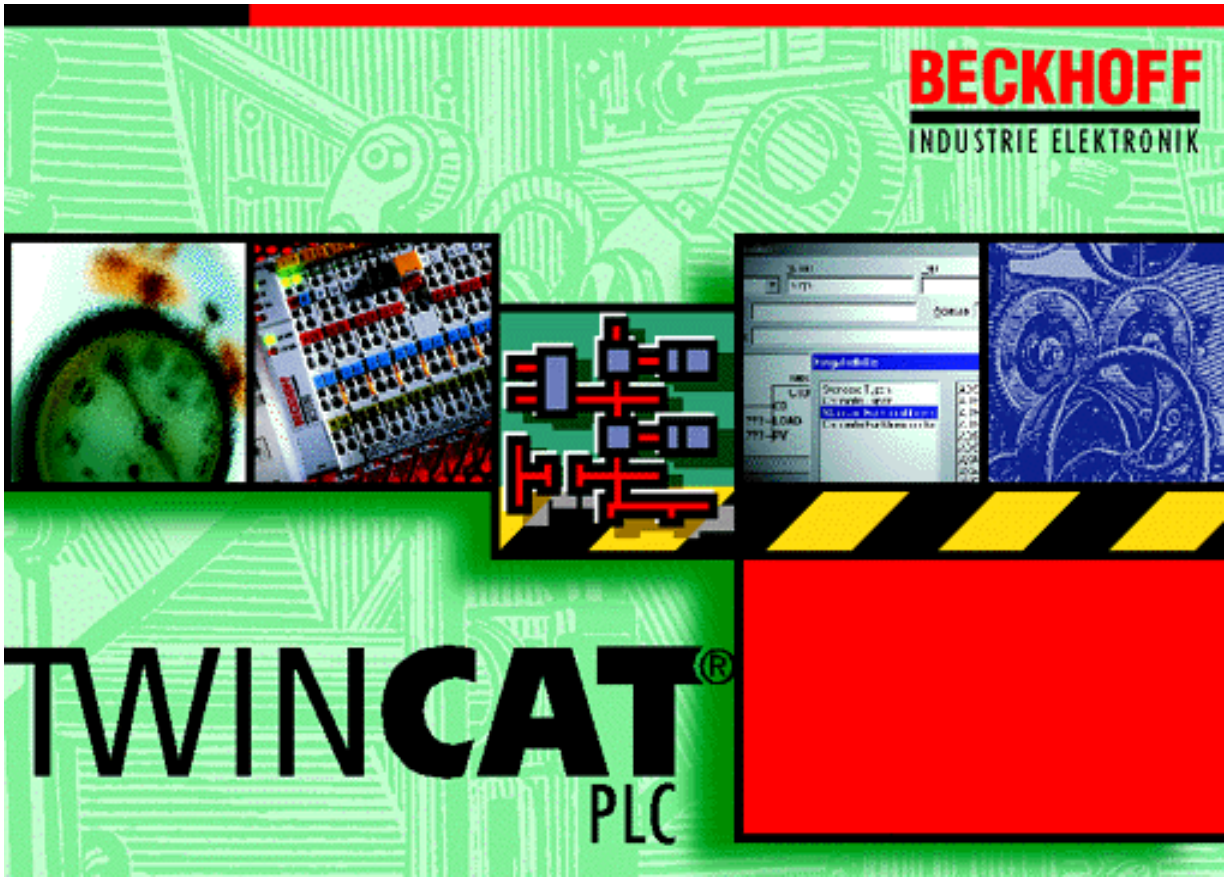


Beckhoff TwinCAT®

Total Windows Control and Automation Technology

Einführung in IEC-1131-3 Programmierung

TwinCAT Version: alle
Letzte Änderung: 16.11.1998

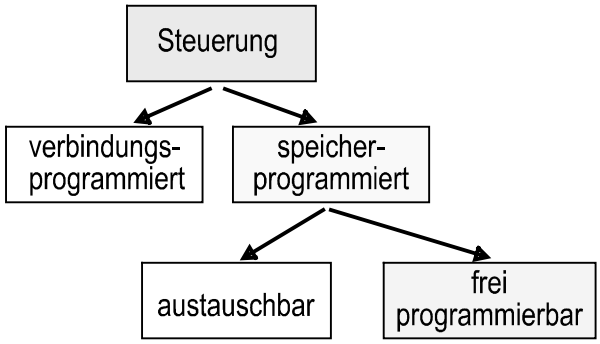
BECKHOFF
INDUSTRIE ELEKTRONIK

Eine Einführung in IEC-1131-3 Programmierung

Speicherprogrammierbare Steuerungen

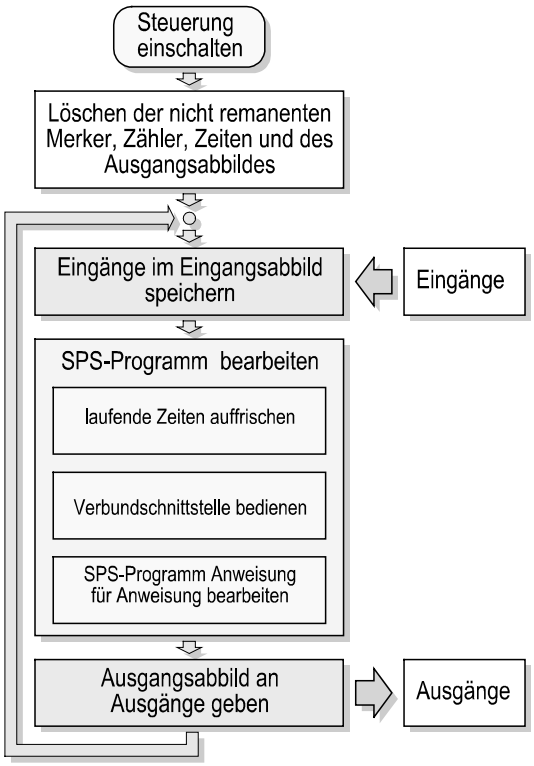
Die Anlagensteuerung über verbindungsorientierte Relais-Technik ist aus der Steuerungstechnik nahezu vollständig verschwunden. Frei programmierbare Steuerungen sind heute Stand der Technik wenn es um die Automatisierung von Anlagen geht. Von kleinen Steuerungsaufgaben mit nur wenigen Eingangs- und Ausgangsgrößen bis hin zu Prozeßrechnern mit vielen tausend Ein- und Ausgängen reicht die Bandbreite der sogenannten Speicherprogrammierbaren Steuerungen (SPS). Entsprechend der Anforderungen sind auch die verfügbaren Steuerungen im Preissegment von einigen hundert DM bis hin zu etlichen zehntausend DM am Markt verfügbar.

Bei den Steuerungen haben sich für Automatisierungsaufgaben frei programmierbare Steuerungen durchgesetzt



Die Funktionsweise und der Aufbau derartiger Systeme folgt, unabhängig von der Größe der SPS, einer einheitlichen Systematik. Eine SPS besteht aus Hardware und Software. Die Hardware besteht aus dem Prozessor mit den entsprechenden Speicherbausteinen und der weiteren Elektronik zum Anschluß von Ein- und Ausgabebaugruppen. Kompakt-SPSen können in der Regel nur einige wenige Eingangs- und Ausgangssignale bearbeiten. Demgegenüber besteht eine modular aufgebaute SPS aus verschiedenen Baugruppen wie Baugruppenträger mit Systembus, Netzteil, Zentraleinheit und Anwendungsspeicher, digitale Ein- und Ausgangs-Baugruppen sowie intelligenten Baugruppen für Analogdatenverarbeitung oder die Steuerung von Antrieben.
Zu den Leistungsmerkmalen gehört auch der direkte Anschluß von Sensorik und Aktorik über einen standardisierten Spannungspegel von 24V. Hierdurch ist eine dem Störungspegel in industrieller Umgebung angepaßte Störfestigkeit sichergestellt.
Auf der Seite der Software muß man zwischen den Bereichen Anwendungssoftware und Betriebssystem-software unterscheiden. Jede Zentraleinheit besitzt ihr Betriebssystem, welches für die Verwaltung der Systemressourcen und der organisatorischen Funktionen verantwortlich ist. Hierzu gehört die Organisation und Verwaltung des freiprogrammierbaren Speichers, der Zähler, Timer und der Koordination der Baugruppen und Einzelkomponenten. Darüber hinaus sorgt das Betriebssystem für einen geregelten Anlauf nach Anlegen der Betriebsspannung, es übernimmt das Fehlermanagement und ermöglicht den Austausch von Informationen über Kommunikationsbaugruppen. Schließlich koordiniert es die Abarbeitung des Anwendungsprogramms, welches den logischen Ablauf der Steuerungsaufgabe abbildet.

Das Besondere an einer SPS ist die zyklische Verarbeitung einer Anweisungsliste. Ziel ist die Bearbeitung des Eingangs- und Ausgangsabbildes innerhalb einer definierten Zeitspanne.



Während Personalcomputer oder auch andere Computer sich nach dem Einschalten mit dem Betriebssystem melden und erst in einer zweiten Stufe die Anwendung gestartet wird, beginnt die CPU einer SPS sofort nach Einschalten des aktiven Betriebs, dem sogenannten RUN-Mode, mit der Bearbeitung des Anwendungsprogramms. Direkt nach dem Einschalten werden alle nicht remanenten Speicher zurückgesetzt und der Bearbeitungszyklus beginnt mit dem Einlesen des Eingangsbildes. Danach erfolgt das Auslesen der Timer und das Auffrischen der Verbundschnittstelle. Schließlich erfolgt die Bearbeitung der gesamten SPS-Anweisungsliste. Anschließend wird das durch die Anweisungsliste erzeugte Ausgangsbild an die Ausgänge gegeben. Der Programmzyklus beginnt von neuem mit dem Einlesen des Eingangsbildes. Die Zykluszeit ist das wohl wichtigste Geschwindigkeitskriterium für eine SPS und liegt typischerweise im praktischen Betrieb im Bereich von einigen ms bis hin zu einigen hundert ms, Vergleichsgröße ist die Bearbeitungsgeschwindigkeit für 1k Befehle unterschiedlichen Typs. Die zyklische Verarbeitung setzt sich fort bis der Programmfluß unterbunden wird. Da das Programm in einem nichtflüchtigen Festwertspeicher untergebracht ist, und das Prozeßabbild in der Regel in nichtflüchtigen Speicher gepuffert wird, ist eine derartige Steuerung weitestgehend vor unvorhergesehenen äußeren Einflüßen geschützt und ermöglicht nach einem Totalausfall ein Wiederanlauf ohne Datenverlust.

Die einfache Ablaufstruktur zeigt ihre Stärken bei der Manipulation von Daten und Programmen im laufenden Betrieb. Nahezu alle Programmierumgebungen ermöglichen die Veränderung sämtlicher Variablen innerhalb eines Verarbeitungszyklus. Durch die feste Programmstruktur ist auch das Nachladen von Programmsequenzen zur Laufzeit möglich, da immer exakte Aufsetztpunkte zu Beginn oder Ende eines Zyklus vorliegen.

Nicht zuletzt durch die vielen Freiheitsgrade bei der Programmierung der SPS und der Manipulationsmöglichkeiten während der Laufzeit haben die SPS-Hersteller schnell interessante Konstrukte gefunden, die das Programmieren schneller, einfacher und besser beherschbar machen. Leider wurde vergessen (?) rechtzeitig einen Standard zu etablieren, so daß zwar in bestimmten Regionen defakto Firmenstandards existieren, wie z.B. in

Europa Siemens STEP 5, in USA Allen Bradley oder in Fernost MITSUBISHI, eine verbindliche Vereinheitlichung war jedoch eine Fehlanzeige.

