Skript zur Vorlesung

VL 2

Angewandte Automatisierungstechnik II

Wirtschaftsingenieurwesen (D) FW

Inhaltsverzeichnis

3 Aufbau und Strukturen industrieller Steuerungssysteme

3.1 Aufbau einer SPS

3.1.1 Programmiergerät (PG)

3.1.2 Anzeige- und Bedienkomponenten (ABK)

3.2 SPS-Arten

3.2.1 Hardware-SPS

3.2.2 Slot-SPS

3.2.3 Soft-SPS

3.2.4 Vor- und Nachteile PC-basierter SPSen

3.3 Informationsverarbeitung in der SPS

3.4 Konventionelle Anbindung der Feldgeräte

3.4.1 Binäre Eingänge der SPS

3.4.2 Binäre Ausgänge der SPS

3.4.3 Analoge Eingänge der SPS

3.4.4 Analoge Ausgänge der SPS

3.5 Busankopplung der Feldgeräte

3.5.1 Feldbussysteme und -strukturen

3.5.2 Datenübertragung in Feldbussystemen

3.6 Bedienen und Beobachten

3.6.1 Prozessvisualisierung

3.6.2 Kopplung zwischen Visualisierungssystemen und SPSen

3.6.3 Prozessleitsysteme

4 SPS-Programmierung nach IEC 61131

4.1 Das Softwaremodell

4.1.1 Steuerungskonfiguration und Ressourcen

4.1.2 Tasks

4.1.3 Programmorganisationseinheiten

4.1.3.1 Funktionen

4.1.3.2 Funktionsbausteine

4.1.4 Variablen

4.2 Das Kommunikationsmodell

4.2.1 Datenaustausch innerhalb eines Programms

4.2.2 Datenaustausch zwischen Programmen

4.3 Das Programmiermodell

4.3.1 Programmiersprachen

4.3.1.1 Textsprachen

4.3.1.2 Grafische Sprachen

4.3.1.3 Empfehlungen für den Einsatz der SPS-Programmiersprachen

4.3.2 Anwender-Datentypen

4.3.3 Anwender-Funktionsbausteine

4.4 Leitfaden für die Softwarestrukturierung

4.4.1 Objektorientierte Softwarestrukturierung

4.4.2 Vorgehensweise bei der strukturierten Programmierung

Zusätzliche Informationen:

Dieses Skript können Sie von der Webseite http://htw-berlin.applicad-atit.de herunter laden.

Dort finden Sie im Verlaufe dieser Veranstaltung weitere Informationen, Hinweise und Aufgaben.

3 Aufbau und Strukturen industrieller Steuerungssysteme

Den heutigen Einsatz industrieller Steuerungssysteme prägen drei wesentliche Entwicklungen:

- **Industrie-PCs** dienen als Standardplattform nicht nur für Programmiersysteme, sondern immer häufiger auch in Form von *Soft-SPS*en als Steuerungen,
- der **Windows-Standard** bietet *offene Schnittstellen* zu anderen PC-basierten Systemen und ermöglicht die Nutzung von Internet und Mobilfunk mit typischen Anwendungen, wie Fernwartung, Fernüberwachung, E-Business u.a.,
- Teile der Steuerungssoftware laufen in **intelligenten Sensoren und Aktoren** ab, was zu einer Dezentralisierung der Automatisierungseinheiten führt.

3.1 Aufbau einer SPS

Eine klassische SPS besteht aus den in Abb. 03.01 dargestellten Hardwaremodulen. Die Stromversorgungsbaugruppe PS (**P**ower **S**upply) wandelt die Netzspannung in eine 24V-Gleichspannung (24 VDC) um, mit der die gesamte Elektronik der SPS versorgt wird. Werden innerhalb der SPS-Baugruppen noch andere Spannungen benötigt (z.B. +5VDC, +/-12VDC), werden diese aus der 24V-Gleichspannung erzeugt.

Die Zentralbaugruppe oder CPU (**C**entral **P**rocessing **U**nit) mit einem Mikroprozessor (µP) zum Ausführen des Programms bildet das Kernstück der SPS. Die aktuell abzuarbeitetenden Programme stehen online im RAM (**R**andom **A**ccess **M**emory). Außer den laufenden Anwenderprogrammen und den Systemprogrammen des Betriebssystems werden im RAM auch die von den Programmen benötigten Variablen und Parameter (Zeiten, Zählerstände, Soll-/Istwerte) gespeichert. Achtung! Der Speicherinhalt des RAMs geht bei Spannungsausfall verloren. Ein RAM ist ein flüchtiger Speicher.

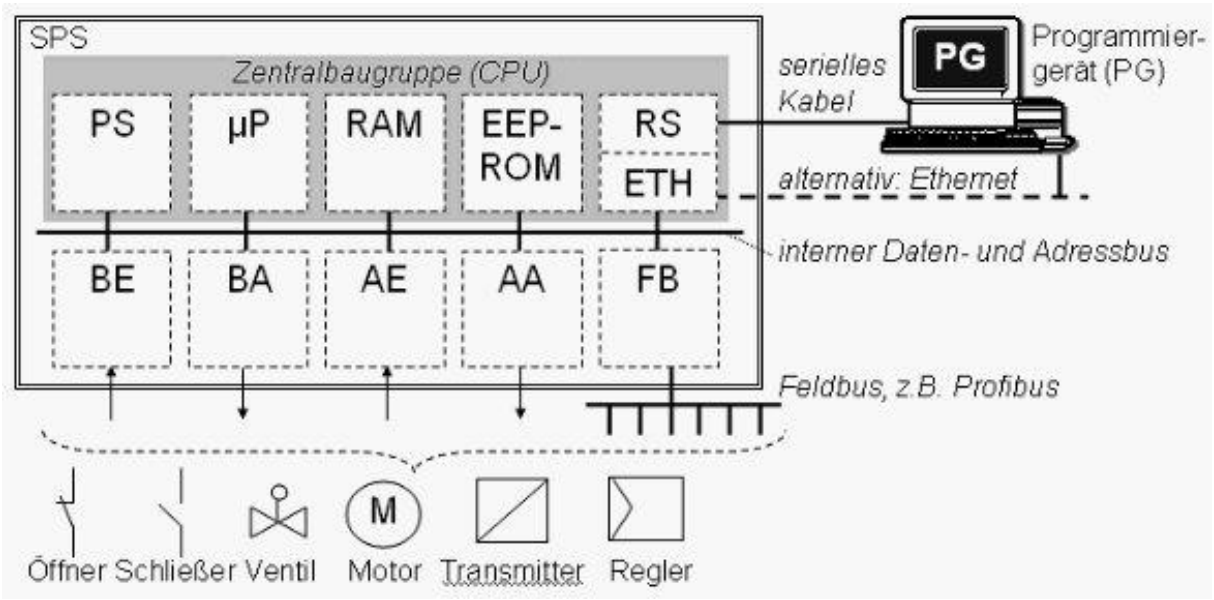


Abb. 03.01 Hardwareaufbau einer SPS mit Stromversorgung (PS), µP, RAM, EEPROM, serieller (RS) bzw. Ethernet-Schnittstelle (ETH) auf einer Zentralbaugruppe und binären Ein- (BE) und Ausgabebaugruppen (BA) sowie analogen Ein- (AE) und Ausgabebaugruppen (AA) und Feldbusschnittstellen (FB).

Die SPS besitzt ein EEPROM (**E**lectrically **E**rasable **P**rogrammable **R**ead **O**nly **M**emory), in dem alle Anwender- und Systemprogramme offline gespeichert werden können. Der EEPROM ist häufig als gesteckte Memory-Card realisiert. In modernen SPSen werden inzwischen Flash-Speicherkarten (CF, SD) verwendet.

Eine weitere Besonderheit einer SPS sind die *Ein-/Ausgabebaugruppen* zum Einlesen von Sensorinformationen und zum Ausgeben von Befehlen an die Aktoren. Hierbei werden die binären bzw. analogen Datenpunkte konventionell über je eine Anschlussleitung an die Ein-/Ausgabebaugruppen angeschlossen.

Busfähige Sensoren und Aktoren werden über Feldbusschnittstellen-Baugruppen an die SPS angeschlossen.

Neben *internen Daten- und Adressbussen* zum Datenaustausch zwischen den Hardwaremodulen kann eine SPS über *serielle* oder *Ethernet-Schnittstellen* an ein Programmiergerät (PG) angekoppelt werden.

Heute wird Ethernet oder besser Industriell Ethernet häufig zur Kopplung von SPSen untereinander und an ein Visualisierungssystem zum Bedienen und Beobachten verwendet.

3.1.1 Programmiergerät (PG)

Das Programmiergerät war früher ein spezieller, sehr robuster und tragbarer PC, der oft mit spezieller Betriebssoftware ausgestattet, allein dem Zweck der Programmierung und Inbetriebnahme/Fehlersuche an einer SPS diente. Heute wird ein Standard-PC oder ein Laptop/Notebook verwendet. Auf diesen Systemen muss dann die für die jeweilige SPS erforderliche Software installiert und eingerichtet werden. Mitunter werden bei einigen SPSen noch serielle Schnittstellen (COM) für die Verbindung zwischen PC/Notebook und SPS benötigt. Die meisten neueren Systeme können aber auch über USB bzw. Ethernet programmiert werden.

Das *Laden der Programme* in die SPS verursacht, dass alle laufenden Programme abgebrochen und die Variablen neu initialisiert werden. Diese Arbeiten sollten also nur von befugtem Personal ausgeführt werden. Manche Steuerungen werden durch spezielle Hardware-Schlüssel oder Passwort vor einem versehentlichen Überschreiben des Programms geschützt.

Bei Wartungsarbeiten an der Anlage, die durch eine SPS gesteuert wird, kann durch die Betriebsart STOP der SPS verhindert werden, dass Signale über die Ausgänge der SPS an die Anlage (z.B. Motor –M01 „EIN“) ausgegeben werden. Für die Einhaltung der Maschinenrichtlinie reicht diese Maßnahme nicht aus. Entsprechende Anlagenteile, die Gefährdungen verursachen können, müssen über Reparaturschalter verfügen. Diese unterbrechen die Ansteuerung durch die SPS im Wartungsfall.

Programme sollten vor Inbetriebnahme an einer realen Anlage in wesentlichen Teilen simuliert werden. Viele Programmiersysteme unterstützen diese Funktion entweder als integraler Bestandteil (z.B. Mitsubishi IEC-Developer) der als zusätzlich zu erwerbenden Software-Modul (z.B. Siemens SIMATIC Step 7).

3.1.2 Anzeige- und Bedienkomponente (ABK)

Auf entsprechenden PCs (Leitrechner) laufen bestimmte Programme (sog. OPC- oder DDE-Server), die die SPS-Daten für andere Anwendungen (z.B. Visualisierungsprogramme) zur

Verfügung stellen. Insbesondere für *Anzeige- und Bedienkomponenten (ABKen)* sind die Daten aus der SPS wichtig, um Anlagenzustände anzuzeigen und Bedieneingriffe wirksam zu machen.

Der Aufbau einer SPS ist demnach von den Feldgeräten ebenso wenig zu trennen wie von Programmiergerät und Anzeige- und Bedienkomponenten. Im Zuge immer komplexerer Anwendungen entstehen dabei Systemstrukturen wie in Abb. 03.02 dargestellt.

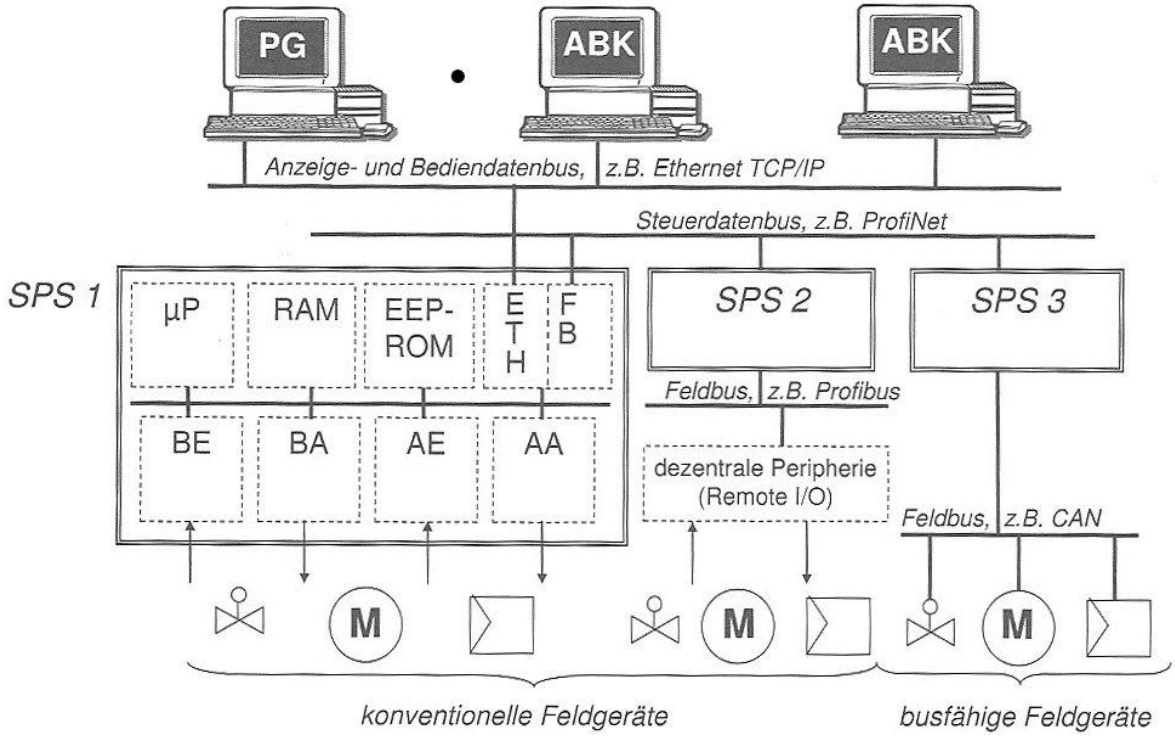


Abb. 03.02 Struktur eines modernen Steuerungssystems mit Feldbustechnik oder Remote-I/O, mehreren SPSen sowie Anzeige- und Bedienkomponenten (ABK).

Diese bestehen beispielsweise aus mehreren SPSen, die über einen Steuerdatenbus kommunizieren können. Das Programmiergerät kann anstatt über die serielle Schnittstelle auch über eine Busschnittstelle mit an den Anzeige- und Bediendatenbus angekoppelt werden und dadurch ebenso wie die Anzeige- und Bedienkomponenten mit den SPSen Daten austauschen. Die Feldgeräte könne auf unterschiedliche Weise an die SPS angekoppelt werden:

- konventionell durch *Kupferdrahtleitungen* (siehe Abb. 03.02 links)
- über *Feldbus*, was natürlich busfähige Feldgeräte erfordert (siehe Abb. 03.02 rechts)
- über eine dezentrale Peripherie (*Remote-I/O*), die einerseits über Feldbus mit der SPS verbunden ist und andererseits nicht busfähige Feldgeräte über dezentrale E/A-Baugruppen anbindet (siehe Abb. 03.02 Mitte)

3.2 SPS - Arten

Man unterscheidet heute drei verschiedene Bauarten von SPSen:

- *Hardware* – SPS,
- *Slot* – SPS und
- *Soft* – SPS .

3.2.1 Hardware - SPS

Der im vorigen Abschnitt beschriebene Aufbau einer SPS bezieht sich auf den klassischen Aufbau einer *Hardware*-SPS. Eine Hardware-SPS ist kompakt (im Wesentlichen aus einer Einheit bestehend) oder modular (aus unterschiedlichen Baugruppen bestehend, die entsprechend den Anforderungen zusammengestellt werden) aufgebaut.

Eine Hardware-SPS bedarf eines externen PCs als Programmiergerät.

3.2.2 Slot - SPS

Eine *Slot*-SPS ist eine Einsteckkarte für den PC, die alle Module einer SPS enthält. Anstatt einer CPU besitzt sie einen Co-Prozessor, auf dem ein eigenes multitaskingfähiges Betriebssystem mit einem multi-ported RAM (von PC und SPS geteilter Speicher, auf den beide zugreifen können) läuft.

Desweiteren befindet sich auf der Slot-SPS mindestens eine Feldbusankopplung zur Anbindung der Sensoren und Aktoren.

Im Grunde genommen benutzt die Slot-SPS lediglich die Stromversorgung des PCs. Mit Hilfe des multi-ported RAM können die CPU des PCs und der Co-Prozessor der Slot-SPS Daten untereinander austauschen.

3.2.3 Soft - SPS

Eine *Soft*-SPS ist reine Software. Sie läuft komplett auf der CPU des Host-PCs. Die Soft-SPS nutzt die Hardware des Host-PCs. Die Anbindung der Sensoren und Aktoren erfolgt wie bei der Slot-SPS über eine Feldbusankopplung, heute häufig über Ethernet.

3.2.4 Vor- und Nachteile PC-basierter SPSen

Die Vorteile der SPS im PC ergeben sich hauptsächlich dadurch, dass die rasante Entwicklung der PC-Leistung für SPSen genutzt werden kann:

- PC-basierte SPSen erreichen *höhere Verarbeitungsgeschwindigkeiten* als Hardware-SPSen.
- Ein PC kann zur *Steuerung, Programmierung und Visualisierung* verwendet werden. Somit ergeben sich preisgünstigere, einfachere und durchgängige Systemstrukturen, mit denen der Anwender gewohnt ist umzugehen.
- Es entstehen *offenere Systeme*, weil der Datenaustausch auf einer einheitlichen Plattform am Häufigsten unter Windows standardisiert wird. Somit wird die

Ankopplung von Bedien- und Beobachtungssystemen sowie von übergeordneten Planungssystemen an die SPS vereinfacht.

Neben diesen Vorteilen muss der PC aber auch die *harten, industriellen Anforderungen* erfüllen, wie z.B.:

- hohe Verfügbarkeit der Hardware,
- Robustheit und Echtzeitfähigkeit des Betriebssystems,
- Erfüllung der Kommunikationsstandards für die E/A-Anbindungen,
- Funktionssicherheit, EMV (elektromagnetische Verträglichkeit),
- Unempfindlichkeit gegenüber rauen Umwelteinflüssen (Erschütterungen, Staub, Temperaturbereich) und
- Absicherung gegenüber Stromausfällen (USV).

Insbesondere bei der Soft-SPS besteht die Gefahr, dass die SPS-Programme von anderen im PC ablaufenden Programmen gestört werden könnten, was im industriellen Einsatz nicht toleriert werden kann.

Durch den Einsatz von Industrie-PCs mit Echtzeit-Kernel-Betriebssystemen, die die Abarbeitung der Programme in Echtzeit, also mit determinierten Wartezeiten (Zykluszeiten), gewährleisten, werden die hohen Anforderungen an PC-basierte SPSen zunehmend erfüllt.

Dennoch ist die konventionelle SPS-Hardware für relativ kleine und kleinste Steuerungsaufgaben, insbesondere in der Gebäudeautomatisierung, häufig kostengünstiger, zu mindestens solange keine Visualisierung benötigt wird.

3.3 Informationsverarbeitung in der SPS – Funktion der SPS

Die Informationsverarbeitung in einer SPS verläuft zyklisch, also immer wiederkehrend. Das SPS-Programm wird zyklisch abgearbeitet. Auch hier findet sich das aus der Informatik bekannte EVA-Prinzip wieder (**E**ingabe, **V**erarbeitung, **A**usgabe).

In Abb. 03.03 wird das Schema der Signalverarbeitung in einer SPS veranschaulicht.

Dabei beschreibt das EVA-Prinzip:

- das Einlesen der Sensordaten bzw. Eingabedaten (Datenpunkte aus der Anlage),
- das Verarbeiten der Eingabedaten und weiterer Informationen im SPS-Programm und
- das Ausgeben der Stellsignale an die Aktoren bzw. das Ausgeben der Ausgabedaten an die Anlage.

Die Eingabedaten (z.B. Messsignale von Sensoren oder Binärsignale von Endschaltern) werden in den Eingangs-Baugruppen elektronisch angepasst. Die CPU fragt nacheinander alle Eingabe-Baugruppen ab und legt die Eingangsdaten im Arbeitsspeicher (RAM) ab. Dieser Speicherbereich befindet sich im PEA-Abbildspeicher (**P**rozess-**E**in-/**A**usgabeabbildspeicher) und wird auch *Eingangsabbild* genannt, weil die hier abgelegten Daten nicht die aktuellen, sondern die zum Abtastzeitpunkt anliegenden Daten sind.

Das SPS-Programm wird dann von der CPU jeweils Schritt für Schritt (Step by Step) abgearbeitet. Dabei werden die im Arbeitsspeicher abgelegten Operationen, wie z.B. LD (LOAD), AND und ST (STORE) einzeln adressiert, interpretiert und mit den angegebenen Operanden (Variablen und Konstanten) ausgeführt. Die Operanden können direkte Adressen

des Ein- oder Ausgangsabbildes sein, sollten aber im Allgemeinen als Variablen (z.B. A, B, C) deklariert werden. Prinzipiell werden in einem Programm nicht nur Daten des Ein- und Ausgangsabbildes, sondern auch Parameter und Zwischenwerte, wie Zählerstände, Zeitwerte, Sollwerte, Zustandswerte u.a. verarbeitet und im RAM gespeichert.

Wenn ein Programm Stellwerte für die Aktoren der Anlage berechnet, werden diese im Ausgangsabbild des PEA abgelegt.

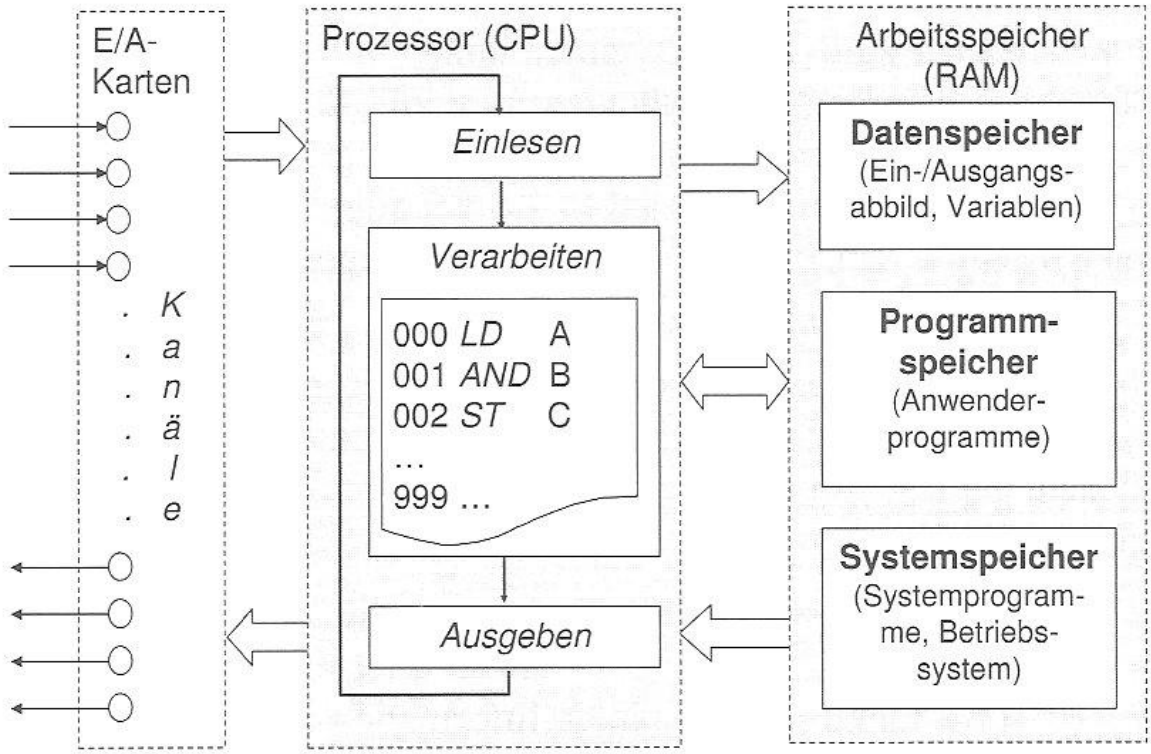


Abb. 03.03 Signalverarbeitung und Arbeitsweise einer SPS.

Erst nach der Abarbeitung des gesamten SPS-Programmes werden die im *Ausgangsabbild* abgelegten Ausgabedaten (Stellwerte) an die Ausgangs-Baugruppen übertragen und stehen erst dann in der Anlage zur Verfügung.

