4.1 Einstieg in das IEC-Programmiersystem CoDeSys

CoDeSys: Controller Development System (Entwicklungssystem für Automatisierungs-Systeme)

4.1.1 Bezeichner und Schlüsselworte

Bezeichner können frei gewählt werden. Sie sind eine Folge von Buchstaben, Ziffern und Unterstrichzeichen (_). Sie müssen mit einem Buchstaben oder Unterstrichzeichen beginnen. Mehrere voranstehende und mehrfach eingebettete Unterstriche sind unzulässig. Außerdem dürfen Bezeichner keine Leerstellen enthalten. Zumindest sechs Zeichen müssen in sämtlichen Systemen unterstützt werden. Schlüsselworte werden grundsätzlich **mit Großbuchstaben** geschrieben!

Beispiel eines Bezeichners: VENTIL_6

Schlüsselwörter sind Zeichenkombinationen, die keine eingebetteten Leerzeichen enthalten dürfen. Schlüsselwörter sind festgelegt und dürfen für keinen anderen Zweck (z.B. als Variablennamen!) benutzt werden.
Beispiele für Schlüsselworte: ACTION...END_ACTION ; ARRAY...OF ; AT

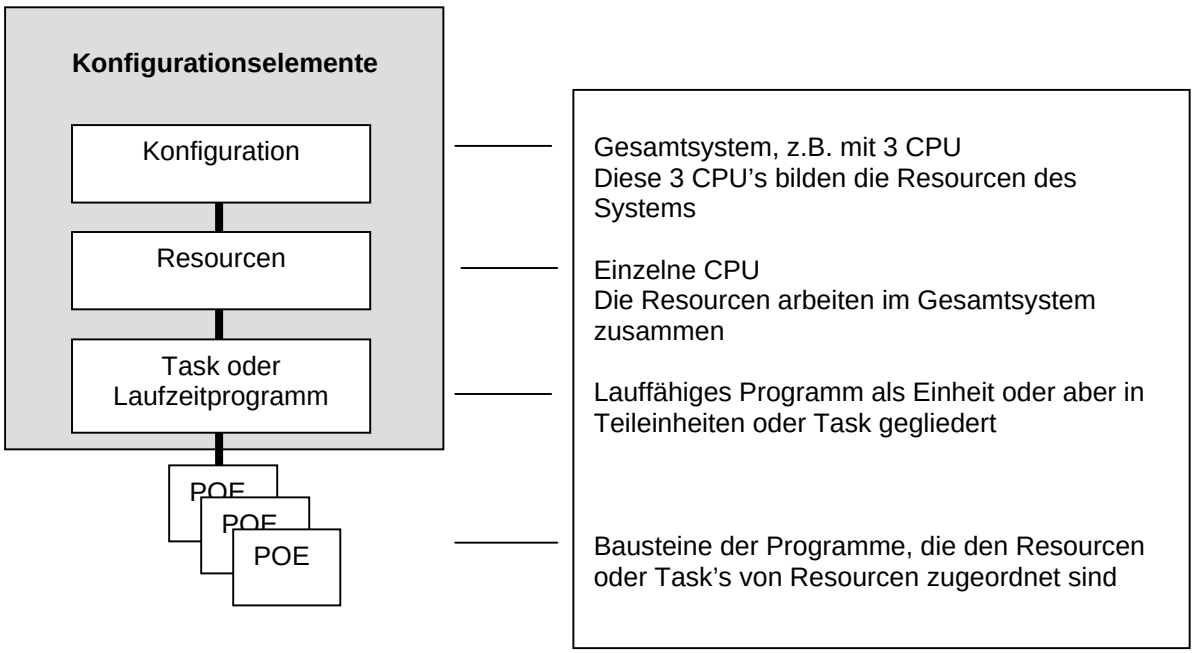
-> hierzu Liste aller Schlüsselworte auf Datei: -> Schlüsselworte

4.1.2 Beschreibung der Struktur von Automatisierungssystemen

Wie bereits in den Grundlagen Abschnitt 1.5.5 ausgeführt, gibt es in Automatisierungssystemen zu **konfigurierende** Elemente und zu **programmierende** Elemente.
Ein Softwaremodell nach IEC 61131-3 muss selbstverständlich Strukturen und Funktionen von Automatisierungssystemen **über die Grenzen von Ein-Prozessor-Programmen hinaus** ermöglichen, beispielsweise das Multi-Tasking einer CPU oder die Zusammenarbeit mehrerer CPU oder die Zusammenarbeit vernetzter Systeme.
Dazu werden

- die Ressourcen der Automatisierungssysteme per Software **konfiguriert**
- die Programme in **Programmorganisationseinheiten (POE's)** gegliedert

Die zu konfigurierenden Elemente sind z.B. die Ressourcen, die Task's, globale Variablen, die über Resourcengrenzen wirken, Kommunikationspfade u.a. Die Begriffe zur Strukturierung der Software in Automatisierungssystemen zeigt nachfolgendes Schema.



Wenn auf einer Hardwareumgebung mehrere Programme unabhängig voneinander und gleichzeitig laufen, muss den Programmen

- eine bestimmte Rechenzeit auf einer bestimmten Hardware und / oder
- eine Priorität und eine Startbedingung (zyklisch oder Interrupt)

zugeteilt werden.
Dann müssen Variablen häufig von mehreren Programmen gleichzeitig genutzt werden können.

Die notwendigen Informationen über die Systeme und ihr Zusammenwirken werden in eine Konfiguration mit dem **Schlüsselwort CONFIGURATION** geschrieben. Nachfolgender Kasten zeigt ein **formales** Beispiel, wie solche Konfigurationen mit entsprechenden Schlüsselwörtern beschrieben werden.

Wichtig in diesem Zusammenhang ist der Begriff „Laufzeitprogramm“. Darunter versteht man ein Gesamtprogramm mit Laufzeit-Eigenschaften, bestehend aus sämtlichen benötigten POE's und allen Task. Mit dieser Definition ist ein Laufzeitprogramm eine in sich abgeschlossene Programmeinheit, die selbständig in einer CPU ablaufen kann.
Ein Automatisierungssystem kann aus mehreren multitask-fähigen CPU's sowie Spezialkomponenten (-> Rechner-Ressourcen) bestehen.

Variablen vom Typ ACCESS werden benutzt, um Kommunikationskanäle zwischen Ressourcen zu beschreiben.

```
CONFIGURATION  Autom_Qualitaetsüberwachung
VAR_GLOBAL
        Auftrag:INT;
        Qualitätskennziffer:REAL;
END_VAR

RESSOURCE CPU_1
        TASK Task_1 (Intervall:=t#20s, PRIORITY:=5)
        TASK Task_2 (Single:=z2, PRORITY:=1)
        PROGRAMM Überwachung
END_RESSOURCE

RESOURCE CPU_2
        TASK Task_2 (Single:=z3, PRORITY:=2)
        PROGRAMM Auswertung
        PROGRAMM Störungsmeldung
END_RESOURCE

VAR_ACCESS
        QUALITAET: RESSOURCE CPU_ 1.%QW2 :WORD ;
END_VAR

END_CONFIGURATION
```

Die Konfiguration ist der POE-Ebene grundsätzlich übergeordnet.

Falls eine Konfiguration nur eine **einzige Ressource** besitzt, kann man auf die Angabe derselben verzichten. Dies ist bei einfachen Systemen der Normalfall! Die Konfigurationen größerer Systeme bleibt deshalb Aufgabe weiterführenden Lehrveranstaltungen.

Achtung!
Die **Programmieroberfläche CoDeSys** stellt Tools für nachfolgend farbig gekennzeichnete Aufgaben zur Verfügung. Sie beginnen mit der Gestaltung eines Projektes, erlauben aber mit den Software-Werkzeugen unter „Ressourcen“ auch eine Reihe von Konfigurationen.

Der hierarische Aufbau Konfiguration – Ressource – Laufzeitprogramm nach Bild 3-1 ist allerdings nicht erkennbar, zumal der Begriff Ressourcen hier im Sinne von „verfügbaren Werkzeugen“ für Konfigurationen verwendet wird.

