Steuerungstechnik

4. SPS-Konfiguration

V4.1

Prof. (FH) DI Dr. Franz Auinger

1

Übersicht

© Dr. Franz Auinger / FH-OÖ / Wels

2

SPS-Programmierung

4.1 Programmierung nach IEC 1131-3

Die Norm IEC 1131 wurde entwickelt, um die unterschiedlichen Sprachen, Befehlssätze und Konzepte im Bereich der Automatisierungs-systeme zu vereinheitlichen. Die Vielzahl der verfügbaren SPS-Konzepte hat in der Vergangenheit dazu geführt, daß Systeme auf unterschiedlichen SPS-Plattformen von unterschied-lichen Herstellern nicht kompatibel sind. Aus diesem Grund mußten bislang für Schulungen sowie für Hard- und Software hohe Investitionen getätigt werden.

In der IEC 1131 werden die Programmiersprachen, die Schnittstellen zwischen SPS und Programmier-system, die unterschiedlichen Anweisungen sowie die Bearbeitung und die Strukturierung von Projekten vereinheitlicht. Der Einsatz IEC 1131 konformer SPS- und Programmiersysteme bietet den Vorteil der Portabilität aller Plattformen und der Verwendung einheitlicher Konzepte, wodurch die Kosten für die Automatisierungssysteme reduziert werden.

Die Norm besteht aus mehreren Teilen und technischen Berichten. Der dritte Teil befaßt sich mit den Programmiersprachen.

Die wesentlichen Veränderungen durch die IEC 1131-3 sind:

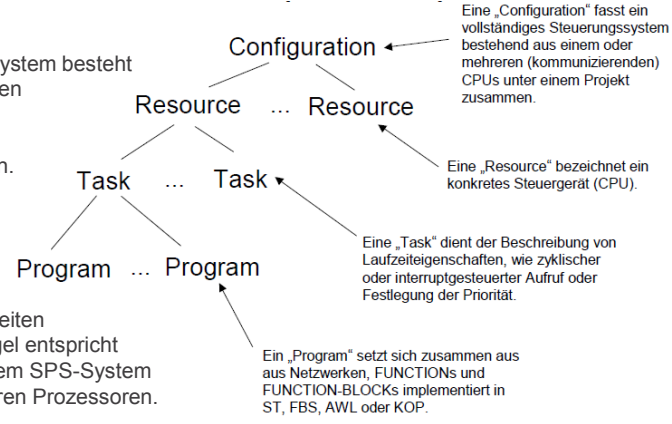
- Die Deklaration der Variablen ähnelt der Variablen-Deklaration in höheren Programmiersprachen.
- Deklaration von Datentypen ist möglich.
- Globale und lokale Daten können voneinander unterschieden werden.
- Programmieren bedeutet symbolisches Programmieren.

Grundlegende Konfigurationselemente

Nach IEC 61131-3 ist das Software-Modell einer SPS hierarchisch aufgebaut. Diese Hierarchie soll eine übersichtliche, einfach zu strukturierende und gut wartbare Automatisierungs-Software begünstigen.

Die oberste Ebene dieser Hierarchie ist die Konfiguration. Sie entspricht einem SPS-System gemäß IEC 61131-1.

Ein Automatisierungssystem besteht aus einer oder mehreren Konfigurationen, die miteinander kommunizieren können. Eine Konfiguration enthalt eine oder mehrere Ressourcen. Konfiguration und Ressourcen werden als Konfigurationseinheiten bezeichnet. In der Regel entspricht eine Konfiguration einem SPS-System mit einem oder mehreren Prozessoren.



4.2 Konfigurationselemente in der IEC 1131-3

4.2.1 Konfigurationen in der IEC 1131-3

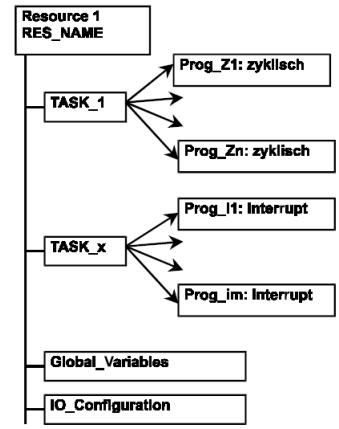
Eine Konfiguration ist vergleichbar mit einem speicherprogrammierbaren Steuerungssystem, z. B. einem Rack. In einer Konfiguration können eine oder mehrere Ressourcen definiert werden.

4.2.2 Ressourcen in der IEC 1131-3

Eine Ressource ist vergleichbar mit einer CPU, die in das Rack eingesetzt werden kann. In einer Ressource können globale Variablen deklariert werden, die nur in dieser Ressource gültig sind. Es können eine oder mehrere Tasks ausgeführt werden.

4.2.3 Tasks in der IEC 1131-3

Tasks bestimmen den Zeitplan der ihnen zugewiesenen Programme. Der Anwender muß dazu einer Task Programme zuweisen, deren zeitlicher Ablauf dann bei der Programmausführung von den Einstellungen der zugeordneten Task gesteuert wird.

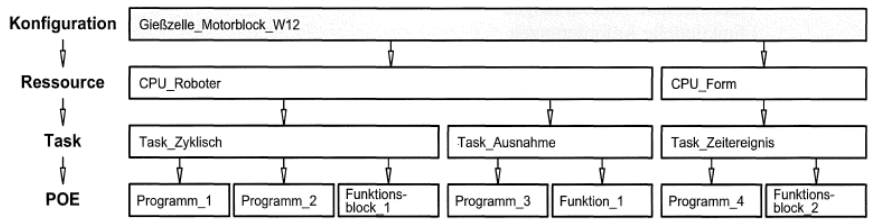


© Dr. Franz Auinger / FH-OÖ / Wels

Eine **Ressource** ist einer Konfiguration untergeordnet und umfasst im Allgemeinen eine CPU eines SPS Systems. Einer Ressource sind Tasks und Programm-Organisationseinheiten untergeordnet.

