ECHTZEITSYSTEME

Vorlesung Wintersemester 2000/2001

Dr. Gerhard Schrott
nach dem Skript von Prof. H.-J. Siegert

Institut für Informatik

Lehrstuhl für Echtzeitsysteme und Robotik

TU München

Inhaltsverzeichnis

Literatur

[1] Adam, J.A.: The Seargant York Gun: a Massive Misfire. IEEE Spectrum, 24(2):28–35, 1987.

[2] Adam, J.A.: Technology combats disabilities. IEEE Spectrum, 31(10):24–26, 1994.

[3] Avizienis, A.: Toward Systematic Design of Fault-Tolerant Systems. IEEE Computer, 30(4):51–58, 1997.

[4] Barnes, J.: Programming in Ada 95. Addison-Wesley, 1995.

[5] Bell, T.E., Esch, K.: The Fatal Flaw in Flight 51-L. IEEE Spectrum, 24(2):36–51, 1987.

[6] Bell, T.E. et al.: Special Report: Designing and Operating a Minimum Risk System. IEEE Spectrum, 26(6), 1989.

[7] Bell, T.E.: Electronic and the Stars. IEEE Spectrum, 32(8):16–24, 1995.

[8] Bender, K.: PROFIBUS - Der Feldbus für die Automation. Carl Hanser Verlag, 2 Aufl., 1992.

[9] Berry, G., Gonthier, G.: The ESTEREL Synchronous Language: Design, Semantic, Implementation. Journal of Science of Computer Programming, 19(2):87–152, 1992.

[10] Blazewicz, J. et al.: Scheduling in Computer and Manufacturing Systems. Springer-Verlag, 1994.

[11] Bolch, G., Vollath, M.M.: Prozeßautomatisierung. Teubner, 2 Aufl., 1991.

[12] Burns, A., Wellings, A.: Real-Time Systems and their Programming Languages. Addison-Wesley, 1992.

[13] Conrads, D.: Datenkommunikation. Vieweg-Verlag, 1993.

[14] Dagnelie, G., Massof, R.W.: Toward an Artificial Eye. IEEE Spectrum, 33(5):20–29, 1996.

[15] Douglas, B.P.: Real-Time UML: Developing Efficient Objects for Embedded Systems. Addison-Wesley, 1997.

[16] Driankov, D., Hellendoorn, H., Reinfrank, M.: An Introduction to Fuzzy Control. Springer-Verlag, 2 Aufl., 1996.

[17] Etschberger, K.: CAN Controller–Area–Network. Carl Hanser Verlag, 1994.

[18] Färber, G.: Prozeßrechentechnik. Springer-Verlag, 2 Aufl., 1992.

[19] Gemmer, A.: Risk Management: Moving Beyond Process. IEEE Computer, 30(5):33–43, 1997.

[20] Gomaa, H.: SofwareDesign Methods for Concurrent and Real-Time Systems. Addison-Wesley, 1993.

[21] Görke, W.: Fehlertolerante Rechensysteme. Oldenbourg Verlag, 1989.

[22] Haag, S., Raja, M.K., Schkade, L.L.: Quality Function Deployment Usage in Software Development. Communications of the ACM, 39(1):41–49, 1996.

[23] Halbwachs, N.: Synchronous Programming of Reactice Systems. Kluwer Academic Publishers, 1993.

[24] Herrtwich, R.G., Hommel, G.: Kooperation und Konkurrenz - Nebenläufige, verteilte und echtzeitabhängige Programmsysteme. Springer-Verlag, 1989.

[25] Hohler, B., Villinger, U.: Normen und Richtlinien zur Prüfung und Qualitätssicherung von Steuerungssoftware. Informatik-Spektrum, 21(2):63–72, 1998.

[26] Hüsener, Th.: Entwurf komplexer Echtzeitsysteme - State of the Art. BI Wissenschaftsverlag, 1994.

[27] Jacobson, E.: Einführung in die Prozeßdatenverarbeitung, 2. Aufl. Carl Hanser Verlag, 1996.

[28] Jakoby, W.: Automatisierungstechnik - Algorithmen und Programme. Springer-Verlag, 1996.

[29] Jurgen, R.K.: The Electronic Motorist. IEEE Spectrum, 32(3):37–48, 1995.

[30] Kahlert, J.: Fuzzy Control für Ingenieure. Vieweg-Verlag, 1995.

[31] Kasser, J.: Applying Total Quality Management to Systems Engineering. Artech House Publishers, 1995.

[32] Kassakian, J.G. et al.: Automotive Electrical Systems circa 2005. IEEE Spectrum, 33(8):22–27, 1996.

[33] Kavi, K.M. (Hrsg.): Real-Time Systems: Abstraction, Languages, and Design Methodologies. IEEE Computer Society Press, 1992.

[34] Kratzer, K.P.: ADA, Eine Einführung für Programmierer. Carl Hanser Verlag, 1996.

[35] Laplante, P.A.: Real-Time Systems: Design and Analysis. IEEE Computer Society Press, 1992.

[36] Lauber, R.: Prozeßautomatisierung I. Springer-Verlag, 2 Aufl., 1989.

[37] Leveson, N.G., Thurner, C.S.: An Investigation of the Therac-25 Accidents. IEEE Computer, 26(7):18–41, 1993.

[38] Pham, Hoang (Hrsg.): Fault-Tolerant Software Systems: Techniques and Applications. IEEE Computer Society Press, 1992.

[39] Rzehak, H. (Hrsg.): Echtzeitsysteme und objektorientierte Entwurfstechniken. Vieweg-Verlag, 1996.

[40] Schiebe, M., Pferrer, S.: Real-Time Systems: Engineering and Applications. Kluwer Academic Publishers, 1992.

[41] Schwarz, K.: Manufacturing Message Specification MMS. Automatisierungstechnische Praxis atp 31, pp. 23–29, 1989.

[42] Siegert, H.-J.: Betriebssysteme - Eine Einführung. Oldenbourg Verlag, 4 Aufl., 1998.

[43] Sweet, W.: The Glass Cockpit. IEEE Spectrum, 32(9):30–38, 1995.

[44] Voges, U.: Software - Diversität und ihre Modellierung. Informatik–Fachberichte 224, Springer-Verlag, 1989.

[45] Zhu, H., Hall, P.A.V., May, J.H.R.: Software Unit Test Coverage and Adequacy. ACM Computing Surveys, 29(4):366–427, 1997.

[46] Zöbel, D., Albrecht, W.: Echtzeitsysteme – Grundlagen und Techniken. Int. Thomson Publ., 1995.

- Zeitschriften:
  - atp–Automatisierungstechnische Praxis
  - Real-Time Systems
  - Embedded Systems
  - Elektronik
- Tagungen:
  - GI Fachgruppe Echtzeitprogrammierung PEARL
  - IEEE Real-Time Systems Symposium
  - IFAC/IFIP Workshop on Real-Time Programming
  - Euromicro Conference on Real-Time Systems
  - Embedded Systems (Messe Nürnberg)
- Ergänzende Vorlesungen:
  - Betriebssysteme
  - Rechnernetze
  - Microprozessoren
  - Zuverlässigkeit und Fehlertoleranz von Rechensystemen
  - Software-Technik

1 Echtzeitbetrieb

1.1 Grundlegende Definitionen

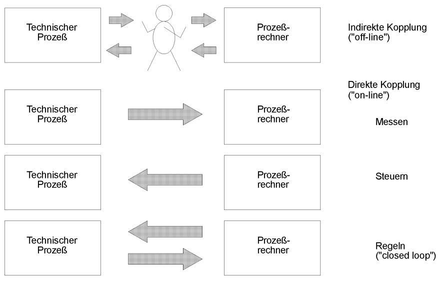
(nach DIN 66201 Prozeßrechensysteme)

- Prozeß
  Umformung und/oder Transport von Materie, Energie und/oder Information (deterministisch oder stochastisch).
  Anmerkung: Prozeß in der Informatik üblicherweise: Ablauf eines Programms, wenn dieser Ablauf Verwaltungseinheit des Betriebssystems ist.
- Technischer Prozeß
  Prozeß, dessen Zustandsgrößen mit technischen Mitteln gemessen, gesteuert und/oder geregelt werden können.

- Prozeßführung
  Erreichen eines vorgegebenen Ziels aus Kenntnis der Istwerte unter Berücksichtigung der Kapazitäten; flexible Anpassung an kleine Störungen; Anzeige in Prozeßwarten.

  Die Führung technischer Prozesse kombiniert Planung, Steuerung und Regelung.
- Steuerung
  Einwirkung mittels Stellgrößen in eine gewünschte vorausgeplante Richtung.
  Beispiel: Heizung um 6:30 Uhr einschalten.
- Regelung
  Überwachung und Minimierung der Abweichungen zwischen Ist– und Sollwerten nach gegebenen Kriterien.
  Beispiel: Kesseltemperatur auf 43 Grad halten.

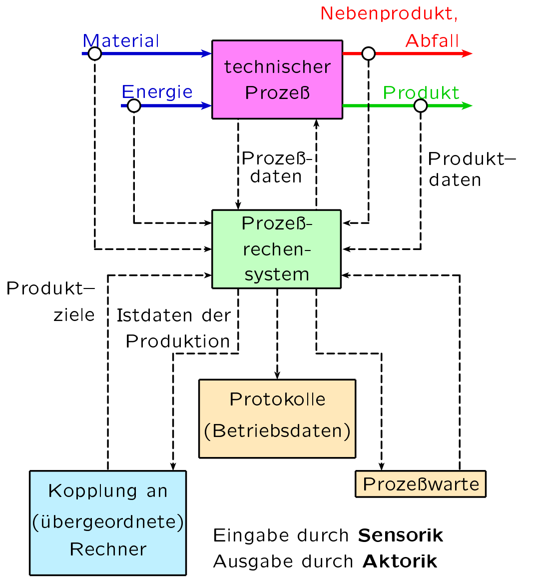
1.2 Aufgaben Echtzeitsysteme



Je nach Komplexität:

- Datensammlung über den technischen Prozeß und Datenaufbereitung
- Steuerung technischer Prozesse
- Regelung technischer Prozesse

1.3 Das Prozeßrechensystem als Regelkreis



Eingabe durch **Sensorik**
Ausgabe durch **Aktorik**

1.4 Anforderungen an Echtzeitsysteme

- Echtzeitfähigkeit
  harte Zeitbedingungen, kurzfristige Reaktionszeiten
- Anschluß vielfältigster Peripherie
  (Geräte, Sensorik, Aktorik)
- Algorithmische Erfassung der Struktur des technischen Prozesses
- Mensch–Maschine–Schnittstelle
  für Datenrepräsentation, Datensammlung und ggf. Datenauswertung
  für die Steuerung
- Zuverlässigkeit, Sicherheit
- Fehlertoleranz
- Intelligenz, um Vorgehenvorschläge in bestimmten Situationen, insbesondere im Störfall, machen zu können

1.5 Historie

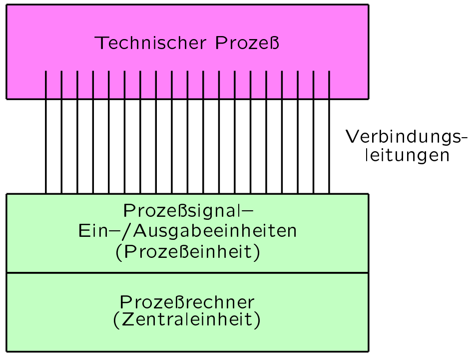
Wichtige Entwicklungsschritte der Echtzeitsysteme:

- Bis 1960 Automatisierung einzelner Regelstrecken (Relaistechnik, Transistorschaltungen, Analogrechner)
- Ab ca. 1960 Einsatz von Digitalrechnern
  Begriff "Prozeßrechner" (historisch,unpräzis) für die damaligen Eigenentwicklungen:
  - ⋆ Kurzwortmaschinen (8- oder 12-Bit-Worte)
  - ⋆ Zentralgerät
  - ⋆ robuste Ausführung
  - ⋆ wenig Speicher (teuer!), Bit–Programmierung
  - ⋆ spezielle EA–Kanäle
  - ⋆ kein allgemeines Betriebssystem, sondern anwendungsspezifische Ablaufsteuerungen

- Ab ca. 1976 Massenproduktion von Mikrocomputern, insbesondere auch für "embedded systems"
  aber auch klassische "Prozeßrechner" wie PDP11, LSI11, VAX
  - ⋆ Führung selbst kleinster technischer Prozesse durch Rechner (Tisch–PC; kundenspezifische Chips (ASIC, application specified IC) in Geräten und Maschinen, embedded systems)
  - ⋆ globale Regelungs– und Überwachungsstrategien
- seit Anfang 90er Standardrechner (Workstations und PC) mit Echtzeit-Betriebssystemen
  - ⋆ Bussysteme und Rechnernetze
  - ⋆ CIM (computer integrated manufacturing)
  - ⋆ Expertensysteme zur Prozeßführung

1.6 Beispiele zur Automatisierung

konventionelles Prozeßrechensystem, Blockschaltbild

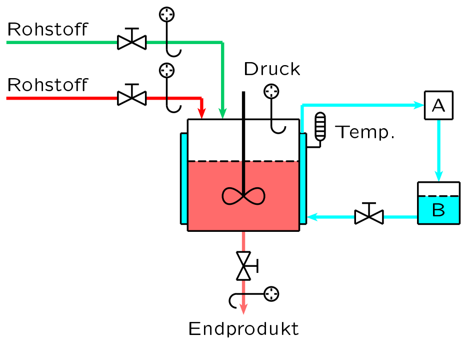


- Ein–/Ausgabeeinheiten und ihre Symbole

Lichtschranke
Füllstandsmelder
Thermostat
Endschalter
Taster
Schrittmotor, Servo–Motor
Ventil, handverstellbar
Magnetventil
Alarm
Lampe

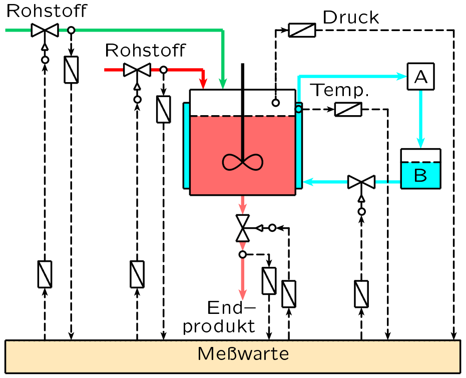
Heizung
Schütz
Weggeber
Widerstandsthermometer
Thermoelement
Hallgenerator
Tachogenerator
XY–Schreiber
automatisches Testsystem
Meßgerät
Meßfühler
Meßwertumformer, Signalumformer
Regler mit Ist–/Sollwerteingang
Stellglied (Stellantrieb+Stellorgan)

- Automatisierung eines Rührkesselreaktors, Stufe 1: handgesteuert



A = Kühlung
B = Kühlflüssigkeit

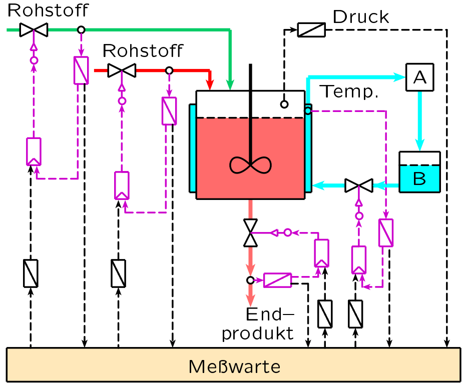
- Automatisierung eines Rührkesselreaktors, Stufe 2: wartengesteuert



A = Kühlung
B = Kühlflüssigkeit

Meßwarte: im einfachsten Fall abgesetzte Bedienung, sonst Prozeßrechner mit Steuerung oder Regelung

- Automatisierung eines Rührkesselreaktors, Stufe 3: wartengesteuert, lokale Regler

