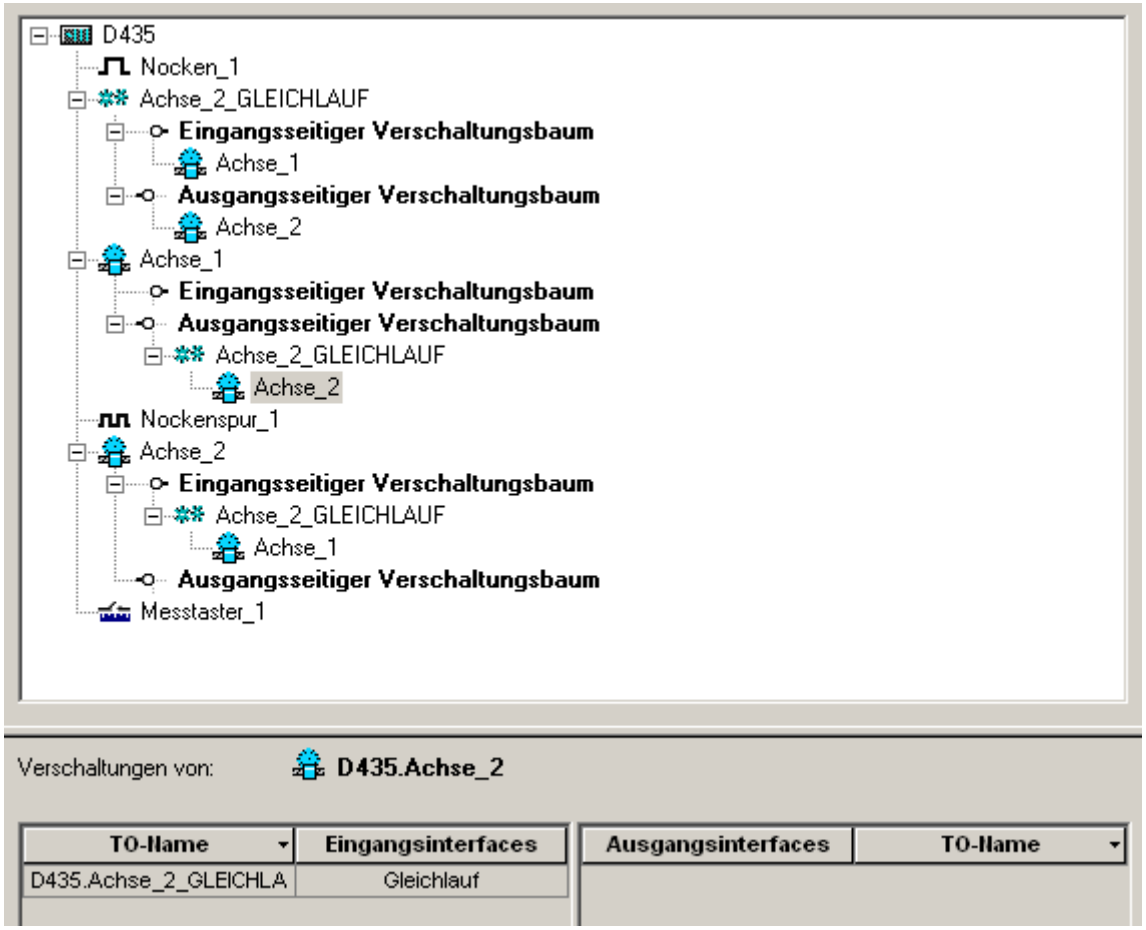


Bild 2-16 Verschaltungsübersicht

Eigenschaften der Verschaltungsübersicht

Die Verschaltungsübersicht starten Sie über die Schaltfläche bzw. über **Bearbeiten > Verschaltungsübersicht**.

- Nach dem Start der Verschaltungsübersicht wird zunächst die Geräteebene (CPUs) dargestellt. Unter jeder CPU können Sie sich dann alle instanziierten TOs flach darstellen lassen, indem Sie den entsprechenden Verschaltungsbaum der CPU öffnen. Die TOs werden sortiert nach TO-Typ, innerhalb eines TO-Typs alphabetisch angezeigt.
- Für die TOs, die direkt unterhalb der CPUs dargestellt werden, werden zwei Verschaltungsbäume angeboten:
  - Eingangsseitiger Verschaltungsbaum; zeigt die Bewegungseingangsverschaltung an
  - Ausgangsseitiger Verschaltungsbaum; zeigt die Bewegungsausgangsverschaltung an

- Alle TOs, die Bewegungseingänge und -ausgänge mit anderen TOs verschaltet haben, z. B. Achsen, Gleichlaufobjekte oder Geber können weiter aufgeklappt werden. Dies ist auf jeder Ebene der Hierarchie des Verschaltungsbaums möglich.
- Rekursion wird angezeigt, wenn im Verschaltungsbaum ein Knoten enthalten ist, der bereits einmal auf dem Pfad zur Wurzel des Baums existiert. Der Unterbaum ist aus Übersichtsgründen abgeschnitten. Fahren Sie dann am Knoten fort, der sich näher an der Wurzel befindet.

Die Verschaltungsbäume können nur sequentiell aufgeklappt werden, nicht alle auf einmal mit einem Klick.

Verschaltungstabelle anzeigen

Wählen Sie in der Verschaltungsübersicht ein TO aus.

Die zugehörige Verschaltungstabelle wird unterhalb der Verschaltungsübersicht angezeigt. Es werden die Eingangsverschaltungen und Ausgangsverschaltungen aller verschalteten TOs aufgelistet.

2.5.2 Implizite Verschaltung beim Anlegen

Beschreibung

Bei folgenden TOs werden automatisch Verschaltungen beim Anlegen generiert:

- Gleichlaufachse; Leit-Objekt wird auf die Folgeachse verschaltet
- Nocken; wird auf den Leitwert verschaltet
- Messtaster; wird auf den Leitwert verschaltet

Diese Verschaltungen werden auch in der Verschaltungsübersicht angezeigt.

2.5.3 Verschaltung über allgemeine Verschaltungsmaske im SCOUT (für Experten)

Zur Verschaltung von verschiedenen Technologieobjekten bietet SCOUT die Ansicht **Verschaltung**.

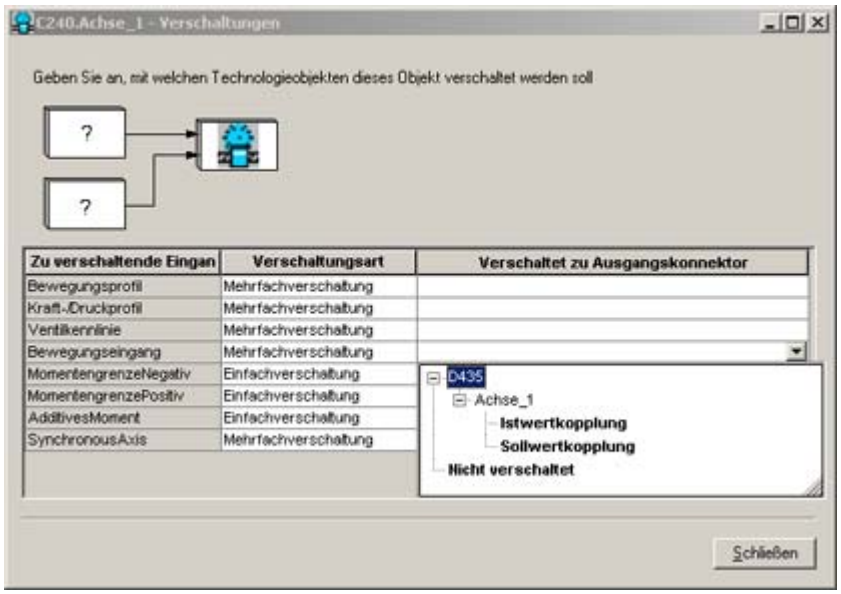


Bild 2-17 Verschaltung im SCOUT - Beispiel Achse

Auf der linken Seite werden alle eingangsseitigen Verschaltungsinterfaces des angewählten Objekts aufgeführt, auf der rechten Seite die typgleichen ausgangsseitigen Verschaltungsinterfaces. Es können nur die fett dargestellten ausgangsseitigen Interfaces ausgewählt werden.

- Bekommt eine dieser Zeilen den Fokus (anklicken), so können Sie einen Baum aufklappen, der alle Verschaltungsmöglichkeiten anzeigt.

Es ist nur möglich, die **fett** dargestellten Elemente auszuwählen.

Die Anzeige der Geräte und Technologieobjekte dient nur zur Orientierung.

- Eine direkte Eingabe in die Eingabezeile ist möglich.

Es werden nur vollständige und richtige Eingaben akzeptiert. Bei falschen Eingaben kann das Eingabefeld nicht verlassen werden.

Besitzt das eingangsseitige Verschaltungsinterface die Eigenschaft Mehrfachverschaltbarkeit und kann somit mit mehreren ausgangsseitigen Verschaltungsinterfaces verschaltet werden, wird nach dem Verbinden mit einem ausgangsseitigen Verschaltungsinterface eine weitere Zeile für die Verschaltung des eingangsseitigen Verschaltungsinterfaces eingefügt.

- Eine Verschaltung kann durch Löschen einer Zeile aufgelöst werden.

Der Wert wird mit Verlassen der Eingabezeile übernommen.

2.5.4 Allgemeine Eigenschaften von Verschaltungsinterfaces

Konfiguration der Verschaltungsinterfaces

Die eingangsseitigen Verschaltungsinterfaces werden angelegt:

- implizit mit der Instanziierung des TO
- über technologiespezifische Verschaltungsmasken
- über allgemeine Verschaltungsmaske

Aktivierung/Deaktivierung

Die eingangsseitigen Verschaltungsinterfaces werden über Befehl am betreffenden TO aktiv geschaltet oder sind defaultmäßig aktiv geschaltet.

Bei Mehrfachverschaltbarkeit eines eingangsseitigen Verschaltungsinterfaces erfolgt die Aktivierung/Umschaltung mit im Befehl für die technologische Funktion, z. B.

- Auswahl der Kurvenscheibe bei **_enableCamming()** über den Parameter **cam**
- Auswahl des Technologieobjekts für den Leitwert im Befehl **_setMaster()** über den Parameter **master**
- Auswahl des Technologieobjekts für den Bezugswert des MotionIn-Interfaces im Befehl **_run...basedMotionIn...()** über den Parameter **mastertype.reference**
- Auswahl eines Profils bzw. der Kurvenscheibe bei den **_run…Profile()**-Befehlen über den Parameter **profile**

Informationen hierzu finden Sie in den Funktionshandbüchern der betreffenden Technologieobjekte.

Anzeige von Status- und Verschaltungswerten

Die Anzeige für den Status der eingangsseitigen Verschaltung und Verschaltungswerte ist an den TO für die einzelnen Interfacetypen definiert.

Informationen hierzu finden Sie in den Funktionshandbüchern der betreffenden Technologieobjekte.

Gültigkeit

Verschaltungsinterfaces haben gültige Werte, wenn die Verschaltung hergestellt, und die Verschaltungswerte den Status ‘gültig’ haben, nichtgültige Verschaltungen/Verschaltungswerte haben den Wert 0.

Wird eine (virtuelle) Achse in Simulation gesetzt, werden die Sollwerte ausgangsseitig weiter aktualisiert, die Istwerte eingefroren.

Ob der Verschaltungswert oder der Ersatzwert nach dem Hochlauf gültig ist, kann aus den in Systemvariablen verfügbaren Status ausgelesen werden.)

Alarmreaktion

Bei einer fehlerhaften Verschaltung wird erst beim Aktivieren einer Funktion auf das Verschaltungsinterface ein Technologischer Alarm erzeugt.

Bei Mehrfachverschaltbarkeit eines eingangsseitigen Verschaltungsinterfaces muss ein Bezugs-TO ausgewählt werden.

Programmierte Funktionen werden beim Übergang von **STOPU** nach **STOP** abgebrochen, Verschaltungsfunktionen bleiben beim Übergang von **STOPU** nach **STOP** aktiv.

2.5.5 Verschaltungsinterfaces an den TO (für Experten)

Die eingangsseitigen Verschaltungsinterfaces können nur mit typgleichen ausgangsseitigen Verschaltungsinterfaces verschaltet werden.

TO	Interface		Typ
Drehzahlachse			
	eingangsseitige Verschaltungsinterfaces:		
		Bewegungseingang	Motion
		Momentengrenze positiv	LREAL
		Momentengrenze negativ	LREAL
		Additives Moment	LREAL
		Bewegungsprofil	MotionProfile
		Kraft-/Druck-Profil	ForceProfile
		Ventilkennlinie (Hydraulikachse)	ValveCharacteristics
	ausgangsseitige Verschaltungsinterfaces:		
		Bewegungssollwert	Motion
		Bewegungsistwert mit Extrapolation	Motion
		Ist-Moment	LREAL
Positionierachse			
	eingangsseitige Verschaltungsinterfaces:		
		wie Drehzahlachse	
	ausgangsseitige Verschaltungsinterfaces:		
		wie Drehzahlachse, zusätzlich:	
		Interface für Nocken, Nockenspur (wird bei Anlegen Nocken, Nockenspur auf Achse implizit vom System verschaltet)	spezifisch
		Interface für Messtaster (wird bei Anlegen Messtaster auf Achse implizit vom System verschaltet)	spezifisch
Folgeachse			
	eingangsseitige Verschaltungsinterfaces:		
		wie Positionierachse, zusätzlich:	
		Interface für Gleichlaufobjekt, (wird bei Anlegen Gleichlaufachse/ Gleichlaufobjekt implizit vom System verschaltet)	spezifisch
	ausgangsseitige Verschaltungsinterfaces:		
		wie Positionierachse, zusätzlich:	

Bahnachse			
	Eingangsseitige Verschaltungsinterfaces		
		Interface wie Folgeachse, zusätzlich:	
		Bahnbewegung	spezifisch
	Ausgangsseitiges Verschaltungsinterface		
		wie Positionierachse	
Gleichlaufobjekt			
	eingangsseitige Verschaltungsinterfaces:		
		Bewegungseingang	Motion
		Kurvenscheibe	CAM
	ausgangsseitige Verschaltungsinterfaces:		
		Interface für Folgeachse (wird bei Anlegen Gleichlaufachse/ Gleichlaufobjekt implizit vom System verschaltet)	spezifisch
Bahnobjekt			
	eingangsseitige Verschaltungsinterfaces:		
		Geschwindigkeitsprofil	MotionProfile
	ausgangsseitige Verschaltungsinterfaces:		
		Bahnbewegung (Bahnachse 1)	spezifisch
		Bahnbewegung (Bahnachse 2)	spezifisch
		Bahnbewegung (Bahnachse 3)	spezifisch
		Bahnsynchronbewegung	spezifisch
		Bewegungsausgang (Bahnobjekt.x)	Motion
		Bewegungsausgang (Bahnobjekt.y)	Motion
		Bewegungsausgang (Bahnobjekt.z)	Motion
Externer Geber:			
	eingangsseitige Verschaltungsinterfaces:		
		keine	
	ausgangsseitige Verschaltungsinterfaces:		
		Interface für Nocken, Nockenspur (wird bei Anlegen Nocken, Nockenspur implizit vom System verschaltet)	spezifisch
		Interface für Messtaster (wird bei Anlegen Messtaster implizit vom System verschaltet)	spezifisch
		Bewegungsistwert	Motion
		Bewegungsistwert mit Extrapolation	Motion
Messtaster			
	eingangsseitige Verschaltungsinterfaces:		
		Messtaster-Interface (wird bei Anlegen Messtaster implizit auf Achse, Externer Geber vom System verschaltet)	spezifisch
		Eingangs-Interface bei 'Mithörenden Messtaster' (ab V4.0) (über allgemeine Verschaltungsmaske)	spezifisch

	ausgangsseitige Verschaltungsinterfaces:		
		Ausgangs-Interface für 'Mithörenden Messtaster' (ab V4.0)	spezifisch
Nocken, Nockenspur			
	eingangsseitige Verschaltungsinterfaces:		
		Nocken-Interface (wird bei Anlegen Nocken, Nockenspur, implizit auf Achse, Externer Geber vom System verschaltet)	spezifisch
Kurvenscheibe			
	eingangsseitige Verschaltungsinterfaces:		
		keine	
	ausgangsseitige Verschaltungsinterfaces:		
		Kurvenscheibe	CAM
		Bewegungsprofil	MotionProfile
		Kraft-/Druck-Profil	ForceProfile
		Ventilkennlinie, (Hydraulikachse)	ValveCharacteristics
Temperaturregler			
	keine Verschaltungsinterfaces		
Addierobjekt			
	eingangsseitige Verschaltungsinterfaces:		
		Bewegungseingang1	Motion
		Bewegungseingang 2	Motion
		Bewegungseingang 3	Motion
		Bewegungseingang 4	Motion
	ausgangsseitige Verschaltungsinterfaces:		
		Bewegungsausgang	Motion
Formelobjekt			
	eingangsseitige Verschaltungsinterfaces:		
		Bewegungseingang 1	Motion
		Bewegungseingang 2	Motion
		Bewegungseingang 3	Motion
		LREAL-Eingang 1	LREAL
		LREAL-Eingang 2	LREAL
		LREAL-Eingang 3	LREAL
		LREAL-Eingang 4	LREAL
		DINT-Eingang 1	DINT
		DINT-Eingang 2	DINT
		DINT-Eingang 3	DINT
		DINT-Eingang 4	DINT
	ausgangsseitige Verschaltungsinterfaces:		
		Bewegungsausgang 1	Motion
		Bewegungsausgang 2	Motion
		Bewegungsausgang 3	Motion
		LREAL-Ausgang 1	LREAL

		LREAL-Ausgang 2	LREAL
		LREAL-Ausgang 3	LREAL
		LREAL-Ausgang 4	LREAL
		DINT-Ausgang 1	DINT
		DINT-Ausgang 2	DINT
		DINT-Ausgang 3	DINT
		DINT-Ausgang 4	DINT
Festes Getriebe			
	eingangsseitige Verschaltungsinterfaces:		
		Bewegungseingang	Motion
	ausgangsseitige Verschaltungsinterfaces:		
		Bewegungsausgang	Motion
Sensor			
	eingangsseitige Verschaltungsinterfaces:		
		keine	
	ausgangsseitige Verschaltungsinterfaces:		
		Sensorwert	LREAL
		Sensorwert Ableitung	LREAL
		Bewegungsausgang	Motion
Reglerobjekt			
	eingangsseitige Verschaltungsinterfaces:		
		Sollwert	LREAL
		Istwert	LREAL
		Vorsteuerung	LREAL
		Bewegungseingang Sollwert	Motion
		Bewegungseingang Istwert	Motion
	ausgangsseitige Verschaltungsinterfaces:		
		Stellwert in Bewegungsausgang	Motion
		Stellwert	LREAL

2.5.6 Verschaltungsinterfacetyp 'Motion' (für Experten)

Der Verschaltungsinterface-Typ **Motion** enthält den Bewegungsvektor (s, v, a) sowie interne Information, wie z. B. Gültigkeitsstatus, und gegebenenfalls Modulo-Information.

Bewegungsbasis

Der Bewegungsvektor kann eine unterschiedliche Bewegungsbasis haben: Position oder Geschwindigkeit.

Bei Bewegungsbasis "Geschwindigkeit" ist die Positionskomponente Null.

Eine Drehzahlachse stellt ausgangsseitig nur den Bewegungsvektor mit der Bewegungsbasis Geschwindigkeit zur Verfügung.

Über Befehle / Befehlsparameter (TO Achse) bzw. über Konfiguration (TO Addierobjekt, TO Festes Getriebe) wird die Bewegungsbasis berücksichtigt / eingestellt.

Die Komponenten des Bewegungsvektors werden am TO Achse, TO Festes Getriebe ausgangsseitig konsistent gehalten (Ableitung Positionsgröße ist Geschwindigkeitsgröße, Ableitung Geschwindigkeitsgröße ist Beschleunigung).

Die Komponenten des Bewegungsvektors werden am TO Addierobjekt und TO Formelobjekt ausgangsseitig nicht konsistent gehalten.

Zur Position kann auch die Geschwindigkeit und Beschleunigung in diesem Punkt gezielt vorgegeben werden.

Eingangsseitige Verschaltungsinterfaces 'Bewegungseingang' von Typ 'Motion'

Ob ein eingangsseitiges Verschaltungsinterface 'Bewegungseingang' vom Typ Motion einfach oder mehrfach verschaltbar ist, ist am TO-Typ festgelegt.

Folgende Technologieobjekte haben z. B. ein eingangsseitiges Verschaltungsinterface 'Bewegungseingang' vom Typ 'Motion':

- Drehzahlachsen
- Positionierachsen
- Gleichlaufachsen (Folgeachsen)
- Festes Getriebe
- Addierobjekt
- Formelobjekt
- Gleichlaufobjekt (Leitwert)

Ausgangsseitige Verschaltungsinterfaces von Typ 'motion'

Folgende TO haben z. B. ein ausgangsseitiges Verschaltungsinterface vom Typ 'Motion':

- Drehzahlachsen
- Positionierachsen
- Gleichlaufachsen (Folgeachsen)
- Bahnobjekt
- Festes Getriebe
- Addierobjekt
- Formelobjekt
- Externer Geber

2.5.7 Verschaltungsinterfacetyp 'LREAL' (für Experten)

Für skalare Größen ist ein Verschaltungsinterface vom Typ LREAL verfügbar.

Eingangseitiges Verschaltungsinterface an der Achse 'MomentengrenzePositiv' und 'MomentengrenzeNegativ' zur Vorgabe der Momentengrenzen

Zur Vorgabe der Momentengrenzen **B+/B-** können die eingangsseitigen Verschaltungsinterfaces 'MomentengrenzePositiv' und 'MomentengrenzeNegativ' der Achse mit ausgangsseitigen Verschaltungsinterfaces vom Typ LREAL von anderen TO verschaltet werden.

Voraussetzung dafür ist die Aktivierung des Technologiedatenblocks am TO Achse.

Die eingangsseitigen Interfaces 'MomentengrenzePositiv' und 'MomentengrenzeNegativ' sind nicht mehrfach verschaltbar und nicht verteilbar.

Ausgangsseitige Verschaltungsinterfaces vom Typ LREAL haben z. B.:

- TO Achse mit dem ausgangseitigen Verschaltungsinterface 'Moment'
- TO Formelobjekt mit dem ausgangsseitigen Verschaltungsinterface 'LRealOut'
- TO Sensor
- TO Reglerobjekt

Ausgangsseitiges Verschaltungsinterface 'Moment' an der Achse zur Bereitstellung des Istmoments

An der Achse ist ein ausgangsseitiges Verschaltungsinterface 'Moment' vom Typ LREAL zur Ausgabe des Istmoments an andere TO verfügbar.

Voraussetzung dafür ist die Aktivierung des Technologiedatenblocks am TO Achse.

Es wird das Istmoment ausgegeben, das vom Antrieb im additiven Technologiedatenblock geliefert wird.

Es wird in der an der Achse eingestellten Einheit ausgegeben.

Das ausgangsseitige Verschaltungsinterface ist mehrfach verschaltbar.

Das ausgangsseitige Verschaltungsinterface 'Istmoment' ist nicht verteilbar.

Eingangseitiges Verschaltungsinterface 'AdditivesMoment' an der Achse zur Vorgabe eines additiven Moments

An der Achse ist ein eingangseitiges Verschaltungsinterface 'AdditivesMoment' vom Typ LREAL zur Vorgabe eines additiven Moments an den Antrieb über den additiven Technologiedatenblock verfügbar.

Zur Vorgabe eines additiven Moments abhängig von anderen TO kann das eingangsseitige Verschaltungsinterface 'AdditivesMoment' an der Achse mit ausgangsseitigen Verschaltungsinterfaces vom Typ LREAL von anderen TO verschaltet werden.

Das eingangsseitige Interface 'AdditivesMoment' ist nicht mehrfach verschaltbar und nicht verteilbar.

Ausgangsseitige Verschaltungsinterfaces vom Typ LREAL haben z. B.:

- TO Achse mit dem ausgangseitigen Verschaltungsinterface 'Moment'
- TO Formelobjekt mit dem ausgangsseitigen Verschaltungsinterface 'LRealOut'
- TO Sensor
- TO Reglerobjekt

2.6 Technologieobjekt-Trace

2.6.1 Einführung

Sie können die Befehle an einem Technologieobjekt beobachten und den Zugriff auf die Systemvariablen und Konfigurationsdaten verfolgen

In SIMOTION können Technologieobjekte zur Laufzeit durch Konfigurationsdaten, Systemvariablen und Befehle beeinflusst werden. Um die Beeinflussungen eines Technologieobjekts zur Laufzeit darzustellen, steht ab V4.2 TO-Trace/Objekttrace zur Verfügung, der die letzten n Aktionen am Technologieobjekt zeitlich geordnet aufzeichnet.

Hinweis

Der Trigger im Trace kann für ein TO, z. B. für eine Achse, nicht gleichzeitig für den Gerätetrace und den TO-Trace gesetzt werden.

Die Aufzeichnung kann auf Anforderung aus dem Technologieobjekt ausgelesen und im ES dargestellt werden.

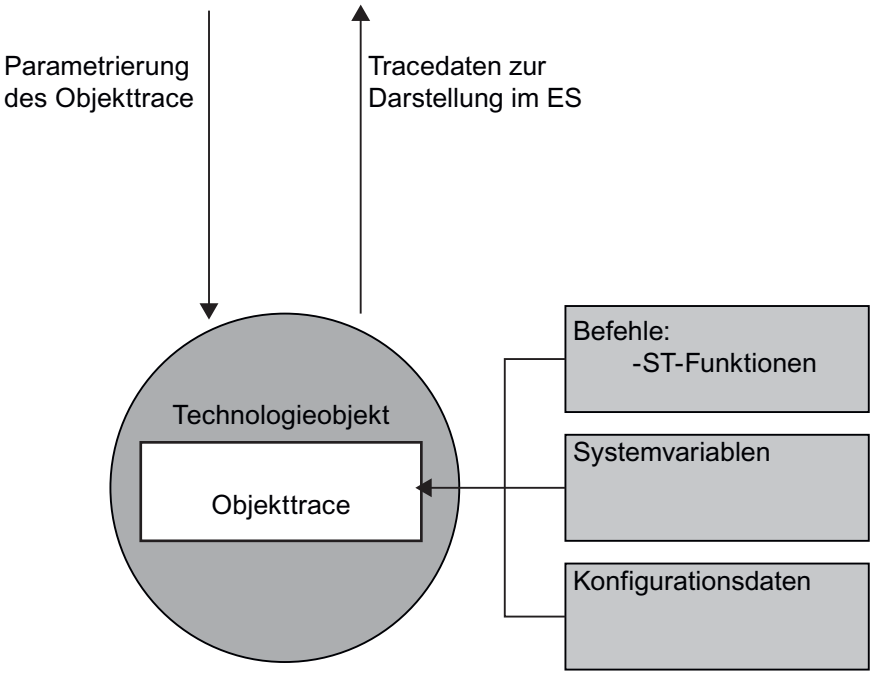


Bild 2-18 Prinzip des Objekttrace

Der TO-Trace dient der Diagnose von TO`s und stellt die Informationen in einer Momentaufnahme dar. Er ist TO-spezifisch, also für jedes Technologieobjekt einzeln zu konfigurieren.

Er zeichnet Befehle und den Zugriff von Konfigurationsdaten und Systemvariablen mit ihrem Befehlsstatus sowie Fehlerstatus beim Auslesen des TO-Trace auf. Der TO-Trace wird in einer Tabelle/Liste mit Zeitstempel dargestellt.

TO-Trace steht für folgende Technologieobjekte zur Verfügung:

- Drehzahlachse
- Positionierachse
- Gleichlaufachse
- Bahninterpolation
- Gleichlaufobjekt
- Externer Geber
- Messtaster
- Nocken
- Nockenspur
- Addierobjekt
- Formelobjekt
- Sensor
- Temperaturkanal
- Festes Getriebe
- Reglerobjekt

2.6.2 Ereignisse

2.6.2.1 Ereignisse

Mit dem TO-Trace ist es möglich Ereignisse, die ein Technologieobjekt zur Laufzeit beeinflussen, aufzuzeichnen.

Ereignisse

Solche Ereignisse sind:

- das Ausführen von Befehlen
- das Schreiben von Systemvariablen und
- das Schreiben bzw. Aktivieren von Konfigurationsdaten

2.6.2.2 TO-Trace für Befehle oder Funktionsbausteine

Sie können die letzten am TO abgesetzten Befehle mit TO-Trace auslesen

Je weiter die Bearbeitung eines Befehls fortgeschritten ist, desto mehr Informationen können Sie über die jeweiligen Befehle auslesen. So können Sie bei bereits beendeten Befehlen den Status (Executed oder Aborted) und einen eventuellen Fehlercode bestimmen. Da TO-Trace eine Momentaufnahme macht, werden diese Informationen zum Zeitpunkt des Auslesens bestimmt. Der Status wird nach dem Ende des Befehls nicht automatisch aktualisiert. Sie können eine Aktualisierung durch das erneute Auslesen bewirken.

Folgende Informationen werden vom TO-Trace bereitgestellt:

Tabelle 2- 4 Informationen zu Befehlen im TO-Trace

Parameter	Beschreibung
Befehlstyp	Aus dem Befehlstyp ist der Klartextname des Befehls ersichtlich.
Zeitstempel 1	Im Zeitstempel ist die RT Zeit beim Absetzen des Befehls hinterlegt.
Zeitstempel 2	Im Zeitstempel ist die RT Zeit beim Wirksamwerden des Befehls hinterlegt.
Zeitstempel 3	Im Zeitstempel ist die RT Zeit beim Ende des Befehls hinterlegt.
Taskstack	Aus dem Taskstack ist die Aufrufstelle des Befehls im Anwenderprogramm ersichtlich.
Befehlsstatus / Bausteinstatus	Zustand des Befehls. Zusätzlich wird bei Bewegungsbefehlen die Phase der aktuellen Bewegung ergänzt. Mögliche Werte sind somit: • BUFFERED • WAIT_SYNC_START • IN_EXECUTION • IN_ACCELERATION • IN_CONSTANT_MOTION • IN_DECELERATION • INTERPOLATION_DONE • EXECUTED • ABORTED
Fehlercode	Rückgabewert beim Befehlsende. Der Fehlercode ist nur beim Bewegungsstatus EXECUTED oder ABORTED gültig.
n Befehlsparameter	Der TO-Trace zeichnet die Übergabeparameter des Befehls auf. Zu jedem Befehlsparameter werden der programmierte Wert und der wirksame Wert aufgezeichnet.
n Rückgabewerte	Neben den Befehlsparametern werden sämtliche Rückgabewerte der Befehle aufgezeichnet. Die Aufzeichnung erfolgt am Befehlsende im Anwenderprogramm.

2.6.2.3 TO-Trace für PLCopen Funktionsbausteine

Bausteinaufrufe der PLCopen Funktionsbausteine werden im Befehlspuffer eingetragen. Flankengetriggerte Bausteine werden mit ihrer steigenden Flanke in den Puffer eingetragen. Bei pegelgetriggerten Bausteinen erfolgt das Eintragen in den Puffer mit jeder Änderung des Eingangspegels.

2.6.2.4 TO-Trace für Systemvariablen

Es werden schreibende Zugriffe mit TO-Trace aufgezeichnet

Tabelle 2- 5 Informationen zu Systemvariablen im TO-Trace

Parameter	Beschreibung
Systemvariablenname	Klartextname der Systemvariablen.
Zeitstempel	Im Zeitstempel ist die RT Zeit beim Schreiben der Systemvariablen hinterlegt.
Datentyp	Datentyp der Systemvariablen.
neuer Systemvariablenwert	In die Systemvariable geschriebener Wert.

2.6.2.5 TO-Trace für Konfigurationsdaten

Es werden schreibende Zugriffe mit TO-Trace aufgezeichnet

Tabelle 2- 6 Informationen zu Konfigurationsdaten im TO-Trace

Parameter	Beschreibung
Konfigurationsdatum	Klartextname des Konfigurationsdatums.
Zeitstempel	Im Zeitstempel ist die RT Zeit beim Schreiben des Konfigurationsdatums hinterlegt.
Zieldatenbereich	Aus dem Zieldatenbereich ist die Wirksamkeit des Konfigurationsdatums erkennbar. Mögliche Werte sind der NEXT Datenbereich oder der ACTUAL Datenbereich.
Datentyp	Datentyp des Konfigurationsdatums.
neuer Wert	In das Konfigurationsdatum geschriebener Wert.

2.6.3 Aufruf

Es gibt verschiedene Möglichkeiten ein TO-Trace aufzurufen

Aufruf über das Hauptmenü

Sie könne den TO-Trace aufrufen wenn Sie unter **Zielsystem -> Technologieobjekt-Trace** auswählen. Dazu muss im Projektnavigator ein Gerät markiert sein, sonst bleibt der Menüpunkt ausgeblendet. Bei mehreren Geräten können Sie den TO-Trace für jedes Gerät einzeln aufrufen.

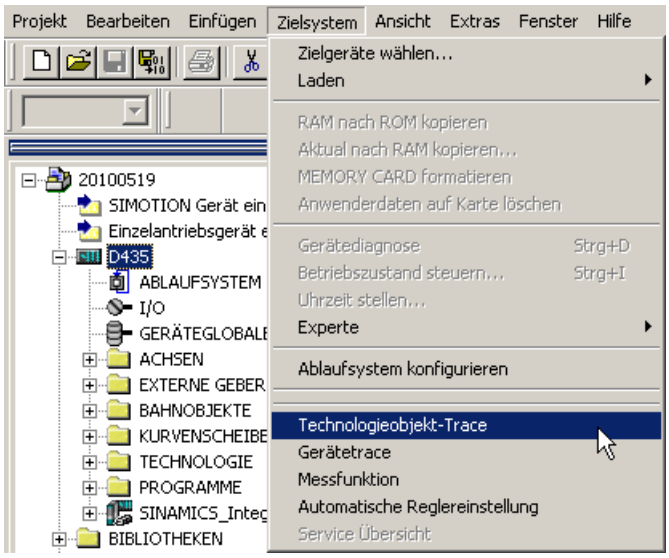


Bild 2-19 Aufruf über das Hauptmenü

Aufruf über das Kontextmenü

Sie können den TO-Trace über das Kontextmenü des Geräts aufrufen.

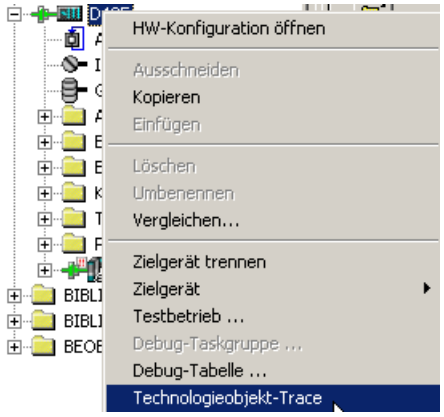


Bild 2-20 Aufruf über das Kontextmenü

Aufruf über Toolleiste

In der Toolleiste befindet sich eine Schaltfläche TO, die den TO-Trace aufruft.

Bild 2-21 Button Trace

2.6.4 Parametrierung

Die Parametrierung und Tracedaten stehen Offline und Online zur Verfügung

Offline ist nur ein Teil der Funktionalität verfügbar:

- Parametrierung des TO-Trace
- Speichern der TO-Trace Parametrierung
- Anzeige des letzten hochgeladenen TO-Trace
- Speichern der Daten in einem TO-Trace File
- Öffnen eines bestehenden TO-Traces

Zusätzliche Onlinefunktionalität:

- Parametrierung des TO-Trace und Download dieser ins RT
- Upload der Parametrierung
- Starten, Stoppen der Traceaufzeichnung
- Upload der Traceaufzeichnung

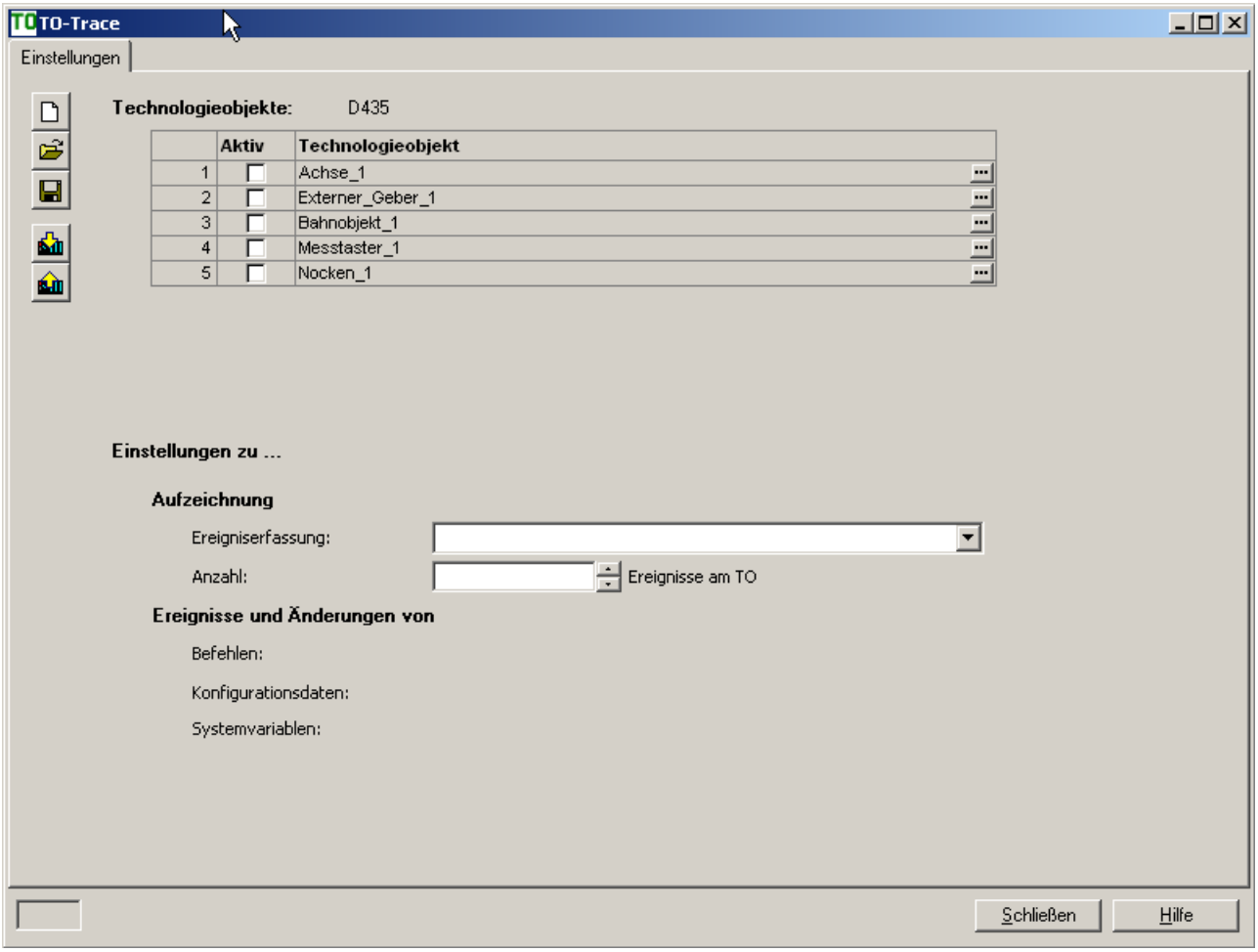


Bild 2-22 Dialog für Parametrierung

Tabelle 2- 7 Mögliche Einstellungen

Funktion	Beschreibung
Menüpunkte	
	Zurücksetzen der Parametrierung.
	Download Parametrierung.
	Upload Parametrierung.

Funktion	Beschreibung
Technologieobjekte	Es werden alle im Projekt angelegten Technologieobjekte unter der Einstellung **Technologieobjekte** angezeigt. Durch klicken auf die Zelle einer Zeile, wird die Zeile markiert. Die Einstellungen zu einer Zeile werden unterhalb der Technologieobjekte angezeigt. Die Einstellungen zu einem Technologieobjekt können über den Button ... geändert werden (Parametrierung eines TOs siehe Dialog Ereignisauswahl Technologieobjekt-Trace (Seite 77)). Default sind alle Technologieobjekte abgewählt. Es muss das jeweilige Technologieobjekt für die Aufzeichnung angewählt werden.
Nr	Nummer des Technologieobjekts.
Aktiv	Trace für das TO aktivieren oder deaktivieren.
Technologieobjekt	Name des Technologieobjekts.
	Name des TO-Traces wenn ein Trace aus dem Katalog geöffnet wird. In anderen Fällen wird ein Defaultname verwendet.
Button ...	Mit diesem Button wird der Dialog **Signalauswahl Trigger** aufgerufen (siehe unten Triggerbedingungen).

Befehle

Im Dialog **Befehle** sehen Sie die Befehle des Technologieobjektes. Die Sortierung und Gruppierung entspricht hier der Darstellung in der Befehlsbibliothek.

Per Voreinstellung sind immer alle Befehle angewählt.

Durch Ab- oder Anwählen der Checkbox eines Knotens wird der ganze darunterliegende Zweig ab- oder angewählt.

Aufzeichnungsdaten

Tabelle 2- 8 Mögliche Einstellungen

Funktion	Beschreibung
Menüpunkte	
	Speichern der Aufzeichnung im CVS Format.
Spalten	
Typ	Typ der Aufzeichnung.
Zeitstempel	Zeitstempel der Aufzeichnung.
Technologieobjekt	Das Bezugsobjekt.
Ereignis	Das aufgezeichnete Ereignis.
Aktueller Status	Status der Aufzeichnung.
Detailinformationen zur Auswahl	
Zeitstempel	
Codestelle	
Ereignis	
Aktueller Status	

Hinweis

Bei PLCopen Basteinen sind die Rückgabewerte gleichzeitig Aktualwerte und ändern sich während der Baustein läuft. Außerden entspricht der Zeitstempel für **abgesetzt** der **Execute-Flanke** am Baustein.

Steuertoolbar

Es gibt eine Toolbar zum Steuern der Aufzeichnungen. Die Toolbar ist immer sichtbar, wenn das SnapIn auch sichtbar ist. Der Trace kann hier für alle Technologieobjekte gleichzeitig gestartet, gestoppt und eine Aufzeichnung hochgeladen werden. Das Gerät für das der Trace gültig ist, ist fest und nicht veränderbar. Für jedes Gerät wird ein eigenes Trace-Fenster geöffnet.

Der aktuelle Status des Trace ist für alle Technologieobjekte in einem Steuerelement sichtbar. Die Statusanzeige gibt den Zustand und Bereitschaft des TO-Trace wieder.

Mögliche Zustände:

- TO-Trace inaktiv (wenn alle beendet oder nicht gestartet sind)
- TO-Trace Parametrierung fehlerhaft (wenn mind. einer eine falsche Parametrierung besitzt)
- TO-Trace aktiv (wenn mindestens einer läuft oder auf Trigger wartet)

Über den Button kann noch der individuelle Zustand an jedem TO angezeigt werden.

Mögliche Zustände:

- inaktiv
- bereit
- warten auf Trigger
- läuft
- Parametrierung fehlerhaft
- Aufzeichnung beendet

Sie können die TO-Tracedaten mit Button **Upload Tracedaten** laden. Der Upload der TO-Trace Daten kann immer erfolgen. Ist beim Hochladen ein endloser Trace eingestellt, wird die Aufzeichnung jeweils am Vorhandenen Trace angehängt. Ist der zeitbegrenzte Trace aktiviert, wird der vorhandenen Trace überschrieben.

Bedingungen für Aufzeichnung

- Einmalige Aufzeichnung
- Aufzeichnung im Ringpuffer (Endlos im Gerät)

Es können maximal 100 Ereignisse aufgezeichnet werden. Defaulteinstellung sind 10 Ereignisse.

Triggerbedingungen

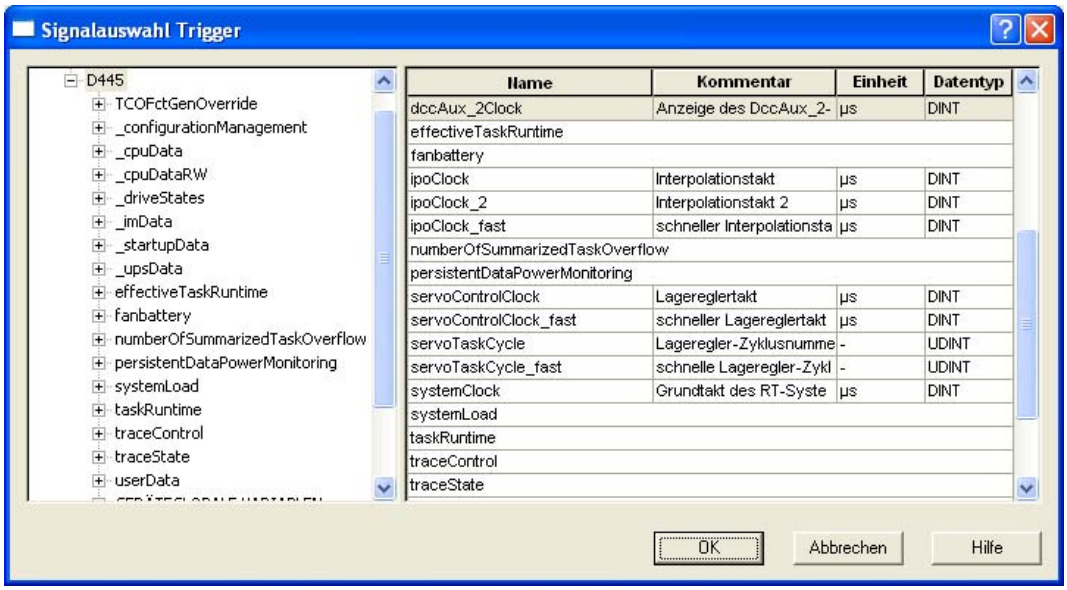


Bild 2-23 Signalauswahl Trigger

Zeitbegrenzter Trace:

- Aufzeichnung sofort
- Starttrigger auf Variable - Positive Flanke
- Starttrigger auf Variable - Negative Flanke
- Starttrigger auf Variable - Innerhalb eines Toleranzbandes
- Starttrigger auf Variable - Außerhalb eines Toleranzbandes
- Starttrigger an Codestelle
- Starttrigger durch Programmaufruf **Trace Trigger 1**
- Starttrigger durch Programmaufruf **Trace Trigger 2**

Endlos Trace:

- Aufzeichnung sofort
- Stoptrigger auf Variable - Positive Flanke
- Stoptrigger auf Variable - Negative Flanke
- Stoptrigger auf Variable - Innerhalb eines Toleranzbandes

- Stoptrigger auf Variable - Außerhalb eines Toleranzbandes
- Stoptrigger an Codestelle
- Stoptrigger durch Programmaufruf **Trace Trigger 1**
- Stoptrigger durch Programmaufruf **Trace Trigger 2**

2.6.5 Ereignisauswahl Technologieobjekt-Trace

Als Defaulteinstellung sind immer alle Befehle, Konfigurationsdaten und Systemvariablen angewählt. Auf den einzelnen Laschen können Signale abgewählt werden.

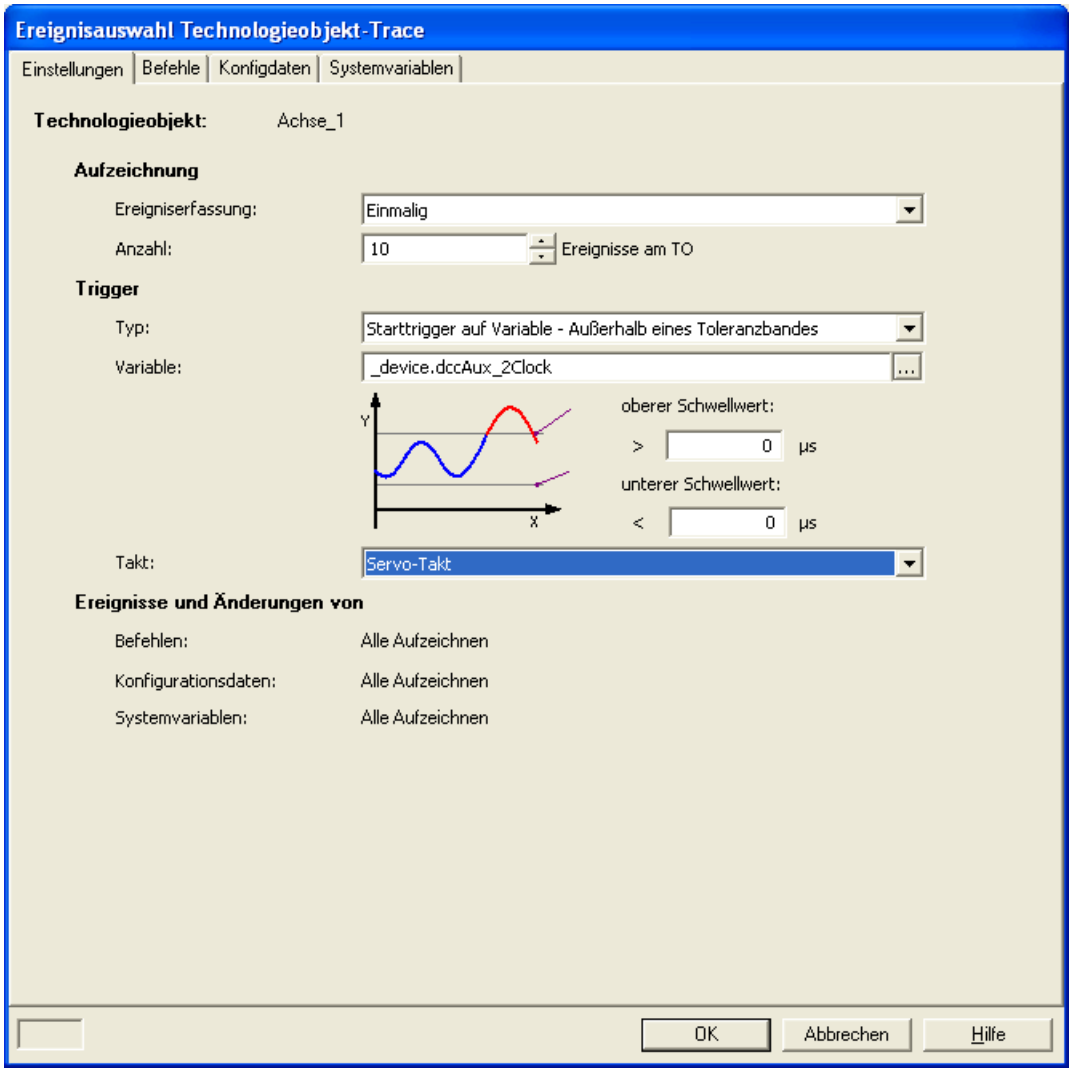


Bild 2-24 Ereignisauswahl Technologieobjekt-trace

Im Dialog **Ereignisauswahl Technologieobjekt-Trace** sehen Sie die Befehle des TO. Die Sortierung und Gruppierung entspricht der Darstellung wie in der Befehlsbibliothek.

Die Konfigurationsdaten und Systemvariablen werden ebenfalls in einer Baumstruktur mit Checkboxen dargestellt. Angezeigt wird per Default nur die 1. Ebene. Durch Ab/Anwählen der Checkbox eines Knotens wird der ganze darunterliegende Zweig ab/angewählt.

Symbolische Zuordnung (ab V4.2) 3

3.1 Symbolische Zuordnung - Einführung

Symbolische Zuordnung von Achse und Antrieb

SIMOTION unterstützt ab der Version V4.2 bei der Projektierung von Technologieobjekten (TO) und I/Os die symbolische Zuordnung auf SINAMICS Antriebsobjekte (DOs, Drive Objects).

Dadurch vereinfachen sich die Projektierung der technologischen Beziehungen einschließlich der Kommunikation zwischen Steuerung und Antrieb.

Durch die symbolische Zuordnung

- werden in einem Zuordnungsdialog nur passfähige Zuordnungspartner angeboten
- wird vom Engineeringsystem automatisch die Kommunikation zwischen Achse und Antrieb eingerichtet und die erforderlichen PROFIdrive Achstelegramme sowie die verwendeten Adressen eingerichtet
- werden abhängig von gewählter TO-Technologie (z. B. SINAMICS Safety Integrated) Telegramme erweitert und Zuordnungen im Antrieb automatisch angelegt
- können Achs- und Antriebsprojektierung zunächst unabhängig voneinander durchgeführt werden
- werden bei der Projektierung von I/O-Variablen auf SINAMICS I/Os automatisch die Kommunikationsverbindungen hergestellt (Telegramme und Adressen werden automatisch eingerichtet sowie die I/Os auf das Telegramm verschaltet).

Außer der symbolischen Zuordnung sind damit keine weiteren Projektierungen für die Kommunikation mehr erforderlich. Da keine Adressen mehr projektiert werden müssen, bleibt die Verbindung auch bei Adressverschiebungen bestehen.

Hinweis

Bei der Projektierung von Antriebsobjekten (DO Antrieb, DO Geber, ...) sowie im Dialog der Telegrammkonfiguration können Sie die **automatische Telegrammkonfiguration** und die **automatische Telegrammanpassung** deaktivieren.

Da Ihnen durch eine Deaktivierung viele der vorstehend genannten Vorteile verloren gehen, empfehlen wir eine Deaktivierung nur in begründeten Ausnahmefällen vorzunehmen.

Die symbolische Zuordnung wird vom TO Achse, TO Externer Geber sowie dem TO Nocken, TO Nockenspur und TO Messtaster unterstützt. Zudem können die Onboard I/Os einer SIMOTION D, einer SINAMICS S110/S120 Control Unit sowie vom TB30 und ausgewählten Terminal Modules symbolisch zugeordnet werden.

Folgende Komponenten unterstützen die symbolische Zuordnung:

Baugruppe	Unterstützt symbolische Zuordnung
SIMOTION C, P, D	ab SIMOTION V4.2
Controller Extension • CX32-2 • CX32	ab SIMOTION V4.2
SINAMICS S110 CU305	ab SINAMICS V4.3
SINAMICS S120 • CU310 • CU320-2 • CU320	 • ab SINAMICS V2.6.2 • ab SINAMICS V4.3 • ab SINAMICS V2.6.2
TB30, TM15 DI/DO, TM31	ab SIMOTION V4.2
TM41 (nicht DI/DO und AI)	ab SIMOTION V4.2
TM15, TM17 High Feature	ab SIMOTION V4.2

Hinweis

Die bisherige Methode der Antriebs- / Achs- und I/O-Projektierung steht weiterhin zur Verfügung. Hierzu muss die symbolische Zuordnung deaktiviert werden. Bei neu angelegten Projekten wird per Default die symbolische Zuordnung verwendet.

Werden Projekte auf V4.2 hochgerüstet, ist als Voreinstellung die symbolische Zuordnung deaktiviert und muss bei Bedarf aktiviert werden.

Die symbolische Zuordnung kann im SIMOTION SCOUT über das Menü "Projekt" > "Symbolische Zuordnung Verwenden" aktiviert/deaktiviert werden (siehe Symbolische Zuordnung aktivieren/deaktivieren (Seite 108)).

Adaption

Neben der symbolischen Zuordnung erleichtert ab SIMOTION V4.2 zusätzlich die automatische Adaption von SINAMICS S120 Daten die Projektierung. Im Hochlauf der SIMOTION Geräte werden Bezugsgrößen sowie Antriebs- und Geberdaten des SINAMICS S120 automatisch für die Konfigurationsdaten der SIMOTION Technologieobjekte "TO Achse" und "TO Externer Geber" übernommen. Diese Daten müssen in SIMOTION nicht mehr eingegeben werden

Weitere Informationen siehe Funktionshandbuch TO Achse elektrisch/hydraulisch, Externer Geber, unter Einstellung als Reale Achse mit digitaler Antriebskopplung).

3.2 Symbolische Zuordnung von TOs

3.2.1 Symbolische Zuordnung von Achse und Antrieb

Beschreibung

Im Achsassistenten zeigt der Zuordnungsdialog die verfügbaren Antriebe an, die symbolisch der Achse zugeordnet werden können.

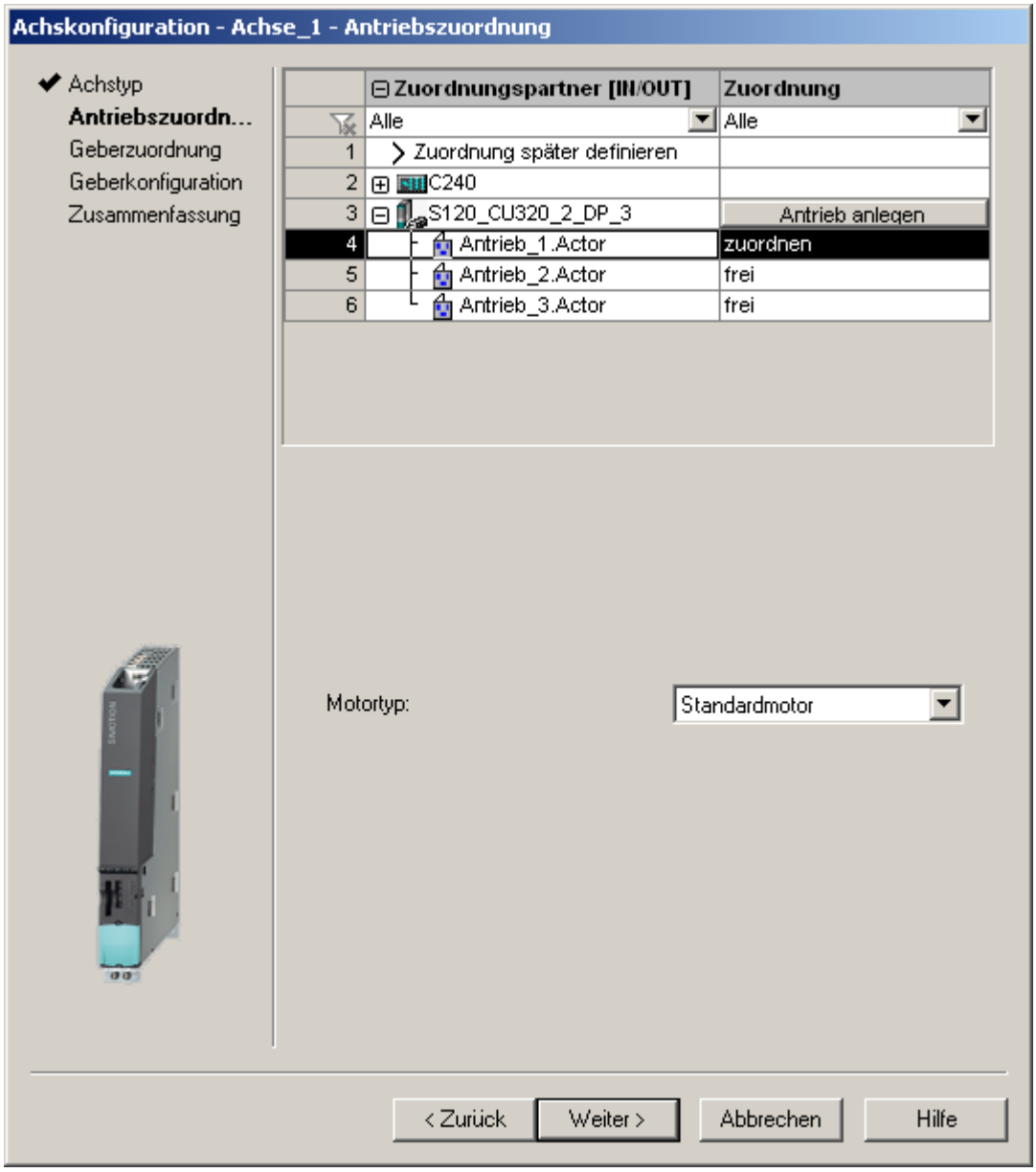


Bild 3-1 Achse und Antrieb zuordnen

Einstellmöglichkeiten

Es stehen folgende Einstellmöglichkeiten zur Verfügung:

- Antrieb zuordnen

  Zuordnen eines bereits projektierten Antriebes

- Später zuordnen

  Die Achse soll erst zu einem späteren Zeitpunkt einem Antrieb zugeordnet werden.

  Dadurch können

  – die PLC- und Motion-Control-Funktionen von einem Programmierer auch ohne Antriebs-Know-how unter Verwendung von Technologieobjekten (z. B. TO Achse) vollständig projektiert und in das Gerät geladen werden,

  – die Antriebe von einem Antriebs-Experten separat projektiert und optimiert werden und erst zu einem späteren Zeitpunkt die Technologieobjekte symbolisch über einen Verschaltungsdialog den Antriebsobjekten zugeordnet werden.

- Antrieb anlegen

  Aus dem Zuordnungsdialog kann hierüber direkt ein neuer Antrieb an einem vorhandenen Antriebsgerät (z. B. S120 CU320-2 oder SINAMICS Integrated) angelegt und der Achse zugeordnet werden. Damit kann die Achse inklusive Antrieb in einem Arbeitsgang angelegt werden. Das Konfigurieren eines Antriebs vor dem Anlegen einer Achse ist nicht erforderlich.

Geberzuordnung

Bei einer Positionierachse wird der Geber 1 am TO Achse mit angelegt (Motorgeber) und automatisch dem ersten Geber am Antrieb zugeordnet.

Falls am TO Achse Geber 2 (direkter Geber) angelegt ist, wird dieser dem 2. Geber der Antriebsregelung zugeordnet. Siehe auch Achsassistent Übersicht.

Nach dem Durchlauf des Achsassistenten wird die symbolische Antriebszuordnung folgendermaßen angezeigt:

- über "Konfiguration" der Achse sowie
- über die Adressliste (Ansicht alle Adressen)

Aus diesen Dialogen kann der Zuordnungsdialog über den Button "..." auch erneut aufgerufen werden.

Es ist ferner möglich, anstatt den Zuordnungsdialog aufzurufen, das Eingabefeld mit dem symbolischen Namen direkt zu editieren.

Technologiedatenblock (TDB) und Safety-Datenblock (SIDB)

Die Aktivierung des Technologiedatenblockes und der Unterstützung der SINAMICS Safety Integrated Extended Functions durch das TO kann im Dialog Konfiguration des TO Achse bei Funktionen unter dem Button "Ändern" eingestellt werden. Die Zuordnung erfolgt hier immer auf das Antriebs-DO des Aktors der Achse. Das System generiert automatisch eine Telegrammverlängerung und die BICO-Verschaltung der relevanten SINAMICS-Parameter.

Zuordnung der IO-Signale am TO Achse

Für die Zuordnung der I/O-Signale am TO Achse, wie z B. die Eingänge für den Referenznocken oder Hardware-Endschalter, wird der Zuordnungsdialog aus den Achsdialogen bzw. auch aus der Adressliste (Ansicht alle Adressen) über den Button "..." aufgerufen.

Den genauen Aufbau und die Bezeichner der Komponententypen der symbolischen Zuordnung können Sie unter Zuordnungstypen bei symbolischer Zuordnung (Seite 671) entnehmen.

Weitere Informationen zur Antriebszuordnung siehe Funktionshandbuch Achse.

Adressliste

In der Adressliste in der Ansicht "Adressen gesamt" erhalten Sie einen Überblick zu den Zuordnungen aller Schnittstellen des TO Achse. Aus dieser Ansicht können die Zuordnungen über den Zuordnungsdialog (Button ...) auch geändert werden.

Siehe auch

Symbolische Zuordnung von I/O-Variablen auf I/O-Klemmen (Seite 92)

Zuordnungsdialog benutzen (Seite 96)

Achsassistent Übersicht

Geber zuordnen

3.2.2 Symbolische Zuordnung von Externen Geber

Beschreibung

Der Zuordnungsdialog zeigt die verfügbaren Geber an, die symbolisch dem externen Geber zugeordnet werden können.

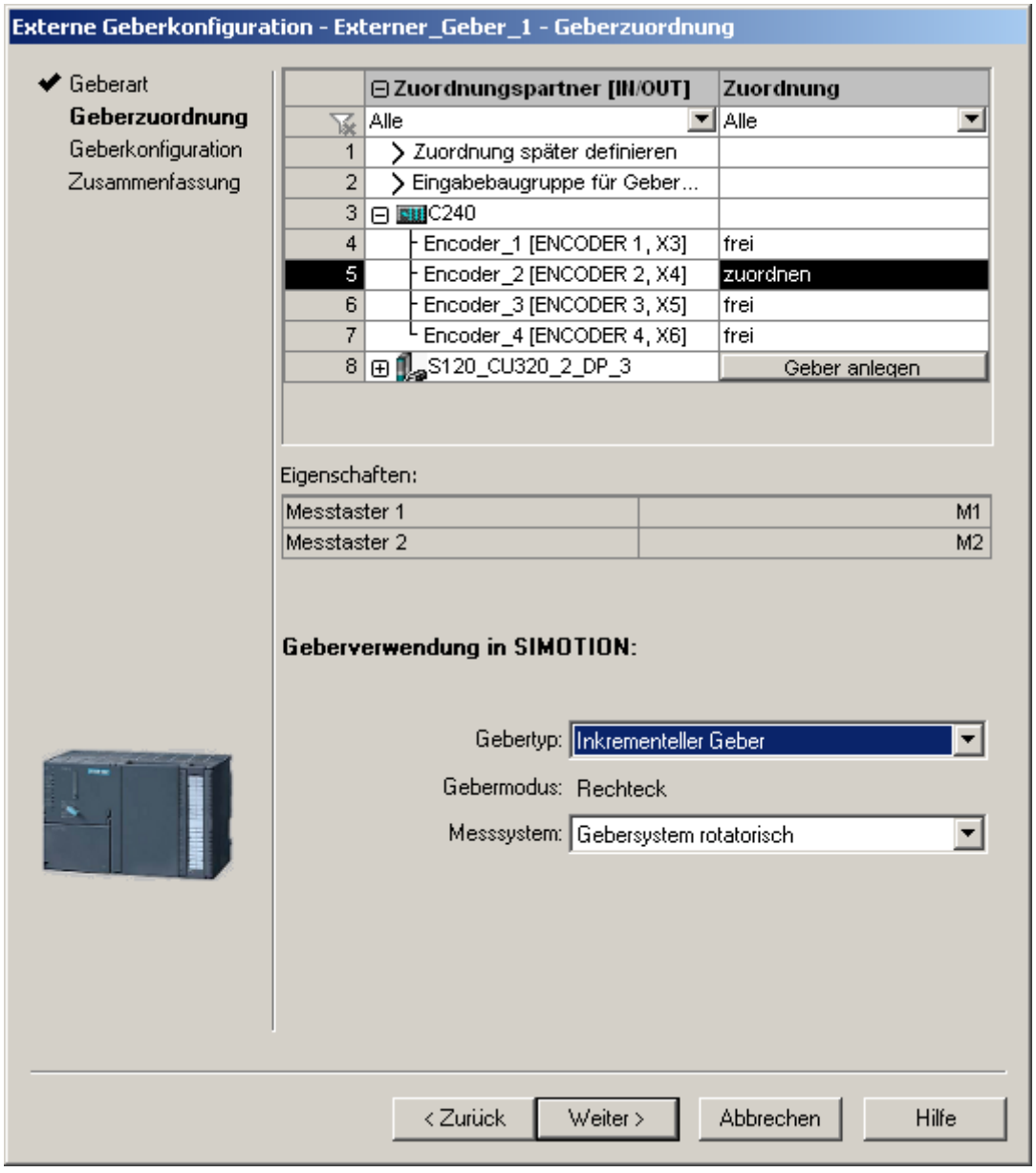


Bild 3-2 TO Externer Geber zuordnen

Für Informationen über den Achsassistenten, mit dem auch das DO Externer Geber konfiguriert wird, siehe Achsassistent Übersicht.

Siehe auch

Zuordnungsdialog benutzen (Seite 96)

3.2.3 Symbolische Zuordnung von Nocken, Nockenspur und Messtaster

Beschreibung

Vor der Konfiguration des Technologieobjektes muss die Funktionalität der Klemmen entsprechend projektiert werden (z. B Projektierung eines DI/DO als Digitalausgang).

Die Projektierung des Zugriffs der Technologieobjekte auf I/Os erfolgt symbolisch, wobei nur "funktionskompatible" I/O-Kanäle im Zuordnungsdialog zur Auswahl angeboten werden.

Beispiel:

Beim TO Messtaster werden nur symbolische Zuordnungen vom Typ MI (Measuring Input = Messtastereingang) zur Auswahl angeboten.

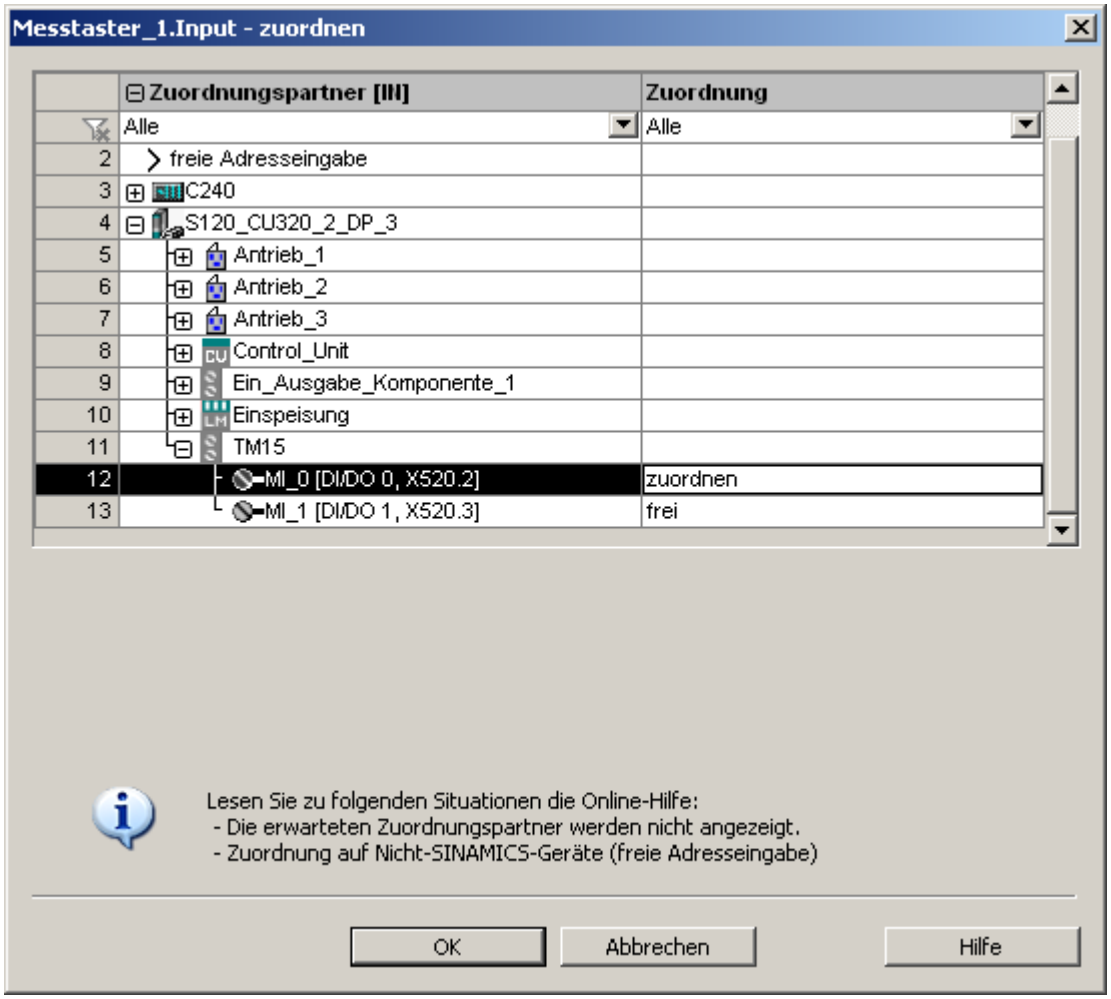


Bild 3-3 Messtastereingang zuordnen

Weitere Informationen siehe Funktionshandbuch Nocken und Messtaster:

- TO Nocken zuordnen
- TO Nockenspur zuordnen
- TO Messtaster zuordnen

Für Informationen zum Zuordnungsdialog, siehe Zuordnungsdialog benutzen (Seite 96).

3.2.4 Symbolische Zuordnung von TO Sensor

Beschreibung

Vor der Zuordnung muss ggf. die Funktionalität der Klemmen entsprechend projektiert werden.

Im Zuordnungsdialog werden für das TO Sensor die analogen Eingänge angeboten.

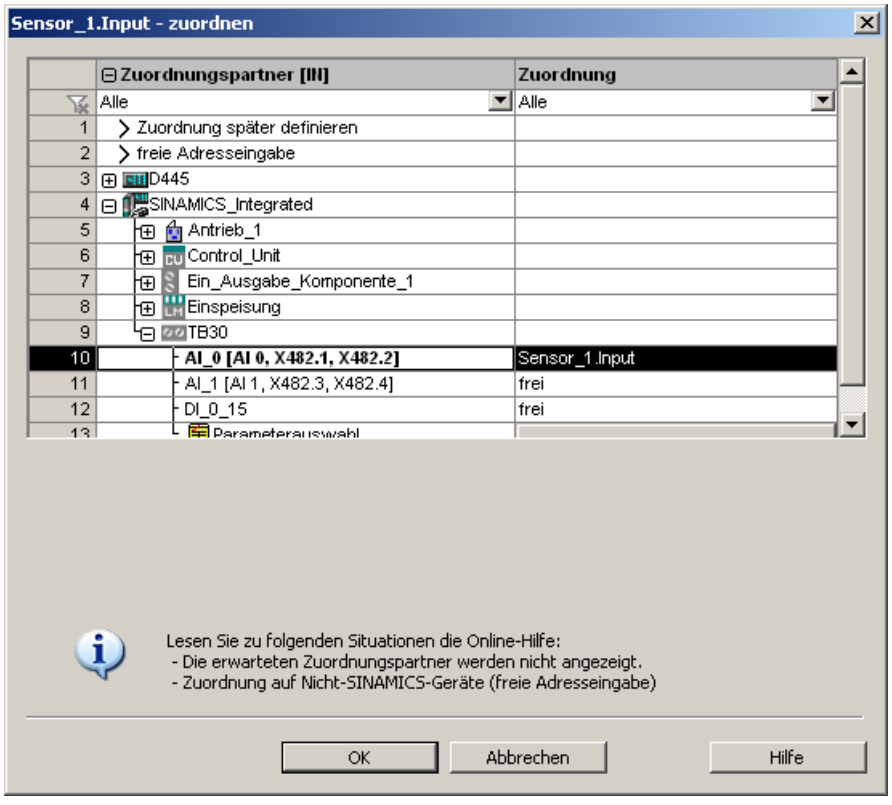


Bild 3-4 TO Sensor zuordnen

Für eine Beschreibung des TO Sensor, siehe Funktionshandbuch Ergänzende Technologieobjekte im Abschnitt TO Sensor.

Eine Beschreibung des Zuordnungsdialogs finden Sie unter Zuordnungsdialog benutzen (Seite 96).

3.3 Symbolische Zuordnung von I/O-Variablen

3.3.1 Symbolische Zuordnung von I/O-Variablen auf das PROFIdrive Telegramm des TO Achse

Beschreibung

I/O-Variablen, die Sie z. B. zu Anzeige- und Diagnosezwecken benötigen, können Sie aus der Adressliste über den Zuordnungsdialog einzelnen Komponenten (z. B. Zustandswort) des PROFIdrive Telegramms zuordnen. Es werden nur die zum Datentyp der I/O-Variable passenden Komponenten angezeigt. Wird kein Datentyp an der I/O-Variable angegeben, wird dieser nach der Auswahl über den Zuordnungspartner bestimmt.

So ordnen Sie eine I/O-Variable einer einzelnen Komponente eines PROFIdrive-Telegramms zu

1. Öffnen Sie die Adressliste und legen Sie eine Variable an.
2. Geben Sie bei der Konfiguration der I/O-Variablen unter IO-Adresse für die Richtung den Bezeichner IN oder OUT ein (oder Auswahl über Klappliste).
3. Wählen Sie den Datentyp und öffnen Sie den Zuordnungsdialog über den Button "...".

   Im Zuordnungsdialog werden dann nur Zuordnungsziele angezeigt, die für diesen Datentyp passen. Wird kein Datentyp ausgewählt, dann werden alle Parameter angeboten. Der Zuordnungsdialog liefert beim Schließen den Datentyp des Parameters zurück.

4. Wählen Sie unter Zuordnungspartner die Komponente, z. B. das Wort aus, das Sie zuordnen möchten.
5. Klicken Sie auf **OK**, um das Komponente zuzuordnen.

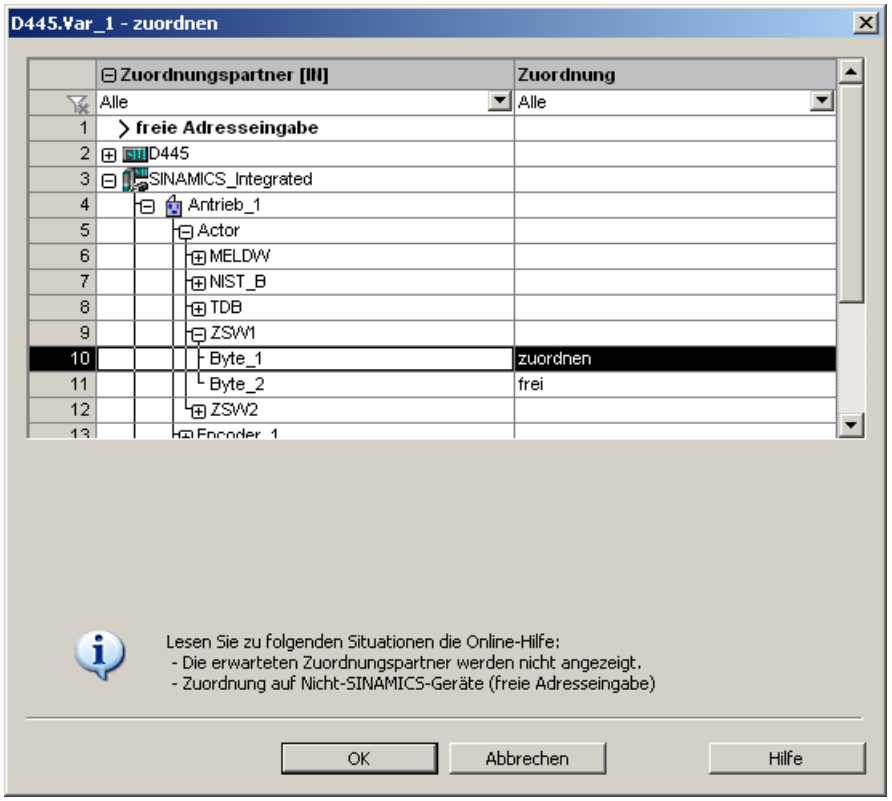


Bild 3-5 I/O-Variable dem PROFIdrive Telegramm von TO Achse zuordnen

Den genauen Aufbau und die Bezeichner der Komponententypen der symbolischen Zuordnung können Sie unter Zuordnungstypen in Standardtelegrammen (Seite 676) nachschlagen.

3.3.2 Symbolische Zuordnung von I/O-Variablen auf Antriebsparameter

Beschreibung

I/O-Variablen können Sie aus der Adressliste über den Zuordnungsdialog auch auf Antriebsparameter zuordnen. Es werden nur die zum Datentyp der I/O-Variable passenden Parameter angezeigt. Wenn kein Datentyp an der I/O-Variable angegeben, wird dieser nach der Parameter-Auswahl bestimmt.

Durch das System erfolgt, für die Übertragung der Parameter, automatisch eine Verlängerung des Standardtelegramms zum Antrieb.

BICO-Verschaltung

Sie können einen BICO-Parameter auf ein Telegramm verschalten, indem Sie die Parameternummern, die Übertragungsbreite (WORD, DWORD) und die Übertragungsrichtung im Zuordnungsdialog vorgeben.

Dabei kann der Parameter auch von einer anderen Signalquelle (SINAMICS-DO) selektiert werden. Der Parameter wird anschließend automatisch mit dem Telegramm des Zuordnungspartners übertragen.

So ordnen Sie I/O-Variablen Antriebsparametern zu

1. Öffnen Sie die Adressliste.
2. Geben Sie bei der Konfiguration der I/O-Variablen unter IO-Adresse für die Richtung den Bezeichner IN oder OUT ein (oder Auswahl über Klappliste).
3. Wählen Sie den Datentyp und öffnen Sie den Zuordnungsdialog über den Button "...".

   Im Zuordnungsdialog werden dann nur Zuordnungsziele angezeigt, die für diesen

   Datentyp passen. Wird kein Datentyp ausgewählt, dann werden alle Parameter angeboten. Der Zuordnungsdialog liefert beim Schließen den Datentyp des Parameters zurück.

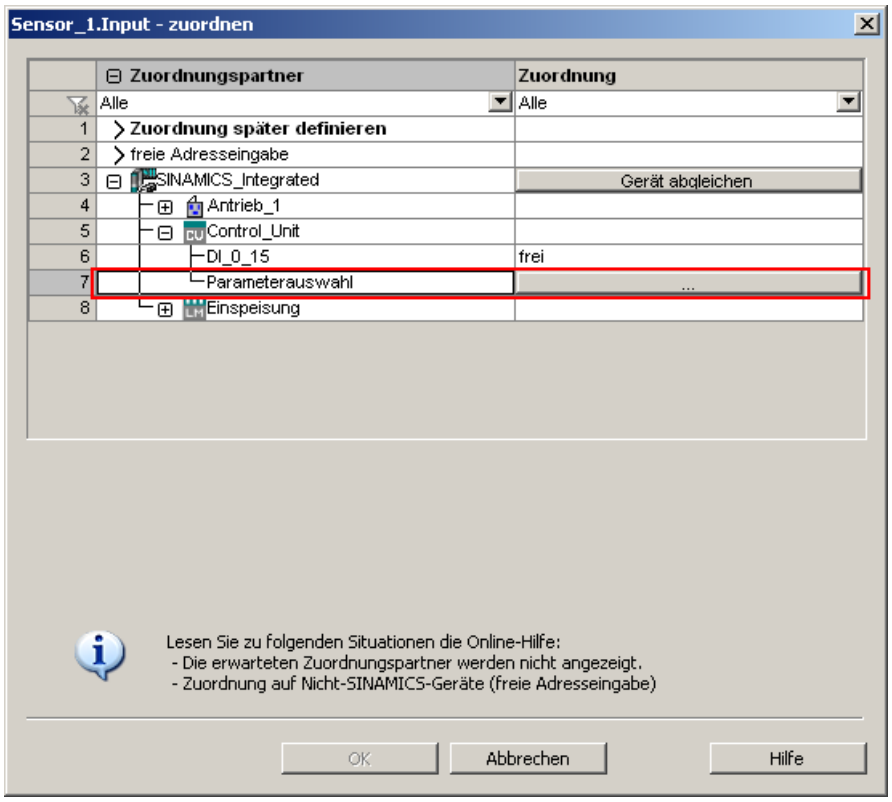


Bild 3-6 Zuordnungsdialog für Antriebsparameter

4. Klicken Sie in der Zeile Parameterauswahl auf die Schaltfläche ..., um die Parameterliste für zu öffnen.

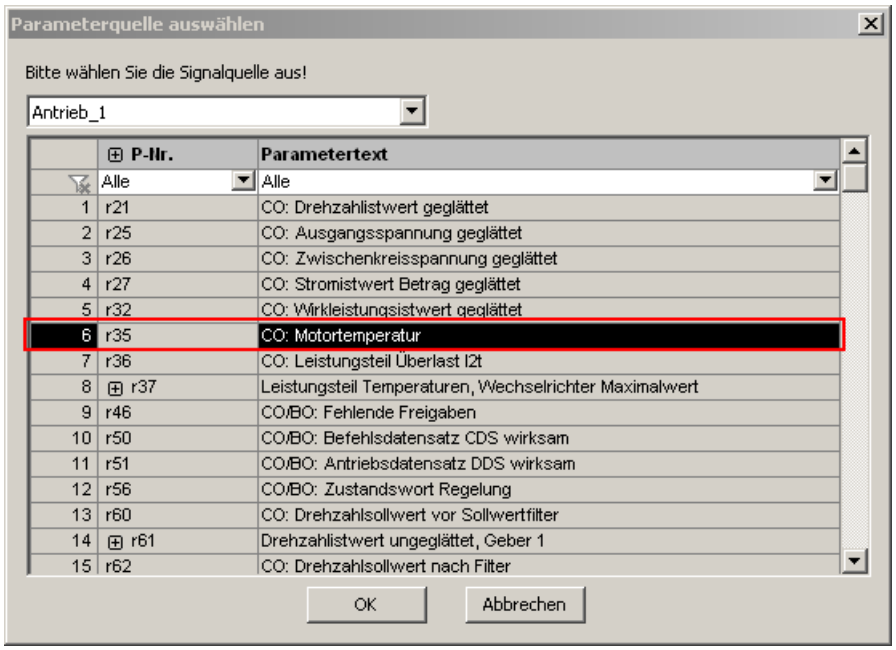


Bild 3-7 Dialog zur Parameter- und DO-Auswahl

5. Wählen Sie ggf. die Signalquelle (SINAMICS-DO) aus, das den Parameter zur Verfügung stellt, und dann den Parameter.
6. Klicken Sie auf **OK**, um die Auswahl zu übernehmen.

7. Der I/O-Variablen im Zuordnungsdialog wird ein SINAMICS-Parameter zugeordnet.

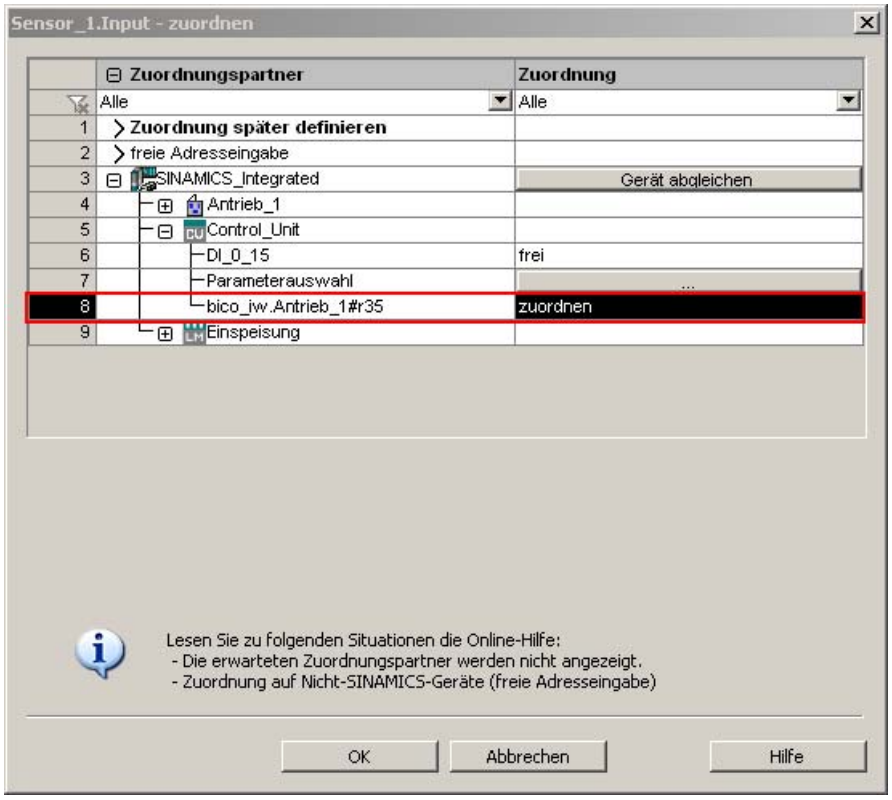


Bild 3-8 Zugeordneter Antriebsparameter

8. Klicken Sie auf **OK**, um den Zuordnung zu übernehmen.

Zuordnungstypen

Die folgende Tabelle zeigt die möglichen Typen der Zuordnung:

Name der BICO-Schnittstelle	Datentyp	Richtung	Übertragbare BICO-Parameter
BICO_IW.<Parameternummer>	WORD	Input	Alle CO-Parameter (BICO-Quelle)
BICO_QW.<Parameternummer>	WORD	Output	Alle CI-Parameter (BICO-Senke)
BICO_ID.<Parameternummer>	DWORD	Input	Alle CO-Parameter (BICO-Quelle)
BICO_QD.<Parameternummer>	DWORD	Output	Alle CI-Parameter (BICO-Senke)

Syntax der Zuordnungsnamen:

- Bei Ausgängen (SINAMICS-Seite = empfangene Daten), die auf mehrere BICO-Senken verschaltet werden können, werden mehrere Parameter durch einen Punkt getrennt angegeben.
- Liegt der übertragene Parameter auf einem anderen DO, so wird der DO-Name dem Parameter vorangestellt. Als Trennzeichen zwischen DO-Name und Parameter wird das "#" verwendet.
- Einzeln übertragene Bits eines Parameters werden in Klammern [x] dargestellt.

3.3.3 Symbolische Zuordnung von I/O-Variablen auf I/O-Klemmen

Beschreibung

Vor der Zuordnung muss ggf. die Funktionalität der Klemmen entsprechend projektiert werden (z. B. Projektierung eines DI/DO als Digitalausgang).

Sie haben dann zwei Möglichkeiten eine IO-Variable Klemmen zuzuordnen:

- Zuordnung eines Klemmensignals über die SIMOTION Vorzugsverschaltung. Dazu müssen Sie für die entsprechenden Ein-/Ausgänge des SINAMICS DOs die SIMOTION Vorzugsverschaltung verwenden. Die BICO-Verschaltung wird automatisch durchgeführt.
- Zuordnung eines Klemmensignals über Schnittstellen, z. B. DI_0_15 bzw. DO_0_15. Die Schnittstellen sind z. B. dem PZD 2 (A_DIGITAL bzw. E_DIGITAL) zugeordnet. Für diese Signale wird zwar ein Telegramm entsprechender Länge erzeugt, die BICO-Verschaltung muss aber noch durchgeführt werden, z. B. bei TB30, TM31 oder TM15DIDO.

So gehen Sie bei der symbolischen Zuordnung mit SIMOTION Vorzugsverschaltung vor:

1. Öffnen Sie die Adressliste.
2. Geben Sie bei der Konfiguration der I/O-Variablen unter IO-Adresse für die Richtung den Bezeichner IN oder OUT ein (oder Auswahl über Klappliste), wählen Sie den Datentyp und öffnen Sie den Zuordnungsdialog über den Button "...".

3. Im Zuordnungsdialog werden dann nur Zuordnungsziele angezeigt, die für diesen Datentyp passen.

   Wird kein Datentyp ausgewählt, dann werden alle bekannten Zuordnungspartner (z. B. Bit, Word, DWord, Ein- Ausgangsklemmen) angeboten. Der Zuordnungsdialog liefert beim Schließen den Datentyp des Zuordnungspartners zurück.

4. Wählen Sie das entsprechende Klemmensignal aus und klicken Sie auf **OK**.

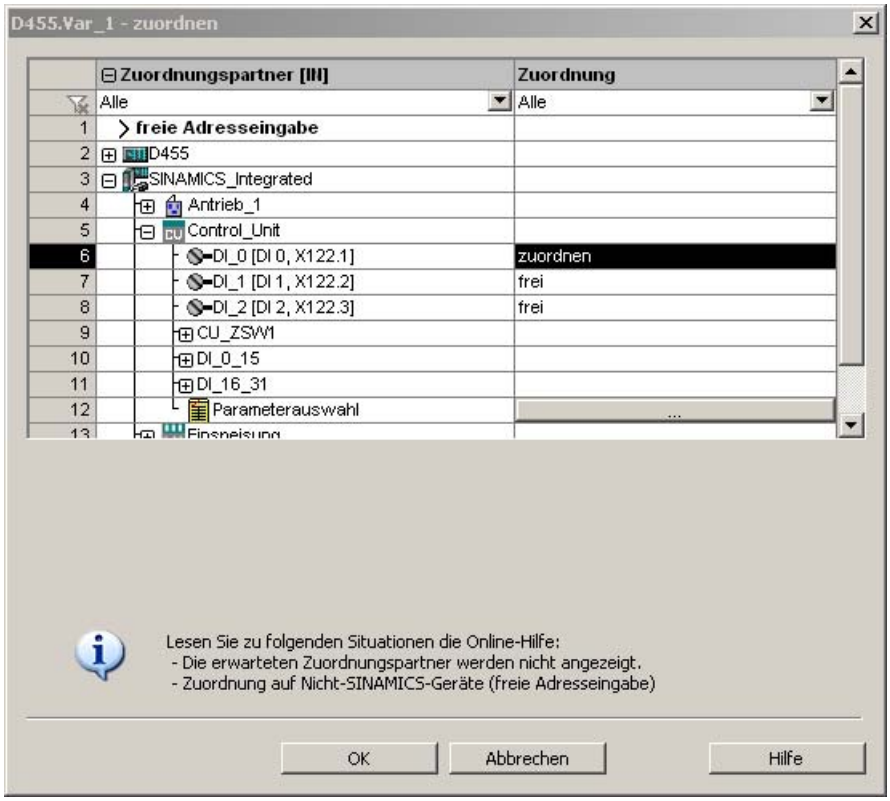


Bild 3-9 Zuordnen einer I/O-Variable

So gehen Sie bei der symbolischen Zuordnung über PZDs von Telegrammen vor:

1. Öffnen Sie die Adressliste.
2. Geben Sie bei der Konfiguration der I/O-Variablen unter IO-Adresse für die Richtung den Bezeichner IN oder OUT ein (oder Auswahl über Klappliste), wählen Sie den Datentyp und öffnen Sie den Zuordnungsdialog über den Button "...".

3. Wählen Sie die Schnittstelle aus, der Sie eine I/O-Variable zuordnen möchten, z. B. DI_0_15.

   Sie können die Variable auch selektiv einem Bit zuordnen.

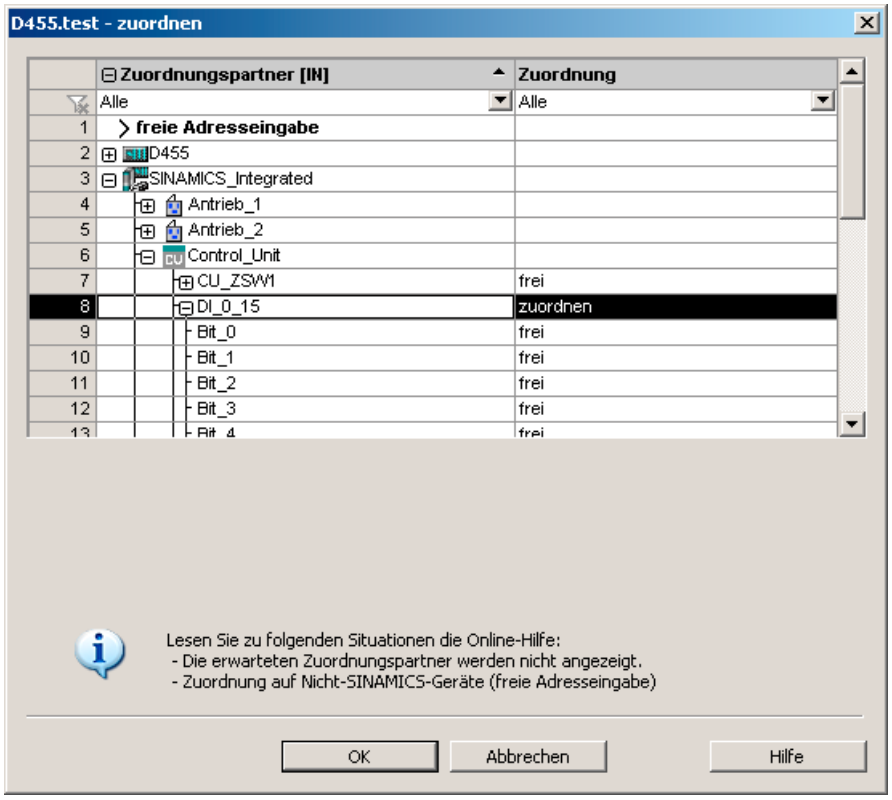


Bild 3-10 Zuordnungsdialog für PZDs

4. Klicken Sie auf **OK**, um die Zuordnung zu übernehmen.
5. Wechseln Sie nun in den Dialog **Kommunikation** des entsprechenden SINAMICS DOs. Sie sehen dort die einzelnen Bits des PZDs (z. B. E_Digital oder A_Digital) aufgelistet.
6. Verschalten Sie das entsprechende Bit des PZDs mit einem Signal, z. B. **Externe Warnung wirksam**. Damit haben Sie die BICO-Verschaltung durchgeführt.

Hinweis

Wenn Sie die Symbolische Zuordnung aktiviert haben, werden für die SINAMICS Control Units die Telegramme 39x automatisch zugeordnet (mindestens Telegramm 390). Die Folgeverschaltung der Klemmen wird dabei nicht gemäß der ursprünglichen Definition der Telegramme (ohne Symbolische Verschaltung) durchgeführt, da mit der Symbolischen Verschaltung die Klemmenkonfiguration durch Sie vorgegeben und von der Symbolischen Verschaltung verwaltet wird.

Zuordnung von I/O-Variablen mit Ersatzwerten

Für I/O-Variable vom Datentyp Bool können keine Ersatzwerte angegeben werden. Wenn Sie dennoch Ersatzwerte benötigen, können Sie wie folgt vorgehen:

1. Ordnen Sie eine Variable vom Typ Bool z. B. der Aktor-Schnittstelle eines Antriebs zu,

   Antrieb_1.Control_Unit.DI_0_15
2. Legen Sie eine übergreifende Variable an (mindestens Datentyp Word).
3. Ordnen Sie der Variable im Zuordnungsdialog die Aktor-Schnittstelle des Antriebs zu,

   Antrieb_1.Control_Unit.MELDW.

   Bit1 des Ersatzwertes muss dann den Ersatzwert für die Bool-Variable enthalten.

   Analog können Sie einen Ersatzwert einem BICO-Parameter zuordnen.

Für verschiedene SINAMICS-DO sind extra übergeordnete Schnittstellen zur Zuordnung von Ersatzwerten eingefügt, siehe SINAMICS-Zuordnungstypen (Seite 674).

Siehe auch

Symbolische Zuordnung von Achse und Antrieb (Seite 81)

Antriebsperipherie symbolisch zuordnen (Seite 307)

3.4 Mit dem Zuordnungsdialog arbeiten

3.4.1 Zuordnungsdialog benutzen

Beschreibung

Über den Zuordnungsdialog kann die Zuordnung eines SIMOTION Objektes zu einem SINAMICS Objekt erfolgen.

Den Zuordnungsdialog können Sie aus verschiedenen TO-Konfigurationsdialogen und der Adressliste aufrufen. Die Darstellung des Dialogs ist davon abhängig, für welche Zuordnung Sie den Dialog aufrufen, z. B. ob Sie einen Eingang, Ausgang oder einen Aktor/Geber zuordnen möchten.

Signal bzw. Zuordnungstyp wird nicht gefunden

Im Zuordnungsdialog werden nur die Zuordnungstypen angezeigt, die für den SIMOTION-Typ passen. Wenn Sie einen anderen Zuordnungstyp erwarten, kontrollieren Sie:

- Den Datentyp des Zuordnungstyps (z. B. bei I/O-Variablen)
- Die Konfiguration der SINAMICS-DO (Ein-/Ausgang)
- Ob die symbolische Zuordnung eingeschaltet ist (**Projekt > Symbolische Zuordnung**)

Aufbau des Zuordnungsdialogs mit aktivierter symbolischer Zuordnung

Der Zuordnungsdialog mit aktivierter symbolischer Zuordnung ist im Achsassistent, in der Adressliste und in TO Konfigurationsmasken verfügbar.

Der Zuordnungsdialog ist folgendermaßen aufgebaut:

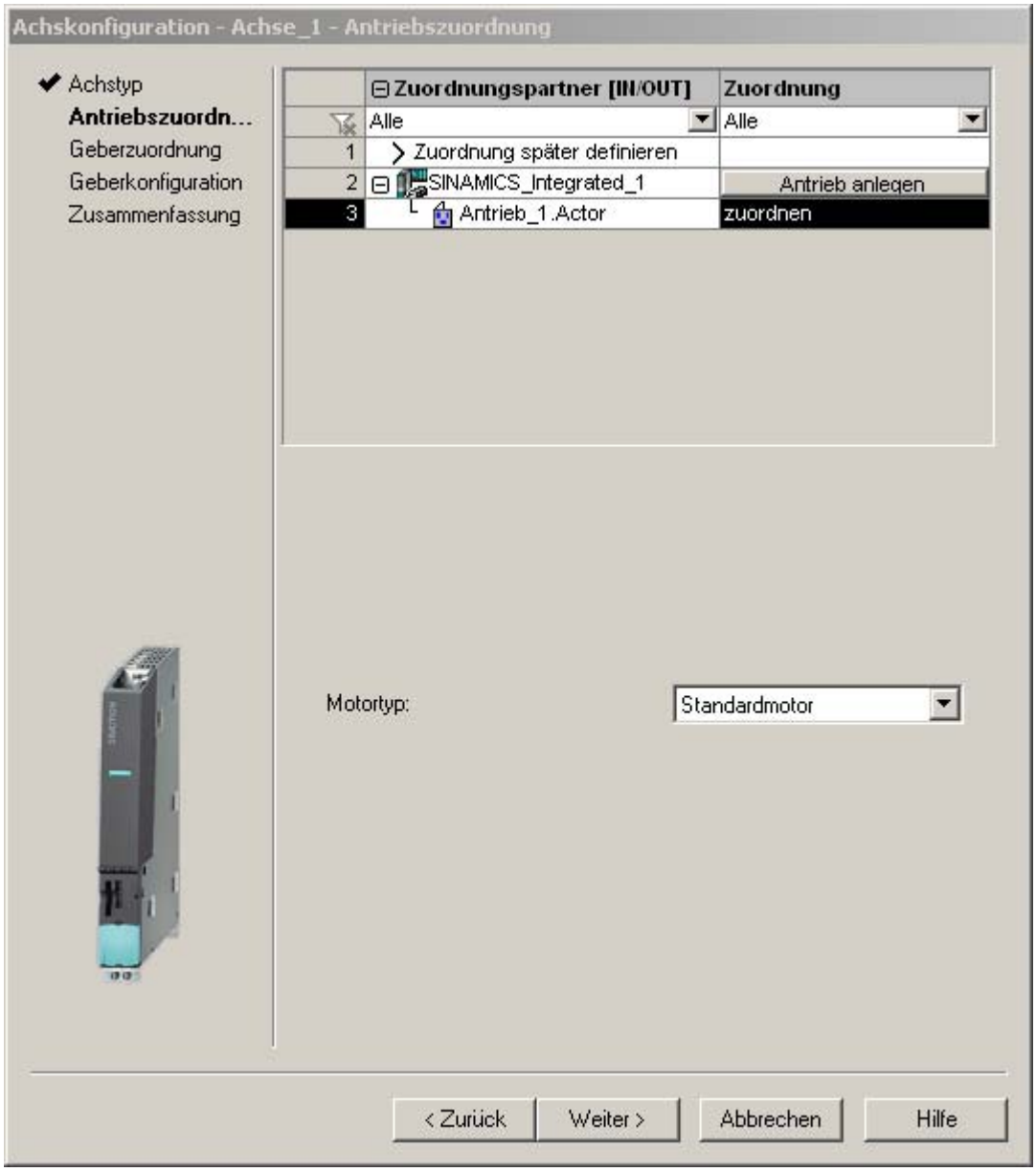


Bild 3-11 Zuordnungsdialog mit aktivierter symbolischer Zuordnung

Feld		Beschreibung
Titelleiste		In der Titelleiste sehen Sie den Namen der SIMOTION-Schnittstelle oder I/O-Variable, die verschaltet werden soll. Der Dialog zeigt nur die Zuordnungspartner an, die für diese Schnittstelle geeignet sind.
Filterzeile		In der Filterzeile können Sie die Anzeige der Tabelle nach den Kriterien **alle**, **frei**, **zugeordnet** filtern.
Zuordnungspartner		In dieser Spalte sehen Sie alle Geräte und DO aufgelistet. Durch Klicken auf das "+"-Zeichen können Sie die Gruppen erweitern und so die geeigneten Schnittstellen anzeigen, z. B. unter Control Unit. Hinweis: Wenn Sie eine I/O-Variable einer I/O-Klemme von SINAMICS-Baugruppen zuordnen, müssen Sie bei Verwendung der Schnittstellen DI_0_15 oder DO_0_15 (Bezeichnung kann je nach Anzahl der Klemmen variieren) noch die BICO-Verschaltung im Antriebs-DO durchführen, siehe auch Symbolische Zuordnung von I/O-Variablen auf I/O-Klemmen (Seite 92).
	Zuordnung später definieren	Wählen Sie **später zuordnen** aus, wenn Sie die SIMOTION-Schnittstelle zu einem späteren Zeitpunkt zuordnen möchten.
	Freie Adresseingabe	Wählen Sie diesen Eintrag aus, wenn Sie die Adresse des Zuordnungspartners selbst eingeben müssen/möchten (z. B. für Geräte, die nicht symbolisch zugeordnet werden, Hydraulikachsen oder analoge Baugruppen). Die Adresse müssen Sie dann in HW Konfig auslesen und hier eintragen.
	Parameterauswahl	Unter Parameterauswahl können Sie Parameter des SINAMICS-DO auswählen. Siehe auch Symbolische Zuordnung von I/O-Variablen auf Antriebsparameter (Seite 88).
Zuordnung		In dieser Spalte wird angezeigt, ob die SIMOTION-Schnittstelle bereits zugeordnet ist: • Frei oder leer, Schnittstelle ist noch nicht zugeordnet. • Zuordnen; Schnittstelle wird dem ausgewählten Zuordnungspartner zugeordnet • <zugeordnete Schnittstelle>; Schnittstelle ist bereits zugeordnet. Es wird die zugeordnete SIMOTION-Schnittstelle angezeigt.
Antrieb anlegen (Achsassistent - Antrieb zuordnen)		Klicken Sie auf die Schaltfläche, um den Antriebsassistent aufzurufen, mit dem Sie einen SINAMICS Antrieb anlegen und konfigurieren können.
Geber anlegen (Achsassistent - Geber zuordnen)		Klicken Sie auf die Schaltfläche, um den Assistent zur Konfiguration des Geber DO aufzurufen.

Aufbau des Zuordnungsdialogs mit deaktivierter symbolischer Zuordnung

Der Zuordnungsdialog mit deaktivierter symbolischer Zuordnung ist nur im Achsassistent verfügbar (Antriebszuordnung und Geberzuordnung).

Der Zuordnungsdialog ist folgendermaßen aufgebaut:

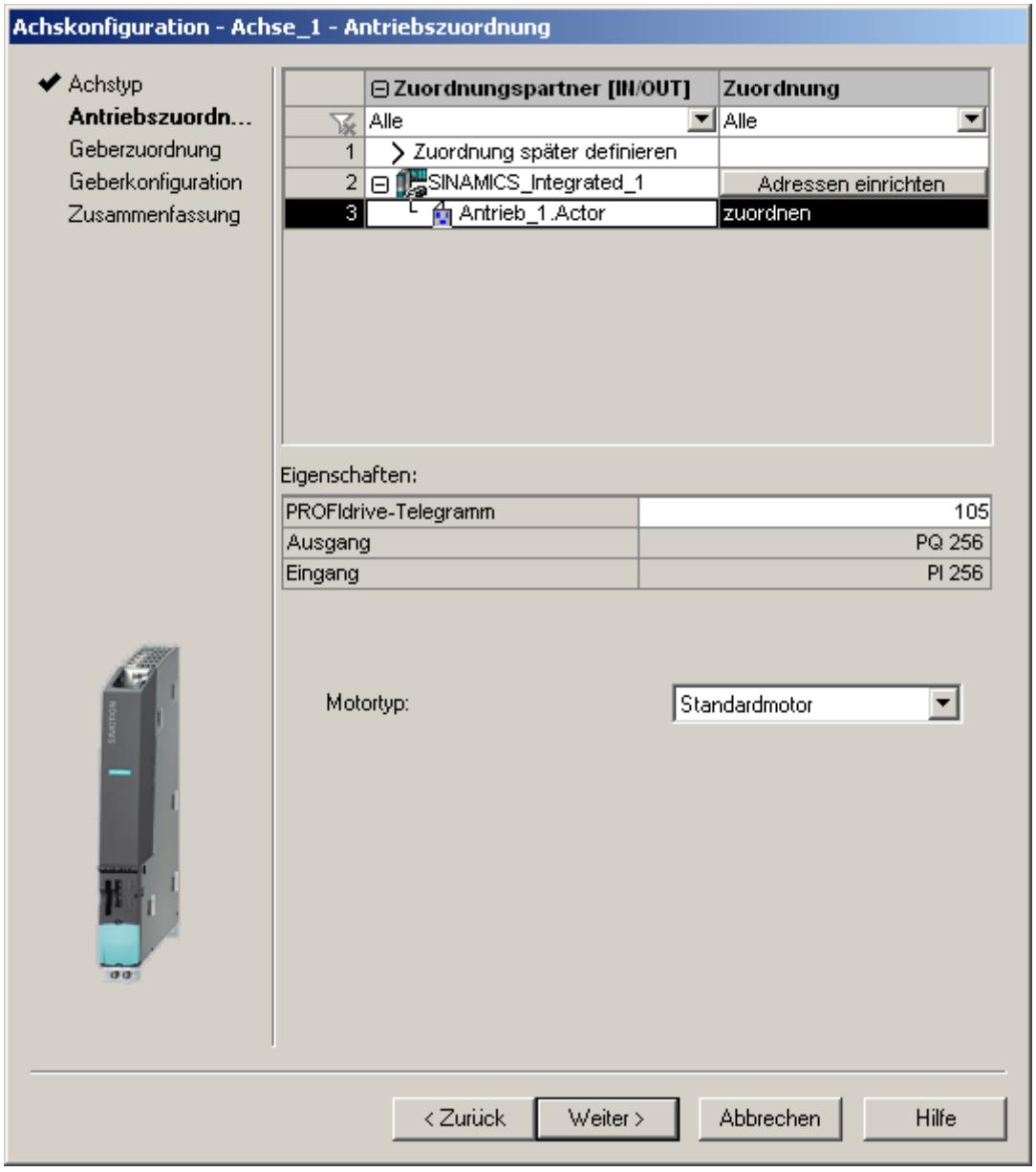


Bild 3-12 Zuordnungsdialog

Feld		Beschreibung
Titelleiste		In der Titelleiste sehen Sie den Namen der SIMOTION-Schnittstelle oder I/O-Variable, die verschaltet werden soll. Der Dialog zeigt nur die Zuordnungspartner an, die für diese Schnittstelle geeignet sind.
Filterzeile		In der Filterzeile können Sie die Anzeige der Tabelle nach den Kriterien **alle**, **frei**, **zugeordnet** filtern.
Zuordnungspartner		In dieser Spalte sehen Sie alle Geräte und DO aufgelistet. Durch Klicken auf das "+"-Zeichen können Sie die Gruppen erweitern und so die geeigneten Schnittstellen anzeigen, z. B. unter Control Unit.
	Freie Adresseingabe	Wählen Sie diesen Eintrag aus, wenn Sie die Adresse des Zuordnungspartners selbst eingeben müssen/möchten (z. B. für Geräte, die nicht symbolisch zugeordnet werden oder Hydraulikachsen). Die Adresse müssen Sie dann in HW Konfig auslesen und hier eintragen.
	Parameterauswahl	Unter Parameterauswahl können Sie Parameter des SINAMICS-DO auswählen.
Zuordnung		In dieser Spalte wird angezeigt, ob die SIMOTION-Schnittstelle bereits zugeordnet ist: • Frei oder leer, Schnittstelle ist noch nicht zugeordnet. • Zuordnen; Schnittstelle wird dem ausgewählten Zuordnungspartner zugeordnet • <zugeordnete Schnittstelle>; Schnittstelle ist bereits zugeordnet. Es wird die zugeordnete SIMOTION-Schnittstelle angezeigt.
	Adresse einrichten	Mit dieser Schaltfläche können Sie die Adressen, Telegramme etc. von Geräten, die nicht symbolisch zugeordnet werden, nach HW Konfig übertragen.
Eigenschaften		In dieser Tabelle können Sie Einstellungen zum Antrieb bzw. Geber eingeben.

Zuordnungsdialog bei deaktivierter symbolischer Zuordnung bzw. bei Antrieben welche die symbolische Zuordnung nicht unterstützten

Die Antriebe sind erst dann vollständig sichtbar, wenn sie "Adressen einrichten" (über HW-Konfig) ausgeführt haben. Dies betrifft z.B. die SIMODRIVE- oder MASTERDRIVES-Antriebe bzw. die PROFIBUS-Baugruppen ADI4 oder IM174. Siehe auch .

Freie Adresseingabe

Neben der symbolischen Zuordnung können Sie unter "freie Adresseingabe" auch direkt eine Hardware-Adresse eingeben. Dies ist z. B. notwendig, wenn die Hardwarekomponente die symbolische Zuordnung nicht unterstützt.

Die Eingabe der Adresse in das Textfeld wird überprüft und ist abhängig vom Datentyp, der

beim Aufruf des Dialogs vorgegeben wurde.

- Wurde kein Datentyp beim Aufruf vorgegeben, wird die korrekte Syntax der Adresse überprüft und über die Adresse der Datentyp ermittelt. Dieser wird dann z. B. in der Adressliste eingetragen.
- Für Eingangsvariablen kann nicht die Adresse einer Ausgangsvariablen eingegeben werden und umgekehrt.

3.5 Adressen und Telegramme einrichten

3.5.1 Adressen und Telegramme einrichten - Übersicht

Übersicht

Nachdem alle SINAMICS Komponenten konfiguriert wurden, müssen die Adressen für den Prozessdatenaustausch zwischen Antrieb und Steuerung ermittelt werden.

Das Vorgehen hängt dabei davon ab, ob **symbolische Zuordnungen** verwendet werden.

- mit **symbolischer Zuordnung** werden die Adressen automatisch vom Engineeringsystem ermittelt, siehe Abschnitt Kommunikation für symbolische Zuordnung einrichten (Seite 101).
- ohne **symbolische Zuordnung** muss die Ermittlung der Adressen manuell angestoßen werden, siehe hierzu Abschnitt Telegrammkonfiguration (Seite 102).

Siehe auch

Azyklische Kommunikation mit dem Antrieb (Seite 477)

3.5.2 Kommunikation für symbolische Zuordnung einrichten

Durch folgende Aktionen wird die Kommunikation für die symbolische Zuordnung eingerichtet:

- über SCOUT-Menü (rufen Sie im Menü: **Projekt > Kommunikation für symbolische Zuordnung einrichten** auf)
- beim Laden ins Zielsystem
- beim Projekt speichern und Änderungen übersetzen

Beim Einrichten der Kommunikation werden die Telegramme, BICO-Verschaltungen und Adressen für das gesamte Projekt eingerichtet.

Siehe auch

Adressen und Telegramme einrichten - Übersicht (Seite 101)

Telegrammkonfiguration (Seite 102)

Rückrüsten auf Versionen <V4.2 (Seite 111)

3.5.3 Telegrammkonfiguration

Voraussetzung

Sie haben das Antriebsgerät konfiguriert.

Hinweis

Diese Schritte müssen Sie nur dann ausführen, wenn die symbolische Zuordnung nicht aktiv ist oder nicht verwendet werden kann, wie z. B. bei Geräten einer Version < V4.2.

Auf Basis dieser Konfiguration soll nun eine/mehrere der nachfolgend aufgeführten Aktionen durchgeführt werden:

- die automatische PROFIdrive-Telegrammeinstellung für ein Antriebsobjekt soll aktiviert/deaktiviert werden,
- die automatische Telegrammverlängerung für ein Antriebsobjekt soll aktiviert/deaktiviert werden,
- die automatische Adressanpassung für ein Antriebsobjekt soll aktiviert/deaktiviert werden,
- es sollen PROFIdrive-Telegramme für Antriebsobjekte konfiguriert werden,
- die Adressen sollen eingerichtet werden,
- Telegramme sollen manuell verlängert werden.

Vorgehensweise

Gehen Sie dazu wie folgt vor:

- Öffnen Sie im Projektnavigator den Eintrag des SINAMICS Antriebsgerätes, z. B. eines externen S120, oder einer SIMOTION D.
- Öffnen Sie im Projektnavigator unter **Control Unit** (SINAMICS Standalone - CU3xx oder **SINAMICS_Integrated)** den **Eintrag Kommunikation > Telegramm-Konfiguration**.
  Der Dialog **Telegrammkonfiguration** mit dem Register **IF1 PROFIdrive PZD-Telegramme** wird angezeigt.

Der Dialog listet alle verfügbaren Antriebsobjekte auf. Nachfolgend werden die möglichen Einstellmöglichkeiten beschrieben.

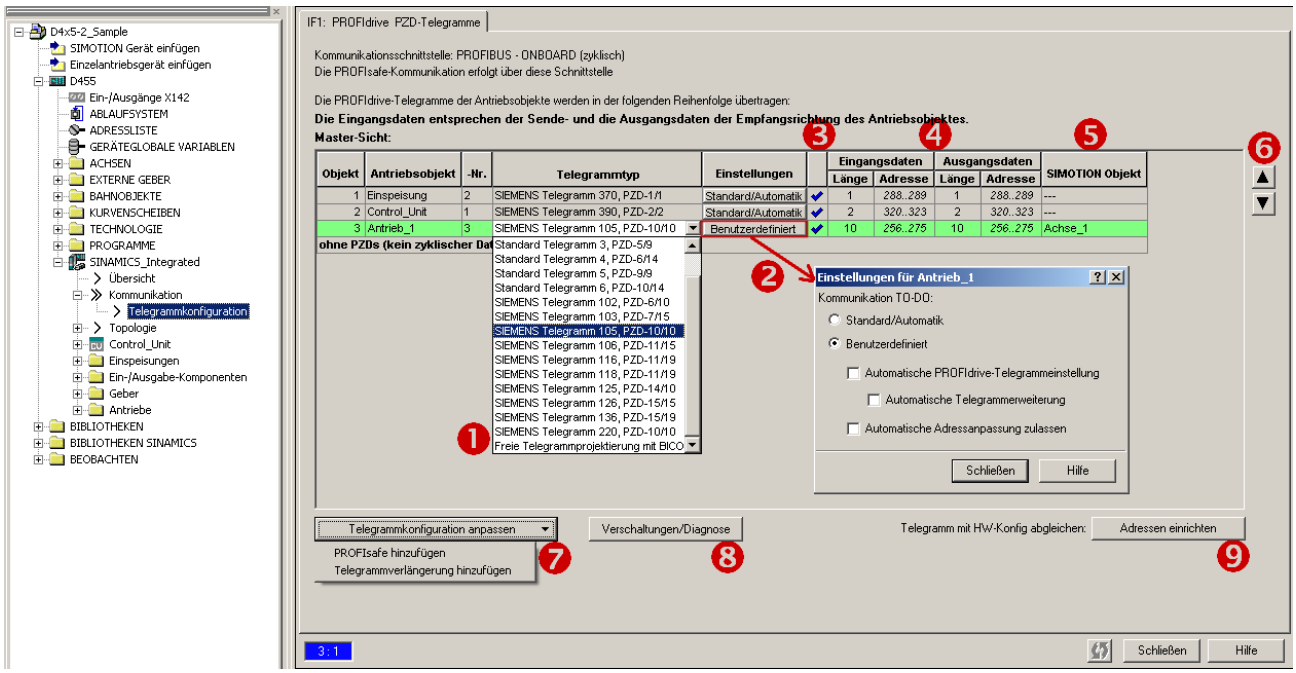


Bild 3-13 Telegrammkonfiguration

Tabelle 3- 1 Erläuterungen zum Bild

Nummer	Bedeutung	
1	Auswahl eines Telegramms • Die Antriebstelegramme (Telegramm 1 ... 6 und Telegramm 1xx) sind gemäß der PROFIdrive Spezifikation definiert und können anhand des benötigten Funktionsumfangs gewählt werden. • Über die Telegramme 39x können Sie für die Control Unit z. B. die Signale der I/Os bzw. der globalen Messtaster übertragen. Das Telegramm 39x ist auch für die Uhrzeitsynchronisation zwischen SIMOTION und SINAMICS erforderlich. • Freie Telegrammprojektierung mit BICO erlaubt es ein Telegramm selbst zu definieren. • Freie Telegrammprojektierung mit p915/p916 (bei TM15/17). • Telegramme 37x für die Ansteuerung der Einspeisung.	
2	Die Einstellung "Standard/Automatik" und "Benutzerdefiniert" ist nur sichtbar, wenn "Symbolische Zuordnung verwenden" aktiviert ist. Es wird generell empfohlen die Einstellung "Standard/Automatik" zu verwenden. Mit der Einstellung "Benutzerdefiniert" kann die automatische Telegrammeinstellung, Telegrammerweiterung und Adressanpassung deaktiviert bzw. aktiviert werden. • bei "Automatische PROFIdrive-Telegrammeinstellung" wird abhängig von der projektierten Technologie das Telegramm vom System eingestellt (Auswahl Telegramm z. B. für Einspeisung, Antrieb und Control Unit inkl. Onboard I/O). • bei "Automatische Telegrammerweiterung" wird abhängig von der projektierten Technologie das Telegramm vom System erweitert (z. B. wenn in der Achskonfiguration der Technologiedatenblock aktiviert wird). • bei "Automatische Adressanpassung zulassen" können z. B. bei Adressverschiebungen die Adressen durch das System angepasst werden. Zu Adressverschiebungen kann es z. B. kommen, wenn ein Telegramm verlängert wird und die angrenzenden Adressen bereits durch andere Telegramme belegt sind. Bei TM15 / TM17 High Feature ist ein Deaktivieren der "Automatischen PROFIdrive-Telegrammeinstellung", der "Automatischen Telegrammerweiterung" und der "Automatischen Adressanpassung" prinzipbedingt nicht möglich, da bei diesen Antriebsobjekten das Telegramm immer entsprechend der parametrierten Klemmenfunktionalität (DI, DO, Nocken, Messtaster) eingerichtet wird und auch nicht erweitert werden kann. Sollen für TM15 DI/DO, TM31 und TB30 die Telegramme manuell projektiert und BICO verschaltet werden, muss hierfür "Automatische PROFIdrive-Telegrammeinstellung" und "Automatische Telegrammerweiterung" deaktiviert werden. Siehe auch Kommunikation für symbolische Zuordnung einrichten (Seite 101).	
3	Über die Symbole in der Statusspalte werden folgende Informationen dargestellt:	
		Das Telegramm ist in HW Konfig anders konfiguriert. Sie müssen einen Abgleich mit HW Konfig durchführen.
		Sie verwenden ein vordefiniertes Standardtelegramm oder freie BICO-Verschaltung.
		Sie verwenden ein verändertes Standardtelegramm, das Sie mit Zusatzdaten verlängert haben.
		Sie verwenden ein Telegramm, bei dem eine der beiden Telegrammlängen (E oder A) zu lang ist. Das Antriebsobjekt kann diesen Eintrag nicht verarbeiten.
4	Länge: zeigt die Größe des Telegrammbestandteils an. Adresse: Adressbereich in HW Konfig. Die Adressen werden erst dann angezeigt, wenn die Adressen eingerichtet wurden.	
5	Zeigt das SIMOTION-Objekt an, welches mit dem SINAMICS-Objekt verschaltet ist (z. B. Achse oder Geber).	

Nummer	Bedeutung
6	Ändern der Telegramm-Reihenfolge. Alle Antriebsobjekte ohne Eingangs-/Ausgangsadressen ("---..---") müssen vor dem Abgleich hinter die Objekte mit noch abzugleichenden ("???..???") oder gültigen Eingang-/Ausgangsadressen verschoben werden.
7	Telegrammkonfiguration "manuell" anpassen (z. B. wenn über das Telegramm zusätzliche Daten wie z. B. eine Motortemperatur übertragen werden soll)
8	Anzeige der einzelnen Steuer- bzw. Zustandsworte des verwendeten Telegramms
9	Einrichten der Adressen (Abgleich der Adressen mit HW Konfig)

Hinweis

Wenn sich die Telegramme von Antriebsobjekten (Antrieben, Terminal Modules, ...) ändern, dann müssen Sie die Adressen erneut einrichten. Die Adressen werden nicht automatisch aktualisiert.

Siehe auch

Adressen und Telegramme einrichten - Übersicht (Seite 101)

3.6 Umstellung von Projekten auf Symbolische Zuordnung

3.6.1 Mit und ohne Symbolischer Zuordnung arbeiten

Beschreibung

Durch die symbolische Zuordnung vereinfacht sich die Projektierung der technologischen Beziehungen einschließlich der Kommunikation zwischen Steuerung und Antrieb erheblich. Auch für die Wartbarkeit von Projekten sind die Bezeichnungen der symbolischen Zuordnungen im Klartext von Vorteil.

Die symbolische Zuordnung ist bei neuen Projekten ab V4.2 daher empfohlen und automatisch aktiv.

Wird in einem Projekt symbolische Zuordnung verwendet, werden standardmäßig Telegramme, Verschaltungen und Adressen vom Engineeringsystem automatisch angelegt.

Das Engineeringsystem stellt dabei die aus Systemsicht "optimalen" PROFIdrive-Telegramme inklusive Telegrammverlängerungen ein, nimmt die erforderlichen BICO-Verschaltungen vor und ermittelt die Adressen.

Symbolische Zuordnung nachträglich aktivieren

Eine Umstellung von hochgerüsteten Projekten auf symbolische Zuordnung ist auch möglich und muss im Einzelfall bewertet werden, da ggf. Nacharbeiten am Projekt erforderlich werden. Vor allem bei freien Telegrammprojektierungen ist mit umfangreicheren Nacharbeiten zu rechnen.

VORSICHT
Wird die symbolische Zuordnung bei einem Projekt nachträglich aktiviert, in dem bereits Telegramme projektiert und verschaltet sind, können dieses einschließlich der BICO-Verschaltungen verändert werden! Legen Sie sich deshalb eine Sicherheitskopie Ihres Projektes an, bevor Sie die symbolische Zuordnung aktivieren.

Wird die symbolische Zuordnung nachträglich aktiviert und im Projekt sind bereits Telegramme projektiert und verschaltet, dann versucht das Engineeringsystem für diese Projektierung symbolische Zuordnungen zu ermitteln.

Konnten für alle Kommunikationsverbindungen eine symbolische Zuordnung ermittelt werden, muss beim erstmaligen Aktivieren der symbolischen Zuordnung anschließend die automatische Telegrammbestimmung /-verlängerung und Adressanpassung für alle Antriebsobjekte (DOs) aktiviert werden.

Ist es nicht möglich für alle Kommunikationsverbindungen eine symbolische Zuordnung zu bestimmen, wird dieses über einen Dialog gemeldet.

Zusätzlich werden in der Ausgabe "Zuordnungen einrichten" entsprechende Warnungen ausgegeben.

Tritt dieser Fall auf, ist VOR DEM ÜBERSETZEN eine der nachfolgenden Maßnahmen erforderlich:

- Möglichkeit 1: Automatik für Antriebsobjekt deaktivieren

  Im Antriebs-Dialog "Kommunikation" > "Telegrammkonfiguration" muss für die betroffenen Antriebsobjekte die automatische Telegrammbestimmung /-verlängerung und Adressanpassung deaktiviert werden (Einstellung benutzerdefiniert, alle Haken weg).

  Beim erstmaligen Aktivieren der symbolischen Zuordnung ist die automatische Telegrammbestimmung /-verlängerung und Adressanpassung standardmäßig für alle Antriebsobjekte (DOs) deaktiviert.

- Möglichkeit 2: Zuordnungen nachprojektieren

  Die symbolischen Zuordnungen müssen über die entsprechenden Zuordnungsdialoge der TO-Konfiguration bzw. der Adressliste vorgenommen werden

Grundlagen

Der Grad gegebenenfalls erforderlicher Nachprojektierungen wird dadurch bestimmt, wie gut "bereits definierte Telegramme/Verschaltungen" sich mit den Einstellungen decken, die das Engineering System zur Optimierung selbst vornimmt.

Aufgrund der Individualität der "Freien Telegrammprojektierung mit BICO" und der "Telegrammverlängerung" ist hier im besonderen Maße damit zu rechnen, dass eine symbolische Zuordnung nicht immer bestimmt werden kann.

Da es für TB30, TM15 DI/DO und TM31 keine Standard-Telegramme gibt und somit "Freie Telegrammprojektierung mit BICO" oder "Telegrammverlängerung" eingesetzt werden, sind diese Komponenten in besonderem Maße betroffen.

Gleiches gilt für die Onboard I/Os einer Control Unit (D4x5-2, CX32-2, CU320-2, ...) wenn hier die "Freien Telegrammprojektierung mit BICO" verwendet wurde.

Beispiel 1:

Werden z. B. Onboard I/Os einer Control Unit über "freie Telegrammprojektierung mit BICO" verschaltet, so sind Telegrammlänge, Telegrammaufbau und Verschaltung sehr individuell.

Wird nun die symbolische Zuordnung aktiviert, erfolgt die Abbildung generell auf Basis der Standardtelegramme 39x, wodurch sich Telegrammlänge, Telegrammaufbau und Verschaltung ändern.

Dieses hat dann z. B. Rückwirkungen auf die Adressierung eines Onboard I/Os, der von SIMOTION aus genutzt werden soll.

Beispiel 2:

Für einen Antrieb ist das Telegramm 106 projektiert (überträgt 2 Geberwerte), es wird aber nur ein Geber genutzt.

Wird nun die symbolische Zuordnung aktiviert, wird automatisch auf Telegramm 105 (überträgt 1 Geberwert) optimiert.

VORSICHT
Wird die symbolische Zuordnung bei einem Projekt nachträglich aktiviert, in dem bereits Telegramme projektiert und verschaltet sind, können dieses einschließlich der BICO-Verschaltungen verändert werden! Legen Sie sich deshalb eine Sicherheitskopie Ihres Projektes an, bevor Sie die symbolische Zuordnung aktivieren. TB30, TM15 DI/DO und TM31 sind hier in besonderem Maße betroffen.

Siehe auch

Hochrüsten von Projekten <V4.2 (Seite 109)

3.6.2 Symbolische Zuordnung aktivieren/deaktivieren

Beschreibung

Die Symbolische Zuordnung von SIMOTION und SINAMICS Schnittstellen kann folgendermaßen ein- bzw. ausgeschaltet werden. Ab V4.2 ist bei neu angelegten Projekten standardmäßig die symbolische Zuordnung aktiviert.

Werden Projekte < V4.2 hochgerüstet, ist als Voreinstellung die symbolische Zuordnung deaktiviert und muss bei Bedarf aktiviert werden.

Symbolische Zuordnung aktivieren

1. Trägt der Menüeintrag **Projekt>Symbolische Zuordnung verwenden** einen Haken, ist die Funktion bereits aktiviert. Andernfalls führen Sie **Projekt>Symbolische Zuordnung verwenden** aus.

   Ein Meldungsdialog wird aufgeblendet.

2. Bestätigen Sie mit **OK**.

   Die symbolische Zuordnung wird aktiviert und die notwendigen Schritte werden ausgeführt:

   – Zuordnungen zwischen den Objekten werden aus den logischen Adressen bestimmt und symbolisch angelegt

   – Adressen werden eingerichtet (spätestens automatisch vor dem Download)

   Hinweis

   Wenn Sie symbolische Zuordnung verwenden, findet die Verwaltung des Telegramms zwischen SIMOTION und den Antriebs-DOs vollständig unter Kontrolle des SCOUT statt. D.h. SCOUT erzeugt eigenständig Telegramme, die alle gemäß der technologischen Projektierung notwendigen Signale enthalten. Die Platzierung der Signale im Telegramm und die Telegrammgröße werden hierbei automatisch von SCOUT verwaltet und können nicht mehr vom Anwender beeinflusst bzw. geändert werden.

   Haben Sie ein Projekt auf V4.2 hoch konvertiert oder ein Projekt zunächst ohne symbolische Zuordnung begonnen, und wählen erst später die symbolische Zuordnung an, dann extrahiert SCOUT automatisch aus der bestehenden Telegrammprojektierung die dort enthaltenen Signalobjekte und erzeugt daraus automatisch neue Telegramme nach eigenem Muster. Dieses bedeutet insbesondere auch, dass sich die Anzahl und Größe der Telegramm Module (PROFIBUS) bzw. Telegramm Submodule (PROFINET) bei Anwahl der symbolischen Zuordnung zwangsläufig verändern wird (z.B. kann sich die Anzahl der Telegramm Module/Submodule zur Performanceoptimierung verringern).

Symbolische Zuordnung deaktivieren

Der Menüeintrag **Projekt>Symbolische Zuordnung verwenden** trägt einen Haken wenn die Funktion aktiviert ist.

1. Führen Sie **Projekt>Symbolische Zuordnung verwenden** aus.

   Ein Meldungsdialog wird aufgeblendet, der darauf hinweist, dass dadurch Adressen und symbolische Bezeichnungen eingerichtet und überprüft werden.

2. Klicken Sie auf **Nein**, wenn Sie die Symbolische Zuordnung aktiviert lassen möchten
3. Klicken Sie auf **Ja**, um die Symbolische Zuordnung zu deaktivieren.

Für die Verbindungen zwischen den Objekten sind nun wieder die logischen Adressen führend. Falls dabei ungültige Adressen auftreten, werden Sie mit einem Meldungsdialog darauf hingewiesen.

3.6.3 Hochrüsten von Projekten <V4.2

Beschreibung

Wenn Sie Projekte mit einer Version <V4.2 hochrüsten möchten, um mit der Symbolischen Zuordnung zu arbeiten, müssen Sie ggf. folgende Schritte durchführen. Siehe auch Mit und ohne Symbolischer Zuordnung arbeiten (Seite 105).

VORSICHT
Wird die symbolische Zuordnung bei einem Projekt nachträglich aktiviert, in dem bereits Telegramme projektiert und verschaltet sind, können dieses einschließlich der BICO-Verschaltungen verändert werden! Legen Sie sich deshalb eine Sicherheitskopie Ihres Projektes an, bevor Sie die symbolische Zuordnung aktivieren. TB30, TM15 DI/DO und TM31 sind hier in besonderem Maße betroffen.

Vorgehensweise

- Projekt im Scout öffnen: Das Projekt wird mit deaktivierter Symbolischer Zuordnung geöffnet. Im Dialog **Telegrammkonfiguration** der entsprechenden SINAMICS Control Unit sind die Dialoge zu den Telegrammeinstellungen der Symbolischen Zuordnung deaktiviert. Dadurch bleiben die bereits projektierten Telegramme, Adressen und BICO-Verschaltungen erhalten.
- Version des Gerätes auf V4.2 hochrüsten. Nur Geräte der Version V4.2 unterstützen die Symbolische Zuordnung. Das Hochrüsten der Geräte können Sie z. B. in HW Konfig durchführen.

- Symbolische Zuordnung aktivieren: Wenn Sie die Symbolische Zuordnung aktivieren, werden die Zuordnungen zwischen Achsen und Antrieben eingefügt und erst aktualisiert, wenn Sie das Projekt speichern und übersetzen. Die Adressen und Telegrammkonfiguration von SINAMICS-DOs bleiben weiterhin solange erhalten, bis Sie im Dialog **Telegrammkonfiguration** die automatische Telegrammkonfiguration aktivieren. Im Register **Zuordnungen einrichten** werden Fehlermeldungen, die beim Zuordnen auftreten, ausgegeben. Anhand dieser Fehlermeldungen können sie die TO-DO-Verschaltungen überprüfen und ggf. ändern.

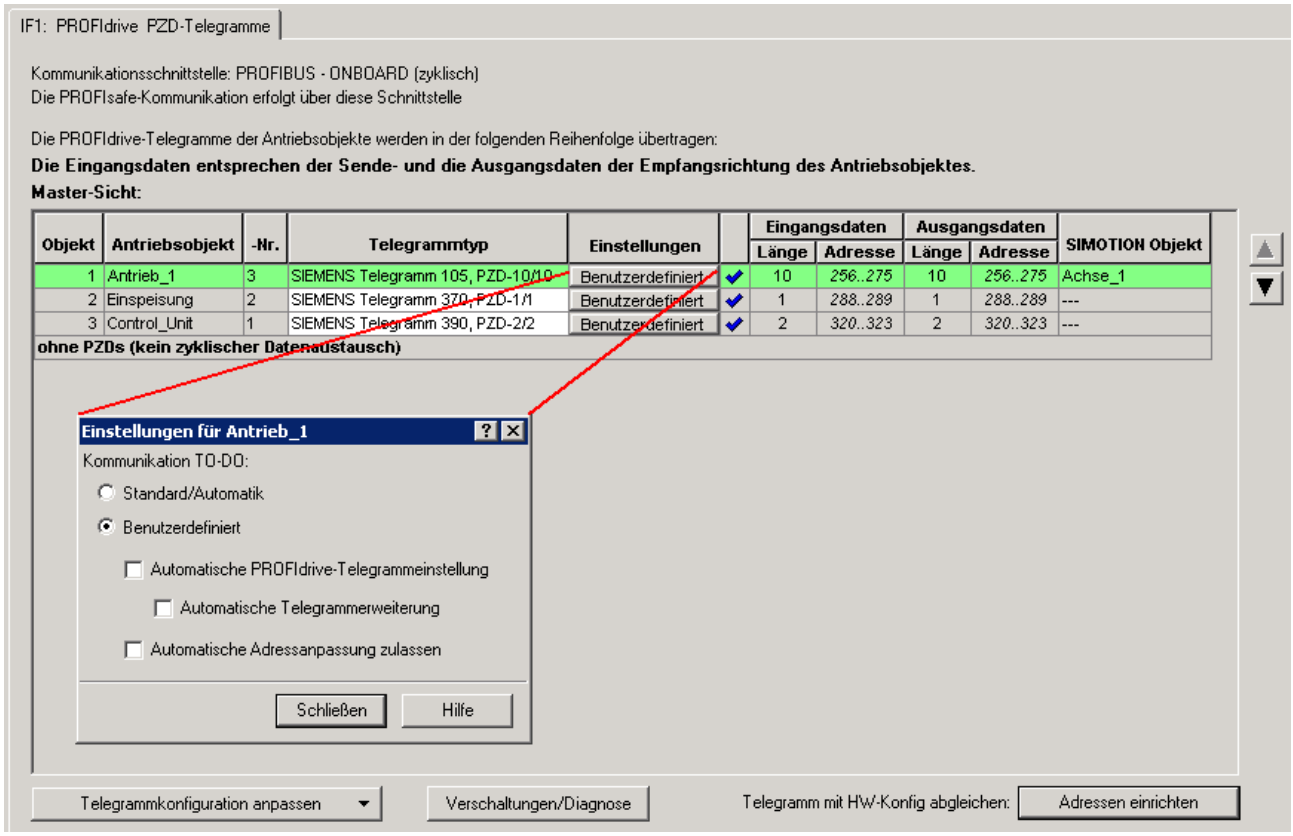


Bild 3-14 Telegrammkonfiguration nach Hochkonvertierung

- SINAMICS DOs zuordnen: Um die SINAMICS DOs an der Symbolischen Zuordnung teilnehmen zu lassen, müssen Sie im Dialog **Telegrammkonfiguration** für jedes DO folgende Schritte ausführen:
  - Dialog **Einstellung für <DO>** aufrufen und die Option **Standard/Automatik** auswählen, wenn Sie die Telegrammkonfiguration automatisch durchführen möchten.

    Oder

    Dialog **Einstellung für <DO>** aufrufen und die Option **Benutzerdefiniert** auswählen, wenn Sie die Telegrammkonfiguration benutzerdefiniert durchführen möchten, also z. B. statt eines Standardtelegramms die Einstellung **Freie Telegrammprojektierung über BICO** nutzen möchten (**Automatische PROFIdrive Telegrammeinstellung** abwählen).
- Kommunikation für symbolische Zuordnung einrichten: Wenn Sie alle Verschaltungen überprüft haben, können Sie **Projekt>Kommunikation für symbolische Zuordnung einrichten** ausführen. Es werden dann alle Telegramme, Adressen und BICO-Verschaltungen über die Symbolische Zuordnung aktualisiert.

3.6.4 Rückrüsten auf Versionen <V4.2

Beschreibung

Wird ein SIMOTION-Gerät über den Gerätetausch in HW Konfig rückgerüstet, so ist zu berücksichtigen, dass symbolische Zuordnungen erst ab der Version V4.2 zur Verfügung stehen.

Werden symbolische Zuordnungen verwendet und auf ein SIMOTION-Gerät <V4.2 rückgerüstet, so muss vor dem Gerätetausch in HW Konfig die "Kommunikation für die symbolische Zuordnung" eingerichtet werden. Hierdurch werden die Adressen ermittelt, die für ein Projekt "ohne symbolische Zuordnung" benötigt werden.

Siehe hierzu auch Kommunikation für symbolische Zuordnung einrichten (Seite 101).

Programmieren mit Technologieobjekten 4

4.1 Definitionen

Im Folgenden lernen Sie die Motion-Bestandteile primär aus der Sicht der Programmiersprache ST kennen. Es handelt sich dabei in erster Linie um Systemfunktionen, Systemvariablen und Konfigdaten. Nachfolgend einige Definitionen:

- Systemfunktionen sind Funktionen zur Handhabung des Systems. Sie bieten Zugang zur technologieneutralen Gerätefunktionalität. Systemfunktionen stehen immer zur Verfügung.
- Ein Technologieobjekt (TO) repräsentiert eine technologische Funktionalität (z. B. Achse positionieren, Nocken parametrieren) im SIMOTION Anwenderprogramm.
- TO-Funktionen oder Technologiebefehle sind Sprachbefehle, welche die einzelnen TO zur Verfügung stellen, d. h. Funktionen zu einem TO.
- Ein Technologiepaket enthält ein oder mehrere Technologieobjekttypen, aus denen mit <Technologieobjekt einfügen> die jeweilige TO-Instanz erzeugt wird.
- Systemvariablen und Konfigdaten sind Attribute der Technologieobjekte und des Grundsystems. Über Systemvariablen können Sie Technologieobjekte und das Grundsystem parametrieren bzw. deren Status ablesen

Hinweis

Weitere Informationen über die Grundlagen, Konfiguration und Programmierung der Motion-Control-Technologie und insbesondere über Technologieobjekte finden Sie in den Funktionshandbüchern SIMOTION Motion Control <Technologieobjekte>.

In der vorliegenden Dokumentation werden die genannten Themen nur angeschnitten.

4.2 Programmierung der Technologieobjekte (TO)

4.2.1 Technologie-Funktionen im Programm verwenden

Um Technologie-Funktionen (TO-Funktionen) verwenden zu können, müssen Sie einmalig ein Technologiepaket wählen. Im Technologiepaket sind Technologieobjekte (TO) und damit TO-Funktionen enthalten. Dabei handelt es sich formal um Funktionen, die einen Funktionsnamen, Eingangsparameter und meistens einen Rückgabewert besitzen. Nachfolgend ein Überblick über die Kennzeichen und Bestandteile der Technologieobjekte und TO-Funktionen außer Eingangsparameter, die in unter **Eingangsparameter der Technologie-Funktionen** beschrieben sind.

Nachfolgendes ist aus Sicht der ST-Programmierung betrachtet. Für KOP/FUP- und MCC-Programmierung siehe jeweilige Programmierhandbücher.

Technologiepaket wählen

Mit folgendem Befehl wählen Sie das Technologiepaket:

```
USEPACKAGE tp-name
```

Dabei ist *tp-name* der Name des Technologiepakets (siehe nächste Tabelle).

Tabelle 4- 1 Technologiepakete

Technologiepaket	Beschreibung des Inhalts
CAM	• Grundlegende Motion-Control-Befehle, Positionieren, Getriebegleichlauf und diskontinuierlicher Gleichlauf (Kurvenscheibe) • Befehle für Externe Geber, Messtaster, Nocken und Nockenspuren
CAM_EXT[1]	Befehle für ergänzende Technologieobjekte (Festes Getriebe, Addierobjekt, Formelobjekt, Sensor, Reglerobjekt)
TControl	Befehle für Temperaturregler

Kennzeichen der Technologieobjekte (TO)

Ein Technologiepaket stellt Technologieobjekte (TO) bereit, die im SIMOTION SCOUT instanziiert werden. Das instanziierte TO wird in der Programmierung über seinen Namen angesprochen (referenziert).

Kennzeichen der Technologie-Funktionen (TO-Funktionen)

TO-Funktionen sind in erster Linie Befehle, die bestimmte Aktionen am Technologieobjekt ausführen, weshalb sie auch TO-Befehle genannt werden. Sie können TO-Funktionen in selbstdefinierten FB einsetzen. Mehr zu den Formvorschriften der selbstdefinierten FCs und FBs können Sie im Programmierhandbuch SIMOTION ST erfahren. Die folgende Tabelle zeigt die formellen Kennzeichen der TO-Funktionen im Überblick.

Tabelle 4- 2 Kennzeichen der TO-Funktionen

Kennzeichen	Beschreibung
Name	Alle Namen der TO-Funktionen (im Beispiel *_enableAxis*) sind im SIMOTION System definierte Bezeichner, die grundsätzlich mit _ (Unterstrich) beginnen. Um selbstdefinierte FBs und FCs von den TO-Funktionen zu unterscheiden, sollten Sie keine Quelldatei-Abschnitte erstellen, die mit dem Zeichen _ beginnen.
Eingangsparameter	TO-Funktionen können beim Aufruf einen oder mehrere Eingangsparameter enthalten und geben immer einen Rückgabewert an die Aufrufstelle zurück. Ausgangsparameter sind nicht möglich. Weitere Informationen siehe Eingangsparameter (Seite 461).
Rückgabewert	Alle Befehle besitzen in der Regel eine Ganzzahl doppelter Genauigkeit (Datentyp DINT) als Rückgabewert. Dieser zeigt an, ob die Befehlsübergabe an das System und die Verarbeitung in Ordnung waren (Rückgabewert Null) oder ob ein Fehler aufgetreten ist (Returnwert ungleich Null).

Beispiel

Wenn die Anforderung an die Arbeitsaufgabe in der Freigabe einer virtuellen Achse besteht, könnten Sie eine Quelldatei so wie im nächsten Bild erstellen.

Folgende Voraussetzungen müssen gegeben sein:

- Es wurde eine Instanz einer Positionierachse oder Drehzahlachse als virtuelle Achse mit dem Namen Achse_1 im SIMOTION SCOUT angelegt.
- Das Programm *myPos* wurde beispielsweise der *MotionTask_1* zugeordnet. In der Taskkonfiguration der *MotionTask_1* wurde die Option **Aktivierung nach StartupTask** angewählt.
- Die Quelle wurde auf das Zielsystem geladen.

Nachdem die CPU in den Betriebszustand RUN versetzt wurde, wird der virtuellen Achse *Achse_1* eine Freigabe erteilt. Der Zustand der Achsfreigabe kann in der Systemvariablen *Achse_1.control* überprüft werden.

Tabelle 4- 3 Beispiel für die Verwendung von TO-Funktionen im Programm

```
INTERFACE
    USEPACKAGE CAM;
    PROGRAM myPos;
END_INTERFACE
IMPLEMENTATION
    // Das folgende Programm muss einer MotionTask zugeordnet
    // werden.
    // In der Taskkonfiguration muss die Option "Aktivierung
    // nach StartupTask" angewählt sein.
    PROGRAM myPos
        VAR
            retVal : DINT;
        END_VAR
        // Achse wird für eine Positionierung freigegeben.
        retVal := _enableAxis (
        axis := Achse_1,
        // Bezeichner der TO-Instanz
        nextCommand := WHEN_COMMAND_DONE,
        // Bedingung für Programmweiterschaltung.
        commandId := _getCommandId() );
        // Eindeutige Befehls-Id
    END_PROGRAM
END_IMPLEMENTATION
```

Hinweis

Verwenden Sie die unter **Eingangsparameter der Technologie-Funktionen** beschriebene Langform der Parameterübergabe (mit Wertzuweisung). Sie ist die übersichtlichere und flexiblere Form.

Tipps zur effizienten Verwendung von Parametern in Systemfunktionen erhalten Sie in **Fehlerquellen und Effizientes Programmieren**.

4.2.2 Programmbeispiel mit Namespace-Option

Mit dem optionalen Zusatz AS *namespace* können Sie einen Namensraum definieren. Sie können dann auch auf Typen, Variablen, Funktionen und Funktionsbausteine des Technologiepakets zugreifen, die namens gleich mit solchen der ST-Quelle sind.

Das folgende Beispiel zeigt, wie Sie das Technologiepaket CAM wählen, ihm den Namensraum Cam1 zuordnen und den Namensraum anwenden:

Tabelle 4- 4 Beispiel für die Wahl eines Technologiepaketes und Verwendung eines Namensraums

```
INTERFACE
    USEPACKAGE CAM AS Cam1;
    USES ST_2;
    FUNCTION function1;
END_INTERFACE

IMPLEMENTATION
    FUNCTION function1 : VOID
    VAR_INPUT
        p_Axis : posAxis;
    END_VAR
    VAR
        retVal : DINT;
    END_VAR

    retVal:= Cam1._enableAxis (
            axis := p_Axis,
            nextCommand := Cam1.WHEN_COMMAND_DONE,
            commandId := _getCommandId() );
    END_FUNCTION
END_IMPLEMENTATION
```

ACHTUNG
Falls ein Namensraum für eine importierte Bibliothek oder Technologiepaket definiert ist, muss dieser **immer** angegeben werden, wenn eine Funktion, Funktionsbaustein oder Datentyp aus dieser Bibliothek oder diesem Technologiepaket verwendet wird. Siehe in obigem Beispiel: Cam1._enableAxis, Cam1.WHEN_COMMAND_DONE.

Siehe auch

Funktionsparameter der Technologie-Funktionen (Seite 118)

Effizient Programmieren - Übersicht (Seite 565)

4.2.3 Unterschiede zwischen zyklischer und sequentieller Programmierung

Zyklische Tasks

Zyklische Tasks (z.B. die BackgroundTask) werden systemseitig nach ihrer Beendigung oder nach einem definierten Zeitraster automatisch neu gestartet. Die Werte statischer Variablen der zugeordneten Programme bleiben dabei erhalten. Zyklische Tasks besitzen eine Zeitüberwachung und eine definierte Fehlerreaktion bei deren Überschreitung. In zyklischen Tasks enthaltener Code muss daher schnell und effizient seine Aufgaben erledigen. Aufgaben mit wartendem Charakter (z.B. Warten auf das Positionieren einer Achse) sind nur in mehreren Aufrufzyklen der zyklischen Task zu erledigen. So sind TO Systembefehle in der Regel mit der Weiterschaltbedingung IMMEDIATELY am Parameter nextCommand aufzurufen. In den nachfolgenden Aufrufzyklen ist dann das Ergebnis des Systembefehls auf erfolgreiche Abarbeitung bzw. Fehler zu prüfen. Dieses Vorgehen wird auch asynchrone Bearbeitung genannt.

Sequentielle Tasks

Sequentielle Tasks (z.B. MotionTasks) werden nach dem Start einmal durchlaufen und dann beendet. Bei jedem Start werden alle lokalen Variablen der zugeordneten Programme initialisiert, zur Variableninitialisierung, siehe Einfluss des Compilers auf die Variableninitialisierung (Seite 317). Sequentielle Tasks haben keine Zeitüberwachung, können also eine beliebig lange Laufzeit erreichen. Sequentielle Tasks unterliegen nur der Kontrolle der Anwendung. D.h. die Anwendung kann diese Tasks starten, stoppen, anhalten und weiterlaufen lassen. In sequentiellen Tasks enthaltener Code arbeitet Aufgaben nacheinander ab, wobei die Folgeaufgabe üblicherweise erst dann ausgeführt wird, wenn die vorherige abgearbeitet wurde. So wird z.B. eine Achse erst freigegeben und danach referenziert. Aufrufe von TO Systembefehlen sollten daher mit der Weiterschaltbedingung WHEN_COMMAND_DONE am Parameter nextCommand erfolgen. Die Systemfunktion kehrt dann erst zur aufrufenden sequentiellen Tasks zurück, wenn der Befehl abgearbeitet wurde. Man spricht in diesem Zusammenhang auch von synchroner Bearbeitung.

Generelle Vorgehensweise

Generell ist es vorteilhaft sequentielle Abläufe in MotionTasks zu programmieren. Die Unterschiede zwischen einer sequentiellen Programmierung in einer MotionTask und der zyklischen Programmierung in der BackgroundTask sind:

- Im Rahmen der sequentiellen Abarbeitung der MotionTask kann zu einem Zeitpunkt auf eine und nur auf eine (jedoch auch verknüpfte) Weiterschaltbedingung gewartet werden. Die Weiterschaltbedingung wird hochprior im IPO-Takt geprüft. Zur Verkürzung der Reaktionszeit beim Fortsetzen der MotionTask kann deren Priorität temporär erhöht werden (-> WAITFORCONDITION).
- Ein zyklisches Programm prüft in jedem Zyklus in der Regel eine Vielzahl von Signalzuständen und Weiterschaltbedingungen. Die Zykluszeit wird dadurch stark belastet und wird länger. Der Vorteil gegenüber der sequentiellen Art der Programmierung liegt in der Parallelverarbeitung von Abfragen und Abläufen.

Synchrone und Asynchrone Befehlsbearbeitung

Synchrone Befehlsbearbeitung:

Es wird am Befehl mit der Programmweiterbearbeitung gewartet, bis der Befehl vorzugsweise in synchronen Tasks vollständig ausgeführt ist.

Asynchrone Bearbeitung:

Die Funktion wird über einen Befehl aktiviert, und die Programmbearbeitung vorzugsweise in zyklischen Tasks fortgesetzt, während die Funktion vom System noch ausgeführt wird.

Weitere Informationen dazu finden Sie unter Übergangs- und Weiterschaltbedingung (Seite 124).

4.2.4 Funktionsparameter der Technologie-Funktionen

Die Funktionsparameter der TO-Funktionen sind:

- *mandatory*, d. h. sie müssen angegeben werden, z. B. die TO-Instanz und die Zielposition beim Positionieren;
- *optional predefined*, d. h. sie können angegeben werden, bei Nichtangabe wird ein systemmäßig eingestelltes Verhalten (Default-Verhalten) wirksam, z. B. IMMEDIATELY bei der Weiterschaltbedingung;
- *optionalUserDefault*, d. h. sie können angegeben werden, bei Nichtangabe wird ein vom Anwender vorgebbares Default-Verhalten wirksam, das in der dazugehörigen userDefault-Variablen per Konfiguration einstellbar oder programmierbar ist. Zur Verwendung der Parameterwerte in der Bewegungsprogrammierung siehe Dokumentation zu den Technologieobjekten (TO).

Bei den TO-Befehlen ist der TO-Instanzname immer anzugeben, da vom System nicht voreinstellbar. In unserem **Beispiel für vorzugebende Variable** müssen Sie die Achse angeben, die aktiviert werden soll (axis := myaxis).

Verbindliche Regeln für die Angabe von Funktionsparametern

In der folgenden Tabelle sehen Sie Regeln, die für die Angabe von Funktionsparametern verbindlich sind.

Tabelle 4- 5 Regeln für die Angabe von Funktionsparametern

Regel	Beispiel
Zuweisungen	
Es werden immer alle Funktionsparameter übergeben, entweder mit den programmierten Werten oder mit den Voreinstellungen.	Im **Beispiel für vorzugebende Variablen** werden die Aktualwerte *myAxis* (Variable) und *IMMEDIATELY* (Wert) den Funktionsparametern *axis* und *nextCommand* zugewiesen.
Mehrere Funktionsparameter	
Mehrere Funktionsparameter werden durch Kommata getrennt.	Im **Beispiel für vorzugebende Variablen**: *_enableAxis (axis:=myAxis, nextCommand := IMMEDIATELY)*
Reihenfolge der Funktionsparameter	
Die Reihenfolge der Funktionsparameter ist beliebig, wenn die ersten beiden Voraussetzungen eingehalten werden.	Im **Beispiel für vorzugebende Variablen** ist auch *_enableAxis (nextCommand:= IMMEDIATELY, axis:=myAxis)* möglich.
Funktionsparameter mit voreingestellten Werten	
Wenn die Parameternamen mit angegeben werden, ist die Reihenfolge beliebig (z.B. "velocity:=..."). Wenn nicht alle Parameternamen mit angegeben werden, müssen alle in der richtigen Reihenfolge eingegeben werden.	Im **Beispiel für vorzugebende Variablen** können Sie den Funktionsparameter *nextCommand := IMMEDIATELY* weglassen, da dies der voreingestellte Wert ist.
Aktualwerte als Variable	
Der Nutzen für Sie besteht in der Wiederverwendbarkeit der Quelldatei-Abschnitte, die diese Variablen verwenden. Beispielsweise soll ein selbstdefinierter Funktionsbaustein (FB) eine TO-Funktion mit variablen Achsbezeichnungen als Funktionsparameter aufrufen. Ein mehrmaliger Aufruf mit konstanten Achsbezeichnungen wäre umständlich und würde den selbstdefinierten FB nicht wieder verwendbar machen. Sie können auch andere Aktualwerte als Variable einsetzen.	Im **Beispiel für vorzugebende Variablen** wird im selbst definierten Funktionsbaustein *posFB* die TO-Instanz *myAxis* definiert (*myAxis:PosAxis*). Beim Aufruf einer TO-Funktion aus diesem FB heraus wird die TO-Instanz als Aktualwerte verwendet (*Axis := myAxis*). Die Werte für die TO-Instanz kommen nicht aus dem selbstdefinierten FB, sondern aus dem Programm, das diesen FB mit *myFB(myaxis := Axis)* aufruft. Aus den genannten Gründen können Sie den FB und somit die TO-Funktion auch mit *myAxis := Axis2* oder *myAxis := Axis3* usw. aufrufen. Dies macht den selbstdefinierten FB in allen Programmen der ST-Quelldatei verwendbar, mit Hilfe der Exportfunktion auch in Programmen anderer ST-Quelldateien.
Aktualwerte als Werte (Enumeratoren)	
Wenn Sie Aktualwerte als Werte eingeben (im Beispiel *nextCommand:= IMMEDIATELY*), müssen Sie meistens einen Wert aus verschiedenen konkreten Zuständen (nur bei Enumeratoren) wählen. Enumeratoren gehören zum abgeleiteten Datentyp Aufzählung. Grundsätzliches zu Enumeratoren finden Sie in Kapitel Anwendedefinierte Datentypen im ST Programmierhandbuch. Sie können aber auch für andere Datentypen Direktwerte vorgeben.	Im **Beispiel für vorzugebende Variablen** sind bei der TO-Funktion *_enableAxis* für den Funktionsparameter *nextCommand* die Enumeratoren IMMEDIATELY und WHEN_COMMAND_DONE vorgesehen. Andere als diese vorgegebenen Werte lehnt der Compiler bei der Übersetzung ab.
Parameter für die Übergangs- und Weiterschaltbedingung	

Regel	Beispiel
Bei vielen Bewegungsbefehlen können Sie festlegen, wann die TO-Funktion am Technologieobjekt ausgeführt wird und die nächste Anweisung bearbeitet wird (siehe **Übergangs- und Weiterschaltbedingung**).	
Parameter zur Befehlsidentifikation	
Alle Bewegungsbefehle müssen einen Parameter zur Befehlsidentifikation enthalten. Sie können damit den Status der Befehlsbearbeitung verfolgen (siehe **Diagnose der Befehlsbearbeitung**)	

Hinweis

Alle Systemdatentypen für Aufzählungen (in Systemfunktionen und Systemvariablen) beginnen mit dem Wort *Enum*. Beispielsweise hat der Funktionsparameter *nextCommand* den Aufzählungsdatentyp *EnumNextCommandEnable* (mit den Werten *IMMEDIATELY* oder *WHEN_COMMAND_DONE*).

