

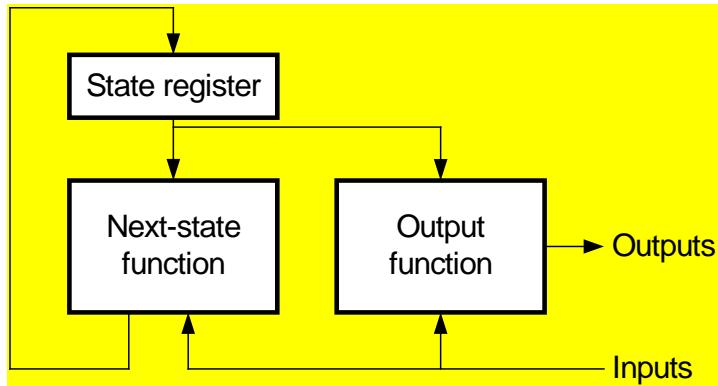
2. ZIELARCHITEKTUREN UND PLATTFORMEN

2.1 Digitale Systemkomponenten

Unterschiedliche Systemkomponenten, je nach Einsatzgebiet und den resultierenden Anforderungen an Rechenleistung. Oft recht heterogene Systemarchitekturen aus verschiedenen kooperierenden Komponenten.

2.1.1 Spezialhardware

Schaltwerke/Endliche Automaten (FSMs = Finite State Machines)



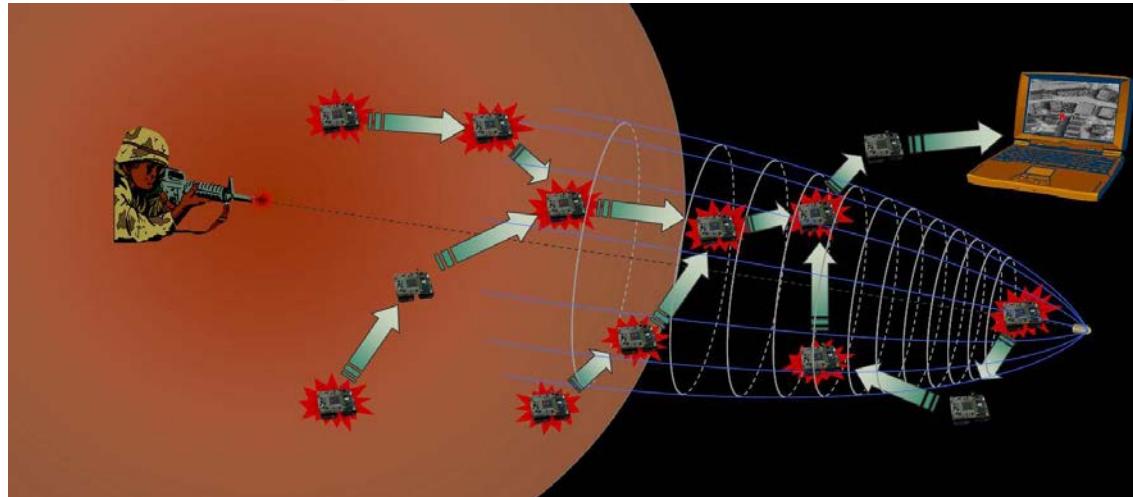
- Zustand des Automaten im Zustandsregister (State reg.)
- Folgezustand durch Übergangsfunktion (Next-state func.)
- Ausgabefunktion (Output function) in Abhängigkeit des Zustands und der Eingabe (Mealy-Automat) oder nur des Zustands (Moore-Automat)

Realisierung meist als FPGA oder ASIC. Nur für einfache diskrete Steuerungsaufgaben geeignet (Zustandsraumexplosion).

Schnell, aber inflexibel!

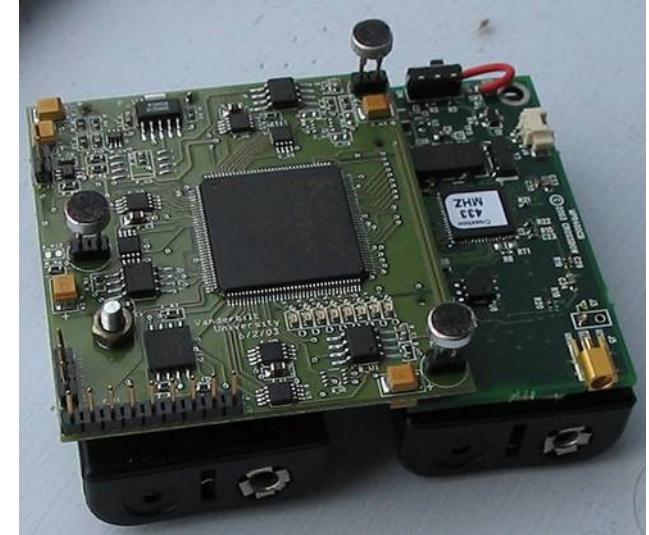
Beispiel: Lokalisierung von Schüssen mit einem Drahtlosen Sensornetz

- Sensornetz zur Lokalisierung von Heckenschützen, Attentätern, Straßenkriminalität
 - Genauigkeit (1m) und Robustheit durch Redundanz
 - Meldung innerhalb von 2s
- Akustische Detektion der Druckwelle der Gewehrmündung



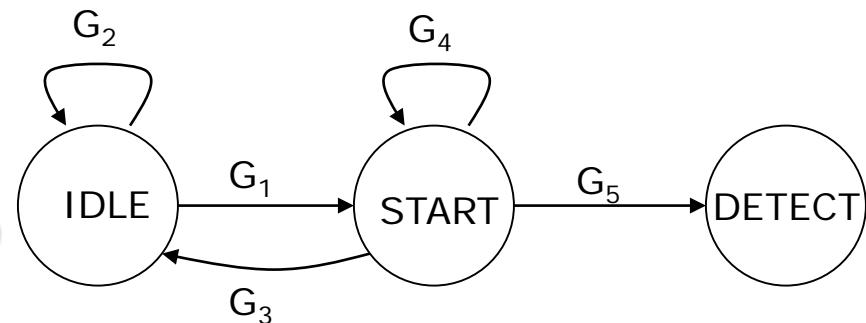
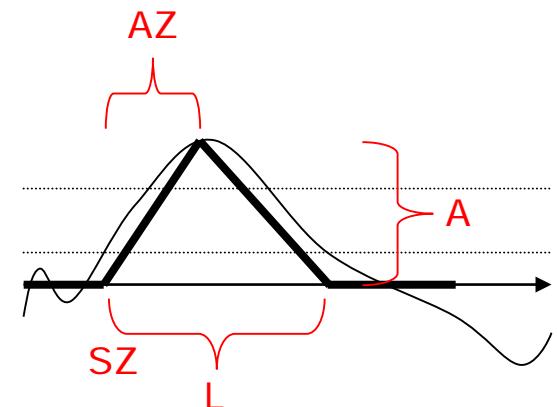
Sensorknoten

- Hardware: Sensorknoten auf Basis von ATmega Mikrocontroller
- Akustisches Sensorboard
 - Mikrofon, Verstärker
 - FPGA für Signalverarbeitung

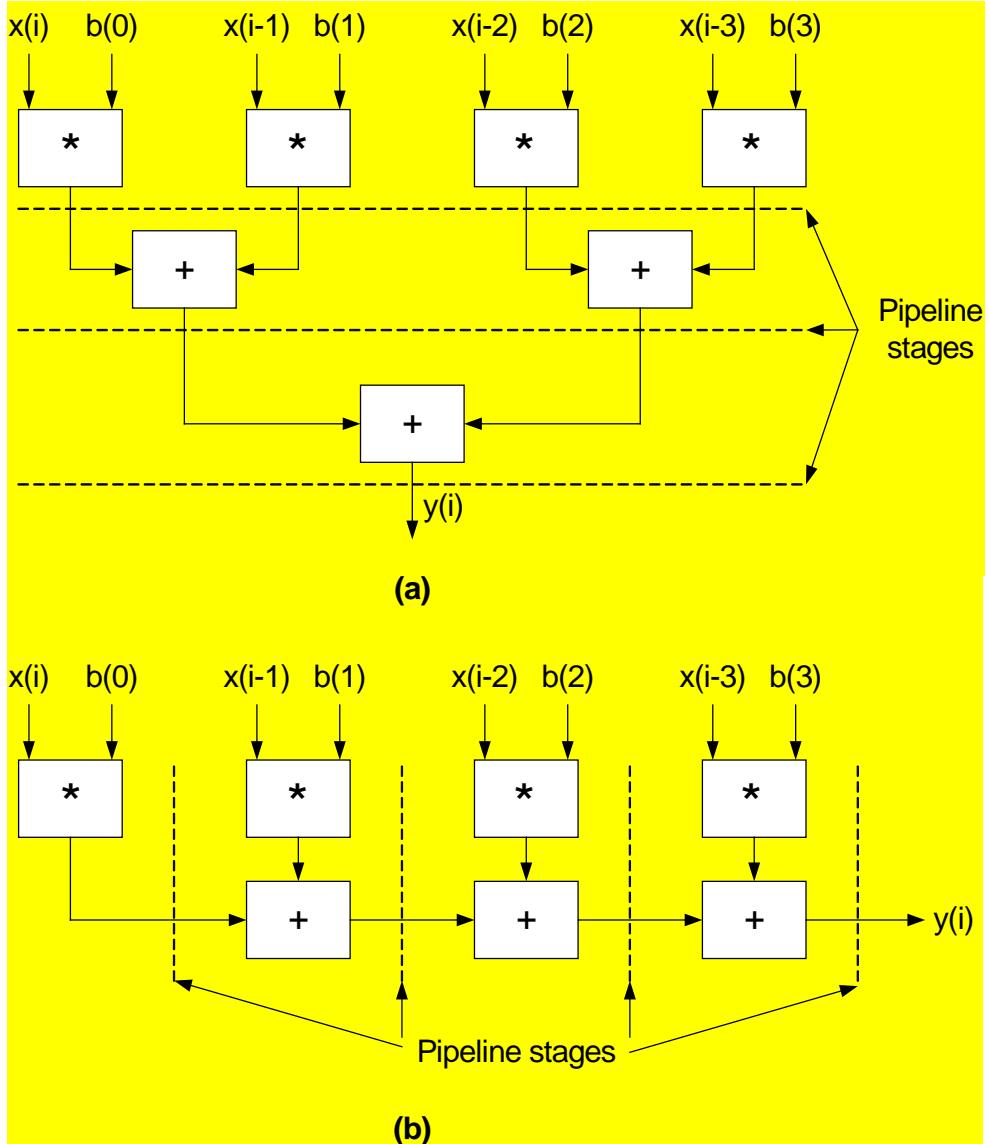


Detektion

- Mikrofon mit 1Mhz abtasten (1us)
- Signalkodierung (ZCC)
 - Startzeit, Länge, Anstiegszeit
 - Amplitude
- Detektion mittels FSM
 - Guards beziehen sich auf ZCC Attribute
 - Zuverlässigkeit ca. 90%



Datenpfad-Architekturen

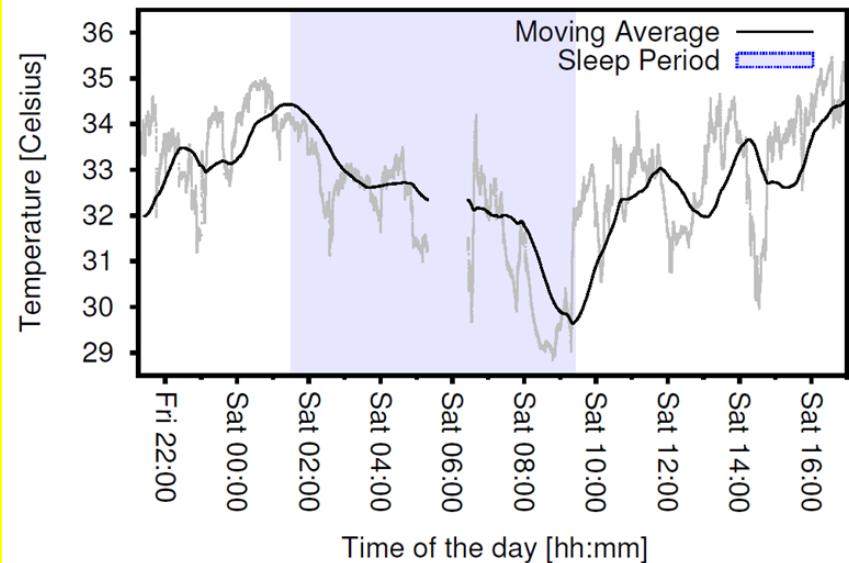


Beispiel:

FIR (Finite-Impulse-Response)-Filter ($N = 4$) als 3-stufige (a) oder 4-stufige (b) Pipeline

Filteroperation:

$$y(i) = \sum_{k=0}^{N-1} x(i-k)b(k)$$



Operation rein datengetrieben, keine explizite Kontrolleinheit.

Anwendung meist auf kontinuierliche Datenströme (Signalverarbeitung).

Implementierung in der Regel als ASIC oder FPGA (Variante (b) besser geeignet, da regelmäßiger).

Komplexe Algorithmen z. B. direkt in Hardware implementierbar.

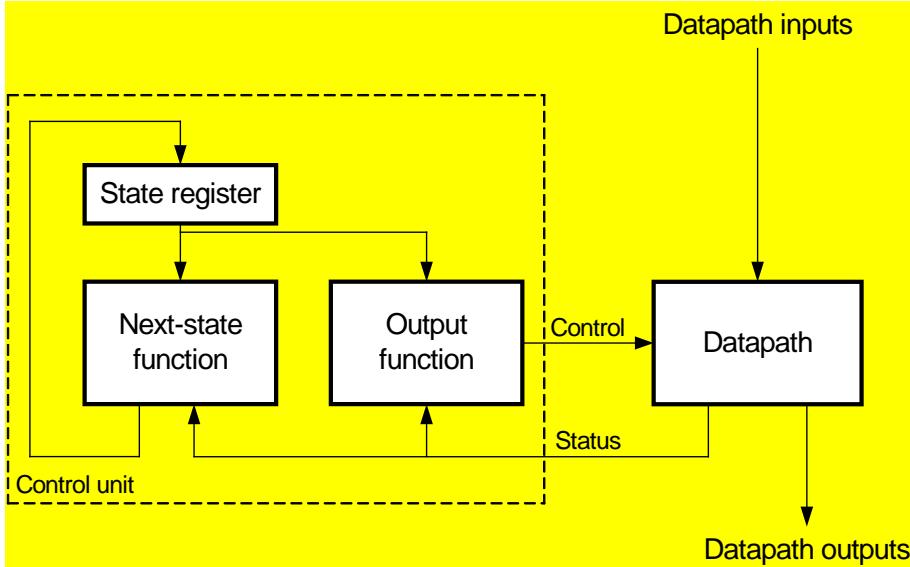
Als IP(Intellectual Property)-Cores für viele gängige Anwendungen verfügbar.

Beispiele:

- Digitale Filter
- Datenkomprimierung (MPEG/JPEG)
- Verschlüsselung/Entschlüsselung

Schnell, aber inflexibel!

Datenpfad mit Ablaufsteuerung



Steuerwerk (Control Unit) steuert den Datenfluss im Operationswerk (Datapath)

Implementierung des Steuerwerks meist als endlicher Automat (FSMD – Finite State Machine with Datapath)

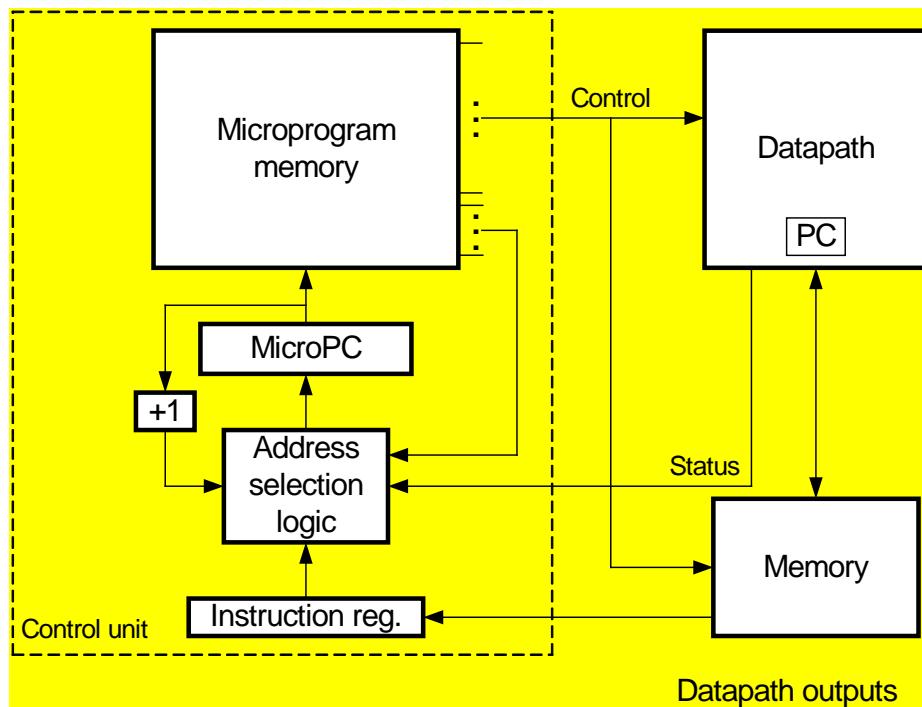
Entwurf auf Registertransfer-Ebene

Implementierung als FPGA oder ASIC

Auch komplexe Schaltwerke realisierbar, aber inflexibel!

2.1.2. Prozessoren

CISC (Complex Instruction Set Computer)-Architekturen



Klassische Prozessorarchitektur mit mikroprogrammiertem Steuerwerk und umfangreichem Befehlssatz.

Einfache 8-Bit- oder 16-Bit Mikrocontroller (z. B. Akkumulatormaschinen wie 68HC08) mit integriertem Speicher und Peripheriekomponenten.

Komfortable Entwicklungsumgebungen für Assembler, C etc.

Standard-Komponenten verfügbar, besonders (Industrie)-PCs.

Flexibel programmierbar, große Software-Basis.

**Veraltet, aber noch
vielfach im Einsatz!**

Weites Spektrum an E/A-Komponenten.

Beispiele Embedded PCs

PC/104 MOPS/520 (Kontron)



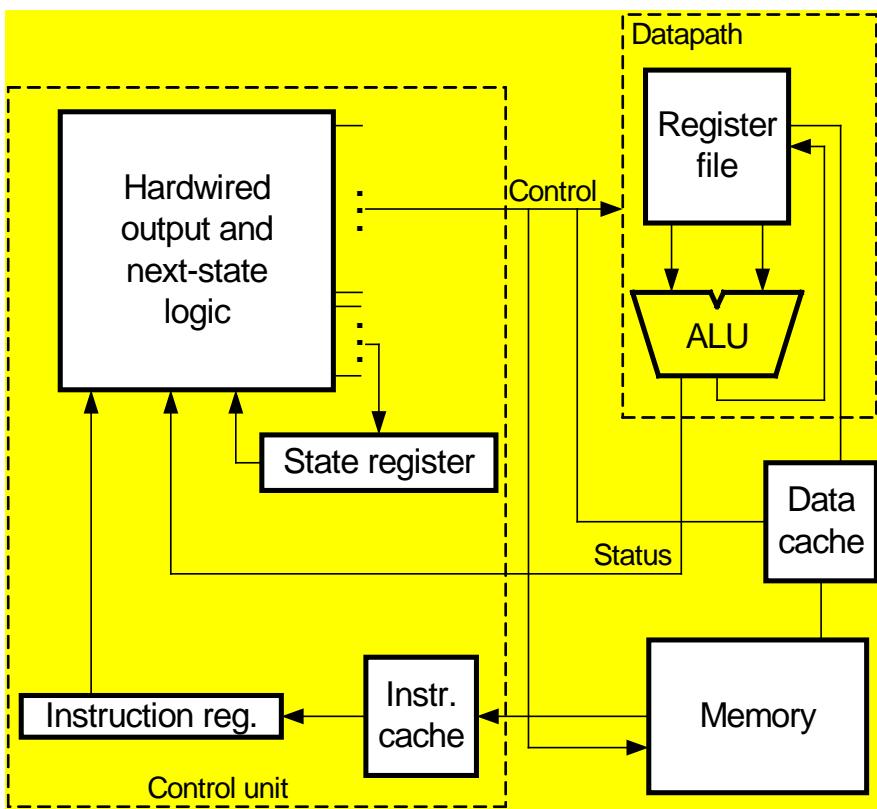
Low Power AMD™ SC520 fanless 133MHz,
onboard 64 MByte SDRAM, 3xRS232C, 1xTTL,
2xUSB, LPT, FDC, IDE, Ethernet (10/100),
without graphic
Windows CE
Embedded Linux

ETX-LX (Kontron)



AMD LX800 with 500 MHz,
graphic, sound,
Ethernet
Windows
Linux

RISC (Reduced Instruction Set Computer)-Prozessor



**Heute Stand der Technik,
auch als IP-Core für FPGAs
(Hardcore, Softcore)!**

Moderne Prozessorarchitektur mit kleinem, effizient implementierbarem Befehlssatz und oft interner Parallelarbeit (z. B. Pipelining).