Eine **Task** ist einer Ressource untergeordnet. In einer Task werden zeitlich zusammengehörige Aufgaben zusammengefasst. Die Task beschreibt die Lauf-Eigenschaften von Programm-Organisationseinheiten. Dazu zahlen z.B. die Task-Art: Zyklische Programmabarbeitung, Programmabarbeitung infolge einer Ausnahme (Exception) oder eines Zeitereignisses (Timer). Außerdem werden in einer Task die Zykluszeit, die Zyklustoleranz und die Prioritat der untergeordneten Programm-Organisationseinheiten festgelegt.

Programm-Organisationseinheiten (POE) sind Programme, Funktionsblocke oder Funktionen. Die Lauf-Eigenschaften erhalten die POEs von der übergeordneten Task.



© Dr. Franz Auinger / FH-OÖ / Wels

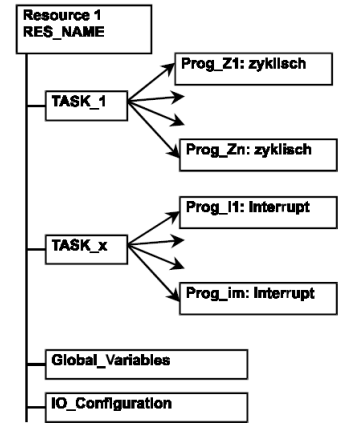
4.2 Konfigurationselemente in der IEC 1131-3

4.2.3 Tasks in der IEC 1131-3

Die IEC 1131-3 definiert verschiedene Arten von Zeitverhalten. Daher wird zwischen drei verschiedenen Task-Typen unterschieden:

- Zyklische Tasks werden innerhalb von bestimmten Zeitintervallen aktiviert und das Programm wird periodisch ausgeführt.
- Fehlertasks werden aktiviert, wenn in einer anderen Task ein Fehler auftritt.
- Ereignis-Tasks werden aktiviert, wenn ein bestimmtes Ereignis eingetreten ist.

Jede Task besitzt eine festgelegte Priorität. In sogenannten Systemen mit vorberechtigtem Aufruf wird eine aktive Task mit niedrigerer Priorität sofort unterbrochen wenn eine höherpriore Task durch ein bestimmtes Ereignis aktiviert wird. In Systemen mit nicht-vorberechtigtem Aufruf können Tasks nicht durch höherpriore Tasks unterbrochen werden.



Hinweis: Welche Task-Typen unterstützt werden, hängt von der eingesetzten SPS ab.

4.3 Bausteine (POEs) in der IEC 1131-3

In der IEC1131-3 werden die Bausteine, aus denen ein Projekt aufgebaut wird, als Programm-Organsationseinheiten (POE) bezeichnet.

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

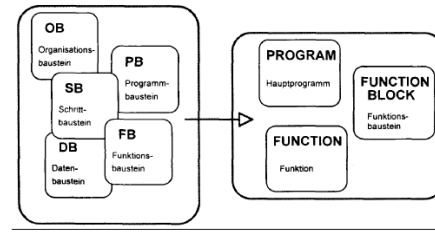
Tabelle 4-1: POE-Typen

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

- Funktion (FUN oder FC). 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.

4.3 Bausteine (POEs) in der IEC 1131-3

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 1131-3 mit Eingangs- und Ausgangsparametern in etwa einem bisher üblichen Funktionsbaustein. Die POE-Typen PROGRAM und FUNCTION besitzen aufgrund ihrer gegenüber FB's erweiterten bzw. eingeschränkten Eigenschaften keine direkten Pendants zu Bausteinen nach DIN 19239.

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

Die IEC 1131-3 benutzt Variablen, um Anwender-Daten zu initialisieren, weiterzuverarbeiten und zwischenzuspeichern. Jede POE besteht daher aus zwei unterschiedlichen Teilen: Dem Deklarationsteil und dem Code-Teil.

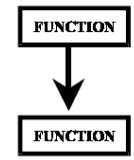
Im Deklarationsteil werden alle notwendigen Variablen deklariert.

Der Anweisungsteil oder Code-Teil einer POE ist der Teil, in dem die Anweisungen in der gewünschten Sprache programmiert werden.

© Dr. Franz Auinger / FH-OÖ / Wels

4.3.1 Funktionen in der IEC 1131-3

Funktionen sind POEs mit mehreren Eingangsparametern und genau einem Ausgangsparameter. Das Aufrufen einer Funktion mit den gleichen Eingangsparametern liefert immer das gleiche Ergebnis. Rückgabewerte können einfache Datentypen sein. In einer Funktion können andere Funktion aufgerufen werden, jedoch keine Funktionsbausteine oder Programme. Rekursive Aufrufe sind nicht zulässig. Die Abkürzung für Funktionen ist FU.



Die IEC 1131-3 definiert Standardfunktionen, die Sie in Ihrem SPS-Programm verwenden können. Je nach verwendetem Hardware- und SPS-Typ sind unter Umständen nicht alle Standardfunktionen verfügbar. Es ist auch möglich, daß zusätzliche Firmware-Funktionen verfügbar sind.

Hinweis: Detaillierte Informationen zu Funktionen und Funktionsbausteinen erhalten Sie in der Dokumentation zu Ihrer Hardware.

