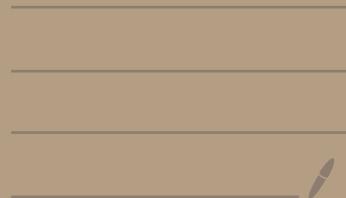


# Echtzeitsysteme

---

ss 20



# Echtzeitprogrammierung

## Anforderungen

### Korrektheit

- Ergebnis nur korrekt wenn logisch + zeitlich korrekt

### Rechtzeitigkeit

- Ausgabedaten müssen rechtzeitig berechnet werden und zur Verfügung stehen

### Zeitbedingungen

- genauer Zeitpunkt



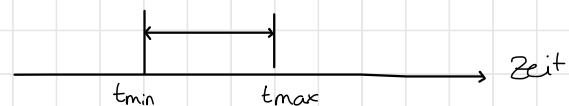
- spätester Zeitpunkt



- frühestes Zeitpunkt



- Zeitintervall



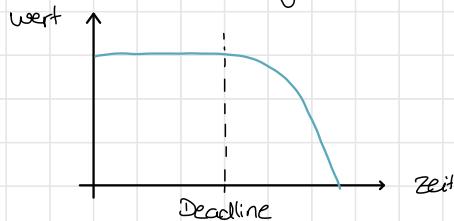
- periodisch

- aperiodisch

## Bewertung

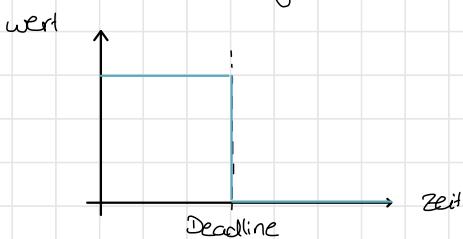
weiche Echtzeitbedingungen = soft deadline

- überschreitung führt nicht direkt zu fatalen Systemzuständen



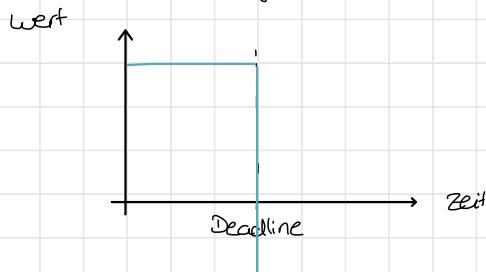
feste Echtzeitbedingungen = firm deadline

- überschreitung führt zu Abbruch der Aktion



harte Echtzeitbedingungen = hard deadline

- müssen eingehalten werden, sonst droht Schaden



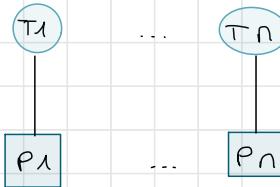
- wert > 0 : ausführen
- = 0 : abbrechen
- < 0 : Schaden

## Gleichzeitigkeit

- Richtigkeit muss für mehrere Aktionen gleichzeitig gewährleistet sein

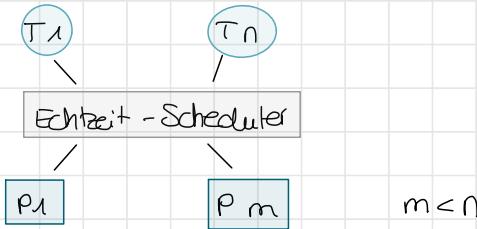
## Erfüllungsmöglichkeiten

- vollständige Parallelverarbeitung in Mehrprozessorsystem

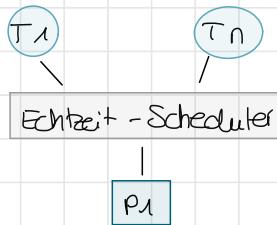


T = task  
P = Processor

- quasi-parallele Verarbeitung in Mehrprozessorsystem



- quasi-parallele Verarbeitung in Einprozessorsystem



## Verfügbarkeit

- Es müssen über längeren Zeitraum dauerhaft verfügbar sein
- keine Unterbrechung für Reorganisationsphasen

# Programmierverfahren

## Synchrone Programmierung

zeitgesteuerte Systeme

- zeitl. Verhalten periodischer Aktionen vor Ausführung geplant
  - ↳ Synchronisation in Zeitraum  $T$
  - ↳ Aktionen ausführen bei ganzzahligen Vielfachen von  $T$
- Zeitraum durch Zeitgeber
- Reihenfolge der Teilprogramme fest vorgegeben

### Zeiteinhaltung falls

1. Summe der Ausführungszeiten <  $T$
2. jede längere Periodenzzeit ist ganzzahliges Vielfaches der nächst kürzeren

- (+)
- festes, vorhersagbares Zeitverhalten
  - einfache Analyse + Tests des Systems
  - Rechtzeitigkeit + Gleichzeitigkeit garantierbar (gute Planung!)

- (-)
- geringe Flexibilität gegenüber Änderung der Aufgabe
  - keine Reaktion auf aperiodische Ereignisse
- ↑  
nur möglich mit periodischer Überprüfung

## Asynchrone Programmierung

- Echtzeitscheduling = Ablaufplan der Aktionen durch Organisationsprogramm  
zur Laufzeit

- Ereignisse überwachen
- Infos über Zeitbedingungen entgegennehmen
- Ablaufreihenfolge der Teilprogr. zur Bearbeitung eingetretener Aufgaben ermitteln
- entspr. Teilprogr. aktivieren

- flexibler Programmablauf
  - Reaktion auf periodische + aperiodische Ereignisse
  - Rechtzeitigkeit nicht immer im Voraus garantierbar
    - ↳ je niedriger die Prio desto größer die mögl. Zeitschwankungen
  - Analyse + Tests schwieriger
    - ↳ ggf. Feasibility Analysis möglich
- Prios vergeben

## Echtzeitbetriebssystem

### Aufgaben

- Taskverwaltung
  - Betriebsmittelverwaltung
  - Interprozesskommunikation
  - Synchronisation
  - Schutzmaßnahmen
  - Wahrung von Rechtzeitigkeit + Gleichzeitigkeit
  - Wahrung der Verfügbarkeit
- } ≈ Standard OS

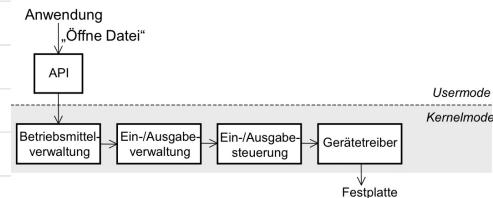
### Schichtenmodell

Anwendung



## Makrokern (monolithisch)

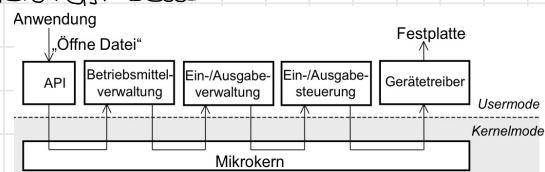
- Usermode: Anwendung
- Kernelmode: OS-Aufgaben



## Mikrokern

- erweiterter Usermode übernimmt Teile der OS-Aufgaben
- Kernelmode: Dinge die nicht unterschreibbar
  - Interprozesskomm.
  - Synchronisation
  - elementare Taskverwaltungsaufgaben

- ⊕
- gut anpassbar an Aufgabe
  - hohe Skalierung durch Erweiterungsmodul
  - einfache Portierbarkeit
  - preemptiver Kern
  - kurze kritische Bereiche



- ⊖ viele wechsel Usermode  $\leftrightarrow$  Kernelmode

## Taskverwaltung

Anwendung = 1...n Tasks

Task = 1...n Threads

### Task

= schwergewichtiger Prozess

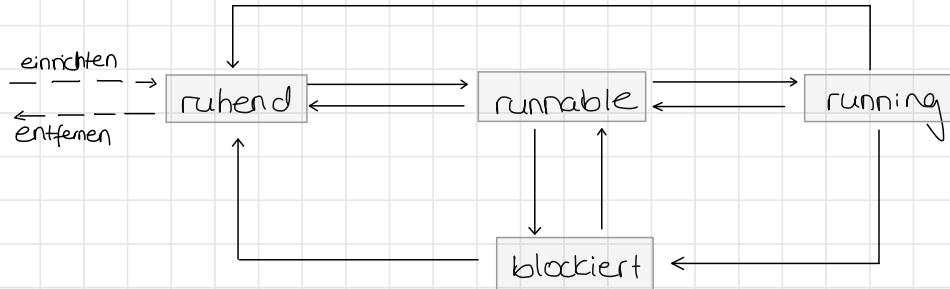
- eigene Variablen, Betriebsmittel, Adr.raum
- abgeschirmt von anderen Tasks
- Kommunikation nur über IPC
- Task-Switches langsam

### Thread

= leichtgewichtiger Prozess

- gemeinsame Variablen, Betriebsmittel, Adr.raum innerhalb einer Task
- wenig Schutz zw. Threads
- Kommunikation über globale Variablen
- Thread-Switches schnell

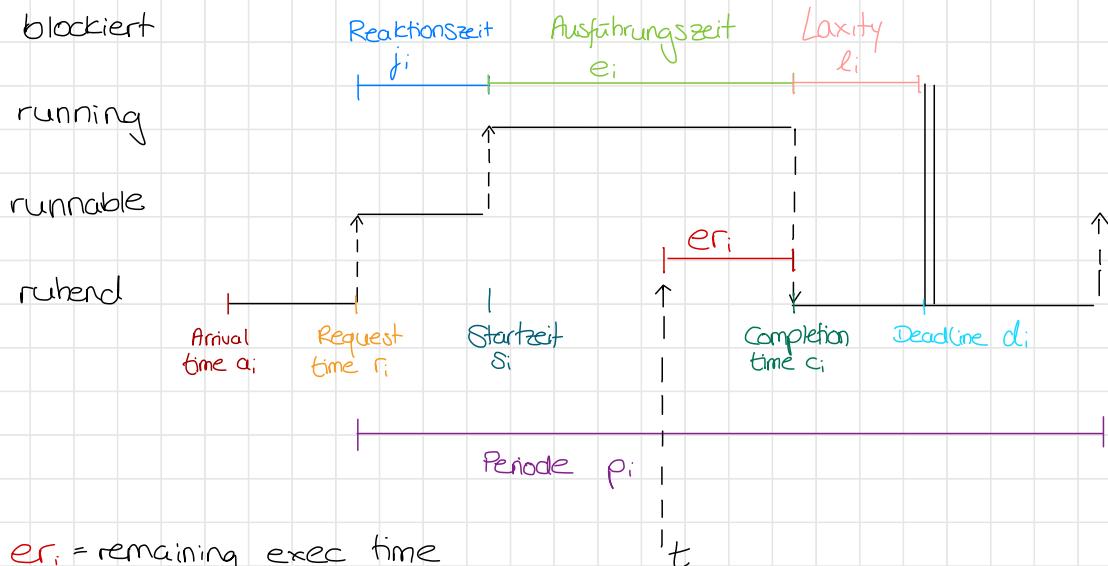
## Zustände



- ruhend = noch nicht ablaufbereit : Zeitbed./Voraussetzungen nicht erfüllt
- runnable = bereit, warte auf Zuteilung
- blockiert = warte auf Ereignis oder Freigabe eines Betriebsmittels

## Zeitparameter

- Annahmen:
- $e_i = \text{const.}$
  - context switches kosten keine Zeit
  - Deadline  $\geq$  Periode



$$l_i = \text{Laxity} = d_i - (t + er_i)$$

## Echtzeit Scheduling

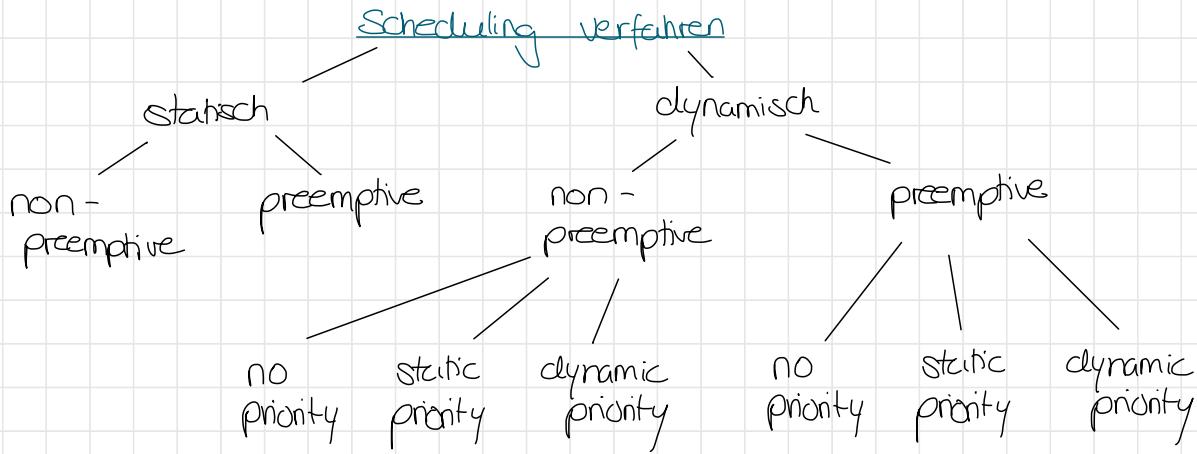
- optimal falls es Schedule findet sodass alle Zeitbedingungen eingehalten

## Feasibility Analysis

- Prozessorauslastung  $H = \sum_{i=1}^n \frac{e_i}{p_i}$

$H > 100\%$   $\Rightarrow$  kein Schedule

$H \leq 100\%$   $\Rightarrow$  Schedule existiert, ob gefunden abh. von Verfahren



## FIFO

- dyn., non-preemptive, no priority
- keine Zeitbed. eingehalten

## Fixed Priority

- dyn., static priority
- preemptive und non-preemptive möglich

$\rightarrow$  abh. von Prioritätszuweisung!