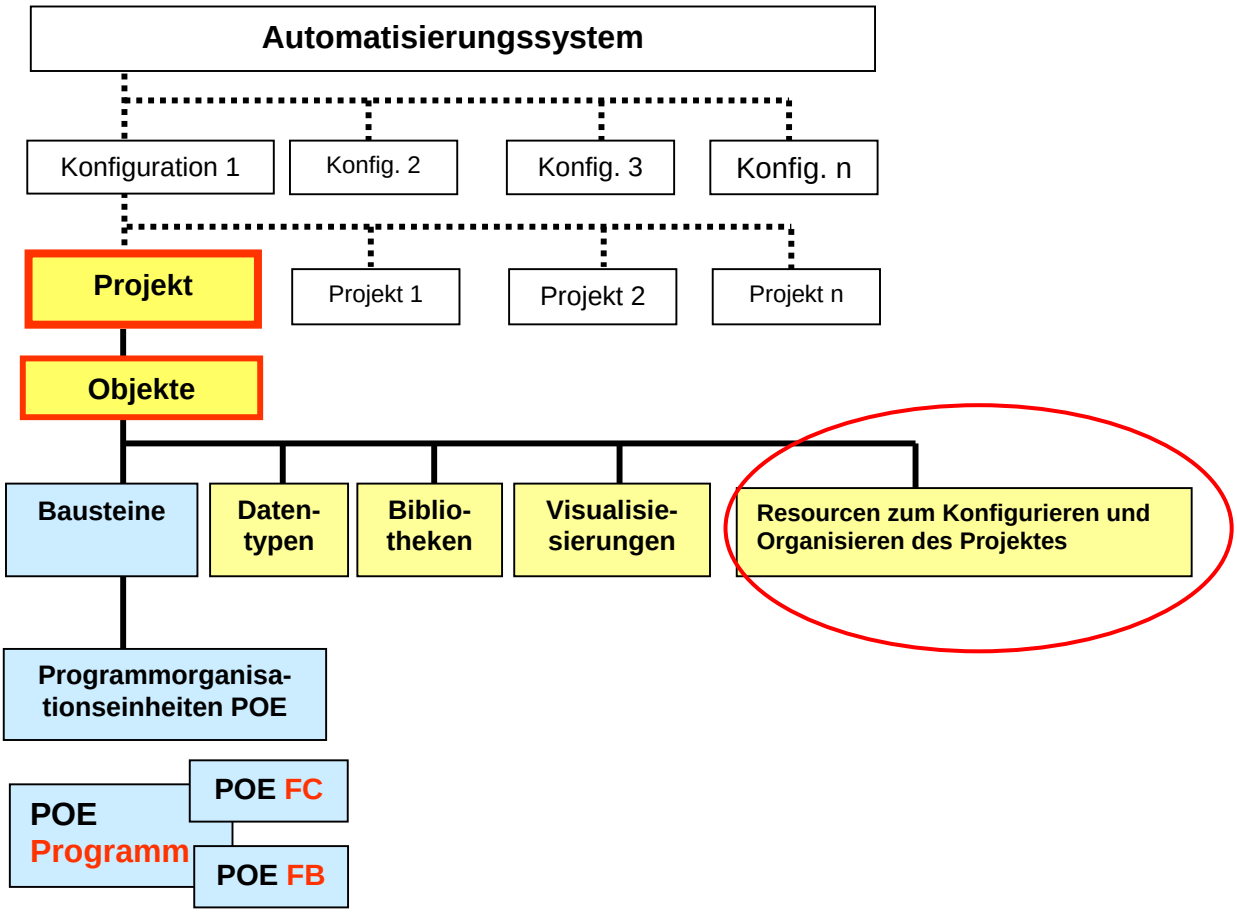
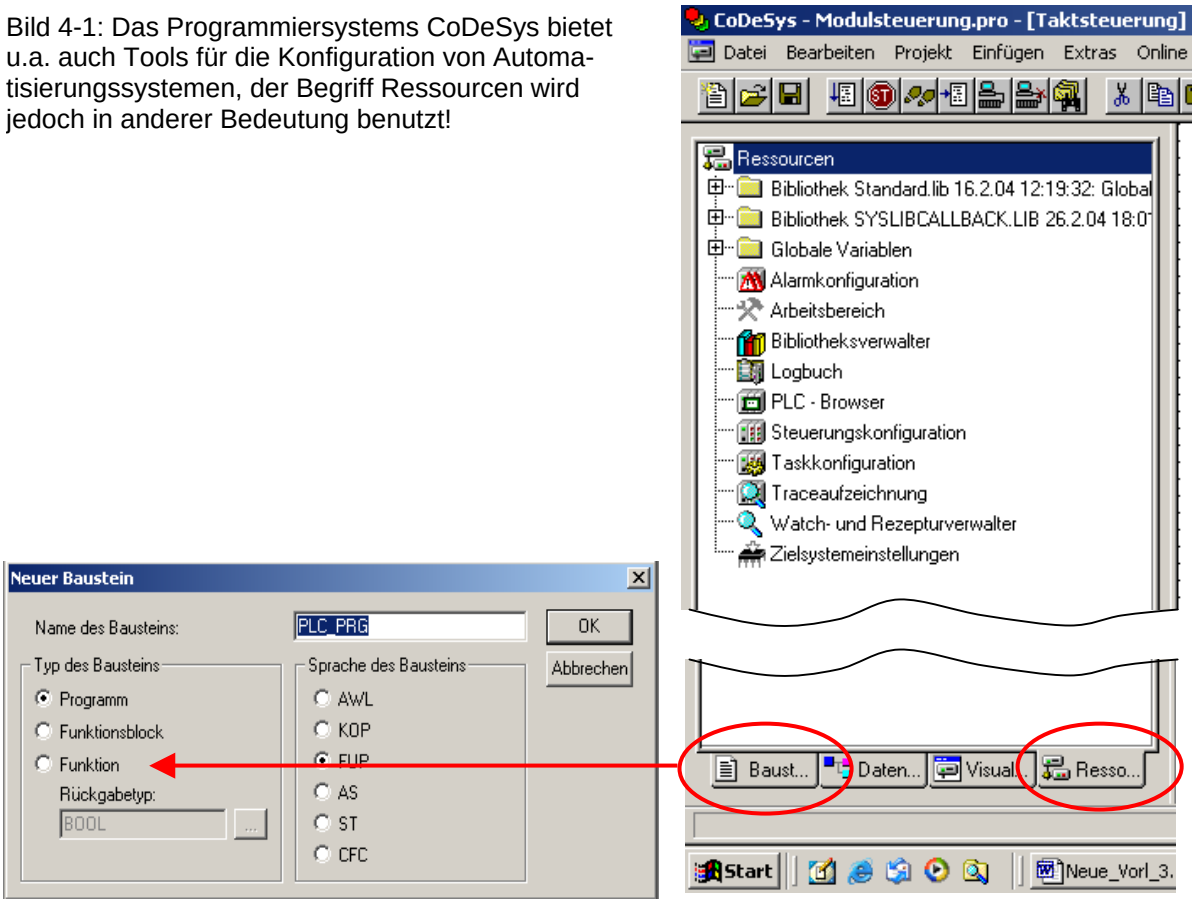


Bild 4-1: Das Programmiersystems CoDeSys bietet u.a. auch Tools für die Konfiguration von Automatisierungssystemen, der Begriff Ressourcen wird jedoch in anderer Bedeutung benutzt!



4.1.3 Programmorganisationseinheiten (POE)

4.1.3.1 Übersicht

- Zur Strukturierung eines Anwenderprogramms stehen nach IEC 61131-3 drei unterschiedliche Organisationseinheiten mit speziellen Eigenschaften zur Verfügung:
  Programme (Schlüsselwort PROGRAM)
  Funktionen (Schlüsselwort FUNCTION) und
  Funktionsblöcke (Schlüsselwort FUNCTION_BLOCK)

- Ein **Programm** ist allgemein ein zusammenhängender Teil eines Anwenderprogramms, der im gesamten Projekt bekannt ist.
  Das Betriebssystem organisiert die zyklische Bearbeitung des „Haupt"-Programms mit dem Schlüsselwort PLC_PRG. Alle anderen POE und damit auch alle anderen Programme müssen für die zyklische Bearbeitung in der POE PLC_PRG oder in anderen bereits aufgerufenen POE aufgerufen werden. Ein Aufruf in einer Funktion ist allerdings nicht möglich, wohl aber in einem Funktionsblock. Programme sind parametrierbar.

  Gegenüber der Funktion kann ein Programm bei der Ausführung einen oder mehrere Werte liefern. Von einer Ausführung bis zur nächsten Ausführung bleiben alle Werte erhalten.

  Gegenüber einem Funktionsblock zeichnet sich ein Programm wie folgt aus: Werden bei einem Aufruf eines Programms Werte geändert, so stehen die Änderungen für den nächsten Aufruf zur Verfügung, auch wenn eine andere POE das Programm aufruft. Beim Funktionsblock dagegen werden Änderungen nur in der jeweiligen Instanz wirksam, nicht aber in anderen Instanzen!

- **Funktionen** können nur für „Sofortlogik" eingesetzt werden, d.h. für die direkte logische Verarbeitung von Signalen. Innere Werte können nicht gespeichert werden. Sie können wie ein Operator benutzt werden. Grundsätzlich können Funktionen nur ein Datenelement – den Rückgabewert – liefern.

  Bei der Deklaration der Funktion und der Name des Rückgabewert und sein Datentyp festgelegt. Der Rückgabewert ersetzt den Funktionsaufruf und wird wie eine Ausgabevariable benutzt. Der Aufruf mit dem Befehl CAL ist nicht möglich.

  Typische Funktionen wie Rechen- oder Umwandlungsoperationen stehen als **Standard-funktionen** in den Programmiersystemen zur Verfügung

  Funktionen sind nur mit Eingangsvariablen parametrierbar. Ausgangsvariablen sind nicht erlaubt.

- **Funktionsbausteine (Funktionsblocks)** können mehrere Ausgangsvariablen besitzen und für gleiche Eingangswerte durchaus unterschiedliche Ausgangswerte zulassen. Das ist möglich, weil sie auch interne Variablen besitzen.

  Typische Funktionsbausteine wie Zähler, Zeitgeber, Bistabile Elemente, Trigger stehen als **Standard-FB** in den Bibliotheken der Programmiersysteme zur Verfügung. Häufig benötigte Programmteile kann der Anwender selbst als Funktionsbaustein definieren.

  Funktionsbausteine sind parametrierbar, wobei Eingangs-, Ausgangs- und innere Variablen benutzt werden können.

  Der Aufruf von Funktionsbausteinen von Programmen und anderen FB's aus erfolgt durch ihre **Instanzierung**. Die Instanz ist vergleichbar mit einer Kopie des deklarierten FB für einen speziellen Anwendungsfall. In der Deklaration wird die Instanz mit Ihrem Typ vereinbart. Damit wird für jede Instanz der notwendige Speicherbereich zur Verfügung gestellt.

  Zwischen den Aufrufen des Bausteins werden die Daten gespeichert werden. Diese Tatsache wird landläufig als „Gedächtnis" des FB beschrieben.

An dieser Stelle wird nachfolgend eine Übersicht aus dem Standardwerk von ...

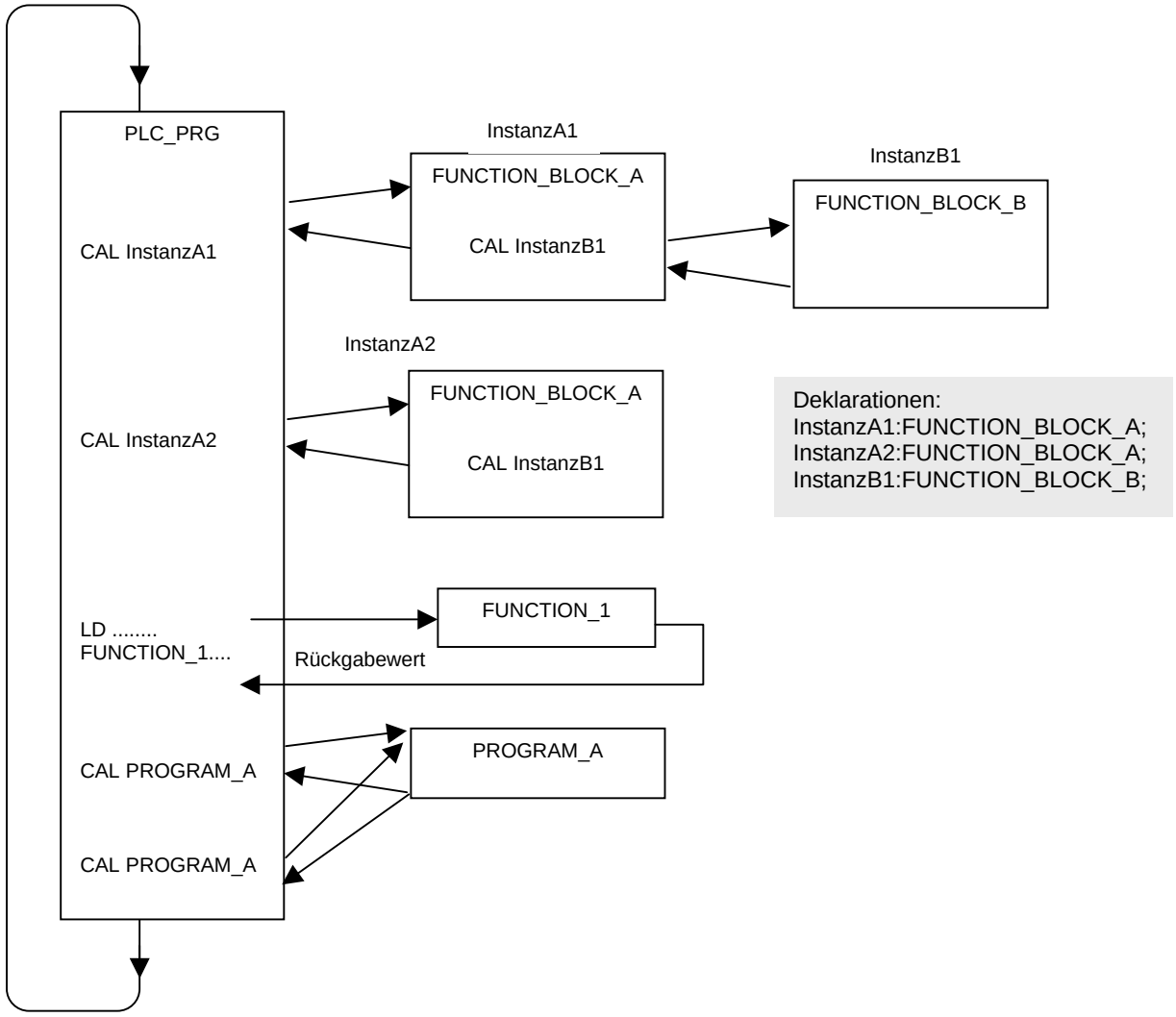
Karl-Heinz John und Michael Tiegelkamp: SPS-Programmierung mit IEC 61131-3 Konzepte und Programmiersprachen, Anforderungen an Programmiersysteme, Entscheidungshilfen 3. Auflage Springer Verlag 2000

...über die Eigenschaften und Möglichkeiten der POE gegeben. Die Einzelheiten dazu werden in den nachfolgenden Abschnitten ausgeführt. Nicht jedes Programmiersystem realisiert alle diese Grundeigenschaften in gleichem Maße!