© Dr. Franz Auinger / FH-OÖ / Wels

4.3.1 Funktionen in der IEC 1131-3

In der IEC 1131-3 werden folgende Standardfunktionen definiert:

Funktion	Beschreibung	Funktion	Beschreibung
*_TO_**	Typumwandlung von Datentyp * in Datentyp **	MAX	Maximum
ABS	Absolutwert	MID	Mitte einer Zeichenfolge
ACOS	Arccos, Hauptwert	MIN	Minimum
ADD	Addition	MOD	Modulo-Division
ADD_T_T	Addition für Zeitgeberwerte	MOVE	Zuweisung
AND	UND-Verbindung	MUL	Multiplikation
ASIN	Arcsin, Hauptwert	MUL_T_AN	Multiplikation für Zeitgeberwerte
ATAN	Arctan, Hauptwert	MUX	noch nicht implementiert
CONCAT	Erweiterbare Aneinanderreihung	NE	Vergleich auf Ungleichheit
COS	Cosinus mit Eingang im Bogenmaß	NEG	Doppelkomplement
DELETE	Abschnitt löschen	NOT	Komplement
DIV	Divisor	OR	ODER-Verbindung
DIV_T_AN	Divisor für Zeitgeberwerte	REPLACE	Abschnitt ersetzen

© Dr. Franz Auinger / FH-OÖ / Wels

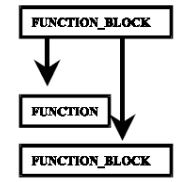
4.3.1 Funktionen in der IEC 1131-3

EQ	Gleichheit	RIGHT	Rechtsstehende Zeichen
EXP	Exponentialfunktion	ROL	Links rotieren
EXPT	Potenzierung	ROR	Rechts rotieren
FIND	Zeichenposition ermitteln	SEL	Binäre Auswahl
GE	Vergleich: Größer / Gleich	SHL	Links schieben
GT	Vergleich: Größer	SHR	Rechts schieben
INSERT	Zeichenfolge einfügen	SIN	Sinus mit Eingang im Bogenmaß
LE	Vergleich: Kleiner / Gleich	SQRT	Quadratwurzel
NACH-LINKS	Linksstehende Zeichen	SUB	Subtraktion
LEN	Länge einer Zeichenfolge	SUB_T_T	Subtraktion für Zeitgeberwerte
LIMIT	Begrenzung	TAN	Tangens mit Eingang im Bogenmaß
LN	Natürlicher Logarithmus	TRUNC	Abschneiden
LOG	Logarithmus zur Basis 10	XOR	EXODER-Verbindung
LT	Vergleich: Kleiner		

© Dr. Franz Auinger / FH-OÖ / Wels

4.3.2 Funktionsbausteine in der IEC 1131-3

Funktionsbausteine sind POEs mit mehreren Eingangs- und Ausgangsparametern und internem Speicher. Der Wert, den ein Funktionsbaustein als Ergebnis zurückgibt, hängt vom aktuellen Wert seines internen Speichers ab. In einem Funktionsbaustein können weitere Funktionsbausteine oder andere Funktionen aufgerufen werden. Dies ist aber nicht möglich für Programme. Rekursive Aufrufe sind nicht zulässig. Die Abkürzung für Funktionsbausteine ist FB.



Die IEC 1131-3 definiert Standardfunktionsbausteine, die Sie in einem SPS-Programm verwenden können. Je nach eingesetzter Hardware und SPS stehen Ihnen eventuell nicht alle Standardfunktionsbausteine zur Verfügung. Es können aber auch zusätzliche Firmware-Funktionsbausteine verfügbar sein.

Hinweis: Detaillierte Informationen zu Funktionen und Funktionsbausteinen erhalten Sie in der Dokumentation zu Ihrer Hardware.

SPS-Programmierung

4.3.2 Funktionsbausteine in der IEC 1131-3

In der IEC 1131-3 werden folgende Standardfunktionsbausteine beschrieben:

FB	Beschreibung
SR	Vorrangig Setzen
RS	Vorrangig Rücksetzen
R_TRIG	Erkennung der steigenden Flanke
F_TRIG	Erkennung der fallenden Flanke
CTU	Aufwärts-Zähler
CTD	Abwärts-Zähler
CTUD	Auf-Abwärts-Zähler
TON	Zeitgeber für Einschalt-Verzögerung
TOF	Zeitgeber für Ausschalt-Verzögerung
TP	Puls

4. SPS-Konfiguration

4.3.3 POE-Typ „Programm

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_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.

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

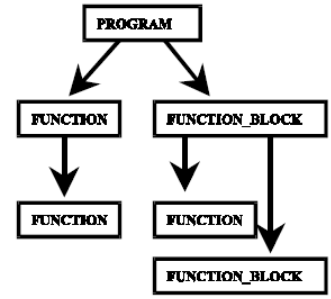
Laufzeit-Eigenschaften und -Sonderbehandlung eines PROGRAM in einer SPS-CPU kommen in der Zuordnung des PROGRAM zu TASKs zum Ausdruck. Dabei wird das Programm instanziiert, kann also mehreren Tasks zugeordnet und dadurch in der SPS mehrfach glechzeitig ausgeführt werden.

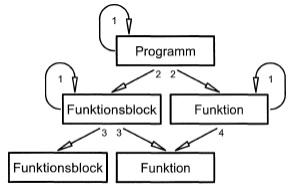
© Dr. Franz Auinger / FH-OÖ / Wels

4.3.3 POE-Typ „Programm

Programme sind POEs, die eine logische Kombination von Funktionen und Funktionsbausteinen enthalten, entsprechend den Erfordernissen des Steuerungsprozesses. Das Verhalten und die Verwendung von Programmen ist ähnlich wie bei Funktionsbausteinen. Programme können Eingangs- und Ausgangsparameter sowie einen internen Speicher haben. Programme müssen Tasks zugewiesen werden.

In einem Programm können Funktionen und Funktionsbausteine aufgerufen werden. Rekursive Aufrufe sind nicht zulässig.