## Rate - Monotonic Scheduling

- preemptive
- für periodische Tasks

· Prio<sub>i</sub> ~  $\frac{1}{p_i}$  Prio von Task umgekehrt prop. zu Periodendauer

Annahmen:

- $p_i = \text{const}$
- $d_i \hat{=} p_i$
- $e_i = \text{const}$ , bekannt
- Tasks blockieren sich nicht gegenseitig

=> optimale Prio-Zuteilung für feste Prios

ABER: kein optimales Scheduling

→ findet nur Lösung falls  $H \leq H_{\max} = n \cdot (\sqrt{2} - 1)$   
 $n = \# \text{Tasks}$

## Earliest - Deadline - First

- dyn., dyn. priorities, preem / non-preem.
- Task am nächsten zur Deadline = höchste Prio

→ optimales Scheduling auf 1-Prozessorsys. (falls  $H \leq 100\%$ )

## Least - Laxity - First

- dyn., dyn. priorities, preem / non-preem.
- Task mit geringster Laxity = höchste Prio  
 $l_i = d_i - (t + e_i)$

→ optimales Scheduling auf 1-Prozessorsys. (falls  $H \leq 100\%$ )

≈ mehr Taskwechsel als EDF  
besseres Verhalten bei Überlast

## Time-Slice

- dyn., preemp., no priorities
  - jeder Task bekommt feste Zeitschreibe (individuell je Task)
  - Reihenfolge = Reihenfolge der runnable Tasks
- Nahezu optimal für feinkörnige Time Slices  
Periodendauer abh. von #Tasks

## Guaranteed Percentage Scheduling

- dyn., preemp., no priorities
- jeder Task bekommt innerhalb einer Periode festen %-Sitz der verfügbaren Prozessleistung
- optimal für Einprozessorsys., geeignet für Multithreading  
Periodendauer unabh. von #Tasks  
viele Taskwechsel

## Synchronisation

gemeinsame Betriebsmittel: Daten, Geräte, Programme

Mutex = mutual exclusion

- verhindert gleichzeitige Zugriffe
- Task belegt Mutex + gibt frei
- falls Mutex belegt: blockiere andere zugreifende Tasks

## Semaphore

- Zählervariable + 2 atomare Ops: Passieren, Verlassen

## Deadlock

- mehrere Tasks warten auf die Freigabe von Betriebsmitteln die sich gegenseitig blockieren

Lösungen: Abkeitsanalyse, Betriebsmittel in Reihenfolge zulegen, Timeout

Livelock = Starvation

- Task wird durch andere Tasks ständig an Ausführung gehindert

### Prioritätsinversion

- $P_3 > P_2 > P_1$
- R nur von einem nutzbar
- $P_1$  belegt R,  $P_3$  will R\* aber kann nicht da Mutex auf R  
   $\rightarrow P_1$  hat effektiv höhere Prio als  $P_3$   $\rightarrow P_1 > P_3$
- $P_2$  verdrängt jetzt  $P_1$  vom Prozessor  
   $\rightarrow P_1$  hält R aber kann nicht beenden  
   $\rightarrow P_3$  bekommt R immernoch nicht  $\rightarrow P_2 > P_3$

### => Lsg: Prioritätsvererbung

- hier erbitt  $P_1$  die Prio von  $P_3$   
 $\hookleftarrow P_2$  kann  $P_1$  nicht verdrängen  
 $\hookleftarrow P_1$  gibt R frei  
 $P_3$  bekommt R  
 $P_1$  erhält wieder ursprüngliche Prio

### Priority Ceiling

- gg. Deadlocks + Prioritätsinversion
- Prioschranke von R = max. Prio eines Prozesses der R will
- Prioschranke von System = max. Prioschranke aller RS  
   $\rightarrow$  von  $R^*$
- $P_1$  möchte  $R_1$  nutzen:
  - $R_1$  verfügbar?
    - $\rightarrow$  nein:
      - $P_1$  blockiert an  $R_1$
      - $P_i$  der  $R_1$  hat erbitt Prio von  $P_1$ ,  $P_i$  bekommt alte Prio zurück wenn alle Ressourcen mit  $Prio \geq P_1$  freigegeben
- ja: Prüfe Prioschranke vom System
  - $P_1 > P(S)$ :  $P_1$  bekommt  $R_1$
  - $P_1 \leq P(S)$ ?  $P_1$  bekommt  $R_1$  nur wenn  $P_1$  auch  $R^*$  hat

## Kommunikation

2 Möglichkeiten

- shared memory (schneller)
- Nachrichten

## prioritäts-basierte Kommunikation

- wählt End-to-End Priorities
- hochprio Nachrichten überholen lowprios
- keine Blockierung

## Koordination

- synchron vs. asynchron
- blockierend vs. nichtblockierend

## Puffer bei async. Komm.

- Briefkasten = individuelle Queue zw. Sender + Empfänger vs.
- Port = Briefkasten mit mehreren Sendern pro Empfänger

## Speicherverwaltung

Speicherabbildungsfunktion  $N$ : logische Adr.  $\rightarrow$  physikalische Adr.

- Speicherzuteilung: statisch vs. dynamisch, verdrängend vs. nicht-v.
- Adressierung: reell vs. virtuell
- Adressbildung: linear vs. streuend

## Linear, statisch, reell, nicht-verdrängend

- (+)
- Tasks im Speicher verschiebbar
  - Tasks voneinander geschützt
  - Zeitverhalten gut vorhersagbar
- (-)
- statisches Speicherschema, nicht online änderbar
  - für jede Task max. Speicher
  - Variierende Task-Anzahl: für alle Speicher reservieren, kein Teilen

statisch = jeder hat festen Platz zugewiesen

## Linear, dynamisch, reell, nicht-verdrängend

- (+) Tasks können sich physischen Speicher teilen
- konst. Zugriffszeit sofern keine Verdrängung zur Laufzeit
- (-) löschriger Speicher → Speicherbereinigung nötig

## Speicherbereinigung und Echtzeit?

- Laufzeit nicht deterministisch
- zB als Idle-Task machen (wenn sonst nichts zu tun)

## Linear, virtuell, Verdrängung

### Verdrängungsstrategie wen auslagern?

- FIFO
- LRU
- LFU
- LRD
- = least reference density



### Zuteilungsstrategie wohin einlagern?

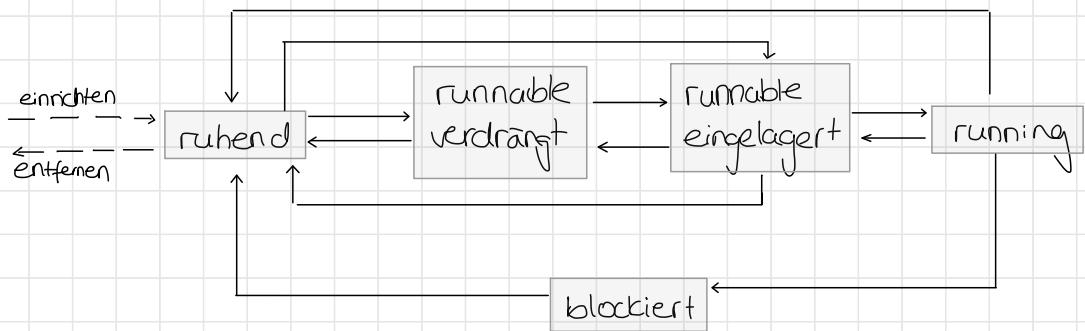
- First Fit
- Best Fit
- Worst Fit

### Nachschubstrategie wann einlagern?

- verlangend
- vorausschauend

Echtzeit OS: Verteilen von Tasks, keine Verdrängung

Linear = Nachbarn im virtuellen Adrraum sind Nachbar im phys.



## streuend

- logischer Adrraum = Seiten  
= Kacheln  $\rightarrow$  gleiche Größe
- phys.

$\rightarrow$  Zuordnung unabh. von Reihenfolge der Seiten im logischen Addr.



- einfache Zuweisung, keine Zuteilungsstrategie nötig
- keine Speicherbereinigung
- nur Verdrängungs-, Nachschubstrategie nötig
- Taskgröße dyn. änderbar



- letzte Seite pro Task ggf. nicht voll
- höherer Seitenverwaltungsaufwand
- Kleine Seiten = viele Datentransfers

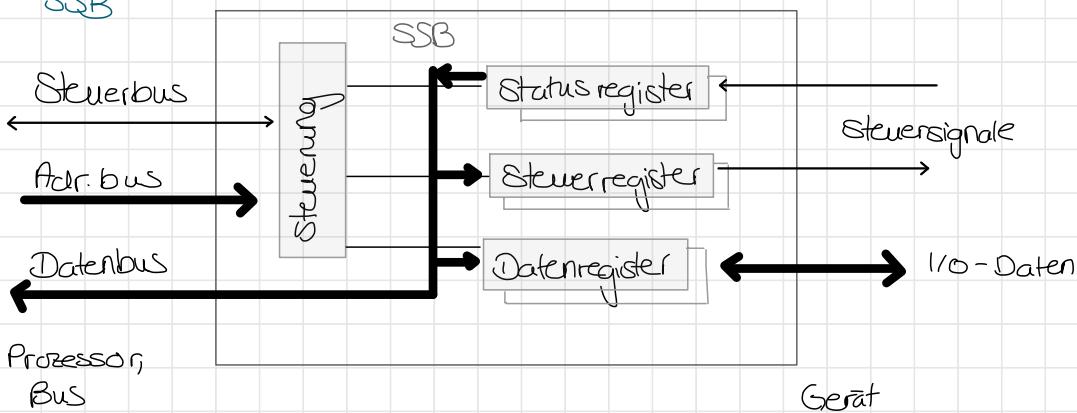
## I/O - Verwaltung

- Zuteilung, Freigabe von Geräten
- Benutzen von Geräten
  - $\rightsquigarrow$  Geschw.keit + Datenformat abh. vom Gerät
  - $\rightsquigarrow$  OS muss geräteunabh. Schnittstelle anbieten

## Synchronisationsmechanismen

komm. zw. Task, Peripherie über SSB = Schnittstellenbaustein

### SSB



## Polling

- regelmäßige Anfrage an Gerät ob bereit

(+)

- einfach
- hohe Übertragungsleistung bei einem Gerät

(-)

- Prozessor ständig aktiv
- keine Nebenaktivitäten
- verlängsamte Reaktionszeit bei mehreren Geräten

## Busy Waiting

- regelmäßige Anfrage an Gerät
- während warten andere Aufgaben machen

(+)

- Nebenaktivitäten möglich

(-)

- verlängsamte Reaktionszeit
- Nebenaktivität muss aufgeteilt werden in kleine Tasks

## Interrupts

- Gerät sendet Interrupt falls bereit
- Implementierung:
  - a) Unterbrechung der Taskverarbeitung
  - b) Integration → Interrupt als neuen Task zu Task-Scheduling hinzufügen
- (+) beliebige Nebenaktivitäten
- (-) verringerte Übertragungsraten

## Handshakes

- zusätzl. zu Polling / Interrupt
- falls Gerät Daten schneller liefert als Task entgegen nehmen kann  
↳ Gerät wartet auf Bestätigung von Task

# Rechnerarchitekturen

Mikrorechner System > Mikrorechner > Mikroprozessor

## Mikrorechnersystem

= Mikrorechner + Peripherie

## Mikrorechner

= Mikroprozessor + HS + I/O -Schnittstellen + Bus

## Mikroprozessor

= Rechenwerk + Steuerwerk + Anbindung an Systemumgebung

→ verfünde Architektur die gestellte Aufgabe mit geringsten Kosten umsetzt

## Interrupts

Arten:

- Exception = Fehlersituation
- SW - Interrupt = Unterbrechungswunsch des laufenden Programms
- HW - Interrupt = Unterbrechungswunsch externer Komponenten (Timer, ...)