Alle Systemdatentypen für Strukturen (in Systemfunktionen und Systemvariablen) beginnen mit dem Wort *Struct*.

Wenn Sie selbstdefinierte Datentypen nicht mit den Zeichenfolgen *Enum* oder *Struct* beginnen lassen, können keine Namensüberschneidungen auftreten. Für eine detaillierte Beschreibung der Namensräume, siehe Programmierhandbuch ST.

Anhang D dieses Handbuchs enthält alle reservierten Bezeichner der Programmiersprache ST (Structured Text), der Systemfunktionen, der Systemfunktionsbausteine und der SIMOTION Geräte.

Die reservierten Bezeichner der SIMOTION Technologiepakete finden Sie in den Listenhandbüchern der SIMOTION Technologiepakete.

Bezugs- und Wertvorgabe für Bewegungsgrößen

Parameter von Bewegungsgrößen (z. B. Geschwindigkeit, Beschleunigung) werden über einen Bezugs- und einen Wertparameter definiert.

- Der Bezugsparameter legt fest, auf welche Systemgröße sich die zu übergebende Bewegungsgröße bezieht und wie ggf. der nachfolgende Wertparameter zu interpretieren ist.

  Die Bezeichnung eines Bezugsparameters ergibt sich aus der Bezeichnung der Bewegungsgröße mit dem Suffix *Type*, z. B. *velocityType*.

- Der Wertparameter gibt an:
  - Bei Bezugsparameter = DIRECT: den Zahlenwert der Bewegungsgröße.
  - Bei Bezugsparameter = USER_DEFAULT: den Skalierungsfaktor des in er Systemvariablen gespeicherten Vorbelegungswerts.

  Bei anderen Bezugsparametereinstellungen ist der Wertparameter ohne Bedeutung.

  Die Bezeichnung eines Wertparameters ist die Bezeichnung der Bewegungsgröße, z. B. *velocity*.

Tabelle 4- 6 Häufige Bezugsparameter und Wirkung der zugehörigen Wertparameter

Bezugsparameter	Wertparameter	Wert der Bewegungsgröße
DIRECT	absoluter Wert	Inhalt des Wertparameters (mit der bei der Konfiguration des Technologieobjekts festgelegten Einheit der Bewegungsgröße). Siehe Beispiel in nachfolgender Tabelle.
USER_DEFAULT	relativer Wert [%]	Voreinstellung * Wertparameter / 100 Die Voreinstellungen für die Bewegungsgrößen sind in einer Systemvariablen gespeichert. Siehe Beispiel in nachfolgender Tabelle.
ACTUAL	–[1]	aktueller Istwert.
CURRENT	–[1]	aktueller Sollwert des Interpolators.
EFFECTIVE	–[1]	letzter programmierter Wert.
[1] Wertparameter wird ignoriert		

Tabelle 4- 7 Beispiel für velocityType (Bezugsparameter) und velocity (Wertparameter)

velocityType	velocity	Wert der Bewegungsgröße
DIRECT	(keine Angabe)	100 [konfigurierte Einheit]
	50.0	50 [konfigurierte Einheit]
	200.0	200 [konfigurierte Einheit]
USER-DEFAULT	(keine Angabe)	100 % des Wertes der Systemvariablen[1]
	50.0	50 % des Wertes der Systemvariablen[1]
	200.0	200 % des Wertes der Systemvariablen[1]
[1] Bei Bewegungsbefehlen einer Achse: Systemvariable *userDefault.velocity*		

Vorgabe der Funktionsparameter ohne Angabe der Funktionsparameterbezeichner

Für die Angabe von Funktionsparametern beim Funktionsaufruf werden zwei Varianten unterstützt (siehe IEC1131/3 2nd Edition, 11/98):

- "call by value" als Aufruf mit fester Anordnung und Anzahl der Eingangsvariablen entsprechend der Funktionsdeklaration, wobei kein Zuweisungsoperator zugelassen ist.
- "call by name" als Aufruf mit variabler Anordnung und Anzahl der Eingangsvariablen, wobei eine Zuweisung des Übergabeparameters zum Formaloperanden notwendig ist. Bei dieser Aufrufvariante können mit Default-Werten belegte Übergabeparameter ungenutzt bleiben.

Bei der Kurzform der Parameterübergabe werden die Funktionsparameter (Parameterbezeichner) weggelassen und nur die Aktualwerte (Parameterwerte) verwendet.

ACHTUNG
Bei der Kurzform der Parameterübergabe müssen Sie *alle* Parameterwerte (auch optionale) durch Kommata getrennt *in der richtigen Reihenfolge* angeben!

Nur bei wenigen Systemfunktionen (siehe **Funktionen zur Laufzeitmessung von Tasks**, **Tasksteuerbefehle** und **Funktionen zur Meldungsprojektierung**) müssen Sie beim Aufruf die Kurzform der Parameterübergabe verwenden. Dies ist bei den betreffenden Funktionen explizit angeben.

Hinweis

Verwenden Sie die oben beschriebene Langform (mit Wertzuweisung). Sie ist die übersichtlichere und flexiblere Form.

Datentyp der TO-Instanz-Variablen

Sie können TO-Instanz-Variablen beispielsweise für die Objektspezifikation in Aufrufe von TO-Funktionen verwenden.

Hinweis

Neue TO-Datentypen (ab V4.0) beginnen mit einem "_", um Konflikte mit Anwendervariablen zu vermeiden.

Davor müssen Sie jedoch diese Variablen deklarieren und dabei aus einer vorgegebenen Auswahl von Datentypen wählen, siehe Tabelle zu **Datentypen von Technologieobjekten**. In unseren Beispielen **Verwendung von TO-Funktionen im Programm** in diesem Abschnitt (**Beispiel für vorzugebende Variablen**) lautet der Datentyp *PosAxis*, da Sie TO-Instanz-Variablen für Positionierachsen anlegen wollen. Das Technologiepaket *Cam* (*Diskontinuierlicher Gleichlauf*), das Sie gewählt haben, enthält den Datentyp *PosAxis* für das Technologieobjekt *Positionierachse*.

Tabelle 4- 8 Datentypen von Technologieobjekten (Namen der TO)

Technologieobjekt	Datentyp	enthalten im Technologiepaket
Drehzahlachse	driveAxis	CAM, PATH, CAM_EXT
Externer Geber	externalEncoderType	CAM, PATH, CAM_EXT
Messtaster	measuringInputType	CAM, PATH, CAM_EXT
Nocken	outputCamType	CAM, PATH, CAM_EXT
Nockenspur	_camTrackType	CAM, PATH, CAM_EXT
Positionierachse	posAxis	CAM, PATH, CAM_EXT
Gleichlaufachse	followingAxis	CAM, PATH, CAM_EXT
Gleichlaufobjekt	followingObjectType	CAM, PATH, CAM_EXT
Bahnachse (ab V4.1)	_pathAxis	PATH, CAM_EXT
Bahnobjekt (ab V4.1)	_pathobjecttype	PATH, CAM_EXT
Kurvenscheibe	camType	CAM, CAM_EXT
Festes Getriebe	_fixedGearType	CAM_EXT
Addierobjekt	_additionObjectType	CAM_EXT
Formelobjekt	_formulaObjectType_	CAM_EXT
Sensor	_sensorType	CAM_EXT
Reglerobjekt	_controllerObjectType_	CAM_EXT

Technologieobjekt	Datentyp	enthalten im Technologiepaket
Temperaturkanal	temperatureControllerType	TControl
allgemeiner Datentyp, dem jedes TO zugewiesen werden kann	ANYOBJECT	

Variablen vom Datentyp der Technologieobjekte werden standardmäßig mit dem Wert TO#NIL initialisiert. Dies können Sie dazu verwenden, um abzufragen, ob einer Variablen ein gültiges TO zugewiesen wurde, siehe **Beispiel für vorzugebende Variablen**.

Beispiel für vorzugebende Variablen

Es folgt ein Beispiel für vorzugebende Variablen vom Datentyp eines Technologieobjekts (ein Beispiel für die optionale Verwendung haben Sie in **Datentypen von Technologieobjekten** gesehen). Es soll ein wieder verwendbarer FB geschrieben werden, der jeweils eine beliebige Achse freischaltet. Da der Achsname variabel ist, müssen Sie im FB eine Variable vom Datentyp dieser Achse definieren.

Die Achse, die freigeschaltet werden soll, wird beim Aufruf des FB mitgegeben. Sicherheitshalber wird dies überprüft. Wenn kein TO vorhanden ist (TO#NIL), dann wird die Ausführung des FB unterbrochen.

Tabelle 4- 9 Beispiel für vorzugebende Variablen vom Datentyp eines Technologieobjekts (TO)

```
// ...
FUNCTION_BLOCK posFB
    VAR_INPUT
        myAxis : posAxis;
    END_VAR

    VAR_OUTPUT
        //Rückgabewert der TO-Funktion,
        //gleichzeitig Ausgangsparameter des FB
        return_value : DINT := -1;
    END_VAR
    // Abfrage nach gültigem TO
    IF myAxis = TO#NIL THEN RETURN; END_IF;
        // Beispiel für Aufruf mit Variablen vom Datentyp des TO
        return_value := _enableAxis (
        axis := myAxis, // TO-Funktion
        nextCommand := IMMEDIATELY, //optional
        commandId    := _getCommandId() );
END_FUNCTION_BLOCK
PROGRAM Example
    VAR
        myFB : posFB;
    END_VAR
    myFB (myaxis := Axis1);
    //Name wird bei Inbetriebnahme im SIMOTION SCOUT angelegt.
    myFB (myaxis := Axis2);
    //Name wird bei Inbetriebnahme im SIMOTION SCOUT angelegt.

END PROGRAM
//...
```

4.2.5 Übergangs- und Weiterschaltbedingungen

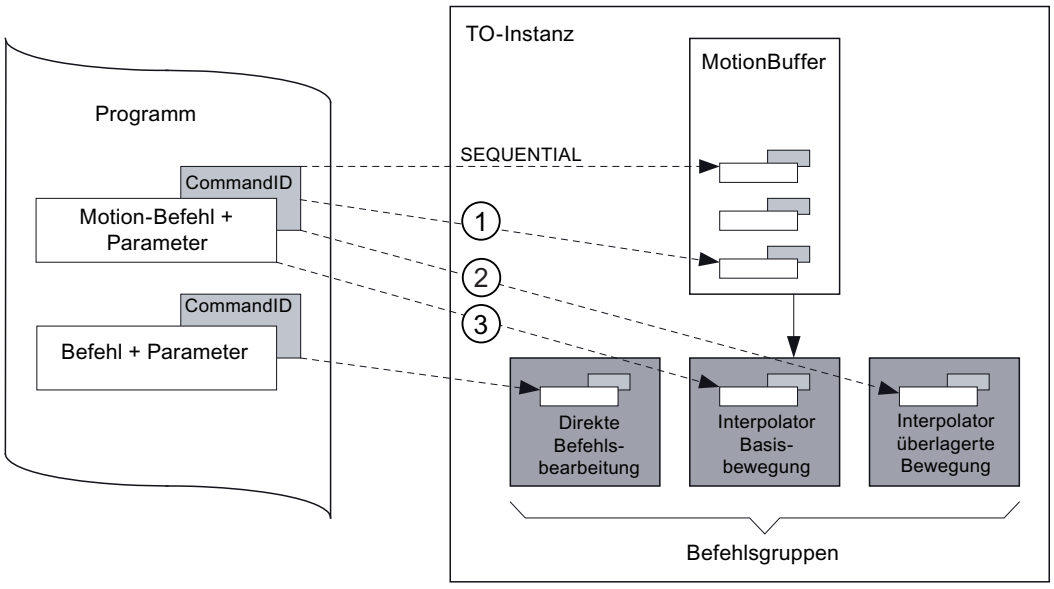
Grundlagen zur Bearbeitung einer TO-Funktion (Befehlsbearbeitung))

Die TO-Funktionen werden als Befehl an das Technologieobjekt zur Bearbeitung übergeben. Das TO bearbeitet bzw. aktiviert diese Befehle in dem Bearbeitungstakt, der bei dessen Konfiguration festgelegt wurde (z. B. IPO-Takt).

Die Technologieobjekte Nocken, Messtaster, Externer Geber und Kurvenscheibe verfügen über eine direkte Befehlsbearbeitung. Ein neuer Befehl am selben Technologieobjekt verdrängt einen dort aktiven Befehl.

Neben Befehlen zur direkten Befehlsbearbeitung können an den Technologieobjekten Drehzahlachse, Positionierachse, Gleichlaufachse und Gleichlaufobjekt auch Motion-Befehle abgesetzt werden. Die Technologieobjekte verfügen über Strukturelemente zur Befehlsverwaltung.

Nachfolgend ist als Beispiel die Befehlsverwaltung für Achsen beschrieben.



1 NEXT COMMAND
2 SUPERIMPOSED_MOTION_MERGE
3 IMMEDIATELY

Bild 4-1 Befehlsbearbeitung an den Achsen

Der MotionBuffer nimmt Motion-Befehle auf, die vom Interpolator sequentiell abgearbeitet werden. Die Anzahl von Motion-Befehlen, die der MotionBuffer aufnehmen kann, wird über das Konfigurationsdatum *TypeOfAxis.DecodingConfig.numberOfMaxbufferedCommandId* definiert. Somit können mehrere Motion-Befehle unabhängig vom Bearbeitungszustand des aktiven Befehls am TO abgesetzt werden.

Jedem Befehl wird beim Absetzen eine **CommandId** übergeben Diese wird am Befehl gespeichert und stellt eine Referenz auf den abgesetzten Befehl dar.

Die Befehle sind Befehlsgruppen zugeordnet. Die anstehenden Befehle der vorhandenen Befehlsgruppen werden parallel vom Interpolator bearbeitet. Es gibt Befehle, die direkt am TO aktiv werden, die sich in den MotionBuffer einreihen oder überlagernde Befehle, die direkt wirksam werden.

Hinweis

Das Verhalten der Befehlsgruppen und Befehlspuffer, z.B. des MotionBuffer ist TO-spezifisch. So können Sie z.B. die beiden Befehle _enableaxistorquelimitpositive und _enableaxistorquelimitnegative nicht gleichzeitig innerhalb eines IPO2-Taktes aufrufen. Nur einer der Befehle wird ausgeführt.

Eine genaue Beschreibung zu Befehlsgruppen und Befehlspuffer finden Sie in den TO-Handbüchern, z.B. im Funktionshandbuch TO Achse elektrisch/hydraulisch, Externer Geber.

Übergangsverhalten vom aktuell wirksamen Bewegungsbefehl

Das Übergangsverhalten vom aktuell wirksamen Bewegungsbefehl am Technologieobjekt legen Sie in der TO-Funktion mit dem Parameter *MergeMode* fest. Hier geben Sie an, wie sich die TO-Funktion in die Bearbeitungsfolge der Befehle am Technologieobjekt einreiht bzw. welcher Befehlsgruppe sie zugeordnet wird.

Tabelle 4- 10 Häufige Mergemodes

MergeMode	Beschreibung
IMMEDIATELY	Die mit dem Befehl vorgegebene Bewegung wird unmittelbar aktiv; bereits aktive Bewegungen werden abgelöst, bereits anstehende Befehle/ Bewegungen werden abgebrochen.
NEXT_MOTION	Ausführen nach der aktiven Bewegung und Löschen weiterer anstehenden Befehle / Bewegungen
SEQUENTIAL	Anhängen an die vorhergehenden Befehle/ Bewegungen
SUPERIMPOSED_ MOTION_MERGE	Neben der Grundbewegung ist eine überlagernde Bewegung an der Achse möglich.

Befehlsweiterschaltbedingung

Bei TO-Funktionen, die im Interpolator ausgeführt werden, kann eine Weiterschaltbedingung programmiert werden. Sie legt fest, wann die nächste Anweisung in der Programmquelle ausgeführt wird. Hierzu dient der Parameter *nextCommand*. Die zulässigen Bedingungen sind befehlsspezifisch und können dem Listenhandbuch SIMOTION Technologiepaket CAM Systemfunktionen (Referenzliste) entnommen werden.

Im Wesentlichen unterscheidet man zwischen asynchroner und synchroner Befehlsbearbeitung:

- Asynchrone Befehlsbearbeitung:

  Die TO-Funktion wird an das Technologieobjekt übergeben und das Programm unmittelbar fortgesetzt. Hierzu wird der *nextCommand* = IMMEDIATELY gesetzt.

  In diesem Fall ist seitens der Applikation sicherzustellen, dass TO-Funktionen nur einmalig angestoßen werden und Rückmeldungen durch Abfrage des Achs- bzw. Befehlsstatus explizit ausgewertet werden (siehe **Diagnose der Befehlsbearbeitung**).

  Beispiel: siehe die beiden folgenden Bilder

  Diese Art der Bewegungsführung wird als *zyklische Programmierung* bezeichnet. Sie ist in allen Tasks des Systems zulässig und ist insbesondere für die Programmierung zyklischer Tasks vorgesehen.

  Die asynchrone Befehlsbearbeitung ist die Voreinstellung, wenn der Parameter *nextCommand* nicht angegeben wird.

- Synchrone Befehlsbearbeitung:

  Die TO-Funktion wird inklusive der Parametrierung zur Weiterschaltung an das Technologieobjekt übergeben und die aufrufende Task angehalten. Das Technologieobjekt führt die Funktion aus und veranlasst die Fortsetzung der Programmbearbeitung, sobald die spezifizierte Weiterschaltbedingung erfüllt ist oder der Befehl abgebrochen wurde. Hierzu wird der *nextCommand* auf die gewünschte Weiterschaltbedingung gesetzt.

  Beispiel: siehe **Asynchrone Programmbearbeitung (sequentielle Programmierung)**.

  Diese Art der Bewegungsführung wird als *sequentielle Programmierung* bezeichnet. Sie wird insbesondere durch die MotionTasks unterstützt.

  Die Programmierung von sequentiellen Befehlsabläufen in zyklischen Tasks führt zu Zeitüberschreitungen der Tasks und somit zu Laufzeitfehlern.

Tabelle 4- 11 Beispiel für asynchrone Programmbearbeitung (zyklische Programmierung) - Teil 1

```
INTERFACE
USEPACKAGE CAM;
    PROGRAM ProgramCycle;
END_INTERFACE

IMPLEMENTATION

PROGRAM ProgramCycle
    VAR
        boStartCommand : BOOL; // Kommando - Befehl absetzen
        boCommandStarted : BOOL; // Hilfsvariable - Befehl abgesetzt
        boCommandDone : BOOL; // Hilfsvariable - Befehl ausgeführt
        i32Ret : DINT; // Rückgabewert Systemfunktionen
        sCommandId : CommandIdType; // CommandId
        // Rückgabewert - _getStateOfAxisCommand
        sRetCommandState : StructRetCommandState;
        // Instanz des System-FB zur Flankenerkennung
        r_trig_1 : R_TRIG;
    END_VAR

    r_trig_1 (boStartCommand); // Aufruf der Flankenerkennung

    IF r_trig_1.q THEN
        // Anforderung einer systemweit eindeutigen CommandId
        sCommandId := _getCommandId ();
        // CommandId am TO registrieren
        // --> Diagnose von Abbruch und Ende des Befehls möglich
        i32Ret := _bufferAxisCommandId (
        axis := Axis_1,
        commandId := sCommandId );
        // Auswertung des Rückgabewertes der Systemfunktion
        // ...
        // Absetzen eines Befehls - Bewegen mit USER_DEFAULT-Werten
        i32Ret := _move(
        axis := Axis_1,
        nextCommand := IMMEDIATELY,
        commandId := sCommandId );
        // Auswertung des Rückgabewertes der Systemfunktion
        // ...
        // Hilfsvariablen zur Koordination der Befehlsbearbeitung
        boCommandStarted := TRUE;
        boCommandDone := FALSE;
//-----------------------------------------------------------------------
// Fortsetzung folgt
```

Tabelle 4- 12 Beispiel für asynchrone Programmbearbeitung (zyklische Programmierung) - Teil 2

```
// Fortsetzung
//-----------------------------------------------------------------------
    ELSIF boCommandStarted AND NOT boCommandDone THEN
        // Abfrage Status der Befehlsbearbeitung
        sRetCommandState := _getStateOfAxisCommand(
        axis := Axis_1,
        commandId := sCommandId );

    IF sRetCommandState.functionResult = 0 THEN
        IF sRetCommandState.commandIdState = EXECUTED THEN
            // Befehl wurde ausgeführt (beendet)
            boCommandStarted := FALSE;
            boCommandDone := TRUE;
            // Registrierte CommandId am TO entfernen
            i32Ret := _removeBufferedAxisCommandId(
            axis := Axis_1,
            commandId := sCommandId );
        END_IF;
    ELSE
        // Fehlerbehandlung Funktionsaufruf _getStateOfAxisCommand
        // ...
        ;
    END_IF;

    ELSIF boCommandDone THEN
        // Bearbeitung nach Befehlsausführung
        // ...
        ;
    END_IF;

 // Ab hier weiteres Anwenderprogram
 // ...

END_PROGRAM

END_IMPLEMENTATION
```

Tabelle 4- 13 Beispiel für synchrone Programmbearbeitung (sequentielle Programmierung)

```
INTERFACE
    USEPACKAGE CAM;
    VAR_GLOBAL
        g_boCommandStarted : BOOL; // Hilfsvariable - Befehl abgesetzt
        g_boCommandDone : BOOL; // Hilfsvariable - Befehl ausgeführt
        END_VAR
        PROGRAM ProgramSequential;
END_INTERFACE

IMPLEMENTATION
    PROGRAM ProgramSequential
        VAR
            i32Ret : DINT; // Rückgabewert der Systemfunktion
        END_VAR;
        g_boCommandStarted := TRUE;
        g_boCommandDone := FALSE;
        // Anweisungen, die vor der Bewegung ausgeführt werden.
        // ...
        i32Ret := _move(
        axis:= Axis_1,
        nextCommand:= WHEN_MOTION_DONE,
        commandId:= _getCommandId () );

        // Anweisungen, die nach der Bewegung ausgeführt werden.
        // ...
        // Auswertung des Rückgabewertes der Systemfunktion
        // ...

        g_boCommandStarted := FALSE;
        g_boCommandDone := TRUE;

    END_PROGRAM
```

4.2.6 Diagnose der Befehlsbearbeitung

Befehlsidentifikation – CommandId

Beim Absetzen eines Befehls durch eine Systemfunktion wird eine CommandId übergeben. Diese CommandId wird während der Bearbeitungsdauer durch das TO am Befehl gespeichert und identifiziert somit den Befehl.

Mit der Systemfunktion *_getCommandId* erhält man eine projektweit eindeutige CommandId. Dies stellt sicher, dass kein weiterer Befehl mit derselben CommandId im System existiert (eindeutige Referenz auf den Befehl).

Tabelle 4- 14 Beispiel für die Verwendung der CommandId

```
//...
VAR
    myCommandId : CommandIdType;
END_VAR
//...
// Eindeutige Id speichern
myCommandId := _getCommandId ();
// Funktion mit Id ausführen
myFC := _pos (axis := myAxis,
        position        := position_1,
        nextComand        :=IMMEDIATELY,
        commandId        :=myCommandId);
//...
```

Nachfolgend ist beschrieben, wie Sie mit Hilfe der CommandId den Bearbeitungsstatus eines Befehls verfolgen können.

Systemfunktionen zum Abfragen des Befehls-/ Bearbeitungsstatus

Technologieobjekte, an denen mehrere Befehle zur Bearbeitung abgesetzt werden können, verfügen über die Systemfunktionen *_getStateOf...Command* (z. B. *_get StateOfAxisCommand*). Der Rückgabewert vom Datentyp *StructRetCommandState* gibt in der Komponente *EnumCommandIdState* Auskunft über den Bearbeitungszustand eines Bewegungsbefehls durch den Interpolator.

Bei *EnumCommandIdState* = ABORTED wird der Abbruchgrund in der Komponente *abortId* (Datentyp DINT) angegeben. Die Werte entsprechen dem Grund im Technologischen Alarm "30002 Befehl abgebrochen".

Statusabfrage nach Ende oder Abbruch eines Befehls

Standardmäßig wird ein Befehl nach Beendigung oder Abbruch aus der internen Befehlsverwaltung des TO entfernt. Somit kann der Status *Ende* oder *Abbruch* eines Befehles nicht durch den Aufruf o. g. Systemfunktionen diagnostiziert werden.

Damit es möglich ist, den Befehlszustand auch nach Beendigung oder Abbruch des Befehls abzufragen, muss die CommandId des betreffenden Befehls der internen Befehlsverwaltung des TO mitgeteilt werden. Dies geschieht über die Systemfunktion *_buffer...CommandId (z. B. _bufferAxisCommandId)*.

Nach erfolgter Auswertung des Status *Ende* bzw. *Abbruch* muss die CommandId explizit aus der Befehlsverwaltung des TO entfernt werden. Dies geschieht über die Systemfunktion *_removeBuffered...CommandId* (z. B. *_removeBufferedAxisCommandId*).

Beispiel 1	Der _buffer... Befehl wird nicht abgesetzt und der _pos Befehl, für den die Abfrage erfolgen soll, ist bereits beendet
Ergebnis	Da zu der im _getStateOf...CommandId angegebenen CommandId kein passender Befehl gefunden wird (der _pos ist ja bereits beendet), wird NOT_EXISTENT ('commandId' ist nicht bekannt oder Befehl ist bereits beendet) zurückgegeben.
Beispiel 2	Der _buffer... Befehl wird abgesetzt und der _pos Befehl, für den die Abfrage erfolgen soll, ist bereits beendet.
Ergebnis	Der _pos Befehl wird zwar nicht mehr gefunden, aber das Ergebnis wurde im CommandId-Puffer abgelegt. Das Ergebnis ist jetzt entweder EXECUTED (Bearbeitung des Befehls beendet) oder ABORTED (Bearbeitung des Befehls abgebrochen).

Die Größe des CommandID-Puffers ist begrenzt und kann z.B. an der Achse mit dem Konfigdatum *TypeOfAxis.DecodingConfig.NumberOfMaxBufferedCommandId* eingestellt werden. Sie können so dem TO mitteilen, wie viele Befehle maximal gleichzeitig verwaltet werden müssen

Beim STOP-RUN Übergang werden die gepufferten Command-ID gelöscht. Der CommandID-Puffer ist danach also leer.

Beispiel: siehe **Asynchrone Programmbearbeitung (zyklische Programmierung), Teil 1 und Teil 2**.

Das Verhalten der "buffer und removeBuffer" Befehle ist bei allen TO, die diese Funktionalität unterstützen gleich (Ausnahmen: Namen der Befehle und Name des Konfigdatums für die Größe des Puffers).

Hinweis

Die Beschreibung gilt sinngemäß auch für den Externen-Geber.

Statusabfrage nach Rücksetzen eines TO

Die CommandID kann so gepuffert werden, dass sie beim Rücksetzen eines Technologieobjekts nicht gelöscht wird. Sie ist dann auch nach Rücksetzen eines TO verfügbar.

- Rufen Sie hierzu die TO-Funktion *_buffer...CommandId* mit dem Parameter *deleteCommandIdWithReset= NO* auf. Die CommandId kann dann nur explizit mit dem Befehl *_removeBuffered...CommandId* gelöscht werden.
- Wenn Sie die TO-Funktion *_buffer...CommandId* mit dem Parameter *deleteCommandIdWithReset= YES* (Voreinstellung) aufrufen, wird die CommandId auch beim Rücksetzen des Technologieobjekts mit der Funktion *_reset...* (z. B. *_resetAxis*) gelöscht.

4.2.7 Bezeichner von Instanzen der Technologieobjekte

Die Bezeichner von Instanzen der Technologieobjekte legen Sie bei deren Konfiguration in SIMOTION SCOUT fest. Sie müssen innerhalb eines SIMOTION Geräts eindeutig definiert sein.

Sie können in der Regel die Instanz eines Technologieobjekts mit ihrem Bezeichner aufrufen.

Wenn jedoch der gleiche Bezeichner in ST-Quellen als Datentyp, Variable, Funktion oder Funktionsbaustein definiert ist oder wenn eine geräteglobale Variable oder I/O-Variable gleichen Namens angelegt wurde, überdecken diese Bezeichner die Instanz des Technologieobjekts.

Um dennoch die Instanz des Technologieobjekts zu erreichen, um z. B. auf deren Systemvariablen oder Konfigurationsdaten zuzugreifen, verwenden Sie den vordefinierten Namensraum *_to* (siehe **Namensräume** im ST-Programmierhandbuch). Setzen Sie die Bezeichnung des Namensraums durch Punkt getrennt vor den entsprechenden Namen, z. B. *_to.to-name*.

Wenn Sie auf die Instanz eines Technologieobjekts auf einem anderen SIMOTION Gerät zugreifen wollen, setzen Sie den Namen des SIMOTION Geräts durch Punkt getrennt vor die Bezeichnung der Instanz, z. B. *dev-name.to-name* oder *dev-name._to.to-name*.

Wenn der Bezeichner eines Geräts überdeckt wird, können Sie den vordefinierten Namensraum *_project* verwenden, z. B. *_project.dev-name.to-name* oder *_project.dev-name._to.to-name*.

Hinweis

Bei projektweit eindeutigem Bezeichner für die Instanz eines Technologieobjekts können Sie den vordefinierten Namensraum *_project* auch für die Kennzeichnung der Instanz verwenden.

4.2.8 Wandlung von TO-Datentypen

Typkonvertierungen innerhalb der hierarchischen Datentypen

Die TO-Datentypen *driveAxis*, *posAxis*, *followingAxis* sind vom Funktionsumfang hierarchisch strukturiert.

- Eine Positionierachse (Datentyp *posAxis*) enthält die Funktionalität einer Drehzahlachse (Datentyp *driveAxis*).
- Eine Gleichlaufachse (Datentyp *followingAxis*) enthält die Funktionalität einer Positionierachse (Datentyp *posAxis*) und damit auch die einer Drehzahlachse (Datentyp *driveAxis*).

Es sind nur Typkonvertierungen innerhalb dieser hierarchischen Datentypen sowie mit dem allgemeinen TO-Datentyp ANYOBJECT möglich.

Hinweis

Andere Typkonvertierungen sind nicht möglich (z. B. zwischen Messtaster und Gleichlaufobjekt).

Implizite Typkonvertierung

Zuweisungen von Variablen (mit TO-Datentyp) oder von TO-Instanzen an folgende Variablen ist ohne Angabe einer Konvertierungsfunktion möglich:

- an Variable eines hierarchisch niedrigeren TO-Datentyps:
  - *followingAxis* an *posAxis* oder *driveAxis*
  - *posAxis* an *driveAxis*
- an Variable des allgemeinen TO-Datentyps ANYOBJECT

Siehe folgende Tabelle.

Tabelle 4- 15 Beispiel für implizite Typkonvertierung

```
// Im Projektnavigator ist folgende TO-Instanz (Achse) konfiguriert:
fol_axis_real als Gleichlaufachse

VAR
    drv_axis1 : driveAxis;
    pos_axis1 : posAxis;
    any_obj1 : ANYOBJECT;
END_VAR

drv_axis1    := pos_axis1;
any_obj1     := fol_axis_real;...
```

Explizite Typkonvertierung

Die Zuweisung von Variablen (mit TO-Datentyp) an Variable mit hierarchisch höherem TO-Datentyp ist mit Hilfe der Typkonvertierungsfunktion *anyObject_to_Object* möglich.

Voraussetzung dafür ist, dass die Quellvariable (mit dem hierarchisch niederen TO-Datentyp) auf eine TO-Instanz verweist, die hierarchisch mindestens dem TO-Datentyp der Zielvariablen entspricht (siehe Beispiel folgende Tabelle).

Tabelle 4- 16 Beispiel für erfolgreiche Typkonvertierungen

```
// Im Projektnavigator sind folgende TO-Instanzen (Achsen)
 // konfiguriert:
 // pos_axis_real als Positionierachse
 // fol_axis_real als Gleichlaufachse

VAR
    drv_axis1 : driveAxis;
    pos_axis1 : posAxis;
    any_obj1 : ANYOBJECT;
END_VAR

// implizite Typkonvertierungen
drv_axis1 := pos_axis_real;
any_obj1 := fol_axis_real;

// erfolgreiche Typkonvertierungen

pos_axis1 := anyObject_to_Object (in := drv_axis1);
 // Typkonvertierung ist erfolgreich,
 // da drv_axis1 auf eine Positionierachse verweist.

pos_axis1 := anyObject_to_Object (in := any_obj1);
 // Typkonvertierung ist erfolgreich,
 // da any_obj1 auf eine Gleichlaufachse verweist.

//...
```

Bei nicht erfolgreichen Typkonvertierungen wird der Zielvariablen der Wert TO#NIL zugewiesen:

Tabelle 4- 17 Beispiel für erfolglose Typkonvertierung

```
// Im Projektnavigator ist folgende TO-Instanz (Achse)
 // konfiguriert:
 // drv_axis_real als Drehzahlachse

VAR
    pos_axis1 : posAxis;
    any_obj1 : ANYOBJECT;
END_VAR

// implizite Typkonvertierungen
any_obj1 := drv_axis_real;

// erfolglose Typkonvertierung
pos_axis1 := anyObject_to_Object (in :=    any_obj1);
        // Typkonvertierung ist nicht erfolgreich,
        // da any_obj1 auf eine Drehzahlachse verweist.
        // pos_axis1 besitzt den Wert TO#NIL.
```

4.2.9 Systemvariablen

Über Systemvariablen können Sie Technologieobjekte und das Grundsystem parametrieren bzw. deren Status ablesen.

Syntax der Systemvariablen

Die Abfrage von Systemvariablen erfolgt mittels Strukturzugriff mit folgender Syntax. Die folgende Tabelle zeigt Ihnen die Bedeutung der einzelnen Syntax-Bestandteile:

[_to.]TO-Name.Variable.Komponente oder

[_device.]Variable.Komponente

Tabelle 4- 18 Syntax-Bestandteile der Systemvariablen

Syntax-Bestandteil	Bedeutung
TO-Name	*TO-Name* steht für den Namen eines Technologieobjekts (TO), z. B. einer Achse. Hierzu haben Sie: • entweder das Technologieobjekt in SIMOTION SCOUT eingefügt, • oder im Programm eine Variable vom Datentyp dieses Technologieobjekts deklariert. Eine Auflistung aller Datentypen für Technologieobjekte finden Sie in **Eingangsparameter der Technologie-Funktionen**.
_to	Das optionale Wort *_to* kennzeichnet den vordefinierten Namensraum für Technologieobjekte. Die sollten es verwenden, um mit *TO-Name* eindeutig ein Technologieobjekt zu spezifizieren. Siehe auch **Namensräume** im ST-Programmierhandbuch.

Syntax-Bestandteil	Bedeutung
_device	Das optionale Wort *_device* kennzeichnet den vordefinierten Namensraum für gerätespezifische Variablen (Systemvariablen der SIMOTION Geräte, I/O-Variablen, geräteglobale Variablen). Sie sollten es verwenden, um die Variable eindeutig als Systemvariable des SIMOTION Geräts zu spezifizieren. Diese Systemvariablen sind auch vorhanden, wenn kein Technologiepaket geladen ist. Siehe auch **Namensräume** im ST-Programmierhandbuch.
Variable	*Variable* steht für den Namen der Systemvariablen, den Sie in der Auflistung aller Systemvariablen finden (siehe Listenhandbuch zu den Systemvariablen der SIMOTION Technologiepakete).
Komponente	*Komponente* steht für den Teil der Struktur, den Sie abfragen wollen. Es kann sich dabei um eine weitere Struktur handeln, die ebenfalls Komponenten enthält. Die Tiefe dieser Struktur hängt von der Systemvariablen ab und kann auch Null sein.

Verwendung der Systemvariablen

Lesender Zugriff auf eine Systemvariable ist immer möglich. Sie können eine Systemvariable auf zwei Arten einer Variablen zuweisen:

- Mit einer Wertzuweisung (siehe auch **Wertzuweisungen** im ST-Programmierhandbuch). Dabei wird im Fehlerfall die ExecutionFaultTask aufgerufen (siehe auch *Ersatzwert bzw. letzten Werte auslesen* am Ende des Kapitels). Zur weiteren Fehlerreaktion siehe Zugriffen auf Systemvariablen) (Seite 149).
- Mit der Systemfunktion *_getSafeValue* und *_setSafeValue* (siehe Funktion _getSafeValue (Seite 406) , Funktion _setSafeValue (Seite 408) und Zugriffe auf Systemvariablen und Ein- Ausgänge (Seite 405)). Dabei ist es möglich, die im Fehlerfall gewünschte Reaktion zu programmieren.

Auch die Verwendung in einem Ausdruck oder als Parameter in einer Funktion oder einem Funktionsbaustein ist möglich. Hier wird im Fehlerfall ebenfalls die ExecutionFaultTask aufgerufen (siehe Fehler bei Zugriffen auf Systemvariablen (Seite 149)).

Ob eine Systemvariable beschrieben werden kann, erkennen Sie im Listenhandbuch (Referenzliste) der Systemvariablen der Technologieobjekte (TO) an folgendem Eintrag:

- Wirksam : *immediately* (kann beschrieben und gelesen werden)
- Wirksam : *read only* (kann nur gelesen werden).

Falls eine Systemvariable beschrieben werden kann, können Sie dieser auf zwei Arten einen Wert zuweisen:

- Mit einer Wertzuweisung (siehe auch **Wertzuweisungen** im ST-Programmierhandbuch). Dabei wird im Fehlerfall die ExecutionFaultTask aufgerufen (siehe auch *Ersatzwert bzw. letzten Werte auslesen* am Ende des Kapitels). Zur weiteren Fehlerreaktion siehe Fehler bei Zugriffen auf Systemvariablen und Konfigurationsdaten sowie auf I/O-Variablen für Direktzugriff (Seite 149)
- Mit der Systemfunktion _setSafeValue (siehe Funktion _setSafeValue (Seite 408)). Dabei ist es möglich, die im Fehlerfall gewünschte Reaktion zu programmieren.

Ein besonderer Variablenwert ist TO#NIL. Mit diesem Wert können Sie abfragen, ob ein gültiges Technologieobjekt vorliegt, siehe Beispiel in **Eingangsparameter der Technologie-Funktionen**.

Hinweis

Greifen Sie aus Gründen der Performance nie mehr als nötig auf Systemvariablen zu, sondern speichern Sie deren Inhalt einmalig in lokale Variablen des gleichen Datentyps. Der Zugriff auf lokale Variablen verbraucht weit weniger Ressourcen, da der Prozessor nicht so stark in Anspruch genommen wird. Näheres siehe **Effizient programmieren**.

Beachten Sie auch eine mögliche Fehlerquelle, wenn Sie REAL-Größen, LREAL-Größen und Systemvariablen (z. B. Achsposition) miteinander vergleichen, siehe **REAL- oder LREAL-Größen vergleichen**.

Hinweis

ONLINE geänderte Systemvariablen können nicht mit "Sichern auf Speicherkarte (Ram2Rom)" bzw. "Sichern im Engineeringprojekt (Laden Konfigurationsdaten in PG)" gespeichert werden.

Damit Werte von Systemvariablen auch im Engineeringprojekt und auf Speicherkarte gesichert werden, muss der Wert von Systemvariablen OFFLINE geändert und dann per Download ins Zielsystem geladen und gesichert werden. Siehe Speicherkonzept im Zielsystem.

Gültigkeitsdauer von Systemvariablen

1. Systemvariablen, z.B. die Statusanzeige stehen ggf nur einen IPO-Takt an.
2. Alle Systemvariablen haben die dokumentierte Eigenschaft **Aktualisierung**.
3. Möchten Sie den Status einer Task mit geringerer Zykluszeit abfragen, sollte der Status mit dem darauf folgenden Status ODER verknüpft werden. Mit der ODER-Verknüpfung stellen Sie sicher, dass alle Status der Applikation berücksichtigt werden. Die Verknüpfung sorgt dafür, dass nachfolgende ENUMS der Statusanzeige zusammengefasst werden.

Beispiele für Systemvariablen

Sie wollen die Achsposition und den dynamischen Achszustand für die Achse *Axis1* abfragen.

Voraussetzungen:

- Sie haben die Achse Axis1 im SIMOTION SCOUT angelegt oder sie im Programm definiert und initialisiert, z. B. mittels Variable *myAxis* vom Datentyp *PosAxis*.
- Sie haben im Programm Variablen für die Aufnahme der Achsposition und des dynamischen Achszustandes definiert. Der Datentyp dieser Variablen muss dem Datentyp der abzufragenden Variablen entsprechen, z. B.:

```
VAR
act_pos : LREAL;
act_motionState : EnumAxisMotionState;
END_VAR
```

Beispiel für Zugriff auf Systemvariable mit Strukturelement eines elementaren Datentyps:

```
act_pos := Axis1.positioningState.actualPosition;
```

Bei *PositioningState* handelt es sich um die Systemvariable und bei *actualPosition* um das Strukturelement vom Datentyp LREAL, das abgefragt wird.

Beispiel für Abfrage einer Systemvariablen mit Aufzählungselement:

```
act_motionState := Axis1.motionStateData.motionState;
```

Bei *motionStateData* handelt es sich um die Systemvariable und bei *motionState* um das Strukturelement vom Aufzählungsdatentyp *EnumAxisMotionState*, das abgefragt wird.

Initialisierung von Systemvariablen

Systemvariablen werden bei einem Download des Projekts in der Regel nicht neu initialisiert; ihre Aktualwerte werden nicht auf die Anfangswerte zurückgesetzt. Systemvariablen werden in der Regel nur beim Wiedereinschalten des SIMOTION Geräts neu initialisiert.

Ersatzwert beziehungsweise letzten Wert von Systemvariablen bei TO-Restart oder deaktiviertem TO (ab V4.1)

Der Zugriff auf Systemvariablen ist auch bei RESTART des TO bzw. bei deaktiviertem TO möglich, ohne dass das System in STOP geht. Anstatt die Systemvariablen über die Funktion _getSaveValue auszulesen, können Sie für einen direkten Zugriff über einen Eintrag in den Konfigdaten (restart.behaviorInvalidSysvarAccess) folgendes konfigurieren:

- letzten Wert auslesen (LAST_VALUE = Voreinstellung)
- Defaultwert auslesen (=Wert bei Laden des Projektes; DEFAULT_VALUE)
- in STOP gehen (STOP_DEVICE)

Ausnahme

Variablen, die den aktuellen TO-Zustand liefern, geben auch bei RESTART den korrekten Status zurück. Das betrifft die Systemvariable restartActivation, auf die Sie über _getSafeValue zugreifen können.

Anmerkung

Schreiben auf die TO-Sytemvariablen während eines TO-Restarts oder bei einem deaktiviertem TO ist möglich (außer bei STOP_DEVICE). Die Werte werden nach dem RESTART übernommen bzw. sind wirksam. Beim Schreiben von Systemvariablen außerhalb der Grenzen geht die CPU in STOP. Ab V4.2 ist Schreiben außerhalb der gültigen Grenzen möglich. Der Grenzwert wird dann übernommen.

4.2.10 Konfigurationsdaten

Konfigurationsdaten legen die grundsätzliche Funktionalität eines Technologieobjekts fest. Sie werden i. d. R. bei der Konfiguration des Technologieobjekts mit SIMOTION SCOUT eingestellt und sind zur Laufzeit überwiegend fest.

Ein Großteil dieser Konfigurationsdaten kann zur Laufzeit geändert werden. Diese Festlegung ist spezifisch für jedes Konfigurationsdatum und ist im Listenhandbuch zu den Konfigurationsdaten der SIMOTION Technologiepakete dokumentiert.

Abhängig vom Konfigurationsdatum stehen folgende Möglichkeiten zur Verfügung:

- Nicht online änderbar:

  Diese Konfigurationsdaten können ausschließlich bei der Konfiguration des Technologieobjekts mit SIMOTION SCOUT verändert werden.

- Online änderbar, wirksam nach Restart

  Diese Konfigurationsdaten können durch einen Variablenzugriff aus dem Anwenderprogramm geändert werden. Die Änderung wird erst nach Restart des Technologieobjekts wirksam (siehe **Zurücksetzen eines Technologieobjekts**).

- Online änderbar, wirksam sofort

  Diese Konfigurationsdaten können durch einen Variablenzugriff aus dem Anwenderprogramm geändert werden. Die Änderung wird sofort wirksam.

Hinweis

Wenn sie ein Konfigurtionsdatum ändern, dessen Änderung erst nach einem TO Restart wirksam wird, so werden auch die nachfolgenden Änderungen von "sofort wirksamen" Konfigurtionsdaten erst nach dem TO Restart wirksam.

Hinweis

ONLINE geänderte Konfigurationsdaten können mit "Aktual nach RAM kopieren" und anschließendem "RAM nach ROM kopieren" auf Speicherkarte gesichert werden bzw. mit "Laden in PG" im Engineeringprojekt gespeichert werden. Siehe Speicherkonzept im Zielsystem.

Lesen der Konfigurationsdaten

Sie können jedes Konfigurationsdatum eines Technologieobjekts lesen und z. B. einer Variablen zuweisen:

- den im RAM des SIMOTION Geräts gespeicherten Wert (ablesbar in der Expertenliste in der Spalte "nächster Wert")

  Verwenden Sie hierzu die Syntax: *TO-name*.setconfigdata.*config-date*.

- den aktuell am Technologieobjekt wirksam Wert

  Verwenden Sie hierzu die Syntax: *TO-name*.activeconfigdata.*config-date*.

Siehe Beispiel in folgender Tabelle.

Es ist nur der Zugriff auf Einzelvariablen der Konfigurationsdaten erlaubt.

Tabelle 4- 19 Lesender Zugriff auf ein Konfigurationsdatum eines Technologieobjekts

```
VAR
    lreal_var : LREAL;
    drive_var : driveaxis;
END_VAR

lreal_var :=
    drive_var.setconfigdata.TypeOfAxis.MaxJerk.maximum;
    // lesender Zugriff auf gespeicherten Wert
lreal_var :=
    drive_var.activeconfigdata.TypeOfAxis.MaxJerk.maximum;
    // lesender Zugriff auf aktuell wirksamen Wert
```

Sie können ein Konfigurationsdatum auf zwei Arten einer Variablen zuweisen:

- Mit einer Wertzuweisung wie im Beispiel der vorhergehenden Tabelle (siehe auch **Wertzuweisungen** im ST-Programmierhandbuch). Dabei wird im Fehlerfall die *ExecutionFaultTask* aufgerufen (siehe auch *Ersatzwert bzw. letzter Wert* am Ende des Kapitels). Zur weiteren Fehlerreaktion siehe Fehler bei Zugriffen auf Konfigdaten (Seite 406).
- Mit der Systemfunktion *_getSafeValue* (siehe Funktion _getSafeValue (Seite 406)). Dabei ist es möglich, die im Fehlerfall gewünschte Reaktion zu programmieren.

Auch die Verwendung in einem Ausdruck oder als Parameter in einer Funktion oder einem Funktionsbaustein ist möglich. Hier wird im Fehlerfall ebenfalls die *ExecutionFaultTask* aufgerufen.

Änderung der Konfigurationsdaten zur Laufzeit (Online-Änderung)

Die Online Änderung von Konfigurationsdaten erfolgt durch einen einfachen schreibenden Variablenzugriff auf den im RAM gespeicherten Wert. Verwenden Sie hierzu die Syntax: *TO-name*.setconfigdata.*config-date*.

Die Wirksamkeit (Übernahme als aktuell am Technologieobjekt wirksamer Wert) ist durch das Konfigurationsdatum vorgegeben (wirksam sofort / wirksam nach Restart).

Siehe Beispiel **Schreibender Zugriff auf ein Konfigurationsdatum**.

Es ist nur der Zugriff auf Einzelvariablen der Konfigurationsdaten erlaubt.

Tabelle 4- 20 Schreibender Zugriff auf ein Konfigurationsdatum eines Technologieobjekts

```
VAR
    lreal_var : LREAL;
    drive_var : driveaxis;
END_VAR

drive_var.setconfigdata.TypeOfAxis.MaxJerk.maximum :=
    200000.0;
    // schreibender Zugriff auf gespeicherten Wert
```

Sie können einem Konfigurationsdatum auf zwei Arten einen Wert zuweisen:

- Mit einer Wertzuweisung wie im Beispiel der vorhergehenden Tabelle (siehe auch **Wertzuweisungen** im ST-Prgrammierhandbuch). Dabei wird im Fehlerfall die *ExecutionFaultTask* aufgerufen (siehe auch *Ersatzwert bzw. letzter Wert* am Ende des Kapitels). Zur weiteren Fehlerreaktion siehe Fehler bei Zugriffen auf Konfigdaten (Seite 149).
- Mit der Systemfunktion *_setSafeValue* (siehe Funktion _setSafeValue (Seite 408)). Dabei ist es möglich, die im Fehlerfall gewünschte Reaktion zu programmieren.

Darüber hinaus besteht die Möglichkeit, das Wirksamwerden über eine Systemvariable zu steuern.

Mit der Systemvariablen des Technologieobjekts *activationModeChangedConfigData* legen Sie fest, wann das Aktivieren geänderter Konfigurationsdaten erfolgen soll:

- Bei der Einstellung *activationModeChangedConfigData* = *ACTIVATE_CHANGED_CONFIG_DATA* werden die geänderten Daten unmittelbar aktiv gesetzt. Mit dem Setzen der Systemvariablen auf diesen Wert werden bisher gesammelte Daten ebenfalls aktiviert.
- Bei der Einstellung *activationModeChangedConfigData* = *COLLECT_CHANGED_CONFIG_DATA* werden die geänderten Daten gesammelt.

  Sie werden geschlossen aktiviert, sobald Sie diese Systemvariable auf *ACTIVATE_CHANGED_CONFIG_DATA* setzen.

  Die gesammelten geänderten Konfigurationsdaten können Sie (ohne Aktivierung) löschen, indem Sie die entsprechende Systemfunktion des Technologieobjekts (z. B. *_resetAxisConfigDataBuffer*) aufrufen.

ACHTUNG
Vom Schreiben des geänderten Konfigurationsdatum bis zum dessen Wirksamwerden vergeht auch bei der Einstellung *activationModeChangedConfigData* = *ACTIVATE_CHANGED_CONFIG_DATA* eine gewisse Zeit. Insbesondere beim Ändern mehrerer Konfigurationsdaten können deshalb Zeitüberläufe in Tasks auftreten. In SynchronousTasks ist das Ändern von Konfigurationsdaten nicht möglich.

Hinweis

Wenn mehrere Konfigurationsdaten gleichzeitig geändert werden sollen, wird empfohlen, diese mit *activationModeChangedConfigData* = *COLLECT_CHANGED_CONFIG_DATA* zu sammeln und anschließend mit der Einstellung *ACTIVATE_CHANGED_CONFIG_DATA* in einer sequentiellen Task geschlossen zu aktivieren.

Zur Laufzeit geänderte Konfigdaten können auf Karte und im ES-Projekt gesichert werden, siehe Speicherzugriffe (Seite 518) .

Hinweis

Achten Sie bei Zugriff auf Konfigurationsdaten aus dem Anwenderprogramm darauf (z.B. in einem allgemeinen FB), dass abhängig vom TO-Typ das jeweilige Konfigurationsdatum auch vorhanden ist.

Zugriff auf Konfigdaten bei TO-Referenzen und Achsarrays

Der Zugriff auf Konfigdaten bei TO-Referenzen ist möglich. Für den Zugriff auf Konfigdaten eines Achsarrays müssen Sie eine Zwischenvariable verwenden. Falls Sie versuchen Achsarrays direkt zu verwenden erhalten Sie eine Compiler-Meldung, dass eine Zwischenvariable benutzt werden soll.

Beispiel

```
Vartemp
    myPosAxis  :posaxis;
End_Var
myPosAxis      :=Pos[2]

myPosAxis.setconfigdata.TypeOfAxis.MaxJerk.maximum :=
    200000.0;
```

Da Sie über eine Referenz direkt auf die Achse zugreifen, ist keine weitere Zuweisung notwendig, um auf den Wert zuzugreifen. Sie müssen nicht myPosAxis wieder pos[2] zuweisen.

Anmerkung

Schreiben auf die Konfigdaten während eines TO-Restarts oder bei einem deaktiviertem TO ist möglich außer bei STOP_DEVICE). Die Werte werden nach dem RESTART übernommen bzw. sind wirksam.

Ersatzwert beziehungsweise letzten Wert von Konfigdaten bei TO-Restart oder bei deaktiviertem TO (ab V4.1)

Der Zugriff auf Konfigdaten ist auch bei RESTART des TO bzw. bei deaktiviertem TO möglich, ohne dass das System in STOP geht. Anstatt die Konfigdaten über die Funktion _getSaveValue auszulesen, können Sie über einen Eintrag in den Konfigdaten (restart.behaviorInvalidSysvarAccess) folgendes konfigurieren:

- letzten Wert auslesen (LAST_VALUE = Voreinstellung)
- Defaultwert auslesen (=Wert bei Laden des Projektes; DEFAULT_VALUE)
- in STOP gehen (STOP_DEVICE)

Beim Schreiben von Konfigdaten außerhalb der gültigen Grenzen geht die CPU in STOP. Ab V4.2 ist Schreiben außerhalb der gültigen Grenzen möglich. Der Grenzwert wird dann übernommen.

4.2.11 Zurücksetzen eines Technologieobjekts

⚠ VORSICHT
Das Zurücksetzen eines Technologieobjekts bricht die aktuelle Bewegung ohne Fehlermeldung ab.

Zur Übernahme von Konfigurationsdaten, die einen Restart des Technologieobjekts erfordern, müssen Sie das Technologieobjekt zurücksetzen. Die Vorgehensweise ist abhängig vom Konfigurationsdatum *restart.restartActivationSetting* des Technologieobjekts:

- Bei der Einstellung *restart.restartActivationSetting* = RESTART_BY_COMMAND:

  Der Restart kann nur mit der entsprechenden Systemfunktion (z. B. *_restartAxis*) erfolgen. Setzen Sie hierzu den Parameter *activateRestart* = ACTIVATE_RESTART.

- Bei der Einstellung *restart.restartActivationSetting* = RESTART_BY_SYSVAR_AND_COMMAND:

  Der Restart kann auf zwei Arten erfolgen:

  – Aufruf der entsprechenden Systemfunktion (z. B. *_restartAxis*). Setzen Sie den Parameter *activateRestart* = ACTIVATE_RESTART.

  – Wertzuweisung an Systemvariable des Technologieobjekts *restartActivation* = ACTIVATE_RESTART. Die Systemvariablen werden initialisiert, das Technologieobjekt verliert alle Zustandsinformationen, wie z. B. Achse referenziert.

  Die Ausführung des Restarts erfolgt immer asynchron. Nach erfolgtem Restart des Technologieobjekts hat diese Systemvariable den Wert NO_RESTART_ACTIVATION.

4.2.12 Verwenden von Technologiepaketen in Bibliotheken

Bibliotheken können auch TO-Funktionen und Zugriffe auf Systemvariablen eines Technologieobjekts enthalten.

Das SIMOTION Gerät und das Technologiepaket, bezüglich derer die Bibliothek übersetzt wird, legen Sie in den Objekteigenschaften der Bibliothek fest:

1. Markieren Sie im Projektnavigator die Bibliothek.
2. Wählen Sie Menü **Bearbeiten > Objekteigenschaften**.
3. Wählen Sie dort das Register **TPs/TOs**.
4. Wählen Sie die SIMOTION Geräte (einschließlich der Versions-Nummer) und die Technologiepakete aus, für welche die Bibliothek übersetzt werden soll.

ACHTUNG
Beachten Sie die Regeln zur Auswahl der SIMOTION Geräte und Technologiepakete in der folgenden Tabelle, um ein Projekt fehlerfrei zu übersetzen!

Die zusätzliche Angabe des verwendeten Technologiepakets in den ST-Quellen der Bibliothek (mit dem USEPACKAGE-Befehl) ist möglich, aber nicht erforderlich.

Tabelle 4- 21 Auswahl der Geräte und Technologiepakete an einer Bibliothek

Auswahl	Beschreibung
Geräteunabhängig	Sie müssen zusätzlich wählen: • die Technologiepakete • die Versionsnummer der ausgewählten Technologiepakete **Beachten Sie:** 1. Die Bibliothek wird ohne Bezug zu einem SIMOTION Gerät und einer Version des SIMOTION Kernels übersetzt. **Es dürfen deshalb nicht verwendet werden:** – Systemfunktionen der SIMOTION Geräte – Systemvariablen der SIMOTION Geräte – Versionsabhängige Systemfunktionen – Konfigurationsdaten der Technologieobjekte 2. Die Bibliothek wird genau zur ausgewählten Version der Technologiepakete übersetzt. Das Verwenden von Systemfunktionen oder -variablen, die in der gewählten Version nicht verfügbar sind, führt beim Übersetzen zu einem Fehler. 3. Soll eine geräteunabhängige Bibliothek für eine andere Version verfügbar sein, muss sie kopiert und unter einem anderen Namen eingefügt werden. Diese Kopie muss unter Angabe einer anderen Version neu übersetzt werden.
SIMOTION Gerät einschließlich Version (Mehrfachauswahl möglich)	Zu Auswahl werden nur die Technologiepakete angezeigt, die bei allen ausgewählten Geräten verfügbar sind. **Beachten Sie:** 1. Die Bibliothek wird für alle ausgewählten Geräte und Technologiepakete (der ausgewählten Geräteversionen) übersetzt. 2. Das Verwenden von Systemfunktionen oder -variablen, die bei einem der gewählten Geräte bzw. im Technologiepaket der jeweiligen Geräteversion nicht verfügbar sind, führt beim Übersetzen zu einem Fehler. 3. Die Bibliothek ist nur für die ausgewählten Geräte und Technologiepakete nutzbar. Wenn Sie die Bibliothek in einer ST-Quelle verwenden, wird deshalb geprüft: – ob die Bibliothek zum SIMOTION Gerät (einschließlich Version) übersetzt ist, das die importierende ST-Quelle enthält. – ob das Technologiepaket, das am SIMOTION Gerät eingestellt und in der ST-Quelle mit dem Befehl USEPACKAGE angegeben ist, mit dem der Bibliothek übereinstimmt. Inkonsistenzen führen zu Fehlern bei der Übersetzung.

4.3 Reaktion auf Störungen und Ereignisse

4.3.1 Störungen und Ereignisse auswerten

Das System SIMOTION hat verschiedene Ablaufebenen, die für die unterschiedliche zeitliche Abfolge von Programmen sorgen. Eine dieser Ablaufebenen ist die Interrupt-Ablaufebene, die bei bestimmten Ereignissen, wie z. B. bei Fehlern, gestartet wird.

Möglichkeiten, auf Störungen und Ereignisse zu reagieren

Es gibt zwei Klassen von Ereignissen, bei denen Tasks der Interrupt-Ablaufebene starten:

- Wenn die Ereignisse anwenderdefiniert sind, spricht man von UserInterruptTasks; die diesen Tasks zugeordneten Programme sind ebenfalls anwenderdefiniert.
- Sind die Ereignisse system- oder technologiebedingt (Fehler am System oder am Technologieobjekt), spricht man von SystemInterruptTasks. Die nachfolgende Tabelle zeigt die zur Verfügung stehenden SystemInterruptTasks.

  Die vom System erzeugten Meldungen nennt man Alarme. Das weitere Verhalten wird in der Alarmkonfiguration festgelegt, siehe unten.

Tabelle 4- 22 Vorgegebene SystemInterruptTasks

SystemInterruptTask	Aufruf bei folgenden Ereignissen
ExecutionFaultTask	Verarbeitungsfehler in Programmen
PeripheralFaultTask	Prozess- und Diagnosealarme von Peripheriebaugruppen
TechnlogicalFaultTask	Alarme, Warnungen und Hinweise von Technologieobjekten
TimeFaultBackgroundTask	Zeitüberlauf bei der BackgroundTask
TimeFaultTask	Zeitüberläufe bei TimerInterruptTasks

Hinweis

Eine weitere Möglichkeit für Rückmeldungen besteht in der Meldungsgenerierung, bei der frei parametrierbare Meldungen in den Programmen beliebig verwendet werden können, z. B. beim Auftreten bestimmter Ereignisse (Vorratsbehälter leer usw.). Näheres siehe Meldungen programmieren (Seite 451).

Alarmkonfiguration

Das Systemverhalten bei technologischen Alarmen können Sie in SIMOTION SCOUT festlegen (Alarmkonfiguration). Sie können wählen zwischen:

- STOP: Übergang in Betriebszustand STOP (Alle System- und AnwenderTasks werden gestoppt.)
- STOP U: Übergang in Betriebszustand STOP U (Nur Anwenderprogramm-Tasks werden gestoppt.)

- STARTE TechnologicalFaultTask: Starten der zugehörigen SystemInterruptTask
- NONE: keine Reaktion

Für jeden Alarm ist eine Reaktion voreingestellt. Details zur Alarmkonfiguration finden Sie unter **Fehlerbehandlung bei Technologieobjekten**.

Wenn Sie STARTE TechnologicalFaultTask wählen, müssen Sie der TechnologicalFaultTask ein Programm zuordnen, das auf den zugehörigen Alarm reagiert.

Alarme haben neben dem konfigurierbaren Verhalten auf den Programmablauf eine Reaktion am Technologieobjekt, siehe Funktionshandbücher SIMOTION MotionControl Technologieobjekte.

4.3.2 Verarbeitungsfehler in Programmen

Das Verhalten bei Verarbeitungsfehlern in Programmen stellen Sie bei der Taskkonfiguration (siehe **Ablaufsystem konfigurieren**) ein.

Dies betrifft z. B. folgende Operationen:

- Ungültige Operation mit Gleitpunktzahlen (z. B. Logarithmus von negativen Zahlen)
- Fehlerhafte Typkonvertierungen
- Division durch Null
- Überschreiten von Feldgrenzen

Die möglichen Fehlerreaktionen sind in der folgenden Tabelle aufgeführt.

Tabelle 4- 23 Fehlerreaktion bei Verarbeitungsfehlern in Programmen

Fehlerreaktion	Taskart	Beschreibung
CPU in STOP	alle Tasks	SIMOTION Gerät geht in den Betriebszustand STOP; die ShutdownTask wird gestartet.
ExecutionFaultTask	Sequentiell (nichtzyklisch)	Die ExecutionFaultTask wird gestartet. Die Task, in welcher der Fehler aufgetreten ist, wird abgebrochen; die abgebrochene Task kann durch den Anwender neu aufgesetzt werden.
	Zyklisch	Die ExecutionFaultTask wird gestartet. Die Task, in welcher der Fehler aufgetreten ist, wird abgebrochen. Nach Beendigung der ExecutionFaultTask geht das SIMOTION Gerät in den Betriebszustand STOP; die ShutdownTask wird gestartet.

Siehe auch

Festlegungen beim Konfigurieren (Seite 314)

4.3.3 Fehler bei Operationen mit Gleitpunktzahlen (FPU-Exceptions)

Ungültige Gleitpunktzahlen

Die Datentypen für Gleitpunktzahlen REAL und LREAL sind mit ihren Bitmustern gemäß der Norm IEEE 754 realisiert. Dementsprechend werden auch folgende Bitmuster für ungültige Gleitpunktzahlen unterstützt:

- Signalisierende NaN (NaNs): Ungültiges Bitmuster (Not a Number), das bei jeder Operation einen Fehler (FPU-Exception) auslöst.
- Stille NaN (NaNq): Ungültiges Bitmuster (Not a Number), das nur bei bestimmten Operationen einen Fehler (FPU-Exception) auslöst.
- Unendlich (Infinity): Bitmuster für + Unendlich oder – Unendlich.

Hinweis

Eine NaN bzw. Infinity können Sie z. B. mit den Systemfunktionen DWORD_TO_REAL, BigByteArray_to_AnyType oder LittleByteArray_to_AnyType aus dem entsprechenden Bitmuster erzeugen.

ACHTUNG
Bei der Anzeige einer Gleitpunktzahl im Engineering-System (z. B. im Symbol-Browser des SIMOTION SCOUT) wird zwischen signalisierenden und stillen NaN nicht unterschieden!

FPU-Exceptions

Auch die Operationen mit Gleitpunktzahlen sind gemäß IEEE 754 realisiert. Wenn bei den nachstehenden Operationen einer der genannten Fehler auftritt, wird eine FPU-Exception ausgelöst:

- Eine beliebige Operation mit einer signalisierenden NAN (NaNs).
- Addition: Beide Summanden sind unendlich, haben aber verschiedene Vorzeichen.
- Subtraktion: Minuend und Subtrahend sind unendlich und haben gleiche Vorzeichen.
- Multiplikation: Ein Faktor ist 0 und der andere unendlich.
- Division:
  - Beide Operanden sind 0 oder unendlich.
  - Divisor ist 0.
- Modulo-Division (x MOD y): x = unendlich oder y = 0.

- Das Argument einer Systemfunktion ist außerhalb des Definitionsbereichs, z. B.:
  - SQRT: Der Radikand ist < 0.
  - LN, LOG: Das Argument ist ≤ 0.
  - EXPT oder Operator "**": Basis ist ≤ 0:

    Ausnahmen ab Version 4.1 des SIMOTION Kernels:

    Basis ist < 0 und Exponent hat ganzzahligen Wert.

    Basis ist = 0 und Exponent ist > 0.
  - SIN, COS, TAN: Das Argument ist unendlich.
  - ASIN, ACOS: Argument ist > 1.
- Konvertierung einer Gleitpunktzahl in eine Ganzzahl bzw. deren zugeordneten Bitdatentyp mit der entsprechenden Systemfunktion (z. B. LREAL_TO_DINT, REAL_VALUE_TO_DWORD):
  - Wertebereich des Zieldatentyps ist überschritten.
  - Argument ist eine signalisierende oder stille NaN (NaNs oder NaNq).
  - Argument ist unendlich.
- Vergleichsoperationen:
  - Mindestens ein Operand ist eine signalisierende oder stille NaN (NaNs oder NaNq).
  - Mindestens ein Operand ist unendlich.
- Überschreiten des Wertebereichs bei Operationen mit gültigen Gleitpunktzahlen

Es wird das für die Verarbeitungsfehler in Programmen (Seite 146) festgelegte Verhalten ausgeführt. Wenn die ExecutionFaultTask aufgerufen wird, ist TSI#executionFaultType = _SC_INVALID_FLOATING_POINT_OPERATION, siehe Taskstartinfo (Seite 153).

Hinweis

Kein Fehler (keine FPU-Exception) wird ausgelöst:

- Bei Operationen mit stillen NaN (NaNq), sofern oben nicht explizit erwähnt.

  Beispielsweise liefert die Addition einer gültigen Gleitpunktzahl zu einer stillen NaN (NaNs) wieder dieselbe stille NaN (NaNs).
- Bei Operationen mit + Unendlich oder – Unendlich, sofern oben nicht explizit erwähnt.

  Beispielsweise liefert die Addition einer gültigen Gleitpunktzahl zu + Unendlich wieder + Unendlich.

4.3.4 Fehler bei Zugriffen auf Systemvariablen und Konfigurationsdaten sowie auf I/O-Variablen für Direktzugriff

Im Folgenden ist das Verhalten beschrieben, wenn Sie auf Systemvariablen, Konfigurationsdaten oder I/O-Variablen mit den üblichen Mechanismen zugreifen (Verwendung des Variablenbezeichners in einem Ausdruck bzw. Variablenzuweisung) und dabei Fehler auftreten, siehe auch Taskstartinfo verwenden (Seite 153).

Fehler bei Systemvariablen und Konfigurationsdaten

Der Zugriff auf Systemvariablen und Konfigurationsdaten ist ab V4.1 SP2 / V4.1 SP3 auch bei RESTART des TO bzw. bei deaktiviertem TO möglich, ohne dass das System in STOP geht.

Über einen Eintrag im Konfigurationsdatum (restart.behaviorInvalidSysvarAccess) können Sie folgendes konfigurieren:

- letzten Wert auslesen (LAST_VALUE = Voreinstellung)
- Defaultwert auslesen (=Wert bei Laden des Projektes; DEFAULT_VALUE)
- in STOP gehen (STOP_DEVICE)

  Die ExecutionFaultTask wird gestartet, die weitere Fehlerreaktion ist abhängig von der Taskart (sequentiell oder zyklisch), in welcher der Fehler auftritt (siehe folgende Tabelle).

  Verhalten beim Start der ExecutionFaultTask bei fehlerhaftem Zugriff auf Systemvariablen und Konfigurationsdaten

Taskart	Beschreibung
Sequentiell (nichtzyklisch)	Die ExecutionFaultTask wird gestartet. Die Task, in welcher der Fehler aufgetreten ist, wird abgebrochen; die abgebrochene Task kann durch den Anwender neu aufgesetzt werden.
Zyklisch	Die ExecutionFaultTask wird gestartet. Die Task, in welcher der Fehler aufgetreten ist, wird abgebrochen. Nach Beendigung der ExecutionFaultTask geht das SIMOTION Gerät in den Betriebszustand STOP; die ShutdownTask wird gestartet.

Ein Schreiben von Systemvariablenwerten außerhalb der gültigen Grenzen wirkt unabhängig vom o. g. Konfigurationsdatum wie STOP_DEVICE. Ab V4.2 wird der Grenzwert geschrieben. Siehe auch Systemvariablen (Seite 135), Allgemeines zum Zugriff auf Systemvariablen und Ein-/Ausgängen (Seite 405) oder Fehler beim Zugriff auf Systemdaten mit _get/_setSafeValue (Seite 184).

Definition von DEFAULT_VALUE und LAST_VALUE

- DEFAULT_VALUE

  Der DEFAULT_VALUE ist der projektierte Wert. Das ist der Wert, der durch einen Download übertragen wird (Systemvariablen und Konfigdaten).

- LAST_VALUE
  - Konfigdaten

    Der letzte Wert kann der konfigurierte Wert sein. Der Wert ist dann äquivalent zu dem der in der Expertenliste unter "aktueller Wert" angezeigt wird.

    ODER

    Der letzte Werte kann der letzte konfigurierte Wert sein. Der Wert ist dann äquivalent zu dem der in der Expertenliste und "nächster Wert" angezeigt wird.

    Je nach angegebenem Konfigdatum kann der eine oder der andere Wert ausgegeben werden.

  - Systemvariablen

    Bei Systemvariablen ist der letzte Wert der aktuell eingestellte und wirksame Wert.

Fehler bei I/O-Variablen (Direktzugriff auf Ein- und Ausgänge)

Die Fehlerreaktion wird bei der Definition der I/O-Variablen festgelegt (siehe **Direktzugriff und Prozessabbild der zyklischen Tasks** im ST-Programmierhandbuch):

- CPU-Stop: Die ExecutionFaultTask wird gestartet. Anschließend geht das SIMOTION Gerät in den Betriebszustand STOP; die ShutdownTask wird gestartet.
- Ersatzwert: Der bei der Definition der I/O-Variablen vorgegebene Ersatzwert wird genommen, die Task wird fortgesetzt.
- Letzter Wert:

  Bei lesendem Zugriff (auf Ein- oder Ausgänge): Der letzte gültige Wert wird übernommen; die Task wird fortgesetzt.

  Bei schreibendem Zugriff (auf Ausgänge): Der Wert wird in die Variable geschrieben. Er wird allerdings erst am Ausgang wirksam, wenn der Ausgang wieder verfügbar ist. Die Task wird fortgesetzt.

In bestimmten Fällen ist es jedoch nötig, auf Fehler gesondert zu reagieren oder von der festgelegten Fehlerreaktion abzuweichen oder durch vorherige Abfragen Fehler zu vermeiden. Hierzu dienen die Funktionen _getSafeValue (Seite 406) , _setSafeValue (Seite 408) und _getInOutByte (Seite 412). Die Funktionen *_getSafeValue* und *_setSafeValue* sind sehr zeitintensiv.

Siehe auch

Konfigurationsdaten (Seite 139)

4.3.5 Fehler beim Bilden des Prozessabbilds

Im Folgenden ist das Verhalten beschrieben, wenn das Prozessabbild mit der zugeordneten Task aktualisiert wird und dabei Peripherie-Zugriffsfehler auftreten. Mögliche Ursachen sind:

- die Peripheriebaugruppe ist nicht vorhanden
- die Peripheriebaugruppe ist ausgeschaltet
- die Verbindung zur Peripheriebaugruppe fehlt oder ist gestört
- die Peripheriebaugruppe meldet Fehler

Das Verhalten ist dann wie folgt:

- Beim Prozessabbild der zyklischen Tasks (siehe **Direktzugriff und Prozessabbild der zyklischen Tasks** in den Programmierhandbüchern)

  Die Fehlerreaktion wird bei der Definition der I/O-Variablen festgelegt:

  – CPU-Stop: Verhalten siehe folgende Tabelle.

  – Ersatzwert: Der bei der Definition der I/O-Variablen vorgegebene Ersatzwert wird genommen, die zyklische Task wird fortgesetzt.

  – Letzter Wert:

    Prozessabbild der Eingänge (Lesen der Eingänge): Wert des Prozessabbilds an der Adresse wird nicht geändert; die zyklische Task wird fortgesetzt.

    Prozessabbild der Ausgänge (Schreiben auf die Ausgänge): Wert wird am Ausgang mit der Adresse erst wirksam, wenn der Ausgang wieder verfügbar ist; die zyklische Task wird fortgesetzt.

- Beim festen Prozessabbild der BackgroundTask (siehe **Zugriffe auf festes Prozessabbild der BackgroundTask** in den Programmierhandbüchern):

  Die Fehlerreaktion ist davon abhängig, ob an derselben Adresse ein Direktzugriff mittels I/O-Variablen definiert wurde:

  – **kein** Direktzugriff ist definiert: Fehlerreaktion immer CPU-Stop, Verhalten siehe folgende Tabelle.

  – Direktzugriff ist definiert: Die bei der Definition der I/O-Variablen festgelegte Fehlerreaktion gilt (siehe oben, wie Prozessabbild der zyklischen Tasks).

Tabelle 4- 24 Fehlerverhalten beim Aktualisieren des Prozessabbilds bei Reaktion CPU-Stop

Ereignis	Beschreibung
Fehler tritt auf	1. Einmalig wird eine kommende Meldung generiert. 2. Wenn kein Programm in der PeripheralFaultTask eingebunden ist, geht das SIMOTION Gerät in den Betriebszustand STOP; die ShutdownTask wird gestartet. 3. Andernfalls: – Die PeripheralFaultTask wird einmalig **sofort** (nicht erst im nächsten IPO-Takt) gestartet: *TSI#interruptId* = _SC_IMAGE_UPDATE_FAILED. *TSI#logBaseAdrIn* bzw. *TSI#logBaseAdrOut* enthält die Adresse, bei welcher der Fehler aufgetreten ist. Siehe Taskstartinfo (Seite 153). – Prozessabbild der Eingänge: dem Wert des Prozessabbilds an der Adresse wird der Ersatzwert zugeordnet. Prozessabbild der Ausgänge: Wert wird am Ausgang mit der Adresse erst wirksam, wenn der Ausgang wieder verfügbar ist – Die zyklische Task, in welcher der Fehler aufgetreten ist, wird fortgesetzt.
Fehler dauert an	• Es werden keine weiteren Meldungen generiert. • Die PeripheralFaultTask wird nicht erneut gestartet. • Prozessabbild der Eingänge: Wert des Prozessabbilds an der Adresse wird nicht geändert. Prozessabbild der Ausgänge: Wert wird am Ausgang mit der Adresse erst wirksam, wenn der Ausgang wieder verfügbar ist.
Fehler verschwindet	1. Einmalig wird eine gehende Meldung generiert. 2. Die PeripheralFaultTask wird einmalig **sofort** (nicht erst im nächsten IPO-Takt) gestartet: *TSI#interruptId* = _SC_IMAGE_UPDATE_OK, siehe Taskstartinfo (Seite 153). 3. Die zyklische Task wird fortgesetzt.

4.3.6 Taskstartinfo verwenden

Zu jeder Task werden in der Taskstartinfo wichtige Informationen zum Start dieser Task gespeichert, z. B.:

- der Startzeitpunkt der Task,
- bei der TechnologicalFaultTask: die auslösende Instanz des Technologieobjekts und die Alarmnummer,
- bei der TimeFaultTask: die TimerInterruptTask, die den Zeitüberlauffehler verursachte.

Innerhalb einer Task können Sie die jeweilige Taskstartinfo dieser Task abfragen. Hierzu verwenden Sie die Systemvariable **TSI#<info>**; wobei **<info>** die jeweilig abzufragende Information ist. Der Inhalt und Umfang der Taskstartinfo sowie die zugehörigen Systemvariablen sind von der jeweiligen Task abhängig (siehe folgende Tabelle).

Sie verwenden die Abfrage der Taskstartinfo vorwiegend bei SystemInterruptTasks (siehe Beispiele **Abfrage von Taskstartinfos in der TechnologicalFaultTask** und **Abfrage von Taskstartinfo in der TimeFaultTask**).

Tabelle 4- 25 Taskstartinfo der Anwenderprogramm-Tasks

Task	Taskstartinfo			Bedeutung
	Bezeichnung	:	Datentyp	
StartupTask				
	TSI#startTime	:	DT	Startzeitpunkt der Task
	TSI#currentTaskId	:	StructTaskId	TaskId der Task
	TSI#cycleTime	:	TIME	projektierte Zykluszeit der Task (= 0, da sequentielle Task)
	TSI#dwuser_1	:	DWORD	reserviert für interne Zwecke
	TSI#dwuser_2	:	DWORD	reserviert für interne Zwecke
MotionTasks				
	TSI#startTime	:	DT	Startzeitpunkt der Task
	TSI#currentTaskId	:	StructTaskId	TaskId der Task
	TSI#cycleTime	:	TIME	projektierte Zykluszeit der Task (= 0, da sequentielle Task)
	TSI#dwuser_1	:	DWORD	reserviert für interne Zwecke
	TSI#dwuser_2	:	DWORD	reserviert für interne Zwecke
BackgroundTask				
	TSI#startTime	:	DT	Zeitpunkt des Zykluskontrollpunkts
	TSI#currentTaskId	:	StructTaskId	TaskId der Task
	TSI#cycleTime	:	TIME	projektierte Zykluszeit der Task (= 0, da nicht äquidistant zyklische Task)
	TSI#dwuser_1	:	DWORD	reserviert für interne Zwecke
	TSI#dwuser_2	:	DWORD	reserviert für interne Zwecke
TimerInterruptTasks				
	TSI#startTime	:	DT	Zeitpunkt des Zykluskontrollpunkts
	TSI#currentTaskId	:	StructTaskId	TaskId der Task
	TSI#cycleTime	:	TIME	projektierte Zykluszeit der Task
	TSI#dwuser_1	:	DWORD	reserviert für interne Zwecke
	TSI#dwuser_2	:	DWORD	reserviert für interne Zwecke

SynchronousTasks				
	TSI#startTime	:	DT	Startzeitpunkt der Task (= Takt, zu dem die Task synchron läuft)
	TSI#currentTaskId	:	StructTaskId	TaskId der Task
	TSI#cycleTime	:	TIME	projektierte Zykluszeit der Task • ServoSynchronousTask: Lageregler-Takt • ServoSynchronousTask_fast: schneller Lageregler-Takt • IPOsynchronousTask: Interpolatortakt IPO • IPOsynchronousTask_fast: schneller Interpolatortakt IPO • IPOsynchronousTask_2: Interpolatortakt IPO_2 • PWMsynchronousTask: PWM-Takt • InputSynchronousTask_1: Takt Input1 • InputSynchronousTask_2: Takt Input2 • PostControlTask_1: Takt Control1 • PostControlTask_2: Takt Control2
	TSI#dwuser_1	:	DWORD	reserviert für interne Zwecke
	TSI#dwuser_2	:	DWORD	reserviert für interne Zwecke
TimeFaultTask				
	TSI#startTime	:	DT	Startzeitpunkt der Task
	TSI#currentTaskId	:	StructTaskId	TaskId der Task
	TSI#cycleTime	:	TIME	projektierte Zykluszeit der Task (= 0, da sequentielle Task)
	TSI#dwuser_1	:	DWORD	reserviert für interne Zwecke
	TSI#dwuser_2	:	DWORD	reserviert für interne Zwecke
	TSI#interruptID	:	UDINT	Auslösendes Ereignis: • _SC_CYCLE_TIMER_OVERFLOW (= 301) Weitere Ereignisse derzeit nicht definiert.
	TSI#taskId	:	StructTaskId	TaskId der TimerInterruptTask, bei der die Überwachung angesprochen hat.

Tabelle 4- 26 Taskstartinfo der Anwenderprogramm-Tasks (Fortsetzung)

Task	Taskstartinfo			Bedeutung
	Bezeichnung	:	Datentyp	
TimeFaultBackgroundTask				
	TSI#startTime	:	DT	Startzeitpunkt der Task
	TSI#currentTaskId	:	StructTaskId	TaskId der Task
	TSI#cycleTime	:	TIME	projektierte Zykluszeit der Task (= 0, da sequentielle Task)
	TSI#dwuser_1	:	DWORD	reserviert für interne Zwecke
	TSI#dwuser_2	:	DWORD	reserviert für interne Zwecke
	TSI#interruptID	:	UDINT	Auslösendes Ereignis: • _SC_BACKGROUND_TIMER_OVERFLOW (= 300) Weitere Ereignisse derzeit nicht definiert.
TechnologicalFaultTask				
	TSI#startTime	:	DT	Startzeitpunkt der Task
	TSI#currentTaskId	:	StructTaskId	TaskId der Task
	TSI#cycleTime	:	TIME	projektierte Zykluszeit der Task (= 0, da sequentielle Task)
	TSI#dwuser_1	:	DWORD	reserviert für interne Zwecke
	TSI#dwuser_2	:	DWORD	reserviert für interne Zwecke
	TSI#alarmNumber	:	DINT	Nummer des ausgelösten Alarms (siehe Beschreibung im Diagnosehandbuch SIMOTION Alarme) Die in der Alarmmeldung ausgegebenen Parameter stehen in *TSI#alarmP1_DINT bis TSI#alarmP5_LREAL* zur Verfügung (z. B.: *TSI#alarmP3_UDINT* ist Parameter 3 mit Datentyp UDINT).
	TSI#toInst	:	ANYOBJECT	Verursachende TO-Instanz, kann mit Funktion *AnyObject_to_Object* gewandelt werden. Auch Vergleich mit der Instanz eines Technologieobjekts ist möglich (siehe Beispiel **Abfrage von Taskstartinfos in der TechnologicalFaultTask**).
	TSI#commandId.low	:	UDINT	CommandId des auslösenden Befehls (niederwertiges Wort)
	TSI#commandId.high	:	UDINT	CommandId des auslösenden Befehls (höherwertiges Wort)

	TSI#alarmP1_DINT TSI#alarmP1_UDINT TSI#alarmP1_LREAL	: : :	DINT UDINT LREAL	Parameter 1 bis 5 (Begleitwert) der Meldung zum ausgelösten Alarm im jeweiligen Datentyp. Beispiel: In *TSI#alarmP3_UDINT* steht Parameter 3 mit Datentyp UDINT zur Verfügung. Die Bedeutung des Parameters können Sie der Beschreibung des Alarms im Diagnosehandbuch SIMOTION Alarme entnehmen. Dort sind die Nummer und der Datentyp des Parameters in folgender Syntax angegeben: */n/%A* Dabei bedeutet: • */n/* : Nummer des Parameters • *%A* : Kürzel für Datentyp – *%d*: DINT – *%X*: UDINT – *%lf* : LREAL Beispiel: */3/%X* bedeutet Parameter 3 mit Datentyp UDINT. Nur die im Diagnosehandbuch SIMOTION Alarme dokumentierten Parameter zum ausgelösten Alarm sind zur Nutzung freigegeben. Für Informationen zu Prozessalarmen, siehe unten unter *Prozessalarme*.
	TSI#alarmP2_DINT TSI#alarmP2_UDINT TSI#alarmP2_LREAL	: : :	DINT UDINT LREAL	
	TSI#alarmP3_DINT TSI#alarmP3_UDINT TSI#alarmP3_LREAL	: : :	DINT UDINT LREAL	
	TSI#alarmP4_DINT TSI#alarmP4_UDINT TSI#alarmP4_LREAL	: : :	DINT UDINT LREAL	
	TSI#alarmP5_DINT TSI#alarmP5_UDINT TSI#alarmP5_LREAL	: : :	DINT UDINT LREAL	

Tabelle 4- 27 Taskstartinfo der Anwenderprogramm-Tasks (Fortsetzung)

Task	Taskstartinfo			Bedeutung
	Bezeichnung	:	Datentyp	
ExecutionFaultTask				
	TSI#startTime	:	DT	Startzeitpunkt der Task
	TSI#currentTaskId	:	StructTaskId	TaskId der Task
	TSI#cycleTime	:	TIME	projektierte Zykluszeit der Task (= 0, da sequentielle Task)
	TSI#dwuser_1	:	DWORD	reserviert für interne Zwecke
	TSI#dwuser_2	:	DWORD	reserviert für interne Zwecke
	TSI#executionFault Type	:	UDINT	Art des Verarbeitungsfehlers im Anwenderprogramm • _SC_DIVISION_BY_ZERO (= 500) Fehler bei Division oder Modulo-Division von Ganzzahlen (Datentyp ANY_INT oder ANY_BYTE): Divisor ist 0.
				• _SC_INVALID_FLOATING_POINT_OPERATION (= 501) Fehler bei Operationen mit Gleitpunktzahlen (FPU-Exception) (Seite 147).
				• _SC_ARRAY_BOUND_ERROR_READ (= 502) Überschreiten der Feldgrenzen bei Lesezugriff
				• _SC_ARRAY_BOUND_ERROR_WRITE (= 503) Überschreiten der Feldgrenzen bei Schreibzugriff
				• _SC_VARIABLE_ACCESS_ERROR_READ (= 504) Fehler beim lesenden Zugriff: – auf eine Systemvariable eines Technologieobjekts: Überschreiten von Feldgrenzen; Zugriff, während gleichzeitig das Technologieobjekt zurückgesetzt wird (RESET); Zugriff auf eine Variable vom Datentyp eines Technologieobjekts, die den Wert TO#NIL hat; – auf eine I/O-Variable mittels Direktzugriff. Siehe Fehler bei Zugriffen auf Systemvariablen und Konfigurationsdaten sowie auf I/O-Variablen für Direktzugriff (Seite 149).

				• _SC_VARIABLE_ACCESS_ERROR_WRITE (= 505) Fehler beim schreibenden Zugriff: – auf eine Systemvariable eines Technologieobjekts: Überschreiten von Feldgrenzen; Zugriff, während gleichzeitig das Technologieobjekt zurückgesetzt wird (RESET); Zugriff auf eine Variable vom Datentyp eines Technologieobjekts, die den Wert TO#NIL hat; – auf eine I/O-Variable mittels Direktzugriff. Siehe Fehler bei Zugriffen auf Systemvariablen und Konfigurationsdaten sowie auf I/O-Variablen für Direktzugriff (Seite 149).
				• _SC_TO_INSTANCE_NOT_EXISTENT (= 506) Zugriff auf nicht vorhandene TO-Instanz: In einer TO-Funktion wird eine Variable vom Datentyp eine Technologieobjekts mit Wert TO#NIL verwendet.
	TSI#taskId	:	StructTaskId	TaskId der Task, in welcher der Verarbeitungsfehler aufgetreten ist.

Tabelle 4- 28 Taskstartinfo der Anwenderprogramm-Tasks (Fortsetzung)

Task	Taskstartinfo			Bedeutung
	Bezeichnung	:	Datentyp	
PeripheralFaultTask				
	TSI#startTime	:	DT	Startzeitpunkt der Task
	TSI#currentTaskId	:	StructTaskId	TaskId der Task
	TSI#cycleTime	:	TIME	projektierte Zykluszeit der Task (= 0, da sequentielle Task)
	TSI#dwuser_1	:	DWORD	reserviert für interne Zwecke
	TSI#dwuser_2	:	DWORD	reserviert für interne Zwecke
	TSI#interruptID	:	UDINT	• _SC_PROCESS_INTERRUPT (= 200) Prozessalarm an Peripheriebaugruppe aufgetreten Weitere Information in folgenden TSI: – TSI#logBaseAdrIn – TSI#logBaseAdrOut – TSI#details – TSI#eventClass – TSI#faultId Für Informationen zu Prozessalarmen, siehe unten unter *Prozessalarme*.
				• _SC_DIAGNOSTIC_INTERRUPT (= 201) Diagnosealarm an Peripheriebaugruppe aufgetreten Weitere Information in folgenden TSI: – TSI#logBaseAdrIn – TSI#logBaseAdrOut – TSI#logDiagAdr – TSI#details – TSI#eventClass – TSI#faultId
				• _SC_STATION_DISCONNECTED (= 202) PROFIBUS DP: Stationsausfall eines DP Slaves PROFINET IO: Stationsausfall eines IO Device Weitere Information in folgenden TSI: – TSI#logDiagAdr – TSI#eventClass – TSI#faultId
				• _SC_STATION_RECONNECTED (= 203) PROFIBUS: Stationswiederkehr eines DP Slaves PROFINET IO: Stationswiederkehr eines IO Device Weitere Information in folgenden TSI: – TSI#logDiagAdr – TSI#eventClass – TSI#faultId

				• _SC_IMAGE_UPDATE_FAILED (= 204) Fehler bei Bilden des Prozessabbilds (bei DP-Slave: in Zusammenhang mit Stationsausfall) Weitere Information in folgenden TSI: – TSI#logBaseAdrIn – TSI#logBaseAdrOut – TSI#ioDiagAdr Siehe Fehler beim Bilden des Prozessabbilds (Seite 151).
				• _SC_PC_INTERNAL_FAILURE (= 205) Fehler des lokalen Controller-Baugruppe Weitere Information in folgenden TSI: – TSI#details

Tabelle 4- 29 Taskstartinfo der Anwenderprogramm-Tasks (Fortsetzung)

Task	Taskstartinfo			Bedeutung
	Bezeichnung	:	Datentyp	
PeripheralFaultTask (Fortsetzung)				
	TSI#interruptID (Fortsetzung)	:	UDINT	• _SC_IMAGE_UPDATE_OK (= 206) Bilden des Prozessabbilds funktioniert wieder (bei DP-Slave: in Zusammenhang mit Stationswiederkehr) Weitere Information in folgenden TSI: – TSI#logBaseAdrIn – TSI#logBaseAdrOut – #logDiagAdr Siehe Fehler beim Bilden des Prozessabbilds (Seite 151).
				• _SC_DP_CLOCK_DETECTED (= 207) Taktsignal erstmalig eingetroffen und gültiges PRM-Telegramm empfangen
				• _SC_DP_SYNCHRONIZATION_LOST (= 208) Mehrfacher Taktausfall oder PLL (Phase-Locked Loop) ausgerastet (im internen Zustand DP_INTERFACES_SYNCHRONIZED). PLL schaltet in ungeregelten Betrieb
				• _SC_DP_SLAVE_SYNCHRONIZED (= 209) PLL in geregelten Betrieb eingerastet
				• _SC_DP_SLAVE_NOT_SYNCHRONIZED (= 210) Mehrfacher Taktausfall oder PLL ausgerastet (im internen Zustand DP_SLAVE_SYNCHRONIZED). PLL bleibt in geregeltem Betrieb
				• _SC_IO_MODULE_SYNCHRONIZED (= 214) z.B. Onboard Messtaster SINAMICS (Control Unit) z.B. TM15 / TM17 High Feature / DO1 mit Telegramm 39x: Synchronisation erreicht. Weitere Information in folgenden TSI: – TSI#logBaseAdrIn – TSI#logBaseAdrOut – TSI#logDiagAdr
				• _SC_IO_MODULE_NOT_SYNCHRONIZED (= 215) z.B. TM15 / TM17 High Feature / DO1 mit Telegramm 39x: Synchronisation ausgefallen. Weitere Information in folgenden TSI: – TSI#logBaseAdrIn – TSI#logBaseAdrOut – TSI#logDiagAdr

				• _SC_PULL_PLUG_INTERRUPT (= 216) PROFINET IO: Ziehen oder Stecken von Baugruppen in einem IO Device Weitere Information in folgenden TSI: – TSI#logBaseAdrIn – TSI#logBaseAdrOut – TSI#eventClass – TSI#faultId – TSI#logDiagAdr
				• _SC_DRIVE_OBJECT_FAULT (= 217) Störungsmeldung vom DO1; Ursache der Störung ist im Störpuffer hinterlegt, kann mit _readDriveFaults ausgelesen werden. Der Alarm muss mit _resetDriveObjectFault quittiert werden. Weitere Informationen in folgenden TSI: – TSI#logBaseAdrIn – TSI#logBaseAdrOut – TSI#logDiagAdr
				• _SC_DRIVE_OBJECT_ALARM (= 218) Warnungsmeldung vom DO1; Warnungen sin in den Warnparametern hinterlegt, können mit _readDriveParameter ausgelesen werden. Der Alarm ist nicht quittierbar. Er wird mit Kommen der ersten und mit Gehen der letzten Warnung ausgelöst (siehe TSI#details). Weitere Information in folgenden TSI: – TSI#logBaseAdrIn – TSI#logBaseAdrOut – TSI#logDiagAdr – TSI#details
	TSI#logBaseAdrIn	:	DINT	Logische Basisadresse bei folgenden *TSI#interruptID*, sofern der Alarm durch einen Eingangsbereich auf der Baugruppe verursacht wurde: • _SC_PROCESS_INTERRUPT (= 200) • _SC_DIAGNOSTIC_INTERRUPT (= 201) • _SC_IMAGE_UPDATE_FAILED (= 204) • _SC_IMAGE_UPDATE_OK (= 206) • _SC_PULL_PLUG_INTERRUPT (= 216) • _SC_IO_MODULE_SYNCHRONIZED (= 214) • _SC_IO_MODULE_NOT_SYNCHRONIZED (= 215) ansonsten _SC_INVALID_ADDRESS (= -1)

Taskstartinfo der Anwenderprogramm-Tasks (Fortsetzung)

Task	Taskstartinfo			Bedeutung
	Bezeichnung	:	Datentyp	
PeripheralFaultTask (Fortsetzung)				
	TSI#logBaseAdrOut	:	DINT	Logische Basisadresse bei folgenden *TSI#interruptID*, sofern der Alarm durch einen Ausgangsbereich auf der Baugruppe verursacht wurde: • _SC_PROCESS_INTERRUPT (= 200) • _SC_DIAGNOSTIC_INTERRUPT (= 201) • _SC_IMAGE_UPDATE_FAILED (= 204) • _SC_IMAGE_UPDATE_OK (= 206) • _SC_PULL_PLUG_INTERRUPT (= 216) • _SC_IO_MODULE_SYNCHRONIZED (= 214) • _SC_IO_MODULE_NOT_SYNCHRONIZED (= 215) ansonsten _SC_INVALID_ADDRESS (= -1)
	TSI#logDiagAdr	:	DINT	Diagnoseadresse eines DP Slaves bzw. IO Device bei folgenden *TSI#interruptID* • _SC_DIAGNOSTIC_INTERRUPT (= 201) • _SC_STATION_DISCONNECTED (= 202) • _SC_STATION_RECONNECTED (= 203) • _SC_IMAGE_UPDATE_FAILED (= 204) • _SC_IMAGE_UPDATE_OK (= 206) • _SC_IO_MODULE_SYNCHRONIZED (= 214) • _SC_PULL_PLUG_INTERRUPT (= 216) • _SC_DRIVE_OBJECT_FAULT (= 217) • _SC_DRIVE_OBJECT_ALARM (= 218) ansonsten _SC_INVALID_ADDRESS (= -1)
	TST#logDiagAdrIoType		DINT	Gibt den IO-Typ einer Diagnoseadresse bei folgenden *TSI#interruptID* • DSC_SVS_DEVICE_INPUT (= 198) • DSC_SVS_DEVICE_OUTPUT (= 199)
	TSI#details	:	DWORD	Detailinformation in Abhängigkeit der TSI#interruptID (siehe Tabelle **Bedeutung der TSI#details**)
	TSI#eventClass	:	UINT	Ereignisklasse in Abhängigkeit der TSI#interruptID Beschreibung in Zusammenhang mit TSI#faultId: Siehe Tabelle **Bedeutung der TSI#eventClass und TSI#faultId**
	TSI#faultId	:	UINT	Störungskennung in Abhängigkeit der TSI#interruptID Beschreibung in Zusammenhang mit TSI#eventClass: Siehe Tabelle **Bedeutung der TSI#eventClass und TSI#faultId**
UserInterruptTasks				
	TSI#startTime	:	DT	Startzeitpunkt der Task
	TSI#currentTaskId	:	StructTaskId	TaskId der Task
	TSI#cycleTime	:	TIME	projektierte Zykluszeit der Task (= 0, da sequentielle Task)
	TSI#dwuser_1	:	DWORD	reserviert für interne Zwecke

	TSI#dwuser_2	:	DWORD	reserviert für interne Zwecke
ShutdownTask				
	TSI#startTime	:	DT	Startzeitpunkt der Task
	TSI#currentTaskId	:	StructTaskId	TaskId der Task
	TSI#cycleTime	:	TIME	projektierte Zykluszeit der Task (= 0, da sequentielle Task)
	TSI#dwuser_1	:	DWORD	reserviert für interne Zwecke
	TSI#dwuser_2	:	DWORD	reserviert für interne Zwecke
	TSI#shutDownInitiator	:	UDINT	Veranlassung für Übergang in STOP: • _SC_MODE_SELECTOR (= 400): Betriebsartenschalter • _SC_DEVICE_COMMAND (= 401): Systemfunktion im Anwenderprogramm • _SC_EXTERNAL_COMMAND (= 402): Kommando in SIMOTION SCOUT • _SC_EXCEPTION (= 403): Exception • _SC_ALARM_CONFIGURATION (= 404) Projektierte Reaktion auf Technologischen Alarm_ Beispiel für Exceptions: • Verarbeitungsfehler in Programmen • Ebenenüberlauf • Ziehen einer zentralen Baugruppe im RUN

Tabelle 4- 30 Bedeutung der TSI#details in Abhängigkeit der TSI#interruptID (PeripheralFaultTask)

TSI#interruptID	Bedeutung der TSI#details	
_SC_PROCESS_INTERRUPT (= 200)	• Alarmdaten der alarmierenden Baugruppe Der Aufbau dieser Daten ist dem Handbuch der Baugruppe zu entnehmen • bei Alarmquelle in einem SIMOTION I-Slave: Aufrufparameter der Systemfunktion *_sendProcessInterrupt()*	
_SC_DRIVE_OBJECT_ALARM (= 218)	• 1 = Warnung kommend • 0 = Warnung gehend	
_SC_DIAGNOSTIC_INTERRUPT (= 201)	DS0 (= Byte 0 - 3 des DS1) der alarmierenden Baugruppe/Station Der Aufbau des DS0 ist dem Handbuch der Baugruppe zu entnehmen.	
_SC_PC_INTERNAL_FAILURE (= 205)	Die Ursache des Alarms wird als ODER-Verknüpfung der nachfolgenden Ursachen (Summe der Hexadezimalwerte) dargestellt.	
	Wert	**Ursache**
	16#00000001	"Fatal Error" des Host-Betriebssystems (Windows Bluescreen - SIMOTION P).
	16#00000002	Temperaturfehler (SIMOTION P, SIMOTION D)
	16#00000004	Temperatur wieder normal (SIMOTION P, SIMOTION D)
	16#00000008	Batteriewarnung (Batteriespannung niedrig, aber noch ausreichend - SIMOTION P, SIMOTION D)
	16#00000010	Batteriespannung wieder normal (SIMOTION P, SIMOTION D)
	16#00000020	Batteriefehler (Batteriespannung zu niedrig – SIMOTION P, SIMOTION D)
	16#00000040	Batteriemodul wurde abgezogen (SIMOTION P, SIMOTION D)
	16#00000080	Lüfter bzw. Doppellüfter ausgefallen (SIMOTION P, SIMOTION D)
	16#00000100	USV im Pufferzustand (SIMOTION P)
	16#00000200	USV Karenzzeit abgelaufen, PC bootet (SIMOTION P)
	16#00000400	USV wieder OK (SIMOTION P)
	16#00000800	USV-Akku Warnung (SIMOTION P)
	16#00001000	USV-Akku Fehler (Akku nicht mehr betriebsbereit – SIMOTION P)
	16#00002000	USV-Akku Warnung (SIMOTION P)
	16#00004000	1 Lüfter in Doppellüfter ausgefallen (SIMOTION P, SIMOTION D)

Tabelle 4- 31 Bedeutung der TSI#eventClass und TSI#faultId in Abhängigkeit der TSI#interruptID (PeripheralFaultTask)

TSI#interruptID	TSI#eventClass	TSI#faultId	Bussystem[1]	Bedeutung
_SC_PROCESS_INTERRUPT (= 200)	16#11	16#41	PROFIBUS DP PROFINET IO P-Bus	Prozessalarm
_SC_DIAGNOSTIC_INTERRUPT (= 201)	16#39[2]	16#42	PROFIBUS DP PROFINET IO P-Bus	Kommender Diagnosealarm
	16#38[3]	16#42	PROFIBUS DP PROFINET IO P-Bus	Gehender Diagnosealarm
_SC_STATION_DISCONNECTED (= 202)	16#39[2]	16#C4	PROFIBUS DP	Stationsausfall eines DP Slaves
		16#CA	PROFINET IO	Systemfehler PROFINET IO[4]
		16#CB	PROFINET IO	Stationsausfall eines IO Device
		16#CC	PROFINET IO	IO Device gestört. Kanaldiagnose oder herstellerspezifische Diagnose steht an.
_SC_STATION_RECONNECTED (= 203)	16#38[3]	16#C4	PROFIBUS DP	Stationswiederkehr eines DP Slaves
		16#CB	PROFINET IO	Stationswiederkehr eines IO Device ohne Fehler
		16#CC	PROFINET IO	Störung der IO Device behoben
		16#CD	PROFINET IO	Stationswiederkehr eines IO Device, aber Fehler: Sollaufbau <> Istaufbau
		16#CE	PROFINET IO	Stationswiederkehr eines IO Device, aber Fehler bei Baugruppenparametrierung
_SC_PULL_PLUG_INTERRUPT (= 216)	16#39[2]	16#51	PROFINET IO	PROFINET IO Modul wurde entfernt oder kann nicht adressiert werden.
		16#61	PROFIBUS DP	PROFINET DP Modul wurde entfernt oder kann nicht adressiert werden.
		16#54	PROFINET IO	PROFINET IO Submodul wurde entfernt oder kann nicht adressiert werden
		16#61	PROFIBUS DP	Station ist gestört bzw. Baugruppe wurde gezogen.
	16#38[3]	16#54	PROFINET IO	PROFINET IO Modul oder Submodul wurde gesteckt, Modultyp OK (Istaufbau = Sollaufbau)
		16#55	PROFINET IO	PROFINET IO Modul oder Submodul wurde gesteckt, aber falscher Modultyp (Istaufbau <> Sollaufbau)
		16#56	PROFINET IO	PROFINET IO Modul oder Submodul wurde gesteckt, aber Fehler bei der Baugruppenparametrierung
		16#58	PROFINET IO	IO Status eines Modul hat sich von BAD nach GOOD geändert

TSI#interruptID	TSI#eventClass	TSI#faultId	Bussystem[1]	Bedeutung
		16#61	PROFIBUS DP	PROFINET DP Modul oder Submodul wurde gesteckt, Modultyp OK
		16#63	PROFIBUS DP	PROFINET DP Modul oder Submodul wurde gesteckt, aber falscher Modultyp bzw. falsche Baugruppe gesteckt

[1] Bussystem, welches die TSI#interruptID mit der angegebenen TSI#eventClass und TSI#faultId meldet

[2] Signalisiert kommendes Ereignis

[3] Signalisiert gehendes Ereignis

[4] Gehendes Ereignis (TSI#eventClass = 16#38) wird für jede vorhandene Station als Stationswiederkehr gemeldet. Je nach Fehlerstatus werden in TSI#faultId die Werte 16#CB, 16#CD oder 16#CE angezeigt.

Das folgende Beispiel zeigt Ihnen, wie Sie in der TechnologicalFaultTask über die Taskstartinfo die auslösende TO-Instanz und die Alarmnummer abfragen. Das Programm *TO_AlarmProg* muss deshalb der TechnologicalFaultTask zugeordnet sein.

Tabelle 4- 32 Beispiel für die Abfrage von Taskstartinfos in der TechnologicalFaultTask

```
PROGRAM TO_AlarmProg
 VAR
 dintVar    : DINT;
 dtVar    : DT;
 END_VAR;
 dtVar:=TSI#startTime;
 dintVar:=TSI#alarmNumber;
    IF TSI#toInst = axis_1 THEN    // vom Ihnen angelegtes TO
        ; // commands
    END_IF;
    IF TSI#alarmNumber = 30002 THEN    // auslösender Alarm
        ; // commands
    END_IF;
END_PROGRAM
```

Für ein weiteres Beispiel, siehe Auswerten im Anwenderprogramm (Seite 182) .

Prozessalarme

Prozessalarme werden über die TSI "_SC_PROCESS_INTERRUPT" angezeigt.

Nach Auslösen eines Prozessalarms durch eine Baugruppe wird der Alarm ausgelesen, die PeripheralFaultTask aufgerufen und der Alarm quittiert. Tritt in dieser Zeit ein Prozessalarm derselben Baugruppe erneut auf, beachten Sie bitte Folgendes:

- Tritt der Prozessalarm im selben Kanal auf, der vorher schon einen Prozessalarm ausgelöst hat, geht der zugehörige Alarm verloren.
- Tritt der Prozessalarm in einem anderen Kanal derselben Baugruppe auf, so kann momentan kein Prozessalarm ausgelöst werden. Dieser Alarm geht jedoch nicht verloren, sondern wird nach Quittierung des ersten Prozessalarms ausgelöst. Das Verhalten ist analog, wenn ein Prozessalarm:
  - Auf einer anderen Baugruppe derselben Station ausgelöst wird
  - Auf einer anderen Baugruppe einer anderen Station ausgelöst wird.

Fehlerbehandlung bei Technologieobjekten 5

5.1 Fehlermöglichkeiten bei Technologieobjekten

Bei der Programmierung von Technologieobjekten können folgende grundsätzlichen Fehlermöglichkeiten auftreten:

- Das Technologieobjekt selbst kann die von der Applikation gewünschte Funktion nicht ausführen bzw. meldet bestimmte Ereignisse oder Zustände zurück:
  → Ein **Technologischer Alarm** wird ausgegeben.

  Informationen zu den einzelnen Alarmen finden Sie in den SIMOTION Referenzlisten.
- Der Befehl an ein Technologieobjekt kann nicht ausgeführt werden:
  → Der **Rückgabewert** des Befehls gibt Auskunft über die Ursache.

  Informationen zu den Rückgabewerten der Befehle finden Sie in den SIMOTION Referenzlisten.
- Fehler bei Zugriffen auf Konfigurationsdaten, Systemvariablen oder I/O-Variablen

  Treten beim Lesen oder Schreiben von Konfigurationsdaten oder Variablen Fehler auf, wird die ExecutionFaultTask aufgerufen.

Siehe auch

Technologische Alarme (Seite 169)

Rückgabewerte von Befehlen (Seite 183)

Fehler beim Zugriff auf Systemdaten mit _get/_setSafeValue (Seite 184)

5.2 Technologische Alarme

Tritt an einem Technologieobjekt ein Ereignis (Fehler, Hinweis) auf, so setzt dieses einen **Technologischen Alarm** ab.

Maximal können gleichzeitig 160 TO-Alarme im System gespeichert werden. Dabei werden je TO identische Alarme wie ein Alarm gezählt. Wenn dieser Puffer überläuft, weil gleichzeitig mehr als 160 Alarme anstehen, geht das System mit dem Diagnoseeintrag **Meldepufferüberlauf** in den Betriebszustand STOP. Der Füllstand des Puffers kann vom Anwender nicht ausgelesen werden.

Treten eine Reihe von gleichen Alarmen hintereinander auf, werden die Detailinformationen nur am ersten Alarm angezeigt. Bei den folgenden Alarmen werden die Informationen erst angezeigt, wenn der zuerst aufgetretene Alarm quittiert wurde.

Auswirkungen von Alarmen

Technologische Alarme lösen Folgereaktionen im System aus. Es wird unterschieden zwischen:

- Auswirkungen auf das betroffene Technologieobjekt selbst: **Lokales Verhalten**
- Auswirkungen auf andere Technologieobjekte bzw. dem Ablaufsystem: **Globales Verhalten**

Für jeden Alarm sind bestimmte Auswirkungen voreingestellt. Sie können diese Einstellungen jedoch für Ihre Anforderungen anpassen.

- Durch Angabe der **Fehleraktivierung** können Sie bestimmen, ob der Alarm sofort, nach mehrmaligem Auftreten oder nach einer bestimmten Zeit ausgelöst werden soll.
- Einige Alarme können Sie ausblenden. Damit ist es z. B. möglich, eine für Sie unwichtige Hinweismeldung zu unterdrücken.

Alarmgruppenzugehörigkeit

Die Alarmnummern der TO sind einer Alarmguppe zugeordnet. Dabei gibt es Alarmgruppen, die für jeden TO-Typ definiert sind und Alarmgruppen, die TO-spezifisch sind. Beispielsweise sind die Alarmnummern 20006 und 20011 bei einem TO der Alarmgruppe "Konfigurationsfehler" zugeordnet.

Über die Systemvariable "errorGroup" wird die Alarmgruppenzugehörigkeit der anstehenden technologischen Alarme angezeigt. Jeder Alarmgruppe ist ein Bit dieser Variable zugewiesen. Die Alarmgruppe "Konfigurationsfehler" ist das Bit 1 (niederwertigstes Bit ist Bit 0) zugeordnet. Ist Bit 1 gesetzt, so heißt dies, dass ein Alarm der Gruppe "Konfigurationsfehler" am TO ansteht. Die folgende Tabelle listet die verschiedenen Alarmgruppen auf.

Bezeichnung	Bitnummer	Bedeutung
SYSTEM_FAULT	0	Systemfehler
CONFIG_FAULT	1	Konfigurationsfehler
USER_FAULT	2	Benutzerfehler
PERIPHERICAL_FAULT	3	Peripheriefehler
FUNCTION_FAULT	4	Funktions-/Befehlsfehler
FUNCTION_ABORTED	5	Funktions-/Befehlsabbruch
RESET_RESTART_FAULT	6	Reset-/Restartfehler
DISTRIBUTED_MOTION_FAULT	7	Fehler bei verteiltem Gleichlauf
FOLLOWING_ERROR	8	Dynamische Schleppabstandsüberwachung
STANDSTILL_POSITIONING_ERROR	9	Stillstands-/Positionierüberwachungsfehler
DYNAMIC_LIMIT	10	Begrenzung von Geschwindigkeit, Beschleunigung oder Ruck
CLAMPING_ERROR	11	Klemmüberwachungsfehler
SOFTWARE_LIMIT	12	Software-Endlagenschalter
LIMIT_SWITCH	13	Hardware-Endlagenschalter
SENSOR_FAULT	14	Geberfehler

Bezeichnung	Bitnummer	Bedeutung
REFERENCE_NOT_FOUND	15	Referenznocken-/Nullmarkenfehler
OUTPUT_LIMIT	16	Ausgangsgrößenbegrenzung
FORCE_DYNAMIC_LIMIT	17	Begrenzung von Kraft und/oder Druck
ADDITIONAL_SENSOR_FAULT	18	Fehler an zusätzlichem Geber
SYNCHRONOUS_MOTION_FAULT	19	Gleichlauffehler
FOLLOWING_OBJECT	20	Gleichlaufobjektfehler
PATH_SYNCHRONOUS_MOTION_FAULT	21	Bahngleichlauffehler
PATH_MOTION_FAULT	22	Bahnbewegungsfehler
PATH_OBJECT	23	Bahnobjektfehler

Siehe auch

Lokales Verhalten (Seite 171)

Globales Verhalten (Seite 172)

Fehleraktivierung (Seite 172)

Technologische Alarme konfigurieren (Seite 174)

Technologische Alarme anzeigen und quittieren (Seite 176)

Quittieren über Anwenderprogramm (Seite 178)

Auswerten im Anwenderprogramm (Seite 182)

5.2.1 Lokales Verhalten

Mit dem *lokalen Verhalten* legt man fest, wie sich das betroffene Technologieobjekt beim Auftreten des Alarms verhalten soll und wie mit weiteren Befehlen für das TO umgegangen wird.

Die Alarmreaktionen sind priorisiert. Zu einem Zeitpunkt kann genau eine Reaktion aktiv sein. Diese entspricht der höchstpriorisierten Reaktion der zu diesem Zeitpunkt anstehenden Alarme. Bei Auftreten einer Reaktion (außer NONE) wird immer der Befehlsdecoder gestoppt. Alle dann noch programmierten Befehle werden abgewiesen. Ein Weiterarbeiten ist nach Quittieren des Alarms erreichbar, falls die Alarme nicht automatisch über die globale Fehlerreaktion ein Power-On erfordern.

Abhängig vom jeweiligen TO und TO-Alarm sind spezifische Reaktionsmöglichkeiten einstellbar.

Informationen zum spezifischen lokalen Verhalten finden Sie in den Funktionshandbüchern der betreffenden Technologieobjekte.

5.2.2 Globales Verhalten

Mit dem *globalen Verhalten* wird die Auswirkung eines TO-Alarms auf das Ablaufsystem beschrieben. Abhängig vom jeweiligen TO-Alarm sind die nachfolgenden Reaktionsmöglichkeiten einstellbar.

- **NONE**

  Keine Reaktion des Systems beim Auftreten des Alarms.

- **START TechnologicalFaultTask**

  Es wird die **TechnologicalFaultTask** gestartet. Programme, die dieser Task zugeordnet sind, werden gestartet. Damit kann der Anwender auf den TO-Alarm applikativ reagieren. Wenn dieser Task kein Programm zugeordnet ist, geht das System in den Zustand STOP.

- **STOP**

  Das System wechselt in den Betriebszustand **STOP**. Im Zustand STOP sind alle Technologieobjekte inaktiv, das Anwenderprogramm wird nicht bearbeitet und alle Ausgänge befinden sich im Zustand 0 (Null). Es sind noch alle Systemdienste aktiv und es können Anwenderprogramme geladen werden.

- **STOPU**

  Das System wechselt in den Betriebszustand **STOPU**. Die Programmbearbeitung wird abgebrochen, die Achsfreigabe und damit auch die Lageregelung wird deaktiviert. Die Technologieobjekte sind weiterhin aktiv und können noch Aufträge für Test- und Inbetriebnahmefunktionen ausführen. Sonst gleiches Verhalten wie im Betriebszustand STOP.

5.2.3 Fehleraktivierung

Art der Fehleraktivierung (TypeOfActivation)

Alarme mit ihren entsprechenden Alarmreaktionen können auf unterschiedliche Weise ausgelöst werden. Je nach Fehlertyp ist die Fehleraktivierung auf einen der Werte voreingestellt.

Tabelle 5- 1 Art der Fehleraktivierung

Parameter TypeOfActivation	Bedeutung
ACTIVATE_IMMEDIATELY	Der Alarm wird sofort beim Auftreten des Fehlers aktiviert.
ACTIVATE_AFTER_NTIME	Der Alarm wird nach dem n-maligen Auftreten des Fehlers aktiviert. Die Zahl n der Fehlerwiederholungen wird im Parameter Continue eingestellt, siehe Tabelle Fehlerwiederholbarkeit.
ACTIVATE_AFTER_NTIME_CONSECUTIVE	Der Alarm wird nach der Zeit t aktiviert. Dabei muss der Fehler über die gesamte Zeit t ohne Unterbrechung anstehen. Die Zeit wird im Parameter Time eingestellt, siehe Tabelle Fehlerreaktionszeit.

Fehlerwiederholbarkeit (Continue)

Tabelle 5- 2 Fehlerwiederholbarkeit

Parameter Continue	Bedeutung
NO_TIME	Der Alarm wird sofort beim Auftreten des Fehlers aktiviert.
TIME_n	Der Alarm wird nach dem n-maligen Auftreten des Fehlers aktiviert.

Der Parameter wirkt nur bei der Einstellung des Parameters **TypeOfActivation** auf den Wert ACTIVATE_AFTER_NTIME.
Für TIME_n können je nach Alarm die Werte TIME_10, TIME_100 und TIME_1000 eingestellt werden.

Fehlerreaktionszeit (Time)

Tabelle 5- 3 Fehlerreaktionszeit

Parameter Time	Bedeutung
NO_TIME	Der Alarm wird sofort beim Auftreten des Fehlers aktiviert.
TIME_n	Der Alarm wird nach einem zeitlich ununterbrochenen Anstehen des Fehlers von n Millisekunden ausgelöst.

Der Parameter wirkt nur bei der Einstellung des Parameters **TypeOfActivation** auf den Wert ACTIVATE_AFTER_NTIME_CONSECUTIVE.
Für TIME_n können je nach Alarm die Werte TIME_1, TIME_10, TIME_100 und TIME_1000 eingestellt werden.

Typ

Im Parameter **Typ** können Sie für bestimmte TO-Alarme festlegen, ob sie angezeigt werden sollen. Wenn Sie den Eintrag *ausgeblendet* wählen, wird beim Auftreten dieses Alarms keine Alarmmeldung angezeigt und kein Eintrag in den Diagnosepuffer geschrieben. Dadurch können Sie einen Überlauf des Alarmpuffers beim häufigen Auftreten eines bestimmten Technologischen Alarms verhindern bzw. eine für Sie unwichtige Hinweismeldung unterdrücken.

5.2.4 Technologische Alarme konfigurieren

Für jeden Alarm sind individuelle Reaktionen voreingestellt. Sie können diese Voreinstellung durch die folgende Vorgehensweise ändern:

1. Wählen Sie im Projektnavigator den Pfad **Ablaufsystem** an. Es öffnet sich das Fenster **Ablaufsystem**. Wählen Sie den Pfad **SystemInterruptTasks → TechnologicalFaultTask** an. Klicken Sie anschließend im Fenster auf den Button **Alarmkonfiguration**.

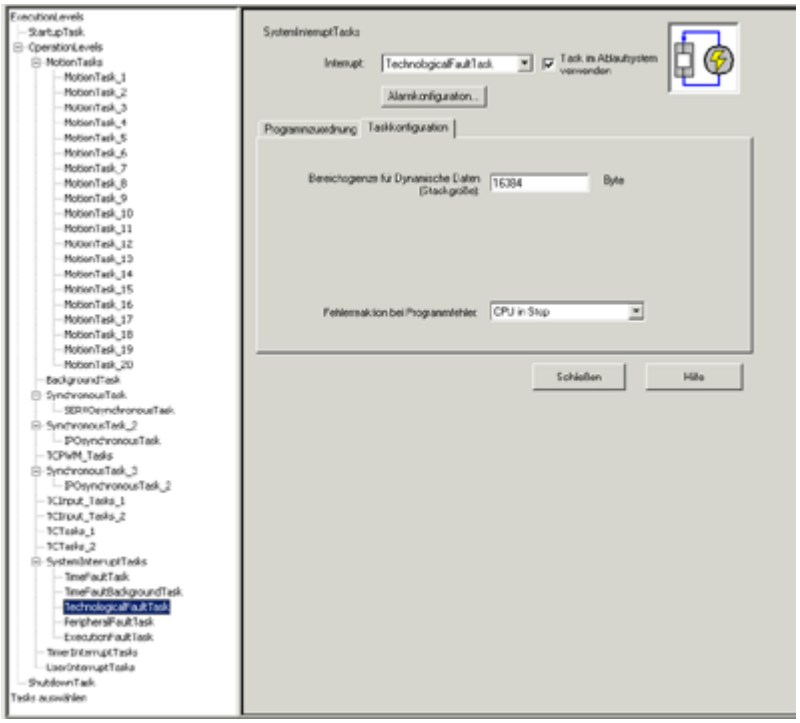


Bild 5-1 Technologischen Alarm konfigurieren

2. Wählen Sie im Kombinationsfeld das Technologieobjekt, dessen Alarme Sie konfigurieren wollen. Die Alarme für das Technologieobjekt werden angezeigt.

3. Wählen Sie den Alarm aus, dessen Reaktion Sie ändern wollen.
4. Wählen Sie in der Auswahlliste für den entsprechenden TO-Alarm die gewünschten Reaktionen aus. Die angebotenen Auswahlmöglichkeiten sind abhängig vom jeweiligen Alarm.

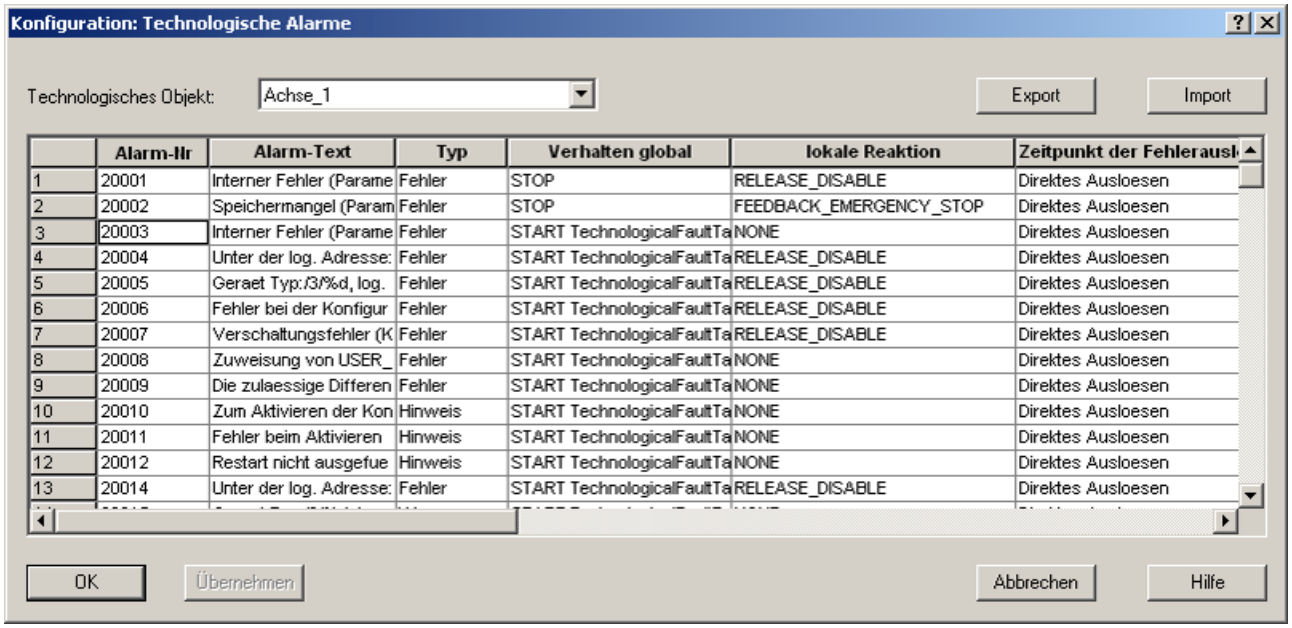


Bild 5-2 Auswahl eines Technologischen Alarms

Hinweis

Das Technologieobjekt, dessen Alarme Sie projektieren wollen, muss bereits konfiguriert sein!

Die Alarmkonfiguration kann über die entsprechenden Tasten im Dialog **Alarmkonfiguration** auf andere Technologieobjekte übertragen werden (über Export/Import).

Technologische Alarme exportieren bzw. importieren

Hinweis

Damit alle Änderungen, die Sie im Dialog **Konfiguration:Technologische Alarme** durchgeführt haben, mit exportiert werden, bleibt die Schaltfläche **Export** so lange inaktiv, bis Sie mit **Übernehmen** alle Änderungen übernommen haben.

So exportieren Sie alle TO Alarme im XML-Format:

1. Öffnen Sie den Dialog **Konfiguration: Technologische Alarme**.
2. Klicken Sie Export, um den Dialog **Alarm Konfiguration exportieren** aufzurufen.
3. Wählen Sie einen Pfad für den Export aus und bestätigen Sie mit **OK**.
   Die Alarme werden dort gespeichert.

So importieren Sie TO Alarme im XML-Format:

1. Klicken Sie im Dialog **Konfiguration: Technologische Alarme** auf Import.
   Der Dialog **Alarm Konfiguration Import** wird aufgerufen.
2. Wählen Sie unter **Quellpfad und Quellname des Imports** die gewünschte XML-Datei aus.
3. Klicken Sie **OK**, um die Daten zu importieren. Die Alarme werden dann im Dialog angezeigt.

Meldungen (Alarme) für alle TOs eines bestimmten TO--Typs konfigurieren

Sie können eine Alarmkonfiguration allen TOs eines bestimmten Typs zuweisen. Beispielsweise können Sie eine Konfiguration allen Positionierachsen zuweisen.

1. Öffnen Sie den Dialog **Konfiguration: Technologische Alarme.**
2. Passen Sie die Konfiguration der Alarme an. Sobald Sie die Einstellungen mindestens eines Alarms geändert haben, werden die Einträge in der Liste **Technologisches Objekt** um die TO-Typen, z. B. Positionierachsen, erweitert.
3. Wählen Sie in der Liste den TO-Typ aus.
4. Klicken Sie auf **OK**, um die Einstellungen allen TOs des ausgewählten Typs zuzuweisen.

5.2.5 Technologische Alarme anzeigen und quittieren

Technologische Alarme können über verschiedene Wege ausgewertet und quittiert werden:

- Im **Online-Modus** des SIMOTION SCOUT werden Alarme und Meldungen in der Detailanzeige der Workbench im Register **Alarme** angezeigt.

  Durch **Quittieren** werden alle Alarme des jeweiligen Typs gelöscht.
- Die Alarme können über das **H**uman **M**achine **I**nterface (HMI) ausgeben, angezeigt und quittiert werden.
- Alle anstehenden bzw. einzeln ausgewählten Alarme eines Technologieobjektes können auch über das Anwenderprogramm abgefragt, ausgewertet und quittiert werden.

ACHTUNG
Sollte genau während des Quittierens von TO-Alarmen, ein neuer Alarm am TO auftreten, der die gleiche oder niedrigere Priorität hat, wird der neue Alarm nicht mehr im System gemeldet. Dieses Verhalten ist bei der Fehlerquittierung zu berücksichtigen.

Quittieren über SIMOTION SCOUT

1. Markieren Sie den Alarm im Register **Alarme** der Detailanzeige
2. Klicken Sie auf **Quittieren**.

Danach werden alle Alarme des jeweiligen Typs gelöscht.

Hinweis

Da Antriebsalarme meist auch TO-Alarme erzeugen, wird mit dem Schalter **Quittieren (TO)** versucht auch die Antriebsalarme zu löschen. Falls aber der Grund eines Antriebsalarms bestehen bleibt, wird sofort ein neuer TO-Alarm ausgelöst. Beseitigen Sie dann zuerst den Grund des Antriebsalarms.

Anzeigen und Quittieren über HMI

1. Anbindung über WinCC flexible

   Die Anzeige erfolgt in einer projektierten Meldezeile oder in Meldefenstern.

   Das Quittieren erfolgt bei den WinCC-Geräten über die ACK-Taste bzw. über anwenderprojektierte Softkeys oder Schaltflächen.

   (Handhabung, siehe Beschreibung **WinCC flexible**)

2. Anbindung über OPC

   Ab SimaticNet V6.0SP4 OPC Alarms & Events können die Alarme angezeigt und quittiert werden. Außerdem kann der Diagnosepuffer ausgelesen und ggf. quittiert werden.

   (Handhabung siehe Produktinformation **Ethernetbasierende HMI und Diagnosefunktionen**, die Sie auf der SIMOTION SCOUT CD Dokumentation finden)

5.2.6 Quittieren über Anwenderprogramm

Alle anstehenden TO-Alarme quittieren

_resetTechnologicalErrors → Quittieren aller aktuell anstehenden TO-Alarme

ST-Aufrufbeispiel: Alle anstehenden Alarme quittieren

```
UNIT ST_1;
INTERFACE
    USEPACKAGE CAM;
    PROGRAM EXAMPLE;
END_INTERFACE
IMPLEMENTATION
 PROGRAM EXAMPLE
    VAR
       s_i_RetVal : DINT;
    END_VAR;
    (* Quittiere alle TO-Alarme *)
    s_i_RetVal:= _resetTechnologicalErrors();

 END_PROGRAM
END_IMPLEMENTATION
```

MCC-Aufruf: Alle anstehenden Alarme quittieren

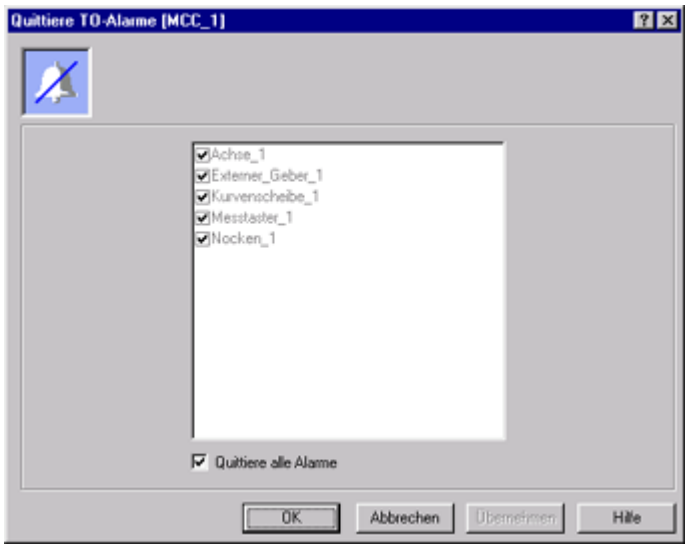


Bild 5-3 MCC-Aufruf: Alle anstehenden Alarme quittieren

Alle anstehenden Alarme eines TO quittieren

ST-Aufrufbeispiel: Alle Alarme an einem TO-Achse, TO-Messtaster und TO-Nocken quittieren

```
UNIT ST_1;
INTERFACE
    USEPACKAGE CAM;
 PROGRAM EXAMPLE;
END_INTERFACE
IMPLEMENTATION
 PROGRAM EXAMPLE
    VAR
        s_i_RetVal : DINT;
    END_VAR;
     (* Quittiere TO-Alarme ('ResetTOAlarms') *)
    s_i_RetVal:= _resetAxisError(axis:=Achse_1);
    s_i_RetVal:= _resetMeasuringInputError(
    measuringInput:=Messtaster_1);
    s_i_RetVal:= _resetOutputCamError(outputCam:=Nocken_1);
 END_PROGRAM
END_IMPLEMENTATION
```

MCC-Aufruf: Alle Alarme an einem TO-Achse, TO-Messtaster und TO-Nocken quittieren

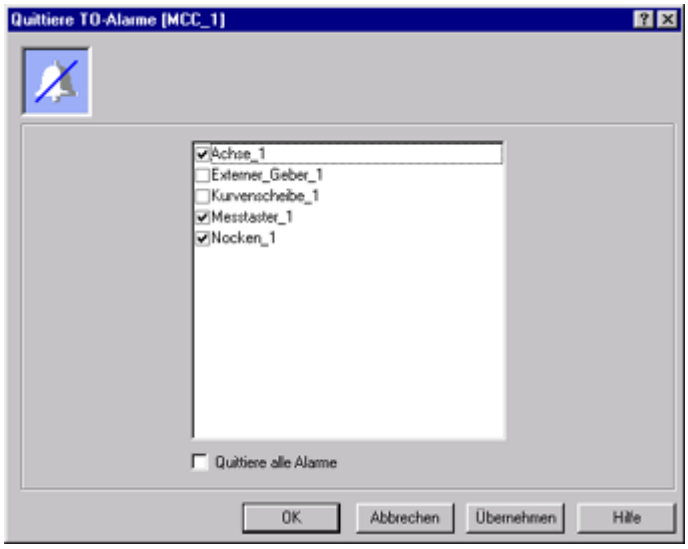


Bild 5-4 MCC-Aufruf: Alle Alarme an einem TO-Achse, TO-Messtaster und TO-Nocken quittieren

Spezifischen Alarm eines TO quittieren

Durch Ergänzen der _**reset..**-Befehle um die Schalter **errorResetMode** := SPECIFIC_ERROR und **errorNumber** := XXXXX kann gezielt ein bestimmter Alarm quittiert werden.

Beispiel:

_resetAxisError (..., **errorResetMode** := SPECIFIC_ERROR, **errorNumber** := 30002)

ST-Aufrufbeispiel: Quittieren des Alarms 30002 an einem TO-Achse

```
UNIT ST_1;
INTERFACE
 USEPACKAGE CAM;
 PROGRAM EXAMPLE;
END_INTERFACE
IMPLEMENTATION
 PROGRAM EXAMPLE
    VAR
        s_i_RetVal : DINT;
    END_VAR;
    (* Quittiere spezifischen TO-Alarm ('ResetSingleTOAlarm') *)
    s_i_RetVal:= _resetAxisError(axis:=Achse_1,
    errorResetMode:=SPECIFIC_ERROR,
    errorNumber:=30002);
 END_PROGRAM
END_IMPLEMENTATION
```

MCC-Aufruf: Quittieren des Alarms 30002 an einem TO-Achse

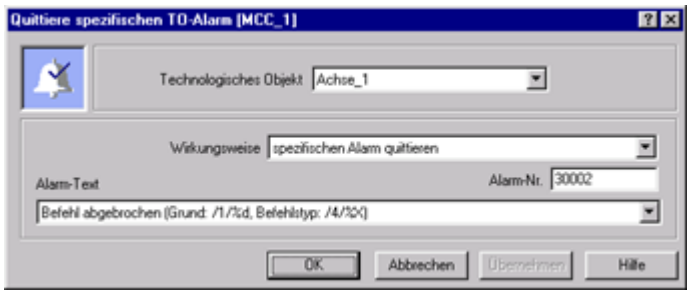


Bild 5-5 MCC-Aufruf: Quittieren des Alarms 30002 an einem TO-Achse

Rücksetzen eines TO

Das TO wird in den Grundzustand versetzt und alle anstehenden Alarme werden quittiert.

Hinweis

Die Befehle zum Rücksetzen haben neben der Fehlerbehebung auch andere Auswirkungen auf das TO. Die Fehlerquittierung sollte daher typischerweise mit den Befehlen **_reset...Error** erfolgen.

Im Unterschied zum **_reset...Error** Befehl überführen die **_reset...** Befehle das jeweilige TO zusätzlich in einen sicheren Zustand. Neben dem Quittieren der Alarme werden bei jedem TO-Typ folgende Aktionen ausgeführt:

- Beenden der aktiven Befehle
- Löschen der Befehlspuffer
- Rücksetzen von Systemvariablen (Parameter userDefaultData)
- Ausführen eines TO Restart (Parameter activateRestart

Zusätzlich werden vom TO-Typ abhängige Aktionen ausgeführt:

- Achsen: Generieren einer Bremsrampe
- Nocken und Nockenspur: Abschalten des Hardwareausgangs
- Kurvenscheibe: Löschen der Kurvengeometrie
- Bahnobjekt: Anhalten des Bahnverbundes

Aufrufbeispiel: TO-Achse, TO-Messtaster und TO-Nocken zurücksetzen

```
UNIT ST_1;
INTERFACE
 USEPACKAGE CAM;
 PROGRAM EXAMPLE;
END_INTERFACE
IMPLEMENTATION
 PROGRAM EXAMPLE
    VAR
        s_i_RetVal : DINT;
    END_VAR;
    (* Setze Objekt zurück ('ResetObject') *)
    s_i_RetVal := _resetAxis(axis:=Achse_1,
    userDefaultData:=DO_NOT_CHANGE);
    s_i_RetVal := _resetMeasuringInput(
    measuringInput:=Messtaster_1,
    userDefaultData:=DO_NOT_CHANGE);
    s_i_RetVal := _resetOutputCam(outputCam:=Nocken_1,
    userDefaultData:=DO_NOT_CHANGE);
 END_PROGRAM
END_IMPLEMENTATION
```

5.2.7 Auswerten im Anwenderprogramm

Bei Alarmen mit dem projektierten globalen Verhalten **StartTechnologicalFaultTask** wird beim Auftreten, mit jedem Alarm einmal, die **TechnologicalFaultTask** aufgerufen. In dieser Task kann die anstehende Alarmnummer und das auslösende TO abgefragt werden. Die Informationen werden der **TechnologicalFaultTask** über die Task-Start-Info (TSI) mitgegeben.

Sie können in der **TechnologicalFaultTask** ein Programm einhängen und abhängig vom auftretenden Alarm eine individuelle Fehlerreaktion programmieren oder z. B. die Alarme abfangen und an eine übergeordnete Alarmauswertung weitermelden.

Hinweis

Wenn Sie kein Programm in die **TechnologicalFaultTask** einhängt haben, geht die CPU in **STOP**, wenn die Task durch einen Alarm aufgerufen wird.

Folgende Parameter werden in **TechnologicalFaultTask** zur Auswertung übergeben:

TSI#startTime	→	Zeitpunkt, an dem der Alarm registriert wurde.
TSI#alarmNumber	→	Alarmnummer
TSI#toInst	→	Name des Technologieobjekts, das den Alarm ausgelöst hat (z. B. Achse_1)

Programmbeispiel

Immer wenn an der Achse_1 der **Alarm 30002** aufgetreten ist soll ein Zähler (**s_i_Count**) erhöht werden und der Alarm automatisch wieder quittiert werden.

Hinweis: Dieses Beispielprogramm muss im Ablaufsystem in die **TechnologicalFaultTask** eingehängt werden!

```
UNIT ST_1;
INTERFACE
   USEPACKAGE CAM;
   PROGRAM TO_AlarmProg;
END_INTERFACE
IMPLEMENTATION
   PROGRAM TO_AlarmProg
      VAR
         s_i_Count : INT;
         s_i_RetVal: DINT;
      END_VAR;

      (*Abfrage ob der Alarm 30002 für die Achse_1 ansteht*)
      IF (TSI#alarmNumber = 30002) AND
                            (TSI#toInst = Achse_1) THEN
         (*Zähler inkrementieren*)
         s_i_Count:= s_i_Count + 1;
         (* Quittiere spezifischen TO-Alarm
                                  ('ResetSingleTOAlarms') *)
         s_i_RetVal := _resetAxisError(axis:=Achse_1,
                                  errorResetMode:=SPECIFIC_ERROR,
                                  errorNumber:=30002);
      END_IF;
   END_PROGRAM
END_IMPLEMENTATION
```

5.3 Rückgabewerte von Befehlen

Ein bearbeiteter Befehl zeigt Ihnen im Anwenderprogramm an, ob er zur Laufzeit erfolgreich ausgeführt werden konnte oder nicht.

Eine entsprechende Fehlerinformation erhalten Sie im Rückgabewert des Bausteins. Diese Fehlerinformationen müssen nicht quittiert werden.

Auswerten des Rückgabewertes

Ein Befehl zeigt durch den Wert 0 (Null) im Rückgabewert an, dass bei der Bearbeitung kein Fehler aufgetreten ist. Wenn ein Fehler aufgetreten ist, wird im Rückgabewert eine Fehlernummer zurückgegeben, mit der die Ursache ermittelt werden kann.

Die möglichen Fehlernummern der jeweiligen Befehle finden sie in den SIMOTION Referenzlisten.

Sie können im Anwenderprogramm auf mögliche Fehler in der Bearbeitung der Systemfunktion reagieren und somit Folgefehler vermeiden.

Auch bei MCC-Befehlen kann über die Angabe einer Rückgabevariablen mit dem Rückgabewert des Befehls eine spezifische Fehlerreaktion ausprogrammiert werden (siehe Programmierhandbuch MCC, Kapitel Register Experte).

Aufrufbeispiel

Wenn beim Abarbeiten des Befehls **_pos** ein Fehler auftritt, soll die Variable **g_bo_error** auf **true** gesetzt, der Rückgabewert in den Parameter **g_i_errornumber** eingetragen und der Baustein abgebrochen werden.

Hinweis: Dieses Beispielprogramm muss im Ablaufsystem in eine **MotionTask** eingehängt werden!

```
UNIT ST_1;
INTERFACE
    USEPACKAGE CAM;
        VAR_GLOBAL
            g_bo_error: BOOL;
            g_i_errornumber: DINT;
            g_i_RetVal: DINT;
        END_VAR
    PROGRAM RETURN_VALUE ;
END_INTERFACE
IMPLEMENTATION
    PROGRAM RETURN_VALUE

         (* Positioniere Achse ('Pos') *)
        g_i_RetVal:= _pos(axis:=Achse_1,
        direction:=SHORTEST_WAY,
        positioningMode:=ABSOLUTE,
        position:=222,
        velocityType:=DIRECT,
```

```
            velocity:=1000,
            velocityProfile:=TRAPEZOIDAL,
            blendingMode:=INACTIVE,
            mergeMode:=IMMEDIATELY,
            nextCommand:=WHEN_MOTION_DONE,
            commandId:=_getCommandId());
             (*Auswertung des Rückgabewertes *)
            IF g_i_RetVal <> 0 THEN
                g_i_errornumber := g_i_RetVal;
                g_bo_error := true;
                Return;  // Baustein beenden
            END_IF;
        // Ab hier weiteres Anwenderprogramm.
        END_PROGRAM
    END_IMPLEMENTATION
```

5.4 Fehler beim Zugriff auf Systemdaten mit _get/_setSafeValue

Fehler bei Zugriffen auf Konfigurationsdaten, Systemvariablen oder I/O-Variablen

Ab V4.1.3 sind diese Systemfunktionen für Systemvariablen und Konfigdaten nicht unbedingt erforderlich. Bei Zugriffsfehlern (z. B. TO im Restart) kann ein "Ersatzwert" oder "letzter Wert" konfiguriert werden, siehe Systemvariablen (Seite 135) oder Konfigurationsdaten (Seite 139).

Treten beim Lesen oder Schreiben von Konfigurationsdaten oder Systemvariablen Fehler auf, wird bei direktem Zugriff die ExecutionFaultTask aufgerufen (siehe Fehler bei Zugriffen auf Systemvariablen und Konfigurationsdaten sowie auf I/O-Variablen für Direktzugriff (Seite 149)).

In bestimmten Fällen ist es jedoch erforderlich, den Aufruf der ExecutionFaultTask zu vermeiden bzw. auf Fehler gesondert zu reagieren oder vom konfigurierten Fehlerverhalten abzuweichen. Hierzu dienen die Funktionen _getSafeValue, _setSafeValue und _getInOutByte. Das Konfigurationsdatum *restartInfo.behaviorInvalidSysvarAccess* kann für den accessMode = CONFIGURED verwendet werden (siehe Systemvariablen (Seite 135) oder Konfigurationsdaten (Seite 139)).

Ersatzwertstrategie sowie Fehler bei Zugriffen auf Systemvariablen, Konfigdaten und I/O-Variablen

Es gibt verschiedene Situationen, in denen das Lesen oder Schreiben einer Systemvariable, eines Konfigurationsdatums oder einer I/O-Variable scheitert.

Tabelle 5- 4 Mögliche Ursachen für das Scheitern von Lesen/Schreiben

Ursache	Systemvariable	Konfigdatum	I/O-Variable
Variable temporär nicht verfügbar	TO im Restart TO deaktiviert	TO im Restart TO deaktiviert	Input gestört
	-	Datum in aktueller Konfiguration nicht sichtbar	Output gestört
unzulässiger Wert (beim Schreiben)	Wert außerhalb der Grenzen	Wert außerhalb der Grenzen	-
	Wert nicht im Raster	Wert nicht im Raster	-

Abhängig von der Ursache sind folgende Reaktionen denkbar:

Tabelle 5- 5 Systemverhalten im Fehlerfall bei Verwendung von _getSafeValue

Gewünschte Reaktion		Systemvariable	Konfigdatum	I/O-Variable
Beim Lesen eines temporär nicht zugreifbaren Wertes:				
in STOP gehen	Aufruf = accessMode	STOP_DEVICE	STOP_DEVICE	STOP_DEVICE
	Antwort	→ STOP	→ STOP	→ STOP
	Wert	unverändert	unverändert	unverändert
nicht lesen	Aufruf = accessMode	NO_CHANGE	NO_CHANGE	nicht realisiert
	Antwort	NO_CHANGE	NO_ACCESS	
	Wert	Letzter Wert	Konfigurierter Wert	
Ersatzwert lesen	Aufruf = accessMode	DEFAULT_VALUE	DEFAULT_VALUE	DEFAULT_VALUE
	Antwort	NO_ACCESS	INVALID_VALUE	DEFAULT_VALUE
	Wert	Defaultwert	Defaultwert	Ersatzwert
letzten gültigen Wert lesen	Aufruf = accessMode	CONFIGURED	CONFIGURED	NO_CHANGE
	Antwort	(entsp. Konfigdatum behavior.....)	(entsp. Konfigdatum behavior.....)	NO_CHANGE
	Wert			letzter gültiger Wert

Tabelle 5- 6 Systemverhalten im Fehlerfall bei Verwendung von _setSafeValue - beim Schreiben eines temporär nicht zugreifbaren Wertes

Gewünschte Reaktion		Systemvariable	Konfigdatum	I/O-Variable
in STOP gehen	Aufruf = accessMode	STOP_DEVICE	STOP_DEVICE	STOP_DEVICE
	Antwort	→ STOP	→ STOP	→ STOP
	Wert	unverändert	unverändert	aktueller Wert
	Rückgabewert setValue	Aktueller Wert	Aktueller Wert	-
nicht schreiben	Aufruf = accessMode	NO_CHANGE	NO_CHANGE	nicht realisiert
	Antwort	NO_CHANGE	NO_ACCESS	
	Wert	unverändert	unverändert	

Gewünschte Reaktion		Systemvariable	Konfigdatum	I/O-Variable
	Rückgabewert setValue	Aktueller Wert	Aktueller Wert	
nicht schreiben	Aufruf = accessMode	DEFAULT_VALUE	DEFAULT_VALUE	nicht realisiert
	Antwort	NO_ACCESS	NO_ACCESS	
	Wert	unverändert	unverändert	
	Rückgabewert setValue	Aktueller Wert	Aktueller Wert	
nicht schreiben	Aufruf = accessMode	CONFIGURED	CONFIGURED	nicht realisiert
	Antwort	(entsp. Konfigdatum behavior.....)	(entsp. Konfigdatum behavior.....)	
	Wert			
	Rückgabewert setValue	Aktueller Wert	Aktueller Wert	
Ersatzwert übernehmen, wird später wirksam	Aufruf = accessMode	nicht realisiert	nicht realisiert	DEFAULT_VALUE
	Antwort			DEFAULT_VALUE
	Wert			Ersatzwert
aktuellen Wert übernehmen, wird später wirksam	Aufruf = accessMode	nicht realisiert	nicht realisiert	NO_CHANGE
	Antwort			NO_CHANGE
	Wert			Aktueller Wert

Tabelle 5- 7 Systemverhalten im Fehlerfall bei Verwendung von _setSafeValue - Fehler beim Schreiben eines ungültigen Wertes

Gewünschte Reaktion		Systemvariable	Konfigdatum	I/O-Variable
in STOP gehen	Aufruf = accessMode	STOP_DEVICE	STOP_DEVICE	tritt nicht ein
	Antwort	→ STOP	→ STOP	
	Wert	unverändert	unverändert	
	Rückgabewert setValue	Aktueller Wert	Aktueller Wert	
nicht schreiben	Aufruf = accessMode	NO_CHANGE	NO_CHANGE	
	Antwort	NO_CHANGE	NO_CHANGE	
	Wert	unverändert	unverändert	
	Rückgabewert setValue	Aktueller Wert	Aktueller Wert	
Defaultwert schreiben	Aufruf = accessMode	DEFAULT_VALUE	DEFAULT_VALUE	
	Antwort	DEFAULT_VALUE	INVALID_VALUE	
	Wert	DEFAULT_VALUE	unverändert	
	Rückgabewert setValue	Aktueller Wert	Aktueller Wert	

Legende zu obigen Tabelllen

- DEFAULT_VALUE

  Der DEFAULT_VALUE ist der projektierte Wert. Das ist der Wert, der durch einen Download übertragen wird (Systemvariablen und Konfigdaten).

- LAST VALUE - Konfigurierter Wert (Konfigdaten)

  Der letzte Wert kann der konfigurierte Wert sein. Der Wert ist dann äquivalent zu dem der in der Expertenliste unter "aktueller Wert" angezeigt wird.

  ODER

  Der letzte Wert kann der letzte konfigurierte Wert sein. Der Wert ist dann äquivalent zu dem der in der Expertenliste und "nächster Wert" angezeigt wird.

  Je nach angegebenem Konfigdatum kann der eine oder der andere Wert ausgegeben werden.

- LAST VALUE - Letzter Wert (Systemvariablen)

  Bei Systemvariablen ist der letzte Wert der aktuell eingestellte und wirksame Wert.

Siehe auch

Systemvariablen (Seite 135)

Funktion _getSafeValue (Seite 406)

Funktion _setSafeValue (Seite 408)

Funktion _getInOutByte (Seite 412)

Konfigurationsdaten (Seite 139)

Ablaufsystem/Tasks/Systemtakte 6

6.1 Das Ablaufsystem

Das SIMOTION **Ablaufsystem** stellt verschiedene **Ablaufebenen** bereit.

- Den Ablaufebenen werden **Anwenderprogramm-Tasks** zugeordnet.
  - Den Anwender-Tasks werden **Programme** zugeordnet.

  Alle Programme - und damit auch Tasks - können PLC- und Motion-Control-Aufgaben enthalten.

Folgende Ablaufebenen sind verfügbar:

- Synchrone Ablaufebenen: synchron zu Regelungs- und Interpolatortakten
- Zeitgesteuerte Ablaufebenen
- Ereignisgesteuerte Ablaufebenen
- Interrupt-gesteuerte Ablaufebenen
- Sequentielle Ablaufebenen
- Freilaufende Ablaufebenen

Aufgabenorientiert stehen verschiedene **Tasks** mit unterschiedlichen Ablaufeigenschaften zur Verfügung:

System-Tasks

System-Tasks werden regelmäßig vom System abgearbeitet.

Der Systemtakt kann vorgegeben werden.

Folgende Aufgaben werden von System-Tasks bearbeitet:

- **Kommunikation**

  System-Tasks zur Anbindung an den taktsynchronen PROFIBUS, an PROFINET IO mit IRT bzw. den Systemtakt sowie zur Bearbeitung der Peripherie.

- **Bewegungsführung (Motion Control)**
  - Basis-Takt / Bus-Takt

    Der Basis-Takt / Bus-Takt (DP-Takt oder PN-Takt) leitet sich vom Zyklus des taktsynchronen Busses ab und bestimmt, in welchen zeitlichen Abständen der Datenaustausch mit der DP-/PN-Peripherie erfolgen soll. Der Basis-Takt/Bus-Takt wird als Grundlage für die Einstellung der weiteren Takte verwendet.
  - Servo-Takt

    In der Zeitscheibe des Servo-Takts erfolgt u. a. die Lageregelung und Überwachungen der Achsen, die Antriebskommunikation sowie die Peripheriebearbeitung. Der Zyklus des Servo-Takts bestimmt den zeitlichen Abstand zwischen zwei Lageregler-Takten. Der Servo-Takt kann als Vielfaches des Basis-/Bus-Taktes eingestellt werden. Empfehlenswert ist ein Taktverhältnis von 1:1.
  - IPO-Takt

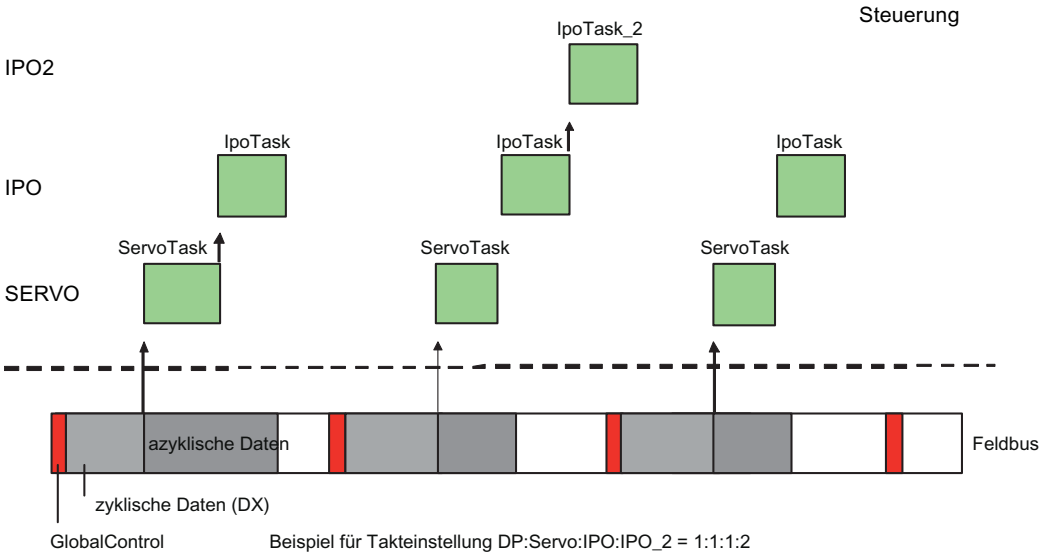
    Die Zeitscheibe des Interpolator-Takts dient u. a. zur Berechnung der Führungsgrößen. Der Zyklus des Interpolator-Takts bestimmt den zeitlichen Abstand zwischen zwei Interpolator-Takten. Beim Start eines oder mehrerer Aufträge kann es kurzzeitig zu einem höheren Zeitbedarf für diese Zeitscheibe kommen. Der IPO-Takt kann als Vielfaches des Servo-Takt eingestellt werden. Empfehlenswert ist ein Taktverhältnis von 1:1.
  - IPO2-Takt 2

    Der Interpolator-Takt 2 erfüllt die gleichen Aufgaben wie der Interpolator-Takt; er kann für niederpriore Technologieobjekte eingesetzt werden. Der IPO2-Takt kann als Vielfaches des IPO-Takt eingestellt werden.
  - Servo_fast/IPO_fast

    Optional kann ab V4.2 ein zusätzlicher schnellerer Servo-Takt und IPO-Takt projektiert werden, siehe auch Servo_fast (Applikationstakt für schnelles Bussystem) (Seite 257).

  Einstellbare Übersetzungsverhältnisse zwischen Bus-Task, Servo und IPO erlauben eine sinnvolle Lastverteilung und optimale Systemauslastung.

  Bei digitalen Antrieben sind diese Ablaufebenen synchronisiert mit dem taktsynchronen PROFIBUS bzw. PROFINET IO mit IRT. Bei analogen Antrieben ohne taktsynchronen PROFIBUS werden die Ablaufebenen von einem internen Timer mit einem einstellbaren Systemtakt versorgt.



- **Temperaturregelung**

  In Verbindung mit dem Technologiepakt **TControl** stehen für die Temperaturregelung Ablaufebenen zur Verfügung: Istwerterfassung, Regelung und Pulsweitenmodulation der Ausgangssignale.

Anwenderprogrammierung

Für die Anwenderprogrammierung stehen Ablaufebenen für die aufgabenbezogene Programmierung zur Verfügung (**Anwenderprogramm-Tasks**): Motion Control, Logik und Technologischen Funktionen.

Siehe auch

Festlegungen beim Konfigurieren (Seite 314)

6.1.1 Ablaufebenen/Tasks

Die Ablaufebenen legen die zeitliche Reihenfolge von Programmen im Ablaufsystem fest. Hierzu enthält jede Ablaufebene eine oder mehrere Tasks.

Eine Task stellt den Ablaufrahmen für die Programme zur Verfügung. Jede Task wird bei definierten Bedingungen abgearbeitet. Jeder Task können Sie ein oder mehrere Anwenderprogramme zuordnen und deren Reihenfolge innerhalb der Task festlegen.

Neben diesen Anwenderprogramm-Tasks sind auch mehrere System-Tasks vorhanden, auf deren Inhalt und Ablaufreihenfolge Sie keinen Einfluss haben.

Ablaufebenen

Folgendes Bild zeigt die Ablaufebenen mit ihren Tasks.