Der Arbeitsspeicher (RAM) lässt sich also in drei Teile gliedern:

- den *Datenspeicher* (PEA) mit Ein- und Ausgangsabbild und den verwendeten Variablen,
- den *Programmspeicher* mit den zyklisch abzuarbeitenden Anwenderprogrammen und
- den *Systemspeicher* mit den SPS-interne Systemprogrammen.

Die folgenden Abschnitte beschreiben wie bei einer SPS die Sensorsignale eingelesen und die Stellsignale ausgegeben werden.

3.4 Konventionelle Anbindung der Feldgeräte

Eine modular aufgebaute Hardware-SPS besitzt aus konventioneller Sicht Baugruppen für *binäre* Ein- und Ausgabesignale (BE bzw. BA) sowie für *analoge* Ein- und Ausgabesignale (AE bzw. AA).

Der Anschluss der Feldgeräte (Sensoren und Aktoren) erfolgt über Kupferleitungen, die mittels Steck- oder Schraubverbindungen an die Kanäle der Ein- oder Ausgangs-Baugruppen angeschlossen werden.

3.4.1 Binäre Eingänge der SPS

Eine binäre Eingangs-Baugruppe unterscheidet zwischen hohen und niedrigen Signalpegel der anliegenden Eingangsspannung. Wie Abb. 03.04 zeigt, wird die Eingangs-Baugruppe von einer 24V-Gleichspannung (DC) versorgt. Hat der binäre Sensor (Niveauschalter) ausgelöst, so ist der Kontakt geschlossen. Es liegt also ein Signalpegel von ca. 24 VDC am Eingang der binären Eingangs-Baugruppe an. Hat der binäre Sensor nicht ausgelöst, so liegt ein Signalpegel von ca. 0 VDC am Eingang der binären Eingangs-Baugruppe an.

Diese beiden Signalpegel werden gleich hinter dem Eingang der binären Eingangs-Baugruppe über ein RC-Filter geleitet, das das binäre Eingangssignal von überlagerten Störsignalen (Rauschen) und Kontaktprellen befreit. Anschließend wird das aus der Anlage kommende Signal über einen Optokoppler geleitet, wodurch eine galvanische Trennung der anlagennahen Signale von den in der SPS verarbeiteten Signale durchgeführt wird, um Störungen bei der Signalverarbeitung innerhalb der SPS zu verhindern.

Nach dem Optokoppler sorgt ein Trigger für die richtige Zuordnung der binären Signale zu den Wahrheitswerten Low (FALSE) und High (TRUE).

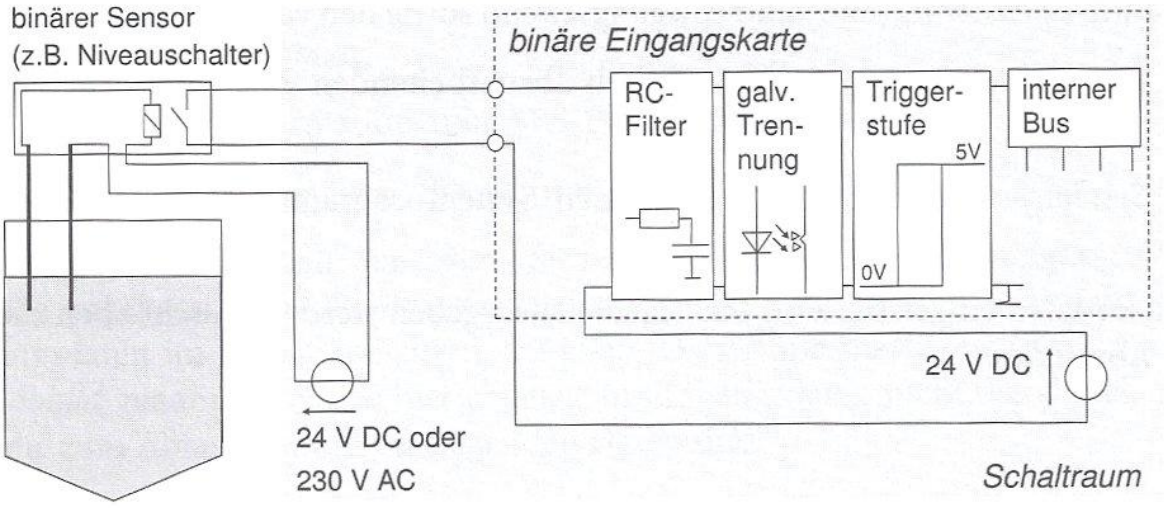


Abb. 03.04 Vereinfachte Darstellung eines Stromlaufplanes zwischen Sensor und binärer Eingangs-Baugruppe.

3.4.2 Binäre Ausgänge der SPS

Binäre Ausgangs-Baugruppen erzeugen aus den Boole'schen Ausgangsvariablen TRUE oder FALSE ein Signals, das über einen Optokoppler an einen binären Schaltverstärker geleitet wird. In vielen Fällen besitzen die binären Ausgangs-Baugruppen Ausgangsrelais, deren Kontakte das Ausgangssignal an die Anlage übergeben. Häufig werden aber auch Schaltverstärker mit Transistoren verwendet, die ein elektronisches Signal an die Anlage ausgeben (siehe Abb. 03.05). Diese Transistorausgänge schalten 24 VDC bei einem Strom von i.d.R. maximal 0,5 A und sind über eine integrierte Freilaufdiode vor Überspannungen beim Anschalten von induktiven Verbrauchern (Relais, Magnetventil) geschützt.

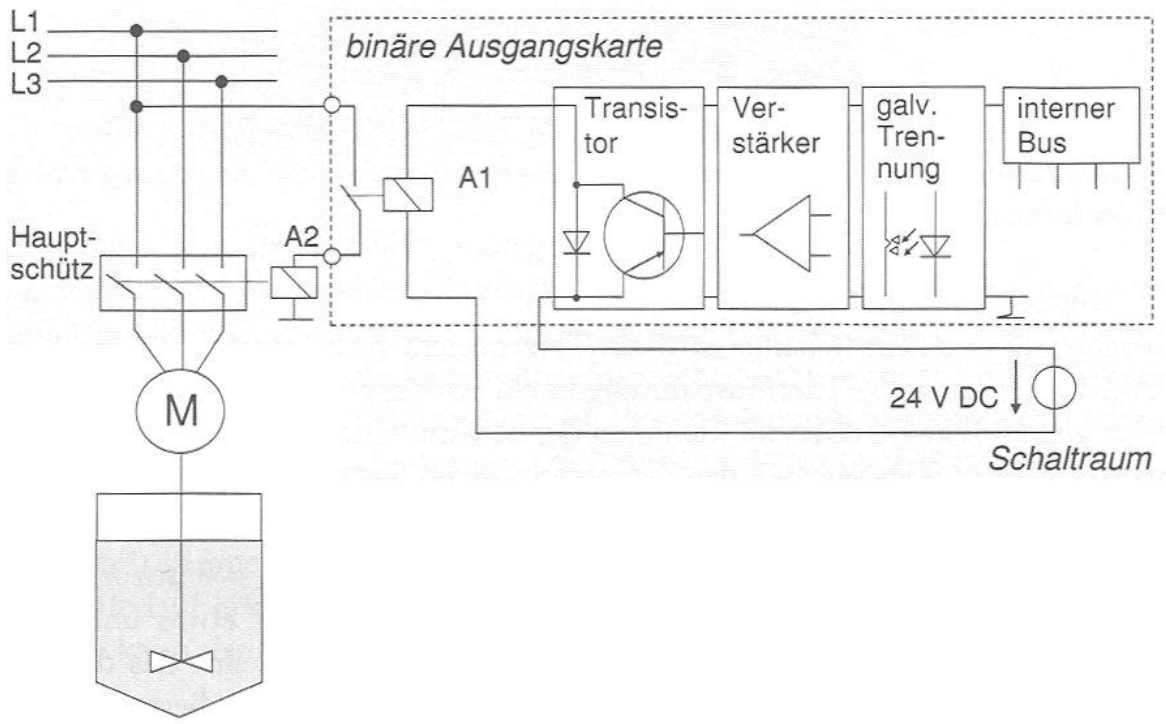


Abb. 03.05 Vereinfachte Darstellung einer Motoransteuerung über eine binärer Ausgangs-Baugruppe.

3.4.3 Analoge Eingänge der SPS

Ein Sensor wandelt die physikalische Messgröße (z.B. Füllstand, Druck, Durchfluss, Temperatur, Drehzahl) entweder in einen elektrischen Strom-, Spannungs- oder Widerstandswert um. Strom- und Spannungsgeber werden im Allgemeinen über zwei Leitungen auf den entsprechenden Kanal einer analogen Eingangs-Baugruppe geführt.

Man unterscheidet aktive und passive Sensoren. Passive Sensoren, sogenannte Zweidraht-Messumformer, werden über die Analogeingangsklemmen von der SPS (24 VDC) gespeist. Abb. 03.06 zeigt das. Aktive Sensoren besitzen dagegen oft eine separate Spannungsversorgung und werden deshalb über eine Vierdraht-Leitung (2 Leitungen für die Stromversorgung, 2 Leitungen für das eigentliche Messsignal) angeschlossen. Moderne aktive Sensoren lassen sich aber z.T. auch über eine Zweidraht-Leitung betreiben, wenn das Messsignal als 4 … 20 mA-Signal übertragen wird. So kann die Elektronik des Sensors aus dem Grundstrom von etwas weniger als 4 mA versorgt werden.

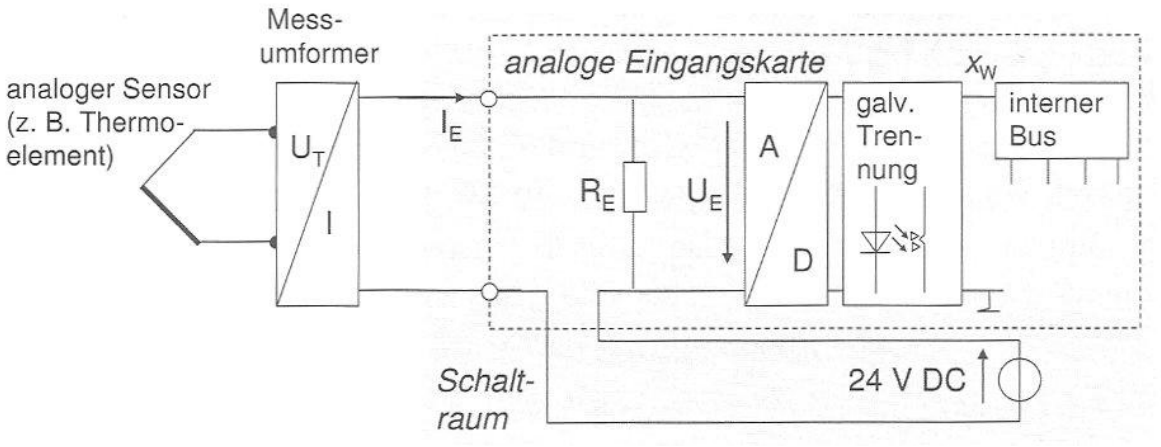


Abb. 03.06 Vereinfachte Darstellung der Ankopplung eines analogen Sensors an eine analoge Eingangs-Baugruppe in Zweileiterschaltung (Zweidraht-Messumformer).

Die analoge Eingangs-Baugruppe führt für jeden Kanal eine Analog-/Digital-Umwandlung durch. Der digitalisierte Messwert x_W wird nach der galvanische Trennung gemäß Tabelle 03.01 als 16-Bit-Datenwort im Eingangsabbild des Arbeitsspeichers (PEA) abgelegt. Je nach Bedarf gibt es Baugruppen mit 10-, 12- oder 16-Bit-Auflösung.

Tabelle 03.01: Umwandlung eines analogen Stromsignals I_E in die digitale Größe x_W, die sowohl dezimal als auch in Form eines 16-Bit-Datenwortes angegeben ist.

I_E	x_W	*Eingangsdatenwort* x_W															
[mA]	*dezimal*	VZ	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
> 22,810	32767	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
22,810	32511	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
20,0005	27649	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	1
20,0	27648	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0
16,0	20736	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0
4,0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3,9995	–1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1,1852	–4864	1	1	1	0	1	1	0	1	0	0	0	0	0	0	0	0
< 1,1852	–32768	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Im vorliegenden Beispiel wird für die Übertragung der analogen Messsignale ein Einheitsstromsignal der Größe 4 … 20 mA verwendet.

Dabei bedeuten die Werte folgendes:

Stromsignal	**Messsignal**	**Beispiel Druck**
4 mA	0 %	0 Pa
12 mA	50 %	125 Pa
20 mA	100%	250 Pa

Ein Bit entspricht einem Strom von **I_E = (20-4) mA / 27648 = 0,58 µA**.

Werte die unterhalb von **I_E = 4 mA** liegen (**x_W < 0**, also **negativ**), lassen auf einen Drahtbruch schließen.

3.4.4 Analoge Ausgänge der SPS

Zur Ansteuerung eines analogen Stellgliedes, wie z.B. eines kontinuierlich verstellbaren Regelventils in Abb. 03.07, gibt die SPS den berechneten analogen Stellwert über eine analoge Ausgangs-Baugruppe aus. Auch hier wird wieder eine galvanische Trennung über Optokoppler und dann eine Digital-/Analog-Umwandlung durchgeführt.

Es entsteht ein analoges Strom- oder Spannungssignal, das an den Aktor geführt wird.

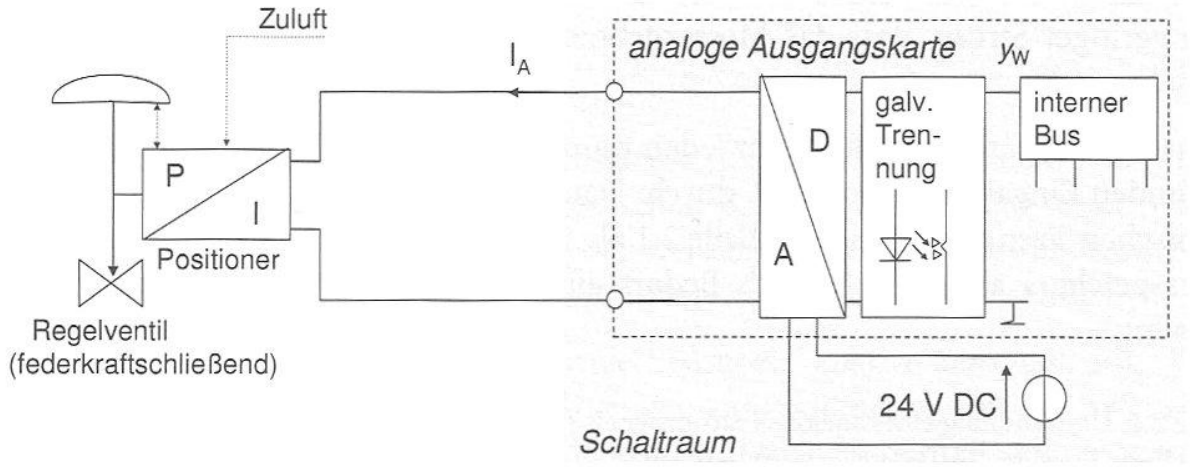


Abb. 03.07 Darstellung der Ankopplung eines Regelventils an eine analoge Ausgangs-Baugruppe.

Die in diesem Abschnitt beschriebenen Möglichkeiten, binäre und analoge Signale in die SPS einzulesen oder von der SPS auszugeben, sind mit einem großen Verkabelungsaufwand verbunden. Für jedes Signal sind mindestens zwei, wenn nicht sogar vier Leitungen, also jeweils ein Kabel zu verlegen.

Damit entsteht ein erheblicher Material- und Verlegungsaufwand, der durch moderne Feldbustechnik wesentlich verringert werden kann. Das soll im folgenden Abschnitt gezeigt werden.

Dennoch wird die klassische Zwei- und Vierleitertechnik nicht verschwinden, da sie für kleine und kompakte Anlagen, aber auch beim Anschluss konventioneller Sensoren und Aktoren an dezentrale Feldbuseinheiten (Remote-I/O), weiterhin eine einfache und zuverlässige Anschlussart darstellt.

3.5 Busankopplung der Feldgeräte

Der Grundgedanke von Feldbussystemen ist es, in erster Linie elektrische Kabel und Leitungen einzusparen.

Mit Feldbussystemen ist es möglich mehrere Signale gleichzeitig über eine einzige Busleitung zu übertragen.

3.5.1 Feldbussysteme und -strukturen

In der klassischen Verdrahtungstechnik muss eine Sortierung der Leitungen von den räumlich benachbarten Sensoren und Aktoren in der Anlage zu den funktional zusammenhängenden Ein- und Ausgabekanälen der Ein- und Ausgangs-Baugruppen der SPS erfolgen.

Wie Abb. 03.08a zeigt, werden räumlich benachbarte Einzelleitungen von den Feldgeräten in einem Feldverteiler zu einem Stammkabel zusammengefasst und in den Schaltraum verlegt. Dort werden die Einzelleitungen in einem Rangierverteiler erneut aufgelegt und geordnet. Diese Umverdrahtung ist notwendig, weil die im Feld räumlich benachbarten Leitungen nun funktional in der Reihenfolge zusammengefasst werden müssen, in der sie mit den E/A-Kanälen der Ein- und Ausgangs-Baugruppen der SPS verbunden werden.

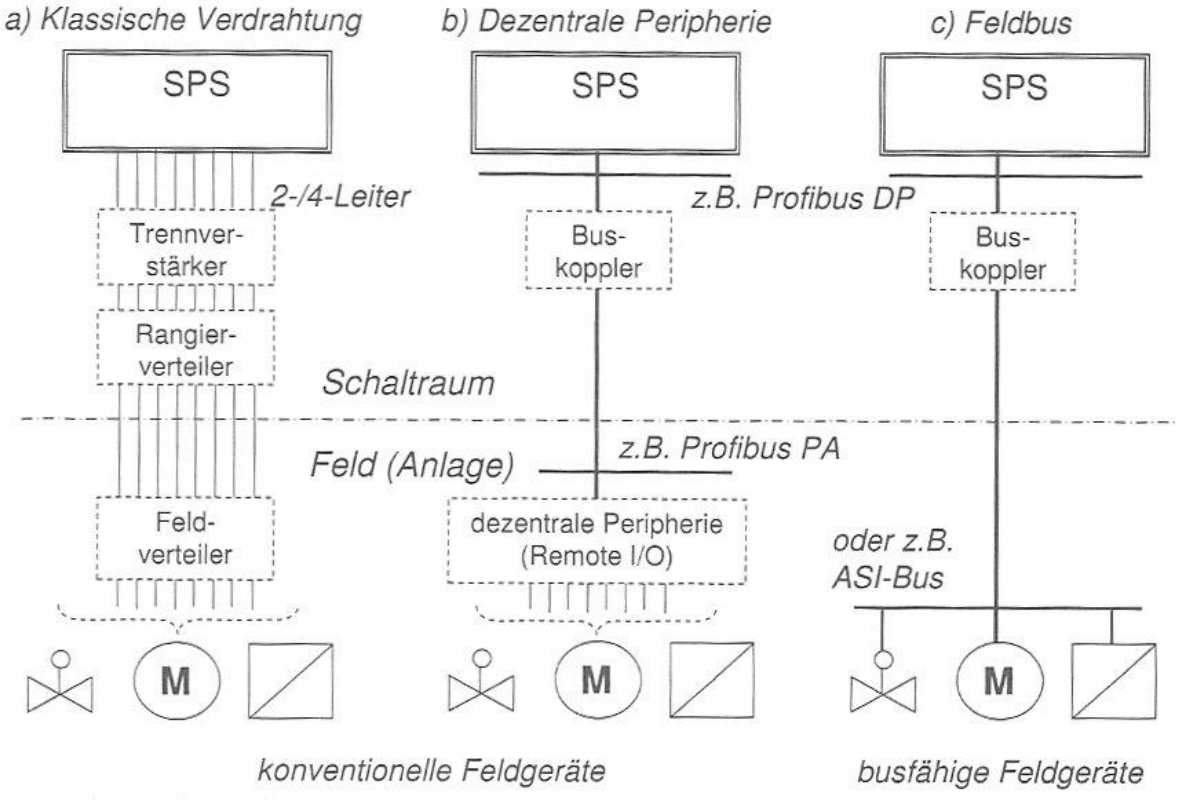


Abb. 03.08 Gegenüberstellung der heutzutage gängigen Verdrahtungskonzepte zur Ankopplung der Sensoren und Aktoren an die SPS.

Mit der in Abb. 03.08c dargestellten Feldbustechnik werden nun Feldgeräte und SPS ausschließlich an eine Busleitung angeschlossen. Auch hier muss ggf. eine Umsetzung zwischen zwei Bussystemen erfolgen.

Voraussetzung für eine Feldbusankopplung ist, dass alle Feldgeräte über einen Mikroprozessor mit busfähiger Schnittstelle verfügen.

Das ist bei den meisten älteren und klassischen Feldgeräten nicht der Fall. Deshalb setzte sich zunächst die in Abb. 03.08b gezeigte „Zwischenlösung“ mit einer Feldgeräteankopplung über einen dezentralen Feldmultiplexer durch. Diese dezentrale Peripherie (DP) ermöglicht es, auch nicht busfähige Feldgeräte anzuschließen, was auch als Remote-I/O-System (RIO) bezeichnet wird.

Das Remote-I/O-System (RIO) verfügt einerseits über die E/A-Baugruppen, die früher zentral im SPS-Schrank steckten und andererseits über eine Feldbusschnittstelle, die eine Busverbindung zur SPS ermöglicht.

Tabelle 03.02: Feldbussysteme und ihre Eigenschaften (Auswahl).

Bussystem	Einsatzfelder	Eigenschaften
ASI-Bus (**A**ktor-**S**ensor-**I**nterface)	Ankopplung einfacher Sensoren und Aktoren, wie Lichtschranken, Ventile, Lampen etc., zumeist in der Fertigungstechnik.	• Kostengünstig. • Schnelle Übertragung weniger binärer Daten. • Übertragung von Energie und Daten über dasselbe Kabel.
Profibus (**Pro**cess **Fi**eld **Bus**)	Profibus-DP für Datenaustausch mit **d**ezentraler **P**eripherie wie Antrieben, Ventilen und Messumformern in Fertigungs- und Verfahrenstechnik.	• Schneller, zyklischer Datenaustausch der SPS mit dezentralen Feldgeräten (Zykluszeit < 10 ms). • Auch Datenaustausch zwischen SPSen im Multi-Masterbetrieb möglich.
	Profibus-PA für Anwendungen in explosionsgefährdeten Umgebungen in der **P**rozessa**a**utomatisierung.	• Vergleichsweise langsame Datenübertragung. • Gewährleistet Eigensicherheit der angekoppelten Sensoren und Aktoren, indem die Versorgungsspannung unterhalb der Zündenergie von explosiven Gasen gehalten wird.
Profinet (**Pro**cess **Fi**eld Ether**Net**)	Profinet-IO (Input Output) zur Ansteuerung von Sensoren und Aktoren durch eine zentrale Steuerung in der Fertigungstechnik.	• Zykluszeiten unter 1 ms im isochronen Modus möglich. • Kompatible Kabel- und Netzwerktechnik mit dem Ethernet der PC-Welt. • Relativ einfache Erweiterbarkeit.
	Profinet-CBA (Component Based Automation) zur Vernetzung von SPSen.	• Wie bei Profinet-IO, Zykluszeiten jedoch im Bereich von 50 ms.
EIB (**E**uropäischer **I**nstallations**b**us)	Kommunikation zwischen Sensoren und Aktoren zur Gebäudeautomatisierung.	• Steuerung erfolgt dabei über den Benutzer selbst oder über einen mit entsprechender Software ausgerüsteten Netzwerkcomputer.
SERCOS (**Se**rial **R**ealtime **Co**mmunication **S**ystem)	Kommunikation zwischen Steuerungen und Antrieben in der Fertigungstechnik.	• Synchronisierung digitaler Servoantriebe hochgenau und in Echtzeit. • Regelung von beispielsweise 40 Achsen mit einer Zykluszeit von 1 ms.
CAN-Bus (**C**ontroller **A**rea **N**etwork)	Vernetzung von Steuergeräten in Automobilen.	• Ausdehnung abhängig von Übertragungsrate 40 m bei 1 Mbit/s, hohe Datensicherheit.