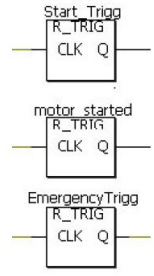


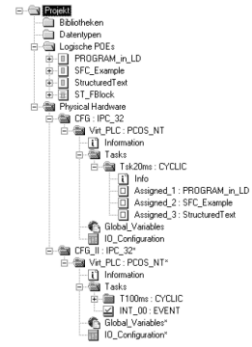
Aufruf von POEs: POEs konnen sich gegenseitig aufrufen. Dabei sind die folgenden Bedingungen zu beachten:
1. Die rekursive Programmierung (POE ruft sich selbst auf) ist nicht erlaubt.
2. Ein Programm darf einen Funktionsblock und eine Funktion aufrufen, jedoch nicht umgekehrt.
3. Ein Funktionsblock darf einen Funktionsblock oder eine Funktion aufrufen.
4. Eine Funktion darf nur Funktionen aufrufen, nicht jedoch Funktionsblocke oder Programme (dadurch kamen Funktionen ja wieder zu einem Gedächtnis).

© Dr. Franz Auinger / FH-OÖ / Wels

4.3.4 Instanziierung

Damit bereits definierte Funktionsbausteine mehrfach verwendet werden können, bietet die IEC 1131-3 die Möglichkeit der Instanziierung. Das bedeutet, daß ein Funktionsbaustein oder ein Programm einmal definiert wird und dann sein interner Speicher verschiedenen Instanzen, also verschiedenen Speicherbereichen, zugewiesen wird. Jede Instanz hat einen Bezeichner (den Instanznamen) und enthält die Eingangs- und Ausgangsparameter sowie den internen Speicher des instanziierten Funktionsbausteins oder Programms. Ein Funktionsbaustein kann in anderen Funktionsbausteinen oder Programmen instanziiert werden. Der entsprechende Instanzname eines Funktionsbausteins muß in der VAR-Deklaration des Programms oder Funktionsbausteins deklariert werden. Programme können nur innerhalb derselben Ressource instanziiert werden.





4.4 Datentypen in der IEC 1131-3

Datentypen legen die Eigenschaften für die Werte einer Variablen fest. Sie definieren den Anfangswert, den Bereich der möglichen Werte und die Anzahl der Bits.

Die IEC 1131-3 unterscheidet drei Arten von Datentypen:

- Elementare Datentypen
- Generische Datentypen
- Anwenderdefinierte Datentypen

4.4.1 Elementare Datentypen

Die Wertebereiche und die Bitgröße elementarer Datentypen sind in der IEC 1131-3 definiert.

Die elementaren Datentypen sind in der folgenden Tabelle dargestellt:

Datentyp	Beschreibung	Bits	Bereich	Standard-Anfangswert
BOOL	Boolesch	1	0...1	0
BYTE	Bitfolge der Länge 8	8	00 bis FF	0
WORD	Bitfolge der Länge 16	16	0000 bis FFFF	0
DWORD	Bitfolge der Länge 32	32	00000000 bis FFFFFFFF	0
LWORD	Bitfolge der Länge 64	64	0 bis FFFFFFFF FFFFFFFF	0
SINT	kurze Ganzzahl	8	-128...127	0
INT	Ganzzahl	16	-32768...32767	0
DINT	doppelte Ganzzahl	32	-2.147.483.648 bis 2.147.483.647	0

LINT	lange Ganzzahl	64	-2^{63} bis $+2^{63}-1$	0
USINT	kurze Ganzzahl ohne Vorzeichen	8	0 bis 255	0
UINT	Ganzzahl ohne Vorzeichen	16	0 bis 65535	0
UDINT	doppelte Ganzzahl ohne Vorzeichen	32	0 bis 4.294.967.295	0
ULINT	lange Ganzzahl ohne Vorzeichen	64	0 bis $+2^{64}-1$	0
REAL	Gleitkomma-Zahl	32	$\pm 2{,}9 \times 10^{-39}$ bis $\pm 1{,}7 \times 10^{38}$	0.0
LREAL	Lange Gleitkomma-Zahl	32	$\pm 5{,}0 \times 10^{-324}$ bis $\pm 1{,}7 \times 10^{308}$	0.0
DATE	Datum			d#0001-01-01
TOD	Uhrzeit (TIME_OF_DAY)			tod#00:00:00
DT	Datum mit Uhrzeit (DATE_AND_TIME)			dt#0001-01-01-00:00:00
TIME	Zeitdauer			t#0s
STRING	Zeichenfolge			" (leere Zeichenfolge)

Der Datentyp STRING ist zwar ein elementarer Datentyp, gehört aber nicht zu der oben angegebenen Gruppe. Sein Format im Speicher hängt vom SPS-Typ ab.

Der Datentyp STRING hat folgende Struktur:

Byte 0-1 Offset zur maximalen Länge (0 entspricht 80)

Byte 2-3 aktuelle Länge

Byte 4-83 Zeichen

Byte 84 Null-Terminator

LINT	lange Ganzzahl	64	-2^{63} bis $+ 2^{63} - 1$	0
USINT	kurze Ganzzahl ohne Vorzeichen	8	0 bis 255	0
UINT	Ganzzahl ohne Vorzeichen	16	0 bis 65535	0
UDINT	doppelte Ganzzahl ohne Vorzeichen	32	0 bis 4.294.967.295	0
ULINT	lange Ganzzahl ohne Vorzeichen	64	0 bis $+ 2^{64} - 1$	0
REAL	Gleitkomma-Zahl	32	$\pm$ 2,9 x 10^{-39} bis $\pm$ 1,7 x 10^{38}	0.0
LREAL	Lange Gleitkomma-Zahl	32	$\pm$ 5,0 x 10^{-324} bis $\pm$ 1,7 x 10^{308}	0.0
DATE	Datum			d#0001-01-01
TOD	Uhrzeit (TIME_OF_DAY)			tod#00:00:00
DT	Datum mit Uhrzeit (DATE_AND_TIME)			dt#0001-01-01-00:00:00
TIME	Zeitdauer			t#0s
STRING	Zeichenfolge			" (leere Zeichenfolge)

Der Datentyp STRING ist zwar ein elementarer Datentyp, gehört aber nicht zu der oben angegebenen Gruppe. Sein Format im Speicher hängt vom SPS-Typ ab.