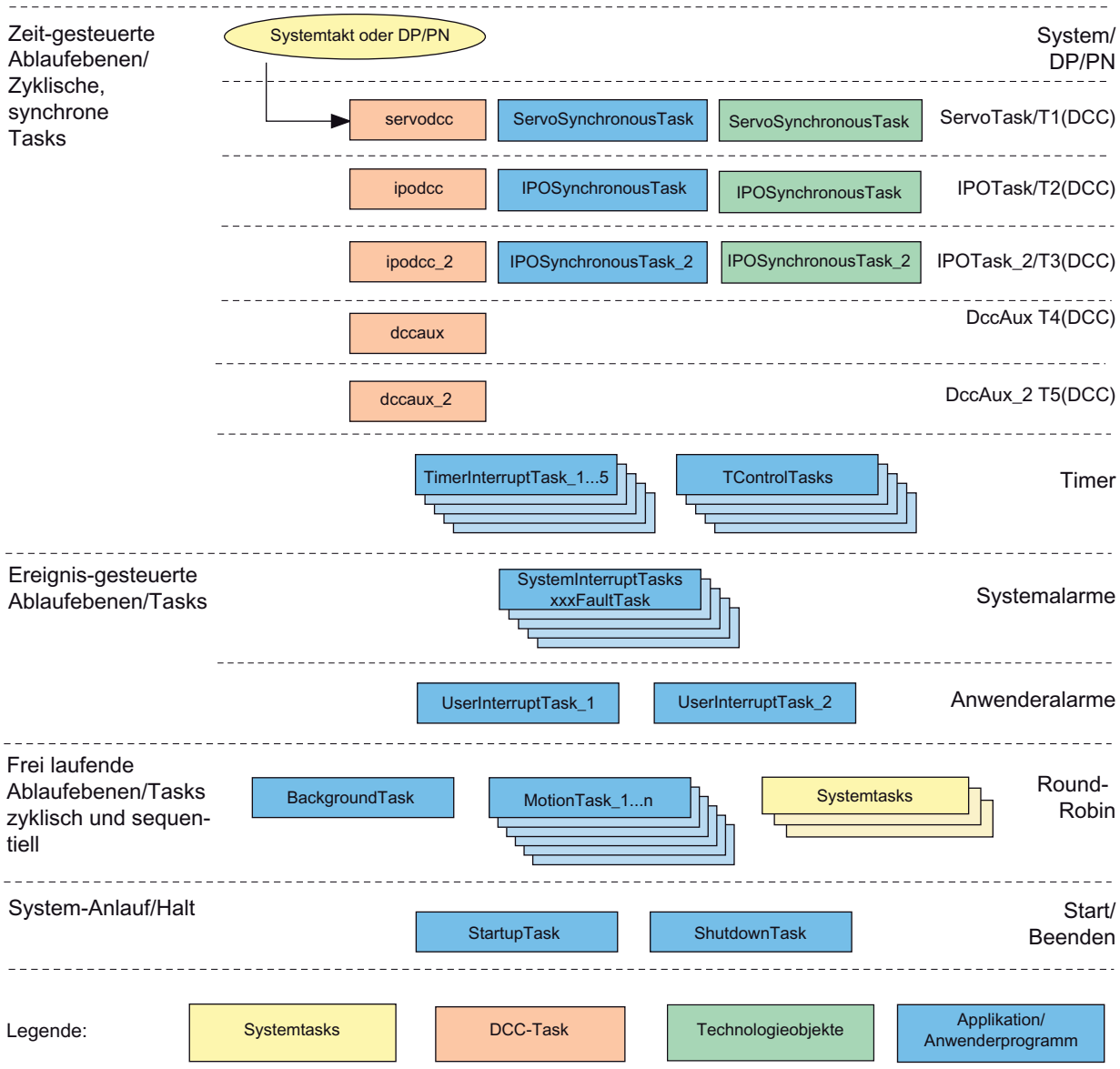


Bild 6-1 Ablaufebenen und Tasks im SIMOTION Runtime-System

Bei der Verwendung der Technologiepakete werden die vom System zur Verfügung gestellten Ablaufebenen automatisch belegt. Die Bearbeitung der technologischen Funktionen in diesen Ablaufebenen kann durch das Anwenderprogramm nicht beeinflusst werden. Systemtasks sind z. B. azyklische Kommunikation (Ethernet, PROFIBUS DP, PROFINET IO), Debugdienste oder Traceaufbereitung.

Die DCC-Ebenen und Tasks sind im Ablaufsystem in der Oberfläche von SCOUT nicht sichtbar. Die Einordnung der DCC-Pläne erfolgt im DCC-Editor über Ablaufgruppen, siehe **Beschreibung des DCC-Editors**.

Das Zusammenwirken der System- und Anwenderprogramm-Tasks wird am Beispiel des Technologieobjektes Achse gezeigt. Am Ende der zyklischen Datenübertragung werden jeweils die ServoSynchronousTask und die ServoTask gestartet, danach die IPOSynchronousTask und die IPOTask.

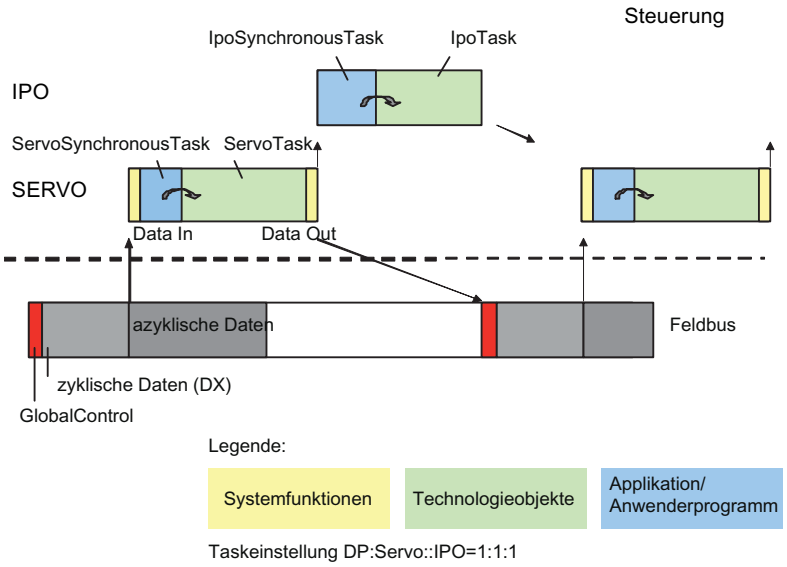


Bild 6-2 Zusammenwirken der Tasks

Weitere Informationen finden Sie auch unter:

Taktsynchrone I/O-Verarbeitung an Feldbussystemen (Seite 276)

Zeitverhalten bzgl. Datenbearbeitung in der Steuerung (Seite 283)

Tasks

Für die Anwenderprogrammierung stehen in den Ablaufebenen jeweils eine oder mehrere Tasks zur Verfügung.

Wesentliche Eigenschaften der Tasks sind:

- Startverhalten: Wann und unter welchen Bedingungen wird die Task gestartet. (siehe Tabelle)
- Priorität: Welche Task wird durch eine andere Task unterbrochen. (siehe Tabelle zu den Taskprioritäten im nächsten Kapitel)

Folgende Tabelle zeigt die Tasks, die für Anwenderprogramme zur Verfügung stehen.

Tabelle 6- 1 Tasks im SIMOTION Runtime-System

Task		Beschreibung
StartupTask		Die **StartupTask** wird beim Übergang des Betriebszustandes **STOP** bzw. **STOPU** zu **RUN** einmalig ausgeführt. Sie ist vorgesehen für Initialisierungen und das Rücksetzen von Technologieobjekten.
Freilaufende Tasks:		In der **Round-Robin-Ablaufebene** werden die **MotionTasks** und **BackgroundTask** im Zeitscheibenverfahren vom System im Hintergrund abgearbeitet.
	MotionTasks	**MotionTasks** sind vorgesehen für die Programmierung von Abläufen, für programmierte Bewegungssteuerung oder sonstige *sequentielle* Bearbeitungen. MotionTasks werden durch Anwenderprogramme gestartet und einmalig abgearbeitet.
	BackgroundTask	Die **BackgroundTask** ist vorgesehen für die Programmierung von *zyklischen* Abläufen ohne festes Zeitraster. Die BackgroundTask wird nach Systemanlauf gestartet und dann zyklisch freilaufend abgearbeitet.
Zeitgesteuerte und synchrone Tasks:		**Zyklische Tasks**. Sie werden zyklisch in einem bestimmten Zeitraster aufgerufen und nach Bearbeitung der zugeordneten Programme automatisch neu gestartet.
	TimerInterruptTasks	**TimerInterruptTasks** sind vorgesehen, um Programme periodisch zu starten.
	SynchronousTasks	**SynchronousTasks** werden synchron zu einem Systemtakt periodisch gestartet.
Ereignisgesteuerte Tasks:		**Sequentielle Tasks**. Sie werden bei Eintritt eines Ereignisses einmalig gestartet, abgearbeitet und dann beendet.
	SystemInterruptTasks	**SystemInterruptTasks** werden beim Eintreffen eines Systemereignisses gestartet und einmalig abgearbeitet.
	UserInterruptTasks	**UserInterruptTasks** werden beim Eintreffen eines anwenderdefinierten Ereignisses gestartet und einmalig abgearbeitet.
ShutdownTask		Die **ShutdownTask** wird beim Übergang des Betriebszustandes **RUN** zu **STOP** bzw. **STOPU** einmalig ausgeführt.

Hinweis

Neben den Anwenderprogramm-Tasks gibt es verschiedene systeminterne Tasks, auf die der Anwender keinen Einfluss hat.

Die **ControlPanelTask** ist z. B. eine solche systeminterne Task. Sie ist in den Task-Laufzeiten der Gerätediagnose sichtbar, nicht aber im Ablaufsystem.

Ab V4.1.2 steht Ihnen auch der TaskTrace zur Verfügung. Der SIMOTION Task Trace erfasst den Ablauf der einzelnen Tasks, kennzeichnet sogenannte User Events, welche Sie per Programmbefehl erzeugen können und stellt diese grafisch dar, siehe Funktionshandbuch Task Trace.

Siehe auch

StartupTask (Seite 204)

MotionTasks (Seite 206)

BackgroundTask (Seite 210)

TimerInterruptTasks (Seite 213)

SynchronousTasks (Seite 216)

SystemInterruptTasks (Seite 224)

UserInterruptTasks (Seite 228)

ShutdownTask (Seite 233)

Taktsynchrone I/O-Verarbeitung an Feldbussystemen (Seite 276)

6.1.2 Ablaufsystem in SIMOTION SCOUT

Alle im Ablaufsystem verwendeten Tasks sind in SIMOTION SCOUT sichtbar.

- Markieren Sie im Projektnavigator das SIMOTION Gerät und wählen Sie im Menü **Zielsystem > Ablaufssystem konfigurieren** oder doppelklicken Sie auf **ABLAUFSYSTEM**.

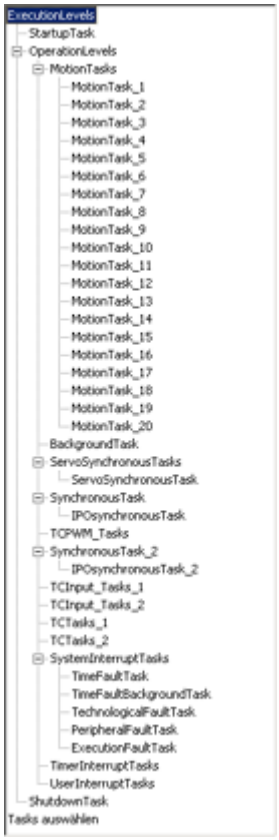


Bild 6-3 Übersicht über die Tasks im SCOUT

Ablaufsystem

Hier wird das Ablaufsystem mit den Ablaufebenen und den zugeordneten Tasks angezeigt.

Ablaufebenenbaum

Der Ablaufebenenbaum zeigt als feste Einträge die zur Verfügung stehenden Ablaufebenen/Tasks.

Im Ordner **OperationLevel** sind die Tasks zusammengefasst, die beim Betriebszustand **RUN** zur Verfügung stehen.

- Durch Klicken auf das Pluszeichen vor dem Ordner **OperationLevel** wird dieser geöffnet.

Unterhalb jeder Ablaufebene bzw. Task-Bezeichnung werden die konfigurierten Tasks mit den zugeordneten Programmen angezeigt.

Nachdem Sie den Tasks Programme zugewiesen haben, werden diese im Ablaufebenenbaum angezeigt.

Task im Ablaufsystem verwenden

In der Liste können Sie die Tasks auswählen, die im Ablaufsystem verwendet werden sollen.

- Aktivieren Sie das Kontrollkästchen der jeweiligen Task.

Im Ablaufebenenbaum werden nur die ausgewählten Tasks angezeigt.

Ablaufsystem - Kontextmenü

Folgende Funktionen können Sie wählen:

Funktion		Bedeutung/Hinweis
Öffnen		Hiermit öffnen Sie die Konfiguration der Ablaufebenen.
Experte		
	Systemtakte einstellen	Hiermit können Sie das Verhältnis der Systemtakte (z. B. Interpolator-Takt zu Servo-Takt) in Abhängigkeit von einer parametrierten Äquidistanz an der PROFIBUS- bzw. PROFINET-Schnittstelle einstellen.
Drucken		Hiermit können Sie den Inhalt des Ablaufsystems drucken. Es werden alle Tasks mit der dazugehörigen Konfiguration gedruckt.
Druckvorschau		Hiermit öffnen Sie die Druckvorschau für das Ablaufsystem.

6.1.3 Task-Prioritäten

Die Tasks mit ihren zugeordneten Programmen werden zu einem bestimmten Zeitpunkt gestartet und abgearbeitet. Sollen mehrere Tasks zum gleichen Zeitpunkt gestartet und abgearbeitet werden, so wird durch die Priorität festgelegt, welche Task Vorrang in der Bearbeitung hat.

Tabelle 6- 2 Überblick über die Taskprioritäten

Priorität	Ablaufebene	Task	zyklisch/ sequentiell	Bedeutung
Hoch	Servo, DP-Kommunikation	-	z	
	Servo / T1 (DCC)	**Servodcc** **ServoSynchronousTask** **ServoTask**	z	Task einer DCC-Ablaufgruppe (T1) Anwenderprogramm-Task im Servo-Takt System-Task
	IPO / T2 (DCC)	**Ipodcc**	z	Task einer DCC-Ablaufgruppe (T2)
		IPOSynchronousTask • Überprüfung UserInterrupt • Anwenderprogramm	z	Anwenderprogramm-Task im Interpolator-Takt
		IPOTask • Überprüfung der Bedingung für WAITFORCONDITION (IPO, System)	z	System-Task im IPO-Takt
	IPO_2 / T3 (DCC)	**Ipodcc_2**	z	Task einer DCC-Ablaufgruppe (T3)
		IPOSynchronousTask_2	z	Anwenderprogramm-Task im Interpolator-Takt 2
		IPOTask2	z	System-Task im IPO-Takt 2
	TControlTasks	**PWMsynchronousTask**	z	Temperaturregler-Task
		PWMTask	z	System -Task
		InputTask_1	z	System -Task
		InputSynchronousTask_ 1	z	Temperaturregler-Task
		InputTask_2	z	System -Task
		InputSynchronousTask_ 2	z	Temperaturregler-Task
		ControlTask_1	z	System -Task
		PostControlTask_1	z	Temperaturregler-Task
		ControlTask_2	z	System -Task
		PostControlTask_2	z	Temperaturregler-Task
	DccAux (DCC)	**Dccaux**	z	Task einer DCC-Ablaufgruppe (T4)
	DccAux_2 (DCC)	**Dccaux_2**	z	Task einer DCC-Ablaufgruppe (T5)
	SystemInterruptTasks	**TimeFaultTask**	s	System-Alarm (in der Reihenfolge der Ereignisse)
		TimeFaultBackgroundTask	s	System-Alarm (in der Reihenfolge der Ereignisse)

Priorität	Ablaufebene	Task	zyklisch/ sequentiell	Bedeutung
Niedrig		**TechnologicalFaultTask**	s	System-Alarm (in der Reihenfolge der Ereignisse)
		PeripheralFaultTask	s	System-Alarm (in der Reihenfolge der Ereignisse)
		ExecutionFaultTask	s	System-Alarm (in der Reihenfolge der Ereignisse)
	TimerInterruptTasks	**TimerInterruptTask1 ... TimerInterruptTask5**	z ... z	Timer-Alarme
	UserInterruptTasks	**UserInterruptTask_1**	s	Anwender-Alarm (in der Reihenfolge der Ereignisse)
		UserInterruptTask_2	s	Anwender-Alarm (in der Reihenfolge der Ereignisse)
	RoundRobin	**BackgroundTask**	z	Anwenderprogramm-Task (Reihenfolge kann nicht festgelegt werden.)
		MotionTask_1 ... MotionTask_32	s ... s	Anwenderprogramm-Tasks (Reihenfolge kann nicht festgelegt werden.)

Prioritäten

Die Priorität einer Task kann *nicht* vom Anwender geändert werden. Für Informationen zu Ablaufebenen von Servo_fast und IPO_fast, siehe auch Prioritäten und Reihenfolgen der synchronen Tasks (Seite 262).

Hinweis

Die Tasks laufen entsprechend ihrer Prioritäten. Daher verdrängen höherpriore Tasks niederpriore.

Bei den niederprioren Tasks kann es daher zu Schwankungen der Zykluszeit kommen.

- **TimerInterruptTasks:**

  Kürzere Zeitscheiben haben höhere Priorität als längere.

- **UserInterruptTasks:**

  Alle Tasks haben gleiche Priorität und werden in der Reihenfolge ihrer Aktivierungsereignisse abgearbeitet.

- **Warte auf Bedingung / WAITFORCONDITION** erhöht temporär die Priorität einer MotionTask:
  - Die Bedingung wird mit der gleichen Priorität geprüft wie bei **UserInterruptTasks**.
  - Wenn die Bedingung erfüllt ist, wird die (bisher in Wartestellung befindliche) MotionTask reaktiviert.
  - Die zwischen **WAITFORCONDITION** und **ENDWAITFORCONDITION** eingeschlossenen Befehle (bei MCC im grauen Bereich hinter dem Befehl) werden mit erhöhter Priorität ausgeführt (zwischen **SystemInterruptTasks** und **TimerInterruptTasks**).

Nähere Information zu **Warte auf Bedingung** / **WAITFORCONDITION** finden Sie in den Programmierhandbüchern **SIMOTION MCC** bzw. **SIMOTION ST**.

Überprüfung der Bedingung für UserInterruptTask

Die Bedingung für die UserInterruptTasks wird in der IPOSynchronousTask überprüft, und zwar vor der Ausführung des Anwenderprogrammes der IPOSynchronousTask.

Auftretende Fehler bei der Bedingung für die UserInterruptTask werden so behandelt, als kämen sie aus der IPOsynchronousTask.

Die Laufzeit für die Prüfung der Bedingung der UserInterruptTask belastet die IPOsynchronousTask.

Überprüfung der Wartebedingung

Wenn Wartemechanismen genutzt werden, wie z. B. **Warte auf Bedingung** / **WAITFORCONDITION**, dann wird die Überprüfung vom System ausgeführt, die Task für die Überprüfung wird nicht gestartet.

Eine synchrone Einstellung am Befehl (z. B. Warten bis Bewegungsende) ist vor allem in sequentiellen Tasks sinnvoll.

Die Bedingung für WAITFORCONDITION wird nach dem Anwenderprogramm der IPOSynchronousTask zu Beginn der IPOTask überprüft.

Auftretende Fehler bei WAITFORCONDITION werden so behandelt, als kämen sie aus der entsprechenden MotionTask.

Die Laufzeit für die Prüfung der Bedingung(en) für WAITFORCONDITION belastet die IPOTask.

Startreihenfolge der Tasks

Nach Ablauf der StartupTask ist der Betriebszustand **RUN** erreicht.

Dann werden gestartet:

- die SynchronousTasks
- die TimerInterruptTasks
- die BackgroundTask
- die MotionTasks, bei denen das Attribut für automatischen Start gesetzt ist

Hinweis

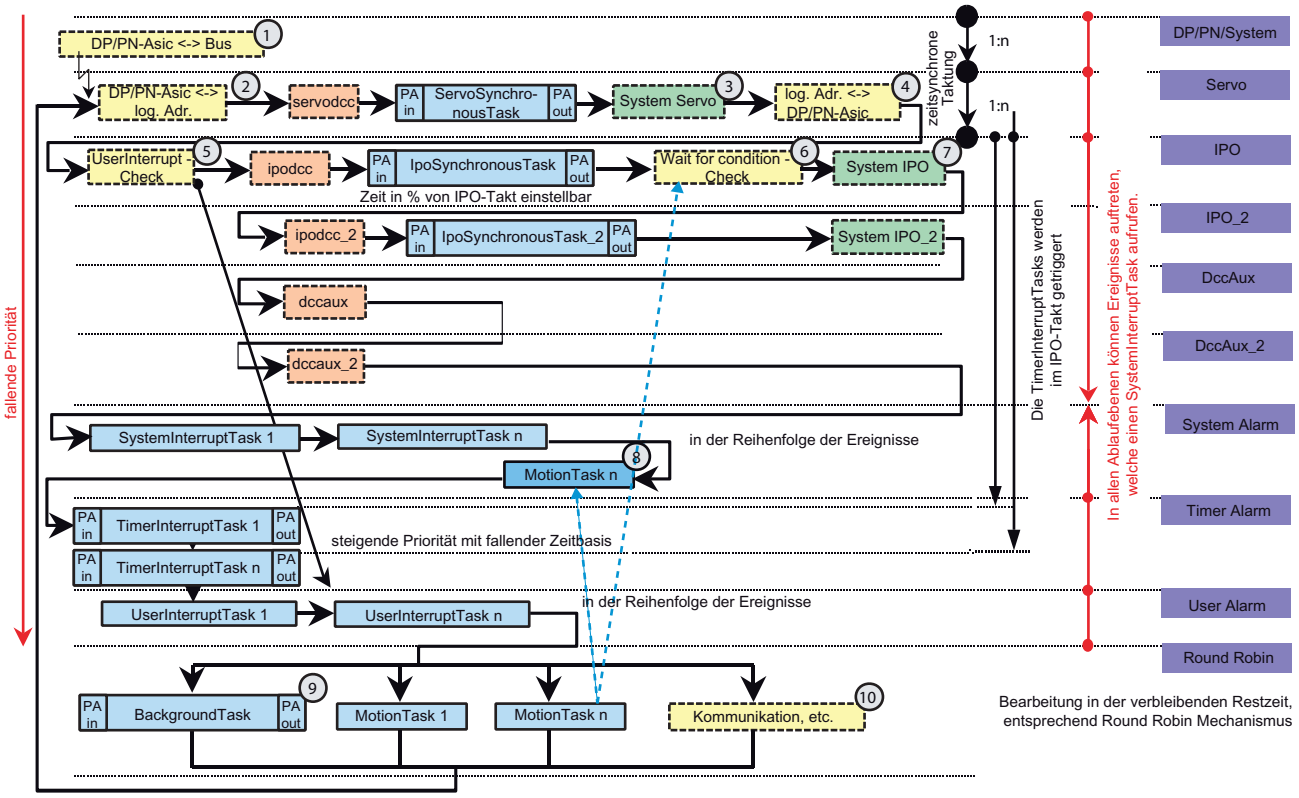
Die Prioritäten der Ablaufebenen bzw. deren Tasks lassen keinen Rückschluss darauf zu, in welcher Reihenfolge die MotionTasks, BackgroundTask und zeitgesteuerten Tasks nach Erreichen des Betriebszustandes **RUN** erstmals gestartet werden.

Prioritäten der Temperaturregler-Tasks

Die PWM-Tasks haben eine eigene Ebene liegen in ihrer Priorität zwischen der Servo- und Ipo-Ebene. Alle anderen Temperaturreglertasks werden von der Priorität her zwischen der DccAux- und den Fault-Tasks einsortiert und befinden sich in der gleichen Ebene.

6.1.4 Laufzeitmodell in SIMOTION

Das folgende Diagramm zeigt den prinzipiellen Ablauf und (von oben nach unten) die Prioritäten der Tasks in SIMOTION.



Erläuterungen zum Bild:

Die Darstellung gilt für ein Verhältnis zwischen DP, Servo und IPO von 1:1:1)

- Farbe blau/grün: für den Anwender zugänglich (Applikation/Technologieobjekte)
- Farbe gelb (gestrichelt): für Anwender unzugänglich (Systemtasks)

Bild 6-4 Laufzeitmodell in SIMOTION - Prinzipieller Ablauf, Prioritäten

Erläuterung der System-Tasks

1. **DP/PN-ASIC <-> Bus:**

   Daten werden vom Kommunikationschip auf den PROFIBUS bzw. PROFINET IO mit IRT kopiert bzw. von dort geholt. Mit dem Ende des Kopiervorgangs wird der Transfer der Peripherieeingänge auf die logischen Adressen (2.) gestartet.

   Der Kopiervorgang läuft auf einem eigenen Prozessor und damit asynchron zum übrigen Ablaufsystem. Der Synchronisationspunkt ist das Ende des Datenaustausches zwischen dem Bus und dem ASIC.

2. **DP/PN-ASIC -> log. Adr.:**

   Die Periphierieingänge werden vom Kommunikationschip geladen.

3. **System Servo:**

   Systemberechnungen im Servotakt (u.a. Lageregler).

   Wenn nicht alle Tasks der Servo-Ebene in einem Zyklus (Bus-Takt) berechnet werden können, kommt es zu einem Ebenenüberlauf und das System geht in den Betriebszustand STOP, die Anlaufsperre wird gesetzt und es erfolgt ein entsprechender Eintrag in den Diagnosepuffer. Erst nach einem Hochlauf (Netz aus/ein) oder Download kann das System wieder in den Betriebszustand RUN gehen.

4. **Log.Adr. -> DP/PN-ASIC:**

   Die Peripherieausgänge werden auf den Kommunikationschip geschrieben.

5. **UserInterrupt - Check:**

   Die Bedingungen der beiden User-Interrupts werden geprüft.

6. **Wait for condition - Check:**

   Die Bedingungen für WAITFORCONDITION (Warte auf Achse, Warte auf Signal etc.) werden geprüft.

7. **System IPO/IPO_2:**

   Die systemseitigen Anteile des IPO-Takts werden gerechnet (Bewegungsführung: Positionierprofile, Gleichlauf etc.).

8. **MotionTask n:**

   Ein MotionTask der mit einer WAITFORCONDITION wartet, wird bei Eintreten der Bedingung bevorzugt (mit höherer Priorität) eingewechselt.

9. **BackgroundTask:**

   Hier läuft die **BackgroundTask**.

   Die Aktualisierung des Background Prozessabbilds (PA) wird zu Beginn und nach Ablauf der kompletten BackgroundTask vorgenommen.

   Zwischen dem Lesen des Eingangsabbilds und dem Schreiben des Ausgangsabbilds liegt im Allgemeinen ein mehrfacher Durchlauf des Laufzeitmodells. D.h. die BackgroundTask wird abhängig von der Größe des Anwenderprogramms mehrfach von höherprioren Tasks (beginnend mit der ServoTask) unterbrochen.

10. **Kommunikation:**

    Kommunikationsfunktionen (HMI, PG/PC etc.).

Siehe auch

SynchronousTasks (Seite 216)

BackgroundTask (Seite 210)

6.1.5 Ablaufsystem im Symbolbrowser

Beschreibung

Sie können Programminstanzdaten von Programmen, die im Ablaufsystem verwendet werden, beobachten und ggf. auch ändern. Es handelt sich dabei um die Werte von lokalen Variablen, die normalerweise an der Unit nicht beobachtet werden können.

Sie haben z. B. einen Funktionsbaustein mehrere Male im Ablaufsystem an MotionTasks verwendet. Im Symbolbrowser des Ablaufsystems können Sie dann unter jeder MotionTask die Programminstanzdaten einsehen.

So zeigen Sie den Symbolbrowser für das Ablaufsystem an:

1. Wählen Sie im Projektnavigator das Ablaufsystem aus.
2. Klicken Sie in der Detailanzeige den Symbolbrowser in den Vordergrund.

   Sie sehen dann für jede Task die verfügbaren Programminstanzdaten eingeblendet.

Hinweis

Wenn Sie als Compilereinstellung "Programminstanzdaten nur einmal anlegen" ausgewählt haben, sehen Sie die Programminstanzdaten nicht mehr. Sie werden dann lokal an der Unit angezeigt.

6.1.6 Synchronität aller Komponenten

Beschreibung

Alle Komponenten (das SIMOTION-System, die Anwenderprogramme, die Antriebe, das Bussystem, und zeitgenaue Peripherie-Baugruppen) arbeiten deterministisch und streng synchron. Die Synchronisierung der einzelnen Komponenten erfolgt dabei über einen isochronen Feldbus (PROFIBUS oder PROFINET).

SIMOTION synchronisiert sich auf den Bustakt und gibt ihn auch an einen zweiten Bus weiter.

Dadurch ist die Deterministik aller Komponenten auch beim Einsatz mehrerer Maschinenmodule und Bussegmente gewährleistet. Auch ein verteilter Gleichlauf ist möglich, z. B. wie im Bild dargestellt:

Die Achsen 1 bis 4 und die Achsen 5 und 6 werden zwar von verschiedenen SIMOTION-Geräten gesteuert, trotzdem können sie im Gleichlauf betrieben werden.

Diese durchgehende Synchronität bewirkt eine hohe Dynamik und Präzision in der gesamten Maschine.

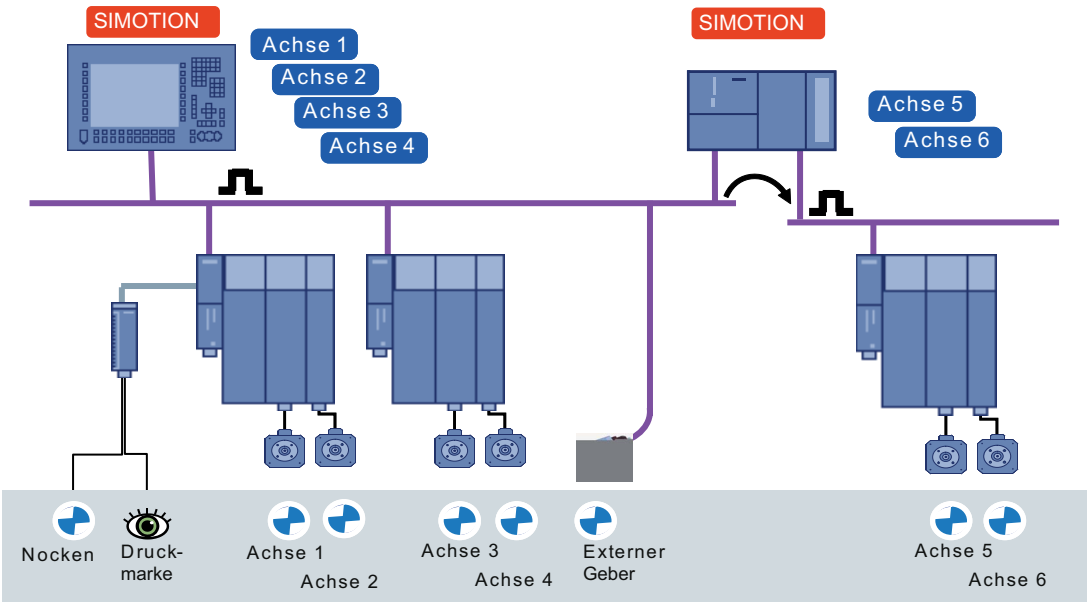


Bild 6-5 Synchronität bei SIMOTION

6.2 Beschreibung der Anwenderprogramm-Tasks

6.2.1 StartupTask

Die **StartupTask** ist vorgesehen für einmalige Initialisierungen und das Rücksetzen der Technologieobjekte.

Sie wird aktiv, wenn der Betriebszustand von **STOP** oder **STOPU** nach **RUN** wechselt.

Sie darf *nicht* verwendet werden für Prozessanlauf oder Referenzieren oder zum Einrichten von Achsen (Motion-Befehle dürfen nicht verwendet werden.)

Während die StartupTask läuft, sind keine anderen Anwenderprogramm-Tasks, außer der **SystemInterruptTask** und der **UserInterruptTask** aktiv.

Zugriff auf das Prozessabbild und symbolische I/O-Variablen ist nur eingeschränkt möglich. Das Prozessabbild der Eingänge wird vor der Startup-Task aktualisiert und bleibt für die Dauer der Startup-Task konstant. Das Prozessabbild der Ausgänge wird vor der Startup-Task genullt und nach der Startup-Task ausgegeben. Direktzugriffe auf Eingänge liefern die aktuellen Werte. Schreibbare I/O-Variablen können auf Initialwerte gesetzt werden. Diese Werte werden allerdings erst nach der Startup-Task mit der Freigabe aller Ausgänge an den Klemmen wirksam.

Nach Beenden der StartupTask ist der Betriebszustand **RUN** erreicht. Zu diesem Zeitpunkt werden folgende Tasks gestartet:

- **SynchronousTasks**
- **TimerInterruptTasks**

- **MotionTasks**, bei denen das Attribut für automatischen Start gesetzt ist
- **BackgroundTask**

Im Register **Programmzuordnung** weisen Sie die erstellten und übersetzten Programme der StartupTask zu und legen die Ablaufreihenfolge fest.

StartupTask konfigurieren

1. Klicken Sie im Ablaufebenenbaum auf **StartupTask**.

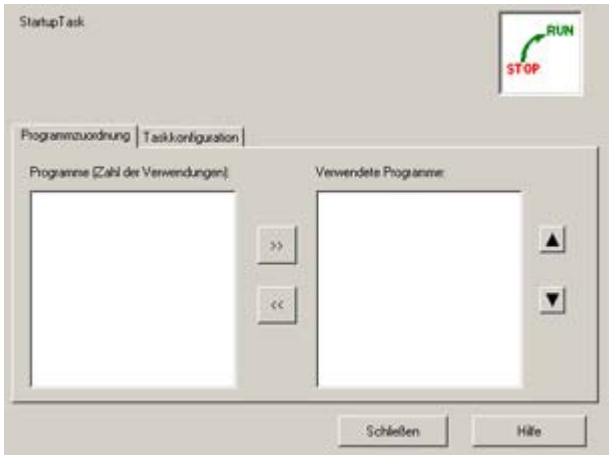


Bild 6-6 Konfiguration der StartupTask (Programmzuordnung)

2. Ordnen Sie im Register **Programmzuordnung** die gewünschten Programme dieser Task zu und legen Sie die Ablaufreihenfolge fest.

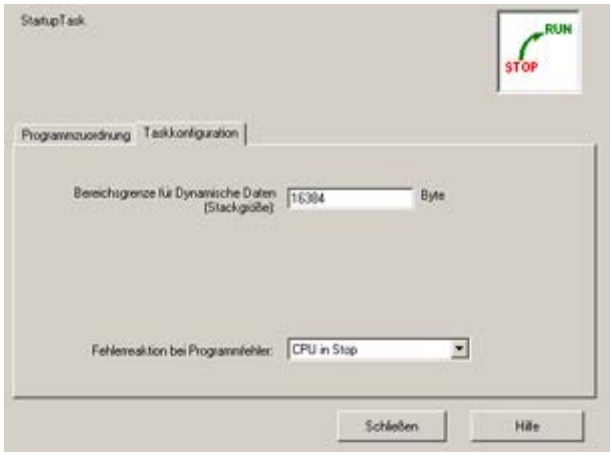


Bild 6-7 Konfiguration der StartupTask (Taskkonfiguration)

3. Wechseln Sie in das Register **Taskkonfiguration**.
4. Geben Sie ggf. die **Bereichsgrenze für dynamische Daten** (**Stackgröße**) an.
5. Legen Sie die **Fehlerreaktion bei Programmfehler** fest (z. B. **ExecutionFaultTask**).

Taskkonfiguration - StartupTask

Im Register **Taskkonfiguration** parametrieren Sie die Fehlerreaktion bei Programmfehlern.

Folgende Parameter können Sie einstellen:

Feld/Schaltfläche		Bedeutung/Hinweis
Bereichsgrenzen für Dynamische Daten		Hier tragen Sie die Größe des Stacks in Byte für diese Task ein. Beim Ablauf der Programme, die dieser Task zugeordnet sind, wird diese Größe im Stack für Daten bereitgestellt. Richtwert ist 16 KB für eine Task.
Fehlerreaktion bei Programmfehler		Hier wählen Sie die Fehlerreaktion, wenn bei Programmen Verarbeitungsfehler auftreten. Programmfehler sind z. B. fehlerhafte Operationen bei Gleitkommazahlen, Division durch Null und Überschreiten von Feldgrenzen.
	CPU in STOP	CPU wechselt in den Zustand **STOP** und die ShutdownTask wird gestartet.
	ExecutionFaultTask	Es wird die ExecutionFaultTask gestartet. Alle Programme, die dieser Task zugewiesen sind, werden gestartet. Sind keine Programme zugeordnet, geht die CPU in **STOP**. Die Task, in der der Fehler auftritt, wird beendet.

Siehe auch

Programme den Ablaufebenen/Tasks zuweisen (Seite 236)

SystemInterruptTasks (Seite 224)

6.2.2 MotionTasks

MotionTasks sind vorgesehen für die Programmierung von Abläufen, für programmierte Bewegungssteuerung oder sonstige sequentielle Bearbeitungen.

Beispiel für die sequentielle Abarbeitung: Eine Achse fährt auf eine Zielposition, wartet auf ein Freigabesignal und fährt dann auf die nächste Zielposition.

Es stehen mehrere MotionTasks zur Verfügung:

- Nur bei SIMOTION D und SIMOTION P: 32 MotionTasks **MotionTask_1** bis **MotionTask_32**

Die Namen der MotionTasks können geändert werden, siehe unten.

MotionTasks werden in der Round-Robin-Ablaufebene ausgeführt.

Hinweis

Die Geräte C2xx verfügen nur über 20 MotionTasks, alle anderen CPUs (siehe oben) verfügen über 32. Bei C2xx können die überzähligen MotionTasks gelöscht werden. Diese gelöschten MotionTasks können nicht wieder angelegt werden.

MotionTasks und **BackgroundTask** teilen sich die freie Zeit neben den höherprioren System- und Anwenderprogramm-Tasks. Das Verhältnis der Zeitscheiben zwischen beiden Ebenen ist parametrierbar, siehe Einstellung der Zeitaufteilung (Seite 269).

Es gibt keine feste Ablaufreihenfolge von MotionTasks und BackgroundTask.

Hinweise zur Beeinflussung des Taskablaufes finden Sie in Übersicht der Tasksteuerbefehle (Seite 331) .

Start einer MotionTask

MotionTasks werden in der Regel vom Anwenderprogramm über Tasksteuerbefehle wie **_startTaskID**, **_stopTaskID**, ... gesteuert. Bei entsprechender Projektierung (gesetztes Attribut) startet eine MotionTask automatisch, nachdem der Betriebszustand **RUN** erreicht ist. Den aktuellen Status der Task können Sie mit dem Systembefehl **_getStateOfTaskID** abfragen.

Eine MotionTask besitzt keine Zeitüberwachung, d. h. eine einmal gestartete MotionTask kann beliebig lange aktiv sein.

Eine MotionTask, die an einem synchronen Befehl wartet, bleibt hinsichtlich ihres Status aktiv.

Beenden einer MotionTask

Eine MotionTask wird beendet nach Ablauf der Task oder bei Übergang in den Betriebszustand **STOP** oder **STOPU** (Start der **ShutdownTask**).

Eine Möglichkeit zur automatischen Suspendierung der Task besteht in der Verwendung von Warte-Befehlen **Warte auf Bedingung** / **WAITFORCONDITION**.

Mit dem Auslösen eines Warte-Befehls wird die Task suspendiert. Die in dem Befehl spezifizierte Bedingung wird im IPO-Takt geprüft. Wenn die Bedingung erfüllt ist, wird die MotionTask automatisch fortgesetzt. Hierbei kann die Priorität der Task beim Fortsetzen am Warte-Befehl beeinflusst werden.

Die zwischen **WAITFORCONDITION** und **ENDWAITFORCONDITION** eingeschlossenen Befehle (bei MCC im grauen Bereich hinter dem Befehl) werden mit erhöhter Priorität ausgeführt (zwischen **SystemInterruptTasks** und **TimerInterruptTasks**).

Im Register **Programmzuordnung** weisen Sie die erstellten und übersetzen Programme den MotionTasks zu und legen die Ablaufreihenfolge fest.

Folgende Parameter können Sie einstellen:

Feld/Schaltfläche	Bedeutung/Hinweis
MotionTask	Unter MotionTask wählen Sie eine der MotionTasks aus, der Sie die Programme zuweisen wollen. Sie können einer MotionTask mehrere Programme zuweisen.
MotionTask_1 bis MotionTask_n	Vordefinierte Namen der möglichen MotionTasks.
Task im Ablaufsystem verwenden	Aktivieren Sie die Kontrollkästchen, wenn die Task im Ablaufsystem angezeigt und verwendet werden soll. Ist die Kontrollkästchen deaktiviert, können Sie dieser Task keine Programme zuweisen.

MotionTasks konfigurieren

Sie können festlegen, welche Tasks automatisch gestartet werden sollen, wenn der Betriebszustand **RUN** erreicht ist. Ansonsten müssen MotionTasks explizit über programmierte Tasksteuerbefehle gestartet werden.

1. Klicken Sie im Ablaufebenenbaum auf **MotionTasks**.

2. Wählen Sie im Auswahlfeld **MotionTask** die gewünschte Task aus. Die Tasknamen können geändert werden.
3. Klicken Sie dazu in das Feld **MotionTask** und geben einen neuen Namen ein, Umlaute, Sonderzeichen sind nicht erlaubt. Der Name wird in der Taskliste aktualisiert.

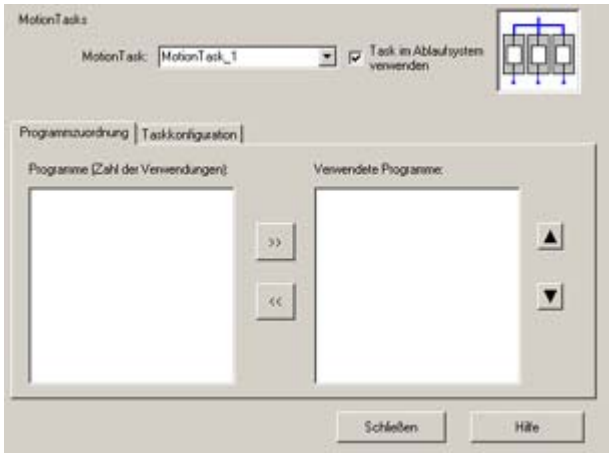


Bild 6-8 Konfiguration von MotionTasks (Programmzuordnung)

4. Ordnen Sie im Register **Programmzuordnung** die gewünschten Programme dieser Task zu und legen Sie die Ablaufreihenfolge fest.

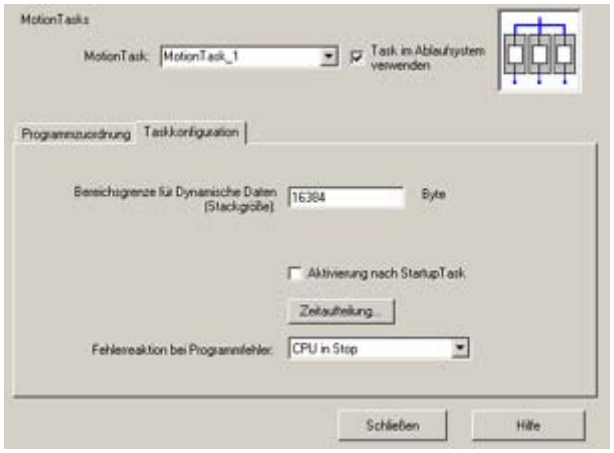


Bild 6-9 Konfiguration von MotionTasks (Taskkonfiguration)

5. Klicken Sie auf das Register **Taskkonfiguration**.
6. Aktivieren Sie die Option **Aktivierung nach StartupTask**, wenn die MotionTask nach der StartupTask einmalig starten soll.
7. Geben Sie ggf. die **Bereichsgrenze für dynamische Daten** (**Stackgröße**) an.
8. Legen Sie die **Fehlerreaktion bei Programmfehler** fest (z. B. **ExecutionFaultTask**).
9. Wiederholen Sie die Schritte 2 bis 7 für alle zu konfigurierenden MotionTasks.
10. Legen Sie die **Zeitaufteilung in der Round-Robin-Ablaufebene** zwischen MotionTasks und BackgroundTask fest, siehe Einstellung der Zeitaufteilung (Seite 269).

Taskkonfiguration - MotionTasks

Im Register **Taskkonfiguration** parametrieren Sie die Zeitaufteilung zwischen MotionTasks und BackgroundTasks in der Round-Robin-Ablaufebene und legen die Aktivierung der MotionTasks fest.

Folgende Parameter können Sie einstellen:

Feld/Schaltfläche	Bedeutung/Hinweis
Bereichsgrenzen für Dynamische Daten	Hier tragen Sie die Größe des Stacks in Byte für diese Task ein. Beim Ablauf der Programme, die dieser Task zugeordnet sind, wird diese Größe im Stack für Daten bereitgestellt. Richtwert ist 16 KB für eine Task.
Aktivierung nach StartupTask	Aktivieren Sie dies, wenn die MotionTask nach dem Erreichen des Betriebszustandes **RUN** (StartupTask ist beendet) einmalig gestartet werden soll.
Zeitaufteilung	Durch Klick auf diese Taste öffnen Sie die Maske für die **Zeitaufteilung in der Round-Robin-Ablaufebene**. Dort parametrieren Sie die Zeitaufteilung zwischen MotionTasks und BackgroundTasks in der Round Robin Ablaufebene.
Fehlerreaktion bei Programmfehler	Hier wählen Sie die Fehlerreaktion, wenn bei Programmen Verarbeitungsfehler auftreten. Programmfehler sind z. B. fehlerhafte Operationen bei Gleitkommazahlen, Division durch Null und Überschreiten von Feldgrenzen.
CPU in STOP	CPU wechselt in den Zustand **STOP** und die ShutdownTask wird gestartet.
ExecutionFaultTask	Es wird die ExecutionFaultTask gestartet. Alle Programme, die dieser Task zugewiesen sind, werden gestartet. Sind keine Programme zugeordnet, geht die CPU in **STOP**. Die Task, in der der Fehler auftritt, wird beendet.

Siehe auch

BackgroundTask (Seite 210)

Programme den Ablaufebenen/Tasks zuweisen (Seite 236)

SystemInterruptTasks (Seite 224)

Zeitaufteilung in der Round-Robin-Ablaufebene (Seite 267)

Programme den Tasks zuordnen (Seite 314)

6.2.3 BackgroundTask

Die **BackgroundTask** ist vorgesehen für die Programmierung von zyklischen Abläufen ohne festes Zeitraster.

Sie wird in der Round-Robin-Ablaufebene zyklisch ausgeführt, d. h. sie wird nach ihrer Beendigung automatisch erneut gestartet.

Die BackgroundTask findet Anwendung bei Programmen, die zyklisch abgearbeitet werden müssen, z. B. Verriegelungsaufgaben, SPS-Aufgaben.

Die Zykluszeit der BackgroundTask wird überwacht. Beim Ansprechen der Zykluszeitüberwachung wird die **TimeFaultBackgroundTask** gestartet. Ist die Task nicht konfiguriert bzw. kein Programm zugeordnet, geht die CPU in **STOP**.

Für die BackgroundTask wird das Prozessabbild der Ein- und Ausgänge in dem Adressraum 0.0 bis 63.7 gebildet. Das Prozessabbild ist für die Bearbeitungszeit der BackgroundTask konsistent.

Die BackgroundTask können Sie u. a. verwenden für die Realisierung von niederprioren, zyklischen Logikfunktionen, Verriegelungen, Berechnungen, Überwachungen.

Die **BackgroundTask** teilt sich mit den **MotionTasks** die freie CPU-Zeit.

Hinweis

Die BackgroundTask teilt sich mit den MotionTasks und Systemtasks (z.B. Kommunikationstasks) die Rechenzeit. Bei der Zeitaufteilung müssen Sie berücksichtigen, dass die Laufzeit bzw. die Performance durch die Einstellungen (Zeitaufteilung der RoundRobin Ablaufebene) beeinflusst werden.

Start der BackgroundTask

Der Start der BackgroundTask erfolgt automatisch, nachdem der Betriebszustand **RUN** erreicht ist oder nach Ablauf der Task (mit dem nächsten Servo-Takt).

Beenden der BackgroundTask

Eine BackgroundTask wird beendet bei Übergang in den Betriebszustand **STOP** oder **STOPU** (Start der **ShutdownTask**).

Im Register **Programmzuordnung** weisen Sie die erstellten und übersetzen Programme der BackgroundTasks zu und legen die Ablaufreihenfolge fest.

BackgroundTask konfigurieren

1. Klicken Sie im Ablaufebenenbaum auf **BackgroundTask**.

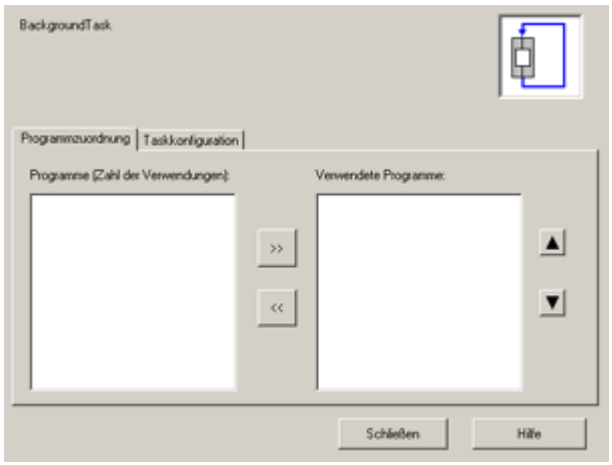


Bild 6-10 Konfiguration der BackgroundTask (Programmzuordnung)

2. Ordnen Sie im Register **Programmzuordnung** die gewünschten Programme dieser Task zu und legen Sie die Ablaufreihenfolge fest.

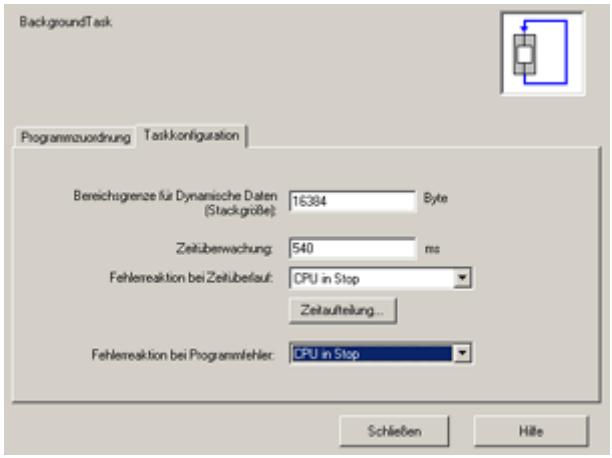


Bild 6-11 Konfiguration der BackgroundTask (Taskzuordnung)

3. Wechseln Sie in das Register **Taskkonfiguration**.
4. Geben Sie ggf. die **Bereichsgrenze für dynamische Daten (Stackgröße)** an.
5. Geben Sie einen Wert an für die **Zeitüberwachung**.

   Bei Überschreiten dieser Zeit kann die zugehörige SystemInterruptTask aufgerufen werden (**TimeFaultBackgroundTask**) oder die **CPU in STOP** gesetzt werden, 0 ms = keine Überwachung).
6. Legen Sie die **Fehlerreaktion bei Zeitüberlauf** fest.
   CPU in STOP oder Aufruf **TimeFaultBackgroundTask**.
7. Legen Sie die **Zeitaufteilung in der Round-Robin-Ablaufebene** zwischen MotionTasks und BackgroundTask fest, siehe **Einstellung der Zeitaufteilung**.
8. Legen Sie die **Fehlerreaktion bei Programmfehler** fest (z. B. **ExecutionFaultTask**).

Taskkonfiguration - BackgroundTask

Im Register **Taskkonfiguration** parametrieren Sie die Zeitüberwachung und die Fehlerreaktion.

Folgende Parameter können Sie einstellen:

Feld/Schaltfläche	Bedeutung/Hinweis
Bereichsgrenzen für Dynamische Daten	Hier tragen Sie die Größe des Stacks in Byte für diese Task ein. Beim Ablauf der Programme, die dieser Task zugeordnet sind, wird diese Größe im Stack für Daten bereitgestellt. Richtwert ist 16 KB für eine Task.
Zeitüberwachung	Hier legen Sie die Zykluszeit in ms fest, die für die Ausführung der BackgroundTask zur Verfügung steht. Tragen Sie einen Wert für die Zeitüberwachung ein. Ist für die Zeitüberwachung kein Wert oder der Wert 0 eingetragen, ist die Zeitüberwachung inaktiv.
Fehlerreaktion bei Programmfehler	Hier wählen Sie die Fehlerreaktion, wenn der Zyklus der BackgroundTask innerhalb des unter Zeitüberwachung eingetragenen Zeitraums noch nicht beendet ist. Als Fehlerreaktion können Sie einen SystemInterrupt wählen und projektieren.
TimeFaultBackgroundTask	Systeminterrupt Zeitüberlauf in der BackgroundTask
CPU in Stop	CPU wechselt in den Zustand **STOP**.
Zeitaufteilung	Durch Klick auf diese Taste öffnen Sie die Maske für die **Zeitaufteilung in der Round-Robin-Ablaufebene**. Dort parametrieren Sie die Zeitaufteilung zwischen MotionTasks und BackgroundTasks in der Round Robin Ablaufebene.
Fehlerreaktion bei Programmfehler	Hier wählen Sie die Fehlerreaktion, wenn bei Programmen Verarbeitungsfehler auftreten. Programmfehler sind z. B. fehlerhafte Operationen bei Gleitkommazahlen, Division durch Null und Überschreiten von Feldgrenzen.
CPU in Stop	CPU wechselt in den Zustand **STOP** und die ShutdownTask wird gestartet.
ExecutionFaultTask	Es wird die ExecutionFaultTask gestartet. Alle Programme, die dieser Task zugewiesen sind, werden gestartet. Die CPU geht anschließend in **STOP**.

Siehe auch

MotionTasks (Seite 206)

Programme den Ablaufebenen/Tasks zuweisen (Seite 236)

Zeitüberwachung (Seite 256)

Einstellung der Zeitaufteilung (Seite 269)

SystemInterruptTasks (Seite 224)

Zeitaufteilung in der Round-Robin-Ablaufebene (Seite 267)

6.2.4 TimerInterruptTasks

TimerInterruptTasks sind vorgesehen, um Programme periodisch zu starten.

Es stehen fünf TimerInterruptTasks, **TimerInterruptTask_1** bis **TimerInterruptTask_5**, für verschiedene Zeitebenen zur Verfügung.

TimerInterruptTasks werden im projektierten festen Zeitraster (z. B. 100 ms) periodisch gestartet und abgearbeitet.

Das Zeitraster muss ein Vielfaches des Interpolator-Taktes IPO sein.

In dieser Task können Regelungsaufgaben oder Überwachungen realisiert werden, die einen reproduzierbaren zeitlichen Bezug ohne die direkte Anbindung an I/O oder die Bewegungsführung der Achsen erfordern.

Für das Technologiepaket TControl werden zusätzliche System- und Anwender-Tasks zur Verfügung gestellt. In den System-Tasks erfolgt die Bearbeitung des Technologiepaketes TControl, in den Anwenderprogramm-Tasks die applikationsspezifischen Anpassungen.

Weitere Informationen finden Sie in der Funktionsbeschreibung des Temperaturreglers, siehe Funktionshandbuch **Motion Control Ergänzende Technologieobjekte**.

Start einer TimerInterruptTask

TimerInterruptTasks werden periodisch gestartet. Dabei ist eine Startverzögerung einstellbar.

Beenden einer TimerInterruptTask

TimerInterruptTasks werden automatisch beendet nach Ablauf der Programme, welche der TimerInterruptTask zugeordnet sind.

Im Register **Programmzuordnung** weisen Sie die erstellten und übersetzen Programme der gewählten TimerInterruptTask zu und legen die Ablaufreihenfolge fest.

Folgende Parameter können Sie einstellen:

Feld/Schaltfläche	Bedeutung/Hinweis
Für Task	Hiermit wählen Sie eine der fünf TimerInterruptTask aus, der Sie die Programme zuweisen wollen. Sie können einer TimerInterruptTask mehrere Programme zuweisen.
TimerInterruptTask_1 bis TimerInterruptTask_5	Vordefinierte Namen der fünf TimerInterruptTask.
festgelegte Zeitebene	Hiermit wählen Sie das Zeitraster in der die TimerInterruptTask wieder gestartet werden soll. Zusätzlich zu der vorhandenen Auswahl können Sie Zeitebenen frei eintragen. Der Wert muss ein Vielfaches des IPO-Takts sein.
Task im Ablaufsystem verwenden	Aktivieren Sie die Kontrollkästchen, wenn die Task im Ablaufsystem angezeigt und verwendet werden soll. Ist die Kontrollkästchen deaktiviert, können Sie dieser Task keine Programme zuweisen.

TimerInterruptTasks konfigurieren

1. Klicken Sie im Ablaufebenenbaum auf **TimerInterruptTasks**.
2. Wählen Sie im Auswahlfeld **Für Task** die gewünschte Task aus. Die Namen sind vorgegeben (**TimerInterruptTask_1** bis **TimerInterruptTask_5**).

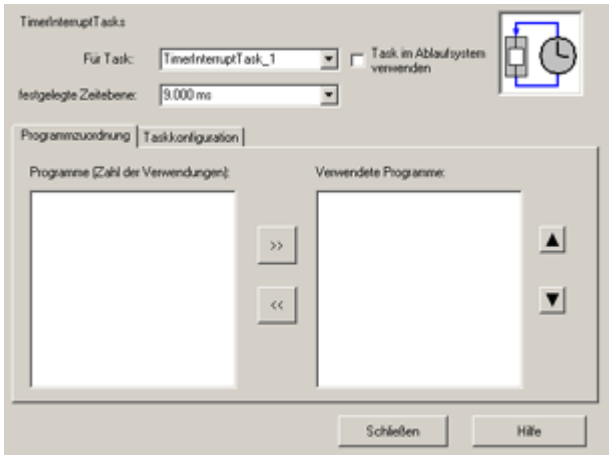


Bild 6-12 Konfiguration der TimerInterruptTasks (Programmzuordnung)

3. Wählen Sie eine **festgelegte Zeitebene** aus oder geben Sie einen beliebigen ganzzahligen Wert ein. Dieser Wert muss ein ganzzahliges Vielfaches der Interpolator-Takte (IPO-Takte) sein. Abweichende Einstellungen werden auf das nächste ganzzahlige Vielfache des IPO-Taktes aufgerundet.
4. Ordnen Sie im Register **Programmzuordnung** die gewünschten Programme dieser Task zu und legen Sie die Ablaufreihenfolge fest.

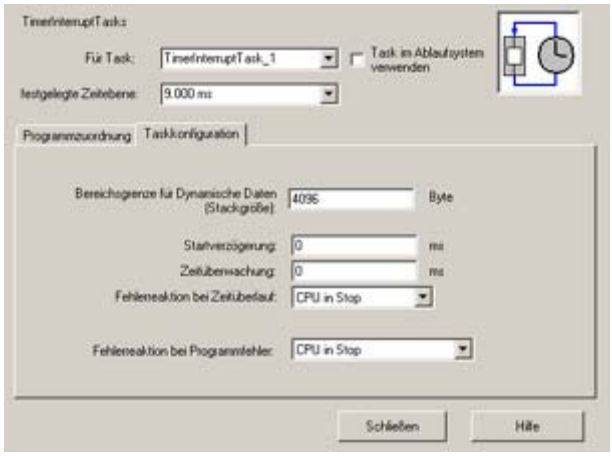


Bild 6-13 Konfiguration der TimerInterruptTasks (Taskkonfiguration)

5. Wechseln Sie in das Register **Taskkonfiguration**.
6. Geben Sie ggf. die **Bereichsgrenze für dynamische Daten (Stackgröße)** an.

7. Geben Sie ggf. die **Startverzögerung** dieser Task ein.

   Um zu vermeiden, dass verschiedene zeitgesteuerte Tasks zum selben Zeitpunkt starten, können Sie für jede Task eine Startverzögerung vorgeben. Damit können Sie die gleichzeitige Bearbeitung verschiedener TimerInterruptTasks vermeiden.

   Beispiel:

   IPO-Takt: 4 ms
   TimerInterruptTask_1: 8 ms
   TimerInterruptTask_2: 16 ms

   In jedem 4. IPO-Takt werden beide TimerInteruptTasks gleichzeitig gestartet. Durch die Startverzögerung einer der beiden Tasks um 4 ms können Sie die gleichzeitige Bearbeitung vermeiden. Hierdurch erreichen Sie eine gleichmäßigere Verteilung der Systemauslastung und damit ein reproduzierbares Verhalten der niederprioren Tasks.

8. Geben Sie einen Wert an für die **Zeitüberwachung**.

   Bei Überschreiten dieser Zeit kann die zugehörige SystemInterruptTask aufgerufen werden (**TimeFaultTask**) oder die **CPU in STOP** gesetzt werden, 0 ms = keine Überwachung).

9. Legen Sie die **Fehlerreaktion bei Zeitüberlauf** fest.
   CPU in STOP oder Aufruf **TimeFaultTask**.

10. Legen Sie die **Fehlerreaktion bei Programmfehler** fest (z. B. **ExecutionFaultTask**).

11. Wiederholen Sie die Schritte 3 bis 10 für alle zu konfigurierenden TimerInterruptTasks.

Taskkonfiguration - TimerInterruptTasks

Im Register **Taskkonfiguration** legen Sie Startverzögerung, Zeitüberwachung und die Fehlerreaktion für die TimerInterruptTasks fest.

Folgende Parameter können Sie einstellen:

Feld/Schaltfläche	Bedeutung/Hinweis
Bereichsgrenzen für Dynamische Daten	Hier tragen Sie die Größe des Stacks in Byte für diese Task ein. Beim Ablauf der Programme, die dieser Task zugeordnet sind, wird diese Größe im Stack für Daten bereitgestellt. Richtwert ist 16 KB für eine Task.
Startverzögerung	Hier tragen Sie einen Zeitwert in ms für die Verzögerung des Starts der Task ein. Der Start der TimerInterruptTask wird um diesen Zeitwert verschoben. Durch die Verschiebung wird sicher gestellt, dass verschiedene TimerInterruptTasks nicht zum gleichen Zeitpunkt starten und dadurch ein Zeitüberlauf entstehen kann.
Zeitüberwachung	Hier legen Sie die Zykluszeit in ms fest, die für die Ausführung der TimerInterruptTask zur Verfügung steht. Tragen Sie einen Wert für die Zeitüberwachung ein. Ist für die Zeitüberwachung kein Wert oder der Wert 0 eingetragen, ist die Zeitüberwachung inaktiv.
Fehlerreaktion bei Zeitüberlauf	Hier wählen Sie die Fehlerreaktion, wenn die TimerInterruptTask innerhalb des unter Zeitüberwachung eingetragenen Zeitraums noch nicht beendet ist. Als Fehlerreaktion können Sie projektieren:
TimeFaultTask	SystemInterrupt Zeitüberlauf in der TimerInterruptTask.

Feld/Schaltfläche		Bedeutung/Hinweis
	CPU in Stop	CPU wechselt in den Zustand **STOP**.
Fehlerreaktion bei Programmfehler		Hier wählen Sie die Fehlerreaktion, wenn bei Programmen Verarbeitungsfehler auftreten. Programmfehler sind z. B. fehlerhafte Operationen bei Gleitkommazahlen, Division durch Null und Überschreiten von Feldgrenzen.
	CPU in STOP	CPU wechselt in den Zustand **STOP** und die ShutdownTask wird gestartet.
	ExecutionFaultTask	Es wird die ExecutionFaultTask gestartet. Alle Programme, die dieser Task zugewiesen sind, werden gestartet. Sind keine Programme zugeordnet, geht die CPU in **STOP**. Die Task, in der der Fehler auftritt, wird beendet.

Siehe auch

Programme den Ablaufebenen/Tasks zuweisen (Seite 236)

Zeitüberwachung (Seite 256)

SystemInterruptTasks (Seite 224)

6.2.5 SynchronousTasks

SynchronousTasks werden synchron zum angegebenen Systemtakt periodisch gestartet. Sie laufen in der zeitgesteuerten Ablaufebene mit hoher Priorität ab.

Es gibt folgende SynchronousTasks:

- **ServoSynchronousTask** (ab V4.0): synchron zum Servo-Takt

  ServoSynchronousTask_fast (ab V4.2): synchron zum Servo_fast-Takt (optional, siehe auch Servo_fast (Applikationstakt für schnelles Bussystem) (Seite 257).

  In der Servo-synchronen Task können Sie zeitkritische Klemme-Klemme-Reaktionen für I/O oder eine schnelle Beeinflussung von Sollwerten auf der Servo-Ebene realisieren.

  Siehe auch Taktsynchrone I/O-Verarbeitung am PROFIBUS DP

  Anwendung z. B. für schnelle Reaktion Klemme - Klemme. Falls ein TO für die Bearbeitung im Servo-Takt konfiguriert ist, können auch TO- und Motion-Befehle abgesetzt werden.

- **IPOSynchronousTask/IPOSynchronousTask_2**: synchron zum Interpolatortakt IPO/IPO_2

  IPOSynchronousTask_fast (ab V4.2): synchron zum Interpolatortakt IPO_fast (optional)

  In den IPO-synchronen Tasks können Sie zeitkritische Funktionen realisieren, die eine direkte Auswirkung auf die Funktionen des Technologieobjekts haben. Das Anwenderprogramm wird vor dem Interpolator bearbeitet, d. h. die programmierten Funktionen können sich im gleichen IPO-Takt auswirken.

  Anwendung z. B. für schnellen Start abhängig von Ereignis, schnelle Reaktion auf Ereignisse Klemme - Achse

Zusätzlich gibt es noch weitere SynchronousTasks für TControl:

- **PWMsynchronousTask**: synchron zum PWM-Takt
- **InputSynchronousTask_1/InputSynchronousTask_2**: synchron zum Takt Input1/2 vor der Temperaturregelung
- **PostControlTask_1/PostControlTask_2**: synchron zum Takt Control1/2 nach der Temperaturregelung

Im Register **Programmzuordnung** weisen Sie die erstellten und übersetzen Programme der gewählten SynchronousTask zu und legen die Ablaufreihenfolge fest.

Folgende Parameter können Sie einstellen:

Feld/Schaltfläche		Bedeutung/Hinweis
Für Taktebene		Hiermit wählen Sie die SynchronousTask aus, der Sie die Programme zuweisen wollen. Sie können einer SynchronousTask mehrere Programme zuweisen.
	ServoSynchronousTask	SynchronousTask, die im Servo-Takt gestartet wird.
	IPOSynchronousTask	SynchronousTask, die im Interpolator-Takt (IPO-Takt) gestartet wird.
	IPOSynchronousTask_2	SynchronousTask, die im zweiten Interpolator-Takt (IPO2-Takt) gestartet wird.
	PWMsynchronousTask (TCPWM_Tasks)	SynchronousTask, die für Stellsignalausgabe des TO Temperaturregler (Temperaturkanal) verwendet wird. Gibt den Grundtakt für die weiteren Tasks des TO Temperaturreglers vor.
	InputSynchronousTask_ 1/2 (TCInput_Tasks_1/2)	SynchronousTask, die für Istwerterfassung des TO Temperaturregler verwendet wird.
	PostControlTask _1/2 (TCTasks_1/2)	SynchronousTask, die für Regelung des TO Temperaturregler verwendet wird.
Task im Ablaufsystem verwenden		Aktivieren Sie die Kontrollkästchen, wenn die Task im Ablaufsystem angezeigt und verwendet werden soll. Ist die Kontrollkästchen deaktiviert, können Sie dieser Task keine Programme zuweisen.

ServoSynchronousTask/ServoSynchronous_fast Task

ServoSynchronousTask sind vorgesehen für Anwendungen, bei denen schnelle Vorgänge mit Hilfe von Taktsynchronität realisiert werden.

ServoSynchronousTasks laufen innerhalb eines Servo-Taktes und verhalten sich ähnlich wie die IPOsynchronousTasks.

ServoSynchronousTasks_fast laufen innerhalb eines Servo_fast-Taktes und verhalten sich ähnlich wie die ServoSynchronousTask.

Die ServoSynchronousTask ist sinnvoll für:

- schnelle Reaktionen (z. B. für kurze Klemme-Klemme Zeiten bei der I/O-Verarbeitung)
- Beeinflussung von Servo-wirksamen Systemvariablen

  Informationen zur Servo-Wirksamkeit der Systemvariablen finden Sie in der SIMOTION Referenzliste Technologiepaket CAM Systemvariablen.
- für Motion-Befehle, wenn ein TO (in Ausnahmefällen) für die Bearbeitung im Servo-Takt konfiguriert ist.

Sie ist nicht sinnvoll für:

- für Kommunikation
- für Motion-Befehle, wenn ein TO für die Bearbeitung im IPO-Takt konfiguriert ist (da diese im IPO-Takt ausgewertet werden)

Falls ein TO für die Bearbeitung im Servo-Takt konfiguriert ist, können auch TO- und Motion-Befehle abgesetzt werden.

Es gelten die gleichen Merkmale wie bei der IPOSynchronousTask.

Die ServoSynchronousTask ist im Ablaufsystem konfigurierbar. (Die Zykluszeit der Servo-Ablaufebene wird beim Servo mit eingestellt). Sie ist standardmäßig aktiv.

Für die ServoSynchronousTask ist wie bei den anderen zyklischen Tasks ein Prozessabbild verfügbar. Ein Performance-Gewinn ist damit nur verbunden, wenn mehrere Male auf die gleichen I/O-Adressen zugegriffen wird.

Ab V4.1 kann auch die Überwachung der ServoSynchronous Task eingestellt werden. Dadurch sind auch MOTION-Befehle in der ServoSynchronous Task erlaubt, deren IPO-Anteile im Servo ausgeführt werden (alle Befehle mit der Weiterschaltbedingung IMMEDIATELY).

Folgende Parameter können Sie einstellen:

Tabelle 6- 3 Taskkonfiguration ServoSynchronousTask

Feld/Schaltfläche		Bedeutung/Hinweis
Bereichsgrenzen für Dynamische Daten		Hier tragen Sie die Größe des Stacks in Byte für diese Task ein. Beim Ablauf der Programme, die dieser Task zugeordnet sind, wird diese Größe im Stack für Daten bereitgestellt. Richtwert ist 16 KB für eine Task.
Überwachung bei der Ausführung von synchronen Funktionen		Hier wählen Sie die Fehlerreaktion, wenn die ServoSynchronousTask innerhalb des Systemtakts noch nicht beendet ist.
	Zeitüberwachung aussetzen und Task unterbrechen	Kompatibel zu den Funktionalitäten bis V4.0 Die Zeitüberwachung wird bei synchronen Funktionen ausgesetzt, ein Servo-Zyklus geht verloren - alle alten Projekte haben diese Einstellung auch, wenn sie auf V4.1 hochgerüstet wurden (CPU Tausch).

Feld/Schaltfläche		Bedeutung/Hinweis
	Zeitüberwachung aktiv lassen	Die Zeitüberwachung wird nicht ausgesetzt, d. h. die CPU wird beim Aufruf synchroner Funktionen, welche die ServoSynchronousTask wirklich unterbrechen, mit Zeitüberlauf in **STOP** gehen. Wenn Überläufe toleriert werden, kann evtl. **STOP** vermieden werden - dies ist die Defaulteinstellung für neu angelegte CPUs. Diagnosepuffereintrag: "Stop durch Ablaufsystem, Ursache: Zeitüberlauf".
	CPU geht in STOP	Es wird keine synchrone Funktion erlaubt. Die CPU geht bei synchronen Funktionen in **STOP** - auch wenn die ServoSynchronousTask im konkreten Fall nicht unterbrochen wird. Diagnosepuffereintrag: "Unerlaubter Aufruf einer System-/Paketfunktion".
Fehlerreaktion bei Programmfehler		Hier wählen Sie die Fehlerreaktion, wenn bei Programmen Verarbeitungsfehler auftreten. Programmfehler sind z. B. fehlerhafte Operationen bei Gleitkommazahlen, Division durch Null und Überschreiten von Feldgrenzen.
	CPU in STOP	CPU wechselt in den Zustand **STOP** und die ShutdownTask wird gestartet.
	ExecutionFaultTask	Es wird die ExecutionFaultTask gestartet. Alle Programme, die dieser Task zugewiesen sind, werden gestartet. Sind keine Programme zugeordnet, geht die CPU in **STOP**. Die Task, in der der Fehler auftritt, wird beendet.

IPOSynchronousTask/IPOSynchronousTask_2/IposynronousTask_fast

IPOSynchronousTasks sind vorgesehen für Anwendungen, bei denen z. B. schnelle und deterministische Reaktionen oder Korrekturbewegungen erforderlich sind. Zeitoptimale Übergaben von Bewegungskommandos an die Bewegungsführung sind damit möglich.

IPOsynchronousTasks laufen innerhalb eines IPO-Taktes unmittelbar vor dem Interpolator. Befehle in diesen Tasks können somit die Bewegungsführung direkt beeinflussen.

Die **IPOSynchronousTask** wird synchron zum IPO-Takt gestartet, die **IPOSynchronousTask_2** synchron zum IPO-Takt_2, einem reduzierten IPO-Takt.

Die **IPOSynchronousTask** wird vor der internen IPOTask, die **IPOSynchronousTask_2** vor der IPOTask_2 abgearbeitet.

Die **IPOSynchronousTask_fast** wird synchron zum IPO_fast gestartet und je nach Variante vor dem Servo-Takt gestartet.

Folgende Eigenschaften sind bei der **IPOSynchronousTasks** vorzugeben:

- Taskkonfiguration
- Summe (Anzahl) der Ebenenüberläufe im IPO/IPO_2-Takt

  Falls die Tasks in der IPO-Ebene (**IPOSynchronousTask**, **IPOTask**) nicht in einem IPO-Takt beendet sind, findet ein Überlauf statt. Ein Überlauf einer Task im Takt (n) muss im nachfolgenden Takt (n+1) abgearbeitet werden.

  Gleiches gilt für die Tasks in der IPO_2-Ebene (**IPOSynchronousTask_2**, **IPOTask_2**).

  Die Anzahl der nacheinander tolerierbaren Überläufe (n = 0 ... 5) können Sie vorgeben. (Das Verhältnis darf n-mal verletzt werden.) Der interne Überwachungszähler wird zurückgesetzt, wenn eine Task ohne Überlauf ausgeführt wurde.

  In den Systemvariablen **numberOfSummarizedTaskOverflow** werden die aufsummierten Ebenenüberläufe angezeigt.

  Für die **ServosynchronousTask_fast** und **IposynchronousTask_fast** sind keine Ebenenüberläufe erlaubt.
- Fehlerreaktion bei Zeitüberlauf (Ebenenüberlauf): Die CPU geht in den Zustand **STOP** und es wird eine Anlaufsperre gesetzt.
- **IPOSynchronousTask/IPOTask**: zeitliches Verhältnis zwischen IPOSynchronousTask und IPOTask.

  Wird dieses Verhältnis von einer IPO-synchronen Task überschritten, kommt es zu einem Zeitüberlauf.

  Sie legen in den Systemtakten auch das Zeitraster für die IPOTask fest. In dem Parameter IPOSynchronousTask/IPOTask kann angegeben werden, wie viel Prozent dieser Zeit für die IPOSynchronousTask pro Takt zur Verfügung gestellt wird.

  Wird z.B. als IPO-Takt 4 ms eingestellt und als Verhältnis 25 % eingestellt, darf eine Laufzeit der IPOSynchronousTask von 1 ms nicht überschritten werden, inklusive möglicher Unterbrechungen höherpriorer Tasks.

  Empfehlung: Geben Sie bei einem Übersetzungsverhältnis von Servo : IPO > 1 einen möglichst großen %-Wert ein.

Hinweis

Wenn in der IPOSynchronous-Task eine synchrone Systemfunktion aufgerufen wird, die damit die IPOSynchronous-Task suspendiert, so geht die CPU mit Diagnosepuffereintrag in **STOP**.

Es ist konfigurierbar, ob eine Zeitüberwachung stattfindet.

SynchronousTasks konfigurieren

1. Klicken Sie im Ablaufebenenbaum auf die entsprechende Task.
2. Wählen Sie im Auswahlfeld **Für Taktebene** die gewünschte SynchronousTask aus.

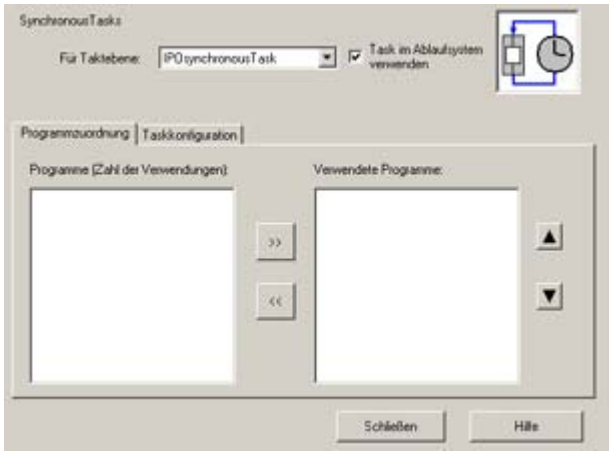


Bild 6-14 Konfiguration der SynchronousTasks (Programmzuordnung)

3. Ordnen Sie im Register **Programmzuordnung** die gewünschten Programme dieser Task zu und legen Sie die Ablaufreihenfolge fest.

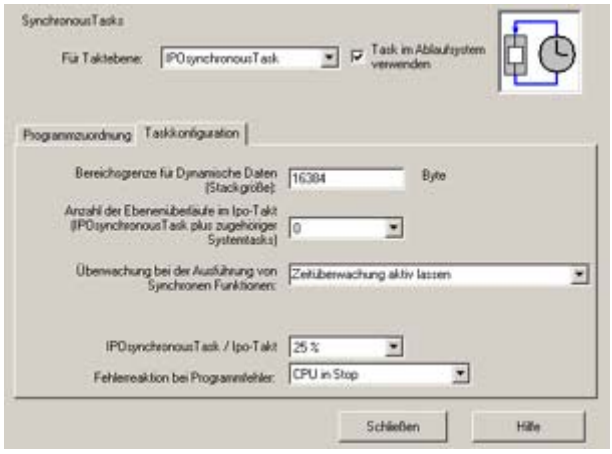


Bild 6-15 Beispiel: Konfiguration der IPOsynchronousTask (Taskkonfiguration)

4. Wechseln Sie in das Register **Taskkonfiguration**.

5. Optional können Sie:
   - die **Stackgröße** angeben
   - im Auswahlfeld **Ebenenüberläufe** die Anzahl (0-5) der erlaubten Ebenenüberläufe festlegen

     Für die ServosynchronousTask_fast und IposynchronousTask_fast sind keine Ebenenüberläufe erlaubt.
   - das Verhältnis zwischen SynchronousTask und IPO-Task auswählen
   - die **Fehlerreaktion bei Programmfehler** auswählen
   - die Zeitüberwachung IPOSynchronousTask/IPOTask konfigurieren
6. Wiederholen Sie die Schritte 3 bis 5 für alle zu konfigurierenden SynchronousTasks.

Taskkonfiguration - SynchronousTasks

Im Register **Taskkonfiguration** wählen Sie die Fehlerreaktion für die SynchronousTask.

Folgende Parameter können Sie einstellen:

Tabelle 6- 4 Taskkonfiguration IPOSynchronousTask

Feld/Schaltfläche		Bedeutung/Hinweis
Bereichsgrenzen für Dynamische Daten		Hier tragen Sie die Größe des Stacks in Byte für diese Task ein. Beim Ablauf der Programme, die dieser Task zugeordnet sind, wird diese Größe im Stack für Daten bereitgestellt. Richtwert ist 16 KB für eine Task.
Anzahl Überläufe IPO/IPO2-Takt		Hier tragen Sie die Anzahl der tolerierbaren Ebenenüberläufe im IPO/IPO_2-Takt für die SynchronousTask ein. Ist die Anzahl der Überläufe kleiner als die eingetragenen Überläufe, erfolgt keine Fehlerreaktion. Bei der Abarbeitung von Rechenprozessen kann es zu Ebenenüberläufen im IPO/IPO_2-Takt kommen, welche vom Anwender für die Ablaufebenen SynchronousTask und SynchronousTask_2 toleriert werden können. Zum Überlaufen in den Ablaufebenen kann es kommen, wenn: • IPOSynchronousTask + IPO-Task > IPO-Takt bzw. • IPOSynchronousTask_2 + IPO_2-Task > IPO_2-Takt Durch diese eingestellte Toleranz wird der nächste IPO- Takt verwendet, um den Rechenprozess des vorhergehenden Taktes zu beenden, ohne das die eingestellte Fehlerreaktion eintritt. Bei Überschreitung bzw. wenn keine Toleranz eingestellt ist, erfolgt ein Eintrag in den Diagnosepuffer und die Fehlerreaktion (CPU in STOP mit Anlaufsperre). Es sind bis zu maximal 5 Ebenenüberläufe einstellbar.
Überwachung bei der Ausführung von synchronen Funktionen		Hier wählen Sie die Fehlerreaktion, wenn die SynchronousTask innerhalb des Systemtakts noch nicht beendet ist.
	Zeitüberwachung aussetzen und Task unterbrechen	Die Zeitüberwachung wird bei synchronen Funktionen ausgesetzt, ein IPO-Zyklus geht verloren.

Feld/Schaltfläche		Bedeutung/Hinweis
	Zeitüberwachung aktiv lassen	Die Zeitüberwachung wird nicht ausgesetzt, d. h. die CPU wird beim Aufruf synchroner Funktionen, welche die IPOSynchronousTask wirklich unterbrechen, mit Zeitüberlauf in **STOP** gehen. Wenn IPO-Überläufe toleriert werden, kann evtl. **STOP** vermieden werden - dies ist die Defaulteinstellung für neu angelegte CPUs. Diagnosepuffereintrag: "Stop durch Ablaufsystem, Ursache: Zeitüberlauf".
	CPU geht in STOP	Es wird keine synchrone Funktion erlaubt. Die CPU geht bei synchronen Funktionen in **STOP** - auch wenn die IPOSynchronousTask im konkreten Fall nicht unterbrochen wird. Diagnosepuffereintrag: "Unerlaubter Aufruf einer System-/Paketfunktion". Das Verhalten gilt für die IPOsynchronousTask, IPOsynchronousTask_2 und die PWMTask.
Task/Takt		Hier wählen Sie für die angegebenen Tasks die Zeitdauer in Prozent vom angegebenen Takt (z. B. IPOsynchrnonousTask/IPO-Takt). Benötigt die Task mehr Zeit als hier eingestellt, reagiert das System mit dem unter Fehlerreaktion gewählten Verhalten. Um die Systemtakte zu konfigurieren, markieren Sie im Projektnavigator die CPU und wählen im Menü Zielsystem > Experte > Systemtakte einstellen. **25%**: Maximale Zeitdauer der Task ist 25% des Takts. **50%**: Maximale Zeitdauer der Task ist 50% des Takts. **75%**: Maximale Zeitdauer der Task ist 75% des Takts.
Fehlerreaktion bei Programmfehler		Hier wählen Sie die Fehlerreaktion, wenn bei Programmen Verarbeitungsfehler auftreten. Programmfehler sind z. B. fehlerhafte Operationen bei Gleitkommazahlen, Division durch Null und Überschreiten von Feldgrenzen.
	CPU in STOP	CPU wechselt in den Zustand **STOP** und die ShutdownTask wird gestartet.
	ExecutionFaultTask	Es wird die ExecutionFaultTask gestartet. Alle Programme, die dieser Task zugewiesen sind, werden gestartet. Sind keine Programme zugeordnet, geht die CPU in **STOP**. Die Task, in der der Fehler auftritt, wird beendet.

Siehe auch

Taktsynchrone I/O-Verarbeitung an Feldbussystemen (Seite 276)

Zeit- und Ebenenüberläufe (Seite 253)

Programme den Ablaufebenen/Tasks zuweisen (Seite 236)

Ablaufmodell für DCC-Bausteine (DCB) (Seite 290)

6.2.6 SystemInterruptTasks

SystemInterruptTasks werden bei Eintritt eines Systemereignisses gestartet und einmalig bearbeitet.

Es gibt folgende SystemInterruptTasks:

- **TimeFaultTask**: startet bei Zeitüberlauf einer TimerInterruptTask
- **TimeFaultBackgroundTask**: startet bei Zeitüberlauf der BackgroundTask
- **TechnologicalFaultTask**: startet bei Fehler an einem Technologieobjekt
- **PeripheralFaultTask**: startet bei Fehler an der Peripherie
- **ExecutionFaultTask**: startet bei Fehler beim Verarbeiten eines Programms

Start einer SystemInterruptTask

Der Start einer SystemInterruptTask erfolgt automatisch, nach Auftreten des eingestellten Ereignisses.

Tritt ein Ereignis auf, das eine SystemInterruptTask startet, muss die SystemInterruptTask im Ablaufsystem verwendet und dieser Task ein Programm zugeordnet werden, sonst geht die CPU in **STOP**.

Es können bis zu acht unterschiedliche Ereignisse im Puffer gespeichert werden. Kommt ein weiteres Ereignis hinzu, so läuft der Puffer über und die CPU geht ebenfalls in **STOP**.

Ereignisse, die zum Aufruf einer SystemInterruptTask führten, können Sie über die jeweilige **TaskStartInfo** dieser Task abfragen und finden Sie unter Taskstartinfo verwenden (Seite 153)).

Beenden einer SystemInterruptTask

Eine SysteminterruptTask wird automatisch nach Ablauf der Programme beendet, welche der SystemInterruptTasks zugeordnet sind.

Im Register **Programmzuordnung** weisen Sie die erstellten und übersetzen Programme der gewählten SystemInterruptTask zu und legen die Ablaufreihenfolge fest.

Folgende Parameter können Sie einstellen:

Feld/Schaltfläche		Bedeutung/Hinweis
Für Task		Hiermit wählen Sie eine der SystemInterruptTasks aus, der Sie die Programme zuweisen wollen. Sie können einer SystemInterruptTask mehrere Programme zuweisen.
	ExecutionFaultTask	Fehler in der Programmverarbeitung z. B. Division durch Null.
	PeripheralFaultTask	Peripheriealarme wie z. B. Prozessalarme, Diagnosealarme, Ziehen und Stecken von Baugruppen.
	TechnologicalFaultTask	Technologische Alarme wie z. B. Alarme, Warnungen und Hinweise.
	TimeFaultBackgroundTask	Zeitüberlauf in der BackgroundTask
	TimeFaultTask	Zeitüberläufe z. B. in den SynchronousTasks, TimerInterruptTasks, Systemumlaufzeit und Zykluszeit.

Feld/Schaltfläche	Bedeutung/Hinweis
Alarmkonfiguration	Hiermit wird das Fenster zur Konfiguration der Technologischen Alarme aufgeblendet. Dort konfigurieren Sie die Alarmreaktion der SystemInterruptTasks bei Auftreten eines Technologischen Alarms.
Task im Ablaufsystem verwenden	Aktivieren Sie die Kontrollkästchen, wenn die Task im Ablaufsystem angezeigt und verwendet werden soll. Ist die Kontrollkästchen deaktiviert, können Sie dieser Task keine Programme zuweisen.

TimeFaultTask

Die **TimeFaultTask** wird gestartet, wenn die Zeitüberwachung einer **TimerInterruptTask** anspricht. Ist die TimeFaultTask nicht konfiguriert bzw. kein Programm zugeordnet, geht die CPU in **STOP**.

In der **TimeFaultTask** können Sie die Reaktion auf Zeitüberläufe der TimerInterruptTasks programmieren.

Weitere Informationen siehe).

TimeFaultBackgroundTask

Die **TimeFaultBackgroundTask** wird gestartet, wenn die Zeitüberwachung der BackgroundTask anspricht. Ist die **TimeFaultBackgroundTask** nicht konfiguriert bzw. kein Programm zugeordnet, geht die CPU in **STOP**.

In der **TimeFaultBackgroundTask** können Sie die Behandlung des Zeitüberlaufes der **BackgroundTasks** programmieren.

TechnologicalFaultTask

Die **TechnologicalFaultTask** wird gestartet, wenn das Technologiepaket eine Alarm- oder Hinweismeldung generiert. Ist die TechnologicalFaultTask nicht konfiguriert bzw. kein Programm zugeordnet, geht die CPU in **STOP**.

Alarmmeldungen haben in der Regel eine direkte Auswirkung auf das Verhalten des Technologiepaketes und müssen quittiert werden, bevor eine erneute Aktivierung von technologischen Funktionen möglich ist.

In der **TechnologicalFaultTask** können Sie z. B. Fehler direkt quittieren und/oder weitere Reaktionen, bezogen auf den Maschinenablauf, einleiten.

Weitere Informationen siehe).

PeripheralFaultTask

Die **PeripheralFaultTask** wird entsprechend ihrer Priorität bei Peripheriezugriffsfehlern sofort gestartet.

Peripheriezugriffsfehler können z. B. auftreten, wenn die Lastspannungsversorgung der Peripheriebaugruppe ausgefallen ist oder andere Fehler an der Peripheriebaugruppe auftreten.

Hinweise hierzu entnehmen Sie den Beschreibungen der *Peripheriebaugruppen*.

Die Task, bei deren I/O-Zugriff ein Fehler aufgetreten ist, *wird nicht beendet*.

Ist die **PeripheralFaultTask** nicht konfiguriert bzw. kein Programm zugeordnet, geht die CPU in **STOP**.

Weitere Informationen siehe)."

ExecutionFaultTask

Die **ExecutionFaultTask** wird entsprechend ihrer Priorität bei Programmlauffehlern sofort gestartet.

Zu Programmlauffehlern zählen z. B.:

- Fehlerhafte Operationen mit Gleitkommazahlen, wie Logarithmus von negativen Zahlen, ungültige Zahl, ...
- Division durch Null
- Überschreiten von Feldgrenzen
- Fehler beim Zugriff auf Systemvariablen

Die Task, bei der der Programmlauffehler aufgetreten ist, wird beendet.

Ist die **ExecutionFaultTask** nicht konfiguriert bzw. ist kein Programm zugeordnet, geht die CPU in **STOP**.

Die Fehlerreaktion CPU in **STOP** ist bei allen Tasks möglich und führt zum Start der **ShutdownTask**. Das SIMOTION Gerät wechselt in den Betriebszustand **STOP**.

Eine Fehlerreaktion der ExecutionFaultTask führt zu einem Neustart der ExecutionFaultTask.

Folgende Tasks können durch Befehle im Programm der ExecutionFaultTask neu gestartet werden:

- StartupTask
- ShutdownTask
- MotionTasks

Bei folgenden Tasks wechselt das SIMOTION Gerät nach Beendigung der ExecutionFaultTask in den Betriebszustand **STOP** – die ShutdownTask wird gestartet:

- BackgroundTask
- TimerInterruptTasks
- SynchronousTasks

Weitere Informationen siehe).

Hinweis

Programmfehler in der **ExecutionFaultTask** und in der **ShutdownTask** führen unmittelbar in den Betriebszustand **STOP**.

SystemInterruptTasks konfigurieren

1. Klicken Sie im Ablaufebenenbaum auf **SystemInterruptTasks**.
2. Wählen Sie im Auswahlfeld **Für Task** eine der vorgegebenen Tasks aus.

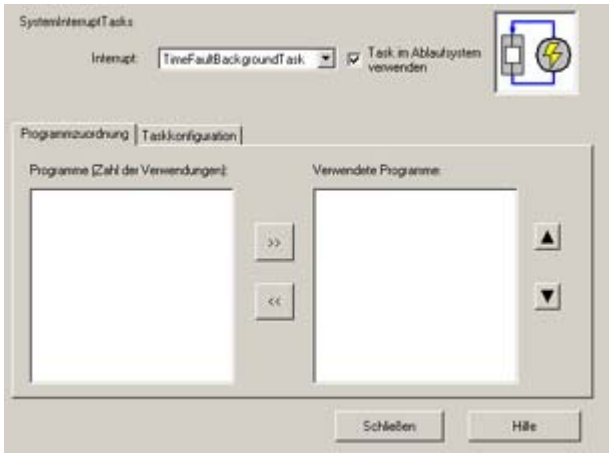


Bild 6-16 Konfiguration der SystemInterruptTasks (Programmzuordnung)

3. Ordnen Sie im Register **Programmzuordnung** die gewünschten Programme dieser Task zu und legen Sie die Ablaufreihenfolge fest.

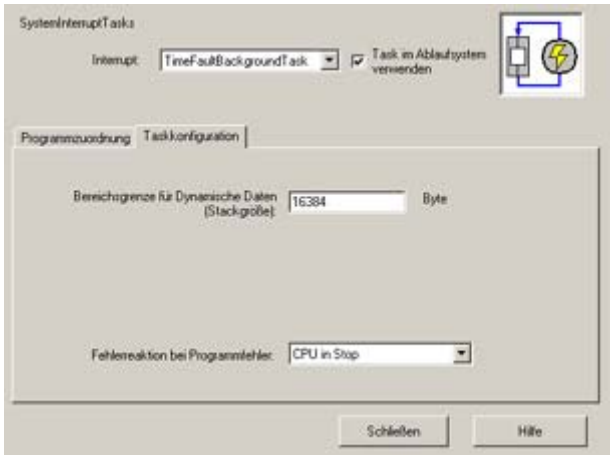


Bild 6-17 Konfiguration der SystemInterruptTasks (Taskkonfiguration)

4. Wechseln Sie in das Register **Taskkonfiguration**.
5. Konfigurieren Sie die Task.
6. Wiederholen Sie die Schritte 3 bis 5 für alle zu konfigurierenden SystemInterruptTasks.
7. Wenn Sie die Technologischen Alarme konfigurieren wollen:
   Klicken Sie auf **Alarmkonfiguration**.

Taskkonfiguration - SystemInterruptTasks
Im Register **Taskkonfiguration** parametrieren Sie die Fehlerreaktion bei Programmfehlern.

Folgende Parameter können Sie einstellen:

Feld/Schaltfläche		Bedeutung/Hinweis
Bereichsgrenzen für Dynamische Daten		Hier tragen Sie die Größe des Stacks in Byte für diese Task ein. Beim Ablauf der Programme, die dieser Task zugeordnet sind, wird diese Größe im Stack für Daten bereitgestellt. Richtwert ist 16 KB für eine Task.
Fehlerreaktion bei Programmfehler		Hier wählen Sie die Fehlerreaktion, wenn bei Programmen Verarbeitungsfehler auftreten. Programmfehler sind z. B. fehlerhafte Operationen bei Gleitkommazahlen, Division durch Null und Überschreiten von Feldgrenzen.
	CPU in STOP	CPU wechselt in den Zustand **STOP** und die ShutdownTask wird gestartet.
	ExecutionFaultTask	Es wird die ExecutionFaultTask gestartet. Alle Programme die dieser Task zugewiesen sind werden gestartet. Sind keine Programme zugeordnet, geht die CPU in **STOP**. Die Task, in der der Fehler auftritt, wird beendet.

Siehe auch

Information zum Start einer Task: TaskStartInfo (TSI) (Seite 255)

6.2.7 UserInterruptTasks

UserInterruptTasks sind vorgesehen für anwenderdefinierte Aktionen.

Es stehen zwei UserInterruptTasks zur Verfügung: **UserInterruptTask_1** und **UserInterruptTask_2**.

Bei der UserInterruptTask muss eine festgelegte Bedingung vorgegeben werden. Jedes Mal wenn die Bedingung erfüllt ist, wird die UserInterruptTask gestartet.

Falls Sie eine **UserInterruptTask** verwenden wollen, muss im Ablaufsystem die **IPOSynchronousTask** verwendet werden.

UserInterruptTasks sind bei **StartupTask** und **ShutDownTask***nicht aktiv*.

Start einer UserInterruptTask

UserInterruptTasks werden automatisch gestartet, sobald die anwenderdefinierte Interrupt-Bedingung erfüllt ist. Die Interrupt-Bedingung wird im Interpolator-Takt geprüft.

Bei gleichzeitigem Start wird die **UserInterruptTask_1** vor der **UserInterruptTask_2** bearbeitet.

Hinweis

Kommt es während des Absteuerns zu einem anwenderdefinierten Interrupt, wird die UserInterruptTask nicht mehr gestartet.

Während des Absteuerns (ShutdownTask) ist das Starten einer UserInterruptTask nur über den Tasksteuerbefehl _startTask() möglich.

Beenden einer UserInterruptTask

UserInterruptTasks werden automatisch nach Ablauf der Programme beendet, welche der UserInterruptTask zugeordnet sind.

Im Register **Programmzuordnung** weisen Sie die erstellten und übersetzen Programme der gewählten UserInterruptTask zu und legen die Ablaufreihenfolge fest.

Folgende Parameter können Sie einstellen:

Feld/Schaltfläche		Bedeutung/Hinweis
Für Task		Hiermit wählen Sie eine der zwei UserInterruptTasks aus, der Sie die Programme zuweisen wollen. Sie können einer UserInterruptTask mehrere Programme zuweisen.
	UserInterruptTask_1 und UserInterruptTask_2	Vordefinierte Namen der UserInterruptTask.
festgelegte Bedingung		Hier legen Sie die Bedingung nach IEC 61131 für die gewählte UserInterruptTask fest. Die eingetragene Bedingung wird beim Betrieb im Interpolator-Takt geprüft. Ist die Bedingung erfüllt, wird die UserInterruptTask mit den zugewiesenen Programmen gestartet. Die Bedingung tragen Sie, mit Hilfe des Symbol-Browser (Variablen per Drag & Drop) und dem Register **Befehlsbibliothek** im Projektnavigator (Operatoren per Drag & Drop), in das Eingabefeld ein. Es sind nur einfache Bedingungen (logische Verknüpfungen von Eingängen und lokalen Variablen der CPU) und boolsche Variablen zugelassen.
Task im Ablaufsystem verwenden		Aktivieren Sie die Kontrollkästchen, wenn die Task im Ablaufsystem angezeigt und verwendet werden soll. Ist die Kontrollkästchen deaktiviert, können Sie dieser Task keine Programme zuweisen.

UserInterruptTasks konfigurieren

1. Klicken Sie im Ablaufebenenbaum auf **UserInterruptTasks**.
2. Wählen Sie im Auswahlfeld **Für Task** eine der UserInterruptTasks aus.
3. Legen Sie die Bedingung für das Starten dieser Task fest.

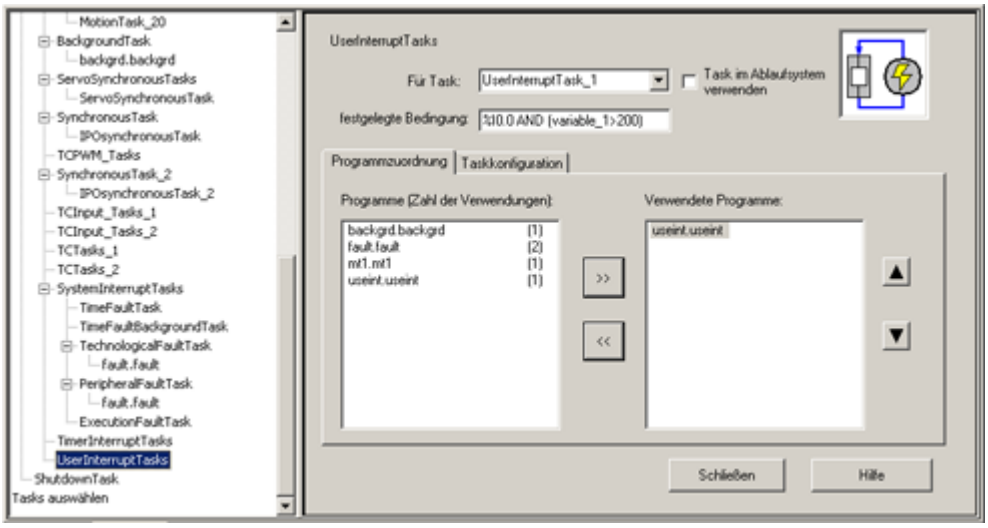
Bild 6-18 Konfiguration der UserInterruptTasks (Programmzuordnung)

4. Ordnen Sie im Register **Programmzuordnung** die gewünschten Programme dieser Task zu und legen Sie die Ablaufreihenfolge fest

Bild 6-19 Konfiguration der UserInterruptTasks (Taskkonfiguration)

5. Wechseln Sie in das Register **Taskkonfiguration**.
6. Konfigurieren Sie die Task.
7. Wiederholen Sie die Schritte 3 bis 6 für die zweite UserInterruptTask.

Bedingung für UserInterruptTask formulieren



Legende zum Bild - Beispiel:

Immer wenn der digitale Eingang 0.0 = TRUE und die globale Variable variable_1" einen Wert > 200.0 hat, dann wird die UserInterruptTask_1 einmal bearbeitet.

Bild 6-20 Konfiguration einer UserInterruptTasks mit Bedingung

Die Bedingung für das Starten einer UserInterruptTask geben Sie als Formel gemäß IEC 61131 (Structured Text) ein. Folgende Variabeln können Sie verwenden:

- geräteglobale Variablen
- Unit-Variablen in einer ST/MCC oder KOP/FUP-Quelle

SIMOTION SCOUT unterstützt Sie bei der Erstellung der Formel.

So formulieren Sie eine Bedingung für den Start des UserInterruptTasks:

1. Wählen Sie unter **Für Task** den UserInterrupt, für den Sie die Startbedingung festlegen wollen.
2. Übernehmen Sie aus dem Register **Befehlsbibliothek** die Operatoren per Drag & Drop ins Textfeld **festgelegte Bedingung**.

   Alternativ können Sie die Operatoren auch direkt in das Textfeld eingeben.
3. Übernehmen Sie aus dem Symbol-Browser die Operanden per Drag&Drop oder durch Kopieren und Einfügen ins Textfeld **festgelegte Bedingung**. Alternativ können Sie die Operanden auch direkt in das Textfeld eingeben.

Bedingungen mit der Befehlsbibliothek erstellen

Im Register **Befehlsbibliothek** sind mathematische Operationen und Funktionen in einer Liste dargestellt. Diese können Sie zum Erstellen von Bedingungen verwenden.

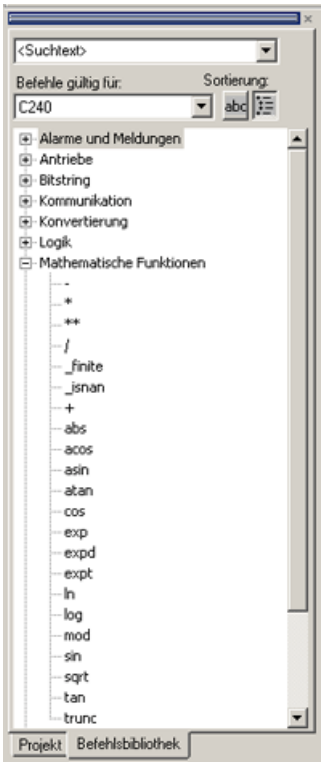


Bild 6-21 Befehlsbibliothek im SCOUT

Einzelne Operationen öffnen Sie durch Klicken auf das Pluszeichen vor den Ordnern. Die Operatoren können Sie per Drag&Drop in das jeweilige Textfeld ziehen. Diese Befehle können Sie nutzen, um z. B. Bedingungen zu definieren.

Taskkonfiguration - UserInterruptTasks

Im Register **Taskkonfiguration** parametrieren Sie die Fehlerreaktion bei Programmfehlern.

Folgende Parameter können Sie einstellen:

Feld/Schaltfläche		Bedeutung/Hinweis
Bereichsgrenzen für Dynamische Daten		Hier tragen Sie die Größe des Stacks in Byte für diese Task ein. Beim Ablauf der Programme, die dieser Task zugeordnet sind, wird diese Größe im Stack für Daten bereitgestellt. Richtwert ist 16 KB für eine Task.
Fehlerreaktion bei Programmfehler		Hier wählen Sie die Fehlerreaktion, wenn bei Programmen Verarbeitungsfehler auftreten. Programmfehler sind z. B. fehlerhafte Operationen bei Gleitkommazahlen, Division durch Null und Überschreiten von Feldgrenzen.
	CPU in STOP	CPU wechselt in den Zustand **STOP** und die ShutdownTask wird gestartet.
	ExecutionFaultTask	Es wird die ExecutionFaultTask gestartet. Alle Programme die dieser Task zugewiesen sind werden gestartet. Sind keine Programme zugeordnet, geht die CPU in **STOP**. Die Task, in der der Fehler auftritt, wird beendet.

Siehe auch

Programme den Ablaufebenen/Tasks zuweisen (Seite 236)

6.2.8 ShutdownTask

Die **ShutdownTask** ist vorgesehen für gezieltes Eingreifen in den Betriebszustandsübergang von **RUN** nach **STOPU**/**STOP** oder zur Programmierung von Anhalteabläufen mit vorparametrierter Bremsrampe wie z. B. gezieltes Setzen von Ausgängen, definiertes Herunterfahren von Achsen.

Die ShutdownTask wird *nicht* bei Spannungsausfall aufgerufen.

Bei der ShutdownTask muss in der **Taskkonfiguration** die **Zeitüberwachung** vorgegeben werden: Die maximale Dauer für die Ausführung der ShutdownTask ist konfigurierbar, 0 ms = keine Überwachung. Nach dieser Zeit geht die CPU in **STOP**.

In der ShutdownTask ist der direkte Zugriff auf die Peripherie möglich. Zugriff auf das Prozessabbild und symbolische I/O-Variablen ist nur eingeschränkt möglich. Der Zugriff über das Prozessabbild ist *nicht sinnvoll*, da das Prozessabbild nicht mehr aktualisiert wird.

Weitere Informationen hierzu siehe **SIMOTION ST Structured Text**, "Zugriff auf Ein- und Ausgänge (Prozessabbild, I/O-Variablen)"

Im Register **Programmzuordnung** weisen Sie die erstellten und übersetzten Programme der ShutdownTask zu und legen die Ablaufreihenfolge fest.

Hinweis

Programmfehler in der **ExecutionFaultTask** und in der **ShutdownTask** führen unmittelbar in den Betriebszustand **STOP**.

ShutdownTask konfigurieren

1. Klicken Sie im Ablaufebenenbaum auf **ShutdownTask**.

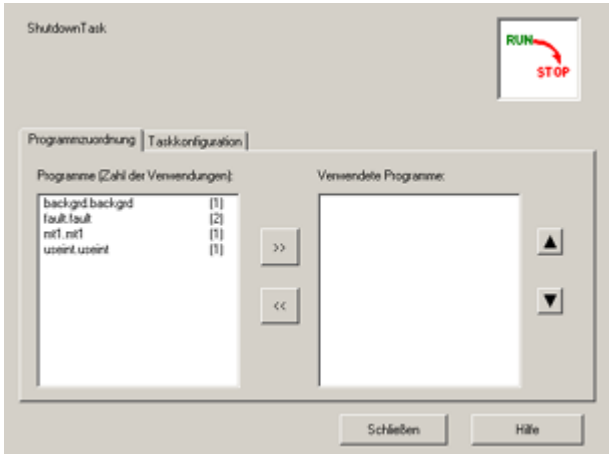


Bild 6-22 Konfiguration der ShutdownTask (Programmzuordnung)

2. Ordnen Sie im Register **Programmzuordnung** die gewünschten Programme dieser Task zu und legen Sie die Ablaufreihenfolge fest.

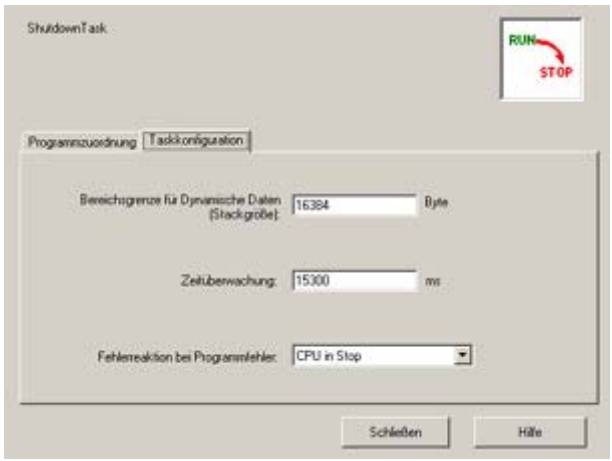


Bild 6-23 Konfiguration der ShutdownTask (Taskkonfiguration)

3. Wechseln Sie in das Register **Taskkonfiguration**.
4. Geben Sie ggf. die **Bereichsgrenze für dynamische Daten (Stackgröße)** an.
5. Geben Sie einen Wert an für die **Zeitüberwachung**.
6. Legen Sie die **Fehlerreaktion bei Programmfehler** fest (z. B. **ExecutionFaultTask**).

Taskkonfiguration - ShutdownTask

Im Register **Taskkonfiguration** parametrieren Sie die Zeitüberwachung.

Folgende Parameter können Sie einstellen:

Feld/Schaltfläche	Bedeutung/Hinweis
Bereichsgrenzen für Dynamische Daten	Hier tragen Sie die Größe des Stacks in Byte für diese Task ein. Beim Ablauf der Programme, die dieser Task zugeordnet sind, wird diese Größe im Stack für Daten bereitgestellt. Richtwert ist 16 KB für eine Task.
Zeitüberwachung	Hier legen Sie die Zykluszeit in ms fest, die für die Ausführung der ShutdownTask zur Verfügung steht. Tragen Sie einen Wert für die Zeitüberwachung ein. Ist für die Zeitüberwachung kein Wert oder der Wert 0 eingetragen, ist die Zeitüberwachung inaktiv.
Fehlerreaktion bei Programmfehler	Hier wählen Sie die Fehlerreaktion, wenn bei Programmen Verarbeitungsfehler auftreten. Programmfehler sind z. B. fehlerhafte Operationen bei Gleitkommazahlen, Division durch Null und Überschreiten von Feldgrenzen.
CPU in STOP	CPU wechselt in den Zustand **STOP**.

Siehe auch

Programme den Ablaufebenen/Tasks zuweisen (Seite 236)

Zeitüberwachung (Seite 256)

6.3 Ablaufsystem konfigurieren

Als Konfigurieren des Ablaufsystems bezeichnet man:

- Zuordnung von Anwenderprogrammen und das Festlegen der Eigenschaften der Tasks
- Freigabe der verwendeten Tasks
- Auswahl der Taktquelle und Einstellung der Systemtakte

Siehe auch

Servo_fast (Applikationstakt für schnelles Bussystem) (Seite 257)

6.3.1 Programme den Ablaufebenen/Tasks zuweisen

Die Programme müssen den Ablaufebenen zugeordnet werden. Nur dann werden die Programme abgearbeitet.

Mit SIMOTION SCOUT können Sie Programme eines MCC-Quelle, einer ST- oder KOP/FUP-Quelle nach der Erstellung einer oder mehreren Tasks zuordnen.

Sie können einer Task mehrere Programme zuordnen.

Die zugeordneten Programme werden in der Reihenfolge der Auflistung, die mit SIMOTION SCOUT vorgegeben und geändert werden kann, abgearbeitet.

Werden mehrere Programme einer Task zugeordnet, muss das erste Programm fertig bearbeitet sein, damit das nächste Programm in dieser Task gestartet wird. Befindet sich z. B. das erste Programm in einer Endlosschleife, wird das zweite Programm nie bearbeitet.

Sie können auch ein Programm mehreren Tasks zuordnen, die dann unabhängig voneinander bearbeitet werden.

Durch Programmzuordnung legen Sie u. a. fest:

- Die Priorität, mit der die Programme ablaufen
- Das Ablaufverhalten: sequentiell/zyklisch
- Das Initialisierungsverhalten von Programmvariablen

  Siehe Programmierhandbuch **SIMOTION MCC** bzw. **SIMOTION ST** - "Zeitpunkt der Variableninitialisierung" und Einfluss des Compilers auf die Variableninitialisierung (Seite 317) .

Bitte beachten Sie bei der Zuordnung eines Programmes zu einer oder mehreren Tasks:

- Bevor Programme zugeordnet werden können, müssen diese fehlerfrei übersetzt (kompiliert) sein.
- Die Zuordnung muss vor dem Laden des Programmes ins Zielsystem geschehen.

- Es besteht die Möglichkeit, dass das Programm (wenn mehreren Tasks zugeordnet) während seiner Ausführung von einer anderen Task aufgerufen wird. Systemseitig werden keine Maßnahmen zur Konsistenzsicherung von Daten getroffen.
- Ist ein Programm einer Task zugeordnet, dann bleibt diese Zuordnung bei einer Neuübersetzung erhalten.

Hinweis

DCC-Tasks werden über den DCC-Editor zugeordnet, siehe Beschreibung des DCC-Editors und Ablaufmodell für DCC-Bausteine (DCB) (Seite 290).

Ablaufsystem - Programmzuordnung

Im Register **Programmzuordnung** weisen Sie erstellte und übersetzte Programme den verschiedenen Tasks der Ablaufebenen zu.

Folgende Parameter können Sie einstellen:

Feld/Schaltfläche	Bedeutung/Hinweis
Programme	Hier wird eine Liste aller übersetzten Programme angezeigt, die im Projekt verfügbar sind. Nicht übersetzte Programme werden nicht angezeigt. Die Zahl hinter dem Programmnamen gibt an, wie oft das Programm den verschiedenen Tasks der Ablaufebenen zugeordnet wurde. MCC Charts und KOP/FUP-Programme werden nach dem Einfügen sofort angezeigt, sofern sie beim Einfügen als "exportierbar" angelegt wurden.
Zuordnen	Hiermit können Sie markierte Programme der Task zuordnen. Markieren Sie das Programm in der Liste Programme und klicken Sie auf die Pfeil-Schaltfläche. Das Programm wird der Task zugeordnet und wird in der Liste der Task angezeigt.
Entfernen	Hiermit können Sie Programme, die einer Task zugeordnet sind, wieder entfernen. Markieren Sie das Programm in der Liste Task und klicken Sie auf die entsprechende Pfeil-Schaltfläche. Das Programm wird aus der Task entfernt.
Task	Hier wird eine Liste aller Programme angezeigt, die dieser Task zugeordnet sind. Die Reihenfolge der Programme in der Liste entspricht der Ablaufreihenfolge beim Abarbeiten der Programme. Das oberste Programm der Liste wird als erstes abgearbeitet.
Pfeil nach oben	Mit dem Pfeil verschieben Sie das markierte Programm innerhalb der Task um eine Position nach oben. Dadurch legen Sie die Ablaufreihenfolge der Programme innerhalb der Task fest.
Pfeil nach unten	Mit dem Pfeil verschieben Sie das markierte Programm innerhalb der Task um eine Position nach unten. Dadurch legen Sie die Ablaufreihenfolge der Programme innerhalb der Task fest.

Programme den Tasks zuordnen

1. Markieren Sie im Projektnavigator das SIMOTION Gerät und wählen Sie im Menü **Zielsystem > Ablaufssystem konfigurieren** oder doppelklicken Sie auf **ABLAUFSYSTEM**.

   Im Arbeitsbereich der Workbench öffnet das Fenster **ABLAUFSYSTEM**.

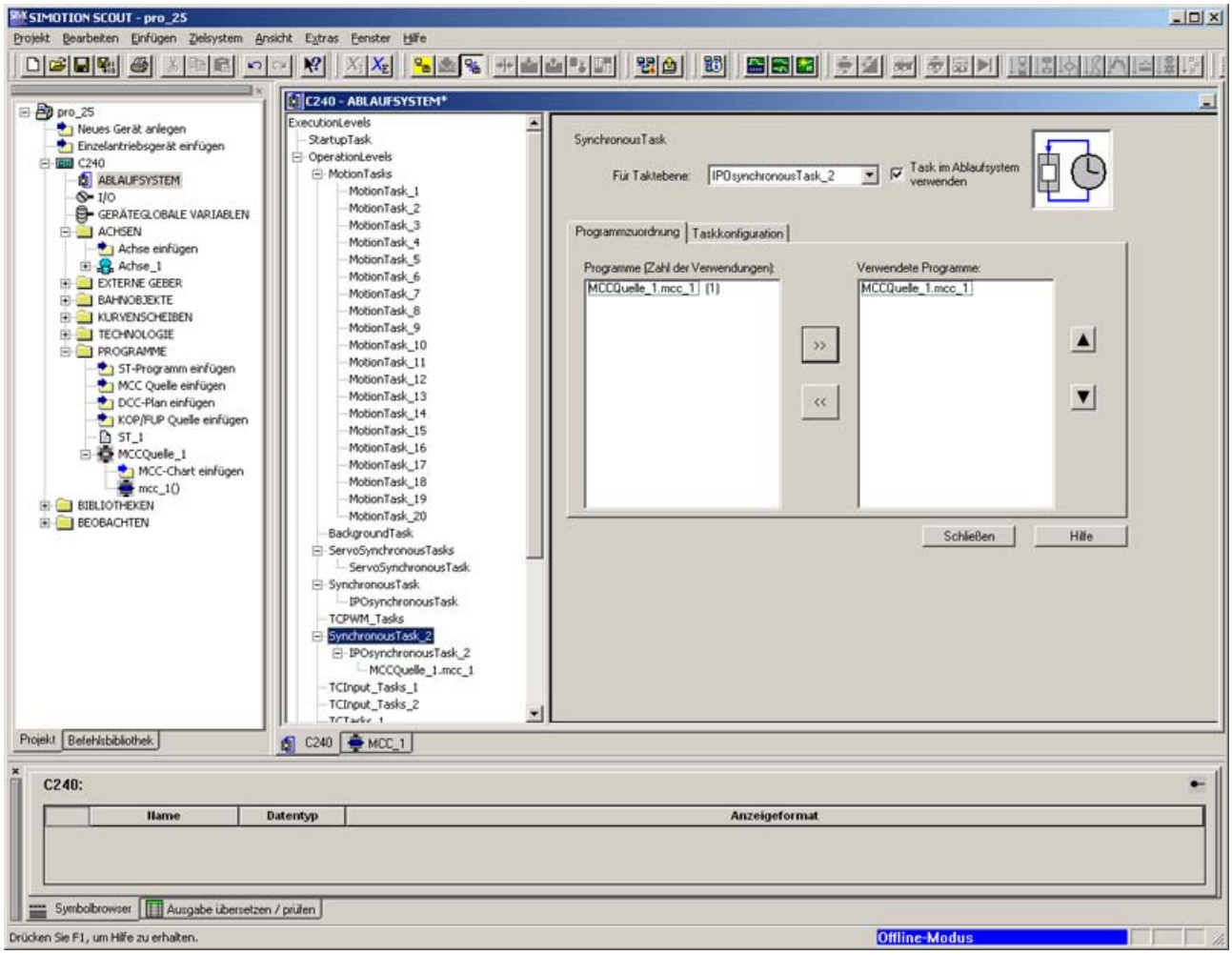


Bild 6-24 Konfiguration des Ablaufsystems im SCOUT

In der linken Fensterhälfte sehen Sie den Ablaufebenenbaum. Er zeigt als Einträge die Ablaufebenen / Tasks an, bei denen **Task im Ablaufsystem verwenden** angeklickt ist.

Im Ordner **OperationLevels** sind die Tasks zusammengefasst, die beim Betriebszustand **RUN** zur Verfügung stehen.

Unterhalb jeder Ablaufebene bzw. Task-Bezeichnung werden die konfigurierten Tasks mit den zugeordneten Programmen angezeigt.

1. Markieren Sie die zu konfigurierende Task.
2. Wählen Sie das Register **Programmzuordnung**.

   Im linken Listenfeld werden alle verfügbaren Programme (ST-Programme, MCC-Charts und KOP/FUP mit Erstelltyp Task) aufgelistet.

3. Markieren Sie im linken Listenfeld die Programme, die Sie der Task zuordnen wollen.
4. Klicken Sie auf ».

   Die zugeordneten Programme werden im rechten Listenfeld aufgelistet.

   Im linken Listenfeld werden sie weiterhin aufgelistet. Programme können mehreren Tasks zugeordnet werden. Die Anzahl der erfolgten Zuordnungen wird in Klammern dargestellt.
5. Wählen Sie das Register **Taskkonfiguration** und nehmen Sie die dort ggf. weitere Einstellungen vor, z. B.:
   - Fehlerreaktion bei Programmfehler
   - Zeitüberwachungen zyklischer Tasks
   - Startverhalten von MotionTasks

Nachdem Sie ein Programm einer oder mehreren Tasks zugeordnet haben, können Sie die Verbindung zum Zielsystem herstellen, das Projekt in das Zielsystem laden und es anschließend starten.

Ablaufreihenfolge ändern

Programme werden in der Reihenfolge ihres Eintrags ausgeführt. Sie können diese Reihenfolge ändern.

1. Markieren Sie im rechten Listenfeld den Eintrag, den Sie verschieben wollen.
2. Klicken Sie auf ▲ bzw. ▼, um das Element aufwärts oder abwärts zu verschieben.
3. Wiederholen Sie Schritt 2. so oft wie nötig.

Task-Name

Die Namen der MotionTasks können geändert werden.

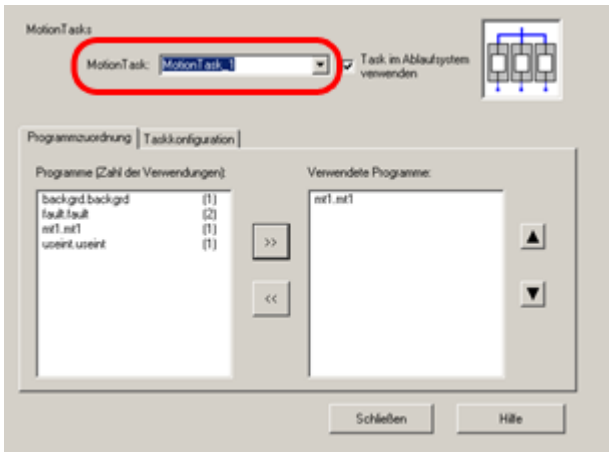


Bild 6-25 Task-Namen im SCOUT

1. Markieren Sie links die gewünschte Task.
2. Geben Sie in der Aufklappliste rechts den gewünschten neuen Task-Namen ein.

Tasksteuerung

Für die Tasksteuerung (z. B starten, stoppen usw.) stehen in den Programmiersprachen MCC, ST und KOP/FUP verschiedene Befehle zur Verfügung.

Siehe Programmierhandbuch **SIMOTION MCC**, **SIMOTION ST** und **SIMOTION KOP/FUP**

In diesen Dokumentationen erhalten Sie auch Informationen zur Verwendung dieser Tasksteuerbefehle und einige Beispiele.

Stackgröße

Im SIMOTION SCOUT kann im Task-Konfigurationsfenster im Register Taskkonfiguration die Stackgröße (Bereichsgrenze für dynamische Daten) der jeweiligen Task eingestellt werden. Ein Defaultwert ist jeweils vorgegeben.

Weitere Information hierzu finden Sie im Programmierhandbuch **SIMOTION ST**, "Größe des Lokaldatenstacks einstellen".

Siehe auch

Task-Prioritäten (Seite 197)

6.3.2 Auswahl der Taktquelle

Die Auswahl der Taktquelle erfolgt durch die Geräteprojektierung in der **HW-Konfiguration**.

Sobald Sie eine der Schnittstellen als taktsynchrone DP/PN-Schnittstelle parametriert haben (Anwahl **Äquidistanz** im Eigenschaftsdialog), wird der eingestellte Takt als Bus-Takt verwendet.

Die DP/PN- Kommunikationsebene sowie die Servo- und Interpolator-Ebene werden auf den Bus-Takt synchronisiert. Diese Einstellung ist zwingend erforderlich, wenn Sie die Motion Control Funktionen des Technologiepaketes **TP CAM/TP PATH/TP CAM_ext** in Verbindung mit digitalen Antrieben nach PROFIDRIVE V3 am PROFIBUS DP / PROFINET benutzen wollen. Antriebe, die diese Kommunikation unterstützen und erfordern sind z. B.: SIMODRIVE 611U, MASTERDRIVES MOTION CONTROL, SINAMICS. Eine Synchronisierung auf den Bus-Takt ist auch erforderlich, wenn Sie aus Ihrer Applikation taktsynchron auf Peripherie zugreifen müssen.

Digitale Antriebe, die den taktsynchronen Betrieb nicht unterstützen, können Sie trotzdem als Drehzahlachsen am taktsynchronen Bus betreiben. Solche Antriebe sind z.B. Micromaster MM4 bzw. MASTERDRIVES VC.

Wenn Sie keine taktsynchrone DP/PN-Schnittstelle verwenden, können Sie den Grundtakt des Systems einstellen. Die Servo- und Interpolator-Ebene werden auf den Grundtakt synchronisiert. Diese Einstellung können Sie wählen, wenn Sie das Technologiepaket **TP CAM/TP PATH/TP CAM_ext** nicht oder ausschließlich mit analogen Antrieben an SIMOTION verwenden.

6.3.3 Systemtakte festlegen

Nach der Auswahl der Taktquelle legen Sie die Abtastzeiten der von dem Grundtakt abgeleiteten isochronen (taktsynchronen) Ablaufebenen fest. Optional kann ab V4.2 ein zusätzlicher schnellerer Servo-Takt und IPO-Takt projektiert werden, siehe auch Servo_fast (Applikationstakt für schnelles Bussystem) (Seite 257).

Hierzu gehören:

- **Basis-Takt / Bus-Takt (DP-Takt / PN-Takt)**: siehe Tabelle zu Ablaufsystem - Systemtakte

  Der "Basis-Takt" ist der Grundtakt, wenn in HW Konfig keine taktsynchrone DP/PNSchnittstelle

  parametriert wurde. Der "Bus-Takt" leitet sich vom äquidistanten Buszyklus ab und wird in HW Konfig eingestellt. Die Bezeichnung ändert sich je nach Einstellung im Dialog **Systemtakte einstellen**. Der Basis-Takt/Bus-Takt (DP-Takt oder PN-Takt) wird als Grundlage für die Einstellung weiterer Takte verwendet.

- **Servo_fast (schneller Lageregler-Takt (ab V4.2)**

  Eingänge/Ausgänge werden im Servo_fast-Takt aktualisiert (optional). Der Servo_fast ist dem schnelleren Bussystem (PROFINET IO) zugeordnet und wird in HW Konfig eingestellt.

- **IPO_fast (schneller Interpolator-Takt) (ab V4.2)**

  Im IPO_fast-Takt wird die Bewegungsführung für die Achsen am schnellen Bussystem berechnet (optional).

- **Servo-Takt (Lageregler-Takt) / T1(DCC)-Takt**

  Eingänge/Ausgänge werden im Servo-Takt aktualisiert.

  Im Servo-Takt wird die Lageregelung für die Achsen sowie die Bearbeitung der zentralen und dezentralen Peripherie bearbeitet. Der Servo-Takt kann im Verhältnis 1:1 bzw. 1:2 zum Bus-Takt betrieben werden. In diesem Takt wird die **ServoSynchronousTask** bearbeitet.

  Bei PROFINET IO Taktsynchron gilt Folgendes:

  – Die DP-Masterschnittstelle muss im gleichen Takt laufen wie der Servo-Takt.

  – Bei SIMOTION C, P und D muss der Bustakt für die externen PROFIBUS-Schnittstellen mindestens 1 ms betragen, um diese taktsynchron betreiben zu können.

  – Ist der PN-Takt untersetzt (z. B. 0,5 ms), dann muss der Servo-Takt gleich dem PROFIBUS-Takt sein (bei SIMOTION D: auch gleich dem Bus-Takt des PROFIBUS Integrated).

  Hinweis

  Das Untersetzungsverhältnis zwischen dem Bus- und Servo - Takt ist auch am Antrieb als Masterapplikations-Zyklus einzustellen. Diese Einstellung ist erforderlich für die gegenseitige Lebenszeichenüberwachung. Hinweise hierzu finden Sie in den Beschreibungen zu den Antrieben.

- **Interpolator-Takt(IPO-Takt) / T2(DCC)-Takt**

  Im IPO-Takt wird die Bewegungsführung für die Achsen berechnet.
  Es wird in diesem Takt die **IPOSynchronousTask** bearbeitet. Der IPO-Takt kann im Verhältnis 1:1 bis 1:6 zum Servo-Takt untersetzt werden.

- **Interpolator-Takt_2(IPO_2-Takt) / T3(DCC) Takt**

  Der IPO-Takt_2 ist Basis für die Bewegungsführung niederpriorer Achsen.
  In diesem Takt werden die **IPOSynchronousTask_2** und die **PWM Task** (TControl) bearbeitet.

  Der IPO_2-Takt kann im Verhältnis 1:2 bis 1:64 zum IPO-Takt untersetzt werden

- **DCCAux T4(DCC) - Takt**

  Der DCCAux Takt kann im Verhältnis 1:2 bis 1:32 zum T3(DCC) Takt untersetzt werden.

- **DCCAux_2 T5(DCC) Takt**

  Der DCCAux_2 Takt kann im Verhältnis 1:2 bis 1:32 zum DCCAux Takt untersetzt werden.

- **PWM-Takt**: für Technologiepaket TControl

Hinweis

Die Einstellungen der Systemtakte beeinflussen den Systemablauf in hohem Maße. Nehmen Sie deshalb die Einstellungen mit großer Sorgfalt vor.

Systemtakte einstellen

1. Rufen Sie das Konfigurationsfenster im Hauptmenü über **Zielsystem > Experte > Systemtakte einstellen...** auf, oder im Kontextmenü des Gerätes bzw. im Kontextmenü des **ABLAUFSYSTEMs** über **Experte > Systemtakte einstellen**.

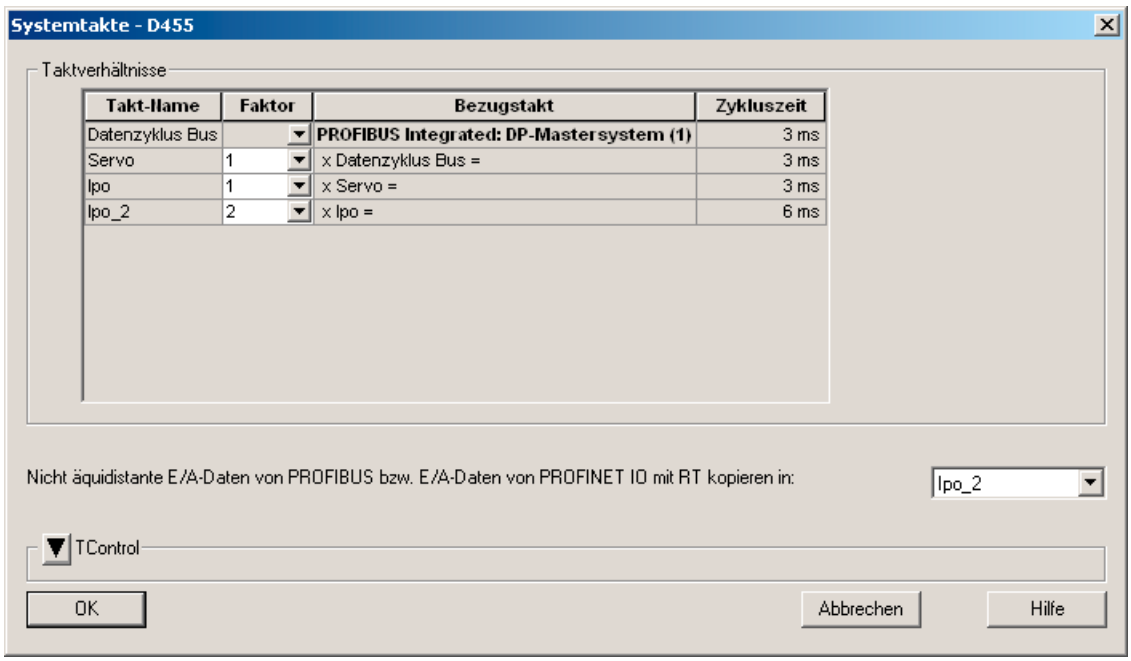


Bild 6-26 Einstellung der Systemtakte im SCOUT

2. Legen Sie den Basis-/Bus-Takt fest (nur wenn Äquidistanz nicht konfiguriert ist). Bei äquidistantem Bus wird der Bus-Takt in HW Konfig eingestellt.

   Mögliche Werte: 1,0 ... 8,0 ms bei PROFIBUS

   Mögliche Werte bei PROFINET:

   – Ab V4.0 bei SIMOTION P mit PROFINET sowie SIMOTION D4x5/D4x5-2: 0,5 ... 4,0 ms
   – Ab V4.1 bei SIMOTION P mit PROFINET: 0,25 ms bis 4 ms.

   Die Zykluszeit bei PROFIBUS muss ein ganzzahliges Vielfaches von 0,125 ms sein, bei SIMOTION C sind es 0,25 ms.
   Die Zykluszeit bei PROFINET muss ein ganzzahliges Vielfaches von 0,125 ms sein.
   Ändern Sie den Wert in **HW Konfig**, falls dies nicht der Fall sein sollte.

3. Legen Sie die Verhältnisse zwischen den Takten fest.

Hinweis

Wenn Sie einen umfangreichen Hardwareaufbau projektieren, kann es bei einem zu klein gewählten Basis-Takt/Bus-Takt dazu führen, dass die CPU nicht in den Betriebszustand RUN wechselt. Beachten Sie in diesem Fall Einträge im Diagnosepuffer zum Thema Zeitüberläufe.

Systemtakte einstellen mit zwei Servo-Takten (Servo_fast)

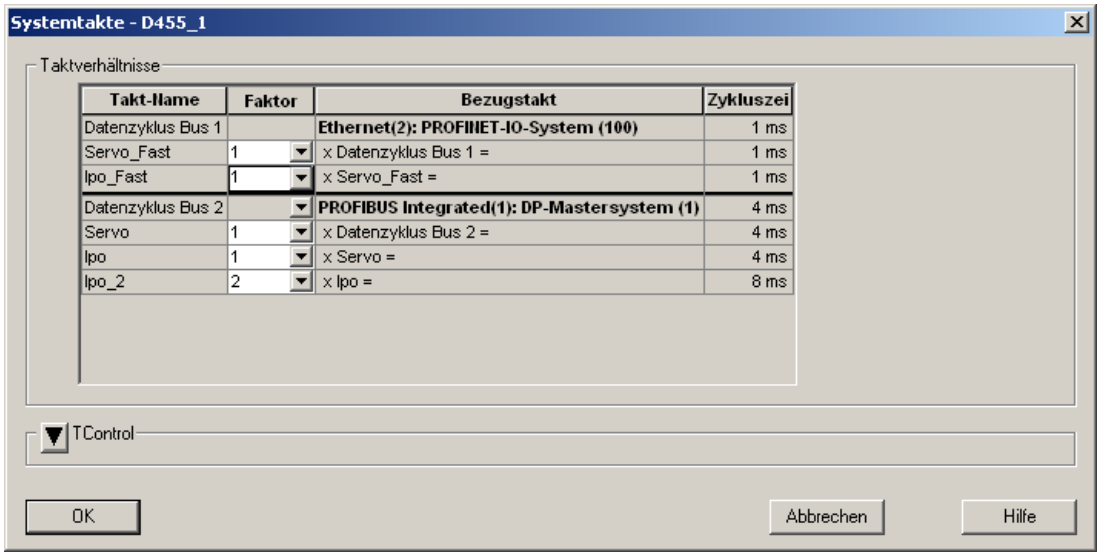


Bild 6-27 Systemtakte Servo und Servo_fast

Wie Sie einen zweiten Servo-Takt projektieren, finden Sie im Funktionshandbuch *Kommunikation* unter Servo_fast, Taktuntersetzung zum Servo an der PROFINET-Schnittstelle beschrieben.

Systemtakte für DCC

Die Einträge für DCC werden dann eingeblendet, sobald ein Plan unter Programme eingefügt worden ist. DCC-Pläne werden automatisch gestartet, wenn die entsprechende Task gestartet wird.

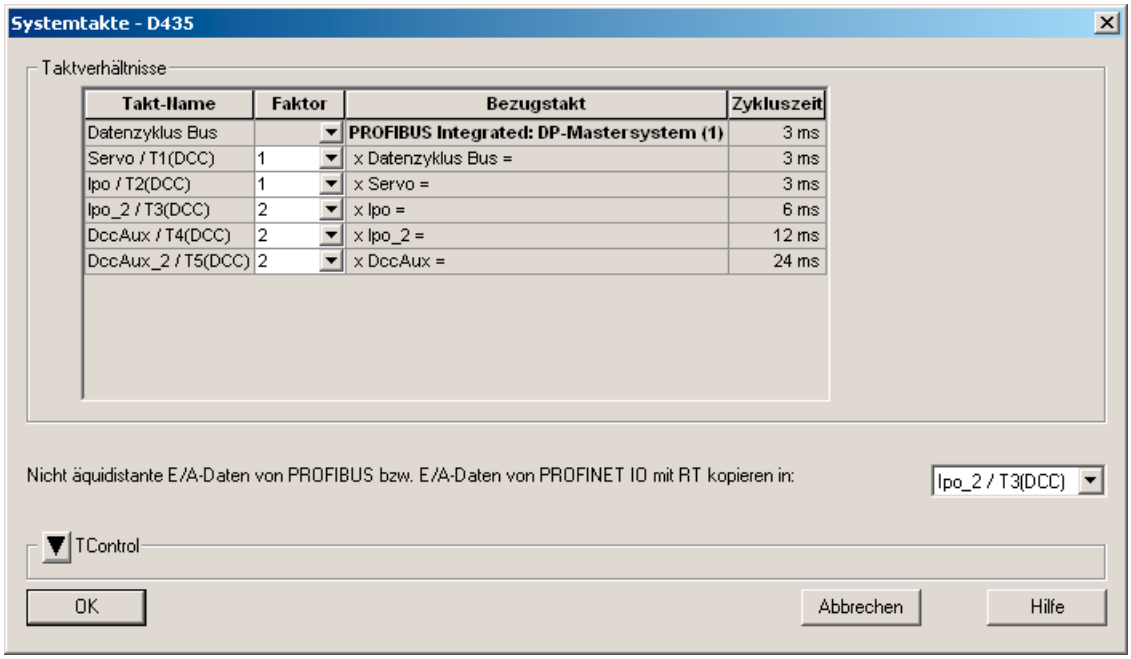


Bild 6-28 Einstellung der Systemtakte im SCOUT mit DCC

Systemtakte für TControl einstellen

1. Im Konfigurationsfenster **Systemtakte** können Sie über die Taste **TControl** die Einstellungen für die TControl-Systemtasks einblenden.
2. Über das Kontrollkästchen **Systemtasks für TControl verwenden** können Sie die TControl-Systemtasks aktivieren oder deaktivieren.

   Ist **Systemtasks für TControl verwenden** aktiviert, werden die speziellen Tasks für die Temperaturkanäle im Ablaufsystem angezeigt.

   Wenn Sie keinen Temperaturkanal projektiert haben, sollten Sie die Systemtasks für TControl deaktivieren, da die Systemtasks dann unnötige Rechenzeit beanspruchen.

3. Weiterhin legen Sie hier die Taktverhältnisse der TControl-Tasks fest.
   - Der **PWM**-Takt muss ein ganzzahliges Vielfaches des Lagereglertakts sein.
   - Der Takt der **InputTask_1/2** ist abhängig vom PWM-Takt.
   - Der Takt der **PostTask_1/2** ist abhängig vom Takt der **InputTask_1/2**.

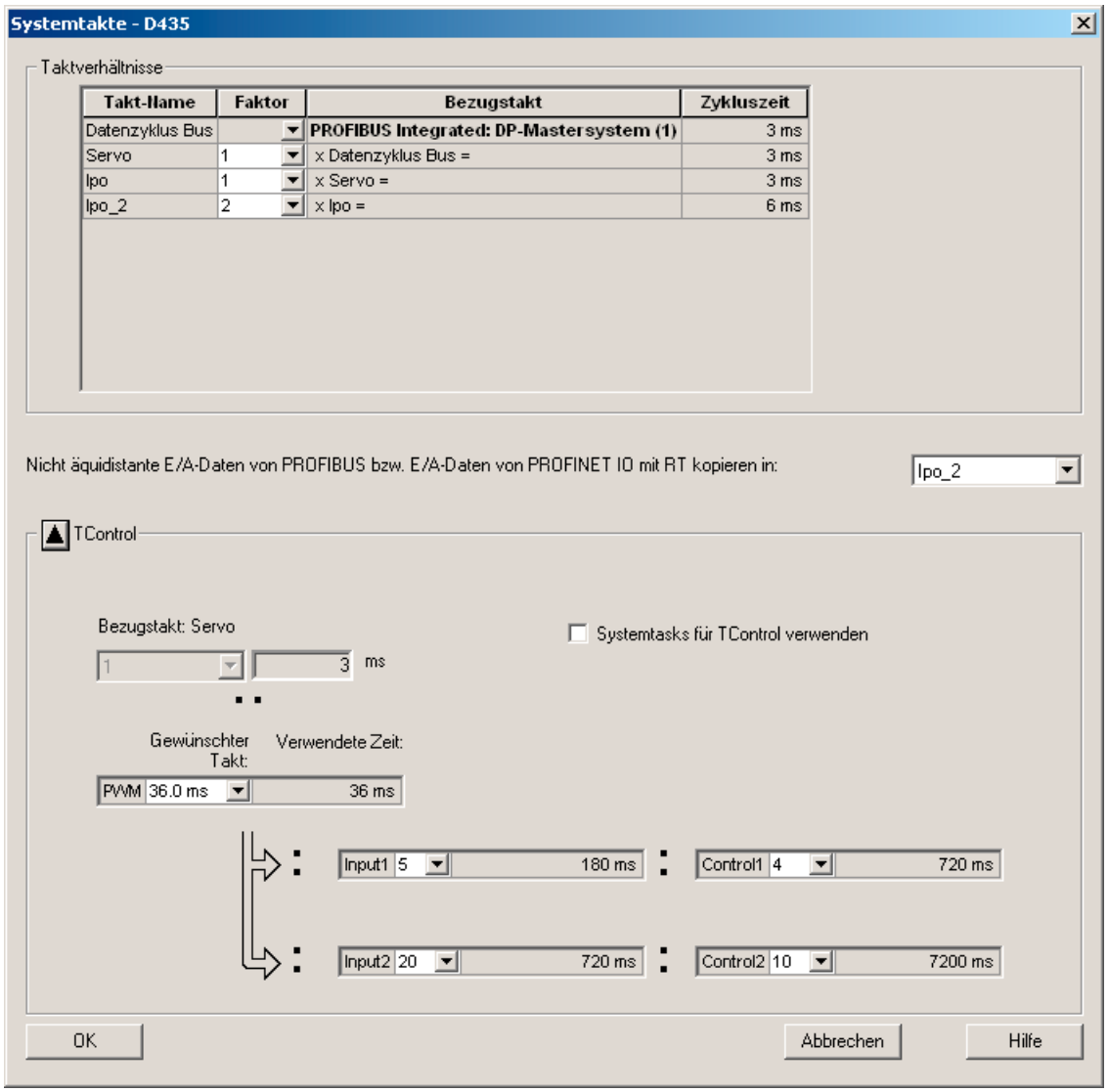


Bild 6-29 Einstellung der Systemtakte für TControl im SCOUT

Ablaufsystem - Systemtakte

Folgende Parameter können Sie einstellen:

Feld/Schaltfläche		Bedeutung/Hinweis
Taktverhältnisse		
	Basis-Takt bzw. **Bus-Takt**	Der Basis-Takt ist der Grundtakt, wenn in HW Konfig keine taktsynchrone DP-/PN-Schnittstelle parametriert wurde. Wenn die Einstellung "Äquidistanter Buszyklus" aktiviert ist, kann der "Bus-Takt" in diesem Dialogfeld nicht eingestellt werden. Der "Bus-Takt" wird in HW Konfig eingestellt. Der äquidistante Bus-Zyklus bildet somit den Grundtakt der Systemtakte. Der Basis-/Bus-Takt (DP-Takt oder PN-Takt) wird als Grundlage für die Einstellung weiterer Takte verwendet.
	Servo_fast	Hier können Sie (optional) den Servo_fast-Takt einstellen. Der Bezugstakt ist der in HW Konfig eingestellte isochrone PROFINET IO -Takt (des schnelleren Bussystems).
	IPO_fast	Hier können Sie den IPO_fast-Takt einstellen. Der Bezugstakt ist der Servo_fast-Takt.
	Servo / T1(DCC)	Hier können Sie ein ganzzahliges Verhältnis des Servo-Takts zum Bus-Takt/Servo_fast-Takt einstellen. In diesem Takt wird u. a. die Lageregelung der Achsen berechnet. In der Regel sollte der Faktor 1 verwendet werden. Wenn Sie den Faktor 2 einstellen, so wird die Dynamik der Regelung schlechter, jedoch steht mehr freie Rechenzeit zur Bearbeitung anderer Aufgaben zur Verfügung. Das Untersetzungsverhältnis zwischen dem Bus- Zyklus und Servo-Takt ist auch am Antrieb als "Masterapplikations-Zyklus" einzustellen. Diese Einstellung ist erforderlich für die gegenseitige Lebenszeichenüberwachung. Hinweise hierzu finden Sie in den Beschreibungen zu den Antrieben. Ist keine Äquidistanz parametriert, wird das Verhältnis Basiszeit zu Servo-Takt mit 1:1 vorgegeben. Es kann dann nicht geändert werden. PROFIBUS DP: Der Servo-Takt kann im Verhältnis 1:1/2/3/4/ betrieben werden. PROFINET IO: Der Servo-Takt kann im Verhältnis 1:1/2/3/4/6/8/10/12/14/16 betrieben werden.
	Ipo / T2(DCC)	Hier können Sie ein ganzzahliges Verhältnis des IPO-Takts zum Servo-Takt einstellen. Standardmäßig wird im Interpolator-Takt die Bewegungsführung der Achsen berechnet. Mit dem Faktor des Interpolator-Taktes bestimmen Sie, wie schnell die Sollwerte der Antriebe berechnet werden.
	Ipo2 / T3(DCC)	Hier können Sie ein ganzzahliges Verhältnis des IPO_2-Takts zum IPO-Takt einstellen. Der Interpolator-Takt 2 dient zur Bewegungsführung niederpriorer Achsen. Mit dem Faktor bestimmen Sie, wie schnell die Sollwerte der niederprioren Antriebe berechnet werden.
	DCCAux T4(DCC)	Hier können Sie ein ganzzahliges Verhältnis des DCC-Taktes DCCAux zum DCC-Takt T3(DCC) einstellen.

Feld/Schaltfläche		Bedeutung/Hinweis
	DCCAux_2 T5(DCC)	Hier können Sie ein ganzzahliges Verhältnis des DCC-Taktes DCCAux_2 zum DCCAux einstellen.
TControl		Klicken Sie auf den Pfeil, um die Konfiguration der Systemtasks für das Technologiepaket TControl zu öffnen. Dort können Sie das Verhältnis der Takte für die Task der Temperaturkanäle im Verhältnis zum Servo-Takt einstellen.
	Systemtasks für TControl verwenden	Aktivieren Sie die Kontrollkästchen, wenn die speziellen Tasks für die Temperaturkanäle im Ablaufsystem angezeigt werden sollen. Wenn Sie keinen Temperaturkanal projektiert haben, sollten Sie die Systemtasks für TControl deaktivieren, da die Systemtasks dann unnötige Rechenzeit beanspruchen.
	Servo-Takt (Masterapplikations- Zyklus)	Hier wird der Servotakt zur Information angezeigt. Um diesen zu ändern, müssen Sie weiter oben im Dialog den Wert auswählen. Alle anderen Takte des Temperaturkanals sind ein Vielfaches des Servo- Takts.
	PWM (Pulsweitenmodulation)	Takt, der für die Stellsignalausgabe des Temperaturkanals verwendet wird. Gibt den Grundtakt für die weiteren Takte vor. Wählen Sie den Takt in ms oder in Hz. Wenn Sie Hz wählen, werden die Verhältnisse zum Input-Takt und zum Control-Takt automatisch festgesetzt.
	Input1/2	Takt, der für die Istwerterfassung des Temperaturkanals verwendet wird. Wählen Sie das Verhältnis dieses Takts zum PWM-Takt. Im gegrauten Textfeld wird die Zeitdauer angezeigt.
	Control1/2	Takt, der für die Regelung des Temperaturkanals verwendet wird. Wählen Sie das Verhältnis dieses Takts zum Input-Takt. Im gegrauten Textfeld wird die Zeitdauer angezeigt.

Asynchrone zyklische I/O-Daten kopieren

Ab der V4.2 können Sie auswählen, wann die asynchronen zyklischen I/O-Daten kopiert werden sollen. Sie können jetzt auswählen, ob die Daten im IPO (mit DCC T2(DCC)) oder IPO_2 (mit DCC T3(DCC)) kopiert werden sollen.

- Wählen Sie dazu in der Klappliste **Nicht äquidistante E/A-Daten von PROFIBUS bzw. E/A-Daten von PROFINET IO mit RT kopieren in** den Takt aus.

Das Kopieren der Eingänge findet immer am Anfang der ersten Task einer Ebene statt. Das Kopieren der Ausgänge findet immer am Ende der letzten Task einer Ebene statt. Gibt es keine DCC- oder Anwenderprogramm-Task, ist die BackgroundTask zugleich erste und letzte Task der Ebene.

Hinweis

Kopiert werden alle nicht äquidistanten Daten am **PROFIBUS DP**, **PROFINET IO** und **P-Bus**.

6.3.4 Systemtakte den Technologieobjekten zuweisen

Systemtakte zuweisen

Nutzen Sie die Leistungsreserven der SIMOTION CPU besser, indem Sie Ihre Technologieobjekte priorisieren und gezielt den Systemtakten zuordnen. Durch die Definition von niederprioren Aufgaben schaffen Sie Leistungsreserven für höherpriore Aufgaben.

Insbesondere bei gleichmäßigen Bewegungen ohne große Beschleunigungen/Verzögerungen kann daher eine höhere Zeitebene für die TOs verwendet werden. (z. B. Aufruf des TOs im Interpolator-Takt 2).

Ändern Sie die Standard-Einstellung,

- wenn Sie eine zu lange Auftragsbearbeitung erkennen
- wenn Sie eine zu hohe Auslastung der Technologie in der CPU erkennen

Weisen Sie den Technologieobjekten Achse und Externen Geber mit niederprioren Aufgaben den InterpolatorTakt_2, den Technologieobjekten Nocken und Messtaster den IPO-Takt oder den InterpolatorTakt_2 zu.

Weisen Sie den Technologieobjekten mit hochprioren Aufgaben den InterpolatorTakt bzw. dem Servo-Takt zu.

Den Technologieobjekten können Sie folgende Takte zuweisen:

Motion Control Aufgabe	hochprior	...	niederprior
Technologieobjekt	Servo-Takt	Interpolator- Takt	Interpolator- Takt 2
Drehzahlachse	(x)	Standard	X
Positionierachse	(x)	Standard	X
Gleichlaufachse	(x)	Standard	X
Externer Geber	(x)	Standard	X
Nocken	X	X	X
Nockenspur	X	X	X
Messtaster	X	X	X

Die Systemtakte von Nocken und Messtastern, sowie für Achsen werden in SIMOTION SCOUT im Dialog **Konfiguration** eingestellt.

Ab V4.1 kann in Ausnahmefällen der IPO-Anteil auch im Servo gerechnet werden, siehe dazu **Bewegungsführung / Interpolator** im Handbuch **TO Achse**.

Siehe auch

Zweiter Servo-Takt (Servo_fast) (Seite 257)

6.3.5 Tasklaufzeiten

Mit Hilfe der Tasklaufzeiten können Sie kontrollieren, ob die Rechenleistung des Systems den Ansprüchen der Applikation genügt.

Die Tasklaufzeiten werden in den Geräte-Systemvariablen **Taskruntime** und **effectiveTaskruntime** angezeigt.

Es wird dabei unterschieden zwischen:

- den Laufzeiten der einzelnen Tasks innerhalb einer Ebene (**Taskruntime**)

  Die **Taskruntime** zeigt die Summe der netto-Laufzeiten der Task an (ohne die Unterbrechungszeiten).

- der Laufzeit der Ebene (**effectiveTaskruntime**)

  Die **effectiveTaskruntime** zeigt die effektive Laufzeit einer Ebene an (inklusive der Unterbrechungszeiten). Dies ist die Zeit vom Start der Ebene bis zum Ende der letzten Task in der Ebene.

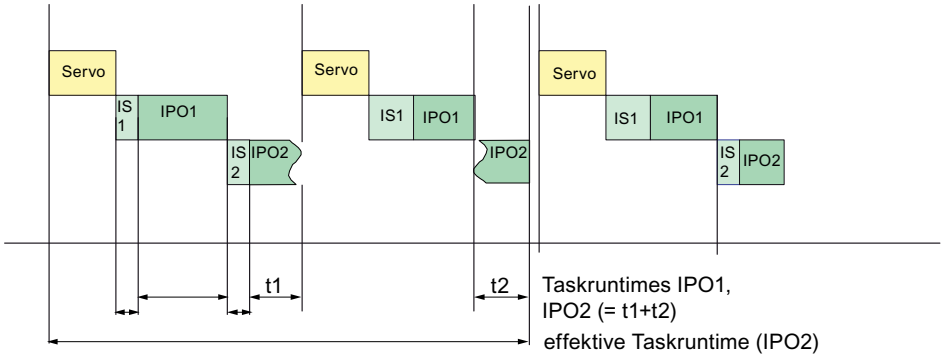


Bild 6-30 Darstellung der Taskruntime und effectiveTaskruntime (Taktverhältnis IPO1 : IPO2 = 2)

Legende:

Servo: Servo-Takt

IS1: IPOSynchronousTask

IS2: IPOSynchronousTask_2

IPO1: IPOTask

IPO2: IPOTask2

Folgende Tabelle zeigt die Tasks und Ebenen für die eine Taskruntime (Tasks) und eine effektive Taskruntime (Ebenen) bestimmt werden kann.

Tasks (Taskruntime)	Ebenen (effektive Taskruntime)
servodcc	Servo / T1(DCC)
servo	
servosynchronous	
servo_fast	Servo_fast
ipodcc	Ipo / T2(DCC)
ipo	

Tasks (Taskruntime)	Ebenen (effektive Taskruntime)
iposynchronous	
ipo_fast	IPO_fast
ipodcc_2	Ipo_2 / T3(DCC)
ipo_2	
Iposynchronous_2	
dccaux	DCCAux T4(DCC)
dccaux_2	DCCAux_2 T5(DCC)
background	Background

Hinweis

Zur Ermittlung der Tasklaufzeiten muss die Systemvariable **taskRuntimeMonitoring** auf **enable** gesetzt werden.

Die Tasklaufzeiten sind auch in der SCOUT Gerätediagnose sichtbar.

Siehe auch

Funktionen zur Laufzeitmessung von Tasks - Übersicht (Seite 344)

Ablaufsystem optimieren (Seite 566)

Effizient Programmieren - Übersicht (Seite 565)

6.3.6 Zeit- und Ebenenüberläufe

Bei der Abarbeitung von Rechenprozessen kann es zu Zeit- und/oder Ebenenüberläufen kommen. Um Tasklaufzeiten zu beobachten, können Sie die Gerätediagnose im SCOUT (nur ONLINE) benutzen. Die Auswertung von Systemvariablen ist unter **Überwachung von Zeit- und Ebenenüberläufen** beschrieben (siehe unten).

Sie haben die Möglichkeit, verschiedene Reaktionen für das Systemverhalten bei Überläufen bestimmter Ablaufebenen zu konfigurieren.

Zeitüberlauf

Für die Ablaufebenen **BackgroundTask**, **TimerInterruptTask**, **SynchronousTasks** und **ShutdownTask** können Sie maximale Ausführungszyklen konfigurieren.

Bei Überschreitung der eingestellten Maximaldauer kann die zugehörige **SystemInterruptTask** (**TimeFaultTask** bzw. **BackgroundFaultTask**) oder eine Standardfunktion (z. B. CPU in **STOP**) aufgerufen werden.

Ebenenüberlauf

Für die Ablaufebenen **IPO1** und **IPO_2** können Sie eine Tolerierung von Überläufen der Ebene einstellen.

Ein Ebenenüberlauf liegt dann vor, wenn eine Ebene (**IPOTask/IPOSynchronousTask** bzw. **IPOTask2/IPOSynchronousTask_2**) nicht innerhalb des eingestellten Systemtaktes IPO- bzw. IPO_2- Takt mit ihrer Bearbeitung fertig wird.

Hinweis

Eine zyklische Anwendertask (IpoSync, Ipo2Sync) muss innerhalb ihres Zyklusses fertig werden.

Ein Tasküberlauf durch einen fehlgeschlagenen Startversuch einer niederprioren Task erzeugt einen Eintrag in den Diagnosepuffer.

Sind die Takte so klein eingestellt, dass ständig Ebenenüberläufe auftreten, ist die CPU nicht mehr funktionsfähig.

Durch eine eingestellte Toleranz wird der nächste IPO-Takt verwendet, um:

- den Rechenprozess des vorhergehenden Taktes zu beenden, ohne dass die eingestellte Fehlerreaktion eintritt,
- den Rechenprozess des aktuellen Taktes zu starten.

Tritt ein Ebenüberlauf wiederholt im nächsten Ebenenzyklus auf (wenn keine Toleranz eingestellt oder die Anzahl überschritten ist), so wird die CPU in **STOP** gesetzt, um eine Blockade des Gesamtsystems zu verhindern. Es erfolgt ein Eintrag in den Diagnosepuffer und die Fehlerreaktion (CPU in **STOP**) mit Anlaufsperre.

Sie können bis zu maximal 5 zu tolerierende Ebenenüberläufe bei der Taskkonfiguration im Ablaufsystem einstellen.

