Programmierung einer SPS

auf der Basis einer SIEMENS S7-314c

Fachhochschule für Technik und Wirtschaft Berlin,
Fachbereich II
Labor für Mechatronik / Masterstudiengang

Vorwort

Der Versuch „Programmierung einer SPS“ (XSPS) basiert im Wesentlichen auf praxisorientierten Lerneinheiten. Darüber hinaus erhalten die Teilnehmer Informationen über den Aufbau und die Funktion einer SPS. Sie lernen die Programmierung anhand von kleinen Übungsaufgaben, die mit dem Programmiergerät (PC - Step 7) erstellt und mit der SPS getestet werden.

Die theoretischen Kenntnisse zum Aufbau, zur Programmierung und zur Handhabung der eingesetzten Programmentwicklungsumgebung sind aus den Handbüchern und der angegebenen Literatur zu erarbeiten.

Mit diesem Skript wird den Teilnehmern nun auch die Möglichkeit gegeben, sich auf die relativ kurze Laboreinheit vorzubereiten. Darüber hinaus dient es hoffentlich auch als Nachschlagewerk (Quick Reference) für spätere SPS-Projekte.

September 2005

Inhaltsverzeichnis

Lesen Sie Bitte auch die

Laboranleitung „Programmierung einer SPS“

1. Grundlagen der SPS-Technik

1.1 Was ist eine Speicherprogrammierbare Steuerung?

Eine Speicherprogrammierbare Steuerung, kurz SPS genannt, ist ein Computer, der speziell für Steuerungsaufgaben entwickelt wurde. Erst durch das Programm und die Beschaltung mit Ein- und Ausgabegeräten wird die SPS zu einer ganz spezifischen Steuerung.

Als Eingabegeräte dienen passive, aber immer häufiger auch aktive Signalgeber. Passive Signalgeber sind z.B. Schalter oder Taster. Aktive Signalgeber liefern die Informationen über z.B. Temperatur, Druck oder Geschwindigkeit in Form von analogen Spannungen, Strömen oder rein Digital.
Ventile, Schütze und Lampen sind klassische Ausgabegeräte. Aber auch hier verwendet man immer häufiger intelligente Geräte (Aktoren).

Intelligente Sensoren/Aktoren enthalten meistens signalverarbeitende Komponenten (Sicherungsmechanismen), die die SPS entlasten. Darüber hinaus sind Bus-Interfaces vorhanden, die eine rein digitale Anschaltung ermöglichen (Feldbus).

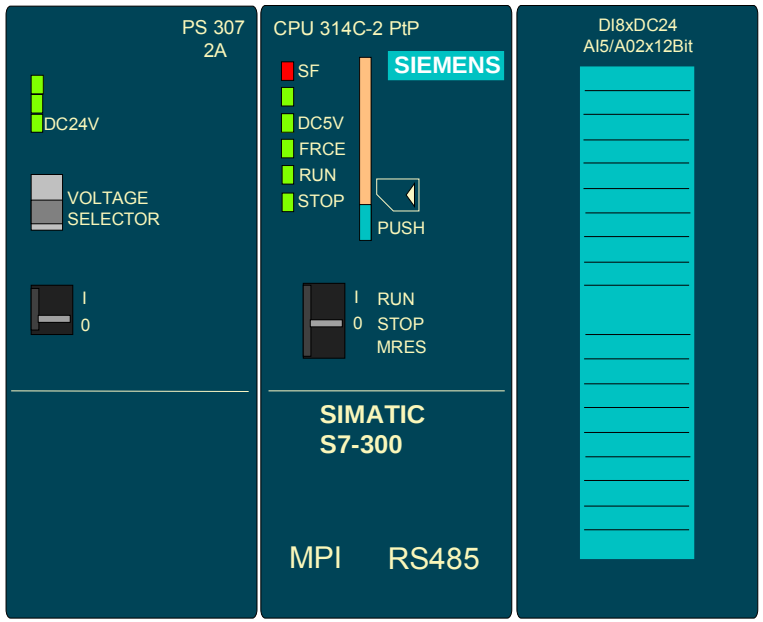
SPS-Steuerungen sind in folgenden Bereichen denkbar:

- Automatisierung von Wohnhäusern
  - Garagentürsteuerung
  - Beleuchtungssteuerung
  - Alarmanlage
  - Fenster-Rolladen-Steuerung
  - Fahrstuhlsteuerung

- Industrieanwendungen
  - Getränkeindustrie
  - Automobilindustrie
  - Pharmaindustrie
  - Förderanlagen
  - Werkzeugmaschine
  - Roboter
  - Nahrungsmittelherstellung
  - Galvanisierungsanlagen
  - Stahlindustrie

1.2 Grundsätzlicher Aufbau einer SPS

Dieser Abschnitt beschreibt den grundsätzlichen Aufbau einer SIEMENS SPS S7-314. Diese Angaben gelten in der Regel auch für Anlagen anderer Hersteller.



SIMATIC S7-314 PtP

- Eine SPS besteht in der Regel aus verschiedenen Einzelkomponenten. Diese Komponenten werden auf ein so genanntes **Busmodul** oder einen Baugruppenträger montiert.
- In jedem Fall ist ein **Netzteil** erforderlich, das die SPS und alle weiteren Bausteine mit 24Volt Systemgleichspannung versorgt.
- Die Zentrale Baugruppe der SPS ist die **CPU-Baugruppe** (Central Processing Unit). Diese Baugruppe enthält das SPS-Programm sowie den zur Abarbeitung erforderlichen Rechner.
- Der CPU-Typ bestimmt die Leistungsfähigkeit der gesamten SPS,

  - Umfang des Befehlsvorrats
  - Programmgröße (Speicherausbau)
  - Befehlsausführungszeiten (Geschwindigkeit)

- Eine SPS benötigt **Eingangsbaugruppen**, mit denen Prozesssignale erfasst werden können. Baugruppen für digitale und analoge Signale werden eingesetzt.
- **Ausgangsbaugruppen** sind erforderlich um Geräte schalten bzw. steuern zu können. Auch hier gibt es konfektionierte digitale und analoge Baugruppen.
- Darüber hinaus stehen noch eine Vielzahl von Sonderbaugruppen für ganz spezielle Anwendungsfälle zur Verfügung. Dies sind z.B.

    - Zählerbaugruppen
    - Positionierbaugruppen
    - Reglerbaugruppen
    - Kommunikationsbaugruppen

1.3 Wie wird eine SPS programmiert?

Damit eine SPS einen Prozess steuern kann, sind folgende Schritte notwendig:

1. Aufbau und Einstellung (Hardwarekonfiguration) der einzelnen Baugruppen
2. Anschluss des Netzteils
3. Verbinden der Ein- und Ausgänge mit dem Prozess
4. Erstellung des SPS-Programms
5. Übertragung des SPS-Programms ins Automatisierungsgerät (AG)

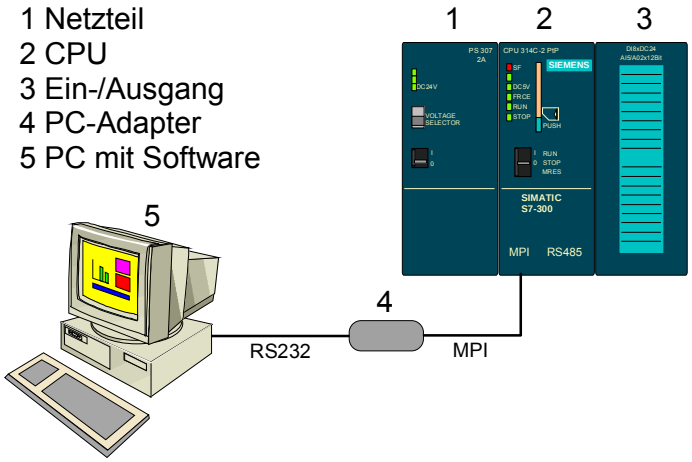
Die Punkte 1 bis 3 sind je nach SPS-Typ unterschiedlich und müssen im jeweiligen Gerätehandbuch nachgelesen werden. Die Punkte 4 und 5 hingegen sind weitestgehend gleich.

Um ein Programm zu erstellen bzw. in eine SPS zu übertragen benötigt man ein Programmiergerät. Dies kann ein speziell dafür vorgesehenes Programmiergerät (PG) oder aber ein handelsüblicher PC sein.
Ein vom SPS-Hersteller angebotenes PG enthält bereits alle erforderlichen Komponenten zur:

- Erstellung des Programms (Editor, Compiler,…)
- MPI-Interface zur Übertragung des Programms auf die CPU
- Steuerung der SPS (z.B. Start/Stop-Funktion)
- Ausdruck des Programms
- Fehlerdiagnose
- Inbetriebnahme

Steht kein PG zur Verfügung, dann muss auf einem PC die jeweilige Programmiersoftware installiert werden. Die Übertragung des Programmcodes erfolgt dann in der Regel über die Serielle Schnittstelle. Bei der SIMATIC S7 wird

die Verbindung zwischen PC und AG mit einem MPI-Adapter hergestellt. Dieser Adapter besitzt einen RS232 und einen MPI Anschluss. Für diese Busankopplung sind auch PC-Karten erhältlich.



Minimalkonfiguration einer SPS mit PC-Programmierung

1.4 Beispiel einer Industrieanlage mit SPS-Steuerung

Eine Anlage in einem Hochregalsystem sortiert Raaco- Kästen anhand ihrer Größe. Dieses einfache Beispiel zeigt noch einmal alle erforderlichen Komponenten und Arbeitsgänge bis hin zur funktionstüchtigen Steuerung.

Prozessbeschreibung:
In der Wahrenannahme eines Betriebes werden elektronische Kleinteile in Raaco-Kästen unterschiedlicher Größe einsortiert und diese auf ein Förderband gestellt. Eine SPS-Steuerung selektiert die Behälter nach ihrer Größe, bevor sie ein Roboter im Hochregal abstellt.

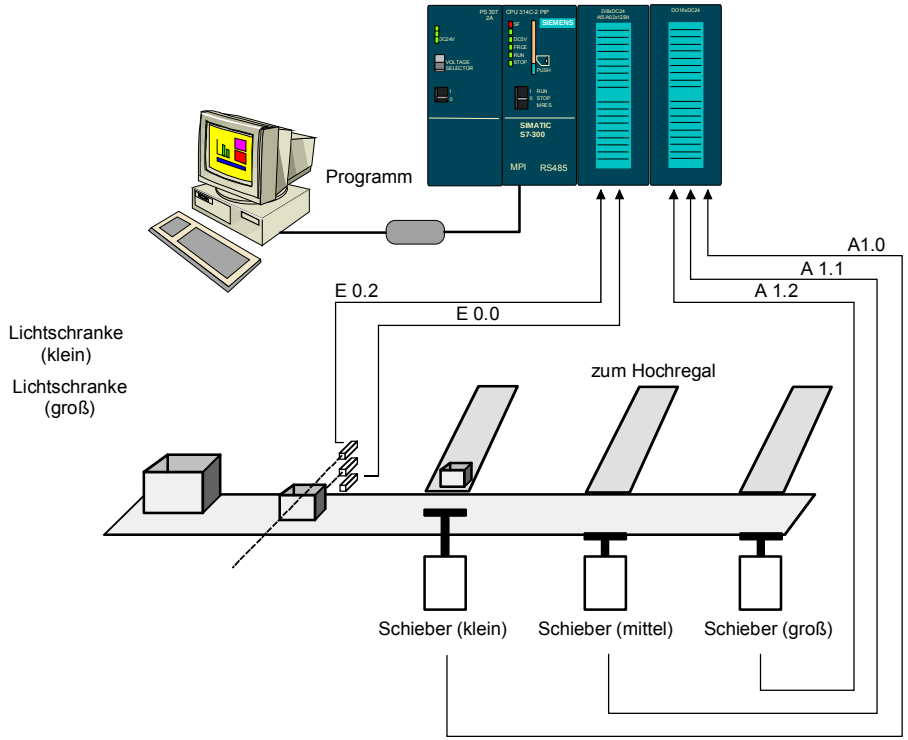
Zur Lösung der Aufgabe wird eine SPS mit mehreren Ein- und Ausgänge benötigt. Zunächst werden alle erforderlichen Ein- und Ausgänge definiert und in einer Zuordnungstabelle (siehe Tab. 1) erfasst.

Die erste Spalte der Tabelle beschreibt die Aktoren und Sensoren, die mit der SPS verbunden sind.
In der zweiten Spalte wird festgehalten, ob es sich um einen Ein- oder Ausgang handelt.

In der dritten Spalte wird die SPS-Belegung (Adresse) aufgeführt. Ein „E“ repräsentiert den Eingang, ein „A“ entsprechend den Ausgang der SPS. Soll ein einzelnes Bit einer Adresse angesprochen werden, so muss dies durch die Angabe der Byte-Adresse sowie des zugehörigen Bits erfolgen. Die Angabe „E 0.1“ kennzeichnet somit das Bit 1 des Eingangsbytes 0.

Tabelle 1:

Aktor / Sensor	Eingang / Ausgang	SPS-Belegung
Schieber (klein	Ausgang	A 1.0
Schieber (mittel)	Ausgang	A 1.1
Schieber (groß)	Ausgang	A 1.2
Lichtschranke (klein)	Eingang	E 0.0
Lichtschranke (mittel)	Eingang	E 0.1
Lichtschranke (groß)	Eingang	E 0.2



SPS-Konfiguration für einen einfachen Industrieprozess

Damit die SPS richtig auf die Signale der Sensoren reagieren kann, muss man wissen, wann die Sensoren welche Signale liefern.
Unsere Lichtschranke liefert z.B. einen Low-Pegel (0Volt) bei Unterbrechung des Lichtstrahls. Anderenfalls beträgt das Signal 24Volt.
Die Schieber werden dagegen betätigt (ausgefahren), wenn der SPS-Ausgang einen High-Pegel (24Volt) liefert.

Da die verwendeten Aktoren und Sensoren nur zwei Schaltzustände kennen, werden auch nur digitale Ausgabe- bzw. Eingabebaugruppen benötigt.

Das SPS-Programm fragt nun kontinuierlich die Signale der drei Lichtschranken ab und entscheidet daraus, welcher Schieber wann betätigt wird. Die folgende Tabelle zeigt die möglichen Zustände.

Lichtschranke	Zustand	Schieber	Zustand
L (klein) L (mittel) L (groß)	1 0 0	S (klein)	1 nach ca. 3s
L (klein) L (mittel) L (groß)	1 1 0	S (mittel)	1 nach ca. 5s
L (klein) L (mittel) L (groß)	1 1 1	S (groß)	1 nach ca. 7s

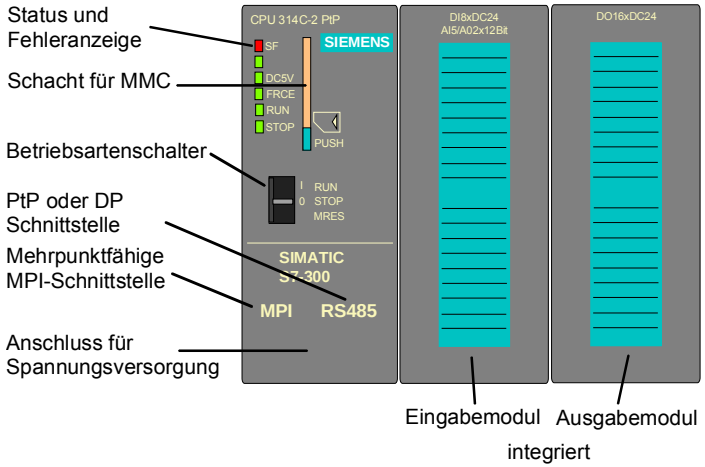
Da bei einem großen Behälter wegen der Anordnung alle drei Lichtschranken unterbrochen sind, muss das Programm stets alle drei Signale auswerten.
Die Schieber sind in definiertem Abstand zum Sensor angebracht und müssen daher mit einer, von der Transportgeschwindigkeit abhängigen, Verzögerung eingeschaltet werden.

2. Der Labor- Arbeitsplatz (Hardware)

2.1 Aufbau der SIMATIC S7-314c PtP bzw. DP

Jeder Arbeitsplatz in unserem Labor ist mit einer SIEMENS SPS des Typs S7-314c ausgerüstet. Diese SPS ist eine Kompakt-SPS. Sie unterscheidet sich von den üblichen Steuerungen durch bereits implementierte digitale und analoge Baugruppen. Leistungsmäßig ist diese SPS eher im unteren bis mittleren Marktsegment einzuordnen. Es besteht jedoch die Möglichkeit die Anzahl der Ein- und Ausgänge durch Anschaltung weiterer Baugruppen zu erhöhen.

Aufbau der S7-314



CPU S7-314C-2 PtP

Zur Aufnahme und Speicherung des Programms besitzt die obige CPU eine so genannte **Micro-Memory-Card**. Diese Karte muss im Betrieb der CPU gesteckt sein.

Alle CPUs besitzen eine MPI-Schnittstelle (MPI := Multi Point Interface) zum Anschluss an ein Programmiergerät oder zur Vernetzung mit einer zweiten SPS. Das Kürzel PtP (Point to Point) weist auf eine weitere Schnittstelle (RS485) zum Anschluss von Ausgabegeräten (Plotter). Beim Kürzel DP ist eine Profibus-Schnittstelle vorhanden.

Neben der CPU befinden sich hinter je einer Klappe die Anschlussklemmen der Ein- und Ausgänge. Zur Verbindung mit den Sensoren bzw. Aktoren ist je ein 40 pol. Adapter mit Schraubklemmen erforderlich.

Einige Daten der S7-314c PtP auf einen Blick:

Integrierter Arbeitsspeicher	48 Kbyte
Ladespeicher	steckbar (MMC) > 64 Kbyte
Bearbeitungszeiten	Bit-/Wortoperation 0.1 / 0.2 µs. Festpunkt-/Gleitpunktarithmetik 2 / 20 µs
Anzahl S7-Zähler S7-Zeiten Merker	256, Bereich 0..999 256, 10ms..9990s 256 Byte
Datenbausteine Funktionsbausteine Funktionen	max. 127, Größe max. 16 KByte max. 128, Größe max. 16 KByte max. 128, Größe max. 16 KByte
Adressbereich gesamt Digitale Kanäle Analoge Kanäle	max. 1024 Byte (frei adressierbar) max. 1016 max. 253
Baugruppenträger	max. 4
Digitale Eingänge	24
Digitale Ausgänge	16
Analoge Eingänge	4 + 1 (12Bit)
Analoge Ausgänge	2 (12Bit)
Technologiefunktionen	4 Zähler 1 Positionieren

Die digitalen Eingänge der Schulungsgeräte sind zu Testzwecken mit Schaltern/Tastern verbunden. Die Ausgänge sind auf Steckbuchsen herausgeführt. Das folgende Bild zeigt ein AG (Automatisierungsgerät) des Laboraufbaus.

2.2 Baugruppen der S7-300

Das vorgestellte Schulungsgerät ist, wie schon erwähnt, die Minimalkonfiguration einer SPS. In großen Industrieanlagen werden nicht selten mehrere 100 Sensoren und Aktoren benötigt, die alle von einer oder gar mehreren Steuerungen bedient werden müssen. Darüber hinaus soll die SPS heutzutage nicht nur Steuerungsaufgaben übernehmen, sondern z.B. auch Überwachungseinrichtungen mit aktuellen Prozessdaten versorgen (Monitoring) oder mit anderen Steuerungen kommunizieren können.

Zur Erweiterung einer vorhandenen SPS bietet SIEMENS, wie wahrscheinlich auch andere Hersteller (Bosch, Klöckner-Möller, usw.) Zusatzbaugruppen an, um die Funktionalität der Anlage zu erweitern.

Die folgenden Baugruppen sind für den Ausbau der S7-3xx erhältlich.

- **Stromversorgungsbaugruppen (PS 307, Typ)**

  Der Typ kennzeichnet den Ausgangsstrom (2, 5 oder 10A)

- **Digital-Eingabebaugruppen (SM 321, Typ)**

- **Digital-Ausgabebaugruppen (SM 322, Typ)**

- **Analog-Eingabebaugruppen (SM 331, Typ)**

- **Analog-Ausgabebaugruppen (SM 332, Typ)**

  Der Typ für die Signalbaugruppen kennzeichnet u.a.

  - Die Anzahl der Ein-/Ausgänge
  - Die Höhe der Ein-/Ausgangsspannung (24V-DC, 120V-AC)

- **POS-Eingabebaugruppen (SM 338)**

  Die POS-Eingabebaugruppe ist eine Schnittstelle zur Anschaltung von SSI-Weggebern)

- **Funktionsbaugruppen (FM 3xx)**

  Diese Baugruppen sind für zeitkritische und speicherintensive Prozesssignalverarbeitung, zum Beispiel für

  - CNC-Steuerungen
  - Nockensteuerungen
  - Positionieraufgaben
  - Regelungen

- **Kommunikationsprozessor-Baugruppe (CP 34x)**

  Diese Baugruppen entlasten die CPU von Kommunikationsaufgaben und sind mit unterschiedlichen Schnittstellen ausgestattet:

  - Profibus DP
  - RS232/485 Point to Point
  - Ethernet

- **Anschaltungsbaugruppen (IM 36x)**

  Diese Baugruppen dienen zur Verbindung mehrerer Baugruppenträger und liefern die Versorgungsspannung für weitere Signalbaugruppen.

- **RS-485 Repeater**

  Ein RS 485-Repeater verstärkt Datensignale auf Busleitungen und koppelt Bussegmente.

