Karl-Heinz John · Michael Tiegelkamp

SPS-Programmierung mit IEC 61131-3

Konzepte und Programmiersprachen, Anforderungen an Programmiersysteme, Entscheidungshilfen

Dritte, neubearbeitete Auflage

Mit 139 Abbildungen

Springer

KARL-HEINZ JOHN
Irrlrinnig 13
D-91301 Forchheim

MICHAEL TIEGELKAMP
Kurpfalzstr. 34
D-90602 Pyrbaum

ISBN 3-540-66445-9 Springer-Verlag Berlin Heidelberg New York

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

John, Karl-Heinz:

SPS-Programmierung mit IEC 61131-3: Konzepte und Programmiersprachen, Anforderungen an Programmiersysteme, Entscheidungshilfen / Karl-Heinz John; Michael Tiegelkamp. – 3. überarbeitete Auflage. – Berlin; Heidelberg; New York; Barcelona; Hongkong; London; Mailand; Paris; Singapur; Tokio: Springer, 2000
ISBN 3-540-66445-9

(VDI-Buch)

Einbandentwurf: Atelier Struve & Partner, Heidelberg
Satzerstellung: Reprofertige Vorlagen von Autoren
Gedruckt auf säurefreiem Papier SPIN: 10741006 62/3020 5 4 3 2 1

Inhaltsverzeichnis

1 Einleitung

Durch den rasanten Fortschritt bei Leistung und Miniaturisierung in der Mikrotechnologie erschließt sich die Speicherprogrammierbare Steuerung (SPS) immer neue Märkte. Speziell konzipierte Regel- und Steuerhardware oder PC-basierte Controller, erweitert um realzeitfähige Hard- und Software, kontrollieren höchst komplexe Automatisierungsprozesse.

Die SPSen decken durch ihre unterschiedlichen Ausprägungen ein weites Aufgabenspektrum ab — dies reicht von kleinen Knotenrechnern in Netzwerken und dezentral eingesetzten Kompaktgeräten bis hin zur modularen, fehlertoleranten Hochleistungs-SPS. Sie unterscheiden sich in Leistungsmerkmalen wie Verarbeitungsgeschwindigkeit, Vernetzungsfähigkeit oder das Angebot verwendbarer Peripherie-Baugruppen.

Der Begriff SPS sei im weiteren als Technologie verstanden; dies weicht vom amerikanischen Begriff PLC (Programmable Logic Controller) ab, der darunter lediglich eine bestimmte Hardware-Architektur — im Gegensatz zu den PC-Controllern (Soft Logic Array) — versteht.

Dieses Spektrum der Leistungsfähigkeit seitens der Hardware erfordert eine entsprechende Unterstützung durch geeignete Programmierwerkzeuge, die eine kostengünstige und qualitätsbewußte Erstellung einfacher wie umfangreicher Anwender-Programme ermöglicht:

- Gleichzeitige Unterstützung mehrerer SPS-Programmiersprachen,
- Änderung der Programme „on-line“ in der SPS,
- Rückdokumentation der Programme aus der SPS,
- Wiederverwendbarkeit von SPS-Programm-Bausteinen,
- Test und Simulation der Anwenderprogramme „off-line“,
- Integrierte Projektierungs- und Inbetriebnahme-Werkzeuge,
- Qualitätssicherung, Projektdokumentation,
- Verwendung von Systemen mit offenen Schnittstellen,
- u.v.a.m.

Auf der Basis von PCs entstanden in den letzten 10 Jahren immer leistungsfähigere SPS-Programmierwerkzeuge.

Die bisher verwendeten klassischen SPS-Programmiersprachen wie Anweisungsliste, Kontaktplan oder Funktionsplan stoßen an ihre Grenzen. Die Anwender fordern einheitliche, herstellerübergreifende Sprachkonzepte, höhere Programmiersprachen und Entwicklungswerkzeuge, wie sie im Bereich der PC-Welt bereits seit längerem üblich sind.

Mit Einführung der internationalen Norm IEC 1131 (inzwischen unbenannt in IEC 61131) wurde eine Basis für eine einheitliche SPS-Programmierung geschaffen, die moderne Konzepte der Software-Technologie berücksichtigt.

1.1 Gegenstand des Buchs

Dieses Buch hat zum Ziel, eine verständliche Einführung in die Konzepte und Sprachen der Norm IEC 61131 zu geben. Dabei werden einfache Beispiele verwendet, um die Ideen und Anwendung der neuen SPS-Programmiersprachen zu erklären. Ein größeres Beispiel-Programm faßt jeweils die Ergebnisse eines Abschnitts nochmals zusammen.

Das Buch versteht sich als hilfreiche Einführung und Erklärungshilfe für Leute in Ausbildung und Praxis, die die Möglichkeiten der neuen Norm kennenlernen und nutzen möchten.

Es setzt geringe Kenntnisse im Umgang mit Personal Computern und Grundkenntnisse im Bereich der SPS-Technik voraus (siehe dazu [Grötsch-96], [Berger-87] oder [Berger-96]). Auch erfahrene SPS-Programmierer finden hier Informationen, die ihnen den Umstieg auf Programmiersysteme der neuen Generation erleichtern. Dazu werden die Begriffswelten bisheriger Systeme mit der Welt der IEC-Programmierung zusammengebracht, sowie die Vorteile durch die Programmierung nach IEC erläutert.

Berufsschüler und Studenten erhalten ein Nachschlagewerk, das ihnen systematisch das Erlernen des neuen Programmierstandards erleichtert.

Mit Hilfe des „Einkaufsberaters“ kann der Leser zusätzlich SPS-Programmiersysteme selbst und individuell beurteilen, siehe beiliegende CD.

Formale Inhalte und Aufbau der IEC werden praxisgerecht dargestellt, schwierige Sachverhalte im Zusammenhang erläutert und Interpretationsspielräume sowie Erweiterungsmöglichkeiten aufgezeigt.

Dieses Buch soll helfen, dem Leser konkrete Antworten auf folgende Fragen zu geben:

- wie erfolgt die Programmierung nach IEC 61131? Was sind die wesentlichen Ideen der Norm, wie werden sie praxisgerecht umgesetzt?

- wo liegen die Vorteile der internationalen Norm IEC 61131 gegenüber bisherigen (nationalen) SPS-Programmierstandards? Worin bestehen die Neuerungen und Chancen?
- worauf hat der Anwender zu achten, wenn er zu einem Programmiersystem der neuen Generation wechseln möchte?
- was müssen heutige Programmiersysteme leisten, um konform zur IEC 61131 zu sein und diese neue Norm zu erfüllen?
- worauf sollte man als Anwender bei der Auswahl eines SPS-Programmiersystems mindestens achten: welche Kriterien sind für die Leistungsfähigkeit von Programmiersystemen ausschlaggebend?

Kapitel 2 stellt die drei Grundbausteine der Norm vor: *Programm, Funktion und Funktionsbaustein*. Ein Einführungsbeispiel, das die wichtigsten Sprachelemente der Norm beinhaltet und gleichzeitig einen Überblick über ihre Programmiermethodik bietet, schafft einen Einstieg in die Begriffswelt der IEC 61131.

Kapitel 3 beschreibt die *gemeinsamen Sprachelemente* der fünf Programmiersprachen, sowie die Möglichkeit der Datenbeschreibung mit Hilfe von Deklarationen.

Es schließen sich in *Kapitel 4* die *fünf Programmiersprachen* der IEC 61131 an, die ausführlich erläutert und jeweils durch ein umfangreicheres Beispiel ergänzt werden.

Die Mächtigkeit der IEC 61131 beruht u.a. auf einer einheitlichen Beschreibung häufig verwendeter Bausteine, den sogenannten *Standard-Funktionen* und *-Funktionsbausteinen*. Ihre Definition und Verwendungsmöglichkeiten sind in *Kapitel 5* beschrieben.

Nach der Programmierung erfolgt mittels der *Konfiguration* eine Zuweisung der Programme und Daten an die Eigenschaften und Hardware der ausführenden SPS. Dies ist in *Kapitel 6* zu finden.

Der SPS-Markt entwickelte sich zu einer Technologie mit sehr spezifischen Anforderungen. Diese *Besonderheiten der Programmerstellung* mit einer SPS sowie die Umsetzung mit den neuen Mitteln der IEC 61131 sind das Thema von *Kapitel 7*.

Kapitel 8 faßt die wichtigsten Eigenschaften der Norm aus Kapitel 2 bis 7 zusammen. Die wesentlichen Vorteile der Norm und konformer Programmiersysteme können nochmals nachgelesen werden.

Kapitel 9 stellt die künftige Norm *IEC 61499* vor, die verteilte Automatisierungsprozesse behandelt. Sie basiert auf IEC 61131-3, erweitert diese jedoch ganz erheblich um eine Sichtweise, die sich aus den Anforderungen an hohe Parallelität und Dezentralisierung moderner Automatisierungsaufgaben ergibt.

Kapitel 10 erläutert die Benutzung der mitgelieferten CD. Sie beinhaltet Programmierbeispiele dieses Buchs, einen Einkaufsberater in Tabellenform, sowie ablauffähige Demo-Versionen zweier IEC -Programmiersysteme.

Die sich anschließenden *Anhänge* liefern weitere Detail-Informationen zu den Kapiteln.

Das Glossar aus *Anhang J* stellt alphabetisch die wichtigsten Begriffe dieses Buchs mit einer Kurzerklärung zusammen.

Anhang K enthält das Literaturverzeichnis, in dem neben Büchern auch Referenzen auf Fachaufsätze zum Thema IEC 61131-3 enthalten sind.

In *Anhang L* ist ein allgemeiner Index zu finden, der zum Auffinden von Stichworten sehr hilfreich sein kann.

1.2 Die Norm IEC 61131

Die mehrteilige Norm IEC 61131 faßt Anforderungsdefinitionen für moderne SPS-Systeme zusammen. Die Anforderungen betreffen die SPS-Hardware und das Programmiersystem.

Die Norm umfaßt sowohl gängige Konzepte der bisherigen SPS-Programmierung als auch Erweiterungen um neue Programmiermethoden.

Die IEC 61131-3 sieht sich als *Richtlinie zur SPS-Programmierung*, nicht als starr bindende Vorschrift. Die enorme Menge der Festlegungen läßt erwarten, daß die Programmiersysteme nur einen mehr oder weniger großen Teil der Norm realisieren werden. Diesen Umfang hat der SPS-Hersteller zu dokumentieren: will er normkonform sein, hat er nachzuweisen, in welchen Teilen er die Norm erfüllt oder nicht.

Dazu beinhaltet die Norm 62 Tabellen (engl.: feature tables) mit Anforderungsdefinitionen, die vom Hersteller jeweils mit Vermerken („ist erfüllt; ist nicht implementiert; folgende Teile sind erfüllt:...“) zu versehen sind.

Die Norm bildet so den *Maßstab*, an dem sich sowohl Hersteller als auch Anwender orientieren können, um festzustellen, in welchem Maß sich ein Programmiersystem an den Standard hält, d.h. konform zur IEC 61131-3 ist.

Zum Nachweis der Konformität definiert die PLCopen, Abschn. 1.3, weitere Tests für Konformitätsklassen, die von unabhängigen Instituten durchgeführt werden können.

Die Norm wurde von der Arbeitsgruppe SC65B WG7 (anfangs: SC65A WG6) der internationalen Normungsgruppe IEC (International Electrotechnical Commission) erarbeitet, die sich aus Vertretern verschiedener SPS-Hersteller, Softwarehäusern und Anwendern zusammensetzt. Dies hat den Vorteil, daß sie von den meisten SPS-Herstellern als Richtlinie akzeptiert wird.

1.2.1 Ziele und Nutzen der Norm

Durch die ständig steigende Komplexität der SPS-Systeme wachsen die Kosten für:

- die Ausbildung der Anwendungsprogrammierer,
- die Erstellung der immer größeren Programme,
- die Implementierung immer komplexerer Programmiersysteme.

Die SPS-Programmiersysteme folgen, zeitlich verzögert, diesem Trend des Software-Massenmarkts der PC-Welt. Wie dort läßt sich der Kostendruck vor allem durch Vereinheitlichung und Synergien mildern. Durch eine deutliche Annäherung der bisher stark herstellerspezifischen Systeme profitieren sowohl Hersteller als auch Anwender von der IEC 61131-3.

Hersteller (SPS- Hardware und -Software).
Mehrere Hersteller können *gemeinsam* in die – mit der heute vom Markt geforderten Funktionalität – viele Millionen DM teure Software investieren.

Die Grundausprägung eines Programmiersystems ist durch die Norm weitgehend festgelegt. Basissoftware wie Editoren kann, bis auf spezifische Teile wie Codegeneratoren oder „on-line“-Module, gemeinsam genutzt werden. Die Marktdifferenzierung erfolgt über sogenannte Zusatzbausteine zum Grundpaket, die in speziellen Marktsegmenten benötigt werden, sowie über die SPS-Hardware.

Durch die Einführung der Norm findet zum Thema Programmierung derzeit ein starker Erfahrungs- und Produktaustausch zwischen den Hard- und Softwareherstellern statt. Durch den Zukauf von Fremdprodukten lassen sich die Entwicklungskosten wesentlich verringern. Die Fehleranfälligkeit neu entwickelter Software wird durch die Verwendung bereits erprobter Software stark reduziert.

Die Gefahr einer Fehlentwicklung (das System entspricht nicht dem Marktbedürfnis) ist deutlich geringer. Die Norm gibt den Weg vor, der dem Anwender auch von anderen IEC 61131-3 Produkten bekannt ist.

Der Entwicklungsaufwand heutiger Programmierwerkzeuge ist durch die geforderte Funktionalität deutlich gewachsen. Durch den Zukauf von Softwarekomponenten oder Komplettsystemen kann die Zeit bis zur Markteinführung wesentlich verkürzt werden, so daß mit der raschen Hardware-Entwicklung Schritt gehalten werden kann.

Anwender.
Anwender arbeiten oft gleichzeitig mit SPS-Systemen unterschiedlicher Hersteller. Mußten bislang Mitarbeiter mehrere Programmier-Ausbildungen durchlaufen, beschränkt sich dies bei IEC 61131-3- Systemen auf die Bedienspezifika der einzelnen Programmiersysteme und zusätzliche Besonderheiten der SPSen. Systemspezialisten und Ausbildungspersonal können daher eingespart, SPS-Programmierer flexibler eingesetzt werden.

Die Anforderungsdefinitionen der Norm erleichtern die Auswahl geeigneter Programmiersysteme, denn sie sind dadurch vergleichbarer geworden.

Ist auch in nächster Zeit nicht abzusehen, daß vollständige Anwenderprogramme zwischen unterschiedlichen SPS-Systemen ausgetauscht werden können, so entsprechen sich doch Sprachelemente und Programmstruktur bei den verschiedenen IEC-Systemen. Dies erleichtert die Portierung auf Fremdsysteme.

1.2.2 Geschichte und Bestandteile

Die Norm IEC 61131 stellt eine Zusammenfassung und Fortschreibung verschiedener Normen dar. Sie verweist allein auf 10 internationale Standards (IEC 50, IEC 559, IEC 617-12, IEC 617-13, IEC 848, ISO/AFNOR, ISO/IEC 646, ISO 8601, ISO 7185, ISO 7498). Sie beinhalten Vorschriften über den verwendeten Zeichencode, Definition der verwendeten Nomenklatur oder den Aufbau grafischer Darstellungen.

Bereits in der Vergangenheit wurden mehrere Versuche unternommen, eine Norm der SPS-Programmier-Technologie zu etablieren. Der IEC 61131- Standard ist die erste Norm, die die nötige internationale (und industrielle) Akzeptanz erhält. Die wichtigsten Vorläuferpapiere zur IEC 61131 sind in der Tab. 1.1 zusammengestellt.

Die Norm umfaßt (Stand August 1999) sechs Teile sowie ein Corrigendum, welches Fehlerverbesserungen der Norm beinhaltet. Zwei technische Reports und ein Amendment ergänzen die Dokumentation ([Otto-9/94]):

Jahr	deutsch	international
1977	DIN 40 719-6 (Funktionspläne)	IEC 848
1979		Start der Arbeitsgruppe für den ersten IEC 1131 Draft
1982	VDI-Richtlinie 2880, Blatt 4 SPS-Programmiersprachen	Fertigstellung erster Draft IEC 1131; Aufteilung in 5 Unterarbeitsgruppen
1983	DIN 19239 SPS-Programmierung	Christensen Report (Allen Bradley) SPS Programmiersprachen
1985		Erste Ergebnisse der IEC 65 A WG6 TF3
1990		IEC 1131 Teil 1 und 2 werden Standard
1992		International Standard IEC 1131-1,2
1993	DIN EN 61131 Teil 3	International Standard IEC 1131-3
1994	DIN EN 61131 Teile 1 und 2	
1995		International Standard IEC 1131-TR4
1996	Beiblatt zu DIN EN 61131 (Leitfaden für Anwender); entspricht IEC 1131-TR4	
1994 - 1997		Corrigendum zu IEC 1131-3
1995, 1996		Technical Reports Type 2 und 3
1997	Beiblatt zu DIN EN 61131 (Leitlinien für die Anwendung und Implementierung von Programmiersprachen); entspricht IEC 1131-TR Type 3	
1996 - 1999		Amendments

Tab. 1.1. Wichtige Vorläufer und Meilensteine der IEC 61131-3

Teil 1: *Allgemeine Informationen* :
Teil 1 enthält „allgemeine Begriffsbestimmungen und typische Funktionsmerkmale, die eine SPS von anderen Systemen unterscheiden. Hierzu zählen Standardeigenschaften wie beispielsweise die zyklische Bearbeitung des Anwenderprogramms mit dem gespeicherten Abbild der Eingangs- und Ausgangswerte oder die Arbeitsteilung zwischen Programmier-, Automatisierungs- und Bedien- und Beobachtungsgerät."

Teil 2: *Betriebsmittelanforderungen und Prüfungen:*
Dieser Teil „definiert die elektrischen, mechanischen und funktionellen Anforderungen an die Geräte sowie entsprechende Typprüfungen. Es sind die Umgebungsbedingungen (Temperatur, Luftfeuchte usw.) und Beanspruchungsklassen der Steuerungen und Programmiergeräte vorgegeben.". Derzeit erfolgt eine Überarbeitung.

Teil 3: *Programmiersprachen:*
Hier „wurden die weltweit verbreiteten SPS-Programmiersprachen in einer harmonisierten und zukunftsweisenden Überarbeitung aufgenommen."

Über formale Definitionen, durch lexikalische, syntaktische und (teilweise) semantische Beschreibungen sowie Beispiele werden das grundlegende Softwaremodell und die Programmiersprachen festgelegt. Eine umfangreiche Überarbeitung liegt derzeit als Committee Draft unter der Bezeichnung IEC 61131-3, 2nd Edition vor, der in den nächsten Jahren verabschiedet werden soll.

Teil 4: *Anwenderrichtlinien*
Der vierte Teil ist „als Leitfaden konzipiert, um den SPS-Anwender in allen Projektphasen der Automatisierung zu beraten. Die praxisgerechten Hinweise reichen von der Systemanalyse über die Gerätewahl bis zur Wartung."

Teil 4 wurde in Deutschland als „Beiblatt zu DIN EN 61131" veröffentlicht.

Teil 5: *Kommunikation:* (bei IEC in Vorbereitung)
Dieser Teil „behandelt die Kommunikation von SPSen unterschiedlicher Hersteller miteinander und mit anderen Geräten."

In Zusammenarbeit mit der ISO 9506 (Manufacturing Message Specification; MMS) erfolgt die Definition von Konformitätsklassen, die SPSen zur Kommunikation (z.B. über Netz) nutzen können. Dies umfaßt die Funktionen Geräteanwahl, Datenaustausch, Alarmverarbeitung, Zugriffskontrolle und Netzwerkverwaltung. Teil 5 ist derzeit noch nicht verabschiedet (Committee Draft CD).

Technical Report 2 „Proposed Extensions to IEC 61131-3":
Eine Vorschlagsliste beschreibt Alternativen, Erweiterungen oder Änderungen zur IEC 61131-3. Sie dient zur Vorbereitung der nächsten Überarbeitung der Norm und liegt als Committee Draft vor.

Technical Report 3 „Guidelines for the application and implementation of programming languages for programmable controllers":
Dieser Report liefert Interpretationen zu offen gebliebenen Fragen der Norm. Er enthält Richtlinien zur Implementierung, Verwendungshinweise für den Endanwender sowie programmiertechnische Hilfestellungen.

In Deutschland wurde der Report als „Beiblatt zur DIN EN 61131" veröffentlicht.

Corrigendum „Proposed Technical Corrigendum to IEC 61131-3":
Das Corrigendum verbessert Fehler der Norm, die nach deren Verabschiedung festgestellt wurden (verabschiedet).

Amendments „Proposed Amendments to IEC 61131-3":
Die Amendments stellen eine Sammlung von Verbesserungen und vor allem Ergänzungen der Norm dar, die noch zu verabschieden sind. Dieses Dokument

wurde eingeführt, um noch vor einer Nachfolgeversion des Standards wichtige Ergänzungen einbringen zu können.

Teil 8: *Fuzzy Control Language*:
Der gerade in der Abstimmungsphase befindliche Draft erweitert die Programmiersprachen um Fuzzy-Logic.

Die Norm beschreibt eine moderne Technologie und ist damit einem starken Innovationsdruck ausgesetzt, weshalb auch eine nationale und internationale Fortentwicklung der erreichten Ergebnisse erfolgt.

Dieses Buch beschäftigt sich mit dem Teil 3 „Programmiersprachen“, kurz IEC 61131-3. Dabei werden Erkenntnisse und Interpretationen der beiden Technical Reports beachtet ([IEC TR2-94] und [IEC TR3-94]), sowie die Verbesserungen des Corrigendums ([IEC CORR-94]) einbezogen.

Die IEC 1131-3 wurde als deutsche Norm „DIN EN 61131-3“ übernommen ([DIN EN 61131-3-94]). Sie ersetzt damit die Normen DIN 19239, DIN 40719T6 sowie die VDI-Richtlinie VDI 2880 Blatt 4.

Zu Inhalt und Entstehung der Norm IEC 61131 siehe auch [Rolle-98].

1.3 Organisation PLCopen

Die PLCopen, gegründet 1992, ist eine hersteller- und produktunabhängige internationale Vereinigung. Ihr gehören viele SPS-Hersteller, Softwarehäuser und unabhängige Institute in Europa und Übersee an.

1.3.1 Ziele

Das Ziel der PLCopen ist die Förderung von Entwicklung und Einsatz kompatibler Software für SPSen ([PLCopen-99]).

Die Maßnahmen zur Erreichung dieses Ziels stützen sich auf:

- die Anwendung des internationalen Standards IEC 61131-3,
- der Zusicherung der Mitglieder, SPS-Produkte anzubieten oder zu verwenden, die IEC 61131-3 konform sind,
- gemeinsame Marketingstrategien wie Messen oder Workshops,
- Hilfestellung für das internationale Normungsgremium IEC WG 65B,
- Hilfestellung für nationale Normungsgremien wie DIN-DKE UK 962.2 SPS,

- Festlegung von Konformitätsklassen zur besseren Bewertung von Programmiersystemen und die Beauftragung von unabhängigen Einrichtungen zur Durchführung entsprechender Prüfungen.

Es handelt sich bei der PLCopen somit um kein weiteres Normungsgremium, sondern um eine Interessengemeinschaft, die existierenden Standards zur internationalen Akzeptanz verhelfen möchte. Detaillierte Information ist über das Internet abrufbar (http://www.PLCopen.org).

1.3.2 Gremien und Arbeitsgebiete

Die PLCopen ist in mehrere Komitees aufgeteilt, die jeweils ein Arbeitsgebiet behandeln:

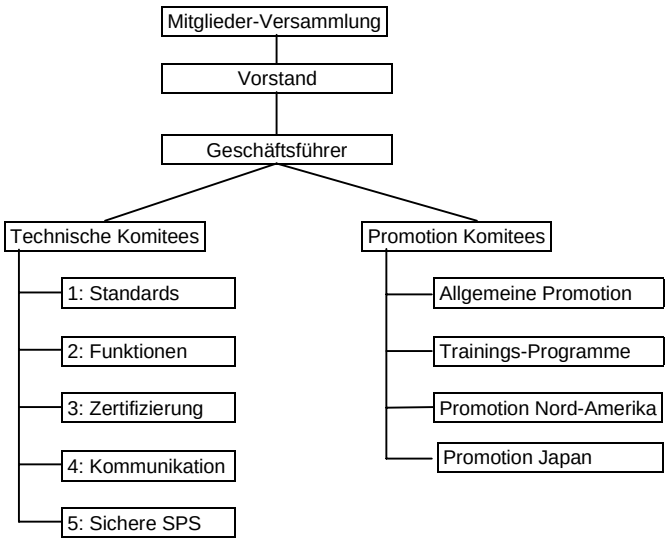


Abb. 1.1. Komitees der PLCopen

Die technischen Komitees erarbeiten Richtlinien und Vorgaben zum gemeinsamen Vorgehen; die Promotion-Gremien sind für Marketingmaßnahmen verantwortlich.

Die Arbeit in den Komitees wird ausschließlich von Vertretern einzelner Firmen und Institute vorgenommen; dies garantiert, daß die erarbeiteten Papiere in den Unternehmen umgesetzt werden können.

1.3.3 Ergebnisse

Durch vorbereitende Maßnahmen der Promotion Komittees konnten mehrere Messen in Europa und USA beschickt werden. Workshops und Fortbildungsseminare brachten der PLCopen die erwünschte internationale Anerkennung.

Als Diskussionsforum von Anwendern, Hersteller und Software-Häusern konnten respektable technische Ergebnisse erzielt werden:

- Zertifizierung für Hersteller von SPS-Programmiersystemen,
- Austauschformat für Anwenderprogramme.

Zertifizierung.

Zertifizierungs-Prüfungen sollen die Norm-Konformität von Programmiersystemen demonstrieren, die nach IEC 61131-3 implementiert wurden.

Dazu wurde eine Anforderungsliste erarbeitet, die ein SPS-Programmiersystem zu erfüllen hat, um das PLCopen-Zertifikat „Konformität nach IEC 61131-3“ zu erhalten. Diese Prüfung wird von unabhängigen Instituten mit Hilfe von Testprogrammen und Kontrollen durchgeführt.

Für jede Programmiersprache der IEC 61131-3 erfolgt eine Abstufung in drei Klassen, die immer schärfere Anforderungen stellen. Dabei dienen die in Abschn. 1.2 genannten Anforderungsdefinitionen der Norm (feature tables) als Grundlage. Über sie wird festgelegt, welche der Anforderungen im jeweiligen Level vorhanden sein müssen.

1) **Base Level**. Mit dem Programmiersystem entwickelte Programme müssen in ihrem Grundaufbau IEC 61131-3 verträglich sein. Die wesentlichen Sprachelemente einer Programmiersprache müssen vorhanden sein.
2) **Portability Level**. Die Auswahl der Anforderungen wird soweit ausgedehnt, daß es möglich ist, reale Software-Bausteine zwischen zertifizierten Programmiersystemen auszutauschen.
3) **Full Level**. Weiterer Ausbau des Portability Level, Einbeziehung von Konfigurations-Information in den Austausch-Prozeß.

Jeder SPS-Hersteller kann damit den Grad seiner Konformität zu anderen IEC 61131-3-kompatiblen Systemen für jede seiner Sprachen dokumentieren.

Die PLCopen garantiert durch die Vorgabe einer definierten Funktionalität eine Mindest-Konformität. Dadurch wird eine Vergleichbarkeit der Systeme untereinander erreicht.

Mehrere dieser Konformitätsklassen sind bereits festgelegt (AWL-Base Level, ST-Base Level, AS-Base Level, AWL-Portability Level, FBD-Base Level). Seit August 1994 gibt es die ersten AWL-zertifizierten SPS-Programmiersysteme. Die restlichen Konformitätsklassen sind in Vorbereitung.

Handelt es sich beim Base Level um einen reinen Offline Test, d.h. die Zertifizierungs-Testprogramme überprüfen das syntaktische Verhalten des Programmier-

systems, erfolgen beim Portability Level zusätzliche Online Tests, die das semantische Verhalten des Programmiersystems in Verbindung mit einer SPS prüfen. Damit wird sichergestellt, daß das Programmiersystem die IEC 61131-3 in der richtigen Art und Weise interpretiert.

Inwieweit eine weitgehende Portabilität tatsächlich realisiert werden kann, wird diskutiert und bleibt abzuwarten. Begrenzende Faktoren sind Hardware-Eigenschaften der herstellerspezifischen SPSen, die oft entscheidenden Einfluß auf die Programmarchitektur besitzen. Die hohe Funktionalität der IEC 61131-3 erschwert weiterhin eine vollständige Funktionsabdeckung durch Programmier- und SPS-Betriebssysteme. Kleinere SPS- Hersteller werden aufgrund ihres Zielmarkts oder den Möglichkeiten ihrer SPS-Familie kaum alle Eigenschaften benötigen, die Realisierungskosten für Teile der IEC 61131-3 sind groß.

Damit steht weniger die Forderung im Raum, sämtliche IEC-Bausteine zwischen allen Programmiersystemen austauschen bzw. portieren zu können. Vielmehr soll eine Grundfunktionalität vorhanden sein und sichergestellt werden, daß eine Funktionalität, die in der IEC 61131-3 zu finden ist, auch in ihrem Sinne implementiert ist.

Letztlich werden die Wünsche und Forderungen der Anwender eine Anwort auf diese Fragen geben.

Austauschformat FxF für Anwenderprogramme.
Um Anwenderprogramme zwischen zwei SPS-Programmiersystemen austauschen zu können, müssen beide dasselbe Dateiformat verstehen, einlesen und in ihr eigenes Format umwandeln können.

Dieser Austausch kann durch Einlesen einer ASCII-Datei in das Zielsystem, soweit eine Import- bzw. Export Schnittstelle vorhanden ist, oder eine Kopiere/ Einfügen-Funktion erreicht werden. Dabei werden die Typdefinitionen, Datendeklarationen und der Anweisungsteil eingelesen.

Für einen gesicherten Austausch einer Programmquelle sind noch weitere Informationen wie Entstehungsort, Version, Datum, Bearbeiter, usw. von Bedeutung. Da in der IEC 61131-3 Norm keine Festlegung oder Vorschrift für ein Dateiformat existiert, definierte die PLCopen ein ASCII-basiertes Format für textuelle Bausteine (Austauschformat FxF).

Nicht nur Programme können in andere IEC 61131-3 Systeme importiert werden. In großen Anwendungen möchte man Daten während des laufenden Betriebs austauschen. Auch hier ist ein besonderes Datenformat notwendig, es müssen Herkunftsort, Verfallsdatum der Information u.a. mitgeführt werden. Die PLCopen schlägt Datenstrukturen vor, die z.B. mit Hilfe der in der IEC 61131-5 beschriebenen Bausteine übertragen werden können.

2 Bausteine der IEC 61131-3

In diesem Kapitel werden die wesentlichen Sprachelemente der Norm IEC 61131-3 in ihrer Bedeutung und Verwendung erläutert. Dabei werden aufeinander aufbauende praxisorientierte Beispiele verwendet.

Der Leser wird in die Begriffe und Denkweise der IEC 61131-3 eingeführt. Unter Verzicht auf die formale Sprachdefinition der Norm [IEC 61131-3-94] selbst werden die grundlegenden Zusammenhänge und Konzepte verständlich und ausführlich erläutert.

Im ersten Abschnitt dieses Kapitels wird der „kompakte“ Einstieg in die Begriffswelt mit Hilfe eines Beispiels gegeben, das die wichtigsten Sprachelemente der Norm beinhaltet und gleichzeitig einen Überblick über die Methodik der SPS-Programmierung nach IEC 61131-3 bietet.

Dabei bildet der Begriff „POE“ (Programm-Organisationseinheit) einen wichtigen roten Faden zum Verständnis der neuen Sprachkonzepte.

Für die Formulierung von Beispielen wird in diesem Kapitel die Programmiersprache Anweisungsliste (AWL) gewählt, da diese Notation den meisten SPS-Programmierern bereits gut vertraut ist. Sie besitzt im europäischen SPS-Markt die größte Verbreitung und wird daher auch für IEC-Programmiersysteme zunächst verfügbar sein.

Die Programmiersprache AWL selbst wird in Abschn. 4.1 erläutert.

2.1 Einstieg in die neue Norm

Die IEC 61131-3 umfaßt nicht nur die SPS-Programmiersprachen selbst, sondern bietet umfassende Konzepte und Richtlinien zum Aufbau eines SPS-Projekts.

Ziel dieses Abschnitts ist es, einen raschen Überblick über wesentliche Begriffe zu geben, ohne dabei auf Einzelheiten einzugehen. Ein gemeinsames Beispiel

ergänzt die Erklärungen. Detaillierte Informationen sind den sich anschließenden Abschnitten und Kapiteln zu entnehmen.

2.1.1 Aufbau von Bausteinen

POEs entsprechen den *Bausteinen* bei bisherigen Programmiersystemen. POEs können sich gegenseitig mit oder ohne Parameter aufrufen. POEs bilden die kleinsten voneinander unabhängigen Software-Einheiten des Anwenderprogramms, wie es ihr Name bereits ausdrückt.

Es gibt drei Arten von POEs: *Funktion (FUN)*, *Funktionsbaustein (FB)* und *Programm (PROG)*, in dieser Reihenfolge mit zunehmender Funktionalität versehen. FUNs unterscheiden sich von FBs vor allem dadurch, daß sie bei gleichen Eingangswerten immer dasselbe Ergebnis (als Funktionswert) zurückliefern, d.h. kein „Gedächtnis“ besitzen dürfen. Dagegen arbeiten FBs jeweils auf einem eigenen Datensatz, können sich also Zustandsinformationen „merken“ (*Instanzbildung*). Programme (PROG) bilden den „Kopf“ eines SPS-Anwenderprogramms und besitzen die Möglichkeit, auf SPS-Peripherie zuzugreifen bzw. diese den übrigen POEs zur Verfügung zu stellen.

In der IEC 61131-3 sind die Aufrufschnittstellen und das Verhalten häufig benötigter *Standard-Funktionen (Std.-FUN)* wie für Arithmetik oder Vergleiche sowie *Standard-Funktionsbausteine (Std.-FB)* wie Zeiten oder Zähler fest vordefiniert.

Deklarationen von Variablen.
Die IEC 61131-3 benutzt *Variablen*, um Informationen zu speichern und zu verarbeiten. Variablen entsprechen den bisher in SPS-Systemen verwendeten Merkern. Allerdings muß ihr Speicherort nicht mehr vom Anwender manuell vergeben werden (Vergabe absoluter Adressen), sondern sie werden vom Programmiersystem automatisch verwaltet und besitzen einen fest zugeordneten *Datentyp*.

Die IEC 61131-3 gibt mehrere Datentypen vor (Bool, Byte, Integer, ...), die sich beispielsweise durch die Anzahl der Bits und Vorzeichenbehandlung unterscheiden. Es können vom Anwender auch eigene Datentypen definiert werden (Strukturen, Felder).

Variablen können auf eine physikalische Adresse abgebildet werden und eine Batteriepufferung (für Stromausfall) erhalten.

Variablen besitzen unterschiedliche Ausprägungen. Sie können außerhalb einer POE definiert (deklariert) und programmweit benutzt werden, als Aufrufparameter in die POE eingebracht werden oder lokal für die POE von Bedeutung sein. Dazu werden sie zur Deklaration in Variablenarten eingeteilt. Sämtliche in einer POE benutzten Variablen sind im Deklarationsteil der POE bekanntzugeben.

Der Deklarationsteil einer POE kann *unabhängig von der verwendeten Programmiersprache* einheitlich in textueller Form erstellt werden. Teile davon (Ein- und Ausgangsparameter der POE) sind auch grafisch darstellbar.

```
VAR_INPUT                       (* Eingangsvariable *)
  GueltigFlag   : BOOL;         (* binärer Wert *)
END_VAR
VAR_OUTPUT                      (* Ausgangsvariable *)
  Drehzahl      : REAL;         (* Gleitpunktwert *)
END_VAR
VAR RETAIN                      (* lokale Variable, batteriegepuffert *)
  MotorNr       : INT;          (* Ganzzahl mit Vorzeichen *)
  MotorName     : STRING [10];  (* Zeichenfolge für Text *)
  NotAus AT %IX2.0 : BOOL;      (* Eingangsbit 2.0 der Peripherie *)
END_VAR
```

Bsp. 2.1. Beispiel für typische Variablendeklarationen einer POE

Bsp. 2.1 zeigt, daß in der dazugehörigen POE ein Integerwert (16 Bit inkl. Vorzeichen) mit Namen MotorNr verwendet wird, und ein 10 Zeichen langer Text MotorName zur Verfügung steht. Die Variable NotAus wird an den physikalischen SPS-Signaleingang 2.0 gelegt (AT). Diese drei Variablen sind nur in dieser POE bekannt (lokal) und können dort manipuliert werden, nach einem Stromausfall bleiben ihre Werte erhalten (RETAIN). Der Wert der Eingangsvariablen GueltigFlag wird von der aufrufenden POE zur Verfügung gestellt und besitzt die booleschen Werte FALSE oder TRUE. Die POE liefert als Ausgangsparameter den Gleitpunktwert von Drehzahl zurück.

Die booleschen Werte TRUE und FALSE können auch gleichwertig mit „1“ und „0“ bezeichnet werden.

Anweisungsteil einer POE.
Der Anweisungsteil folgt dem Deklarationsteil und beschreibt die Befehle, die die SPS ausführen soll.

Eine POE wird in einer textuellen Programmiersprache Anweisungsliste (AWL) oder Strukturierter Text (ST) bzw. in den grafischen Darstellungen Kontaktplan (KOP) oder Funktionsbausteinsprache (FBS) programmiert. Während in AWL maschinennah programmiert werden kann, bietet die IEC 61131-3 mit ST eine höhere Programmiersprache. KOP ist für boolesche (binäre) Verknüpfungsschaltungen geeignet. Mit FBS können sowohl boolesche (binäre) als auch arithmetische Verknüpfungen in grafischer Darstellung programmiert werden.

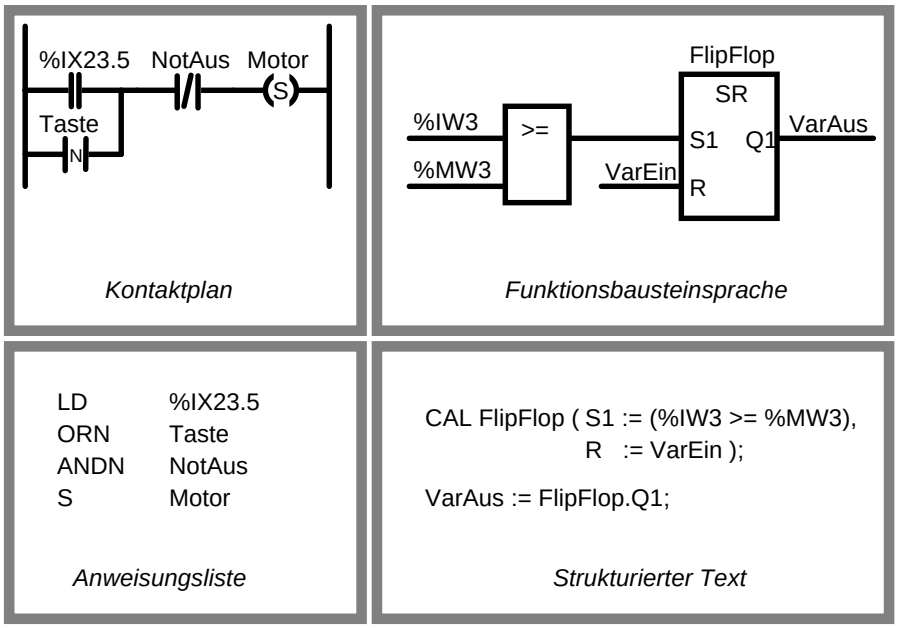


Abb. 2.1. Einfache Beispiele zu den Programmiersprachen KOP, FBS, AWL und ST. Die Darstellungen in KOP und AWL sowie in FBS und ST sind jeweils zueinander äquivalent.

Zusätzlich kann die Ablaufsprache (AS) verwendet werden, um die *Struktur* (den Aufbau) eines SPS-Programms zu beschreiben, indem die sequentiellen und parallelen Abläufe dargestellt werden. Die darin enthaltenen AS-Teilabschnitte (Schritte und Transitionen) können voneinander unabhängig in einer der IEC 61131-3-Sprachen formuliert werden.

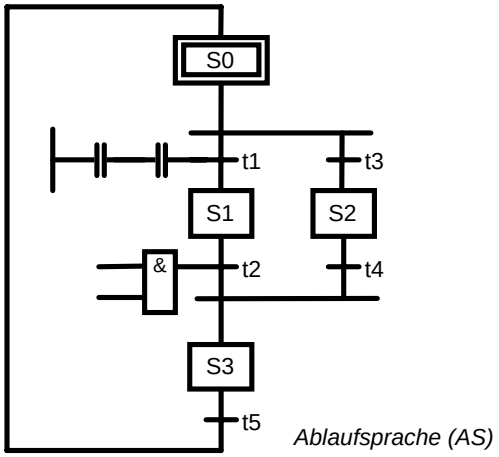


Abb. 2.2. Schematisches Beispiel zur Darstellungsart AS. Die Ausführungsteile zu den Schritten (S0 bis S3) und Transitionsbedingungen (t1 bis t5) werden in der gewünschten Programmiersprache erstellt.

Abb. 2.2 zeigt ein AS-Beispiel: Die Schritte S0, S1 und S3 werden sequentiell ausgeführt, alternativ zu S1 kann S2 zur Ausführung kommen. Die Transitionen t1 bis t5 stellen die Freischaltebedingungen dar, um von einem Schritt zum nächsten zu gelangen.

2.1.2 Einführungsbeispiel in AWL

Im folgenden wird das Beispiel eines IEC 61131-3-Programms vorgestellt, für das in Abb. 2.3 zunächst seine POE-Aufrufhierarchie charakterisiert wird.

Dieses Beispiel ist nicht als ausführbares Programm ausformuliert, sondern dient lediglich zum Aufzeigen der POE-Struktur.

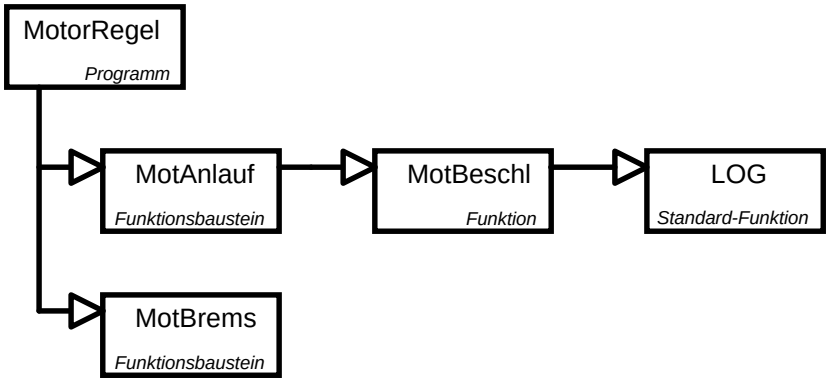


Abb. 2.3. Aufrufhierarchie von POEs für ein Beispiel

Die AWL-äquivalente Darstellung ist in Bsp. 2.2 zu sehen.

```
PROGRAM MotorRegel                          POE-Typ mit Namen
VAR_INPUT               (* Eingangsvariable *)
  MaxDrehzahl : INT;
END_VAR
VAR                     (* lokale Variable *)
  HochLauf      : MotAnlauf;                Deklarationsteil
  Abbremsen     : MotBrems;
END_VAR

  ...                   (* FB-Aufruf *)
  CAL  HochLauf (DZ := MaxDrehzahl)
  LD   HochLauf.laeuft
  ...
  CAL  Abbremsen  (* FB-Aufruf *)           Anweisungsteil
  ...

END_PROGRAM                                 POE-Ende
```

Bsp. 2.2. Deklaration des Programms MotorRegel der Abb. 2.3, sowie zugehörige Anweisungsteile in AWL. Kommentare werden in Klammern dargestellt „(* ... *)“.

```
FUNCTION_BLOCK MotAnlauf                          (* Funktionsbaustein *)
VAR_INPUT      DZ: INT;       END_VAR             (* Variablendeklaration *)
VAR_OUTPUT     laeuft: BOOL;  END_VAR             (* Variablendeklaration *)
  ...
  LD           DZ
  MotBeschl    100                                (* FUN-Aufruf *)
  ...
END_FUNCTION_BLOCK
```

```
FUNCTION_BLOCK MotBrems                           (* Funktionsbaustein *)
...
END_FUNCTION_BLOCK
```

```
FUNCTION MotBeschl : REAL                         (* Funktion *)
VAR_INPUT Param1, Param2: INT; END_VAR             (* Variablendeklaration *)
  LD    Param2
  LOG                                             (* Aufruf Std.-FUN Log. *)
```

Bsp. 2.3. Die drei Unterprogramme der Abb. 2.3 in AWL. LOG ist eine vordefinierte Standard-Funktion der IEC 61131-3.

MotorRegel ist das Hauptprogramm. Beim Start dieses Programms ist die Variable MaxDrehzahl, wie später zu sehen ist, mit einem Wert belegt, der beim Aufruf mitgegeben wird. Diese POE ruft den Baustein HochLauf, beschrieben durch MotAnlauf, auf, der seinerseits die REAL-Funktion MotBeschl mit zwei Eingangsparametern (Drehzahl DZ und 100) aufruft. Diese ruft LOG auf – die IEC 61131-*Standard-Funktion* „Logarithmus“. Wurde HochLauf (MotAnlauf) durchlaufen, wird MotorRegel wieder aktiv, wertet das FB-Ergebnis laeuft aus und ruft nun Abbremsen, beschrieben durch MotBrems, auf.

Wie in Bsp 2.2 zu sehen ist, werden die FBs MotAnlauf und MotBrems nicht unmittelbar mit diesen Namen aufgerufen, sondern über die sogenannten „*Instanz*-Namen“ HochLauf bzw. Abbremsen.

2.1.3 SPS-Zuordnung

Eine SPS kann aus mehreren Verarbeitungseinheiten wie CPUs oder Spezialprozessoren bestehen, die in der IEC 61131-3 *Ressourcen* genannt werden. Auf einer Ressource können mehrere Programme ablaufen, die sich durch Wichtigkeit (Priorität) oder Ausführungstyp (periodisch, zyklisch, Interrupt) unterscheiden. Jedem Programm wird eine *Task* zugeordnet, wodurch sich ein *Laufzeitprogramm* ergibt. Programme dürfen auch mehrfach zugeordnet werden (*Instanziierung*).

Um das in Bsp. 2.2 und 2.3 beschriebene Programm in eine SPS zu laden, sind weitere Informationen nötig, damit die zugewiesene Task die gewünschten Eigenschaften besitzt:

- Auf welchem SPS-Typ, bzw. welcher Ressource soll das Programm ablaufen?
- Wie erfolgt die Programm-Ausführung und mit welcher Priorität?
- Müssen Variable auf physikalische SPS-Adressen gelegt werden?
- Müssen Bezüge zu anderen Programmen über globale oder externe Variablen hergestellt werden?

Diese Informationen werden als *Konfiguration* (CONFIGURATION) abgelegt, wie textuell in Bsp. 2.4 und grafisch mit Abb. 2.4 gezeigt wird.

```
CONFIGURATION MotorSteu
  VAR_GLOBAL Trigger AT %IX2.3 : BOOL;  END_VAR
  RESOURCE Res_1 ON CPU001
    TASK T_1            (INTERVAL := t#80ms,   PRIORITY := 4);
    PROGRAM MotR  WITH T_1 : MotorRegel (MaxDrehzahl := 12000);
  END_RESOURCE
  RESOURCE Res_2 ON CPU002
    TASK T_2            (SINGLE := Trigger, PRIORITY := 1);
    PROGRAM MotP  WITH T_2 : MotorProg (...);
  END_RESOURCE
END_CONFIGURATION
```

Bsp. 2.4. Zuordnung der Programme aus Bsp. 2.3 zu Tasks und Ressourcen innerhalb einer Konfiguration

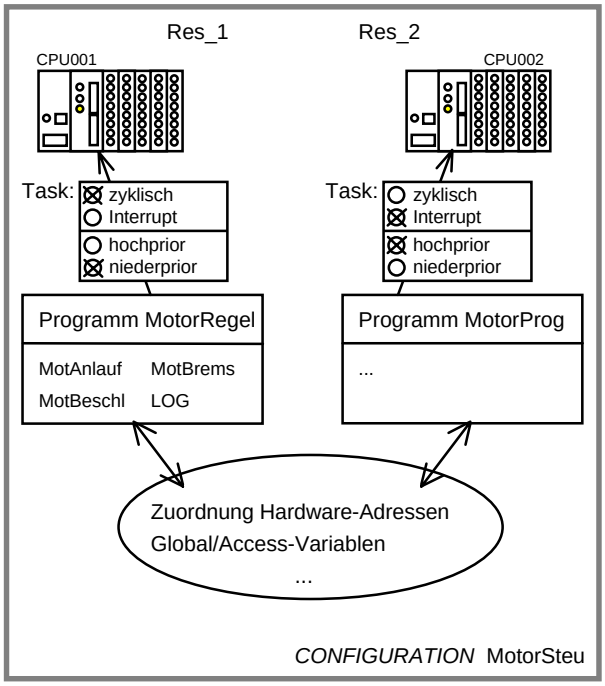


Abb. 2.4. Verknüpfung der Programme einer Motorsteuerung MotorSteu mit Tasks in der SPS-„Konfiguration“. Die so erhaltenen Laufzeitprogramme werden durch die Ressourcen (Prozessoren) des SPS-Systems ausgeführt.

Abb. 2.4 führt Bsp. 2.3 fort. Das Programm MotorRegel läuft zusammen mit seinen FUNs und FBs auf der Ressource CPU001. Dabei wird mit der Task angegeben, daß MotorRegel mit niedriger Priorität zyklisch ablaufen soll. Das Programm MotorProg läuft hier auf der CPU002 ab, könnte aber durchaus auch auf CPU001 ausgeführt werden, wenn diese CPU Multitasking unterstützt.

Weiterhin werden in der Konfiguration Variablen der SPS-Peripherie zugewiesen sowie globale und Kommunikations-Variablen verwaltet. Dies ist ebenfalls innerhalb eines PROGRAM möglich.

SPS-Projekte bestehen aus POEs, die vom SPS-Hersteller mitgeliefert oder vom Anwender selbst programmiert werden. Aus Anwenderprogrammen können Bibliotheken gebildet werden, deren ausgetestete POEs in neue Projekte einfließen. Die IEC 61131-3 unterstützt diesen Aspekt der Wiederverwendbarkeit, indem Funktionen und Funktionsbausteine – soweit wie möglich – allgemeingültig, d.h. hardwareunabhängig bleiben.

Nach diesem kurzen Überblick werden in den weiteren Abschnitten die Eigenschaften und Besonderheiten von POEs ausführlich erläutert.

2.2 Die Programmorganisationseinheit (POE)

Als Programm-Organisationseinheiten (POE) bezeichnet die IEC 61131-3 die Bausteine, aus denen ein Projekt aufgebaut wird. POEs entsprechen den üblichen Programm-, Organisations-, Schritt- und Funktionsbausteinen der bisherigen SPS-Programmierwelt.

Ein wesentliches Ziel bei der Normierung war es, die Vielfalt und oft implizite Bedeutung von Bausteinarten zu beschränken und in ihrem Wesen zu vereinfachen.

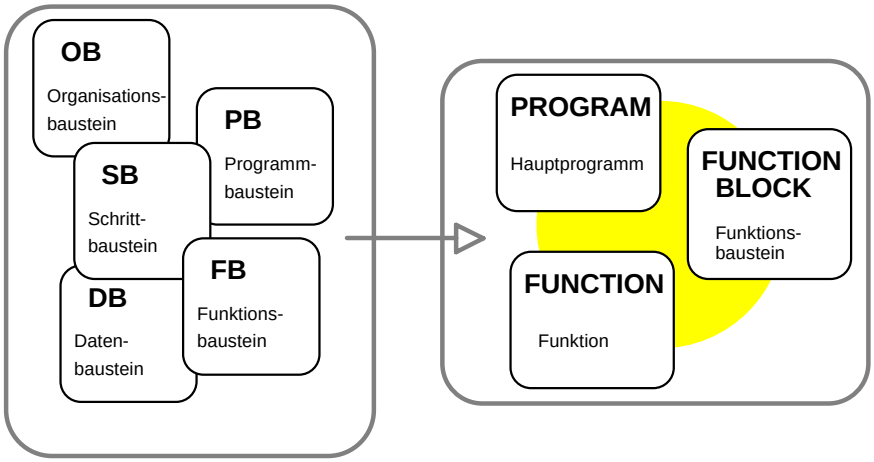


Abb. 2.5. Übergang von bisherigen Bausteintypen zu POEs der IEC 61131-3

Wie Abb. 2.5 zeigt, reduziert die IEC 61131-3 die diversen Bausteintypen einzelner SPS-Hersteller auf drei einheitliche Grundtypen. Datenbausteine werden durch FB-Datenbereiche („Instanzen“, siehe unten) oder globale Multielement-Variable (vgl. Kap. 3) abgelöst.

Folgende drei *POE-Typen* bzw. „Bausteinarten“ gibt es in der neuen Norm:

POE-Typ	Schlüsselwort	Bedeutung
Programm	PROGRAM	Hauptprogramm mit Zuordnung der SPS-Peripherie, globalen Variablen und Zugriffspfaden
Funktions-baustein	FUNCTION_BLOCK	Baustein mit Ein- und Ausgangs-Variablen; ist der zur Programmierung hauptsächlich benutzte POE-Typ
Funktion	FUNCTION	Baustein mit Funktionswert zur Erweiterung des SPS-Operationsvorrats

Tab. 2.1. Die drei POE-Typen der IEC 61131-3 mit ihrer Bedeutung

Diese drei POE-Typen unterscheiden sich durch besondere Eigenschaften in ihrer Verwendung:

- **Funktion (FUN)**. Parametrierbare POE ohne statische Variablen (ohne Gedächtnis), die bei denselben Eingangsparametern stets dasselbe Ergebnis als Funktionswert liefert.
- **Funktionsbaustein (FB)**. Parametrierbare POE mit statischen Variablen (mit Gedächtnis). Ein FB (z.B. Zeit- oder Zählerbaustein) liefert bei gleichen Eingangswerten Ergebnisse, die auch vom Zustand seiner internen (VAR) und externen (VAR_EXTERNAL) Variablen abhängen können, die zwischen FB-Aufrufen erhalten bleiben.
- **Programm (PROG)**. Dieser POE-Typ stellt das „Hauptprogramm“ dar. Alle Variablen des Gesamtprogramms, denen physikalische Adressen (z.B. Ein- und Ausgänge der SPS) zugewiesen sind, müssen in dieser POE oder oberhalb (Ressource bzw. Konfiguration) deklariert werden. Sonst wie FB.

PROG und FB können Ein- und Ausgangsparameter besitzen, Funktionen haben Eingangsparameter sowie ihren Funktionswert als Rückgabewert. Solche Eigenschaften waren bisher nur Funktionsbausteinen vorbehalten.

Daher entspricht FUNCTION_BLOCK der IEC 61131-3 mit Eingangs- *und* Ausgangsparametern in etwa einem bisher üblichen Funktionsbaustein. Die POE-Typen PROGRAM und FUNCTION besitzen aufgrund ihrer gegenüber FBs erweiterten bzw. eingeschränkten Eigenschaften keine direkten Pendants zu Bausteinen nach DIN 19239.

Eine POE ist eine für sich abgeschlossene Einheit, die vom Compiler unabhängig von anderen Programmteilen übersetzt werden kann. Der Compiler benötigt allerdings Informationen über die Aufrufschnittstellen derjenigen POEs, die in diesem Baustein aufgerufen werden (Prototypen). Übersetzte POEs können später gemeinsam zusammengebunden werden („Link“-Vorgang), um ein Gesamtprogramm zu bilden.

Der Name einer POE ist projektweit bekannt und darf daher nicht mehrfach vergeben werden. Lokale Unterprogramme wie in manchen anderen (Hoch-)

Sprachen kennt die IEC 61131-3 nicht. Damit steht nach der Programmierung einer POE, der sogenannten *Deklaration*, ihr Name und ihre Aufrufschnittstelle sämtlichen anderen POEs des Projekts global zur Verfügung.

Diese Selbständigkeit der POEs ermöglicht eine weitreichende Modularisierung der Automatisierungsaufgabe sowie die *Wiederverwendbarkeit* von bereits fertig implementierten und getesteten Software-Teilen.

In den nächsten Abschnitten werden zunächst Gemeinsamkeiten der POE-Typen erläutert. Anschließend werden die POE-Typen charakterisiert, Aufrufbeziehungen beschrieben und schließlich zusammenfassend miteinander verglichen.

2.3 Elemente einer POE

Eine POE besteht aus den in Abb. 2.6 gezeigten Teilen:

- Angabe des POE-Typs mit POE-Namen (und Datentyp bei Funktionen),
- Deklarationsteil mit Variablendeklarationen,
- POE-Rumpf mit Anweisungen.

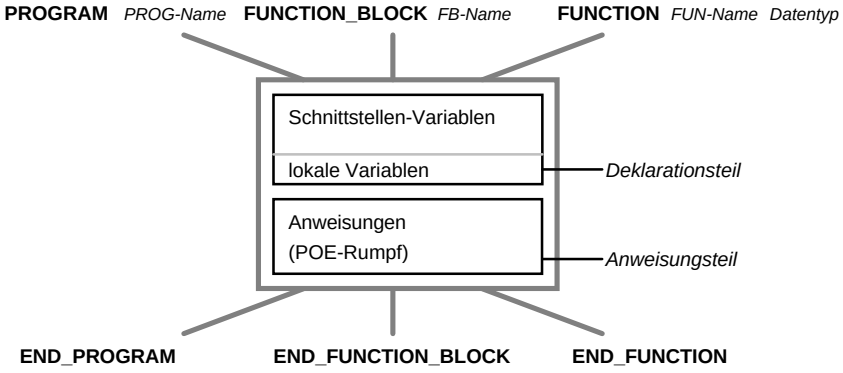


Abb. 2.6. Der gemeinsame Aufbau der drei POE-Typen Programm (links), Funktionsbaustein (Mitte) und Funktion (rechts). Der Deklarationsteil enthält Schnittstellen- und lokale Variablen.

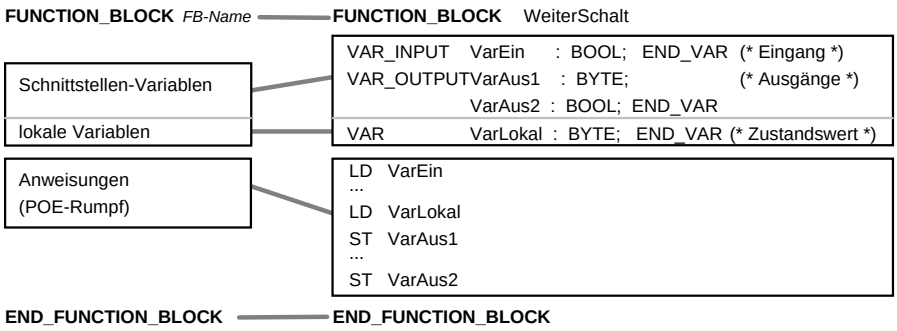
Die *Deklarationen* dienen zur Festlegung sämtlicher von der POE verwendeten Variablen. Dabei wird zwischen nach außen sichtbaren Variablen (POE-Schnittstelle) und den lokalen Variablen der POE unterschieden. Diese Möglichkeiten werden im nächsten Abschnitt sowie detaillierter in Kap. 3 erklärt.

Im *Anweisungsteil* (Rumpf) einer POE wird die Verknüpfungs- oder Berechnungsvorschrift in der gewünschten Programmiersprache programmiert. Sprachen der IEC 61131-3 werden im Kap. 4 vorgestellt und erläutert.

Deklarationen und Anweisungen können graphisch oder textuell programmiert werden.

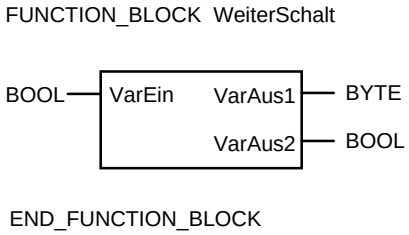
2.3.1 Beispiel

In Bsp. 2.5 werden die Elemente einer POE veranschaulicht.



Bsp. 2.5. Elemente einer POE (links) und Beispiel für einen FB in AWL (rechts). Der FB enthält einen Eingangsparameter VarEin sowie zwei Rückgabewerte VarAus1 und VarAus2. VarLokal ist eine lokale Variable.

Der in der Programmiersprache AWL geschriebene Funktionsbaustein WeiterSchalt enthält den Eingangsparameter VarEin sowie die beiden Rückgabewerte VarAus1 und VarAus2 und die lokale Variable VarLokal. Im FB-Rumpf wird die Verwendung der AWL-Operatoren LD (Laden) und ST (Speichern) angedeutet.



Bsp. 2.6. Grafische Darstellung der Aufrufschnittstelle des FB WeiterSchalt in Bsp. 2.5.

In der grafischen Darstellung der Aufrufschnittstelle sind lokale FB-Variablen wie VarLokal nicht sichtbar.

2.3.2 Deklarationsteil

Die IEC 61131-3 benutzt Variablen, um Anwender-Daten zu initialisieren, weiterzuverarbeiten und zwischenzuspeichern. Diese Variablen werden zu Beginn jeder POE *deklariert*, d.h. ihre Zuordnung zu einem bestimmten *Datentyp* (wie BYTE oder REAL) bekanntgegeben.

Bei dieser Deklaration können Variablen-*Eigenschaften* wie Batteriepufferung, Anfangswerte oder Zuordnungen zu physikalischen Adressen festgelegt werden.

Die Deklaration der POE-Variablen erfolgt blockweise für *Variablenarten*, die jeweils deren Verwendungszweck bestimmen, wie Bsp. 2.7 zeigt. Jeder solcher *Deklarationsblock* (VAR_*...END_VAR) betrifft eine Variablenart und kann eine oder mehrere Variablen enthalten. Reihenfolge der Blöcke und Häufigkeit von Blöcken derselben Variablenart sind wie in Bsp. 2.8 beliebig bzw. implementierungsabhängig (für das jeweilige Programmiersystem).

```
(* Lokale Variable *)
VAR                VarLokal : BOOL;   END_VAR     (* lokale boolesche Var. *)
(* Aufrufschnittstelle: Eingangsparameter *)
VAR_INPUT          VarEin : REAL;     END_VAR     (* Eingangsvariable *)
VAR_IN_OUT         VarEinAus : UINT;  END_VAR     (* Ein- und Ausgangsvariable *)
(* Rückgabewerte: Ausgangsparameter *)
VAR_OUTPUT         VarAus : INT;      END_VAR     (* Ausgangsvariable *)
(* Globale Schnittstelle: Globale/Externe Variablen und Zugriffspfade *)
VAR_EXTERNAL       VarGlob : WORD;    END_VAR     (* extern von anderer POE *)
VAR_GLOBAL         VarGlob : WORD;    END_VAR     (* global für andere POEs *)
VAR_ACCESS         VarPfad : WORD;    END_VAR     (* Zugriffspfade der Konfiguration *)
```

Bsp. 2.7. Beispiele für Deklarationen der verschiedenen Variablenarten.

```
(* Deklarationsblock 1 *)
VAR          VarLokal1, VarLokal2, VarLokal3: BOOL; END_VAR
(* Deklarationsblock 2 *)
VAR_INPUT    VarEin1 : REAL;                       END_VAR
(* Deklarationsblock 3 *)
VAR_OUTPUT   VarAus : INT;                         END_VAR
(* Deklarationsblock 4 *)
VAR          VarLokal4, VarLokal5 : BOOL;          END_VAR
(* Deklarationsblock 5 *)
VAR_INPUT    VarEin2, VarEin3 : REAL;              END_VAR
(* Deklarationsblock 6 *)
VAR_INPUT    VarEin4 : REAL;                       END_VAR
```

Bsp. 2.8. Beispiele für Deklarationsblöcke: Reihenfolge und Häufigkeit der Blöcke werden durch die IEC 61131-3 nicht festgelegt.