Systemübergreifende Steuerungssoftware

Die IEC-1131-Norm

SPSen findet man in allen Bereichen von Kleinstrechnern bis hin zu komplexen fehlerrendundanten Mehrrechnerarchitekturen. Die gesamte Leistungsbandbreite wird mit am Markt befindlichen Komponenten abgedeckt. Nicht nur die Komplexität der Steuergeräte stieg, auch der Programmieraufwand stieg überproportional. Darüber hinaus ist der Trend zu einem höheren Automatisierungsgrad und einer weitreichenden Internationalisierung des Maschinenbaus zu erkennen.
Unter diesen Randbedingungen war eine weitere Diversifizierung der Sterungssysteme nicht mehr zu akzeptieren. Es verstärkten sich die Anzeichen, daß auch in der Automatisierungstechnik die Nutzung von Synergien und die Vereinheitlichung der Systementwicklung genutzt werden mußten. Es war nur zu offensichtlich, daß propäritäre Lösungen die gestellten Anforderungen nicht mehr befriedigen konnten. Seit 1982, ungefähr zeitgleich mit den VDI-Richtlinien 2880 zu den SPS-Programmiersprachen, wurde so der erste Draft der Internationalen Norm IEC-1131 gezeichnet - ein Schritt in die Vereinheitlichung der Programmier-systematik und der Programmiersprachen für speicherprogrammierbare Steuerungen. 1993 wurde schließlich der Internationale Standard der IEC-1131-3 etabliert, der sich auch in der DIN EN 61131 Teil 3 wiederspiegelt. Heute kann man feststellen, daß sich nahezu alle SPS-Hersteller und im besonderen Hersteller von Software-SPSen an der Norm anlehnen und somit für eine Marktakzeptanz gesorgt haben. Hierzu hat nicht zuletzt auch der Druck der Industrie geführt, der eine weitestgehend Hardware-unabhängige Programmiersprache und -umgebung gefordert hat.
Die IEC-1131 faßt die Anforderungen an ein moderes SPS-System zusammen. Sie ist nicht als starre Spezifikation gedacht, sondern als Richtlinie zur SPS-Programmierung anzusehen. Mit dieser Zielrichtung beschreibt dann auch die Norm die wesentlichen Eigenschaften einer SPS, läßt aber andererseits den Herstellern genügend Freiraum, eine eigene Implementation einzusetzen. In diesem Fall heißt Norm-Konformität lediglich, daß die Norm-gerechten und die Norm-abweichenden Merkmale dokumentiert sind. Die Konformität wird durch durch unterschiedliche Konformitätsklassen nachgewiesen und durch unabhängige Institute zertifiziert.
Die Norm selbst besteht aus mehreren Abschnitten, wobei für die weiteren Betrachtungen im wesentlichen der Teil 3 - Programmiersprachen - berücksichtigt wird.

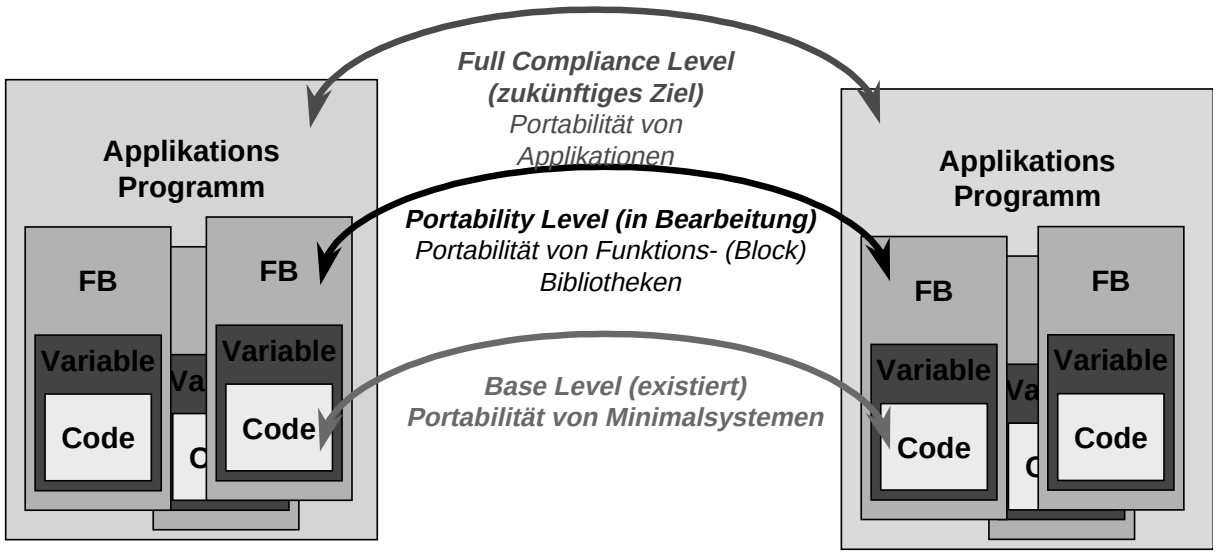
Teil 1	**Allgemeine Informationen**	**Dieser Teil enthält allgemeine Bestimmungen und typische Funktionsmerkmale, die eine SPS von anderen Systemen unterscheidet.**
Teil 2	Betriebsmittelanforderung und Prüfung	Teil 2 definiert die elektrischen, mechanischen und funktionellen Eigenschaften eines Steuerungsgerätes. Hierzu gehören die Umgebungsbedingungen hinsichtlich Temperatur und Luchtfeuchte sowie die Beanspruchungsklassen.
Teil 3	Programmiersprachen	Im Teil 3 wurden die weltweit verbreiteten SPS-Programmiersprachen in einer harmonisierten und zukunftsweisenden Überarbeitung aufgenommen.
Teil 4	Anwenderrichtlinien	Der vierte Teil ist als Leitfaden konzipiert, um dem SPS-Anwender in allen Phasen der Projektierung und der Automatisierung zu beraten.
Teil 5	Kommunikation	Teil 5 beschäftigt sich mit der Interprozeßkommunikation zwischen SPSen unterschiedlicher Hersteller. Dieser Teil ist in Vorbereitung.
Teil 8	Fuzzy Control Language	Dieser Teil der Norm befindet sich in der Abstimmungsphase und erweitert die bestehenden Programmiersprachen um Fuzzy-Logic.

Herstellerunabhängige kompatible Software - die PLCOpen

Die Normung alleine reicht nicht aus, um eine große Akzeptanz bei den Anwendern oder Herstellern von Automatisierungslösungen zu finden. Um der Normung einen hinreichenden Nachdruck zu geben und die Bedürfnisse der Anwender auf die Normung rückzukoppeln startete 1992 die PLCOpen als hersteller- und produktunabhängige internationale Organisation. Wesentliches Ziel dieser Interessengemeinschaft ist die Förderung von Entwicklung und Einsatz kompatibler Software für Speicherprogrammierbare Steuerungen.
Kommitees der PLCopen erarbeiten Normvorschläge in enger Zusammenarbeit mit IEC WG 65B, beraten nationale Normungsgremien, bieten Prüfungen zur Normkonformität an und erarbeiten Vermarktungsstrategien zur internationalen Verbreitung der Norm.
Eine besondere Bedeutung hat hierbei die Erarbeitung von Kompatibilitätslevel für Programmier-sprachen der IEC1131-3. Drei Ebenen der Kompatibilität wuden seitens der PLCopen definiert:

- Base Level,
- Portability Level,
- Full Compliance Level.

Innerhalb der Kompatibilitätstests sind Testprozeduren definiert, um Produkte durch unabhängige Testinstitute zertifiziern zu lassen. Zum derzeitigen Zeitpunkt sind eine Reihe von Baselevel - zertifizierten Produkten am Markt. Die Portabilitäts-Ebene ist weiterer Definition unterlegen, die die Kompatibilität auf Bibliotheks- bzw. Funktionsblockebene nachweist. Ziel ist den "Full Compliance Level" zu erreichen, in dem Applikationen auf beliebigen zertifizierten Plattformen, ohne eine weitere Programmänderung, lauffähig sind.



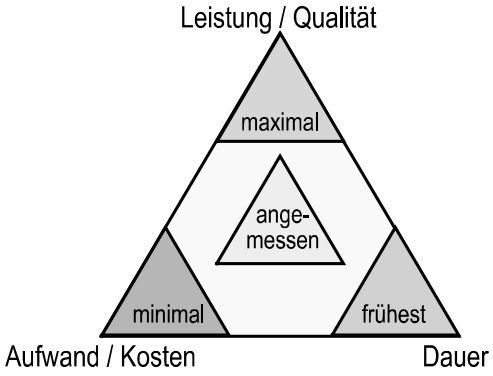
Kompatibilitätslevel nach IEC1131-3

Grundlagen der Norm IEC-1131-3

Softwareengineering

Die Vereinheitlichung der Programmiersprachen ist ein Teil der Norm. Genauso interessant ist der erstmals in der Steuerungstechnik vollzogene Ansatz, modernes Softwareengineering in den Entwicklungszyklus zu integrieren. Dieses wurde auch dringend notwendig, da die gestiegenen Anforderungen hinsichtlich der Komplexität der Abläufe und der Funktionen an den SPSen nicht vorübergegangen sind und auch hier zu der viel zitierten Softwarekriese geführt haben. Die fatalen Auswirkungen sind dem Software-Ersteller und -Benutzer nur zu bekannt. Projektbudgets steigen an, Zeiten werden nicht eingehalten, die Funktionalität entspricht nicht den Erwartungen und die Dokumentation beschreibt eher die Unternehmensgeschichte als den Projektstand.

Der Dreikampf zwischen Kosten-Qualität und Dauer zwingt auch den SPS-Softwareentwicklern ein strukturiertes Vorgehen auf



Die Wiederverwendung ausgetesteter und standardisierter Software-Bausteine erscheint auch hier die Lösung, was aber durch das häufig in der SPS-Welt angewendete Programmiermodell mit direkter Adressierung erschwert wurde. Auch das Arbeiten von untypisierten Variablen ohne Typenprüfung treibt den Hochsprachenprogrammierer Schweißperlen auf das Gesicht und ist ein direkter Wiederspruch zu einer sicheren Programmierung. Dennoch muß dem speziellen Charakter der SPS hinsichtlich online-Parametrierung und Flexibilität Rechnung getragen werden. Allgemein kann man die Zielsetzung der IEC 1131 festhalten zu:

- Einsatz von Softwareengineering-Methoden mit der Zielsetzung der Wiederverwendung von Softwarebausteine.
- Ganzheitliche Betrachtung von Problemlösungsansätzen
- Abstraktion von komplexen Aufgaben in überschaubare Module
- Definition von eindeutigen Schnittstellen
- Vereinheitlichung des Sprachumfangs zur Erhöhung der Portabilität

Das Programmiermodell

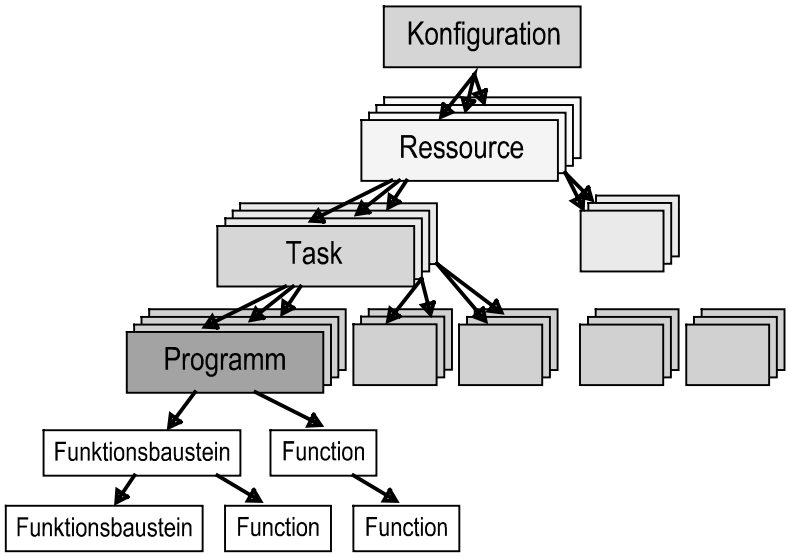
Mit der IEC 1131 wurde eine Basis für eine einheitliche SPS-Programmierung geschaffen, die wesentliche Sprachelemente einer modernen Software-Technologie berücksichtigt. Hierbei wird nicht nur die SPS Programmierung selbst, sondern umfassende Konzepte zum Aufbau eines SPS-Projektes festgelegt. In der Norm IEC1131 werden neben den Elementen zur Programmierung und Organisation des Anwenderprogrammes auch Richtlinien zur Modellierung und Strukturierung von SPS-Verbundsystemen gegeben. Zur Systemstrukturierung werden die die Begriffe "Konfiguration" und "Ressource" eingeführt, die im weiteren näher erläutert werden.

