Prozessautomatisierung

Prof. Dr.rer.nat. Horst Hahn

geschrieben: Oswald, Michael
Siepchen, Benedict
Erdmann, Dominik

WS04/05
SS05

Inhaltsverzeichnis

1 Einleitung

Was ist Automatisierungstechnik?

1.1 Geschichtliche Entwicklung der Automatisierungstechnik

Die Entwicklung der Automatisierungstechnik verläuft im 20. Jahrhundert in mehrere Phasen, die durch Neuentwicklungen auf den Gebieten der

- Elektrotechnik
- Elektronik
- Halbleitertechnologie

eingeleitet wurden.

Beginn des 20. Jahrhunderts:

- Entwicklung von industriellen Produktionsverfahren
- wachsendes Produktionsvolumen

Dies führte zu der Forderung

- einer genaueren Überwachung der Produktionsanlagen
- und einer zunehmenden Steuerung ihrer Betriebsweise

Typische Merkmale dieser anfänglichen industriellen Automatisierungstechnik (1920):

- im Feld (vor Ort) angeordnete Anzeige- und Bedieneinrichtungen
- Anlagenpersonal beobachtet und bedient unmittelbar vor Ort

Zunehmende räumliche Ausdehnung der Anlagen führen verschärfte Forderung bzgl.:

- Erhöhung der Anlagenausnutzung
- Steigerung der Produktqualität

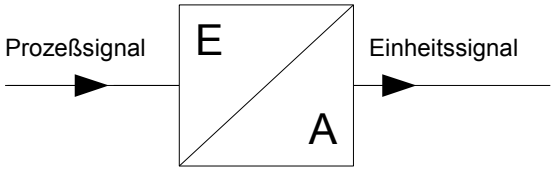
Erreicht wurde dies ab etwa 1940 durch:

- Entwicklung von Reglern für einzelne Prozessgrößen
- Entlastung des Personals durch räumliche Konzentration von Einrichtungen zum Beobachten und Bedienen der Anlage
- Zusammenfassung von Leiteinrichtungen in zentralen Warten
- Fernmess- und Fernstelleinrichtungen

  Beispiel: Anlagen der Treibstoffchemie auf Stein- und Braunkohlebasis

Ab 1950 wurde eine wesentliche Steigerung der Leistungsfähigkeit von Automatisierungsproblemen erzielt durch:

- Einsatz von Röhren- und Halbleiterverstärkern
- Festlegung des Einheitssignals (DIN 19250)
    - 0..20 mA
    - 4..20 mA
    - 0..10 V
    - 0,2..1,0 bar
- Messumformer



Weiterhin wachsende Anlagengrößen mit erweiterter Anforderung an die Anlagenführung und zunehmend an die Schonung von Ressourcen und Umwelt führten zu dem Einsatz digitaler Leit- und Automatisierungseinrichtung.

1960: Einsatz des ersten Prozessrechners in der Industrie

Merkmale:

- geeignet zur Lösung komplexer und umfangreicher Aufgaben
- teilweise unzureichende Echtzeitfähigkeit
- mangelnde Verfügbarkeit und Zuverlässigkeit
- hohe Kosten

Entwicklung der Halbleitertechnologie: Integrationstechnik
IC's: µP, RAM, ROM

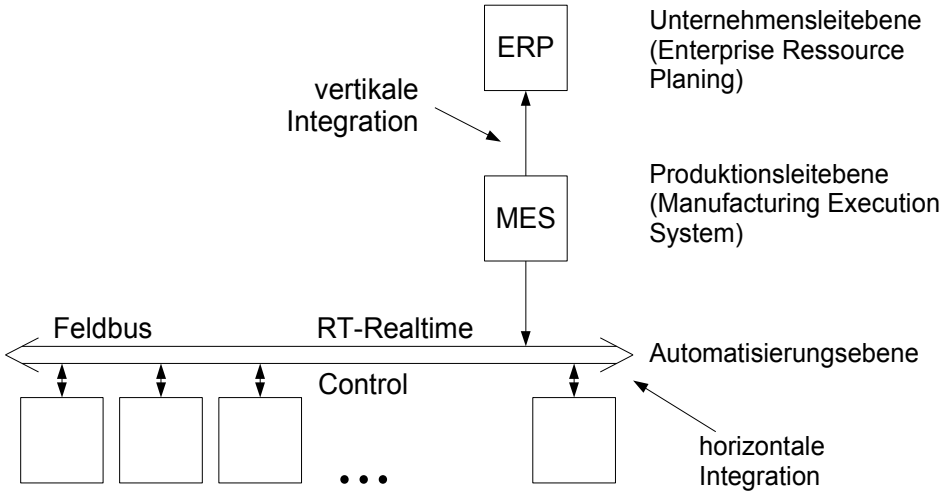
1975: Entwicklung von kompakten und leistungsfähigen Mikrorechnern mit anpassungsfähigen Leistungsvermögen

1980: Einführung von LAN's in der Bürowelt (Ethernet)

1985: Einführung von Feldbussen in der Automatisierungstechnik mit folgenden Merkmalen:

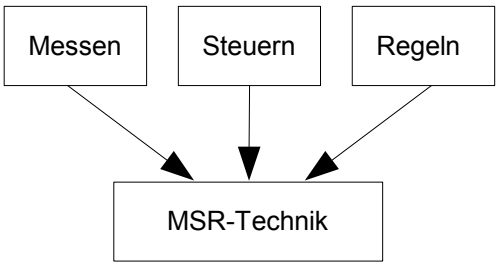
- neue Automatisierungsstrukturen
- Parallelisierung von Teilaufgaben
- bessere Echtzeitfähigkeit
- bessere Verfügbarkeit und Zuverlässigkeit
- geringere Kosten

2000:



1.2 Grundbegriffe der Automatisierungstechnik

typische Funktionen der Automatisierungstechnik:



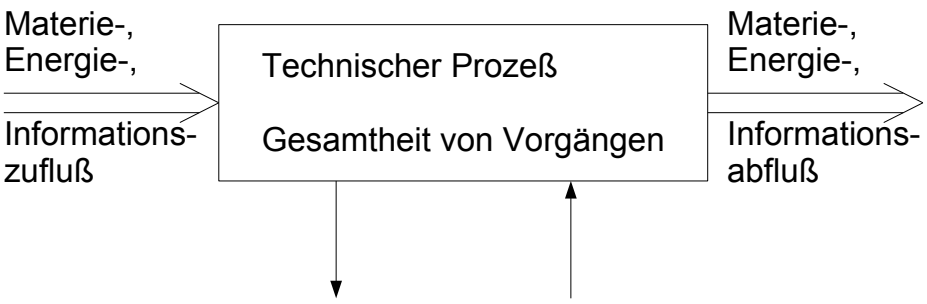
neue Funktionen: Informationen übertragen, verarbeiten und darstellen

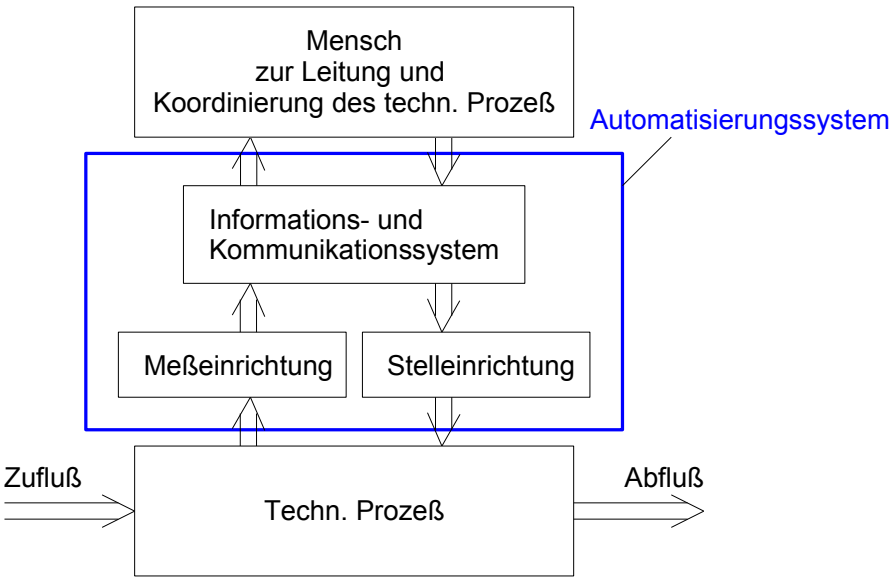
neue Namensgebung: Prozessautomatisierung

Prozessleittechnik
Prozessdatenverarbeitung

Definition von Prozess (DIN 66201):

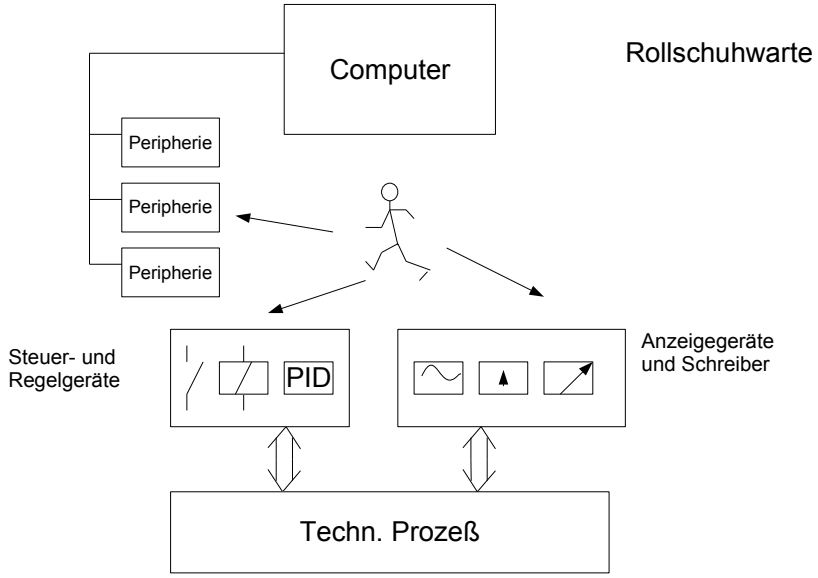
> Ein Prozess ist eine Gesamtheit von aufeinanderfolgenden einwirkenden Vorgängen in einem System, durch die Materie, Energie oder Information umgeformt, transformiert oder gespeichert wird.
>
> Ein technischer Prozess ist ein Prozess, dessen physikalischen Größen mit technischen Mitteln erfasst und beeinflusst werden können.




Mensch
zur Leitung und
Koordinierung des techn. Prozeß
Automatisierungssystem
Informations- und
Kommunikationssystem
Meßeinrichtung
Stelleinrichtung
Zufluß
Techn. Prozeß
Abfluß

2 Einsatzarten von Computern zur Automatisierung technischer Prozesse

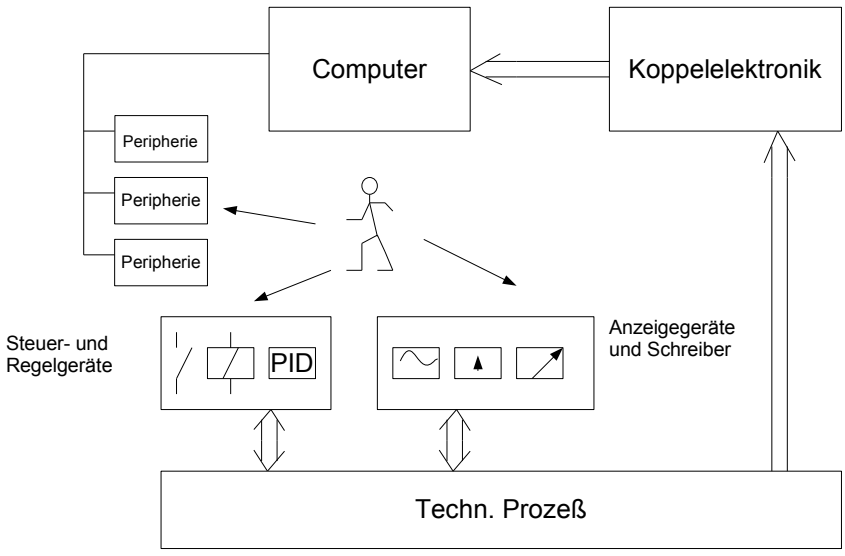
2.1 Geringer Automatisierungsgrad (Offline-Betrieb eines Computers)



Kennzeichen des offline-Betriebs:

- zeitliche und gerätemäßige Entkoppelung von techn. Prozess und Computer
- Der Prozess wird vom Personal gefahren:
  - Messdaten ablesen, dokumentieren
  - Entscheidungen treffen zur Prozessführung
  - zur Unterstützung komplexer Aufgaben Computer benutzen
  - Steuer- und Regelgeräte bedienen

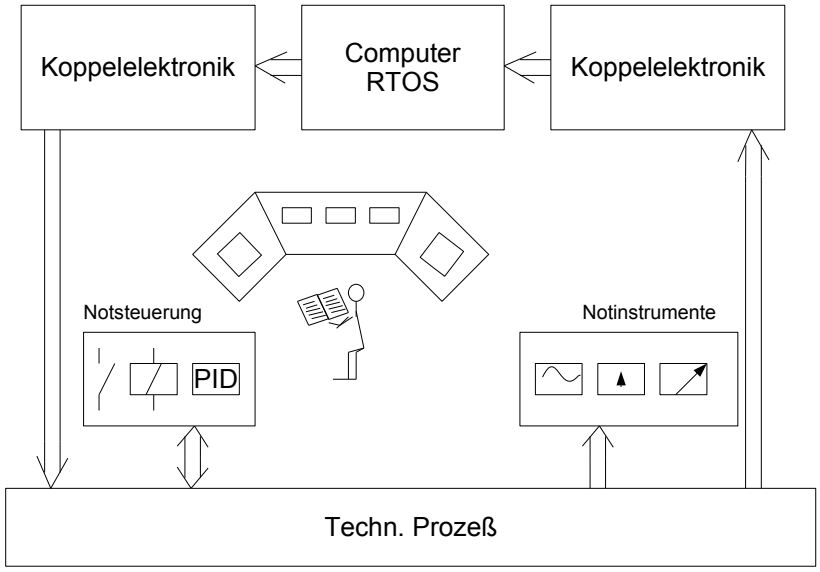
2.2 Mittlerer Automatisierungsgrad (Online- und open loop-Betrieb)



Kennzeichen:

- Gerätemäßige und zeitliche Kopplung von Computer und techn. Prozess
- Datenerfassung, Datenreduzierung, Protokollierung, Prozessvisualisierung durch den Computer (Prozessrechner PR)
- Hohe Anforderung bzgl. des Echtzeitverhaltens: Echtzeitbetriebssystem, Realtime Operating System (RTOS)
- Der Prozess wird weiterhin von dem Personal gefahren:
    - Entscheidung fällen für die Lenkung des techn. Prozesses unter Zuhilfenahme des Computers
    - Steuer- und Regelgeräte müssen unverändert bedient werden.
    - Entlastung des Personals

2.3 Hoher Automatisierungsgrad (Online- und closed loop-Betrieb)

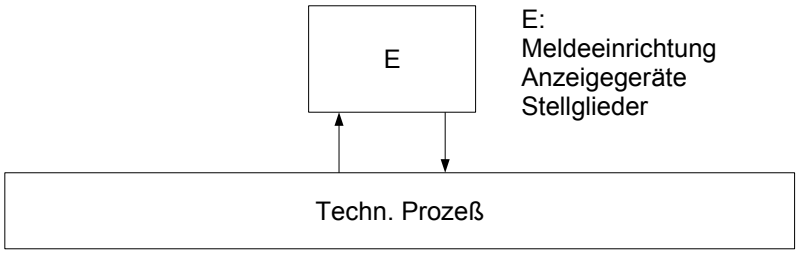


Kennzeichen:

- Gerätemäßige und zeitliche Kopplung von tech. Prozess
    - Daten erfassen, reduzieren, dokumentieren, visualisieren durch den Computer
    - Steuern und Regeln durch den Computer
    - höchste Anforderungen an das Echtzeitverhalten
- Die Betriebsleitung verbleibt weiterhin in der Verantwortung des Personals
- Das Personal hat im Normalfall nur noch überwachende Funktionen
- Prozesseingriffe bei Rechnerausfall werden vom Personal mit Hilfe der Notinstrumentierung und Notsteuerung vorgenommen

3 Entwicklungsstufen und Strukturen von Automatisierungssystemen

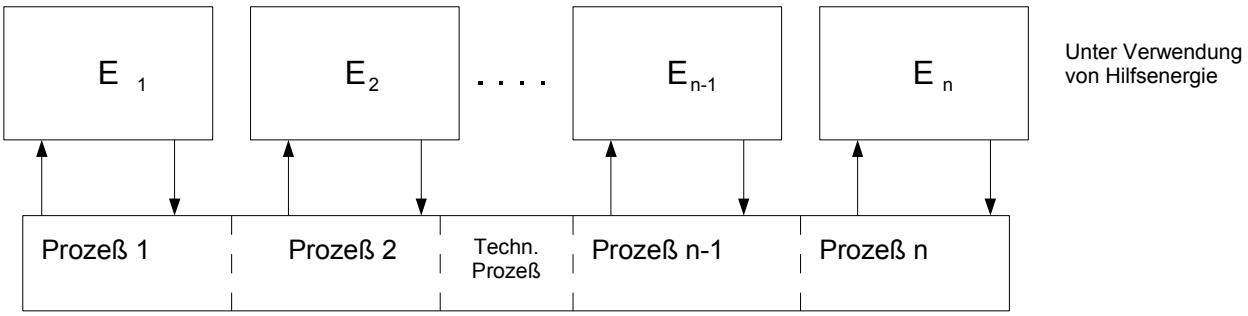
3.1 Einzelgerätetechnik



Automatisierung einzelner Maschinen

3.2 Dezentrale Struktur von Einzelgeräten, räumlich verteilt

Erweiterung der Prozessanlagen und Forderung nach Leistungsteigerung führten zu einem vermehrten Einsatz von Einzelgeräten, deren Leistung mit Hilfe von Hilfsenergie erhöht werden konnte. Das Automatisierungssystem besitzt eine dezentrale Struktur, die entsprechend des technischen Prozesses räumlich verteilt ist:

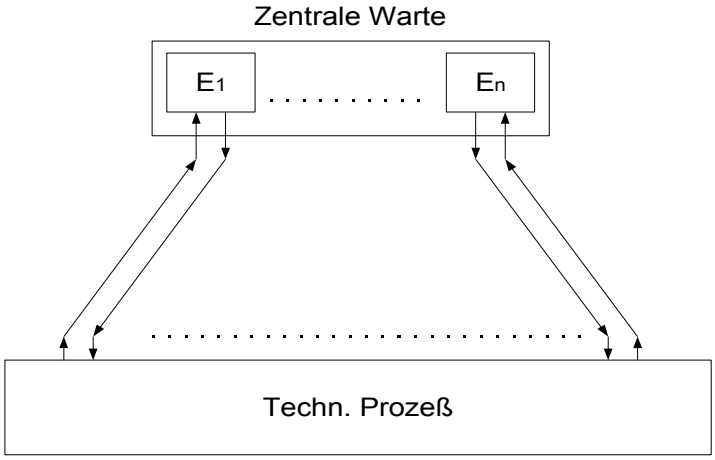


Eigenschaften:

- Die Einzelgeräte sind unabhängig von einander und arbeiten gleichzeitig parallel.
- Die Zuverlässigkeit kann je nach Zuverlässigkeit der Einzelgeräte sehr gut sein.
  Der Ausfall eines Einzelgeräts führt im Allgemeinen nicht zum Ausfall des gesamten techn. Prozesses.
- Ist nur bis zu einem gewissen Grad der Komplexität des techn. Prozesses möglich
- Die Kosten steigen linear mit der Anzahl der Geräte.

3.3 Dezentrale Struktur mit räumlicher Konzentration

Mit der zunehmenden räumlichen Ausdehnung der Produktionsanlagen und der Forderung nach besserer Anlagenausnutzung bei steigender Produktqualität entstand der Wunsch nach besseren Möglichkeiten der Prozessbeobachtung und des Prozesseingriffs. Dies führte zur räumlichen Konzentration der Einzelgeräte in Warten. Die Prozesserfassung und der Prozesseingriff erfolgt mit Hilfe von Fernmesseinrichtungen und Fernwirkeinrichtungen.

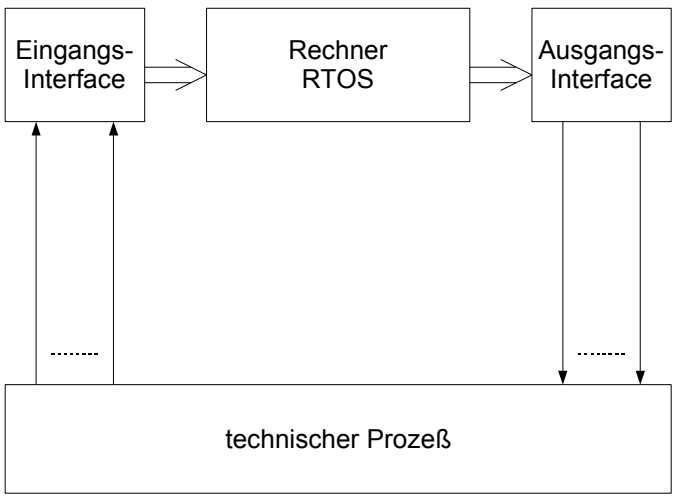


Eigenschaften:

- Unabhängigkeit der Einzelgeräte
- Zuverlässigkeit kann sehr gut sein
- Kosten steigen linear mit der Anzahl der Einzelgeräte
- Leistungssteigerung bedingt
  - Fernmesseinrichtungen
  - Fernwirkungseinrichtungen
  - Kompaktregler für einzelne Messgrößen

3.4 Zentraler Prozessrechner

Mit komplexer werdenden Aufgaben ist eine Digitalisierung der Informationsverwaltung notwendig. Ab etwa 1960 wird der erste klassische Prozessrechner in der Industrie eingeführt. In seiner ersten Einsatzform wurde der Prozessrechner als zentrales Prozessdatenverarbeitungsystem eingesetzt. D.h. ein Digitalrechner wurde über geeignete Interface-Schaltungen an den technischen Prozess angekoppelt und übernahm dann alle Automatisierungsaufgaben:

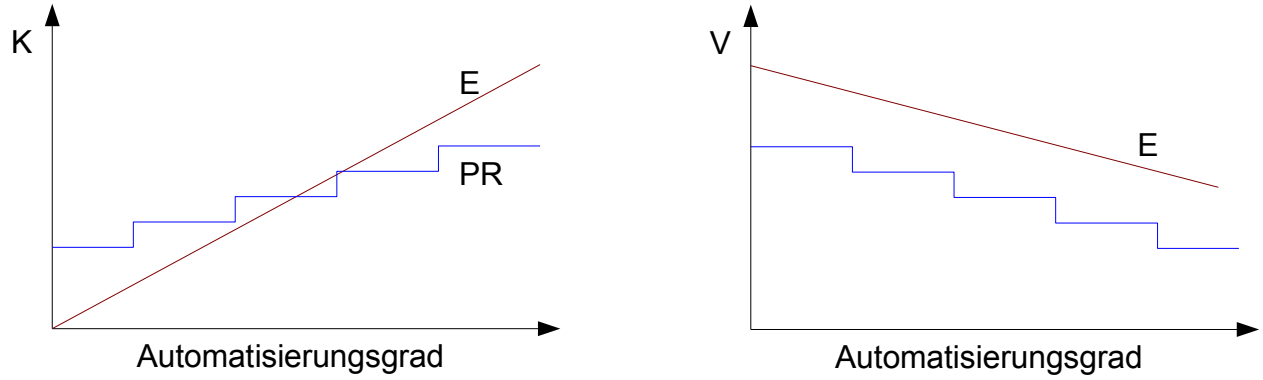


Die Abfrage der Eingangssignale, ihre Verarbeitung und die Bildung der Ausgangssignale erfolgt durch den zentralen Prozessrechner in sequentieller Arbeitsweise.

Eigenschaften:

- Einsetzbar für komplexe und rechenintensive Aufgaben
- Der Prozessrechner erfordert hohe Grundkosten.
- Stufenförmige Kostenzunahme bei Systemerweiterung, insgesamt mit geringerer Zunahme als bei Einzelgerätetechnik
- geringere Zuverlässigkeit als bei konventioneller Technik
- auf Grund der sequentiellen Arbeitsweise relativ niedrige Arbeitsgeschwindigkeit

Vergleich Einzelgerätetechnik E (konventionell) mit klassischem Prozessrechner PR



V: Verfügbarkeit

K: Kosten

3.5 Redundante Automatisierungssysteme

Bei technischen Anlagen, deren Ausfall mit unzulässig hohen Kosten oder mit der Gefährdung von Menschen verbunden ist, muss die erforderliche Zuverlässigkeit und Sicherheit durch geeignete Maßnahmen garantiert werden. Das Risiko solcher technischer Anlagen muss innerhalb tollerierbarer Grenzen liegen. Das Risiko wird bestimmt einerseits durch das Schadenspotenzial, das eine technische Anlage besitzt, und andererseits durch die Wahrscheinlichkeit, mit der ein Schadensfall eintreten kann:

$$Risiko = Schadenspotenzial \cdot Schadenswahrscheinlichkeit$$

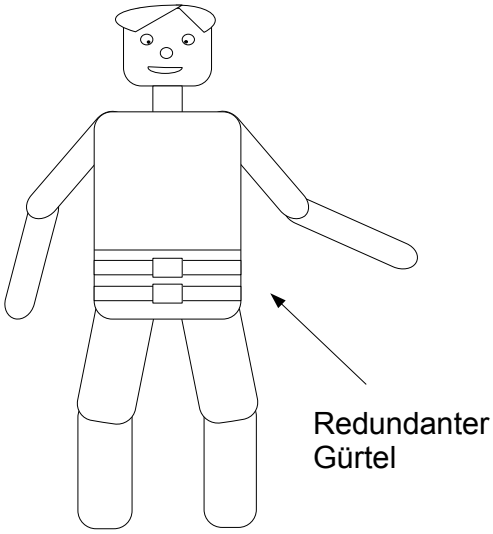
Geeignete Maßnahmen zur Reduzierung des Risikos:

- Nutzen von Verfahren und Systemen mit inhärenter Sicherheit:
  Bei Ausfall wird das System in einen sicheren Zustand überführt. (fehlersicher, fail safe)
- Überdimensionierung der Anlage,
  z.B. durch redundante Ausführung von Automatisierungssystemen.

Definition von Redundanz:

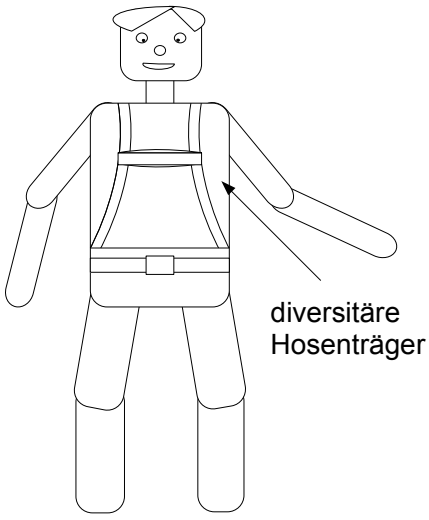
> Als Redundanz bezeichnet man den Aufwand, der über das zur Funktion erforderliche Maß hinausgeht.

z.B.: doppelter Gürtel



Problem: Keine Sicherheit bzgl. systematischer Fehler!

z.B.: Gürtel mit Hosenträger



Lösung: Andersartige Redundanz, Diversität

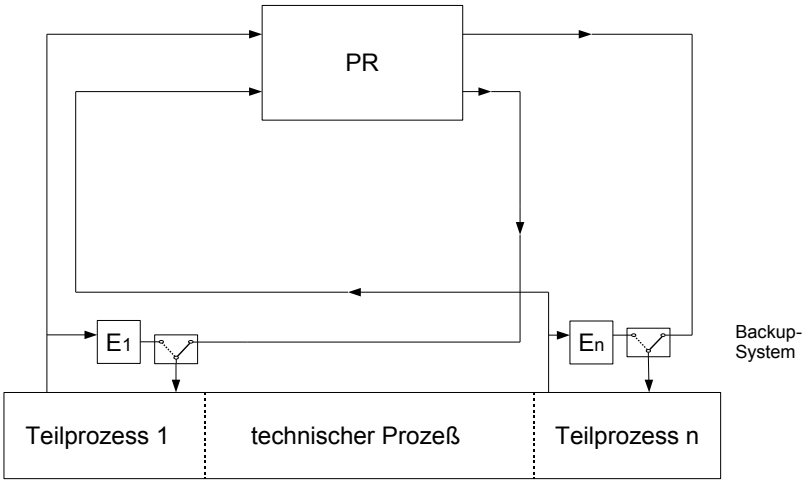
3.5.1 Systeme mit dynamischer Redundanz

3.5.1.1 Blinde Redundanz

Das redundante System ist im fehlerfreien Fall nicht tätig

a) Prozessrechner mit Back-up-System

Kopplung eines Prozessrechners mit redundanten Einzelgeräten (Back-up-System)



- Überwachung des Prozessrechners mit Fehlererkennung
- Automatischer Umschaltung bei Fehlerverhalten des PR's auf das Back-up-System

b) Prozessrechner mit Stand-by-System

Doppelrechnersystem: PR_1 und PR_2

PR_1 erledigt alle Automatisierungsaufgaben im Normalfall

PR_2 ist ein Bereitschaftsrechner mit folgenden Aufgaben:

- Überwachung von PR_1
- sichern der Prozessdaten und Ergebnisse von PR_1
- Bei Fehlverhalten von PR_1 Übernahme der Gesamt- oder Teilfunktion

3.5.1.2 Funktionsbeteiligte Redundanz: Master-Slave-System

Reduktion des Aufwands durch Aufteilung der Gesamtfunktion auf zwei Rechner. Beide Rechner können dann mit einem geringeren Aufwand realisiert werden. Die Aufgabenverteilung des Doppelrechnersystems (Master-Slave-System) sieht folgendermaßen aus:

Slave: Durchführung unbedingt notwendiger Funktionen unabhängig von der Funktionsweise des Masters

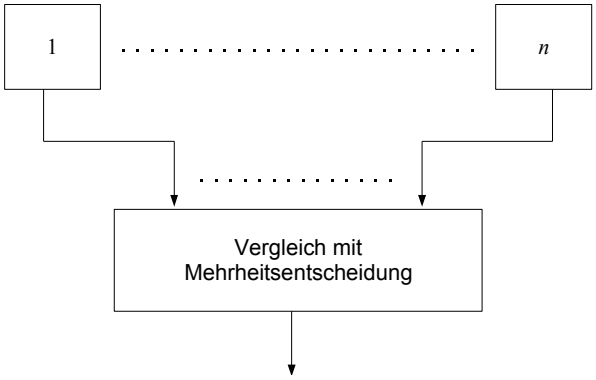
Master: Im Normalbetrieb Durchführung weniger notwendiger Aufgaben (Optimierungsrechnung, Prozessmanagement, ...) und Überwachung des Slaves
Bei Ausfall des Slaves Übernahme dessen Aufgaben unter Abwurf eigener, weniger wichtigen Aufgaben

Durch die Aufgabenverteilung ist gewährleistet, dass bei Ausfall eines der beiden Rechner die unbedingt notwendigen Funktionen erhalten bleiben und nur eine Einschränkung des Betriebs hinsichtlich der weniger wichtigen Aufgaben in Kauf genommen werden muss.

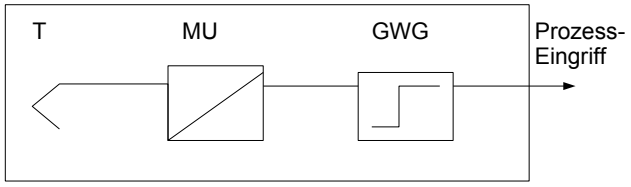
3.5.2 Systeme mit statischer Redundanz oder m–von–n–Redundanz

n Teilsysteme, jedes Teilsystem ist voll funktionsfähig

Die Teilsysteme sind unabhängig von einander.



Beispiel für ein Teilsystem:



GWG: Grenzwertgeber

Der Auswahllogik liegt eine Mehrheitsentscheidung nach folgendem Prinzip zu Grunde:

Gegeben sind *n* parallele Systeme. Die Mehrheitsentscheidung richtet sich nach einer vorzugebenden Zahl *m*, mit $1 \leqslant m \leqslant n$.

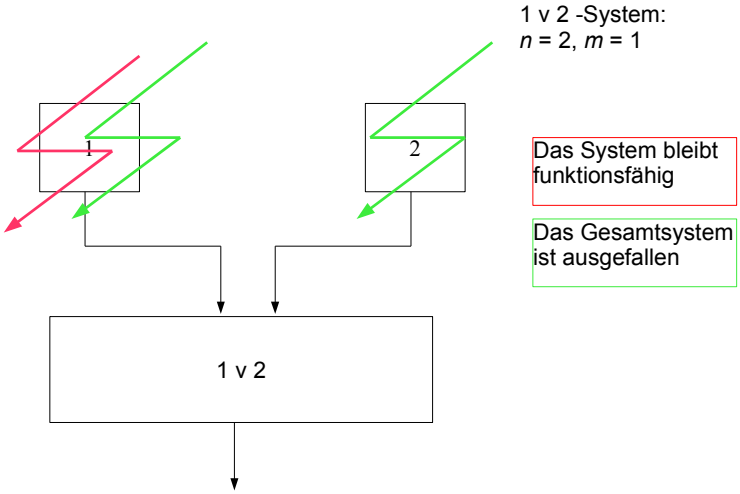
Ein Prozesseingriff wird dann vorgenommen, wenn mindestens *m* der *n* Systeme einen Prozesseingriff fordern. Man spricht daher von einer

m-von-*n*-Auswahllogik

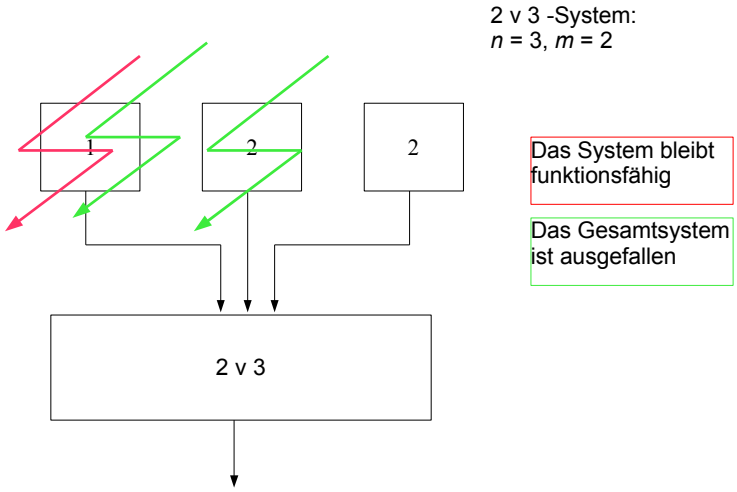
oder abgekürzt

m v *n*-Auswahllogik

In der Praxis häufig vorkommende Schaltungen:



Bei Ausfall eines von zwei Teilsystemen bleibt das Gesamtsystem funktionsfähig.



Bei Ausfall eines von drei Teilsystemen beliebt das Gesamtsystem funktionsfähig.

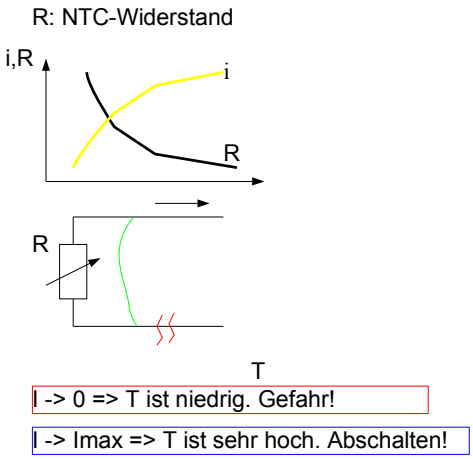
Bei einem Vergleich der beiden obigen mvn-Strukturen gewinnt man den Eindruck, dass trotz des Mehraufwands bei dem 2v3-System kein besseres Ausfallverhalten erreicht worden ist. Um dies besser beurteilen zu können, muss das Verhalten der Systeme bzgl. unterschiedlicher Fehlerarten untersucht werden.

Fehler
die allein oder im Zusammenwirken mit anderen die
Sicherheit Verfügbarkeit
der Anlage mindern, bezeichnet man als
gefährliche Fehler ungefährliche Fehler

Beispiel: Temperaturmessung mit einem NTC-Widerstand

Temperatur**zunahme** führt zur Widerstands**abnahme** und damit zur Strom**zunahme**.
Temperatur**abnahme** führt zur Widerstands**zunahme** und damit zur Strom**abnahme**.

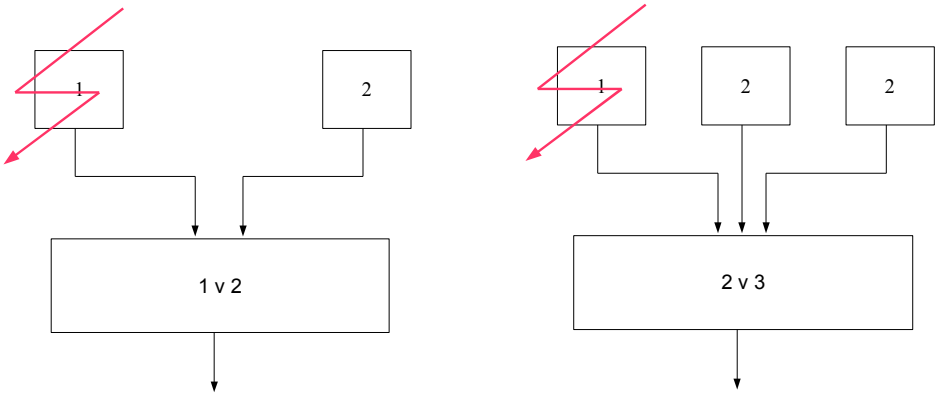
Bei einem **Leitungsbruch** wird der Strom Null. Dies signalisiert eine sehr niedrige Temperatur. Ein zu hohe Temperatur kann nicht erkannt werden, und erforderliche Gegenmaßnahmen daher können nicht eingeleitet werden: **gefährlicher Fehler**!



Bei einem **Kurzschluss** wird er Strom maximal. Dies signalisiert eine zu hohe Temperatur. Die Anlage wird abgeschaltet, obwohl keine Notwendigkeit dazu besteht: **ungefährlicher Fehler**!

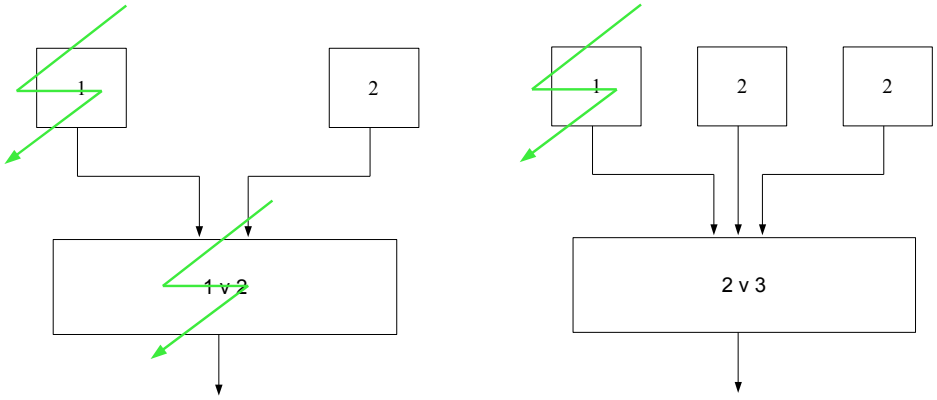
Unter Beachtung der beiden oben definierten Fehlerarten lässt sich jetzt ein unterschiedliches Systemverhalten feststellen:

Vergleich bzgl. gefährliche Fehler:



Bei einem gefährlichen Fehler sind beide Gesamtsysteme funktionsfähig. Mit den verbleibenden intakten Teilsystemen kann eine zu hohe Temperatur erkannt und eine erforderliche Aktion über die Auswahlschaltung ausgelöst werden. Wenn mehr Teilsysteme ausfallen, sind beide Gesamtsysteme ausgefallen.

Vergleich bzgl. ungefährliche Fehler:



Bei einem ungefährlichen Fehler ist das 1v2–System ausgefallen, da ein unnötiger Eingriff ausgelöst wird (1v2). Das 2v3-System bleibt funktionsfähig, da ein unnötiger Eingriff erst nach zwei Teilsystem-Ausfällen erfolgen würde.

Beide Strukturen verhalten sich also bei gefährlichen Fehlern gleich. Bei ungefährlichen Fehlern zeigt das 2v3-System das bessere Systemverhalten. Diese Struktur bringt also eine Verbesserung gleichermaßen bei gefährlichen wie ungefährlichen Fehlern.

Zuverlässigkeits–Analyse

Um über den bisherigen qualitativen Vergleich auch eine quantitative Beurteilung von mvn-Systemen vornehmen zu können, muss eine Zuverlässigkeits-Analyse durchgeführt werden. Dazu ist erforderlich Wahrscheinlichkeitsaussagen über das Eintreten von Ereignissen zu machen.

Beispiel von Ereignissen:

- Das System erfüllt seine Aufgabe: verfügbar *v*
- Das System ist ausgefallen: unverfügbar *u*

Für eine vereinfachte Behandlung der Zuverlässigkeits-Analyse werden folgende Annahmen gemacht:

Die Teilsysteme sind reparierbar.

Es liegen statistische Daten vor:

MTBF (meantime between failure) oder Ausfallrate: $\lambda = MTBF^{-1}$
MTTR(meantime to repair)

Damit lassen sich folgende Wahrscheinlichkeitswerte definieren:

Wahrscheinlichkeit der Verfügbarkeit:

$$v = \frac{MTBF}{MTBF + MTTR}$$

Wahrscheinlichkeit der Unverfügbarkeit:

$$u = \frac{MTTR}{MTBF + MTTR}$$

Setzt sich ein Gesamtsystem aus *n* Teilsystemen zusammen, für die voneinander unabhängige Einzelergebnisse $E_i \ (i = 1, \ldots, n)$ vorliegen, dann kann für das Gesamtsystem ein Gesamtergebnis *E* angegeben werden, das ist aus einer Kombination von Einzelergebnissen zusammengesetzt:

$$E = E_1 \wedge E_2 \wedge \ldots \wedge E_n$$ (Probability),

Wahrscheinlichkeit für das Eintreten des Einzelergebnisses E_i:

$$P_i = P(E_i) \qquad 0 \leqslant P_i \leqslant 1$$

Wahrscheinlichkeit für das Eintreten des Gesamtergebnisses *E*:

$$P = P(E) = P_1 \cdot P_2 \cdot \ldots \cdot P_n = \prod_{i=1}^{n} P_i$$

Wenn mehrere Kombinationsmöglichkeiten zu einem Gesamtergebnis *E* führen

$$E = (E_1 \wedge \ldots \wedge E_n)^1 \vee (E_1 \wedge \ldots \wedge E_n)^2 \vee \ldots \vee (E_1 \wedge \ldots \wedge E_n)^k$$

dann lässt dich die Wahrscheinlichkeit für das Eintreten dieses Gesamtereignisses *E* mit *k* Kombinationsmöglichkeiten folgendermaßen bestimmen:

$$P = P(E) = \left(\prod_{i=1}^{n} P_i\right)^1 + \left(\prod_{i=1}^{n} P_i\right)^2 + \ldots = \sum_{j=1}^{k} \left(\left(\prod_{i=1}^{n} P_i\right)^j\right) = \sum_{j=1}^{k} P^j$$

mit $P^j = \left(\prod_{i=1}^{n} P_i\right)^j$

Bei den n Teilsystemen eines mvn-Systems kommen zwei Einzelereignisse vor: „verfügbar“ oder „unverfügbar“

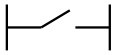
Deren Wahrscheinlichkeit ist:

$$P(E_i) = u_i \, oder \, v_i$$

Da alle Teilsysteme gleich sind, können für sie gleiche Wahrscheinlichkeitswerte angenommen werden:

$u_i = u, v_i = v,$ mit $i = 1 \, dots \, low \, n$

Zur Veranschaulichung können die Ereignisse mit Hilfe von Schaltern graphisch dargestellt werden:

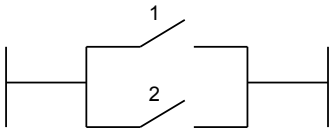
ausgefallen: 

verfügbar: 

Ein Teilsystem ist ausgefallen, wenn keine leitende Verbindung von links nach rechts gegeben ist, bzw. es ist verfügbar, wenn eine leitende Verbindung existiert.

Das Gesamtergebnis wird durch eine geeignete Kombination von Einzelereignissen gebildet, d.h. durch eine geeignete Kombination von Schaltern dargestellt. Dabei gilt ebenso für das Gesamtsystem, dass es ausgefallen ist, wenn keine leitende Verbindung von links nach rechts besteht, bzw. es ist verfügbar, wenn es eine leitende Verbindung existiert.