8-Bit-RISC-Mikrocontroller (z. B. ATmega) mit integrierter Peripherie für einfache Anwendungen.

Komfortable Entwicklungsumgebungen für Assembler, C.

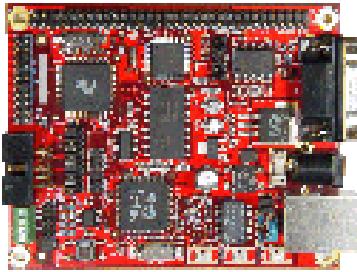
32-Bit-RISC-Prozessoren bei höheren Leistungsanforderungen (z. B. ARM, MIPS, PowerPC, xScale).

Einsatz von schnellen Zwischenspeichern (Caches).

Spezielle Betriebssysteme (z. B. QNX, VxWorks, Windows CE, RT Linux).

Beispiele RISC-Controller

Ethernut 2.1 mit ATmega128 (Egnite)



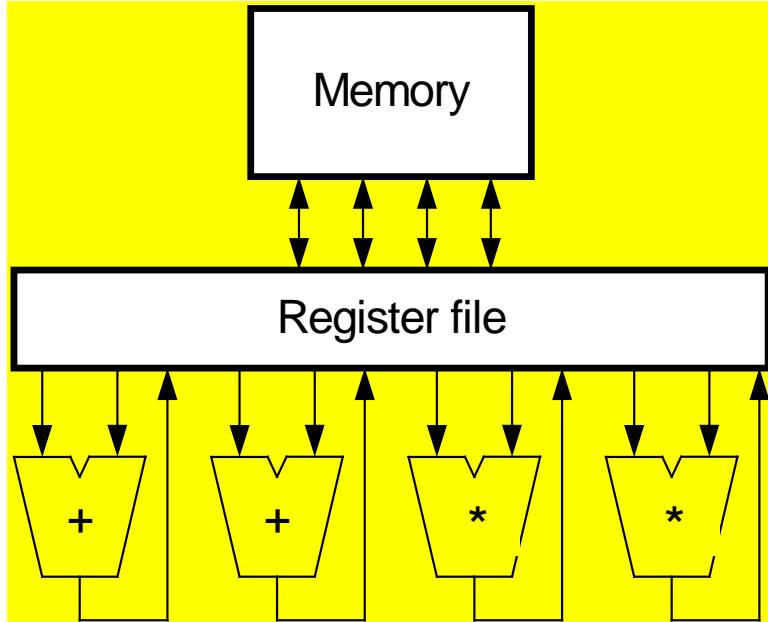
4-layer board with ATmega128 running at 14.7456 MHz,
100 MBit SMSC Ethernet Controller,
512 kBytes banked SRAM
512 kBytes serial flash memory
RT Operating Systems Nut/OS, AvrX

ECO 920 Board mit ARM9 (Garz & Fricke)



High performance 180 MHz ARM9 RISC controller
Flash, RAM, Ethernet, USB, Audio, SD-Card Bus, UARTs
Sehr geringe Leistungsaufnahme (typ. 0,5 Watt)
Erweiterter Temperaturbereich
kostengünstiges Evaluation Board verfügbar
Windows CE verfügbar
Embedded Linux verfügbar

VLIW (Very Large Instruction Word)-Architekturen



Interne Parallelarbeit durch lange Befehlswörter, die gleichzeitig mehrere Rechenwerke zu nutzen gestatten.

Spezielle Compiler erforderlich.

Bisher noch wenig verbreitet, eventuell für die Zukunft interessant!

Digitale Signalprozessoren (DSPs)

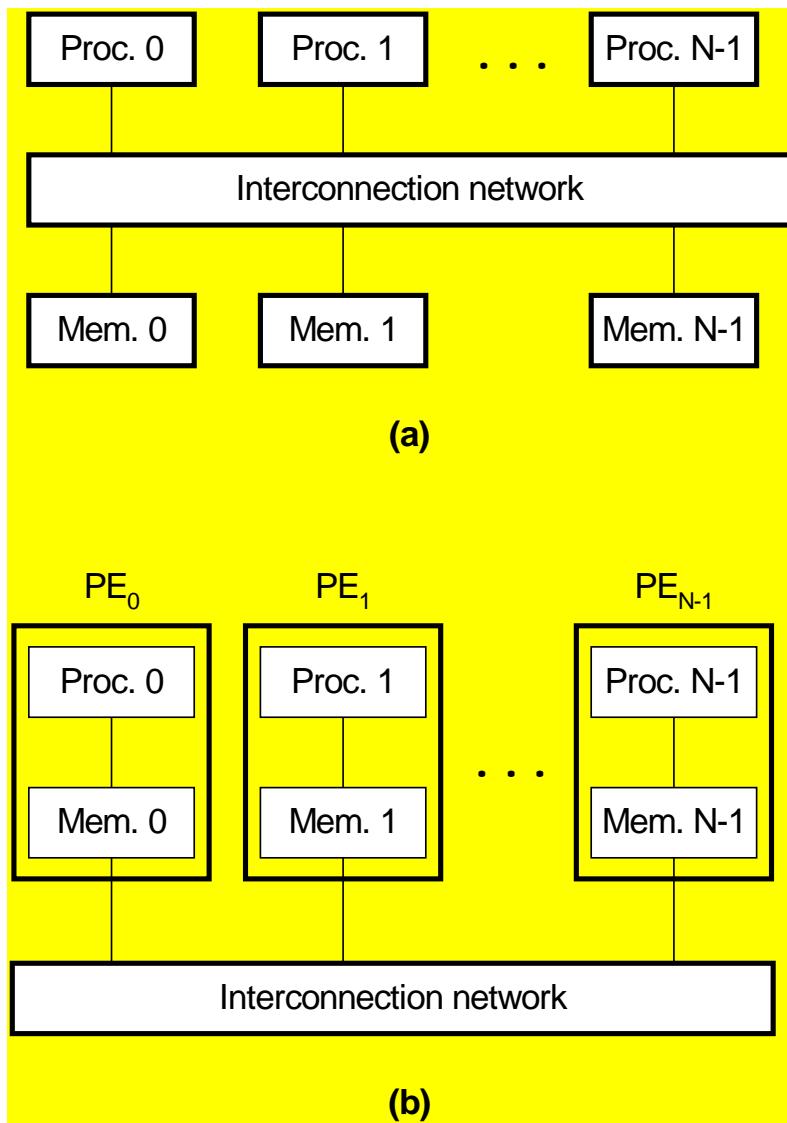
Prozessoren, deren Architektur und Befehlssatz für die Ausführung von Algorithmen der Signalverarbeitung optimiert sind.

Spezielle Programmierwerkzeuge (Assembler, C-Compiler etc.), recht hardwarenahe Programmierung.

Liegen bzgl. Leistung und Flexibilität zwischen Spezialhardware und Universalprozessoren.

Weit verbreitet in eingebetteten Systemen mit Signalverarbeitungskomponenten.

2.1.3 Parallelrechner



MIMD-Rechner (Multiple Instruction Multiple Data)

Multiprozessoren (Shared Memory Machines) (a): Mehrere Prozessoren greifen über ein Verbindungsnetzwerk auf gemeinsamen Speicher zu.

Multicomputer (Message Passing Machines) (b): Mehrere Prozessoren mit lokalem Speicher kommunizieren über ein Verbindungsnetzwerk mittels Nachrichten.

Einsatz bei hohen Rechenzeitanforderungen ('embedded Super-computing'), z. B. in der Bildverarbeitung oder Musterkennung oder Paketverarbeitung in Routern.

Spezielle parallele Programmierung erforderlich.

Multiprozessoren auf einem Chip, z. B. Multicore/Manycore-Architekturen (MPSoC – Multiprocessor System on Chip).

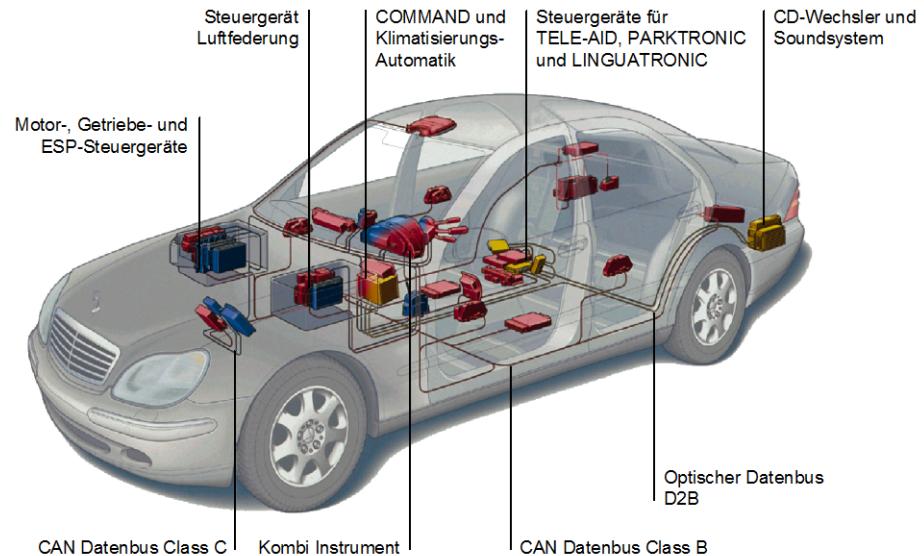
Derzeit wichtiges Forschungsthema!

2.2 Architekturen Eingebetteter Echtzeitsysteme

2.2.1 Beispielanwendung Fahrzeug

Vielzahl verschiedener Funktionen:

- Klimaanlage (Temperaturregulierung)
- Zentralverriegelung (Türen beim Wegfahren automatisch verriegeln)
- Servolenkung (Radeinschlag gemäß Lenkradwinkel)
- = ...



Sensoren

Analog:

- Temperatur
- Radeinschlag

Digital:

- Bewegung (1 = fährt, 0 = steht)

Aktoren

Analog:

- Heizung/Kühlung
- Radeinschlag

Digital:

- Türverriegelung (0 = auf, 1 = zu)

Charakteristika (typisch für eingebettete Echtzeitsysteme)

Sowohl digitale also auch analoge Signale

Zeitbedingungen (Servolenkung)

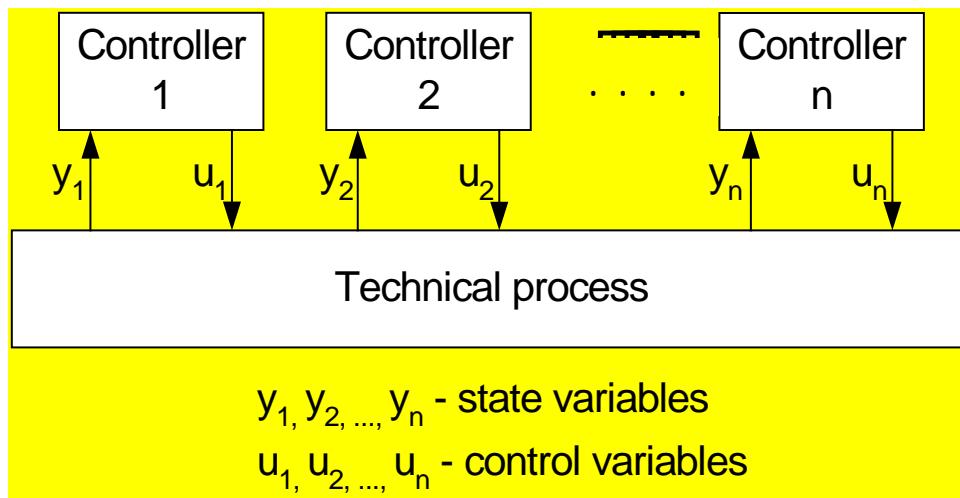
Mehrere parallele Vorgänge, die gleichzeitig kontrolliert werden müssen

Schnittstelle zum Prozessbediener (Fahrer)

Verschiedene Architektualternativen!

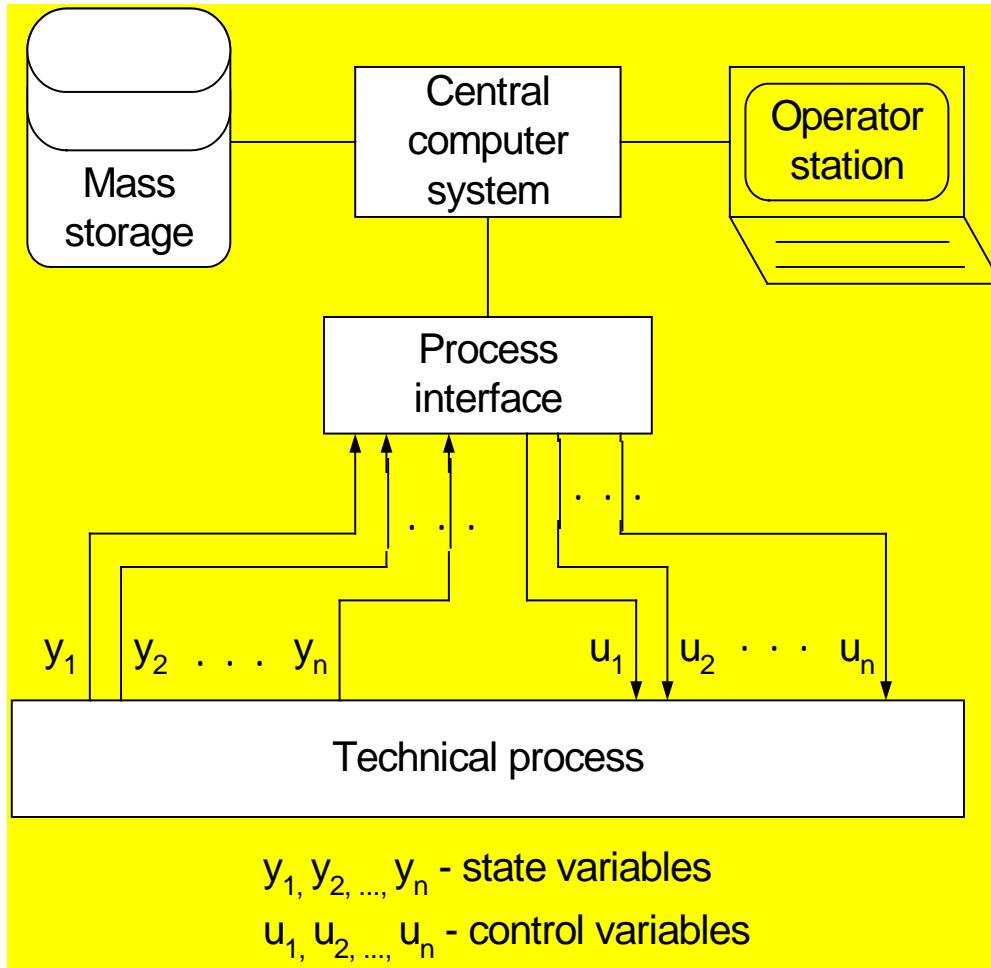
2.2.2 Prinzipielle Controller-Architekturen

Dezentrales Kontrollsysteem



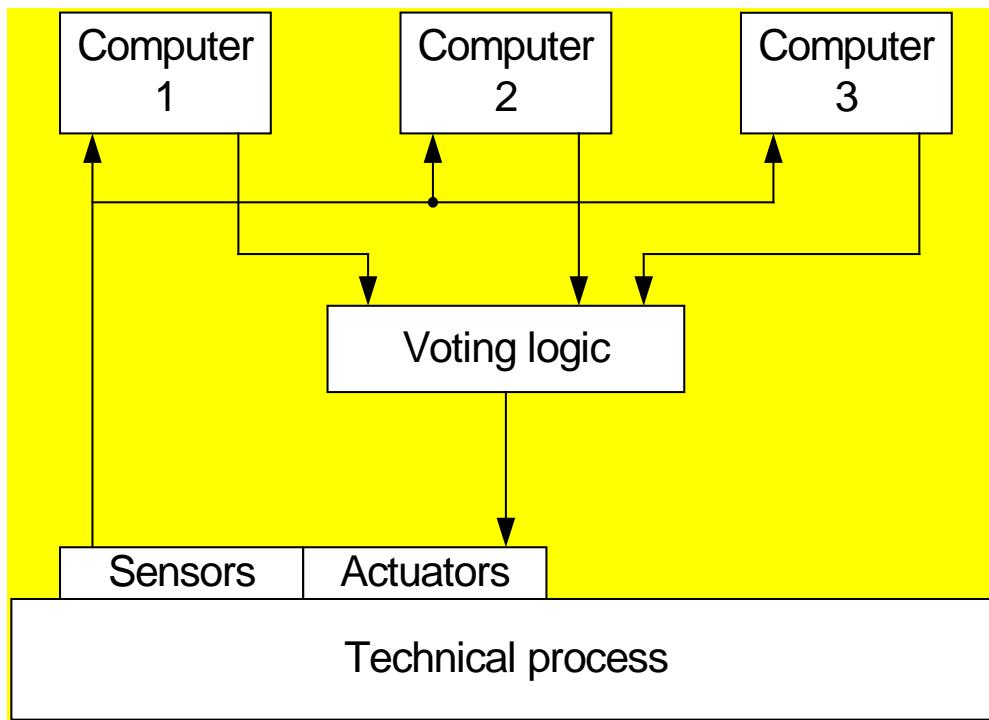
Unabhängige spezielle Regel- oder Steuerkreise
(Spezialhardware oder eigene Mikrocontroller)

Zentralisiertes Kontrollsysteem



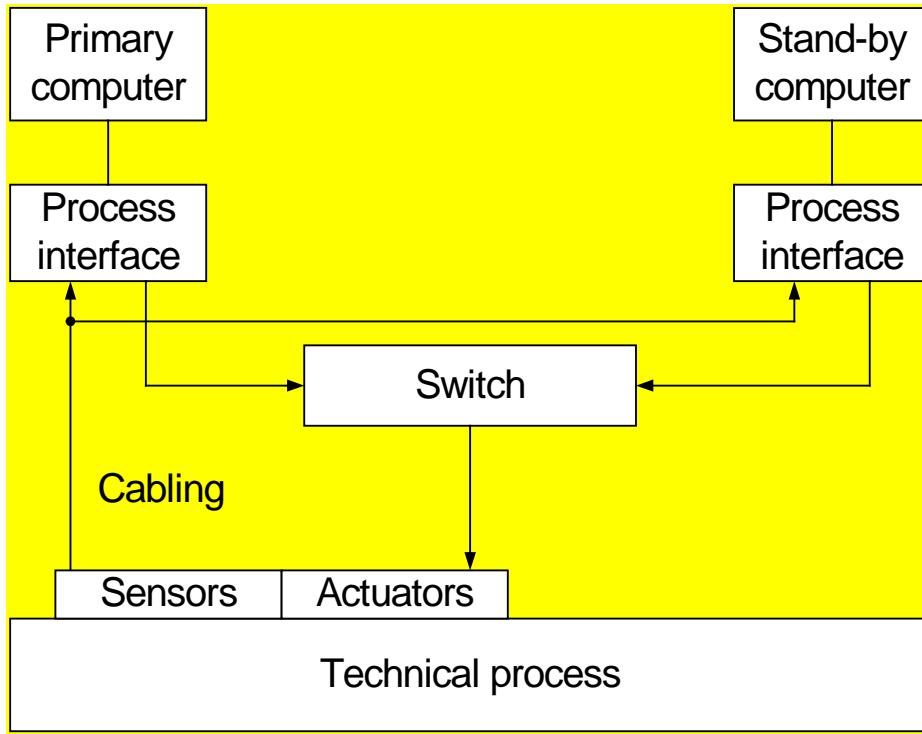
Zentraler Prozessrechner,
Implementierung der Regel- oder Steuerkreise
per Software.