## Interrupt Service Routine

- Behandelt Interrupts
- zuständige Routine mit Interrupt Vektor Tabelle bestimmen
  - ↳ Interrupt Quelle liefert Interrupt - Vektornummer
  - ↳ Tabelle enthält Startadr. der entspr. Routine

## Interrupt Maskenregister

- Prioritätsebenen im Mikroprozessor
- jedes laufende Programm erhält Prioebene im Maskenregister
- nur Interrupts mit höherer Prio können Progr. unterbrechen

# Prioritätensteuerung

## intern

- Unterbrechungsbus mit n Leitungen  
~ codiert  $2^n$  Prioritätsebenen

## extern

- Dezentral: Daisy Chain

- Interruptquellen gemäß prio in Serie schalten
- $P(1) > P(2) > \dots$
- falls  $P(i)$  keinen Interrupt hat: durchschalten an  $P(i+1)$
- natürliche Wahrung der Prios durch Reihenfolge

- Zentral: Interrupt Controller

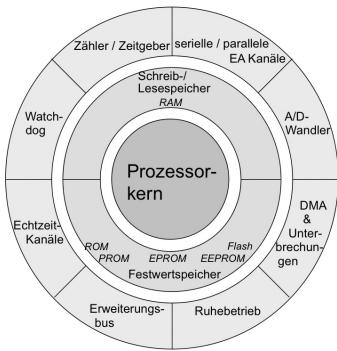
- Interruptquellen an Intur. contr. angeschlossen

- (-) · starre Prios  $\Rightarrow$  Echtzeitproblem: zeitl. Vorhersage nur für höchste prioebene

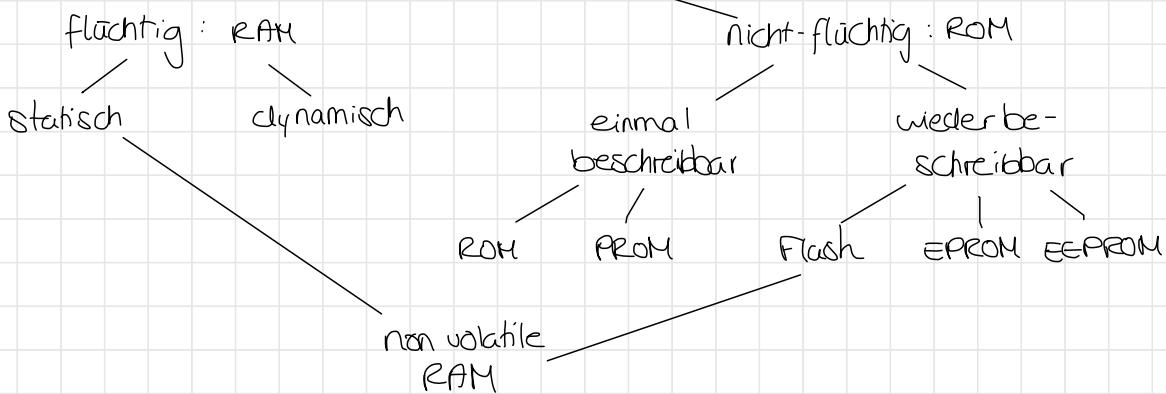
# Mikrocontroller

- ein-Chip-Mikrorechner
- speziell zugeschnittene Peripherie für Steuerungsaufgaben

# Schalenmodell



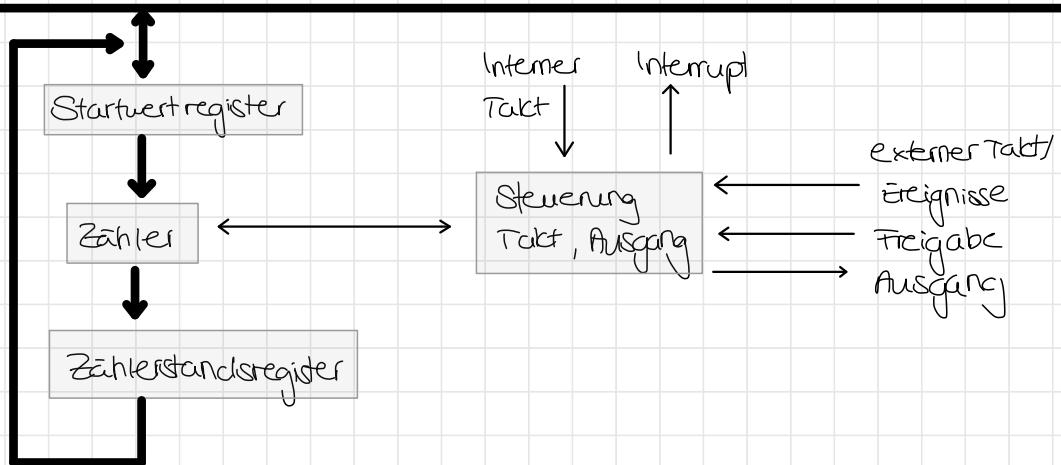
## Speicher



## non volatiler RAM

- Betrieb: RAM Speichermatrix benutzt
- Spannungsabfall: Kopiere RAM  $\rightarrow$  Flash RAM
- Spannungsanstieg: Kopiere Flash RAM  $\rightarrow$  RAM

## Counter/Timer



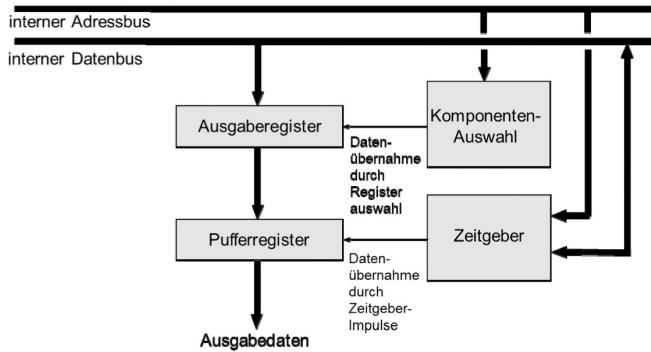
## Watchdog

Zähler der bei Ablauf Reset auslöst

## Jitter

- Speicherhierarchie: Zugriffszeit variiert je nach Zugriffsebene  
→ zeitl. Fluktuation der Ausführungszeit

Lösung: Echtzeit-Ausgabeeinheit

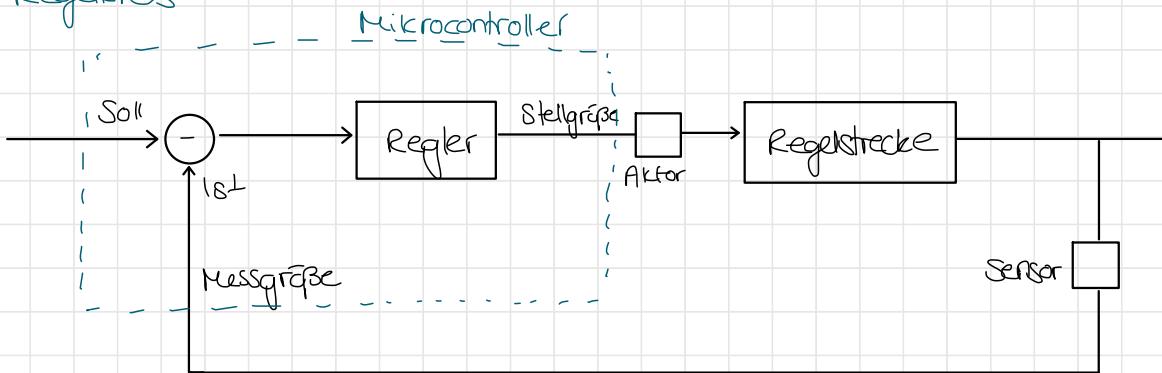


- Zwischenpuffern der Ausgabe

↳ Ausgabe triggern durch Zeitgeber

- Regelmäßigkeit durch zusätzliche Verzögerung

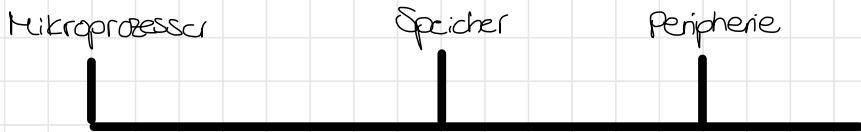
Embedded ES  
Regelkreis



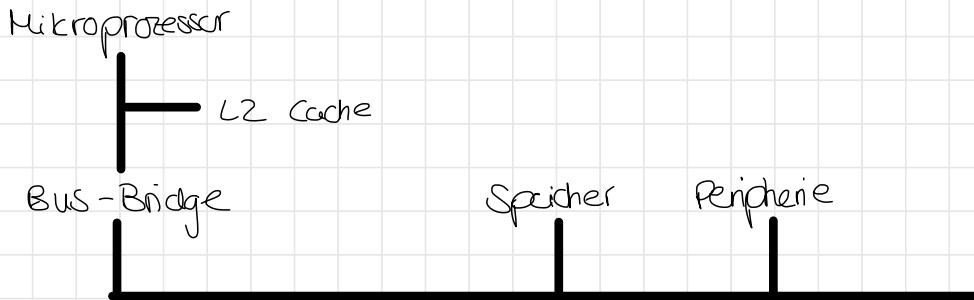
## Systembus

prozessorabhängig

= alle hängen am gleichen Bus



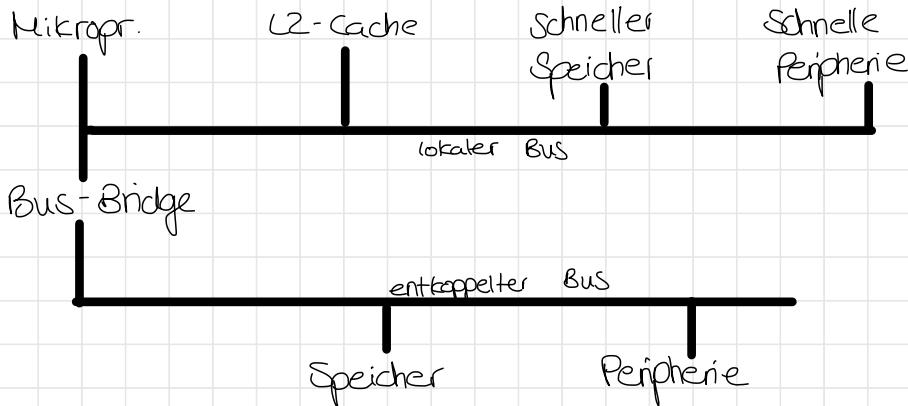
prozessorunabh.



(-) erhöhter HW-Aufwand

gepufferter, entkoppelter Bus

- zentraler Bus: langsame Komponenten blockieren Bus  
→ Priority inversion, Starvation, ...



## Synchroner Bus

Datenübernahme nur an Taktflanken

## Synchron + Wurkezyklen

- zusätzliche WAIT-Leitung
- erlaubt Anschluss von langsamen Komponenten an Bus

## asynchroner Bus

- 2 zus. Signale
  - Address Strobe:  $AS=0$  heißt Adr. gültig
  - Data Acknowledge:  $DTACK=0$  heißt Daten liegen auf Bus

## Buszuweisung

- Masterzugriff: aktiver Zugriff (Buszyklen auslösen, Kompon. adressieren, ...)
- Slavezugriff: passiver Zugriff (Datentransfer auf Anforderung, ...)

→ mehrere Master: Buszuweisung nötig

## Arbiter

- externer Schiedsrichter der Buszuweisung überwacht
- bus request vom Master
- bus grant vom Arbiter

## Zentral

- ein externer Arbiter für alle Komponenten
- z.B. nach Daisy-Chain Prinzip

- |   |                                   |   |                        |
|---|-----------------------------------|---|------------------------|
| + | einfacher Systemaufbau            | - | hoher Leistungsaufwand |
|   | beliebige, schnelle Priorisierung |   |                        |

## dezentral

- je Master ein Arbiter, Arbiter verbunden über z.B. Daisy-Chain

- |   |                      |   |                      |
|---|----------------------|---|----------------------|
| + | geringe Leitungszahl | - | starre Priorisierung |
|---|----------------------|---|----------------------|

## Identifikationsbus

- jeder Master bekommt prio  
→ bei > 1 Requests sucht ID-Bus höchsten Prio - Master

## Echtzeitbewertung

zeitl. Vorhersage wichtig!