Anwendung für

a) 1 v 2–System: 

E_1	E_2	E^i	E	P_1	P_2	P^i	P
offen	offen	offen	offen	u	u	u^2	u^2
offen	geschlossen	geschlossen	geschlossen	u	v	uv	$v^2+2\text{uv}$
geschlossen	offen	geschlossen		v	u	vu	
geschlossen	geschlossen	geschlossen		v	v	v^2	

$$V = P(1 \vee 2 \text{ - } \textit{System ist verfügbar}) = v^2 + 2\text{uv}$$

$$U = P(1 \vee 2 \text{ - } \textit{System ist ausgefallen}) = u^2$$

$$P(1 \vee 2 \text{ - } \textit{System ist ausgefallen oder verfügbar}) = V + U = 1$$

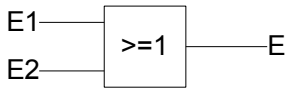
$$v^2 + 2\text{uv} + u^2 = (u+v)^2 = 1^2 = 1$$

Eine weiter grafische Darstellung stellt der Funktionsbaum dar, bestehend aus UND- und ODER-Gliedern. Die Ereignisse werden als binäre Zustände dargestellt:

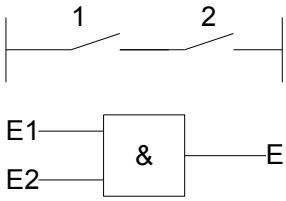
„0“ entspricht „unverfügbar“
„1“ entspricht „verfügbar“

Die logische Schaltung stellt das Gesamtereignis dar. Es gilt als ausgefallen, wenn der Ausgang eine logische „0“ besitzt, es gilt als verfügbar, wenn der Ausgang eine logische „1“ besitzt.

z.B. 1v2-System:



b) 2 v 2–System:

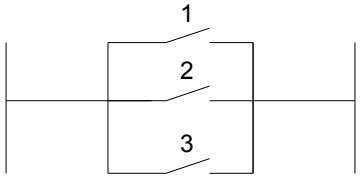


$$U = 2\text{uv} + u^2$$

$$V = v^2$$

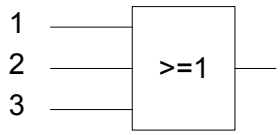
$$U + V = 1$$

E_1	E_2	E^i	E	P_1	P_2	P^i	P
offen	offen	offen	offen	u	u	u^2	$u^2+2\text{uv}$
offen	geschlossen	offen		u	v	uv	
geschlossen	offen	offen		v	u	vu	
geschlossen	geschlossen	geschlossen	geschlossen	v	v	v^2	v^2

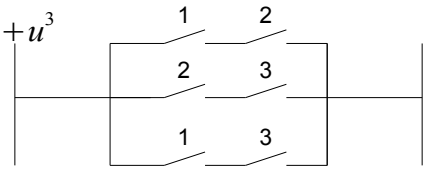


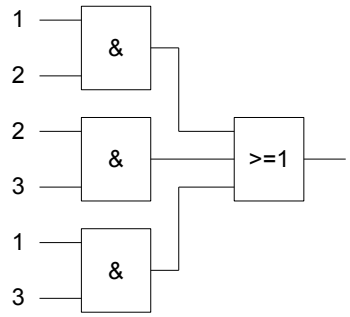
c) 1 v 3–System:

$$V = v^3 + 3v^{2u} + 3vu^2$$
$$U = u^3$$



$$1 = v^3 + 3v^{2u} + 3vu^2 + u^3$$





d) 2 v 3–System:

$$V = 3uv^2 + v^3$$
$$U = 3u^{2v} + u^3$$

$$1 = V + U = v^3 + 3v^{2u} + 3vu^2 + u^3$$

Schrittweise Verallgemeinerung für mvn-Systeme:

Für ein Einzelsystem: $1 = u + v$

2 Teilsysteme: $1 = 1^2 = (u+v)^2 = (v+u)^2 = u^2 + 2uv + v^2$

3 Teilsysteme: $1 = 1^3 = (u+v)^3 = (v+u)^3 = u^3 + 3u^{2v} + 3uv^2 + v^3$

n Teilsysteme bilden das Gesamtsystem: $1 = 1^n = (u+v)^n = (v+u)^n$

$$(v+u)^n = \sum_{k=0}^{n} \binom{n}{k} v^k n^{n-k}$$

m v *n*:
$$V = \sum_{k=m}^{n} \binom{n}{k} v^k u^{n-k}$$
$$U = \sum_{k=0}^{m-1} \binom{n}{k} v^k u^{n-k}$$

Eine andere Form der Gleichung ergibt sich, wenn man von folgender Beziehung ausgeht:

$$(u+v)^n = \sum_{k=0}^{n} \binom{n}{k} u^k v^{n-k}$$

Das Gesamtsystem ist verfügbar, wenn mindestens m Teilsysteme verfügbar sind. Dies entspricht allen Summanden

mit $n-k \geq m$ oder $n-m \geq k$

d.h. $$V = \sum_{k=0}^{n-m} \binom{n}{k} u^k v^{n-k}$$

$$U = \sum_{k=n-m+1}^{n} \binom{n}{k} u^k v^{n-k}$$

Eine Umformung ist auch möglich auf Grund der Eigenschaft:

$$\binom{n}{k} = \binom{n}{n-k}$$

Beispiel 1: Ein Netzteil:

$$\lambda = \frac{1 Ausfall}{2\text{a}}$$

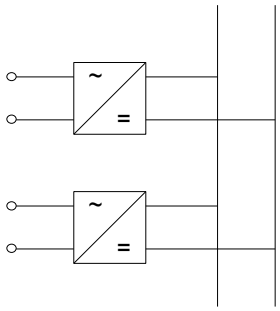
$$MTTR = 1\text{d}\,; \quad 1\text{a} = 365\text{d}$$

$$MTBF = \lambda^{-1} = 2\text{a}$$

gesucht: u, v in Exponentialdarstellung mit 3 Stellen hinter dem Komma

$$v = \frac{MTBF}{MTBF + MTTR} = 9{,}986 \cdot 10^{-1}$$

$$u = \frac{MTTR}{MTBF + MTTR} = 1{,}368 \cdot 10^{-3}$$



Zur Erhöhung der MTBF werden zwei Netzteile parallelgeschaltet
Dies entspricht einem 1v2-System.

gesucht: V, U, λ_{ges}

$$V = u^2 + 2\text{uv} = 1{,}000 \quad \Rightarrow \quad U = 0 \quad ?$$

Achtung bei Angaben mit Rundung!
Eine genau Rechnung liefert:

$$U = u^2 = 1{,}871 \cdot 10^{-6}$$

Damit ergibt sich folgende mittlere ausfallfreie Zeit MTBF des Gesamtsystems:

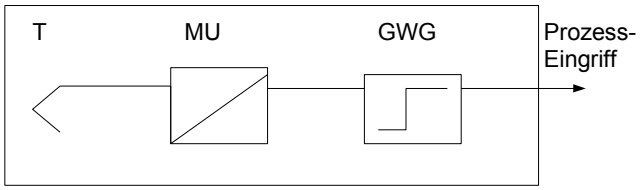
$$U = \frac{MTTR}{MTBF + MTTR}$$

$$U\,(MTBF + MTTR) = MTTR$$

$$MTBF = \frac{1-U}{U} \cdot MTTR = \left(\frac{1}{U} - 1\right) \cdot MTTR = \left(\frac{1}{1{,}871} \cdot 10^6 - 1\right) \cdot 1\text{d} \approx \frac{10^6}{1{,}871 \cdot 365}\, a \approx 1465\text{a}$$

In Anbetracht dieser riesigen Zahl sollte man sich verdeutlichen, dass statistische Aussagen durchaus einer kritischen Beurteilung unterzogen werden sollten. Denn kein Elektrogerät wird auch nach beliebiger häufiger Reparatur diese Zeitspanne überleben.

Beispiel 2: Messkette



$$u_T = 0{,}001$$

$$u_{MU} = u_{GWG} = 0{,}005$$

$$U_{MK},\ V_{MK} \quad ?$$

$$V_{MK} = v_T \cdot v_{MU} \cdot v_{GWG} = (1-u_T)(1-u_{MU})(1-u_{GWG}) = (1-u_T)(1-2u_{MU}+u_{MU}^2) =$$
$$= 1-2u_{MU}+u_{MU}^2-u_T+2u_T u_{MU}-u_T u_{MU}^2$$
$$= 1-(u_{MU}+u_{GWG}+u_T)+u_{MU}^2+2u_T u_{MU}-u_T u_{MU}^2 =$$
$$= 1-U$$

$$U = (u_{MU}+u_{GWG}+u_T)-(u_{MU}^2+2u_T u_{MU})+u_T u_{MU}^2$$
$$= 0{,}001+2\cdot 0{,}005-\ldots(\approx 0{,}011) = 1{,}097\cdot 10^{-2}$$

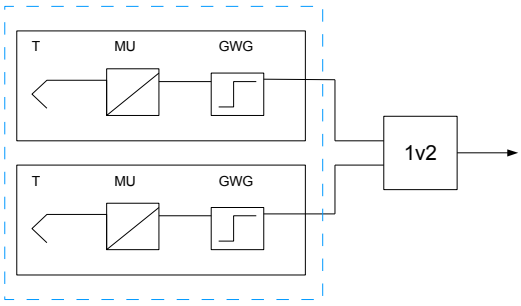
Nun zwei Messketten parallel:

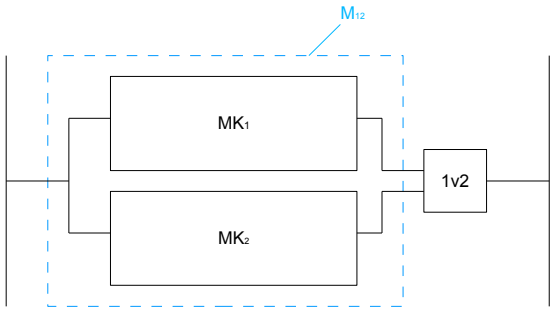
$$V_{MK} = v_T \cdot v_{MU} \cdot v_{GWG}$$

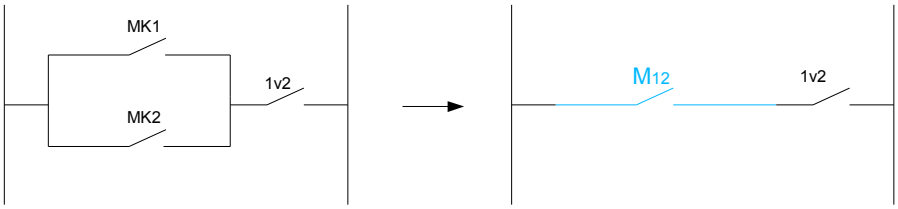
$$U_{MK} = 1{,}097\cdot 10^{-2}$$

$$u_{1v2} = 10^{-5}$$

$$U_{ges} = ?$$







$$U_{12} = U_{MK}^2 = 1{,}203 \cdot 10^{-4}$$

$$V_{ges} = V_{12} \cdot v_{1v2} = (1 - U_{12})(1 - u_{1v2}) = 1 - (U_{12} + u_{1v2} - U_{12} \cdot u_{1v2})$$

$$U_{ges} = U_{12} + u_{1v2} - U_{12} \cdot u_{1v2} \approx U_{12} + u_{1v2} = 1{,}203 \cdot 10^{-4} + 10^{-5} = 1{,}303 \cdot 10^{-4}$$

Betrachtung von Ausfallwahrscheinlichkeiten bzgl. gefährlichen bzw. ungefährlichen Fehlern:

gefährliche Fehler: u, v, U

ungefährliche Fehler: $\underline{u}$, $\underline{v}$, $\underline{U}$

gefährliche Fehler für *m* v *n*-System: $U_{m v n} = ?$

verfügbar: wenn mindestens m Teilsysteme verfügbar sind.

unverfügbar: wenn $m-1$ oder weniger Teilsysteme verfügbar sind oder wenn n-(m-1) oder mehr Teilsysteme ausgefallen sind

$$U_{m v n} = \sum_{k=n-m+1}^{n} \binom{n}{k} u^k \cdot v^{n-k}$$

ungefährliche Fehler *m* v *n*-System: $\underline{U}_{m v n} = ?$

Ausfall ist dann gegeben, wenn ein ungewollter Prozesseingriff erfolgt, d.h. wenn mindestens *m* Teilsysteme einen unnötigen Eingriff fordern:

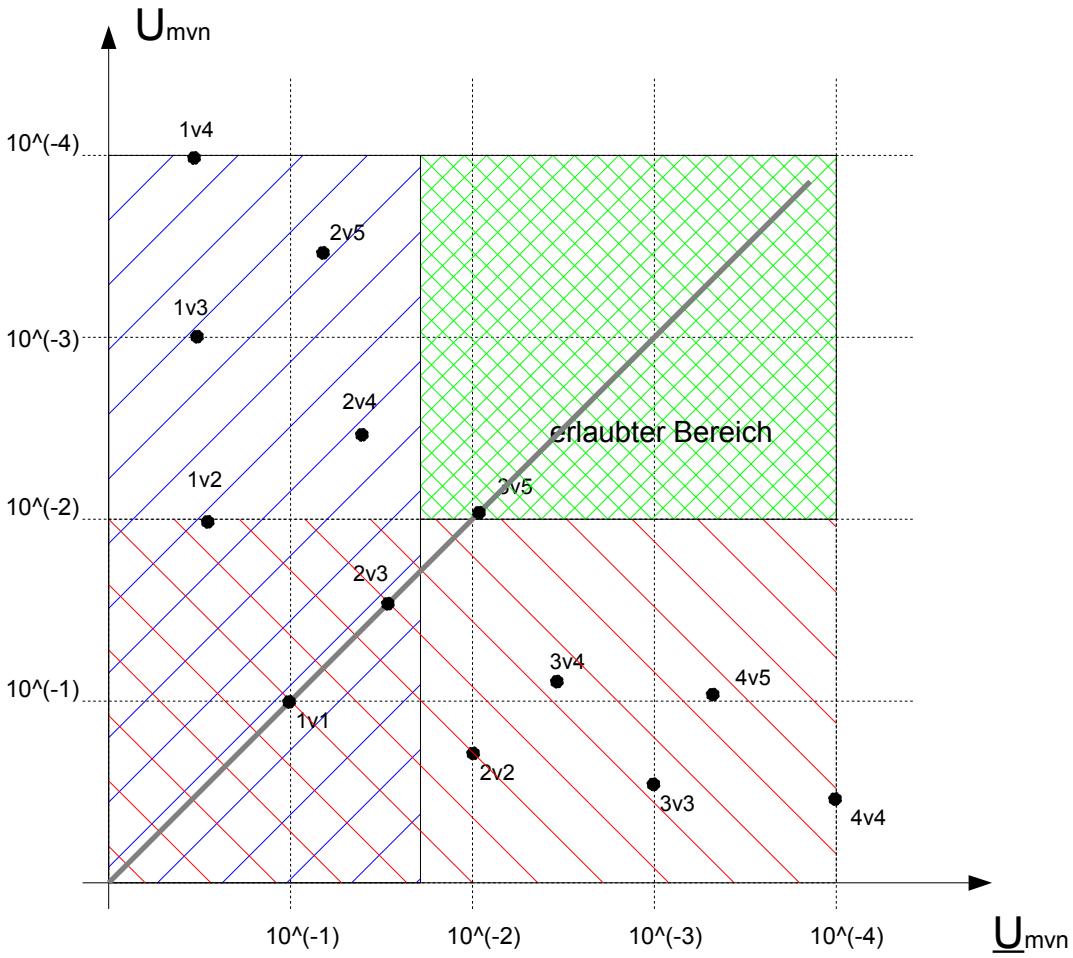
$$\underline{U}_{m v n} = \sum_{k=m}^{n} \binom{n}{k} \underline{u}^k \cdot \underline{v}^{n-k}$$

Annahme: $u = \underline{u} = 0{,}1$

m v *n*	$U_{m v n}$	$\underline{U}_{m v n}$	$-\log U_{m v n}$	$-\log \underline{U}_{m v n}$
1 v 1	0,1	0,1	1	1
1 v 2	0,01	0,19	2,00	0,72
2 v 2	0,19	0,01	0,72	2,00
1 v 3	0,001	0,272	3	0,57
2 v 3	0,028	0,028	1,55	1,55
3 v 3	0,272	0,001	0,57	3
1 v 4	0,0001	0,3459	4	0,46
2 v 4	0,0037	0,0523	2,43	1,28
3 v 4	0,0523	0,0037	1,28	2,43
4 v 4	0,3459	0,0001	0,46	4
2 v 5	0,0005	0,0815	3,3	1,09
3 v 5	0,0086	0,0086	2,07	2,07
4 v 5	0,0815	0,0005	1,09	3,3

Bsp.: Forderung Betreiber: $\underline{U} \leq 5 \cdot 10^{-2}$ $-\log(\underline{U}) = 1{,}3$

Forderung Genehmigungsbehörde: $U \leq 10^{-2}$ $-\log(U) = 2$

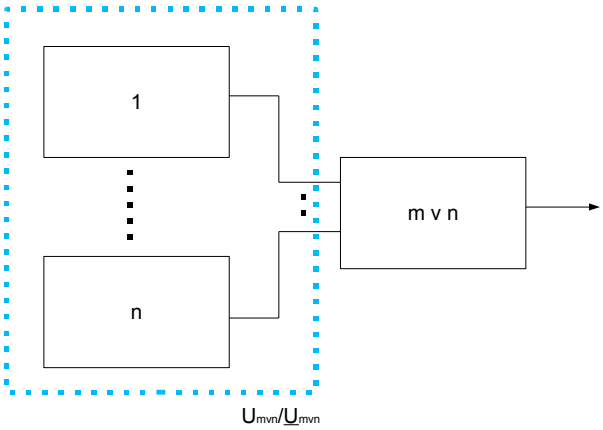


Der geringste Aufwand zur Erfüllung beider Forderungen: **3 v 5–System**

Aufgabe: $u = \underline{u} = 10^{-2}, \quad u_{mvn} = \underline{u}_{mvn} = 10^{-6}$

Forderung: $U < 10^{-5}$

$\underline{U} < 10^{-3}$



Welches *m* v *n*-System erfüllt die Forderung bei geringstem Aufwand?

Bevor unnötige Rechnungen durchgeführt werden, sollte man einige Fälle von vornherein ausscheiden. Ein 1vn-System kommt ebenso wenig in Frage wie ein mvn-System:

$\underline{U}_{1vn} > \underline{U}_{1v(n-1)} > \ldots > \underline{U}_{1v1} = \underline{u} > \underline{U}$ und $U_{mvn} > U_{(n-1)v(n-1)} > \ldots > U_{1v1} = u > U$

Erster Versuch mit 2v3-system für härtere Anforderung: $U_{2v3} < U$?

$$U_{2v3} = \sum_{k=2}^{3} \binom{3}{k} u^k \cdot v^{3-k} = 3u^{2v} + u^3 = 3\text{left}(¿)^2 (1-10^{-2}) + (10^{-2})^3 = 2{,}98 \cdot 10^{-4} > 10^{-5}$$

Eine Berechnung von $\underline{U}_{2v3}$ erübrigt sich somit:

Nächster Versuch mit $U_{2v4} < U$?

$$U_{2v4} = \sum_{k=3}^{4} \binom{4}{k} u^k \cdot v^{4-k} = 4u^{3v} + u^4 = 4 \cdot 10^{-6} \cdot \left(1 - 10^{-2}\right) + 10^{-8} = 3{,}97 \cdot 10^{-6}$$

$$V_{ges} = V_{2v4} \cdot v_{2v4} = (1 - U_{2v4})(1 - u_{2v4})$$

$$U_{ges} = 1 - V_{ges} = 1 - (1 - U_{2v4})(1 - u_{2v4}) = U_{2v4} + u_{2v4} - U_{2v4} \cdot u_{2v4} =$$

$$= 3{,}97 \cdot 10^{-6} + 10^{-6} - 3{,}97 \cdot 10^{-12} = 4{,}97 \cdot 10^{-6} < 10^{-5}$$

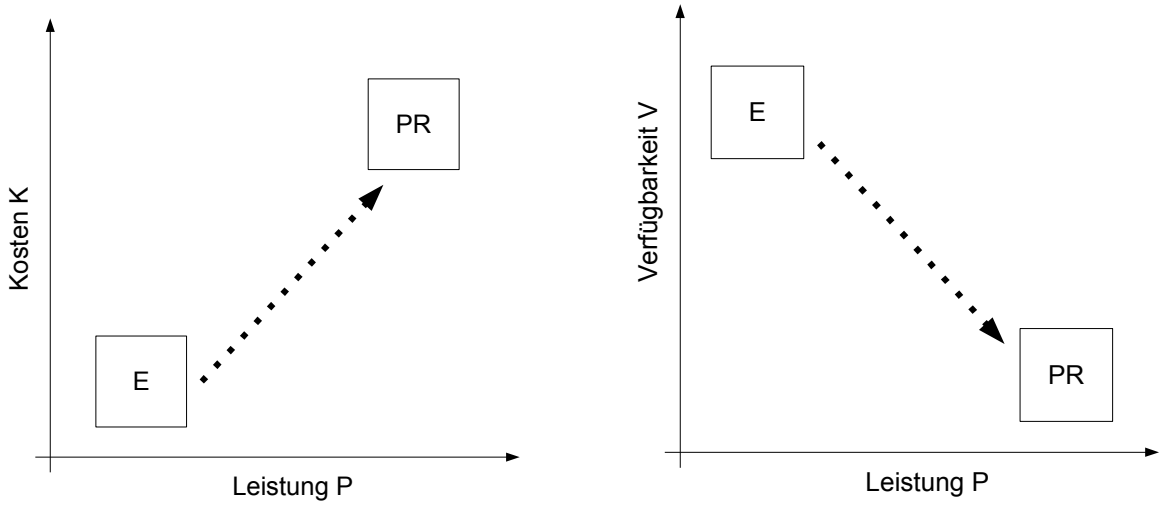
$$\underline{U}_{2v4} > \underline{U} \quad ?$$

$$\underline{U}_{2v4} = \sum_{k=2}^{4} \binom{4}{k} \underline{u}^k \cdot \underline{v}^{4-k} = 6\underline{u}^2 \underline{v}^2 + 4\underline{u}^3 \cdot \underline{v} + \underline{u}^4 = 5{,}920 \cdot 10^{-4}$$

$$\underline{U}_{ges} = 5{,}920 \cdot 10^{-4} + 10^{-6} - 5{,}920 \cdot 10^{-10} = 5{,}93 \cdot 10^{-4} < 10^{-3}$$

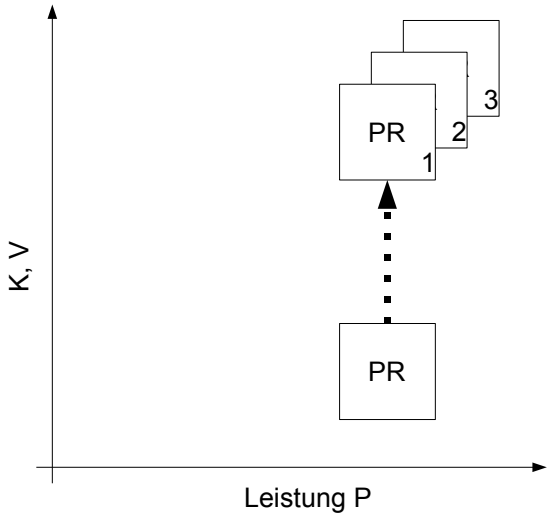
Das 2v4-System erfüllt somit mit dem geringsten Aufwand beide Forderungen bzgl. gefährlichen und ungefährlichen Fehlern.

3.6 Dezentrale Rechnersysteme

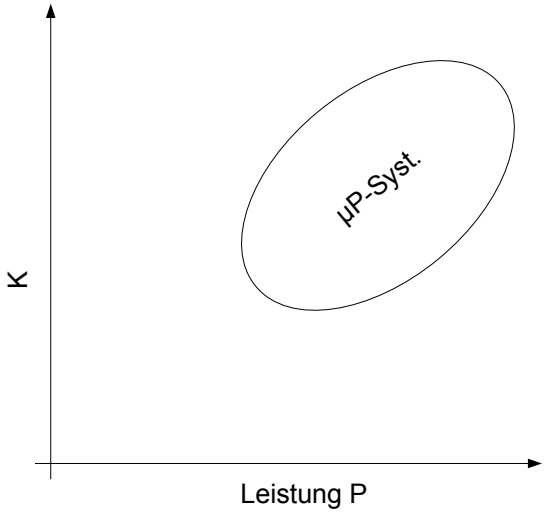


PR: Prozessrechner
E: Einzelsystem

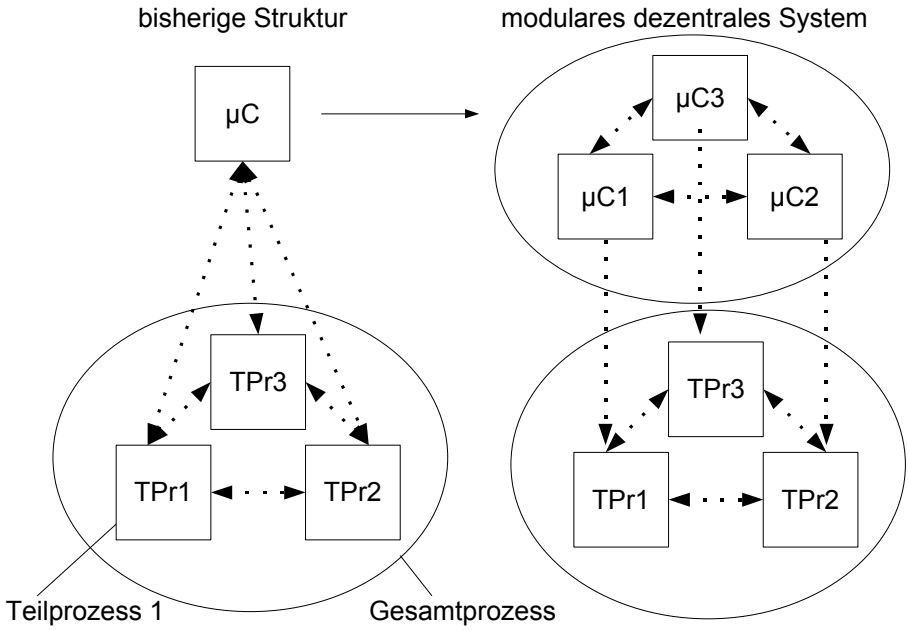
Eine Verbesserung der Verfügbarkeit konnte mit Hilfe verschiedener Redundanzsysteme erzielt werden. Durch den damit verbundenen Mehraufwand hat sich allerdings das Preis-Leistungsverhältnis bei den Prozessrechnersystemen noch weiter verschlechtert:



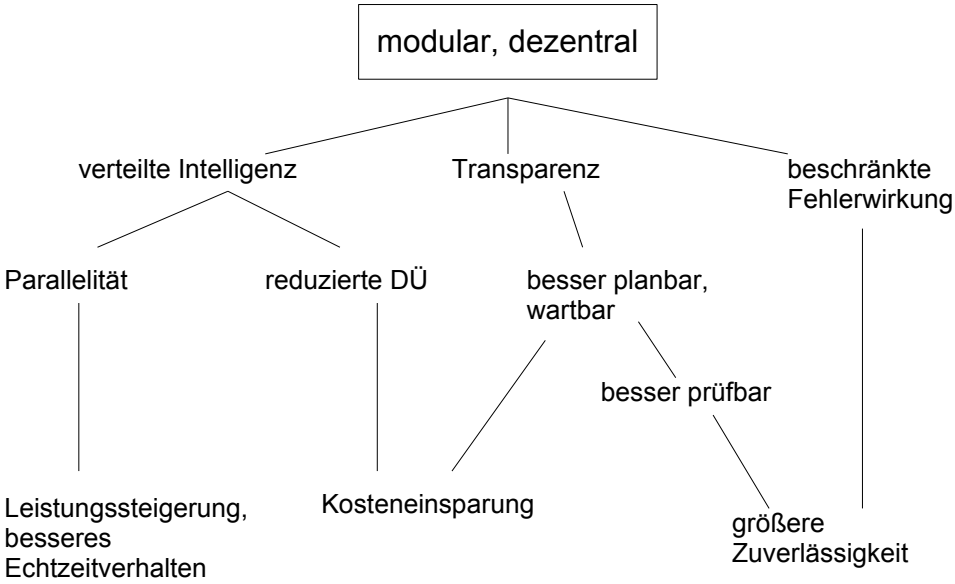
Mit der Einführung des Mikroprozessors und anderer Halbleiterbausteine konnte das Kostenproblem durch optimale Anpassung verschiedener Gerätesysteme an die Komplexität der Automatisierungsaufgabe deutlich reduziert werden:



Ungelöst blieb das Zuverlässigkeitsproblem bei nicht redundanten Systemen. Der Ausfall eines zentralen Rechners führt immer zum Ausfall des Gesamtsystems. Eine Verbesserung dieses Problems, aber auch der Kostensituation konnte erst mit Einführung neuer Automatisierungsstrukturen erreicht werden.

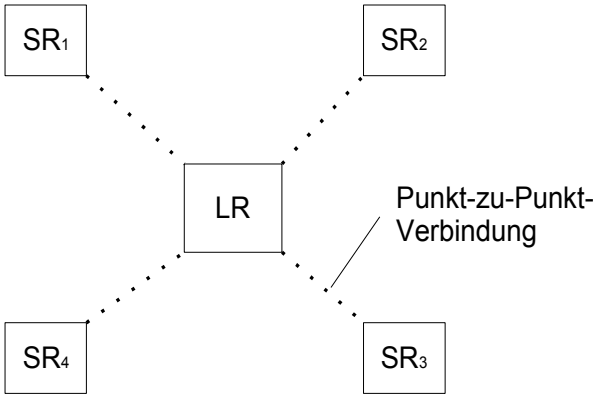


Merkmale:



DÜ: Datenübertragung

3.6.1 Sternstruktur



LR: Leitrechner

SR: Satellitenrechner

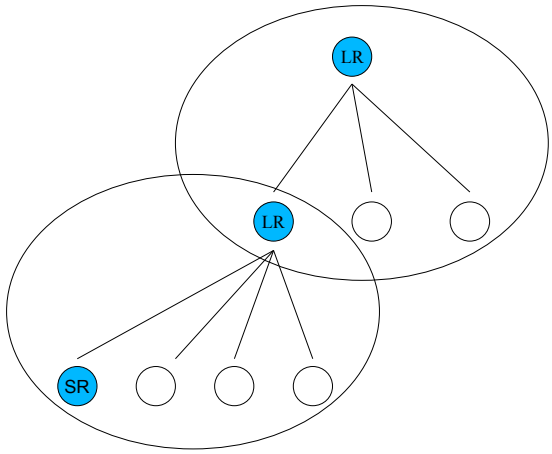
Die Sternstruktur besteht aus einem zentralen Leitrechner (LR) und mehreren dezentralen Satellitenrechnern (SR). Der Leitrechner ist über Punkt-zu-Punktverbindungen mit den Satellitenrechnern gekoppelt. Diese Verbindungen können mit wenigen Adern über weite Strecken erfolgen. Eine Kommunikation der SR miteinander geht nur indirekt über den LR.

Der Leitrechner übernimmt übergeordnete Funktionen, während die Satellitenrechner die prozessnahen Aufgaben ausführen.

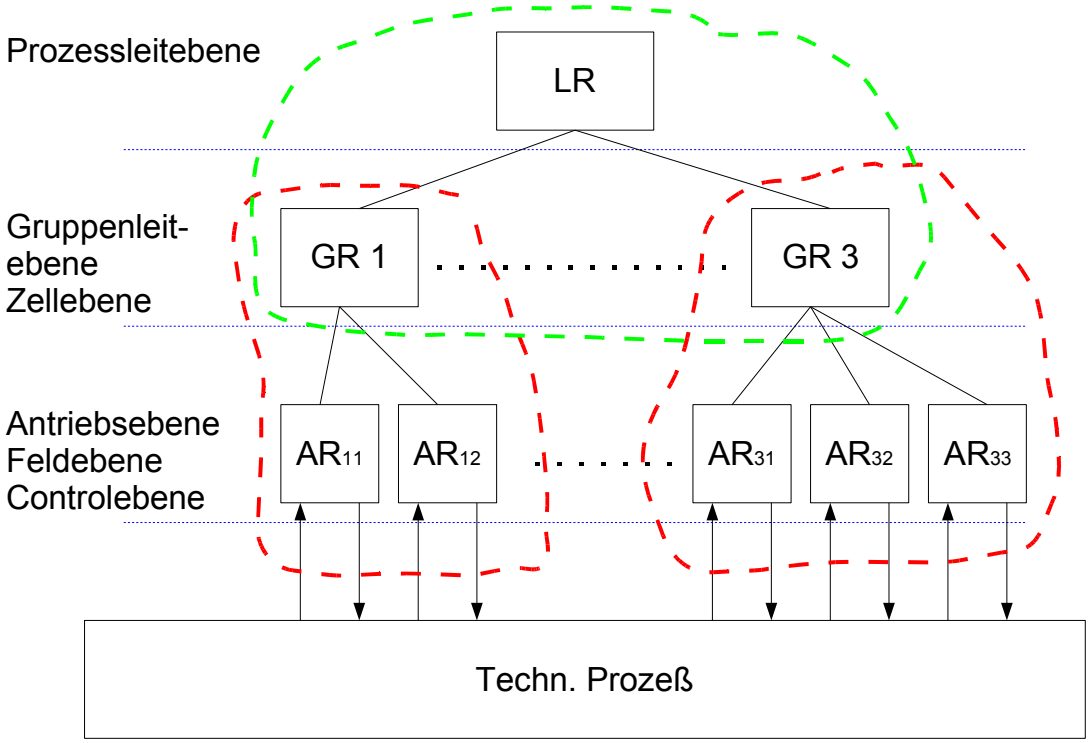
Typisches Einsatzgebiet: Energieverteilungssysteme mit ihren Fernwirksystemen

3.6.2 Mehrschichtige Hierarchiestruktur

Koppelt man mehrere Sternstrukturen miteinander zu einer hierarchischen Struktur, so erhält man die mehrschichtige Hierarchiestruktur:



Diese Struktur entspricht i.W. der hierarchischen Organisationsstruktur eines Betriebs mit den drei Organisationsebenen Betriebsleitung, Abteilungsebene und Sachbearbeiterebene. Die hierarchische Struktur eines Automatisierungssystems ist entsprechend unterteilt in Prozessleitebene, Gruppenleitebene oder Zellebene und Feldebene oder Controlebene oder Antriebsebene. Je nach Komplexität des Gesamtsystems können auch weniger oder mehr als drei Hierarchieebenen vorkommen.



AR (Antriebsrechner):

- prozessnahe Aufgaben: Messen, Steuern, Regeln
- Informationsfluss besteht aus kleinen Datenpaketen
- Übertragung erfolgt unter Echtzeitbedingungen

GR (Gruppenrechner):

- Koordinieren der AR's
- Vorgabe von Stellgrößen zur Führung der unterlagerten Einzelrechner

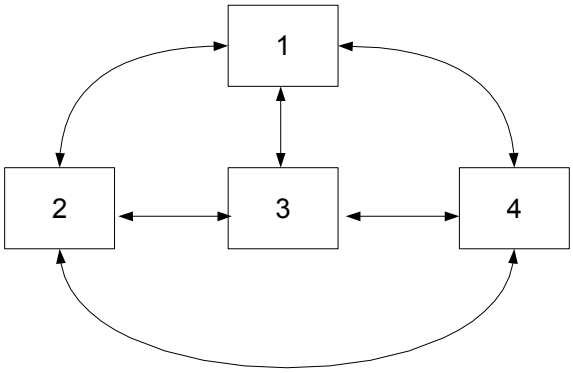
LR (Leitrechner):

- Überwachung und Steuerung der unterlagerten Steuerungskreisen der Gruppenebene
- Berechnung von Prozesskennwerten
- Unterstützung des Leitpersonals
- Vorgabe von Führungsgrößen

Ein direkter Datenaustausch innerhalb einer Ebene ist nicht möglich.

3.6.3 Netzstruktur

Die Dezentralisierung ist bei der Stern- und mehrschichtigen Hierarchiestruktur nur unvollständig gelungen, der bzw. die Leitrechner haben eine zentrale Bedeutung behalten. Eine Kommunikation zwischen benachbarten Satellitenrechnern kann nur unmittelbar über den Leitrechner erfolgen, was zu einer unnötigen Belastung dieses Rechners führt. Durch Aufgabe von zentralen Funktionen des Leitrechners und direkter Kommunikation aller Rechner miteinander über Punkt–zu–Punkt–Verbindungen erhält man die Netzstruktur:



Merkmale:

- Leistungssteigerung durch verbesserte Kommunikationsfähigkeit
- verbesserte Zuverlässigkeit
- starker Anstieg der Verkabelung:

  Anzahl der Punkt–zu–Punkt–Verbindung bei *n* Teilsystemen:

$$\frac{n\cdot(n-1)}{2}$$

3.6.4 Netzwerke

Auf Grund des oben erwähnten Nachteils sollte in modernen Automatisierungssystemen eine Struktur mit Punkt–zu–Punkt–Verbindungen vermieden werden.

Die Kommunikation findet statt dessen über ein gemeinsames Medium statt

Man spricht in diesem Fall von einem Netzwerk.

Bezeichnungen von Netzwerken:

Aktor-/ Sensor-Bus (AS-BUS)

Feldbus

LAN

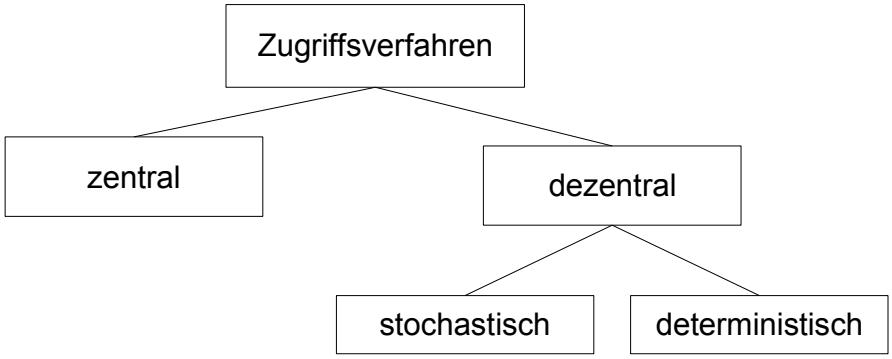
(MAN)

WAN

Merkmale von Netzwerken

3.6.4.1 Zugriffsverfahren

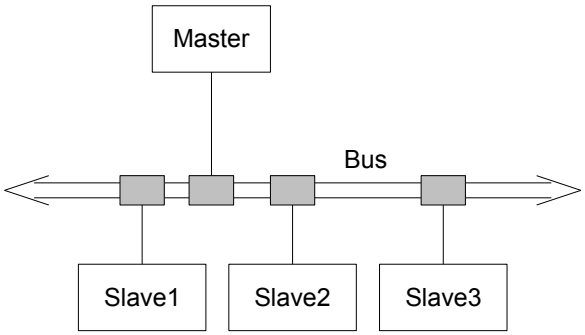
Verfahren, nach dem Teilnehmer auf das gemeinsame Übertragungsmedium zugreifen dürfen



- Bei dem zentralen Verfahren wird die Zugriffsberechtigung durch ein dediziertes Gerät zentral vergeben.
- Bei dem dezentralen stochastischen Verfahren erfolgt der Zugriff in einem Wettbewerb. Konkurrenzsituation wird nach einem Zufallsprinzip gelöst.
- Bei dem dezentralen deterministischen Verfahren wird die Zugriffsberechtigung nach einer gemeinsamen Regel innerhalb vorgegebener Zeitschranken reihum unter den Teilnehmern weitergereicht.

Zentrales Zugriffsverfahren

Master–Slave–Prinzip



Beim Master-Slave-Prinzip wird der Bus zentral durch den Master verwaltet. Der Master fragt nach einer vorgegebenen Reihenfolge die Teilnehmer (Slaves) nach einem Sendewunsch ab.

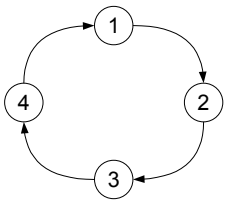
Bei vorliegender Forderung bekommt der Teilnehmer vom Master für eine beschränkte Zeit T die Zugriffsberechtigung erteilt (Polling-Verfahren). Die Reihenfolge wird durch den Anwender in einer Teilnehmerliste festgelegt. Die Zeit zum Abarbeiten aller Teilnehmer nach der Teilnehmerliste bezeichnet man als Buszykluszeit:

$$maximale\ Buszykluszeit = n \cdot T$$

mit $n = Anzahl\ der\ Slaves$

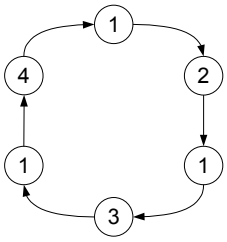
$T = maximale\ Zugriffszeit$

Bei einer Teilnehmerliste mit vier Teilnehmern ist die Buszykluszeit 4T. Dies ist die Zeit, die jeder Teilnehmer maximal warten muss, um wieder die Buszugriffsberechtigung zu erhalten.



Sl. 1
Sl. 2
Sl. 3
Sl. 4

Ist die Buszykluszeit für einen zeitkritischen Teilnehmer zu lang, dann lässt sich seine Wartezeit durch mehrmaliges Eintragen in die Teilnehmerliste verkürzen. Die Buszykluszeit für die restlichen Teilnehmer erhöht sich entsprechend:



Sl. 1
Sl. 2
Sl. 1
Sl. 3
Sl. 1
Sl. 4

$$Wartezeit \leq 2\mathrm{T} \quad \textit{für Sl 1}$$

$$Buszykluszeit = 6\mathrm{T} \quad \textit{für Sl } i,\ i \neq 1$$

Weitere Möglichkeit zur Verkürzung der Wartezeit:

- Eintragen in eine Alarmliste
- Jeder dieser Teilnehmer kann einen Alarm erzeugen. Der Master unterbricht die z.Z. laufenden Busaktionen und übergibt die Sendeberechtigung dem alarmerzeugenden Teilnehmer.
  Bevorzugtes Einsatzgebiet: Feldbussysteme
- Bei Vorliegen eines Alarms wird die Abarbeitung der normalen Teilnehmerliste unterbrochen und stattdessen die Alarmliste abgearbeitet.

Dezentrales Zugriffsverfahren

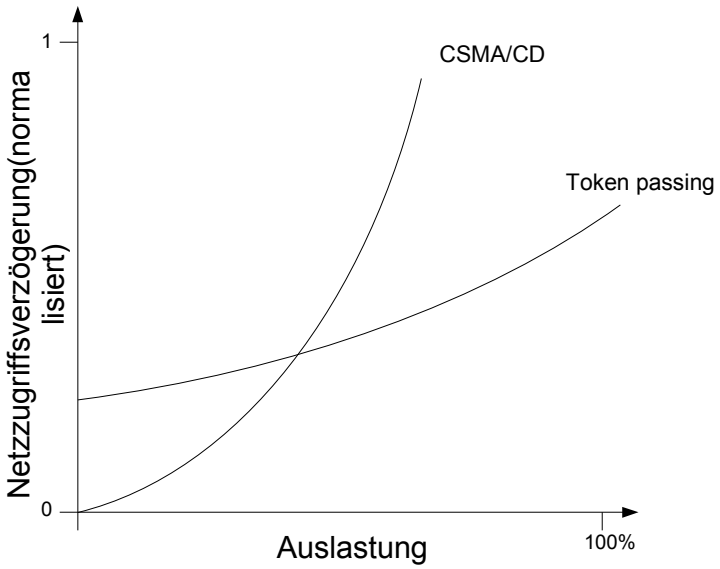
Stochastisches Verfahren

CSMA / CD, z.B. Ethernet

Deterministisches (dez.) Verfahren

Token passing, z.B. IMB-Ring, H3 von Siemens

Vergleich CSMA / CD mit Token passing:



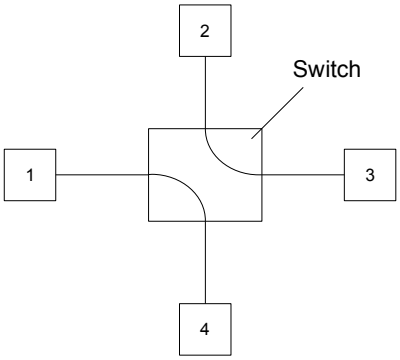
Prinzipielle Möglichkeiten für Echtzeitfähigkeiten von Ethernet:

- deterministisches Anwenderprotokoll
  - Mater–Slave–Prinzip
  - Token–Passing–Verfahren auf Anwenderebene

    z.B. RETHER: Realtime Ethernet

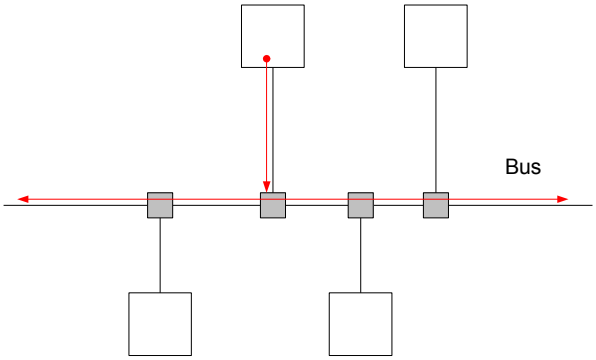
    Zum Teil geht der Vorteil des Ethernets verloren.
- Lastbeschränkung

  zwar nicht deterministisch, aber voraussagbares optimales Verhalten, Predictable Ethernet
- Kollisionsvermeidung mit Hilfe von Switch–Modulen



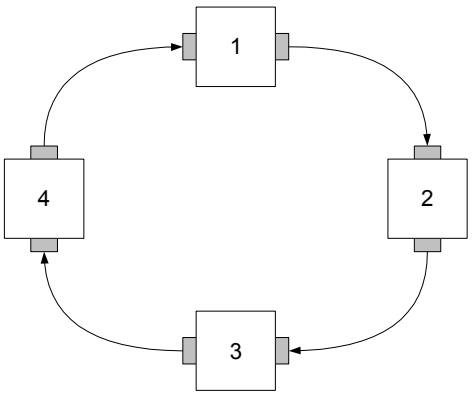
3.6.4.2 Netz–Topologie

Bus–Struktur



– quasi gleichzeitiger Empfang bei allen Teilnehmer
– besonders zuverlässige Struktur (keine zentralen Einrichtungen)

Ring–Struktur



Jeder Teilnehmer besitzt zwei Schnittstellen.