Variablenarten in POE-Typen.
Je nach POE-Typ sind unterschiedliche Variablenarten zulässig, wie Tab. 2.2 zeigt:

Variablenart	**erlaubt in:**		
	PROGRAM	*FUNCTION_BLOCK*	*FUNCTION*
VAR	ja	ja	ja
VAR_INPUT	ja	ja	ja
VAR_OUTPUT	ja	ja	nein
VAR_IN_OUT	ja	ja	nein
VAR_EXTERNAL	ja	ja	nein
VAR_GLOBAL	ja	nein	nein
VAR_ACCESS	ja	nein	nein

Tab. 2.2. Verwendung von Variablenarten innerhalb der drei POE-Typen

Wie Tab. 2.2 zeigt, können in Programmen sämtliche Variablenarten verwendet werden. Funktionsbausteine können keine globalen Variablen anderen POEs zur Verfügung stellen, dies ist nur in Programmen, Ressourcen und Konfigurationen erlaubt. FBs greifen auf solche globale Daten über die Variablenart VAR_EXTERNAL zu.

Funktionen sind am stärksten eingeschränkt, da für sie nur lokale und Eingangsvariablen zulässig sind. Sie geben ihr Berechnungsergebnis im Funktionswert zurück.

Bis auf die lokalen Variablen können mit jeder Variablenart Daten in die POE eingebracht bzw. nach außen geliefert werden, dienen also dem Datenaustausch zwischen POEs. Die Merkmale dieser *POE-Schnittstelle* werden im folgenden betrachtet.

Merkmale der POE-Schnittstelle.
Durch die Zuordnung von POE-Variablen zu Variablenarten (in Deklarationsblöcken) werden die Schnittstellen der POE sowie ihr lokaler Datenbereich festgelegt. Die POE-Schnittstelle gliedert sich in:

- Aufrufschnittstelle: Formalparameter (Ein- und Ein-/Ausgangsparameter),
- Rückgabewerte: Ausgangsparameter oder Funktionswert,
- Globale Schnittstelle mit globalen/externen Variablen und Zugriffspfaden.

Aufrufschnittstelle und die Rückgabewerte (bzw. Funktionswert) einer POE sind in den Sprachen KOP und FBS auch grafisch darstellbar.

Die Variablen der Aufrufschnittstelle werden auch als *Formalparameter* bezeichnet. Beim Aufruf der POE werden die Formalparameter mit *Aktualparametern* versorgt, d.h. mit aktuellen (Variablen-) Werten oder Konstanten belegt.

In Bsp. 2.3 besitzt der FB MotAnlauf den einzigen Formalparameter DZ, der in Bsp. 2.2 mit dem Wert des Aktualparameters MaxDrehzahl versorgt wird, sowie den Ausgangsparameter laeuft. Die Funktion MotBeschl mit zwei Formalparametern (einer davon mit dem Wert 100 versorgt) liefert ihr Berechnungsergebnis im Funktionswert zurück.

Diese Vorstellung faßt Tab. 2.3 zusammen.

	Variablenart	**Bemerkungen**
Aufrufschnittstelle (Formalparameter)	VAR_INPUT, VAR_IN_OUT	Eingangsparameter, grafisch darstellbar
Rückgabewerte	VAR_OUTPUT	Ausgangsparameter, grafisch darstellbar
Globale Schnittstelle	VAR_GLOBAL, VAR_EXTERNAL, VAR_ACCESS	globale Daten
Lokale Daten	VAR	POE-interne Daten

Tab. 2.3. Variablenarten für Schnittstelle und lokale Daten einer POE, siehe auch Kommentierung in Bsp 2.7.

Formalparameter und Rückgabewerte einer POE.
Die beiden Aufrufschnittstellen bzw. der Rückgabewert unterscheiden sich in der Zugriffsmethode bzw. im Zugriffsrecht (siehe auch [IEC TR3-94]).

Formalparameter (VAR_INPUT): Die Aktualparameter werden der POE als *Werte* übergeben, d.h. nicht die Variable selbst wird an die POE weitergereicht, sondern lediglich ihre Kopie. Dadurch ist sichergestellt, daß diese Eingangsvariable innerhalb der aufgerufenen POE nicht verändert werden kann. Dieses Konzept wird auch als *call-by-value* bezeichnet.

Formalparameter (VAR_IN_OUT): Die Aktualparameter werden der aufgerufenen POE als *Zeiger* auf ihren Speicherort übergeben, d.h. die Variable selbst wird an die POE weitergereicht, so daß sie dort gelesen und verändert werden kann. Änderungen wirken sich also automatisch auf die außerhalb der aufgerufenen POE deklarierten Variable aus. Dieses Konzept wird auch als *call-by-reference* bezeichnet.

Diese Variablenart bietet in diesem Sinne „Zeiger“ (engl.: pointer), die auch in höheren Programmiersprachen wie C typischerweise für Rückgabewerte von Unterprogrammen verwendet werden, indem die Speicheradressen der entsprechenden Parameter übergeben werden.

Rückgabewerte (VAR_OUTPUT) werden der aufgerufenen POE nicht übergeben, sondern stehen dort als Werte zur Verfügung. Sie sind daher nicht Bestandteil der Aufrufschnittstelle. Sie werden zwar grafisch zusammen mit VAR_INPUT und VAR_IN_OUT dargestellt, ihre *Werte* in textuellen Sprachen wie AWL oder ST jedoch *nach* Aufruf der POE lesend weiterverarbeitet.

Ihre Übergabe an die aufrufende POE erfolgt ebenfalls *return-by-value*, indem ihre Werte der aufrufenden Instanz (FB oder PROG) zur Verfügung stehen. Dadurch sind die Ausgangsparameter der POE gegenüber Änderungen durch die aufrufende POE geschützt. Beim POE-Typ PROGRAM werden die Ausgangsparameter beim Aufruf durch die Ressource zusammen mit den Aktualparametern angegeben und entsprechenden Variablen zur Weiterverarbeitung zugewiesen (vgl. Beispiele des Kap. 6).

Falls einer POE beim Aufruf umfangreiche Felder oder Datenstrukturen als Variablen übergeben werden, führt die Verwendung von VAR_IN_OUT zu effizienten Programmen, da die Variablen zur Laufzeit nicht umkopiert werden müssen (VAR_INPUT bzw. VAR_OUTPUT), sondern einfach ihr Zeiger benutzt wird. Allerdings sind solche Variablen nicht vor (unerwünschter) Manipulation durch die aufgerufene POE geschützt.

Externer und interner Zugriff auf POE-Variablen

Formalparameter und Rückgabewerte besitzen die besondere Eigenschaft, auch außerhalb (extern) der sie verwendenden POE *sichtbar* zu sein: die aufrufende POE kann (muß aber nicht) ihre Namen explizit verwenden, um Eingangsvariable zu setzen.

Dadurch wird die Aufrufstelle einer POE besser dokumentierbar und es können Parameter vertauscht bzw. weggelassen werden. Ein- und Ausgangsvariablen besitzen in diesem Zusammenhang zusätzlich einen Schutz gegen unberechtigtes Lesen bzw. Schreiben.

Tab. 2.4 faßt sämtliche Variablenarten mit deren Bedeutung im Überblick zusammen. Dabei werden zu jeder Variablenart Schreib- bzw. Lese-Rechte angegeben, die kennzeichnen, ob eine Variable:

- in der aufrufenden POE („extern“) sichtbar und dort schreib- oder lesbar ist,
- innerhalb der POE („intern“) schreib- oder lesbar ist.

Variablenart	Zugriffsrechte[a]		Erläuterung
	extern	intern	
VAR *lokale Variable*	-	SL	Eine lokale Variable ist nur innerhalb der POE sichtbar und kann nur dort bearbeitet werden.
VAR_INPUT *Eingangsvariable*	S	L	Eine Eingangsvariable ist in der aufrufenden POE sichtbar und dort beschreibbar. Sie darf innerhalb der POE nur gelesen werden.
VAR_OUTPUT *Ausgangsvariable*	L	SL	Eine Ausgangsvariable ist in der aufrufenden POE sichtbar und kann dort nur gelesen werden. Sie ist durch die POE les- und beschreibbar.
VAR_IN_OUT *Ein- und Ausgangsvariable*	SL	SL	Eine Ein-/Ausgangsvariable beinhaltet die kombinierten Eigenschaften von VAR_INPUT und VAR_OUTPUT: sie ist innerhalb und außerhalb der POE les- und beschreibbar.
VAR_EXTERNAL *Externe Variable*	SL	SL	Eine Externe Variable wurde durch eine andere POE als *global* deklariert und ist für alle POEs les- und beschreibbar. Sie kann innerhalb der POE wie eine lokale Variable geändert werden. Ihr (geänderter) Wert wird auch nach außen wirksam.
VAR_GLOBAL *Globale Variable*	SL	SL	Eine globale Variable wird innerhalb der POE deklariert und ist dort sowie nach außen (als extern) für sämtliche POEs les- und beschreibbar. Sie kann innerhalb der POE wie eine lokale Variable geändert werden. Ihr (geänderter) Wert wird auch nach außen wirksam.
VAR_ACCESS *Zugriffspfade*	SL	SL	Globale Variable von Konfigurationen als Kommunikationskanal zwischen Komponenten (Ressourcen) von Konfigurationen (vgl. Kap. 6). Sie kann innerhalb der POE wie eine globale Variable benutzt werden.

a S=schreibend, L=lesend, SL=schreibend und lesend

Tab. 2.4. Bedeutung der Variablenarten. In der linken Spalte ist das Schlüsselwort der Variablenart jeweils fett dargestellt. In der Spalte „Zugriffsrechte“ wird angegeben, welche Schreib-/Leserechte für die aufrufende POE (extern) bzw. innerhalb der POE (intern) bestehen.

Für Eingangs- und Ausgangsvariablen bietet die IEC 61131-3 einen weitgehenden Zugriffsschutz, wie aus der Tab. 2.4 für VAR_INPUT und VAR_OUTPUT ersichtlich ist: Eingangsvariable dürfen innerhalb der POE nicht verändert werden, Ausgangsparameter nicht außerhalb.

Die Ausführungen dieses Abschnitts „Deklarationsteil einer POE“ haben vor allem für die Funktionsbausteine Bedeutung. Darauf wird in Abschn. 2.4.1 im Zusammenhang mit der FB-Instanziierung nochmals eingegangen.

Auf Formalparameter und Rückgabewerte von POEs kann sowohl beim Aufruf (externer Zugriff) als auch innerhalb der POE (interner Zugriff) zugegriffen werden, wie folgende Beispiele für Funktionsbausteine zeigen:

```
FUNCTION_BLOCK  FBZwei
VAR_INPUT
  EingVar  :  BYTE;
END_VAR
VAR_OUTPUT
  AusgVar  :  BYTE;
END_VAR
VAR LokVar : BYTE; END_VAR

...
LD    EingVar
AND   LokVar
ST    AusgVar
...
END_FUNCTION_BLOCK
```

```
FUNCTION_BLOCK  FBEins
VAR BspFB : FBZwei; END_VAR

...
LD    44
ST    BspFB.EingVar
CAL   BspFB          (* FB-Aufruf *)
LD    BspFB.AusgVar
...
END_FUNCTION_BLOCK
```

Bsp. 2.9. Interner (links) und externer Zugriff (rechts) auf die FB-Formalparameter EingVar und AusgVar.

Im Bsp. 2.9 ruft FBEins den Baustein BspFB (beschrieben durch FBZwei) auf. Dazu übergibt er ihm die Konstante 44 als Aktualparameter für die Eingangsvariable EingVar. D.h. in FBEins ist diese Eingangsvariable sichtbar und wird vorbesetzt. Ebenso ist AusgVar sichtbar und kann in FBEins lesend weiterverarbeitet werden. Innerhalb FBZwei können EingVar gelesen (LD) und AusgVar geschrieben (ST) werden.

Weitere Eigenschaften und Besonderheiten von Variablen und Variablenarten werden in Abschn. 3.4 beschrieben.

2.3.3 Anweisungsteil

Der Anweisungsteil (Rumpf) einer POE folgt dem Deklarationsteil und enthält die von der SPS auszuführenden Anweisungen. Zur anwendungsgerechten Formulierung der Steuerungsaufgabe stehen in der IEC 61131-3 fünf Programmiersprachen zur Verfügung, von denen drei eine grafische Darstellung besitzen.

Da die Art der Programmierung zwischen den einzelnen Sprachen stark voneinander abweicht, eignen sich diese für unterschiedliche Aufgaben und Anwendungsbereiche. Als Anhaltspunkt kann man beispielsweise angeben:

AS	*Ablaufsprache*: Zerlegung der Steuerungsaufgabe in sequentielle und parallel auszuführende Bestandteile (Teilabschnitte), sowie deren Ausführungskontrolle. AS beschreibt anschaulich den Programmablauf, indem angegeben wird,

	welche Aktion des zu steuernden Prozesses zu welchem Zeitpunkt freigegeben bzw. beendet oder gesperrt wird. Die IEC 61131-3 betont die Bedeutung von AS als *Darstellungsart* zur Strukturierung des SPS-Programms.
KOP	*Kontaktplan*: Grafische Verknüpfung („Stromlaufplan“) von booleschen Variablen (Kontakte und Spulen), geometrische Sicht einer Schaltung ähnlich früheren Relais-Steuerungen. In KOP geschriebene POEs werden in *Netzwerke* genannte Abschnitte unterteilt.
FBS	*Funktionsbausteinsprache*: Grafische Verknüpfung unterschiedlicher Funktionselemente (arithmetische, boolesche, sonstige Funktionen und FBs). In FBS geschriebene POEs werden wie bei KOP in *Netzwerke* unterteilt. Häufig können *boolesche* Netzwerke auch in KOP dargestellt werden und umgekehrt.
AWL	*Anweisungsliste*: Maschinennahe Programmiersprache, die von den meisten Programmiersystemen angeboten wird.
ST	*Strukturierter Text*: Höhere Programmiersprache (ähnlich PASCAL) für Steuerungsaufgaben sowie komplexe Berechnungsverfahren.

Tab. 2.5. Merkmale der Programmiersprachen der IEC 61131-3

Darüber hinaus erlaubt die Norm ausdrücklich, daß weitere Programmiersprachen (wie beispielsweise C oder BASIC) zur Programmierung zulässig sind, sofern sie folgende Bedingungen erfüllen:

- normgerechte Verwendung von Variablen wie in den übrigen Programmiersprachen der IEC 61131-3, also konform zum Deklarationsteil einer POE,
- normgerechte Aufrufe von Funktionen und Funktionsbausteinen, insbesondere der Standard-FUN bzw. -FB,
- keine Widersprüche zu übrigen Programmiersprachen bzw. der Darstellungsart AS.

Auf diese Programmiersprachen und ihre Darstellungsweise wird im Kap. 4 detailliert eingegangen.

2.4 Der Funktionsbaustein

Funktionsbausteine bilden das wesentliche Hilfsmittel zur Strukturierung von SPS-Programmen. Sie werden von Programmen und FBs aufgerufen und können selbst Funktionen und Funktionsbausteine aufrufen.

In diesem Abschnitt wird auf die prinzipiellen Eigenschaften von Funktionsbausteinen eingegangen.

Ein ausführliches FB-Beispiel ist in Anh. C zu finden.

Das Konzept der „Instanziierung von FBs“ besitzt große Bedeutung in der Norm und bildet ein wesentliches Unterscheidungskriterium zwischen den drei POE-Typen. Daher wird zunächst dieses Konzept vorgestellt, bevor auf die übrigen Eigenschaften der POE-Typen eingegangen wird.

2.4.1 Instanziierung von Funktionsbausteinen

Was ist eine Instanz?

Das Anlegen von Variablen durch den Programmierer unter Angabe des Variablen-Namens und Datentyps bei der Deklaration wird *Instanziierung* oder Instanzbildung genannt.

Im folgenden Beispiel 2.10 ist die Variable namens Ventil eine *Instanz* des Datentyps BOOL:

```
Ventil   :        BOOL;              (* boolesche Variable *)
  └─ Variablen-Name     └─ Datentyp

Motor1   :        MotorTyp;          (* FB-Instanz *)
  └─ FB-Instanz-Name    └─ FB-Typ (benutzerdefiniert)
```

Bsp. 2.10. Deklaration einer Variable als „Instanz eines Datentyps“ (oben). Deklaration einer FB-„Variable“ als „Instanz eines benutzerdefinierten FB-Typs“ (unten).

Wie Variablen werden auch Funktionsbausteine instanziiert: In Bsp. 2.10 wird zum benutzerdefinierten Funktionsbaustein (FB-Typ) MotorTyp innerhalb einer POE-Deklaration die FB-Instanz Motor1 deklariert. Nach der Instanziierung kann ein FB (als Instanz) innerhalb der POE verwendet und aufgerufen werden.

Das Prinzip der Instanziierung mag auf den ersten Blick ungewöhnlich erscheinen, ist jedoch nichts wirklich Neues.

Bisher wurden beispielsweise Funktionsbausteine mit Zeit- oder Zählaufgaben, kurz Zeiten und Zähler genannt, zumeist durch ihren Typ (wie Zählrichtung, Zeitverlauf) sowie eine vom Benutzer vergebene Nummer eindeutig festgelegt (z.B. FB 19).

In der neuen Norm wird nun anstelle dieser absoluten Nummer ein Variablenname unter Angabe des entsprechenden Zeiten- oder Zählertyps verlangt. Dazu wird eine Deklaration im Deklarationsteil der POE erforderlich. Das Programmiersystem kann beim Übersetzen der POE für diese FB-Variablen intern automatisch absolute Nummern für die SPS erzeugen.

Mit Hilfe dieser Namen greift der SPS-Programmierer übersichtlich auf verschiedene Zeiten und Zähler desselben Typs zu, ohne daß sie sich gegenseitig beeinflussen würden.

Die IEC 61131-3 vereinheitlicht durch die Instanziierung die Verwendung von herstellerspezifischen (wie typischerweise Zeiten und Zähler) und benutzerdefinierten FBs. Der Instanzname entspricht von der Handhabung her dem entsprechenden symbolischen Bezeichner, der FB-Typ der FB-Aufrufschnittstelle. FB-Instanzen bieten darüber hinaus noch wesentlich mehr: „Struktur“ und „Gedächtnis“ für FBs werden in den beiden nächsten Abschnitten beschrieben.

Der Begriff „Funktionsbaustein“ wird häufig für zwei verschiedene Bedeutungen verwendet: er dient sowohl als Synonym für den FB-Instanznamen wie auch als FB-Typ (= Name des FB selbst). In diesem Buch wird mit „Funktionsbaustein“ der *FB-Typ* bezeichnet, während die FB-Instanz jeweils ausdrücklich als Instanzname gekennzeichnet wird.

Bsp. 2.11 zeigt einige Deklarationen von Funktionsbausteinen (hier Std.-FBs) und Variablen im Vergleich:

```
VAR
  FUELLST      : UINT;     (* vorzeichenlose Ganzzahl-Variable *)
  NOT_AUS      : BOOL;     (* boolesche Variable *)
  Zeit9        : TON;      (* Zeitglied vom Typ Einschaltverzögerung *)
  Zeit13       : TON;      (* Zeitglied vom Typ Einschaltverzögerung *)
  ZaehlRueck   : CTD;      (* Rückwärtszähler *)
  AllgZaehler  : CTUD;     (* Auf- und Abwärtszähler *)
END_VAR
```

Bsp. 2.11. Beispiele zur Variablendeklaration und Instanziierung von Standard-Funktionsbausteinen (fett dargestellt)

Obwohl Zeit9 und Zeit13 in diesem Beispiel auf demselben FB-Typ (TON) eines Zeiten-FB beruhen, sind es eigenständige Zeitbausteine, die als Instanzen getrennt

voneinander aufgerufen und ausgewertet werden, also zwei „Zeiten“ repräsentieren.

FB-Instanzen sind innerhalb der POE sichtbar und benutzbar, in der sie deklariert wurden. Werden sie als global deklariert, stehen sie mit VAR_EXTERNAL sämtlichen POEs zur Verfügung.

Funktionen sind dagegen immer projektweit sichtbar und können von jeder POE (ohne Deklaration) aufgerufen werden. Ebenso sind FB-**Typen** projektweit bekannt und können in jeder POE zur Deklaration von Instanzen benutzt werden.

Deklaration und Aufruf der Standard-FBs werden in Kap. 5 ausführlich beschrieben. Ihre Verwendung wird in der jeweiligen Programmiersprache erläutert, siehe Kap. 4.

Instanz bedeutet „Struktur“.
Das Konzept der Instanziierung führt am Beispiel von Funktionsbausteinen wie Zeiten und Zählern zu *strukturierten* Variablen, die:

- die Aufrufschnittstelle wie eine Datenstruktur beschreiben,
- den aktuellen Zustand der Zeit oder des Zählers beinhalten,
- eine Methode zum Aufruf des FB darstellen.

Dies ermöglicht eine flexible Parametrierung beim Aufruf eines Funktionsbausteins und wird am Beispiel eines Vor-/Rückwärtszählers dargestellt:

```
VAR
  Zaehler  :  CTUD;     (* Vor-/Rückwärtszählers *)
END_VAR
```

Bsp. 2.12. Deklaration eines Vor-/Rückwärtszählers nach IEC 61131-3

Auf die Ein- und Ausgänge dieses Zählers kann nach dieser Deklaration mit Hilfe einer durch die IEC 61131-3 implizit definierten Datenstruktur zugegriffen werden, die in Bsp. 2.13 zur Verdeutlichung als Ersatzdarstellung angegeben wird.

```
TYPE CTUD :    (* Datenstruktur einer FB-Instanz mit FB-Typ CTUD *)
  STRUCT
    (* Eingänge *)
    CU :        BOOL;    (* count up *)
    CD :        BOOL;    (* count down *)
    R  :        BOOL;    (* reset *)
    LD :        BOOL;    (* load *)
    PV :        INT;     (* preset value *)
    (* Ausgänge *)
    QU :        BOOL;    (* output up *)
    QD :        BOOL;    (* output down *)
    CV :        INT;     (* current value *)
  END_STRUCT;
END_TYPE
```

Bsp. 2.13. Datenstruktur zum Vor-/Rückwärtszähler (Standard-FB) in Bsp. 2.12 als Ersatzdarstellung

Die in Bsp. 2.13 dargestellte Datenstruktur spiegelt die Formalparameter (Aufrufschnittstelle) und Rückgabewerte des Std.-FB CTUD wider. Sie stellt in diesem Sinne die Aufrufer-Sicht für diesen FB dar. Im FB enthaltene lokale oder als extern deklarierte Variablen bleiben verdeckt.

Diese Datenstruktur wird durch das Programmiersystem automatisch verwaltet und kann zur Parametrierung eines FBs beim Aufruf, wie in Bsp. 2.14 in AWL gezeigt, komfortabel benutzt werden:

```
LD    34
ST    Zaehler.PV     (* Zählwert setzen *)
LD    %IX7.1
ST    Zaehler.CU     (* hochzählen *)
LD    %M3.4
ST    Zaehler.R      (* Zähler rücksetzen *)
CAL   Zaehler        (* Aufruf des FB mit den aktuellen Parametern *)
LD    Zaehler.CV     (* aktuellen Zählstand  abfragen *)
```

Bsp. 2.14. Parametrierung und Aufruf des Vor-/Rückwärtszählers in Bsp. 2.12

In diesem Beispiel wird die Instanz Zaehler mit den Parametern 34, %IX7.1 und %M3.4 versorgt, bevor Zaehler in der fett dargestellten Zeile mit CAL aufgerufen wird. Anschließend wird der aktuelle Zählstand abgefragt.

Wie in Bsp. 2.14 zu sehen ist, wird auf die FB-Ein- und Ausgänge mit Hilfe des FB-Instanznamens und einem trennenden Punkt zugegriffen, wie es für Strukturvariablen (vgl. Abschn. 3.5.2) ebenfalls festgelegt ist.

Beim Aufruf nehmen nicht verwendete Eingangs- oder Ausgangsparameter Anfangswerte an, die ggf. innerhalb des FB anzugeben sind.

In Abschn. 4.1.4 werden zwei weitere Methoden vorgestellt, mit denen FBs in AWL über ihre Instanznamen aufgerufen werden können.

Instanz bedeutet „Gedächtnis“.
Durch die Deklaration mehrerer Variablennamen für denselben FB-Typ wird für jede so gebildete Instanz gewissermaßen eine „Daten-Kopie“ des FBs im Speicher erzeugt. Diese Kopie beinhaltet sämtliche Werte der lokalen (VAR) sowie Ein- und Ausgangs-Variablen (VAR_INPUT, VAR_OUTPUT) des FBs (Datensatz). Sie erstreckt sich nicht auf VAR_IN_OUT (keine Werte selbst, sondern nur Zeiger auf Variable) oder VAR_EXTERNAL (sind globale Variable).

Dadurch kann die Instanz über mehrere Aufrufe hinweg lokale Datenwerte sowie die Ein- und Ausgangsparameter speichern, besitzt also eine Art „Gedächtnis“. Ein solches Gedächtnis ist für FBs wie Flip-Flops oder Zähler wichtig, deren Verhalten abhängig vom aktuellen *Zustand* ihrer Merker bzw. Zählwerte ist.

Sämtliche Variablen dieses Gedächtnisses liegen in einem der FB-Instanz (durch die Deklaration) *fest* zugeordneten Speicherbereich. Dieser Speicherbereich muß daher *statisch* sein.

Dies bedeutet, daß für die Verwaltung lokaler *temporärer* Variablen nicht wie oft üblich der Programmkeller (engl.: stack) verwendet werden kann!

Insbesondere bei Funktionsbausteinen, die große lokale Datenbereiche wie Tabellen verwalten, können solche lokalen Daten zu einem (unnötig) großen statischen Speicherbedarf der FB-Instanz führen.

Programmiersysteme verwenden daher neben den statischen Datenbereichen VAR ... END_VAR zusätzlichen einen weiteren Typ wie „VAR_DYN ... END_VAR“, der auf den Stack der SPS abgebildet wird, um Speicherplatz zu sparen. Der SPS-Programmierer könnte mit einem solchen dynamischen Datenbereich den Platzbedarf seiner temporären Daten optimieren. Solche Überlegungen werden ebenfalls in der IEC-Normungsgruppe diskutiert und werden voraussichtlich in den Technical Report 2 einfließen.

Ebenso können umfangreiche Ein- und Ausgangsparameter zu speicherplatzintensiven FB-Instanzen führen. Dabei kann die Verwendung von VAR_IN_OUT anstelle von VAR_INPUT bzw. VAR_OUTPUT helfen, Speicherplatz zu sparen.

In Abschn. 2.3.2 wurde beschrieben, daß Ein- und Ausgangsvariablen von POEs Einschränkungen beim lesenden bzw. schreibenden Zugriff unterliegen. Dies ist insbesondere für die FB-Instanz von Bedeutung:

- Eingangsparameter (Formalparameter) einer FB-Instanz bleiben für den nächsten Aufruf erhalten. Könnte der aufgerufene FB seine Eingangsvariablen ändern, wären diese beim nächsten Aufruf der Instanz verfälscht, ohne daß die aufrufende POE dies merken würde.
- Ausgangsparameter (Rückgabewerte) einer FB-Instanz bleiben ebenfalls zwischen Aufrufen erhalten. Könnten sie durch die aufrufende POE verändert werden, könnte der aufgerufene FB dies nicht berücksichtigen und von falschen Annahmen über seinen eigenen Ausgangszustand ausgehen.

FB-Instanzen können mit Hilfe des Schlüsselworts RETAIN ebenso wie normale Variable batteriegepuffert werden, d.h. sie behalten bei einem Stromausfall ihre lokalen Werte und die ihrer Aufrufschnittstelle bei.

Abschließend wird ein Zusammenhang zwischen FB-Instanzen und der bisher üblichen Verwendung von Datenbausteinen (DB) gezeigt.

Zusammenhang zwischen FB-Instanz und Datenbaustein.
Vor Aufruf eines (bisherigen) FBs, der – außer Übergabeparametern – keinen lokalen Datenbereich besitzt, wird oft ein Datenbaustein aktiviert, der beispielsweise Rezepturdaten oder Parametrierungsinformation besitzt. Innerhalb des FBs kann dieser DB auch als lokaler Datenbereich dienen. D.h. der Programmierer kann einen (bisherigen) FB mit individuellen „Instanzdaten“ verwenden, wobei er die eindeutige Zuordnung zwischen Daten und FB selbst zu kontrollieren hat. Diese Daten bleiben wie bei Instanzen ebenfalls zwischen den Aufrufen des FB erhalten, da der DB global zur Verfügung steht, wie Bsp 2.15 veranschaulicht:

```
SPA   DB 14    (* globaler DB *)
SPA   FB 14    (* Aufruf FB *)
...
```

a) bisheriges DB/FB-Paar

```
VAR_GLOBAL
  FB_14 : FB_Bsp;     (* globale Instanz *)
END_VAR
CAL   FB_14           (* Aufruf FB-Instanz *)
...
```

b) FB-Instanz nach IEC 61131-3

Bsp. 2.15. Verwendung eines übliches DB/FB-Paares ähnlich wie eine FB-Instanz nach IEC 61131-3.

Auf diese Methode wird in Abschn. 7.8 noch einmal eingegangen.

Diese Form der Instanziierung beschränkt sich auf Funktionsbausteine, sie ist für Funktionen (FUNCTION) prinzipiell nicht möglich.

Programme werden innerhalb der Konfiguration als oberste Hierarchie-Ebene ebenfalls instanziiert und aufgerufen. Allerdings unterscheidet sich diese (mächtigere) Instanz-Form von der des FBs, indem sie bei gleichzeitiger Zuordnung zu Tasks zur Bildung von Laufzeitprogrammen führt. Dies wird in Kap. 6 beschrieben.

2.4.2 Wiederverwendbarkeit und Objektorientierung von FB

Funktionsbausteine unterliegen einigen Einschränkungen, die ihre Wiederverwendbarkeit sicherstellen:

- Die Deklaration von Variablen mit fester Zuordnung zu SPS-Adressen (vgl. Kap. 3: „Direkt dargestellte Variablen“: %Q, %I, %M) als lokale Variable ist in Funktionsbausteinen unzulässig, damit sie unabhängig von spezieller SPS-Hardware bleiben können. Eine Verwendung der SPS-Adressen als globale Variable in VAR_EXTERNAL bleibt davon unberührt.
- Die Deklaration von Zugriffspfaden mit VAR_ACCESS (vgl. Kap. 3) oder globalen Variablen mit VAR_GLOBAL ist ebenfalls nicht zulässig. Durch VAR_EXTERNAL kann auf globale Daten und damit indirekt auch auf Zugriffspfade zugegriffen werden.
- Dateninformation von außerhalb kann ausschließlich über die zulässigen Variablenarten als POE-Schnittstelle (Parameter und externe Variable) in den FB gebracht werden. Es gibt keine „Vererbungs-“ Eigenschaften wie in manchen anderen Programmiersprachen.

Daher werden Funktionsbausteine auch als *gekapselt* bezeichnet, um auszudrücken, daß sie universell verwendbar und bei ihrer Ausführung frei von unerwünschten Seiteneffekten sind. Die (lokalen) Daten des FB und damit ihre Funktion hängen nicht unmittelbar von globalen Variablen, der SPS-Peripherie oder systemweiten Kommunikationspfaden ab. FBs können solche Datenbereiche nur indirekt über ihre (dokumentierbare) Schnittstelle manipulieren.

Im vorangegangenen Abschnitt wurde das Instanz-Modell der Funktionsbausteine mit den Eigenschaften „Strukturierung“ und „Gedächtnis“ erläutert. Zusammen mit den Eigenschaften zur Wiederverwendbarkeit ergibt sich eine neuartige Sicht von Funktionsbausteinen, die man folgendermaßen umschreiben könnte:

> „Ein FB ist eine eigenständige, nach außen gekapselte Datenstruktur mit einer auf ihr definierten Berechnungsvorschrift“.

Die Berechnung wird durch den Anweisungsteil des FBs dargestellt. Die Datenstruktur entspricht der FB-Instanz und kann „aufgerufen“ werden, was bei normalen Datenstrukturen (Abgeleitete Datentypen bzw. Typdefinitionen siehe Abschn. 3.3) nicht möglich ist. Von jedem FB-Typ können beliebig viele Instanzen gebildet werden, die voneinander unabhängig sind. Jede Instanz besitzt einen eindeutigen Namen mit eigenem Datenbereich.

Aufgrund dieser Sichtweise werden Funktionsbausteine nach IEC 61131-3 auch als *objektorientiert* bezeichnet. Allerdings dürfen diese Eigenschaften nicht mit den umfassenderen Sprachmitteln gleichgesetzt werden, die heute moderne „Objektorientierte Programmiersprachen (→OOP)“ wie beispielsweise C++ mit ihren Klassenhierarchien bieten!

Zusammengefaßt arbeiten FBs auf ihrem *eigenen Datensatz*, der Ein- und Ausgangsvariablen sowie lokale Variablen enthält. Bei bisheriger SPS-Programmierung greifen FBs üblicherweise auf *globale Daten* (wie Merkerbereiche und Datenbausteine) zu.

2.4.3 Variablenarten in FBs

Ein FB kann keine oder beliebig viele Ein- und Ausgangsparameter sowie lokale und als extern deklarierte Variablen benutzen.

Zusätzlich oder alternativ zur Batteriepufferung einer FB-Instanz können auch die lokalen oder Ausgangsvariablen *innerhalb* des FB-Deklarationsteils als gepuffert deklariert werden.

Die Werte von Eingangs- bzw. Ein-/Ausgangsvariablen sind innerhalb der FB-Deklaration nicht als gepuffert (RETAIN) angebbar, da es sich um Übergabe-Parameter handelt.

Für VAR_IN_OUT ist zu beachten, daß ihre **Zeiger** auf Variablen zwar in einer mit RETAIN geschützten Instanz erhalten bleiben, die dazugehörigen Variablen-**werte** selbst jedoch verloren sein können, wenn sie nicht in der aufrufenden POE als batteriegepuffert deklariert wurden.

Direkt dargestellte Variable dürfen in FBs wegen der geforderten Peripherie-Unabhängigkeit nicht lokal deklariert, sondern nur als globale Variable über VAR_EXTERNAL „importiert“ werden.

Eine Besonderheit bei der Variablendeklaration stellen flankengesteuerte Parameter in der IEC 61131-3 dar. Die IEC stellt die Standard-FBs R_TRIG und F_TRIG zur Flankenerkennung zur Verfügung (siehe auch Kap. 5).

Die Verwendung der Flankenerkennung als Attribut von Variablenarten ist nur für Eingangsvariablen vorgesehen (siehe Abschn. 3.5.4).

Für die Definition einiger SPS-typischen Grundfunktionen wie Zeiten und Zähler werden FBs benötigt, da sie eigene Zustandsinformationen besitzen. Dazu definiert die IEC 61131-3 eine Reihe von Standard-FBs, die in Kap. 5 anhand von Beispielen erläutert werden.

2.5 Die Funktion

Der Grundgedanke bei der Definition von Funktionen nach IEC 61131-3 besteht darin, daß Funktionen die Werte ihrer Eingangsvariablen über die Anweisungen im Funktionsrumpf zu einem *eindeutigen* Funktionswert („seiteneffektfrei“) verknüpfen. In diesem Sinne stellen sie im einfachsten Fall eine hersteller- oder anwenderspezifische Erweiterung des Operationsvorrats der SPS dar.

Für Funktionen gilt daher die Regel: gleiche Eingangswerte liefern *immer* denselben Funktionswert, unabhängig davon, wie oft oder zu welchem Zeitpunkt die Funktion aufgerufen wird, sie besitzen also kein Gedächtnis wie die FBs.

Funktionen können als AWL-Operatoren ebenso verwendet werden wie als Operanden in ST-Ausdrücken. Wie FB-Typen, jedoch nicht wie die FB-Instanzen, sind auch die Funktionen sämtlichen POEs eines Projekts bekannt.

Um die Definition der Grund-Funktionalität eines SPS-Systems zu vereinfachen und gleichzeitig zu vereinheitlichen, legt die IEC 61131-3 eine Sammlung von häufig verwendeten Standard-Funktionen fest, deren Eigenschaften, Laufzeitverhalten und Aufrufschnittstelle normiert sind (siehe auch Kap. 5).

Mit Hilfe benutzerdefinierter Funktionen wird diese Sammlung um gerätespezifische Erweiterungen oder individuelle Funktionsbibliotheken ergänzt.

Ein ausführliches FUN-Beispiel ist in Anh. C zu finden. Funktionen besitzen gegenüber den beiden anderen POE-Typen einige Einschränkungen, die zur Seiteneffektfreiheit benötigt werden und dazu dienen, daß sie auch in Ausdrücken (z.B. in ST) verwendet werden können. Dies wird im nächsten Abschnitt näher betrachtet.

2.5.1 Variablenarten in Funktionen und ihr Funktionswert

Funktionen besitzen einen oder beliebig viele Eingangsparameter. Im Unterschied zu FBs besitzen sie keine Ausgangsparameter, sondern liefern genau *ein Element* als Funktionswert zurück.

Der Funktionswert kann ein beliebiger, auch abgeleiteter Datentyp sein. Ein einfacher boolescher Wert oder ein Gleitpunkt-Doppelwort ist ebenso zugelassen wie ein Feld oder eine komplexe Datenstruktur, die aus mehreren Datenelementen bestehen kann (Multielement-Variable), wie in Kap. 3 anschließend beschrieben wird.

In jeder Sprache der IEC 61131-3 wird der Funktionsname als besondere Variable innerhalb des Funktionsrumpfs verwendet, um explizit den Funktionswert zuweisen zu können.

Da Funktionen bei gleichen Eingangsvariablen dasselbe Ergebnis liefern müssen, dürfen sie sich keine Zwischenergebnisse von einem Aufruf zum anderen merken („gedächtnislos“).

Daher können Funktionen lokale Variablen für Zwischenergebnisse benutzen, diese gehen allerdings bei Beendigung der Funktion verloren. Lokale Variablen können daher auch nicht als gepuffert deklariert werden.

Ebenso dürfen innerhalb Funktionen keine Funktionsbausteine wie beispielsweise Zeiten, Zähler oder Flankenerkennungen aufgerufen werden.

Auch die Verwendung von globalen Variablen ist innerhalb von Funktionen unzulässig.

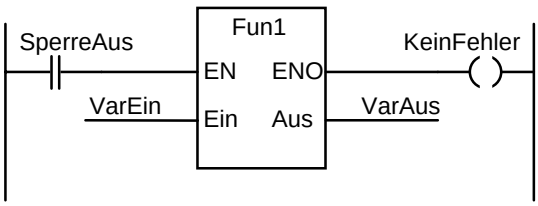
Wie ein SPS-System im Falle einer Spannungs-Unterbrechung und danach mit (insbesondere komplexen) Funktionen und deren aktuellen Variablenwerten umgeht, wird durch die Norm nicht festgelegt. Daher muß diejenige POE für die gewünschte Variablenpufferung sorgen, die eine Funktion aufruft.

Auf jeden Fall ist es sinnvoll, in Unterprogrammen mit sensiblen Daten anstelle einer Funktion ggf. einen FB zu verwenden.

Weiterhin können in Funktionen – wie in FBs – keine Direkt dargestellten Variablen (physikalische Speicheradressen der SPS) deklariert werden.

2.5.2 Ausführungssteuerung mit EN und ENO

In KOP und FBS gibt es für Funktionen gegenüber den übrigen Programmiersprachen der IEC 61131-3 eine Besonderheit: hier muß eine Funktion nach IEC 61131-3 mit je einem zusätzlichen Ein- und Ausgang dargestellt werden. Dazu dienen der boolesche Freigabe-Eingang EN (Enable In) und der boolesche Ausgang ENO (Enable Out).



Bsp. 2.16. Aufruf einer Funktion mit EN/ENO in KOP

Bsp. 2.16 zeigt die grafische Darstellung für den Aufruf der Funktion Fun1 mit EN und ENO in KOP. In diesem Beispiel wird Fun1 nur dann ausgeführt, wenn der Eingang EN auf „1“ (TRUE) ist, d.h. der Kontakt SperreAus geschlossen ist. Dann

liegt nach fehlerfreier Abarbeitung der Funktion am Ausgang ENO ebenfalls TRUE, so daß die Variable KeinFehler gesetzt bleibt.

Mit Hilfe des EN/ENO-Paars ist es in KOP-Darstellung möglich, jede Funktion, also auch solche mit nicht-booleschen Ein- und Ausgängen wie Fun1 in Bsp. 2.16, in den „Stromfluß“ der KOP-Linien zumindest teilweise zu integrieren.

Aus dieser Vorstellung leitet sich die Bedeutung von EN/ENO ab, die in Tab. 2.6 skizziert wird:

EN	**Erläuterung**[a]	**ENO**
EN = FALSE	Ist EN beim Aufruf der Funktion FALSE, so darf der Anweisungsteil der Funktion nicht ausgeführt werden, sie wird also nicht durchlaufen. Der Ausgang ENO wird beim Verlassen der Funktion ebenfalls auf FALSE gesetzt, um zu signalisieren, daß die Funktion nicht ausgeführt wurde.	ENO = FALSE
EN = TRUE	Ist EN beim Aufruf der Funktion TRUE, so wird der Anweisungsteil der Funktion normal ausgeführt. ENO wird in diesem Fall **vor** Ausführung des Anweisungsteils auf TRUE gesetzt.	ENO = TRUE
	ENO kann anschließend durch Anweisungen im Rumpf der Funktion auf TRUE oder FALSE gesetzt werden.	ENO = indiv. Wert
	falls bei der Ausführung des Anweisungsteils einer der Programm- oder Systemfehler auftritt, wie sie in Anh. E beschrieben werden, wird ENO durch das SPS-System auf FALSE zurückgesetzt.	ENO = FALSE (Fehler aufgetreten)

a TRUE = logisch „1“, FALSE = logisch „0“

Tab. 2.6. Bedeutung von EN und ENO in Funktionen

Aus Tab. 2.6 ergibt sich, daß EN und ENO den *Kontrollfluß* in einem grafischen Netzwerk steuern, nämlich die bedingte Ausführung einer Funktion mit der entsprechenden Nachbehandlung. EN kann nicht nur mit einem Kontakt wie in Bsp. 2.16 beschaltet werden, sondern auch mit einer Verknüpfung mehrerer Kontakte. ENO kann entsprechend mit einer komplexeren Nachverknüpfung (Kontakte und Spulen) versehen werden. Diese Kontrollfluß-Steuerung sollte im SPS-Programm allerdings logisch von den KOP-Verknüpfungen unterschieden werden, die den *Datenfluß* des Netzwerks bilden!

Diese speziellen Ein- und Ausgänge EN und ENO werden in der IEC 61131-3 nicht wie gewöhnliche Ein- oder Ausgänge von Funktionen behandelt, sondern sind ausschließlich für die eben beschriebenen Aufgaben vorgesehen. Für

Funktionsbausteine ist diese Möglichkeit in KOP oder FBS durch die IEC 61131-3 bisher nicht vorgesehen, wird allerdings voraussichtlich noch ergänzt werden.

In den anderen Sprachen der IEC 61131-3 ist die Verwendung dieser zusätzlichen Ein- und Ausgänge nicht vorgesehen mit Ausnahme von FBS. Hier kann eine entsprechende Darstellung mit EN/ENO auch zusätzlich realisiert sein.

Der Funktionsaufruf in Bsp. 2.16 kann in AWL dargestellt werden, falls das Programmiersystem EN/ENO implizit als Systemvariable unterstützt.

Unterstützt ein Programmiersystem den Gebrauch von EN und ENO, sind die so programmierten POEs schwierig in eine textuelle Form zu transformieren. Um dies zu ermöglichen, müßten EN/ENO auch Schlüsselworte in AWL oder ST sein und dort wie in KOP/FBS automatisch generiert werden. Dann könnte eine in KOP aufgerufene Funktion beispielsweise auch in AWL geschrieben werden und das ENO-Flag im Fehlerfall setzen. Ansonsten könnten in KOP/FBS nur solche Funktionen aufgerufen werden, die auch in einer dieser Sprachen geschrieben wurden. Die Norm macht allerdings keine Aussage darüber, wie EN und ENO als Schlüsselworte bzw. Grafik-Elemente in KOP oder FBS zu verwenden sind, um sie setzen oder rücksetzen zu können.

Außerdem stellt sich die Frage, inwieweit die Verwendung von EN und ENO bei Vergleichsfunktionen (Std.-FUN, siehe Anh. A) vorteilhaft ist. Ein Vergleicher besitzt dann zwei boolesche Ausgänge, die jeweils mit einer Spule versehen werden könnten. Falls dieser Vergleicher im Parallelzweig eines KOP-Netzwerks liegt, müssen ENO und der Ausgang Q getrennt voneinander weiterverknüpft werden: ENO führt den Parallelzweig fort, während mit Q gewissermaßen ein neues Teilnetzwerk beginnt.

Aufgrund dieser Komplexität verwenden nur einige der heutigen IEC-Programmiersysteme EN und ENO. Anstatt in KOP/FBS das boolesche Paar EN und ENO *fest* vorzuschreiben, gibt es auch denkbare Alternativen:

- EN und ENO können in sämtlichen Programmiersprachen explizit und implizit benutzt werden,
- jede in KOP/FBS aufrufbare Funktion muß mindestens einen binären Eingang und Ausgang besitzen,
- nur Standard-Funktionen besitzen ein EN/ENO-Paar, das für benutzerdefinierte Funktionen nicht verwendet werden kann.

Die dritte Alternative kommt der Definition nach IEC 61131-3 am nächsten. Dies würde allerdings bedeuten, daß EN und ENO vom SPS-Programmierer nicht beeinflußbare System-Variablen in der SPS sind.

2.6 Das Programm PROGRAM

Während Funktionen und Funktionsbausteine „Unterprogramme“ darstellen, bilden POEs vom Typ PROGRAM das „Hauptprogramm“ der SPS. Auf einer Multitasking-fähigen Steuerungs-Hardware können mehrere Hauptprogramme parallel ablaufen. Daher besitzen PROGRAMs gegenüber FBs besondere Eigenschaften, die hier erläutert werden.

Für ein PROGRAM stehen dem SPS-Programmierer zusätzlich zu den Eigenschaften der Funktionsbausteine folgende weitere Merkmale zur Verfügung:

- Deklaration Direkt dargestellter Variablen zum Ansprechen physikalischer SPS-Adressen (%Q, %I, %M) ist zulässig,
- Verwendung von VAR_ACCESS oder VAR_GLOBAL ist möglich,
- PROGRAM wird innerhalb der SPS-Konfiguration einer Task zugeordnet, um ein Laufzeitprogramm zu bilden, d.h. Programme werden nicht explizit durch andere POEs aufgerufen.

Ein Programm beinhaltet die Zuordnung von Variablen zur SPS-Peripherie, indem Direkt dargestellte bzw. Symbolische Variable global oder als POE-Parameter verwendet werden.

Weiterhin werden im Programm Mechanismen beschrieben, mit denen Kommunikation und globaler Datenaustausch zu anderen Programmen (innerhalb und außerhalb der Konfiguration) stattfindet. Dafür wird die Variablenart VAR_ACCESS verwendet.

Diese Eigenschaften können auch auf der Ebene von Ressourcen und Konfigurationen genutzt werden, was bei komplexen SPS-Projekten zu empfehlen ist.

Für kleinere Projekte kann man durch die Funktionaliät der Programm-POE auch ohne eine Konfigurationsdefinition auskommen: Das PROGRAM übernimmt die Aufgaben der Zuordnung des Programms zur SPS-Peripherie.

Solche Möglichkeiten sind abhängig von der Funktionalität eines Programmiersystems und werden hier nicht weiter betrachtet.

Ein ausführliches PROGRAM-Beispiel ist in Anh. C zu finden.

Laufzeit-Eigenschaften und -Sonderbehandlung eines PROGRAM in einer SPS-CPU kommen in der Zuordnung des PROGRAM zu TASKs zum Ausdruck. Dabei wird das Programm instanziiert, kann also mehreren Tasks zugeordnet und dadurch in der SPS mehrfach gleichzeitig ausgeführt werden. Diese Instanziierung unterscheidet sich allerdings von der für FB-Instanzen.

Diese Zuordnung wird innerhalb der CONFIGURATION vorgenommen und in Kap. 6 erläutert.

2.7 Aufrufe von Funktionen und Funktionsbausteinen

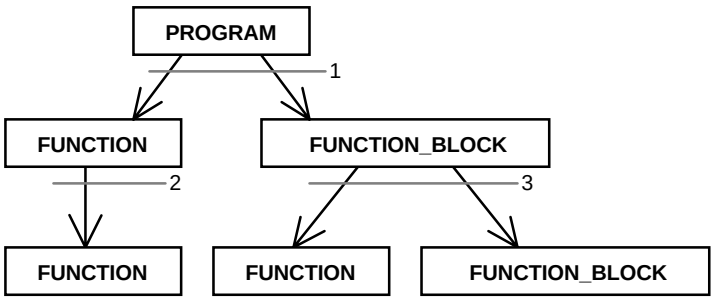
In diesem Abschnitt wird auf die Besonderheiten eingegangen, die beim Aufruf von Funktionen und Funktionsbausteinen zu beachten sind. Diese Besonderheiten gelten sowohl für die Standard- als auch die benutzerdefinierten Funktionen und Funktionsbausteine.

Die nachfolgenden Beispiele werden weiterhin in AWL angegeben. Verwendung in ST sowie die grafische Darstellung in KOP und FBS sind Gegenstand des Kap. 4.

2.7.1 Gegenseitiger Aufruf zwischen POEs

Für den gegenseitigen Aufruf von POE-Typen gelten die durch Abb. 2.7 veranschaulichten Regeln:

- PROGRAM darf FUNCTION_BLOCK und FUNCTION aufrufen, jedoch nicht umgekehrt,
- FUNCTION_BLOCK darf FUNCTION aufrufen, jedoch nicht umgekehrt,
- POE-Aufrufe dürfen nicht rekursiv sein (POE darf „sich nicht selbst“ aufrufen), auch nicht indirekt.



1 Programm ruft Funktion oder Funktionsbaustein auf

2 Funktion ruft Funktion auf

3 Funktionsbaustein ruft Funktion oder Funktionsbaustein auf

Abb. 2.7. Aufruf-Möglichkeiten zwischen POE-Typen: drei erlaubte Fälle

Programme und FB-Instanzen können FB-Instanzen aufrufen, während Funktionen FB-Instanzen nicht aufrufen können, da sonst die Seiteneffektfreiheit der Funktionen nicht mehr gewährleistet wäre.

Programme (PROGRAM) werden in der Konfiguration mit Hilfe der TASK zu Laufzeitprogrammen instanziiert und durch die Ressource aufgerufen.

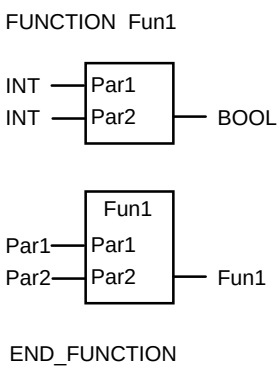
2.7.2 Rekursive Aufrufe sind unzulässig

Die IEC 61131-3 legt fest, daß sich POEs nicht selbst (auch nicht indirekt) aufrufen dürfen (*Rekursion*), d.h. eine POE kann nicht eine POE desselben Typs aufrufen. Das hieße, daß eine POE „durch sich selbst“ definiert wird, wenn ihr Name innerhalb des eigenen Rumpfes deklariert bzw. aufgerufen wird. In manchen anderen Programmiersprachen ist Rekursion allerdings möglich.

Ließe man Rekursion zu, wäre eine Berechnung des zur Laufzeit maximalen Speicherbedarfs eines rekursiven SPS-Programms durch das Programmiersystem nicht mehr möglich.

Rekursion kann immer durch entsprechende iterative Konstrukte, d.h. durch Bildung von Programmschleifen ersetzt werden.

Die beiden folgenden Abbildungen geben zwei Beispiele für unzulässige Aufrufe:




```
FUNCTION  Fun1 : BOOL

VAR
  Par1, Par2 : INT;
END_VAR


LD    Par1
Fun1  Par2
ST    Fun1

END_FUNCTION
```

Bsp. 2.17. Unzulässiger rekursiver Aufruf einer Funktion in grafischer und AWL-Darstellung: Aufrufverschachtelung.

In diesem Bsp. 2.17 wird innerhalb der Funktion Fun1 dieselbe Funktion nochmals aufgerufen.

Der obere Teil dieser grafischen Darstellung beschreibt den Deklarationsteil der Funktion (Eingangsvariablen Par1 und Par2 vom Datentyp INT) sowie den Funktionswert vom Typ BOOL.

Im unteren Teil wird diese Funktion mit denselben Eingangsvariablen versorgt und aufgerufen, so daß sich zur Laufzeit der Funktion eine endlose (rekursive) Kette von Aufrufen bilden würde.

```
FUNCTION_BLOCK FunBst
VAR_INPUT
  In1   :   DINT;                          (* Eingangsvariable *)
END_VAR
VAR
  InstFunBst   : FunBst;          (* Instanz desselben FB-Typs unzulässig *)
  Var1 :   DINT;                           (* lokale Variable *)
END_VAR
...
  CALC   InstFunBst (In1 := Var1);    (* rekursiver Aufruf unzulässig! *)
...
END_FUNCTION_BLOCK
```

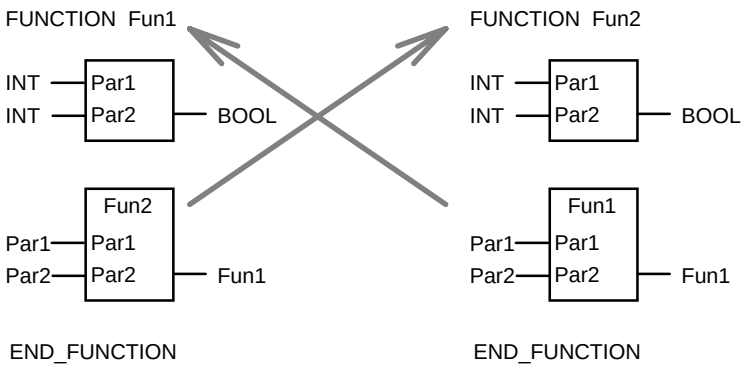
Bsp. 2.18. Unzulässiger rekursiver Aufruf eines Funktionsbausteins in AWL: Verschachtelung bereits im Deklarationsteil.

Bsp. 2.18 zeigt den Funktionsbaustein FunBst, in dessen lokaler Variablendeklaration (VAR) eine Instanz vom eigenen *Typ* (FunBst) deklariert wird. Im Programmrumpf wird diese Instanz aufgerufen, so daß sich eine endlos tiefe Verschachtelung bereits innerhalb des Deklarationsteil bei der Instanzbildung ergäbe und die zur Laufzeit benötigte Größe der Instanz nicht mehr bestimmt werden kann.

Ob POEs im SPS-Programm durch (un)gewollte Programmierung nicht trotzdem rekursiv aufgerufen werden, muß der Programmierer selbst überprüfen oder sein Programmiersystem bzw. SPS-System vornehmen.

Diese Prüfung kann bereits bei der Programmerstellung anhand des POE-Aufrufbaums durchgeführt werden, da das Verbot der Rekursion an die FB-Typen und nicht an deren Instanzen gebunden ist. Dies ist sogar dann möglich, wenn FB-Instanznamen als Eingangsparameter übergeben werden (vgl. Abschn. 2.7.5).

Das folgende Beispiel zeigt, wie rekursive Aufrufe auch ohne direkten Aufruf einer Funktion oder einer FB-Instanz von sich selbst entstehen können, indem sie sich wechselseitig aufrufen.



Bsp. 2.19. Rekursion durch wechselseitigen Aufruf in grafischer Darstellung

Auch solche Arten der Rekursion sind in der IEC 61131-3 prinzipiell nicht zugelassen. Die Aufrufbedingung kann man wie folgt definieren: eine von einem Aufrufer A aufgerufene POE sowie sämtliche in der Aufrufhierarchie nach ihr stehenden POEs dürfen nicht den Namen des ersten Aufrufers A verwenden.

Rekursion wird durch die IEC 61131-3 im Unterschied zu den meisten bekannten höheren Programmiersprachen (wie C) also generell verhindert. Dies macht SPS-Programme sicherer gegen Programmierfehler durch ungewollte Rekursion.

2.7.3 Aufruf mit Formalparametern

Beim FUN/FB-Aufruf werden Eingangsparameter an die Eingangsvariablen der POE übergeben. Diese Eingangsvariablen werden auch als *Formalparameter* bezeichnet, die Eingangsparameter auch als *Aktualparameter*, um auszudrücken, daß sie die aktuellen Eingangswerte beinhalten.

Beim Aufruf einer Funktion können die Formalparameter ausdrücklich mit angegeben werden oder nicht. Diese Angabe ist sowohl abhängig vom POE-Typ (FUN oder FB) als auch von der Programmiersprache, in der dieser POE-Aufruf definiert wird (vgl. a. Kap. 4).

Tab. 2.7 gibt einen Überblick darüber, welche POE-Typen in textueller und grafischer Darstellung mit bzw. ohne Angabe von Formalparametern aufgerufen werden können.

Sprache	Funktion	Funktionsbaustein	Programm
AWL	ohne	mit [a]	mit
ST	ohne oder mit [b]	mit	mit
KOP und FBS	mit [b]	mit	mit

a auf drei verschiedene Arten möglich, vgl. Abschn. 4.1.4
b bei Std.-FUN: soweit Parametername überhaupt vorhanden

Tab. 2.7. Mögliche explizite Angabe von Formalparametern („ohne“ bzw. „mit“) beim POE-Aufruf

Bei FBs und PROGs sind die Formalparameter immer, unabhängig von der Programmiersprache, ausdrücklich anzugeben. In AWL gibt es dazu verschiedene Möglichkeiten (siehe Abschn. 4.1.4).

Funktionen können in ST wahlweise mit oder ohne Angabe der Formalparameter versorgt werden.

Viele Formalparameter von Standard-Funktionen besitzen keinen Namen (vgl. Anh. A). Daher können diese in grafischer Darstellung nicht angezeigt und in textueller Darstellung nicht explizit angegeben werden.

Die IEC 61131-3 gibt für den Aufruf benutzerdefinierter Funktionen in AWL nicht ausdrücklich an, ob Formalparameter angegeben werden dürfen. Damit solche Funktionsaufrufe konsistent zu den Aufrufen der Standardfunktionen bleiben, wird angenommen, daß der Name der Formalparameter bei Funktionsaufrufen in AWL prinzipiell *nicht* mit angegeben werden darf.

Beim Aufruf von Standard-Funktionen und -Funktionsbausteinen gelten dieselben Regeln. Bsp. 2.20 zeigt für FUN und FB zur Tab. 2.7 jeweils einen Beispiel-Aufruf.

FB-Deklaration:

```
FUNCTION_BLOCK FBaust
VAR_INPUT
  Par1 :   TIME;
  Par2 :   WORD;
  Par3 :   INT;
END_VAR

... (* Anweisungen *)
END_FUNCTION_BLOCK
```

FUN-Deklaration:

```
FUNCTION Funktion :
INT
VAR_INPUT
  Par1 :   TIME;
  Par2 :   WORD;
  Par3 :   INT;
END_VAR
... (* Anweisungen *)
END_FUNCTION
```

PROG-Deklaration:

```
PROGRAM Programm
 VAR_GLOBAL
   FunBst :   FBaust;
   VarGlob:   INT;

 END_VAR

 ... (* Anweisungen *)
 END_PROGRAM
```

```
(* 1. Aufrufe in AWL *)
LD          t#20:12
Funktion    %IW4, VarGlob                                  (* FUN-Aufruf *)
CAL FunBst  (Par1 := t#20:12, Par2 := %IW4, Par3 := VarGlob)  (* FB-Aufruf *)
```

```
(* 2. Aufrufe in ST *)
Funktion    (t#20:12, %IW4, VarGlob)                       (* FUN-Aufruf *)
Funktion    (Par1 := t#20:12, Par2 := %IW4, Par3 := VarGlob); (* FUN-Aufruf *)
FunBst      (Par1 := t#20:12, Par2 := %IW4, Par3 := VarGlob); (* FB-Aufruf *)
```

Bsp. 2.20. Äquivalente FUN- bzw. FB-Aufrufe mit und ohne explizite Angabe von Formalparametern in den textuellen Sprachen AWL und ST. Der Aufruf erfolgt jeweils in Programm.

In AWL wird der erste Eingangsparameter vor dem eigentlichen Aufruf als Aktuelles Ergebnis (AE) geladen, wie beim Aufruf der Funktion Funktion in Bsp. 2.20 zu sehen ist. Beim Funktionsaufruf werden die beiden anderen Parameter durch Kommata getrennt angegeben, sie können nicht zusätzlich mit der Angabe der Formalparameter versehen werden.

Die beiden dazu äquivalenten Aufrufe in ST können mit oder ohne Angabe der Formalparameter erfolgen. Die Eingangsparameter werden jeweils in Klammern eingeschlossen.

Der Aufruf der FB-Instanz FunBst erfolgt in diesem Beispiel sowohl in AWL als auch in ST mit vollständiger Angabe der drei Formalparameter.

Die Benutzung von Formal- und Aktualparametern in grafischer Darstellung wird in Bsp. 3.18 dargestellt.

2.7.4 Aufrufe mit fehlenden oder vertauschten Eingangsparametern

Funktionen und Funktionsbausteine können auch aufgerufen werden, wenn die Liste der Eingangsparameter nicht vollständig ist bzw. nicht jedem tatsächlich ein Wert zugewiesen wurde.

Bei *fehlenden* Eingangsparametern müssen die Namen der tatsächlich benutzten Formalparameter explizit mit angegeben werden, damit das Programmiersystem die richtige Zuordnung von Aktual- und Formalparametern vornehmen kann.

Auch wenn beim FUN/FB-Aufruf die *Reihenfolge* der Parameter geändert werden soll, ist die ausdrückliche Benennung der Formalparameter erforderlich. Diese Situationen werden in Bsp. 2.21 beispielhaft für AWL dargestellt.

```
(* 1.   vollständiger FB-Aufruf *)
CAL   FunBst (Par1 := t#20:12, Par2 := %IW4, Par3 := VarGlob);

(* 2.   vollständiger FB-Aufruf mit vertauschten Parametern *)
CAL   FunBst (Par2 := %IW4, Par1 := t#20:12, Par3 := VarGlob);

(* 3.   unvollständiger FB-Aufruf *)
CAL   FunBst (Par2 := %IW4);

(* 4.   unvollständiger FB-Aufruf mit vertauschten Parametern *)
CAL   FunBst (Par3 := VarGlob, Par1 := t#20:12);
```

Bsp. 2.21. Beispiele für FB-Aufruf aus Bsp. 2.20 mit fehlenden und vertauschten Parametern in AWL

Dies bedeutet, daß entweder sämtliche Formalparameter aufgeführt sein können, so daß die Parameter-Reihenfolge keine Rolle spielt, oder kein Formalparameter verwendet wird und die Reihenfolge zu beachten ist. Bei FB-Aufrufen sind Formalparameter immer anzugeben, für Funktionen ist dies sprachabhängig (vgl. Tab. 2.7).