Beispiele

Nachfolgend sind die Ablaufverhältnisse bei Ebenenüberläufen beispielhaft dargestellt:

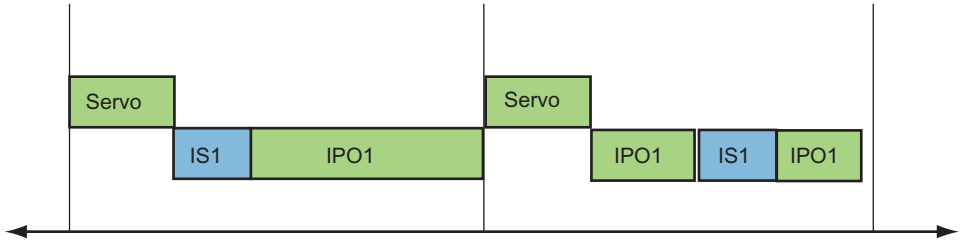
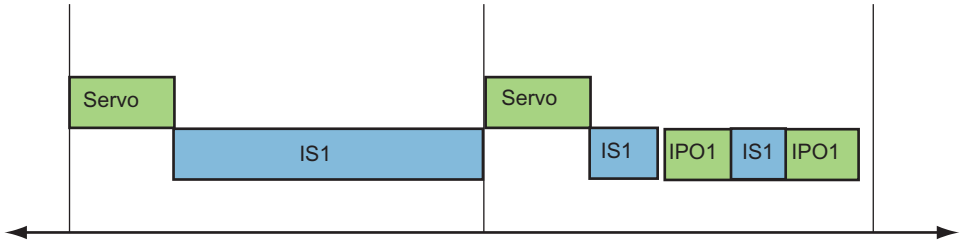


Bild 6-31 Beispiel IPO-Überlauf



Servo: Servo-Takt
IS1: IPOSynchronousTask
IPO1: IPOTask

Bild 6-32 Beispiel Überlauf der IPOSynchronousTask

Performantes Programmieren

Treten Ebenenüberläufe auf, aber ein Vergrößern des Bus/Servo-Takt ist nicht möglich, können folgende Maßnahmen durchgeführt werden:

- In der Taskkonfiguration bei IPOSynchronousTask
  Verhältnis IPOSynchronousTask : IPO-Takt=75%
  2 IPO-Überläufe tolerieren
- Optimierten PROFIBUS verwenden

  Benutzerdefiniertes Profil: HSA=2 (Höchste PROFIBUS-Adresse)
  bei z. B. 2 Teilnehmer am PROFIBUS, RetryLimit=1, zyklisches Verteilen der Busparameter abschalten (Damit ist allerdings kein PG am taktsynchronen PROFIBUS mehr anschließbar.)
- Verhältnis BackgroundTask : MotionTasks ändern

  Wenn z. B. der Schwerpunkt bei der MotionTask liegt, wird damit erreicht, dass z. B. bei einem Ereignis in der BackgroundTask eine MotionTask schnell gestartet und mit wenig Unterbrechungen behandelt wird.
- Systemvariablen nur einmal lesen und für weiteren Gebrauch in lokale Variable zwischenspeichern

Weitere Informationen zu performantem Programmieren, siehe Zugriff auf Ein- und Ausgänge optimieren (Seite 565) .

Überwachung von Zeit- und Ebenenüberläufen

Mit den Systemvariablen **Taskruntime** und **effectiveTaskruntime** kann festgestellt werden, ob ein Ebenenüberlauf oder ein Zeitüberlauf aufgetreten ist.

- Ist die **Taskruntime** im IPO/IPO_2-Takt *gleich* der **effectiveTaskruntime** im IPO/IPO_2-Takt, dann wurde die Task *nicht* durch eine höherpriore Task unterbrochen.
- Ist die **Taskruntime** im IPO/IPO_2-Takt *ungleich* der **effectiveTaskruntime** im IPO/IPO_2-Takt, dann wurde die Task durch eine höherpriore Task unterbrochen.

  Empfehlung: Geben Sie bei einem Übersetzungsverhältnis von Servo : IPO : IPO_2 > 1 bei der Taskkonfiguration der **SynchronousTasks** einen möglichst großen %-Wert für die Zeitdauer in Prozent vom IPO-Takt ein.
- Ist die **effectiveTaskruntime** der übergelaufenen Ebene *kleiner* als der eingestellte Takt der Ebene, so liegt ein *Zeitüberlauf* vor.

  Sie können diesen verhindern, indem Sie die Überwachungszeit der Ebene den Werten der Systemvariable **effectiveTaskruntime** anpassen.
- Ist die **effectiveTaskruntime** der übergelaufenen Ebene *sehr nah oder größer* als der eingestellte Systemtakt der Ebene, so liegt ein *Ebenenüberlauf* vor.

  Sie können diesen vermeiden, indem Sie die Systemtakte anpassen, oder Sie tolerieren diese Überläufe.

Siehe auch

SynchronousTasks (Seite 216)

Tasklaufzeiten (Seite 251)

6.3.7 Information zum Start einer Task: TaskStartInfo (TSI)

Beim Start jeder Task werden in der **TaskStartInfo** wichtige Informationen gespeichert, z. B.:

- der Startzeitpunkt der Task
- bei der **TechnologicalFaultTask** die auslösende Instanz des Technologieobjekts und die Alarmnummer
- bei der **TimeFaultTask**: die **TimerInterruptTask**, die den Zeitüberlauffehler verursachte

Innerhalb einer Task können Sie die jeweilige **TaskStartInfo** dieser Task abfragen.

Siehe auch

Taskstartinfo verwenden (Seite 153)

6.3.8 Zeitüberwachung

Die maximale Dauer für die Ausführung einer Task (Zykluszeit) kann konfiguriert werden. Bei Überschreiten dieser Zeit kann die zugehörige SystemInterruptTask (**TimeFaultTask** bzw. **TimeFaultBackgroundTask**) oder die Reaktion CPU in **STOP** aufgerufen werden.

Die Zykluszeit-Überwachung kann für folgende Tasks aktiviert werden:

- BackgroundTask
- TimerInterruptTasks
- SynchronousTasks
- ShutdownTask

Zeitüberwachung konfigurieren

- Wechseln Sie im Konfigurationsfenster der Task in das Register **Taskkonfiguration**.

Bei BackgroundTask, TimerInterruptTask, ShutdownTask:

- Geben Sie die maximale Ausführungsdauer in das Feld **Zeitüberwachung** ein (0 ms = keine Überwachung).

Bei IPOSynchronousTask:

- Wählen Sie das Verhältnis der maximalen Ausführungsdauer zum IPO-Takt.

Nicht bei ShutdownTask:

- Wählen Sie als Fehlerreaktion bei Zeitüberschreitung die zugehörige **SystemInterruptTask** oder die Reaktion CPU in **STOP**.

Hinweis

Bei zeitgesteuerten Tasks: Wird die Task erneut gestartet, bevor sie abgearbeitet ist, erfolgt ein Zeitüberlauffehler. Die SystemInterruptTask wird gestartet oder die CPU geht in **STOP**.

Siehe auch

SystemInterruptTasks (Seite 224)

6.4 Zweiter Servo-Takt (Servo_fast)

6.4.1 Servo_fast (Applikationstakt für schnelles Bussystem)

Beschreibung

Im Servo-Takt erfolgt die Synchronisation (Daten kopieren, zyklisches Lebenszeichen, Zeitstempel für Nocken und Messtaster) zur gesamten synchronen I/O-Peripherie (Antrieb, externer Geber, Messtastereingang, Nockenausgang, digitale und analoge Ein-/Ausgänge). Von diesem Takt werden weitere synchrone Takte abgeleitet (IPO, IPO_2, DccAux, DccAux_2).

Bei nur einem Servo steht für alle Bussysteme nur ein gemeinsamer Applikationstakt zur Verfügung. Dieser Applikationstakt kann nicht kleiner sein als der Takt des langsamsten Bussystems (Beispiel: PROFIBUS mit 1 ms). Das schnellere Bussystem (z. B. PROFINET IO mit 250 µs Sendetakt) kann zwar die Daten mit einem schnelleren Takt übertragen, die Auswertung der Signale erfolgt aber trotzdem nur im langsamen Applikationstakt.

Zweiter Servo-Takt (Servo_fast) für das schnelle Bussystem

Ab V4.2 können Sie über den Servo_fast synchrone Geräte (Antrieb, Messeingang oder Nockenausgang, I/Os) am schnellen Bussystem (PROFINET) betreiben. Der Servo_fast wird dabei immer im ersten Sendetakt des schnellen Bussystems beendet. Damit können zwei Bussysteme in unterschiedlichen Applikationstakten betrieben werden. Für beide Applikationstakte steht je ein zugeordneter Servo- und IPO-Takt zur Verfügung.

Wie Sie einen zweiten Servo-Takt projektieren, finden Sie im Funktionshandbuch *Kommunikation* unter Servo_fast, Taktuntersetzung zum Servo an der PROFINET-Schnittstelle beschrieben.

Ein zweiter Servo-Takt ist nur in Anwendungen mit besonderen Anforderungen erforderlich, z. B.:

- für eine schnelle I/O-Verarbeitung über PROFINET IO (für besonders kurze Abtast- und Reaktionszeiten),
- wenn neben den elektrischen Achsen (z. B. Verstellantriebe) auch Hydraulikachsen mit einer besonders performanten Druck-/Lageregelung realisiert werden,
- wenn eine Aufteilung der elektrischen Achsen in 2 Performance-Klassen erforderlich ist (schneller Servo - langsamer Servo)

Hinweis

Für V4.2 unterstützen nur die SIMOTION Baugruppen SIMOTION D445-2 DP/PN und SIMOTION D455-2 DP/PN den zweiten Servo-Takt.

Sollen die Achsen in 2 Performance-Klassen aufgeteilt werden, reicht in der Regel eine Aufteilung auf IPO-Ebene (IPO, IPO_2) aus, eine Aufteilung auf Servo-Ebene (Servo, Servo_fast) ist nur in Ausnahmefällen erforderlich.

- Im IPO erfolgt die Führungsgrößenberechnung/Bewegungsprofilberechnung der Achsen.
- Im Servo erfolgt die Lageregelung und Überwachungen der Achsen.

Berücksichtigen Sie auch, dass ein zusätzlicher Servo_fast/IPO_fast das SIMOTION Laufzeitsystem in Summe belastet.

Zudem wird mit DSC (Dynamic Servo Control) bei Servo-Achsen der dynamisch wirksame Teil des Lagereglers im Antrieb in der Frequenz des Drehzahlregelkreises (d. h. bei SINAMICS S120 üblicher Weise mit 125 µs) ausgeführt. Wird symbolische Zuordnung mit automatischer Telegrammermittlung verwendet, ist DSC automatisch aktiviert.

Servo_fast und IPO_fast

Die zusätzlichen schnellen Applikationstakte sind damit zusätzlich bei den TO auswählbar:

Erweiterung um Servo_fast und IPO_fast	Erweiterung um IPO_fast
TO Achse	TO Addierobjekt
TO Externer Geber	TO Bahnobjekt
TO Gleichlauf	TO Festes Getriebe
TO Messtaster	TO Formelobjekt
TO Nocken	TO Sensor
TO Nockenspur	TO Reglerobjekt

Siehe auch

Taktmodell mit einem Servotakt (Seite 259)

6.4.2 Taktmodell mit einem Servotakt

Beschreibung

Die Verwendung des Servo_fast ist optional und nur für bestimmte Anwendungen vorgesehen. Siehe auch Servo_fast (Applikationstakt für schnelles Bussystem) (Seite 257). Für Standardanwendungen genügt ein Applikationstakt.

Die folgende Grafik zeigt den Ablauf mit einem Servo-Takt.

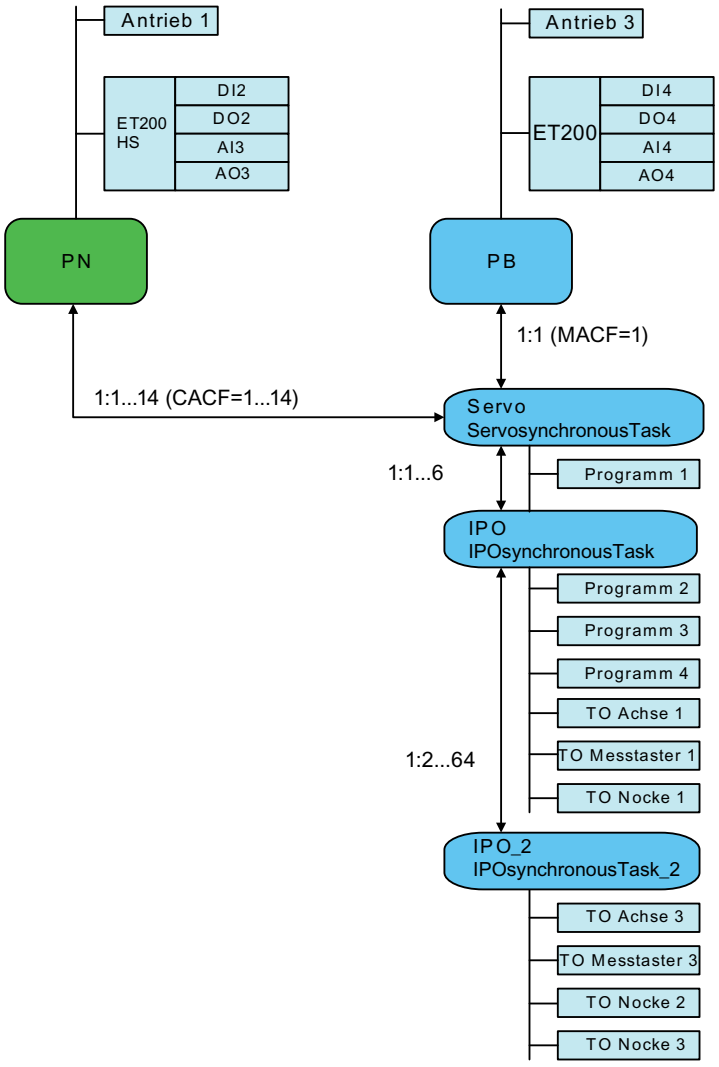


Bild 6-33 I/O-Peripherie mit einem Servo-Takt

- MACF: Der Master Application Cycle Factor beschreibt das Untersetzungsverhältnis des PROFIBUS DP Sendetaktes zum Takt einer übergeordneten Steuerung.
- CACF: Der Contoller Application Cycle Factor beschreibt das Untersetzungsverhältnis des PROFINET IO Sendetaktes zum Takt einer übergeordneten Steuerung.

6.4.3 Taktmodell mit zwei Servo-Takten

Beschreibung

Mit dem zweiten Servo-Takt können Sie zwei Bussysteme in unterschiedlichen Applikationstakten betreiben. Für beide Applikationstakte steht je ein zugeordneter Servo- und IPO-Takt zur Verfügung. Damit können Sie Ihre Applikation in einen langsamen und einen schnellen Teil (Servo_fast und IPO_fast) aufteilen.

Das Grundprinzip ist dabei:

- Die TOs und I/Os am schnellen Bussystem werden im schnellen Servo_fast/IPO_fast taktsynchron bearbeitet.
- Die TOs und I/Os am langsamen Bussystem werden im langsamen Servo/IPO/IPO_2 taktsynchron bearbeitet.

Ausnahme:

- Die TO Nocken- und Messtaster im langsamen Servo/IPO/IPO_2 können auch die Peripherie vom schnellen Bussystem nutzen.

Die folgende Grafik zeigt die Nutzung von zwei Servo-Takten.

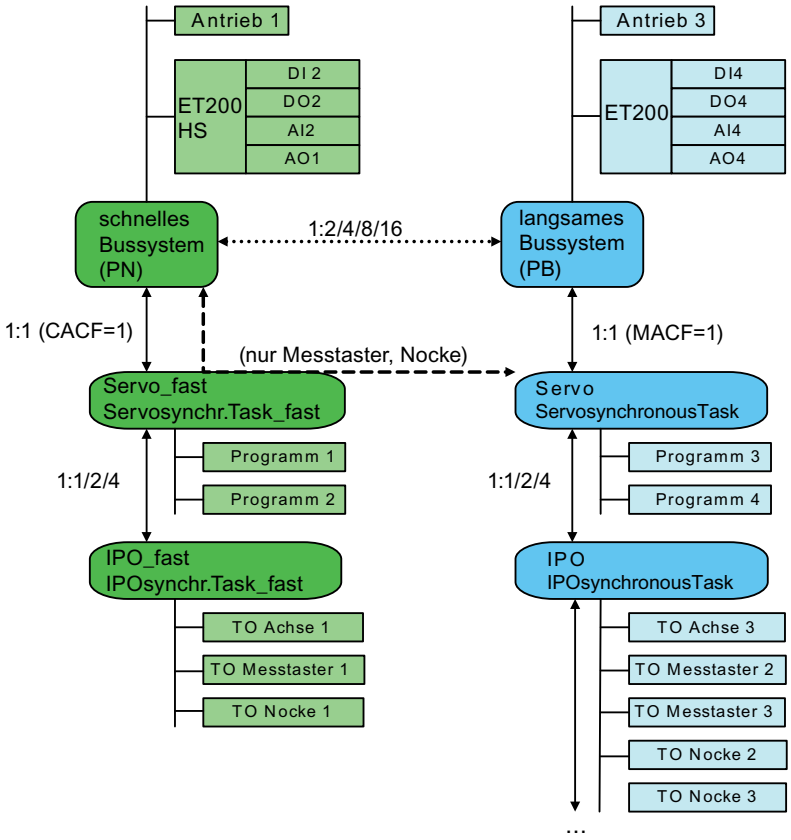


Bild 6-34 I/O-Peripherie mit zwei Servo-Takten.

Für Information zu CACF und MACF, siehe auch Taktmodell mit einem Servotakt (Seite 259).

6.4.4 Taktverhältnisse und Bedingungen bei zwei Servo-Takten

Beschreibung

Für die taktsynchrone Nutzung wird in HW Konfig an der I/O-Baugruppe der Applikationszyklus eingestellt:

- Bei PROFIBUS DP-Slaves wird der MACF (Master-Applikationszyklus) eingestellt. Wenn Sie zusätzlich PROFINET IO nutzen, wird nur MACF=1 unterstützt. Alle PROFIBUS Geräte laufen im langsamen Servo-Takt.
- Bei PROFINET-IO-Devices wird der CACF eingestellt. Ab V4.2 können Sie an IO-Devices zusätzlich den Servo-Takt auswählen (Servo_fast)

Folgende Einstellungen sind möglich:

Produktstufe	Eigenschaft	Applikationszyklus der Geräte am	
		PROFINET IO	PROFIBUS DP
Bis V4.2	1 Servo-Takt	Servo	Servo
V4.2	2 Servo-Takte	Servo_fast	Servo

Applikationszyklus am Peripheriegerät

Der in HW Konfig am Peripheriegerät eingestellte Applikationszyklus hat folgende Auswirkungen auf die taktsynchrone Nutzung der TO´s und Anwenderprogramme:

- Standard-I/O werden nur im jeweiligen Servo exakt taktsynchron genutzt. Ein "IPO-synchrones" Prozessabbild der Ausgänge kann im 1., 2., 3., ... Servotakt innerhalb eines IPO (bei Taktuntersetzung Servo zu IPO) ausgegeben werden (abhängig von der Servo- + IPO-Laufzeit). Der Abstand zwischen 2 Ausgaben kann damit schwanken.
- Antriebs- und Geber-Telegramme werden im jeweiligen Servo zwischen Controller und Device synchronisiert. Im zugehörigen IPO können sie vom TO Achse taktsynchron genutzt werden.
- Messeingänge und Nockenausgänge mit Zeitstempel können auch in einem langsameren Takt synchron genutzt werden.

Eingestellter Applikationstakt	Takte für taktsynchrone Nutzung von		
	Standard I/O	Antrieb, Externer Geber	Messeingang, Nockenausgang
Servo_fast	Servo_fast	Servo_fast IPO_fast	Servo_fast IPO_fast Servo IPO IPO_2
Servo	Servo	Servo IPO IPO_2	Servo IPO IPO_2

Faktoren und Bezugstakte bei zwei Servo-Takten

Die nachfolgende Tabelle zeigt die möglichen Faktoren und Bezugstakte bei 2 Servos:

Takt-Name	einstellbare Faktoren	Bezugstakt
Servo_fast	1	Datenzyklus vom schnellen isochronen PROFINET IO
IPO_fast	1, 2, 4	Servo_fast
Servo	1	Datenzyklus vom äquidistanten PROFIBUS DP
IPO	1, 2, 4	Servo
IPO_2	2, 3, 4, 5, …, 64	IPO
DccAux	2, 3, 4, 5, …, 32	IPO_2
DccAux_2	2, 3, 4, 5, …, 32	DccAux

Mögliche Takteinstellungen für äquidistante Takte

Bei Verwendung von 2 Servo-Takten sind zudem nachfolgende Bedingungen einzuhalten:

- Bustakt PROFIBUS DP = N x Sendetakt PROFINET IO (N = 2, 4, 8, 16)
- IPO ≥ IPO_fast
- Servo_fast und IPO_fast sind immer der PROFINET IO-Schnittstelle zugeordnet
- die PROFINET IO-Schnittstelle kann nur als Sync-Master betrieben werden (damit ist z. B. kein verteilter Gleichlauf über die PROFINET-Schnittstelle möglich)

Wird die PROFINET IO-Schnittstelle als Sync-Slave projektiert, wird die fehlerhafte Projektierung über einen Diagnosepuffereintrag gemeldet und die Steuerung geht in Anlaufsperre.

6.4.5 Prioritäten und Reihenfolgen der synchronen Tasks

6.4.5.1 Taktaufteilung bei zwei Servo-Takten

Beschreibung

Für die Taskprioritäten gelten folgende Regeln:

- Taskebenen mit einer kürzeren Zykluszeit haben eine höhere Priorität als Taskebenen mit einer längeren Zykluszeit.
- Bei gleicher Zykluszeit hat die Servo-Ebene eine höhere Priorität als die IPO-Ebene.
- Bei gleicher Zykluszeit hat die schnelle IPO-Ebene (IPO_fast) eine höhere Priorität als die langsame IPO-Ebene (IPO).

Die Reihenfolge der synchronen Tasks ergibt sich abhängig von den Takteinstellungen in zwei Varianten. Die folgende Grafik zeigt die Varianten für die Priorität und Reihenfolge der Taskebenen.

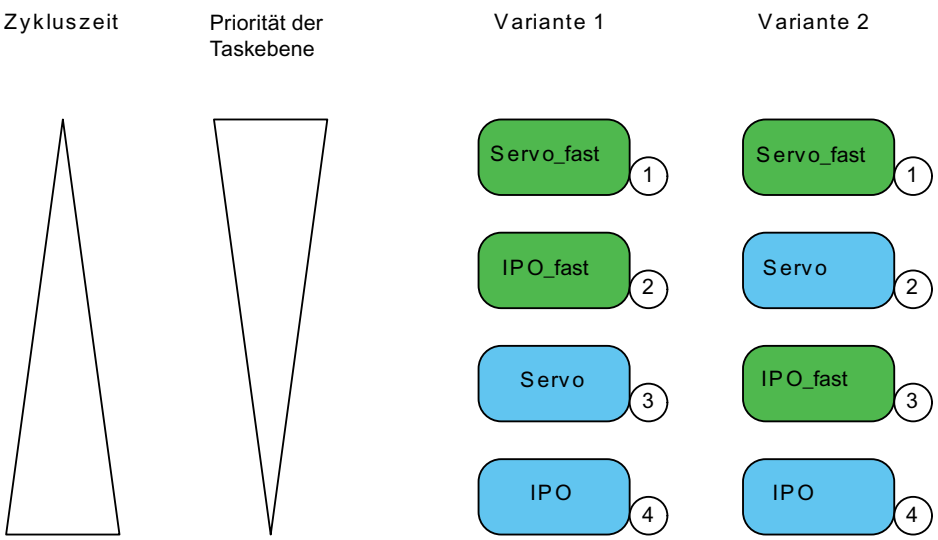


Bild 6-35 Taskaufteilung bei zwei Servo-Takten

Regeln für die Aufteilung der Takte

- Mit der Einstellung einer Taktuntersetzung zwischen PROFINET IO und PROFIBUS DP sowie zwischen Servo_fast und IPO_fast wählt der Anwender zwischen Variante 1 (Servo_fast – IPO_fast – Servo – IPO) und Variante 2 (Servo_fast – Servo – IPO_fast – IPO).
- Hat IPO_fast eine kürzere Laufzeit als der Servo werden intern im Ablaufebenensystem die Prioritäten für Variante 1 eingestellt.
- Eine sehr schnelle **Klemme-Klemme-Reaktion** erfordert sehr kurze Takte für Servo_fast. Ist damit zu rechnen, dass IPO_fast bei hoher Systembelastung nicht rechtzeitig vor dem Servo fertig wird, muss der IPO_fast gleich schnell oder langsamer als Servo gewählt werden. Damit werden intern im Ablaufebenensystem die Prioritäten für Variante 2 eingestellt.
- Der Takt IPO_fast ist nicht länger als der Takt IPO.
- Die neuen Takte Servo_fast und IPO_fast dürfen nicht überlaufen
- Bei einem Ebenenüberlauf geht das System in den Betriebszustand STOP. Die Anlaufsperre wird gesetzt und es erfolgt ein entsprechender Eintrag in den Diagnosepuffer. Erst nach einem Hochlauf (Netz aus/ein) oder Download kann das System wieder in den Betriebszustand RUN gehen.

6.4.5.2 Taktaufteilung bei 2. Servo Variante 1

Taskzuordnung Variante 1 (Servo_fast – IPO_fast – Servo – IPO)

Die folgende Grafik zeigt die Variante 1 der Taktzuordnung.

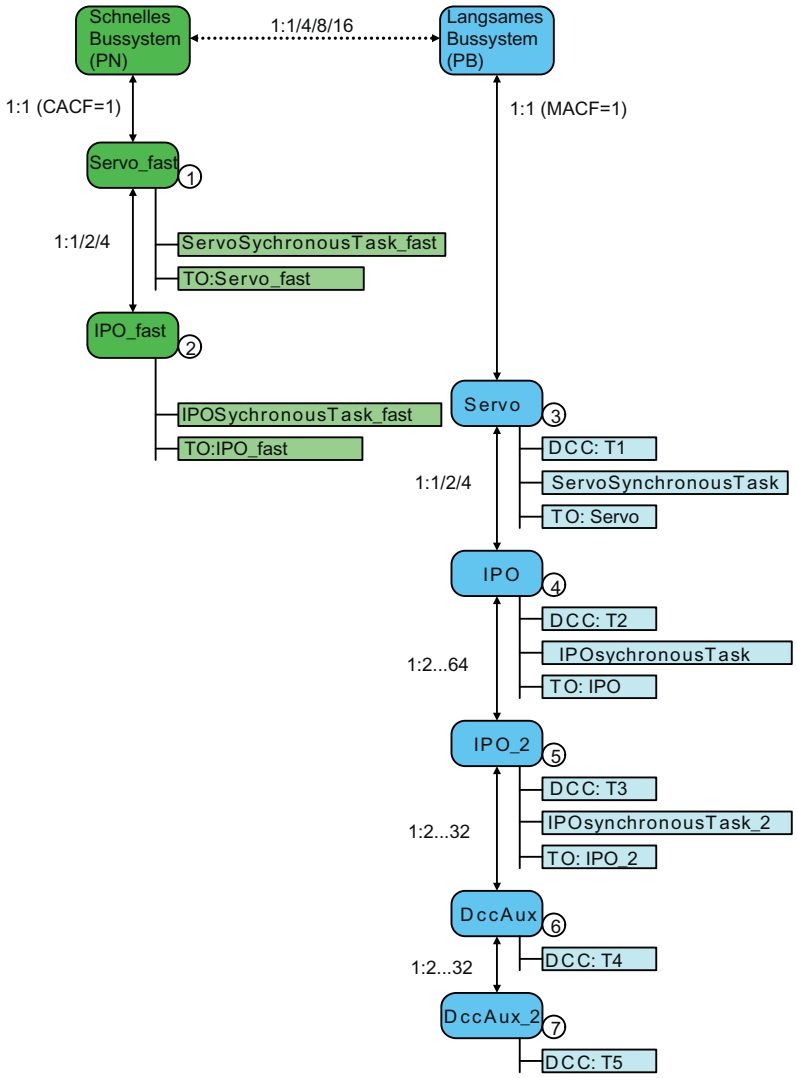


Bild 6-36 Taktaufteilung Variante 1

6.4.5.3 Taktaufteilung bei 2. Servo Variante 2

Taskzuordnung Variante 2 (Servo_fast– Servo– IPO_fast – IPO)

Die folgende Grafik zeigt die Variante 2 der Taktzuordnung.

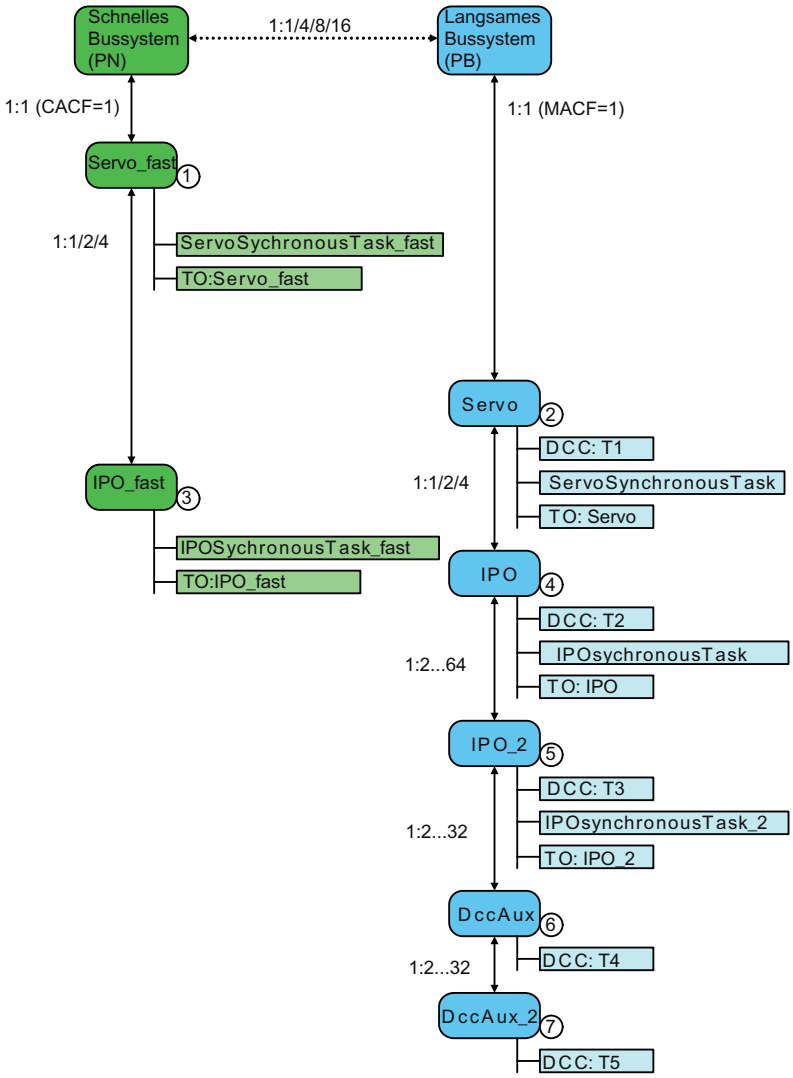


Bild 6-37 Taktzuordnung Variante 2

6.4.6 Synchronisation von Takt- und Bussystemen

Beschreibung

Die Synchronisation zwischen dem langsameren Bussystem und dem schnelleren erfolgt automatisch, wobei der Takt des Servo_fast immer ein ganzzahliges Vielfaches des Servo beträgt.

Beispiel für Takt- und Bussynchronisation

Das Beispiel zeigt das Verhalten der einzelnen Takte im Verlauf eines Zyklus.

- Zyklus schneller Servo: 250 µs
- Zyklus schneller IPO: 500 µs
- Zyklus langsamer Servo: 1 ms
- Zyklus langsamer IPO: 2 ms

Takt	0	1	2	3	4	5	6	7	8	9
Zeitpunkt	0000	0250	0500	0750	1000	1250	1500	1750	2000	2250
Start schneller Servo_fast	x	x	x	x	x	x	x	x	x	x
Start schneller IPO_fast	x		x		x		x		x	
Start langsamer Servo	x				x				x	
Start langsamer IPO	x								x	

6.5 Zeitaufteilung in der Round-Robin-Ablaufebene

Die nach Abarbeiten der hochprioren Anwender- und SystemTasks *verbleibende* Zeit wird von den **MotionTasks** und der **BackgroundTask** genutzt.

Die Zuteilung dieser Zeit zu der BackgroundTask und den MotionTasks erfolgt im *Round-Robin-Verfahren*.

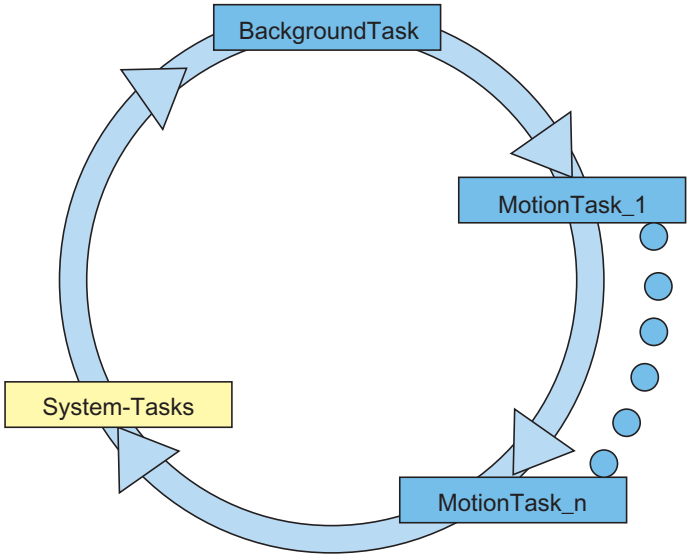


Bild 6-38 Übersicht der Zeitaufteilung in der Round-Robin-Ablaufebene

Round-Robin-Verfahren

Die nicht durch höherpriore Tasks verbrauchte Rechenzeit wird im Round-Robin-Verfahren auf die übrigen Tasks verteilt (Background Task, MotionTasks und Systemtasks); d.h. diese Tasks laufen reihum. Dadurch findet eine quasiparallele (quasi-"gleichzeitige") Bearbeitung dieser Tasks statt.

Die nächste Task wird jeweils gestartet, wenn die vorherige die Rechenzeit abgibt (Task zu Ende oder im Zustand "warten"). Die Rechenzeit einer Round-Robin-Task wird außerdem auf eine maximale Anzahl von Servotakten begrenzt. Im nächsten Round-Robin-Umlauf wird die Rechenzeit an der Unterbrechungsstelle fortgesetzt. Es wird sichergestellt, dass eine Round-Robin-Task jeweils mindestens die Restrechenzeit eines Servotaktes bekommt. Die BackgroundTaks kann bevorzugt werden, so dass ihr die Rechenzeit mehrerer aufeinanderfolgender Servotakte zur Verfügung steht (Schieberegler Ablaufsystem).

Weitere Tasks

In der Round-Robin-Ablaufebene laufen neben den Anwenderprogramm-Tasks (BackgroundTask, MotionTask) auch System-Tasks (z.B. für Kommunikation), die ebenfalls Rechenzeit im Round-Robin-Zyklus belegen.

Reihenfolge

Prinzipiell laufen die Tasks nacheinander im Round-Robin-Zyklus ab. (Die Reihenfolge ist *nicht deterministisch* und kann sich z.B. durch einen Download, Power-ON durchaus jedes mal ändern.)

Zeitaufteilung

Eine Task kann eine festgelegte Anzahl von Servo-Takten (Restlaufzeit) hintereinander laufen, bevor sie die Rechenzeit an die nächste Task abgeben muss. Sie kann aber auch schon vorher an die nächste Task abgeben, wenn sie "nichts zu tun hat".

Für den konstruierten Fall, dass alle Tasks in der Round-Robin-Ablaufebene schon einmal behandelt wurden, wird die erste Task wieder gestartet. Somit gibt es *keine* IDLE-Zeit (Leerlaufzeit).

Es gibt Zeiten in denen kein Anwenderprogramm gerechnet wird. Die ist z. B. der Fall, wenn BackgroundTask und MotionTask sehr kurz sind (kleiner der Restlaufzeit in einem Servo-Takt), dann wird die restliche Laufzeit von den System-Tasks genutzt.

Neustart/Prozessabbild

Die BackgroundTask wird nach ihrem Ende erst wieder gestartet, wenn das Prozessabbild aktualisiert wurde. Die Aktualisierung erfolgt in der **ServoTask**.

Performance

Eine Endlosschleife in einer MotionTask, ohne Wartebefehle oder synchrone Bewegungsbefehle bewirken, dass die MotionTask ihre maximale Rechenzeit belegt. Folglich wird der Zyklus einer BackgroundTask zusätzlich belastet (eine größerer Anteil der Rechenzeit fließt in die MotionTask). Auch die System-Task in der Round-Robin-Ebene werden dadurch in größeren Zeitabständen gerufen, wodurch z.B. die Kommunikation beeinflusst kann.

Hinweis

MotionTasks mit (Endlos-)Schleifen ohne _waitTime (0s) belasten die Round-Robin-Ablaufebene, da hier die MotionTask zwei volle Servo-Takte verwendet.

Bei der Aufteilung der Round-Robin-Ablaufebene für BackgroundTask und Motion-Tasks, erhält die BackgroundTask in einem Round-Robin-Gesamtzyklus mindestens 1 Zeitscheibe, eine MotionTask 2 Zeitscheiben (falls diese, z.B. wegen einer Endlosschleife benötigt wird).

Zyklische MotionTasks

Hinweis wenn Motion Tasks zyklisch laufen sollen:

Wenn Sie Endlosschleifen in MotionTasks verwenden wollen, dann setzen Sie in jedem Zyklus z.B. ein **_waitTime(0s)** ab. Dann gibt diese MotionTask an die folgende MotionTask ab.

Hinweis

MotionTasks mit (Endlos-)Schleifen ohne _waitTime (0s) belasten die Round-Robin-Ablaufebene, da hier die MotionTask zwei volle Servo-Takte benötigt.

Wenn MotionTasks zyklisch durchlaufen werden sollen, ist es besser, statt einer Endlosschleife oder Sprunganweisung zum Programmanfang, die Task normal zu beenden und die MotionTask aus einer anderen Task (z. B. BackgroundTask) erneut zu starten. Dies hat auch Vorteile für einen Download im **RUN**.

Siehe auch

Ablaufsystem optimieren (Seite 566)

6.5.1 Einstellung der Zeitaufteilung

Sie können über einen Schieberegler, der über die Taste **Zeitaufteilung** bei **MotionTasks** oder bei der **BackgroundTask** zugänglich ist, die zeitliche Wichtung der BackgroundTask zu den restlichen Tasks der Round-Robin-Ablaufebene einstellen.

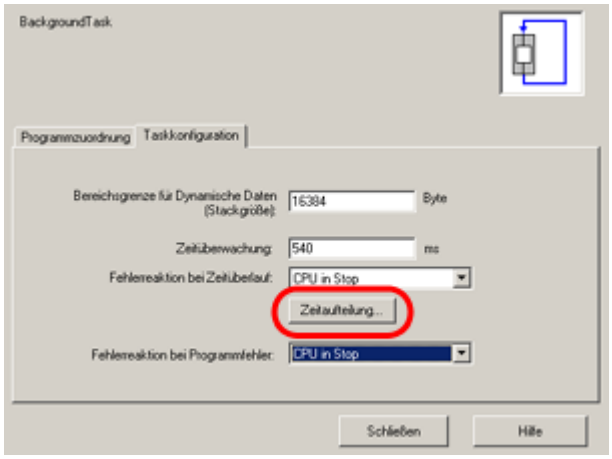


Bild 6-39 Taskkonfiguration im SCOUT

Zeitaufteilung in der Round-Robin-Ablaufebene festlegen

1. Wählen Sie im Konfigurationsfenster einer **MotionTask** oder der **BackgroundTask** das Register **Taskkonfiguration**.
2. Klicken Sie auf **Zeitaufteilung**.

   Das Fenster Zeitaufteilung in der Round-Robin-Ablaufebene öffnet sich.
3. Stellen Sie mit dem Schieberegler das Verhältnis der Rechenzeiten ein:
4. Bestätigen Sie mit **OK**.

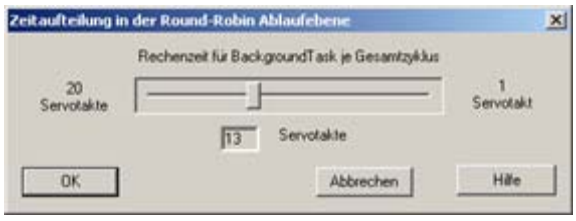


Bild 6-40 Zeitaufteilung in der Round-Robin Ablaufebene

Zeitaufteilung in der Round-Robin-Ablaufebene

Hier legen Sie das Verhältnis der Zeitaufteilung zwischen der BackgroundTask und allen anderen Round Robin Tasks fest (MotionTasks und Systemtasks, siehe Ablaufebenen/Tasks (Seite 191)).

Die BackgroundTask bekommt (maximal) n-mal die Rechenzeit einer anderen Round Robin Task (alle anderen Round Robin Tasks, z.B. MotionTask_1, haben jeweils den gleichen maximalen Rechenzeitanteil). Diesen Faktor können Sie mit dem Schieberegler einstellen.

Wenn die BackgroundTask zu Ende gelaufen ist, wird sie frühestens wieder gestartet, wenn die Eingänge aktualisiert werden (nach dem nächsten Servodurchlauf).

Wird die Rechenzeit der BackgroundTask hoch eingestellt und dadurch die anderen Round Robin Tasks benachteiligt, so wird dadurch z.B. die Kommunikation zu SCOUT/OP langsamer. Eine solche Einstellung sollte also nur bei sehr zeitkritischen Anwendungen gemacht werden.

Tabelle 6- 5 Auswirkungen der Rechenzeitverteilung zwischen MotionTasks und BackgroundTask

Hohe Rechenzeit für	Auswirkung
MotionTasks	Zeit von Anfang bis Ende der BackgroundTask wird länger; im Extremfall spricht Zeitüberwachung an.
BackgroundTask	Programme in der MotionTask brauchen unter Umständen länger, bis sie bearbeitet werden.
Die Rechenzeit für die BackgroundTask enthält auch die für die azyklische Kommunikation (über PROFIBUS bzw. PROFINET IO mit IRT) benötigte Zeit.	

6.5.2 Einstellungen (Beispiele)

Zur Erläuterung der Zeitaufteilung in der Round-Robin-Ablaufebene finden Sie hier zwei Beispiele zur Einstellung des Schiebereglers.

Fall 1: Schieberegler ganz rechts

Die BackgroundTask läuft für minimal einen Servo-Takt.

Bild 6-41 Zeitaufteilung in der Round-Robin: Einstellung Rechenzeit für MotionTasks

In dieser Einstellung läuft die BackgroundTask für einen Servo-Takt, anschließend laufen alle MotionTasks (jede MotionTask läuft für maximal zwei Servo-Takte, falls sie vorher nicht in den Suspend-Mode geht), dann wieder einen Servo-Takt lang die BackgroundTask, usw.

Tabelle 6- 6 Beispiel:

Servo-Takt 1	Servo-Takt 2-3	Servo-Takt 4-5	Servo-Takt 6	Servo-Takt 7-8
Background Task	MotionTask 1	MotionTask 2	Background Task	MotionTask 1

Programmbeispiel:

Gleichzeitig mit der BackgroundTask laufen MotionTask 1 und 2;
Servo-Takt = 3 ms

- In der BackgroundTask wird ein Zähler in einer While-Schleife inkrementiert (grüne Linie).
- In der MotionTask 1 wird ein Zähler in einer While-Schleife inkrementiert (rote Linie).
- In der MotionTask 2 wird ein Zähler in einer While-Schleife inkrementiert (blaue Linie).

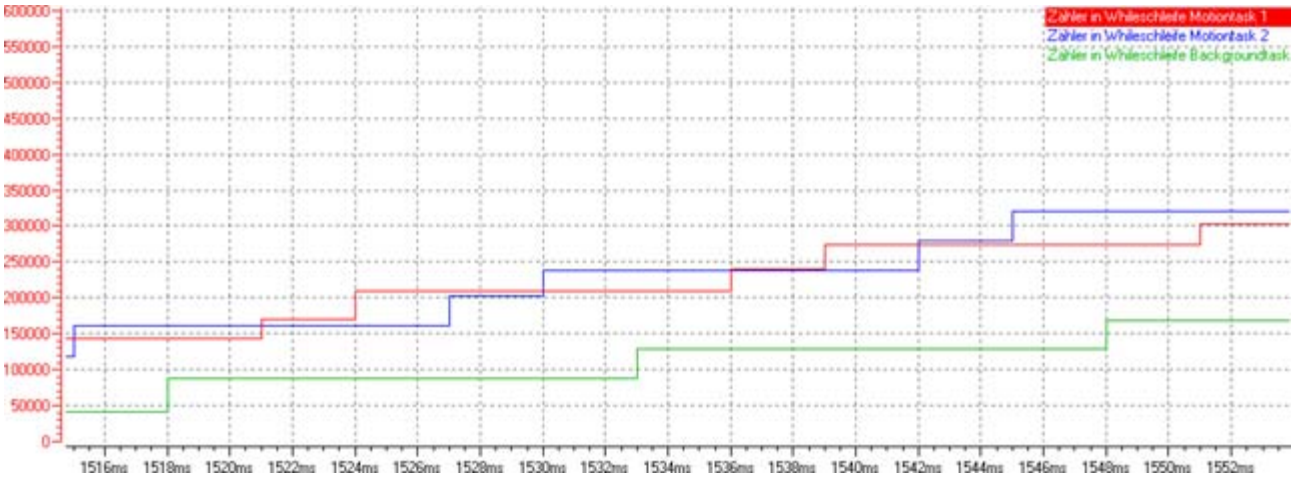


Bild 6-42 Ablaufbeispiel für Zeitaufteilung in der Round-Robin: Einstellung Rechenzeit für MotionTasks

t = 1518 ms	BackgroundTask läuft für einen Servo-Takt
t = 1521ms, 1524 ms	MotionTask 1 läuft für zwei Servo-Takte
t = 1527ms, 1530 ms	MotionTask 2 läuft für zwei Servo-Takte
t = 1533 ms	BackgroundTask läuft für einen Servo-Takt

Fall 2: Schieberegler ganz links

Die BackgroundTask läuft für 20 Servo-Takte.

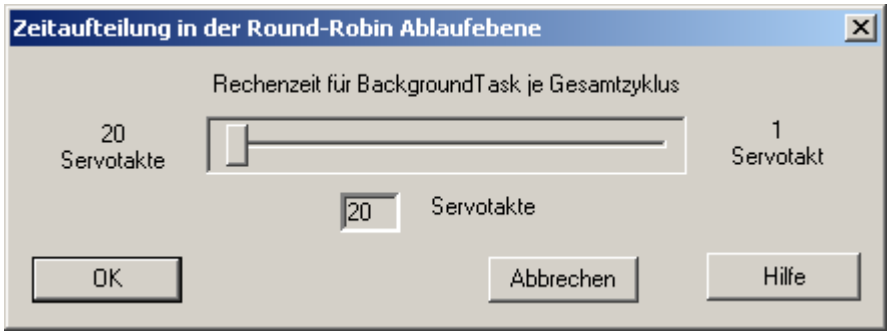


Bild 6-43 Zeitaufteilung in der Round-Robin: Einstellung Rechenzeit für BackgroundTask

In dieser Einstellung läuft die Background-Task für 20 Servo-Takte, anschließend laufen alle MotionTasks (jede MotionTask für maximal zwei Servo-Takte, falls sie vorher nicht in den Suspend-Mode geht), dann wieder 20 Servo-Takte lang die Background Task, usw.

Tabelle 6- 7 Beispiel:

Servo-Takt 1-20	Servo-Takt 21-22	Servo-Takt 23-24	Servo-Takt 25-40	Servo-Takt 41-42
Background Task	MotionTask 1	MotionTask 2	Background Task	MotionTask 1

Programmbeispiel:

Gleichzeitig mit der BackgroundTask laufen MotionTask 1 und 2;
Servo-Takt = 3 ms

- In der BackgroundTask wird ein Zähler in einer While-Schleife inkrementiert (grüne Linie).
- In der MotionTask 1 wird ein Zähler in einer While-Schleife inkrementiert (rote Linie).
- In der MotionTask 2 wird ein Zähler in einer While-Schleife inkrementiert (blaue Linie).

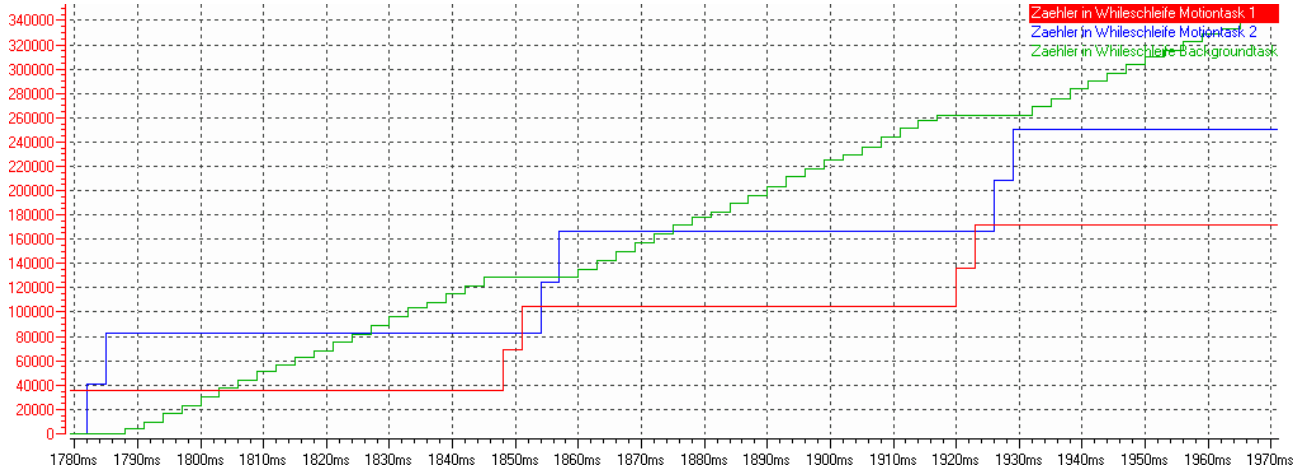


Bild 6-44 Ablaufbeispiel für Zeitaufteilung in der Round-Robin: Einstellung Rechenzeit für BackgroundTask

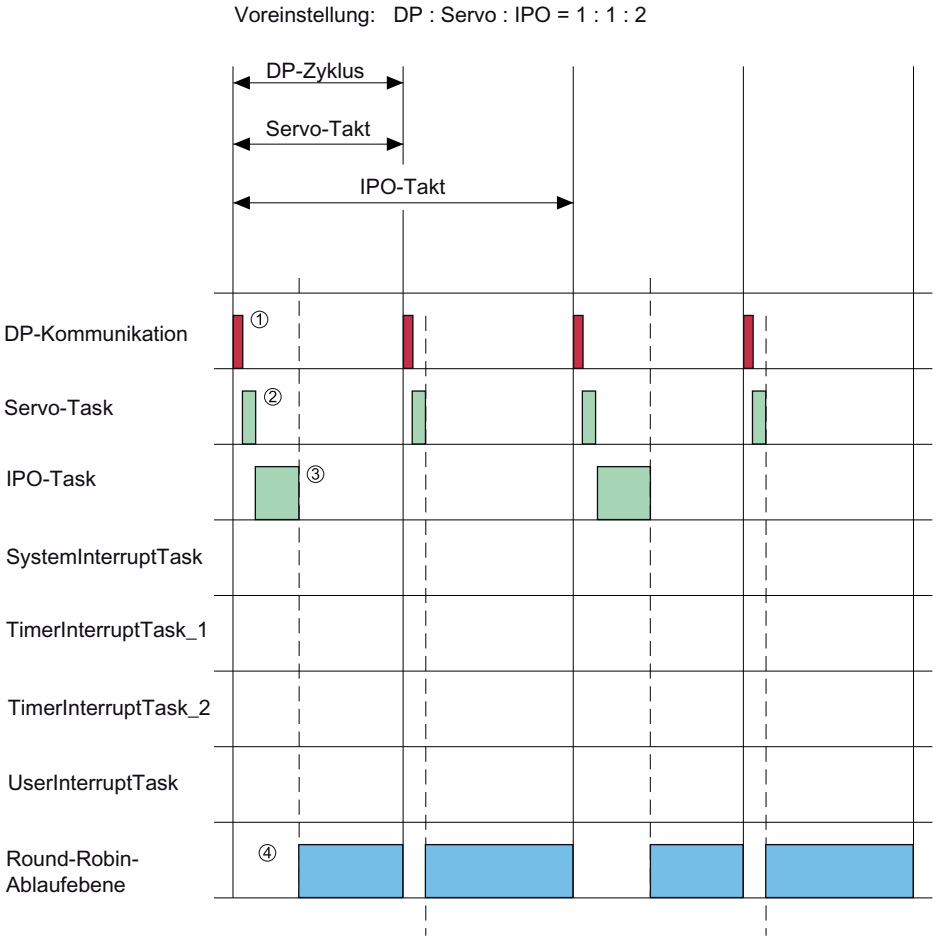
t = 1788-1845 ms	BackgroundTask läuft für 20 Servo-Takte
t = 1848 ms, 1851 ms	MotionTask 1 läuft für zwei Servo-Takte
t = 1854 ms, 1857 ms	MotionTask 2 läuft für zwei Servo-Takte
t = 1860-1917 ms	BackgroundTask läuft für 20 Servo-Takte

6.5.3 Abarbeitung der Tasks (Beispiele)

In den folgenden Beispielen soll skizzenhaft erläutert werden, wie die unterschiedlichen Tasks zeitlich bearbeitet werden.

Die Beispiele beziehen sich auf PROFIBUS und gelten analog für PROFINET IO mit IRT.

Beispiel 1: Zeitlicher Ablauf, wenn keine InterruptTask aktiv ist



1 Im Ablaufsystem sind die Takte in diesem Beispiel folgendermaßen gewählt:
DP-Zyklus: Servo-Takt: IPO-Takt: 1:1:2
Das bedeutet, dass in jedem DP-Zyklus die Servo-Task bearbeitet wird und in nur jedem zweiten DP-Zyklus die IPO-Task bearbeitet wird.

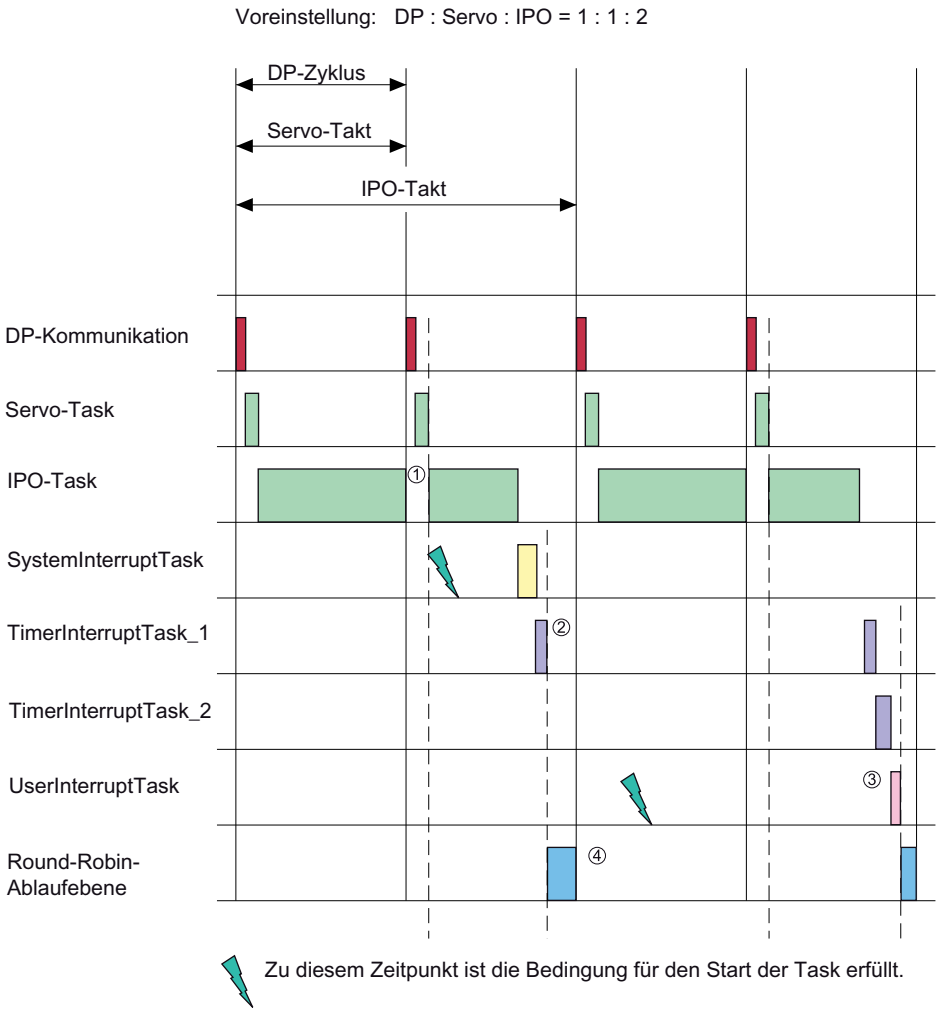
2 Die DP-Kommunikation hat die höchste Priorität. Danach kommt die Servo-Task

3 Die IPO-Task wird nach der Servo-Task bearbeitet.

4 Die Round-Robin-Ablaufebene wird in der restlichen Zeit bearbeitet.

Bild 6-45 Zeitliche Task-Bearbeitung, Beispiel: keine InterruptTask aktiv

Beispiel 2: Zeitlicher Ablauf, wenn eine InterruptTask aktiv ist und die IPOTask länger als 1 Servo-Takt dauert



1 Das Programm, das in der IPO-Task abgearbeitet wird, dauert länger als der Servo- Takt. Deshalb wird die IPO-Task unterbrochen und die DP-Kommunikation und der Servo-Task bearbeitet. Anschließend wird die IPO-Task wieder bearbeitet.

2 Nachdem die IPO-Task beendet ist, wird die SystemInterruptTask bearbeitet. Der Start der niederprioren TimerInterruptTask erfolgt im Anschluss.

3 Auch der UserInterruptTask wird erst bearbeitet, wenn die höherprioren Tasks fertig sind, auch wenn die Bedingung für die UserInteruptTask schon früher erfüllt war.

4 In der restlichen Zeit wird die Round-Robin-Ablaufebene bearbeitet.

Bild 6-46 Zeitliche Task-Bearbeitung, Beispiel: mit InterruptTask aktiv und IPOTask dauert länger als 1 Servo-Takt

Beispiel 3: Besonderheiten beim Taskwechsel im Multitasking

Beim Abfragen von Bedingungen in MotionTasks aber auch in der BackgroundTask muss das Verhalten des Multitasking mit Taskwechseln z.B. bei Wartebefehlen oder synchronen Befehlen berücksichtigt werden.

Beispiel:

Tabelle 6- 8 Beispiel:

	BackgroundTask	**MotionTask1**	**MotionTask2**
Quellen:	x=5	…	…
	… x=4 …	if x <> 5 _enableAxis Endif	if x = 5 _stopAxis Endif
		…	…
	BackgroundTask	**MotionTask1**	**MotionTask2**
Ablaufreihenfolge:	x=5		
	Taskwechsel →	if x <> 5 Bedingung nicht erfüllt endif	
		Taskwechsel →	if x = 5 Bedingung erfüllt Taskwechsel →
	x=4		
	Taskwechsel →	…	
		Taskwechsel →	…
			_stopAxis

Obwohl der Ablauf in MotionTask2 klar erscheint, kann es vorkommen, dass sich durch Taskwechsel der Ablauf anders als gedacht verhält.

Nach dem erneuten Wechsel in die BackgroundTask (x=4) sind die if-Abfragen bereits bearbeitet, die Abfrageergebnisse von vor dem Taskwechsel sind noch gültig. D. h., obwohl die Bedingung x=5 zu diesem Zeitpunkt nicht mehr erfüllt ist:

- wird in MotionTask1 _enableAxis nicht ausgeführt
- wird in MotionTask2 _stopAxis ausgeführt

Die Bearbeitung ist abhängig davon, wann der Taskwechsel ausgeführt wird.

Hinweis

Für einen reibungslosen Ablauf von Multitasking ist es empfehlenswert globale Variable nicht aus unterschiedlichen Tasks zu verwenden.

6.6 Task Trace Übersicht

6.6.1 Task Trace verwenden

Beschreibung

Der Task Trace ist ein Hilfsmittel zur Fehlersuche in der SIMOTION Multitasking-Umgebung. Folgende Möglichkeiten bietet der Task Trace:

- Ablauf der einzelnen Tasks und User Events (per Programmbefehl erzeugt) grafisch darstellen.
- Trace der einzelnen Anwendertasks.
- Konfiguration des Task Trace über IT Diag oder über Anwenderprogramm (Systemfunktionen) möglich.
- Ablage des Tracefiles auf Speicherkarte.
- Start des SIMOTION Task Profiler als eigene Applikation über IT Diag oder über die SCOUT Gerätediagnose.

Detaillierte Informationen über den Task Trace finden Sie im Funktionshandbuch *Task Trace*.

6.7 Taktsynchrone I/O-Verarbeitung an Feldbussystemen

Nachfolgend ist der prinzipielle Ablauf einer taktsynchronen I/O-Verarbeitung in einem Steuerungssystem mit I/O-Ankopplung über ein Feldbussystem dargestellt:

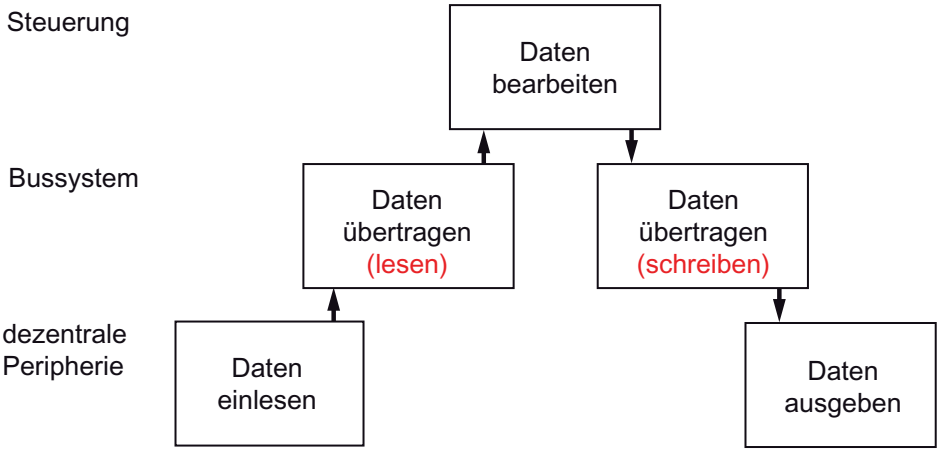


Bild 6-47 Datenfluss einer taktsynchronen I/O-Verarbeitung an Feldbussystemen

In einem taktsynchronen System sind die einzelnen Aktionen zeitlich miteinander synchronisiert. Dadurch werden kurze und reproduzierbare (d. h. gleich lange) Reaktionszeiten erreicht.

Zeittakt und Zeitbezug werden dabei von einem isochronen, d. h. getakteten Bussystem vorgegeben.

6.7.1 Datenprotokoll am PROFIBUS DP

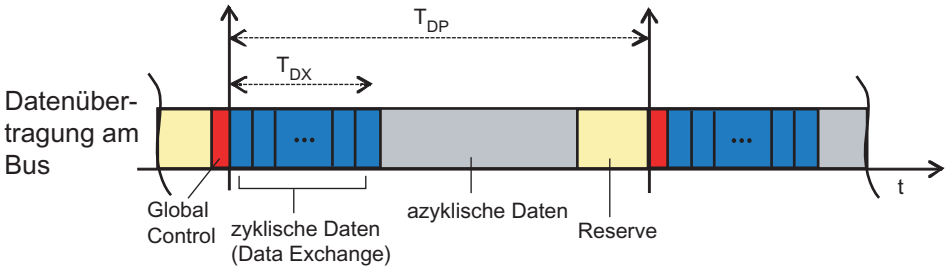


Bild 6-48 Datenprotokoll am PROFIBUS DP

Das Datenprotokoll am Bus beinhaltet

- ein äquidistantes **Global Control Telegramm (GC)**, welches den Bustakt definiert
- **zyklische Daten**: Daten, die in jedem Takt zwischen den Teilnehmern übertragen werden

  Die zyklische Datenkommunikation ermöglicht besonders kurze und reproduzierbare Prozessreaktionszeiten. Die Informationsweitergabe erfolgt dabei *in jedem Zyklus*.

  Seitens eines Anwenderprogrammes wird auf diese Daten über das Prozessabbild bzw. über Peripheriedirektzugriffe zugegriffen.
- **azyklische Daten**: Daten, die nur bei Bedarf und in größeren Mengen übertragen werden

  Die azyklische Übertragung ist besonders für eine nicht zeitkritische Übertragung großer Datenmengen geeignet, die Daten können dabei auf mehrere Takte aufgeteilt werden.

  Beispiel für azyklische Daten: Alarme, Diagnosedienste und Datensätze

  Seitens eines Anwenderprogrammes wird auf diese azyklischen Daten z. B. über die Befehle _readrecord, _writerecord zugegriffen.
- **Reserve**: Restzeit bis zum nächsten Global Control.

  Die Restzeit ist in Abhängigkeit von der aktuell laufenden azyklischen Kommunikation unterschiedlich groß und beträgt maximal T_{DP}-T_{DX}.

6.7.2 Datenprotokoll am PROFINET IO

Beschreibung

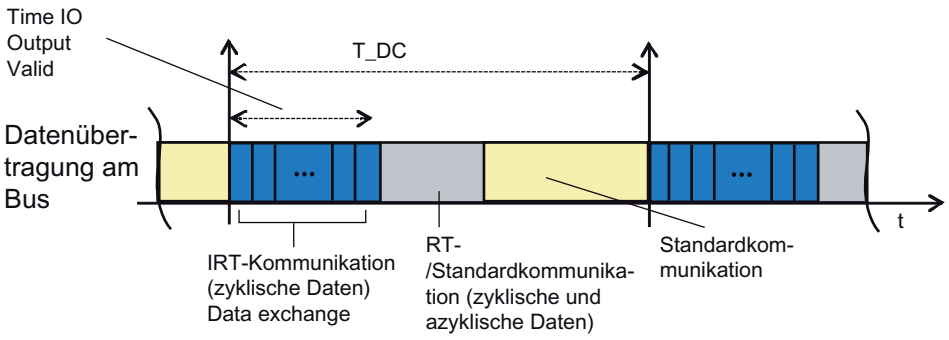


Bild 6-49 Datenprotokoll am PROFINET IO

Das Datenprotokoll am Bus enthält:

- Die IRT-Kommunikation, die in einem geplanten Zeitintervall zyklisch Daten überträgt.
- Die RT- und Standardkommunikation, in der folgende Daten übertragen werden können:
  - Zyklische RTC-Telegramme, mit denen z. B. IO-Daten übertragen werden können.
  - Azyklische RTA-Telegramme, mit denen z. B. Alarme übertragen werden können.
  - NRT-Daten (Standardkommunikation), Standard-Ethernet Telegramme, mit denen alle, auf Standard-Ethernet basierende Protokolle, übertragen werden können.
- Standardkommunikation (letztes Intervall vor Synchronisationstelegramm); hier darf nur Kommunikation stattfinden, die bis zum Einsetzen des Synchronisationstelegramms beendet ist.

Time IO Output Valid beschreibt die Zeit, in der die neuen Output-Daten verfügbar sind.

6.7.3 Taktsynchrone Datenbearbeitung

Taktsynchrone Applikation

Bei einer taktsynchronen Applikation sind der Verarbeitungstakt der Steuerung und eventuell angeschlossene Antriebe bzw. dezentrale Peripherie auf den Bus-Zyklus synchronisiert.

Taktsynchrone Datenübertragung am PROFIBUS DP

Bei PROFIBUS ist der Bus-Zyklus auf das Global Control Telegramm (GC) synchronisiert. Das GC verkörpert den Bustakt und ist gleichzeitig der Systemtakt für das Gesamtsystem.

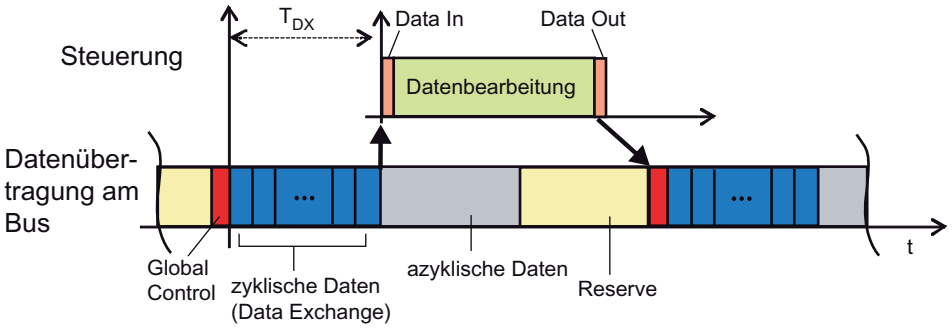


Bild 6-50 Aufbau eines Bus-Zyklusses bei PROFIBUS DP

Taktsynchrone Datenübertragung am PROFINET IO

Bei PROFINET IO ist der Verarbeitungstakt der Steuerung auf die IRT-Kommunikation synchronisiert.

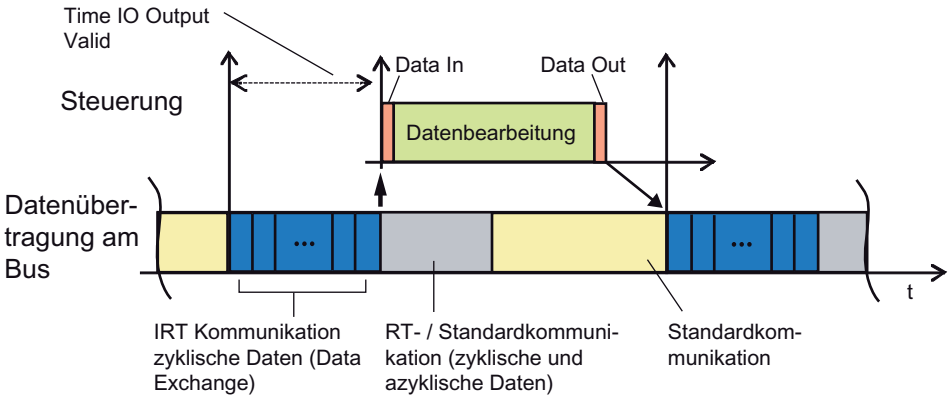


Bild 6-51 Aufbau eines Bus-Zyklus bei PROFINET IO

Die mittels zyklischer Datenkommunikation übertragenen Daten können somit taktsynchron verarbeitet werden.

Taktsynchrone Datenerfassung mit PROFIBUS DP

In einem taktsynchronen Gesamtsystem sind Datenerfassung, Datenbearbeitung und Datenausgabe auf den gemeinsamen Systemtakt bezogen.

Die Applikation wird gestartet, wenn der zyklische Datentransfer beendet ist (T_{DX}).

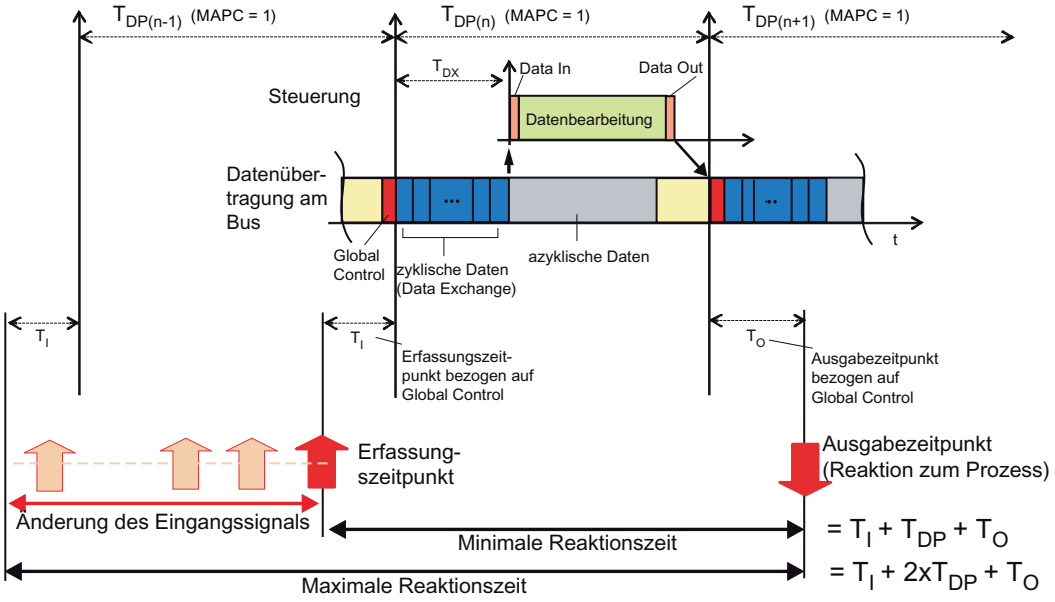


Bild 6-52 Berechnung der Reaktionszeiten (PROFIBUS DP)

Taktsynchrone Datenübertragung mit PROFINET IO

Die Applikation wird gestartet, wenn der zyklische Datentransfer beendet ist (Time IO Output Valid).

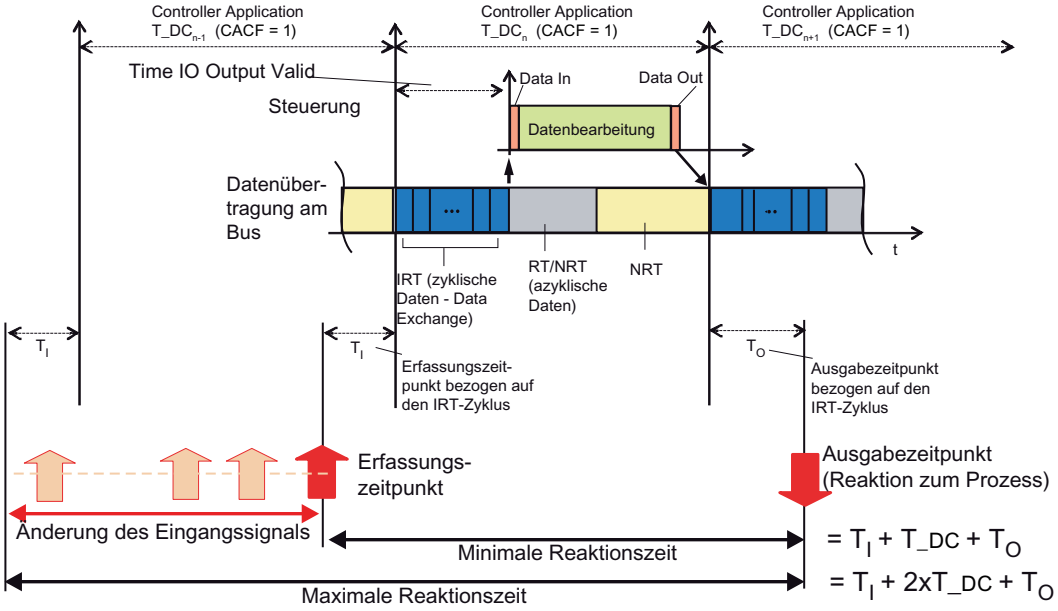


Bild 6-53 Berechnung der Reaktionszeiten (PROFINET IO)

Berechnung der Reaktionszeiten (PROFIBUS DP und PROFINET IO)

Damit zum Startzeitpunkt des neuen Bus-Zyklus ein konsistenter Zustand der Eingänge eingelesen werden kann, muss der Einlesevorgang in der dezentralen Peripherie um die Zeit T_I versetzt vorverlegt werden.

Die Zeit T_I umfasst die mindestens die Signalaufbereitungs- und Wandlungszeit an den Elektronikmodulen und im Falle eines modularen ET 200 Peripheriesystems auch die Übertragungszeit der Eingänge am ET 200-Rückwandbus. Im Allgemeinen wird T_I so gewählt, dass es für alle Baugruppen gleich ist, die langsamste Baugruppe bestimmt dabei die Zeit.

Über die Zeit T_O wird sichergestellt, dass die Prozessreaktionen des Anwenderprogramms (Datenbearbeitung) zeitgleich und konsistent auf die "Klemmen" der dezentralen Peripherie am Bus durchgeschaltet werden. Analog dazu wird bei Antrieben der Sollwert für die Antriebsregelung zur Verfügung gestellt. Die Zeit T_O umfasst mindestens die Zeit für den zyklischen Datenaustausch aller Teilnehmer am Bus, im Falle eines modularen ET 200 Peripheriesystems die Übertragungszeit der Ausgänge am ET 200-Rückwandbus und die Signalaufbereitungs- und Wandlungszeit an den Elektronikmodulen der dezentralen Peripherie.

Reaktionszeiten bzw. Totzeiten (bei Antrieben)

Durch die Synchronisierung der Einzelzyklen wird es möglich, im Takt "n-1" die Eingangsdaten zu lesen, im Takt "n" die Daten zu übertragen und zu bearbeiten und zu Beginn des Taktes "n+1" die errechneten Ausgangsdaten zu übertragen und noch im selben Takt auf die "Klemmen" zu schalten.

PROFIBUS

Hiermit ergibt sich eine reale Prozessreaktionszeit von:

- minimal $T_I + T_{DP} + T_O$
- maximal $T_I + 2 \cdot T_{DP} + T_O$

PROFINET

Hiermit ergibt sich eine reale Prozessreaktionszeit von:

- minimal $T_I + T\_DC + T_O$
- maximal $T_I + 2 \cdot T\_DC + T_O$

6.7.4 Zeitverhalten bzgl. Datenbearbeitung in der Steuerung

Datenbearbeitung in der ServoSynchronousTask

- Bei einer Datenbearbeitung in der ServoSynchronousTask kann bei der Einstellung Bustakt : Servo = 1 : 1 eine Verarbeitungszeit in der Steuerung von einem Buszyklustakt erreicht werden.

  Dies gilt z. B. für das Setzen von Ausgangsdaten, z. B. DO oder AO, abhängig von Eingangssignalen.

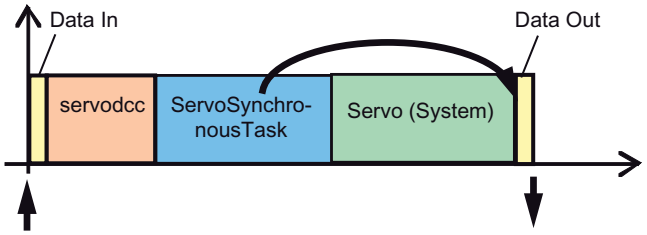


Bild 6-54 Datenverarbeitung in der Steuerung - Setzen der Ausgangsdaten

- Bei einer Beeinflussung von Achsdaten / Achsbewegung ist die Verarbeitungszeit davon abhängig, ob die Daten bereits im Servo wirksam sind (z. B. überlagerter Sollwert oder Stellwert).

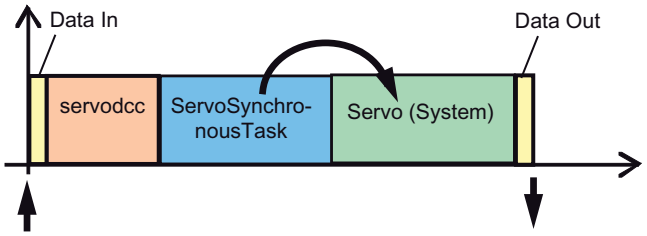


Bild 6-55 Datenverarbeitung in der Steuerung - Beeinflussung von Bewegungsdaten

- Werden die Daten oder Befehle erst im IPO wirksam (z. B. Absetzen von Bewegungsbefehlen), so werden die Ausgangsdaten auf die Ausgänge erst mit dem nächsten Servo wirksam.

  Hinweis: Um die Laufzeit der ServoSynchronousTask gering zu halten (und damit die Zeit bis zum Data Out) ist eine Programmierung solcher Aufgaben in diesem Fall in der IPOSynchronousTask zu bevorzugen.

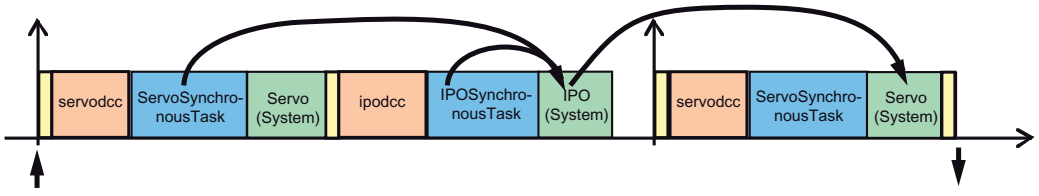


Bild 6-56 Datenverarbeitung in der Steuerung - Ausgangsdaten im IPO wirksam

Datenverarbeitung in der IPOSynchronousTask

- Bei einer Datenverarbeitung in der IPOSynchronousTask werden Ausgangsdaten auf I/O mit dem nächsten Servo wirksam.

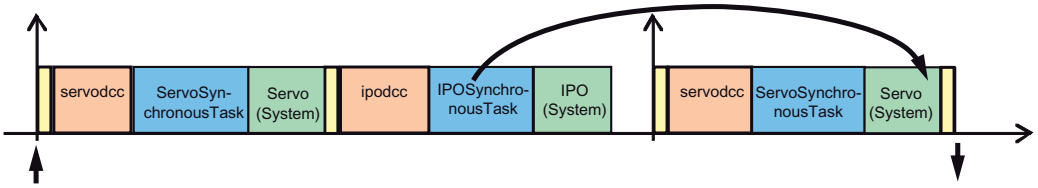


Bild 6-57 Datenverarbeitung in der Steuerung - Ausgangsdaten im nächsten Servo wirksam

- Bzw. Bewegungsdaten in dem darauf folgenden IPO bzw. Servo (siehe oben)

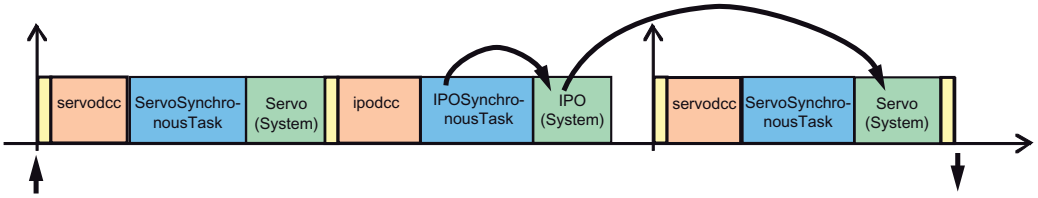


Bild 6-58 Datenverarbeitung in der Steuerung - Ausgangsdaten im IPO wirksam - Beeinflussung von Bewegungsdaten

In den Fällen, in denen die Ausgangsdaten der I/O erst im nächsten Servo übertragen werden, erhöht sich die Reaktionszeit um jeweils einen Bustakt (bei Bustakt : Servo : IPO = 1 : 1 : 1).

Szenario	Empfehlung für Anwenderprogramm-Task
schnelle Klemme-Klemme - Reaktion bei I/O	ServosynchronousTask verwenden
Schnelle Beeinflussung Sollwert (Servo- Ebene)	ServosynchronousTask verwenden
Umschalten /Überlagerung Bewegung (z. B. _stop, _move, _pos, …) z. B. Druckmarke	IPOsynchronousTask verwenden

Ab V4.1 kann in Ausnahmefällen der IPO-Anteil auch im Servo gerechnet werden, siehe dazu **Bewegungsführung / Interpolator** im Handbuch **TO Achse**.

6.7.5 Zeitverhalten bzgl. Datenübertragung von der Erfassung zur Bearbeitung

Bei Verwendung eines Bussystems PROFIBUS oder PROFINET sind bei der Kalkulation des Systemtaktes die für die Datenübertragung erforderlichen Zeiten (T_{DX}) zu berücksichtigen. Kleine T_{DX} erlauben kürzere Taktzyklen oder mehr Bearbeitungszeit für die Applikation bei gleicher Taktlänge.

- Bei PROFIBUS DP wird T_{DX} von HW Konfig ausgewiesen
- Am SIMOTION D stellt sich am PROFIBUS-integrated je nach Ausbaugrad des SINAMICS-Projektes (Peripherieausbau an SIMOTION D und SIMOTION CX32) automatisch eine Übertragungszeit im Bereich 125 µs ≦ T_{DX} ≦ 375 µs ein
- Bei PROFINET IO wird T_{DX} = 0,5 • T_{DP} eingestellt

6.7.6 Zeitverhalten für Datenerfassung und Datenausgabe

Bei der Projektierung der Zeit T_O ist T_{DX} zu berücksichtigen. Es gilt immer: $T_{DX} \leqq T_O$.

- Die Zeiten können in HW Konfig eingestellt werden.
- Bei Verwendung von TM15/TM17 High Feature Modulen zur Erfassung und Ausgabe sind folgende Zeiten anzusetzen:
  - Für die Erfassung T_I + 1 DRIVE-CLiQ Zyklus (typisch 125 µs)
  - Für die Ausgabe T_O + 1 DRIVE-CLiQ Zyklus (typisch 125 µs)

Siehe auch

Ermittlung von Tdp, Ti und To über HW Konfig bei ET 200-Peripherie am PROFIBUS (Seite 285)

6.7.7 Ermittlung von Tdp, Ti und To über HW Konfig bei ET 200-Peripherie am PROFIBUS

In HW Konfig können in der Maske **Taktsynchronisation** (Aufruf über Menü **Bearbeiten > Taktsynchronisation**) die Zeiten für T_{DP}, T_I und T_O ermittelt werden.

Das Dialogfeld gibt einen Überblick über die für die Taktsynchronität eingestellten Parameter und Zeiten der beteiligten Objekte PROFIBUS, Slaves und Module in den jeweiligen Slaves.

Alle Zeiten im Dialog sind in Millisekunden angegeben.

- T_I/T_O-Einstellung
  - **im Netz** bedeutet, dass die Zeiten T_I und T_O zentral "für alle Slaves gleich" eingestellt wurden
  - **im Slave** bedeutet, dass die Zeiten T_I und T_O an jedem Slave individuell eingestellt werden
- T_I, T_O, T_{DP}

  Zeigt die aktuell eingestellten bzw. errechneten Werte T_I, T_O und T_{DP} für das markierte DP-Mastersystem an.

Eine detaillierte Beschreibung der Maske sowie aller weiteren Parameter sind der Online-Hilfe von HW Konfig zu entnehmen.

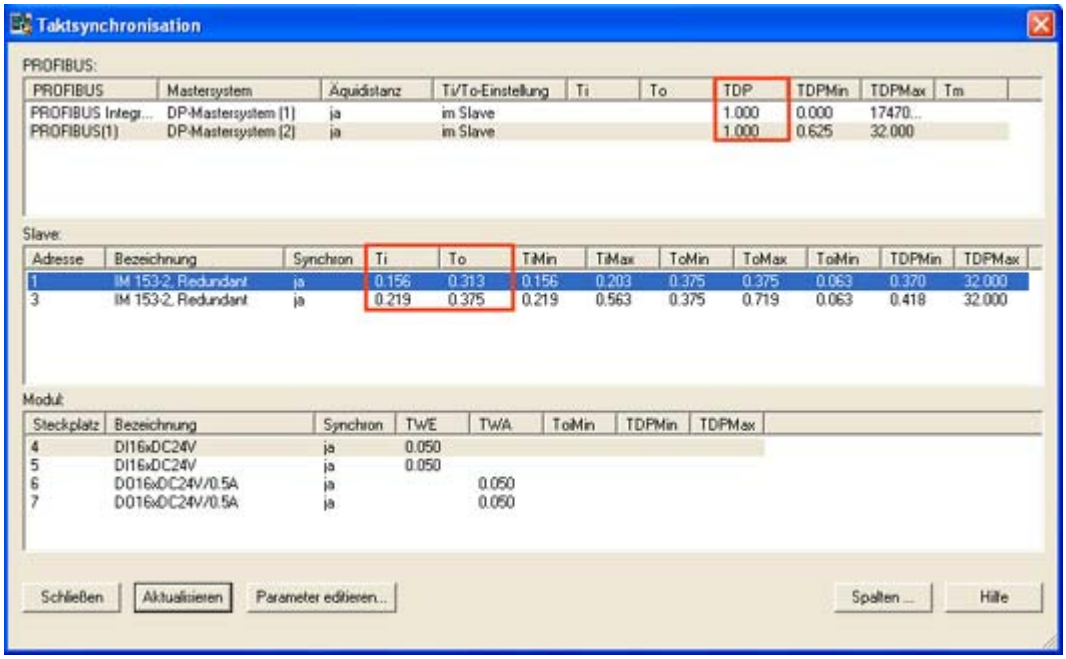


Bild 6-59 Maske Taktsynchronisation in HW Konfig

Hinweis

Eine volle Taktsynchronität von "Klemme" zu "Klemme" ist nur dann möglich, wenn alle in der Kette beteiligten Komponenten die Systemeigenschaft "Taktsynchronität" unterstützen. Achten Sie bei der Auswahl im PM10 Katalog bzw. im Hardware Katalog von HW Konfig auf den Eintrag **Taktsynchronität** im Informationsfeld der Baugruppe.

Eine aktuelle Liste der taktsynchronen Peripheriebaugruppen finden Sie im Internet unter http://support.automation.siemens.com/WW/view/de/14747353

6.7.8 Handhabung von Taktuntersetzung

Bei Taktuntersetzung ist folgendes zu beachten:

- Am Ende einer IPOSynchronousTask wird das Prozessabbild mit dem nächstmöglichen Servo (Data Out) ausgegeben (= reaktionszeitoptimiert). Dieses kann bei Untersetzung Servo-Takt zu IPO-Takt dazu führen, dass die Daten einen /mehrere Servo-Takte früher oder später innerhalb eines IPO-Taktes ausgegeben werden, wenn die Peripheriezugriffe über die IPOSynchronousTask erfolgen.
- Am Ende der Servo-Ablaufebene wird das Prozessabbild der ServoSynchronousTask mit dem nächstmöglichen Bus-Takt ausgegeben (= reaktionszeitoptimiert).
  - Bei PROFINET und Untersetzung PROFINET-Takt zu Servo-Takt kann dieses dazu führen, dass die Daten einen /mehrere Bus-Takte früher oder später innerhalb eines Servo-Taktes ausgegeben werden, wenn die Peripheriezugriffe über die ServoSynchronousTask erfolgen und die Laufzeit der Servo-Ablaufebene über einen Bustakt hinaus schwankt.
  - Bei PROFIBUS werden die Daten immer mit dem ersten Bus-Takt ausgegeben, da die Servo-Ablaufebene immer mit dem ersten Bus-Takt abgeschlossen sein muss.

Bei unterschiedlicher Laufzeit der Servo-Ablaufebene in den einzelnen Takten kann somit die Klemme-Klemme-Zeit variieren.

Soll statt eines reaktionszeitoptimierten Verhaltens eine immer konstante Reaktionszeit erreicht werden, so muss:

- bei PROFIBUS:
  - ein Untersetzungsverhältnis Servo : IPO = 1 : 1 eingestellt werden, damit Peripheriezugriffe aus der IPOSynchronousTask immer taktsynchron erfolgen.

    Anmerkung: Peripheriezugriffe aus der ServoSynchronousTask sind bei PROFIBUS immer taktsynchron
- bei PROFINET:
  - ein Untersetzungsverhältnis Bus-Takt : Servo : IPO = 1 : 1 : 1 eingestellt werden, damit Peripheriezugriffe aus der IPOSynchronousTask immer taktsynchron erfolgen
  - ein Untersetzungsverhältnis Bus-Takt : Servo = 1 : 1 eingestellt werden, damit Peripheriezugriffe aus der ServoSynchronousTask immer taktsynchron erfolgen

6.7.9 PROFIBUS DP in HW Konfig laufzeitoptimiert projektieren

Einleitung

Mit optimaler Projektierung in HW Konfig kann ein günstiges Laufzeitverhalten der SIMOTION-Firmware bewirkt werden.

Laufzeitoptimierte Projektierung hinsichtlich Stationstopologie

An einem isochronen PROFIBUS DP sollten isochrone Antriebe, die für ein TO Achse vorgesehen sind, sowie isochrone DP-Slaves fortlaufende Stationsadressen erhalten (z.B. 11, 12, 13, 14, …). Dabei dürfen Adresslücken auftreten (z.B. 11, 13, 16 …), in die jedoch keine anderen Slaves eingeordnet werden dürfen.

Laufzeitoptimierte Projektierung bei i-Slave

Bei der Slot-Definition einer I-Slave-Projektierung sollten die Slots, für die keine Konsistenz über den gesamten Slot eingestellt wird, in fortlaufenden Slotnummern angeordnet werden.

Laufzeitoptimierte Projektierung bei ET200 Slaves

Mit der Wahl einer geeigneten PROFIBUS DP-Topologie lässt sich der Systemtakt bzw. die Prozessreaktionszeit positiv beeinflussen:

- Um kurze und reproduzierbare Prozessreaktionszeiten zu gewährleisten, sollte die taktsynchrone und nicht taktsynchrone Peripherie auf eigene DP-Mastersysteme aufgeteilt werden.
- Die Taktsynchronität sollte nur für die Stationen und Peripheriemodule aktiviert werden, für die diese Eigenschaft auch benötigt wird.
- Um den taktsynchronen DP-Zyklus möglichst kurz zu halten, können durch eine Reduzierung der Slave-Stationen die Grundlastzeiten kurz gehalten werden. Aus diesem Grund sollte am taktsynchronen DP-Mastersystem auf den Einsatz von Operatorpanels und Textdisplays verzichtet werden. Betreiben Sie deshalb diese Anzeigegeräte an einem eigenen, nicht taktsynchronen DP-Mastersystem oder über die Ethernet-Schnittstelle.
- Kurze Filterzeiten an den Peripherieeingängen verkürzen die Prozessreaktionszeit (daher z.B. kleinste Filtereinstellung bei den ET 200S High Feature Digitaleingaben wählen)

- Manche Peripheriebaugruppen verfügen über eine spezielle Betriebsart, um möglichst kurze Zykluszeiten zu erreichen (z.B. Fast Mode bei ET 200M Analogeingabe oder ET 200S SSI-Modul, ...)
- Die Anzahl der Peripherie-Eingänge und -Ausgänge pro Station, geht in die Zeiten T_I und T_O ein. Die größten T_I- und T_O-Zeiten der Einzelkomponenten bestimmen die T_I- und T_O-Zeiten des gesamten Systems. Achten Sie auf eine zweckmäßige Peripherieverteilung, um die Zeiten für T_I und T_O möglichst kurz zu halten.

  Folgende Konstellationen erweisen sich im Zeitverhalten als ungünstig:

  – Wenige Interfacemodule (ET 200M, ET 200S) mit vielen Peripheriebaugruppen
  – Viele Interfacemodule (ET 200M, ET 200S) mit wenigen Peripheriebaugruppen

  Folgende Konstellationen erweisen sich im Zeitverhalten als günstig:

  – Wenige Slaves (Interfacemodule) am DP-Strang
  – Interfacemodule mit wenigen Peripheriebaugruppen
  – Interfacemodule mit ausschließlich Eingabebaugruppen
  – Interfacemodule mit ausschließlich Ausgabebaugruppen
  – ET 200M: Stecken Sie die Ausgabebaugruppen mit der längsten Verarbeitungszeit links in der ET 200M.
  – ET 200M: Stecken Sie die Eingabebaugruppen mit der längsten Verarbeitungszeit rechts in der ET 200M.
  – Eine hohe PROFIBUS-Übertragungsgeschwindigkeit trägt zur Verkürzung der DP Zykluszeit bei.

Sie können die resultierende DP-Zykluszeit, die Zeit für T_I und T_O den Dialogfeldern von HW Konfig entnehmen und dort ändern, wenn Sie Ihre geplante DP-Toplogie in HW Konfig beispielhaft konfigurieren.

Siehe auch

Zeit- und Ebenenüberläufe (Seite 253)

6.8 Einbindung von DCC in das SIMOTION Ablaufsystem

6.8.1 Einleitung

Einleitung

Drive Control Chart (DCC) ermöglicht es Ihnen reglungs- und steuerungstechnische Aufgaben antriebsnah zu implementieren. Dazu stehen ihnen ein Satz von Bausteinen (DCB), zur Verfügung, die Sie über ein Projektierungstool (DCC-Editor) in so genannten Plänen grafisch miteinander verschalten können. Die DCBs werden Ihnen in Form einer Bibliothek (DCBLIB) zur Verfügung gestellt.

Hinweis

Eine Exception in den DCC-Tasks ruft keine SIMOTION-Fault-Task auf.

Die erstellten Pläne können sowohl auf den SIMOTION Plattformen also auch auf den SINAMICS-Antrieben zum Ablauf kommen.

Im Nachfolgenden wird nur DCC für Simotion beschrieben.

Weitere Literatur

DCB-Lib und DCC-Editor-Beschreibung

6.8.2 Ablaufmodell für DCC-Bausteine (DCB)

Beschreibung

Die einzelnen DCC-Bausteine (DCB – Drive Control Block) werden im DCC-Editor in Ablaufgruppen organisiert. Diese Ablaufgruppen können Sie den DCC-Tasks im DCC-Editor frei zuordnen (Task T1 bis T5). Die DCC-Tasks im DCC-Editor entsprechen den DCC-Tasks im SIMOTION Ablaufsystem.

Task im DCC-Editor	Ablaufebene	Task
T1	Servo	servoDcc
T2	IPO	ipoDcc
T3	IPO_2	IpoDcc_2
T4	DccAux	dccAux
T5	DccAux_2	dccAux_2

Ablaufgruppen können zur Laufzeit über binären Ausgänge von Bausteinen aktiviert bzw. deaktiviert werden. -Detaillierte Beschreibung im Editor.

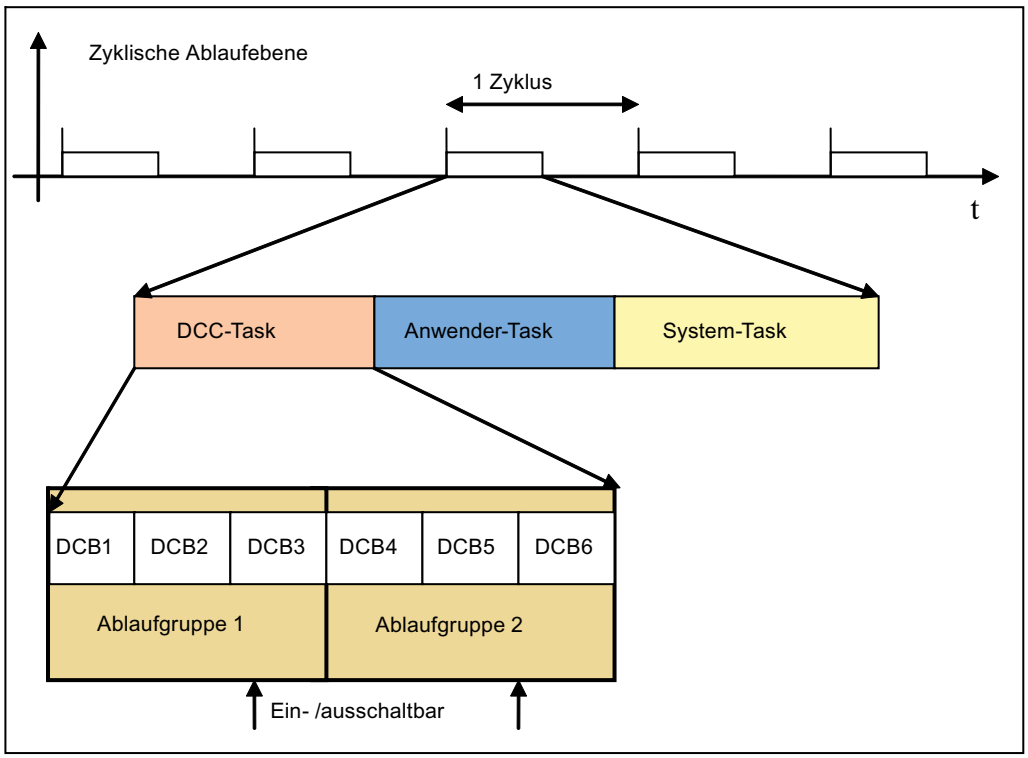


Bild 6-60 Ablaufmodell der Ablaufgruppen für DCBs

Im Anwendermodell ordnet sich die DCC-Task in der Ablaufebene neben den Anwenderprogramm-Tasks (Ablaufumgebung für Anwenderprogramme in ST, MCC, KOP/FUP) und den System-Tasks ein. Die DCC-Task wird vom Engineeringsystem (ES) bei Bedarf angelegt; sie ist somit *nicht* standardmäßig im System vorhanden.

Ausführung der DCC-Pläne

Die Ausführung von DCC-Plänen ist nicht an den Taskzustand geknüpft, sondern an den Zustand der ihr übergeordneten Ablaufgruppe. Diese werden abhängig von Betriebszustandswechseln ein- bzw. ausgeschaltet.

Betriebszustand	Aktion
ANLAUF->RUN	Ablaufgruppen EIN
ABSTEUERN->STOPU	Ablaufgruppen AUS
ABSTEUERN->STOP	Ablaufgruppen AUS

Alle anderen Änderungen im Betriebszustand haben keine Auswirkung auf die Ablaufgruppen.

Siehe auch

servoDccTask in der Servo-Ebene (Seite 292)

ipoDcc Task in der IPO-Ebene (Seite 293)

ipoDcc_2 Task in der IPO2-Ebene (Seite 295)

Ablaufebenen für DccAux und DccAux_2 (Seite 296)

6.8.3 servoDccTask in der Servo-Ebene

Beschreibung

Der T1(DCC)-Task läuft in der Servo-Ebene des SIMOTION-Ablaufsystems. Die Servo-Ebene wird synchron zum Datenaustausch mit der taktsynchronen Peripherie aufgerufen und wird vorzugsweise für schnelle Regelungsaufgaben eingesetzt.

- Beim Start werden die zyklischen I/O-Daten für die Technologieobjekte und servosynchronen I/O-Variablen von der Schnittstelle zur taktsynchronen Peripherie in das Ablaufsystem kopiert.
- Am Ende werden die zyklischen I/O-Daten wieder aus dem Ablaufsystem kopiert und z. B. über PROFIBUS zu einem PROFIBUS-Teilnehmer übertragen.

Die folgende Grafik zeigt den zeitlichen Ablauf einer Task in der Servo-Ebene.

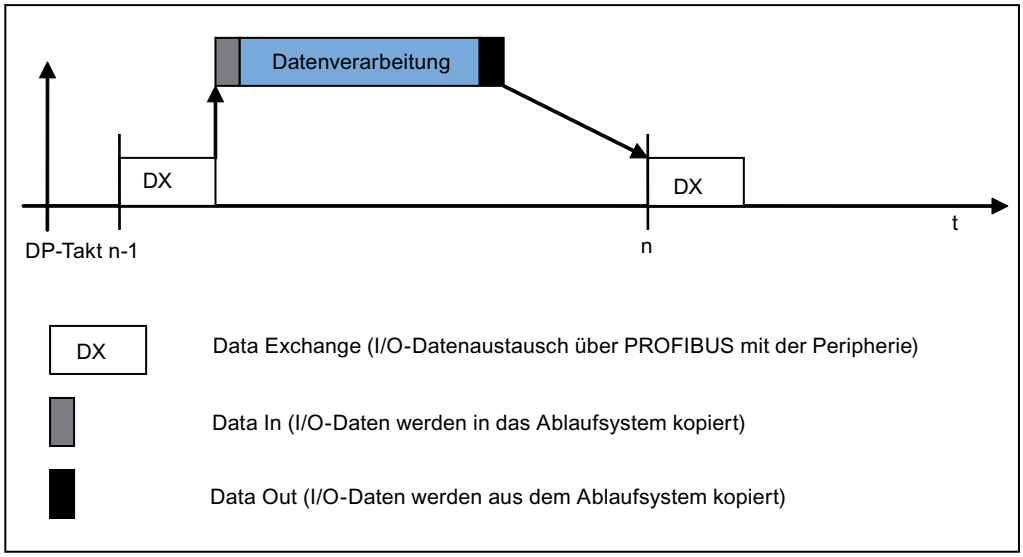


Bild 6-61 T1(DCC) Task in Servo-Ebene

Die T1(DCC) Task wird zu Beginn der Servo-Ebene automatisch gestartet und ausgeführt.

Die folgende Grafik zeigt die Reihenfolge der Tasks in der Servo-Ebene:

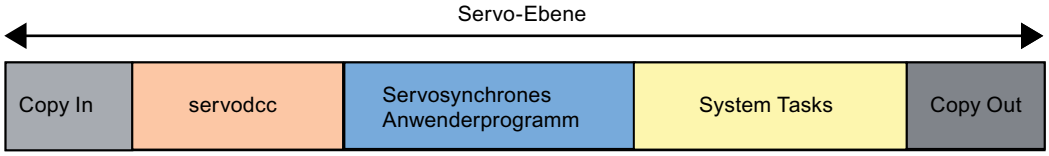


Bild 6-62 Servo-Ebene mit T1(DCC) Task

Ebenenüberlauf in der Servo-Ebene

Tasks in der Servo-Ebene laufen mit gleicher Priorität und unterbrechen sich deswegen nicht gegenseitig. Wenn nicht alle Tasks der Servo-Ebene in einem Zyklus berechnet werden können, kommt es zu einem Ebenenüberlauf und das System geht in den Betriebszustand STOP, die Anlaufsperre wird gesetzt. Erst nach einem erneuten Hochlauf (Netz aus /ein) oder Download kann das System wieder in den Betriebszustand RUN gehen.

Siehe auch

Systemtakte festlegen (Seite 241)

6.8.4 ipoDcc Task in der IPO-Ebene

Beschreibung

Die IPO-Ebene läuft mit der gleichen Zykluszeit wie die Servo-Ebene oder in einem ganzzahligen Untersetzungsverhältnis. Tasks in der Servo-Ebene haben eine höhere Priorität und können somit Tasks in der IPO-Ebene unterbrechen. Zusätzlich werden in der IPO-Ebene alle I/O-Daten, die nicht in der Servo-Ebene kopiert werden, in das Ablaufsystem kopiert. Die Daten können sowohl von der taktsynchronen wie der nicht taktsynchronen Peripherie kommen.

In der Systemtask der IPO-Ebene laufen die Befehlsverarbeitung und die Sollwertgenerierung der Technologieobjekte ab.

Die folgende Grafik zeigt den zeitlichen Ablauf in der IPO-Ebene.

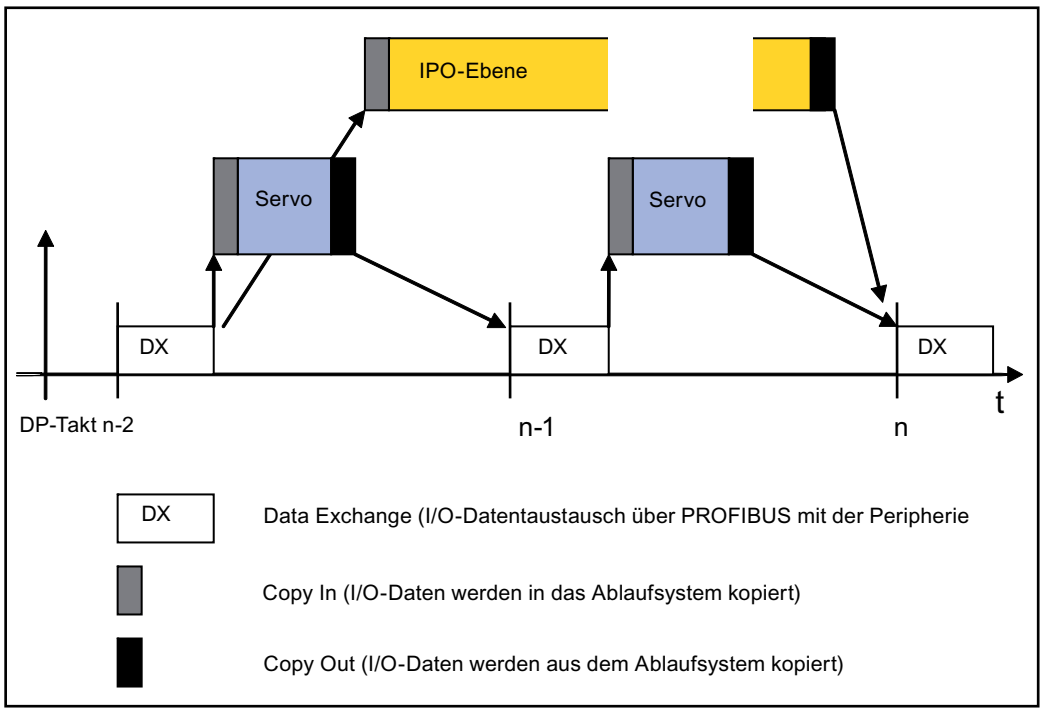


Bild 6-63 IPO-Ebene mit T2(DCC)-Task

Die T2(DCC) Task wird zu Beginn der IPO-Ebene automatisch gestartet und ausgeführt.

Die folgende Grafik zeigt die Ablaufreihenfolge in der IPO-Ebene

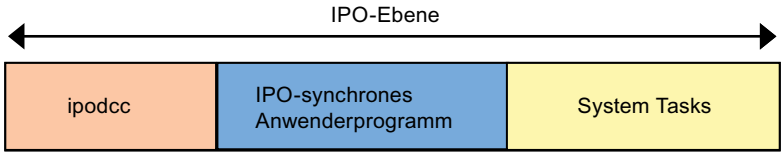


Bild 6-64 Ablaufreihenfolge in der IPO-Ebene

Ebenenüberlauf in der IPO-Ebene

Tasks in der IPO-Ebene laufen mit gleicher Priorität und unterbrechen sich deswegen nicht gegenseitig. Eine Task in der höher prioren Servo-Ebene kann jederzeit die Bearbeitung in der IPO-Ebene unterbrechen, was eventuell zu einem Ebenenüberlauf führen kann, da nicht alle Tasks in der Ebene bearbeitet werden können.

Mit dem Standardverhalten geht das Gerät bei einem Ebenenüberlauf in den Betriebszustand STOP und die Anlaufsperre wird gesetzt. Erst nach einem erneuten Hochlauf (Netz aus/ein) oder Download kann das System wieder in den Betriebszustand RUN gehen.

Sie können allerdings auch konfigurieren, dass das System bis zu 5 Überläufe hintereinander toleriert. Das Verhalten müssen Sie entsprechend in der Task-Konfiguration einstellen.

Siehe auch

Systemtakte festlegen (Seite 241)

Zeit- und Ebenenüberläufe (Seite 253)

SynchronousTasks (Seite 216)

Taktsynchrone Datenbearbeitung (Seite 279)

6.8.5 ipoDcc_2 Task in der IPO2-Ebene

Beschreibung

Die Zykluszeit der IPO2-Ebene ist ein ganzzahliges Vielfaches der Zykluszeit der IPO-Ebene. Es ist allerdings nicht möglich, den IPO2-Takt gleich dem IPO-Takt einzustellen.

Die Priorität der IPO2-Ebene ist niedriger als die der IPO-Ebene. Somit kann sowohl die IPO-Ebene wie die Servo-Ebene die Bearbeitung unterbrechen.

Die T3(DCC)-Task wird in der IPO2-Ebene wie folgend eingeordnet:

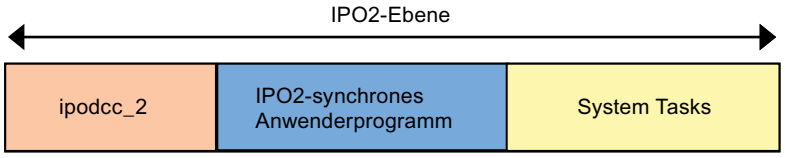


Bild 6-65 Reihenfolge der Abarbeitung in der IPO2-Ebene

Ebenenüberlauf in der IPO2-Ebene

Die IPO2-Ebene verhält sich bei einem Ebenenüberlauf wie die IPO-Ebene.

Siehe auch

ipoDcc Task in der IPO-Ebene (Seite 293)

Systemtakte festlegen (Seite 241)

6.8.6 Ablaufebenen für DccAux und DccAux_2

Beschreibung

Die beiden DCC Ablaufebenen DccAux und DCCAux_2 werden synchron zum System aufgerufen.

Die Ebenen DCCAux und DCCAux_2 haben folgenden Eigenschaften:

- DccAux hat vielfachen Takt von Ipo_2.
- DccAux_2 hat vielfachen Takt von DccAux.
- DccAux hat eine höhere Priorität als DccAux_2
- Synchrone Trace-Aufzeichnungen sind möglich.
- Bis zu 5 Ebenenüberläufe werden toleriert (Standardeinstellung ist 1). Die aktuelle Anzahl der Ebenenüberläufe kann ausgelesen werden.
- Bei einem Ebenenüberlauf wird die DCCTask in der nächsten Task zu Ende gerechnet.

Tabelle 6- 9 Systemvariablen zum Auslesen der Ebenenüberläufe

Systemvariable	Datentyp	Bedeutung
_device.numberOfSummarizedTaskOverflow.DccAux	UDINT	Anzahl der Überläufe der DccAux-Ebene seit Systemhochlauf
_device.numberOfSummarizedTaskOverflow.DccAux_2	UDINT	Anzahl der Überläufe der DccAux_2-Ebene seit Systemhochlauf

Siehe auch

Systemtakte festlegen (Seite 241)

6.8.7 Datenaustausch zwischen Bausteinen

6.8.7.1 Datenaustausch zwischen Blöcken (Überblick)

Überblick

Im DCC-Editor können Sie die Anschlüsse verschiedener Bausteine miteinander verschalten. Für den Datenaustausch zwischen den Bausteinen sind grundsätzlich drei Szenarien zu unterscheiden:

- Sie haben Anschlüsse von Bausteinen miteinander verschaltet, die in der gleichen Ablaufebene liegen.
- Sie haben den Ausgang eines Bausteins mit dem Eingang eines Bausteins in einer höheren Ablaufebene verschaltet.
- Sie haben den Ausgang eines Bausteins mit dem Eingang eines Bausteins in einer niedrigeren Ablaufebene verschaltet.

Siehe auch

Datenaustausch zwischen Bausteinen in der gleichen Ebene (Seite 298)

Daten von Bausteinen aus einer niederprioren Ebenen (Seite 300)

Daten aus Bausteinen aus einer höherprioren Ebene (Seite 303)

6.8.7.2 Datenaustausch zwischen Bausteinen in der gleichen Ebene

Beschreibung

Haben Sie die Bausteine in der gleichen Ablaufebene miteinander verschaltet, werden die Daten gemäß der Ablaufreihenfolge der Bausteine übertragen. Dadurch ist die Aktualität des Wertes am Eingang von der Ablaufreihenfolge bestimmt.

Wenn Sie die Ablaufreihenfolge gemäß dem Datenfluss festgelegt haben, ergibt sich in der Ebene keine zusätzliche Totzeit.

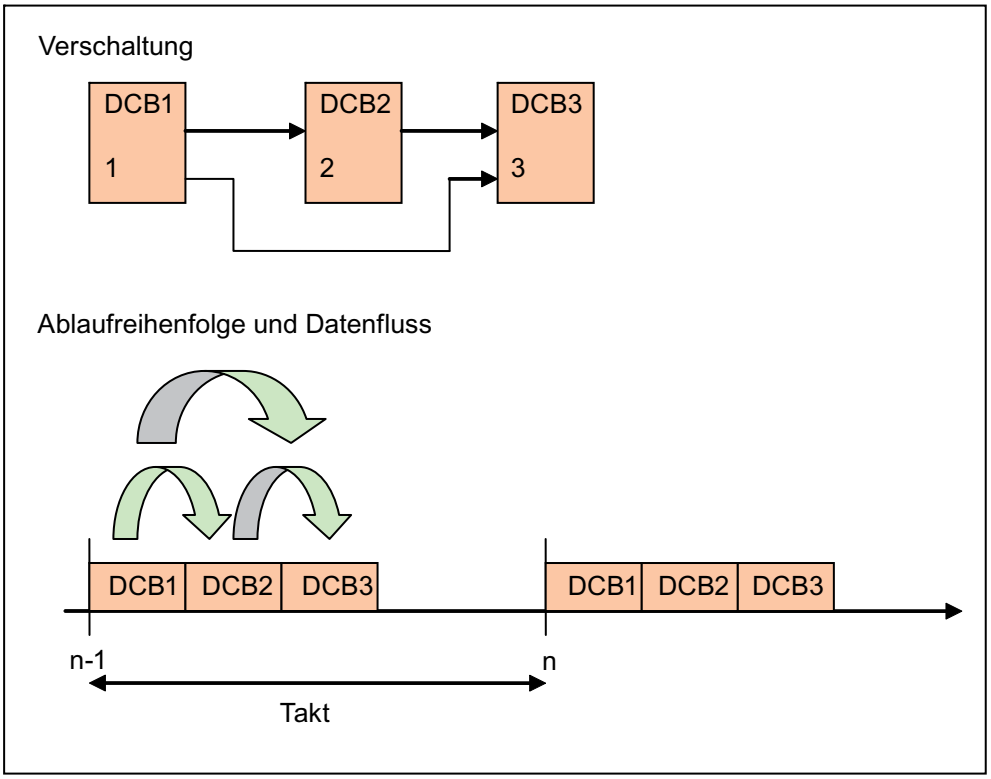


Bild 6-66 Verschaltungsfolge und Ablaufreihenfolge sind gleich

Entspricht die Ablaufreihenfolge der Bausteine in einer Ebene nicht dem Datenfluss, ergeben sich zusätzliche Totzeiten, da eingangsseitig auf Werte aus dem vorhergehenden Takt zugegriffen wird.

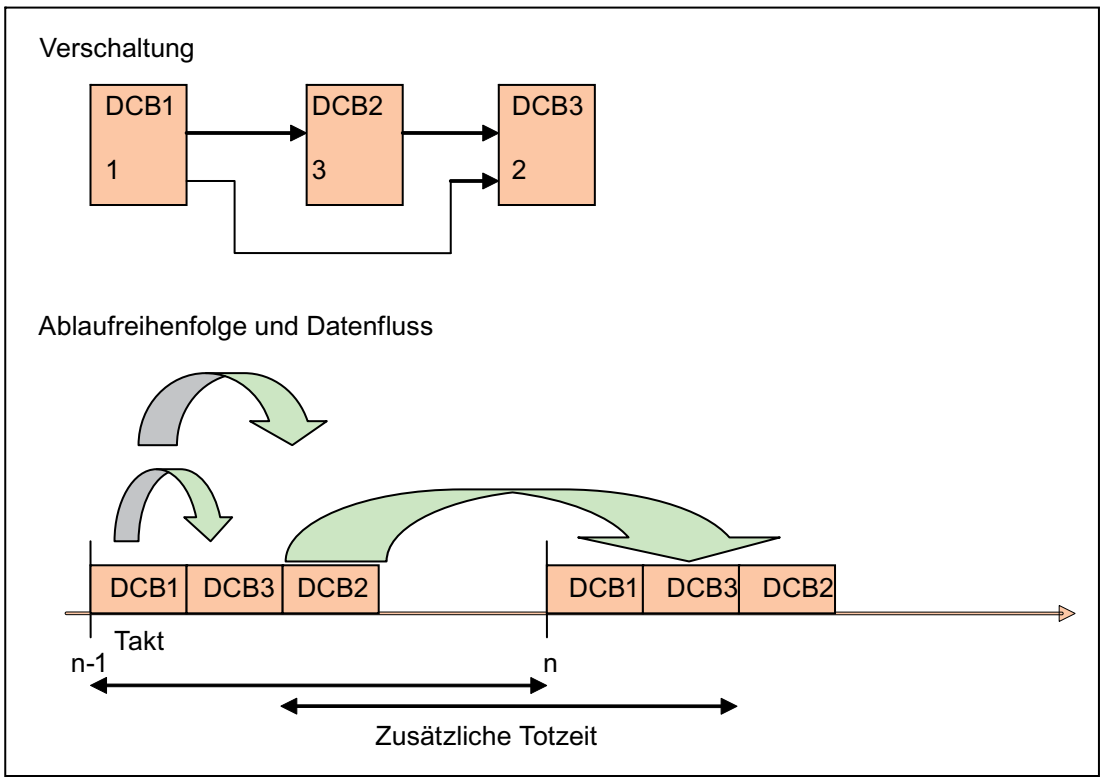


Bild 6-67 Verschaltungsfolge und Ablaufreihenfolge sind unterschiedlich

Um Totzeiten zu optimieren, orientieren Sie die Ablaufreihenfolge stets am Datenfluss.

6.8.7.3 Daten von Bausteinen aus einer niederprioren Ebenen

Beschreibung

Alle Bausteine in einer Ablaufebene müssen berechnet werden, bevor deren ausgangseitige Werte Bausteinen in einer anderen Ablaufebene eingangsseitig zur Verfügung stehen. Der Zugriff auf die ausgangsseitigen Werte aus einer höherprioren Ebene, erfolgt in einem dedizierten Takt (Übernahmetakt), um die Äquidistanz von Eingangswerten zu gewährleisten. Die folgende Grafik veranschaulicht die äquidistante Wertübernahme in einer höherprioren Ebene.

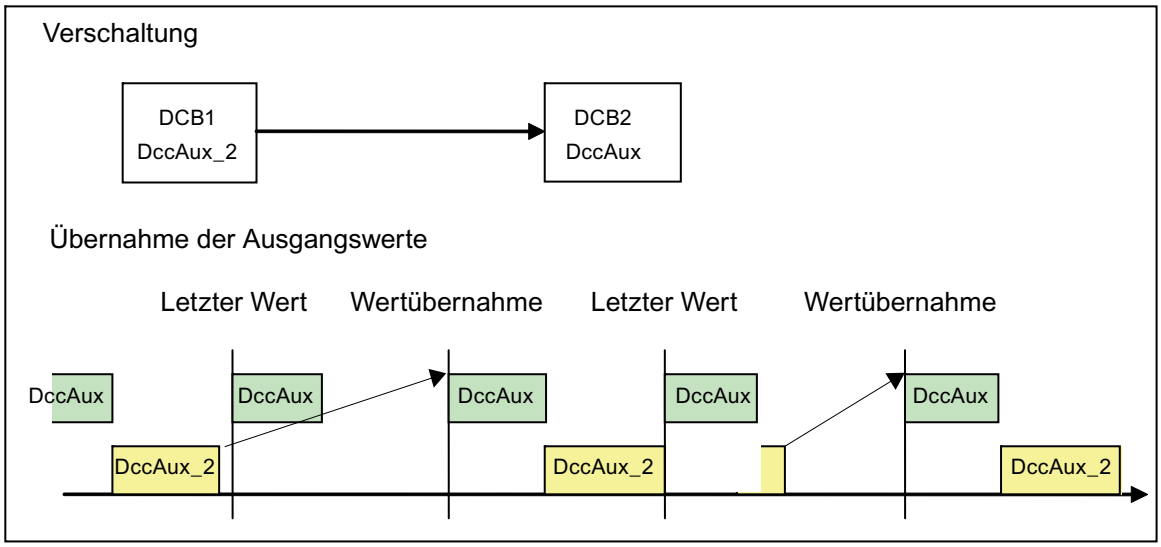


Bild 6-68 Beispiel für den Datenaustausch aus einer niederpriorer Ebene

Das Taktverhältnis zwischen den Ebenen ist 1:2. Die Task in der höherprioren Ebene übernimmt den Wert immer in jedem zweiten Takt, unabhängig davon, ob die niederpriore Task schon im vorhergehenden Takt zu Ende gerechnet wurde.

Ebenenüberlauf der niederprioren Ebene

Kommt es zu einem Ebenenüberlauf der niederprioren Ebene, sind zum Übernahmetakt die aktuellen Werte noch nicht berechnet. In diesem Fall wird eingangsseitig bis zu dem nächsten Übernahmetakt mit dem alten Wert gerechnet.

Die folgende Grafik verdeutlicht das Überlaufverhalten bei einem Taktverhältnis von 1:2. Es kommt zu einem Ebenenüberlauf von T2. Damit werden die alten Werte beim Übernahmezeitpunkt (Takt 1) in die höherpriore Task übernommen. Beim nächsten Übernahmezeitpunkt, sprich zwei Takte später, wird ein neuer Wert übernommen.

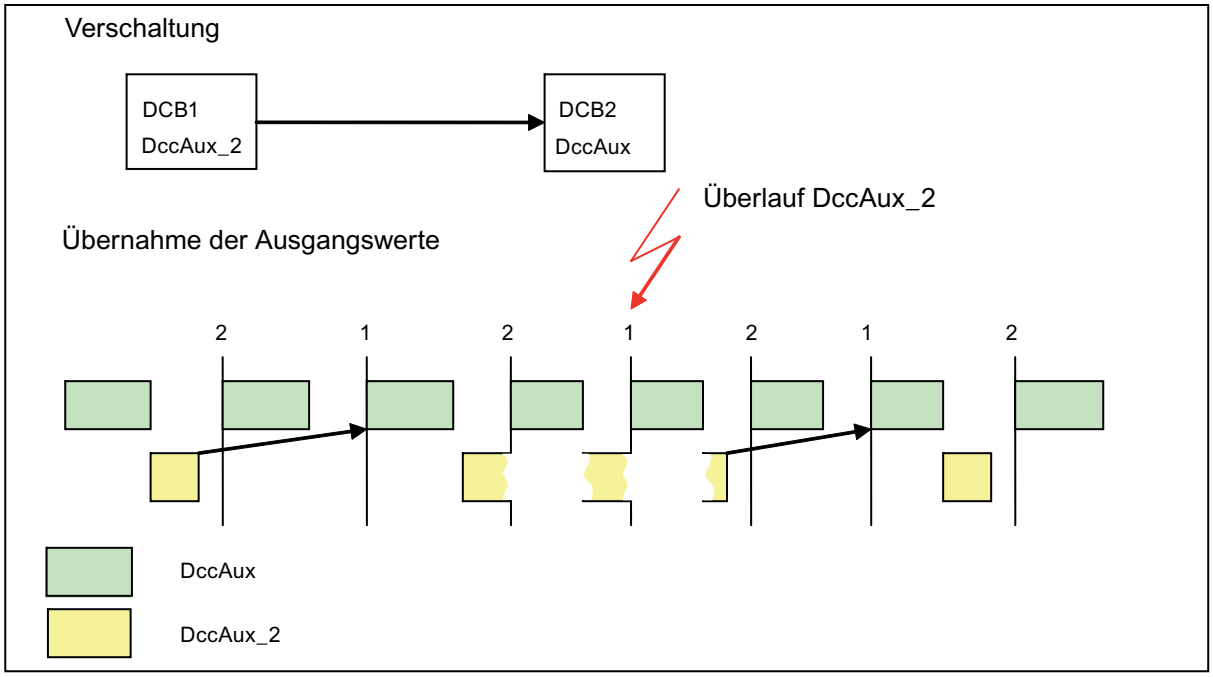


Bild 6-69 Datenaustausch aus niederpriorer Ebene mit Überlauf

Das beschriebene Überlaufszenario gilt nur für ein ideales System. In einer realen Anwendung laufen in einer Ebene neben der DCC-Task auch noch Anwendertasks und Systemtasks.

Beispielhafte Berücksichtigung von Anwenderprogrammen und Systemtasks

Kommt es zum Überlauf während der Bearbeitung des iposynchronen Anwenderprogramms oder der IPO-Systemtask, wurden die Werte bereits in der ipoDcc-Task berechnet und können in die höherpriore Ebene übernommen werden. Allerdings wird im nächsten Takt kein neuer Wert berechnet. Führt die ipoDcc-Task schon zu einem Überlauf, wird erst im übernächsten Takt wieder ein korrekter Wert übernommen. In den zwei Takten dazwischen wird jeweils der Wert um einen Takt verzögert übernommen.

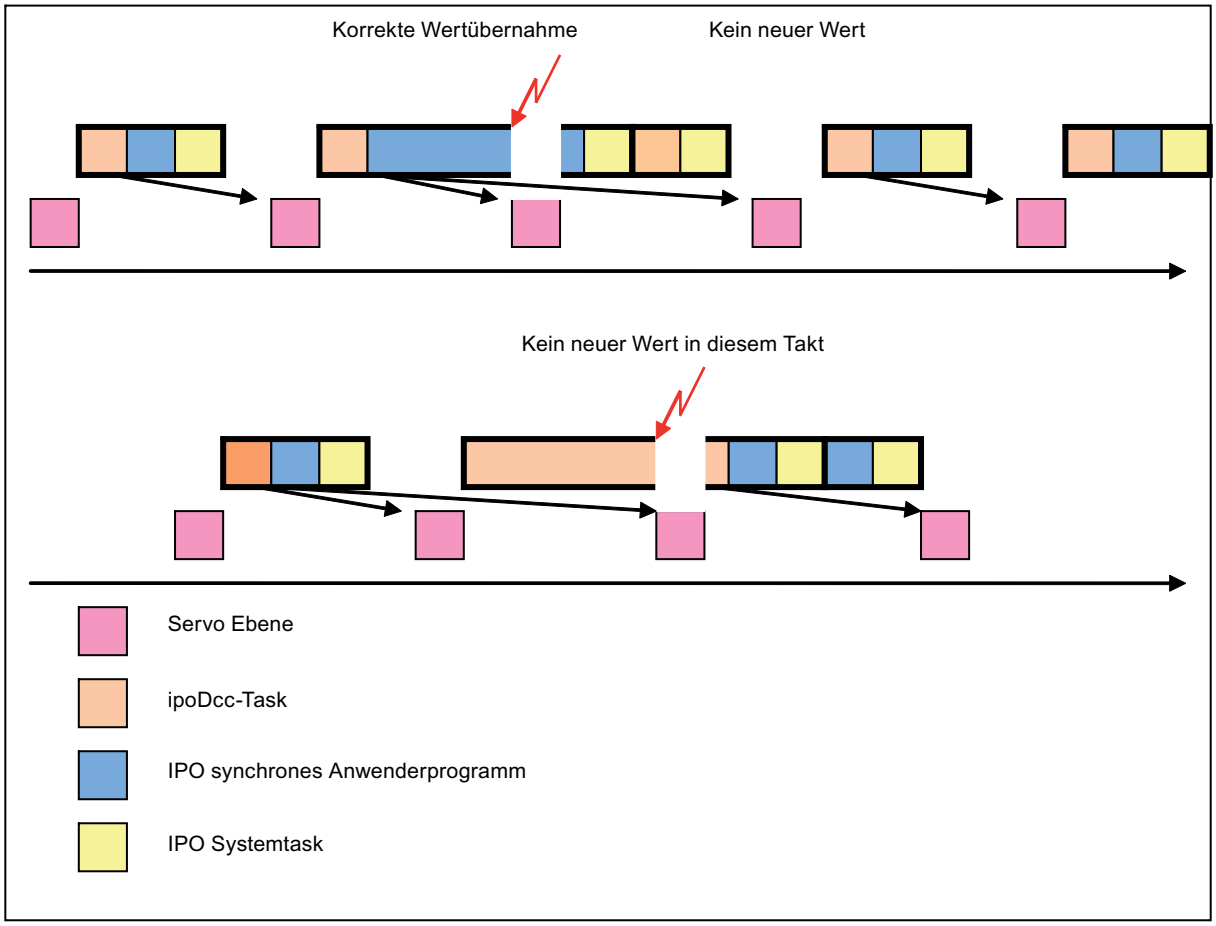


Bild 6-70 Überlauf im Ablaufkontext mit anderen Tasks

Hinweis

Um eine äquidistante Datenübergabe zwischen der ipoDcc in der IPO-Ebene und der servoDcc in der Servo-Ebene zu gewährleisten, darf die ipoDcc in der IPO-Ebene für sich betrachtet nicht schon zu einem Ebenenüberlauf führen. Ist dies der Fall, steht der höherprioren Task bei der Übernahme der Werte aus der niederprioren Ebene keine aktualisierten Werte zur Verfügung. Aktualisierte Werte werden dann erst nach dem nächsten Zyklus der niederprioren Ebenen übernommen.

6.8.7.4 Daten aus Bausteinen aus einer höherprioren Ebene

Beschreibung

Für den Zugriff auf die ausgangsseitigen Werte aus einer niederprioren Ebene, werden die Werte aus einem dedizierten Takt der höherprioren Ebene übernommen, um die Äquidistanz von Eingangswerten zu gewährleisten.

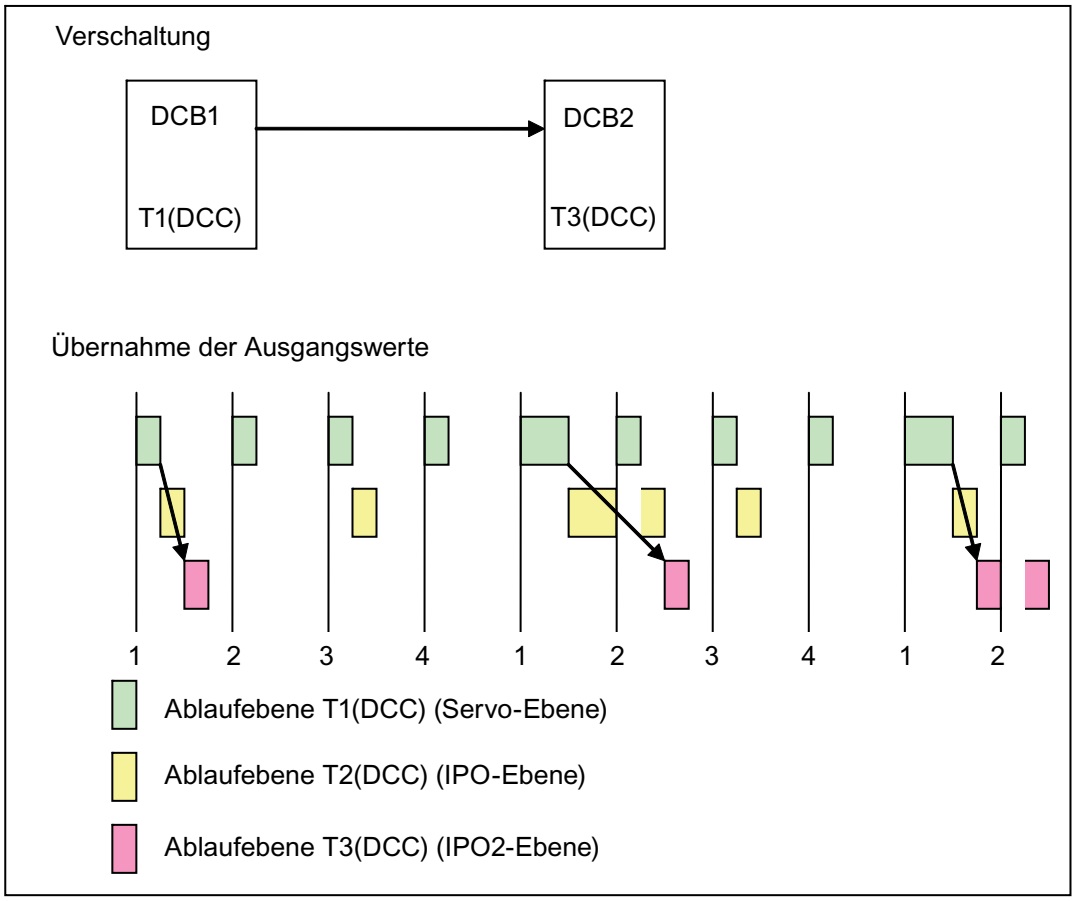


Bild 6-71 Datenübernahme aus höherpriorer Ebene

6.8.8 Verschaltung von Bausteinen mit Variablen

6.8.8.1 Verschaltung mit Variablen

Überblick über Verschaltungsmöglichkeiten mit Variablen

Neben der Möglichkeit Bausteine miteinander zu verschalten, kann ein PIN auch auf eine Variable verschaltet werden. Sie können so eine Verbindung mit der Peripherie, den Technologie-Objekten, einem Anwenderprogramm und einem HMI herstellen.

Folgende Variablen können mit Bausteinen verschaltet werden:

- I/O-Variablen.
- Systemvariablen
- Folgende Anwendervariablen können mit DCC verschaltet werden:
  - Geräteglobale Variablen
  - Variablen im Interface einer Quelle

Verschaltung von Technologieobjekten

- Verschaltung über Systemvariablen und zyklisches Interface
- Systemfunktionen der TOs können nicht direkt aus DCC-Plänen aufgerufen werden.

PIN mit Variablen verschalten

Die verschiedenen Ein- bzw. Ausgänge eines Bausteins können Sie im DCC-Editor verschalten. Detaillierte Informationen dazu finden Sie in der Beschreibung des DCC-Editors.

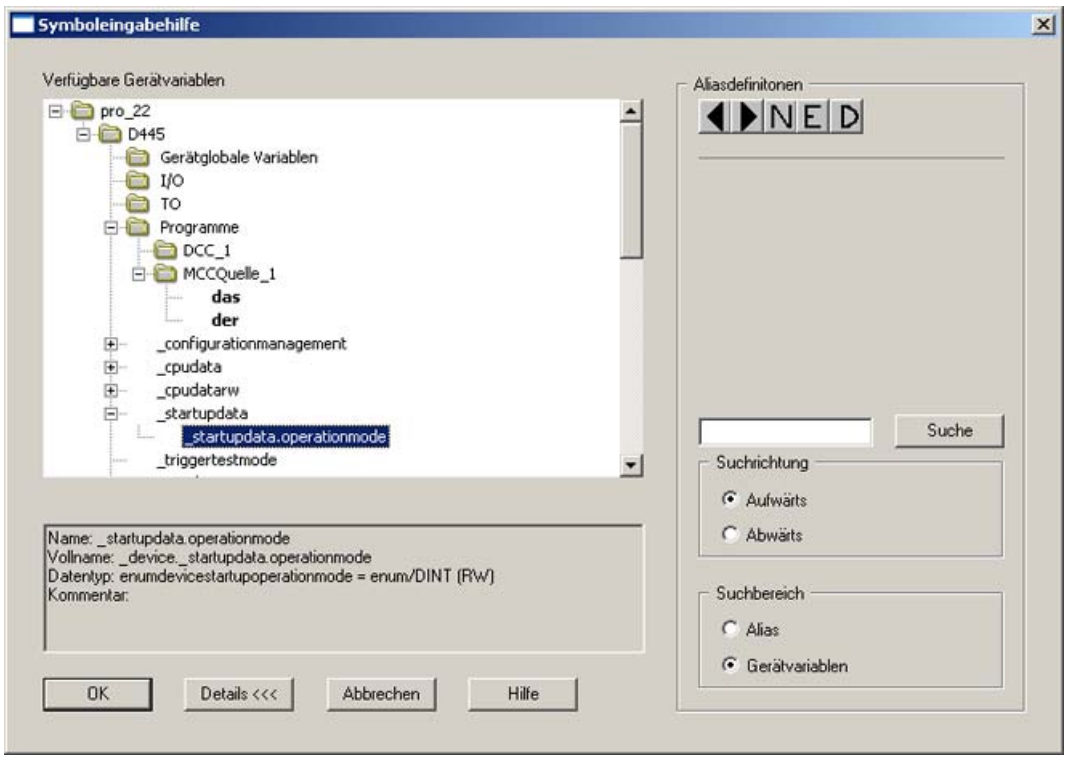


Bild 6-72 Verschaltung von Anwender-Variablen im DCC-Editor

Hinweis

Es gibt keine Einschränkungen, wie oft ausgangsseitig auf eine Variable verschaltet werden kann. Es ist immer der Ausgangswert des zuletzt gerechneten Bausteins wirksam.

6.8.8.2 Verhalten bei FPU-Exceptions

Beschreibung

Folgendes Verhalten bei FPU (Floating Point Unit) Exceptions zeigen DCC Tasks:

Exception	Beschreibung	Reaktion SIMOTION
Invalid Operation	Ungültige Operation (z. B. 0/0, INF/INF, INF - INF, SQRT(-1)	Gerät wechselt in Betriebszustand STOP
Devision by Zero	Division durch Null	Gerät wechselt in Betriebszustand STOP
Overflow	Das absolute Ergebnis einer Operation ist größer als abs(+/-- Nmax)	Gerät wechselt in Betriebszustand STOP
Underrun	Das absolute Ergebnis einer Operation ist kleiner als abs(+/-- Nmax)	Das Ergebnis ist 0
Inexact	Ergebnis lässt sich nicht exakt darstellen.	Das Ergebnis wird gerundet.

Siehe Fehler bei Operationen mit Gleitpunktzahlen (FPU-Exceptions) (Seite 147).

6.9 Einbindung von Antriebsperipherie

6.9.1 Antriebsperipherie symbolisch zuordnen

Beschreibung

Antriebsperipherie kann ab V4.2 symbolisch den I/O-Variablen zugeordnet werden. Weitere Informationen finden Sie unter Symbolische Zuordnung von I/O-Variablen auf I/O-Klemmen (Seite 92).

6.9.2 Terminal Modules TM15 und TM17 High Feature

Beschreibung

Die Terminal Modules TM15 und TM17 High Feature werden vom Motion Control System SIMOTION taktsynchron ausgewertet. Detaillierte Angaben zum Timing können dem Inbetriebnahmehandbuch Terminal Modules TM15 / TM17 High Feature entnommen werden.

6.9.3 Terminal Modules TMxx, TB30 und Onboard I/Os auf SIMOTION D bzw. CX32/CX32-2 und CU3xx/CU3xx-2

Beschreibung

TM31, TM41, TM15 DI/DO, TB30 und die Onboard I/Os werden bei SINAMICS freilaufend (nicht taktsynchron) betrieben.

Der Aktualisierungszyklus wird durch die folgenden Antriebsparameter festgelegt und ist defaultmäßig auf 4 ms eingestellt:

- p4099 für TM31, TM41 und TM15 DI/O bzw.
- p0799 für TB30 und die Onboard I/O

Die Zugriffe erfolgen in einer SINAMICS-Hintergrundtask (z.B. alle 4 ms). Der Zugriffszeitpunkt auf die Ein- und Ausgaben innerhalb des eingestellten Aktualisierungszyklus kann von Zyklus zu Zyklus unterschiedlich sein - d.h. es ist lediglich sichergestellt, dass innerhalb eines jeden Zyklus eine Aktualisierung erfolgt mit einer möglichen Schwankungsbreite entsprechend dem Aktualisierungszyklus. Über die Einstellung von p4099 bzw. p0799 kann dies beeinflusst werden.

Hinweis

Das folgende Bild dient nur zur schematischen Darstellung der Zeiten und soll nicht über den absoluten Betrag Auskunft geben.

Für die I/O ergibt sich folgendes Timing:

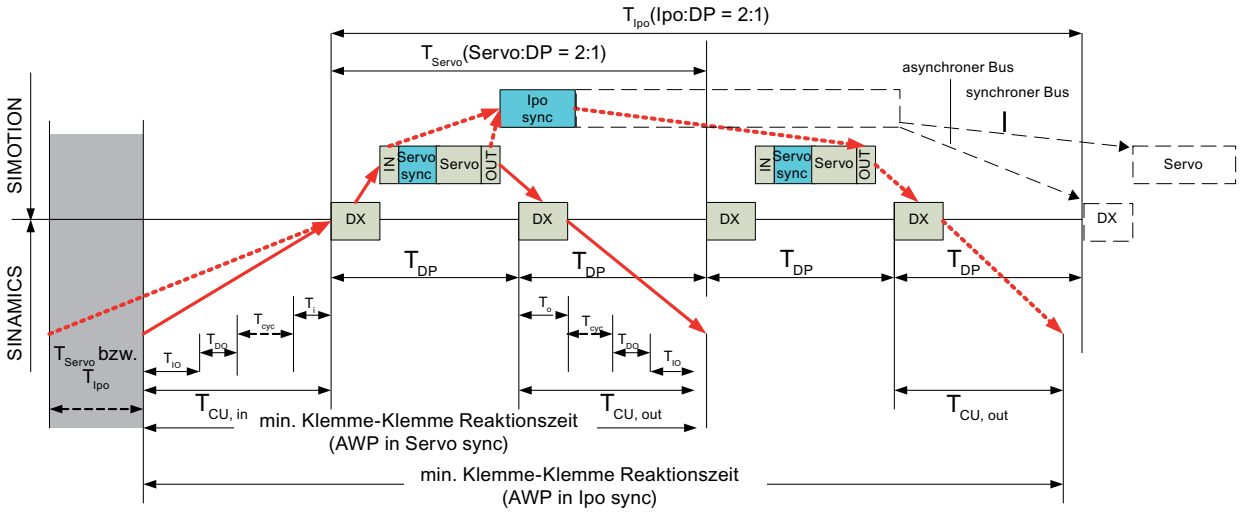


Bild 6-73 I/O Timing

Legende

- T_{DP}: Taktzyklus PROFIBUS (siehe Einstellung in HW Konfig)
- T_i : Latchzeit der Eingänge bei taktsynchronem PROFIBUS (siehe Einstellung in HW Konfig für den Antrieb)
- T_o: Verzögerung der Ausgänge bei taktsynchronem PROFIBUS (siehe Einstellung in HW Konfig für den Antrieb)
- T_{IO}: Baugruppenspezifische Signalverzögerung (entspricht einem DRIVE-CLiQ Zyklus + der Eingangsverzögerungszeit bei Digitaleingängen bzw. der lastabhängigen Ausgangsverzögerungszeit bei Digitalausgängen)
- T_{cyc}: I/O-Aktualisierungszyklus im SINAMICS (Parameter p4099 bzw. p0799)
- Servo: Servo-Takt SIMOTION; Vielfaches von T_{DP} (siehe Takteinstellungen bei SIMOTION)
- IPO: IPO-Takt SIMOTION, Vielfaches vom Servo-Takt (siehe Takteinstellungen bei SIMOTION)

- T_{DQ}: Taktzyklus DRIVE-CLiQ (entspricht zu V4.1 immer dem Stromregler-Takt)
- AWP: Anwenderprogramm
- Ausgabezeit auf der CU: $T_{CU\_out} = T_O + T_{cyc} + T_{DQ} + T_{IO}$
- Einlesezeit auf der CU: $T_{CU\_in} = T_i + T_{cyc} + T_{DQ} + T_{IO}$

In der Abbildung sind alle Zeiten angegeben, welche die Klemme-Klemme-Reaktionszeit beeinflussen. Für die gestrichelten Zeiten sind die worst case Werte anzusetzen, d.h.:

- Der eingestellte Aktualisierungszyklus T_{cyc} (= Maximalzeit für den Zugriffszeitpunkt)
- Die Zeit T_{servo} bzw. T_{IPO} im grauen Bereich. Abhängig vom Zeitpunkt des Auftretens des Eingangsereignisses kann die maximale Klemme-Klemme-Reaktionszeit bis zu einem Servo- bzw. IPO-Takt länger sein als die minimale Klemme-Klemme-Reaktionszeit.

Der Bereich, in dem ein Eingangsereignis erfasst wird und beim nächsten Aufruf des Anwenderprogrammes (AWP) bearbeitet wird, ist durch den grauen Bereich dargestellt. Die Größe dieses Bereichs hängt davon ab, ob das Anwenderprogramm in der servosynchronen Task oder IPO synchronen Task ausgeführt wird.

Hinweis

Bitte beachten Sie, dass eine Reduzierung des Aktualisisierungszyklus zu einer höheren Auslastung der SINAMICS-Regelung bzw. der SIMOTION D führt, wodurch unter Umständen sich das Mengengerüst an Antrieben, Terminal Modules etc. reduzieren kann!

Aus dem SIMOTION Anwenderprogramm kann auf I/O-Komponenten im SINAMICS Antriebsgerät zugegriffen werden:

- Ab V4.2 ist dies durch eine symbolische Zuordnung der I/O-Variable zur SINAMICS Peripherie möglich. Notwendige BICO-Verschaltungen und Telegramme werden durch SIMOTION automatisch konfiguriert.
- In V4.1 ist die Verwendung durch spezielle Telegramme (39x) möglich.
- In früheren Versionen werden die I/O-Komponenten mittels BICO-Verschaltung als I/O-PZD (Prozessdaten) in das PROFIdrive-Telegramm eingefügt (siehe SIMOTION D Inbetriebnahme- und Montagehandbuch, Kapitel "Telegrammprojektierung für Onboard I/O und Antriebsobjekte").

Das Timing ist abhängig davon, auf welche I/O-Komponente zugegriffen wird. Nachfolgende Tabelle gibt die maximalen Verzögerungszeiten an. Für die Ermittlung der Klemme-Klemme-Zeiten sind die Werte für "Klemme ->Anwenderprogramm" und "Anwenderprogramm -> Klemme" zu addieren.

Hinweis

Die gestrichelten Bereiche in der Grafik **IO Timing** für die IPOsynchrone Task gelten nur für PROFINET IO. Analog dazu sind in der folgenden Tabelle in der Spalte **AWP in IPOsynchroner Task** die Zeiten für PROFNIET IO dargestellt. Für PROFIBUS DP müssen Sie jeweils in der Zeile **OUT** T_{IPO} durch T_{Servo} ersetzen. Der gestrichelte Bereich ist für PROFIBUS nicht relevant, da dort der Servo nicht über mehrere Buszyklen laufen kann.

			Maximale Verzögerungszeiten		
I/O Modul			**AWP in servosynchroner Task**	**AWP in IPOsynchroner Task**	**T_{cyc}**
TM31, TM41, TM15 DI/O	Out	AWP -> Klemme	$T_{DP} + T_{CU\_out}$	$T_{IPO} + T_{DP} + T_{CU\_out}$	p4099
	In	Klemme -> AWP	$T_{Servo} + T_{CU\_in}$	$T_{IPO} + T_{CU\_in}$	
TB30	Out	AWP -> Klemme	$T_{DP} + T_o + T_{cyc} + T_{IO}$	$T_{IPO} + T_{DP} + T_o + T_{cyc} + T_{IO}$	p0799[3)]
	In	Klemme -> AWP	$T_{Servo} + T_i + T_{cyc} + T_{IO}$	$T_{IPO} + T_i + T_{cyc} + T_{IO}$	
Onboard I/O SIMOTION D, CX32, CX32-2, CU3xx, CU3xx-2	Out	AWP -> Klemme[1)]	75 µs	75 µs	p0799[3)]
		AWP -> Klemme[2)]	$T_{DP} + T_o + T_{cyc}$	$T_{IPO} + T_{DP} + T_o + T_{cyc}$	
	In	Klemme -> AWP	$T_{Servo} + T_i + T_{cyc}$	$T_{IPO} + T_i + T_{cyc}$	

1) Zugriff auf integrierte Antriebe SIMOTION D in Verbindung mit Standardtelegramm 39x für das DO1. Ohne Standardtelegramm gilt das Timingverhalten wie für CX32 bzw. CX32-2.

2) Zugriff auf CX32, CX32-2 sowie über externen PROFIBUS bzw. PROFINET IO auf CU3xx und CU3xx-2

3) Bei taktsynchronem Betrieb ist Tcyc = max (T_{DP}, p0799)

Weitere Informationen

Weitere Informationen zu diesem Thema finden Sie auch:

- In den Inbetriebnahmehandbüchern zu SIMOTION D410 und D4x5/D4x5-2
- in einem FAQ auf der Utilities & Applications CD unter FAQs
- im Internet unter der Adresse http://support.automation.siemens.com/WW/view/de/27585482.

Programmieren Ablaufsystem/Tasks/Systemtakte 7

7.1 Ablaufsystem

7.1.1 Allgemeines zum Ablaufsystem

Im Ablaufsystem des SIMOTION Geräts werden alle Programme abgearbeitet. Das Ablaufsystem stellt eine Reihe von Ablaufebenen mit unterschiedlichen Ablaufeigenschaften bereit.

Programme müssen deshalb zu ihrer Ausführung den Ablaufebenen zugeordnet werden. Hierzu werden die Programme der Quellen einer oder mehreren Tasks zugeordnet.

So wird die zeitliche Abfolge festgelegt, in der sie abgearbeitet werden.

Hinweis

Bevor Programme den Ablaufebenen zugeordnet werden können, müssen die ST/MCC/KOP/FUP-Quellen übersetzt sein.

7.1.2 Ablaufebenen und Tasks

Die Ablaufebenen legen die zeitliche Abfolge von Programmen im Ablaufsystem fest. Hierzu enthält jede Ablaufebene eine oder mehrere Tasks.

Eine Task stellt den Ablaufrahmen für die Programme zur Verfügung. Jeder Task können Sie ein oder mehrere Anwenderprogramme zuordnen und deren Reihenfolge innerhalb der Task festlegen.

Neben diesen Anwenderprogramm-Tasks sind auch mehrere System-Tasks vorhanden, auf deren Inhalt und Ablaufreihenfolge Sie keinen Einfluss haben.

Die Tabelle zeigt die Ablaufebenen mit ihren Tasks, die den Anwenderprogrammen zur Verfügung stehen (Anwenderprogramm-Tasks).

Mit Tasksteuerbefehlen können Sie das Ablaufsystem beeinflussen.

Tabelle 7- 1 Ablaufebenen in SIMOTION

Ablaufebene	Beschreibung
Zeitgesteuert	Zyklische Tasks, sie werden nach Abarbeiten der zugeordneten Programme automatisch neu gestartet.
• SynchronousTasks	Tasks werden synchron zum angegebenen Systemtakt periodisch gestartet. • ServoSynchronousTask: synchron zum Lageregler-Takt (ab Version 4.0 des SIMOTION Kernels) • ServoSynchronousTask_fast: synchron zum Servo_fast (ab Version 4.2 des SIMOTION Kernels) optional • IPOsynchronousTask: synchron zum Interpolatortakt IPO • IPOsynchronousTask_fast: synchron zum Interpolatortakt IPO_fast (ab Version 4.2 des SIMOTION Kernels) optional • IPOsynchronousTask_2: synchron zum Interpolatortakt IPO_2 • Synchrone Tasks für Technologiepaket Tcontrol: – PWMsynchronousTask: synchron zum PWM-Takt InputSynchronousTask_1: synchron zum Takt Input1 – InputSynchronousTask_2: synchron zum Takt Input2 – PostControlTask_1: synchron zum Takt Control1 – PostControlTask_2: synchron zum Takt Control2
• TimerInterruptTasks	Tasks werden in festem Zeitraster periodisch gestartet. Das Zeitraster muss ein Vielfaches des Interpolatortaktes IPO sein.
Interrupts	Sequentielle Tasks; sie werden nach Start einmalig abgearbeitet und dann beendet.
• SystemInterruptTasks	Sie werden bei Eintreten eines Systemereignisses gestartet: Ausführliche Beschreibung • ExecutionFaultTask: Fehler beim Verarbeiten eines Programms • PeripheralFaultTask: Fehler an der Peripherie • TechnologicalFaultTask: Fehler am Technologischen Objekt • TimeFaultBackgroundTask: Zeitüberlauf der BackgroundTask • TimeFaultTask: Zeitüberlauf einer TimerInterruptTask
• UserInterruptTasks	Sie werden bei Eintreten eines anwenderdefinierten Ereignisses flankengetriggert gestartet.
Round-Robin	MotionTasks und BackgroundTask teilen sich die freie Zeit, die nach Abarbeiten der höherpriorigen System- und Anwendertasks bleibt. Das Verhältnis zwischen beiden Ebenen ist parametrierbar.
• MotionTasks	Sequentielle Tasks; sie werden nach Start einmalig abgearbeitet und dann beendet. Der Start erfolgt: • Explizit über einen Tasksteuerbefehl in einem Programm, das einer anderen Task zugeordnet ist. • Automatisch nach Erreichen des Betriebszustandes RUN, wenn bei der Taskkonfiguration das entsprechende Attribut gesetzt wurde. Die Priorität einer MotionTask kann mit der Systemfunktion WAITFORCONDITION temporär erhöht werden (siehe **MotionTask auf Erfüllung einer Bedingung warten lassen**).

Ablaufebene	Beschreibung
• BackgroundTask	Zyklische Task; sie wird nach Abarbeiten der zugeordneten Programme automatisch neu gestartet; die Zykluszeit der Task ist abhängig von der Laufzeit.
StartupTask	Wird beim Übergang des Betriebszustandes STOP bzw. STOP U zu RUN einmalig ausgeführt. SystemInterruptTasks werden über ihr auslösendes Systemereignis gestartet.
ShutdownTask	Wird beim Übergang des Betriebszustandes RUN zu STOP bzw. STOP U einmalig ausgeführt. Der Betriebszustand STOP bzw. STOP U wird folgendermaßen erreicht: • entsprechende Stellung des Betriebsartenwahlschalters • entsprechende Systemfunktion des SIMOTION Geräts • Alarm (Störung) mit entsprechender Fehlerreaktion SystemInterruptTasks und PeripheralFaultTasks werden über ihr auslösendes Systemereignis gestartet.
Zum Verhalten sequentieller und zyklischer Tasks: • Bei der Initialisierung lokaler Programmvariablen siehe Tabelle Initialisierung lokaler Programmvariablen in Abhängigkeit vom Ablaufverhalten der Task (Seite 315). • Bei Verarbeitungsfehlern im Programm siehe Kapitel Verarbeitungsfehler in Programmen (Seite 146). Mögliche Zugriffe auf das Prozessabbild und I/O-Variablen siehe Kapitel *Wichtige Eigenschaften von Direktzugriff und Prozessabbild*, das Sie in den verschiedenen Programmierhandbüchern finden.	

7.1.3 Startreihenfolge der Tasks

Nach Ablauf der StartupTask ist der Betriebszustand RUN erreicht.

Dann werden gestartet:

- die SynchronousTasks
- die TimerInterruptTasks
- die BackgroundTask
- die MotionTasks mit Startup-Attribut.

Hinweis

Die Reihenfolge, in der diese Tasks nach Erreichen des Betriebszustandes RUN erstmals gestartet werden, richtet sich nicht nach den Prioritäten der Tasks.

7.1.4 Ablaufsystem konfigurieren

7.1.4.1 Festlegungen beim Konfigurieren

Beim Konfigurieren des Ablaufsystems ordnen Sie den Tasks die Programme zu, die in der jeweiligen Task ablaufen sollen. Dadurch legen Sie u. a. fest:

- Die Priorität, mit der die Programme ablaufen,
- Das Ablaufverhalten (sequentiell, zyklisch),
- Das Initialisierungsverhalten lokaler Programmvariablen.

Die Zuordnung eines Programms zu einer oder mehreren Tasks kann nur nach dem Übersetzen stattfinden und muss vor dem Laden des Programms ins Zielsystem geschehen.

Nachdem Sie ein Programm einer oder mehreren Tasks zugeordnet haben, können Sie die Verbindung zum Zielsystem herstellen, das Programm in das Zielsystem laden und es anschließend starten.

Siehe auch

Ablaufsystem konfigurieren (Seite 236)

Programme den Ablaufebenen/Tasks zuweisen (Seite 236)

7.1.4.2 Programme den Tasks zuordnen

In Kapitel *Beispielprogramm ausführen* haben Sie mit dem Beispielprogramm bereits eine Taskzuordnung vorgenommen, hier in Kurzform das Vorgehen:

1. Markieren Sie im Projektnavigator das SIMOTION Gerät und wählen Sie das Menü **Zielsystem > Ablaufssystem konfigurieren**.