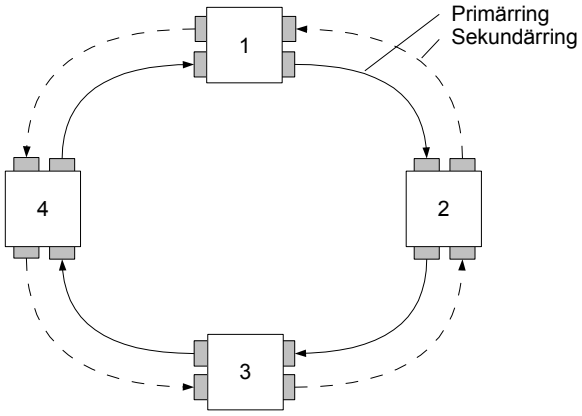
Eine Übertragung von Teilnehmer *i* zu Teilnehmer *j* erfolgt über die dazwischenliegenden Teilnehmer.

Die Teilnehmer sind ständig aktiv und regenerieren durchlaufende Daten -> große Entfernungen für die Datenübertragung möglich.

Nachteil: Jeder Teilnehmer muß ständig betriebsbereit sein. Ausfall eines Teilnehmers führt zum Ausfall des Rings.

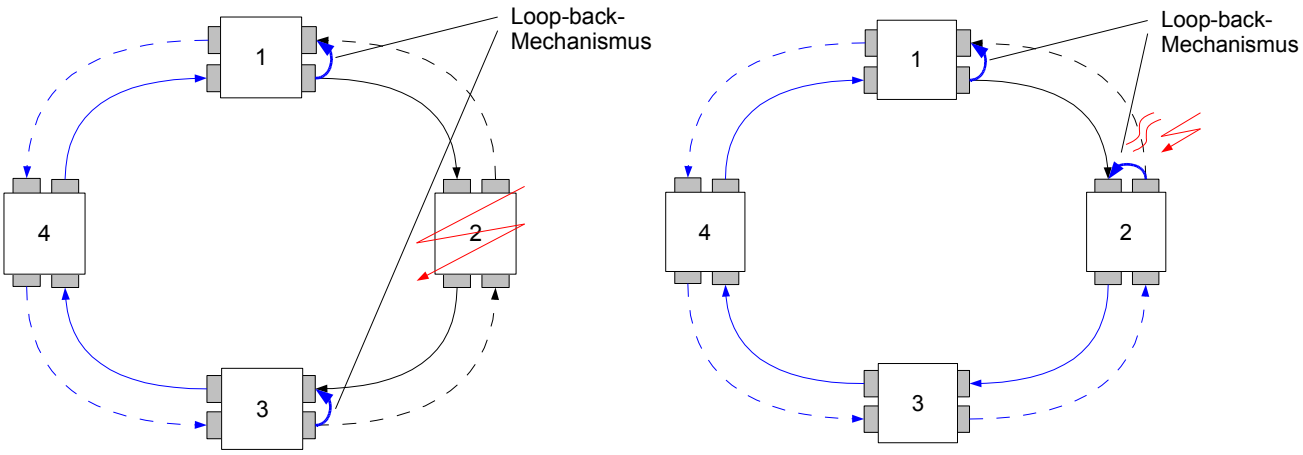
Behebung dieses Nachteils: Doppelring

z.B. H3 von Siemens

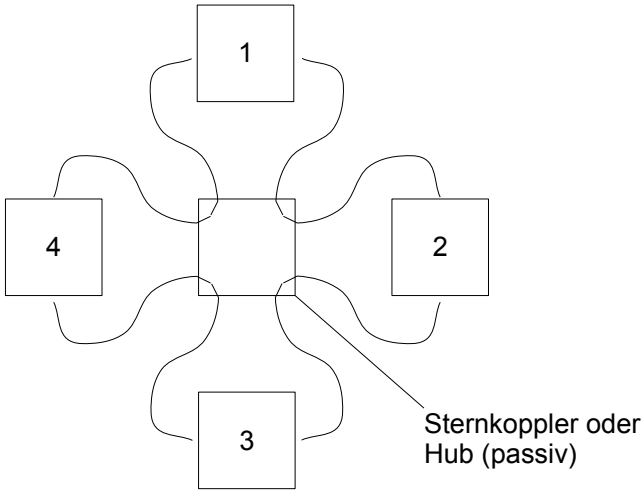


Datenübertragung im Normalfall über den Primärring. Bei Ausfall eines Teilnehmers oder Leitungsbruch im Primärring wird der Sekundärring zur Reparatur herangezogen:

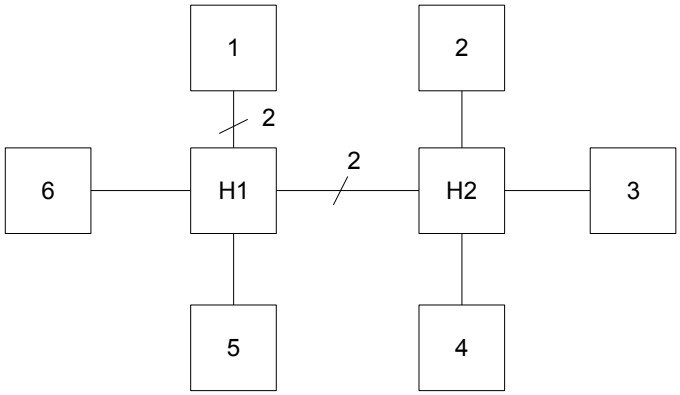
Loop–back–Mechanismus



Stern-Struktur



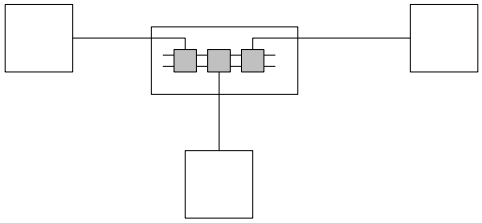
aktive Hubs erlauben die Kopplung mehrerer Sterne



typisches Beispiel: Arcnet

weitere Realisierungsmöglichkeiten eines Hubs:

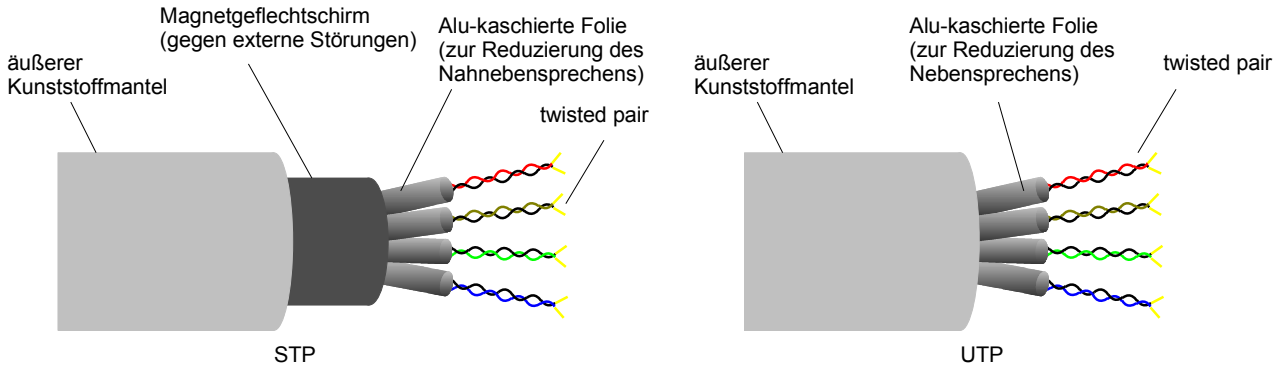
– Hochgeschwindigkeitsbus



– Switch

3.6.4.3 Übertragungsmedien

Zweidrahtleitung (Twisted Pair)



Kabel mit ein oder mehreren ineinander verdrillten Adernpaare mit Alu-kaschierter Kunststofffolie, evtl. mit Schirmgeflecht und Mantel.

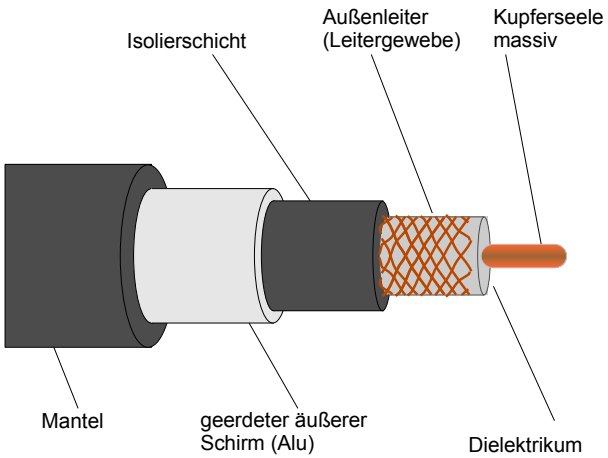
STP: Shielded Twisted Pair

UTP: Unshielded Twisted Pair

bis 100 MBit/s

Koaxialkabel

doppelt geschirmtes Koaxialkabel



Kabel mit einem Innenleiter (massives Cu) in einem Dielektrikum, darum der Außenleiter (Drahtgewebe), darum Isolierschicht und geerdeter äußerer Schirm (Alu), danach Mantel

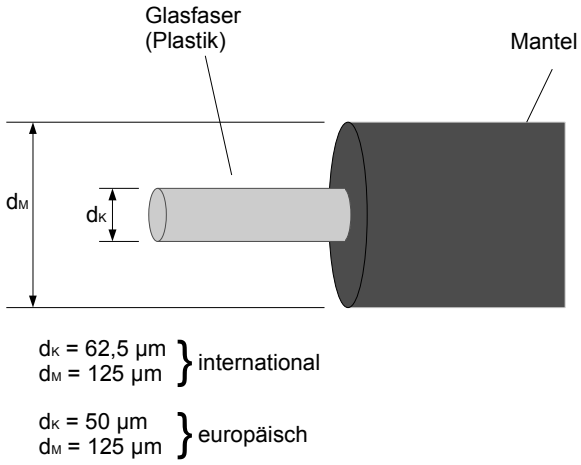
Biegeradius: 20 cm

aufwendiger Aufbau, wenig empfindlich gegen Störeinflüsse, aufwendiger Abgriff über einen Transceiver (MCU: Medium Access Unit) und das Dropcable

einfaches Koaxialkabel (bevorzugt in Bürowelt), Biegeradius: 8 cm

Lichtwellenleiter (LWL)

Glas, Kunststoff



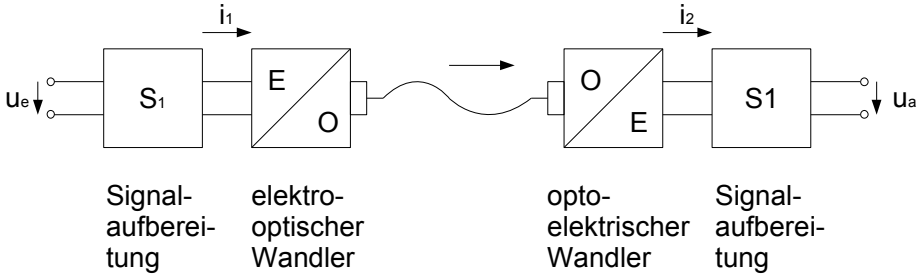
Multimode–Gradientenfaser

Übertragungsrate:	Gbit/s–Bereich
Übertragungslänge:	bis 100 km

Eigenschaften:

- keine Erdungsprobleme
- Potentialtrennung zwischen Sender und Empfänger
- keine elektromagnetischen Störeinflüsse
- sehr niedrige, frequenzunabhängige Grunddämpfung ⇒ hohe Übertragungsrate
- geringes Gewicht
- leichte Verlegbarkeit

Übertragung erfolgt über eine End–to–End–Verbindung:



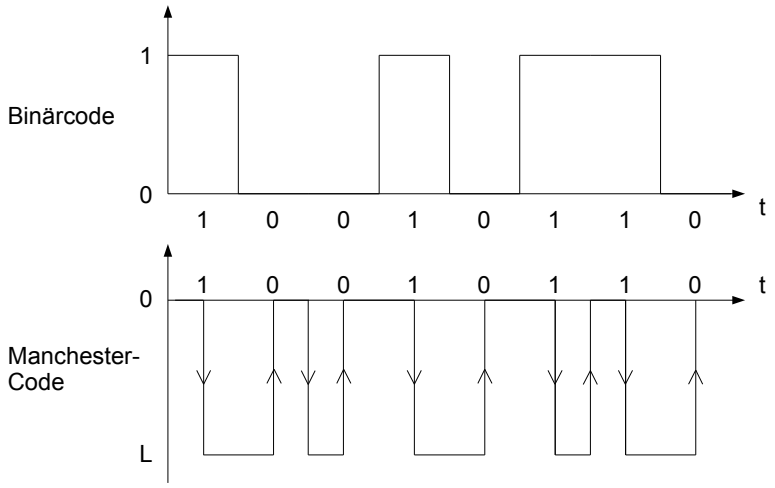
Bei bidirektionaler Kommunikation Verwendung von 2 LWL

3.6.4.4 Übertragungsverfahren

Basisband

Ein Übertragungskanal, Nutzung im Zeitmultiplex für die verschiedenen Teilnehmer.

Die Signale (0, 1) werden unmoduliert übertragen. Die Übertragung erfolgt bidirektional. Häufig verwendeter Code mit Taktrückgewinnung, z.B. Manchester-Code

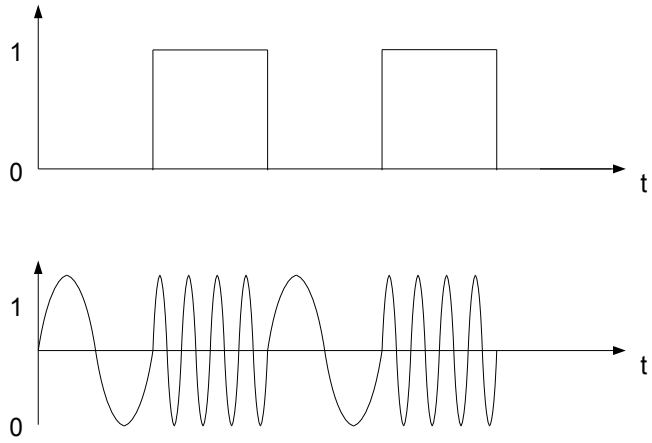


Trägerband oder Carrierband

Datensignale werden auf eine Trägerfrequenz aufmoduliert

(eine Trägerfrequenz)

Übertragung erfolgt bidirektional



Frequenzmodulation

Einsatz bevorzugt bei Koaxkabel.

Breitbandverfahren (Broadband)

Mehrere Frequenzbänder

Entsprechend viele Übertragungskanäle

Unterschiedliche Nutzung der Kanäle:

Daten, Video, Sprachen

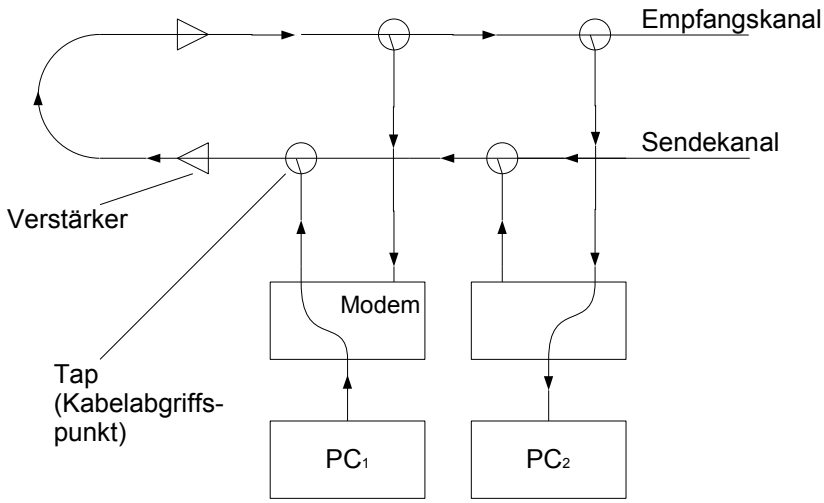
Ursprung, Kabelfernsehen (CATV)

Auf Grund der vorhandenen Verstärker kann in einem Kanal nur unidirektional übertragen werden.

⇒ zwei getrennte Übertragungspfade für bidirektionale Übertragung.

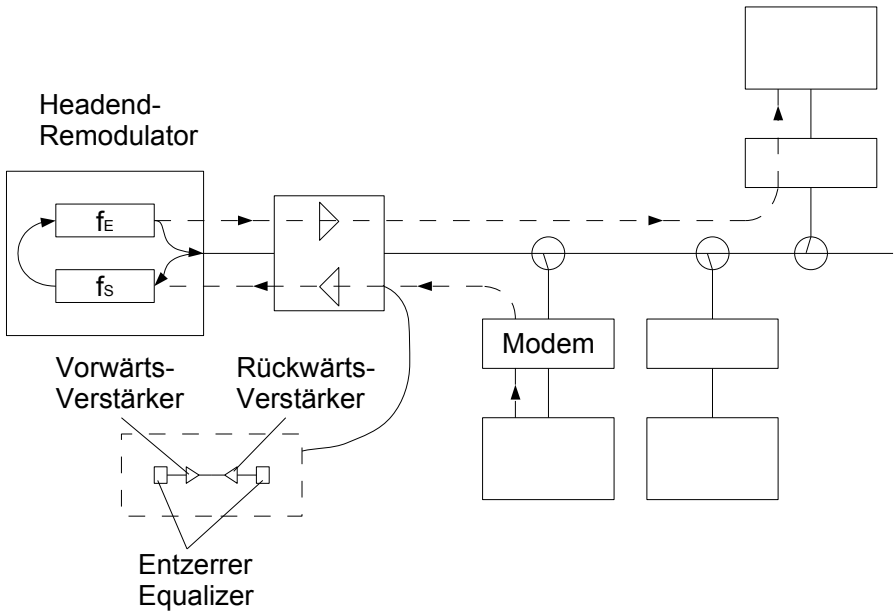
a) Zweikabel-Breitband-System

Beispiel: Wang-Net

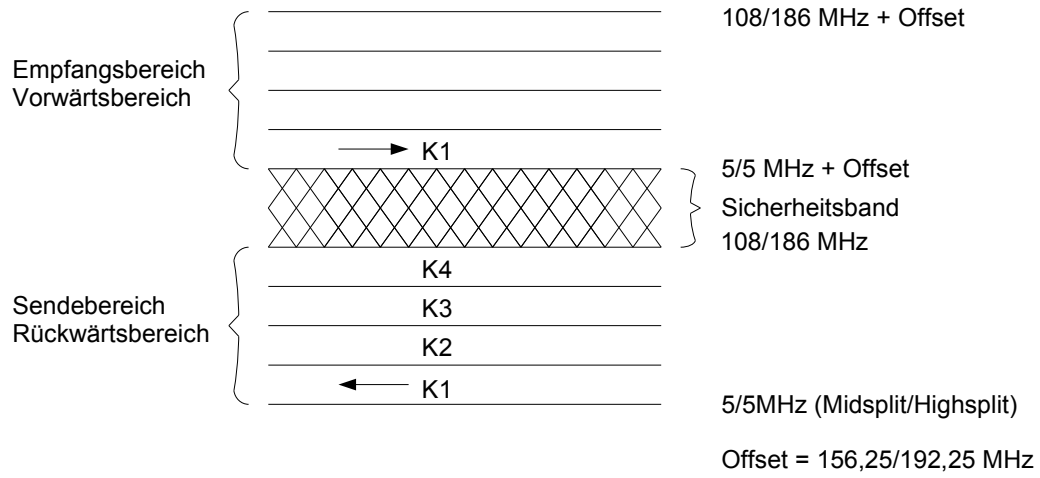


b) Midsplit-Verfahren (seit Mitte der 70er)

Highsplit-Verfahren (seit den 80er)

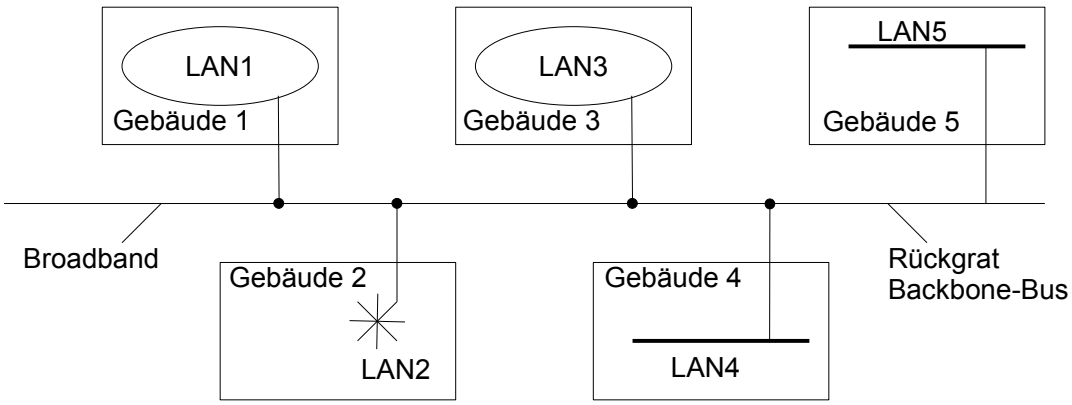


Frequenzen



Sicherheitsband: diese Frequenzen werden nicht genutzt

Kostenaufwendig



3.6.4.5 Kommunikationsregeln

ISO / OSI-7-Schichten-Referenzmodell

ISO's OSI-Modell

7	Application	Anwendung
6	Presentation	Darstellung
5	Session	Sitzung
4	Transport	Transport
3	Network	Netzwerk
2	Data Link	Datensicherung
1	Physical	Bitübertragung

1. Physikalische Schicht (Physical Layer)

   Die erste Schicht regelt die ungesicherte Übertragung der Information als Bitstrom über das Übertragungsmedium:

   Bitübertragungsschicht

   Das Protokoll beinhaltet die elektrischen, die mechanischen und die funktionellen Spezifikationen für das Senden und Empfangen unstrukturierter Bitströme, z.B.

   – elektrische Darstellung der Signale
   – Datenrate
   – Übertragungstechnologie
   – elektrische und mechanische Eigenschaften von Stecker und Buchse
   – Bedeutung der Schnittstellenleitung

   Beispiele:

   X.21, V.24, RS232, RS499, RS485

2. Sicherungsschicht (Link Layer)

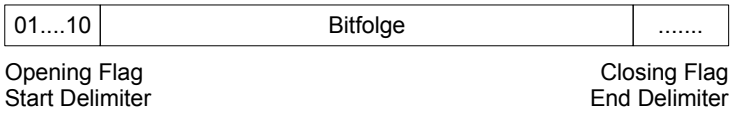
   Aufgabe:

   Bereitstellung einer gesicherten und transparenten Übertragung für die Netzwerkschicht (3. Schicht)

   Grundsätzliche Probleme:

   – Erkennung und eventuell Behebung von Übertragungsfehlern
   – Flusskontrolle
   – Medienzugriff

   Zur Fehlerbehandlung und Flusskontrolle wird die Bitfolge in Rahmen (Frames) unterteilt:



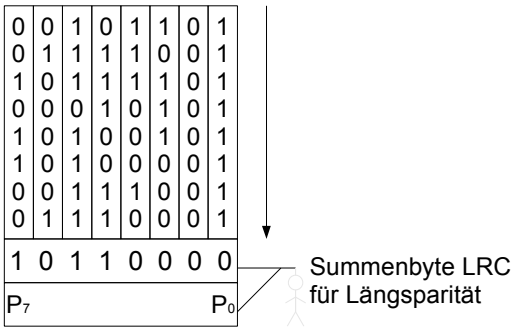
Datensicherung:

– Mit Hilfe eines Paritätsbits pro Zeichen: Querparität

$$\underbrace{01\,\text{dotslow}\,01}_{8\ Bit}\,P$$

Zweifachfehler werden nicht erkannt.

– Am Ende eines Frames wird ein Paritätswort hinzugefügt: Summenbyte LRC (Longitudinal Redundancy Check) für Längsparität



– Bessere Übertragungssicherheit über das Verfahren der zyklischen Blockprüfung zur Erzeugung einer Prüfsumme CRC (Cyclic Redundancy Check)

Flusskontrolle:

Anpassung von Sende–und Empfangsgeschwindigkeit:

– XON-/ XOFF-Protokoll

Verwendung der ASCII–Steuerzeichen

DC1 (11h) → XON

DC2 (13h) → XOFF

Aussenden dieser Steuerzeichen durch den Empfänger:

DC1 → Start der DÜ (XON)

DC2 → Stop der DÜ (XOFF) (DÜ: Datenübertragung)

– Stop-and Wait-Verfahren

Empfangspuffer mit der Speicherkapazität für einen Frame

Sender sendet einen Frame und wartet dann bis er die Freigabe für das Senden des nächsten Frames erhält.

Die Freigabe sendet der Empfänger in Form des Steuerzeichens

ACK (06h)

nach fehlerfreien Empfang des Frames.

Lag ein Übertragungsfehler vor, dann wird stattdessen das Steuerzeichen

NAK (15h)

zurückgesendet.

Folge: Wiederholung des Sendens dieses Frames.

– Fensterverfahren

Festlegung einer Fenstergröße *n*,

$$n = 1,\ 2,\ \ldots$$

Es dürfen maximal *n* Frames ohne ACK des Empfängers gesendet werden.

Der Sender verwendet einen internen Zähler *i*, der mit *i* = 0 initialisiert wird.

Jeder Sendevorgang eines Frames inkrementiert *i* (*i* ++).

Jedes Empfangen eines ACK-Signals dekrementiert *i* (*i* --).

Voraussetzung für das Senden eines Frames: $i < n$.

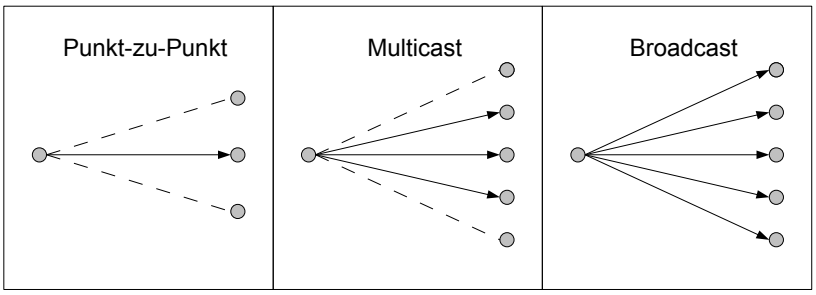
Das Stop-and-Wait-Verfahren ist ein Spezialfall des Fensterverfahrens (n=1).

Medienzugriff:

- CSMA / CD z.B. LAN
- Token Passing z.B. LAN
- Master-Slave z.B. bei Feldbus
- Hybrid-Verfahren (Token Passing und Master-Slave) z.B. bei Feldbus

- Weitere Unterteilung der 2. Schicht in zwei Teilschichten bei LAN's:

1. LLC (Logical Link Control, IEEE802.2):
   - nichtbestätigter verbindungsloser Dienst oder Datagrammdienst
     Die Datenframes werden unabhängig voneinander gesendet, die Reihenfolge muss nicht eingehalten werden.
     Mehrere Übertragungsmöglichkeiten



   - verbindungsorientierter Dienst:
     - Nur Punkt-zu-Punkt-Verbindung
     - Vor einer DÜ muss ein Verbindungsaufbau erfolgen.
     - Jeder Frame muss quittiert werden.
     - Die Reihenfolge eines Frames muss beim Senden beachtet werden.
     - Nach der DÜ muss die Verbindung abgebaut werden.
2. MAC (Media Access Control)

   CSMA / CD (IEEE802.3)

   Token Bus (IEEE802.4)

   Token Ring (IEEE802.5)

Verfahren der zyklischen Blockprüfung (DIN 66219)

Die Bitfolge eines Frames wird als die Koeffizientenfolge eines Polynoms aufgefasst.

z.B. $10010011 \equiv P(x) = x^7 + x^4 + x + 1$

Dividiert man *P(x)* durch ein anderes Polynom, Generatorpolynom *G(x)*, dann erhält man im Allgemeinen einen Teilerrest *R(x)*. Die Polynomdivision erfolgt nach der Methode der Modulo-2-Division.

Die Koeffizienten des Restpolynoms stellen die Prüfsumme CRC dar.

Der Empfänger führt mit der empfangenen Bitfolge (inklusive des CRC) ebenso die Division mit dem selben Generatorpolynom durch.

Ist der $Teilerrest \neq 0$, dann liegt ein Übertragungsfehler vor:

$$R(x) \neq 0 \;\Rightarrow\; Fehler$$

Es gilt **nicht**:

$$R(x) = 0 \;\Rightarrow\; kein\ Fehler$$

sondern:

Mit großer Wahrscheinlichkeit liegt kein Fehler vor.

Division: Modulo-2-Division

Gegeben ist:

Bitfolge:	$P = p_n p_{n-1} \dots p_0$		
Zugehöriges Polynom:	$P(x) = p_n x^n + \dots + p_0$		
Generatorpolynom:	$G(x) = g_m x^m + \dots g_0$	mit:	$g_m = g_0 = 1$
Polynomgrad:	$P(x)$: n		
	$G(x)$: m		

Erweiterung von $P(x)$ zu $P'(x)$ mit

$$P'(x) = x^m \cdot P(x)$$
$$Grad(P'(x)) = n + m$$

Bitfolge: $P' = p_n \dots p_0 \underbrace{0 \dots 0}_{m-1 \dots 0}$

Mit Hilfe des Generatorpolynoms kann P'(x) folgendermaßen aufgeteilt werden:

$$P'(x) = N(x) \cdot G(x) + R(x)$$
$$Grad(N(x)) = n$$
$$Grad(R(x)) < m$$

$$P'(x) - R(x) = N(x) \cdot G(x)$$

$P'(x) - R(x)$ ist durch $G(x)$ ohne Rest teilbar.

$$P'(x) - R(x) = p_n x^{n+m} + \dots + p_0 x^m \underbrace{- r_{m-1} x^{m-1} - \dots - r_0}_{-R(x)}$$

$$P' - R: \; p_n \dots p_0 \underbrace{(-r_{m-1}) \dots (-r_0)}_{entspricht:\ CRC}$$

Wenn auf den Empfangsseite die Division von $P'(x) - R(x)$ nicht teilerfrei ist, dann liegt ein Übertragungsfehler vor.

Beispiel:

$$P(x)=x^2+2x+1 \qquad Grad(P)=n=2$$

$$G(x)=x-1 \qquad Grad(G)=m=1$$

$$P'(x)=P(x)\cdot x^m=P(x)\cdot x=x^3+2x^2+x \qquad Grad(P')=n+m=3$$

Koeffizientenfolgen:

$P=$		1	2	1
$G=$			1	-1
$P'=$	1	2	1	0

$$P'(x):G(x)=(x^3+2x^2+\ \ x+0):(x-1)=\overbrace{x^2+3x+4}^{N(x)}+\underbrace{\frac{\overbrace{4}^{R(x)}}{x-1}}_{G(x)}$$

$$\begin{array}{r} \underline{x^3-\ \ x^2} \\ 3x^2+\ \ x \\ \underline{3x^2-3x} \\ 4x+0 \\ \underline{4x-4} \\ 4 \quad \leftarrow R(x) \end{array}$$

$$P'(x)-R(x)=x^3+2x^2+x-4$$

1 2 1 -4 $\rightarrow$ DÜ

Auf der Empfängerseite erfolgt eine Polynomdivision mit dem selben Generatorpolynom:

$$(P'(x)-R(x)):G(x)=(x^3+2x^2+1-4):(x-1)=x^2+3x+4$$

Da die Division bei diesem Beispiel keinen Teilerrest ergab, kann angenommen werden, dass die DÜ mit hoher Wahrscheinlichkeit fehlerfrei war. Die Wahrscheinlichkeit der Fehlerfreiheit hängt von der übertragenen Datenmenge und der Größe des Generatorpolynoms ab (s. Tabelle auf Seite 54).

Auf der Menge $\{0;\ 1\}$ müssen folgende Operationen erklärt werden, damit diese Polynomdivision und -subtraktion möglich ist:

Addition und Multiplikation

Addition

$(a\,,\,b)\rightarrow c \quad oder \quad a+b\rightarrow c \qquad$ mit $a\,,\,b,\,c\in\{0,\,1\}$

Funktionstabelle für die Addition

0	+	0	=	0
0	+	1	=	1
1	+	0	=	1
1	+	1	=	0

Eigenschaften:

(1) kommutativ

(2) assoziativ

(3) 0 ist das neutrale Element der Addition

(4) jedes Element ist zu sich selbst invers $\quad (\; a+a=0 \;\Rightarrow\; a=-a \;)$

Multiplikation

$(a\,,\,b)\rightarrow c \quad oder \quad a\cdot b\rightarrow c \quad oder \quad a\times b\rightarrow c$

Funktionstabelle für die Multiplikation

0	·	0	=	0
0	·	1	=	0
1	·	0	=	0
1	·	1	=	1

Eigenschaften:

(1) kommutativ

(2) assoziativ

(3) distributiv $\quad a(b+c)=a\,b+a\,c$

(4) 1 ist das neutrale Element der Multiplikation

(5) Existenz des inversen Elements x für alle $\quad a\in\{0,\,1\}$ mit $\quad a\neq 0:$

$a\cdot x=1 \;\rightarrow\; x=1\;\textit{für}\;a=1$

Mengen mit diesen Operationen und diesen Eigenschaften ((1), ..., (4) bei Addition und (1), ..., (5) bei Multiplikation) bezeichnet man als Körper.

Die Addition lässt sich als EXOR-Verknüpfung darstellen,

die Multiplikation als UND-Verknüpfung.

Insbesondere gilt:

$$-R(x)=+R(x)$$

Die Division mit Polynomen mit den Koeffizienten 0, 1 bezeichnet man als Modulo-2-Division

Bsp.: Gegeben:

$$G(x)=x^8+x^2+x+1\equiv G=1\,0000\,0111$$

$$Grad(G)=m=8$$

zu übertragende Information: $1001\,0011$

$$P(x)=x^7+x^4+x+1$$

$$Grad(P)=7$$

$$P'(x)=P(x)\cdot x^8$$

$$P'=1001\,0011\,0000\,0000$$

```
- 1 0 0 0 0 0 1 1 1                                  1
  0 0 1 0 0 0 0 1 0
- 0 0 0 0 0 0 0 0 0                                  0
  0 1 0 0 0 0 1 0 0
- 0 0 0 0 0 0 0 0 0                                  0
  1 0 0 0 0 1 0 0 0
- 1 0 0 0 0 0 1 1 1                                  1
  0 0 0 0 1 1 1 1 0
- 0 0 0 0 0 0 0 0 0                                  0
  0 0 0 1 1 1 1 0 0
- 0 0 0 0 0 0 0 0 0                                  0
  0 0 1 1 1 1 0 0 0
- 0 0 0 0 0 0 0 0 0                                  0
  0 1 1 1 1 0 0 0 0
- 0 0 0 0 0 0 0 0 0                                  0
  1 1 1 1 0 0 0 Rest = CRC
```

Zu übertragen ist die Nutzinformation mit angehängtem CRC:

$$\underbrace{1001\,0011}_{Nutzinfo}\ \underbrace{1111\,0000}_{CRC}$$

$$P'-R=P'+R$$

P':	1001	0011	0000	0000	
$\pm R$:				1111	0000
	1001	0011	1111	0000	

Überprüfung einer fehlerhaften DÜ: 4-fach-Fehler für P mit CRC:

1001	0011	⇒	100**0**	0**1**11
1111	0000	⇒	111**0**	0**1**00

```
( 1 0 0 0 0 1 1 1 1 1 1 0 0 1 0 0 ) : ( 1 0 0 0 0 0 1 1 1 ) = 1 0 0 0 0 1 0 0 + Rest/G
- 1 0 0 0 0 0 1 1 1
        1 0 0 0 1 1 0 0 1
      - 1 0 0 0 0 0 1 1 1
            1 1 1 1 0 0 0   !=   0   =>Fehler
```

Entwicklung eines Divisionsalgorithmus (Divisionsschaltung):

$$P(x)=p_0+\ldots+p_n x^n \qquad Grad(P)=n$$

$$G(x)=g_0+\ldots+g_m x^m \quad \text{mit} \quad g_0=g_m=1 \qquad Grad(G)=m$$

$$P'(x)=P(x)\cdot x^m=p_0 x^m+\ldots+p_n x^{m+n} \qquad Grad(P')=m+n$$

$$P'(x)-R(x)=\underbrace{N(x)\cdot G(x)}_{Q(x)} \qquad Grad(N)=n$$

$$Q(x)=N(x)\cdot G(x) \qquad Grad(Q)=m+n$$

$$=q_0+\ldots+q_{m+n}\cdot x^{m+n}$$

$$=r_0+\ldots+r_{m-1}x^{m-1}+p_0 x^m+\ldots+p_n x^{m+n}$$

Koeffizientenfolge:

$$P=(p_0,\ldots,p_n,0,\ldots)$$

$$P'=(0,\ldots,0,\underset{0}{p_0},\ldots,\underset{m-1}{p_{m-1}},\underset{m}{p_m},\ldots,\underset{m+n}{p_n},0,\ldots)$$

$$Q=(q_0,\ldots,q_{m+n},0,\ldots)=(r_0,\ldots,r_{m-1},p_0,\ldots,p_n,0,\ldots)$$

$$G=(g_0,\ldots,g_m,0,\ldots) \quad \text{mit} \quad g_0=g_m=1$$

$$R=(r_0,\ldots,r_{m-1},0,\ldots)$$

$$N=(N_0,\ldots,N_n,0,\ldots)$$

Es gilt:

$$Q=P'-R=P'+R=N\cdot G$$

$$Q=(q_0,\ldots,q_{m+n},0,\ldots)=(N_0,\ldots,N_n,0,\ldots)\cdot(g_0,\ldots,g_m,0,\ldots)$$

Die Multiplikation der Polynome N und G und die Zusammenfassung aller Koeffizienten zur selben Potenz von x ergibt für q_0 bis q_{m-1}:

$$q_0=r_0=N_0\cdot g_0=N_0$$

$$q_1=r_1=N_1\cdot g_0+N_0\cdot g_1=N_1+N_0\cdot g_1$$

$$q_2=r_2=N_2\cdot g_0+N_1\cdot g_1+N_0\cdot g_2=N_2+N_1\cdot g_1+N_0\cdot g_2$$

$$\vdots$$

$$\boxed{q_i=r_i=N_i+\sum_{j=1}^{i}N_{i-j}\cdot g_j} \qquad für\ i=1,\ldots,m-1$$

und $\boxed{r_0=N_0}$

für q_{m+n} *bis* q_m: *setze* $n+m=\lambda$

$$q_\lambda=p_n=g_m\cdot N_n=N_n$$

$$q_{\lambda-1}=p_{n-1}=g_m\cdot N_{n-1}+g_{m-1}\cdot N_n=N_{n-1}+g_{m-1}\cdot N_n$$

$$q_{\lambda-2} = p_{n-2} = g_m \cdot N_{n-2} + g_{m-1} \cdot N_{n-1} + g_{m-2} \cdot N_n = N_{n-2} + g_{m-1} \cdot N_{n-1} + g_{m-2} \cdot N_n$$

$$\vdots$$

$$q_{\lambda-i} = p_{n-i} = N_{n-i} + \sum_{j=1}^{i} g_{m-j} \cdot N_{n-i+j}$$

setze $n-i=i'$*:*

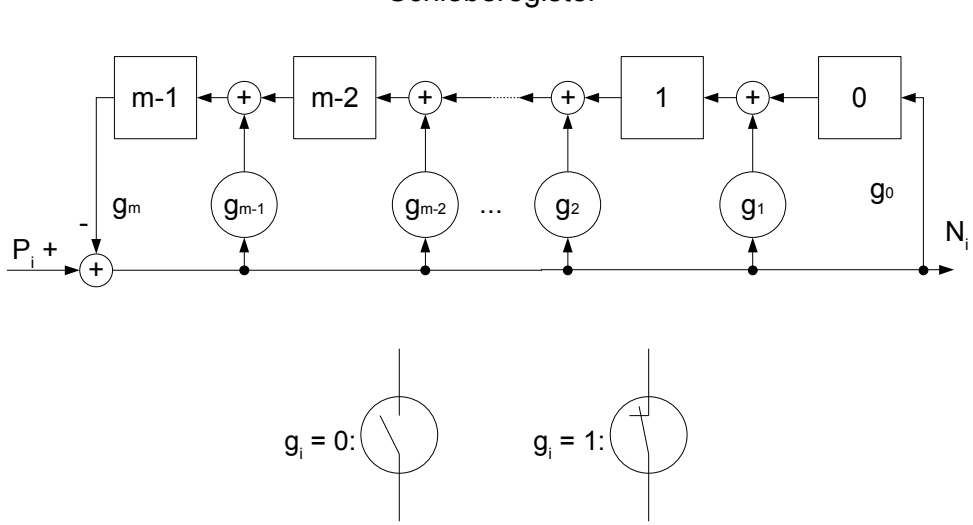
$$p_{i'} = N_{i'} + \sum_{j=1}^{n-i'} g_{m-j} \cdot N_{i'+j} \qquad \textit{mit } i' = n-1, \ldots, 0$$

setze $i'=i$*:*

$$\boxed{p_i = N_i + \sum_{j=1}^{n-i} g_{m-j} \cdot N_{i+j}} \qquad \textit{für } i = n-1, \ldots, 0$$

und $\boxed{p_n = N_n}$

Divisionsschaltung für $N = \frac{P}{G}$, $g_0 = g_m = 1$



Vor der Übertragung ist das Schieberegister mit Null initialisiert.

Beispiel:

$P=(p_0\ p_1\ p_2\ p_3\ p_4)$ $n=4$

$G=(g_0\ g_1\ g_2\ g_3)$ $n=3$ $g_{0,}\ g_n$ *müssen immer 1 sein*

2. Formel anwenden: $N_4=p_4$

$$N_i=p_i-\sum_{j=1}^{4-i} g_{3-j}\cdot N_{i+j} \qquad i=3,\ \ldots,\ 0$$

$N_4=p_4$

$N_3=p_3-g_2\ N_4$

$N_2=p_2-(g_2\ N_3+g_1\ N_4)$

$N_1=p_1-(g_2\ N_2+g_1\ N_3+N_4)$

$N_0=p_0-(g_2\ N_1+g_1\ N_2+N_3)$ $g_{n+1}=g_{-1}=0$

1. Formel anwenden: $r_i=N_i+\sum_{j=1}^{i} N_{i-j}\cdot g_j$ $i=2,\ 1$

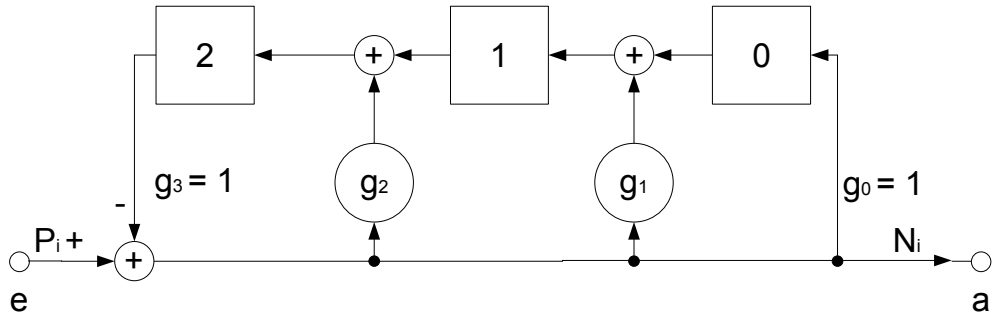
$r_0=N_0$

$r_2=N_2+(g_1\ N_1+g_2\ N_0)$

$r_1=N_1+g_1\ N_0$

$r_0=N_0$

Divisionsschaltung:



taktweises Durchschieben von P_i

i	e	N_i	2	1	0
			0	0	0
4	p_4	p_4	g_4	g_4	N_4
3	p_3	$p_3 - g_4$	$g_4 g_3$	$g_3 N_4$	N_3
2	p_2	$p_2 - g_3 g_4$	$g_3 g_2 N_4$	$g_2 N_3$	N_2
1	p_1	$p_1 - (g_3 g_2 N_4)$	$g_2 g_1 N_3$	$g_1 N_2$	N_1
0	p_0	$p_0 - (g_2 g_1 N_3)$	$g_1 g_0 N_2$	$g_0 N_1$	N_0
			r_2	r_1	r_0

Nach der Division muss für die DÜ lediglich der Inhalt des Schieberegisters (CRC) der Nutzinformation P nachgeschoben werden.