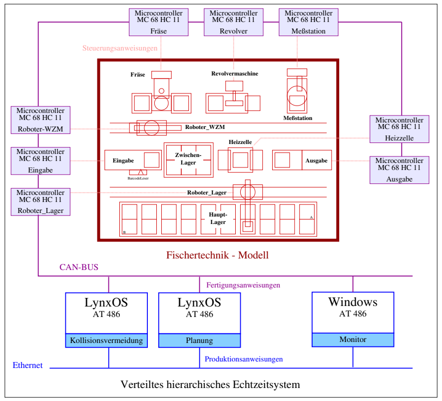


A = Kühlung
B = Kühlflüssigkeit

Lokale Regler können als unterste Ebene einer hierarchischen Prozeßführung betrachtet werden

Moderne Entwicklungen:

- Zentraler Prozeßrechner durch vernetzte Rechensysteme ersetzt
- lokale Prozeßrechner bei den einzelnen Komponenten des Prozesses
- heutige Kosten der Rechner erlaubt Regelungen durch Rechner
- Punkt-zu-Punkt-Verbindungen zu den Ein-/Ausgabeeinheiten durch Feldbusse (Netze) ersetzt
- intelligente automatisierte Geräte
  Meßgeräte,
  Robotersteuerung,
  NC-Steuerung,
  SPS



Beispiel der verteilten Steuerung einer Fertigungsanlage

1.7 Einsatzbeispiele

- Steuerung einzelner Geräte und Maschinen
  - ⋆ Geräte des täglichen Bedarfs (Mikrowellenherd, CD-Spieler, Kamera, Video-Recorder, Anti-Blockier-System, Anti-Schlupf-Regelung und Elektronisches Stabilitätsprogramm im Auto)
  - ⋆ Fertigungsmaschinen (Werkzeugmaschine, Roboter)
  - ⋆ Verkehrsmittel (Aufzug, Magnetbahn, Flugzeug)
  - ⋆ naturwissenschaftlich-medizinische Geräte (Massenspektrometer, Tomograph)
  - ⋆ militärische Geräte (Panzer, Radarsystem, Rakete)
- Führung komplexer technischer Vorgänge
  - ⋆ Steuerung von Fertigungsanlagen, Walzwerken, etc.
  - ⋆ Kontrolle und Überwachung von Kraftwerksanlagen (Bsp. 700 MW Kohlekraftwerk)
  - ⋆ Umweltschutz-Warnanlagen

  - ⋆ Verkehrslenkung
- Konkrete Einzelbeispiele
  - ⋆ Boeing 777: über 1000 Rechner/Mikrocontroller pro Flugzeug z.B. 10 Stück fürs Bremssystem
  - ⋆ 700 MW Kohlekraftwerk:
    750 + 1250 Antriebe
    3000 + 3000 Meßwerte
    2200 + 5000 Zustandsmeldungen
    4500 + 15000 Alarmmeldungen
    Hauptblock + Nebenanlagen Entschwefelung
  - ⋆ Channel Tunnel: ca. 100 Rechner und Workstations, ca 200 Mikrocontroller
  - ⋆ Herzschrittmacher:
    500 000 Lines of Code (loc) in C
  - ⋆ TGV-Subsystem:
    380 000 loc, davon 220 000 loc in Ada
  - ⋆ Autoradio: 1 MB Code
  - ⋆ Fernsehgerät: Übergang zum Multitasking

1.8 Ziel der Automatisierung

- monotone oder gefährliche Arbeiten konstant gut ausführen (z.B. lackieren, schweißen, füllen, prüfen)
- stets hohe Präzision und Geschwindigkeit einhalten, trotzdem schnelle Reaktion auf Fehler
- große Datenflut bewältigen und verwalten, insbesondere auch bei Störfällen
- viele (oft kleine) Regelkreise überwachen
- komplexe Steuerungsvorgänge durchführen
- Wirtschaftschaftlichkeit steigern, Termine halten und verkürzen (flexibel, schnell, kleine Loszahlen, billig)

1.9 Zusammenwirken

Automatisierungstechnik ist ein Zusammenwirken von:

- Halbleitersensorik, Mikroperipherik, Gerätetechnik (neue Materialien)
- rechnergestützter Meßtechnik
- Steuerungs–und Regelungstechnik (globale, robuste, adaptive Algorithmen; Fuzzy–Regelungen)
- VLSI–Entwurfssysteme und Techniken für Spezialchips in kleiner Stückzahl (ASIC–Chips)
- Informatik:
  - ⋆ Spezifikations– und Entwurfssysteme
  - ⋆ Sprachen
  - ⋆ Betriebssysteme
  - ⋆ Feldbus– und Netzprotokolle
  - ⋆ Expertensystem-Rahmen
  - ⋆ Entwurf und Realisierung der Anwendungen
  - ⋆ Integration in technische und organisatorische Umgebung

1.10 Themen der Prozeßinformatik (Echtzeit–Datenverarbeitung)

Fragestellungen aufgrund (detaillierterer) Anforderungen, wie:

- Vielfältige technische Prozesse
- viele Peripherie–Varianten
- spezielle Scheduling-Strategien
- ein Echtzeitsystem muß mit dem vom technischen Prozeß vorgegebenem Zeitverhalten Schritt halten
- Rechtzeitigkeit, Gleichzeitigkeit, ständige Betriebsbereitschaft (determinierter vorhersehbarer Gesamtablauf bei stochastischen Ereignissen)
- besonders hohe Verfügbarkeit und Sicherheit (insbesondere bei militärischen oder medizinischen Anwendungen, bei Personenverkehr, in der Produktion)
  - ⋆ robust gegen unsachgemäße Handhabung,
  - ⋆ unempfindlich gegen Einzelfehler,

  - ⋆ allmählicher Leistungsabfall bei Fehlern (gracefull degradation)
  - ⋆ fail–safe–Übergänge, d.h. keine Gefahr für Geräte und Menschen bei beliebigen Fehlerkombinationen
- TÜV-Abnahme

Daneben nichttechnische Fragen, wie:

- neues betriebswirtschaftliches Vorgehen, z.B. lean production mit neuen Strukturen der Bereiche und ihrer Verantwortung
- Schonung der Resourcen
- Kostenminimierung
- Qualitätsmanagement
  - ⋆ Qualitätssicherung
  - ⋆ Qualitätsoptimierung
  - ⋆ Produkthaftung
  - ⋆ Senkung Reklamationskosten
  - ⋆ Senkung Produktionskosten bei frühzeitiger Fehlererkennung
  - ⋆ Qualitätsnachweise

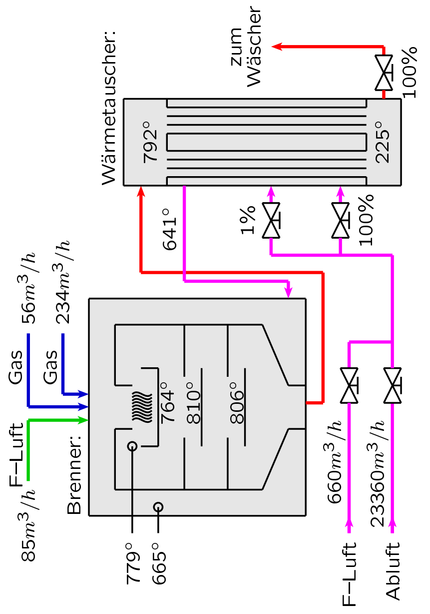
- Chargennachweise
  - ⋆ Rückverfolgung auf "Verursacher"
  - ⋆ Nachweis der Verwendung
  - ⋆ Verfolgung des Materials vom Wareneingang bis Warenausgang über Zeit– und Ortsbezug
- Mensch–Maschine–Kommunikation
  - ⋆ (humane) Leitwarten
  - ⋆ selbsterklärende Bedienung (auch bei seltener Benutzung einer Aktion oder Anzeige)
  - ⋆ Anzeige nur der relevanten Daten, die in der jeweiligen Situation aktuell sind
  - ⋆ alle Aktionen des Menschen, ggf. nach Warnung, Begründung warum eigentlich nicht zulässig und Bestätigung, erlauben (negatives Beispiel: Flugzeug verunglückte, da Pilot Umkehrschub beim Schleifen auf vereister Piste nicht einschalten konnte; Grund: Programmierer ging davon aus, daß das Flugzeug in der Luft ist, wenn sich Räder nicht drehen und deshalb Umkehrschub zum Absturz führt)

  - ⋆ Unterstützung durch intelligente Systeme im Fehlerfall
- Sicherheitsprobleme
  - ⋆ Spezifikationsfehler
  - ⋆ Hardware–,Software–, Schnittstellenfehler
  - ⋆ blindes Vertrauen?,
  - ⋆ Restrisiko,
  - ⋆ Verwundbarkeit
  - ⋆ Viren
- soziale und gesellschaftspolitische Fragen (höhere Qualifikation, Wegrationalisieren von Arbeitsplätzen, wirklich humaner?, Fehlen menschlicher Kontakte)

1.11 Funktionen zur Prozeßführung

- Betreiben der Peripherie
  Lesen/Schreiben der Meß– und Stellgrößen (digital/analog)
- Überwachen und Melden
  - ⋆ Test auf Grenzwertüberschreitung und Änderungsgeschwindigkeit
  - ⋆ Test auf gleichartiges Verhalten abhängiger Größen, beispielsweise gleicher Trend bei Druck und Temperatur
  - ⋆ Test auf defekte Komponenten, beispielsweise durch
    - ∘ Plausibilitätskontrolle
    - ∘ zeitliche Redundanz (mehrfaches Lesen)
    - ∘ Selbst- und Fremdtests
    - ∘ redundante Komponenten
    - ∘ verschiedene Meßmethoden

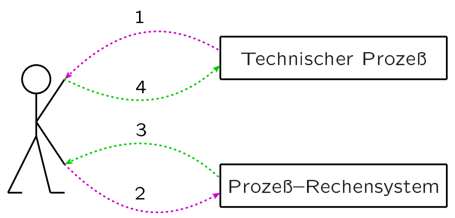
- Protokollierung, Anzeige
  (Prozeßwarte, Logbuch, Störungsstatistik)
- Höhere Meßwertverarbeitung
  - ⋆ Datenreduktion
  - ⋆ Mittelwerte
  - ⋆ Glätten
  - ⋆ Filtern
  - ⋆ nur Anzeige momentan relevanter Daten
  - ⋆ Trendanzeigen)
- Führung lokaler Regelstrecken
  (z.B. PI–,PID–Regler; robuste und adaptive Verfahren; Fuzzy–Regler)
- lokal optimierte Ablaufsteuerung (Leitsysteme)
- globale Gesamtoptimierung von Fertigungszellen (Planungssysteme)
- Gesamtkoordination: Optimierung nach betriebswirtschaftlichen und technischen Gesichtspunkten (cooperate model)
- Prozeßvisualisierung



Beispiel 2 zur Prozeßvisualisierung (mit Zuständen)

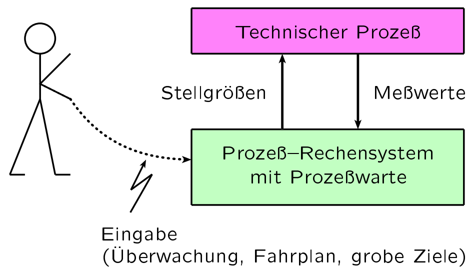
1.12 Arten Rechnerkopplung an Prozeß

- Indirekte Kopplung (off line–Betrieb)
  - ⋆ Zeitliche und gerätemäßige Entkopplung
  - ⋆ Bedienung durch Personal, z.B.
    - ∘ Versuchsauswertungen
    - ∘ Medizin EKG
    - ∘ Profilaufnahme von Vorgängen
    - ∘ Qualitätsstatistik
  - ⋆ Nachteil: fehleranfällig, langsam



1: Messen 3: Steuerungsvorschlag
2: Zustandsdaten 4: Stellen

- Direkte Kopplung (on line–Betrieb)
  Der Rechner übernimmt die Meßwerte direkt vom technischen Prozeß und greift mittels Stellgrössen ein



- ⋆ offene Prozeßkopplung
  (on line open loop control)
  Die Eingabedaten (Meßwerte) haben keine Rückwirkung auf die Ausgabedaten (Stellwerte), z.B. Stimulation eines Versuchs mit festen Eingabewerten, Protokollierung (z.B. Prüfstände), Qualitätskontrolle (Profil eingeben und Reaktion beobachten).

- ⋆ geschlossene Prozeßkopplung
  (on line closed loop control)
  Die Eingabedaten (Meßwerte) beeinflussen die Ausgabedaten (Stellwerte), z.B. Rückführregelung, nachgeführtes oder neues Betriebsziel.
  Stichwort DDC (**d**irect **d**igital **c**ontrol)
  "dedicated systems": viele kleine Regelkreise mit vorgefertigten Programmbausteinen zum Konfigurieren und mit automatischer Dokumentation.

1.13 Prozeßmodelle

- Typische Prozeßgrößen
  - ⋆ Kennwerte oder Anlagenwerte (z.B. Geometrie, Maße, Frequenzen, Grenzwerte)
  - ⋆ Zustandswerte (Prozeßvariable), setzen Modell voraus
  - ⋆ Parameter (einstellbare Werte, z.B. Waschprogramm)
  - ⋆ Eingangsgrößen (unabhängig vom Prozeß, z.B.
    - ∘ Zufluß
    - ∘ Stellgrößen
    - ∘ Störgrößen (von außen, unbeabsichtigt; z.B. Wärme, Rauschen)
  - ⋆ Ausgangsgrößen (z.B. Abfluß, Meßgrößen, Alarme)

- Klassifikation der Prozeßmodelle nach
  - ⋆ Aufgabenstellung (Produktion, Transport, Qualitätsprüfung)
  - ⋆ Material (Fertigung, Verarbeitung in der Chemie, in der ...)
  - ⋆ Energie (Wärme, Strom, Gas)
  - ⋆ Information (Meß– und Prüftechnik; Fernsehen)
  - ⋆ zeitlichem und räumlichem Ablauf (kontinuierlich bzw. diskontinuierlich)
  - ⋆ Modellbildung
    - ∘ analytisch (math., phys. oder chem. Gleichungen)
    - ∘ empirisch aus Beobachtung oder Erfahrungen
    - ∘ Prozeßidentifikation aus Meßwerten
    - ∘ gegenständliches Modell in anderem Maßstab

- Genauer betrachtet: Klassifikation nach zeitlichem und räumlichem Ablauf:
  - ⋆ Fließprozesse (dynamische P., kontinuierliche P.)
    - ∘ (stückweise) stetige Prozeßgrössen
    - ∘ zeit– und/oder ortsabhängige Variable
    - ∘ mathematische Modelle durch DGL–Systeme oder Übertragungsfunktionen
  - ⋆ Folgeprozesse (sequentielle P., diskontinuierliche P.)
    - ∘ binäre, diskrete Informationsfolgen, die Ereignissen oder bestimmten Zeitpunkten zugeordnet sind
    - ∘ mathematische Modelle durch Boole'sche Gleichungen, Automaten, Petri-Netze oder Ablaufpläne
  - ⋆ Stückgutprozesse
    - ∘ objektbezogen einzeln identifizierbare Objekte, die Position und/oder Zustand verändern
    - ∘ Simulationsmodelle

- ⋆ Chargenprozesse
  Teilmengen (Phasen) eines sonst kontinuierlichen Prozesses (vor allem in der chemischen Industrie)
- Genauer betrachtet: Klassifikation der analytischen Modelle
  - ⋆ deterministisch (exakt), z.B. Fahrpläne beim Schienenverkehr
  - ⋆ stochastisch (Wahrscheinlichkeitsaussagen), z.B. Straßenverkehr
  - ⋆ zeitabhängig
    - ∘ statische Ein– und Ausgangsgrößen
    - ∘ stationärer Zusammenhang der Größen
    - ∘ dynamisch (zeitl. Änderungen der Größen)
  - ⋆ adaptiv (Parameter und Koeffizienten werden laufend aktuell nachjustiert0, z.B. zur Elimination von Störungen
  - ⋆ lernend (Bestimmung von Typ und Grad der Funktionen aus Meßwerten), z.B. neuronale Netze