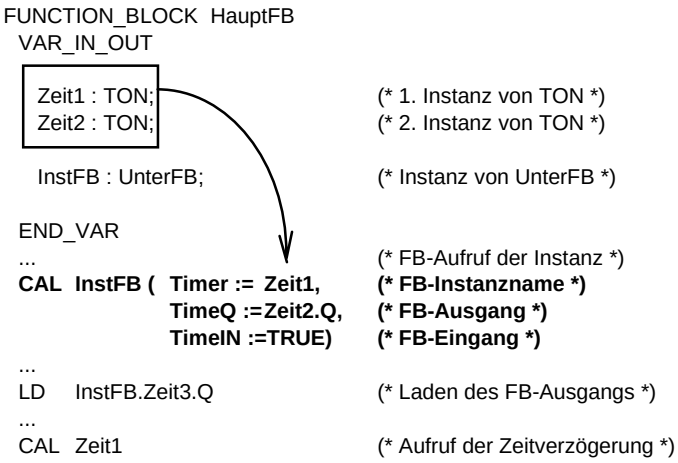
Für fehlende Zuweisungen an Eingangsvariablen gibt es die Möglichkeit, innerhalb des POE-Deklarationsteils Eingangsvariable zu „initialisieren“. Anstelle der fehlenden Aktualparameter wird dann standardmäßig der Anfangswert verwendet. Wird kein benutzerspezifischer Anfangswert angegeben, wird der Anfangswert der Standard-Datentypen der IEC 61131-3 verwendet, so daß Eingangsvariablen *immer* einen Wert besitzen.

Für FBs erfolgt eine Initialisierung jeweils nur für den ersten Aufruf einer Instanz. Danach sind immer die Werte des letzten Aufrufs vorhanden, da die Instanzdaten (inkl. Eingangsvariablen) erhalten bleiben.

2.7.5 FB-Instanzen als FB-Aktualparameter

Dieser Abschnitt beschreibt die Verwendung von FB-Instanznamen sowie deren Ein- und Ausgänge als Aktualparameter beim Aufruf anderer Funktionsbausteine.

Anhand des Bsp. 2.22 wird in diesem Abschnitt erläutert, welche Möglichkeiten in der IEC 61131-3 zur Verfügung stehen, indirekte Aufrufe oder Versorgung von FB-Instanzen vorzunehmen.



Bsp. 2.22. Übergabe von FB-Instanznamen Zeit1 und des Ausgangs Zeit2.Q als Aktualparameter eines anderen Funktionsbausteins. Die Zeiten Zeit1 und Zeit2 sind Instanzen des Standard-FB TON (Einschaltverzögerung, vgl. Kap. 5). UnterFB wird in Bsp. 2.23 deklariert.

Instanznamen bzw. Ein- und Ausgänge von Instanzen können als Aktualparameter für Eingangsvariable oder Ein-/Ausgangsvariable verwendet werden.
Um diese Verwendungsmöglichkeiten zu beschreiben, unterscheidet Tab. 2.8 zwischen den zulässigen (in der Tabelle: „ja“) und unzulässigen Fällen („nein“).

		Als Aktualparameter für UnterFB		Rückgabewert, Externe Variable
Instanz	Beispiel	VAR_INPUT	VAR_IN_OUT (Zeiger)	VAR_EXTERNAL VAR_OUTPUT
Instanzname	Zeit1	ja [a]	ja [b]	ja [c]
-Eingang	Zeit2.IN	-	-	-
-Ausgang	Zeit2.Q	ja	nein [d]	-

a Instanz kann in UnterFB nicht aufgerufen werden (kein indirekter FB-Aufruf möglich)
b Indirekter FB-Aufruf möglich, Instanz-Ausgang darf im UnterFB nicht verändert werden
c Direkter FB-Aufruf, Instanz-Ausgang darf im HauptFB nicht verändert werden
d Funktionswert einer Funktion ebenso nicht verwendbar

Tab. 2.8. Möglichkeiten, FB-Instanzen als FB-Aktualparameter zu verwenden bzw. indirekt aufzurufen. Die Spalte „Beispiel“ bezieht sich auf das Bsp. 2.22 Die letzte Spalte zeigt, daß FB-Instanzen auch als externe Variablen bzw. als Rückgabewerte verwendet werden können. Zeit2.IN darf nicht lesend verwendet und damit nicht als Parameter übergeben werden.

Wie aus dieser Übersicht hervorgeht, können nur einige Kombinationen verwendet werden, um Funktionsbausteinen in den verschiedenen Variablenarten Instanznamen oder deren Ein- und Ausgänge zu übergeben.

VAR_INPUT: FB-Instanzen sowie ihre Ausgänge können im UnterFB nicht aufgerufen bzw. verändert werden, wenn sie als VAR_INPUT übergeben wurden. Auf sie kann allerdings lesend zugegriffen werden.

VAR_IN_OUT: Die Übergabe eines Ausgangs einer FB-Instanz, dessen Zeiger übergeben werden würde, ist für diesen Variablenblock unzulässig. Dadurch wird eine versehentliche Änderung dieses Ausgangs verhindert. Dementsprechend ist auch eine Parametrierung einer VAR_IN_OUT-Variable mit dem Zeiger auf den Funktionswert einer Funktion **nicht** zulässig.

Die übergebene Instanz kann aufgerufen werden, so daß ein *indirekter FB-Aufruf* realisiert werden kann.

Ausgänge des übergebenen FB-Instanznamens dürfen nicht überschrieben werden. Auf FB-Instanz-Eingänge kann frei zugegriffen werden.

VAR_EXTERNAL, VAR_OUTPUT: FB-Instanzen werden direkt aufgerufen, ihre Ein- und Ausgänge können von der aufrufenden POE nur lesend weiterverarbeitet werden.

Beispiel für indirekten FB-Aufruf.
Das Bsp. 2.23 zeigt (zusammen mit Bsp. 2.22) die Verwendung einiger in Tab. 2.8 zulässigen Fälle innerhalb des Funktionsbausteins UnterFB.

```
FUNCTION_BLOCK  UnterFB
  VAR_INPUT
    TimeIN :  BOOL;        (* boolesche Eingangsvariable *)
    TimeQ  :  BOOL;        (* boolesche Eingangsvariable *)
  END_VAR
  VAR_IN_OUT
    Timer  :  TON;         (* Zeiger auf Instanz Zeit1 von TON - Ein-/Ausgangsvariable *)
  END_VAR
  VAR_OUTPUT
    Zeit3  :  TON;         (* 3. Instanz von TON *)
  END_VAR
  VAR
    Start : BOOL := TRUE; (* lokale boolesche Variable *)
  END_VAR
...
(* Indirekter Aufruf von Zeit1 mit Setzen/Abfragen der Aktualparameter über Timer *)
LD    Start
ST    Timer.IN         (* Starten des Timers Zeit1 *)
CAL   Timer            (* Indirekter Aufruf der Zeitverzögerung Zeit1 *)
LD    Timer.Q          (* Abfrage des Ausgangs von Zeit1 *)
...
(* Direkter Aufruf von Zeit3; indirekter Zugriff auf Zeit2 *)
LD    TimeIN           (* indirekte Abfrage des Eingangs von Zeit2 nicht möglich *)
ST    Zeit3.IN         (* Starten des Timers über Zeit3.IN *)
CAL   Zeit3            (* Direkter Aufruf der Zeitverzögerung Zeit3 *)
LD    Zeit3.Q          (* Abfrage des Ausgangs über Zeit3.Q *)
...
LD    TimeQ            (* indirekte Abfrage des Ausgangs von Zeit2 *)
...
END_FUNCTION_BLOCK
```

Bsp. 2.23. Möglichkeiten des indirekten Aufrufs des Zeitverzögerung-FB Zeit1 aus Bsp. 2.22 sowie der Verwendung seiner Ein- und Ausgänge

Dieses Beispiel zeigt den *indirekten Aufruf* des FB Zeit1, dessen Instanzname dem Funktionsbaustein UnterFB in Bsp. 2.22 als Ein-/Ausgangsvariable übergeben wurde. Zur Übersichtlichkeit wurden die FB-Variablen in Bsp. 2.23 mit englischen Bezeichnungen versehen, um Verwechslungen zu Namen in Bsp. 2.22 zu vermeiden. Dem Funktionsbaustein UnterFB wird der FB-Instanzname Zeit1 erst zur Laufzeit von HauptFB übergeben. In UnterFB wird Zeit1 (als Eingangsvariable Timer) mit Parametern versorgt (Timer.IN) und anschließend aufgerufen.

Wie an Bsp. 2.23 dargestellt, ist es ebenfalls möglich, auf die Ein- und Ausgänge des als Instanzname übergebenen FBs zuzugreifen. Dabei können Instanz-Eingänge (Timer.IN) lesend und schreibend, -Ausgänge (Timer.Q) nur lesend verarbeitet werden.

Die FB-Instanz Zeit3 dient in diesem Beispiel zum Vergleich der Behandlung von Eingangsparametern und Rückgabewerten eines FB als Ausgangsvariable.

FB-Instanznamen als Aktualparameter von Funktionen.
Instanznamen (wie Zeit1) bzw. Komponenten einer FB-Instanz (wie Zeit2.Q) können auch als Aktualparameter für Funktionen verwendet werden. Diese Tatsache klingt zunächst nach einem Widerspruch zur Forderung, daß Funktionen bei denselben Eingangsvariablen auch dasselbe Ergebnis liefern müssen und keine FBs aufrufen können.

Dieser Widerspruch besteht nur scheinbar: die so übergebene FB-Instanz darf nicht **aufgerufen** werden, doch ihre Ein- und Ausgangsvariablen werden wie die Elemente einer gewöhnlichen Datenstruktur behandelt, siehe Abschn. 2.4.1.

Funktionswerte als Aktualparameter.
Funktionen bzw. ihre Funktionswerte können ebenfalls als Aktualparameter für Funktionen oder Funktionsbausteine verwendet werden. Die Eingangsvariablen besitzen denselben Datentyp wie die Funktion und bekommen den Funktionswert beim Aufruf zugewiesen.

Über diese Möglichkeit macht die IEC 61131-3 allerdings keine expliziten Angaben, die Realisierung ist daher implementierungsabhängig.

2.8 POE-Merkmalsübersicht

Mit der folgenden Tabelle werden die in diesem Kapitel vorgestellten wesentlichen Eigenschaften der POEs im Überblick zusammengefaßt.

Konzept	Funktion	Funktionsbaustein	Programm
Eingangsparameter	ja	ja	ja
Ausgangsparameter	nein	ja	ja
Ein-/Ausgangsparameter	nein	ja	ja
Funktionswert	ja	nein	nein
Aufruf von Funktionen	ja	ja	ja
Aufruf von Funktionsbausteinen	nein	ja	ja
Aufruf von Programmen	nein	nein	nein
Deklaration globaler Variablen	nein	nein	ja
Zugriff auf externe Variablen	nein	ja	ja
Deklaration Direkt dargestellter Variablen [a]	nein	nein	ja
Deklaration lokaler Variablen	ja	ja	ja
Deklaration einer FB-Instanz	nein	ja	ja
Überladbar, erweiterbar [b]	ja	nein	nein
Flankenerkennung verwendbar	nein	ja	ja
Verwendung von EN/ENO [c]	ja	nein	nein
Pufferung von lokalen und Ausgangs-Variablen	nein	ja	ja
Indirekter FB-Aufruf	nein	ja	ja
Verwendung von Funktionswerten als Eingangsparameter [d]	ja	ja	ja
Verwendung von FB-Instanzen als Eingangsparameter	ja	ja	ja
Rekursiver Aufruf	nein	nein	nein

a bei Funktionsbausteinen nur in VAR_EXTERNAL
b für Standard-Funktionen vorgesehen
c für Funktionsbausteine zukünftig vorgesehen
d nicht in AWL, sonst: implementierungsabhängig

Tab. 2.9. POE- Merkmalsübersicht als Zusammenfassung wichtiger Ergebnisse dieses Kapitels. Die Einträge „ja“ und „nein“ bedeuten „zulässig“ bzw. „unzulässig“ für den entsprechenden POE-Typ.

3 Variablen, Datentypen und gemeinsame Elemente

Inhalt: siehe Papierversion.

4 Die neuen Programmiersprachen der IEC 61131-3

Inhalt: siehe Papierversion.

5 Standardisierte SPS-Funktionalität

Die IEC standardisiert nicht nur die Syntax der Programmiersprachen, sondern geht noch einen Schritt weiter, um die Verwendung typischer SPS-Funktionalitäten wie Zeiten, Zähler oder spezielle Arithmetik-Operationen zu vereinheitlichen.

Dazu werden in der Norm SPS-typische Funktionen und -Funktionsbausteine vordefiniert, die in ihrem Verhalten exakt beschrieben sind. Diese Bausteine werden *Standard-Funktionen* und *Standard-Funktionsbausteine* genannt. Ihre Namen sind Reservierte Schlüsselworte.

Programmiersysteme bzw. Bausteinbibliotheken unterschiedlicher Hersteller müssen sich an diese Vorgaben halten, wenn ihre Funktionen und Funktionsbausteine den entsprechenden Namen tragen. Daneben ist es jederzeit möglich, zusätzliche SPS-Funktionalität anzubieten, die beispielsweise Hardware-Eigenschaften berücksichtigt oder sonstige Charakteristika eines SPS-Systems unterstützt.

Eine solche Festlegung auf einen eindeutigen SPS-Funktionsstandard bildet eine wichtige Voraussetzung für einheitliche, hersteller- und projektübergreifende Ausbildung, Programmierung und Dokumentation.

In diesem Kapitel wird eine Übersicht über die wichtigsten Standard-Funktionen und -Funktionsbausteine sowie die verwendeten Konzepte gegeben:

1) **Standard-Funktionen (Std.-FUN)**
   - Aufrufschnittstelle
   - Erweiterbarkeit
   - Überladen
   - Beispiele
2) **Standard-Funktionsbausteine (Std.-FB)**
   - Aufrufschnittstelle
   - Beispiele

Während Standard-Funktionen den grundlegenden Verknüpfungs-Operatoren (Addition, Schieben, Vergleich u.a.) in herkömmlichen SPS-Systemen entsprechen, übernehmen die Standard-Funktionsbausteine die zustandsbehaftete SPS-Funktionalität von Zeiten, Zählern, R/S-Gliedern und zur Flankenerkennung.

In den folgenden Abschnitten wird die Aufrufschnittstelle (Ein- und Ausgangsvariablen bzw. Funktionswert) für Std.-FUN und Std.-FB ausführlich dargestellt. Anschließend werden jeweils Beispiele zur praxisgerechten Verwendung erklärt.

In den Anhängen Anh. A und Anh. B wird die grafische Deklaration sämtlicher Standard-Funktionen und -Funktionsbausteinen einschließlich einer Beschreibung ihrer Funktionalität dargestellt.

Die prinzipielle Handhabung von Funktionen und Funktionsbausteinen wurde bereits in Kap. 2 beschrieben, Eigenschaften ihrer Formalparameter in Kap. 3.

5.1 Standard-Funktionen

Die IEC 61131-3 definiert folgende acht Gruppen von Standard-Funktionen:

1) Funktionen zur Typumwandlung (Konvertierung von Datentypen),
2) Numerische Funktionen,
3) Arithmetische Funktionen,
4) Bitfolge-Funktionen (Schiebe- und bitweise boolesche Funktionen),
5) Funktionen für Auswahl und Vergleich,
6) Funktionen für Zeichenfolgen (String-Operationen),
7) Sonderfunktionen für Datentyp Zeit,
8) Sonderfunktionen für Datentypen der Aufzählung.

Tab. 5.1 faßt sämtliche Standard-Funktionen der Norm in dieser Weise komprimiert zusammen. Die Sonderfunktionen für die Datentypen Zeit (ADD, SUB, MUL, DIV, CONCAT) und Aufzählung (SEL, MUX, EQ, NE) werden zusammen mit den übrigen Funktionen zu Arithmetik, Vergleich, Auswahl und Zeichenfolge aufgeführt.

Dabei werden Funktionsname und Datentyp des Funktionswerts sowie eine stichwortartige Kurzbeschreibung angegeben.

Zusätzlich können dieser Tabelle – in Verbindung mit Tab. 5.4 – die Namen der Eingangsvariablen sowie deren Datentypen entnommen werden.

Standard-Funktionen (mit Datentypen der Eingangs-variablen)	Datentyp Funktions-wert	Kurzbeschreibung	über-lad-bar	erwei ter-bar
Typumwandlung				
_TO_* (ANY)	ANY	Datentyp umwandeln	ja	nein
TRUNC (ANY_REAL)	ANY_INT	Ganzzahlig machen	ja	nein
BCD_TO_** (ANY_BIT)	ANY	Wandeln von BCD	ja	nein
***_TO_BCD** (ANY_INT)	ANY_BIT	Wandeln nach BCD	ja	nein
DATE_AND_TIME_TO_-TIME_OF_DAY (DT)	TOD	Wandeln nach Tageszeit	nein	nein
DATE_AND_TIME_TO_-DATE (DT)	DATE	Wandeln nach Datum	nein	nein
Numerik				
ABS (ANY_NUM)	ANY_NUM	Absolutbetrag	ja	nein
SQRT (ANY_REAL)	ANY_REAL	Wurzel (Basis 2)	ja	nein
LN (ANY_REAL)	ANY_REAL	natürl. Logarithmus	ja	nein
LOG (ANY_REAL)	ANY_REAL	Logarithmus zur Basis 10	ja	nein
EXP (ANY_REAL)	ANY_REAL	Exponentiation	ja	nein
SIN (ANY_REAL)	ANY_REAL	Sinusfunktion	ja	nein
COS (ANY_REAL)	ANY_REAL	Cosinusfunktion	ja	nein
TAN (ANY_REAL)	ANY_REAL	Tangensfunktion	ja	nein
ASIN (ANY_REAL)	ANY_REAL	Sinus-Umkehrfunktion	ja	nein
ACOS (ANY_REAL)	ANY_REAL	Cosinus-Umkehrfunktion	ja	nein
ATAN (ANY_REAL)	ANY_REAL	Tangens-Umkehrfunktion	ja	nein
Arithmetik (IN1,IN2)				
ADD {+} (ANY_NUM, ANY_NUM)	ANY_NUM	Addition	ja	ja
ADD {+} [a] (TIME, TIME)	TIME	Zeitaddition	ja	nein
ADD {+} [a] (TOD, TIME)	TOD	Zeitaddition	ja	nein
ADD {+} [a] (DT, TIME)	DT	Zeitaddition	ja	nein
MUL {*} (ANY_NUM, ANY_NUM)	ANY_NUM	Multiplikation	ja	ja
MUL {*} [a] (TIME,ANY_NUM)	TIME	Zeitmultiplikation	ja	nein
SUB {-} (ANY_NUM, ANY_NUM)	ANY_NUM	Subtraktion	ja	nein
SUB {-} [a] (TIME, TIME)	TIME	Zeitsubtraktion	ja	nein
SUB {-} [a] (DATE, DATE)	TIME	Zeitsubtraktion	ja	nein
SUB {-} [a] (TOD, TIME)	TOD	Zeitsubtraktion	ja	nein
SUB {-} [a] (TOD, TOD)	TIME	Zeitsubtraktion	ja	nein
SUB {-} [a] (DT, TIME)	DT	Zeitsubtraktion	ja	nein
SUB {-} [a] (DT, DT)	TIME	Zeitsubtraktion	ja	nein
DIV {/} (ANY_NUM, ANY_NUM)	ANY_NUM	Division	ja	nein
DIV {/} [a] (TIME,ANY_NUM)	TIME	Zeitdivision	ja	nein
MOD (ANY_NUM, ANY_NUM)	ANY_NUM	Rest-Bildung (Modulo)	ja	nein
EXPT {**} (ANY_NUM, ANY_NUM)	ANY_NUM	Exponent	ja	nein
MOVE {:=} (ANY_NUM, ANY_NUM)	ANY_NUM	Zuweisung	ja	nein

a Sonderfunktionen für Datentyp Zeit

Tab. 5.1. Übersicht über die Standard-Funktionen (wird fortgesetzt)

Standard-Funktionen (mit Datentypen der Eingangs-variablen)		Datentyp Funktions-wert	Kurzbeschreibung	über-lad-bar	erwei ter-bar
Schieben	***(IN1,N)***				
SHL	(ANY_BIT, N)	ANY_BIT	Schieben nach links	ja	nein
SHR	(ANY_BIT, N)	ANY_BIT	Schieben nach rechts	ja	nein
ROR	(ANY_BIT, N)	ANY_BIT	Rotieren nach rechts	ja	nein
ROL	(ANY_BIT, N)	ANY_BIT	Rotieren nach links	ja	nein
Bitweise	***(IN1,IN2)***				
AND {&}	(ANY_BIT,ANY_BIT)	ANY_BIT	Bitweise UND-verknüpf.	ja	ja
OR {>=1}	(ANY_BIT,ANY_BIT)	ANY_BIT	Bitweise ODER-verknüpf.	ja	ja
XOR{=2k+1}	(ANY_BIT,ANY_BIT)	ANY_BIT	Bitweise EXODER-verkn.	ja	ja
NOT	(ANY_BIT,ANY_BIT)	ANY_BIT	Bitweise negieren	ja	nein
Auswahl	***(IN1,IN2)***				
SEL	(G, ANY, ANY)	ANY	Binäre Auswahl (1 aus 2)	ja	nein
SEL [b]	(G, AUFZ, AUFZ)	AUFZ	Binäre Auswahl (1 aus 2)	nein	nein
MAX	(ANY, ANY)	ANY	Maximum	ja	ja
MIN	(ANY, ANY)	ANY	Minimum	ja	ja
LIMIT	(MN, ANY, MX)	ANY	Begrenzung	ja	nein
MUX	(K, ANY, ..., ANY)	ANY	Multiplexer (1 aus N)	ja	ja
MUX [b]	(K, AUFZ, ..., AUFZ)	AUFZ	Multiplexer (1 aus N)	nein	nein
Vergleich	***(IN1,IN2)***				
GT {>}	(ANY, ANY)	BOOL	auf größer als	ja	ja
GE {>=}	(ANY, ANY)	BOOL	auf größer gleich	ja	ja
EQ {=}	(ANY, ANY)	BOOL	auf gleich	ja	ja
EQ {=} [b]	(AUFZ, AUFZ)	BOOL	auf gleich	nein	nein
LT {<}	(ANY, ANY)	BOOL	auf kleiner als	ja	ja
LE {<=}	(ANY, ANY)	BOOL	auf kleiner gleich	ja	ja
NE {<>}	(ANY, ANY)	BOOL	auf ungleich	ja	nein
NE {<>} [b]	(AUFZ, AUFZ)	BOOL	auf ungleich	nein	nein
Zeichenfolge	***(IN1, IN2)***				
LEN	(STRING)	INT	Länge einer Zeichenfolge	nein	nein
LEFT	(STRING, L)	STRING	Zeichenfolge „links von“	ja	nein
RIGHT	(STRING, L)	STRING	Zeichenfolge „rechts von“	ja	nein
MID	(STRING, L, P)	STRING	Zeichenfolge „aus Mitte“	ja	nein
CONCAT	(STRING, STRING)	STRING	Aneinanderreihung	nein	ja
CONCAT [a]	(DATE, TOD)	DT	Zeit-Aneinanderreihung	nein	nein
INSERT	(STRING, STRING, P)	STRING	Einfügen	ja	ja
DELETE	(STRING, L, P)	STRING	Ausschneiden	ja	ja
REPLACE	(STRING, STRING, L, P)	STRING	Ersetzen	ja	ja
FIND	(STRING, STRING)	INT	Suche Position	ja	ja

a Sonderfunktionen für Datentyp Zeit
b Sonderfunktionen für Datentyp der Aufzählung

Tab. 5.1 (Fortsetzung)

Die Abkürzungen zu den Datentypen von Eingangsvariablen und Funktionswerten in Tab. 5.1 werden, bis auf die Allgemeinen Datentypen, in Tab. 5.4 zusammengestellt. Diese Abkürzungen entsprechen den Namen für Eingangsvariablen, die die IEC 61131-3 für die betreffenden Standard-Funktionen verwendet. AUFZ ist eine zusätzliche Abkürzung, die hier gewählt wurden, um Tab. 5.1 übersichtlich zu halten.

Eingänge	**Bedeutung**	**Datentyp**
N	Anzahl der zu schiebenden Bits (Number)	UINT
L	linke Position innerhalb Zeichenfolge (Left)	UINT
P	Position innerhalb Zeichenfolge (Position)	UINT
G	Auswahl aus 2 Eingängen (Gate)	BOOL
K	Auswahl aus n Eingängen	ANY_INT
MN	Minimalwert für LIMIT (MiNimum)	ANY
MX	Maximalwert für LIMIT (MaXimum)	ANY
AUFZ	Datentyp der Aufzählung	

Tab. 5.2. Abkürzungen und Bedeutung für Eingangsvariablen in Tab. 5.1

In der ersten Spalte von Tab. 5.1 werden linksbündig die Funktionsnamen angegeben. Sternchen (*) im Funktionsnamen der Gruppe „Typumwandlung“ stehen als Abkürzungen für (vgl. Anh. A):

* Datentyp des Eingangs (rechts in Spalte 1)
** Datentyp des Funktionswerts (Spalte 2)

Der Name der Eingangsvariable(n) einer Funktion wird, soweit für eine Gruppe einheitlich angebbar bzw. überhaupt vorhanden, in der kursiv geschriebenen Gruppen-Überschrift angegeben. Beispielsweise heißen die Eingangsvariablen für Arithmetik IN1, IN2 und bei Erweiterung IN3, IN4,.... Bei Funktionen mit nur einer Eingangsvariablen besitzt diese keinen Namen.

Falls eine Funktion mit mehreren Eingängen davon nur einen mit einem beliebigen Datentyp (überladbar) besitzt, ist sein Variablenname „IN“. Dies trifft zu für LIMIT, LEFT, RIGHT, MID und DELETE.

In der IEC 61131-3 bildet SEL eine Ausnahme dieser einheitlichen Benennung (Definitionsfehler?). Die Eingänge werden nämlich dort mit G, IN0 und IN1 (statt IN1, IN2) bezeichnet.

Einige Standard-Funktionen besitzen in ihrer grafischen Darstellung alternativ Funktionsnamen mit Sonderzeichen, die in Tab. 5.1 unmittelbar hinter dem

Funktionsnamen in geschweiften Klammern angegeben sind. Beispielsweise wird die Addition mit ADD (als Operator in AWL) oder „+“ (als grafisches Symbol in KOP/FBS oder innerhalb Ausdrücken in ST) aufgerufen.

5.1.1 Überladen und Erweitern

Rechts neben dem Funktionsnamen werden in Tab. 5.1 die Datentypen der Eingangsvariablen in Klammern angegeben. Dabei werden zur Erläuterung auch Allgemeine Datentypen verwendet, die mit Tab. 3.9 bereits vorgestellt wurden. Jede Funktion, deren Eingangsvariablen mit einem Allgemeinen Datentyp beschrieben wird, heißt *überladbar* und hat ein „ja“ in der entsprechenden Spalte von Tab. 5.1. Die Funktion ist nicht auf einen Typ von Eingangsvariablen beschränkt, sondern kann auf verschiedene Datentypen angewendet werden.

Der Datentyp des Funktionswerts (2. Spalte) entspricht in der Regel den Datentypen der Eingänge. Ausnahmen bilden Funktionen wie LEN, die als Eingabeparameter eine Zeichenfolge erwarten, als Rückgabewert jedoch einen INT-Wert liefern.

Kann eine Standard-Funktion eine variable Anzahl von Eingängen (2, 3, 4,...) besitzen, wird sie *erweiterbar* genannt. Solche Funktionen sind mit „ja“ in der entsprechenden Spalte von Tab. 5.1 gekennzeichnet.

Die Angabe von formalen Parametern beim Aufruf von erweiterbaren Funktionen entfällt. Der Aufruf erfolgt in textuellen Sprachen einfach mit durch Kommata getrennte Aktualparameter – in der grafischen Darstellung fehlen die Parameternamen innerhalb des Kästchens.

Diese Eigenschaften sind in der IEC 61131-3 ***nicht*** für benutzerdefinierte Funktionen vorgesehen, können allerdings als Ergänzung zur Norm auch auf solche Funktionen (und andere POE-Typen) ausgedehnt werden.

Erweiterbarkeit und Überladen bei Standard-Funktionen werden in den beiden nächsten Abschnitten anhand von Beispielen erläutert.

Überladen von Funktionen.
Überladene Funktionen können unter gleichem Namen für die Verarbeitung mehrerer Datentypen verwendet werden.

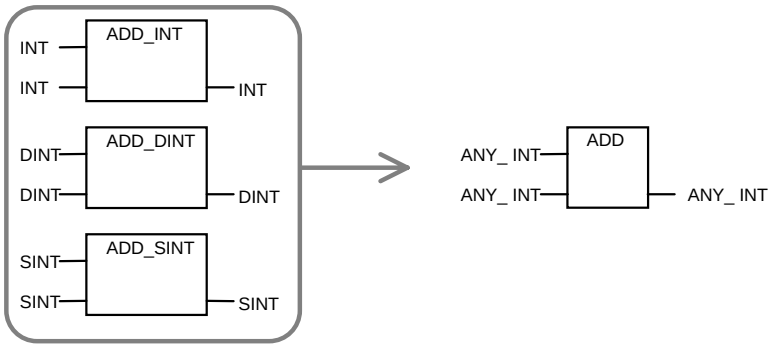
Dabei unterstützt eine überladene Funktion nicht immer *sämtliche* Datentypen eines Allgemeinen Datentyps, wie er in Kap. 3 erläutert wird.

Kennt ein SPS-System beispielsweise die Integer-Datentypen INT, DINT und SINT, werden für eine überladene Funktion ADD, die den allgemeinen Datentyp ANY_INT unterstützt, typischerweise genau diese drei Datentypen zugelassen.

Falls eine Standard-Funktion nicht überladen wird, sondern auf einen speziellen Elementaren Datentyp eingeschränkt wurde, ist ihr Name um einen Unterstrich und den betreffenden Datentyp zu erweitern: z.B. ADD_SINT ist eine auf den

Datentyp SINT eingeschränkte Addition. Solche Funktionen werden *typisiert* genannt. Demgegenüber heißen überladene Funktionen auch *typunabhängig*.

Dies wird in Bsp. 5.1 anhand der Integer-Addition veranschaulicht:



*Typisierte Standard-funktionen ADD_**

Überladene Standardfunktion ADD

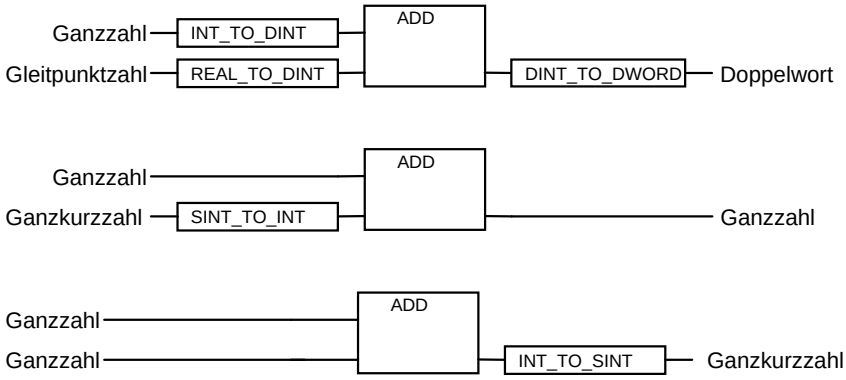
Bsp. 5.1. Typisierte Standard-Funktionen ADD_INT, ADD_DINT und ADD_SINT für ganzzahlige Additionen und die überladene Standard-Funktion ADD

Das Programmiersystem hat bei der Verwendung überladener Funktionen die passende typisierte Funktion auszuwählen. Wenn beispielsweise ein Funktionsaufruf der Addition ADD in Bsp. 5.1 erfolgt, bei dem die Datentypen der Aktualparameter vom Datentyp DINT sind, wird für den Anwender (nicht direkt sichtbar) die Variante ADD_DINT ausgewählt und aufgerufen.

Beim Aufruf von Standard-Funktionen müssen sämtliche überladenen Eingänge und ggf. auch der Funktionswert denselben Datentyp besitzen. D.h. es ist unzulässig, die Eingangsvariablen gleichzeitig mit Variablen unterschiedlicher Datentypen zu versorgen.

Bei gemischten Eingangs-Datentypen sind vom Anwender explizite Typumwandlungsfunktionen für die betroffenen Eingänge vorzuschalten bzw. dem Funktionswert nachzuschalten, wie es Bsp. 5.2 für ADD_DINT und ADD_INT zeigt. In solchen Fällen ist anstelle der überladbaren Funktion ADD ihre typisierte Variante (z.B. ADD_DINT) zu verwenden.

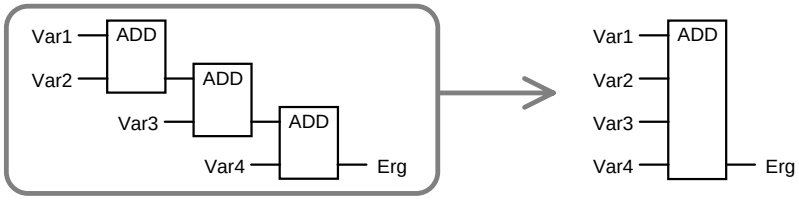
```
VAR
  Ganzzahl       :   INT;
  Ganzkurzzahl   :   SINT;
  Gleitpunktzahl :   REAL;
  Doppelwort     :   DWORD;
END_VAR
```



Bsp. 5.2. Aufruf der überladenen Standard-Funktion ADD mit Typumwandlungen zur korrekten Versorgung der Eingangsvariablen. Im oberen Fall wird ADD durch das Programmiersystem auf die typisierte Standard-Funktion ADD_DINT abgebildet, in den beiden unteren auf die Funktion ADD_INT.

Erweiterbarkeit von Funktionen.
Erweiterbare Standard-Funktionen können eine variable Anzahl von Eingängen besitzen, die zwischen zwei und einer vom SPS-System gegebenen Obergrenze liegt. Sie werden auch als „ausziehbar“ bezeichnet, um anzudeuten, daß ihre Kastenhöhe in grafischer Darstellung abhängig von der Eingangszahl ist.

Die Erweiterung der Eingänge einer Standard-Funktion um weitere Eingänge dient sowohl bei den textuellen als auch bei den grafischen Programmiersprachen der IEC 61131-3 als Ersatz für einen kaskadierten Aufruf dieser Funktion. Insbesondere bei den grafischen Sprachen KOP und FBS kann dadurch viel Platz in der Darstellung eingespart werden.



Kaskadierung von Addierern *erweiterter Addierer*

Anweisungsliste (AWL)

```
LD    Var1
ADD   Var2
ADD   Var3
ADD   Var4
ST    Erg
```

kann ersetzt werden durch:

```
LD    Var1
ADD   Var2, Var3, Var4
ST    Erg
```

Strukturierter Text (ST)

```
Erg := Var1 + Var2;
Erg := Erg  + Var3;
Erg := Erg  + Var4;
```

kann ersetzt werden durch:

```
Erg := Var1 + Var2 + Var3 + Var4;
```

Bsp. 5.3. Kaskadierte Addition als Ersatzdarstellung für eine erweiterbare Funktion am Beispiel der Addition in grafischer und textueller (AWL und ST) Darstellung

In diesem Beispiel wird der dreifache Aufruf der Standard-Funktion ADD durch einen einzigen Aufruf mit erweiterten Eingängen ersetzt. Auch für die textuellen Darstellungen in AWL und ST ergeben sich Vereinfachungen.

5.1.1 Beispiele

In diesem Abschnitt wird die Aufrufschnittstelle von Standard-Funktionen exemplarisch beschrieben. Auf Möglichkeiten und Besonderheiten beim Aufruf von Funktionen wurde bereits in Kap. 2 eingegangen.

Für jede der in Tab. 5.1 angegebenen Funktionsgruppen wird nachfolgend mindestens ein Beispiel herausgegriffen und erläutert. Dieses wird jeweils in den textuellen Sprachen AWL und ST sowie in den grafischen Darstellungen für KOP und FBS angegeben. In AWL und ST werden bei Aufrufen die Namen der Formalparameter nicht explizit mit angegeben.

Für die nachfolgenden Beispiele wird das PROGRAM ProgRahmenFUN in Bsp. 5.4 als Rahmen für den gemeinsamen Deklarationsteil der benötigten Variablen verwendet.

```
TYPE                                          (* Aufzählungstyp für Farbskala *)
  FARBEN   :   ( hRot, hGelb, hGruen,              (* hell *)
                 Rot, Gelb, Gruen,                 (* normal *)
                 dRot, dGelb, dGruen);             (* dunkel *)
END_TYPE

PROGRAM ProgRahmenFUN        (* gemeinsamer Deklarationsteil Std.-FUN *)
  VAR                                             (* lokale Daten *)
    Drehzahl     : REAL:= 10.5;                   (* Drehzahl *)
    DZ1          : REAL;                          (* Drehzahl 1 *)
    DZ2          : REAL := 46,8895504;            (* Drehzahl 2 *)
    Stufe        : UINT := 1;                     (* Drehzahl-Stufe *)
    Status       : BYTE     := 2#10101111;        (* Status *)
    Erg          : BYTE;                          (* Zwischenergebnis *)
    Maske        : BYTE     := 2#11110000;        (* Bit-Maske *)
    SPSNorm      : STRING [10]:= 'IEC 61131-5';   (* Zeichenfolge *)
    AT %IB2      : SINT;                          (* für MUX-Auswahl *)
    AT %QX3.0    : BOOL;                          (* Ausgangsbit  *)
    Zeitpunkt    : DT := dt#1994-12-23-01:02:03;  (* Datum mit Uhrzeit *)
    Uhrzeit      : TIME := t#04h57m57s;           (* Uhrzeit *)
    Ampel        : FARBEN;                        (* Ampelfarben *)
    Farbskala1   : FARBEN := hGelb;               (* Hellgelb Farbskala 1 *)
    Farbskala2   : FARBEN := Gelb;                (* Gelb Farbskala 2 *)
    Farbskala3   : FARBEN := dGelb;               (* Dunkelgelb Farbskala 3 *)
    Skala        : INT    := 2;                   (* Auswahl Farbskala *)
  END_VAR
  ...              (* Programmrumpf mit nachfolgenden Beispielen *)
END_PROGRAM
```

Bsp. 5.4. Gemeinsame Deklarationen für Beispiele zur Verwendung der Standard-Funktionen

Funktionen zur Typumwandlung.

Anweisungsliste (AWL)

```
LD          Drehzahl
REAL_TO_UINT
ST          Stufe
```

Strukturierter Text (ST)

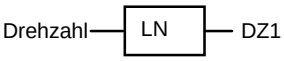
```
Stufe := REAL_TO_UINT (Drehzahl);
```

Bsp. 5.5. Beispiel für „Konvertierung REAL nach UINT“

Dieses Beispiel zeigt die Typumwandlung des REAL-Wertes Drehzahl (Gleitpunktzahl) in den ganzzahligen UINT-Wert Stufe.

Die Variable Stufe besitzt nach Ausführung der Funktion den Wert 10, indem von 10.5 abgerundet wird.

Numerische Funktionen.



Anweisungsliste (AWL)

```
LD      Drehzahl
LN
ST      DZ1
```

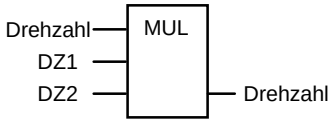
Strukturierter Text (ST)

```
DZ1 := LN    (Drehzahl);
```

Bsp. 5.6. Beispiel für „Natürlicher Logarithmus“

Dieses Beispiel zeigt die Berechnung des Natürlichen Logarithmus. Die Variable DZ1 besitzt nach der Ausführung den Wert 2,3513...

Arithmetische Funktionen.



Anweisungsliste (AWL)

```
LD      Drehzahl
MUL     DZ1
MUL     DZ2
ST      Drehzahl
```

Strukturierter Text (ST)

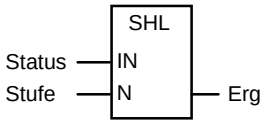
```
Drehzahl :=  Drehzahl * DZ1 * DZ2;
```

Bsp. 5.7. Beispiel für „Multiplikation“. Anstelle der zweifachen Verwendung von MUL kann auch geschrieben werden: MUL DZ1, DZ2.

Dieses Beispiel verwendet die überladene Funktion Multiplikation. Sie wird aufgrund der REAL-Eingangsvariablen auf die typisierte Funktion MUL_REAL abgebildet. Die Variable Drehzahl besitzt nach der Ausführung den Wert 1157,625.

Anstelle des Schlüsselworts MUL kann in der grafischen Darstellung auch das übliche Multiplikationszeichen „*“ verwendet werden, wie in diesem Beispiel für ST gezeigt.

Schiebe-Funktionen.



Anweisungsliste (AWL)

```
LD      Status
SHL     Stufe
ST      Erg
```

Strukturierter Text (ST)

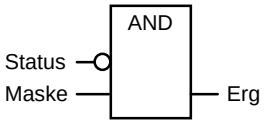
```
Erg :=  SHL ( IN    := Status,
              N     := Stufe );
```

Bsp. 5.8. Beispiel für „Schieben links“

In Bsp 5.8 wird die Schiebefunktion SHL verwendet, um den Wert der Variable Status um so viele Bitstellen nach links zu schieben, wie Stufe angibt.

Das Schiebe-Ergebnis Erg besitzt nach Funktions-Ausführung den Wert 2#01011110, d.h. beim Links-Schieben wird eine „0“ von rechts nachgeschoben.

Bitfolge-Funktionen.



Anweisungsliste (AWL)

```
LD      Status
NOT
AND     Maske
ST      Erg
```

Strukturierter Text (ST)

```
Erg := NOT Status  &  Maske;
```

Bsp. 5.9. Beispiel für „AND“

Da das logische AND eine erweiterbare Funktion darstellt, entfallen wie bei MUL die Angaben der Eingangsparameter-Namen. AND ist eine überladene Funktion, ihre Eingänge und ihr Funktionswert sind hier vom Typ BYTE, das Programmiersystem verwendet also die typisierte Funktion AND_BYTE.

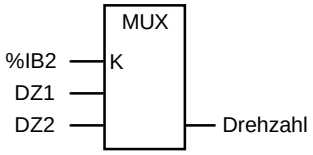
Anstelle der beiden AWL-Befehle LD und NOT könnte in Bsp. 5.9 gleichwertig auch der boolesche Operator LDN (Laden negiert) verwendet werden. Die Invertierung wird grafisch durch einen Funktionseingang mit kleinem Kreis dargestellt.

Anstelle des Schlüsselworts AND kann in der grafischen Darstellung auch das übliche Und-Zeichen „&“ verwendet werden, wie in diesem Beispiel für ST gezeigt.

In Bsp. 5.9 wird AND verwendet, um bestimmte Bits des Status-Werts mit Hilfe einer Bit-Maske zu extrahieren.

Das Ergebnis Erg besitzt nach Funktions-Ausführung den Wert 2#01010000, d.h. die unteren vier Bits wurden durch die Maske ausgeblendet.

Funktionen für Auswahl.



Anweisungsliste (AWL)

```
LD      %IB2
MUX     DZ1, DZ2
ST      Drehzahl
```

Strukturierter Text (ST)

```
Drehzahl := MUX ( K   := %IB2,
                  DZ1,
                  DZ2 );
```

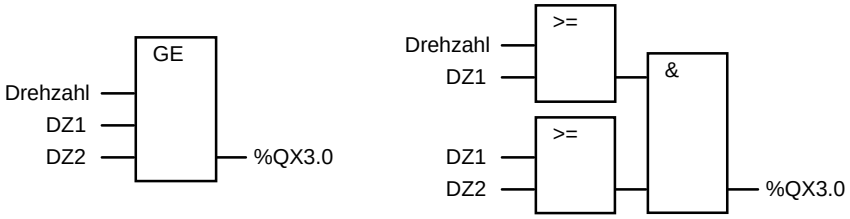
Bsp. 5.10. Beispiel für „MUX“

Der Multiplexer MUX besitzt neben dem Integer-Eingang K die überladenen Eingänge vom gleichen Typ wie der Funktionswert. Daher können bis auf K die Bezeichnungen für die Eingangsparameter entfallen. K wird in Bsp. 5.10 mit dem Datentyp SINT versorgt (ganzzahliges Byte mit Vorzeichen).

Falls das Eingangsbyte %IB2 den Wert „1“ besitzt, bekommt Drehzahl nach Ausführung den Wert von DZ2 zugewiesen, bei „0“ den Wert von DZ1.

Falls der Wert am Eingang K kleiner als 0 oder größer ist als die Anzahl der übrigen Eingänge, meldet das Programmiersystem bzw. Laufzeitsystem einen entsprechenden Fehler (vgl. Anh. E).

Funktionen für Vergleich.



Erweiterter Vergleicher *Ersatz-Darstellung*

Anweisungsliste (AWL)

```
LD    Drehzahl
GE    DZ1
GE    DZ2
ST    %QX3.0
```

Strukturierter Text (ST)

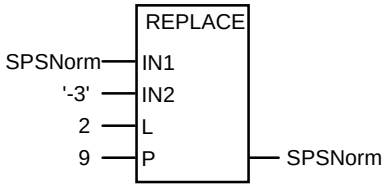
```
%QX3.0 := GE (  Drehzahl,
                DZ1,
                DZ2 );
```

Bsp. 5.11. Beispiel für „Erweiterten Vergleicher“. In grafischer Darstellung wird rechts eine Ersatz-Darstellung angegeben.

Vergleicher verwenden überladene Eingänge, ihr Ausgang Q ist boolesch. Sie bilden das „Bindeglied“ zwischen numerischen/arithmetischen Berechnungen und booleschen Verknüpfungen.

In der grafischen Darstellung des Bsp. 5.11 ist die um einen Eingang erweiterte Vergleichsfunktion zusätzlich als Ersatzbild angegeben. In dieser Ersatzdarstellung sind die Schlüsselworte GE und AND durch ihre gleichwertigen Kürzel (>= und &) ersetzt.

Funktionen für Zeichenfolgen.



Anweisungsliste (AWL)

```
LD        SPSNorm
REPLACE   '-3', 2, 9
ST        SPSNorm
```

Strukturierter Text (ST)

```
SPSNorm:= REPLACE (
              IN2  := '-3',
              IN1  := SPSNorm,
              P    := 9,
              L    := 2 );
```

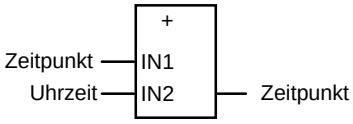
Bsp. 5.12. Beispiel für „REPLACE“ in AWL und in ST

Die Funktion REPLACE bietet keine überladbaren Eingänge, daher wird jeder Eingangsparameter beim Aufruf in grafischer Darstellung (und in ST) mit seinem Variablennamen angegeben.

Bsp. 5.12 zeigt, daß die Reihenfolge dieser Eingänge dadurch beliebig sein kann (vgl. ST-Bsp). Die Reihenfolge ist fest vorgeschrieben (vgl. AWL-Bsp.), wenn keine Namen der Eingangsparameter verwendet werden.

Die Zeichenfolge SPSNorm besitzt nach Ausführung den STRING-Wert 'IEC 61131-3'.

Funktionen für Datentyp Zeit.



Anweisungsliste (AWL)

```
LD    Zeitpunkt
ADD   Uhrzeit
ST    Zeitpunkt
```

Strukturierter Text (ST)

```
Zeitpunkt := Zeitpunkt + Uhrzeit;
```

Bsp. 5.13. Beispiel für „ADD Zeit“ in AWL und in ST

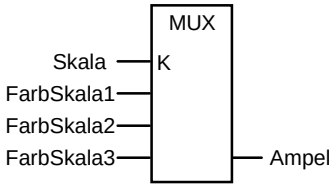
Diese Zeit-Addition (ebenso wie die entsprechende Subtraktion) kann als Fortsetzung der überladenen Addition betrachtet werden — bezogen auf gemischte Argumente: TIME, TIME_OF_DAY (TOD) und DATE_AND_TIME (DT).

Die Variable Zeitpunkt besitzt nach der Funktionsausführung den Wert DT#1994-12-23-06:00:00.

Addition und Subtraktion von Zeiten sind nicht symmetrisch. Für die Subtraktion kommen gegenüber der Addition die drei Differenzen für die Eingangs-Datentypen DATE, TOD und DT hinzu. Diese Operationen gibt es bei der Addition nicht, da es wenig Sinn macht, beispielsweise den 10. Oktober auf den 12. September zu addieren.

Für die Addition fehlt die Möglichkeit, auf DATE eine Zeit TIME addieren zu können, während dies für TIME, TOD und DT möglich ist. Um es auch für DATE zu ermöglichen, wird zunächst nach DT gewandelt und anschließend addiert. Dadurch können Fehlprogrammierungen bei der Zeitrechnung weitgehend vermieden werden.

Funktionen für Aufzählungstypen.



Anweisungsliste (AWL)

```
LD      Skala
MUX     (   FarbSkala1,
            FarbSkala2,
            FarbSkala3)
ST      Ampel
```

Strukturierter Text (ST)

```
Ampel := MUX (    K   := Skala,
                  FarbSkala1,
                  FarbSkala2,
                  FarbSkala3);
```

Bsp. 5.14. Beispiel für „Aufzählungs-MUX“

Für den Datentyp Aufzählung bietet die IEC 61131-3 Funktionen, von denen in Bsp 5.14 die Auswahl (MUX) gezeigt wird.

Mit Hilfe der INT-Variablen Skala wird eine Variable mit Datentyp Aufzählungselement (Typdeklaration FARBEN) ausgewählt.

Nach Ausführung der MUX-Funktion erhält die Variable Ampel die Werte „hGelb“, „Gelb“ und „dGelb“ der Farbskalen (hell, normal und dunkel), wenn Skala die Werte 0, 1 und 2 durchläuft.

5.2 Standard-Funktionsbausteine

Die IEC 61131-3 definiert eine Reihe von Standard-Funktionsbausteinen, die dem SPS-Programmierer die wichtigsten SPS Funktionsblöcke (mit speicherndem Verhalten) zur Verfügung stellen.

Die IEC 61131-3 definiert folgende fünf Gruppen von Standard-FBs:

1) Bistabile Elemente (= Flip-Flops, R/S-Glieder),
2) Flankenerkennung,
3) Zähler,
4) Zeiten,
5) Funktionsbausteine für Kommunikation.

Tab. 5.3 faßt sämtliche Standard-FBs der Norm in dieser Weise komprimiert zusammen. Der Aufbau dieser Tabelle entspricht prinzipiell dem der Standard-Funktionen (vgl. Tab. 5.1). FBs für Kommunikation werden im Teil 5 der IEC 61131 definiert und in diesem Buch nicht behandelt.

Für Ein- und Ausgangsvariablen werden anstelle ihrer Datentypen ihre Namen aufgeführt, die in Tab. 5.2 mit Angabe des dazugehörigen Elementaren Datentyps zu finden sind.

Name des Std.-FB mit Namen der Eingangsparameter		**Namen der Rückgabewerte**	**Kurzbeschreibung**
R/S-Glieder			
SR	(S1,R,	Q1)	Setzen vorrangig
RS	(S,R1,	Q1)	Rücksetzen vorrangig
Flanken			
R_TRIG {->}	(CLK,	Q)	Erkennung steigende Flanke
F_TRIG {-<}	(CLK,	Q)	Erkennung fallende Flanke
Zähler			
CTU	(CU,R,PV,	Q,CV)	Vorwärts-Zähler
CTD	(CD,LD,PV,	Q,CV)	Rückwärts-Zähler
CTUD	(CU,CD,R,LD,PV,	QU,QD,CV)	Vorwärts/Rückwärts-Zähler
Zeiten			
TP	(IN,PT,	Q,ET)	Impulsgeber
TON {T---0}	(IN,PT,	Q,ET)	Einschalt-Verzögerung
TOF {0---T}	(IN,PT,	Q,ET)	Ausschalt-Verzögerung
RTC	(EN,PDT,	Q,CDT)	Echtzeit-Uhr
Kommunikation			***siehe IEC 61131-5***

Tab. 5.3. Übersicht der Standard-Funktionsbausteine

Eingänge / Ausgänge	Bedeutung	Datentyp
R	Rücksetz-Eingang (Reset)	BOOL
S	Setz-Eingang (Set)	BOOL
R1	R vorrangig	BOOL
S1	S vorrangig	BOOL
Q	Ausgang	BOOL
Q1	Ausgang (RS, SR)[1]	BOOL
CLK	(Takt-) Eingang (CLocK)	BOOL
CU	Eingang Zähle vorwärts (Count Up),	R_EDGE
CD	Eingang Zähle abwärts (Count Down)	R_EDGE
LD	Lade Zählwert (LoaD)	INT
PV	Zählwert (Preset Value)	INT
QD	Abwärts-Ausgang (Down)	BOOL
QU	Aufwärts-Ausgang (Up)	BOOL
CV	Aktueller Zählwert (Current Value)	INT
IN	Zeit-Eingang (INput)	BOOL
PT	Zeitwert (Preset Time)	TIME
ET	Aktueller Zeitwert (End Time)	TIME
PDT	Datum/Zeitwert (Preset Date and Time)	DT
CDT	Aktuelles Datum mit Zeit (Current Date and Time)	DT

Tab. 5.4. Abkürzungen und Bedeutung für Ein- und Ausgangsvariablen in Tab. 5.3

Die Zähler-Eingänge CU und CD sind vom Datentyp BOOL mit dem Zusatz R_EDGE, d.h. an ihnen muß eine steigende Flanke erkannt werden, um vorwärts oder rückwärts zu zählen.

Die Standard-Rückgabewerte sämtlicher Standard-FBs sind Null, wenn der Funktionsbaustein zum ersten Mal aufgerufen wird. Lediglich die Echtzeit-Uhr liefert an ihrem Ausgang CDT (Current Date and Time) sofort das aktuelle Datum mit Zeit.

Die Namen der Eingangsparameter jedes Standard-FB sind Schlüsselworte. In AWL können sie als Operatoren auf FB-Instanzen verwendet werden, wie es in Abschn. 4.1.4 beschrieben wurde.

Die Eingangsparameter R und S werden in AWL allerdings mit einer zweiten Bedeutung versehen, indem sie Operatoren zum Rücksetzen und Setzen von allgemeinen booleschen Variablen darstellen. Diese Tatsache bedeutet eine Schwierigkeit, die bei Implementierung von Programmiersystemen zu überwinden ist.

[1] In der Norm wird der Q-Ausgang von Flip Flops im Unterschied zu den übrigen Standard-Funktionsbausteinen mit Q1 statt mit Q bezeichnet.

5.2.2 Beispiele

Wie für die Standard-Funktionen wird in diesem Abschnitt die Aufrufschnittstelle von Standard-Funktionsbausteinen exemplarisch beschrieben. Auf Möglichkeiten und Besonderheiten beim Aufruf von FBs wurde bereits in Kap. 2 eingegangen.

Für jede der in Tab. 5.3 angegebenen Funktionsgruppen wird nachfolgend mindestens ein Beispiel herausgegriffen und erläutert. Dieses wird jeweils in den textuellen Sprachen AWL und ST sowie in den grafischen Darstellungen für FBS angegeben.

In AWL und ST werden im folgenden die Namen der FB-Eingangsparameter explizit angegeben, um die Benutzung der FB-Instanzen möglichst gut zu verdeutlichen.

Für AWL wird dabei diejenige Variante des Funktionsaufrufs (vgl. Abschn. 4.1.4) verwendet, die Eingangsparameter und Rückgabewerte als Strukturelemente der FB-Instanz behandelt.

Für die nachfolgenden Beispiele wird das PROGRAM ProgRahmenFB in Bsp. 5.15 als Rahmen für den gemeinsamen Deklarationsteil der benötigten Variablen und FB-Instanznamen angenommen.

```
PROGRAM ProgRahmenFB                    (* Gemeinsamer Deklarationsteil Std.-FB *)
  VAR_GLOBAL RETAIN                     (* Globale, batteriegepufferte Daten *)
    EchtZeit      :  RTC;               (* Echtzeit-Uhr *)
    Zeitdauer     :  TIME := t#63ms;    (* 63 Millisekunden Initialwert *)
  END_VAR
```

Bsp. 5.15. Gemeinsame Deklarationen für Beispiele zur Verwendung der Standard-FBs (wird fortgesetzt)

```
  VAR                                          (* Lokale Daten *)
    M_Schalt        :  RS;                     (* Merker Schaltbedingung *)
    TippTaster      :  R_TRIG;                 (* Flankenerkennung Taster *)
    VR_Zaehler      :  CTUD;                   (* Vorwärts/Rückwärts-Zähler *)
    V_Impuls        :  TP;                     (* Verlängerter Impuls als Zeit *)
    Impuls          :  BOOL;                   (* Impulsmerker *)
    NotAus          :  BOOL;                   (* Merker NotAus *)
    AT %IX1.4       :  BOOL;                   (* Not-Aus-Taster *)
    AT %IX2.0       :  BOOL;                   (* Zähle hoch *)
    AT %IX2.1       :  BOOL;                   (* Lade Zaehler *)
    AT %IX2.2       :  BOOL;                   (* Starte Zeit *)
    AT %IX3.0       :  BOOL;                   (* Zähle runter *)
    AT %IW5         :  INT;                    (* Zählgrenze *)
    AT %MX3.2       :  BOOL;                   (* Schalt-Merker *)
    AT %QX3.2       :  BOOL;                   (* Schalt-Ausgang *)
    MaxErreicht     :  BOOL;                   (* Zähler oben *)
    MinErreicht     :  BOOL;                   (* Zähler unten *)
    Zaehlwert  AT %MW2 : INT;                  (* Aktueller Wert des Zählers *)
    Zeitwert        :  TIME;                   (* Zeitwert *)
    DatumUndZeit    :  DT;                     (* Aktuelles Datum mit Uhrzeit *)
  END_VAR
  ...               (* Programmrumpf für nachfolgende Beispiele *)
END_PROGRAM
```

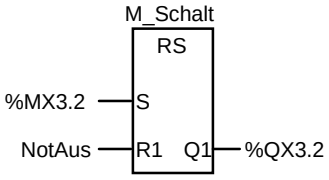
Bsp. 5.15. (Fortsetzung)

Diese Deklarationen beinhalten:

- FB-Instanznamen (M_Schalt bis V_Impuls),
- Direkt dargestellte Variable (%IX1.4 bis %QX3.2),
- Symbolische Variable (Zaehlwert),
- Allgemeine Variable (sonst).

Im Abschnitt VAR werden sie als lokale Variablen deklariert und im Abschnitt VAR_GLOBAL RETAIN als batteriegepufferte globale Variable.

Bistabile Elemente (Flip-Flops).



Anweisungsliste (AWL)

```
LD      %MX3.2
ST      M_Schalt.S
LD      NotAus
ST      M_Schalt.R1
CAL     M_Schalt
LD      M_Schalt.Q1
ST      %QX3.2
```

Strukturierter Text (ST)

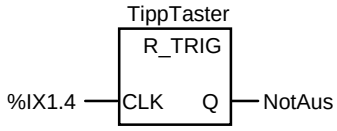
```
CAL M_Schalt ( S  := %MX3.2,
               R1 := NotAus);
```

Bsp. 5.16. Beispiel für ein „Bistabiles Element“ Flip-Flop

Bsp. 5.16 zeigt die Benutzung eines Flip-Flops zur Speicherung einer binären Zustandsinformation, hier den Wert des Merkers %MX3.2.

Mit Hilfe des Eingangs R1 kann der Ausgang Q1 vorrangig rückgesetzt werden, d.h. auch wenn an beiden Eingängen eine „1“ ansteht, bleibt der Ausgang auf „0“.

Flankenerkennung.



Anweisungsliste (AWL)

```
LD    %IX1.4
ST    TippTaster.CLK
CAL   TippTaster
LD    TippTaster.Q
ST    NotAus
```

Strukturierter Text (ST)

```
CAL TippTaster ( CLK := %IX1.4);

NotAus := TippTaster.Q;
```

Bsp. 5.17. Beispiel zur Flankenerkennung R_TRIG

Der FB TippTaster in Bsp. 5.17 wertet das am Peripherie-Bit anliegende Signal bei steigender Flanke aus und liefert bei einem 0→1-Übergang an Q eine „1“. Dazu führt der FB R_TRIG einen internen Flankenerkennungsmerker mit, der den „alten“ Wert von CLK speichert, um ihn jeweils mit dem aktuellen vergleichen zu können.

Diese Information wird einen Programmzyklus lang (bis zum erneuten Aufruf) gemerkt und kann dadurch von übrigen Programmteilen ausgewertet werden, auch wenn %IX1.4 bereits wieder auf „0“ ist. Beim nächsten Aufruf im folgenden Zyklus wird der Flanken-Merker TippTaster wieder rückgesetzt.

Das bedeutet, daß der FB R_TRIG für Direkt dargestellte Variablen nur solche Flanken erkennen kann, die in Abständen mindestens größer als die Programmzykluszeit auftreten.

Die IEC 61131-3 bietet die FBs R_TRIG und F_TRIG nicht nur zur unmittelbaren, wie in Bsp. 5.17 gezeigten Verwendung an. Daneben werden diese FBs zur Flankenerkennung implizit zur Realisierung der Variablen-Attribute R_EDGE und F_EDGE verwendet (vgl. Kap. 3).

Bsp. 5.18 zeigt die Variablendeklaration mit der Angabe eines flankengesteuerten Eingangs (fett dargestellt) im Deklarationsteil des FB BspFlanke.

```
FUNCTION_BLOCK  BspFlanke
VAR_INPUT
  Flanke      :   BOOL  R_EDGE;          (* flankengesteuert *)
END_VAR
VAR_OUTPUT
  Merker      :   BOOL;
END_VAR
...
LD      Flanke;                    (* Zugriff auf Flanken-Merker *)
ST      Merker;
...
END_FUNCTION_BLOCK
```

Bsp. 5.18. Deklaration mit R_EDGE zur Flankenerkennung und Verwendung in AWL

Zum besseren Verständnis der flankengesteuerten Variablen erhält das Bsp. 5.18 in Bsp. 5.19 zusätzliche Anweisungen, die diese Erkennung realisieren. Eine entsprechende Funktionalität wird bei der Verwendung jener Variablen – für den Anwender unsichtbar – vom Programmiersystem eingesetzt.

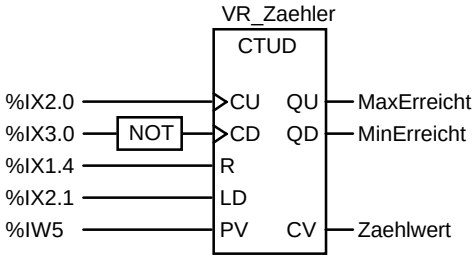
```
FUNCTION_BLOCK  BspFlanke
VAR_INPUT
  Flanke      :   BOOL;              (* flankengesteuert *)
END_VAR
VAR_OUTPUT
  Merker      :   BOOL;
END_VAR
VAR
   FlErkenn   :   R_TRIG;            (* FB-Instanz „steigende Flanke“ *)
END_VAR
...
CAL   FlErkenn (CLK := Flanke);      (* FB-Aufruf zur Flankenerkennung *)
LD      FlErkenn.Q;                  (* FB-Ergebnis laden *)
ST      Merker;
...
END_FUNCTION_BLOCK
```

Bsp. 5.19. Automatische Umsetzung des Beispiels aus Bsp. 5.18 mit R_TRIG

Die Deklaration des FB FlErkenn in Bsp. 5.19 wird implizit vom Programmiersystem in das Programm eingefügt. Dieser FB wird mit der Eingangsvariable Flanke aufgerufen. Sein Ausgangswert FlErkenn.Q wird dann überall dort eingesetzt, wo auf den Wert von Flanke zugegriffen wird.

Dieses Beispiel zeigt, warum diese Art der Flankenerkennung in der IEC 61131-3 nicht für Ausgangsvariablen zulässig ist: solche könnten an beliebigen Stellen der POE überschrieben werden. Dadurch würde jedoch die Regel verletzt werden, daß Funktionsbausteine nicht die Ausgänge aufgerufener anderer FBs ändern dürfen! Siehe auch Abschn. 2.3.2.

Vorwärts/Rückwärts-Zähler.