Auf der Empfangsseite wird nach der Nutzinformation P außerdem das mitgesandte Restpolynom CRC durchgeschoben:

i	e	a	r_2	r_1	r_0
	r_2	0	r_1	r_0	0
	r_1	0	r_0	0	0
	r_0	0	0	0	0

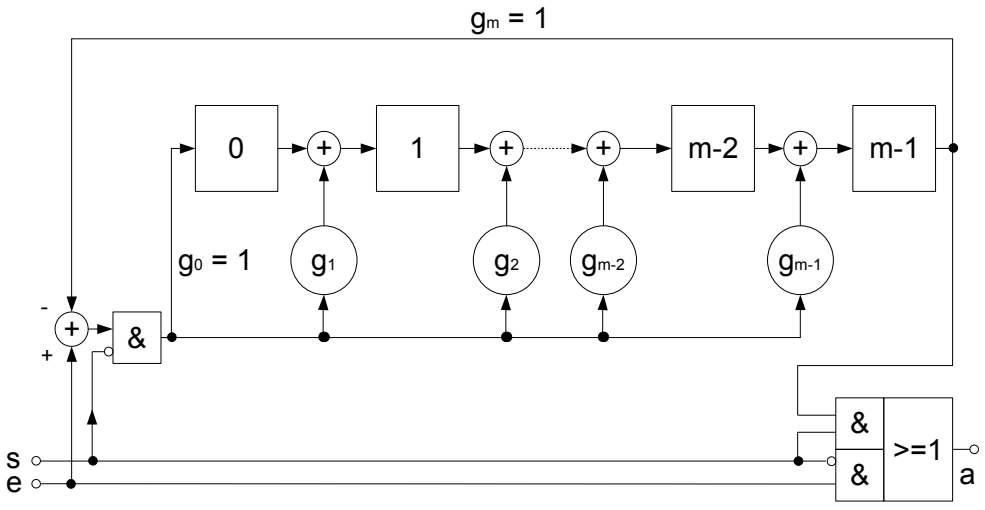
Die letzten Werte von r_0, r_1 und r_2 sind der „Divsionsrest beim Empfänger“.

Schaltung:

Verwendung eines Steuersignals:

$s=0$: Die Nutzinformation wird unverändert weitergeleitet. Begleitend erfolgt die Modulo-2-Division.

$s=1$: Der Inhalt des Schieberegisters (CRC) wird nachgeschoben.



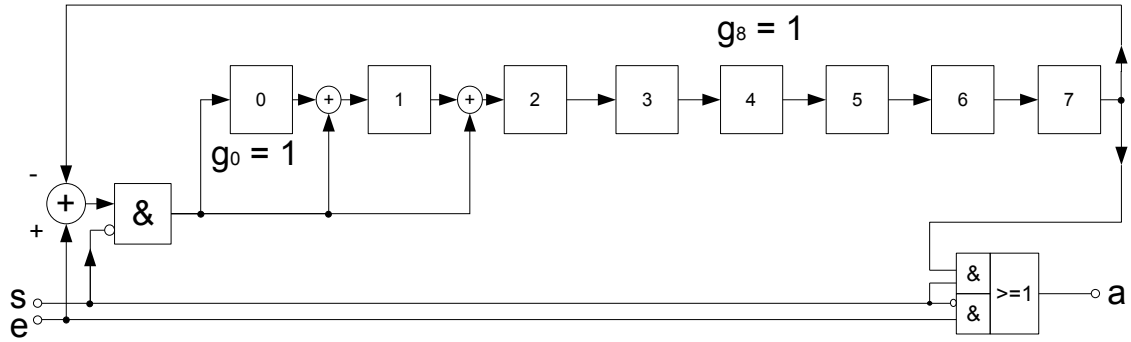
Für einige in der Praxis verwendete Generatorpolynome ergeben sich in Abhängigkeit einer Hammingdistanz von d=3 bzw. 4 folgende maximal zu übertragende Blocklängen:

$G(x)$		max. Blocklänge	
		$d=3$	$d=4$
PDV:	x^8+x^2+x+1	2^8-9	2^7-1
CCITT:	$x^{16}+x^{12}+x^5+1$	$2^{16}-17$	$2^{15}-1$
IEEE 802:	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+$ $+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$	$2^{32}-33$	$2^{31}-1$

Aufgabe:

Entwickle eine CRC-Schaltung für $G(x)=x^8+x^2+x+1$:

Lösung:



4 Prozessrechnersysteme

digitale, programmierbare Gerätesysteme zur Automatisierung technischer Prozesse

Ursprung: klassischer Prozessrechner

Minirechner mit geeigneter Erweiterung:

- Schnittstellen für den Anschluss an den technischen Prozess
- Echtzeitbetriebssystem

Entwicklung der Halbleitertechnologie ermöglichte eine Miniaturisierung und damit auch eine Spezialisierung für verschiedene Anwendungsbereiche

Prozessleitsystem (PLS)		Fertigungsleitsystem (FLS)		Verkehrsleitsystem	Gebäudeleitsystem
SPS, PR	SPS, PR	CNC-St, SPS	Roboter-St, SPS		SPS
SPS	SPS	NC-St, SPS	SPS	Ampelsteuerung	Heizungs-St
Kraftwerkstech.	**Verfahrenstech.**	**Fertigungstech.**	**Montagetech.**	**Verkehrstech.**	**Gebäudetech.**

St: Steuerung
NC: Numerical Control
CNC: Computerized NC
SPS: Speicherprogrammierbare Steuerung

Grund für diese Spezialisierung:

unterschiedliche Aufgabenstellungen, z.B.

Montagetechnik:

- Roboter-Steuerung mit Visionsystem (DBV-System) (digitale Bildverarbeitung)
- komplexe Rechenaufgaben
- schnelle Reaktion

Kraftwerkstechnik:

- Tausende von binären und analogen Signalen
- ausgedehnter regionaler Bereich
- DÜ über Busse oder LAN's
- Informationsdarstellung
- großes Gefahrenpotential erfordert hohe Zuverlässigkeit

wünschenswert:

Entwicklung nach Grundfunktionen:

Messen, Steuern, Regeln } MSR-Technik
Übertragen
Überwachen
Schützen
Informieren

Teilweise ist dies geglückt, z.B. bei Siemens mit:

S7 (Fertigungstechnik, Montagetechnik)
M7 (Verfahrenstechnik) } Migration

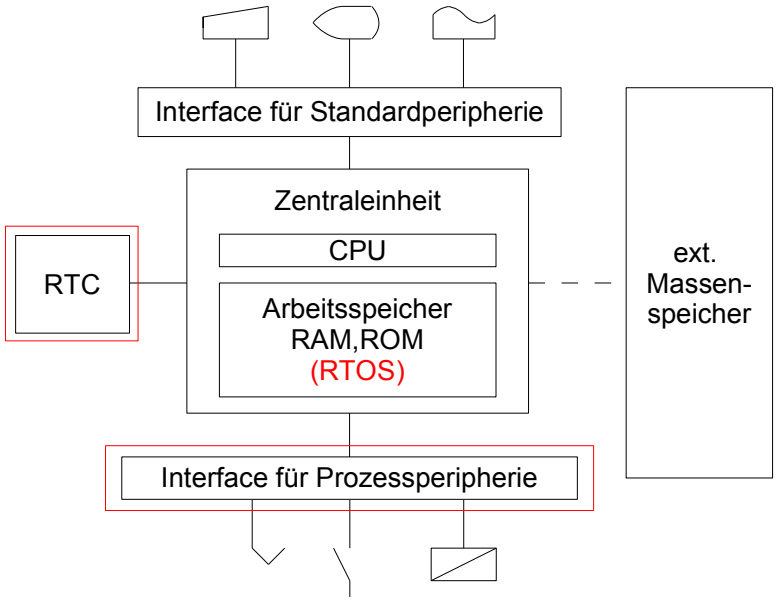
Beispiele von Prozessrechnersystemen:

klassischer Prozessrechner	hohes Leistungsvermögen bereichsübergreifend kostenaufwendig
Mikrocomputersysteme	modulare Rechner mittleres bis hohes Leistungsvermögen bereichsübergreifend kostengünstiger
Mikroprozessor-Baugruppensysteme	sehr stark modularisiert Modul: bestückte Platine für unterschiedliche Aufgaben Verbindung der Module über Standardbusse (z.B. VME-Bus) bereichsübergreifend Spezialkenntnisse bei Hard- und Software erforderlich
Kompaktcomputer	Kleinrechner für einfache MSR-Aufgaben
SPS (Speicherprogrammierbare St)	modulares Rechnersystem mit hoher Zuverlässigkeit niedriger bis hoher Leistungsbereich bereichsübergreifend einfache Bedienung
IPC (Industrie-PC)	höherwertiger Aufbau Standard-SW ist verwendbar
NC / CNC-Steuerung	Steuerung für Werkzeugmaschinen stark spezialisiert für die Fertigungstechnik
Roboter-Steuerung	Steuerung für Handhabungsautomaten (Roboter) stark spezialisiert für die Montagetechnik
Fernwirksystem	Rechnersystem für Energieverteilungssysteme
PC, Workstation	Digitalrechner Einsatzbereiche: unterschiedliche Leitsysteme, Büroautomatisierung

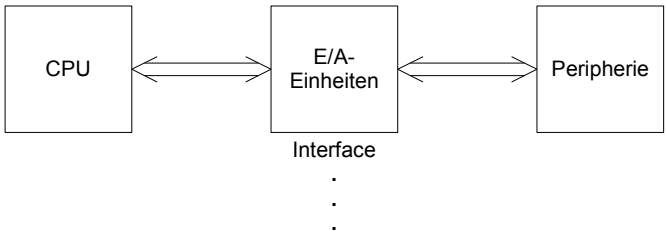
Aufbau eines Prozessrechners (PR):

Digitalrechner mit HW- und SW-Erweiterung:

Prozesskopplung (Interfaceschaltungen)
Betriebssystem mit Echtzeitverhalten (RTOS: Real Time Operating System):
Die Reaktionszeit eines PR's muss immer kleiner als die prozessbedingte Bearbeitungszeit sein.
Echtzeituhr (RTC: Real Time Clock) zur Synchronisierung von Rechenprozessen und technischen Prozessen



Ein-Ausgabe-Schnittstellen



Aufgaben der E/A-Einheiten:

1. Physikalische Signalanpassung, z.B.
   analog-digital-Wandlung
   Spannungspegelanpassung
   Leistungsanpassung
   parallel-seriell-Wandlung
   Synchronisieren (Geschwindigkeitsanpassung)
2. Organisation der E/A zur Entlastung der CPU
   Einteilung:
   a) nichtprogrammierbare Interface-Schaltungen
      Betriebsweise ist durch die HW-mäßige Schaltung definiert.
      Änderung an eine veränderte Betriebsweise ist nicht möglich.
      Die Organisation der E/A verbleibt im wesentlichen bei der CPU.
      preiswert
   b) programmierbare Interface-Schaltungen, Controller
      hochintegrierte Bausteine
      Betriebsweise kann SW-mäßig an veränderte Situationen während des Betriebs angepasst werden.
      Die Organisation der E/A kann von diesen Controllern weitestgehend autonom wahrgenommen werden.
      Die CPU wird von den Interfaceschaltungen stark entlastet.

      Beispiele von programmierbaren Interface–Schaltungen:

      USART (für die serielle DÜ)
      par. E/A–Bausteine
      DMA–Controller
      Interrupt–Controller
      LAN–Controller
      Feldbus–Controller
      ...

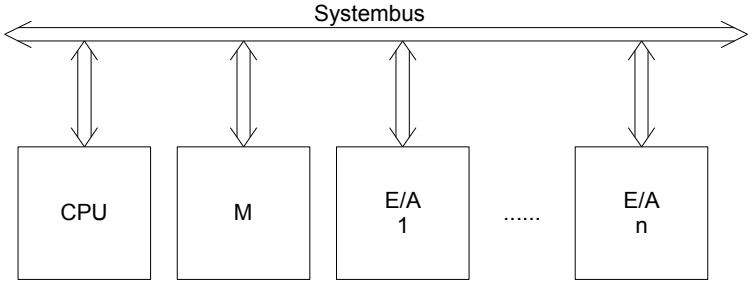
Rechnerseitige Ankopplung der E/A-Einheiten:

Bussysteme, paralleles Leitungsbündel zur Übertragung von Daten, Adressen, Stellsignalen und Meldesignalen

Einteilung in:

Universalbus
Mehrfachbus

<u>Universalbus:</u>



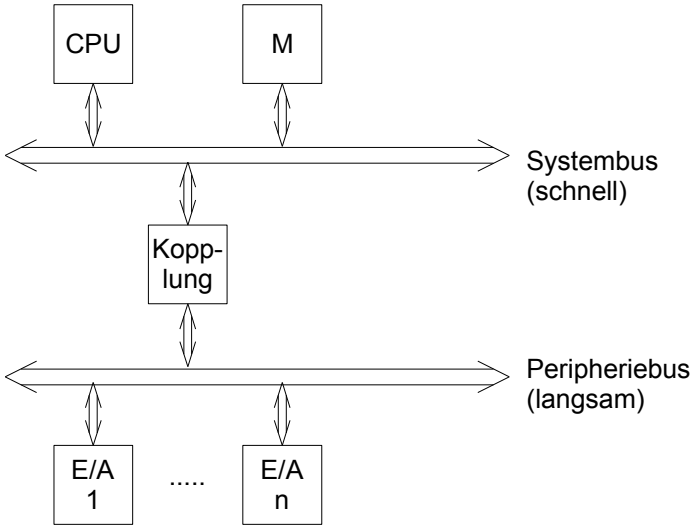
Vorteil: einfache Struktur, preiswert

Nachteil: Der Adressraum muss auf Speicher und E/A–Einheiten aufgeteilt werden.
Es können nicht mehrere Datenübertragungen gleichzeitig laufen.

typ. Vertreter: ECB, SMP, Multibus I

Bei leistungsfähigeren Computersystemen trifft man den Mehrfachbus:

Zweifachbus:



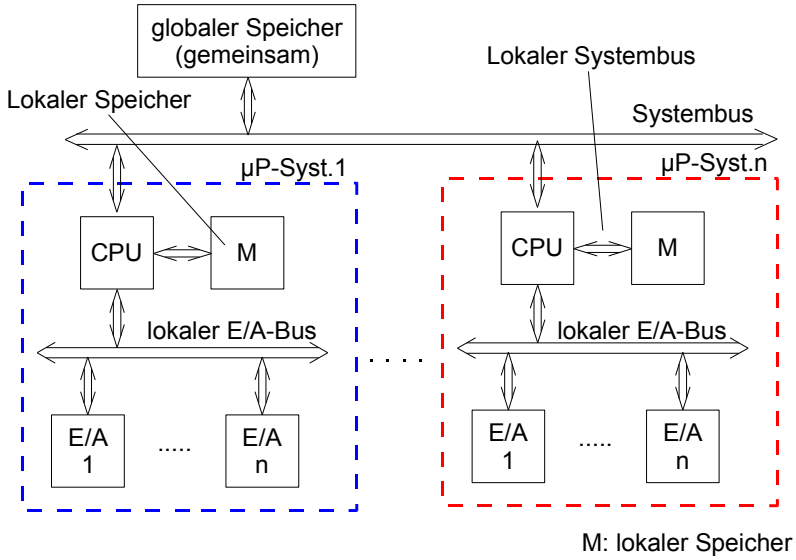
Vorteil: Durch die Entkoppelung können Übertragungen gleichzeitig über den System- und den Peripheriebus stattfinden.
Busbreite kann unterschiedlich sein.
Die meist langsame Übertragungsgeschwindigkeit des Peripheriebus muss den Systembus nicht belasten.

Nachteil: höherer Aufwand

typ. Vertreter:

PC-Systeme: ISA, EISA, PCI
Prozessrechner : PDP11, IBM3090, VME-Systeme (Motorola), Multibus II (Intel)

Dreifachbus bei Multiprozessorsystemen:

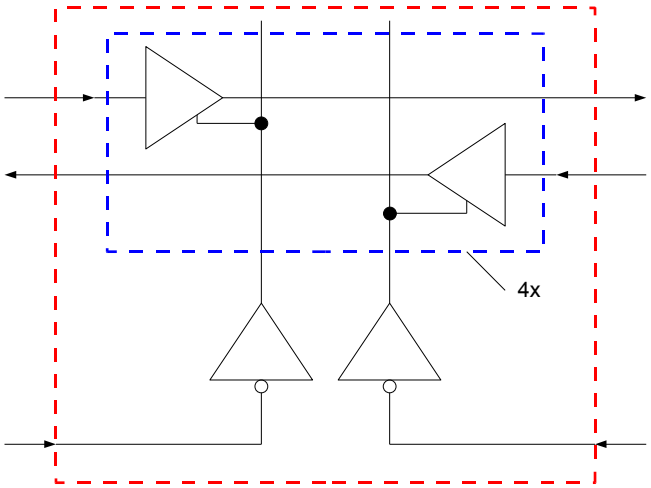


Zusätzlich zu den parallelen Peripheriebus-Systemen gibt es auch serielle Busse, z.B.
USB bei PC–Systemen
VMS bei VME-Systemen

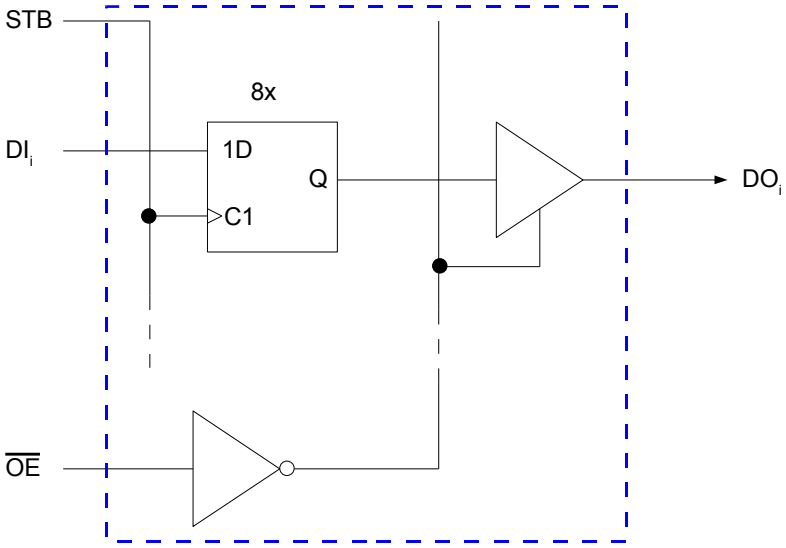
E/A-Einheiten

Prinzipschaltungen von nicht programmierbaren Schnittstellenbausteinen:

Busankoppelbaustein, z.B. 74LS244

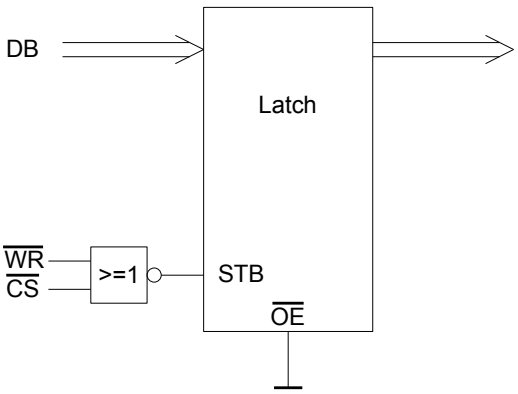


Puffer oder Latch



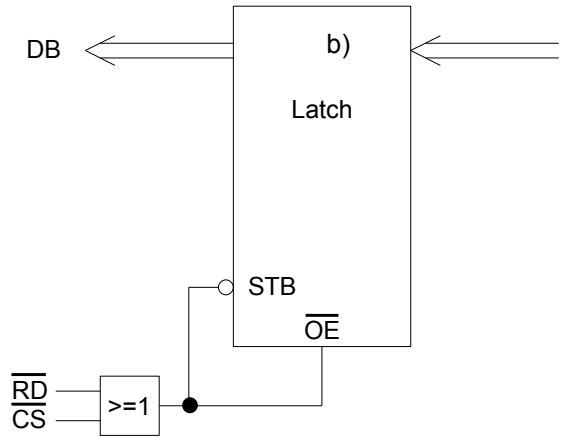
OE: Output enable

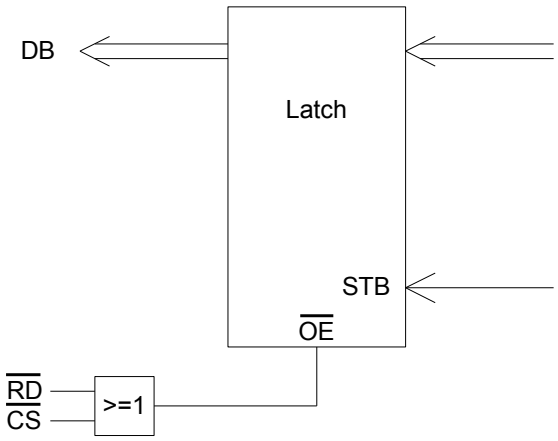
Ausgangsschaltung:



Eingangsschaltungen:

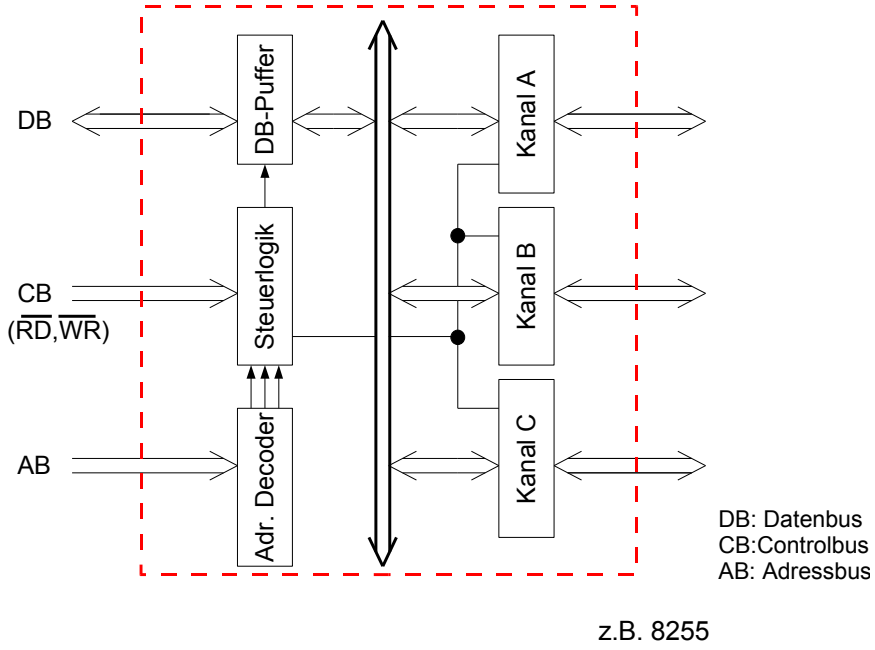
a)





Prinzipschaltungen von programmierbaren Schnittstellenbausteinen:

E/A-Interface für parallele E/A:



z.B. 8255

Interruptsteuerungen:

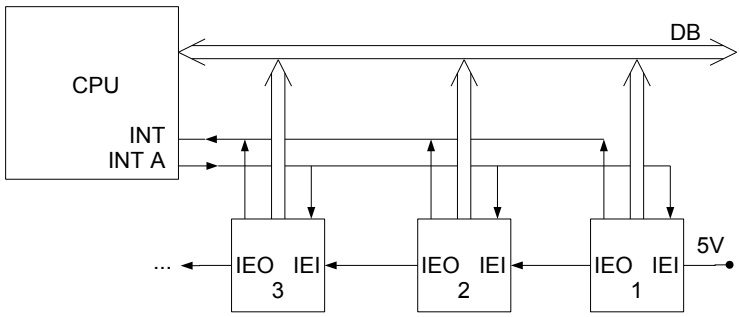
a) CPU-intern, z.B. beim 8085 (RST5.5 bis RST7.5, TRAP)

Vorteil: schnelle Reaktion
Nachteil: nicht ausbaufähig

b) externe Interruptsteuerungen

b1) Daisy Chain-Methode (Gänseblümchenkette-Methode)

Daisy Chain Methode



IEI: Interrupt Enable Input
IEO: Interrupt Enable Output

Interruptanforderung von Gerät i ist dann möglich, wenn IEI_i=1.

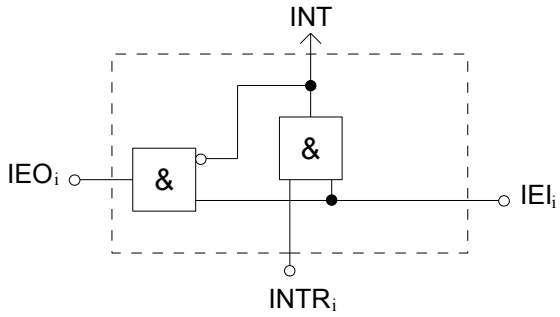
Die Interruptanforderung wird gesperrt, wenn IEI = 0

Wenn ein Gerät i einen Interrupt bei der CPU beantragt hat, setzt es gleichzeitig seinen IEO auf Null und sperrt damit seinen linken Nachbarn.

IEO wird auf Null gesetzt, wenn IEI = 0 ist.

IEO ist immer 1, wenn IEI = 1 und kein Interrupt beantragt wird.

Logik der Interruptsteuerung:



Nach INT = 1 unterbricht die CPU das laufende Programm, bestätigt die Unterbrechung mit INTA = 1.

Das anfordernde Gerät übergibt dann der CPU einen CALL-Befehl für die zugehörige Interrupt Service Routine (ISR)

"CALL adr_i "

adr_i : Anfangsadresse der ISR_i

Im Prinzip genügt ein Erkennungscode von Gerät i.

Vorteil:

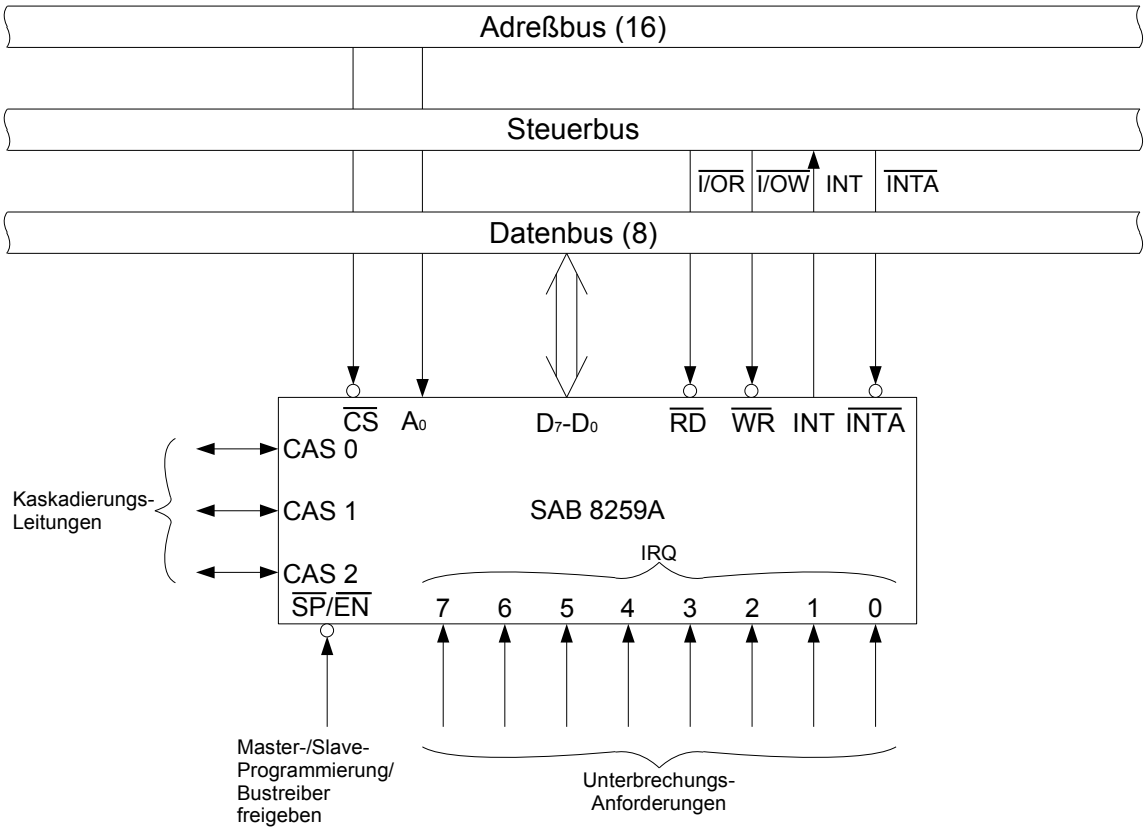
- CPU benötigt nur eine einfache Interrupt-Steuerung
- sehr einfache Realisierung durch Reihung
- keine Begrenzung der Anzahl der Interrupt-Quellen

Nachteil:

- Zeitverlust durch Codeabfrage
- Die Schnittstellenkarten der Geräte müssen daisy-chain-fähig sein.

Programmierbare Interruptcontroller (IPC)

Beispiel: 8259



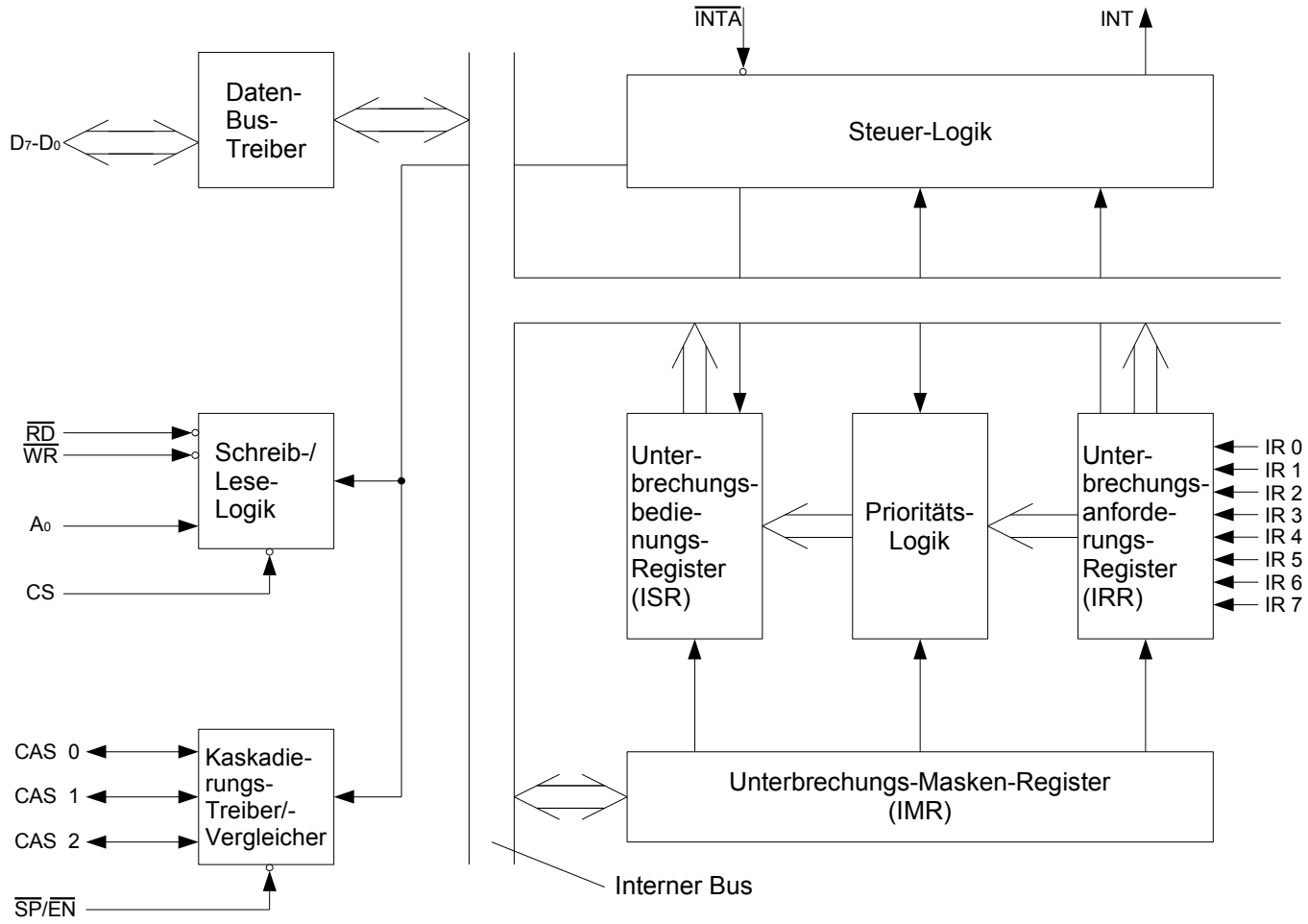
Priorität der IRQ's:

von IRQ_0 bis IRQ_7 absteigend

Priorität ist veränderbar durch Befehle beim Initialisieren an Datenbus

```
MVI   A, COM
OUT   adr
```

$$adr = \underbrace{A_7 \ldots A_1}_{CS}\ A_0$$



Unterbrechungsablauf (Beispiel):

- Eine oder mehrere IRQ's gehen auf H(igh), d.h. beantragen einen Interrupt.
- Die zugehörigen Bits des IRR werden gesetzt.
- Ermittlung des Interrupts mit der höchsten Priorität.
- Der 8259 aktiviert das INT-Signal, d.h. beantragt von der CPU einen Interrupt.
- Die CPU meldet die Unterbrechungsbereitschaft mit dem Bestätigungssignal $\overline{INTA}$.
- Mit dem $\overline{INTA}$ -Impuls wird das zugehörige ISR-Bit gesetzt und das zugehörige IRR-Bit wieder zurückgesetzt. Über den I/O-Treiber wird außerdem der Opcode vom CALL-Befehl ausgegeben.
- Es folgen zwei weitere $\overline{INTA}$ -Impulse mit denen die zugehörige Anfangsadresse adr_i der ISR vom 8259 an die CPU übergeben wird:

  zunächst: $adr_{i\ low}$
  dann: $adr_{i\ high}$

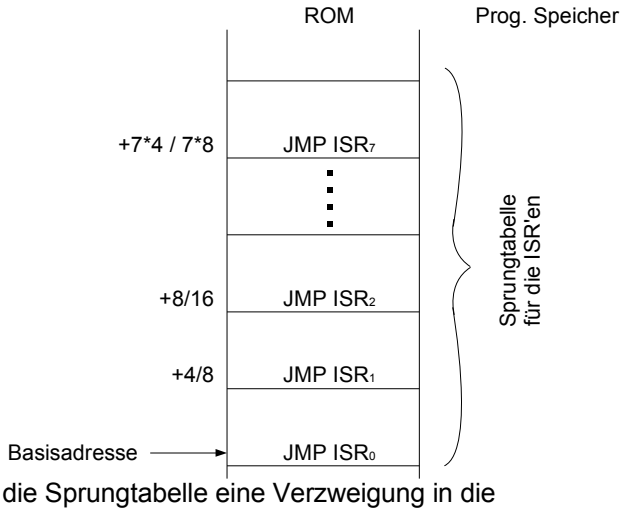
$$adr_i = \underbrace{A_{15} \ldots A_8\ A_7\ A_6 A_5\ 0 \text{dotslow} 0}_{Basisadresse} + \Delta_i$$

$$adr_0 = Basisadresse \quad , \Delta_i = i \cdot 4 \quad bzw. \quad i \cdot 8$$

$$adr_1 = Basisadresse + 4/8$$

$$adr_2 = Basisadresse + 8/16$$

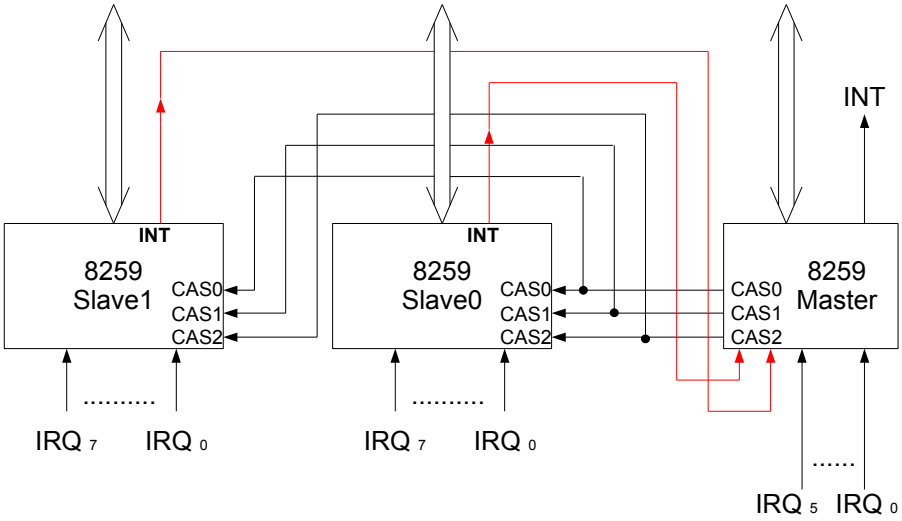
$$\ldots$$



Mit der Übergabe des vollständigen CALL-Befehls erfolgt über die Sprungtabelle eine Verzweigung in die zugehörige ISR.

Die ISR wird vor dem RETURN–Befehl mit einem EOI–Befehl (End Of Interrupt) beendet.
→ ISR-Bit wird wieder zurückgesetzt.

Kaskadiermöglichkeit:

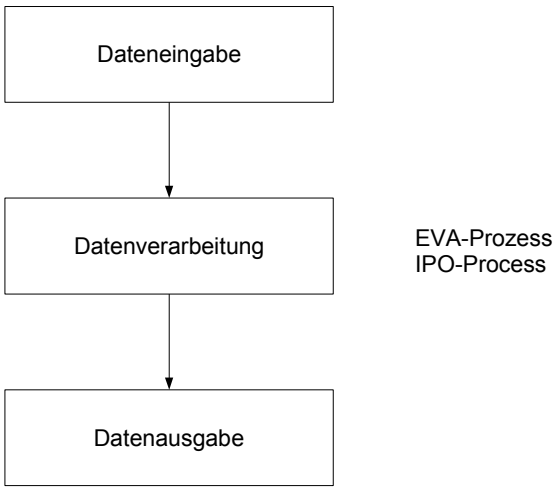


Damit können 6+2*8=22 Interruptquellen bedient werden.

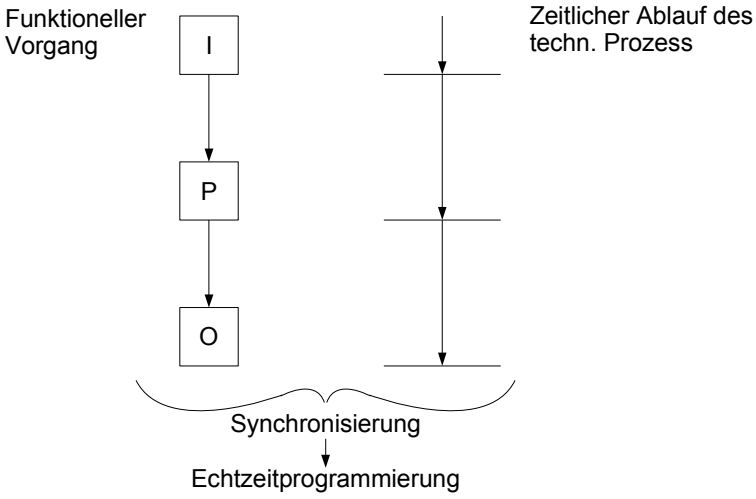
5 Echtzeitprogrammierung

5.1 Echtzeit-Anforderungen

allgemeine Datenverarbeitung:



Einschränkung durch zeitliche Anforderungen:



Echtzeitprogrammierung:

Erstellung von Programmen, so dass bei der Datenverarbeitung im Computer, die zeitlichen Anforderungen an die

- Erfassung der Eingabedaten
- Verarbeitung der Daten
- Ausgabe der Ausgabedaten

erfüllt werden.

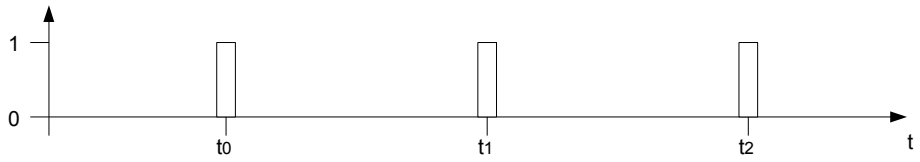
⇒ Forderung nach Rechtzeitigkeit
⇒ Forderung nach Gleichzeitigkeit

5.1.1 Forderung nach Rechtzeitigkeit

Eingabedaten müssen rechtzeitig abgerufen werden.
Ausgabedaten müssen rechtzeitig zur Verfügung stehen.

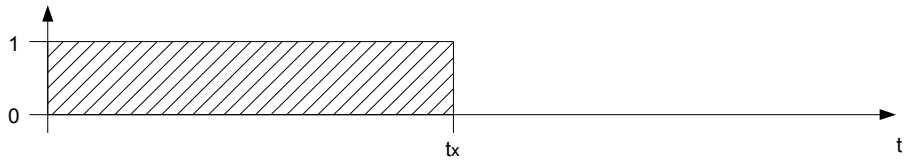
mögliche Ausführungsanforderungen:

– zu bestimmten Zeitpunkten

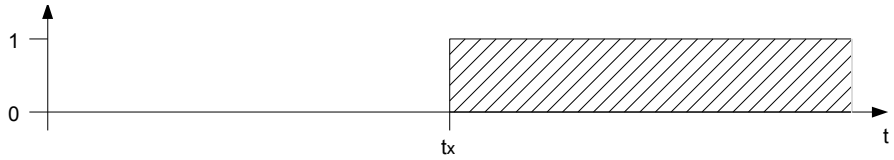


– zu bestimmten Zeitpunkten unter Zulassung von Toleranzintervallen

– bis spätestens zu einem bestimmten Zeitpunkt



– frühestens ab einem bestimmten Zeitpunkt



Zeitvorgaben:

– Absolutzeitbedingung

  Die Zeitangabe bezieht sich auf die Uhr.

– Relativzeitbedingung

  Die Zeitangabe bezieht sich auf stochastisch eintretende Ereignisse.

5.1.2 Forderung nach Gleichzeitigkeit

Das gleichzeitige Ablaufen mehrerer Vorgänge im technischen Prozess erfordert die gleichzeitige Reaktion durch den Rechner.

Ideale Lösung: Jedem Teilprozess wird eine eigene CPU zugeordnet.

Kompromiss: Quasiparallele Bearbeitung der Teilaufgaben durch eine CPU.
Voraussetzung: CPU ist schnell genug.
⇒ simultane Arbeitsweise (Bsp: Simultanschach)

5.2 Echtzeit-Programmierverfahren

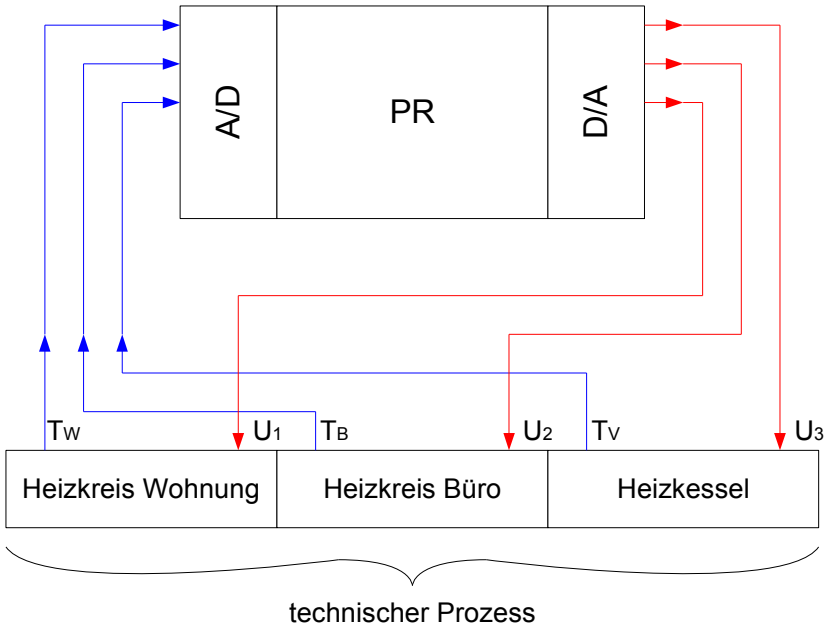
5.2.1 Synchrone Programmierung

Der zeitliche Ablauf der Teilprogramme wird vor der Ausführung festgelegt.

Prinzipielles Vorgehen:

- Die Teilprogramme werden zyklisch abgearbeitet.
- Der zyklische Ablauf wird mit Hilfe einer Echtzeituhr über ein vorgegebenes Zeitraster synchronisiert.
- Die Reihenfolge der Teilprogramme wird vorher festgelegt.