Die wichtigsten Vorteile der Feldbustechnik sind:

- Einsparung von Kabeln zwischen Anlage und Schaltraum,
- höhere Auflösung der Messwerte,
- Übertragung zusätzlicher Geräteinformationen (Grenzwerte, Zustände, Alarme, Betriebszeiten u.a.),
- zentrale Diagnosemöglichkeit (d.h. Fehlersuche von der Leitwarte aus),

- dezentrale Intelligenz (Rechenoperationen, wie z.B. Dosieren im Feldgerät),
- Platzeinsparung im Schaltraum,
- geringerer Montage- und Planungsaufwand,
- geringerer Nachrüst- und Änderungsaufwand und
- zentrale Konfigurierung der Bussysteme am PC.

Derzeit gibt es am Markt verschiedene Feldbussysteme, die sich für verschiedene Industrieanwendungen durchgesetzt haben. Während für die Fertigungsautomatisierung Zykluszeiten von kleiner 10 ms in relativ kleinen Bussegmenten (Kabellängen kleiner 100 m) gefordert werden, ist die Ausdehnung von Chemieanlagen i.d.R. größer, und die Zykluszeitanforderungen liegen dabei im Bereich von 100 ms und sogar darüber.

In Tabelle 03.02 werden die zurzeit gängigsten Feldbussysteme gegenüber gestellt.

Exemplarisch werden im Folgenden die Eigenschaften des Profibus DP beschrieben.

3.5.2 Datenübertragung in Feldbussystemen

Wie erfolgt nun die Datenübertragung in Feldbussystemen? Ähnlich wie beim PC, bei dem man zwischen einer parallelen und seriellen Datenübertragung zu Drucker bzw. Maus und Tastatur unterscheidet, kann man die klassische 2-/4-Leitertechnik als parallele Schnittstelle ansehen, von der aus die Daten für jedes Signal zeitgleich ein- und ausgegeben wird.

Dementsprechend erfolgt die Feldbusankopplung über eine serielle Schnittstelle, denn hier werden die Daten hintereinander über zwei Leitungen (eine zum Senden TxD, eine zum Empfangen RxD) übertragen. Den Vorteil der erheblich kleineren Anzahl von Leitungen erkauft man sich aber mit höheren Übertragungszeiten, denn die Daten der einzelnen Signale müssen nacheinander übertragen und durch eine Start- und Ende-Kennzeichnung voneinander getrennt werden.

Wie in Abb. 03.09 dargestellt, ist die Übertragung mit verschiedenen physikalischen Übertragungsmedien möglich, angefangen von der verdrillten Zweidrahtleitung (Doppelader DA) aus Kupfer, über ein Koaxialkabel, Lichtwellenleiter (LWL), bis hin zu Funkwellen (Bluetooth, WLAN).

Die Wahl des Mediums hängt von den in Abb. 03.09 gegenübergestellten Faktoren ab.

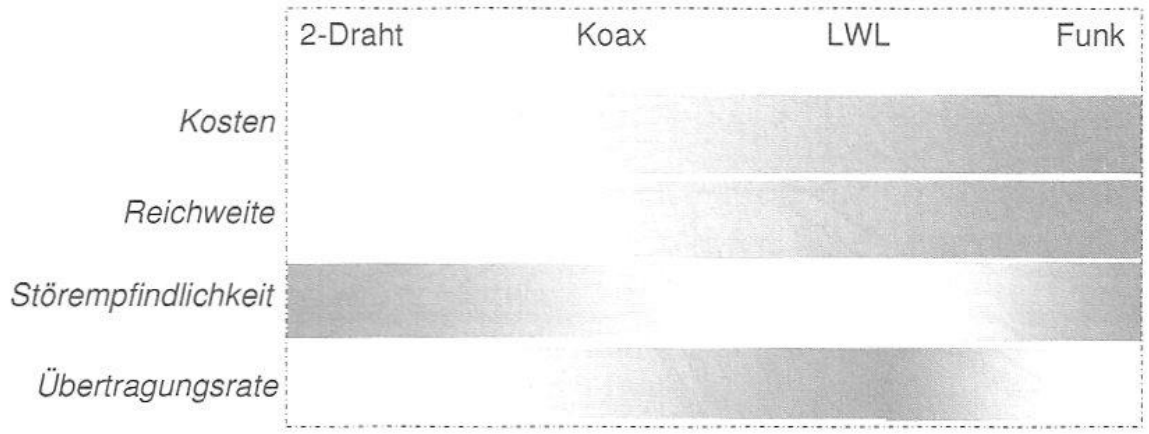


Abb. 03.09 Eigenschaften der Übertragungsmedien (hell bedeutet niedrig, dunkel hoch).

Die elektrischen Eigenschaften der Busleitung sind durch den Schnittstellenstandard RS485 festgelegt. Hiernach kann ein Bussegment bis zu 1200 m für 1 Master und 31 Slaves mit einer Übertragungsrate von 1,5 Mbit/s ausgelegt werden.

Im Gegensatz dazu leistet eine serielle PC-Schnittstelle (COM) nach dem Schnittstellenstandard RS232 (V.24) nur eine maximale Leitungslänge von 15 m bei 1 Master und 1 Slave und einer Übertragungsrate von 19,2 kBit/s.

Die RS485-Schnittstelle ist also busfähig, schneller und von größerer Reichweite.

Damit nun die SPS mit den Feldgeräten Daten austauschen kann, müssen alle Busteilnehmer determiniert auf den Bus zugreifen können. Das für diese Kommunikation erforderliche Buszugriffsverfahren ist das Master-/Slave-Verfahren. Dabei stellt i.d.R. die SPS den Master und die Feldgeräte die Slaves.

Master-/Slave-Verfahren:

Nur der Master darf die Kommunikation anstoßen. Die Slaves antworten nur auf die Anforderungen des Masters.

Damit ergibt sich folgender Kommunikationsablauf:

- Will der Master Daten von einem Slave empfangen, so fordert er die Daten durch ein Request-Telegramm an.
- Dieses Telegramm wird nun auf dem Bus von jedem Slave dahin gehend geprüft, ob die Anfrage an ihn gerichtet ist. Ist das der Fall, handelt es sich bei dem Slave also um den adressierten Slave, so antwortet der Slave mit einem Response-Telegramm, mit dem er die angeforderten Daten an den Master zurücksendet.
- Will der Master Daten an den Slave senden, so schreibt er neben der Zieladresse des anzusprechenden Slaves auch die Sendedaten in das Send-Telegramm.
- Der angesprochene Slave liest die Daten ein und sendet ein Acknowledge-Telegramm an den Master zurück.

Der Aufbau der vier Telegrammtypen umfasst folgende Informationen:

<Startbyte, Quelladresse, Zieladresse, Steuerbyte, Daten, Prüfbyte, Endbyte>

Im Steuerbyte wird unterschieden, ob es sich um ein Send-, Request-, Response- oder Acknowledge-Telegramm handelt.

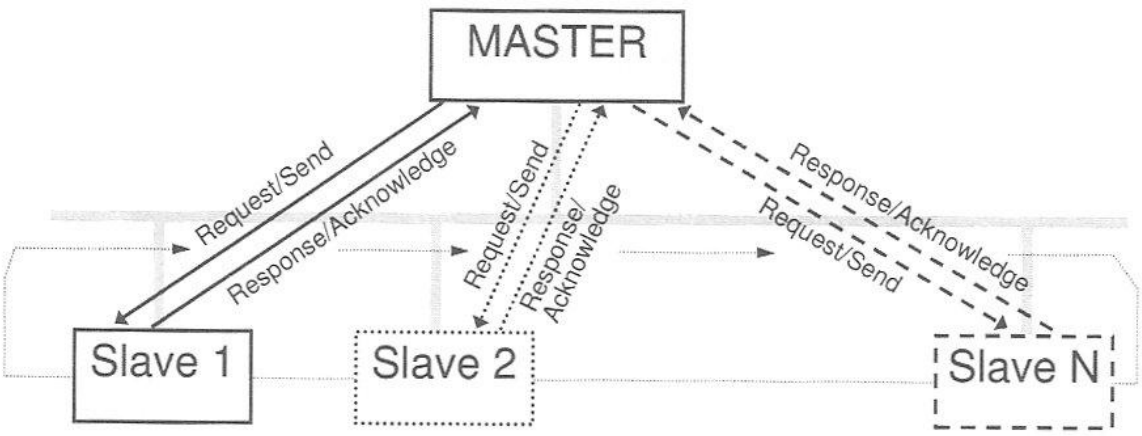


Abb. 03.10 Die SPS kommuniziert in jedem Zyklus nacheinander mit den einzelnen Slaves (Polling).

Da stets nur der angesprochene Slave Daten senden darf, kann erst dann eine neue Abfrage erfolgen, wenn die aktuelle Kommunikation abgeschlossen ist, der Master also die angeforderten Daten erhalten hat.

Die SPS muss also die Feldgeräte nacheinander ansprechen. Wir sprechen daher auch von Polling (to poll = wählen).

Voraussetzung für die Buskommunikation ist, dass alle Geräte und deren Eigenschaften bekannt sind. Vor der Inbetriebnahme muss die Busstruktur konfiguriert werden. Dabei ist festzulegen, welcher Master, welches Bussystem und welche Slaves an der Kommunikation teilnehmen.

Da die Sensoren und Aktoren von verschiedenen Herstellern stammen, muss jeder Hersteller eine Gerätestammdatei (GSD) für sein Feldgerät mitliefern. Diese Datei wird bei der Buskonfiguration eingelesen.

Die GSD beinhaltet charakteristische Gerätemerkmale wie

- Hersteller, Gerätetyp,
- Baudrate (Übertragungsgeschwindigkeit),
- Diagnose-Codes (Meldungstypen),
- Art und Anzahl der Messwerte bzw. Stellwerte (Datenformate u.a.)

3.6 Bedienen und Beobachten

Die Bedienung und Beobachtung des von der SPS automatisierten Prozesses erfolgt häufig noch vor Ort über ein Bedientableau in unmittelbarer Nähe des Prozesses (siehe Abb. 03.11a).

Auf dem Bedientableau befinden sich zum einen Schalter zur Bedienung des Prozesses und zum anderen Leuchten zur Signalisierung bestimmter Prozesszustände. Die Schalter und Leuchten sind dabei als physikalische Ein- und Ausgänge der SPS verdrahtet.

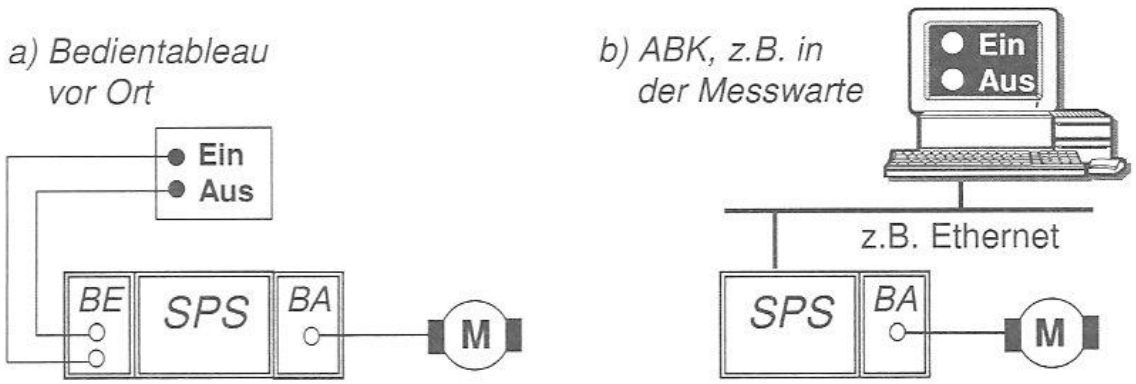


Abb. 03.11 Bedienung eines Motors durch Schalter oder Taster vor Ort (a) und über eine Anzeige- und Bedienkomponente (ABK) aus der Messwarte (b).

Heutzutage werden die meisten Prozesse, Maschinen und Anlagen auch vor Ort durch Anzeige- und Bedienkomponenten (ABKen) bedient und beobachtet. Eine ABK ist ein Rechner mit einer Software zur Prozessvisualisierung. Damit wird es dem Bediener ermöglicht, Prozesszustände auch aus der Ferne zu beobachten und per Mausklick Geräte zu Bedienen (siehe Abb. 03.11b).

3.6.1 Prozessvisualisierung

Ein Prozessvisualisierungssystem besitzt eine spezielle Software, mit der Prozesse schematisch dargestellt und Prozesszustände farblich gekennzeichnet werden. In dem in Abb. 03.12 dargestellten Beispiel ist der Behälter leer, so dass der Niveauschalter LS- farblich als TRUE markiert ist. Das wird dadurch erreicht, dass im Dialog zur Dynamisierung unter Farbwechsel der Name der Variablen in der SPS eingetragen wird, die den Zustand des Sensorsignals angibt.

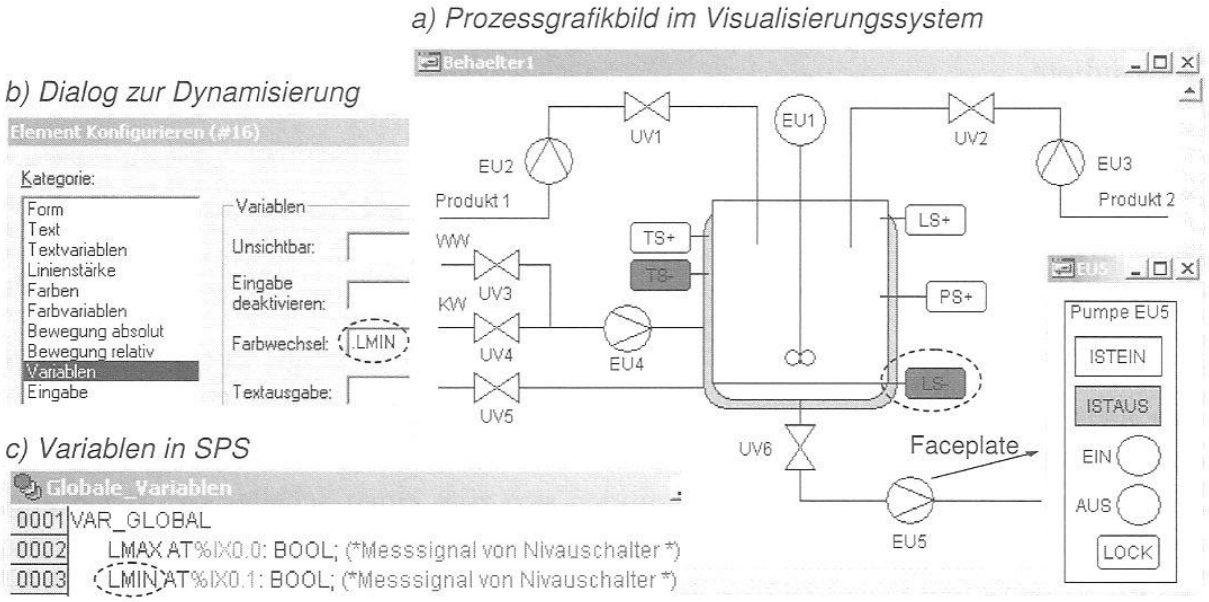


Abb. 03.12 Prozessgrafikbild eines verfahrenstechnischen Prozesses mit dem Visualisierungssystem CoDeSys HMI (a) und Dynamisierung der Messstelle LS- (b) durch die globale Variable LMIN in der SPS (c).

Darüber hinaus kann der Bediener Schalter oder Taster als Soft-Keys betätigen, wodurch eine Aktivierung der SPS-Variablen erfolgt. Per Mausklick auf das Symbol (oder direkt mit dem Finger bei einem Touch-Screen) der Pumpe EU5 lässt sich hierfür das zugehörige Bedienfenster, das sogenannte Faceplate, öffnen, über das der Bediener die Pumpe ein- oder ausschalten kann.

3.6.2 Kopplung zwischen Visualisierungssystemen und SPSen

Für die Kopplung zwischen Visualisierungssystemen und SPSen gibt es wie in Abb. 03.13 dargestellt, verschiedene Konzepte.

Prinzipiell unterscheidet man Visualisierungssysteme, die wie z.B. InTouch (Fa. Wonderware) getrennt von der SPS laufen, und solche, die in das SPS-Programm integriert sind, wie z.B. CoDeSys (Fa. 3S). Wenn diese auf einer PC-basierten SPS implementiert sind, kann die ABK wie in Abb. 03.13a direkt an die Monitorschnittstelle des PCs angeschlossen werden.

Der Datenaustausch zwischen diesen integrierten Visualisierungssystemen und der SPS erfolgt über denselben Mechanismus, der auch für das Programmiersystem verwendet wird. Denn auch dort können die Zustände der Variablen in der SPS online beobachtet werden. Für die Prozessvisualisierung muss der Anwender lediglich, wie in Abb. 03.12 dargestellt, die Grafikelemente dynamisieren.

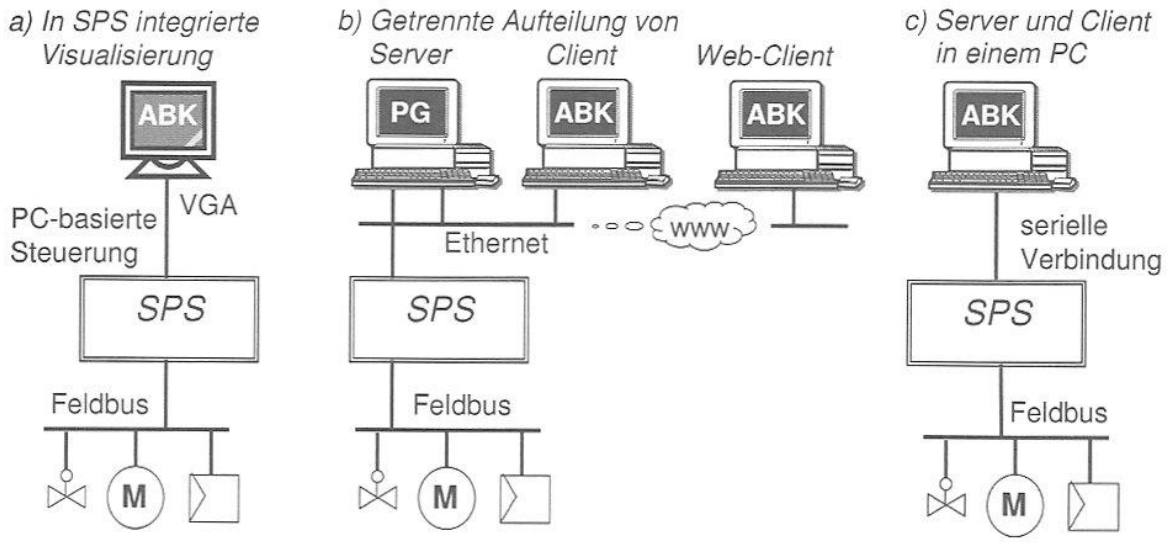


Abb. 03.13 Einsatz der Prozessvisualisierung in eine PC-basierte Steuerung integriert (a) oder getrennt auf mehreren PCs (b) oder gemeinsam mit dem PG in einem PC (c).

Für nichtintegrierte Visualisierungssysteme muss zusätzlich der Datenaustausch zur SPS konfiguriert werden. Dieser erfolgt über eine Client-Server-Verbindung. Der Server ist ein Programm, das auf einem PC läuft, der mit der SPS verbunden ist. Er stellt dabei die Daten der SPS auf dem Netz zur Verfügung, ohne sich darum zu kümmern, wer diese Daten verwendet.

Ein beliebiger Client, also z.B. eine ABK, kann die Dienste des Servers in Anspruch nehmen.

Um zu kennzeichnen, welche Signale zwischen Client und Server ausgetauscht werden sollen, sind mindestens folgende Festlegungen zu treffen (siehe Abb. 03.14):

- Name des Servers (SPS-Programm, z.B. STEP 7 oder CoDeSys),
- Pfad und Dateiname des Projekts, in dem die Anwenderprogramme laufen,
- Variablennamen der auszutauschenden Signale (z.B. Sensor- oder Stellsignale, interne Zustandsdaten der SPS).

Die gängigste Kopplung zwischen Prozessvisualisierung und SPS ist, wenn der Server auf einem PC des Programmiersystems läuft und über Ethernet Daten mit einem anderen PC austauscht, auf dem die Prozessvisualisierung als Client läuft (siehe Abb. 03.13b).

Da die meisten Visualisierungssysteme auch webfähig sind, kann der Client auch über das Internet mit dem PC des Servers in Verbindung treten. Dabei ist zusätzlich zu den oben genannten Festlegungen auch die IP-Adresse des Servers im Client zu spezifizieren. Natürlich können Server und Client auch auf ein- und demselben PC laufen (siehe Abb. 03.13c).

Unter Windows gibt es verschiedene Server-Typen. Die bekanntesten und in der Automatisierungstechnik genutzten sind DDE- (**D**ynamic **D**ata **E**xchange) und OPC-Server (**O**bject Linking and Embedding for **P**rocess **C**ontrol). Diese Server nutzen eine Windows-interne Schnittstelle, über die z.B. Daten aus einer Excel-Tabelle automatisch nach Word geschrieben und ständig aktualisiert werden. DDE- und OPC-Server sprechen diese Schnittstelle mit bestimmten Windows-Befehlen an. Insofern sind DDE- und OPC-Server nichts anderes als spezielle Datenaustauschprogramme, die parallel zu den Anwenderprogrammen der SPS laufen.

Während der DDE-Server nur Anwendungen koppeln kann. Die wie in Abb. 03.13c auf demselben PC laufen, kann ein OPC-Server auch Daten für OPC-Clients auf anderen PCs bereitstellen (siehe Abb. 03.13b).

Deshalb hat sich OPC zu dem Standard-Kopplungsmechanismus für die vertikale Integration entwickelt. Trotzdem wird eine DDE-Kopplung noch häufig eingesetzt, weil sie einfach und kostengünstig ist.