   Das Konfigurationsfenster für das Ablaufsystem des SIMOTION Geräts wird geöffnet.
2. Markieren Sie die zu konfigurierende Task.
3. Wählen Sie das Register **Programmzuordnung** und ordnen Sie die gewünschten Programme der Task zu.
4. Wählen Sie das Register Taskkonfiguration und nehmen Sie die dort ggf. weitere Einstellungen vor, z. B.:
   - im Feld Bereichsgrenze für dynamische Daten die Größe des Lokaldatenstacks (siehe Kapitel *Speicherbereiche der Variablentypen*)
   - Fehlerreaktion bei Programmfehler (siehe Kapitel *Verarbeitungsfehler in Programmen*)
   - Zeitüberwachungen zyklischer Tasks
   - Startverhalten von MotionTasks

Siehe auch

Programme den Ablaufebenen/Tasks zuweisen (Seite 236)

7.1.5 Einfluss des Ablaufverhaltens einer Task auf die Variableninitialisierung

7.1.5.1 Zeitpunkt für die Initialisierung lokaler Programmvariablen

Das Ablaufverhalten der Task (sequentiell oder zyklisch) bestimmt die Initialisierung der lokalen Variablen der zugeordneten Programme. Die Ausführungen gelten analog für Instanzen von Funktionsbausteinen, die in den Programmen als lokale Variablen deklariert wurden.

Eine Zusammenfassung aller Variablenarten und den Zeitpunkt ihrer Initialisierung erhalten Sie im Kapitel *Zeitpunkt der Variableninitialisierung*, das Sie in den verschiedenen Programmierhandbüchern finden.

Tabelle 7- 2 Initialisierung lokaler Programmvariablen in Abhängigkeit vom Ablaufverhalten der Task

Ablaufverhalten	Tasks	Initialisierung lokaler Programmvariablen
Sequentiell (nicht zyklisch)	MotionTasks, UserInterruptTasks, SystemInterruptTasks, StartupTask, ShutdownTask.	Sequentielle Tasks werden nach dem Start einmal durchlaufen und dann beendet. Bei jedem Start werden alle lokalen Variablen der zugeordneten Programme initialisiert. Die Zeit für die Dateninitialisierung geht in die Laufzeit der Task ein. Verhalten mit Compilerschalter "Einmalige Programmdateninitialisierung": Die lokalen Variablen der zugeordneten Programme werden nur einmalig (beim Download) initialisiert. Siehe Download im RUN von geänderten Quellen (Seite 535) und Einfluss des Compilers auf die Variableninitialisierung (Seite 317) .
Zyklisch	BackgroundTask, SynchronousTasks, TimerInterruptTasks	Zyklische Tasks werden nach ihrer Beendigung automatisch neu gestartet; die Werte statischer lokaler Variablen der zugeordneten Programme (in VAR / END_VAR deklariert) bleiben dabei erhalten. Statische Variablen werden nur einmalig beim Übergang von STOP auf RUN initialisiert. Temporäre Variablen (in VAR_TEMP / END_VAR deklariert) werden bei jedem Start der Task initialisiert. Die einmalige Initialisierung statischer Variablen geht nicht in die Laufzeit der Task ein. Verhalten mit Compilerschalter "Programminstanzdaten nur einmalig anlegen": Die lokalen Variablen der zugeordneten Programme werden einmalig (beim Download) instantiiert. Siehe Download im RUN von geänderten Quellen (Seite 535)und Einfluss des Compilers auf die Variableninitialisierung (Seite 317) .

Für Informationen zur Initialisierung bei einem STOP - RUN - Übergang, siehe Initialisierung von Daten bei einem STOP - RUN - Übergang (Seite 542) .

7.1.5.2 Zuweisen von Anfangswerten an Unit-Variablen

Unit-Variablen und geräteglobale Variablen werden beim Übergang vom Betriebszustand STOP bzw. STOPU nach RUN nicht initialisiert (siehe Kapitel *Zeitpunkt der Variableninitialisierung*, das Sie in den verschiedenen Programmierhandbüchern finden).

Wenn Sie diesen Variablen dennoch Anfangswerte zuweisen wollen, verwenden Sie hierzu die StartupTask. Für Informationen zur Initialisierung bei einem STOP - RUN - Übergang und dem Pragma BlockInit_OnDeviceRun siehe auch Initialisierung von Daten bei einem STOP - RUN - Übergang (Seite 542).

ACHTUNG
Nach dem Übergang in den Betriebszustand RUN ist die Startreihenfolge der Tasks nicht festgelegt (siehe **Startreihenfolge der Tasks**). Bei Verwendung einer anderen Tasks als der StartupTask ist eine korrekte Anfangswertzuweisung nicht gewährleistet.

7.1.5.3 Mehrere VAR_GLOBAL, VAR_GLOBAL RETAIN Blöcke verwenden

Beschreibung

Sie können mehrere VAR_GLOBAL, VAR_GLOBAL RETAIN Blöcke im Interface- und Implementation-Bereich einer UNIT (ab V4.1) anlegen.

Im Interface- und im Implementierungsbereich können Sie mehrere Deklarationsblöcke in beliebiger Reihenfolge angeben. Jeder dieser Blöcke wird getrennt versioniert. Änderungen innerhalb eines Blockes (ob direkt oder indirekt über Datentypänderung) führen damit zu einer Neuinitialisierung diese Blocks beim Nachladen. Damit wirkt der (RETAIN-)Datenerhalt blockweise.

Neue Blöcke können am Ende hinzugefügt und die geänderten Quellen im RUN nachgeladen werden, ohne die vorhandenen Datenblöcke zu beeinflussen. Wird in einem Quellabschnitt (Aussage gilt jeweils für Interface und Implementation getrennt) ein Block vor einem bereits vorhandenen Block gleicher Art (RETAIN oder nicht RETAIN) eingefügt, ändern sich alle nachfolgenden Blöcke und werden beim Download neu initialisiert. Ein Download im RUN nach so einer Änderung ist demzufolge nicht möglich.

Über die Funktionen _saveUnitDataSet /_loadUnitDataSet und _exportUnitDataSet /_importUnitDataSet können die Unitdaten-Blockinformationen gespeichert werden. Fehlen beim Lesen eines Datensatzes einer oder mehrere Blöcke, ist dies am Returncode erkennbar, die restlichen Blöcke werden allerdings gelesen. Siehe auch Allgemeines zum Speichern von Datensätzen aus dem Anwenderprogramm (Seite 414) und Datensicherung und -initialisierung aus Anwenderprogramm - Funktionen und Hinweise (Seite 459) .

Verlust der Retaindaten durch Initialisierung

Diese Daten können Sie zuvor im SCOUT über die Funktion "Variablen sichern" sichern und über die Funktion "Variablen wiederherstellen" wieder einlesen.

Alternativ können Sie auch in der Applikation die Runtime-Funktionen _exportUnitDataSet / _importUnitDataSet verwenden.

7.1.5.4 Einfluss des Compilers auf die Variableninitialisierung

Compilerschalter "Einmalige Programmdateninstanziierung"

Der Compiler-Schalter kann an jeder Quelle gesetzt werden und überschreibt damit die globale Einstellung. Der Compilerschalter wirkt unabhängig von der Erstellsprache, ist daher auch in KOP/FUP und MCC anwendbar.

Der Compilerschalter steuert, wie die Instanzdaten der in einer Quelle enthaltenen PROGRAM's anzulegen sind. Instanzdaten eines PROGRAMS werden dabei durch den Inhalt des VAR Deklarationsblocks gebildet.

Generell gilt: ein Programm kann pro Task im Ablaufsystem einmal und nur einmal eingehängt werden.

Ist "einmalige Programmdateninstanziierung" nicht eingestellt (DEFAULT) dann gilt folgendes Verhalten:

- Das Programm kann nicht aus einer anderen POE aufgerufen werden
- die Instanzdaten eines Programms werden für jede Task getrennt angelegt, zu der das Programm zugeordnet ist.
- Die Dateninitialisierung der Instanzdaten erfolgt mit Starten der TASK; (sequentielle Task mit Taskstart, zykl. Tasks beim STOP-RUN Übergang)
- Instanzdaten von PROGRAMs in sequentiellen Tasks sind in diesem Modus in RUN änderbar (wenn ein Einwechseln prinzipiell möglich ist).

Die Aktivierung des Compilerschalters "einmalige Programmdateninstanziierung" (auch bei Verwendung eines Programms in verschiedenen Tasks) bewirkt Folgendes:

- Instanzdaten so übersetzter Programme werden nur einmal angelegt. Die Instanzdaten liegen in der Quelle, in der das PROGRAM deklariert ist.
- Jede Verwendung eines so erstellten PROGRAM's arbeitet auf denselben Daten. Dies betrifft die Zuordnung zu (ggf. mehreren) Tasks und den Aufruf in anderen PROGRAMS oder Funktionsbausteinen.
- Bei Einstellung 'einmalige Programmdateninstanziierung' erfolgt die Dateninitialisierung nach den Regeln der globalen Dateninitialisierung (siehe Downloadeinstellungen am Projekt) mit dem Download der Quelle/ des Codes, in der/dem das Programm liegt (deklariert ist). Siehe auch Download im RUN von geänderten Quellen (Seite 535).

VORSICHT
Verändertes Verhalten bei der Dateninitialisierung, wenn Sie "Programminstanzdaten nur einmal anlegen" ausgewählt haben. Die Dateninitialisierung erfolgt bei sequentiellen Tasks nicht mehr mit einem Taskstart bzw. bei zyklischen Tasks mit dem STOP-RUN-Übergang, sondern generell nur bei einem Download. Ggf. muss nun die Dateninitialisierung applikativ in der StartUpTask bzw. zu Beginn der Programme in sequentiellen Tasks erfolgen. Initialisierung im STOP-RUN-Übergang ist mit einem Pragma möglich und ab V4.2 über den Dialog **Eigenschaften > Einstellungen** über das Kontextmenü am Gerät, siehe Initialisierung von Daten bei einem STOP - RUN - Übergang (Seite 542). Einfluss des Compilerschalters in Zusammenhang mit dem Download im RUN, siehe Download im RUN von geänderten Quellen (Seite 535). Ist ein Programm mehreren Tasks zugeordnet, wird auf den gleichen Daten gearbeitet.

Initialisierung von Variablen bei einem STOP - RUN - Übergang

Für Informationen über die Initialisierung bei einem STOP - RUN - Übergang und das Pragma BlockInit_OnDeviceRun, siehe Initialisierung von Daten bei einem STOP - RUN - Übergang (Seite 542).

Compilerschalter "Spracherweiterungen zulassen" (für Nicht-IEC_Konformität)

Der Compiler-Schalter kann an jeder Quelle gesetzt werden und überschreibt damit die globale Einstellung. Er wirkt unabhängig von der Erstellsprache und ist daher auch in KOP/FUP und MCC anwendbar.

Der Compilerschalter erlaubt Folgendes:

- Direkter Bitzugriff, Bitadressierung bei Bitstringvariablen (außer BOOL)
- INPUT-Variablen von Funktionsbausteinen außerhalb des "Baustein" Scopes lesen und schreiben
- Zulässiger Aufruf, "program in program".

Ein Programm kann innerhalb eines anderen Programms wie eine globale FB-Instanz innerhalb einer POE aufgerufen werden, z. B. Aufruf von "myprog" innerhalb eines anderen Programms, innerhalb eines anderen FB, nicht aber innerhalb einer Funktion.

Für "program in program" ist die globale Verfügbarkeit der Instanzdaten eine Voraussetzung. Diese kann entweder dadurch erfüllt sein, das das PROGRAM über keine Instanzdaten verfügt oder aber durch Anwendung des Compilerschalters "einmalige Programmdateninstanziierung" beim Übersetzen des zu rufenden PROGRAM's (siehe **Einmalige Programmdateninitialisierung**).

Hinweis

Wenn Sie die Compilerschalter nicht setzen, bleibt das Verhalten im Vergleich zu V4.0 unverändert (entspricht DEFAULT).

Neuinitialisierung von Variablen-Blöcken

In VAR_GLOBAL, VAR_GLOBAL RETAIN Blöcken (Interface- und Implementation-Teil der Quelle/Unit):

- `BlockInit_OnChange := true`; bewirkt (nur IMPLEMENTATION), dass bei Änderungen am Blockaufbau beim Download im Run eine Neuinitialisierung der Daten mit den in der Quelle spezifizierten Werten vorgenommen wird.
- Bei MCC und KOP/FUP kann ab V4.2 über Pragma-Zeilen in den Deklarationstabellen das Verhalten eingestellt werden.

Dieses Pragma ist auch in VAR-Deklarationen von PROGRAMS anwendbar. Es wirkt allerdings dort nur, wenn die Compileroption - "Einmalige Programmdateninstanziierung" gewählt ist.

Siehe auch Download im RUN von geänderten Quellen (Seite 535).

7.1.5.5 HMI-relevante Daten kennzeichnen

Daten als "nicht HMI-relevante Daten" kennzeichnen

Standardmäßig stehen für Bedienen&Beobachten die im Interface-Abschnitt einer Unit deklarierten Variablen zur Verfügung.

Über ein Compiler-Pragma am Blockanfang können Sie einen Block im Interface-Bereich als nicht HMI-relevant kennzeichnen. Es werden dann keine Bedienen&Beobachten Adressen für die enthaltenen Variablen mehr generiert und Änderungen an diesem Block wirken sich nicht mehr auf die HMI-Konsistenz aus.

Tabelle 7- 3 Beispiel

```
VAR_GLOBAL
    {HMI_Export := false;  }
    x : INT;
    y : INT;
END_VAR
```

Daten als HMI-relevante Daten kennzeichnen

Über ein Compiler-Pragma am Blockanfang können Sie einen Block im Interface-Bereich als HMI-relevant kennzeichnen. Es werden dann B&B Adressen für die enthaltenen Variablen generiert und Änderungen an diesem Block wirken sich auf die HMI-Konsistenz aus.

Tabelle 7- 4 Beispiel

```
VAR_GLOBAL
    {HMI_Export := true;  }
    x : INT;
    y : INT;
END_VAR
```

Es gilt, dass alle HMI-relevanten Variablen unterhalb der 64kByte Adressgrenze für HMI-Zugriff liegen müssen. Dies entspricht für Geräte-Varianten kleiner V4.1 der Stellung des Blocks innerhalb der Quelle, wobei RETAIN Blöcke unabhängig von der Zugehörigkeit zum Quellabschnitt vor den dynamischen Blöcken eingeordnet werden (keine Änderung möglich).

Bei Geräte-Varianten ab V4.1 belegen nur noch HMI-relevante Blöcke Bereiche im HMI-Adressraum.

Wird die 64kByte Adressgrenze überschritten, erfolgt eine Warnung, ab welcher Variablen ein HMI-Zugriff nicht mehr möglich ist. Werden Datenblöcke explizit durch Angabe des Compiler-Pragmas für HMI exportiert und es ist nicht möglich Variablen des Blocks über HMI zu erreichen, erfolgt bereits beim Übersetzen eine Fehlermeldung.

Hinweis

Der Zugriff von HMI ohne Konsistenzcheck-Einstellung ist möglich, wenn Variablen am Ende des gesamten VAR_GLOBAL -Bereiches angefügt werden, d.h. im letzten VAR_GLOBAL Block Variablen angefügt werden oder ein ganzer VAR_GLOBAL Definitionsblock am Ende ergänzt wird.

Siehe auch Kopplung HMI (Human Machine Interface) (Seite 484) und HMI-Variablen in einer eigenen Unit (Seite 568) .

Detaillierte Informationen zum Verhalten von HMI-relevanten Daten, siehe *ST Programmierhandbuch*, Abschnitt *Variablen und HMI-Geräte*.

Neuinitialisierung von Variablen Blöcken mit HMI-relevanten Daten

HMI-relevante Daten können in VAR_GLOBAL, VAR_GLOBAL RETAIN Blöcken durch folgendes Pragma am Beginn des Blocks gekennzeichnet werden:

`HMI_Export := [true|false];` bewirkt einen Adressexport für HMI-Geräte abweichend von der Default-Lage (INTERFACE wird exportiert, IMPLEMENTATION nicht).

Bei MCC und KOP/FUP kann ab V4.2 über Pragma-Zeilen in den Deklarationstabellen das Verhalten eingestellt werden.

7.1.6 Taskstatus

7.1.6.1 Abfrage und Bedeutung der Taskstatus

Den Taskstatus können Sie mit der Funktion _getStateOfTaskId(*taskId*) abfragen. Diese Funktion benötigt als Eingabeparameter die TaskId und liefert als Rückgabewert einen Wert vom Datentyp DWORD.

Die Tabelle zeigt die möglichen Taskstatus in Hexadezimaldarstellung und als symbolische Konstanten. Kombinationen der Taskstatus sind möglich und werden als Summe der Hexadezimalwerte angezeigt.

Tabelle 7- 5 Taskstatus und deren Bedeutung

Symbolische Konstante	Hex-Darstellung	Symbolische Konstante
TASK_STATE_INVALID	16#0000	Die Task existiert nicht.
TASK_STATE_STOP_PENDING	16#0001	Task hat Signal zum Beenden erhalten; sie befindet sich zwischen den Zuständen TASK_STATE_RUNNING und TASK_STATE_STOPPED. Bis zum Beenden der Task können noch Aktionen ausgeführt werden.
TASK_STATE_STOPPED	16#0002	Task gestoppt (z. B. durch Funktion _resetTask), beendet oder noch nicht gestartet.
TASK_STATE_RUNNING	16#0004	Task läuft, z. B.: • durch Funktion _startTask, • als aktive zyklische Task.
TASK_STATE_WAITING	16#0010	Task in Wartestellung wegen Funktion _waitTime oder WAITFORCONDITION.
TASK_STATE_SUSPENDED	16#0020	Task ausgesetzt durch Funktion *_suspendTaskid*.
TASK_STATE_WAIT_NEXT_CYCLE	16#0040	TimerInterruptTask wartet auf ihren Starttrigger.
TASK_STATE_WAIT_NEXT_INTERRUPT	16#0080	SystemInterruptTask oder UserInterruptTask wartet auf Eintreten des auslösenden Ereignisses.
TASK_STATE_LOCKED	16#0100	Task gesperrt durch Funktion *_disableScheduler*.

7.1.6.2 Kombinationen der Taskstatus

Als Rückgabewert liefern *_getStateOfTaskId* (bzw. *_getStateOfTask*) oft Kombinationen der in Tabelle *Schlüsselwörter zur Deklaration statischer und temporärer Variablen* in Abhängigkeit vom Quelldatei-Abschnitt genannten Taskstatus, die durch eine ODER-Verknüpfung gebildet werden. Häufige Kombinationen sind in der Tabelle aufgelistet.

Tabelle 7- 6 Häufige Kombinationen von Taskstatus

Kombination	Hex-Darstellung	Bedeutung
TASK_STATE_WAITING OR TASK_STATE_RUNNING	16#0014	Task läuft, aber z. Zt. in Wartestellung z. B. durch *_waitTime* oder WAITFORCONDITION.
TASK_STATE_SUSPENDED OR TASK_STATE_RUNNING	16#0024	Task läuft, aber z. Zt. ausgesetzt durch *_suspendTaskid*.
TASK_STATE_WAIT_NEXT_CYCLE OR TASK_STATE_RUNNING	16#0044	TimerInterruptTask läuft, aber wartet z. Zt. auf ihren Starttrigger.
TASK_STATE_WAIT_NEXT_CYCLE OR TASK_STATE_SUSPENDED OR TASK_STATE_RUNNING	16#0064	TimerInterruptTask läuft, aber wartet z. Zt. auf ihren Starttrigger und ist ausgesetzt durch *_suspendTaskid*.
TASK_STATE_WAIT_NEXT_INTERRUPT OR TASK_STATE_RUNNING	16#0084	SystemInterruptTask oder UserInterruptTask läuft, aber wartet z. Zt. auf ihr auslösendes Ereignis.
TASK_STATE_WAIT_NEXT_INTERRUPT OR TASK_STATE_SUSPENDED OR TASK_STATE_RUNNING	16#00A4	SystemInterruptTask oder UserInterruptTask läuft, aber wartet z. Zt. auf ihr auslösendes Ereignis und ist ausgesetzt durch *_suspendTaskid*.
TASK_STATE_LOCKED OR TASK_STATE_RUNNING	16#0104	Task läuft, aber z. Zt. gesperrt durch *_disableScheduler*.

Weitere Kombinationen sind möglich.

7.1.6.3 Beispiel für Verwendung der Taskstatus

Im folgenden Beispiel wird der Taskstatus einer MotionTask abgefragt, um zu entscheiden, ob diese gestartet werden kann. Hierzu werden die relevanten Bits des Rückgabewerts ausgewertet. Bei positivem Ergebnis wird die MotionTask gestartet.

Tabelle 7- 7 Beispiel für Abfrage, ob eine MotionTask gestartet werden kann

```
ret_dword := _getStateOfTaskId (id := _task.motionTask_1);
    IF (ret_dword AND
        (TASK_STATE_STOPPED OR TASK_STATE_STOP_PENDING)
        ) <> 0 THEN
        // MotionTask kann gestartet werden.
        ret_dword := _restartTaskId (id := _task.motionTask_1);
    ELSE
        ; // MotionTask kann nicht gestartet werden.
    END_IF;
```

Für ein Beispiel mit MCC, siehe MCC-Programmierhandbuch unter Befehl Taskzustand.

7.1.7 MotionTask auf Erfüllung einer Bedingung warten lassen

7.1.7.1 Syntax der Bedingung der EXPRESSION

Nachfolgendes betrifft die Programmierung in ST. Bei MCC erfolgt die Definition der Wartebedingung direkt in den Befehlen.

Mit dem Befehl WAITFORCONDITION können Sie in einer MotionTask auf die Erfüllung einer Bedingung (z. B. Eintreten eines Ereignisses) warten. Die MotionTask, in der die Anweisung aufgerufen wird, wird so lange in den Zustand TASK_STATE_WAITING versetzt, bis die Bedingung erfüllt ist.

Bei Verwendung der Durchgangsparameter VAR_IN_OUT können Sie eine Zeitüberwachung bei WAITFORCONDITION ermöglichen.

Diese Bedingung wird in Form einer EXPRESSION formuliert.

Die Expression ist ein Spezialfall einer Funktionsvereinbarung (zur Syntax siehe Kapitel **Expressions** im ST Programmierhandbuch):

- Der Datentyp des Rückgabewerts kann als BOOL festgelegt werden und wird nicht explizit angegeben.
- Die Verwendung der Durchgangsparameter VAR_IN_OUT und Eingangsparameter VAR_INPUT ist möglich.

Eine Expression kann ausschließlich im Implementationsabschnitt der Unit vereinbart werden.

Optional können im Deklarationsabschnitt lokale (temporäre) Variablen deklariert werden. Andere Deklarationen (z. B. von Eingangsparametern oder Konstanten) sind nicht möglich.

Im Anweisungsabschnitt kann zugegriffen werden:

- auf diese lokalen Variablen der Expression
- auf Unit-Variablen
- auf geräteglobale Variablen, I/O-Variablen und das Prozessabbild

Im Anweisungsabschnitt der Expression muss ein Ausdruck vom Datentyp BOOL der Expression-Bezeichnung zugeordnet werden (siehe Kapitel **Expressions** im ST Programmierhandbuch).

Hinweis

Der Anweisungsabschnitt der Expression darf keine Funktionsaufrufe oder Schleifen enthalten.

7.1.7.2 Syntax der WAITFORCONDITION-Anweisung

Der Aufruf des Befehls für das Warten einer Task in Abhängigkeit von einer Expression erfolgt mit folgender Syntax:

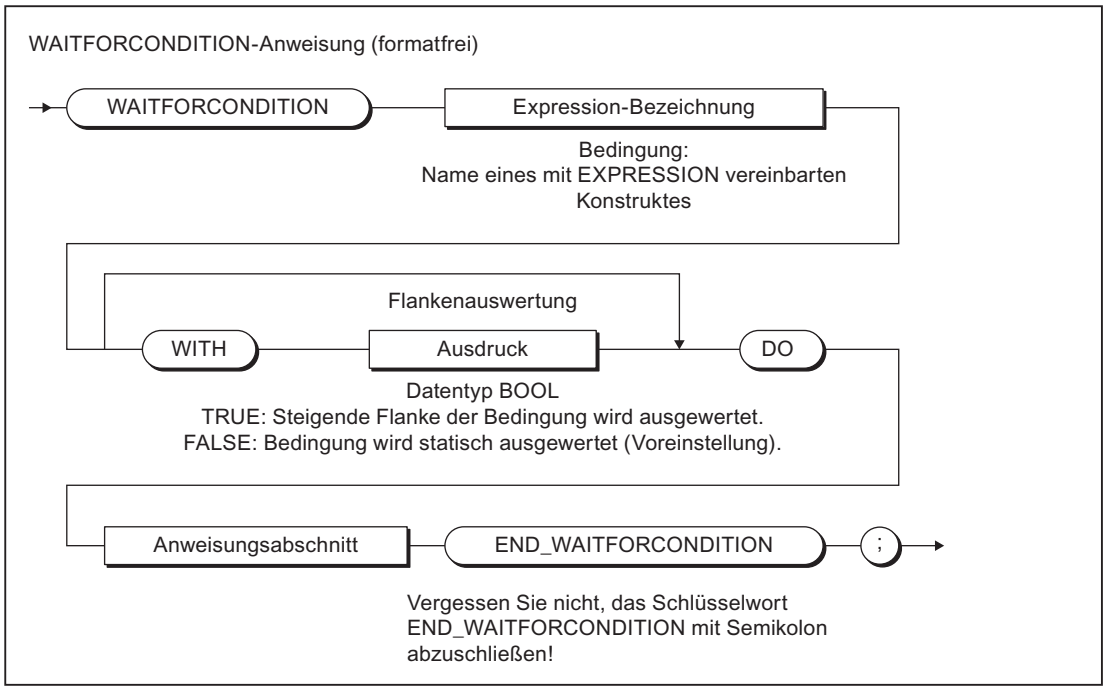


Bild 7-1 Syntax WAITFORCONDITION-Anweisung

Expression-Bezeichnung ist ein mit EXPRESSION vereinbartes Konstrukt; ihr Wert bestimmt (ggf. zusammen mit *WITH Flankenauswertung*), ob die Bedingung als erfüllt gilt.

Die Folge *WITH Flankenauswertung* ist optional. Dabei ist *Flankenauswertung* ein Ausdruck vom Datentyp BOOL; er bestimmt, wie der Wert von *Expression-Bezeichnung* ausgewertet wird:

- *Flankenauswertung* = TRUE: Die steigende Flanke von *Expression-Bezeichnung* wird ausgewertet; d. h. die Bedingung ist erfüllt, wenn der Wert von *Expression-Bezeichnung* von FALSE nach TRUE **wechselt**.
- *Flankenauswertung* = FALSE: Der statische Wert von *Expression-Bezeichnung* wird aus gewertet; d. h. die Bedingung ist erfüllt, wenn der Wert von *Expression-Bezeichnung* TRUE **ist**.

Bei Nichtangabe von WITH Flankenauswertung ist die Voreinstellung FALSE, d. h. der statische Wert von *Expression-Bezeichnung* wird ausgewertet.

Eine negative Flanke können Sie mittels *NOT Expression-Bezeichnung* abfragen.

7.1.7.3 Arbeitsweise der WAITFORCONDITION-Anweisung

Die WAITFORCONDITION-Anweisung setzt die Task, aus der heraus sie aufgerufen wird, so lange in den Zustand TASK_STATE_WAITING, bis die Bedingung (Expression) wahr (TRUE) wird.

Wenn die Expression wahr wird, wird die Priorität der Task erhöht. Die Priorität liegt oberhalb der UserInterruptTasks und unterhalb der SysteminterruptTasks, d. h. die MotionTask kommt vor den anderen Tasks der Round-Robin-Ablaufebene sowie den UserInterruptTasks und TimerInterruptTasks an die Reihe.

Die Höherpriorisierung der Task gilt für alle Anweisungen, die zwischen WAITFORCONDITION und END_WAITFORCONDITION eingeschlossen sind. Mit dem Befehl END_WAITFORCONDITION endet die Höherpriorisierung der Task.

Der Anweisungsteil muss mindestens eine Leeranweisung enthalten.

Bitte beachten Sie Folgendes, wenn Sie den Befehl WAITFORCONDITION einsetzen:

- Der Befehl dient zum Warten auf ein Ereignis in einer MotionTask. Ist er innerhalb einer anderen Task programmiert, so wird er ignoriert.
- In einer MotionTask gibt es keine Zeitüberwachung.

  Achten Sie deshalb darauf, dass die Bedingung jemals wahr wird. Andernfalls bleibt die MotionTask immer im Wartezustand. Dies führt zu keinem Zeitüberlauffehler.
- Ab 4.1 kann in einer Motion Task eine Zeitüberwachung bei WAITFORCONDITION programmiert werden.
- Die Struktur WAITFORCONDITION ... END_WAITFORCONDITION darf nicht geschachtelt werden.
- Innerhalb der Round-Robin-Ebene wird die wartende Task bei Eintritt des Ereignisses als Nächstes bearbeitet, sofern sie nicht durch höherpriorige Aufgaben (z. B. Alarme) behindert wird.
- Die Zeitscheibe der aktiven Task in der Round-Robin-Ebene wird abgebrochen.
- Verhalten bei Taskunterbrechung siehe *_suspendTaskid*.
- Die Expression wird im IPO-Takt hochpriorig überprüft.

Siehe auch

Task-Prioritäten (Seite 197)

MotionTasks (Seite 206)

7.1.7.4 Beispiel zur Verwendung von WAITFORCONDITION

Im folgenden Beispiel wird unterstellt, dass das Programm *feeder* in einer MotionTask abläuft. Für diese MotionTask ist die Option **Aktivierung nach StartupTask** ausgewählt.

Die Zuordnung von Programmen zu Tasks nehmen Sie im SIMOTION SCOUT vor (siehe **Programme den Ablaufebenen/Tasks zuweisen**).

Tabelle 7- 8 Beispiel zu Verwendung des Befehls WAITFORCONDITION

```
INTERFACE
 USEPACKAGE CAM;
 PROGRAM feeder;
END_INTERFACE

IMPLEMENTATION
    // condition for WAITFORCONDITION in MotionTask_1
    EXPRESSION automaticExpr
        automaticExpr := IOfeedCam; // digital input
    END_EXPRESSION
    // feeder (MotionTask_1)
    PROGRAM feeder
        VAR
            retVal : DINT ;
    END_VAR ;
    retVal := _enableAxis(axis := realAxis,
        enableMode := ALL,
        servoCommandToActualMode := INACTIVE,
        nextCommand := WHEN_COMMAND_DONE,
        commandId := _getCommandId() );
        // wait for automatic start
    WAITFORCONDITION automaticExpr WITH TRUE DO
        retVal := _pos(axis := realAxis,
            positioningMode := RELATIVE,
            position := 500,
            velocityType := DIRECT,
            velocity:= 300,
            positiveAccelType := DIRECT,
            positiveAccel:= 400,
            negativeAccelType := DIRECT,
            negativeAccel := 400,
            velocityProfile:= TRAPEZOIDAL,
            mergeMode:= IMMEDIATELY,
            nextCommand := WHEN_MOTION_DONE,
            commandId:= _getCommandId() );
            // reduce priority after WAITFORCONDITION
    END_WAITFORCONDITION;
    retVal := _disableAxis(axis := realAxis,
    disableMode := ALL,
    servoCommandToActualMode := INACTIVE,
    nextCommand := WHEN_COMMAND_DONE,
    commandId := _getCommandId() );
    END_PROGRAM
END_IMPLEMENTATION
```

7.1.7.5 Beispiel zur Zeitüberprüfung mittels FB

Beispiel zur Verwendung von WAITFORCONDITION mit Zeitüberprüfung mittels FB (ab V4.1)

Im folgenden Beispiel wird die Zeitüberwachung der *Expressions* (WAITFORCONDITION) gezeigt. Daneben wird die Bindung der Expression an der Aufrufstelle an Instanzdaten eines rufenden Funktionsbausteins inklusive der Zeitüberwachung dargestellt. Die Referenzvariable ist vom Typ TON. Der Aufruf erfolgt innerhalb der Expression mit Abfrage des Outputs.

```
INTERFACE
    VAR_GLOBAL
        v1, v2 : INT;
        t1, t2 : TIME;
        Auswertung: INT;
    END_VAR
    PROGRAM Test_1;
END_INTERFACE
IMPLEMENTATION
    EXPRESSION mcccont1
        VAR_IN_OUT
            v : INT;
            t : TON;
        END_VAR
        t();
        mcccont1 := v > 100 or t.q ;
    END_EXPRESSION
    FUNCTION_BLOCK waitfb
        VAR_IN_OUT
            refpar1 : INT;
            refpar2 : TIME;
        END_VAR
        VAR_TEMP
            expr_timeout : TON;
        END_VAR
        expr_timeout(pt := refpar2, IN := TRUE);
        // Überwachungszeit setzen und
        //Timer aktivieren
        Auswertung:=0;
        WAITFORCONDITION mcccont1(v := refpar1, t := expr_timeout ) DO
           Auswertung:=100 ; //Anweisung
            IF (expr_timeout.q) THEN
                // Fehlerbehandlung machbar, wenn Time-Out
                Auswertung:=20005;
            END_IF;
        END_WAITFORCONDITION
        expr_timeout( IN := FALSE);
        ;
    END_FUNCTION_BLOCK
    PROGRAM test_1
```

```
        VAR
            my_waitfb1 : waitfb;
        END_VAR
        my_waitfb1 (refpar1 :=v1, refpar2:=t1);
        //Warte bis V1>100 und Zeit
    END_PROGRAM
END_IMPLEMENTATION
```

Der Aufruf der Instanz vom Typ waitfb kann dann an beliebiger Stelle mit jeweils unterschiedlichen Variablen erfolgen. Diese globalen Variablen werden dann von zyklischen Tasks upgedated:

```
my_waitfb_1(refpar1 := v1, refpar2:=t1); //Warte bis V1>100 und Zeit t1 nicht
abgelaufen

my_waitfb_2(refpar1 := v2, refpar2:=t1);
```

7.1.7.6 Beispiel zur Verwendung von WAITFORCONDITION mit Zeitüberwachung direkt in nicht zyklischer Task / Motion Task

Beschreibung

Beispiel zur Zeitüberwachung der Expression (WAITFORCONDITION).

Der Aufruf erfolgt direkt in einer MotionTask.

Die Zeitüberwachung ist vom Typ TON, mit Aufruf innerhalb der Expression und Abfrage des Outputs.

```
INTERFACE
    TYPE
        eDiagType : (TIMEOUT, COUNTER_REACHED, NOTHING);
    END_TYPE
    VAR_GLOBAL
        eDiag   : eDiagType := NOTHING;
        v1      : INT  := 0;
        t1      : TIME := T#0d_0h_0m_3s_0ms;
    END_VAR
    PROGRAM testExpression;
    PROGRAM increaseV1;
END_INTERFACE
IMPLEMENTATION
    EXPRESSION expr
        VAR_IN_OUT
            v : INT;
            t : TON;
        END_VAR
        t();
        expr := v > 500 OR t.q;
    END_EXPRESSION
```

```
                        //MotionTask
    PROGRAM testExpression
        VAR_TEMP
            expr_timeout : TON;
        END_VAR
                        // Überwachungszeit setzen, und Timer aktivieren
expr_timeout(pt := t1, IN := TRUE);
                       //Warten bis v1 > 500 oder Zeit t1 abgelaufen
        WAITFORCONDITION expr(v := v1, t := expr_timeout) DO
            IF (expr_timeout.q) THEN
                       // Fehlerbehandlung
                       // Zeit t1 ist abgelaufen ohne das v1 > 500 wurde
                eDiag := TIMEOUT;
                … ;
            ELSE
                       // Gutfall,  v1 > 500
                eDiag := COUNTER_REACHED;
                 … ;
            END_IF;
        END_WAITFORCONDITION;
    END_PROGRAM
                        //BackgroundTask
    PROGRAM increaseV1
        v1 := v1 + 1;
    END_PROGRAM
END_IMPLEMENTATION
.....
```

Sie können v1 in beliebigen zyklischen Tasks updaten.

7.1.8 Tasks eine definierte Zeitdauer warten lassen

Task in den Wartezustand versetzen

Sie können eine Task eine vorgegebene Zeit in den Zustand TASK_STATE_WAITING (siehe Taskstati (Seite 321)) versetzen. Hierzu verwenden Sie die Funktion *_waitTime*. Sie versetzt die aufrufende Task für die angegebene Zeit in den Wartezustand.

Hinweis

Im Wartezustand benötigt eine Task (fast) keine Rechenzeit. Das Zielsystem wird nur mit der periodischen Prüfung belastet, ob die Wartezeit abgelaufen ist. Diese Prüfung findet im IPO-Takt statt.

Der Aufruf von *_waitTime(timeValue := T#0ms)* in einer MotionTask inaktiviert diese kurzzeitig und gibt die Programmkontrolle an den Scheduler zurück. Dies ist z.B. für längere Schleifen empfehlenswert, wenn die Programmkontrolle bewusst an die nächste Round Robin Task übergeben werden soll (auch in BackgroundTask möglich).

ACHTUNG

Die Funktion sollte nur in MotionTasks verwendet werden, die Verwendung in zyklischen Tasks kann zu Zeitüberwachungsfehlern führen!

- Bei SynchronousTasks: Sie können konfigurieren, ob die Zeitüberwachung ausgesetzt wird. Standardmäßig ist die Zeitüberwachung aktiv.

  Beachten Sie zusätzlich bei der IPOsynchronousTask: Die UserInterruptTasks werden nicht mehr durch ihr auslösendes Ereignis gestartet!
- Bei anderen zyklischen Tasks (BackgroundTask, TimerInterruptTasks): Die Zeitüberwachung ist immer aktiv.

Verwenden Sie in zyklischen Tasks die Systemfunktionsbausteine Zeitgeber (siehe Kapitel *Zeitgeber*), um Wartezeiten zu realisieren.

Als Eingangsparameter verwenden Sie einen Ausdruck vom Datentyp TIME, der Rückgabewert vom Datentyp DINT ist immer 0.

Nähere Erläuterungen zur Funktion (Syntax) finden Sie in der Beschreibung der Funktion **_waitTime**.

Folgendes Beispielprogramm setzt die MotionTask, der es zugeordnet ist, für 10 Sekunden in den Wartezustand:

Tabelle 7- 9 Beispiel für die Funktion _waitTime

```
INTERFACE
    PROGRAM waitTime;
END_INTERFACE

IMPLEMENTATION
    PROGRAM waitTime
        VAR
            retVal :DINT;
        END_VAR;
        retVal := _waitTime(timeValue := T#10s);
    END_PROGRAM
END_IMPLEMENTATION
```

7.2 Tasksteuerbefehle

7.2.1 Übersicht der Tasksteuerbefehle

Zum Steuern von Tasks stehen Ihnen die in der Tabelle aufgelisteten Befehle zur Verfügung.

Tabelle 7- 10 Tasksteuerbefehle in SIMOTION ST

Tasksteuerbefehl	Bedeutung
_startTaskId(*TaskId*) _	Startet eine MotionTask die sich im Zustand TASK_STATE_STOPPED; ihr Startup-Code wird ausgeführt. Nur anwendbar auf MotionTasks. Befehl darf nicht direkt auf _resetTaskId folgen. Verwenden Sie stattdessen _*restartTaskId*.
_resetTaskId(*TaskId*) _	Setzt eine MotionTask in den Zustand TASK_STATE_STOPPED zurück Nur anwendbar auf MotionTasks.
_restartTaskId(*TaskId*)	Setzt eine MotionTask in den Zustand TASK_STATE_STOPPED zurück und startet sie neu; ihr Startup-Code wird ausgeführt. Nur anwendbar auf MotionTasks. Auf diesen Befehl darf nicht direkt _*startTaskId* folgen. Verwenden Sie stattdessen _*restartTaskId*.
_suspendTaskId(*TaskId*)	Die betreffende Task wird ausgesetzt (in den Zustand TASK_STATE_SUSPENDED setzen), bei zyklischen Tasks wird die Zeitüberwachung ausgesetzt. Nicht anwendbar auf die SynchronousTasks, StartupTask und ShutdownTask.
_resumeTaskId(*TaskId*)	Die betreffende Task, die sich im Zustand TASK_STATE_SUSPENDED, wird wieder aufgenommen; Zeitüberwachungen werden wieder. Nicht anwendbar auf die SynchronousTasks, StartupTask und ShutdownTask.
_getStateOfTaskId(*TaskId*)	Fragt den Zustand der betreffenden Task ab.
_retriggerTaskIdControlTime(*TaskId*)	Für die betreffende Task wird die Überwachungszeit einmalig zurückgesetzt.

MCC-Blöcke für Taskbefehle sind im MCC-Programmierhandbuch beschrieben.

Beispiel für einen Tasksteuerbefehl siehe Tasksteuerbefehle.

Siehe auch Taskstati.

TaskId

Die Task wird bei diesen Funktionen über eine eindeutige TaskId vorgegeben. Die TaskId zum Namen einer Task erhalten Sie auf folgende Weise:

- Als Variable _task.*name* (z. B. _task.motionTask_1). Hierbei bedeutet:
  - _task bezeichnet den vordefinierten Namensraum für TaskId (siehe Kapitel *Namensräume*). Der Bezeichner des Namensraums steht mit Punkt getrennt vor dem nachfolgenden Bezeichner
  - *name* ist der Bezeichner der Task wie Ablaufsystem vorgegeben (z. B. motionTask_1).
- Mit der Funktion _getTaskId(*name*)
  Die Task wird über ihren Namen (wie im Ablaufsystem) vorgegeben.
- Mit der Funktion _checkEqualTask können Sie überprüfen, ob eine TaskId zu einer bestimmten Task gehört (siehe **Funktion _checkEqualTask**).

Darüber hinaus stellen die SIMOTION Geräte Systemfunktionen zum Steuern des Schedulers zur Verfügung. Diese sind in der Tabelle aufgeführt (Beschreibung siehe Listenhandbücher der SIMOTION Geräte):

Tabelle 7- 11 Befehle zum Steuern des Schedulers

Tasksteuerbefehl	Bedeutung
_disableScheduler()	Verhindert solange das Einwechseln aller Anwenderprogramm-Tasks (außer SynchronousTasks), bis *_enableScheduler* aufgerufen wird. System-Tasks sind davon nicht betroffen. **Hinweis**: Die Zeitüberwachung für zyklische Tasks wird **nicht** ausgesetzt. **Hinweis**: Der Befehl verhindert auch das Einwechseln der SystemInterruptTasks und UserInterruptTasks!
_enableScheduler()	Hebt Wirkung von *_disableScheduler* auf.

ACHTUNG
Die Funktionen _checkEqualTask und _getTaskId sowie Variablen der Form _task.*name* dürfen in Bibliotheken nicht verwendet werden.

Siehe auch

Beispiel Verwendung eines Tasksteuerbefehls (Seite 333)

Abfrage und Bedeutung der Taskstatus (Seite 321)

7.2.2 Beispiel Verwendung eines Tasksteuerbefehls

Das folgende Beispiel setzt voraus, dass Sie im SIMOTION SCOUT folgende Zuordnungen getroffen haben:

- Das Programm *myStart* ist der StartupTask zugeordnet.
- Das Programm *myMotion* ist der MotionTask_1 zugeordnet.

Nach dem Hochlauf der Steuerung (Übergang von STOP auf RUN) läuft die StartupTask automatisch an. In dieser Task wird das Programm *myStart* ausgeführt.

Im Programm *myStart* wird die MotionTask_1 über den Tasksteuerbefehl *_startTaskId* gestartet. In der MotionTask_1 wird das Programm *myMotion* ausgeführt.

```
INTERFACE
 PROGRAM myStart;
 PROGRAM myMotion;
END_INTERFACE

IMPLEMENTATION
    VAR
        RetDWord : DWORD;
    END_VAR

 PROGRAM myStart
    RetDWord := _startTaskid (_task.motionTask_1);
    // starte MotionTask_1

 END_PROGRAM
    // hier Befehle im Programm, z. B. Achsbefehle
 PROGRAM myMotion

 END_PROGRAM
END_IMPLEMENTATION
```

7.2.3 Funktion _getStateOfTaskId

Die Funktion gibt den Zustand der betreffenden Task zurück. Die Task wird über eine projektweit eindeutige TaskId vorgegeben (siehe Übersicht der Tasksteuerbefehle (Seite 331)).

Die Funktion ist auf alle Tasks anwendbar, außer StartupTask und ShutdownTask.

Siehe auch Taskstatus. Dort ist auch ein Beispiel angegeben, wie Sie anhand des Taskstatus einer MotionTask entscheiden können, ob diese gestartet werden kann (siehe **Beispiel für Verwendung der Taskstatus**).

Deklaration

```
_getStateOfTaskId (
             { id     : StructTaskId // TaskId
             }
             ) : DWORD
```

Eingangsparameter

id (optional)

Datentyp: StructTaskId

Voreinstellung: TaskId der aktuellen Task, in welcher die Funktion aufgerufen wird.

TaskId der Task, die gesteuert werden soll (siehe Übersicht der Tasksteuerbefehle (Seite 331)).

Rückgabewert

Datentyp: DWORD

Zustand der Task wird als ODER-Verknüpfung folgender Werte angezeigt (siehe auch **Taskstati**):

16#0000	Task existiert nicht oder TaskId ist ungültig TASK_STATE_INVALID
16#0001	Task im Übergang von Running nach Stopped TASK_STATE_STOP_PENDING
16#0002	Task gestoppt TASK_STATE_STOPPED
16#0004	Task läuft TASK_STATE_RUNNING
16#0010	Task in Wartestellung TASK_STATE_WAITING
16#0020	Task suspendiert (ausgesetzt) TASK_STATE_SUSPENDED
16#0040	TimerInterruptTask wartet auf ihren Starttrigger TASK_STATE_WAIT_NEXT_CYCLE
16#0080	SystemInterruptTask bzw. UserInterruptTask wartet auf eintretendes auslösendes Ereignis TASK_STATE_WAIT_NEXT_INTERRUPT
16#0100	Task gesperrt (durch _disableScheduler) TASK_STATE_LOCKED

Siehe auch

Abfrage und Bedeutung der Taskstatus (Seite 321)

7.2.4 Funktion _resetTaskId

Die Funktion setzt eine MotionTask in den Zustand TASK_STATE_STOPPED zurück. Die Task wird über eine projektweit eindeutige TaskId vorgegeben (siehe Übersicht der Tasksteuerbefehle (Seite 331)).

Die Funktion ist nur auf MotionTasks anwendbar.

Direkt nach dieser Funktion darf nicht die Funktion _startTaskId folgen. Verwenden Sie stattdessen die Funktion _restartTaskId.

Deklaration

```
_resetTaskId (
         id    : StructTaskId    // nur MotionTasks
         ) : DWORD
```

Eingangsparameter

id

Datentyp: StructTaskId

TaskId der Task, die gesteuert werden soll.

Rückgabewert

Datentyp:	DWORD
0	Kein Fehler
16#FFFF_FFFE	TaskId bezieht sich nicht auf eine MotionTask
16#FFFF_FFFF	TaskId ist ungültig

Siehe auch

Funktion _restartTaskId (Seite 336)

Funktion _startTaskId (Seite 339)

7.2.5 Funktion _restartTaskId

Die Funktion setzt eine MotionTask in den Zustand TASK_STATE_STOPPED zurück und startet sie neu; ihre Daten werden initialisiert. Die Task wird über eine projektweit eindeutige TaskId vorgegeben (siehe Übersicht der Tasksteuerbefehle (Seite 331)).

Die Funktion ist nur auf MotionTasks anwendbar.

Um zu verhindern, dass mit dieser Funktion eine laufende MotionTask gestoppt wird, können Sie deren Status abfragen und auswerten (siehe Funktion _getStateOfTaskId sowie das Beispiel für Verwendung der Taskstatus in **Taskstatus**).

Deklaration

```
_restartTaskId (
         id    : StructTaskId  // nur MotionTasks
         ) : DWORD
```

Eingangsparameter

id

Datentyp: StructTaskId

TaskId der Task, die gesteuert werden soll (siehe Übersicht der Tasksteuerbefehle (Seite 331)).

Rückgabewert

Datentyp:	DWORD
0	Kein Fehler
16#FFFF_FFFE	TaskId bezieht sich nicht auf eine MotionTask
16#FFFF_FFFF	TaskId ist ungültig

7.2.6 Funktion _resumeTaskId

Die Funktion nimmt eine Task wieder auf, die sich im Zustand TASK_STATE_SUSPENDED befindet. Dieser Zustand wurde durch die Funktion _suspendTaskId erreicht. Die Task wird über eine projektweit eindeutige TaskId vorgegeben (Übersicht der Tasksteuerbefehle (Seite 331)).

Die Funktion ist anwendbar auf MotionTasks, BackgroundTask, TimerInterruptTasks, UserInterruptTasks, SystemInterruptTasks.

Bei zyklischen Tasks (BackgroundTask, TimerInterruptTasks) wird die Zeitüberwachung der Tasks wieder eingeschaltet.

Deklaration

```
_resumeTaskId (
         id    : StructTaskId
         ) : DWORD
```

Eingangsparameter

id

Datentyp: StructTaskId

TaskId der Task, die gesteuert werden soll (siehe Übersicht der Tasksteuerbefehle).

Rückgabewert

Datentyp:	DWORD
0	Kein Fehler
16#FFFF_FFFE	TaskId bezieht sich auf eine nicht zulässige Task
16#FFFF_FFFF	TaskId ist ungültig

Siehe auch

Funktion _suspendTaskId (Seite 340)

7.2.7 Funktion _retriggerTaskIdControlTime

Die Funktion setzt die Überwachungszeit für die betreffende Task einmalig zurück. Die Task wird über eine projektweit eindeutige TaskId vorgegeben (siehe Übersicht der Tasksteuerbefehle (Seite 331)).

Die Funktion ist anwendbar auf MotionTasks, BackgroundTask, TimerInterruptTasks, UserInterruptTasks, SystemInterruptTasks.

Deklaration

```
_retriggerTaskIdControlTime (
                    { id        : StructTaskId
                    }
                  ) : DWORD
```

Eingangsparameter

id (optional)

Datentyp:	StructTaskId
Voreinstellung:	TaskId der aktuellen Task, in welcher die Funktion aufgerufen wird.

TaskId der Task, die gesteuert werden soll (siehe Übersicht der Tasksteuerbefehle (Seite 331)).

Rückgabewert

Datentyp:	DWORD
0	Funktion ordnungsgemäß ausgeführt.
ungleich 0	Funktion nicht ordnungsgemäß ausgeführt.

7.2.8 Funktion _startTaskId

Die Funktion startet eine MotionTask, die sich im Zustand TASK_STATE_STOPPED befindet; ihre Daten werden initialisiert. Die Task wird über eine projektweit eindeutige TaskId vorgegeben (siehe Übersicht der Tasksteuerbefehle (Seite 331)).

Die Funktion ist nur auf MotionTasks anwendbar.

Sie hat keinen Einfluss auf MotionTasks in folgenden Zuständen:

- TASK_STATE_RUNNING
- TASK_STATE_WAITING
- TASK_STATE_SUSPENDED

Sie können vor Verwendung der Funktion den Status der MotionTask abfragen und auswerten (siehe Funktion _getStateOfTaskId sowie Beispiel für Verwendung der Taskstati in **Taskstati**).

Sie darf nicht direkt auf die Funktion _resetTaskId folgen. Verwenden Sie stattdessen die Funktion _restartTaskId.

Deklaration

```
_startTaskId (
         id    : StructTaskId  // nur MotionTasks
         ) : DWORD
```

Eingangsparameter

id

Datentyp: StructTaskId

TaskId der Task, die gesteuert werden soll (siehe Übersicht der Tasksteuerbefehle (Seite 331) – nur MotionTasks).

Rückgabewert

Datentyp:	DWORD
0	Kein Fehler.
16#FFFF_FFFE	TaskId bezieht sich nicht auf eine MotionTask
16#FFFF_FFFF	TaskId ist ungültig

Siehe auch

Funktion _getStateOfTaskId (Seite 333)

Funktion _resetTaskId (Seite 335)

Funktion _restartTaskId (Seite 336)

7.2.9 Funktion _suspendTaskId

Die Funktion setzt die betreffende Task aus (in den Zustand TASK_STATE_SUSPENDED). Die Task wird über eine projektweit eindeutige TaskId vorgegeben (siehe Übersicht der Tasksteuerbefehle (Seite 331)).

Die Funktion ist anwendbar auf MotionTasks, BackgroundTask, TimerInterruptTasks, UserInterruptTasks, SystemInterruptTasks.

Bei zyklischen Tasks (BackgroundTask, TimerInterruptTasks) wird die Zeitüberwachung der Task angehalten.

Mit der Funktion _resumeTaskId wird die Task und ihre Zeitüberwachung wieder aufgenommen.

Deklaration

```
_suspendTaskId (
             id        : StructTaskId
         ) : DWORD
```

Eingangsparameter

id

Datentyp: StructTaskId

TaskId der Task, die gesteuert werden soll (siehe Übersicht der Tasksteuerbefehle (Seite 331)).

Rückgabewert

Datentyp:	DWORD
0	Kein Fehler.
16#FFFF_FFFE	TaskId bezieht sich auf eine nicht zulässige Task
16#FFFF_FFFF	TaskId ist ungültig

Unterbrechung der Auswertung einer WAITFORCONDITION

Bei der Überprüfung der Wertebedingung einer WAITFORCONDITION wird der Taskstatus nicht automatisch berücksichtigt. Sie haben in einer MotionTask eine Wartebedingung, z. B. warten auf Überfahren eines Initiators, mit WAITFORCONDITION programmiert. Nun wird die Task mit _suspendTaskId unterbrochen, bevor die Wartebedingung erfüllt ist. Anschließend wird die Bedingung temporär erfüllt, z. B. durch manuelles Überfahren eines Initiators. Nach _resumeTask wird die Task so abgearbeitet, als wäre die Bedingung im normalen Ablauf erfüllt worden. Sie können dies jedoch vermeiden, indem Sie in der Bedingung für WAITFORCONDITION zusätzlich den Taskstatus abfragen.

```
// condition for WAITFORCONDITION in MotionTask_1
EXPRESSION automaticExpr
    automaticExpr := IOfeedCam AND (task_status = (TASK_STATE_RUNNING OR
    TASK_STATE_WAITING));  // digital input
END_EXPRESSION
```

Siehe auch

Funktion _resumeTaskId (Seite 337)

7.2.10 Funktion _getTaskId

Die Funktion generiert aus dem Namen einer Task (wie im Ablaufsystem) eine projektweit eindeutige TaskId. Diese TaskId kann einer Variablen vom Datentyp StructTaskId zugewiesen werden und als Eingangsparameter in den folgenden Funktionen verwendet werden:

- Tasksteuerbefehle
- Funktionen zur Laufzeitmessung von Tasks

ACHTUNG
Die Funktion darf in Bibliotheken nicht verwendet werden. Der Aufruf der Funktion darf nur in Kurzform geschehen, d. h. mit vollständiger Auflistung aller Parameterwerte, jedoch ohne Angabe der Formalparameter.

Deklaration

```
_getTaskId (  // nur Kurzform erlaubt
         name    : Task_Name  // Name der Task
         ) : StructTaskId
```

Eingangsparameter

name (nur Kurzform erlaubt)

Name der Task wie im Ablaufsystem.

Rückgabewert

Datentyp: StructTaskId

Der Rückgabewert enthält die TaskId der Task.

7.2.11 Funktion _checkEqualTask

Diese Funktion gibt an, ob eine TaskId zu einer Task gehört.

ACHTUNG
Die Funktion darf in Bibliotheken nicht verwendet werden.
Der Aufruf der Funktion darf nur in Kurzform geschehen, d. h. mit vollständiger Auflistung aller Parameterwerte, jedoch ohne Angabe der Formalparameter.

Deklaration

```
_checkEqualTask (  // nur Kurzform erlaubt
        id     : StructTaskId,
        Name : TaskName // Name der Task
    ) : BOOL
```

Eingangsparameter

id

(nur Kurzform erlaubt)

Datentyp: StructTaskId

Zu überprüfende TaskId, z. B. TSI#taskId.

Name

(nur Kurzform erlaubt)

Datentyp: TaskName

Name einer Task, wie im Ablaufsystem vorgegeben.

Rückgabewert

Datentyp: BOOL

Der Rückgabewert gibt an, ob die TaskId zur angegebenen Task gehört:

TRUE: TaskId gehört zur angegebenen Task.

FALSE TaskId gehört nicht zur angegebenen Task.

7.3 Funktionen zur Laufzeitmessung von Tasks

7.3.1 Funktionen zur Laufzeitmessung von Tasks - Übersicht

Die Funktionen zur Laufzeitmessung sind für alle Tasks zulässig. Die Messung wird jedoch nicht unterstützt von der IPOsynchronousTask, der ServosynchronousTask und der ShutDownTask. Die Funktionen zur Laufzeitmessung von Tasks liefern die Laufzeiten in ms.

Folgende Laufzeiten können gemessen werden:

- Maximale Laufzeit der Task seit dem letzten STOP-RUN-Übergang (siehe Funktion _getMaximalTaskIdRunTime (Seite 345))
- Minimale Laufzeit der Task seit dem letzten STOP-RUN-Übergang (siehe Funktion _getMinimalTaskIdRunTime (Seite 346))
- Laufzeit aus dem vorangegangenen Durchlauf der Task (siehe Funktion _getCurrentTaskIdRunTime (Seite 347))
- Durchschnittswert der Laufzeit der Task aus den letzten 10 vorangegangenen Durchläufen (siehe Funktion _getAverageTaskIdRunTime (Seite 348))

Die Task wird bei diesen Funktionen über eine eindeutige TaskId vorgegeben.

Hinweis

Die Laufzeit von Tasks können Sie auch über den Task Trace bestimmen. Informationen über den Task Trace finden Sie im Funktionshandbuch Task Trace.

Funktionen für die genaue Laufzeitbestimmung

Mit folgenden Systemfunktionen können Sie die Zeit seit Systemstart µs-genau messen. Damit können Sie für Abschnitte innerhalb der Applikation die Zeit genau messen, um so z.B. die Applikation zu optimieren.

- Funktion _getInternalTimeStamp (Seite 349)
- Funktion _getTimeDifferenceOfInternalTimeStamp (Seite 349)

Siehe auch

Tasklaufzeiten (Seite 251)

7.3.2 Funktion _getMaximalTaskIdRunTime

Die Funktion liefert die maximale Laufzeit der Task seit dem letzten STOP-RUN-Übergang, einschließlich aller Unterbrechungen von höherpriorigen Tasks. Die Task wird über eine projektweit eindeutige TaskId vorgegeben (siehe **Funktion _startTaskId**).

Folgende Funktionen unterbrechen die Messung nicht:

- _suspendTaskId
- _disableScheduler (siehe Listenhandbücher der SIMOTION Geräte)

Die ermittelte Laufzeit ist ein Vielfaches des Servo-Takts; bei Laufzeiten kleiner als der Lagereglertakt wird als Messwert T#MIN (= T#0ms) zurückgegeben.

Die Funktion ist für alle Tasks zulässig. Die Messung wird jedoch nicht unterstützt von der IPOsynchronousTask und der ShutDownTask. Bei Aufrufen mit diesen Tasks wird als Messwert T#MIN (= T#0ms) zurückgegeben.

Deklaration

```
_getMaximalTaskIdRunTime (
                { id    : StructTaskId
                }
                ) : TIME
```

Eingangsparameter

id (optional)

Datentyp:	StructTaskId
Voreinstellung:	TaskId der aktuellen Task, in welcher die Funktion aufgerufen wird.

TaskId der Task, deren Laufzeit gemessen werden soll (siehe Seite 6-330).

Rückgabewert

Datentyp:	TIME
T#MIN (= T#0ms = T#1ms * UDINT#0)	Messung wird nicht unterstützt oder ist noch nicht beendet.
größer T#MIN und kleiner T#MAX	Größte aufgetretene Laufzeit.
T#MAX (= T#49d_17h_2m_47s_295ms = T#1ms * UDINT#16#FFFF_FFFF)	TaskId ist ungültig

Siehe auch

Funktion _startTaskId (Seite 339)

Funktion _suspendTaskId (Seite 340)

7.3.3 Funktion _getMinimalTaskIdRunTime

Die Funktion liefert die minimale Laufzeit der Task seit dem letzten STOP-RUN-Übergang, einschließlich aller Unterbrechungen von höherpriorigen Tasks. Die Task wird über eine projektweit eindeutige TaskId vorgegeben (siehe **Funktion _startTaskId**).

Folgende Funktionen unterbrechen die Messung nicht:

- _suspendTaskId
- _disableScheduler (siehe Listenhandbücher der SIMOTION Geräte)

Die ermittelte Laufzeit ist ein Vielfaches des Servo-Takts; bei Laufzeiten kleiner als der Lagereglertakt wird als Messwert T#MIN (= T#0ms) zurückgegeben.

Die Funktion ist für alle Tasks zulässig. Die Messung wird jedoch nicht unterstützt von der IPOsynchronousTask und der ShutDownTask. Bei Aufrufen mit diesen Tasks wird als Messwert T#MIN (= T#0ms) zurückgegeben.

Deklaration

```
_getMinimalTaskIdRunTime (
                { id     : StructTaskId
                }
                ) : TIME
```

Eingangsparameter

id (optional)

Datentyp: StructTaskId

Voreinstellung: TaskId der aktuellen Task, in welcher die Funktion aufgerufen wird.

TaskId der Task, deren Laufzeit gemessen werden soll (siehe **Funktion _startTaskId**)

Rückgabewert

Datentyp:	TIME
T#MIN (= T#0ms = T#1ms * UDINT#0)	Messung wird nicht unterstützt oder ist noch nicht beendet.
größer T#MIN und kleiner T#MAX	Minimal aufgetretene Laufzeit.
T#MAX (= T#49d_17h_2m_47s_295ms = T#1ms * UDINT#16#FFFF_FFFF)	TaskId ist ungültig

Siehe auch

Funktion _startTaskId (Seite 339)

Funktion _suspendTaskId (Seite 340)

7.3.4 Funktion _getCurrentTaskIdRunTime

Die Funktion liefert die Laufzeit aus dem vorangegangenen Durchlauf der Task, einschließlich aller Unterbrechungen von höherprioritigen Tasks. Die Task wird über eine projektweit eindeutige TaskId vorgegeben (siehe **Funktion _startTaskId**).

Folgende Funktionen unterbrechen die Messung nicht:

- _suspendTaskId
- _disableScheduler (siehe Listenhandbücher der SIMOTION Geräte)

Die ermittelte Laufzeit ist ein Vielfaches des Servo-Takts; bei Laufzeiten kleiner als der Lagereglertakt wird als Messwert T#MIN (= T#0ms) zurückgegeben.

Die Funktion ist für alle Tasks zulässig. Die Messung wird jedoch nicht unterstützt von der IPOsynchronousTask und der ShutDownTask. Bei Aufrufen mit diesen Tasks wird als Messwert T#MIN (= T#0ms) zurückgegeben.

Deklaration

```
_getCurrentTaskIdRunTime (
                { id     : StructTaskId
                }
                ) : TIME
```

Eingangsparameter

id (optional)

Datentyp:	StructTaskId
Voreinstellung:	TaskId der aktuellen Task, in welcher die Funktion aufgerufen wird.

TaskId der Task, deren Laufzeit gemessen werden soll (siehe **Funktion _startTaskId**)

Rückgabewert

Datentyp:	TIME
T#MIN (= T#0ms = T#1ms * UDINT#0)	Messung wird nicht unterstützt oder ist noch nicht beendet.
größer T#MIN und kleiner T#MAX	Im vorangegangenen Durchlauf aufgetretene Laufzeit.
T#MAX (= T#49d_17h_2m_47s_295ms = T#1ms * UDINT#16#FFFF_FFFF)	TaskId ist ungültig

Siehe auch

Funktion _startTaskId (Seite 339)

Funktion _suspendTaskId (Seite 340)

7.3.5 Funktion _getAverageTaskIdRunTime

Diese Funktion liefert einen Durchschnittswert der Laufzeit der Task aus den letzten 10 vorangegangenen Durchläufen, einschließlich aller Unterbrechungen von höherpriorigen Tasks. Die Task wird über eine projektweit eindeutige TaskId vorgegeben (siehe Funktion **_startTaskId**).

Folgende Funktionen unterbrechen die Messung nicht:

- _suspendTask
- _disableScheduler (siehe Listenhandbücher der SIMOTION Geräte)

Die ermittelte Laufzeit ist ein Vielfaches des Servo-Takts; bei Laufzeiten kleiner als der Lagereglertakt wird als Messwert T#MIN (= T#0ms) zurückgegeben.

Die Funktion ist für alle Tasks zulässig. Die Messung wird jedoch nicht unterstützt von der IPOsynchronousTask und der ShutDownTask. Bei Aufrufen mit diesen Tasks wird als Messwert T#MIN (= T#0ms) zurückgegeben.

Deklaration

```
_getAverageTaskIdRunTime (    // nur Kurzform erlaubt
     { id    : : StructTaskId
     }
    ) : TIME
```

Eingangsparameter

id (optional)

Datentyp	StructTaskId
Voreinstellung:	TaskId der aktuellen Task, deren Laufzeit gemessen werden soll, wie im Ablaufsystem vorgegeben.

TaskId der Task, deren Laufzeit gemessen werden soll (siehe **Funktion _startTaskId**)

Rückgabewert

Datentyp:	TIME
T#MIN (= T#0ms = T#1ms * UDINT#0)	Messung wird nicht unterstützt oder ist noch nicht beendet.
größer T#MIN und kleiner T#MAX	Durchschnittlich aufgetretene Laufzeit.
T#MAX (= T#49d_17h_2m_47s_295ms = T#1ms * UDINT#16#FFFF_FFFF)	TaskId ist ungültig

Siehe auch

Funktion _startTaskId (Seite 339)

Funktion _suspendTaskId (Seite 340)

7.3.6 Funktionen für die genaue Laufzeitmessung von Tasks

Beschreibung

Mit folgenden Systemfunktionen können Sie die Zeit seit Systemstart µs-genau messen. Damit können Sie für Abschnitte innerhalb der Applikation die Zeit genau messen, um so z.B. die Applikation zu optimieren.

Folgende Funktionen können Sie verwenden:

- Funktion _getInternalTimeStamp (Seite 349)
- Funktion _getTimeDifferenceOfInternalTimeStamp (Seite 349)

7.3.7 Funktion _getInternalTimeStamp

Beschreibung

Die Funktion _getInternalTimeStamp liefert einen spezifischen internen Zeitstempel als UDINT zurück. Sie können jeweils einen Zeitstempel, z. B. am Anfang und am Ende einer Task setzen und diese dann in der Funktion _getTimeDifferenceOfInternalTimeStamp (Seite 349) als UDINT t1 und UDINT t2 verwenden.

```
_getInternalTimeStamp        :UDINT;
```

7.3.8 Funktion _getTimeDifferenceOfInternalTimeStamp

Beschreibung

Die Funktion _getTimeDifferenceOfInternalTimeStop liefert einen Zeitwert aus zwei internen Zeitstempeln. Die beiden Zeitstempel müssen Sie über die Funktion _getInternalTimeStamp (Seite 349) generieren (Rückgabewert UDINT).

```
_getTimeOfDifferenceOfInternalTimeStamps
    (
     UDINT t1,
     UDINT t2
    ):UDINT;
```

Der Startzeitpunkt des zu messenden Zeitintervalls ist t1, der Endzeitpunkt ist t2. Die Funktion liefert die Differenz (t2 - t1) als UDINT (in µs) zurück.

Anwendung

Mit der Funktion können Sie z. B. die Laufzeiten von Codeabschnitten oder z. B. in einer IPOsynchronen Task Kommunikationszeiten messen.

7.4 Funktionen zur Meldungsprogrammierung (AlarmS)

7.4.1 Allgemeines zur Meldungsprogrammierung

Sie können frei projektierte Meldungen (z. B. Fehlermeldungen) an die angemeldeten Anzeigegeräte versenden bzw. deren Status abfragen.

Im Einzelnen können Sie:

- Eine nicht quittierungspflichtige Meldung versenden (siehe **Funktion _alarmSId**)
- Eine quittierungspflichtige Meldung versenden (siehe **Funktion _alarmSqId**)
- Den Zustand einer Meldung und deren Quittierungszustand abfragen (siehe **Funktion _alarmScId**)
- Alle anstehenden Alarme auflisten (**siehe Funktion _getPendingAlarms**) (ab V4.1)
- Alarme auf "gehend" setzen (**siehe Funktionen _resetAlarmId und _resetAllAlarmId**) (ab V4.1)

Die Meldung wird bei diesen Funktionen über eine eindeutige AlarmId vorgegeben.

Beispiele zur Meldungsprogrammierung sind in Meldungen programmieren (Seite 451).

Funktionen zur Meldungsprogrammierung in MCC siehe Programmierhandbuch MCC (Kapitel Meldung gekommen und Meldung gegangen).

AlarmId

Die AlarmId zu einem projektierten Meldungsnamen erhalten Sie auf folgende Weise:

- Als Variable _alarm.name. Hierbei bedeutet:
  - _alarm bezeichnet den vordefinierten Namensraum für AlarmId (siehe **Namensräume** im ST-Programmierhandbuch). Der Bezeichner des Namensraums steht mit Punkt getrennt vor dem nachfolgenden Bezeichner.
  - *name* ist der Bezeichner der Meldung wie in SIMOTION SCOUT projektiert.
- Mit der Funktion _getAlarmId(name) (siehe **Funktion _getAlarmId**). Die Meldung wird über den projektierten Meldungsnamen vorgegeben:

ACHTUNG
Die Funktion _getAlarmId sowie Variablen der Form _alarm.*name* dürfen in Bibliotheken nicht verwendet werden.

Siehe auch

Funktion _getAlarmId (Seite 357)

Funktion _alarmScId (Seite 356)

Funktion _alarmSqId (Seite 353)

Funktion _alarmSId (Seite 351)

7.4.2 Funktion _alarmSId

Die Funktion generiert bei einer Pegeländerung des auslösenden Signals (Sig) eine nicht quittierungspflichtige Meldung, die an alle dafür angemeldeten Anzeigegeräte verschickt wird. Die zu generierende Meldung wird über eine projektweit eindeutige AlarmId vorgegeben (siehe AlarmId (Seite 453)).

Optional kann ein Begleitwert angehängt werden.

Maximal 40 Meldungen können gleichzeitig verarbeitet werden.

Deklaration

```
_alarmSId (
         Sig        : BOOL
         Ev_Id    : StructAlarmId
         { sd     : ANY_NUM, ANY_BIT
         }
         )    : DWORD
```

Eingangsparameter

Sig

Datentyp: BOOL

Das meldungsauslösende Signal wird folgendermaßen interpretiert:

Wenn das Signal – bezogen auf den letzten Aufruf mit dieser AlarmId – eine positive Flanke darstellt, wird eine kommende Meldung generiert. Eine kommende Meldung wird auch generiert, wenn das Signal beim erstmaligen Aufruf mit diesem Meldungsnamen den Zustand TRUE hat.

Wenn das Signal – bezogen auf den letzten Aufruf mit dieser AlarmId – eine negative Flanke darstellt, wird eine gehende Meldung generiert.

Ev_Id

Datentyp: StructAlarmId

AlarmId der zu generierenden Meldung (siehe AlarmId (Seite 453))

sd (optional)

Datentyp: ANY_NUM, ANY_BIT

Vorbelegung: 0

Begleitwert: Erlaubt sind hier alle elementaren Datentypen. Die Eingabe von Werten ist nicht erlaubt, nur Variablen oder vorher definierte Bezeichner für Konstanten von einem der erlaubten Datentypen.

Angabe des Begleitwerts ist erforderlich, wenn bei der Meldungsprojektierung in SIMOTION SCOUT ein Begleitwert definiert wurde.

Eine Prüfung mit dem Datentyp des in der Meldung projektierten Begleitwerts ist während des Compilierens nicht möglich.

Rückgabewert

Datentyp: DWORD

Der Rückgabewert informiert den Anwender über das Ergebnis des Aufrufs.

Die angegebenen Werte können als ODER-Verknüpfung zwischen der Konstanten ALARMS_ERROR und den Konstanten DSC_SVS_DEVICE_ALARMS_xxx dargestellt werden.

16#0000	Kein Fehler Bei kommender Meldung ist Eintrag in Meldeliste erfolgt. Bei gehender Meldung wurde Eintrag in der Meldeliste gestrichen.
16#8001	Meldungsname nicht zulässig. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_ILLEGAL_EVENT_ID
16#8002	Meldungsverlust durch Überlauf (kein Platz mehr in der Meldeliste.) Es sind alle 40 Einträge der Meldeliste belegt. Eintrag in Meldeliste ist nicht erfolgt. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_LAST_ENTRY_USED
16#8003	Meldungsverlust durch Überlauf (Signal noch nicht gesendet, Signal-Overflow.) Sendepuffer für Benachrichtigung der Clients ist noch durch das letzte Ereignis belegt. Eintrag in Meldeliste ist nicht erfolgt. Fehler tritt evtl. auch auf, wenn Funktionsaufrufe mit steigender und fallender Flanke kurz hintereinander kommen. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_LAST_SIGNAL_USED
16#8004	Doppelmeldung, Meldung abgewiesen (Aufruf mit Meldung zum 2. Mal hintereinander gekommen oder hintereinander gegangen.). Eintrag in Meldeliste ist nicht erfolgt. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_IV_CALL
16#8005	Kein Anzeigegerät angemeldet (Meldung wird trotzdem in Liste eingetragen.) ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_EVENT_ID_NOT_USED

16#8006	Meldungsname in niederpriorer Ebene bereits in Bearbeitung Tritt bei SIMOTION nicht auf. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_EVENT_ID_IN_USE
16#8007	Mit diesem Meldungsnamen wurde noch kein Auftrag gestartet (Erstaufruf mit Sig = FALSE.) Fallende Flanke (gehende Meldung) kam ohne vorherige steigende Flanke (kommende Meldung) Eintrag in Meldeliste ist nicht erfolgt. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_IV_FIRST_CALL
16#8008	Meldungsname bereits belegt. Tritt bei SIMOTION nicht auf. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_IV_SFC_TYP
16#8009	Interner Fehler ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_INTERNAL_ERROR
16#8010	Eintrag wurde zurückgewiesen; Meldequittierspeicher voll. Nur bei _alarmSqId ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_NO_ENTRY

Beispiel

Siehe Beispiel für die Meldungsgenerierung (Seite 455).

7.4.3 Funktion _alarmSqId

Die Funktion generiert bei einer Pegeländerung des auslösenden Signals (Sig) eine quittierungspflichtige Meldung, die an alle dafür angemeldeten Anzeigegeräte verschickt wird. Die zu generierende Meldung wird über eine projektweit eindeutige AlarmId vorgegeben (siehe AlarmId (Seite 453)).

Optional kann ein Begleitwert angehängt werden.

Maximal 40 Meldungen können gleichzeitig verarbeitet werden.

Deklaration

```
_alarmSqId (
         Sig    : BOOL
         Ev_Id    : StructAlarmId
         { sd     : ANY_NUM, ANY_BIT
         }
         )    : DWORD
```

Eingangsparameter

Sig

Datentyp: BOOL

Das meldungsauslösende Signal wird folgendermaßen interpretiert:

Wenn das Signal – bezogen auf den letzten Aufruf mit diesem Meldungsnamen – eine positive Flanke darstellt, wird eine kommende Meldung generiert. Eine kommende Meldung wird auch generiert, wenn das Signal beim erstmaligen Aufruf mit diesem Meldungsnamen den Zustand TRUE hat.

Wenn das Signal – bezogen auf den letzten Aufruf mit diesem Meldungsnamen – eine negative Flanke darstellt, wird eine gehende Meldung generiert.

Ev_Id

Datentyp: StructAlarmId

AlarmId der zu generierenden Meldung (siehe AlarmId (Seite 453))

sd (optional)

Datentyp: ANY_NUM, ANY_BIT

Vorbelegung: 0

Begleitwert: Erlaubt sind hier alle elementaren Datentypen. Die Eingabe von Werten ist nicht erlaubt, nur Variablen oder vorher definierte Bezeichner für Konstanten von einem der erlaubten Datentypen.

Angabe des Begleitwerts ist erforderlich, wenn bei der Meldungsprojektierung in SIMOTION SCOUT ein Begleitwert definiert wurde.

Eine Prüfung mit dem Datentyp des in der Meldung projektierten Begleitwerts ist während des Compilierens nicht möglich.

Rückgabewert

Datentyp: DWORD

Der Rückgabewert informiert den Anwender über das Ergebnis des Aufrufs.

Die angegebenen Werte können als ODER-Verknüpfung zwischen der Konstanten ALARMS_ERROR und den Konstanten DSC_SVS_DEVICE_ALARMS_xxx dargestellt werden.

16#0000	Kein Fehler Bei kommender Meldung ist Eintrag in Meldeliste erfolgt. Bei gehender Meldung wurde Eintrag in der Meldeliste gestrichen.
16#8001	Meldungsname nicht zulässig. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_ILLEGAL_EVENT_ID
16#8002	Meldungsverlust durch Überlauf (kein Platz mehr in der Meldeliste.) Es sind alle 40 Einträge der Meldeliste belegt. Eintrag in Meldeliste ist nicht erfolgt. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_LAST_ENTRY_USED

16#8003	Meldungsverlust durch Überlauf (Signal noch nicht gesendet, Signal-Overflow.) Sendepuffer für Benachrichtigung der Clients ist noch durch das letzte Ereignis belegt. Eintrag in Meldeliste ist nicht erfolgt. Fehler tritt evtl. auch auf, wenn Funktionsaufrufe mit steigender und fallender Flanke kurz hintereinander kommen. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_LAST_SIGNAL_USED
16#8004	Doppelmeldung, Meldung abgewiesen (Aufruf mit Meldung zum 2. Mal hintereinander gekommen oder hintereinander gegangen.). Eintrag in Meldeliste ist nicht erfolgt. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_IV_CALL
16#8005	Kein Anzeigegerät angemeldet (Meldung wird trotzdem in Liste eingetragen.) ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_EVENT_ID_NOT_USED
16#8006	Meldungsname in niederpriorer Ebene bereits in Bearbeitung Tritt bei SIMOTION nicht auf. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_EVENT_ID_IN_USE
16#8007	Mit diesem Meldungsnamen wurde noch kein Auftrag gestartet (Erstaufruf mit Sig = FALSE.) Fallende Flanke (gehende Meldung) kam ohne vorherige steigende Flanke (kommende Meldung) Eintrag in Meldeliste ist nicht erfolgt. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_IV_FIRST_CALL
16#8008	Meldungsname bereits belegt. Tritt bei SIMOTION nicht auf. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_IV_SFC_TYP
16#8009	Interner Fehler ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_INTERNAL_ERROR
16#8010	Eintrag wurde zurückgewiesen; Meldequittierspeicher voll. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_NO_ENTRY

Beispiel

Siehe Beispiel für die Meldungsgenerierung (Seite 455) Meldungen programmieren.

7.4.4 Funktion _alarmScId

Die Funktion zeigt den Zustand einer Meldung und deren Quittierungszustand an. Die Meldung wird über eine projektweit eindeutige AlarmId vorgegeben (siehe AlarmId (Seite 453)).

Deklaration

```
_alarmScId (
         Ev_Id    : StructAlarmId
         )    : DWORD
```

Eingangsparameter

Ev_Id

Datentyp: StructAlarmId

AlarmId der zu generierenden Meldung (siehe AlarmId (Seite 453))

Rückgabewert

Datentyp: DWORD

Der Rückgabewert informiert den Anwender über das Ergebnis des Aufrufs und über den Zustand einer Meldung. Nachfolgend Konstantenwerte und symbolische Konstanten, die Sie gleichberechtigt verwenden können.

Erste Abfrage nach Fehler, siehe auch Meldungen programmieren (Seite 451):

16#8000	Filter für Fehler ALARMS_ERROR
16#8001	Meldungsname nicht zulässig. ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_ILLEGAL_EVENT_ID

Zweite Abfrage nach Meldungszustand, siehe auch Meldungen programmieren (Seite 451):

16#0000	Meldung gegangen, nicht quittiert (a).
16#0001	Meldung gekommen, nicht quittiert (b). ALARMS_STATE
16#0010	Zu diesem Meldungsnamen gibt es keine Meldung. (3 Möglichkeiten: Meldung noch nie ausgelöst (1), Meldung über AlarmS ausgelöst, aber auch gegangen (2), Meldung über _AlarmSq ausgelöst, gegangen und am Anzeigegerät quittiert (3).)

16#0101	Meldung gekommen, quittiert (c). ALARMS_STATE OR ALARMS_QSTATE

Meldungen a–c lassen auf _alarmSqId bzw. _alarmSq schließen, Meldung b lässt auf _alarmScId bzw. _alarmSc schließen.

Beispiel

Siehe **Fehlernummer und Status einer Meldung abfragen (Rückgabewerte filtern)** auf Seite 5-261 in Kapitel 5.7.1.

7.4.5 Funktion _getAlarmId

Die Funktion generiert aus einem anwendungsspezifisch projektierten Meldungsnamen die AlarmId der Meldung. Diese AlarmId kann einer Variablen vom Datentyp StructAlarmId zugewiesen werden und als Eingangsparameter in folgenden Funktionen verwendet werden:

- Funktion _alarmSId
- Funktion _alarmSqId
- Funktion _alarmScId

ACHTUNG
Die Funktion darf in Bibliotheken nicht verwendet werden. Der Aufruf der Funktion darf nur in Kurzform geschehen, d. h. mit vollständiger Auflistung aller Parameterwerte, jedoch ohne Angabe der Formalparameter.

Deklaration

```
_getAlarmId (        // nur Kurzform erlaubt
        Al_Name : Alarm_Name // Name der Meldung
        ) : StructAlarmId
```

Eingangsparameter

Al_Name

Anwendungsspezifisch projektierter Meldungsname, der bei der Meldungsprojektierung in SIMOTION SCOUT angelegt wird.

Rückgabewert

Datentyp: StructAlarmId

Der Rückgabewert enthält die AlarmId der projektierten Meldung.

Siehe auch

Funktion _alarmSId (Seite 351)

Funktion _alarmScId (Seite 356)

AlarmID (Seite 453)

Funktion _alarmSqId (Seite 353)

7.4.6 Funktion _getPendingAlarms

Beschreibung

Die Funktion liefert entsprechend der maximalen Anzahl der Einträge in die Alarmliste max 40 anstehende Alarme zurück. (pending alarms). Die Anzahl der anstehenden Alarme wird in *numberOfPendingAlarms* angezeigt.

Der Typ des jeweiligen Alarms wird in *_type* angezeigt.

Der Status des Alarms wird in *state* angezeigt.

Syntax

```
_getPendingAlarms            :StructRetGetPendingAlarms

StructRetGetPendingAlarms    :STRUCT
    numberOfPendingAlarms    :UINT;
    alarm                    :Array[1..40] of StructPendingAlarmState;
END_STRUCT
StructPendingAlarmState      :STRUCT
    Id                       :StructAlarmId;
    _type                    :EnumAlarmIdType [ALARM_S |ALARM_SQ]
    State                    :EnumAlarmIdState [INCOMING|OUTGOING]
END_STRUCT
```

7.4.7 Funktionen _resetAlarmId und _reset_AllAlarmId

Beschreibung

Mit den Funktionen _resetAlarmId und _resetAllAlarmId können Sie eine oder alle Alarme auf "gehend" setzen. Die Quittierung der SQ-Alarme müssen Sie weiter über ein HMI oder den SCOUT durchführen.

Hinweis

"gehend" setzen und gleichzeitiges "Quittieren" der AlarmSQ ggfs. in einem 2. Schritt;

Mit dem Rücksetzen der Alarme werden die AlarmS gelöscht,

Die AlarmSQ müssen Sie selbst quittieren.

Einzelnen Alarm auf "gehend" setzen

```
_resetAlarmId    //( vgl. _alarmSId + Erweiterung, kein Fehler, Alarm
nicht anstehend, ID nicht vorhanden, interner Fehler )
    (
      Id         :StructAlarmId;
     ) : DWORD
```

Alle anstehenden Alarme auf "gehend" setzen

```
_resetAllAlarmId    : DWORD //( vgl. _alarmSId + Erweiterung, kein Fehler,
interner Fehler )
      (
      Id         :StructAlarmId;
     ) : DWORD
```

Programmierung allgemeiner Standardfunktionen 8

8.1 Programmierung allgemeiner Standardfunktionen - Überblick

SIMOTION stellt Ihnen für die Lösung häufig auftretender Aufgaben eine Reihe von Standardfunktionen zur Verfügung, die Sie in Ihren Quellen aufrufen können.

Hinweis

Der Aufruf einiger der folgenden Standardfunktionen darf nur in Kurzform erfolgen, d. h. mit vollständiger Auflistung aller Parameterwerte, jedoch ohne Angabe der Formalparameter:

- Ergebnis := Funktionsname (1. Parameterwert, 2. Parameterwert)
  statt
- Ergebnis := Funktionsname (Formalparameter1 := 1. Parameterwert, ...)

Dies ist bei der Beschreibung der jeweiligen Funktion vermerkt.

Zu den allgemeinen Datentypen (z. B. ANY_INT, ANY_BIT) siehe **Elementare Datentypen**, Tabelle Allgemeine Datentypen im ST-Programmierhandbuch.

Befehlsbibliothek verwenden

Die Befehlsbibliothek ist ein Register des Projektnavigators. Sie enthält die verfügbaren Systemfunktionen, Systemfunktionsbausteine und Operatoren.

Diese Elemente können Sie mit Drag&Drop aus der Befehlsbibliothek mit Drag&Drop ziehen:

- in das Fenster des ST-Editors
- in ein Netzwerk des KOP/FUP-Editors
- in den Befehl Systemfunktionsaufruf des MCC-Editors

Vergleich der Systemfunktionen zwischen SIMOTION und SIMATIC

Eine Gegenüberstellung der SIMATIC S7 und SIMOTION Systemfunktionen finden Sie auf der Utilities & Applications CD unter FAQs.

8.2 Numerische Standardfunktionen

8.2.1 Besonderheiten einer numerischen Funktion

Jede numerische Standardfunktion hat einen Eingangsparameter. Das Ergebnis ist immer der Rückgabewert. Die allgemeinen numerischen, die logarithmischen und die trigonometrischen Standardfunktionen spezifizieren jeweils eine Gruppe von numerischen Standardfunktionen durch die Funktionsnamen und die Datentypen.

8.2.2 Allgemeine numerische Standardfunktionen

Allgemeine numerische Standardfunktionen dienen:

- zur Berechnung des Absolutwerts einer Größe,
- zur Berechnung der Quadratwurzel einer Größe,
- zum Abschneiden einer Größe auf ihren ganzzahligen Anteil.

Tabelle 8- 1 Allgemeine numerische Standardfunktionen

Funktions-name	Eingangsparameter		Rückgabewert	Beschreibung
	Name	Datentyp	Datentyp	
ABS	in	ANY_NUM	ANY_NUM[1]	Absolutwert
SQRT	in	ANY_REAL	ANY_REAL[1]	Quadratwurzel
TRUNC	in	ANY_REAL	ANY_INT	Abschneiden des Wertes auf ganzzahligen Anteil (Richtung 0)
[1] Datentyp des Eingangsparameters *in*				

Die folgende Tabelle zeigt mögliche Aufrufe allgemeiner numerischer Standardfunktionen und die jeweiligen Ergebnisse:

Tabelle 8- 2 Aufrufe allgemeiner numerischer Standardfunktionen

Aufruf	Ergebnis
`Result:=ABS (-5);` `Result:=ABS (in := -5);`	`5`
`Result:=SQRT (81.0);` `Result:=SQRT (in := 81.0);`	`9.0`
`Result:=TRUNC (-3.141_59);` `Result:=TRUNC (in := -3.141_59);`	`-3`

8.2.3 Logarithmische Standardfunktionen

Logarithmische Standardfunktionen sind die Funktionen zur Berechnung eines Exponentialwertes oder eines Logarithmus.

Tabelle 8- 3 Logarithmische Standardfunktionen

Funktions-name	Eingangsparameter		Rückgabewert	Beschreibung
	Name	Datentyp	Datentyp	
EXP	in	ANY_REAL	ANY_REAL[1]	e^x (e-Funktion)
EXPD	in	ANY_REAL	ANY_REAL[1]	10^x
EXPT	in1	ANY_REAL[2]	ANY_REAL[3]	Potenzierung (siehe auch Operator ** in **Arithmetische Ausdrücke** im ST-Programmierhandbuch)
	in2	ANY_REAL		
LN	in	ANY_REAL	ANY_REAL[1]	Natürlicher Logarithmus
LOG	in	ANY_REAL	ANY_REAL[1]	Dekadischer Logarithmus

[1] Datentyp des Eingangsparameters *in*.

[2] Der Eingangsparameter *in1* muss größer Null sein.
Ausnahmen ab Version V4.1 des SIMOTION Kernels:
– Wenn *in2* eine Ganzzahl ist, kann *in1* auch kleiner Null sein.
– Wenn *in2* positiv ist, kann *in1* auch gleich Null sein.
Bis Version V4.0 des SIMOTION Kernels gilt: Wenn *in1* gleich Null ist, kann eine eventuelle Fehlermeldung mit der ExecutionFaultTask abgefangen werden.

[3] Datentyp des Eingangsparameters *in1*.

Die folgende Tabelle zeigt mögliche Aufrufe logarithmischer Standardfunktionen und die jeweiligen Ergebnisse:

Tabelle 8- 4 Aufrufe logarithmischer Standardfunktionen

Aufruf	Ergebnis
`Result := EXP (4.1);` `Result := EXP (in := 4.1);`	`60.3403 ...`
`Result := EXPD (3.0);` `Result := EXPD (in := 3.0);`	`1_000.0`

`Result := EXPT (2.0,4.0);`
`Result := EXPT (in1 := 2.0, in2 := 4.0);`
`Result := 2.0 ** 4.0;`
	`16.0`

`Result := LN (2.718_281);` `Result := LN (in := 2.718_281);`	`1.0`
`Result := LOG (245);` `Result := LOG (in := 245);`	`2.389_166`

8.2.4 Trigonometrische Standardfunktionen

Die dargestellten trigonometrischen Standardfunktionen erwarten und berechnen Größen von Winkeln im Bogenmaß.

Tabelle 8- 5 Trigonometrische Standardfunktionen

Funktions-name	Eingangsparameter		Rückgabewert	Beschreibung
	Name	Datentyp	Datentyp	
ACOS	in	ANY_REAL	ANY_REAL[1] [2]	Arcus-Cosinus (Hauptwert)
ASIN	in	ANY_REAL	ANY_REAL[1] [2]	Arcus-Sinus (Hauptwert)
ATAN	in	ANY_REAL	ANY_REAL[1] [2]	Arcus-Tangens (Hauptwert)
COS	in	ANY_REAL[2]	ANY_REAL[1]	Cosinus (Eingang Bogenmaß)
SIN	in	ANY_REAL[2]	ANY_REAL[1]	Sinus (Eingang Bogenmaß)
TAN	in	ANY_REAL[2]	ANY_REAL[1]	Tangens (Eingang Bogenmaß)
[1] Datentyp des Eingangsparameters *in* [2] Winkel im Bogenmaß				

Die folgende Tabelle zeigt mögliche Aufrufe trigonometrischer Standardfunktionen und die jeweiligen Ergebnisse:

Tabelle 8- 6 Aufrufe trigonometrischer Standardfunktionen

Aufruf	Ergebnis

`PI:= 3.141592; //PI ist eine Variable!`
`Result := SIN (PI / 6);`
`Result := SIN (in := PI / 6);`
	`0.5`

`Result := ACOS (0.5);` `Result := ACOS (in := 0.5);`	`1.047_197 //entspr. PI/3`

8.2.5 Bitstring-Standardfunktionen

Jede Bitstring-Standardfunktion hat zwei Eingangsparameter:

- in (Datentyp ANY_BIT):

Bitstring, an dem die Bitschiebeoperationen durchgeführt werden.

- n (Datentyp USINT):
  - Anzahl der zu rotierenden Stellen bei ROL und ROR
  - Anzahl der zu schiebenden Stellen bei SHL und SHR.

Das Ergebnis ist immer der Rückgabewert. Die folgende Tabelle zeigt die Funktionsnamen und die Datentypen der zwei Eingangsparameter und des Rückgabewertes.

Tabelle 8- 7 Bitstring-Standardfunktionen

Funktions-name	Eingangsparameter		Rückgabewert	Beschreibung
	Name	Datentyp	Datentyp	
ROL	in n	ANY_BIT USINT	ANY_BIT[1]	Der im Parameter *in* vorhandene Bitstring wird um so viele Stellen nach **links** rotiert, wie es der Inhalt des Parameters *n* angibt.
ROR	in n	ANY_BIT USINT	ANY_BIT[1]	Der im Parameter *in* vorhandene Bitstring wird um so viele Stellen nach **rechts** rotiert, wie es der Inhalt des Parameters *n* angibt.
SHL	in n	ANY_BIT USINT	ANY_BIT[1]	Der im Parameter *in* vorhandene Bitstring wird um so viele Stellen nach **links** geschoben und diese durch 0 ersetzt, wie es der Inhalt des Parameters *n* angibt[2].
SHR	in n	ANY_BIT USINT	ANY_BIT[1]	Der im Parameter *in* vorhandene Bitstring wird um so viele Stellen nach **rechts** geschoben und diese durch 0 ersetzt, wie es der Inhalt des Parameters *n* angibt[2].

[1] Datentyp des Eingangsparameters *in*
[2] Es werden nur die fünf niederwertigen Bits des Parameters n ausgewertet, so dass z. B.:
SHL (16#FFFF_FFFF,32) = 16#FFFF_FFFF
SHR (16#FFFF_FFFF,32) = 16#FFFF_FFFF

Hinweis

Werden als Eingangsparameter Zahlenwerte verwendet, wird der jeweils kleinstmögliche Datentyp angenommen (z. B. BOOL bei 1, BYTE bei 2).

Die folgende Tabelle zeigt mögliche Aufrufe von Bitstring-Standardfunktionen und die jeweiligen Ergebnisse.

Tabelle 8- 8 Beispiele für Aufrufe von Bitstring-Standardfunktionen

Aufruf	Ergebnis
`Result := ROL (2#1101_0011, 5);` `// 1. Parameter entspricht 211 dezimal`	`2#0111_1010` `(= 122 dezimal)`
`Result := ROR (2#1101_0011, 2);` `// 1. Parameter entspricht 211 dezimal`	`2#1111_0100` `(= 244 dezimal)`
`Result := SHL (2#1101_0011, 3);` `// 1. Parameter entspricht 211 dezimal`	`2#1001_1000` `(= 152 dezimal)`
`Result := SHR (2#1101_0011, 2);` `// 1. Parameter entspricht 211 dezimal`	`2#0011_0100` `(= 52 dezimal)`
`Result := SHL (1, 3);` `// 1. Parameter Datentyp BOOL`	`2#0000_0000` `(= 0 dezimal)`
`Result := SHL (2, 3);` `// 1. Parameter Datentyp BYTE`	`2#0001_0000` `(= 16 dezimal)`

8.3 Zugriffe auf Bits in Bitstrings

8.3.1 Funktion _getBit

Diese Funktion liefert den Wert des angegebenen Bits einer Bitstring-Variablen.

Deklaration

```
FUNCTION _getBit (          // Bitstring-Variable
        in    : ANY_BIT,    // Nummer des Bits
        n     : USINT
        )     : BOOL
```

Eingangsparameter

in

Datentyp ANY_BIT

Bitstring-Variable

n

Datentyp USINT

Nummer des Bits, dessen Wert ausgegeben werden soll.

Zulässige Werte: [0..7] für BYTE
[0..15] für WORD
[0..31] für DWORD

Bei der Angabe unzulässiger Werte liefert die Funktion den Rückgabewert FALSE.

Rückgabewert

Datentyp: BOOL

Wert des Bits

Beispiel

```
myBit := _getBit    (in    := myBitString,
                     n     := 5);
```

Die Anwendervariable *myBit* enthält das Bit 5 der Anwendervariablen *myBitString*.

Bitadressierung (ab V4.1)

Sie können die Bitnummer einer Bitstringvariablen (außer BOOL) über die Syntax einer Strukturadressierung angeben. Dabei ist die Angabe der Bitnummer als Integerzahl oder über symbolische Konstante vom Typ ANY_INT in den Grenzen der Bitstringlänge möglich. Die Funktion wird über den Compilerschalter "Spracherweiterungen zulassen" freigeschaltet

```
FUNCTION f : VOID
VAR CONSTANT
        BIT_7 : INT := 7;
END_VAR
VAR
        dw : DWORD;
    b: BOOL;
END_VAR
b := dw.BIT_7; // Zugriff auf Bit Nummer 7
b := dw.3;     // Zugriff auf Bit Nummer 3
b := dw.33;    // Übersetzungsfehler; Bitbreite nicht ausreichend!
END_FUNCTION
```

Hinweis

Bei der Anwendung der Bitstring-Adressierung auf I/O und Systemvariablen kann infolge des getrennten, von einer anderen Task unterbrechbaren Lese-, Operations- und Rückschreibvorgangs kein in jedem Fall konsistenter Zugriff zugesichert werden. Der Fehler wird vom System nicht erkannt, lesender Zugriff ist allerdings möglich.

8.3.2 Funktion _setBit

Diese Funktion liefert den Wert einer Bitstring-Variablen, bei der das angegebene Bit auf einen bestimmten Booleschen Wert (TRUE/FALSE) gesetzt wurde.

Deklaration

```
FUNCTION _setBit (
        in        : ANY_BIT,    // Bitstring-Variable
        n         : USINT,      // Nummer des Bits
        { value   : BOOL        // Wert des Bits
        }
    ) : ANY_BIT
```

Eingangsparameter

in

Datentyp: ANY_BIT

Bitstring-Variable

n

Datentyp: USINT

Nummer des Bits, dessen Wert gesetzt werden soll.

Zulässige Werte: [0..7] für BYTE
[0..15] für WORD
[0..31] für DWORD

Bei der Angabe unzulässiger Werte wird (ohne weitere Meldung) der unveränderte Wert der Bitstring-Variablen zurückgegeben.

value

(optional)

Voreinstellung: TRUE

Wert, der dem zu setzenden Bit zugewiesen wird.

Rückgabewert

Datentyp: ANY_BIT

Datentyp des Eingangsparameters *in*.

Wert der Bitstring-Variablen mit geändertem Bit.

Hinweis: Zur direkten Änderung der Bitstring-Variablen kann ihr der Rückgabewert zugewiesen werden.

Beispiel

```
myBitString := _setBit  (in    := myBitString,
                         n     := 5,
                         value := FALSE);
```

Es wird das Bit 5 der Anwendervariablen *myBitString* auf FALSE (logisch 0) gesetzt.

Bitadressierung (ab V4.1)

Sie können die Bitnummer einer Bitstringvariablen (außer BOOL) über die Syntax einer Strukturadressierung angeben. Dabei ist die Angabe der Bitnummer als Integerzahl oder über symbolische Konstante vom Typ ANY_INT in den Grenzen der Bitstringlänge möglich. Die Funktion wird über den Compilerschalter "Spracherweiterungen zulassen" freigeschaltet

```
FUNCTION f : VOID
VAR CONSTANT
        BIT_7 : INT := 7;
END_VAR
VAR
        dw : DWORD;
         b : BOOL;
         b = 1;
END_VAR
dw.BIT_7 := b;    // Bit Nummer 7 schreiben
dw.BIT_3 := b;    // Bit Nummer 3 schreiben
END_FUNCTION
```

Hinweis

Bei der Anwendung der Bitstring-Adressierung auf I/O und Systemvariablen kann infolge des getrennten, von einer anderen Task unterbrechbaren Lese-, Operations- und Rückschreibvorgangs kein in jedem Fall konsistenter Zugriff zugesichert werden. Der Fehler wird vom System nicht erkannt, lesender Zugriff ist allerdings möglich.

8.3.3 Funktion _toggleBit

Diese Funktion liefert den Wert einer Bitstring-Variablen, bei der das angegebene Bit invertiert ist.

Deklaration

```
FUNCTION _toggleBit (
        in        : ANY_BIT,    // Bitstring-Variable
        n         : USINT,      // Nummer des Bits
                ) : ANY_BIT
```

Eingangsparameter

in

Datentyp: ANY_BIT

Bitstring-Variable

n

Datentyp: USINT

Nummer des Bits, dessen Wert invertiert (von TRUE auf FALSE bzw. von FALSE auf TRUE gesetzt) werden soll.

Zulässige Werte: [0..7] für BYTE
[0..15] für WORD
[0..31] für DWORD

Bei der Angabe unzulässiger Werte wird (ohne weitere Meldung) der unveränderte Wert der Bitstring-Variablen zurückgegeben.

Rückgabewert

Datentyp: ANY_BIT

Datentyp des Eingangsparameters *in*.

Wert der Bitstring-Variablen mit invertiertem Bit.

Hinweis: Zur direkten Änderung der Bitstring-Variablen kann ihr der Rückgabewert zugewiesen werden.

Beispiel

```
myBitString := _toggleBit    (in    := myBitString, n    := 5);
```

Es wird das Bit 5 der Anwendervariablen *myBitString* invertiert.

8.4 Bitoperationen auf numerische Datentypen

Die nachfolgenden Funktionen ermöglichen Bitoperationen auf numerische Datentypen. Jedes Bit des Rückgabewertes wird aus den entsprechenden Bits der Eingangsparameter gebildet.

Tabelle 8- 9 Bitoperatoren auf numerische Datentypen

Funktions-name	Eingangsparameter		Rückgabewert	Beschreibung
	Name	Datentyp	Datentyp	
_NOT[1]	in	ANY_INT	ANY_INT[2]	Bitweise Negation
_AND	in1 in2	ANY_INT[3] ANY_INT[3]	ANY_INT[4]	Bitweise Konjunktion (UND-Verknüpfung): Ein Bit des Rückgabewerts ist nur dann 1, wenn alle entsprechenden Bits der Eingangsparameter 1 sind, sonst 0).
_OR	in1 in2	ANY_INT[3] ANY_INT[3]	ANY_INT[4]	Bitweise Disjunktion (ODER-Verknüpfung): Ein Bit des Rückgabewerts ist 1, wenn mindestens eines der entsprechenden Bits der Eingangsparameter 1 ist, sonst 0).

Funktions-name	Eingangsparameter		Rückgabewert	Beschreibung
	Name	Datentyp	Datentyp	
_XOR	n1 in2	ANY_INT[3] ANY_INT[3]	ANY_INT[4]	Bitweise exklusive Disjunktion (exklusive ODER-Verknüpfung): Ein Bit des Rückgabewerts ist 1, wenn genau eines der entsprechenden Bits der Eingangsparameter 1 ist, sonst 0).

[1] Der Aufruf der Funktion _NOT darf nur in Kurzform geschehen, d. h. mit vollständiger Auflistung aller Parameterwerte, jedoch ohne Angabe der Formalparameter.

[2] Datentyp des Eingangsparameters.

[3] Die Datentypen von *in1* und *in2* müssen implizit in einen gemeinsamen Datentyp vom allgemeinen Typ ANY_INT konvertiert werden können.

[4] Kleinster gemeinsamer Datentyp, in den die Eingangsparameter implizit konvertiert werden können.

8.5 String-Bearbeitung (ab V4.0)

8.5.1 Funktionen zur String-Bearbeitung

Die nachfolgenden Funktionen ermöglichen das Bearbeiten von Variablen des Datentyps STRING.

Tabelle 8- 10 Funktionen zur String-Bearbeitung

Funktions-name	Eingangsparameter		Rückgabewert	Beschreibung
	Name	Datentyp	Datentyp	
CONCAT	in1 in2	STRING[254] STRING[254]	STRING[254]	Hängt String in2 an String in1 an[1]. Fehlerflags: TSI#ERRNO := 1 wenn length(IN1)+length(IN2) > 254
DELETE	In l p	STRING[254] INT INT	STRING[254]	Löscht l Zeichen aus String in, beginnend an Position p[2 4 6]. Fehlerflags: TSI#ERRNO := 2 wenn L < 0, P < 1, P > length(IN)
FIND	in1 in2	STRING[254] STRING[254]	INT	Liefert Position, an der String in2 innerhalb des Strings in1 beginnt. Ergebnis ist 0, wenn String in2 nicht in String in1 enthalten ist. Fehlerflags: keine

	Eingangsparameter		Rückgabewert	
INSERT	in1 in2 p	STRING[254] STRING[254] INT	STRING[254]	Fügt String in2 in String in1 ein, beginnend an Position p[1 2 5 7]. Fehlerflags: TSI#ERRNO := 2 wenn P < 0, P > length(IN1) TSI#ERRNO := 1 wenn length(IN1)+length(IN2) > 254
LEFT	In l	STRING[254] INT	STRING[254]	Liefert die l ersten Zeichen im String in[2 3]. Fehlerflags: TSI#ERRNO := 2 wenn L < 0
LEN	in	INT[254]	INT[254]	Liefert Anzahl der Zeichen im String in. Fehlerflags: keine
MID	In l p	STRING[254] INT INT	STRING[254]	Liefert l Zeichen aus String in, beginnend an Position p[2 3]. Fehlerflags: TSI#ERRNO := 2 wenn L < 0, P < 1, P > length(IN)
REPLACE	in1 in2 l p	STRING[254] STRING[254] INT INT	STRING[254]	Ersetzt l Zeichen aus String in1 mit String in2, beginnend an Position p[2 4 7]. Fehlerflags: TSI#ERRNO := 2 wenn L < 0, P < 1, P > length(IN1) TSI#ERRNO := 1 wenn length(IN1)+length(IN2)-L > 254
RIGHT	In l	STRING[254] INT	STRING[254]	Liefert die l letzten Zeichen im String in[2 3]. Fehlerflags: TSI#ERRNO := 2 wenn L < 0

[1] Falls LEN (in1) + LEN (in2) > 254: String wird abgeschnitten.

[2] Falls l < 0 oder p < 0: Leerer String wird zurückgegeben.

[3] Falls l = 0 oder p = 0: Leerer String wird zurückgegeben.

[4] Falls l = 0 oder p = 0: String in bzw. in1 bleibt unverändert.

[5] Falls p = 0: String in1 wird an String in2 angehängt.

[6] Falls p > LEN(in) : String in bleibt unverändert.

[7] Falls p > LEN(in1): String in2 wird an String in1 angehängt

Tabelle 8- 11 Beispiele für Aufrufe der Funktionen zur String-Bearbeitung

Aufruf	Ergebnis
`A := CONCAT (in1 := 'ASTRING', in2 := '123');`	'ASTRING123'.

`A := DELETE (in1 := 'ASTRING', l := 2, p := 4);`
`A := DELETE (in1 := 'ASTRING', l := 2, p := 0);`
`A := DELETE (in1 := 'ASTRING', l := 2, p := 8);`
`A := DELETE (in1 := 'ASTRING', l := 0, p := 4);`
`A := DELETE (in1 := 'ASTRING', l := 10, p := 4);`
`A := DELETE (in1 := 'ASTRING', l := -1, p := 4);`
`A := DELETE (in1 := 'ASTRING', l := 2, p := -1);`
	'ASTNG'.
	'ASTRING'.
	'ASTRING'.
	'ASTRING'.
	'AST'.
	''.
	''.

`B := FIND (in1 := 'ASTRING', in2 := 'RI');` `B := FIND (in1 := 'ASTRING', in2 := 'RB');`	4. 0.

`A := INSERT (in1 := 'ASTRING', in2 := '123', p := 1);`
`A := INSERT (in1 := 'ASTRING', in2 := '123', p := 0);`
`A := INSERT (in1 := 'ASTRING', in2 := '123', p := 10);`
`A := INSERT (in1 := 'ASTRING', in2 := '123', p :=-1);`
	'A123STRING'.
	'123ASTRING'.
	'ASTRING123'.
	''.

`A := LEFT (in := 'ASTRING', l := 3);`
`A := LEFT (in := 'ASTRING', l := 10);`
`A := LEFT (in := 'ASTRING', l := -1);`
	'AST'.
	'ASTRING'.
	''.

`B := LEN (in := 'ASTRING');`	7.

`A := MID (in := 'ASTRING', l :=3, p :=2 );`
`A := MID (in := 'ASTRING', l :=3, p :=6 );`
`A := MID (in := 'ASTRING', l :=3, p :=8 );`
`A := MID (in := 'ASTRING', l :=3, p :=0 );`
	'STR'.
	'NG'.
	''.
	''.

`A := REPLACE (in1 := 'ASTRING', in2 := '123', l := 4, p := 2);`
`A := REPLACE (in1 := 'ASTRING', in2 := '123', l := 4, p := 1);`
`A := REPLACE (in1 := 'ASTRING', in2 := '123', l := 0, p := 2);`
`A := REPLACE (in1 := 'ASTRING', in2 := '123', l := 4, p := 0);`
`A := REPLACE (in1 := 'ASTRING', in2 := '123', l := 2, p := 10);`
`A := REPLACE (in1 := 'ASTRING', in2 := '123', l := 4, p := 5);`
`A := REPLACE (in1 := 'ASTRING', in2 := '123', l := 4, p := -1);`
`A := REPLACE (in1 := 'ASTRING', in2 := '123', l := -1, p : =2);`
	'A123NG'.
	'123ING'.
	'ASTRING'.
	'ASTRING'.
	'ASTRING123'.
	'ASTRI123'.
	''.
	''.

`A := RIGHT (in := 'ASTRING', l := 3);`
`A := RIGHT (in := 'ASTRING', l := 10);`
`A := RIGHT (in := 'ASTRING', l := -1);`
	'ING'.
	'ASTRING'.
	''.

Für Informationen über Konvertierungsfunktionen für STRINGs, siehe Funktionen zur Konvertierung von INT/REAL/LREAL- und STRING-Datentypen (Seite 382)

8.5.2 Fehlerauswertung bei der String-Bearbeitung

Beschreibung

Ein aufgetretener Fehler bei einer Stringfunktion wird in der Taskstartinfo für jede Task getrennt hinterlegt. Er ist damit im Task-Kontext realisiert und kann somit direkt anschließend in derselben Task, z.B. BackgroundTask entsprechend abgefragt werden.

```
Variable:TSI#ERRNO : DINT
```

Der Wert 0 kennzeichnet Fehlerfreiheit. Für die Stringfunktionen werden die Fehler in P (Position im String) und L (Anzahl Zeichen) getrennt von der Überschreitung der maximalen Stringlänge mit unterschiedlichen Werten gespeichert.

- Wert 0 für Fehlerfreiheit
- Wert 1 für Überschreitung der maximalen Stringlänge
- Wert 2 für P oder L außerhalb des gültigen Bereichs

Hinweis

Die TSI#ERRNO kann zu der Untersuchung der Stringlänge nicht benutzt werden (gilt für DINT_TO_STRING, UDINT_TO_STRING, REAL_TO_STRING und LREAL_TO_STRING).

In der TSI#ERRNO wird in diesem Fall (Überschreitung der Stringlänge) der Fehlercode 0 gesetzt.

Der von Ihnen angegebene String muss deshalb genügend groß angegeben werden, bzw. im Anwenderprogramm muss die Länge der Zahl vor der Konvertierung geprüft werden.

Rücksetzen des TSI#ERRNO-Wertes:

- der Fehler bleibt so lange in der Taskstartinfo stehen, bis der Inhalt von TSI#ERRNO explizit durch Sie rückgeschrieben wird.
- bei Ausführen einer neuen String-Funktion wird der Wert von TSI#ERRNO nur im Fehlerfall neu geschrieben, eine korrekt ausgeführte String-Funktion überschreibt eine ggfs. vorliegende Fehlerkennung in TSI#ERRNO nicht.
- beim Neustart der Task wird die Fehlerkennung zurückgesetzt.
- das zyklische Ausführen einer Task, z.B. IpoSynchrone Task überschreibt die Fehlerkennung nicht. Die Fehlerkennung bleibt wie der Inhalt von lokalen Variablen erhalten.

Tabelle 8- 12 Beispiel

```
VAR
  A : STRING [254];
END_VAR

A := DELETE (in := 'ASTRING',_ l:= -1, p := 4);
// Ergebnis ist leerer String
IF (TSI#ERRNO = 2)then  // Fehler bei Stringbearbeitung
// L < 0, P < 1, P > length(IN)
A := 'ERROR';
END_IF;
```

8.6 Standardfunktionen zur Daytentypkonvertierung

8.6.1 Funktionen zur Konvertierung von numerischen Datentypen und Bit-Datentypen

Eine explizite Konvertierung ist immer dann notwendig, wenn Informationsverlust möglich ist, z. B. durch Verkleinerung des Wertebereichs oder durch Verringerung der Genauigkeit, wie bei der Konvertierung von LREAL nach REAL.

Der Compiler gibt bei verlustbehafteten Konvertierungen Warnungen aus.

ACHTUNG
Das Ergebnis der Typkonvertierung kann zur Laufzeit des Programms zu Fehlern führen, es wird dann die bei der Taskkonfiguration eingestellte Fehlerreaktion ausgelöst, siehe Verarbeitungsfehler in Programmen (Seite 146). Besondere Vorsicht ist bei der Konvertierung von DWORD zu REAL geboten. Der Bitstring aus DWORD wird ungeprüft als REAL-Wert übernommen. Achten Sie selbst darauf, dass der Bitstring in DWORD dem Bitmuster einer normalisierten Gleitpunktzahl nach IEEE entspricht. Sie können hierzu die Funktionen _finite (Seite 394) und _isNaN (Seite 395) verwenden. Andernfalls kann ein Fehler (FPU-Exception (Seite 147)) ausgelöst werden, sobald der REAL-Wert erstmals bei einer Rechenoperation verwendet wird (z. B. im Programm oder beim Beobachten im Symbol-Browser).

Die explizite Datentyp-Konvertierung führen Sie mit Standardfunktionen durch, die in der folgenden Tabelle sind.

- **Eingangsparameter**

  Jede Funktion zur Konvertierung eines Datentyps hat genau einen Eingangsparameter; sein Name ist *in*.

- **Rückgabewert**

  Das Ergebnis ist immer der Rückgabewert der Funktion. In der folgenden Tabelle ist die jeweilige Konvertierungsregel angegeben.

- **Namensgebung**

  Da die Datentypen des Eingangsparameters und des Rückgabewerts aus dem jeweiligen Funktionsnamen hervorgehen, sind sie in der folgenden Tabelle nicht gesondert aufgelistet: z. B. bei Funktion BOOL_TO_BYTE ist der Datentyp des Eingangsparameters BOOL, der Datentyp des Rückgabewerts BYTE.

Tabelle 8- 13 Funktionen zur Konvertierung numerischer Datentypen und Bit-Datentypen

Funktionsname	Konvertierungsregel	Implizit möglich
BOOL_TO_BYTE	Übernehme als niederstwertiges Bit und fülle den Rest mit 0 auf.	ja
BOOL_TO_DWORD	Übernehme als niederstwertiges Bit und fülle den Rest mit 0 auf.	ja

Funktionsname	Konvertierungsregel	Implizit möglich
BOOL_TO_WORD	Übernehme als niederstwertiges Bit und fülle den Rest mit 0 auf.	ja
BOOL_VALUE_TO_DINT	Übernehme Booleschen Wert als DINT-Wert (0 oder 1)	nein
BOOL_VALUE_TO_INT	Übernehme Booleschen Wert als INT-Wert (0 oder 1)	nein
BOOL_VALUE_TO_LREAL	Übernehme Booleschen Wert als LREAL-Wert (0.0 oder 1.0)	nein
BOOL_VALUE_TO_REAL	Übernehme Booleschen Wert als REAL-Wert (0.0 oder 1.0)	nein
BOOL_VALUE_TO_SINT	Übernehme Booleschen Wert als SINT-Wert (0 oder 1)	nein
BOOL_VALUE_TO_UDINT	Übernehme Booleschen Wert als UDINT-Wert (0 oder 1)	nein
BOOL_VALUE_TO_UINT	Übernehme Booleschen Wert als UINT-Wert (0 oder 1)	nein
BOOL_VALUE_TO_USINT	Übernehme Booleschen Wert als USINT-Wert (0 oder 1)	nein
BYTE_TO_BOOL	Übernehme das niederstwertige Bit.	nein
BYTE_TO_DINT	Übernehme Bitstring als niederstwertiges Byte und fülle Rest mit 0 auf.	nein
BYTE_TO_DWORD	Übernehme Bitstring als niederstwertiges Byte und fülle Rest mit 0 auf.	ja
BYTE_TO_INT	Übernehme Bitstring als niederwertiges Byte und fülle Rest mit 0 auf.	nein
BYTE_TO_SINT	Übernehme Bitstring als SINT-Wert.	nein
BYTE_TO_UDINT	Übernehme Bitstring als niederstwertiges Byte und fülle Rest mit 0 auf.	nein
BYTE_TO_UINT	Übernehme Bitstring als niederwertiges Byte und fülle Rest mit 0 auf.	nein
BYTE_TO_USINT	Übernehme Bitstring als USINT-Wert.	nein
BYTE_TO_WORD	Übernehme Bitstring als niederwertiges Byte und fülle Rest mit 0 auf.	ja
BYTE_VALUE_TO_LREAL	Interpretiere Bitstring als USINT-Wert und übernehme diesen Wert.	nein
BYTE_VALUE_TO_REAL	Interpretiere Bitstring als USINT-Wert und übernehme diesen Wert.	nein
DINT_TO_BYTE	Übernehme das niederstwertige Byte als Bitstring.	nein
DINT_TO_DWORD	Übernehme Bitstring.	nein
DINT_TO_INT	Übernehme die 2 niederwertigen Bytes als INT-Wert.	nein
DINT_TO_LREAL	Übernehme Wert.	ja
DINT_TO_REAL	Übernehme Wert (Genauigkeit kann verloren gehen).	nein
DINT_TO_SINT	Übernehme das niederstwertige Byte als SINT-Wert.	nein
DINT_TO_UDINT	Übernehme Wert als Bitstring.	nein
DINT_TO_UINT	Übernehme die 2 niederwertigen Bytes als UINT-Wert.	nein
DINT_TO_USINT	Übernehme das niederstwertige Byte als USINT-Wert.	nein
DINT_TO_WORD	Übernehme die 2 niederwertigen Bytes als Bitstring.	nein
DINT_VALUE_TO_BOOL	FALSE (0), wenn DINT-Wert = 0; TRUE (1) sonst.	nein
DWORD_TO_BOOL	Übernehme das niederstwertige Bit.	nein
DWORD_TO_BYTE	Übernehme das niederstwertige Byte als Bitstring	nein
DWORD_TO_DINT	Übernehme Bitstring als DINT-Wert.	nein
DWORD_TO_INT	Übernehme die 2 niederwertigen Bytes als INT-Wert.	nein

Funktionsname	Konvertierungsregel	Implizit möglich
DWORD_TO_REAL	Übernehme Bitstring als REAL-Wert (Prüfung auf Gültigkeit der REAL-Zahl findet nicht statt!). Beachten Sie den wichtigen Hinweis: Achtung auf Seite 6-300.	nein
DWORD_TO_SINT	Übernehme das niederstwertige Byte als SINT-Wert.	nein
DWORD_TO_UDINT	Übernehme Bitstring als UDINT-Wert.	nein
DWORD_TO_UINT	Übernehme die 2 niederwertigen Bytes als UINT-Wert.	nein
DWORD_TO_USINT	Übernehme das niederstwertige Byte als USINT-Wert.	nein
DWORD_TO_WORD	Übernehme die 2 niederwertigen Bytes als Bitstring.	nein
DWORD_VALUE_TO_LREAL	Interpretiere Bitstring als UDINT-Wert und übernehme diesen Wert.	nein
DWORD_VALUE_TO_REAL	Interpretiere Bitstring als UDINT-Wert und übernehme diesen Wert (Genauigkeit kann verloren gehen).	nein
INT_TO_BYTE	Übernehme das niederwertige Byte als Bitstring.	nein
INT_TO_DWORD	Übernehme Bitstring als niederwertiges Wort (2 Bytes) und fülle Rest mit 0 auf.	nein
INT_TO_DINT	Übernehme Wert.	ja
INT_TO_LREAL	Übernehme Wert.	ja
INT_TO_REAL	Übernehme Wert.	ja
INT_TO_SINT	Übernehme das niederwertige Byte als SINT-Wert.	nein
INT_TO_USINT	Übernehme das niederwertige Byte als USINT-Wert.	nein
INT_TO_UDINT	Übernehme Bitstring als niederwertiges Wort (2 Bytes) und fülle Rest mit 0 auf.	nein
INT_TO_UINT	Übernehme Bitstring als UINT-Wert.	nein
INT_TO_WORD	Übernehme Bitstring.	nein
INT_VALUE_TO_BOOL	FALSE (0), wenn INT-Wert = 0; TRUE (1) sonst.	nein
LREAL_TO_DINT	Runde auf nächstliegende Ganzzahl[1].	nein
LREAL_TO_INT	Runde auf nächstliegende Ganzzahl[1].	nein
LREAL_TO_REAL	Übernehme Wert (Genauigkeit kann verloren gehen).	nein
LREAL_TO_SINT	Runde auf nächstliegende Ganzzahl[1].	nein
LREAL_TO_UDINT	Runde Betrag auf nächstliegende Ganzzahl[1].	nein
LREAL_TO_UINT	Runde Betrag auf nächstliegende Ganzzahl[1].	nein
LREAL_TO_USINT	Runde Betrag auf nächstliegende Ganzzahl[1].	nein
LREAL_VALUE_TO_BOOL	FALSE (0), wenn LREAL-Wert = 0.0; TRUE (1) sonst.	nein
LREAL_VALUE_TO_BYTE	Runde Betrag auf nächstliegende Ganzzahl[1] und übernehme Wert als Bitstring.	nein
LREAL_VALUE_TO_DWORD	Runde Betrag auf nächstliegende Ganzzahl[1] und übernehme Wert als Bitstring (nur für positive Zahlen[2])	nein
LREAL_VALUE_TO_WORD	Runde Betrag auf nächstliegende Ganzzahl[1] und übernehme Wert als Bitstring.	nein
REAL_TO_DINT	Runde auf nächstliegende Ganzzahl[1].	nein
REAL_TO_DWORD	Übernehme Bitstring.	nein
REAL_TO_INT	Runde auf nächstliegende Ganzzahl[1].	nein
REAL_TO_LREAL	Übernehme Wert.	ja

Funktionsname	Konvertierungsregel	Implizit möglich
REAL_TO_SINT	Runde auf nächstliegende Ganzzahl[1].	nein
REAL_TO_UDINT	Runde Betrag auf nächstliegende Ganzzahl[1].	nein
REAL_TO_UINT	Runde Betrag auf nächstliegende Ganzzahl[1].	nein
REAL_TO_USINT	Runde Betrag auf nächstliegende Ganzzahl[1].	nein
REAL_VALUE_TO_BOOL	FALSE (0), wenn REAL-Wert = 0.0; TRUE (1) sonst.	nein
REAL_VALUE_TO_BYTE	Runde Betrag auf nächstliegende Ganzzahl[1] und übernehme Wert als Bitstring.	nein
REAL_VALUE_TO_DWORD	Runde Betrag auf nächstliegende Ganzzahl[1] und übernehme Wert als Bitstring.	nein
REAL_VALUE_TO_WORD	Runde Betrag auf nächstliegende Ganzzahl[1] und übernehme Wert als Bitstring.	nein
SINT_TO_BYTE	Übernehme Bitstring.	nein
SINT_TO_DINT	Übernehme Wert.	ja
SINT_TO_DWORD	Übernehme Bitstring als niederstwertiges Byte und fülle Rest mit 0 auf.	nein
SINT_TO_INT	Übernehme Wert.	ja
SINT_TO_LREAL	Übernehme Wert.	nein
SINT_TO_REAL	Übernehme Wert.	nein
SINT_TO_UDINT	Übernehme Bitstring als niederstwertiges Byte und fülle Rest mit 0 auf.	nein
SINT_TO_UINT	Übernehme Bitstring als niederwertiges Byte und fülle Rest mit 0 auf.	nein
SINT_TO_USINT	Übernehme Bitstring.	nein
SINT_TO_WORD	Übernehme Bitstring als niederwertiges Byte und fülle Rest mit 0 auf.	nein
SINT_VALUE_TO_BOOL	FALSE (0), wenn SINT-Wert = 0; TRUE (1) sonst.	nein
StructAlarmId_TO_DINT	Übernehme Bitstring	nein
UDINT_TO_BYTE	Übernehme das niederstwertige Byte als Bitstring.	nein
UDINT_TO_DINT	Übernehme Bitstring.	nein
UDINT_TO_DWORD	Übernehme Bitstring.	nein
UDINT_TO_INT	Übernehme die 2 niederwertigen Bytes als INT-Wert.	nein
UDINT_TO_LREAL	Übernehme Wert.	ja
UDINT_TO_REAL	Übernehme Wert (Genauigkeit kann verloren gehen).	nein
UDINT_TO_SINT	Übernehme das niederstwertige Byte als SINT-Wert.	nein
UDINT_TO_UINT	Übernehme die 2 niederwertigen Bytes als UINT-Wert.	nein
UDINT_TO_USINT	Übernehme das niederstwertige Byte als USINT-Wert.	nein
UDINT_TO_WORD	Übernehme die 2 niederwertigen Bytes als Bitstring.	nein
UDINT_VALUE_TO_BOOL	FALSE (0), wenn UDINT-Wert = 0; TRUE (1) sonst.	nein
UINT_TO_BYTE	Übernehme das niederstwertige Byte als Bitstring.	nein
UINT_TO_DINT	Übernehme Wert.	ja
UINT_TO_DWORD	Übernehme Bitstring als niederwertiges Wort (2 Bytes) und fülle Rest mit 0 auf.	nein
UINT_TO_INT	Übernehme Bitstring.	nein

Funktionsname	Konvertierungsregel	Implizit möglich
UINT_TO_LREAL	Übernehme Wert.	nein
UINT_TO_REAL	Übernehme Wert.	ja
UINT_TO_SINT	Übernehme das niederwertige Byte als SINT-Wert.	nein
UINT_TO_UDINT	Übernehme Wert.	ja
UINT_TO_USINT	Übernehme das niederwertige Byte als USINT-Wert.	nein
UINT_TO_WORD	Übernehme Bitstring.	nein
UINT_VALUE_TO_BOOL	FALSE (0), wenn UINT-Wert = 0; TRUE (1) sonst.	nein
USINT_TO_BYTE	Übernehme Bitstring.	nein
USINT_TO_DINT	Übernehme Wert.	nein
USINT_TO_DWORD	Übernehme Bitstring als niederstwertiges Byte und fülle Rest mit 0 auf.	nein
USINT_TO_INT	Übernehme Wert.	ja
USINT_TO_LREAL	Übernehme Wert.	nein
USINT_TO_REAL	Übernehme Wert.	nein
USINT_TO_SINT	Übernehme Bitstring als SINT-Wert.	nein
USINT_TO_UDINT	Übernehme Wert.	ja
USINT_TO_UINT	Übernehme Wert.	ja
USINT_TO_WORD	Übernehme Bitstring als niederwertiges Byte und fülle Rest mit 0 auf.	nein
USINT_VALUE_TO_BOOL	FALSE (0), wenn USINT-Wert = 0; TRUE (1) sonst.	nein
WORD_TO_BOOL	Übernehme das niederstwertige Bit.	nein
WORD_TO_BYTE	Übernehme das niederwertige Byte als Bitstring.	nein
WORD_TO_DINT	Übernehme Bitstring als niederwertiges Wort (2 Bytes) und fülle Rest mit 0 auf.	nein
WORD_TO_DWORD	Übernehme Bitstring als niederwertiges Wort (2 Bytes) und fülle Rest mit 0 auf.	ja
WORD_TO_INT	Übernehme Bitstring als INT-Wert.	nein
WORD_TO_SINT	Übernehme das niederwertige Byte als SINT-Wert.	nein
WORD_TO_UDINT	Übernehme Bitstring als niederwertiges Wort (2 Bytes) und fülle Rest mit 0 auf.	nein
WORD_TO_UINT	Übernehme Bitstring.	nein
WORD_TO_USINT	Übernehme das niederwertige Byte als USINT-Wert.	nein
WORD_VALUE_TO_LREAL	Interpretiere Bitstring als UINT-Wert und übernehme diesen Wert.	nein
WORD_VALUE_TO_REAL	Interpretiere Bitstring als UINT-Wert und übernehme diesen Wert.	nein

[1] Bei gleichem Abstand zu den beiden nächstliegenden Ganzzahlen: Runde auf die nächstliegende gerade Ganzzahl

[2] Die Funktion LREAL_VALUE_TO_DWORD verhält sich analog zur Funktion LREAL_TO_UDINT. Es werden daher nur positive Zahlen konvertiert. Soll eine Floating-Point Zahl mit Vorzeichen in einen Bitstring gewandelt werden, so ist die Kombination aus LREAL_TO_DINT und DINT_TO_DWORD einzusetzen.

8.6.2 Funktionen zur Konvertierung von Datentypen für Datum und Zeit

Für Datentypen des Datums und der Zeit stehen nachstehende Standardfunktionen zur Verfügung.

Zu arithmetischen Ausdrücken mit den Datentypen für Datum und Zeit siehe **Arithmetische Ausdrücke** im ST-Programmierhandbuch.

Tabelle 8- 14 Standardfunktionen für Datum und Zeit

Funktionsname	Eingangsparameter		Rückgabewert	Beschreibung
	Name	Datentyp	Datentyp	
CONCAT_DATE_TOD	in1 in2	DATE TIME_OF_DAY	DATE_AND_TIME	Fasse DATE und TIME_OF_DAY zu DATE_AND_TIME (DT) zusammen.
DATE_AND_TIME_TO_TIME_OF_DAY oder DT_TO_TOD	in	DATE_AND_TIME	TIME_OF_DAY	Übernehme Tageszeitangabe
DATE_AND_TIME_TO_DATE oder DT_TO_DATE	in	DATE_AND_TIME	DATE	Übernehme Datumsangabe
INT_TO_TIME	in	INT	TIME	Übernehme Wert als Zeitangabe (Einheit ms) Die Funktion liefert für negative Werte keine brauchbaren Ergebnisse.
REAL_TO_TIME	in	REAL	TIME	Runde auf ganzzahligen Anteil und übernehme als Zeitangabe (Einheit ms)
TIME_TO_INT	in	TIME	INT	Übernehme Zeitangabe (Einheit ms) als Wert
TIME_TO_REAL	in	TIME	REAL	Übernehme Zeitangabe (Einheit ms) als Wert
TIME_TO_UDINT	in	TIME	UDINT	Übernehme Zeitangabe (Einheit ms) als Wert
UDINT_TO_TIME	in	UDINT	TIME	Übernehme Wert als Zeitangabe (Einheit ms). Die Funktion liefert für negative Werte keine brauchbaren Ergebnisse.

8.6.3 Funktionen zur Konvertierung von Aufzählungsdatentypen

Mit der Funktion ENUM_TO_DINT erhalten Sie den numerischen Wert eines Elements eines Aufzählungsdatentyps. Beim Aufruf der Funktion spezifizieren Sie den Datentyp des Aufzählungselements, indem Sie den Bezeichner des Datentyps und das Zeichen # vor den Bezeichner des Aufzählungselements stellen (*enum_type#enum_value*).

Tabelle 8- 15 Standardfunktionen für Datum und Zeit

Funktionsname	Eingangsparameter		Rückgabewert	Beschreibung
	Name	Datentyp	Datentyp	
ENUM_TO_DINT	in	beliebiger Aufzählungs-datentyp	DINT	Liefert numerischen Wert des Aufzählungselements

Alle durch den Anwender vereinbarten ENUM-Datentypen ordnen die Werte in aufsteigender Reihenfolge mit 0 beginnend. Die ENUM-Datentypen von TPs etc. weichen davon ab. Die Werte finden Sie in den entsprechenden Referenzhandbüchern aufgelistet.

8.6.4 Konvertierungen zwischen BYTE und STRING

Beschreibung

Die implizite Konvertierung von BYTE nach STRING und von STRING nach BYTE ermöglicht es ein Byte in einen String zu schreiben bzw. ein Byte aus einem String zu lesen (ASCII-Format, 1 Byte pro Zeichen).

Hinweis

Strings sind als ARRAY[1...stringsize] aufzufassen.

Umwandlung von BYTE nach STRING

Die Konvertierung erfolgt über direkte Zuweisung des Bytes auf das Stringelement n:

```
mySTring[n] := myByte;
```

Regeln:

1. Ist n > len(myString) und n < maxlen(myString), wird die Länge des Strings angepasst. Sämtliche Zeichen zwischen myString[len(myString)] ... myString[n] werden mit dem Wert "0" belegt.
2. Ist n > maxlen(myString), wird die TSI#ERRNO auf den Wert 1 gesetzt (Überschreitung der Maximalen Stringlänge) und myString bleibt unverändert.

Umwandlung von STRING nach BYTE

Die Konvertierung erfolgt über direkte Zuweisung des Stringelements k auf Byte:
```
myByte := myString[k];
```

Regel

1. Ist k > len(myString), wird die TSI#ERRNO auf den Wert 2 gesetzt (Wert außerhalb des gültigen Bereichs) und myByte mit dem Wert 0 belegt.

Anwendungsfälle

- Interne Variable auslesen und ggf. INT nach STRING oder DINT nach STRING konvertieren.
- STRING direkt an HMI ausgeben, z. B. WIN CC Op.
- STRING nach ARRAY of Byte konvertieren und so an HMI ausgeben

8.6.5 Funktionen zur Konvertierung von INT/REAL/LREAL- und STRING-Datentypen

Beschreibung

Die nachfolgenden Funktionen dienen zum Umrechnen von Zahlen in Strings zur Darstellung von Zahlen der Datentypen INT/REAL(LREAL.

Die explizite Datentyp-Konvertierung führen Sie mit Standardfunktionen durch, die in der folgenden Tabelle sind.

- Eingangsparameter

Jede Funktion zur Konvertierung eines Datentyps hat genau einen Eingangsparameter; sein Name ist IN.

- Rückgabewert

Das Ergebnis ist immer der Rückgabewert der Funktion.

Regeln für die Konvertierung von DINT/UDINT/REAL/LREAL nach STRING

- Werte werden linksbündig als Dezimalzahl bzw. Gleitpunktzahl in den STRING geschrieben.
- Eventuell vorhandene Vorzeichen werden vor die Ziffern geschrieben
- Reicht die Länge des Strings nicht aus, wird die Ziffernfolge rechts abgeschnitten (Überschreitung der Stringlänge).
- Bei der Konvertierung nach STRING erfolgt die Zahlendarstellung dezimal.

Regeln für die Konvertierung von STRING nach DINT/UDINT/REAL/LREAL

1. Führende Whitespaces werden nicht berücksichtigt, als Whitespaces werden Leerzeichen und Tabulatoren erkannt.
2. Die Konvertierung endet am Ende des Strings oder am ersten Zeichen, das keine Ziffer ist.
3. Enthält der String keine gültige Zahl oder wird der Wertebereich überschritten, wird TSI#ERRNO auf den Wert 3 (ungültige Zahlendarstellung) gesetzt und 0.0 (REAL/LREAL) ausgegeben.
4. Führende Nullen werden weggelassen.

Funktion	Bespiel	Beschreibung
DINT_TO_STRING	myString:=DINT_TO_STRING(myDint)	Regeln beachten
UDINT_TO_STRING	myString:=UDINT_TO_STRING(myUDint) myString:=UDINT_TO_STRING(myUsint)	Regeln beachten
STRING_TO_DINT	myDint:=STRING_TO_DINT(myString)	Eine gültige Zahl hat die Form [whitespace [sign][digits] Bei der Konvertierung von STRING, muss die Zahl dezimal vorliegen. Oktale und hexale Schreibweise wird nicht unterstützt.
STRING_TO_UDINT	myUDint:=STRING_TO_UDINT(myString)	Eine gültige Zahl hat die Form [whitespace [+][digits] Bei der Konvertierung von STRING, muss die Zahl dezimal vorliegen. Oktale und hexale Schreibweise wird nicht unterstützt.
REAL_TO_STRING LREAL_TO_STRING	myString:=REAL_TO_STRING(myReal) myString:=LREAL_TO_STRING(myLReal)	Regeln beachten
STRING_TO_REAL STRING_TO_LREAL	myReal:=STRING_TO_REAL(myString) myLReal:=STRING_TO_LREAL(myString)	Eine gültige Zahl hat die Form [whitespace [sign][digits][.digits][{ e I E }[sign]digits].

Anwendungsfall

- Ein HMI holt Texte aus dem Filesystem (Rezepturspeicher und lädt über eine Sequenz Text für Text in das Run Time System von SIMOTION (Unit Variable).
- Im Runtime System werden die Daten mit _saveUnitDataSet oder _exportUnitDataSet gespeichert. Ergänzung der STRINGs mit aktuellen SIMOTION-Daten (z. B. Istposition)
- Die Ausgabe der Texte erfolgt über die serielle Schnittstelle (z. B. ET200)

8.7 Konvertierung zwischen beliebigen Datentypen und Byte-Feldern

8.7.1 Allgemeines

Die nachfolgend angegebenen Funktionen ermöglichen das Wandeln von Variablen beliebigen Datentyps (elementare Datentypen, Standarddatentypen der Technologiepakete und Geräte, anwenderdefinierte Datentypen) in Byte-Felder und umgekehrt.

Weitere Informationen (z. B. zur Anordnung der Bytefelder, Anwendungsbeispiel) siehe Konvertieren zwischen beliebigen Datentypen und Byte-Feldern (Marshalling) (Seite 465)).

Diese Funktionen werden häufig verwendet, um definierte Übertragungsformate für den Datenaustausch zwischen verschiedenen Geräten zu schaffen (siehe auch Kommunikationsfunktionen (Seite 469)).

8.7.2 Funktion AnyType_to_BigByteArray, Funktion AnyType_to_LittleByteArray

Die Funktionen wandeln eine Variable beliebigen Datentyps (elementare Datentypen, Standarddatentypen der Technologiepakete und Geräte, anwenderdefinierte Datentypen) in ein Bytefeld.

- Bei AnyType_to_BigByteArray:

  Das Bytefeld ist vom Typ Big Endian (höchstwertiges Byte an niedriger Speicheradresse).

- Bei AnyType_to_LittleByteArray:

  Das Bytefeld ist vom Typ Little Endian (niedrigstwertiges Byte an niedriger Speicheradresse).

Ein optionaler konstanter Offset (Vorbelegung = 0) ist der Feldindex des ersten zu belegenden Elements im Feld. Er muss innerhalb der Feldgrenzen liegen.

Beim Übersetzen der ST-Quelle wird überprüft, ob der Offset innerhalb der Feldgrenzen liegt und ob die Variable vollständig auf das Bytefeld (zwischen Offset und oberer Feldgrenze) abgebildet werden kann.

Es werden nur die Elemente des Bytefelds mit Werten belegt, die von der zu konvertierenden Variablen überdeckt werden. Andere Elemente des Bytefelds bleiben unverändert.

Hinweis

Aufruf und Verarbeitung der Funktionen müssen entweder nur in einer Task erfolgen oder bei Einsatz von mehreren Tasks müssen diese bzgl. Aufruf und Verarbeitung durch geeignete Mittel synchronisiert werden (z.B. _testAndSetSemaphore, _releaseSemaphore). Wenn der Aufruf und die Verarbeitung des Ergebnisses in verschiedenen Tasks liegen, können undefinierbare Werte entstehen. Falls bei der Funktion BigByteArray_to_AnyType der Zielspeicher eine globale Variable ist, die in einer höherprioren Task ausgewertet wird, sollte die Konvertierung zunächst über eine temporäre Zielvariable erfolgen. Diese wird dann nach der Konvertierung auf die globale Variable umkopiert. Dies gilt auch bei einfachen Datentypen.

ACHTUNG
Variablen vom Datentyp BOOL (auch als Komponenten innerhalb eines strukturierten Datentyps) belegen im Bytefeld jeweils ein Byte.

Deklaration

```
FUNCTION AnyType_to_BigByteArray (
    anydata : ANY,               // beliebiger Datentyp
    { offset : DINT              // nur Konstante erlaubt
    }
    ) : ARRAY [..] OF BYTE       // Big Endian
FUNCTION AnyType_to_LittleByteArray (
        anydata    : ANY,        // beliebiger Datentyp
        { offset    : DINT       // nur Konstante erlaubt
        }
        ) : ARRAY [..] OF BYTE   // Little Endian
```

Eingangsparameter

anydata

Datentyp: ANY

Variable beliebigen Datentyps

Folgende Datentypen sind erlaubt:

- Technologieobjekte

offset

(optional)

Datentyp: DINT

Vorbelegung 0

Konstante, gibt das erste zu belegende Element im Feld an.

Rückgabewert

Datentyp: ARRAY [..] OF BYTE

- Bei AnyType_to_BigByteArray:

  in Anordnung Big Endian (höchstwertiges Byte an niedriger Speicheradresse).
- Bei AnyType_to_LittleByteArray:

  In Anordnung Typ Little Endian (niedrigstwertiges Byte an niedriger Speicheradresse)

Siehe auch

Konvertieren zwischen beliebigen Datentypen und Byte-Feldern (Marshalling) (Seite 465)

8.7.3 Funktion BigByteArray_to_AnyType, Funktion LittleByteArray_to_AnyType

Die Funktionen wandeln ein Bytefeld in eine Variable beliebigen Datentyps (elementare Datentypen, Systemdatentypen, anwenderdefinierte Datentypen).

- Bei BigByteArray_to_AnyType

  Das Bytefeld ist vom Typ Big Endian (höchstwertiges Byte an niedriger Speicheradresse).
- Bei LittleByteArray_to_AnyType

  Das Bytefeld ist vom Typ Little Endian (niedrigstwertiges Byte an niedriger Speicheradresse).

Ein optionaler konstanter Offset (Vorbelegung = 0) ist der Feldindex des ersten auszuwertenden Elements im Feld. Er muss innerhalb der Feldgrenzen liegen.

Beim Übersetzen der ST-Quelle wird überprüft, ob der Offset innerhalb der Feldgrenzen liegt und ob das Bytefeld (zwischen Offset und oberer Feldgrenze) die Variable vollständig überdeckt.

Hinweis

Aufruf und Verarbeitung der Funktionen müssen entweder nur in einer Task erfolgen oder bei Einsatz von mehreren Tasks müssen diese bzgl. Aufruf und Verarbeitung durch geeignete Mittel synchronisiert werden (z. B. _testAndSetSemaphore, _releaseSemaphore). Wenn der Aufruf und die Verarbeitung des Ergebnisses in verschiedenen Tasks liegen, können undefinierbare Werte entstehen.

ACHTUNG
Variablen vom Datentyp BOOL (auch als Komponenten innerhalb eines strukturierten Datentyps) wird aus dem Bytefeld jeweils ein Byte zugeordnet.

Deklaration

```
FUNCTION BigByteArray_to_AnyType (
    byteArray : ARRAY [..] OF BYTE, // Big Endian
    { offset : DINT                 // nur Konstante erlaubt
    }
    ) : ANY
FUNCTION LittleByteArray_to_AnyType (
     byteArray        : ARRAY [..] OF BYTE, // Little Endian
     { offset        : DINT                 // nur Konstante erlaubt
     }
    ) : ANY
```

Eingangsparameter

bytearray

Datentyp: ARRAY [..] OF BYTE

- Bei BigByteArray_to_AnyType

  in Anordnung Big Endian (höchstwertiges Byte an niedriger Speicheradresse)
- Bei LittleByteArray_to_AnyType

  in Anordnung Little Endian (niedrigstwertiges Byte an niedriger Speicheradresse)

offset

(optional)

Datentyp: DINT

Vorbelegung 0

Konstante, gibt das erste zu belegende Element im Feld an.

Rückgabewert

Datentyp: ANY

Beliebiger Datentyp. Folgende Datentypen sind nicht erlaubt:

- Technologieobjekte

ACHTUNG
Das Ergebnis der Marshalling-Funktionen kann bei Laufzeit des Programms zu Fehlern führen, es wird dann die bei der Taskkonfiguration eingestellte Fehlerreaktion ausgelöst, siehe Verarbeitungsfehler in Programmen (Seite 146). Besondere Vorsicht ist bei der Konvertierung von Byte-Feldern in den allgemeinen Datentyp ANY_REAL oder in Strukturen geboten, die diesen Datentyp enthalten. Der Bitstring aus dem Byte-Feld wird ungeprüft als ANY_REAL-Wert übernommen. Achten Sie selbst darauf, dass der Bitstring des Byte-Felds dem Bitmuster einer normalisierten Gleitpunktzahl nach IEEE 754 entspricht. Sie können hierzu die Funktionen _finite (Seite 394) und _isNaN (Seite 395) verwenden. Andernfalls kann ein Fehler (FPU-Exception (Seite 147)) ausgelöst werden, sobald der ANY_REAL-Wert erstmals bei einer Rechenoperation verwendet wird (z. B. im Programm oder beim Beobachten im Symbol-Browser).

Siehe auch

Konvertieren zwischen beliebigen Datentypen und Byte-Feldern (Marshalling) (Seite 465)

8.8 Zusammenfassen von Bitstring-Datentypen

8.8.1 Allgemeines zum Zusammenfassen von Bitstring Datentypen

Nachstehende Funktionen ermöglichen das Zusammenfassen von mehreren Variablen eines Bitstring-Datentyps zu einer Variablen eines übergeordneten Datentyps.

Die Umkehrfunktionen sind als Funktionsbausteine realisiert (siehe Zerlegen von Bitstring-Datentypen (Seite 506))

8.8.2 Funktion _BYTE_FROM_8BOOL

Diese Funktion fasst 8 Variablen vom Datentyp BOOL zu einer Variablen vom Datentyp BYTE zusammen.

Deklaration

```
FUNCTION _BYTE_FROM_8BOOL (
        { bit0,             // niederstwertige Bit
         bit1, bit2, bit3, bit4, bit5, bit6,
         bit7 : BOOL        // höchstwertiges Bit
        }
        ) : BYTE
```

Eingangsparameter

bit0 (optional)

...

bit7 (optional)

Datentyp: BOOL

Vorbelegung FALSE

Maximal 8 Variablen vom Datentyp BOOL, die zu einer Variablen vom Datentyp BYTE zusammengefasst werden sollen

bit0: niederwertigstes Bit

...

bit7: höchstwertiges Bit

Rückgabewert

Datentyp: BYTE

Aus den Eingangsparametern zusammengefasstes Byte.

8.8.3 Funktion _WORD_FROM_2BYTE

Diese Funktion fasst 2 Variablen vom Datentyp BYTE zu einer Variablen vom Datentyp WORD zusammen.

Deklaration

```
FUNCTION _WORD_FROM_2BYTE (
        { byte0,             // niederwertiges Byte
         byte1     : BYTE    // höherwertiges Byte
        }
    ) : WORD
```

Eingangsparameter

byte0 (optional)
byte1 (optional)

Datentyp: BYTE

Vorbelegung BYTE#0

Maximal 2 Variablen vom Datentyp BYTE, die zu einer Variablen vom Datentyp WORD zusammengefasst werden sollen

byte0: niederwertiges Byte
byte1: höherwertiges Byte

Rückgabewert

Datentyp: WORD

Aus den Eingangsparametern zusammengefasstes Wort.

8.8.4 Funktion _DWORD_FROM_2WORD

Diese Funktion fasst 2 Variablen vom Datentyp WORD zu einer Variablen vom Datentyp DWORD zusammen.

Deklaration

```
FUNCTION _DWORD_FROM_2WORD (
        { word0,          // niederwertiges Wort
         word1 : WORD     // höherwertiges Wort
        }
    ) : DWORD
```

Eingangsparameter

word0 (optional)
word1 (optional)

Datentyp: WORD

Vorbelegung WORD#0

Maximal 2 Variablen vom Datentyp WORD, die zu einer Variablen vom Datentyp DWORD zusammengefasst werden sollen

word0: niederwertiges Wort
word1: höherwertiges Wort

Rückgabewert

Datentyp: DWORD

Aus den Eingangsparametern zusammengefasstes Doppelwort.

8.8.5 Funktion _DWORD_FROM_4BYTE

Diese Funktion fasst 4 Variablen vom Datentyp BYTE zu einer Variablen vom Datentyp DWORD zusammen.

Deklaration

```
FUNCTION _DWORD_FROM_4BYTE (
        { byte0          // niederstwertiges Byte
         byte1, byte2,
         byte3 : BYTE    // höchstwertiges Byte
        }
    ) : DWORD
```

Eingangsparameter

byte0 (optional)
byte1 (optional)
byte2 (optional)
byte3 (optional)

Datentyp: BYTE

Vorbelegung BYTE#0

Maximal 4 Variablen vom Datentyp BYTE, die zu einer Variablen vom Datentyp DWORD zusammengefasst werden sollen

byte0: niederstwertiges Byte
...
byte3: höchstwertiges Byte

Rückgabewert

Datentyp: DWORD

Aus den Eingangsparametern zusammengefasstes Doppelwort.

8.9 Wandlung von Datentypen technologischer Objekte

8.9.1 Funktion AnyObject_to_Object

Die Funktion wandelt Variablen eines hierarchischen TO-Datentyps (driveAxis, posAxis, followingAxis) oder des allgemeinen Typs ANYOBJECT in einen kompatiblen TO-Datentyp.

Beispiele und weitere Informationen finden Sie in Wandlung von TO-Datentypen (Seite 133).

Deklaration

```
FUNCTION AnyObject_to_Object (
        in    : ANYOBJECT
    ) : ANYOBJECT
```

Eingangsparameter

in

Datentyp: ANYOBJECT

Variable eines TO-Datentyps (auch ANYOBJECT) oder eine TO-Instanz

Rückgabewert

Datentyp: ANYOBJECT

Wert ist TO#NIL, wenn Typkonvertierung nicht möglich

8.10 Funktionen zur Überprüfung von Gleitpunktzahlen

8.10.1 Funktion _finite

Die Funktion überprüft, ob der Eingangsparameter dem Bitmuster für unendlich nach IEEE 754 entspricht.

Zusammen mit der Funktion *_isNaN* wird sie insbesondere zur Prüfung verwendet, ob in Gleitpunktzahlen umgewandelte Bitstrings dem Bitmuster einer normalisierten Gleitpunktzahl nach IEEE entsprechen.

Dies verhindert, dass die bei der Taskkonfiguration eingestellte Fehlerreaktion ausgelöst (siehe **Verarbeitungsfehler in Programmen**) wird, sobald eine ungültige Gleitpunktzahl erstmals bei einer Rechenoperation verwendet wird (z. B. im Programm oder beim Beobachten im Symbol-Browser).

Deklaration

```
_finite (
        in : ANY_REAL
     ) : BOOL
```

Eingangsparameter

in

Datentyp: ANY_REAL

Variable, die überprüft werden soll

Rückgabewert

Datentyp:	BOOL
FALSE	Bitmuster für unendlich nach IEEE7 54
TRUE	kein Bitmuster für unendlich nach IEEE 754, d. h gültige Gleitpunktzahl innerhalb des Wertebereichs oder ungültiges Bitmuster (NaN Not a Number)

Beispiel

```
var_real := DWORD_TO_REAL (var_dword);
IF NOT _finite (var_real) OR _isNaN (var_real) THEN
    ; // Fehlerbehandlung
```

```
ELSE
    var_real := SQRT (var_real);
END_IF;
```

Siehe auch

Funktion _isNaN (Seite 395)

Verarbeitungsfehler in Programmen (Seite 146)

8.10.2 Funktion _isNaN

Die Funktion überprüft, ob der Eingangsparameter einem ungültigen Bitmuster einer Gleitpunktzahl nach IEEE 754 entspricht (is Not a Number NaN).

Zusammen mit der Funktion *_finite* wird sie insbesondere zur Prüfung verwendet, ob in Gleitpunktzahlen umgewandelte Bitstrings dem Bitmuster einer normalisierten Gleitpunktzahl nach IEEE entsprechen.

Dies verhindert, dass die bei der Taskkonfiguration eingestellte Fehlerreaktion ausgelöst (siehe **Verarbeitungsfehler in Programmen**) wird, sobald eine ungültige Gleitpunktzahl erstmals bei einer Rechenoperation verwendet wird (z. B. im Programm oder beim Beobachten im Symbol-Browser).

Deklaration

```
_isNaN (
        in : ANY_REAL
    ) : BOOL
```

Eingangsparameter

in

Datentyp: ANY_REAL

Variable, die überprüft werden soll

Rückgabewert

Datentyp:	BOOL
FALSE	gültiges Bitmuster oder Bitmuster für unendlich nach IEEE 754
TRUE	ungültiges Bitmuster nach IEEE 754 (NaN Not a Number)

Beispiel

```
var_real := DWORD_TO_REAL (var_dword);
IF NOT _finite (var_real) OR _isNaN (var_real) THEN
    ; // Fehlerbehandlung
ELSE
    var_real := SQRT (var_real);
END_IF;
```

Siehe auch

Funktion _finite (Seite 394)

Verarbeitungsfehler in Programmen (Seite 146)

8.11 Auswahlfunktionen

8.11.1 Funktion SEL

Binäre Auswahl

Der Rückgabewert ist einer der Eingangsparameter in0 oder in1 in Abhängigkeit vom Wert des Eingangsparameters g.

Deklaration

```
FUNCTION SEL (
        g    : BOOL,
        in0    : ANY,
        in1    : ANY
        ) : ANY
```

Eingangsparameter

g

Datentyp: BOOL

Auswahl des Eingangsparameters in0 bzw. in1

in0, in1

Datentyp: ANY

Die Eingangsparameter in0 und in1 müssen vom gleichen Datentyp sein oder durch implizite Konvertierung in einen gemeinsamen Datentyp gewandelt werden können (siehe **Konvertierung elementarer Datentypen** im ST-Programmierhandbuch).

Rückgabewert

Datentyp: ANY

Ausgewählter Eingangsparameter

in0 falls g = 0 (FALSE)

in1 falls g = 1 (TRUE)

Der Datentyp entspricht dem gemeinsamen Datentyp der Eingangsparameter in0 und in1.

8.11.2 Funktion MUX

Erweiterbare Multiplexfunktion

Der Rückgabewert ist einer der n Eingangsparameter in0 bis in*n*–1 in Abhängigkeit vom Wert des Eingangsparameters k.

Hinweis

Diese Beschreibung ist nicht für KOP/FUP relevant. Die entsprechende Funktion ist im KOP/FUP Programmierhandbuch beschrieben.

Deklaration

```
FUNCTION MUX (
        k    : ANY_INT,
        in0    : ANY,
        ...
        inn-1    : ANY
        ) : ANY
```

Eingangsparameter

k

Datentyp: ANY_INT

Auswahl des Eingangsparameters in0 bis in*n*–1.

Der Wertebereich ist abhängig von der Anzahl n der Eingangsparameter in0...in*n*–1: 0 ≤ k ≤ *n*-1.

Bei der Angabe unzulässiger Werte wird Eingangsparameter in0 ausgewählt.

in0
...
in*n*–1

Datentyp: ANY

Die Anzahl n der Eingangsparameter in0 und in*n*–1 ist variabel.

Alle Eingangsparameter in0 und in*n*–1 müssen vom gleichen Datentyp sein oder durch implizite Konvertierung in einen gemeinsamen Datentyp gewandelt werden können (siehe **Konvertierung elementarer Datentypen** im ST-Programmierhandbuch).

Werden beim Aufruf der Funktion die n Formalparameter in0 bis in*n*–1 angegeben, muss dies in aufsteigender, ununterbrochener Reihenfolge geschehen; bei z. B. 4 Eingangsparameter lauten die Bezeichner der Formalparameter: in0, in1, in2, in3.

Rückgabewert

Datentyp: ANY

Eingangsparameter in*m*, falls Eingangsparameter k den Wert *m* hat.

Der Datentyp entspricht dem gemeinsamen Datentyp der Eingangsparameter in0 bis in*n*–1.

8.11.3 Funktion MAX

Erweiterbare Maximum-Funktion.

Der Rückgabewert ist der Maximalwert der n Eingangsparameter in0 bis in*n*–1.

Hinweis

Diese Beschreibung ist nicht für KOP/FUP relevant. Die entsprechende Funktion ist im KOP/FUP Programmierhandbuch beschrieben.

Deklaration

```
FUNCTION MAX (
        in0         : ANY_ELEMENTARY,
        ...
        inn-1         : ANY_ELEMENTARY
        ) : ANY_ELEMENTARY
```

Eingangsparameter

in0
...
in*n*–1

Datentyp: ANY_ELEMENTARY

Die Anzahl n der Eingangsparameter in0 und in*n*–1 ist variabel.

Alle Eingangsparameter in0 und in*n*–1 müssen vom gleichen Datentyp sein oder durch implizite Konvertierung in den mächtigsten Datentyp gewandelt werden können (siehe **Konvertierung elementarer Datentypen** im ST-Programmierhandbuch).

Werden beim Aufruf der Funktion die n Formalparameter in0 bis in*n*–1 angegeben, muss dies in aufsteigender, ununterbrochener Reihenfolge geschehen; bei
z. B. 4 Eingangsparameter lauten die Bezeichner der Formalparameter: in0, in1, in2, in3.

Rückgabewert

Datentyp: ANY_ELEMENTARY

Maximum der Eingangsparameter

Der Datentyp entspricht dem mächtigsten Datentyp der Eingangsparameter in0 bis in*n*–1.

8.11.4 Funktion MIN

Erweiterbare Minimum-Funktion.

Der Rückgabewert ist der Minimalwert der n Eingangsparameter in0 bis in*n*–1.

Hinweis

Diese Beschreibung ist nicht für KOP/FUP relevant. Die entsprechende Funktion ist im KOP/FUP Programmierhandbuch beschrieben.

Deklaration

```
FUNCTION MIN (
        in0         : ANY_ELEMENTARY,
        ...
        inn-1         : ANY_ELEMENTARY
        ) : ANY_ELEMENTARY
```

Eingangsparameter

in0
...
in*n*–1

Datentyp: ANY_ELEMENTARY

Die Anzahl n der Eingangsparameter in0 und in*n*–1 ist variabel.

Alle Eingangsparameter in0 und in*n*–1 müssen vom gleichen Datentyp sein oder durch implizite Konvertierung in den mächtigsten Datentyp gewandelt werden können (siehe **Konvertierung elementarer Datentypen** im ST-Programmierhandbuch).

Werden beim Aufruf der Funktion die n Formalparameter in0 bis in*n*–1 angegeben, muss dies in aufsteigender, ununterbrochener Reihenfolge geschehen; bei z. B. 4 Eingangsparameter lauten die Bezeichner der Formalparameter: in0, in1, in2, in3.