2.3 Vernetzung aller Geräte

Wie bereits angesprochen, enthalten unsere AGs Schnittstellen zur Vernetzung. Vier der Geräte besitzen neben einer MPI-Schnittstelle auch ein Profibus-Interface. Da die Steckverbindungen bzw. Übertragungsleitungen für beide Interfaces die gleichen sind, läst sich die Hardware unterschiedlich konfigurieren.

1. Für die Kommunikation via MPI-Bus können alle 8 AGs zu einem Subnetz bzw. Segment verbunden werden.

2. Für die Kommunikation via Profibus DP können 4 Geräte ein MPI-Subnetz und 4 weitere ein Profibus-Subnetz bilden. Eine DP-Station übernimmt dabei die Routing- Funktion.

Darüber hinaus verfügen wir im Labor über einen SIMATIC NET-Kommunikationsprozessor (S7-343-1 IT). Dieses Gerät besitzt ein LAN-Interface (RJ-45) und ermöglicht den Zugang zum Internet. Der Kommunikationsprozessor verfügt über einen WEB-Server und unterstützt diverse Mail-Funktionen. Er wird über den S7-Rückwandbus mit einer unserer Stationen verbunden und ist somit auch Teilnehmer im MPI-Subnetz.

Ein weiterer Kommunikationsprozessor (CP343-2) ermöglicht den Netzübergang zum ASI-Bus.

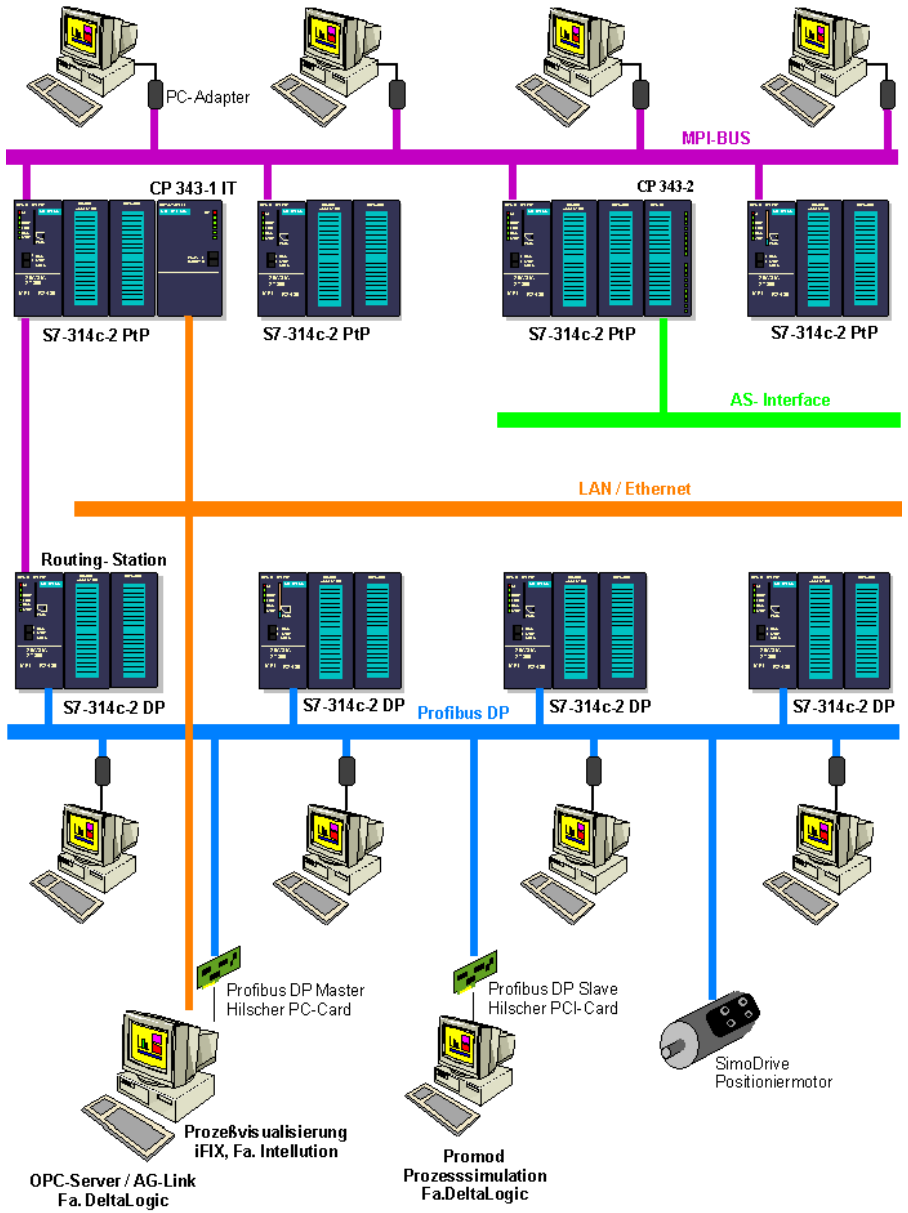
Zur Prozessvisualisierung bzw. Steuerung mittels eines PCs verwenden wir in unserem Labor zwei Profibus-Interfaces in Form einer PCI-Steckkarte der Firma Hilscher.
Eine der beiden Karten läst sich als Master/Slave, die andere als Slave konfigurieren.

Ein PC in unserem Netz verfügt über einen OPC-Server der Fa. Deltalogic. Dieser kann über den PC-Adapter, die Hilscher Profibus DP- Karte und über den PC-LAN-Anschluss auf unsere Automatisierungsgeräte zugreifen und deren Daten weiteren Anwendungen bereitstellen.

Zur Prozessvisualisierung verwenden wir die Software iFIX von Intellusion. Ihr Client übernimmt die Prozessdaten vom OPC-Server.

Darüber hinaus können wir auch aus eigenen Anwendungen mit Hilfe den AG-LINK-Funktionen auf Prozessdaten zugreifen und diese verändern. Wir üben diese Möglichkeit im Programmierkurs (Graphische Programmierung mit LabVIEW).

Schließlich steuern wir in unserem Automatisierungsnetz einen SIEMENS Positioniermotor mit Profibusinterface. Das folgende Bild zeigt den gesamten Laboraufbau.



3. Die Projektierung eines S7-300 Aufbaus

3.1 Vorbemerkungen

Der Markt für Automatisierungsgeräte bietet eine große Anzahl von Geräten, mit denen ganz unterschiedliche Steuerungsaufgaben erfüllt werden können. Nicht nur der Preis, sondern auch die Ausstattung der Geräte variiert in hohem Maße. Neben einer Reihe von namhaften Herstellern wie z.B. SIEMENS, MOELLER, KUHNKE, und MITSUBISHI gibt es viele so genannte Low-Cost- Geräte mittlerer und kleinerer Unternehmen. Hat man sich erst einmal für einen Hersteller entschieden, so folgt die Qual der Wahl. Beinahe für jede Steuerungsaufgabe gibt es eine passende SPS. Die Suche nach der preiswerten, akzeptablen Lösung kann hier zu einer langwierigen Recherche führen.

Wir haben uns bei der Anschaffung unserer S7-300-Schulungsgeräte aus folgenden Gründen für den deutschen Hersteller SIEMENS entschieden:

- Vorhandene Erfahrungen auf der Basis von S5-Steuerungen
- Gute Dokumentation
- Schneller Support (Ansprechpartner vor Ort)
- Flexibles System mit großem Erweiterungsangebot
- Hervorragendes Entwicklungssystem
- Gute Integrationseigenschaften (Anbindung anderer Systeme)
- Hoher Marktanteil (Bekanntheitsgrad)

3.2 Anforderungen an die SPS

Zu Beginn der Projektierung einer neuen Anlage ist es in jedem Fall erforderlich die Steuerungsaufgabe so exakt wie möglich zu beschreiben. Hieraus können dann die Anforderungen an die SPS formuliert werden. Die Abarbeitung der folgenden Checkliste erleichtert das Aufstellen eines Anforderungsprofils.

1. Anzahl der digitalen Eingänge?
   - welche Schnittstellen besitzen die Sensoren (U, I, Spannungshöhe)?

2. Anzahl der digitalen Ausgänge?
   - welche Schnittstellen besitzen die Aktoren (U, I, Spannungshöhe)

3. Anzahl der analogen Eingänge (U, I, Auflösung, Abtastrate)?

4. Anzahl der analogen Ausgänge (U, I, Auflösung, Abtastrate)?

5. Werden Laststromversorgungen zum Anschluss der Aktoren benötigt?
   - Ein digitaler Ausgang liefert in der Regel nur 20mA Strom

6. Welche Sonderfunktionen werden benötigt (Regler, Zähler, usw.)?
   - Jede Sonderfunktion entlastet die CPU

7. Sind neben der Steuerung Kommunikationsaufgaben zu erledigen?
   - z.B. Monitoring, Netzwerkbetrieb (Profibus, Ethernet)

8. Welche Stromversorgung ist erforderlich?

9. Soll die SPS in einem Netzwerk betrieben werden?
   - welche Schnittstellen sind erforderlich?
   - gibt es große Leitungslängen zu überbrücken

10. Ist ein mehrzeiliger Aufbau erforderlich?
    - wie viele Anschaltungsbaugruppen werden benötigt?
    - Wird der Einsatz mehrerer CPUs erforderlich?

11. Wie umfangreich ist die Steuerungsaufgabe (Speichergröße)?
    - Der RAM-Speicher einer CPU ist in der Regel nicht erweiterbar

12. Wie Zeitkritisch ist die Steuerung (Zykluszeit)?

13. Welche Umweltbedingungen gelten für den Betrieb der SPS?
    - Einsatz spezieller Outdoor-Baugruppen

3.3 Auswahl der CPU

Die CPU ist das Herzstück einer SPS-Anlage. Sie bestimmt im Wesentlichen die Leistungsfähigkeit bezüglich der

- max. Anzahl der Ein-/Ausgänge
- der Programmgröße
- der Verarbeitungsgeschwindigkeit

Im Kapitel 2.1 wurde bereits ein kleiner Auszug der technischen Daten unserer CPU S7-314 angegeben. Für eine ausführlichere Beschreibung der CPU-Daten verweise ich auf das jeweilige Handbuch in unserem Labor oder online im Internet.

Um dennoch einen Eindruck über die Leistungsfähigkeit unterschiedlicher CPUs zu verschaffen habe ich die wichtigsten Daten von zwei weiteren S7-CPUs in der folgenden Tabelle gegenübergestellt.

Eigenschaften	CPU-312 IFM	CPU-318-2
Integrierter Arbeitsspeicher	6 KByte	256 KByte
Ladespeicher	20 KByte RAM nicht erweiterbar	64 KByte bis 2 MB erweiterbar
Bearbeitungszeiten - Bitoperation - Gleitpunktarithmetik	 0,6 µs 60 µs	 0,1 µs 0,6 µs
S7-Zähler	32	512
S7-Zeiten	64	512
Digitale Kanäle	256+10integriert	65536
Analoge Kanäle	64	4096

Darüber hinaus bietet das Produktspektrum der Firma SIEMENS natürlich weitere CPUs sowohl unterhalb (S7-2xx), als auch oberhalb (S7-4xx) der im Beispiel aufgeführten Geräte.

Die richtige CPU sollte in jedem Fall dem aufgestellten Anforderungsprofil genügen. Weiterhin sollte bei der Projektierung eine Ausbaureserve vorgesehen werden, d.h. der Auslastungsgrad der CPU sollte zwischen 60 und 80% liegen. So schafft man sich einen kleinen Spielraum für spätere Erweiterungen des Systems.

3.4 Regeln für der Aufbau einer Steuerung

Beim Projektieren des mechanischen und elektrischen Aufbaus der Anlage sind je nach Hersteller verschieden Regeln zu beachten. Diese Regeln haben durchaus auch wieder Einfluss auf die Anzahl der zu beschaffenden Komponenten. Daher werden einige dieser Regeln in diesem Kapitel angeführt.

- Die Baugruppen einer S7-300 sind offene Betriebsmittel, d.h. Sie dürfen die Anlage nur in Gehäusen, Schränken oder elektrischen Betriebsräumen aufbauen.

- Die Baugruppenträger der S7-300 können waagerecht oder senkrecht aufgebaut werden. Die zulässige Umgebungstemperatur beträgt:

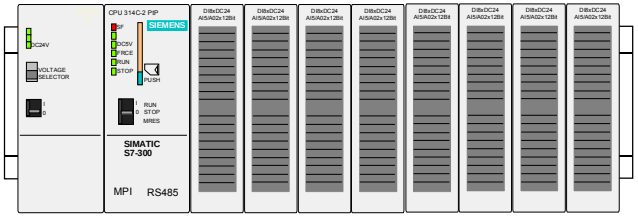
  - waagerechter Einbau 0 bis 60°C
  - senkrechter Einbau 0 bis 40°C

- Die Mindestabstandsmaße für Baugruppenträger müssen den Angaben des Herstellers entsprechen.

  - Zulässige Erwärmung der Baugruppen
  - Platz zum Ein- und Ausbau der Baugruppen
  - Platz zum Verlegen von Leitungen

- Bei der Anordnung der Baugruppen auf einem Baugruppenträger gelten folgende Regeln:

  - Maximal 8 Baugruppen (SM, FM, CP) dürfen rechts neben der CPU stecken.
  - Die Anzahl steckbarer Baugruppen (SM, FM, CP) ist begrenzt durch deren Stromaufnahme aus dem Rückwandbus.

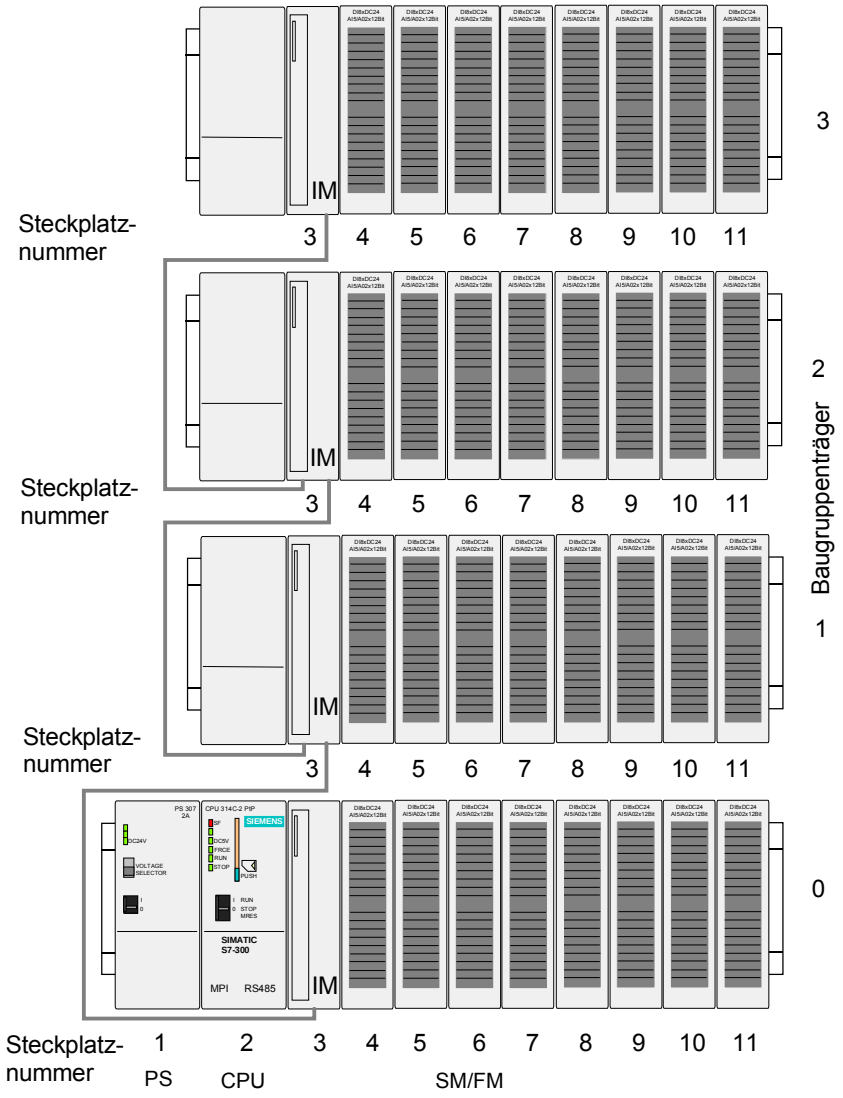


PS CPU SM/FM/CP

- Für die Anordnung der Baugruppen auf mehreren Baugruppenträgern benötigt man so genannte Anschaltungsbaugruppen. Es gelten folgende Regeln:

  - die Anschaltungsbaugruppe belegt immer den Steckplatz 3 und befindet sich immer links neben der ersten Signalbaugruppe.
  - Es dürfen je Baugruppenträger maximal 8 Baugruppen (SM, FM, CP) gesteckt werden.
  - Die Anzahl der gesteckten Baugruppen (SM, FM, CP) ist begrenzt durch die zulässige Stromentnahme.
  - Der Abstand zwischen zwei Anschaltungsbaugruppen darf maximal 10m betragen.
  - Die maximale Anzahl an Baugruppenträgern ist von der verwendeten CPU abhängig.

Das folgende Bild zeigt als Beispiel den Maximalausbau für die im Labor verwendete SPS. Der Baugruppenträger besteht aus einer speziellen Profilschiene, auf die alle S7-300 Komponenten aufgesteckt und arretiert werden. Die Stromversorgung erfolgt für alle Baugruppen rechts der CPU über ein Busmodul an der Rückwand. Bei den Baugruppenträgern 1 bis 3 enthält die Anschaltungsbaugruppe (IM) eine eigene Stromversorgung.



Aufbau einer SPS mit 4 Baugruppenträgern

3.5 Adressierung der einzelnen Baugruppen

Steckplatzorientierte Adressvergabe

Die steckplatzorientierte Adressvergabe entspricht der Defaultadressierung, d.h. STEP7 ordnet jeder Steckplatznummer eine festgelegte Baugruppen-Anfangsadresse zu.
Als Beispiel sei hier der im obigen Bild dargestellte Maximalausbau einer S7-300 angeführt. Der Steckplatz 0 des ersten Baugruppenträgers ist immer für ein Programmiergerät reserviert und taucht daher in der Zeichnung nicht auf. Daneben erhält die Stromversorgung die Nummer 1, die CPU die Nummer 2 und die Anschaltungsbaugruppe immer die Steckplatznummer 3. Die übrigen Baugruppen stecken wie angegeben auf den Plätzen 4 bis 11.

Die den Steckplätzen zugeordneten Anfangsadressen können der folgenden Tabelle entnommen werden.

Träger	Bau-gruppe	Steckplatznummer										
		1	2	3	4	5	6	7	8	9	10	11
0	digital analog	PS	CPU	IM	0 256	4 272	8 288	12 304	16 320	20 336	24 352	28 368
1	digital analog			IM	32 384	36 400	40 416	44 432	48 448	52 464	56 480	60 496
2	digital analog			IM	64 512	68 528	72 544	76 560	80 576	84 592	88 608	92 624
3	digital analog			IM	96 640	100 656	104 672	108 688	112 704	116 720	120 736	124 752

Beispiel zur Adressierung:

Der Steckplatz Nr. 4 des Baugruppenträgers 0 enthält eine digitale Eingabebaugruppe der Länge 2 Byte. Die einzelnen Bits der Baugruppe sind somit durch die Operanden

E 0.0 - E 0.7 oder E 1.0 - E 1.7

adressierbar. Da der nächste Steckplatz mit der Adresse 4 beginnt, sind in diesem Beispiel zwei Byte nicht adressierbar. Dies liegt daran, dass der max. Ausbau pro Steckplatz 4 Byte beträgt. Dieses Problem läst sich jedoch durch die freie Adressvergabe umgehen.

Die analogen Baugruppen belegen pro Steckplatz max. 16 Byte, z.B. 8 Kanäle mit 12 Bit Auflösung (8 x 2Byte)

Freie Adressvergabe

Freie Adressvergabe heißt, Sie können jeder Baugruppe (SM, FM, CP) eine Adresse Ihrer Wahl zuordnen. Diese Zuordnung nehmen Sie in STEP7 vor. Sie legen dabei die Baugruppenanfangsadresse fest, auf der dann alle weiteren Adressen der Baugruppe basieren.
Man kann so die verfügbaren Adressräume optimal nutzen. Außerdem kann man bei der Erstellung von Standardsoftware Adressen angeben, die unabhängig von der jeweiligen Konfiguration der S7-300 sind.

3.6 Vernetzung einer S7-300

Die S7-300 besitzt im Gegensatz zum Vorgängermodell (S5) einen MPI-BUS. Dieses neue Interface (Mehrpunktfähige Schnittstelle) ermöglicht die Verbindung mehrerer S7-CPUs zu einem lokalen Steuerungsnetzwerk (Subnetz).
Zum einen kann man über diesen Bus von jeder Anschlussstelle aus auf alle angeschlossenen CPUs zugreifen, zum anderen können CPUs untereinander kommunizieren.

Der Aufbau eines MPI-Subnetzes gleicht prinzipiell dem eines Profibus-Subnetzes. Für den Aufbau gelten die gleichen Regeln, werden die gleichen Komponenten benötigt.

Voraussetzungen für den Aufbau eines Subnetzes

Damit alle Teilnehmer miteinander kommunizieren können, muss ihnen eine eindeutige Adresse zugewiesen werden.