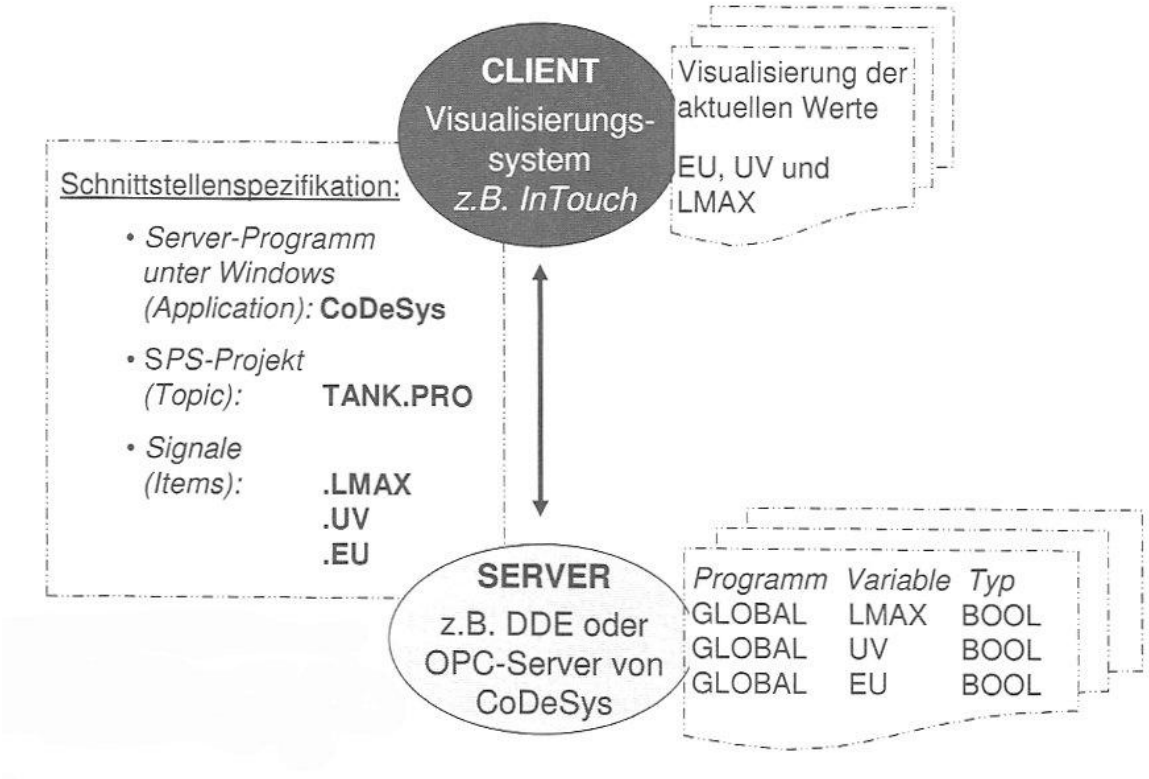


Abb. 03.14 Beispiel für den Datenaustausch zwischen Visualisierungssystem und SPS nach dem Client-Server-Prinzip

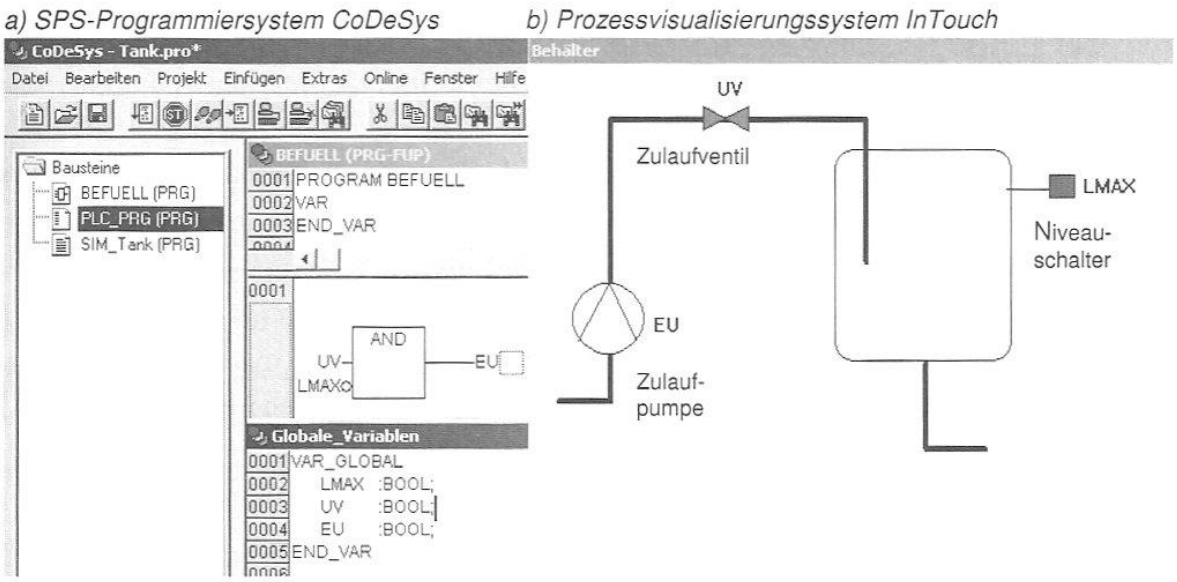


Abb. 03.15 Darstellung des SPS-Programms (a) mit zugehöriger Visualisierung (b). Per Mausklick auf das Ventilsymbol wird UV geöffnet und anschließend die Zulaufpumpe EU angesteuert, bis der Füllstand den Niveauschalter LMAX erreicht.

3.6.3 Prozessleitsysteme

Der Begriff Leiten bezieht sich auf das Weiterleiten der Informationen von den Sensoren und Aktoren über die SPS zum Leitsystem und umgekehrt.

Nach DIN 19222 umfasst Leiten „die Gesamtheit aller Maßnahmen, ggf. auch unter Mitwirkung des Menschen, um anhand der aus dem Prozess oder der Umgebung erhaltenen Daten einen gewünschten Prozessablauf nach bestimmten, festgelegten Zielen zu bewirken“.

Beispiele für solche Maßnahmen sind:

- Bedienung und Beobachtung des Prozesses und seiner Geräte,
- Meldung und Alarmierung bei Störungen,
- Protokollierung und Archivierung von Messwerten,
- Auswertung und Optimierung von Prozesszuständen.

Systeme, die alle diese Tätigkeiten ausführen, werden auch **S**upervisory-**C**ontrol-**a**nd-**D**ata-**A**nalysis-Systeme (SCADA) bezeichnet.

Für reine Bedien- und Beobachtungssysteme gibt es branchenabhängig verschiedene Namen und Abkürzungen, wie Visualisierungssystem (Visualisierung) oder Anzeige- und Bedienkomponente (ABK) sowie Man Machine Interface (MMI) oder Human Machine Interface (HMI).

Immer häufiger realisieren auch diese Systeme weitere der oben aufgeführten Tätigkeiten und nähern sich damit den Prozessleitsystemen an.

4 SPS-Programmierung nach IEC 61131

Die IEC 61131 hat sich in den letzten Jahren zu dem Standard für die Programmierung industrieller Automatisierungssysteme entwickelt, unabhängig davon, ob die Software auf einer klassischen SPS, im PC, in einem Prozessleitsystem oder einem intelligenten Antrieb abläuft. Die meisten Hersteller folgen dieser Norm und haben sich in der internationalen Nutzerorganisation PLCopen zusammengeschlossen.

Auch wenn nicht alle Automatisierungssysteme genau gleich zu programmieren sind, bieten die meisten Hersteller doch die Möglichkeit, in den von der Norm vorgeschlagenen Programmiersprachen zu programmieren. Auch die Architektur der Programmiersysteme ist in den meisten Systemen ähnlich und die Möglichkeiten zur Strukturierung der Software sind weitgehend gleich.

Dieses Kapitel vermittelt deshalb, wie Automatisierungsaufgaben zu strukturieren und nach IEC 61131 zu programmieren sind.

Zielsetzung der IEC 61131

Trotz der stetig wachsenden Leistungsfähigkeit moderner SPSen sahen sich die Hersteller zu Beginn der 90er Jahre immer größeren Anforderungen gegenübergestellt. SPSen verfügten zwar über einen großen Funktionsumfang, doch die zu automatisierenden Anlagen wurden immer komplexer und der Druck zur Kostensenkung stieg.

Eine Folge dieser Entwicklung war, dass SPSen heute nicht mehr singulär betrachtet werden können, sondern wie im vorigen Kapitel dargestellt nur im Verbund mit Leitsystemen, Feldbussystemen und weiteren SPSen, mit denen die Automatisierungsaufgabe verteilt gelöst werden kann.

Vor diesem Hintergrund wurden folgende Anforderungen in der Norm IEC 61131 formuliert:

- *Standardisierung* der Vorgehensweise bei der Programmierung,
- *objektorientierte* Strukturierung der Programme,
- *herstellerunabhängige* Programmiersprachen, Einbindung von Hochsprachen, wie z.B. PASCAL oder C,
- *offene*, standardisierte Schnittstellen,
- *Simulations*- und Testmöglichkeiten.

Die Verwirklichung dieser Ziele ermöglichte es, Software für das Planungs- und Wartungspersonal verständlicher und transparenter zu gestalten. Außerdem ergaben sich erhebliche Synergieeffekte durch wieder verwendbare Softwaremodule und die Portabilität der Software zwischen Automatisierungssystemen unterschiedlicher Hersteller. Auch verringerten sich der Lernaufwand und damit die Qualifizierungskosten zur Programmierung solcher Systeme.

Die Norm IEC 61131 gliedert sich in die drei Teile:

1. Das ***Softwaremodell*** beschreibt,
   welche Softwaremodule es gibt und wie sie mit der SPS-Hardware verknüpft sind,
2. das ***Kommunikationsmodell*** beschreibt,
   wie der Datenaustausch zwischen den Softwaremodulen erfolgt,
3. das ***Programmiermodell*** beschreibt,
   wie Anwenderprogramme programmiert werden.

4.1 Das Softwaremodell

Das Softwaremodell stellt die *Architektur* des Programmiersystems (siehe Abb. 04.01) dar. Die Steuerungs-Konfiguration beschreibt die Hardware des Steuerungssystems. Diese umfasst i.d.R. verschiedene Ressourcen, wie z.B. CPUn, E/A-Baugruppen, Kommunikationsprozessoren. Auf jeder CPU können verschiedene Tasks gleichzeitig ablaufen. Einer Task werden ein oder mehrere Programme zugeordnet, deren Ablauf durch die Task organisiert wird. Damit laufen auf einer CPU mehrere Programme gleichzeitig. Jedes Programm kann aus mehreren Funktionsbausteinen (Function Blocks, FBs) und Funktionen (Function Codes, FCs) bestehen. Globale Variablen ermöglichen den Datenaustausch zwischen den Programmen.

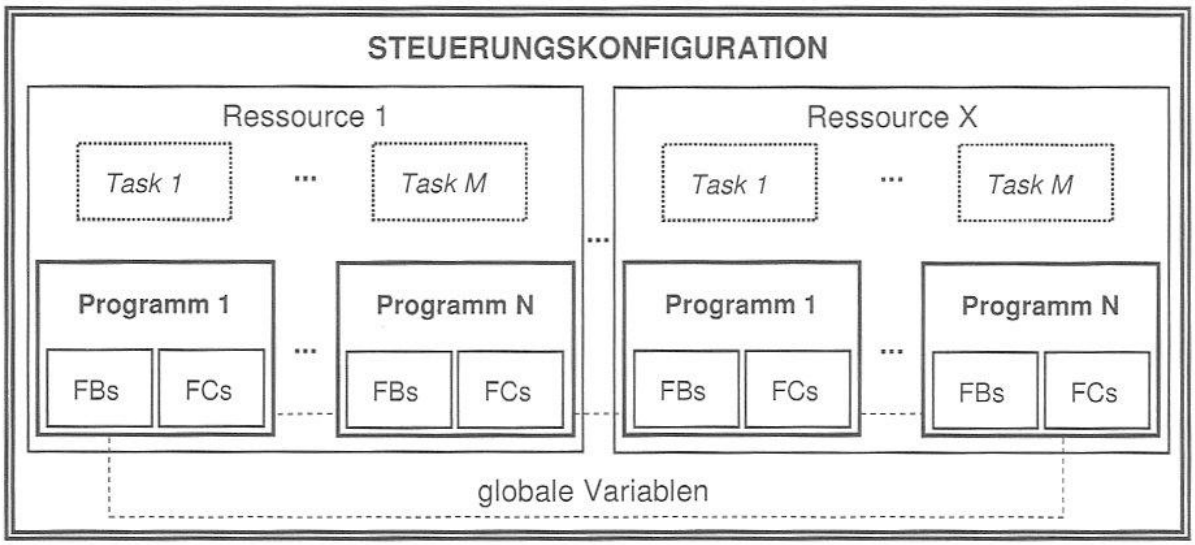


Abb. 04.01 Die Architektur oder Steuerungskonfiguration eines SPS-Systems nach IEC 61131 besteht aus mehreren Ressourcen mit verschiedenen Tasks, denen jeweils unterschiedliche Programme mit Funktionsbausteinen (FB) und Funktionen (FC) zugeordnet sein können. Globale Variablen ermöglichen den programmübergreifenden Datenaustausch.

Diese Module der Architektur eines SPS-Programmiersystems werden nachfolgend näher erläutert.

4.1.1 Steuerungskonfiguration und Ressourcen

In der Steuerungskonfiguration wird die bestehende *SPS-Hardwarestruktur* in der Programmierumgebung bekannt gemacht, damit beispielsweise Variablen mit den zugehörigen E/A-Kanälen der E/A-Baugruppen verbunden oder die Software auf verschiedene CPUn verteilt werden kann.

Abb. 04.02 zeigt ein Beispiel für eine Steuerungskonfiguration, bestehend aus mehreren Ressourcen. Unter *Ressourcen* versteht man die Baugruppen/Einsteckkarten des Steuerungssystems, auf die die Software zugreift. Im Fall von Hardware-SPSen bestehen die Ressourcen aus CPUn, E/A-Baugruppen, Kommunikationsbaugruppen und weiteren z.B. Spezial- bzw. Sonderbaugruppen (schnelle Zähler, Motion Contol u.a.).

In den meisten Programmiersystemen erfolgt die Steuerungskonfiguration menügesteuert, d.h. man kann die gewünschten Baugruppen auswählen und dann per Drag and Drop an die entsprechende Stelle in der Konfiguration ziehen.

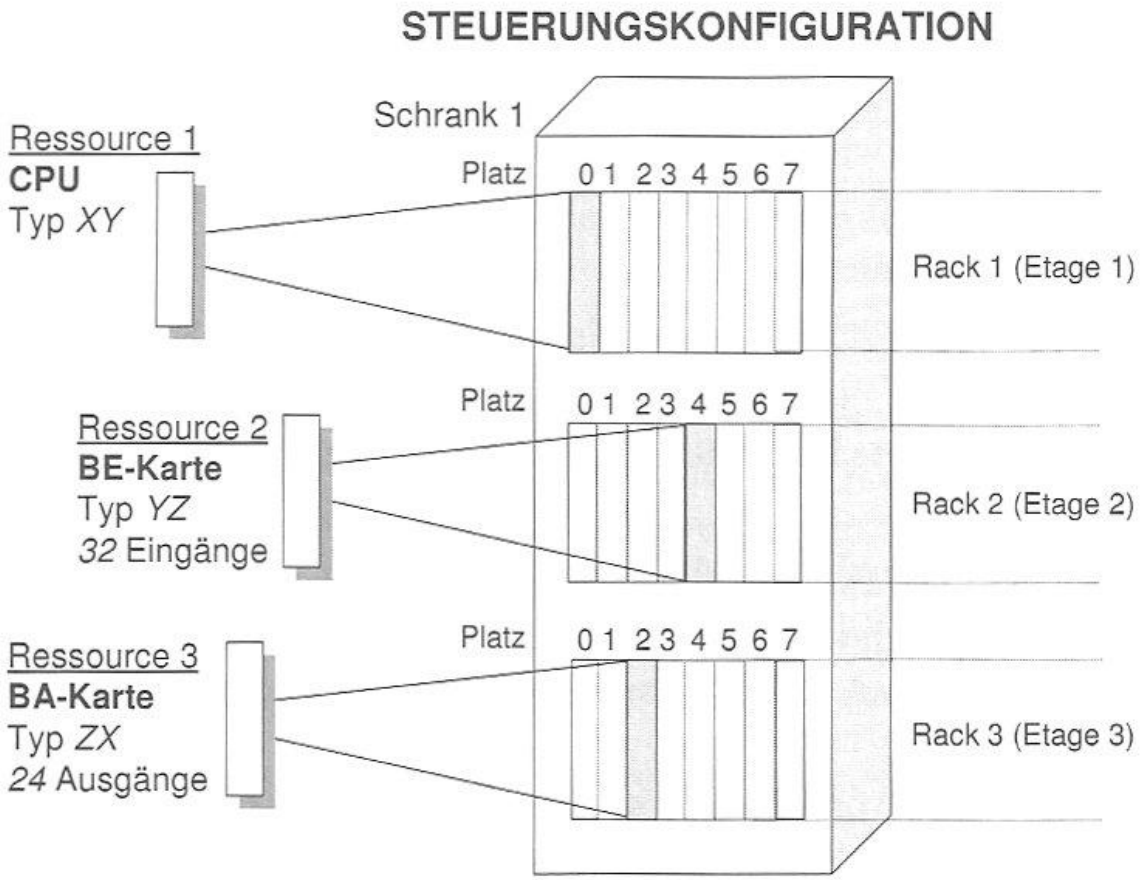


Abb. 04.02 Festlegung der Ressourcen in der Steuerungskonfiguration per Drag and Drop.

Die Adressierung der E/A-Kanäle von E/A-Baugruppen erfolgt im Rahmen der Variablendeklaration mit dem Schlüsselwort AT. Wie Tabelle 04.01 zeigt, werden Eingangskanäle mit einem „%I“ für Input und Ausgangskanäle mit einem „%Q“ für Output gekennzeichnet. Der danach folgende Bauchstabe beschreibt, ob es sich bei dem E/A-Signal um ein Bit, Byte, Word oder Doubleword handelt. Demnach wird beispielsweise der 2. Kanal einer binären Eingangskarte am Steckplatz 1 durch die Bezeichnung „%IX1.2“ angesprochen.

Kennzeichnung der E/A-Adressierung	1. Folgebuchstabe	2. Folgebuchstabe	Beispiel
AT %	I Eingang Q Ausgang	X Bit B Byte (8 Bit) W Word (16 Bit) D Doubleword (32 Bit)	AT %IX1.2 AT %IB0 AT %QW7 AT %QD5

Tabelle 04.01 Adressierung der E/A-Kanäle.

Das folgende Beispiel erläutert die Vorgehensweise zur Festlegung der Steuerungskonfiguration für eine Soft-SPS mit dem SPS-Programmiersystem CoDeSys der Fa. 3S.

Beispiel 04.01: Steuerungskonfiguration

Mit dem Programmiersystem CoDeSys der Fa. 3S in der vorliegenden Version 2.3.7.0, dass Sie von meiner Website unter http://htw-berlin.applicad-atit.de/gf8wiw herunterladen können, können Sie sich in die Programmierung von SPSen nach IEC 61131 einarbeiten. Enthalten sind eine Soft-SPS und ein Visualisierungssystem, so dass Sie die erstellten

Programme testen und sich in das Thema Visualisierung einarbeiten können. Die Beispiele finden Sie ebenfalls auf meine Webseite und laufen unter Windows Vista.

Die hier bereitgestellte Version ist bis Windows XP freigegeben. Ein Test auf dem Betriebssystem Windows Vista verlief durchaus positiv. Es stehen nicht wie bei Windows XP drei Soft-SPSen (Targets) zur Verfügung, sondern nur eine. Bitte lesen Sie dazu auch das Einführungsbeispiel „EinfBspCoDeSys.pdf“. Bei der Verwendung von Windows 7 könnte es zu ähnlichen Effekten kommen wie unter Windows Vista.

Will man, wie in Abb. 04.03 gezeigt, „echte“ Sensoren und Aktoren an die Soft-SPS anschließen, muss im PC auch eine „echte“ Profibus-Schnittstellenkarte (z.B. von der Fa. Hilscher) eingebaut sein. Am Profibus können dann entsprechende Feldbusgeräte bzw. dezentrale E/A-Baugruppen (z.B. ET 200M von Siemens), über die dann konventionelle Datenpunkte angeschlossen werden, betrieben werden.

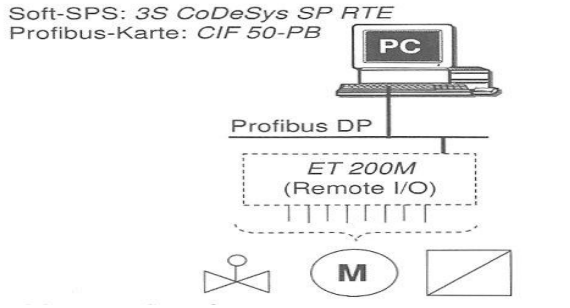


Abb. 04.03 Aufbau des Steuerungssystems.

An dieser Stelle führen Sie am besten die Anfangsübung des Einführungsbeispiels „EinfBspCoDeSys.pdf“ komplett aus.

Abb. 04.04 zeigt beispielhaft die Steuerungskonfiguration der Soft-SPS CoDeSys.

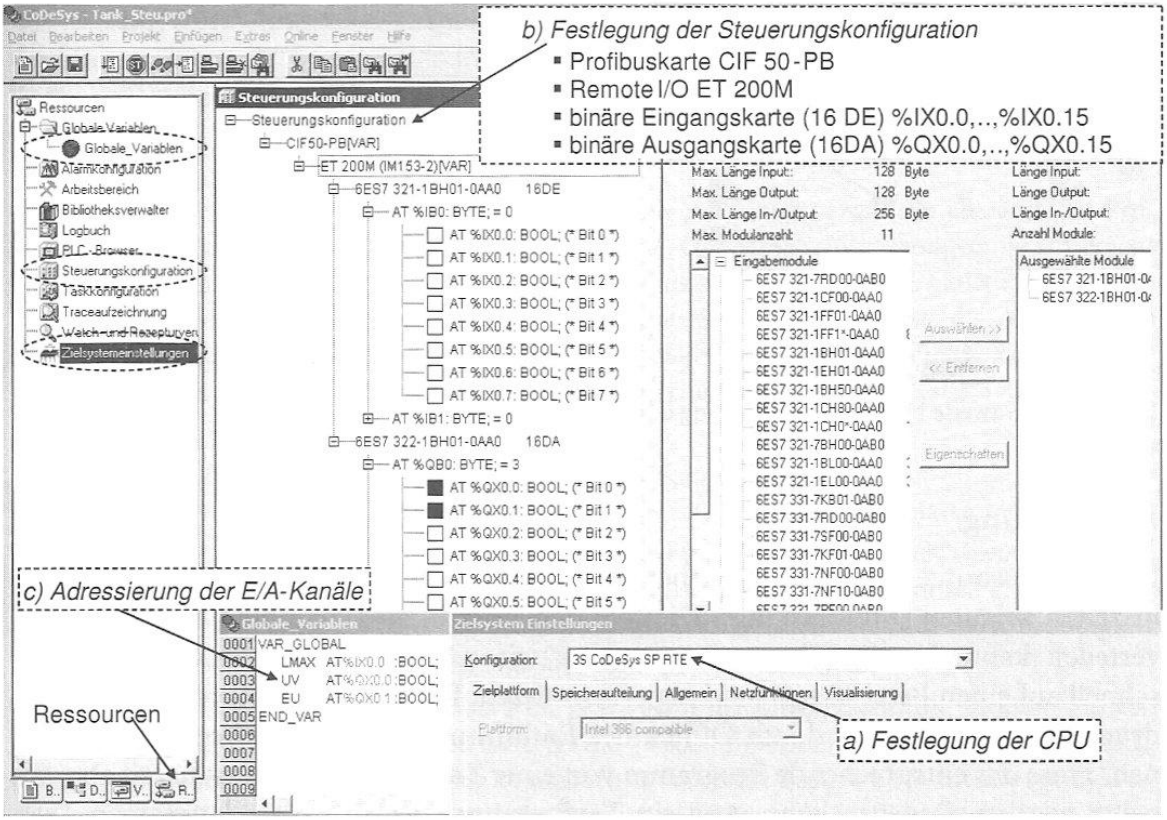


Abb. 04.04 Festlegung der Steuerungskonfiguration mit CoDeSys.

4.1.2 Tasks

In einer CPU können eine oder mehrere Tasks ablaufen. Eine Task organisiert den zeitlichen Ablauf der Programme. Die Programme innerhalb einer Task werden in der Reihenfolge ihres Aufrufes nacheinander abgearbeitet (siehe Abb. 04.05).

Einer Task wird eine Zykluszeit zugeordnet. Alle Programme einer Task laufen also innerhalb der gleichen Zykluszeit ab. Bei der Standardtask, wie *Main* oder *PLC_PRG* bei CoDeSys kann es sein, dass die Zykluszeit nicht eingestellt werden muss, sondern sich aus der Programmverarbeitungszeit der Task plus einen gewissen zeitlichen Overhead ergibt.

Innerhalb der Task werden die Daten streng nach dem EVA-Prinzip verarbeitet:

- Zuerst werden die Eingangsdaten aller Programme eingelesen,
- dann werden die Programme nacheinander abgearbeitet und
- schließlich werden die Ausgangsdaten aller Programme ausgegeben.

Die Zykluszeit einer Task wird vom Prozess aus, also von außerhalb der Steuerung gesehen. Sie wird gemessen zwischen Einlesen und Ausgeben der Prozessdaten und beinhaltet damit auch die Reaktionszeit der Steuerung.

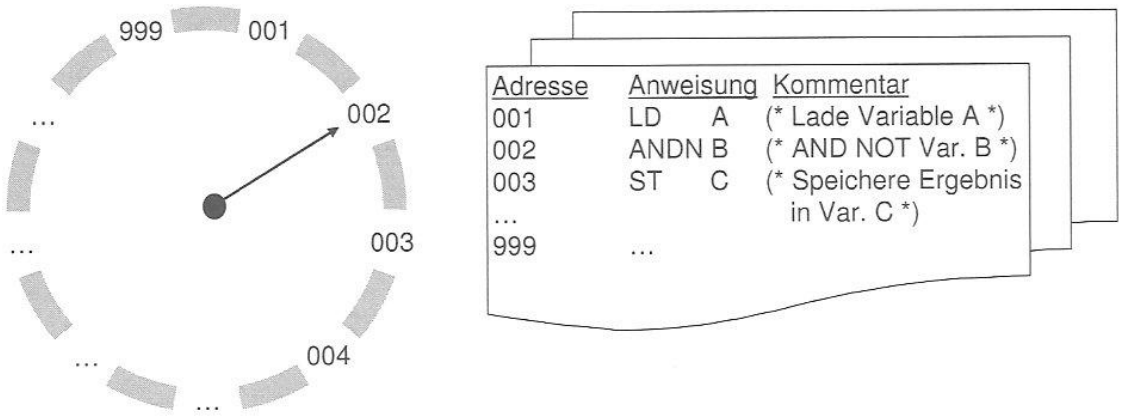


Abb. 04.05 Schrittweise Programmabarbeitung durch eine Task.