Prinzipiell geht Norm geht bei der Definition der Begriffe von einer maximal leistungsfähigen SPS aus. Insbesondere Eigenschaften wie

- Multiprozessorsysteme,
- moderne SPS Betriebssysteme mit Multitasking – Eigenschaften,
- unbegrenzte Anzahl analoger und digitaler Ein- und Ausgänge,
- Kommunikationsfähigkeit zu SPS oder Rechnern ,

werden berücksichtigt. Eine Konfiguration definiert die Struktur eines Gerätes. Beispielsweise kann dieses eine SPS mit mehreren unter umständen auch vernetzten Zentraleinheiten (CPU) auf Maschinenzellen-Ebene sein.
Eine Konfiguration beinhaltet eine oder mehrere Resourcen, die Teilsteuerungen mit eigener "Signalverarbeitungsfunktion" darstellen. In einer realen SPS Konfiguration wird eine Resource von einer meist multitasking-fähigen SPS CPU repräsentiert.
Die Strukturierung einer Resource erfolgt durch ein oder mehrere Programme, die von Tasks gesteuert werden. Unter einer Task wird eine ablauffähige Programmeinheit verstanden, der eine Priorität und ein Ausführungstyp zugeordnet wird. Hierdurch wird es möglich, daß innerhalb eines Programmes Ausführungsfäden mit unterschiedlichen Charakteristika zu formulieren. So sind nicht nur Zyklische Tasks mit einer systemweit einheitlichen Zykluszeit möglich. Vielmehr können Zykluszeiten kombiniert werden oder gar ereignisgesteuerte Programmeinheiten im System bereitgestellt werden. Über die Priorisierung der Tasks erfolgt die Zuteilung CPU-Zeit innerhalb einer Ressource.
Durch die Zuordnung von Programmen zu einer Task werden die Laufzeiteigenschaften des Gesamtprogramms definierten, welches selbständig in einer CPU ablaufen kann. Die Flexibilität der Systemmodullierumg läßt zu, daß ein Programm zu mehreren Tasks gehört. Hierdurch werden mehrere Instanzen mit unterschiedlichen Laufzeiteigenschaften erzeugt.

Eine Konfiguration besteht aus mehreren Ressourcen, die wiederum unabhängige Tasks bzw. Programme beinhalten können



Die IEC1131-3 unterstützt lokale Daten, die in Programmen, Funktionsbausteinen oder Funktionen deklariert werden können. Lokale Daten sind nur in der jeweiligen Programmhierarchieebene zugänglich und stellen damit einen Mechanismus zur Datenkapslung dar. Selbstverständlich sind auch Ressourcen-weit zugängliche Globale Daten ermöglicht, die für alle Programmelemente zugänglich sind. Beim Einsatz von Mutlitaskingsysteme stellen jedoch Zugriffe auf globale Daten eine Risikoquelle für inkonsitente Daten dar. Darüber hinaus existieren direkt zugängliche Daten mit festen Adressen innerhalb des SPS Adressraums. Dieses sind in der Regel die Adressen der Eingänge (Input, I), Ausgänge (Output, O) und Merker (M, auch: Flags).
Ein anderer Aspekt des Programmiermodells beschreibt das Wiederanlaufverhalten der Steuerung. Beschrieben werden sowohl der Kaltstart als auch der Warmstart. Bei einem Kaltstart wird das Programm neu geladen. Alle

Variablen werden auf Ihren Initialwert gesetzt. Entweder wird ein „Default“-Initialwert oder der vom Programmierer definierte Wert gesetzt. Alle Tasks der Resource werden gestartet. Demgegenüber werden bei einem Warmstart (Wiederanlauf) die Variablen nicht auf Ihren Initialwert gesetzt, sondern es werden die vor der Unterbrechung zum Warmstart vorhandenen Werte im Speicher übernommen.

Das Kommunikationsmodell

Ein wesentlicher Aspekt bei der Beschreibung der Strukturelemente ist der Datenaustausch. Mit Hilfe des in der IEC 1131 definierten Kommunikationsmodells wird es möglich, gut strukturierte und vor allem modularisierte SPS-Programme zu erstellen. Dieses ist eine fundamentale Basisigenschaft zur Entwicklung anwendungsgerechter, wiederverwendbarerer Programm-Module. Durch diese Eigenschaft ist eine nachhaltig positive Veränderung beim SPS Programmentwurf zu erwarten..
Prinzipiell sind in der IEC1131 sind die folgenden Kommunikationsmöglichkeiten vorgesehen:

- Zugriffspfade (VAR_ACCESS)
- Globale Variable (VAR_GLOBAL, VAR_EXTERNAL)
- Aufrufparameter
- Kommunikations-Bausteine (IEC 1131-5)
-

Sämtliche Elemente einer Konfigurationen kommunizieren untereinander oder auch mit weiteren Rechnersystemen ausschließlich über definierte "Zugriffspfade". Darüber hinaus dienen "Globale Variablen" der einfachen Kommunikations von Programmen innerhalb einer Konfiguration. Globale Variablen können auf der Ebene von Konfiguration, Ressource und Programm angelegt und benutzt werden.
Der Datenaustausch innerhalb von Programmen erfolgt mittels "Aufrufparameter", bzw. Ein- Ausgangsvariablen oder Funktionswerte. Dieses Strukturmittel ist dem Hochsprachenprogrammierer wohl vertraut, bringt aber grundsätzlich neue Aspekte in die bisherige SPS-Programmierung. Aufrufparameter und Transfervariablen ermöglichen die Definition eindeutiger Schnittstellen und bringen so einen wichtigen Beitrag zur Kapselung von Funktionalität.
Neben den bisher beschriebenen Elementen des Kommunikationsmodells können auch spezielle "Kommunikations-Bausteine" Verwendung finden. Diese sind monolithischer Natur und werden zu einem Programm gebunden. Der Datenaustausch zwischen Sender und Empfänger wird durch diese Bausteine vollkommen autark realisieren. Kommunikationsdienste werden im Teil 5 der IEC 1131 definiert, der sich allerdings noch in Bearbeitung befindet.
Betrachtet man das Kommunikationsmodell der IEC1131, so stellt man im besonderen die gute Unterstützung standardisierter Software-Bausteine fest. Durch die Kapselung von Funktionalität und Daten, einer klar definierte Schnittstelle und einem "rückwirkungsfreiem" Verhalten wird die Akzeptanz der Module beim späteren Anwender nachhaltig gesteigert.

Allgemeine Eigenschaften und Datentypen

Gerd Hoppe, Beckhoff Industrieelektronik

Eine wichtige Vereinheitlichung der Programmiersprachen nach IEC1131-3 ist die Definition von Datentypen. Die Norm kennt verschiedene elementare Datentypen, aus denen abgeleitete und benutzerdefinierte Datentypen zusammengestellt werden können.
Die elementaren Datentypen sind:

- Bit,
- Byte,
- Integer,
- Real,
- Zeit- und Datum – Typen
- String.

Elementare Datentypen
Die folgende Tabelle enthält die in der Norm definierten emlementaren Datentypen sowie – soweit sinnvoll - deren Länge.

Name	Beschreibung	Anzahl Bits
BOOL	Single Bit	1
BYTE	Bit Array of 8 Bits	8
WORD	Bit Array of 16 Bits	16
DWORD	Bit Array of 32 Bits	32
LWORD	Bit Array of 64 Bits	64
SINT	Short Integer	8
INT	Integer	16
DINT	Double Integer	32
LINT	Long Integer	64
USINT	Unsigned Short Integer	8
UINT	Unsigned Integer	16
UDINT	Unsigned Double Integer	32
ULINT	Unsigned Long Integer	64
REAL	Real Numbers	32
LREAL	Long Real Numbers	64
TIME	Time duration	32
DATE	Calendar date	
TIME_OF_DAY oder TOD	Time of day	
DATE_AND_TIME oder DT	Date and time of a day	
STRING	Character Strings	

Datentypen - Notation

Die oben aufgelisteten Datentypen können in unterschiedlichen Darstellungsweisen aufgeführt werden:

BOOL, BYTE, WORD, DWORD, LWORD

Diese Datentypen können wie folgt dargestellt werden:

- TRUE oder 1
- FALSE oder 0
- Dezimale, hexadezimale (16#), oktale (8#) oder binäre (2#) Darstellung

Beispiel für WORD: 234, 16#ff, 2#1001_1100_0011_1111

SINT, INT, DINT, LINT, USINT, UINT, UDINT, ULINT

Dezimale, hexadezimale (16#), oktale (8#) oder binäre (2#) Darstellung

Der Unterstrich (_) trennt Einheiten

Beispiele:

Dezimale Darstellung für INT: -123, +234, 0, 1_000

Hexadezimale Darstellung für INT: 16#F1, 16#0A_1B

Binäre Darstellung für INT: 2#0001_0011_0111_1111

REAL, LREAL

Normale dezimale Darstellung mit Dezimalpunkt oder Exponentielle Darstellung

Beispiel: 1000.23 und 1.23e3 und 1.23E3 und 1.23E03 werden identisch interpretiert

TIME (Zeitdauer)

TIME#, t# oder T# steht am Anfang einer Zeit/Datum - Bezeichnung

Überlauf ist gestattet (z.B. 25 Stunden)

d steht für Tage, h für Stunden, m für Minuten, s für Sekunden und ms für Millisekunden

Der Unterstrich (_) trennt Einheiten

Beispiel: T#2d_26h_4m_12s_123ms

DATE, TIME_OF_DAY oder TOD, DATE_AND_TIME oder DT

DATE# oder D# steht für ein Datum

TIME_OF_DAY# oder TOD# steht für Tageszeit

DATE_AND_TIME# oder DT# steht für Tageszeit und Datum

Datum: D#1998-12-07 steht für 7. Juli 1998

Tageszeit Notation: TOD#12:00:00.123

Datum und Zeit: 1998-12-07-12:00:00.123

STRING

Hochkomma ‘ ‘ rahmen einen String ein

Das Dollarzeichen $ führt Steuerzeichen an (Line feed, Tabs)

Beispiele:

String, Steuerzeichen: ‘This is a line feed character $L‘

Leerer String: ‘‘

Abgeleitete Datentypen - Notation

Abgeleitete Datentypen erlauben es dem Programmierer, komplexe Strukturen mit einfachen Konstrukten zu verwalten: Funktionen und Funktionsbausteine können mit nur einer Zuweisung komplexe Daten übergeben werden. Die abgeleiteten Datentypen können aus den Basisdatentypen und aus abgeleiteten Datentypen zusammengestellt werden.

Eine abgeleitete Datentype wird mit der Klammer TYPE .. END_TYPE definiert.

Beispiel:

```
TYPE
    myOwnReal :     REAL;
END_TYPE

TYPE
    myArray     :   ARRAY[0..1000] OF BOOL;
END_TYPE
```

Datentypen – Strukturen

Mit der Nutzung von Strukturen können unterschiedlichste Basisdatentypen zu einem „Paket“ verschnürt und mit einem Namen gehandabt werden. Die einzelnen Elemente der Struktur bleiben weiterhin auch einzeln für Zugriffe erreichbar.