Rückgabewert

Datentyp: ANY_ELEMENTARY

Minimum der Eingangsparameter

Der Datentyp entspricht dem mächtigsten Datentyp der Eingangsparameter in0 bis in*n*–1.

8.11.5 Funktion LIMIT

Begrenzungsfunktion

Eingangsparameter in wird auf Werte zwischen dem unteren Begrenzungswert *mn* und dem oberen Begrenzungswert *mx* beschränkt.

Deklaration

```
FUNCTION LIMIT (
        mn    : ANY_ELEMENTARY,
        in    : ANY_ELEMENTARY,
        mx    : ANY_ELEMENTARY
        ) : ANY_ELEMENTARY
```

Eingangsparameter

mn

Datentyp: ANY_ELEMENTARY

Unterer Begrenzungswert

Alle Eingangsparameter müssen vom gleichen Datentyp sein oder durch implizite Konvertierung in den mächtigsten Datentyp gewandelt werden können (siehe **Konvertierung elementarer Datentypen** im ST-Programmierhandbuch).

in

Datentyp: ANY_ELEMENTARY

Zu begrenzender Wert

Alle Eingangsparameter müssen vom gleichen Datentyp sein oder durch implizite Konvertierung in den mächtigsten Datentyp gewandelt werden können (siehe **Konvertierung elementarer Datentypen** im ST-Programmierhandbuch).

mx

Datentyp: ANY_ELEMENTARY

Oberer Begrenzungswert

Alle Eingangsparameter müssen vom gleichen Datentyp sein oder durch implizite Konvertierung in den mächtigsten Datentyp gewandelt werden können (siehe **Konvertierung elementarer Datentypen** im ST-Programmierhandbuch).

Rückgabewert

Datentyp: ANY_ELEMENTARY

MIN (MAX (in, mn), mx)

Der Datentyp entspricht dem mächtigsten Datentyp der Eingangsparameter.

8.12 Konsistenter Zugriff auf globale Variablen abgeleiteter Datentypen (UDT)

8.12.1 Allgemeines

Bei Zugriffen auf **globale** Variablen abgeleiteter Datentypen (siehe **Anwenderdefinierte Datentypen (UDT)** in den Programmierhandbüchern) muss der Anwender selbst für die Konsistenz der Daten sorgen, falls mehrere Tasks auf dieselben Anwendervariablen zugreifen (I/O-Variablen, Systemvariablen, Systemvariablen der Technologieobjekte, geräteglobale und unitglobale Anwendervariablen, siehe **Variablenmodell** in den Programmierhandbüchern).

Hinweis

Innerhalb einer Task ist konsistenter Datenzugriff immer gewährleistet.

Um das konsistente Schreiben und Lesen von globalen Variablen abgeleiteter Datentypen (UDT) sicherzustellen, können Sie mit Semaphoren arbeiten.

Eine globale Variable vom Datentyp DINT dient als Semaphore. Mit folgenden Funktionen ändern und prüfen Sie den Status des Semaphores:

- _testAndSetSemaphore
- _releaseSemaphore

Unter folgenden Bedingungen ist nun konsistenter Datenzugriff auf globale Variablen gewährleistet:

1. Alle Tasks signalisieren den Zugriff auf globale Variablen durch Setzen eines Semaphores.
2. Alle Tasks greifen auf globale Variablen nur bei freigegebenem Semaphore zu.

Weitere Informationen zum konsistenten Datenzugriffen und Semaphoren finden Sie in **Konsistentes Schreiben und Lesen von Variablen (Semaphoren).**

Siehe auch

Konsistenter Datenzugriff (Seite 457)

8.12.2 Funktion _testAndSetSemaphore

Mit dieser Funktion überprüfen Sie, ob das Semaphore gesetzt ist.

Nach Beenden der Funktion ist das Semaphore immer gesetzt. Weitere Aufrufe der Funktion (auch aus anderen Programmen) ergeben den Rückgabewert FALSE, solange bis die Funktion _releaseSemaphore (semaA) aufgerufen wird.

Deklaration

```
_testAndSetSemaphore     (
                          sema    : DINT
                         ) : BOOL
```

Eingangsparameter

sema

Datentyp DINT

Sema ist eine **globale** Variable vom Datentyp DINT; sie dient als Semaphore. Sie darf nicht indiziert sein. Falls sie Element eines Arrays ist, muss der Index bereits beim Compilieren festgelegt werden (z. B. a[2]).

Rückgabewert

Datentyp: BOOL

Anhand dieses Rückgabewertes können Sie feststellen, ob das Semaphore gesetzt ist:

TRUE	Semaphore ist freigegeben.
FALSE	Semaphore gesetzt.

Beispiel

Siehe **Konsistentes Schreiben und Lesen von Variablen (Semaphoren)**.

Siehe auch

Beispiel: Konsistenter Datenzugriff mit Semaphoren (Seite 458)

8.12.3 Funktion _releaseSemaphore

Mit dieser Funktion geben Sie das Semaphore frei. Der nächste Aufruf der Funktion _testAndSetSemaphore (auch aus anderen Programmen) ergibt den Rückgabewert TRUE.

Deklaration

```
_releaseSemaphore     (
                       sema    : DINT
                      ) : VOID
```

Eingangsparameter

sema

Datentyp: DINT

Sema ist eine globale Variable vom Datentyp DINT; sie dient als Semaphore. Sie darf nicht indiziert sein. Falls sie Element eines Arrays ist, muss der Index bereits beim Compilieren festgelegt werden (z. B. a[2]).

Rückgabewert

keiner

Beispiel

Siehe **Konsistentes Schreiben und Lesen von Variablen (Semaphoren)**.

Siehe auch

Beispiel: Konsistenter Datenzugriff mit Semaphoren (Seite 458)

8.13 Zugriffe auf Systemvariablen und Ein-/Ausgänge

8.13.1 Allgemeines zum Zugriff auf Systemvariablen und Ein-/Ausgängen

Die Funktionen _getSafeValue und _setSafeValue erlauben eine gesonderte Fehlerbehandlung bei Zugriffen auf Systemvariablen, Konfigurationsdaten oder I/O-Variablen. Das Verhalten im Fehlerfall ist abweichend vom konfigurierten Verhalten (siehe Fehler bei Zugriffen auf Systemvariablen und Konfigurationsdaten sowie auf I/O-Variablen für Direktzugriff (Seite 149)) einstellbar.

Die Systemfunktionen _getSafeValue und _setSafeValue sind sehr laufzeitintensiv. Nutzen Sie diese daher nur bedingt, z.B. in einer IF-Anweisung, um den Restart eines TOs abzuwarten.

Ab V4.1 können Sie für Zugriffe auf Systemvariablen und Konfigdaten (ab V4.1.3) im Fehlerfall (z.B. TO ist im Restart) auch "Ersatzwert" oder "letzter Wert" einstellen. Hierfür ist das Konfigurationsdatum restartInfo.behaviorInvalidSysvarAccess relevant. Siehe hierzu Systemvariablen (Seite 135) bzw. Konfigurationsdaten (Seite 139).

Die Funktion _getInOutByte ermöglicht einen lesenden Direktzugriff auf einzelne Bytes der Peripherie unter Angabe der Adresse.

Siehe auch

Fehler bei Zugriffen auf Systemvariablen und Konfigurationsdaten sowie auf I/O-Variablen für Direktzugriff (Seite 149)

Systemvariablen (Seite 135)

Konfigurationsdaten (Seite 139)

8.13.2 Funktion _getSafeValue

Diese Funktion liest die angegebene Systemvariable (bzw. Konfigurationsdatum) oder I/O-Variable und liefert den Wert in einer weiteren Variablen zurück.

Bei Verwenden dieser Funktion (an Stelle einer Variablenzuweisung) können Sie den Übergang in den Betriebszustand STOP verhindern, wenn bei Zugriffen auf Systemvariablen, Konfigurationsdaten oder I/O-Variablen ein Fehler auftritt (z. B. beim Restart eines Technologieobjekts oder bei Peripherieausfall).

ACHTUNG
Die Laufzeit dieser Funktion kann sehr lang sein. Die Funktion ist daher für die Verwendung in schnellen zyklischen Tasks nicht geeignet.

Fehlerreaktion festlegen

Über den Parameter *accessmode* steuern Sie die Fehlerreaktion:

- CONFIGURED (Vorbelegung): Die durch *restart.behaviorInvalidSysvarAccess* festgelegte Fehlerreaktion wird verwendet, siehe Fehler bei Zugriffen auf Systemvariablen und Konfigurationsdaten sowie auf I/O-Variablen für Direktzugriff (Seite 149)
- NO_CHANGE:
  - Systemvariablen und Konfigurationsdaten:

    Variable wird nicht gelesen und der Wert bleibt unbestimmt (siehe Systemvariablen (Seite 135)).
  - I/O-Variablen:

    Bei lesendem Zugriff (auf Ein- oder Ausgänge): Der letzte gültige Wert wird übernommen.
- DEFAULT_VALUE: Ersatzwert bzw. Grenzwert wird gelesen.
- STOP_DEVICE: SIMOTION Gerät geht in den Betriebszustand STOP. Bei Übergang in den Betriebszustand STOP wird die ExecutionFaultTask nicht gestartet.

Anhand des Rückgabewertes können Sie feststellen, ob der Zugriff erfolgreich war.

Deklaration

```
_getSafeValue (
         variable      : ANY,        //Systemvariable,
                                     // Konfigurationsdatum oder
                                     // I/O-Variable
         accessMode    : EnumAccessMode,
         getValue      : ANY
         )             : EnumSetAndGetSafeValue
```

Eingangsparameter

variable

Datentyp: ANY

Systemvariable, Konfigurationsdatum oder I/O-Variable, die gelesen werden soll.

accesssMode

Datentyp: EnumAccessMode

Verhalten, falls beim lesenden Zugriff ein Fehler auftritt.

```
TYPE EnumAccessMode : (
// Verhalten wie konfiguriert:
    CONFIGURED  // Systemvaribeln und Konfigdaten:
                // Verhalten abhängig vom Konfigdatum
                // restartInfo.behaviorInvalidSysvarAccess
                // Bei I/O-Variablen:
                // die beim Anlegen der I/O-Variable
                // festgelegte Strategie
// Von der Konfiguration abweichendes Verhalten:
    NO_CHANGE      // Bei Systemvariablen und
                   // Konfigurationsdaten: Variable
                   // nicht lesen.
                   // Bei I/O-Variablen: Letzten
                   // verfügbaren (gültigen) Wert
                   // übernehmen.
    DEFAULT_VALUE  // Projektierter Wert bzw.
                   // Ersatzwert bei I/O Variablen
    STOP_DEVICE    // Gerät schaltet in den Betriebszustand STOP.
END_TYPE
```

getvalue

Datentyp: ANY

Name der Variablen, in die der aktuelle Wert der Systemvariablen (bzw. Konfigurationsdatum) oder I/O-Variablen geschrieben wird.

Für den Datentyp gilt:

- er muss gleich dem Datentyp der zu lesenden Variablen sein oder
- der Datentyp der zu lesenden Variablen muss durch implizite Typkonvertierung (siehe **Störungen und Ereignisse auswerten**) in diesen Datentyp wandelbar sein.

Rückgabewert

Datentyp: EnumSetAndGetSafeValue

Anhand des Rückgabewerts können Sie feststellen, ob der Zugriff erfolgreich war.

```
TYPE EnumSetAndGetSafeValue : (
 // Zugriff erfolgreich:
 OK            // Zugriff war erfolgreich.
 // Zugriffsfehler:
, NO_CHANGE  // Bei Systemvariablen und Konfigdaten: Variable wurde
             // nicht gelesen, letzter Wert
             // Bei I/O-Variablen: Letzter
             // verfügbarer (gültiger) Wert wurde
             // übernommen.
, DEFAULT_VALUE    // Nur bei I/O-Variablen:
                   // Ersatzwert wurde übernommen.
, INVALID_VALUE    // Nur bei Konfigurationsdaten:
                   // Konfigurationsdatum wurde
                   // nicht gelesen, Defaultwert wird zurückgegeben,
                   // unzulässiger Parameter
                   //(accessMode = DEFAULT_VALUE).
, NO_ACCESS )      // Nur bei Konfigurationsdaten:
                   // Konfigurationsdatum wurde
                   // nicht gelesen, Defaultwert wird zurückgegeben
END_TYPE
```

Siehe auch

Störungen und Ereignisse auswerten (Seite 145)

Fehler beim Zugriff auf Systemdaten mit _get/_setSafeValue (Seite 184)

8.13.3 Funktion _setSafeValue

Das Verhalten der Funktion ändert sich ab V4.1.3. Sie schreibt den angegebenen Wert auf die Systemvariable, das Konfigurationsdatum oder die I/O-Variable und liefert optional den aktuell geschriebenen Wert in einer weiteren Variablen zurück. Das Verhalten im Fehlerfall ist abweichend vom konfigurierten Verhalten (siehe **Fehler bei Zugriffen auf Systemvariablen und Konfigurationsdaten**) einstellbar.

Bei Verwenden dieser Funktion (an Stelle einer Variablenzuweisung) können Sie den Übergang in den Betriebszustand STOP verhindern, wenn bei Zugriffen auf Systemvariablen, Konfigurationsdaten oder I/O-Variablen ein Fehler auftritt (z. B. beim Restart eines Technologieobjekts oder bei Peripherieausfall).

ACHTUNG
Die Laufzeit dieser Funktion kann sehr lang sein. Die Funktion ist daher für die Verwendung in schnellen zyklischen Tasks nicht geeignet.

Fehlerreaktion festlegen

Über den Parameter *accessMode* steuern Sie die Fehlerreaktion:

- CONFIGURED (Vorbelegung): Die in *restart.behaviorInvalidSysvarAccess* festgelegte Fehlerreaktion wird verwendet, siehe Fehler bei Zugriffen auf Systemvariablen und Konfigurationsdaten sowie auf I/O-Variablen für Direktzugriff (Seite 149)
- NO_CHANGE:
  - Systemvariablen und Konfigurationsdaten:

    Wert wird nicht geschrieben und bleibt unverändert bzw. letzter verfügbarer Wert.
  - I/O-Variablen:

    Bei schreibendem Zugriff (auf Ausgänge): Der Wert wird in die Variable geschrieben. Er wird allerdings erst am Ausgang wirksam, wenn der Ausgang wieder verfügbar ist.
- DEFAULT_VALUE: Ersatzwert bzw. Grenzwert wird geschrieben.
- STOP_DEVICE: SIMOTION Gerät geht in den Betriebszustand STOP. Bei Übergang in den Betriebszustand STOP wird die ExecutionFaultTask nicht gestartet.
- Wertebereich der Variable

Der Wert wird auf den Wertebereich begrenzt, unabhängig vom Parameter accessMode.

Anhand des Rückgabewertes können Sie feststellen, ob der Zugriff erfolgreich war.

Deklaration

```
_setSafeValue (
        variable     : ANY,        //Systemvariable,
                                  // Konfigurationsdatum oder
                                  // I/O-Variable
        value        : ANY,
        accessMode   : EnumAccessMode,
        setValue     : ANY
         ) : EnumSetAndGetSafeValue
```

Eingangsparameter

variable

Datentyp: ANY

Systemvariable, Konfigurationsdatum oder I/O-Variable, die beschrieben werden soll.

value

Datentyp: ANY

Wert, der in die Systemvariable (bzw. Konfigurationsdatum) oder I/O-Variable geschrieben werden soll. Er muss vom Datentyp der zu beschreibenden Variablen sein oder durch implizite Typkonvertierung (siehe **Störungen und Ereignisse auswerten**) in diesen Datentyp wandelbar sein.

accessMode

Datentyp: EnumAccessMode

Verhalten, falls beim schreibenden Zugriff ein Fehler auftritt:

```
TYPE EnumAccessMode : (
 // Verhalten wie konfiguriert:
     CONFIGURED        // Systemvariablen und Konfigdaten:
                       // Verhalten abhängig vom Konfigdatum
                       // restartInfo.behaviorInvalidSysvarAccess
                       // Bei I/O-Variablen:
                       // die beim Anlegen der I/O-Variablen
                       // festgelegte Strategie
 // Von der Konfiguration abweichendes Verhalten:
    , NO_CHANGE        // Bei Systemvariablen und
                       // Konfigurationsdaten: aktueller Wert
                       // wird nicht geschrieben.
                       // Bei I/O-Variablen
                       // Der im Parameter value übergebene
                       // Wert wird in die Variable
                       // geschrieben. Er wird am Ausgang
                       // erst wirksam, wenn der Ausgang
                       // wieder verfügbar ist.
, DEFAULT_VALUE
                       // Bei Systemvariablen und Konfigdaten:
                       // Die Variable wird nicht  beschrieben.
                       // Bei I/O-Variablen:
                       // Die Variable wird mit dem beim
                       // Anlegen der Variable festgelegten
                       // Ersatzwert beschrieben.
, STOP_DEVICE )        // Gerät schaltet in Betriebszustand
                       // STOP
END_TYPE
```

setvalue

Datentyp: ANY

Name einer Variablen, in die der aktuelle Wert der Systemvariablen, des Konfigurationsdatums bzw. der I/O-Variablen geschrieben wird.

Für den Datentyp gilt:

- er muss gleich dem Datentyp der zu beschreibenden Variablen sein oder
- der Datentyp der zu beschreibenden Variablen muss durch implizite Typkonvertierung (siehe **Störungen und Ereignisse auswerten**) in diesen Datentyp wandelbar sein.

Wenn z. B. 100 geschrieben werden sollte, aber nur 90 eintragbar war, enthält *setValue* den Wert 90.

Rückgabewert

Datentyp: EnumSetAndGetSafeValue

Anhand des Rückgabewerts können Sie feststellen, ob der Zugriff erfolgreich war.

```
TYPE EnumSetAndGetSafeValue : (
 // Zugriff erfolgreich:
 OK                // Zugriff war erfolgreich.
 // Zugriffsfehler:
, NO_CHANGE        // Bei Systemvariablen und Konfigdaten:
                   // Wert wird nicht geschrieben
                   // Bei I/O-Variablen: Der im Parameter
                   // value übergebene Wert wurde
                   // geschrieben. Er wird am Ausgang
                   // erst wirksam, wenn der Ausgang
                   // wieder verfügbar ist.
, DEFAULT_VALUE    // Bei Systemvariablen: Begrenzung
                   // aktiv, Grenzwert wurde
                   // geschrieben.
                   // Bei I/O-Variablen: Ersatzwert wurde
                   // geschrieben. Er wird am Ausgang
                   // erst wirksam, wenn der Ausgang
                   // wieder verfügbar ist.
, NO_ACCESS )      // Bei Konfigurationsdaten und Systemvariablen:
                   // Wert des Konfigurationsdatums wurde
                   // nicht geändert,
                   // Konfigurationsdatum nicht
                   // verfügbar.
END_TYPE
```

Siehe auch

Störungen und Ereignisse auswerten (Seite 145)

Fehler beim Zugriff auf Systemdaten mit _get/_setSafeValue (Seite 184)

Systemvariablen (Seite 135)

8.13.4 Funktion _getInOutByte

Diese Funktion ermöglicht einen lesenden Direktzugriff auf einzelne Bytes der Peripherie, indem die Adresse der Ein-/Ausgänge angegeben wird.

Bei einem Zugriffsfehler wird die PeripheralFaultTask nicht aufgerufen, sondern ein entsprechender Wert in der Komponente functionResult des Rückgabewertes zurückgegeben.

Wenn für die angegebene I/O-Adresse eine I/O-Variable (für Direktzugriff oder das Prozessabbild der zyklischen Task) (siehe **Direktzugriff und Prozessabbild der zyklischen Tasks** im ST-Programmierhandbuch) definiert ist und ein Ersatzwert angegeben ist, wird bei einem Zugriffsfehler dieser Ersatzwert zurückgegeben.

Durch Auswerten des Rückgabewertes kann man z. B. feststellen, ob der Adresse in der Hardware ein Ein- oder Ausgang zugeordnet ist.

ACHTUNG
Die Laufzeit dieser Funktion kann sehr lang sein. Die Funktion ist daher für die Verwendung in schnellen zyklischen Tasks nicht geeignet.

Deklaration

```
_getInOutByte (
            mode          : EnumInOutDirection,
            logAddress    : DINT
         )                 : StructRetGetInOutByte
```

Eingangsparameter

mode

Datentyp: EnumInOutDirection

Angabe, ob auf den Ein- oder Ausgang mit der logischen Adresse logAddress zugegriffen wird.

```
TYPE EnumInOutDirection : (
      IN            // Zugriff auf Eingang
  ,   OUT )        // Zugriff auf Ausgang
END_TYPE
```

logAddress

Datentyp: DINT

Logische Adresse des Ein- bzw. Ausgangs

Rückgabewert

Datentyp: StructRetGetInOutByte

Ergebnis des Funktionsaufrufs und Wert des gelesenen Bytes.

```
TYPE StructRetGetInOutByte : STRUCT
    functionResult    : DINT;    // Ergebnis des
                                 // Funktionsaufrufs
    value             : BYTE;    // Wert des gelesenen Bytes
 END_STRUCT;
END_TYPE
```

Mögliche Werte von functionResult (Ergebnis des Funktionsaufrufs):

0	kein Fehler, Zugriff o.k.
16#FFFF_FFFA	Nicht genügend Speicher verfügbar
16#FFFF_FFFE	Zugriffsfehler
16#FFFF_FFFF	Ein-/Ausgang nicht vorhanden

8.14 Datensicherung aus Anwenderprogramm

8.14.1 Allgemeines zum Speichern von Datensätzen aus dem Anwenderprogramm

Nachstehende Funktionen dienen:

- zum Speichern, Laden oder Initialisieren folgender Werte:
  - nicht remanente oder remanente unitglobale Variablen des Interface- oder Implementationsabschnitts einer Unit (z. B. ST-Quelle, MCC-Quelle)
  - nicht remanente oder remanente geräteglobale Variablen (über Projektnavigator deklariert)

  Durch Wahl der entsprechenden Funktion erfolgt die Datensicherung
  - binär: Der gesicherte Datensatz ist nach Änderung der Versionskennung des Datensegments nicht mehr lesbar.
  - im XML-Format: Der gesicherte Datensatz ist nach Änderung der Versionskennung des Datensegments lesbar
- zum Verwalten der Datensätze, in denen die Werte gesichert sind

Die Werte werden in einem Datensatz gesichert, der durch Angabe des Speicherorts, des Namens der Unit und einer Datensatznummer eindeutig gekennzeichnet ist.

Die Anwendung der Funktionen, insbesondere der Weiterschaltbedingung, ist in Datensicherung und -initialisierung aus Anwenderprogramm (Seite 459) ausführlich beschrieben.

Neben dem Speichern von remanenten Variablen als einzelne Datensätze besteht auch die Möglichkeit die gesamten remanenten Daten über die Funktion _savePersistentMemoryData auf die Speicherkarte zu sichern. Weitere Informationen, siehe Listenhandbuch *Sytemfunktionen -/variablen Geräte*.

8.14.2 Funktion _saveUnitDataSet

Die Werte der folgender Variablen werden binär als Datensatz gesichert:

- nicht remanente oder remanente unitglobale Variablen des Interface- oder Implementationsabschnitts einer Unit (z. B. ST-Quelle, MCC-Quelle, KOP/FUP-Quelle)
- nicht remanente oder remanente geräteglobale Variablen (über Projektnavigator deklariert)

Sie können den Ort wählen, an dem der Datensatz gespeichert wird:

- temporäre Datenablage (RAM-Disk), wird bei Netzausfall gelöscht
- permanente Datenablage (MemoryCard), bleibt bei Netzausfall erhalten

Achten Sie auf die Konsistenz der zu sichernden Daten (siehe Konsistentes Schreiben und Lesen von Variablen (Semaphoren) (Seite 457)).

Beim Aufruf in Kurzform müssen alle Parameter (auch alle optionalen Parameter) angegeben werden.

Mit der SCOUT-Funktion "Variable sichern" und "Variable wiederherstellen" ist es möglich, die mit _saveUnitDataSet gespeicherten Datensätze auch bei einem Versionswechsel oder geänderten Datensegmenten zu erhalten. Nähere Informationen finden Sie im Projektierungshandbuch SIMOTION SCOUT oder der Onlinehilfe.

Deklaration

```
_saveUnitDataSet (
            unitName        : STRING,
            id              : UDINT,
            storageType     : EnumDeviceStorageType,
            { path          : STRING,
             overwrite      : BOOL,
             nextCommand    : EnumNextCommandMode,
             dataScope      : EnumDeviceDataScope,
             kindOfData     : EnumDeviceKindOfData,
            }
             ): StructRetUnitDataSetCommand
```

Eingangsparameter

unitName

Datentyp: STRING

Name der Unit (z. B. ST-Quelle, MCC-Quelle, KOP/FUP-Quelle), deren Unit-Variablen gesichert werden sollen.

Der Name muss in Kleinbuchstaben und durch Hochkommata begrenzt angegeben werden (z. B. 'st_unit1').

Bei Angabe von '_device' werden die geräteglobalen Variablen gesichert.

id

Datentyp: UDINT

Nummer des Datensatzes, unter dem die Werte der Variablen gesichert werden (max.1_000_000 Datensätze je Unit).

storageType

Datentyp: EnumDeviceStorageType

Ort, an dem die Daten gespeichert werden

```
TYPE EnumDeviceStorageType : (
    TEMPORARY_STORAGE        // temporäre Datenablage
                             // (RAM-Disk),
                             // wird bei Netzausfall gelöscht
    , PERMANENT_STORAGE      // permanente Datenablage
                             // (MemoryCard), bleibt bei
```

```
                              // Netzausfall erhalten
    , USER_DEFINED )          // mit Pfadangabe
                              // (nur Voreinstellung zulässig)
END_TYPE
```

path (optional)
Datentyp: STRING
Voreinstellung: ' ' (leerer String)
Zielpfad, wenn storageType = USER_DEFINED.
Nur die Angabe des Voreinstellungswertes ist zulässig.

overwrite (optional)
Datentyp: BOOL
Voreinstellung: FALSE
wenn TRUE, wird vorhandener Datensatz überschrieben

nextCommand (optional)
Datentyp: EnumNextCommandMode
Voreinstellung: IMMEDIATELY
Weiterschaltung zum nächsten Befehl

```
TYPE EnumNextCommandMode : (
    IMMEDIATELY                 // sofort
    , WHEN_COMMAND_DONE )       // nach Beenden oder Abbruch
                                // des Befehls
END_TYPE
```

dataScope (optional)
Datentyp: EnumDeviceDataScope
Voreinstellung _INTERFACE
Angabe des Abschnitts, dessen Unit-Variablen gesichert werden sollen.

```
Type EnumDeviceDataScope : (
    _INTERFACE                          // Interfaceabschnitt
    , _IMPLEMENTATION                   // Implementationsabschnitt
    , _INTERFACE_AND_IMPLEMENTATION )   // Interface- und
                                        // Implementationsabschnitt
END_TYPE
```

Bei Angabe von unitName = '_device' sind für dataScope nur die Werte _INTERFACE oder _INTERFACE_AND_IMPLEMENTATION zulässig.

KindOfData (optional)

Voreinstellung NO_RETAIN_GLOBAL

Angabe, ob nicht remanente oder remanente globale Variable gesichert werden.

```
TYPE EnumDeviceKindOfData : (
    NO_RETAIN_GLOBAL              // nicht remanente Variablen
    , _RETAIN                     // remanente Variablen
    , ALL_GLOBAL )                // remanente und nicht remanente
                                  // Variablen
END_TYPE
```

Rückgabewert

Datentyp: StructRetUnitDataSetCommand

Der Rückgabewert ist eine Struktur vom Datentyp StructRetUnitDataSetCommand. In ihm sind zusammengefasst:

- eine Komponente functionResult : EnumDeviceUnitDataSetCommand.

  Diese gibt Ihnen Auskunft über Fehler und den aktuellen Zustand.
- eine Komponente handle : UDINT.

  Dies gibt Ihnen die Möglichkeit, mittels der Funktion _getStateOfUnitDataSetCommand (siehe **Funktion _getStateOfUnitDataSetCommand**) den aktuellen Zustand einer Datensicherungsfunktion abzufragen (nützlich insbesondere bei Weiterschaltbedingung IMMEDIATELY).

```
TYPE
 EnumDeviceUnitDataSetCommand: (
 DONE // Ausführung bzw. Start erfolgreich
 , ACTIVE                       // in Arbeit
 , INTERNEL_ERROR               // interner Fehler
                                // (rufen Sie bitte die Hotline an)
 , COMMAND_FAILED               // Befehl nicht ausführbar
 , NO_COMMAND_BUFFER_AVAILABLE  // Befehlspuffer voll
 , COMMAND_NOT_FOUND            // Befehl (Handle) nicht gefunden
 , DATASET_ID_NOT_VALID         // Datensatznummer ungültig
 , READ_ERROR                   // Daten-Lesefehler
```

```
                                // (Speichermedium defekt)
  , NO_STORAGE_AVAILABLE        // kein Speicherplatz verfügbar
  , ACCESS_DENIED               // Zugriff verweigert
                                // (fehlende Schreib-/Leserechte)
  ,DATASET_ALREADY_EXISTS       // Datensatz existiert bereits
  ,DATASET_NOT_FOUND            // Datensatz nicht gefunden
  ,VERSION_MISSMATCH            // Falsche Versionskennung
     // des zu importierenden Datenbereichs (z. B. Abschnitt der Unit
  ,UNIT_NOT_FOUND// Unit (z. B. ST-Quelle,
                                // MCC-Quelle) nicht gefunden
  ,DATA_INCOMPLETE              // Daten wurden unvollständig importiert
  ,DATA_MISMATCH                // Zu importierender
     // Datenbereich ist nicht im Datensatz enthalten
  ,SYMBOL_INFORMATION_NOT_AVAILABLE ) //Symbolinformation
                               // nicht verfügbar (an Programmquelle den
                               // OPC-XML-Export aktivieren
  StructRetUnitDataSetCommand
  : STRUCT
  functionResult : EnumDeviceUnitDataSetCommand;
  handle : UDINT;
  END_STRUCT;
 END_TYPE
```

Für Information über Datensicherung, siehe Anwendung der Datensicherung und -initialisierung aus Anwenderprogramm (Seite 459).

Siehe auch

Allgemeines zum Speichern von Datensätzen aus dem Anwenderprogramm (Seite 414)

Kommunikation über Ethernet mit UDP-Protokoll (Seite 477)

8.14.3 Funktion _loadUnitDataSet

Die Werte der folgender Variablen werden aus einem mit _saveUnitDataSet binär gespeicherten Datensatz geladen:

- nicht remanente oder remanente unitglobale Variable des Interface- oder Implementationsabschnitts einer Unit (z. B. ST-Quelle, MCC-Quelle, KOP/FUP-Quelle)
- nicht remanente oder remanente geräteglobale Variablen

Das Laden des Datensatzes ist nur möglich, wenn seit dem Speichern des Datensatzes die Versionskennungen aller zu ladenden Datensegmente (z. B. nicht remanente und remanente Variable des Interfaceabschnittes der Unit) unverändert geblieben sind. Zu Datensegmenten und deren Versionskennung siehe **Zeitpunkt der Variableninitialisierung** in den Programmierhandbüchern zu ST, MCC oder KOP/FUP.

Es kann eine Untermenge der im Datensatz gesicherten Datensegmente geladen werden (z. B. wenn sich die Versionskennungen einiger Datensegmente geändert haben).

Beim Aufruf in Kurzform müssen alle Parameter (auch alle optionalen Parameter) angegeben werden.

Mit der SCOUT-Funktion "Variable sichern" und "Variable wiederherstellen" ist es möglich, die mit _saveUnitDataSet gespeicherten Datensätze auch bei einem Versionswechsel oder geänderten Datensegmenten zu erhalten.

Nähere Informationen finden Sie im Projektierungshandbuch SIMOTION SCOUT oder der Onlinehilfe.

Deklaration

```
_loadUnitDataSet (
            unitName        : STRING,
            id              : UDINT,
            storageType     : EnumDeviceStorageType,
            { path          : STRING,
             nextCommand    : EnumNextCommandMode,
             dataScope      : EnumDeviceDataScope,
             kindOfData     : EnumDeviceKindOfData,
            }
            ): StructRetUnitDataSetCommand
```

Eingangsparameter

unitName

Datentyp: STRING

Name der Unit (z. B. ST-Quelle, MCC-Quelle), deren Unit-Variablen geladen werden sollen.

Der Name muss in Kleinbuchstaben und durch Hochkommata begrenzt angegeben werden (z. B. 'st_unit1').

Bei Angabe von ’_device’ werden die geräteglobalen Variablen geladen.

id

Datentyp: UDINT

Nummer des Datensatzes, aus dem die Werte der Variablen geladen werden (max.1_000_000 Datensätze je Unit).

storageType

Datentyp: EnumDeviceStorageType

Ort, an dem die Daten gespeichert werden

```
TYPE EnumDeviceStorageType : (
    TEMPORARY_STORAGE           // temporäre Datenablage
                                // (RAM-Disk),
```

```
                                // wird bei Netzausfall gelöscht
    , PERMANENT_STORAGE         // permanente Datenablage
                                // (MemoryCard), bleibt bei
                                // Netzausfall erhalten
    , USER_DEFINED )            // mit Pfadangabe, nur die
                                // Voreinstellung ist gültig
END_TYPE
```

path (optional)

Datentyp: STRING

Voreinstellung: ' ' (leerer String)

Zielpfad, wenn storageType = USER_DEFINED.

Nur die Angabe des Voreinstellungswertes ist zulässig.

nextCommand (optional)

Datentyp: EnumNextCommandMode

Voreinstellung: IMMEDIATELY

Weiterschaltung zum nächsten Befehl

```
TYPE EnumNextCommandMode : (
    IMMEDIATELY                 // sofort
    , WHEN_COMMAND_DONE )       // nach Beenden oder Abbruch
                                // des Befehls
END_TYPE
```

dataScope (optional)

Datentyp: EnumDeviceDataScope

Voreinstellung _INTERFACE

Angabe des Abschnitts, dessen Unit-Variablen gesichert werden sollen.

```
Type EnumDeviceDataScope : (
    _INTERFACE                          // Interfaceabschnitt
    , _IMPLEMENTATION                   // Implementationsabschnitt
    , _INTERFACE_AND_IMPLEMENTATION )   // Interface- und
                                        // Implementationsabschnitt
END_TYPE
```

Bei Angabe von unitName = '_device' sind für *dataScope* nur die Werte _INTERFACE oder _INTERFACE_AND_IMPLEMENTATION zulässig.

KindOfData (optional)

Voreinstellung NO_RETAIN_GLOBAL

Angabe, ob nicht remanente oder remanente globale Variablen gesichert werden.

```
TYPE EnumDeviceKindOfData : (
    NO_RETAIN_GLOBAL            // nicht remanente Variablen
    , _RETAIN                   // remanente Variablen
    , ALL_GLOBAL )              // remanente und nicht remanente
                                // Variablen
END_TYPE
```

Rückgabewert

Datentyp: StructRetUnitDataSetCommand

Der Rückgabewert ist eine Struktur vom Datentyp StructRetUnitDataSetCommand. In ihm sind zusammengefasst:

- eine Komponente *functionResult* : EnumDeviceUnitDataSetCommand.

  Diese gibt Ihnen Auskunft über Fehler und den aktuellen Zustand.
- eine Komponente *handle* : UDINT.

  Dies gibt Ihnen die Möglichkeit, mittels der Funktion _getStateOfUnitDataSetCommand (siehe **Funktion _getStateOfUnitDataSetCommand**) den aktuellen Zustand einer Datensicherungsfunktion abzufragen (nützlich insbesondere bei Weiterschaltbedingung IMMEDIATELY).

Weitere Informationen zu den Datentypen StructRetUnitDataSetCommand und EnumDeviceUnitDataSetCommand sehen Sie im Abschnitt Rückgabewert der Funktion _saveUnitDataSet (siehe **Funktion _saveUnitDataSet**).

Für Information über Datensicherung, siehe Anwendung der Datensicherung und -initialisierung aus Anwenderprogramm.

Siehe auch

Funktion _getStateOfUnitDataSetCommand (Seite 432)

Funktion _saveUnitDataSet (Seite 414)

Allgemeines zum Speichern von Datensätzen aus dem Anwenderprogramm (Seite 414)

Datensicherung und -initialisierung aus Anwenderprogramm - Funktionen und Hinweise (Seite 459)

8.14.4 Funktion _exportUnitDataSet

Die Werte der folgenden Variablen werden im XML-Format als Datensatz exportiert:

- nicht remanente unitglobale Variablen des Interface- und Implementationsabschnitts einer Unit (z. B. ST-Quelle, MCC-Quelle oder KOP/FUP-Quelle)
- remanente unitglobale Variablen des Interface- und Implementationsabschnitts einer Unit

Sie können den Ort wählen, an dem der Datensatz gespeichert wird:

- temporäre Datenablage (RAM-Disk), wird bei Netzausfall gelöscht
- permanente Datenablage (MemoryCard), bleibt bei Netzausfall erhalten

Nicht remanente oder remanente geräteglobale Variablen können nur mit der Funktion _saveUnitDataSet gesichert werden.

Stellen Sie sicher, dass für die betreffende Unit die Symbolinformationen der Unit-Variablen im SIMOTION Gerät zur Verfügung stehen. Aktivieren Sie deshalb an der Programmquelle in den lokalen Einstellungen des Compilers die Option **OPC-XML ermöglichen** (siehe Abschnitte Compileroptionen bzw. Einstellungen des ST-Compilers in den Programmierhandbüchern).

Achten Sie auf die Konsistenz der zu exportierenden Daten (siehe **Konsistentes Schreiben und Lesen von Variablen (Semaphoren)**).

Der exportierte Datensatz kann mit der Funktion _importUnitDataSet auch dann importiert werden, wenn sich inzwischen die Versionskennung des Datenbereichs (z. B. remanente Variablen des Interfaceabschnitts der Unit) geändert hat (z. B. durch Änderung der Datenstruktur).

Bei Aufruf in Kurzform müssen alle Parameter (auch alle optionalen Parameter) angegeben werden.

Deklaration

```
_exportUnitDataSet (
            unitName        : STRING,
            id              : UDINT,
            storageType     : EnumDeviceStorageType,
            { path          : STRING,
             overwrite      : BOOL
             nextCommand    : EnumNextCommandMode,
             dataScope      : EnumDeviceDataScope,
             kindOfData     : EnumDeviceKindOfData,
            }
             ): StructRetUnitDataSetCommand
```

Eingangsparameter

unitName

Datentyp: STRING

Name der Unit (z. B. ST-Quelle, MCC-Quelle), deren Unit-Variablen exportiert werden sollen.

Der Name muss in Kleinbuchstaben und durch Hochkommata begrenzt angegeben werden (z. B. 'st_unit1').

Die Angabe von '_device' (für den Export geräteglobale Variablen) ist hier nicht zulässig (nur bei _saveUnitDataSet möglich).

id

Datentyp: UDINT

Nummer des Datensatzes, unter dem die Werte der Variablen gesichert werden (max.1_000_000 Datensätze je Unit).

storageType

Datentyp: EnumDeviceStorageType

Ort, an dem die Daten gespeichert werden

Hinweis
Für die Einstellung USER_DEFINED ist nur die Voreinstellung zulässig.

```
TYPE EnumDeviceStorageType : (
 TEMPORARY_STORAGE        // temporäre Datenablage
                          // (RAM-Disk),
                          // wird bei Netzausfall gelöscht
 , PERMANENT_STORAGE      // permanente Datenablage
                          // (MemoryCard), bleibt bei
                          // Netzausfall erhalten
 , USER_DEFINED )         // mit Pfadangabe
                          // (nur Voreinstellung zulässig)
END_TYPE
```

path (optional)

Datentyp: STRING

Voreinstellung: ' ' (leerer String)

Zielpfad, wenn storageType = USER_DEFINED.

Hinweis

Es ist nur Angabe des Voreinstellungswertes zulässig. Andernfalls wird eine Fehlermeldung ausgegeben.

overwrite (optional)

Datentyp: BOOL

Voreinstellung: FALSE

wenn TRUE, wird vorhandener Datensatz überschrieben

nextCommand (optional)

Datentyp: EnumNextCommandMode

Voreinstellung: IMMEDIATELY

Weiterschaltung zum nächsten Befehl

```
TYPE EnumNextCommandMode : (
 IMMEDIATELY                  // sofort
 , WHEN_COMMAND_DONE )        // nach Beenden oder Abbruch
                              // des Befehls
END_TYPE
```

dataScope (optional)

Datentyp: EnumDeviceDataScope

Voreinstellung _INTERFACE

Angabe des Abschnitts, dessen Unit-Variablen exportiert werden sollen.

```
Type EnumDeviceDataScope : (
 _INTERFACE                          // Interfaceabschnitt
 , _IMPLEMENTATION                   // Implementationsabschnitt
 , _INTERFACE_AND_IMPLEMENTATION )   // Interface- und
                                     // Implementationsabschnitt
END_TYPE
```

KindOfData (optional)

Voreinstellung NO_RETAIN_GLOBAL

Angabe, ob nicht remanente oder remanente globale Variablen exportiert werden.

```
TYPE EnumDeviceKindOfData : (
 NO_RETAIN_GLOBAL            // nicht remanente Variablen
 , _RETAIN                   // remanente Variablen
 , ALL_GLOBAL )              // remanente und nicht remanente
                             // Variablen
END_TYPE
```

Rückgabewert

Datentyp: StructRetUnitDataSetCommand

Der Rückgabewert ist eine Struktur vom Datentyp StructRetUnitDataSetCommand. In ihm sind zusammengefasst:

- eine Komponente *functionResult* : EnumDeviceUnitDataSetCommand.

  Diese gibt Ihnen Auskunft über Fehler und den aktuellen Zustand.
- eine Komponente *handle* : UDINT.

  Dies gibt Ihnen die Möglichkeit, mittels der Funktion _getStateOfUnitDataSetCommand (siehe **Funktion _getStateOfUnitDataSetCommand**) den aktuellen Zustand einer Datensicherungsfunktion abzufragen (nützlich insbesondere bei Weiterschaltbedingung IMMEDIATELY).

Weitere Informationen zu den Datentypen StructRetUnitDataSetCommand und EnumDeviceUnitDataSetCommand sehen Sie im Abschnitt Rückgabewert der Funktion _saveUnitDataSet (siehe **Funktion _saveUnitDataSet**).

Für Information über Datensicherung, siehe Anwendung der Datensicherung und -initialisierung aus Anwenderprogramm (Seite 459).

Siehe auch

Konsistenter Datenzugriff (Seite 457)

Funktion _getStateOfUnitDataSetCommand (Seite 432)

Allgemeines zum Speichern von Datensätzen aus dem Anwenderprogramm (Seite 414)

8.14.5 Funktion _importUnitDataSet

Die Werte folgender Variablen werden aus einem Datensatz importiert, der mit _exportUnitDataSet exportiert wurde.

- nicht remanente unitglobale Variablen des Interface- und Implementationsabschnitts einer Unit (z. B. ST-Quelle, MCC-Quelle oder KOP/FUP-Quelle)
- remanente unitglobale Variablen des Interface- und Implementationsabschnitts einer Unit

Stellen Sie sicher, dass für die betreffende Unit die Symbolinformationen der Unit-Variablen des Interfaceabschnitts im SIMOTION Gerät zur Verfügung stehen. Aktivieren Sie deshalb an der Programmquelle in den lokalen Einstellungen des Compilers die Option **OPC-XML ermöglichen** (siehe Abschnitte Compileroptionen bzw. Einstellungen des ST-Compilers in den Programmierhandbüchern).

Der Import des Datensatzes ist auch möglich, wenn sich seit dem Speichern des Datensatzes die Versionskennung eines zu ladenden Datensegments (z. B. nicht remanente Variablen des Interfaceabschnittes der Unit) geändert hat:

- Nicht mehr vorhandene Variablen werden ignoriert.
- Bei hinzugekommenen Variablen bleibt der Wert unverändert.

- Bei Variablen mit geändertem Datentyp wird der Wert übernommen, wenn er in den neuen Datentyp konvertiert werden kann; andernfalls wird der Wert der Variablen beibehalten.
- Der Rückgabewert der Funktion ist DATA_INCOMPLETE.

Um unerwünschte Werte in Variablen zu vermeiden, können Sie die betreffenden Datensegmente vor dem Import des Datensatzes mit der Funktion _resetUnitData (siehe Listenhandbuch zu den Systemfunktionen der SIMOTION Geräte) initialisieren.

Zu Datensegmenten und deren Versionskennung siehe **Zeitpunkt der Variableninitialisierung** im ST-Programmierhandbuch.

Es kann eine Untermenge der im Datensatz exportierten Datensegmente importiert werden.

Bei Aufruf in Kurzform müssen alle Parameter (auch alle optionalen Parameter) angegeben werden.

Deklaration

```
_importUnitDataSet (
            unitName        : STRING,
            id              : UDINT,
            storageType     : EnumDeviceStorageType,
            { path          : STRING,
             nextCommand    : EnumNextCommandMode,
             dataScope      : EnumDeviceDataScope,
             kindOfData     : EnumDeviceKindOfData,
            }
             ): StructRetUnitDataSetCommand
```

Eingangsparameter

unitName

Datentyp: STRING

Name der Unit (z. B. ST-Quelle, MCC-Quelle), deren Unit-Variablen importiert werden sollen.

Der Name muss in Kleinbuchstaben und durch Hochkommata begrenzt angegeben werden (z. B. 'st_unit1').

Die Angabe von '_device' (für den Import geräeglobaler Variablen) ist hier nicht zulässig (nur bei _loadUnitDataSet möglich).

id

Datentyp: UDINT

Nummer des Datensatzes, unter dem die Werte der Variablen gesichert werden (max.1_000_000 Datensätze je Unit).

storageType

Datentyp: EnumDeviceStorageType

USERDEFINED (nur Voreinstellung zulässig)

Ort, an dem die Daten gespeichert werden

```
TYPE EnumDeviceStorageType : (
  TEMPORARY_STORAGE    // temporäre Datenablage
                       // (RAM-Disk), wird bei Netzausfall gelöscht
  , PERMANENT_STORAGE  // permanente Datenablage
                       // (MemoryCard), bleibt bei Netzausfall erhalten
END_TYPE
```

path (optional)

Datentyp: STRING

Voreinstellung: ' ' (leerer String)

Zielpfad, wenn storageType = USER_DEFINED.

Hinweis

Es ist nur die Angabe des Voreinstellungswertes zulässig. Andernfalls wird eine Fehlermeldung ausgegeben.

overwrite (optional)

Datentyp: BOOL

Voreinstellung: FALSE

wenn TRUE, wird vorhandener Datensatz überschrieben

nextCommand (optional)

Datentyp: EnumNextCommandMode

Voreinstellung: IMMEDIATELY

Weiterschaltung zum nächsten Befehl

```
TYPE EnumNextCommandMode : (
    IMMEDIATELY            // sofort
    , WHEN_COMMAND_DONE )  // nach Beenden oder Abbruch
                           // des Befehls
END_TYPE
```

dataScope (optional)

Datentyp: EnumDeviceDataScope

Voreinstellung _INTERFACE

Angabe des Abschnitts, dessen Unit-Variablen importiert werden sollen.

```
Type EnumDeviceDataScope : (
    _INTERFACE                          // Interfaceabschnitt
    , _IMPLEMENTATION                   // Implementationsabschnitt
    , _INTERFACE_AND_IMPLEMENTATION )   // Interface- und
                                        // Implementationsabschnitt
END_TYPE
```

KindOfData (optional)

Voreinstellung NO_RETAIN_GLOBAL

Angabe, ob nicht remanente oder remanente globale Variable importiert werden.

```
TYPE EnumDeviceKindOfData : (
    NO_RETAIN_GLOBAL            // nicht remanente Variablen
    , _RETAIN                   // remanente Variablen
    , ALL_GLOBAL )              // remanente und nicht remanente
                                // Variablen
END_TYPE
```

Wenn im exportierten Datensatz remanente und nicht remanente Variablen gespeichert sind, ist es möglich, die remanenten oder nicht remanenten Variablen selektiv zu importieren.

Rückgabewert

Datentyp: StructRetUnitDataSetCommand

Der Rückgabewert ist eine Struktur vom Datentyp StructRetUnitDataSetCommand. In ihm sind zusammengefasst:

- eine Komponente *functionResult* : EnumDeviceUnitDataSetCommand.

  Diese gibt Ihnen Auskunft über Fehler und den aktuellen Zustand.
- eine Komponente *handle* : UDINT.

  Dies gibt Ihnen die Möglichkeit, mittels der Funktion _getStateOfUnitDataSetCommand (siehe **Funktion _getStateOfUnitDataSetCommand**) den aktuellen Zustand einer Datensicherungsfunktion abzufragen (nützlich insbesondere bei Weiterschaltbedingung IMMEDIATELY).

Weitere Informationen zu den Datentypen StructRetUnitDataSetCommand und EnumDeviceUnitDataSetCommand sehen Sie im Abschnitt Rückgabewert der Funktion _saveUnitDataSet (siehe **Funktion _saveUnitDataSet**).

Für Information über Datensicherung, siehe Anwendung der Datensicherung und -initialisierung aus Anwenderprogramm (Seite 459).

Siehe auch

Funktion _getStateOfUnitDataSetCommand (Seite 432)

Allgemeines zum Speichern von Datensätzen aus dem Anwenderprogramm (Seite 414)

Funktion _loadUnitDataSet (Seite 418)

8.14.6 Funktion _deleteUnitDataSet

Ein einzelner Datensatz mit den gesicherten Werten folgender Variablen wird gelöscht:

- gesicherte nicht remanente oder remanente Unit-Variablen des Interface- oder Implementationsabschnitts einer Unit (z. B. ST-Quelle, MCC-Quelle)
- gesicherte nicht remanente oder remanente geräteglobale Variablen.
- exportierte nicht remanente oder remanente Unit-Variablen des Interfaceabschnitts einer Unit (z. B. ST-Quelle, MCC-Quelle)

Bei Aufruf in Kurzform müssen alle Parameter (auch alle optionalen Parameter) angegeben werden.

Deklaration

```
_deleteUnitDataSet (
            unitName       : STRING,
            id             : UDINT,
            storageType    : EnumDeviceStorageType,
            { path         : STRING,
             nextCommand   : EnumNextCommandMode,
            }
             ): StructRetUnitDataSetCommand
```

Eingangsparameter

unitName

Datentyp: STRING

Name der Unit (z. B. ST-Quelle, MCC-Quelle); der Name muss in Kleinbuchstaben und durch Hochkommata begrenzt angegeben werden (z. B. 'st_unit1').

Bei Angabe von '_device' wird ein Datensatz für geräteglobale Variablen gelöscht (möglich ab Version 3.2 des SIMOTIONKernels).

id

Datentyp: UDINT

Nummer des Datensatzes, der gelöscht werden soll (max.1_000_000 Datensätze je Unit).

storageType

Datentyp: EnumDeviceStorageType

Ort, von dem der Datensatz gelöscht werden soll.

Hinweis

Für die Einstellung USER_DEFINED ist nur die Voreinstellung zulässig.

```
TYPE EnumDeviceStorageType : (
    TEMPORARY_STORAGE          // temporäre Datenablage
                               // (RAM-Disk),
                               // wird bei Netzausfall gelöscht
    , PERMANENT_STORAGE        // permanente Datenablage
                               // (MemoryCard), bleibt bei
                               // Netzausfall erhalten
    , USER_DEFINED )           // mit Pfadangabe
                               // (in Vorbereitung)
END_TYPE
```

path (optional)

Datentyp: STRING

Voreinstellung: ' ' (leerer String)

Zielpfad, wenn storageType = USER_DEFINED.

Hinweis

Für die Einstellung USER_DEFINED ist nur die Voreinstellung zulässig.

nextCommand (optional)

Datentyp: EnumNextCommandMode

Voreinstellung: IMMEDIATELY

Weiterschaltung zum nächsten Befehl

```
TYPE EnumNextCommandMode : (
    IMMEDIATELY                  // sofort
    , WHEN_COMMAND_DONE )        // nach Beenden oder Abbruch
                                 // des Befehls
END_TYPE
```

Rückgabewert

Datentyp: StructRetUnitDataSetCommand

Der Rückgabewert ist eine Struktur vom Datentyp StructRetUnitDataSetCommand. In ihm sind zusammengefasst:

- eine Komponente *functionResult* : EnumDeviceUnitDataSetCommand.

  Diese gibt Ihnen Auskunft über Fehler und den aktuellen Zustand.
- eine Komponente *handle* : UDINT.

  Dies gibt Ihnen die Möglichkeit, mittels der Funktion _getStateOfUnitDataSetCommand (siehe Funktion **_getStateOfUnitDataSetCommand**) den aktuellen Zustand einer Datensicherungsfunktion abzufragen (nützlich insbesondere bei Weiterschaltbedingung IMMEDIATELY).

Weitere Informationen zu den Datentypen StructRetUnitDataSetCommand und EnumDeviceUnitDataSetCommand sehen Sie im Abschnitt Rückgabewert der Funktion _saveUnitDataSet (siehe Funktion **_saveUnitDataSet**).

Für Information über Datensicherung, siehe Anwendung der Datensicherung und -initialisierung aus Anwenderprogramm (Seite 459).

Siehe auch

Funktion _deleteAllUnitDataSets (Seite 435)

Allgemeines zum Speichern von Datensätzen aus dem Anwenderprogramm (Seite 414)

Funktion _exportUnitDataSet (Seite 422)

8.14.7 Funktion _getStateOfUnitDataSetCommand

Sie liefert den Status der Funktionen zur Datensicherung.

Deklaration

```
_getStateOfUnitDataSetCommand (
            handle        : UDINT
            )             : EnumDeviceUnitDataSetCommand
```

Eingangsparameter

handle

Datentyp: DINT

Das Handle der Datensicherungsfunktion, deren Status abgefragt werden soll. Dieses Handle haben Sie als Komponente des Rückgabewertes der Datensicherungsfunktion erhalten.

Rückgabewert

Datentyp: EnumDeviceUnitDataSetCommand

Auskunft über Fehler und den aktuellen Zustand.

Weitere Informationen zum Datentyp EnumDeviceUnitDataSetCommand sehen Sie im Abschnitt Rückgabewert der Funktion _saveUnitDataSet (siehe **Funktion _saveUnitDataSet**).

Für Informationen über Datensicherung, siehe Anwendung der Datensicherung und -initialisierung aus Anwenderprogramm (Seite 459).

Siehe auch

Funktion _exportUnitDataSet (Seite 422)

Allgemeines zum Speichern von Datensätzen aus dem Anwenderprogramm (Seite 414)

8.14.8 Funktion _checkExistingUnitDataSet

Es wird überprüft, ob der angegebene Datensatz mit den gesicherten Werten folgender Variablen auf dem Speichermedium vorhanden ist:

- gesicherte nicht remanente oder remanente Unit-Variablen des Interface- oder Implementationsabschnitts einer Unit (z. B. ST-Quelle, MCC-Quelle)
- gesicherte nicht remanente oder remanente geräteglobale Variablen.
- exportierte nicht remanente oder remanente Unit-Variablen des Interfaceabschnitts einer Unit (z. B. ST-Quelle, MCC-Quelle)

Bei Aufruf in Kurzform müssen alle Parameter (auch alle optionalen Parameter) angegeben werden.

Deklaration

```
_checkExistingUnitDataSet (
            unitName       : STRING,
            id             : UDINT,
            storageType    : EnumDeviceStorageType,
            { path         : STRING,
             nextCommand   : EnumNextCommandMode,
            }
            )              : StructRetUnitDataSetCommand
```

Eingangsparameter

unitName

Datentyp: STRING

Name der Unit (z. B. ST-Quelle, MCC-Quelle); der Name muss in Kleinbuchstaben und durch Hochkommata begrenzt angegeben werden (z. B. 'st_unit1').

Bei Angabe von '_device' wird ein Datensatz für geräteglobale Variablen überprüft (möglich ab Version 3.2 des SIMOTION Kernels).

id

Datentyp: UDINT

Nummer des Datensatzes (max. 1_000_000 Datensätze je Unit).

storageType

Datentyp: EnumDeviceStorageType

Ort, an dem die Daten gespeichert sind.

```
TYPE EnumDeviceStorageType : (
    TEMPORARY_STORAGE          // temporäre Datenablage
                               // (RAM-Disk),
```

```
                                // wird bei Netzausfall gelöscht
    , PERMANENT_STORAGE         // permanente Datenablage
                                // (MemoryCard), bleibt bei
                                // Netzausfall erhalten
    , USER_DEFINED )            // mit Pfadangabe
                                // (Nur Voreinstellung ist zulässig)
END_TYPE
```

path (optional)

Datentyp: STRING

Voreinstellung: ' ' (leerer String)

Zielpfad, wenn storageType = USER_DEFINED.

Nur die Angabe des Voreinstellungswertes ist zulässig.

nextCommand (optional)

Datentyp: EnumNextCommandMode

Voreinstellung: IMMEDIATELY

Weiterschaltung zum nächsten Befehl

```
TYPE EnumNextCommandMode : (
    IMMEDIATELY                 // sofort
    , WHEN_COMMAND_DONE )       // nach Beenden oder Abbruch
                                // des Befehls
END_TYPE
```

Rückgabewert

Datentyp: StructRetUnitDataSetCommand

Der Rückgabewert ist eine Struktur vom Datentyp StructRetUnitDataSetCommand. In ihm sind zusammengefasst:

- eine Komponente *functionResult* : EnumDeviceUnitDataSetCommand.

  Diese gibt Ihnen Auskunft über Fehler und den aktuellen Zustand.
- eine Komponente *handle* : UDINT.

  Dies gibt Ihnen die Möglichkeit, mittels der Funktion _getStateOfUnitDataSetCommand (siehe **Funktion _getStateOfUnitDataSetCommand**) den aktuellen Zustand einer Datensicherungsfunktion abzufragen (nützlich insbesondere bei Weiterschaltbedingung IMMEDIATELY).

Weitere Informationen zu den Datentypen StructRetUnitDataSetCommand und EnumDeviceUnitDataSetCommand sehen Sie im Abschnitt Rückgabewert der Funktion _saveUnitDataSet (siehe **Funktion _saveUnitDataSet**).

Für Informationen über Datensicherung, siehe Anwendung der Datensicherung und -initialisierung aus Anwenderprogramm (Seite 459).

Siehe auch

Funktion _exportUnitDataSet (Seite 422)

8.14.9 Funktion _deleteAllUnitDataSets

Alle Datensätze mit gesicherten Werten folgender Variablen werden gelöscht.

- gesicherte nicht remanente oder remanente Unit-Variablen des Interface- oder Implementationsabschnitts einer Unit (z. B. ST-Quelle, MCC-Quelle)
- gesicherte nicht remanente oder remanente geräteglobale Variablen.
- exportierte nicht remanente oder remanente Unit-Variablen des Interfaceabschnitts einer Unit (z. B. ST-Quelle, MCC-Quelle)

Bei Aufruf in Kurzform müssen alle Parameter (auch alle optionalen Parameter) angegeben werden.

Deklaration

```
_deleteAllUnitDataSets (
            unitName       : STRING,
            storageType    : EnumDeviceStorageType,
            { path         : STRING,
             nextCommand   : EnumNextCommandMode,
            }
            )              : StructRetUnitDataSetCommand
```

Eingangsparameter

unitName

Datentyp: STRING

Name der Unit (z. B. ST-Quelle, MCC-Quelle); der Name muss in Kleinbuchstaben und durch Hochkommata begrenzt angegeben werden (z. B. 'st_unit1').

Bei Angabe von '_device' werden alle Datensätze für geräteglobale Variablen gelöscht (möglich ab Version 3.2 des SIMOTION Kernels).

storageType

Datentyp: EnumDeviceStorageType

Ort, von dem der Datensatz gelöscht werden soll..

```
TYPE EnumDeviceStorageType : (
    TEMPORARY_STORAGE        // temporäre Datenablage
                             // (RAM-Disk),
                             // wird bei Netzausfall gelöscht
    , PERMANENT_STORAGE      // permanente Datenablage
                             // (MemoryCard), bleibt bei
                             // Netzausfall erhalten
    , USER_DEFINED )         // mit Pfadangabe
                             // (nur Voreinstellung zulässig)
END_TYPE
```

path (optional)

Datentyp: STRING

Voreinstellung: '' (leerer String)

Zielpfad, wenn storageType = USER_DEFINED.

Nur die Angabe des Voreinstellungswertes ist zulässig.

nextCommand (optional)

Datentyp: EnumNextCommandMode

Voreinstellung: IMMEDIATELY

Weiterschaltung zum nächsten Befehl

```
TYPE EnumNextCommandMode : (
    IMMEDIATELY                 // sofort
    , WHEN_COMMAND_DONE )       // nach Beenden oder Abbruch
                                // des Befehls
END_TYPE
```

Rückgabewert

Datentyp: StructRetUnitDataSetCommand

Der Rückgabewert ist eine Struktur vom Datentyp StructRetUnitDataSetCommand. In ihm sind zusammengefasst:

- eine Komponente *functionResult* : EnumDeviceUnitDataSetCommand.

  Diese gibt Ihnen Auskunft über Fehler und den aktuellen Zustand.
- eine Komponente *handle* : UDINT.

  Dies gibt Ihnen die Möglichkeit, mittels der Funktion _getStateOfUnitDataSetCommand (siehe **Funktion _getStateOfUnitDataSetCommand**) den aktuellen Zustand einer Datensicherungsfunktion abzufragen (nützlich insbesondere bei Weiterschaltbedingung IMMEDIATELY).

Weitere Informationen zu den Datentypen StructRetUnitDataSetCommand und EnumDeviceUnitDataSetCommand sehen Sie im Abschnitt Rückgabewert der Funktion _saveUnitDataSet (siehe **Funktion _saveUnitDataSet**).

Für Informationen über Datensicherung, siehe Anwendung der Datensicherung und -initialisierung aus Anwenderprogramm (Seite 459).

Siehe auch

Funktion _exportUnitDataSet (Seite 422)

8.15 Funktionen für CommandId

8.15.1 Funktion _getCommandId

Die Funktion liefert eine projektweit eindeutige CommandId, die zur eindeutigen Identifikation von Befehlen verwendet werden kann.

Es wird immer eine CommandId geliefert, es erfolgt keine Fehlerrückmeldung.

Deklaration

```
_getCommandId ( ) : CommandIdType
```

Eingangsparameter

keine

Rückgabewert

Datentyp: CommandIdType

Projektweit eindeutige CommandId zum Verfolgen des Befehlsstatus.

```
TYPE
 CommandIdType : STRUCT
    SystemId_low        : UDINT;    // niederwertiger Teil
    SystemId_high       : UDINT;    // höherwertiger Teil
 END_STRUCT
END_TYPE
```

Siehe auch

Den Parameter commandId richtig verwenden (Seite 559)

8.15.2 Funktion _getSyncCommandId

Die Funktion liefert dem Anwender eine projektweit eindeutige syncCommandId. Diese kann den Systemfunktionen BEGIN_SYNC und _startSyncCommands (siehe Listenhandbücher der SIMOTION Geräte) übergeben werden, um Bewegungsabläufe synchron zu starten.

Es wird immer eine syncCommandId geliefert, es erfolgt keine Fehlerrückmeldung.

Deklaration

```
_getSyncCommandId ( ) : CommandIdType
```

Eingangsparameter

keine

Rückgabewert

Datentyp: CommandIdType

Projektweit eindeutige syncCommandId zum Verfolgen des Befehlsstatus.

```
TYPE
 CommandIdType : STRUCT
    SystemId_low        : UDINT;    // niederwertiger Teil
    SystemId_high       : UDINT;    // höherwertiger Teil
 END_STRUCT
END_TYPE
```

Siehe auch

Den Parameter commandId richtig verwenden (Seite 559)

8.16 Wartezeit definieren

8.16.1 Funktion _waitTime

Die Funktion unterbricht die Task, die diese Funktion absetzt, bis die im Aufruf spezifizierte Zeit abgelaufen ist.

ACHTUNG
Die Funktion sollte nur in MotionTasks verwendet werden, die Verwendung in zyklischen Tasks kann zu Zeitüberwachungsfehlern führen! • Bei SynchronousTasks: Sie können konfigurieren, ob die Zeitüberwachung ausgesetzt wird. Standardmäßig ist die Zeitüberwachung aktiv. Beachten Sie zusätzlich bei der IPOsynchronousTask: Die UserInterruptTasks werden nicht mehr durch ihr auslösendes Ereignis gestartet! • Bei anderen zyklischen Tasks (BackgroundTask, TimerInterruptTasks): Die Zeitüberwachung ist immer aktiv. Verwenden Sie in zyklischen Tasks die Systemfunktionsbausteine Zeitgeber (siehe **Zeitgeber**), um Wartezeiten zu realisieren.

Die Funktion _waitTime ist immer ausführbar, der Rückgabewert = 0.

Für Informationen, wie Sie eine Task eine bestimmte Zeit warten lassen können, siehe Tasks eine definierte Zeitdauer warten lassen (Seite 329).

Deklaration

```
_waitTime (
        timeValue    : TIME        // Wartezeit
        ) : DINT                   // immer = 0
```

Eingangsparameter

timeValue

Datentyp: TIME

Angabe der Zeit, in der die Taskbearbeitung unterbrochen wird.

Rückgabewert

Datentyp: DINT

Ist immer 0..

Siehe auch

Zeitgeber (Seite 502)

Zeitaufteilung in der Round-Robin-Ablaufebene (Seite 267)

Wartezeiten in zyklischen Tasks (Seite 558)

8.17 Gerätespezifische Funktionen

8.17.1 Funktion _getDeviceId

Die Funktion liest die Hardware-Kennung des SIMOTION Geräts aus dessen Hardware-Informationsblock aus. Den Typ der auszulesenden Kennung geben Sie beim Aufruf der Funktion als Eingangsparameter an.

Deklaration

```
_getDeviceId (
              idType        : EnumDeviceIdType
             )              : StructRetGetDeviceId0
```

Eingangsparameter

idType

Datentyp: EnumDeviceIdType

Angabe der auszulesenden Kennung

```
TYPE EnumDeviceIdType : (
    SERIAL_NUMBER          // Seriennummer
    , HW_TYPE              // Type der Hardware
    , SPECIFIC_NUMBER      // spezielle OEM-Nummer
    , ORDER_ID )           // MLFB der Baugruppe
END_TYPE
```

Rückgabewert

Datentyp: StructRetGetDeviceId

Der Rückgabewert ist eine Struktur vom Datentyp StructRetGetDeviceId. In ihm sind zusammengefasst:

- eine Komponente *functionResult* : DINT.

  Diese gibt Ihnen Auskunft über Fehler.
- eine Komponente *id* : STRING[254].

  Diese enthält die ausgelesene Hardware-Kennung der MemoryCard.

```
TYPE StructRetGetDeviceId : STRUCT
    functionResult : DINT;        //Fehlerstatus
                                  // 0: Kein Fehler
                                  // <> 0: Fehler
    id : STRING[254];             // Ausgelesene Hardware-Kennung
    END_STRUCT;
END_TYPE
```

8.17.2 Funktion _getMemoryCardId

Die Funktion liest die Hardware-Kennung einer MemoryCard aus deren Hardware-Informationsblock aus. Den Typ der auszulesenden Kennung geben Sie beim Aufruf der Funktion als Eingangsparameter an (z. Zt. nur Seriennummer möglich).

Deklaration

```
_getMemoryCardId (
                  idType   : EnumMemoryCardIdType
                 )         : StructRetGetMemoryCardId0
```

Eingangsparameter

idType

Datentyp: EnumMemoryCardIdType

Angabe der auszulesenden Kennung

```
TYPE EnumMemoryCardIdType : (
    SERIAL_NUMBER )        //Seriennummer
END_TYPE
```

Rückgabewert

Datentyp: StructRetGetMemoryCardId

Der Rückgabewert ist eine Struktur vom Datentyp StructRetGetMemoryCardId. In ihm sind zusammengefasst:

- eine Komponente *functionResult* : DINT.

  Diese gibt Ihnen Auskunft über Fehler.
- eine Komponente *id* : STRING[254].

  Diese enthält die ausgelesene Hardware-Kennung der MemoryCard.

```
TYPE StructRetGetMemoryCardId : STRUCT
    functionResult : DINT;        //Fehlerstatus
                        // 0: Kein Fehler
                        // <> 0: Fehler
    id : STRING[254];            // Ausgelesene Hardware-Kennung
    END_STRUCT;
END_TYPE
```

8.17.3 Funktion _setDeviceErrorLED

Die Funktion setzt am SIMOTION Gerät den Fehler **Unterlizenzierung Technologie-/Optionsobjekte**. Die entsprechende LED am SIMOTION Gerät blinkt rot (Siehe Gerätehandbuch des SIMOTION Geräts).

Deklaration

```
_setDeviceErrorLED ( ) : DINT
```

Eingangsparameter

keine

Rückgabewert

Datentyp:	DINT
0	Kein Fehler
<> 0	Fehler

8.17.4 Funktion _setDriveObjectSTW

Beschreibung

Mit der Systemfunktion _setDriveObjectSTW können Sie ab V4.1.2 frei verwendbare bzw. nicht vom SIMOTION-RT unterstützte Bits im STW1_Device des Telegramms 39x zum DO1 hin setzen.

Deklaration

```
_setDriveObjectSTW
                   (DINT  logAddress,
                    UINT  STW1BitMask,
                    UINT  STW1BitSet) : DINT
```

Eingangsparameter

logAddr	Logische Basisadresse des Ausgangstelegramms zum Antriebsobjekt.
STW1BitMask	Bitmaske zur Selektierung der zu beeinflussenden Bits im Steuerwort des Ausgangstelegramms. Bits, die in der Kopplung zum Antriebsobjekt von SIMOTION autonom bedient werden (wie z.B. zur Synchronisation, Störungshandlung usw.), sind von dieser Funktion nicht erreichbar.
STW1BitSet	Bit-codierter Wert zum Schreiben in das Steuerwort des Ausgangstelegramms. Es werden nur die Bits geschrieben, die über STW1BitMask selektiert sind

Rückgabewert

Datentyp: DINT	Status der Funktion	Bedeutung des Returnwertes
hex		
0x0000 0000	Auftrag fehlerfrei beendet	Bits wurden in das Steuerwort geschrieben Bits waren im STW1 bereits im vorgegebenen Zustand
0xFFFF 8090	Auftrag abgebrochen	Logische Adresse ist nicht vorhanden
0xFFFF 8091		Adressiertes DO unterstützt die Funktion nicht
0xFFFF 8093		STW1BitMask enthält für das AWP gesperrte Bits
0xFFFF 809F		Interner Fehler, Funktion nicht ausführbar, z. B. Funktion in Version kleiner als 4.1.2 aufgerufen

Bitbelegung und Reservierung der STW-Bits in den 39x-Telegrammen:

Bit	Name	Bedeutung	Zugriffsrecht
12-15	MLZ	Dyn. Master Lebenszeichen	SIMOTION-FW
11	-	Reserviert - System	Reserviert
10	DAG	Demand from AG	SIMOTION-FW
9	-	Reserviert - System	Reserviert
8	-	Reserviert - System	Reserviert
7	FACK	Fault acknowledge	_resetDriveObjectFault
6	-		_setDriveObjectSTW
5	-		_setDriveObjectSTW
4	-		_setDriveObjectSTW
3	-		_setDriveObjectSTW
2	-		_setDriveObjectSTW
1	PING	Startimpuls für Uhrzeitsynchronisation	_setDriveObjectSTW
0	SYN	Dyn. Synchronisationsflag für Systemzeitzyklus	SIMOTION-FW

8.18 Speichergröße einer Variable bzw. eines Datentyps bestimmen

8.18.1 Funktion _sizeOf

Die Funktion liefert die für eine Variable oder einen Datentyp benötigte Speichergröße in Byte zurück.

- Standardmäßig wird der Rückgabewert zum Übersetzungszeitpunkt ermittelt. Die Funktion kann dann in Konstantenausdrücken (z. B. bei der Initialisierung) verwendet werden.
- Ausnahme - Nur ab Version 4.2 des SIMOTION Kernels:

  Bei einem ARRAY mit dynamischer Länge (d. h. die Indexgrenzen sind nicht deklariert - nur als Durchgangsparameter in Funktionen oder Funktionsbausteinen möglich) wird der Rückgabewert zur Laufzeit ermittelt.

Deklaration

```
_sizeOf (
         in  : ANY  // Bezeichner des Datentyps oder
                    // der Variablen
         )   : DINT
```

Eingangsparameter

in

Datentyp: ANY

Bezeichner der Variablen oder des Datentyps, dessen Größe zu ermitteln ist.

Rückgabewert

Datentyp: DINT

Benötigte Speichergröße in Byte.

Die Speichergröße wird unter Berücksichtigung der natürlichen Ausrichtung angegeben, d. h. gemäß der Belegungsmöglichkeiten der Datentypen im Speicher. Es wird somit die effektive Größe ermittelt, die bei einer Verwendung des Datentyps in einem ARRAY benötigt wird.

Die tatsächlich benötigte Größe kann geringer sein.

Beispiel

```
TYPE
    a_type  : STRUCT
        a   : LREAL;    // 8 Byte
        b   : BOOL;     // 1 Byte
    END_STRUCT;
END_TYPE
//...
x := _sizeOf (in := a_type);    // liefert Wert 16
```

8.18.2 Funktion _firstIndexOf

Die Funktion liefert für ein ARRAY (Variable oder Datentyp) die untere Indexgrenze zurück.

- Bei einem ARRAY mit definierter Länge (d. h. die Indexgrenzen sind deklariert), wird der Rückgabewert zum Übersetzungszeitpunkt ermittelt. Die Funktion kann dann in Konstantenausdrücken (z. B. bei der Initialisierung) verwendet werden.
- Nur ab Version 4.2 des SIMOTION Kernels:

  Bei einem ARRAY mit dynamischer Länge (d. h. die Indexgrenzen sind nicht deklariert - nur als Durchgangsparameter in Funktionen oder Funktionsbausteinen möglich) wird der Rückgabewert zur Laufzeit ermittelt.

Deklaration

```
_firstIndexOf (
         in  : ARRAY OF ANY  // Bezeichner des ARRAY
                             // (Datentyp oder Variable)
         )   : DINT
```

Eingangsparameter

in

Datentyp: ARRAY OF ANY

Bezeichner des ARRAY (Variable oder Datentyp), dessen untere Indexgrenze zu ermitteln ist.

Rückgabewert

Datentyp: DINT

Untere Indexgrenze.

Beispiel

```
TYPE
    array_1 : ARRAY [5..29] OF DINT;
END_TYPE
// ...
x := _firstIndexOf (in := array_1);      // liefert Wert 5
```

8.18.3 Funktion _lastIndexOf

Die Funktion liefert für ein ARRAY (Variable oder Datentyp) die obere Indexgrenze zurück.

- Bei einem ARRAY mit definierter Länge (d. h. die Indexgrenzen sind deklariert), wird der Rückgabewert zum Übersetzungszeitpunkt ermittelt. Die Funktion kann dann in Konstantenausdrücken (z. B. bei der Initialisierung) verwendet werden.
- Nur ab Version 4.2 des SIMOTION Kernels:

  Bei einem ARRAY mit dynamischer Länge (d. h. die Indexgrenzen sind nicht deklariert - nur als Durchgangsparameter in Funktionen oder Funktionsbausteinen möglich) wird der Rückgabewert zur Laufzeit ermittelt.

Deklaration

```
_lastIndexOf (
         in  : ARRAY OF ANY  // Bezeichner des ARRAY
                             // (Datentyp oder Variable)
         )   : DINT
```

Eingangsparameter

in

Datentyp: ARRAY OF ANY

Bezeichner des ARRAY (Variable oder Datentyp), dessen obere Indexgrenze zu ermitteln ist.

Rückgabewert

Datentyp: DINT

Untere Indexgrenze.

Beispiel

```
TYPE
    array_1 : ARRAY [5..29] OF DINT;
END_TYPE
// ...
x := _lastIndexOf (in := array_1);      // liefert Wert 29
```

8.18.4 Funktion _lengthIndexOf

Die Funktion liefert für ein ARRAY (Variable oder Datentyp) die Anzahl der Feldelemente zurück.

- Bei einem ARRAY mit definierter Länge (d. h. die Indexgrenzen sind deklariert), wird der Rückgabewert zum Übersetzungszeitpunkt ermittelt. Die Funktion kann dann in Konstantenausdrücken (z. B. bei der Initialisierung) verwendet werden.
- Nur ab Version 4.2 des SIMOTION Kernels:

  Bei einem ARRAY mit dynamischer Länge (d. h. die Indexgrenzen sind nicht deklariert - nur als Durchgangsparameter in Funktionen oder Funktionsbausteinen möglich) wird der Rückgabewert zur Laufzeit ermittelt.

Deklaration

```
_lengthIndexOf (
         in  : ARRAY OF ANY  // Bezeichner des ARRAY
                             // (Datentyp oder Variable)
         )   : DINT
```

Eingangsparameter

in

Datentyp: ARRAY OF ANY

Bezeichner des ARRAY (Variablen oder Datentyp), bei dem die Anzahl der Feldelemente zu ermitteln ist.

Rückgabewert

Datentyp: DINT

Anzahl der Feldelemente.

Es gilt:
_lengthIndexOf (*array-name*) := _lastIndexOf (*array-name*) - _firstIndexOf (*array-name*) +1.

Beispiel

```
TYPE
    array_1 : ARRAY [5..29] OF DINT;
END_TYPE
// ...
x := _lengthIndexOf (in := array_1);      // liefert Wert 25
```

8.19 Weitere verfügbare Systemfunktionen

In SIMOTION sind weitere Systemfunktionen verfügbar, die z. B. durch die SIMOTION Geräte und Technologieobjekte eingebracht werden. Folgende Tabelle gibt einen Überblick, wo diese beschrieben sind.

Tabelle 8- 16 Überblick über weitere Systemfunktionen und -funktionsbausteine in SIMOTION ST

Systemfunktion	Beschreibung
Systemfunktionen der Technologieobjekte	Listenhandbuch SIMOTION Technologiepaket CAM Systemfunktionen (Referenzliste) Listenhandbuch SIMOTION Technologiepaket TControl (Referenzliste) Siehe ergänzend Funktionshandbücher zu den Technologieobjekten
Systemfunktionen der SIMOTION Geräte	Listenhandbuch der SIMOTION Geräte (Referenzliste)
Systemfunktionen zum Ansteuern von Achsen nach PLCopen-Norm	Listenhandbuch SIMOTION Technologiepaket CAM Systemfunktionen (Referenzliste)
Standardfunktionen zum Ansteuern von Peripheriebaugruppen und Antriebskomponenten	Listenhandbuch zur entsprechenden Peripheriebaugruppe und Antriebskomponente (Referenzliste)

8.20 Anwendung einiger Systemfunktionen

8.20.1 Meldungen programmieren

8.20.1.1 Allgemeines

Sie können Meldungen, z. B. auch Fehlermeldungen, mit folgenden Funktionen programmieren bzw. deren Status abfragen:

- _alarmSId (Generieren einer Meldung ohne Quittierung)
- _alarmSqId (Generieren einer Meldung mit Quittung)
- _alarmScId (Abfrage nach Meldungszustand)

Voraussetzung ist jedoch ein anwendungsspezifisch projektierter Meldungsname

Hinweis

Für benutzerdefinierte Einträge in den Diagnosepuffer steht die Funktion *_writeAndSendMessage* zur Verfügung. Die Beschreibung der Funktion finden Sie in den Listenhandbüchern der SIMOTION Geräte.

Siehe auch Allgemeines zur Meldungsprogrammierung (Seite 350).

8.20.1.2 Übersicht der Funktionen

Die beschriebenen Funktionen können in Bibliotheken verwendet werden.

Mit **_alarmSId** generieren Sie bei jedem Aufruf eine **nicht quittierungspflichtige Meldung**, die in Abhängigkeit von einem Signal ausgelöst wird und an die ein Begleitwert angehängt werden kann. Die Meldung wird an alle dafür angemeldeten Anzeigegeräte verschickt.

Mit **_alarmSqId** generieren Sie bei jedem Aufruf eine **quittierungspflichtige Meldung**, die in Abhängigkeit von einem Signal ausgelöst wird und an die ein Begleitwert angehängt werden kann. Die Meldung wird an alle dafür angemeldeten Anzeigegeräte verschickt und kann an diesen Geräten quittiert werden.

Als Eingabeparameter für die Funktionen verwenden Sie:

1. Das meldungsauslösende Signal:

   Dieses wird folgendermaßen interpretiert:

   – Wenn das Signal – bezogen auf den letzten Aufruf mit diesem Meldungsnamen – eine positive Flanke darstellt, wird eine kommende Meldung generiert. Eine kommende Meldung wird auch generiert, wenn das Signal beim erstmaligen Aufruf mit diesem Meldungsnamen den Zustand TRUE hat.

   – Wenn das Signal – bezogen auf den letzten Aufruf mit diesem Meldungsnamen – eine negative Flanke darstellt, wird eine gehende Meldung generiert.

2. Die zu generierende Meldung:

   Sie wird über eine projektweit eindeutige AlarmId vorgegeben.

   Zur AlarmId siehe den folgenden Abschnitt.

3. Optional einen Begleitwert, sofern in der Meldungsprojektierung ein Begleitwert angegeben wurde.

Die Funktion **_alarmScId** fragt den Zustand einer Meldung und deren Quittierungszustand ab. Hier sind folgende Fälle zu unterscheiden: Die Meldung wird über eine eindeutige AlarmId vorgegeben.

Zum formalen Aufbau der Funktionen siehe **_alarmSId** und **_alarmSId**.

Zum anwendungsspezifisch projektierten Meldungsnamen siehe Online-Hilfe.

Hinweis

Eine gehende Meldung sollten Sie nur nach einer kommenden generieren, ansonsten erfolgt eine Fehlermeldung.

Der Aufruf der Befehle, bei denen der projektierte Meldungsname übergeben wird, darf nur in Kurzform geschehen, d. h. mit vollständiger Auflistung aller Parameterwerte, jedoch ohne Angabe der Formalparameter.

Der Begleitwert (optional) muss eine Variable von einem elementaren Datentyp sein. Vermeiden Sie Konstantenwerte im Funktionsaufruf!

8.20.1.3 AlarmID

Bei den Funktionen _alarmSId, _alarmSqId und _alarmScId wird die zu generierende Meldung über die AlarmId vorgegeben. Zu einem projektierten Meldungsnamen erhalten Sie die AlarmId auf folgende Weise:.

- Als Variable _alarm.name, wobei *name* der Bezeichner der Meldung ist, wie in SIMOTION SCOUT projektiert.
- Mit der Funktion _getAlarmId(*name*).

Siehe auch

Funktion _alarmSId (Seite 351)

Funktion _getAlarmId (Seite 357)

8.20.1.4 Pufferverwaltung von AlarmS

Beschreibung

Es steht für die AlarmS-Meldungen eine Meldeliste mit 40 Pufferplätzen zur Verfügung. In diese Meldeliste werden die AlarmS-Meldungen mit ihrer ID eingetragen. Zu jedem gehenden AlarmS muss ein kommender AlarmS derselben ID in der Meldeliste existieren. Für jeden der insgesamt 40 Listeneinträge gibt es zusätzlich einen Sendepuffer. Dieser Sendepuffer wird genutzt, um die Benachrichtigung der angemeldeten Clients (HMI oder SIMOTION SCOUT) zu organisieren.

Meldeliste und Sendepuffer

Meldeliste und Sendepuffer werden wie folgt benutzt:

Rückgabewert		Bedeutung
Rückgabewerte bei kommendem AlarmS (Funktionsaufruf mit steigender Flanke)		
	16#8002	Es sind bereits alle Einträge in der Meldeliste belegt, Eintrag des AlarmS in die Meldeliste ist nicht erfolgt.
	16#8003	Es steht noch ein gehender AlarmS für diese ID in der Meldeliste und der Sendepuffer ist noch belegt, Eintrag des AlarmS in die Meldeliste ist nicht erfolgt
	16#8004	In der Meldeliste steht bereits ein kommender AlarmS für diese ID, Eintrag des AlarmS in die Meldeliste ist nicht erfolgt.

Rückgabewert		Bedeutung
	16#0000	ID des AlarmS wird in die Meldeliste eingetragen. Der zugehörige Sendepuffer wird belegt. Die Systemfunktion kehrt mit dem Rückgabewert 0000 zurück. Die angemeldeten Clients werden benachrichtigt. Nach erfolgreicher Benachrichtigung der Clients wird der Sendepuffer wieder frei gegeben. Die ID des AlarmS bleibt als kommender AlarmS in der Meldeliste.
Rückgabewerte bei gehendem AlarmS (Funktionsaufruf mit fallender Flanke)		
	16#8003	Es steht noch ein kommender AlarmS für diese ID in der Meldeliste und der Sendepuffer ist noch belegt, Eintrag des AlarmS in die Meldeliste ist nicht erfolgt.
	16#8004	In der Meldeliste steht bereits ein gehender AlarmS für diese ID, Eintrag des AlarmS in die Meldeliste ist nicht erfolgt.
	16#8007	Es wurde zu der ID kein kommender Eintrag gefunden, Eintrag des AlarmS in die Meldeliste ist nicht erfolgt.
	16#0000	Wird ein gekommener AlarmS zur ID in der Meldeliste gefunden und dessen Sendepuffer ist nicht mehr belegt ist, so wird dieser Eintrag mit dem gehenden AlarmS überschrieben. Der zugehörige Sendepuffer wird belegt. Die Systemfunktion kehrt mit dem Rückgabewert 0000 zurück. Die angemeldeten Clients werden benachrichtigt. Nach erfolgreicher Benachrichtigung der Clients wird der Sendepuffer wieder frei gegeben. Der Eintrag in der Meldeliste wird gelöscht.

8.20.1.5 Beispiel für die Meldungsgenerierung

Das Beispiel in der Tabelle prüft die Temperatur und generiert eine nicht quittierungspflichtige kommende Meldung (etwa *Temperatur zu hoch, kommend*), falls die Temperatur zu hoch ist. Falls die Temperatur unter dem festgelegten Maximalwert zurückgeht, wird eine gehende Meldung generiert (kommende Meldung verschwindet).

Die Meldung mit dem Namen *SCOUT_alarm_name* wurde in SIMOTION SCOUT projektiert und lautet z. B.: *Temperatur zu hoch: @1l%2d@* Grad. Eine Statusvariable verhindert das Wiederholen der gleichen Meldung. Das Programm *handleAlarm* wird der BackgroundTask zugeordnet.

Tabelle 8- 17 Beispiel für die Meldungsgenerierung

```
INTERFACE
 PROGRAM handleAlarm;
END_INTERFACE

IMPLEMENTATION
 PROGRAM handleAlarm
    VAR
        retVal         : DWORD;        // Rückgabewert
        temperature     : INT;         // abzufragender Zustand
        maxTemperature: INT := 60;     // Vergleichswert für Zustand
        mySignal     : BOOL := FALSE;  // melden ja/nein
    END_VAR
    //...
    IF temperature > maxTemperature THEN
        IF mySignal = FALSE THEN
                                       // kommende Meldung, nicht quittierungspflichtig
            retVal := _alarmSId (
            Sig     := TRUE,
            Ev_id := _alarm.SCOUT_alarm_name,
            Sd     := temperature);
            mySignal := TRUE;
        END_IF;
    ELSE
        IF mySignal = TRUE THEN
                                       // gehende Meldung, nicht quittierungspflichtig
            retVal := _alarmSId (
            Sig := FALSE,
            Ev_id := _alarm.SCOUT_alarm_name,
            Sd := temperature);
            mySignal := FALSE;
        END_IF;
    END_IF;
    //...
 END_PROGRAM
```

8.20.1.6 Fehlernummer und Status einer Meldung abfragen (Rückgabewerte filtern)

Der Rückgabewert der Funktionen *_alarmSId* und *_alarmSqId* enthält die Fehlernummer und gibt somit Auskunft, ob bei der Ausführung ein Fehler aufgetreten ist. Wie bei den meisten Systemfunktionen zeigt ein Rückgabewert = 0 eine fehlerfreie Aufführung an.

Der Rückgabewert der Funktion *_alarmScId* zeigt jedoch sowohl die Fehlernummer als auch den Zustand einer Meldung an. Deshalb müssen Sie bei einer Abfrage des Status mit diesen Funktionen zuerst den Rückgabewert mit der Konstanten ALARMS_ERROR (= 16#8000) filtern. Damit stellen Sie fest, ob ein Fehler bei der Ausführung der Funktion aufgetreten ist. Der Filter und die Fehlernummern sind so gewählt, dass sie bei einer UND-Verknüpfung wahr sind. Wenn kein Fehler aufgetreten ist, können Sie den Status der Meldung auswerten.

Die vollständige Auflistung der Fehlernummern und Meldungszustände finden Sie unter **Funktionen zur Meldungsprogrammierung**.

Demzufolge können Sie die Abfrage nach einem Fehler beim Absetzen des Befehls *_alarmScId* wie folgt gestalten (die Variable *retVal* vom Datentyp DWORD beinhaltet den Rückgabewert der Funktion):

Tabelle 8- 18 Beispiele für Fehlerabfrage

```
retVal := _alarmScId (Ev_id := _alarm.SCOUT_alarm_name);
// Hier Fehlerabfrage
    IF (retVal AND ALARMS_ERROR) <> 0 THEN
    // Bedingung erfüllt, also ist ein Fehler aufgetreten.
        IF retVal = (ALARMS_ERROR OR
            DSC_SVS_DEVICE_ALARMS_ILLEGAL_EVENT_ID) THEN
            ; // Meldungsnummer nicht zulässig.
        END_IF;
    ELSE
        // Bedingung nicht erfüllt, also kein Fehler.
        // Abfrage des Meldungs- und Quittierungszustands
        IF retVal = 16#0000 THEN
            ; // Meldung gegangen, nicht quittiert.
            ELSIF retVal = ALARMS_STATE THEN
            ; // Meldung gekommen, nicht quittiert.
            ELSIF retVal = 16#0010 THEN
            ; // Meldung nicht vorhanden.
            ELSIF retVal = (ALARMS_QSTATE OR ALARMS_STATE) THEN
            ; // Meldung gekommen, quittiert.
        END_IF;
 //...
    END_IF;
```

Hinweis

Für die Fehlerabfrage können Sie Konstantenwerte und symbolische Konstanten gleichberechtigt verwenden, siehe **Funktionen zur Meldungsprogrammierung**.

8.20.2 Konsistentes Schreiben und Lesen von Variablen (Semaphoren)

8.20.2.1 Konsistenter Datenzugriff

Alle Zugriffe auf Variablen elementarer Datentypen (siehe Kapitel *Elementare Datentypen*) werden vom System konsistent gehandhabt. Es wird gewährleistet, dass diese Variablen nicht zwischendurch geändert werden, während Sie sie bearbeiten.

Bei Zugriffen auf **globale** Variablen abgeleiteter Datentypen (siehe Kapitel *Anwenderdefinierte Datentypen (UDT))* muss der Anwender selbst für die Konsistenz der Daten sorgen, falls mehrere Tasks auf dieselben Variablen zugreifen (symbolische I/O-Variablen, Systemvariablen der SIMOTION Geräte, Systemvariablen der Technologieobjekte, geräteglobale Variablen und Unit-Variablen, siehe Kapitel *Variablenmodell*.

Zugriffe auf **lokale** Variablen abgeleiteter Datentypen sind immer konsistent, da sie nur innerhalb des Programms (bzw. Funktion oder Funktionsbausteinen) verwendet werden können, in dem sie definiert sind.

Hinweis

Innerhalb einer Task ist konsistenter Datenzugriff immer gewährleistet.

8.20.2.2 Semaphoren

Um das konsistente Schreiben und Lesen von globalen Variablen sicherzustellen, arbeiten Sie mit Semaphoren.

Als Semaphore dient eine globale Variable (z. B. *semaA*) vom Datentyp DINT. Falls sie Element eines Arrays ist, muss der Index bereits beim Compilieren festgelegt werden (z. B. a[2]).

Mit folgenden Funktionen ändern und prüfen Sie den Status des Semaphores:

- _testAndSetSemaphore (sema : DINT) : BOOL

  Mit dieser Funktion überprüfen Sie, ob das Semaphore gesetzt ist:

  – Rückgabewert TRUE: Das Semaphore ist freigegeben.

  – Rückgabewert FALSE: Das Semaphore ist gesetzt.

  Nach Beenden der Funktion ist das Semaphore immer gesetzt. Weitere Aufrufe der Funktion (auch aus anderen Programmen) ergeben den Rückgabewert FALSE, solange bis die Funktion _releaseSemaphore (semaA) aufgerufen wird.

- _releaseSemaphore (sema: DINT) : VOID

Zum formalen Aufbau der Funktionen siehe Kapitel *Variablenhandling*.

Das Semaphore wird frei gegeben.

Unter folgenden Bedingungen ist nun konsistenter Datenzugriff auf globale Variablen gewährleistet:

1. Alle Tasks signalisieren den Zugriff auf globale Variablen durch Setzen eines Semaphores.
2. Alle Tasks greifen auf globale Variablen nur bei frei gegebenem Semaphore zu.

8.20.2.3 Beispiel: Konsistenter Datenzugriff mit Semaphoren

Das Beispiel in der Tabelle verdeutlicht den Gebrauch von Semaphoren in je einem Programm, das Daten schreibt bzw. liest.

Tabelle 8- 19 Beispiel für die Sicherstellung eines konsistenten Zugriffs auf globale Variablen mit Hilfe von Semaphoren

```
IMPLEMENTATION
    VAR_GLOBAL
        myArray : ARRAY [0..1] OF DINT;
        semaA : DINT;
    END_VAR

    PROGRAM Writer
        // konsistentes Schreiben von Variablen
        IF _testAndSetSemaphore(sema := semaA) THEN
            myArray[0] := 18;
            myArray[1] := 19;
            _releaseSemaphore(sema := semaA);
            // Das Semaphore muss im TRUE-Zweig der Abfrage
            // frei gegeben werden; hier ist sicher gestellt,
            // dass es nur dann frei gegeben wird, wenn es neu
            // gesetzt wurde.
        ELSE
            ; // Fehlerbehandlung
        END_IF;
        // _releaseSemaphore(sema := semaA);
        // An dieser Stelle wäre die Freigabe falsch;
        // das Semaphore würde immer frei gegeben werden.
    END_PROGRAM
    PROGRAM Reader
        VAR
            var0        : DINT;
            var1        : DINT;
        END_VAR

        // konsistentes Lesen von Variablen
        IF _testAndSetSemaphore(sema := semaA) THEN
            var0 := myArray[0];
            var1 := myArray[1];
            _releaseSemaphore(sema := semaA);
            // Das Semaphore muss im TRUE-Zweig der Abfrage
```

```
            // frei gegeben werden; hier ist sicher gestellt,
            // dass es nur dann frei gegeben wird, wenn es neu
            // gesetzt wurde.
        ELSE
            ; // Fehlerbehandlung
        END_IF;
        // _releaseSemaphore(sema := semaA);
        // An dieser Stelle wäre die Freigabe falsch;
        // das Semaphore würde immer frei gegeben werden.
    END_PROGRAM

END_IMPLEMENTATION
```

8.20.3 Datensicherung und -initialisierung aus Anwenderprogramm

8.20.3.1 Datensicherung und -initialisierung aus Anwenderprogramm - Funktionen und Hinweise

Aus einem Anwenderprogramm heraus können die Werte folgender Variablen in Datensätzen gesichert, geladen oder initialisiert werden:

- Nicht remanente oder remanente Unit-Variablen des Interface- oder Implementationsabschnitts einer Unit (z. B. ST-Quelle, MCC-Quelle)
- Nicht remanente oder remanente geräteglobale Variablen.

Hierzu stehen verschiedene Funktionen zur Verfügung, siehe Datensicherung aus Anwenderprogramm (Seite 414).

- *_saveUnitDataSet*: Die Werte werden binär als Datensatz gespeichert (. Funktion _saveUnitDataSet (Seite 414))
- *_loadUnitDataSet*: Die Werte werden aus einem mit *_saveUnitDataSet* binär gespeicherten Datensatz geladen (). Funktion _loadUnitDataSet (Seite 418)

  Das Laden des Datensatzes ist nur möglich, wenn seit dem Speichern des Datensatzes die Versionskennungen aller zu ladenden Datensegmente unverändert geblieben sind.

  Beachten Sie auch den Hinweis weiter unten.
- *_exportUnitDataSet* : Die Werte werden im XML-Format als ZIP-Archiv (Dateiname *.dat) exportiert (Funktion _exportUnitDataSet (Seite 422)).

- *_importUnitDataSet* : Die Werte werden aus einem Datensatz importiert, der mit *_exportUnitDataSet* im XML-Format als ZIP-Archiv (Dateiname *.dat) exportiert wurde (Funktion _importUnitDataSet (Seite 425))..

  Der Import des Datensatzes ist auch möglich, wenn sich seit dem Speichern des Datensatzes die Versionskennung eines zu ladenden Datensegments geändert hat:

  – Nicht mehr vorhandene Variablen werden ignoriert.

  – Bei hinzugekommenen Variablen bleibt der Wert unverändert.
  Sie können z. B. vor dem Import eines Datensatzes mit *_importUnitDataSet* die betreffenden Datensegmente initialisieren, um unerwünschte Werte in Variablen zu vermeiden.

  – Bei Variablen mit geändertem Datentyp wird der Wert übernommen, wenn er in den neuen Datentyp konvertiert werden kann; andernfalls wird der Wert der Variablen beibehalten.

- *_deleteUnitDataSet*: Ein einzelner Datensatz wird gelöscht (Funktion _deleteUnitDataSet (Seite 429)).
- *_checkExistingUnitDataSet*: Es wird überprüft, ob der angegebene Datensatz auf dem Speichermedium vorhanden ist (Funktion _checkExistingUnitDataSet (Seite 433)).
- *_deleteAllUnitDataSets*: Alle Datensätze werden gelöscht (Funktion _deleteAllUnitDataSets (Seite 435)).
- *_resetUnitData* : Die Werte der Variablen werden initialisiert, siehe Listenhandbuch *Systemfunktionen der Geräte*.

Zu Datensegmenten und deren Versionskennung siehe Kapitel *Zeitpunkt der Variableninitialisierung*, Tabelle *Versionskennung globaler Variablen und deren Initialisierung beim Download*, die Sie in den verschiedenen Programmierhandbüchern finden.

Die Beschreibung zu *_resetUnitData* finden Sie im Listenhandbuch (Referenzliste) zu den Systemfunktionen der SIMOTION Geräte).

Im Folgenden sind die Parameter näher erläutert.

ACHTUNG
Wenn bei einer Unit oder bei den geräteglobalen Variablen die Datenstruktur eines Datensegments verändert wird, so gilt beim Laden des Projekts in das SIMOTION Gerät: • Alle temporär gesicherten Datensätze werden gelöscht. • Alle binär (mit *_saveUnitDataSet*) gespeicherten Datensätze sind für dieses Datensegment nicht mehr lesbar.

Bis zu 16 Datensicherungsfunktionen können gleichzeitig auf einem Gerät laufen.

Die Anzahl der Datensätze, die gespeichert werden können, ist abhängig vom verfügbaren Speicherplatz, maximal 1_000_000.

Achten Sie auf die Konsistenz der zu sichernden oder zu exportierenden Daten (siehe Kapitel *Konsistentes Schreiben und Lesen von Variablen (Semaphoren)*).

Hinweis

Die mit *_saveUnitDataSet* oder *_exportUnitDataSet* gespeicherten Datensätze können Sie aus dem SIMOTION Gerät hochladen (Upload). Verwenden Sie hierzu in SIMOTION SCOUT die Funktion **Variablen sichern**. Die mit *_saveUnitDataSet* gespeicherten Datensätze werden dabei automatisch in das XML-Format konvertiert.

Mit der Umkehrfunktion **Variablen wiederherstellen** können Sie diese Datensätze und Variablen wieder in das SIMOTION Gerät laden (Download). Wählen Sie hierzu im Projektnavigator das entsprechende SIMOTION Gerät aus und wählen Sie im Kontextmenü die Funktion. Weitere Informationen siehe Projektierungshandbuch SIMOTION SCOUT.

Damit ist es z. B. möglich, die mit _saveUnitDataSet gespeicherten Daten zu erhalten, obwohl Sie durch einen Download des Projekts initialisiert oder unbrauchbar werden (z. B. bei Versionswechsel von SIMOTION).

8.20.3.2 Eingangsparameter

Die wichtigsten Parameter sind im Folgenden kurz dargestellt. Eine ausführliche Beschreibung der einzelnen Funktionen und ihrer Parameter finden Sie unter Datensicherung aus Anwenderprogramm (Seite 414).

- *unitName*: Name der Unit (z. B. ST-Quelle, MCC-Quelle)

  Bei Angabe von '*_device*' wird die Datensicherungsfunktion auf geräteglobale Variablen angewendet (möglich bei _saveUnitDataSet).

- *id*: Datensatznummer

  Durch Angabe einer Datensatznummer als Index können mehrere Versionen der Unit-Variablen gesichert werden und gezielt wieder geladen werden.

  id < 1_000_000

- *storageType*: Die Angabe des Speicherorts erlaubt die Auswahl:
  - TEMPORARY_STORAGE : Daten werden auf der RAM-Disk gespeichert (sind nach Netzausfall gelöscht)
  - PERMANENT_STORAGE: Daten werden auf MemoryCard gespeichert (bleiben nach Netzausfall erhalten)

- *nextCommand*: Weiterschaltbedingung

  Durch Angabe der Weiterschaltbedingung können Sie wählen:
  - Sofortige Bearbeitung des nächsten Befehls in der ST-Quelle (IMMEDIATELY)
  - Warten, bis Befehl beendet ist (WHEN_COMMAND_DONE)

    Zur Weiterschaltbedingung beachten Sie bitte die weiteren Ausführungen in diesem Kapitel.

- *dataScope*

  Er ermöglicht die Auswahl, auf welchen Abschnitt der Unit die Datensicherungsfunktion angewendet wird.

  – _INTERFACE: Funktion wird auf den Interfaceabschnitt einer Unit angewendet

  – _IMPLEMENTATION: Funktion wird auf den Implementationsabschnitt einer Unit angewendet

  – _INTERFACE_AND_IMPLEMENTATION: Funktion wird auf den Interface- und Implementationsabschnitt einer Unit angewendet.

    Bei Angabe von *unitName* = '*_device*' sind für *dataScope* nur die Werte _INTERFACE oder _INTERFACE_AND_IMPLEMENTATION zulässig.

- *KindOfData*:

  Er ermöglicht die Auswahl, ob die Datensicherungsfunktion auf remanente oder remanente globale Variablen angewendet wird.

  – NO_RETAIN_GLOBAL: Funktion wird auf nicht remanente globale Variablen angewendet

  – _RETAIN: Funktion wird auf remanente globale Variablen angewendet

  – ALL_GLOBAL: Funktion wird auf remanente und nicht remanente globale Variablen angewendet

  Wenn in einem Datensatz remanente und nicht remanente Variablen gespeichert sind, ist es möglich, die remanenten oder nicht remanenten Variablen selektiv zu laden bzw. zu importieren.

8.20.3.3 Rückgabewert

Der Rückgabewert ist eine Struktur vom Datentyp *StructRetUnitDataSetCommand*. In ihm sind zusammengefasst:

- eine Komponente *functionResult* : *EnumDeviceUnitDataSetCommand*.

  Diese gibt Ihnen Auskunft über Fehler und den aktuellen Zustand.

- eine Komponente *handle* : UDINT.

  Dies gibt Ihnen die Möglichkeit, mit Hilfe der Funktion *_getStateOfUnitDataSetCommand* den aktuellen Zustand der Datensicherungsfunktion abzufragen (notwendig bei Weiterschaltbedingung IMMEDIATELY).

8.20.3.4 Speicherort und Speicherbedarf

Den Speicherort legen Sie mit dem Eingangsparameter *storageType* fest:

- TEMPORARY_STORAGE: Datensätze werden auf der RAM-Disk gespeichert.
- PERMANENT_STORAGE: Datensätze werden auf der MemoryCard (unter \USER\SIMOTION\USER_DIR\) gespeichert.

Die Anzahl der Datensätze, die gespeichert werden können, ist abhängig vom freien Speicherplatz am jeweiligen Speicherort. Information über den freien Speicherplatz erhalten Sie mittels der Gerätediagnose, Register **Systemauslastung** (siehe Online-Hilfe).

ACHTUNG
Belegen Sie nicht den gesamten freien Speicherplatz mit Datensätzen! Bei vergrößerten Projektdaten ist sonst das Laden des Projekts ins Zielsystem oder das Kopieren RAM nach ROM nicht mehr möglich!

Den benötigten Speicherplatz für binär (mit *_saveUnitDataSet*) gespeicherte Datensätze können Sie mit folgenden Informationen abschätzen:

- Elementare Datentypen belegen ihre natürliche Datenbreite auf dem Speicher (siehe Tabelle *Bitbreiten und Wertebereiche der elementaren Datentypen* im Kapitel *Elementare Datentypen*, Datentyp BOOL belegt 1 Byte).
- Zusätzlicher Speicherbedarf entsteht durch:
  - Anpassung der Speicheradressen an Wort- oder Doppelwortgrenzen
  - Konsistenzinformationen (pro Datensatz ca. 100 Byte)
  - übliche Zusatzdaten eines Filesystems (z. B. Sektorköpfe, Directory, Belegen nur ganzer Sektoren möglich).

Für im XML-Format (mit *_exportUnitDataSet*) exportierte Datensätze ist der Speicherbedarf wesentlich höher und nicht auf diese Weise zu ermitteln.

8.20.3.5 Weiterschaltbedingung

Die Weiterschaltbedingung wird im Eingangsparameter *nextCommand* angegeben. Wenn dieser auf WHEN_COMMAND_DONE gesetzt wird, wird der nächste Befehl der ST-Quelle erst ausgeführt, wenn die Funktion beendet ist (synchrone Ausführung).

Der Rückgabewert enthält in der Komponente *functionResult* das Ergebnis der ausgeführten Funktion (siehe Beispiel).

Tabelle 8- 20 Funktionsaufruf einer Datensicherungsfunktion mit Weiterschaltbedingung WHEN_COMMAND_DONE

```
VAR_GLOBAL
    ds_ret : StructRetUnitDataSetCommand;
    error : BOOL := FALSE;
END_VAR

PROGRAM save_data_seq
    // Programm ist einer sequentiellen Task zugeordnet.
    // Funktion synchron ausführen:
    ds_ret := _loadUnitDataSet (
    unitName := 'ds3',
    id := 1,
```

```
    storageType := TEMPORARY_STORAGE,
    nextCommand := WHEN_COMMAND_DONE);
    // Funktion ist beendet, Ergebnis auswerten
    IF (ds_ret.functionResult <> DONE) THEN
        error := TRUE;    // Fehler
    END_IF;
END_PROGRAM
```

Dieses Vorgehen wird vorwiegend in sequentiellen Tasks verwendet.

Die Ausführung der Funktion kann jedoch lange dauern, weshalb bei zyklischen Tasks (z. B. BackgroundTask*)* die Zeitüberwachung ansprechen kann. Deshalb kann man die Funktion auch asynchron ausführen, indem man den Parameter *nextCommand* auf IMMEDIATELY setzt. In diesem Fall wird die Funktion gestartet und anschließend sofort der nächste Befehl in der Quelle bearbeitet.

Aus dem Rückgabewert können Sie entnehmen:

- ob der Start erfolgreich war (Komponente functionResult = DONE)
- einen Handle zur weiteren Statusabfrage (Komponente handle)

Bei erfolgreichem Start des Kommandos müssen Sie mit der Funktion *_getStateOfUnitDataSetCommand* und dem Handle den aktuellen Status der Datensicherungsfunktion so lange abfragen, bis das Ergebnis verschieden von ACTIVE ist (siehe Beispiel).

Tabelle 8- 21 Funktionsaufruf einer Datensicherungsfunktion mit Weiterschaltbedingung IMMEDIATELY

```
VAR_GLOBAL
    error : BOOL := FALSE;
    ds_rslt      : EnumDeviceUnitDataSetCommand;
    ds_ret       : StructRetUnitDataSetCommand;
    cmd_busy     : BOOL := FALSE;
    cmd_done     : BOOL := FALSE;
END_VAR

PROGRAM save_data_cycl
    // Programm ist einer zyklischen Task zugeordnet.
    IF NOT cmd_busy THEN
        cmd_busy := TRUE;
        // Funktion asynchron ausführen:
        ds_ret := _saveUnitDataSet (
        unitName := 'ds1',
        id := 1,
        storageType := TEMPORARY_STORAGE,
        overwrite := TRUE,
        nextCommand := IMMEDIATELY);
        IF (ds_ret.functionResult <> DONE) THEN
            cmd_busy := FALSE;
            error := TRUE; // Start der Funktion fehlgeschlagen
            // (z. B. zu viele Dienste)
```

```
        END_IF;
    ELSE
        // Funktion läuft, auf Ergebnis warten:
        ds_rslt := _getStateOfUnitDataSetCommand (
        ds_ret.handle);
        IF (ds_rslt <> ACTIVE) THEN
            cmd_busy := FALSE;
            IF (ds_rslt = DONE) THEN
                cmd_done := TRUE;// Funktion erfolgreich
                // beendet
                ELSE
                error := TRUE;// Funktion fehlgeschlagen
            END_IF;
        END_IF;
    END_IF;
END_PROGRAM
```

8.20.4 Konvertieren zwischen beliebigen Datentypen und Byte-Feldern (Marshalling)

Das Konvertieren von Variablen beliebigen Datentyps in Byte-Felder und umgekehrt wird häufig verwendet, um definierte Übertragungsformate für den Datenaustausch zwischen verschiedenen Geräten zu schaffen (siehe auch Kapitel *Kommunikationsfunktionen*).

Folgende Funktionen stehen zur Verfügung; sie wandeln Variablen beliebigen Datentyps (elementare Datentypen, Standarddatentypen der Technologiepakete und Geräte, anwenderdefinierte Datentypen) in Byte-Felder und umgekehrt:

- AnyType_to_BigByteArray (Seite 384)
- AnyType_to_LittleByteArray (Seite 384)
- BigByteArray_to_AnyType (Seite 386)
- LittleByteArray_to_AnyType (Seite 386)

Bei allen Funktionen kann optional ein Offset für das erste zu belegende oder auszuwertende Element im Byte-Feld angegeben werden.

Es wird unterschieden nach:

- Richtung der Wandlung (von bzw. nach Byte-Felder)
- Anordnung der Bytes im Feld (siehe Tabelle):
  - Big Endian: höchstwertiges Byte an niedriger Speicheradresse (Motorola, SUN Sparc, SIMATIC S7)
  - Little Endian: niedrigstwertiges Byte an niedriger Speicheradresse (Intel, DEC Alpha)

Tabelle 8- 22 Beispiele für Byte-Anordnung (Big Endian und Little Endian

Adresse	Zahl 34677374 = 16#2_11_22_7E (Datentyp UDINT)		Zeichenfolge "Byte" (Datentyp DWORD)	
	Big Endian	Little Endian	Big Endian	Little Endian
2#...11	16#7E = 126	16#02 = 2	16#65 = "e"	16#42 = "B"
2#...10	16#22 = 34	16#11 = 17	16#74 = "t"	16#79 = "y"
2#...01	16#11 = 17	16#22 = 34	16#79 = "y"	16#74 = "t"
2#...00	16#02 = 2	16#7E = 126	16#42 = "B"	16#65 = "e"

ACHTUNG

TO-Datentypen können nicht konvertiert werden (Zur Konvertierung von TO-Datentypen siehe Kapitel *Wandlung von Datentypen technologischer Objekte).*

Wenn Strukturen und Felder gewandelt werden sollen, ist die Konsistenz nur auf dem Niveau elementarer Variablen gewährleistet. Für weitergehende Konsistenz hat der Anwender selbst Sorge zu tragen (siehe Kapitel *Konsistentes Schreiben und Lesen von Variablen (Semaphoren)* und *Variablenhandling*).

Hinweis

Folgende Datentypen sind nicht portabel konvertierbar, d. h. die Übertragungsformate sind nicht systemübergreifend definiert. Konvertierungen zwischen SIMOTION Geräten sind uneingeschränkt möglich:

- Zeittypen: Übertragungsformat siehe Tabelle
- Enumeratoren (Aufzählungsdatentypen)

Beim Konvertieren dieser Datentypen gibt der Compiler eine Warnung aus (16013)

Tabelle 8- 23 Übertragungsformate der Zeitdatentypen bei den Mashalling-Funktionen

Datentyp	Übertragungsformat
TIME	Zeitangabe in ms (UDINT)
TIME_OF_DAY (TOD)	Tageszeit ab TOD#0:0:0 in ms (UDINT)
DATE	Anzahl der Tage seit DATE#1992_01_01 (UDINT). DATE#1992_01_01 entspricht 1
DATE_AND_TIME (DT)	Zusammenfassung von TOD und DATE in der genannten Reihenfolge.

ACHTUNG

Das Ergebnis der Marshalling-Funktionen kann bei Laufzeit des Programms zu Fehlern führen, es wird dann die bei der Taskkonfiguration eingestellte Fehlerreaktion ausgelöst, siehe Verarbeitungsfehler in Programmen (Seite 146).

Besondere Vorsicht ist bei der Konvertierung von Byte-Feldern in den allgemeinen Datentyp ANY_REAL oder in Strukturen geboten, die diesen Datentyp enthalten. Der Bitstring aus dem Byte-Feld wird ungeprüft als ANY_REAL-Wert übernommen. Achten Sie selbst darauf, dass der Bitstring des Byte-Felds dem Bitmuster einer normalisierten Gleitpunktzahl nach IEEE 754 entspricht. Sie können hierzu die Funktionen _finite (Seite 394) und _isNaN (Seite 395) verwenden.

Andernfalls kann ein Fehler (FPU-Exception (Seite 147)) ausgelöst werden, sobald der ANY_REAL-Wert erstmals bei einer Rechenoperation verwendet wird (z. B. im Programm oder beim Beobachten im Symbol-Browser).

Tabelle 8- 24 Beispiel zur Verwendung der Marshalling-Funktionen

```
TYPE
    Struct_1 : STRUCT
        m_word            : WORD;
        m_byte            : BYTE;
    END_STRUCT;
    Struct_2 : STRUCT
        m_struct        : ARRAY [0..2] OF Struct_1;
        m_lreal         : LREAL;
    END_STRUCT;
END_TYPE

VAR
    gsbVar             : Struct_2;
    big_b_Array             : ARRAY [0..16] OF BYTE;
    lit_b_Array             : ARRAY [0..16] OF BYTE;
END_VAR

// Zuweisung der Werte an die Struktur
gsbVar.m_struct[0].m_word        := WORD#16#7FF1;
gsbVar.m_struct[0].m_byte        := BYTE#16#F9;
gsbVar.m_struct[1].m_word        := WORD#16#9FF7;
gsbVar.m_struct[1].m_byte        := BYTE#16#80;
gsbVar.m_struct[2].m_word        := WORD#16#A881;
gsbVar.m_struct[2].m_byte        := BYTE#16#BC;
gsbVar.m_lreal            := LREAL#-12345.6789e123;
// Wandlung nach Big Endian
big_b_Array := AnyType_to_BigByteArray (
                    anyData     := gsbVar,
                    offset         := 0);
 // Inhalt der Elemente von big_b_array (Big Endian):
```

```
 // Siehe 2. Spalte in nachfolgender Tabelle

// Wandlung nach Little Endian
lit_b_Array := AnyType_to_LittleByteArray (
                    anyData     := gsbVar,
                    offset         := 0);
 // Inhalt der Elemente von lit_b_array (Little Endian):
 // Siehe 3. Spalte in nachfolgender Tabelle

// Wandlung von Big Endian
gsbVar := BigByteArray_to_AnyType (
                    bytearray := big_b_Array,
                    offset := 0);

// Wandlung von Little Endian
gsbVar := BigByteArray_to_AnyType (
                    bytearray := lit_b_Array,
                    offset := 0);
```

Tabelle 8- 25 Inhalt der Feldelemente von big_b_array und lit_b_array aus Beispiel

Bytefeld big_b_array bzw. lit_b_array			Komponenten der Variablen gsbVar	
Feld-index	big_b_array (Big Endian)	lit_b_array (Little Endian)	Name	Wert
16	BYTE#16#07	BYTE#16#DA	m_lreal	LREAL#-12345.6789e123
15	BYTE#16#F0	BYTE#16#52		
14	BYTE#16#43	BYTE#16#3C		
13	BYTE#16#68	BYTE#16#EC		
12	BYTE#16#EC	BYTE#16#68		
11	BYTE#16#3C	BYTE#16#43		
10	BYTE#16#52	BYTE#16#F0		
9	BYTE#16#DA	BYTE#16#07		
8	BYTE#16#BC	BYTE#16#BC	m_struct[2].m_byte	BYTE#16#BC
7	BYTE#16#81	BYTE#16#A8	m_struct[2].m_word	WORD#16#A88
6	BYTE#16#A8	BYTE#16#81		
5	BYTE#16#80	BYTE#16#80	m_struct[1].m_byte	BYTE#16#80
4	BYTE#16#F7	BYTE#16#9F	m_struct[1].m_word	WORD#16#9FF7
3	BYTE#16#9F	BYTE#16#F7		
2	BYTE#16#F9	BYTE#16#F9	m_struct[0].m_byte	BYTE#16#F9
1	BYTE#16#F1	BYTE#16#7F	m_struct[0].m_word	WORD#16#7FF1
0	BYTE#16#7F	BYTE#16#F1		

8.20.5 Kommunikationsfunktionen

8.20.5.1 Verfügbare Funktionen

Für die Kommunikation über nicht projektierte Verbindungen stellt ST folgende Funktionen zur Verfügung:

- _Xsend, siehe Parameterbeschreibung für _Xsend (Seite 470)
- _GetStateOfXCommand, siehe Parameterbeschreibung für _GetStateOfXCommand (Seite 472)
- _Xreceive, siehe Parameterbeschreibung für _Xreceive (Seite 471)
- _tcpsend, siehe Kommunikation über Ethernet mit TCP/IP-Protokoll (Seite 476)
- _tcpreceive
- _udpsend, siehe Kommunikation über Ethernet mit UDP-Protokoll (Seite 477)
- _udpreceive

Diese Kommunikationsfunktionen dienen zum Versenden und Empfangen von Daten

- zwischen SIMOTION Geräten,
- zwischen SIMOTION und SIMATIC S7 Geräten (S7-300, S7-400, M7-300, M7-400 usw.).
- über azyklische Kommunikation, siehe auch Azyklische Kommunikation mit dem Antrieb (Seite 477).

Die zweiseitigen Funktionen **_Xsend** und **_Xreceive** erlauben die transparente Übertragung von Datenpaketen, die vom Anwenderprogramm des Clients koordiniert gesendet und vom Anwenderprogramm des Servers koordiniert empfangen werden.

Weitere Informationen

Zusätzliche Informationen finden Sie auch im **System-/Projektierungshandbuch Kommunikation** oder unter Azyklische Kommunikation mit dem Antrieb (Seite 477).

Anhand einer frei wählbaren Ganzzahl, die am Datenpaket angehängt wird, erkennt das Programm des Empfängers, ob es sich um das zu empfangende Datenpaket handelt. Außerdem erfolgt ein erfolgreicher Datenaustausch nur, wenn das Anwenderprogramm des Empfängers das Datenpaket übernimmt und nicht ablehnt.

Bei den gesendeten Daten handelt es sich um Bytefolgen in einem Array, d. h. die Daten besitzen keine logische Struktur. SIMOTION Geräte können maximal 200 Byte an einem Stück senden oder empfangen; die tatsächliche Nutzdatenlänge ist abhängig vom Kommunikationspartner.

Den Zustand eines XSend- bzw. XReceive-Auftrags können Sie mit dem Befehl **_GetStateOfXCommand** abfragen.

8.20.5.2 Parameterbeschreibung für _Xsend

Mit der Funktion _Xsend senden Sie ein Datenpaket mit transparenten Daten an einen Kommunikationspartner. Die ausführliche Syntax der Parameter finden Sie im Listenhandbuch zu den SIMOTION Geräten.

Nachfolgend eine Übersicht:

- Sie können zwischen zwei Kommunikationsmodi (Parameter *communicationMode*) wählen: Verbindung bleibt nach der Datenübertragung bestehen oder nicht.
- Mit dem Parameter *address* wird die Zieladresse des Kommunikationspartners angegeben. Der Parameter ist vom Datentyp *StructXSendDestAddr*. Die folgende Tabelle listet die Bedeutung der einzelnen Komponenten auf.

Tabelle 8- 26 Aufbau der Zieladresse

Parameter / Datentyp	Bedeutung / Werte	Werte	
deviceId (USINT	Anschlusspunkt der Verbindung	SIMOTION C	1 für X8 2 für X9
		SIMOTION P350	1 für X101 2 für X102
		SIMOTION D4x5/D4x5-2	1 für X126 2 für X136
		D410	1 für X21 2 für X200/X201
remoteSubnetIdLength (USINT)	Länge der Subnetzmaske	**0** bei MPI, PROFIBUS	
remoteStaddrLength (USINT)	Länge der Stationsadresse (Teilnehmernummer) des Zielsystems.	**1** bei MPI, PROFIBUS	
nextStaddrLength (USINT)	Länge der Adresse des Routers	**0** bei MPI, PROFIBUS	
remoteSubnetId (ARRAY [0..5] OF USINT)	Subnetzmaske	(ohne Bedeutung bei MPI, PROFIBUS)	
remoteStaddr (ARRAY [0..5] OF USINT)	Stationsadresse des Zielsystems (eigentliche Zieladresse)	Teilnehmernummer bei MPI, PROFIBUS: z. B.: remoteStaddr[0] = 25	
nextStaddr (ARRAY [0..5] OF USINT)	Adresse des Routers	(ohne Bedeutung bei MPI, PROFIBUS)	

Weitere Parameter der Funktion _Xsend sind:

- An der **Auftragskennung** (Parameter *messageId*), die Sie dem Datenpaket hinzufügen, identifiziert der Empfänger das Datenpaket.
- Sie können zwischen zwei **Modi der Datenübertragung** (Parameter *nextCommand*) wählen: synchron oder asynchron.
  - Bei der synchronen Datenübertragung wird mit der Programmfortführung so lange gewartet, bis der Empfang des Datenpaketes vom Empfänger bestätigt wird. Dies geschieht bei Annahme des Datenpaketes automatisch.
  - Bei der asynchronen Datenübertragung wird das Programm sofort nach Absetzen des Befehls fortgeführt. Den Status des Befehls können Sie mit *_GetStateOfXCommand* überprüfen.
- Der Pflicht-Parameter **commandId** dient der internen Erkennung eines Befehls im ST. Der Parameterwert sollte mit der Funktion *_getCommandId* in einer lokalen Variablen gespeichert (Datentyp *CommandIdType*) werden. Diese Variable kann als Parameterwert verwendet werden.
- Bei dem **Datenpaket** (Parameter *data*) handelt es sich um eine Liste mit 200 Einträgen von jeweils einem Byte. Die Liste muss nicht diese Breite haben, wenn Sie weniger als das Maximum an Daten senden.
- Mit der **Datenlänge** (Parameter *dataLength*) geben Sie die tatsächlich zu über tragende Länge des Datenpaketes an.
- Anhand des **Rückgabewertes** können Sie feststellen, ob die Befehlsausführung erfolgreich war (Rückgabewert = 0).

  Bei Rückgabewerten verschieden von 0 ist ein Fehler aufgetreten (siehe die Befehlssyntax im Listenhandbuch zu den SIMOTION Geräten).

8.20.5.3 Parameterbeschreibung für _Xreceive

Mit der Funktion *_Xreceive* empfangen Sie die von einem Kommunikationspartner mit *_Xsend* gesendeten transparenten Daten.

Nachfolgend eine kurze Übersicht der Funktionsparameter (siehe auch Listenhandbuch zu den SIMOTION Geräten, Kapitel Systemfunktionen):

- An der **Auftragskennung** (Parameter *messageId*), das der Sender hinzugefügt hat, identifizieren Sie das erwartete Datenpaket.
- Wie beim Senden können Sie zwischen zwei **Modi des Datenempfangs** (Parameter *nextCommand*) wählen: synchron oder asynchron.
  - Beim synchronen Datenempfang wird mit der Programmfortführung so lange gewartet, bis das Datenpaket angekommen ist. Anschließend wird dem Sender der Empfang automatisch bestätigt.
  - Bei der asynchronen Datenübertragung wird das Programm sofort nach Absetzen des Befehls fortgeführt. Den Status der Befehle können Sie mit *_GetStateOfXCommand* überprüfen.

- Der Pflicht-Parameter **commandId** dient der internen Erkennung eines Befehls im ST. Der Parameterwert sollte mit der Funktion *_getCommandId* (siehe Kapitel Funktionshandbuch *SIMOTION Basisfunktionen*) in einer lokalen Variablen gespeichert (Datentyp *CommandIdType*) werden. Diese Variable kann als Parameterwert verwendet werden.
- Der **Rückgabewert** ist eine Struktur:
  - Anhand des Elements *functionResult* können Sie feststellen, ob die Befehlsausführung erfolgreich war (functionResult = 0).

    Bei Werten verschieden von 0 ist ein Fehler aufgetreten (siehe die Befehlssyntax im Listenhandbuch zu den SIMTOTION-Geräten).
  - Das Element *dataLength* stellt die Datenlänge des empfangenen Datenpaketes dar.
  - Das Element *data* stellt das empfangene Datenpaket dar (Array von bis zu 200 Einträgen von je einem Byte Länge).

8.20.5.4 Parameterbeschreibung für _GetStateOfXCommand

Mit der Funktion _GetStateOfXCommand fragen Sie den Zustand des Befehls *_Xsend* oder *_Xreceive* ab.

Nachfolgend eine kurze Übersicht der Funktionsparameter (siehe auch Dokumentation zu den Technologiefunktionen):

- Anhand des Pflicht-Parameters *commandId*, der jeder Sende- und Empfangsfunktion eindeutig zugeordnet wurde (siehe Erläuterungen zu diesem Parameter für *_Xsend* und *_Xreceive* oben), erkennt die Statusabfrage, um welchen Befehl es sich handelt.
- Der Rückgabewert besteht aus einer Fehlernummer (Null wenn Befehlsausführung in Ordnung, ansonsten größer Null) und dem Zustand des abgefragten Befehls (siehe die Befehlssyntax in den Systemfunktionen zu den SIMOTION Geräten).

8.20.5.5 Kommunikation zwischen SIMOTION und SIMATIC S7 Geräten

Bei der Kommunikation zwischen SIMOTION und SIMATIC S7 Geräten müssen Sie Folgendes beachten:

- Die Kommunikation ist nur mit *_Xsend* und *_Xreceive* möglich.
- Die maximal übertragbare Datenmenge in einem Paket ist auf 76 Byte beschränkt. Wenn Sie größere Datenpakete übertragen, erhalten Sie eine Fehlermeldung.
- Die SIMOTION Schnittstelle muss mit der **MPI-Schnittstelle** der SIMATIC S7 Geräte verbunden werden. An der SIMOTION Schnittstelle muss die Baudrate entsprechend der Baudrate des SIMATIC S7 Gerätes eingestellt werden. Beispielsweise muss bei einer SIMATIC S7-300 die Baudrate auf 187,5 kBit/s konfiguriert werden (siehe Dokumentation zu den entsprechenden SIMATIC S7 Geräten).

ACHTUNG
Die Parameter der Befehle *_Xsend* und *_Xreceive* sind im SIMOTION System anderslautend und haben teilweise eine andere Bedeutung als die, die Sie in Ihrem früheren SIMATIC S7 System verwendet haben. Einen Vergleich sehen Sie in den nachfolgenden Tabellen. Den Befehl *_GetStateOfXCommand* gibt es in einem SIMATIC S7 System nicht, so dass sich der Vergleich erübrigt.

Tabelle 8- 27 Vergleich der Parameter für _Xsend bei SIMATIC S7 und SIMOTION Geräten

SIMATIC S7 Gerät (Parameter für SFC 65 X_SEND)	SIMOTION Gerät (Parameter für _Xsend)
REQ (Request to activate, Datentyp BOOL)	– (nicht vorhanden)
DEST_ID (MPI-Stationsnummer des Kommunikationspartners)	address (Zieladresse des Kommunikationspartners als Struktur vom Datentyp StructXSendDestAddr)
CONT (Fortsetzen der Verbindung, boolescher Wert TRUE oder FALSE)	communicationMode (Fortsetzen der Verbindung, Enum.-Wert M_TRUE oder M_FALSE)
– (nicht vorhanden)	nextCommand (Modus der Datenübertragung, Enum.-Werte für synchron und asynchron)
REQ_ID (Auftragskennung, Wert vom Datentyp DWORD)	messageId (Auftragskennung, Wert vom Datentyp UDINT)
SD (Variablenadresse für gesendete Daten, Wert vom Datentyp ANY-Pointer, max. 76 Byte lang)	data (zu sendendes Datenpaket, eindim. Array mit max. 200 Werten vom Datentyp USINT))
– (nicht vorhanden)	dataLength (Datenlänge des zu sendenden Datenpakets in Byte, Wert vom Datentyp UDINT)
– (nicht vorhanden)	commandId (interne Befehlskennung, siehe Erläuterungen in der Parameterbeschreibung oben)

SIMATIC S7 Gerät (Parameter für SFC 65 X_SEND)	SIMOTION Gerät (Parameter für _Xsend)
BUSY (Aktivierungszustand, Werte vom Datentyp BOOL)	– (nicht vorhanden)
<Rückgabewert> (= 0, wenn kein Fehler <> 0, wenn Fehler; siehe Dokumentation SIMATIC S7	<Rückgabewert, (= 0, wenn kein Fehler; <> 0, wenn Fehler, siehe Listenhandbuch zu den SIMOTION Geräten)

Tabelle 8- 28 Vergleich der Parameter für _Xreceive bei SIMATIC S7 und SIMOTION Geräten

SIMATIC S7 Gerät (Parameter für SFC 66 X_RCV)	SIMOTION Gerät (Parameter für _Xreceive)
EN_DT (Enable Data Transfer, Datentyp BOOL)	– (nicht vorhanden)
REQ_ID (Auftragskennung, Wert vom Datentyp DWORD)	messageId (Auftragskennung, Wert vom Datentyp UDINT)
– (nicht vorhanden)	nextCommand (Modus des Datenempfangs, Enum.-Werte für synchron und asynchron)
– (nicht vorhanden)	commandId (interne Befehlskennung, siehe Erläuterungen in der Parameterbeschreibung oben)
RD (Variablenadresse für empfangene Daten, Wert vom Datentyp ANY-Pointer, max. 76 Byte lang)	<Rückgabewert, Strukturelement data> (empfangenes Datenpaket; ein Array von bis zu 200 Einträgen von je einem Byte Länge)
– (nicht vorhanden)	<Rückgabewert, Strukturelement dataLength> (Datenlänge des empfangenen Datenpaketes)
<Rückgabewert> (= 0, wenn kein Fehler <> 0, wenn Fehler; siehe Dokumentation SIMATIC S7	<Rückgabewert, Strukturelement functionResult> (= 0, wenn kein Fehler; <> 0, wenn Fehler, siehe Listenhandbuch zu den SIMITION Geräten)

8.20.5.6 Beispiel Sende- und Empfängerprogramm

Die folgenden Bilder zeigen den Quelltext für das Sende- und für das Empfängerprogramm:

Tabelle 8- 29 Beispiel für das Sendeprogramm

```
INTERFACE
    PROGRAM xsend_control;
END_INTERFACE

IMPLEMENTATION
                          // Das folgende Programm muss einer MotionTask
                          // zugeordnet werden.
                          // In der Taskkonfiguration muss die Option "Aktivierung
                          // nach Startup Task" angewählt sein.
PROGRAM xsend_control
VAR
    retVal                          : DINT;
    myAddress                       : StructXSendDestAddr;
    myStaddr                        : ARRAY[0..4] OF BYTE;
    myData                          : ARRAY[0..199] OF BYTE;
END_VAR
                          // Zieladresse und PROFIBUS-Schnittstelle festlegen ////////
myAddress.deviceID                  := 1;
                          // PROFIBUS-Schnittstelle X8
myAddress.remoteStaddrLength        := 1;
                          // ist immer mit 1 zu belegen
myAddress.remoteStaddr[0]           := 4;
                          // PROFIBUS-Adresse der Empfangsstation

                          // Sendedaten
myData[0] := 170;

                          // Aufruf der Sendefunktion
retVal := _Xsend( communicationMode := ABORT_CONNECTION
             , address              := myAddress
             , messageId            := 1
             , nextCommand          := WHEN_COMMAND_DONE
             , commandId            := _getCommandId()
             , data                 := myData
             , dataLength           := 1
             );
END_PROGRAM
END_IMPLEMENTATION
```

Tabelle 8- 30 Beispiel für das Empfängerprogramm

```
INTERFACE
    PROGRAM xreceive_control;
END_INTERFACE

IMPLEMENTATION
VAR_GLOBAL
    retVal                          : StructRetXReceive;
END_VAR
                           // Das folgende Programm muss einer MotionTask
                           // zugeordnet werden.
                           // In der Taskkonfiguration muss die Option "Aktivierung
                           // nach Startup Task" angewählt sein.
PROGRAM xreceive_control

// Aufruf der Empfangsfunktion
retVal := _Xreceive( messageId       := 1
            , nextCommand            := WHEN_COMMAND_DONE
            , commandId              := _getCommandId()
            );
END_PROGRAM
END_IMPLEMENTATION
```

8.20.5.7 Kommunikation über Ethernet mit TCP/IP-Protokoll

Bei SIMOTION Geräten mit Ethernet-Schnittstelle ist auch die Kommunikation über das TCP/IP-Protokoll möglich.

Die folgende Tabelle listet die einzelnen Schritte auf, um eine Kommunikation zwischen einem Sender (Client) und einem Empfänger (Server) herzustellen:

Tabelle 8- 31 Kommunikationsschritte einer TCP/IP-Verbindung und entsprechende Systemfunktionen

Kommunikationsschritte		Systemfunkion
Kommunikationsverbindung aufbauen		
1	Empfänger (Server) wartet auf Kommunikationsanforderung.	_tcpOpenServer
2	Sender (Client) fordert einen Verbindungsaufbau zum Empfänger.	_tcpOpenClient
3	Empfänger (Server) hat Verbindungsanforderung aufgebaut.	
Kommunizieren		
4	Sender sendet Daten an den Empfänger.	_tcpSend
5	Empfänger empfängt Daten vom Sender.	_tcpReceive
Kommunikationsverbindung beenden		
6	Sender sendet keine Daten mehr und schließt die Verbindung.	_tcpCloseConnection
7	Es wird keine weitere Verbindung mehr benötigt.	_tcpCloseServer

Die Systemfunktionen sind ausführlich im Listenhandbuch zu den SIMOTION Geräten, Kapitel Systemfunktionen, beschrieben.

Hinweis

Ein Sender oder Empfänger kann beim Verbindungsaufbau sowohl Client als auch Server sein.

Bei dem TCP/IP Verbindungsaufbau muss es mindestens einen Client und einen Server geben.

Die Client-Server-Beziehung gilt nur bis zum Abschluss des Verbindungsaufbaus. Nach dem Verbindungsaufbau sind beide Kommunikationspartner gleichwertig, d. h. jeder der beiden kann zu einem beliebigen Zeitpunkt senden oder empfangen oder die Verbindung schließen.

Eine ausführliche Beschreibung finden Sie unter den *Frequently Asked Questions* (FAQ) auf *SIMOTION Utilities & Applications* (DVD2 - CD_14) unter FAQs.

8.20.5.8 Kommunikation über Ethernet mit UDP-Protokoll

Bei SIMOTION Geräten mit Ethernet-Schnittstelle ist auch die Kommunikation über das UDP-Protokoll möglich.

- Mit der Funktion *_udpSend* senden Sie ein Datenpaket an einen Kommunikationspartner, den Sie mit IP-Adresse und Port-Nummer spezifizieren.

  Die zu sendenden Daten übergeben Sie als ARRAY [0..1399] OF BYTE an die Funktion.

- Mit der Funktion *_udpReceive* empfangen Sie ein Datenpaket, das ein Kommunikationspartner mit *_udpSend* gesendet hat.

  Die empfangenen Daten werden in einer Variable vom Datentyp ARRAY [0..1399] OF BYTE abgelegt, deren Bezeichner sie als Parameter an die Funktion übergeben.

Die Systemfunktionen sind ausführlich im Listenhandbuch zu den SIMOTION Geräten, Kapitel Systemfunktionen, beschrieben.

Eine ausführliche Beschreibung finden Sie auf *SIMOTION Utilities & Applications* (DVD2 - CD_14) unter FAQs.

8.20.5.9 Azyklische Kommunikation mit dem Antrieb

Übersicht

PROFIdrive-Antriebsgeräte werden mit Steuer-Signalen und Soll-Werten von der Steuerung versorgt und liefern Status-Signale und Ist-Werte zurück. Diese Signale werden normalerweise zyklisch (d. h. ständig) zwischen Steuerung und Antrieb übertragen.

Beim SINAMICS S110/S120 projektieren Sie hierzu die Achstelegramme für den Datenaustausch (siehe Adressen und Telegramme einrichten - Übersicht (Seite 101)).

Neben dem zyklischen Datenaustausch verfügen PROFIdrive-Antriebsgeräte auch über einen azyklischen Kommunikationskanal. Dieser wird insbesondere für das Lesen und Schreiben von Antriebs-Parametern (z. B. Fehler-Codes, Warnungen, Reglerparameter, Motordaten,...) verwendet.

D. h. die Daten werden nicht "zyklisch", sondern bedarfsorientiert "azyklisch" übertragen. Das azyklische Lesen und Schreiben von Parametern bei PROFIdrive-Antrieben erfolgt dabei über die DP-V1-Dienste "Datensatz lesen" und "Datensatz schreiben".

Die azyklischen DP V1-Dienste werden parallel zu der zyklischen Kommunikation über PROFIBUS bzw. PROFINET übertragen. Das PROFIdrive Profil legt fest, wie genau diese grundlegenden Mechanismen für die Lese-/Schreibzugriffe auf Parameter eines PROFIdrive-konformen Antriebs genutzt werden.

Die PROFIdrive Norm legt dabei fest, dass in PROFIdrive Antrieben kein Pipelining von Aufträgen unterstützt wird. Dieses bedeutet:

- Zu einem Antriebsgerät (z. B. SINAMICS S110/S120 Control Unit oder SINAMICS Integrated einer SIMOTION D) ist immer nur ein "Datensatz Schreiben/Lesen" gleichzeitig möglich.
- Sind mehrere PROFIdrive-Antriebsgeräte an einer Steuerung angeschlossen, so kann zu jedem dieser Antriebsgeräte je ein Auftrag zeitlich parallel abgewickelt werden. Die Höchstzahl aller Aufträge in Summe ist dann steuerungsabhängig (bei SIMOTION maximal acht Aufträge gleichzeitig).

Für den azyklischen Datenaustausch mit SINAMICS-Antrieben bedeutet dies, dass Sie die Schreib- und Leseaufträge untereinander koordinieren müssen (= Buffermanagement). Es muss verriegelt werden, dass die Applikation bzw. unterschiedliche Teile der Applikation gleichzeitig bzw. überlappend Aufträge an dasselbe PROFIdrive-Antriebsgerät senden.

Weitere Literatur

Weitere Informationen zur Handhabung von DP-V1-Diensten finden Sie im Systemhandbuch *SIMOTION Kommunikation*.

In den *SIMOTION Utilities & Applications* finden Sie zudem eine DP-V1 Bibliothek mit Funktionen, welche typische Koordinierungsaufgaben im Zusammenhang mit azyklischer Kommunikation übernehmen. Die Bibliothek koordiniert dabei nicht nur den Zugriff der Systemfunktionen _ReadRecord / _WriteRecord / _readDriveParameter / _writeDriveParameter/..., sondern erweitert auch den Funktionsumfang für häufig benötigte Aufgabenstellungen wie z. B. das Auslesen von Fehlern und Warnungen aus dem Antriebsgerät.

Die *SIMOTION Utilities & Applications sind* im Lieferumfang von SIMOTION SCOUT enthalten.

Folgende Funktionen stehen in der DP-V1 Bibliothek u. a. zur Verfügung:

- Buffermanagement (Koordinierung von mehreren parallelen DP-V1-Diensten)
- StartUp (Funktion, um den Hochlauf des SINAMICS Antriebs mit SIMOTION zu koordinieren)
- TimeSync (applikative Uhrzeitsynchronisation: Übernahme der SIMOTION Uhrzeit in SINAMICS Antriebe)

- SetActIn (Aktivieren und Deaktivieren von Objekten in SIMOTION und in SINAMICS)
- RwnPar (Lesen und Schreiben von Antriebsparametern)
- GetFault (Fehler und Warnung vom Antrieb lesen)

Siehe auch

Verfügbare Funktionen (Seite 469)

8.20.6 Synchroner Start

Beim synchronen Start werden mehrere Befehle innerhalb eines IPO-Taktes bzw. IPO_2-Taktes gestartet.

So gehen Sie vor:

1. Zuerst lassen Sie sich eine eindeutige SyncCommandId vom System geben. Sie benötigen diese SyncCommandId zur eindeutigen Kennzeichnung der synchronen Befehle.

   Verwenden Sie hierzu die Systemfunktion _getSyncCommandId().

2. Die Befehle, die synchron ausgeführt werden sollen, schließen Sie durch die Funktionen BEGIN_SYNC(SyncCommandId) und END_SYNC() ein. Sie sind damit für den synchronen Start definiert..

   Innerhalb der Struktur BEGIN_SYNC / END_SYNC sind alle Bewegungsbefehle erlaubt. Als Parameter nextCommand muss der Wert IMMEDIATELY übergeben werden.

3. Der synchrone Start selbst erfolgt mit der Funktion _startSyncCommand (SyncCommandId). Die für den synchronen Start definierten Befehle werden parallel abgearbeitet.

Siehe Beispielprogramm.

ACHTUNG
Der synchrone Start ist bei Bewegungsbefehlen nur dann gewährleistet, wenn nachstehende Bedingungen erfüllt sind: 1. Die im synchronen Start eingeschlossenen Befehle müssen auf verschiedene Technologieobjekte wirken. 2. Die beteiligten Technologieobjekte müssen im gleichen ExecutionLevel (IPO bzw. IPO_2) liegen. 3. Bei Verwendung des Befehls Synchroner Start in MCC muss die UserInterruptTask_1 angelegt werden, da diese bei einem Fehler aufgerufen wird. In dieser Task kann eine Fehlerreaktion programmiert werden, siehe UserInterruptTasks (Seite 228) .

Zum Zeitpunkt des Synchronen Starts wird die Tasksteuerung temporär ausgeschaltet. Sie wird erst wieder eingeschaltet:

- wenn alle in den Zweigen vorhandenen Einzelachsbefehle und Befehle für Gleichlauf und Kurvenscheibe gestartet sind und
- wenn alle in den Zweigen vorhandenen Basisbefehle beendet sind.

Unterbrechungen des Starts durch andere Tasks (außer SynchronousTasks) werden dadurch verhindert. Dies kann zu einem Zeitüberlauf in zyklischen Tasks (BackgroundTask, TimerInterruptTasks) führen. Diesen Fehler können Sie durch geeignete Programmierung der TimeFaultBackgroundTask bzw. TimeFaultTask erkennen und abfangen.

Die Funktionen BEGIN_SYNC, END_SYNC und *_startSyncCommand* sind Systemfunktionen der SIMOTION Geräte; nähere Erläuterungen finden Sie deshalb im Listenhandbuch des entsprechenden SIMOTION Geräts.

Hinweis

Oft wird der synchrone Start zusammen mit dem WAITFORCONDITION-Konstrukt verwendet. Hierbei wird vor dem synchronen Start das Eintreffen einer Bedingung abgewartet. Die Funktion *_startSyncCommand* darf nicht innerhalb des WAITFORCONDITION-Konstrukts auftreten.

Tabelle 8- 32 Beispielprogramm für synchronen Start zweier Achsen mit WAITFORCONDITION

```
INTERFACE
    USEPACKAGE cam;
    PROGRAM sync_motion;
END_INTERFACE

IMPLEMENTATION

    EXPRESSION wait_sync_expression
        wait_sync_expression := TRUE;
    END_EXPRESSION

    PROGRAM sync_motion
        VAR
            ret_val : DINT;
            sync_id : CommandIdType;
        END_VAR

        sync_id := _getSyncCommandId();
        BEGIN_SYNC(sync_id);
        (* Positioniere Achse ('Pos') *)
        ret_val := _pos    (axis := Axis_1,
                    positioningMode := ABSOLUTE,
                    position := 100,
                    mergeMode := IMMEDIATELY,
                    nextCommand := IMMEDIATELY,
```

```
                    commandId := _getCommandId() );
        (* Positioniere Achse ('Pos') *)
        ret_val := _pos    (axis := Axis_2,
                    positioningMode := ABSOLUTE
                    position := 50
                    mergeMode := IMMEDIATELY
                    nextCommand := IMMEDIATELY,
                    commandId := _getCommandId() );
    END_SYNC();

        WAITFORCONDITION wait_sync_expression DO
            ;
        END_WAITFORCONDITION;

        ret_val := _startSyncCommands(sync_id);
        // Im Fehlerfall hier Start der UserInterruptTask
        //IF (_ret_val <> 0) THEN
        _startTask(UserInterruptTask_1);
END_IF;

    END_PROGRAM
END_IMPLEMENTATION
```

Um die korrekte Beendigung des Synchronen Starts zu prüfen, können Sie abfragen, ob die Befehle der beiden im vorherigen Beispiel synchron gestarteten Achsen ohne Fehler beendet wurden.

Das nachfolgende MCC-Beispielprogramm zeigt einen Synchronen Start mit zwei Achsen und anschließender Abfrage der Sammelvariable _MccRetSyncStart. Diese ist ungleich 0, wenn ein Befehl innerhalb von Sync-Start einen Rückgabewert ungleich 0 hatte.

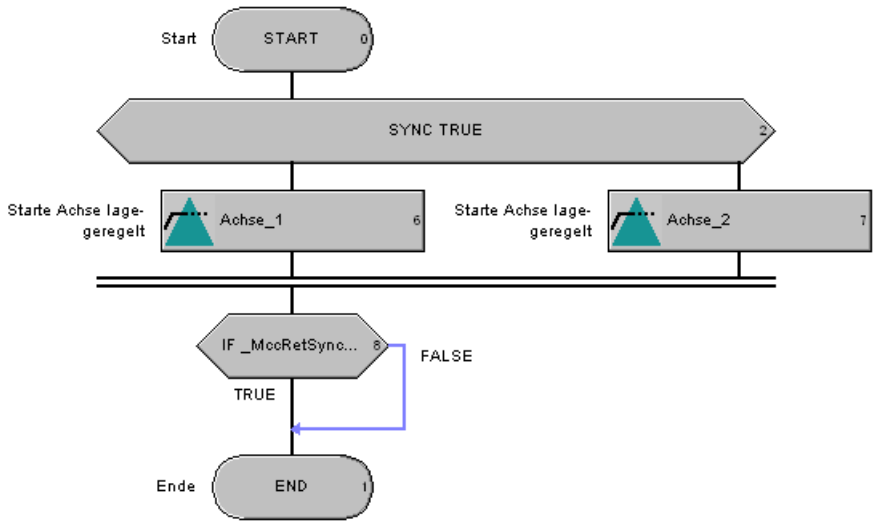


Bild 8-1 MCC Synchroner Start

Nachfolgend der Ausschnitt des entsprechenden ST-Programms (hier über Export aus MCC-Quelle erzeugt)

Tabelle 8- 33 ST Synchroner Start

```
(* Synchroner Start ('StartSync') *)

     _MccCommand1 := _getCommandId();
     _MccCommand2 := _getCommandId();

     _MccSync := _getSyncCommandId();
        BEGIN_SYNC(_MccSync);
           (* Starte Achse lagegeregelt ('Move') *)
        _MccRetDINT_1:=_move(axis:=Achse_1, velocityType:=DIRECT,
           velocity:=v_blue, moveTimeOutType:=WITHOUT_TIME_LIMIT,
           mergeMode:=IMMEDIATELY, nextCommand:=IMMEDIATELY,
           commandId:=_MccCommand1, movingMode:=POSITION_CONTROLLED);
           (* Starte Achse lagegeregelt ('Move') *)
        _MccRetDINT_2:=_move(axis:=Achse_2, velocityType:=DIRECT,
            velocity:=v_red, moveTimeOutType:=WITHOUT_TIME_LIMIT,
            mergeMode:=IMMEDIATELY, nextCommand:=IMMEDIATELY,
            ommandId:=_MccCommand2, movingMode:=POSITION_CONTROLLED);

        (* Synchroner Start ('EndSync') *)
        END_SYNC();

           WAITFORCONDITION _MccMCC_1Condition1 DO
            ;
        END_WAITFORCONDITION;

        _MccRetDINT := _startSyncCommands(_MccSync);
// Start der UserInterruptTask
        IF (_MccRetDINT <> 0) THEN
            _startTask(UserInterruptTask_1);
        END_IF;

        _MccRetStructRetMotionCommandState := _getMotionStateOfAxisCommand
         (Achse_1, _MccCommand1);

//Abfrage, ob beide Achsbefehle fertig sind

        WHILE ((_MccRetStructRetMotionCommandState.functionResult = 0 )
           AND (_MccRetStructRetMotionCommandState.motionCommandIdState <>
           IN_CONSTANT_MOTION) AND
           (_MccRetStructRetMotionCommandState.motionCommandIdState <>
            NOT_EXISTENT)) DO
            _MccRetDINT := _waitTime(T#0d_0h_0m_0s_0ms);
            _MccRetStructRetMotionCommandState :=
            _getMotionStateOfAxisCommand(Achse_1, _MccCommand1);
        END_WHILE;
```

```
_MccRetStructRetMotionCommandState :=
 _getMotionStateOfAxisCommand(Achse_2, _MccCommand2);

WHILE ((_MccRetStructRetMotionCommandState.functionResult = 0 )
   AND (_MccRetStructRetMotionCommandState.motionCommandIdState <>
   IN_CONSTANT_MOTION) AND
   (_MccRetStructRetMotionCommandState.motionCommandIdState <>
   NOT_EXISTENT)) DO
   _MccRetDINT := _waitTime(T#0d_0h_0m_0s_0ms);
   _MccRetStructRetMotionCommandState :=
   _getMotionStateOfAxisCommand(Achse_2, _MccCommand2);
END_WHILE;

_MccRetSyncStart := 0; //Abfrage ob beide Achsbefehle ohne Fehler laufen
IF (_MccRetDINT_1 + _MccRetDINT_2 <> 0) THEN
    _MccRetSyncStart := -1;
END_IF;

(* IF: Programmverzweigung ('If') *)
IF (_MccRetSyncStart <> 0) THEN
;//Fehlerbehandlung

(*  ('Else') *)
ELSE
;

(*  ('EndIf') *)
END_IF;
```

8.21 Kopplung HMI (Human Machine Interface)

8.21.1 Schnittstelle HMI und SCOUT bzw. SIMOTION

SIMOTION erlaubt die Kommunikation mit HMI-Geräten (Bediengeräte für den Endanwender), z. B. mit Operator Panels.

Wenn eines oder mehrere SIMOTION Geräte am PROFIBUS oder PROFINET angeschlossen sind, kann das HMI-Gerät Variablen, Meldungen und Alarme der SIMOTION Geräte anzeigen. Umgekehrt können Sie über das HMI-Gerät und ein Programm programmierte Funktionen im SIMOTION Gerät auslösen.

Für die Realisierung der genannten Aufgaben am HMI-Gerät steht Ihnen das Softwarepaket WinCC flexible zur Verfügung. Mit WinCC flexible können Sie auf Systemvariablen und im Interfaceabschnitt deklarierte Unit-Variablen lesend und schreibend zugreifen. Siehe auch HMI-Variablen in einer eigenen Unit (Seite 568) .

Darüber hinaus stellt SIMOTION einen OPC-Server zur Verfügung. Über ihn können Sie mit herstellerunabhängiger Bedien- und Beobachtungssoftware auf die Prozessdaten zugreifen. In der Projektierungssoftware für WinCC flexible sowie mittels OPC kann direkt auf die symbolischen Variablen eines SIMOTION Gerätes zugegriffen werden. Die Projektierungsdaten von WinCC flexible müssen sich hierzu im selben Projekt wie die SIMOTION Geräte befinden.

Hinweis

Variablen, die in HMI verfügbar sein sollen, müssen immer als Unit-Variablen im Interfaceabschnitt einer Quelle angelegt werden.

Weitere Informationen zu WinCC flexible

Hinweise zur Projektierung mit WinCC flexible sind zu finden

- in der Onlinehilfe von WinCC flexible
- in der Onlinehilfe von SCOUT, wenn WinCC flexible im SCOUT integriert ist

Zusätzlich sind im Internet weitere FAQs hinterlegt unter:

http://support.automation.siemens.com/WW/view/de/28767398

http://support.automation.siemens.com/WW/view/de/23751257

Zu folgenden Themen finden Sie weitere Informationen auf SIMOTION Utilities & Applications unter FAQs:

- Uhrzeitsynchronisation SIMOTION
- WinCC/WinCCFlexible
- SIMOTION und HMI (paralleles Arbeiten am SIMOTION- und HMI Projekt)

Beispiele für HMI-Projektierungen sind in den auf SIMOTION Utilities & Applications unter Applikationen enthaltenen Standardapplikationen zu finden.

- Beispiele für HMI Implementierung

Siehe auch HMI-Variablen in einer eigenen Unit (Seite 568) .

Alarme

Alles, was in der Alarmliste im Scout angezeigt wird, kann auch in WinCC flexible bei entsprechender Projektierung angezeigt werden, außer SINAMICS Alarme!

Bei OPC Alarm&Event funktionieren alle SIMOTION und SINAMICS Alarme.

Der Mechanismus ist heute wie folgt:

Beim Generieren des HMI-Projekts werden die Scout-Texte (Achsnamen, Achs-Alarme, Alarm_S, ...) an das HMI synchronisiert; jedes TO hat dabei eine Nummer.

Bei Eintreten eines Alarms wird also TO-Nummer, Alarm-Nummer, Beiwert, ... gemeldet und das HMI schreibt dann im Klartext TO-Name, Alarmtext mit Beiwert, Zeitstempel, ...

Warn- und Fehlermeldungen der Technologieobjekte (Achsen, Nocken, Messtaster etc.) werden bei WinCC flexible direkt auf dem HMI ausgegeben und werden durch den Bediener quittiert. Mit dem AlarmS-Konzept steht Ihnen außerdem ein Verfahren zur Projektierung und Programmierung von Anwendermeldungen zur Verfügung. Projektiert werden Anwendermeldungen in SIMOTION SCOUT, zur Laufzeit aufgerufen und quittiert werden die Meldungen durch Aufruf entsprechender Systembefehle. OPC unterstützt die gleichen Meldungsmechanismen. Beim OPC-Export ist hierzu zusätzlich OPC-Alarm/Event zu aktivieren.

8.21.2 Konsistenter Datenzugriff mit HMI-Geräten (Beispiel)

Auch HMI-Geräte können konsistent auf Anwenderdaten des SIMOTION Geräts zugreifen. Das folgende Beispiel umfasst ein ST-Programm auf dem SIMOTION Gerät (siehe Beispiel) und eine Applikation (Visual Basic-Programm) auf dem HMI-Gerät (siehe Beispiel).

Der Anwender fordert von der HMI-Applikation aus (hier Visual Basic; könnte auch Visual C++ ... sein) das konsistente Lesen von Anwendervariablen (inkl. Arrays) an. Dazu wird die Variable consistencyFlag mit einem ungeraden Wert beschrieben. Daraufhin kopiert das SIMOTION Anwenderprogramm die Daten. Die HMI-Applikation wartet, bis das SIMOTION Anwenderprogramm das Ende des Kopiervorgangs bestätigt, indem ein gerader Wert in die Variable *consistencyFlag* geschrieben wird. Daraufhin liest HMI diese Variablen konsistent aus, da niemand zwischendurch die Daten verändert.

Das ST-Programm kann einer zyklischen Task zugewiesen werden; eventuell muss die IPOsynchronousTask genommen werden, wenn z. B. Achs-Positionen und -Geschwindigkeiten zum selben Zeitpunkt (als Snapshot) gelesen werden sollen.

Beispiele:

Tabelle 8- 34 ST-Programm für konsistenten Datenzugriff von HMI-Geräten

```
INTERFACE
    VAR_GLOBAL
        consistencyFlag : DINT;
        myDint    : DINT;
        myArray: ARRAY[0..10] OF LREAL;
    END_VAR
    PROGRAM OPC_Prog;
END_INTERFACE

IMPLEMENTATION
    PROGRAM OPC_Prog
        IF (consistencyFlag MOD 2) = 1 THEN
            myDint := 99;
            myArray[0]    := 0.0;
            myArray[1]    := 1.0;
            myArray[10]    := 10.0;
            consistencyFlag := consistencyFlag + 1;
        END_IF;
    END_PROGRAM
END_IMPLEMENTATION
```

Tabelle 8- 35 HMI-Applikation für konsistenten Datenzugriff (Visual Basic)

```
Option Explicit
Option Base 1

    Dim g_Server As OPCServer
    Dim g_GroupObj As OPCGroup

    Dim g_myItem1 As OPCItem
    Dim g_myItem2 As OPCItem

    Dim g_myItem3 As OPCItem
    Const OPC_DS_DEVICE = 2

    Dim consistencyFlag As Long

Private Sub Form_Load()
    Set g_Server = New OPCServer
    g_Server.Connect ("OPC.SimaticNet")
    Set g_GroupObj = g_Server.OPCGroups.Add("Test1")
    g_GroupObj.IsActive = False
    Set g_myItem1 = g_GroupObj.OPCItems.AddItem("C240.consistencyFlag", 2)
    Set g_myItem2 = g_GroupObj.OPCItems.AddItem("C240.myDint", 2)
```

```
    Set g_myItem3 = g_GroupObj.OPCItems.AddItem("C250.myArray", 3)
    consistencyFlag = 1
End Sub
Private Sub Form_Unload(Cancel As Integer)
    Set g_myItem1 = Nothing
    Set g_myItem2 = Nothing
    Set g_myItem3 = Nothing
    Set g_GroupObj = Nothing
    Set g_Server = Nothing
    End
End Sub

Private Sub cmdRead_Click()
    Static reentrancyFlag As Boolean
    Dim var1 As Variant
    Dim var2 As Variant
    Dim var3 As Variant

    If reentrancyFlag = False Then
        reentrancyFlag = True
        consistencyFlag = consistencyFlag + 2
        g_myItem1.Write consistencyFlag
    Do
        g_myItem1.Read OPC_DS_DEVICE, var1
    Loop Until var1 = consistencyFlag + 1
        g_myItem2.Read OPC_DS_DEVICE, var2
        g_myItem3.Read OPC_DS_DEVICE, var3
        reentrancyFlag = False
    End If
End Sub
```

Programmierung allgemeiner Systemfunktionsbausteine

9

9.1 Übersicht der Funktionsbausteine

ST besitzt eine Reihe von Systemfunktionsbausteinen, die Sie in Ihren ST-Quellen verwenden können, ohne sie zuvor vereinbaren zu müssen. Sie müssen lediglich eine Instanz anlegen und diese mit den erforderlichen Parametern versorgen.