Da diese Zeit endlich ist (z.B. 150 ms), ergibt sich daraus auch, dass ein Prozess, der von einer Steuerung gesteuert werden soll, die Zykluszeit praktisch nicht wahrnehmen soll. Die Zykluszeit einer Steuerung muss also an die zeitlichen Anforderungen eines zu steuernden Prozesses angepasst sein. Relativ langsame, träge Prozesse z.B. in der Verfahrenstechnik benötigen Reaktionszeiten von mehreren hundert Millisekunden bis zu einigen Sekunden. Hierfür sind Zykluszeiten von z.B. 150 … 300 ms in den meisten Fällen völlig ausreichend. Anlagen der Fertigungstechnik mit Montage- und Antriebssystemen sind zeitlich wesentlich anspruchsvoller und erwarten Reaktionen im Bereich von Millisekunden bis einigen zehn Millisekunden. Hier müssen also Steuerungen eingesetzt werden, die sehr kleine Zykluszeiten realisieren können.

Multitasking

In einer Ressource bzw. CPU können auch *mehrere* Task mit unterschiedlichen Zykluszeiten praktisch parallel ablaufen. Wir sprechen dann von *Multitasking*. Damit ist es möglich, die Rechenleistung auf die verschiedenen Programme zu verteilen. Voraussetzung dabei ist jedoch, dass die CPU über eine ausreichende Rechenleistung verfügt. Den Ablauf in einer CPU mit einer Task zeigt Abb. 04.06a und mit mehreren Tasks zeigt Abb. 04.06b.

Das Multitasking wird durch eine Zeitgebereinheit in der CPU realisiert, die mit einem Interrupt-Signal die eine laufende Task unterbricht, wenn eine andere Task abgearbeitet werden soll. Abhängig vom Rechenaufwand einer Task wird ihr ein bestimmter Prozentteil der verfügbaren Rechenzeit der CPU zugeteilt. Bei Vorhandensein nur einer CPU nennt man die Fähigkeit, mehrere Tasks quasi parallel ablaufen zu lassen auch *Pseudo-Multitasking*. Ein wirkliches paralleles Ablaufen von verschiedenen Tasks erreicht man genau genommen nur bei Einsatz von je einer CPU pro Task. In der Praxis ist das jedoch fast nie der Fall. Es kommen zwar bei SPSen auch mehrere CPUn zum Einsatz, doch laufen auch bei diesen SPSen i.d.R. mehrere Tasks auf einer CPU.

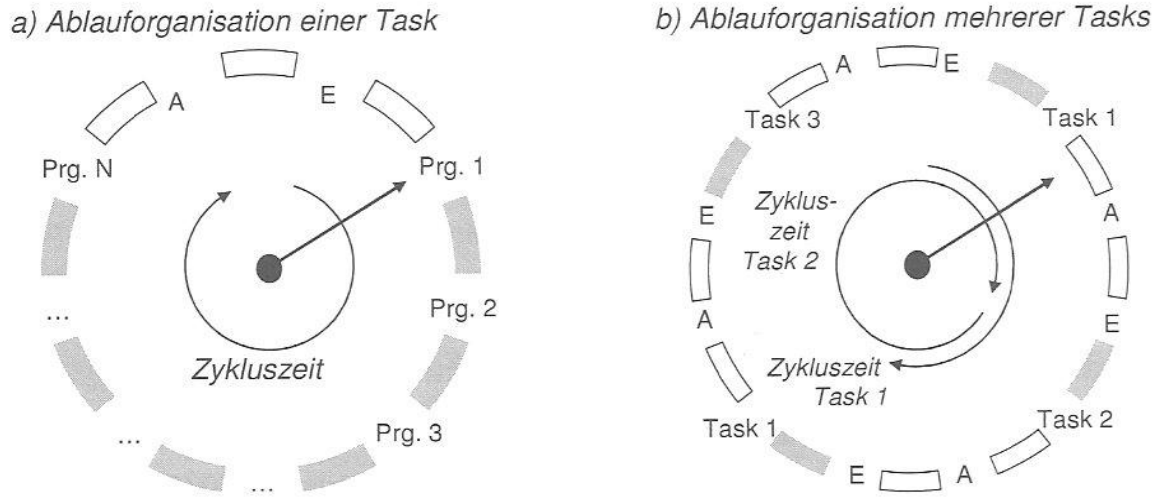


Abb. 04.06 a) Beim Ablauf einer Task werden zunächst die Eingangsdaten aller Programme eingelesen (E), dann die Programme Prg. 1 … N abgearbeitet (V), bevor alle Ausgangsdaten ausgegeben werden (A).
b) In einer Ressource könne aber auch mehrere Tasks mit unterschiedlichen Zykluszeiten nacheinander abgearbeitet werden.

Beispiel 04.02: Taskkonfiguration

In den meisten SPSen gibt es die Möglichkeit, in einer Art Hauptprogramm *Main* alle anderen Programme nacheinander aufzurufen und abzuarbeiten. Die Simatic-Steuerungen von der Fa. Siemens bezeichnet dieses Hauptprogramm als Organisationsbaustein OB1, in CoDeSys heißt es PLC_PRG. Es wird automatisch vom Betriebssystem aufgerufen und arbeitet die Anweisungen, die in ihm aufgeführt sind, zyklisch ab. Um die Funktionsweise einer Task, wie in Abb. 04.06a gezeigt, zu realisieren, ruft man die abzuarbeitenden Programme mit dem Befehl CAL einfach nacheinander auf:

```
PROGRAM PLC_PRG
     CAL   BEFUELL    (* Aufruf des Steuerungsprogramms *)
     CAL   SIM_Tank   (* Aufruf des Prozesssimulationsprogramms *)
```

Die Programme werden also nacheinander abgearbeitet. Sie laufen dabei mit der gleichen, aber unbekannten Zykluszeit ab. Die Funktionsweise kann mit dem SPS-System CoDeSys veranschaulicht werden, in dem das Beispiel 04.01 ausgeführt wird. Die Zykluszeit hängt vom Umfang der Programme, genau genommen von deren Ausführungszeiten, ab und kann nur zur Laufzeit, also nach der Programmierung, gemessen werden.

Ermitteln der der Zykluszeit:

Bei einigen SPSen kann die Zykluszeit während der Programmabarbeitung in den Systemeigenschaften eingesehen werden (z.B. Simatic S7). Ist das nicht möglich, muss der Anwender ein Programm dafür schreiben. Ein solches Programm kann z.B. beinhalten, einen *Timer* (Zeitglied) zu starten und nach n maligem Durchlaufen des Hauptprogrammes zu stoppen. Die Anzahl der Durchläufe wird mit Hilfe eines *Counters* (Zähler) gesteuert. Die Zykluszeit ergibt sich dann aus der Division von Timerwert (Zeit) durch den Counterwert (Anzahl der Durchläufe).

Um Task mit einer definierten Zykluszeit ablaufen zu lassen, können Programme unabhängig vom Hauptprogramm (Main, PLC_PRG, OB1) in entsprechend definierten Task ablaufen. Abb. 04.07 zeigt die Taskkonfiguration unter CoDeSys für das Beispiel der Tankbefüllung. Um die Taskkonfiguration zu verdeutlichen wurden zwei Task, die Task *T500* mit einer Zykluszeit von 500 ms und die Task *T5000* mit einer Zykluszeit von 5000 ms = 5 s, angelegt. Der Task *T500* wurde das Simulationsprogramm `SIM_Tank` und der Task *T5000* das Steuerungsprogramm `BEFUELL` zugeordnet.

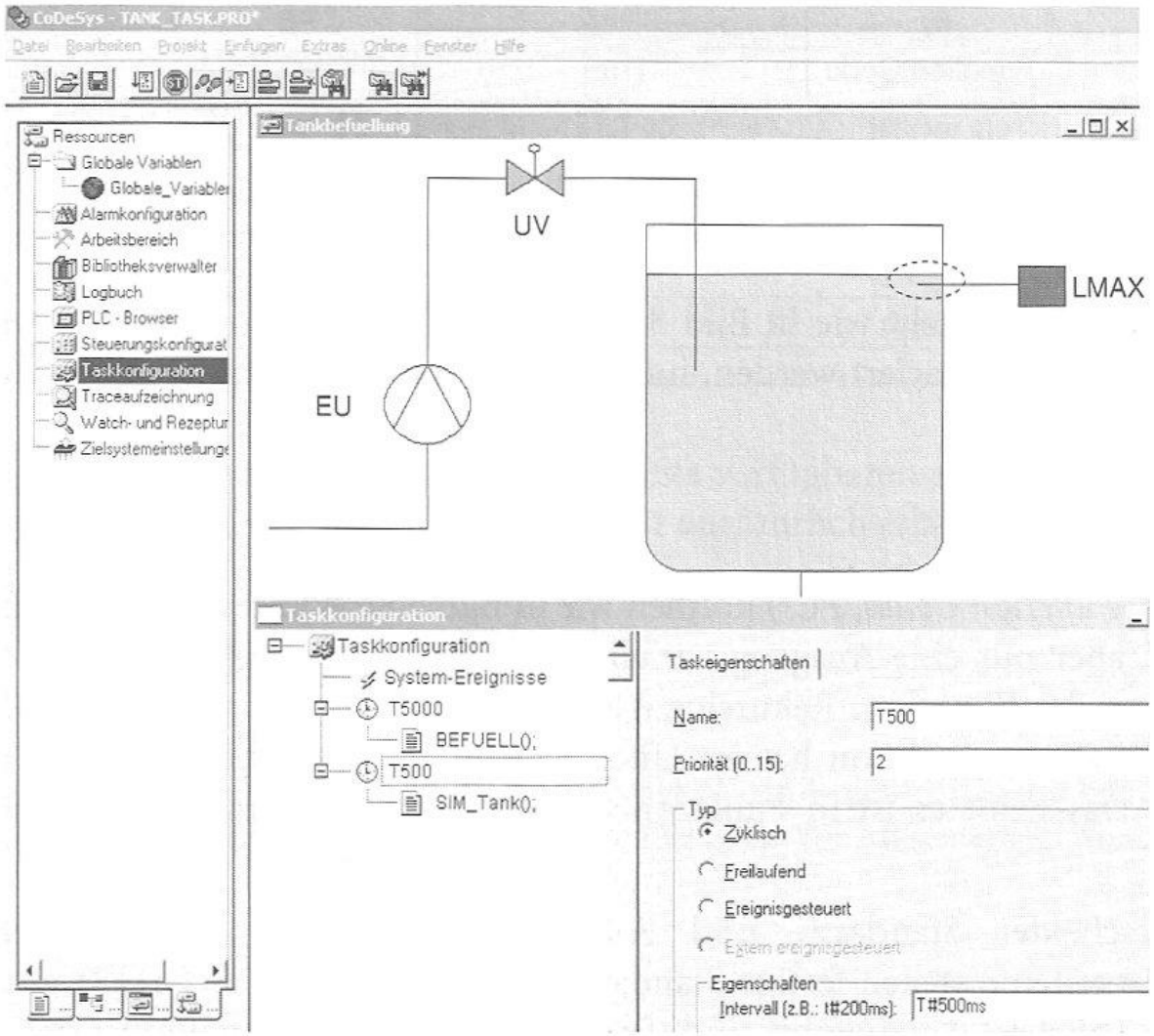


Abb. 04.07 Durch Taskkonfiguration in CoDeSys kann das Programm `SIM_Tank` zur Simulation der Tankbefüllung mit der Zykluszeit von exakt 500 ms bearbeitet werden, während das Steuerungsprogramm `BEFUELL` zehnmal langsamer abläuft.

Startet man nun das Programm, stellt man im Vergleich zum Beispiel 04.01 zunächst fest, dass sich der Behälter sehr viel langsamer füllt. Das Simulationsprogramm `SIM_Tank` läuft jetzt viel langsamer als bei der Abarbeitung im Hauptprogramm PLC_PRG ab. Des Weiteren wird der Behälter etwas überfüllt, weil das Programm `BEFUELL` zehn Mal seltener abgearbeitet wird als `SIM_Tank`. Die Pumpe kann also erst 5 s nach Erreichen des Füllstandes `LMAX` abgeschaltet werden.

Wie man bei genauem Hinsehen in Abb. 04.07 erkennen kann, können Task auch mit einer *Priorität* versehen werden. Wenn zwei Tasks die gleiche oder vielfache Zykluszeit haben, entscheidet ihre Priorität darüber, welche von beiden zum Interrupt-Zeitpunkt abgearbeitet wird.

4.1.3 Programmorganisationseinheiten

Ein *Anwenderprogramm*, wie SPS-Programme üblicherweise genannt werden, das zur Automatisierung einer Anlage dient, besteht aus ***P**rogramm**o**rganisations**e**inheiten* **POE**en, im englischen sind das **P**rogram **O**rganisation **U**nits **POU**s.

Programmorganisationseinheiten POEen können sein:

- Programme,
- Funktionen (siehe Abb. 04.08a) und
- Funktionsbausteine (siehe Abb. 04.08b).

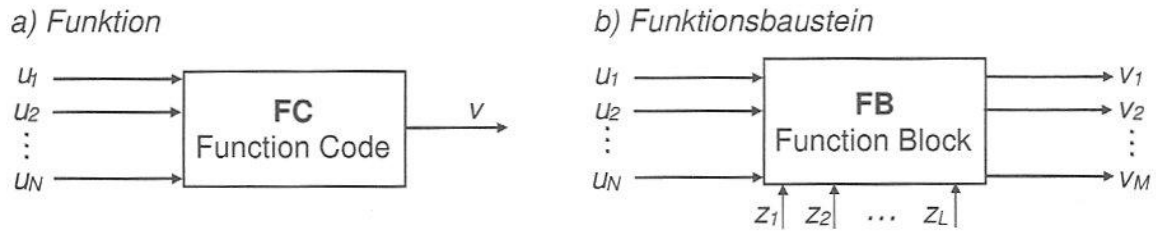


Abb. 04.08 a) Eine Funktion verknüpft mehrere Eingänge ***u*** zu einem Ausgangssignal ***v***.
b) Ein Funktionsbaustein erzeugt aus den Eingängen ***u*** unter Einbeziehung interner Zustände ***z*** ein oder mehrere Ausgangssignale ***v***.

Jede dieser POEen besteht aus einem Deklarationsteil, in dem die Variablen deklariert werden, und einem Anweisungsteil, der den Programmcode enthält. In Abb. 04.09a wird das veranschaulicht.

4.1.3.1 Funktionen

Funktionen (*Function Codes*, *FCs*) können mehrere Eingangsvariablen ***u***, aber nur eine Ausgangsvariable **v** haben. Die Ausgangsvariable ist dabei der Rückgabewert der Funktion. Rekursive Aufrufe einer Funktion sind nach IEC 61131 nicht möglich. Eine Funktion hat im Gegensatz zu einem Funktionsbaustein kein Gedächtnis. Das bedeutet, dass es in Funktionen nicht erlaubt ist, Zwischenwerte zu speichern.

Man unterscheidet Standard- und Anwenderfunktionen. Die meisten SPS-Programmiersysteme stellen fertige Standard-Funktionen nach IEC 61131 zur Verfügung. In Tabelle 04.02 sehen sie eine Übersicht. Diese Standard-Funktionen befinden sich in einer Programmbibliothek und müssen vom Anwender nicht mehr programmiert werden, sondern er kann sie direkt in seinen Programmen verwenden.

Darüber hinaus hat der Anwender, wie im nachfolgenden Beispiel gezeigt wird, die Möglichkeit, eigene Anwender-Funktionen zu schreiben.

Boole'sche Funktionen:		**Bitfolge-Funktionen:**	
AND	UND-Verbindung	ROL	Links rotieren
NOT	Negation (NICHT)	ROR	Rechts rotieren
OR	ODER-Verbindung	SHL	Links schieben
XOR	EXOR-Verbindung	SHR	Rechts schieben
Arithmetische Funktionen:		**Numerische Funktionen:**	
ADD	Addierer	ABS	Absolutwert
DIV	Divisor	ACOS	Arccos
EXPT	Potenzierung	ASIN	Arcsin
MOD	Modulo-Division	ATAN	Arctan
MUL	Multiplizierer	COS	Cosinus
NEG	Negierer	EXP	Exponent
SUB	Subtrahierer	LN	Natürlicher Logarithmus
Funktionen zur Typumwandlung:		LOG	Logarithmus zur Basis 10
BYTE_TO_WORD		SIN	Sinus
BYTE_TO_INT		SQRT	Quadratwurzel
BYTE_TO_REAL		TAN	Tangens
UINT_TO_INT		**Funktionen für Auswahl:**	
INT_TO_UINT		LIMIT	Begrenzung
DINT_TO_INT		MAX	Maximum
INT_TO_DINT		MIN	Minimum
INT_TO_BYTE		SEL	Binäre Auswahl
INT_TO_WORD		**Funktionen für Vergleich:**	
INT_TO_REAL		EQ	gleich (equal)
REAL_TO_BYTE		LT	kleiner als (less than)
REAL_TO_WORD		GT	größer als (greater than)
REAL_TO_INT		LE	kleiner gleich (less or equal)
TIME_TO_DINT		GE	größer gleich (greater or eq.)
DINT_TO_TIME		NE	ungleich (not equal)

Tabelle 04.02 Standard-Funktionen nach IEC 61131 (Auswahl).

Beispiel 04.03: Anwender-Funktionen

Eine Gebäudeheizung, wie in Abb. 04.09 gezeigt, soll in Abhängigkeit von der Außentemperatur gesteuert werden. Hierfür wird die Gaszufuhr über den Öffnungsgrad ***y*** des Gaszufuhrventils linear zur Außentemperatur ***T*** eingestellt:

$$y = \begin{cases} 0 & \forall \quad T \geq 20'C \\ k \cdot (20'C - T) & \forall \quad 0'C < T < 20'C \\ 100 & \forall \quad T \leq 0'C \end{cases} \qquad \text{(Gl. 04.01)}$$

Abb. 04.09a zeigt die Software für die **Funktion Y**, die den Öffnungsgrad des Gaszufuhrventils linear zur Außentemperatur steuert. Sie besteht aus einem Deklarationsteil zur Festlegung der Variablen und einem Anweisungsteil, der die Programmlogik enthält. Die

Funktion Y besitzt die Eingangsvariablen ***T***, ***K***, ***MAN*** und ***y_MAN***. Die Ausgangsvariable der **Funktion Y** ist der Rückgabewert der Funktion.

Gl. 04.01 wird durch die IEC-Standard-Funktion **LIMIT** realisiert. Weiterhin wird die IEC-Standard-Funktion **SEL** als Schalter oder Multiplexer verwendet. Sie weist der Variablen ***Y*** den Wert der Variablen ***y_MAN*** zu, wenn die Variable ***MAN*** auf `TRUE` gesetzt ist, die Steuerung sich also in der Betriebsart MANuell (Handbetrieb) befindet. Ist ***MAN*** dagegen auf `FALSE` gesetzt, die Steuerung befindet sich in der Betriebsart AUTOmatik (Automatikbetrieb), wird ***Y*** mit dem Ausgang der IEC-Standard-Funktion **LIMIT** verknüpft.

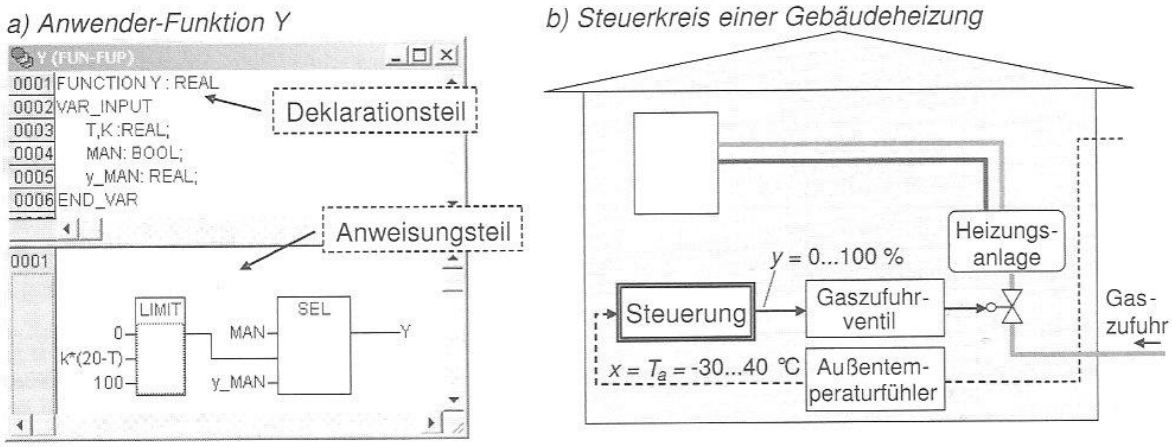


Abb. 04.09 Zur Steuerung der Gaszufuhr für eine Gebäudeheizung (a) bestimmt die Anwender-Funktion Y den Öffnungsgrad des Gaszufuhrventils in Abhängigkeit der Außentemperatur T_a (b).

Außer Standard-Funktionen stehen in den meisten SPS-Programmiersystemen auch *Standard-Funktionsbausteine* zur Verfügung.

4.1.3.2 Funktionsbausteine

In Funktionsbausteinen (*Function Blocks, FBs*) können auch interne Zwischenwerte, Zustände genannt, gespeichert werden. Im Unterschied zu einer Funktion kann ein Funktionsbaustein mehrere Ausgänge bzw. Ausgangsvariablen ***v*** haben. Die werden durch Verarbeitung der Eingänge bzw. Eingangsvariablen ***x*** und der internen Zustände ***z*** erzeugt (siehe Abb. 04.08b).

In Tabelle 04.03 sind einige wichtige IEC-Standard-Funktionsbausteine aufgeführt. Wie diese Bausteine für verschiedene automatisierungstechnische Aufgabenstellungen verwendet werden und wie sie aufgebaut sind, findet man in den Handbüchern zu den SPS-Programmiersystemen, so auch im Handbuch von CoDeSys. Die Datei des Handbuches im pdf-Format „**CoDeSys_V23_D.pdf**“ finden sie auch auf meiner Website unter http://htw-berlin.applicad-atit.de/gf9wiw .

Da das einfache Speicherglied *RS-Flip-Flop* sehr häufig Verwendung findet und damit ein wichtiger IEC-Standard-Funktionsbaustein ist, soll sein Aufbau und seine Funktion nachfolgend kurz vorgestellt werden.

Speicher *(s. Abschnitt 4.2.1)*	SR	Flip-Flop mit vorrangig Setzen
	RS	Flip-Flop mit vorrangig Rücksetzen
Trigger	F_TRIG	Erkennung einer fallenden Flanke
	R_TRIG	Erkennung einer steigenden Flanke
Zähler *(s. Abschnitt 4.2.2)*	CTU	Aufwärts-Zähler
	CTD	Abwärts-Zähler
	CTUD	Auf- und Abwärts-Zähler
Timer *(s. Abschnitt 4.2.3)*	TP	Pulstimer (Mono-Flop)
	TON	Timer mit Einschaltverzögerung
	TOF	Timer mit Ausschaltverzögerung

Tabelle 04.03 Standard-Funktionsbausteine nach IEC 61131 (Auswahl).

Beispiel 04.04: IEC-Funktionsbaustein *RS-Flip-Flop*

Am besten lässt sich der Aufbau und die Funktion des FBs *RS-Flip-Flop* am Beispiel erlernen, in dem man es selbst durchführt. Es soll also der Funktionsbaustein eines RS-Flip-Flops programmiert werden. Das Charakteristische des RS-Flip-Flops ist, dass der aktuelle Ausgangswert ***OUT*** gespeichert und im nächsten Bearbeitungszyklus als interner Zustand ***z*** auf die Eingangslogik zurückgeführt wird.