Fehlertolerantes zentralisiertes Kontrollsyste



Beispiel 1:

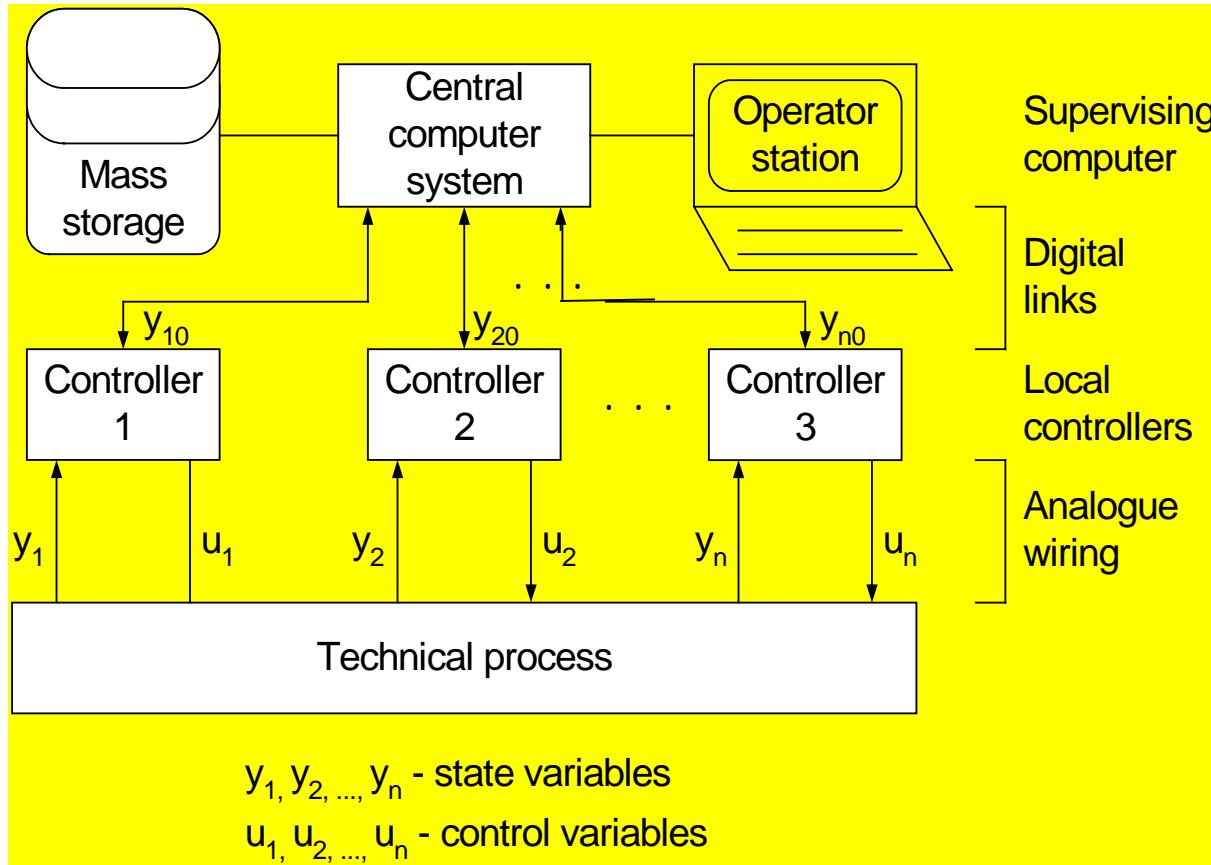
TMR-System mit 2-aus-3-Mehrheitsentscheidung
durch Hardware-Voter (Voting logic)
=> Ein ausgefallener Computer tolerierbar.



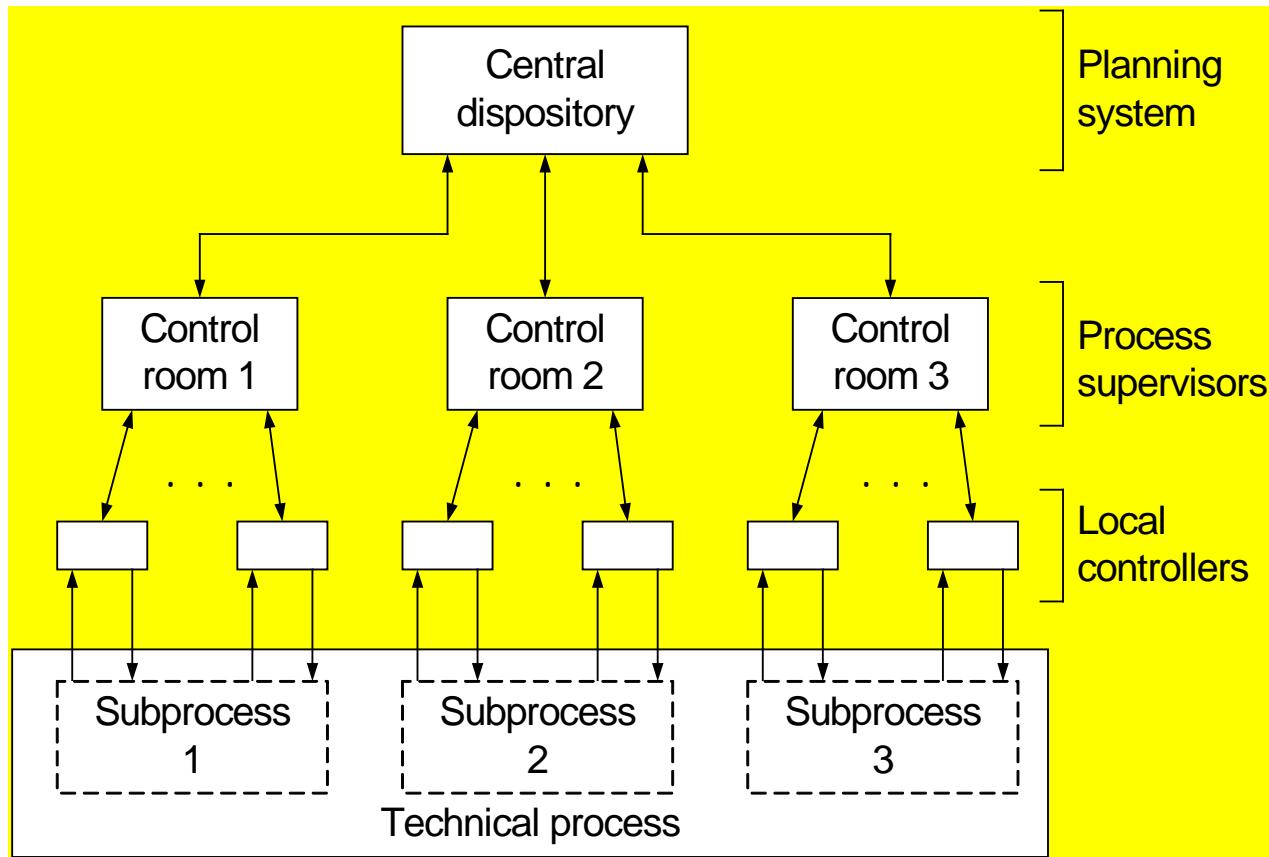
Beispiel 2:

Stand-by Kontrollsyste mit Reserve-Controller, auf den bei Ausfall des primären Rechners mittels eines Schalters (Switch) automatisch umgeschaltet wird.

Mehrebenen-Kontrollsysteme

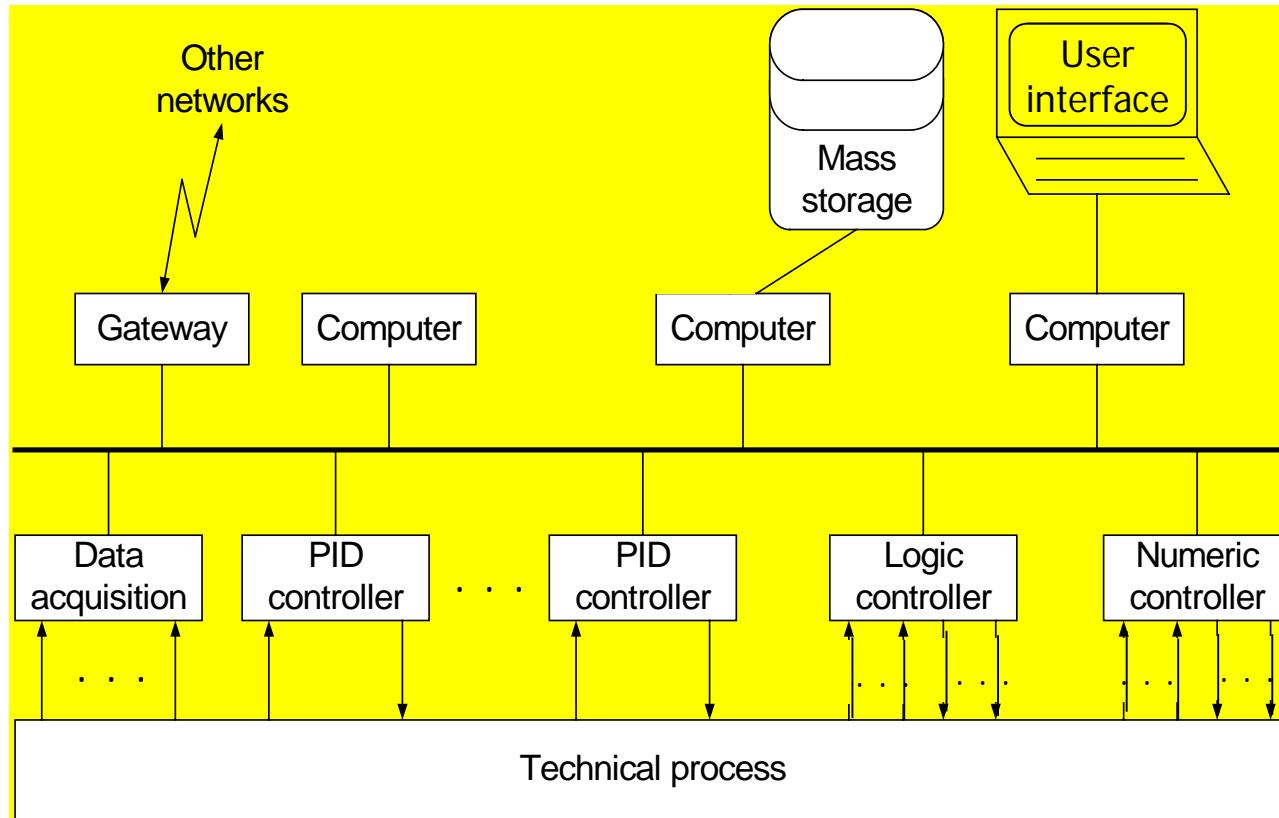


Beispiel 1: Set-Point Kontrollsyste mit mehreren Kontrollsleifen (Regler oder Steuerungen), die von einem Zentralrechner koordiniert werden.



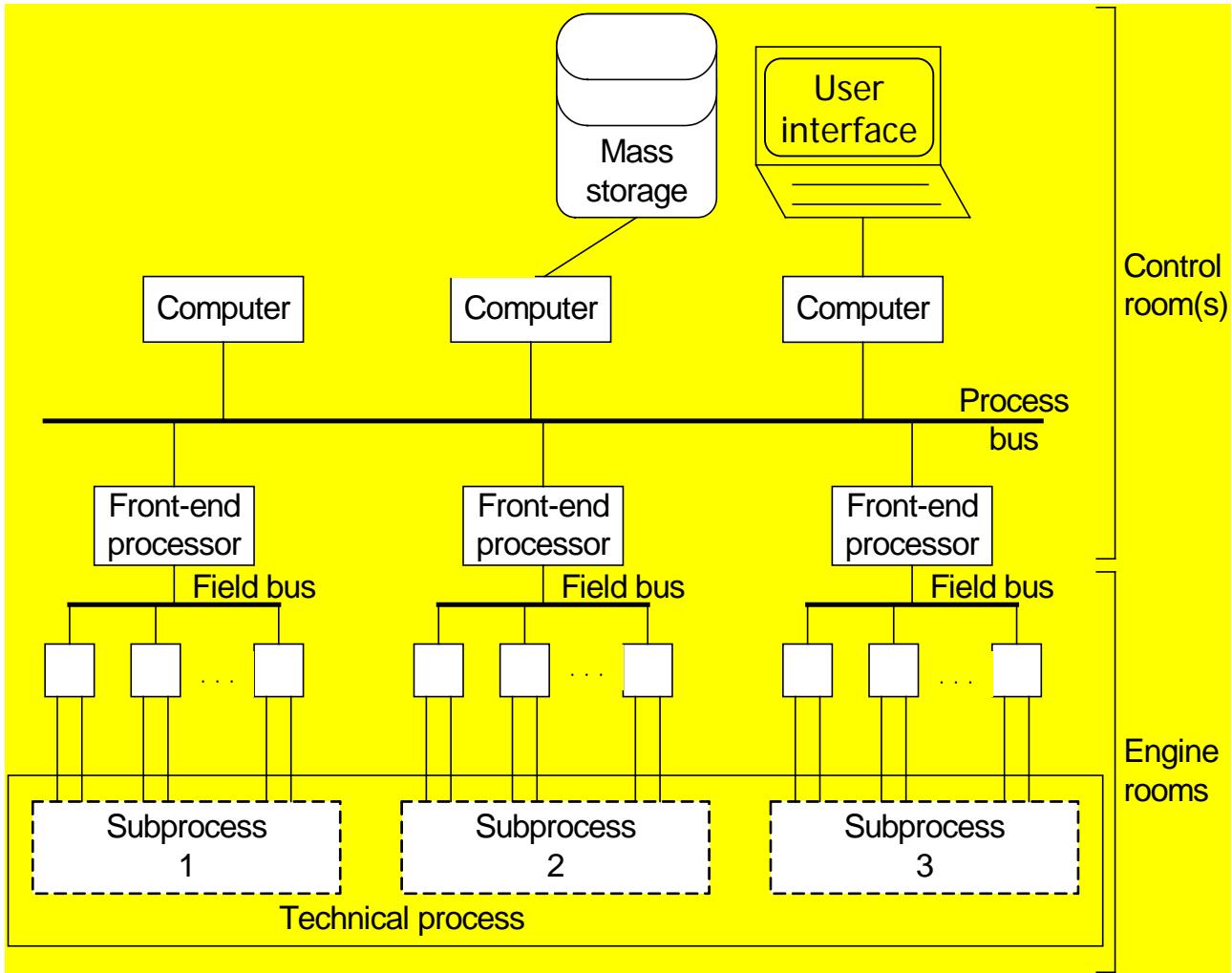
Beispiel 2:
3-Ebenen-Kontrollsyste

Netzwerkbasierte verteilte Kontrollsysteme



Beispiel 1:

LAN-basiertes zweistufiges Kontrollsyste

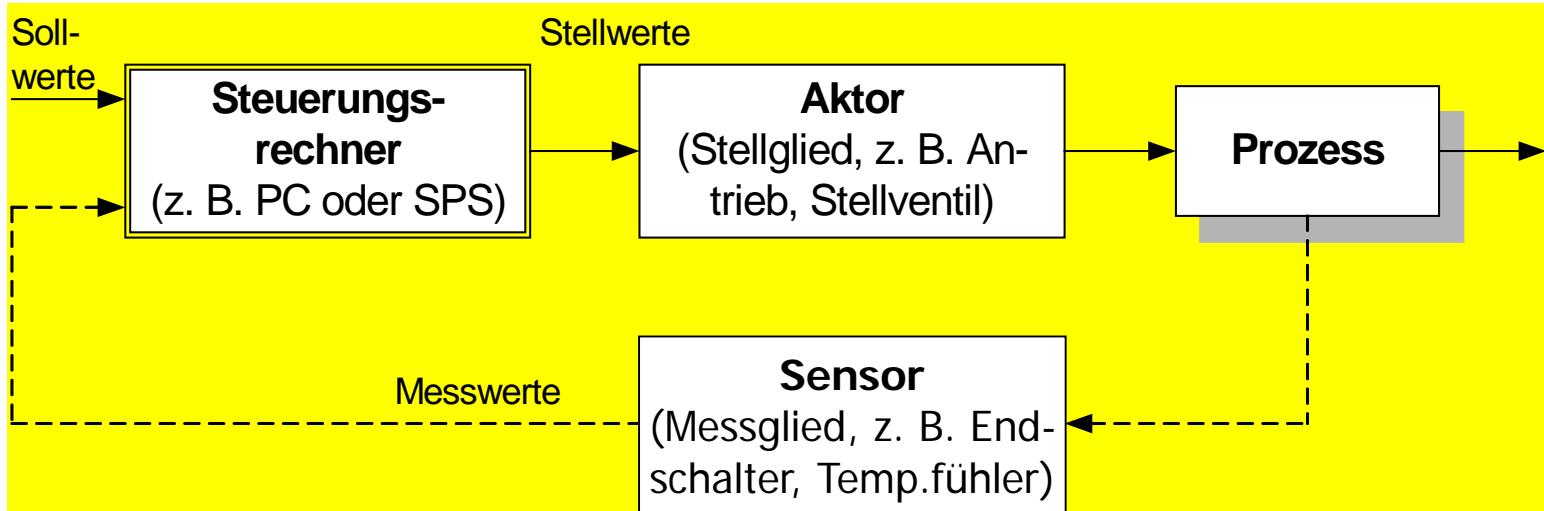


Beispiel 2:

3-stufiges Kontrollsysteem mit Feldbussen (hart echtzeitfähig) zur Verbindung von 'intelligenten' Sensoren und Aktoren mit Front-end Prozessoren und LAN (weich echtzeitfähig) als Prozessbus

2.3 Steuerungsrechner (Control Computer)

2.3.1 Steuern und Regeln



Steuerung eines technischen Prozesses

Vorgang, bei dem eine oder mehrere Größen als Eingangsgrößen andere Größen als Ausgangsgrößen aufgrund der dem System eigenen Gesetzmäßigkeiten beeinflussen (DIN 19226).

Arten von Steuerungen

- **Offener Wirkungsweg (ohne Rückführung):**
Steuersignale wirken auf Prozess ohne Erfassen des Prozesszustandes direkt ein (z. B. Getriebesteuerung für Motoren)
- **Geschlossener Wirkungsweg (mit Rückführung):**
Messgrößen werden erfasst, wirken sich aber nicht kontinuierlich auf die Stellgrößen aus (z. B. Endschalter, Alarmanlagen)
- **Analoge Steuerungen:**
überwiegend kontinuierliche Steuersignale
- **Digitale und binäre Steuerungen:**
Steuerung mit digitalen bzw. binären (zweiwertigen) Signalen

- **Verknüpfungssteuerungen:**

Steuergröße entsteht durch Verknüpfung mehrerer Signale

- **Ablaufsteuerungen:**

Steuervorgänge werden schrittweise zeit- oder prozessabhängig ausgelöst

- **Verbindungsprogrammierte Steuerungen:**

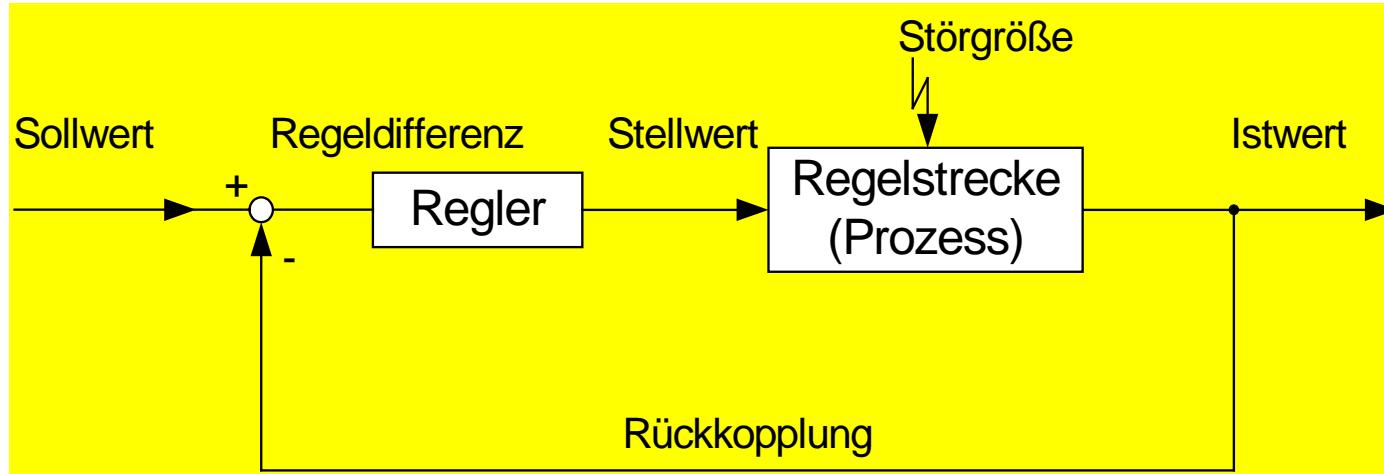
Leitungsverbindungen (Verdrahtung) bestimmen den Programmablauf (z. B. Schütz- oder Relaissteuerungen)

- **Speicherprogrammierte Steuerungen (SPS):**

enthalten elektronischen Programmspeicher, der frei programmiert werden kann

Regelung technischer Prozesse

Vorgang, bei dem die zu regelnde Größe *fortlaufend* erfasst und so beeinflusst wird, dass sie sich der Führungsgröße angleicht (z. B. Heizungsthermostat).