- einfaches System, synchron, kein warten  
→ deterministisch ✓
- - " - , mit warten  
→ determ. falls Wartezyklen bekannt ✓  
→ variable Wartezyklen: Oberschranke + ggf. Abbruch
- mehrere Busmaster  
→ Echtzeitverhalten abh. vom Konfliktfall  
→ zur Wahrung der ES nötig:
  - Prioritäten
  - Preemption
  - Blocktransfers müssen unterbrechbar sein
  - Busmonitor

## PCI - Bus

- synchron
- Multiplex - Bus
- Burst - Transfers erlaubt
- Fehlererkennung
- Multi - Master fähig
- Echtzeitfähig
- Bridge - Konzept
- zentraler Arbiter, Prio definierbar vom Anwender

- Transaktion = 1 Adrphase + ≥1 Datenphase
- Agent = Busteilnehmer
- Initiator = Master
- Target = Slave

Transferzeiten  $n = \# \text{ Worte}$

- Standard Write =  $n \cdot (T_{\text{Adr}} + T_{\text{Datentransfer}})$
- Standard Read =  $n \cdot (T_A + T_{\text{wait}} + T_D)$   $T_{\text{wait}} = 1$  bei PCI
- Burst Write =  $T_A + n \cdot T_D$
- Burst Read =  $T_A + T_{\text{wait}} + n \cdot T_D$

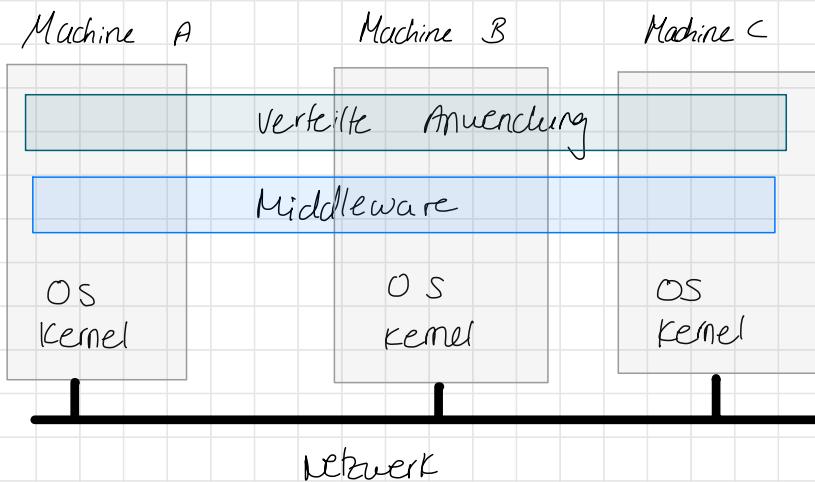
Übertragungsrate

$$= \frac{\# \text{ Bytes}}{T_C \cdot \# \text{ Zyklen}}$$

# Echtzeitmiddleware

## Middleware

- Vermittler zw. Anwendungen
- verdeckt Komplexität + Infrastruktur der Anwendungen
- zur Kommunikation zw. Prozessen



## Aufgaben

- Identifikation verteilter Objekte
- Lokalisierung der Objekte
- Interaktion
- Erzeugung + Vernichtung
- Fehlerbehandlung

## Vorteile

- Anwendung muss sich nicht um Verteilung kümmern
  - ↳ verteilte, lokale Anwendungen können gleich entwickelt werden
  - ↳ Änderung der Verteilung unabh. von Anwendung
- standardisierte SW übernimmt Aufgaben der Verteilung
- Portierbarkeit der Anwendung steigt
- bessere Testbarkeit + Wartbarkeit
- ~> geringere Entwicklungszeit + -Kosten

## Nachteile

- Speicher-, Verarbeitungsverhead
- Standard-Middleware unbrauchbar für ES  
→ brauche extra ES-Middleware

## Anforderungen ES

- Wahrung E2E Prioritäten (→ Priorisiere Kommunikation!)
- Zeitbedingungen, -schranken definieren
- RT fähige Funktionsaufrufe, Ereignisbehandlung
- ES Scheduling

## Beispiel: DDS



## • 2 APIs:

- Data-Centric Publish-Subscribe: für lower layers  
↳ tausche Daten mit anderen aus
- Data Local Reconstruction Layer: für upper layers  
↳ baue lokalen Cache auf sodass Anwendung denkt die Daten wären lokal gespeichert

• definiert QoS

# Echtzeitkommunikation

## ISO/OSI Modell:

- 7 Application Layer
- 6 Presentation
- 5 Session
- 4 Transport
- 3 Network
- 2 Data Link
- 1 Physical
- - - Übertragungsmedium

} In ES: brauchen nur 1, 2, 7

## ES Netztopologien

- Stern
- Ring
- Bus
- Baum
- Linie

## Zugriffsverfahren

### Polling

- Steuerung durch Arbitr
- fragt Teilnehmer zyklisch ab

### CSMA/CD

unbrauchbar ab ~40% Auslastung

- alle schreiben einfach
- alle hören Bus ab

↪ Kollision? alle ziehen sich zurück

### CSMA/CA

- Kollisionen vermeiden

## Token-Passing

- Token zirkuliert
- wer Token hat darf zugreifen für gewisse Zeit

## TDMA

- Zeitscheiben
- es muss gelten  $\sum_j t_j \leq T = \text{zykluszeit}$

## Signalkodierung

NRZ = no return to zero

- 1  $\hat{=}$  hohe Spannung
- 0  $\hat{=}$  niedr. Sp.

RZ = return to zero

- 1. Takthälfte = NRZ
- 2. Takthälfte immer niedr. Spannung

## Manchester

- NRZ xor Takt
- 1  $\hat{=}$  wechselt niedr.  $\rightarrow$  hoch } ab Taktmitte
- 0  $\hat{=}$  wechselt hoch  $\rightarrow$  niedr. }
- ⊖ geringere Übertragungsbandbreite
- ⊕ enthält Takt im Signal  $\Rightarrow$  synchr. Empfänger

## Ethernet

- Bus / Linie / Baum
- CSMA / CD
- Manchester Code
- Ethernet Paket: Präambel mit 7 Bytes (je 10101010)

## TCP / UDP

TCP: mit Verbindungsaufbau, Quittierung

UDP: ohne Quittierung  $\rightarrow$  besser für ES

## Feldbusse

-> Komm. zw. Steuerung, Geräten

Anforderungen:

- ES fähig
- Interoperabilität von Geräten

## Profinet

= Linie

- NRZ od. Manchester
- 4 Bit Hamming - Distanz => 3 Bit Fehlerdet. (1 Bit Korrektur)
- > Monomaster: Telegram mit Function Code an Slaves
- > Multimaster: Token - Passing  
↳ zykl. Polling der Slaves für Alarm, ...

- 4 Telegramtypen:
- 1 Komponenten finden
  - 2 Daten senden variable Datenlänge
  - 3 fixe
  - 4 Tokenpassing

## CAN-Bus

- = Linie
- NRZ
- 6 Bit Hamming - Distanz
- gleichberechtigte Teilnehmer → Multimaster
- Prio basierter Zugriff
- Telegramm hat ID → bestimmt Prio
- alle senden wenn sie wollen → Dominante ⚡  
↳ komp. die 1 senden wollte zieht sich zurück

## CAN-Open

- verfügbare Funktionalität als Objekte beschrieben  
zB PDO = Prozessdatenobjekte

## INTERBUS

- = aktiver Ring
- NRZ
- Hamming Dist. 4, Loopback wird
- Monomaster, festes Zeitraeste
- Summenrahmentelegramm: Teilnehmer entnimmt Daten, fügt Eingabedaten entspr. ein

## AS1-BUS

- = Linie, Baum (Stern)
- Monomaster + Polling
- 1 Paritätsbit → Wahl. gestörter Telegramme

## Sicherheitsbusse

- Sicherheit für Menschen

~ Übertragungsfehler:

- falsche Wahl von Telegrammen
- falsches Einfügen von Telegr.
- falsche Abfolge - " -
- Verzerrung - " -
- Verlust ...

## Safety Bus P

- ≈ CAN-Bus + Redundant ausgetragte Komponenten
- zusätzliche Sicherheitsprotokolle
  - Quittierung der Telegramme innerhalb gewisser Zeit
  - Telegram nur gültig wenn alle Kanäle korrekt Empfangen

## RT-Ethernet

Problem Ethernet: CSMA/CD nicht RT-fähig  
Ziel: weniger Jitter (Varianz der Latenz)  
geringere Latenz

Option 1: Minimierung der Kollisionsdomäne durch Ethernet-Switches

## Powerlink

- Zugriff gesteuert durch Managing Node (MN)
- Time Slice, Monomaster

- 1) MN sendet Start-of-cyclic = Beginn der deterministischen Phase
- 2) Teilnehmer nacheinander auffordern mit unicast PollRequest  
GTN senden Antwort als Broadcast PollResponse
- 3) alle TN durch: MN sendet End-of-cyclic  
= Beginn cyclerische Phase
- 4) nach Periodendauer sendet MN wieder SOC

## EtherCAT

- Monomaster sendet zykl. Telegramm → durchläuft alle TN
- Daten austausch nur über dieses Telegramm

## PROFINET

- zyklisch, 2 Kanäle

## SERCOS

- standard Ethernet-Frames
- zykl. + cycler. Komm. in Timeslice

# Analogschnittstellen

Ohmsches Gesetz:  $U = R \cdot I$

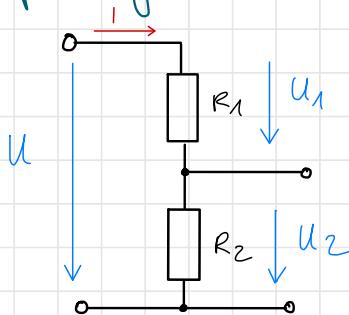
Parallele Widerst.:

$$\frac{1}{R_{\text{ges}}} = \frac{1}{R_1} + \frac{1}{R_2} + \dots + \frac{1}{R_n}$$

Serial

$$R_{\text{ges}} = R_1 + R_2 + \dots + R_n$$

Spannungsteiler



· Strom  $I$  durch beide Widerst. gleich

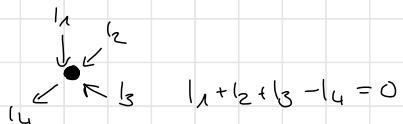
$$I_{\text{ges}} = \frac{U_{\text{ges}}}{R_1 + R_2}$$

$$U_2 = R_2 \cdot I_{\text{ges}} = \frac{R_2}{R_1 + R_2} \cdot U_{\text{ges}}$$

Kirchhoff'sche Regeln

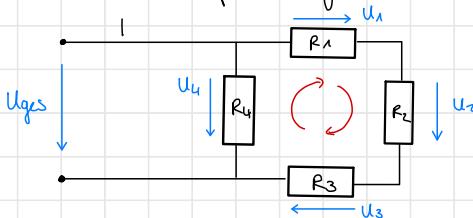
1. Knotenregel

· Summe aller Ströme in Knoten = 0



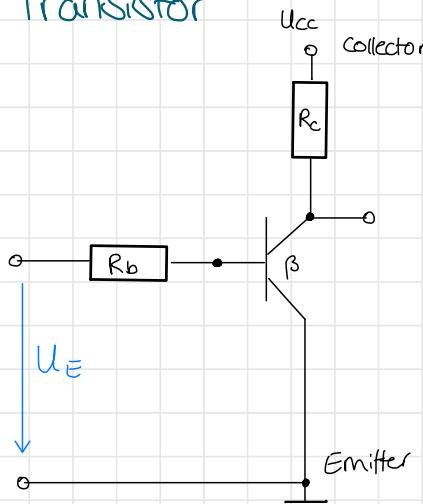
2. Maschenregel

· Summe aller Spannungen in Masche = 0



$$U_1 + U_2 + U_3 - U_4 = 0$$

# Transistor



$$I_C = I_B \cdot \beta$$

$\beta$  = Verstärkungsfaktor

$$I_E = I_B + I_C$$

$$\sim I_B = \frac{(U_E - 0,6V)}{R_b}$$

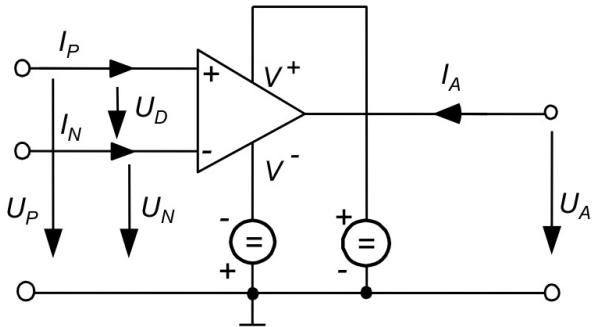
← liegt immer an,  
damit was passiert  
muss  $U_E > 0,6V$  sein