Das Verhalten des RD-Flip-Flops drückt sich in folgender Wahrheitstabelle aus:

Eingänge		Zustand	Ausgang
SET	RESET	Z	OUT
0	0	0	0
1	0	0	1
0	0	1	1
1	0	1	1
1	1	1	0
1	1	0	0
0	1	0	0
0	1	1	0

0 entspricht *FALSE*
1 entspricht *TRUE*

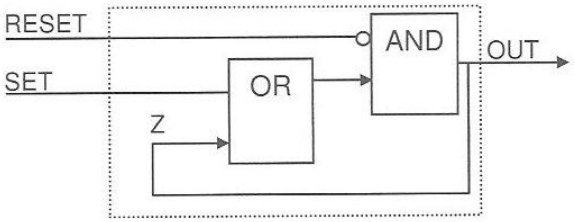
Wie man sieht, ist das RS-Flip-Flop rücksetzdominant, denn der Ausgang ***OUT*** kann nur dann gesetzt, also `TRUE` werden, wenn der Rücksetzeingang ***RESET*** = `FALSE` ist. Das Speicherverhalten drückt sich dadurch aus, dass der Ausgang so lange gesetzt bleibt, bis ***RESET*** = `TRUE` wird, auch wenn ***SET*** = `FALSE` ist.

Somit ergibt sich die Schaltfunktion für ***OUT***:

$$OUT = \overline{RESET} \wedge (SET \vee Z) \qquad \text{(Gl. 04.02)}$$

Die Programmierung des Funktionsbausteins **RS** geschieht folgendermaßen:

```
FUNCTION_BLOCK RS
  VAR_INPUT
    RESET : BOOL;     (* Rücksetz-Eingang *)
    SET   : BOOL;     (* Setz-Eingang *)
  END_VAR
  VAR_OUTPUT
    OUT   : BOOL;     (* Ausgang *)
  END_VAR
  VAR
    Z     : BOOL;     (* gespeicherter Zustand: Ausgangswert vom letzten Zyklus *)
  END_VAR
```



4.1.4 Variablen

Das letzte Element des Softwaremodells aus der Abb. 04.01 sind die *Variablen*, mit denen die Kommunikation zwischen den Programmorganisationseinheiten POEen realisiert wird.

Man unterscheidet prinzipiell folgende Variablenarten, gekennzeichnet durch die Schlüsselwörter:

- **VAR**: *Lokale* Variablen sind nur innerhalb der Programmorganisationseinheit POE gültig, in der sie deklariert wurden.
- **VAR_GLOBAL**: *Globale* Variablen gelten in allen Programmen, Funktionen und Funktionsbausteinen.
- **VAR_INPUT**: Durch *Eingangsvariablen* werden Werte in Funktionen oder Funktionsbausteinen hineingeschrieben.
- **VAR_OUTPUT**: Durch *Ausgangsvariablen* werden Werte von Funktionsbausteinen ausgegeben.
- **VAR_IN_OUT**: Im Gegensatz zu reinen Eingangsvariablen können Ein- und Ausgangsvariablen innerhalb des Funktionsbausteins verändert und dann ausgegeben werden.
- **VAR_RETAIN**: Diese Variablen behalten ihren Wert, wenn die SPS aus- und wieder eingeschaltet wird.
- **VAR_PERSISTENT**: Persistente Variablen behalten ihren Wert, wenn die Software erneut in die SPS geladen wird.

Die beiden letzten Variablentypen werden auch als ***permanente** Variablen* bezeichnet und können miteinander kombiniert werden. Sie lassen sich lokal oder auch global deklarieren.

Adressiert eine Variable einen E/A-Kanal, z.B. durch „*AT %IX0.0*“, spricht man auch von einer direkten Variablen.

Eine Variablendeklaration besteht aus folgenden 5 Bestandteilen (optionale Bestandteile werden *kursiv* dargestellt):

<Variablenname> *AT<Adresse>* :<Datentyp> :=<*Initialwert*>; *(* Kommentar *)*

z.B. LMAX AT %IX0.0 :BOOL := FALSE; (* Niveau *)
oder LMIN AT %IX0.1 :BOOL

Über die Wahl der *Variablennamen* lohnt es sich ein wenig nachzudenken. Der Variablenname sollte in großem Maße selbsterklärend (selbstdokumentierend) sein, um die Lesbarkeit eines Programms zu erleichtern. Weit verbreitet ist die sogenannte *ungarische Notation*, in der Datentypen im Variablennamen mit einkodiert werden, um in großen Programmen den Programmierer zu unterstützen. Oft, z.B. in der Prozessautomatisierung, ist das Ziel aber ein anderes. Es geht in erster Linie darum, die Software für den Anwender verständlich zu gestalten, damit er leicht Änderungen und Ergänzungen vornehmen kann. Aus diesem Grund sollten Variablen stets so bezeichnet werden, dass ihr Zweck durch den Namen verdeutlicht wird. Dann lässt sich ein Programm oft auch ohne bzw. mit kurzen Kommentaren verstehen.

Jede Variable ist durch einen Datentyp charakterisiert. Die *Standard-Datentypen* nach IEC 61131, wie `BOOL`, `INT`, `REAL` und `TIME`, sind in Tabelle 04.04 aufgeführt.

	Datentyp	Größe	Wertebereich	Beispiel
Bitfolgen	BOOL	1 Bit	FALSE/TRUE	FALSE
	BYTE	8 Bit	16#00...16#FF *(Hex-Darstellung)*	16#00
	WORD	16 Bit	16#0000...16#FFFF	16#0000
	DWORD	32 Bit	16#00000000...16#FFFFFFFF	16#00000000
Ganze Zahlen	SINT	8 Bit	–128...127	0
	INT	16 Bit	–32768...32767	0
	DINT	32 Bit	–2147483648... 2147483647	0
Ganze Zahlen ohne Vorzeichen	USINT	8 Bit	0...255	0
	UINT	16 Bit	0...65535	0
	UDINT	32 Bit	0...4294967295	0
Fließkommazahlen	REAL	32 Bit	$-3{,}4 \cdot 10^{38}$...$3{,}4 \cdot 10^{38}$ (Auflösung $1{,}18 \cdot 10^{38}$)	0.0
Zeit	TIME			t#2h:1m:0s:0ms
Uhrzeit	TIME_OF_DAY			tod#15:23:17.456
Datum	DATE			d#2003-18-01
Zeichenfolge	STRING			STRING(4)='Text'

Tabelle 04.04 Standard-Datentypen nach IEC 61131 (Auswahl).

4.2 Das Kommunikationsmodell

Das Kommunikationsmodell der IEC 61131 beschreibt, wie der Datenaustausch *innerhalb* eines Programms und *zwischen* Programmen erfolgt.

4.2.1 Datenaustausch innerhalb eines Programms

Die Kommunikation zwischen den Programmen, die in den Programmorganisationseinheiten POEen formuliert wurden erfolgt i.d.R. über globale Variablen (VAR_GLOBAL). Das gilt im Übrigen auch bei der Kommunikation zwischen mehreren SPSen bzw. SPSen und Visualisierung etc.

Innerhalb eines Programms, also einer POE, erfolgt die Kommunikation zwischen den Funktionen FCs untereinander, den Funktionsbausteinen FBs untereinander sowie zwischen

Funktionen und Funktionsbausteinen über die Ein- und Ausgangsvariablen (VAR_INPUT, VAR_OUTPUT, VAR_IN_OUT) der Bausteine. Einer Funktion müssen dabei genau genommen nur die Eingangsvariablen (VAR_INPUT) übergeben werden, da der Rückgabewert der Funktion ja direkt im aufrufenden Funktionsbaustein oder Programm verarbeitet wird. Eine Verknüpfung über globale Variablen (VAR_GLOBAL) ist natürlich immer möglich, zeugt jedoch an dieser Stelle von einem schlechten Programmierstil.

Eine Besonderheit beim Aufruf eines Funktionsbausteins soll an dieser Stelle nicht unerwähnt bleiben. Wird ein Funktionsbaustein in einem Programm aufgerufen, so wird genau genommen eine Instanz dieses Funktionsbausteins, also eine Art Kopie, aufgerufen. Die Instanz ist als Variable vom Typ des Funktionsbausteins zu deklarieren. Die deklarierten Variablen eines Funktionsbausteins gelten für die aufgerufene Instanz und sind über die gesamte Laufzeit gültig. Wird der gleiche Funktionsbaustein an verschiedenen Stellen eines SPS-Programms aufgerufen, so werden mehrere unabhängige Instanzen dieses Bausteins ausgeführt, die alle unabhängig voneinander ablaufen und sich in keiner Weise gegenseitig beeinflussen.

Beispiel 04.05: Datenaustausch innerhalb eines Programms

Es soll die mathematische Funktion $F1 = \frac{A \cdot B}{C}$ vom Funktionsbaustein `FB1` aufgerufen und dem Programm `PRG1` Werte übergeben werden (siehe Abb. 04.10).

Zum Aufruf des Funktionsbausteins `FB1` im Programm `PRG1` ist dessen Instanz `INSTANZ1` als Variable vom Typ `FB1` zu deklarieren. Der Funktionsbaustein wiederum soll in diesem Beispiel die Funktion `F1` aufrufen, die eine arithmetische Funktion durchführt.

Das Beispiel zeigt, wie die vom Programm vorgegebenen Werte `WERT1=2.0`, `WERT2=10.0` und `WERT3=2.5` über die Eingangsvariablen `X`, `Y`, und `Z` des Funktionsbausteins an die Eingangsvariablen `A`, `B` und `C` der Funktion übergeben werden.

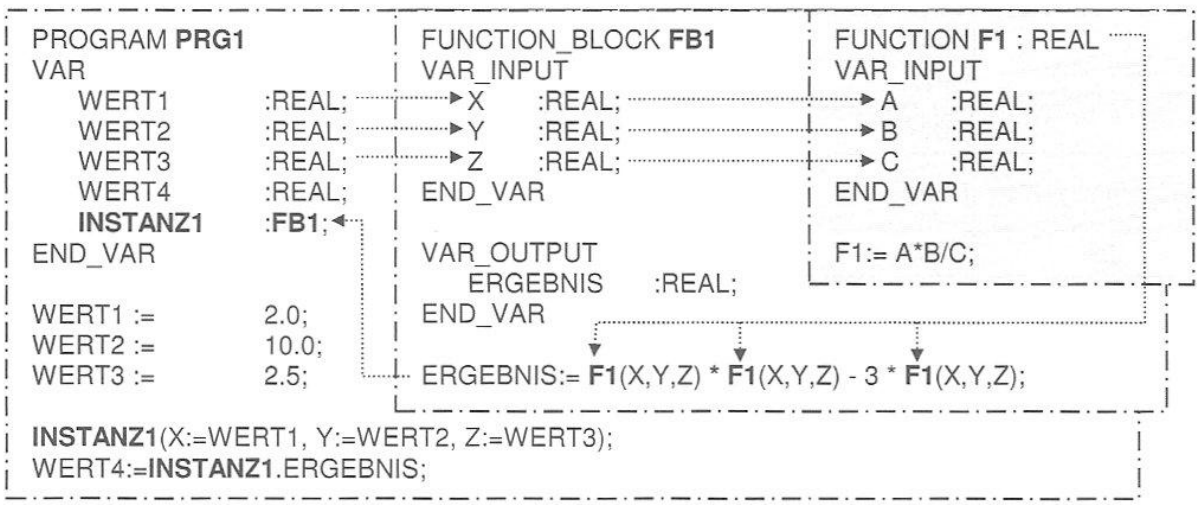


Abb. 04.10 Beispiel für das Aufrufen und Auslesen von Funktionsbausteinen und Funktionen. Durch Übergabe der Werte berechnet das Programm `WERT4=40`.

Der Rückgabewert der Funktion ergibt sich zu $F1 = \frac{A \cdot B}{C} = \frac{2{,}0 \cdot 10{,}0}{2{,}5} = 8$.

Die Funktion `F1` wird im Funktionsbaustein `FB1` weiterverarbeitet zum Ausgabewert **`INSTANZ1.`**`ERGEBNIS`.

$$ERGEBNIS = F1^2 - 3 \cdot F1 = 8^2 - 3 \cdot 8 = 64 - 24 = 40$$

Das Programm `PRG1` liest nun diesen Wert durch den Konstrukt **`INSTANZ1.`**`ERGEBNIS` aus dem Funktionsbaustein `FB1` aus und weist den Wert der Variablen `WERT4` zu.

```
WERT4 := INSTANZ1.ERGEBNIS
WERT4  = 40
```

4.2.2 Datenaustausch zwischen Programmen

Die Kommunikation zwischen verschiedenen Programmen kann nach IEC 61131 nur mittels *globaler Variablen* (VAR_GLOBAL) realisiert werden. Das unterstützt nicht gerade einen guten Programmierstil. Globale Variablen schränken die Übersichtlichkeit ein und führen zu unerwünschten Effekten, wenn Variablen mehrfach, d.h. lokal und global, deklariert werden. Jeder Programmierer sollte sich bemühen, die Anzahl der globalen Variablen innerhalb eines Projekts so gering wie möglich zu halten, also auf das notwendige Minimum zu beschränken.

Werden in einem großen Projekt sehr viele Variablen zwischen einzelnen Programmen (POEen) einer SPS ausgetauscht oder enthält das Projekt mehrere SPSen zwischen denen sehr viele Variablen übertragen werden müssen, so lässt sich eine gewisse Unübersichtlichkeit bei den globalen Variablen nicht vermeiden. Umso wichtiger ist hierbei eine überlegte und systematische Planung der Variablennamen.

Beispiel 04.06: Datenaustausch zwischen Programmen

Der programmübergreifende Datenaustausch soll am nachfolgenden Beispiel dargestellt werden. Abb. 04.11 zeigt den Datenaustausch zwischen den Programmen `P1` und `P2` .

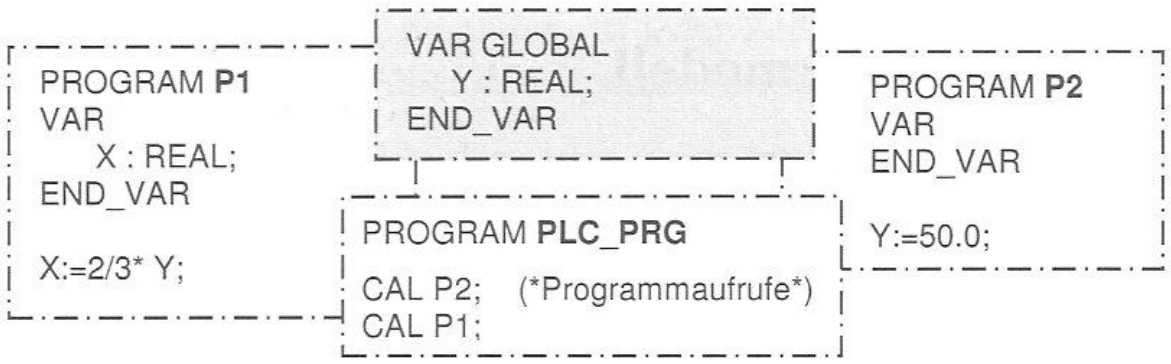


Abb. 04.11 Beispiel für den Datenaustausch zwischen zwei Programmen, die beide vom Programm `PLC_PRG` aufgerufen werden.

Im Programm P2, das zuerst, also vor Programm P1, aufgerufen wird, wird der globalen Variablen Y der Wert 50.0 zugewiesen. Der Wert von Y=50.0 ist jetzt für alle anderen Programme verfügbar. Im Programm P1 wird nun der Wert von Y entsprechend dem Programmcode verarbeitet.

$$\text{X := 2/3 Y} \quad \rightarrow \quad X = \frac{2}{3} \cdot Y = \frac{2}{3} \cdot 50{,}0 = 33{,}3333333333$$

Signalmodell einer SPS

Prinzipiell empfiehlt es sich, zwischen folgenden *Signaltypen* von Programmen zu unterscheiden:

- Der programmübergreifende Datenaustausch erfolgt durch die *Steuersignale* ***u*** und die *Statussignale* ***v***, die jeweils global zu deklarieren sind.
- Dagegen werden die *Messsignale* $\boldsymbol{x}_e$ von den Sensoren über die SPS-Eingangskanäle eingelesen und die *Stellsignale* $\boldsymbol{y}_a$ von den Programmen über die SPS-Ausgangskanäle an die Aktoren übertragen.

Die Statussignale ***v*** eines Programms (z.B. Variable `Y` des Programms `P2` in Abb. 04.11) können als Steuersignale ***u*** eines anderen Programms (z.B. in `P1`) verwendet werden und verknüpfen somit die beiden Programme. Abb. 04.12 zeigt, dass jedes Programm über seine Steuer- und Statussignale mit anderen Programmen und durch Mess- und Stellsignale mit den Sensoren und Aktoren kommunizieren kann.

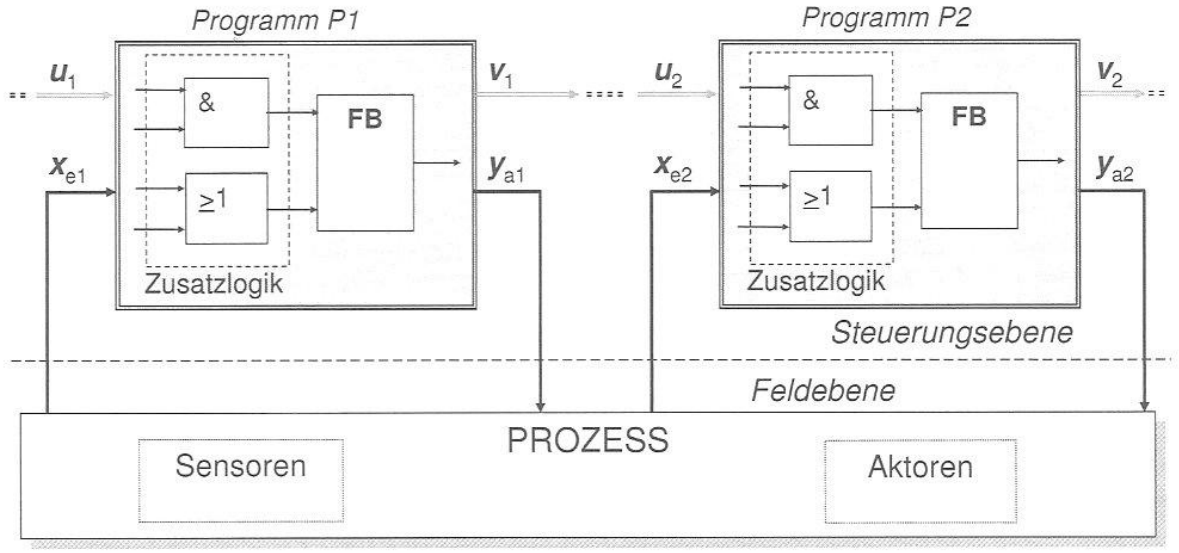


Abb. 04.12 Programme erhalten durch die Messsignale $\boldsymbol{x}_e$ Informationen vom Prozess und beeinflussen den Prozess über die Stellsignale $\boldsymbol{y}_a$. Der Datenaustausch zwischen den Programmen erfolgt durch die SPS-internen Steuersignale ***u*** bzw. die Statussignale ***v***.

Für den programmübergreifenden Datenaustausch sind die Steuer- und Statussignale ***u*** bzw. ***v*** als *globale Variablen* zu deklarieren. Dagegen können die Mess- und Stellsignale $\boldsymbol{x}_e$ bzw. $\boldsymbol{y}_a$ lokal deklariert werden. Sie müssen jedoch die vorgegeben E/A-Kanäle adressieren, um mit dem Prozess zu kommunizieren (z.B. `AT %IX0.0`).

4.3 Das Programmiermodell

Das Programmiermodell beschreibt, wie eine *Anwendersoftware* erstellt werden soll. Bei der Erstellung der Anwendersoftware besteht die Möglichkeit, neben den in der IEC-Norm definierten Standard-Funktionen und –Funktionsbausteinen eigene, sogenannte Anwender-Funktionen und –Funktionsbausteine, zu entwickeln. Davon wird in Projekten der verschiedenen Branchen auch rege Gebrauch gemacht. Die Syntax zur Erstellung dieser Anwender-Funktionen und –Funktionsbausteine wurde im Beispiel 04.05 veranschaulicht. Verwendet wurde in diesem Beispiel die Programmiersprache *Strukturierter Text (ST)*.

Außer dem Strukturierten Text (ST) sind in der IEC-Norm noch weitere Programmiersprachen definiert: *Anweisungsliste (AWL)*, *Funktionsbausteinsprache oder Funktionsplan (FBS oder FUP)*, *Kontaktplan (KOP)* und *Ablaufsprache (AS)*. Diese werden in Abb. 04.13 dargestellt.

4.3.1 Programmiersprachen

Die Einigung auf diese Sprachen ist historisch begründet. So bildet der *Kontaktplan (KOP)*, im Englischen *Ladder Diagramm (LD)*, die Relais-Schaltung früherer verbindungsprogrammierter Steuerungen nach. Er wird im amerikanischen Raum auch heute noch gern und oft eingesetzt. In Europa wird als grafische Sprache hauptsächlich die *Funktionsbausteinsprache (FBS)*, im Englischen *Function Block Diagram (FBD)*, verwendet, die an Blockschaltbilder und Logikpläne angelehnt ist.

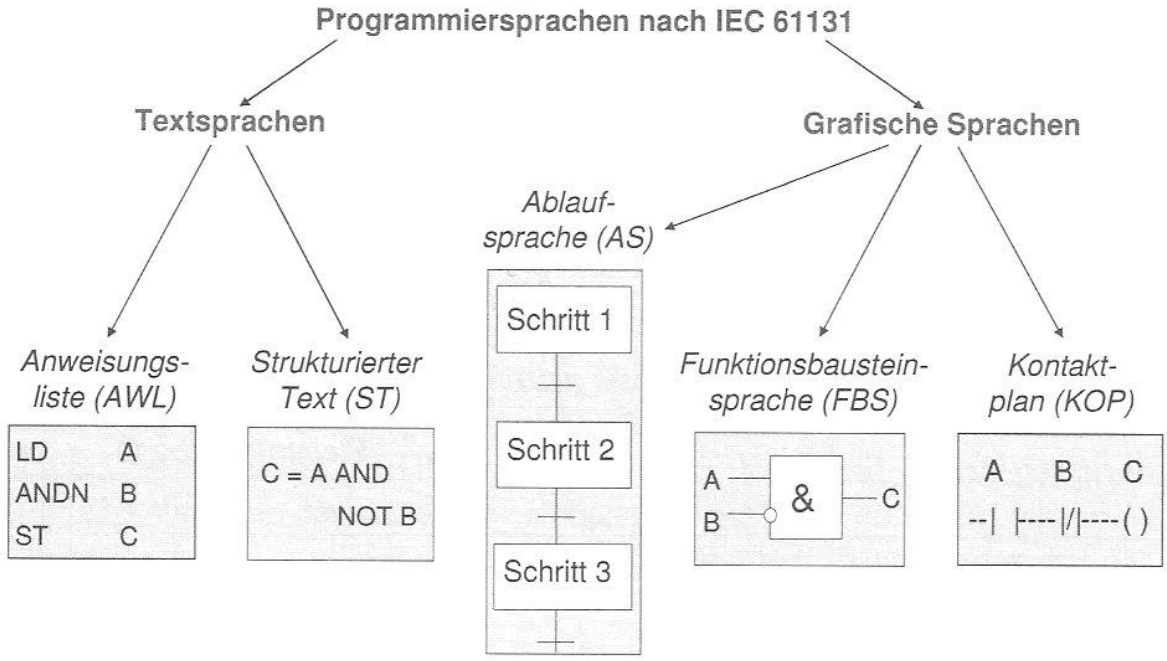


Abb. 04.13 SPS-Programmiersprachen nach IEC 61131.

4.3.1.1 Textsprachen

Von den Textsprachen verwenden die Steuerungstechniker in Europa bis heute gerne die *Anweisungsliste (AWL)*, im Englischen *Instruction List (IL)*. Sie ist der Assemblerprogrammierung sehr ähnlich und wurde in den Anfangszeiten der SPS eingesetzt um die Rechenzeit gering zu halten. Wie in Abb. 04.13 zu sehen ist, enthält jede Zeile genau eine Anweisung. Jede Anweisung besteht aus einem Operator gefolgt von einem Operanden. Bei jeder Operation wird das aktuelle, im Arbeitsspeicher (auch Akku genannt) vorhandene Ergebnis über den Operator mit dem Operanden verknüpft.

Im Beispiel aus Abb. 04.13 wird in der ersten Zeile der Anweisungsliste der Operand `A` durch den Operator `LD` in den Akku geladen. In der zweiten Zeile erfolgt eine UND-Verknüpfung mit dem negierten Operanden `B`. Das Ergebnis dieser Verknüpfung wird schließlich in der dritten Zeile durch den Operator `ST` in der Variablen `C` abgespeichert.