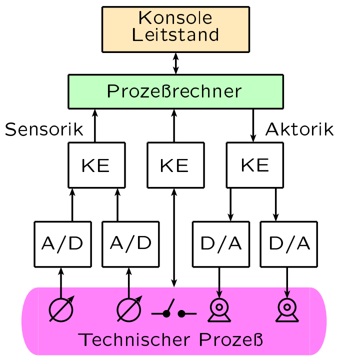
2 Architekturen von Prozeßrechensystemen

- Grundformen
  - ⋆ klassischer Prozeßrechner
  - ⋆ eingebettete Systeme (embedded systems)
  - ⋆ hierarchische Rechnerarchitektur

2.1 Klassischer Prozeßrechner

- "kleiner" Rechner
- früher oft Spezialrechner (Kosten), heute Tendenz zu Standardrechnern, z.B. PC
- EA-Anschlüsse für vielfältige Peripherie
- robust gegen Umwelteinflüsse, z.B.
  - ⋆ schmutzsicher
  - ⋆ spritzwasserdicht
  - ⋆ abgeschirmt gegen Vibrationen
  - ⋆ unempfindlich gegen EMS
  - ⋆ rüttelsicher

• Beispiel:



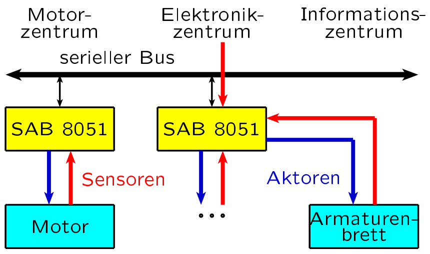
∅	Meßwertgeber
	Stellwertgeber
	Signalein-/ausgabe
A/D	Analog/Digital–Wandler
D/A	Digital/Analog–Wandler
KE	Koppelelektronik

- Die vor 20 bis 30 Jahren übliche Struktur findet sich heute vereinfacht in den speicherprogrammierbaren Steuerungen (SPS). Diese Rechner erfassen periodisch die Eingangssignale und geben nach einfachen logischen Verknüpfungen entsprechende Ausgangssignale ab.
- ggf. erhöhte Zuverlässigkeit, u.a. durch
  - ⋆ redundante Geräte
  - ⋆ redundante EA-Kanäle
  - ⋆ unterbrechungsfreie Stromversorgung (USV)
  - ⋆ nichtflüchtiger Speicher (batteriegepuffert, EEPROM)
  - ⋆ "Powerfail"-Unterbrechung

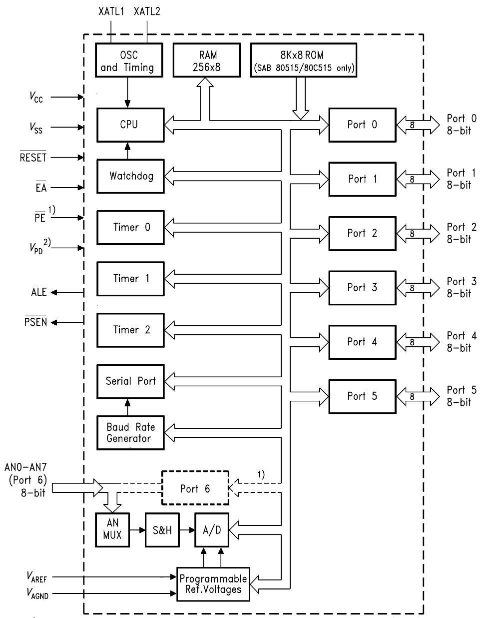
2.2 Eingebettete Systeme (embedded systems)

- Rechner ist in technischen Prozeß eingebettet
- meist für Steuerung eines einzigen Geräts
- meist ohne vielfältige Peripherie (z.B. keine Platte, kein Bildschirm, keine Tastatur)
- Anwendungsbeispiele
  - ⋆ Waschmaschine
  - ⋆ Telefon
  - ⋆ Kraftfahrzeug (ABS)
  - ⋆ Videogerät
  - ⋆ Werkzeugmaschine
- Verbindung eingebettete Systeme über Feldbus (z.B CAN–Bus, PROFI–Bus), IrDA, GSM oder Firewire.
- Ein-Chip-Systeme (Microcontroller)

• Applikationsbeispiel



- Microcontroller SAB 80(C)515 (Siemens)
  - ⋆ 8-Bit-CPU
  - ⋆ 8 kB ROM, 256 B RAM on chip
  - ⋆ 6 parallele Ports je 8 bit, 1 serieller Port
  - ⋆ A/D-Wandler, 8 Analog-Eingänge
  - ⋆ 3 Zähler je 16 bit, Wecker, Uhr
  - ⋆ 7 externe Unterbrechungseingänge
  - ⋆ 4 μs Multiplikation, 12 MHz
- für Massenmarkt optimiert



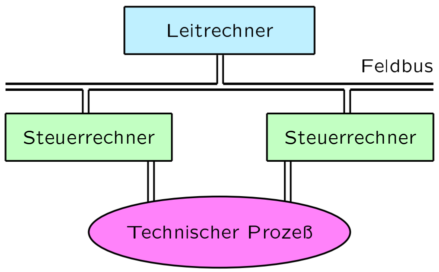
1) Additional feature of the ACMOS versions
2) Additional feature of the MYMOS versions

- kundenspezifische Schaltung anwendungsspezifisches IC application specified integrated circuit (ASIC)
  - ⋆ programmable logic device (PLD, PLA, GAL, FPLD, CPLD)
  - ⋆ programmable gate array (PGA, FPGA)
  - ⋆ Entwicklung der Programme mit eigenem Entwicklungssystem (host) für den Zielrechner (target): Cross–Compiling
  - ⋆ Laden in ASIC: download
  - ⋆ Maskenerstellung und Fertigung in Stückzahlen
  - ⋆ Testen nach Integration in den technischen Prozeß mit speziellen Geräten: remote debugging

2.3 Hierarchische Rechner–Architektur

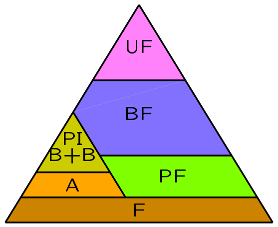
- Ziele
  - ⋆ Aufbau komplexer,kooperierender Systeme aus einfacheren Komponenten
  - ⋆ Verringerung Komplexität durch Dezentralisierung
  - ⋆ Einhaltung von Zeitbedingungen durch Widmung kleinerer Rechner
  - ⋆ abgestufte Zeitforderungen auf den Hierarchieebenen
  - ⋆ Standard-Systeme (mit der guten Programmier- bzw. Benutzerumgebung) wenn nicht harte Echtzeitfähigkeit
  - ⋆ Erweiterbarkeit durch Dezentralisierung
  - ⋆ Integration auch der technischen Prozesse in die Gesamtorganisation

• Beispiel: zweistufige Hierarchie



- ⋆ Leitrechner plant dynamisch die Bearbeitungsvorgänge in den Prozeßführungssystemen (Steuerrechner)
- ⋆ Für Steuerrechner gelten härtere Echtzeitbedingungen als für Leitrechner angelegt

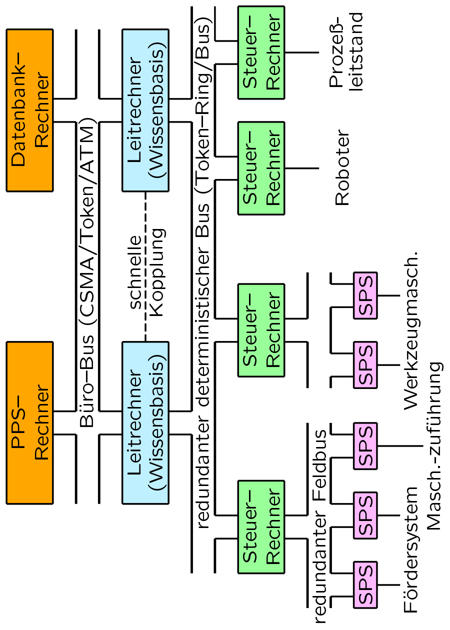
- Mehrstufige Hierarchie



Dabei bedeuten:

- ⋆ UF: Unternehmensführung. Zielvorgaben, Planung, Optimierung der Gewinne.
- ⋆ BF: Betriebsführung. Management der Ressourcen, Optimierung der Erzeugungskosten.
- ⋆ PF: Prozeßführung. Steuerung und Regelung des technischen Prozesses. Ziele: Qualität, Effizienz und Zuverlässigkeit.
- ⋆ PI: Sammlung von Prozeßinformation.
- ⋆ B+B: Bedienen und Beobachten (Wartung)
- ⋆ A: Automatisierung des Betriebs.
- ⋆ F: Feldebene, Feldgeräte.

- Verbindungsstrukturen
  - ⋆ LAN, insbesondere Token-Ring (Bus) auf höheren Ebenen
    CSMA/CD ginge auch, aber traditionell Scheu davor in Echtzeitsystemen
  - ⋆ Feldbus auf tieferen Ebenen
  - ⋆ zur Verbesserung der Zuverlässigkeit redundante Wege
- Fertigungsumgebung:
  Führung des technischen Prozesses mit vierstufiger Hierarchie
  - ⋆ Planungsebene
  - ⋆ Leitebene
  - ⋆ Steuerebene
  - ⋆ Feldebene



- Zu den verschiedenen Ebenen
  - ⋆ Planungsebene
    - ∘ Standard–Betriebssystem
    - ∘ Auftragsplanung
    - ∘ Gesamtdatenhaltung
    - ∘ Logistik, kaufmännische Organisation
    - ∘ Produktionsplanung (PPS)
    - ∘ Archivierung der Daten
    - ∘ Verbindung zum Vertrieb
  - ⋆ Leitebene
    - ∘ meist Standard–Betriebssystem
    - ∘ Bürobus– und Steuerbus
    - ∘ Wissensbasis (Fakten, Regeln auswerten)
    - ∘ Auftragsvergabe und –verfolgung (Optimierung, Logistik)
    - ∘ Fehler–und Störungsbehandlung mit evtl. Neuplanung
  - ⋆ Steuerebene
    - ∘ Echtzeit–Betriebssystem
    - ∘ Steuerbus– und Feldbus
    - ∘ Auftragsabwicklung
    - ∘ Transformation in Einzelaufgaben

- ∘ Koordination der nachgeordneten Geräte
- ∘ Fehler–und Störungsbehandlung

- ⋆ Feldebene
  - ∘ ASIC, SPS, kleine Spezialrechner
  - ∘ spezielles kleines Echtzeit–Betriebssystem oder nur die Anwendung
  - ∘ Feldbus–Protokoll in Hardware
  - ∘ Betreiben der Sensorik/Aktorik
  - ∘ Meßdatenerfassung/ –aufbereitung
  - ∘ Up–/Down–Laden
  - ∘ Automatische Erzeugung von Betriebssystem–Varianten aus Bausätzen (Bibliotheken von Funktionen, Gerätetreibern, Standard–Algorithmen)
  - ∘ Wegen Komplexität des Zusammenbaus Konfigurierungssysteme (Expertensysteme)
  - ∘ Häufig noch maschinenorientierte Programmierung

3 Prozeßrechner–Peripherie

3.1 Sensoren und Aktoren

- Meßwertgeber (Sensoren) liefern elektrische Signale als Funktion des physikalischen Werts
- Stellglieder (Aktoren, Effektoren) benötigen elektrische Signale, Impulse oder Impulsfolgen sowie Hilfsenergie (elektrisch, pneumatisch)
- Koppelelemente von Sensoren und Aktoren mit dem Rechner haben u.a. folgende Aufgaben zu erfüllen:
  - ⋆ halten des Meßsignals
  - ⋆ filtern
  - ⋆ verstärken
  - ⋆ umsetzen
  - ⋆ regeln
  - ⋆ zählen
  - ⋆ logisch oder arithmetisch verknüpfen.

- Folge bei Eingabe Signal:
  Meßgröße x → Signalverarbeitung →
  → A–/D–Wandler → Rechner erhält $\hat{x}$
- Folge bei Ausgabe Signal:
  Rechner ermittelt Stellwert s →
  → D–/A–Wandler → Umsetzer/Stellantrieb →
  → Stellwert $\hat{s}$ im Prozeß
- Charakteristische Größen der Umwandlung
  - ⋆ Haltezeit (sample– and hold– time)
  - ⋆ Umsetzungszeit (conversion time)
  - ⋆ Quantisierungsfehler (Diskretisierungsfehler)
  - ⋆ Schaltungsunzulänglichkeiten
    - ∘ Verstärkungsfehler (gain error)
    - ∘ Nullpunktsverschiebung (offset error)
    - ∘ Linearitätsfehler
    - ∘ (lokales) Verletzen der Monotonie
- heute oft komplexe Kompensationsschaltungen auf Chip–Ebene (Qualitätsunterschiede)

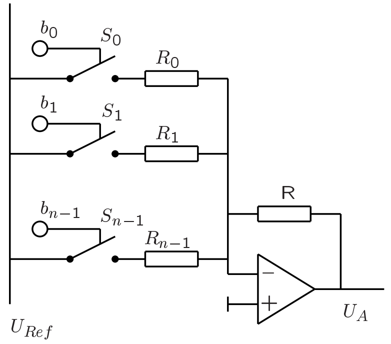
- Beispiele für Sensoren:
  - ⋆ Potentiometer
    Alternativen: Mehrgang-Drehpotentiometer, Lineare Weggeber
  - ⋆ Inkremental-Winkelcodierer mit Drehrichtungserkennung
    Alternative: absolut kodierte Systeme
  - ⋆ temperaturabhängige Widerstände, Temperatur-IC
  - ⋆ Lichtschranken, induktive Sensoren
  - ⋆ mechanische Schalter
    ! mechanische Notabschalter
  - ⋆ optische Sensoren
    Abstandssensor, Farbsensor, Kamera
- Störungen
  - ⋆ elektrische S. (kapazitive Einkoppelung)
  - ⋆ magnetische S. (induktive Einkoppelung)
  - ⋆ Hochfrequenz-Störungen
  - ⋆ elektrostatische Störungen
  - ⋆ Störungen durch Erdschleifen

  Abhilfe: Kabel mit Abschirmung
  und/oder verdrillten Adern

3.2 Digital–/Analog–Umsetzer (DAU)

- digital/analog converter (DAC)
- ein Digitalwert wird in ein elektrisches Signal (Strom oder Spannung) umgesetzt
- Ausgangsspannung
  z. B. 0 bis 10 V oder -5 bis 5 V
- wichtig kleine Wandlungszeit
  (10 nsec bis einige μsec Einschwingzeit)
- wichtig hohe Genauigkeit
  (8, 10, 12, 16 Bit)

- Realisierungsprinzip: Stromsummenwandlung
  - ⋆ Binärwert B $= b_{n-1} \ \ldots \ b_1 b_0 = \sum_{i=0}^{n-1} b_i 2^i$
  - ⋆ gesuchter Analogwert U_A
  - ⋆ gestaffelte Widerstände
  - ⋆ Sequenz von parallel geschalteten Widerständen R, R/2, R/4, R/8 $\cdots$ R/2^{n-1}
  - ⋆ Widerstand $R_i = R/2^i$ durch b_i geschaltet
  - ⋆ Schalter auf, falls $b_i = 0$
  - ⋆ Schalter zu, falls $b_i = 1$
  - ⋆ Blockschaltbild