Konzept	Funktion	Funktions-baustein	Programm
Eingangsparameter VAR_INPUT	ja	ja	ja
Ausgangsparameter VAR_OUTPUT	nein	ja	ja
Ein-Ausgangsparameter VAR_IN_OUT	nein	ja	ja
Funktionswert (Rückgabewert)	ja	nein	nein
Aufruf von Funktionen	ja	ja	ja
Aufruf von Funktionsbausteinen	nein	ja	ja
Aufruf von Programmen	nein	nein	nein
Deklaration globaler Variablen VAR_GLOBAL	nein	nein	ja
Zugriff auf externe Variablen VAR_EXTERNAL	nein	ja	ja
Deklaration direkt dargestellter Variablen	nein	nein, bei FB's nur in VAR_EXTERNAL	ja
Deklaration lokaler Variablen VAR	ja	ja	ja
Deklaration einer FB-Instanz	nein	ja	ja
Flankenerkennung verwendbar	nein	ja	ja
Verwendung von EN/ENO	ja	ja	nein
Pufferung von lokalen und Ausgangsdaten	nein	ja	ja
Indirekter FB-Aufruf	nein	ja	ja
Verwendung von Funktionswerten als Eingangsparameter	ja	ja	ja
Verwendung von FB-Instanzen als Eingangsparameter	ja	ja	ja
Rekursiver Aufruf	nein	nein	nein

Übersicht aus Karl-Heinz John und Michael Tiegelkamp: SPS Programmierung nach IEC 61131-3

Nachfolgendes Schema zeigt das Wesen der zyklischen Programmbearbeitung mit den POE PROGRAM, FUNCTION und FUNKTION_BLOCK



Für alle drei Typen von POE sieht IEC 61131-3 grundsätzlich zwei Teile vor, die mit Programmkopf (Header) und Programm-Rumpf (Body) bezeichnet werden. Für das Ausführen beider Teile sind exakte Regeln zu beachten!
Der **Programm-Kopf** enthält Deklarationen (und Konfigurationen).
Der **Programm-Rumpf** enthält das eigentliche Programm.

Zu Beginn jedes Programms bzw. jeder Programmorganisationseinheit (POE) sind alle benutzten Variablen im Programmkopf eindeutig zu deklarieren**! Dies ist ein deutliches Kennzeichen von IEC-konformen Programmen und sollte nicht unterlaufen werden!** Mit der Deklaration der Variablen wird der erforderliche Platz für alle Daten im Datenspeicher des Automatisierungsgerätes reserviert, oder es werden eine Menge von Ein- oder Ausgangsparametern für die Schnittstellen der POE zueinander festgelegt.

> Die Norm definiert die Variablen als „Mittel zur Identifizierung von Datenobjekten, deren Inhalt sich ändern darf, d.h. Daten, die mit Eingängen, Ausgängen oder Speicherplätzen der SPS verbunden sind."

Das Programm selbst wird ohne Bezug auf die Hardware – d.h. ohne Bezug auf Eingänge, Ausgänge oder spezielle Datenbereiche – geschrieben. Nur im Deklarationsteil werden Variablen bei Bedarf „auf Adressen gelegt", an besten nur im Hauptprogramm PLC_PRG. Eine Änderung der Verdrahtung hat dann keinen Einfluss auf das Programm, sondern lediglich auf den Deklarationsteil

4.1.3.2 Regeln für die Variablendeklaration

Nachfolgender Kasten zeigt alle Schlüsselworte für die Variablendeklaration.

Schlüssselwort	**Anwendung**
VAR	lokal innerhalb der POE
VAR_INPUT	von außen kommend (gelesen), in der POE nicht änderbar
VAR_OUTPUT	von der POE nach außen geliefert (geschrieben)
VAR_IN_OUT	von außen kommend, kann innerhalb der POE geändert werden
VAR_GLOBAL	über alle POE eines Projektes gültig
VAR_EXTERNAL	von einer Konfiguration geliefert und innerhalb der POE änderbar
VAR_END	Ende eines Variablenblockes
RETAIN	Remanente Daten
PERSISTENT	Nullspannungssicher gepuffert
AT (%Adresse)	Zuweisung des Speicherortes (-> “Legen auf Adresse”)

Jede Variablendeklaration wird in Blöcke geschrieben, die durch die **Schlüsselworte VAR und END_VAR** begrenzt werden. Die Reihenfolge der Blöcke spielt keine Rolle.
Bei der Variablendeklaration wird der Name der Variablen mit einem Datentyp verbunden.

Der **Name oder Bezeichner** ist im Rahmen der zugelassenen Zeichen frei wählbar. Es ist sinnvoll, **prozessverständliche** Namen zu wählen! Groß- und Kleinschreibung werden nicht unterschieden und sind deshalb ohne Belang.

Mindestangaben einer Variablendeklaration:

Name : Datentyp ; Beispiel: **Teilezahl:INT;**

Erläuterung zu diesem Beispiel:
Mit der Deklaration Teilezahl:INT; wird für die Variable mit dem Namen „Teilezahl“ ein Speicherraum von 16 Bit reserviert. Der 16-Bit-Inhalt des Wortes im Datenspeicher wird dann als Ganzzahl interpretiert.

Durch **Optionen** können zusätzliche Festlegungen getroffen werden wie Legen auf Adresse und Vorgabe eines Anfangwertes (Initialwertes). Die Initialisierung erfolgt durch Vorgabe von Anfangswerten mit dem Zeichen := .

Mögliche Optionen:

Name : Datentyp := Initialwert ;	zusätzliche Angabe eines Anfangwertes
Name AT Adresse : Datentyp ;	Mit Schlüsselwort AT wird Variable auf bestimmte Adresse gelegt, zumeist auf Eingänge oder Ausgänge (im Ausnahmefall auf Merker)
Name AT Adresse : Datentyp := Initialwert ;	Maximaldeklaration
Beispiele:	**Sensoren_0_5 AT %IB4:Byte;** **Start AT %IX0.2:BOOL :=FALSE;** **Wert:REAL:=0.23;**

Werden keine Anfangswerte vorgegeben, so starten die Variablen mit einem vordefinierten Wert, der überwiegend Null zum Inhalt hat.

Bei der Variablendeklaration werden weiter auch der Wirkungsbereich (global oder lokal) und weitere Eigenschaften (wie nullspannungsfest u.a.) festgelegt.
Dazu wird das Schlüsselwort VAR ergänzt. Beispiele sind VAR_GOBAL, VAR_RETAIN u.a.

Beispiele:

```
VAR
Summand_1:INT:=200;
Summand_2:INT;
Summe:INT;
Anzeige AT %QX0.1:BOOL;
Stoerung AT %IX2.3:BOOL;
Wartezeit:TIME:=5s;
END_VAR
```

```
VAR_GLOBAL
BCD_Schalter_1 AT %IW 2:WORD;
Ventile_1-16 AT %QW12:WORD;
Temperatur_A:REAL;
Fehlerquote:REAL;
Teilezahl:INT;
END_VAR
```

In die Variablendeklaration können **Konstanten** einbezogen werden. Die Norm spricht von Typed Literals. Konstanten werden mit dem **Schlüsselwort CONSTANT** gekennzeichnet. Sie können lokal für eine POE oder global für alle POE deklariert werden.

Beispiele:

```
VAR CONSTANT
Divisor:INT:=20;
Zeitdifferenz:TIME:=T#12h;
END_VAR
```

```
VAR_GLOBAL CONSTANT
Immer_1:BOOL:=TRUE;
Multiplikator_A:RAL:=3.14;
Eintrag:STRING:='€';
END_VAR
```

Nutzen Sie die Auflistung möglicher Konstanten unter Index-Stichwort „Konstanten“ der Online-Hilfe CoDeSys

Durch Benutzung des Schlüsselwortes **RETAIN** werden **remanente Daten** erzeugt. Solche Variablen behalten beim Aus- und Einschalten der Steuerung oder Online-Reset ihre Werte, d.h. bei erneutem Start wird mit den gespeicherten aktuellen Werten weitergearbeitet.

Dagegen werden alle anderen Variablen in diesem Fall neu initialisiert, entweder mit ihren Standard-Initialisierungswerten (zumeist Inhalt Null) oder nach Vorgabe der Deklaration.

Beispiele:

```
VAR RETAIN
Teilezaehler:INT;
Endtemperatur:REAL;
END_VAR
```

```
VAR_GLOBAL RETAIN
Produktion:DINT;
Restzeit:TIME;
END_VAR
```

Nicht remanent sind RETAIN-Variablen allerdings bei bestimmten (systemabhängigen) Befehlen wie „Reset auf Ursprung“ oder erneutem Download des Programms.

Für solche Probleme sieht die Norm weiter das **Schlüsselwort PERSISTENT** vor.
Nachfolgende Tabelle zeigt das Verhalten der Variablen RETAIN und PERSISTENT im System CoDeSys. Für VAR RETAIN PERSISTENT kann auch geschrieben werden VAR PERSISTENT RETAIN

Online-Befehl	VAR	VAR RETAIN	VAR PERSISTENT	VAR RETAIN PERSISTENT
RESET	neu initial.	**remanent**	neu initial.	**remanent**
RESET KALT	neu initial.	neu initial.	neu initial.	neu initial.
RESET URSPRUNG	neu initial.	neu initial.	neu initial.	neu initial.
DOWNLOAD	neu initial.	neu initial.	**remanent**	**remanent**
ONLINE CHANGE	**remanent**	**remanent**	**remanent**	**remanent**

ONLINE CHANGE bedeutet die Funktion, bei laufender Steuerung Änderungen des Programms vorzunehmen.