Dadurch entstehen für komplexere Operationen sehr lange und unübersichtliche Listen, mitunter sogar mit Sprüngen, was für Wartungs- und Planungspersonal sehr schlecht nachvollziehbar ist.

In Tabelle 04.05 sind spezielle AWL-Operatoren (Anweisungen) aufgelistet. Darüber hinaus können auch die IEC-Funktionen, z.B. `AND` aus Abb. 04.13, als Operatoren verwendet werden, wobei durch den Modifikator `N` die Operanden negiert werden können.

Operator	Beschreibung
LD	lädt den Wert einer Variablen in den Arbeitsspeicher.
LDN	lädt den Wert einer Variablen negiert in den Arbeitsspeicher.
ST	speichert aktuelles Ergebnis in einer Variablen.
STN	speichert aktuelles Ergebnis negiert in einer Variablen.
S	setzt Boole'sche Variable auf TRUE, wenn aktuelles Ergebnis TRUE ist.
R	setzt Boole'sche Variable auf FALSE, wenn aktuelles Ergebnis TRUE ist.
JMP	Sprung zur Marke (≡ Text mit nachfolgendem Doppelpunkt zu Beginn einer Zeile).
JMPC	Sprung zur Marke, wenn das Ergebnis TRUE ist.
JMPN	Sprung zur Marke, wenn das Ergebnis FALSE ist.
CAL	Aufruf eines Programms, eines Funktionsbausteins oder einer Funktion.
CALC	Aufruf, wenn das Ergebnis TRUE ist.
CALN	Aufruf, wenn das Ergebnis FALSE ist.
RET	Rücksprung von Funktion oder Funktionsbaustein.
RETC	Rücksprung, wenn das Ergebnis TRUE ist.
RETN	Rücksprung, wenn das Ergebnis FALSE ist.

Tabelle 04.05 Operatoren (Anweisungen) für die Anweisungsliste (AWL).

	Befehl	Beschreibung (Beispiel)
Auswahlbefehle	IF	IF a < b THEN c:= 1; ELSIF a = b THEN c:= 2; ELSE c:= 3; END_IF
	CASE	CASE f OF 1: a:=3; 2: a:=4; ELSE a:=0; END_CASE;

Schleifen	FOR	FOR a:= 1 TO 10 BY 2 DO
		f[a] := b;
		END_FOR;

	WHILE	WHILE b > 1 DO
		b:=b/2;
		END_WHILE

	REPEAT	REPEAT a:= a * b; UNTIL a > 10000 END_REPEAT;
Abbruch	EXIT	Beendigung einer Schleife unabhängig von der Abbruchbedingung
Rücksprung	RETURN	Rücksprung aus einer Funktion oder einem Funktionsbaustein
Programmaufruf		PRG1();
Funktionsbau-steinaufruf		FB1(IN:=a, PT:=t#5s); c:=FB1.Q; (s. auch Bild 3.10)
Funktionsaufruf		F1(X,Y,Z); (s. auch Bild 3.10)

Tabelle 04.06 Befehle für den Strukturierten Text (ST).

Heute spielt die Rechenzeit eine immer kleinere Rolle, so dass die Hochsprachen wie Pascal, C oder *Strukturierter Text (ST)* sowohl zur Programmierung von Mikrocomputern als auch von SPSen immer mehr durchsetzen. In den Beispielen 04.05 … 04.07 kann man sehen, dass der Strukturierte Text große Ähnlichkeit mit der Hochsprache C oder Pascal hat. Im Strukturierten Text werden Befehle wie in Hochsprachen üblich bedingt (z.B. durch `IF..THEN..ELSE`) und /oder Schleifen (z.B. mit `FOR..TO..DO`) ausgeführt. Jeder Befehl endet mit einem Semikolon. Diese speziellen ST-Befehle sind in Tabelle 04.06 zusammengestellt.

4.3.1.2 Grafische Sprachen

Die Funktionsbausteinsprache kann Funktionen und Funktionsbausteinen wie in einem Blockschaltbild anordnen. Sie eignet sich sehr gut zur Darstellung von Regelungen und binäre Logikschaltungen. Diese Darstellung wird weitläufig auch als *Funktionsplan (FUP)* bezeichnet.

Eine Sonderstellung nimmt die *Ablaufsprache (AS)*, im Englischen *Sequential Function Chart (SFC)* ein, denn mit ihr können nur sequentielle Steuerungsaufgaben in Form von Schrittketten programmiert werden.

4.3.1.3 Empfehlungen für den Einsatz der SPS-Programmiersprachen

Generell kann man sagen, dass die Textsprachen *unübersichtlicher* sind als die grafischen Sprachen. Zur Entwicklung umfangreicher numerischer Algorithmen eignen sich Textsprachen aber häufig besser, da mit ihnen *effizienter* programmiert werden kann.

Da SPSen jedoch stets Anlagen steuern, die gewartet und ab und zu umkonfiguriert werden müssen, ist die *Verständlichkeit* und *Übersichtlichkeit* der Software oberstes Gebot. Deshalb sollten übergeordnete Programme übersichtlich und änderungsfreundlich gestaltet und mit *grafischen Sprachen*, wie z.B. FBS oder AS, programmiert werden.

Es wird folgender Einsatz der Programmiersprachen empfohlen:

- *Prozessabläufe* lassen sich sehr gut als Schrittketten beschreiben und werden deshalb in der *Ablaufsprache AS* programmiert,
- *Ansteuerprogramme* für Aktoren und Sensoren bestehen zumeist aus Verknüpfungslogik und werden deshalb in der *Funktionsbausteinsprache FBS* programmiert,
- *Anwender-Funktionen und –Funktionsbausteine*, die das Wartungspersonal in der Regel ohnehin als Black-Box ansieht, können effizient in Textsprachen wie *AWL* oder *ST* programmiert werden.

4.3.2 Anwender-Datentypen

Für eine übersichtliche Programmstrukturierung hat der Anwender die Möglichkeit, eigene Anwender-Datentypen mit dem Schlüsselwort `TYPE` zu definieren. Anwender-Datentypen können für verschiedene Zwecke verwendet werden. Tabelle 04.07 zeigt eine Übersicht und im folgenden Beispiel 04.07 wird das verdeutlicht.

Datentyp	Erläuterung	Beispiel
Referenz	Alternativer Name für Standard-Datentyp.	TYPE *Helligkeit* : USINT; END_TYPE
Aufzählung	Aufzählung von String-Konstanten.	TYPE *Ampeldaten* : (Rot,Gelb,Gruen); END_TYPE
Unterbereiche	Datentyp, dessen Wertebereich nur eine Untermenge eines Basistypen umfasst.	TYPE *SubInt* : INT (-4095..4095); END_TYPE
Datenfeld	Vektoren oder Matrizen elementarer Datentypen.	VAR *Bild* : ARRAY[0..8] OF *Helligkeit*; END_VAR
Struktur	Zusammenfassung ungleicher Datentypen.	TYPE *Merkmale* : STRUCT X_POS : REAL; Y_POS : REAL; GROESSE : INT; END_STRUCT END_TYPE

Tabelle 04.07 Anwender-Datentypen.

Beispiel 04.07: Anwender-Datentypen zur Auswertung einer Sensormatrix

Der Zweibackengreifer eines Roboterarms in Abb. 04.14 soll einen Gegenstand sicher greifen. Hierfür ist es notwendig, dass der Gegenstand möglichst mittig zwischen den beiden Greiferbacken sitzt. Um das gewährleisten zu können, sind in der linken Greiferbacke Lichtquellen angebracht und in der rechten Greiferbacke 9 lichtempfindliche Sensoren.

Gelangt nun ein zu greifendes Objekt wie in Abb. 04.14b zwischen die Greiferbacken, wird es von Licht bestrahlt. Das Licht fällt infolge des Objektschattens nicht auf alle Sensoren. Jeder Sensor überträgt einen Helligkeitswert zwischen 0 und 255 an die SPS. Die Sensoren, die nur wenig Licht empfangen, senden einen niedrigen Helligkeitswert, die anderen einen hohen Helligkeitswert an die SPS. In der SPS soll ein Programm den Schatten eines Objektes anhand der Helligkeitswerte erkennen.

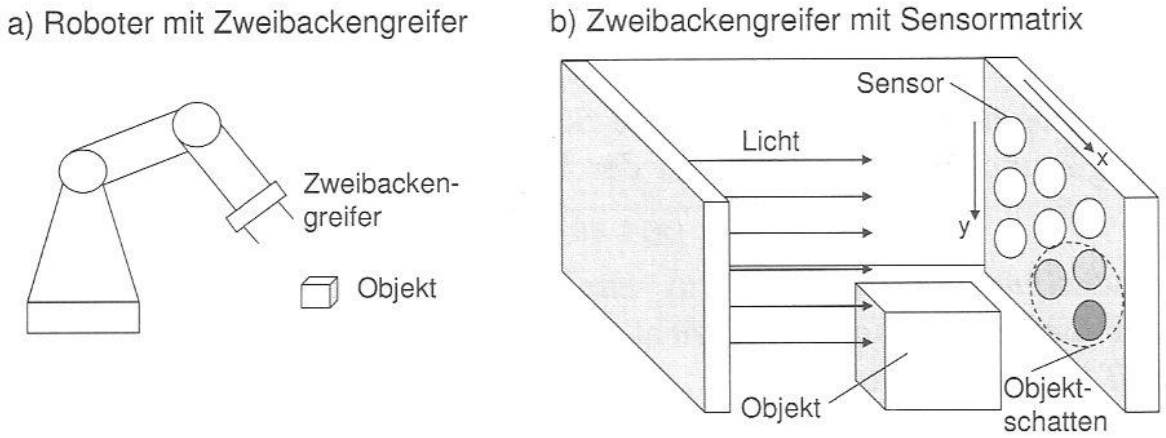


Abb. 04.14 Um ein Objekt mittig zu greifen, muss sich der Zweibackengreifer eines Roboterarms (a) so an das Objekt annähern, dass der Objektschatten in der Mitte der Sensormatrix liegt (b).

Zur Auswertung der Matrix wird der neue Datentyp `Helligkeit` (siehe Tabelle 04.07) eingeführt, mit dem die Helligkeit der Bildpunkte in einer Auflösung von 8 Bit beschrieben wird (Wertebereich 0 … 255).

Für die Sensormatrix wird der Datentyp `Bild` als Array mit 9 Elementen vom Typ `Helligkeit` eingeführt. In diesem Array sind die Helligkeitswerte der Bildpunkte von links nach rechts beginnend in der obersten Zeile aneinander gereiht.

Dunkle Bildpunkte ($x_i|y_i$), deren Helligkeitswerte kleiner als 50 (Schwellwert) sind, bilden auf der Sensormatrix einen Objektschatten. Sein Flächenschwerpunkt ergibt sich durch die Mittelwerte der x- und y-Koordinaten der dunklen Bildpunkte:

$$x_{POS} = \frac{1}{N}\sum_{i=0}^{N-1} x_i \qquad y_{POS} = \frac{1}{N}\sum_{i=0}^{N-1} y_i \qquad \text{(Gl. 04.05)}$$

Dabei ist N die Anzahl der Zeilen bzw. der Spalten der Sensormatrix, d.h. in diesem Beispiel ist N = 3. Der Flächenschwerpunkt ($x_{POS}|y_{POS}$) wird im folgenden Programm im Datentyp Merkmale angegeben, der eine Struktur aus der x- und y-Position des Objektes darstellt.

Das in Abb. 04.15 dargestellte Programm durchläuft also die Sensormatrix zeilenweise von links nach rechts und prüft, ob ein Sensorwert kleiner als 50 ist. Für die abgebildete Sensormatrix würden sich beispielsweise die Sensoren bei (x|y) = (2|1), (1|2), (2|2) finden. Ihr Flächenschwerpunkt ergibt sich nach Gl. 04.05 an der Position (1,67|1,67)

4.3.3 Anwender-Funktionsbausteine

Neben der Verwendung von IEC-Standard-Funktionsbausteinen, wie Counter, Timer, Speicher, hat der Anwender auch die Möglichkeit, eigene sogenannte *Anwender-Funktionsbausteine* zu erzeugen. Wie jeder Funktionsbaustein können auch die Anwender-Funktionsbausteine mehrfach in Programmen eingesetzt werden. Diese Wiederverwendbarkeit von Funktionsbausteinen erlaubt es, die Software großer Projekte übersichtlich und *modular* zu strukturieren, wodurch sich der Programmcode reduziert und der Aufwand für Engineering und Test verringert.

Die Kunst der *modularen Programmstrukturierung* besteht darin, Programmteile zu finden, die sich mehrfach wiederholen, und diese als Anwender-Funktionsbausteine zu programmieren.

Einen Anwender-Funktionsbaustein würde man in der Informatik als *Klasse* mit bestimmten Eigenschaften bezeichnen, die in der Logik des Bausteins enthalten sind. Durch Einfügen des Anwender-Funktionsbausteins in ein Programm entsteht ein *Objekt* oder eine sogenannte *Instanz* des Bausteins. Instanzen müssen in den Programmen, in denen sie aufgerufen werden, wie Variablen mit Name und Typ deklariert werden (z.B. `INSTANZ1:KLASSE_A` in Abb. 04.16).

Man kann eine Instanz als Kopie oder spezielles Exemplar (Objekt) des Funktionsbausteins ansehen. In einer Software kann es mehrere Instanzen einer Klasse geben. Deshalb ist eine wichtige Eigenschaft der *Instanzierung*, dass die eingefügten Instanzen unabhängig voneinander ablaufen können und sich nicht gegenseitig beeinflussen. Beispielsweise bestehen `INSTANZ1` und `INSTANZ2` in Abb. 04.16 beide aus dem Baustein `KLASSE_A`. Im Programm `P1` wird `INSTANZ1` aber im Allgemeinen mit anderen Werten abgearbeitet als `INSTANZ2` im Programm `P2`.

```
PROGRAM OBJEKT_Position
  VAR
    SENSOR : Bild := 218, 223, 243,
                     234, 198,  32,
                     187,  17,  37;    (* eingelesene Sensormatrix *)
    OBJEKT  : Merkmale;      (* Objektmerkmale *)
    x: INT;                  (* Laufvariable für x-Komponente *)
    y: INT;                  (* Laufvariable für y-Komponente *)
  END_VAR

OBJEKT.GROESSE:= 0;                (* Initialisierung *)
OBJEKT.X_POS:= 0.0;
OBJEKT.Y_POS:= 0.0;

FOR y:=0 TO 2 DO                   (* Bildzeilen *)
    FOR x:=0 TO 2 DO               (* Bildspalten *)
      IF SENSOR[ y + 3 * x ] < 50 THEN          (* Nur dunkle Bildpunkte stellen Objektschatten dar *)
          OBJEKT.X_POS:= OBJEKT.X_POS +x; (* Summe der x-Positionen der dunklen Pixel *)
          OBJEKT.Y_POS:= OBJEKT.Y_POS +y; (* Summe der y-Positionen der dunklen Pixel *)
          OBJEKT.GROESSE:= OBJEKT.GROESSE + 1;  (* Anzahl der dunklen Pixel *)
      END_IF
    END_FOR
END_FOR

IF OBJEKT.GROESSE > 0    THEN              (* Wurde ein Objekt detektiert? *)
          OBJEKT.X_POS:= OBJEKT.X_POS / OBJEKT.GROESSE ;        (* mittlere x-Position *)
          OBJEKT.Y_POS:= OBJEKT.Y_POS / OBJEKT.GROESSE ;        (* mittlere y-Position *)
ELSE
          OBJEKT.X_POS:= -10.0;             (* x-Position negativ, d.h. nicht definiert *)
          OBJEKT.Y_POS:= -10.0;             (* y-Position negativ, d.h. nicht definiert *)
END_IF
```

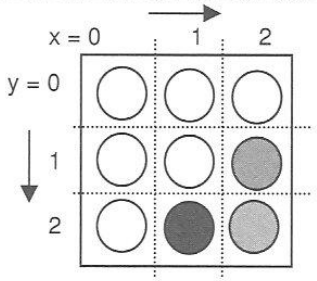


Abb. 04.15 Programm zur Auswertung der Sensormatrix aus Beispiel 04.07.

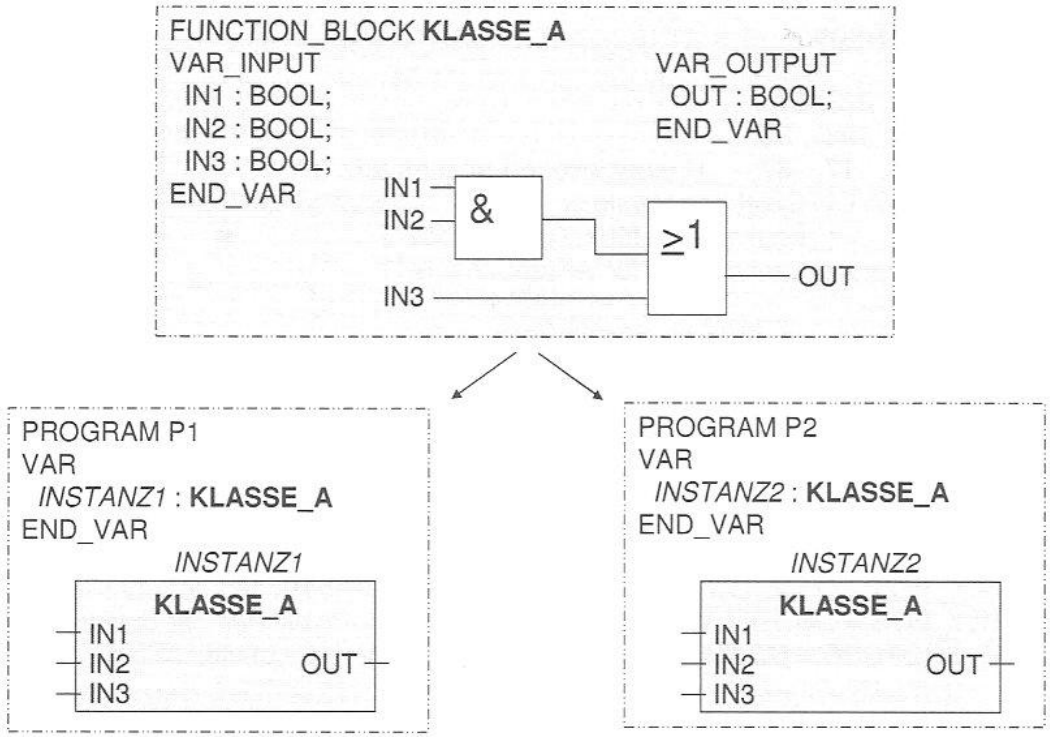


Abb. 04.16 Instanzierung des Funktionsbausteins `KLASSE_A` als `INSTANZ1` und `INSTANZ2` in verschiedenen Programmen.

4.4 Leitfaden für die Softwarestrukturierung

Innerhalb großer Automatisierungsprojekte müssen hunderte von Datenpunkten (Sensoren und Aktoren) aufgenommen, verarbeitet und angesteuert werden. Die daraus resultierenden Softwareprojekte sind entsprechend umfangreich und müssen transparent gestaltet werden. Eine objektorientierte Strukturierung ist deshalb nicht nur sinnvoll, sondern auch angeraten.

4.4.1 Objektorientierte Softwarestrukturierung

Objektorientiertheit bedeutet, die *Objekte der realen Welt in der Software nachzubilden*. In der Fabrik (Fertigungs- oder Prozessautomatisierung) sind die Objekte die Feldgeräte innerhalb der Anlage wie z.B. Motor EU4711 in Abb. 04.17. Die Feldgeräte unterscheiden sich i.d.R. nur durch relativ wenige Feldgerätetypen voneinander, lassen sich also ordnen und gruppieren.

Die Anlagenstruktur wird durch folgende Modularisierung in der Anwendersoftware nachgebildet:

- Für jedes Feldgerät innerhalb der Anlage wird ein eigenes *Ansteuerprogramm* in der SPS erstellt.
- Für die Daten jedes Feldgeräts wird eine *globale Strukturvariable* angelegt.
- Für jeden Feldgerätetyp, z.B. Motor, Ventil, Regler oder Messsonde, wird mindestens ein *Anwender-Funktionsbaustein* entwickelt.
- Für die Daten jedes Feldgerätetyps wird ein *Anwender-Datentyp* angelegt.

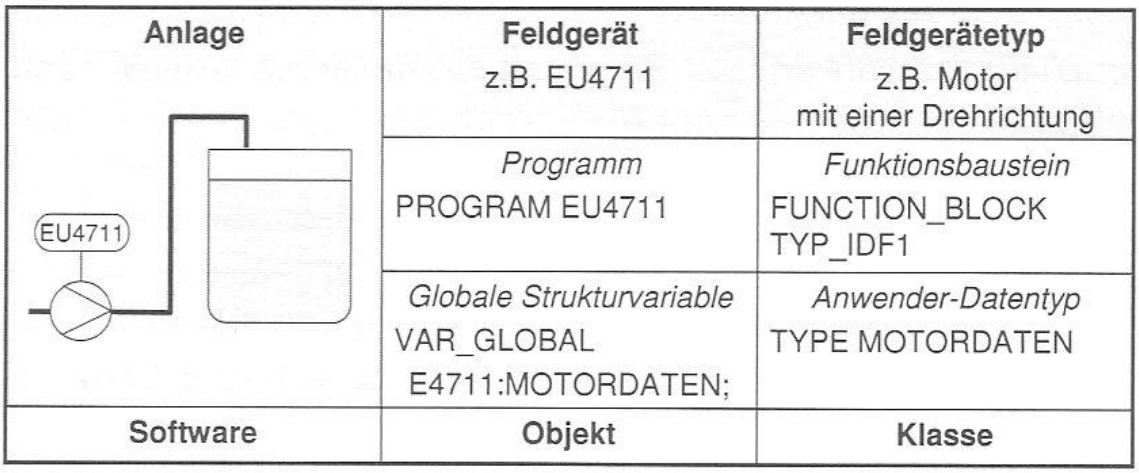


Abb. 04.17 Objektorientierte Softwarestrukturierung am Beispiel einer Pumpe.

Durch die Vielzahl der Feldgeräte entstehen so zwar viele Programme und globale Variablen. Die Liste der Programme und die der globalen Variablen entsprechen dabei aber exakt der Feldgeräteliste (Messstellen- oder Datenpunktliste) des Anlagenplaners.

Somit erreicht man eine eindeutige Abbildung zwischen der *realen Welt* in der Anlage und der *Steuerungssoftware*, was sehr zur Verständlichkeit der SPS-Programme beiträgt. Ziel ist es jedoch, mit möglichst wenigen Funktionsbausteinen (Klassen) eine Vielzahl von Geräten (Objekten) ansteuern zu können.

4.4.2 Vorgehensweise bei der strukturierten Programmierung

Auf der Basis der objektorientierten Softwarestrukturierung hat sich bei der Entwicklung der Automatisierungssoftware ein Vorgehen anhand der folgenden fünf Stufen als zweckmäßig erwiesen:

Stufe 1: Softwarestrukturierung

a) Pro *Feldgerätetyp* werden ein *Funktionsbaustein* und ein Anwender-Datentyp definiert.

b) Pro *Feldgerät* sind ein *Ansteuerprogramm* und eine globale *Strukturvariable* vorzusehen.

Stufe 2: Entwurf der Funktionsbausteine und Funktionen

a) Deklaration der *Ein-* und *Ausgangsvariablen*.

b) Entwurf der *Schaltungslogik*.

Stufe 3: Entwurf der Programme

a) Deklaration der Variablen für *Mess-* und *Stellsignale* $\boldsymbol{x_e}$, bzw. $\boldsymbol{y_a}$, und ihrer *E/A-Zuordnung*.

b) Deklaration der globalen Variablen für *Steuer-* und *Statussignale* $\boldsymbol{u}$ bzw. $\boldsymbol{v}$ möglichst für jedem Gerätetyp gesammelt als globale Strukturvariable.

c) *Instanzierung* der Funktionsbausteine aus Stufe 2.

d) Entwurf der *Zusatzlogik*, die nicht im Funktionsbaustein vorhanden, aber zur Ansteuerung erforderlich ist.

Stufe 4: Implementierung der Software

a) *Implementierung* der entworfenen Programme, Funktionsbausteine, Datentypen und Variablen.

b) Festlegung der *Steuerungskonfiguration*.

c) *Taskzuordnung* der implementierten Programme.