Beispiel:



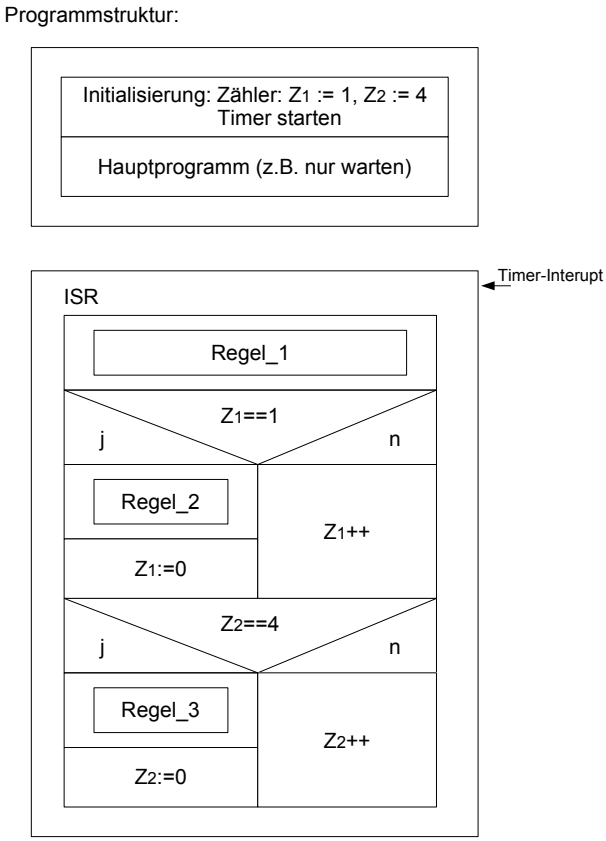
Analyse des Zeitverhaltens:

Zeitraster für den zyklischen Aufruf der Teilprogramme

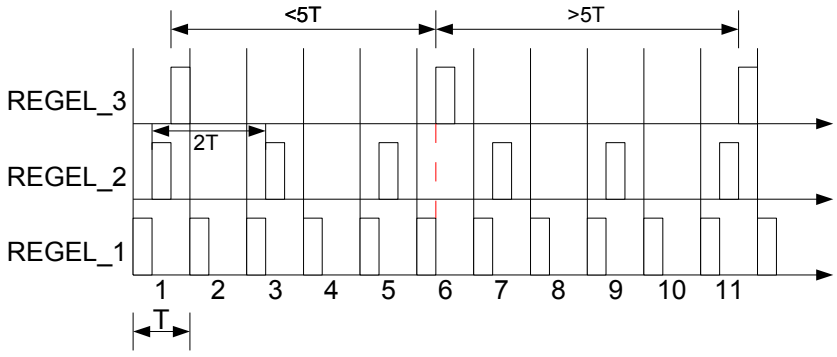
Regelstrecken	Teilprogramme	Abtastzeit in τ
Regelstrecke 1 (Heizkreis Wohnzimmer)	REGEL_1	$\tau_1 = \tau$
Regelstrecke 2 (Heizkreis Büro)	REGEL_2	$\tau_2 = 2 \cdot \tau$
Regelstrecke 3 (Heizkessel)	REGEL_3	$\tau_3 = 5 \cdot \tau$

Echtzeituhr erzeugt nach jeder Zeiteinheit τ einen Interrupt: Zeitraster

Mit jedem Interrupt wird ein Steuerprogramm aufgerufen, das entsprechend den vorgegebenen Zeitbedingungen die jeweiligen Teilprogramme aufruft.



zeitlicher Ablauf der Teilprogramme:



Eigenschaften der synchronen Programmierung:

- Die Forderung nach Rechtzeitigkeit wird nur näherungsweise erfüllt.
- Die Forderung nach Gleichzeitigkeit wird erfüllt, wenn dieses Zeitraster τ klein ist gegenüber den zeitlichen Abläufen im technischen Prozess.
- Das Verfahren eignet sich für zyklische Vorgänge, es ist nicht geeignet für Reaktionen auf nicht vorhersehbare Ereignisse aus dem technischen Prozess.
- Die gesamte Programmstruktur muss geändert werden, wenn die Aufgabenstellung geändert wird. Das Steuerprogramm ist unflexibel.

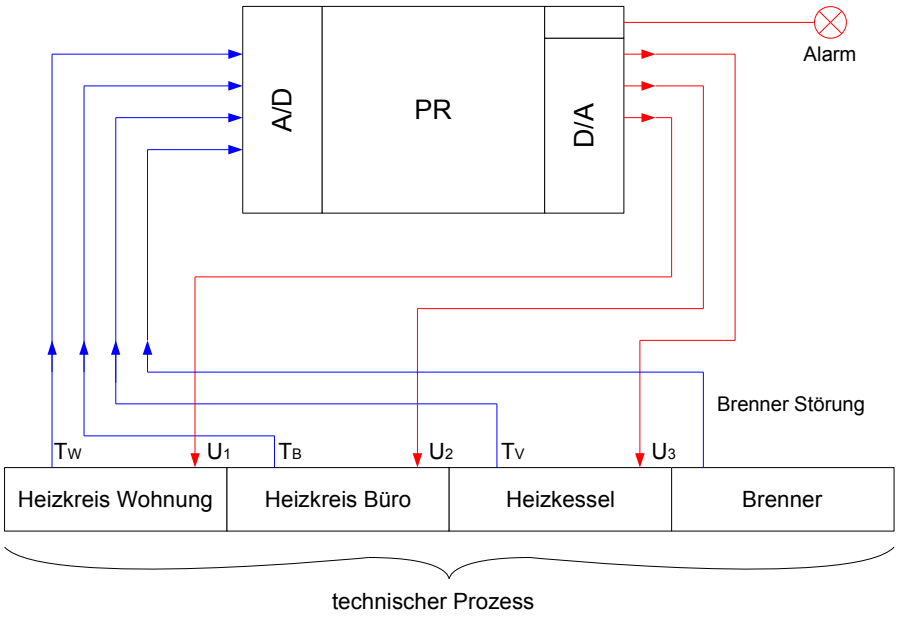
5.2.2 Asynchrone Programmierung oder Parallelprogrammierung

Der zeitliche Ablauf der Teilprogramme wird nicht im Voraus geplant, sondern durch das Organisationsprogramm während des Programmablaufs bestimmt.

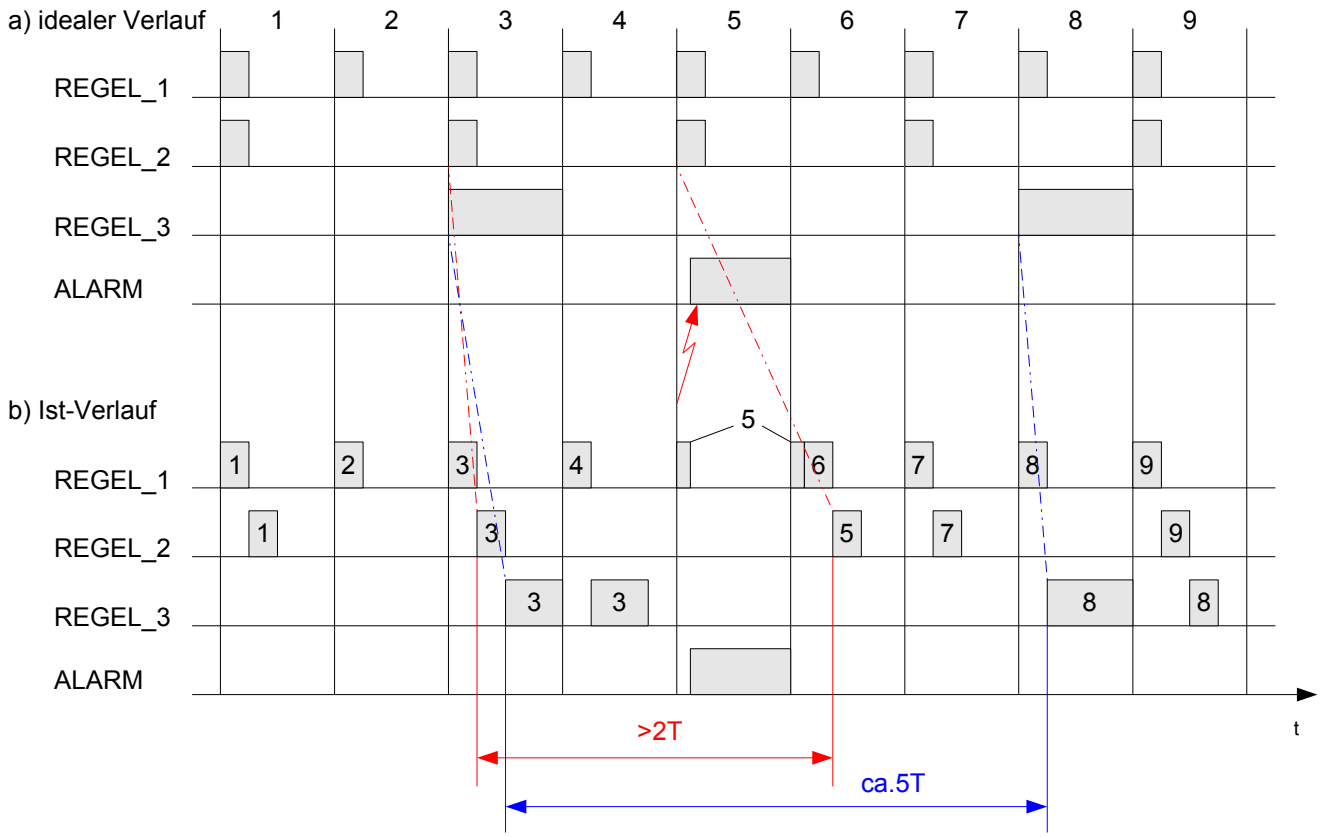
Der Aufruf der Teilprogramme erfolgt nach Zeitbedingungen und Anforderungen aus dem Prozess.

Eine Konfliktsituation wird gelöst durch Vergabe von Prioritäten.

erweitertes Beispiel:



Teilprozess	Teilprogramm	Abtastzeit	Priorität
Brenner Störung	ALARM	-	höchste
Regelstrecke 1 (Heizkreis Wohnung)	REGEL_1	$\tau_1 = \tau$	zweithöchste
Regelstrecke 2 (Heizkreis Büro)	REGEL_2	$\tau_2 = 2 \cdot \tau$	zweitniedrigste
Regelstrecke 3 (Heizkessel)	REGEL_3	$\tau_3 = 5 \cdot \tau$	niedrigste



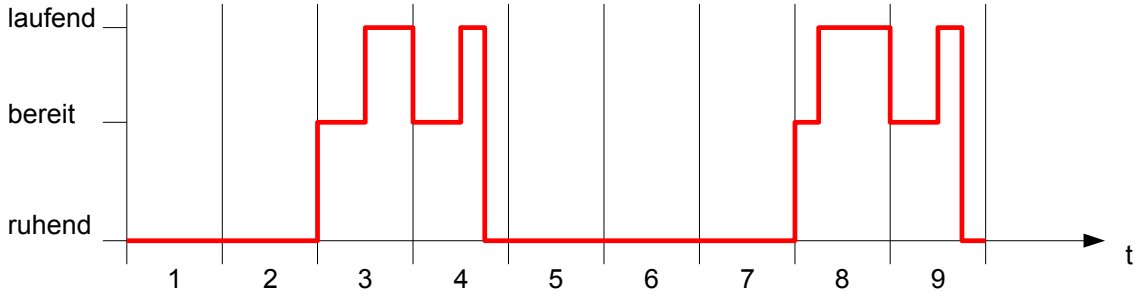
- Die Rechtzeitigkeit ist in einigen Fällen nur näherungsweise erfüllt.
  Je höher die Priorität ist, desto besser wird die Anforderung erfüllt.
- Der Ist-Zeitlauf kann sich gegenüber dem Soll-Zeitlauf stark verschieben.
- Die Aufeinanderfolge der Teilprogramme ist nicht determiniert (s. Raster 6).
  Sie kann durch sporadisch auftretende Ereignisse dynamisch beeinflusst werden.
- Dieses Verfahren zeigt ein günstiges Reaktionsverhalten auf Anforderungen aus dem technischen Prozess.

Rechenprozesse oder Tasks

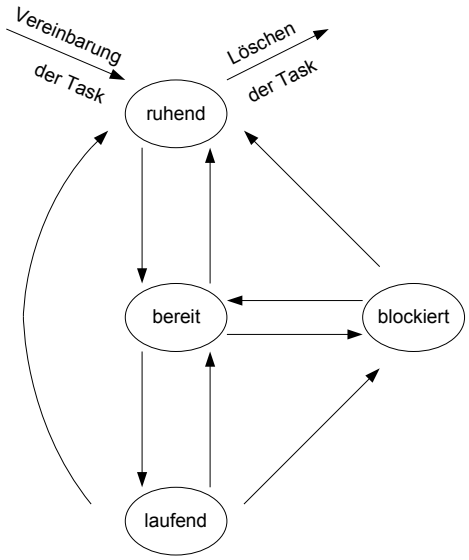
Definition:

Ein Rechenprozess (Task) ist der zeitliche Vorgang der Abarbeitung eines sequentiellen Ablaufs/Programms auf einem Rechner. Ein Rechenprozess existiert nicht nur während der Abarbeitung der Befehle, sondern auch während der Wartezeiten. Er beginnt mit dem Eintrag in eine Liste des Echtzeitbetriebssystems (RTOS: Real-Time Operating System) und endet mit dem Löschen aus dieser Liste.

Zustände des Rechenprozesses von REGEL_3:



Zustandsdiagramm für Rechenprozesse:



ruhend: (dormant)	Der Rechenprozess ist nicht ablaufbereit, weil die Zeitbedingungen nicht erfüllt sind.
bereit: das(runnable)	Die Zeitbedingung für die Abarbeitung des Programms ist erfüllt. Es fehlt der Start durch RTOS.
laufend: (running)	Das Teilprogramm ist in Bearbeitung.
blockiert: (suspended)	Der Rechenprozess muss auf ein Ereignis warten.

Echtzeitprogrammiersprache:	PEARL
Echtzeitbetriebssystem:	OS9

5.2.3 Synchronisierung von Rechenprozessen

Forderung: Rechenprozesse sollen mit den Vorgängen in dem technischen Prozess synchron ablaufen.

Möglichkeiten zur Synchronisierung von Rechenprozessen

- Semaphor-Variablen
- kritische Regionen
- Rendez-vous-Konzept

Synchronisierung mit Hilfe von Semaphor-Variablen:

Definition einer Semaphor-Variable *S*

mit $S \in \mathbb{N} \cup \{0\}$

Auf *S* sind 2 Operationen definiert:

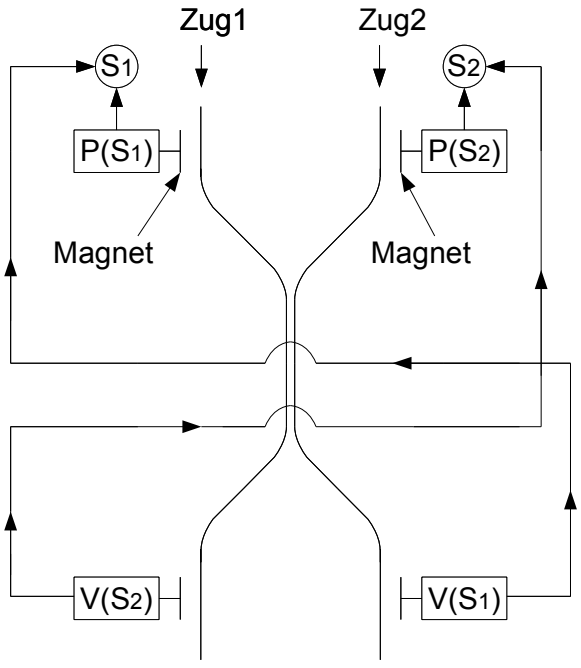
- V(*S*) (Verlassen-Operation):
  $S \rightarrow S+1$; die Task wird fortgesetzt.
- P(*S*) (Passieren-Operation):
  S wird abgefragt:
  $S>0$: $S \rightarrow S-1$; die Task wird fortgesetzt.
  $S=0$: *S* bleibt unverändert, die Task geht in den Zustand "blockiert" über. Ein Verlassen dieses Zustands ist nur möglich, wenn diese Semaphor-Variable *S* in einer anderen Task inkrementiert wird ($S>0$).

Vor der Verwendung der Semaphor-Variable muss diese initialisiert werden!

Beispiel: 2 Züge mit einer gemeinsamen Gleisstrecke.

Die Züge dürfen diese Gleisstrecke abwechselnd nutzen.

Initialisierung:
$S_1 = 1$, $S_2 = 0$



Ersetzt man die beiden Züge durch die Rechenprozesse Task1 und Task2 und das Einfahren der Züge auf das gemeinsame Gleisstück durch Prozeduren, die auf ein gemeinsames Datenfeld zugreifen, dann lässt sich das Zugbeispiel auf folgende Programmstruktur übertragen:

Initialisierung:

```
INIT:   ...
        ...
        S1:=1
        S2:=0
        ...
        ...
```

Programme für

```
        Task1:                  Task2:
        Prog1:  ...             Prog2:  ...
                ...                     ...
                P(S1)                   P(S2)
                CALL PR1                CALL PR2
                V(S2)                   V(S1)
                ...                     ...
                ...                     ...
                END                     END
```

6 Speicherprogrammierbare Steuerung (SPS)

6.1 Einleitung

6.1.1 Geschichtliche Entwicklung

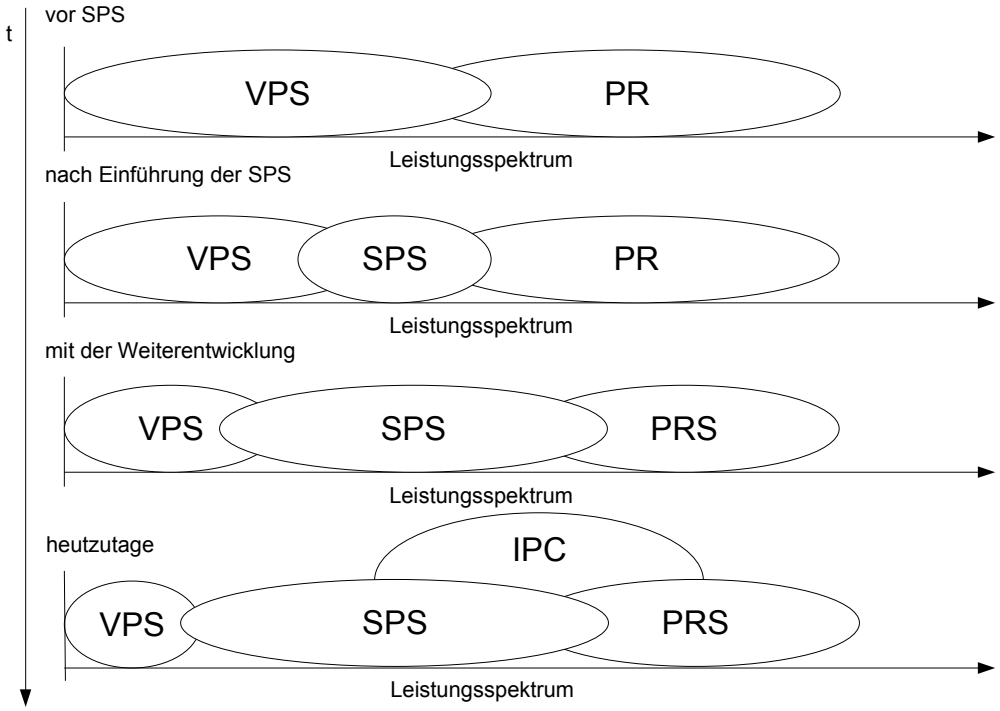
Konventionelle Steuerung mit

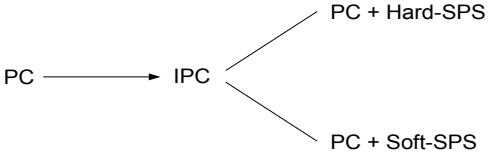
Schaltelementen (mechanisch, elektrisch, elektronisch, pneumatisch, hydraulisch)

Verbindung der Schaltelemente

Die Verbindung (z.B. Verdrahtung) der Schaltelemente bestimmt das Verhalten der Steuerung (≡ Programm)
⇒ verbindungsprogrammierte Steuerung (VPS)

1970 Werkzeugmaschinenmesse in Chicago: erste speicherprogrammierbare Steuerung



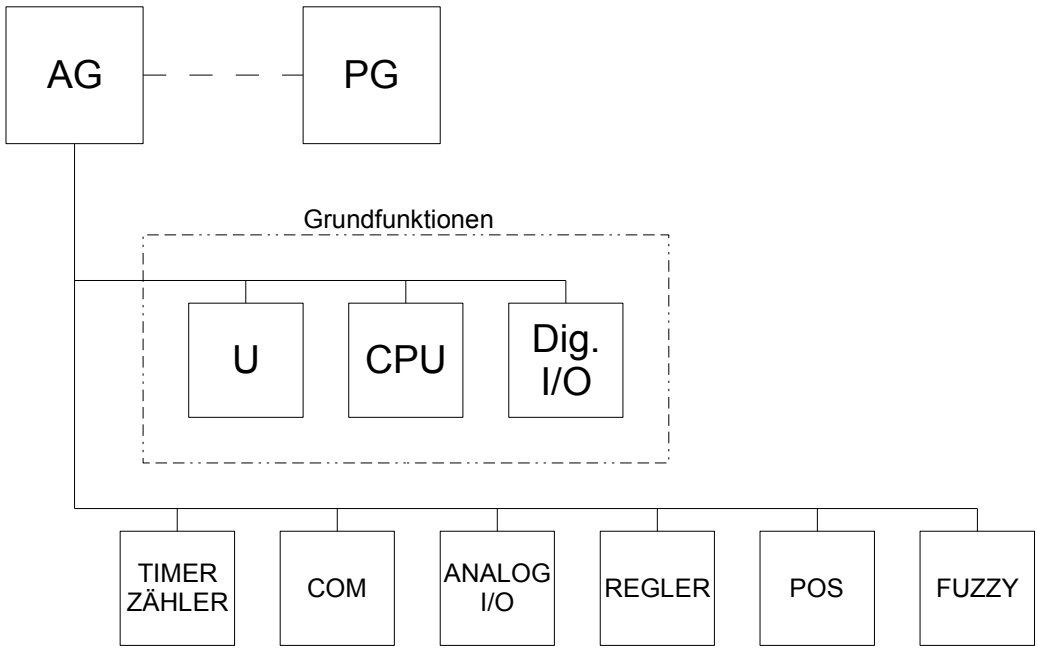


PR: Prozessrechner
PRS: Prozessrechnersystem
IPC: Industrie-PC
SPS: Speicherprogrammierbare Steuerung

Bezeichnungen:

PIC	Programmable Interface Controller
SPC	Stored-Program Controller
PC	Programmable Controller
PLC	Programmable Logic Controller (heute übliche Bezeichnung im angelsächsischen Sprachraum)
PAC	Programmable Automation Controller (neue Bezeichnung für PLC mit erweiterter Funktionalität)
FPS	Freiprogrammierbare Steuerung
SPS	Speicherprogrammierbare Steuerung (seit 1981 Norm im deutschen Sprachbereich)

6.2 Komponenten einer SPS



AG: Automatisierungsgerät

PG: Programmiergerät

Handgeräte (online)
- Funktionstastatur

tragbare PC (offline, online)
- Standardtastatur (Funktionstastatur, EPROM-Programmierung)

PC (offline)
- Tischgerät
- größerer Komfort

Aufgaben:
- Programme entwickeln
- Simulieren (offline)
- Programmtest in der SPS (online)
- Inbetriebnahme
- Wartung

U: Stromversorgungsbaugruppe

Anschlussspannungen: AC 230 V, 115 V
DC 24 V
Ausgangsspannungen: DC 5 V, 15 V, 24 V

CPU: Zentralbaugruppe

Steuerwerk mit Bearbeitungszeiten:
z.B. für die 300er Familie von Siemens
Binäroperationen: 0.05-0.20 µs
Wortoperation: 0.10-2.00 µs
Festpunktarithmetik: 0.10-5.00 µs
Gleitpunktarithmetik: 1.00-6.00 µs

Speicher für:
Betriebssystem (deterministisch, unveränderlich)
System-SW (unveränderlich)
Anwenderprogramme
Daten

Speichertypen:
EPROM, EEPROM, Flash-EPROM
RAM (teilweise Batterie-gepuffert)

Dig. I/O: Digitale Ein-Ausgabebaugruppe

Ein-/Ausgänge sind potenzialgetrennt.

Ausgänge in Halbleiter- oder Relaisausführung

Anzahl der Ein-/Ausgänge je Baugruppe: 4, 8, 2x8, 4x8

Eingangsspannungen bzw. Ausgangsspannungen: DC 24 V
AC 24 V, ..., 115 V, 230 V

Ausgangsbelastung: 0.5 A, 2 A, (5A)

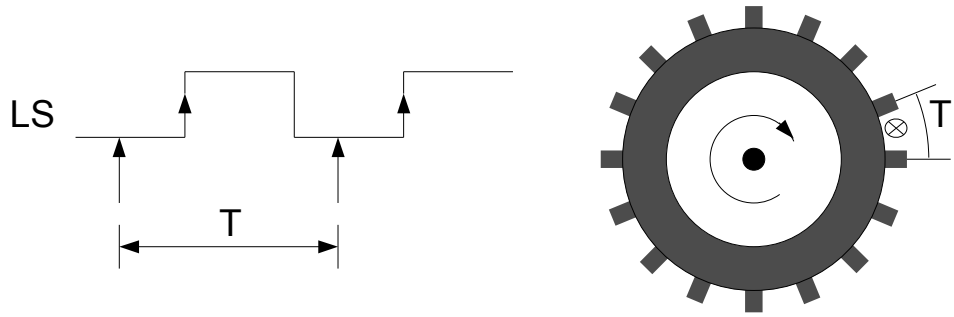
TIMER, ZÄHLER: Baugruppe für Zeit- und Zählfunktionen

z.B. Zeitfunktionen: Impuls, Ein-/Ausschaltverzögerung
Zählfunktionen: Inkrementieren bzw. Dekrementieren eines Zählwerts in Abhängigkeit von Ereignissen (steigende Flanke eines Eingangssignal)

Die Bedienung erfolgte früher über
Potentiometer (RC-Glied)
dualcodierte DIP-Schalter

In der Mitte der 90er Jahren wurden diese Funktionen in die Systemsoftware integriert (Firmware).
Die Funktionen konnten im Programm verwendet werden.
Vorteil: Komfort
Nachteil: begrenzte Zählfrequenz

Beispiel: Überwachung der Drehzahl eines Zahnrads mit Hilfe einer Lichtschranke LS



Die Abtastung des LS-Signals erfolgt zu Beginn des Programmzyklus. Ist die Impulsdauer von LS kürzer als die Programmzykluszeit T, dann wird der Impuls nicht erkannt.

Für hohe Zählfrequenzen werden daher weiterhin externe Baugruppen verwendet. Diese Baugruppen besitzen einen eigenen µP und können programmiert werden. Zählfrequenz bis 500 kHz.

COM: Kommunikationsbaugruppe

Punkt-zu-Punkt-Kopplung:	V.24 / V.28 20 mA bis 19.2 kBit/s
Feldbus-Kopplung:	9.6 kBit/s bis 12 Mbit/s
LAN (Ethernet):	100 Mbit/s

ANALOG I/O: analoge Ein-/Ausgabebaugruppe

Anzahl der Ein-/Ausgänge je Baugruppe: 4, 8, 2x8

Eingangsbereiche:	± 50 mV... bis ± 10 V 0 V ... 1V/10 V ± 20 mA 0/4 mA ... 20 mA Pt 100
Ausgangsbereiche:	± 10 V 0 V ... 5/10 V ± 20 mA 0/4 mA ... 20 mA

REGLER: Reglerbaugruppe

mit analogen und binären Ein-/Ausgängen

Reglerfunktionen:	Zwei-/Dreipunktregler PID–Regler Antriebsregler

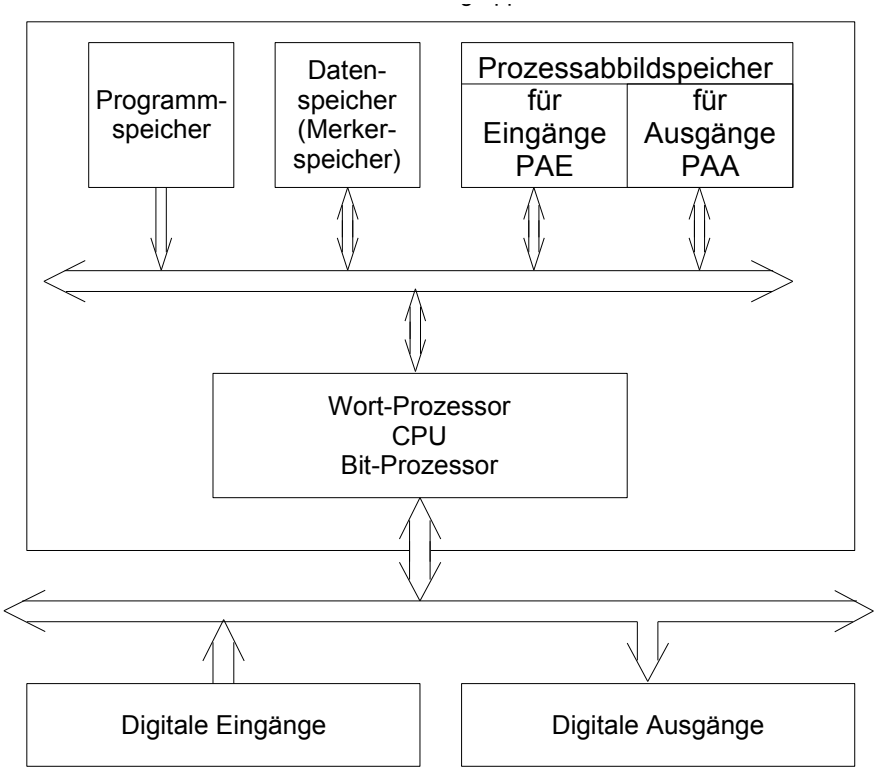
POS: Positionierbaugruppe

Dient zum geregelten Positionieren unterschiedlicher Antriebe.

FUZZY: Fuzzy-Baugruppe

Reglerbaugruppe, die nach der Methode der "unscharfen Logik" arbeitet.

6.3 Schematischer Aufbau und Wirkungsweise einer minimalen SPS



Realisierung der CPU:

Bit–Prozessor: ASIC mit entsprechender Koordinierung durch das BeSy
Wort–Prozessor: Standard-µP

oder Mikrocontroller mit boolschem Prozessor und Wortprozessor (siehe 8051)

Programmspeicher für: BeSy
die System-SW
die Anwenderprogramme

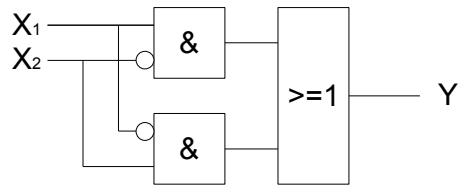
Datenspeicher für: binäre Operanden (Merker), bitadressierbar
Wortoperanden

Prozessabbildspeicher für:

Eingänge (PAE)

Ausgänge (PAA)

Dieser Speicher dient zur Synchronisierung der internen Datenverarbeitungmit den zeitlichen Abläufen des technischen Prozesses. Das folgende Beispiel einer EXOR-Verknüpfung soll die Notwendigkeit dieses Speichers verdeutlichen:



Bei der sequentiellen Arbeitsweise einer CPU kann es bei der Abfrage von Signalen an verschiedenen Zeitpunkten zu fehlerhaften Ergebnissen führen:

$$t_1 \ : \ x_1 \rightarrow Akku$$

$$t_2 \ : \ \overline{x_2} \wedge Akku \rightarrow Akku$$

$$t_3 \ : \ Akku \rightarrow Zwischenspeicher$$

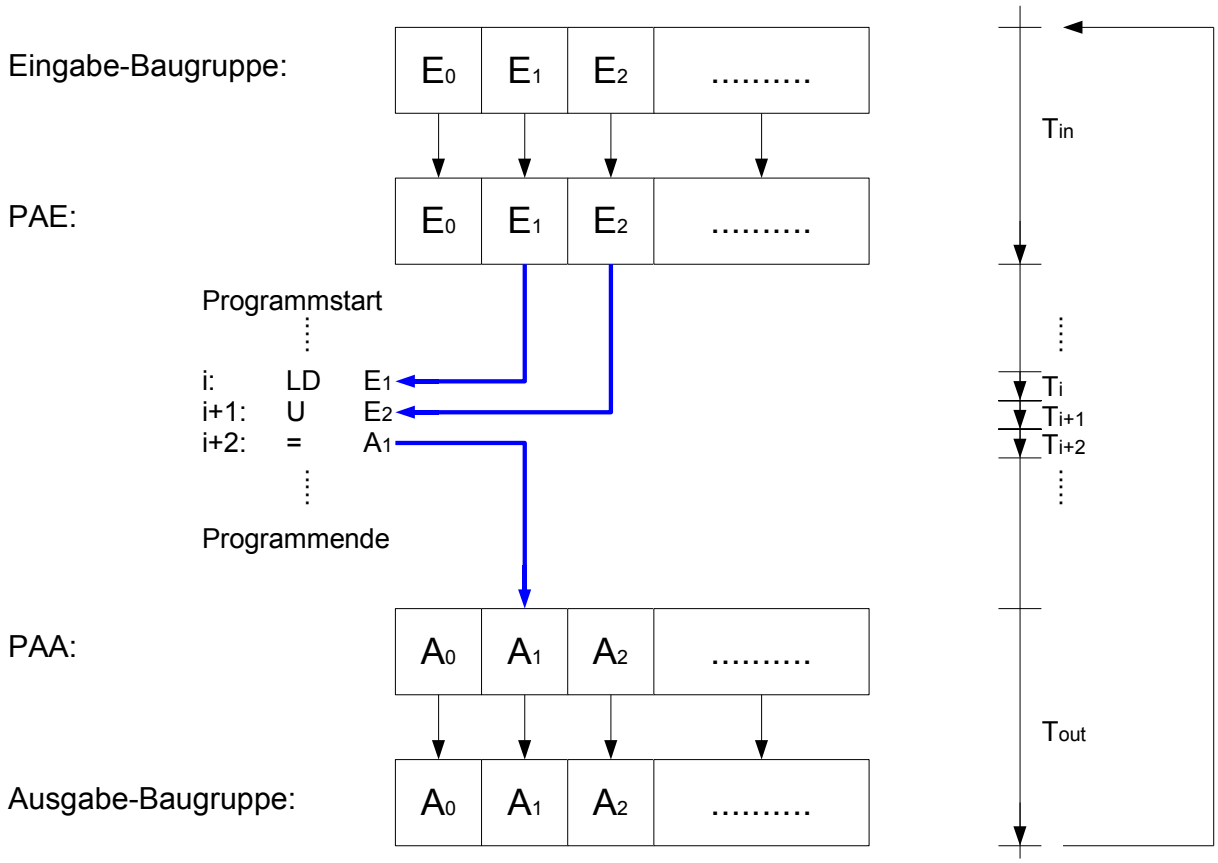
$$t_4 \ : \ \overline{x_1} \rightarrow Akku$$

$$t_5 \ : \ x_2 \wedge Akku \rightarrow Akku$$

$$t_6 \ : \ Zwischenspeicher \vee Akku \rightarrow y = (x_1(t_1) \wedge \overline{x_2}(t_2)) \vee (\overline{x_1}(t_4) \wedge x_2(t_5)) \neq x_1(t_i) \ll x_2(t_i)$$

Zur Vermeidung dieses Fehlers werden alle Eingangssignale zu Beginn eines Programmzyklus zeitgleich erfasst und im PAE abgespeichert. Eine nachfolgende EXOR-Verknüpfung ergibt dann ein richtiges Ergebnis bezogen auf den Zeitpunkt der Signalerfassung. Entsprechendes gilt auch für die Ausgangssignale.

Beispiel für einen Programmzyklus:



$$T_{Prog} = T_{in} + \sum_{i=0}^{N} T_i + T_{out}$$

Die Programmzykluszeit ist abhängig von:

- Verarbeitungsgeschwindigkeit der SPS (maschinenbedingt)
- Anzahl der Anweisungen $(N+1)$ (problembedingt)

Bei der Projektierung muss die Reaktionszeit einer SPS ermittelt werden.

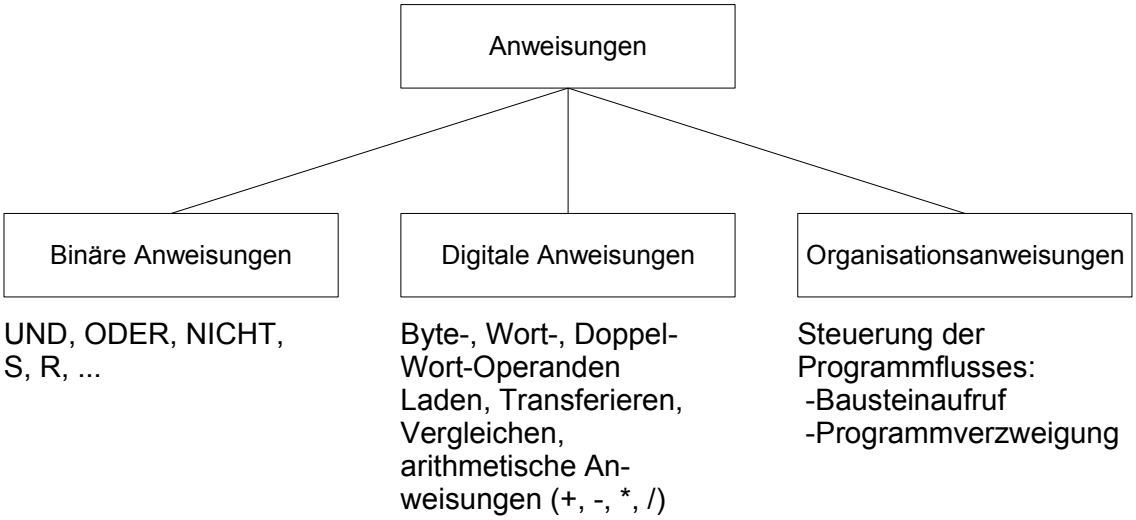
Lösung, wenn Reaktionszeit zu groß ist:

- Umgehung des Prozessabbildspeichers: direkt gesteuerte Eingabe/Ausgabe
- Auslagerung von Teilaufgaben an spezielle Baugruppen, z.B. Regler-Baugruppe, Positionier-Baugruppe, ...
- alarmgesteuerte Betriebsweise (Interrupt)
- Dezentralisierung (mehrere SPSen, sonstige Automatisierungsbausteine, bis hin zu intelligenten Sensoren und Aktoren)

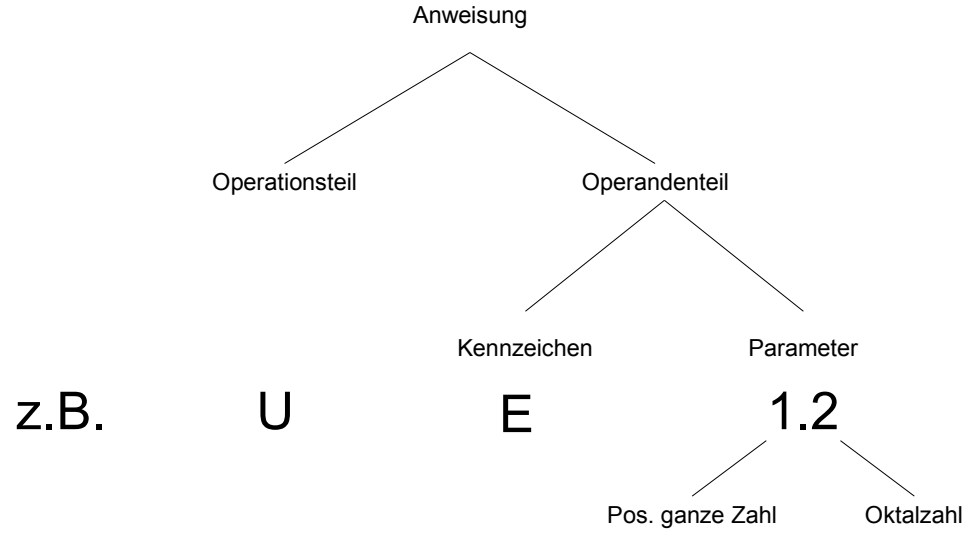
7 Programmentwicklung nach DIN 19239 (alt)

7.1 Programmstruktur

Anweisung: kleinste selbstständige Einheit eines Programms



Struktur einer Anweisung: (Akkumulator-Befehle mit 1 Operanden)

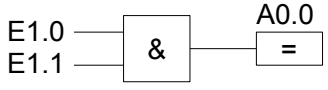


Kennzeichen:

E:	Eingang
A:	Ausgang
M:	Merker
Z:	Zähler
T:	Zeit

Netzwerk: Zusammenfassung von logisch zusammengehörigen Anweisungen zu einer Funktionseinheit, z.B.