Beispiel:

```
VAR RETAIN PERSISTENT
Betriebsstundenzähler:TIME;
Sollwert:REAL;
END_VAR
```

Achtung! Im System CoDeSys gilt hinsichtlich Wirkung und Speicherplatzbedarf:

Wenn eine lokale Variable in einem **Programm** als RETAIN deklariert ist, wird genau diese Variable im RETAIN-Bereich gespeichert (wie eine globale RETAIN-Variable).

Wenn eine lokale Variable in einem **Funktionsblock** als RETAIN deklariert ist, wird die komplette Instanz dieses Funktionsblocks im RETAIN-Bereich gespeichert (alle Daten des Bausteins), wobei jedoch nur die deklarierte RETAIN-Variable als solche behandelt wird.

Wenn eine lokale Variable in einer **Funktion** als RETAIN deklariert ist, hat dies keine Auswirkung. Die Variable wird nicht im Retain-Bereich gespeichert! Wird eine lokale Variable in einer Funktion als PERSISTENT deklariert, bleibt dies ebenfalls ohne Wirkung!

4.1.3.3 Direkt dargestellte Variablen („Legen auf Adressen")

Bestimmte Variablen erhalten ihre Inhalte von Eingangskartenbaugruppen oder schreiben ihre Werte in Ausgangskartenbaugruppen.
In Sonderfällen möchten Anwender den Wert von Variablen in Datenspeicherbereiche schreiben, die sie selbst verwalten wollen.

In diesen Fällen werden die Variablen mit dem Schlüsselwort AT auf Bit-, Byte, Wort- oder Doppelwortadressen gelegt. Die Kennungen dieser Speichergrößen und der Speicherorte sind nachfolgend dargestellt. Grundsätzlich werden Adressen durch das **Zeichen %** gekennzeichnet.

Bit:	Kennung: **X**	
Byte:	Kennung: **B**	
Wort:	Kennung: **W**	
Doppelwort:	Kennung: **DW**	(32 Bit)
Langwort:	Kennung: **L**	(62 Bit)

Eingang:	Kennung: **I**	(Input)
Ausgang:	Kennung: **Q**	(Quit)
Merker:	Kennung: **M**	(nur in Ausnahmefällen anwenden!)

Die Adressierung erfolgt mit Bit- Byte- und Wortadressen (siehe Grundlagen Abschnitt 1.5.3).
Zur Problematik von Variablen und Symbole siehe Abschnitt 4.3.5.

Werden Merker benutzt, so werden die Daten selbstätig in einen nullspannungsfesten RETAIN-Speicher geschrieben. Das Wort Merker fungiert hier wie ein Schlüsselwort.

Achtung! **Um unkontrollierte Überschreibungen zu vermeiden, sollten Merker und VAR_RETAIN niemals gleichzeitig benutzt werden!**

Achtung! **Nur Variable und Globale Variable - nicht aber VAR_INPUT und VAR_OUTPUT – dürfen auf Adressen gelegt werden.**

4.1.3.4 Parameterübergabe an POE und parametrierbare Bausteine

Für die Übergabe von Daten an POE und für die Ausgabe von Daten aus POE sieht IEC 1131-3 nachfolgende spezielle Variablen vor, die auch als Parameter bezeichnet werden:

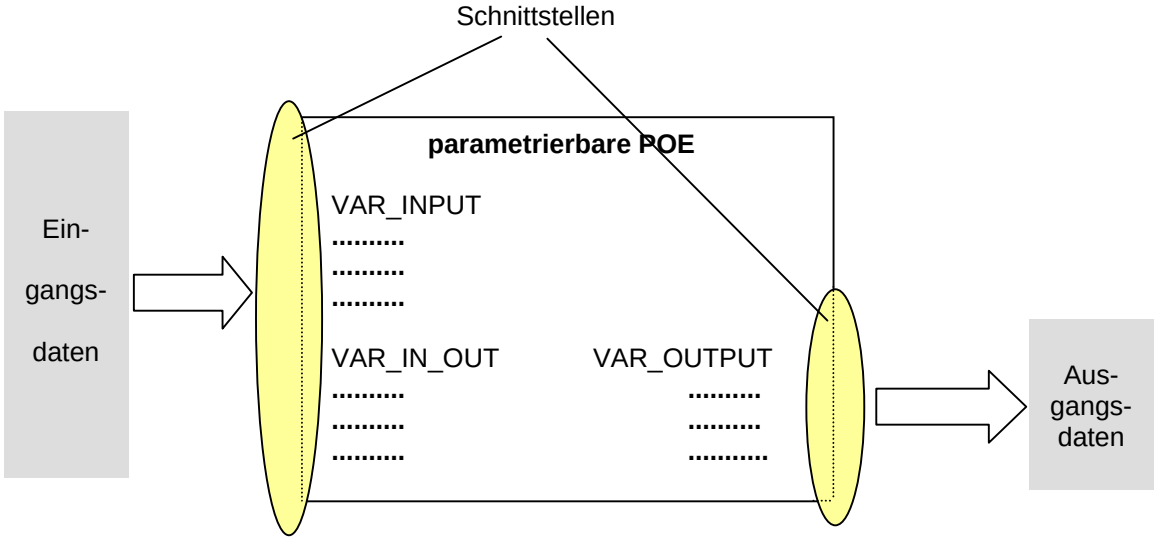
Variablen Typ	**Funktion der Variablen**
VAR_INPUT	Mit diesen Variablen erhält die POE Eingangsinformationen
VAR_OUTPUT	Mit diesen Variablen gibt die POE Ausgangsinformationen zurück
VAR_IN_OUT	Mit diesen Variablen erhält die POE Eingangsinformationen und gibt diese verändert zurück

In Funktionen gibt es abweichend nur Parameter vom Typ VAR_INPUT.

Durch Anwendung solcher Variablen entstehen **parametrierbare Bausteine**. Das sind allgemein formulierte Programmteile, die ihre konkrete Funktion erst beim Aufruf im Hauptprogramm durch **Übergabe konkreter, d.h. aktueller Parameter** erlangen.

Die Variablen der Typen VAR_INPUT, VAR_OUTPUT und VAR_IN_OUT stellen somit die **Schnittstelle** einer POE zu anderen POE dar.

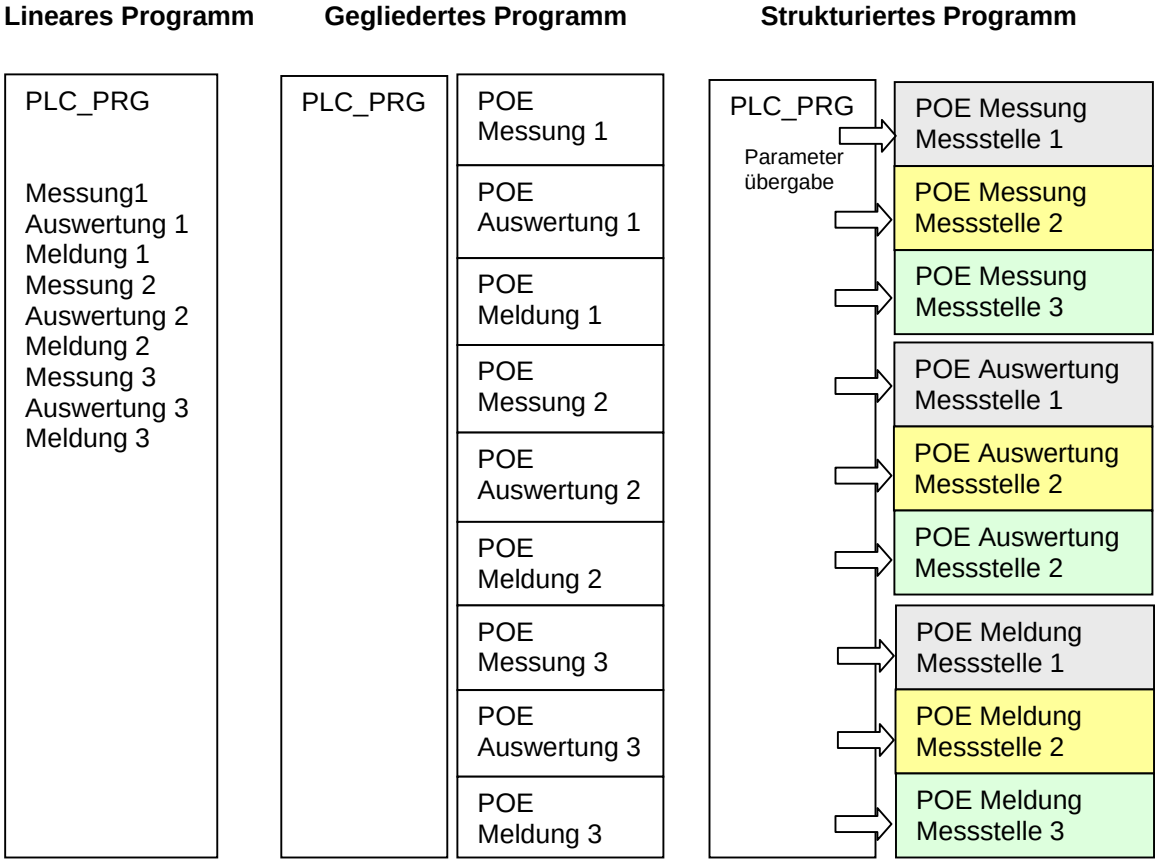
Die Verwendung parametrierbarer POE ist ein wesentliches Element von IEC 1131-3. Das Verfahren erlaubt den Einsatz wiederverwendbarer Programme in Form selbst geschriebener POE und vielfältiger Biblioteken. Bibliotheken gehören zum Programmiersystem. Von Grundsatz her sind alle Bibliotheksbausteine parametrierbare POE.



Wird die Abarbeitung eines Programms nicht durch Task organisiert, muss das Projekt den Baustein PLC-PRG enthalten (im System Simatic S5 / S7 der Organisationsbaustein OB1). Ein vom Umfang

her begrenztes Programm kann durchaus allein im Baustein PLC_PRG ohne Verzweigungen zu anderen Bausteinen niedergeschrieben werden. Ein solches Programm bezeichnet man als **lineares Programm.**
In POE **gegliederte Programme** sind gegenüber linearen Programmen übersichtlicher. Bei Änderungen müssen nur bestimmte POE berücksichtigt werden.

In der Automatisierungstechnik müssen oftmals gleichartige Programmteile mehrfach angewendet werden. Strukturiert man mit parametrierbaren POE, dann können durch Export- / Import-Funktionen Programmteile in anderen Projekten wiederwendet werden oder aber mehrmals im gleichen Projekt. In Fachkreisen spricht man erst bei Anwendung parametrierbarer POE von **strukturierten Programmen**.