- im MPI-Subnetz eine „MPI-Adresse“ sowie eine „Höchste MPI-Adresse“.
- im PROFIBUS-Subnetz eine „PROFIBUS-Adresse“ sowie eine „Höchste PROFIBUS-Adresse“.

Die Teilnehmer eines Netzes erhalten die Adressen durch das Programmiergerät. Die folgende Tabelle zeigt zulässige MPI- und PROFIBUS-Adressen.

MPI-Adressen	PROFIBUS-Adressen
0 bis 126	0 bis 125
davon reserviert: 0 für PG (Programmiergerät) 1 für OP (Bediengerät) 2 für CPU	davon reserviert: 0 für PG

Vor der Vergabe von MPI- oder PROFIBUS-Adressen sind folgende Regeln zu beachten:

- Alle MPI-/PROFIBUS-Adressen in einem Subnetz müssen unterschiedlich sein

- Die höchste MPI-/PROFIBUS-Adresse muss ≥ der größten tatsächlichen Adresse sein und bei allen Teilnehmern gleich eingestellt sein.

Regeln für den Aufbau eines Subnetzes

Ein Subnetz kann aus mehreren Einheiten, so genannten Segmenten bestehen. Ein Segment:

- Ist eine Busleitung zwischen zwei Abschlusswiderständen

- Kann bis zu 32 Teilnehmer enthalten (CPU-Typ)

- Wird begrenzt durch die zulässige Leitungslänge in Abhängigkeit von der Baudrate.

Wenn Sie mehr als 32 Teilnehmer in einem Subnetz betreiben, dann müssen Sie die Bussegmente über RS 485-Repeater koppeln.

In einem PROFIBUS-Subnetz müssen alle Bussegmente zusammen **mindesten** einen DP-Master und einen DP-Slave haben.

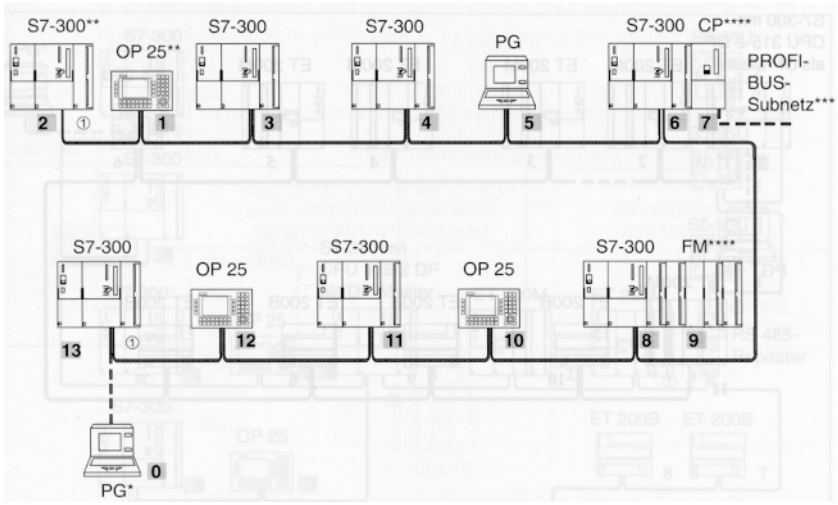
Erdgebundene und Erdfreie Bussegmente müssen über einen RS 485-Repeater gekoppelt werden.

Der erste und letzte Teilnehmer eines Segmentes muss einen Abschlusswiderstand (Leitungs-Wellenwiderstand) besitzen. Dieser befindet sich im Bus-Anschlussstecker.

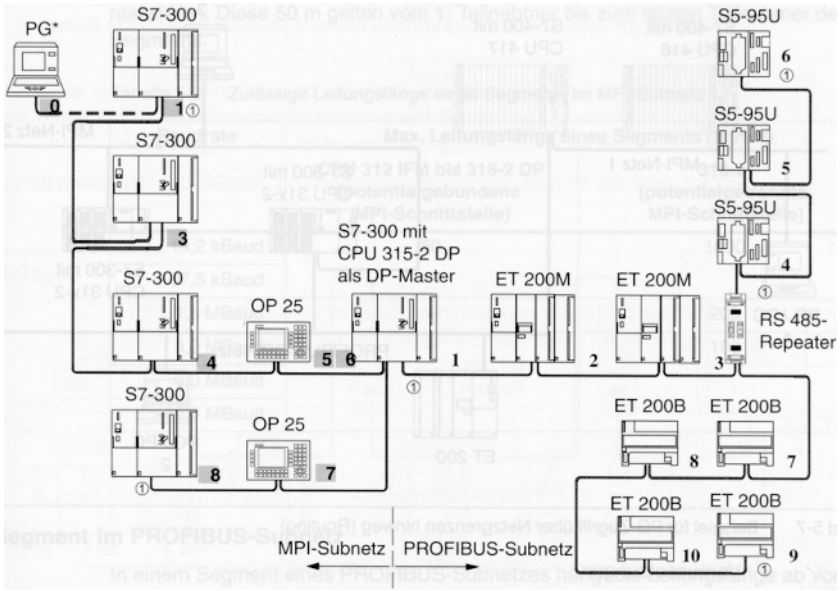
Zur Verbindung der Teilnehmer wird ein Bus-Anschlussstecker sowie ein Profibuskabel benötigt.

Die folgenden Bilder zeigen den Aufbau eines MPI-Subnetzes und die Verbindung eines MPI- und PROFIBUS-Subnetzes.

Die mit einer 1 versehenen Kreissymbole kennzeichnen die Aktivierung der Abschlusswiderstände.



Beispiel für ein MPI-Subnetz



Beispiel für den Übergang zwischen MPI- und PROFIBUS-Subnetz

4. Der Labor- Arbeitsplatz (Software)

4.1 SIEMENS Programmentwicklungsumgebung STEP 7

Für die Programmierung der SPS verwenden wir in unserem Labor das STEP 7 prof. Trainer Package. Diese Software vereint alle erforderlichen Tools unter dem Dach eines Projektmanagers. Ein eingebauter Simulator ermöglicht das sofortige Testen des STEP7 Programms. Die Bedienung der Software bedarf keiner großen Einarbeitungszeit und ist daher für Schulungszwecke ideal.

Hinweis:
Das Trainer Package enthält zeitlich limitierte Studentenversionen, die wir zu Beginn eines Semesters an unsere Studenten ausgeben. So kann bei Bedarf auch zu Hause programmiert und experimentiert werden.

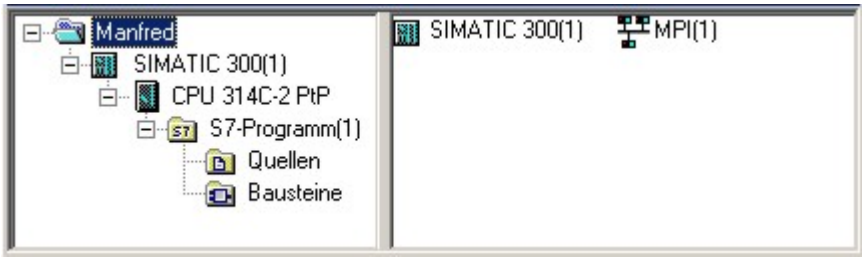
Im Folgenden werden die Schritte der Programmerstellung kurz erläutert. Wir werden uns dabei allerdings nur auf das Wesentliche konzentrieren. Eine detaillierte Beschreibung der Tools liefert in hervorragender Weise die Hilfe des Programms.

4.1.1 Der Projektmanager

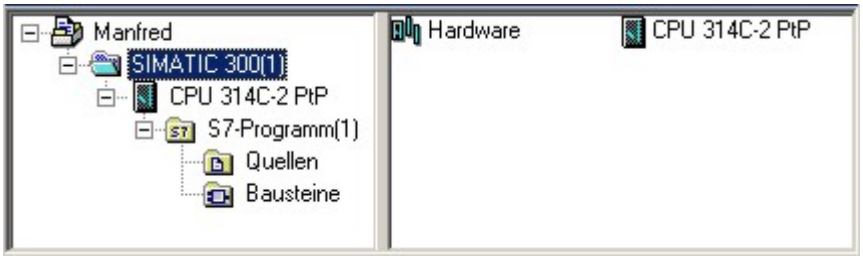
Bevor Sie mit dem Editieren des Steuerungsprogramms beginnen können, muss ein Projekt definiert werden. Dies erfolgt entweder mit dem sehr hilfreichen Assistenten oder manuell. Nach dem Aufruf des „**SIMATIC Manager**“ sollten Sie das Angebot des Assistenten nicht ausschlagen und ein Projekt definieren. Hierzu sind die folgenden Angaben nötig:

- CPU-Typ (Liste)
- MPI-Adresse (2)
- Einsprung-OB (OB1)
- Darstellung (AWL,FUP,KOP)
- Projektnamen.

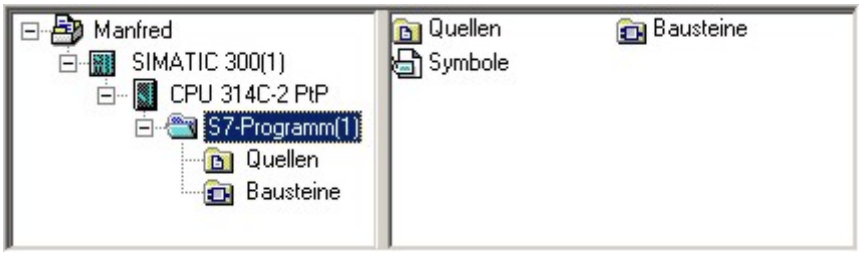
Ist das Projekt angelegt, so erscheint im Projektmanager das folgende Fenster.



Unter dem Projektnamen „**Manfred**“ wurde vom Assistenten ein Ordner „SIMATIC-Station 300(1)“ eingerichtet. Hinter dem Eintrag „MPI(1)“ verbirgt sich des Tool „**NetPro**“. Dieses Programm ermöglicht die Konfiguration eines MPI-Subnetzes.



Der Ordner „**SIMATIC 300(1)**“ enthält einen weiteren Ordner mit dem Namen der gewählten CPU. In diesem Fall handelt es sich um unsere Labor-CPU. Der Eintrag „**Hardware**“ weist auf ein weiteres Tool „**HW Konfig**“. Dies ist ein sehr hilfreiches und notwendiges Konfigurationstool um die Hardware (Baugruppen der SPS) und deren Adressierung festzulegen. Eine im Programm verwendete Eingabeadresse wird vom Compiler erst dann akzeptiert, wenn die zugehörige Baugruppe im Hardwarekonfigurator eingebunden wurde.



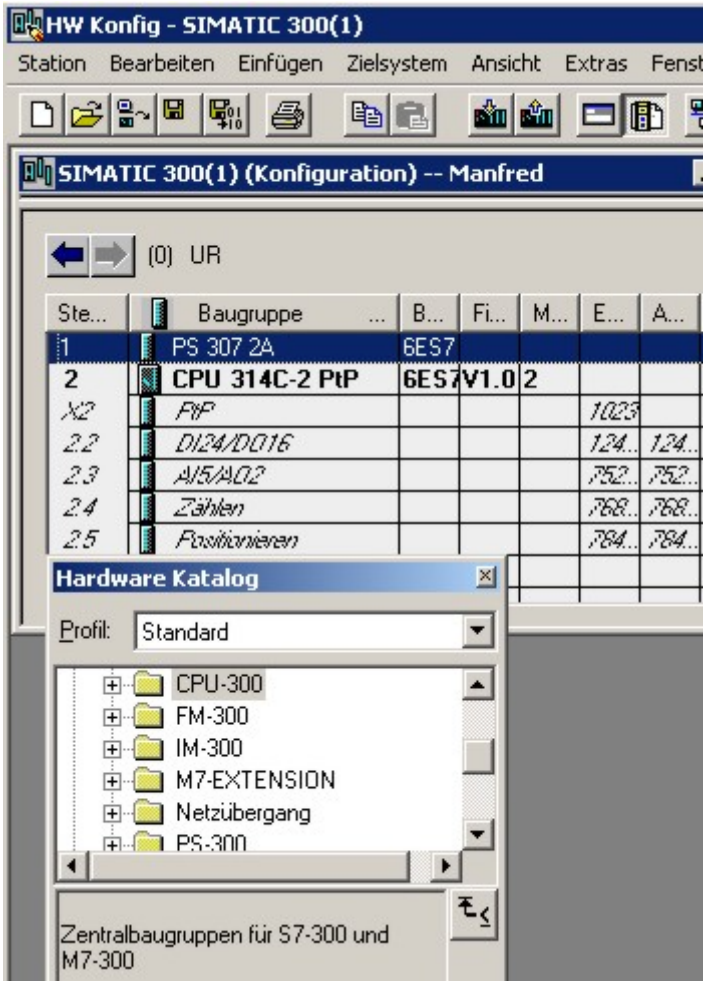
Der Ordner „**CPU 314-2 PtP**“ enthält mindestens einen Ordner für die eigentlichen Programmdaten „**S7-Programm(1)**“.
Der Ordner „**Bausteine**“ enthält alle programmierten S7-Bausteine (OBs, Fcs, FBs, DBs). Die Symboldatei enthält Daten zur Beschreibung der verwendeten Operanden.

Aus den obigen Fenstern heraus kann jede, zur Bearbeitung des Projektes erforderliche Funktion aufgerufen werden. Zum Beispiel

- das Einfügen neuer Bausteine
- das Anlegen einer Symboldatei
- der Aufruf des STEP 7-Editors
- das Übertragen der Bausteine ins AG
- der Aufruf des SPS-Simulators

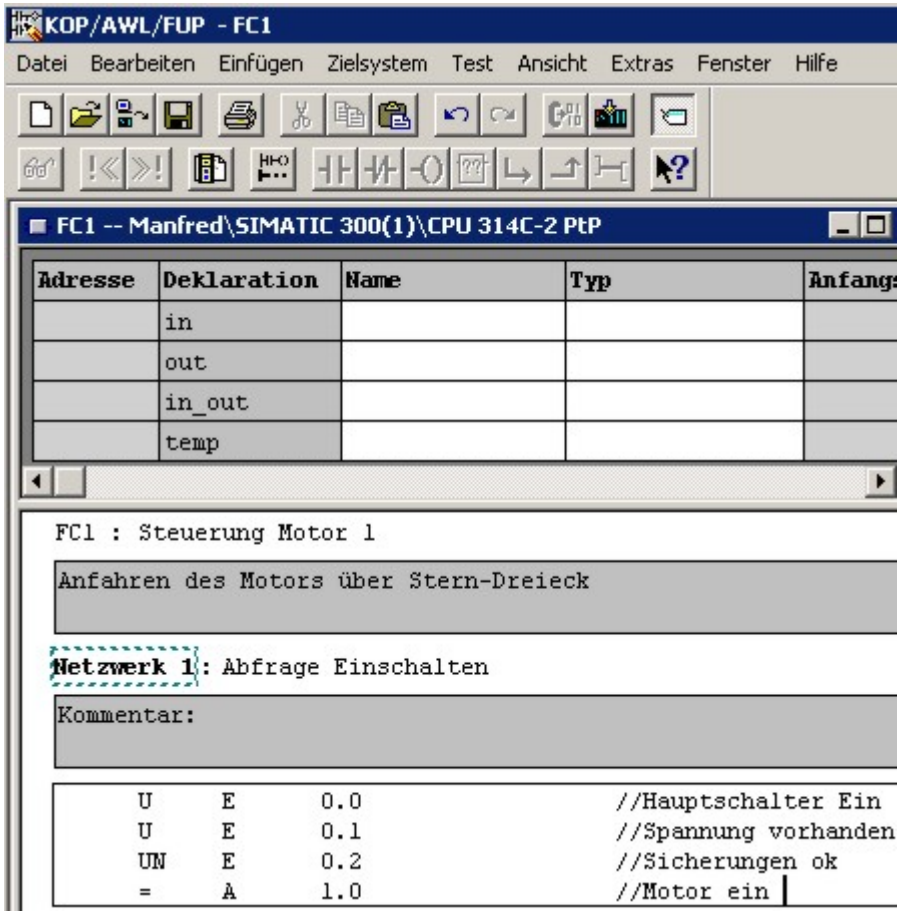
4.1.2 Der Hardwarekonfigurator

Das folgende Bild zeigt einen Ausschnitt des Hardware-Konfigurators. Dieser erstellt ein Abbild der vorhandene SPS. Die benötigten Hardwarekomponenten werden einem Katalog entnommen und auf einen Baugruppenträger platziert. Dabei wird die Adressvergabe zunächst automatisch (steckplatzorientiert) vorgenommen. Möchte man die Adressen frei vergeben, dann kann das in diesem Modul geschehen.



4.1.3 Der STEP 7 Programmeditor

Der Programmeditor ist das am häufigsten verwendete Tool der Software. Durch Doppelklicken auf einen Baustein (OB, FC, FB, DB) im Bausteinverzeichnis des Projektmanagers wird der Editor gestartet und erscheint wie im folgenden Bild dargestellt.



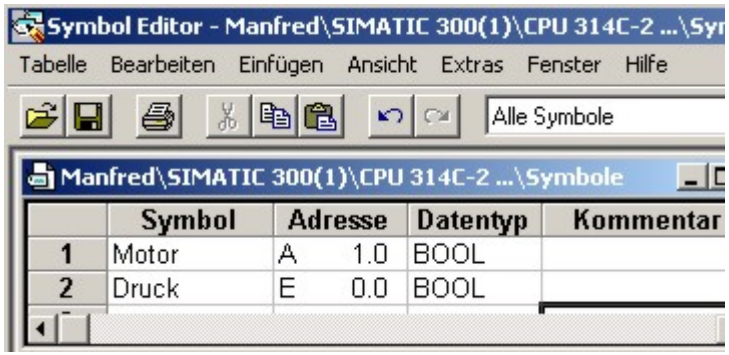
Im oberen Teil des Fensters werden die Variablen des Bausteins definiert. Darunter beginnt der Eingabebereich für den Programmcode. Neben den Dokumentationsmöglichkeiten wie Titel und Kommentare folgt für jedes definierte Netzwerk (kleinster Programmteil eines Bausteins) der Programmcode (AWL, FUP, KOP). Fehler bei der Programmerstellung werden unmittelbar nach der Eingabe

angezeigt. Ist das Programm fertig, so kann man es direkt ins Zielsystem übertragen.

4.1.4 Der S7 Symboleditor

Der Symboleditor dient zum Erstellen einer Symboldatei (Kap.9). Er wird durch Doppelklicken auf das File „Symbole“ im SIMATIC-Manager aufgerufen.

Nach dem Aufruf erscheint der Editor wie folgt in einem eigenen Fenster.



In dem obigen Beispiel wurde dem Ausgangsbit 0 der Byteadresse 1 das Symbol „Motor“ zugeordnet. Darüber hinaus wird der Datentyp und optional ein Kommentar angegeben.

Die Symboldatei wird in der Regel zu Beginn des Projektes aufgestellt, kann aber jederzeit geändert oder erweitert werden.

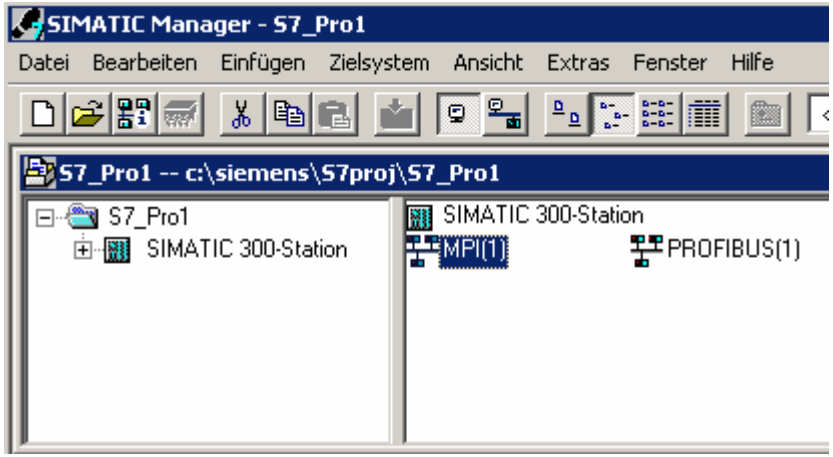
Während des Programmierens lässt sich die Symbolliste durch Anklicken der rechten Maustaste aufrufen und das richtige Symbol kann eingefügt werden. Ein äußerst hilfreiches, übersichtliches Instrument.

4.1.5 Der Netzwerkkonfigurator „NetPro“

Werden mehrere AGs vernetzt, so ist darauf zu achten, dass alle Geräte unterschiedliche Adressen erhalten. Es gibt grundsätzlich zwei Möglichkeiten ein Netzwerk zu konfigurieren.

1. Sie erstellen ein Projekt mit der erforderlichen Anzahl von Stationen. Die Netzwerkkonfiguration (Busanschluss und Adressvergabe) jeder einzelnen Station erfolgt im Hardware-Manager. Hier wird zunächst der Aufbau konfiguriert. In den Objekteigenschaften der projektierten CPU kann dann der Schnittstelle eine Adresse zugewiesen und die Vernetzung aktiviert werden.