Anweisungsliste (AWL)

```
LD      %IX2.0
ST      VR_Zaehler.CU
LDN     %IX3.0
ST      VR_Zaehler.CD
LD      %IX1.4
ST      VR_Zaehler.R
LD      %IX2.1
ST      VR_Zaehler.LD
LD      %IW5
ST      VR_Zaehler.PV
CAL     VR_Zaehler

LD      VR_Zaehler.QU
ST      MaxErreicht
LD      VR_Zaehler.QD
ST      MinErreicht
LD      VR_Zaehler.CV
ST      Zählwert
```

Strukturierter Text (ST)

```
CAL VR_Zaehler ( CU := %IX2.0
                 CD := NOT(%IX3.0),
                 R  := %IX1.4,
                 LD := %IX2.1,
                 PV := %IW5);

MaxErreicht := VR_Zaehler.QU;
MinErreicht := VR_Zaehler.QD;
Zaehlwert := VR_Zaehler.CV;
```

Bsp. 5.20. Beispiel für den Vorwärts/Rückwärts-Zähler CTUD

In diesem Beispiel wird der Vorwärts/Rückwärts-Zähler VR_Zaehler an sämtlichen Eingängen beschaltet, was nicht der Regelfall sein muß. Üblicherweise werden nur die benötigten Eingangs- und Ausgangsvariablen beschaltet.

Die Eingänge CU und CD können beide gleichzeitig mit einer steigenden Flanke aktiv sein. In diesem Fall würde der Zählstand gleich bleiben, wenn das Minimum oder das Maximum noch nicht erreicht wurde.

Der VR_Zaehler in Bsp. 5.20 zählt mit steigender Flanke von %IX2.0 vorwärts und bei fallender Flanke %IX3.0 rückwärts. Werden CU und CD mit derselben Variablen oder Konstanten belegt, würde der Zähler im Takt zu diesem Signal abwechselnd um eins hoch und gleich darauf wieder herunter zählen.

Der an PV anliegende Setzwert wird bei einem Aufruf von der Peripherie %IW5 nachgeladen, wenn gleichzeitig der Lade-Eingang LD aktiv ist. Hier braucht keine steigende Flanke anliegen.

Zeitgeber (Zeiten).
Bsp. 5.21 zeigt stellvertretend die Benutzung der Zeitgeber-FBs. Dieses Beispiel demonstriert deutlich, wie die Zeitgeber als Instanzen den Zustand ihrer Daten, insbesondere auch den der Eingangsparameter, zwischen den einzelnen Aufrufen beibehalten.

Prinzipiell kann bei jedem Aufruf eines Zeitgebers (bzw. eines beliebigen FBs) jede einzelne Eingangsvariable unmittelbar vorher gesetzt werden.

Eine solche Umparametrierung des Zeitgebers zur Laufzeit führt jedoch dazu, daß dieser innerhalb derselben POE zur Steuerung mehrerer Prozeß-Zeiten gleichzeitig verwendet wird. Dadurch wird das Programm schwer lesbar und fehleranfällig, so daß solche Fälle in der Programmierpraxis wohl selten vorkommen.

Es reicht, den Zeitwert PT pro Instanz des Zeitgeber beim ersten Aufruf nur einmal zu setzen und von da an immer wieder zu benutzen. Dies bedeutet, daß der Aufruf der Zeit primär dem *Starten* der Zeitfunktion über den Eingang IN dient.

Die Ausgangsvariablen des Zeitgebers können an jeder beliebigen Programmstelle abgefragt werden, brauchen also nicht unmittelbar nach Aufruf ausgewertet zu werden.

Allerdings werden die Ausgangsparameter bei jedem Aufruf des Zeit-FBs gesetzt, indem sie aufgrund der Werte des physikalisch im Hintergrund laufenden Timers aktualisiert werden. Dadurch „veraltet“ der Zeitwert zwischen zwei Aufrufen. Deshalb sollte der Zeit-FB in einer periodischen Task häufig genug und nicht zu lang vor der Auswertung von Q und ET aufgerufen werden, um die Zeit nicht zu verfälschen.

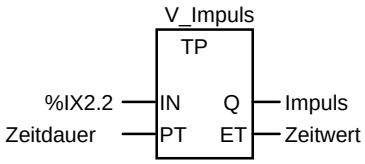
Während der Ausgang Q dazu dient, festzustellen, ob die Zeit abgelaufen ist oder nicht, wird der Ausgang ET verwendet, um die bis dahin noch verbleibende Zeit anzuzeigen.

Daher werden Zeiten gewöhnlich nach folgendem Muster aufgerufen:

1) Setzen des Zeitwerts,
2) periodisches Starten mit Aktualisieren,
3) Abfrage der Zeit.

Oft werden in einem periodisch laufenden Programm diese drei Schritte in einem Aufruf vereint. Dies vereinfacht das Programm und erleichtert die grafische Darstellung des Aufrufs.

Das Zeitverhalten der Zeitgeber-Typen kann detailliert Anh. B entnommen werden.



Anweisungsliste (AWL)

```
        (* 1. Impulsdauer setzen *)
LD      Zeitdauer
ST      V_Impuls.PT
...
        (* 2. Zeit starten *)
LD      %IX2.2
ST      V_Impuls.IN
CAL     V_Impuls
...
        (* 3. Zeit abfragen *)
LD      V_Impuls.Q
ST      Impuls
LD      V_Impuls.ET
ST      Zeitwert
```

Strukturierter Text (ST)

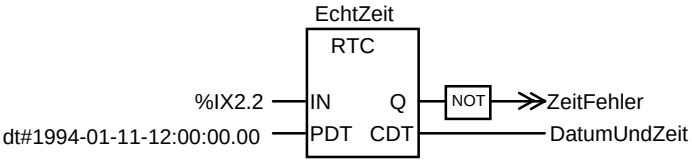
```
        (* 1. Impulsdauer setzen *)
V_Impuls.PT := Zeitdauer;

...


        (* 2. Zeit starten *)
CAL  V_Impuls  ( IN  := %IX2.2);
...
        (* 3. Zeit abfragen *)

Impuls := V_Impuls.Q;

Zeitwert := V_Impuls.ET;
```

Bsp. 5.21. Beispiel zur Zeit: Impulsgeber TP

Bsp. 5.21 zeigt die Verwendung des Impulsgebers mit Hilfe der beschriebenen drei Schritte zum Aufruf der Instanz V_Impuls:

1) Der Zeitwert für V_Impuls wird auf 63 Millisekunden festgelegt,
2) V_Impuls wird gestartet durch Eingangsbit 2.2,
3) Die Abfrage von V_Impuls mit Q und ET.

Bsp. 5.22 zeigt den Zeitgeber-FB RTC (Echtzeit-Uhr).



Anweisungsliste (AWL)

```
          (* 1. Datum setzen *)
LD        dt#1994-01-11-12:00:00.00
ST        EchtZeit.PDT
...
          (* 2. Uhr starten *)
LD        %IX2.2
ST        EchtZeit.IN
CAL       EchtZeit
...
          (* 3. Zeit abfragen *)
LD        EchtZeit.Q
JMPCN     ZeitFehler
LD        EchtZeit.CDT
ST        DatumUndZeit
JMP       IrgendwoHin
Zeitfehler:
...       ...
```

Strukturierter Text (ST)

```
          (* 1. Datum setzen *)
EchtZeit.PDT := dt#1994-01-11-12:00;

...

          (* 2. Uhr starten *)
CAL  EchtZeit      ( IN  := %IX2.2);
...
          (* 3. Zeit abfragen *)
IF EchtZeit.Q THEN
   DatumUndZeit := EchtZeit.CDT;
ELSE
   ...       (* Zeitfehler *)
ENDIF
```

Bsp. 5.22. Beispiel zur Zeit: Echtzeit-Uhr RTC

Bei einer steigenden Flanke an %IX2.2 wird die Datumskonstante NeuesDatum am Eingang PDT übernommen. Solange IN auf 1 gesetzt bleibt, liegt am Ausgang CDT das laufende Datum mit Uhrzeit an.

Der Ausgang Q wird dazu verwendet, um festzustellen, ob das laufende Datum gültig ist (Q ist Kopie von IN). Falls das Datum ungültig ist, wird die Fehlerbehandlung Zeitfehler angesprungen.

6 Moderne SPS-Konfiguration

Die IEC 61131-3 trägt der technologischen Entwicklung der letzten Jahre Rechnung, indem mit modernen Konzepten die Modellierung von SPS-Projekten über die Grenzen von Ein-Prozessor-Programmen hinaus möglich wird.

Das Softwaremodell der IEC 61131-3 erlaubt die praxisgerechte Strukturierung der Anwendung in Programmteile (Modularisierung in POEs). Dadurch werden Wartbarkeit, Dokumentation oder Diagnose-Möglichkeiten der SPS-Programme wesentlich unterstützt.

Durch eine einheitliche Softwarearchitektur werden wesentliche Voraussetzungen für die Portabilität von Programmen erfüllt, indem die Verarbeitungseinheiten der SPS (Ressourcen) eindeutig beschriebene Eigenschaften besitzen und so eine Plattform für hardwareunabhängigere Programme bieten.

Die bisher übliche Baustein-Strukturierung von SPS-Projekten (vgl. Abb. 2.5) diente einerseits der Programmstrukturierung und andererseits beinhalteten einige besondere Bausteintypen (wie Organisations-Bausteine) oft implizite Laufzeit-Eigenschaften. Demgegenüber bietet die IEC 61131-3 wesentlich erweiterte und standardisierte Möglichkeiten.

In diesem Kapitel werden die *Konfigurationselemente* der IEC 61131-3 erläutert, die ein wesentliches Strukturierungshilfsmittel für das Zusammenwirken der POEs darstellen. Sie definieren Laufzeit-Eigenschaften der Programme, Kommunikationsbeziehungen sowie die Zuordnung zur SPS-Hardware.

Diese Konfigurationselemente der IEC 61131-3 unterstützen heutige moderne Betriebssysteme auf der SPS-Seite. Eine CPU kann beispielsweise mehrere Programme gleichzeitig abarbeiten (Multi-Tasking).

6.1 Projekt-Strukturierung durch Konfigurationselemente

Nachdem in den vorangegangenen Kapiteln Programmierung und Verwendung der POEs erläutert wurden, dient dieser Abschnitt dazu, einen Überblick über die Modellbildung und Strukturierung von SPS-Programmen auf einer übergeordneten Ebene zu gewinnen.

Dazu wird die Abb. 2.7 (POE-Aufruf-Hierarchie) in die Gesamtsicht eines SPS-Programms einbezogen.

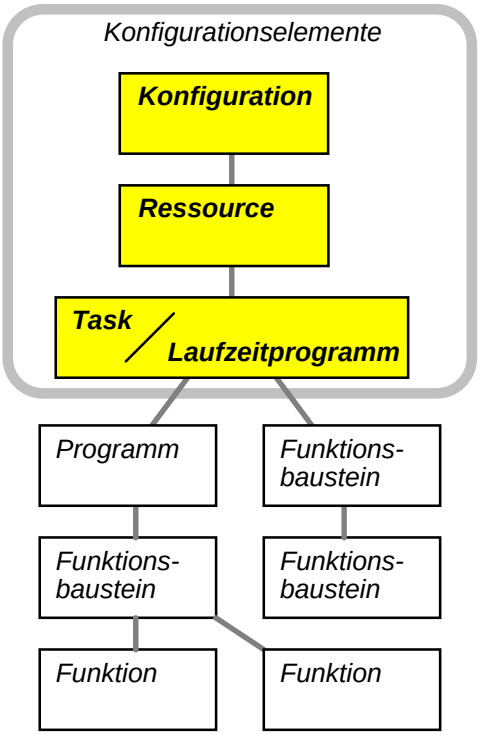


Abb. 6.1. Gesamtstruktur von SPS-Programmen nach IEC 61131-3 mit POEs und Konfigurationselementen

Wie Abb. 6.1 zeigt, liegen die *Konfigurationselemente* Konfiguration, Ressource und Task mit Laufzeitprogramm hierarchisch oberhalb der POE-Ebene.

Während POEs die Aufruf-Hierarchie bilden, dienen die Konfigurationselemente dazu, diesen POEs Eigenschaften zuzuordnen:

- Laufzeit-Eigenschaften von PROGRAMs und FBs festlegen,
- Kommunikationsbeziehungen zwischen Konfigurationen definieren,
- Abbildung der Programmvariablen auf die SPS-Hardware-Adressen vornehmen.

Zunächst werden Aufbau und Inhalt der Konfigurationselemente selbst vorgestellt.

6.2 Elemente einer realen SPS-Konfiguration

Konfigurationselemente entsprechen realen Elementen eines SPS-Systems:

- *Konfiguration*: SPS-System, z.B. Steuerung als Einbaurahmen mit mehreren (auch vernetzten) Zentraleinheiten (CPUs) auf Maschinenzellen-Ebene
- *Ressource*: (ggf. multitasking-fähige) SPS-CPU
- *Task*: Laufzeit-Eigenschaften für Programm(e) und Funktionsbausteine („Typ“ eines SPS-Programms)
- *Laufzeitprogramm*: Einheit aus PROGRAM bzw. FUNCTION_BLOCK und zugewiesener TASK

Hauptprogramm(e) einer CPU sind POEs vom Typ PROGRAM. Handelt es sich um ein größeres Anwenderprogramm, wird es bevorzugt in Ablaufsprache geschrieben und koordiniert den Aufruf der übrigen POEs.

Hauptprogrammen und Funktionsbausteinen werden Laufzeit-Eigenschaften wie „periodische Ausführung“ oder „Prioritätsstufe“ zugeordnet, wie es in Abb. 6.2 angedeutet wird.

Unter dem Begriff „Laufzeitprogramm“ versetht man die Einheit aus sämtlichen benötigten POEs *und* der TASK, d.h. ein Gesamtprogramm *mit* Laufzeit-Eigenschaften. Ein Laufzeitprogramm bildet in diesem Sinn eine gebundene, in sich abgeschlossene Programmeinheit, die selbständig in einer CPU ablaufen kann.

Abb. 6.2 veranschaulicht die Zuordnung von Konfigurationselementen zu Komponenten eines realen SPS-Systems (vgl. auch Abb. 2.4):

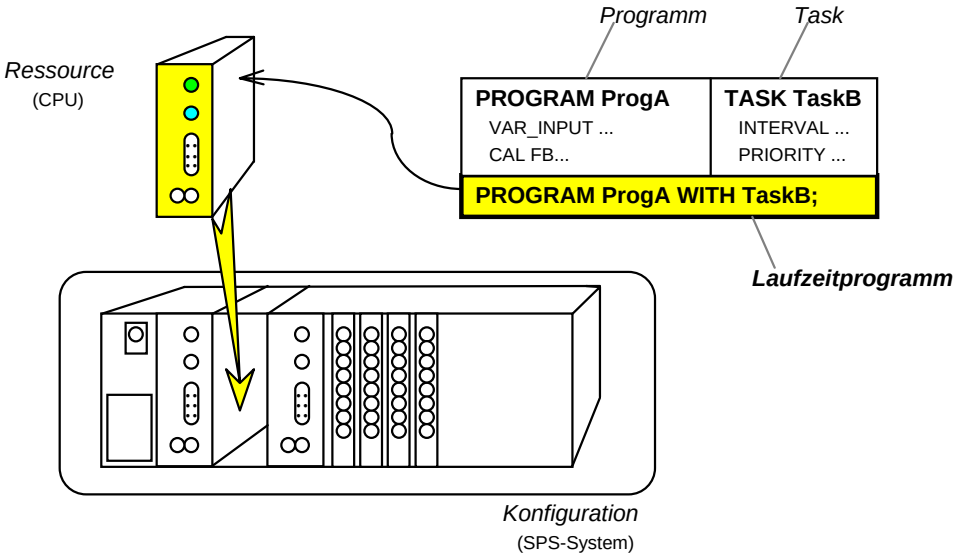


Abb. 6.2. Darstellung einer realen Konfiguration. ProgA und TaskB werden zu einem Laufzeitprogramm verbunden und der Ressource CPU des SPS-Systems zugeordnet.

Die tatsächliche Zuordnung von Konfigurationselementen zu den Elementen eines SPS-Systems hängt von dessen realer Hardware-Architektur ab.

Mit Hilfe der Konfigurationsbeschreibung können die Tasks *einer* CPU zugewiesen werden, auf der sie simultan ablaufen, oder auf mehrere CPUs, soweit vorhanden, verteilt werden.

Die Betrachtung der RESOURCE als eine CPU oder als CPU-Familie in einem Einbaurahmen hängt daher von der konkreten SPS-Architektur ab.

Für „kleine SPS-Systeme“ können die Aufgaben zur Konfiguration auch allein durch eine POE vom Typ PROGRAM wahrgenommen werden: Programme können die dazu notwendigen Globalen Variablen sowie Zugriffspfade und Direkt dargestellte bzw. Symbolische Variablen deklarieren. Laufzeit-Eigenschaften, CPU-Zuordnung usw. werden (implizit) durch Mechanismen des Programmiersystems bzw. SPS-Fähigkeiten ersetzt. Diese Vorgehensweise zur SPS-Projektierung entspricht der von herkömmlichen SPS-Systemen.

Diese Mächtigkeit der Programm-POEs begünstigt eine stückweise Migration von bestehenden zu IEC-konformen Programmen.

6.3 Die Konfigurationselemente

Zunächst werden die Aufgaben der Konfigurationselemente im Überblick beschrieben und anschließend detailliert erläutert. Dabei werden Teile des Beispiels in Abschn. 6.4 zur Erklärung verwendet.

6.3.1 Aufgaben

Die Konfigurationselemente erfüllen im einzelnen die in Tab. 6.1 zusammengefaßten Aufgaben:

Konfigurationselement	**Aufgabenbeschreibung**
Konfiguration	- Definition globaler Variablen (innerhalb Konfiguration gültig) - Zusammenstellung von Ressourcen eines SPS-Systems - Definition der Zugriffspfade zwischen Konfigurationen - Deklaration (globaler) Direkt dargestellter Variablen
Ressource	- Definition globaler Variablen (innerhalb Ressource gültig) - Zuordnung von Tasks und Programmen zur Ressource - Aufruf der Laufzeitprogramme mit Eingangs- und Ausgangsparametern - Deklaration (globaler) Direkt dargestellter Variablen
Task	- Definition von Laufzeit-Eigenschaften
Laufzeitprogramm	- Zuordnung der Laufzeit-Eigenschaften zu PROGRAM und FUNCTION_BLOCKs

Tab. 6.1. Aufgaben der Konfigurationslemente. Innerhalb des PROGRAM können ebenfalls (globale) Direkt dargestellte Variablen sowie Zugriffspfade deklariert werden.

Durch die Deklaration der Direkt dargestellten Variablen wird die Abbildung der Gesamt-Konfiguration auf die Peripherie-Adressen der SPS-Hardware vorgenommen. Um diese Zuordnungen POEs zugänglich zu machen, kann die Deklaration in VAR_GLOBAL in Konfiguration, Ressource oder im PROGRAM erfolgen, d.h. POEs können darauf mit Hilfe VAR_EXTERNAL zugreifen.

Zusammengefaßt ergeben sämtliche Deklarationsabschnitte aller POEs für Direkt dargestellte Variablen die Belegungsliste des SPS-Programms. Damit kann auch eine Umverdrahtung, also die Neuzuordnung von symbolischen zu absoluten SPS-Peripherie-Adressen durchgeführt werden.

Die Konfigurationselemente werden weitgehend in textueller Form deklariert. Die Norm definiert eine grafische Darstellung für TASK, doch die grafische Realisie-

rung der übrigen Konfigurationselemente ist dem konkreten Programmiersystem überlassen und daher implementierungsabhängig.

6.3.2 Die CONFIGURATION

Die IEC 61131-3 verwendet die Konfiguration (CONFIGURATION), um die Ressourcen (RESOURCE) eines SPS-Systems zusammenzufassen und ihnen Möglichkeiten zum Datenaustausch zur Verfügung zu stellen. Eine Konfiguration besteht aus den in Abb. 6.3 gezeigten Teilen:

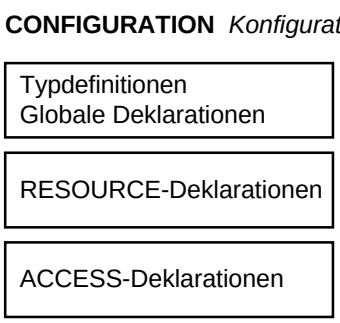


Abb. 6.3. Aufbau der Deklaration einer Konfiguration

Innerhalb von Konfigurationen können Typdefinitionen erfolgen, die global für das gesamte SPS-Projekt gelten. Diese Möglichkeit besteht in den anderen Konfigurationselementen nicht.

Die Kommunikation zwischen Konfigurationen, d.h. ihr Datenaustausch untereinander, erfolgt über Zugriffspfade in VAR_ACCESS. Variablen in VAR_GLOBAL sind nur innerhalb der Konfiguration gültig und stehen dort sämtlichen Ressourcen, Programmen und FBs zur Verfügung. Die Variablenart VAR_EXTERNAL ist deshalb in der Konfigurationsbeschreibung nicht verwendbar.

Kommunikationsbausteine dienen der globalen Kommunikation in einem SPS-System (zwischen Konfigurationen) und sind Inhalt des Teils 5 der IEC 61131 (vgl. auch Abschn. 6.5).

```
CONFIGURATION SPS_Zelle1
  VAR_GLOBAL ...  END_VAR
  RESOURCE CPU_Band ON CPU_001 ... END_RESOURCE
  RESOURCE CPU_Walz ON CPU_002... END_RESOURCE
  VAR_ACCESS ... END_VAR
END_CONFIGURATION
```

Bsp. 6.1. Elemente der CONFIGURATION von Bsp. 6.6

Bsp. 6.1 zeigt Abschnitte zur Deklaration einer Konfiguration mit Namen SPS_Zelle1.

Sie enthält einen Abschnitt mit globalen Variablen, der für andere Konfigurationen nicht sichtbar ist, sondern nur den Ressourcen CPU_Band und CPU_Walz sowie den darauf laufenden POEs zur Verfügung stehen.

Der Kommunikationsbereich VAR_ACCESS wird benutzt, um Daten zwischen den Ressourcen einer bzw. mehrerer Konfigurationen austauschen zu können (Zugriffspfade).

Konfigurationen und Ressourcen enthalten keine Anweisungen wie POEs, sondern regeln ausschließlich die Beziehung der zugewiesenen POEs untereinander und nach außen.

6.3.3 Die RESOURCE

Die Deklaration von Ressourcen dient der Zuordnung von TASKs an die physikalischen Betriebsmittel (engl.: resource) eines SPS-Systems. Ressourcen bestehen aus den in Abb. 6.4 gezeigten Teilen:

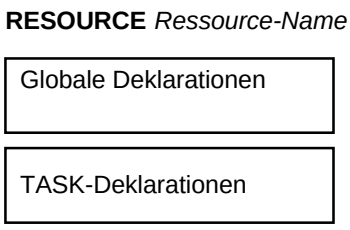


Abb. 6.4. Aufbau der Deklaration einer Ressource

Durch den Ressourcen-Namen wird einer SPS-CPU ein symbolischer Name zugeordnet. Die Typen und Nummern der in einem SPS-System zur Verfügung stehenden Ressourcen (individuelle CPU-Bezeichnungen) werden vom Programmiersystem zur Verfügung gestellt und auf ihre korrekte Verwendung überwacht.

Mit Hilfe der auf Ressource-Ebene zulässigen globalen Variablen können die auf eine CPU begrenzten Daten verwaltet werden.

```
RESOURCE CPU_Band ON CPU_001
  TASK ...
  PROGRAM ... WITH ...
END_RESOURCE
RESOURCE CPU_Walz ON CPU_002
  VAR_GLOBAL ... END_VAR
  TASK ...
  PROGRAM ... WITH ...
END_RESOURCE
```

Bsp. 6.2. Elemente der Ressourcen von Bsp. 6.6 (Detailausschnitt aus Bsp. 6.1)

Bsp. 6.2 zeigt Abschnitte zur Deklaration zweier Ressourcen. Auf die globalen Daten der Ressource CPU_002 kann von Ressource CPU_001 aus nicht zugegriffen werden.

Das Schlüsselwort PROGRAM besitzt innerhalb einer Ressource-Definition eine andere Bedeutung als zu Beginn einer Programm-POE!

Innerhalb der Ressource wird PROGRAM ... WITH zur Verknüpfung der TASK mit dem Programm verwendet, das vom Typ PROGRAM sein muß.

6.3.4 Die TASK mit Laufzeitprogramm

Die Aufgabe der TASK-Definition liegt in der Vereinbarung und Zuweisung von Laufzeiteigenschaften für Programme und deren FBs.

Bisher war es bei SPS-Systemen üblich, spezielle Bausteinarten zu verwenden (z.B. Organisationsbausteine OB), die implizit vordefinierte Laufzeit-Eigenschaften besitzen. Diese können eine zyklische Bearbeitung beinhalten oder Eigenschaften des SPS-Systems für Interrupt-Verarbeitung oder Fehler-Reaktionen ausnutzen.

Mit der Definition einer TASK nach IEC 61131-3 werden solche Programm-Eigenschaften explizit und herstellerunabhängig formuliert. Dadurch können solche Programme besser dokumentiert und einfacher gewartet werden.

Abb. 6.5 zeigt den prinzipiellen Aufbau einer textuellen TASK-Deklaration.

TASK Task-Name (Task-Eigenschaften)

PROGRAM Programm-Name **WITH** Task-Name : (PROGRAM-Schnittstelle)

Abb. 6.5. Aufbau der textuellen Deklaration eines Laufzeitprogramms durch Definition der Task und Zuordnung von Task zum Programm (PROGRAM). Die „Task-Eigenschaften" beschreiben die Parameterwerte der Task, die „PROGRAM-Schnittstelle" enthält die Zuordnung von Aktual- zu Formalparametern.

Durch die Zuordnung von PROGRAM und TASK entsteht ein *Laufzeitprogramm* mit Programm-Name. Dieses bildet den Instanznamen des Programms vom Typ PROGRAM, dessen „Aufrufschnittstelle" bei der Deklaration angegeben wird. Diese Schnittstelle beinhaltet Ein- *und* Ausgangsparameter der Programm-POE (im Unterschied zu FBs) und wird einmalig beim Starten der Ressource initialisiert. Ein PROGRAM kann durch diese Deklaration in mehreren Instanzen (Laufzeitprogramme) ausgeführt werden.

Eine TASK kann neben dieser textuellen Darstellung auch grafisch wie in Abb. 6.6 deklariert werden. Dabei wird die Liste der TASK-Eigenschaften als Eingänge eines Kastens dargestellt. Die grafische Darstellung der Zuordnung zum Programm (PROGRAM...WITH...) wird durch die IEC 61131-3 allerdings nicht vorgegeben.

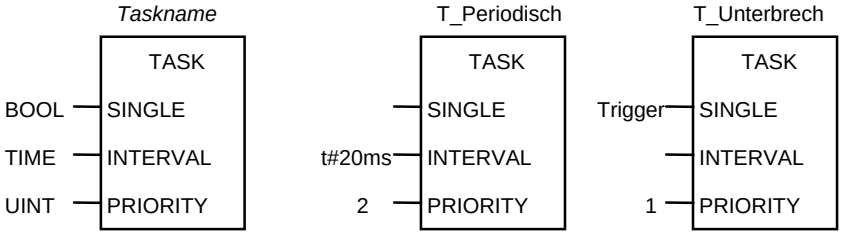


Abb. 6.6. Grafische Deklaration einer Task. Links in allgemeiner Darstellung, rechts zwei Tasks von Bsp. 6.6

Für die Task-Eigenschaften können die in Tab. 6.2 enthaltenen Eingangsparameter der TASK angegeben werden.

TASK-Parameter	**Bedeutung**

SINGLE	Mit steigender Flanke an diesem Eingang werden die mit der TASK verknüpften Programme einmal aufgerufen und ausgeführt.
INTERVAL	Wenn dieser Eingang einen Zeitwert ungleich Null besitzt, werden die mit der TASK verknüpften Programme zyklisch (periodisch) aufgerufen. Dabei gibt der Wert dieses Eingangs den Abstand zwischen zwei Aufrufen an. Dieser Wert kann also zum Einstellen und Überwachen der Programm-Zykluszeiten benutzt werden. Ist dieser Eingang Null, wird die Task *nicht* aufgerufen.
PRIORITY	Dieser TASK-Eingang legt die Priorität eines verknüpften Programms gegenüber anderen gleichzeitig laufenden Programmen fest, koordiniert also Multi-Tasking eines SPS-Systems. Die Bedeutung ist daher implementierungsabhängig (siehe Text).

Tab. 6.2. TASK-Eigenschaften als Eingangsparameter

Die Bedeutung des Eingangs PRIORITY hängt davon ab, auf welche Weise das SPS-System das Zusammenspiel mehrerer Tasks regelt und ist daher implementierungsabhängig. Wird eine Task aktiviert, die eine höhere Priorität aufweist als die gerade laufende besitzt, gibt es prinzipiell zwei Möglichkeiten, diesen Laufzeitkonflikt dieser beiden Tasks auf dieselbe Ressource (CPU) zu lösen. Es hängt von der Funktionalität des SPS-Betriebssystems ab, ob eine laufende Task *unterbrechbar* ist oder nicht:

1) Die laufende Task wird sofort *unterbrochen* und durch die höherpriore Task ersetzt. Dieses Verfahren nennt die IEC 61131-3 „vorberechtigter Aufruf“ (engl.: preemptive scheduling) einer Task und bringt damit zum Ausdruck, daß TASKs mit höherer Priorität absoluten Vortritt besitzen und *sofort* zur Ausführung gelangen.
2) Die laufende Task wird *nicht* unterbrochen, sondern normal beendet. Danach kommt diejenige TASK zur Ausführung, die von allen *wartenden* TASKs die höchste Priorität besitzt. Dieses Verfahren wird „nicht-vorberechtiger Aufruf“ (engl.: non-preemptive scheduling) genannt.

Beide Verfahren dienen dazu, der höherprioren Task die Kontrolle über die angeforderte Ressource zu geben. Besitzt die anfordernde TASK dieselbe Priorität wie die laufende, muß sie warten. Falls TASKs derselben Priorität warten, wird jeweils diejenige mit der längsten Wartezeit ausgewählt.

```
TASK T_Schnell          (INTERVAL := t#8ms,   PRIORITY := 1);
PROGRAM Antrieb WITH T_Schnell  : ProgA (          RegPar := %MW3,
                                          R_Wert => FehlerCode);

TASK T_Unterbrech     (SINGLE := Trigger, PRIORITY := 1);
PROGRAM Schmier WITH T_Unterbrech  : ProgC;
```

Bsp. 6.3. Elemente TASK und PROGRAM...WITH... von Bsp. 6.6

In Bsp. 6.3 werden die beiden TASKs T_Schnell (periodisch mit kurzer Zykluszeit) und T_Unterbrech (Unterbrechung mit hoher Priorität) definiert.

T_Schnell wird im Abstand von 8 Millisekunden angestoßen. Wenn die Ausführungszeit des damit verknüpften Programms Antrieb diesen Wert überschreiten sollte (z.B. aufgrund Unterbrechungen durch höherpriore Tasks), meldet das SPS-System den Laufzeitfehler „Zykluszeit-Überschreitung".

Das Programm Schmier besitzt dieselbe Priorität wie Antrieb, muß also warten, falls die Eingangsbedingung Trigger auf TRUE wechselt.

Bei der Zuordnung eines Programms können Aktualparameter übergeben werden, wie hier für RegPar des ProgA mit einer Direkt dargestellten Variable gezeigt. Zur Laufzeit werden diese Parameter bei jedem Aufruf des Laufzeitprogramms gesetzt. Im Unterschied zu FBs können hier neben den Eingangsparametern auch Ausgangsparameter angegeben werden, die jeweils bei Beendigung des Programms aktualisiert werden. In Bsp. 6.3 erfolgt dies für R_Wert des ProgA. Zuweisungen solcher *Ausgangs*parameter an Variablen werden mit „=>" anstelle von „:=" versehen, um den Unterschied zur Versorgung von *Eingangs*parametern deutlich hervorzuheben.

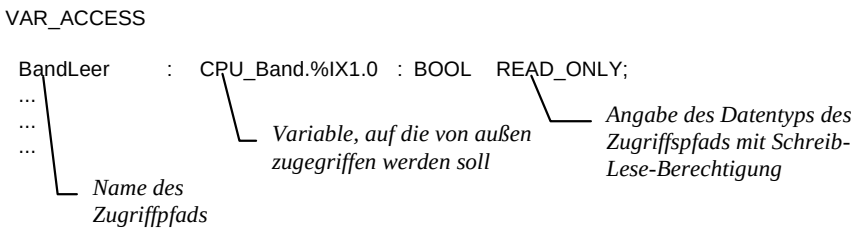
Wird ein PROGRAM *ohne* TASK-Verknüpfung deklariert, hat dieses Programm gegenüber allen anderen niedrigste Priorität und wird standardmäßig zyklisch aufgerufen.

6.3.5 Die ACCESS-Deklaration

Mit dem Sprachkonstrukt VAR_ACCESS ... END_VAR können sogenannte *Zugriffspfade* als transparente Kommunikationsverbindungen zwischen Konfigurationen definiert werden.

Sie stellen eine Erweiterung gegenüber den globalen Variablen dar, die innerhalb der Konfiguration gelten. Für Zugriffspfade können Schreib- und Lese-Attribute angegeben werden.

Variablen der Konfiguration werden unter symbolischen Bezeichner anderen Konfigurationen bekanntgegeben.



Bsp. 6.4. Deklaration von Zugriffspfaden

Bsp. 6.4 zeigt den Aufbau für Deklarationen von Zugriffspfaden anhand der Variable BandLeer von Bsp. 6.6.

Zugriffspfade können für folgende Variablenarten gebildet werden:

- Ein- oder Ausgangsvariablen von PROGRAM,
- globale Variable,
- Direkt dargestellte Variable.

Zugriffspfade machen diese Variablen unter neuem Namen über eine Konfiguration hinaus bekannt, so daß beispielsweise über Kommunikationsbausteine darauf zugegriffen werden kann.

Falls diese Variablen vom Typ Feld oder Struktur sind, kann der Zugriffspfad nur für ein einzelnes ihrer Feld- oder Strukturelemente gebildet werden.

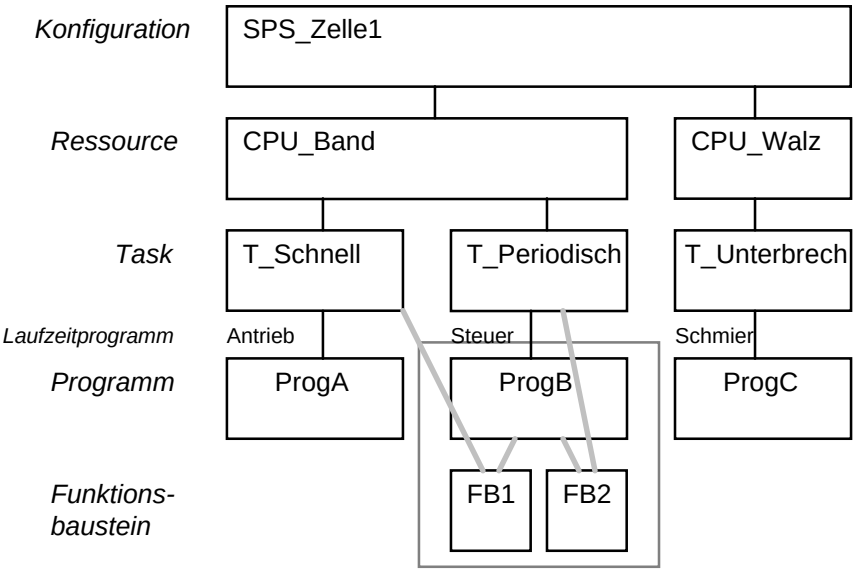
Für Zugriffspfade ist standardmäßig nur ein lesender Zugriff (READ_ONLY) zulässig. Durch die Angabe READ_WRITE kann ein Überschreiben des Zugriffspfads durch andere Ressourcen explizit erlaubt werden.

Diese Angaben werden unmittelbar hinter den Datentyp der Zugriffsvariablen gesetzt. Der Datentyp von Zugriffsvariablen muß identisch mit dem der dazugehörigen Variablen sein.

6.4 Konfigurations-Beispiel

Bsp. 6.5 zeigt eine Konfiguration in Übersichts-Darstellung. Dieses Beispiel wird in Bsp. 6.7 und Bsp. 6.6 textuell deklariert. Die Konfiguration besteht aus einem SPS-System mit zwei CPUs, denen mehrere Programme und Funktionsbausteine als Laufzeitprogramme zugeordnet werden.

Teilabschnitte dieses Beispiels wurden in den vorangegangenen Abschnitten bereits erläutert.



Bsp. 6.5. Beispiel für Konfigurationselemente mit POEs in Übersichts-Darstellung

In Bsp. 6.6 besteht die Maschinenzelle SPS_Zelle1 physikalisch aus 2 CPUs, von denen die erste zwei Tasks ausführen kann: eine schnelle, periodische mit kurzer Zykluszeit und eine langsamere periodische, während auf der zweiten CPU eine Task mit Interrupt-Verhalten läuft.

```
CONFIGURATION SPS_Zelle1

VAR_GLOBAL
  FehlerCode    :   DUINT;
  AT %MW3       :   WORD;
  Start         :   INT;
END_VAR

RESOURCE CPU_Band ON CPU_001
  TASK T_Schnell       (INTERVAL := t#8ms,   PRIORITY := 1);
  TASK T_Periodisch    (INTERVAL := t#20ms, PRIORITY := 3);
  PROGRAM Antrieb WITH T_Schnell  : ProgA (RegPar := %MW3);
  PROGRAM Steuer  WITH T_Periodisch : ProgB (InOut := Start,
                                             R_Wert => FehlerCode,
                                             FB1 WITH T_Schnell,
                                             FB2 WITH T_Periodisch);
END_RESOURCE

RESOURCE CPU_Walz ON CPU_002
  VAR_GLOBAL
    Trigger  AT %IX2.5 : BOOL;
  END_VAR
  TASK T_Unterbrech   (SINGLE := Trigger, PRIORITY := 1);
  PROGRAM Schmier WITH T_Unterbrech  : ProgC;
END_RESOURCE

VAR_ACCESS
  RegelP        : CPU_Band.Antrieb.RegPar  : WORD READ_WRITE;
  BAND_LEER     : CPU_Walz.%IX1.0          : BOOL  READ_ONLY;
END_VAR

END_CONFIGURATION
```

Bsp. 6.6. Textuelle Darstellung des Beispiels in Bsp. 6.5. Die Variablennamen und Namen von Programmen und FBs sind fett dargestellt.

```
PROGRAM ProgA
  VAR_INPUT
    RegPar : WORD;
  END_VAR

END_PROGRAM
```

```
PROGRAM ProgB
  VAR_IN_OUT
    InOut : INT;
  END_VAR
  VAR_OUTPUT
    R_Wert : DUINT;
  END_VAR
  ...
  CAL Inst_FB2
  ...
END_PROGRAM
```

```
PROGRAM ProgC


  ...
  CAL Inst_FB3
  ...
END_PROGRAM
```

Bsp. 6.7. Programme zum Beispiel in Bsp. 6.6; FB3 ist dort nicht sichtbar, FB1 könnte z.B. der Ausnahme- und Fehlerbehandlung dienen.

In diesem Beispiel werden die Laufzeitprogramme Antrieb, Steuer und Schmier durch Verknüpfung der PROGRAMs ProgA, ProgB und ProgC mit TASK-Definitionen gebildet.

Sowohl Programm Antrieb als auch FB-Instanz FB1 (dadurch unabhängig vom Programm Steuer) laufen auf CPU_Band als schnelle Tasks (T_Schnell), FB2 auf derselben CPU (CPU_001) in Programm Steuer als periodische Task (T_Periodisch). Das Programm Steuer wird hier verwendet, um auch das Laufzeitverhalten der verwendeten FB-Tasks festzulegen. Auf diese Weise Tasks zugeordnete FBs werden nach IEC 61131-3 **unabhängig** vom Programm ausgeführt.

Beim periodischen Aufruf des Laufzeitprogramms Steuer wird der Eingangsparameter InOut mit der Variablen Start belegt. Nach Beendigung von Steuer wird dessen Ausgangsvariable R_Wert der globalen Variable FehlerCode zugewiesen.

Auf CPU_Walz (zweite CPU) läuft das Programm Schmier als Interruptgetriebene Task (T_Unterbrech). FB3 erhält als Unterprogramm von ProgC dadurch automatisch dieselben Laufzeit-Eigenschaften.

In diesem Beispiel sind CPU_001 und CPU_002 keine Variablen, sondern Hersteller-Bezeichnungen für SPS-Zentraleinheiten von SPS_Zelle1.

6.5 Kommunikation bei Konfigurationen und POEs

Dieser Abschnitt beschreibt Möglichkeiten zum Datenaustausch zwischen Konfigurationen sowie innerhalb einer Konfiguration über gemeinsame Datenbereiche.

Solche (strukturierten) Datenbereiche dienen der Kommunikation verschiedener Programmteile untereinander, zum Datenaustausch, für Synchronisationsmechanismen oder zur Diagnose-Unterstützung.

Ziel der IEC 61131-3 ist es, mit Hilfe einheitlichen Kommunikationsmodells gut strukturierte SPS-Programme zu ermöglichen, die sowohl Inbetriebnahme und Diagnose erleichtern als auch die Anlagen-Dokumentation verbessern.

Durch anwendungsgerechte Modularisierung einer Aufgabe entstehen wiederverwendbare Programmteile, die den Innovationszyklus bei der Entwicklung von SPS-Programmen verkürzen helfen.

Die IEC 61131-3 legt fest, wie Programmteile miteinander kommunizieren, also Daten austauschen können. Dazu gibt es folgende Möglichkeiten:

- Direkt dargestellte Variable,
- Ein- und Ausgangsvariable bzw. Funktionswerte bei POE-Aufrufen,
- globale Variable (VAR_GLOBAL, VAR_EXTERNAL),
- Zugriffspfade (VAR_ACCESS),
- Kommunikations-Bausteine (IEC 61131-5)
- Aufrufparameter.

Die drei ersten Fälle dienen der Kommunikation *innerhalb* einer Konfiguration, während Zugriffspfade und Kommunikations-Bausteine *zwischen* Konfigurationen bzw. nach außen verwendet werden.

Die Direkt dargestellten Variablen sind eigentlich nicht für die Kommunikation zwischen Programmteilen vorgesehen, obwohl dies als prinzipielle Möglichkeit hier mit aufgenommen wurde. Insbesondere ist das Schreiben auf SPS-Eingänge (%I...) dazu ungeeignet. Ausgänge (%Q...) sollten zur Steuerung des Prozesses und nicht zur Zwischenablage von Informationen verwendet werden.

Diese Mechanismen können, wie in Tab. 6.3 gezeigt, auf mehreren Ebenen benutzt werden, wobei Konfigurationselemente und POEs verschiedene Zugriffsmöglichkeiten besitzen.

Kommunikationspfad	**CONF**	**RES**	**PROG**	**FB**	**FUN**
Zugriffspfade	x		x		
Direkt dargest. Variable	x	x	x		
globale Variable	x	x	x		
externe Variable			x	x	
Kommunikations-Bausteine			x	x	
Aufrufparameter		x	x	x	x

Legende: CON: CONFIGURATION
RES: RESOURCE
PROG: PROGRAM
FB: FUNCTION_BLOCK
FUN: FUNCTION

Tab. 6.3. Kommunikationsmöglichkeiten von Konfigurationselementen und POEs

Zugriffspfade dienen dem Datenaustausch zwischen Konfigurationen, also über die Grenzen eines SPS-Systems hinaus und sind auf der Ebene von Konfigurationen und Programmen zulässig.

Direkt dargestellte Variable der SPS (z.B. über ein Prozeßabbild oder durch direkten Zugriff auf die E/A-Peripherie, „%I.., %Q.., %M.“) erlauben ebenfalls eine

(bedingte) Kommunikation zwischen Programmteilen, da sie systemweit wie globale Datenbereiche zur Verfügung stehen. Merker (%M..) können als SPS-globaler Datenbereich (z.B. zur Synchronisation von Ereignissen) eine Rolle spielen.

Diese Variablen dürfen nur auf Programmebene oder höher deklariert werden (z.B. als global) und von Funktionsbausteinen als externe Variablen verwendet werden. Dies stellt ein wichtiges Unterscheidungsmerkmal der Programmierung nach IEC 61131-3 gegenüber bisher üblicher SPS-Programmierung dar.

Globale Variable können für Konfigurationen, Ressourcen und Programme angelegt und auf diesen Ebenen benutzt werden.

Funktionsbausteine können diese Datenbereiche zwar lesend und schreibend (als externe Variablen) benutzen, nicht jedoch selbst anlegen oder initialisieren. Funktionen besitzen keinen Zugriff auf globale oder externe Variablen.

Externe Variable können von Programmen und Funktionsbausteinen importiert werden, wenn sie an anderer Stelle als global deklariert wurden..

Kommunikations-Bausteine sind spezielle Funktionsbausteine, die Daten vom Aufrufer zum Empfänger „paketweise transportieren“. Da diese FBs zum Programm dazugebunden werden, sind sie lokal innerhalb der Konfiguration bekannt und können nicht nach außen sichtbar sein.

Die Definition geeigneter Standard-Kommunikationsbausteine befindet sich mit Teil 5 der IEC 61131 („Kommunikationsdienste“) noch in Bearbeitung und wird hier nicht ausgeführt.

Aufrufparameter werden als Ein- und Ausgangsparameter beim Aufruf von POEs verwendet. Durch sie können Daten in POEs hinein bzw. aus POEs zurückgegeben werden.

Wie in Kap. 2 erläutert wurde, kann die Parametrierung von Eingangsvariablen bzw. die Abfrage der Ausgangsvariablen von Funktionsbausteinen auch beliebig weit vor oder nach dem eigentlichen Aufruf erfolgen, so daß sich auch hier der Charakter eines Kommunikationsmechanismus ergibt, der über die Möglichkeiten bisheriger SPS-Programmierung hinausgeht.

Ressourcen können Programmen bei der Zuordnung von Tasks Werte übergeben, wie es für Antrieb in Bsp. 6.6 gezeigt wurde. Bei jedem Aufruf des Programms werden diese Werte als Aktualparameter erneut übergeben bzw. als Ausgangsparameter gelesen.

7 Innovative SPS-Programmiersysteme

Dieses Kapitel geht über die Festlegungen der IEC 61131-3 hinaus und beschreibt allgemeine Anforderungen an die neue Generation von Programmiersystemen. Dies umfaßt vor allem Markterfordernisse, die sich aus dem speziellen SPS-Umfeld ergeben und nun im Rahmen der neuen Technologie und Norm in eine geeignete Form umzusetzen sind.

7.1 Anforderungen an innovative Programmierwerkzeuge

Die Leistungsfähigkeit eines SPS- Programmiersystems kann man an folgenden drei Eigenschaften messen:

- Technologische Neuerung,
- Erfüllung der SPS-spezifischen Anforderungen,
- Kosten/Nutzen- Verhältnis.

Die nachfolgenden Abschnitte gehen auf diese Eigenschaften sowie die wesentlichsten Bestandteile eines Programmiersystems ein.

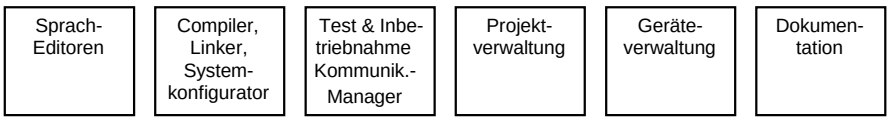


Abb. 7.1. Wichtige Bestandteile eines modernen Programmiersystems

7.2 Technologischer Wandel

Die rasante technologische Entwicklung der PCs beeinflußt, wie in den meisten anderen EDV-Bereichen, die Ausprägung der SPS-Programmierwerkzeuge. Wichtige Neuerungen sind:

1) die erhöhte Prozessorleistung,
2) der Einsatz von vollgrafischen Bildschirm- und Druckausgaben,
3) leistungsfähigere Betriebssysteme,
4) einheitliche Mensch/Maschinen- Schnittstelle.

7.2.1 Prozessorleistung

Die verbesserte Prozessorleistung der PCs ermöglicht den Einsatz leistungsstarker grafischer Editoren zur Darstellung komplexer Strukturen am Bildschirm.

Außer den klassischen Assembler- (AWL) und Grafiksprachen (AS, KOP, FBS) können Hochsprachen-Compiler (ST, C) eingesetzt werden, die einen effizienten Code generieren, indem sie spezielle SPS-Prozessor- und Betriebssystemeigenschaften optimal nutzen.

7.2.2 Vollgrafische Bildschirm- und Druckausgaben

Die Ausgabe auf Bildschirm und Drucker kann durch Zoomfunktionen (vergrößerte bzw. verkleinerte Darstellung) variabel eingestellt werden.

Durch die verbesserte Auflösung lassen sich gegenüber semigrafischen Systemen mehr Informationen gleichzeitig am Bildschirm oder auf dem Ausdruck darstellen.

Der sichtbare Bildschirmausschnitt ist lediglich ein Teil des ungleich größeren Arbeitsbereichs, auf dem beliebig (horizontal und vertikal) positioniert werden kann. Im Bereich der grafischen Sprachen lassen sich somit deutlich komplexere Verknüpfungs- und Ablaufschaltungen entwickeln.

7.2.3 Betriebssysteme

MS-DOS-basierte Systeme waren an die bekannte 640 KB-Grenze gebunden, die nur durch aufwendige und zeitraubende Speicherverwaltungen umgangen werden konnte. OS/2 und Windows lösen diese Grenzen auf. Die 16 Bit-Adressierung von Windows 3.1 besitzt zwar immer noch unerwünschte Grenzen (Segmentgrößen liegen bei max. 64 KB); seit Windows 95 bzw. Windows NT ist auch diese Grenze gefallen.

Durch die Multitasking-Technik solcher neuen Betriebssysteme können mehrere Aufgaben der Programmierwerkzeuge gleichzeitig ablaufen; die Synchronisation erfolgt an genau festgelegten Programmpunkten.

7.2.4 Einheitliche Mensch- / Maschinen- Schnittstelle

Die von IBM eingeführte Industrienorm SAA/CUA hatte zum Ziel, für Software eine möglichst gleichartige Bedienoberfläche vorzugeben, unabhängig davon, ob sie auf einem Großrechner oder PC abläuft.

Ähnlich geartete Funktionen sind über identische Tasten (z.B. die Hilfefunktion über die Funktionstaste F1) aufrufbar. Eine vergleichbare Bildschirmaufteilung (Menüs, Funktionsleisten, ...) erleichtert das Zurechtfinden in unterschiedlichen Softwarepaketen. Umfangreiche Hilfesysteme verstärken diesen Vorteil. Die Kommunikation eines Programms mit dem Anwender wurde über einheitliche Dialogfelder vereinfacht.

Dieses einheitliche Erscheinungsbild wurde in den beiden Betriebssystemen OS/2 und MS-Windows konsequent realisiert.

Da beide die Basis für die meisten neuen IEC 61131-3- Programmiersysteme bilden, wird die Bedienung der Programmiersysteme unterschiedlicher Hersteller immer ähnlicher. Kopieren, Ausschneiden oder Suchen von Daten besitzen gleichlautende Menübezeichnungen oder sind über die gleichen Kurzwahltasten anwählbar. Die Menüleiste besitzt den gleichen Aufbau wie Standard-PC-Programme (Word, Excel für Windows, ...).

7.3 Rückübersetzung (Rückdokumentation)

Die Rückdokumentation ist eine klassische Forderung im SPS-Markt. Das SPS-Programm sollte rücklesbar sein, wenn es sich (außerhalb von Büroumgebungen) bereits in der Anlage befindet und beispielsweise zu Wartungszwecken geändert wird. Dabei sollte der SPS-Techniker die Möglichkeit erhalten, auch ohne daß die ursprünglichen Programmquellen auf einem PC zur Verfügung stehen, das Programm wieder lesen und ausdrucken sowie ändern zu können.

Rückübersetzbarkeit beschreibt die Fähigkeit, die zur Darstellung einer POE benötigte Information allein aus der SPS gewinnen (Rückdokumentation) und für Programmänderungen verwenden zu können.

AWL Code	INTEL 8086 Mnemo-Code (Quelldarstellung)
VAR V1, V2, Vst: SINT; END_VAR	

LD V1
AND V2
ST Vst
	MOV AL, V1
	AND AL, V2
	MOV Vst, AL

Bsp. 7.1. Beispiel eines rückübersetzbaren Quellcodes; zusätzlich bedarf es Information über die Variablen (Name, Adresse, Typ).

Die Eigenschaft der Rückübersetzung ist in verschiedenen Abstufungen zu finden:

- keine Rückübersetzung,
- Rückübersetzung mit Symbolik und Kommentaren,
- Rückübersetzung inkl. Grafik-Informationen,
- Quellcode in der SPS.

7.3.1 Keine Rückübersetzung

Die meisten neuen IEC-Systeme besitzen keine Rückübersetzungs-Eigenschaft. Die Anzahl von frei wählbaren symbolischen Namen, die bei einer Rücktransformation aus dem Maschinencode benötigt werden, hat sich deutlich vergrößert, was den Speicherausbau heutiger SPSen oft sprengt.

SPS-Prozessoren sind selten vom SPS-Hersteller entwickelte Bausteine (wie ASIC oder BitSlice), die mit einem eigens dafür entworfenen Maschinencode arbeiten. Aus Kostengründen werden immer häufiger Standardprozessoren verwendet. Diesen ausführbaren Code wieder in einen AWL- oder ST- Code zurückzuwandeln, ist ungleich schwieriger.

Selbstverständlich müssen Programme nach einer Inbetriebnahme noch modifizierbar sein. Stand der Technik ist es, die gesamte Projekt-Information (Quellen, Bibliotheken, Schnittstellen- und Konfigurations- Information) auf der Festplatte zu halten. Idealerweise liegen dabei die Quellen in einer sprachunabhängigen, anwenderlesbaren Form vor und können in einer beliebigen Sprache dargestellt werden. Geeignete Sicherungsprogramme müssen vor dem Beginn einer weiteren Projektbearbeitung sicherstellen, daß die Programme auf der SPS mit dem Stand des Projekts auf der Festplatte übereinstimmen.

7.3.2 Rückübersetzung mit Symbolik und Kommentaren

Der Binärcode eines Programms, der aus der SPS geladen wird, reicht allein nicht aus, um eine modifizierbare Quelle zu generieren. Es sind von der SPS Listen zu liefern, die Angaben über die aktuelle Verdrahtung beinhalten (CONFIGURATION). Dies umfaßt die Zuordnung von Symolischen Variablen an die physikalischen Adressen, die Verbindung globaler Variablen oder die Abbildung einzelner Programme an Tasks, Ressourcen, usw.

Ein weiteres Informationspaket stellen die symbolischen Namen (für Variablen, Sprungmarken, etc.) dar, die es in einem ausführbaren Code in der Regel nicht mehr gibt. Diese Symboltabelle aus der Programm-Erstellungsphase mit der Zuordnung von Namen zu Adressen ist, evtl. ergänzt um Kommentare zu Anweisungen und Deklarationen, in der SPS zu speichern, wenn eine Rückübersetzung direkt aus der SPS benötigt wird.

7.3.3 Rückübersetzung inkl. Grafik- Information

In der SPS befindet sich ausführbarer Code. Um aus diesem Code eine grafische Darstellung (KOP, FBS, AS) am PC zu erzeugen, muß der Code bestimmten syntaktischen Regeln gehorchen oder es sind im Code Zusatzdaten enthalten, die zum Aufbau genutzt werden. Die erste Variante besitzt deutlich kürzere Programme, doch es gibt Einschränkungen beim Aufbau der grafischen Darstellung.

7.3.4 Quellcode in der SPS

Die Komplexität heutiger IEC-Systeme erschwert die Realisierung der Informationsverpackung im Code. Um alle zur Rückübersetzung nötigen Informationen auf der SPS hinterlegen zu können, bietet sich eine Lösung an, bei der komplette Projekte komprimiert in spezielle (langsame und billige) Festwertspeicher der SPS geladen werden. Von dort können sie später komplett auf die Festplatte des PCs rücktransportiert werden. Mit diesen Daten kann dann das Programmiersystem arbeiten wie in der Phase der Programmerstellung.

7.4 Sprachverträglichkeit

Die fünf verschiedenen Sprachen der IEC 61131-3 stehen in einer speziellen Relation zueinander.

Eine besondere Stellung nimmt die Ablaufsprache mit ihren beiden Darstellungsarten (textuell, grafisch) ein, da sie nicht zur Formulierung eines Berech-

nungsalgorithmus, sondern eher zur Strukturierung und Ablaufsteuerung eines Programms verwendet wird.

Die Verknüpfungen und Berechnungen selbst sind in einer der anderen Programmiersprachen definiert und befinden sich in den Aktionsblöcken; siehe dazu die Bsp. 4.48 und 4.49 aus Abschn. 4.5.6.

Ein Programm besteht aus Bausteinen (POEs), die durch Aufrufe miteinander verbunden sind. Diese POEs sind soweit voneinander unabhängig, daß sie in unterschiedlichen Sprachen, ja selbst in IEC-fremden Sprachen definiert sein dürfen, solange sie sich an die Aufrufkonvention halten.

Die IEC 61131-3 geht über diese Schnittstellen-Festlegung nicht hinaus. Es stellt sich folgende Frage:

> Besteht die Notwendigkeit, die in einer Sprache entwickelte Befehlssequenz in einer anderen Sprache darzustellen?

Diese mehrsprachige Verwendung innerhalb eines Programms und evtl. wahlfreie Darstellbarkeit von POEs wird in den nächsten beiden Abschnitten unter den Stichwörtern *Querübersetzbarkeit* und *Sprachunabhängigkeit* gezeigt.

7.4.1 Querübersetzbarkeit

Die IEC 61131-3 setzt nicht voraus, daß eine POE, die in *einer* Sprache entwickelt wurde, in einer *anderen* darstellbar sein muß. Die Diskussion über eine solche Notwendigkeit ist so alt wie die SPS selbst.

Welche Gründe gibt es, dies dennoch zu fordern?

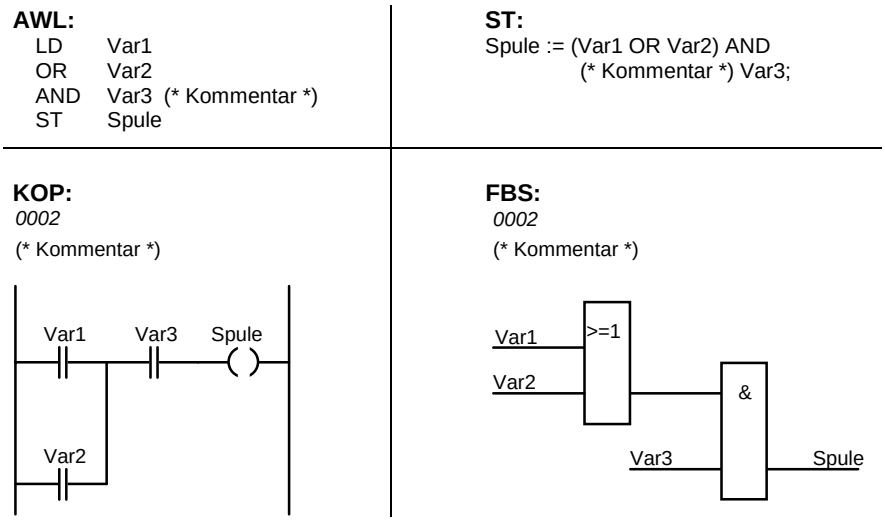
Motivation für Querübersetzbarkeit.
Ein wesentlicher Grund für eine Querübersetzbarkeit von Programmteilen sind Ausbildungsgrad und Tätigkeitsbereich von Technikern und Ingenieuren. Sie werden branchenspezifisch in unterschiedlichen Programmiersprachen geschult, was die notwendige Zusammenarbeit oft erschwert.

Die Automobilbranche in den USA bevorzugt Kontaktplan, im europäischen Bereich ist AWL stärker vertreten. Anlagenbauer ziehen den funktionsstrukturierten Aufbau von FBS vor. Ein Informatiker hat sicherlich keinerlei Schwierigkeiten mit ST. Also für jeden Geschmack etwas und doch von jedem bearbeitbar?

Es gibt Aufgaben, die sich für bestimmte Sprachen besser eignen. Ein Speicherverwaltungssystem läßt sich sicherlich besser in AWL oder ST lösen als in Kontaktplan. Eine Förderbandsteuerung liest sich in Kontaktplan übersichtlicher als in ST. AS ist die geeignete Sprache für eine Schrittkettensteuerung.

In vielen Fällen ist die Entscheidung für eine bestimmte bzw. einzige Programmiersprache nicht mehr so leicht. Oft werden sogar dieselben Programmstücke von verschiedenen Anwendern benötigt.

Ein SPS-Hersteller liefert beispielsweise in AWL geschriebene POEs, um dem Benutzer die Ansteuerung der SPS-Peripherie zu erleichtern. Beim Anwender kann es sich um einen Förderbandhersteller handeln, der die SPS zur Kontrolle und Steuerung von Endschaltern und Motoren benutzt – dabei bevorzugt in KOP arbeitet. Er paßt die AWL-Quellen in KOP-Darstellung an seine Erfordernisse an, um dann das Förderband dem Anlagenbauer zu liefern. Dieser erstellt seine gesamte Anlage wiederum in FBS und benötigt dabei auch die Ansteuerungsprogramme für seine vollständige und einheitliche Dokumentation.



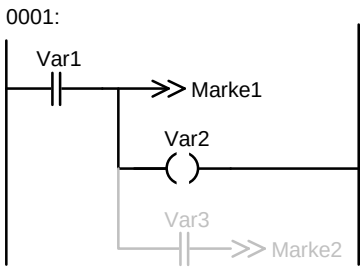
Bsp. 7.2. Beispiel für eine Querübersetzung in vier verschiedenen Programmiersprachen der IEC 61131-3

Unterschiedlicher Ansatz der grafischen und textuellen Sprachen.
Eine Schwierigkeit bei der Querübersetzung liegt in Art und Sichtweise der Werteberechnung. KOP und FBS kommen aus der Binär- bzw. Analogwertverarbeitung: es „fließt Strom“ oder: Werte werden ungetaktet und „parallel“ weitergegeben. Die textuellen Sprachen AWL und ST sind dagegen prozedurale Sprachen, d.h. ein Befehl wird nach dem anderen ausgeführt.

Dies zeigt sich besonders bei näherer Betrachtung der Netzwerk-Auswertung aus Abschn. 4.4.4. Dazu zeigt Bsp. 7.3 ein Beispiel in Kontaktplan (FBS entsprechend).

```
LD    Var1
JMPC  Marke1
ST    Var2
AND   Var3
JMPC  Marke2
```

a) AWL



b) KOP

Bsp. 7.3. Sequentielle Ausführung eines Programmstücks in AWL und parallele Netzwerk-Auswertung in KOP stehen im Widerspruch zu der Forderung nach einfacher Querübersetzbarkeit.

Nach der Auswerteregel für grafische Sprachen (Abschn. 4.4.4) erhält Var2 in allen Fällen einen Wert zugewiesen. Wird das Netzwerk direkt nach AWL abgebildet, erhält Var2 nur eine Wertzuweisung, wenn Var1 = FALSE. Ansonsten müßte bei der Querübersetzung eine Umordnung der Elemente erfolgen (alle ST vor bedingtem JMP oder CAL); damit ändert sich aber bei einer erneuten Querübersetzung nach KOP das grafische Aussehen des Netzwerks.

Sollen in KOP Netzwerke aus AWL, ähnlich der Erweiterung von Var3 in Bsp. 7.3, aufgebaut werden, ist mit der Auswerteregel nicht mehr entscheidbar, ob nach Marke1 oder Marke2 gesprungen werden soll, wenn Var1 und Var3 TRUE-Werte besitzen.