$$\Delta I_C = \Delta I_B \cdot \beta$$

$$\sim \Delta U_{CE} = - \frac{R_C}{R_B} \cdot \beta \cdot \Delta U_E$$

→ Transistor funktioniert als Schalter

## Operationsverstärker



$U_D$  = Eingang

$U_A$  = Ausgang

$\gamma_D$  = Verstärkung  $\approx 10^4$  bis  $10^7$

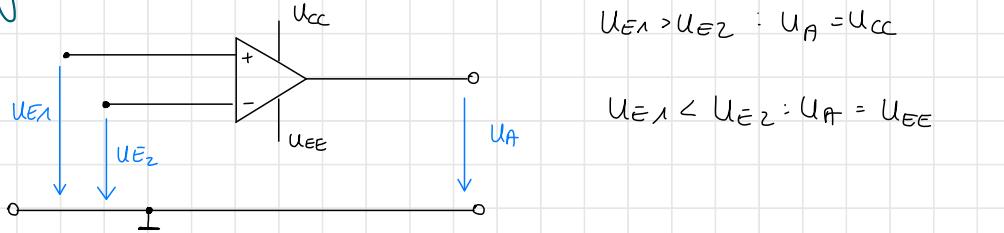
$$U_A = \gamma_D \cdot U_D$$

$$U_D = U_P - U_N$$

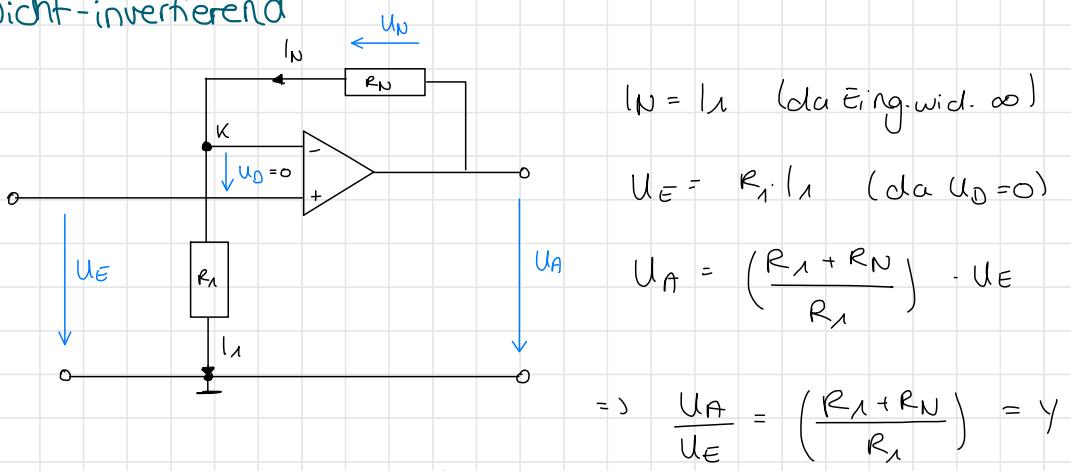
Annahmen: idealer OP

- Verstärkung  $\gamma = \infty$
- Eingangswiderst.  $= \infty$
- Ausgangswiderst.  $= 0$

Vergleicher

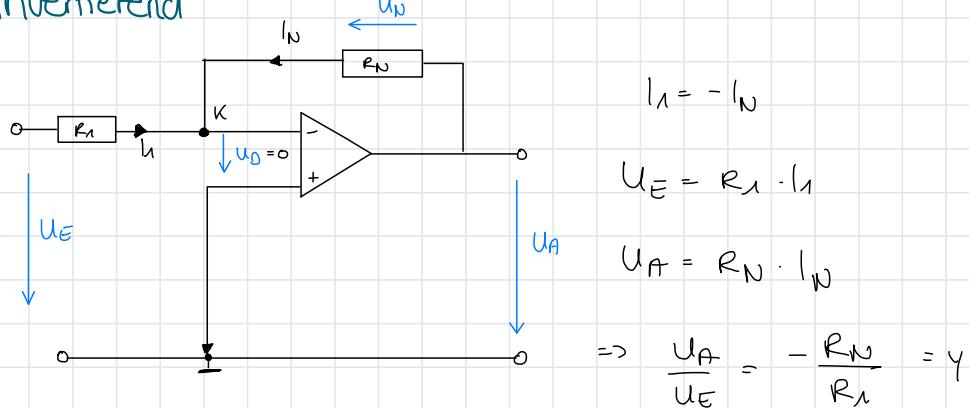


Nicht-invertierend

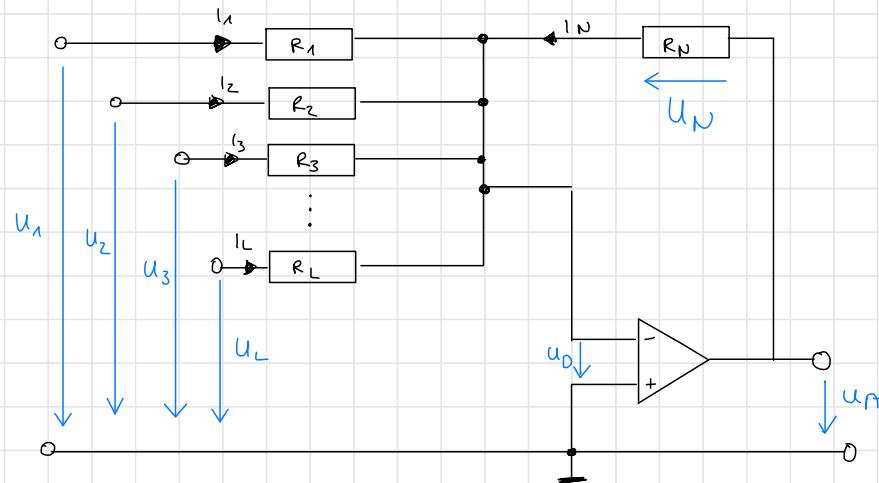


$\gamma \hat{=} \text{verst.-faktor der Schaltung, nicht OP!}$

## Invertierend



## Invertierender Addierer

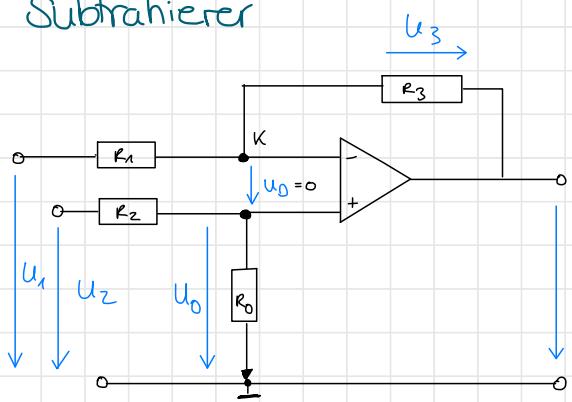


$$I_1 + I_2 + \dots + I_N = 0$$

$$\frac{U_1}{R_1} + \frac{U_2}{R_2} + \dots + \frac{U_L}{R_L} + \frac{U_A}{R_N} = 0$$

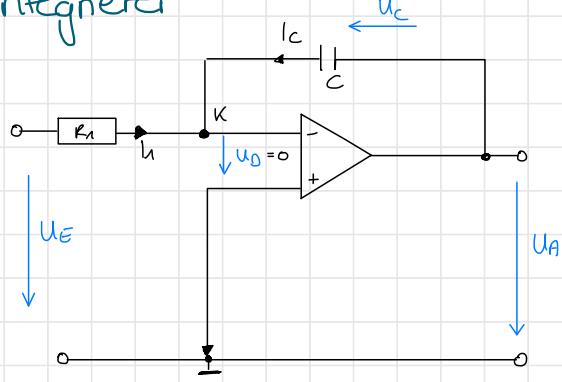
$$\Rightarrow U_A = - \left( \frac{U_1}{R_1} + \frac{U_2}{R_2} + \dots + \frac{U_L}{R_L} \right) \cdot R_N$$

## Subtrahierer



$$U_A = a \cdot (U_2 - U_1)$$

## Integrierer



$$U_A = U_C$$

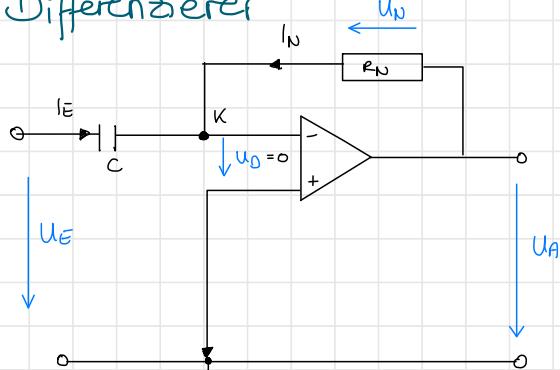
$$U_E = R_1 \cdot I_1, \quad I_1 + I_C = 0$$

$$U_C = \frac{Q}{C} = \frac{1}{C} \int I_C \cdot dt$$

$$\Rightarrow U_A = \frac{1}{C} \int I_C \cdot dt$$

$$= \frac{1}{R_1 C} \int U_E \cdot dt$$

## Differenzierer



$$I_E + I_N = 0$$

$$U_A = U_N$$

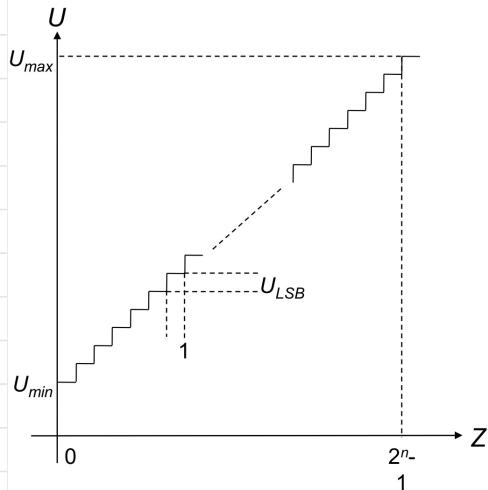
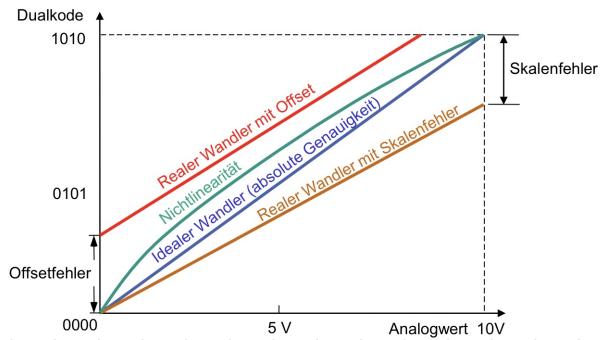
$$I_E = \frac{dQ_C}{dt} = C \cdot \frac{dU_E}{dt}$$

$$\Rightarrow U_A = -R_N \cdot I_N$$

$$= -R_N \cdot C \cdot \frac{dU_E}{dt}$$

$$= -\tau \cdot \frac{dU_E}{dt}$$

# AD/DA - Wandler



$U_{LSB} = \text{kleinste Schrittweite} = \text{Auflösung}$

$$= \frac{U_{max} - U_{min}}{2^n}$$

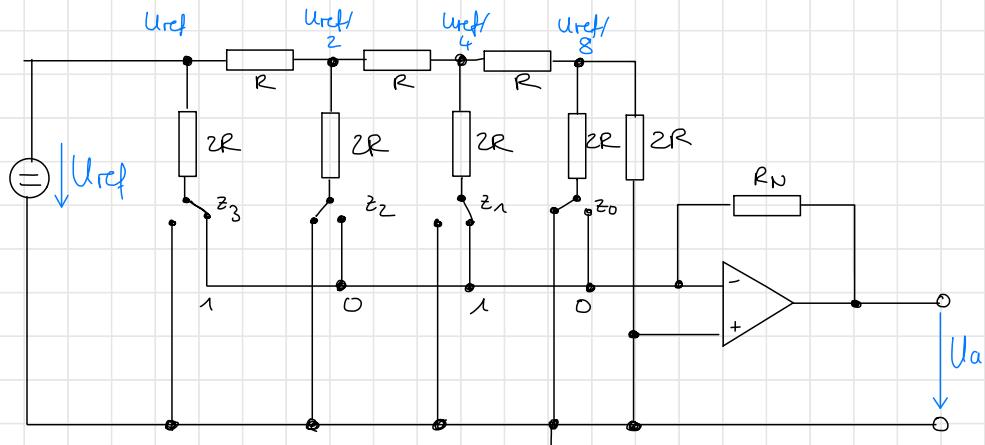
$U_{MSB} = \text{wert der obersten Stufe}$

$$= \frac{U_{max} - U_{min}}{2}$$

Dualzahl  $\rightarrow$  Spannung :  $U = z \cdot U_{LSB} + U_{min}$

Spannung  $\rightarrow$  Dualzahl :  $z = \frac{U - U_{min}}{U_{LSB}}$

## D/A Wandler mit Leiternetzwerk



→ Register enthält Wert  $1010_2 = 10_{10}$

$$\begin{aligned} \frac{U_A}{U_{\text{ref}}} &= -\frac{R_N}{2R} \left( z_3 + \frac{1}{2}z_2 + \frac{1}{4}z_1 + \frac{1}{8}z_0 \right) \\ &= -\frac{R_N}{16R} (8z_3 + 4z_2 + 2z_1 + z_0) \end{aligned}$$