4.1.3.5 Variablen und Symbolik

Einige Programmiersysteme ermöglichen zur besseren Lesbarkeit von Programmen für die **direkt adressierten Operanden** symbolische Namen zu verwenden. Diese werden kurz **Symbole** genannt.

Dazu gehört Step7 des Automatisierungssystems Siemens Simatic S7 und ebenso der Vorgänger Step5. Für die vom Anwender zu adressierenden Operanden im Merkerbereich, im Bereich der globalen Datenbausteine, im Prozessabbild der Eingänge (PAE) und Ausgänge (PAA) sowie für Peripheriebereiche und für Timer und Zähler werden hier in einer Symboltabelle frei wählbare Symbole verwaltet. Bei der Programmierung kann der Anwender entscheiden, ob er das Programm mit absoluten Operanden, mit symbolischen Operanden oder in der Option „mit Symbolinformation“ mit beiden betrachten und bearbeiten will. Da alle diese Operanden die Funktion globaler Variablen haben, stehen Symbole grundsätzlich für **globale Variablen**.

Symbole werden bei Step7 mit doppeltem Hochkomma gekennzeichnet. Für Symboltabellen wird auch der Begriff Zuordnungstabellen benutzt.

Beispiele für Step7:

Operand	Symbol
M100.0	“Steuerung_eingeschaltet“
DB5.DBW12	“Sollstueckzahl“
E10.3	“T_Antrieb_ein“
A2.7	“LED_Anrieb2“
PEW304	“Temperatur“
T12	“Z_Ueberwachung“
Z1	“Teilezeaehler“

Symbole dürfen nicht mit den Bezeichnern von Variablen verwechselt werden! Hinter jedem Symbol liegt eine vom Anwender festgelegte Adresse, während hinter Variablen nur im Sonderfall beim Legen auf Adressen mit Schlüsselwort AT ein Hardwarebezug besteht!

Bei Programmen nach IEC 61131 erhöht die grundsätzliche Verwendung von Variablen die Flexibilität eines Programms, denn solange die Anbindung an die Peripherie noch nicht festgelegt ist, ist ein Programm universeller einsetzbar.

Die erforderliche Adressierung sollte immer zum spätest möglichen Zeitpunkt und in einer möglichst hohen Ebene, bei großen Projekten in einer Konfiguration oder Ressource festgelegt werden.

Es ist zumeist ausreichend, für den direkten Zugriff auf die Peripherie allein in der POE PLC_PRG mit absoluten Adressen zu arbeiten. In Funktionen und Funktionsbausteinen ist die Verwendung von absoluten Adressen grundsätzlich zu vermeiden!

4.1.3.6 Zusammenfassung: Wo werden welche Variablen deklariert?

Variablendeklaration mit Schlüsselwort ....

VAR Diese Variablen gelten nur in der POE, in der sie deklariert wurden. Es sind lokale Variablen.

VAR_GLOBAL Diese Variablen gelten für alle POE im Gesamtprogramm. Es sind globale Variablen. Sie sind von allen POE's lesbar und änderbar.

Globale Variablen werden vorrangig für die Adressierung der Peripherie verwendet.

VAR_INPUT dienen zur Parameterübergabe beim Aufruf einer Funktion, eines Funktions-bausteins oder eines Programms.
Diese Variablen können in diesen POE abgefragt werden. In der graphischen Darstellung stehen diese Variablen auf der linken Seite des FB/FUN-Symbols. Es können beim Aufruf nur Werte mit dem angegebenen Datentyp übergeben werden.

VAR_OUTPUT sind die Ausgangsvariablen eines Funktionsbausteins oder eines Programms.
Hier stehen die Ergebnisse im angegebenen Datentyp für die Verwendung in anderen POE bereit.
In der graphischen Darstellung stehen diese Variablen auf der rechten Seite des FB/FUN-Symbols.
Bei einer Funktion wird das Ergebnis als Rückgabewert auf dem Funktionsnamen abgelegt. Es wird also kein VAR_OUTPUT benötigt.

VAR_IN_OUT Diese Variablen wirken bei FB und Programm wie VAR_INPUT, sie können aber in der POE geschrieben und verändert ausgegeben werden.
In Funktionen sind diese Variablen nicht zulässig.
In der graphischen Darstellung stehen diese Variablen auf der linken Seite des FB/FUN-Symbols.

VAR_EXTERNAL Eine globale Variable aus der Konfiguration kann dann in einem Element dieser Konfiguration benutzt werden, wenn sie in dem Element zusätzlich als VAR_EXTERNAL definiert wird.

VAR_ACCESS Mit diesen Kommunikationsvariablen werden Zugriffspfade für die Kommunikation definiert.

Achtung! **Var_IN_OUT und VAR_EXTERNAL können nicht mit Anfangswerten initialisiert werden**

Hinweis zu Step7: Hinsichtlich der Datenhaltung bestehen deutliche Unterschiede zwischen der Norm IEC 61131-3 und Step7 (hierzu auch Abschnitt 3.5)

Bei konventioneller Arbeit mit **Step7** schreibt man Daten in Merker und in Datenelemente der Datenbausteine. Beide Bereiche sind globale Daten und müssen vom Anwender **selbst adressiert** werden, was eine nicht zu unterschätzende Fehlerquelle ist!! Die Fehlerhäufigkeit beim Arbeiten mit Merkern und Elementen von Datenbausteinen kann durch Verwendung von **Symbolen** deutlich gesenkt werden!
Das ändert aber nichts an der Notwendigkeit, dass der Programmierer die Speicherzellen selbst adressieren muss! Bei Step7 steht hinter jedem Symbol eine Adresse (siehe Abschnitt 3.5)

Dagegen adressieren Programmiersysteme wie CoDeSys bei der Variablendeklaration die Speicher selbst.

Dieses Verfahren erfolgt nun wiederum auch bei Step7, wenn man lokale Variablen Daten in der Deklarationstabelle der Funktionen (FC) und Funktionsbausteine (FB) verwendet.

FC erlauben neben den Parametern IN, OUT und IN_OUT nur lokale Variable vom Typ TEMP (temporäre Variable).
Eine temporäre Variable dient als Zwischenspeicher („Schmierzettel“), um Werte innerhalb eines Bausteins im Programmablauf „von oben nach unten“ weiter zu geben (Prinzip: erst schreiben, dann lesen!) Die Information der TEMP- Variablen steht nach einem Programmzyklus nicht mehr zur Verfügung.

FB erlauben neben den Parametern IN, OUT und IN_OUT sowie lokalen temporären Variablen auch lokale statische Variablen (Typ STAT). Diese werden im Instanzdatenbaustein verwaltet, so dass der Baustein über ein „Gedächtnis“ verfügt. Die Daten bleiben solange erhalten, bis sie mit neuen Werte überschrieben werden, auch über Zyklen und Bausteinaufrufe.

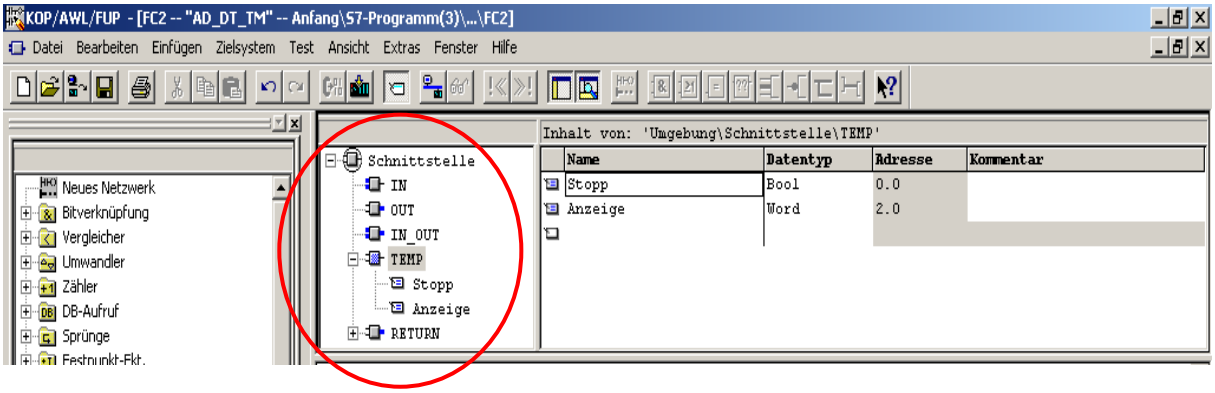


Bild 4-2: Deklaration lokaler Variablen in Step7: Die Funktion erlaubt neben den Parametern IN, OUT und IN_OUT nur temporäre Variable.
Deklariert wurden im Bild die Variablen „Stopp:BOOL“ und „Anzeige:WORD“.

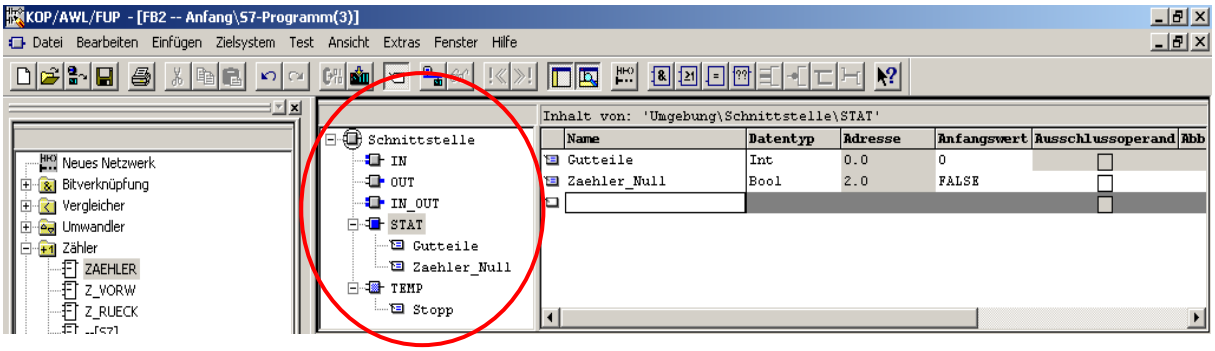


Bild 4-3: Deklaration lokaler Variablen in Step7: Der Funktionsbaustein erlaubt neben den Parametern IN, OUT und IN_OUT temporäre und statische Variable.
Deklariert wurden im Bild die statischen Variablen „Gutteile:INT“ und „Zaehler_Null:BOOL“ und die temporäre Variable “Stopp:BOOL”.

Details der Variablendeklaration in Step7 – Funktionen und – Funktionsbausteinen werden in den Abschnitten 8 und 9 behandelt.

4.1.4 Datenformate

IEC 61131-3 definiert **elementare und zusammengesetzte Datentypen**. Zusammengesetzte Datentypen wie ARRAY und STRUCT sowie spezielle **User defined Typs (UDT)** werden im Abschnitt 14 behandelt.