Der Datentyp STRING hat folgende Struktur:

Byte 0-1 Offset zur maximalen Länge (0 entspricht 80)

Byte 2-3 aktuelle Länge

Byte 4-83 Zeichen

Byte 84 Null-Terminator

In der Variablendeklaration können den Variablen Anfangswerte übergeben werden, die die Steuerung bei Programmstart übernimmt und entsprechend den Verknüpfungen aktualisiert.

Achtung: Wird z.B. einer Eingangsvariablen mit direkter Adressierung (%IQ...) ein Anfangswert übergeben, so wird dieser Eingangswert beim Lesen der Eingänge – falls der Eingang physikalisch vorhanden ist – überschrieben.

SPS-Programmierung

4.4.2 Generische Datentypen

Generische Datentypen sind Datentypen, in denen Gruppen elementarer Datentypen zusammengefaßt sind. Beispielsweise beinhaltet ANY_INT die elementaren Datentypen DINT, INT, SINT, UDINT, UINT und USINT. Wenn eine Funktion mit ANY_INT verbunden werden kann, bedeutet das, daß sie Variablen der Datentypen DINT, INT, SINT, UDINT, UINT und USINT verarbeiten kann.

Die folgende Liste zeigt die generischen Datentypen:

ANY		
ANY_NUM	ANY_REAL	REAL
	ANY_INT	SINT, INT, DINT, LINT, USINT,UINT, UDINT, ULINT
ANY_BIT	DWORD, WORD, BYTE, BOOL	
ANY_DATE	DATE, TIME_OF_DAY, DATE_AND_TIME	
STRING (abgeleitet)		
TIME (abgeleitet)		

4. SPS-Konfiguration

Hierarchische Entwicklung der elementaren Datentypen aus den allgemeinen Datentypen

Allgemeiner Datentyp				elementarer Datentyp	Bits	Bereich	Initialwert	Beschreibung
ANY	ANY_NUM	ANY_REAL		REAL	32	nach IEEE 754	0.0	reelle Zahl
				LREAL	64		0.0	lange reelle Zahl
		ANY_INT	UNSIGNED *1	USINT	8	$0 .. 2^8 -1$	0	vorzeichenlose kurze ganze Zahl
				UINT	16	$0 .. 2^{16} -1$	0	vorzeichenlose ganze Zahl
				UDINT	32	$0 .. 2^{32} -1$	0	vorzeichenlose doppelte ganze Zahl
				ULINT	64	$0 .. 2^{64} -1$	0	vorzeichenlose lange ganze Zahl
			SIGNED *1	SINT	8	$-2^7 .. 2^7 -1$	0	kurze ganze Zahl
				INT	16	$-2^{15} .. 2^{15} -1$	0	ganze Zahl
				DINT	32	$-2^{31} .. 2^{31} -1$	0	doppelte ganze Zahl
				LINT	64	$-2^{63} .. 2^{63} -1$	0	lange ganze Zahl
	ANY_BIT			BOOL	1	0, 1	0	boolsche Zahl
				BYTE	8	kein numerischer Wertebereich	alle Bits 0	kurze Bit-Folge
				WORD	16		alle Bits 0	Bit-Folge
				DWORD	32		alle Bits 0	doppelte Bit-Folge
				LWORD	64		alle Bits 0	lange Bit-Folge
	ANY_DATE			DATE_AND_TIME	impl. abhängig	implementations-abhängig	DT1-1-1-0:0:0	Datum und Uhrzeit
				DATE			D#0001-01-01	Datum
				TIME_OF_DAY			TOD#00:00:00	Uhrzeit
				TIME			T#0s	Zeitdauer
				STRING		kein numerischer Wertebereich	Leerstring	Zeichenfolge

*1 : keine Begriffe der Norm IEC 61131

Beschreibung einiger Datentypen für B&R AS V3.0

Variablentyp	Byte	Bereich	Bemerkung
REAL	4	$-3{,}4 \cdot 10^{38} .. +3{,}4 \cdot 10^{38}$	REAL entspricht dem IEEE 754 Standard Fließkommaformat mit drei Byte-Mantisse und einem Byte Exponent.
DATE_AND_TIME	4	01.01.1970 - 31.12.2099	Die seit 1.1.1970 00:00 vergangenen Sekunden werden in einer 32 Bit-Zahl abgelegt.
TIME	4	ca. ±24 Tage	1 ms Auflösung
STRING	80	-	80 Zeichen

4.4.3 Anwenderdefinierte Datentypen

Anwenderdefinierte Datentypen müssen mit dem Schlüsselwort TYPE ... END_TYPE in einem Datentyp-Arbeitsblatt deklariert werden. Anwenderdefinierte Datentypen (sie werden auch als abgeleitete Datentypen bezeichnet) können Aufzählungen, Strukturen oder Felder sein.

Der Datentyp Feld schließt mehrere Elemente eines Datentyps ein. Ein Feld kann verwendet werden, um mehrere Elemente des gleichen Typs in einer einzigen Zeile der Typendeklaration zu deklarieren.

Beispiel: Ganzzahlen für x-Positionen, x-y-Positionen, x-y-z-Positionen,

```
TYPE
    Pos_1Dim  :  ARRAY [1..99]      OF  INT;   (* 99 Meßpunkte, 1-dimensional *)
    Pos_2Dim  :  ARRAY [1..99,1..2] OF  INT;   (* 99 Meßpunkte, 2-dimensional *)
    Pos_3Dim  :  ARRAY [1..99,1..3] OF  INT;   (* 99 Meßpunkte, 3-dimensional *)
END_TYPE
```

Der Datentyp Struktur enthält mehrere Elemente unterschiedlicher Datentypen.