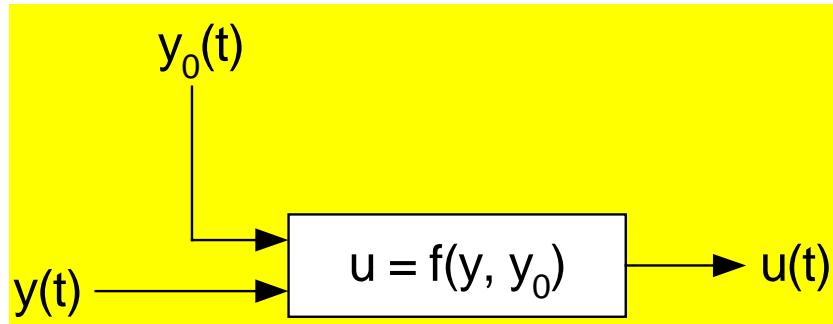


Regelungen haben immer einen *geschlossenen Wirkungskreis*. Analoge und digitale bzw. binäre Regelungen sind möglich.

Festwertreglung: Sollwert bleibt konstant

Folgeregelung: Sollwert ist eine vorgegebene Funktion der Zeit

Regler für kontinuierliche Signale



y(t): Istwert (vom Sensor)
y₀(t): Sollwert (vom Prozessbediener)
u(t): Stellwert (an den Aktor)
Ziel: Istwert soll möglichst nahe am Sollwert gehalten werden.

Häufig verwendeter Ansatz:

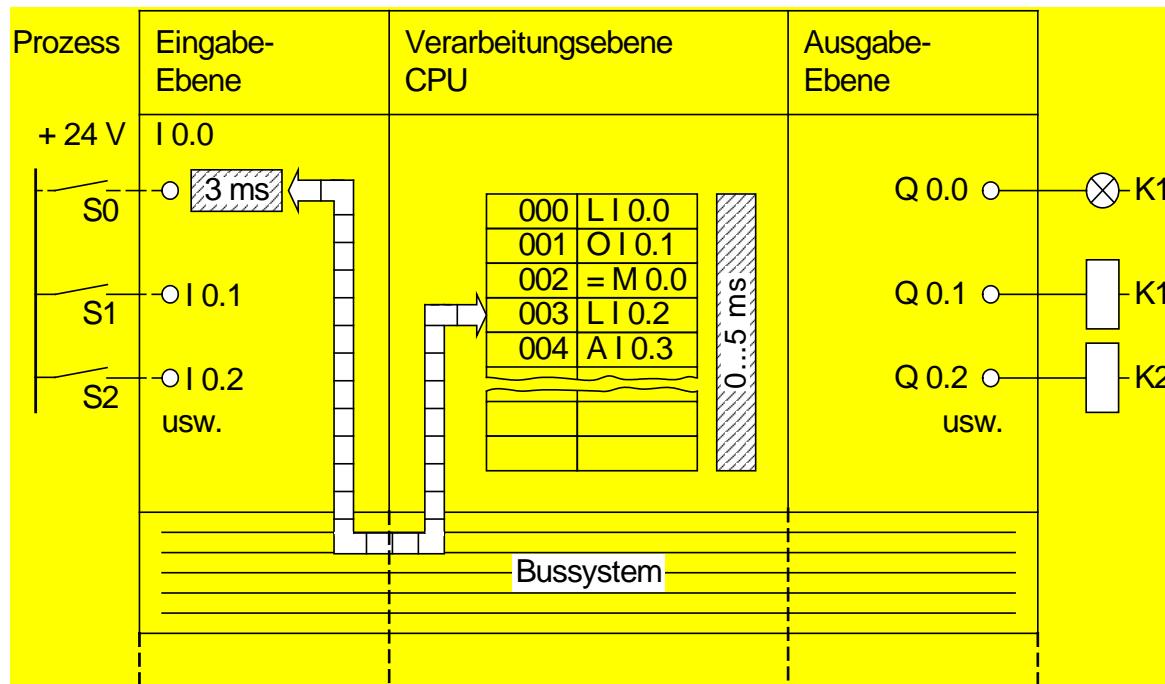
PID (Proportional-Integral-Differential)-Regler

$$u(t) = K_p \left((y_0(t) - y(t)) + \frac{1}{T_i} \int_0^t (y_0(\tau) - y(\tau)) d\tau + T_d \frac{d(y_0(t) - y(t))}{dt} \right)$$

Koeffizienten K_p , T_i , T_d gewichten den P-, I- und D-Anteil und stimmen die Dynamik des Reglers auf den Prozess ab (werden bei der Inbetriebnahme des Systems ermittelt).

2.3.2 SPS (PLC – Programmable Logic Controller)

Hart-echtzeitfähiges Automatisierungsgerät, das per Software einfache Steuerungen und Regelungen zu implementieren erlaubt. Ersetzt festverdrahtete (verbundungsorientierte) elektromechanische oder elektronische Steuerungen (z. B. Schaltschütze, TTL-Logik).

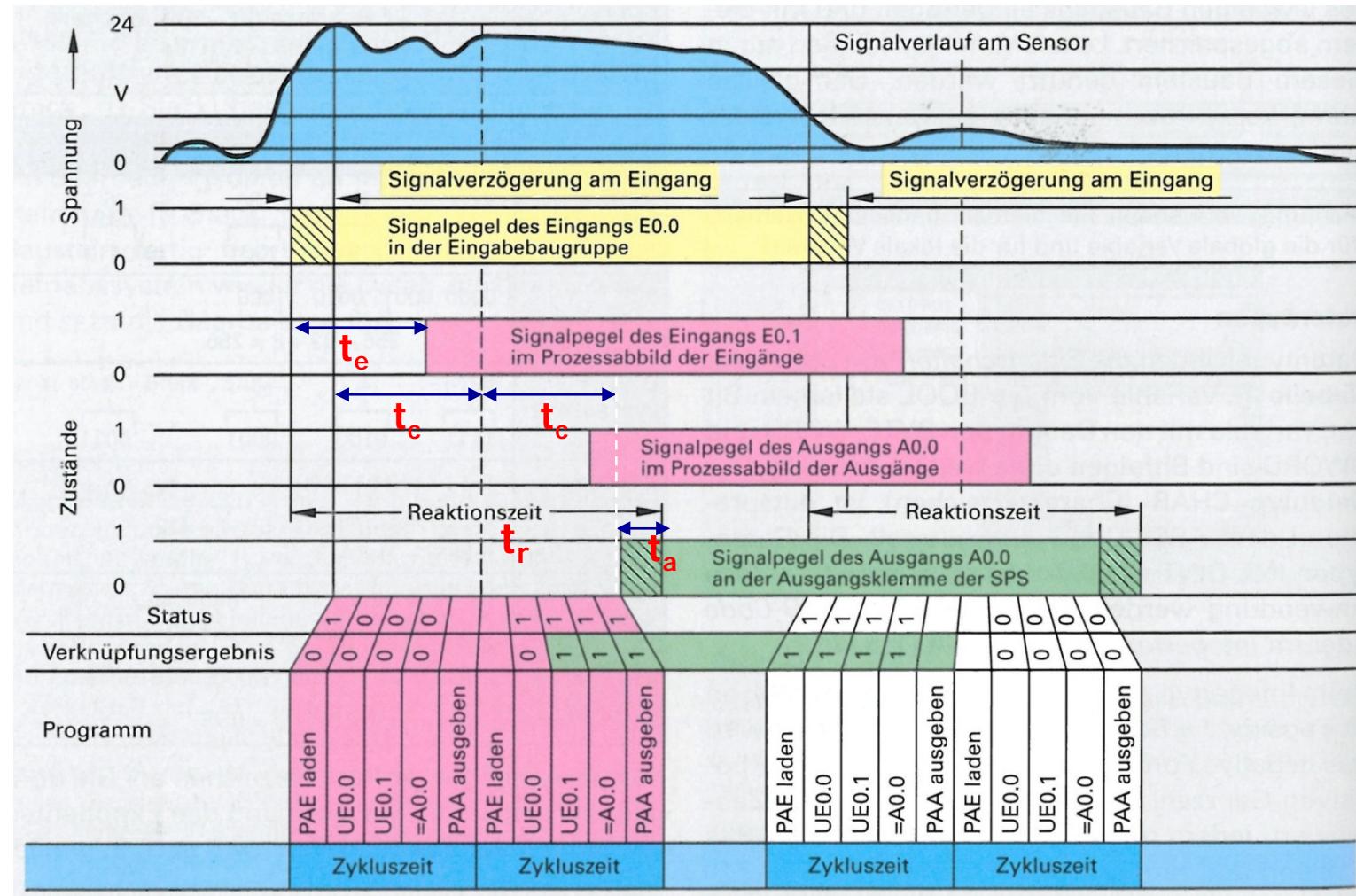


Streng *zyklische Arbeitsweise*
(typ. Zykluslänge: wenige Millisekunden):

- Einlesen der Sensoren (Signalgeber) in Eingabe-Abbildungsregister (während Berechnung konstant)
- Verarbeitung (sammeln der Ergebnisse im Ausgabe-Abbildungsregister)
- Ausgabe an Aktoren (Stellgeräte, Anzeiger)

EVA-Prinzip

Reaktionszeit einer SPS



Ausgabezeit oft vernachlässigt, da klein.

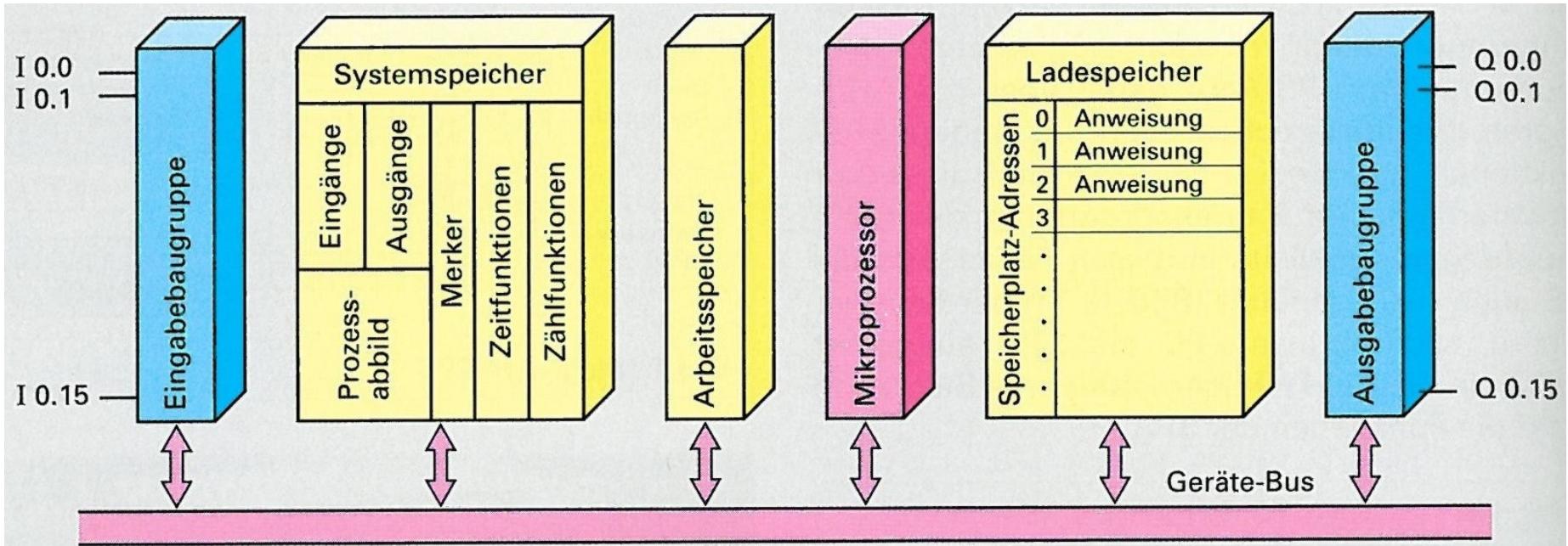
Oft werden in dem Zyklus auch Systemprogramme wie z. B. Selbsttests ausgeführt.

Bild zeigt nicht den Worst Case!

Maximale Reaktionszeit t_r von Eingabe bis Ausgabe:

$$t_r = t_e + 2 t_c + t_a \quad t_e : \text{Eingabezeit}, t_c : \text{Zykluszeit}, t_a : \text{Ausgabezeit}$$

Blockschaltbild einer SPS

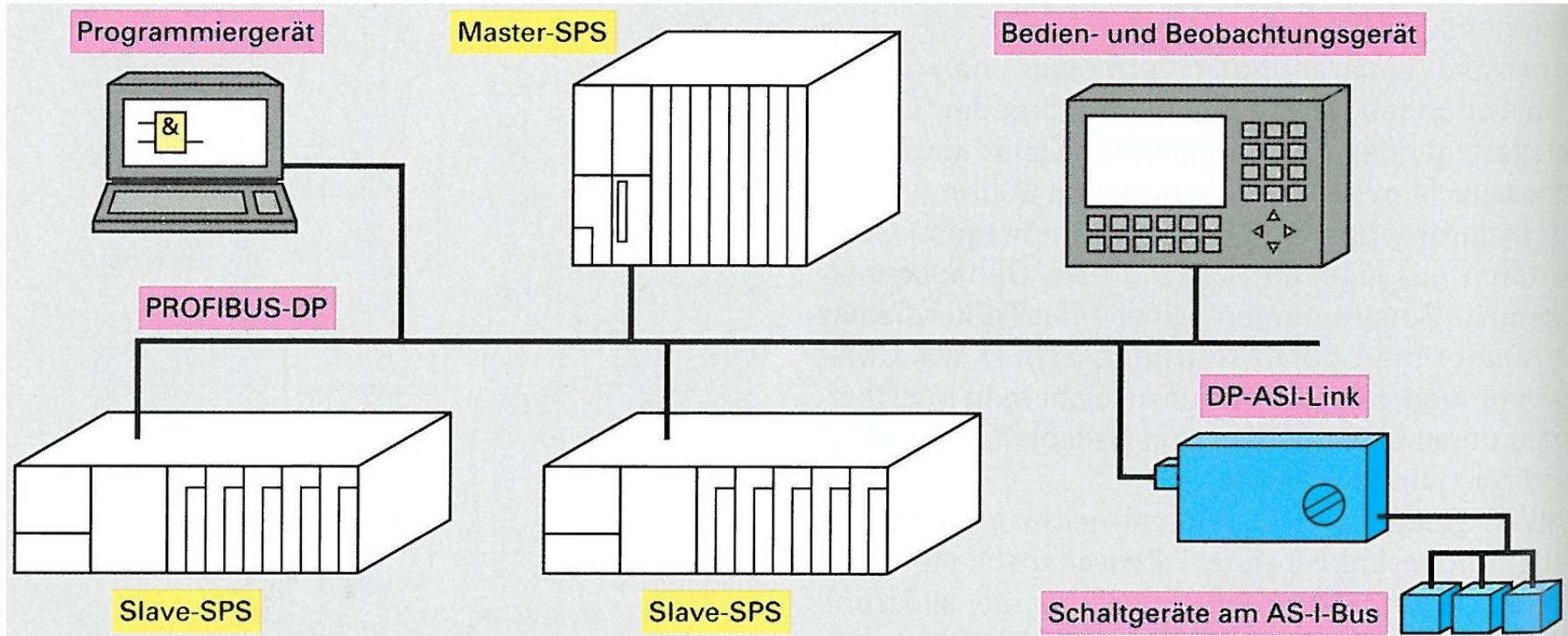


(Standard-)Mikroprozessor zur zyklischen Programmausführung

Speicher für Programm und Daten (Lade- bzw. Arbeitsspeicher), Systemspeicher für Prozessabbild, Merker (Variablen) sowie Zeit und Zählfunktionen

Galvanische Trennung und Aufbereitung der E/A-Signale. Verschiedene Ausführungen mit unterschiedlicher Anzahl von binären und analogen E/A-Signalen

Vernetzte SPS



Heute oft vernetzte SPSen mit hierarchischer Konfiguration (Master/Slave), Programmiergeräten (PCs), Bedien- und Beobachtungsgeräten (HMI – Human Machine Interface).

Auch Sensoren/Aktoren über Busse ankoppelbar.

Spezielle echtzeitfähige Busse erforderlich (PROFIBUS-DP, ASI-Bus etc.)

Schrank-SPS

klassische Form, bei der die SPS in einen Schaltschrank auf eine Hutschiene eingebaut wird (Kompaktgerät oder modulare SPS)

Programmierung offline per PC.

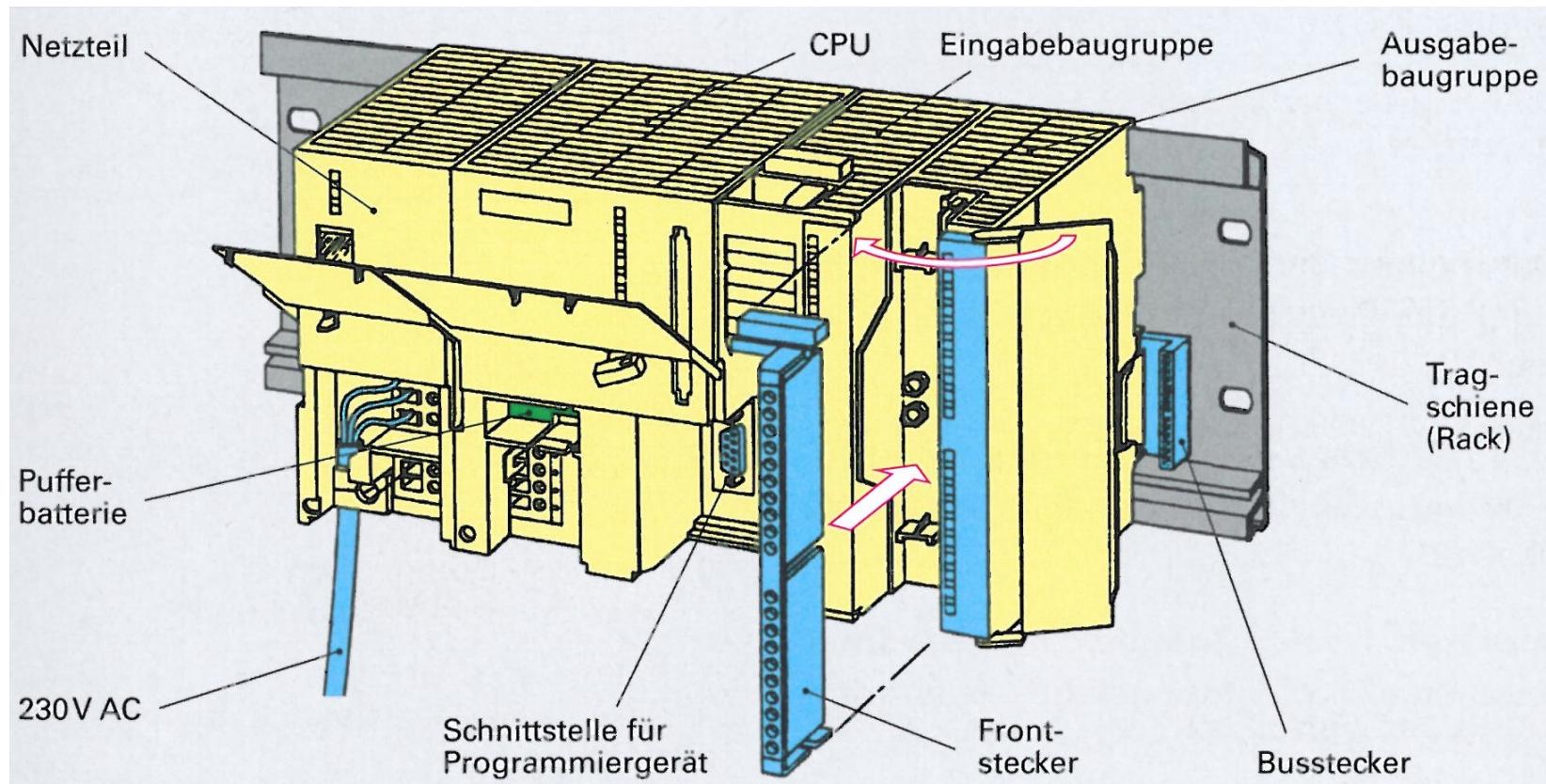
Beispiel: Kompakt-SPS Siemens SIMATIC S7-1200 CPU 1212C

- 8 binäre Eingänge
- 5 binäre Ausgänge
- 2 analoge Eingänge
- Programmspeicher 1 MByte
- Arbeitsspeicher 25 Kbyte
- Bitoperationszeit: $0.1\mu\text{s}/\text{Operation}$
- schnelle Zähler 1x30, 3x100 kHz
- Ethernet-Schnittstelle
- internes 220 V-Netzteil
- 24 V Stromversorgung für Aktoren/Sensoren
- Erweiterungsmodule (digitale, analoge E/A, Netzwerkschnittstellen etc.)
- Programmierung: STEP 7 Basic (KOP, FUP)



Komplette SPS
in einem Modul!