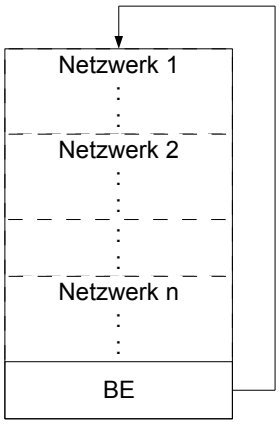
Netzwerk 1:



Kommentar

```
U     E1.0
U     E1.1
=     A0.0
```

Programm: Zusammenfassung logisch zusammengehöriger Netzwerke zu einer Funktionseinheit mit Abschlussanweisung (BE).



modulare Programmstruktur

unterschiedliche SW-Module:

OB n: Organisationsbaustein, n= 1, ...
Verwaltung der Anwender-Programme

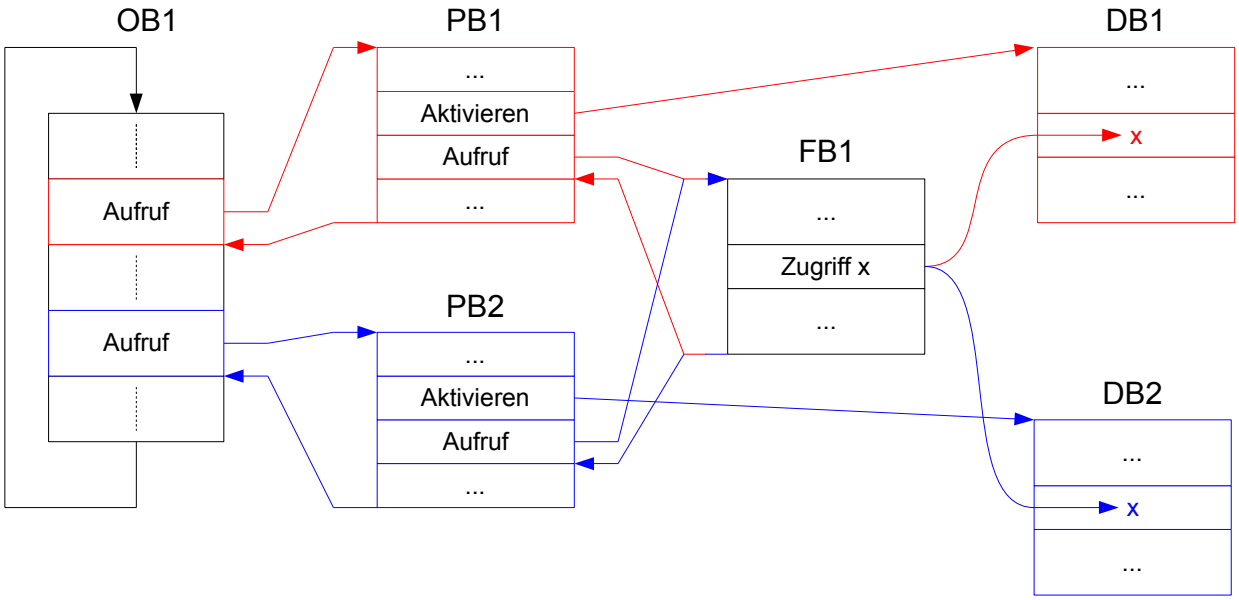
PB n: Programmbaustein
für anwenderspezifische Aufgaben

FB n: Funktionsbaustein
komplexe Aufgaben, häufig wiederkehrende Funktionen (z.B. PID-Regler)

DB n: Datenbaustein
lokaler Speicher, der nach Aktivierung verwendet werden darf

SB n: Schrittbaustein
Verwendung bei Ablaufsteuerungen

Beispiel einer modularen Programmstruktur:



7.2 Binäre Grundfunktionen

Programmiersprachen:

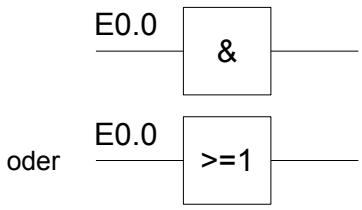
Anweisungsliste (textuell):	AWL
Funktionsplan (graphisch):	FUP
Kontaktplan (graphisch):	KOP

Laden eines Operanden:

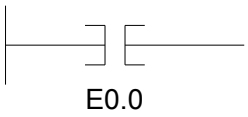
AWL:

```
      U     E0.0        A := E0.0   ; A ist der binäre Akku
oder  O     E0.0        A := E0.0
```

FUP:



KOP:

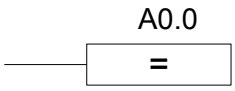


Zuweisung an einen Operanden:

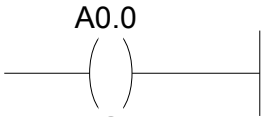
AWL:

```
      =     A0.0        A0.0 := A
```

FUP:



KOP:

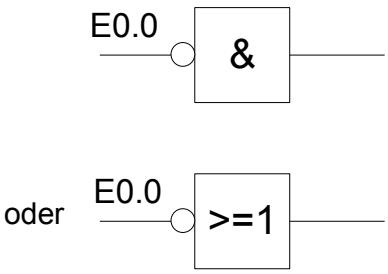


Negation:

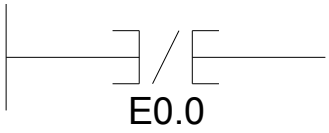
AWL:

```
      UN    E0.0
      ON    E0.0
```

FUP:



KOP:

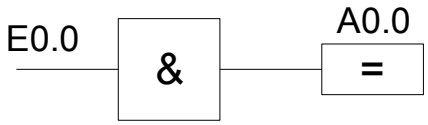


Aufruf und Zuweisung:

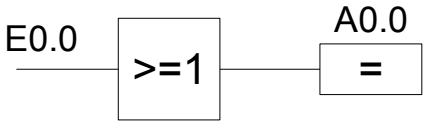
AWL:

```
U     E0.0        oder        O     E0.0
=     A0.0                    =     A0.0
```

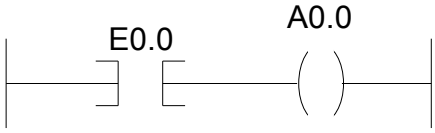
FUP:



oder



KOP:

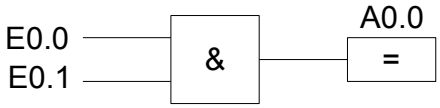


UND-Verknüpfung:

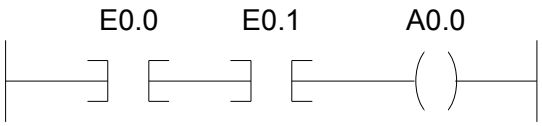
AWL:

```
U     E0.0                    O     E0.0
U     E0.1                    U     E0.1
=     A0.0                    =     A0.0
```

FUP:



KOP:

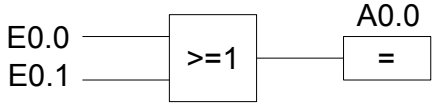


ODER-Verknüpfung:

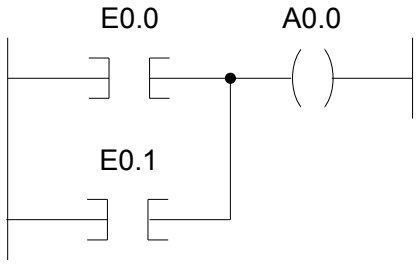
AWL:

```
O     E0.0                    U     E0.0
O     E0.1                    O     E0.1
=     A0.0                    =     A0.0
```

FUP:

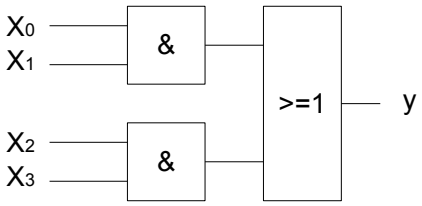


KOP:



UND-vor-ODER-Verknüpfung

$$y=(x_0 \wedge x_1) \vee (x_2 \wedge x_3)=(x_0 \cdot x_1)+(x_2 \cdot x_3)=x_0 \cdot x_1 + x_2 \cdot x_3$$



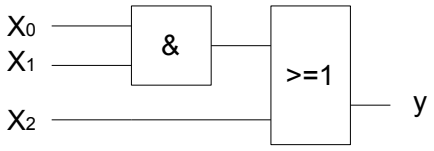
ZOL (Zuordnungsliste):

E0.0	x_0
E0.1	x_1
E0.2	x_2
E0.3	x_3
A0.0	y

AWL:

```
U     E0.0
U     E0.1
O
U     E0.2
U     E0.3
=     A0.0
```

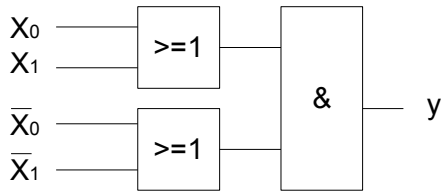
Beispiel:




```
U     E0.0        oder            U     E0.0
U     E0.1                        U     E0.1
O                                 O     E0.2
U     E0.2                        =     A0.0
=     A0.0
```

EXOR-Verknüpfung

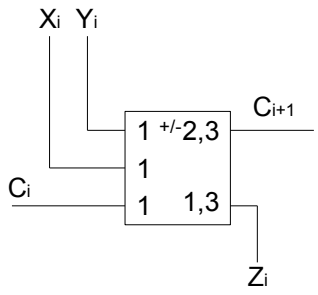
$$y=(x_0\wedge\overline{x_1})\vee(\overline{x_0}\wedge x_1)=(x_0\vee x_1)\wedge(\overline{x_0}\vee\overline{x_1})$$



AWL:

```
U     E0.0        oder        U(
UN    E0.1                    O     E0.0
O                             O     E0.1
UN    E0.0                    )
U     E0.1                    U(
=     A0.0                    ON    E0.0
                              ON    E0.1
                              )
                              =     A0.0
```

Volladdierer

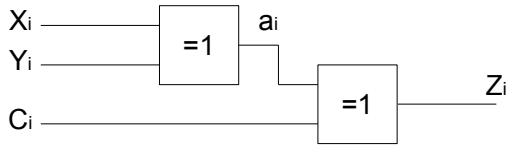


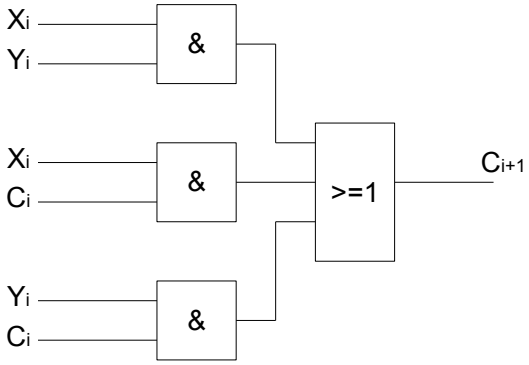
ZOL:

E0.0	x_i
E0.1	y_i
E0.2	c_i
A0.0	z_i
A0.1	c_{i+1}
M0.0	a_i

$$z_i = (x_i \; exor \; y_i) \; exor \; c_i$$

$$c_{i+1} = (x_i \wedge y_i) \vee (x_i \wedge c_i) \vee (y_i \wedge c_i)$$





AWL: Netzwerk 1: Zwischenergebnis a_i :

```
U     E0.0
UN    E0.1
O
UN    E0.0
U     E0.1
=     M0.0   ; ai
```

Netzwerk 2: Summenbit z_i :

```
U     M0.0
UN    E0.2
O
UN    M0.0
U     E0.2
=     A0.0   ; zi
```

Netzwerk 3: Übertrag c_{i+1} :

```
U     E0.0
U     E0.1
O
U     E0.0
U     E0.2
O
U     E0.1
U     E0.2
=     A0.1   ; ci+1
```

7.3 Speicherfunktion

Zwischenmerker

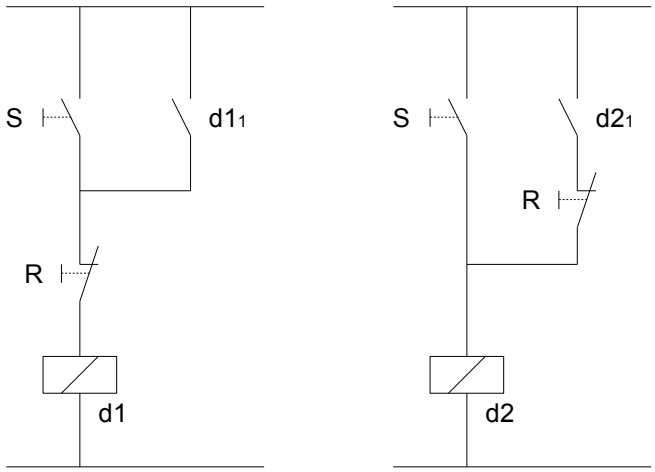
Bildung eines Verknüpfungsergebnisses (VKE) und Zuweisung an den Zwischenmerker

AWL:

```
...
...                    ; VKE
=       M0.0           ; Zwischenergebnis
```

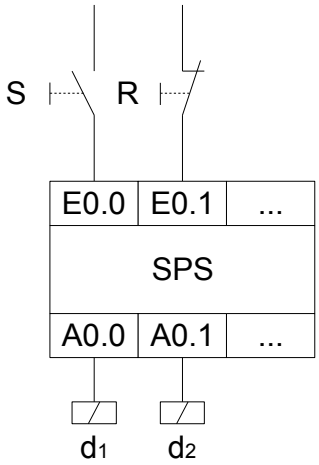
Speichererhalt ist nur für die Dauer eines Programmzyklus gewährleistet.

Selbsthalteschaltung



ZOL:

E0.0	S
E0.1	R (Öffner)
A0.0	d1
A0.2	d2



Klemmenanschlußplan

AWL: Netzwerk 1: Rücksetzdominanz

```
O     E0.0
O     A0.0
U     E0.1
=     A0.0
```

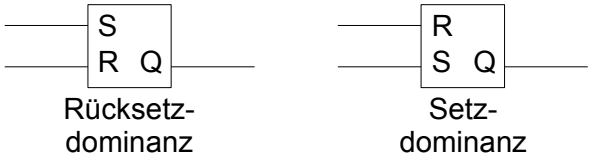
Netzwerk 2: Setzdominanz

```
U     A0.2
U     E0.1
O     E0.0
=     A0.2
```

RS-Speicher

(direkt gesteuert)

FUP:

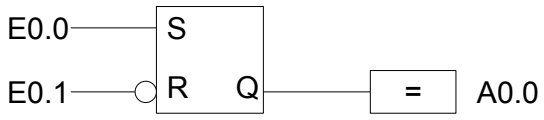


AWL:

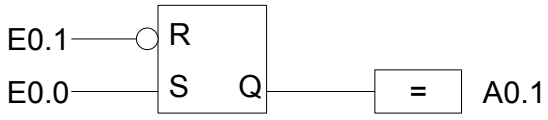
Setz Operand: S Operand
Rücksetz Operand: R Operand

Befehle werden in Abhängigkeit eines vorher gebildeten VKE ausgeführt.

FUP: Netzwerk 1: Rücksetz-Dominanz



Netzwerk 2: Setz-Dominanz



Da der Rücksetztaster ein Öffner ist, muss dieser Eingang negiert abgefragt werden.

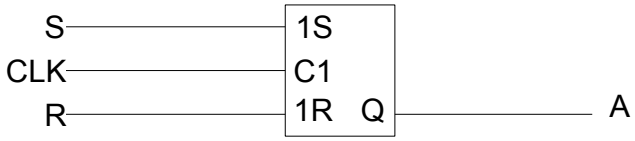
AWL: Netzwerk 1: Rücksetzdominanz

```
U           E0.0
S           A0.0
UN          E0.1
R           A0.0
```

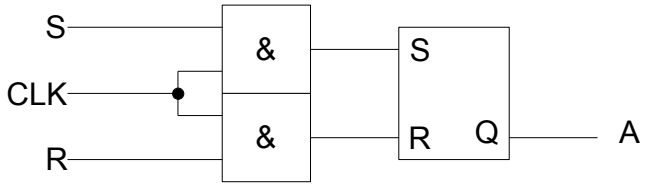
Netzwerk 2: Setzdominanz

```
UN          E0.1
R           A0.2
U           E0.0
S           A0.2
```

Zustandgesteuertes RS-FF



ZOL:



E0.0	S
E0.1	R
E1.0	CLK
A0.0	A

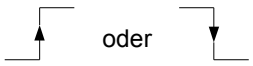
Anmerkung: Setzdominanz kann durch Tauschen der beiden Anweisungsblöcke für Setzen und Rücksetzen erreicht werden.

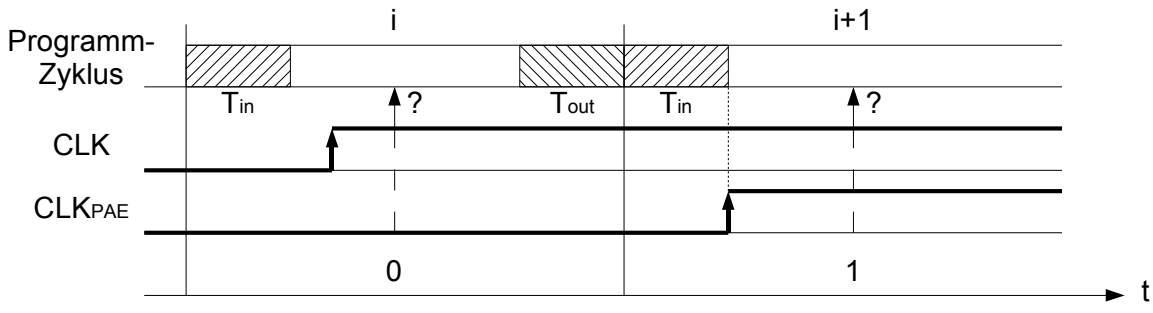
AWL:

```
U     E0.0
U     E1.0
S     A0.0
U     E0.1
U     E1.0
R     A0.0
```

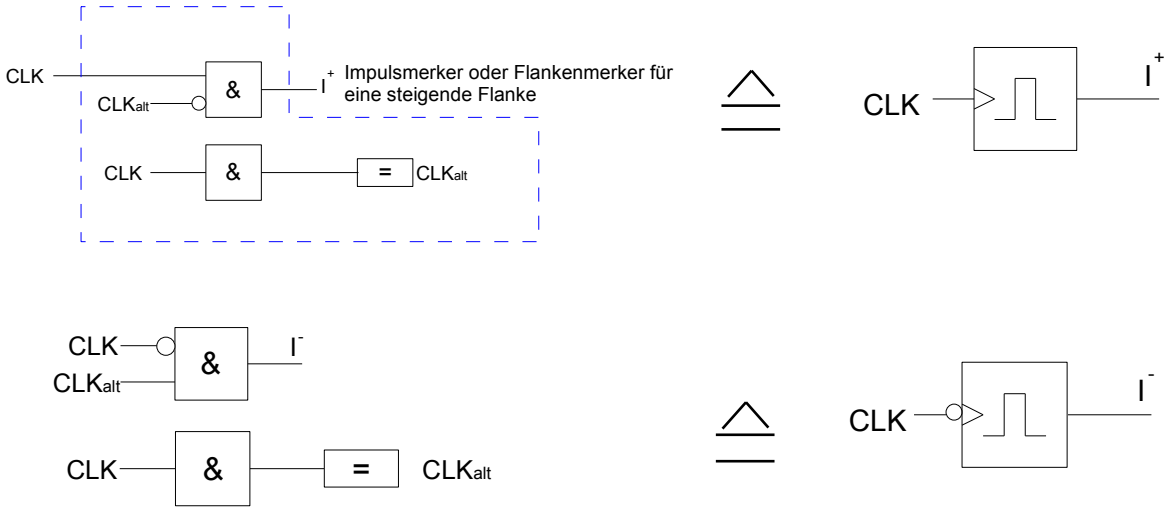
Dynamische Kippglieder

(flankengesteuert)





konventionelle Steuerung: Wischrelais (erzeugt einen Wischimpuls bei einer aktiven Flanke)



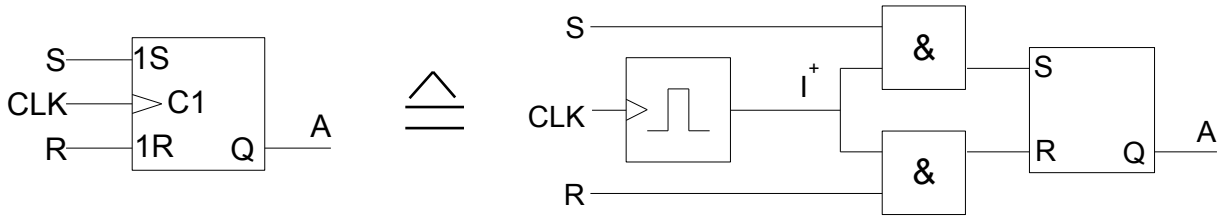
ZOL:

E1.0	CLK
M0.0	I^+/I^-
M0.1	CLK_{alt}

AWL:

```
U       E1.0
UN      M0.1
=       M0.0        ; I+

U       E1.0
=       M0.1        ;  CLKalt  aktualisieren

UN      E1.0
U       M0.1
=       M0.0        ; I-

U       E1.0
=       M0.1        ;  CLKalt  aktualisieren
```

Flankengesteuertes RS-Kippglied



ZOL:

E0.0	S
E0.1	R
E1.0	CLK
A0.0	A
M0.0	I^+
M0.1	CLK_{alt}

AWL:

```
U       E1.0
UN      M0.1
=       M0.0        ; I+

U       E1.0
=       M0.1        ; CLKalt aktualisieren

U       E0.0
U       M0.0
S       A0.0

U       E0.1
U       M0.0
R       A0.0        ; Rücksetzdominanz
```

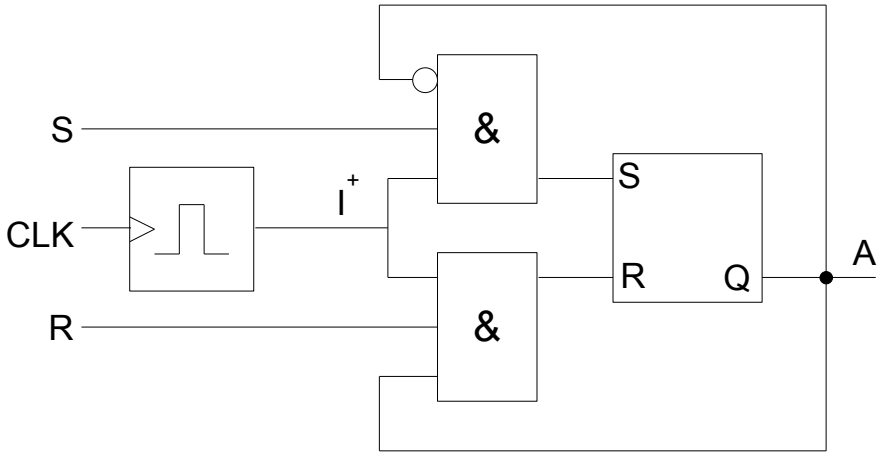
Anmerkung: Setzdominanz kann durch Vertauschen der letzten beiden Anweisungsblöcke erreicht werden!

JK-Kippglied

Wechselndes Dominanzverhalten: S=R=1 => Q:=0, wenn Q=1 und I^+=1

Q:=1, wenn Q=0 und I^+=1

naheliegendste Lösung:



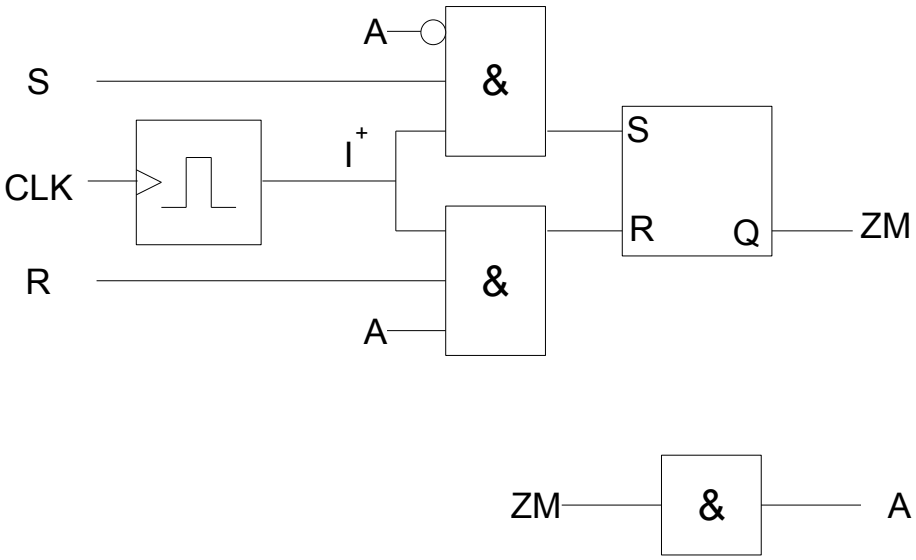

```
U           M0.0
U           E0.1
```

Überprüfung des Programms: S=R=1 ; I^+=1

Befehl	A=	
	0	1
...	...	...
S A0.0	1	1
...	...	...
R A0.0	0	0

Wie aus der Tabelle ersichtlich findet bei A=0 kein Zustandswechsel statt!

=> Einführung eines Zwischenmerkers:



ZOL:

. . .	...
M0.2	ZM

AWL:

```
U           E1.0
UN          M0.1
=           M0.0        ; I+

U           E1.0
=           M0.1

UN          A0.0
U           E0.0
U           M0.0
S           M0.2

U           M0.0
U           E0.1
U           A0.0
R           M0.2

U           M0.2
=           A0.0
```

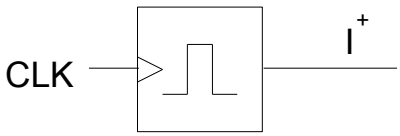
Überprüfung des Programms: S=R=1 ; I^+=1

Befehl	A=		ZM
	0	1	
. . .	...	...	...
S M0.2	...	...	1
. . .	...	...	...
R M0.2	...	...	1 bzw. 0
. . .	...	...	...
= A0.0	1	0	...

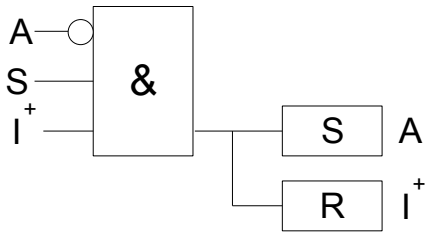
Wie aus der Tabelle ersichtlich, findet hier in beiden Fällen ein Zustandswechsel statt!

Anmerkung: Vertauschen der Rollen von A und ZM führt ebenfalls zu einem richtigen Ergebnis/Programm!

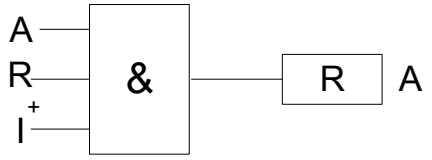
Eine vereinfachte Lösung ergibt sich, wenn nach einem Setzen des Ausgangs direkt danach der Impulsmerker zurückgesetzt wird. Das Fehlverhalten der ersten „naheliegenden Lösung“ wird dadurch verhindert:




```
U     E1.0
UN    M0.1
=     M0.0
U     E1.0
=     M0.1
```



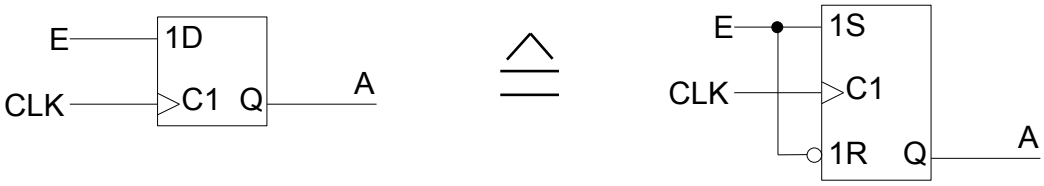

```
UN    A0.0
U     E0.0
U     M0.0
S     A0.0
R     M0.0
```




```
U     A0.0
U     E0.1
U     M0.1
R     A0.0
```

D-Kippglied

ZOL: wie bei Flankengesteuertes RS-Flip-Flop



AWL:

```
U       E1.0
UN      M0.1
=       M0.0        ; I+
U       E1.0
=       M0.1

U       E0.0
U       M0.0
S       A0.0
UN      E0.0
U       M0.0
R       A0.0
```

7.4 Digitale Anweisungen

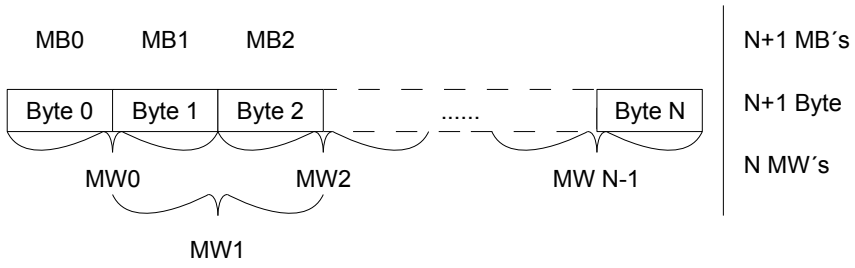
Operanden: 1 Byte
2 Byte: 1 Wort
(4 Byte: 1 Doppelwort)

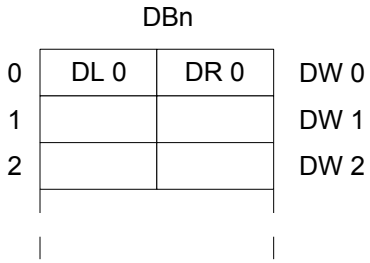
Operand besteht aus:
Kennzeichen *K* und Parameter *n* (0, ...)

Kennzeichen:

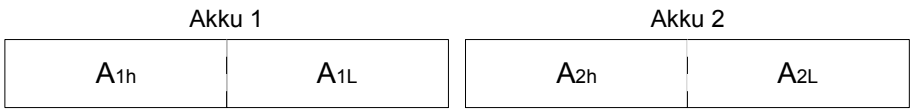
MB	Merkerbyte
MW	Merkerwert
EB	Eingangsbyte (PAE)
EW	Eingangswort (PAE)
AB	Ausgangsbyte (PAA)
AW	Ausgangswort (PAA)
PY	Peripheriebyte direktes Byte
PW	Peripheriewort direktes Wort
DW	Datenwort des aktive DB's (DB: Datenbaustein)
DR	rechtes Byte von DW
DL	linkes Byte von DW
T	Zeitwert
Z	Zählwert
BS	Systemdatenwort

Der Operandenbereich ist mit Ausnahme des DB's byteorganisiert:





Die Anweisungen benutzen 2 Akkus:



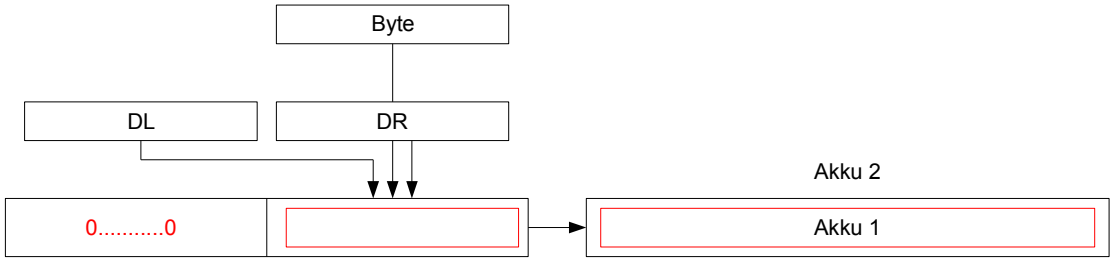
Die digitalen Anweisungen können oft nur in AWL programmiert werden!

7.4.1 Ladefunktion

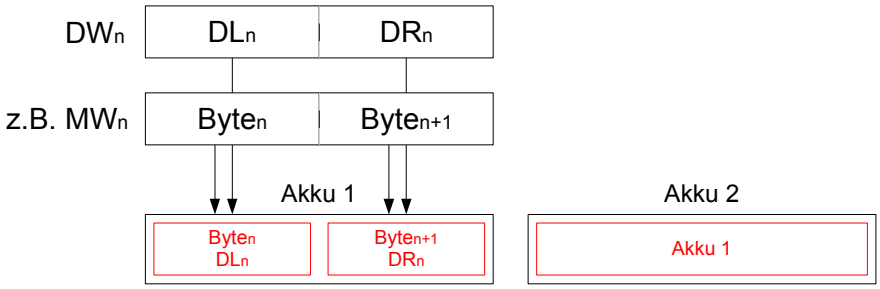
Die Ladefunktion bewirkt:

1. Akku 1 -> Akku 2
2. Operand -> Akku 1 (rechtsbündig)

Byte-Operand:



Wort-Operand:



Symbol der Ladefunktion (AWL):

L
LC (nur bei Zeit- und Zählwert)

z.B.

L EB 2 Byte 2 aus PAE -> Akku 1
L PY 2 Byte 2 der Eingabe-Baugruppe -> Akku 1
L MW 3 Byte 2 und 3 des Merkerspeichers -> Akku 1

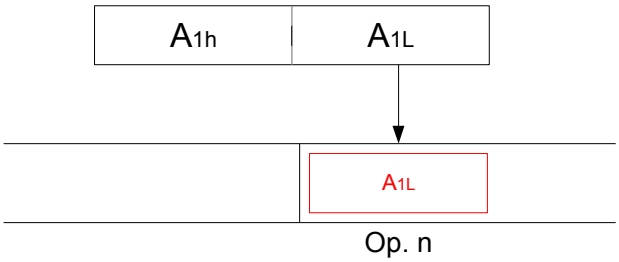
konstante Operanden:

KB 1-Byte-Dualzahl (0, ..., 255)
KF 16-Bit-Integer ($-2^{15}, \ldots, +2^{15}-1$)
KM 16-Bit-Muster
KH 4-stellige Hexzahl (0000, ..., FFFF)
KY 2 1-Byte-Dualzahlen
KC 2 ASCII-Zeichen
KT Zeitwert mit Zeitbasis
KZ Zählwert (0, ..., 999)

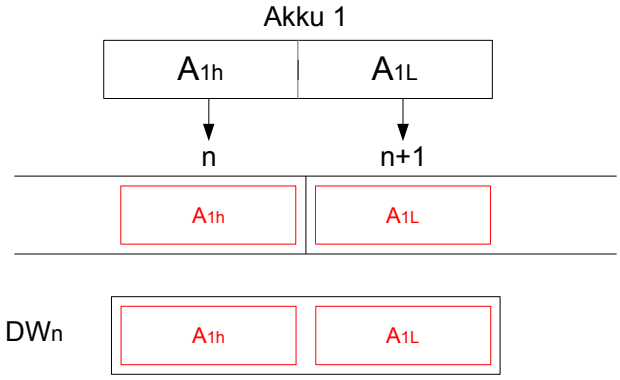
7.4.2 Transferfunktion

Übertragen (Kopieren) des Akku 1-Inhaltes zu einem Zieloperanden.

Byte-Operand:



Wort-Operand:



Symbol der Transferfunktion (AWL): T

z.B.

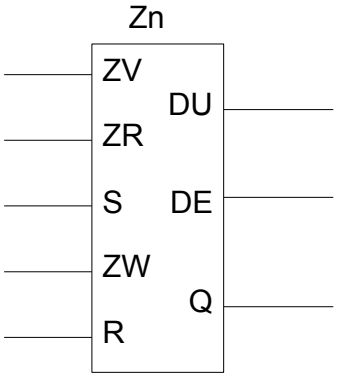
T EB 0	;	A_{1l}	-> Byte 0 von PAE
T PY 0	;	A_{1l}	-> Byte 0 der Ausgabe-Baugruppe
T MW 2	;	A_{1h}	-> Byte 2 des Merkerspeichers
		A_{1l}	-> Byte 3 des Merkerspeichers

7.4.3 Zählfunktion

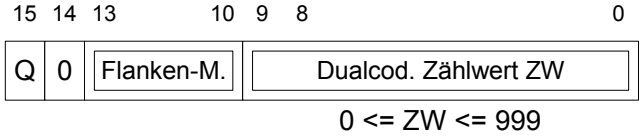
Funktionen:

- Setzen eines Zählwertes auf einen Vorgabewert
- Zählen: Inkrementieren bzw. Dekrementieren des Zählwerts
- Löschen des Zählwerts

symbolische Darstellung (FUP, KOP):



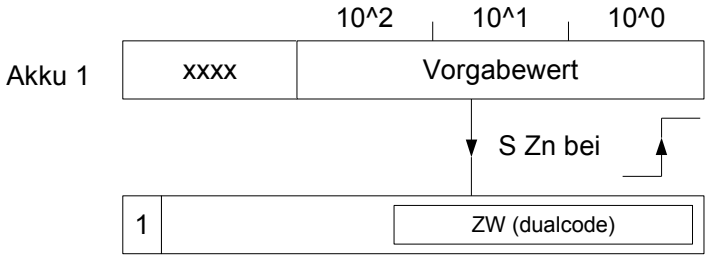
Der Operand Zn wird durch ein Speicherwort dargestellt:



Q: binärer Zustand von Zn: Zn > 0 -> Q = 1
Zn = 0 -> Q = 0

Bedeutung der Eingänge:

S: Mit steigender Flanke wird der Vorgabewert als Zählwert in den Operanden Zn übernommen.
Dieser Vorgabewert muss sich BCD-codiert in Akku 1 befinden:



R: Eine "1" bewirkt ein Löschen des ZW's und des binären Zustandes Q.

ZW: Zählwert, der als 3-stelliger BCD-codierter Vorgabewert aus Akku 1 übernommen werden kann.

ZV: Zähle vorwärts: mit steigender Flanke: ZW++ ZW <= 999

ZR: Zähle rückwärts: mit steigender Flanke: ZW-- 0 <= ZW

Bedeutung der Ausgänge:

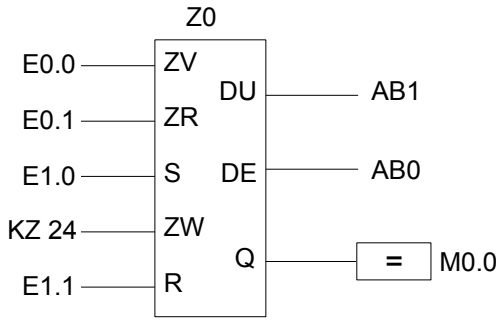
Q: binärer Zustand von Zn
Kann mit U bzw. O geladen werden (z.B. U Zn) und mit weiteren binären Operationen verarbeitet werden, z.B.

```
U     Zn
=     A0.0
```

DU: Zählwert, der dualcodiert mit der Operation L in den Akku 1 rechtsbündig übernommen werden kann.

DE: Zählwert, der BCD-codiert mit der Operation LC in den Akku 1 rechtsbündig übernommen werden kann.

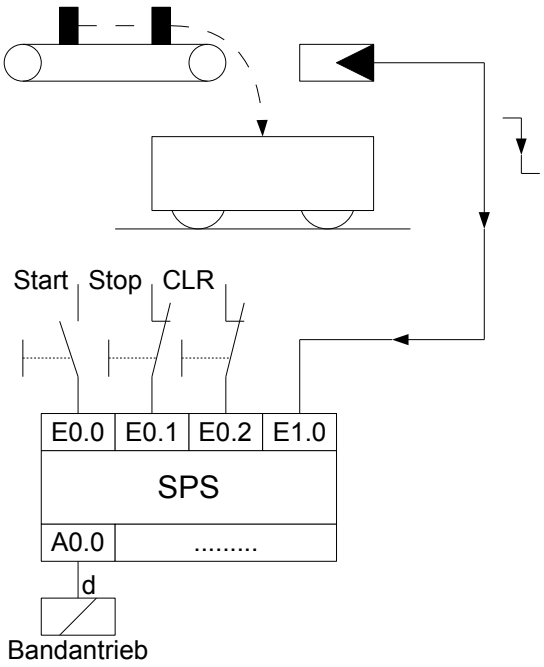
Beispiel in FUP-Darstellung:



AWL:

```
U       E0.0
ZV      Z0          ; ZW++, wenn steigende Flanke an E0.0
U       E0.1
ZR      Z0          ; ZW--, wenn steigende Flanke an E0.1

U       E1.0
L       KZ 24       ; 24 -> Akku 1
S       Z0          ; ZW := 24, wenn steigende Flanke an E1.0

U       E1.1
R       Z0          ; wenn E1.1 = 1 -> ZW := 0

U       Z0
=       M0.0        ; Q -> M0.0

L       Z0          ; ZW(dual) -> Akku 1
T       AB 1        ; ZW(dual) -> Byte 1 von PAA

LC      Z0          ; ZW(BCD) -> Akku1
T       AB0         ; ZW(BCD) -> Byte 0 von PAA
```

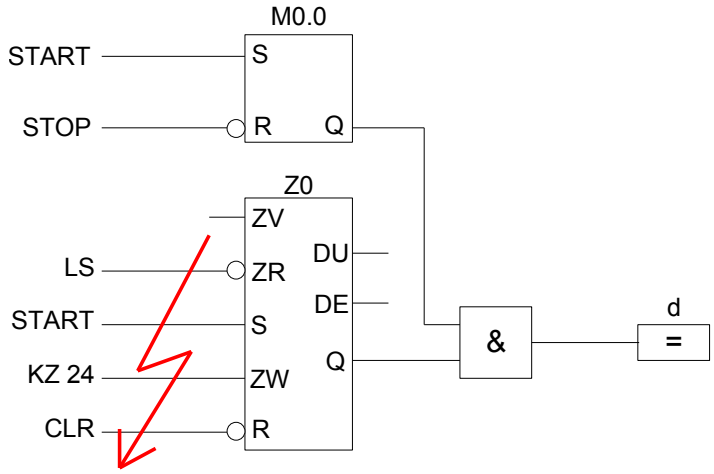
Beispiel: Förderband



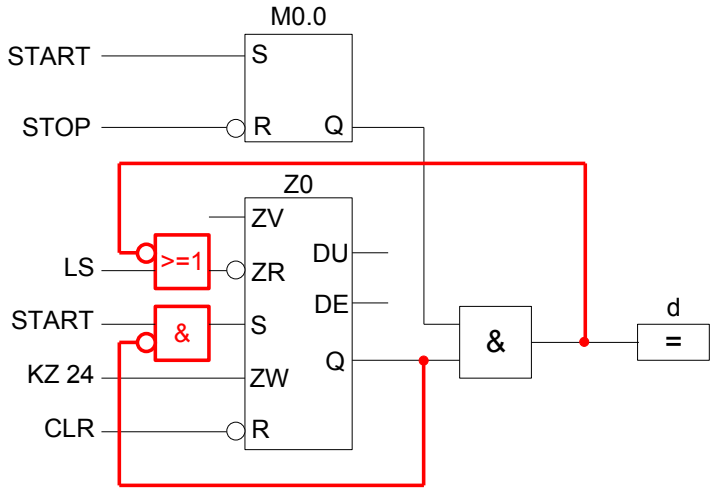
ZOL:

E0.0	START	Starttaster (S)
E0.1	STOP	Stoptaster (Ö)
E0.2	CLR	Löschtaster (Ö)
E1.0	LS	Lichtschranke (fallende Flanke)
A0.0	d	Schütz für Bandantrieb
Z0		Zähler für Anzahl
M0.0		Start-Stop-Merker

FUP:



Um ein nachträgliches Triggern der Zählereingänge ZR und S zu verhindern, werden diese mit weiteren Gattern und Rückkopplungen versehen.

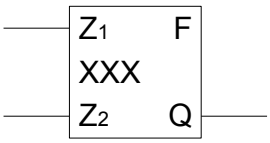


Eine weitere Verbesserung programmtechnischer Natur (in AWL) erhält man, wenn mit Hilfe der DeMorgan'schen Regel das NOR-Glied durch ein UND-Glied ersetzt wird.

AWL:

```
U           E0.0
S           M0.0
UN          E0.1
R           M0.0
U           A0.0
UN          E1.0
ZR          Z0
U           E0.0
UN          Z0
L           KZ 24 ; Konstante 24 muss in Akku1 für S Z0 stehen
S           Z0
UN          E0.2
R           Z0
U           Z0
U           M0.0
=           A0.0
```

7.4.4 Vergleichsfunktionen



Vergleichsfunktionen "xxx":

!=	$z_1 = z_2$
><	$z_1 \neq z_2$
>	$z_1 > z_2$
>=	$z_1 \geq z_2$
<	$z_1 < z_2$
<=	$z_1 \leq z_2$

wenn Ausdruck wahr, dann Q := 1

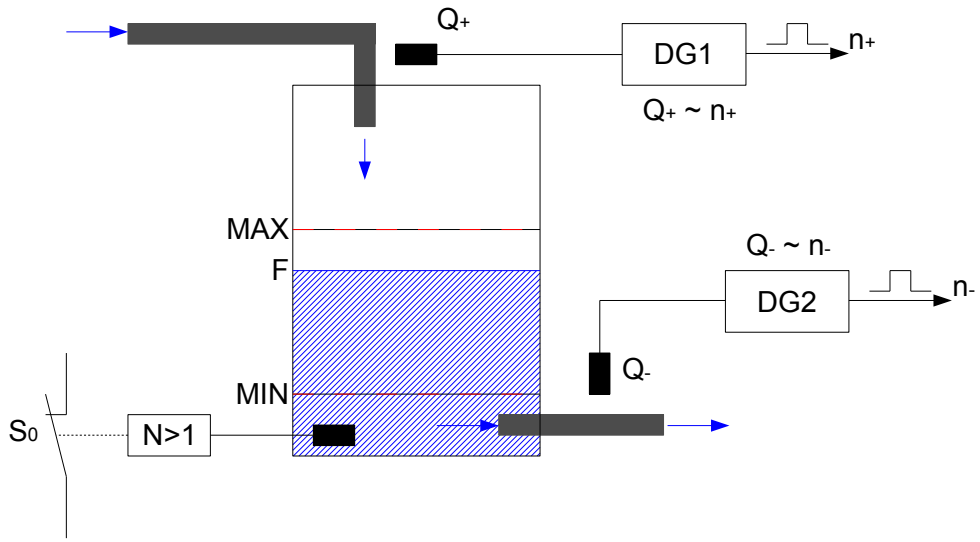
AWL-Darstellung:

1. Operand z_1 laden (z_1 -> Akku1)
2. Operand z_2 laden (z_1 -> Akku2, z_2 -> Akku1)
3. Anwenden einer Vergleichsanweisung:

```
!=F
><F
...
<=F
```

Das führt zu einem binären Verknüpfungsergebnis, das wie jeder andere binäre Operand weiter verknüpft werden kann.

Beispiel: Füllstandsanzeige



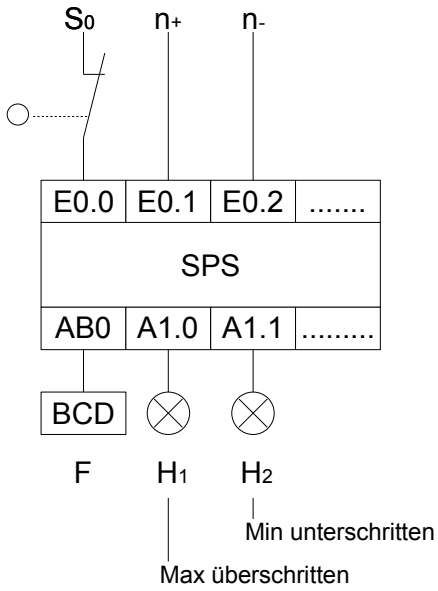
$S_0 = 1$, wenn $F = 0$

$F = Q_+ - Q_- = kn_+ - kn_- = k(n_+ - n_-)$

Zur Vereinfachung setze: $k = 1 \Rightarrow F = k_+ - k_-$

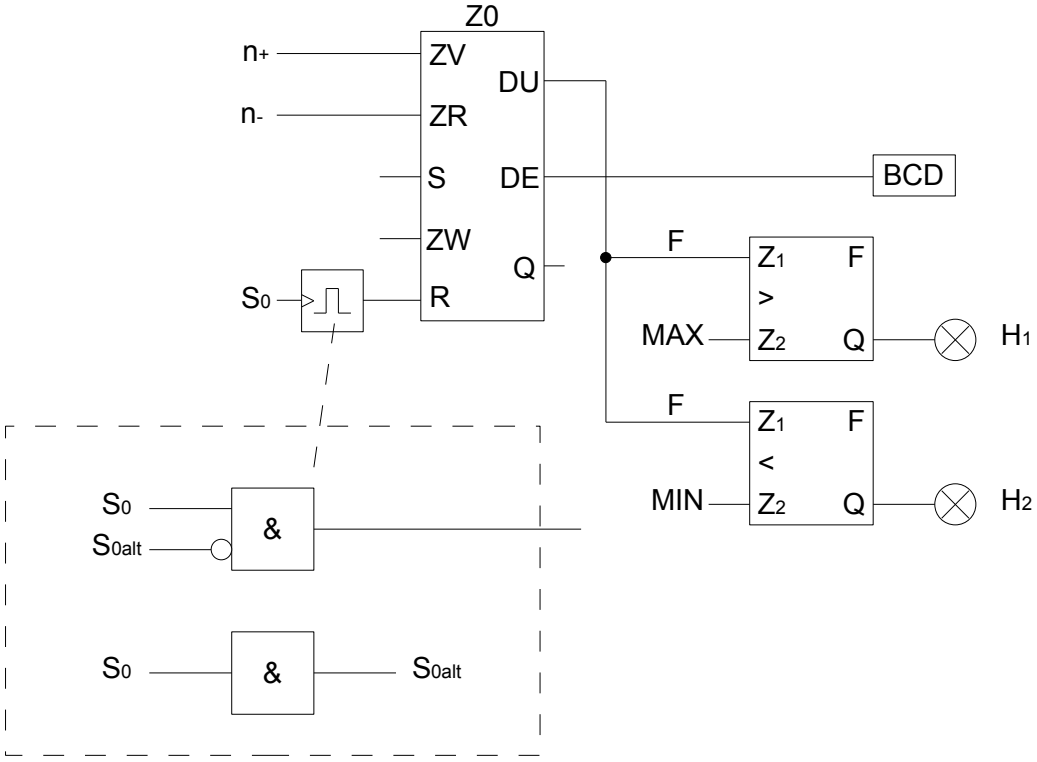
Anzeige:

$F < MIN \rightarrow H_2 : MIN\ unterschritten$
$F > MAX \rightarrow H_1 : MAX\ überschritten$
$F \rightarrow BCD\text{-}Anzeige$



ZOL:

E0.0	S_0	Endschalter "Behälter leer"
E0.1	n_+	Impulsrate Zulauf
E0.2	n_-	Impulsrate Ablauf
AB0	*F*	BCD-codierter Füllstand
A1.0	H_1	Anzeige "MAX überschritten"
A1.1	H_2	Anzeige "MIN unterschritten"
Z0		Zähler für Füllstand
M0.0	S_{0alt}	Merker für altes S_0-Signal



AWL:

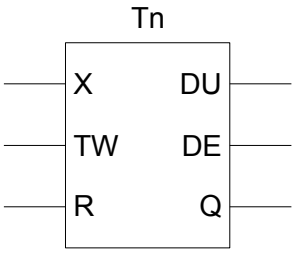
```
U       E0.1        ; n+
ZV      Z0
U       E0.2        ; n-
ZR      Z0
U       E0.0
UN      M0.0
R       Z0
U       E0.0
=       M0.0

LC      Z0          ; F_BCD -> Akku1
T       AB0         ; BCD-Ausgabe
L       Z0          ; F
L       KF 10       ; MAX
>F
=       A1.0        ; Ausgabe an H1

L       Z0          ; F
L       KF 3        ; MIN
<F
=       A1.1        ; Ausgabe an H2
```

7.4.5 Zeitfunktionen

z.B. Impulse
Einschaltverzögerung
Ausschaltverzögerung

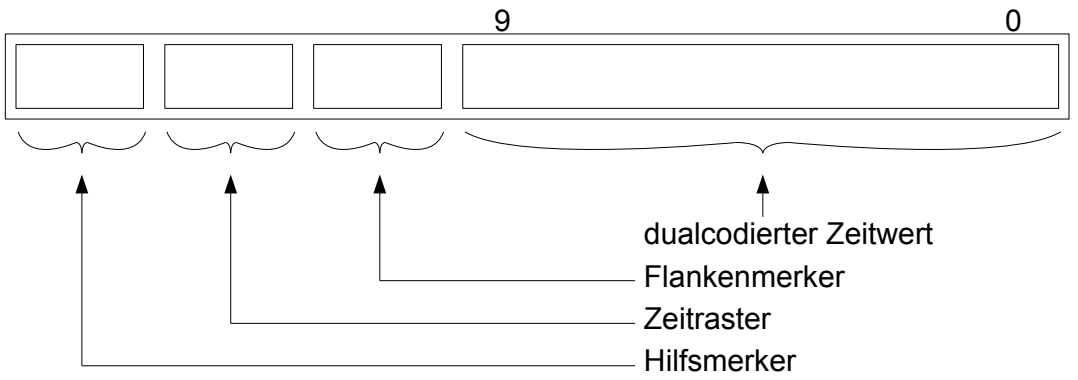
In Abhängigkeit einer aktiven Flanke eines Eingangssignals wird ein vorzugebender Zeitwert T_W geladen. Dieser Zeitwert läuft in Schritten einer vorzugebenden Zeiteinheit bis auf Null ab.