Beispiel: Steuerung eines Motors

```
TYPE
    Motor:                                     (* Variablenname für STRUCT *)
    Drehzahl_Soll  :  INT;
    Drehzahl_Ist   :  INT;
    Freigabe       :  BOOL;
    Start          :  BOOL;
    Stop           :  BOOL;
    Stoerung       :  BOOL;
    Motor_EIN      :  BOOL;
END_TYPE
```


Der Datentyp Aufzählung begrenzt den Bereich der möglichen Werte auf die jeweils aufgezählten. Eine Variable dieses Datentyps kann jeweils nur einen der aufgezählten Werte annehmen.

Beispiel: Farberkennung für rot, grün, blau

```
TYPE
    Farbe      :  (rot, gruen, blau);    (* Aufzählungen können als Text-Kontsante *)
END_TYPE                                 (* verstanden werden *)
```

Mit dem abgeleiteten Datentyp Bereichsangabe kann für eine Variable der Wertebereich eingeschränkt werden. Damit kann sowohl bei der Programmierung als auch zur Laufzeit eine Fehlermeldung ausgegeben bzw. ausgewertet werden.

Beispiel: Temperaturerfassung mit Fühler

```
TYPE
    Temp_Sensor  :  INT (-25..120);  (* gültiger Bereich – 25 bis 120 °C *)
END_TYPE
```

Hinweis: Die Verwendung von Datentypen oder eventuelle Einschränkungen hängen auch von der Hardware ab (Dokumentation beachten).

Aufzählung

```
TYPE
   werkstueck (feder,bolzen,gehaeuse);
   (* Konstanten: feder=1,bolzen=2,gehaeuse=3 *)
END_TYPE
VAR
   teil : werkstueck := feder;
   (* Zuweisung eines Anfangswertes ist optional *)
END_VAR
(* teil:=4 ist eine Fehler! *)
```

Bereich

```
TYPE
   tankfuellung : INT(0...58);
   (* eingeschraenkter Wertebereich *)
END_TYPE
```

Eine Verletzung des Bereiches gibt einen Laufzeitfehler, der mit der IEC Check Library abgefangen werden kann (Funktionen CheckSignedSubrange und CheckUnsigned Subrange).

Typenkonversion

Variablen, Konstanten und Funktionen haben einen bestimmten Typ. Bei gemischter Verwendung ist eine explizite Typkonversion notwendig.

```
VAR i : INT;
      r : REAL;
END_VAR

r:=INT_TO_REAL(i);
i:=TRUNC(r);
(* INT_TO_REAL() und TRUNC() sind Standardfunktionen *)
```


Beispiele: Zusammengesetzte Datentypen

Feld

```
TYPE
   messwert : ARRAY[1..50] of REAL;
END_TYPE
VAR
   druckverlauf : messwert;
   druck : REAL;
END_VAR
druck:=druckverlauf[15];
```

Struktur (Datenrecord)

```
TYPE
   ventil : STRUCT
      vorhanden : BOOL;
      ausschuss : BOOL;
      zylinder : BOOL;
      farbe : INT;
   END_STRUCT
END_TYPE
VAR
   teile_liste : ARRAY[1..100] of ventil;
END_VAR
teile_liste[7].vorhanden:=TRUE;
```

4.5 Variablen in der IEC 1131-3

In der IEC 1131-3 werden drei Variablentypen beschrieben:

- Symbolische Variablen
- Direkt dargestellte Variablen
- Adressierte Variablen

4.5.1 Schlüsselwörter zur Variablen-Deklaration

Variablen müssen unter Verwendung von Schlüsselwörtern im Variablen-Arbeitsblatt der POE deklariert werden, in der sie verwendet werden.

Zur Variablen-Deklaration müssen entsprechende Schlüsselwörter verwendet werden. Die Schlüsselwörter sind in der folgenden Tabelle dargestellt::

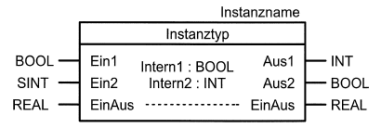
Schlüsselwort	Beschreibung
VAR	für interne Variablen, die nur innerhalb der POE verwendet werden können zur Deklaration der Instanzen von Funktionsbausteinen zur Deklaration von direkt dargestellten, adressierten und symbolischen Variablen zusammen mit dem Schlüsselwort ' RETAIN' zur Deklaration permanenter Variablen
VAR_INPUT	Eingangsvariable von Funktionen, Funktionsbausteinen und Programmen kann nur zur Deklaration von symbolischen Variablen verwendet werden
VAR_OUTPUT	Ausgangsvariable von Funktionsbausteinen und Programmen zusammen mit dem Schlüsselwort ' RETAIN' zur Deklaration permanenter Variablen
VAR_IN_OUT	Adresse der Variablen wird übergeben (Referenzparameter) die Variable kann gelesen oder geschrieben werden typischerweise für komplexe Datentypen wie Folgen, Felder und Strukturen verwendet
VAR_EXTERNAL	für globale Variablen in der POE der Wert wird durch die Deklaration von VAR_GLOBAL geliefert der Wert kann innerhalb der POE geändert werden kann nur zur Deklaration von symbolischen Variablen verwendet werden
VAR_GLOBAL	für globale Variablen, die in allen Programmen und Funktionsbausteinen des Projekts verwendet werden können zur Deklaration von direkt dargestellten, adressierten und symbolischen Variablen zusammen mit dem Schlüsselwort ' RETAIN' zur Deklaration permanenter Variablen
END_VAR	um einen Variablen-Deklarationsblock zu beenden

Variablenarten in POE-Typen

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

Beispiel

Beispiel Funktionsblock



Deklaration:

```
VAR_INPUT
  Ein1     : BOOL;
  Ein2     : SINT;
END_VAR

VAR_OUTPUT
  Aus1     : INT;
  Aus2     : BOOL;
END_VAR

VAR_IN_OUT
  EinAus   : REAL;
END_VAR

VAR
  Intern1  : BOOL;
  Intern2  : INT;
END_VAR
```

Hinweis: Die internen Variablen (Deklaration VAR) werden im Funktionsblock nicht angezeigt.