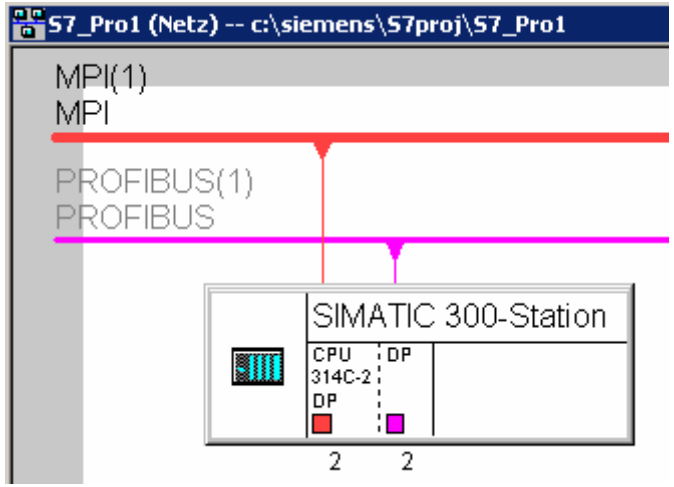
2. Die Netzwerkkonfiguration erfolgt mit dem graphischen Tool NetPro. Dieses Tool wird beim Doppelklick auf das Netzwerksymbol MPI(1) bzw. PROFIBUS(1) im Projektmanager gestartet.



Wurden (wie unter 1.) bereits Stationen konfiguriert, so können diese unmittelbar ins Arbeitsfeld eingefügt und verbunden werden.
Das Einfügen einer leeren Station, führt Sie automatisch in den Hardwarekonfigurator. Erst nach der Konfiguration ist eine Verbindung möglich.

Änderungen der Netzwerkkonfiguration werden in beiden Programmen angezeigt. Werden viele Stationen miteinander vernetzt, so ist die Arbeit mit NetPro wegen der grafischen Darstellung aller Geräte übersichtlicher (siehe Bild).

Die Bedienung von NetPro ist sehr einfach und soll hier nicht weiter beschrieben werden.

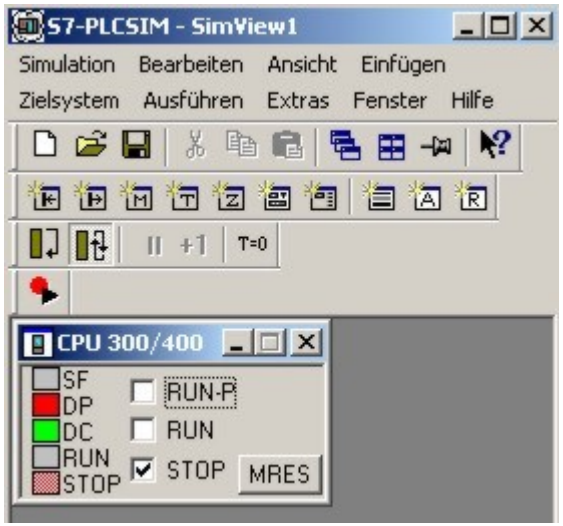


Anschluss einer S7-314c2 DP an ein MPI- u. Profibus- Subnet

4.1.6 Der S7 Simulator

Der eingebaute S7-Simulator ermöglicht den Programmtest ohne angeschlossenes Automatisierungsgerät. Dies ist zu Testzwecken, oder wenn man nur die Sprache trainieren will, ganz hilfreich. Damit das Programm auf diese Weise getestet werden kann, muss zuvor der Simulator wie folgt aus dem Projektmanager gestartet werden.

Der Simulator „S7-PLCSIM“ erscheint danach in einem eigenen Fenster. Die bereits vorhandene CPU kann nun um die gewünschten Ein-/Ausgabe-Baugruppen, Zähler und Zeitgeber erweitert werden.



Der Simulator kann auf diese Weise an die Bedürfnisse des Programms angepasst werden. Zusammenfassend sind die folgenden Arbeitsschritte zu tun, damit ein Programm im Simulator getestet werden kann:

1. SIMATIC-Manager starten und Projekt definieren.
2. Bausteine definieren und mit dem Editor programmieren.
3. Organisationsbaustein OB1 mit Programmeinsprung anlegen.
4. Simulator starten und konfigurieren
5. Bausteine aus dem Manager ins Zielsystem übertragen.
6. Simulator starten (CPU auf RUN setzen)

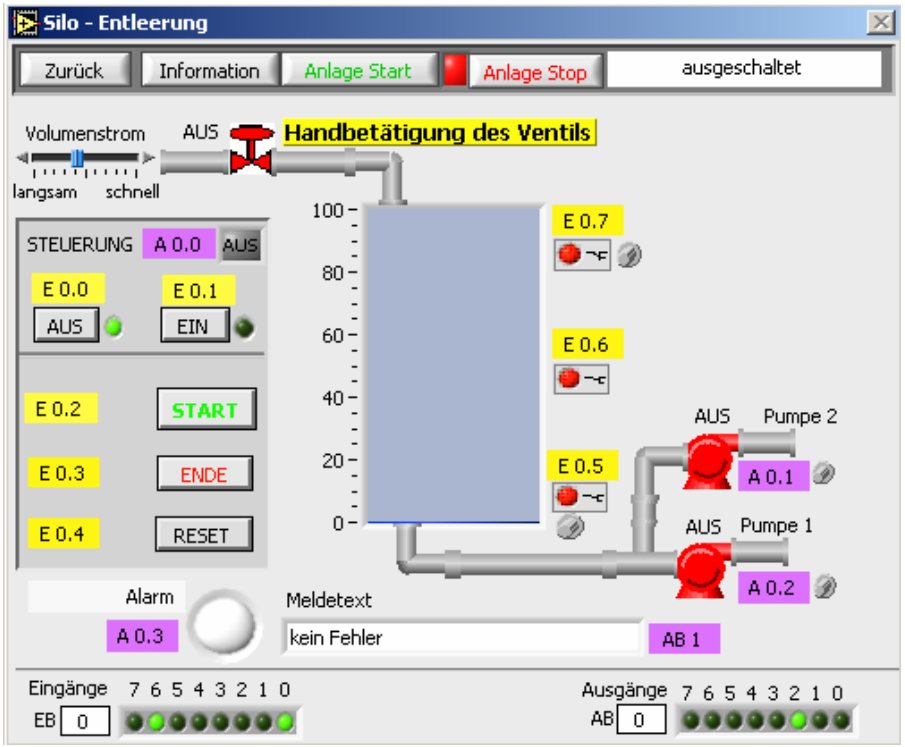
4.2 Prozessvisualisierung und Simulation mit LabVIEW

Der oben beschriebene Simulator ist zum Testen kleiner Programme oder Programmmodule sehr hilfreich. Mit ihm kann ein Programm jedoch nur statisch getestet werden, d.h. der Anwender muss die Eingangs-Bits/Bytes von Hand setzen/rücksetzen. Ein dynamischer Vorgang mit vielen Ein-/Ausgängen ist für den Anwender nicht oder nur sehr schwer überschaubar.

In unserem Labor haben wir die Möglichkeit geschaffen, dass die SPS während des Betriebes mit dem PC Daten austauschen kann. Diese Kommunikation erfolgt seriell mit einer Übertragungsrate von 38.4 KBaud. Auf diese Weise wird es möglich, komplexe Prozesse auf dem PC zu simulieren/visualisieren und diese mit der SPS zu steuern. Wir sparen durch diese Möglichkeit die Bereitstellung teurer Prozessmodelle.

Zur Simulation solcher Prozessmodelle verwenden wir seit dem WS 2004 das LabVIEW- Programm ProMod der Firma Deltalogic. Es enthält viele interessante Prozessmodelle aus unterschiedlichen Bereichen der Industrie.

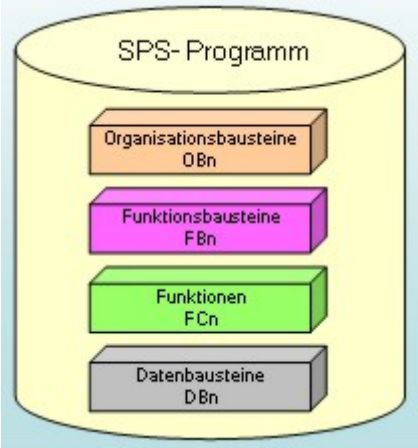
Das Programm kommuniziert über einen Treiber (AG-LINK, Fa. Deltalogic) via RS232 mit unseren AGs. Darüber hinaus ist eine Steuerung von ProMod auch mit dem SIEMENS SPS-Simulator PLCSim möglich.



ProMod Prozessbeispiel (Steuerung einer Behälterentleerung)

5. Strukturierte Programmierung

Eine Grundregel für jeden Programmierer ist die Strukturierte Programmierung. Sie sorgt dafür, dass ein Programm jederzeit übersichtlich und auch für Dritte überschaubar und nachvollziehbar ist. Auf höchster Ebene bietet STEP 7 zur Strukturierung des Programms Bausteine an. Ein Baustein kann man sich als eine Ansammlung von Befehlen vorstellen, welche ein bestimmtes Teilproblem lösen. Jeder Baustein ist damit im Prinzip ein Unterprogramm.



5.1 Der Organisationsbaustein (OB)

Organisationsbausteine enthalten in der Regel Befehle, deren Ausführung abhängig von bestimmten Ereignissen des Systems ist. So wird zum Beispiel der OB1 regelmäßig vom Betriebssystem der CPU aufgerufen (freier Zyklus). Der OB1 stellt damit den Startpunkt eines SPS-Programms dar. Von dort aus wird in die einzelnen Bausteine des Programms verzweigt. Sind alle Befehle des OB1 abgearbeitet, beginnt die CPU nach einer Betriebssystemroutine wieder beim ersten Befehl dieses Bausteins.

Wichtig: **Jedes STEP 7-Programm benötigt einen OB1.**

Neben dem OB1 gibt es weitere OBs. Jeder dieser Bausteine ist einer Prioritätsklasse zugeordnet, von denen es 28 gibt. Durch die Prioritätsklasse wird die Reihenfolge der vom Betriebssystem aufgerufenen OBs festgelegt. Die Prioritätsklassen können mit Ausnahme der OB1, 121 und 122 vom Anwender vorgegeben werden. Die nachfolgende Tabelle enthält die S7-OBs und deren Default-Prioritäten.

Baustein	Priorität	Ereignis
OB1	1	zyklische Programmbearbeitung
OB10-OB17	2	Uhrzeitalarm
OB20-OB23	3 – 6	Verzögerungsalarm
OB30-OB38	7 – 15	Weckalarm, Zyklustrigger
OB40-OB47	16 – 23	Prozessalarm
OB50-OB51	24	Kommunikationsalarm
OB60	25	Mehrprozessoralarm
OB80-OB87	26	Asynchrone Fehler
OB100-OB101	27	Anlauf
OB121-OB122	Gleiche Priorität wie aufrufender Fehler	Synchronfehler

S7-Organisationsbausteine mit Voreingestellter Priorität

5.2 Die Funktion (FC)

Eine Funktion stellt ein Unterprogramm dar. Eine Funktion kann Formalparameter besitzen, die beim Aufruf der Funktion mit Aktualparametern zu versorgen sind.
Formalparameter sind Werte oder Operanden, die innerhalb des SPS-Programms der Funktion verarbeitet werden.
Bei einer Funktion **müssen** alle Formalparameter mit Aktualparameter versorgt werden. Eine Funktion kann einen Funktionswert zurückliefern, kann allerdings darüber hinaus, weitere sog. Ausgabeparameter besitzen.
Funktionen werden immer dann verwendet, wenn **keine statischen Daten** zur Ausführung benötigt werden (siehe Kap.5.3).

5.3 Der Funktionsbaustein (FB)

Im Gegensatz zu Funktionen haben Funktionsbausteine die Möglichkeit, Daten zu speichern. Diese Fähigkeit wird durch einen Datenbaustein erreicht, welcher dem Aufruf (der Instanz) eines FBs zugeordnet ist. Ein solcher Datenbaustein wird **Instanz-Datenbaustein** genannt. Ein Instanz-DB besitzt die gleiche Datenstruktur wie der ihm zugeordnete FB. Hier werden die Daten bis zum nächsten Aufruf des FB abgelegt. Die Daten des Instanz-DB können auch aus anderen Bausteinen gelesen und geschrieben werden.

5.4 Der Datenbaustein (DB)

In einem Datenbaustein werden Daten abgelegt. Der Datenbaustein enthält somit keine STEP-7 Befehle. STEP 7 unterscheidet zwei Datenbausteine. Es sind dies der „normale“ Globaldatenbaustein, mit einer vom Programmierer festgelegten Datenstruktur und der Instanz-DB, welcher die Datenstruktur des Funktionsbausteins besitzt, dem er zugeordnet ist.

In STEP 7 sind Datenbausteine **byteorientiert**. Dies bedeutet, die einzelnen Datenwörter überschneiden sich.

Beispiel:
Beim Zugriff auf die Datenwörter 0 und 1 bzw. 1 und 2 überschneiden sich die Daten.
Man sollte daher bei Wortoperationen nur **geradzahlige** Adressen verwenden.

Datenbaustein / Adresse	Datenwort / Adresse	Datenwort / Adresse
Byte 0	0 HiByte	
Byte 1	LoByte	1 HiByte
Byte 2	2 HiByte	LoByte
Byte 3	LoByte	

Beim Zugriff auf ein Datenwort befindet sich das HiByte auf der niederwertigen Byte-Adresse.

5.5 Der Systemdatenbaustein (SDB, SFC, SFB)

Systemdatenbausteine können vom Anwender weder erstellt, noch geändert werden. Diese Bausteine werden von der Programmiersoftware erzeugt, um z.B. die Konfigurationsdaten einer Baugruppe darin abzulegen.

Neben den SDBs gibt es auch System- Funktionen und -Funktionsbausteine. Diese enthalten systemspezifische Unterprogramme.

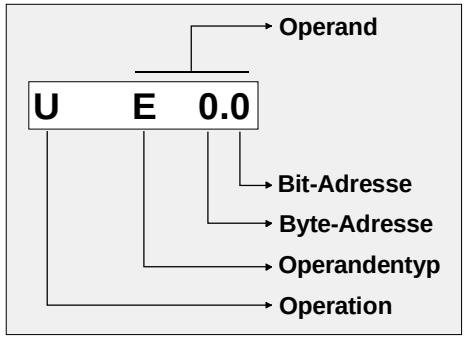
6. Darstellungsarten

STEP 7 stellt drei unterschiedliche Darstellungsarten für den Programmcode zur Verfügung. Die Darstellungsarten (AWL, FUP, KOP) können im Editor während der Programmentwicklung beliebig umgeschaltet werden.

6.1 Anweisungsliste (AWL)

Die Anweisungsliste (AWL) ist die klassische Darstellungsart für STEP 7-Code. Sie ist zeilenorientiert und ähnelt eher einer maschinennahen Prozessorsprache bei der die einzelnen Befehlszeilen nahezu den Bearbeitungsschritten der CPU entsprechen. Die AWL ist die einzige Darstellungsart, in der alle Befehle von STEP7 dargestellt werden können.

Aufbau einer AWL-Zeile



Zum Verständnis kann man sich folgenden Sachverhalt merken:

- Die Operation gibt an, was getan werden soll
- Der Operand gibt an, worauf die Operation angewendet wird

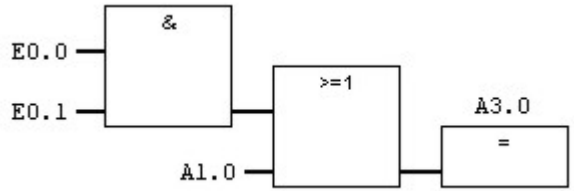
Eine Ausnahme hinsichtlich der angegebenen Bit-Adresse bilden die Zähler und Zeiten. Dort werden keine Bit-Adressen angegeben. Außerdem gibt es Befehle, bei denen kein Operand anzugeben ist, z.B. die arithmetischen Funktionen (>=I, +I, usw.).

6.2 Funktionsplan (FUP)

Der Funktionsplan ist eine Art graphischer Programmcode-Darstellung. Die Operationen werden dabei durch Blockstrukturen dargestellt. Die Ein- und Ausgänge der Blöcke sind konfigurierbar und enthalten wie auch bei der AWL die

Operanden. Der Funktionsplan ähnelt der bekannten Darstellung von logischen Gattern und ist wegen seiner graphischen Komponente ausgesprochen übersichtlich. Zur Programmierung in FUP-Darstellung steht eine umfangreiche Funktionsblockbibliothek zur Verfügung. Das folgende Bild zeigt einen Ausschnitt aus dem Programmeditor in FUP-Darstellung.

Das Ausgangsbit A3.0 ist dann auf Hi-Pegel, wenn entweder die Eingänge E0.0 und E0.1 oder der Ausgang A1.0 einen Hi-Pegel liefern.

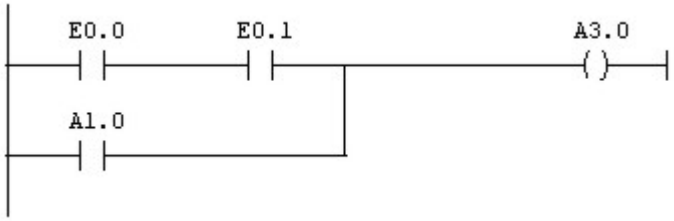


Nicht alle STEP7-Funktionen können in der Darstellung FUP eingefügt werden. Ausnahmen sind z.B. die Lade und Transferbefehle. Auch die indizierte Programmierung ist nur in der Darstellung AWL möglich.

6.3 Kontaktplan (KOP)

Die Darstellungsart „Kontaktplan“ (KOP) spricht im Wesentlichen den Programmierer an, der sich häufig mit Stromlaufplänen beschäftigt. Dies sind in der Regel Handwerker/Meister aus den elektrotechnischen/mechanischen Berufen. Das nachfolgende Bild zeigt unser Beispiel in der Darstellung KOP.

Ein Stromfluss durch das Relais A3.0 kommt erst dann zustande, wenn entweder A1.0 oder aber E 0.0 und E 0.1 Strom führen.



Wir werden in unseren Übungen die Darstellung im Kontaktplan nicht verwenden.

7. Operanden in STEP 7

In jedem SPS-Programm wird mit Operanden gearbeitet. Will man beispielsweise den Status eines Tasters, welcher an einer Eingangsbaugruppe der SPS angeschlossen ist, im SPS-Programm abfragen, so verwendet man einen Operanden des Typs E (Eingänge).

7.1 Eingangs- Ausgangsoperanden

Die Operanden des Typs **E** und **A** sind so genannte Bit-Operanden, d.h. diese haben entweder den Zustand 1 oder 0.
Eingänge und Ausgänge können als Bit-, Byte-, Wort und Doppelwort angesprochen werden.

E 0.0	Bit 0 des Eingangsbytes 0
EB 1	Eingangsbyte 1
EW 0	Eingangswort 0 (Byte 0 = Hi-Byte, Byte 1 = LoByte
ED 0	Eingangsdoppelwort 0 (Byte 0, 1, 2 und 3)

7.2 Merkeroperanden

Die Operanden des Typs **M** (Merker) dienen zum Verarbeiten und „merken“ interner Zwischenergebnisse. Merkzustände werden in einem bestimmten Speicherbereich der CPU abgelegt. Merker sind im Gegensatz zu Ein- oder Ausgängen nur interne Zustände.

M 12.1	Bit des Merkerbyte 12
MB 2	Byte des Merker 2
MW 5	Merkerwort 5 (2Byte)
MD 7	Merkerdoppelwort (4Byte)

7.3 Lokaloperanden

Lokaloperanden sind im Prinzip temporäre Variablen. Zur Laufzeit des Programms wird jedem Baustein ein bestimmter Speicherbereich zugewiesen, in dem die temporären Variablen ablegt werden. Diese Variablen sind nach dem Verlassen des Bausteins nicht mehr gültig. Sie dienen lediglich als Zwischenspeicher.

L 10.2	Bit 2 des Lokaldatenbyte 10
LB 1	Lokaldatenbyte 1
LW 2	Lokaldatenwort 2 (2Byte)
LD 3	Lokaldatendoppelwort (4Byte)

7.4 Daten eines Datenbausteins

Ein besonderer Operandenbereich, stellt der Bereich Daten (**D**) dar. Dieser kann erst verwendet werden, wenn ein Datenbaustein aktiv ist. Mit diesen Operanden ist es möglich, Inhalte von Datenwörtern zu verarbeiten.

DBX 0.0 Bit 0 des Byte 0 aus Datenbaustein
DBB1 Byte 1 aus Datenbaustein
DBW1 Wort 1 aus Datenbaustein (2Byte)

7.5 Timer

Operanden vom Typ **T** ermöglichen es, Zeitverhalten innerhalb eines SPS-Programms zu realisieren. Dazu stehen verschiedene Zeittypen zur Verfügung. Näheres über Zeiten in einem späteren Kapitel.

T1 Zeitgeberbaustein 1

7.6 Zähler

Operanden des Typs Z bieten eine Zählfunktion. Es kann dabei ein Vorwärts- und Rückwärtszähler realisiert werden. Auch für die Zähler gibt es ein eigenes Kapitel.

Z1 Zählerbaustein 1

7.7 Peripherieeingänge / Ausgänge

Mit den Operanden PE / PA können die physikalischen Ein- / Ausgänge direkt ein-/ausgelesen werden. Im Gegensatz zum Operandentyp E / A, wo auf die Daten im Prozessabbild der Ein- /Ausgänge zugegriffen wird.
Mit Operanden dieses Typs kann ein einzelnes Bit nicht gelesen bzw. ausgegeben werden.