Strukturen werden in einer STRUCT .. END_STRUCT Klammer definiert.

```
TYPE myStruct:
    STRUCT
        status :    BOOL;
        inputValue  :   REAL;
    END_STRUCT
END_TYPE
```

Datentypen – Enumerated

„Enum“ Datentypen sind Teil der IEC1131-Definition von Datentypen: Mit Hilfe einer Deklaration werden sie verfügbar. Werte können den einzelnen Elementen einer Enum- Funktion zugeweisen werden (siehe Beispiel):

Beispiel:

```
TYPE Modes:
    (Initialisation := 0, Running, Idle, Reset, Faulty);
END_TYPE
```

Der Wert von Initialisation ist gleich 0, Running ist gleich 1.

Mehrdimensionale Felder

Ein Array (Feld) ist eine Vereinigung von Daten mit gleichem Datentyp, der ein elementarer oder definierter Datentyp sein kann. IEC1131-3 definiert Arrays mit bis zu drei Dimensionen.

Beispiel:

```
TYPE matrix:
ARRAY[1..23, 0..1] OF INT;
END_TYPE
```

Variablen

Lokale und globale Variablen aller Datentypen werden deklariert (ebenso wie die Datentypen). Das Schlüsselwort dazu ist der Rahmen: VAR..END_VAR. Innerhalb dieses Rahmens werden Variablen gleichen Typs aufgeführt und dabei durch Komma getrennt.

IEC1131-3 kennt fünf unterschiedliche Variablenklassen:

- Globale Variablen,
- Lokale Variablen,
- Eingangsvariablen

- Ausgangsvariablen
- Ein- und Ausgangsvariablen

Die Bezeichnungen sind im allgemeinen selbsterklärend. Eingangs, Ausgangs- sowie Ein-/Ausgangsvariablen werden bezogen auf ein Programm, eine Funktion oder einen Funktionsblock, sie können nur in der spezifischen Weise benutzt werden. Innerhalb der zugeorneten Programmeinheit (POU) können sie lesend und schreibend verändert werden, außerhalb nur in der definierten Weise.

Variablen - Deklaration

Lokale Variablen werden innerhalb eines Quellcodes lokal definiert, sie sind nur innerhalb der POU zugänglich, was zur Übersicht beiträgt und Seiteneffekte vermeidet.

Globale Variablen werden – für alle POUs erreichbar – mit dem entsprechenden Schlüsselwort deklariert. Der Anwender muß selbst für die Bereinigung von Seiteneffekten Sorge tragen.

Ebenso wie globale Variablen werden Eingangs- Ausgangs- und Ein-Ausgangsvariablen per Schlüsselwort deklariert.

Beispiele:

Deklaration von lokalen und globalen Variablen:

```
VAR
a,b,c  :      REAL;
d,e    :      BOOL;
f      :      ARRAY[1..12] OF BOOL;
END_VAR

VAR_GLOBAL
      var    :      UDINT;
END_VAR
```

Deklaration von Input – Variablen

```
VAR_INPUT
      a,b,c  :      REAL;
END_VAR
```

Deklaration von Output – Variablen

```
VAR_OUTPUT
      d,e    :      INT;
END_VAR
```

Deklaration von In_Out – Variablen

```
VAR_IN_OUT
      x      :      STRING;
END_VAR
```

Variablen - Attribute

- RETAIN: Variablenwerte werden über Abschalten und Spannungswiederkehr beibehalten
- CONSTANT: Variablenwerte können nicht geändert werden
- AT: Variablen besitzen einen festen Platz im Speicherabbild (feste Adresse)

Lokierte – fest adressierte - Variablen

Diese Variablen sind der herkömmlichen SPS Technik angelehnt. Sie besitzen eine gewisse Bedeutung in IEC1131-3 Systemen, weil zwei Eigenschaften damit verbunden werden können:

- alle Prozeßsignale werden über lokierte Variablen angebunden
- Überlappungen von lokierten Variablen sind gestattet und können als Mittel der Programmierung eingesetzt werden.

Die Festlegung einer bestimmten Speicherstelle für Variabeln beginnt mit dem „AT“ Bezeichner und definiert drei Parameter:

- die Startadresse (nach dem „AT“ Bezeichner als Offset von Null)
- den Speicherbereich (Eingang, Ausgang, Merker)
- die Länge der Variablen im Speicher (mittels einer Variablentypenabkürzung)

Der erste Buchstabe der Längendefinition ist ein „%“, der zweite ein

- I für Input (Eingang), Q für Output (Ausgang), M für Merker (Flag),
- die Länge der Variablen im Speicher (mittels einer Variablentypenabkürzung)

der dritte definiert die Länge:

- X für Bits (hier wird die Bitadresse N stets in „Byte.N“ ergänzt, z.B. 1.0),
- B für Byte,
- W für Word,
- D für Double und
- L für Long Word.

Beispiele:

%IB24, %QX1.1, %MW12

Überlappende Lokation von Variablen

Überlappende Lokation von Variablen ist gestattet, so ist z.B. %MB12 das erste Byte von %MW12 und das erste Byte von %MD12. Auch auf der Bitebene ist diese Überlappung gewünscht und sinnvoll einzusetzen: %MX12.0 ist das erste (niderwertigste) Bit von %MB12.

Initialisierung von Variablen

Grundsätzlich wird nach einem Kaltstart jede Variable initialisiert. Der Grundwert (Default) beträgt üblicherweise 0 oder FALSE. Eine anwenderspezifische Einstellung auf einen anderen Wert ist natürlich möglich und wird mit einem = Zeichen in der Deklaration zugewiesen.

Die Initialisierung kann auch innerhalb von abgeleiteten Datentypen (Arrays, Strukturen) vorgenommen werden. Das Beispiel zeigt die notwendige Syntax.

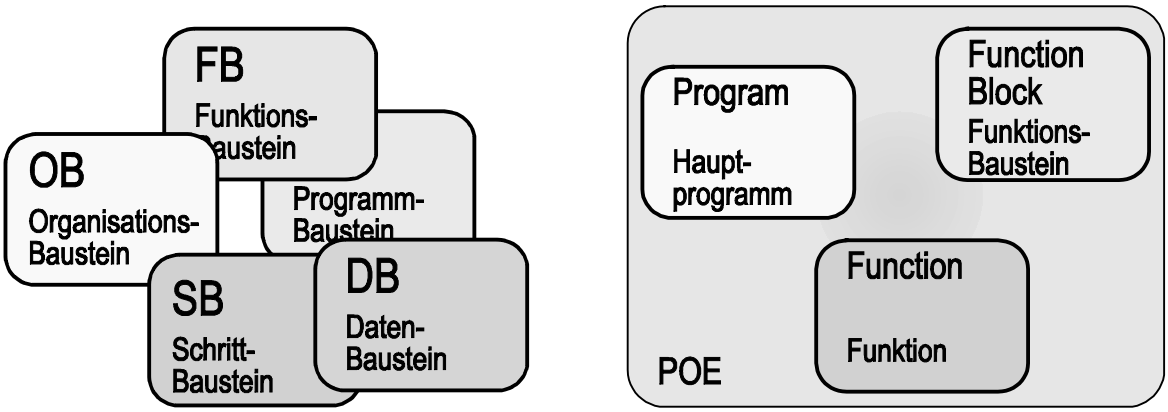
Beispiel:

```
VAR
a       :       INT := 13;
b       :       STRING := ‘this is a string‘;
c       :       REAL := 1.1;
END_VAR

VAR
```

```
a       : myStruct :=
                (
                        status := TRUE,
                        inputValue := 2.5
                );
END_VAR

VAR
a       : ARRAY[1..10] OF INT :=
          1, 2, 2(4), 5, 6, 7, 8, 9,10;
END_VAR
```

Programmorganisationseinheiten POE

Die IEC 1131-3 versucht die Vielfalt der in bestehenden, herstellerspezifischen SPS-Programmiermodellen vorhandenen Bausteinarten zu beschränken. Mußte man sich bisher mit Organisations-, Funktions-, Schritt-, Daten- und Programmbausteinen herumschlagen, wird die Programmorganisation über sogenannte Programmorganisationseinheiten (POEs) deutlich homogenisiert. Im besonderen hatte in bisherigen Systemen die implizite, nicht transparente und herstellerabhängige Bedeutung einzelner Bausteine oder Bausteinbereiche selbst erfahrenen Programmentwickler den Umstieg auf eine SPS eines anderen Herstellers erschwert oder gar unmöglich gemacht. Sollten dann auch noch komplexe und standardisierte Bausteine übernommen werden, konnte ein Projekt schon mal zu einem unkalkulierbaren Risiko werden. Die Problematik gipfelt darin, daß selbst verschiedene CPU-Familien eines einzigen Herstellers Unterschiede aufweisen. Mit der Strukturierung einer POE in Programm, Function und Functionblock wird eine überschaubare Tiefe gewählt, die es ermöglicht, die gesamte anwendungsspezifische Implementierung definiert zu verwalten.



Die in der SPS-Welt übliche Bausteinvielfalt weicht einer klar strukturierten Organisation in Programmorganisationseinheiten.

Das "Programm" bildet das Hauptprogramm mit Zuordnung der SPS Peripherie, globalen Variablen und Zugriffspfaden. Eine "Funktion" beschreibt eine komplexe Verknüpfungslogik, die aber kein "Gedächtnis", also keine statischen Variablen besitzt. Kennzeichnend an einer Funktion ist, daß sie bei gleichen Eingangsgrößen stets dasselbe Ergebnis zurückgibt.
Braucht man einen intelligenten Baustein mit Gedächnis, bietet der "Funktionsbaustein" (FB) mit lokalen statischen Variablen die notwendige Basis. Ein FB kann bei gleichen Eingangsgrößen unterschiedliche Ergebnisse zurückgeben (z.B. bei einem Zeit- oder Zählerbaustein). Jede Instanz eines FBs besitzt nach der neuen Norm ihren eigenen "gekapselten" Datenbereich, auf dem seine Berechnungen ausgeführt werden: die Instanz (siehe unten).
Um typische SPS-Funktionalitäten zu standardisieren, führt die neue Norm Standard-Funktionen und Funktionsbausteine ein. Diese "Bibliothek" bildet eine wichtige Grundlage für eine einheitliche, herstellerunabhängige Programmierung von SPS-Systemen.
Zusammenfassend kann man eine POE als eine Einheit verstehen, die vom Compiler einer SPS-Programmierumgebung, unabhängig von anderen Programmteilen, übersetzt werden kann. Die Eigenschaften einer POE ermöglicht eine weitreichende Modularisierung der Anwenderprogramms und eine Wiederverwendung bereits implementierter und getesteter Software-Bausteine. Um Programmodulen den Zugriff auf POEs zu ermöglichen, wird mindestens die Deklaration der Aufrufschnittstelle benötigt (Prototyp). Übersetzte Programmteile könne später zu einem gemeinsamen Gesamtprogramm zusammengebunden werden (Linker). Entgegen manchen Hochsprachen kennt die IEC 1131 allerdings keinen Gültigkeitsbereich von POEs. Der Name einer POE ist in einem Projekt global und kann nicht mehrfach vergeben werden. Eine POE steht nach ihrer Deklaration allen anderen POEs global zur Verfügung.