Kommunikation zwischen Programmen über globale Variablen

In POEs können lokale und globale Variablen deklariert werden.

Lokale Variablen sind nur in der POE sichtbar, in der sie deklariert wurden. Dadurch können in unterschiedlichen POEs physikalisch unterschiedliche Variablen mit dem gleichen Bezeichner deklariert werden.

Globale Variablen werden in einer POE deklariert, sind aber im ganzen Projekt sichtbar. Die Kommunikation (z.B. Datenübergabe, Synchronisation) zwischen Programmen ist so einfach über das Lesen und Schreiben von globalen Variablen zu realisieren.

Besonderheit: Bei B&R können auch Paketlokale Variablen deklariert werden. Die CPU unterstützt diese jedoch (noch) nicht direkt und so werden diese als global behandelt.

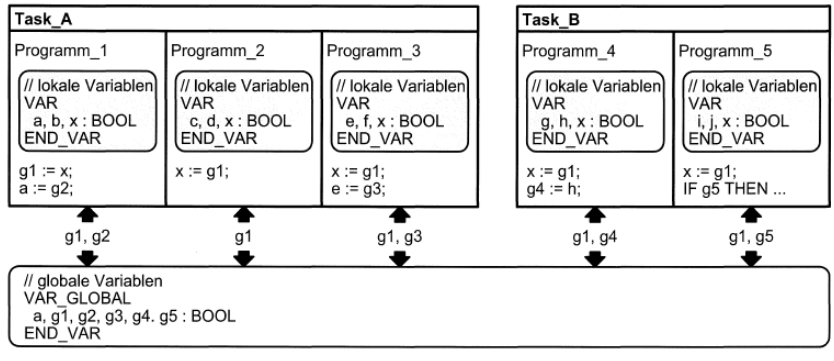
© Dr. Franz Auinger / FH-OÖ / Wels

Kommunikation zwischen Programmen über globale Variablen

Im Bild ist eine Konfiguration mit zwei Tasks, fünf Programmen und globalen sowie lokalen Variablen zu sehen. Beachten Sie, dass jedes Programm eine lokale Variable x besitzt. Dabei handelt es sich um fünf unterschiedliche Variablen mit unterschiedlichen Adressen und dem gleichen Bezeichner *x.*

Beachten Sie auch, dass im *Programm 1* eine lokale Variable mit dem Bezeichner *a* deklariert ist und im *Programm 5* eine globale Variable mit dem Bezeichner *a* deklariert ist.

Beim Zugriff von *Programm 1* auf die Variable *a* wird immer die lokale Variable *a* von *Programm 1* gelesen oder geschrieben. Eine Kommunikation von *Programm 1* mit *Programm 5* über die Variable *a* ist somit nicht möglich..



© Dr. Franz Auinger / FH-OÖ / Wels

Beisp. in der ST zur Kommunikation und Synchronisation der Programme

1.) **Datenkommunikation**

Der Inhalt der lokalen Variablen *x* von *Programm_1* wird an alle anderen lokalen Variablen *x* der Programme *Programm_2* .. *Programm_5* geschrieben. Dazu wird die globale Variable *g1* benutzt:

Programm_1	Programm_2	Programm_3	Programm_4	Programm_5
`g1 := x;`	`x := g1;`	`x := g1;`	`x := g1;`	`x := g1;`

2.) **Synchronisation von Programmaktivitäten**

Programm_1 startet durch Setzen der Variablen *g2* in den Programmen *Programm_2* .. *Programm_5* Aktivitäten.

Programm_1	Programm_2	Programm_3	Programm_4	Programm_5

`IF g1 THEN`
`  g2 := TRUE;`
`END_IF`
	`IF g2 THEN`
	`  g3 := TRUE;`
	`  // weitere`
	`  // Aktivit.`
	`END_IF`
		`IF g3 THEN`
		`  g4 := TRUE;`
		`  // weitere`
		`  // Aktivit.`
		`END_IF`
			`IF g4 THEN`
			`  g5 := TRUE;`
			`  // weitere`
			`  // Aktivit.`
			`END_IF`
				`IF g5 THEN`
				`  g1 := FALSE;`
				`  // weitere`
				`  // Aktivit.`
				`END_IF`

SPS-Programmierung

4.5.2 Symbolische Variablen

Symbolische Variablen werden mit einem symbolischen Namen und einem Datentyp deklariert. Bei Bedarf kann eine symbolische Variable mit einem Anfangswert initialisiert werden.

Symbolische Variablen werden vom System in freien Bereichen des SPS-Speichers abgelegt, d. h. die Adressen sind dem Anwender nicht bekannt.

Das folgende Beispiel zeigt eine Variablen-Deklaration zweier symbolischer Variablen:

```
VAR
    var1:   BOOL;
    var2:   INT (-22..12);
END_VAR
```

Sie können symbolische Variablen initialisieren und/oder sie als remanente Variablen deklarieren, indem Sie das Schlüsselwort ' RETAIN' verwenden.

4.5.3 Direkt dargestellte und adressierte Variablen

Direkt dargestellte Variablen werden nicht mit einem symbolischen Namen, sondern mit einer logischen Adresse deklariert.

Adressierte Variablen werden mit einem symbolischen Namen und mit einer logischen Adresse deklariert.

Direkt dargestellte Variablen und adressierte Variablen werden beide an der deklarierten logischen Adresse gespeichert. Der Programmierer muß deshalb darauf achten, daß er keine Speicheradresse doppelt vergibt.

Die Deklaration einer Speicheradresse besteht aus dem Schlüsselwort AT, dem Prozentzeichen "%", dem Präfix für den Speicherort, dem Präfix für die Größe und dem Namen der logischen Adresse.