## A/D - Wandler

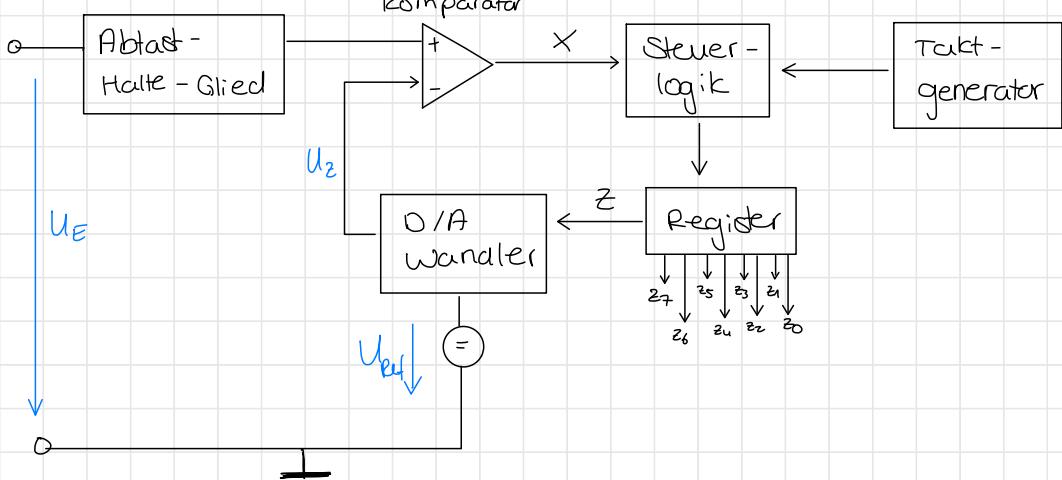
Arten:

- Wägverfahren
- Kompensationsverfahren
- Integrationsverfahren
- Parallelverfahren

• A/D - Wandler brauchen gewisse "Anlaufzeit"  
 ↳ Eingangsspannung muss konstant sein  
 => verwendet Abtast-, Halteglied

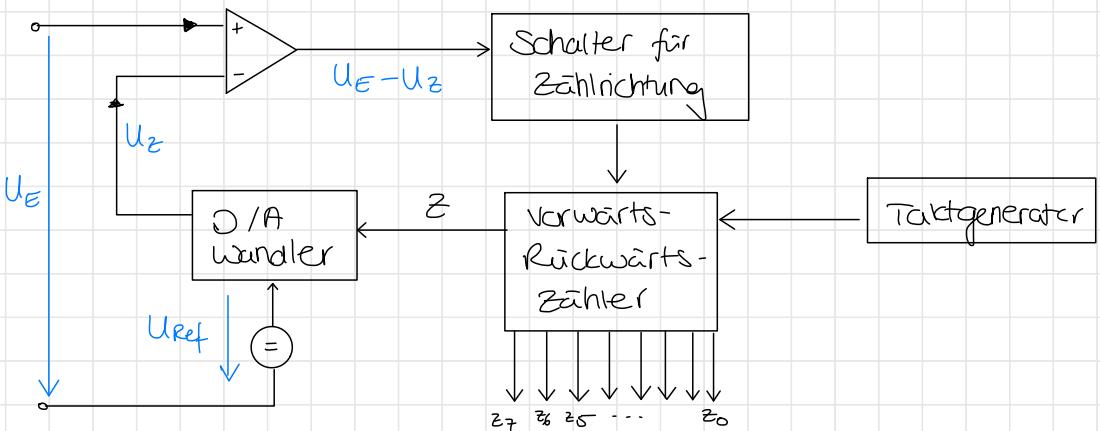
## Wägeverfahren

### SAR - A/D - Wandler



- Init: alle Bits  $z_7, \dots, z_0 = 0$
- Setze  $z_7 = 1$
- Vergleiche  $U_Z$  mit  $U_E$ 
  - $U_E \geq U_Z$  :  $X = 1$ ,  $z_7$  bleibt 1
  - $U_E < U_Z$  :  $X = 0$ ,  $z_7 = 0$
- vergleiche so  $z_6$  bis  $z_0$
- + hohe Umsetzrate: n Takte bei n Bit  $\rightarrow$  gut für RT

## Kompensationsverfahren



- Init: Zähler = 0
- Loop:  $U_E, U_Z$  vergleichen
  - $U_E > U_Z$ : Zähler ++
  - $U_E < U_Z$ : Zähler --

$\rightarrow$  max. Dauer:  $2^n - 1$  Takte

### Parallelverfahren : Flash-Wandler

- jeder mögliche Wert ( $2^n - 1$  Stück) wird parallel mit  $U_E$  verglichen
- falls  $U_E >$  eine der Spannungen: setze 1 im FF  
 $\hookrightarrow$  Kodierer wandelt gesetzte Bits in Dualzahl um

- ⊕ · schnellstes Verfahren  $\rightarrow$  Wandlung 1 Takt

### Delta-Sigma-Wandler

Oversampling des Eingangssignals  $\rightarrow$  Erzeuge Bitstream aus Analogsignal  
 $\rightarrow$  digitaler Tiefpassfilter  $\rightarrow$  Decimation

### Analogmessverfahren

n Sensoren, wie verschalten?

#### geringe Messrate:

- n Sensoren + Schalter  $\rightarrow$  Auswahl durch Steuerung
- geschaltete Sensoren  $\rightsquigarrow$  OP  $\rightsquigarrow$  Wandler  $\rightsquigarrow$  CPU

#### mittlere Messrate:

- je Sensor ein OP, hintendran Schalter
- geschaltete Werte  $\rightsquigarrow$  AD Wandler  $\rightsquigarrow$  CPU

#### hohe Messrate:

- je Sensor ein OP + AD Wandler, hintendran Schalter  $\rightsquigarrow$  CPU

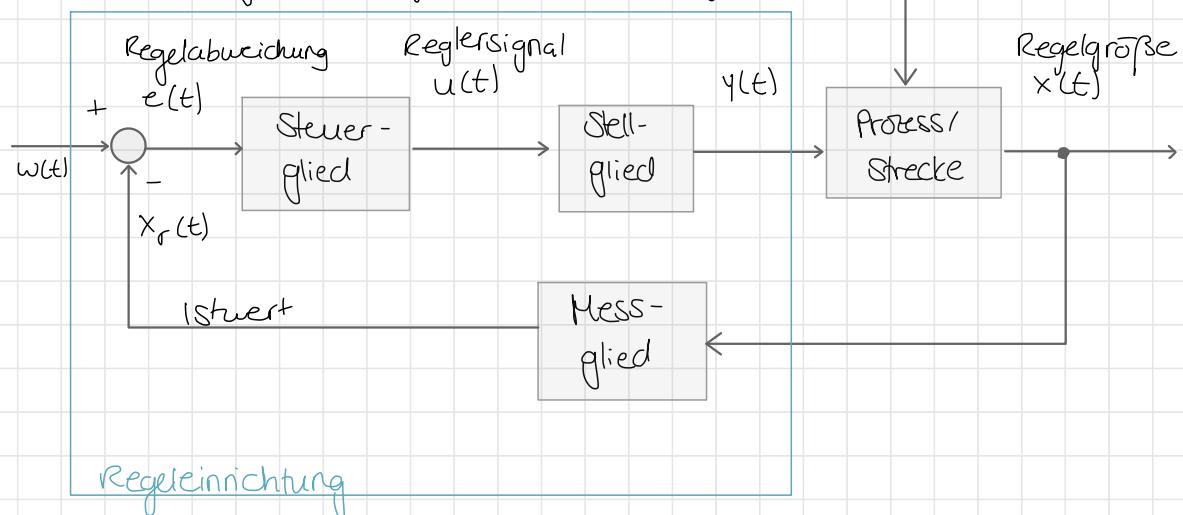
# Regelung

## Steuerung



## Regelung

Rückführung der gemessenen Steuergr.



## Reglerentwurf

1. Anforderungen definieren  $\rightarrow$  Gütekriterien Stabilität, Schnelligkeit, Robustheit, Genauigkeit
  2. Strecke modellieren (Strecke = was wird angepasst, zB Ruder/Fahrwerk)
  3. Reglertyp wählen / Regler entwerfen
  4. Reglerparams mit Simulation optimieren
  5. Regler realisieren, im realen Kontext prüfen, optimieren
- z-5 ggf. Wiederholen

# Kontinuierliche Systeme

## Modellierung

Zeitbereich

- DGL
- Sprungantwort
- Zustandsraumdarstellung

Bildbereich

- Übertragungsfkt
- Frequenzgang

DGL Eingangssignal  $w(t)$ , Ausgangssignal  $x(t)$

↪ lineare DGL

$$x^{(n)} + a_1 x^{(n-1)} + \dots + a_n x = b_0 w^{(m)} + b_1 w^{(m-1)} + \dots + b_m w$$

$\nearrow$  nte Ableitung

Anfangsbed. :  $x(t=0) = x_0$   
 $\dot{x}(t=0) = \dot{x}_0$   
 $\vdots$   
 $x^{(n)}(t=0) = x_0^{(n)}$

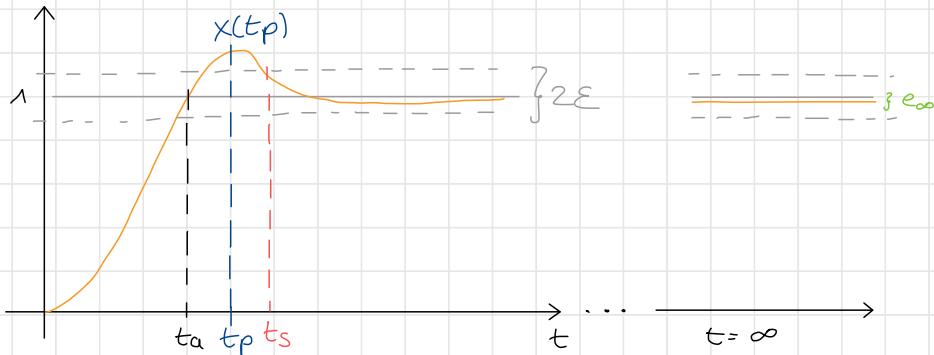
$m \leq n$ : Ausgang nur von vergangenen Eingängen abh.!