PEB 10 Byte 10 der Eingangsperipherie
PEW12 Wort 12 der Eingangsperipherie (2Byte)
PAB 0 Byte 0 der Ausgangsperipherie
PAD 12 Doppelwort 12 der Ausgangsperipherie (4Byte)

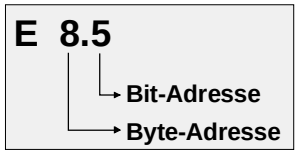
8. Adressierung der Operanden

Dieses Kapitel beschreibt die verschiedenen Adressierungsmöglichkeiten von Operanden.
Bei der Verwendung von Operanden muss immer die Adresse mit angegeben werden. In STEP 7 sind folgende Operanden möglich:

Operandenart	Datenbreite In Bit	Beispiel
BIT	1	E 4.4, M4.4, DBX 3.3
BYTE	8	EB4, MB4, DBB10
WORD	16	EW4, MW4, DBW10
DWORD	32	ED4, MD4, DBD10

8.1 Bitoperanden

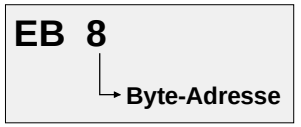
Bei Bit-Operanden muss immer die Byte- und Bitadresse angegeben werden. Byte- und Bitadresse werden immer durch einen Punkt getrennt:



Die Bitadresse muss hierbei immer zwischen 0 und 7 liegen. Die max. Byteadresse ist vom CPU-Typ abhängig.

8.2 Byteoperanden

Bei Byteoperanden fehlt die Angabe der Bits. Der Operand sieht wie folgt aus:



Das gesamte Byte 8 wird gelesen, geschrieben oder gespeichert.

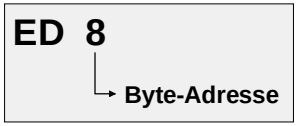
8.3 Wortoperanden

Mit Wortoperanden, wie EW 8, kann ein Eingangswort ab der Byteadresse 8 in den AKKU (Akkumulator) geladen werden.

Dabei liest die CPU EB 8 (Hi-Byte des Wortes) und EB 9 (Lo-Byte des Wortes).

8.4 Doppelwortoperanden

Mit Doppelwortoperanden können ähnlich wie unter 8.3 insgesamt 4 Byte gelesen, ausgegeben oder gespeichert werden.



8.5 Hinweise zur Adressierung

An dieser Stelle sei nochmals darauf hingewiesen, dass es beim Zugriff auf Datenwörtern und -Doppelwörtern zu Überschneidungen kommen kann.

In STEP 7 werden Daten byteorientiert gespeichert !!!

Das Ausgangswort AW32 besteht aus AB32 und AB33.
Das Ausgangswort AW33 besteht aus AB33 und AB34.

Dies bedeutet, dass AB33 in AW32 und in AW33 enthalten ist. Um diese Überschneidung zu vermeiden, sollten immer **geradzahlige Adressen verwendet** werden. (AW32, AW34, AW36).

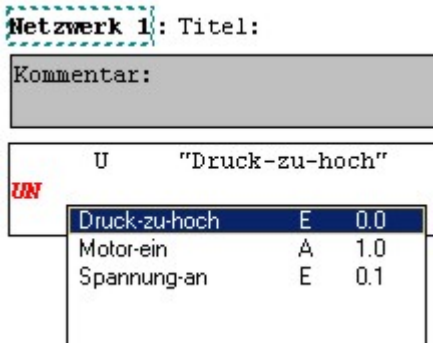
9. Symbolische Programmierung

Bisher haben wir gelernt, dass Operanden in einem SPS-Programm Eingänge, Ausgänge, Merker usw. beschreiben. In der Regel verbirgt sich jedoch hinter jedem Eingang ein Signalgeber und hinter jedem Ausgang ein Signalempfänger. Werden in einem Programm viele Ein-/Ausgänge benutzt, so wird es für den Programmierer zunehmend schwieriger, die Zuordnung vorzunehmen. Aus diesem Grund kann man jedem Operanden ein Symbol (kurzer Text, z.B. Motor, Lampe) zuordnen. Dieser Text wird dann an Stelle des Operanden im Quelltext angegeben.

In Kapitel 4.1.4 wurde bereits auf den STEP 7-Symboleditor eingegangen. Jeder Symboleintrag in der Symboldatei besteht demnach insgesamt aus 4 Informationen:

1. Das Symbol (maximal 24 Zeichen)
2. Absolutoperand (E, A, M usw.)
3. Der Datentyp (BOOL, usw.)
4. Ein Kommentar (maximal 80 Zeichen)

Damit der Programmeditor die symbolische Anzeige und Eingabe unterstützt, muss diese Eigenschaft unter „**Ansicht - Anzeige mit – symbolische Darstellung**“ eingeschaltet werden. Symbole können entweder direkt oder über eine angezeigte Liste eingetragen werden. Bei der direkten Eingabe wird der Operand (das Symbol) wie unten dargestellt in Hochkommata eingegeben. Klickt man an Stelle des Operanden auf die rechte Maustaste, wählt „Symbol einfügen“, dann öffnet sich ein Pull-Down-Menü mit allen definierten Symbolen. Der Eintrag erfolgt durch Anklicken, ganz ohne Schreibarbeit.



Es dürfen keine Symbole mehrfach verwendet werden. Ist dies dennoch der Fall, wird die Datei vom Editor nicht abgespeichert (Fehlermeldung).

Anmerkung:
Datenwörter können nicht durch ein Symbol ersetzt werden. Um ein Datenwort eindeutig bestimmen zu können, ist die Angabe des Datenbausteins notwendig. Die kombinierte Angabe „DBW100.DBW10“ ist im Symboleditor nicht vorgesehen.
Dies hat folgenden Grund:
Wenn in S7 ein Datenbaustein erstellt wird, sind Variablen im Kopf des Datenbausteins zu deklarieren. Diese Variablen können einen beliebigen Namen haben. Datenwörter können über diese Variablen angesprochen werden.

Beispiel:
Die Variable „DB100.Betriebsstunden“ repräsentiert je nach Datentyp (BYTE, WORD, …) einen bestimmten Datenbereich im Datenbaustein.
Datenbausteine können demnach auch ohne Zuordnungsliste „symbolisch“ programmiert werden.
Folgende Operanden können durch ein Symbol ersetzt werden:

Operand	Beschreibung	Beispiel
E	Eingang	E30.2
EB	Eingangsbyte	EB 2
EW	Eingangswort	EW 10
ED	Eingangsdoppelwort	ED 20
A	Ausgang	A 10.1
AB	Ausgangsbyte	AB 2
AW	Ausgangswort	AW 6
AD	Ausgangsdoppelwort	AD 8
M	Merker	M 2.1
MB	Merkerbyte	MB 3
MW	Merkerwort	MW 10
MD	Merkerdoppelwort	MD 18
PEB	Periferie-Eingangs-Byte	PEB 20
PEW	Periferie-Eingangs-Wort	PEW 40
PED	Periferie-Eingangs-Doppelwort	PED 26
PAB	Periferie-Ausgangs-Byte	PAB 20
PAW	Periferie-Ausgangs-Wort	PAW 40
PAD	Periferie-Ausgangs-Doppelwort	PAD 26
T	Timer	T 1
Z	Zähler	Z 2
FB	Funktionsbaustein	FB 3
FC	Funktion	FC 5
OB	Organisationsbaustein	OB 10
DB	Datenbaustein	BD 1
SFB	System-Funktionsbaustein	SFB 10
SFC	System-Funktion	SFC 20

10. Verknüpfungsoperationen

Verknüpfungsoperationen dienen dazu, bestimmte „wenn…dann“-Befehle zu definieren.

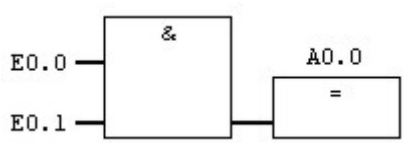
10.1 UND-Verknüpfung

Die UND-Verknüpfung zwischen zwei Eingängen ergibt als Ergebnis ‚1’, wenn **alle** Eingänge den Signalzustand ‚1’ haben.

AWL-Beispiel:

```
U     E 0.0     Wenn E0.0 und
U     E 0.1     E0.1 ‚1’ ist, dann
=     A 0.0     Ausgang A0.0 auf ‚1’ schalten
```

FUP-Beispiel:



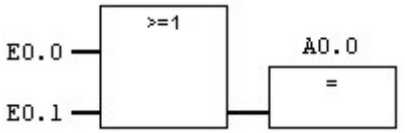
10.2 ODER-Verknüpfung

Die ODER-Verknüpfung zwischen zwei Eingängen ergibt als Ergebnis ‚1’, wenn **mindestens** 1 Eingang den Signalzustand ‚1’ hat.

AWL-Beispiel:

```
O     E 0.0     Wenn E0.0 oder
O     E 0.1     E0.1 ‚1’ ist, dann
=     A 0.0     Ausgang A0.0 auf ‚1’ schalten
```

FUP-Beispiel:



10.3 EXKLUSIV-ODER-Verknüpfung

Die EXKLUSIV-ODER-Verknüpfung zwischen zwei Eingängen ergibt als Ergebnis ‚1', wenn nur einer der beiden" Eingang den Signalzustand ‚1' hat.

AWL-Beispiel:

X	E 0.0	Wenn nur der Eingang E0.0 oder
X	E 0.1	nur der Eingang E0.1 ‚1' ist, dann
=	A 0.0	Ausgang A0.0 auf ‚1' schalten

FUP-Beispiel:

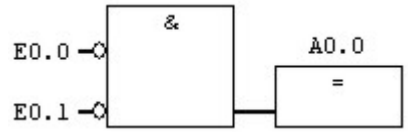
10.4 NICHT-Verknüpfung

Die NICHT-Verknüpfung gibt es in STEP 7 nur in Verbindung mit einer UND/ODER-Verknüpfung. Bei der NICHT-Verknüpfung wird immer der invertierte Zustand des Operanden betrachtet. Daraus ergibt sich, dass die NICHT-Verknüpfung den Signalzustand ‚0' abfragt.

AWL-Beispiel:

UN	E 0.0	Wenn der Eingang E0.0 = ‚0' ist und
UN	E 0.1	der Eingang E0.1 = ‚0' ist, dann
=	A 0.0	Ausgang A0.0 auf ‚1' schalten

FUP-Beispiel:

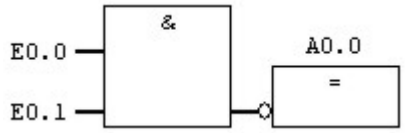


10.5 UND-NICHT-Verknüpfung

Die UND-NICHT-Verknüpfung negiert das Ergebnis der UND-Verknüpfung.
AWL-Beispiel:

U	E 0.0	Wenn der Eingang E0.0 = ‚1' ist und
U	E 0.1	der Eingang E0.1 = ‚1' ist, dann
		ist das Verknüpfungsergebnis (VKE) = 1
NOT		negiert das VKE
=	A 0.0	Ausgang A0.0 = 0 schalten

FUP-Beispiel:



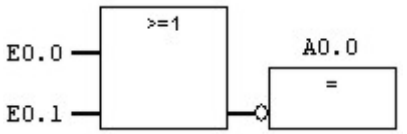
10.6 ODER-NICHT-Verknüpfung

Die ODER-NICHT-Verknüpfung negiert das Ergebnis der ODER-Verknüpfung.

AWL-Beispiel:

U(		Klammerregeln, siehe Kap. 10.1)
O	E 0.0	Wenn der Eingang E0.0 = ‚1' ist oder
O	E 0.1	der Eingang E0.1 = ‚1' ist, dann
)		ist das Verknüpfungsergebnis (VKE) = 1
NOT		negiert das VKE
=	A 0.0	Ausgang A0.0 = 0 schalten

FUP-Beispiel:



10.7 Verknüpfungsergebnis (VKE)

Bei einer Verknüpfung zweier Operanden wird das Ergebnis der Verknüpfung als VKE (Verknüpfungsergebnis) bezeichnet.

Programmbeispiel:

Zeile	AWL	Status des Operanden	VKE
0001	O E 0.0	0	0
0002	O E 0.1	1	1
0003	O E 0.2	0	1
0004	= A 0.0	1	1

Das VKE ist demnach ein Zwischenspeicher, der entweder ‚1' oder ‚0' ist. Wird eine Verknüpfung neu begonnen (Zeile 1), wird das VKE auf den Wert des Operanden (‚0' oder ‚1') gesetzt. Bei den nachfolgenden Verknüpfungen (Zeile 2 und Zeile 3) wird der Operand mit dem VKE verknüpft.
Dies wird so lange durchgeführt, bis das VKE einem Operanden zugewiesen wird (Zeile 4) oder exakter ausgedrückt, bis ein VKE-begrenzender Befehl bearbeitet wird.

VKE-Begrenzung
Nachdem das VKE zugewiesen worden ist, wird das VKE begrenzt und es kann eine neue Verknüpfung begonnen werden.

VKE begrenzende Operationen

VKE- begrenzende Operation	Beispiel
Zuweisung	= M0.0, = A2.1
Klammerauf-Befehle	U(, O(, …
Setz- und Rücksetzbefehle	S M0.0, S A2.0
Zeitoperationen	SE T1, SA T10, …
Zähloperationen	ZV Z1, ZR Z1, …
Sprungbefehle	SPA M001, SPN M002, …
Rücksprungbefehle	BE, BEB, BEA

10.8 Klammerbefehle

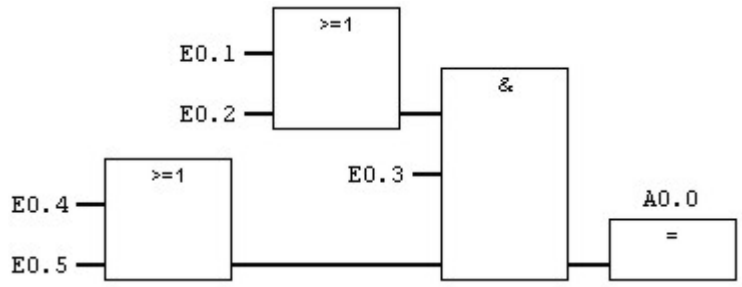
Mit Klammerbefehlen kann man die gewünschte Reihenfolge von Verknüpfungen festlegen.
Für die Anwendung von Klammern gelten die folgenden Regeln:

- Es müssen genauso viele Klammern geschlossen werden, wie geöffnet wurden.
- Eine Verknüpfung mit Klammern darf nicht über Netzwerkgrenzen hinausgehen.
- Innerhalb einer Klammer sollte man keine Sprungmarken platzieren, da sonst das Ergebnis nicht nachvollziehbar ist.
- Klammern dürfen auch verschachtelt sein. Die maximale Klammerverschachtelung muss im Gerätehandbuch des jeweiligen AGs nachgelesen werden.
- Ein **Klammer-Auf-Befehl ist immer VKE-begrenzend**, d.h. es fängt eine neue Verknüpfung an.
- Ein **Klammer-Zu-Befehl ist nicht VKE-begenzend**, da die Klammer-Zu-Operation als Zwischenspeicher verwendet wird.

Folgende Befehle stehen für die Klammersetzung zur Verfügung:

Operation	Erklärung
U(	UND-Klammer aufmachen
O(	ODER-Klammer aufmachen
X(	EXCLUSIV-ODER-Klammer aufmachen
UN(	UND-NICHT-Klammer aufmachen
ON(	ODER-NICHT-Klammer aufmachen
XN(	EXCLUSIV-ODER-NICHT-Klammer aufmachen
)	Klammer schließen

Beispiel:
Die folgende Schaltung soll in AWL umgesetzt werden:



Lösung:

```
U(              |
O   E 0.0       |--- Block 1
O   E 0.1       |
)
U   E 0.3       ---- Block 2   A0.0 = Block 1 UND Block 2 UND Block 3
U(              |
O   E 0.4       |--- Block 3
O   E 0.5       |
)
=   A 0.0
```

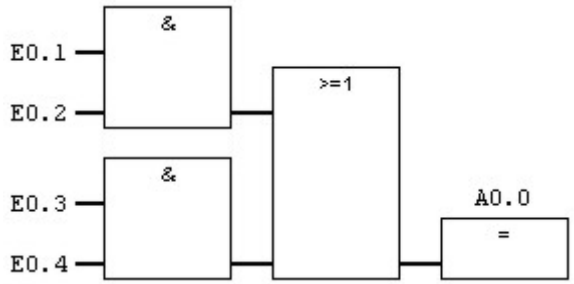
Die Klammerung der ODER-Verknüpfung bewirkt, dass die ODER-Verknüpfung vor der UND-Verknüpfung bearbeitet wird.

Innerhalb der Klammer fängt eine neue Verknüpfung an, da der Klammer-Auf-Befehl VKE-begrenzend ist.
Der Klammer-Zu-Befehl ist nicht VKE-begrenzend. Deshalb kann nach einem Klammer-Zu-Befehl das Ergebnis der Klammer weiter verknüpft werden.

10.9 ODER-Verknüpfung von UND-Verknüpfungen

Möchte man einen Block aus UND-Verknüpfungen mit ODER verknüpfen, dann kann man den Befehl „O“ verwenden.
Eine Klammerung ist nicht notwendig, da eine UND-Verknüpfung vor einer ODER-Verknüpfung bearbeitet wird.

Beispiel:



Lösung:

Variante 1	Variante 2
U E 0.1 U E 0.2 O U E 0.3 U E 0.4 = A 0.0	O(U E 0.1 U E 0.2) O(U E 0.3 U E 0.4) = A 0.0

Beide Varianten werden von STEP 7 akzeptiert.

Der Oder-Befehl ist ein separater STEP 7-Befehl. Er wird wie ein ODER-Klammer-Auf-Befehl gesehen. Durch den begrenzenden Befehl „= A0.0“ wird die gedachte Klammer wieder geschlossen.

10.10 Setz- Rücksetzbefehle

Mit einem Setzbefehl kann man Binäroperanden auf ‚1’ setzen. Dieser bleibt dann solange auf ‚1’, bis er wieder zurückgesetzt wird.

Mit einem Rücksetzbefehl kann man Binäroperanden auf ‚0’ setzen. Dieser bleibt dann solange auf ‚0’, bis er wieder gesetzt wird.

Diese Befehle werden auch Speicher genannt, da diese den Zustand des Operanden speichern.

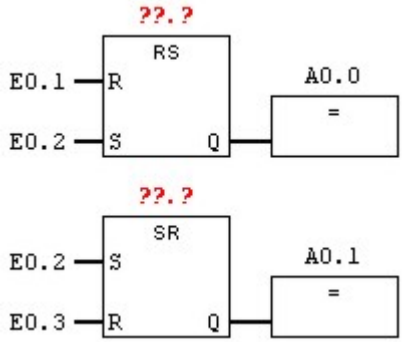
Beispiel:

```
U   E 0.1
S   A 0.1       Ausgang A 0.1 auf ‚1’ setzen
U   E 0.2
R   A 0.1       Ausgang A 0.1 auf ‚0’ setzen
```

Setz- und Rücksetzdominanz

In obigem Beispiel ist es denkbar, dass beide Eingänge den Wert ‚1’ haben. Dann wird der Ausgang zunächst gesetzt und kurz danach wieder rückgesetzt. Man spricht in diesem Fall von **Rücksetzdominanz**.

Wird der Rücksetz-Befehl vor dem Setz-Befehl programmiert, dann spricht man von **Setzdominanz**.



Bei einem setzdominanten Speicher steht der Setzeingang unterhalb des Rücksetzeinganges (obere Darstellung FUP).

Bei einem rücksetzdominanten Speicher steht der Rücksetzeingang unterhalb des Setzeinganges (untere Darstellung FUP).

11. Datentypen

Datentypen kennzeichnen in STEP 7 den Aufbau und die Länge von Variablen und Operanden. Der Eingang E 5.0 bezeichnet ein einzelnes Bit des Eingangsbytes 5. Dieses Bit kann zwei Zustände (1 oder 0) annehmen. Sein Datentyp ist daher der Typ „BOOL“.

Zum Verständnis der unterschiedlichen Datentypen sind nachfolgend noch einmal die unterschiedlichen Datenlängen in STEP 7 angegeben.

Bit:
Ein Bit ist die kleinste darstellbare Informationseinheit.

Byte:

Eingangsbyte EB 1.x							
7	6	5	4	3	2	1	0

Wort:

Eingangswort EW0															
Eingangsbyte EB0.x									Eingangsbyte EB1.x						
7	6	5	4	3	2	1	0		7	6	5	4	3	2	1	0

Doppelwort:

Eingangsdoppelwort ED0											
Eingangsbyte 0.x			Eingangsbyte 1.x			Eingangsbyte 2.x			Eingangsbyte 3.x		
7	…	0	7	…	0	7	…	0	7	…	0

Die Programmiersprache STEP 7 kennt drei Arten von Datentypen. Dies sind:

- **Elementare Datentypen**
- **Zusammengesetzte Datentypen**
- **Parametertypen**

Nachfolgend werden die Datentypen dargestellt.

11.1 Elementare Datentypen

Elementare Datentypen haben eine maximale Länge von 32 Bit (Doppelwort). Diese Datentypen können als ganzes mit einem STEP 7-Befehl angesprochen (z.B. geladen) werden.
Die nachfolgende Tabelle zeigt die elementaren Datentypen mit deren Konstantenschreibweise.