Symbolische Darstellung einer Zeitfunktion:



T_n ist der Zeitoperand, $n=0,\ 1,\ ...$

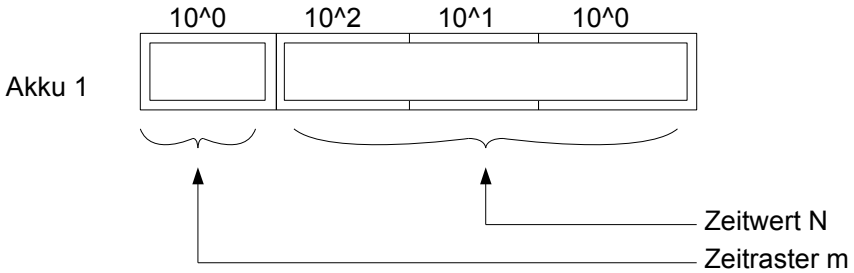
Darstellung des Zeitoperanden T_n im Speicher durch ein 16-Bit-Wort:



Bedeutung der Eingänge:

X: Starteingang der Zeitfunktion X, die noch näher beschrieben wird.
Mit einer aktiven Flanke wird ein Vorgabewert (Zeitwert T_W und Zeitraster) aus dem Akku1 in das Zeitwort des Operanden übernommen. Der übernommene Zeitwert läuft dann in Schritten der vorgegebenen Zeiteinheit bis auf Null ab.

TW: Vorgabewert aus Akku1, bestehend aus Zeitwert und Zeitraster im BCD-Code.



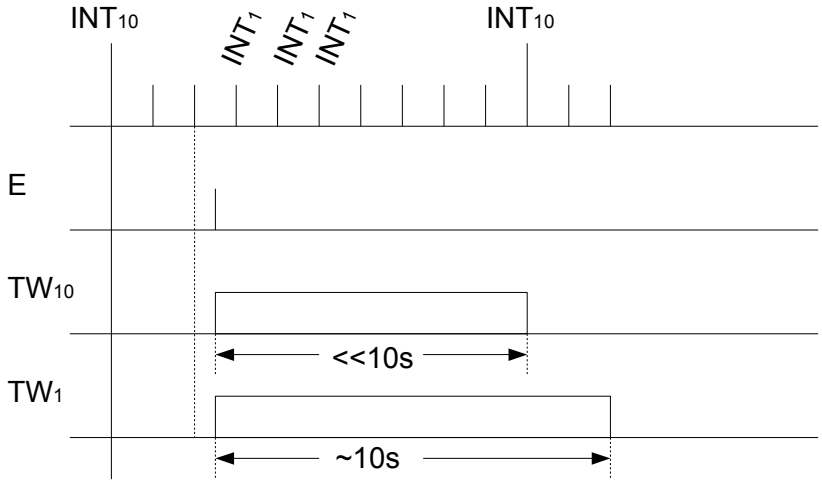
$0 \le N \le 999$
$0 \le m \le 3$

m	Zeiteinheit in s
0	0.01
1	0.1
2	1
3	10

Der Vorgabewert kann durch folgenden Befehl in den Akku1 geladen werden: `L KT N.m`

Eine RTC erzeugt Interrupts in 0.01 s, 0.1 s, 1 s und 10 s.

Annahme: Realisierung von 10 s
mögliche Vorgabewerte:
KT 1.3
KT 10.2
KT 100.1

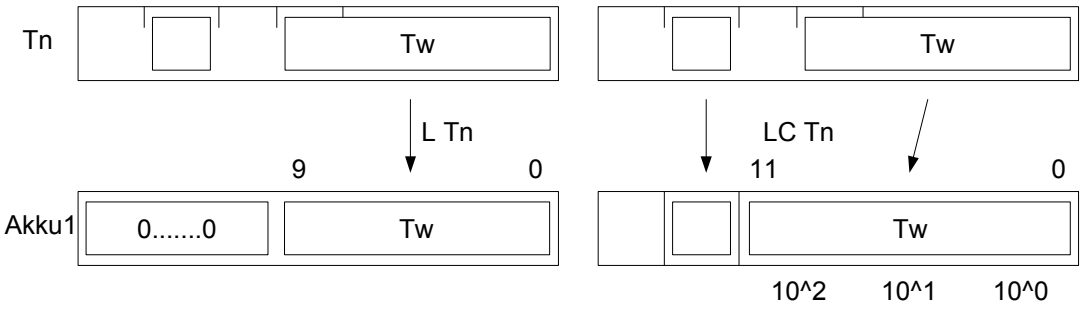


R: Rücksetzeingang, bei einer 1 an diesem Eingang wird eine laufende Zeit beendet, d.h. der Zeitwert auf Null gesetzt.

Bedeutung der Ausgänge:

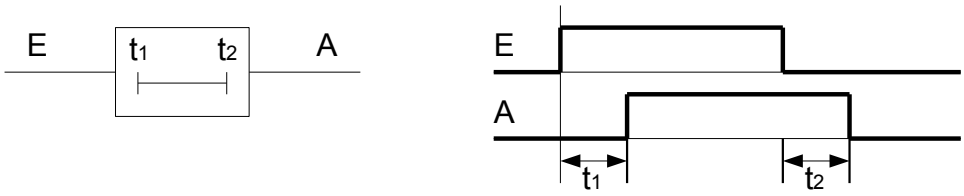
DU: digitaler Ausgang, über den der aktuelle Zeitwert dualcodiert mit dem Befehl L in den Akku1 geladen wird.

DE: digitaler Ausgang, über den der aktuelle Zeitwert BCD-codiert mit dem Befehl LC in den Akku1 geladen wird.



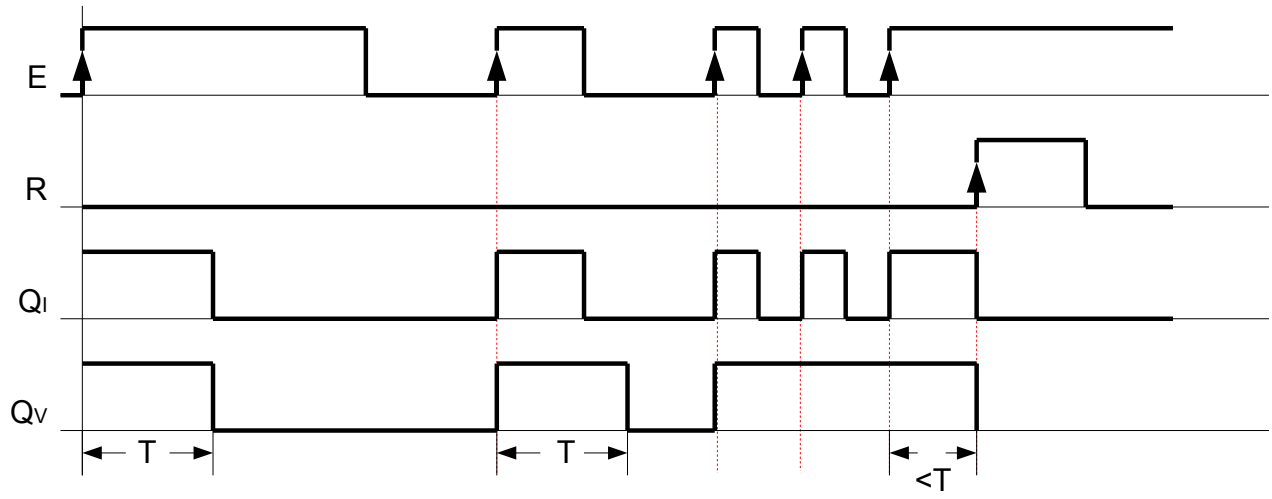
Q: binärer Ausgang, der durch die beiden Hilfsmerker bestimmt wird.
Er kann wie auch andere binäre Operanden abgefragt und logisch verknüpft werden:
Laden des Operanden: `U Tn`

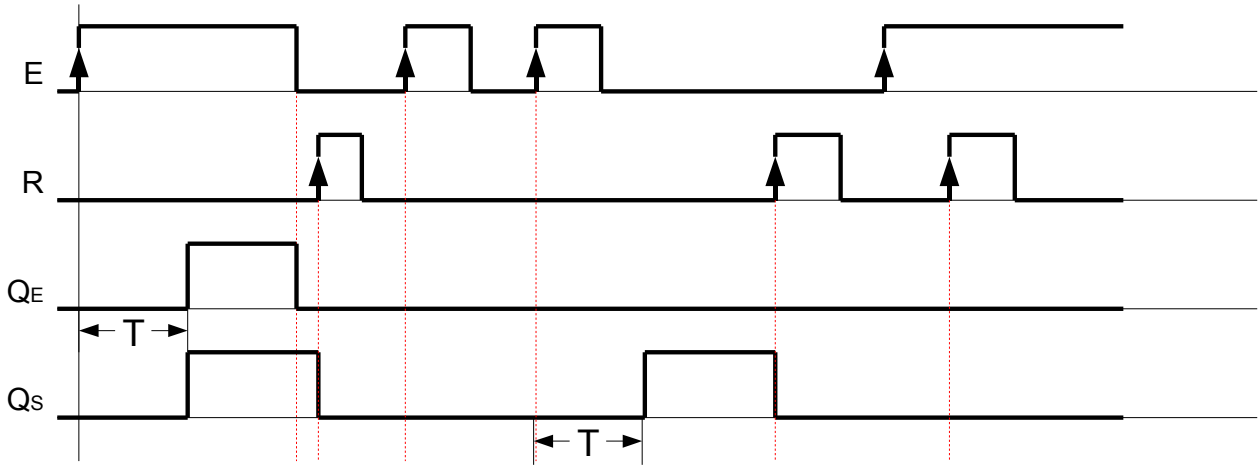
Allgemeines Symbol eines Zeitglieds mit Ein- und Ausschaltverzögerung:

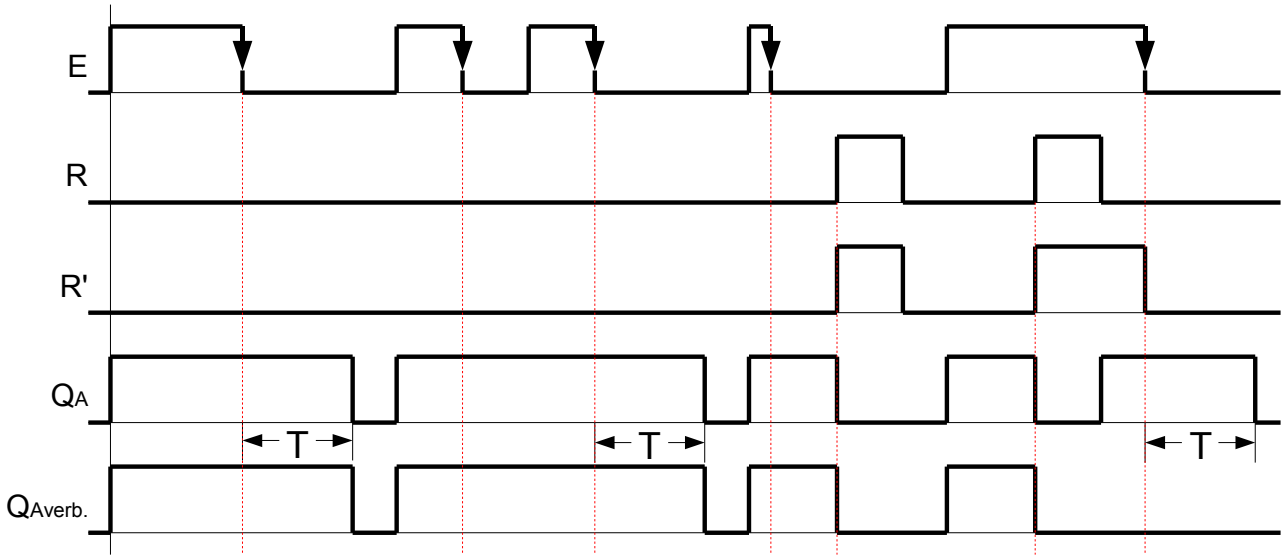


Zeitfunktion	Symbol der Zeitfunktion (an Stelle von X)	Befehl in der AWL
Impuls	1	`SI Tn`
Impuls verlängert	1 V	`SV Tn`
Einschaltverzögerung	T 0	`SE Tn`
speichernde Einschaltverzögerung	T S	`SS Tn`
Ausschaltverzögerung	0 T	`SA Tn`

Impulsdiagramme:

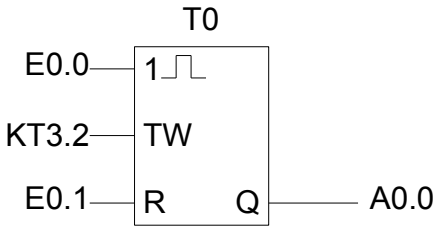




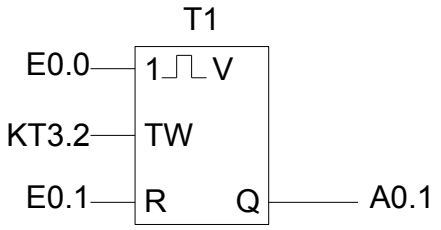


E 0.0	E	Eingangssignal
E 0.1	R	Rücksetzzignal
A0.0	Q_I	Ausgang eines Impulses
A0.1	Q_V	Ausgang eines verlängerten Impulses
A0.2	Q_E	Ausgang einer Einschaltverzögerung
A0.3	Q_S	Ausgang einer speichernden Einschaltverzögerung
A0.4	Q_A	Ausgang einer Ausschaltverzögerung
A0.5	$Q_{A\ verbessert}$	Ausgang einer verbesserten Ausschaltverzögerung
M0.0	R'	Verlängertes Rücksetzsignal für $QA_{verbessert}$

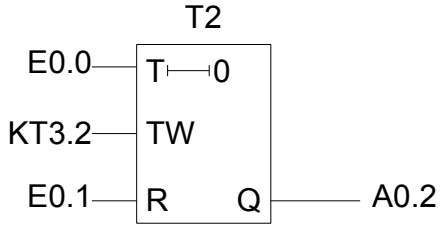
AWL der verschiedenen Zeitfunktionen:



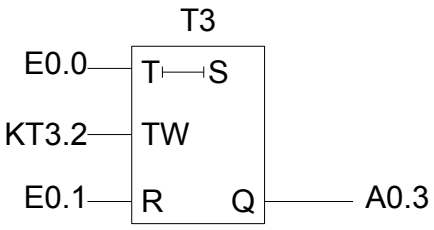

```
U       E0.0
L       KT 3.2
SI      T0
U       E0.1
R       T0
U       T0
=       A0.0
```

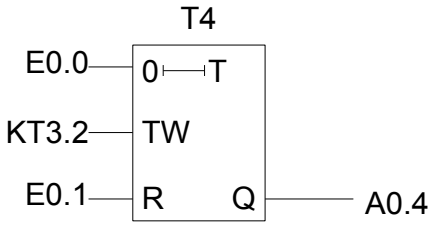



```
U       E0.0
L       KT 3.2
SV      T1
U       E0.1
R       T1
U       T1
=       A0.1
```

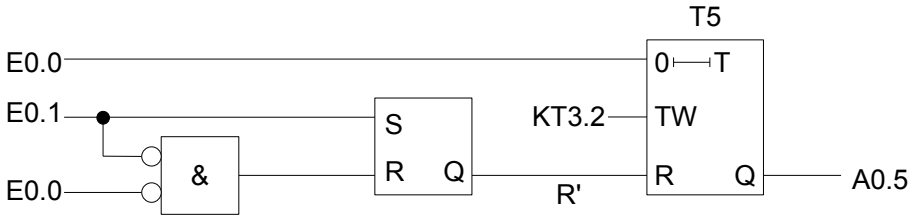



```
U       E0.0
L       KT 3.2
SE      T2
U       E0.1
R       T2
U       T2
=       A0.2
```



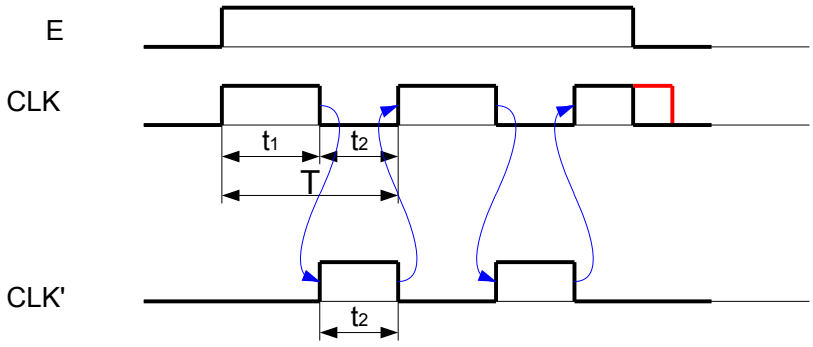


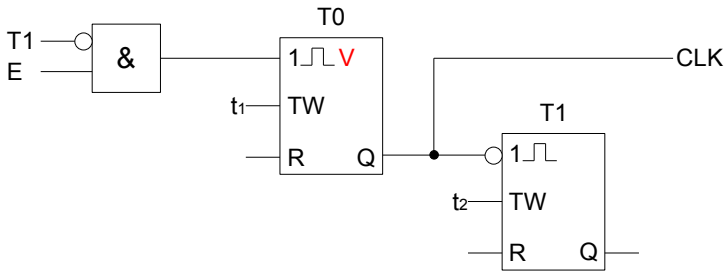

```
U           E0.0
L           KT 3.2
SS    T3
U           E0.1
R           T3
U           T3
=           A0.3

U           E0.0
L           KT 3.2
SA    T4
U           E0.1
R           T4
U           T4
=           A0.4
```




```
U           E0.0
L           KT 3.2
SA    T5
U           E0.1
S           M0.0
UN    E0.1
UN    E0.0
R           M0.0
U           M0.0
R           T5
U           T5
=           A0.5
```

Taktgeneratoren





8 Internationale SPS-Norm IEC 61 131

neue Norm IEC 1 131
inzwischen IEC 61 131

7 Teile:

Teil 1: Allgemeine Informationen
Teil 2: Betriebsmittelanforderungen und Prüfungen
Teil 3: Programmiersprachen
Teil 4: Anwenderrichtlinien
Teil 5: Kommunikation
Teil 6: Kommunikation über Feldbus (zur Zeit noch nicht veröffentlicht)
Teil 7: Fuzzy-Systeme in der SPS

IEC 61 131-3: Programmierung

Beschreibung eines umfassenden Konzepts zur Realisierung eines SPS-Projekts:
Konfiguration von HW und SW

Die Konfiguration des gesamten Projekts kann ein oder mehrere Verarbeitungseinheiten (CPU's oder Spezialprozessoren) besitzen, die man als Ressourcen bezeichnet.

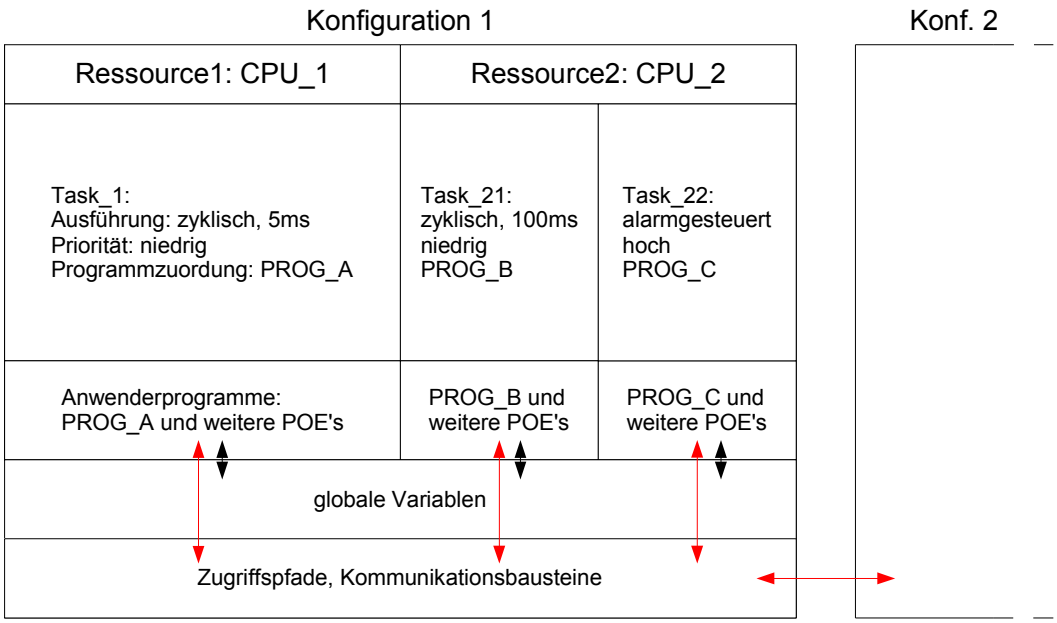
Die Konfiguration und die Ressourcen werden als Konfigurationsorganisationseinheiten (KOE) bezeichnet. Mittels der KOE's erfolgt die Anpassung der SW an die HW.

Auf jeder Ressource laufen ein oder bei Multitaskingfähigkeit mehrere Rechenprozesse (Tasks). Unter der Kontrolle der Tasks werden verschiedene Softwaremodule abgearbeitet. Diese Softwaremodule werden als Programmorganisationseinheiten (POE) bezeichnet.

Zur Kommunikation zwischen POE's, Tasks und Konfigurationen zu Fremdsystemen werden Variablen verwendet:

- lokale Schnittstellenvariablen
- globale Variablen
- Zugriffspfade, Kommunikationsbausteine

Softwaremodell eines SPS-Projekts



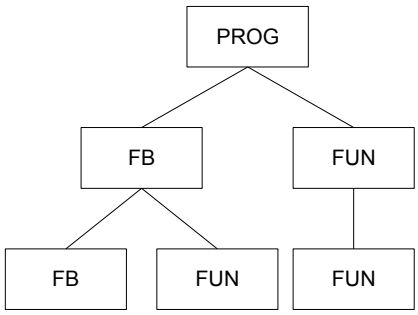
Programmorganisationseinheit (POE's)

Eine POE ist eine abgeschlossene Softwareeinheit. Die POE's werden je nach Funktionalität in drei Typen eingeteilt:

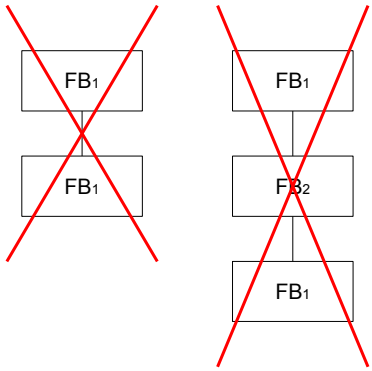
FUN: Funktion (FUNCTION)
Die Funktion liefert einen Rückgabewert (Funktionswert), der nur von den Eingangsparametern der Funktion abhängt.
Die Funktion besitzt keine statische Variablen, d.h. sie hat keine Gedächtnisfunktion.

FB: Funktionsbaustein (FUNCTION_BLOCK)
Der FB besitzt gegenüber der FUN einen eigenen lokalen Speicher, d.h. er besitzt statische Variablen und hat damit eine Gedächtnisfunktion.
Für die Ausführung eines FB's wird eine Instanz angelegt mit einem neuen Namen (Instanzname).
Jede so erzeugte Instanz eines FB's hat ihren eigenen lokalen Speicher.

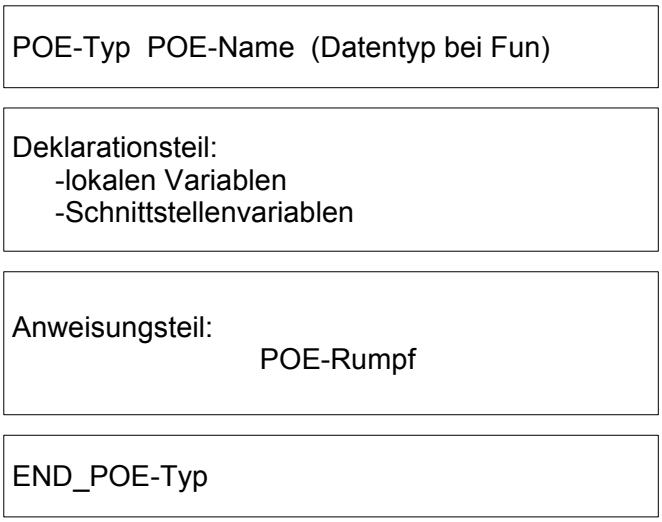
PROG: Programm (PROGRAM)
Das Programm hat die Funktionalität eines FB's. Es kann zusätzlich auf die SPS-HW zugreifen und diese den anderen POE's zur Verfügung stellen.
Das PROG entspricht in etwa dem OB der alten Norm.
Aufrufhierarchie:



Ein rekursiver Aufruf einer POE ist nicht erlaubt, auch nicht indirekt:



Struktur einer POE:



POE-Typ:	FUNCTION FUNCTION_BLOCK PROGRAM
POE-Name:	Name der POE, ist vom Ersteller der POE zu vergeben. Dieser Name ist projektweit bekannt und darf daher nur einmal vorkommen.
Datentyp bei FUN:	Festlegung des Rückgabewerts der Funktion

Deklarationsteil

Alle Variablen einer POE werden aufgelistet unter Festlegung
der Variablenart und
des Datentyps
Außerdem kann bei Bedarf zusätzlich festgelegt werden:
Anfangswert
Feldgrenzen
physikalische Adresse
Attribut

Die Deklaration erfolgt blockweise nach Variablenwert, z.B.

```
VAR_INPUT (*Deklaration von Eingangsvariablen einer POE*)
        EIN:    BOOL;
        ...
END_VAR
```

Erläuterung:

VAR_INPUT	Variablenart (hier Eingsangsvariable einer POE)
EIN	Variablenname
BOOL	Datentyp

Anweisungsteil

Befehlsfolge, die bei Aufruf der POE von der CPU ausgeführt wird.

Textuelle Sprachen:

- Anweisungsliste (AWL), engl.: Instructionlist (IL)
  Assemblerähnliche Programmiersprache
  vor allem in Europa verbreitet
- Strukturierter Text (ST), engl.: Structured Text (ST)
  Problemorientierte Hochsprache, an Pascal angelehnt mit SPS-spezifischen Erweiterungen

Graphische Sprachen:

- Kontaktplan (KOP), engl.: Ladder Diagram (LD)
  Entspricht einem Stromlaufplan (Kontakte, Spulen, Kästen),
  vor allem im angelsächsischen Sprachraum und in der Automobilbranche verbreitet
- Funktionsbausteinsprache (FBS), engl.: Functional Block Diagram (FBD)
  Programmdarstellung als Logikplan (siehe FUP),
  parallel zur AWL in Europa verbreitet
- Ablaufsprache (AS), engl.: Sepuential Function Chart (SFC)
  Beschreibung von Ablaufketten in Form von Schritten und Transitionen

In der alten Norm wurden bereits
AWL, KOP, FBS (FUP), und AS
benutzt.

Bei den POE's FUN und FB sind auch weitere Programmiersprachen erlaubt, sofern

- die Variablen normgerecht verwendet werden,
- und die Ausführung der Festlegung der Norm entspricht.

Datentyp (Dt)

Je nach Dt wird den Daten ein entsprechender Speicherplatz zugewiesen.

Je nach Dt können auf Grund ihrer Eigenschaft mit diesen Daten nur ganz bestimmte Operationen ausgeführt werden.

Vorteile:

- Daten werden typgerecht einheitlich gespeichert.
- bessere Portierbarkeit
- Überprüfung einer typgerechten Verwendung der Daten
- Verringerung des Fehlerpotenzials
- effizientere Wartung

<u>Elementare Dt's</u>

Von der Norm vorgegebene standardisierte Dt's mit der Festlegung von :

- Kennzeichnung z.B. BOOL, INT
- Speicherbedarf z.B. 1 Bit, 16 Bit
- Wertebereich z.B. 0/FALSE und 1/TRUE, $-2^{15}, \ldots, +2^{15}-1$
- Initialwert z.B. FALSE, 0

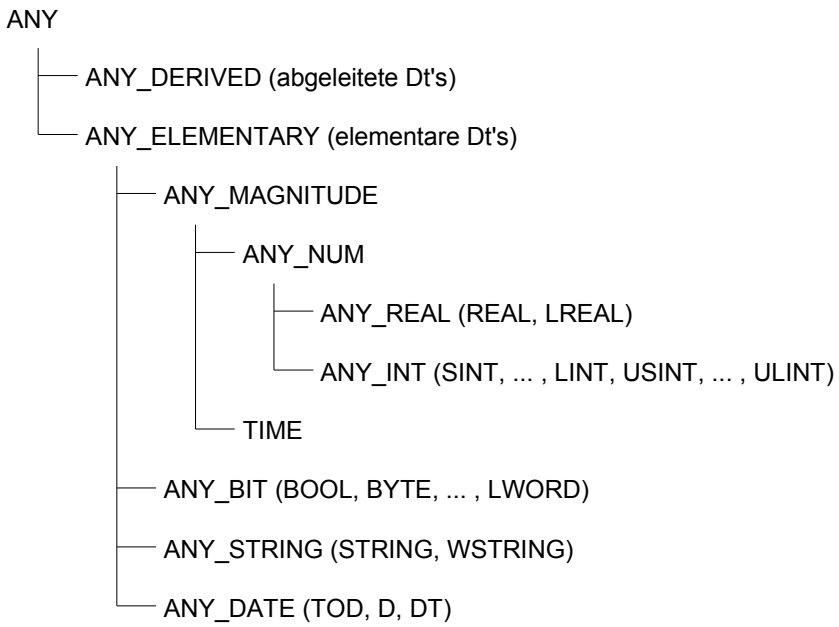
Bei einigen Dt's sind Speicherplatzbedarf und Wertebereich implementierungsabhängig (Datum, Uhrzeit und String)

Beispiele:

Schlüsselwort	*Datentyp*	*Größe*	*Schreibweise und Wertebereiche*	*Anfangswert*
Bit-Datentype				
BOOL	Boolsche Variablen	1 Bit	FALSE (0), TRUE (1)	FALSE (0)
BYTE	Bit-Folge und 8 Bit-Hex-Zahlen	8 Bit	(B#)16# 0...FF	00
WORD	Bit-Folge und 16 Bit-Hex-Zahlen	16 Bit	(W#)16# 0000...FFFF	0000
DWORD	Bit-Folge und 32 Bit-Hex-Zahlen	32 Bit	(DW#)16# 0000_0000...FFFF_FFFF	0000 0000
LWORD	Bit-Folge und 64 Bit-Hex-Zahlen	64 Bit		0000 0000 0000 0000
CHAR	ASCII-Zeichen	8 Bit	'A'	
Arithmetische Typen				
SINT	Kurze ganze Zahlen (Festpunktzahlen)	8 Bit	-128 bis +127	0
INT	Ganze Zahlen (Festpunktzahlen)	16 Bit	-32768 bis +32767	0
DINT	Doppelte ganze Zahlen (Festpunktzahlen)	32 Bit	L#-2147483648 bis +2147483647	0
LINT	Lange ganze Zahlen (Festpunktzahlen)	64 Bit	-2^{63} bis $2^{63}-1$	0
USINT	Vorzeichenlose kurze ganze Zahlen (Festpunktzahlen)	8 Bit	0 bis 255	0
UINT	Vorzeichenlose ganze Zahlen (Festpunktzahlen)	16 Bit	0 bis 65535	0
UDINT	Vorzeichenlose doppelte ganze Zahlen (Festpunktzahlen)	32 Bit	0 bis 4294967295	0
ULINT	Vorzeichenlose lange ganze Zahlen (Festpunktzahlen)	64 Bit	0 bis $2^{64}-1$	0
REAL	Reelle Zahlen (Gleitpunktzahlen)	32 Bit	Dezimalzahl mit Punkt: 341.7 oder Exponentialdarstellung: 3.417 E+02	0.0
LREAL	Lange reelle Zahlen (Gleitpunktzahlen)	64 Bit		0.0
Zeittypen				
S5TIME	Zeitdauer (S5-Format)	16 Bit	S5T#0ms bis 9990s	
TIME	Zeitdauer (IEC-Format)	32 Bit	TIME#-24d20h31m bis +24d20h31m	TIME#0s
TIME OF DAY	Uhrzeit (Tageszeit)	32 Bit	TIME_OF_DAY#23:59:59,9	TIME_OF_DAY#00:00:00
DATE	Datum	16 Bit	DATE#1990-01-01	DATE#0001-1-01
DATE_AND_TIME	Datum und Uhrzeit			DATE_AND_TIME#0001-01-01-00:00:00
Zeichentypen				
Zeichenfolge (1-Byte)	STRING			" leere Zeichenkette
Zeichenfolge (2-Byte)	WSTRING			" "leere Zeichenkette

Allgemeine oder generische Dt's

Zusammenfassung von Dt's mit ähnlichen Eigenschaften zu Gruppen:



Operationen, FUN's oder FB's, die nicht auf einem Dt festgelegt sind, bezeichnet man als *überladbar* oder *nicht typisiert*.
Ist jedoch nur ein Dt zulässig, so bezeichnet man sie als *typisiert*.

Beispiel einer überladbaren Funktion: MUL

Überladbare Op./FUN/FB's lassen sich typisieren durch Anhängen eines Dt-Suffix, z.B.

MUL → MUL_INT (nur INT-Daten dürfen damit multipliziert werden) (_INT == Suffix)

Abgeleitete Dt's

Auf Basis von elementaren oder bereits abgeleiteten Dt's vom Anwender selbst definierte Dt's
→ abgeleitete Dt's

Deklaration eines abgeleiteten Dt's:

```
TYPE
      Dt_Name: ElternDt [ (Bereichseinschränkung) ] [ Initialisierung ];
      Dt_Name: (Aufzählungstypen) [ Initialisierung ];
END_TYPE
```

Als ElternDt kommt in Frage:

- EinzelelementDt's
  einfache Dt's
  BereichsDt's
  AufzählungsDt's
- MultielementDt's
  Arrays
  Strukturen
- StringDt's

Beispiel:

Einfache Dt-Ableitung

```
TYPE
      INT1: INT := 1;
END_TYPE
```

INT1 ist vom Dt INT mit Initialwert 1.

BereichsDt's

Dt_Name: ElternDt (Untergrenze .. Obergrenze);
Erlaubte ElternDt's: ANY_INT
Dt's, die von ANY_INT abgeleitet sind, außer BereichsDt

```
TYPE
      ANZ: USINT(0..10) := 1;
END_TYPE
```

ANZ ist vom Dt USINT, aber auf die Zahlen von 0 bis 10 begrenzt und dem Initialwert 1.

AufzählungsDt's

```
TYPE
      WOCHE: (Mo, Di, Mi, Do, Fr, Sa, So);
      WO_ENDE: (Sa, So);
      AMPEL: (grün, gelb, rot) := rot;
END_TYPE
```

WOCHE kann die Bezeichnungen Mo, Di, ... , So annehmen.
Wo_ENDE kann Sa, So und AMPEL grün, gelb, rot annehmen.

Erstes Element ist defaultmäßig Initialwert, Ampel ist explizit mit rot initialisiert.

Die Verwendung von Aufzählungselementen im Bausteinrumpf erfordert eine Ergänzung des Elementnames mit dem Präfix "Dt-Name#", z.B. `WOCHE#Di` oder `AMPEL#grün`

FeldDt ARRAY

Zusammenfassung mehrerer Elemente gleichen Dt's.

Dekl.:

```
Feldname: ARRAY   [ Bereich ] OF Dt;
      ............[ Bereiche ] OF Dt;
```

Bereich: Untergrenze .. Obergrenze mit Unter-/Obergrenze vom Dt ANY_INT

Beispiele:

```
TYPE
      Feld1: ARRAY [ 1..10 ] OF INT;
      Feld2: ARRAY [ 1..10, 1..2 ] OF BOOL;
END_TYPE
```

Initialisierungen:

```
:= [ 1,2,3,4,5,6 ]
:= [ 1,2,3,1,2,3 ] oder :=[ 2(1,2,3) ]
```

```
TYPE
      Feld3: ARRAY [ 1..5, 1..2 ] OF BOOL := [ 6(TRUE) ];
END_TYPE
```

Initialisierung: Feld3[i,1] = TRUE , i =1,...,5
Feld3[1,2] = TRUE
Feld3[i,2] = FALSE , i = 2,...,5

StrukturDt STRUCT

Umfasst mehrere Elemente, die unterschiedliche Dt'en besitzen können.
Wenn ein Element vom Dt ARRAY oder STRUCT ist, dann spricht man von hierarchischen Strukturen.

Beispiel einer STRUCT-Ableitung:

```
TYPE
      Name: STRUCT
                  Messung:    ARRAY[ 0..32, 1..2 ] OF REAL;
                  U, I, P:    REAL;
                  TAG:        WOCHE;
                  XYZ:        Feld1 := [ 10(1) ];
      END_STRUCT;
END_TYPE
```

StringDt-Ableitung

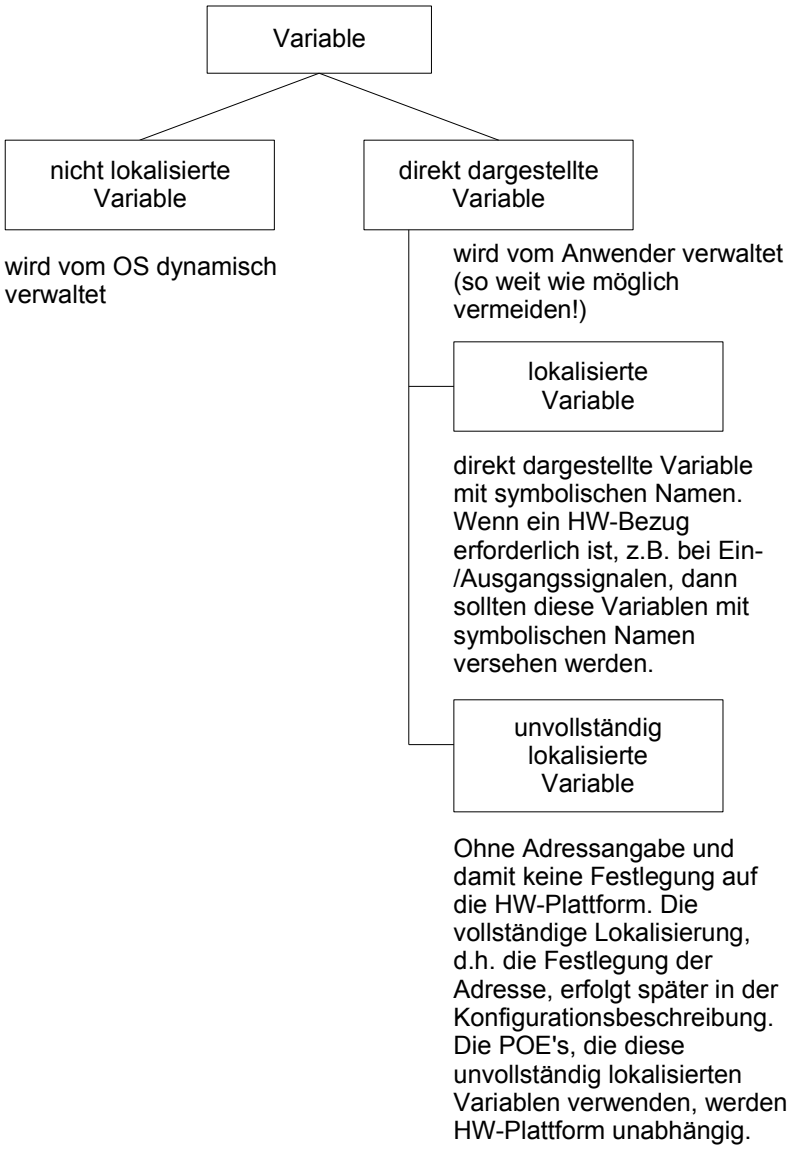
```
TYPE
      Name1: STRING[ n ];                 Name1 darf max n Zeichen (1 Byte) lang sein.
      Name2: STRING[ 10 ] := 'ABC';       mit Initialisierung der ersten 3 Bytes mit A, B und C
      Name3: WSTRING[ 5 ];          String mit maximal 5 2-Byte-Zeichen
END_TYPE
```

Variablen

Die Variable ist in dieser neuen Norm von zentraler Bedeutung.
Die Variable besitzt einen Namen, hinter der sich ein Wert verbirgt, der während des Programmablaufs veränderlich sein kann (mit Ausnahme der konstanten Variablen).
Eigenschaften einer Variablen werden bestimmt durch

- Variablenart (Gültigkeitsbereich, z.B. Eingangsvariable)
- zugeordneter Dt (Speichergröße, Speicherstruktur, zulässige Operationen)
- Attribut (optional, z.B. remanent)

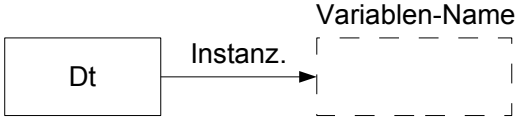
Klassifikation von Variablen:



Grundsätzliche Regeln:

– Alle in einer KOE oder POE verwendeten Variablen müssen in dieser KOE oder POE deklariert werden. Die Deklaration einer Variablen erfolgt in dem Deklarationsteil der KOE bzw. POE.

– Jeder Variablen muss ein Dt zugeordnet werden. Dieser bestimmt die Speichergröße, Speicherstruktur und zulässige Operationen der Variablen.
Die Reservierung eines dem Dt entsprechenden Speicherplatzes mit Namensgebung bezeichnet man als Instanzbildung oder Instanzierung



– Einteilung in drei Gruppen:
interne Variablen
lokale Schnittstellenvariablen
globale Schnittstellenvariablen

Variablenarten:

Variablengruppe	Variablenart	Schlüsselwort der Variablenart
interne Variablen	lokale Variablen	VAR
	temporäre Variablen	VAR_TEMP (seit der 2nd Edition)
lokale Schnittstellenvariablen	Eingangsvariablen	VAR_INPUT
	Ausgangsvariablen	VAR_OUTPUT
	Ein-/Ausgangsvariablen	VAR_IN_OUT
globale Schnittstellenvariablen	globale Variablen	VAR_GLOBAL
	externe Variablen	VAR_EXTERNAL
	Zugriffspfade der Konfiguration	VAR_ACCESS
	Instanzspezifische Variablen der Konfiguration	VAR_CONFIG

– Die Deklaration von Variablen findet blockweise nach Variablenart statt.