Lösung:

$$\begin{aligned} x_n(t) &= C_k e^{\lambda_k t} = C_k e^{(\delta_k + i\omega_k) t} \\ &= C_k e^{\delta_k t} (\cos \omega_k t + i \cdot \sin \omega_k t) \end{aligned}$$

## Springantwort

Modelliert wie System reagiert bei Änderung des Signals von 0 auf 1



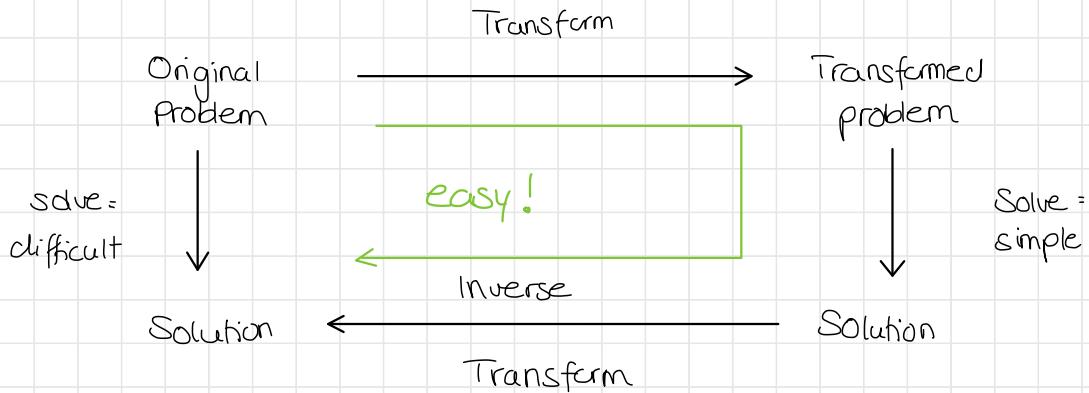
$t_a$  = Anregelzeit = Zeit bei der  $t_a$  das erste Mal = Sollwert

$x(t_p)$  = Überschwingweite = max. Wert der Sprungantwort

$t_s$  = Ausregelzeit = Zeit ab dem Wert  $\leq \varepsilon$  vom stationären Wert  $x(\infty)$  abweicht

$e_{\infty}$  =  $1 - x(\infty)$  = bleibende Regeldifferenz

## Laplace - Transformation



## Transformation

$$x(t) \xrightarrow{\text{---}} X(s) \quad \text{mit} \quad x(t < 0) = 0$$

$$X(s) = \mathcal{L}\{x(t)\} = \int_0^\infty x(t) e^{-st} dt$$

komplexe Frequenz  $s = \sigma + j\omega$

## Rücktransformation

$$x(t) = \mathcal{L}^{-1}\{X(s)\} = \frac{1}{2\pi j} \int e^{st} X(s) ds$$

## Regeln

Linearität  $c \cdot f(t) + d \cdot g(t) \xrightarrow{\text{---}} c \cdot F(s) + d \cdot G(s)$

Differenzierung  $f'(t) \xrightarrow{\text{---}} s \cdot F(s) - f(0)$

Integration  $\int_0^t f(\tau) d\tau \xrightarrow{\text{---}} \frac{F(s)}{s}$

## wichtige Funktionen

$$f(t)$$

$$> K$$

$$> t$$

$$> e^{-kt}$$

$$> t e^{-kt}$$

$$> \delta(t) / \delta(t - t_0)$$

$$> \sigma(t) \quad \text{Sprungfkt.}$$

$$F(s)$$

$$\frac{K}{s}, \quad s > 0$$

$$\frac{1}{s^2}$$

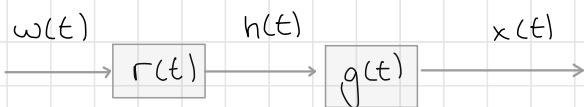
$$\frac{1}{s+k}, \quad s > -k$$

$$\frac{1}{(s+k)^2}, \quad s > -k$$

$$1 / e^{-t_0 s}$$

$$\frac{1}{s}$$

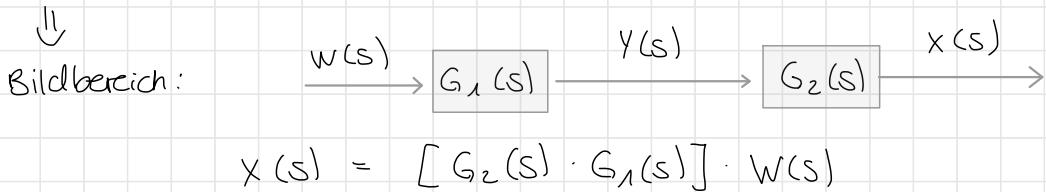
# Systembeschreibung



Zeitbereich:  $x(t) = \int_0^t h(t-\tau) \cdot w(\tau) d\tau$

mit  $h(t) = \int_0^t r(t-\tau) \cdot g(\tau) d\tau$

Faltungen  
kompliziert



## Frequenz

$s = \sigma + j\omega$  komplexe Frequenz

$\sigma > 0 \rightarrow$  System schwingt auf  
 $\sigma < 0 \rightarrow$  System schwingt ab

## Übertragungsfunktion

DGL  $x^{(n)} + \dots + a_n x = b_0 w^{(m)} + \dots + b_m w \quad (m \leq n)$

wird zu  $(s^n + a_1 s^{n-1} + \dots + a_n) X(s) = (b_0 s^m + \dots + b_m) w(s)$

↪ komplexe Übertragungsfkt

$$G(s) = \frac{X(s)}{W(s)} = \frac{b_0 s^m + b_1 s^{m-1} + \dots + b_m}{s^n + a_1 s^{n-1} + \dots + a_n} = b_0 \frac{\prod_{i=1}^m (s - \mu_i)}{\prod_{j=1}^n (s - \lambda_j)}$$

Ausgangsgr.

Eingangsgr. → forme so um dass im Nenner  $1 + s \cdot (\text{irgendwas})$  steht

## Umformungsregeln

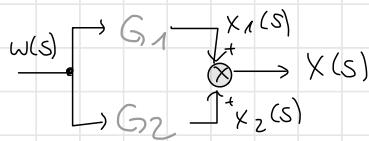
### Reihenschaltung

$$\omega(s) \rightarrow G_1 \xrightarrow{y(s)} G_2 \xrightarrow{x(s)} \Leftrightarrow \omega(s) \rightarrow G_1 \cdot G_2 \xrightarrow{x(s)}$$

$$X(s) = G_2(s) \cdot Y(s) \quad Y(s) = G_1(s) \cdot \omega(s)$$

$$\left. \begin{array}{l} X(s) = [G_2(s) \cdot G_1(s)] \cdot \omega(s) \\ \end{array} \right\}$$

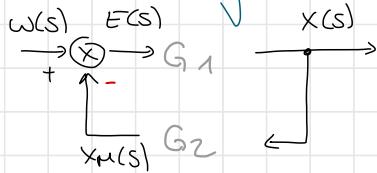
### Parallelschaltung



$$\omega(s) \rightarrow G_1 + G_2 \rightarrow X(s)$$

$$X(s) = [G_1(s) + G_2(s)] \cdot \omega(s)$$

### Rückkopplung $\hat{\wedge}$ Regelung



$$\begin{aligned} X &= G_1 \cdot E \\ E &= \omega - X_M \\ X_M &= G_2 \cdot X \end{aligned}$$

$$\Rightarrow X(s) = \left[ \frac{G_1(s)}{1 + G_1(s)G_2(s)} \right] \cdot \omega(s)$$

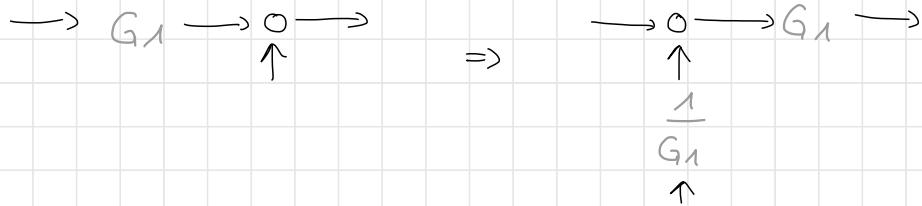
### Mittkopplung

- wird zu +

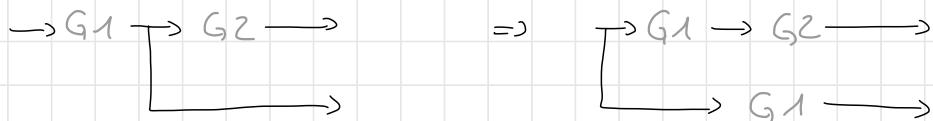
$$\Rightarrow X(s) = \left[ \frac{G_1(s)}{1 - G_1(s)G_2(s)} \right] \cdot \omega(s)$$

Verschiebung

Summationsstelle



Verschiebung Verzweigungsstelle



# Übertragungsglieder

Name

OGL

Übertragungs-  
fkt

Sprung-  
antwort

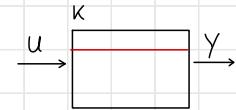
Symbol

P-Glied

Proportional

$$y = K \cdot u$$

$$K$$

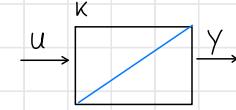
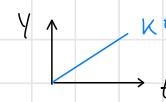


I-Glied

Integration

$$y = K \int_0^t u(r) dr$$

$$\frac{K}{S}$$

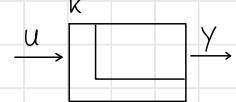
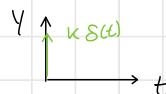


D-Glied

Differenzierung

$$y = K \cdot \dot{u}$$

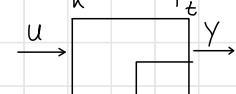
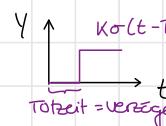
$$K \cdot S$$



TZ-Glied

Totzeit

$$y(t) = K \cdot u(t - T) e^{-\frac{t-T}{T}}$$

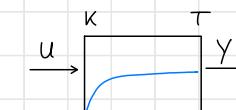
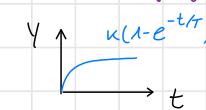


VZ-Glied

= PT1

$$T \dot{y} + y = Ku$$

$$\frac{K}{1+TS}$$

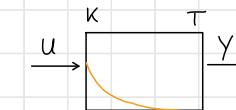


VID-Glied

$$T \dot{y} + y = K T u$$

$$\frac{KTS}{1+TS}$$

↑ verstärkung  
Zeitkonstante



## Charakterisierung dynamischer Systeme

- Sprungfkt siehe oben
- Impulsfkt
- Frequenzgang
- Ortskurve
- Bode-Diagramm
- Pol-Nullstellen-Diagramm

## Impulsfunktion

· in Realität nicht umsetzbar

Dirac-Impuls: unendlich starker, unendlich kurzer Impuls

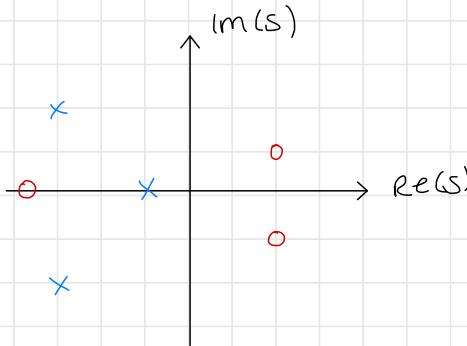
## Bode-Diagramm

Frequenzgang  $G(j\omega) = \text{Re}\{G(j\omega)\} + j \cdot \text{Im}\{G(j\omega)\}$   
aufteilen nach Betrag, Phase

- 2 Diagramme:
- $|G|_{\log}$  über  $\omega_{\log}$  → Amplitude
  - $\varphi$  Phase über  $\omega_{\log}$  → zeigt Phasenverschiebung

## Pol-Nullstellen-Diagramm

$$G(s) = \frac{X(s)}{W(s)} = \frac{b_0 s^m + b_1 s^{m-1} + \dots + b_m}{s^n + a_1 s^{n-1} + \dots + a_n} = b_0$$



$$\frac{\prod_{i=1}^m (s - \mu_i)}{\prod_{j=1}^n (s - \lambda_j)}$$

Nullstellen  
m

Pole

# Gütekriterien Regelungssysteme

Stabilität

Schnelligkeit

Genaugigkeit

## Stabilität

- abh. von Systemantwort

Stabiles System: BIBO = bounded input, bounded output

~ Anregung mit beschränkten Stellgrößen  
= Antwort mit beschränkten Ausgabesignalen

Instabiles System: Anregung mit beschränkten Stellgrößen

= Antwort mit unbeschränkten Ausgabesignalen

## Kriterien

- lineares System asymptotisch stabil wenn alle Pole der Übertragungsfkt einen negativen Realteil haben ↑  
= Nullstellen des Nenners

## Hurwitz - Kriterium

Stabil falls

1. Alle Koeffizienten  $a_i$  der DGL  $x^{(n)} + a_{n-1}x^{(n-1)} + \dots + a_0x = 0$  gilt:  $a_i > 0$

2. alle Hauptabschnittsdeterminanten der Hurwitz - Matrix  $> 0$

$$H = \begin{bmatrix} a_{n-1} & a_{n-3} & a_{n-5} & a_{n-7} & \cdots & 0 \\ 1 & a_{n-2} & a_{n-4} & a_{n-6} & \cdots & 0 \\ 0 & a_{n-1} & a_{n-3} & a_{n-5} & \cdots & 0 \\ 0 & 1 & a_{n-2} & a_{n-4} & \cdots & 0 \\ 0 & 0 & a_{n-1} & a_{n-3} & \cdots & 0 \\ 0 & 0 & 1 & a_{n-2} & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \ddots & a_0 \end{bmatrix}$$