Stufe 5: Test und Inbetriebnahme

a) *Test* der implementierten Software in der Simulation.

b) *Laden* der Software in die SPS.

c) *Inbetriebnahme* von Software und SPS im Zusammenspiel mit der Anlage.

d) *Protokollierung* der Tests anhand der Entwurfsdokumentation aus Stufe 2 und 3.

Diese eben gezeigte Vorgehensweise der strukturierten Programmierung wird in folgendem Beispiel veranschaulicht.

Beispiel 04.08: Vorgehensweise der strukturierten Programmierung am Beispiel einer Wasseraufbereitungsanlage

Für die in Abb. 04.18 dargestellte Anlage soll ein *Halb-Automatikbetrieb* programmiert werden. Dabei sollen die Geräte über die Anzeige- und Bedienoberfläche (Visualisierung) *manuell* bedient werden, um das erhitzte Schmutzwasser aus Behälter B in den Behälter C zu befördern. Danach soll Reinigungsmedium aus dem Behälter A hinzu dosiert werden, so dass sich ein gewünschter pH-Wert einstellt. Zum Schluss soll das neutralisierte Schmutzwasser abgelassen werden.

Die Programmierung der Software orientiert sich an den oben aufgeführten 5 Stufen.

Stufe 1: Softwarestrukturierung

Die Anlage besteht aus folgenden Gerätetypen:

- Einem *Antriebsmotor mit einer Drehrichtung* als Rührer `EU1`,
- drei *AUF-/ZU-Ventilen* `UV1`, `UV2` und `UV3`,

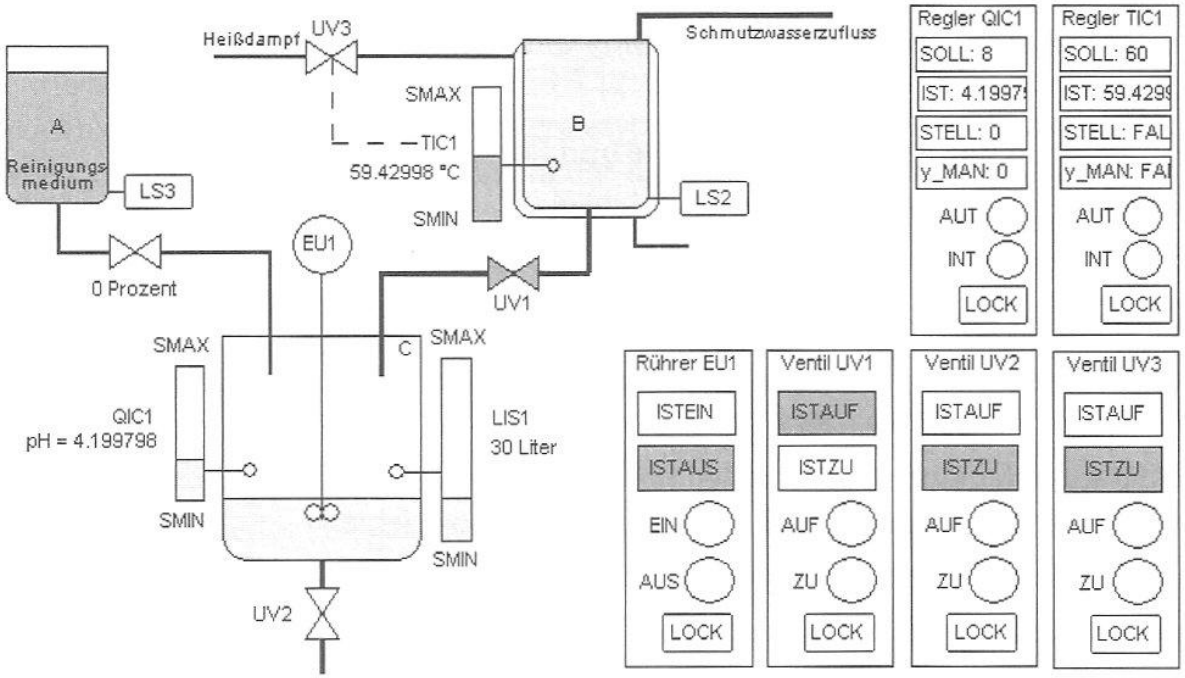


Abb. 04.18 Anzeige- und Bedienoberfläche einer Schmutzwasserreinigungsanlage.

- einer *analogen pH-Wertsonde* `QIC1`, mit der über ein *Regelventil* (Öffnungsgrad 0…100%) die Dosierung des Reinigungsmediums A in den Behälter C realisiert wird,
- zwei *Niveauschaltern* `LIS2` und `LIS3` an den Behältern A und B, die bei Erreichen des unteren Grenzwertes ein Bool‘sches Signal an die SPS melden,
- einer *analogen Standsonde* `LIS1`, die den analogen Füllstandswert im Behälter C misst,
- einem *Temperaturregler* `TIC1`, der Heißdampf in den Mantel des Behälters B führt, wenn die Temperatur unterhalb eines gewünschten Sollwertes liegt.

In Tabelle 04.08 wird jedem Feldgerät (Objekt) ein Gerätetyp (Klasse) zugeordnet, der als Funktionsbaustein zu programmieren ist. Eine kleine Ausnahme bilden die Reglerprogramme `QIC1` und `TIC1`, die außer den Reglerbausteinen auch den Funktionsbaustein `TYP_AIN` zum Einlesen des Istwertes umfassen.

Feldgerät (Objekt)		*Feldgerätetyp (=Typical, Klasse)*	
Programm	**Glob. Variable**	**Funktionsbaustein**	**Datentyp**
EU1	E1	TYP_IDF1 (Individual Drive Function)	Motordaten
LIS1	L1	TYP_AIN (Analog Input, *s. Beispiel 4.11)*	Analogdaten
LS2	L2	TYP_BIN (Binary Input, *s. Übung 4.10)*	Binärdaten
LS3	L3	TYP_BIN (Binary Input, *s. Übung 4.10)*	Binärdaten
QIC1	L1	TYP_AIN, TYP_PID, TYP_AOUT (PID-Regler, *s. Beispiel 4.13)*	Reglerdaten
TIC1	T1	TYP_AIN, TYP_IDF1 (Zweipunktregler mit Individual Drive Function, *s. Beispiel 4.12)*	Zweipunkt- Reglerdaten
UV1	U1	TYP_IDF1 (Individual Drive Function)	Ventildaten
UV2	U2	TYP_IDF1 (Individual Drive Function)	Ventildaten
UV3	U3	TYP_IDF1 (Individual Drive Function)	Ventildaten

Tabelle 04.08 Typ-Zuordnung für jedes Feldgerät.

Stufe 2: Entwurf der Funktionsbausteine und Funktionen

Vom Prinzip her sind die Funktionen des Motors `EU1` und der Ventile `UV1`, `UV2` und `UV3` zwar nicht identisch, aber sehr ähnlich. Die grundlegende Funktionalität umfasst das *Aktivieren* und *Deaktivieren* des Stellsignals über einen binären Ausgang im Feld. Dadurch wird ein Motor ein- oder ausgeschaltet bzw. ein Ventil auf- oder zugefahren. Deshalb kann für alle vier Aktoren der gleiche Funktionsbaustein `TYP_IDF1` (siehe Abb. 04.19) verwendet werden.

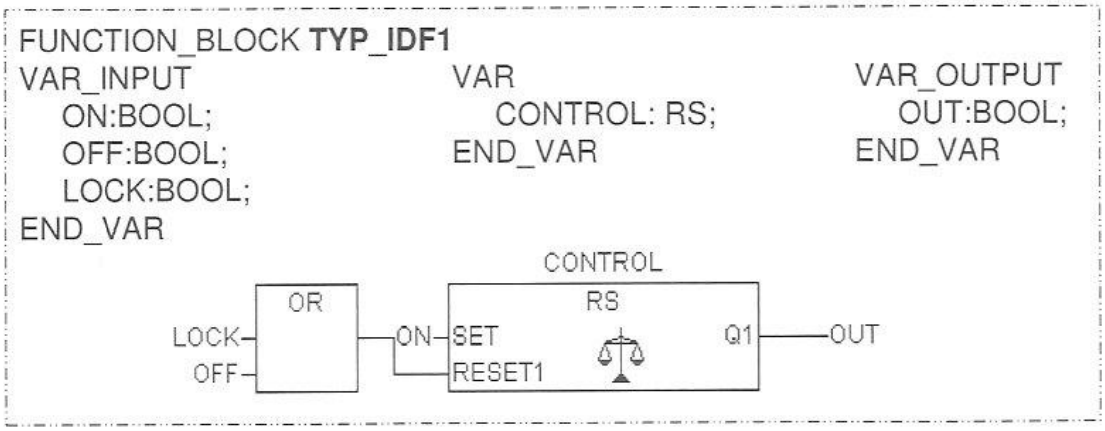


Abb. 04.19 Realisierung einer Typ-Zuordnung als Einzelsteuerfunktionsbaustein (Individual Drive Function, IDF) zur Steuerung eines Motors oder Ventils.

Die Eingangsvariablen `ON (EIN/AUF)` und `OFF (AUS/ZU)` des Bausteins aktivieren bzw. deaktivieren das RS-Flip-Flop, dessen Ausgangsvariable `OUT` als Stellsignal für das Motorschütz bzw. die Magnetventile dient. Darüber hinaus wird die Ansteuerung durch das Signal `LOOK` verriegelt und abgeschaltet, wenn z.B. ein Anlagen- oder Gerätefehler anliegt. Solange `LOOK = TRUE` ist, kann das Gerät nicht wieder aktiviert werden.

In diesem Beispiel soll zunächst nur diese sehr einfache Logik zur Ansteuerung des Motors und der Ventile zu Grunde gelegt werden.

Stufe 3: Entwurf der Programme

Das prinzipielle Verhalten der Aktoren `EU1`, `UV1`, `UV2` und `UV3` ist also gleich, so dass in diesen Programmen (siehe Abb. 04.20) der gleiche Funktionsbaustein vom Typ *Einzelsteuerfunktion* (Individual Drive Function, IDF) verwendet werden kann. Die Instanzierung eines Funktionsbausteins erfordert immer auch seine Deklaration als Variable, wie z.B. `VENTIL:TYP_IFD1`.

Die Stellsignale adressieren im Rahmen der Deklaration der lokalen Programmvariablen direkt SPS-Ausgangskanäle (z.B. `STELL AT%QX0.0:BOOL;`). Das Ein- und Ausschalten des Motors bzw. das Auf- und Zufahren der Ventile erfolgt über die Steuersignale `EIN/AUS` bzw. `AUF/ZU`. Der Status der Geräte wird durch die Signale `ISTEIN` bzw. `ISTAUF` nach außen bekannt gemacht. Da die Steuer- und Statussignale programmübergreifend verwendet werden, sind sie global zu deklarieren.

Um die Menge der globalen Variablen dennoch klein zu halten, empfiehlt es sich wie in Abb. 04.21, die Steuer- und Statussignale eines Feldgerätetyps in einer *Struktur* als Anwender-Datentypen zusammenzufassen. Hierfür werden die gerätespezifischen Strukturen `Motordaten`, `Ventildaten` und `Analogdaten` als Anwender-Datentypen definiert.

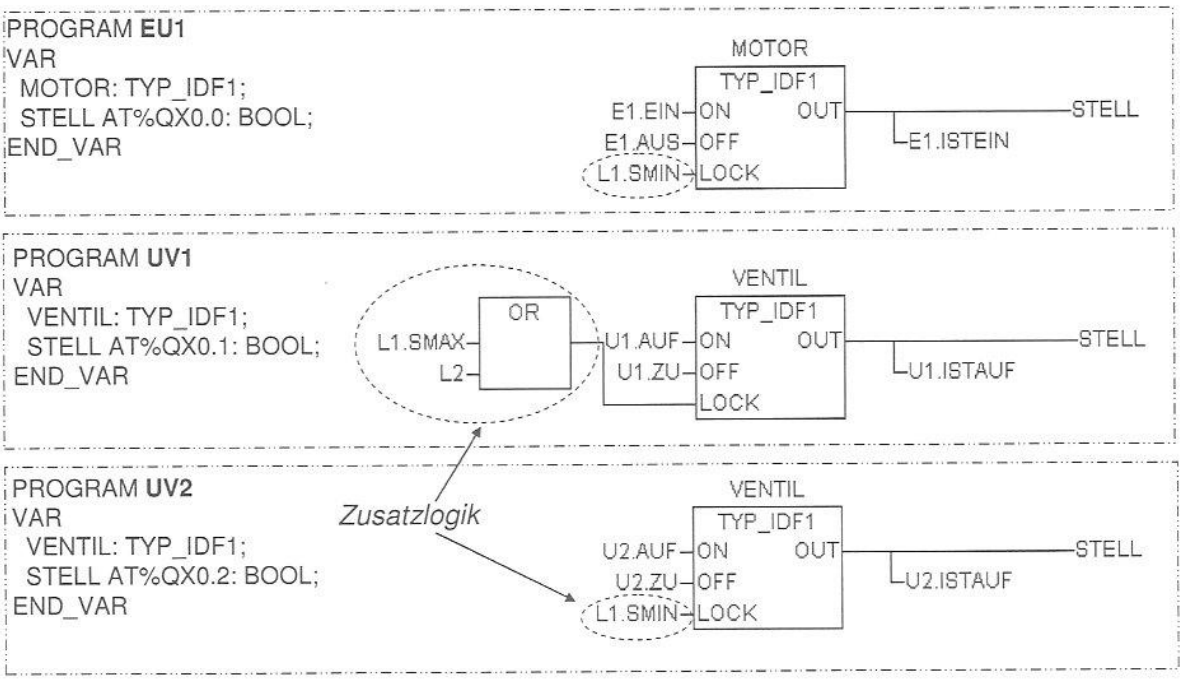


Abb. 04.20 Instanzierung des Funktionsbausteins `TYP_IDF1` in den Ansteuerprogrammen für die Feldgeräte `EU1`, `UV1` und `UV2`.

Genauso wie für jedes Feldgerät nach Tabelle 04.08 ein Programm geschrieben wird, sollte auch für jedes Feldgerät eine globale Strukturvariable deklariert werden, in der die Steuer- und Statusvariablen zusammengefasst sind. Der Aufruf dieser *globalen Strukturvariablen* erfolgt wie in den Programmen in Abb. 04.20 veranschaulicht z.B. durch `V1.AUF`, d.h. Variablenname und Strukturelement werden durch einen Punkt voneinander getrennt.

Somit beschreibt jedes Programm zusammen mit der zugehörigen Strukturvariablen die Eigenschaften eines Feldgerätes, wodurch die *realen Objekte* der Anlage transparent in der Software abgebildet werden.

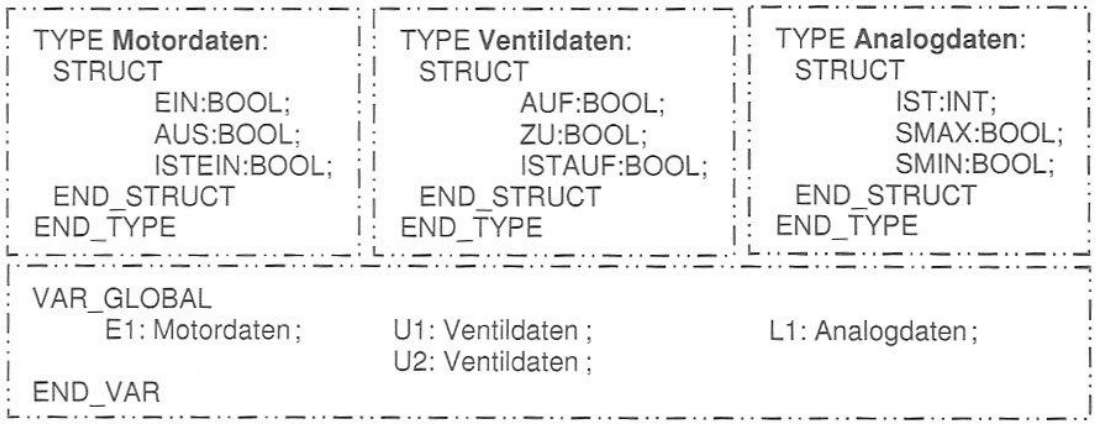


Abb. 04.21 Deklaration der Steuer- und Statussignale als globale Strukturvariablen in Form einer gerätespezifischen Struktur.

Entwurf einer Zusatzlogik:
Bislang umfassen die Ansteuerprogramme Standardeigenschaften der Feldgeräte wie das normale Ein- und Ausschalten. Nun sind aber häufig auch prozessbedingte Funktionen als *Zusatzlogik* in die Ansteuerprogramme einzubauen, die mitunter auch Signale anderer Programme erfordern. Tabelle 04.09 zeigt eine sogenannte Cause-Effekt-Matrix, in der die geforderten Programmverknüpfungen für das Beispiel zusammengestellt sind.

				Effekt		
Ursache			Aktor	EU1	UV1	UV2
Feld-gerät	Variable	Wert	Aktion / Prozesszustand	LOCK	LOCK	LOCK
LIS1	L1.SMIN	TRUE	Motor könnte ohne Widerstand der Flüssigkeit durchdrehen.	x		x
LIS1	L1.SMAX	TRUE	Behälter C könnte überlaufen.		x	
LS2	L2	TRUE	Behälter B ist leer.		x	

Tabelle 04.09 Cause-Effekt-Matrix zur Beschreibung kombinatorischer Schaltungen in der Anlage, z.B. zur Verriegelung einzelner Feldgeräte in einem bestimmten Prozesszustand.

Die diese Verriegelung auslösenden Statussignale `L1.SMIN` und `L1.SMAX` sowie `L2` werden in der Zusatzlogik der Programme `EU1`, `UV1` und `UV2` auf den `LOCK`-Eingang des Funktionsbausteins `TYP_IDF1` geschaltet (siehe Abb. 04.20).

Wie bei der Instanzierung von `TYP_IDF1` in den Programmen `EU1`, `UV1`, `UV2`, und `UV3` ist nun auch für die Instanzierung der Bausteine `TYP_PID`, `TYP_AIN` und `TYP_BIN` in die Programme `QIC1`, `LIS1` bzw. `LS2` und `LS3` vorzugehen.

Stufe 4: Implementierung der Software

Nach Implementierung der Funktionspläne nach Abb. 04.19 … 21 in das Programmiersystem (hier System CoDeSys 2.3.7.0) ist nun zunächst die CPU im Menü

`Zielsystemeinstellung` auszuwählen. Außerdem sind in der `Steuerungskonfiguration` die E/A-Karten gemäß Abb. 04.04 anzugeben.

Für dieses Beispiel sind dabei folgende E/A-Kanäle vorzusehen, die in der dezentralen Peripherie „ET-200M“ untergebracht sind und über Profibus DP in die Soft-SPS des PCs eingelesen werden (Profibuskarte „CIF 50-PB“):

- binäre Eingangskarte mit den Adressen *%IB0* und *%IB1* für die binären Eingänge von `L2` (%IX0.0) und `L3` (%IX0.1),
- binäre Ausgangskarte mit den Adressen *%QB0* und *%QB1* für die binären Ausgänge der Stellsignale von `EU1` (%QX0.0), `UV1` (%QX0.1), `UV2` (%QX0.2) und `UV3` (%QX0.3),
- analoge Eingangskarte mit den Adressen *%IW1*, *%IW2* und *%IW3* für die analogen Eingänge von `QIC1` (%IW1), `LIS1` (%IW2) und `TIC1` (%IW3) und
- analoge Ausgangskarte mit der Adresse *%QW1* für den analogen Ausgang von `QIC1` (%QW1).

In diesem Beispiel sollen die Programme nur von einer Task aufgerufen werden, und zwar vom Programm `PLC_PRG`, das unter CoDeSys die Programmaufrufe in AWL folgendermaßen realisiert:

```
PROGRAM PRG_PLC
    CAL    LIS1   (* Messung des Füllstands im Reaktor *)
    CAL    LS2    (* Überwachung des Niveauschalters in Behälter B *)
    CAL    LS3    (* Überwachung des Niveauschalters in Behälter A *)
    CAL    QIC1   (* pH-Wertregelung im Reaktor *)
    CAL    TIC1   (* Temperaturregelung in Behälter B *)
    CAL    EU1    (* Rührer im Reaktor *)
    CAL    UV1    (* Zulaufventil in Reaktor *)
    CAL    UV2    (* Ablaufventil des Reaktors *)
    CAL    UV3    (* Heißdampfventil *)
```

Stufe 5: Test und Inbetriebnahme der Software

Der Softwaretest kann zunächst im *Simulationsmodus* der CPU des PC’s ablaufen. Dies ist deshalb von Vorteil, weil einzelne Module direkt nach ihrer Programmierung getestet werden können, ohne dass sie schon auf die Aktoren und Sensoren zugreifen und eventuell Schäden verursachen könnten. Außerdem ist es unbedingt erforderlich, die programmierten Funktionsbausteine zunächst nur in einer Instanz zu testen, bis sie tatsächlich frei von Fehlern sind.

Der Simulationstest umfasst:

- den *Modultest* der Funktionsbausteine und
- den *Integrationstest* der Zusatzlogik in den Programmen.

Ein Testplan kann dabei in Tabellenform, wie in Tabelle 04.10 gezeigt, erstellt werden. Dabei handelt es sich jedoch nicht um eine statische Wahrheitstabelle, sondern eher um eine Schaltfolgetabelle, weil das Speicherverhalten der z.B. RS-Flip-Flops berücksichtigt werden muss.

Die linke Spalte beschreibt dabei den *Testvorgang* und die mittlere Spalte das *Akzeptanzkriterium*. Um den Testplan später auch gleich als Testprotokoll verwenden zu können, wird die rechte Spalte für das *Testergebnis* reserviert.

Tabelle 04.10 zeigt exemplarisch den Plan bzw. das Protokoll für den Test des Bausteins `TYP_IDF1` im Programm `EU1`. Wenn der Test erfolgreich verlief, kann man auf die Tests der Instanzen in den Programmen `UV1` und `UV2` verzichten, denn es handelt sich ja um den gleichen Baustein.

Test-Nr.	Testvorgang Aktivierung der Steuersignale			Akzeptanzkriterium Überprüfung der Statussignale	Test-ergebnis
Schritt	ON	OFF	LOCK	OUT	OUT
1	0	0	0	0	OK
2	1	0	0	1	OK
3	0	0	0	1	OK
4	0	0	1	0	OK
5	1	0	1	0	OK
6	1	0	0	1	OK
7	0	1	0	0	OK
8	1	1	0	0	OK

Tabelle 04.10 Testprotokoll für den *Modultest* des Funktionsbausteins `TYP_IDF1` nach Abb. 04.19.

Zu testen bleiben allerdings noch die *Zusatzlogik* und die *Programmverknüpfungen*. Gerade diese sind besonders sicherheitsrelevant, weil sie die Aktoren verriegeln, etwa um den Überlauf eines Behälters zu verhindern. Auch bei diesen Tests sind, wie Tabelle 04.11 zeigt, *Testvorgang*, *Akzeptanzkriterium* und *Testergebnis* in Tabellenform aufzuführen.

Test-Nr.	Testvorgang Aktivierung der Steuersignale	Akzeptanzkriterium Überprüfung der Statussignale	Test-ergebnis
1	Aktivierung von U1.AUF bis L2=1	UV1 wird verriegelt (STELL=0, LOCK=1)	OK
2	Neues Schmutzwasser fließt zu (L2=0)	Verriegelung wird aufgehoben (STELL=0, LOCK=0)	OK
3	Aktivierung von U1.AUF bis L1.SMAX=1	UV1 wird verriegelt (STELL=0, LOCK=1)	OK
4	Aktivierung von U2.AUF bis L1.SMAX=0	Verriegelung wird aufgehoben (STELL=0, LOCK=0)	OK

Am Ende erfolgt der Test an der realen Anlage, also im Zusammenspiel mit den realen Sensoren und Aktoren.

4.4.3 Umsetzung des Leitfadens auf speziellen Steuerungen

Das vorgestellte Konzept der strukturierten Programmierung lässt sich in den meisten Industriesteuerungen umsetzen, weil die Programmstrukturen nach IEC 61131 genormt sind.

Beim Programmiersystems STEP7 von Siemens müssen die hier gezeigten Verfahren sinngemäß angewandt werden, da sich STEP7 nicht an die IEC 61131 hält, sondern sich nur an diese anlehnt.