Die Problematik der Querübersetzbarkeit lösen viele Programmiersysteme (soweit sie überhaupt diese Funktionalität besitzen), indem in einem Netzwerk:

- nach einer Zuweisung keine weitere Verknüpfung erlaubt ist,
- der Verwendungsteil eines grafischen Netzwerks (Bsp. 4.38) von oben nach unten berechnet wird und bei einem Befehl der Ausführungssteuerung verlassen wird.

Unterschiede in den Sprachen beeinflussen die Querübersetzbarkeit.
Es läßt sich nicht jede Sprache in eine andere überführen. Da AS eine Sprache für die Strukturierung darstellt, die sich zwar der anderen Sprachen bedient, vom Aufbau und Ablauf aber völlig anders konzipiert wurde, wird im weiteren allein die Querübersetzbarkeit zwischen AWL, ST, KOP und FBS betrachtet.

Einschränkungen bei KOP / FBS.
Sprünge lassen sich zwar über Marken realisieren, widersprechen aber der Idee eines „parallel“ schaltenden Netzwerks. Manche Aufgaben wie Funktionen zur Betriebsmittelverwaltung (Stackoperationen,...) lassen sich nur sehr kompliziert und unleserlich programmieren.

Konstrukte wie CASE-, FOR-, WHILE- oder REPEAT- Anweisungen sind nicht vorgesehen und müßten über Standard-Funktionen wie EQ oder aufwendige Netzwerk-Anordnungen realisiert werden.

Beide Sprachen lassen im Gegensatz zu ST nur einfache Ausdrücke als Index-bezeichner von Feldern zu.

KOP wurde zur Verarbeitung von booleschen Signalen (TRUE und FALSE) definiert. Alle anderen Datentypen wie Integer können zwar über Funktionen und FBs verarbeitet werden, es muß aber mindestens ein Ein- und Ausgang vom Typ BOOL sein und mit der „stromführenden Leitung“ verbunden sein. Dies kann das Programm unübersichtlich machen. Für diese nicht-boolesche Werteverarbeitung ist eine Darstellung in FBS günstiger.

Nicht jedes textuelle Element besitzt eine Entsprechung in der grafischer Darstellung. Beispielsweise fehlt in KOP und FBS ein Teil der Negativ-Modifizierer von AWL. Möglich sind zwar JMP, JMPC, RET oder RETC, doch fehlen JMPCN und RETCN. Dies muß über weitere Verknüpfungen durch den Anwender oder zusätzliche (ungenormte) Grafikelemente des Programmiersystems formuliert werden.

Einschränkungen bei AWL / ST.
Der in KOP und FBS verwendete Begriff des Netzwerks findet sich in den textuellen Sprachen nicht wieder.

In Kontaktplan lassen sich Attribute wie Pufferung von Variablen (M, SM, RM) oder Flankenerkennung (P, N) direkt durch grafische Elemente darstellen. Diese Attributdarstellung im Anweisungsteil durchbricht das Konzept der sonst strikten Angabe der Variablen-Eigenschaften im Deklarationsteil. Hierfür gibt es in den textuellen Sprachen keine Anweisung.

Die Verwendung von EN / ENO birgt eine weitere Schwierigkeit, da in AWL oder ST kein entsprechendes Sprachelement vorhanden ist. Somit muß jede in einer textuellen Sprache entwickelte benutzerdefinierte Funktion eine Auswertung und Zuweisung dieser Variablen vornehmen, um in den grafischen Sprachen darstellbar zu sein. Verwendet man EN/ENO in den Grafiksprachen und verzichtet darauf in den textuellen Sprachen, sind zwei Versionen von Standard-Funktionen notwendig (mit und ohne EN/ENO Behandlung), siehe auch Abschn. 2.5.2.

Querübersetzbarkeit AWL / ST.
Die Hochsprache (ST) läßt sich leichter und effizienter in eine Assemblersprache (AWL) transformieren als umgekehrt. Beide Sprachen besitzen jedoch Einschränkungen bzgl. der gegenseitigen Querübersetzbarkeit, von denen je eine exemplarisch erwähnt wird:

ST kennt keine Sprünge, während diese in AWL („Assembler“) zur Programmierung von Schleifen u.ä. benutzt werden müssen. Dies schränkt die Umsetzung mancher AWL-Programme ein.

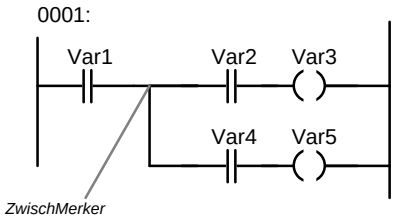
AWL kennt im Gegensatz zu ST nur einfache Ausdrücke für Indexbezeichner von Feldern und Aufrufparametern von Funktionen und FBs.

Volle Querübersetzbarkeit nur durch Zusatzinformation erreichbar.
Erlaubt ein KOP-System innerhalb eines Netzwerks mehrere Spulen und/oder bedingte Anweisungen (in FBS die mehrfache Weiterverarbeitung einer ausgehenden Linie) mit **unterschiedlichen** Werten (siehe dazu Bsp. 7.4), müssen vom Querübersetzer Hilfsvariablen eingefügt werden. Weiterhin ist sicherzustellen, daß im umgekehrten Fall aus den AWL-Anweisungen (entsprechend auch ST) von Bsp. 7.4 nicht mehrere KOP-Netzwerke aufgebaut werden.

Geringfügige Modifikationen des AWL-Programms können daher große Änderungen am querübersetzten KOP-Netzwerk auslösen.

```
LD    Var1
ST    ZwischMerker
AND   Var2
ST    Var3
LD    ZwischMerker
AND   Var4
ST    Var5
```

a) AWL

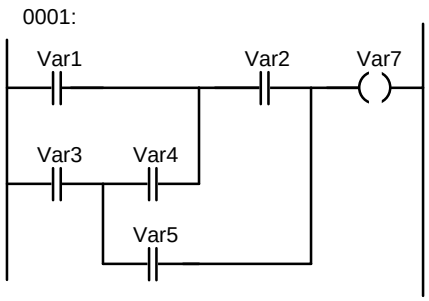


b) KOP

Bsp. 7.4. Werden innerhalb eines Netzwerks Spulen und bedingte Anweisungen mit unterschiedlichen Wert-Zuweisungen verwendet, ist keine direkte Querübersetzbarkeit möglich. Dazu wird z.B. in AWL die Hilfsvariable ZwischMerker notwendig.

```
LD    Var1
OR (  Var3
ST    ZwischMerker   (* meist unzulässig *)
AND   Var4
      )
AND   Var2
OR (  ZwischMerker
AND   Var5
      )
ST    Var7
```

a) AWL



b) KOP

Bsp. 7.5. Grenzen der Querübersetzbarkeit entstehen durch komplizierte Klammer-Konstrukte und Zwischenmerker bei der Übersetzung in eine textuelle Sprache.

Das grafisch übersichtliche KOP-Netzwerk aus Bsp. 7.5 ist nicht direkt, d.h. mit einfachen Befehlen in AWL darstellbar. Es sind dazu eine Hilfsvariable (wie ZwischMerker) notwendig, die sich den Wert zwischen Var3 und Var4 „merkt“, und mehrere Klammerausdrücke. Die Abspeicherung einer Variablen mitten in einem Klammerausdruck wird von der IEC 61131-3 zwar nicht explizit ausgeschlossen, sollte aber vermieden werden, da auch vergleichbare Operatoren wie der bedingte Sprung nicht in Klammerausdrücken verwendet werden dürfen.

Es gibt Programmiersysteme, die die Grafik-Elemente aufgrund ihrer logischen Zusammengehörigkeit automatisch (optimiert) plazieren.

Andere Systeme erlauben eine individuelle Anordnung durch den Anwender. Diese topografische Information muß in AWL oder ST mit Hilfe von Kommentarinformation gespeichert werden, soll später die gleiche Grafik aufgebaut werden. Das Mitführen solcher Topografie-Daten ist einem AWL/ST-Anwender nicht zumutbar. Deshalb besitzen die meisten Programmiersysteme die Plazierungsinformation nur in der internen Datenhaltung. Sie geht bei einer Querübersetzung nach AWL oder ST verloren. Somit gibt es keinen Weg von den textuellen Sprachen zurück zur Grafik, ohne daß sich die Struktur der Netzwerke mehr oder weniger stark verändert.

Gütekriterien für die Querübersetzbarkeit.
Wie den vorausgegangenen Ausführungen zu entnehmen ist, kommt es bei der Querübersetzbarkeit nicht auf komplette Abbildungsvorschriften an, die im strengen Sinn nicht zu erreichen sind.

Die Güte eines Programmiersystems zeigt sich eher daran, inwieweit es folgende Bedingungen erfüllt:

1) Die Umsetzungsregeln sollten so einfach und nachvollziehbar sein, daß der Anwender stets die Umsetzung in eine andere Sprache nachvollziehen kann.
2) Eine Umsetzung darf zu keinem anderen Programmaufbau führen (nur lokale Änderungen; Einheiten müssen zusammenbleiben).
3) Durch Querübersetzung darf es zu keiner Laufzeitänderung des Programms kommen.
4) Die Umsetzung darf keine Seiteneffekte mit sich bringen (andere Programmteile werden beeinflußt).
5) Semantische Mehrdeutigkeiten wie in Bsp. 7.3 müssen geklärt sein.

Aufgrund der nicht einfachen Realisierbarkeit der Querübersetzbarkeit verzichten viele Programmiersysteme auf diese Eigenschaft.

7.4.2 Sprachunabhängigkeit aufgerufener POEs

Ein wesentliches Element der IEC 61131-3 ist die POE als eigenständige Einheit. Zur Programmierung eines Aufrufs dieser POE muß lediglich deren äußere Schnittstelle bekannt sein, es sind keine Kenntnisse über den Code erforderlich. So sollte es auch für den Top-Down Entwurf eines Projekts möglich sein, zuerst alle Schnittstellen der POEs festzulegen und erst danach den Code zu programmieren.

Diese externe Schnittstelle wird üblicherweise vom Programmiersystem nach einer POE-Definition projektglobal zur Verfügung gestellt und besteht:

- bei Funktionen aus: Funktionsname und -typ, Namen und Datentypen der VAR_INPUT- Parameter,
- bei FBs aus: FB-Name, Namen und Datentypen der VAR_INPUT-, VAR_OUTPUT- und VAR_IN_OUT- sowie der EXTERNAL- Parameter.

Eine POE kann andere Funktionen und FBs aufrufen, ohne die Sprache zu kennen, in der diese POE programmiert wurde. Somit muß das Programmiersystem auch nicht für jede Sprache einen eigenen Satz von Standard-Funktionen und -FBs zur Verfügung stellen.

Diese Methode der „Blackbox“ kann auch für IEC 61131-3- fremde Sprachen genutzt werden, d.h. der Aufrufer besitzt bis auf die externe Schnittstelle keinerlei Kenntnisse über die Realisierung des aufgerufenen Bausteins. Sind die Aufrufschnittstellen des IEC 61131-3 Programmiersystems und die eines C-Compilers verträglich, kann ebenso ein C-Unterprogramm aufgerufen werden.

Die IEC 61131-3 läßt ausdrücklich die Hinzunahme weiterer Programmiersprachen zu.

7.5 Dokumentation

Zur kostengünstigen Pflege von Anwenderprogrammen, sowie zur Erfüllung der heutigen Qualitätsanforderungen wie ISO 9000 gibt es verschiedene Dokumentationsformen:

1) *Querverweisliste*. Eine Tabelle gibt darüber Auskunft, welche symbolische Namen (Bezeichner wie Variablen, Sprung- oder Netzwerknamen, POE-Namen und -Instanzen usw.) in welchen POEs verwendet werden.
2) *Programmstruktur*. Übersicht über die Aufrufhierarchie der POEs. Jede POE erhält dabei eine Liste aller von ihr aufgerufenen POEs. Die Schachtelungstiefe ist je nach System unterschiedlich. Diese Darstellung kann tabellarisch oder grafisch erfolgen.
3) *Zuordnungsliste* (auch *Verdrahtungsliste*). Diese Liste enthält die (symbolisch) deklarierten physikalischen Ein- und Ausgänge des Programms (Direkt dargestellte und Symbolische Variablen) und weist sie Hardware-Adressen zu (E/A-Peripherie).
4) *Belegungsliste*. Tabellenartige Übersicht über die Verwendung der E/A-Peripherie durch das Programm, nach SPS-Adressen sortiert. Mit Hilfe der Belegungsliste lassen sich bei einer Programmerweiterung schnell freie Plätze der E/A-Peripherie finden sowie die Zuordnung von Programm und SPS maschinennah dokumentieren.
5) *Anlagen-Dokumentation*. Eine (meist grafische) Beschreibung einer kompletten Anlage. Das einzelne Automatisierungsgerät tritt dabei nur noch als „Blackbox“ auf. Eine Gesamtanlage besteht oft aus mehreren SPSen, Maschinen, Ausgabegeräten usw. Die Beschreibung dieser Ebene ergänzt die Dokumentation nach oben hin. Sie enthält die topologische Anordnung der einzelnen SPS-Stationen und anderer Komponenten sowie deren Verdrahtung untereinander. Dafür wird oft auf ein marktübliches Zeichenprogramm oder CAD-System zurückgegriffen.
6) *Programmdokumentation*. Programmquellen der POEs, die vom Programmiersystem erstellt werden. Die Druckausgaben sollten möglichst die gleiche Struktur und Inhalte der POEs zeigen wie am Bildschirm.
7) *Konfiguration*. Die Konfiguration – definiert in der IEC 61131-3 – beschreibt, welche Programme auf den SPS-Ressourcen mit welchen Eigenschaften ablaufen.

Diese Möglichkeiten sind zwar durch die IEC 61131-3 nicht vorgeschrieben, haben sich aber im Laufe der Jahre zur Beschreibung des Programms (oft einheitlich) entwickelt.

7.5.1 Querverweisliste

Diese Liste ist auch unter dem englischen Namen „Cross Reference List“ in der PC-Welt bekannt. Sie beinhaltet :

- alle in einem Projekt verwendeten *symbolischen Namen*,
- den *Datentyp* von Variablen, *FB-Typ* bei Instanznamen, *Funktionstypen* bei Funktionen sowie besondere *Attribute* wie RETAIN, CONSTANT oder READ_ONLY und *Variablenart* (VAR_INPUT, ...),
- den *POE-Namen* mit *Typ*, sowie die *Zeilennummer*, in der der symbolische Name (ggfs. instanziiert) verwendet wird,
- die *Zugriffsart* an diesen Programmstellen,
- die *POE*, in der die Deklaration vorgenommen wird.

Manche Systeme stellen diese Symbole nur POE-weise zusammen oder beschränken die Liste auf globale/externe Daten.

Die Querverweisliste eignet sich sehr gut, um z.B. bei der Fehlersuche das Auffinden von Programmstellen zu erleichtern.

Die Sortierkriterien sind vielfältig. Die wohl wichtigste Darstellung ist eine alphabetische Auflistung der Symbole. Weiterhin kann z.B. nach Symboltyp (Ein-/Ausgangsvariable, Funktionsname, FB-Name,...) oder Datentyp (wie Byte, INT, usw.) sortiert werden.

Symbolname	POE-Name	POE-Typ	Zeilen-Nr	Zugriffs-art	Datentyp Fun-Typ FB-Typ	Attribut	Var Art
Temp_Sensor	SOND.POE	Prog	10	Deklarat.	INT	Retain	GLOBAL
	SOND.POE	Prog	62	Schreiben			EXTERN.
	STEUE.POE	FB	58	Lesen			EXTERN.
. . .							

Bsp. 7.6. Ausschnitt aus einer Querverweisliste, sortiert nach Symbolnamen

7.5.2 Zuordnungsliste (Verdrahtungsliste)

Die Zuordnungsliste stellt alle Variablen, denen im PROGRAM oder in der KONFIGURATION physikalische E/A-Peripherie zugewiesen werden, und die Zugriffspfade zusammen.

Hier sind oft Hilfsmittel erforderlich, mit denen diese symbolischen Variablen auf andere physikalische Adressen gelegt werden können (*Umverdrahtung*). Dies wird dann erforderlich, wenn das Programm in eine andere Umgebung (z.B. eine andere Anlage mit anderen Anschlüssen) portiert wird.

Symbolname	E/A-Peripherie
Temp_Sensor_1	%IB0
Temp_Sensor_2	%IB2
Temp_Steuer	%QB4
Temp_Stand	%MB1
. . .	

Bsp. 7.7. Ausschnitt aus einer Zuordnungsliste (Verdrahtungsliste)

7.5.3 Kommentierbarkeit

Ein wesentlicher Bestandteil der Dokumentation ist erklärender Text in Form von Kommentar. In Programmquellen kann er an folgenden Stellen, siehe auch Bsp. 7.2, einfließen:

- Kommentar kann in ST überall dort eingefügt werden, wo Leerzeichen erlaubt sind. In AWL kann er am Zeilenende stehen.
- Die IEC 61131-3 macht keine Aussagen zu Kommentaren in den grafischen Sprachen. Es hat sich aber bereits in der Vergangenheit gezeigt, daß Netzwerkkommentare, die den Aufbau und das Verhalten des folgenden Netzwerks beschreiben, eine wichtige Dokumentationsstütze darstellen.

Manche Programmiersysteme verhindern über Menü-Einstellungen das (versehentliche) Überschreiben des Programms, um z.B. nur Kommentaränderungen zuzulassen.

7.6 Projektverwaltung

Die Projektverwaltung hat die Aufgabe, sämtliche zur Programmierung eines Projekts notwendigen Informationen konsistent zu verwalten. Dazu zählen:

- *Quell- Information,*
  - die Quellen der erstellten POEs mit
    - Typ-Definitionen, Deklarationen von globalen Daten, Beschreibung von VAR_ACCESS Variablen, ... ,
    - Aufruf-Schnittstellenbeschreibung aller POEs zur Darstellung und zur Durchführung der erforderlichen Schnittstellenprüfungen,
  - Versionskontroll-Systeme zur Datenhaltung von Quellen,
  - Zugriffs-Beschränkungen (Paßwort-Ebenen) zu:
    - POEs ändern,

    - Programme ausgeben,
    - Bibliotheken.
- *Objekt- Information*,
    - übersetzte Quellen (Zwischencode, Objekte, ausführbare Dateien),
    - Projekterstellungs-Prozeduren (Aufrufabhängigkeiten, Erstellungs- und Änderungsinformation zur Steuerung von zeitlichen Übersetzungs-abhängigkeiten wie MAKE, Batches, ...),
    - Bibliotheken (Standard-Funktionen, Standard-FBs, Hersteller-FBs, Kommunikations-FBs).
- *On-line- Information*,
    - Dateien zur Parametrierung (Rezeptur) des Projekts,
    - Geräte- und Konfigurations-Beschreibungen (SPS-Gerätebeschreibung, Peripheriebaugruppen, ...),
    - Spezielle Information für den „on-line“-Test (Symbole, Haltepunkte, ...),
    - Kommunikations-Information (herstellerspezifische Übertragungsproto-kolle, Schnittstellen, ...).
- *Dokumentation* (wie Querverweisliste, Belegungsliste, Programmstruktur,...).

Die Projektverwaltung ordnet und archiviert alle anfallenden Daten. Warum reicht dazu in der Regel kein reiner Dateimanager (z.B. Windows- Explorer) aus?

Die POEs besitzen Abhängigkeiten. Die gleiche POE kann in einem Projekt von mehreren Programmen verwendet werden, obwohl sie evtl. nur einmal als Datei vorhanden ist. Funktionsnamen und FB-Typen besitzen projektglobale Gültigkeit (FB-Instanzen, soweit sie nicht GLOBAL deklariert sind, nur POE- weit!). Beim POE-Aufruf (CAL) benötigen die Compiler und Editoren Schnittstellen-Information über die aufgerufene POE (Parametertypen, ...). Unterschiedliche Informationen wie Binär-Dateien und Konfigurationsdateien müssen zusammen-gebracht werden. Um einen ständigen Neuaufbau solcher Informationspakete zu vermeiden, kann eine Projektverwaltung erzeugte Informationen aufnehmen und bei Bedarf anderen Komponenten eines Programmiersystems zurückliefern.

Abb. 7.2 zeigt das Festplattenverzeichnis (D:), das sich in Unterverzeichnisse aufgliedert. Um die **Zusammenhänge** eines Projekts zu visualisieren, müssen zusätzlich Aufruf- und Umgebungsabhängigkeiten ausgewertet werden, damit diese Relationen, siehe Abb. 7.3, gezeigt werden können.

Das Programmiersystem sollte dabei helfen, den Aufbau eines Projekts zu ver-stehen. Ein Projekt kann aus mehreren Konfigurationen (SPSen) bestehen. Die Beschreibung einer solchen Konfiguration ist in Kap. 6 zu finden. Eine Konfiguration kann mehrere Ressourcen (CPUs) verwenden. Jede Ressource besitzt eine Spezifikation ihrer Hardware (Resource1 Datei). Auf der einzelnen Ressource können mehrere Programme (Programm1, ...) ablaufen, die ihrerseits durch POEs und deren Aufrufe beschrieben sind. Somit kann es vorkommen, daß zwei POE-Instanzen in Abb. 7.3, die in unterschiedlichen Programmen verwendet werden, durch dieselbe POE beschrieben werden, die nur einmal auf der Festplatte oder Datenbank hinterlegt ist.

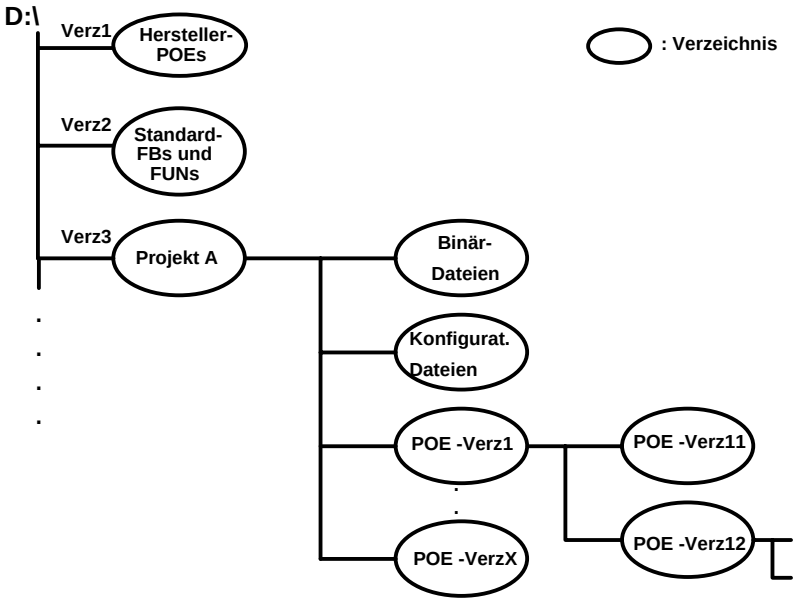


Abb. 7.2. Datenhaltung für Quellen und Bibliotheken auf der Festplatte (hier D:). Das Programmiersystem sollte eine Verzeichnisstruktur zulassen, die der Anwender entsprechend den Projekterfordernissen frei wählen kann; z.B. Verwendung des MS-DOS Dateisystems.

Zur Informationssuche können unterschiedliche Regeln vereinbart sein. Soll z.B. das auf der Resource1 ablaufende Programm1 übersetzt werden, muß das System für einen Übersetzungs- und Linkerlauf alle dazugehörenden Anwender-, System- und Hersteller- POEs zusammensuchen. Ist ein aufgerufener POE-Name doppelt vorhanden, ist zu entscheiden, ob eine Überdeckung (z.B. Anwender-POE geht vor Hersteller-POE) oder Fehlermeldung erfolgt; die IEC macht hierüber keine Aussage.

Aus Gründen der Übersetzungszeit ist es wünschenswert, wenn bei Änderungen nur die zur Konfiguration, Ressource oder zum Programmzweig gehörenden Teile übersetzt werden. Wurde beispielsweise nur im Programm1 geändert, müssen nicht die übrigen Programme (oder andere Ressourcen), die diese POE nicht benötigen, übersetzt werden.

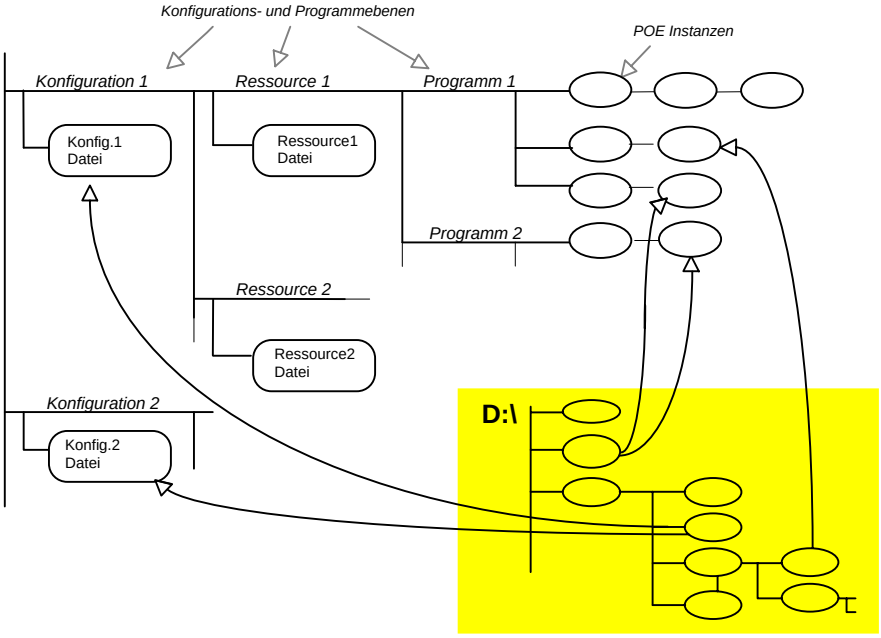


Abb. 7.3. Die logische Struktur von Projekten besteht aus Konfigurationen, Ressourcen und POE-Aufrufhierarchien. Da diese Struktur nicht mit der physikalischen Verzeichnisstruktur von Abb. 7.2 übereinstimmt, sind weitere Visualisierungshilfen wünschenswert.

Zu den Aufgaben einer Projektverwaltung gehören somit:

- Eintragen neu erstellter Dateien (Quellen, Fehlermeldungsdateien, Schnittstellenbeschreibungen),
- Einkopieren von projektfremden Dateien (aus anderen Projekten),
- Anzeigen aller vorhandener POEs,
- Umbenennen, Löschen von POEs,
- Aufbau einer Informations-Struktur, die den Projektaufbau (Aufrufhierarchie, Zusammenhänge) für den Anwender transparent macht,
- Verwaltung und Lieferung aller Informationen (POEs, Schnittstellen, Aufrufhierarchie, Binärdateien usw.), die das restliche Programmiersystem zur Erstellung eines Projekts benötigt.

Manche IEC 61131-3 Programmiersysteme hinterlegen die Daten in einzelnen Dateien (z.B. eine pro POE) auf der Festplatte, manche in Datenbanken. Diese Daten dürfen nur über die Projektverwaltung manipuliert werden. Um Anfragen an die Projektverwaltung (z.B. vom Linker) möglichst schnell beantworten zu können, kann ein Teil der Information bereits vorverarbeitet werden, ohne bei jeder Anfrage das gesamte Dateisystem neu untersuchen zu müssen.

7.7 Test&Inbetriebnahme- Funktionen

Die Programmentwicklung erfolgt im ersten Schritt auf einem PC ohne die SPS. Sind die wichtigsten Teile erstellt, erfolgt erfahrungsgemäß die weitere Arbeit direkt mit der SPS in der realen Umgebung. Dabei werden typischerweise folgende Aufgaben durchgeführt:

- Übertragung des gesamten Projekts oder einzelner POEs (nach Änderungen) auf die Steuerung,
- Übertragung des Projekts von der Steuerung auf den PC,
- Ändern des Programms in der SPS (im Zustand „RUN“ oder „STOP“),
- Starten und Stoppen der SPS,
- Anzeigen der Werte bestimmter Variablen (Status),
- Direktes Setzen der E/A-Peripherie oder Variablen (Forcing),
- „Abschalten“ der physikalischen SPS-Ausgänge, um unsichere Anlagenzustände während des Tests zu vermeiden. Die Programmabarbeitung und die Wertzuweisung an direkte Variable erfolgt wie im normalen Betrieb. Zusätzliche Software oder Hardware sorgt aber dafür, daß die errechneten Werte nicht auf die physikalische Ausgangsperipherie geschrieben werden,
- SPS-Systemdaten, Kommunikation- und Netzwerk-Information auslesen,
- Programmsteuerung (Haltepunkte, Einzelschritt, ...).

Diese Funktionalitäten werden in unterschiedlichem Umfang und Ausprägung von den Programmiersystemen angeboten. Die IEC 61131-3 macht darüber (noch) keine Aussagen.

7.7.1 Programmtransfer

Nachdem (oder während) eine POE mit einem Editor erstellt und syntaktisch überprüft wurde, erfolgt die Erstellung des SPS-spezifischen Codes. Dies kann Maschinencode sein, der direkt auf der SPS ausgeführt wird, oder Anweisungen, die auf der SPS durch einen Interpreter abgearbeitet werden. Dieser Objektcode wird vom Programmiersystem mit den übrigen POEs des Programms zusammengepackt (Link-Vorgang).

Es fehlen nun noch:

- die Verbindung des ausführbaren Codes mit der Beschreibung der *CONFIGURATION* (Task-Zuweisung),
- eine evtl. Anpassung der Hardware-Adressen an die tatsächlichen physikalischen Adressen der SPS,
- spezielle Parametrierung des SPS-Betriebssystems.

Diese Aufgaben können durch ein *Systemkonfigurator* genanntes Softwareprogramm durchgeführt werden. Es gibt auch Implementierungen, bei denen das SPS- Betriebssystem einen Teil dieser Aufgaben übernimmt.

Das Gesamtpaket muß nun vom PC auf die Steuerung gebracht werden. Ein als *Kommunikationsmanager* bezeichnetes Programm sorgt dafür, daß eine physikalische und logische Verbindung zu der SPS aufgebaut wird (z.B. ein firmenspezifisches Protokoll über die COM1-Schnittstelle oder eine Feldbusverbindung). Diese Software führt eine Reihe von Überprüfungen aus, die zum Teil für den Anwender nicht sichtbar sind, wie:

- ist die Kontaktaufnahme mit der SPS erfolgreich?
- befindet sich die SPS in einem Zustand, um neue Programme aufnehmen zu können? Kann die SPS in diesen Zustand gebracht werden?
- Entspricht die Hardware-Konfiguration der SPS den im Programm vorgegebenen Bedingungen?
- Besteht eine Zugangsberechtigung des aktuellen Anwenders zu dieser SPS?
- Visualisierung des aktuellen Zustands der Kommunikation.

Dann erfolgt die Übertragung des Programms sowie der zum Ablauf relevanten Daten. Das SPS-Programm kann starten (Neustart).

7.7.2 Online-Änderung des Programms

Sollen Bausteine eines aktiv laufenden SPS-Programms (SPS in „RUN“) geändert werden, kann dies erfolgen,

- indem die POE im PC geändert und das gesamte Programm vom Programmiersystem völlig neu erstellt wird. Anschließend wird alles komplett in die SPS geladen. Soll dies im laufenden Betrieb erfolgen, muß ein zweiter Speicherbereich zur Verfügung stehen, auf den nach erfolgtem Transfer umgeschaltet wird, da die Transferzeit i.a. zu lange dauert, um das zyklische Programm für diese Zeit anzuhalten.
- indem die POE im PC geändert und nur dieser Teil auf die SPS transferiert wird. Dies erfordert aber eine Bausteinverwaltung auf der SPS, die diese POE entgegennimmt und nach erfolgter Übertragung den alten Baustein aus- und den neuen einhängt. Da die nun ungültige POE aufgrund der unterschiedlichen Größen nicht vollständig überschrieben werden kann, erfolgt zu gegebener Zeit

ein „Zusammenschieben“ oder „Komprimieren“ aller gültigen POEs im Speicher der SPS (engl.: „garbage collection“).
- indem nur einzelne Netzwerke (AS, KOP oder FBS) oder Anweisungen (AS, ST und AWL) ausgetauscht werden. Dies ist nur dann möglich, wenn damit die anderen Teile der POE nicht beeinflußt werden. So dürfen sich beispielsweise die Sprungmarken in den anderen Netzwerken nicht verschieben, wenn das SPS-Betriebssystem keine entsprechende Sprungmarkenverwaltung beinhaltet.

7.7.3 Fernbedienung: Start und Stop der SPS

Die Hardware einer SPS besitzt üblicherweise einen Schalter zum Starten und Stoppen. Diese Funktionalität kann auch oft vom Programmiersystem gesteuert werden.

Die IEC 61131-3 definiert dazu verschiedene Anlaufverhalten der SPS (siehe auch Abschn. 3.5.3):

1) *Kaltstart (*auch: *Neustart)*. Die SPS startet ihr Programmsystem ohne jegliches Vergangenheitsgedächtnis. Dies ist insbesondere nach dem Laden des Programms der Fall.
2) *Warmstart (*auch: *Wiederanlauf)*. Das Programm setzt seine Abarbeitung nach einem Spannungsausfall an der unterbrochenen Stelle (z.B. innerhalb eines FBs) fort. Alle mit dem Attribut „RETAIN“ versehenen Variablen sind mit dem Wert belegt, den sie vor der Unterbrechung besaßen.
3) *Warmstart am Programmstart.* Das Programm startet ebenfalls mit dem Wert der Retain-Variablen, jedoch nicht an der unterbrochenen Stelle, sondern am Programm-Anfang, da die parametrierte maximale Unterbrechungszeit überschritten wurde.

Vom Anwender kann in der Regel nur ein Kaltstart veranlaßt werden, die Warmstarts erfolgen automatisch bei Spannungswiederkehr.

Weitere Einstellungsmöglichkeiten eines Programmiersystems sind:

- Anhalten der SPS (mit den aktuellen oder „sicheren“ Werten der physikalischen Ausgänge),
- Löschen von Speicherbereichen, um unkontrolliertes Anlaufen zu vermeiden,
- Einstellen spezieller Betriebssystemzustände wie Testmodus, max. Zykluszeit... .

7.7.4 Variablen- und Programm- Status

Der wichtigste Test während der SPS-Test&Inbetriebnahme ist die Beobachtung von Variablen („Merker“) und E/A-Peripherie. Die Darstellung erfolgt idealerweise in einer benutzerwählbaren Form, wie in Bsp. 7.8 zusammengefaßt.

Variable	**Typ**	**Wert**
Spule	BOOL	1
Var1	BOOL	1
Var2	BOOL	0
Var3	BOOL	1

a) Variablen-Liste

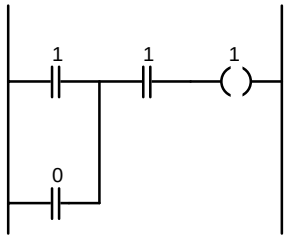
```
LD    Var1    1
OR    Var2    0
AND   Var3    1
ST    Spule   1
```

b) AWL (Variablenstatus)

```
Spule := Var1 OR Var2 AND Var3
         1      1      0       1
```

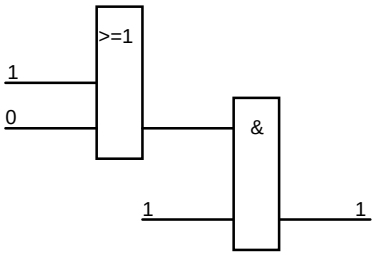
c) ST (Variablenstatus)

0002:
(* Kommentar *)



d) KOP (Variablenstatus)

0002:
(* Kommentar *)



e) FBS (Variablenstatus)

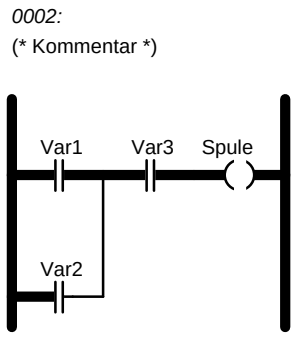
```
LD    Var1    1
OR    Var2    1
AND   Var3    1
ST    Spule   1
```

f) AWL (Aktuelles Ergebnis)

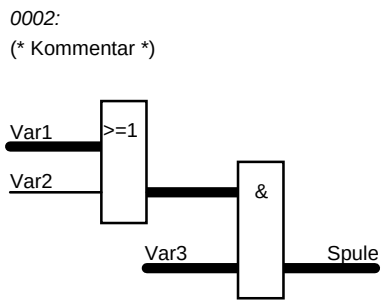
```
Spule := Var1 OR Var2 AND Var3
         1
                1
                       1
                              1
```

g) ST (Ausdruck)

Bsp. 7.8. Das in Bsp. 7.2 aufgeführte Beispielprogramm in einer Momentaufnahme des Status während der Test&Inbetriebnahme-Sitzung in verschiedenen Darstellungsarten. Diese können auch miteinander kombiniert werden (wird fortgesetzt).



h) KOP (Stromflußanzeige)



i) FBS (Programmstatus)

Bsp. 7.8. (Fortsetzung)

Es gibt, je nach Realisierung in der SPS und/oder im Programmiersystem, mehrere Varianten, um einen Überblick über den aktuellen Datenfluß zu erhalten:

1) *Variablenausgabe*: Ausgabe von einer Variablenliste (a). Die darin enthaltenen Variablen werden von der SPS abgefragt und ihre Werte laufend am Bildschirm aktualisiert. Diese Ausgabe wird typischerweise dann verwendet, wenn Werte beobachtet werden, die aus unterschiedlichen Programmteilen stammen. Dabei können strukturierte Variablen auch mit den Werten ihrer Strukturkomponenten angezeigt werden.
2) *Variablenstatus*: Es werden alle Variablen eines bestimmten Code-Abschnitts angezeigt. In Bsp. 7.8 werden die Werte von Var1, Var2, Var3 und Spule in einem separaten Fenster (a) oder direkt in der Grafik angezeigt (b-e).
3) *Programmstatus* (auch: Power Flow; Stromflußanzeige). Statt einzelner Variablenwerte werden die Verknüpfungsergebnisse (wie Aktuelles Ergebnis; siehe Abschn. 4.1.2) ausgegeben (f-i). Dies erfolgt bei den grafischen Sprachen durch die Ausgabe von dicken/dünnen Linien (TRUE/FALSE) bei booleschen Verknüpfungen oder Angabe von Zahlenwerten an den Verbindungslinien. Durch Zwischenwert-Berechnungen des PCs können auch Werte angezeigt werden, die nicht unmittelbar an Variablen geknüpft sind (z.B. der Operator NOT; ODER-Verbindung in KOP).

Die Qualität der erhaltenen Werte ist von der Funktionalität, Geschwindigkeit und den Synchronisationsmöglichkeiten des Programmiersystem und der angekoppelten SPS abhängig. Das „Sammeln“ der Werte auf der SPS kann, abhängig von den Möglichkeiten des Betriebssystems und der Hardware, zu verschiedenen

Zeitpunkten erfolgen:

- synchron,
- asynchron,
- bei Änderung.

Synchroner Status: Die Werteausgabe erfolgt am Ende des Programmzyklus. Die Werte werden somit immer an der gleichen Programmstelle erzeugt (konsistenter Datensatz). Dies ist auch der Augenblick, in dem die Variablen, die mit der E/A-Peripherie verbunden sind, auf die Peripherie geschrieben werden (Austausch des Prozeßabbilds). Im zyklischen Betrieb ist das Programm allerdings wesentlich schneller als die Frequenz der Anfragen, so daß nur in jedem x-ten Zyklus ein Wertepaket zurückgereicht werden kann.

Asynchroner Status: Das Programmiersystem fordert ständig (asynchron zum SPS-Programm) einen neuen Datensatz der angegebenen Variablen an, die auf der SPS zum Zeitpunkt des Eintreffens der Anforderung zusammengestellt werden. Es handelt sich somit um eine Momentaufnahme der spezifizierten Speicherbereiche.

Status bei Änderung: Die Ausgabe eines Werts erfolgt nur bei Änderung (z.B. Flanke) einer Variablen. Dies erfordert in der Regel spezielle Hardwarevorrichtungen (Adreßüberwachung) in der SPS.

Komfortable Systeme bieten zusätzlich Datenanalyse-Tools an. Damit können Variablen in ihrem zeitlichen Verlauf beobachtet werden (Logikanalyse), wie Abb. 7.4 skizziert.

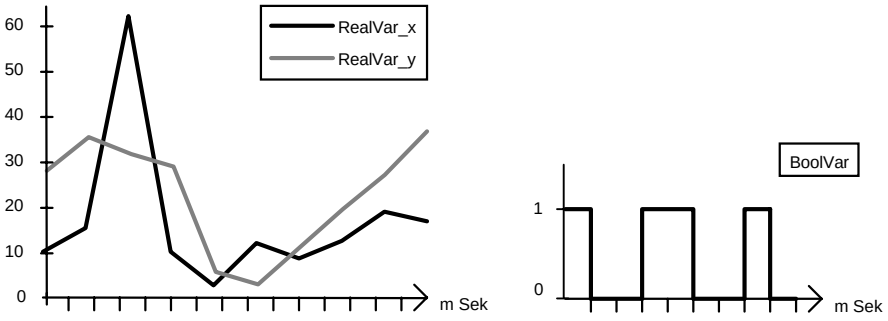


Abb. 7.4. Beispiel einer Datenanalyse zweier Real- und einer Bool-Variablen

Die Anzeige von Variablenwerten in der Programmgrafik erreicht schnell ihre Grenze, wenn Felder oder umfangreiche Strukturen mit ihren Werten angezeigt

werden sollen. Hierfür sind separate Variablenfenster wegen der Fülle der Information geeigneter.

Zu den anzeigbaren Werten zählen nicht nur die Variablenwerte, dazu gehören auch Parameter von Funktionen und FBs.

Derzeit noch selten realisiert ist die Ausgabe des Datenflusses zwischen einzelnen POEs, wie in Abb. 7.5 dargestellt.

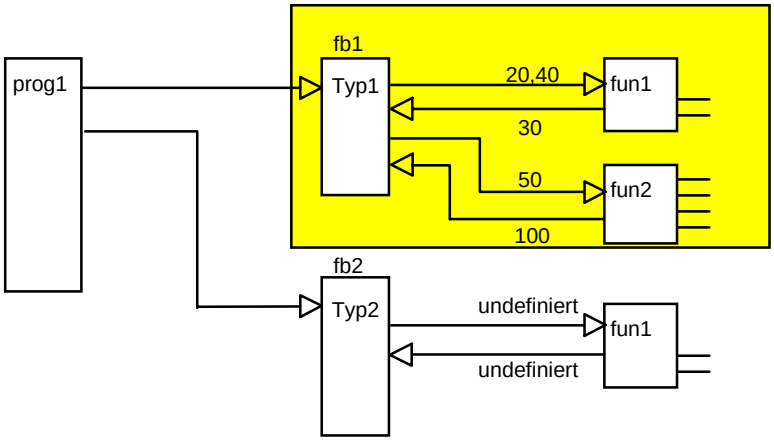


Abb. 7.5. Darstellung der Aufruf-Hierarchie der POEs mit ihren aktuellen Parametern zur Laufzeit des SPS-Programms

Es reicht bei der Anforderung zur Ausgabe von Variablenwerten nicht aus, den Namen der gewünschten POE anzugeben. Mit der alleinigen Angabe von „fun1“ in Abb. 7.5 ist unklar, welche Daten ausgegeben werden sollen, da diese POE zweimal benutzt wird und bedingt durch unterschiedliche Aufrufparameter unterschiedliche Werte liefern wird.

Bei FBs ist die Angabe des POE-Namen nicht ausreichend, da der FB-Typ mehrfach instanziiert sein kann und damit unterschiedliche Instanzdaten besitzt. Auch die zusätzliche Angabe des Instanznamen ist nicht eindeutig, da ein FB-Typ in verschiedenen POEs mit gleichem Instanznamen instanziiert sein kann und damit mehrere Instanzen mit gleichem Namen verwendet werden.

Zur eindeutigen Identifikation ist somit der Aufrufpfad, bei Funktionen sogar die Aufrufstelle, mitanzugeben.

Eine weitere Schwierigkeit liegt darin, daß Funktionen bzw. deren lokale Variablen nur während der Abarbeitung dieser POE existieren, andernfalls undefiniert sind.

7.7.5 Forcing

Um das Programm- und Anlagenverhalten bei der Inbetriebnahme testen zu können, ist das „Zwangssetzen“ bestimmter Speicherbereiche der SPS hilfreich.

Um dem zu testenden Programm einen gewissen Anlagenzustand vorzuspiegeln – damit soll z.B. die Bearbeitung bestimmter Programmteile erzwungen werden – werden über Eingaben am Programmiersystem bestimmte Variablen fest mit Werten belegt und an die SPS übertragen, die diese Werte anstelle der tatsächlichen Werte verwendet.

Lokale Variablen

Variable	**Datentyp**	**POE-Instanz**	**zugewiesener Wert**
Var_Loc	BOOL	FB1	1
Var3	INT	FB1	20
...			

Direkt dargestellte Variablen (WORD)

%IW0: **0 0 1 0 0 0 1 0 0 0 0 0 1 0 1 0** *(Bits 0 bis 15)*
%IW1: **0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0** *(Bits 16 bis 31)*
...

Symbolische Variablen (Integer)

Start Adresse	**Variablen-Name**	**zugewiesener Wert**
%QW20:	Endwert	11.233
...		

Bsp. 7.9. Beispiel für das Forcing von Variablen und E/A-Peripherie (hier boolesche und ganzzahlige Werte). Bei der Namensangabe („POE-Instanz“) muß ggf. die Aufrufhierarchie berücksichtigt werden, bei Funktionen auch die Programmstelle; siehe dazu Abschn. 7.7.4.

Solange die SPS vom Programmiersystem im Zustand „Forcing“ gehalten wird und beispielsweise mit den in Bsp. 7.9 beschriebenen Parameter versorgt wird, erhält das Programm jedesmal den eingegebenen booleschen Wert 1, wenn es %IX0.2 oder %IX0.6 anspricht.

Die Variablen werden beim Erreichen des Programmanfangs (Zyklus) einmal belegt und können dann vom Programm überschrieben werden. In anderen Implementierungsvarianten sorgt das SPS-System dafür, daß kein Überschreiben erfolgt.

7.7.6 Programmtest

Zum Testen von Programmen gibt es die Funktionen Haltepunkt und Einzelschritt, die auch aus dem PC-Bereich bekannt sind:

- *Haltepunkt (Rück-) Setzen*. An einer vom Anwender beschriebenen Stelle soll die Programmausführung gestoppt werden und auf weitere Anweisungen des

Anwenders gewartet werden. Komfortable Systeme bieten hierbei die Möglichkeit mehrstufiger Bedingungen an: „Wenn Funktion_1 aufgerufen, Variable_X gesetzt und Funktion_2 erreicht ist, dann halte bei Befehlszeile 20 des FB4...“.

Auch hier gilt bei IEC-Programmen, daß neben der Angabe der Anweisungszeile der POE-Name bzw. Instanzname nicht ausreicht. Vielmehr muß die Stelle in der Aufruf-Hierarchie genannt werden; siehe Abb. 7.5.

- *Einzelschritt*. Nach dem Erreichen eines Haltepunkts erfolgt die Ausführung des nächsten Befehls bzw. grafischen Elements.

7.7.7 Programmtest in Ablaufsprache

Eine besondere Hilfestellung ist bei AS-strukturierten Programmen gefordert. Je nach Komfort unterstützt das Programmiersystem den Anwender bei folgenden Aufgaben:

- (Rück-) Setzen von Transitionen, um die Weiterschaltung auf den nächsten oder den Verbleib im aktuellen Schritt zu erzwingen.
- Aktivierung von Schritten oder Transitionen, um die Bearbeitung der POE an einer bestimmten Stelle zu beginnen.
- Dauerhaftes Blockieren von Schritten und Transitionen, um die Aktivierung dieser Teile zu vermeiden.
- Setzen und Lesen von AS-relevanten Systemdaten wie Schritt-Merker, Schritt-Zeit, sowie Rückkopplungsvariable.

Es handelt sich hierbei um eine Art Forcing auf der Ebene der Ablaufsprache.

7.8 Datenbausteine und Rezepturen

Eine *Rezeptur* im SPS-Sprachgebrauch stellt eine Sammlung von Datenwerten dar, die unter Beibehaltung des Programms je nach Steuerungsaufgabe der SPS gegen eine andere ausgetauscht werden kann. So lassen sich z.B. in einem Automatisierungsprozeß unterschiedliche Produktionsvarianten steuern, die sich in Parametern wie Länge, Gewicht oder Temperatur unterscheiden. Mit Rezepturen werden bestimmte Prozeß-Daten des Programms im laufenden Betrieb geändert.

Diese Umparametrierung kann durch einen einfachen Austausch der Datensätze durch das Programm selbst erfolgen. In diesem Fall sind die möglichen Datensätze bereits im Speicher der SPS enthalten. Eine andere Möglichkeit besteht darin, den entsprechenden Datensatz während eines geeigneten Zeitpunkts im laufenden Betrieb neu in die SPS zu laden, d.h. einen dort vorhandenen Datensatz gegen den neuen auszutauschen.

Einige SPS-Systeme lösen dies bisher durch sogenannte *Datenbausteine* (DB). Ein solcher Datenbaustein ist projektweit (global) bekannt. Er besteht aus einer Anzahl von Datenwörtern mit Datenformat (Datentypen).

Über die Datenbaustein-Nummer erhält das Programm die Basisadresse des gerade aktiven Datenbausteins. Auf die Daten wird über einen Index (Datenwort-Nummer) zugegriffen. Bei einem DB-Wechsel wird lediglich die Basisadresse auf den neuen Baustein gestellt, d.h. der neue Datenbaustein aktiviert.

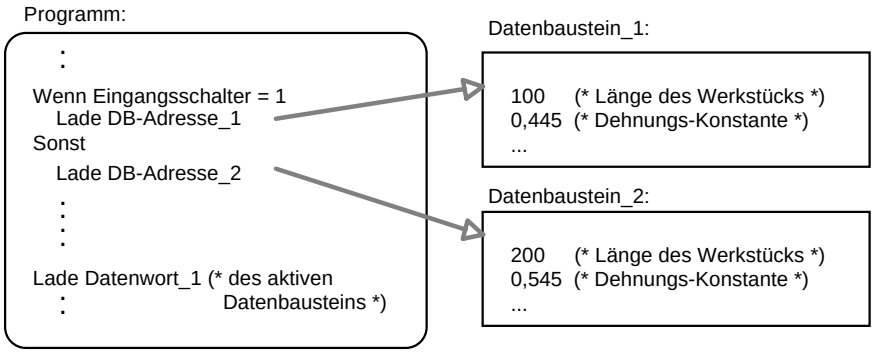


Abb. 7.6. Arbeiten mit Datenbausteinen. Genau *ein* Datenbaustein kann zu einem Zeitpunkt aktiv sein.

In Abb. 7.6 lädt „Lade Datenwort_1“, abhängig von der aktuell eingestellten Datenbaustein-Nummer, entweder 100 oder 200, ohne an dieser Stelle den aktiven Datenbaustein zu kennen.

Ein wichtiger Anwendungsbereich liegt in der Verwendung der DBs als Datenbereiche für (bisherige) Funktionsbausteine: vor Aufruf eines FBs wird der entsprechende Datensatz („Rezeptur“) angewählt, den der FB anschließend als Parametrier- und Arbeitsbereich verwendet.

Zusammengefaßt besitzen Datenbausteine üblicherweise folgende Eigenschaften:

- DBs sind selbständig wie Bausteine ladbar bzw. austauschbar,
- DBs stehen als globale Datenbereiche jedem Baustein zur Verfügung,
- Zu einem Zeitpunkt ist jeweils genau ein DB aktiv,
- DBs besitzen Informationen über die Datentypen ihrer Daten.

Spätestens hier stellt sich die Frage „wo finde ich die DBs in der neuen Norm wieder?“, nachdem Datenbausteine in diesem Buch bisher kein Thema waren.

Die IEC 61131-3 behandelt keine Datenbausteine. Anstelle der Aktivierung eines (globalen) Datenbausteins für Parametrierungsdaten und als Arbeitsbereich für FBs schlägt sie das in Kap. 2 erläuterte Instanziierungskonzept bei Funktions-

bausteinen vor: jeder FB erhält automatisch seinen eigenen „lokalen“ Datensatz zugeordnet. Zusätzlich kann ein FB auf globale Daten zugreifen, so daß die Funktionalität der Datenbausteine für Parametrierung und Arbeitsbereich durch die IEC 61131-3 abgedeckt wird.

Allerdings sind solche „Instanzdatenbereiche“ von FBs **nicht** wie POEs separat austauschbar und initialisierbar. Datensätze wie Rezepturen sind also entweder im Programm der SPS bereits enthalten oder werden durch Austausch der gesamten POE, die die neuen Daten enthält, gewechselt.

Diese fehlende Eigenschaft ist der IEC-Normungsgruppe wohl bewußt und wird voraussichtlich im Technical Report 2 behandelt werden.

Im folgenden wird beschrieben, wie sich DB-Eigenschaften weitgehend mit Mitteln der IEC 61131-3 realisieren lassen.

Zusammengehörende Daten werden über Strukturen und Felder deklariert; wie Datenbausteine bestehen diese aus einer Anzahl von Werten und Datenformaten. Diese Datenbereiche können flexibel als Ein- oder Ausgangsparameter, lokale oder globale Daten verwendet werden. Über solche Strukturen können auch Rezepturen komfortabel gebildet werden.

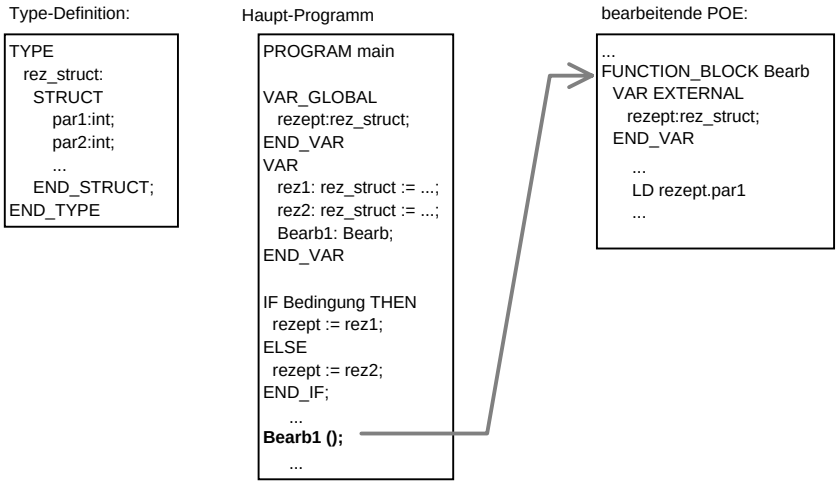
Der Wechsel von Datensätzen kann auf unterschiedliche Arten erfolgen:

1) Die SPS wird angehalten und ein Programm mit anderen Daten komplett vom Programmiersystem auf die SPS geladen. Dies erfordert, daß der PC stets mit der SPS verbunden ist und der zu steuernde Prozeß eine solche Unterbrechung zuläßt.
2) Es werden einzelne Bausteine (POEs) ausgewechselt. Die vom PC nachgeladenen POEs ersetzen POEs mit gleichem Namen, besitzen aber unterschiedliche Anfangswerte für ihre Variablen oder rufen andere POEs auf. Dazu muß das SPS Betriebssystem Funktionen zur Bausteinverwaltung bereitstellen, die ein Nachladen im laufenden Betrieb ermöglichen.
3) Die (z.B. globalen) Datensätze werden dynamisch durch ein Programm zum „Bedienen & Beobachten“ gesetzt bzw. ausgetauscht (via Kommunikation).
4) Es befinden sich alle benötigten Datensätze bereits in der SPS. Das Programm ruft je nach Situation unterschiedliche POEs auf, die unterschiedliche Datensätze besitzen, oder versorgt seine FBs mit unterschiedlichen Parametersätzen.

Ein Austausch erscheint für solche POEs sinnvoll, die vor allem:

- globale Daten zur Verfügung stellen,
- Daten kopieren bzw. initialisieren.

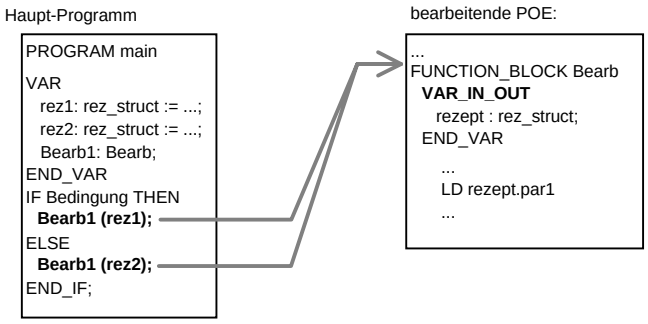
Eine übliche Methode zur Werteversorgung von Programmteilen wie FBs erfolgt über globale bzw. externe Variable:



Bsp. 7.10. Im Hauptprogramm stehen verschiedene lokale Datensätze (rez1, rez2) zur Verfügung. Die global deklarierte Struktur rezept erhält bedingungsabhängig Werte zugewiesen, die von allen anderen POEs mit entsprechender EXTERNAL-Deklaration bearbeitet werden können.

Diese Werteversorgung über GLOBAL-Deklarationen besitzt den Nachteil, daß ständig alle Datensätze auf der SPS geladen sein müssen. Systeme, die die Konfiguration von globalen Variablen unterstützen, können dies umgehen. Eine Ressource kann Werte ihrer globalen Daten von anderen Ressourcen oder über Zugriffspfade erhalten.

Globale Daten sind bei Programm-Änderungen prinzipiell fehleranfällig, da aufgrund der möglichen Seiteneffekte sämtliche Stellen zu berücksichtigen sind, an denen sie verwendet werden. Sie verletzen in diesem Zusammenhang die von der IEC 61131 beabsichtigte Informationskapselung (Objektorientierte Daten). Um diesen Nachteil zu vermeiden, können die in Bsp. 7.10 verwendeten globalen Daten auch durch Aufrufparameter ersetzt werden:



Bsp. 7.11. Um Seiteneffektfreiheit gegenüber der Methode in Bsp. 7.10 zu erhalten, werden die Rezepturdatensätze `rez1` und `rez2` als Ein/Ausgangsparameter (per Zeiger; damit muß nicht die gesamte Datenstruktur kopiert werden) weitergereicht.

7.9 FB- Verschaltung

7.9.1 Datenaustausch und Koordination von Bausteinen in verteilten Systemen

Die IEC 61131-3 definiert PROGRAM-Bausteine, die hierarchisch FBs und Funktionen aufrufen (mit CAL) und mit Parametern versorgen. Jedes PROGRAM bzw. ein Teil der FB-Instanzen erhält eine Task zugeordnet. Zwischen den Tasks kann über Globale Variable oder ACCESS-Variable kommuniziert werden.

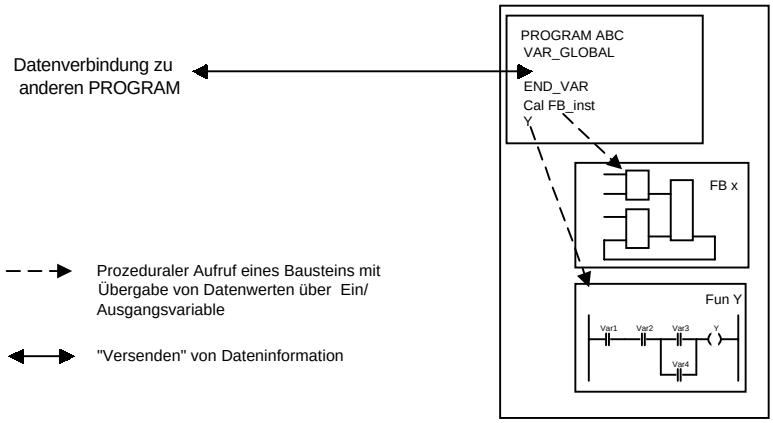


Abb. 7.7. Ein Programm ruft weitere Bausteine auf (prozedural) und versorgt den aufgerufenen Baustein über Parameter mit Information. Mit anderen Programmen werden lediglich Daten ausgetauscht.

Dezentrale SPS-Systeme, vorwiegend in der Anlagentechnik, Energieversorgung oder Gebäudeautomatisierung eingesetzt, erfordern:

- eine parallele autonome Bearbeitung einzelner Verarbeitungsaufgaben,
- geografisch getrennte Verarbeitungsknoten,
- einen asynchronen Datenaustausch.

Zur Realisierung verteilter Anwendungen wird heute zumeist folgendes Vorgehen angewendet: Aus Programm-Bibliotheken werden vorgefertigte Bausteine in das zu entwickelnde Projekt kopiert, individuell erstellte Bausteine ergänzen fehlende Funktionalität. Ein PROGRAM bildet zusammen mit den aufzurufenden FBs und Funktionen eine ablauffähige Einheit.

Allen PROGRAM-Bausteinen werden nun einzelne Netzknoten (Tasks von Steuerungen im Netzwerk) zugewiesen und die Ein- und Ausgänge miteinander verbunden. Nicht-verbundene Eingänge von Programm-Instanzen erhalten bei Bedarf individuelle Parameterwerte. Dies ist in Abb. 7.8 dargestellt.

Bibliotheksbausteine erstellen meist Spezialisten der Hardwarehersteller bzw. stehen als fest installierter Code (EPROM, ...) in Steuerungen bzw. Netzwerkknoten zur Verfügung.

Dies stellt einen Extremfall des IEC 61131-3-Modells dar. Das Anwenderprogramm wird über vorgefertigte Bausteine „konfiguriert“. Die Programme laufen weitgehend autonom ab, ohne von einem anderen Baustein „aufgerufen“ zu

werden. Vom PROGRAM aufgerufene Funktionen und FBs im Sinne der IEC 61131-3 werden lokal vom Knoten zur Verfügung gestellt. Ein direkter Aufruf (CAL) eines Bausteins der benachbarten CPU ist nicht möglich (PROGRAM darf kein PROGRAM aufrufen).

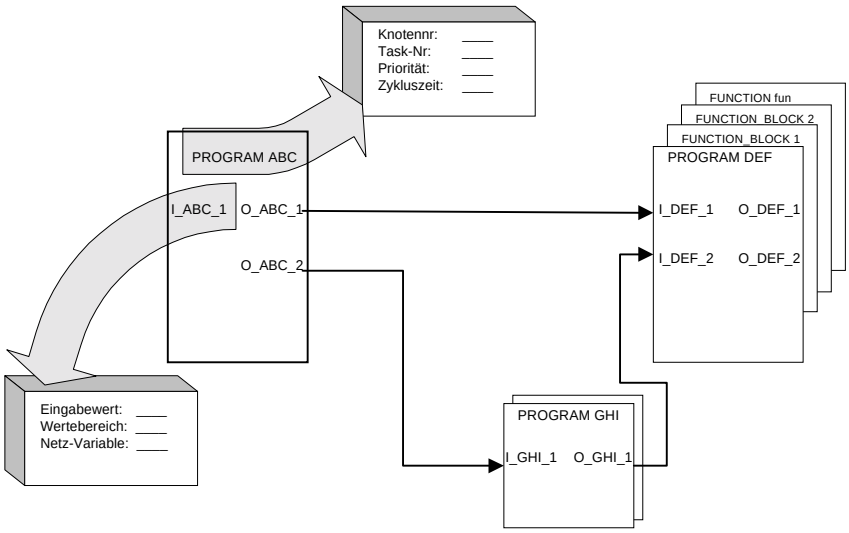


Abb. 7.8. Zuweisung von Bausteinen an Netzwerkknoten und SPS-Tasks. Die PROGRAM-Bausteine laufen alle unabhängig und sind nur über ACCESS-Variable verbunden.