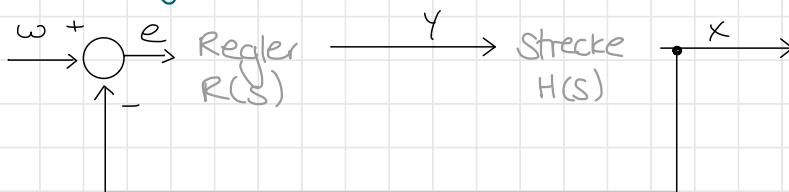
# Schnelligkeit, Genauigkeit

Betrachte Sprungantwort

$\rightarrow \tau_a$  für Schnelligkeit

Überschwingweite,  $e_{\infty}$  für Genauigkeit  $M_p = \frac{x(t_p)}{x(\infty)}$

## PID - Regler



$$G(s) = \frac{R(s) H(s)}{1 + R(s) H(s)}$$

## P - Glied

- je größer Regelabweichung desto größer Stellgröße
- Verstärkungsfaktor  $K_p$  stellt Regelgeschw. ein
- (-) hohes  $K_p$   $\rightarrow$  Instabilität (Schwingungen)
- (-) kann nicht vollst. auf Regeldifferenz = 0 regeln

## I - Glied

- solange Regelabweichung: ändere Stellgröße
- (+) Regeldiff wird ausgeregelt
- (-) führen leicht zu Instabilitäten

## D - Glied

- je stärkere Änderung der Regelabweichung, desto stärker ändere Stellsignal
- (+) verbessern Regelgeschw., dynamische Regelabw.
- (-) Neigung zu Schwingungen

$$P\text{-Regler} \quad R(s) = K_p$$

$$PI\text{-Regler} \quad R(s) = K_p \left( 1 + \frac{1}{T_N s} \right)$$

$$PD\text{-Regler} \quad R(s) = K_p (1 + T_V s)$$

$T_N$  = Nachstellzeit = Zeit in der P nachregelt

$T_V$  = Verhaltezeit = Zeit in der P den D Anteil sofort ausgibt

PID - Regler

$$R(s) = K_p \left( 1 + \frac{1}{T_N s} + T_V s \right)$$

-> als Kompensationsregler

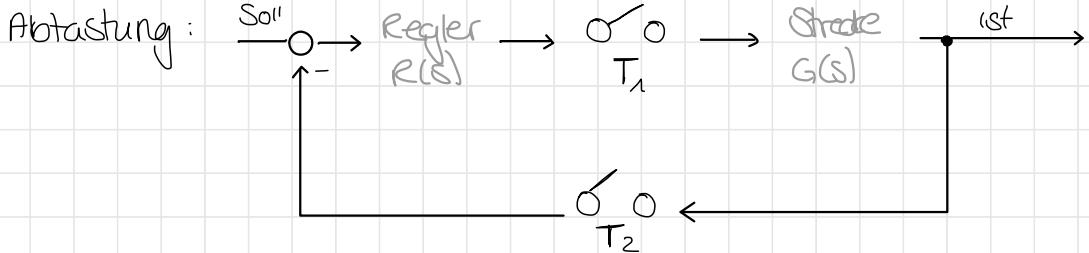
Faustregel: Kompensiert die 2 größten Zeitkonstanten

$$H(s) = \frac{K_s}{(1 + T_{S1}s)(1 + T_{S2}s) \dots (1 + T_{Sn}s)}$$

$$R(s) = K_p \frac{(1 + T_{R1}s)(1 + T_{R2}s)}{T_{R1}s}$$

~ Kompensierte  $T_{R1} = T_{S1}$ ;  $T_{R2} = T_{S2}$

# Diskrete Systeme



## Shannon Abtasttheorem

Abtasten mit  $> 2 \cdot f_{\max}$  um ursprüngliches Signal ohne Infoverlust zurück gewinnen

## Modellierung

	Zeitbereich	Frequenzbereich
kontinuierlich	DGL	Laplace - Trafo
diskret	Differenzengleichung	Z - Trafo

## DGL $\rightarrow$ Differenzengleichung

$$\begin{aligned}
 x(t) &\rightsquigarrow x(k) \\
 \frac{dx}{dt} &\rightsquigarrow \Delta x \\
 \frac{d^2x}{dt^2} &\rightsquigarrow \Delta^2 x
 \end{aligned}
 \quad a_2 \ddot{x}(t) + a_1 \dot{x}(t) + a_0 x(t) = b_0 w(t)$$

$\left. \begin{array}{l} \\ \\ \end{array} \right\}$

$$a_2 \frac{\Delta^2 x}{\Delta t} + a_1 \frac{\Delta x}{\Delta t} + a_0 x(k) = b_0 w(k)$$

$$\Rightarrow \text{Allgemein: } (x(k) = x_k)$$

$$x_k = -a_1 x_{k-1} - \dots - a_{n-1} x_{k-n+1} - a_n x_{k-n}$$

$$+ b_0 w_k + b_1 w_{k-1} + \dots + b_n w_{k-n}$$

## Z-Transformation

1. Signal  $x(t)$  abtasten  $\rightarrow$  Impulsfolge

$$x^*(t) = \sum_{k=0}^{\infty} x(kT) \delta(t - kT)$$

2. Impulsfolge  $\rightarrow$  Laplace-Träfo

$$X^*(s) = \sum_{k=0}^{\infty} x_k e^{-kTs}$$

3. Ersetze  $z = e^{Ts}$

$$X_z(z) = \sum_{k=0}^{\infty} x_k z^{-k}$$

### Regeln

Linearität

$$c_1 f_1(kT) + c_2 f_2(kT) \rightsquigarrow c_1 F_1(z) + c_2 F_2(z)$$

Faltung

$$f_1(kT) * f_2(kT) \rightsquigarrow F_1(z) \cdot F_2(z)$$

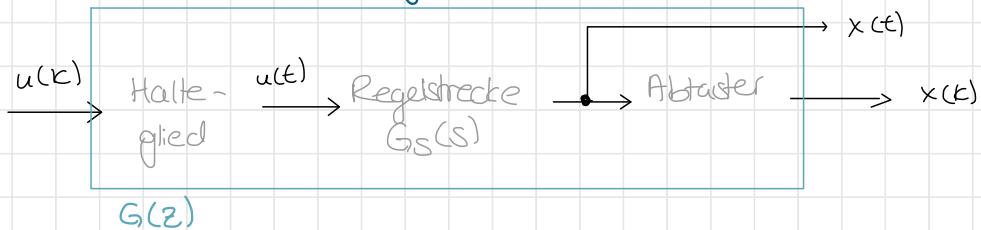
## Z-Übertragungsfunktion

$$X(z) = G(z) \cdot W(z)$$

$$\rightsquigarrow G(z) = b_0 \frac{\prod_{i=1}^m (z - z_i)}{\prod_{j=1}^n (z - p_j)}$$

$\Rightarrow$  asympt. stabil falls  $\underline{|p_j|} = \sqrt{x_j^2 + y_j^2} < 1 \quad \forall p_j = x_j + iy_j$

# Zeitdiskrete Ersatzregelstrecke



$$1. \text{ Sprungantwort} \quad G_1(s) = \frac{G_S(s)}{s}$$

$$2. \quad \mathcal{L}^{-1}\{G_1(s)\} = g_1(t)$$

$$3. \quad z\text{-Träfo von } g_1(t): \quad G_1(z) = z \{g_1(kT)\}$$

$$4. \quad G_1(z) = G(z) \sigma(z)$$

$$G(z) = \frac{x(z)}{u(z)} = \frac{G_1(z)}{\sigma(z)}$$

$$\Rightarrow G(z) = \frac{z-1}{z} \cdot G_1(z)$$

## Filter

### Tiefpass



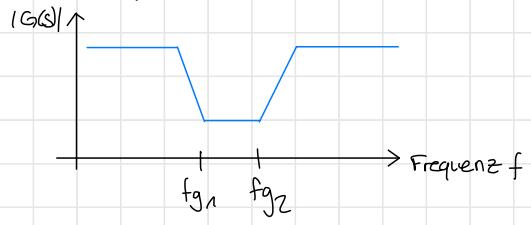
### Hochpass



### Bandpass



### Bandsperr



# Sicherheitskritische Systeme

Norm: IEC 61508

Safety - Schutz des Menschen vor der Maschine

Security = Schutz der Maschine vor dem Menschen

## Safe State

Hazard = Zustand / Event mit potentieller Gefahr für Menschen

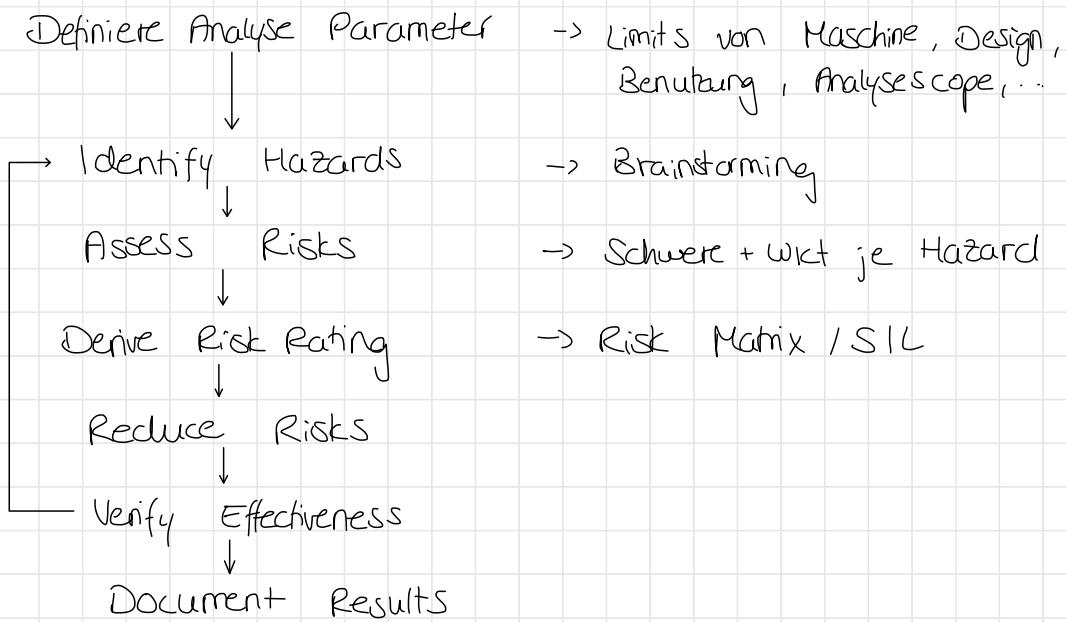
Safety Function = Funktionalität des Systems / Risk Reduction Measures  
um Safe State zu erreichen / halten

Safe State = Systemzustand in dem Safety gegeben ist

## Hazard Analyse

Risiko eines Hazards = Wkt dass Hazard auftritt  
+ Schwere falls Hazard auftritt

## Risk Assessment Protocol



→ Prozess beendet wenn Restrisiko akzeptabel ist  
! das heißt nicht dass alle Risiken gefunden wurden

## Reduce Risks

z.B. durch

- Design ändern sodass Hazard nicht mehr auftritt
- Protection of human
- Warn the user
- Train user(s)
- Personal protective equipment

## System Development

- gemäß HAZOP Phasen

## Hardware

- permanente Fehler
- transiente Fehler : temporäre Fehler, zB Bitkippen

## Risk Matrix / Safety Integrity Level

nicht  
erlaubt

5	SIL3	SIL4	X	X	X
4	SIL2	SIL3	SIL4	X	X
3	SIL1	SIL2	SIL3	SIL4	X
2	-	SIL1	SIL2	SIL3	SIL4
1	-	-	SIL1	SIL2	SIL3

Frequency

Severity of Consequence

ungefährlich →

· Charakterisiert über

- Single Failure Fraction
- Probability of Fail on Demand Average
- Probability of Fail per Hour

· Fehlerrate  $\lambda$

## Software

· KEINE zufälligen Fehler, nur systematische

Bugs verhindern:

- Testen
- SW Dev Process

Coding Rules: Clarity, Predictability, Simplicity, Defense, Compliance, Process, Performance

Reliability  $\leftrightarrow$  Safety



System führt spezifizierte Funktion aus

→ Maßnahmen:

- Redundanz
- Safety Faktoren, Margins
- Derating
- Self-Tests
- Timed component replacements