Datentyp	Beschreibung	Breite In Bit	Konstantenbeispiel
BOOL	Einzelnes Bit	1	FALSE (0), TRUE (1)
BYTE	Hex-Zahl	8	B#16#A1
CHAR	Einzelnes ASCII-Zeichen	8	'T'
WORD	Vorzeichenlose Zahl, darstellbar in Hex, binär, als Zähler-Wert 2 x 8Bit	16	W#16#ABCD 2#00110011_11110000 C#128 B#(81, 54)
INT	Integer oder Festpunktzahl	16	-12123 -32768 bis 32767
S5TIME	Zeitwert im S5-Zeitformat	16	S5T#1h10m20s
DATE	Datumsangabe	16	D#1998-04-14
DWORD	Vorzeichenlose Zahl Darstellbar in Hex, binär 4 x 8 Bit	32	DW#16#1234_5678 2#10001000_10011001_ 11110000_11110000 B#(12, 13, 14, 15)
DINT	Integer 32-Bit	32	L#35434 -2147483648 bis 2147483647
REAL	Gleitpunkt	32	12.3 oder 1.230000e+01
TIME	IEC-Zeitformat	32	T#14d20h45m23s123ms
TOD	Tageszeit	32	TOD#17:53:17:333

11.2 Zusammengesetzte Datentypen

Zusammengesetzte Datentypen haben eine Länge, die 32 Bit überschreitet. Aus diesem Grund kann ein einzelner STEP 7-Befehl nur einen Teil dieses Datentyps verarbeiten. Zur vollständigen Bearbeitung sind mehrere Befehle nötig. Diese Datentypen sind in der nachfolgenden Tabelle aufgeführt.

Datentyp	Beschreibung	Konstantenbeispiel
DT	Uhrzeit und Datumsangabe 64-Bit	
STRING	Angabe einer ASCII- Zeichenkette mit der max. Länge 254	'Dies ist ein String'
ARRAY	Zusammenfassung von Elementen (Feldern) gleichen Typs, mit max. 6 Dimensionen.	-
STRUCT	Strukturen werden benutzt, um mehrere Komponenten in einem einzigen Überbegriff zusammenzufassen. Dabei können die Komponenten unterschiedlichen Datentypen angehören.	-

11.3 Parametertypen

Parametertypen sind nur in Verbindung mit Bausteinparametern verwendbar, d.h. Operanden dieses Typs können als Aktualparameter verwendet werden. In der folgenden Tabelle werden diese benannt.

Parametertyp	Beschreibung	Beispiel
BLOCK_FC	Funktion	FC20
BLOCK_FB	Funktionsbaustein	FB 3
BLOCK_DB	Datenbaustein	DB10
BLOCK_SDB	Systemdatenbaustein	SDB104
TIMER	Zeitfunktion	T12
COUNTER	Zählfunktion	Z3
POINTER	DB-Zeiger	P#E12.3
ANY	ANY-Zeiger	P#M10.0 BYTE 10

12. Lade- und Transferbefehle

Lade- und Transferbefehle kommen immer dann zum Einsatz, wenn man mit Byte-Wort- oder Doppelwortoperanden arbeitet.

12.1 Laden von Bytes, Wörtern und Doppelwörtern

Soll zum Beispiel der Inhalt eines Eingangsbytes in einem Merkerbyte zwischengespeichert werden, so entspricht dies einer Verschiebeoperation. Hierzu ist es unumgänglich, das Eingangsbyte zunächst in den AKKU (Akkumulator) zu laden. Der anzuwendende Befehl lautet:

```
L     EB   1     //Lade Eingangsbyte 1 in AKKU 1
```

CPUs der S7-Reihe besitzen insgesamt 2 AKKUs. Jeder AKKU hat eine Länge von 4-Byte, also einem Doppelwort.

AKKU 1			
High-Wort		Low-Wort	
High-Byte	Low-Byte	High-Byte	Low-Byte

Der obige Lade-Befehl schiebt den Inhalt von EB1 ins Low-Byte des Low-Wort von AKKU 1.
Der vorherige Inhalt von AKKU 1 wird automatisch in den AKKU 2 übertragen. Der vorherige Inhalt aus AKKU 2 geht dabei verloren.

Mit dem Transferbefehl wird im zweiten Schritt der Inhalt von AKKU 1 (Low-Byte) ins Merker-Byte übertragen.

```
T     MB   1     //Transferiert den AKKU 1 ins Merkerbyte
```

Die nachstehende Tabelle zeigt die Operanden, die in Verbindung mit dem Lade- und Transferbefehl verwendet werden. Die Tabelle gilt gleichermaßen für Bytes (B), Wörter (W) und auch Doppelwörter (D).

Operand	Lade-Befehl	Beschreibung
E	L EB 0	Laden eines Eingangsbytes
A	L AB 0	Laden eines Ausgangsbytes
M	L MB 0	Laden eines Merkerbytes
D	L DBB 0	Laden eines Bytes aus einem DB
PE	L PEB 0	Laden eines Bytes direkt aus der Peripherie

12.2 Laden von Konstanten

Im Folgenden wird je an einem Beispiel gezeigt, wie Konstanten der elementaren Datentypen geladen und transferiert werden.

Byte

```
L     B#16#AA     // Laden einer Byte-Konstante
T     MB    12    // Transferieren nach MB12
```

Es wird eine Hex-Zahl 'AA' geladen und in das Merkerbyte MB12 übertragen.
Die Angabe "B#16" kennzeichnet eine Hex-Zahl mit 2 Stellen.

CHAR

```
L     'A'         // Laden eines einzelnen Zeichens
T     AB    0     // Transfer in ein Ausgangsbyte
```

Es wird der ASCII-Code für das Zeichen 'A' ins Ausgangsbyte übertragen. Maximal 4 Zeichen können mit einem Befehl geladen und transferiert werden.

INT

```
L     2230        // Laden der Zahl 2230
T     MW    10    // Transferieren nach MW10
```

Der übertragbare Zahlenbereich beträgt -32768 bis 32767.

WORD

```
L     W#16#AB12   // Laden der Hex-Zahl AB12
T     AW    18    // Transferieren nach AW 18
```

Es wird eine 16-Bit breite Hex-Zahl ins Ausgangswort 18 übertragen. Die Angabe "W#16" kennzeichnet eine Hex-Zahl mit 4 Stellen.

DINT

```
L     L#3453600   // Laden einer DINT-Konstanten
T     MD    10    // Transferieren nach MD10
```

Übertragen einer 32-Bit breiten Integerkonstanten ins Merker-Doppelwort 10. Der Zahlenbereich beträgt -2147483648 bis 2147483647.

DWORD

```
L     DW#16#AAAABBBB   // Laden der DWORD-Konst.
T     AD    10         // Transferieren nach AD 10
```

Es wird eine 32-Bit breite Hex-Zahl ins Ausgangsdoppelwort 10 übertragen. Die Angabe “DW#16“ kennzeichnet eine Hex-Zahl mit 8 Stellen.

REAL

```
L     1.245600e+01     // Laden einer Zahl 12.456
T     MD    20         // Transferieren nach MD 20
```

Es handelt sich beim Datentyp REAL um eine Gleitpunktzahl, die 32-Bit breit ist. Eine Konstante dieses Typs kann als eine Dezimalzahl mit Kommastelle oder in exponentieller Darstellung eingegeben werden. Die Eingabe wird aber immer in die exponentielle Darstellung gewandelt.

S5TIME

```
L     S5T#3M20S    // Laden der Zeit 3 Minuten 20 Sekunden
T     MW    10     // Transferieren ins Merkerwort 10
```

Der Datentyp S5TIME definiert einen Zeitwert im S5-Format. Der Datentyp ist 16-Bit breit und besteht aus der Kennung “S5T#“ oder “S5TIME#“.
Eine Eingabe kann mit der Angabe von Stunden (H), Minuten (M), Sekunden (S) oder Millisekunden (MS) erfolgen.
Die maximale Zeitangabe beträgt 2 Stunden 46 Minuten und 30 Sekunden.

TIME

```
L     T#2D3M20S    // Laden der Time- Konstanten
T     MD    12     // Transferieren nach MD 12
```

Der Datentyp TIME stellt einen Zeitwert im IEC-Format dar. Der Datentyp ist 32-Bit breit. Die Eingabe erfolgt durch Angabe von Tagen (T), Stunden (S), Minuten (M), Sekunden (S) und Millisekunden (MS). Dabei müssen nicht alle Angaben erfolgen, es ist beispielsweise möglich, nur die Tage und Minuten anzugeben.

TIME OF DAY

```
L     TOD#12:23:45.0    // Laden einer Tageszeit
T     AD    20          // Transferieren nach AD 20
```

Der Datentyp TIME OF DAY ist die Angabe der Tageszeit. Der Datentyp ist 32-Bit breit. Die Eingabe erfolgt durch eine Uhrzeitangabe, wobei die Stunden, Minuten, Sekunden jeweils durch das Zeichen “:“ getrennt sind. Die Angabe der Millisekunden erfolgt hinter der Sekundenangabe, getrennt durch einen Punkt. Diese Angabe ist nicht zwingend.
Intern enthält das Doppelwort die Anzahl der Millisekunden seit dem Zeitpunkt 0:00 Uhr.

13. Funktionen

In Kapitel 5.2 wurde die Funktion bereits kurz angesprochen. Dieses Kapitel zeigt anhand von Beispielprogrammen den Aufbau, den Aufruf und die Parameterübergabe von Funktionen.

13.1 Aufruf von Funktionen

In STEP 7 gibt es drei Befehle zum Aufruf einer Funktion:

- **CALL FCn:** Unbedingter Aufruf einer Funktion mit der Nummer n Dieser Aufruf ermöglicht die Übergabe von Parametern.
- **UC FCn:** **(Unconditional Call)** Unbedingter Aufruf einer Funktion mit der Nummer n. Die Funktion darf keine Bausteinparameter besitzen.
- **CC FCn:** **(Conditional Call)** Bedingter Aufruf einer Funktion mit der Nummer n. Die Funktion darf keine Bausteinparameter besitzen. Der Aufruf der Funktion erfolgt nur, wenn das VKE = 1 ist.

13.2 Deklaration der Variablen

Im Kopf einer Funktion stehen in der Regel die deklarierten Variablen mit ihrem Namen und dem Datentyp. Der Editor von STEP 7 sieht hierfür einen eigenen Eingabebereich im oberen Teil des Fensters vor.

Adresse	Deklaration	Name	Typ
0.0	in	FPIn	INT
2.0	out	FPOut	BYTE
4.0	in_out	FPInOut	WORD
0.0	temp	Tempo	BYTE

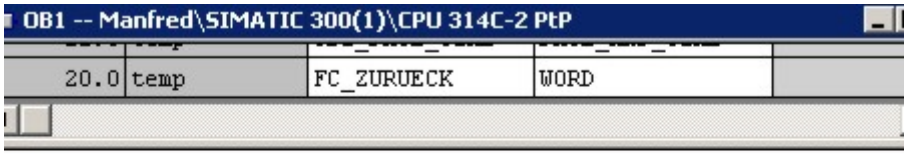
Eine Funktion kann Eingangs- (in), Ausgangs- (out), Ein- und Ausgangs- (in_out) und temporäre (temp) Variablen deklarieren. Die Deklaration kennzeichnet die Datenflussrichtung. Temporäre Variablen sind nur innerhalb der Funktion gültig.
Die Adressierung der Variablen wird automatisch vorgenommen.

13.3 Formalparameter

Alle Variablen, die Daten in die Funktion hinein oder herausgeben, nennt man Formalparameter. Im obigen Beispiel sind das die ersten drei Variablen. Werden Formalparameter deklariert, dann muss die Funktion mittels eines CALL-Befehls aufgerufen werden.

13.4 Beispiel zum Aufruf einer Funktion

Ein kleines Beispielprogramm soll zeigen, wie eine Funktion mit Formalparametern aufgerufen wird. Für das Beispiel wurde der Funktionsaufruf im OB1 programmiert. Das folgende Bild zeigt die AWL des OB1.

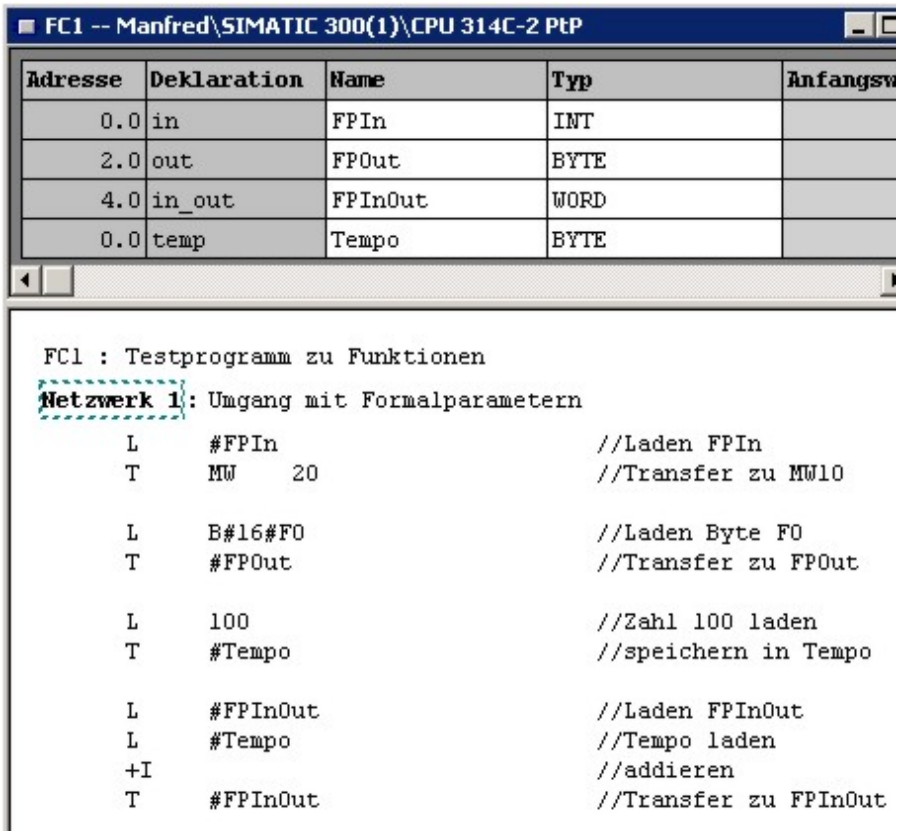



```
OB1 :  "Main Program Sweep (Cycle)"
Netzwerk 1: Aufruf des FC1 mit CALL
      L     W#16#0                    //Laden einer 0
      T     #FC_ZURUECK               //Transfer nach FC_Zurue

      CALL  FC     1
       FPIn   :=500                   //500 nach FPIn
       FPOut  :=MB0                   //FPOut nach MB0
       FPInOut:=#FC_ZURUECK           //FPInOut nach FC_Zuruec

      L     #FC_ZURUECK               //FC_Zurueck laden
      T     MW    10                  //Transfer nach MW10
```

Im Kopf des OB1 wurde eine neue temporäre Variable “FC_ZURUECK“ deklariert. Diese Variable wird im Netzwerk 1 zunächst auf 0 gesetzt. Dem Aufruf CALL FC1 folgen drei Formalparameter. Diese müssen zuvor in der FC1 deklariert worden sein. Die Zahl 500 (FPIn) wird an die Funktion übergeben. Der Rückgabewert (FPOut) wird hier dem Merkerbyte 0 zugewiesen. Mit FPInOut wird die Variable FC_Zurueck an die Funktion übergeben.

Eine Variable wird durch das #-Zeichen angezeigt. Die Funktion kann FPInOut ändern und zurückliefern. Letztlich wird FC_Zurueck noch nach MW 10 transferiert.

Die AWL der Funktion FC1 zeigt das folgende Bild.



Die Funktion enthält im Kopf die Deklaration der Formalparameter und der temporären Variablen. Die Kommentarzeilen im Netzwerk 1 erklären den Befehlsablauf. Im unteren Teil der AWL wird der Inhalt von AKKU1 und AKKU2 addiert und das Ergebnis nach FPInOut transferiert.

Hat eine Funktion keine Formalparameter, dann sieht der Aufruf in OB1 wie folgt aus:

```
      UC    FC 1          //sofortiger Sprung nach FC1
oder
      U     E 0.0         //wenn E 0.0 = 1, dann
      CC    FC 1          //springe nach FC1 (bedingter Sprung)
```

14. Funktionsbausteine

In Kapitel 5.3 wurde der Funktionsbaustein bereits kurz angesprochen. In Bezug auf den Aufruf, die Variablendeklaration und die Formalparameter gilt für den Funktionsbaustein das gleiche wie für die oben angesprochene Funktion. Das Beispiel aus Kap. 13 ist genauso mit einem Funktionsbaustein realisierbar. Dieses Kapitel zeigt daher im Wesentlichen nur die Ergänzungen zur Funktion.

14.1 Aufruf von Funktionsbausteinen

In STEP 7 gibt es drei Befehle zum Aufruf eines Funktionsbausteins:

- **CALL FBn,DBn:** Unbedingter Aufruf eines Funktionsbausteins mit der Nummer n und des zugehörigen Instanz-Datenbausteins DBn. Dieser Aufruf ermöglicht die Übergabe von Parametern.
- **UC FBn:** Unbedingter Aufruf eines Funktionsbausteins mit der Nummer n. Die Funktion darf keine Bausteinparameter besitzen.
- **CC FBn:** Bedingter Aufruf eines Funktionsbausteins mit der Nummer n. Der FB darf keine Bausteinparameter besitzen. Der Aufruf des FB erfolgt nur, wenn das VKE = 1 ist.

14.2 Deklaration der Variablen

Im Kopf eines Funktionsbausteins stehen in der Regel die deklarierten Variablen mit ihrem Namen und den Datentyp.

FB1 -- Manfred\SIMATIC 300(1)\CPU 314C-2 PtP

Adresse	Deklaration	Name	Typ	Anfangs
0.0	in	FPIn	INT	0
2.0	out	FBOut	WORD	W#16#0
4.0	in_out	FPInOut	BYTE	B#16#0
6.0	stat	Statisch	BOOL	FALSE
0.0	temp	Tempo	INT	

Im Gegensatz zur Funktion kann ein Funktionsbaustein Daten speichern. Hierfür stehen so genannte **Statische Variablen** zur Verfügung. Neu ist auch, dass bis auf die temporären Variablen alle übrigen mit einem Anfangswert versehen werden. Dies ist nur möglich, weil der FB seine Daten in einem so genannten Instanz-Datenbaustein DB1 (siehe Bild) ablegt.

14.3 Anlegen eines Instanz-Datenbausteins

Besitzt ein Funktionsbaustein Formalparameter und/oder statische Variablen, so muss im SIMATIC-Manager ein Datenbaustein erstellt werden. Hierbei ist anzugeben, dass es sich dabei um einen Instanz-Datenbaustein (DBn) für den Funktionsbaustein FBn handelt (Bild unten).

Wie man sieht, ist die Struktur des DB1 (Bild unten) bis auf die temporären Variablen gleich der Variablendeklaration im FB1.
Im DB1 können die Variablen außerdem mit einem Anfangswert versehen werden.

DB1 -- Manfred\SIMATIC 300(1)\CPU 314C-2 PtP

dresse	Deklaration	Name	Typ	Anfangs
0.0	in	FPIn	INT	0
2.0	out	FPOut	WORD	W#16#0
4.0	in_out	FPInOut	BYTE	B#16#0
6.0	stat	Statisch	BOOL	TRUE

14.4 Beispiel zum Aufruf eines Funktionsbausteins

Die folgende Bild zeigt das vollständige Beispielprogramm des Funktionsbausteins.

Zwei Eingangsgrößen des FB1 werden verglichen. Das Ergebnis (größer-gleich oder kleiner) wird in der statischen Variablen “Statisch“ zwischengespeichert. Abhängig vom Zustand der statischen Variablen wird die Ausgangsgröße auf den Wert 16 oder 8 gesetzt. SPB ist ein bedingter, und SPA ein absoluter Sprung.
BE ist der Befehl Baustein-Ende.

FB1 -- Manfred\SIMATIC 300(1)\CPU 314C-2 PtP

Adresse	Deklaration	Name	Typ	Anfangswe
0.0	in	FPIn	INT	0
2.0	out	FPOut	WORD	W#16#0
4.0	in_out	FPInOut	BYTE	B#16#0
6.0	stat	Statisch	BOOL	TRUE
0.0	temp	Tempo	BOOL	

```
FB1 : Beispielprogramm Funktionsbaustein
Netzwerk 1: Titel:
      L     #FPIn                     //FPIn in AKKU1
      L     #FPInOut                  //FPInOut in AKKU1,
//FPIn in AKKU2
      >=I                             //AKKU1 >= AKKU2 ?
      =     #Statisch                 //VKE nach #Statisch

      U     #Statisch                 //wenn Statisch, dann
      SPB   L_1                       //Springe nach L_1

      L     W#16#F                    //sonst lade Zahl 16
      T     #FPOut                    //Transfer nach FPOut
      SPA   L_2                       //springe nach L_2

L_1:  L     W#16#8                    //lade die Zahl 8
      T     #FPOut                    //Transfer nach FPOut

L_2:  BE                              //Baustein ende
```


Der zugehörige Aufruf des Funktionsbausteins erfolgt in unserem Beispiel wieder aus dem OB1. Das folgende Bild zeigt die AWL des OB1.

OB1 -- Manfred\SIMATIC 300(1)\CPU 314C-2 PtP

20.0	temp	FBInOut	BYTE

```
OB1 :  "Main Program Sweep (Cycle)"
Netzwerk 1: Aufruf des FB1 mit CALL
      L     B#16#A0
      T     #FBInOut

      CALL  FB     1 , DB1
       FPIn   :=EW0
       FPOut  :=MW0
       FPInOut:=#FBInOut

      AUF   DI     1
      L     DIB    6
      T     AB     1
```

Die schon im letzen Beispiel deklarierte Variable “FBInOut“ wird mit dem Wert A0 (Dez 160) vorbesetzt. Dann erfolgt der Aufruf des FB1 mit dem zugehörigen Instanz-DB1. Es ist zu beachten, dass der FB1 und der DB1 vor dem Aufruf erstellt werden müssen. An den FB1 werden das Eingangswort EW0 und die Variable FBInOut übergeben. Der Rückgabewert FPOut wird im Merkerwort 0 abgelegt.