Hinweis: Direkt dargestellte Variablen und adressierte Variablen können im globalen Variablen-Arbeitsblatt oder in Programmen deklariert werden. Zur Deklaration muß das Schlüsselwort VAR_GLOBAL verwendet werden.

Die folgende Tabelle zeigt die Präfixe für den Speicherort und die Größe direkt dargestellter und adressierter Variablen:

Präfix für den Speicherort	Beschreibung
I	Physikalischer Eingang
Q	Physikalischer Ausgang
M	Physikalische Adresse im SPS-Speicher
Präfix für die Größe	Beschreibung
X	Einzelbitgröße (nur mit Datentyp BOOL)
None	Einzelbitgröße
B	Byte-Größe (8 Bits)
W	Wort-Größe (16 Bits)
D	Doppelwort-Größe (32 Bits)
L	Langwort-Größe (64 Bits)

Das folgende Beispiel zeigt eine Deklaration direkt dargestellter und lokaler Variablen:

```
VAR
        var1    AT %QX 2.4      : BOOL;
        var2    AT %IW4         : WORD;
                AT %QB 7        : BYTE;
END_VAR
```

4.5.4 Remanente Variablen

Remanente Variablen sind Variablen, deren Werte auch gespeichert bleiben, wenn die Stromversorgung abgeschaltet wird. Bei einem Warmstart wird der letzte Wert der Variablen verwendet.

Zum Deklarieren remanenter Variablen muß das Schlüsselwort RETAIN verwendet werden, wie im folgenden Beispiel dargestellt:

```
VAR RETAIN
        var1    :       BOOL  := TRUE;
END_VAR
```

In diesem Beispiel hat die Variable den Anfangswert ' TRUE' , der den Anfangswert für einen Kaltstart angibt. Bei einem Warmstart wird der aktuelle Wert der Variablen verwendet.

Permanente Variablen

Permanente Variablen sind wie Remanente Variablen solche, deren Werte auch gespeichert bleiben, wenn die Stromversorgung abgeschaltet wird. Permanente Variablen sind im Gegensatz zu Remanenten Variablen auch Kaltstartsicher.
Da Permanente Variablen eine Besonderheit des B&R Programmiersystems darstellen, müssen die permanenten Variablen in einem eigenen Editor deklariert werden.

In B&R:

```
VAR_CONFIG
        myPermanentVar MeWi %MW0;
END_VAR
```

Voraussetzung: Es ist auf der CPU ein Speicherbereich für Permanente Variablen angelegt worden.

4.5.5 Initialisieren von Variablen

Gemäß der IEC 1131-3 können Variablen Anfangswerte zugewiesen werden. Das bedeutet, daß eine Variable, die zum ersten Mal in dem SPS-Programm verwendet wird, mit ihrem Anfangswert aufgerufen wird. Anfangswerte können allen Variablen zugewiesen werden, außer in VAR-EXTERNAL-Deklarationen.

Anfangswerte müssen am Ende der Deklarationszeile der Variablen mit ' :=' eingefügt werden, wie in dem folgenden Beispiel dargestellt:

```
VAR
        var1:   INT := 28;
        var2:   TIME := T#1s;
        var3:   AT%QX0.0 := TRUE;
END_VAR
```

Hinweis: Variablen an physikalischen Eingängen können nicht initialisiert werden.

Der Anfangswert muß zu dem Datentyp passen. Es ist beispielsweise nicht möglich, einer Variable vom Datentyp BOOL den Anfangswert ' 5' zuzuweisen. In diesem Fall zeigt das System eine Fehlermeldung an.

Variablen müssen nicht initialisiert werden. Wenn kein Anfangswert zugewiesen wird, wird die Variable mit dem Standard-Anfangswert des Datentyps oder bei remanenten Variablen mit dem gepufferten Wert initialisiert.

Bit Adressierung

Bit Adressierung erlaubt es, auf einzelne Bits einer Variable schreibend oder lesend zuzugreifen (Erweiterung zum IEC-Standard!).

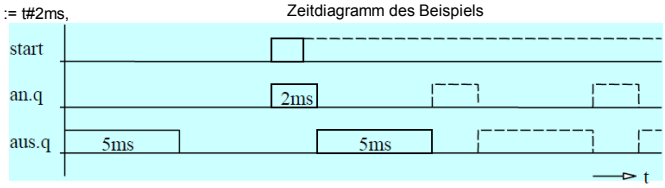
<Variable_name>.<Bit_number> Beispiel: y.3:=True;
<Variable_name>.<Constant_name> If k.4 Then …
<Variable_name>.<Enumerator>

Beispiel blinkende Signallampe

```
FUNCTION_BLOCK takt
    VAR_INPUT start : BOOL; zeit1, zeit2 : TIME; END_VAR
    VAR_OUTPUT q : BOOL; END_VAR
    VAR an, aus :TP; END_VAR
    an(IN := start & NOT aus.q, PT := zeit1);
    aus(IN := NOT an.q, PT := zeit2);
    q := an.q;
END_FUNCTION_BLOCK

PROGRAM signallampe
    VAR schalter AT %IX2.1 : BOOL; lampe AT %QX3.5 : BOOL;
        blinken : takt ; END_VAR
    blinken(start := schalter, zeit1 := t#2ms,
        zeit2 := t#5ms);
    lampe := blinken.q;
END_PROGRAM
```

Zeitdiagramm des Beispiels



Absolute Adressen:
1. Zeichen: %
2. Zeichen: I : Eingang, Q : Ausgang, M : Merker
3. Zeichen: X: Bit, B : Byte, W : Wort, D : Doppelwort, L : Langwort
4. Eigentliche Adresse.
Beispiel: %QB7

Absolute Adressen sollten nur in Datendeklarationen verwendet werden. In den Anweisungsteilen sollten nur symbolische Adressen eingesetzt werden.

© Dr. Franz Auinger / FH-OÖ / Wels