Mit dem Schlüsselwort ANY_ fasst man die elementaren Datentypen in fünf Gruppen zusammen:

- Bitfolgen: ANY_BIT
- Alle Zahlen: ANY_NUM
- Ganzzahlen mit und ohne Vorzeichen: ANY_INT
- Gleitpunktzahlen: ANY_REAL
- Datum, Zeitpunkt: ANY_DATE
- Zeichenfolge, Zeitdauer ANY (allgemeinste Vorgabe)

<table>
<tr><th colspan="5">ANY</th></tr>
<tr><th>ANY_BIT</th><th colspan="3">ANY_NUM</th><th>ANY_DATE</th><th rowspan="3"></th></tr>
<tr><td rowspan="2">BOOL
BYTE
WORD
DWORD
LWORD</td><th colspan="2">ANY_INT</th><th>ANY_REAL</th><td rowspan="2">DATE
TIME_OF_DAY
DATE_AND_TIME</td></tr>
<tr><td>INT
SINT
DINT
LINT</td><td>UINT
USINT
UDINT
ULINT</td><td>REAL
LREAL</td><td>TIME
STRING</td></tr>
</table>

Bedeutung der Schlüsselworte:

DWORD:	Double Word:	Doppelwort	
LWORD:	Long Word:	Langwort	
INT:	Integer:	Ganzzahl	
SINT:	Short Integer:	kurze Ganzzahl	
DINT:	Double Integer:	Doppelte Ganzzahl	
LINT:	Long Integer:	Lange Ganzzahl	
UINT:	Unsigned Integer:	Ganzzahl ohne Vorzeichen	(0..+65 535)
USINT:	Unsigned Short Integer:	kurze Ganzzahl ohne Vorzeichen	(0..+255)
UDINT:	Unsigned Double Int:	Doppelte Ganzzahl ohne Vorzeichen	(0..+2^{32}-1)
ULINT:	Unsigned Long Integer:	Langzahl ohne Vorzeichen	(0..+2^{64}-1
REAL:	Real:	Gleitpunktzahl	
LREAL:	Long Real:	Lange Gleitpunktzahl	

Nicht alle Hersteller unterstützen alle Datentypen. Nachfolgende elementare Datenformate bilden das unverzichtbare **Grundgerüst der Variablendeklaration**.

Schlüsselwort	Datentyp	Größe	Schreibweise / Wertebereiche
Bit-Datentypen			
BOOL	Boolesche Variable	1 Bit	FALSE (0) , TRUE (1)
BYTE	8 Bit-Folge oder 2 Hex-Zahlen	8 Bit	B#16#00...FF
WORD	16 Bit-Folge oder 4 Hex-Zahlen	16 Bit	W#16#0000...FFFF
DWORD	32 Bit-Folge oder 8 Hex-Zahlen	32 Bit	DW#16#0000_0000...FFFF_FFFF
LWORD	64 Bit-Folge oder 16 Hex-Zahlen	64 Bit	LW#16#0...FFFF_FFFF_FFFF_FFFF
CHAR	ASCII-Zeichen	8 Bit	‘ X‘ , ‘ +‘, ‘ &‘
Arithmetische Typen			
INT	Ganze Zahlen (Festpunktzahlen)	16 Bit	-32768 .... +32767 (0...65535)
DINT	Ganze Zahlen (Festpunktzahlen)	32 Bit	L# -2 147 483 648 ... 2 147 483 647
REAL	Reelle Zahlen (Gleitpunktzahlen)	32 Bit	Beispiel: 634.57 oder 6.3457e+02
Zeittypen			
TIME	Zeitdauer (IEC)	32 Bit	TIME# -24d20h31min ... + 24d20h31min
TIME OF DAY	Uhrzeit (Tageszeit)	32 Bit	TOD# 23:59:59.9
DATE	Datum	32 Bit	DATE#1990-01-01
DATE_AND_TIME	Zeitstempel: Datum und Uhrzeit	32 Bit	DT#2007-02-04-10:15:30
S5TIME	Zeitdauer (S5-Format, nur bei Step7)	16 Bit	S5T# 0ms... 9990s

Erläuterungen zu den Datentypen für Zeit und Datum

Datentyp	Schlüsselwort	Interne Wertung
Zeitdauer	TIME	Zeitdauer in Millisekunden
Tageszeit	TIME_OF_DAY oder TOD	Zeit im Millisekunden ab 00:00Uhr
Datum	DATE	Zeit in Sekunden ab 01.01.1970 00:00 Uhr
Datum und Uhrzeit (Zeitstempel)	DATE_AND_TIME oder DT	Zeit in Sekunden ab 01.01.1970 00:00 Uhr

TIME: Zeitbasis ist die Millisekunde. Jede andere Zeitangabe wird intern in eine Anzahl von Millisekunden umgerechnet. Die Gesamtzahl von Millisekunden steht dualcodiert in einem 32 Bit breiten Doppelwort (DWORD). Der größtmögliche Dezimalwert eines Doppelwortes ist 4.294.967.295.

Werden größere Werte vorgegeben, meldet das System CoDeSys den Fehler „Überlauf in Zeitkonstante“. Es ist aber zu beachten, dass die Datenbreite in unterschiedlichen Programmiersystemen auch systemabhängig sein kann.

TOD: Auch wenn die Tageszeit im „Klartext“ eingetragen werden kann, so wird sie intern doch als Anzahl der Millisekunden ab 00:00 Uhr gewertet.

DATE: Auch hier wird die Angabe in „Klartext“ intern in eine Anzahl von Sekunden ab dem festgelegten Zeitpunkt 01.01.1970 00:00 Uhr umgerechnet.

DT: ebenfalls Umrechnung in die Anzahl von Sekunden ab dem festgelegten Zeitpunkt 01.01.1970 00:00 Uhr

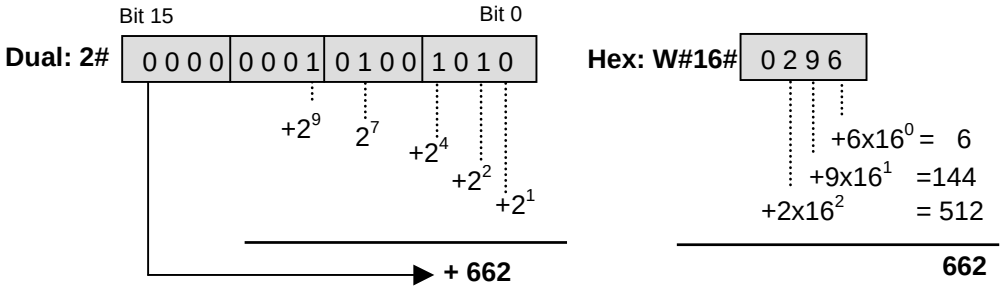
- **Intern werden alle Zeitdaten-Typen wie DWORD behandelt!**

Erläuterungen zu den Datentypen INT, DINT und REAL

- Die Inhalte von Ganzahlen im Umfang von Byte, Wort und Doppelwort können im **Dezimal-Code**, **Dual-Code oder Hexadezimal-Code** angegeben werden.
- Das Automatisierungsgerät arbeitet grundsätzlich nur auf Basis des Dualcodes. Aus ihm werden alle anderen Codes abgeleitet. Diese sind zumeist für Darstellungen am Bildschirm oder in Geräten erforderlich.
- Im Hexa-Code wird jeweils eine Tetrade (4 Bit) zu einer Hexaziffer zusammengefasst.
- Bei INT-Datentypen gilt: Das höchstwertige Bit enthält das Vorzeichen:
  Wert 0 = positiv, Wert 1 = negativ
- Negative Zahlen werden im Zweierkomplement angegeben. Zweierkomplemente ermittelt man durch Umkehr aller Bitwerte und Addition eines Bitwertes, wobei Überträge zu berücksichtigen sind.
- REAL werden nicht mit Komma, sondern mit Punkt eingegeben

Beispiele:

Dezimalzahl 662, darstellbar als INT := 662, 2# 0000_0001_0100_1010 oder W#16# 0296



Die Hexaziffern für die Darstellung von vier Bit:

Tetrade	Dezimal	Hex-Ziffer	Tetrade	Dezimal	Hex-Ziffer
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010	10	A
0011	3	3	1011	11	B
0100	4	4	1100	12	C
0101	5	5	1101	13	D
0110	6	6	1110	14	E
0111	7	7	1111	15	F

Beispiel: 2# 1110_1010_0001_1000_1111_0111_1011_1001

DW#16# E A 1 8 F 7 B 9

Dezimal: 3 927 504 825

Um sowohl sehr kleine Kommazahlen wie z.B. 0.000000000000342 als auch große Kommazahlen wie z.B.1233231456. 012 mit 32 Bit verschlüsseln zu können, wird bei REAL das Prinzip des gleitenden Kommas benutzt. Intern wird dazu die exponentielle Darstellung der Zahlen und eine festgelegte Anzahl Bits jeweils für Mantisse und für Exponent verwendet (**Bild 4-4**).

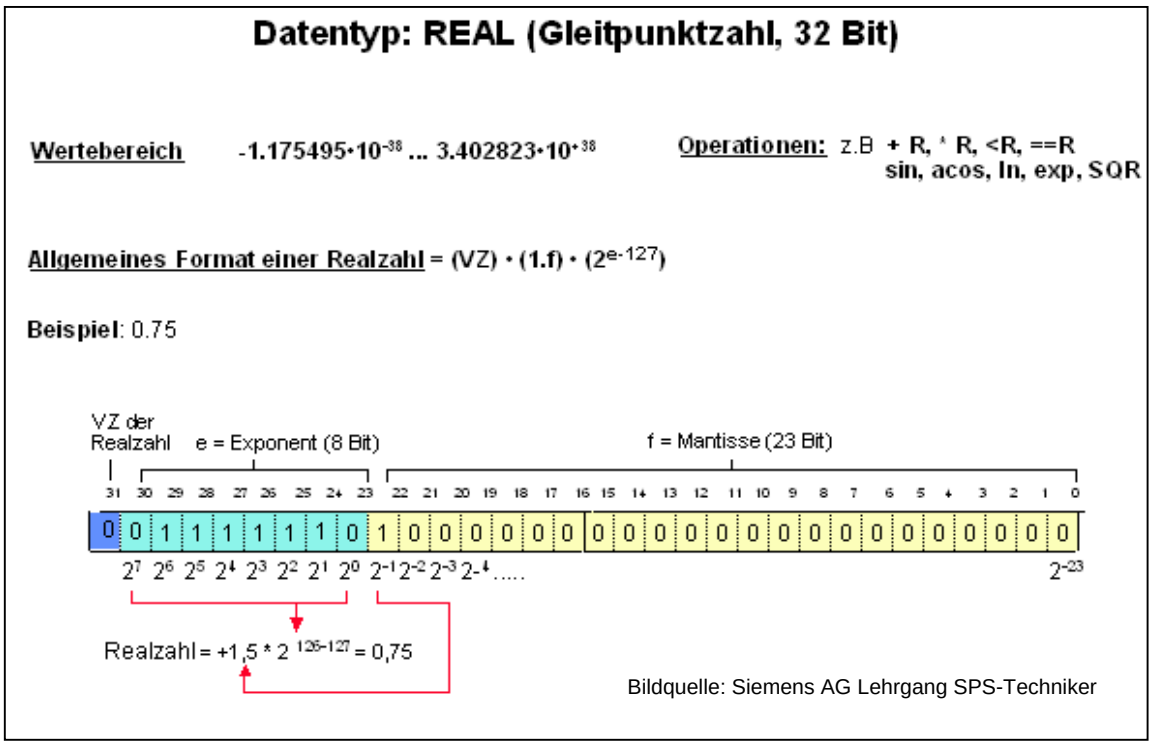


Bild 4-4: Prinzip der Verschlüsselung von 32 Bit-Gleitpunktzahlen

Binär codierte Dezimalzahlen und BCD-Code

Zifferncodierschalter und 7-Segment-Anzeigen enthalten Codewandler, die eine Tetrade in eine Dezimalziffer wandeln. Grundlage für die Darstellung einer Dezimalziffer durch vier Bit ist der Dualcode.