Der Zugriff auf einen Instanz-DB1 ist auch außerhalb des FB1 möglich. Hierzu muss dieser zunächst mit dem Befehl “AUF DI“ aufgeschlagen werden. Dann kann das Instanz-Datenbyte (DIB) 6 gelesen werden. Dieses Byte enthält die statische Variable “Statisch“.

15. Zähler

Zum Beispiel dürfen die Besucher einer Veranstaltung das Gebäude aus Sicherheitsgründen erst dann betreten, wenn die maximale Anzahl von Besuchern nicht überschritten wurde. Diese Aufgabe ist eine klassische Zählaufgabe, für die in STEP 7 fertige Zählfunktionen vorhanden sind.
Es ist bei dieser Funktion zu beachten, dass nur die ansteigende Flanke eines Impulses gezählt wird, also nur der Signalwechsel von 0 nach 1.

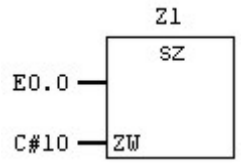
15.1 Zähler setzen und Rücksetzen

Ein Zähler wird gesetzt, sobald das VKE am Setzeingang von 0 nach 1 wechselt. Durch das Setzen ist es möglich, einen Zähler mit einem Wert vorzubelegen. Dabei wird zuerst der Zählwert "C#010" in den AKKU geladen und danach wird der Setzbefehl ausgeführt.

AWL-Beispiel:

```
U    E 0.0     // wenn E 0.0 von 0 auf 1, dann
L    C#010     //Lade konstanten Zählwert 10
S    Z 1       //Setze Zähler 1 auf Anfangswert 10
```

FUP-Beispiel:



Ein Zähler wird rückgesetzt, wenn das VKE am Rücksetzeingang des Zählers den Zustand 1 hat.
Zum Rücksetzen ist kein Flankenwechsel am Rücksetzeingang notwendig.

AWL-Beispiel:

```
U    E 0.1     // wenn E 0.1 = 1, dann
R    Z 1       //Rücksetzen Zähler 1
```


15.2 Zähler abfragen

Der Zählerstand kann entweder absolut oder binär abgefragt werden. Möchte man den aktuellen Zählerstand im Programm auswerten, so ist dies mit den Befehlen LC (Lade codiert, BCD) oder L (Lade, dualcodiert) möglich.

AWL-Beispiel:

```
L    Z 1       // dualcodiert laden
T    MW 10     // Transfer ins Merkerwort 10
LC   Z 1       // BCD-codiert laden
T    MW 12     // Transfer ins Merkerwort 12
```

Bei einer binären Abfrage liefert das Ergebnis einer UND-Verknüpfung den Wert 1, solange der Zählerstand **größer NULL** ist.

AWL-Beispiel:

```
U    Z 1       // wenn Zählerstand von Z 1 ungleich 0, dann
=    M 0.0     //Merkerbit 0.0 = 1
```

15.3 Zählwert laden

Ein Zähler **kann** mit einem Zählwert vorbelegt werden. Den zu ladenden Wert übernimmt der Zähler nur bei einer positiven Flanke am Setzeingang.

AWL-Beispiel:

```
L    C# 100    // Laden eines konstanten Zählwertes (0 – 999)

L    DBW 10    // Laden eines Datenwortes (BCD-codiert)
L    EW 0      // Laden eines Eingangswortes (BCD-codiert)
L    AW 2      // Laden eines Ausgangswortes (BCD-codiert)
L    MW 4      // Laden eines Merkerwortes (BCD-codiert)
L    LW 6      // Laden eines Lokaldatenwortes (BCD-codiert)
```

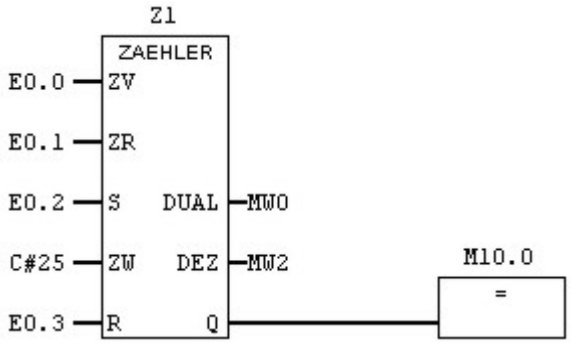
15.4 Vorwärtszähler / Rückwärtszähler

Bei einem Vorwärtszähler wird der Wert des Zählers mit jeder positiven Flanke des VKE am Eingang ZV um eins erhöht.
Ist der maximale Zählerstand von 999 erreicht, so wird er nicht weiter erhöht.
Eine binäre Abfrage des Zählers liefert den Wert 1, sobald der Zählerstand von Null verschieden ist.

AWL-Beispiel:

```
U     E 0.0      // wechselt E 0.0 von 0 auf 1, dann
ZV    Z 1        // Zählerstand + 1
U     E 0.1      // wechselt E 0.1 von 0 auf 1, dann
ZR    Z 1        // Zählerstand – 1
U     E 0.2      // wenn E 0.2 = 1, dann
S     Z 1        // Zähler mit Zählwert 25 vorbesetzen
U     E 0.3      // wenn E 0.3 = 1, dann
R     Z 1        // Zähler rücksetzen
L     Z 1        // Zählerstand laden
T     MW 0       // Transfer nach MW 0
LC    Z 1        // Zählerstand laden (BCD-codiert)
T     MW 2       // Transfer nach MW 2
U     Z 1        // binäre Abfrage Zähler 1
=     M 10.0     // M 10.0 = 1 wenn Z1 != 0
```

FUP-Beispiel:



Das obige Beispiel zeigt noch einmal alle Funktionen, die in Verbindung mit einem Zähler möglich sind. In der Darstellung FUP sind auch Teilfunktionen (wie in Kap.15.1) möglich.

16. Zeiten

Bei vielen Steuerungsaufgaben müssen zeitgesteuerte Vorgänge irgendwelcher Art eingebaut werden.
Die Programmiersprache STEP 7 stellt fünf verschiedene Zeittypen zur Verfügung:

1. Der Impuls SI
2. Der verlängerte Impuls SV
3. Die Einschaltverzögerung SE
4. Die speichernde Verzögerung SS
5. Die Ausschaltverzögerung SA

16.1 Zeitfunktion mit Zeitwert laden

Die Zeitfunktion wird durch einen Lade-Befehl mit einem Anfangswert belegt. Das Betriebssystem zählt diesen Anfangswert in einem bestimmten Zeitintervall bis auf Null zurück. Damit ist die Zeit abgelaufen. Dieser Anfangszeitwert muss beim Start der Zeit im AKKU1 vorhanden sein.

Der Zeitwert wird wie folgt in den AKKU 1 geladen:

AWL-Beispiel:

```
L     S5T#5S     // Laden eines konstanten Zeitwertes (5Sek.)
                 // Laden eines Zeitwertes in Form:
L     DBW 10     // eines Datenwortes (BCD-codiert)
L     EW 0       // eines Eingangswortes (BCD-codiert)
L     AW 2       // eines Ausgangswortes (BCD-codiert)
L     MW 4       // eines Merkerwortes (BCD-codiert)
L     LW 6       // eines Lokaldatenwortes (BCD-codiert)
```

Die Struktur des konstanten Zeitwertes wurde in Kap.12.2 ausführlich angegeben.

16.2 Starten und Rücksetzen einer Zeit

Zum Rücksetzen einer Zeit muss das VKE am Rücksetzeingang den Zustand 1 haben. Ist dies der Fall, so wird der programmierte Zeitwert auf 0 gesetzt. Solange das VKE am Rücksetzeingang den Zustand 1 behält, liefert eine binäre Abfrage des Zeitgliedes den Zustand 0.

Anders als beim Starteingang, ist beim Rücksetzeingang kein Flankenwechsel des VKEs notwendig, damit die Aktion ausgeführt wird.

16.3 Abfragen einer Zeit

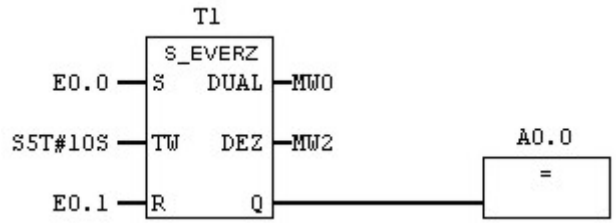
Eine Zeit kann über binäre Operationen auf ihren Zustand abgefragt werden. Es ist somit möglich, eine Zeit abzufragen und das Ergebnis in andere binäre Verknüpfungen mit einzubinden.

Der absolute Zeitwert des Zeitgliedes kann über die Operationen LC (Lade codiert) und L (Lade dualcodiert) in den AKKU1 geladen werden. Damit ist es möglich, den Wert im SPS-Programm weiter zu verarbeiten.

AWL-Beispiel:

```
U     E 0.0      // wechselt E 0.0 von 0 auf 1, dann
L     S5T#10S    // erst Zeitwert 10s in AKKU 1
SE    T 1        // Timer 1 starten als Einschaltverzögerung
U     E 0.1      // wenn E 0.1 = 1, dann
R     T 1        // Timer rücksetzen
L     T 1        // Zeitwert dualcodiert laden
T     MW 0       //Transfer nach Merkerwort 0
LC    T 1        // Zeitwert BCD-codiert laden
T     MW 2       // Transfer nach Merkerwort 2
U     T 1        // Ist die Zeit abgelaufen und E0.0 = 1, dann
=     A 0.0      // setze A0.0 auf 1
```

FUP-Beispiel:

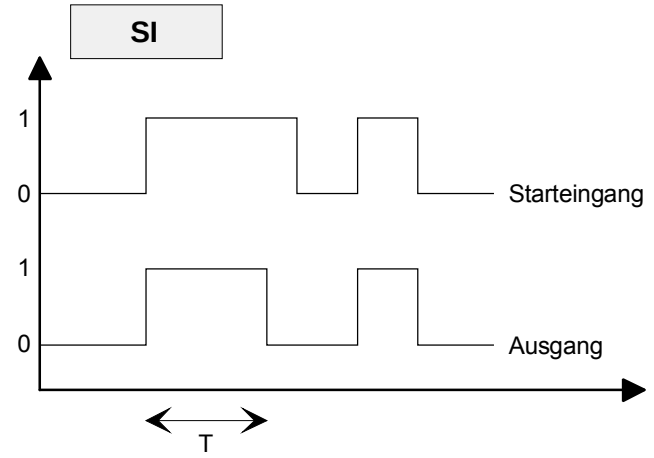


Im Folgenden werden nun die unterschiedlichen Zeitarten vorgestellt und anhand von Beispielen erläutert.

16.4 Die Zeitart SI (Impuls)

Mit der Zeitart SI kann ein Impuls definierter Länge erzeugt werden. Wechselt der Starteingang seinen Zustand von 0 nach 1, dann liefert die binäre Abfrage der Zeit den Zustand 1. Wechselt der Starteingang vor Ablauf der Zeit nach 0, dann liefert jedoch auch die Abfrage des Ausgangs 0.

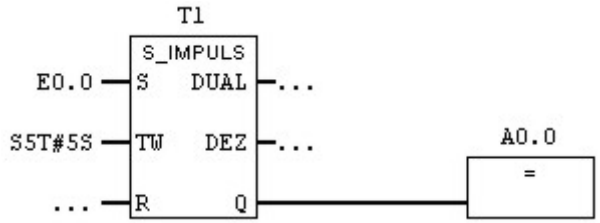
Zeitdiagramm:



AWL-Beispiel:

```
U     E 0.0      // Starteingang
L     S5T#5S     // Zeitwert T = 5s
SI    T 1        // SI-Zeit starten
U     T 1
=     A 0.0      //Ausgang
```

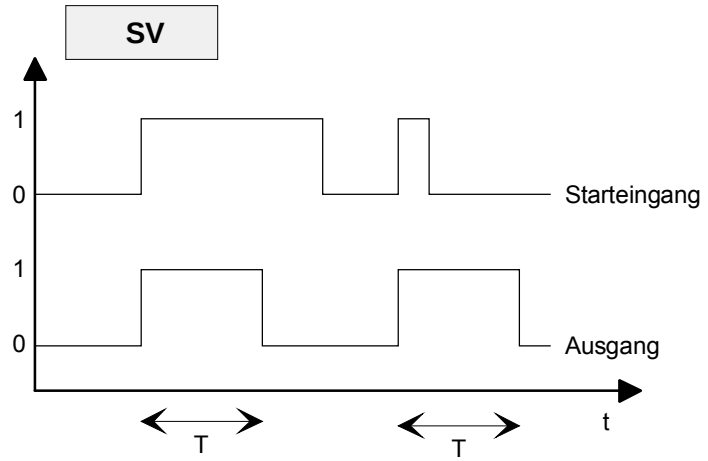
FUP-Beispiel:



16.5 Die Zeitart SV (verlängerter Impuls)

Die Zeitart SV funktioniert im Prinzip wie SI. Der Ausgang bleibt jedoch auf 1 auch wenn der Zustand des Starteingangs vor Ablauf der Zeit auf 0 wechselt.

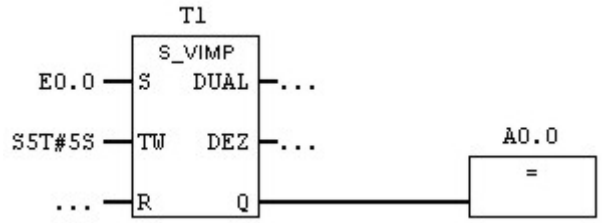
Zeitdiagramm:



AWL-Beispiel:

```
U     E 0.0      // Starteingang
L     S5T#5S     // Zeitwert T = 5s
SV    T 1        // SV-Zeit starten
U     T 1
=     A 0.0      //Ausgang
```

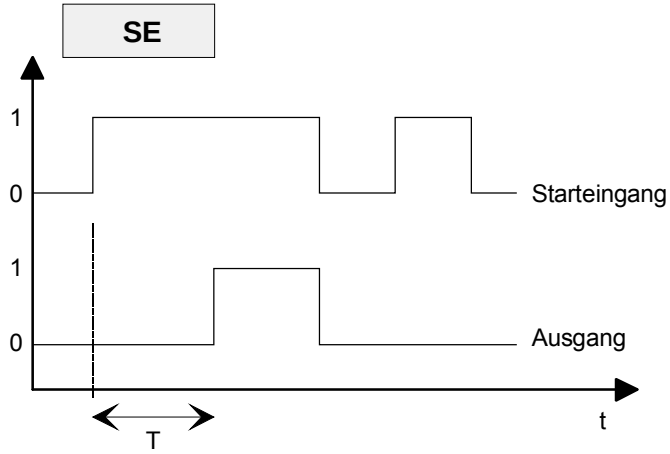
FUP-Beispiel:



16.6 Die Zeitart SE (Einschaltverzögerung)

Mit SE kann eine Einschaltverzögerung realisiert werden. Mit dem Wechsel am Starteingang von 0 auf 1 läuft die geladene Zeit ab. Erst nach Ablauf (verzögert) liefert die binäre Abfrage des Ausgangs den Zustand 1.

Zeitdiagramm:

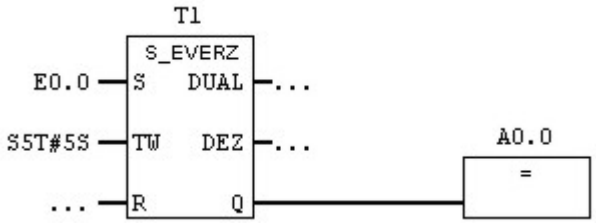


Aus dem Zeitdiagramm ist zu ersehen, dass der Ausgang nicht auf 1 geht, wenn der Starteingang vorzeitig auf 0 gesetzt wird.

AWL-Beispiel:

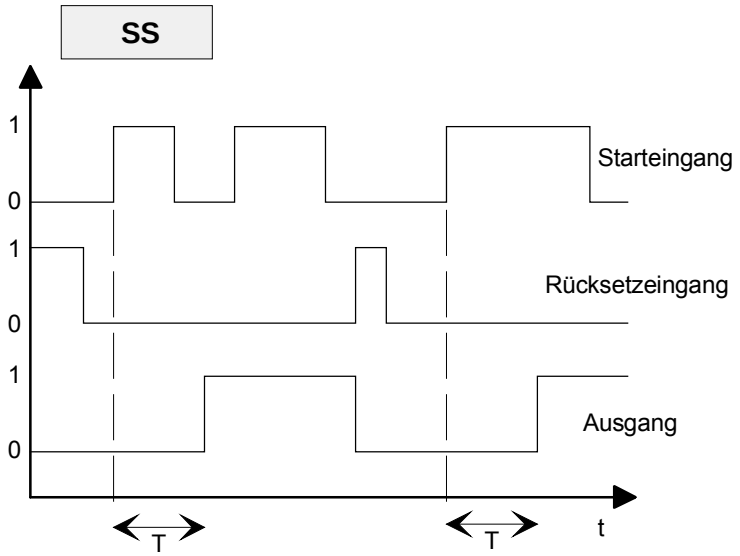
```
U     E 0.0      // Starteingang
L     S5T#5S     // Zeitwert T = 5s
SE    T 1        // SE-Zeit starten
U     T 1
=     A 0.0      //Ausgang
```

FUP-Beispiel:



16.7 Die Zeitart SS (Speichernde Einschaltverzögerung)

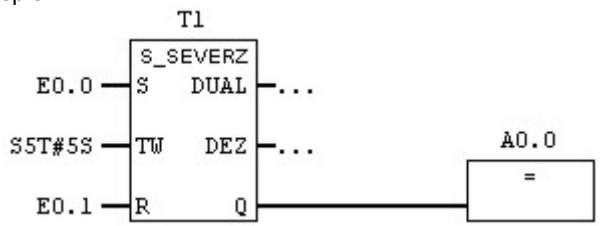
Die Zeitart SS funktioniert im Prinzip wie SE. Der Ausgang geht jedoch auch dann für die Dauer der geladenen Zeit auf 1, wenn der Zustand des Starteingangs vor Beginn bzw. Ablauf der Zeit auf 0 wechselt. **Diese Zeitart muss somit explizit rückgesetzt werden (Zeitdiagramm).**



AWL-Beispiel:

```
U     E 0.0      // Starteingang
L     S5T#5S     // Zeitwert T = 5s
SS    T 1        // SS-Zeit starten
U     E 0.1
R     T 1        // Zeit rücksetzen
U     T 1
=     A 0.0      //Ausgang
```

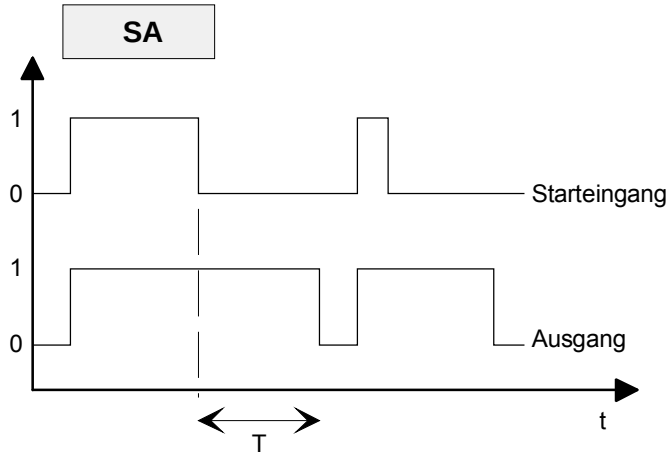
FUP-Beispiel:



16.8 Die Zeitart SA (Ausschaltverzögerung)

Mit der Zeitart SA wird eine Ausschaltverzögerung realisiert. Wechselt dar Zustand am Starteingang von 1 auf 0 (ausschalten) dann wird die Zeit gestartet. Nach Ablauf der Zeit, wechselt auch der Ausgang seinen Zustand nach 0.
Eine binäre Abfrage der Zeit liefert den Zustand 1, solange das VKE am Starteingang den Zustand 1 hat oder die Zeit läuft.