- ⋆ Der ideale Verstärker regelt wegen der Rückkopplung mit R so, daß der Strom am Eingang vernachläßigbar klein ist.
- ⋆ die gestaffelten Widerstände haben folgenden resultierenden Wert

$$\frac{1}{R_P} = \left(\sum_{i=0}^{n-1} \frac{b_i}{R/2^i} \right)$$

- ⋆ sei j der Strom durch R, dann gilt (Ohmsches Gesetz)

$$U_{Ref} = j * R_P$$

$$U_A = -j * R$$

- ⋆ daraus folgt:

$$U_A = -U_{Ref} * \sum_{i=0}^{n-1} b_i 2^i = -U_{Ref} * B$$

- ⋆ die Ausgangsspannung ist also proportional dem Binärwert B

- ⋆ Probleme
  - ∘ wegen der stark variierenden Größenordnungen sind die Widerstände aus verschiedenen Materialien (abweichendes Temperaturverhalten)
  - ∘ der kleinste Widerstand muß am genauesten sein (allg.: $\triangle R_i / R_i \leq 2^{-i}$ wegen Monotonie)
  - ∘ Nachweis
    - ▷ n Widerstände wie oben, alle exakt bis auf R_k
    - ▷ betrachtet wird B $= b_k 2^k$
    - ▷ Differenz zu nächstem Wert: 1
    - ▷ aufgrund Abweichungen in R_k entsteht Fehler, der einer Binärdifferenz $\triangle B$ entspricht
    - ▷ damit muß gelten $\triangle B \leq 1$

- ▷ ausgehend von Formel für U_{Ref}

$$\begin{aligned}\triangle U_A &= U_{Ref} * B - U_{Ref} * R * \textstyle\sum_{i=0}^{n-1} 1/R_i \\ &= U_{Ref} * \triangle B\end{aligned}$$

hieraus

$$\begin{aligned}\triangle B &= B - R * \textstyle\sum_{i=0}^{n-1} 1/R_i \\ &= B - R * \textstyle\sum_{i=0}^{n-1} 2^i/R + 2^k - R/R_k \\ &= 2^k - R/R_k\end{aligned}$$

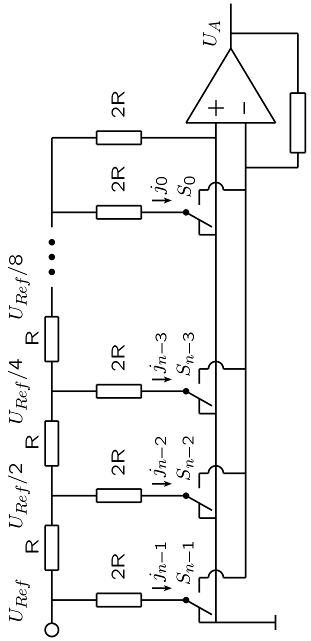
mit $R_k = R_k^s oll + \triangle R_k$

$$\begin{aligned}\triangle B &= 2^k - R/\left(R_k^s oll(1 + \triangle R_k / R_k^s oll)\right) \\ &\approx 2^k - 2^k * (1 - \triangle R_k / R_k) \\ &= 2^k * \triangle R_k / R_k\end{aligned}$$

wegen $\triangle B \leq 1$

$$\triangle R_k \leq 2^{-k} * R_k$$

- Besseres Verfahren
  - ⋆ R–2R–Widerstandsnetzwerke
  - ⋆ zwar 2∗n Widerstände, aber alle gleiche Größenordnung



Schalter S_i in Stellung für $b_i = 0$

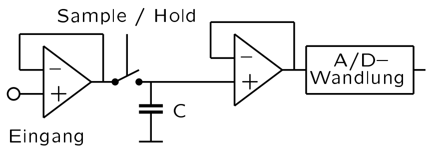
Realisierung eines DAU mit einem Widerstandsnetzwerk

3.3 Analog–/Digital–Umsetzer (ADU)

- analog/digital converter (ADC)
- elektrisches Signal wird zur Verarbeitung im Rechner in einen Digitalwert gewandelt
- verschiedene Verfahren mit unterschiedlicher Genauigkeit und Geschwindigkeit (etwa 10 nsec bis 50 msec)
- Verfahren
  - ⋆ Momentanwandlung
    - ∘ direkte (parallele) Wandlung (schnell)
    - ∘ Stufenumsetzer (mittelschnell)
    - ∘ Zählverfahren (langsam)
  - ⋆ integrierende Wandlung
    - ∘ "Dual slope"–Verfahren (langsam)
    - ∘ Spannungs–Frequenz–Umsetzung (langsam)

- Schema eines A–/D–Ablaufs mit "Sample–and–Hold"–Verstärkerschaltung
  - ⋆ Rechner gibt Startbefehl mit Adresse des Sensors
  - ⋆ Steuerung und Multiplexer schalten den verstärkten Sensorwert auf die Sample–Phase
  - ⋆ in der Sample–Phase wird ein Kondensator aufgeladen (Akquisitionszeit), ca 0.1 nsec bis 50 μsec bis gewünschte Genauigkeit (z.B. 0.01%) erreicht ist
  - ⋆ Nach Ablauf der Akquisitionszeit wird auf die Hold–Phase geschaltet und die A–/D–Wandlung beginnt
  - ⋆ die Hold–Phase hält den Analogwert (Spannung des Kondensators) während der Dauer der A/D–Wandlung fest (hochohmiger Verstärkereingang)
  - ⋆ Der A–/D–Wandler meldet das Ende der Umsetzung an die Steuerung
  - ⋆ Die Steuerung übergibt, evtl. mit Interrupt, den Digitalwert an den Rechner

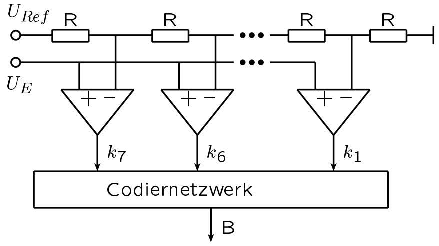
- "Sample–and–Hold"–Verstärkerschaltung



- Direkte (parallele) Wandlung
  - ⋆ sehr schnelles Verfahren (5 bis 100 nsec Wandlungszeit)
  - ⋆ aufwendig: für n Bits werden $m = 2^n - 1$ Vergleichselemente (Komparatoren) eingesetzt
  - ⋆ (z.B. n=8 → m=255)
  - ⋆ der Komparator i vergleicht die Eingangsspannung U_E mit einer Referenzspannung $U_{Ref}(i) = i * U_{Ref}/2^n$ und liefert das Ergebnis k_i:

    k_i := if ($U_E \geq U_{Ref}(i)$) then 1 else 0;
  - ⋆ die Ergebnisse werden in ein Codiernetzwerk eingespeist, das den Binärwert B für den Analogwert U_E ausgibt

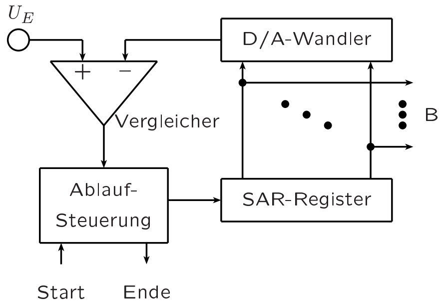
- ⋆ Beispiel für n=3
  - ∘ m=7
  - ∘ Schaltung



  - ∘ Tabelle für Codiernetzwerk

k_7	k_6	k_5	k_4	k_3	k_2	k_1	B
0	0	0	0	0	0	0	000
0	0	0	0	0	0	1	001
0	0	0	0	0	1	1	010
0	0	0	0	1	1	1	011
0	0	0	1	1	1	1	100
0	0	1	1	1	1	1	101
0	1	1	1	1	1	1	110
1	1	1	1	1	1	1	111

- Stufenumsetzer
  - ⋆ auch Wägecodierer
  - ⋆ sukzessive Approximation
  - ⋆ mittelschnelles Verfahren
  - ⋆ in n aufeinanderfolgenden Vergleichsschritten wird je 1 Bit des Ergebnisses bestimmt, beginnend bei MSB bis zu LSB
  - ⋆ Blockschaltbild



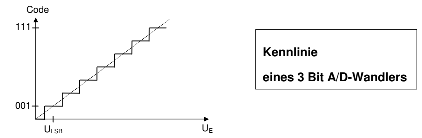
- ⋆ Ablauf
  - ∘ das SAR–Register (Sucessive Approximation Register) enthält einen Binärwert B
  - ∘ B wird mit 0 initialisiert
  - ∘ am Ende des Ablaufs enthält B den Digitalwert für die Analogspannung U_E
  - ∘ der jeweils eingestellte Wert B wird über einen DAU in einen Analogwert gewandelt und mit U_E verglichen
  - ∘ im SAR–Register (Sucessive Approimation Register) wird zu den bereits fixierten Bits das nächstniedere Bit gesetzt und bei negativem Vergleichsergebnis zurückgesetzt

- Zählverfahren
  - ⋆ Analogsignal U_E in Binärwert B umzuwandeln
  - ⋆ Zähler, dessen Wert Z über einen DAU in die Analogspannung U_Z gewandelt wird
  - ⋆ Zähler von 0 beginnend solange um 1 erhöhen, bis $U_Z \geq U_E$
  - ⋆ langsames Verfahren, z.B. bei n = 12 und 1 MHz Oszillator max. 4 msec, im Mittel 2 msec
  - ⋆ Variante: Folgeumsetzer (tracking) Zähler ist Vorwärts–/Rückwärtszähler, der vom Komparator richtungsgesteuert wird. Meßwert wird laufend verfolgt (Momentanwert in einem Register)

- Zweiflanken–Verfahren (dual slope)
  - ⋆ relativ langsam, aber genau
  - ⋆ viele Fehlerquellen, z.B. Temperaturabhängigkeit von R oder C, Netzbrumm werden eliminiert
  - ⋆ Prinzip: Aufladen und Entladen Kondensator
  - ⋆ Ablauf
    - ∘ Phase 1 Ein Kondensator wird in vorgegebener Zeit $\triangle t_i$ mit der zu messenden Spannung aufgeladen (integrierend, z.B. kann sich der Netzbrumm eliminieren, falls $\triangle t_i$ ein Vielfaches von 20 msec ist)
    - ∘ Phase 2 Entladen des Kondensators; ein Zeitmesser (Zähler) mißt die Entladezeit $\triangle t_E$; diese Zeit ist proportional zur Eingangsspannung

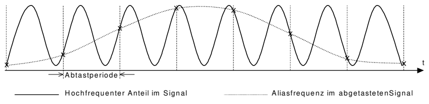
- Spannungs–/Frequenz–Umwandler
  - ⋆ Voltage/frequency converter (VFC)
  - ⋆ Idee
    - ∘ zu messende Spannung wird in eine dem Wert proportionale Frequenz umgewandelt
    - ∘ die Impulse in einem festen vorgegebenem Zeitintervall werden gezählt
    - ∘ der Inhalt des Zählers ist der gesuchte Binärwert
  - ⋆ auch hier integrierend gewandelt
  - ⋆ Verfahren ist schnell und genau, die Frequenzübertragung wenig störanfällig

- statische Fehler bei A/D-Wandlung



  - ⋆ Quantisierungsfehler: Durch die Auflösung von n Bits ergibt sich ein systematischer Quantisierungsfehler von $\pm 0.5 * 2^{-n}$.
  - ⋆ Nullpunktfehler: Kennlinie horizontal verschoben ist.
  - ⋆ Verstärkungsfehler: falschen Steigung Beide Fehlerarten können i.A. durch Eichung (z.B. mittels Wendelpotentiometer) hinreichend genau beseitigt werden.
  - ⋆ Linearitätsfehler: rechnergestützte Korrektur

- dynamische Fehler bei A/D-Wandlung



Aliasfrequenz bei Abtastung unter Verletzung des Abtasttheorems

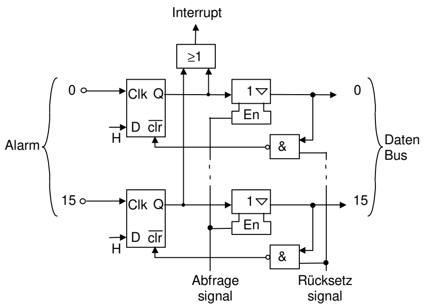
- ⋆ Abtasttheorem $f_s > 2 * f_{max}$
  Voraussetzung für die spätere Rekonstruktion des Orginalsignals
  Verletzung führt zu unerwünschten Aliasfrequenzen im abgetasteten Signal
  Abhilfe: Signal vor der Abtastung in seiner Bandbreite begrenzt (aktives Tiefpaßfilter)
- ⋆ zeitliche Unsicherheit Δt_a des Abtastzeitpunktes.
  Multipliziert mit der maximalen Steilheit des Signales ergibt Amplituden-Unsicherheit der Messung.
  Abhilfe: Sample- and Hold-Verstärkers zu einem definierten Zeitpunkt

3.4 Digital–Ein–/Ausgabe

- häufig nur binäre Signale (Kontakte, Schalter), jedoch auch Byte oder Wort
- binäre Signale werden meist in Gruppen (Worte) zusammengefaßt
- programmierbare Steuerlogik
  jede Leitung parametrisierbar
- Interrupterzeugung pro Bit, Byte, Gruppe
- Einschübe in Schaltschränken
- Rangierverteiler

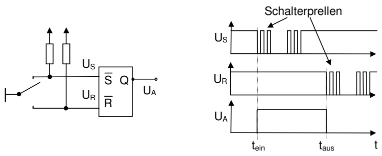
- Digitalausgabe
  - ⋆ Ausgabe von Zustandsinformation
  - ⋆ einfachste Variante der Prozeßperipherie
  - ⋆ Bits eines Registers an Schnittstelle
  - ⋆ Pegelwandlung, Verstärkung
  - ⋆ Ansteuerung von Schaltern, Relais, Anzeigen, Motoren, Schrittmotoren, Ventilen usw.
  - ⋆ Techniken
    - ◦ statische Ausgänge (Ausgaberegister)
    - ◦ dynamische Ausgänge
      Impulsausgänge mit (wählbarer) Pulsdauer
    - ◦ Impulsfolgen
      programmierbare Anzahl von Impulsen
    - ◦ Pulsweitenmodulation (variable Impulsbreite),
      Pulsfolgemodulation (variable Freqeunz)

- Digitaleingabe
  - ⋆ Schalter, Taster, Zustandsanzeigen (Kontaktstellungen, Grenzwertüberwacher), Zähler, Digitalmeßwerte (Binärcode), Strichcode usw.
  - ⋆ Eingangssignale auf Bits eines Registers, Schwelle
  - ⋆ statische Eingänge
    Momentanwert in einem Register
  - ⋆ gepufferte Eingänge
    Setzen durch externes Signal; der Rechner erhält beim Lesen nicht unbedingt den Momentanwert; Signal geht nicht verloren.
  - ⋆ dynamische Eingänge
    Nur Signaländerungen (Flankenübergänge) werden als "1" in einem Register gespeichert, evtl. mit Interruptmeldung. Die Hardware/Software muß die gelesenen Bits im Byte/Wort individuell zurücksetzen

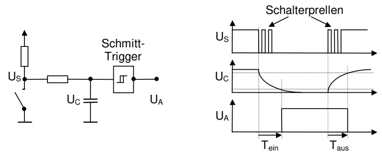


Alarm-Eingabe-Gruppe (aus: Färber)

- ⋆ Galvanische Trennung (Optokoppler, Trafo)
- ⋆ Entprellung



Entprellen durch ein RS-Flip Flop (aus: Färber)



Entprellen durch RC-Glied mit Schmitt-Trigger (aus: Färber)

3.5 Echtzeituhren

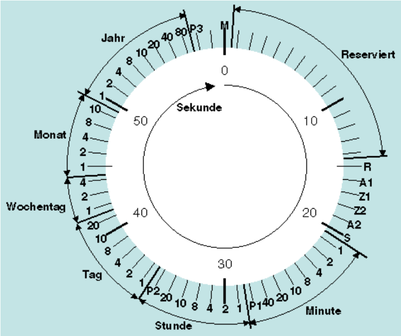
- Netzuhr (20 msec)
- Uhren mit programmierbarer hoher Auflösung (bis zu 1 μsec)
- kaskadierbare Zähler am Prozessortakt
- Aufgaben
  - ⋆ Absolutzeitgeber (batteriegepuffert)
    Datum, Uhrzeit,
    zeitabhängige Aufträge,
    Zeitstempel, Logbuch,
    Ursache–Wirkung–Feststellung
  - ⋆ Relativzeitgeber
    verzögern, messen von Zeitintervallen,
    zyklisch anstoßen, abtasten,
    Zeitüberwachung von Wartezuständen
- Synchronisierung der Uhren in Rechnernetzen nichttrivial, schwierig;
  evtl. nur Näherungen mit grober Toleranz