Aufbau von Programmorganisationseinheiten

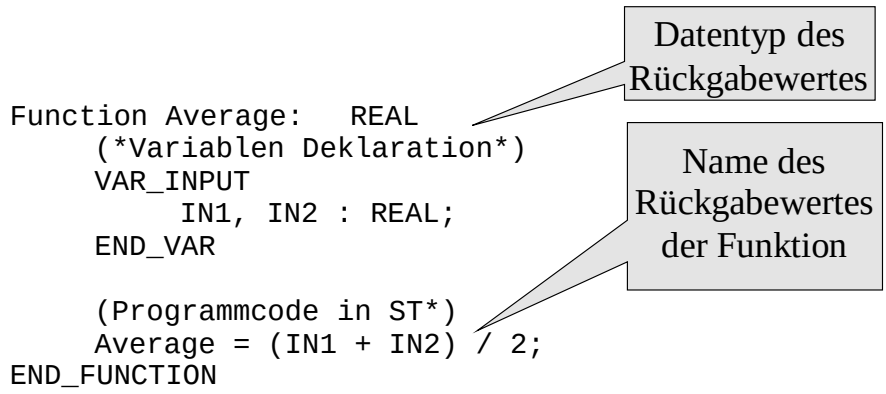
Gerd Hoppe, Beckhoff Industrieelektronik

Funktionen
Funktionen speichern Ihre intern berechneten Daten nicht von Aufruf zu Aufruf: sie werden bei jedem Aufruf temporär bearbeitet. Funktionen können mehere Eingabewerte, jedoch nur einen Ausgabe- (Rückgabe-) Wert besitzen. Die Bearbeitung einer Berechnung erfolgt direkt und seiteneffektfrei. Da keine Zwischenspreicherung von lokalen Daten erfolgt, sind nicht gestattet:

- Nutzung globaler Variablen,
- Aufruf von Funktionsbausteinen, z.B. für Zeiten, Zähler, Flankenerkennung,
- Deklaration von direkt adressierten Variablen.
-

Funktionen können in allen Sprachen, jedoch nicht in SFC, codiert werden. Der Rückgabewert ist naturgemäß mit dem Namen der Funktion deklariert.

Beispiel:




```
Function Average:   REAL
     (*Variablen Deklaration*)
     VAR_INPUT
          IN1, IN2 : REAL;
     END_VAR

     (Programmcode in ST*)
     Average = (IN1 + IN2) / 2;
END_FUNCTION
```

Einige Funktionen (auch: Operationen) können mehrere Datentypen gleichzeitig verarbeiten:

```
a, b   :      REAL;
c, d   :      INT;

a      := ABS(b);   (* nutzt REAL für Input und Output*)
c      := ABS(d);   (* nutzt INT für Input und Output *)
```

Funktionen zur Typkonvertierung

Funktionen müssen verwendet werden, um Variablen unterscheidlicher Typen einander zuzuweisen, z.B. eine Realvariable und eine Integervariable:

```
a      :      REAL;
b      :      INT;

b      :=     REAL_TO_INT(a);
```

Standard IEC 1131-3 Funktionen

Die Liste zeigt die von IEC1131-3 definierten Standardfunktionen, die von den meisten Implementierungen unterstützt werden. Unterschiede bestehen oft in der Unterstützung aller Datentypen für dies Funktionen, sowie un der Austattung mit zusätzlichen funktionen, die über den definierten Umfang hinausgehen.

Bit Array	**AND, OR, XOR, NOT, SHL, SHR, ROL, ROR**
Numerisch	ADD, SUB, MUL, DIV, MOD, EXPT, ABS, SQRT, LN, LOG, EXP, SIN, COS, TAN, ASIN, ACOS, ATAN
Typ Konvertierung	e.g. BOOL_TO_BYTE, REAL_TO_DINT
Selektion	SEL, MIN, MAX, LIMIT, MUX
Vergleich	GT, GE, EQ, LT, LE, NE
String	LEN, LEFT, RIGHT, MID, CONCAT, INSERT, DELETE, REPLACE, FIND

Funktionsblöcke

Funktionsblöcke werden verwendet, um Eingänge, Ausgänge und interne Variablen zu setzen, Zustände eines FB- Aufrufs werden von Zyklus zu Zyklus zwischengespeichert. Der Programmcode des FB erzeugt dabei Veränderungen der Ein- und Ausgänge sowie der internen Variablen.

Vom aufrufenden Programm aus sind nur die Ein- und Ausgangsvariablen des FB erreichbar. Aufrufe von anderen FBs sind gestattet, und zwar von allen Sprachen in alle Sprachen hinein.

Beispiel:

```
FUNCTION_BLOCK Counter
     VAR_INPUT
          Mode : INT;    (*0=Reset, 1=Count*)
     END_VAR
     VAR_OUTPUT
          Out : INT;     (*aktueller Zählwert*)
     END_VAR
     IF Mode = 0
     THEN
          Out == 0;      (*Reset*)
     ELSEIF Mode = 1
     THEN
          Out := Out + 1;
     ENDIF;
END_FUNCTION_BLOCK
```

Addiert 1 zum alten Zählerwert um den neuen Zählerwert zu erhalten

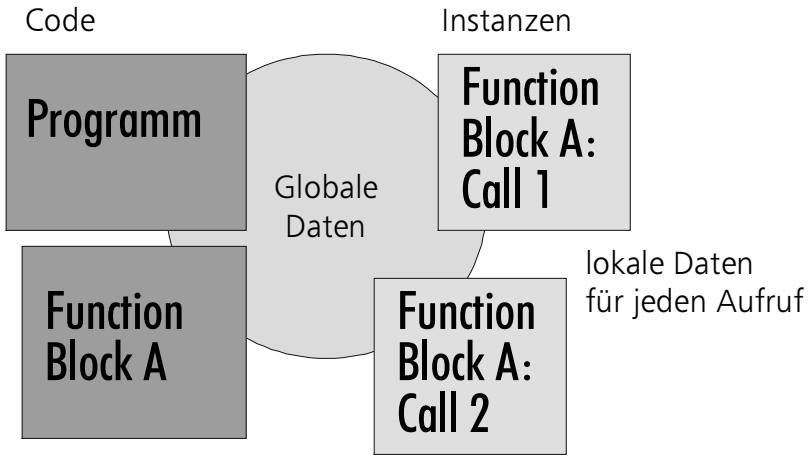
Instanzierung

Die IEC1131-3 sieht die Instanzierung von Funktionsblöcken vor: eine Instanz ist eine Struktur, in der alle internen Variablen, Eingänge und Ausgänge eines Aufrufs eines FB gespeichert werden: Also besitzt ein Programm, welches FB1 fünfmal aufruft, 5 Instanzen von FB1, eine für jeden Aufruf. Der Vorteil dieser ungewohnten „objektorientierten“ Vorgehensweise: die Programmdiagnose kann aufrufgenau und seiteneffektfrei erfolgen. Moderne Tools helfen mit einer automatischen Deklaration, diese Instanzierung durchzuführen: für einen Aufruf eines FB wird einfach ein Instanzname festgelegt, der die Daten dieses Aufrufs verwaltet.

Wichtiges Merkmal: alle Instanzen verwenden den gleichen Programmcode des FB. Änderungen am Programmcode wirken sich also in allen Aufrufen gleich aus. Eine Instanz ist also keine Kopie des FB für einen Aufruf.

Beispiel:

Für jede Instanz wird eine Kopie der Datenbereiche angelegt



Funktionsblock A wird in zwei Aufrufen instanziert: es exisitiert für jeden Aufruf eine Struktur (mit dem Namen der Instanz), die die aufrufspezifischen Variablenwerte seiteneffektfrei enthält.

Die Instanz 1 und Instanz 2 in dem oben gezeigten Beispiel sind lokale Daten (Strukturen) im rufenden Programm. Diese lokalen Instanzdaten können Eingangsvariblen für andere Funktionsblöcke oder Programme sein .

Die für den traditionellen SPS Programmierer ungewohnte Instanzierung ermöglicht eine seiteneffekfreie Verarbeitung von Variablen und deren Diagnose mit dem Programmiertool, wie es in modernen Programmierumgebungen auf PC Basis seit langem Standard ist. Der sehr angenehme Nutzen der unbedingten Seiteneffektfreiheit für die Diagnose des Programmablaufes rechtfertigt den Mehraufwand der Instanzierung bei weitem: in älteren SPS Umgebungen wurde dazu in der Praxis der Code eines FB mehrfach kopiert und Programmänderungen danach in jeder Codekopie ausgeführt.

Standard IEC 1131-3 Funktionsblöcke

Die Liste zeigt die von IEC1131-3 definierten Standardfunktionsblöcke, die von den meisten Implementierungen unterstützt werden. Sinngemäß gilt das Gesagte für die Standardfunktionen.

Bistabile	SR, RS, SEMA
Flankendetektion	R_TRIG, F_TRIG
Zähler	CTU, CTD, CTUD
Zeiten	TP, TON, TOF, RTC

Programme

Programme sind die übergeordnete Programmogranisationseinheiten: Ein Programm ruft Funktionen und Funktionsbausteine auf, in einigen Implementierungen auch andere Programme. Wie stets, wird ein Programm aus globalen und lokalen Variablen und einem Codeteil gebildet. Programme können selbstverständlich in allen Sprachen geschrieben werden.

Im Gegensatz zu Funktionsbausteinen werden Programme nicht instanziert. Sie besitzen keine Erinnerungsfunktion für lokale Daten, falls sie mehrfach aufgerufen werden.

Programmbeispiel:

```
PROGRAM Main
    VAR
        counter_1 : Counter;      (* Instanz von FB Counter *)
        actCount  : INT;
    END_VAR
    IF bfirstCycle
    THEN
        counter_1(Mode := 0);     (* Aufruf von FB Counter mit
Resetmode *)
    ELSE
        counter_1(Mode := 1);     (* Aufruf von FB Counter mit
Zählermode*)
    END_IF
    actCount := counter_1.Out;    (* Zugriff auf Ausgangsvaria-
ble von counter_1*)

END_PROGRAM
```

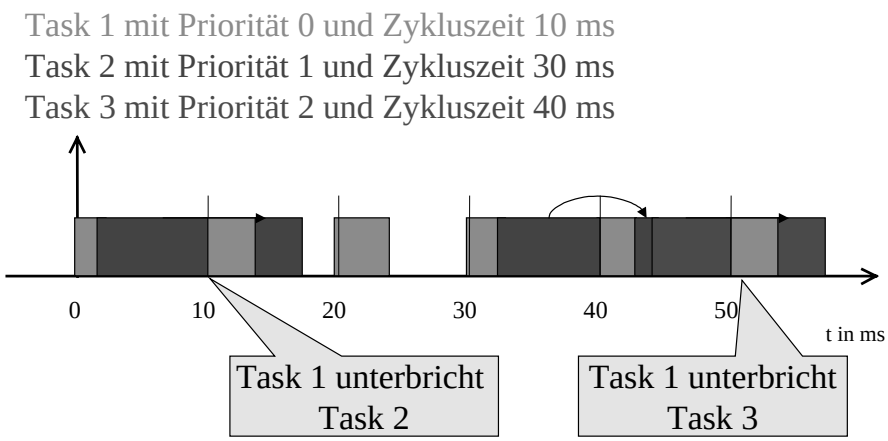
Tasks

Tasks sind Elemente zur Ablaufkontrolle, die den Programmen übergeordnet sind: Tasks rufen Programme auf. Üblicherweise sind Tasks mit Eigenschaften ausgestattet, die den Programmablauf steuern:

- Priorität,
- Zykluszeit,
- Ereigniskoppelung.

Auf diese Weise können Programme mit unterschiedlicher Zykluszeit zyklisch bearbeitet werden oder durch ein Ereignis gesteuert anlaufen.. Je nach Implementierung können preemptive oder nicht preemptive Taskschedulingverfahren eingesetzt sein. Um eine Task ohne zeitlichen Aufrufjitter ausführen zu können, muß sie die höchste Priorität besitzen.