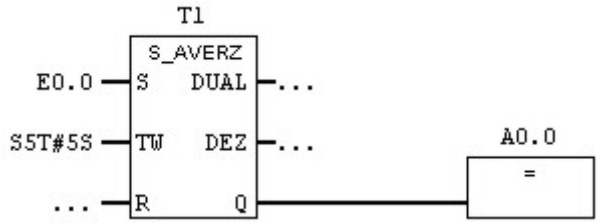
Zeitdiagramm:



AWL-Beispiel:

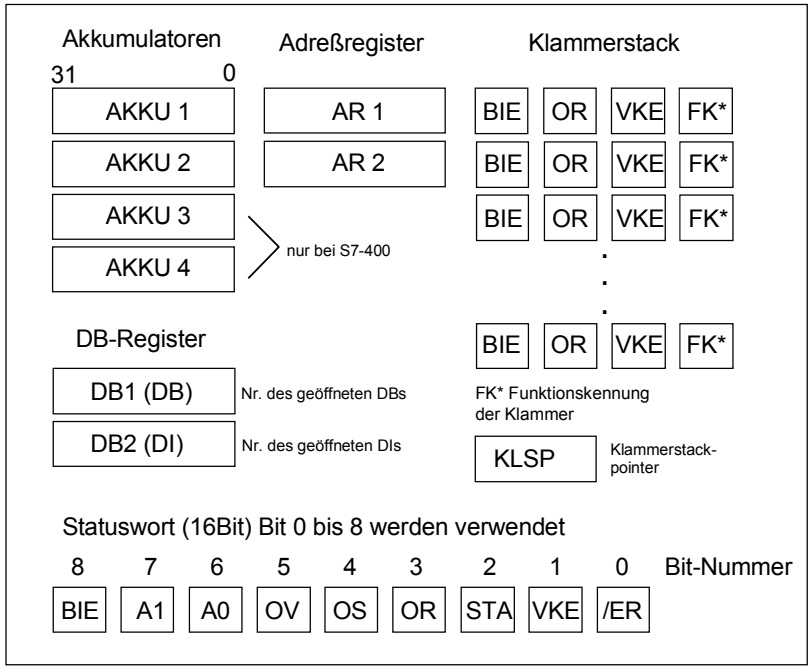
```
U     E 0.0      // Starteingang
L     S5T#5S     // Zeitwert T = 5s
SA    T 1        // SA-Zeit starten
U     T 1
=     A 0.0      //Ausgang
```

FUP-Beispiel:



17. Die Register der CPU

Die Register einer CPU sind interne Speicher, die zur Abarbeitung des SPS-Programms benötigt werden. Das folgende Bild zeigt die Registerstruktur einer SIMATIC S7.



17.1 Akkumulatoren

Die Akkumulatoren werden von nahezu allen Befehlen des Programms verwendet. Lade-, Transfer-, Vergleichs- und Rechenoperationen nutzen alle diese Register. Unsere S7-300 besitzt 2 Akkumulatoren.

17.2 Adressregister

Die Adressregister werden bei der indirekten Adressierung (Pointer) verwendet.

17.3 DB-Register

In den DB-Registern wird der geöffnete Global-Datenbaustein (z.B. DB1) bzw. der geöffnete Instanzdatenbaustein (z.B. DB2) gespeichert werden (es können max. 2 Datenbausteine geöffnet sein).

17.4 Das Statuswort

Das Statuswort besteht aus einem Word, wobei nur die Bits 0 bis 8 verwendet werden. Die einzelnen Bits haben folgende Bedeutung:

Bit-Nr.	Bedeu-tung	Beschreibung
0	/ER	Erstverknüpfung (0 bedeutet, nächste Verknüpfung ist Erstverknüpfung
1	VKE	Verknüpfungsergebnis In diesem Bit wird das Ergebnis einer Verknüpfungsoperation gespeichert
2	STA	Status-Bit In diesem Bit wird der Zustand des zuletzt verwendeten Bitoperanden in einer Verknüpfungsoperation gespeichert.
3	OR	ODER-Flag Dieses Bit wird verwendet, wenn UND-Blöcke mit dem Befehl “O“ verknüpft werden. Es wird das Ergebnis der UND-Verknüpfung gespeichert.
4	OS	Overflow-Speichernd Dieses Bit wird mit dem Bit OV gesetzt. Es wird bei der nächsten Operation nicht auf ’0’ gesetzt – ist also speichernd. Das OD-Bit wird nur durch den Sprungbefehl “SPS“ zurückgesetzt.
5	OV	Overflow (Überlauf) Dieses Bit zeigt einen Fehler bei einer arithmetischen Operation an.
6	A0	Die Anzeige-Bits A0 und A1 informieren über folgende Ergebnisse: - Ergebnis einer Vergleichsoperation - Ergebnis einer Wortverknüpfung - Ergebnisse von Schiebeoperationen - Ergebnisse bei arithmetischen Operationen
7	A1	
8	BIE	Binärergebnis Mit Hilfe des Bits BIE kann das VKE zwischengespeichert und restauriert werden.

18. Übersicht zur Programmabarbeitung

18.1 Die Betriebszustände der SIMATIC-S7

Eine S7-SPS kennt folgende Betriebszustände:

1. RUN-Betrieb

   Die SPU bearbeitet das Anwenderprogramm.

2. STOP-Betrieb

   Die CPU bearbeitet kein Anwenderprogramm

3. ANLAUF-Betrieb

   Bevor die CPU nach dem Einschalten mit der Bearbeitung des Anwenderprogramms beginnt, wird ein Anlaufprogramm bearbeitet. Im Anlaufprogramm können Sie durch entsprechende Programmierung von Anlauf-OBs bestimmte Voreinstellungen für Ihr zyklisches Programm festlegen.

   - Neustart (Warmstart, OB100)
     z.B. beim Einschalten der Netzspannung

   - Wiederanlauf (OB101)
     z.B. nach einer Unterbrechung der Netzspannung im laufenden Betrieb

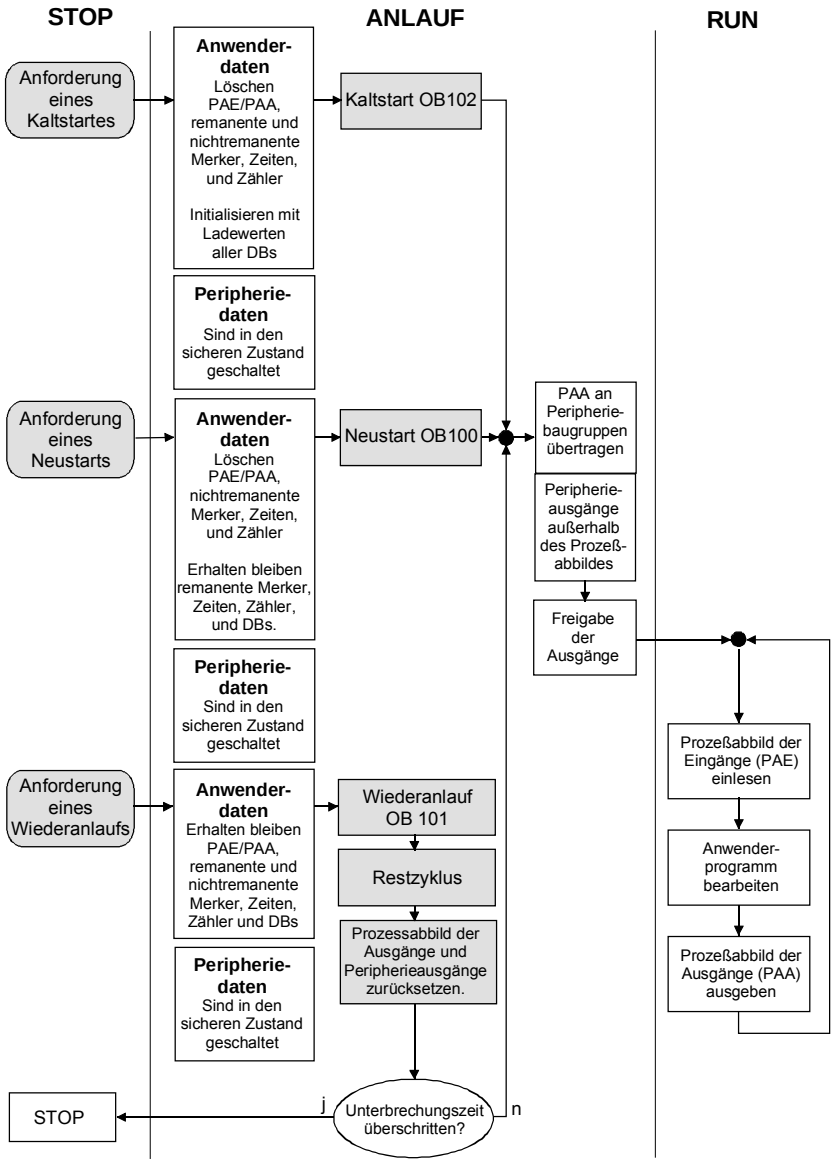
   - Kaltstart (OB102)

4. HALT-Betrieb

   Testbetrieb, wenn die CPU den Programmfluss aufgrund eines Breakpoints unterbricht.

18.2 Ablauf einer zyklischen Programmbearbeitung

Der OB1 wird vom Betriebssystem zyklisch aufgerufen. Neben der Abarbeitung des Anwenderprogramms erledigt die SPS weitere Aufgaben wie. z.B. das Schreiben und Lesen der Peripherie, das Reagieren auf Ereignisse und das Laden und Löschen von Bausteinen. Die Zyklusdauer kann daher unterschiedlich lang sein. Das folgende Bild zeigt zusammenfassend die Aktivitäten der SPS.



Tätigkeiten der CPU in den Betriebszuständen ANLAUF und RUN.

19. STEP 7 Befehlsübersicht

Die nachfolgend angegebene Befehlsübersicht ist keine vollständige Abhandlung des Befehlscodes, sondern lediglich eine Übersicht der STEP 7 Befehle. Informationen über die Auswirkung auf das Statuswort und die anzuwendenden Operanden müssen in der vollständigen Befehlsfibel oder der Online-Hilfe nachgeschlagen werden.

Verknüpfungsoperationen	
Operation	**Beschreibung**
U / UN	UND / UND-NICHT
O / ON	ODER / ODER-NICHT
X / XN	EXKLUSIV-ODER / (EXKLUSIV-ODER-NICHT)
Klammeroperationen	
U(	UND-Klammer-Auf
UN(	UND-NICHT-Klammer-Auf
O(	ODER-Klammer-Auf
ON(	ODER-NICHT-Klammer-Auf
X(	EXKLUSIV-ODER-Klammer-Aus
XN(	EXKLUSIV-ODER-NICHT-Klammer-Aus
)	Klammer-Zu
O	ODER-Verknüpfung von UND-Funktionen nach UND-vor-ODER (ohne Operand)
Bitoperationen mit Timern und Zählern	
U / UN	UND / UND-NICHT
O / ON	ODER / ODER-NICHT
X / XN	EXKLUSIV-ODER / (EXKLUSIV-ODER-NICHT)
Wort- und Doppelwortverknüpfungen	
UW	UND AKKU2-L bzw. 16Bit-Konstante
OW	ODER AKKU2-L bzw. 16Bit-Konstante
XOW	EXKLUSIV-ODER AKKU2-L bzw. 16Bit-Konstante
UD	UND AKKU2 bzw. 32Bit-Konstante
OD	ODER AKKU2 bzw. 32Bit-Konstante
XOD	EXKLUSIV-ODER AKKU2 bzw. 32Bit-Konstante
Flankenoperation	
FP / FN	Anzeigen der steigenden / fallenden Flanke mit VKE=1. Flankenhilfsmerker ist der in der Operation adressierte Bitoperand
Speicheroperationen	
S	Setze adressiertes Bit auf 1

R	Setze adressiertes Bit auf 0
=	Zuweisen des VKE
VKE-Operation	
CLR	Setze VKE auf 0
SET	Setze VKE auf 1
NOT	Negiere das VKE
Zeitoperationen	
SI	Starte Timer als Impuls bei Flankenwechsel von 0 nach 1
SV	Starte Timer als verlängerten Impuls bei Flankenwechsel von 0 nach 1
SE	Starte Timer als Einschaltverzögerung bei Flankenwechsel von 0 nach 1
SS	Starte Timer als speichernde Einschaltverzögerung bei Flankenwechsel von 0 nach 1
SA	Starte Timer als Ausschaltverzögerung bei Flankenwechsel von 0 nach 1
FR	Freigabe eines Timers für das erneute Starten bei Flankenwechsel von 0 nach 1
R	Rücksetzen einer Zeit
Zähleroperationen	
S	Vorbelegen eines Zählers bei Flankenwechsel von 0 nach 1
R	Rücksetzen des Zählers auf 0 bei VKE=1
ZV	Zähle um 1 vorwärts bei Flankenwechsel von 0 nach 1
ZR	Zähle um 1 rückwärts bei Flankenwechsel von 0 nach 1
FR	Freigabe eines Zählers bei Flankenwechsel von 0 nach 1
Ladeoperationen	
L	Lade (Konstante, Byte, Wort, Doppelwort, …) dualcodiert
LC	Lade (Konstante, Byte, Wort, Doppelwort, …) BCD-codiert
Transferoperationen	
T	Transferiere Inhalt von AKKU 1 …
Zugriffsoperationen auf Adressregister	

LAR1	Lade Inhalt aus AKKU1 … in AR1
LAR2	Lade Inhalt aus AKKU1 … in AR2
TAR1	Transferiere Inhalt aus AR1 in AKKU1, …
TAR2	Transferiere Inhalt aus AR2 in AKKU2, …
TAR	Tausche den Inhalt von AR1 und AR2
Zugriffsoperation auf das Statuswort	
L STW	Lade Statuswort in AKKU 1
T STW	Transferiere AKKU1 (Bits 0-8) ins Statuswort
Datenbausteinoperationen	
L DBNO	Lade Nummer des Datenbausteins
L DINO	Lade Nummer des Instanz-Datenbausteins
L DBLG	Lade Länge des Datenbausteins in Byte
L DILG	Lade Länge des Instanz-Datenbausteins in Byte
TDB	Tausche Datenbausteine
Mathematische Operationen	
+I	Addiere zwei Integerzahlen (16Bit) AKKU1-L = AKKU1-L + AKKU2-L
-I	Subtrahiere zwei Integerzahlen (16Bit) AKKU1-L = AKKU2-L - AKKU1-L
*I	Multipliziere zwei Integerzahlen (16Bit) AKKU1 = AKKU2-L * AKKU1-L
/I	Dividiere zwei Integerzahlen (16Bit) AKKU1-L = AKKU2-L / AKKU1-L In AKKU1-H steht der Rest der Division
+D	Addiere zwei Integerzahlen (32Bit) AKKU1 = AKKU1 + AKKU2
-D	Subtrahiere zwei Integerzahlen (32Bit) AKKU1 = AKKU2 - AKKU1
*D	Multipliziere zwei Integerzahlen (32Bit) AKKU1 = AKKU2 * AKKU1
/D	Dividiere zwei Integerzahlen (32Bit) AKKU1 = AKKU2 / AKKU1
MOD	Dividiere zwei Integerzahlen (32Bit) und lade den Rest der Division in AKKU1
+R	Addiere zwei Realzahlen (32Bit) AKKU1 = AKKU1 + AKKU2
-R	Subtrahiere zwei Realzahlen (32Bit) AKKU1 = AKKU2 - AKKU1
*D	Multipliziere zwei Realzahlen (32Bit) AKKU1 = AKKU2 * AKKU1

/D	Dividiere zwei Realzahlen (32Bit) AKKU1 = AKKU2 / AKKU1
NEGR	Negiere Realzahl in AKKU1
ABS	Bilde Betrag der Realzahl in AKKU1
SQRT	Berechne die Quadratwurzel einer Realzahl in AKKU1
SQR	Quadriere die Realzahl in AKKU1
LN	Bilde den natürlichen Logarithmus einer Realzahl in AKKU1
EXP	Berechne den Exponentialwert einer Realzahl in AKKU1 zur Basis e
SIN	Berechne den Sinus einer Realzahl
ASIN	Berechne den Arcussinus einer Realzahl
COS	Berechne den Cosinus einer Realzahl
ACOS	Berechne den Arcuscosinus einer Realzahl
TAN	Berechne den Tangens einer Realzahl
ATAN	Berechne den Arcustangens einer Realzahl
+ i16	Addiere eine 16Bit Integer-Konstante
+ i32	Addiere eine 32Bit Integer-Konstante
Vergleichsoperationen mit INT, DINT, REAL	
==I	AKKU2-L = AKKU1-L
<>I	AKKU2-L != AKKU1-L
<I	AKKU2-L < AKKU1-L
<=I	AKKU2-L <= AKKU1-L
>I	AKKU2-L > AKKU1-L
>=I	AKKU2-L >= AKKU1-L
==D	AKKU2 = AKKU1
<>D	AKKU2 != AKKU1
<D	AKKU2 < AKKU1
<=D	AKKU2 <= AKKU1
>D	AKKU2 > AKKU1
>=D	AKKU2 >= AKKU1
==R	AKKU2 = AKKU1
<>R	AKKU2 != AKKU1
<R	AKKU2 < AKKU1
<=R	AKKU2 <= AKKU1
>R	AKKU2 > AKKU1
>=R	AKKU2 >= AKKU1
Schiebebefehle	

SLW	Schiebe Inhalt von AKKU1-L nach links
SLD	Schiebe Inhalt von AKKU1 nach links
SRW	Schiebe Inhalt von AKKU1-L nach rechts
SRD	Schiebe Inhalt von AKKU1 nach rechts
SSI	Schiebe Inhalt von AKKU1-L mit Vorzeichen nach rechts, Freiwerdende Stellen werden mit dem Vorzeichen (Bit15) aufgefüllt.
SSD	Schiebe Inhalt von AKKU1 mit Vorzeichen nach rechts, Freiwerdende Stellen werden mit dem Vorzeichen (Bit31) aufgefüllt.
Rotierbefehle	
RLD	Rotiere Inhalt von AKKU1 nach links
RRD	Rotiere Inhalt von AKKU1 nach rechts
RLDA	Rotiere Inhalt von AKKU1 um eine Bitposition nach links über Anzeigenbit A1
RRDA	Rotiere Inhalt von AKKU1 um eine Bitposition nach rechts über Anzeigenbit A1
Akku-Befehle	
TAW	Umkehr der Reihenfolge der Bytes im AKKU1-L
TAD	Umkehr der Reihenfolge der Bytes im AKKU1
TAK	Tausche Inhalt von AKKU1 und AKKU2
ENT	Inhalt von AKKU2 und 3 wird nach AKKU3 und 4 übertragen
LEAVE	Inhalt von AKKU3 und 4 wird nach AKKU2 und 3 übertragen
PUSH	Inhalt von AKKU1, 2 und 3 wird nach AKKU2, 3 und 4 übertragen
POP	Inhalt von AKKU2, 3 und 4 wird nach AKKU1, 2 und 3 übertragen
INC k8	Inkrementiere AKKU1-LL
DEC k8	Dekrementiere AKKU1-LL
Bild- und Null-Befehle	
BLD k8	Bildaufbauoperation; wird von der CPU wie eine Nulloperation behandelt
NOP 0/1	Nulloperation
Typenkonvertierungen	
BTI	Konvertiere AKKU1-L von BCD (0 bis +/-999) in Integerzahl (16Bit) BCD-to-Int
BTD	Konvertiere AKKU1 von BCD (0 bis +/-9 999 999) in Double-Integerzahl (32Bit) BCD-to-DoubleInt
DTR	Konvertiere AKKU1 von DoubleInt (32Bit) nach Real
ITD	Konvertiere AKKU1 von Integer (16Bit) in DoubleInt (32Bit)
ITB	Konvertiere AKKU1-L von Integer (16Bit) nach BCD
DTB	Konvertiere AKKU1 von Double-Integer (32Bit) nach BCD (0 bis +/- 9 999 999)
RND	Wandle Realzahl in 32Bit-Integerzahl um
RND-	Wandle Realzahl in 32Bit-Integerzahl um. Es wird abgerundet zur nächsten ganzen Zahl.
RND+	Wandle Realzahl in 32Bit-Integerzahl um. Es wird aufgerundet zur nächsten ganzen Zahl.
TRUNC	Wandle Realzahl in 32Bit-Integerzahl um. Es werden die Nachkommastellen abgeschnitten.
Komplementoperationen	
INVI	Bilde 1er-Komplement von AKKU1-L
INVD	Bilde 1er-Komplement von AKKU1
NEGI	Bilde 2er-Komplement von AKKU1-(Int))
NEGD	Bilde 2er-Komplement von AKKU1 (DoubleInt)
Bausteinaufrufe FC, FB, DB	
CALL	Unbedingter Aufruf eines (FB, SFB, FC, SFC) mit Parameterübergabe
UC	Unbedingter Aufruf eines (FB, SFB, FC, SFC) ohne Parameterübergabe
CC	Bedingter Aufruf eines (FB, SFB, FC, SFC) mit Parameterübergabe
AUF	Aufschlagen eines DB, DI
Baustein-Endeoperationen	
BE	Beenden Baustein
BEA	Beenden Baustein absolut
BEB	Beenden Baustein bedingt, wenn VKE=1
Sprungbefehle	
SPA	Springe unbedingt
SPB	Springe bedingt, bei VKE=1
SPBN	Springe bedingt, bei VKE=0
SPBB	Springe bei VKE=1, VKE in das BIE-Bit retten
SPBNB	Springe bei VKE=0, VKE in das BIE-Bit retten
SPBI	Springe bei BIE-Bit=1
SPBIN	Springe bei BIE-Bit=0
SPO	Springe bei Überlauf speichernd (OV=1)

SPS	Springe bei Überlauf speichernd (OS=1)
SPU	Springe bei unzulässiger Arithmetikoperation (A1=1 und A0=1)
SPZ	Springe bei Ergebnis = 0 (A1=0 und A0=0)
SPP	Springe bei Ergebnis > 0 (A1=1 und A0=0)
SPM	Springe bei Ergebnis < 0 (A1=0 und A0=1)
SPN	Springe bei Ergebnis != 0 (A1=1 und A0=0) oder (A1=0 und A0=1)
SPMZ	Springe bei Ergebnis <= 0 (A1=1 und A0=1) oder (A1=0 und A0=0)
SPPZ	Springe bei Ergebnis >= 0 (A1=1 und A0=1) oder (A1=0 und A0=0)
SPL	Sprungverteiler Der Operation folgt eine Liste von Sprungoperationen. Der Operand ist eine Sprungmarke auf die der Liste folgenden Operation.
LOOP	Dekrementiere AKK1-L und springe bei AKKU1-L !=0 (Schleifenprogrammierung)