Modulare SPS



CPU und Netzteil separat. Baugruppen für digitale oder analoge EA, Spezialfunktionen (z. B. Regler), Kommunikation etc. können direkt angereiht werden.

Flexibel an Anwendung anpassbar!

Beispiel: Modulare SPS Siemens S7-300



Bis zu 2 MByte Arbeits-, 8 MByte Programmspeicher

Bitoperationszeit min. 0.004µs.

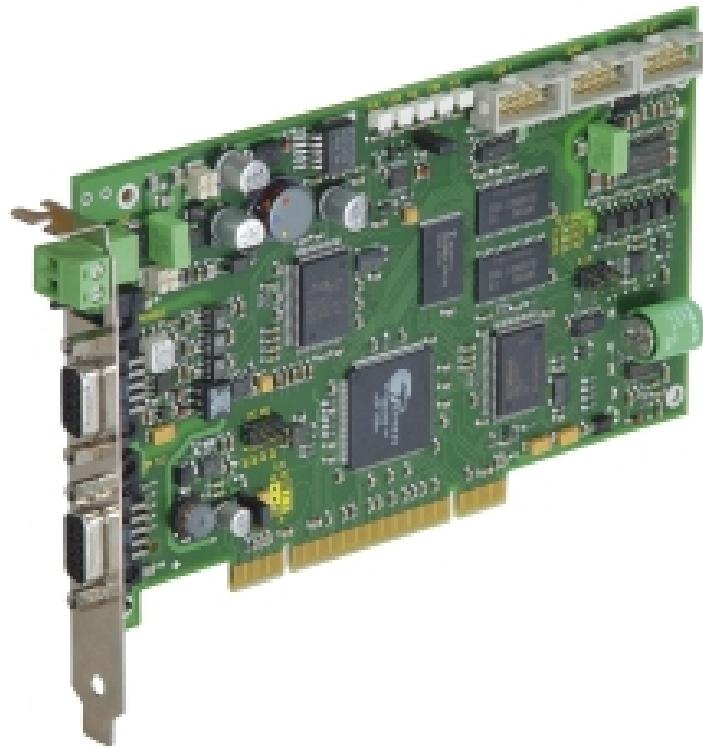
Integrierte Kommunikationsschnittstellen (RS 485, Profibus, Ethernet)

Programmierung: STEP7 (AWL, FUP, KOP, S7-GRAF, SCL)

Slot-SPS

Einsteckkarte für PC, die alle Module einer SPS enthält. PC dient zur Programmierung (offline) oder zur Prozessvisualisierung (online)

Beispiel: KUHNKE PC Control 645-12M PCI



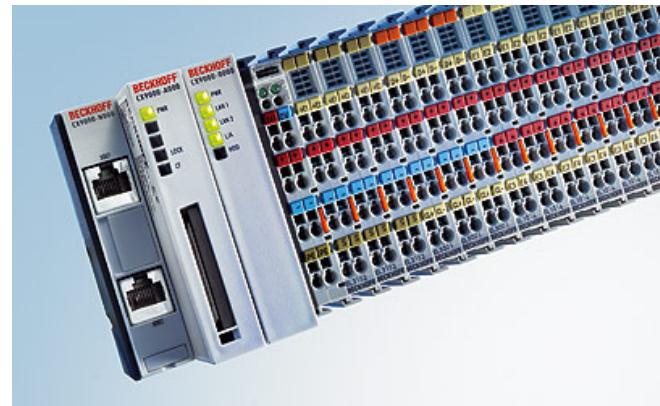
PCI-Bus-Karte mit

- CPU
- 2 digitalen Eingänge
- 2 digitale Ausgänge
- Feldbus-Schnittstelle
(PROFIBUS DP) für
weitere Sensoren/Aktoren
- Serielle Schnittstelle

Reine Softwarelösung auf Standard- oder Industrie/Embedded-PC-Basis. Sensor/Aktorankopplung über Einsteckkarten für Feldbusse oder Standard-Ethernet. Bei Embedded PCs auch direkter Anschluss von Busklemmen.



Industrie-PC Beckhoff C3320



Embedded PC Beckhoff CX9000 mit Busklemmen

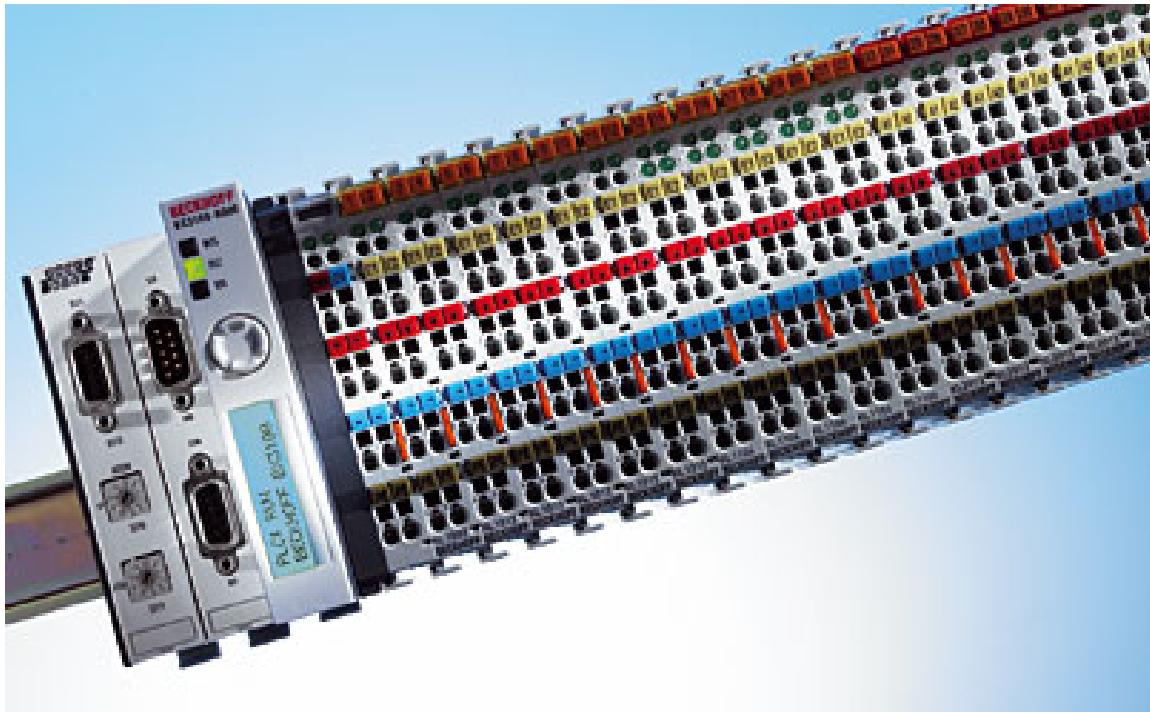
Beispiel: CoDeSys von 3S bzw. TwinCAT von Beckhoff

- Unterstützung aller gängigen SPS-Programmiersprachen
- „objektorientierte“ Programmierung
- Prozesssimulation und -animation mit graphischer Oberfläche
- Feldbusunterstützung z. B. für Profibus oder ProfiNet/Ethernet

Busklemmen

Anbindung von Sensoren/Aktoren bei Slot-SPS und Soft-SPS nicht direkt möglich, sondern über Feldbusse (Profibus, CAN-Bus etc.) bzw. Ethernet.

Busklemmensystem: Busklemmen-Controller für gängige Feldbussysteme bzw. Ethernet plus Module für digitale oder analoge Ein/Ausgabe, Winkel/ Wegmessung etc. Abschluss mit Busendklemme. Montage auf Hutschiene.

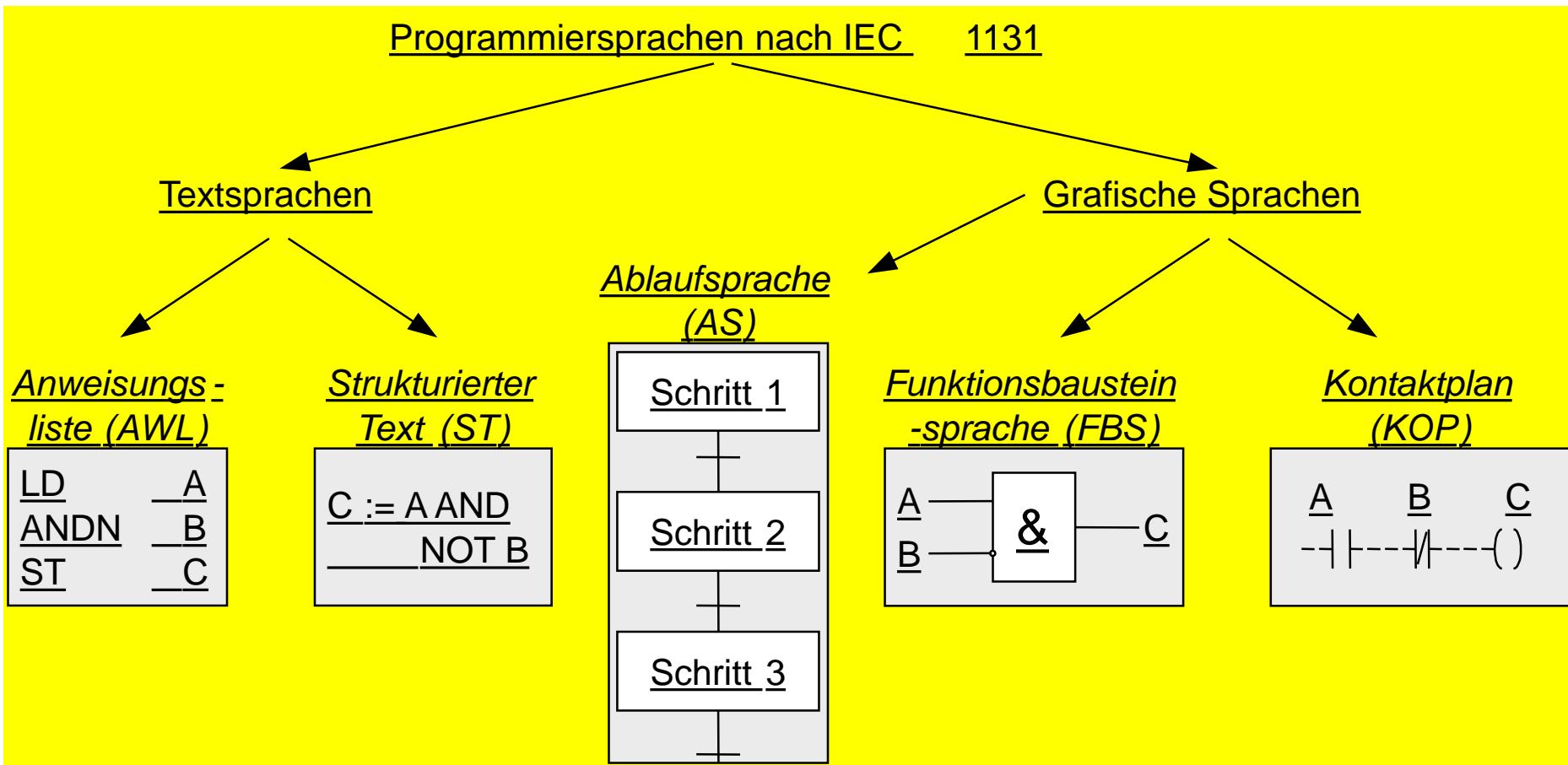


Beispiel:
Beckhoff-Busklemmen mit Controller

Vorteil:
Durch modularen Aufbau flexibel für jeweilige Anwendung und jeweiliges Bussystem konfigurierbar

Auch als modulare SPS erhältlich (CPU statt Busklemmen-Controller)

2.3.3 Programmiersprachen für SPS (genormt nach IEC 1131)



IEC-Sprachen

AWL:

Angelehnt an den Assembler einer einfachen Akkumulatormaschine

ST:

Pascal-ähnliche Programmiersprache

KOP:

Angelehnt an Stromlaufplan einer Schützsteuerung (Kontakte, Spulen)

FBS/FUP:

Angelehnt an Schaltbilder für Schaltnetze und Schaltwerke (Symbole für Gatter, Flipflops etc.). Auch als Funktionsplan (FUP) bezeichnet.

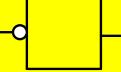
AS:

Angelehnt an Petri-Netze. Speziell für Ablaufsteuerungen (Schrittketten), deren Schritte in einer der anderen Sprachen programmiert werden.

Sprachen können teilweise auch in einem Programm gemischt werden.

Trotz Normierung viele herstellerspezifische 'Dialekte':

SPS-Grundfunktionen nach IEC 1131 (Auswahl)

Benennung	Darstellung		
	AWL	FBS/FUP	KOP
UND/AND	AND		
ODER/OR	OR		
NICHT/NOT (Eingang)	NOT		
NICHT/NOT (Ausgang)	NOT		
Zuweisung	ST		
Setzen/Set	S		
Rücksetzen/ Reset	R		

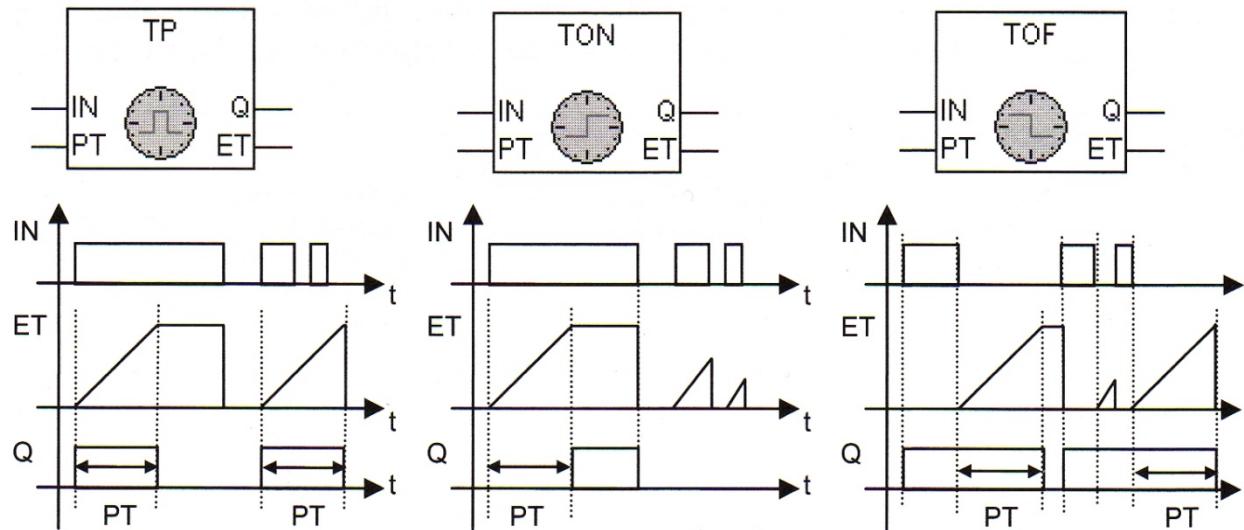
Logische Verknüpfungen (UND/ODER/NICHT) sowie einfache Speicherfunktionen (Flipflops)

Funktionsbausteine (FBs)

Standard-FBs (vordefiniert)

- Flipflops
- Zähler
- Schieberegister
- Zeitglieder
- Vergleicher
- etc.

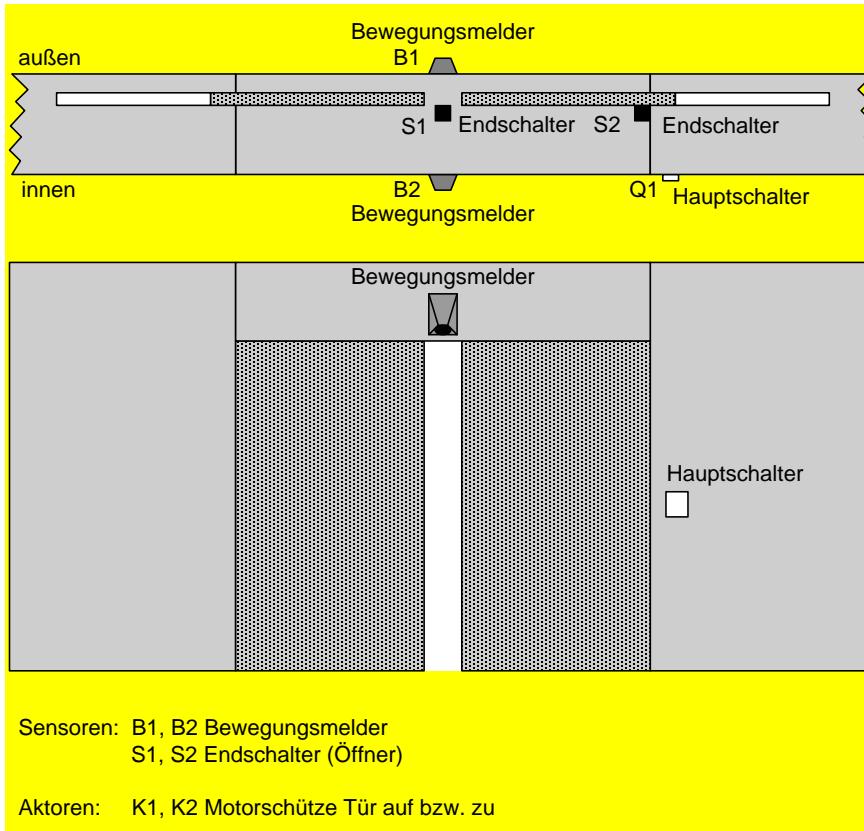
Beispiel: Zeitglieder (IEC 1131-3) in CoDeSys/TwinCAT



TP: Puls-Timer
TON: Eingangsverzögerung
TOF: Ausgangsverzögerung

Auch komplexe (z. B. PID-Regler) oder selbst geschriebene FBs.

2.3.4 Elementare SPS-Programmierung (Verknüpfungssteuerungen)



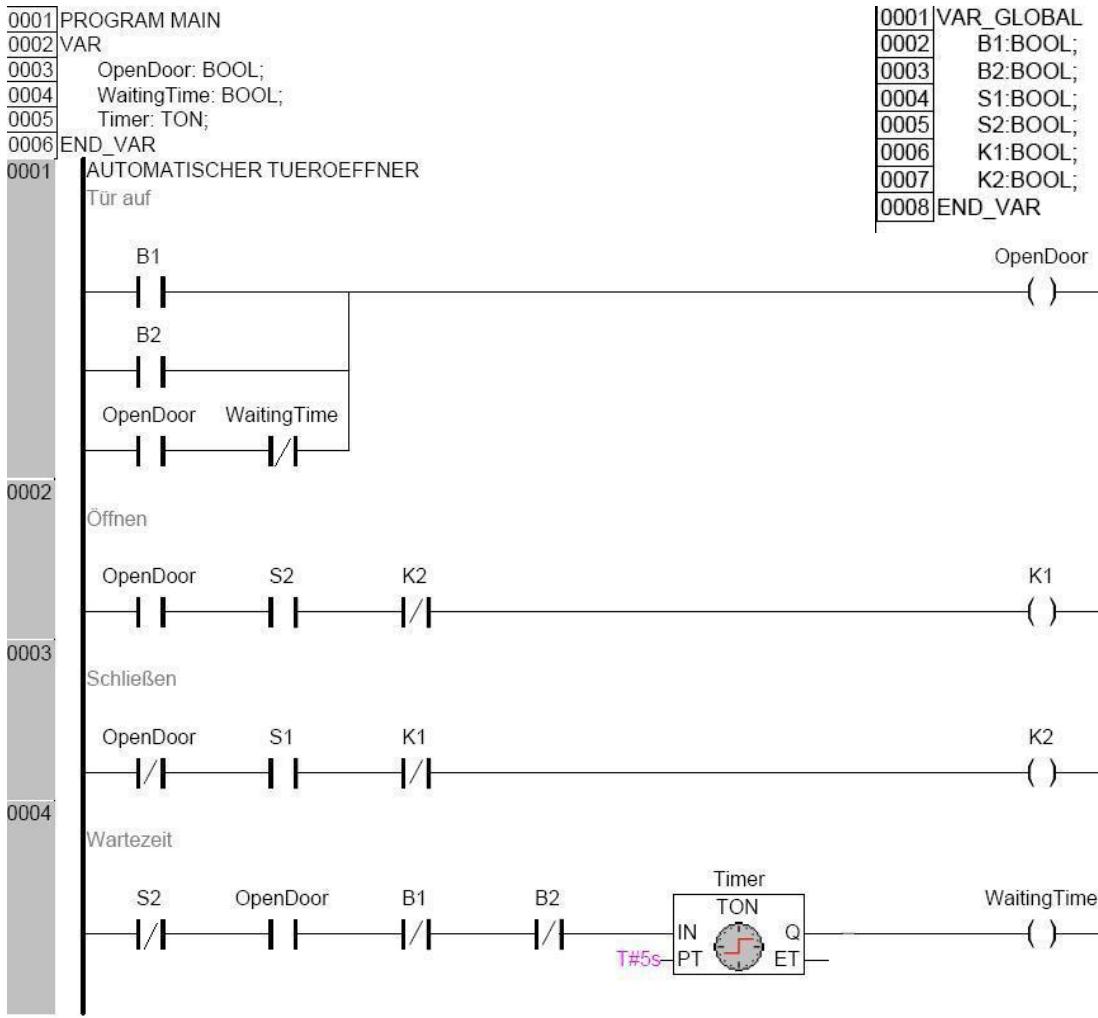
Anwendungsbeispiel: Automatische Tür

Tür	S1	S2
zu	0	1
in Bewegung	1	1
offen	1	0

Funktionsbeschreibung:

- Beim Annähern einer Person soll sich die Tür automatisch öffnen (auch, wenn sie sich gerade schließt).
- Die Tür soll so lange geöffnet bleiben, bis sich keine Person mehr im Durchgang befindet.
- Wenn sich niemand mehr im Durchgang befindet, muss die Tür nach kurzer Wartezeit automatisch schließen.
- Motor mit Rutschkupplung, so dass keine Personen verletzt werden können.