- Zeitcode-Geräte
  - ⋆ Funkuhr der Physikalisch Technischen Bundesanstalt
    - ◦ Zeitinformation: Minute, Stunde, Tag im Jahr, Wochentag (1 bis 7), Jahr (00 bis 99)
    - ◦ jede Dezimalziffer wird für sich codiert (4-Bit-Code); bei eingeschränktem Ziffernbereich weniger als 4 Bit

Minute	3+4=7
Stunde	2+4=6
Tag im Jahr	2+4+4=10
Wochentag	3
Jahr	4+4=8
Summe	34

- ◦ Zeitinformation benötigt also 34 Bit; plus ein Startbit und drei Prüfbits
- ◦ die 3 Prüfbits ergänzen Minute, Stunde, Jahr je auf gerade Anzahl von "1"
- ◦ jede Sekunde ein Bit der Zeitinformation gesendet, Zeitinformation in 38 Sekunden einer Minute

- Sekunde nicht in Zeitinformation, da Sekunde beim Empfänger mitgezählt werden kann
- in der 60. Sekunde keine Information übertragen, dies ist Zeichen für Nullstellen des Zählers
- laufend 77.5 KHz–Signal
- Impulsmodulation (0.2 sec hohe Amplitude je sec ist "1", 0.1 sec niedere Amplitude je sec ist "0")



⋆ Zeitcode–Geräte am Bus

- internationale InteRange Instrumentation Group
- IRIG–A (0.1 sec), IRIG–B (1 sec), IRIG–G (0.01 sec)
- Beispiel IRIG–B
  - ▷ 1 KHz Trägerfrequenz
  - ▷ amplitudenmoduliert in 100 Pulse à 10 Signale pro sec
  - ▷ "0" ≡ 2 hohe + 8 niedere Impulse
  - ▷ "1" ≡ 5 hohe + 5 niedere Impulse
  - ▷ "Positions–Identifier" ≡ 8 hohe + 2 niedere Impulse (zwei Positions–Identifier jede sec am Rahmenanfang)

3.6 Mikroperipherik

- Neue Materialtechniken erlauben den Einsatz von neuen physikalischen Effekten in der Sensorik und Aktorik. Vor allem die Kombination mit Halbleitern ermöglicht die Miniaturisierung und Integration der Verarbeitungsschaltung auf Chip-Ebene.
- Definition VDE/VDI für intelligenten Mikrosensor

  Sensorelement und Signalverarbeitung sind in einem miniaturisierten Gehäuse.

  Das Ausgangssignal ist weitgehend frei von Ungenauigkeiten, die im Meßprinzip oder der Fertigung begründet sind.

  Mikrosensoren zum Messen von Kraft, Druck, Beschleunigung, Lage, Temperatur, Feuchte, el. und magn. Felder

  Rezeptoren für chemische Stoffe

- physikalische Konzepte für Mikroaktoren
  - ⋆ Piezo–Aktoren
    Verformung von Kristallen durch elektrisches Feld; feinste Hubbewegungen mit großen Kräften bei kurzen Stellzeiten (msec)
  - ⋆ Magnetostriktive Aktoren
    Eisenlegierungen und Seltene Erden; in magnetischen Feldern Ausdehnung oder Kontraktion
  - ⋆ Elektrorheologische Flüssigkeiten
    Suspensionen ändern im elektrischen Feld im sec–Bereich ihre Viskosität (erstarren bzw. werden dünnflüssig); Einsatz z.B. Kraftkopplungen, Stoßdämpfer
  - ⋆ Chemische Aktoren
    Gasentwicklung beim Anlegen von Gleichspannungen erzeugt Druck; Umpolen baut Druck ab (Einsatz z.B. Lüftungsklappen, Dosierung)
  - ⋆ Memory–Metalle
    Sie ändern bei bestimmter Temperatur ihre Form in genau vorhersagbarer Weise (Einsatz z.B. beim Brandschutz); keine Hilfsenergie nötig

- Anwendungen Mikroperipherik
  Prozeßautomatisierung,
  Medizin,
  Kfz,
  Haushalt,
  Umwelt usw.
- Probleme:
  - ⋆ Montage
    (Halbleiter sind empfindlich gegen mechanische Spannungen)
  - ⋆ Umweltstabilität
    (vor allem temperatur– und feuchtigkeitsempfindlich)
  - ⋆ Lebensdauer

3.7 Identifikationssysteme

- Identifikation ist Erkennen individueller Objekte anhand von Codierungen
- Ziele
  - ⋆ eindeutige Zuordnungen
  - ⋆ Produktverfolgung
  - ⋆ Betriebsdatenerfassung
  - ⋆ permanente Übersicht (Inventur).
- Verfahren
  - ⋆ mechanisch: Abtasten, Kontakte mit Spulen
  - ⋆ Codierleisten: Lesen mit Reflex–Lichtschranken
  - ⋆ Strichcodes, Bar–Codes
    (auch matrixartige 2-dimensionale "dot codes")
    z.B. 16–Bit Information in 10 msec
  - ⋆ Auslesen von elektronischen Datenträgern mit Hochfrequenz
    - Radio frequency ID (RF–ID)
    - z.B. 125 KHz

- zum Auslesen mit Hochfrequenz
  - ⋆ Arten
    - passive Datenträger
      - ▷ ohne eigene Energiequelle
      - ▷ Energie durch Strahlung des Lesers
      - ▷ geben in ROM oder EPROM gespeicherte Information (meist unter 128 Bit) über Antenne zurück
    - aktive ID–Systeme
      - ▷ mit Hilfsenergie (z.B. Batterie)
      - ▷ auslesen, speichern, löschen oder ändern größerer Datenmengen
      - ▷ EEPROM mit bis zu 32 KBit Speicher

3.8 Industrieterminals

- Mensch/Maschine-Schnittstelle
- BDE-Terminals (*Betriebsdatenerfassung*)
- LCD-Displays, CRT-Displays
  gegen Staub, Flüssigkeiten, Erschütterungen geschützt
- große Tastaturen

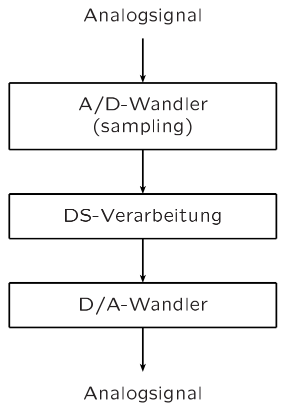
3.9 Spezielle Prozessoren

- Schnittstellen für Aktoren und Sensoren meist im Prozeßrechner(-chip) integriert, z.B.
  - ⋆ Digital-Ausgänge
  - ⋆ Digital-Eingänge
  - ⋆ Analog-Digital-Wandler (A/D-Wandler)
  - ⋆ Digital-Analog-Wandler (D/A-Wandler)
  - ⋆ Zähler, Uhren
- Einsatz von Zusatzprozessoren (ASIC–Chips), z.B. für
  - ⋆ Gleitpunktarithmetik
  - ⋆ Signalverarbeitung (digital signal processors, DSP)
  - ⋆ Fast–Fourier–Transformation (FFT)
  - ⋆ Fuzzy-logic Regelung
  - ⋆ Mustererkennung (u.a. Transputer–Netzwerke)
  - ⋆ Feldbus–Protokolle
- für Leistungsansteuerung von Geräten und komplexere Steuerungsaufgaben wird Speicherprogrammierbare Steuerung (SPS) zwischen Prozeßrechner und Gerät eingesetzt

4 Digital Signal Processor (DSP)

4.1 Grundablauf, Motivation

- Digitale Signalverarbeitung in drei Schritten



- Warum digital und nicht analog?
  - ⋆ Funktionen eventuell nicht oder schwer in analogen Bausteinen implementierbar
  - ⋆ keine Abhängigkeiten (und damit Veränderungen der Funktion) durch Fertigungstoleranzen der Bauelemente bei analogen Schaltungen
  - ⋆ Reproduzierbarkeit
  - ⋆ Stabilität
  - ⋆ keine Alterung und Temperaturabhängigkeit
  - ⋆ Hardware weitgehend anwendungsunabhängig
  - ⋆ programmierbar
  - ⋆ durch Fernladen des Programms ist Gerät mit DSP aktualisierbar, z.B. bei Modems wichtige Eigenschaft
  - ⋆ Software kann lernen und sich an Einsatzumgebung anpassen (adaptive Algorithmen)
  - ⋆ Entwicklungszeit kürzer bei Programmierung als bei Neuentwicklung Hardware

- Anwendungsbereiche
  - ⋆ Aufbereitung von Signalen (Filter, Echo, Störungen)
  - ⋆ Modulatio/Demodulation bei der Signalübertragung
  - ⋆ Kompression/Dekompression
  - ⋆ Mustererkennung
  - ⋆ Erzeugung synthetischer Signale
  - ⋆ komplexe Steuer- und Regelaufgaben
- Geräte
  - ⋆ DVD, CD, MPEG-Player
  - ⋆ digitales Radio, digitales Fernsehen
  - ⋆ Modems
  - ⋆ GSM-Mobiltelefon
  - ⋆ Sprachsynthesizer
  - ⋆ Objekterkennung
  - ⋆ intelligente Sensoren
- Markt
  - ⋆ weltweit 831 Millionen Chips im Jahr 2000
  - ⋆ durchschnittlicher Verkaufspreis 7 Dollar
  - ⋆ Marktführer: Texas Instruments (48 %)

4.2 Eigenschaften am Beispiel TMS320C30

- Grunddaten

cycle Time	60 ns
DMA	1
frequency (MHz)	33
MIPS	16.667
MOPS	183.33
RAM (Words)	2Kx32Bit
ROM (Words)	4Kx32Bit
cache	64 words
off-chip memory	16Mx32
serial ports	2
timers	2
addresses	24 Bit
word	32 Bit

- Wichtigste Eigenschaft der DSP-Bausteine
  - ⋆ Multiplikation (int oder float) in einem Takt in der Multipliziereinheit
  - ⋆ Parallel zur Multiplikation im gleichen Takt ALU-Operation möglich
  - ⋆ MAC-Operation: multiply accumulate
  - ⋆ Abfrage der Schleifenbedingung im Taktzyklus: zero-overhead looping
  - ⋆ Anwendung: oft zu berechnen

$$S = \sum_i a_i * b_i$$

  - ⋆ Vorgehen mit Pipelining in Schleifen: Berechnung einer Multiplikation parallel zur Addition des Ergebnisses der Multiplikation im vorherigen Schritt zu einer Summe

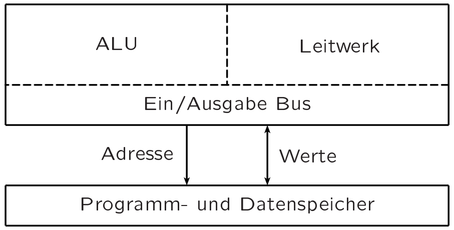
- Beispiel

$$S = \sum_{i=1}^{N} a_i * b_i$$

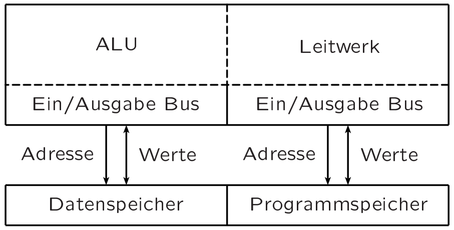
```
⋮
LDI   @ADDR1,AR0                  ; AR0 points to array a(i)
LDI   @ADDR2,AR1                  ; AR1 points to array b(i)
LDF   0.0,R0                      ; Initialize R0
MPYF3 *AR0++(1),*AR1++(1),R1      ; Compute first product

RPTS  511                         ; Repeat 512 times
MPYF3 *AR0++(1),*AR1++(1),R1      ; Compute next product
|| ADDF3 R1,R0,R0                 ; and accumulate the
                                  ; previous one

ADDF  R1,R0                       ; One final addition
⋮
```

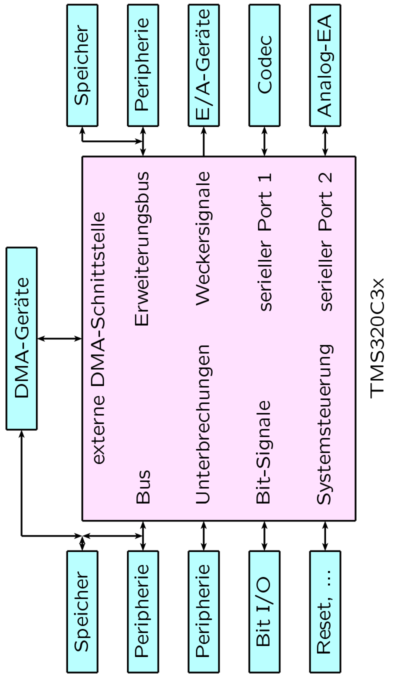

4.3 Architektur und Programmier-Schnittstelle

- keine von-Neumann-Architektur

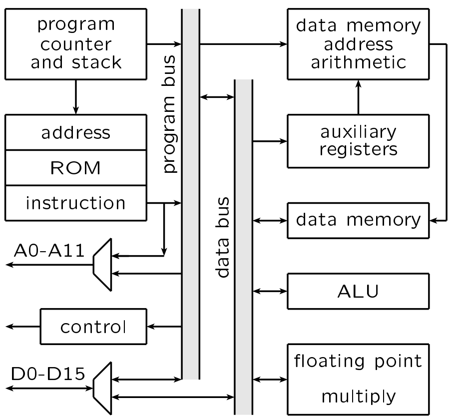


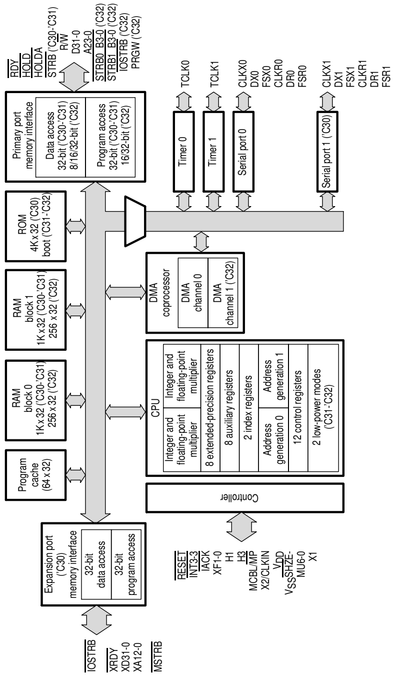
- sondern Harvard-Architektur



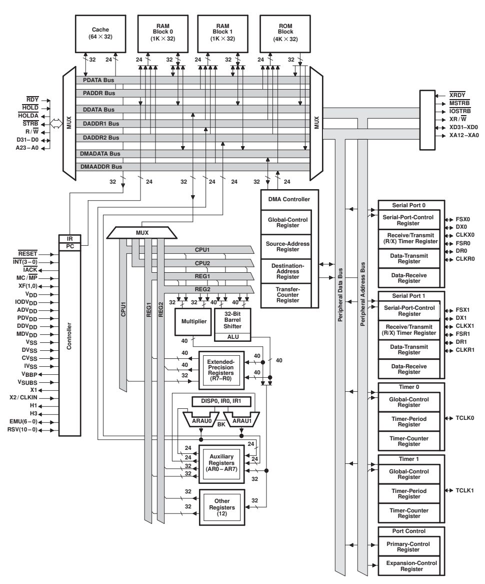


- Blockdiagramm





TMS320C30 functional block diagram



- Pipelining (4 Stufen)
  - ⋆ holen Befehl
  - ⋆ dekodieren Befehl und Adreßrechnung
  - ⋆ lesen Operanden vom Speicher
  - ⋆ parallele Ausführungen:
    - ◦ Operationen in ALU
    - ◦ Operationen in Multipliziereinheit
    - ◦ lesen von/schreiben zu Register
    - ◦ schreiben der Ergebnisse der vorhergehenden Operation in den Speicher, falls notwendig
- ALU (arithmetic and logical unit)
  - ⋆ Datenkonversion
  - ⋆ Shift
  - ⋆ Addition, Subtraktion
  - ⋆ Logische Operationen
- ARAU (auxiliary register arithmetic unit)
  - ⋆ 2 ARAUs
  - ⋆ können zusammen zwei Adressen in einem Zyklus erzeugen (Adreßberechnung)