Weicht man von der Definition der IEC 61131 ab, daß nur Programme Verbindungen zu Task-fremden Bausteinen haben dürfen, kann man FBs beliebig auf Verarbeitungsknoten (Tasks) verteilen und miteinander verschalten. Damit kann ein Algorithmus wesentlich granularer und modularer auf die Netzwerk-Architektur verteilt werden. Die Leistung eines Automatisierungsanlage kann ohne Programmänderung durch Hinzunahme weiterer Verarbeitungsknoten und Umkonfiguration erweitert werden.

Wie mit Abb. 7.8 verdeutlicht wird, ergeben sich daraus zwei Schwierigkeiten:

1) Das Ablaufverhalten der einzelnen Bausteine muß stärker aufeinander abgestimmt werden, denn eine Koordination allein über die Datenleitungen ist sehr unübersichtlich.
2) Es bedarf eines Mechanismus, der die Konsistenz des Datenflusses zwischen den Bausteinen (Netzknoten) sicherstellt, d.h. definiert, wann ein Datenwert oder eine Gruppe von Datenwerten gültig ist.

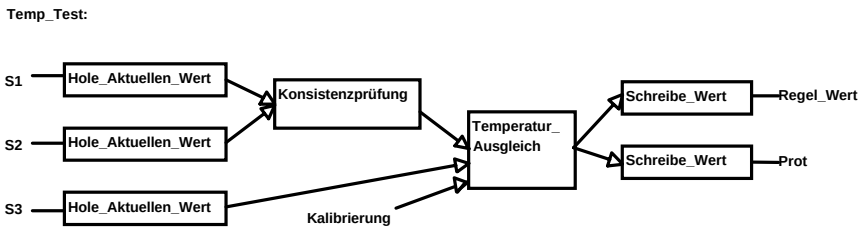
Solange so verschaltete Bausteine auf *einem* Netzwerkknoten ablaufen, kann das Ablaufverhalten durch implizite Festlegungen (Aktivierung der Bausteine gemäß FB-Netzwerk von oben nach unten oder links nach rechts) oder explizite Vorgaben (Vergabe von Ausführungsnummern) konfiguriert werden. Besitzen die Tasks der Bausteine jedoch keine gemeinsame Zeitbasis, da sie auf unterschiedlicher Hardware ablaufen, sind weitere Kontrollmechanismen einzuführen.

Mit diesem Thema beschäftigen sich derzeit Arbeitskreise der IEC, siehe dazu Kapitel 9 und VDI [VDI 3696/1-93] [VDI 3696/3-93].

Durch eine solche Verschaltungstechnik erfolgt eine weitere Trennung von Programmierung und Konfigurierung: Der Programmierer beschreibt sein Programm mit FBs, Funktionen und deren Verbindungen. Danach erfolgt die Zuweisung der Funktionsblöcke auf die Verarbeitungsknoten. Dieselbe Anwendung kann lokal auf einer Task, aber auch verteilt auf vielen Tasks ablaufen, ohne daß die Programmstruktur geändert werden muß.

7.9.2 Makrotechnik bei FB-Verschaltung

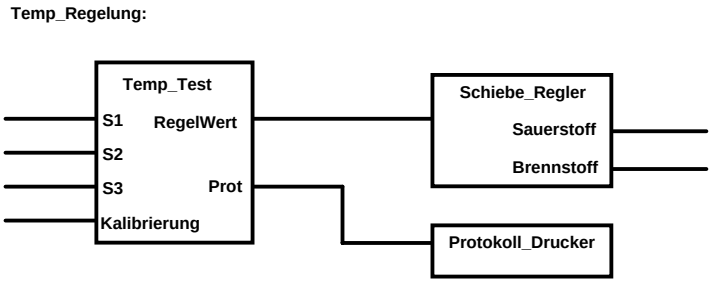
Ein Projekt mit einer in Abschn. 7.9.1 genannten Verschaltungstechnik besteht aus einer Vielzahl von Bausteinen. Zur Strukturverbesserung können lokal plazierte Bausteine zeichnerisch zusammengefaßt werden; diese werden im weiteren als einzelne Baustein dargestellt. Zusammengehörende Funktionsteile können einzelnen Arbeitsblättern, sogenannten *Funktionsplänen*, zugeordnet werden. Auf diese *Makrotechnik* wird im folgenden eingegangen. Bsp. 7.12 zeigt ein Beispiel aus der Anlagentechnik.



Bsp. 7.12. Verschaltung einfacher Grundelemente in der Anlagentechnik

Die Plazierung und Verbindung von Bausteinen (Hole_Aktuellen_Wert mit Konsistenzprüfung) erfolgt grafisch durch den Anwender. Die Bausteine haben eine genau beschriebene Funktionalität. Datendeklarationen müssen nicht eingegeben werden, da sie implizit durch die Instanziierung des Bausteins vorgenommen werden. Lediglich bei den Ein-/Ausgangsparametern muß beachtet werden, daß die beiden Datentypen zusammenpassen (Typverträglichkeit).

Die vom Hersteller zur Verfügung gestellten Elementar-Bausteine können nun durch eine Art „Makrotechnik“ zu größeren Funktionsblöcken zusammengefaßt werden. Diese Methode entspricht dem Aufbau von Datenstrukturen aus Elementaren Datentypen, man spricht deshalb auch von *Abgeleiteten Funktionsbausteinen*.



Bsp. 7.13. Die Bausteine aus Bsp. 7.12 erreichen durch eine Makrotechnik, d.h. Zusammenfassung von Bausteinen, eine immer höhere Funktionalität und Spezialisierung.

Nach der Festlegung des Bsp. 7.13 steht der Baustein Temp_Regelung als eigener Baustein für weitere Aufgaben zur Verfügung.

Handelt es sich bei den elementaren FBs beispielsweise um Hersteller-Bausteine mit einem genau beschriebenen Zeitverhalten, lassen sich auch gute Simulationsergebnisse (Laufzeit, Änderungsverhalten bei anderer Hardware-Zuteilung oder Änderung der Kommunikationswege,...) erzielen.

7.10 Diagnose, Fehlererkennung und - Reaktion

Diagnose heißt im wesentlichen: „Erkennen von Fehlerzuständen im laufenden Betrieb und deren Lokalisierung“. Es gibt vier Bereiche, in denen Fehler auftreten können:

1) Hardware der SPS inkl. Verbindungen zu anderen Geräten,
2) Software der SPS (Betriebssystem),
3) Software des Anwenders,
4) Prozeßverhalten. Der zu steuernde Prozeß gerät in unerwünschte Zustände.

Dabei kann generell zwischen Systemfehlern und Programmierfehlern unterschieden werden.

Die Hersteller bieten verschiedene Hilfsmittel zur Diagnose-Unterstützung an. Dies kann zusätzliche Hardware sein, die Fehlersituationen abprüft und relevante Daten liefert, oder Softwarefunktionen, die eingebunden werden. Auch AS eignet sich sehr gut, um Fehler in der laufenden Anlage zu finden (wie: „Transition XY schaltet nicht“) oder nach Ausnahmesituationen vordefinierte Reaktionen anzusteuern.

Allgemeines Fehlerkonzept der IEC 61131-3.
Die IEC 61131-3 beschränkt sich auf einen allgemeinen Ansatz, der dem Anwender gewisse Hilfen für 2) und 3) in die Hand gibt. Sie beschreibt in einer Fehlerliste, siehe Anh. E, mögliche Fehlerquellen, die jeweils in einem SPS-System durch eine der folgenden Möglichkeiten berücksichtigt sein müssen:

1) Der Fehler wird nicht beachtet; dies muß in einer separaten Anwender-Dokumentation festgehalten sein.
2) Das System macht auf mögliche Fehlersituationen beim Erstellen oder Laden aufmerksam (Warnung).
3) Das System meldet während der Programm-Ausführung (Laufzeit) in geeigneter Weise Fehler und bietet herstellerspezifische Möglichkeiten zum Eingriff.

Qualitätssicherung spielt in der Automatisierungswelt eine sehr gewichtige Rolle. Die Güte heutiger Compiler verhindert manche typischen Fehler bereits während der Programmerstellung. Die Sprachdefinition der IEC 61131-3 bietet Konzepte wie die strenge Typüberprüfung von Daten, die manche Fehler erst gar nicht entstehen lassen. Im wesentlichen handelt es sich dabei um „Fehlerprophylaxe“ während der Programmierung. Viele Fehler lassen sich jedoch erst während der Laufzeit feststellen.

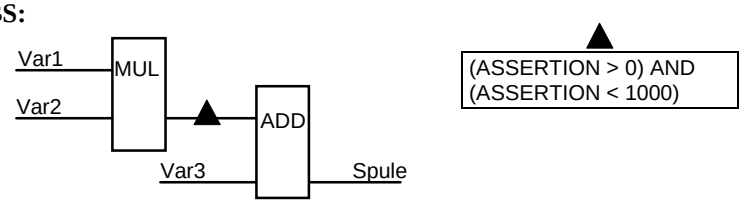
Es gibt eine Reihe von Fehlersituationen wie Division durch Null (siehe Anhang E), die vom SPS-System abgeprüft werden sollten. Die IEC 61131-3 [IEC TR3-94] schlägt vor, eine einheitliche globale (herstellerspezifische) Datenstruktur als Fehler-Datentyp zu definieren, die den Status (Fehler ja/nein) einer Operation, den Fehlertyp (Division durch Null, ...) und die Fehlerstelle (POE-Namen, ...) beinhaltet. Eine solche Statuskennung kann dann vom Anwender-Programm abgefragt werden oder, falls realisiert, mit dem SINGLE-Eingang einer TASK verbunden werden, siehe auch Abschn. 6.3.4. Diese Task wird mit einer System- oder Fehler-Routine verbunden.

Im Fehlerfall setzt ein solches SPS-System den Wert des Fehlerstatus auf TRUE und belegt den Fehlertyp. Der erst genannte Wert startet mit steigender Flanke die Fehlertask.

Erweitertes Fehlermodell (nicht IEC).
Zur Verbesserung der Programm-Qualität wäre es wünschenswert, wenn der Anwender selbst Fehler-Situationen in standardisierter Form definieren könnte. Vorstellbar hierfür wären Sprachhilfsmittel wie „zugesicherte Eigenschaften“. Dies sind vom Programmierer eingestreute System-Testroutinen, die spezielle Überprüfungen z.B. von Variablen durchführen:

- befindet sich eine Variable noch innerhalb des an dieser Programmstelle gültigen Wertebereichs?
- besteht Übereinstimmung zwischen zwei Variablen?

FBS:



AWL:

```
LD    Var1
MUL   Var2
ASSERTION((AE > 0) AND (AE < 1000))
ADD   Var3
ST    Spule
```

Bsp. 7.14. Zur Erkennung von Laufzeitfehlern sollten Abfragen möglich sein, die vom System selbst berechnet werden. In der grafischen Darstellung könnten Verbindungen über Zusicherungs-Symbole abgesichert werden. Die Ausdrücke sollten in einer genormten Sprache (z.B. ST) geschrieben sein.

Diese Zusicherungen könnten einfache Ausdrücke für das Aktuelle Ergebnis (AE) in AWL wie in Bsp. 7.14 sein. Mit Hilfe von Zusicherungen (engl.: assertion) werden zur Laufzeit „wichtige“ Werte überprüft, die das SPS-Programm zum korrekten Ablauf benötigt. Es sind auch komplexe Ausdrücke denkbar, die an vorgegebenen Programmstellen Ein- und Ausgangswerte vergleichen, Datenkonsistenzberechnungen durchführen oder wichtige Prozeßparameter in ihrem Zusammenhang überprüfen.

Erste Ansätze zur automatischen Fehlerüberwachung sind in einigen Programmiersystemen zu finden, die z.B. eine Feldüberwachung durchführen, d.h. der Index zu einem Feld darf die deklarierte Feldanzahl nicht überschreiten oder negativ werden. Diese Vorgaben werden auch durch die IEC 61131-3 sprachlich unterstützt; siehe dazu Kap. 3.

Es muß parametrierbar sein, was bei einer Verletzung der Zusicherung zu geschehen hat: Programm-Stopp, Ausgabe einer Fehlermeldung an die Visualisierungs-Soft- und Hardware. Für ein sogenanntes „Exception Handling“ sollten mehrere Fehler-Routinen (Anwender- oder Standard-Fehler- POEs), ähnlich dem oben beschriebenen Mechanismus, definierbar sein, die jeweils – abhängig vom Reaktionswunsch – einer oder mehreren Zusicherungen zugewiesen werden. Diesen Routinen sollten Sonderrechte wie die Möglichkeit eines Programmstopps, Neustarts o.ä. eingeräumt werden.

Derzeit müssen solche Zusicherungsbedingungen und die entsprechende Fehler-Reaktion (ohne Sonderrechte) vom Anwender selbst programmiert werden, was nicht unbedingt zur Übersichtlichkeit des Programms beiträgt.

Die Architektur der Fehlerbehandlung zieht sich quer durch die Programme und derzeitige Lösungen sind herstellerabhängig. Das Fehlen von standardisierten Fehlerüberwachungs- und -Reaktionsmöglichkeiten erschwert die Portierung von Programmen auf Fremdsysteme und erfordert wieder Systemspezialisten mit besonderer Ausbildung.

7.11 Hardware-Abhängigkeiten

Untersuchungen bei der Portierung von Nicht-IEC-Programmen auf IEC Programmiersysteme haben gezeigt, daß selbst bei sehr guten Cross-Compilern selten mehr als 60 % automatisch übersetzt werden können. Ursache dafür sind starke Hardware-Abhängigkeiten, die an vielen Stellen der Programme benutzt werden. Es werden Bausteine zur Ansteuerung weiterer Steuerbaugruppen, herstellerspezifische Kommunikationsbausteine eingesetzt oder Hardware-Adressen (Statusregister, spezielle Speicherbereiche, ...) zugelassen.

Diese Herstellerbesonderheiten kann und will die IEC 61131-3 nicht eliminieren. Schließlich soll durch eine Vielzahl unterstützender Soft- und Hardware eine möglichst hohe Funktionalität erreicht werden. Um die Portierbarkeit zu erleichtern, schlägt die IEC 61131-3 folgende Mechanismen vor:

- Alle Informationen, die ein Programm von außerhalb benötigt, erhält es über die IEC 61131-3 konformen globalen Variablen oder Zugriffspfade, sowie über spezielle Kommunikations-FBs, die in der IEC 61131-5 beschrieben sind.
- Verwendung von E/A-Peripherie sind im PROGRAM oder der Konfiguration zu deklarieren.
- Hardwareabhängigkeiten sind in einer speziellen Tabelle zu hinterlegen, die jeder Hersteller mit seiner Software mitzuliefern hat.

Die Liste der implementierungsabhängigen Parameter ist in Anh. F zu finden.

7.12 Offenheit für neue Funktionalität

Die Funktionalität der neuen, vollgrafischen Programmiersystem-Generation übersteigt den Realisierungsaufwand bisheriger Systeme um ein Vielfaches. Um diese Kosten aufzufangen, ist eine engere Kooperation der Hard- und Softwarehäuser unumgänglich.

Ein Schritt in diese Richtung stellt die PLCopen als unabhängiges Forum für SPS Soft- und Hardware Hersteller dar.

Im Zug der Europäischen Gemeinschaft wurden mehrere Projekte durchgeführt, bzw. sind weitere geplant, die eine engere Kooperation der Branche zum Ziel haben. Durch gemeinsame Maßnahmen soll erreicht werden, daß:

- der Anwender seine Programme auf mehreren Plattformen verwenden kann und damit Neuentwicklungen und Testkosten reduzieren kann,
- durch Schnittstellen-Vereinheitlichung wie OLE oder OPC die gemeinsame Entwicklung von Funktionserweiterungen (wie Simulationswerkzeuge, Logikanalyse) ermöglicht wird.

7.12.1 Austausch von Programmen und Daten

Verwendet ein Anwender unterschiedliche SPS-Systeme, so sieht er sich oft mit der Tatsache konfrontiert, daß er während der Entwicklung Programm-Bausteine austauschen möchte (statischer Programmaustausch) oder während der operativen Phase Daten zwischen den unterschiedlichen SPSen zu transferieren hat (dynamischer Datenaustausch). Dies wird in Abb. 7.9 skizziert.

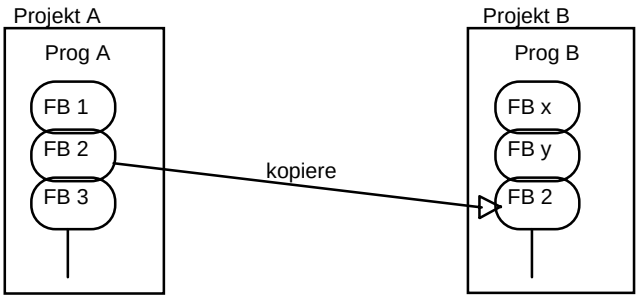
Beim Datenaustausch wird Dateninformation über Kommunikationsbausteine während der Ausführung der SPS-Programme an andere Ressourcen bzw. Tasks gesendet. Da diese Ressourcen evtl. andere Prozessoren bzw. Betriebssysteme mit zugehörigen Nachrichtenmechanismen besitzen und es sich um einen beidseitigen Datenaustausch handelt, müssen diese Daten von einer internen in eine allgemeingültige Form gewandelt werden, die die Partner-SPS auch versteht.

Die IEC 61131-5 definiert die Schnittstelle von Standardbausteinen zur Kommunikation, läßt jedoch Struktur und Inhalt (Semantik) der übermittelten Information offen.

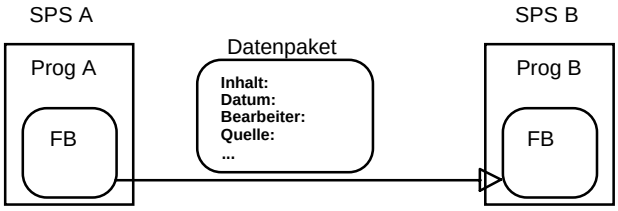
Beim Programmaustausch wird eine POE in ein anderes Projekt kopiert, um die Funktionalität ohne Neuschreiben zur Verfügung zu haben (Wiederverwendung). Hierbei handelt es sich um eine Funktionalität des Programmiersystems. Derzeit ist

dieser Austausch oft nur POE-weise möglich („Kopieren“ der POE in das Projekt). Die PLCopen definierte dazu ein Datenformat, daß zusätzliche Information wie Entstehungsort, Sicherungscodes usw. mitliefert.

Der nächste Schritt wird die Definition einer Quellbibliothek sein, in die Quellen geschrieben oder ausgelesen werden können.



a) Das Programmiersystem erlaubt das Einkopieren eines FBs (statischer Programmaustausch)



b) SPS A schickt an SPS B Nachrichten (dynamischer Datenaustausch)

Abb. 7.9. Statischer Programmaustausch (a) und dynamischer Datenaustausch (b) zwischen zwei unterschiedlichen Projekten bzw. SPS-Familien.

7.12.2 Ergänzung durch weitere Softwarepakete

Derzeit verfügbare IEC 61131-3-Programmiersysteme benutzen zwar teilweise dasselbe Betriebssystem, besitzen aber solch unterschiedliche Ausprägungen, daß ein Austausch von Teilpaketen (Komponenten des Programmiersystems) schwierig ist. Viele Systeme lassen die dazu erforderliche Modularität – trotz der nun verbesserten Voraussetzungen durch Windows oder OS/2 – vermissen.

Eine mögliche modulare Aufteilung eines Programmiersystems zeigt Abb. 7.10.

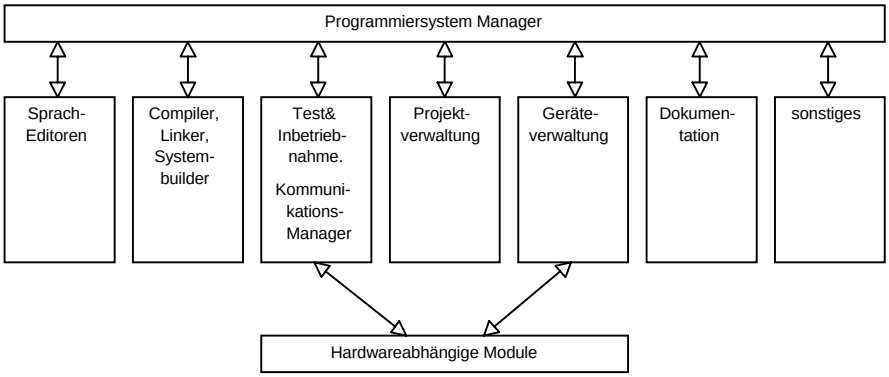


Abb. 7.10. Modularisierung eines SPS- Programmiersystems ist eine Voraussetzung für künftige Erweiterungen des Systems.

Derzeit existieren weder Festlegungen über die Art einer solchen Architektur noch die Art des Kommunikationsaustausches (z.B. über Datei, OLE, OPC, Klassen-bibliotheken usw.).

Ergänzungen zu den oben genannten Teilpaketen bilden beispielsweise:

- Werkzeuge zur Anlagen-Projektierung,
- Simulations-Werkzeuge,
- Allgemeine Datenverwaltungssysteme,
- Parametrier-Editoren,
- Logik-Analysator,
- Anlage-Diagnose,
- BDE-, SCADA Systeme, Protokollgeräte- Anschluß,
- CAD- Anschluß,
- Netzverwaltung.

8 Stärken der IEC 61131-3

Kapitel 1 beschreibt Ziele und Nutzen der Norm IEC 61131-3 für Hersteller und Anwender. Inwieweit erfüllt dieser neue Programmierstandard die an ihn gestellten Erwartungen?

In den vorangegangenen Kapiteln wurden viele Eigenschaften und Konzepte der neuen SPS-Programmierung beschrieben und erläutert. Darin spiegelten sich die zentralen Konzepte wider, die hier abschließend zusammengefaßt werden.

Folgende herausragenden Eigenschaften der SPS-Programmierung mit IEC 61131-3 sind besonders bemerkenswert:

- Komfort und Sicherheit durch Variablen und Datentypen,
- Bausteine mit erweiterten Möglichkeiten,
- SPS-Konfiguration mit Laufzeitverhalten,
- Einheitliche (Programmier-) Sprachen,
- Strukturierte SPS-Programme,
- Trend zu offeneren SPS-Programmiersystemen.

8.1 Komfort und Sicherheit durch Variablen und Datentypen

Lokale und Globale Variable statt Hardware-Adressen
Bisher war es üblich, auf sämtliche Datenbereiche der SPS über globale Adressen zuzugreifen, wobei der Programmierer selbst dafür sorgen mußte, daß Teilprogramme nicht versehentlich Daten eines anderen Programmteils überschreiben. Dies gilt insbesondere beim Zugriff auf Peripherie, Merkerbereiche oder Datenbausteine.

Durch die IEC 61131-3 werden die globalen Hardware-Adressen durch Variable ersetzt und für sie Gültigkeitsbereiche eingeführt: Das Programmiersystem trennt *automatisch* die Verwendung POE-**lokaler** und **globaler** Variablen. Auf globale

Hardware-Adressen kann dennoch zugegriffen werden, indem im Deklarationsteil eine Zuweisung der Adresse an einen später verwendeten Variablennamen erfolgt.

Typgerechter Zugriff auf SPS-Daten
Der SPS-Programmierer mußte in der Vergangenheit darauf achten, daß eine SPS-Adresse jeweils mit demselben Datentyp gelesen und beschrieben wird. So ist es möglich, ein Merker- oder Datenwort an verschiedenen Programmstellen einmal als Ganzzahl (Integer) oder als Gleitpunktzahl (Real) zu interpretieren.

Dadurch entstehende Programmierfehler können mit der IEC 61131-3 ausgeschlossen werden, da jeder Variablen (auch direkte Hardware-Adressen) ein Datentyp zugeordnet werden muß. Dadurch kann das Programmiersystem eine Überprüfung automatisch vornehmen.

Jederzeit definierte Anfangswerte für Anwenderdaten
Sämtliche Daten werden zusammen mit ihrem Datentyp als Variable explizit bekanntgegeben. Jeder Datentyp besitzt (standardmäßig oder benutzerdefiniert) einen Anfangswert, so daß jede Programm-Variable ihren Eigenschaften entsprechend korrekt initialisiert wird.

Variable können zusätzlich mit dem Attribut „batteriegepuffert“ versehen werden, die Zuordnung zu physikalisch gepufferten Speicherbereichen erfolgt dann automatisch.

Felder und Datenstrukturen für jeden Zweck
Basierend auf vordefinierten Datentypen kann der SPS-Programmierer Felder und komplexe Datenstrukturen für anwendungsgerechte Daten entwerfen, wie sie bereits von höheren Programmiersprachen her bekannt sind.

Feldgrenzen und Wertebereiche werden durch das Programmiersystem sowie zur Laufzeit überwacht.

Einheitliche Variablendeklaration
Die umfangreichen Möglichkeiten der Verwendung von Variablen ist weitgehend für alle IEC-61131- Sprachen identisch.

8.2 Bausteine mit erweiterten Möglichkeiten

Wiederverwendbarkeit von Bausteinen
Bausteine (POEs) wie Funktionen und Funktionsbausteine können unabhängig vom herstellerspezifischen System formuliert werden. Daraus ergibt sich die Möglichkeit, Baustein-Bibliotheken herstellerübergreifend in verschiedenen SPS-Projekten einzusetzen.

Aufruf-Parameter, Rückgabewerte sowie lokale Daten jedes FBs, d.h. jeder FB-Instanz bleiben zwischen den Aufrufen erhalten. Jede Instanz besitzt nach der

neuen Norm automatisch ihren eigenen Datenbereich („gekapselt“), auf dem sie unabhängig von externen Daten ihre Berechnungen ausführen kann. Der vorherige Aufruf eines Datenbausteins, auf dem der FB arbeitet, kann dadurch entfallen.

Auch (Haupt-) Programme können als Instanzen für mehrere Tasks einer CPU gleichzeitig verwendet werden.

Leistungsfähige Baustein-Parametrierung

Zur Übergabe von Daten an Bausteine werden mehrere Mechanismen zur Verfügung gestellt, mit denen POE-Parameter übergeben bzw. zurückgegeben werden können:

- VAR_INPUT: Wert einer Variable
- VAR_IN_OUT: Zeiger auf eine Variable
- VAR_OUTPUT: Rückgabewert
- VAR_EXTERNAL: Globale Variable einer anderen POE
- VAR_ACCESS: Zugriffspfad innerhalb der Konfiguration

Aus diesem Spektrum wurden bei bisherigen SPS-Systemen zumeist nur globale Daten sowie die Möglichkeit der Übergabe von Werten an den aufgerufenen Baustein realisiert (und nicht umgekehrt).

Standardisierte SPS-Funktionalität

Um typische SPS-Funktionalität zu standardisieren, führt die neue Norm Standard-Funktionen und -Funktionsbausteine ein, die bzgl. Aufrufschnittstelle, grafischer Darstellung und ihrem Laufzeitverhalten fest vorgegeben sind.

Diese standardisierte Baustein- „Bibliothek“ für jedes SPS-System bildet eine wichtige Grundlage für die einheitliche und herstellerübergreifende Ausbildung, Programmierung und Dokumentation.

8.3 SPS-Konfiguration mit Laufzeitverhalten

Konfigurationen strukturieren ein SPS-Projekt

Die Zuordnung der Bausteine zur Steuerung erfolgt auf der obersten Ebene eines SPS-Projekts (Konfiguration). Dabei werden Laufzeiteigenschaften, Schnittstellen nach außen, SPS-Adressen und -Peripherie der Teilprogramme festgelegt.

Laufzeiteigenschaften für SPS-Programme

Während bei bisherigen SPS-Systemen Laufzeiteigenschaften wie Zyklusdauer oder Priorität von SPS-Programmen oft systemspezifisch vorgegeben wurden, können solche Parameter nun individuell durch die Definition von Tasks vorgegeben und dokumentiert werden.

SPS-Programme dürfen nicht rekursiv sein. Daher kann der zur Laufzeit benötigte Speicherplatz off-line ermittelt werden und die Programme werden sicherer gegen versehentliche Rekursion.

8.4 Einheitliche Sprachen

Die IEC 61131-3 definiert fünf Programmiersprachen, mit denen ein breites Spektrum von Anwendungen abgedeckt werden kann.

Aufgrund der internationalen Normierung werden SPS-Fachleute zukünftig eine einheitlichere Ausbildung erhalten. Sie werden firmen- und branchenübergreifend „dieselbe Sprache sprechen“.

Ausbildungs- und Weiterbildungskosten werden deutlich sinken, da sich der herstellerspezifische Anteil auf die Besonderheiten eines SPS-Systems beschränken kann.

Die Anlagendokumentation wird beim gemischten Einsatz unterschiedlicher SPS-Systeme einheitlicher.

8.5 Strukturierte SPS-Programme

Durch Verwendung der vielfältigen Sprachelemente der IEC 61131-3 lassen sich Anwenderprogramme gut strukturieren; der Bogen reicht von der Baustein- und Datendefinition bis zur Konfiguration.

Dies unterstützt sowohl den strukturierten Programmentwurf (top down, bottom up) als auch die Pflege und Wartung der Programme wesentlich.

Mit dem „Strukturierungsmittel“ Ablaufsprache steht dem Anwender darüber hinaus eine geeignete Sprache zur übersichtlichen und anwendungsbezogenen Formulierung der Steuerungsaufgabe zur Verfügung.

8.6 Trend zu offeneren SPS-Programmiersystemen

Eine Standardisierung von Programmiersprachen führt zu Standardsoftware, die herstellerunabhängigere, portable Programme ermöglicht, wie es beispielsweise bei den gängigen Programmiersprachen (Assembler, C, COBOL) im Personal Computer-Bereich inzwischen der Fall ist. Der Bewertungsmaßstab ist durch die „feature tables“ der IEC 61131-3 vorgegeben. Dadurch werden SPS-Programmiersysteme unterschiedlicher Hersteller mit einer gemeinsamen Grundfunktionalität vergleichbarer. Differenzierungsmerkmale der Hersteller erstrecken sich jetzt mehr auf Zusatzfunktionen wie Logik-Analysator und Off-Line-Simulator als auf die Merkmale der Programmiersprachen selbst.

Das gemeinsame Look&Feel der SPS-Programmierung wird auch international einheitlicher. Selbst bisher sich abgrenzende SPS-Märkte in Europa, USA oder Asien rücken näher zusammen.

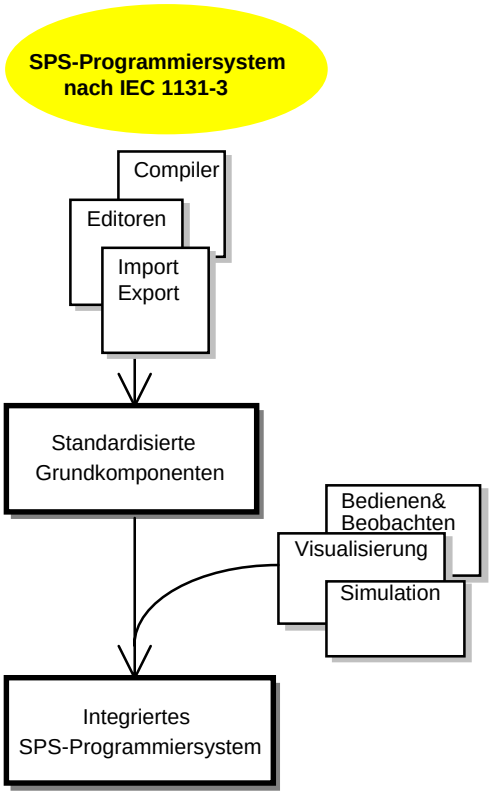


Abb. 8.1. Trend zu offenen, standardisierten Komponenten auf der Basis von IEC-konformen Programmiersystemen

SPS-Programmiersysteme der neuen Generation werden standardisierte Grundkomponenten neben hoch differenzierten Zusatzpaketen enthalten, um ein breites Anwendungsspektrum abdecken zu können.

Wie es Abb. 8.1 ausdrückt, fördert die Standardisierung der IEC 61131-3 integrierte Systeme auf Basis von austauschbaren Standard-Komponenten wie Editoren, Compiler oder Export- und Importfiltern mit offenen, d.h. mehrfach nutzbaren Schnittstellen.

Bisher oft außerhalb der SPS-Programmiersysteme realisierte klassische Zusatzpakete wie Bedienen&Beobachten, Simulation oder Visualisierung bewirken die Entstehung von De-facto-Standard-Schnittstellen dieser Komponenten.

8.7 Fazit

Die Einführung der IEC 61131-3 bedeutet sowohl für SPS-Hersteller als auch für Anwender Umsteigen von „eingefahrenen Geleisen“ zugunsten einer fortschrittlicheren Programmier-Technologie. Der Schritt zur umfassenden Programmiernorm IEC 61131 war sicher notwendig, um ein einheitliches Vorgehen bei der Innovation von Konfiguration und Programmierung von SPS-Systemen zu erreichen.

Mit der herstellerübergreifenden Einheitlichkeit wird der Einarbeitungs- und Schulungsaufwand für die SPS-Programmierer sinken, es werden zuverlässigere Programme geschrieben und die Funktionalität der SPS-Programmiersysteme wird sich den leistungsfähigen Software-Produktionsumgebungen der PCs angleichen.

Konformitätsgrad zur IEC 61131-3, Migrationspfade von bestehenden Systemen hin zu den neuen Architekturen sowie eine ergonomische, leistungsfähige Bedienoberfläche bilden für die Anwender die wichtigsten Kriterien zur Akzeptanz neuer SPS-Programmiersysteme.

Insgesamt führen die heutigen komplexen Anforderungen und wirtschaftlichen Randbedingungen des Marktes zu flexiblen, offenen und damit herstellerübergreifenden SPS-Programmiersystemen.

9 Programmierung durch Konfigurierung nach IEC 61499

Die Programmierung mit Hilfe von grafischen Elementen, die der Vorstellungswelt der zu programmierenden Anwendung entnommen sind, gewinnt zunehmend an Bedeutung.

Mit den bisher vorgestellten grafischen Programmiersprachen KOP, FBS oder AS lassen sich Datenfluß und logische Verarbeitungsfolge symbolisch programmieren bzw. dokumentieren. Es liegt nahe, daneben auch die topologische Verteilung von Programmen, ihre übergreifende Konfiguration und Vernetzung mit anderen Teilen eines verteilten Automatisierungs-Projekts grafisch darzustellen. Dies erfolgt auf einer höheren, abstrakteren Ebene als die bisher kennengelernte Programmierung von POEs.

Dementsprechend heißen solche Werkzeuge zur Konfiguration komplexer verteilter Anwendungen *Konfigurations-Editoren*. Programmteile wie Funktionsbausteine werden zu größeren Einheiten zusammengefaßt. Die Methode wird *Verschaltung von Funktionsbausteinen* genannt.

Um die sprachlichen Mittel für diese Zwecke einheitlich zu definieren, wird derzeit als Ergänzung zur IEC 61131 an der internationalen Norm IEC 61499 gearbeitet. Dieses Kapitel gibt einen raschen Überblick über die Grundlagen und Ideen dieser weiteren Norm und stellt den Zusammenhang zur IEC 61131 her. Eine detailliertere Darstellung der neuen Norm ist eher Gegenstand eines weiteren Buchs.

9.1 Programmierung durch FB-Verschaltung mit IEC 61131-3

Um die Unterschiede zwischen IEC 61499 und IEC 61131-3 zu verdeutlichen, wird im folgenden zunächst auf Besonderheiten der verteilten Programmierung eingegangen.

Die in Kap. 4 beschriebenen Programmiersprachen werden verwendet, um Algorithmen für Bausteine zu definieren. Funktionsbausteine und Funktionen rufen sich gegenseitig auf, teilen sich Information über ihre Parameter mit und bilden zusammen mit der POE PROGRAM ein Programm. Ein Programm läuft als Task auf einer Ressource, d.h. Prozessoreinheit (CPU) einer SPS, ab. Die IEC 61131-3 konzentriert sich weitgehend auf die Beschreibung einzelner Programme mit ihren Ausführungs-Bedingungen. Der Informationsaustausch der Programme untereinander erfolgt über ACCESS-Variable oder globale Datenbereiche. Diese Thematik wurde bereits in Abschn. 7.9 angesprochen und mit Abb. 7.8 verdeutlicht.

Komplexe dezentrale Automatisierungs-Aufgaben besitzen eine umfangreiche Kommunikations- und Ausführungsstruktur. Räumlich getrennte Steuerungen tauschen intensiv Daten aus. Die inhaltlichen und zeitlichen Abhängigkeiten sind zu spezifizieren.

Dazu werden Programme den Tasks der Netzwerkknoten zugeordnet, Ausführungsbedingungen wie in Abschn. 6.1 beschrieben festgelegt sowie die Ein- und Ausgänge der Programme (wie Netzwerkadressen oder Parameterwerte) verschaltet.

Die Verteilung einer Automatisierungs-Lösung, d.h. Funktionsbausteine auf physikalisch und örtlich unterschiedliche Hardware zu konfigurieren sowie deren Synchronisation, ist Gegenstand der künftigen Norm IEC 61499.

9.2 IEC 61499 - die Norm für verteilte Systeme

Der sequentielle Bausteinaufruf der IEC 61131-3 ist keine geeignete Methode zur Programmstrukturierung in verteilten Systemen. Dies wird bereits in Abb. 7.8 deutlich. Ziel eines verteilten, dezentral organisierten Systems ist es, Programme auf mehrere Steuerungen zu verteilen und dadurch parallel auszuführen (im Unterschied zur sequentiellen Abarbeitung beim Bausteinaufruf über CAL). Dabei ist besonders wichtig, Datenkonsistenz zwischen Knoten des vernetzten Systems sicherzustellen, also Zeitpunkte des gegenseitigen Datenaustauschs eindeutig zu definieren.

In der IEC 61499 spielen zwei Arten des Informationsaustausches eine wesentliche Rolle:

1) der Datenfluß mit Anwenderdaten,
2) der Kontrollfluß, der die Gültigkeit der Anwenderdaten als Ereignisinformation steuert.

Das Zusammenspiel von Daten- und Kontrollfluß kann auch mit Mitteln der IEC 61131-3 unter Verwendung von globalen Variablen und Zugriffspfaden programmiert werden. Das Gesamtprogramm wird dabei jedoch leicht unübersichtlich und langsamer.

Um in einem verteilten, also vernetzten Automatisierungssystem die Beziehung zwischen Programmteilen und Elementen der Steuerungs-Hardware einfach und exakt beschreiben zu können, verwendet die IEC 61499 ein Modell („top down"-Ansatz) mit mehreren hierarchischen Ebenen:

- System
- Gerät
- Ressource
- Anwendung
- Funktionsbaustein

Die Begriffe Ressource und Funktionsbaustein werden allerdings gegenüber der IEC 61131-3 in einem erweiterten Sinn definiert, wie im folgenden näher erläutert wird. Die begriffliche Abstimmung zwischen diesen beiden Normen wird durch die Normungsgremien noch erfolgen müssen.

Anstelle der Zuordnung von PROGRAM und TASK zu einer Ressource können den Funktionsbausteinen der IEC 61499 über die Ressource Laufzeiteigenschaften *direkt* zugeordnet werden.

9.2.1 System-Modell

In einer realen Automatisierungsumgebung bearbeiten mehrere Steuerungen, im weiteren *Geräte* genannt, dieselbe oder verschiedene Anwendungen parallel, dies wird in Abb. 9.1 skizziert.

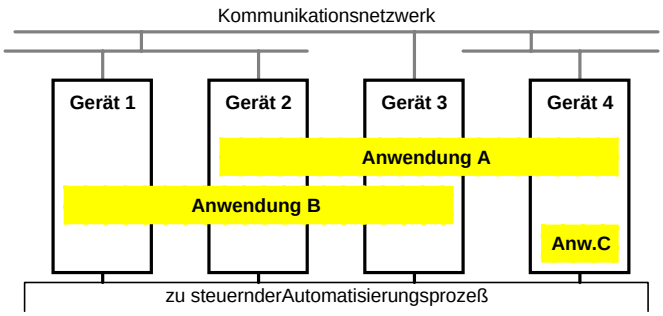


Abb. 9.1. Die Steuerung eines Prozesses kann auf mehrere Geräte **verteilt** erfolgen. Weiterhin können wie bei der IEC 61131-3 mehrere Programme für ein Gerät konfiguriert werden. Die Programmteile tauschen dazu Informationen über Netzwerke aus.

9.2.2 Geräte-Modell

Betrachtet man ein Gerät wie in Abb. 9.2 im Detail, besteht es aus:

- seinen Anwenderprogrammen,
- einer Schnittstelle zum Kommunikationsnetz,
- einer Schnittstelle zum Automatisierungsprozeß und aus
- der Geräte-Hardware, auf der Ressourcen ablaufen.

Eine Ressource stellt eine eigenständig ablauffähige und parametrierbare Einheit (Task im allgemeinen Sinne) dar. Es können mehrere Ressourcen auf einem Gerät laufen, die dieselbe oder unterschiedliche Anwendungen realisieren.

Die IEC 61499 verwendet zwei Sichten auf ein verteiltes Programm, die im weiteren nacheinander aufzuzeigen sind. Zum einen betrachtet diese Norm die Hierarchie System—Gerät—Ressource, um den Systemaufbau und die dazugehörigen Ablaufeigenschaften zu beschreiben. Zum anderen wird orthogonal dazu auch die anwendungsbezogene Sicht eines verteilten Programms definiert. Diese Anwendersicht wird durch weiter unten zu beschreibende Anwendungs- und Funktionsbaustein-Modelle zusammengefaßt.

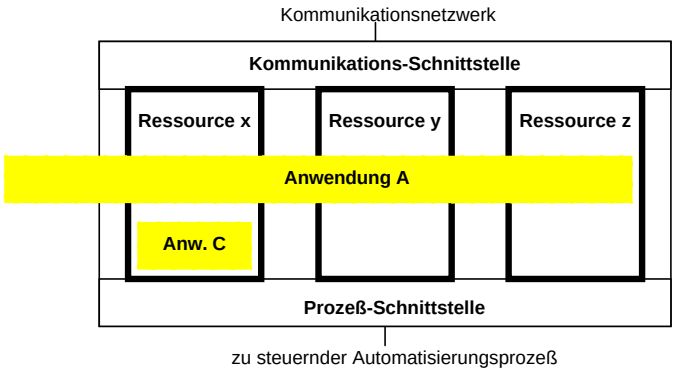


Abb. 9.2. Ein Gerät kann mehrere Ressourcen enthalten, die über gemeinsame Schnittstellen Information mit anderen Automatisierungs-Einheiten sowie dem Automatisierungsprozeß austauschen.

9.2.3 Ressource-Modell

Die Bausteine einer Ressource bestehen aus Funktionsblöcken, die über besondere Schnittstellen sowohl Ereignis- als auch Daten-Information austauschen. Man unterscheidet zwei Arten von Funktionsbausteinen:

1) *Service-Schnittstellen-Funktionsbausteine*, die als Standard-FBs die Schnittstellen zum Automatisierungsprozeß und dem Kommunikationsnetzwerk bilden.
2) benutzerdefinierte Funktionsbausteine, die das eigentliche Anwenderprogramm darstellen.

Wie in der IEC 61131-3 erfolgt eine Unterscheidung zwischen FB-Typ und FB-Instanz.

Jedem Funktionsbaustein werden in der Ressource Laufzeiteigenschaften zugeordnet: maximale Anzahl von Instanzen, seine Ausführungszeit, die Anzahl der Verbindungen etc.

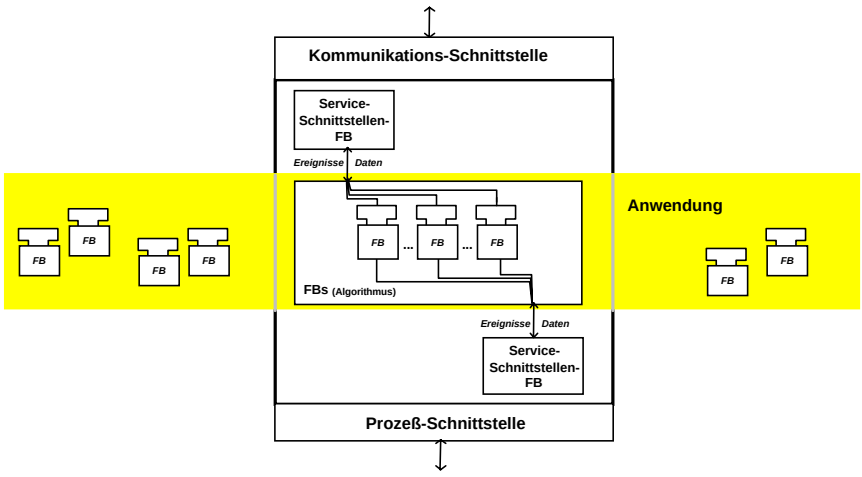


Abb. 9.3. Eine Ressource besteht aus Funktionsbausteinen zur Steuerung und Datenverarbeitung (Algorithmen) mit Schnittstellenbausteinen (Kommunikation / Prozeß).

Die „Verschaltung“ der FBs durch den Anwender erfolgt nicht auf der Ebene der Ressource, sondern der Anwendung; siehe nächster Abschnitt. Der tatsächliche Informationsaustausch zwischen den einzelnen FBs eines Anwenderprogramms erfolgt für den Anwender „unsichtbar“ über die Kommunikations- und Prozeß-Schnittstelle.

Eine oder mehrere Ressourcen können eine Anwendung realisieren.

9.2.4 Anwendungs-Modell

Dieser Abschnitt geht auf die anwendungsbezogene Sicht eines Programms ein. Diese Sicht entspricht dem horizontalen, grauen Balken „Anwendung“ in Abb. 9.3, der sich über mehrere Geräte oder Ressourcen erstrecken kann.

Die Anwendungsebene bildet die eigentliche Programmierebene, indem FBs unabhängig von den Ressourcen miteinander verbunden und verschaltet werden. Sie beschreibt die - später gegebenenfalls auf mehrere Ressourcen - verteilte Anwendung.

Nach der Zuweisung eines Anwenderprogramms, bestehend aus mehreren FBs, an die Ressourcen und dem Programmstart, erfolgt die Kommunikation transparent über die Service-Schnittstellen gemäß der vom Anwender spezifizierten Verschaltung.

Abb. 9.4 zeigt, wie eine Anwendung sowohl steuernde Programmteile (mit Ereignissen) als auch datenverarbeitende Teile (Algorithmen) zusammenfaßt. In dieser Abbildung ist die gegenüber IEC 61131-3 andere graphische Darstellung der Funktionsbausteine zu sehen.

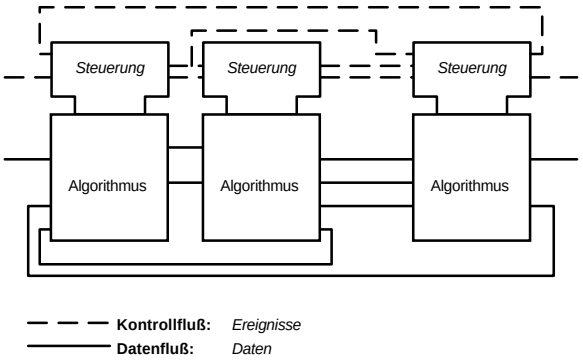


Abb. 9.4. Die Anwendung besteht aus verschalteten Funktionsbausteinen, von denen jeder sowohl steuernde (Kontrollfluß) als auch datenverarbeitende (Datenfluß) Funktionen besitzt.

Steuer- und Dateninformation fließt jeweils von links in einen Funktionsbaustein hinein und wird, entsprechend verarbeitet, von seinen rechten Anschlüssen weitergegeben.

9.2.5 Funktionsbaustein-Modell

Die Funktionsbausteine bilden die Ebene der kleinsten Programmeinheit (wie POEs). Im Unterschied zu den gleichnamigen FBs der IEC 61131-3 besteht hier ein Funktionsbaustein prinzipiell aus zwei Teilen:

1) Ausführungskontrolle: Erzeugen und Verarbeiten von Ereignissen mit Steuer-ein- und ausgängen (Kontrollfluß-Steuerung),
2) Algorithmus mit Datenein- und ausgängen sowie internen Daten (Datenfluß und -verarbeitung).

Die Spezifikation kann textuell oder grafisch erfolgen. Funktionsbausteine werden zur Programmierung wie in der IEC 61131-3 instanziiert. Daher sind die sprachlichen Hilfsmittel zur Beschreibung der FB-Schnittstelle sehr ähnlich; siehe dazu auch Kap. 2.

Abb. 9.5 zeigt die grafische Darstellung eines Funktionsbausteins nach IEC 61499.

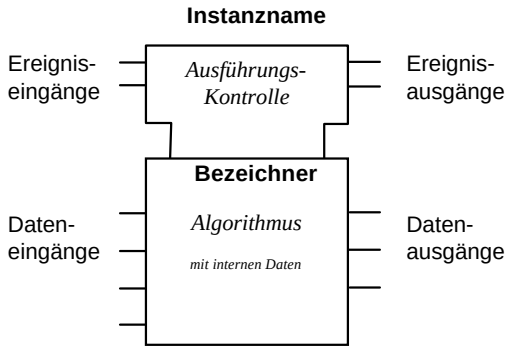
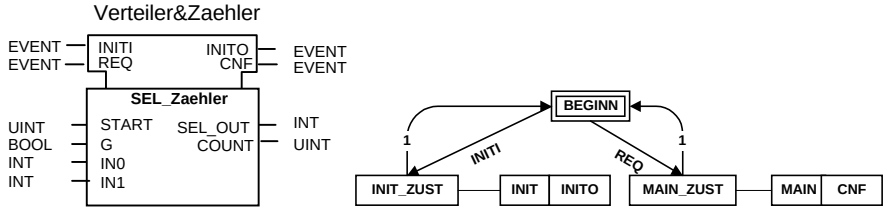


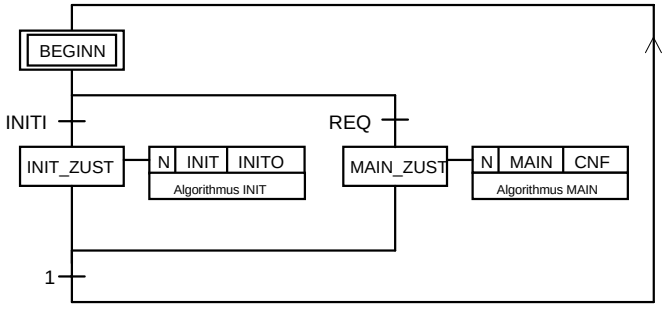
Abb. 9.5. Grafische Darstellung eines Funktionsbausteins. Inhalte der Ausführungskontrolle, des Algorithmus' sowie interne Daten werden auf dieser Ebene nicht dargestellt.

Der Algorithmus-Teil wird wie ein POE-Rumpf nach IEC 61131-3 programmiert.

Der Teil Ausführungskontrolle wird durch ein Zustandsdiagramm bzw. Ablaufsprache (AS nach IEC 61131-3) dargestellt. Die Ereignis-Eingänge dienen als Eingangswerte für sogenannte *Zustandsdiagramme* (engl.: ECC: *Execution Control Chart*). Dabei handelt es sich um Zustandsautomaten, die, abhängig vom Zustand und eintreffenden Ereignissen, die Ausführungszeitpunkte des Algorithmus regeln.



Bsp. 9.1. Beispiel eines Funktionsbausteins mit typisierten formalen Parametern und Zustandsdiagramm



Bsp. 9.2. Ausführungskontrolle des Bsp. 9.1 mit Ablaufsprache (AS) nach IEC 61131-3. Die Ausgangs-Ereignisse CNF und INITO werden vom Anwenderprogramm gesetzt bzw. durch den Aufruf von Standard-FBs verwaltet.

Das Bsp. 9.1 enthält den Funktionsbaustein SEL_ZAEHLER (Instanzname Verteiler&Zaehler), der aus einem ECC-Kontrollteil und dem Algorithmusteil besteht, welcher sich wiederum aus den beiden Algorithmen INIT und MAIN zusammensetzt. Die Ausführungskontrolle bestimmt, welcher Algorithmusteil wann aktiv ist.

In Bsp. 9.1 wird bei Eintreffen des Ereignisses INITI die FB-Steuerung vom Initialzustand BEGINN in den Zustand INIT_ZUST versetzt und der Algorithmus INIT ausgeführt. Danach erfolgt ein SET der Ereignis-Ausgangsvariablen INITO, gefolgt von einem RESET (insgesamt also ein Signalimpuls). Jetzt wertet die Ausführungskontrolle die folgende Transistion, hier mit „1“ parametriert, aus. Dadurch wird in den Zustand BEGINN weitergeschaltet. Bei Eintreffen von REQ wird sinngemäß verfahren.

Dieses Verhalten ist äquivalent zu den Aktionen der Ablaufsprache (AS) der IEC 61131-3 und wird mit Bsp. 9.2 angedeutet. Die IEC 61499 geht allerdings davon aus, daß es günstiger ist, die Ausführungskontrolle über Zustandsgrafen zu spezifizieren, da bei dieser Methode zu einem Zeitpunkt immer nur genau **ein** Zustand aktiv sein kann. In Ablaufsprache kann man dies erreichen, indem Simultan-Verzweigungen verboten werden.

In Bsp. 9.3 wird die textuelle Definition des FB-Typs von Bsp. 9.1 dargestellt.

```
FUNCTION_BLOCK SEL_ZAEHLER
  EVENT_INPUT
        INITI  WITH START;
        REQ  WITH G, IN0, IN1;
  END_EVENT

  EVENT_OUTPUT
        INITO  WITH COUNT;
        CNF  WITH SEL_OUT, COUNT;
  END_EVENT

  VAR_INPUT
        START:          UINT;
        G:              BOOL;
        IN0, IN1:       INT;
  END_VAR

  VAR_OUTPUT
        SEL_OUT:        INT;
        COUNT:          UINT;
  END_VAR

  VAR
        INTERNAL_COUNT: UINT;
  END_VAR

  END_FUNCTION_BLOCK
```

```
ALGORITHM INIT:
        INTERNAL_COUNT = START;
        COUNT = INTERNAL_COUNT;
END_ALGORITHM

ALGORITHM MAIN:
        IF G = 0 THEN SEL_OUT := IN0;
  ELSE
        SEL_OUT := IN1;
  END_IF
  INTERNAL_COUNT =
    INTERNAL_COUNT +1;
  COUNT = INTERNAL_COUNT;
END_ALGORITHM
```

Bsp. 9.3. Bsp. 9.1 in textueller Repräsentation (Structured Text ST der IEC 61131-3).

Das Schlüsselwort WITH verbindet ein Ereignisein- oder ausgang mit einem Datenein- oder ausgang. Ist ein solcher Ereignis-Parameter gesetzt, zeigt er die Gültigkeit der mit WITH zugeordneten Datenleitung an.

Zusammengesetzte Funktionsbausteine.
Für eine übersichtliche bzw. objektorientierte Darstellung können die bisher erläuterten *Basis-Funktionsbausteine* zu sogenannten *Zusammengesetzten Funktionsbausteinen* (engl.: Composite function block) verbunden werden, die, einmal verschmolzen, einen neuen Funktionsbaustein darstellen, wie es Abb. 9.6 zeigt.

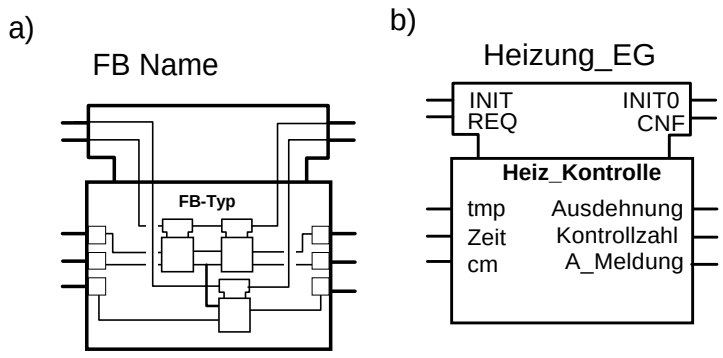


Abb. 9.6. Ein zusammengesetzter Funktionsbaustein besteht aus mehreren Funktionsbausteinen, die durch Verschaltung über eine gemeinsame Schnittstelle verfügen.
a) Beispiel: Innere Struktur eines Zusammengesetzten FBs
b) Beispielhaftes äußeres Erscheinungsbild dieses FBs

Zusammengesetzte Funktionsbausteine besitzen keine eigene Ausführungskontrolle, denn diese ergibt sich aus der Summe der Steuerungen der einzelnen FBs. In der grafische Darstellung der Abb. 9.6 a) ist daher der „FB-Kopf leer".

9.2.6 Erstellung einer Anwendung

Künftige Anwenderprogrammierer der IEC 61499 werden programmieren, indem sie vorgefertigte Funktionsbausteine konfigurieren und parametrieren.

FB	Erläuterung
Standard-FBs	• FBs mit Funktionalität wie in IEC 61131-3, • Service-Schnittstellen-FBs (standardisierte Kommunikationsdienste), • Ereignis-FBs (standardisierte Ereigniserzeugung und -verarbeitung)
Benutzerdefinierte FBs	Algorithmen und ECC-Ausführungskontrolle z.B. nach IEC 61131-3 programmiert

Tab. 9.1. Unterscheidung von Funktionsbausteinen nach IEC 61499

Die in Tab. 9.1 genannten Funktionsbausteine können als Basis-FBs oder zusammengesetzte FBs mit gemeinsamer Schnittstelle realisiert sein.

Die Ereignis-FBs stellen beispielsweise Funktionen zur Zusammenführung (Merge) und Vervielfachung (Split) oder zur Erzeugung einmaliger oder zyklisch wiederkehrender Ereignisse zur Verfügung.

Mit einem Konfigurations-Editor erfolgt die Verteilung der Bausteine auf die Ressourcen (Geräte) sowie die Verbindung der FBs untereinander.

9.3 Überblick über die Teile der IEC 61499

Die zukünftige Norm IEC 61499 wird aus zwei Teilen mit den in Tab. 9.2 zusammengefaßten Schwerpunkten bestehen ([IEC 61499-97]):

Teil	**Stichworte zum Inhalt**
1. *Architektur*	Einführung und Modellbildung, beschreibt den Geltungsbereich, definiert gemeinsame Begriffe, Spezifikation der Funktionsbausteine, Service-Schnittstellen, Konfiguration und Syntax.
2. *Modell Entwicklungszyklus*	enthält Beschreibungen zur Unterstützung des Lebenszyklus von verteilten Programmen. Dieses Dokument befindet sich noch in der Entwurfsphase (Stand 9/1999).

Tab. 9.2. Aufbau und Inhalte der zukünftigen Norm IEC 61499

10 Inhalt der beiliegenden CD

10.1 IEC-Programmiersysteme STEP 7 und OpenPCS

Auf CD liegen diesem Buch folgende Informationen und Programme bei:

1) **STEP 7 Demo Software**[1] als Demo-Version zur SPS-Programmierung nach IEC 61131-3 mit den Sprachen: AWL, KOP, FUP, S7-GRAPH, S7-SCL, CFC und S7-HiGraph; ablauffähig unter Windows 95/98 sowie Windows NT,
2) **Open PCS** als Demo-Version zur SPS-Programmierung nach IEC 61131-3 mit den Sprachen: AWL, KOP, FBS und ST sowie Smart PLC; ablauffähig unter Windows 3.x, Windows 95/98 sowie Windows NT,
3) **AWL-Beispiele** dieses Buchs,
4) **Einkaufsberater** für Programmiersysteme nach IEC 61131-3.

Die auf CD mitgelieferte Datei LIESMICH.TXT beinhaltet wichtige Informationen zur Installation und Benutzung der Dateien und Programme. Sie zeigt insbesondere, wie die Dateien auf Festplatte kopiert werden können und gibt Hinweise zur Benutzung der Beispiele und des Einkaufsberaters.

Die Datei LIESMICH.TXT ist eine ASCII-Datei und kann mit einem üblichen Editor (z.B. in Windows) gelesen werden.

Die übrigen Dateien der beiden Programmiersysteme sind in einem selbstentpackenden Format bzw. als Installationsroutinen hinterlegt, d.h. sie können nicht direkt gelesen werden, sondern müssen sich erst selbst dekomprimieren oder installieren. Zusätzliche Software ist nicht erforderlich.

---

[1] FUP entspricht FBS, S7-GRAPH entspricht AS, S7-SCL entspricht ST

Demo-Versionen STEP 7 (Fa. Siemens) und Open PCS (Fa. infoteam).
Mit Hilfe der Demo-Versionen zweier ausgewählter Programmiersysteme kann der Leser sämtliche Beispiele des Buchs sowie eigene programmieren, verändern, erweitern und prüfen lassen, um die SPS-Programmierung nach IEC 61131-3 zu trainieren.

STEP 7 enthält mit CFC und S7-HiGraph Werkzeuge zur FB-Verschaltung und Programmierung mit Zustandsgraphen. OpenPCS liegt ein Ausführungspaket (Smart PLC) bei, das zusätzlich die Ausführung eines SPS-Programms auf dem PC erlaubt (Offline-Simulation).

Hinweise zum Einsatz der Vollversionen sowie Bezugsmöglichkeiten (auch: Hardware) zu den einzelnen Programmiersystemen sind den jeweiligen Unterverzeichnissen der CD zu entnehmen.

Die Verfasser zeichnen für Inhalt und Funktionsweise dieser Demo-Versionen nicht verantwortlich. Diese Demo-Versionen enthalten Einschränkungen gegenüber der Funktionalität der Produkte. Ihre Verwendung ist nur im Zusammenhang mit dem Buch zu Lern- und Schulungszwecken gestattet (bestimmungsgemäßer Gebrauch).

AWL - Beispiele
Um die Programmteile dieses Buchs nicht abtippen zu müssen, liegen die wichtigsten AWL-Beispiele auf CD vor. Nähere Informationen dazu sind ebenfalls im LIESMICH.TXT zu finden.

10.2 Einkaufsberater für SPS-Programmiersysteme nach IEC 61131-3

Diesem Buch liegt ein Einkaufsberater (CD) als Datei im Format „Word für Windows ab Version 6.0“ bei.

Inhalt des Einkaufsberaters (Datei **Eink-Ber.doc**):

> **Einkaufsberater für SPS-Programmiersysteme nach IEC 61131-3**
>
> **Checklisten zur Bewertung von SPS-Programmiersystemen**
> **Benutzung der Checklisten**
> **Checklisten für SPS-Programmiersysteme**
> Konformität zur IEC 61131-3
> Sprachumfang, Rück- und Querübersetzbarkeit
> Werkzeuge
> Arbeitsumgebung, Offenheit und Dokumentation
> Allgemeines, Kosten

Dieser Einkaufsberater besteht im Kern aus „Checklisten“ genannten Tabellen, mit denen eine objektive Bewertung von SPS- Programmiersystemen möglich ist, die normkonform zur IEC 61131-3 sind. Der Gebrauch dieser Checklisten wird zunächst ausführlich erläutert, bevor die einzelnen Bewertungskriterien für SPS-Programmiersysteme vorgestellt werden.

Durch Kopieren der Datei kann sowohl eine mehrfache Produktzusammenstellung als auch eine individuelle Bearbeitung der Checklisten vorgenommen werden. Die Tabellen sind in drei verschiedenen Dateiformaten hinterlegt:

1) Microsoft Word für Windows, ab Version 6.0 (Datei TABELLEN.DOC),
2) Microsoft Excel für Windows, ab Version 4.0 (Datei TABELLEN.XLS),
3) als ANSI- Text (Datei TABELLEN.ANS),
4) als ASCII-Text (Datei TABELLEN.ASC).

Die Excel-Version besitzt den Vorteil, daß sämtliche Berechnungen automatisch erfolgen.

A Standard-Funktionen

Dieser Anhang stellt die in Kap. 2 exemplarisch beschriebenen SPS-Standard-Funktionen vollständig zusammen. Für jede Standard-Funktion der IEC 61131-3 werden hier ihre:

- Grafische Deklaration,
- (Semantische) Beschreibung,
- Spezifizierung einiger Funktionen in Strukturiertem Text (ST)

angegeben.