SPS-Programm in KOP (Beckhoff TwinCAT)



Deklaration von *globalen* Variablen (Sensoren/Aktoren) und *lokalen* Variablen (sonstige inkl. FBs wie Timer)

Netzwerk 1 (Tür auf) aktiviert durch Bewegungsmelder als Selbstthalteschaltung bis Wartezeit abgelaufen ist

Netzwerk 2 (Öffnen) u. 3 (Schließen) gegenseitig verriegelt über K1, K2 (hier überflüssig, da Exklusivität mittels *OpenDoor* bereits gesichert)

Netzwerk 4 (Wartezeit) über Zeitglied (Einschaltverzögerung) realisiert, startet bei jedem Ansprechen der Bewegungsmelder neu

Allgemeine Bemerkungen zum Programm

KOP

Darstellung der Programmteile als parallele Strompfade aus 'Kontakten' und 'Spulen' (grafische Programmierung, ähnlich konventioneller Schützsteuerung)



Schließer, unbetätigt bzw. Öffner, betätigt:
ein TRUE-Signal wird als TRUE verknüpft
ein FALSE-Signal als FALSE



Öffner, unbetätigt bzw. Schließer, betätigt:
das Eingangssignal wird umgekehrt:
ein TRUE-Signal wird als FALSE-Signal verknüpft
ein FALSE-Signal als TRUE



Signal-Ausgang:
Bei Ansteuerung mit TRUE-Signal gibt der Signal-Ausgang ein
TRUE-Signal nach außen ab. Dieses Symbol wird immer am
Ende eines Strompfades gezeichnet. Es schließt den Strompfad
ab (Ende der Sequenz).

Signalausgang ('Spule')

schaltet (Wert TRUE), wenn die vorangehenden Kontakte (Schließer oder Öffner) im Strompfad so geschaltet sind, dass ein Strom durch sie fließt.

Der *Signalausgang* kann

- eine *Ausgabe* (externes Signal)
- einen *Merker* (internes Signal)

darstellen.

Der *Signaleingang* kann

- eine *Eingabe* (externes Signal)
- einen *Merker* (internes Signal)

darstellen.

TwinCAT: Deklaration von binären Eingabe/Ausgabesignalen als lokale oder globale Variable (Typ BOOL).

Deklaration von Merkern als lokale Variable (Typ BOOL).

Funktionsbausteine müssen ebenfalls deklariert werden (z. B. Timer: TON).

Tritt der Bezeichner eines Ausgangs bei einem Eingang auf, erhalten beide den gleichen Wert. (Spule bewirkt Schalten des Kontaktes bei Schützsteuerungen).

Bem.: Durch die zyklische Abarbeitung wird der Eingabewert in der SPS aber erst im *nächsten* Zyklus gesetzt.

Dadurch Verkoppelung der Strompfade (inkl. Rückkopplung wie in Netzwerk 1).

In den Strompfad können auch Funktionsbausteine (Flipflops, Zeitglieder etc.) eingeschaltet sein.

Nebenläufige Vorgänge können leicht durch parallele Strompfade ausgedrückt werden.

Sequentielle Vorgänge müssen durch zeitweises Sperren einzelner Strompfade durch Merker realisiert werden (unübersichtlich!).

Verknüpfungssteuerung: Den Eingangssignalen werden durch die Verknüpfungslogik kontinuierlich, d. h. in jedem Zyklus, bestimmte Ausgangssignale zugeordnet.

Bindung von Variablen an physikalische Adressen (Sensoren/Aktoren)

VAR_GLOBAL

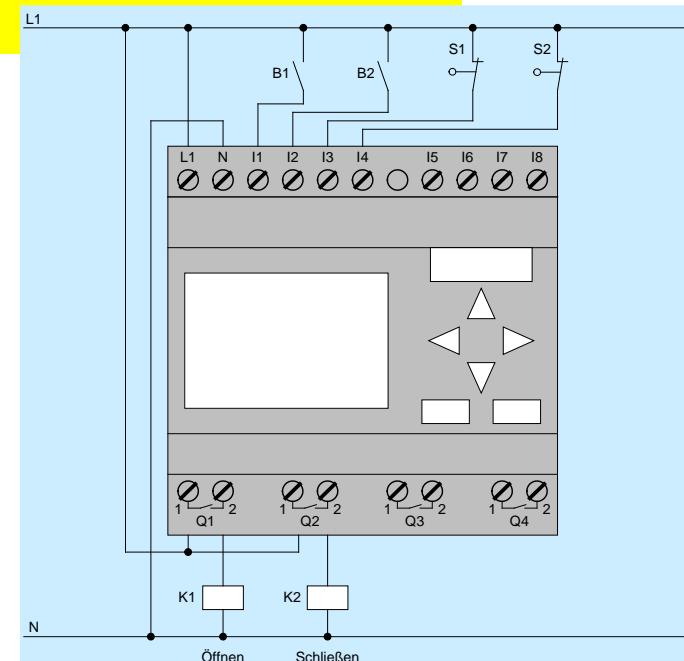
B1 AT	%I0.1: BOOL;	(*Bewegungsmelder außen*)
B2 AT	%I0.2: BOOL;	(*Bewegungsmelder innen*)
S1 AT	%I0.3: BOOL;	(*Endschalter Tür zu (Öffner!)*)
S2 AT	%I0.4: BOOL;	(*Endschalter Tür auf (Öffner!)*)
K1 AT	%Q0.1: BOOL;	(*Hauptschütz Tür öffnen*)
K2 AT	%Q0.2: BOOL;	(*Hauptschütz Tür schließen*)

END_VAR

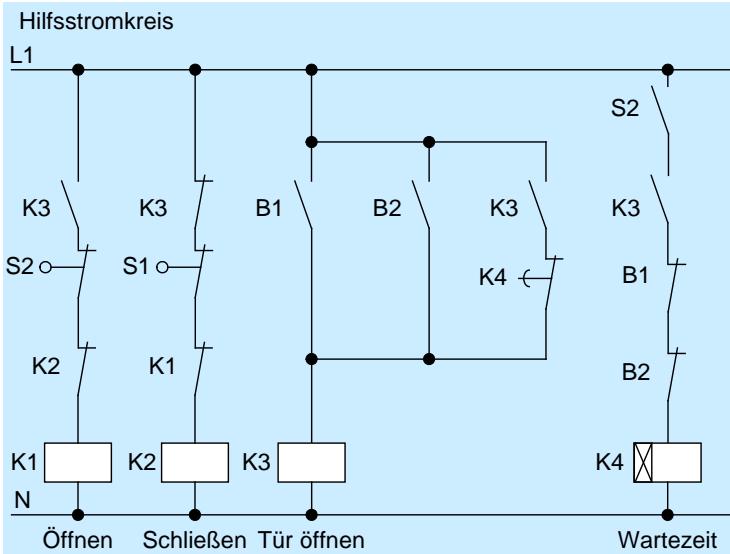
Eingangsbezeichner **%Ix.y** und Ausgangsbezeichner **%Qx.y** bezeichnen die physikalischen Adressen der binären Ein- bzw. Ausgänge (x Bytenr., y Bitnr.).

Zuordnung zu globalen oder lokalen Variablen mittels Attribut **AT**.

Festlegung der physikalischen Adressen bei der Konfiguration der Steuerung (Bei Beckhoff-Busklemmen in aufsteigender Reihenfolge je nach Abstand zum Buskontroller).



Konventionelle Schützschaltung für Türsteuerung



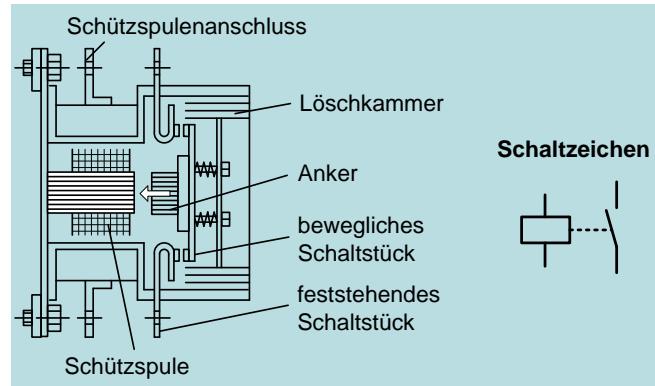
Sobald einer der Bewegungsmelder B1 oder B2 eine Person erfasst, wird über K3 das Öffnen der Tür eingeleitet.

Wenn der Erfassungsbereich der beiden Bewegungsmelder für eine Mindestzeit frei ist, gibt K4 den Schließvorgang frei.

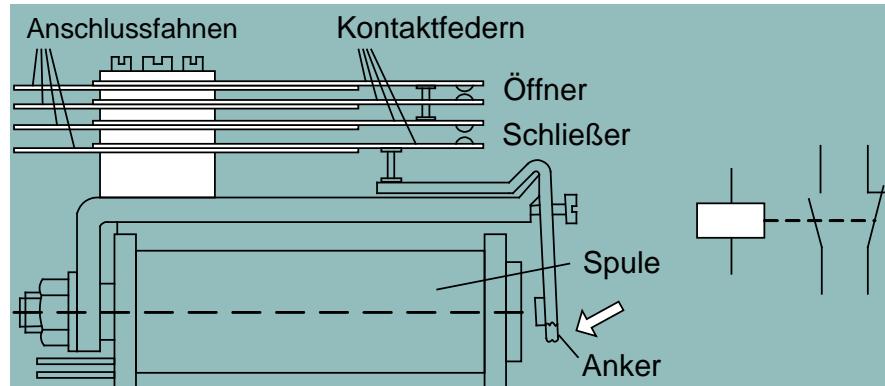
K3 Hilfsschütz

K4 Hilfsschütz mit Anzugsverzögerung (Timer)

Schütz (1 - 500 kW)

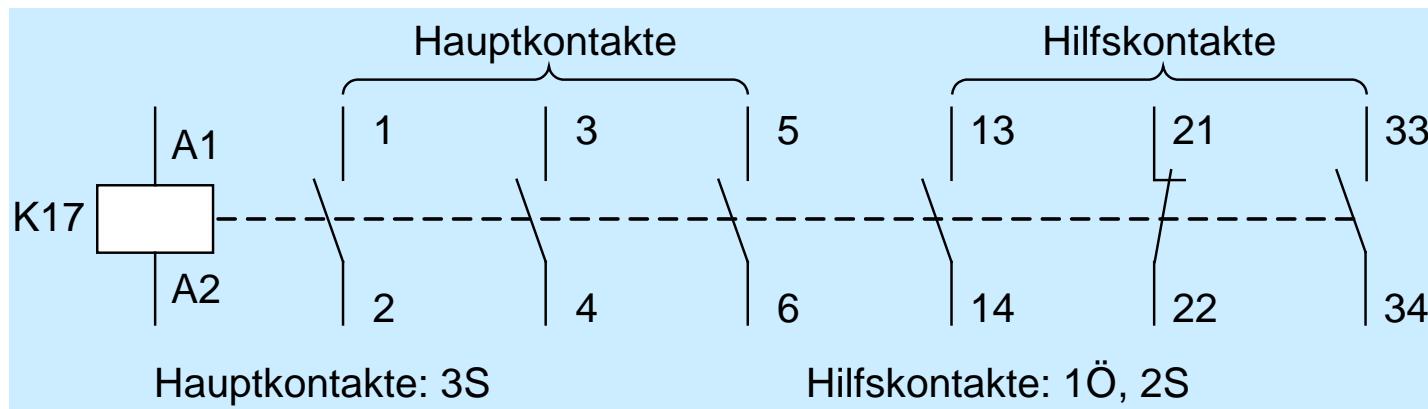


Relais (1mW - 1kW)



Schütz mit Haupt- und Hilfskontakten

Schaltsymbol



Hauptkontakte für Verbraucher, Hilfskontakte zur Steuerung

SPS-Programm in AWL (TwinCAT)

Netzwerk 1

(*Tür auf*)

```
LD      OpenDoor  
ANDN   WaitingTime  
OR     B1  
OR     B2  
ST      OpenDoor
```

Netzwerk 3

(*Schließen*)

```
LDN    OpenDoor  
AND    S1  
ANDN   K1  
ST     K2
```

Netzwerk 2

(*Öffnen*)

```
LD      OpenDoor  
AND    S2  
ANDN   K2  
ST     K1
```

Netzwerk 4

(*Wartezeit*)

```
LDN    S2  
AND    OpenDoor  
ANDN   B1  
ANDN   B2  
ST     Temp  
CAL    Timer (IN:=Temp, PT:=T#5s)  
LD     Timer.Q  
ST     WaitingTime
```

Akkumulatormaschine

Einzelne Strompfade werden durch Anweisungsblöcke in AWL dargestellt (sequentielle Ausführung).

Verknüpfung der Eingangssignale nacheinander im Akkumulator (logische Operationen LD, OR, AND, ORN, ANDN etc.).

Ergebnis geht an Ausgangssignal oder Merker (Operation ST).

Akkumulatormaschine, d. h. immer nur ein Operand kann in den Akku geladen (LD) und dort verknüpft werden, danach speichern (ST).

Negation des Operanden durch Zusatz N möglich, (z. B. LDN, STN).

Aufruf von Funktionsbausteinen mit CAL, ggf. Parameterübergabe (siehe Manual) für Eingaben.

Ausgaben von FBs als Variable mit Name des Bausteins als Qualifizierer abfragbar (z. B. Timer.Q).

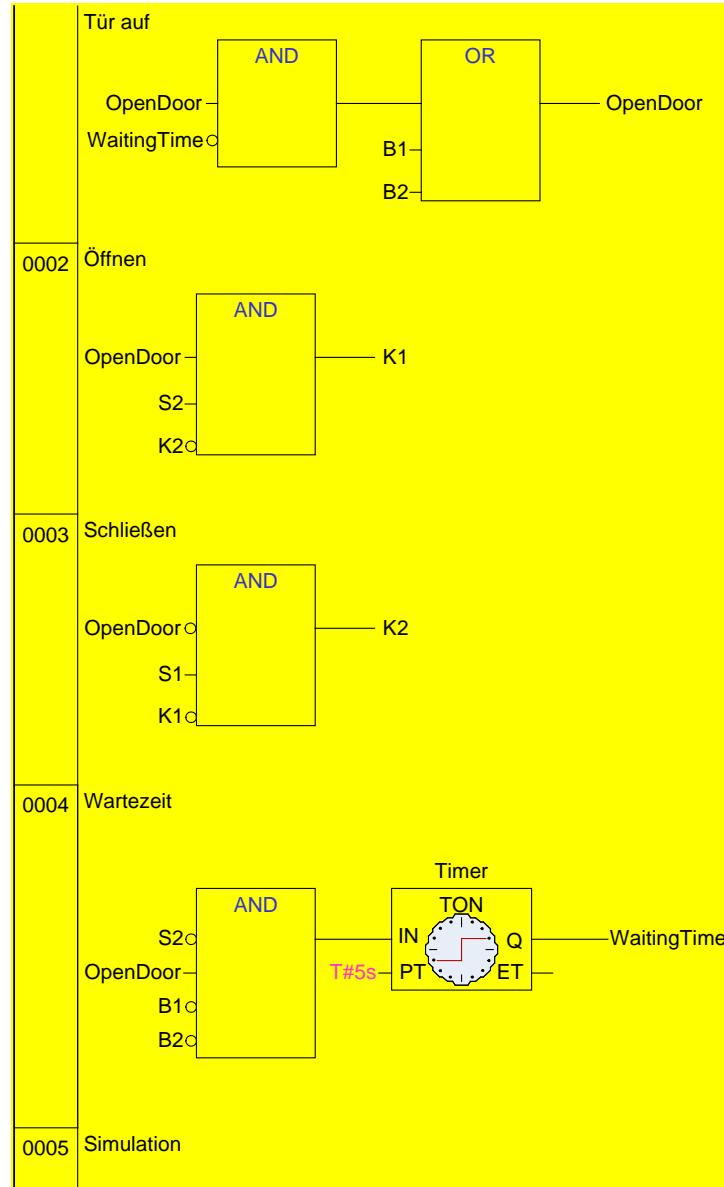
Verkopplung der einzelnen AWL-Blöcke durch Variable (siehe KOP).

Trotz sequentieller Ausführung der AWL-Anweisungen konzeptionell nebenläufige Abarbeitung der AWL-Blöcke (entsprechend Strompfaden in KOP), da permanent zyklische Abarbeitung des ganzen Programms.

Steuerung sequentieller Abläufe über Variable (siehe KOP).

AWL ist damit eine Art textuelle Darstellung von KOP, wird auch als ‚Maschinensprache‘ der SPS bezeichnet (aber nicht identisch mit Maschinensprache des Mikrocontrollers!).

SPS-Programm in FUP (TwinCAT)



FUP/FBS

Einzelne Strompfade (Netzwerke) werden durch logische Schaltungen mit *Gattern* grafisch dargestellt (UND, ODER etc.).

Negationen an Eingängen und Ausgang möglich (dargestellt durch kleinen Kringel).

Ausgänge können entweder Aktoren betätigen oder über Variablen Eingänge für andere Netzwerke sein. Außer Gattern auch komplexere Funktionsbausteine für Timer, Zähler, Flipflops etc. möglich.

Nebenläufigkeit durch parallele Netzwerke (vgl. Hardwareschaltungen).

Sequentieller Ablauf über Variablen erreichbar (siehe KOP, AWL).

Alternative graphische Darstellung zu KOP (kann bei manchen Umgebungen sogar automatisch umgeschaltet werden).

SPS-Programm in ST (TwinCAT)

(* Tür auf *)

```
IF (OpenDoor AND NOT WaitingTime) OR B1 OR  
B2 THEN
```

```
    OpenDoor:=TRUE;
```

```
ELSE
```

```
    OpenDoor:= FALSE;
```

```
END_IF;
```

(* Öffnen *)

```
IF OpenDoor AND S2 AND NOT K2 THEN
```

```
    K1:= TRUE;
```

```
ELSE
```

```
    K1:= FALSE;
```

```
END_IF;
```

(* Schließen *)

```
IF NOT OpenDoor AND S1 AND NOT K1  
THEN
```

```
    K2:=TRUE;
```

```
ELSE
```

```
    K2:=FALSE;
```

```
END_IF;
```

(* Wartezeit *)

```
IF NOT S2 AND OpenDoor AND NOT B1  
AND NOT B2 THEN
```

```
    Temp:=TRUE;
```

```
ELSE
```

```
    Temp:=FALSE;
```

```
END_IF;
```

```
Timer(IN:=Temp, PT:=T#5s);
```

```
WaitingTime:=Timer.Q;
```

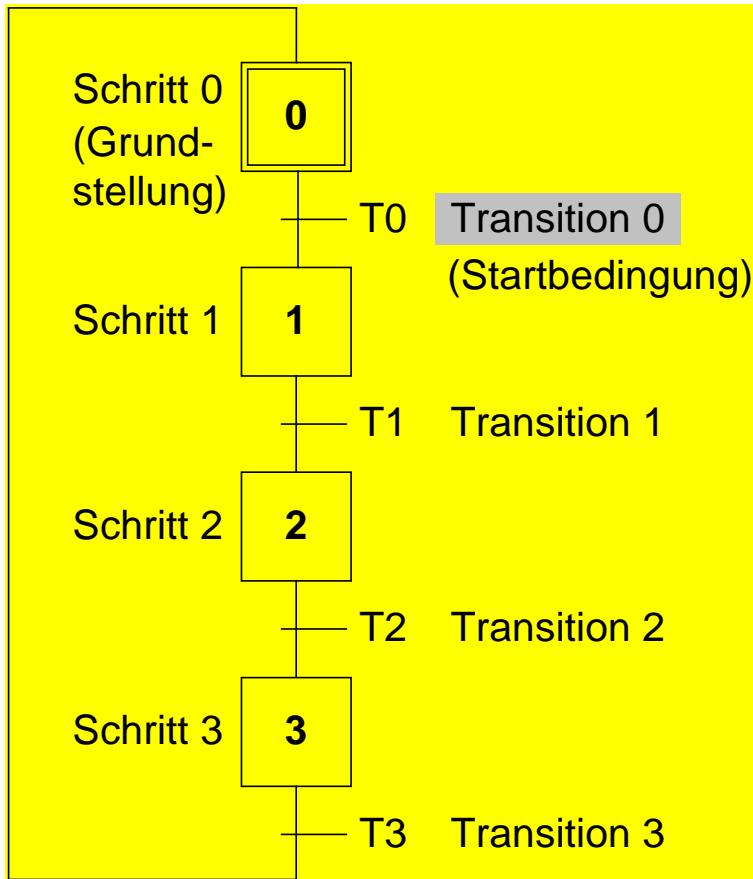
Einzelne Netzwerke durch *IF-THEN-ELSE-Regeln* darstellbar. Alternativ auch *Boolesche Wertzuweisungen* möglich, z. B. K1 := OpenDoor AND S2 AND NOT K2;

Aufruf von Funktionsbausteinen als Prozedur mit Eingangsparametern (siehe Handbuch). Ausgangsparameter mit Name des Funktionsbausteins als Qualifizierer abrufbar (vgl. AWL).