- Register
  - ⋆ 8 allg. Register R0-R7 mit 40 Bit (interne höhere Präzision)
  - ⋆ 8 Hilfsregister AR0-AR7 zur Generierung von Adressen, Zugriff von CPU und ARAUs, modifiziert durch Multipliziereinheit und ALU
  - ⋆ 2 Indexregister IR0, IR1
  - ⋆ 9 Spezialregister, darunter Stackadreßregister SP, Repeat-Counter RC, Data-Page-Pointer DP und Statusregister ST
  - ⋆ Befehlszähler PC
  - ⋆ Befehlsregister IR
- Speicheradressierungsarten
  - ⋆ Operand im Register (Registeradressierung)
  - ⋆ Operand im Befehl (short/16 Bit or long/24 Bit immediate)
  - ⋆ direkt: 24-Bit-Adresse durch Konkatenation 8 LSB aus Register DP (Data-Page-Pointer) mit 16 Bit Speicheradresse im Befehl
  - ⋆ indirekt: Adresse steht in Hilfsregister, optionales Displacement im Befehl direkt oder im Indexregister

- ◦ 4 Typen charakterisiert durch 2 aufeinanderfolgende Schritte bei der Adreßberechnung, wobei K und L unabhängig voneinander 0 oder 1 sein können:
  1. Schritt: addr = cont(AR) $\pm$ disp $*$ K
  2. Schritt: AR = cont(AR) $\pm$ disp $*$ L
- ⋆ relativ zu PC (nur bei Sprüngen)
- ⋆ in Kombination mit:
  - ◦ Bit-Reversed Adressing für FFT (fast fourier transform)
  - ◦ Circular-Adressing bei Ringpuffer, Zirkuläre Addition des Displacements bei indirekter Adressierung
- ⋆ Einschränkungen der Adressierungsarten bei parallel auszuführenden Befehlen, bei 3-Adreßbefehlen und bei Sprungbefehlen, damit Parallelarbeit in HW gewährleistet bleibt

- Befehl RPTB (repeat block of instructions)

Beispiel Maximum von N = 147 Zahlen

```
        :
        LDI    146,RC        ; Initialize repeat counter to 147±1
        LDI    @ADDR,AR0     ; AR0 points to beginning of array
        LD     *AR0++(1),R0  ; Initialize MAX to the first value
        RPTB   LOOP
        CMPF   *AR0++(1),R0  ; Compare number to the maximum
LOOP    LDFLT  *AR0(1),R0    ; If greater, this is a new maximum
        :
```

- Befehl RPTS (repeat single instruction),
  Repeatcounter als Register vorhanden
  Beispiel Blockweises Laden vom externen Speicher

```
extern  .word  01000H
block1  .word  0809C00H
:
        LDI   @extern,AR0   ; Source address
        LDI   @block1,AR1   ; Destination address
        LDF   *AR0++,R0     ; Load the first number
        RPTS  510           ; Repeat following instruction 511 times
        LDF   *AR0++,R0     ; Load the next number, and
||      STF   R0,*AR1++     ; store the previous one
        STF   R0,*AR1       ; Store the last number
:
```

- Alternativ: Laden über DMA-Coprozessor parallel zu CPU-Arbeit

- Synchronisationsbefehle (Interlock)
  - ⋆ LDII Load Integer Interlocked, setzt Interlock und liest
  - ⋆ STII Store Integer Interlocked, schreibt und loescht Interlock
  - ⋆ Interlock über I-Ausgang und I-Eingang des Chips geführt, damit auch Synchronisation mit anderen Komponenten des Systems möglich.
    Einfachster Fall: I-Ausgang zu I-Eingang
  - ⋆ Wirkung:
    - ◦ I-Ausgang wird gesetzt, während Interlock gesetzt
    - ◦ lesen bei LDII erfolgt erst, wenn I-Eingang (von externen Komponenten) gesetzt

- Beispiel Synchronisation (Busy-Waiting Loop)
  Synchronisationsvariable SEMA (binärer Semaphor)

```
    LDI   1,R0        ; Put 1 into R0
L1: LDII  @SEMA,R1    ; Interlocked operation begun
                      ; Contents of SEMA → R1
    STII  R0,@SEMA    ; Put R0 (= 1) into SEMA
                      ; Interlocked operation ended
    BNZ   L1          ; Keep trying until SEMA = 0
```


4.4 Anhang: Ausgewählte Befehle

- Load and Store Instructions

LDF	Load floating-point value
LDI	Load integer
LDP	Load data page pointer
STF	Store floating-point value
STI	Store integer

- 2-Operand Instructions (Auswahl)
  The 'C3x supports 35 2-operand arithmetic and logical instructions. The two operands are the source and destination. The source operand can be a memory word, a register, or a part of the instruction word. The destination operand is always a register.

ABSF	Absolute value of a floating-point number
ABSI	Absolute value of an integer
ADDC	Add integers with carry
ADDF	Add floating-point values
ADDI	Add integers
CMPF	Compare floating-point values
CMPI	Compare integers

FLOAT	Convert integer to floating-point value
MPYF	Multiply floating-point values
MPYI	Multiply integers
NEGB	Negate integer with borrow
NEGF	Negate floating-point value
NEGI	Negate integer
NOT	Bitwise-logical complement
OR	Bitwise-logical OR
RND	Round floating-point value
ROL	Rotate left
ROLC	Rotate left through carry
SUBB	Subtract integers with borrow
SUBC	Subtract integers conditionally
SUBF	Subtract floating-point values
SUBI	Subtract integer

- 3-Operand Instructions (Auswahl)
  Whereas 2-operand instructions have a single source operand (or shift count) and a destination operand, 3-operand instructions can have two source operands (or one source operand and a count operand) and a

destination operand. A source operand can be a memory word or a register. The destination of a 3-operand instruction is always a register.

ADDC3	Add with carry
ADDF3	Add floating-point values
ADDI3	Add integers
AND3	Bitwise-logical AND
ANDN3	Bitwise-logical AND with complement
ASH3	Arithmetic shift
CMPF3	Compare floating-point values
CMPI3	Compare integers
LSH3	Logical shift
MPYF3	Multiply floating-point values
MPYI3	Multiply integers
OR3	Bitwise-logical OR
SUBB3	Subtract integers with borrow
SUBF3	Subtract floating-point values
SUBI3	Subtract integers
TSTB3	Test bit fields
XOR3	Bitwise-exclusive OR

- Program-Control Instructions

Bcond	Branch conditionally (standard)
BcondD	Branch conditionally (delayed)
BR	Branch unconditionally (standard)
BRD	Branch unconditionally (delayed)
CALL	Call subroutine
CALLcond	Call subroutine conditionally
DBcond	Decrement and branch conditionally (standard)
DBcondD	Decrement and branch conditionally (delayed)
IACK	Interrupt acknowledge
IDLE	Idle until interrupt
NOP	No operation
RETIcond	Return from interrupt conditionally
RETScond	Return from subroutine conditionally
RPTB	Repeat block of instructions
RPTS	Repeat single instruction
SWI	Software interrupt
TRAPcond	Trap conditionally

- Parallel Instructions (Auswahl)

ABSF || STF	Absolute value of a floating-point number and store floating-point value
ADDI3 || STI	Add integers and store integer
AND3 || STI	Bitwise-logical AND and store integer
FIX || STI	Convert floating-point to integer and store integer
LDF || STF	Load floating-point value and store floating-point value
MPYF3 || STF	Multiply floating-point values and store floating-point value
MPYI3 || STI	Multiply integer and store integer
NEGF || STF	Negate floating-point value and store floating-point value

SUBI3 || STI	Subtract integer and store integer
XOR3 || STI	Bitwise-exclusive OR values and store integer
LDF || LDF	Load floating-point value
LDI || LDI	Load integer
MPYF3 || ADDF3	Multiply and add floating-point value
MPYF3 || SUBF3	Multiply and subtract floating-point value
MPYI3 || ADDI3	Multiply and add integer
MPYI3 || SUBI3	Multiply and subtract integer

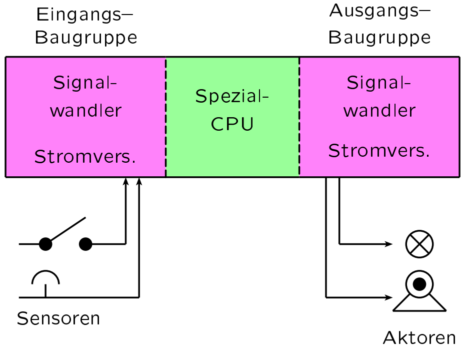
5 Speicherprogrammierbare Steuerungen (SPS)

5.1 Hardware

- englisch
  - ⋆ programmable logic control (PLC)
  - ⋆ programmable machine control (PMC)
- Einsatz für schrittweise Abläufe
  - ⋆ Ersatz Relaistechnik
  - ⋆ zeitprogrammierte Schritte
  - ⋆ Maschinen–Steuerungen
  - ⋆ einfache Regelung
- SPS sind Spezialrechner mit integrierten Peripherie–Anschlüssen
- Einschubkarten für PC
- zunehmend auch Soft-SPS im PC
- geringe Kosten (kleine SPS einige 100 DM)
- Aufbau für rauhe Umgebungen

- Vorteil: riesige Marktpalette mit vielfältigen Ein– und Ausgabe–Baugruppen, z.B.
  - ⋆ Digital- und Analog-E/A
  - ⋆ Zähler
  - ⋆ Schrittmotorsteuerung
  - ⋆ Regler
  - ⋆ Busanschluß
  - ⋆ verschiedenste Signalpegel
  - ⋆ Potentialtrennung
  - ⋆ Entstörung
  - ⋆ Kurzschlußschutz
- Beispiel: Produktpalette Simatic S7

- Blockschaltbild



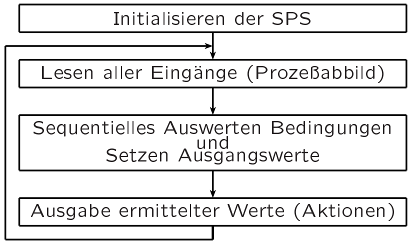
Sensoren: Schalter, Relais, Signale
Aktoren: Anzeigen, Schalter, Schrittmotoren

- Beispiel für einfache Ausführung
  - ⋆ nur Bitoperationen (AND, OR, NOT, setze/lösche Bit)
  - ⋆ 40 Merker (Speicher für Zwischenergebnisse)

  - ⋆ 8 Ein– und 8 Ausgänge (verschiedene Pegel)
  - ⋆ Notsperre aller Ausgänge
- Beispiel für große SPS
  - ⋆ Wortrechner
  - ⋆ 512 KB Speicher
  - ⋆ 16000 Merker
  - ⋆ 512 Zähler, 512 Wecker
  - ⋆ A/D– und D/A–Wandler
  - ⋆ Anschluß für Terminal, PC oder Feldbus
- SPS z.T. auch mit mächtigeren Funktionen, wie GP-Arithmetik (ADD, MUL, DIV), Vergleich, Unterprogrammtechnik
- Einsatz der SPS auf der Feldebene der Prozeßsteuerung
- Vernetze SPS zur Automatisierung komplexer Großanlagen
  hauptsächlich chemische Verfahrenstechnik

5.2 Betriebsweise einer SPS

- Zyklus
  - ⋆ Zyklische Abarbeitung einer Sequenz von Anweisungen
  - ⋆ Anweisung der Form (Bedingung ⟶ Aktion)
  - ⋆ Schema



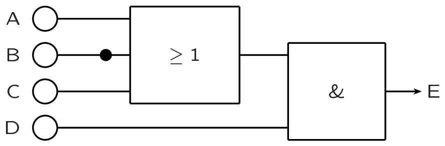
  - ⋆ Zyklusdauer
    - ∘ abhängig vom CPU–Chip
    - ∘ von Verzögerungen beim Erfassen Eingangswerte
    - ∘ von Anzahl Bedingungen
    - ∘ von Verzögerung bei Ausgabe
    - ∘ typische Zyklusdauer: einige msec bis sec

5.3 Programmierung

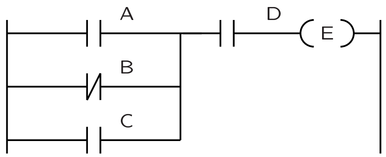
- meist in graphischer benutzerorientierter Darstellung auf PC
- Programme werden durch Ladegeräte, EPROM, über PC–Anschluß oder "download" über Feldbus geladen
- Internationale Norm IEC 1131
  - ⋆ Part 1: general information;
  - ⋆ Part 2: hardware requirements;
  - ⋆ Part 3: programming languages;
  - ⋆ Part 4: user guidelines;
  - ⋆ Part 5: communication.
- In Teil 3 der Norm werden die Sprachen standardisiert.
- In IEC 1131-5 (in Entwicklung) wird Kommunikation mittels "channels" zwischen SPS-Steuerungen definiert

- Sprachen
  - ⋆ AWL Anweisungsliste (DIN 1923)
    IL instruction list (IEC 1131-3)
  - ⋆ KOP Kontaktplan (DIN 1923)
    LD ladder diagram (IEC 1131-3)
  - ⋆ FUP Funktionsplan (DIN 40719)
    FBD function block diagram (IEC 1131-3)
  - ⋆ ST Strukturierter Text (IEC 1131-3)
    structured text
    Pascal-ähnlich
  - ⋆ AS Ablaufsprache (IEC 1131-3)
    SFC Sequential Funktion Chart
    Grafische Sprache für Ablauf von Schrittketten
  - ⋆ Transitionsnetze
  - ⋆ Regelsysteme
  - ⋆ Synchrone Sprachen

- Sprachbeispiele
  - ⋆ Pascal: E:= (A or (not B) or C) and D
  - ⋆ FUP:



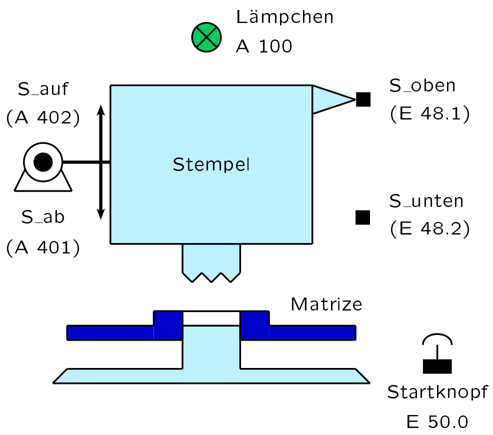
  - ⋆ KOP:



  - ⋆ AWL

```
: U(
: O   E1.1
: ON  E1.2
: O   E1.3
: )
: U   E1.4
: =   A1.0
```

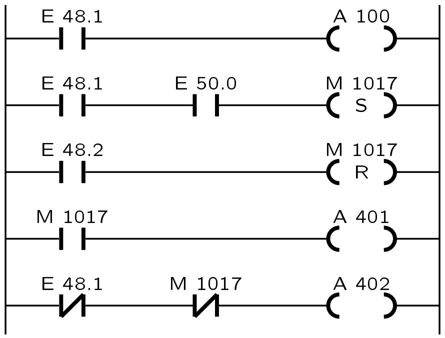

- Anwendungsbeispiel: Eine Presse

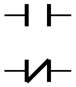


- C-Code für die Presse

```
#define PresseOben E48.1
#define PresseUnten E48.2
#define MotorAufw A402
#define MotorAbw A401
#define LampeGrün A100
#define Start E50.0
#define Merker1 M1017

if (PresseOben)
  then ein(LampeGrün);
if (PresseOben and Start)
  then Merker1 := true;
if (PresseUnten)
  then Merker1 := false;
if (Merker1)
  then ein(MotorAbw);
    /* Presse nach unten */
if ((not PresseOben) and (not Merker1))
  then ein(MotorAufw);
    /* Presse nach oben */
```