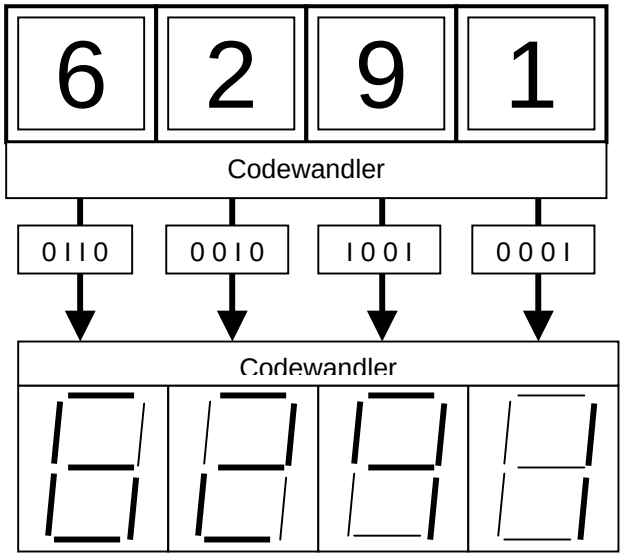
Der BCD Code ist ein Code, bei dem eine Dezimalzahl nicht als Ganzes in den Dualcode überführt wird, sondern jede einzelne Dezimalziffer für sich.

Dezimalziffer	Dualcode der Dezimalziffer = BCD		Hexacode
0	2# 0000		16# 0
1	2# 0001		16# 1
2	2# 0010		16# 2
3	2# 0011		16# 3
4	2# 0100		16# 4
5	2# 0101		16# 5
6	2# 0110		16# 6
7	2# 0111		16# 7
8	2# 1000		16# 8
9	2# 1001		16# 9
Weiter verfügbare Tetraden	2# 1010	**In BCD nicht erlaubt!**	16# A
	2# 1011		16# B
	2# 1100		16# C
	2# 1101		16# D
	2# 1110		16# E
	2# 1111		16# F

Erkenntnis:
Im Bereich der Dezimalziffern 0..9 sind BCD- und Hexacode formal gleich.

Die Hexa-Ziffern A..F sind im BCD-Code nicht erlaubt!
Bei SimaticS7 wirken fehlerhafte BCD-Tetraden, die zu Ziffern A..F führen würden, als Stopp-Fehler!)

Beispiel:



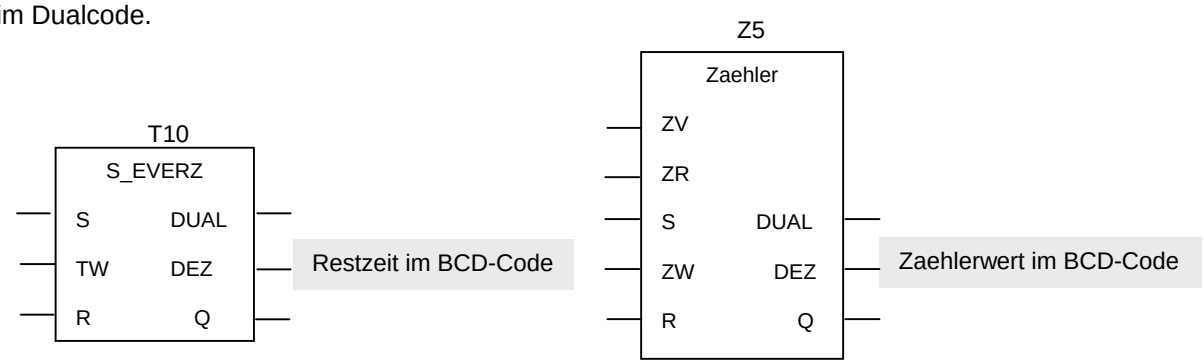
BCD-Code,
dargestellt durch vier Tetraden:

0 I I 0_0 0 I 0_I 0 0 I_0 0 0 I

Im Gegensatz dazu ist der Dualcode von 6291:

2# 0 0 0 I_I 00 0_I 0 0 I_ 0 0 I I

Bei den speziellen Simatic S5–Timern und -Zählern – die auch in Step7 weiter verwendet werden -, liefern die Ausgänge „DEZ“ Inhalte im BCD-Code. Die Ausgänge „DUAL“ liefern die gleichen Inhalte im Dualcode.



So wie zunehmend Codierschalter und 7-Segment-Anzeigen durch Touch Panels ersetzt werden, verliert der BCD-Code an Bedeutung.

- **BCD-Datentypen und Zählerdatentypen sind in IEC 61131-3 nicht explizit definiert!**

An dieser Stelle sei auf Möglichkeiten der Konvertierung von Datenformaten nach IEC 61131-3 verwiesen. Konvertierungen benötigt man bei der Weiterverarbeitung von Daten, wenn formale Typkonflikte auftreten. Mit einer Konvertierung TIME_TO_DWORD ist es beispielsweise möglich, einen allgemeinen Wert TIME in einen Dezimalwert von Millisekunden zu überführen. Datenkonvertierung wird im Abschnitt 7 behandelt.

- **Ansprechen einzelner Bit’s in Variablen:**

In Programmiersystemen nach IEC 61131-3 ist es sehr einfach, einzelne Bit’s in Variablen der Typen BYTE, WORD, DWORD, LWORD anzusprechen: Man benutzt dazu den Bezeichner und – getrennt durch einen Punkt – die Bitadresse.

Beispiel: Wortvariable:WORD;
Zugriff auf Bit 0 .. 15: Wortvariable.0 ... Wortvariable.15

4.1.5 Die Verwendung von Bibliotheken

Eine Vielzahl oft benötigter Funktionen wie z.B. Timer (Zeitglieder), Counter (Zähler), Flankenauswertung (Trigger) oder RS-Flip-Flop (Bistabile Kippstufen) liegt als fertiges Programmelement in Bibliotheken bereit. Der Form nach sind es parametrierbare Bausteine, die ihre aktuellen Variablen durch Parameterübergabe mit Variablen der Typen VAR_INPUT, VAR_OUTPUT und VAR_IN_OUT erhalten.

Die wichtigste Bibliothek ist die Standardbibliothek (Standard Library), die nach IEC 1131-3 allen Programmiersystemen beigefügt werden muss.
Im Programmiersystem CoDeSys ist diese Bibliothek bereits in alle Projekte eingebunden, so dass ihre Programmelemente als Standard-Funktionsblöcke genutzt werden können.

Bild 4-5 zeigt einen solchen Fall: In dem in der graphischen Sprache FUP geschriebenen Programm wird eine positive Flankenauswertung benötigt. Nach Einfügen eines Blockes in das Programm wird entweder das richtige Schlüsselwort eingetragen (in diesem Falle R_TRIG), oder es wird mit Hilfe der Einfügetaste F2 aus dem Angebot der Standard-Funktionsbausteine ausgewählt.

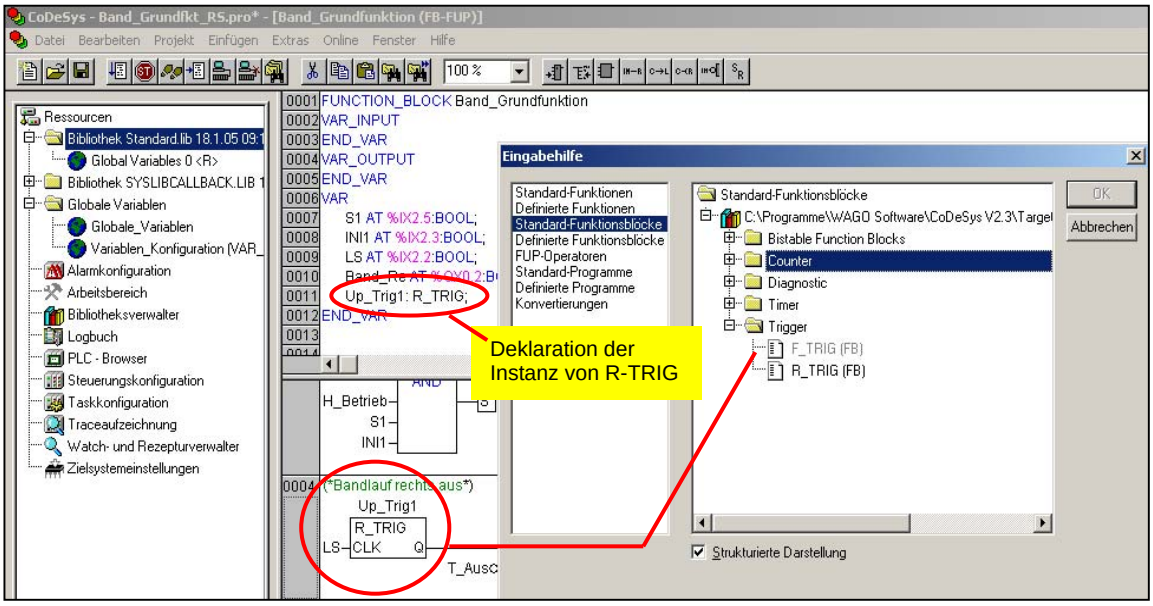


Bild 4-5 : Die Verwendung von Funktionsbausteinen der Standard-Bibliothek

Da Funktionsbausteine (FB) prinzipiell instanziert werden müssen, trifft dies auch für FB's aus der Standardbibliothek zu. Sie erhalten einen Namen (den Instanznamen oder einfacher „die Instanz") und werden mit diesem Namen im Programmkopf nach dem Schema <Name:Typ;> deklariert (im Beispiel Up_Trig1:R_TRIG;).

Schreibt man selbst FB's, so würden diese im obigen Menu unter „Definierte Funktionsblöcke" erscheinen. Ausführlich wird die Arbeit mit FB's und ihre Instanzierung in Abschnitt 9 beschrieben.

Neben der Standardbibliothek gibt es eine Vielzahl weiterer Bibliotheken. Sie stehen auf Homepages von Software-Entwicklern (z.B. www.3S-Software.com) oder Hardware-Anbietern (z.B. www.wago.com) bereit. Sind die Files auf dem Rechner vorhanden, werden sie bei Bedarf in das Projekt eingefügt. Danach können ihre Bausteine genutzt werden.