Pascal-artige höhere Programmiersprache. Anweisungen werden zwar sequentiell in der Programmreihenfolge angegeben, letzteres wird aber permanent zyklisch ausgeführt.

2.3.5 Fortgeschrittene SPS Programmierung (Ablaufsteuerungen)

Programmablaufplan (nach IEC 1131-3)

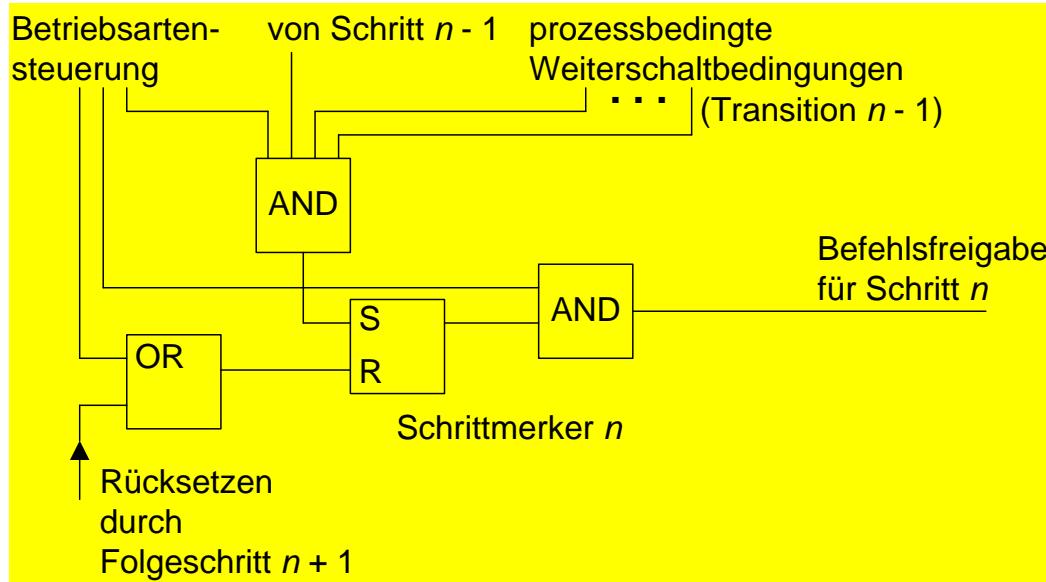


Graphische Darstellung von *Schritten* (*Steuerungsbefehle*) und *Weiterschaltbedingungen* (*Transitionen*) in einem Programmablaufplan (vgl. Petri-Netze).

Einzelnen Schritte z. B. in KOP, AWL, FUP oder ST programmierbar.

Schritte können auch in AS programmiert werden, d. h. Schachtelung von Programmablaufplänen möglich.

Realisierung der Schrittfortschaltung mit SR-Flipflop in FUP



Weiterschaltbedingungen bewirken Setzen des SR-FFs und damit Aktivierung des Schrittes.

Rücksetzen (Deaktivierung) durch Folgeschritt.

Zusätzliche Aktivierung/ Deaktivierung durch Betriebsartensteuerung möglich (z. B. manueller Einzelschrittbetrieb).

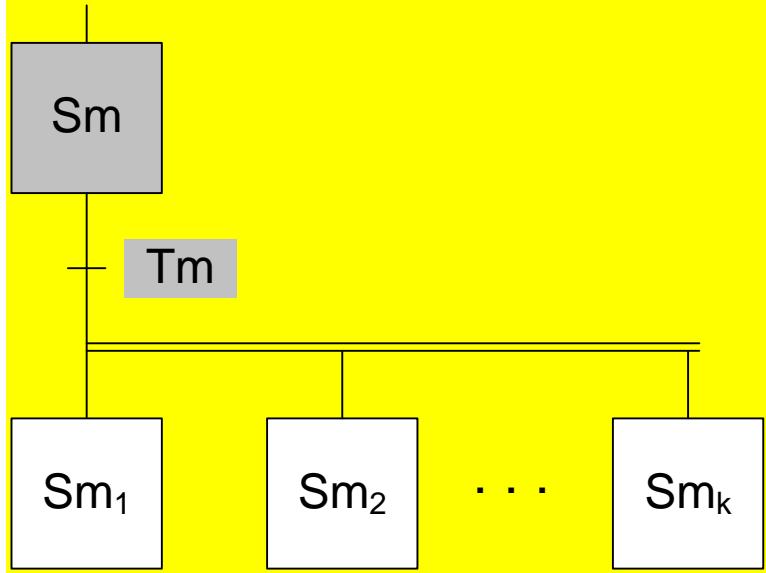
KOP, AWL ganz entsprechend.

Strukturierte Programmierung

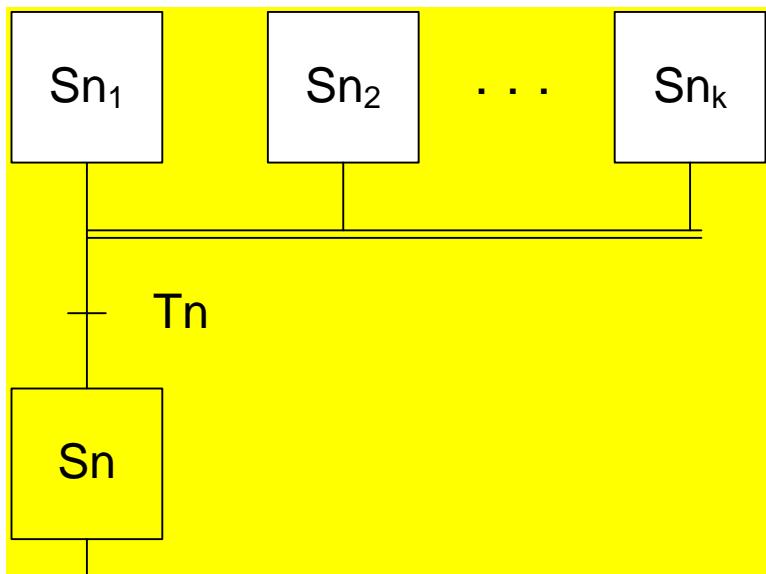
Vor allem Ablaufsteuerungen mit mehreren Zeitschritten werden in AWL, KOP oder FUP schnell unübersichtlich.

⇒ Funktionale Gliederung des SPS-Programms in Funktionsbausteine (FBs) für Betriebsarten, Ablaufkette, Steuerbefehle und Meldungen.

Prinzipielle Strukturen von Ablaufsteuerungen



Simultanverzweigung ('*Fork*') paralleler Ablaufketten

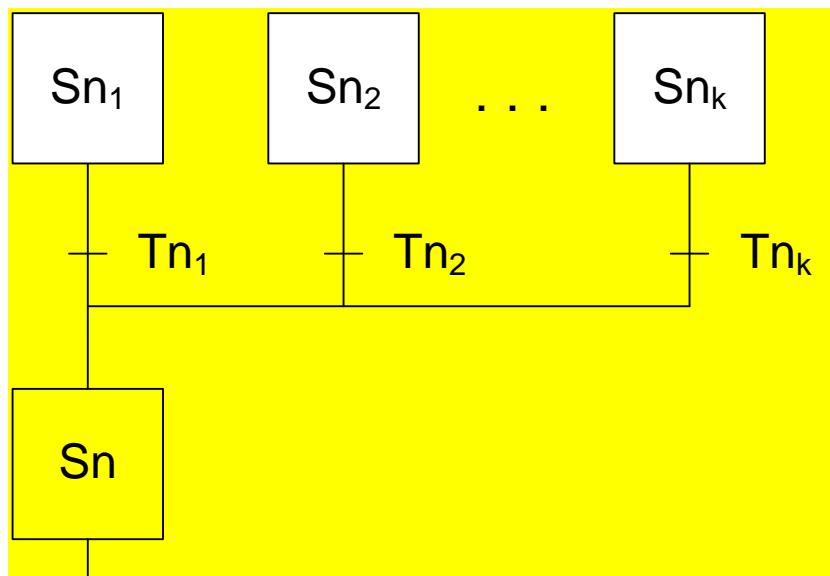
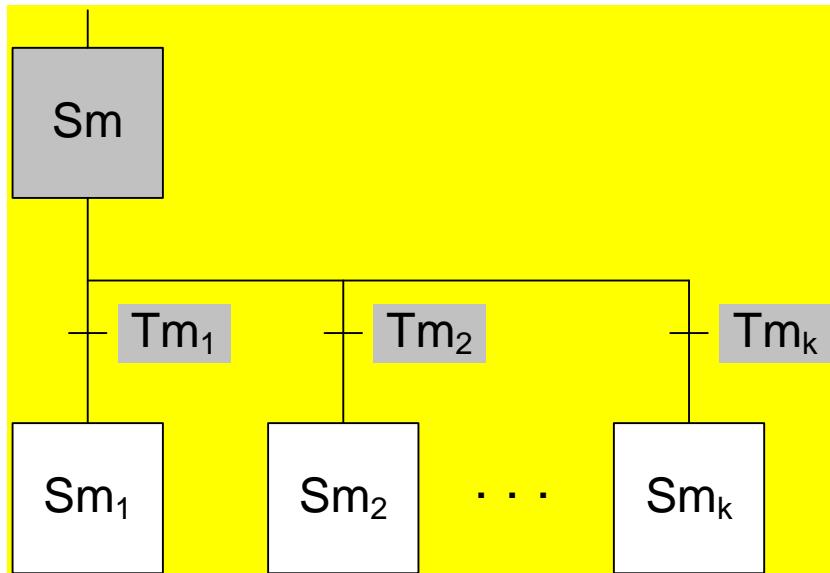


Alle k Zweige werden parallel ausgeführt!

Synchronisation ('*Join*') paralleler Ablaufketten)

Erst wenn *alle k parallelen Zweige* terminiert haben und Tn erfüllt ist, wird mit Sn fortgefahrene

Alternativverzweigung und -zusammenführung ('Case'-Mehrwegeverzweigung)

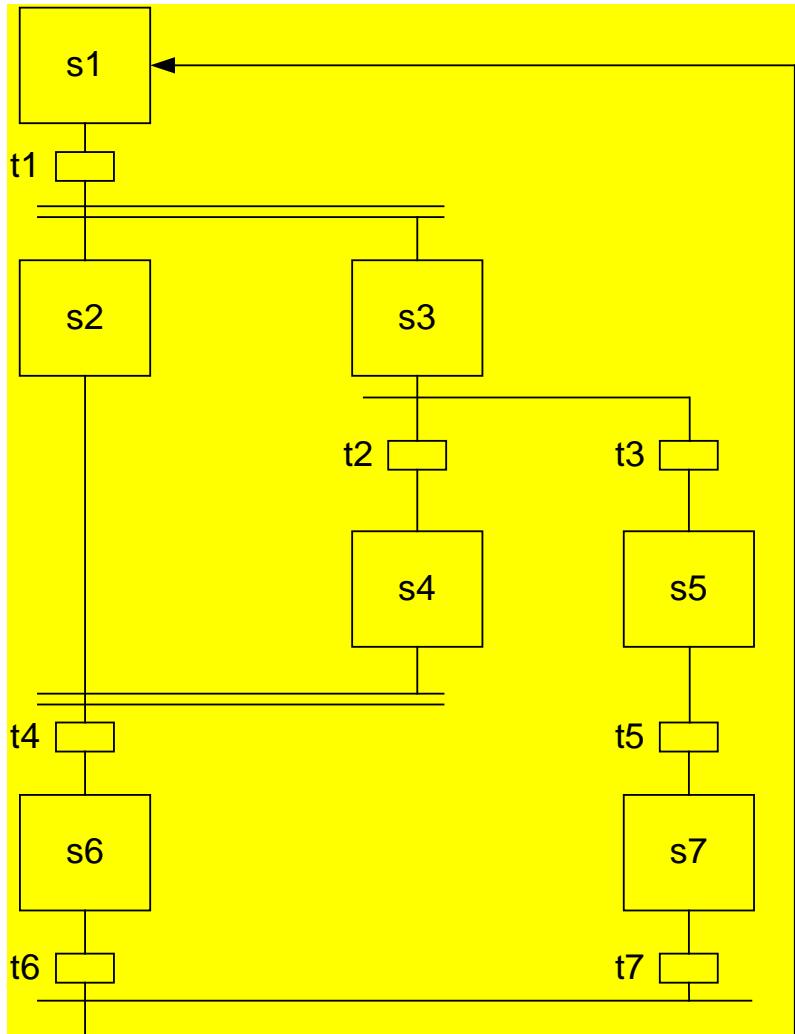


Nur *ein* Zweig wird ausgeführt!

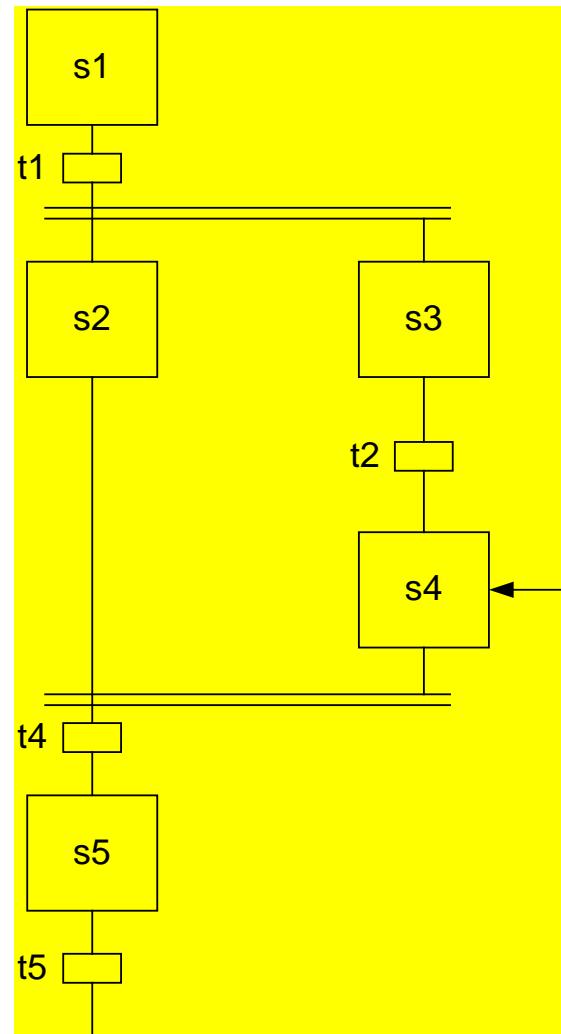
Exklusivität der Bedingungen

Tm_1, Tm_2, \dots, Tm_k muss vom Programmierer sichergestellt werden.

Unsichere und unerreichbare Ketten



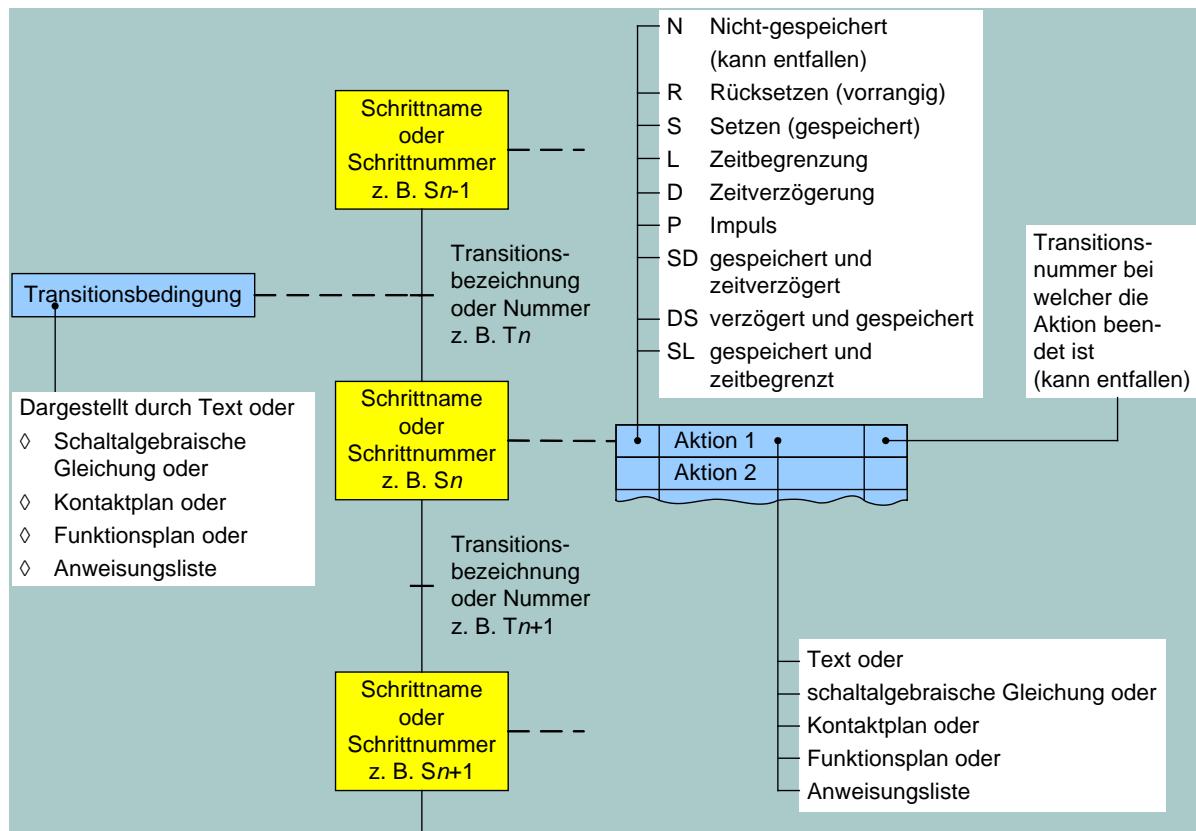
Erreichbarkeit von s6 unsicher!



Verklemmung (Deadlock)
nach Rücksprung in s4!

Komplexere Ablaufsteuerungen sind aufgrund der Speicher und Zustände in KOP, FUP, AWL schwer durchschaubar. Daher grafische Programmiersprache AS, in der die Ablaufsteuerung unmittelbar als Ablauf-Funktionsplan formuliert werden kann.

Schritte und Transitionen in AS mit (optionalen) Aktionsattributen



Ablaufschritte (Zustände), dargestellt durch Kästchen.

Aktionen, die in dem Schritt ausgeführt werden.

Transitionen
(Weiterschaltbedingungen), die jeweils den nächsten Schritt freigeben.

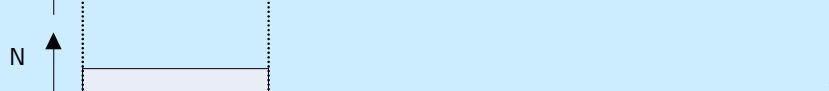
Transitionen und Schritte auch in AWL, FUP KOP oder ST programmierbar.

Bestimmungszeichen der Aktionsattribute

SCHRITTNAME .