Übersicht der Systemfunktionsbausteine

Nachfolgend sehen Sie eine Übersicht aller implementierten Systemfunktionsbausteine, die Definition und weitere Spezifikationen finden Sie ab Kapitel 7.1.

Tabelle 9- 1 Systemfunktionsbausteine

Sammelbegriff	Name	Eingangsparameter		Ausgangsparameter	
	Umschreibung	Name	: Typ	Name	: Typ
Bistabile Funktionsbausteine	**SR**[1] vorrangig setzen	S1 R	: BOOL; : BOOL;	Q1	: BOOL;
	RS[1] vorrangig rücksetzen	SR1	: BOOL; : BOOL;	Q1	: BOOL;
Flankenerkennung	**R_TRIG**[1] Steigende Flanke	CLK	: BOOL;	Q	: BOOL;
	F_TRIG[1] Fallende Flanke	CLK	: BOOL;	Q	: BOOL;
Zähler	**CTU**[1] Aufwärtszähler Datentyp: INT	CU R PV	: BOOL; : BOOL; : INT;	Q CV	: BOOL; : INT;
	CTU_DINT[1] Aufwärtszähler Datentyp: DINT	CU R PV	: BOOL; : BOOL; : DINT;	Q CV	: BOOL; : DINT;
	CTU_UDINT[1] Aufwärtszähler Datentyp: UDINT	CU R PV	: BOOL; : BOOL; : UDINT;	Q CV	: BOOL; : UDINT;
	CTD[1] Abwärtszähler Datentyp: INT	CD LD PV	: BOOL; : BOOL; : INT;	Q CV	: BOOL; : INT;
	CTD_DINT[1] Abwärtszähler Datentyp: DINT	CD LD PV	: BOOL; : BOOL; : DINT;	Q CV	: BOOL; : DINT;

Sammelbegriff	Name	Eingangsparameter		Ausgangsparameter	
	CTD_UDINT[1] Abwärtszähler Datentyp: UDINT	CD LD PV	: BOOL; : BOOL; : UDINT;	Q CV	: BOOL; : UDINT;
	CTUD[1] Auf-, Abwärtszähler Datentyp: INT	CU CD R LD PV	: BOOL; : BOOL; : BOOL; : BOOL; : INT;	QU QD CV	: BOOL; : BOOL; : INT;
	CTUD_DINT[1] Auf-, Abwärtszähler Datentyp: DINT	CU CD R LD PV	: BOOL; : BOOL; : BOOL; : BOOL; : DINT;	QU QD CV	: BOOL; : BOOL; : UINT;
	CTUD_UDINT[1] Auf-, Abwärtszähler Datentyp: UDINT	CU CD R LD PV	: BOOL; : BOOL; : BOOL; : BOOL; : UDINT;	QU QD CV	: BOOL; : BOOL; : UDINT;
Zeitgeber	**TP**[1] Puls	IN PT	: BOOL; : TIME;	Q ET	: BOOL; : TIME;
	TON[1] Einschaltverzögerung	IN PT	: BOOL; : TIME;	Q ET	: BOOL; : TIME;
	TOF[1] Einschaltverzögerung	IN PT	: BOOL; : TIME;	Q ET	: BOOL; : TIME;
	RTC[1] Echtzeituhr	SET READ PDT	: BOOL; : BOOL; : DT;	CDT	: DT;
Zerlegen von Bitstring-Datentypen	**_BYTE_TO_8BOOL**	n	: BYTE	bit0, bit1, bit2, bit3, bit4, bit5, bit6, bit7	: BOOL;
	_WORD_TO_2BYTE	in	: WORD	byte0, byte1	: BYTE;
	_DWORD_TO_2WORD	in	: DWORD	word0, word1	: WORD;
	_DWORD_TO_4BYTE	in	: DWORD	byte0, byte1, byte2, byte3	: DWORD;

Sammelbegriff	Name	Eingangsparameter		Ausgangsparameter	
Emulation von SIMATIC S7-Befehlen	**_S7_COUNTER**[1]	CU CD PV S R	: BOOL; : BOOL; : INT; : BOOL; : BOOL;	Q CV	: BOOL; : INT;
	_S7_TIMER[1]	T_Type PR S R TV TV_BCD	: WORD; : BOOL; : BOOL; : BOOL; : TIME; : WORD;	Q BI	: BOOL; : TIME;
[1] Diese Funktionsbausteine setzen einen wiederholten Aufruf (z. B. in einer zyklischen Task) voraus					

Vergleich der Systemfunktionsbausteine zwischen SIMOTION und SIMATIC

Eine Gegenüberstellung der SIMATIC S7 und SIMOTION Systemfunktionsbausteine finden Sie im Verzeichnis 2_FAQ auf der Utilities & Applications CD.

9.2 Bistabile Elemente (Flipflop setzen)

Im ST können Sie mit den Systemfunktionsbausteinen SR das Flipflop setzen rücksetzen (vorrangig setzen) und mit RS rücksetzen setzen (vorrangig rücksetzen).

Bistabiler Funktionsbaustein SR (vorrangig setzen)

Am Ausgang Q1 kann der gespeicherte Wert abgegriffen werden. Der Ausgang Q1 ist 1, wenn S1 gleich 1 ist. Wenn S1 und R gleich 0 sind, ändert sich der Ausgang Q1 nicht.)

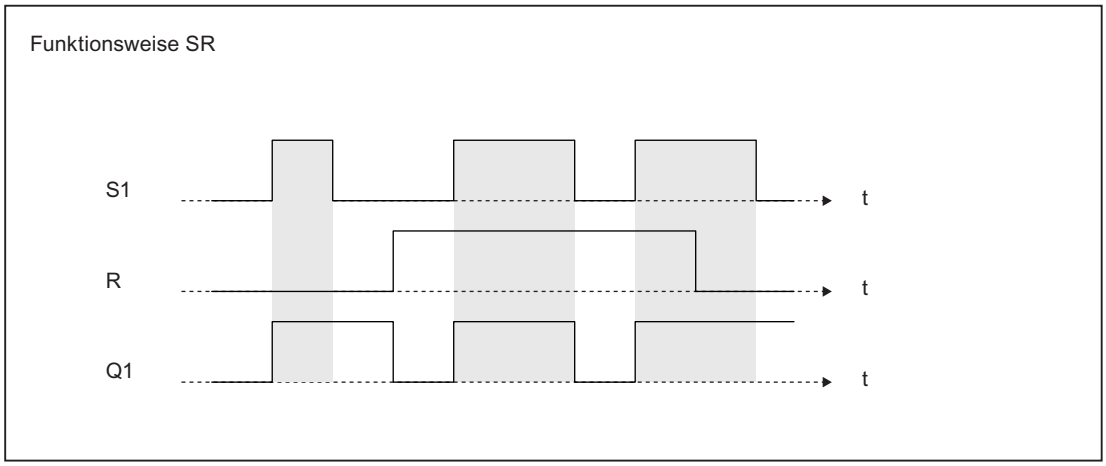


Bild 9-1 Funktionsweise des bistabilen Funktionsbausteins SR

Tabelle 9- 2 Programmcode des bistabilen Funktionsbausteins SR

```
FUNCTION_BLOCK SR

    VAR_INPUT
        S1,R        : BOOL;
    END_VAR

    VAR_OUTPUT
        Q1        : BOOL;
    END_VAR

    Q1 := S1 OR (NOT R AND Q1);

END_FUNCTION_BLOCK
```

Tabelle 9- 3 Aufrufparameter für SR

Bezeichner	Parameter	Datentyp	Beschreibung
S1	Eingang	BOOL	Setzen
R	Eingang	BOOL	Rücksetzen
Q1	Ausgang	BOOL	Signalzustand

Bistabiler Funktionsbaustein RS (vorrangig rücksetzen)

Am Ausgang Q1 kann der gespeicherte Wert abgegriffen werden. Der Ausgang Q1 ist 0, wenn R1 gleich 1 sind. Wenn R1 und S gleich 0 sind, ändert sich der Aus gang Q1 nicht.

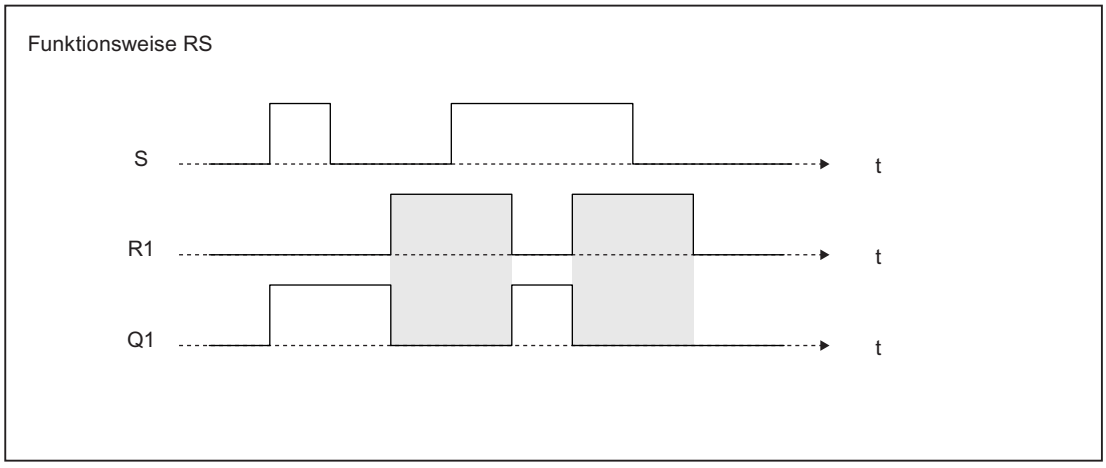


Bild 9-2 Funktionsweise des bistabilen Funktionsbausteins RS

Tabelle 9- 4 Programmcode des bistabilen Funktionsbausteins RS

```
FUNCTION_BLOCK RS

    VAR_INPUT
        R1,S        :   BOOL;
    END_VAR

    VAR_OUTPUT
        Q1          :    BOOL;
    END_VAR

    Q1 := NOT R1 AND (S OR Q1);

END_FUNCTION_BLOCK
```

Tabelle 9- 5 Aufrufparameter für RS

Bezeichner	Parameter	Datentyp	Beschreibung
S	Eingang	BOOL	Setzen
R1	Eingang	BOOL	Rücksetzen
Q1	Ausgang	BOOL	Signalzustand

9.3 Flankenerkennung

Mit dem Systemfunktionsbaustein R_TRIG können Sie eine steigende, mit F_TRIG eine fallende Flanke erkennen. Sie können dies nutzen, um beispielsweise eine Ablaufkette von eigenen Funktionsbausteinen aufzubauen.

Erkennung der steigenden Flanke R_TRIG

Wenn am Eingang eine steigende Flanke (R_TRIG, Rising Trigger), d. h. ein Zustandswechsel von 0 auf 1 vorliegt, wird am Ausgang für die Dauer einer Zykluszeit eine 1 angelegt.

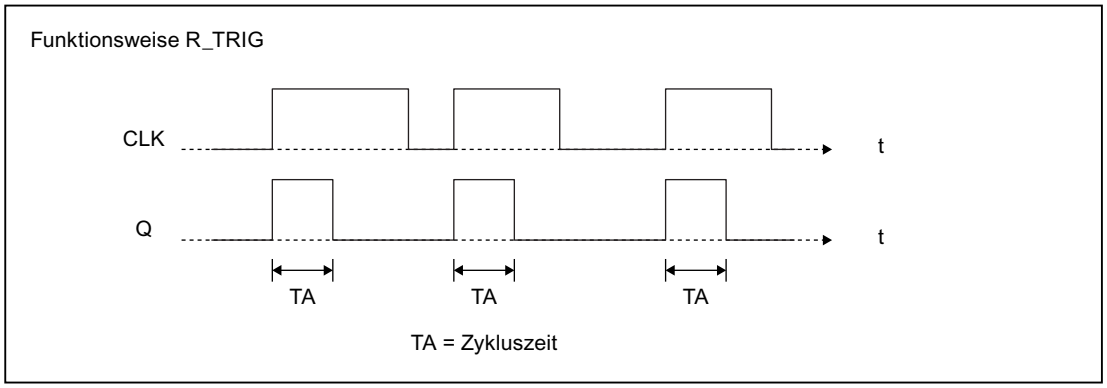


Bild 9-3 Funktionsweise des Funktionsbausteins R_TRIG (steigende Flanke)

Tabelle 9- 6 Programmcode des Funktionsbausteins R_TRIG (steigende Flanke)

```
FUNCTION_BLOCK R_TRIG

    VAR_INPUT
        CLK    : BOOL;
    END_VAR

    VAR_OUTPUT
        Q      : BOOL;
    END_VAR

    VAR
        M      : BOOL := 0;
    END_VAR

    Q := CLK AND NOT M;
    M := CLK;

END_FUNCTION_BLOCK
```

Tabelle 9- 7 Aufrufparameter für R_TRIG

Bezeichner	Parameter	Datentyp	Beschreibung
CLK	Eingang	BOOL	Eingang für Flankenerkennung
Q	Ausgang	BOOL	Zustand der Flanke

Erkennung der fallenden Flanke F_TRIG

Wenn am Eingang eine fallende Flanke (F_TRIG, Falling Trigger), d. h. ein Zustandswechsel von 1 auf 0 vorliegt, wird am Ausgang für die Dauer einer Zykluszeit eine 1 angelegt.

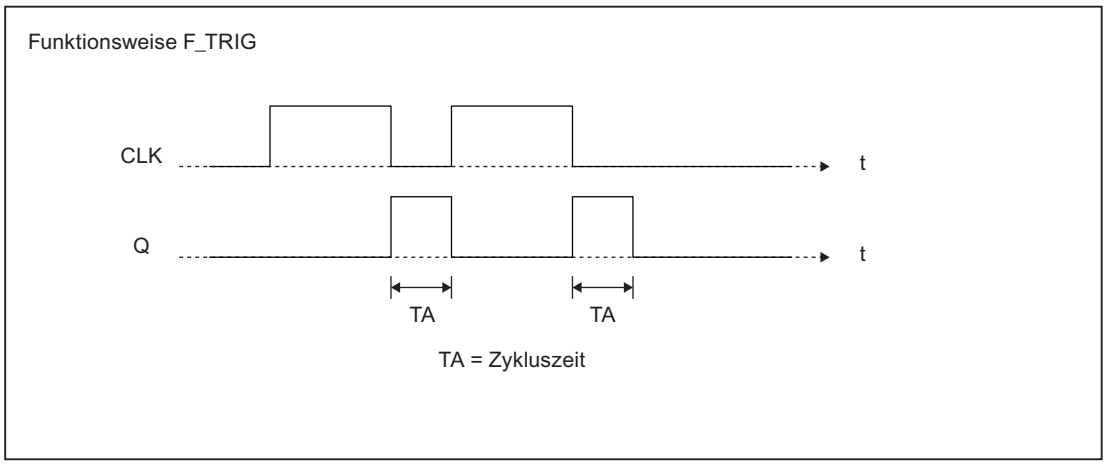


Bild 9-4 Funktionsweise des Funktionsbausteins F_TRIG (fallende Flanke)

Tabelle 9- 8 Programmcode des Funktionsbausteins F_TRIG (fallende Flanke)

```
FUNCTION_BLOCK F_TRIG

    VAR_INPUT
        CLK    : BOOL;
    END_VAR

    VAR_OUTPUT
        Q    : BOOL;
    END_VAR

    VAR
        M    : BOOL    := 1;
    END_VAR

    Q := NOT CLK AND NOT M;
    M := NOT CLK;

END_FUNCTION_BLOCK
```

Tabelle 9- 9 Aufrufparameter für F_TRIG

Bezeichner	Parameter	Datentyp	Beschreibung
CLK	Eingang	BOOL	Eingang für Flankenerkennung
Q	Ausgang	BOOL	Zustand der Flanke

9.4 Zähler

9.4.1 Allgemeines zu Zählern

ST stellt eine Reihe von Systemfunktionsbausteinen für Zähler zur Verfügung, die Sie in Ihrem ST-Programm verwenden können.

9.4.2 Aufwärtszähler CTU

Mit dem Zähler CTU können Sie Aufwärts-Zähloperationen durchführen:

- Wenn beim Aufruf des FB der Eingang R = TRUE ist, wird der Ausgang CV auf 0 zurückgesetzt.
- Wenn beim Aufruf des FB der Eingang CU von FALSE auf TRUE (von 0 auf 1) wechselt (= positive Flanke), wird der Ausgang CV um 1 hochgezählt.
- Der Ausgang Q gibt an, ob CV größer oder gleich dem Vergleichswert PV ist.

Die Parameter CV und PV haben den Datentyp INT, der Zählerstand kann deshalb maximal 32_767 (= 16#7FFF) betragen.

Prototyp der Anwenderschnittstelle

Beispielcode

```
FUNCTION_BLOCK CTU
 VAR_INPUT
 CU : BOOL; // Zählen
 R : BOOL; // Rücksetzen
 PV : INT; // Vergleichswert
 END_VAR
 VAR_OUTPUT
 Q : BOOL; // Status
 CV : INT; // Zählerstand
 END_VAR
 // ... (Code)
END_FUNCTION_BLOCK
```

Eingangsparameter

CU

Datentyp: BOOL

Aufwärtszählen, wenn Wert von FALSE auf TRUE wechselt (positive Flanke)

R

Datentyp: BOOL

TRUE: Rücksetzen des Zählers auf 0

PV

Datentyp: INT

Vergleichswert

Ausgangsparameter

Q

Datentyp: BOOL

Status des Zählers (CV >= PV)

CV

Datentyp: INT

Zählerstand

9.4.3 Aufwärtszähler CTU_DINT

Die Funktionsweise entspricht dem Aufwärtszähler CTU (siehe Kapitel 7.3.1) mit Ausnahme:

Die Parameter CV und PV haben den Datentyp DINT, der Zählerstand kann deshalb maximal 2_147_483_647 (= 16#7FFF_FFFF) betragen.

Tabelle 9- 10 Parameter für CTU_DINT

Bezeichner	Parameter	Datentyp	Beschreibung
CU	Eingang	BOOL	Aufwärtszählen, wenn Wert von FALSE auf TRUE wechselt (positive Flanke)
R	Eingang	BOOL	Rücksetzen des Zählers auf 0
PV	Eingang	DINT	Vergleichswert
Q	Ausgang	BOOL	Status des Zählers (CV >= PV)
CV	Ausgang	DINT	Zählerstand

9.4.4 Aufwärtszähler CTU_UDINT

Die Funktionsweise entspricht dem Aufwärtszähler CTU mit Ausnahme:

Die Parameter CV und PV haben den Datentyp UDINT, der Zählerstand kann deshalb maximal 4_294_967_295 (=16#FFFF_FFFF) betragen.

Tabelle 9- 11 Parameter für CTU_UDINT

Bezeichner	Parameter	Datentyp	Beschreibung
CU	Eingang	BOOL	Aufwärtszählen, wenn Wert von FALSE auf TRUE wechselt (positive Flanke)
R	Eingang	BOOL	Rücksetzen des Zählers auf 0
PV	Eingang	UDINT	Vergleichswert
Q	Ausgang	BOOL	Status des Zählers (CV >= PV)
CV	Ausgang	UDINT	Zählerstand

Siehe auch

Aufwärtszähler CTU (Seite 496)

9.4.5 Abwärtszähler CTD

Mit dem Zähler CTD können Sie Abwärts-Zähloperationen durchführen.

- Wenn beim Aufruf des FB der Eingang LD = TRUE ist, wird der Ausgang CV auf den Anfangswert PV zurückgesetzt.
- Wenn beim Aufruf des FB der Eingang CD von FALSE auf TRUE (von 0 auf 1) wechselt (= positive Flanke), wird der Ausgang CV um 1 abwärts gezählt.
- Der Ausgang Q gibt an, ob CV kleiner oder gleich 0 ist.

Die Parameter CV und PV haben den Datentyp INT, der Zählerstand kann deshalb minimal –32_768 (= 16#8000) betragen.

Tabelle 9- 12 Aufrufparameter für CTD

Bezeichner	Parameter	Datentyp	Beschreibung
CD	Eingang	BOOL	Abwärtszählen, wenn Wert von FALSE auf TRUE wechselt (positive Flanke)
LD	Eingang	BOOL	Rücksetzen des Zählers auf Anfangswert
PV	Eingang	INT	Anfangswert des Zählers
Q	Ausgang	BOOL	Status des Zählers (CV <= 0)
CV	Ausgang	INT	Zählerstand

9.4.6 Abwärtszähler CTD_DINT

Die Funktionsweise entspricht dem Aufwärtszähler CTD mit Ausnahme:

Die Parameter CV und PV haben den Datentyp DINT, der Zählerstand kann deshalb minimal –2_147_483_648 (= 16#8000_0000) betragen.

Tabelle 9- 13 Aufrufparameter für CTD_DINT

Bezeichner	Parameter	Datentyp	Beschreibung
CD	Eingang	BOOL	Abwärtszählen, wenn Wert von FALSE auf TRUE wechselt (positive Flanke)
LD	Eingang	BOOL	Rücksetzen des Zählers auf Anfangswert
PV	Eingang	DINT	Anfangswert des Zählers
Q	Ausgang	BOOL	Status des Zählers (CV <= 0)
CV	Ausgang	DINT	Zählerstand

9.4.7 Abwärtszähler CTD_UDINT

Die Funktionsweise entspricht dem Aufwärtszähler CTD mit Ausnahme:

Die Parameter CV und PV haben den Datentyp UDINT, der Zählerstand kann deshalb minimal 0 betragen.

Tabelle 9- 14 Aufrufparameter für CTD_UDINT

Bezeichner	Parameter	Datentyp	Beschreibung
CD	Eingang	BOOL	Abwärtszählen, wenn Wert von FALSE auf TRUE wechselt (positive Flanke)
LD	Eingang	BOOL	Rücksetzen des Zählers auf Anfangswert
PV	Eingang	UDINT	Anfangswert des Zählers
Q	Ausgang	BOOL	Status des Zählers (CV = 0)
CV	Ausgang	UDINT	Zählerstand

9.4.8 Auf-/Abwärtszähler CTUD

Mit dem Zähler CTUD können Sie sowohl Auf- als auch Abwärts-Zähloperationen durchführen.

- Rücksetzen der Zählvariable CV:
  - Wenn beim Aufruf des FB der Eingang R = TRUE ist, wird der Ausgang CV auf 0 zurückgesetzt.
  - Wenn beim Aufruf des FB der Eingang LD = TRUE ist, wird der Ausgang CV auf den Anfangswert PV zurückgesetzt.
- Zählen:
  - Wenn beim Aufruf des FB der Eingang CU von FALSE auf TRUE (von 0 auf 1) wechselt (= positive Flanke), wird der Ausgang CV um 1 hochgezählt.
  - Wenn beim Aufruf des FB der Eingang CD von FALSE auf TRUE (von 0 auf 1) wechselt (= positive Flanke), wird der Ausgang CV um 1 abwärts gezählt.
- Zählerstatus QU bzw. QD:
  - Der Ausgang QU gibt an, ob CV größer oder gleich dem Vergleichswert PV ist.
  - Der Ausgang QD gibt an, ob CV kleiner oder gleich 0 ist.

Tabelle 9- 15 Parameter für CTUD

Bezeichner	Parameter	Datentyp	Beschreibung
CU	Eingang	BOOL	Aufwärtszählen, wenn Wert von FALSE auf TRUE wechselt (positive Flanke)
CD	Eingang	BOOL	Abwärtszählen, wenn Wert von FALSE auf TRUE wechselt (positive Flanke)
R	Eingang	BOOL	Rücksetzen des Zählers auf 0 (Aufwärtszähler)
LD	Eingang	BOOL	Rücksetzen des Zählers auf Anfangswert PV (Abwärtszähler)
PV	Eingang	INT	Vergleichswert (für Aufwärtszähler) Anfangswert (für Abwärtszähler)
QU	Ausgang	BOOL	Status als Aufwärtszähler (CV >= PV)
QD	Ausgang	BOOL	Status als Abwärtszähler (CV <= 0)
CV	Ausgang	INT	Zählerstand

9.4.9 Auf-/Abwärtszähler CTUD_DINT

Die Funktionsweise entspricht dem Aufwärtszähler CTUD mit Ausnahme:

Die Parameter CV und PV haben den Datentyp DINT.

Tabelle 9- 16 Parameter für CTU_DINT

Bezeichner	Parameter	Datentyp	Beschreibung
CU	Eingang	BOOL	Aufwärtszählen, wenn Wert von FALSE auf TRUE wechselt (positive Flanke)
CD	Eingang	BOOL	Abwärtszählen, wenn Wert von FALSE auf TRUE wechselt (positive Flanke)
R	Eingang	BOOL	Rücksetzen des Zählers auf 0 (Abwärtszähler)
LD	Eingang	BOOL	Rücksetzen des Zählers auf Anfangswert PV (Abwärtszähler)
PV	Eingang	DINT	Vergleichswert (für Aufwärtszähler) Anfangswert (für Abwärtszähler)
QU	Ausgang	BOOL	Status als Aufwärtszähler (CV >= PV)
QD	Ausgang	BOOL	Status als Abwärtszähler (CV <= 0)
CV	Ausgang	DINT	Zählerstand

9.4.10 Auf-/Abwärtszähler CTUD_UDINT

Die Funktionsweise entspricht dem Aufwärtszähler CTUD mit Ausnahme:

Die Parameter CV und PV haben den Datentyp UDINT.

Tabelle 9- 17 Parameter für CTU_UDINT

Bezeichner	Parameter	Datentyp	Beschreibung
CU	Eingang	BOOL	Aufwärtszählen, wenn Wert von FALSE auf TRUE wechselt (positive Flanke)
CD	Eingang	BOOL	Abwärtszählen, wenn Wert von FALSE auf TRUE wechselt (positive Flanke)
R	Eingang	BOOL	Rücksetzen des Zählers auf 0 (Aufwärtszähler)
LD	Eingang	BOOL	Rücksetzen des Zählers auf Anfangswert PV (Abwärtszähler)
PV	Eingang	UDINT	Vergleichswert (für Aufwärtszähler) Anfangswert (für Abwärtszähler)
QU	Ausgang	BOOL	Status als Aufwärtszähler (CV >= PV)
QD	Ausgang	BOOL	Status als Abwärtszähler (CV = 0)
CV	Ausgang	UDINT	Zählerstand

9.5 Zeitgeber

Zeiten sind Elemente in Ihrem Programm, die zeitgesteuerte Abläufe ausführen und überwachen. ST stellt eine Reihe von Systemfunktionsbausteinen zur Verfügung, auf die Sie mit ST zugreifen können. Mit Zeitoperationen können Sie in Ihrem Programm:

- Wartezeiten einstellen,
- Überwachungszeiten ermöglichen,
- Impulse erzeugen,
- Zeiten messen.

Puls TP

Mit einem Signalzustandswechsel von 0 auf 1 am Eingang IN wird die Zeit ET gestartet. Der Ausgang Q bleibt so lange auf 1, bis die abgelaufene Zeit ET gleich dem programmierten Zeitwert PT ist. Solange die Zeit ET läuft, hat der Eingang IN keine Wirkung.

Das folgende Bild veranschaulicht die Funktionsweise des Zeitgebers Puls TP.

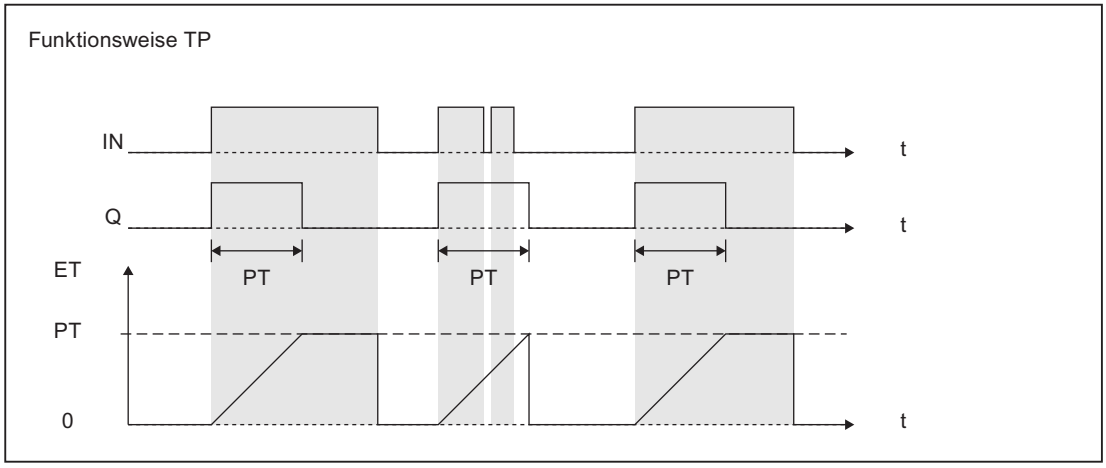


Bild 9-5 Funktionsweise des Pulses TP

Die folgende Tabelle zeigt Ihnen die Aufrufparameter:

Tabelle 9- 18 Aufrufparameter für TP

Bezeichner	Parameter	Datentyp	Beschreibung
IN	Eingang	BOOL	Starteingang
PT	Eingang	TIME	Zeitdauer des Impulses.
Q	Ausgang	BOOL	Status der Zeit
ET	Ausgang	TIME	abgelaufene Zeit

Einschaltverzögerung TON

Mit dem Signalzustandswechsel von 0 auf 1 am Eingang IN wird die Zeit ET gestartet. Das Ausgangssignal Q wechselt nur von 0 auf 1, wenn die Zeit ET = PT abgelaufen ist und das Eingangssignal IN noch immer 1 beträgt. D. h. der Ausgang Q wird verzögert eingeschaltet. Eingangssignale, deren Zeitdauer kürzer als die der programmierten Zeit PT sind, erscheinen am Ausgang nicht.

Das folgende Bild veranschaulicht die Funktionsweise des Zeitgebers Einschaltverzögerung TON.

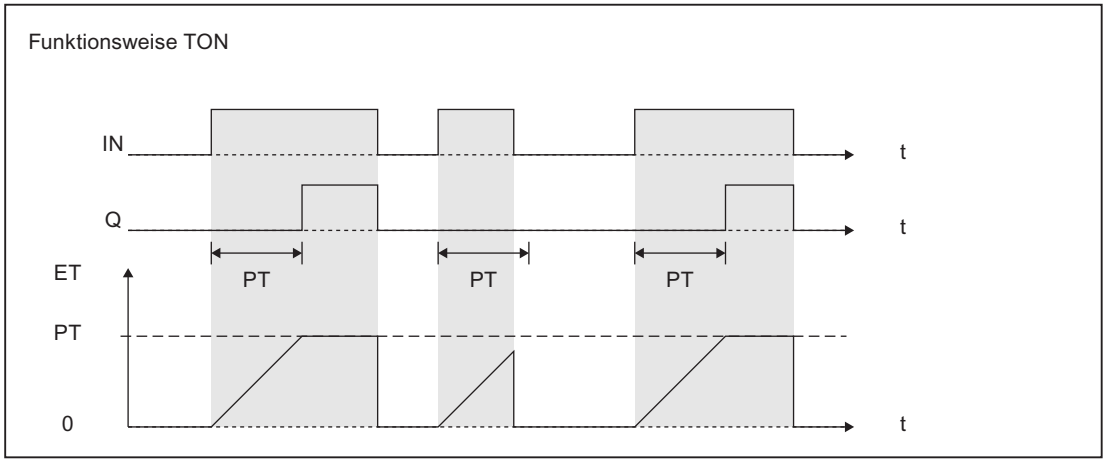


Bild 9-6 Funktionsweise der Einschaltverzögerung TON

Die folgende Tabelle zeigt Ihnen die Aufrufparameter:

Tabelle 9- 19 Aufrufparameter für TON

Bezeichner	Parameter	Datentyp	Beschreibung
IN	Eingang	BOOL	Starteingang
PT	Eingang	TIME	Zeitdauer, um welche die steigende Flanke am Eingang IN verzögert wird.
Q	Ausgang	BOOL	Status der Zeit
ET	Ausgang	TIME	abgelaufene Zeit

```
VAR
    Mytimeout : TON;
End_VAR
    Mytimeout(pt := T#2s, IN := TRUE);
IF (mytimeout.q) THEN
    ; //Zeit abgelaufen
END_IF;
```

Ausschaltverzögerung TOF

Bei einem Signalzustandswechsel von 0 nach 1 am Starteingang IN erscheint am Ausgang Q der Zustand 1. Wechselt der Zustand am Starteingang IN von 1 nach 0, wird die Zeit ET gestartet. Erfolgt vor dem Ablauf der Zeit ET ein Wechsel am Eingang IN von 0 auf 1, wird der Zeitablauf zurückgesetzt. Ein erneuter Start erfolgt wieder mit dem Zustandswechsel des Eingangs IN von 1 auf 0. Erst nach Ablauf der Zeitdauer ET = PT nimmt der Ausgang Q den Signalzustand 0 an. Der Ausgang wird also verzögert abgeschaltet.

Das folgende Bild veranschaulicht die Funktionsweise des Zeitgebers Ausschaltverzögerung TOF.

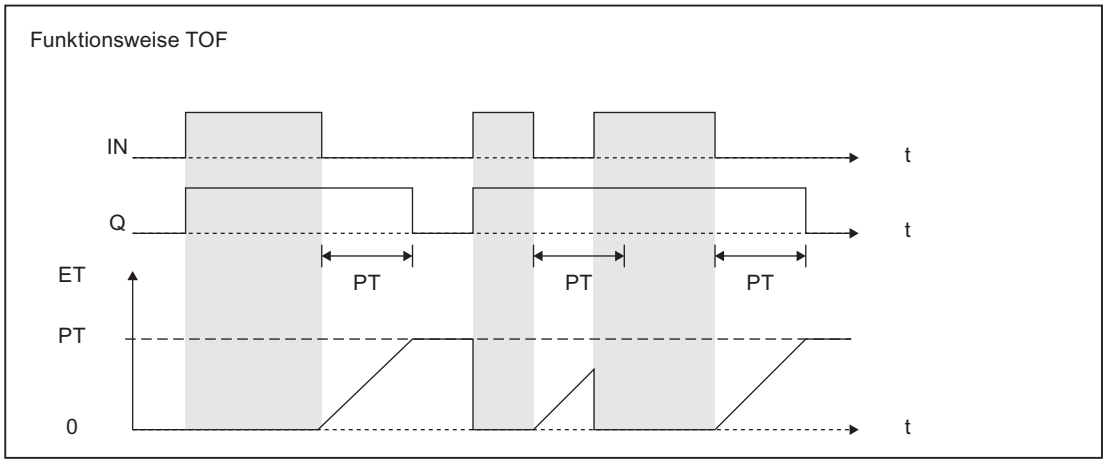


Bild 9-7 Funktionsweise der Ausschaltverzögerung TOF

Die folgende Tabelle zeigt Ihnen die Aufrufparameter:

Tabelle 9- 20 Aufrufparameter für TOF

Bezeichner	Parameter	Datentyp	Beschreibung
IN	Eingang	BOOL	Starteingang
PT	Eingang	TIME	Zeitdauer, um welche die fallende Flanke am Eingang IN verzögert wird.
Q	Ausgang	BOOL	Status der Zeit
ET	Ausgang	TIME	abgelaufene Zeit

Echtzeituhr RTC

Die positive Flanke an SET stellt die Echtzeituhr auf den Wert von PDT; PDT wird in CDT übernommen. Wird READ auf TRUE gesetzt, wird die aktuelle Systemzeit ausgelesen und steht am Ausgang CDT zur Verfügung.

Tabelle 9- 21 Aufrufparameter für RTC

Bezeichner	Parameter	Datentyp	Beschreibung
SET	Eingang	BOOL	Zeit setzen, standardmäßig FALSE
READ	Eingang	BOOL	Zeit lesen, standardmäßig FALSE
PDT	Eingang	DT	Wert, auf den die Echtzeituhr gesetzt werden soll, standardmäßig DT#0001-01-01-0:0:0. Ist der Wert kleiner als der Voreinstellungswert der Echtzeituhr des SIMOTION Geräts, wird die Echtzeituhr auf ihren Voreinstellungswert gesetzt (z. B. bei C2xx: DT#1994-01-01-00:00:00).
CDT	Ausgang	DT	Aktuelle Systemzeit

Die Granularität beim Setzen der Echtzeituhr beträgt 1ms, die Genauigkeit ist vom Lageregler-Takt bestimmt.

9.6 Zerlegen von Bitstring-Datentypen

9.6.1 Allgemeines zum Zerlegen von Bitstring Datentypen

Nachstehende Funktionsbausteine ermöglichen das Zerlegen einer Variablen eines Bitstring-Datentyps in mehrere Variablen eines untergeordneten Datentyps.

Die Umkehrfunktionen sind als Funktionen realisiert (siehe **Zusammenfassen von Bitstring-Datentypen**)

Siehe auch

Allgemeines zum Zusammenfassen von Bitstring Datentypen (Seite 389)

9.6.2 Funktionsbaustein _BYTE_TO_8BOOL

Diese Funktion zerlegt eine Variable vom Datentyp BYTE in 8 Variablen vom Datentyp BOOL.

Prototyp der Anwenderschnittstelle

```
FUNCTION_BLOCK _BYTE_TO_8BOOL
    VAR_INPUT
        bytein : BYTE;
    END_VAR
    VAR_OUTPUT
        bit0,                   // niederstwertiges Bit
        bit1, bit2, bit3, bit4, bit5, bit6,
        bit7 : BOOL;            // höchstwertiges Bit
    END_VAR
    // ... (Code)
END_FUNCTION_BLOCK
```

Eingangsparameter

bytein

Datentyp: BYTE

Variable vom Datentyp BYTE, die in 8 Variablen vom Datentyp BOOL zerlegt werden soll.

Ausgangsparameter

bit0

...

bit7

Datentyp: BOOL

8 Variablen mit den einzelnen Bits des Eingangsparameters

bit0: niederwertigstes Bit

...

bit7: höchstwertiges Bit

9.6.3 Funktionsbaustein _WORD_TO_2BYTE

Diese Funktion zerlegt eine Variable vom Datentyp WORD in 2 Variablen vom Datentyp BYTE.

Prototyp der Anwenderschnittstelle

```
FUNCTION_BLOCK _WORD_TO_2BYTE
    VAR_INPUT
        wordin : WORD;
    END_VAR
    VAR_OUTPUT
        byte0,                  // niederwertiges Byte
        byte1 : BYTE;           // höherwertiges Byte
    END_VAR
    // ... (Code)
END_FUNCTION_BLOCK
```

Eingangsparameter

wordin

Datentyp: WORD

Variable vom Datentyp WORD, die in 2 Variablen vom Datentyp BYTE zerlegt werden soll.

Ausgangsparameter

byte0 (optional)
byte1 (optional)

Datentyp: BYTE

2 Variablen mit den einzelnen Bytes des Eingangsparameters

byte0: niederwertiges Byte
byte1: höherwertiges Byte

9.6.4 Funktionsbaustein _DWORD_TO_2WORD

Diese Funktion zerlegt eine Variable vom Datentyp DWORD in 2 Variablen vom Datentyp WORD.

Prototyp der Anwenderschnittstelle

```
FUNCTION_BLOCK _DWORD_TO_2WORD
    VAR_INPUT
        dwordin : DWORD;
    END_VAR
    VAR_OUTPUT
        word0,                // niederwertiges Wort
        word1 : WORD;         // höherwertiges Wort
    END_VAR
    // ... (Code)
END_FUNCTION_BLOCK
```

Eingangsparameter

dwordin

Datentyp: DWORD

Variable vom Datentyp DWORD, die in 2 Variablen vom Datentyp WORD zerlegt werden soll.

Ausgangsparameter

word0
word1

Datentyp: WORD

2 Variablen mit den einzelnen Bytes des Eingangsparameters

word0: niederwertiges Wort
word1: höherwertiges Wort

9.6.5 Funktionsbaustein _DWORD_TO_4BYTE

Diese Funktion zerlegt eine Variable vom Datentyp DWORD in 4 Variablen vom Datentyp BYTE.

Prototyp der Anwenderschnittstelle

```
FUNCTION_BLOCK _DWORD_TO_4BYTE
    VAR_INPUT
        dwordin : DWORD;
    END_VAR
    VAR_OUTPUT
        byte0,               // niederstwertiges Byte
        byte1, byte2,
        byte3 : BYTE;        // höchstwertiges Byte
    END_VAR
    // ... (Code)
END_FUNCTION_BLOCK
```

Eingangsparameter

in		
	Datentyp:	DWORD
	Variable vom Datentyp DWORD, die in 4 Variablen vom Datentyp BYTE zerlegt werden soll.	

Ausgangsparameter

byte0 **byte1** **byte2** **byte3**		
	Datentyp:	BYTE
	Vorbelegung	BYTE#0
	2 Variablen mit den einzelnen Bytes des Eingangsparameters byte0: niederstwertiges Byte ... byte3: höchstwertiges Byte	

9.7 Emulation von SIMATIC S7 Befehlen

9.7.1 Allgemeines

Nachfolgende Funktionsbausteine sind schnittstellenkompatibel mit den Befehlen für Zähler und Zeiten der SIMATIC S7; siehe Referenzhandbuch SIMATIC Anweisungsliste (AWL) für S7-300/400.

Hinweis

Verwenden Sie in der Regel die Funktionsbausteine nach IEC 61131-3 (**Zähler** und **Zeitgeber**).

Siehe auch

Allgemeines zu Zählern (Seite 496)

Zeitgeber (Seite 502)

9.7.2 Funktionsbaustein _S7_COUNTER

Prototyp der Anwenderschnittstelle

```
FUNCTION_BLOCK _S7_COUNTER
    VAR_INPUT
        CU : BOOL;
        CD : BOOL;
        PV : INT;
        S : BOOL;
        R : BOOL;
    END_VAR
    VAR_OUTPUT
        Q : BOOL;
        CV : INT;
    END_VAR;
    // ... (Code)
END_FUNCTION_BLOCK
```

Eingangsparameter

CU

Datentyp: BOOL

Aufwärtszählen (mit Flanke)

CD

Datentyp: BOOL

Abwärtszählen (mit Flanke)

PV

Datentyp: INT

Vorwahlwert

S

Datentyp: BOOL

Setzen Vorwahlwert (mit Flanke)

R

Datentyp: BOOL

Rücksetzen Zähler (statisch)

Ausgangsparameter

Q
Datentyp: BOOL
Status Zähler

CV
Datentyp: INT
Zählerstand

9.7.3 Funktionsbaustein _S7_TIMER

Dieser Funktionsbaustein dient zur Emulation der Zeitfunktionen der SIMATIC S7.

Prototyp der Anwenderschnittstelle

```
FUNCTION_BLOCK _S7_TIMER
    VAR_INPUT
        T_Type : WORD;
        PR : BOOL;
        S : BOOL;
        R : BOOL;
        TV : TIME;
        TV_BCD : WORD;
    END_VAR
    VAR_OUTPUT
        Q : BOOL;
        BI : TIME;
    END_VAR
    // ... (Code)
END_FUNCTION_BLOCK
```

Eingangsparameter

T_TYPE

Datentyp: WORD

Auszuführende Funktion des S7-Timers

TYPE_S7T_SI_ = 16#0001 SI SP
TYPE_S7T_SV_ = 16#0002 SV SEE
TYPE_S7T_SE_ = 16#0004 SE SD
TYPE_S7T_SS_ = 16#0008 SS SS
TYPE_S7T_SA_ = 16#0010 SA SF
TYPE_S7T_BCD_IN = 16#0020 SA SF
TYPE_S7T_SI_ = 16#0040 R FR
TYPE_S7T_SI_ = 16#0080 L LC (U; UN; X ...)

FR

Datentyp: BOOL

Freigabe

S

Datentyp: BOOL

Setze Vorwahlwert

R

Datentyp: BOOL

Rücksetzen Timer

TV

Datentyp: TIME

Vorwahlwert (als Datentyp TIME)

TV_BCD

Datentyp: WORD

Vorwahlwert in S7-Codierung

Ausgangsparameter

Q1

Datentyp: BOOL

Status Timer

BI

Datentyp: TIME

Aktuelle Zeit

SIMOTION Speicherkonzept (im Zielgerät) 10

10.1 Überblick über die Speicher im Zielgerät

Speicherarten

SIMOTION besitzt ein mehrstufiges Speicherverwaltungskonzept. Im Zielgerät wird zwischen dem DRAM (RAM-Disk und RAM) und dem SRAM/NVRAM unterschieden. Die Daten (TOs, Units und TPs) im DRAM gehen nach dem Ausschalten verloren, während das SRAM/NVRAM kleinere Datenmengen remanent speichern kann. Zusätzlich ist als ROM-Speicher ein Wechselmedium (CF Card oder Memory Card) vorhanden.

RAM-Disk

Die RAM-Disk ist ein virtueller Speicher, der analog einer Festplatte angesteuert wird, auf den aber viel schneller zugegriffen werden kann. Für die RAM-Disk ist ein fester Speicherbereich im DRAM des Zielgeräts reserviert. In der RAM-Disk befinden sich nach dem Download die Hardware- und Gerätekonfiguration, Technologiepakete (TP), Projektierungsdaten der Technologieobjekte und die Programm Units. Bei **RAM nach ROM kopieren** wird der Inhalt der RAM-Disk in den ROM (auf Karte) geschrieben. Bei einem Upload werden die Daten über die RAM-Disk in das PG geladen. Die Auslastung der RAM-Disk können Sie sich in der Gerätediagnose unter Systemauslastung (nur ONLINE) anzeigen lassen.

RAM

Im RAM-Speicher werden Anwenderdaten (User-RAM) und Systemdaten (System-RAM) gespeichert. Der User-RAM enthält den TO Aktual-Speicher, den TO Next-Speicher der Technologieobjekte (näheres siehe Systemvariablen und Konfigurationsdaten online ändern) und dient daneben zur Speicherung von Technologiepaketen, Units (Variablen und Code). Beim Hochlauf oder Download werden die Daten aus der RAM-Disk in den RAM geladen.

Der User-RAM, der die TP und Units enthält, kann zur Veranschaulichung in einen Datenspeicher (z. B. Variablen) und Codespeicher (compilierte Anwenderprogramme) unterteilt werden. Retain-Variablen werden remanent (netzausfallsicher) im SRAM/NVRAM gespeichert.

Im System-RAM sind der Kernel, Kerneldaten und weitere Einstellungen wie Kommunikationseinstellungen, Takteinstellungen sowie der Inhalt des Diagnosepuffers gespeichert.

Die Auslastung des RAM-Speichers können Sie sich in der Gerätediagnose unter Systemauslastung (nur ONLINE) anzeigen lassen.

ROM (CF Card oder Memory Card)

Daten werden auf der Speicherkarte (ROM) netzausfallsicher gespeichert. Sie müssen dazu den Befehl **RAM nach ROM kopieren** ausführen. Vom ROM-Speicher werden die zuletzt durch RAM nach ROM kopieren gesicherten Daten beim Hochlauf in die RAM-Disk und von dort in den RAM geladen. Die Auslastung der CF/Memory Card können Sie sich in der Gerätediagnose unter Systemauslastung (nur ONLINE) anzeigen lassen.

SRAM/NVRAM (Remanente Daten - Netz-Aus feste Daten)

Im SRAM/NVRAM werden Daten remanent und netzausfallsicher gespeichert. Die Daten werden beim Hochlauf aus dem SRAM/NVRAM in den RAM-Speicher geladen. Die Belegung des SRAM/NVRAM können Sie sich in der Gerätediagnose unter Systemauslastung (Remanente Daten, nur ONLINE) anzeigen lassen.

Folgende Netz-Aus-feste Daten gibt es in einem SIMOTION Gerät:

Netz-Aus-feste Daten	Inhalt
Kernel-Daten	Letzter Betriebszustand
	IP-Parameter (IP-Adresse, Subnetzmaske, Router-Adresse)
	DP-Parameter (PROFIBUS DP-Adresse, Baudrate)
	Diagnosepuffer
Retain Variablen	Variablen im Interface- oder Implementationsabschnitt einer Unit, die mit VAR_GLOBAL RETAIN deklariert sind
	Geräteglobale Variablen mit dem Attribut "RETAIN" gesetzt
Retain-TO	Absolutgeberoffset
DCC-Bausteine	SAV-Bausteine und anwenderdefinierte Bausteine mit Retain-Verhalten.

Mit der Systemfunktion "_savePersistentMemoryData" kann über das Anwenderprogramm der Inhalt der Netz-Aus-festen Daten auf die Speicherkarte gesichert werden. Somit sind die Retain-Variablen sowie die Stellung des Absolutwertgebers z. B. für einen Ersatzteilfall gesichert (näheres siehe Listenhandbuch *Systemfunktionen/-variablen Geräte*).

Nachfolgende Grafik zeigt den prinzipiellen Aufbau der Speicher im Zielgerät

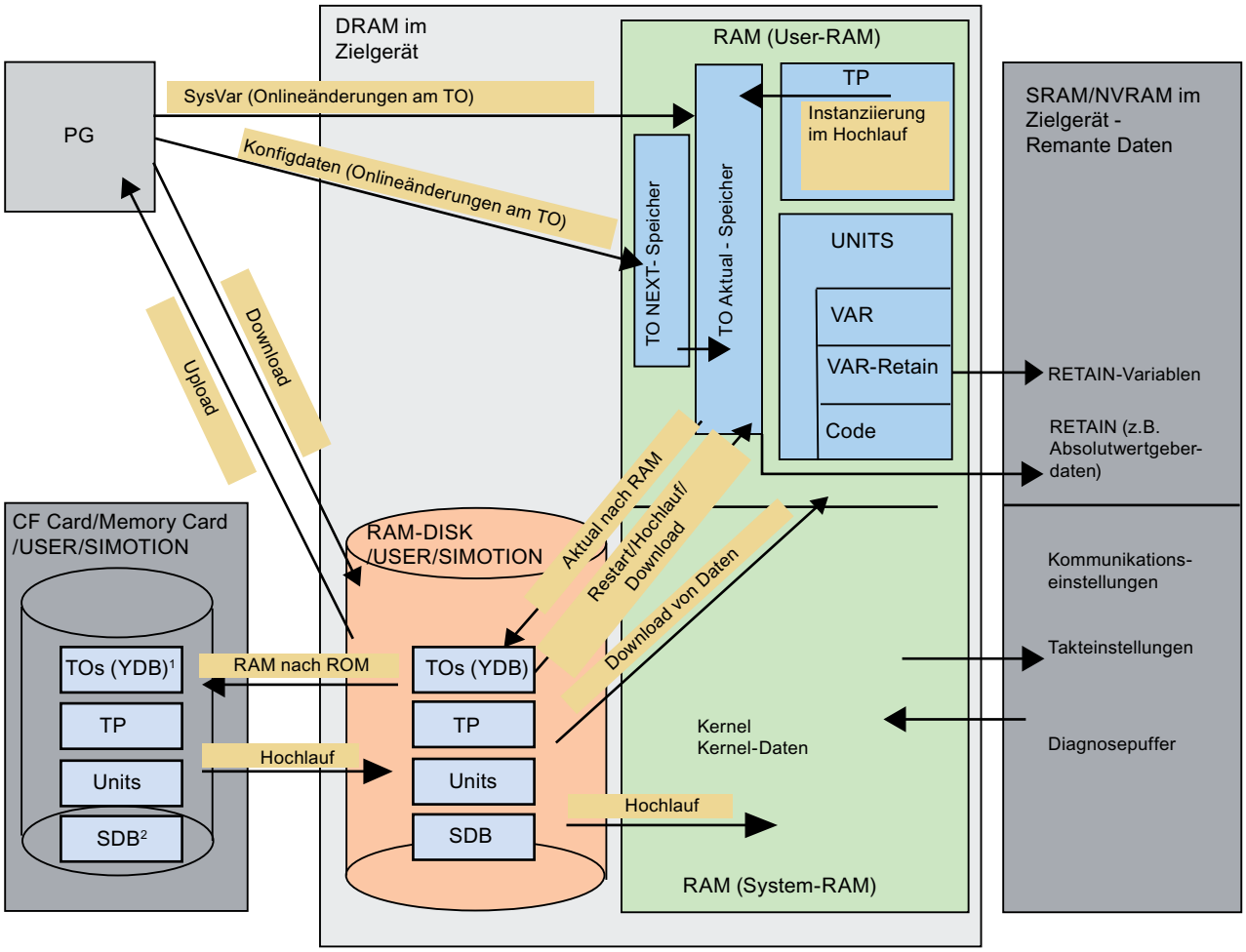


Bild 10-1 Speicherkonzept, prinzipielle Darstellung

[1] YDBs enthalten die Konfigurationsdaten von TOs wie z. B. Systemvariablen, Konfigdaten oder Alarme

[2] SDB sind Systemdatenbausteine

Siehe auch

Download im RUN (Seite 535)

Übersicht zum Download von Daten (Seite 523)

Speicherzugriffe (Seite 518)

RAM nach ROM kopieren (Seite 555)

10.2 Speicherzugriffe

Hochlauf des Zielgerätes

Bei einem Hochlauf werden die Daten der TOs, Technologiepakete, Units und weitere Daten aus dem ROM-Speicher über die RAM-Disk in den USER-RAM übernommen. Die TO-Daten werden in den TO Aktual Speicher geladen, die Technologiepakete (TP) und Units in den USER-RAM. Dabei werden die einzelnen TOs instanziiert und mit den entsprechenden Werten vorbelegt. Kommunikationseinstellungen, Takteinstellungen und Diagnosepufferinhalt (falls vorhanden) sowie die Retain-Daten werden aus dem SRAM/NVRAM in den System-RAM übernommen.

Hinweis

IP- und DP-Parameter in den Netz-Aus-festen Daten

Befindet sich eine Projektierung auf der Speicherkarte, dann werden die IP- und DP-Parameter im Hochlauf von der Speicherkarte geladen und vom SIMOTION Gerät verwendet. Über die darin definierten Adressen erfolgt das Online gehen. Im Hochlauf werden die IP und DP-Parameter auf der Speicherkarte auch in die Netz-Aus-festen Daten geschrieben. Wird anschließend mit einer CF Card hochgelaufen, die keine Projektierung enthält, dann bleiben die IP- und DP-Parameter in den Netz-Aus-festen Daten erhalten und werden vom SIMOTION Gerät verwendet. Somit kann auf ein SIMOTION Gerät auch weiterhin online gegangen werden, wenn zumindest einmal eine Projektierung mit dem SIMOTION SCOUT geladen wurde, bzw. das SIMOTION Gerät mit einer Speicherkarte hochgelaufen ist, auf der sich eine Projektierung befand.

Netz-Ein und Urlöschen

Daten (Konfigurationsdaten, Defaultwerte der Systemvariablen) werden vom ROM (Speicherkarte) in die RAM-Disk geladen und anschließend in den TO Aktual Speicher. Beim Urlöschen werden vor dem Hochlauf das RAM und die remanenten Daten im SRAM/NVRAM, bis auf die Kommunikationsprojektierung (Baudraten, Netzwerkadressen etc.) gelöscht. Ebenfalls werden die Unit-Daten beim Urlöschen gelöscht und dann beim Hochladen wieder aus dem ROM geladen.

Restart eines Technologieobjektes

Daten des TO (Konfigurationsdaten, Defaultwerte der Systemvariablen, Kurvenscheiben) werden von der RAM-Disk in den TO Aktual Speicher geladen.

STOP - RUN - Übergang

Der Wechsel vom Betriebszustand STOP in RUN bewirkt keinen Speicherzugriff auf einen der verschiedenen Speicher.

Download in das Zielgerät

Bei einem Download vom PG in das Zielgerät werden die Daten zuerst in die RAM-Disk und von dort in den RAM Speicher geschrieben, mit optionaler Initialisierung der Daten. Siehe auch Übersicht zum Download von Daten (Seite 523) und Getrennte Initialisierung von Quell- und TO-Daten bei einem Download (Seite 534).

Mit dem Download können zusätzliche Daten (z. B. Quellen) auf das Zielgerät geladen werden.

Diese Daten werden benötigt für

- Online-Objektvergleich (z. B. zusätzliche Eigenschaften)
- Diverse Detailvergleiche (z. B. ST-Quellenvergleich)
- Funktion "Laden ins PG" (Komplett-Upload ins Offline Projekt)
- Synchronisation mit Online-Objekten

Um Quellen oder Zusatzdaten eines Projektes ins PG laden zu können, muss im Projekt unter **Extras >Einstellungen > Download > Zusatzdaten auf dem Zielgerät ablegen** eingestellt gewesen sein.

Systemvariablen und Konfigurationsdaten online ändern

Ändern Sie Werte von Systemvariablen, werden die Werte von Systemvariablen sofort im TO Aktual Speicher wirksam. Bei Konfigurationsdaten werden die neuen Werte zunächst im Next Speicher gepuffert. Sofort wirksame Konfigurationsdaten werden automatisch in den TO Aktual Speicher übernommen. Konfigurationsdaten, die erst nach einem RESTART am Technologieobjekt (Systemvariable *restartactivation* auf den Wert ACTIVATE_RESTART setzen) wirksam werden, werden erst nach dem RESTART in den TO Aktual Speicher geschrieben.

Zur Sicherung der online geänderten Konfigurationsdaten ins Offline-Projekt müssen Sie mit dem Menübefehl **Zielsystem > Aktual nach RAM kopieren** zunächst den Inhalt des TO Aktual Speichers in die RAM-Disk übernehmen.

Danach ist die Projektierung im SCOUT nicht mehr konsistent zur Projektierung im Zielgerät, da die Konsistenzprüfung auf die Daten der RAM-Disk erfolgt. Aus der RAM-Disk lesen Sie die Daten dann durch den Menübefehl **Zielsystem > Laden > Laden in PG** (nur die Konfigurationsdaten) und stellen damit wieder einen konsistenten Zustand sicher. Zum netzaussicheren Speichern der Projektierung auf CF Card oder Memory Card führen Sie den Menübefehl **Zielsystem > RAM noch ROM kopieren** durch. Über diesen Menübefehl kann auch implizit die Funktion **Aktual nach ROM kopieren** und **Laden ins PG** ausgeführt werden.

Ab V4.2 werden Konfigurationsdaten im Hochlauf mit Parameterwerten des SINAMICS-Antriebs gefüllt (Adaption). Für nähere Informationen siehe Adaption unter Symbolische Zuordnung - Einführung (Seite 79). Bei einem "Aktual nach RAM kopieren" oder "RAM nach ROM kopieren" werden adaptierte Werte erkannt und automatisch auch ein "Laden ins PG" vorselektiert.

Hinweis

Die Werte von Systemvariablen werden beim **Aktual nach RAM kopieren** nicht in den RAM Speicher übernommen. Damit ist kein "Sichern auf Speicherkarte (RAM nach ROM kopieren)" bzw. "Sichern im Engineeringprojekt (Laden in PG)" möglich.

Damit Werte von Systemvariablen auch im Engineeringprojekt und auf Speicherkarte gesichert werden, muss der Wert von Systemvariablen OFFLINE geändert und dann per Download ins Zielgerät geladen und gesichert werden.

Wie sich TOs, Variablen und Programme bei einem Download im RUN verhalten, siehe Download im RUN (Seite 535)

Netz-Aus-feste Daten sichern

Zum Sichern der Netz-Aus-festen Daten auf die Speicherkarte haben Sie folgende Möglichkeiten:

- im Anwenderprogramm:

  mit der Systemfunktion "_savePersistentMemoryData" kann das Anwenderprogramm den Inhalt der Netz-Aus-festen Daten auf die Speicherkarte sichern. Somit sind die Retain-Variablen sowie die Stellung des Absolutwertgebers für den Ersatzteilfall gesichert.

- per Schalter/Taster (Service-Wahlschalter oder DIAG-Taster der SIMOTION D) oder IT DIAG.

Netz-Aus-feste Daten zurücksichern

Mit _savePersistentMemoryData auf CF Card gesicherte Daten werden bei folgenden Szenarien zurückgesichert:

1. nach einem Urlöschen
2. per Schalterstellung bei SIMOTION D
3. Verlust des SRAM-Inhaltes wegen leerer Batterie. In diesem Fall wird der Inhalt des gesamten SRAMs aus der Datei restauriert.

Wenn nach einem Baugruppentausch Retain-Daten auf der Baugruppe vorhanden sind, werden diese genommen und bleiben gültig sofern der TO-Name oder Unit-Name übereinstimmt. Ansonsten sind diese Daten ungültig.

Das bedeutet, dass nach einem Baugruppentausch, wenn das Projekt nicht zu den gespeicherten Retain-Daten passt, der Punkt 1 oder 2. ausgeführt werden sollte.

Die gesicherten Daten in der Datei "USER/SIMOTION/PMEMORY.XML" werden während

des Hochlaufs der Steuerung in das SRAM/NVRAM zurück kopiert.

Die Systemvariable device.persistentDataPowerMonitoring.persistentDataState zeigt nach einem Hochlauf an, in welchem Zustand sich die Netz-Aus-feste Daten befinden. Der Zustand FROM_RAM signalisiert, dass die Netz-Aus-festen Daten nicht verloren gegangen sind, sondern aus dem SRAM/NVRAM geladen wurden.

Tabelle 10- 1 Zustand der Netz-Aus-festen Daten nach Hochlauf (Systemvariable persistentDataState)

Zustand	Bedeutung
FROM_RAM (1)	Netz-Aus-feste Daten im SIMOTION Gerät werden verwendet
FROM_FILE (2)	Netz-Aus-feste Daten aus der Sicherungsdatei wiederhergestellt
FROM_BACKUP (3)	Netz-Aus-feste Daten aus der Backup-Sicherungsdatei wiederhergestellt
INVALID (4)	Daten in den Netz-Aus-festen Daten und in Sicherungsdatei / Backup-Sicherungsdatei ungültig bzw. nicht vorhanden/gelöscht. Das SIMOTION Gerät hat die Werkseinstellungen in die Netz-Aus-festen Daten kopiert und ist mit diesen hochgefahren.

Daten in das Zielgerät laden 11

11.1 Übersicht zum Download von Daten

Beschreibung

Die Projektdaten, die Sie mit dem SCOUT erstellt haben, müssen Sie über einen Download ins Zielsystem laden. Das Zielsystem kann mehrere CPUs (SIMOTION-Steuerungen oder Antriebe) enthalten. Die Projektdaten enthalten dabei die Programme (ST, MCC, KOP/FUP und DCC), die Sie erstellt und compiliert haben, die Hardware Konfiguration und die Technologiepakete, die Sie angelegt und parametriert haben.

Sie müssen ein Projekt erneut ins Zielsystem laden, wenn Sie:

- Programme erstellt oder geändert haben.
- die Definitionen globaler Variablen oder symbolischer I/O-Variablen geändert haben.
- Änderungen im Ablaufsystem vorgenommen haben.
- Konfigurationen der Technologieobjekte geändert haben.

Voraussetzungen für einen Download

Folgende Voraussetzungen müssen erfüllt sein, damit das Projekt ins Zielsystem geladen werden kann:

- Alle Anschlüsse (Kabel) sind gesteckt und die Schnittstellen konfiguriert
- Alle Programmquellen sind übersetzt (Menü Projekt > Speichern und Änderungen übersetzen), oder es erfolgt das Übersetzen der Programmquellen vor dem Download automatisch
- Die Konsistenz des Projekts ist ggf. überprüft (Menü Projekt > Konsistenz prüfen)
- System befindet sich im Status ONLINE (Menü Projekt > Mit Zielsystem verbinden)

Bei Auftreten eines Fehlers wird die Übertragung der Daten zum Zielsystem abgebrochen. Im Register **Ausgabe Zielsystem**, in dem der Ablauf der Übertragung und eventuelle Fehler protokolliert werden, finden Sie einen Hinweis auf den Fehler.

Hinweis

Tritt während des Downloads ein Spannungsausfall am SIMOTION Gerät auf, erscheint eine Meldung im Diagnosepuffer: Anlaufsperre gesetzt, nachdem Sie in den Betriebszustand RUN geschaltet haben. Gehen Sie erneut online und führen Sie den Download nochmals durch.

In früheren Versionen kann es zu unterschiedlichen Fehlern führen, die durch das System nicht abgefangen werden können. Es können unerklärliche Fehlermeldungen erscheinen. Tritt dieser Fall auf:
Gehen Sie erneut online und führen Sie den Download nochmals durch.

Download-Umfang

Sie können wahlweise das ganze Projekt hinunter laden oder auch einen Download spezifisch für ein einzelnes Gerät durchführen:

- Projekt ins Zielsystem laden (alle Zielgeräte) (Seite 527)
- CPU/Antriebsgerät ins Zielgerät laden (Seite 530)

Betriebszustand

Sie können einen Download im Betriebszustand STOP, aber auch im Betriebszustand RUN durchführen.

Weitere Möglichkeiten für einen Download im OFFLINE Modus

Die Projektdaten im SCOUT können Sie auch über zusätzliche Funktionen im OFFLINE-Modus auf die Speicherkarte übertragen:

- Download direkt auf Speicherkarte oder Festplatte, siehe Download direkt auf Speicherkarte oder Festplatte (Seite 549)
- Geräte Update Tool, siehe Online Hilfe zu Geräte hochrüsten.

11.2 Speichern und übersetzen

Vor einem Download muss das Projekt zuerst gespeichert und übersetzt werden. SIMOTION SCOUT unterscheidet dabei zwischen drei Befehlen im Projekt-Hauptmenü, die Unterschiede beim Speichern und Übersetzen bewirken:

- Speichern; das Projekt wird auf der Festplatte gespeichert.
- Speichern und Änderungen übersetzen
- Speichern und alles neu übersetzen

Daneben gibt es noch im Kontextmenü einer Quelle folgenden Befehl:

- Übernehmen und übersetzen (einer einzelnen Quelle)

Hinweis

Befinden sich im Projekt Quellen, die nicht fehlerfrei übersetzbar sind, oder die noch nicht übersetzt werden sollen, können diese zwischenzeitlich in einer Bibliothek abgelegt werden, die nicht verwendet wird.

Speichern

Das Projekt wird auf der Festplatte gespeichert. Die Änderungen werden im Projekt übernommen. Es wird kein weiterer Vorgang wie Übersetzen oder Konsistenz prüfen mit dem Projekt angestoßen.

Speichern und Änderungen übersetzen

Bei diesem Befehl wird das gesamte Projekt nach Änderungen durchsucht. Wird eine Quelle, die geändert wurde oder keine Übersetzungsergebnisse hat, gefunden, wird genau diese und die verlinkten Quellen (z. B. bei FB Aufruf) übersetzt und gespeichert. Es erfolgt also nur eine Übersetzung der Änderungen. Verwenden Sie diesen Befehl für die tägliche Arbeit innerhalb einer SCOUT-Version.

Speichern und alles neu übersetzen

Mit diesem Befehl werden alle Quellen des gesamten Projektes neu übersetzt.

Der Befehl **Speichern und alles neu übersetzen** eignet sich dafür, wenn ganz sicher alle alten Daten aus älteren SCOUT Versionen entfernt und durch neue Übersetzungsergebnisse ersetzt werden sollen. Er setzt sich aus folgenden Schritten zusammen:

- Projektweites Löschen aller Übersetzungsergebnisse
- Neukompilierung aller Objekte

Verwenden Sie diesen Befehl, wenn Sie ein Projekt gezielt von einer älteren SCOUT-Version auf eine Neuere umstellen möchten. Sie übernehmen damit alle Fehlerbehebungen und Optimierungen im neuen SCOUT. Damit werden aber die Übersetzungsergebnisse im SCOUT und im SIMOTION-RT inkonsistent. Der Projektnavigator und der Objektvergleich zeigen im ONLINE-Modus die Objekte dann als "inkonsistent" an. Um das Projekt debuggen zu können, müssen Sie das komplette Projekt ins Zielsystem laden.

Vergleich der Übersetzungsergebnisse im Projekt und im SIMOTION-RT

Bis SIMOTION SCOUT V4.0 wurden die Übersetzungsergebnisse anhand ihres Zeitstempels verglichen. Ab V4.0 vergleicht der SCOUT die Übersetzungsergebnisse über eine Hash-Funktion. Eine Hash-Funktion dient zum Vergleichen von größeren Datenmengen anhand des Hash-Codes. Nur wenn der Hash-Code der Übersetzungsergebnisse im Projekt und im SIMOTION-RT gleich ist, werden sie als "konsistent" angezeigt. Ein "gleicher" Hash-Code wird nur dann geliefert, wenn Code und Compiler (SCOUT-Version beachten) gleich sind.

Informationen zu Fehlern

Detailinformationen zu Fehlern beim Übernehmen und Übersetzen werden bei der Übersetzung von einzelnen Quellen ausgegeben (OFFLINE **Ausgabe übersetzen/prüfen** und ONLINE **Ausgabe Zielsystem**). Durch Doppelklick auf die Fehlermeldung im Detailfenster springt der Cursor auf die Fehlerstelle in der Quelle. Beheben Sie die Fehler, indem Sie z. B.:

- die Konfiguration der Technologieobjekte ändern
- die Programme ändern

Wiederholen Sie die Schritte so oft, bis keine Fehler angezeigt werden.

Hinweis

Sie haben die Möglichkeit die Detailinformationen zu Fehlern auch beim projektweiten Übersetzen auszugeben (bei Speichern und alles (neu) übersetzen). Führen Sie dazu **Extras>Einstellungen> Compiler>Bei "Speichern und Änderungen übersetzen" sämtliche Meldungen anzeigen** aus.

11.3 Konsistenzprüfung durchführen

Projekt auf Konsistenz prüfen und übersetzen

Bevor Sie das Projekt ins Zielsystem laden, müssen die Programmquellen fehlerfrei übersetzt und die Konsistenz gewährleistet sein. Dabei werden z. B. IO-Adressen oder TO-Konfigurationen überprüft.

1. Wählen Sie Menü **Projekt > Speichern und Änderungen übersetzen**.

SIMOTION SCOUT übersetzt alle geänderten Quellen.
Weitere Informationen zum Projekt speichern und übersetzen finden Sie in Speichern und übersetzen (Seite 525) .

2. Wählen Sie Menü **Projekt > Konsistenz prüfen**.

   SIMOTION SCOUT überprüft, ob z. B. alle Technologieobjekte konfiguriert sind und die Quellen fehlerfrei übersetzt sind. Vor einem Download kann automatisch eine Konsistenzprüfung durchgeführt werden, wenn die entsprechende Option unter **Extras > Download > Konsistenz vor dem Laden prüfen** aktiviert ist.

   Während des Downloads wird im Zielsystem immer eine Konsistenzprüfung durchgeführt. Sofern Download-Fehler auftreten, werden Sie durch Einträge in verschiedenen Ausgabefenstern (ONLINE im Fenster **Ausgabe Zielsystem**) der Detailanzeige auf die mögliche Ursache hingewiesen.

Hinweis

Wenn Sie einen Download mit einem älteren Projekt durchführen, kann es ggf. vorkommen, dass die Quellen nicht mehr konsistent sind. Führen Sie dann **Projekt > Speichern und alles neu übersetzen** durch.

11.4 Projekt ins Zielsystem laden (alle Zielgeräte)

Bei einem Projekt-Download wird das gesamte Projekt in alle Geräte, auf denen man online ist, geladen (am grünen Steckersymbolen im Projektnavigator erkennbar). Damit laden Sie Daten auf alle Geräte, die im Dialog **Zielgeräteauswahl** angewählt sind. Dazu zählen auch alle Geräte, die unterhalb eines SIMOTION-Geräts angeordnet sind (z. B. SINAMICS Int, externe PROFIBUS/PROFINET Slaves). Gehen Sie vor dem Download selektiv bei einem Gerät offline, dann wird auf dieses Gerät kein Download durchgeführt. Ändern Sie die Auswahl im Dialog **Zielgeräteauswahl, nachdem Sie online gegangen sind**, hat das keine Auswirkungen auf den Download.

Hinweis

Einen Projekt-Download können Sie nur im Betriebszustand STOP und für alle Zielgeräte, mit denen Sie ONLINE sind, durchführen. Auf Antriebe, die nicht im SCOUT projektiert werden können, wie z. B. MASTERDRIVE oder 611U, können Sie keinen Download durchführen. Die Projektdaten werden in alle ONLINE verbundenen Geräte und deren untergeordnete Antriebsgeräte (sofern diese im Dialog **Zielgeräteauswahl** selektiert sind) geladen. Dies kann nur im Betriebszustand STOP erfolgen.

Vorgehensweise

1. Klicken Sie auf die Schaltfläche **Projekt ins Zielsystem laden**. Der Dialog **Laden ins Zielsystem** wird eingeblendet.

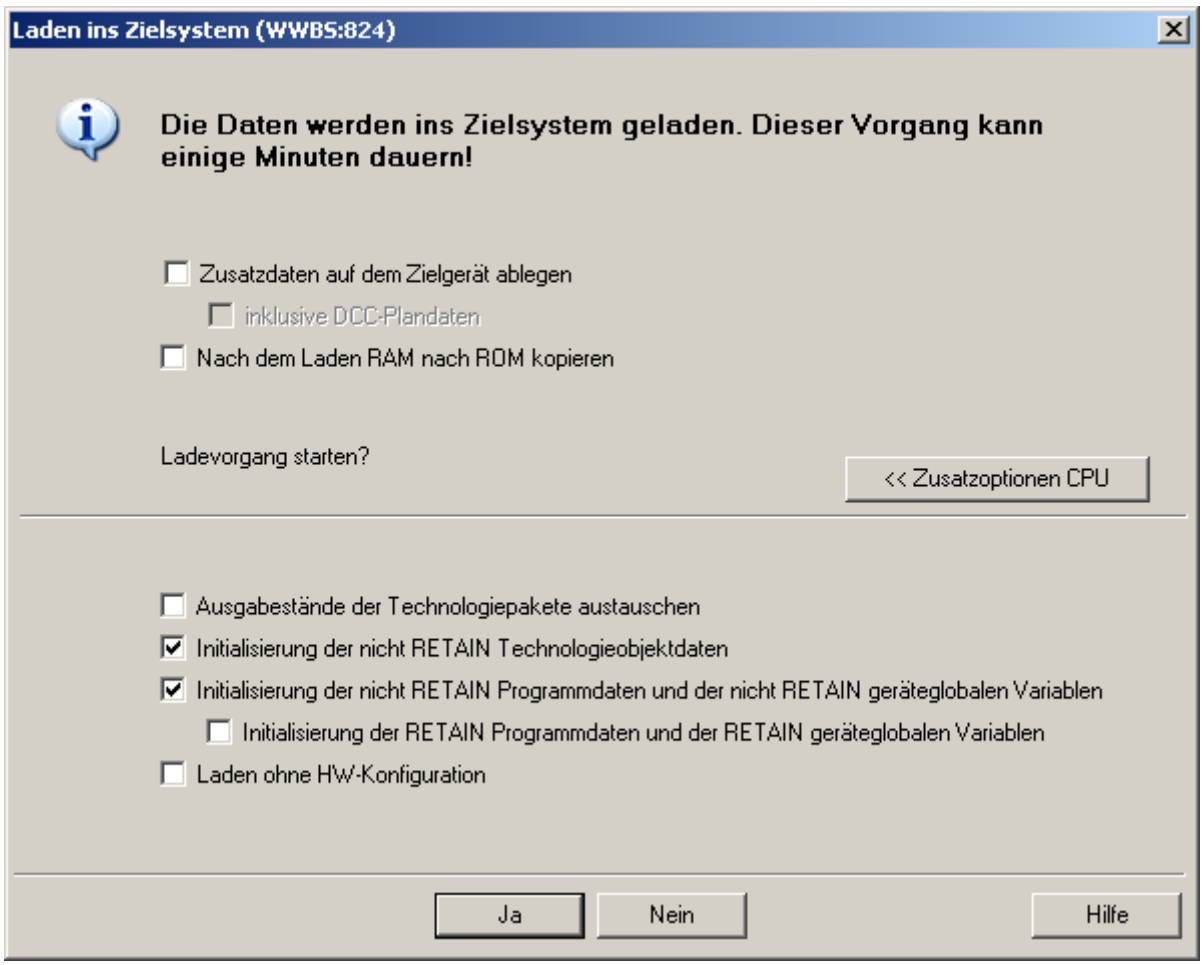


Bild 11-1 Laden ins Zielsystem

Folgende Optionen können Sie in Abhängigkeit vom Gerät auswählen:

- **Zusatzdaten auf dem Zielgerät ablegen**; mit dieser Option können Sie Zusatzdaten, z. B. Programmquellen auf dem Zielgerät ablegen.
  - Inklusive DCC-Plandaten
    Bei DCC können zusätzlich zu den Zusatzdaten, die den Upload eines funktionsfähigen Plans ermöglichen auch noch die grafischen Plandaten abgelegt werden, sodass dieser Plan im kompletten Quelltext vorliegt und damit auch weiterbearbeitet werden kann.
- Nach dem **Laden RAM nach ROM kopieren**; führt RAM nach ROM nach dem Download für alle Zielgeräte durch.

- Klicken Sie auf **Zusatzoptionen CPU** um folgende Optionen einzublenden:
  - **Ausgabestände der Technologiepakete austauschen**; lädt die Technologiepakete in das Zielsystem, die im Dialog Technologiepakete auswählen ausgewählt sind. Siehe auch Ausgewählte Technologiepakete laden (Seite 533).
  - Initialisierung der nicht RETAIN Technologieobjektdaten (siehe Getrennte Initialisierung von Quell- und TO-Daten bei einem Download (Seite 534))
  - Initialisierung der nicht RETAIN Programmdaten und der nicht RETAIN Geräteglobalen Variablen
    Initialisierung der RETAIN Programmdaten und der RETAIN Geräteglobalen Variablen (siehe Getrennte Initialisierung von Quell- und TO-Daten bei einem Download (Seite 534))
  - Laden ohne HW-Konfiguration (siehe Download ohne HW Konfig Information (Seite 545)
- Klicken Sie auf **Ja**, um den Download anzustoßen.

Die Zusatzoptionen sind auch dann wirksam, wenn sie nicht sichtbar sind. Es werden dann die Einstellungen verwendet, mit denen der letzte Download durchgeführt wurde.

Bei Antriebsgeräten können Sie nur **Laden RAM nach ROM kopieren** auswählen.

Mögliche Fehler bei Projekten bis V4.1.1

Wenn ein Projekt V4.0 ins Zielsystem geladen wird, kann es dazu führen, dass beim Laden der I/O-Konfiguration folgender Fehler auftritt: "IOM Konfiguration inkonsistent".

Der Fehler kann nach bestimmten Änderungen am I/O Container (Editieren mit SCOUT V4.1.1.x oder älter) auftreten. Dabei wird der Änderungsstempel nicht korrekt berechnet bzw. gespeichert. Es können dadurch zwei unterschiedliche Fehlerbilder entstehen:

- Einmal kann es zum Codeumsetzungsfehler mit Hinweis auf I/O beim Laden einer Quelle kommen. Hier ist eine Abhilfe dadurch möglich, dass die betreffende Quelle neu übersetzt wird (entweder einzeln oder gleich Speichern und neu Übersetzen des gesamten Geräts).
- Ein anderes Fehlerbild mit gleicher Ursache ist, dass beim Laden der I/O-Konfiguration eine unberechtigte Überschneidung von I/O Variablen seitens des Zielgeräts gemeldet wird. Hier kann nur durch Speichern und neu Übersetzen des gesamten betroffenen Geräts eine Abhilfe erreicht werden.

Siehe auch

Speichern und übersetzen (Seite 525)

11.5 CPU/Antriebsgerät ins Zielgerät laden

Beschreibung

Neben einem Projektdownload können Sie selektiv Daten in jedes CPU/Antriebsgerät laden. Die Standard-Vorgehensweise ist der Download im Betriebszustand STOP. Unter bestimmten Bedingungen können Sie aber auch einen Download im Betriebszustand RUN durchführen, Näheres siehe unter Download im RUN (Seite 535).

Vorgehensweise

1. Wählen Sie zuerst das Gerät aus, in das Sie die Daten laden möchten, indem Sie es im Projektnavigator auswählen (Gerät muss ONLINE sein).
2. Klicken Sie auf die Schaltfläche **CPU/Antriebsgerät ins Zielsystem laden**
   oder
   Klicken Sie im Projektnavigator mit der rechten Maustaste auf das Gerät und führen im Kontextmenü **Zielgerät > Laden ins Zielgerät** aus.
3. Der Dialog **Laden ins Zielgerät** wird eingeblendet. Der Inhalt ist davon abhängig, für welches Gerät Sie Daten ins Zielgerät laden möchten.

CPU ins Zielgerät laden

Ein Download kann für die gesamte CPU (ohne Antriebsgerät), aber auch getrennt für einzelne, zusammen gehörende Download-Einheiten durchgeführt werden.

- CPU ohne Antriebsgerät laden
- Alle Technologieobjekte der CPU laden
- Alle Programme der CPU laden

Folgende Optionen können Sie auswählen:

- Zusatzdaten auf dem Zielgerät ablegen; mit dieser Option können Sie Zusatzdaten, z. B. Programmquellen auf dem Zielgerät ablegen.
  - Inklusive DCC-Plandaten
    Bei DCC können zusätzlich zu den Zusatzdaten, die den Upload eines funktionsfähigen Plans ermöglichen auch noch die grafischen Plandaten abgelegt werden, sodass dieser Plan im kompletten Quelltext vorliegt und damit auch weiterbearbeitet werden kann.
- Nach dem Laden RAM nach ROM kopieren; führt RAM nach ROM nach dem Download auf dem selektierten Gerät durch.
- Download im RUN durchführen; führt einen Download im Betriebszustand RUN durch (siehe unten).

  Klicken Sie auf **Zusatzoptionen CPU** um folgende Optionen einzublenden:

- Ausgabestände der Technologiepakete austauschen; lädt die Technologiepakete in das Zielsystem, die im Dialog Technologiepakete auswählen ausgewählt sind.
- Initialisierung der nicht RETAIN Technologieobjektdaten - wirkt nur im Betriebszustand STOP (siehe Getrennte Initialisierung von Quell- und TO-Daten bei einem Download (Seite 534))

- Initialisierung der nicht RETAIN Programmdaten und der nicht RETAIN Geräteglobalen Variablen - wirkt nur im Betriebszustand STOP.
  - Initialisierung der RETAIN Programmdaten und der RETAIN Geräteglobalen Variablen (siehe Getrennte Initialisierung von Quell- und TO-Daten bei einem Download (Seite 534))
- Laden ohne HW-Konfiguration (siehe Download ohne HW Konfig Information (Seite 545)

Die globale Voreinstellung können Sie unter **Extras > Einstellungen > Download** bzw. **CPU-Download** vornehmen.

Bild 11-2 Laden in CPU/Antriebsgerät

Antrieb ins Zielsystem laden

Für Antriebe können Sie nur die Antriebsdaten (Parametrierung etc.) in das Antriebgerät hinunter laden.

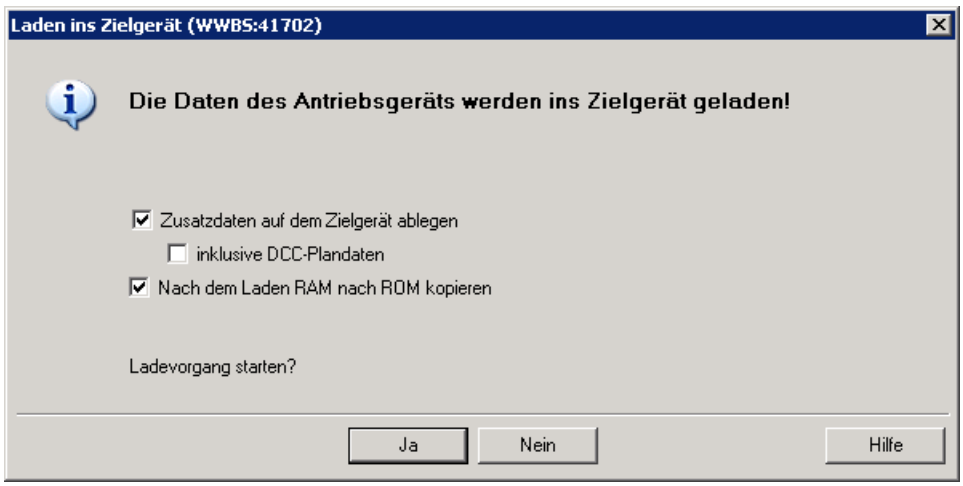


Bild 11-3 Laden in Antriebgerät

Standardmäßig erfolgt der Download im STOP. Die CPU kann nach einem erfolgreichen Download wieder in den vorhergehenden Zustand gebracht werden.

Die Option "Nach dem Laden RAM nach ROM kopieren" führt RAM nach ROM nach dem Download auf dem selektierten Gerät durch.

Siehe auch

Download im RUN von geänderten Quellen (Seite 535)

Download von geänderten Technologieobjekten (Seite 546)

11.6 Ausgewählte Technologiepakete laden

Beschreibung

Ab der Version V4.2 können Sie den Ausgabestand von Technologiepaketen innerhalb einer Version auswählen (auch Service Packs und Hotfixe). Durch **Ausgabestände der TP austauschen** werden die Technologiepakete in das Zielsystem geladen. Damit können Sie neuere oder ältere Stände des Technologiepaketes im Zielsystem überschreiben.

So laden Sie einen bestimmten Ausgabestand eines Technologiepaketes in das Zielsystem

1. Klicken Sie mit der rechten Maustaste im Projektnavigator auf ein SIMOTION Gerät und führen Sie **Technologiepakete auswählen** aus.

   Der Dialog wird geöffnet.
2. Wählen Sie das Technologiepaket aus, das Sie aktualisieren möchten.
3. Wählen Sie unter TP-Version und Ausgabestand die Version bzw. den Ausgabestand aus. Ab V4.2 ist hier auch die Auswahl von Hotfix-Ständen möglich.
4. Klicken Sie auf **OK**, um den Dialog zu schließen und die Einstellungen zu übernehmen.
5. Gehen Sie mit dem Gerät online und klicken Sie auf **Projekt ins Zielsystem laden**.

   Der Dialog wird geöffnet.
6. Wählen Sie **Ausgabestände der Technologiepakete austauschen** aus und klicken Sie auf **OK**.
7. Das Technologiepaket wird in das Zielsystem hinuntergeladen, auch wenn bereits ein Technologiepaket existiert.

Randbedingungen für das Austauschen von Technologiepaketen

- Wenn im Dialog **Technologiepakete auswählen** kein Technologiepaket ausgewählt ist, wird das neueste Technologiepaket der entsprechenden Firmware-Version in das Zielsystem geladen.
- Wenn Sie einen Upload durchführen, werden die Versionen der im Zielsystem vorhandenen Technologiepakete mit hoch geladen, am Gerät eingetragen und im Dialog **Technologiepakete auswählen** angezeigt.
- Ausgabestände von Technologiepaketen werden mit exportiert und importiert.
- Falls der eingestellte Ausgabestand nicht auf dem PG/PC vorhanden ist, wird eine Fehlermeldung ausgegeben.
- Das Laden der Technologiepakete wird in der Detailanzeige des SCOUT angezeigt.

Siehe auch

Projekt ins Zielsystem laden (alle Zielgeräte) (Seite 527)

11.7 Getrennte Initialisierung von Quell- und TO-Daten bei einem Download

Beschreibung

Ab V4.1.2 können Sie nicht-RETAIN und RETAIN-Daten von Programmen (Units) mit Geräteglobalen Variablen und Technologieobjekten über zwei Einstellungen getrennt initialisieren. Bei der Projekterstellung werden normalerweise die Daten gemeinsam initialisiert. Im Service-Fall oder wenn Sie nur Programme (Units, Quellen) oder Technologieobjekte nachladen möchten, können Sie die Daten getrennt initialisieren.

- Initialisierung der nicht RETAIN Technologieobjektdaten; bei TOs können nur nicht RETAIN Daten initialisiert werden. Die Programmdaten (Quelldaten) werden nicht beeinflusst.
- Initialisierung der nicht RETAIN Programmdaten und der nicht RETAIN Geräteglobalen Variablen. Zusätzlich können die RETAIN Programmdaten und RETAIN Geräteglobalen Variablen initialisiert werden. Die Daten der Technologieobjekte werden nicht beeinflusst. Die Geräteglobalen Variablen werden gleichzeitig mit den Einstellungen zu den Programmdaten (Quelldaten) initialisiert.

Wie Sie die Einstellungen vornehmen, siehe CPU/Antriebsgerät ins Zielgerät laden (Seite 530).

Generelle Informationen zur Variableninitialisierung finden Sie unter "Zeitpunkt der Variableninitialisierung" in den Programmierhandbüchern.

11.8 Download im RUN

11.8.1 Download im RUN durchführen

Download im RUN durchführen

Wie bei einem Download im STOP werden auch bei einem Download im RUN immer alle Änderungen geladen. Für einen Download im RUN sollten daher jeweils immer nur überschaubare Änderungen vorgenommen werden. Sie können ein ganzes Zielgerät oder auch nur Teilmengen des Zielgerätes laden. Variablen werden standardmäßig nicht initialisiert, da Sie sonst die aktuellen Werte des Systems überschreiben würden.

Download im RUN ermöglichen

1. Wählen Sie im SCOUT unter **Extras > Einstellungen > CPU-Download** die Option **Download/RAM nach ROM im RUN ermöglichen** aus. Dadurch wird die Option **Download im Run durchführen** im Dialog **Laden ins Zielsystem** aktiv.
2. Klicken Sie die Option **Download im Run durchführen** an.

Für detaillierte Information zu den einzelnen Downloads, siehe

- Download ohne HW Konfig Information (Seite 545)
- Download im RUN von geänderten Quellen (Seite 535)
- Download von geänderten Technologieobjekten (Seite 546)

11.8.2 Download im RUN von geänderten Quellen

Download im RUN von geänderten Quellen

Sind in einer Quelle (Unit) mehrere Programme, FBs oder FCs enthalten, so wird immer die gesamte Unit geladen.

Ein Beispiel für einen Download finden Sie unter Beispiel für einen Download von geänderten Quellen (Seite 571) .

Randbedingungen für den Download im RUN von geänderten Quellen

- Die Programme, FB´s und FC´s einer Unit dürfen zum Einwechselzeitpunkt nicht durchlaufen werden.
  - Bei zyklischen Tasks wird einige Sekunden versucht diese im Zykluskontrollpunkt (dem Zeitpunkt, zu dem alle betroffenen zyklischen Tasks neu gestartet werden bzw. nicht laufen) einzuwechseln, danach wird der Vorgang abgebrochen.
  - Laufende MotionTasks, wie Langläufer, können nicht eingewechselt werden. Sie können aus dem SCOUT heraus MotionTasks stoppen und wieder starten, siehe Steuern von Motion Tasks aus dem SCOUT (Seite 543) .
- Die Anzahl der zyklischen Tasks, in deren Programmen (POEs) Änderungen vorgenommen werden können, ist auf 4 begrenzt. Es muss ein Einwechselzeitpunkt innerhalb der Zeitdauer für alle einzuwechselnden Tasks gefunden werden, um alle gleichzeitig einzuwechseln.
- Wenn die Anzahl der Quellen (Units) und der betroffenen Tasks aus Zeit- oder Mengengründen zu groß ist, kann die Summe der Änderungen nicht bearbeitet (geladen) werden. Dieser Umstand wird im Ausgabefenster angezeigt und die kompletten Änderungen nicht in den Ablauf übernommen. Die alten Daten bleiben wirksam.
- Es dürfen keine Änderungen in VAR_GLOBAL Blöcken vorgenommen werden. Mit dem Pragma BlockInit_OnChange bzw. PRAGMA-Zeilen in den Deklarationstabellen können Sie diese Änderungen trotzdem ermöglichen (siehe unten).
- Programminstanzdaten sind die statischen Variablen der Programme (VAR)
- Es dürfen keine Änderungen in VAR Blöcken vorgenommen werden, wenn diese von zyklischen Tasks verwendet werden (die Daten einer zyklischen Task bleiben über den Durchlauf der Task erhalten). VAR_TEMP bei Programmen und FBs, sowie VAR bei FCs können geändert werden. Mit dem Pragma BlockInit_OnChange und dem Compilerschalter "Programminstanzdaten nur einmal anlegen" können Sie diese Änderungen trotzdem möglich machen (siehe unten).
- Die Programminstanzdaten nicht-zyklischer Tasks (MotionTasks) können geändert werden, da diese bei jedem Start neu initialisiert werden.

Download im RUN ermöglichen

Unterstützen Sie durch geeignete Programmierung, dass ein Download von geänderten Quellen im Betriebszustand RUN möglich ist:

- Verwenden Sie die USES-Anweisung möglichst im Implementationsabschnitt und nicht im Interfaceabschnitt. Es sind dadurch beim Laden weniger Units betroffen. Siehe dazu auch *Uses-Anweisung in einer importierenden Unit* im *ST-Programmierhandbuch*.
- Wenn das Programm der geänderten Unit einer MotionTask zugeordnet ist, sehen Sie Möglichkeiten vor, dass die MotionTask nicht aktiv ist, damit sie eingewechselt werden kann:
  - Vermeiden Sie Endlosschleifen in MotionTasks! Verwenden Sie z. B. statt einer WHILE-Schleife die Funktion _restartTaskId()
  - Sehen Sie eine Bedienfunktion (z. B. Betriebsart Einzeltakt) vor, damit Sie einzelne, mehrere oder alle MotionTask zurücksetzen können. Diese Funktion ist ab V4.1.2 auch über den SCOUT verfügbar (siehe Motion Tasks steuern (Seite 543)).

- Vor allem bei MotionTasks ist es günstig Programmteile, die immer laufen (z. B. Ablaufsteuerung in MCC) und Programmteile die fallweise aufgerufen werden, in eigenen Units abzulegen. Diese aufgerufenen Programmteile können dann, wenn sie nicht im Eingriff sind, ausgetauscht werden.
- Legen Sie Units entsprechend der Taskzuordnung der Programme an. Dies verringert Abhängigkeiten und fördert die Übersichtlichkeit.
  Nicht so zuordnen (Beispiel, wie Sie es nicht machen sollten): Eine Unit enthält zwei Programme, eines davon ist der BackgroundTask zugeordnet, das andere einer MotionTask. Dann ist ein Download im RUN dieser Unit nicht möglich, solange die MotionTask aktiv ist.
- In Taskkonfiguration die Größe des Lokaldatenstacks projektieren (siehe Kap. Ablaufsystem konfigurieren sowie Kap. Größe des Lokaldatenstacks einstellen): Berücksichtigen Sie eine Reserve für den Download im RUN. So kann z. B. während des Downloads im RUN temporär zusätzlicher Speicher auf dem Lokaldatenstack benötigt werden (z. B. für neue lokale Variablen und Funktionsparameter).
- Anstatt eines zusätzlichen Variablenblocks ist es auch möglich eine neue UNIT mit den neuen Daten anzulegen und diese mit USES (im Implementation-Teil) zu verbinden. Dies ist auch in MCC und KOP/FUP möglich.
- Werden neue I/O-Variablen benötigt, müssen diese im Prozessabbild der Backgroundtask (0-63 Byte) liegen (eine detaillierte Beschreibung dazu finden Sie im Programmierhandbuch SIMOTION ST).
  Beispiel:

```
VAR_GLOBAL
 anio AT %I2.7 : BOOL;
END_VAR
```

Verbesserung für einen Download im RUN (ab V4.1)

Durch Verwendung eines Compiler-Schalters und -Pragmas kann ein Download im Run verbessert werden:

- Setzen Sie den Compilerschalter "Programminstanzdaten nur einmal anlegen", da dann durch geänderte Instanzdaten eines Programms keine anderen Programme betroffen sind (ab V4.1, siehe unten).
- Instanzdaten von Programmen (VAR) können über das Compiler-Pragma "BlockInit_OnChange := True;" geändert bzw. ab V4.2 über eine entsprechende PRAGMA-Zeile in den Deklarationstabellen (MCC, KOP/FUP) geändert und neu initialisiert werden (siehe unten).
- Änderungen von TYPE und globalen Variablen im Interface- und Implementation-Abschnitt einer Unit sind über einen zusätzlichen Variablenblock oder durch das Compiler-Pragma "BlockInit_OnChange := True;" möglich (ab V4.1) bzw. ab V4.2 über eine entsprechende PRAGMA-Zeile in den Deklarationstabellen (MCC, KOP/FUP) (siehe Einfluss des Compilers auf die Variableninitialisierung (Seite 317).)
- Der Download im RUN eines Programms in einer zyklischen Task ist bei geänderten Instanzdaten nur möglich, wenn keine Task (zyklische als auch sequentielle) aktiv ist.

Compiler-Schalter für einmalige Programmdateninstanziierung

Die Programmdateninstanziierung und Ablage der Programminstanzdaten ist wichtig im Zusammenhang mit einem Download im Run.

Compiler-Schalter "Programminstanzdaten nur einmal anlegen" ist nicht gesetzt (Vorbelegung)

- Die Dateninitialisierung erfolgt mit dem Starten der Task.
- Die Instanzdaten aller Programme liegen in einem zentralen Speicherbereich (zur Diagnose sind diese im Symbolbrowser zum Ablaufsystem zu finden). Ist ein Programm mehreren Tasks zugeordnet, so werden auch je Task eigene Instanzdaten angelegt. Wenn Instanzdaten eines Programms geändert werden, sind (durch die zentrale Ablage) auch die Instanzdaten der anderen aktiven Programme davon betroffen, was Einschränkungen beim Download im RUN mit sich bringt.
- Ab V4.1.2 können Sie MotionTasks auch vom SCOUT aus steuern. Sie können so MotionTasks stoppen, um für einen Download im RUN einen Einwechselzeitpunkt zu bekommen und anschließend die Motion Tasks wieder starten, siehe Motion Tasks steuern.
- Der Download im RUN eines Programms in einer nicht-zyklischen Task (sofern diese und andere nicht-zyklischen Tasks nicht aktiv sind) ist auch bei einer Änderung der Instanzdaten möglich, da der Initialisierungszeitpunkt der Programmdaten der Taskstart ist.

Compiler-Schalter "Programminstanzdaten nur einmal anlegen" ist gesetzt.

- Hier erfolgt die Dateninitialisierung mit dem Download des Programms bzw. der Quelle (Unit), in der das Programm liegt bzw. mit dem Hochlauf der CPU.
- Instanzdaten so übersetzter Programme werden nur einmal angelegt, auch wenn das Programm in mehreren Tasks verwendet wird. Die Instanzdaten liegen dann in der Quelle bzw. im Code des Programms (zur Diagnose sind diese daher im Symbolbrowser der Unit zu finden).
- Dies hat Vorteile für einen Download im RUN, da hier bei geänderten Instanzdaten eines Programms nicht die Instanzdaten anderer Tasks betroffen sind.
- Der Download im RUN eines Programms in einer zyklischen oder sequentiellen Task ist bei geänderten Instanzdaten auch möglich, wenn andere Tasks (zyklische als auch sequentielle) aktiv sind.

Informationen, welche Tasks sequentiell oder zyklisch laufen können, finden Sie unter Ablaufebenen/Tasks (Seite 191).

VORSICHT
Verändertes Verhalten bei der Dateninitialisierung, wenn Sie "Programminstanzdaten nur einmal anlegen" ausgewählt haben. Die Dateninitialisierung erfolgt bei sequentiellen Tasks nicht mehr mit einem Taskstart bzw. bei zyklischen Tasks mit dem STOP-RUN-Übergang, sondern generell nur bei einem Download bzw. beim Hochlauf der CPU. Ggf. muss nun die Dateninitialisierung applikativ in der StartUpTask bzw. zu Beginn der Programme in sequentiellen Tasks erfolgen. Eine Initialisierung mit dem STOP-RUN-Übergang ist ab V4.1.2 über das Pragma "BlockInit_OnDeviceRun" möglich, siehe Initialisierung von Daten bei einem STOP - RUN - Übergang (Seite 542) . Ab V4.2 kann die Initialisierung bei Stop-Run-Übergängen auch im Dialog **Eigenschaften > Einstellungen** am Gerät (Kontextmenü) aktiviert werden. Ist ein Programm mehreren Tasks zugeordnet, wird auf den gleichen Instanzdaten gearbeitet.

Einstellung des Compilerschalters

- Für die globale Einstellung aktivieren Sie unter **Extras > Einstellungen > Compiler** die Einstellung **Programminstanzdaten nur einmal anlegen**.

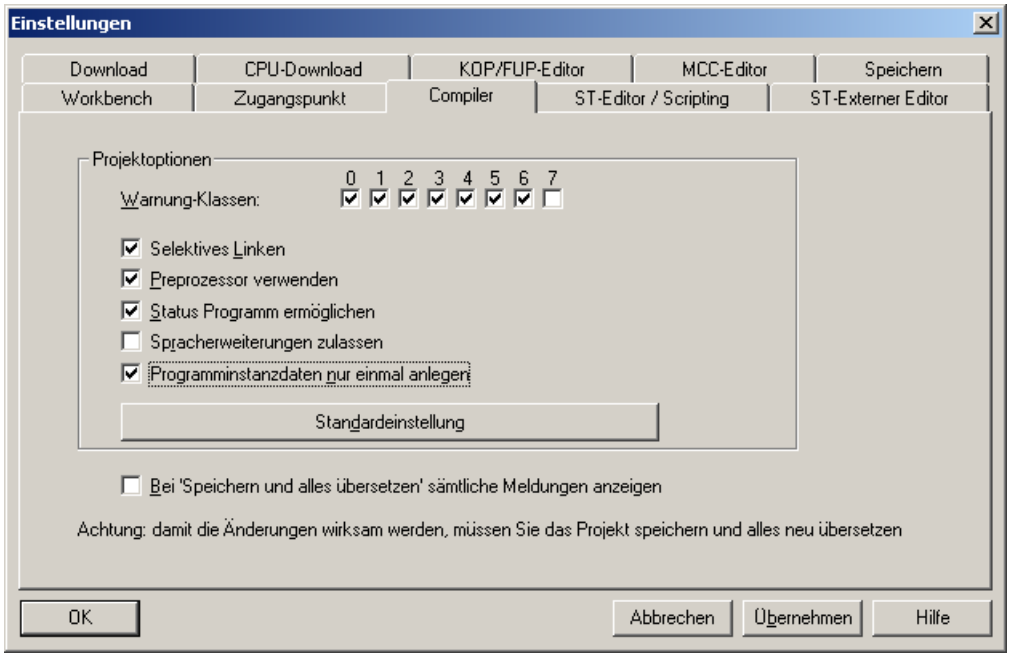


Bild 11-4 Compilereinstellungen

- Für die lokale Einstellung aktivieren Sie beim Einfügen einer Programmquelle (ST und MCC) im Dialog **xxx einfügen** bzw. im Dialog **Eigenschaften** der Quelle unter Compiler die **Einstellung Programminstanzdaten nur einmal anlegen** (xxx = Programm, Funktion, Funktionsbaustein). Beachten Sie bitte, dass eine Änderung nur dann möglich ist, wenn das Optionskästchen weiß ist (zweimal auf das Kästchen klicken). Der Compiler-Schalter muss vor einem Download im RUN gesetzt sein (d.h. Quelle muss vorher mit einem Download im STOP geladen worden sein).

Neuinitialisierung von Programminstanzdaten durch Compiler Pragma

Über Pragmazeilen in den Deklarationstabellen bzw. über das Pragma "{BlockInit_OnChange := TRUE;}" kann bewirkt werden, dass bei Änderungen am Blockaufbau beim Download im RUN eine Neuinitialisierung der Daten mit den in der Quelle spezifizierten Werten vorgenommen wird. Das Pragma ist in VAR_GLOBAL Blöcken im Interface- und Implementation-Teil der Quelle (Unit) anwendbar.

Bei Änderungen in VAR_GLOBAL Blöcken im INTERFACE-Abschnitt verhindern ggf. laufende MotionTasks trotz des oben genannten Pragmas einen Download im RUN (Abhilfe siehe Motion Tasks steuern (Seite 543)).

Bei Änderungen in VAR_GLOBAL Blöcken im INTERFACE-Abschnitt können weitere Quellen betroffen sein, die dann mit geladen werden müssen. Dies betrifft Quellen, die über USES die geänderte Quelle importieren (ST) bzw. in der Deklarationstabelle mit der geänderten Quelle verbunden sind (MCC, KOP/FUP). Hingegen ist bei Änderungen in VAR_GLOBAL Blöcken im IMPLEMENTATION-Abschnitt nur die geänderte Quelle beim Download betroffen.

Das Pragma ist auch in VAR-Deklarationen von Programmen anwendbar. Für VAR-Deklarationen von Programmen müssen Sie zusätzlich "Programminstanzdaten nur einmal anlegen" eingestellt haben. Es kommt eine Meldung, wenn beim Übersetzen der Quelle das Pragma nicht wirkt.

Das Pragma darf nur am Anfang eines Blocks stehen.

Das Pragma BlockInit_OnChange steht zurzeit nur in ST-Quellen zur Verfügung.

Beispiel für die Syntax von BlockInit_OnChange:

```
Var_Global
   {BlockInit_OnChange := TRUE;}
   Interface_Var_Global1 : INT;
   Interface_Var_Global2 : REAL;
END_VAR
```

In KOP/FUP und MCC kann in Deklarationstabellen über das Einfügen von Pragmazeilen (Kontextmenü) die Initialisierung aktiviert werden (siehe entsprechende Programmierhandbücher zu KOP/FUP und MCC). Das Pragma kann jederzeit bei Bedarf gesetzt werden.

Beispiel:

Ist ein Download im RUN auf Grund einer geänderten VAR im Programm nicht möglich, kann anschließend das Pragma im VAR-Block gesetzt werden. Nach dem Übersetzen ist ein Download im RUN möglich, da dieser VAR-Block neu initialisiert wird.

Vor dem nächsten Download kann dann das Pragma wieder entfernt werden. Falls nicht, wird eine Neuinitialisierung jedoch nur bei einer Änderung im VAR-Block durchgeführt.

Siehe auch Einfluss des Compilers auf die Variableninitialisierung (Seite 317).

Fehlermeldungen, wenn Download nicht möglich

Falls eine Änderung im RUN nicht möglich ist, werden beim Laden Fehlermeldungen ausgegeben, z. B. UPP Änderung um RUN unmöglich (UPP = UserProgramProcessing). UPP zeigt dann an, dass ein Fehler beim Einwechseln eines Anwenderprogramms aufgetreten ist. Das alte Programm bleibt in der Steuerung erhalten, die Änderung wird verworfen. Siehe hierzu auch Beispiel für einen Download von geänderten Quellen (Seite 571) .

Die Änderung der Quelle im SCOUT-Projekt kann einfach wieder rückgängig gemacht werden, indem die aktuellen Daten auf der Steuerung wieder ins Engineeringsystem geladen werden. Dadurch wird das SCOUT-Projekt wieder online konsistent.

Siehe hierzu Daten aus dem Zielgerät ins PG/PC laden (Seite 552).

Siehe auch

Download von geänderten Technologieobjekten (Seite 546)

11.8.3 Initialisierung von Daten bei einem STOP - RUN - Übergang

Beschreibung

Durch den Compilerschalter "Programminstanzdaten nur einmal anlegen" erfolgt die Dateninitialisierung nur bei einem Download oder im Hochlauf der CPU. Ab V4.1.2 können Sie mit einem Pragma in den Units einstellen, dass Globale Unit Variablen und Programm Variablen auch bei einem STOP - RUN - Übergang initialisiert werden. Die Initialisierung erfolgt unmittelbar vor Start der StartupTask. Sie haben damit die Möglichkeit bei einem STOP - RUN - Übergang auf jeden Fall eine Initialisierung der Variablen durchzuführen.

Sie müssen dazu das Pragma "BlockInit_OnDeviceRun" einfügen.

Daten in ST Units immer /nie initialisieren

- In ST-Units können Sie die Initialisierung bei einem STOP - RUN - Übergang durch das Pragma "BlockInit_OnDeviceRun", am Anfang der Variablenblöcke, beeinflussen. Wird dieses Pragma im Var_Global Block oder im Var Block von Programmen (wenn Programminstanzdaten nur einmal angelegt sind) auf "ALWAYS" gestellt, werden die Variablen bei jedem STOP - RUN - Übergang initialisiert. Die Einstellung ist immer nur für den Variablenblock gültig, in dem das Pragma verwendet wird. Möchte man eine Initialisierung im Stop/Run Übergang gezielt verhindern, setzt man das Pragma mit Einstellung "DISABLE". Um das Pragma für VAR-Deklarationen von Programmen nutzen zu können, müssen Sie zusätzlich "Programminstanzdaten nur einmal anlegen" eingestellt haben.

Deklararionsblock	BlockInit_OnDeviceRun
Interface	
Var_Global	Ist möglich
Implementation	
Var_Global	Ist möglich
Program	
Var	nur bei Compileroption "Programminstanzdaten nur einmal anlegen", sonst nein

Der Compiler gibt eine Warnung aus, wenn das Pragma an einer Stelle verwendet wird, an der es nicht wirksam ist.

Einstellmöglichkeiten des Pragmas ("immer" oder "nie")

Initialisierung ... STOP-RUN - Übergang	
Immer	BlockInit_OnDeviceRun := ALWAYS
Nie	BlockInit_OnDeviceRun := DISABLE

Beispiel für "immer" initialisieren:

```
VAR_GLOBAL
     {BlockInit_OnDeviceRun := ALWAYS;}
     Test_1 : REAL;
     Test_2 : REAL;
 END_VAR
```

Beispiel für "nie" initialisieren (Compilerschalter "Programminstanzdaten nur einmal anlegen" ist aktiv):

```
VAR_GLOBAL
     {BlockInit_OnDeviceRun := DISABLE;}
     Test_1 : REAL;
     Test_2 : REAL;
 END_VAR
```

Initialisierung über den Dialog Geräteeigenschaften (ab V4.2)

Wenn Sie die Option **Initialisierung der nicht-retain globalen (var_global und gerätegloba le Variable) und Programmvariablen (var) bei STOP - RUN - Übergang** im Eigenschaftsdialog eines Gerätes auswählen, können Sie die Initialisierung unmittelbar vor dem Start der StartupTask anstoßen. Diese globale Einstellung kann mit einem Pragma (siehe oben) überschrieben werden.

11.8.4 Steuern von Motion Tasks aus dem SCOUT

11.8.4.1 Motion Tasks steuern

Beschreibung

Sie können ohne ein selbst erstelltes Anwenderprogramm MotionTasks aus dem SCOUT heraus steuern.

Damit können Sie Programme testen und gezielt auf den Ablauf von MotionTasks einwirken. Sie können ausgewählte MotionTasks anhalten, für den Ablauf sperren oder wieder starten.

Download im RUN unterstützen

Falls Sie Änderungen an Quellen gemacht haben und diese durch einen Download im RUN nachladen möchten, kann eine aktive MotionTask verhindern, dass ein Einwechselzeitpunkt für den Tausch der Quellen gefunden wird. Wird sie nicht beendet, können Sie gezielt MotionTasks mit dem SCOUT beenden, um einen Download im RUN durchzuführen.

11.8.4.2 Debug-Modus einschalten und MotionTasks steuern

Voraussetzung

Um MotionTasks steuern zu können müssen Sie ONLINE sein und in den Debug-Modus wechseln.

1. Klicken Sie dazu mit der rechten Maustaste auf das entsprechende Gerät und führen Sie im Kontextmenü **Testbetrieb** aus.
2. Wählen Sie dann im Dialog **Testbetrieb** die Option **Debug-Modus** aus, lesen Sie die Sicherheitshinweise, akzeptieren diese durch Anklicken der Option und klicken Sie auf **OK**.

Vorgehensweise

1. Wenn Sie den Debug-Modus aktiviert haben, rufen Sie dann über **Zielsystem > Gerätediagnose** die Gerätediagnose auf.
2. Wählen Sie eine MotionTask aus und klicken Sie mit der rechten Maustaste darauf. Ein Kontextmenü wird aufgeblendet.
3. Wählen Sie **Taskstart sperren** aus, wenn Sie den Start der MotionTask sperren möchten. Die Task wird dann daran gehindert wieder anzulaufen, sie wird nicht gestartet (TASKSTART_LOCKED im Taskstatus).
4. Wählen Sie **Setze Task zurück** aus, wenn Sie die Task in den Zustand STOPPED setzen möchten. In diesem Zustand kann ein Download im RUN durchgeführt werden.
5. Wählen Sie **Taskstart frei geben** aus, wenn Sie den Taskstart frei geben möchten. Sie können dann die Task aus dem Programm oder über den SCOUT wieder starten.
6. Wählen Sie **Starte Task** aus, wenn Sie die MotionTask starten möchten.

Nach dem Ausführen der Tasksteuerbefehle dauert es etwas, bis die Anzeige im SCOUT aktualisiert wird. Sie können auch auf die Schaltfläche **Aktualisieren (F5)** klicken.

Mehrere MotionTasks gleichzeitig steuern

1. Wählen Sie die gewünschten Tasks aus, indem Sie sie mit gedrückter Strg und Mauszeiger anklicken.
   Die Tasks werden ausgewählt.
2. Klicken Sie auf die Schaltfläche MotionTasks steuern und wählen dann im aufgeblendeten Kontextmenü den entsprechenden Befehl aus.

Hinweis

Die Tasksteuerbefehle sind unabhängig von den systemeigenen Tasksteuerbefehlen. Den Taskstatus können Sie im Register **Tasklaufzeiten** unter **Taskstatus** ablesen.

Verhalten beim Ändern der Betriebsarten im Testbetrieb

Das System zeigt folgendes Verhalten, wenn Sie bewusst oder aus Versehen den Debug-Modus verlassen und mindestens eine Task gesperrt ist.

- Wird die Betriebsart **Debug-Modus** verlassen, ohne dass vorher alle Tasks frei gegeben wurden, geht die CPU in STOP und die Sperren werden aufgehoben.
- Gehen Sie OFFLINE, ohne dass vorher alle Tasks frei gegeben wurden, geht die CPU in STOP und die Sperren werden aufgehoben. Nach dem ONLINE-Gehen befindet sich die CPU wieder in Prozessbetrieb (Debug-Modus wird aufgehoben).
- Ein RUN - STOP - RUN Übergang wirkt sich nicht auf den Taskzustand aus. War die Task vor dem Start gesperrt, bleibt sie für den Start gesperrt.

Die CPU geht nach STOP, wenn der SCOUT die Kommunikation zur Steuerung unbeabsichtigt verliert. Der SCOUT geht gleichzeitig OFFLINE.

11.8.5 Download ohne HW Konfig Information

Download mit inkonsistenter HW Konfig

Ab V4.1.2 ist es möglich, einen Download auch mit inkonsistenter HW Konfig durchzuführen. Dabei wird ermittelt, ob ein Download ohne HW Konfig möglich ist oder nicht. Die Überprüfung ermittelt, ob die Änderungen an der Hardware Konfiguration so gravierend sind, dass auf jeden Fall die HW Konfig mit geladen werden muss. Andernfalls kann auf den Download der HW Konfig verzichtet werden.

Hinweis

Die Einstellungen für die Telegrammlänge, Adresse und Datentypen werden in HW Konfig im Dialog **DP Slave Eigenschaften** im Register **Konfiguration** angepasst.
Die Takteinstellungen werden im Register **Taktsynchronisation** durchgeführt.

Diese Parameteränderungen verhindern einen Download ohne HW Konfig

- Logische Adressen
  - Adresse in HW Konfig verschieben
  - Separate Slots zu DP Teilnehmern oder zu vorhandenen Teilnehmern hinzuzufügen
- Telegrammlänge
  - Die Telegrammlänge muss immer gleich bleiben.
- Takteinstellungen
  - DP Zykluszeit und PN Sendetakt darf nicht verstellt werden.

Download ohne HW Konfig einstellen

In den Dialogen **Laden ins Zielsystem** und **Laden in CPU/Antriebsgerät** unter **Zusatzoptionen CPU** können sie diese Zusatzoption auswählen. Die Voreinstellung der Zusatzoption können Sie unter **Extras > Einstellungen > Download** vornehmen.

11.8.6 Download von geänderten Technologieobjekten

Beschreibung

Sie können offline vorgenommen Änderungen an (durch Restart und sofort wirksamen) Konfigurationsdaten sowie an Systemvariablen eines TOs im RUN nachladen. Gegebenenfalls wird nach dem Download ein TO Restart ausgeführt.

TOs hinunter laden

Der Download eines geänderten TOs ist unter den folgenden Umständen bzw. Voraussetzungen möglich:

- Es können nur Technologiobjekte geladen werden, deren Struktur erhalten bleibt (siehe unten)
- Ein Download wird auch ausgeführt, wenn ein TO von einem Programm benutzt wird, z. B. Achse ist frei gegeben. Gegebenenfalls werden dann TO-Alarme ausgegeben.
- Die während des Downloads gelieferten Ersatzwerte des TO werden entsprechend der Einstellung von "restart.behaviorInvalidSysvarAccess" (LAST_VALUE, DEFAULT_VALUE) zurückgeliefert. Kommt es bei einem Download von TOs zu einem Restart, können Zugriffsfehler aus dem Anwenderprogramm auftreten. Die Reaktion der CPU bzw. der Rückgabewert ist dann abhängig von den Einstellungen des Konfigdatums "restart.behaviorInvalidSysvarAccess". Ist dieses Konfigdatum auf STOPPED gesetzt, geht die CPU in STOP. Weitere Informationen finden Sie unter Systemvariablen (Seite 135) .
- Kann beim Download die Änderung des TOs nicht eingewechselt werden oder tritt ein Fehler auf, bleibt die ursprüngliche Konfiguration erhalten bzw. wird wieder hergestellt (Situation wie vor dem Download).
- Die TOs werden nacheinander geladen und sind dann sofort wirksam.

Verhalten, wenn das Laden von mehreren TOs nicht möglich ist

Laden Sie eine Gruppe von TOs und der Download eines TOs ist nicht möglich:

- bleibt die Konfiguration für bereits geladenen TOs aktiv
- der Download für das TO, das nicht geladen werden konnte, wird verworfen
- der Download für die noch nicht geladenen TOs wird fortgesetzt.

TO Restart erlauben

Bevor der Download durchgeführt wird, können Sie mit einer Option auswählen, ob ein RESTART der TOs erlaubt sein soll oder nicht. Dadurch können Sie einen TO-RESTART verhindern. Ein RESTART wird dann unabhängig von der Einstellung unter RestartCondition ausgeführt.

Bild 11-5 TO Übersicht bei Download

1. Wählen Sie die Option **Restart an Technologieobjekten erlauben** aus.

Die Option ist standardmäßig ausgewählt. Wenn Sie sie abwählen, wird diese Einstellung für weitere Downloads mit diesem Projekt gespeichert, der Dialog wird aber weiterhin eingeblendet. Das TO verhält sich dann so, als ob der Download nicht funktioniert.

Änderbare Parameter, die einen Download im RUN ermöglichen

Grundsätzlich wirken sich download-änderbare Konfigurationsdaten auf die Struktur des TOs aus. Daneben wirken sich auch Änderungen an Einheiten, Verschaltungen, Alarmkonfiguration und Profile auf die Struktur eine TOs derart aus, dass kein Download im RUN mehr möglich ist. Bei einem Download im RUN können Sie deshalb folgende Daten ändern:

- Sofort wirksame Konfigurationsdaten
- Restart-wirksame Konfigurationsdaten
- Systemvariablen

In der Expertenliste der TOs können Sie für die Konfigurationsdaten in ONLINE- sowie im OFFLINE-Betrieb in der Spalte **Wirksamkeit** erkennen, ob die Änderung an einem Konfigurationsdatum sofort wirksam wird, einen Restart benötigt oder nur über einen Download (dann im STOP) möglich ist.

Download auf Runtime mit Version V4.1.1 durchführen

Wenn Sie mit SCOUT V4.1.2 einen Download im RUN mit den erweiterten Funktionen zu V4.1.2 durchführen, wir der SCOUT den Download erlauben. Das Runtime-System V4.1.1 wird den Download jedoch ablehnen.

11.8.7 RAM nach ROM kopieren im RUN

Beschreibung

Wenn Sie einen Download im RUN durchgeführt haben, können Sie die Änderungen auch noch auf die Speicherkarte netzausfest sichern.

RAM nach ROM kopieren kann bereits über eine Option im Download-Dialog angewählt werden. Nach einem Download ist dies zusätzlich über eine separate Funktion möglich (über die Schaltfläche **RAM nach ROM kopieren** in der Funktionsleiste oder im Geräte-Kontextmenü unter Zielgerät).

Die Voreinstellung für die Option im Download-Dialog ist unter **Extras > Einstellungen > CPU-Download** möglich. Hier kann auch angewählt werden, ob vor dem RAM nach ROM kopieren die Aktualwerte der TO-Konfigurationsdaten ins RAM übernommen werden.

Hinweis

Bei einem "RAM nach ROM kopieren im RUN" darf die CPU-Auslastung nicht im kritischen Bereich liegen (siehe Gerätediagnose), da durch diese Funktion zusätzliche Last entsteht.

Download / RAM2ROM im RUN in Projekten mit älteren Runtime-Versionen (< V4.1) ist nicht möglich und wird aktiv verhindert.

Inkonsistentes TO nach einem Download im RUN

Nach einem Download im RUN kann es vorkommen, das TOs inkonsistent werden, weil in der Applikation ein TO-Konfigurationsdatum geändert wurde. Sie haben Folgendes eingestellt:

- unter **Einstellungen > CPU-Download > Bei RAM nach ROM Aktualwerte übernehmen**
- unter **Download > RAM nach ROM kopieren**

Die durch die Applikation geänderten Konfigurationsdaten liegen dann im Aktualspeicher. Durch die Einstellungen werden sie ins RAM und auf die Speicherkarte (ROM) kopiert. Dadurch zeigt der Projektnavigator eine Inkonsistenz an. Die projektierten Werte im Projekt und die Werte im RAM sind unterschiedlich.

Sie können dann folgendermaßen vorgehen:

- TO-Inkonsistenz ignorieren und nachfolgend einen Download von Programmen über "Laden aller Programme" durchführen.
- Die TO-Inkonsistenz über den Projektvergleich analysieren
- Die TO-Konfigurationsdaten über **Laden in PG** ins PG laden
- Die TO-Konfigurationsdaten über einen Download ins Gerät laden

11.9 Download direkt auf Speicherkarte oder Festplatte

Mit **Laden ins Dateisystem** können Sie die ablauffähigen Projektdaten aus dem SCOUT auf die Festplatte des PC/PG bzw. direkt über einen Kartenadapter auf das Speicherkärtchen speichern.

Es stehen 2 Arten der Speicherung zur Verfügung:

- Mit der Option "Speichern normal" wird das User-Verzeichnis mit den SIMOTION - und SINAMICS - Unterverzeichnissen erzeugt.
- Mit der Option "Speichern komprimiert" wird der gleiche Inhalt als ZIP-Archiv erzeugt die z. B. für das Laden über SIMOTION IT-Diag verwendet werden kann.

Das Laden auf das Speicherkärtchen kann wie ein Download mit anschließendem RAM nach ROM betrachtet werden. Diese Funktion ist sehr nützlich, z. B. bei der Serien-Inbetriebnahme.

Vorgehensweise

Zum Speichern der Daten muss SCOUT im OFFLINE-Modus sein und das Gerät muss im Projektnavigator markiert sein, damit diese Funktionalität zur Verfügung steht.

- Führen Sie **Bearbeiten > Laden ins Dateisystem** oder über das Kontextmenü der CPU **Laden ins Dateisystem** aus, um die Daten eines Geräts auf eine Memory Card/CF Card bzw. lokal auf einer Festplatte speichern.

SCOUT erkennt angeschlossene Lese- und Speichergeräte nicht automatisch. Das Speicher-/Lesegerät muss explizit über **Ziel wählen** gewählt werden. Beim Speichern der Projektdaten entsteht ein ablauffähiges Projekt.

Hinweis

Wenn Sie die Daten speichern (sowohl auf Festplatte wie auf CF-Karte) wird automatisch ein USER-Verzeichnis angelegt, so dass Sie die Daten direkt in das ROOT-Verzeichnis einer CF-Karte kopieren können.

Daten von Festplatte auf Karte laden

Sie können die Daten von der Festplatte auf die CF Card kopieren. Ab V4.2 werden die Inhalte der folgenden Verzeichnisse gelöscht bzw. überschrieben:

- \USER\SIMOTION\RT_DIR
- \USER\SINAMICS\DATA\RT_DIR

Die Anwenderdaten in den unten genannten Verzeichnissen bleiben erhalten.

Sichern Sie aber ggf. vor dem Laden der Daten auf die Speicherkarte die USER-Verzeichisses, in denen sich Ihre auf der Karte gespeicherten Anwenderdaten befinden.

Beispiel:

- Sicherung der Retain-Daten (mit _savePersistentMemoryData gesicherte Netz-Aus-feste Daten) abgelegt im Verzeichnis:
  - user\simotion\pmemory.xml
- Sicherung von IT DIAG Anwenderfiles, Einstellungen (z. B. trace.xml), Task Trace-Daten, Log-Files und Java-Files (Klassen, Archive, Anwenderfilesystem, ..), abgelegt in den Verzeichnissen:
  - user\simotion\hmicfg
  - user\simotion\hmi
- Sicherung von Unit-Daten (mit _saveUnitDataSet /_exportUnitDataSet auf CF Card gesicherte Daten), abgelegt im Verzeichnis:
  - user\simotion\user dir\<unitname>

Hinweis

Die gespeicherten Daten müssen zur Firmware der Karte passen. Falls Sie z. B. eine ältere Projektierung auf eine Karte mit aktueller Firmware laden, erhalten Sie Fehlermeldungen beim Hochlauf.

11.10 Download über Geräte Update Tool

Beschreibung

SIMOTION bietet eine komfortable Lösung für das Hochrüsten von SIMOTION Geräten bzw. SIMOTION Projekten für Maschinenhersteller und Maschinenbetreiber. Hochrüsten bedeutet nicht nur ein Update für eine höhere Firmware, sondern generell ein Wechsel auf eine definierte Konfiguration, z. B. ein Projekt-Update. Dabei ist es auch möglich auf eine vorhergehende Konfiguration zurückzugreifen (Rückrüsten). SIMOTION Geräte können vor Ort oder Remote auf einfache Weise hoch- bzw. rückgerüstet werden. Die Daten können über ein portables und einfach zu handhabendes Speichermedium (z. B. auch USB-Stick) bzw. eine Kommunikationsverbindung in ein SIMOTION Gerät eingespielt werden.

Hinweis

Die Firmware von angeschlossenen SINAMICS-Antrieben wird nur dann aktualisiert, wenn das automatische Firmwareupdate in Parameter p7826 eingestellt ist (in jedem Antriebsgerät, das aktualisiert werden soll).

Detaillierte Informationen finden Sie unter SIMOTION Geräte hochrüsten.

11.11 Daten aus dem Zielgerät ins PG/PC laden

Mit **Laden ins PG** werden die Daten aus dem Zielgerät ins PG/PC geladen. Damit kann das aktuelle geladene Projekt aus dem Zielgerät mit allen Einstellungen auf die lokale Festplatte geladen werden.

Quellen oder Zusatzdaten eines Projektes können nur dann ins PG geladen werden, wenn diese vorher in das Zielsystem geladen worden sind. Im Projekt musste dazu **Extras > Einstellungen > Download > Zusatzdaten** auf dem Zielgerät ablegen eingestellt gewesen sein.

Der Menüeintrag **Laden ins PG** ist nur dann aktiv, wenn Sie ONLINE sind und ein Gerät ausgewählt haben.

Laden mit vorhandenem Originalprojekt

Sie kommen zum Service auf eine in Betrieb genommene Anlage und bringen ein Projekt mit oder laden das archivierte Projekt von der Speicherkarte der Steuerung. Das Projekt ist nicht online konsistent, gleicht aber im Wesentlichen dem Projekt, welches auf dem Zielgerät zum Ablauf kommt. Über den Upload mittels **Laden ins PG** können Sie die Online-Konsistenz zwischen dem SCOUT-Projekt und der CPU herstellen. Danach ist es möglich, Fehler zu suchen und Änderungen auf das Zielgerät zu laden.

Hinweis

Die Objekte im OFFLINE-Projekt werden überschrieben. Damit werden Objekte, die nur OFFLINE vorhanden sind, gelöscht. Wenn Sie sich die Möglichkeit erhalten möchten, nach dem Upload noch auf das OFFLINE-Projekt zuzugreifen, klicken Sie **Speichern vor dem Laden ins PG** an, um die aktuelle Version des OFFLINE-Projektes zu speichern. Sie können das OFFLINE-Projekt erhalten, indem Sie nach dem Upload das Projekt ohne zu speichern verlassen.

Beachten Sie dabei aber, dass verschiedene Funktionen (wie z. B. Status Programm) solange nicht aktiv bzw. benutzbar sind, bis die hochgeladenen Objekte mit dem Projekt gespeichert werden.

Der Abgleich zwischen Offline- und Online-Projekt kann auch über die Vergleichsfunktionen (Projektvergleich) erfolgen.

Vorgehensweise mit vorhandenem Projekt

1. Führen Sie **Zielsystem > Laden ins PG** aus.
   Der Dialog **Laden ins PG** wird eingeblendet.
2. Wählen Sie die Option **Zielgerät ins PG laden** aus, wenn Sie alle Projektdaten aus dem Zielgerät ins PG laden möchten. Beachten Sie dazu auch die Hinweise im Dialog.
3. Wählen Sie die Option **Speichern vor dem Laden ins PG**, wenn Sie das Projekt vor dem Hochladen speichern möchten.
4. Wählen Sie die Option **Vorhandene Bibliotheken überschreiben**, wenn Sie die Bibliotheken des lokal auf Ihrem PG vorhandenen Projekts überschreiben möchten.

Nur Konfigurationsdaten ins PG laden

1. Wählen Sie die Option **Nur Konfigurationsdaten ins PG laden**, wenn Sie die Konfigurationsdaten aus dem RAM ins PG laden möchten.

Aktual nach RAM übernehmen

Falls Sie geänderte Konfigurationsdaten, die sich im Aktualspeicher des Zielgeräts befinden, vor dem Laden ins PG ins RAM laden möchten, klicken Sie die Checkbox **Aktual nach RAM übernehmen** an. Wenn Sie die Aktualdaten nicht ins RAM übernehmen, werden die Aktualdaten nicht mit in das PG geladen.

Ab V4.2 werden Konfigurationsdaten im Hochlauf mit Parameterwerten des SINAMICS-Antriebs gefüllt (Adaption). Für nähere Informationen, siehe **Adaption** unter Symbolische Zuordnung - Einführung (Seite 79). Bei einem "Laden ins PG" werden adaptierte Werte erkannt und automatisch auch ein "Aktual nach RAM übernehmen" vorselektiert.

Laden ins PG über Projektvergleich (objektgranular)

- Über die Funktionalität Projektvergleich können Sie auch die Daten einzelner Objekte in das PG laden. Detaillierte Informationen finden Sie in der Online Hilfe zum Projektvergleich bzw. im Handbuch SIMOTION Projektvergleich.

Siehe auch

Übersicht zum Download von Daten (Seite 523)

11.12 Aktual nach RAM kopieren

Beschreibung

Mit **Aktual nach RAM kopieren** kopieren Sie die aktuellen, während des Zustands RUN geänderten Konfigurationsdaten in das RAM. Änderungen von Konfigurationsdaten im RUN werden abhängig vom Datum unterschiedlich wirksam (z. B. Sofort oder erst nach einem Restart des TO).

Nach dem Restart am TO werden die geänderten Konfigurationsdaten im Aktual-Speicher abgelegt. Um die Werte vom Aktual-Speicher ins RAM zu kopieren, müssen Sie explizit diese Funktion ausführen. Erst nach dem Kopieren sind die Daten im RAM vorhanden und werden beim erneuten Hochlauf aus diesem gelesen und werden wirksam. Siehe auch Überblick über die Speicher im Zielgerät (Seite 515).

Hinweis

Damit bei online geänderten Konfigurationsdaten die Projektdaten im SCOUT mit den Projektdaten im Zielsystem konsistent sind, müssen Sie noch einen Upload ins PG/PC durchführen (Menü **Zielsystem > Laden > Konfigurationsdaten ins PG**).

Adaption der Konfigurationsdaten

Im Hochlauf der SIMOTION Geräte werden Bezugsgrößen, sowie Antriebs- und Geberdaten des SINAMICS S120 (SINAMICS Integrated) automatisch für die Konfigurationsdaten der SIMOTION Technologieobjekte "TO Achse" und "TO Externer Geber" übernommen. Diese Daten müssen daher in SIMOTION nicht mehr eingegeben werden. Die adaptierten Daten befinden sich im Aktualspeicher. Bei einem **Aktual nach RAM kopieren** gelangen diese Werte in das RAM. Das TO würde damit online inkonsistent. Um diese Inkonsistenz aufzulösen, müssen Sie auch die Konfigurationsdaten ins PG laden. Dies wird im Dialog erkannt und entsprechend vorbelegt. Falls Sie selbst Konfigurationsdaten online geändert haben, werden auch diese mit ins PG geladen. Für nähere Informationen zur Adaption, siehe Symbolische Zuordnung - Einführung (Seite 79).

Aktual nach RAM durchführen

1. Führen Sie **Zielsystem > Aktual nach RAM kopieren** aus.

   Der Dialog **Aktual nach RAM kopieren** öffnet sich. Der Dialog zeigt an, ob die Konfigurationsdaten durch Adaption geändert wurden.

2. Wählen Sie unter folgenden Optionen aus:
   - Von RAM nach ROM kopieren
   - In das PG laden

3. Klicken Sie auf **Ja**, um das Kopieren und die eventuell ausgewählten Optionen anzustoßen.

Weitere Einstellmöglichkeiten, um **Aktual nach RAM kopieren** anzustoßen finden Sie unter:

- Dialog **Laden in PG**
- **Extras>CPU-Download>RAM nach ROM kopieren Aktualwerte ins RAM übernehmen**

Siehe auch

RAM nach ROM kopieren (Seite 555)

11.13 RAM nach ROM kopieren

Beschreibung

Mit **RAM nach ROM kopieren** sichern Sie das Projekt aus dem flüchtigen Speicher (RAM) in den remanenten Speicher (ROM). Für eine detaillierte Beschreibung, siehe Überblick über die Speicher im Zielgerät (Seite 515).

RAM nach ROM kopieren mit adaptierten Konfigurationsdaten

Im Hochlauf der SIMOTION Geräte werden Bezugsgrößen, sowie Antriebs- und Geberdaten des SINAMICS S120 (SINAMICS Integrated) automatisch für die Konfigurationsdaten der SIMOTION Technologieobjekte "TO Achse" und "TO Externer Geber" übernommen. Diese Daten müssen daher in SIMOTION nicht mehr eingegeben werden. Die adaptierten Daten befinden sich im Aktualspeicher und **nicht** im RAM-Speicher.

Bei einem **RAM nach ROM kopieren** wird erkannt, ob Werte adaptiert wurden. Um in diesem Falle auch diese Konfigurationsdaten auf die Speicherkarte (ROM) zu sichern und auch den Abgleich mit dem Engineeringprojekt durchzuführen, sind **Aktual nach RAM kopieren** und **Konfigurationsdaten in das PG laden** zusätzlich vorselektiert. Siehe auch Aktual nach RAM kopieren (Seite 553).

11.14 Anwenderdaten von der Speicherkarte löschen

Ein Löschen der Anwenderdaten auf der Speicherkarte ist z. B. erforderlich:

- wenn Sie auf der Speicherkarte ein anderes (neues) Projekt aufspielen möchten und daher ggf. auf der Speicherkarte vorhandene Anwenderdaten eines "alten Projekts" (z. B. Unit-Datensätze) löschen möchten.
- wenn Sie die Technologiepakete auf der Speicherkarte löschen möchten, z. B.
  - wenn Sie ein kleineres Technologiepaket einsetzen wollen (z. B. CAM statt CAM_EXT)
  - um **innerhalb einer Version** ein Hochrüsten des Technologiepakets auf ein neues Hotfix oder Service-Pack zu erzwingen.

  Hinweis

  Bei einem Versionswechsel ist ein Löschen der Anwenderdaten nicht erforderlich. In diesem Fall werden die Technologiepakete auf der Speicherkarte immer aktualisiert.

- Die Anwenderdaten können Sie mit SIMOTION SCOUT löschen. Ab V4.2 können Sie auswählen, welche Daten Sie löschen möchten.

Sie können also weiterhin mit Ihrem PG/PC auf die SIMOTION Baugruppe online gehen. Die Lizenzen auf der Speicherkarte bleiben erhalten.

Anwenderdaten löschen

1. Öffnen Sie im SIMOTION SCOUT das Projekt, das Sie bearbeiten möchten.
2. Gehen Sie mit der Baugruppe online.
3. Markieren Sie die Baugruppe im Projektnavigator und führen Sie **Zielsystem>Anwenderdaten auf Karte löschen** aus.

   Der Dialog **Anwenderdaten von Karte löschen** wird eingeblendet. Dort können Sie auswählen, welche Daten Sie löschen möchten:

   – Projektdaten (Programme, TOs, TPs, SINAMICS-Integrated, …)
   – Unit-Datensätze
   – Netz-Aus-Feste Daten (Urlöschen)
   – Archiviertes Projekt
4. Aktivieren Sie die Optionen, die Sie löschen möchten.
5. Klicken Sie auf **OK**, um die Daten zu löschen.

Hinweis

Netz-Aus-Feste Daten und Unit-Datensätze dürfen nur zusammen mit Projektdaten gelöscht werden.

Fehlerquellen und effizientes Programmieren 12

12.1 Fehlerquellen bei der Programmierung

12.1.1 Fehlerquellen bei der Programmierung

Nachfolgend finden Sie Erläuterungen zu den wichtigsten Fehlerquellen beim Programmieren. Das Kapitel gibt Ihnen auch Lösungsmöglichkeiten zur Hand, um diese Fehlerquellen zu beheben.

12.1.2 Datentypen bei der Zuweisung arithmetischer Ausdrücke beachten

In Ausdrücken wird das Ergebnis immer im größten Zahlenformat des Ausdrucks berechnet.

Eine Wertzuweisung des Ausdrucks an eine Variable ist nur in folgenden Fällen möglich:

- Der berechnete Ausdruck und die zuzuweisende Variable haben den gleichen Datentyp
- Der Datentyp des berechneten Ausdrucks kann implizit in den Datentyp der zuzuweisenden Variablen konvertiert werden.

Wenn Sie also einen Ausdruck einer Variablen zuweisen wollen, achten Sie darauf, dass die Variable von einem genügend großen Datentyp ist, oder führen Sie eine explizite Datentypkonvertierung für den zu großen Teil des Ausdrucks durch (siehe Funktionshandbuch *SIMOTION Basisfunktionen*).

Beispiel siehe .

Geben Sie ggf. bei Zahlen den Datentyp explizit an (z. B. UINT#127, wenn die Zahl 127 vom Datentyp UINT statt USINT sein soll).

Siehe auch

Funktionen zur Konvertierung von numerischen Datentypen und Bit-Datentypen (Seite 375)

12.1.3 Start von Funktionen in zyklischen Tasks immer abfragen

TO-Funktionen, z. B. Funktionen zum Positionieren von Achsen, sollten Sie in zyklischen Tasks nur dann absetzen, wenn sie nicht bereits laufen. Wenn Sie dies nicht tun, werden die Befehle in Ihrem Programm bei jedem neuen Zyklusdurchlauf erneut abgesetzt.

Den Start einer TO-Funktion in zyklischen Tasks können Sie von einer Bedingung abhängig machen, z. B. vom Inhalt einer Hilfsvariablen, die bei Befehlsausführung gesetzt wird.

Tabelle 12- 1 Beispiel für den korrekten Start einer TO-Funktion in einer zyklischen Task

```
//...
IF myStart = 0 THEN       // Wenn Hilfsvariable noch nicht gesetzt
    myStart := 1;         // Hilfsvar. setzen (Funktion gestartet)
    myCommandID := _getCommandId ();
    myFC := _pos (axis := myAxis, // Funktion ausführen
             position        := position_1,
             nextCommand     := IMMEDIATELY,
             commandID       := myCommandID);
END_IF;
//...
IF myAxis.positioningState.actualPosition = position_1 THEN
    myStart := 0;         // Hilfsvariable zurücksetzen, wenn
                          // Funktionsausführung erwünscht.
END_IF;
//...
```

12.1.4 Wartezeiten in zyklischen Tasks

Wenn Sie für Systemfunktionen in zyklischen Tasks, z. B. für den Befehl *_pos*, die Befehlsweiterschaltung an Bedingungen knüpfen, kann dies zu einer Zykluszeitüberschreitung und damit zum Stopp der CPU führen.

Dies kann bei allen Systemfunktionen auftreten, bei denen der Parameter *nextCommand* einen Wert ungleich IMMEDIATELY annimmt, z. B. den Wert WHEN_MOTION_DONE.

Die Zykluszeitüberschreitung tritt auf, wenn die in SIMOTION SCOUT projektierte Zykluszeit durch eine Weiterschaltbedingung aber auch durch programmierte Wartezeiten, z. B. mittels *_waitTime*, überschritten wird.

Hinweis

Sie sollten in zyklischen Tasks **keine Befehle mit Wartezeiten**, z. B. *_waitTime* verwenden.

Verwenden Sie für Systemfunktionen in zyklischen Tasks nur den Eingabeparameter **nextCommand := IMMEDIATELY**.

12.1.5 Den Parameter commandId richtig verwenden

Alle TO-Befehle müssen einen Parameter zur Befehlsidentifikation enthalten, siehe **Eingangsparameter der Technologie-Funktionen.**

Vor dem Aufruf des entsprechenden Befehls können Sie mit dem Befehl *_getCommandId* eine projektweit eindeutige Befehls-ID holen. Speichern Sie die Befehls-ID in eine lokale Variable und verwenden Sie diese als Parameter im TO-Befehl, oder verwenden Sie alternativ als Parameter direkt den Funktionsaufruf in *commandId:=_getCommandId()*.

Diese eindeutige Befehls-ID müssen Sie zur Statusabfrage des Motion-Befehls verwenden, z. B. wenn Sie mit *_getStateOfAxisCommand* den Status einer abgesetzten Positionierbewegung abfragen wollen. Nur anhand der Befehls-ID kann das System den Motion-Befehl eindeutig identifizieren!

Tabelle 12- 2 Beispiel für den Einsatz einer TO-Funktion mit Befehlsidentifikation

```
//...
VAR
    myCommandID                 : commandIdType;
    myState                     : StructRetCommandState;
END_VAR
//...
myCommandID := _getCommandId ();
// Eindeutige ID speichern
myFC := _pos (axis := myAxis,
// Funktion mit ID ausführen
             position           := position_1,
             nextCommand        := IMMEDIATELY,
             commandID          := myCommandID);
myState := _getStateOfAxisCommand (axis:=myAxis,
             commandID          := myCommandID);
// Statusabfrage
IF myState.commandIdState = WAITING_FOR_SYNC_START THEN
    ;
//...
END_IF;
//...
```

Siehe auch

Funktionsparameter der Technologie-Funktionen (Seite 118)

Funktion _getCommandId (Seite 438)

Funktion _getSyncCommandId (Seite 439)

12.1.6 Fehler eingrenzen (ST Programme)

Folgende Fehlermeldung kann beim Übersetzen auftreten:

Fehler 7000, 7010, 7011 oder 7014 in Zeile ...

Ein Syntax-Fehler ist aufgetreten. Mögliche Ursachen sind:

- *nicht korrekt abgeschlossenen Kontrollstrukturen (z. B. END_IF vergessen),*
- *nicht mit ; abgeschlossene Anweisungen,*
- *fehlende Klammern.*

Überprüfen Sie zuerst in der angegebenen Zeile, ob dieser Fehler tatsächlich aufgetreten ist, d. h. ob Sie eine Kontrollstruktur nicht abgeschlossen haben, die Zeile nicht mit einem Semikolon beendet haben oder eine Klammer fehlt.

Wenn Sie keinen der angegebenen Fehler finden, müssen Sie den Fehler eingrenzen:

1. Kommentieren Sie das Programm vor der Zeile mit der Fehlermeldung blockweise aus, d. h. schließen Sie den gewählten Abschnitt zwischen dem Zeichenpaar (* und *) ein. Dabei darf die Zeile mit der angezeigten Fehlermeldung nicht auskommentiert werden.
2. Übersetzen Sie das Programm neu.
3. Wenn nach Schritt 1 und 2 immer noch eine Fehlermeldung erscheint, haben Sie den auskommentierten Teil zu klein gewählt. Vergrößern Sie ihn, bis die Fehlermeldung verschwindet.
4. Wenn keine Fehlermeldung mehr erscheint, ist der Fehler im auskommentierten Teil. Nun können Sie den auskommentierten Teil zeilenweise verkleinern und das Programm neu übersetzen, bis die Fehlermeldung erneut erscheint. Die zuletzt frei gegebene Zeile ist die Zeile mit der Fehlermeldung.

Hinweis

Eine Auflistung aller Fehlermeldungen des Compilers finden Sie im Anhang des ST Programmierhandbuches.

12.1.7 Fehler beim Download

Wenn beim Download Ihres Programms eine Fehlermeldung auftritt, wird das Protokoll in der Detailanzeige des SIMOTION SCOUT angehalten. Sehen Sie im Fehlerprotokoll nach der Fehlerursache. Prüfen Sie Ihre Hardwarekonfiguration oder das Programm, beispielsweise auf Adressen, die nicht vorhanden sind.

Näheres zur Hardwarekonfiguration und zu den Adressierungen finden Sie im Projektierungshandbuch SIMOTION SCOUT.

Fehlermeldungen beim Download, welche die Abkürzung UPP (UserProgramProcessing) enthalten treten beim Abarbeiten eines Anwenderprogramms auf, z. B. falls Änderungen im RUN nicht durchgeführt werden können.

12.1.8 CPU geht nicht in RUN

Wenn die CPU gleich nach dem Start Ihrer Programme wieder in den Betriebszustand STOP wechselt, überprüfen Sie die Gerätediagnose und das Alarmfenster im SCOUT. Im Diagnosepuffer werden die STOP-Ursachen eingetragen. Im Alarmfenster werden die Alarme der Technologieobjekte angezeigt, die auch einen CPU-STOP verursachen können.

12.1.9 CPU geht in STOP

Beschreibung

Wenn die CPU in den Betriebszustand STOP wechselt, überprüfen Sie die Gerätediagnose und das Alarmfenster im SCOUT. Im Diagnosepuffer werden die STOP-Ursachen eingetragen. Im Alarmfenster werden die Alarme der Technologieobjekte angezeigt, die auch einen CPU-STOP verursachen können.

Mögliche Gründe, warum eine CPU nach STOP gehen kann, sind:

- Fehlerhafter direkter I/O Zugriff
- Konfigdaten- oder Systenvariablen-Zugriff auf TO, wenn im RESTART (ab V4.1 sind jedoch Ersatzwerte für Systemvariablen möglich, siehe Systemvariablen (Seite 135)).
- Fehlende PeripheralFaultTask (bei fehlerhaftem Zugriff auf Prozessabbild)
- Fehlende TechnologicalFaultTask (bei Fehler am Technologischen Objekt)
- Verarbeitungsfehler in Programmen (bzw. fehlende ExecutionFaultTask)
- Überwachung Zeitüberlauf der IPO-/ServoTask
- Überwachung Zeitüberlauf der BackgroundTask
- Überwachung Zeitüberlauf einer TimerInterruptTask
- Lebenszeichenüberwachung Simotion - Antrieb
- TO Alarme mit STOP-Reaktion (z. B. 20001).
- Betriebszustand über Systemvariable setzen

Betriebszustand anzeigen und ändern

Mit der Geräte-Systemvariable *modeOfOperation* können Sie sich den aktuellen Betriebszustand anzeigen lassen bzw. ändern.

Syntaxbeispiel

```
modeOfOperation  :EnumDeviceModeOfOperation  //lesbar, schreibbar, sofort
wirksam
EnumDeviceModeOfOperation:
    [
     _STOP I
     _RUN
    ]
```

Sie können damit z. B. von einem lokalen HMI durch Schreiben der Geräte-Systemvariable eine in STOP gegangene SIMOTION CPU wieder in RUN schalten.

Detaillierte Informationen über die Systemvariable *modeOfOperation* finden Sie im Handbuch *Systemfunktionen/-variablen Geräte* oder in der Online Hilfe.

Siehe auch

Verarbeitungsfehler in Programmen (Seite 146)

Fehler bei Zugriffen auf Systemvariablen und Konfigurationsdaten sowie auf I/O-Variablen für Direktzugriff (Seite 149)

SystemInterruptTasks (Seite 224)

Fehlermöglichkeiten bei Technologieobjekten (Seite 169)

12.1.10 Systemtakte überprüfen und einstellen

Eine häufige Ursache, dass das SIMOTION Gerät in den Betriebszustand STOP wechselt, sind falsch eingestellte Systemtakte (Servo-Takte, Interpolator-Takte, PWM-Takt) sein.

Überprüfen Sie die Laufzeiten der SynchronousTasks (ServoSynchronousTask, ServoSynchronousTask_fast, IPOsynchronousTask, IPOsynchronousTask_fast, IPOsynchronousTask_2, Tasks für das Technologiepaket TControl). Eventuell benötigen die Anwender- bzw. die Systemprogramme zu den einzelnen Tasks mehr Zeit, als in den Systemtakten im SIMOTION SCOUT eingestellt ist.

Versuchen Sie außerdem, die Laufzeit der SynchronousTasks zu minimieren. Verlagern Sie Programme möglichst in MotionTasks; teilen Sie ihre Programme ggf. entsprechend auf.

Achten Sie auf ein ganzzahliges Verhältnis zwischen den einzelnen Tasks. Andernfalls werden niederpriore SynchronousTasks in nicht periodischen Zeitabständen gestartet.

Deaktivieren Sie nicht benötigte Tasks.

Hinweis

Die System-Tasks zum Technologiepaket TControl können Sie im Fenster zur Einstellung der Systemtakte abschalten.

Wie Sie die Gerätediagnose überprüfen, das Alarmfenster interpretieren und Ihre Systemtakte überprüfen/ändern, können Sie in der Online-Hilfe nachlesen.

Zur Überprüfung der Laufzeiten stehen Systemfunktionen und ein Task-Trace zur Verfügung. Über den Task-Trace kann der Ablauf der einzelnen Tasks und User Events (per Programmbefehl erzeugt) grafisch dargestellt werden, siehe Funktionshandbuch Task Trace.

Siehe auch Taktsynchrone Datenbearbeitung (Seite 279) , Funktionen zur Meldungsprogrammierung (AlarmS) (Seite 350)

12.1.11 REAL- oder LREAL-Größen vergleichen

Wenn Sie REAL-Größen oder LREAL-Größen (auch entsprechende Systemvariablen, z. B. Achsposition) miteinander vergleichen, sollten Sie nie "=" verwenden. Aufgrund unterschiedlicher interner Zahlendarstellung sind die miteinander zu vergleichenden Zahlen nie identisch. Werten Sie stattdessen z. B. die Fahrrichtung aus und verwenden Sie ">" oder "<" bzw. die Systemvariable für "Positionsfenster erreicht".

12.1.12 Sequentielle Task ist unterbrochen

Beschreibung

Bei sequentiellen Tasks (MotionTasks) besteht die Möglichkeit, dass diese Tasks verschiedene Zustände annehmen, unter anderem auch TASK_STATE_WAITING. Der Status kann in der Diagnoseanzeige der CPU ausgelesen werden.

Wenn Sie nicht wissen, weshalb die Task wartet, müssen Sie aufwändig die Programmstelle suchen, an der die Task auf die Bedingung wartet.

Zur Suche der Stellen verwenden Sie folgende Funktionen:

- Funktion Programm-Durchlauf anzeigen
- Codestelle anzeigen (z. B. Zeile einer ST-Quelle), die eine MotionTask durchläuft

Detaillierte Informationen finden Sie im *ST Programmierhandbuch* im Abschnitt *Programm-Durchlauf*.

12.1.13 Bereichsüberläufe beachten

Bereichsüberläufe, d. h. die Überschreitung der Bereichsgrenzen für einen Datentyp, werden vom Compiler nicht gemeldet. Wenn Sie also Rechenoperationen mit Variablen durchführen, sollten Sie immer mögliche Bereichsüberläufe abfragen.

Tabelle 12- 3 Beispiel für die Abfrage nach Bereichsüberlauf

```
PROGRAM myRange
    VAR
        a,b  : SINT := 100;
        c    : SINT;
    END_VAR

    c := a + b;
    // Wenn c außerhalb Bereich, dann verlasse Programm.
    IF (a > 0) AND (c < a) AND (c < b) OR
       (a < 0) AND (c > a) AND (c > b) THEN
       // Bereichsüberlauf
       RETURN;
    ELSE
```

```
        ; // OK
    END_IF;
END_PROGRAM
```

12.1.14 Größe des Lokaldatenstacks einstellen

Bei der Konfiguration des Ablaufsystems stellen Sie für jede Task die Größe des reservierten Lokaldatenstacks ein. Eine Information, welchen Speicherbedarf ein Programm mit allen aufgerufenen POE (FC und FB) auf dem Stack hat, erhalten Sie mit der Funktion *Programmstruktur* (siehe Programmstruktur im ST Programmierhandbuch). Berücksichtigen Sie bei der Konfiguration genügend Reserven. So kann z. B. während des Downloads im RUN temporär zusätzlicher Speicher auf dem Lokaldatenstack benötigt werden.

Wenn der reservierte Lokaldatenstack den verfügbaren Speicherplatz im RAM überschreitet, wird der Download mit Fehlermeldung abgebrochen.

Wenn der reservierte Lokaldatenstack zur Ausführung der Programme nicht ausreicht, wird das Programm bei der Ausführung abgebrochen.

Überprüfen Sie im Fehlerfall:

- in der Gerätediagnose die Auslastung des RAM
- die Quellen (Programme usw.) z. B. anhand der Querverweisliste auf große Datenstrukturen (Arrays, Strukturen)
- die Größe des Lokaldatenstacks der Tasks.

Siehe auch

Festlegungen beim Konfigurieren (Seite 314)

12.1.15 Fehlerquellen bei Multitasking

Hauptfehlerquellen beim Multitasking sind:

- Verwendung von FB-Instanzen in unterschiedlichen Tasks
- Verwendung von globalen Variablen in unterschiedlichen Tasks

Durch das Verwenden von FB-Instanzen und globalen Variablen in unterschiedlichen Tasks ist eine korrekte Bearbeitung dieser Daten nicht sicher gestellt. Bei Verwendung von globalen Variablen in unterschiedlichen Tasks kann es z. B. vorkommen, dass sich der Wert durch einen Taskwechsel und durch die Bearbeitung in einer anderen Task ungewollt ändert.

12.2 Effizient programmieren

12.2.1 Effizient Programmieren - Übersicht

Bei vielen Steuerungssystemen bzw. den zugehörigen Programmierumgebungen besteht prinzipell die folgende Diskrepanz:

- Gut strukturierte und übersichtliche Anwenderprogramme mit besonderem Wert auf Modifizierbarkeit und Erweiterbarkeit verhalten sich bezüglich der Laufzeit nicht optimal.
- Auf der anderen Seite sind laufzeitoptimierte Programme schwer erweiterbar oder modifzierbar.

Die folgenden Beschreibungen geben Hinweise zur laufzeitoptimierenden und zur änderungsoptimierenden Programmierung. Je nach Aufgabe bzw. bei lokalen Optimierungen sollten Sie den Schwerpunkt auf eine der beiden Möglichkeiten legen.

12.2.2 Laufzeitoptimierte Programmierung

12.2.2.1 Laufzeitoptimierende Programmierung

In der folgenden Beschreibung finden Sie Hinweise zur laufzeitoptimierenden Programmierung. Bitte beachten Sie, dass die Hinweise zur laufzeitoptimierenden Programmierung oft nicht im Einklang mit den Regeln zur strukturierten Programmierung sind.

12.2.2.2 Zugriff auf Ein- und Ausgänge optimieren

Der Zugriff auf das Prozessabbild der zyklischen Tasks ist wesentlich schneller als der Direktzugriff auf Ein- oder Ausgänge (siehe Direktzugriff und Prozessabbild der zyklischen Tasks im ST Programmierhandbuch). Ordnen Sie deshalb eine I/O-Variable dem Prozessabbild derjenigen Task zu, in der die Variable verwendet wird. Neben dem schnelleren Zugriff ist ein weiterer Vorteil, dass die I/O-Variable während der gesamten Laufzeit der Task konsistent ist.

12.2.2.3 Zugriff auf Systemvariablen optimieren

Der Zugriff auf Systemvariablen (siehe Systemvariablen (Seite 135)) ist deutlich langsamer als der Zugriffe auf Variablen, die im dynamischen Speicher abgelegt werden (lokale Variablen, nicht remanente Unit-Variablen).

In schnellen zyklischen Tasks (z. B. SynchronousTasks) sind deshalb nur wenige direkte Zugriffe auf Systemvariablen möglich. Deshalb empfiehlt es sich bei vielen Zugriffen auf eine Systemvariable: Kopieren Sie die gesamte Struktur der Systemvariable zu Beginn eines Zyklus (Programm in der zyklischen Task) in eine lokale Variable des entsprechenden Systemdatentyps. Greifen Sie während des Programmzyklus auf diese Variable zu.

Die Datentypen für die Deklaration der lokalen Variablen finden Sie in den Listenhandbüchern der SIMOTION Technologieobjekte.

12.2.2.4 Variablen optimal deklarieren

Ordnen Sie die Variablen innerhalb eines Deklarationsblocks (z. B. VAR/END_VAR) in aufsteigender Größe an. Dadurch nutzen Sie den Speicherplatz optimal aus.