Standard-Funktionen (Std.-FUN) besitzen Eingangsvariablen (Formalparameter) sowie einen Funktionswert (der Rückgabewert der Funktion). Einige Eingangsvariablen sind namenlos.

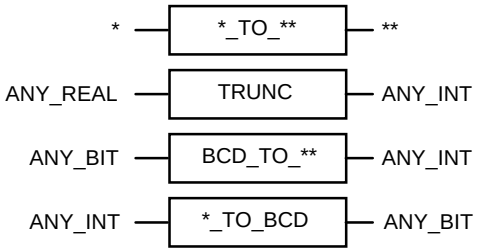
Um ihr Funktionsverhalten zu beschreiben, gelten folgende Vereinbarungen:

- eine einzelne Eingangsvariable ohne Namen wird mit „IN“ bezeichnet,
- mehrere Eingangsvariablen ohne Namen werden mit „IN1, IN2, ... INn“ durchnumeriert,
- der Funktionswert wird mit „F“ bezeichnet.

Zur Beschreibung werden u.a. Allgemeine Datentypen (wie ANY oder ANY_BIT) verwendet, deren Bedeutung in Abschn. 3.4.3 erläutert wird und die in Tab. 3.9 zusammengefaßt werden. Die Bezeichnung ANY steht dabei als Abkürzung für einen beliebigen der Datentypen: ANY_BIT, ANY_NUM, STRING, ANY_DATE oder TIME.

Viele Standard-Funktionen besitzen einen textuellen Namen sowie eine alternative Darstellung mit einem Symbol (z.B. ADD und „+“). In den Abbildungen und Tabellen werden jeweils beide Varianten angegeben.

A.1 Funktionen zur Typwandlung



*	Datentyp des Eingangs, z.B. REAL
**	Datentyp des Funktionswerts, z.B. INT
*_TO_**	Funktionsname, z.B. REAL_TO_INT

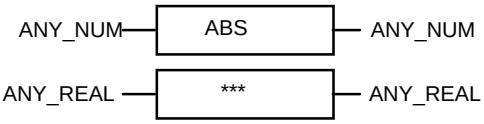
Abb. A.1. Grafische Deklarationen der Funktionen zur Typumwandlung

Diese Standard-Funktionen wandeln die Eingangsvariablen in den Datentyp ihres Funktionswerts (Typumwandlung, Konvertierung).

Name	**Funktion / Beschreibung**
*_TO_**	Bei der Umwandlung von REAL-Werten in INT-Werte wird zur nächsten ganzzahligen Zahl auf- bzw. abgerundet, halbe Anteile wie 0,5 oder 0,05 werden dabei aufgerundet.
TRUNC	Mit dieser Funktion werden die Stellen eines REAL-Wertes hinter dem Komma abgeschnitten, um einen ganzzahligen Wert zu bilden.
BCD	Die Ein- bzw. Ausgangswerte vom Typ ANY_BIT stellen BCD-kodierte Bitfolgen für die Datentypen BYTE, WORD, DWORD und LWORD dar. Die BCD-Kodierung wird durch die IEC 61131-3 nicht festgelegt, sie ist implementierungsabhängig.

Tab. A.1. Beschreibung der Funktionen zur Typwandlung

A.2 Numerische Funktionen



*** Kürzel für: SQRT, LN, LOG, EXP,
SIN, COS, TAN,
ASIN, ACOS, ATAN

Abb. A.2. Grafische Deklarationen der Numerischen Funktionen

Name	**Funktion**	**Beschreibung**
ABS	Absolutwert	F := | IN |
SQRT	Quadratwurzel	F := √IN
LN	Natürlicher Logarithmus	F := $\log_e$ (IN)
LOG	Logarithmus Basis 10	F := $\log_{10}$ (IN)
EXP	Exponent Basis e	F := e^{IN}
SIN	Sinus, IN in Bogenmaß	F := SIN (IN)
COS	Cosinus, IN in Bogenmaß	F := COS (IN)
TAN	Tangens, IN in Bogenmaß	F := TAN (IN)
ASIN	Arcsin, Hauptwert	F := ARCSIN (IN)
ACOS	Arccos, Hauptwert	F := ARCCOS (IN)
ATAN	Arctan, Hauptwert	F := ARCTAN (IN)

Tab. A.2. Beschreibung der Numerischen Funktionen

A.3 Arithmetische Funktionen

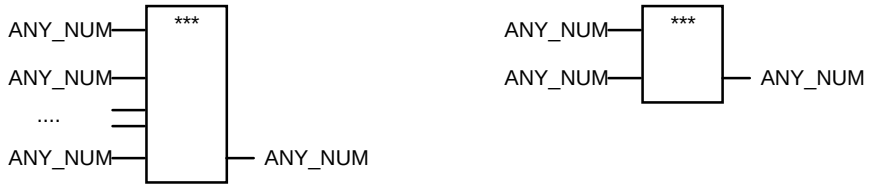


Abb. A.3. Grafische Deklarationen der Arithmetischen Funktionen ADD, MUL, SUB und DIV

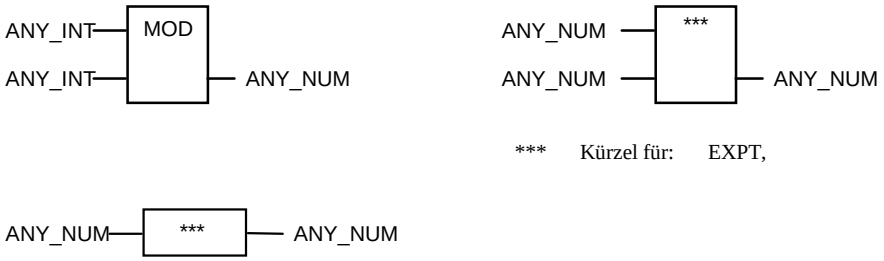


Abb. A.4. Grafische Deklarationen der Arithmetischen Std-FUN MOD, EXPT und MOVE

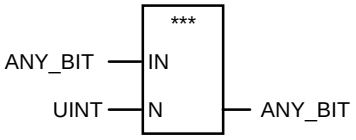
Name	Symbol	Funktion	Beschreibung
ADD	+	Addition	F := IN1 + IN2 + ... + INn
MUL	*	Multiplikation	F := IN1 * IN2 * ... * INn
SUB	-	Subtraktion	F := IN1 - IN2
DIV	/	Division	F := IN1 / IN2
MOD		Restbildung	F := IN1 - (IN1/ IN2)*IN2
EXPT	**	Exponentiation	F := $IN1^{IN2}$
MOVE	:=	Zuweisung	F := IN

Tab. A.3. Beschreibung der arithmetischen Funktionen

Bei der Division von ganzen Zahlen muß das Ergebnis wieder eine ganze Zahl sein, ggf. wird das Ergebnis in Richtung Null abgeschnitten.

Falls der Eingangsparameter IN2 Null ist, wird zur Laufzeit ein Fehler mit Fehlerursache „Division durch Null“ gemeldet, vgl. Anh. E.

A.4 Bitschiebe-Funktionen



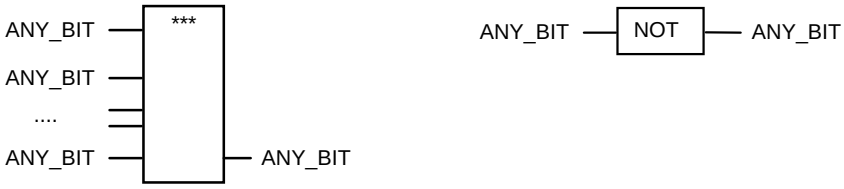
*** Kürzel für: SHL, SHR, ROL, ROR

Abb. A.5. Grafische Deklarationen der Bitschiebe-Funktionen SHL, SHR, ROR und ROL

Name	Funktion	Beschreibung
SHL	Schieben nach links	IN um N Bits nach links schieben, von rechts mit Nullen füllen
SHR	Schieben nach rechts	IN um N Bits nach rechts schieben, von links mit Nullen füllen
ROR	Rotieren nach rechts	IN um N Bits ringförmig nach rechts schieben
ROL	Rotieren nach links	IN um N Bits ringförmig nach links schieben

Tab. A.4. Beschreibung der Bitschiebe-Funktionen

A.5 Bitweise Boolesche Funktionen



*** Kürzel für: AND, &, OR, >=1, XOR, =2k+1

Abb. A.6. Grafische Deklarationen der Bitweise Booleschen Funktionen AND, OR, XOR und NOT

Name	Symbol	Funktion	Beschreibung
AND	&	Bitweise UND	F := IN1 & IN2 & ... & INn
OR	>=1	Bitweise ODER	F := IN1 v IN2 v ... v INn
XOR	=2k+1	Bitweise EXODER	F := IN1 XOR IN2 XOR ... XOR INn
NOT		Negation	F := ¬ IN

Tab. A.5. Beschreibung der Bitweise Booleschen Funktionen

Die Verknüpfung zwischen den Eingangsparametern erfolgt bitweise, d.h. jede Bitstelle eines Eingangs wird mit der entsprechenden Bitstelle des anderen Eingangs zur gleichen Bitstelle des Funktionswerts verknüpft.

Die grafische Darstellung einer Invertierung kann auch durch einen Kreis „o“ am booleschen Ein- oder Ausgangs einer Funktion erfolgen.

A.6 Auswahl-Funktionen für Max., Min. und Grenzwert

*** Kürzel für: MIN, MAX

Abb. A.7. Grafische Deklarationen der Auswahlfunktionen MAX, MIN und LIMIT

Name	**Funktion**	**Beschreibung**
MAX	Maximum-Bildung	F := MAX (IN1, IN2, ... , INn)
MIN	Minimum-Bildung	F := MIN (IN1, IN2, ... , INn)
LIMIT	Begrenzung	F := MIN (MAX (IN, MN), MX)

Tab. A.6. Beschreibung der Auswahlfunktionen MAX, MIN und LIMIT

Diese drei Standard-Funktionen werden in Bsp. A.1 und Bsp. A.2 durch Deklaration in ST spezifiziert.

```
FUNCTION      MAX : ANY          (* Maximum-Bildung; ANY steht für INT, ... *)
  VAR_INPUT   IN1, IN2, ... INn  : ANY;     END_VAR
  VAR         Elem               : ANY;     END_VAR
  IF IN1 > IN2 THEN        (* erster Vergleich *)
      Elem    := IN1;
  ELSE
      Elem    := IN2;
  END_IF;
  IF IN3 > Elem THEN       (* nächster Vergleich *)
      Elem    := IN3;
  END_IF;
  ...
  IF INn > Elem THEN       (* letzter Vergleich *)
      Elem    := INn;
  END_IF;
  MAX    :=   Elem;        (* Schreiben des Funktionswerts *)
END_FUNCTION
```

Bsp. A.1. Spezifizierung der Auswahlfunktion MAX in ST; für MIN sämtliche „>“ durch „<“ ersetzen.

Die Spezifizierung der Minimum-Funktion MIN kann man aus MAX dadurch erhalten, daß in Bsp. A.1 sämtliche Vorkommen von „>“ durch „<“ ersetzt werden.

```
FUNCTION      LIMIT : ANY        (* Grenzwert-Bildung *)
  VAR_INPUT
    MN   :   ANY;
    IN   :   ANY;
    MX   :   ANY;
  END_VAR
  MAX   :=  MIN ( MAX ( IN, MN), MX);        (* Aufrufe von MIN von MAX *)
END_FUNCTION
```

Bsp. A.2. Spezifizierung der Auswahlfunktion LIMIT in ST

A.7 Auswahl-Funktionen für Binäre Auswahl und Multiplexer

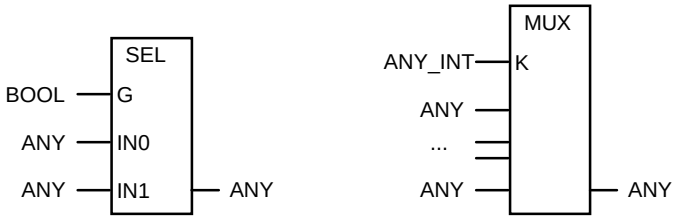


Abb. A.8. Grafische Deklarationen der Auswahlfunktionen SEL und MUX

Name	Funktion	Beschreibung
SEL	Binäre Auswahl	F := IN0, falls G = 0, IN1 sonst
MUX	Multiplexer	F := INi, falls K = i

Tab. A.7. Beschreibung der Auswahlfunktionen SEL und MUX

Diese beiden Standard-Funktionen werden im folgenden durch Deklaration in ST spezifiziert:

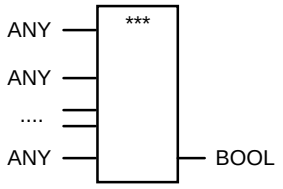
```
FUNCTION      SEL : ANY          (* Binäre Auswahl *)
  VAR_INPUT
    G     :   BOOL;
    IN0   :   ANY;
    IN1   :   ANY;
  END_VAR
  IF G = 0 THEN
      SEL    :=  IN0;      (* Auswahl des oberen Eingangs *)
  ELSE
      SEL    :=  IN1;      (* Auswahl des unteren Eingangs *)
  END_IF;
END_FUNCTION
```

Bsp. A.3. Spezifizierung der Auswahlfunktion SEL in ST

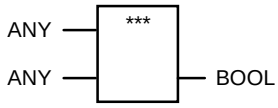
```
FUNCTION      MUX : ANY          (* Multiplexer *)
  VAR_INPUT
    K     :   ANY_INT;
    IN0   :   ANY;
    IN1   :   ANY;
    ...
    INn   :   ANY;
  END_VAR
  IF (K < 0) OR (K>n) THEN
      ... Fehlermeldung Nr....;        (* K negativ oder zu groß *)
  END_IF;
  CASE  K  OF
      0: MUX := IN0;       (* Auswahl des oberen Eingangs *)
      1: MUX := IN1;       (* Auswahl des zweiten Eingangs *)
      ...
      n: MUX := INn;       (* Auswahl des untersten Eingangs *)
  END_CASE;
END_FUNCTION
```

Bsp. A.4. Spezifizierung der Auswahlfunktion MUX in ST

A.8 Vergleichs-Funktionen



*** Kürzel für: GT, >, GE, >=,
LT, <, LE, <=,
EQ, =



*** Kürzel für: NE, <>

Abb. A.9. Grafische Deklarationen der Vergleichsfunktionen GT, GE, LT, LE, EQ, NE

Name	**Funktion**	**Beschreibung**
GT	Vergleich auf „> “	F := 1, falls INi > IN(i+1), 0 sonst
GE	Vergleich auf „> =“	F := 1, falls INi >= IN(i+1), 0 sonst
LT	Vergleich auf „< “	F := 1, falls INi < IN(i+1), 0 sonst
LE	Vergleich auf „< =“	F := 1, falls INi <= IN(i+1), 0 sonst
EQ	Vergleich auf „= “	F := 1, falls INi = IN(i+1), 0 sonst
NE	Vergleich auf „<>“	F := 1, falls INi <> IN(i+1), 0 sonst

Tab. A.8. Beschreibung der Vergleichsfunktionen

Diese Standard-Funktionen werden in Bsp. A.5 stellvertretend durch Deklaration von GT in ST spezifiziert, von der die übrigen leicht abgeleitet werden können.

```
FUNCTION        GT : BOOL           (* Vergleich auf größer *)
  VAR_INPUT     IN1, IN2, ... INn   : ANY;      END_VAR
  IF        (IN1 > IN2)
  AND       (IN2 > IN3)
  ...
  AND       (IN(n-1) > INn) THEN
        GT :=   TRUE;           (* Eingänge sind „sortiert“: monoton steigend *)
  ELSE
        GT :=   FALSE;          (* Bedingung nicht erfüllt *)
  END_IF;
END_FUNCTION
```

Bsp. A.5. Spezifizierung der Vergleichsfunktion GT in ST

A.9 Funktionen für Zeichenfolgen

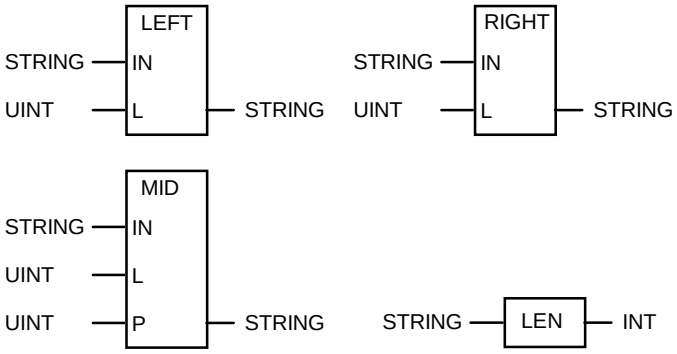


Abb. A.10. Grafische Deklarationen der Zeichenfolge-Funktionen LEFT, RIGHT, MID und LEN

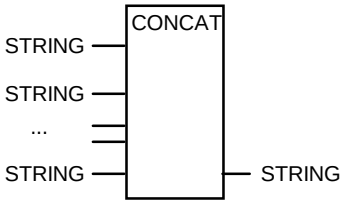


Abb. A.11. Grafische Deklarationen der Zeichenfolge-Funktion CONCAT

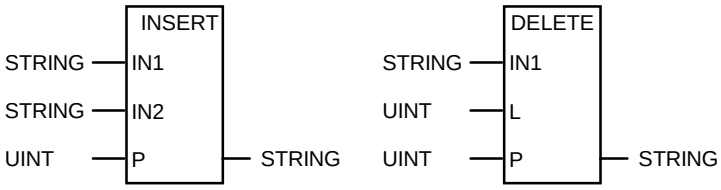


Abb. A.12. Grafische Deklarationen der Zeichenfolge-Funktionen INSERT und DELETE

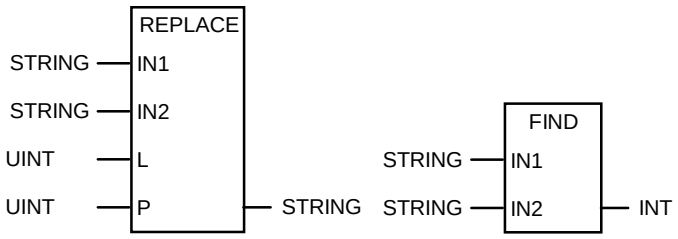


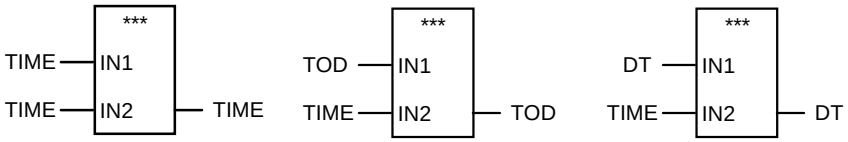
Abb. A.13. Grafische Deklarationen der Zeichenfolge-Funktionen REPLACE und FIND

Name	**Funktion**	**Beschreibung**
LEN	ermittelt die Länge einer Zeichenfolge	F := Anzahl der Zeichen in IN
LEFT	Anfangsabschnitt einer Zeichenfolge	F := Anfangsabschnitt mit L Zeichen
RIGHT	Endeabschnitt einer Zeichenfolge	F := Endeabschnitt mit L Zeichen
MID	Mittelabschnitt einer Zeichenfolge	F := Mittelabschnitt ab Position P mit L Zeichen
CONCAT	Aneinanderreihung von Zeichenfolgen	F := Gesamt-Zeichenfolge
INSERT	Einfügen einer Zeichenfolge in eine andere	F := Gesamt-Zeichenfolge mit neuem Teil ab Position P
DELETE	Löscht Abschnitt in einer Zeichenfolge	F := Rest-Zeichenfolge mit gelöschtem Teil (L Zeichen) ab Position P
REPLACE	Ersetzt einen Abschnitt in einer Zeichenfolge durch einen anderen	F := Gesamt-Zeichenfolge mit ersetztem Teil (L Zeichen) ab Position P
FIND	ermittelt die Position eines Abschnitts in einer Zeichenfolge	F := Index der gefundenen Position, sonst 0

Tab. A.9. Beschreibung der Zeichenfolge-Funktionen

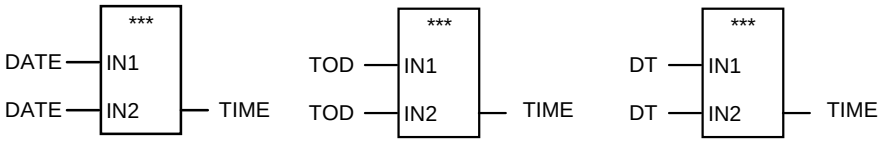
Die Position eines Zeichens innerhalb einer Zeichenfolge vom Typ STRING wird beginnend mit „1“ angegeben.

A.10 Funktionen für Datentypen der Zeit



*** Kürzel für: ADD, +, SUB, -

Abb. A.14. Grafische Deklarationen der gemeinsamen Zeit-Funktionen für Addition und Subtraktion



*** Kürzel für: SUB, -

Abb. A.15. Grafische Deklarationen der zusätzlichen Zeit-Funktionen für Subtraktion

*** Kürzel für: MUL, *, DIV, /

Abb. A.16. Grafische Deklarationen der Zeit-Funktionen für MUL, DIV und CONCAT

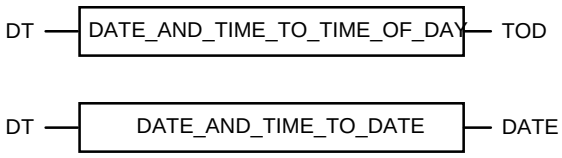


Abb. A.17. Grafische Deklarationen der Zeit-Funktionen zur Typumwandlung

Die Abkürzungen TOD und DT können gleichwertig anstelle der längeren Schlüsselworte TIME_OF_DAY bzw. DATE_AND_TIME verwendet werden.

A.11 Funktionen für Datentypen der Aufzählung

Die Standard-Funktionen SEL, MUX, EQ und NE können ebenfalls für Aufzählungs-Datentypen verwendet werden. Sie sind in diesen Fällen sinngemäß wie für ganze Zahlen anzuwenden (Werte von Aufzählungen entsprechen vom Programmiersystem „codierten“ Konstanten).

B Standard-Funktionsbausteine

Dieser Anhang stellt die in Kap. 2 exemplarisch beschriebenen Standard-Funktionsbausteine vollständig zusammen. Für jeden Standard-FB der IEC 61131-3 werden hier seine:

- Grafische Deklaration,
- (Semantische) Beschreibung,
- Spezifizierung einiger FBs in Strukturiertem Text (ST)

angegeben.

In diesem Buch werden die Ein- und Ausgänge der Std.-FBs mit Namen versehen, die die aktuelle Fassung der IEC 61131-3 derzeit vorschreibt ([IEC 1131-3-94]). Dabei wurde noch nicht berücksichtigt, daß aufgrund der Aufrufmöglichkeit der FBs als „AWL-Operatoren“ Konflikte entstehen. Diese werden entweder zu einer Änderung der Norm selbst oder zu geänderter Nomenklatur bei der Realisierung von Programmiersystemen nach IEC 61131-3 führen.

Diese Konflikte treten auf für die AWL-Operatoren (vgl. Abschn. 4.1.) LD, R und S, die zur Überprüfung der korrekten Programmsyntax von den FB-Eingängen LD, R und S unterschieden werden müssen. Diese drei Formaloperanden könnten zukünftig beispielsweise mit LOAD, RESET und SET benannt werden.

B.1 Bistabile Elemente (Flip-Flops)

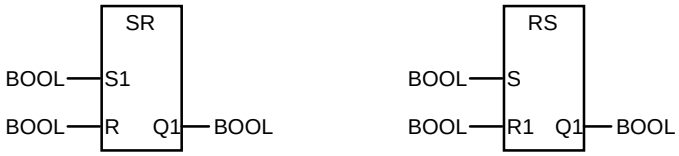


Abb. B.1. Grafische Deklarationen der Funktionsbausteine SR und RS

```
FUNCTION_BLOCK      SR          (* FlipFlop mit vorrangigem Setzen *)
  VAR_INPUT
    S1    :   BOOL;
    R     :   BOOL;
  END_VAR
  VAR_OUTPUT
    Q1    :   BOOL;
  END_VAR
  Q1      :=  S1 OR ( NOT R AND Q1);
END_FUNCTION_BLOCK

FUNCTION_BLOCK      RS          (* FlipFlop mit vorrangigem Rücksetzen *)
  VAR_INPUT
    S     :   BOOL;
    R1    :   BOOL;
  END_VAR
  VAR_OUTPUT
    Q1    :   BOOL;
  END_VAR
  Q1      :=  NOT R1 AND ( S OR Q1);
END_FUNCTION_BLOCK
```

Bsp. B.1. Spezifizierung der Funktionsbausteine SR und RS in ST

Diese beiden Flip-Flops realisieren Vorrangiges Setzen und Rücksetzen.

B.2 Flankenerkennung

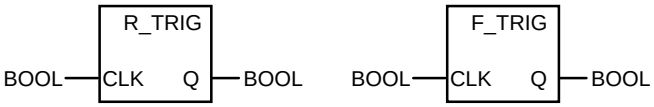


Abb. B.2. Grafische Deklarationen der Funktionsbausteine R_TRIG und F_TRIG

```
FUNCTION_BLOCK      R_TRIG     (* steigende Flanke *)
  VAR_INPUT
    CLK  :    BOOL;
  END_VAR
  VAR_OUTPUT
    Q    :    BOOL;
  END_VAR
  VAR RETAIN
    MEM :     BOOL := 0;                  (* Flankenmerker initialisieren *)
  END_VAR
  Q       :=  CLK AND NOT MEM;            (* steigende Flanke erkennen *)
  MEM     :=  CLK;                        (* Flankenmerker rücksetzen *)
END_FUNCTION_BLOCK

FUNCTION_BLOCK      F_TRIG     (* fallende Flanke *)
  VAR_INPUT
    CLK  :    BOOL;
  END_VAR
  VAR_OUTPUT
    Q    :    BOOL;
  END_VAR
  VAR RETAIN
    MEM :     BOOL := 1;                  (* Flankenmerker initialisieren *)
  END_VAR
  Q       :=  NOT CLK AND NOT MEM;        (* fallende Flanke erkennen *)
  MEM     :=  NOT CLK;                    (* Flankenmerker rücksetzen *)
END_FUNCTION_BLOCK
```

Bsp. B.2. Spezifizierung der Funktionsbausteine R_TRIG und F_TRIG in ST

Bei den FBs R_TRIG und F_TRIG ist zu beachten, daß sie bereits beim **ersten** Aufruf eine „Flanke“ erkennen, wenn der Eingang bei R_TRIG auf TRUE und bei F_TRIG auf FALSE liegt.

B.3 Zähler

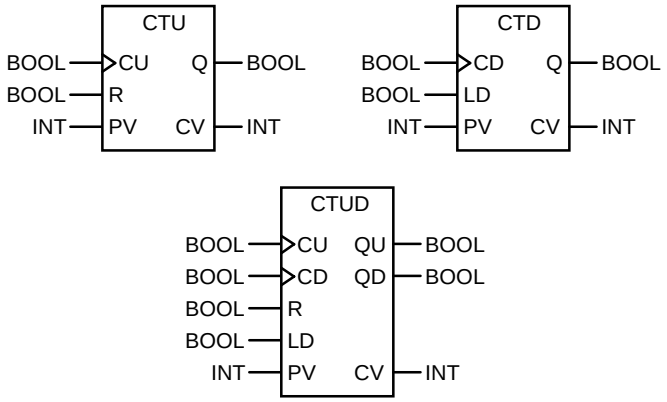


Abb. B.3. Grafische Deklarationen der Funktionsbausteine CTU, CTD und CTUD

```
FUNCTION_BLOCK        CTU      (* Vorwärts-Zähler *)
  VAR_INPUT
    CU    :    BOOL    R_EDGE;    (* CU mit steigender Flanke *)
    R     :    BOOL;
    PV    :    INT;
  END_VAR
  VAR_OUTPUT
    Q     :    BOOL;
    CV    :    INT;
  END_VAR
  IF  R  THEN                       (* Zähler rücksetzen *)
        CV      :=   0;
  ELSIF  CU AND ( CV < PV)  THEN
        CV      :=   CV + 1;        (* hochzählen *)
  ENDIF;
  Q     :=     (CV >= PV);          (* Grenzwert erreicht *)
END_FUNCTION_BLOCK
```

Bsp. B.3. Spezifizierung der Funktionsbausteine CTU und CTD in ST (wird fortgesetzt)

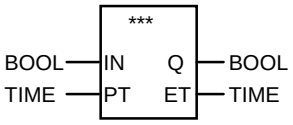
```
FUNCTION_BLOCK        CTD      (* Rückwärts-Zähler *)
  VAR_INPUT
    CD   :   BOOL    R_EDGE;   (* CD mit steigender Flanke *)
    LD   :   BOOL;
    PV   :   INT;
  END_VAR
  VAR_OUTPUT
    Q    :   BOOL;
    CV   :   INT;
  END_VAR
  IF  LD  THEN                  (* Zähler rücksetzen *)
        CV      :=  PV;
  ELSIF  CD AND ( CV > PV)  THEN
        CV      :=  CV - 1;     (* runterzählen *)
  ENDIF;
  Q     :=    (CV <= 0);        (* Null erreicht *)
END_FUNCTION_BLOCK
```

Bsp. B.3. (Fortsetzung)

```
FUNCTION_BLOCK        CTUD     (* Vorwärts/Rückwärts-Zähler *)
  VAR_INPUT
    CU   :   BOOL    R_EDGE;   (* CU mit steigender Flanke *)
    CD   :   BOOL    R_EDGE;   (* CD mit steigender Flanke *)
    R    :   BOOL;
    LD   :   BOOL;
    PV   :   INT;
  END_VAR
  VAR_OUTPUT
    QU   :   BOOL;
    QD   :   BOOL;
    CV   :   INT;
  END_VAR
  IF  R  THEN                   (* Zähler vorrangig rücksetzen *)
        CV   :=    0;
  ELSIF  LD  THEN
        CV   :=    PV;                    (* auf Zählwert setzen *)
  ELSIF  CU AND ( CV < PV)  THEN
        CV   :=    CV + 1;      (* hochzählen *)
  ELSIF  CD AND ( CV > PV)  THEN
        CV   :=    CV - 1;      (* runterzählen *)
  ENDIF;
  QU   :=    (CV >= PV);        (* Grenzwert erreicht *)
  QD   :=    (CV <= 0);         (* Null erreicht *)
END_FUNCTION_BLOCK
```

Bsp. B.4. Spezifizierung des Funktionsbausteins CTUD in ST

B.4 Zeitgeber (Zeiten)



*** Kürzel für: TON, T---0, TOF, 0---T, TP

Abb. B.4. Grafische Deklarationen der Funktionsbausteine TON, TOF und TP

Die Spezifizierung der Zeitgeber-Elemente TP, TON und TOF wird im folgenden anhand von Zeitdiagrammen vorgenommen.

Dieses Zeitverhalten setzt jeweils voraus, daß die Zykluszeit des periodischen SPS-Programms, in dem die Zeit verwendet wird, vernachlässigbar gering zur Zeitdauer PT ist, wenn die Zeit im Zyklus nur einmal aufgerufen wird.

In den Diagrammen wird das an den Ausgängen Q und ET zu beobachtende Verhalten der Zeit in Abhängigkeit vom Eingang IN dargestellt. Die Zeitachse läuft dabei von links nach rechts und ist mit „t“ beschriftet. Die booleschen Variablen IN und Q wechseln zwischen „0“ und „1“ und für ET wird der Verlauf des anwachsenden Zeitwerts dargestellt.

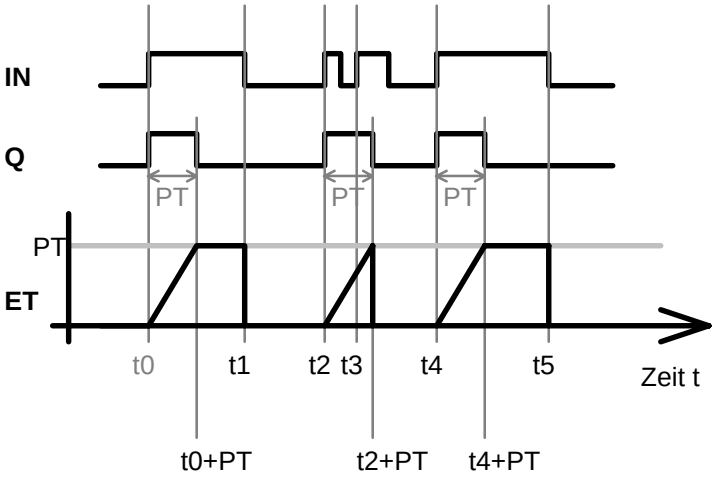


Abb. B.5. Zeitverhalten für Zeitimpuls TP abhängig vom Eingang IN

Der Std-FB „TP“ stellt einen Impulsgeber dar, der bei steigender Flanke am IN-Eingang am Ausgang Q einen konstant langen Impuls liefert. Am Ausgang ET kann jederzeit die bis dahin aufgelaufene Zeit abgefragt werden.

Wie man in Abb. B.5 erkennt, sind Zeiten vom Typ TP nicht „re-triggerbar“. Falls die Abstände zwischen den Eingangsimpulsen auf IN kürzer sind als die voreingestellte Zeitdauer, bleibt die Impulsdauer dennoch konstant; siehe Zeitraum [t2; t2+PT]. Die Zeitdauer beginnt also *nicht* mit jeder steigenden Flanke an IN.

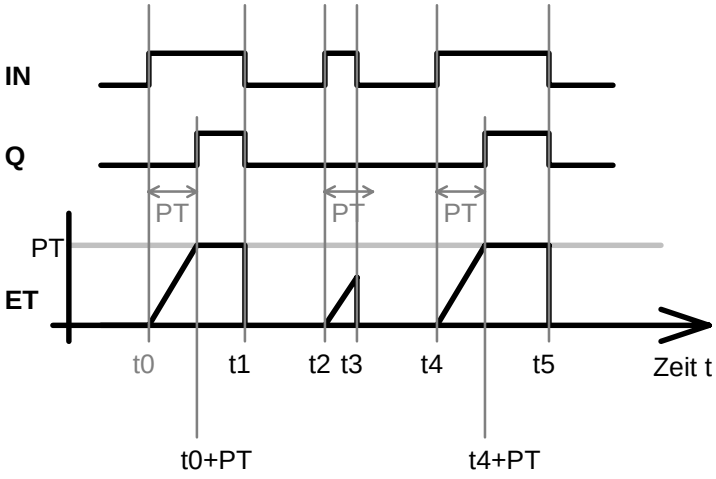


Abb. B.6. Zeitverhalten für Einschalt-Verzögerung TON abhängig vom Eingang IN

Die Einschaltverzögerung TON liefert an Q zeitverzögert den Eingangswert IN, wenn für IN eine steigende Flanke erkannt wird. Ist der Eingang IN jedoch nur für einen kurzen Impuls (kleiner PT) auf „1“, wird die Zeit für diese Flanke nicht mehr gestartet.

Am Ausgang ET kann die aufgelaufene Zeit abgelesen werden.

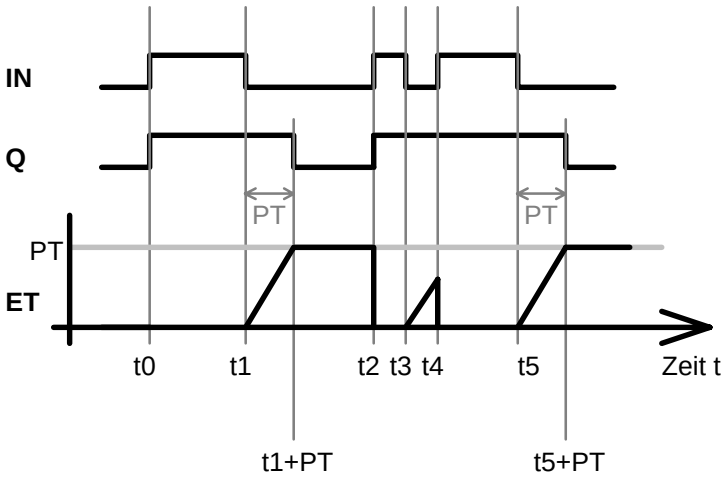


Abb. B.7. Zeitverhalten für Ausschalt-Verzögerung TOF abhängig vom Eingang IN

Die Ausschaltverzögerung ist die inverse Funktion zu TON, d.h. sie verzögert in derselben Weise eine fallende Flanke, wie TON eine steigende.

Wie sich die Zeit verhält, wenn PT während der Zeitoperation verändert wird, ist implementierungsabhängig.

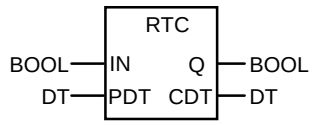


Abb. B.8. Grafische Deklarationen des Funktionsbausteins RTC (Echtzeituhr)

```
FUNCTION_BLOCK      RTC        (* Echtzeituhr *)
  VAR_INPUT
    IN    :   BOOL;
    PDT   :   DATE_AND_TIME;
  END_VAR
  VAR_OUTPUT
    Q     :   BOOL;
    CDT   :   DATE_AND_TIME;
  END_VAR
  VAR
    IN_Flanke     : BOOL  R_EDGE;
    Aktuelle_Zeit : DATE_AND_TIME;        (* SPS-Hardware-Uhr *)
  END_VAR
  IF  IN_Flanke (IN)  THEN                (* Flanke erkannt? *)
      Aktuelle_Zeit := PDT;                       (* Uhr in der SPS laden *)
  ENDIF;
  CDT     :=  Aktuelle_Zeit;              (* undefiniert ohne Flanke an IN *)
  Q       :=  IN;                         (* Q=1: kein Fehler *)
END_FUNCTION_BLOCK
```

Bsp. B.5. Spezifizierung der Funktionsbausteins RTC in ST

Die Echtzeituhr RTC liefert die Aktuelle Uhrzeit inklusive Datum. Bei steigender Flanke an IN wird die Uhrzeit gesetzt und läuft, solange IN „1“ bleibt. Der Eingang IN könnte in einem realen SPS-System anzeigen, daß die zentrale Stromversorgung aktiv ist, die Uhr also laufen kann. Nach Spannungsausfall bzw. Neustart des Systems würde die Uhrzeit mit einem „0→1“-Übergang gesetzt werden. Falls am Eingang IN keine steigende Flanke auftritt, gilt CDT als „undefiniert“.

Mit AktuelleZeit ist hier (informell) die zentrale Uhrzeit in der SPS gemeint, die per Software oder üblicherweise als Hardware-Uhr läuft. Die Realisierung von AktuelleZeit ist dementsprechend implementierungsabhängig.

C AWL-Beispiele

Dieser Anhang beinhalte für jeden POE-Typ jeweils ein ausführliches Beispiel zur SPS-Programmierung nach IEC 61131-3, um die Erläuterungen in Kap. 2 und Kap. 4 zu unterstützen.

Diese Beispiele sind auf der diesem Buch beiliegenden CD zu finden.

C.1 Beispiel für FUNCTION

Die Funktion ByteExtr extrahiert das obere oder untere Byte eines Eingangsworts und gibt es als Funktionswert zurück.

```
FUNCTION  ByteExtr : BYTE     (* Extraktion Byte aus Wort *)
                              (* Beginn Deklarationsteil *)
VAR_INPUT                     (* Eingangsvariablen *)
  Wort   :   WORD;            (* Wort besteht aus oberem + unterem Byte *)
  Oben   :   BOOL;            (* TRUE: oberes Byte, sonst unteres nehmen *)
END_VAR
                              (* Beginn Anweisungsteil *)
LD        Oben                (* Oberes oder unteres Byte extrahieren? *)
EQ        FALSE               (* unteres? *)
JMPCN     ObByte              (* Sprung bei Extraktion des oberen Bytes *)
LD        Wort                (* Wort laden *)
JMP       Ende                (* nix zu tun *)
ObByte:                       (* Sprungmarke *)
LD        Wort                (* Wort laden *)
SHR       8                   (* oberes Byte 8 Bits nach unten schieben *)
Ende:                         (* fertig! *)
WORD_TO_BYTE                  (* Umwandlung für Typverträglichkeit *)
ST        ByteExtr            (* Zuweisung an den Funktionswert *)
RET                           (* Rücksprung mit Funktionswert in AE *)
                              (* FUN-Ende *)
END_FUNCTION
```

Bsp. C.1. Beispiel für die Deklaration einer Funktion in AWL

Die Funktion ByteExtr in Bsp. C.1 besitzt den Eingangsparameter „Wort“ vom Typ WORD und den booleschen Eingang Oben. Der Rückgabewert der Funktion ist vom Typ BYTE. Diese Funktion benötigt keine lokalen Variablen, weshalb der Deklarationsteil VAR ... VAR_END fehlt.

Der Rückgabewert steht im aktuellen Ergebnis (AE), wenn die Funktion mit RET zum Aufrufer zurückkehrt. Das AE ist an dieser Stelle (Sprungmarke Ende:) vom Datentyp WORD, denn Wort wurde zuvor geladen, der Funktionswert nach der Typkonvertierung vom Typ BYTE.

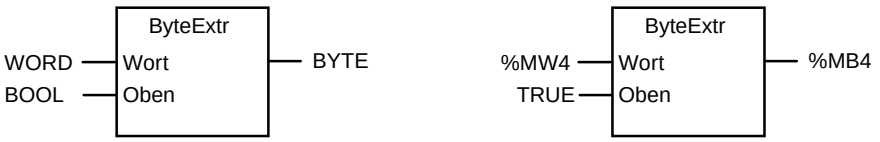
Die IEC 61131-3 fordert an Stellen wie dieser immer eine strenge „Typverträglichkeit“. Dies konsequent zu überprüfen, muß Eigenschaft des Programmiersystems sein. Daher wird in Bsp. C.1 eine Standard-Funktion zur Typumwandlung aufgerufen (WORD_TO_BYTE).

Bsp. C.2 zeigt den Anweisungsteil von ByteExtr in der Programmiersprache ST.

```
FUNCTION  ByteExtr : BYTE     (* Extraktion Byte aus Wort *)
VAR_INPUT ... END_VAR         (* wie oben *)
IF Oben THEN
  ByteExtr := WORD_TO_BYTE (SHR (Wort, 8) );
ELSE
  ByteExtr := WORD_TO_BYTE (Wort);
END_IF;
END_FUNCTION
```

Bsp. C.2. Anweisungsteil von Bsp. C.1 in ST

FUNCTION



...

END_FUNCTION

Bsp. C.3. Grafischer Deklarationsteil der Funktionsdeklaration zu Bsp. C.1 (links) mit einem Aufruf-Beispiel (rechts)

Bsp. C.3 zeigt den Deklarationsteil und einen Beispiel-Aufruf der Funktion ByteExtr in grafischer Darstellung. Der Aufruf ersetzt das untere Byte des Merkerworts 4 durch sein oberes Byte.

C.2 Beispiel für FUNCTION_BLOCK

Der Funktionsbaustein DivMRest berechnet das Divisionsergebnis zweier ganzer Zahlen und liefert sowohl das Divisionsergebnis als auch den Divisionsrest zurück. In einem Ausgangsflag wird „Division durch Null“ angezeigt.

```
FUNCTION_BLOCK DivMRest     (* Division mit Rest *)
                            (* Beginn Deklarationsteil *)
VAR_INPUT                   (* Eingangsparameter *)
  Divident  :  INT;         (* zu teilende ganze Zahl *)
  Divisor   :  INT;         (* ganzzahliger Teiler *)
END_VAR
VAR_OUTPUT RETAIN           (* batteriegepufferte Ausgangsparameter *)
  Quotient  :  INT;         (* Ergebnis der Division *)
  DivRest   :  INT;         (* Divisionsrest *)
  DivFehler :  BOOL;        (* Flag für Division durch Null *)
END_VAR
                            (* Beginn Anweisungsteil *)
LD      0                   (* Null laden *)
EQ      Divisor             (* Ist Divisor Null? *)
JMPC    Fehler              (* Fehlerfall abfangen *)
LD      Divident            (* Dividenten laden, Divisor ungleich Null *)
DIV     Divisor             (* Division durchführen *)
ST      Quotient            (* ganzzahliges Divisionsergebnis speichern *)
MUL     Divisor             (* Divisionsergebnis mit Divisor multiplizieren *)
ST      DivRest             (* Zwischenergebnis merken *)
LD      Divident            (* Dividenten laden *)
SUB     DivRest             (* Zwischenergebnis davon abziehen *)
ST      DivRest             (* ergibt den ganzzahligen „Rest“ der Division *)
LD      FALSE               (* logisch „0“ für Fehlerflag: rücksetzen *)
ST      DivFehler           (* Fehlerflag rücksetzen *)
JMP     Ende                (* fertig, Sprung auf FB-Ende *)
Fehler:                     (* Behandlung des Fehlers „Division durch Null“ *)
LD      0                   (* Null, da Ausgänge im Fehlerfall ungültig sind *)
ST      Quotient            (* Resultat rücksetzen *)
ST      DivRest             (* Rest rücksetzen *)
LD      TRUE                (* logisch „1“ für Fehlerflag: setzen *)
ST      DivFehler           (* Fehlerflag setzen *)
Ende:
RET
                            (* FB-Ende *)
END_FUNCTION_BLOCK
```

Bsp. C.4. Beispiel für die Deklaration eines Funktionsbausteins in AWL

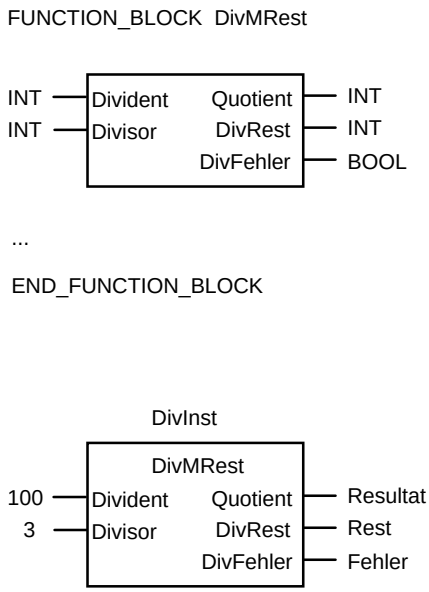
Der FB DivMRest in Bsp. C.4 berechnet die ganzzahlige Division mit Rest der beiden Eingangsparameter Divident und Divisor. Bei Division durch Null wird der Fehler-Ausgang DivFehler gesetzt und die beiden anderen Ausgänge werden definiert auf Null gesetzt, da sie ungültig sind. Die Ausgänge sind batteriegepuffert, d.h. sie bleiben innerhalb der FB-Instanz erhalten, von der aus DivMRest aufgerufen wurde.

Dieses Beispiel kann so nicht als Funktion formuliert werden, da drei Ausgangsvariablen vorkommen.

Bsp. C.5 zeigt den Anweisungsteil von DivMRest in der Programmiersprache ST.

```
FUNCTION_BLOCK DivMRest       (* Division mit Rest *)
VAR_INPUT ... END_VAR         (* wie oben *)
VAR_OUTPUT RETAIN ... END_VAR
IF Divisor = 0 THEN
  Quotient := 0;
  DivRest := 0;
  DivFehler := TRUE;
ELSE
  Quotient := Divident / Divisor;
  DivRest := Divident - (Quotient * Divisor);
  DivFehler := FALSE;
END_IF;
END_FUNCTION_BLOCK
```

Bsp. C.5. Anweisungsteil von Bsp. C.4 in ST



Bsp. C.6. Grafischer Deklarationsteil zu Bsp. C.4 (oben) und ein Aufruf-Beispiel der FB-Instanz DivInst (unten)

Bsp. C.6 zeigt den Deklarationsteil und einen Beispiel-Aufruf der Funktion DivMRest in grafischer Darstellung. Der Aufruf setzt die Instanziierung dieses FB voraus (DivInst).

Die Ausgangsvariablen dieser Instanz besitzen nach Ausführung des FB die Werte: DivInst.Quotient = 33, DivInst.DivRest = 1 und DivInst.DivFehler = FALSE. Diese Rückgabewerte werden jeweils den Variablen Resultat, Rest und Fehler zugewiesen.

C.3 Beispiel für PROGRAM

Das Programm HptProg in Bsp. C.7 ist kein abgeschlossenes Programmierbeispiel, sondern zeigt Möglichkeiten zur Realisierung von Aufgaben und Benutzung von Variablen durch den POE-Typ PROGRAM.

HptProg startet zunächst eine Echtzeituhr DatUhr, die Datum und Uhrzeit mitführt (Std.-FB RTC). Anhand dieser Zeit wird festgestellt, wie lange eine Unterbrechung dieses Programm gedauert hat (ZeitDiff). Voraussetzung dafür ist, daß das SPS-System die Unterbrechung feststellen kann (Ress_Laeuft) und eine Hardware-Uhr von der E/A-Peripherie abfragen kann (AktDatum).

```
PROGRAM     HptProg    (* Beispiel für ein Hauptprogramm *)
                                          (* Beginn Deklarationsteil *)
VAR_INPUT                                 (* Eingangsvariablen *)
  T_Start        :  BOOL := FALSE;        (* Eingang Startbedingung *)
END_VAR
VAR_OUTPUT                                (* Ausgangsvariablen *)
  T_Stoerung     :  BOOL := FALSE;        (* Ausgang „Störung“ *)
END_VAR
VAR_GLOBAL RETAIN                         (* globaler gepufferter Datenbereich *)
  Ress_Laeuft  AT %MX255.5 : BOOL;        (* Laeuft-Flag der Ressource/SPS-CPU *)
  DatUhr         :  RTC;                  (* Programm-Uhr: Datum mit Uhrzeit *)
  AktDatum AT %MD2 : DT;                  (* Hardware-Uhr: Akt. Datum mit Uhrzeit *)
END_VAR
VAR_GLOBAL                                (* globaler Datenbereich *)
  NotAus  AT %IX255.0 : BOOL;             (* Kontakt NOT-AUS *)
  ProgRun        :  BOOL := FALSE;        (* „läuft“-Merker *)
  Fehler         :  BOOL;                 (* Fehler-Merker *)
  F_Code         :  UDINT := 0;           (* Fehler-Code, 32 Bit ohne Vorzeichen *)
END_VAR
VAR                                       (* lokale Variablen *)
  AT %IX250.2 :  BOOL;                    (* direkt dargestellte Variable *)
  FehlerBeh      :  ZFehBeh;              (* FB-Instanz *)
  Flanke         :  R_TRIG;               (* Flankenerkennung *)
  ZeitDiff       :  TIME := t#0s;         (* Zeit-Differenz *)
END_VAR
                                          (* Beginn Anweisungsteil *)
LD      FALSE
ST      Fehler                            (* Fehler rücksetzen *)

...                       (* ermitteln, wie lange ein Stromausfall bzw. eine Unterbrechung
*)                        (* der CPU gedauert hat; Uhr „DatUhr“ ist batteriegepuffert *)
LD      t#0s                              (* Null Sekunden *)
ST      ZeitDiff                          (* rücksetzen *)
LD      DatUhr.Q                          (* Zeit gültig = Uhr läuft *)
JMPC    Weiter                            (* gültig - nichts zu tun *)
LD      AktDatum                          (* aktuelle Zeit abfragen *)
SUB     DatUhr.CDT                        (* letzte Zeit vor Stromausfall *)
ST      ZeitDiff                          (* Zeitdauer Stromausfall *)
Weiter:
...
LD      Ress_Laeuft                       (* CPU laeuft *)
ST      DatUhr.IN                         (* Uhr laeuft, falls CPU laeuft *)
LD      AktDatum                          (* Anfangswert Datum/Uhrzeit *)
ST      DatUhr.PDT                        (* Anfangswert Uhr wird geladen, *)
                                          (* falls steigende Flanke an IN *)
CAL     DatUhr                            (* Starte Echtzeit-Uhr)
...
```

Bsp. C.7. Beispiel für die Deklaration eines Hauptprogramms in AWL. Der FB ZFehBeh („Zentrale Fehler-Behandlung“) muß bereits vorhanden sein (wird fortgesetzt).

```
...
LD      ZeitDiff                    (* Unterbrechungsdauer *)
LT      t#50m                       (* weniger als 50 Sekunden? *)
JMPC    MachtNichts                 (* Anlage noch warm genug... *)
NOT
S       Fehler                      (* Fehler setzen *)
LD      16#000300F2
ST      F_Code                      (* Fehlerursache merken *)
MachtNichts:
LD      NotAus                      (* Not-Aus gedrückt? *)
ST      Flanke.CLK                  (* Eingang Flankenerkennung *)
CAL     Flanke                      (* Vergleich Eingang mit Flankenmerker *)
LDN     Flanke.Q                    (* Flanke an NotAus erkannt? *)
AND     T_Start                     (* UND Start-Flag gesetzt *)
ANDN    Fehler                      (* UND kein neuer Fehler *)
AND     FehlerBeh.Quitt             (* Fehlerursache behoben *)
ST      ProgRun                     (* Globale „läuft“-Bedingung *)

....                    (* ...eigentliche Anweisungen, Aufrufe von FUN/FBs... *)

LD      Fehler                      (* Fehler aufgetreten? *)
AND     %IX250.2
R       ProgRun                     (* Globale Startbedingung rücksetzen *)
LD      ProgRun
JMPNC   Ende                        (* bei Fehler: starte Fehlerbehandlung *)
CALC    FehlerBeh (Code := F_Code)  (* FB mit zentraler Fehlerbehandlung *)
LD      FehlerBeh.Quitt             (* Flag: Fehler wurde quittiert *)
Ende:
LD      ProgRun                     (* Programm laeuft nicht *)
ST      T_Stoerung                  (* Ausgangsparameter setzen *)
RET                                 (* Rücksprung bzw. Ende *)
                                    (* Programm-Ende *)
END_PROGRAM
```

Bsp. C.7. (Fortsetzung)

In diesem Programm wird weiterhin eine Flankenerkennung verwendet, um festzustellen, ob die NotAus-Taste betätigt wurde.

Im globalen Datenbereich wird die Variable ProgRun deklariert, die auch allen unterhalb von HptProg aufgerufenen FBs zur Verfügung steht (dort als VAR_EXTERNAL). Mit ihr wird die Startbedingung verknüpft unter der Bedingung, daß kein NotAus gegeben wurde.

Die FB-Instanz FehlerBeh kann die Bearbeitung eines aufgetretenen Fehlers mit Fehlercode F_Code übernehmen. Wenn der Fehler behoben und quittiert wurde, wird der entsprechende Ausgang Quitt auf TRUE gesetzt.

```
RESOURCE ZentralEinh_1 ON CPU_001
  TASK Periodisch (INTERVAL := time#13ms, PRIORITY := 1);
  PROGRAM Applik WITH Periodisch : HptProg (  T_Start := %I250.0,
                                              T_Stoerung => %Q0.5);
END_RESOURCE
```

Bsp. C.8. Ressource-Definition mit Laufzeitprogramm Applik zu Bsp. C.7

Das Programm HptProg wird mit den Eigenschaften der periodischen Task Periodisch versehen und dadurch zum Laufzeitprogramm Applik der Ressource (SPS-CPU) ZentralEinh_1. Dieses Laufzeitprogramm läuft mit höchster Priorität (1) als zyklische Task mit einer maximalen Zykluszeit von 13 ms ab.

HptProg wird mit dem Wert des Peripheriebits %I250.0 für die Eingangsvariable T_Start aufgerufen und setzt mit T_Stoerung das Ausgangsbit %Q0.5.

D Standard-Datentypen

Dieser Anhang faßt sämtliche Elementaren Datentypen zusammen und charakterisiert tabellenartig ihre Eigenschaften. Ihre Verwendung wird in Kap. 3 erläutert.

Die IEC 61131-3 definiert fünf Gruppen von Elementaren Datentypen, zu denen in Klammern der Allgemeine Datentyp angegeben wird (siehe Tab. 3.9):

- Bitfolge (ANY_BIT),
- Ganzzahl mit und ohne Vorzeichen (ANY_INT),
- Gleitpunkt (ANY_REAL),
- Datum, Zeitpunkt (ANY_DATE),
- Zeichenfolge, Zeitdauer, Abgeleitete (ANY).

Zu jedem dieser Datentyp-Gruppen wird im folgenden eine Tabelle angegeben, die seine Eigenschaften charakterisiert:

- Name (Schlüsselwort),
- Stichworte (Kurzbeschreibung),
- Anzahl der Bits (Datenbreite),
- Werte-Bereich (unter Verwendung der IEC-Literale),
- Anfangswerte „initial“ (Standardwerte).

Die Datenbreite der Datentypen in Tab. D.5 und Tab. D.6 ist ebenso wie der zulässige Wertebereich implementierungsabhängig.

Datentyp	Stichworte	Bits	Bereich	initial
BOOL	boolesch	1	[0,1]	0
BYTE	Bitfolge 8	8	[0,...,16#FF]	0
WORD	Bitfolge 16	16	[0,...,16#FFFF]	0
DWORD	Bitfolge 32	32	[0,...,16#FFFF FFFF]	0
LWORD	Bitfolge 64	64	[0,...,16#FFFF FFFF FFFF FFFF]	0

Tab. D.1. Datentypen „Binär und Bitfolge“

Datentyp	Stichworte	Bits	Bereich	initial
SINT	kurze Ganzzahl	8	[-128,...,+127]	0
INT	Ganzzahl	16	[-32768,...,+32767]	0
DINT	Doppelte Ganzzahl	32	$[-2^{31},...,+2^{31}-1]$	0
LINT	Lange Ganzzahl	64	$[-2^{63},...,+2^{63}-1]$	0

Tab. D.2. Datentypen „Ganzzahl mit Vorzeichen“

Datentyp	Stichworte	Bits	Bereich	initial
USINT	kurze Ganzzahl	8	[0,...,+255]	0
UINT	Ganzzahl	16	[0,...,+65535]	0
UDINT	Doppelte Ganzzahl	32	$[0,...,+2^{32}-1]$	0
ULINT	Lange Ganzzahl	64	$[0,...,+2^{64}-1]$	0

Tab. D.3. Datentypen „Ganzzahl ohne Vorzeichen“

Datentyp	Stichworte	Bits	Bereich	initial
REAL	Gleitpunktzahl	32	s. IEC 559	0.0
LREAL	lange Gleitpunktzahl	64	s. IEC 559	0.0

Tab. D.4. Datentypen „Gleitpunktzahl“

Datentyp	Stichworte	initial
DATE	Datum	d#0001-01-01
TOD	Uhrzeit	tod#00:00:00
DT	Datum mit Uhrzeit	dt#0001-01-01-00:00:00

Tab. D.5. Datentypen „Datum und Zeiten“

Anstelle der Bezeichnung TOD kann gleichwertig auch das Schlüsselwort TIME_OF_DAY, anstelle von DT auch DATE_AND_TIME benutzt werden.

Datentyp	Stichworte	initial
TIME	Zeitdauer	t#0s
STRING	Zeichenfolge	''

Tab. D.6. Datentypen „Zeitdauer und Zeichenfolge“. Der Anfangswert für STRING ist die „leere“ Zeichenfolge.

E Fehlerursachen

Die IEC 61131-3 fordert vom Hersteller eine Liste, in der er seine Fehlerreaktion zu den nachfolgenden Fehlersituationen beschreibt; siehe dazu auch Abschn. 7.10. Die Meldung erfolgt auf vier verschiedene Arten:

1) keinerlei Systemreaktion (%),
2) als Warnung während der Programmerstellung (PeW),
3) als Fehlermeldung während der Programmerstellung (PeF),
4) Fehlermeldung und Fehlerreaktion während der Laufzeit (LzF).

Nr.	Fehlerursache	Zeitpunkt
1	Wert einer Variablen übersteigt den festgelegten Bereich.	LzF
2	Länge der Initialisierungsliste paßt nicht zur Anzahl der Feldeinträge.	PeF
3	Typ-Verletzung bei Konvertierung.	PeF
4	Numerisches Ergebnis einer Standard-Funktion überschreitet den Wertebereich eines Datentyps; Division durch Null einer Standard-Funktion.	LzF PeF, LzF
5	Unterschiedliche Eingangsdatentypen bei einer Auswahlfunktion (Std.-FUN); Selektor (K) ist außerhalb des Bereichs der MUX-Funktion.	PeF LzF
6	Ungültige Angabe einer Zeichenposition; Ergebnis überschreitet die maximale Zeichenkettenlänge (INSERT/CONCAT Ergebnis ist zu lang).	PeW, LzF PeW, LzF
7	Ergebnis überschreitet den Bereich des Datentyps „Zeit“.	LzF

Tab. E.1. Fehlerursachen; der Hersteller liefert eine Tabelle mit, in der die Systemreaktion auf oben beschriebene Fehlerfälle vermerkt ist. Der angegebene *Zeitpunkt* ist die gemäß IEC 61131-3 geeignete Stelle (wird fortgesetzt).

8	Die als Eingangsparameter übergebene FB-Instanz besitzt keine Parameterwerte.	PeF
9	Ein VAR_IN_OUT Parameter besitzt keinen Wert.	PeF
10	Kein oder mehr als ein Anfangsschritt in einem AS-Netzwerk; Anwenderprogramm versucht den Schritt-Merker oder -Zeit zu ändern.	PeF PeF
11	Auswahl einer AS Auswahlverzweigung besitzt mehrere erfüllte Transitionen, ohne daß eine Priorisierung vorhanden ist.	PeW, PeF
12	Seiteneffekte bei der Auswertung von Transitionsbedingungen.	PeF
13	Fehler innerhalb des Aktionskontrollblocks.	PeW, LzF
14	Unsicheres oder unerreichbares AS Netzwerk vorhanden.	PeF
15	Typ-Konflikt bei VAR-ACCESS Variablen.	PeF
16	Anforderungen einer Task bzgl. Prozessor-Ressourcen zu hoch; Ausführungsende wurde nicht erreicht; Allgemeine Scheduling Probleme bei der Taskverwaltung.	PeF PeF PeW, LzF
17	Das numerische Ergebnis einer Operation überschreitet den Datentypbereich (AWL).	PeW, LzF
18	Das Aktuelle Ergebnis und der Operandentyp stimmen nicht überein (AWL).	PeF
19	Division durch NULL (ST); Ungültiger Datentyp einer Operation (ST).	PeW,PeF,LzF PeF
20	Rücksprung einer Funktion ohne Funktionswert (ST).	PeF
21	Iterationsschleife terminiert nicht (ST).	PeW,PeF,LzF
22	Doppelter Bezeichner für Marke und Elementname (KOP / FBS).	PeF
23	Nicht initialisierte Rückkopplungsvariable	PeF

Tab. E.1. (Fortsetzung)

Diese Tabelle ist als Anhaltspunkt zu sehen. Es gibt weitere Fehlermöglichkeiten, die in der IEC-Tabelle nicht aufgeführt sind. Aus diesem Grund sollte sie von jedem Hersteller in geeigneter Form erweitert werden.

F Implementierungsabhängige Parameter

Tab. F.1 gibt die implementierungsabhängigen Parameter wieder, die durch die IEC 61131-3 benannt werden; siehe auch Abschn. 7.11.

Nr.	Implementierungsabhängige Parameter
1	Fehler-Behandlungsmechanismen, die vom Hersteller unterstützt werden.
2	Angabe über Gültigkeit folgender Normzeichen bzw. deren Ersatz: £ statt #, falls # durch nationales Zeichen belegt, Passendes Währungszeichen statt $, falls $ durch nationales Zeichen belegt, ! statt |, falls | durch nationales Zeichen belegt.
3	Maximale Zeichenanzahl von Bezeichnern.
4	Maximale Länge eines Kommentars (ohne führende/schließende Klammerzeichen).
5	Wertebereich von Zeitkonstanten (Angaben über Dauer). Z.B. 0<=Std<1000; 0<=Min<5000,...
6	Wertebereich von Variablen des Datentyps TIME (z.B. 0<=TIME_VAR<1000 Tage); Genauigkeit der Sekundenangaben bei Datentypen TIME_OF_DAY und DATE_AND_TIME.
7	Maximale - Anzahl von Elementen in einem Feld (wieviele Datenelemente), - Feldgröße (wieviele Bytes kann ein Array max. besitzen), - Anzahl von Elementen in einer Struktur, - Strukturgröße (wieviele Bytes kann eine Struktur max. besitzen), - Anzahl von Variablen pro Deklaration, die (durch Komma getrennt) mit einem Datentyp deklariert werden können.
8	Maximale Anzahl von Datenelementen eines Datentyps ENUM.