Taskverwaltung in einem preemtiven Multitaskingsystem



Die Programmiersprachen

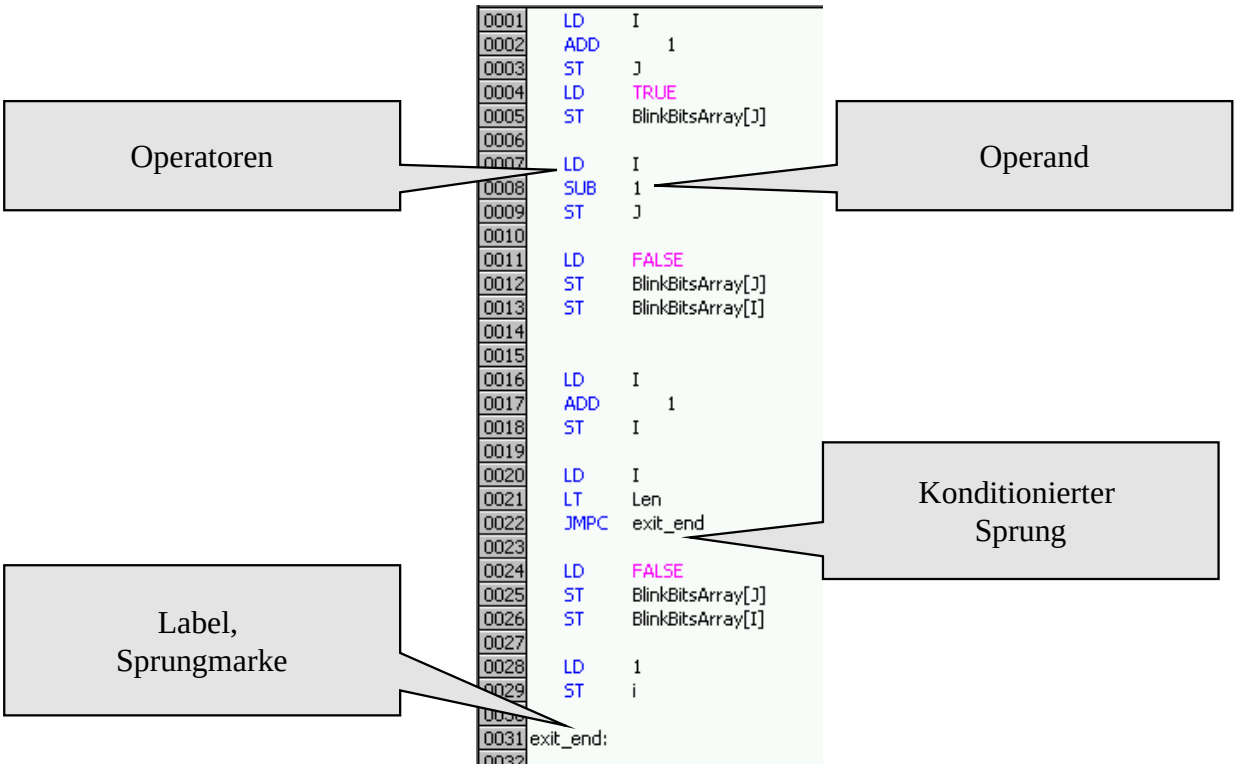
Mit der Normung der Programmiersprachen ist das so eine Sache. Hier gibt es durchaus Bemühungen, die bis in die achziger Jahren zurückgehen. Leider haben sich jedoch immer Firmen-spezifische Dialekte als Quasi-Standard etabliert, so daß eine Vereinheitlichung bisher eher Wunsch als Realität ist. Hier geht die IEC-1131-3 den konsequenten Weg, umfangreiche Definitionen von Sprachelemente und Programmierungsarten international zu etablieren. Gleichzeitig läßt sie jedoch soviel Freiheiten, daß Hersteller-spezifische Eigenheiten in die Sprache übernommen werden können. Der Schwerpunkt der Norm liegt in diesem Fall auf der Dokumentation der Normabweichungen.

Als Programmiersprache werden Ablaufsprache (AS) , Anweisungsliste (AWL), Kontaktplan (KOP), Funktionsbausteinsprache (FBS) und Strukturierter Text (ST) unterstützt. Jede Sprache für sich hat spezielle Anwendungsfälle bzw. Ausprägungen ist zur Lösung von bestimmten Problemfällen im besonderen geeignet. Die Norm schreibt die Syntax der Programmiersprachen exakt vor, so daß der Anwender davon ausgehen kann, daß in allen IEC-1131-konformen Entwicklungspaketen zumindest die identische Syntax verwendet wird. Dennoch ergeben sich erhebliche Unterschiede in der Implementierung und der Bedienoberflächen, so daß genaues Hinschauen beim Vergleich dere Systeme unabdingbar ist.

AWL

AWL (Anweisungsliste) oder IL (Instruction List) ist der Assembler unter den Programmiersprachen für SPSen. Sie ist die Urmutter der SPS-Programmierung und vorwiegend im europäischen Raum verbreitet. Im

Laufe der Entwicklung haben sich vielfältige Dialekte von AWL entwickelt, die durch die Neufassung in der IEC 1131-3 wieder auf eine einheitliche Basis gestellt werden. AWL eignet sich gut zur Verarbeitung von einfachen sequentiellen Programmen. Sobald Schleifenkonstrukte notwendig werden, neigt AWL zur Unübersichtlichkeit.



AWL bzw. IL ist der Assembler unter den SPS-Programmiersprachen

AWL - Sprachelemente

AWL ist eine frühen Assemblern angelehnte „Low Level"-Sprache, die akkumulatororientiert arbeitet. Pro Zeile Programmcode kann nur eine Aktion (Lade in Akku, Speichere in Register) ausgeführt werden. Flußkontrolle, wie z.B. Verzweigungen, werden mit konditionierten und unkonditionierten Sprüngen und Sprungmarken. Kommentare werden nach den Steueranweisungen in der Zeile angefügt.

Beispiel:

Sprungmarke	Operator	Operand	Kommentar
	LD	TRUE	(* Lade TRUE *)
	ST	var1	(* Speichere in var1 *)
	JMPC	Label1	(* Springe bedingt*)
	LD	FALSE	(* Lade FALSE *)
	ST	var2	(*Speichere var 2*)
Label1:	LD	12	(* Lade Integer Const.*)
	ADD	var4	(* Addiere zum Akku *)
	ST	var3	(* Speichere var3*)

Aufrufe von POEs aus AWL werden in drei unterschiedlichen Notationsweisen gestattet:

- Mittels einer Aufrufliste, die die Übergabewerte enthält:
  CAL FB1(in := TRUE, mode := 4)

- Durch Laden der Übergabeparameter nacheinander, dann erfolgt der Aufruf der POE durch CAL:

```
LD      TRUE
ST      FB1.in
LD      4
ST      FB1.mode
CAL FB1                 (hier erfolgt der Aufruf)
```

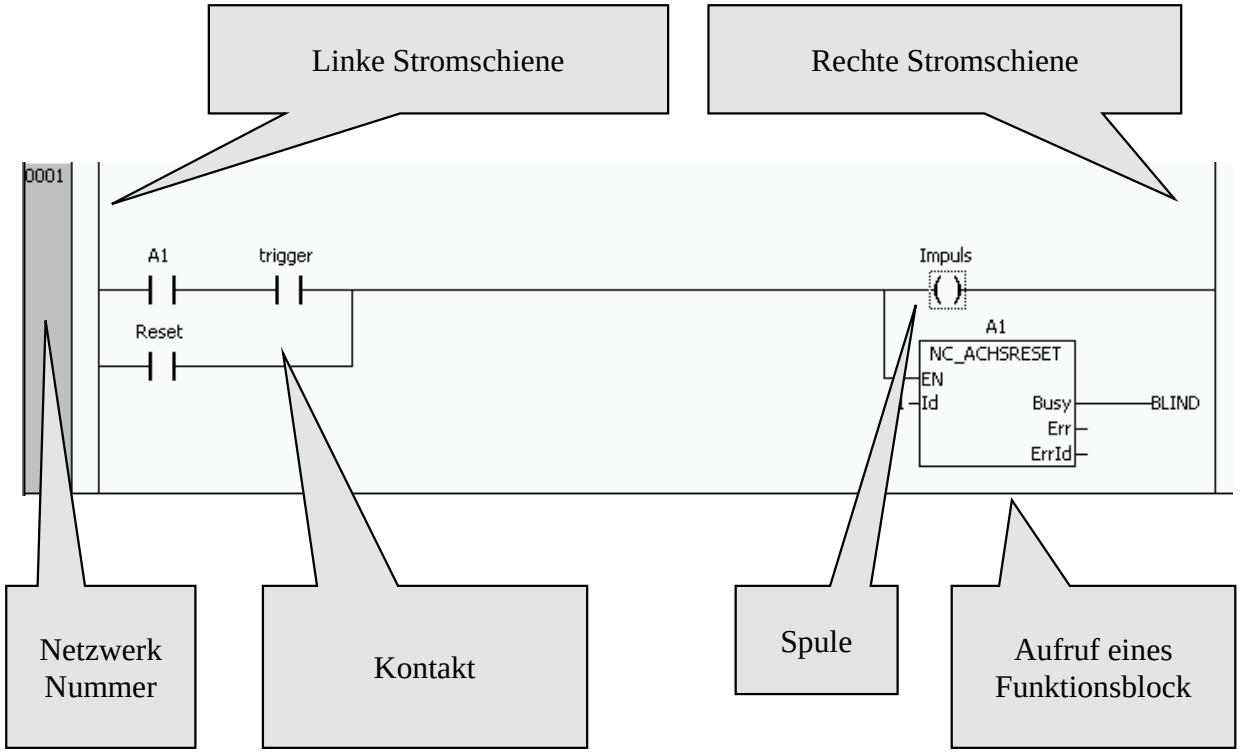
- Aufruf implizit durch Benutzung der Eingabevariablen als Operatoren: diese Methode ist nur für Standard FB erlaubt,da die Eingangsvariablen wie Operatoren bekannt sien müssen. (sei im Beispiel Zeit1 eine Instanz von TON):

```
LD      t#500ms
PT      Zeit1
LD      Output
IN      Zeit1           (hier erfolgt der Aufruf)
```

KOP

Eine Programmierung in KOP (Kontaktplan) bzw. LD (Ladder Diagramm) ist wohl der einfachste Weg einer Steuerung wohldefinierte Funktionalität zu geben. KOP hat dann seine Stärken, wenn die Steuerung als Ersatz für eine verdrahtete Logik implementiert werden soll, oder z.B. um Grundfunktionen, etwa Anlaufverriegelungen, zu programmieren. Gerade für komplexe UND / ODER Logiken ist KOP/LD sehr gut geeignet und wird z.B. in den USA wegen seiner Einfachheit bevorzugt eingesetzt. Da die Norm vorsieht, daß komplexe Bausteine (mit anderen Sprachen geschrieben) mit Hilfe eines Enable Eingangs gestartet werden können, können mit "Ladder Logic" mächtige Konstrukte angesteuert werden.

Die grafische Darstellung simuliert einen Stromfluß durch eine "linke Stromschiene" durch Eingangs- Schalter (repräsentiert durch Variablen) über Strompfade zu Ausgangsaktoren (Spulen), wiederum druch Variablen repräsentiert. Sprünge und Rücksprung an den Anfang sind erlaubt.



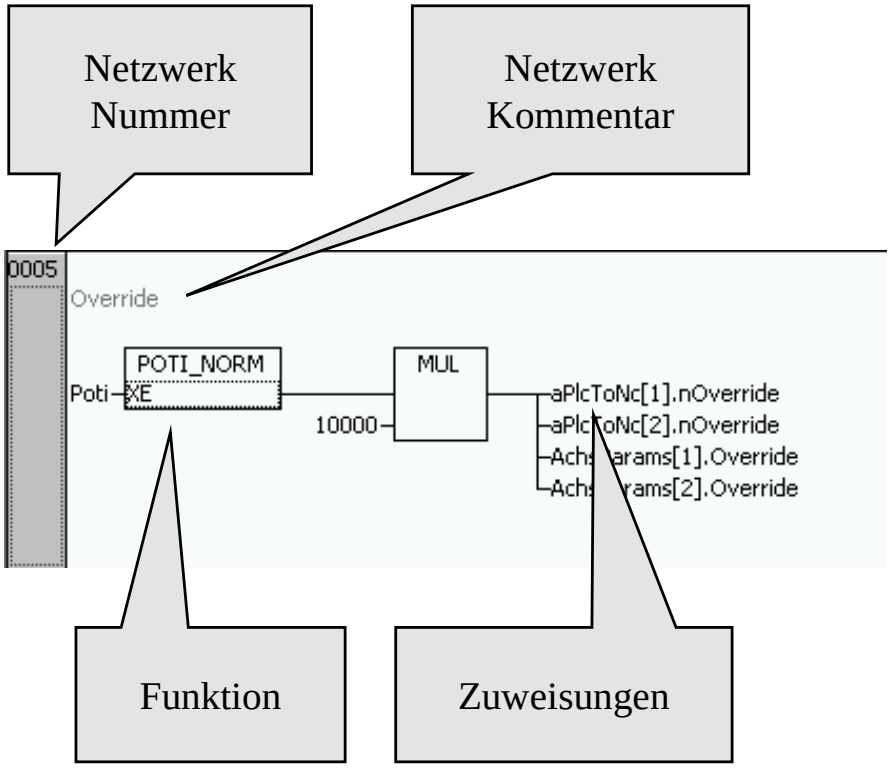
Das Ladder Diagramm (LD) bzw. KOP ist besonders in Amerika populär

FBD

Auch komplizierte Steuerungsaufgaben lassen sich mit der graphischen Programmiersprache FBS (Funktions Baustein Sprache) bzw. FBD (Function Block Diagram) realisieren. Basierend auf definierten Funktionsbausteinen lassen sich beliebige Programmabläufe mit der Hilfe von Verbindungselementen realisieren. Der Datenfluß durch das Programm kann schematisch dargestellt werden und hilft, die Programmabläufe transparent zu machen. Schwierigkeiten bereiten z.B. ein Schleifenkonstrukt oder Verzweigungen. Häufig werden auch Hardwarekomponenten mit den zugehörigen Funktionsbausteinen angeboten, so das sowohl auf Hard- und Softwareebene korrespondierende Module existieren.