Bild 4-6 zeigt die Vorgehensweise bei CoDeSys: In der Mitte sind die standardmäßig vorhandenen Bibliotheken sichtbar, rechts verfügbare Zusatzbibliotheken. Mit den Menu's ->Fenster -> Bibliotheksverwaltung und -> Einfügen -> Neue Bibliothek kann man aus diesem Fenster auswählen und einfügen.

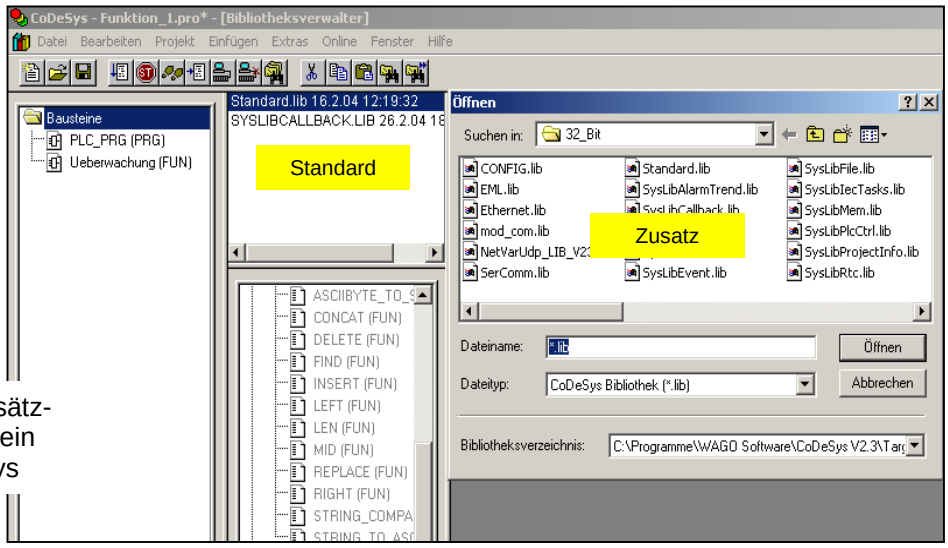


Bild 4-6: Einfügen zusätzlicher Bibliotheken in ein Projekt unter CoDeSys

4.1.6 Online-Hilfe als Lehrmittel

Das beste Lehrbuch für das System CoDeSys ist die Online-Hilfe. **Bild 4-7** zeigt die Übersicht der Themen und als Beispiel Erläuterungen zum Befehl CAL.

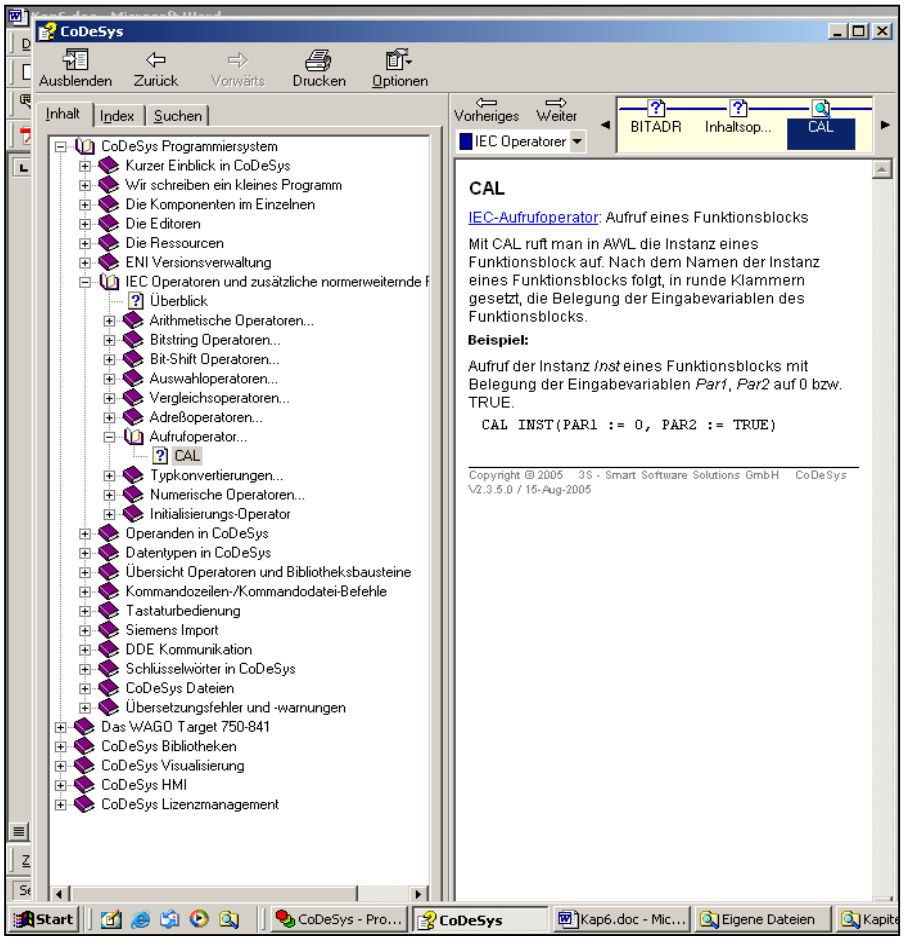


Bild 4-7: Übersicht der Themen in der Online Hilfe

Die Online-Hilfe ist unverzichtbarer Bestandteil zeitgemäßer Programmiersysteme. Beim Programmieren kann nach dem Markieren fraglicher Blöcke mit der Taste F1 konkrete Hilfe abgerufen werden. (Beispiel **Bild 4-8**: Hilfe zum Counter Typ CTUD). Sehr praktikabel ist die Auswahlliste verwandter Themen am oberen Bildrand (**Bild 4-9**).

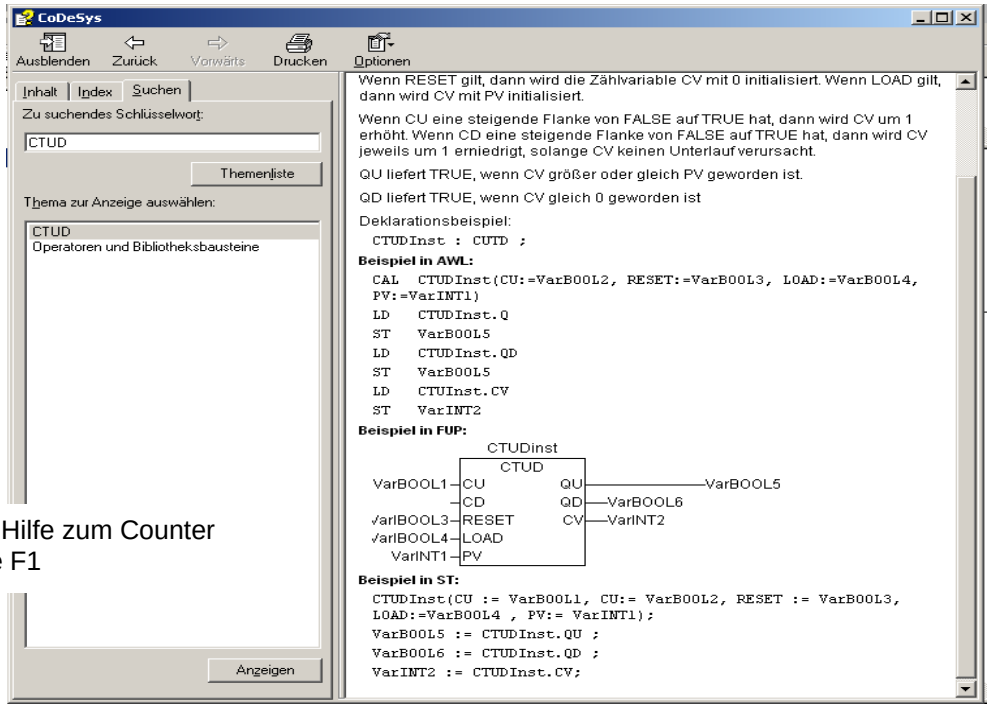


Bild 4-8: Online Hilfe zum Counter CTUD mit Taste F1

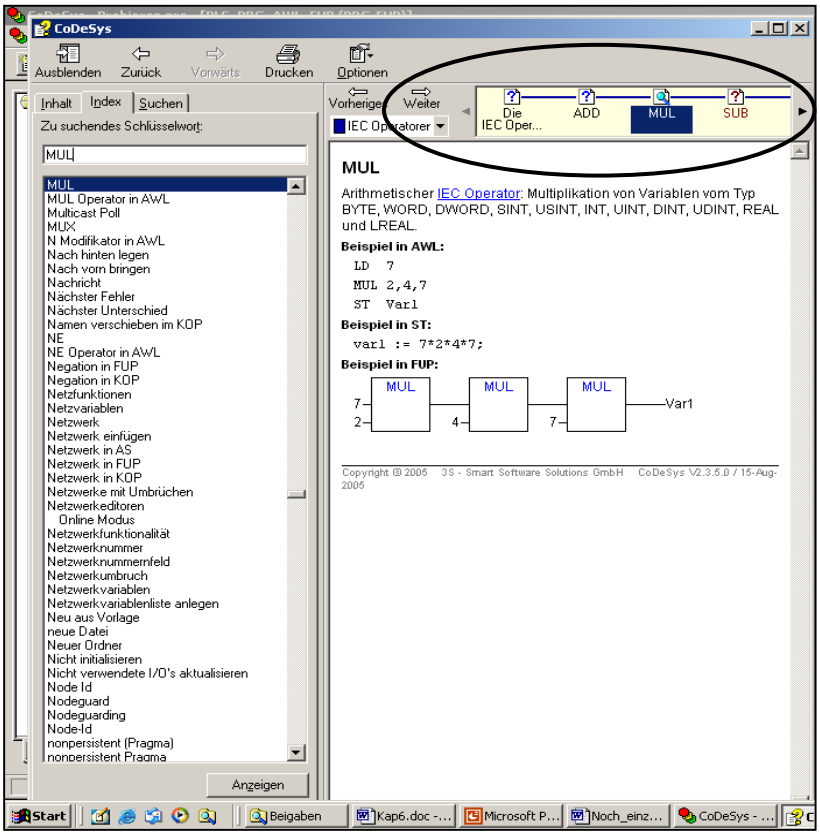


Bild 4-9: Benutzung der oben sichtbaren Auswahlliste für artverwandte Themen in der Online Hilfe

4.1.7 Integrierte Visualisierung

Gemeinsam mit der Simulation der Programm ist auch die in CoDeSys integrierte Visualisierung sehr hilfreich für den Lernenden. Mit einer im Vergleich zu großen Visualisierungstools schnell erstellten einfachen Visualisierung können bequem Programme getestet werden, ohne auf Hardware angewiesen zu sein! Einzelheiten zur Visualisierung sind Folge 12 der „Praktischen Einführung in CoDeSys“ zu entnehmen.

-> Praktische Einführung in CoDeSys Folge 12