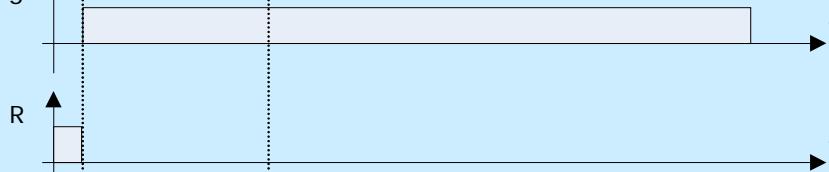
nicht speichernd



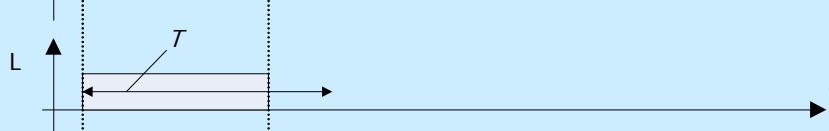
speichernd bis Rücksetzung



Rücksetzen eines gesetzten Signals



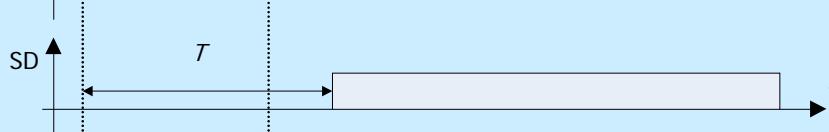
zeitlich limitiert um T#xs, solange Schritt noch aktiv



zeitlich um T#xs verzögert (delayed)



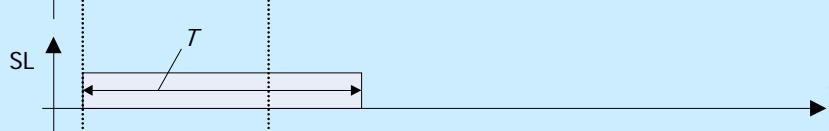
verzögert speichernd, auch wenn Schritt nicht mehr aktiv



verzögert speichernd nur wenn Schritt noch aktiv



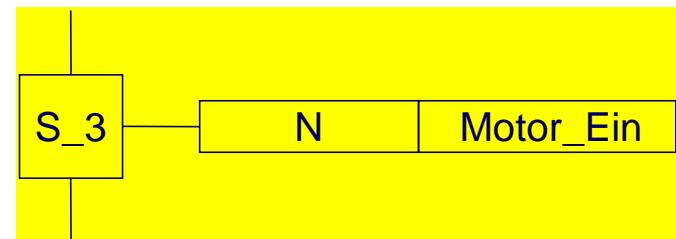
limitiert setzen, auch wenn Schritt nicht mehr aktiv



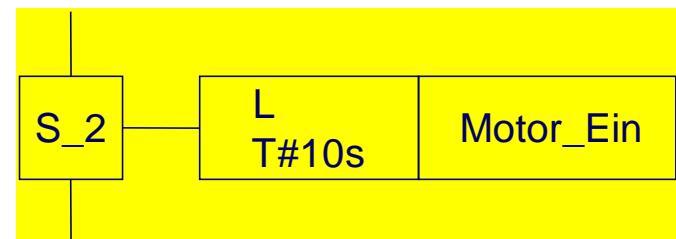
Puls (nur 1 Zyklus)



Beispiel:

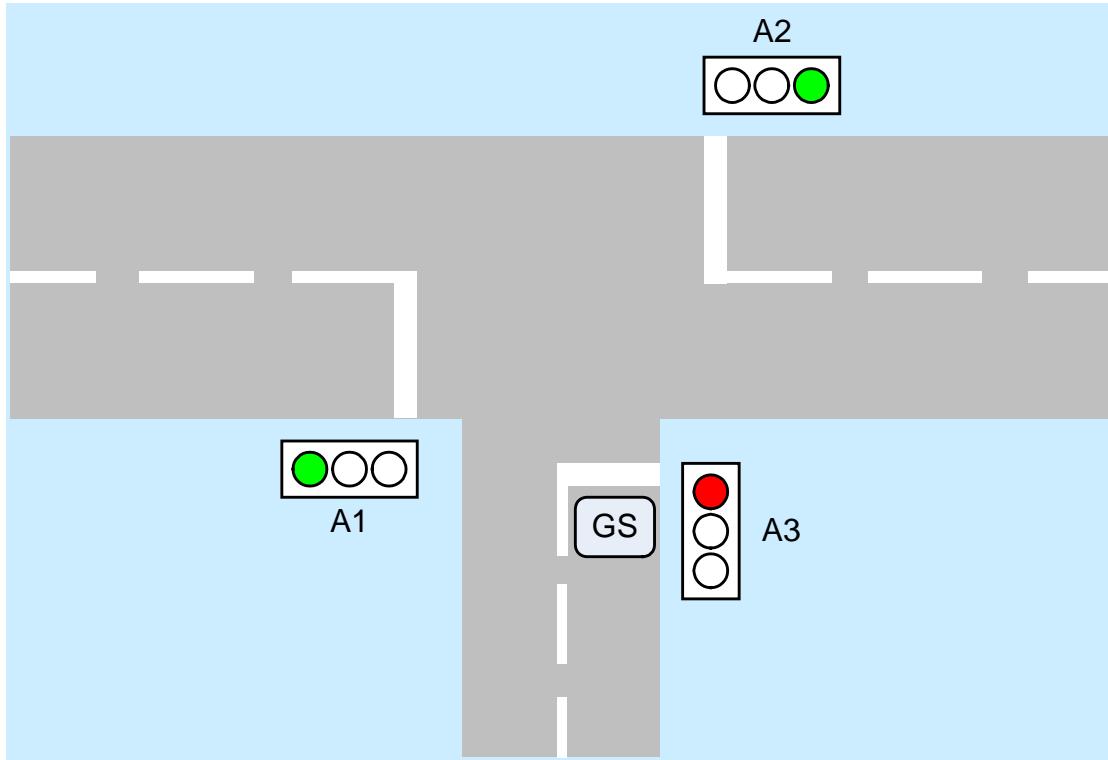


Motor wird für die **gesamte Dauer** des Schrittes eingeschaltet und bei Schrittende ausgeschaltet (häufigstes Bestimmungszeichen).



Motor wird in Schritt S_2 **für 10 s** eingeschaltet. Ist der Schritt kürzer, nur bis zum Schrittende.

Beispiel: Ampelsteuerung



Timing:

A1, A2 dürfen nur dann grün zeigen, wenn A3 mindestens 15 s rot war und umgekehrt.

Grünphase generell 60 s.

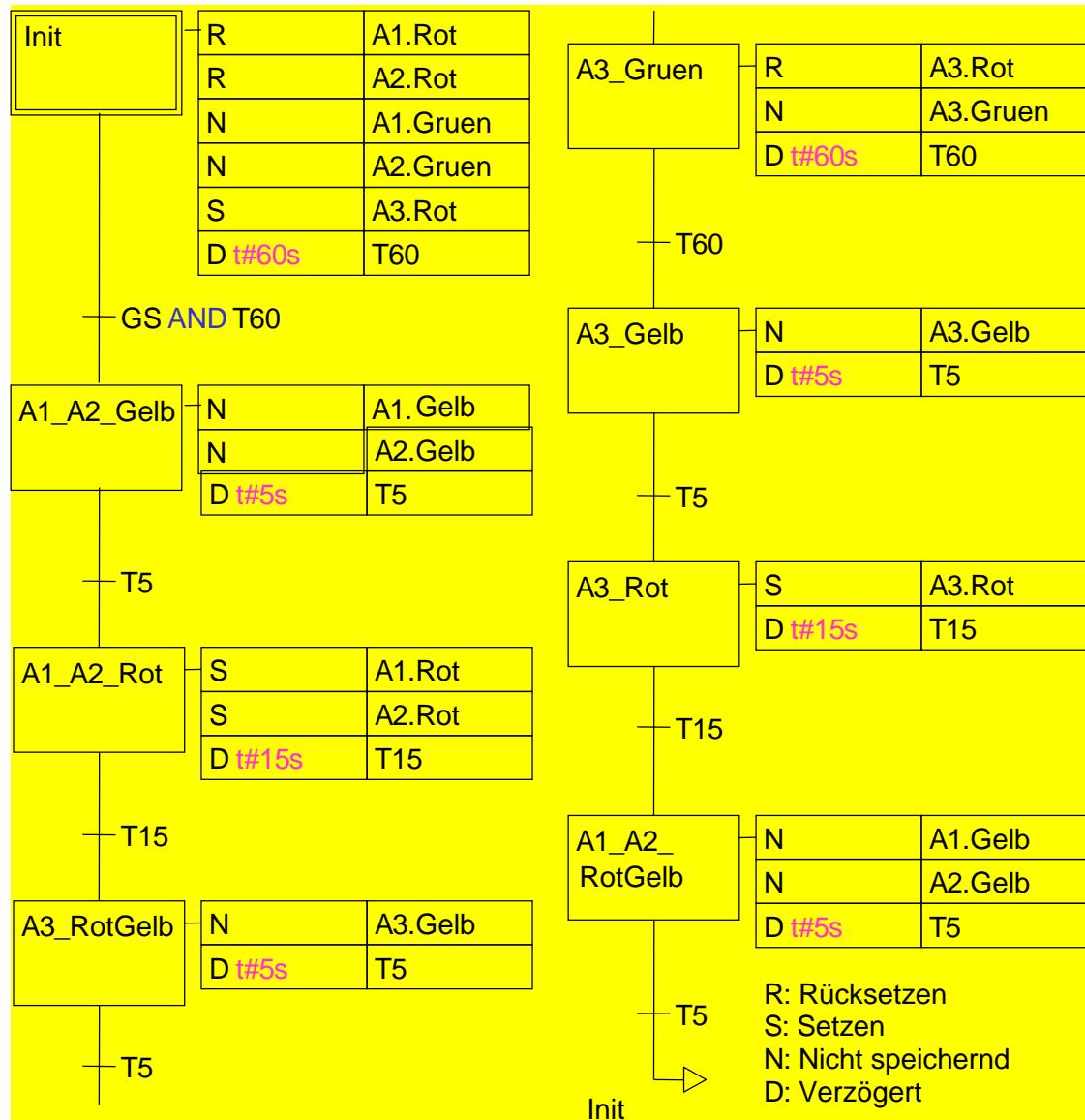
Umschalten über Grün nach Rot und umgekehrt über Gelbphase von 5 s.

Bedarfssampel für Nebenstraße

Grundzustand: A1 und A2 grün, A3 rot

Auto auf Induktionsschleife GS: A3 Grün, A1, A2 rot

Ablaufkette in AS



```

TYPE Ampeldaten :
STRUCT
  Rot: BOOL;
  Gelb: BOOL;
  Gruen: BOOL;
END_STRUCT
END_TYPE
  
```

```

PROGRAM Ampelsteuerung
VAR
  
```

```

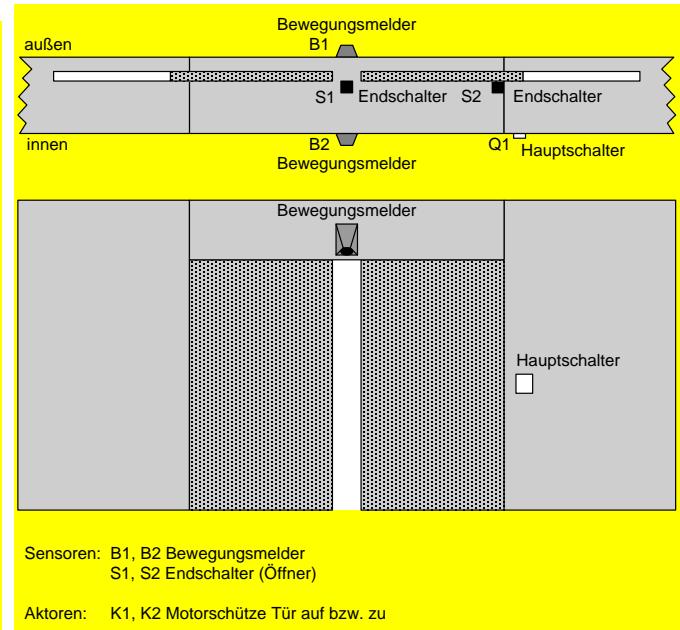
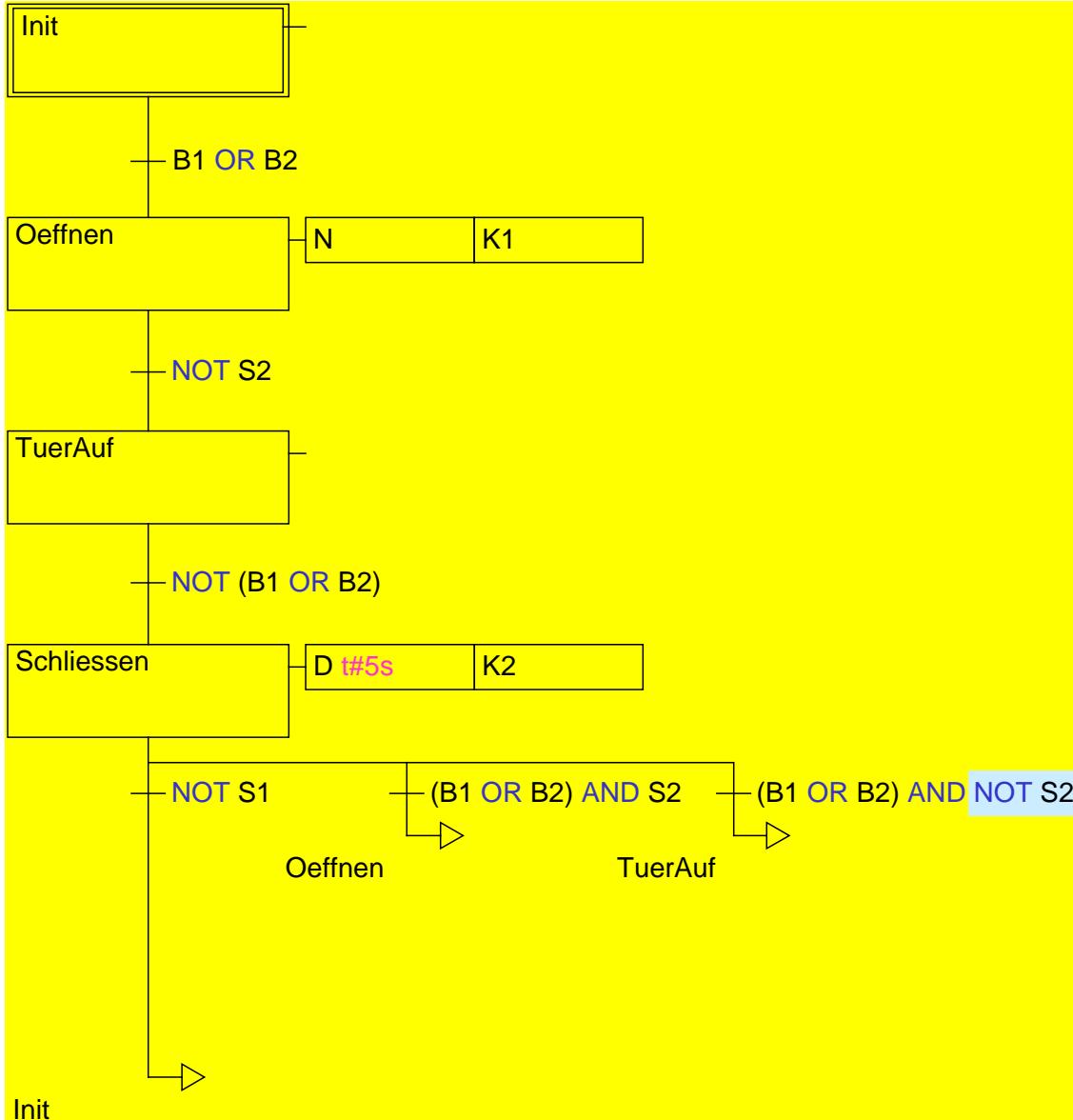
  GS: BOOL;
  A1: Ampeldaten;
  A2: Ampeldaten;
  A3: Ampeldaten;
  T10: BOOL;
  T15: BOOL;
  T60: BOOL;
  T5: BOOL;
  
```

```

END_VAR
  
```

*Zeitabhängige Steuerung:
Schrittfortschaltung hängt von
Zeitvorgaben ab.*

Beispiel: Türsteuerung in AS (vgl. Abschnitt 2.2.4)



Gleiches Verhalten wie Realisierung in KOP, FUP etc.

Übersichtlichere Darstellung (höhere Abstraktionsebene)

Prozessabhängige Steuerung: Schrittfortschaltung hängt vom Prozesszustand ab (Sensorwerte).

Aktionszuordnung zu Schritten

Neben der Angabe von Aktionen mittels Attributen (‘IEC-Schritte’) können Aktionen auch in einer der IEC-Sprachen (KOP, FUP, AWL, ST, AS) formuliert und den Schritten zugeordnet werden.

Dabei ist zu unterscheiden zwischen

- *Aktionen*, die in jedem Zyklus ausgeführt werden, solange der Schritt aktiv ist.
- *Eingangsaktionen*, die nur einmal bei Betreten des Schrittes ausgeführt werden.
- *Ausgangsaktionen*, die nur einmal bei Verlassen des Schrittes ausgeführt werden.

Damit sehr flexible und mächtige Programmierung von Schrittketten möglich.

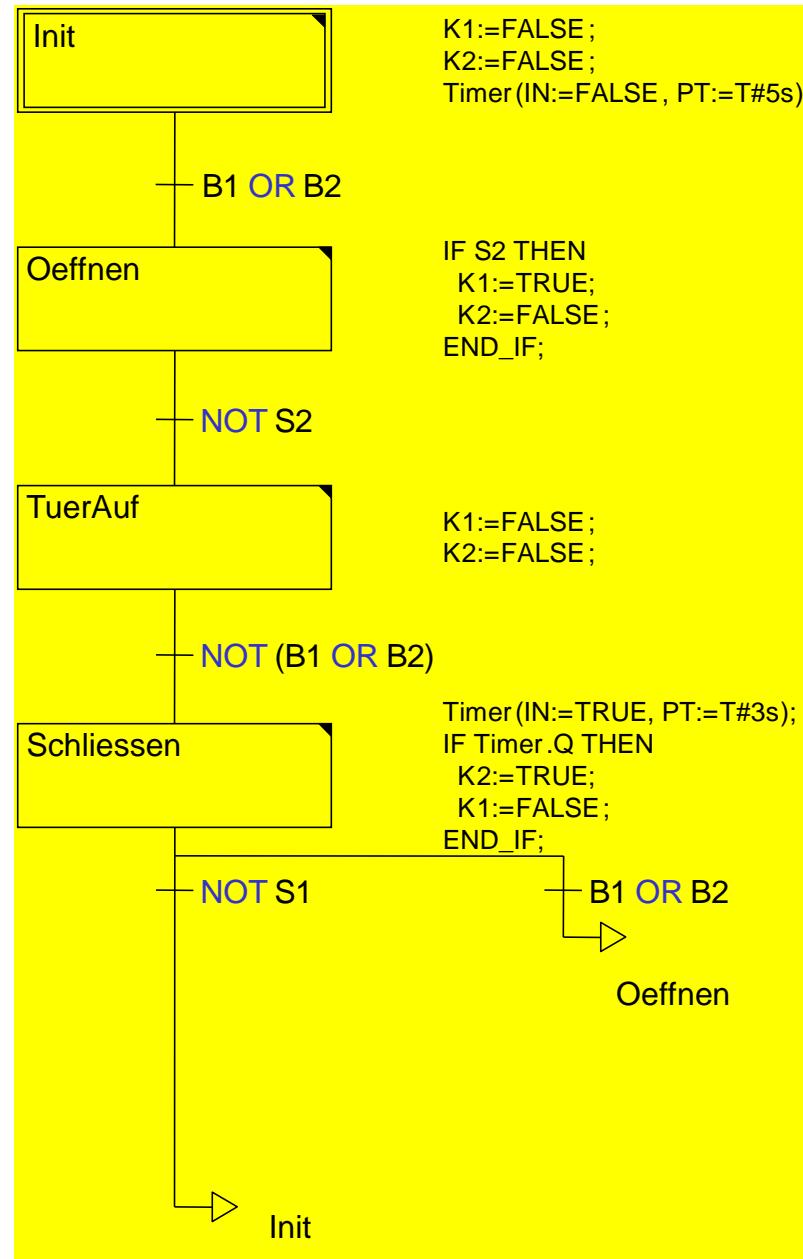
Aber: weniger anschaulich, da Aktionen erst nach Anklicken des Schrittes sichtbar (im Schrittsymbol nur kleines schwarzes Dreieck in rechter oberer Ecke als Markierung, die darauf hinweist, dass der Schritt Aktionen enthält).

Transitionen

können auch aus einer Folge von Instruktionen mit einem Booleschen Ergebnis in ST oder einer anderen IEC-Sprache enthalten, dürfen aber keine Programme, Funktionen oder Funktionsblöcke enthalten.

Beispiel: Türsteuerung

```
PROGRAM Control
VAR
    Timer: TON;
END_VAR
```



Einfachere und klarere Struktur, dafür komplexere Operationen innerhalb der Zustände (hier in ST).