Tab. F.1. Implementierungsabhängige Parameter, die jeder Hersteller eines IEC 61131-3 Systems beschreiben muß. Die Gliederung in einzelne Gruppen bezieht sich auf einzelne Abschnitte in der Norm. Werden keine konkreten Zahlen gefordert, kann der Hersteller eine informelle Beschreibung beifügen (wird fortgesetzt).

9	Maximale Standardlänge von String-Variablen; Maximale Länge von String-Variablen.
10	Anzahl der Hierarchie-Ebenen bei direkten Variablen (%QX1.1.1.2...); Abbildung der Namen (%IX....) auf die reale Hardware.
11	Maximale Anzahl von Indizes zur Bestimmung eines Feldelements; Maximaler Wertebereich, den ein Index annehmen kann; Maximale Anzahl von Ebenen bei Strukturen (Tiefe der Untergliederung).
12	Initalisierungswert der Eingänge (%IX...) beim Systemstart.
13	Information zur Bestimmung der Ausführungszeiten von POEs (es sind keine näheren Angaben in der IEC 61131-3 vermerkt).
14	Funktionsdarstellung: textuell oder grafisch; Art der Funktionsbenennung (Symbol wie + oder Name wie ADD, ...).
15	Max. Anzahl möglicher Anwenderfunktionen (soweit Begrenzung vorhanden).
16	Max. Anzahl von Eingängen bei erweiterbaren Funktionen.
17	Genauigkeit der einzelnen Typ-Umwandlungsfunktionen (REAL_TO_INT,...).
18	Genauigkeit der numerischen Transformationsfunktionen (LOG, SIN, ...); Implementierte arithmetische (überladene) Funktionen.
19	Maximale Anzahl von Anwender-Funktionsbausteinen und möglichen Instanzen (soweit Begrenzung vorhanden).
20	Wertebereich des Parameters PV (Endewert bei Standard-Zeitbausteinen).
21	Reaktion des Systems auf eine Änderung des PT-Eingangs (Endewert bei Standard-Zeitbausteinen) während einer laufenden Zeitoperation .
22	Anzahl und Längenangaben der in der IEC 61131-5 spezifizierten SEND-Eingängen und RCV-Ausgängen.
23	Maximale Länge von Programmen (ausführbarer Code).
24	Funktionale Beschreibung der AS- Implementierung (Umfang/Besonderheiten; zeitliche Bedingungen;...); informelle Beschreibung.
25	Zeitauflösung der Variablen Schritt-Zeit; Maximale Anzahl von Schritten pro AS- Netzwerk und pro POE.
26	Maximale Anzahl von Transitionen pro AS- Netzwerk und pro POE.
27	Funktionsweise der Aktions-Steuerung (ACTION-CONTROL Baustein soweit vorhanden).
28	Maximale Anzahl von Aktionsblöcken / Schritt.
29	Art der Anzeige von aktiven Schritten (z.B. durch inverse Darstellung, zusätzliche Zeichen, ...); die Art der Darstellung ist normfrei; Minimale Transitionsschaltzeit (verursacht durch Berechnungszeit der SPS), Maximale Anzahl von Vorgänger und Nachfolger Schritten (bei Simultan- / Alternativ Verzweigungen).
30	Beschreibung der Ressource-Bibliothek. Jede Verarbeitungseinheit (z.B. Prozessortyp) erhält eine Beschreibung aller Funktionen (Std-FBs, Std.-Funktionen, Datentypen, ...), die von dieser Ressource verarbeitet werden können.

Tab. F.1. (Fortsetzung)

31	Maximale Anzahl von Tasks / Ressource;

	Task Intervall Auflösung (Zeitbereich möglicher Aufrufe für periodische Tasks); Art der Task-Steuerung (mit / ohne Pre-emptive Scheduling).
32	Maximallänge von ST Ausdrücken (Anzahl von Operanden, Operatoren); Vorgehen beim Auswerten von Teilausdrücken von booleschen Variablen (zur Vermeidung von unerwünschten Seiteneffekten);
33	Maximallängen von ST Anweisungen (IF; CASE; FOR; ...), Einschränkungen.
34	Maximale Anzahl von möglichen CASE Alternativen.
35	Wert der Laufvariablen von FOR-Schleifen nach Verlassen der Schleife.
36	Art (semi-/ oder -grafisch) der Darstellung bei grafischen Programmiersprachen; evtl. Einschränkungen bzgl. der Netzwerktopologie bei KOP /FBS.
37	Auswertungsreihenfolge von Rückkopplungsschleifen.
38	Beschreibung der Ausführungsreihenfolge von Netzwerken.

Tab. F.1. (Fortsetzung)

Darüber hinaus gibt es eine Vielzahl weiterer Parameter, die bei der Implementierung eines Programmiersystems nach IEC 61131-3 zu beachten sind.

Aus dieser Menge stellt die Tab. F.2 eine subjektive Auswahl zusammen.

Nr.	**Implementierungsabhängige Parameter**
1	Umfang der syntaktischen und semantischen Überprüfung, die von den textuellen und grafischen Editoren angeboten wird (bereits während der Eingabe).
2	Freie Plazierung von grafischen Elementen bei Editoren mit Linien-Aktualisierung („Gummiband“) oder automatische Plazierung nach syntaktischen Regeln.
3	Deklaration und Verwendung Direkt dargestellter Variablen (DdV): - Die Deklaration erfolgt immer mit symbolischem Namen; im Anweisungsteil darf nur das Symbol *oder* beide (Symbol; direkte Adresse) verwendet werden oder - DdV können auch ohne symbolische Namen deklariert werden (Verwendung nur der direkten Adresse) oder - DdV werden vom Programmiersystem implizit deklariert.
4	Reihenfolge der VAR_*...VAR_END Blöcke, sowie Mehrfachverwendung gleichnamiger Blöcke.
5	Realisierte Funktions-Aufrufmöglichkeiten (mit / ohne Formal-Parameter) in ST.
6	Umfang der Realisierung der EN/ENO Parameter in KOP und FBS, sowie mögliche Auswirkungen auf AWL/ST-Quellen.

Tab. F.2. Weitere implementierungsabhängige Parameter (nicht Teil der IEC 61131-3) (wird fortgesetzt)

7	Möglichkeit der Übergabe von benutzerdefinierten Strukturen (TYPE) als Funktions- und FB-Parameter (derzeit nicht in der Norm festgelegt).
8	Bekanntmachung von Anwender definierten Datentypen (TYPE...END_TYPE): global- und POE-weit (derzeit nicht in der Norm festgelegt).
9	Umfang der Datenbereichsüberprüfung bei der Programm-Erstellung und während der Laufzeit.
10	Grafische Darstellung der Bestimmungszeichen/Attribute bei der Variablen-deklaration.
11	Mehrfachinstanziierung der Realzeit Uhr RTC oder Verwendung *einer* Instanz für alle Aufrufe.
12	Einschränkung in der Verwendung komplexer Datentypen (Funktionstyp auch für Typ „String“ oder anwenderdefinierte Typen zugelassen, ...?).
13	Algorithmus zur Auswertung von KOP/FBS-Netzwerken (siehe Abschn. 4.3.4 und 7.4.1).
14	Hilfen zur Nachvollziehbarkeit der Querübersetzung (soweit vorhanden).
	...

Tab. F.2. (Fortsetzung)

Die IEC 61131-3 erlaubt Funktionalitäten, die über den Standard hinausgehen. Sie müssen allerdings beschrieben sein. Einige davon wurden für Tab. F.3 zusammengestellt.

Nr.	**Erweiterungen**
1	Ausdehnung der Wertebereichsüberprüfung auf mehr Datentypen (z.B. ANY_NUM) als nur Integer (ANY_INT).
2	FB-Instanzen als Feld-Variable zulassen.
3	FB-Instanzen in Strukturen erlauben.
4	Überladen auch für benutzerdefinierte Funktionen, FBs, und Programme.
5	Zeitpunkt der Berechnung der Schrittweite bei FOR-Anweisungen.
6	Möglichkeit der Verwendung von Präprozessor-Anweisungen für Literale, Makros, bedingte Übersetzung, Include- Anweisungen (zum Einlesen von Dateien mit FB-Schnittstelleninformation, mit der Liste der verwendeten direkt darstellbaren Variablen oder EXTERNAL- Deklarationen, ...).
7	Zusätzliche Deklarationsmöglichkeit in FBs zum Anlegen dynamischer (nicht statischer) Variablen (z.B. VAR_DYN...END_VAR), um den Speicherplatz in der SPS zu reduzieren.
8	Verwendung unterschiedlicher Speichermodelle in der SPS (Small; Compact; ...).
	...

Tab. F.3. Mögliche Erweiterungen (nicht Teil der IEC 61131-3)

G Beispiel einer AWL-Syntax

Viele der in diesem Buch aufgeführten Beispiele sind in Anweisungsliste (AWL) formuliert. Diese Programmiersprache ist weit verbreitet und wird von den meisten Programmiersystemen unterstützt. Durch die Möglichkeit, Datentypen verwenden zu können, die meist nur in Hochsprachen zu finden waren wie Felder oder Strukturen, ergeben sich mit der IEC 61131-3 gegenüber herkömmlichem AWL neue Möglichkeiten.

Die nachfolgend beschriebene AWL-Syntax wird mit Hilfe von Syntaxgraphen in vereinfachter Form wiedergegeben.

Syntaxgraphen erläutern die zulässige Verwendung von Begrenzungszeichen, Schlüsselworten, Literalen und Bezeichnern auf anschauliche Weise. Sie können zur Entwicklung von Compilern leicht in eine textuelle Form gebracht werden und definieren so den formalen Aufbau einer Programmiersprache (*Syntax* einer Sprache). Der Anwender kann sich ihrer bedienen, um nachzuschlagen, wie die von ihm zu programmierenden Sprachkonstrukte aufzubauen sind.

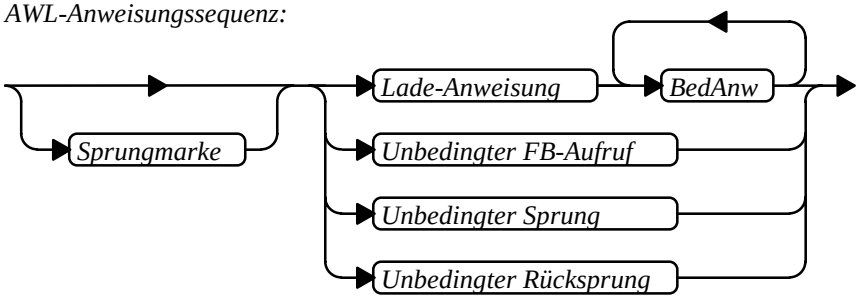
Die Syntaxbeschreibung dieses Kapitels geht über die IEC 61131-3 hinaus, da zusätzlich zu der reinen Syntaxbeschreibung semantische Bedingungen (Folgerichtiger Gebrauch des Aktuellen Ergebnisses, Verwendung von Funktionsparametern,...) mit eingearbeitet sind; die IEC 61131-3 bietet hierfür lediglich eine informelle Beschreibung.

Die Regeln sind im Abschn. G.1 zu finden. Die Verwendung dieser Graphen wird im Abschn. G.2 anhand eines Beispiels erklärt.

G.1 Syntaxgraphen für AWL

Gibt es für einen Knoten des Syntaxgraphen eine weitere Untergliederung (Untergraphen), ist der Name dieses Knoten kursiv geschrieben. Schlüsselwörter oder terminale Symbole, die nicht weiter ersetzt werden, besitzen Normal-Schrift.

AWL-Anweisungssequenz:



BedAnw:

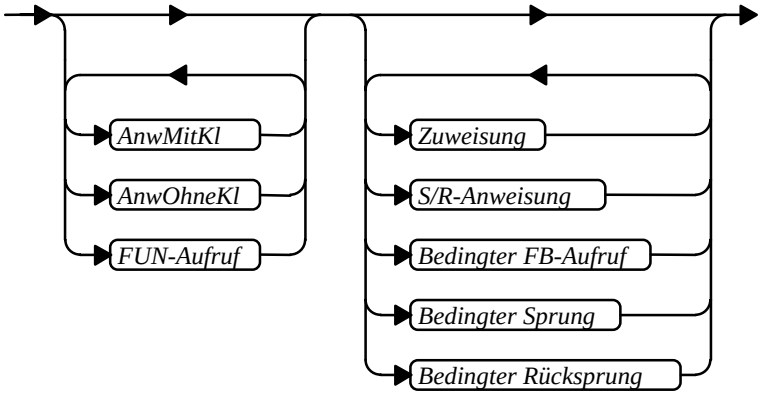


Abb. G.1. Syntaxgraphen einer AWL-Anweisungs-Sequenz mit den Unterelementen „Bedingte Anweisung“ und „Sprungmarke“ (wird fortgesetzt)

Sprungmarke:

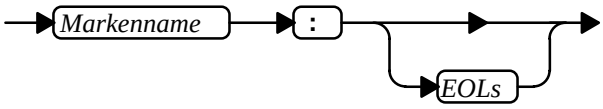


Abb. G.1. (Fortsetzung)

Eine AWL-Anweisungs-Sequenz beginnt optional mit einer Sprungmarke; es schließt sich entweder eine Lade-Anweisung, gefolgt von einer oder mehreren bedingten Anweisungen BedAnw, eine unbedingte Anweisung mit FB-Aufruf, Sprung oder Rücksprung an; siehe dazu den Syntaxgraphen der Abb. G.1.

Die Bedingte Anweisung beginnt mit einer Folge von Anweisungen mit und ohne Klammern bzw. Funktionsaufrufen. Daran schließt sich eine Folge von (S/R-) Zuweisungen, bedingten Aufrufen oder Sprüngen an.

Die Sprungmarke besteht aus einer Marken-Bezeichnung, gefolgt von einem Doppelpunkt. Sie steht entweder unmittelbar vor der ersten Anweisung der Folge oder wird von EOLs (engl.: end-of-line) gefolgt. Letzteres ist eine Erweiterung der IEC 61131-3, wird aber von einigen Programmiersystemen für eine bessere optische Blockstrukturierung von Anweisungssequenzen zugelassen.

EOLs stellen das Zeilenende dar; einem EOL kann *ein* Kommentarfeld vorangestellt werden.

Es folgen die Syntaxgraphen für Anweisungen, Aufrufe und Sprünge.

AnwOhneKl:

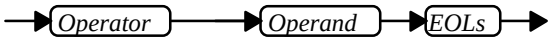


Abb. G.2. Syntaxgraph einer Anweisung ohne Klammer

Eine Anweisung ohne Klammer (Abb. G.2) besteht aus einem AWL-Operator mit einem Operanden und dem Zeilenabschluß durch EOLs.

AnwMitKl:

AnwInKl:

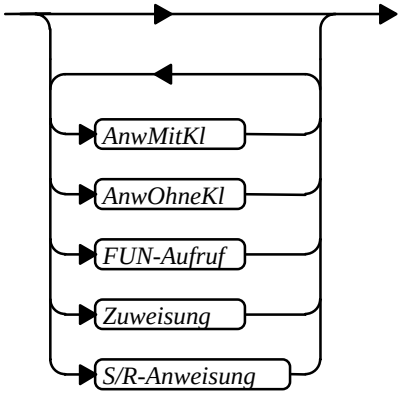


Abb. G.3. Syntaxgraphen einer Anweisung mit Klammer

Die in Abb. G.3 gezeigten Syntaxgraphen zeigen den Aufbau einer Anweisung mit Klammer. Diese Anweisungsart beginnt mit einem die Klammerung einleitenden Operator mit Operanden als erste Anweisung. Darauf folgen beliebig viele Anweisungen AnwInKl innerhalb der Klammer, die mit einer schließenden Klammer und EOLs abgeschlossen wird.

Diese inneren Anweisungen können wiederum Klammern beinhalten (Schachtelung) sowie FUN-Aufrufe und Zuweisungen besitzen.

FUN-Aufruf:

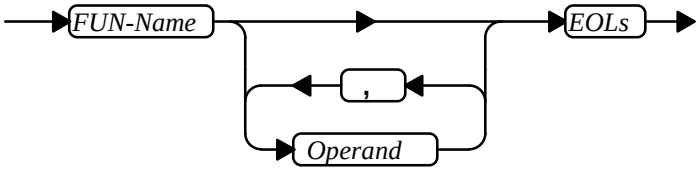
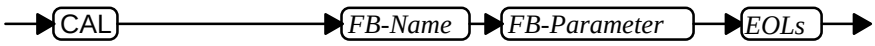


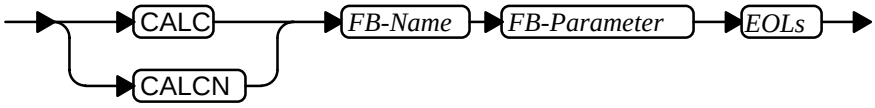
Abb. G.4. Syntaxgraph für den Aufruf einer Funktion

Wie Abb. G.4 zeigt, besteht der Aufruf einer Funktion aus der Angabe des Funktionsnamens sowie einer Anzahl durch Kommata getrennter Operanden als Aktualparameter dieser Funktion.

Unbedingter FB-Aufruf:



Bedingter FB-Aufruf:



FB-Parameter:

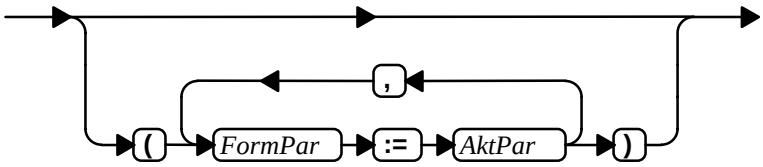


Abb. G.5. Syntaxgraphen für den Aufruf eines Funktionsbausteins

Die Syntaxgraphen für bedingten und unbedingten Aufruf eines Funktionsbausteins zeigt Abb. G.5. Der Aufruf wird beim unbedingten Aufruf mit CAL, beim bedingten mit CALC bzw. CALCN begonnen. Darauf folgen der Name der FB-Instanz sowie in Klammern die Parameter des FB.

Die Übergabe eines Aktualparameters an einen Formalparameter wird durch eine Zuweisung mit „:=“ dargestellt. Solche Zuweisungen sind für jeden Parameter erforderlich und werden durch Kommata getrennt.

Unbedingter Sprung:

Bedingter Sprung:

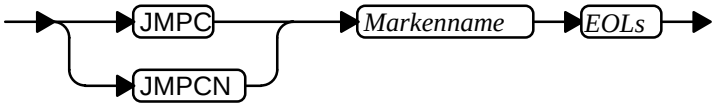
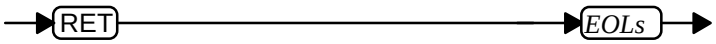


Abb. G.6. Syntaxgraphen für Bedingten und Unbedingten Sprung

Bei Sprüngen wird hinter dem Sprung-Operator JMP (unbedingt) bzw. JMPC oder JMPCN (bedingt) der Name der Sprungmarke angegeben (Abb. G.6).

Unbedingter Rücksprung:



Bedingter Rücksprung:

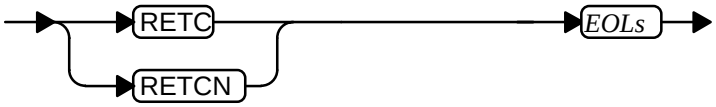


Abb. G.7. Syntaxgraphen für Bedingten und Unbedingten Rücksprung

Die in Abb. G.7 gezeigten Rücksprünge besitzen keine Operanden bzw. Parameter, können aber wie jeder AWL-Anweisung kommentiert werden.

Lade-Anweisung:

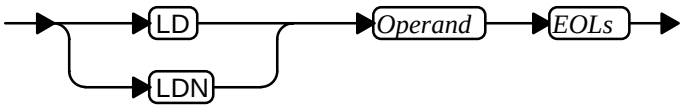


Abb. G.8. Syntaxgraph für die Lade-Anweisung

Die Lade-Anweisung der Abb. G.8 besitzt einen einzelnen (invertierbaren) Operanden. Sie kann nicht mit einer Klammer kombiniert werden oder innerhalb einer Klammerung verwendet werden.

Zuweisung:

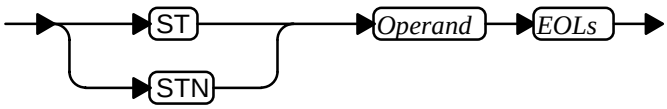


Abb. G.9. Zuweisung

Zuweisungen (Abb. G.9) bestehen aus dem Operator ST bzw. STN und der Angabe des zu speichernden Operanden.

S/R-Anweisung:

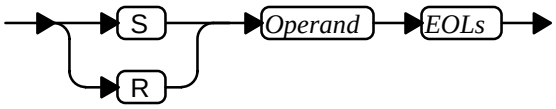


Abb. G.10. S/R-Anweisung

Eine S/R-Anweisung (Abb. G.10) besteht aus den AWL-Operatoren S oder R und einem Operanden.

Operator:

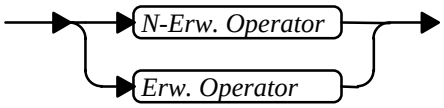


Abb. G.11. Operator als Zusammenfassung der Erweiterbaren und Nicht-Erweiterbaren Operatoren

Die in Abb. G.11 dargestellten Operatoren besitzen eine verknüpfende Funktion, dienen also nicht zum Laden oder Speichern. Es wird dabei zwischen Erweiterbaren und Nicht-Erweiterbaren Operatoren unterschieden, die im folgenden dargestellt werden.

Erw. Operator:

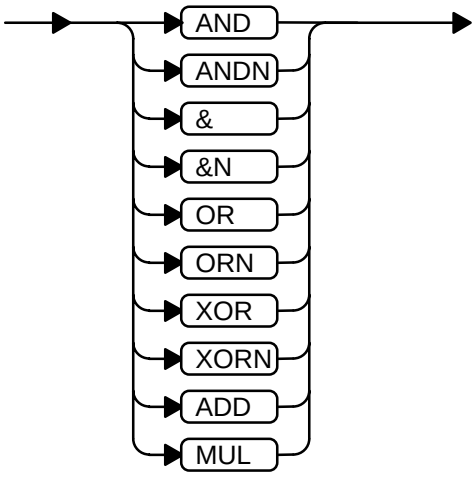


Abb. G.12. Erweiterbare Operatoren: Bitweise Operationen sowie Addition und Multiplikation

Die Erweiterbaren Operatoren zeigt Abb. G.12. Sie können mehr als zwei Eingangsparameter besitzen. Die bitweise booleschen Operatoren (Standard-Funktionen) können optional mit Invertierung verwendet werden.

N-Erw. Operator:

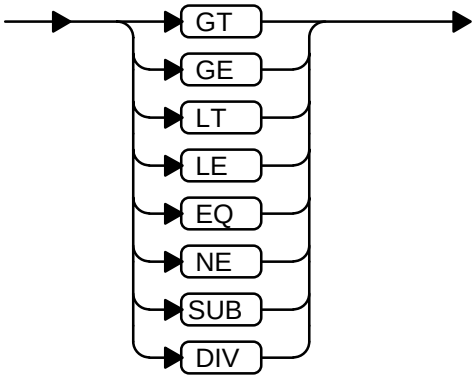


Abb. G.13. Nicht-Erweiterbare Operatoren: Vergleichen, Subtraktion, Division

Die Nicht-Erweiterbaren Operatoren in Abb. G.13 haben (einschließlich AE) genau zwei Eingangsparameter.

EOLs:

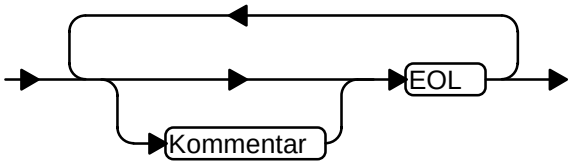
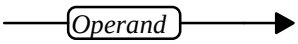


Abb. G.14. Syntaxgraph für das Zeilenende (engl.: end of line) einer AWL-Anweisung mit Kommentar

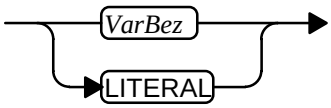
Eine AWL-Zeile wird mit einem einzelnen EOL-Zeichen (z.B. Carriage Return / Line Feed) oder einem Kommentar, der von EOL gefolgt wird, abgeschlossen (Abb. G.14). Diese Elemente können einmal oder beliebig oft hintereinander vorkommen.

Ein Kommentar beginnt mit „(*“,endet mit „*)“ und enthält dazwischen beliebig viele alphanumerische Zeichen **ohne** EOL. Kommentare können nicht geschachtelt werden.

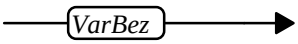
AktPar:



Operand:



Markenname:
FUN-Name:
FB-Name:
FormPar:



VarBez:

Abb. G.15. Operanden, Parameter und Bezeichner werden durch Identifier bzw. Literale gebildet.

Abb. G.15 zeigt die Abbildung von Parametern, Operand und verschiedenen Bezeichnertypen auf Bezeichner (IDENTIFIER) und Werte (LITERAL).

Zur Vereinfachung werden hier die Syntaxgraphen der Bezeichner und Literale nicht dargestellt. Ihre prinzipielle Darstellung ist in Abschn. 3.2 erläutert.

G.2 AWL-Beispiel zu Syntaxgraphen

Zu den soeben dargestellten AWL-Syntaxgraphen wird nun ein Beispiel gegeben. Dieses zeigt, wie aus Syntaxgraphen Programmbeispiele konstruiert werden und umgekehrt, wie nämlich AWL-Beispiele auf ihre Richtigkeit hin überprüft werden können.

```
                                    Beginn der AWL-Sequenz
0001        SequenzEins:            (* Sprungmarke *)
0002                                (* Einfache Verknüpfung mit Sprung *)
0003        LD      Var1            (* Lade-Anweisung *)
0004                                (* Beginn bedingte Anweisungen *)
0005        ANDN    Var2            (* Anweisung ohne Klammer *)
0006        ORN     ( Var3          (* Anweisung mit Klammer *)
0007        AND     Var4
0008        )                       (* Ende der Klammerung *)
0009        AND     Var5
0010        ST      Var6            (* Zuweisung *)
0011        S       Var7            (* S/R-Anweisung *)
0012        RETC                    (* Bedingter Rücksprung *)
0013                                (* Ende der bedingten Anweisungen *)
0014                                (* Ende der AWL-Sequenz *)
```

Bsp. G.1. AWL-Beispiel. Die Kommentare weisen auf den entsprechenden Syntaxgraphen hin. Die links stehenden Zeilennummern dienen der Referenzierung in Tab. G.1.

Um zu sehen, wie sich das AWL-Beispiel in Bsp. G.1 aus den Syntaxgraphen des Abschn. G.1 aufbauen läßt, werden in Tab. G.1 zu jeder AWL-Zeile die anzuwendenen Syntaxgraphen angegeben.

Zeile in Bsp. G.1	Syntaxgraph	Abbildung
0001-0014	AWL-Anweisungs-Sequenz	Abb. G.1
0001-0002	Sprungmarke	Abb. G.1
0003-0004	Lade-Anweisung	Abb. G.8
0005	Anweisung ohne Klammer	Abb. G.2
0005	Erweit. Operator ANDN	Abb. G.12
0006-0008	Anweisung mit Klammer	Abb. G.3
0006,0007	Erweit. Operatoren ORN, AND	Abb. G.12
0007,0009	Anweisung ohne Klammer	Abb. G.2
0007,0009	Erweit. Operator AND	Abb. G.12
0010	Zuweisung	Abb. G.9
0011	S/R-Anweisung	Abb. G.10
0012-0014	Bedingter Rücksprung	Abb. G.7

Tab. G.1. Anzuwendende Syntaxgraphen zu Zeilen der AWL-Sequenz in Bsp. G.1

Dieses Beispiel zeigt, wie sich durch mehrfache Anwendung der Syntaxgraphen ein konkretes AWL-Programm ergibt: Im Syntaxgraphen für eine AWL-Anweisungs-Sequenz (Abb. G.1) wird zunächst die Sprungmarke mit Bezeichner, Doppelpunkt und Kommentaren eingesetzt, gefolgt von der ersten (Lade-) Anweisung.

Der Teil „bedingte Anweisungen“ ergibt sich aus zwei Anweisungen, von denen die erste mit Klammer wiederum einige Anweisungen in Klammern enthält. Nach den bedingten Anweisungen wird die Sequenz mit Zuweisungen und Rücksprung abgeschlossen.

Auf diese Weise ist es möglich, aus den einzelnen Syntaxgraphen zulässige AWL-Sequenzen zu erzeugen. Ebenso können umgekehrt zu AWL-Zeilen die erforderlichen Syntaxgraphen gefunden werden, um festzustellen, ob ein Programmstück syntaktisch korrekt ist.

H Reservierte Schlüsselworte und Begrenzungszeichen

Es wird von der IEC 61131-3 ausdrücklich zugelassen, daß Schlüsselworte und Begrenzungszeichen durch Übersetzungstabellen an nationale Zeichensätze angepaßt werden können.

H.1 Reservierte Schlüsselworte

In Tab. H.1 sind sämtliche Reservierten Schlüsselworte für Programmiersprachen der IEC 61131-3 in alphabetischer Reihenfolge aufgelistet. Sie dürfen *nicht* als Namen für benutzter-definierte Elemente verwendet werden.

A	ABS	ACOS
	ACTION	ADD
	AND	ANDN
	ANY	ANY_BIT
	ANY_DATE	ANY_INT
	ANY_NUM	ANY_REAL
	ARRAY	ASIN
	AT	ATAN
B	BOOL	BY
	BYTE	

Tab. H.1. Reservierte Schlüsselworte der IEC 61131-3 (wird fortgesetzt)

C	CAL	CALC
	CALCN	CASE
	CD	CDT
	CLK	CONCAT
	CONFIGURATION	CONSTANT
	COS	CTD
	CTU	CTUD
	CU	CV
D	D	DATE
	DATE_AND_TIME	DELETE
	DINT	DIV
	DO	DS
	DT	DWORD
E	ELSE	ELSIF
	END_ACTION	END_CASE
	END_CONFIGURATION	END_FOR
	END_FUNCTION	END_FUNCTION_BLOCK
	END_IF	END_PROGRAM
	END_REPEAT	END_RESOURCE
	END_STEP	END_STRUCT
	END_TRANSITION	END_TYPE
	END_VAR	END_WHILE
	EN	ENO
	EQ	ET
	EXIT	EXP
	EXPT	
F	FALSE	F_EDGE
	F_TRIG	FIND
	FOR	FROM
	FUNCTION	FUNCTION_BLOCK
G	GE	GT
I	IF	IN
	INITIAL_STEP	INSERT
	INT	INTERVAL

Tab. H.1. (Fortsetzung)

J	JMP	JMPC
	JMPCN	
L	L	LD
	LDN	LE
	LEFT	LEN
	LIMIT	LINT
	LN	LOG
	LREAL	LT
	LWORD	
M	MAX	MID
	MIN	MOD
	MOVE	MUL
	MUX	
N	N	NE
	NEG	NOT
O	OF	ON
	OR	ORN
P	P	PRIORITY
	PROGRAM	PT
	PV	
Q	Q	Q1
	QU	QD
R	R	R1
	R_TRIG	READ_ONLY
	READ_WRITE	REAL
	RELEASE	REPEAT
	REPLACE	RESOURCE
	RET	RETAIN
	RETC	RETCN
	RETURN	RIGHT
	ROL	ROR
	RS	RTC
	R_EDGE	

Tab. H.1. (Fortsetzung)

S	S	S1
	SD	SEL
	SEMA	SHL
	SHR	SIN
	SINGLE	SINT
	SL	SQRT
	SR	ST
	STEP	STN
	STRING	STRUCT
	SUB	
T	TAN	TASK
	THEN	TIME
	TIME_OF_DAY	TO
	TOD	TOF
	TON	TP
	TRANSITION	TRUE
	TYPE	
U	UDINT	UINT
	ULINT	UNTIL
	USINT	VAR
V	VAR_ACCESS	VAR_EXTERNAL
	VAR_GLOBAL	VAR_INPUT
	VAR_IN_OUT	VAR_OUTPUT
W	WHILE	WITH
	WORD	
X	XOR	XORN

Tab. H.1. (Fortsetzung)

H.2 Begrenzungszeichen

Begrenzungszeichen stellen „Sonderzeichen“ in der Syntax der Programmiersprachen dar und besitzen je nach Verwendung unterschiedliche Bedeutung. So können runde Klammern beispielsweise zur Kennzeichnung von Anfang und Ende einer Aktualparameterliste bei einem Funktionsaufruf oder zusammen mit dem Stern zur Begrenzung von Kommentaren benutzt werden.

Sämtliche Begrenzungszeichen und deren Kombinationen werden in Tab. H.2 mit ihren möglichen Bedeutungen aufgeführt.

Zeichen zur grafischen Darstellung der Linienführung werden hier nicht berücksichtigt.

Begrenzungszeichen	**Bedeutung, Erläuterungen**
Leerzeichen	können überall eingefügt werden - außer innerhalb Schlüsselworten, Literalen, Bezeichnern, Direkt dargestellten Variablen oder Kombinationen von Begrenzungszeichen (wie „(*“ oder „*)“). Über Tabulatoren (TABs) macht die IEC 61131-3 keine Aussage, sie werden üblicherweise wie Leerzeichen behandelt.
Zeilenende	zum Abschluß einer Anweisungszeile in AWL, in ST auch innerhalb Anweisungen zulässig. Nicht zulässig innerhalb AWL-Kommentaren. (engl. EOL end of line), üblicherweise durch CR&LF (engl.: carriage return & Line feed) implementiert.
Kommentaranfang **(***	Beginn eines Kommentars (nicht schachtelbar)
Kommentarende ***)**	Ende eines Kommentars
Plus +	1. Führendes Vorzeichen eines Dezimal-Literals, auch im Exponenten eines Gleitpunkt-Literals 2. Additionsoperator in Ausdrücken
Minus -	1. Führendes Vorzeichen eines Dezimal-Literals, auch im Exponenten eines Gleitpunkt-Literals 2. Subtraktionsoperator in Ausdrücken 3. Negationsoperator in Ausdrücken 4. Jahr-Monat-Tag-Trennzeichen in Zeitliteralen
Nummernkreuz #	1. Basiszahl-Trennzeichen in Literalen 2. Zeitliteral-Trennzeichen

Tab. H.2. Begrenzungszeichen der IEC 61131-3 (wird fortgesetzt)

Begrenzungszeichen	Bedeutung, Erläuterungen
Punkt **.**	1. Ganzzahl/Bruch-Trennzeichen 2. Trennzeichen innerhalb Hierarchischer Adressen von Direkt dargestellten und Symbolischen Variablen 3. Trennzeichen zwischen Komponenten einer Datenstruktur (beim Zugriff) 4. Trennzeichen für Komponenten einer FB-Instanz (beim Zugriff)
e, E	führendes Zeichen für Exponenten der Gleitpunkt-Literale
Anführungszeichen **'**	Anfang und Ende von Zeichenfolgen
Dollar-Zeichen **$**	Anfang von Sonderzeichen innerhalb Zeichenfolgen
Präfix Zeitliterale **t#, T#** **d#, D#** **d, D** **h, H** **m, M** **s, S** **ms, MS** **date#, DATE#** **time#, TIME#** **time_of_day#** **TIME_OF_DAY#** **tod#, TOD#** **date_and_time#** **DATE_AND_TIME#** **dt#, DT#**	Einleitende Zeichen von Zeitliteralen, Kombinationen von Klein- und Großbuchstaben sind ebenfalls zulässig.
Doppelpunkt **:**	Trennzeichen für: 1. Uhrzeit innerhalb Zeitliterale 2. Festlegung des Datentyps bei Variablendeklaration 3. Festlegung eines Datentyp-Namens 4. Schrittnamen 5. PROGRAM...WITH... 6. Funktionsname/Datentyp 7. Zugriffspfad: Name/Typ 8. Sprungmarke vor nächster Anweisung 9. Netzwerkname (Marke) vor nächster Anweisung
Zuweisung **:=**	1. Operator für Anfangswert-Zuweisung 2. Eingangsverbindungs-Operator (Zuweisung von Aktualparameter an Formalparameter beim POE-Aufruf) 3. Anweisungsoperator

Tab. H.2. (Fortsetzung)

Begrenzungszeichen	Bedeutung, Erläuterungen

Runde Klammerung	**(...)**	Anfang und Ende für: 1. Aufzählungsliste 2. Anfangswert-Liste, auch: Mehrfache Anfangswerte (mit Wiederholungszahl) 3. Bereichsangabe 4. Operator in AWL (Berechnungsebene) 5. Parameterliste beim POE-Aufruf 6. Unterausdruck-Hierarchie
Eckige Klammern	**[...]**	Anfang und Ende für: 1. Feldindex (Zugriff auf ein Feldelement) 2. Zeichenfolgenlänge (bei Deklaration)
Komma	**,**	Trennzeichen für: 1. Aufzählungsliste 2. Anfangswert-Liste 3. Feldindex (mehrdimensional) 4. Variablennamen (bei Mehrfachdeklaration desselben Datentyps) 5. Parameterliste beim POE-Aufruf 6. Operandenliste in AWL 7. CASE-Werteliste
Semikolon	**;**	Abschluß von: 1. Definition eines (Daten)-Typs 2. Deklaration (z.B. Variablen) 3. ST-Anweisung
Punkt-Punkt	**..**	Trennzeichen für: 1. Bereichsangabe 2. CASE-Bereich
Prozent	**%**	Einleitendes Zeichen für Hierarchische Adressen bei Direkt dargestellten und Symbolischen Variablen
Zuweisung2	**=>**	Ausgangsverbindungs-Operator (Zuweisung von Formalparameter an Aktualparameter beim PROGRAM-Aufruf)
Vergleich	**>, < >=, <=, =, <>,**	Vergleichsoperatoren in Ausdrücken
Exponent	******	als Operator in Ausdrücken
Multiplikation	*****	Multiplikations-Operator in Ausdrücken
Division	**/**	Divisions-Operator in Ausdrücken
Kaufmanns-UND	**&**	AND-Operator in Ausdrücken

Tab. H.2. (Fortsetzung)

I Geplante Änderungen am Standard

In die vorliegende Auflage wurde das aktuell vorliegende Corrigendum vollständig eingearbeitet. Wie eingangs erwähnt, handelt es sich dabei um ein Dokument mit Fehlerkorrekturen und Klarstellungen zum Standard.

Die Amendments dagegen beschreiben Erweiterungen und umfangreiche Änderungen des Standards. Da sie noch nicht endgültig verabschiedet sind (Stand 1999), sei im folgenden nur auf die wichtigsten Funktionen hingewiesen:

- Einführung von *typisierten Literale* für Bool- und numerische Konstanten. Ein Konstante kann über ein Prefix <Elementarer Datentyp>#<Datenwert> gekennzeichnet werden. Beispiel: SINT#20 oder BOOL#0
- Verwendung des *ISO 10646 Zeichencodes* (1 / 2 Byte Länge) für Zeichenketten. Damit sind auch nationale Buchstaben wie Umlaute ä, ü, ö,… darstellbar.
- Einführung der Variablenart *VAR_TEMP … END_VAR*. In diesem Block deklarierte Variablen werden bei jedem FB-Aufruf neu initialisiert und behalten nicht ihren Wert zwischen Aufrufen (entspricht der VAR-Deklaration in Funktionen).
- Variablenblöcke erhalten die Attribute RETAIN oder *NON_RETAIN*. Ist keines der Attribute angegeben, entscheiden die "Implementierungsabhängigen Parameter", ob diese attributlosen Variablenblöcke gepuffert werden oder nicht.
- Physikalische Adresse können während der Programmerstellungsphase durch Verwendung des Zeichens "*" teilspezifiziert werden, Beispiel: *%Q**. Die endgültige Festlegung erfolgt in der Konfigurationsphase.
- Bei der Deklaration von Funktionsblockinstanzen können die Initialwerte von Aufrufparameter und lokalen Variablen dieser Instanz festgelegt werden. Damit wird erreicht, daß verschiedene Instanzen desselben Bausteintyps unterschiedliche Initialwerte besitzen können.
- Die Übergabe von Aktual- an Formalparameter erfolgt wie bisher mit ":=". Die Zuweisung von Ausgangsparameterwerten erfolgt mit "=>".
  Beispiel:
  CAL fb_instanz(eingabe_parameter:= 2, rueckgabe_wert => aufruf_variable).
- Einführung eines allgemeinen Datentyps *ANY_MAGNITUDE* als Obermenge von ANY_NUM und TIME. *ANY_STRING* beschreibt die beiden Mengen

STRING (1 Byte Zeichenlänge) und *WSTRING* (2 Byte Zeichenlänge). ANY untergliedert sich in *ANY_DERIVED* für alle abgeleiteten Datentypen und *ANY_ELEMENTARY*.

J Glossar

Im folgenden werden wichtige Begriffe und Abkürzungen erläutert, die in diesem Buch verwendet werden. Sie werden in alphabetischer Reihenfolge aufgeführt.

Begriffe, die durch die IEC 61131-3 festgelegt sind, sind entsprechend mit „IEC “ gekennzeichnet.

Abgeleiteter Datentyp	IEC	Mit Hilfe einer *Typdefinition* wird ein benutzerspezifischer *Datentyp* erzeugt, dessen Elemente aus *Elementaren* und/oder wiederum *Abgeleiteten Datentypen* bestehen.
Ablaufsprache	IEC	Ablaufsprache AS (engl.: Sequential Function Chart – SFC) ist eine Programmiersprache zur Beschreibung sequentieller und paralleler Steuerungsabläufe mit Zeit- und Ereignis-Steuerung.
AE		Abk. für *Aktuelles Ergebnis*
Aktion	IEC	Boolesche Variable oder eine Reihe von Anweisungen, die über einen *Aktionsblock* angesteuert werden können, in *AS*.
Aktionsblock	IEC	Aktivierungsbeschreibung von *Aktionen* in AS
Aktionskontrollblock	IEC	Steuereinheit für jede *Aktion* in AS, die über *Aktionsblöcke* die Eingangsbedingung für die zugeordnete Aktionsfreischaltung erhält.
Aktualparameter		Versorgung einer Eingangsvariablen (*Formalparameter*) einer *POE* mit einem aktuellen Wert.
Aktuelles Ergebnis	IEC	Zwischenergebnis in *AWL* von beliebigem *Datentyp*
Allgemeiner Datentyp	IEC	Zusammenfassung von *Elementaren Datentypen* zu Gruppen, um *Überladen von Funktionen* beschreiben zu können; wird auch „Generischer Datentyp“ genannt.
Anfangswert		Wert einer *Variable*, der bei ihrer Initialisierung vergeben wird.

Anweisungsliste	IEC	Anweisungsliste (engl.: Instruction List – IL) ist eine weit verbreitete Assembler-ähnliche Programmiersprache für SPS-Systeme.
AS	IEC	Abk. für *Ablaufsprache*
Aufzählung		Besonderer *Datentyp* zur Definition von ganzzahligen Werten.
AWL	IEC	Abk. für *Anweisungsliste*
Batteriepufferung		Fähigkeit einer SPS, bestimmte Datenbereiche bei Stromausfall gegen Verlust zu sichern. Die IEC 61131-3 verwendet dazu das Schlüsselwort RETAIN.
Baustein		(Unter-) Programmeinheit, aus denen SPS-Programme zusammengesetzt werden. Bausteine können oft unabhängig voneinander in die SPS geladen werden. vgl. *POE*.
Bereichsangabe		Angabe eines zulässigen Wertebereichs für einen *Datentyp* oder eine *Variable*.
CIM		Computer Integrated Manufacturing
CPU		Central Processing Unit (Zentraleinheit, z.B. einer SPS)
Datenbaustein		Global zur Verfügung stehender, aktivierbarer Datenbereich. vgl. *Baustein*. In der IEC 61131-3 gibt es dafür keine direkte Entsprechung, sie werden hier durch globale, strukturierte Datenbereiche bzw. *FB-Instanz*-Datenbereiche ersetzt.
Datentyp		definiert die Bitlänge und Eigenschaften des Wertebereichs einer *Variablen.*
Deklaration	IEC	Bekanntgabe von *Variablen* und *FB-Instanzen* in einem *Deklarationsblock* unter Angabe eines Bezeichners, des *Datentyps* bzw. *FB-Typs* sowie ggf. *Anfangswerte*, Bereichsangabe und Feldeigenschaften. Die Definition bzw. die Programmierung von *POEs* wird ebenfalls als *Deklaration* bezeichnet, da hierbei ihre Schnittstellen dem *Programmiersystem* bekannt gemacht werden.
Deklarationsblock	IEC	Zusammenfassung von *Deklarationen* einer Variablenart zu Beginn der *POE*.
Direkt dargestellte Variable	IEC	*Variable* ohne weiteren Bezeichner, die einer *Hierarchischen Adresse* entspricht.
E/A-Peripherie		Die zu einem *SPS-System* gehörenden Eingangs- und Ausgangs-Module mit ihren *Hierarchischen Adressen.*
Einzelelement-Variable	IEC	*Variable*, die auf einem einzelnen *Datentyp* basiert.

Elementarer Datentyp	IEC	Ein durch die IEC 61131-3 vordefinierter Standard-*Datentyp*.
Erweiterung von Funktionen	IEC	Eine *Funktion* kann eine variable Anzahl von Eingängen besitzen.
FB	IEC	Abk. für *Funktionsbaustein*
FB-Instanz	IEC	siehe *Instanz*
FB-Typ	IEC	Name eines *Funktionsbausteins* mit Aufruf- und Rückgabeschnittstelle
FBS	IEC	Abk. für *Funktionsbausteinsprache*
Feld		Aneinanderreihung von Elementen gleichen *Datentyps.*
Flanke		Mit einer „steigenden" Flanke wird der 0→1-Übergang einer booleschen Variable bezeichnet. Eine „fallende" Flanke ist dementsprechend der 1→0-Übergang.
Formalparameter		Name (Bezeichner) einer Eingangsvariablen (alle *POEs*) oder Ausgangsvariablen (*Funktionsbaustein* und *Programm*).
Funktion	IEC	Eine *POE* vom Typ FUNCTION
Funktionsbaustein	IEC	Eine *POE* vom Typ FUNCTION_BLOCK
Funktionsbausteinsprache	IEC	Funktionsbausteinsprache FBS (engl.: Function Block Diagram – FBD) ist eine Programmiersprache zur Beschreibung von Netzwerken mit gleichzeitig arbeitenden booleschen, arithmetischen und ähnlichen Elementen.
Hierarchische Adresse	IEC	Physikalische Steckplatzadressen der E/A-Module eines SPS-Systems (vgl. *E/A-Peripherie*).
Indirekter FB-Aufruf		Aufruf einer *FB-Instanz*, dessen Name als VAR_IN_OUT-Parameter einer *POE* übergeben wurde.
Instanz	IEC	Strukturierter Datensatz eines FBs durch *Deklaration* eines *Funktionsbausteins* unter Angabe des *FB-Typs.*
Kaltstart	IEC	Programmstart, bei dem sämtliche Variablen und Speicherbereiche (neu) initialisiert werden (engl.: Cold Restart). Dieser auch *Neustart* genannte Vorgang kann bei bestimmten Ereignissen automatisch oder auch manuell durch den Anwender erfolgen.
Kommentar		Zwischen Klammern und Sternchen eingeschlossener Text (nicht schachtelbar!) zur Erläuterung des Programms, wird vom *Programmiersystem* nicht interpretiert.
Konfiguration	IEC	Sprachelement CONFIGURATION, das einem *SPS-System* entspricht.
KOP	IEC	Abk. für *Kontaktplan*

Kontaktplan	IEC	Kontaktplan (engl.: Ladder Diagram – LD) ist eine Programmiersprache zur Beschreibung von Netzwerken mit gleichzeitig arbeitenden booleschen, elektromechanischen Elementen wie Kontakten und Spulen.
Laufzeitprogramm		Zu einer ausführbaren Einheit gebundenes Programm vom *POE*-Typ PROGRAM (durch Zuordnung einer *Task*).
Multielement-Variable	IEC	*Variable* vom Typ *Feld* oder *Struktur*, die aus mehreren *Datentypen* zusammengesetzt ist.
Neustart		siehe *Kaltstart*
PC		Personal Computer. Gleichzeitig in der IEC 61131 (englische Version) verwendete Abkürzung für „Programmable Controller“.
POE	IEC	Abk. für *Programm-Organisationseinheit*
Programm	IEC	Eine *POE* vom Typ PROGRAM
Programm-Organisationseinheit	IEC	Ein *Baustein* der IEC 61131-3 des Typs *Funktion*, *Funktionsbaustein* oder *Programm*, aus dem Anwenderprogramme hierarchisch aufgebaut werden.
Programmiergerät		siehe *SPS-Programmiergerät*
Programmiersystem		siehe *SPS-Programmiersystem*
Querübersetzung		Konvertierung der Darstellung einer *POE* zwischen verschiedenen Programmiersprachen, typischerweise zwischen textuellen und grafischen Sprachen.
Rekursion		ist in der IEC 61131-3 unzulässig. Bezeichnet: a) die *Deklaration* eines *FB*s unter Verwendung des eigenen Namens bzw. *FB-Typs*, b) den gegenseitigen Aufruf von *FB*s. Rekursion wird als Fehler betrachtet und muß bei der Programmierung bzw. zur Laufzeit erkannt werden.
Ressource	IEC	Sprachelement RESOURCE, das einer Zentraleinheit des *SPS-Systems* entspricht.
Rückübersetzung		Rückgewinnung der Quelle einer *POE* aus der *SPS*.
Schritt	IEC	Zustandsknoten in einem *AS*-Programm, in dem Anweisungen der zu einem *Schritt* zugehörigen *Aktion* angestoßen werden.
Semantik		Bedeutung der Sprachelemente einer Programmiersprache sowie ihre Auslegung und Anwendung.

SPS		Speicherprogrammierbare Steuerung (engl.: „Programmable Controller“).
SPS-Programmiergerät		Einheit aus Computer, *Programmiersystem* und sonstiger Peripherie zur Programmierung der *SPS*.
SPS-Programmiersystem		Gesamtheit aller Programme, die zur Programmierung eines *SPS-Systems* notwendig sind: Erstellung und Übersetzung des Programms, Übertragen in die *SPS* sowie Programmtest- und Inbetriebnahme-Funktionen.
SPS-System		Gesamtheit aller zur Ausführung eines SPS-Programms benötigten Teile.
ST	IEC	Abk. für *Strukturierter Text*
Standard-Funktionen	IEC	Menge der durch die IEC 61131-3 fest vordefinierten *Funktionen* zur Realisierung SPS-typischer Funktionalität.
Standard-Funktionsbausteine	IEC	Menge der durch die IEC 61131-3 fest vordefinierten *Funktionsbausteine* zur Realisierung SPS-typischer Funktionalität.
Std.-FB		Abk. für *Standard-Funktionsbausteine*
Std.-FUN		Abk. für *Standard-Funktionen*
Strukturierter Text	IEC	Strukturierter Text (engl.: Structured Text) ist eine Programmiersprache zur Beschreibung von Algorithmen und Ausführungssteuerung mit den Mitteln einer modernen Hochsprache.
Symbolische Variable	IEC	*Variable* mit Bezeichner, der eine *Hierarchische Adresse* zugeordnet wird.
Syntax		Aufbau und strukturelles Zusammenwirken der Sprachelemente einer Programmiersprache.
Task	IEC	Definition von Laufzeiteigenschaften eines Programms.
Transition	IEC	Übergang von einem AS-*Schritt* zum nächsten durch Auswertung der Transitionsbedingung.
Typdefinition		Definition eines benutzerspezifischen *Datentyps* auf der Basis bereits vorhandener *Datentypen.*
Überladen von Funktionen	IEC	Eine *Funktion* stellt unter gleichem Namen Funktionsklassen mit unterschiedlichen Eingangs-*Datentypen* (jeweils vom gleichen Typ) zur Verfügung.
Variable		Bezeichnung eines Datenspeichers, der Werte annehmen kann, die durch den *Datentyp* sowie Angaben bei der Variablen-*Deklaration* festgelegt werden.

Warmstart	IEC	Programmstart an der Stelle, an der ein Spannungsausfall stattgefunden hat (engl.: Hot Restart). Dabei bleiben sämtliche gepufferten Datenbereiche des Programms erhalten und das Programm kann weiterlaufen, als hätte es die Unterbrechung nicht gegeben. Im Unterschied zum *Warmstart am Programmanfang* muß die Unterbrechungsdauer prozeßabhängig innerhalb eines vorgegebenen Intervalls liegen. Dazu muß das SPS-System eine separat gesicherte Echtzeit-Uhr besitzen, um die Unterbrechungsdauer berechnen zu können.
Warmstart am Programmanfang	IEC	Programmstart wie beim *Warmstart* mit dem Unterschied, daß am Programmanfang wieder begonnen wird, wenn die Unterbrechungszeit eine maximale Zeitspanne überschritten hat. Das Anwenderprogramm kann anhand eines entsprechenden Status-Flags diese Situation erkennen, um eine spezielle Vorbesetzung seiner Daten vornehmen zu können. (engl.: Warm Restart).
Wiederanlauf		Zusammenfassender Begriff für *Warmstart* bzw. *Warmstart am Programmanfang*.
Zuordnungsliste		Liste, die zentral die Zuordnung von Symbolen zu SPS-Adressen beinhaltet.
Zyklus		Ein Durchlauf des (periodisch aufgerufenen) Anwenderprogramms.
Zykluszeit		Die Zeit, die ein Anwenderprogramm für einen *Zyklus* benötigt.

K Literaturverzeichnis

Bücher zur SPS-Programmierung:

[Adam-97] Adam Hans-Joachim, Adam Mathias
„Programmieren in Anweisungsliste nach IEC 1131-3“
Elektor Verlag,
ISBN 3-89576-056-0

[Berger-87] Hans Berger
„Automatisieren mit S5-115U“
Hrsg.: Siemens-Aktiengesellschaft, Berlin München, 1987
ISBN 3-8009-1484-0

[Berger-96] Hans Berger
„Automatisieren mit STEP 7 in AWL“ Speicher-
programmierbare Steuerungen SIMATIC S7-300/400
Hrsg.: Siemens-Aktiengesellschaft, Berlin München,
Publicis-MCD-Verlag Erlangen 1996,
ISBN 3-89578-036-7

[Grötsch-96] Eberhard E. Grötsch
„SPS, speicherprogrammierbare Steuerungen, Bd.1,
Einführung und Übersicht“,
R. Oldenbourg Verlag, München Wien 1996,
ISBN 3-486-23054-9

[Grötsch-97] Eberhard E. Grötsch, L. Seubert
„SPS 2- Speicherprogrammierbare Steuerungen“,
Programmbeispiele und Produkte
R. Oldenbourg Verlag, München Wien 1997,
ISBN 3-486-23669-5

[Lewis-98] R. W. Lewis
„Programming industrial control systems using IEC 1131-3“, IEE Control Engineering, The Institution of Electrical Engineers, 1998,
ISBN 0-852-96950-3

[Neumann-98] Peter Neumann, Eberhard E. Grötsch, Christoph Lubkoll, René Simon
„SPS - Standard: IEC 1131-3, Programmieren in verteilten Automatisierungssystemen“
R. Oldenbourg Verlag, München Wien 1998,
ISBN 3-486-24153-2

[Rolle-98] I. Rolle (Hrsg.), A. Lehmann, H.-P. Otto, L. Trapp, H. Wegmann
„IEC 61131, Wozu?“,
VDE Taschenbuch, 1998,
ISBN 3-800-72309-3

[Wratil-96] Peter Wratil
„Moderne Programmiertechnik für Automatisierungssysteme“, EN 61131 (IEC 1131) verstehen und anwenden
Vogel Verlag Würzburg 1996,
ISBN 3-8023-1575-8

Normen zur SPS-Programmierung:

[DIN EN 61131-1-94] Deutsche Norm DIN EN 61131 Teil 1
„Speicherprogrammierbare Steuerungen Teil 1: Allgemeine Informationen“
(IEC 1131-1: 1992)
Beuth Verlag GmbH, Berlin 8/94

[DIN EN 61131-2-94] Deutsche Norm DIN EN 61131 Teil 2
„Speicherprogrammierbare Steuerungen Teil 2: Betriebsmittelanforderungen und Prüfungen“
(IEC 1131-2: 1992)
Beuth Verlag GmbH, Berlin 1994
Ergänzungen dazu:
Berichtigung zu DIN EN 61131 Teil 2 (VDE 0411 Teil 500)
DIN EN 61131 Teil 2, Berichtigung 1, Ausgabe: 1998-08
Beuth Verlag GmbH, Berlin 1998

Deutsche Norm DIN EN 61131-2/:1994/A11

„Speicherprogrammierbare Steuerungen Teil 2:
Betriebsmittelanforderungen und Prüfungen“
Beuth Verlag GmbH, Berlin 94
Ausgabe 1996-12
Beuth Verlag GmbH, Berlin 1996

DIN EN 61131-2/A11 Berichtigung 1
Berichtigung zu DIN EN 61131-2/A11 (VDE 0411 Teil 500/A11)
Ausgabe 1998-08
Beuth Verlag GmbH, Berlin 1998

[DIN EN 61131-3-94] Deutsche Norm DIN EN 61131 Teil 3
„Speicherprogrammierbare Steuerungen - Programmier-sprachen“
(IEC 1131-3: 1993)
Beuth Verlag GmbH, Berlin 8/94

[DIN EN 61131-3 B1] Deutsche Norm DIN EN 61131 Teil 3; Beiblatt 1
„Speicherprogrammierbare Steuerungen – Leitlinien für die Anwendung und Implementierung von Programmiersprachen für Speicherprogrammierbare Steuerungen“,
Beuth Verlag GmbH, Berlin 1997-11

[IEC EN 61131-3 B1] Committee Draft - IEC 61131-3, 2nd Ed.
„Programmable controllers - programming languages“,
IEC 65B/WG7/TF3(PT3E2ACD)1
Committee Draft, Houston, 10/1998

[IEC AMEND-98] IEC SC65B/WG7/TF3, Ergänzungsvorschläge zur IEC 1131-3:
„Draft Amendments to IEC 1131-3“,
Draft Version, Venedig, Italien, 8/98

[IEC CORR-97] IEC SC65B/WG7/TF3, Korrekturvorschläge zur IEC 1131-3:
„Revised Technical Corrigendum to IEC 1131-3“,
Draft Version, Yokohama, Japan, 5/97

[IEC TR2-97] IEC SC65B/WG7/TF3, Type 2 Technical Report
„Proposed Extensions to IEC 1131-3“,
Committee Draft, Paris, Frankreich, 9/96
Stand: 05/1997

[IEC 61499-98] IEC TC65/WG6(PT1CD+PT2CD)
„Function blocks for industrial-process measurement and control systems“,
Committee Draft, Parts 1+2, 1998

[VDI 3696/2-95] VDI/VDE-Richtlinien, VDI/VDE 3696 Blatt 2
„Herstellerneutrale Konfigurierung von Prozeßleitsystemen- Standard-Funktionsbausteine“
10/95.
VDI/VDE-Handbuch Regelungstechnik.

[VDI 3696/3-95] VDI/VDE-Richtlinien, VDI/VDE 3696 Blatt 3,
„Herstellerneutrale Konfigurierung von Prozeßleitsystemen- Syntax des Funktionsbaustein-Textes“
10/95.
VDI/VDE-Handbuch Regelungstechnik.

Aufsätze zur IEC 61131-3 in Fachzeitschriften und zu Kongressen

[Brendel-4/99] Wolfgang Brendel
„IEC 1131-3 SoftSPS – PC based control mit Windows CE“
in: SPS-Magazin, 4/99
TeDo-Verlag, Cölbe

[Brendel-6/97] Wolfgang Brendel
„Strukturiert in die Zukunft“
in: industrie-elektrik+elektronik (iee), Heft 6/97
Hüthig-Verlag, Heidelberg

[Brendel-7/96] Wolfgang Brendel
„Software-Recycling - Anlagenprojektierung mit CFC“
in: industrie-elektrik+elektronik (iee), Heft 7/96
Hüthig-Verlag, Heidelberg

[Brendel-7/94] Wolfgang Brendel
„Grundlagen der einheitlichen SPS-Programmierung“

in: industrie-elektrik+elektronik (iee), Heft 7/94
Hüthig-Verlag, Heidelberg

[Fraunhofer-5/99] Fraunhofer IPA
„Defizite&Trends in der Automatisierung“
in: Markt&Technik, Ausgabe Nr. 22 vom 28. Mai 1999
Markt&Technik-Verlag, Haar

[Frost & Sullivan-95] "World programmable logic controller markets: increasing functionality promises continued growth"
Study on the PLC market 1993-2000
Frost & Sullivan

[John-4/92] Karl-Heinz John
„SPS im Wandel - die wachsende Bedeutung der Programmier-Software“, Teile 1 und 2
in: SPS-Magazin, Heft 3/92 und 4/92
TeDo-Verlag, Cölbe

[John-94] Karl-Heinz John
„Durch Normung droht keine SPS-Monokultur“
in: Markt&Technik, Ausgabe Nr. 6 vom 4. Februar 1994
Markt&Technik-Verlag, Haar

[Käsbeck-6/97] Toni Käsbeck
„Programm-Mauern schleifen“
in: industrie-elektrik+elektronik (iee), Heft 6/97
Hüthig-Verlag, Heidelberg

[Kubas-10/98] Jörg Kubas
„IEC 1131-3 lernt dazu - das S5 Programm lebt weiter“
in: SPS-Magazin, Heft 10/98
TeDo-Verlag, Cölbe

[Maier-11/96] Helmut Maier, Bertram Matz
„Kompatibel oder normgerecht?“ Die Simatic S7: Spagat zwischen Simatic S5 und IEC-Norm, 2-teilig
in: Elektronik, Heft 24+25/96
Hüthig-Verlag, Heidelberg

[Oblasser-6/97] Siegfried Oblasser
„Spagat zwischen STEP 5 und IEC 1131 geschafft“
in: industrie-elektrik+elektronik (iee), Heft 6/97
Hüthig-Verlag, Heidelberg

[Otto-9/94] Peter Otto
„Die neue IEC 1131-3 für SPS tritt in Kraft“
in: SPS-Magazin, Heft 4/94
TeDo-Verlag, Cölbe

[OMAC-94] „Requirements of Open, Modular Architecture Controllers for Applications in the Automotive Industry“
Version 1.1
13.12.94; Chrysler; Ford; G M

[PLCopen-96] PLCopen / Michael Babb
„IEC 1131-3: A standard programming resource for PLCs“
in: Control Engineering, Heft 2/96

[PLCopen-99] Zeitschrift „PLCopening“ der PLCopen
Jahrgänge 1992-99
PLCopen, Zaltbommel, Niederlande

[Sperber-93] Michael Sperber
„Einheitliche SPS-Programmierung mit IEC 1131-3“, Teil 3
in: SPS-Magazin, Heft 6/93
TeDo-Verlag, Cölbe

[SPS/IPS/Drives-98] „Die Zukunft der SPS – IEC 61131-3 nur Kompromiß“
Podiumsdiskussion
in: OpenAutomation 1/ 1999, VDE Verlag

[Tiegel-10/93] Michael Tiegelkamp
„Einheitliche SPS-Programmierung mit IEC 1131-3“,
Teile 1 und 2
in: SPS-Magazin, Hefte 4 und 5/93
TeDo-Verlag, Cölbe

[Tiegel-11/93] Michael Tiegelkamp
„SPS-Programmiersysteme nach IEC 1131-3“,
Kongreß zu SPS/IPC/Drives 1993, Sindelfingen

[Tiegel-10/99] Michael Tiegelkamp et al.
„Programmieren nach IEC 61131 in der Praxis“,
in: SPS-Magazin, Hefte 5 – 10 (vierteilig)
TeDo-Verlag, Cölbe

L Index