Initialisieren Sie Variablen mit Werten ungleich 0 nur, wenn es nötig ist. Die Initialisierung beim Start einer Task oder einer POE benötigt Zeit. Dies wirkt sich besonders aus bei temporären Variablen sowie bei Variablen von Programmen, die sequentiellen Tasks zugeordnet sind.

12.2.2.5 Zugriff auf Parameter der Funktionsbausteine optimieren

Wenn Sie Funktionsbausteine (FB) erstellen, die Werte bearbeiten sollen, können Sie dies auf zweierlei Weise tun. Sie können die entsprechenden Variablen im FB mit Eingangsparametern mittels VAR_INPUT besetzen, dort bearbeiten und mit Ausgangsparametern mittels VAR_OUTPUT für die Weitergabe kennzeichnen.

Wenn ein großes Datenvolumen an den FB zu übergeben ist, kann das Verwenden von Durchgangsparametern (VAR_IN_OUT) schneller sein als das Verwenden von Ein- und Ausgangsparametern (VAR_INPUT und VAR_OUTPUT). Die Übergabe der Parameter erfolgt schneller, da Kopiervorgänge entfallen, jedoch ist der Zugriff auf die Variable vom FB aus unter Umständen langsamer.

ACHTUNG
Beachten Sie, wenn Sie mit Durchgangsparametern auf Unit-Variablen oder geräteglobale Variablen zugreifen: Andere Tasks können gleichzeitig auf diese Variablen zugreifen.

12.2.2.6 Programmstruktur optimieren

Achten Sie auf eine übersichtliche Gliederung ihrer ST-Quellen und der darin enthaltenen POE. Modularisieren Sie die Quellen jedoch nicht zu stark, da der Zugriff auf Funktionen und Variablen importierter Units etwas mehr Zeit benötigt als inner halb einer Unit.

12.2.2.7 Ablaufsystem optimieren

Ordnen Sie der IPOsynchronousTask nur ein einziges Programm zu. Sie mindern dadurch die Gefahr einer Laufzeitüberschreitung.

Verwenden Sie Funktionen, die synchron abgearbeitet werden, nur in MotionTasks. (Bei der synchronen Bearbeitung wird der nächste Befehl erst ausgeführt, wenn die Bearbeitung des anstehenden Befehls eine Bedingung erfüllt, z. B. beendet ist.)

Achten Sie darauf, dass nicht zu viele MotionTasks gleichzeitig aktiv sind.

Gehen Sie zur effizienteren Programmierung zyklischer Tasks (vor allem der BackgroundTask) zur schrittkettengesteuerten Programmierung über. D.h. über Fallunterscheidungen steuern Sie den Programmfluss bewusst und durchlaufen nur die Codeanteile, welche im aktuellen Zustand relevant sind (z.B. mittels CASE-Anweisung).

Außerdem empfiehlt sich eine flanken- anstatt einer pegelgetriggerten Programmierung.

Durchlaufen Sie den relevanten Code nicht zyklisch (wenn der Pegel der Bedingung TRUE ist), sondern nur einmal. Dies erfolgt durch Abfrage auf steigende Flanke der erfüllten Bedingung (Neuzustand der Bedingung ist TRUE, der alte Zustand hingegen war FALSE).

12.2.2.8 Verwendung von USEPACKAGE für mehrere Technologiepakete

Syntax für die Verwendung von "USEPACKAGE" für mehrere Technologiepakete

Wenn Sie mehrere Technologiepakete gleichzeitig verwenden möchten, können Sie diese mit folgender Syntax auf einmal auswählen. Siehe auch .

Tabelle 12- 4 Formale Beschreibung

```
usepackagestatement ::= 'USEPACKAGE' packagelist ';'
packagelist         ::= packageentry {',' packageentry }
packageentry        ::= simplepackname | namespacepackname
simplepackname      ::= packagename
namespacepackname   ::= packagename 'AS' namespaceident
//packagename und namespaceident müssen gültige Bezeichner sein
```

12.2.3 Änderungsoptimierte Programmierung

12.2.3.1 Änderungsoptimierende Programmierung

In der folgenden Beschreibung finden Sie Hinweise zur änderungsoptimierenden Programmierung.

Information zum Download im RUN finden Sie unter Download im RUN von geänderten Quellen (Seite 535) .

12.2.3.2 Hinweise zum Programmentwurf

Beschreibung

Nützliche Hinweise z. B. zu:

- wie strukturieren Sie das Programm am besten
- wie verwenden Sie am besten die unterschiedlichen Tasks (MotionTask, BackgroundTask, IPO-Task etc.)
- wie verteilen Sie am besten Ihr Programm auf die unterschiedlichen Tasks?

finden Sie im Applikationsleitfaden für SIMOTION.

Den Applikationsleitfaden für SIMOTION finden Sie in den **SIMOTION Utilities & Applications**, die im Lieferumfang von SIMOTION SCOUT enthalten sind (unter **Dokumentation > Applikationsleitfaden für SIMOTION**).

12.2.3.3 Generische Programmierung

Beschreibung

Programmieren Sie Achsen in Arrays und Schleifen. Dieses bringt vor allem Vorteile bei Erweiterungen um neue Achsen, Fehlerkorrekturen (nur einmal in der Schleife und nicht für jede Achse separat) und führt zu kompakteren Code. Die Lesbarkeit wird erheblich erhöht.

Bei MCC- und KOP/FUP-Quellen kann die Programminitialisierung auch über eine Pragma-Zeile in den Deklarationstabellen aktiviert werden (Pragma "BlockInit_OnChange").

Informationen zu Achsarrays finden Sie in den SIMOTION Utilities & Applications unter **Dokumentation > Applikationsleitfaden für SIMOTION** im Abschnitt **Generische Programmierung**, die im Lieferumfang von SIMOTION SCOUT enthalten sind.

12.2.3.4 Deklaration remanenter Variablen in einer eigenen Unit

Deklarieren Sie alle remanenten Variablen im Interfaceabschnitt einer einzigen Unit. Dies hat folgenden Vorteil:

- Sie nutzen den beschränkten Speicherplatz optimal aus.
- Die Variablen werden nur initialisiert, wenn in dieser Unit der Interfaceabschnitt geändert wurde.
- Ab V4.1 können Sie auch mehrere Deklarationsblöcke verwenden, siehe Mehrere VAR_GLOBAL, VAR_GLOBAL RETAIN Blöcke verwenden (Seite 316) .

12.2.3.5 HMI-Variablen in einer eigenen Unit

Deklarieren Sie Variablen, die an HMI-Geräten exportiert werden (siehe Kopplung HMI (Human Machine Interface (Seite 484)). Bei größeren Projekten bzw. strikt abgegrenzten Softwaremodulen ist eine separate HMI-Unit je Modul sinnvoll.

- Ein erneuter Download des HMI-Projekts (z. B. WinCC flexible) ist nur erforderlich, wenn in dieser Unit der Interfaceabschnitt geändert wurde.
- Optional können Sie während der Inbetriebnahme den Konsistenz-Check auf eigene Verantwortung abschalten (**Extras > Einstellungen > Download**). Beachten Sie dabei, dass dann keine Prüfung auf Inkonsistenzen (z. B. auf gültige Hardwareadressen) erfolgt und somit auch unkontrollierte Prozesszugriffe stattfinden können.
- Sie können auch HMI-relevante Daten kennzeichnen, siehe HMI-relevante Daten kennzeichnen (Seite 319) .

Siehe auch Kopplung HMI (Human Machine Interface) (Seite 484) .

12.2.3.6 Verwendung Geräteglobaler Variablen versus Unitglobaler Variablen

Beschreibung

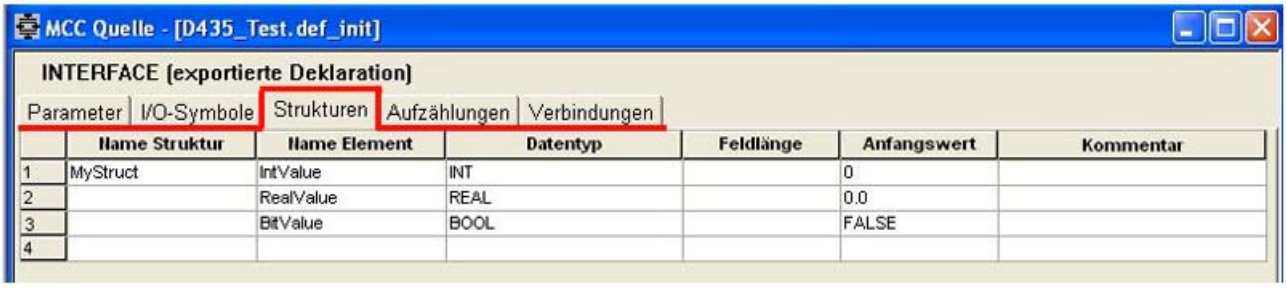
Die Verwendung von globalen Unit-Variablen in Quellen ist der Verwendung von Geräteglobalen Variablen (über den Projektnavigator) vorzuziehen.

Vorteile:

- Es können Variablen-Strukturen verwendet werden
- Initialisierung (Anfangswerte) der Variablen bei STOP-RUN-Übergang ist möglich (über Programm in StartupTask)
- Bei neu angelegten globalen Unit-Variablen ist auch ein Download im RUN möglich (in einem neuen Deklarationsblock)

Vorgehensweise

1. Definieren Sie unitglobale Variablen in einer Quelle im Interface-Abschnitt. Retainvariablen und HMI-Variablen sollten ebenfalls jeweils in einer separaten Quelle bzw. Deklarationsblock deklariert werden (siehe oben).
2. Ordnen Sie das Programm mit der Initialisierung der Variablen der StartupTask zu. Die Variablen werden im Anlauf auf einen definierten Anfangswert gesetzt.



	Name Struktur	Name Element	Datentyp	Feldlänge	Anfangswert	Kommentar
1	MyStruct	IntValue	INT		0	
2		RealValue	REAL		0.0	
3		BitValue	BOOL		FALSE	
4						

Bild 12-1 Beispiel der Deklaration von Strukturen in einer MCC-Quelle

	Name	Variablentyp	Datentyp	Feldlänge	Anfangswert	Kommentar
1	m	VAR_GLOBAL CONSTANT	INT		16	
2	BitArray	VAR_GLOBAL	BOOL	m	16(FALSE)	
3	IntArray	VAR_GLOBAL	INT	m	16(0)	
4	RealArray	VAR_GLOBAL	REAL	m	16(0.0)	
5	StructVar	VAR_GLOBAL	MyStruct			
6						

Bild 12-2 Beispiel der Deklaration von Parametern in einer MCC-Quelle

Tabelle 12- 5 Beispiel-Deklaration und Programm in einer ST-Quelle (Unit)

```
INTERFACE
    //global types
    TYPE
       MyStruct : STRUCT
                   Intvalue : INT ;
                   Realvalue: REAL;
                   Bitvalue : BOOL;
                  END_STRUCT
    END_TYPE
    //global constants
    VAR_GLOBAL CONSTANT
        n : INT := 0 ;
        m : INT := 15;
    END_VAR
    //global variables
    VAR_GLOBAL
        Bitarray : ARRAY [n..m] OF BOOL := [16 (FALSE)];
        Intarray : ARRAY [0..15] OF INT := [16 (0)    ];
        Realarray: ARRAY [0..15] OF REAL:= [16 (0.0)  ];
        StructVar: MyStruct;
    END_VAR
    //programms
    PROGRAM Init;
    //end of the interface
END_INTERFACE

IMPLEMENTATION
 PROGRAM Init
  ////////////////////////////////////////////////
  //initialisation of the variables during startup//
  ////////////////////////////////////////////////
  StructVar.Intvalue :=0;
  StructVar.Realvalue:=0;
  StructVar.Bitvalue :=FALSE;
 END_PROGRAM
END_IMPLEMENTATION
```

Das Programm wird dann der StartupTask zugeordnet.

Immer wenn unitglobale Variablen in einer anderen Quelle verwendet werden sollen, müssen die Quellen, welche die Deklaration enthält, verbunden werden (USES). Siehe auch **Verbindung zu anderen Programmquellen oder zu Bibliotheken** im MCC Programmierhandbuch oder **Verwendung der USES-Anweisung im Interface- oder Implementationsabschnitt einer importierenden Unit** im ST Programmierhandbuch.

12.2.3.7 Start und Zurücksetzen von MotionTasks an zentraler Stelle

Programmieren Sie wegen der Übersichtlichkeit das Starten und Zurücksetzen von MotionTasks zentral an einer Stelle.

12.2.3.8 Beispiel für einen Download von geänderten Quellen

Beispielprogramm für Download im RUN

Die folgende Grafik zeigt den Aufbau des Ablaufsystems eines Projektes, mit dem aufgezeigt werden soll, wann und unter welchen Bedingungen ein Download im RUN möglich ist, wenn Sie eine Variable hinzufügen oder ändern möchten. Im Ablaufsystem sehen Sie die Tasks, die wiederum verschiedene Units enthalten.

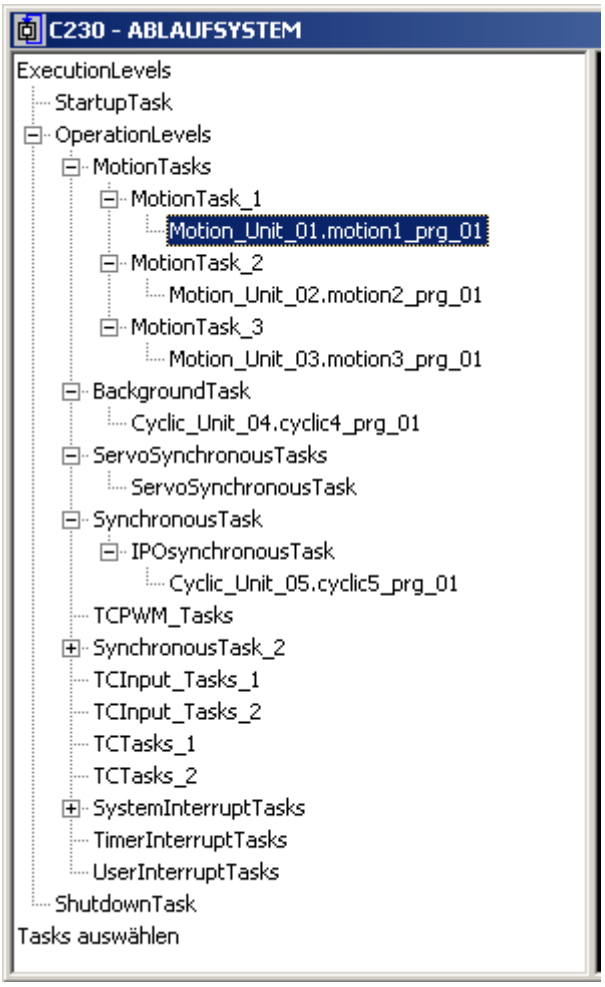


Bild 12-3 Aufbau des Ablaufsystems

Die Grafik *Download im Run* veranschaulicht, welche Programmeinheiten nacheinander aufgerufen werden. So ruft *Motion_Unit_01 FunctionBlock_01* auf, der wiederum die beiden Funktionen *Function_01* und *Function_02* aufruft.

Für die *Motion_Unit_01* und die *Cyclic_Unit_05* bestehen nur unter den in Download im RUN von geänderten Quellen (Seite 535) beschriebenen Randbedingungen.

Der rot hinterlegte Bereich markiert die Blöcke des Programms, bei denen bei einem Download im RUN Probleme auftreten können.

Motion_Unit_02 enthält eine While TRUE Schleife, von der aus wiederum FunctionBlock_02 und dann Function_02 aufgerufen werden (siehe Beispiel für eine While-Schleife am des Abschnitts). Da die Schleife einen Download im RUN verhindert (kein Einwechselzeitpunkt möglich), sind auch der in ihr aufgerufene Funktionsbaustein und die Funktion blockiert. Unter welchen Umständen dennoch ein Download im RUN möglich ist, finden Sie in den folgenden Tabellen beschrieben.

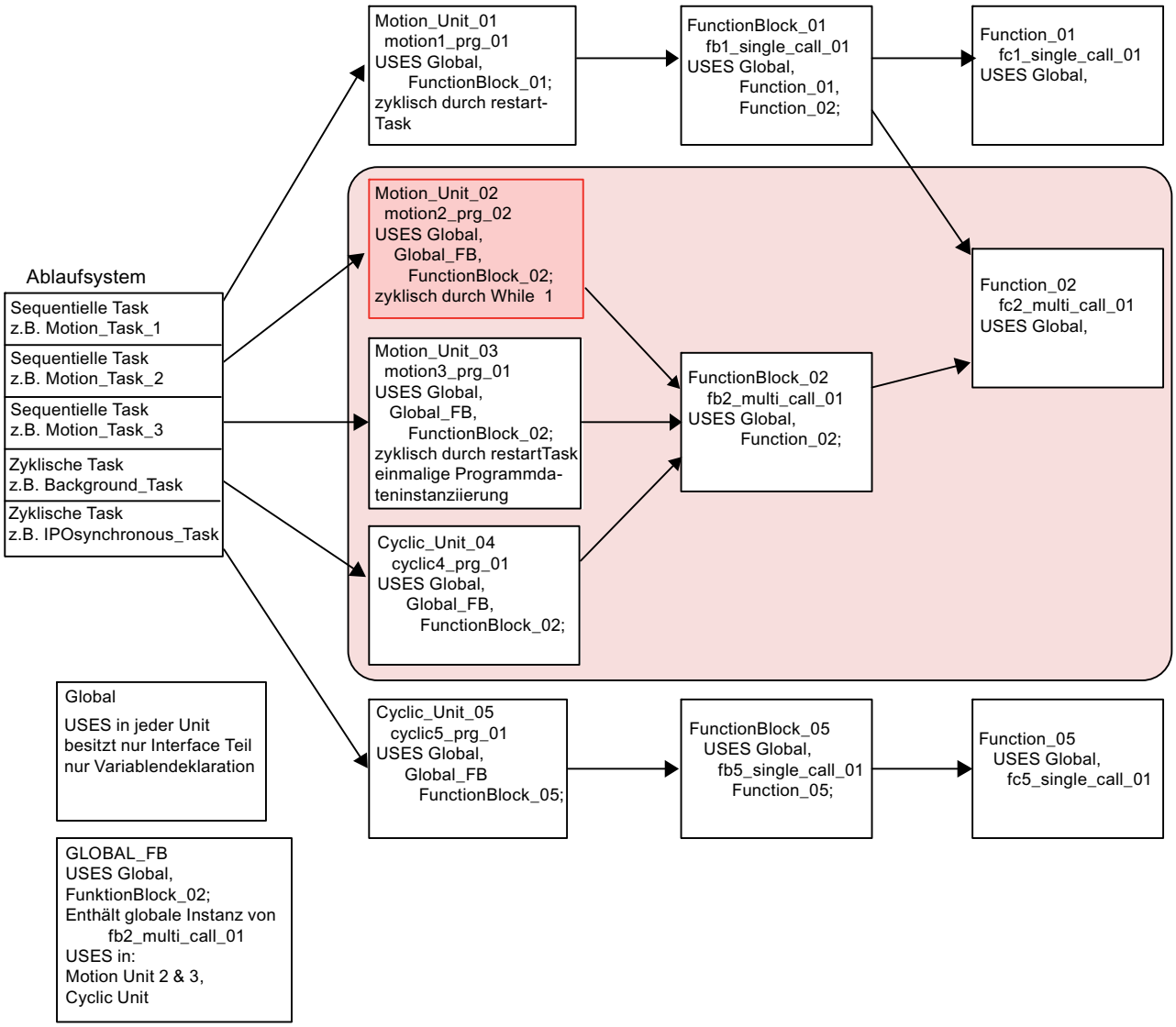


Bild 12-4 Download im Run

Mit SIMOTION V4.1.2 können Sie MotionTasks auch vom SCOUT aus steuern. Sie können also die *MotionUnit_02,* stoppen den Download durchführen und dann die *MotionUnit_02* wieder starten, siehe Steuern von Motion Tasks aus dem SCOUT (Seite 543).

Übersicht über die Möglichkeiten beim Download im RUN

Die folgenden Tabellen zeigen, welche Daten wo geändert werden können.

Änderungsort	I/O-Variable	
Änderungsort	Geräteglobale Variablen	

Änderungsort	Interface					
	Type	Var_Global	Var_Global Retain	Var_Global Constant	USES	USEPACKAGE
Motion_Unit_01	4) 2)	4)	4)	6)		
Motion_Unit_02	1)	1)	1)	6)	1)	1)
Motion_Unit_03	4) 2)	4)	4)	6)		
Cyclic_Unit_04	4) 2)	4)	4)	6)		
Cyclic_Unit_05	4) 2)	4)	4)	6)		
Function_Block_01	4) 2)	4)	4)	6)		
Function_Block_02	1)	1)	1)	6)	1)	1)
Function_Block_05	4) 2)	4)	4)	6)		
Function_01	4) 2)	4)	4)	6)		
Function_02	1)	1)	1)	6)	1)	1)
Function_05	4) 2)	4)	4)	6)		
Global	1)	1)	1)	6)	1)	1)
Global_FB		1)		6)	1)	1)

Änderungsort	Implementation					
	Type	Var_Global	Var_Global Retain	Var_Global Constant	USES	USEPACKAGE
Motion_Unit_01	4) 2)	4)	4)	6)		
Motion_Unit_02	1)	1)	1)	6)	1)	1)
Motion_Unit_03	4) 2)	4)	4)	6)		
Cyclic_Unit_04	4) 2)	4)	4)	6)		
Cyclic_Unit_05	4) 2)	4)	4)	6)		
Function_Block_01	4) 2)	4)	4)	6)		
Function_Block_02	1)	1)	1)	6)	1)	1)

Änderungsort	Implementation					
	Type	Var_Global	Var_Global Retain	Var_Global Constant	USES	USEPACKAGE
Function_Block_05	4) 2)	4)	4)	6)		
Function_01	4) 2)	4)	4)	6)		
Function_02	1)	1)	1)	6)	1)	1)
Function_05	4) 2)	4)	4)	6)		

Änderungsort	Programm					
		Type	Var	Var_Temp	Var_Constant	Codeänderung
Motion_Unit_01	motion1_prg_01					
Motion_Unit_02	motion2_prg_01	1)	1)	1)		1)
Motion_Unit_03	motion3_prg_01	2)7)	3)			
Cyclic_Unit_04	cyclic4_prg_01	2)5)	5)			
Cyclic_Unit_05	cyclic5_prg_01	2)7)	3)			

Änderungsort	FunctionBlock								
		Type	Var	Var_ Temp	Var Constant	Codeänderung	Var_Input	Var_In_Out	Var_Output
FunctionBlock_01	Fb1_single_call_01								
FunctionBlock_02	Fb2_multi_call_01	1)2)	1)7)		6)		1)7)	1)7)	1)7)
FunctionBlock_03	Fb5_single_call_01	2)7)	7)		6)		7)	7)	7)

Änderungsort	Function							
		Type	Var	Var Constant	Codeänderung	Var_Input	Var_In_Out	Rückgabewert
Function_01	Fc1_single_call_01							
Function_02	Fc2_multi_call_01					1)	1)	1)
Function_03	Fc5_single_call_01							

Legende zu den Tabellen

Fußnote	Beschreibung
1)	MotionTask_2 muss vor dem Einwechseln gestoppt werden: - Download wird durch eine Endlosschleife (WHILE TRUE) im Programm verhindert. - Gleiches Verhalten, wenn das Programm zu lange bei "WAITFORCONDITION wartet oder auf "_waitTime" steht, oder auf das Beenden synchroner Befehle wartet.
2)	Strukturänderungen verhalten sich bei Verwendung wie das Anlegen/Modifizieren einer Variablen.
3)	Pragma "BlockInit_OnChange" verwenden oder ab V4.2 eine entsprechende PRAGMA-Zeile in den Deklarationstabellen (MCC, KOP/FUP) einfügen.
4)	Zusätzliche Variable kann durch einen zusätzlichen Variablen Block, durch das Pragma "BlockInit_OnChange" bzw. ab V4.2 über eine entsprechende PRAGMA-Zeile in den Deklarationstabellen (MCC, KOP/FUP) hinzugefügt werden.
5)	Nur möglich, wenn die Auswahl "Programminstanz nur einmal anlegen" angewählt ist und das Pragma "BlockInit_OnChange" verwendet wird bzw. ab V4.2 eine entsprechende PRAGMA-Zeile in den Deklarationstabellen (MCC, KOP/FUP) einfügt ist.
6)	Änderbarkeit abhängig von der Verwendung: - Code geht immer - Feldlängen; Verhalten wie bei Anlegen/Modifizierung einer neuen Variable, möglich mit Pragma "BlockInit_OnChange" bzw. ab V4.2 über eine entsprechende PRAGMA-Zeile in den Deklarationstabellen (MCC, KOP/FUP. - Initialwert, geht immer, analog zu Code
7)	Neue Initialisierung notwendig, eventuell Pragma "BlockInit_OnChange" verwenden bzw. ab V4.2 eine entsprechende PRAGMA-Zeile in den Deklarationstabellen (MCC, KOP/FUP).
Units müssen immer komplett einwechselbar sein. Funktioniert ein Programm von mehreren (z. B. Interface, Implementation) nicht, wird das Einwechseln komplett verhindert.	
	Immer änderbar
	Bedingt änderbar mit Beschreibung, was zu beachten ist, siehe Fußnote.
	Nicht änderbar, Begründung, siehe Fußnote.

Hinweis

Das Compiler-Pragma "BlockInit_OnChange := True;" möglich (ab V4.1) bzw. ab V4.2 kann über eine entsprechende PRAGMA-Zeile in den Deklarationstabellen (MCC, KOP/FUP) eingestellt werden.

Siehe auch

Motion Tasks steuern (Seite 543)

Anhang A

A

A.1 Symbolische Konstanten

Die folgenden Tabellen zeigen Namen von reservierten Konstanten, die Sie nicht für individuelle Variablennamen verwenden dürfen.

Tabelle A- 1 Symbolische Konstanten der Taskstartinfo

Symbolische Konstante	Datentyp	Wert
_SC_ALARM_CONFIGURATION	UDINT	404
_SC_ARRAY_BOUND_ERROR_READ	UDINT	502
_SC_ARRAY_BOUND_ERROR_WRITE	UDINT	503
_SC_BACKGROUND_TIMER_OVERFLOW	UDINT	300
_SC_CYCLE_TIMER_OVERFLOW	UDINT	301
_SC_DEVICE_COMMAND	UDINT	401
_SC_DIAGNOSTIC_INTERRUPT	UDINT	201
_SC_DIVISION_BY_ZERO	UDINT	500
_SC_DP_CLOCK_DETECTED	UDINT	207
_SC_DP_SLAVE_NOT_SYNCHRONIZED	UDINT	210
_SC_DP_SLAVE_SYNCHRONIZED	UDINT	209
_SC_DP_SYNCHRONIZATION_LOST	UDINT	208
SC_DRIVE_OBJECT_FAULT	UDINT	217
_SC_DRIVE_OBJECT_ALARM	UDINT	218
_SC_EXCEPTION	UDINT	403
_SC_EXTERNAL_COMMAND	UDINT	402
_SC_IMAGE_UPDATE_FAILED	UDINT	204
_SC_IMAGE_UPDATE_OK	UDINT	206
_SC_INVALID_ADDRESS	DINT	–1
_SC_INVALID_FLOATING_POINT_OPERATION	UDINT	501
_SC_IO_MODULE_NOT_SYNCHRONIZED	UDINT	215
_SC_IO_MODULE_SYNCHRONIZED	UDINT	214
_SC_MODE_SELECTOR	UDINT	400
_SC_PC_INTERNAL_FAILURE	UDINT	205
_SC_PULL_PLUG_INTERRUPT	UDINT	216
_SC_PROCESS_INTERRUPT	UDINT	200
_SC_STATION_DISCONNECTED	UDINT	202
_SC_STATION_RECONNECTED	UDINT	203
_SC_TO_INSTANCE_NOT_EXISTENT	UDINT	506
_SC_VARIABLE_ACCESS_ERROR_READ	UDINT	504
_SC_VARIABLE_ACCESS_ERROR_WRITE	UDINT	505

Tabelle A- 2 Symbolische Konstanten der Taskstati

Symbolische Konstante	Datentyp.	Hex-Darstellung
TASK_STATE_INVALID	DWORD	16#0000
TASK_STATE_LOCKED	DWORD	16#0100
TASK_STATE_RUNNING	DWORD	16#0004
TASK_STATE_STOP_PENDING	DWORD	16#0001
TASK_STATE_STOPPED	DWORD	16#0002
TASK_STATE_SUSPENDED	DWORD	16#0020
TASK_STATE_WAIT_NEXT_CYCLE	DWORD	16#0040
TASK_STATE_WAIT_NEXT_INTERRUPT	DWORD	16#0080
TASK_STATE_WAITING	DWORD	16#0010

Tabelle A- 3 Symbolische Konstanten der Alarmmeldungen

Symbolische Konstante	Datentyp	Hex-Darstellung
ALARMS_ERROR	DWORD	16#8000
ALARMS_QSTATE	DWORD	16#0100
ALARMS_STATE	DWORD	16#0001
DSC_SVS_DEVICE_ALARMS_EVENT_ID_IN_USE	DWORD	16#0006
DSC_SVS_DEVICE_ALARMS_EVENT_ID_NOT_USED	DWORD	16#0005
DSC_SVS_DEVICE_ALARMS_ILLEGAL_EVENT_ID	DWORD	16#0001
DSC_SVS_DEVICE_ALARMS_INTERNAL_ERROR	DWORD	16#0009
DSC_SVS_DEVICE_ALARMS_IV_CALL	DWORD	16#0004
DSC_SVS_DEVICE_ALARMS_IV_FIRST_CALL	DWORD	16#0007
DSC_SVS_DEVICE_ALARMS_IV_SFC_TYP	DWORD	16#0008
DSC_SVS_DEVICE_ALARMS_LAST_ENTRY_USED	DWORD	16#0002
DSC_SVS_DEVICE_ALARMS_LAST_SIGNAL_USED	DWORD	16#0003
DSC_SVS_DEVICE_ALARMS_NO_ENTRY	DWORD	16#0010

Tabelle A- 4 Symbolische Konstanten für die Wertebereichsgrenzen elementarer Datentypen

Symbolische Konstante	Datentyp	Wert	Hex-Darstellung
SINT#MIN	SINT	-128	16#80
SINT#MAX	SINT	127	16#7F
INT#MIN	INT	-32768	16#8000
INT#MAX	INT	32767	16#7FFF
DINT#MIN	DINT	-2147483648	16#8000_0000
DINT#MAX	DINT	2147483647	16#7FFF_FFFF
USINT#MIN	USINT	0	16#00
USINT#MAX	USINT	255	16#FF
UINT#MIN	UINT	0	16#0000
UINT#MAX	UINT	65535	16#FFFF
UDINT#MIN	UDINT	0	16#0000_0000
UDINT#MAX	UDINT	4294967295	16#FFFF_FFFF
T#MIN TIME#MIN	TIME	T#0ms	16#0000_0000[1]
T#MAX TIME#MAX	TIME	T#49d_17h_2m_47s_295ms	16#FFFF_FFFF[1]
TOD#MIN TIME_OF_DAY#MIN	TOD	TOD#00:00:00.000	16#0000_0000[1]
TOD#MAX TIME_OF_DAY#MAX	TOD	TOD#23:59:59.999	16#0526_5BFF[1]
[1] Nur interne Darstellung			

Tabelle A- 5 Symbolische Konstanten für ungültige Werte ausgewählter Datentypen

Symbolische Konstante	Datentyp	Bedeutung
STRUCTALARMID#NIL	StructAlarmId	Ungültige AlarmId
STRUCTTASKID#NIL	StructTaskId	Ungültige TaskId
TO#NIL	ANYOBJECT	Ungültiges Technologieobjekt

A.2 Bezeichner mit definierter Bedeutung in SIMOTION

In diesem Kapitel sind alphabetisch alle Bezeichner aufgelistet, die in SIMOTION eine vordefinierte Bedeutung haben. Die Liste enthält:

- Die reservierten und geschützten Bezeichner der Programmiersprache SIMOTION ST (Structured Text).
- Die reservierten und geschützten Bezeichner anderer Programmiersprachen
- Die allgemeinen Standardfunktionen und Standardfunktionsbausteine mit den zugehörigen Datentypen
- Die Funktionen zur Tasksteuerung, Laufzeitmessung und Meldungsprogrammierung (siehe Programmieren Ablaufsystem/Tasks/Systemtakte (Seite 311)) mit den zugehörigen Datentypen
- Die Systemfunktionen und Systemvariablen der SIMOTION Geräte mit den zugehörigen Datentypen
- Die Datentypen der Technologieobjekte
- Die Systemfunktionen, Systemvariablen (und Konfigurationsdaten) der Technologiepakete

Die Reaktion bei Verwendung dieser Bezeichner ist unterschiedlich, z. B:

- Compilerfehler bei reservierten Bezeichnern.
- Mögliche Verdeckung und resultierende Nichterreichbarkeit der Bezeichner.

  Der Compiler kann unter bestimmten Umständen keine Warnung absetzen, wenn z. B. das zugehörige Technologiepaket nicht importiert ist.

Symbole

_2D

_3D

_abortAllReadWriteDriveParameterJobs

_abortReadWriteRecordJobs

_activateConfiguration

_activateDpSlave

_activateDpSlaveAddress

_activateNameOfStation

_activateTo

_adaptAxisConfigData

_adaptExternalEncoderConfigData

_additionObjectType

_addPointToCam

_addPolynomialSegmentToCam

_addSegmentToCam
_addSegmentToCam_V2_0
_alarmS
_alarmSc
_alarmScId
_alarmSId
_alarmSq
_alarmSqId
_allocateTokenToVariableName
_AND
_BOOL
_bufferActuatorCommandId
_bufferAdditionObjectCommandId
_bufferAxisCommandId
_bufferAxisCommandId_V3_1
_bufferCamCommandId
_bufferCamTrackCommandId
_bufferControllerObjectCommandId
_bufferExternalEncoderCommandId
_bufferExternalEncoderCommandId_V3_1
_bufferFixedGearCommandId
_bufferFollowingObjectCommandId
_bufferFollowingObjectCommandId_V3_1
_bufferFormulaObjectCommandId
_bufferMeasuringInputCommandId
_bufferOutputCamCommandId
_bufferPathObjectCommandId
_bufferSensorCommandId
_BYTE_FROM_8BOOL
_BYTE_TO_8BOOL
_calculateTControllerParameter
_camTrackType
_cancelAxisCommand
_cancelExternalEncoderCommand
_cancelFollowingObjectCommand

_cancelPathObjectCommand
_changeEnableModeOfAdditionObjectIn
_changeEnableModeOfControllerObjectIn
_changeEnableModeOfFormulaObjectIn
_changeEnableOfFormula
_changeOperationMode
_checkEqualTask
_checkExistingUnitDataSet
_configurationManagement
_continue
_continue_V3_1
_continuePath
_continuePath_V4_2
_controllerObjectType
_copyTControllerShadow
_cpuData
_cpuDataRW
_DATE
_DATE_AND_TIME
_deactivateDpSlave
_deactivateTo
_deallocateTokenToVariableName
_defineFormula
_deleteAllUnitDataSets
_deleteUnitDataSet
_DINT
_disableAdditionObjectIn
_disableAxis
_disableAxis_V2_0
_disableAxis_V3_1
_disableAxisAdditiveTorque
_disableAxisInterface
_disableAxisSimulation
_disableAxisTorqueLimitNegative
_disableAxisTorqueLimitPositive

_disableCamming
_disableCamming_V3_0
_disableCamTrack
_disableCamTrackSimulation
_disableCommandToActual
_disableControllerObject
_disableControllerObjectIn
_disableExternalEncoder
_disableExternalEncoderSimulation
_disableFixedGearing
_disableFixedGearMotionIn
_disableFollowingObjectSimulation
_disableFollowingObjectSimulation_V3_1
_disableForceLimiting
_disableFormula
_disableFormulaObjectIn
_disableGearing
_disableGearing_V3_0
_disableMaster_V3_0
_disableMeasuringInput
_disableMeasuringInputSimulation
_disableMonitoringOfEncoderDifference
_disableMovingToEndStop
_disableOutputCam
_disableOutputCamSimulation
_disablePathObjectSimulation
_disableQFAxis
_disableQFAxis_V3_0
_disableQFAxis_V3_1
_disableScheduler
_disableSensor
_disableTorqueLimiting
_disableVelocityGearing
_disableVelocityLimiting
_disableVelocityLimiting_V4_3

_driveStates
_DWORD_FROM_2WORD
_DWORD_FROM_4BYTE
_DWORD_TO_2WORD
_DWORD_TO_4BYTE
_enableAdditionObjectIn
_enableAxis
_enableAxis_V2_0
_enableAxis_V3_1
_enableAxis_V3_2
_enableAxisAdditiveTorque
_enableAxisInterface
_enableAxisSimulation
_enableAxisTorqueLimitNegative
_enableAxisTorqueLimitPositive
_enableCamming
_enableCamming_V3_0
_enableCamTrack
_enableCamTrackSimulation
_enableCommandToActual
_enableControllerObject
_enableControllerObjectIn
_enableDistributedMotionDelayValueCalculation
_enableDpInterfaceSynchronizationMode
_enableExternalEncoder
_enableExternalEncoderSimulation
_enableFixedGearing
_enableFixedGearMotionIn
_enableFollowingObjectSimulation
_enableFollowingObjectSimulation_V3_1
_enableForceControlByCondition
_enableForceControlByCondition_V2_1
_enableForceControlByCondition_V3_1
_enableForceControlByCondition_V4_0
_enableForceLimitingByCondition

_enableForceLimitingByCondition_V4_0
_enableForceLimitingValue
_enableForceLimitingValue_V3_1
_enableFormula
_enableFormulaObjectIn
_enableGearing
_enableGearing_V3_0
_enableGearing_V3_1
_enableMaster_V3_0
_enableMeasuringInput
_enableMeasuringInputCyclic
_enableMeasuringInputCyclic_V3_2
_enableMeasuringInputSimulation
_enableMonitoringOfEncoderDifference
_enableMotionInPositionLockedForceLimitingProfile
_enableMotionInPositionLockedVelocityLimitingProfile
_enableMotionInPositionLockedVelocityLimitingProfile_V4_0
_enableMotionInPositionLockedVelocityLimitingProfile_V4_3
_enableMovingToEndStop
_enableMovingToEndStop_V2_0
_enableOutputCam
_enableOutputCamSimulation
_enablePathObjectSimulation
_enablePathObjectTracking
_enablePathObjectTrackingSuperimposed
_enablePositionLockedForceLimitingProfile
_enablePositionLockedForceLimitingProfile_V3_1
_enablePositionLockedVelocityLimitingProfile
_enablePositionLockedVelocityLimitingProfile_V4_0
_enableQFAxis
_enableQFAxis_V3_0
_enableQFAxis_V4_0
_enableScheduler
_enableSensor
_enableTimeLockedForceLimitingProfile

_enableTimeLockedForceLimitingProfile_V3_1
_enableTimeLockedVelocityLimitingProfile
_enableTimeLockedVelocityLimitingProfile_V4_0
_enableTimeLockedVelocityLimitingProfile_V4_3
_enableTorqueLimiting
_enableTorqueLimiting_V2_0
_enableVelocityGearing
_enableVelocityLimitingValue
_enableVelocityLimitingValue_V4_0
_enableVelocityLimitingValue_V4_3
_exportUnitDataSet
_exportUnitDataSet_01
_exportUserDataSet
_FALSE
_FB_MF_MultiAxis
_FB_MF_SingleAxis
_FB_MF_STGP
_FB_STGP_BasicMC
_FB_STGP_Cam
_FB_STGP_Cam_Ext
_FB_STGP_Gear
_FB_STGP_Path
_FB_STGP_Position
_finite
_firstIndexOf
_fixedGearType
_forceTControllerIdentification
_formulaObjectType
_getActiveDpSlaveAddress
_getActiveNameOfStation
_getActuatorErrorNumberState
_getActuatorErrorState
_getAdditionObjectErrorNumberState
_getAdditionObjectErrorState
_getAlarmId

_getAverageTaskIdRunTime
_getAverageTaskRunTime
_getAxisDataSetParameter
_getAxisDataSetParameter_V3_1
_getAxisDataSetParameter_V4_1
_getAxisErrorNumberState
_getAxisErrorState
_getAxisInternalPosition
_getAxisSpecificState
_getAxisSpecificState2
_getAxisStoppingData
_getAxisUserPosition
_getBit
_getCamErrorNumberState
_getCamErrorState
_getCamFollowingDerivative
_getCamFollowingMinMax
_getCamFollowingValue
_getCamLeadingValue
_getCamSpecificState
_getCamSpecificState2
_getCamTrackErrorNumberState
_getCamTrackErrorState
_getCamTrackSpecificState
_getCircularPathData
_getCircularPathData_V4_2
_getCircularPathGeometricData
_getCommandId
_getConfigurationData
_getControllerObjectErrorNumberState
_getControllerObjectErrorState
_getCurrentTaskIdRunTime
_getCurrentTaskRunTime
_getDataByToken
_getDeviceId

_getDoIndexNumberFromLogAddress
_getDpStationAddressFromLogDiagnosticAddress
_getExternalEncoderErrorNumberState
_getExternalEncoderErrorState
_getExternalEncoderSpecificState
_getExternalEncoderSpecificState2
_getFixedGearErrorNumberState
_getFixedGearErrorState
_getFollowingObjectErrorNumberState
_getFollowingObjectErrorState
_getForceControlDataSetParameter
_getForceControlDataSetParameter_V3_1
_getForceControlDataSetParameter_V4_1
_getFormulaObjectErrorNumberState
_getFormulaObjectErrorState
_getGearAxisSpecificState
_getGearAxisSpecificState2
_getGeoAddressFromLogAddress
_getInOutByte
_getInternalTaskIdIdx
_getInternalTaskIdx
_getInternalTimeStamp
_getIO_Part_4
_getIPConfig
_getLinearPathData
_getLinearPathData_V4_2
_getLinearPathGeometricData
_getLogDiagnosticAddressFromDpStationAddress
_getMasterValue
_getMaximalTaskIdRunTime
_getMaximalTaskRunTime
_getMeasuringInputErrorNumberState
_getMeasuringInputErrorState
_getMeasuringInputSpecificState
_getMeasuringInputSpecificState2

_getMemoryCardId
_getMinimalTaskIdRunTime
_getMinimalTaskRunTime
_getMotionStateOfAxisCommand
_getMotionStateOfFollowingObjectCommand
_getMotionStateOfPathObjectCommand
_getNextLogAddress
_getOutputCamErrorNumberState
_getOutputCamErrorState
_getOutputCamSpecificState
_getOutputCamSpecificState2
_getPathAxesData
_getPathAxesPosition
_getPathCartesianData
_getPathCartesianPosition
_getPathGeometricData
_getPathObjectBcsFromOcsData
_getPathObjectErrorNumberState
_getPathObjectErrorState
_getPathObjectOcsFromBcsData
_getPendingAlarms
_getPnInterfacePortNeighbour
_getPNSyncCounter
_getPolynomialPathData
_getPolynomialPathData_V4_2
_getPolynomialPathGeometricData
_getPosAxisSpecificState
_getPosAxisSpecificState2
_getProgrammedTargetPosition
_getQFAxisDataSetParameter
_getQFAxisDataSetParameter_V3_1
_getSafeValue
_getSegmentIdentification
_getSensorErrorNumberState
_getSensorErrorState

_getSlaveValue
_getStateOfActuatorCommand
_getStateOfAdditionObjectCommand
_getStateOfAllDpSlaves
_getStateOfAllDpStations
_getStateOfAxisCommand
_getStateOfCamCommand
_getStateOfCamTrackCommand
_getStateOfControllerObjectCommand
_getStateOfDiagnosticDataCommand
_getStateOfDpSlave
_getStateOfExternalEncoderCommand
_getStateOfFixedGearCommand
_getStateOfFollowingObjectCommand
_getStateOfFormulaObjectCommand
_getStateOfIO
_getStateOfMeasuringInputCommand
_getStateOfMotionBuffer
_getStateOfOutputCamCommand
_getStateOfPathObjectCommand
_getStateOfPathObjectMotionBuffer
_getStateOfProcessInterruptCommand
_getStateOfRecordCommand
_getStateOfSensorCommand
_getStateOfSingleDpSlave
_getStateOfTask
_getStateOfTaskId
_getStateOfTo
_getStateOfUnitDataSetCommand
_GetStateOfXCommand
_getStationType
_getSyncCommandId
_getSystemTime
_getTaskId
_getTimeDifferenceOfInternalTimeStamp

_getTimeDifferenceOfInternalTimeStamps
_homing
_imData
_IMPLEMENTATION
_importUnitDataSet
_importUnitDataSet_01
_importUserDataSet
_INT
_INTERFACE
_INTERFACE_AND_IMPLEMENTATION
_internalTest
_interpolateCam
_isNaN
_lastIndexOf
_lengthIndexOf
_LINT
_loadUnitDataSet
_loadUnitDataSet_01
_LREAL
_MC_CamIn
_MC_CamIn_V4_0
_MC_CamInMode
_MC_CamOut
_MC_CamSwitch
_MC_CamSwitchDirection
_MC_CamSwitchOff
_MC_Direction
_MC_DisableMode
_MC_EnableMode
_MC_GearIn
_MC_GearInMode
_MC_GearOut
_MC_Home
_MC_HomingMode
_MC_Jog

_MC_MoveAbsolute
_MC_MoveAdditive
_MC_MoveRelative
_MC_MoveSuperimposed
_MC_MoveVelocity
_MC_Phasing
_MC_PositionProfile
_MC_Power
_MC_Power_V4_0
_MC_ReadActualPosition
_MC_ReadAxisError
_MC_ReadBoolParameter
_MC_ReadBoolParameter_V4_0
_MC_ReadParameter
_MC_ReadParameter_V4_0
_MC_ReadStatus
_MC_Reset
_MC_Stop
_MC_StopMode
_MC_VelocityProfile
_MC_WriteBoolParameter
_MC_WriteBoolParameter_V4_0
_MC_WriteParameter
_MC_WriteParameter_V4_0
_modifyCountES
_modifyCountRT
_move
_movePathCircular
_movePathCircular_V4_1
_movePathCircular_V4_2
_movePathLinear
_movePathLinear_V4_1
_movePathLinear_V4_2
_movePathPolynomial
_movePathPolynomial_V4_1

_movePathPolynomial_V4_2
_movePointToPoint
_movePointToPoint_V4_1
_MRES
_NOT
_OR
_pathAxis
_pathObjectTpye
_pos
_readDiagnosticData
_readDriveFaults
_readDriveMultiParameter
_readDriveMultiParameterDescription
_readDriveParameter
_readDriveParameterDescription
_readRecord
_readSystemClock
_readVariableDiagnosticData
_readVariableRecord
_REAL
_redefineExternalEncoderPosition
_redefinePathObjectOcs
_redefinePosition
_releaseSemaphore
_removeBufferedActuatorCommandId
_removeBufferedAdditionObjectCommandId
_removeBufferedAxisCommandId
_removeBufferedCamCommandId
_removeBufferedCamTrackCommandId
_removeBufferedControllerObjectCommandId
_removeBufferedExternalEncoderCommandId
_removeBufferedFixedGearCommandId
_removeBufferedFollowingObjectCommandId
_removeBufferedFormulaObjectCommandId
_removeBufferedMeasuringInputCommandId

_removeBufferedOutputCamCommandId
_removeBufferedPathObjectCommandId
_removeBufferedSensorCommandId
_reset_AllAlarmId
_resetActuator
_resetActuatorConfigDataBuffer
_resetActuatorError
_resetAdditionObject
_resetAdditionObjectConfigDataBuffer
_resetAdditionObjectError
_resetAlarmId
_resetAllAlarmId
_resetAxis
_resetAxis_V2_0
_resetAxis_V3_0
_resetAxisConfigDataBuffer
_resetAxisError
_resetAxisError_V3_0
_resetCam
_resetCam_V2_0
_resetCam_V3_0
_resetCam_V4_2
_resetCamConfigDataBuffer
_resetCamError
_resetCamError_V3_0
_resetCamTrack
_resetCamTrackConfigDataBuffer
_resetCamTrackError
_resetControllerObject
_resetControllerObjectConfigDataBuffer
_resetControllerObjectError
_resetDriveObjectFault
_resetExternalEncoder
_resetExternalEncoder_V2_0
_resetExternalEncoder_V3_0

_resetExternalEncoderConfigDataBuffer
_resetExternalEncoderError
_resetExternalEncoderError_V3_0
_resetFixedGear
_resetFixedGearConfigDataBuffer
_resetFixedGearError
_resetFollowingObject
_resetFollowingObject_V2_0
_resetFollowingObject_V3_0
_resetFollowingObject_V3_1
_resetFollowingObjectConfigDataBuffer
_resetFollowingObjectError
_resetFollowingObjectError_V3_0
_resetFormulaObject
_resetFormulaObjectConfigDataBuffer
_resetFormulaObjectError
_resetMeasuringInput
_resetMeasuringInput_V2_0
_resetMeasuringInput_V3_0
_resetMeasuringInputConfigDataBuffer
_resetMeasuringInputError
_resetMeasuringInputError_V3_0
_resetMotionBuffer
_resetOutputCam
_resetOutputCam_V2_0
_resetOutputCam_V3_0
_resetOutputCamConfigDataBuffer
_resetOutputCamError
_resetOutputCamError_V3_0
_resetPathObject
_resetPathObjectConfigDataBuffer
_resetPathObjectError
_resetPathObjectMotionBuffer
_resetSensor
_resetSensorConfigDataBuffer

_resetSensorError
_resetTask
_resetTaskId
_resetTController
_resetTControllerError
_resetTechnologicalErrors
_resetUnitData
_restartTask
_restartTaskId
_resumeTask
_resumeTaskId
_RETAIN
_retriggerTaskControlTime
_retriggerTaskIdControlTime
_RUN
_runMotionInPositionLockedForceProfile
_runMotionInPositionLockedVelocityProfile
_runPositionBasedMotionIn
_runPositionBasedMotionIn_V3_2
_runPositionLockedForceProfile
_runPositionLockedVelocityProfile
_runTimeLockedForceProfile
_runTimeLockedForceProfile_V3_1
_runTimeLockedPositionProfile
_runTimeLockedVelocityProfile
_runVelocityBasedMotionIn
_runVelocityBasedMotionIn_V3_2
_S7_COUNTER
_S7_TIMER
_saveConfigData
_savePersistentMemoryData
_saveUnitDataSet
_saveUnitDataSet_01
_SC_ALARM_CONFIGURATION
_SC_ARRAY_BOUND_ERROR_READ

_SC_ARRAY_BOUND_ERROR_WRITE
_SC_BACKGROUND_TIMER_OVERFLOW
_SC_BATTERY_WARNING_LEVEL1
_SC_BATTERY_WARNING_LEVEL2
_SC_CYCLE_TIMER_OVERFLOW
_SC_DEVICE_COMMAND
_SC_DIAGNOSTIC_INTERRUPT
_SC_DIVISION_BY_ZERO
_SC_DP_CLOCK_DETECTED
_SC_DP_SLAVE_NOT_SYNCHRONIZED
_SC_DP_SLAVE_SYNCHRONIZED
_SC_DP_SYNCHRONIZATION_LOST
_SC_DRIVE_OBJECT_ALARM
_SC_DRIVE_OBJECT_FAULT
_SC_EXCEPTION
_SC_EXTERNAL_COMMAND
_SC_IMAGE_UPDATE_FAILED
_SC_IMAGE_UPDATE_OK
_SC_INVALID_ADDRESS
_SC_INVALID_FLOATING_POINT_OPERATION
_SC_IO_MODULE_NOT_SYNCHRONIZED
_SC_IO_MODULE_SYNCHRONIZED
_SC_MODE_SELECTOR
_SC_PC_INTERNAL_FAILURE
_SC_PROCESS_INTERRUPT
_SC_PULL_PLUG_INTERRUPT
_SC_STATION_DISCONNECTED
_SC_STATION_RECONNECTED
_SC_TO_INSTANCE_NOT_EXISTENT
_SC_VARIABLE_ACCESS_ERROR_READ
_SC_VARIABLE_ACCESS_ERROR_WRITE
_sendProcessInterrupt
_sensorType
_SERVICE
_setAndGetEncoderValue

_setAxisDataSetActive
_setAxisDataSetParameter
_setAxisDataSetParameter_V3_1
_setAxisDataSetParameter_V4_1
_setAxisSTW
_setAxisSTW_V4_0
_setBit
_setCammingOffset
_setCammingOffset_V1_1
_setCammingOffset_V3_1
_setCammingScale
_setCammingScale_V1_1
_setCammingScale_V3_1
_setCamOffset
_setCamScale
_setCamTrackState
_setControllerObjectPIDControl
_setDataByToken
_setDeviceErrorLED
_setDpSlaveAddress
_setDriveObjectSTW
_setExternalEncoderValue
_setFixedGearingOffset
_setFixedGearMaster
_setForceCommandValue
_setForceCommandValue_V3_1
_setForceControlDataSetParameter
_setForceControlDataSetParameter_V3_1
_setForceControlDataSetParameter_V4_1
_setFormula
_setFormulaObjectOutputValue
_setGearingOffset
_setGearingOffset_V3_1
_setIO_Part_4
_setIPConfig

_setIpoClockBorderToPosition
_setMaster
_setMaster_V3_0
_setMasterValue_V3_0
_setModeSelfAdaptingConfiguration
_setNameOfStation
_setOutputCamCounter
_setOutputCamState
_setOutputCamState_V4_0
_setPathObjectOcs
_setQFAxisDataSetParameter
_setQFAxisDataSetParameter_V3_1
_setQFAxisFCharacteristics
_setQFAxisQCharacteristics
_setSafeValue
_setSystemTime
_setTControllerActualIdentificationType
_setTControllerControlRangeParameter
_setTControllerCycleParameter
_setTControllerCycleParameterSecondary
_setTControllerDPIDParameter
_setTControllerDPIDParameterSecondary
_setTControllerIdentificationModifiedTangentMethodParameter
_setTControllerIdentificationModifiedTangentMethodProcessParameter
_setTControllerIdentificationStandardTangentMethodParameter
_setTControllerIdentificationStandardTangentMethodProcessParameter
_setTControllerInputDisplayValueParameter
_setTControllerInputFilterParameter
_setTControllerInputGradientCheckParameter
_setTControllerInputLimitCheckParameter
_setTControllerIntegrator
_setTControllerIntegratorSecondary
_setTControllerLowerPlausibilityParameter
_setTControllerLowerPlausibilityParameterSecondary
_setTControllerManualOutputValue

_setTControllerOperatingMode
_setTControllerProcessModeParameter
_setTControllerPWMParameter
_setTControllerPWMParameterSecondary
_setTControllerSetpoint
_setTControllerUpperPlausibilityParameter
_setTControllerUpperPlausibilityParameterSecondary
_SHUTDOWN
_SINT
_sizeOf
_startSyncCommands
_startTask
_startTaskId
_STARTUP
_startupData
_stop
_stop_V3_0
_stop_V4_2
_stopEmergency
_stopEmergency_V3_0
_stopEmergency_V4_2
_stopPath
_stopPath_V4_1
_STOPU
_StructCamTrackArrayOfSingleCamSettings
_StructCamTrackSingleCamSettings
_suspendTask
_suspendTaskDebug
_suspendTaskId
_suspendTaskIdDebug
_synchronizeDpInterfaces
_synchronizeExternalEncoder
_taskTraceStart
_taskTraceStop
_taskTraceUserEvent

_taskTraceWriteout
_tcpCloseConnection
_tcpCloseServer
_tcpOpenClient
_tcpOpenServer
_tcpReceive
_tcpSend
_testAndSetSemaphore
_testSFBSysDataInit
_TIME
_TIME_OF_DAY
_toggleBit
_triggerCommissioningMode
_triggerTestMode
_TRUE
_UDINT
_udpAddMulticastGroupMembership
_udpDropMulticastGroupMembership
_udpReceive
_udpSend
_UINT
_ULINT
_upsData
_USINT
_waitTime
_WORD_FROM_2BYTE
_WORD_TO_2BYTE
_writeAndSendMessage
_writeDriveMultiParameter
_writeDriveParameter
_writeRecord
_writeVariableRecord
_XOR
_Xreceive
_Xsend

A

ABORT_CONNECTION
ABORT_CURRENT_COMMAND
ABORTED
abortPosition
ABS
ABSOLUTE
absoluteEncoder
ACCELERATING
ACCELERATING_FORCE
ACCESS_DENIED
accessState
ACCORDING_TO_DEFINITION_ORDER
ACOS
ACTION
ACTIVATE_CHANGED_CONFIG_DATA
ACTIVATE_CONFIGURATION_DATA
ACTIVATE_FALL_BACK_CONFIGURATION
ACTIVATE_RESTART
ACTIVATED
ACTIVATED_NO_ALARM
activationModeChangedConfigData
ACTIVE
ACTIVE_ABSOLUTE
ACTIVE_AND_WAITING_FOR_CAM_TRACK_OUTPUT_INACTIVE
ACTIVE_AND_WAITING_FOR_DATA
ACTIVE_AND_WAITING_FOR_NEXT_CYCLE
ACTIVE_HOMING
ACTIVE_RELATIVE
ACTIVE_WITH_CIRCULAR_MOVE
ACTIVE_WITH_CONSTANT_LIMITS
ACTIVE_WITH_CONSTANT_LIMITS_AND_SPECIFIC_VALUE
ACTIVE_WITH_DYNAMIC_ADAPTION
ACTIVE_WITH_POLYNOMIAL_MOVE
ACTIVE_WITH_VARIABLE_LIMITS

ACTIVE_WITH_VARIABLE_LIMITS_AND_SPECIFIC_VALUE
ACTIVE_WITHOUT_DYNAMIC_ADAPTION
activeCam
activeMaster
ACTOR_ADAPTION_DATA
actorData
actorMonitoring
ACTUAL
ACTUAL_ACTIVATED
ACTUAL_AND_DEFAULT_VALUE
ACTUAL_DIRECTION_PASSIVE
ACTUAL_VALUE
ACTUAL_VALUE_INSIDE_POSITIONING_WINDOW
ACTUAL_VALUE_OUT_OF_POSITIONING_WINDOW
ACTUAL_VALUE_TOLERANCE
actualCycleData
ActualDPIDData
actualIdentificationModifiedTangentMethodData
actualIdentificationStandardTangentMethodData
actualIdentificationType
actualInputData
actualInputLimitCheckData
actualInputState
actualTorque
actualValueDefault
actualValueIn
ADAPTED_AND_DIFFERENT
ADAPTED_AND_EQUAL
ADAPTION_ERROR
ADD
ADD_DT_TIME
ADD_TIME
ADD_TOD_TIME
additionalSensorData
additionResult

additiveTorque
additiveTorqueIn
ADVANCED
ADVANCED_TYPE
ALARM_S
ALARM_SQ
ALARMS_ERROR
ALARMS_QSTATE
ALARMS_STATE
ALL
ALL_ADAPTION_DATA
ALL_AXIS_MOTION
ALL_EDGES
ALL_ERRORS
ALL_ERRORS_WITH_ABORT_SYNCHRONIZATION
ALL_GLOBAL
ALL_ID
ALL_POSITION_RELATED_MOTION
ANALOG
ANALOG_TEMPERATURE
AND
ANDN
ANYOBJECT
ANYOBJECT_TO_OBJECT
ANYTYPE_TO_BIGBYTEARRAY
AnyType_to_LittleByteArray
APPLICATION
APPROACH_MOVE
APPROACH_NEGATIVE
APPROACH_NEGATIVE_PASSIVE
APPROACH_POSITIVE
APPROACH_POSITIVE_PASSIVE
ARRAY
ARTICULATED_ARM
ARTICULATED_ARM_2D

AS
ASIN
ASSEMBLY_BASE_DRIVE
ASSEMBLY_BASE_EXTERN
ASSEMBLY_BASE_LINEAR
ASSEMBLY_BASE_LOAD
AT
AT_DECELERATION_START
AT_END_OF_COMMAND
AT_MOTION_START
AT_PROFILE_START
AT_SYNCHRONIZING_START
AT_THE_END_OF_CAM_CYCLE
ATAN
ATTACHED_STEADILY
AUTOMATIC_INTERFACE_SYNCHRONIZATION
AUTOMATICALLY
AVAILABLE
AVERAGING
AXIS_DISABLED
AXIS_HOMED
AXIS_STOPPED_AT_POSITION

B

B_SPLINE
BASIC
BASIC_AND_SUPERIMPOSED_MOTION_ACTIVE
BASIC_AND_SUPERIMPOSED_POS_MOTION_ACTIVE
BASIC_MOTION
BASIC_MOTION_ACTIVE
BASIC_MOTION_ACTIVE_WITH_LENGTH_OUTPUT
BASIC_NON_POS_AND_SUPERIMPOSED_POS_MOTION_ACTIVE
BASIC_POS_AND_SUPERIMPOSED_NON_POS_MOTION_ACTIVE
BASIC_POS_MOTION_ACTIVE
basicMotion

Baudrate_1
Baudrate_2
Baudrate_3
Baudrate_4
Baudrate_5
BCD_TO_BYTE
BCD_TO_DINT
BCD_TO_DWORD
BCD_TO_INT
BCD_TO_LWORD
BCD_TO_SINT
BCD_TO_WORD
bcs
BE_SYNCHRONOUS_AT_POSITION
BEFORE_ANTIWINDUP_LIMITATION
BEGIN_OF_CAM
BEGIN_SYNC
BEGIN_TO_STOP_WHEN_POSITION_REACHED
BEHIND_ANTIWINDUP_LIMITATION
BigByteArray_to_AnyType
BIGBYTEARRAY_TOANYTYPE
BIN_CODE
BOOL
BOOL_TO_BYTE
BOOL_TO_DWORD
BOOL_TO_WORD
BOOL_VALUE_TO_DINT
BOOL_VALUE_TO_INT
BOOL_VALUE_TO_LREAL
BOOL_VALUE_TO_REAL
BOOL_VALUE_TO_SINT
BOOL_VALUE_TO_UDINT
BOOL_VALUE_TO_UINT
BOOL_VALUE_TO_USINT
BOTH

BOTH_DIRECTION
BOTH_EDGES
BOTH_EDGES_FIRST_FALLING
BOTH_EDGES_FIRST_RISING
BUFFERED
BY
BY_CAM_TRACK_END
BY_CENTER_AND_ARC
BY_COMMAND
BY_END_POSITION
BY_PROFILE_END
BY_START_POSITION
BY_STW_BIT
BY_VALUE
BYTE
BYTE_TO_BCD
BYTE_TO_BOOL
BYTE_TO_DINT
BYTE_TO_DWORD
BYTE_TO_INT
BYTE_TO_SINT
BYTE_TO_UDINT
BYTE_TO_UINT
BYTE_TO_USINT
BYTE_TO_WORD
BYTE_VALUE_TO_LREAL
BYTE_VALUE_TO_REAL

C

C_SPLINE
CACHE_ID
CAL
CALC
CALCN
CAM_AND_ZM_PASSIVE

CAM_DATA_RESET
CAM_PASSIVE
CAMMING
cammingAdjustments
CAMTRACK_DISABLE
camTrackPosition
camType
CARTESIAN
CASE
CHANGE_DS_IS_LOCKED
CHANGE_FAILED
CHANNEL_0
CHANNEL_1
CHANNEL_2
CHANNEL_3
CHANNEL_4
CHANNEL_5
CHANNEL_6
CHANNEL_7
CIRCULAR_TRANSITION
circularPathCommand
CLAMP_BY_FOLLOWING_ERROR_DEVIATION
CLAMP_WHEN_TORQUE_LIMIT_REACHED
CLOSE_ON_EXIT
CMD_CONTINUEAXIS
CMD_DISABLEAXIS
CMD_DISABLEAXISSIMULATION
CMD_ENABLEAXIS
CMD_ENABLEAXISSIMULATION
CMD_ENABLEMEASURINGINPUT
CMD_HOMING
CMD_INIT
CMD_MOVE
CMD_POSABS
CMD_POSREL

CMD_RESETAXIS
CMD_SETDATASETPARAM
CMD_STOPAXIS
CMD_STOPEMERGENCY
COLLECT_CHANGED_CONFIG_DATA
COMMAND_DONE
COMMAND_DYNAMICS
COMMAND_FAILED
COMMAND_ID
COMMAND_NOT_FOUND
COMMAND_VALUE
COMMAND_VALUE_BASIC_MOTION
COMMAND_VALUE_SUPERIMPOSED_MOTION
COMMAND_VALUE_TOLERANCE
CommandIdType
COMPATIBILITY_MODE
COMPLETE_RANGE
CONCAT
CONCAT_DATE_TOD
CONDITION_1
CONDITION_1_AND_CONDITION_2
CONDITION_1_OR_CONDITION_2
CONDITION_2
CONFIG_TO_SHADOW
CONFIGURATION
CONFIGURATION_DELETED
CONFIGURATION_ERROR
CONFIGURATION_ID_NOT_FOUND
CONFIGURATION_NOT_FOUND
CONFIGURATION_VERSION
CONFIGURED
CONFIGURED_OUTPUT_VALUE
CONNECTED
CONSTANT
CONSTANT_FORCE

CONSTANT_MOVE
CONTINUOUS
control
CONTROL_STOP
controlDeviation
controllerOutput
controlRangeParameter
COS
counterCamData
counterMeasuredValue1
counterMeasuredValue2
CRITICAL
CTD
CTD_DINT
CTD_LINT
CTD_UDINT
CTD_ULINT
CTU
CTU_DINT
CTU_LINT
CTU_UDINT
CTU_ULINT
CTUD
CTUD_DINT
CTUD_LINT
CTUD_UDINT
CTUD_ULINT
CUBIC_MODE
CURRENT
CURRENT_MODE
currentMasterData
currentSlaveData
currentSyncPosition
cycleParameter
cycleParameterSecondary

CYCLIC
CYCLIC_ABSOLUTE
CYCLIC_RELATIVE
cyclicMeasuringEnableCommand

D

DATA_EXCHANGE_INACTIVE
DATA_INCOMPATIBLE
DATA_INCOMPLETE
DATA_MISMATCH
DATASET_ALREADY_EXISTS
DATASET_ID_NOT_VALID
DATASET_NOT_FOUND
dataSetMonitoring
DATE
DATE_AND_TIME
DATE_AND_TIME_TO_DATE
DATE_AND_TIME_TO_TIME_OF_DAY
dccAux_2Clock
dccAuxClock
DEACTIVATED
DEACTIVATED_NO_ALARM
DECELERATING
DECELERATING_FORCE
DECODE_STOP
DEFAULT_MODE
DEFAULT_PASSIVE
DEFAULT_UNIT
DEFAULT_VALUE
defaultAdditiveTorque
defaultMotionIn
defaultTorqueLimitNegative
defaultTorqueLimitPositive
definitionState
DELETE

DELTA_2D_PICKER
DELTA_3D_PICKER
DEPENDING_ON_OUTER_INPUT_LIMITCHECK_STATUS
derivedValue
DESYNCHRONIZING
DIFF_NEGATIVE
DIFF_POSITIVE
DIFFERENTIATION
DINT
DINT_TO_BCD
DINT_TO_BYTE
DINT_TO_DWORD
DINT_TO_INT
DINT_TO_LREAL
DINT_TO_REAL
DINT_TO_SINT
DINT_TO_STRING
DINT_TO_UDINT
DINT_TO_UINT
DINT_TO_USINT
DINT_TO_WORD
DINT_VALUE_TO_BOOL
DINTIn1
DINTIn1Default
DINTIn2
DINTIn2Default
DINTIn3
DINTIn3Default
DINTIn4
DINTIn4Default
DINTOut1
DINTOut1Default
DINTOut2
DINTOut2Default
DINTOut3

DINTOut3Default
DINTOut4
DINTOut4Default
DIRECT
DIRECT_HOMING
DIRECT_HOMING_RELATIVE
DIRECT_MODE
DIRECT_OUTPUT
DIRECT_TO_SHADOW
DIRECT_VALUE
DISABLE
DISABLE_ALL
DISABLE_BLENDING
DISABLE_CONTROLLER
DISABLE_DRIVE_IMMEDIATELY
DISABLE_MEASURE_AND_AVARAGE_OUTPUT_VALUE
DISABLE_MEASURE_AND_MANUAL_OUTPUT_VALUE
DISABLE_MOTION
DISABLE_OUTPUT
disableCommand
DISCONTINUOUS
distributedMotion
DISTURBED
DIV
DIVTIME
DO
DO_NOT_CHANGE
DO_NOT_CLAMP
DO_NOT_CLOSE_ON_EXIT
DO_NOT_SET_DP_ALARM
DO_NOT_STOP
DONE
DP_1
DP_2
DP_3

DP_CLOCk_DETECTED
DP_INTERFACES_SYNCHRONIZED
DP_SLAVE_NOT_SYNCHRONIZED
DP_SLAVE_SYNCHRONIZED
DP_TEL1_STANDARD
DP_TEL101_611U_VELCTRL_NO_ENCODER
DP_TEL102_611U_POSCTRL_1_ENCODER
DP_TEL103_611U_POSCTRL_2_ENCODER
DP_TEL105_611U_DSC_1_ENCODER
DP_TEL106_611U_DSC_2_ENCODER
DP_TEL2_STANDARD
DP_TEL3_STANDARD
DP_TEL4_STANDARD
DP_TEL5_STANDARD
DP_TEL6_STANDARD
DP_TEL81_STANDARD
DP_TEL82_STANDARD
DP_TEL83_STANDARD
DPIDParameter
DPIDParameterSecondary
DPMASTER
DRIVE
DRIVE_INTERFACE
driveAxis
driveData
DSC_SVS_DEVICE_ALARMS_EVENT_ID_IN_USE
DSC_SVS_DEVICE_ALARMS_EVENT_ID_NOT_USED
DSC_SVS_DEVICE_ALARMS_ILLEGAL_EVENT_ID
DSC_SVS_DEVICE_ALARMS_INTERNAL_ERROR
DSC_SVS_DEVICE_ALARMS_IV_CALL
DSC_SVS_DEVICE_ALARMS_IV_FIRST_CALL
DSC_SVS_DEVICE_ALARMS_IV_SFC_TYP
DSC_SVS_DEVICE_ALARMS_LAST_ENTRY_USED
DSC_SVS_DEVICE_ALARMS_LAST_SIGNAL_USED
DSC_SVS_DEVICE_ALARMS_NO_ENTRY

DT
DT_TO_DATE
DT_TO_TOD
DWORD
DWORD_TO_BCD
DWORD_TO_BOOL
DWORD_TO_BYTE
DWORD_TO_DINT
DWORD_TO_INT
DWORD_TO_REAL
DWORD_TO_SINT
DWORD_TO_UDINT
DWORD_TO_UINT
DWORD_TO_USINT
DWORD_TO_WORD
DWORD_VALUE_TO_LREAL
DWORD_VALUE_TO_REAL

E

EDGE_NEG_SIDE_NEG
EDGE_NEG_SIDE_NEG_PASSIVE
EDGE_NEG_SIDE_POS
EDGE_NEG_SIDE_POS_PASSIVE
EDGE_POS_SIDE_NEG
EDGE_POS_SIDE_NEG_PASSIVE
EDGE_POS_SIDE_POS
EDGE_POS_SIDE_POS_PASSIVE
EFFECTIVE
effectiveData
effectiveTaskRuntime
ELSE
ELSIF
EMPTY
EN
ENABLE

ENABLE_OFFSET_OF_ABSOLUTE_ENCODER
enableCommand
enableForceControlByConditionCommand
enableForceLimitingByConditionCommand
enableValidCam
ENCODER_ADAPTION_DATA
ENCODER_DISABLE
END_ACTION
END_CASE
END_CONFIGURATION
END_EXPRESSION
END_FOR
END_FUNCTION
END_FUNCTION_BLOCK
END_IF
END_IMPLEMENTATION
END_INTERFACE
END_LABEL
END_MOUNTED_SWITCH
END_OF_INTERPOLATION
END_OF_MOTION_STOP
END_POINT
END_PROGRAM
END_REPEAT
END_RESOURCE
END_STEP
END_STRUCT
END_SYNC
END_TRANSITION
END_TYPE
END_VAR
END_WAITFORCONDITION
END_WHILE
ENDAT
ENO

Enum_STGP_CMD
Enum_STGP_STATE
ENUM_TO_DINT
EnumAbsBaudrate
EnumAbsMsgFormat
EnumAbsMsgLength
EnumAbsoluteRelative
EnumAbsState
EnumAcceleration
EnumAccessMode
EnumActivatedDeactivated
EnumActiveAbsRelInactive
EnumActiveInactive
EnumActiveInactiveNoChange
EnumActiveNoChange
EnumActiveWaitingForDataInactive
EnumActualDirect
EnumActualValueFormat
EnumAdditionExecution
EnumAdditionObjectErrorReaction
EnumAlarmIdState
EnumAlarmIdType
EnumApproachPositionType
EnumAxisAbortAcceleration
EnumAxisActuatorInterfaceAllocationConfig
EnumAxisAdditionalSensorType
EnumAxisApproachDirection
EnumAxisCompareMode
EnumAxisControllerOutput
EnumAxisControllerType
EnumAxisCyclicSetUpInForceLimiting
EnumAxisDataAdaption
EnumAxisDelayValueCalculationMode
EnumAxisDelayValueState
EnumAxisDioActorType

EnumAxisDriverMode
EnumAxisEnableMovingMode
EnumAxisEncoderAssemblyType
EnumAxisEncoderIdentification
EnumAxisEncoderMode
EnumAxisEncoderSystem
EnumAxisEncoderType
EnumAxisEncoderValueType
EnumAxisErrorReaction
EnumAxisExtrapolatedVelocitySwitch
EnumAxisFilterMode
EnumAxisFineInterpolatorMode
EnumAxisFollowingErrorConfigMode
EnumAxisFollowingMotionState
EnumAxisForceCommand
EnumAxisForceDerivativeLimitingMode
EnumAxisForceExtrapolationType
EnumAxisForceLimitingCommandType
EnumAxisForceProfileModeConditionCommand
EnumAxisForceProfileProcessingState
EnumAxisForceState
EnumAxisForceValueConditionCommand
EnumAxisFrictionReference
EnumAxisHomingMode
EnumAxisHomingReverseCamType
EnumAxisHomingState
EnumAxisIdentification
EnumAxisInterfaceActivationConfig
EnumAxisIntervalCounterMode
EnumAxisKindOfDataAdaption
EnumAxisLimitingProfileProcessingState
EnumAxisMotionCommand
EnumAxisMotionInCommandState
EnumAxisMotionState
EnumAxisMotorType

EnumAxisOperatingMode
EnumAxisPassiveApproachDirection
EnumAxisPassiveHomingMode
EnumAxisPathMotionState
EnumAxisPathPosToleranceCommandValue
EnumAxisPosCommandSpecification
EnumAxisProfileProcessingState
EnumAxisProgrammedTargetPosition
EnumAxisPulsesEnabledEvaluation
EnumAxisReferenceCamType
EnumAxisReferenceMaxNominal
EnumAxisSensorInterfaceAllocationConfig
EnumAxisServoMonitoringPositioningState
EnumAxisStateMachineControl
EnumAxisSwitchingCondition
EnumAxisSwitchingCondition_1
EnumAxisSwitchingCondition_2
EnumAxisSwLimitModeSpecificMonitoring
EnumAxisTelegramType
EnumAxisTorqueForceReductionGranularity
EnumAxisType
EnumAxisUpdateCycle
EnumAxisUserDefaultHighLow
EnumAxisValueReferenceType
EnumAxisVelocityLimitingCommandType
EnumAxisVelocityLimitingDirection
EnumAxisVelocityLimitingValueMode
EnumBackLashDiff
EnumBackLashType
EnumBalanceFilterMode
EnumBlendingMode
EnumCamBSplineInterpolation
EnumCamCSplineInterpolation
EnumCamData
EnumCamDataMode

EnumCamDefinitionState
EnumCamErrorReaction
EnumCamExtremumType
EnumCamFollowingRangeSpecificationMode
EnumCamInterpolationMode
EnumCamLeadingRangeStartPoint
EnumCammingActivationMode
EnumCammingDirection
EnumCammingMode
EnumCamMode
EnumCamModify
EnumCamPositionMode
EnumCamRange
EnumCamRangeSpecification
EnumCamReferenceBySlaveModeRelative
EnumCamRepresentation
EnumCamTrackActivationMode
EnumCamTrackErrorReaction
EnumCamTrackNextCommand
EnumCamTrackOutputType
EnumCamTrackPositionReference
EnumCamTrackSetState
EnumCamTrackStartMode
EnumCamTrackState
EnumCamTrackStopMode
EnumCamTrackTaskLevel
EnumCamTrackType
EnumCamTrackValue
EnumCamValueType
EnumChangeMode
EnumCommandIdState
EnumCommandValueQuantizationMode
EnumCompactControllerControllerType
EnumConfigurationInfoId
EnumContinueDynamicType

EnumContinueSpecification
EnumControllerObjectErrorBehaviorMode
EnumControllerObjectErrorReaction
EnumControllerObjectInputType
EnumControllerObjectISetMode
EnumControllerObjectPrecontrolAddupMode
EnumControllerObjectValueBehaviorMode
EnumConversionDataType
EnumDataDefault
EnumDataSetChangingState
EnumDataType
EnumDecodeSequentialMotionCommand
EnumDefaultValueDirect
EnumDeviceAbortReadWriteJobs
EnumDeviceCommandIdState
EnumDeviceConfigurationActivationState
EnumDeviceDataActivationState
EnumDeviceDataScope
EnumDeviceDpAlarmMode
EnumDeviceIdType
EnumDeviceKindOfData
EnumDeviceModeOfOperation
EnumDeviceStartupOperationMode
EnumDeviceStateOfDpSlave
EnumDeviceStorageType
EnumDeviceUnitDataSetCommand
EnumDirectEffectiveUserDefault
EnumDirection
EnumDirectionType
EnumDisableQFAxisOutputEnableMode
EnumDisableQFAxisOutputMode
EnumDoNotChangeDirect
EnumDpInterfaceSyncMode
EnumDpInterfaceSyncState
EnumDpSegmentId

EnumDpSlaveSyncState
EnumDriveFaultsType
EnumEffectiveUserDefault
EnumEnableAxisMode
EnumEnableDisable
EnumEncoderErrorReaction
EnumEncoderIdentification
EnumEndBehaviourOfProfile
EnumErrorReporting
EnumErrorReset
EnumEventClass
EnumExtendedValue
EnumExtendedValueType
EnumFanBattery
EnumFctGenStartStop
EnumFctGenState
EnumFixedGearDirectEffectiveUserDefault
EnumFixedGearDirection
EnumFixedGearDisableMode
EnumFixedGearEnableMode
EnumFixedGearErrorBehaviorMode
EnumFixedGearErrorReaction
EnumFixedGearGearingMode
EnumFixedGearGearingType
EnumFixedGearMergeModeDisableGearing
EnumFixedGearMergeModeEnableGearing
EnumFixedGearMotionOutBehaviorMode
EnumFixedGearNextCommandDisableGearing
EnumFixedGearNextCommandEnableGearing
EnumFollowingObjectDynamicMergeMode
EnumFollowingObjectDynamicReference
EnumFollowingObjectErrorReaction
EnumFollowingObjectMotionImpact
EnumFollowingObjectStateSetMasterCommand
EnumFollowingObjectSynchronizeWithLookAhead

EnumFollowingObjectSynchronizingDirection
EnumFollowingObjectSynchronizingState
EnumFollowingObjectSyncMode
EnumFollowingObjectVelocityMode
EnumFollowingState
EnumForceControlByConditionCommandState
EnumForceController
EnumForceControllerFilterType
EnumForceControllerState
EnumForceControllerType
EnumForceControllerTypeOfSensorData
EnumForceDirection
EnumForceLimitingByConditionCommandState
EnumFormulaErrorReaction
EnumFormulaObjectErrorBehaviorMode
EnumFormulaObjectOutBehaviorMode
EnumGearingActivationMode
EnumGearingDirection
EnumGearingMode
EnumGearingPosToleranceCommandValue
EnumGearingType
EnumGetValue
EnumHomingMode
EnumInactiveNoChange
EnumIncomingOutgoing
EnumInOutDirection
EnumInSensorIdentification
EnumInSensorMode
EnumInSensorSystem
EnumInSensorType
EnumInSensorValueType
EnumInsertMode
EnumIntegratorMode
EnumInterfaceID
EnumInterfaceValueDefaultValue

EnumInterpolationState
EnumIoIdType
EnumJerk
EnumLastValidInterfaceValueDefaultValue
EnumLeadingRangeEndPoint
EnumLimitExceededOk
EnumLogicOperation
EnumMasterMode
EnumMeasuredEdge
EnumMeasuringInputAccess
EnumMeasuringInputCyclicMode
EnumMeasuringInputErrorReaction
EnumMeasuringInputInputType
EnumMeasuringInputTaskLevel
EnumMeasuringRangeMode
EnumMeasuringState
EnumMemoryCardIdType
EnumMergeMode
EnumMergeModeDisableCamming
EnumMergeModeDisableGearing
EnumMergeModeEnableCamming
EnumMergeModeEnableGearing
EnumMergeModeEnableMotionIn
EnumMergeModeForceTimeProfile
EnumMergeModeStop
EnumModeSelfAdaptingConfiguration
EnumMotionBaseType
EnumMotionBufferState
EnumMotionCommandIdState
EnumMountSwitch
EnumMoveTimeOut
EnumMovingMode
EnumMovingModeStopCommand
EnumNextCommand
EnumNextCommandCamming

EnumNextCommandCancelCommand
EnumNextCommandDataAdaption
EnumNextCommandDelayValueCalculation
EnumNextCommandDisableClamping
EnumNextCommandDisableForceLimiting
EnumNextCommandDisableLimiting
EnumNextCommandDisableTorqueLimiting
EnumNextCommandEnable
EnumNextCommandEnableAxisInterface
EnumNextCommandEnableClamping
EnumNextCommandEnableForceControl
EnumNextCommandEnableForceLimitingValue
EnumNextCommandEnableLimitingValue
EnumNextCommandEnableMeasuring
EnumNextCommandEnableMotionIn
EnumNextCommandEnableTorqueLimiting
EnumNextCommandEncoderSimulation
EnumNextCommandForceProfile
EnumNextCommandForceValue
EnumNextCommandGearing
EnumNextCommandHoming
EnumNextCommandIpoClockBorderToPosition
EnumNextCommandMode
EnumNextCommandPathMove
EnumNextCommandProfile
EnumNextCommandReset
EnumNextCommandScaling
EnumNextCommandSensor
EnumNextCommandSetMaster
EnumNextCommandSyncEncoder
EnumNextEnablePathObjectMotionTracking
EnumOffsetMode
EnumOkFaulted
EnumOnOff
EnumOperationMode

EnumOutputCamErrorReaction
EnumOutputCamInvert
EnumOutputCamOutputType
EnumOutputCamPosition
EnumOutputCamState
EnumOutputCamTaskLevel
EnumOutputCamType
EnumOutputCamValue
EnumPathBlendingMode
EnumPathCartesianKinematicsType
EnumPathCircularDirection
EnumPathCircularType
EnumPathDataSource
EnumPathDynamicAdaption
EnumPathErrorReaction
EnumPathIjkMode
EnumPathKinematicsConfig2D
EnumPathKinematicsType
EnumPathMergeMode
EnumPathMode
EnumPathMotionBufferState
EnumPathMotionCommand
EnumPathMotionCommandIdState
EnumPathMotionState
EnumPathObjectBlendingAcceleration
EnumPathObjectCsType
EnumPathObjectOcsSettingType
EnumPathObjectRedefineOcsMode
EnumPathObjectTrackingSynchronizingMode
EnumPathOverlappingBlendingDistances
EnumPathPlane
EnumPathPointType
EnumPathPolynomialMode
EnumPathPositionIndication
EnumPathStopMode

EnumPathSyncAxisMotionState
EnumPathTransitionState
EnumPathTransitionType
EnumPathTransitionVelocity
EnumPathVelocity
EnumPathVelocityProfile
EnumPathWDirection
EnumPathWDynamics
EnumPathWMode
EnumPersistentDataState
EnumPositioningMode
EnumPositiveNegative
EnumPosValue
EnumProfile
EnumProfileDynamicsLimiting
EnumPulsesEnabledActiveInactiveNotEvaluated
EnumQFAxisOutputEnableMode
EnumQFAxisOutputMode
EnumQFAxisOutputSetMode
EnumRange
EnumRecognitionMode
EnumRedefineMode
EnumRedefineSpecification
EnumRemoveMode
EnumReqActDeactGetStateMode
EnumReqSysFunctMode
EnumRestartAxisCondition
EnumSaveState
EnumScaleSpecification
EnumScalingSpecification
EnumScopeOfIO
EnumSensorDataType
EnumSensorErrorReaction
EnumSensorState
EnumSensorValueBehaviorMode

EnumSetAndGetSafeValue
EnumSetNoSet
EnumSlaveMode
EnumStartStop
EnumStateOfDpSlave
EnumStateOfDpStation
EnumStateOfIO
EnumStateOfTo
EnumStopDriveMode
EnumStopMode
EnumStopSpecification
EnumSyncCommandState
EnumSynchronizingMode
EnumSyncModeCamming
EnumSyncModeGearing
EnumSyncOffModeCamming
EnumSyncOffModeGearing
EnumSyncOffPositionReference
EnumSyncPositionReference
EnumSyncProfileReference
EnumSystemDirect
EnumTControllerActivationModeChangedConfigData
EnumTControllerBinaryIObits
EnumTControllerCommandDestination
EnumTControllerControllerType
EnumTControllerCopyShadow
EnumTControllerDataResetMode
EnumTControllerErrorReaction
EnumTControllerErrorResetMode
EnumTControllerExecutionLevel
EnumTControllerIdentificationAvailableType
EnumTControllerIdentificationModifiedTangentMethodStage
EnumTControllerIdentificationStandardTangentMethodStage
EnumTControllerIdentificationTransitionMode
EnumTControllerIdentificationType

EnumTControllerInputGradientCheckMode
EnumTControllerInputLimitCheckMode
EnumTControllerInputLimitCheckState
EnumTControllerInputType
EnumTControllerIntegrationLimitState
EnumTControllerIntegrationMode
EnumTControllerNextCommand
EnumTControllerOperatingMode
EnumTControllerOutputLimitState
EnumTControllerOutputType
EnumTControllerPlausibilityCheckMode
EnumTControllerPlausibilityState
EnumTControllerProcessControlState
EnumTControllerProcessMode
EnumTControllerRestartActivation
EnumTControllerRestartActivationSetting
EnumTControllerRestartCondition
EnumTControllerSimulationMode
EnumTcpNextCommandMode
EnumToActivationModeSetConfigData
EnumToCommandExecution
EnumToExecutionLevel
EnumToInvalidSysvarAccess
EnumToRestartActivation
EnumToRestartActivationSetting
EnumTorqueLimitUnitType
EnumToSetStateOfTo
EnumTraceState
EnumTrackingState
EnumTransferSuperimposedPosition
EnumTrueFalse
EnumUdpCommunicationMode
EnumUpsBatteryState
EnumUpsState
EnumUserDefaultDirect

EnumUserDefaultYesNo
EnumValueSpecification
EnumValueType
EnumVelocity
EnumXCommandIdState
EnumXCommunicationMode
EnumXNextCommandEnable
EnumYesNo
EQ
error
errorGroup
errorReaction
EXCLUSIVE
EXECUTED
EXISTING
EXIT
EXP
EXPD
EXPRESSION
EXPT
EXTENDED_LOOK_AHEAD
externalEncoderType
extrapolationData
extrapolationValue

F

F_EDGE
F_TRIG
FAILED
FALLING_EDGE
FALLING_EDGES_ONLY
FALSE
fanbattery
FAST
FAULTED

FCTGEN_DISABLED
FCTGEN_ENABLED
FCTGEN_LIMIT_EXCEEDED
FCTGEN_PARAM_VALID
FCTGEN_RUNNING
FCTGEN_START
FCTGEN_START_FG1
FCTGEN_START_FG2
FCTGEN_STARTING
FCTGEN_STOP
FCTGEN_STOPPING
FCTGEN_WAIT_FG1
FCTGEN_WAIT_FG2
FEEDBACK
FEEDBACK_EMERGENCY_STOP
FILTER_AND_CONSTANT_LIMIT
FIND
FINISHED
FIXED_GEAR_DISABLE
fixedGearValue
FLEXIBLE_MOUNTED_SWITCH
FOLLOWING_OBJECT_DISABLE
FOLLOWING_RANGE
followingAxis
followingObjectType
followingRange
followingRangeSettings
FOR
FORCE_AND_INPUT
FORCE_AND_POSITION
FORCE_AND_TIME
FORCE_CONDITION
FORCE_CONTROLLED
FORCE_LIMITED
FORCE_OR_INPUT

FORCE_OR_POSITION
FORCE_OR_TIME
FORCE_POSITION
FORCE_TIME
FORCE_VALUE
forceActualValueSettings
forceControllerData
forceControllerMonitorings
forceControllerSettings
forceLimitingCommand
forceMotionInPositionProfileCommand
forcePositionProfileCommand
forceStateData
forceTimeProfileCommand
FROM
FROM_BACKUP
FROM_FILE
FROM_RAM
FULL
FUNCTION
FUNCTION_BLOCK
FUNCTION_IS_ACTIVATED
FUNCTION_IS_ACTIVE
FUNCTION_IS_WAITING_FOR_ACTIVATION

G

GE
GEARING
GEARING_WITH_FRACTION
GEARING_WITH_RATIO
gearingAdjustments
GLOBAL_EXTREMUM
GOTO
GOTO
GRAY_CODE

GREATER_EQUAL
GT

H

HARDWARE_LIMIT_SWITCH
HARDWARE_LIMIT_SWITCH_NEGATIVE
HARDWARE_LIMIT_SWITCH_POSITIVE
HEAT_AND_COOL_OUTPUT_ZERO
HEAT_OUTPUT_ZERO
HEATING
HIGH
HIGHER_VELOCITY
HOLD_CONNECTION
HOME_POSITION_ENTRY_MOVE
homingCommand
HW_TYPE

I

IDENTIFICATION
identificationModifiedTangentMethodParameter
identificationStandardTangentMethodParameter
IE_01
IE_02
IE_1
IE_2
IF
IMMEDIATELY
IMMEDIATELY_AND_BE_SYNCHRONOUS_AT_MASTER_POSITION
IMMEDIATELY_AND_SLAVE_POSITION
IMMEDIATELY_BY_CAM_TRACK_OUTPUT_INACTIVE
IMMEDIATELY_BY_TIME_PROFILE
IMMEDIATLY
IMPLEMENTATION
IN
IN_ACCELERATION

IN_ACTIVATION
IN_ADAPTION
IN_ALL_CONTROL_MODES
IN_CALCULATION
IN_CLOSED_LOOP_POSITION_CONTROL
IN_CONSTANT_MOTION
IN_DEACTIVATION
IN_DECELERATION
IN_DEFINED_TIME
IN_EXECUTION
IN_INTERPOLATION
IN_INTERPOLATION_BUT_NOT_ON_PROFILE
IN_MOTION
IN_OPERATION
IN_POSITION
IN_STANDSTILL
INACTIVE
INACTIVE_AND_WAITING_FOR_NEXT_CYCLE
INCOMING
INCOMING_ALARM
INCREMENT_BASED
INCREMENT_COUNTER
INCREMENT_DURATION
INITIAL_STEP
INPUT
INPUT_CONDITION
inputDisplayValueParameter
inputFilterParameter
inputGradientCheckParameter
inputLimitCheckParameter
INSERT
INT
INT_TO_BCD
INT_TO_BYTE
INT_TO_DINT

INT_TO_DWORD
INT_TO_LREAL
INT_TO_REAL
INT_TO_SINT
INT_TO_TIME
INT_TO_UDINT
INT_TO_UINT
INT_TO_USINT
INT_TO_WORD
INT_VALUE_TO_BOOL
INTERFACE
INTERFACE_VALUE
INTERNAL_ERROR
INTERNAL_RELATED
internalServoSettings
internalToTrace
INTERNEL_ERROR
INTERPOLATED
interpolation
INTERPOLATION_DONE
INTERRUPTED_COMMAND
INTERVAL_COUNTER
INVALID
INVALID_VALUE
IPO
IPO_2
IPO_FAST
ipoClock
ipoClock_2
ipoClock_fast

J

JMP
JMPC
JMPCN

K

kinematicsData

L

LABEL
LAST_INTERFACE_VALUE
LAST_OPERATION_MODE
LAST_VALID_INTERFACE_VALUE
LAST_VALUE
LD
LDN
LE
LEADING_RANGE
LEADING_RANGE_END
LEADING_RANGE_START
LEADING_RANGE_VALUE
leadingRange
leadingRangeSettings
LEFT
LEN
LENGTH_13
LENGTH_21
LENGTH_25
LESS
lifeSign
LIMIT
LIMIT_EXCEEDED
LIMITING_BY_DIRECT_VALUE
LIMITING_BY_USER_DEFAULT_VALUE
limitsOfPathDynamics
LINEAR
LINEAR_CONVERSIONDATA
LINEAR_MODE
LINEAR_MOTORTYPE
LINEAR_SYSTEM

linearPathCommand
LINT
LittleByteArray_to_AnyType
LITTLEBYTEARRAY_TOANYTYPE
LN
LOAD_CONFIGURED_DATA
LOCAL_EXTREMUM
LOG
LONG_RUN_NEGATIVE
LONG_RUN_POSITIVE
LOW
LOWER_PRIMARY_FAILED
LOWER_SECONDARY_FAILED
LOWER_VELOCITY
lowerPlausibilityParameter
lowerPlausibilityParameterSecondary
LREAL
LREAL_INTERFACE
LREAL_TO_DINT
LREAL_TO_INT
LREAL_TO_REAL
LREAL_TO_SINT
LREAL_TO_STRING
LREAL_TO_UDINT
LREAL_TO_UINT
LREAL_TO_USINT
LREAL_VALUE_TO_BOOL
LREAL_VALUE_TO_BYTE
LREAL_VALUE_TO_DWORD
LREAL_VALUE_TO_WORD
LREALIn1
LREALIn1Default
LREALIn2
LREALIn2Default
LREALIn3

LREALIn3Default
LREALIn4
LREALIn4Default
LREALOut1
LREALOut1Default
LREALOut2
LREALOut2Default
LREALOut3
LREALOut3Default
LREALOut4
LREALOut4Default
LT
LWORD
LWORD_TO_BCD

M

MAINTAIN_PROGRAMMED_DATA
MANDATORY
manualOutputValue
MASTER_RANGE
MASTER_SLAVE_ALARMMESSAGES_1
masterState
matchPosition
MAX
MAX_CONFIGURED_DYNAMICS
MAX_DYNAMICS
MAX_VALUE
mcs
measuredValue1
measuredValue2
MEASURING_AND_MANUAL_OUTPUT
MEASURING_AND_OUTPUT_ZERO
MEASURING_ERROR
MEASURING_INPUT_DISABLE
measuringInputType

MEMORY_CARD_FULL
MEMORY_CARD_SERIAL_NUMBER
MID
MIN
MIN_MAX
minusLimitsOfDynamics
MOD
MODE_1
MODE_2
MODE_CAM
MODE_CAM_AND_ZM
MODE_NO_REFERENCE
MODE_REFERENCE_CAM_AND_EXTERNAL_ZM
MODE_ZM
modeOfDpInterfaceSynchronization
modeOfOperation
MODIFICATION_ACTIVE
MODIFIED_TANGENTMETHOD
modulo
monitorings
MOTION_DONE
MOTION_EMERGENCY_ABORT
MOTION_EMERGENCY_STOP
MOTION_IN_POSITION_LOCKED_PROFILE
MOTION_INTERFACE
MOTION_STOP
motionIn
MotionIn1
MotionIn1Default
MotionIn2
MotionIn2Default
MotionIn3
MotionIn3Default
MotionIn4
MotionIn4Default

MotionInDefault
MotionOut
MotionOut1
MotionOut1Default
MotionOut2
MotionOut2Default
MotionOut3
MotionOut3Default
MotionOutDefault
motionState
motionStateData
motionType
MOVE_WITH_CONSTANT_SPEED
moveCommand
MOVING_WITH_REDUCED_VELOCITY
movingToEndStopCommand
MUL
MULTIME
MUX

N

NE
NEGATIVE
NEGATIVE_ALL_HOMING
NEGATIVE_DIRECTION
NEVER
NEXT_CAM_CYCLE
NEXT_CAM_TRACK_CYCLE
NEXT_IPO_CYCLE
NEXT_MOTION
NEXT_WITH_REFERENCE
NO
NO_ACCESS
NO_ACTIVATE
NO_ALARMMESSAGES

NO_CHANGE
NO_COMMAND_BUFFER_AVAILABLE
NO_CONSTRAINTS
NO_CYCLIC
NO_DP_SEGMENT
NO_HOMING_MODE
NO_POS_MOTION_ACTIVE
NO_REPORTING
NO_REQUEST_TO_ACTIVATE
NO_RESSOURCES_AVAILABLE
NO_RESTART_ACTIVATION
NO_RETAIN_GLOBAL
NO_SET
NO_STORAGE_AVAILABLE
NO_TELEGRAM
NO_TRANSITION
NO_TYPE
NO_VALID_CONFIGURATION_ID
NO_VALID_STATE
NOCYCLIC
NODEF
NOMINAL_VALUE
NON_AVAILABLE
NON_CONTINUOUS
NON_EXCLUSIVE_AND_STARTUP_ACTIVATED
NON_EXCLUSIVE_AND_STARTUP_DEACTIVATED
NON_INCREMENT_BASED
NONE
NOT
NOT
NOT_ACTIVATED
NOT_ADAPTED
NOT_APPLICABLE
NOT_EMPTY
NOT_ENOUGH_RAM

NOT_EVALUATED
NOT_EXISTENT
NOT_EXISTING
NOT_INTERPOLATED
NOT_LIMITED
NOT_MANDATORY
NOT_POSSIBLE
NOT_PRESENT
NOT_VALID
NOTSUPP
numberOfSummarizedTaskOverflow

O

O_K_
OBJECT
OCS
OF
OFF
OFFSET_MOVE
OFFSET_SCALE
OK
ON
ON_MASTER_AND_SLAVE_POSITION
ON_MASTER_POSITION
ON_POSITION
ON_PROFILE
ON_SLAVE_POSITION
ONBOARD
ONLY_POSITION_COMMAND
OPEN_POSITION_CONTROL
operatingMode
OPERATION_AND
OPERATION_OR
OPTIONAL
OPTIONAL_RTC

OR
ORDER_ID
ORN
OUT
OUTGOING
OUTGOING_ALARM
output
OUTPUT_CAM_DISABLE
OUTPUT_PATH_LENGTH
OUTPUT_PATH_LENGTH_ADDITIVE
outputCamType
outputData
outputDefault
outputDerivative
outputDerivativeDefault
outputMonitoring
outputSettings
outputValue
outputValueSecondary
OVER
OVER_POSITION_TO_ENDPOSITION
override

P

P_CONTROLLER
PARABOLIC
PASSIVE_HOMING
path
pathMotion
pathSyncMotion
PD
PERMANENT_STORAGE
persistentDataPowerMonitoring
PID
PID_ACTUAL

PID_CONTROLLER
pidController
pidControllerDefault
pidControllerEffective
pidControllerMonitorings
PINETREE
plusLimitsOfDynamics
PN_1
POINTS_ONLY
POLYNOMIAL
POLYNOMIAL_TRANSITION
polynomialPathCommand
posAxis
posCommand
POSITION
POSITION_AND_DIRECT_NIST_B
POSITION_AND_DYNAMIC_BASED
POSITION_AND_INPUT
POSITION_AND_PROFIDRIVE_ENCODER_NIST_B
POSITION_AND_PROFIDRIVE_NIST_B
POSITION_AND_TIME
POSITION_AT_START_OF_CAMMING
POSITION_BASED
POSITION_CONDITION
POSITION_CONTROLLED
POSITION_LOCKED_PROFILE
POSITION_OR_INPUT
POSITION_OR_TIME
positionBasedMotionInCommand
positioningState
positionTimeProfileCommand
POSITIVE
POSITIVE_ALL_HOMING
POSITIVE_DIRECTION
POWER

precontrol
precontrolValueDefault
precontrolValueIn
PREDEFINED_APPLICATION_EVENTS
PREDEFINED_MODULE_INFORMATION
PRESSURE_DIFFERENCE_MEASUREMENT
PREVIOUS_ACTIVATED
PRIMARY
processedValue
processModeParameter
PROFIDRIVE
PROFIDRIVE_NIST_A
PROFIDRIVE_NIST_B
PROFILE_DONE
PROFILE_END
PROGRAM
PROGRAM
PT1
PT2
PV
PWM
PWMParameter
PWMParameterSecondary

Q

QUADRATIC_MODE

R

R_EDGE
R_TRIG
RAM_DISK_FULL
RANGE_EXTREMUM
RATED
rawValue
READ_AND_WRITE_JOBS

READ_ERROR
READ_JOBS
REAL
REAL_AXIS
REAL_AXIS_WITH_FORCE_CONTROL
REAL_AXIS_WITH_SIGNAL_OUTPUT
REAL_QFAXIS
REAL_QFAXIS_WITH_CLOSED_LOOP_FORCE_CONTROL
REAL_QFAXIS_WITH_OPEN_LOOP_FORCE_CONTROL
REAL_QFAXIS_WITH_OPEN_LOOP_FORCE_CONTROL_ONLY
REAL_TO_DINT
REAL_TO_DWORD
REAL_TO_INT
REAL_TO_LREAL
REAL_TO_SINT
REAL_TO_STRING
REAL_TO_TIME
REAL_TO_UDINT
REAL_TO_UINT
REAL_TO_USINT
REAL_VALUE_TO_BOOL
REAL_VALUE_TO_BYTE
REAL_VALUE_TO_DWORD
REAL_VALUE_TO_WORD
RECTANGLE_TTL
REDUCED_BLENDING_DISTANCE
REDUCTION_TO_ZERO
REDUNDANT
REFER_TO_ACTUAL_SENSOR_VALUE_RESOLUTION
reference
REFERENCE_CAM_AND_EXTERNAL_ZM_PASSIVE
RELATE_SYNC_PROFILE_TO_LEADING_VALUE
RELATE_SYNC_PROFILE_TO_TIME
RELATIVE
RELEASE_DISABLE

REPEAT
REPLACE
REQUEST_ABORT
REQUEST_FALSE
REQUEST_TRUE
reset
RESOLVER
RESOURCE
RESTART_BY_COMMAND
RESTART_BY_SYSVAR_AND_COMMAND
RESTART_NONE
restartActivation
RESULTING
RET
RETAIN
RETC
RETCN
RETURN
REVERSE
RIGHT
RIGHT_MARGIN
RISING_EDGE
RISING_EDGES_ONLY
ROL
ROLL_PICKER
ROR
ROTARY_ARM
ROTATORY
ROTATORY_SYSTEM
RS
RTC
RUN

S

SAFETY

SAME_DIRECTION
SAVE_ABORTED
SAVE_FINISHED
SCARA
SEARCH_ENDPOINT
SEARCH_INFLECTIONPOINT
SEARCH_STARTPOINT
SECONDARY
SEGMENTS_AND_POINTS
SEGMENTS_ONLY
SEL
SEMA
SENSOR_ABSOLUTE
SENSOR_ANALOG
SENSOR_CYCLIC_ABSOLUTE
SENSOR_INCREMENTAL
SENSOR_POSITION_DIFFERENCE_MEASUREMENT
sensorData
sensorMonitoring
sensorSettings
SEQUENTIAL
SERIAL_NUMBER
SERVO
SERVO_FAST
servoControlClock
servoControlClock_fast
servodata
servoMonitoring
servoSettings
servoTaskCycle
servoTaskCycle_fast
SET
SET_ACTUAL_VALUE
SET_DP_ALARM
SET_OFFSET_OF_ABSOLUTE_ENCODER_BY_POSITION

SET_VALUE
setForceCommandValueCommand
setMasterCommand
setpoint
SETPOINT_VALUE
setpointDefault
setpointIn
SETTING_OF_COEFFICIENTS
SETTING_UP
SHADOW
SHADOW_TO_DIRECT
SHL
SHORTEST_WAY
SHR
simulation
SIMULATION_ABORT
SIMULATION_STOP
SIN
SINGLE
singleCamState
SINT
SINT_TO_BCD
SINT_TO_BYTE
SINT_TO_DINT
SINT_TO_DWORD
SINT_TO_INT
SINT_TO_LREAL
SINT_TO_REAL
SINT_TO_UDINT
SINT_TO_UINT
SINT_TO_USINT
SINT_TO_WORD
SINT_VALUE_TO_BOOL
SINUS_1VPP
SINUSOIDAL

SLAVE_ALARMMESSAGES_1
SLAVE_RANGE
SLOW
SMOOTH
SPECIFIC
SPECIFIC_AXIS_MOTION
SPECIFIC_ERROR
SPECIFIC_ID
SPECIFIC_NUMBER
SPECIFIC_PART_OF_RANGE
SPECIFIC_POINT
SPECIFIC_START_DATA
SPEED_CONTROLLED
speedMode
SQRT
SR
SSI_MODE
ST
STANDARD
STANDARD_AND_INVERSE
STANDARD_MOTORTYPE
STANDARD_TANGENTMETHOD
STANDARD_TYPE
STANDSTILL
STANDSTILL_MONITORING_ACTIVE
START
START_POINT
STARTPOINT_ENDPOINT
STARTUP_ACTIVATED
STARTUP_DEACTIVATED
state
STATE_HOMED
STATE_HOMING
STATE_INITIAL
STATE_MACHINE_CONTROL_BY_APPLICATION

STATE_MOVING
STATE_POSABS
STATE_POSITION_END
STATE_POSREL
STATE_STOPPED
STATE_TOACTIVE
stateCammingAdjustments
stateGearingAdjustments
stateOfDpInterfaceSynchronization
stateOfDpSlaveSynchronization
stateSetMasterCommand
STEP
STEPMOTOR
STN
STOP
STOP_ALL_FORMULA
STOP_AND_ABORT
STOP_AND_ABORT_AND_HOLD
STOP_DEVICE
STOP_IN_DEFINED_TIME
STOP_IN_PROFILE_END
STOP_SPECIFIC_FORMULA
STOP_SYMMETRIC_WITH_POSITION
STOP_WHEN_PROFILE_END_REACHED
STOP_WITH_COMMAND_VALUE_ZERO
STOP_WITH_DYNAMIC_PARAMETER
STOP_WITH_MAXIMAL_DECELERATION
STOP_WITHOUT_ABORT
stopEmergencyCommand
STOPPED
STOPPED_WITHOUT_ABORT
STOPU
STRING
STRING_TO_DINT
STRING_TO_LREAL

STRING_TO_REAL
STRING_TO_UDINT
STRUCT
StructAdditionalSensorNumber
StructAdditionObjectMotionIn
StructAdditionObjectMotionOut
StructAlarmId
StructAlarmId_TO_DINT
StructAxisAbsoluteEncoder
StructAxisActorData
StructAxisActorMonitoring
StructAxisActualTorque
StructAxisAdditionalSensorData
StructAxisAdditiveTorque
StructAxisAdditiveTorqueIn
StructAxisDataSetReadWrite
StructAxisDataSetReadWrite_V3_1
StructAxisDataSetReadWrite_V4_1
StructAxisDefaultType
StructAxisDistributedMotion
StructAxisDriveData
StructAxisDriveSafetyExtendedFunctionsInfoData
StructAxisDynamicLimit
StructAxisDynamicQFData
StructAxisExtrapolationData
StructAxisForceActualValueSettings
StructAxisForceControlByConditionCommand
StructAxisForceControlDataSet
StructAxisForceControlDataSet_V3_1
StructAxisForceControlDataSet_V4_1
StructAxisForceControllerData
StructAxisForceControllerMonitorings
StructAxisForceControllerSettings
StructAxisForceLimitingByConditionCommand
StructAxisForceLimitingCommand

StructAxisForceMotionInPositionProfileCommand
StructAxisForcePositionProfileCommand
StructAxisForceStateData
StructAxisForceTimeProfileCommand
StructAxisHomingCommand
StructAxisHomingDefault
StructAxisInvertQOutput
StructAxisInvertSetPointHydraulicType
StructAxisModuloData
StructAxisMotionData
StructAxisMotionInData
StructAxisMotionStateData
StructAxisMoveCommand
StructAxisMovingToEndStopCommand
StructAxisOperatingData
StructAxisOverride
StructAxisPathMotion
StructAxisPathSyncMotion
StructAxisPosCommand
StructAxisPositionBasedMotionInCommand
StructAxisPositioningDefault
StructAxisPositioningState
StructAxisPositionTimeProfileCommand
StructAxisSensorData
StructAxisSensorMonitoring
StructAxisSensorSettings
StructAxisServoData
StructAxisServoMonitoring
StructAxisServoSettings
StructAxisSetForceCommandValueCommand
StructAxisSwLimit
StructAxisSwLimitState
StructAxisTorqueLimitIn
StructAxisTorqueLimitingCommand
StructAxisUserDefaultClamping

StructAxisUserDefaultForceLimiting
StructAxisUserDefaultTorqueLimit
StructAxisVelocityBasedMotionInCommand
StructAxisVelocityLimitingCommand
StructAxisVelocityMotionInPositionProfileCommand
StructAxisVelocityPositionProfileCommand
StructAxisVelocityTimeProfileCommand
StructCamFollowingRange
StructCamInterpolation
StructCammingAdjustments
StructCammingSettings
StructCamRange
StructCamScaleRange
StructCamSettings
StructCamTrackArrayOfSingleCamSettings
StructCamTrackEffectiveData
StructCamTrackSingleCamSettings
StructCamTrackUserDefault
StructCamUserDefault
StructClampingMonitoring
StructControllerDynamic
StructControllerObjectControlDeviation
StructControllerObjectOutputData
StructControllerObjectOutputMonitoring
StructControllerObjectPControllerData
StructControllerObjectPControllerDefault
StructControllerObjectPIDControllerData
StructControllerObjectPIDControllerDefault
StructControllerObjectPIDControllerEffective
StructControllerObjectPIDControllerMonitorings
StructControllerObjectPrecontrolData
StructControllerObjectValueIn
StructControllerType
StructControllerType_V3_1
StructCpuDriveStates

StructCyclicMeasuringEnableCommand
StructDataSetMonitoring
StructDeviceConfigurationData
StructDeviceConfigurationManagement
StructDeviceCpuData
StructDeviceCpuDataRW
StructDeviceDataActivationState
StructDeviceFanBattery
StructDeviceIm0
StructDeviceIm0Softwareversion
StructDeviceIm0Version
StructDeviceImData
StructDeviceStartUp
StructDeviceUpsData
StructDpStationAddressType
StructDynamicComp
StructDynamicComp_V4_1
StructDynamicData
StructDynamicFollowing
StructDynamicFollowing_V4_1
StructEffectiveTaskRuntimeType
StructEncoderEffective
StructEncoderMotionState
StructEncoderNumber
StructEncoderSyncCommand
StructEncoderUserDefault
StructFilterForceControl
StructFixedGearingAdjustments
StructFixedGearingOffset
StructFixedGearingSettings
StructFixedGearMotionIn
StructFixedGearMotionOut
StructFixedGearUserDefault
StructFollowingObjectCurrentSyncPosition
StructFollowingObjectDistributedMotion

StructFollowingObjectMasterDynamics
StructFollowingObjectUserDefault
StructForceControllerData2
StructForceControllerData2_V3_1
StructForceControllerData2_V4_1
StructForceControllerDifference
StructForceControlSwitchingData
StructForceControlUserDefault
StructForceLimitingSwitchingData
StructFormulaObjectDINTIn
StructFormulaObjectDINTOut
StructFormulaObjectLREALIn
StructFormulaObjectLREALOut
StructFormulaObjectMotionIn
StructFormulaObjectMotionOut
StructGear
StructGearingAdjustments
StructGearingOffset
StructGearingSettings
StructInternalServoSettings
StructInternalToTrace
StructLimitsOfPathDynamics
StructMeasuringInputEffective
StructMeasuringInputUserDefault
StructMotionVector
StructOutputCamCounterData
StructOutputCamEffectiveData
StructOutputCamUserDefault
StructOutputControllerOutput
StructOutputData
StructOutputLimits
StructOutputMonitoring
StructOutputSettings
StructPathAbortPosition
StructPathAxisMotionVector

StructPathBcsData
StructPathBlending
StructPathCircularCommand
StructPathData
StructPathDynamicAdaption
StructPathDynamics
StructPathKinematicsData
StructPathLinearCommand
StructPathMcsData
StructPathMotionActual
StructPathMotionVector
StructPathObjectCsFrame
StructPathObjectOcs
StructPathObjectUserDefaultOcs
StructPathOverride
StructPathPolynomialCommand
StructPathSettings
StructPathUserDefault
StructPathVector
StructPathWSettings
StructPDController
StructPendingAlarmState
structPersistentDataPowerMonitoringtype
StructPID_Controller
StructPID_Controller_V3_1
StructPID_Controller_V4_1
StructPIDController
StructPIDController_V3_1
StructPNSyncCounter
StructProcessModel
StructPVController
StructPVController_V3_1
StructQFAxisDataSet
StructQFAxisDataSet_V3_1
StructQFAxisMaxDerivative

StructQFAxisUserDefault
StructRetAllocateToken
StructRetCommandState
StructRetConfiguration
StructRetDeviceCommandState
StructRetDeviceGetStateOfAllDpStations
StructRetDeviceGetStateOfDpSlave
StructRetDeviceGetStateOfIO
StructRetDeviceNameOfStation
StructRetDiagnosticData
StructRetDpSlaveAddress
StructRetEncoderValue
StructRetGetAxisProgrammedTargetPosition
StructRetGetAxisSpecificState
StructRetGetAxisSpecificState2
StructRetGetAxisStoppingData
StructRetGetCamFollowingDerivative
StructRetGetCamSpecificState
StructRetGetCamSpecificState2
StructRetGetCamTrackSpecificState
StructRetGetCircularPathData
StructRetGetCircularPathData_V4_2
StructRetGetCircularPathGeometricData
StructRetGetConfigurationData
StructRetGetDataByToken
StructRetGetDeviceId
StructRetGetDoIndexNumberFromLogAddress
StructRetGetDpStationAddressFromLogDiagnosticAddress
StructRetGetErrorNumberState
StructRetGetExternalEncoderSpecificState
StructRetGetExternalEncoderSpecificState2
StructRetGetFollowingMinMax
StructRetGetForceControlDataSetParameter
StructRetGetForceControlDataSetParameter_V3_1
StructRetGetForceControlDataSetParameter_V4_1

StructRetGetGearAxisSpecificState
StructRetGetGearAxisSpecificState2
StructRetGetGeoAddressFromLogAddress
StructRetGetInOutByte
StructRetGetLinearPathData
StructRetGetLinearPathData_V4_2
StructRetGetLinearPathGeometricData
StructRetGetLogDiagnosticAddressFromStationAddress
StructRetGetMeasuringInputSpecificState
StructRetGetMeasuringInputSpecificState2
StructRetGetMemoryCardId
StructRetGetNextLogAddress
StructRetGetOutputCamSpecificState
StructRetGetOutputCamSpecificState2
StructRetGetPathGeometricData
StructRetGetPathMotionBuffer
StructRetGetPathObjectBcsFromOcsData
StructRetGetPathObjectOcsFromBcsData
StructRetGetPendingAlarms
StructRetGetPendingAlarmState
StructRetGetPolynomialPathData
StructRetGetPolynomialPathData_V4_2
StructRetGetPolynomialPathGeometricData
StructRetGetPosAxisSpecificState
StructRetGetPosAxisSpecificState2
StructRetGetQFAxisDataSetParameter
StructRetGetQFAxisDataSetParameter_V3_1
StructRetGetSegmentIdentification
StructRetGetStateOfAllDpSlaves
StructRetGetStateOfSingleDpSlave
StructRetGetStateOfTo
StructRetGetStationType
StructRetGetToError
StructRetGetValue
StructRetIPConfig

StructRetMotionBuffer
StructRetMotionCommandState
StructRetPathAxesData
StructRetPathAxesPosition
StructRetPathCartesianData
StructRetPathCartesianPosition
StructRetPathMotionCommandState
StructRetReadDriveFaults
StructRetReadDriveMultiParameter
StructRetReadDriveMultiParameterDescription
StructRetReadDriveParameter
StructRetReadDriveParameterDescription
StructRetReadGetAxisDataSet
StructRetReadGetAxisDataSet_V3_1
StructRetReadGetAxisDataSet_V4_1
StructRetReadRecord
StructRetTcpOpenClient
StructRetTcpOpenServer
StructRetTcpReceive
StructRetUdpReceive
StructRetUnitDataSetCommand
StructRetWriteDriveMultiParameter
StructRetWriteDriveParameter
StructRetXCommandState
StructRetXreceive
StructScaleOffset
StructSensorMonitorings
StructStateOfDpStations
StructStateOfIO
StructSyncDynamics
StructSyncMonitoring
StructSyncPositions
StructSyncProfileDefinitions
StructSystemLoad
StructTaskId

StructTaskOverflowType
StructTaskRuntimeType
StructTaskRuntimeValues
StructTCOFctGenOverride
StructTControllerActualDPIDData
StructTControllerActualIdentificationModifiedTangentMethodData
StructTControllerActualIdentificationStandardTangentMethodData
StructTControllerActualInputData
StructTControllerActualInputLimitCheckData
StructTControllerActualInputSingleLimitCheckData
StructTControllerControlRangeParameter
StructTControllerCycleParameter
StructTControllerDPIDParameter
StructTControllerIdentificationModifiedTangentMethodParameter
StructTControllerIdentificationModifiedTangentMethodProcessParameter
StructTControllerIdentificationStandardTangentMethodParameter
StructTControllerIdentificationStandardTangentMethodProcessParameter
StructTControllerIdentificationStaticCondition
StructTControllerInputDisplayValueParameter
StructTControllerInputFilterParameter
StructTControllerInputGradientCheckParameter
StructTControllerInputLimitCheckParameter
StructTControllerInputSingleLimitCheckParameter
StructTControllerOutputValue
StructTControllerPlausibilityParameter
StructTControllerProcessModeParameter
StructTControllerPWMParameter
StructTorqueLimit
StructTraceControl
StructTraceState
StructUserData
StructValueAndDerivedMasterValue
StructValueAndDerivedSlaveValue
StructVelocityGearingSettings
StructXsendDestAddr

SUB
SUB_DATE_DATE
SUB_DT_DT
SUB_DT_TIME
SUB_TIME
SUB_TOD_TIME
SUB_TOD_TOD
SUPERIMPOSED_MOTION
SUPERIMPOSED_MOTION_ACTIVE
SUPERIMPOSED_MOTION_MERGE
SUPERIMPOSED_POS_MOTION_ACTIVE
superimposedMotion
SWITCH_BY_VALUE
SWITCHING_CONDITION_DONE
swLimit
swLimitState
SYMBOL_INFORMATION_NOT_AVAILABLE
SYMBOL_INFORMATION_NOT_FOUND
syncCommand
SYNCHRONIZE_SYMMETRIC
SYNCHRONIZE_WHEN_POSITION_REACHED
SYNCHRONIZED
SYNCHRONIZING
SYNCHRONIZING_NOT_POSSIBLE
synchronizingState
syncMonitoring
syncState
SYSTEM
SYSTEM_DEFINED
systemClock
systemLoad

T

TAN
TARGET_POSITION_MODE

TASK_EXECUTION
TASK_STATE_INVALID
TASK_STATE_LOCKED
TASK_STATE_RUNNING
TASK_STATE_STOP_PENDING
TASK_STATE_STOPPED
TASK_STATE_SUSPENDED
TASK_STATE_WAIT_NEXT_CYCLE
TASK_STATE_WAIT_NEXT_INTERRUPT
TASK_STATE_WAITING
taskRuntime
taskRuntimeMonitoring
TCOFctGenOverride
temperatureControllerType
TEMPORARY_STORAGE
THEN
TIME
TIME_AND_INPUT
TIME_CONDITION
TIME_LOCKED_PROFILE
TIME_OF_DAY
TIME_OR_INPUT
TIME_OUT
TIME_STAMP
TIME_TO_INT
TIME_TO_REAL
TIME_TO_UDINT
TO
TO_CONNECTION
TO_EXECUTION
TO_INTERFACE
TOD
TOF
TON
TORQUE

torqueLimitingCommand
torqueLimitNegative
torqueLimitNegativeIn
torqueLimitPositive
torqueLimitPositiveIn
TOTAL_MOVE
tOutput
TP
TRACE_ABORTED
TRACE_FINISHED
TRACE_INACTIVE
TRACE_MISMATCH
TRACE_NO_TIME
TRACE_PARAM_VALID
TRACE_RUNNING
TRACE_WAIT_FOR_TRIGGER
traceControl
traceState
TRANSFER
TRANSFER_MERGE
TRANSFER_RESET
TRANSFER_STANDSTILL
TRANSIENT_BEHAVIOR_DIRECT
TRANSIENT_BEHAVIOR_WITH_DYNAMICS
TRANSIENT_BEHAVIOR_WITH_NEXT_SYNC
TRANSITION
TRAPEZOIDAL
TRIGGER_OCCURED
TRUE
TRUNC
TYPE
TYPE_REVERSE
TYPE_SWITCH
TYPE_TIME
TYPE_TIME_WITH_MAX_LENGTH

TYPE_WAY
typeOfAxis

U

UDINT
UDINT_TO_BYTE
UDINT_TO_DINT
UDINT_TO_DWORD
UDINT_TO_INT
UDINT_TO_LREAL
UDINT_TO_REAL
UDINT_TO_SINT
UDINT_TO_STRING
UDINT_TO_TIME
UDINT_TO_UINT
UDINT_TO_USINT
UDINT_TO_WORD
UDINT_VALUE_TO_BOOL
UINT
UINT_TO_BYTE
UINT_TO_DINT
UINT_TO_DWORD
UINT_TO_INT
UINT_TO_LREAL
UINT_TO_REAL
UINT_TO_SINT
UINT_TO_UDINT
UINT_TO_USINT
UINT_TO_WORD
UINT_VALUE_TO_BOOL
ULINT
UNDER
UNI_DIRECTION
UNIT
UNIT_NOT_FOUND

UNTIL
UPPER_PRIMARY_FAILED
UPPER_SECONDARY_FAILED
upperPlausibilityParameter
upperPlausibilityParameterSecondary
USELIB
USEPACKAGE
USER
USER_DEFAULT
USER_DEFINED
USER_EVENTS_1
USER_EVENTS_2
USER_STORAGE
userData
userDefault
userDefaultClamping
userDefaultDynamics
userDefaultForceControl
userDefaultForceLimiting
userDefaultHoming
userDefaultOcs
userDefaultPositioning
userDefaultQFAxis
userDefaultTorqueLimiting
USES
USINT
USINT_TO_BYTE
USINT_TO_DINT
USINT_TO_DWORD
USINT_TO_INT
USINT_TO_LREAL
USINT_TO_REAL
USINT_TO_SINT
USINT_TO_UDINT
USINT_TO_UINT

USINT_TO_WORD
USINT_VALUE_TO_BOOL

V

VALID
VALUE
VALUE_LEFT_MARGIN
VALUE_LEFT_MARGIN_WITHOUT_SIGN
VALUE_RIGHT_MARGIN
VALUE_RIGHT_MARGIN_WITHOUT_SIGN
VAR
VAR_ACCESS
VAR_ALIAS
VAR_EXTERNAL
VAR_GLOBAL
VAR_IN_OUT
VAR_INPUT
VAR_OBJECT
VAR_OUTPUT
VAR_TEMP
VELOCITY
VELOCITY_BASED_LIMIT
VELOCITY_CONTROLLED
VELOCITY_GEARING
velocityBasedMotionInCommand
velocityLimitingCommand
velocityMotionInPositionProfileCommand
velocityPositionProfileCommand
velocityTimeProfileCommand
VERSION_MISSMATCH
VIRTUAL_AXIS
VOID

W

WAIT_FOR_CONDITION

WAIT_FOR_DP_CLOCK
WAIT_FOR_VALID
WAITFORCONDITION
WAITING
WAITING_FOR_CHANGE_OF_MASTER_DIRECTION
WAITING_FOR_MERGE
WAITING_FOR_RESTART
WAITING_FOR_SYNC_POSITION
WAITING_FOR_SYNC_START
WAITING_FOR_TRACKING_START
WAITING_FOR_TRIGGER
WAITING_TO_START
WARNING
WHEN_ACCELERATION_DONE
WHEN_AXIS_HOMED
WHEN_AXIS_SYNCHRONIZED
WHEN_BUFFER_READY
WHEN_CAM_TRACK_DONE
WHEN_COMMAND_DONE
WHEN_DATA_ACTIVE
WHEN_ENCODER_SYNCHRONIZED
WHEN_ENDSTOP_REACHED
WHEN_FORCE_CONTROL_ENABLED
WHEN_FORCE_LIMIT_REACHED
WHEN_FORCE_LIMITING_ACTIVATED
WHEN_FUNCTION_DISABLED
WHEN_GEARING_START
WHEN_INTERPOLATION_DONE
WHEN_LIMIT_REACHED
WHEN_LIMITING_COMMAND_ACTIVATED
WHEN_MOTION_DONE
WHEN_PROFILE_DONE
WHEN_TORQUELIMIT_GONE
WHEN_TORQUELIMIT_REACHED
WHEN_TRIGGER_OCCURED

WHILE
WITH
WITH_CAM_TRACK_ACTIVATION
WITH_COMMAND_VALUE_ZERO
WITH_DYNAMIC_RESTRICTION
WITH_DYNAMICS
WITH_INTERPOLATION
WITH_JERK
WITH_MAXIMAL_DECELERATION
WITH_NEXT_SYNCHRONIZING
WITH_RADIUS_AND_ENDPOSITION
WITH_SPECIFIC_AREA
WITH_TIME_LIMIT
WITHOUT_APPROXIMATION
WITHOUT_CHANGE
WITHOUT_DYNAMIC_RESTRICTION
WITHOUT_INTERPOLATION
WITHOUT_INTERPOLATION_AND_HOLD_ACTIVE_CAM
WITHOUT_JERK
WITHOUT_LIMITING
WITHOUT_REDUCTION
WITHOUT_SPECIFIC_AREA
WITHOUT_TIME_LIMIT
WORD
WORD_TO_BCD
WORD_TO_BOOL
WORD_TO_BYTE
WORD_TO_DINT
WORD_TO_DWORD
WORD_TO_INT
WORD_TO_SINT
WORD_TO_UDINT
WORD_TO_UINT
WORD_TO_USINT
WORD_VALUE_TO_LREAL

WORD_VALUE_TO_REAL
WRITE_JOBS
WRITEABLE

X

X_Y
X_Y_Z
XOR
XOR
XORN

Y

Y_Z
YES

Z

Z_X
ZERO_VALUE
ZM_PASSIVE

A.3 Zuordnungstypen bei symbolischer Zuordnung

A.3.1 Zuordnungstypen der Technologieobjekte

Beschreibung

SIMOTION Technologieobjekte und SINAMICS Objekte stellen für die symbolische Zuordnung Schnittstellen und Zuordnungstypen zur Verfügung, die im Folgenden im Einzelnen aufgeführt sind.

Es können nur typgleiche Schnittstellen zugeordnet werden.

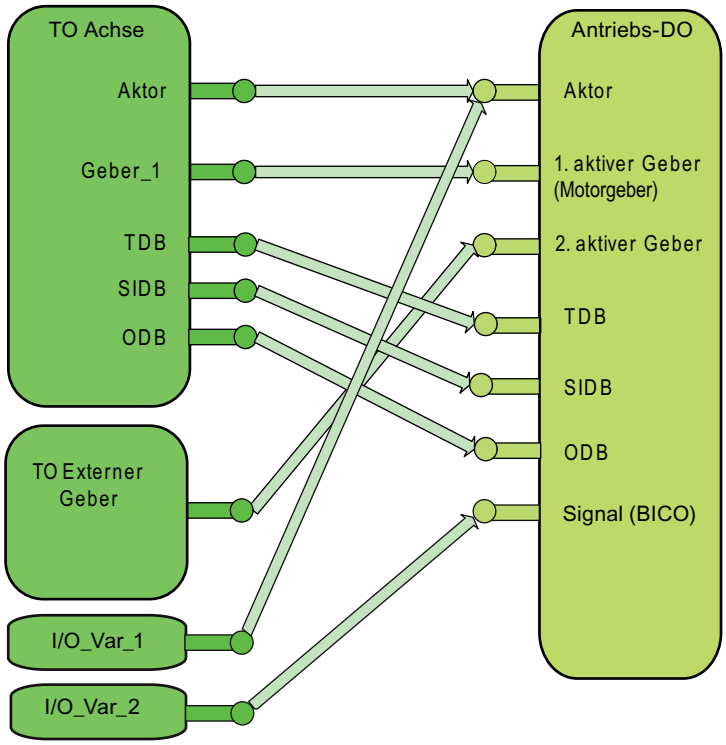


Bild A-1 Beispiel TO - DO Zuordnung

Zuordnungstypen am Technologieobjekt Achse

Bezeichnung des Zuordnungspartners	Zuordnungstyp	Beschreibung
Aktor	• ActorProfidrive • ActorOnboard • WordOutput	Sollwertinterface Achse • über PROFIdrive-Telegramm • onboard • Analogausgang bei Hydraulik
Technologiedaten	TDB	Technologiedaten an der Achse
Safetydaten	SIDB	Safetydaten an der Achse
Geber_<n>	BitInput	Geberinterface des n-ten Gebers
StatusExterneNullmarke	BitInput	Status Externe Nullmarke eines digitalen Antriebs. Bei Inkrementellen Gebern.
Referenznocken	BitInput	Referenznocken
ReferenznockenPassivesReferenzieren	BitInput	Referenznocken für Passives Referenzieren
Bereitstatuskennung	BitInput	Bereitstatuskennung für Analogsensor
Fehlerstatuskennung	BitInput	Fehlerstatuskennung für Analogsensor
Aktualisierungszähler	BitInput	Aktualisierungszähler Analogsensor
Umkehrnocken positiv	BitInput	Umkehrnocken positiv
Umkehrnocken negativ	BitInput	Umkehrnocken negativ
HW-Endchalter positiv	BitInput	Hardwareendschalter positiv
HW-Endchalter negativ	BitInput	Hardwareendschalter negativ
Q-Ventil	WordOutput	Q-Ausgang bei Hydraulik, Q-Ventil
Q-Ventil-Freigabe	BitOutput	Freigabe-Bit für Q-Ventil
P-Ventil	WordOutput	F-Ausgang bei Hydraulik, P-Ventil
P-Ventil-Freigabe	BitOutput	Freigabe-Bit für P-Ventil
Drucksensor	Wordinput	Drucksensor
Digitaleingang	BitInput	Digitaleingang für schnelles Umschalten nach Druckregelung

Zuordnungstypen am TO Externer Geber

Bezeichnung des Zuordnungspartners	Zuordnungstyp	Beschreibung
Geber_1	• sensorPROFIdrive • sensorOnboard • wordInput • dwordInput	Geberschnittstelle des Gebers
StatusExterneNullmarke	bitInput	Status Externe Nullmarke eines digitalen Antriebs. Bei Inkrementellen Gebern.
Referenznocken	bitInput	Referenznocken

Bezeichnung des Zuordnungspartners	Zuordnungstyp	Beschreibung
ReferenznockenPassivesReferenzieren	bitInput	Referenznocken für Passives Referenzieren
Bereitstatuskennung	bitInput	Bereitstatuskennung für Analogsensor
Fehlerstatuskennung	bitInput	Bereitstatuskennung für Analogsensor
Aktualisierungszähler	bitInput	Aktualisierungszähler Analogsensor
Messtaster		Messtastereingang bei lokalen Messen

Zuordnungstypen am TO Nocken

Bezeichnung des Zuordnungspartners	Zuordnungstyp	Beschreibung
Ausgang	• bitOutput • bitOutputWithTimeStamp	Nockenausgang

Zuordnungstypen am TO Messtaster

Bezeichnung des Zuordnungspartners	Zuordnungstyp	Beschreibung
Eingang	bitInputWithTimeStamp	Messtastereingang

Zuordnungstypen am TO TControl

Am TO Sensor sind folgende Schnittstellentypen verfügbar:

Bezeichnung des Zuordnungspartners	Zuordnungstyp	Beschreibung
Temperatureingang	wordInput	Analogeingang Temperatur
PWM-Signal	bitInput	Ausgang PWM-Signal
PWM-Signal_2	bitInput	Ausgang PWM-Signal_2

Zuordnungstypen am Technologieobjekt Sensor

Am TO Sensor ist folgender Schnittstellentyp verfügbar:

Bezeichnung des Zuordnungspartners	Zuordnungstyp	Beschreibung
Analogsensor	wordInput	Analogeingang

A.3.2 SINAMICS-Zuordnungstypen

Beschreibung

Die verschiedenen SINAMICS DO stellen die nachfolgend aufgeführten I/O-Schnittstellen zu den SIMOTION Schnittstellen zur Verfügung.

DO Servo, Vektor und TM41 (Antriebs-DO)

Name	Schnittstellentyp	Beschreibung
Actor	actorProfidrive	Sollwertschittstelle Achse (über PROFIdrive Telegramm)
Encoder_1	sensorPROFIdrive	Geber 1 am Antrieb
Encoder_2	sensorPROFIdrive	Geber 2 am Antrieb
Bico_iw.<BICO> Bico_id.<BICO>	wordInputDriveParameter	Zusätzlich über BICO-Verschaltung übertragener Parameter (eingangsseitig)
Bico_qw.<BICO> Bico_qd.<BICO>	dwordInputDriveParameter	Zusätzlich über BICO-Verschaltung übertragener Parameter (ausgangsseitig)
Informationen zu den Schnittstellen finden Sie unter: Ein-/Ausgänge TM41 in der SCOUT Online Hilfe.		

Geber DO

Name	Schnittstellentyp	Beschreibung
encoder	sensorPROFIdrive	PROFIdrive-Geberschnittstelle
Bico_iw.<BICO> Bico_id.<BICO>	wordInputDriveParameter	Zusätzlich über BICO-Verschaltung übertragener Parameter (eingangsseitig)
Bico_qw.<BICO> Bico_qd.<BICO>	dwordInputDriveParameter	Zusätzlich über BICO-Verschaltung übertragener Parameter (ausgangsseitig)
Informationen zu den Schnittstellen finden Sie unter: Geber DO in der SCOUT Online Hilfe.		

DO TM15/TM17 High Feature

Name	Schnittstellentyp	Beschreibung
DI_<n> MI_<n> DO_<n> CAM_<n>	bitInput bitInputWithTimeStamp bitOutput bitOutputWithTimeStamp	Schnittstellen DI, DO, Messtaster, Nocken für Klemmen X520, X521, X522, Schnittstellentyp ist abhängig von Klemmenkonfiguration TM15: 24 Kanäle, TM17: bis zu 16 Kanäle;
DI_0_15 DI_16_23 DO_0_15 DO_16_23	_wordInputInterfaceType wordOutputInterfaceType	Sammelschnittstellen für Ersatzwerte (Wordinput, Wordoutput)
Informationen zu den Schnittstellen finden Sie unter: Ein-/Ausgänge TM17 Ein-/Ausgänge TM15 in der SCOUT Online Hilfe.		

DO Control Unit

Name	Schnittstellentyp	Beschreibung
DI_<n>	bitInput	Digitaleingänge der Control Unit bei SIMOTION D, CX32/CX32-2 und SINMAICS S110/S120 Control Units
DI_<n> MI_<n> DO_<n> CAM_<n>	bitInput bitInputWithTimeStamp bitOutput bitOutputWithTimeStamp	Bidirektionale Digitalein- /ausgänge DI-Schnittstellen (Bidirektionaler digitaler Eingang) Messtastereingang DO-Schnittstellen (Bidirektionaler digitaler Ausgang) Nockenausgang für die Klemmen
DI_0_15 DO_0_15	_wordInputInterfaceType _wordOutputInterfaceType	Sammelschnittstellen für Ersatzwerte (Wordinput, Wordoutput)
Bico_iw.<BICO> Bico_id.<BICO>	wordInputDriveParameter	Zusätzlich über BICO-Verschaltung übertragener Parameter (eingangsseitig).
Bico_qw.<BICO> Bico_qd.<BICO>	wordOutputDriveParameter	Zusätzlich über BICO-Verschaltung übertragener Parameter (ausgangsseitig)
Informationen zu den Schnittstellen finden Sie unter: Ein-/Ausgänge Control Unit CU320 in der SCOUT Online Hilfe.		

Weitere SINAMICS DO (wie z. B. TM31, TB30 und TM15 DI/DO)

Name	Schnittstellentyp	Beschreibung
Schnittstellen gemäß Standardtelegramm		
Bico_iw.<BICO>	WordInputDriveParameter	Zusätzlich über BICO-Verschaltung übertragener Parameter (eingangsseitig). Die Anzahl der verschaltbaren Parameter ist abhängig vom verwendeten Telegramm.
Bico_qw.<BICO>	WordOutputDriveParameter	Zusätzlich über BICO-Verschaltung übertragener Parameter (ausgangsseitig). Die Anzahl der verschaltbaren Parameter ist abhängig vom verwendeten Telegramm.

A.3.3 Zuordnungstypen in Standardtelegrammen

Beschreibung

Standardtelegramme sind standardisiert und übertragen eine feste Menge von Interfaces mit bestimmten Eigenschaften.

Sie enthalten die folgenden Schnittstellen:

Standardtelegramm 1

Name	Datentyp	Beschreibung
actor.STW1	WORD	Actor-Interface, Steuerwort 1
actor.NSOLL_A	WORD	Actor-Interface, Drehzahlsollwert A 16 Bit
actor.ZSW1	WORD	Actor-Interface, Zustandswort 1
actor.NIST_A	WORD	Actor-Interface, Drehzahlistwert A 16 Bit

Standardtelegramm 2

Name	Datentyp	Beschreibung
actor.STW1	WORD	Actor-Interface, Steuerwort 1
actor.NSOLL_B	DWORD	Actor-Interface, Drehzahlsollwert B 32 Bit
actor.STW2	WORD	Actor-Interface, Steuerwort 2
actor.ZSW1	WORD	Actor-Interface, Zustandswort 1
actor.NIST_B	DWORD	Actor-Interface, Drehzahlistwert B 32 Bit
actor.ZSW2	WORD	Actor-Interface, Zustandswort 2

Standardtelegramm 3

Name	Datentyp	Beschreibung
actor.STW1	WORD	Actor-Interface, Steuerwort 1
actor.NSOLL_B	DWORD	Actor-Interface, Drehzahlsollwert B 32 Bit
actor.STW2	WORD	Actor-Interface, Steuerwort 2
actor.ZSW1	WORD	Actor-Interface, Zustandswort 1
actor.NIST_B	DWORD	Actor-Interface, Drehzahlistwert B 32 Bit
actor.ZSW2	WORD	Actor-Interface, Zustandswort 2
encoder_1.Gn_STW	WORD	Geber 1, Steuerwort
encoder_1.Gn_ZSW	WORD	Geber 1, Zustandswort
encoder_1.Gn_XIST1	DWORD	Geber 1, Lageistwert 1
encoder_1.Gn_XIST2	DWORD	Geber 1, Lageistwert 2

Standardtelegramm 4

Name	Datentyp	Beschreibung
actor.STW1	WORD	Actor-Interface, Steuerwort 1
actor.NSOLL_B	DWORD	Actor-Interface, Drehzahlsollwert B 32 Bit
actor.STW2	WORD	Actor-Interface, Steuerwort 2
actor.ZSW1	WORD	Actor-Interface, Zustandswort 1
actor.NIST_B	DWORD	Actor-Interface, Drehzahlistwert B 32 Bit
actor.ZSW2	WORD	Actor-Interface, Zustandswort 2
encoder_1.Gn_STW	WORD	Geber 1, Steuerwort
encoder_1.Gn_ZSW	WORD	Geber 1, Zustandswort
encoder_1.Gn_XIST1	DWORD	Geber 1, Lageistwert 1
encoder_1.Gn_XIST2	DWORD	Geber 1, Lageistwert 2
encoder_2.Gn_STW	WORD	Geber 2, Steuerwort
encoder_2.Gn_ZSW	WORD	Geber 2, Zustandswort
encoder_2.Gn_XIST1	DWORD	Geber 2, Lageistwert 1
encoder_2.Gn_XIST2	DWORD	Geber 2, Lageistwert 2

Standardtelegramm 5

Name	Datentyp	Beschreibung
actor.STW1	WORD	Actor-Interface, Steuerwort 1
actor.NSOLL_B	DWORD	Actor-Interface, Drehzahlsollwert B 32 Bit
actor.STW2	WORD	Actor-Interface, Steuerwort 2
actor.ZSW1	WORD	Actor-Interface, Zustandswort 1
actor.NIST_B	DWORD	Actor-Interface, Drehzahlistwert B 32 Bit
actor.ZSW2	WORD	Actor-Interface, Zustandswort 2
encoder_1.Gn_STW	WORD	Geber 1, Steuerwort
encoder_1.Gn_ZSW	WORD	Geber 1, Zustandswort
encoder_1.Gn_XIST1	DWORD	Geber 1, Lageistwert 1
encoder_1.Gn_XIST2	DWORD	Geber 1, Lageistwert 2
actor.XERR	DWORD	Actor-Interface, Lageabweichung
actor.KPC	DWORD	Actor-Interface, Lageregler Verstärkungsfaktor

Standardtelegramm 6

Name	Datentyp	Beschreibung
actor.STW1	WORD	Actor-Interface, Steuerwort 1
actor.NSOLL_B	DWORD	Actor-Interface, Drehzahlsollwert B 32 Bit
actor.STW2	WORD	Actor-Interface, Steuerwort 2
actor.ZSW1	WORD	Actor-Interface, Zustandswort 1
actor.NIST_B	DWORD	Actor-Interface, Drehzahlistwert B 32 Bit
actor.ZSW2	WORD	Actor-Interface, Zustandswort 2
actor.MOMRED	WORD	Actor-Interface, Momentenreduzierung
actor.MELDW	WORD	Actor-Interface, Meldungswort
encoder_1.Gn_STW	WORD	Geber 1, Steuerwort
encoder_1.Gn_ZSW	WORD	Geber 1, Zustandswort
encoder_1.Gn_XIST1	DWORD	Geber 1, Lageistwert 1
encoder_1.Gn_XIST2	DWORD	Geber 1, Lageistwert 2
encoder_2.Gn_STW	WORD	Geber 2, Steuerwort
encoder_2.Gn_ZSW	WORD	Geber 2, Zustandswort
encoder_2.Gn_XIST1	DWORD	Geber 2, Lageistwert 1
encoder_2.Gn_XIST2	DWORD	Geber 2, Lageistwert 2

SIEMENS Telegramm 101

Name	Datentyp	Beschreibung
actor.STW1	WORD	Actor-Interface, Steuerwort 1
actor.NSOLL_B	DWORD	Actor-Interface, Drehzahlsollwert B 32 Bit
actor.STW2	WORD	Actor-Interface, Steuerwort 2
actor.ZSW1	WORD	Actor-Interface, Zustandswort 1
actor.NIST_B	DWORD	Actor-Interface, Drehzahlistwert B 32 Bit
actor.ZSW2	WORD	Actor-Interface, Zustandswort 2
actor.MOMRED	WORD	Actor-Interface, Momentenreduzierung
actor.MELDW	WORD	Actor-Interface, Meldungswort

SIEMENS Telegramm 102

Name	Datentyp	Beschreibung
actor.STW1	WORD	Actor-Interface, Steuerwort 1
actor.NSOLL_B	DWORD	Actor-Interface, Drehzahlsollwert B 32 Bit
actor.STW2	WORD	Actor-Interface, Steuerwort 2
actor.ZSW1	WORD	Actor-Interface, Zustandswort 1
actor.NIST_B	DWORD	Actor-Interface, Drehzahlistwert B 32 Bit
actor.ZSW2	WORD	Actor-Interface, Zustandswort 2
actor.MOMRED	WORD	Actor-Interface, Momentenreduzierung
actor.MELDW	WORD	Actor-Interface, Meldungswort
encoder_1.Gn_STW	WORD	Geber 1, Steuerwort
encoder_1.Gn_ZSW	WORD	Geber 1, Zustandswort
encoder_1.Gn_XIST1	DWORD	Geber 1, Lageistwert 1
encoder_1.Gn_XIST2	DWORD	Geber 1, Lageistwert 2

SIEMENS Telegramm 103

Name	Datentyp	Beschreibung
actor.STW1	WORD	Actor-Interface, Steuerwort 1
actor.NSOLL_B	DWORD	Actor-Interface, Drehzahlsollwert B 32 Bit
actor.STW2	WORD	Actor-Interface, Steuerwort 2
actor.ZSW1	WORD	Actor-Interface, Zustandswort 1
actor.NIST_B	DWORD	Actor-Interface, Drehzahlistwert B 32 Bit
actor.ZSW2	WORD	Actor-Interface, Zustandswort 2
actor.MOMRED	WORD	Actor-Interface, Momentenreduzierung
actor.MELDW	WORD	Actor-Interface, Meldungswort
encoder_1.Gn_STW	WORD	Geber 1, Steuerwort
encoder_1.Gn_ZSW	WORD	Geber 1, Zustandswort
encoder_1.Gn_XIST1	DWORD	Geber 1, Lageistwert 1
encoder_1.Gn_XIST2	DWORD	Geber 1, Lageistwert 2
encoder_2.Gn_STW	WORD	Geber 2, Steuerwort
encoder_2.Gn_ZSW	WORD	Geber 2, Zustandswort
encoder_2.Gn_XIST1	DWORD	Geber 2, Lageistwert 1
encoder_2.Gn_XIST2	DWORD	Geber 2, Lageistwert 2

SIEMENS Telegramm 105

Name	Datentyp	Beschreibung
actor.STW1	WORD	Actor-Interface, Steuerwort 1
actor.NSOLL_B	DWORD	Actor-Interface, Drehzahlsollwert B 32 Bit
actor.STW2	WORD	Actor-Interface, Steuerwort 2
actor.ZSW1	WORD	Actor-Interface, Zustandswort 1
actor.NIST_B	DWORD	Actor-Interface, Drehzahlistwert B 32 Bit
actor.ZSW2	WORD	Actor-Interface, Zustandswort 2
actor.MOMRED	WORD	Actor-Interface, Momentenreduzierung
actor.MELDW	WORD	Actor-Interface, Meldungswort
encoder_1.Gn_STW	WORD	Geber 1, Steuerwort
encoder_1.Gn_ZSW	WORD	Geber 1, Zustandswort
encoder_1.Gn_XIST1	DWORD	Geber 1, Lageistwert 1
encoder_1.Gn_XIST2	DWORD	Geber 1, Lageistwert 2
actor.XERR	DWORD	Actor-Interface, Lageabweichung
actor.KPC	DWORD	Actor-Interface, Lageregler Verstär-kungsfaktor

SIEMENS Telegramm 106

Name	Datentyp	Beschreibung
actor.STW1	WORD	Actor-Interface, Steuerwort 1
actor.NSOLL_B	DWORD	Actor-Interface, Drehzahlsollwert B 32 Bit
actor.STW2	WORD	Actor-Interface, Steuerwort 2
actor.ZSW1	WORD	Actor-Interface, Zustandswort 1
actor.NIST_B	DWORD	Actor-Interface, Drehzahlistwert B 32 Bit
actor.ZSW2	WORD	Actor-Interface, Zustandswort 2
actor.MOMRED	WORD	Actor-Interface, Momentenreduzierung
actor.MELDW	WORD	Actor-Interface, Meldungswort
encoder_1.Gn_STW	WORD	Geber 1, Steuerwort
encoder_1.Gn_ZSW	WORD	Geber 1, Zustandswort
encoder_1.Gn_XIST1	DWORD	Geber 1, Lageistwert 1
encoder_1.Gn_XIST2	DWORD	Geber 1, Lageistwert 2
encoder_2.Gn_STW	WORD	Geber 2, Steuerwort
encoder_2.Gn_ZSW	WORD	Geber 2, Zustandswort
encoder_2.Gn_XIST1	DWORD	Geber 2, Lageistwert 1
encoder_2.Gn_XIST2	DWORD	Geber 2, Lageistwert 2
actor.XERR	DWORD	Actor-Interface, Lageabweichung
actor.KPC	DWORD	Actor-Interface, Lageregler Verstär-kungsfaktor

Standardtelegramm 81

Name	Datentyp	Beschreibung
encoder_1.STW2	WORD	Steuerwort 2
encoder_1.ZSW2	WORD	Zustandswort 2
encoder_1.Gn_STW	WORD	Geber 1, Steuerwort
encoder_1.Gn_ZSW	WORD	Geber 1, Zustandswort
encoder_1.Gn_XIST1	DWORD	Geber 1, Lageistwert 1
encoder_1.Gn_XIST2	DWORD	Geber 1, Lageistwert 2

Standardtelegramm 82

Name	Datentyp	Beschreibung
encoder_1.STW2	WORD	Steuerwort 2
encoder_1.ZSW2	WORD	Zustandswort 2
encoder_1.Gn_STW	WORD	Geber 1, Steuerwort
encoder_1.Gn_ZSW	WORD	Geber 1, Zustandswort
encoder_1.Gn_XIST1	DWORD	Geber 1, Lageistwert 1
encoder_1.Gn_XIST2	DWORD	Geber 1, Lageistwert 2
encoder_1.NIST_A	WORD	Geber 1, Drehzahlistwert A 16 Bit

Standardtelegramm 83

Name	Datentyp	Beschreibung
encoder_1.STW2	WORD	Steuerwort 2
encoder_1.ZSW2	WORD	Zustandswort 2
encoder_1.Gn_STW	WORD	Geber 1, Steuerwort
encoder_1.Gn_ZSW	WORD	Geber 1, Zustandswort
encoder_1.Gn_XIST1	DWORD	Geber 1, Lageistwert 1
encoder_1.Gn_XIST2	DWORD	Geber 1, Lageistwert 2
encoder_1.NIST_B	DWORD	Geber 1, Drehzahlistwert B 32 Bit

Struktur der Telegrammerweiterung TDB

Name	Datentyp	Beschreibung
M_ADD	WORD	Zusatzdrehmoment
M_LIMIT_PLUS	WORD	Momentengrenze positiv
M_LIMIT_MINUS	WORD	Momentengrenze negativ
M_ACTUAL	WORD	Aktuelles Moment

Struktur der Telegrammerweiterung SIDB

Name	Datentyp	Beschreibung
S_ZSW1B	WORD	Safety Zustandswort
S_SPEED_LIMIT	DWORD	Geschwindigkeitsgrenzwertsignal

A.3.4 Bezeichnungen für die Zuordnungstypen

Beschreibung

Folgende Zuordnungstypen sind für SIMOTION definiert:

Achs- und Geberschnittstellentypen

Tabelle A- 6 Achs-Schnittstellentypen

Name in ST Nomenklatur	Name in der Adressliste
_actorProfidriveInterfaceType	ActorProfidrive
_actorOnboardInterfaceType	ActorOnboard
_wordOutputInterfaceType	WordOutput

Tabelle A- 7 Geber-Schnittstellentypen

Name in ST Nomenklatur	Name in der Adressliste
_sensorProfidriveInterfaceType	SensorProfidrive
_sensorOnboardInterfaceType	SensorOnboard
_wordInputInterfaceType	WordInput
_dwordInputInterfaceType	DWordInput

Tabelle A- 8 Datenblöcke (erweiterte Kommunikation)

Name in ST Nomenklatur	Name in der Adressliste
_tdbInterfaceType	TDB
_sidbInterfaceType	SIDB

Nocken Messtaster

Name in ST Nomenklatur der Nocken-Zuordnungstypen	Name in der Adressliste
_bitOutputInterfaceType	BitOutput
_bitOutputWithTimeStampInterfaceType	BitOutputWithTimeStamp

Name in ST Nomenklatur der Messtaster Zuordnungstypen	Name in der Adressliste
_bitInputWithTimeStampInterfaceType	BitInputWithTimeStamp

Standard-I/O-Schnittstellen

Name in ST Nomenklatur	Name in der Adressliste
_bitInputInterfaceType	BitInput
_bitOutputInterfaceType	BitOutput
_byteInputInterfaceType	ByteInput
_byteOutputInterfaceType	ByteOutput
_wordInputInterfaceType	WordInput
_wordOutputInterfaceType	WordOutput
_dwordInputInterfaceType	DWordInput
_dwordOutputInterfaceType	DWordOutput

SINAMICS-Parameter-Schnittstellentypen

Name in ST Nomenklatur	Name in der Adressliste
_wordInputDriveParameterInterfaceType	WordInputDriveParameter
_dwordInputDriveParameterInterfaceType	DWordInputDriveParameter
_wordOutputDriveParameterInterfaceType	WordOutputDriveParameter
_dwordOutputDriveParameterInterfaceType	DWordOutputDriveParameter

A.3.5 Syntax der Zuordnungen

Die symbolischen Zuordnungen für I/O-Variable können Sie auch über Textfelder in der Adressliste und TO Konfigurationsdialogen eingeben (neben die Schaltflächen zum Aufrufen des Zuordnungsdialogs). Die dazu notwendige Syntax finden Sie in der nachfolgenden Tabelle.

Tabelle A- 9

Verschaltungsziel	Syntax
Onboard-I/O von SIMOTION D4x5/D4x5-2, Klemme X122/X132 (SINAMICS Integrated);	<device>.<DO_name>.<interface_name> [<terminal_name>] **Beispiel D4x5-2:** SINAMICS_Integrated.Control_Unit.DO_8 [DI/DO 8, X122.9] -> Digital-Ausgang "DO_8" auf Gerät "SINAMICS_Integrated", Drive Object "Control_Unit", Klemme "X122.9", Bezeichnung des Pins "DI/DO 8"
Onboard-I/O von SIMOTION D4x5-2, Klemme X142	<device>.<<interface-name> **Beispiel:** D455-2, X142: D455.CAM_6 [DI/DO 6, X142.12] -> Ausgang mit Nocken-Qualität "CAM_6" auf Gerät "D455", Klemme "X142.12", Bezeichnung des Pins "DI/DO 6"
Onboard-I/O SINAMICS S120 CU3xx und Controller Extension CX32/32-2	<device>.<DO_name>.<interface_name> [<terminal_name>] **Beispiel:** SIMOTION_CX32.Control_Unit.DI_1 [DI 1, X122.2] S120_CU320.Control_Unit.DI_1 [DI 1, X122.2] -> Digital-Eingang "DI_1" auf Gerät "SIMOTION_CX32" bzw. "S120_CU320", Drive Object "Control_Unit", Klemme "X122.2", Bezeichnung des Pins "DI 1"
TM15, TM17, TM41	<device>.<DO_name>.<interface_name> [<terminal_name>] **Beispiele:** TM15: SINAMICS_Integrated.TM15.MI_8 [DI/DO 8, X521.2] -> Eingang mit Messtaster-Qualität "MI_8" auf Gerät "SINAMICS_Integrated", Drive Object "TM15", Klemme "X521.2", Bezeichnung des Pins "DI/DO 8" TM41: SINAMICS_Integrated.TM41.AI_0 [AI 0, X523] -> Analog-Eingang "AI_0" auf Gerät "SINAMICS_Integrated", Drive Object "TM41", Klemme "X523", Bezeichnung des Pins "AI 0"

Verschaltungsziel	Syntax
TB30, TM31, TM15 DI/DO	<device>.<DO_name>.<interface_name> [<terminal_name>] **Beispiele:** TB30: SINAMICS_Integrated.TB30.AI_0 [AI 0, X482.1, X482.2] -> Analog-Eingang "AI_0" auf Gerät "SINAMICS_Integrated", Drive Object "TB30", Klemmen "X482.1" bzw. "X482.2", Bezeichnung des Pins "AI 0" TM31: SINAMICS_Integrated.TM31.DI_2 [DI 2, X520.3] -> Digital-Eingang "DI_2" auf Gerät "SINAMICS_Integrated", Drive Object "TM31", Klemme "X520.3", Bezeichnung des Pins "DI 2" TM15 DI/DO: SINAMICS_Integrated.TM15DIDO.DO_23 [DI/DO 23, X522.9] -> Digital-Ausgang "DO_23" auf Gerät "SINAMICS_Integrated", Drive Object "TM15DIDO", Klemme "X522.9", Bezeichnung des Pins "DI/DO 23"
freie Telegramm-Erweiterungen SINAMICS	<device>.<do-name>.BICO_xx.<Parameternummer> mit (xx = IW, QW, ID, QD) bei Verschaltung auf einen Parameter im gleichen Drive Object (DO), an das sich das Telegramm richtet <device>.<do-name>. BICO_xx.<do-name2>#<Parameternummer> bei Verschaltung auf einen Parameter im Drive Object (DO) <do-name2>, wobei das Telegramm an das DO <do-name> projektiert ist
Telegramm-Bestandteile	<device>.<DO-name>.<interface-name>.<sub-interface-name> **Beispiele:** SINAMICS_Integrated.Axis_Red.actor.STW1 SINAMICS_Integrated.Axis_Blue.actor.ZSW1 SINAMICS_Integrated.Axis_Red.encoder_1.Gx_ZSW
Telegramm der Einspeiseeinheit	Ist die automatische Telegramm-Bestimmung aktiviert, wird das Telegramm für die Einspeisung automatisch festgelegt (PROFIdrive-Telegramm 370). In der Adressliste können Sie nun für das Zustands- und Steuerwort (ZSW1/STW1) der Einspeisung über symbolische Zuordnung zwei I/O-Variablen im WORD-Format anlegen. Diese I/O-Variablen benötigen Sie für den **FB _LineModule_control** zur Ansteuerung des Line Modules. Weitere Informationen siehe Funktionshandbuch *Standardfunktion für SINAMICS S120 Line Modules*
Onboard-I/O SIMOTION C	<device>.<interface-name> **Beispiele:** C240_1.AnalogActor_3 [ANALOG/STEPPER 3, X2] C240_1.Encoder_2 [ENCODER 2, X4] C240_1.DI_0 [I 0, X1.28] C240_1.DO_7 [Q7, X.19] C240_1.M1 [M1, X1.26] C240_1.B2 [B2, X1.23]

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Z