Ausgänge von Funktionsblöcken werden mit Eingängen folgender Blöcke verbunden. Der Datenfluß kann vollgrafisch dargestellt werden. Sprünge und Rücksprung erleichtern die Programmierung.9

Mit der Funktions-Baustein-Sprache FBS bzw. FBD lassen sich auch komplexe Datenflüsse gut modellieren



ST

Die Programmiersprache ST (Structured Text) kann als Hochsprache, vergleichbar mit C oder Pascal angesehen werden. Der Sprachsyntax ist dem von Pascal angelehnt. Bei ST ist eine leistungsfähige Schleifenprogrammierung auch ohne Sprungbefehle möglich, darüber hinaus können mathematische Funktionen hervorragend abgebildet werden: Iterationen und konditionierte Befehle Bestandteil dieser Sprache. Die Praxiserfahrung zeigt, daß ST Konstrukte - wie von Pascal gewohnt - sehr gut lesbar und verständlich sind. SPS Programmierer in Europa wählen sehr oft diese Sprache als hauptsächlich Verwendete, wenn sie die Sprachvielfalt von IEC1131 nutzen können.

ST eignet sich bestens für Schleifen, Berechnungen und komplexe Programme, eine Farbkodierung hebt die Schlüsselworte hervor.

```
IF TRUE THEN

    CASE Mode OF
    1: (* einfaches rauf mit Wiederholung *)
            LLAWL(Len:=Len , BlinkBitsArray:= BlinkBitsArray);
    2: (* rauf und runter  *)
            LLFUP(Len:=Len , BlinkBitsArray:= BlinkBitsArray);
    3: (* auf halber Laenge 2 bit rauf und runter  *)
            LLKOP(Len:=Len , BlinkBitsArray:= BlinkBitsArray);
    4: (* auf halber Laenge alle bit rauf und runter  *)
            LLAS(LEN:=LEN , BlinkBitsArray:= BlinkBitsArray);

    END_CASE

    IF Mode <> altMode
    THEN
        FOR i := 0 TO Len DO
            BlinkBitsArray[i] := FALSE;
        END_FOR
        i := 1;
    END_IF
    altMode := Mode;

    (* copy to DWORD *)
    dwOut := 16#00000002;
    FOR count := 1 TO Len DO
```

ST - Sprachelemente

Zuweisungen und Ausdrücke werden wie folgt codiert:

```
A[i] := B;
A[i+1] := SIN(SQRT(A[i+3]));
C := timer.Q;     (* Timer ist eine Instanz von FB TOF *)
D := E/F + COS(A[i+1]);
bFlag := X AND Y OR Z;
```

Bedingte Anweisungen (If .. Then .. Else, Case) lehnen sich an Pascal an, haben jedoch eine etwas andere Syntax:

```
IF <boolean expression>
THEN
        <statements>
ELSIF <boolean expression>
THEN
        <statements>
ELSE
        <statements>
END_IF
```

```
CASE <integer expression> OF
<integer selector value1> : <statements>
<integer selector value2> : <statements>
...
<integer selector valuen> : <statements>
ELSE
        <statements>
END_CASE;
```

Iterationen (For, While, Repeat) sind ebenso wie in Pascal enthalten, sie werden jeweils mit einem Beispiel dargestellt:

```
FOR <initialise iteration variable>
        TO      <final value expression>
        BY      <increment expresssion> DO
        <statements>
END_FOR;
```

Beispiel:

```
FOR i := 1 TO 100 BY 1 DO
        a[i] := 0;
END_FOR;
```

```
WHILE <boolean expression> DO
        <statements>
END_WHILE;
```

Beispiel:

```
i := 1;
WHILE i < 100 DO
        a[i] := 0;
        i := i+1;
END_WHILE;
```

```
REPEAT
        <statements>
UNTIL <boolean expression>
END_REPEAT;
```

Beispiel:

```
i := 1;
REPEAT
        a[i] := 0;
```

```
        i := i+1;
    UNTIL i > 100
    END_REPEAT;
```

Steueranweisungen (Exit, Return) eignen sich zur Programmbeendung:

EXIT: dieser Befehl veranlaßt ST, eine Schleife zu verlassen und den nächsten Befehl nach der Schleife abzuarbeiten.

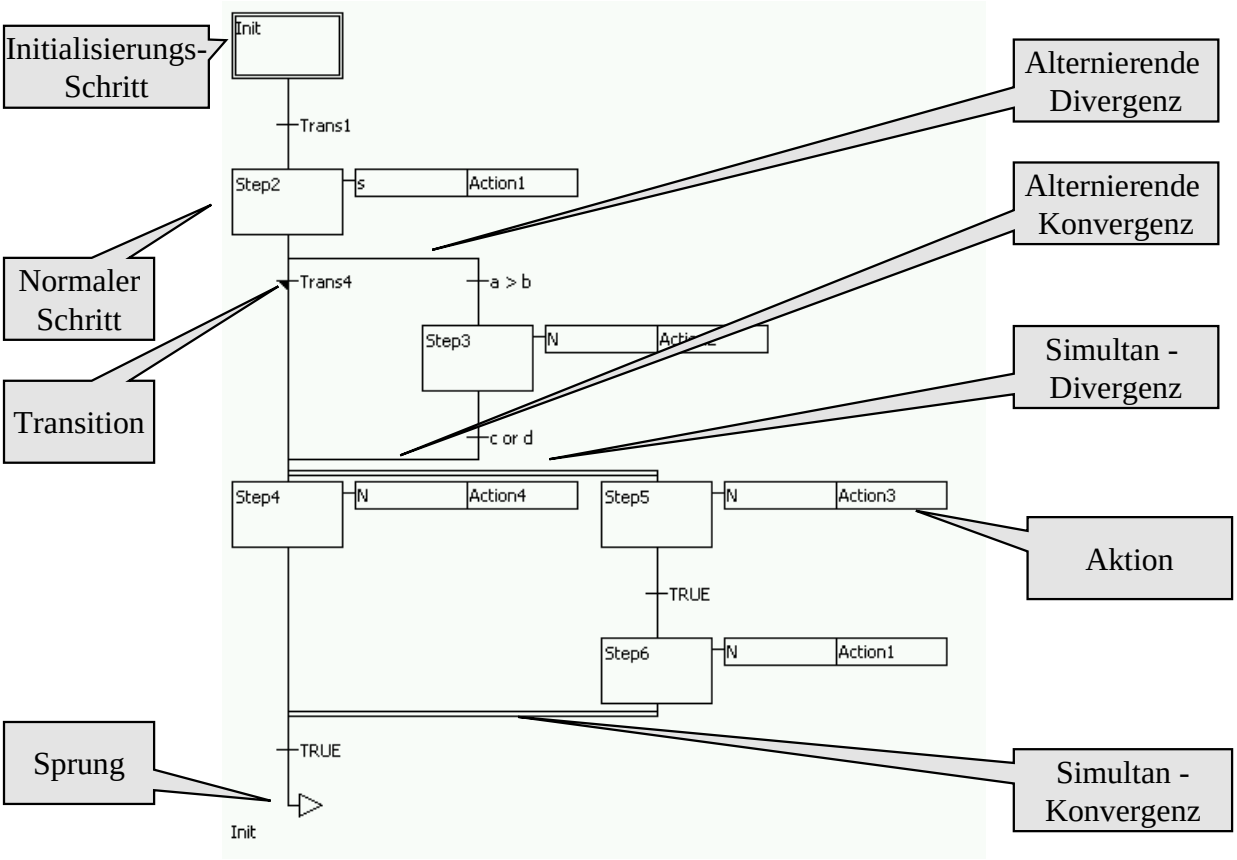
RETURN: Mit RETURN wird eine POU (FB oder Function) beendet und zum rufenden Programm zurückgekehrt: Der Befehl nach dem Ruf der POU wird bearbeitet.

AS

Die Programmiersprache AS (Ablaufsprache) oder Sequential Function Chart (SFC) ist eine grafische Sprache zur Darstellung eines Zustandsautomaten, der in einem Zyklus eine einzige Transition in einen (nächsten) Aktionszustand ausführt. Die grafische Darstellung von Transitionen und Aktionenerinnert an ein Flußdiagramm, ist sehr gut lesbar und eignet sich hervorragend für die Programmierung übergeordneter Zustandsabläufe. Oft entsteht jedoch das Mißverständnis bei Programmierern, der Programgraph werde in einem Programmzyklus komplett abgearbeitet, was nicht der Fall ist. Schnelle komplexe Berechnungen lassen sich daher nicht gut in AS / SFC, sondern nur in den dort vorgesehenen Aktionsbausteinen ausführen, die wiederum in beliebigen Sprachen gefüllt werden können.

Die Übergänge zwischen Aktionszuständen werden als Boolsche Gleichungen bei TRUE aktiv. AS / SFC erlaubt Verzweigungen in alternativer und paralleler Form:

- Aktionen in alternativen Verzweigungen werden ausgeführt, wenn die jeweilige Eintrittstransition erfüllt ist. Die Prüfungsreihenfolge (Priorität) liegt von links nach rechts fest.
- Aktionen in parallelen Verzweigungen werden alle gemeinsam gestartet. Eine Transition am Ende der parallelen Verzweigung definiert das Austrittsereignis.



AS / SFC wird der übergeordneten Ablaufprogrammierung gerecht.