- Kontaktplan für die Presse





Schließer

Öffner

S Setzen

R Rücksetzen

- Anweisungsliste für die Presse

```
---
   U   E48.1
   =   A100   -- A100 := E48.1
---
   U   E48.1
   U   E50.0  -- if E48.1 and E50.0
   S   M1017  -- then M1017 := 1
---
   U   E48.2  -- if E48.2
   R   M1017  -- then M1017 :=0;
---
   U   M1017
   =   A401   -- A401 := M1017
---
   UN  E48.1  -- if not E48.1
   UN  M1017  -- and not M1017
   =   A402   -- then A402 := 1
---
```

- Demonstration IEC 1131-3
  mit CoDeSys am PC

6 Feldbusprotokolle

6.1 Signalübertragung

- Charakteristika:
  - ⋆ maximale Übertragungsrate in Mbps
  - ⋆ Dämpfung in db pro km
  - ⋆ überbrückbare Entfernung
  - ⋆ Materialeigenschaften (z.B. für Installation)
  - ⋆ Störungsempfindlichkeit
  - ⋆ Kosten, Marktproduktpalette

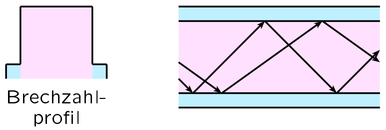
6.1.1 Übertragungsmedien

- Einfache (symmetrische) Kupferkabel
  - ⋆ zwei– oder vieradrig
  - ⋆ Gefahr des Übersprechens (Abhilfe evtl. Abschirmung)

- Verdrillte Kabel (twisted pair)
  - ⋆ zwei verdrillte Doppeladern mit 20 bis 30 Umschlingungen pro m
  - ⋆ gleichen viele induktiven Störungen aus
  - ⋆ mit Abschirmung STP (shielded TP, teuer), sonst UTP (unshielded TP)
  - ⋆ Übertragungsraten von kbps im km–Bereich bis zu 100 Mbps im Meterbereich
  - ⋆ Anwendung: strukturierte LAN-Verkabelung
- Koaxialkabel
  - ⋆ Konzentrischer Signalleiter im Kabelinnern, dann Dielektrikum, Abschirmung, Isolationsmaterial
  - ⋆ wenig Abstrahlung
  - ⋆ wenig Störungsanfällig
  - ⋆ Dämpfung nimmt mit steigender Frequenz rasch zu
  - ⋆ Anwendung: Ethernet LAN's (50 Ω)
    - ∘ nur ein gemeinsamer Kanal
    - ∘ 10 Mbps auf 500 m
    - ∘ größere Entfernungen durch Verstärkung (Repeater)

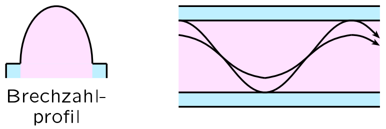
    - ∘ Stationsanschluß durch "Transceiver" (Tap– oder Schraubanschluß)
  - ⋆ Breitbandübertragung (75 Ω)
    - ∘ mehrere Frequenzbänder (Kanäle)
    - ∘ z.B. MAP im Bereich um 192 MHz mit 3 Kanälen à 12 MHz Breite
- Lichtwellenleiter
  - ⋆ Übertragung in Glasfasern (Kern mit 10 bis 100 μm Dicke und Glasmantel)
  - ⋆ noch selten Plastikfasern
  - ⋆ monochromatisches Licht
  - ⋆ unempfindlich gegen EMS und Übersprechen
  - ⋆ schwer abhörbar
  - ⋆ geringe Dämpfung und wenig Verzerrung
  - ⋆ geringes Gewicht
  - ⋆ übliche Wellenlängen im Bereich minimaler Dämpfung, z.B. 850 nm, 1300 nm und 1550 nm

* Arten
  * Multimode mit Stufenprofil
    * Glaskern mit höherem Brechungsindex wie Glasmantel
    * Lichtführung: Totalreflexion



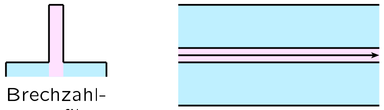
Brechzahl-
profil

  * Gradientenfaser
    * Glaskern mit kontinuierlich sich ändernden Brechungsindex
    * Lichtführung: Brechung



Brechzahl-
profil

* Monomode-Fasern
  * LAN und Post
  * kleinster Glaskern mit höherem Brechungsindex wie Glasmantel
  * Lichtführung: Wellenführung



Brechzahl-
profil

* Charakteristika

	Stufenprofil
Lichtquelle	LED oder Laser
Bandbreite	≤200 MHz/km
Kosten	billig
Datenrate	≈ 100 Mbps
Dämpfung	0.2–50 db/km
Dispersion	50 nsec/km
Kern∅	50–200μm
	Gradientenprofil
Lichtquelle	LED oder Laser
Bandbreite	200 MHz/km – 3 GHz/km
Kosten	mittel
Datenrate	≈ 300 Mbps und ≤ 20 km
Dämpfung	0.2–15 db/km
Dispersion	1 nsec/km
Kern∅	50–100μm
	Monomode
Lichtquelle	Laser
Bandbreite	3GHz/km – 50 GHz/km
Kosten	höher, aber Massenmarkt
Datenrate	≤ 2 Gbps und ≤ 80 km
Dämpfung	0.2–2db/km
Dispersion	0.3 nsec/km
Kern∅	2–10 μm

6.1.2 Elektromagnetische Störungen (EMS)

* Analogsignale von Sensoren oft im mV–Bereich
* Störspannungen durch
  * thermisches Rauschen
  * kapazitive und/oder induktive Einkopplungen (durch Nachbarkanal, Schaltvorgänge, Hochfrequenz)
  * schlechte Erdung
  * schlechte Kontakte (erzeugen mV Spannungsabfall)
* Abhilfen durch
  * kurze Kabelwege (Kontaktmatrix, Bilden von Unterstationen mit hochwertigem Kabel zum Rechner)
  * Potentialtrennung (Optokoppler, transformatorisch)
  * Schutz vor Immissionen (Wärme, Kälte, Feuchtigkeit, Staub)

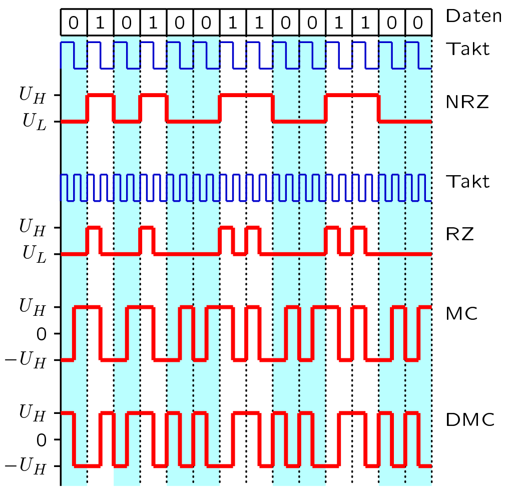
6.1.3 Signalcodierungen

* Ziele
  * eindeutige Darstellung der Zustände "0" und "1" und deren Folgen durch elektrische Signale
  * Störungsunempfindlich
  * Möglichkeit der Erkennung von Fehlern
  * möglichst geringer Gleichstromanteil
  * bei synchroner Übertragung keine eigene Taktleitung, sondern Ableitung des Sendetaktes aus den empfangenen Signalen (Taktrückgewinnung, selbsttaktend)
* Grundverfahren
  * Basisbandübertragung
  * modulierte Übertragung mit Trägerfrequenz
* Basisbandübertragung
  * Die gesamte Bandbreite des Wegs wird für die Übertragung eines Signals benutzt
  * Es wird eine Impulsfolge ohne Modulation mit einer Trägerfrequenz übertragen
  * Problem: Impulsverbreiterung durch frequenzabhängige Ausbreitungsgeschwindigkeit und Dämpfung

* modulierte Übertragung mit Trägerfrequenz
  * zur Übertragung wird das Signals mit einer Trägerfraquenz moduliert
  * gesendet wird $U(t) = A * cos(\omega t + \varphi)$
  * Trägerfrequenz ω
  * drei Varianten der Modulation
    * Amplitudenmodulation (amplitude shift keying ASK) z.B. A = 0 (1) bedeutet Binärzeichen 0 (1)
    * Frequenzmodulation (frequency shift keying FSK) z.B. ω = 1180 Hz (980 Hz) bedeutet Binärzeichen 0 (1)
    * Phasenmodulation (phase shift keying PSK) z.B. keine Änderung von φ bedeutet Binärzeichen 0, ein Phasensprung um 180 Grad bedeutet 1
  * mehrere Kanäle, deren Frequenzbänder sich nicht überlappen, können gleichzeitig über ein Medium übertragen werden

* Impulsdarstellung eines Signals mit Takt △ t:
  * einstufige Übertragungsverfahren Codeelemente 0 und 1
    * unipolar: 0 (1) durch 0 Volt (+U Volt)
    * polar: 0 (1) durch –U Volt (+U Volt)
    * bipolar: 1 durch abwechselnd +U, –U Volt
  * Mehrstufige Übertragungsverfahren Codeelemente haben n >2 Kennzustände (n=3 ternär, n=4 quaternär)
  * Baudrate = Anzahl Signale (Takte) je sec
  * Bitrate = Anzahl übertragene Bits je sec
  * bei einstufigen Verfahren: Bitrate = Baudrate
* Übliche Codierungen
  * NRZ (non return to zero)
    * 1: U_H in der Taktlänge
    * 0: U_L in der Taktlänge
    * Problem: Taktrückgewinnung bei langen Folgen gleicher Zeichen

* RZ (return to zero)
  * 1: U_H in erster Takthälfte, dann U_L
  * 0: U_L in der Taktlänge
  * Problem: Taktrückgewinnung bei langen Folgen gleicher Zeichen
* Manchester Code MC
  * Biphase–Code (mind. 1 Übergang je Takt)
  * 1: Übergang in Taktmitte von U_H nach U_L
  * 0: Übergang in Taktmitte von U_L nach U_H
  * Vorteile: kein Gleichstromanteil, selbsttaktend
  * Nachteil: doppelte Taktfrequenz
* Differential Manchester Code DMC
  * immer Übergang in der Taktmitte für Taktrückgewinnung
  * 0: Spannungswechsel am Taktanfang
  * 1: kein Spannungswechsel am Taktanfang
  * J–Codeverletzung: Kein Übergang am Anfang und kein Übergang in Mitte
  * K–Codeverletzung: Übergang am Anfang, kein Übergang in Mitte



6.1.4 Nachrichtenrahmen

- Nachrichten werden in Rahmen (frames) eingefügt
- neben der Nutzinformation (Daten) gibt es notwendige Verwaltungsinformation, wie
  - ⋆ Absenderadresse
  - ⋆ Empfängeradresse
  - ⋆ Priorität
  - ⋆ Länge Daten
  - ⋆ Sequenznummer
  - ⋆ Quittungen
  - ⋆ Fehlersicherungscode (frame check sequence, FCS)
  - ⋆ Anfangskennung einer Nachricht
  - ⋆ Endekennung einer Nachricht

Vorspann (Kopf)	Daten	Nachspann (Schluß)

Anf.-kennung	Verw.-Inf.	Daten	FCS	Ende-kennung

- asynchrone und synchrone Nachrichtenübertragung unterschieden:
  - ⋆ Asynchroner Start–/Stop–Betrieb
    - ∘ jedes Zeichen (5 bis 7 Bit) synchronisiert sich mit einem vorangestellten Startbit (negative Flanke) und einem nachgestellten Stopbit
    - ∘ Sender und Empfänger benötigen wegen des kurzen Gleichlaufs nur geringe Taktübereinstimmung
  - ⋆ Synchrone Übertragung
    - ∘ die Zeichen einer Nachricht werden unmittelbar nacheinander übertragen
    - ∘ erfordert hohe Zeitstabilität beider Taktgeber bzw. Taktrückgewinnung
    - ∘ ggf. vor Rahmen noch spezielle Steuerzeichen zur Synchronisation (SYN im ISO–7–Bit–Code)

- Code–transparente Datenübertragung
  - ⋆ Ziel:
    Übertragung beliebiger Zeichen (Binärdaten)
  - ⋆ Problem:
    Anfangs– oder Endekennung dürfen in den Daten nicht auftreten
  - ⋆ Lösung bei zeichenorientierter Übertragung
    - ∘ Aussehen Nachricht:

STX	Daten	ETX

      mit STX Start of Text
      ETX End of Text
    - ∘ Darstellung von STX und ETX im Text durch vorausgestelltes Fluchtsymbol DEL (byte–stuffing)
    - ∘ Fluchtsymbol DEL im Text wird DEL DEL
  - ⋆ Lösung bei bitorientierter Übertragung
    - ∘ Aussehen Nachricht (HDLC–Protokoll):

01111110	Daten	01111110

    - ∘ in Daten wird nach 5 aufeinanderfolgenden 1 eine 0 eingeschoben und beim Empfänger wieder entfernt (bit–stuffing)

6.1.5 Erkennen von Übertragungsfehlern

- Einzelfehler, insbesondere aber Bündelfehler
- Beispiel: Bei 4800 bps stört EMS von 10 msec 48 Bits
- Fehlerraten, z.B. Telefon 10^{-5}, LAN 10^{-9}, LWL 10^{-12}
- Methoden zur Fehlerkennung
  - ⋆ redundante Übertragung mit Vergleich der Nachrichten auf Übereinstimmung beim Empfänger
  - ⋆ Sicherungscode
- Sicherungscodes
  - ⋆ Querparität (vertical redundancy check VRC)
    - ∘ Paritätsbit pro 7-Bit-Zeichen oder Wort
    - ∘ Bitfehler ungerader Anzahl erkennbar (quer)

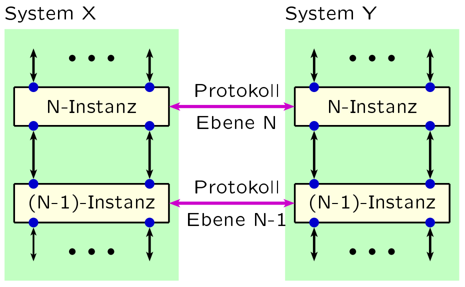
- ⋆ Längsparität (longitudinal redundancy check LRC)
  - ∘ EXOR über alle Bytes bzw. Worte
  - ∘ Bitfehler ungerader Anzahl erkennbar (längs)
- ⋆ Kreuzsicherung
  - ∘ Längs- und Querparität
- ⋆ Zyklischer Sicherungscode (cyclic redundancy check CRC)
  - ∘ Bits einer Nachricht als Koeffizienten eines Polynoms betrachtet
  - ∘ Division durch geeignetes Generatorpolynom $G_r(x)$ des Grads r
  - ∘ Rest wird als Sicherungscode (Länge: r Bit) übertragen
  - ∘ Bündelfehler bis zur Länge r erkennbar
  - ∘ Bündelfehler der Länge größer r mit Wahrscheinlichkeit 2^{-r} nicht erkennbar
  - ∘ heute übliches Verfahren; mit $G_{16}(x) = x^{16} + x^{12} + x^5 + 1$ (bei HDLC)

- Fehlerbehandlung
  - ⋆ Nachrichten sind numeriert
  - ⋆ Quittieren der Nachricht (acknowledge ACK)
  - ⋆ manchmal auch negative Quittung, falls Fehler erkannt
  - ⋆ Ausbleiben der positiven Quittung (Zeitüberschreitung) bedeutet Fehler

    (hervorgerufen auch durch Verlust der Nachricht im Netz oder durch Nichterkennung von Nachrichten wegen Fehlern in Blockkennzeichen)
  - ⋆ erneutes Senden der Nachricht
  - ⋆ damit aber Problem der Erkennung von Nachrichtenduplikaten
  - ⋆ Duplikate anhand Sequenznummer erkennbar (sliding window protocol)

6.2 Protokolle

- Protokolle sind Regeln für den Datenaustausch zwischen Partnern auf derselben Ebene (peer-to-peer)



↔ (Peer-to-Peer) Protokoll
• Dienstezugangspunkt (service access point)
↕ Diensteprotokoll, Diensteschnittstelle

- Rechnernetz–Protokolle durch ISO/OSI–Referenzmodell beschrieben
- international genormt
- Schichten (layers)

7 Application	Anwendung
6 Presentation	Darstellung
5 Session	Sitzung
4 Transport	Transport
3 Network	Vermittlung in Netzen
2 Data–Link	Sicherung (Punkt-zu-Punkt-Verbindungen, (Mehrpunktverbindungen)
1 Physical	Physikaklische Verbindung

- Feldbusprotokolle sind Spezialversionen für die Geräte der Feldebene. Sie füllen wegen der geringeren Anforderungen und aus Kostengründen meist nur die Schichten 1,2 und 7 aus.