Definitionsblock:

Schlüsselwort der Variablenart [Attribut] ([..] == optional)
Deklaration von Variablen
END_VAR

Struktur von Variablendeklarationen:

- nicht lokalisierte Variable:

```
Name: Dt [ Attribut ] [ := Initialwert ];
```

Weisen mehrere Variablen dieselbe Deklaration auf, dann lassen sie sich als Liste zusammenfassen, z.B.

```
X, Y, Z : INT;
```

- direkt dargestellte Variable:

```
AT „direkt dargestellte Variable“: Dt [ Attribut ] [ := Initialwert ]
```

Direkt dargestellte Variablen setzen sich zusammen aus:

– %
– Buchstabe, der den Variablenbereich kennzeichnet (I, Q, M)
– Buchstabe für den Datentyp (X, B, W, D, L)
– Adresse im Variablenbereich

z.B.
```
%IX0.0 == E0.0
%QB3   == AB3
%MW3   == MW3
```

- lokalisierte Variable

```
Name AT „direkt dargestellte Variable“: Dt [ Attribut] [ := Initialisierung ];
```

Direkt dargestellte und lokalisierte Variablen lassen sich nur in PROG verwenden.

- unvollständig lokalisierte Variable

Deklaration ist nur erlaubt in PROG und FB

```
Name AT %IX*: BOOL;
```
*: Platzhalter für die Adresse

Die Vervollständigung der Lokalisierung erfolgt in der Deklaration für VAR_CONFIG:

```
VAR_CONFIG
      RessourceName.PROG_Name[ .FB_Name ] Name AT %IX0.0: BOOL;
END_VAR
```

Attribute

Zusätzliche Zuordnung von Eigenschaften (optional).

```
RETAIN, NON_RETAIN                  ; remanent, nicht remanent (2nd Edition)
CONSTANT                            ; Konstante
R_EDGE, F_EDGE                      ; steigende, fallende Flankenerkennung
READ_ONLY, READ_WRITE               ; nur lesbar, les- und schreibbar
AT                                  ; direkt dargestellte Variable
```

RETAIN, NON_RETAIN, CONSTANT sind nur für den ganzen Block zulässig:

```
VAR_xxx     RETAIN
            NON_RETAIN
            CONSTANT
```

Programmorganisationseinheiten (POE)

Funktion (FUN)

Eine Funktion liefert bei gleichen Eingangsparametern immer das gleiche Ergebnis: Rückgabewert (Einzelelement, Multielement, String)

Eingangsvariablen (VAR_INPUT) sind zwingend notwendig. Lokale Variablen (VAR) sind möglich, besitzen aber nur temporären Charakter.

Seit der 2nd Edition sind auch VAR_OUTPUT und VAR_IN_OUT zulässig.

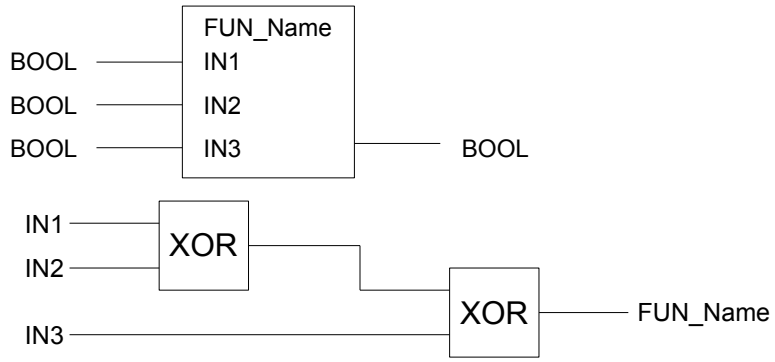
Beispiel der Deklaration der Funktion FUN_Name:

textuelle Form:

```
FUNCTION FUN_Name: BOOL
      VAR_INPUT
            IN1, IN2, IN3: BOOL;
      END_VAR
      LD          IN1
      XOR         IN2
      XOR         IN3
      ST          FUN_Name
END_FUNCTION
```

graphische Form:

```
FUNCTION
```




```
END_FUNCTION
```

Aufruf einer Funktion ist in textueller und graphischer Form möglich.

In AWL ist allgemein zu beachten:

Der erste Eingangsparameter muss sich vor dem Funktionsaufruf als aktuelles Ergebnis (AE) im Ergebnisspeicher befinden.

Der Funktionsname wird wie ein Operator verwendet, d.h. die restlichen Eingangsparameter werden durch Komma getrennt nach dem Funktionsnamen aufgelistet.

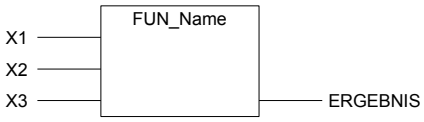
Der Rückgabewert befindet sich dann nach der Ausführung der Funktion als AE im Ergebnisspeicher und kann als AE weiterverarbeitet werden.

Aufrufmöglichkeiten in einer POE mit den dort deklarierten Variablen X1, X2, X3, ERGEBNIS:

AWL: graphisch:

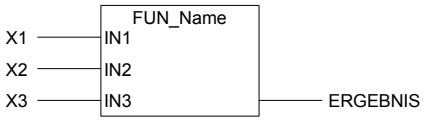
mit nonformaler Parameterliste

```
LD    X1
FUN_Name X2, X3
ST    ERGEBNIS
```



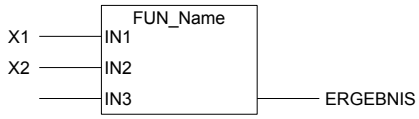
mit formaler Parameterliste

```
LD    X1
FUN_Name IN3:=X3, IN2:=X2
ST    ERGEBNIS
```



mit unvollständigem Aufruf

```
LD    X1
FUN_Name IN2:=X2
ST    ERGEBNIS
```



Bemerkung: Der fehlende Wert von X3 wird durch den Initialwert des Dt's von IN3 ersetzt.

Seit der 2nd Edition (AWL):

```
FUN_Name (
      IN1:=X1,
      IN2:=X2,
      IN3:=X3
      )
ST          ERGEBNIS
```

Aufrufmöglichkeiten im Strukturierten Text (ST):

nonformale Parameterliste:

```
ERGEBNIS := FUN_Name(X1, X2, X3)
```

formale Parameterliste:

```
ERGEBNIS:=FUN_Name(IN1:=X1, IN3:=X3, IN2:=X2)
```

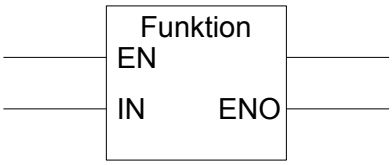
unvollständige Parameterliste:

```
ERGEBNIS:=FUN_Name(IN1:=X1, IN3:=X3)
```

Ausführungssteuerung einer Funktion

Die Ausführung kann in KOP und FBS mit Hilfe zweier boolschen Variablen EN und ENO kontrolliert werden. Diese Variablen werden vom System deklariert:

```
VAR_INPUT EN: BOOL := TRUE (1); END_VAR
VAR_OUTPUT ENO: BOOL; END_VAR
```



Seit der 2nd Edition ist die Verwendung auch in AWL und ST möglich.

Bedeutung von EN und ENO:

EN bei Fkt's-Aufruf	Möglichkeit der Ausführung der Funktion	ENO bei Verlassen der Funktion
FALSE (0)	keine Ausführung der Funktion	FALSE (0)
TRUE(1)	Ausführung der Funktion ohne Fehlererkennung ENO kann während der Ausführung beeinflusst werden: - ohne Beeinflussung - mit Beeinflussung	 TRUE(1) individueller Wert
TRUE(1)	Auftritt von Fehlern während der Ausführung	FALSE(0)

Standardfunktionen

Schnittstellen und Verhalten nach außen gegenüber der aufgerufenen POE sind bei Standardfunktionen durch die Norm vorgeschrieben.

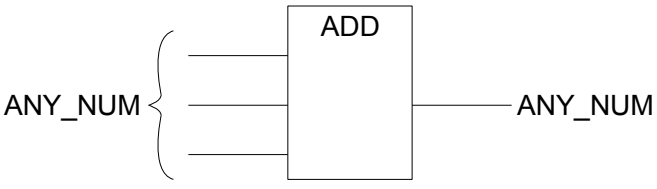
Kann die Anzahl der Eingänge von Anwendern bestimmt werden, dann spricht man von erweiterbaren Standardfunktionen, z.B. AND, OR, XOR

Beispiele von Standardfunktionen:

Funktionen zur Typumwandlung	INT_TO_REAL, ...
Numerische Funktionen	ABS, SQRT, EXP, ...
Bit-Folge-Funktionen	AND, OR, SHL, SHR
Auswahl- und Vergleichsfunktionen	SEL, MUX, MIN, LIMIT, EQ, NE
Zeichenfolgefunktionen	LEN(STRING)
Funktionen für Zeit-Dt's	ADD oder ADD_TIME (seit 2nd Edition)
Funktionen für Aufzählungs-Dt's	SEL, MUX, EQ, NE, ...

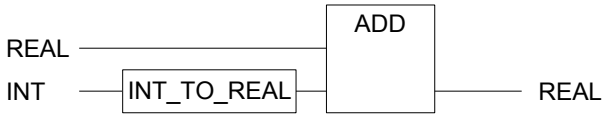
Überladen von Funktionen

Funktionen, die mehrere Dt's als Eingangsgrößen zulassen, heißen überladbar oder nicht typisiert, z.B. die Funktion ADD.

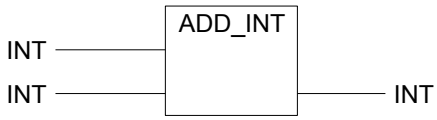


Die verwendeten Eingangsparameter müssen aber immer den selben Dt haben.

Unterscheiden sich die Dt's der Eingangsparameter, dann muss eine Typangleichung stattfinden, z.B.



Untypisierte Standardfunktionen lassen sich typisieren, z.B.



Vorteil: schneller, weniger aufwendig
Nachteil: aufwendigere Softwarepflege

Funktionsbaustein (FB)

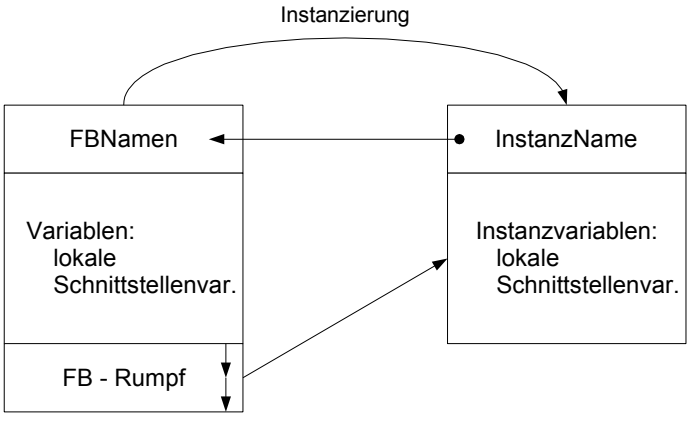
Deklaration:

```
FUNCTION_BLOCK FBName
      [ Variablen-Dekl. ]
      [ FB-Rumpf ]
END_FUNCTION_BLOCK
```

Variablen-Deklarationen:

- lokale Variablen (statisch, temporär seit 2nd Edition)
- lokale Schnittstellenvariablen
- unvollständig lokalisierte Variablen
- globale Variablen (`EXTERNAL`)

Zur Benutzung eines FB's muss in der aufrufenden POE eine Instanz dieses FB's gebildet werden.

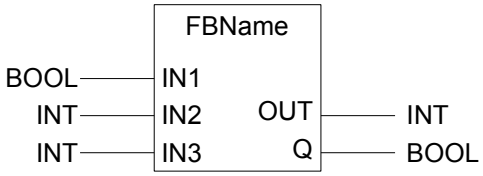


Die Instanzierung erfolgt in einem Deklarationsblock, z.B. In der aufrufenden POE:

```
VAR
      U, I, P: REAL;
      MOTOR1: MOTOR;      MOTOR ist ein FB
      MOTOR2: MOTOR;
END_VAR
```

Möglichkeiten von FB-Aufrufen:

gegeben ist folgender FB:



Deklaration von Variablen und Instanz von FBName in der aufrufenden POE:

```
VAR
      Var1, Var2, Var3: INT;
      EIN, AUS: BOOL;
      Inst_Name: FBName;
END_VAR
```

Aufruf in der AWL kann erfolgen

– unbedingt: `CAL Inst_Name( Eingangsparameter )`

– bedingt: `CALC Inst_Name( Eingangsparameter )`, wenn AE = 1
`CALCN Inst_Name( Eingangsparameter )`, wenn AE = 0

Aufruf mit nonformaler Parameterliste:

```
CAL Inst_Name( EIN, Var1, Var2 )
LD Inst_Name.Q
ST AUS
LD Inst_Name.OUT
ST Var3
```

Aufruf mit formaler Parameterliste:

```
CAL Inst_Name( IN2:=Var1, IN3:=Var2, IN1:=EIN )
LD Inst_Name.Q
ST AUS
LD Inst_Name.OUT
ST Var3
```

Seit der 2nd Edition ist auch folgender Aufruf möglich:

```
CALL(
      IN1 := EIN,
      IN2 := Var1,
      IN3 := Var2,
      OUT => Var3,
      Q => AUS );
```

=> Ausgabeoperator

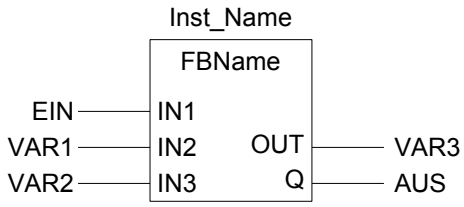
ST-Aufruf mit formaler Parameterliste:

```
Inst_Name( IN1:=EIN, IN2:=Var1, IN3:=Var2 );
AUS := Inst_Name.Q;
Var3 := Inst_Name.OUT;
```

ST-Aufruf mit nonformaler Parameterliste:

```
Inst_Name( EIN, Var1, Var2, Var3, AUS );
```

graphischer Aufruf:



Standard-Funktionsbausteine

Beispiel:

RS, SR (RS-Flip-Flop)
R_TRIG, F_TRIG (Flankenauswertung)
CTU, CTD, CTUD (Zähler)
TP, TON, TOF (Timer)
RTC (Echtzeituhr)
READ, WRITE (FB's zur Kommunikation)
...

Programm (PROG)

Funktionalität wie FB

zusätzliche Deklarationsmöglichkeiten:

- direkt dargestellte Variablen
- lokalisierten Variablen
- globale Variablen (GLOBAL)
- Zugriffspfade mittels VAR_ACCESS

Bei multitaskingfähigen Ressourcen (CPU's, ...) können mehrere Instanzen eines Programms mit eigenen Variablen und Hardwareadressen verschiedene Laufzeitprogramme bilden.

Deklaration eines Programms:

```
PROGRAM ProgName
      [ Variablen-Dekl. ]
      [ PROG-Rumpf ]
END_PROGRAM
```

9 STEP 7-Programmierung

Simatic-Manager:

graphische Benutzeroberfläche zur Projektrealisierung:

HW-Konfiguration:
- Ressourcen (CPU, ...)
- Baugruppen
- Baugruppen-Adressen
- Parametrierung

SW-Konfiguration:
- POE's
- globale Variablen
- Tasks
- Kommunikation

Darstellung der Projektstruktur:

Projektname

SPS-Familie (z.B. S7 300)

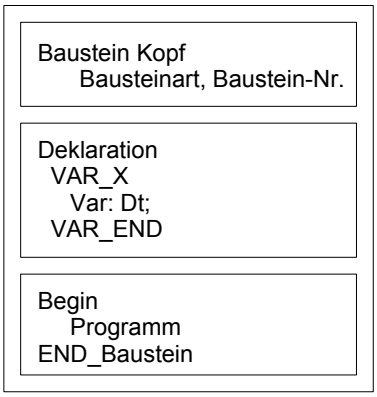
CPU (Ressource, z.B. CPU 315-2DP)

S7-Programme

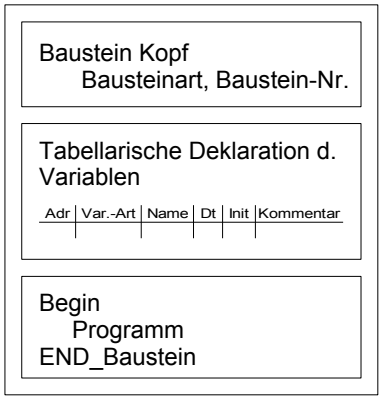
Quellen (S7-Quellprogramme)
Bausteine (OB, FB, FC, DB) — OB, FB, FC == POE's; DB == Instanz eines FB's

POE-Struktur:

quellenorientierter Editor



inkrementeller Editor

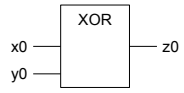


10 Anhang/SPS-Beispielprogramme

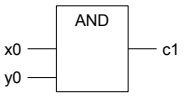
10.1 Volladdierer

```
0001 PROGRAM OB1
0002 VAR
0003     x0 AT %IX0.0: BOOL;
0004     x1 AT %IX0.1: BOOL;
0005     x2 AT %IX0.2: BOOL;
0006     y0 AT %IX1.0: BOOL;
0007     y1 AT %IX1.1: BOOL;
0008     y2 AT %IX1.2: BOOL;
0009     z0 AT %QX2.0: BOOL;
0010     z1 AT %QX2.1: BOOL;
0011     z2 AT %QX2.2: BOOL;
0012     z3 AT %QX2.3: BOOL;
0013     c1, c2: BOOL;
0014 END_VAR
0015
0016
0017
0018 VAR
0019     S_E_T_A_S_S_E_T: BOOL;
0020
     END_VAR
```

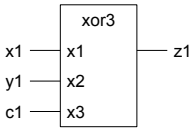
0001 Halbaddierer für z0



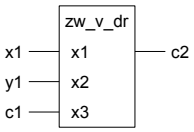
0002



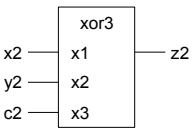
0003 Volladdierer für z1



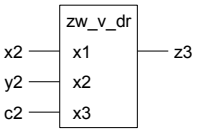
0004

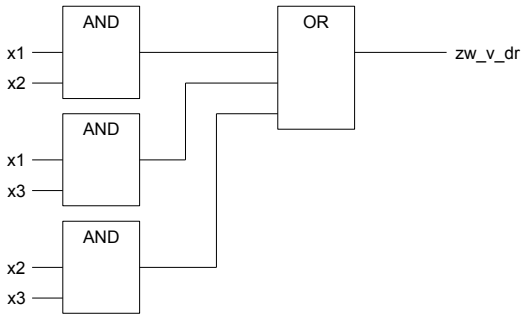


0005 Volladdierer für z2 und z3



0006



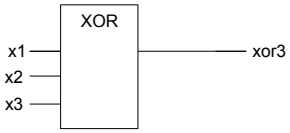
0001	FUNCTION zw_v_dr : BOOL
0002	VAR_INPUT
0003	x1, x2, x3: BOOL;
0004	END_VAR
0005	VAR
0006	END_VAR
0007	VAR
0008	S_E_T_A_S_S_E_T: BOOL;
0009	
0010	END_VAR
0001	

```
0001  FUNCTION zw_v_dr : BOOL
0002  VAR_INPUT
0003      x1, x2, x3: BOOL;
0004  END_VAR
0005  VAR
0006  END_VAR
0007  VAR
0008      S_E_T_A_S_S_E_T: BOOL;
0009
0010  END_VAR
```

```
0001  LD    x1
0002  AND   x2
0003  OR(    x1
0004  AND   x3
0005  )
0006  OR(    x2
0007  AND   x3
0008  )
0009  ST    zw_v_dr
0010
```

```
0001  FUNCTION xor3 : BOOL
0002  VAR_INPUT
0003      x1, x2, x3: BOOL;
0004  END_VAR
0005  VAR
0006  END_VAR
0007  VAR
0008      S_E_T_A_S_S_E_T: BOOL;
0009
0010  END_VAR
```

0001



```
0001 FUNCTION xor3 : BOOL
0002 VAR_INPUT
0003     x1, x2, x3: BOOL;
0004 END_VAR
0005 VAR
0006 END_VAR
0007 VAR
0008     S_E_T_A_S_S_E_T: BOOL;
0009
0010 END_VAR
```

```
0001 LD     x1
0002 XOR(     x2
0003 XOR     x3
0004 )
0005
```

10.2 Füllstandsanzeige

SIMATIC ...-Station\CPU315-2 DP(1)\...\OB1-<offline> 23.01.2005 17:48:24

OB1-<offline>

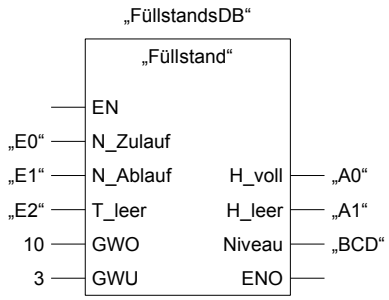
"Füllstandsanzeige"

Name: **Familie:**
Autor: **Version:** 0.1
Bausteinversion: 2
Zeitstempel Code: 23.01.2005 17:46:52
Interface: 23.01.2005 17:46:52
Längen (Baustein / Code / Daten): 00236 00118 00026

Adresse	Deklaration	Name	Typ	Anfangswert	Kommentar
0.0	temp	OB1_EV_CLASS	BYTE		Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
1.0	temp	OB1_SCAN_1	BYTE		1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of OB 1)
2.0	temp	OB1_PRIORITY	BYTE		Priority of OB Execution
3.0	temp	OB1_OB_NUMBR	BYTE		1 (Organization block 1, OB1)
4.0	temp	OB1_RESERVED_1	BYTE		Reserved for system
5.0	temp	OB1_RESERVED_2	BYTE		Reserved for system
6.0	temp	OB1_PREV_CYCLE	INT		Cycle time of previous OB1 scan (milliseconds)
8.0	temp	OB1_MIN_CYCLE	INT		Minimum cycle time of OB1 (milliseconds)
10.0	temp	OB1_MAX_CYCLE	INT		Maximum cycle time of OB1 (milliseconds)
12.0	temp	OB1_DATE_TIME	DATE_AND_TIME		Date and time OB1 started

Baustein: OB1 Füllstandsanzeige

Netzwerk: 1



SIMATIC ...-Station\CPU315-2 DP(1)\...\OB1-<offline> 23.01.2005 17:48:35

OB1-<offline>

"Füllstandsanzeige"

Name: **Familie:**
Autor: **Version:** 0.1
Bausteinversion: 2
Zeitstempel Code: 23.01.2005 17:46:52
Interface: 23.01.2005 17:46:52
Längen (Baustein / Code / Daten): 00236 00118 00026

Adresse	Deklaration	Name	Typ	Anfangswert	Kommentar
0.0	temp	OB1_EV_CLASS	BYTE		Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
1.0	temp	OB1_SCAN_1	BYTE		1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of OB 1)
2.0	temp	OB1_PRIORITY	BYTE		Priority of OB Execution
3.0	temp	OB1_OB_NUMBR	BYTE		1 (Organization block 1, OB1)
4.0	temp	OB1_RESERVED_1	BYTE		Reserved for system
5.0	temp	OB1_RESERVED_2	BYTE		Reserved for system
6.0	temp	OB1_PREV_CYCLE	INT		Cycle time of previous OB1 scan (milliseconds)
8.0	temp	OB1_MIN_CYCLE	INT		Minimum cycle time of OB1 (milliseconds)
10.0	temp	OB1_MAX_CYCLE	INT		Maximum cycle time of OB1 (milliseconds)
12.0	temp	OB1_DATE_TIME	DATE_AND_TIME		Date and time OB1 started

Baustein: OB1 Füllstandsanzeige

Netzwerk: 1

```
U     "E0"
=     L       20.0
BLD   103
U     "E1"
=     L       20.1
BLD   103
U     "E2"
=     L       20.2
BLD   103
CALL  "Füllstand" , "FüllstandsDB"
 N_Zulauf     :=L20.0
 N_Ablauf     :=L20.1
 T_Leer       :=L20.2
 GWO    :=10
 GWU    :=3
 H_voll       :="A0"
 H_leer       :="A1"
 Niveau       :="BCD"
NOP   0
```

SIMATEC ...-Station\CPU315-2 DP(1)\...\FB1-<offline> 23.01.2005 17:36:59

FB1-<offline>

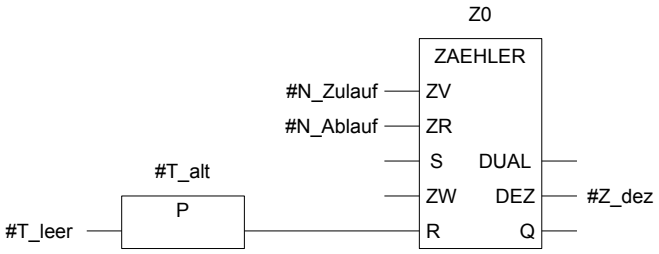
"Füllstand"

Name: **Familie:**
Autor: **Version:** 0.1
Bausteinversion: 2
Zeitstempel Code: 16.01.2005 18:53:16
Interface: 16.01.2005 18:31:32
Längen (Baustein / Code / Daten): 00210 00088 00000

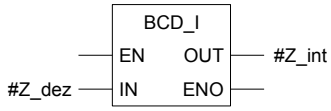
Adresse	Deklaration	Name	Typ	Anfangswert	Kommentar
0.0	in	N_Zulauf	BOOL	FALSE	
0.1	in	N_Ablauf	BOOL	FALSE	
0.2	in	T_leer	BOOL	FALSE	
2.0	in	GWO	INT	0	Grenzwert oben
4.0	in	GWU	INT	0	Grenzwert unten
6.0	out	H_voll	BOOL	FALSE	
6.1	out	H_leer	BOOL	FALSE	
8.0	out	Niveau	WORD	W#16#0	
	in_out				
10.0	stat	Z_dez	WORD	W#16#0	
12.0	stat	Z_int	INT	0	
14.0	stat	T_alt	BOOL	FALSE	
	temp				

Baustein: FB1 Füllstandsanzeige

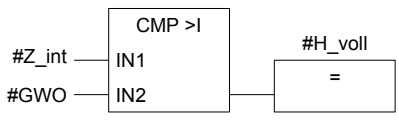
Netzwerk: 1 Zu- und Ablaufbilanz



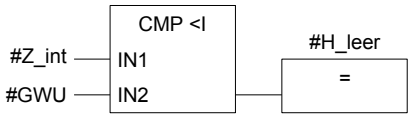
Netzwerk: 2 BCD-->INT



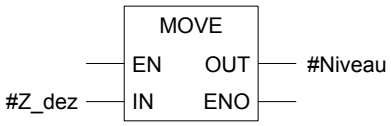
Netzwerk: 3 Grenzwertüberschreitung von GWO?



Netzwerk: 4 Grenzwertunterschreitung von GWU?



Netzwerk: 5 BCD-Anzeige



SIMATEC ...-Station\CPU315-2 DP(1)\...\FB1-<offline> 23.01.2005 17:42:09

FB1-<offline>

"Füllstand"

Name: **Familie:**
Autor: **Version:** 0.1
Bausteinversion: 2
Zeitstempel Code: 16.01.2005 18:53:16
Interface: 16.01.2005 18:31:32
Längen (Baustein / Code / Daten): 00210 00088 00000

Adresse	Deklaration	Name	Typ	Anfangswert	Kommentar
0.0	in	N_Zulauf	BOOL	FALSE	
0.1	in	N_Ablauf	BOOL	FALSE	
0.2	in	T_leer	BOOL	FALSE	
2.0	in	GWO	INT	0	Grenzwert oben
4.0	in	GWU	INT	0	Grenzwert unten
6.0	out	H_voll	BOOL	FALSE	
6.1	out	H_leer	BOOL	FALSE	
8.0	out	Niveau	WORD	W#16#0	
	in_out				
10.0	stat	Z_dez	WORD	W#16#0	
12.0	stat	Z_int	INT	0	
14.0	stat	T_alt	BOOL	FALSE	
	temp				

Baustein: FB1 Füllstandsanzeige

Netzwerk: 1 Zu- und Ablaufbilanz

```
      U     #N_Zulauf
      ZV    Z     0
      U     #N_Ablauf
      ZR    Z     0
      NOP   0
      NOP   0
      U     #T_leer
      FP    #T_alt
      R     Z     0
      NOP   0
      LC    Z     0
      T     #Z_dez
      NOP   0
```

Netzwerk: 2 BCD-->INT

```
      L     #Z_dez
      BTI
      T     #Z_int
      NOP   0
```

Netzwerk: 3 Grenzwertüberschreitung von GWO?

```
      L     #Z_int
      L     #GWO
      >I
      =     #H_voll
```

Netzwerk: 4 Grenzwertunterschreitung von GWU?

```
      L     #Z_int
      L     #GWU
      <I
      =     #H_leer
```

SIMATEC ...-Station\CPU315-2 DP(1)\...\FB1-<offline> 23.01.2005 17:36:59

Netzwerk: 5 BCD--Anzeige

```
      L     #Z_dez
      T     #Niveau
      NOP   0
```

SIMATEC ...-Station\CPU315-2 DP(1)\...\DB1-<offline> 23.01.2005 17:44:42

DB1-<offline>

"FüllstandsDB"

Name: **Familie:**
Autor: **Version:** 0.1
Bausteinversion: 2
Zeitstempel Code: 16.01.2005 18:49:40
Interface: 16.01.2005 18:31:32
Längen (Baustein / Code / Daten): 00126 00018 00000

Baustein: DB1

Adresse	Deklaration	Name	Typ	Anfangswert	Kommentar
0.0	in	N_Zulauf	BOOL	FALSE	
0.1	in	N_Ablauf	BOOL	FALSE	
0.2	in	T_leer	BOOL	FALSE	
2.0	in	GWO	INT	0	Grenzwert oben
4.0	in	GWU	INT	0	Grenzwert unten
6.0	out	H_voll	BOOL	FALSE	
6.1	out	H_leer	BOOL	FALSE	
8.0	out	Niveau	WORD	W#16#0	
10.0	stat	Z_dez	WORD	W#16#0	
12.0	stat	Z_int	INT	0	
14.0	stat	T_alt	BOOL	FALSE	

SIMATEC ...-ation\CPU315-2 DP(1)\S7-Programm (1)\Symbole 18.07.2004 19:35:44

Symboltabellen-Eigenschaften

Name:	Symbole
Kommentar:	
Erstellt am:	11.11.2003 22:38:03
Zuletzt geändert am:	18.07.2004..19:35:21
Letztes Filterkriterium:	Alle Symbole
Anzahl der Symbole:	9/ 9
Letzte Sortierung:	Symbol aufsteigend

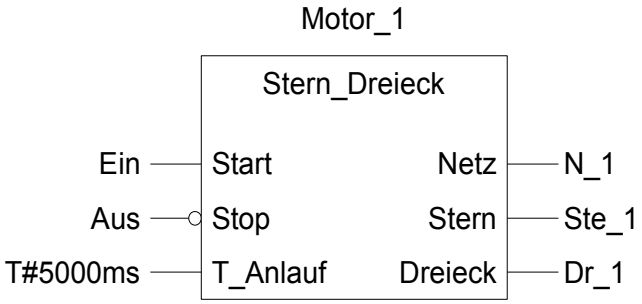
Symbol	Adresse	Datentyp	Kommentar
A0	A 0.0	BOOL	Behälter Voll
A1	A 0.1	BOOL	Behälter Leer
BCD	AW 1	WORD	BCD_Anzeige des Füllstands
E0	E 0.0	BOOL	DG Zulauf
E1	E 0.1	BOOL	DG Ablauf
E2	E 0.2	BOOL	Taster Leer
Füllstand	FB 1	FB 1	
Füllstandsanzeige	OB 1	OB 1	
FüllstandsDB	DB 1	FB 1	

10.3 Stern-Dreieck-Motor

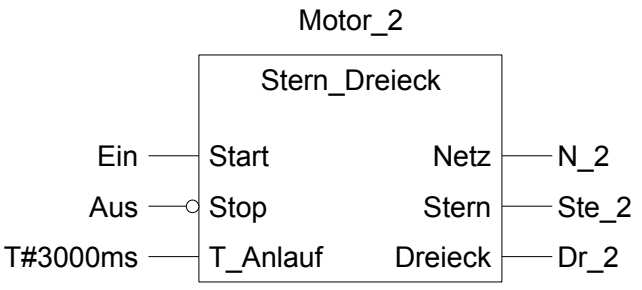
Variante 1

```
0001 PROGRAM OB1
0002 VAR
0003     Ein AT %IX0.0: BOOL;
0004     Aus AT %IX0.1: BOOL:=TRUE;
0005     N_1 AT %QX0.0: BOOL;
0006     Ste_1 AT %QX0.1: BOOL;
0007     Dr_1 AT %QX0.2: BOOL;
0008     Motor_1: Stern_Dreieck;
0009     N_2 AT %QX1.0 :BOOL;
0010     Ste_2 AT %QX1.1: BOOL;
0011     Dr_2 AT %QX1.2: BOOL;
0012     Motor_2: Stern_Dreieck;
0013 END_VAR
0014 VAR
0015     S_E_T_A_S_S_E_T: BOOL;
0016
0017 END_VAR
```

0001



0002

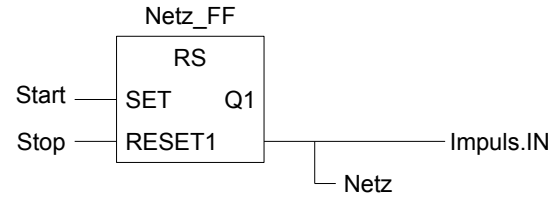


```
0001 PROGRAM OB1
0002 VAR
0003     Ein AT %IX0.0: BOOL;
0004     Aus AT %IX0.1: BOOL:=TRUE;
0005     N_1 AT %QX0.0: BOOL;
0006     Ste_1 AT %QX0.1: BOOL;
0007     Dr_1 AT %QX0.2: BOOL;
0008     Motor_1: Stern_Dreieck;
0009     N_2 AT %QX1.0: BOOL;
0010     Ste_2 AT %QX1.1: BOOL;
0011     Dr_2 AT %QX1.2: BOOL;
0012     Motor_2: Stern_Dreieck;
0013 END_VAR
0014 VAR
0015     S_E_T_A_S_S_E_T: BOOL;
0016
0017 END_VAR
```

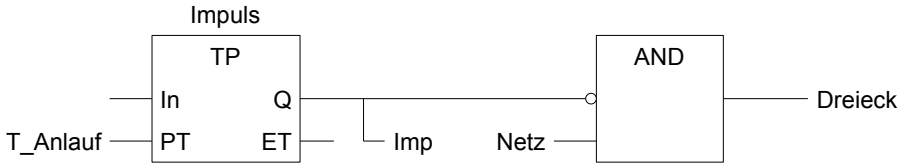
```
0001 LDN    Aus
0002 ST   Motor_1.Stop
0003 CAL    Motor_1(Start := Ein, T_Anlauf := T#5000ms)
0004
0005 LD   Motor_1.Stern
0006 ST   Ste_1
0007
0008 LD   Motor_1.Dreieck
0009 ST   Dr_1
0010
0011 LD   Motor_1.Netz
0012 ST   N_1
0013
0014 LDN    Aus
0015 ST   Motor_2.Stop
0016 CAL    Motor_2(Start := Ein, T_Anlauf := T#3000ms)
0017
0018 LD   Motor_2.Stern
0019 ST   Ste_2
0020
0021 LD   Motor_2.Dreieck
0022 ST   Dr_2
0023
0024 LD   Motor_2.Netz
0025 ST   N_2
0026
```

```
0001 FUNCTION_BLOCK Stern_Dreieck
0002 VAR_INPUT
0003     Start: BOOL;
0004     Stop: BOOL;
0005      T_Anlauf: TIME;
0006 END_VAR
0007 VAR_OUTPUT
0008     Netz: BOOL;
0009     Stern: BOOL;
0010     Dreieck: BOOL;
0011 END_VAR
0012 VAR
0013     Impuls: TP;
0014     Netz_FF: RS;
0015     Imp: BOOL;
0016 END_VAR
0017 VAR
0018     S_E_T_A_S_S_E_T: BOOL;
0019
0020 END_VAR
```

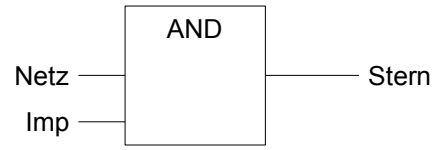
0001



0002



0003



```
0001 FUNCTION_BLOCK Stern_Dreieck
0002 VAR_INPUT
0003     Start: BOOL;
0004     Stop: BOOL;
0005      T_Anlauf: TIME;
0006 END_VAR
0007 VAR_OUTPUT
0008     Netz: BOOL;
0009     Stern: BOOL;
0010     Dreieck: BOOL;
0011 END_VAR
0012 VAR
0013     Impuls: TP;
0014     Netz_FF: RS;
0015     Imp: BOOL;
0016 END_VAR
0017 VAR
0018     S_E_T_A_S_S_E_T: BOOL;
0019
0020 END_VAR
```

```
0001 CAL     Netz_FF(SET := Start, RESET1 := Stop)
0002
0003 LD   Netz_FF.Q1
0004 ST   Netz
0005 ST   Impuls.IN
0006
0007 CAL     Impuls(PT := T_Anlauf)
0008
0009 LD   Impuls.Q
0010 ST   Imp
0011 NOT
0012 AND     Netz
0013 ST   Dreieck
0014
0015 LD   Netz
0016 AND     Imp
0017 ST   Stern
0018
```

10.4 Stern-Dreieck-Motor

Variante 2

SIMATIC ...-Station\CPU315-2 DP(1)\...\OB1-<offline> 18.07.2004 19:27:07

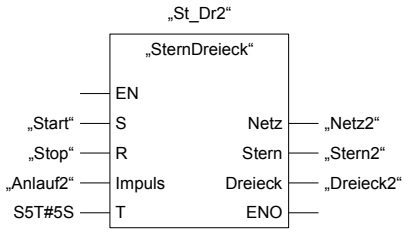
OB1-<offline>

"Cycle Execution"
Name: **Familie:**
Autor: **Version:** 0.1
Bausteinversion: 2
Zeitstempel Code: 02.12.2003 22:39:32
Interface: 15.02.1996 16:51:12
Längen (Baustein / Code / Daten): 00418 00302 00026

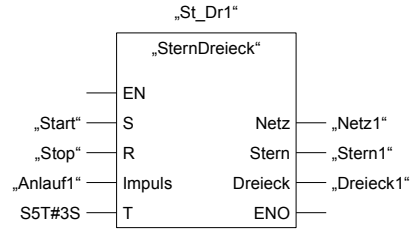
Adresse	Deklaration	Name	Typ	Anfangswert	Kommentar
0.0	temp	OB1_EV_CLASS	BYTE		Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
1.0	temp	OB1_SCAN_1	BYTE		1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of OB 1)
2.0	temp	OB1_PRIORITY	BYTE		Priority of OB Execution
3.0	temp	OB1_OB_NUMBR	BYTE		1 (Organization block 1, OB1)
4.0	temp	OB1_RESERVED_1	BYTE		Reserved for system
5.0	temp	OB1_RESERVED_2	BYTE		Reserved for system
6.0	temp	OB1_PREV_CYCLE	INT		Cycle time of previous OB1 scan (milliseconds)
8.0	temp	OB1_MIN_CYCLE	INT		Minimum cycle time of OB1 (milliseconds)
10.0	temp	OB1_MAX_CYCLE	INT		Maximum cycle time of OB1 (milliseconds)
12.0	temp	OB1_DATE_TIME	DATE_AND_TIME		Date and time OB1 started

Baustein: OB1 "Main Program Sweep (Cycle)"

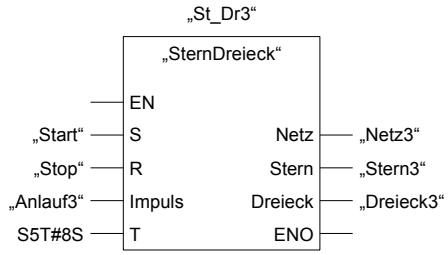
Netzwerk: 1 Motor 1



Netzwerk: 2 Motor 2



Netzwerk: 3 Motor 3



SIMATIC ...-Station\CPU315-2 DP(1)\...\FB1-<offline> 18.07.2004 19:27:53

FB1-<offline>

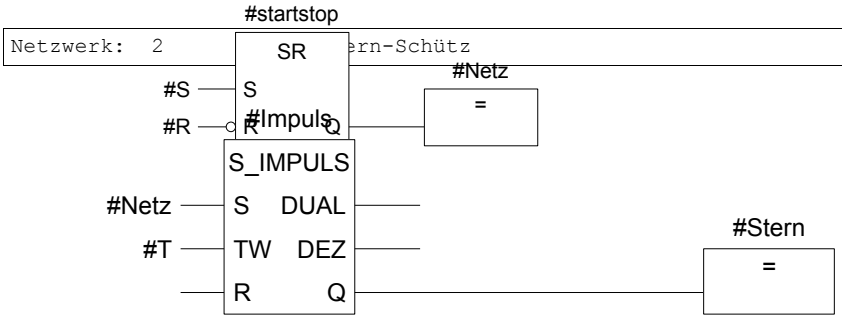
"SternDreieck"

Name: **Familie:**
Autor: **Version:** 0.1
Bausteinversion: 2
Zeitstempel Code: 02.12.2003 22:14:25
Interface: 02.12.2003 22:14:25
Längen (Baustein / Code / Daten): 00228 00116 00010

Adresse	Deklaration	Name	Typ	Anfangswert	Kommentar
0.0	in	S	BOOL	FALSE	
0.1	in	R	BOOL	FALSE	
2.0	in	Impuls	TIMER		
4.0	in	T	S5TIME	S5T#0MS	
6.0	out	Netz	BOOL	FALSE	
6.1	out	Stern	BOOL	FALSE	
6.2	out	Dreieck	BOOL	FALSE	
	in_out				
8.0	stat	startstop	BOOL	FALSE	
	temp				

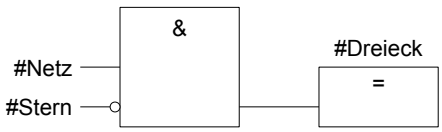
Baustein: FB1 Stern-Dreieck-Anlauf

Netzwerk: 1 Netz-Schütz



Netzwerk: 2 ern-Schütz

Netzwerk: 3 Dreieck-Schütz



SIMATIC ...-Station\CPU315-2 DP(1)\...\DB1-<offline> 18.07.2004 19:28:21

DB1-<offline>

"St_Dr1"

Name: **Familie:**
Autor: **Version:** 0.1
Bausteinversion: 2

Zeitstempel Code: 02.12.2003 22:20:19
Interface: 02.12.2003 22:14:25
Längen (Baustein / Code / Daten): 00114 00012 00000

Baustein: DB1

Adresse	Deklaration	Name	Typ	Anfangswert	Kommentar
0.0	in	S	BOOL	FALSE	
0.1	in	R	BOOL	FALSE	
2.0	in	Impuls	TIMER		
4.0	in	T	S5TIME	S5T#0MS	
6.0	out	Netz	BOOL	FALSE	
6.1	out	Stern	BOOL	FALSE	
6.2	out	Dreieck	BOOL	FALSE	
8.0	stat	startstop	BOOL	FALSE	

DB2-<offline>

"St_Dr2"

Name: **Familie:**
Autor: **Version:** 0.1
Bausteinversion: 2

Zeitstempel Code: 02.12.2003 22:25:26
Interface: 02.12.2003 22:14:25
Längen (Baustein / Code / Daten): 00114 00012 00000

Baustein: DB2

Adresse	Deklaration	Name	Typ	Anfangswert	Kommentar
0.0	in	S	BOOL	FALSE	
0.1	in	R	BOOL	FALSE	
2.0	in	Impuls	TIMER		
4.0	in	T	S5TIME	S5T#0MS	
6.0	out	Netz	BOOL	FALSE	
6.1	out	Stern	BOOL	FALSE	
6.2	out	Dreieck	BOOL	FALSE	
8.0	stat	startstop	BOOL	FALSE	

DB3-<offline>

"St_Dr3"

Name: **Familie:**
Autor: **Version:** 0.1
Bausteinversion: 2

Zeitstempel Code: 02.12.2003 22:33:16
Interface: 02.12.2003 22:14:25
Längen (Baustein / Code / Daten): 00114 00012 00000

Baustein: DB3

Adresse	Deklaration	Name	Typ	Anfangswert	Kommentar
0.0	in	S	BOOL	FALSE	
0.1	in	R	BOOL	FALSE	
2.0	in	Impuls	TIMER		
4.0	in	T	S5TIME	S5T#0MS	
6.0	out	Netz	BOOL	FALSE	
6.1	out	Stern	BOOL	FALSE	
6.2	out	Dreieck	BOOL	FALSE	
8.0	stat	startstop	BOOL	FALSE	

Symboltabellen-Eigenschaften

Name: Symbole
Kommentar:

Erstellt am: 11.11.2003 22:37:26
Zuletzt geändert am: 02.12.2003..23:33:16
Letztes Filterkriterium: Alle Symbole
Anzahl der Symbole: 19/ 19
Letzte Sortierung: Symbol aufsteigend

Symbol	Adresse	Datentyp	Kommentar
Anlauf1	T 0	TIMER	
Anlauf2	T 1	TIMER	
Anlauf3	T 2	TIMER	
Cycle Execution	OB 1	OB 1	
Dreieck1	A 0.2	BOOL	
Dreieck2	A 1.2	BOOL	
Dreieck3	A 2.2	BOOL	
Netz1	A 0.0	BOOL	
Netz2	A 1.0	BOOL	
Netz3	A 2.0	BOOL	
St_Dr1	DB 1	FB 1	
St_Dr2	DB 2	FB 1	
St_Dr3	DB 3	FB 1	
Start	E 0.0	BOOL	
Stern1	A 0.1	BOOL	
Stern2	A 1.1	BOOL	
Stern3	A 2.1	BOOL	
SternDreieck	FB 1	FB 1	
Stop	E 0.1	BOOL	