AS Schritte können Aktionen aus allen Programmiersprachen besitzen - aus Gründen der Übersichtlichkeit sollte man nicht eine mehrfache Schachtelung von SFC Bausteinen eingehen. Schritte werden dargestellt als:

- Initialschritt (nur einer per AS /SFC Diagramm)
- normaler Schritt
- das Flag <Schrittname>.X zeigt den aktiven Schritt an
- das Flag <Schrittname>.T zeigt die aktive Zeit für den Schritt an und eignet sich zur Überwachung

Transitionen können programmiert werden als

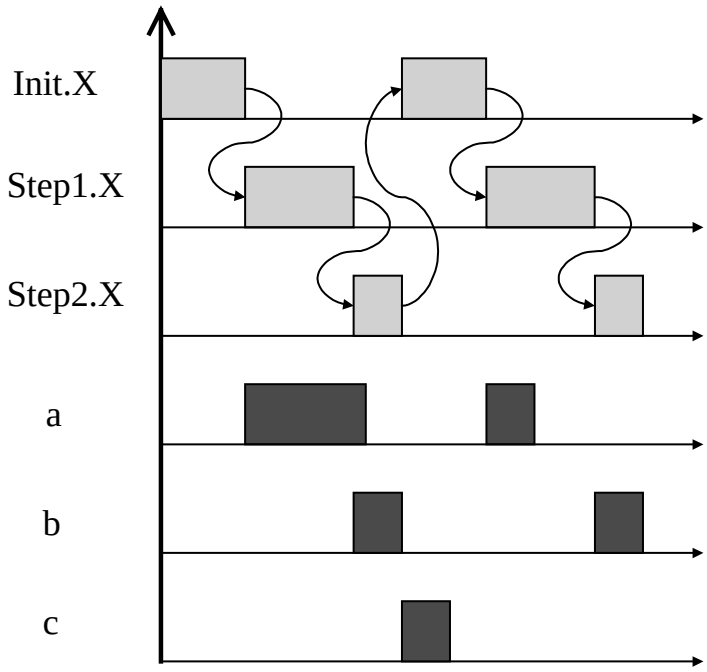
- Boolsche Variable
- Boolsche Gleichung (A>B)
- ST - Ausdruck mit boolschem Ergebnis

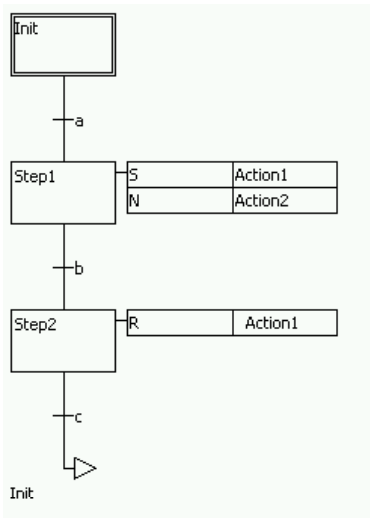
Aktionen werden mit "Qualifiern" in ihrer Ausführung gesteuert:

- N, None Nichtspeichernd (default)
- R Setzt eine Aktion zurück
- S Setzt eine (gespeicherte) Aktion
- L Aktion terminiert nach abgelaufener Zeit (Limited)
- D Aktion startet nach abgelaufener Zeit (Delayed)
- P Pulsaktion, einmal bei Schrittstart und einmal bei Schrittende

- SD Gespeichert und zeitverzögert, speichert Aktion nach Zeitperiode
- DS Aktion ist verzögert und dann gespeichert
- SL Aktion ist gespeichert und zeitbegrenzt

Das Diagramm zeigt einen einfachen Ablauf von Aktionen in gespeicherter, normaler und Reset- Betriebsart:

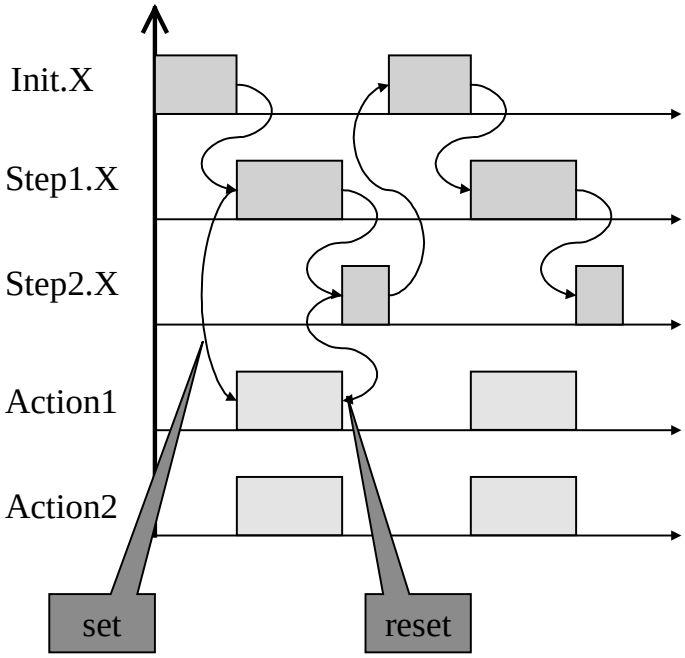


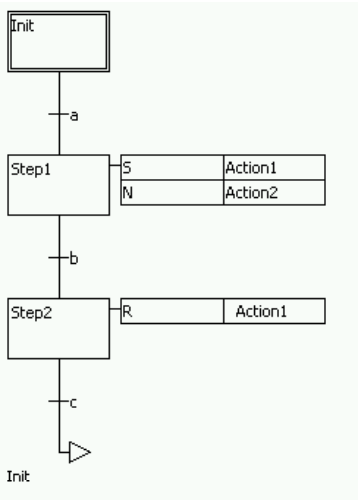


Die Transitionen a, b und c steuern den Ablauf der Schritte 1 und 2.

Die Aktionsqualifier steuern die Ausführung des Codes unabhängig vom aktiven Schritt: gespeicherte Aktionen werden erst in folgenden Schritten deaktiviert.

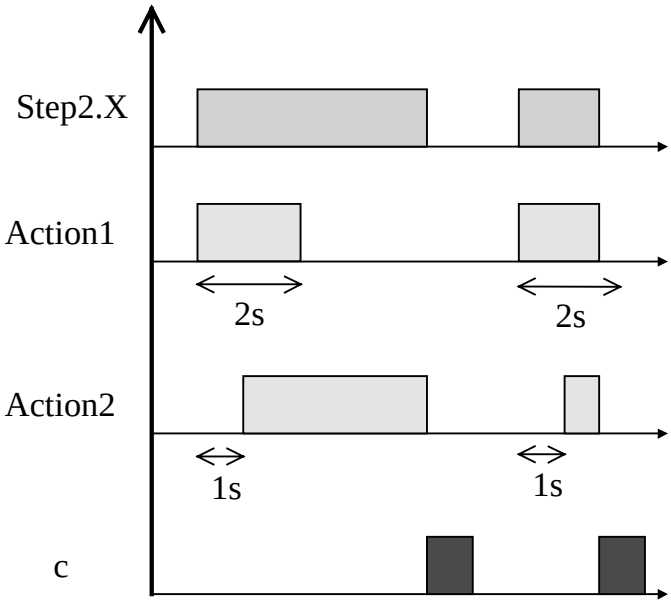
Aktions - Qualifier

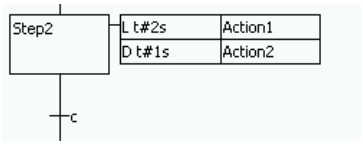




Die Aktion 1 wird nur durch Reset in Schritt 2 beendet, nicht durch Ende von Schritt 1.

Die Aktionsqualifier L und D in ihrer Arbeitsweise: begrenzte Bearbeitung durch L und verzögerter Anlauf der Bearbeitung bis zum Schrittende durch D





Die Aktionsqualifier L und D zur Zeitsteuerung von Aktionen.

Sprachkonvertierungen

Von Anwenderseite wird oft eine automatische Konvertierung zwischen Sprachen gewünscht: Editieren in ST, Fehler beseitigen in FBD. Dieser Wunsch überfordert den derzeitigen Stand der Softwaretechnik dann doch, denn für eine sinnvolle Konvertierung muß jede Sprache Rücksicht auf die Fähigkeit der anderen nehmen - und der Programmierer von vornherein mit. In der Praxis führen angebotene Konvertierungen zu unlesbaren Codeabschnitten. In der Praxis wird ein SPS - Programm aus einer Mischung von Programmiersprachen angewendet, je nach Eignung für eine

bestimmte Aufgabe.

Portierung von Projekten

Wiederverwendung von Software ist ein Mittel, um Engineeringkosten zu verkleinern. Das Ziel der Wiederverwendung von Software hat sich IEC1131 und PLCOpen mit der Definition der Sprachnorm und der Portabilitätsebenen auf die Fahnen geschrieben, der Weg dahin ist jedoch weit und beschwerlich. Spezielle Implementierungen von Funktionen, Bibliotheken oder der Datendarstellung im Speicher bewirken, daß eine große Anzahl von speziellen IEC1131 Produkten inkompatibel zueinander in Funktionsumfang und Funktionsimplementierung sind. Die Norm beschreibt zwar die Sprachdefinitionen, nicht jedoch die Implementierungen auf bestimmten Produkten.

Beispiel:

Zwei Produkte bieten Baselevel - Kompatibilität für AWL an, die Bibliotheken sind jedoch unterschiedlich umfangreich - ein einfacher Fall mit einer Reihe von Fehlermeldungen des Compilers.

Sollten die Bibliotheken identisch sein, so mag das eine Produkt in seinen Bibliotheksfunktionen eine andere Zahl von Datentypen als das andere unterstützen - wiederum wird ein Compiler Fehlermeldungen generieren.

Gelingt nun endlich der Abgleich der Softwarefunktionen, unterscheiden sich die Produkte vielleicht im Speicherraum (leicht zu prüfen) oder an oft ungenannten internen Grenzen (z.B. Anzahl der verwalteten Variablen).

Die Portierung von Projekten scheitert oft an vielen kleinen Unterschieden der IEC1131 - Produkte, mit denen Hersteller sich - oft sehr angenehm - aus der Mehrzahl der Implementierungen abheben. Derzeit ist dem schnellen Voranschreiten der Vereinheitlichungen Grenzen durch retardierende Einflüsse von Herstellern gesetzt, die sehr spezielle Implementierungen auf beschränkten Ressourcen betreiben, wie es historisch gewachsene Produktlinien, SPS - Systeme mit kleinen Prozessoren und Arbeitsspeichern erfordern. Eine Reihe von PC - Implementierungen dagegen könnte die nahezu unbeschränkte Speichergröße und Prozessorleistung zu einheitlicher Datendarstellung auf 32/64 Bit Basis und den entsprechenden 32 Bit - Adreßraum nutzen und schneller zu einer Portierbarkeit von Software gelangen. Den Einfluß und Standardisierungsdruck dieser Software - SPS - PC -Systeme zu weiteren Normungs- und "Portability" - Fortschritten wird man mit Interesse beobachten.

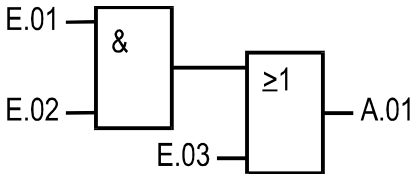
Einige Sprachformen der IEC 1131-3 im Vergleich:

Sprache **Implementation**

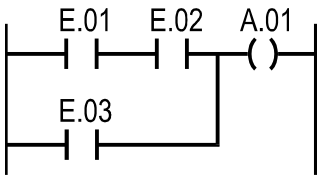
AWL

```
LD    E.01
AND   E.02
OR    E.03
ST    A.01
```

FBS



KOP



ST

```
A.01 := E.01 & E.02 OR E.03;
```

Schlußbetrachtung

Man kann festhalten, daß die IEC-Norm den Anlauf nimmt einen einheitlichen Sprachstandard im Bereich der Speicherprogrammierbaren Steuerungen zu etablieren. Die Ansätze sind durchaus verheißungsvoll, gerade bei den Software-SPSen scheint es keine Alternative zur IEC1131 zu geben. Man muß jedoch auch feststellen, daß sich die etablierten SPS-Hersteller die Freiheiten eines ”Überstandards” herausnehmen, um ihre eigenen Haus-internen Lösungen zu unterstützen. Der Wunsch der Anwender, Softwareprojekte von System zu System übertragen zu können, bleibt derzeit durch viele implementierungsspezifische Besonderheiten, wecle durchaus auchNorm-konform sein können, verwehrt. Was bleibt, ist immerhin eine recht einheitliche Bearbeitungs- und Programmierweise sowie ein einheitlicher Ansatz zur Ausbildung und Systemdiagnose: Programmierer, die einmal mit IEC1131 Systemen gearbeitet haben, können leicht auch andere IEC1131 Systeme nutzen. Das Ziel der Norm ist jedoch nicht nur eine Vereinheitlichung des Sprachumfangs. Ein wesentlicher Fokus der Norm liegt in der Übertragung der Methoden des modernen Softwareengineerings in die Maschinentechnik , um die Effizienz bei der Beherrschung großer Projekte zu steigern.