6.3 Aufgabe und Anforderungen

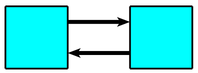
- Feldbus ist Verbindung zwischen Prozeßrechner und den (oft einfachen) Geräten (auf Feldebene)
- viele Geräte am Bus
- Echtzeitbedingungen
  - ⋆ deterministisches Zugriffsverfahren, d.h. projektierbares Zeitverhalten
  - ⋆ Zykluszeiten 1 bis 10 msec bei 40 bis 60 Geräten
  - ⋆ effizientes Protokoll auch bei kurzen Nutzdatensätzen, d.h. Rahmenorganisation kurz
  - ⋆ Prioritäten der Nachrichten (für Alarme)
  - ⋆ Optimierung von zyklischen Übertragungen, da die Regelungstechnik häufig ein mögliches äquidistantes Zeitraster fordert
- weitere Anforderungen
  - ⋆ zuverlässig in rauher Umgebung
  - ⋆ Übertragungsrate muß nicht unbedingt sehr hoch sein (serieller Bus reicht)

  - ⋆ geringe Kosten für Bus
  - ⋆ das Protokoll soll einfach und stabil sein
  - ⋆ einfache, preiswert zu realisierende Schnittstellen zum Bus, da Teil einfacher und preisgünstiger Sensorik
- Vielzahl von Feldbus–Entwicklungen
  - ⋆ Manufacturing Automation Protocol (MAP), USA
  - ⋆ Factory Instrumentation Protocol (FIP), Frankreich
  - ⋆ Process Field Bus (PROFIBUS), BMFT
  - ⋆ PROFIBUS–DP (dezentrale Peripherie)
  - ⋆ BITBUS, Intel
  - ⋆ INTERBUS–S, Phönix Contact
  - ⋆ Controller Area Network (CAN), Bosch
  - ⋆ Serielles Echtzeit Kommunikationssystem (SERCOS), Bosch
  - ⋆ Local Operating Network (LON), Echelon
  - ⋆ Aktor-Sensor-Interface (ASI), VDMA
  - ⋆ European Installation Bus (EIB)

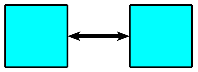
- schon ab 1982 von General Motors das inzwischen genormte Protokoll MAP (manufacturing automation protocol) entwickelt, das alle 7 Schichten des ISO/OSI–Modells umfaßt. Es zeigte sich, daß MAP für die Belange der Feldebene oft zu umfangreich, zu wenig effizient und daher zu teuer ist
- daher bei Feldbusprotokollen oft nur Schicht 1, 2 und 7, restliche Schichten rudimentär

6.4 Verbindungsstrukturen

- Punkt-zu-Punkt-Verbindung, vollduplex

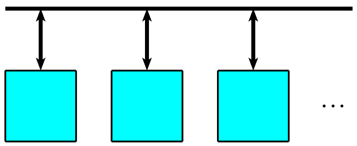


  - ⋆ keine Konflikte
- Punkt-zu-Punkt-Verbindung, halbduplex



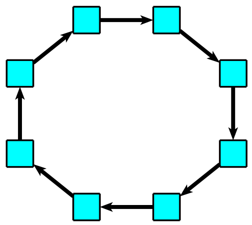
  - ⋆ Konflikt bei gleichzeitigem Schreiben
  - ⋆ gleichzeitige Benutzung des Mediums muß durch eine Zugriffsstrategie (Arbitrierung) vermieden werden

- Bus–Struktur



  - ⋆ Arbitrierung notwendig
  - ⋆ Beispiele: Aloha, CSMA/CD (Ethernet), versch. Reservierungsprotokolle, CAN-Bus
  - ⋆ Ethernet nicht echtzeitfähig, da maximale Wartezeit beliebig hoch im belasteten Fall

- Ringstruktur



  - ⋆ Teilnehmer an geschlossenen Übertragungsring angekoppelt
  - ⋆ Daten durchlaufen den Ring und werden vom Sender wieder entfernt
  - ⋆ Arbitrierung, meist durch Tokenweitergabe, nötig
  - ⋆ Beispiele: Token–Ring, FDDI
  - ⋆ Token–Ring echtzeitfähig, da maximale Wartezeit determiniert

6.5 Buszugang

- Frequenzmultiplex (eigentlich keine Arbitrierung)
  - ⋆ ein Teilnehmer je Kanal
  - ⋆ mehrere Teilnehmer je Kanal
- Zeitmultiplex
  - ⋆ zentrale Arbitrierung
    - ◦ Bus–Arbiter (zentraler Voter)
    - ◦ Daisy-Chain
    - ◦ Master–Slave
  - ⋆ dezentrale Arbitrierung
    - ◦ Token–passing–Verfahren
    - ◦ Spezialösungen, z.B. beim CAN-Bus
  - ⋆ zufälliger Buszugriff
    - ◦ ohne Reservierung (Aloha, CSMA/CA, CSMA/CD)
    - ◦ mit Reservierung (Multi Level Multi Access, Reservation Aloha)

6.6 CAN–Bus

- CAN = Contoller Area Network
- Ziele
  - ⋆ zuverlässiger Bus zur Vernetzung von Sensoren und Aktoren;
  - ⋆ Haupteinsatzgebiet zunächst Kfz
- von Bosch entwickelt
- Lizenzen an INTEL, PHILIPS, MOTOROLA, SIEMENS u.a.
- Norm ISO 11519 und 11898 (Schicht 1 und 2)
- Nutzerorganisation: CAN in Automation CiA jetzt weltweiter Einsatz in Automatisierung

6.6.1 Schicht 1

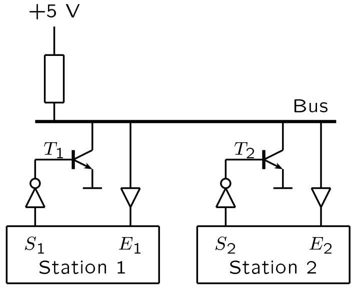
- High–speed:
  125 kbps bis 1 Mbps (max. 40 m)
- Low–speed:
  5 kbps (10 km) bis 125 kbps (530 m)

- Übertragungsmedium nicht festgelegt; meist 2–Drahtleitung (Spannungsdifferenz RS-485), empfohlen ISO/IS 11898
- NRZ–Code, deshalb Bitstuffing
  (nach 5 gleichen Bits wird ein komplementäres Bit eingefügt und auf der Empfangsseite wieder entfernt)
- CSMA/CA– (carrier sense multi access / collision avoidance) Verfahren wird dominantes D–Bit und ein rezessives R–Bit durch entsprechende Buspegel definiert

  D–Bit: logisch "0"
  R–Bit: logisch "1"

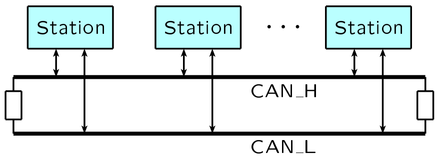
- D–Bit überwiegt R–Bit, falls zwei Geräte gleichzeitig senden
- Ein R–Bit–Sender kann somit das gleichzeitige Senden eines D–Bits am Buspegel erkennen
- eine Station i vergleicht empfangenes Bit E_i mit gesendetem Bit S_i und hört auf zu senden, falls $E_i \neq S_i$.

- Realisierung von D– und R–Bit z.B. durch Wired–And–Schaltung

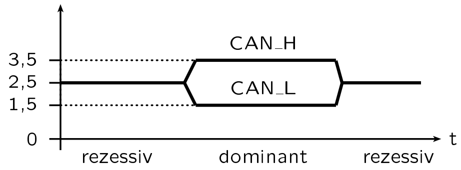


- ⋆ Transistor T_i leitet, falls $S_i = 0$
  → Buspegel = 0
- ⋆ also: D–Bit überwiegt R–Bit

- Busstruktur



- Buspegel nach ISO 11898



6.6.2 Schicht 2

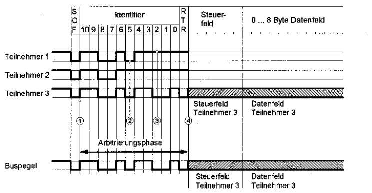
- Nachrichtenrahmen

≥ 3	1	11	1	6	0..64	16	2	7	Bit
IS	SOF	ID	RTR	C	Daten	CRC	ACK	EOF	

IS Interframe Space
 mindestens 3 R–Bits zwischen Nachrichten
SOF Start of Frame (1 D–Bit)
ID Identifier (11 Bits)
 davon für Benutzer nur 2032 IDs
 extended Identifier (29 Bits)
RTR Remote Transmission Request
 D–Bit: Es folgen Daten
 R–Bit: Sendeaufforderung an Empfänger
C Count, 6 Bit: C[0:5]
 erste beide Bits sind Null (c[5]=c[4]=0)
 C[3]...C[0] enthält Länge Datenfeld
 in Bytes (max. 8)
CRC Cyclic Redundancy Check
 15 Bit Prüfsumme und vorangestellte 0
ACK Acknowledge (2 Bits: a und dann R–Bit)
EOF End of Frame: 7 R–Bits

- Maximale Länge einer Nachricht bei 8 Byte Nutzdaten:
  $3 + 1 + 11 + 1 + 6 + 64 + 16 + 2 + 7 = 111 bit$

- daraus maximale Verzögerung bei Sendewunsch einer hochprioren Nachricht
- Busarbitrierung
  - ⋆ eine Station darf senden, falls Bus frei (carrier sense)
  - ⋆ Nachricht beginnt mit Feld zur eindeutigen Busarbitrierung (enthält Nachrichten-ID), falls mehrere Stationen gleichzeitig zu senden anfangen sollten
  - ⋆ es gibt keine gleichzeitigen Sendevorgänge mit gleicher ID
  - ⋆ jede Sendestation hört auf zu senden, wenn auf Bus nicht das gesendete Zeichen erkennbar
  - ⋆ die Station mit der kleinsten Nachrichten-ID setzt sich durch (0 ist dominant)
  - ⋆ also: je kleiner ID desto höher Priorität beim Zugriff
  - ⋆ Übertragung des Siegers wird nicht gestört, da Ausbreitungszeit am Bus (bestimmt durch die Buslänge) viel kleiner als eine Bitdauer



Beispiel eines Arbitrierungsvorgangs im CAN-Protokoll. Die Teilnehmer 1, 2 und 3 beginnen gleichzeitig einen Arbitrierungsversuch (1). Teilnehmer 2 verliert zum Zeitpunkt (2), Teilnehmer 1 zum Zeitpunkt (3) das Buszugriffsrecht. Beide Teilnehmer gehen damit in den Empfangszustand; am Ende der Arbitrierungsphase (4) besitzt nur noch Teilnehmer 3 das Buszugriffsrecht und schaltet seine Nachricht auf den Bus.

- ID ist Objekt–Kennzeichen, durch das der Anwender den Inhalt der Nachricht erkennt
- keine Zuordnung ID zu Station
- Station nimmt nur die für sie bestimmten Nachrichten an (Akzeptanzfilter, Einzelkommunikation oder multicasting)
- Wegen dieser Akzeptanzstrategie sprechen die Entwickler von "objektorientiertem Verfahren" oder "verteiltem Speicher"
- Quittungen

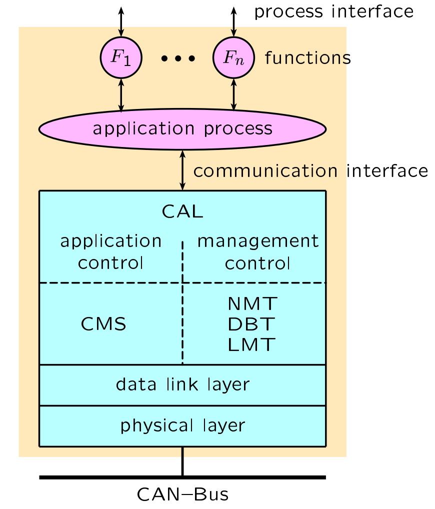
- ⋆ Sender setzt a in ACK auf R
- ⋆ falls der Empfänger Nachricht korrekt erhält (CRC–Test ok), setzt er noch während des Empfangs a auf D
- ⋆ Absender hört beim Senden mit und erkennt so, daß a auf D gesetzt wurde, also eine positive Quittung eines Empfängers vorliegt
- ⋆ Erkennen Stationen einen Fehler (bit–, bitstuff–, CRC–Fehler), dann senden diese im Anschluß an das ACK–Feld eine Folge von dominanten Bits als Fehlerkennzeichen (mindestens 6 aufeinanderfolgende dominante Bits, da diese in gültiger Nachricht wegen Bitstuffing nicht auftreten können). Die Nachricht wird daraufhin erneut gesendet. Durch ein Zählverfahren bemerkt jede Station die Anzahl ihrer Fehler. Beim Überschreiten des Grenzwertes werden Stationen vom Bus abgeschaltet.

- Puffer der Schicht 2
  - ⋆ BASIC CAN
    - ◦ 1 Sendepuffer
    - ◦ 2 Eingangspuffer
    - ◦ Wechselpufferkonzept
  - ⋆ FULL CAN
    - ◦ mehrere Sendepuffer
    - ◦ mehrere Empfangspuffer
    - ◦ alle Puffer je einem festen Identifier zugeordnet
    - ◦ Puffer zu einem ID werden überschrieben, falls neue Nachricht dieser ID eintrifft

6.6.3 Schicht 7

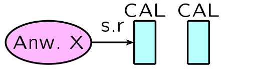
- CAN Application Layer (CAL)
- Erweiterungen
  - ⋆ bestätigter Datentransfer
  - ⋆ Übertragung von Paketen > 8 Bytes
  - ⋆ ID Distribution
  - ⋆ Spezifikation von Standard Device Modules
  - ⋆ Startup
  - ⋆ Knotenüberwachung
- Standards
  - ⋆ CAL/CANopen (CiA)
  - ⋆ Device Net (Allen Bradley)
  - ⋆ SDS (Smart Distribution System), Honeywell Micro Switch

- Struktur eines CAN/CAL–Moduls



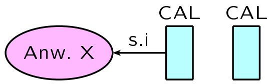
- Literatur
  K. Etschberger: CAN: Grundlagen, Protokolle, Bausteine, Anwendungen. Hanser 1994
- Benutzerorganisation CiA legte für CAL standardisierte Objekte (Variable, Ereignisse u.a.) und Dienste fest
  `http://www.can-cia.de/`
- Moduln
  - ⋆ CMS Can–based message specification
    - ∘ Objekte durch Attribute wie Name, Typ, Priorität, minimale Sendewiederholzeit beschrieben
    - ∘ Übliche Dienste:
      Read, Write, Notify, Load usw.
  - ⋆ DBT Distributor
    - ∘ dynamische Zuordnung von CAN–Identifiern zu CMS–Objekten bei der Initialisierung
    - ∘ erleichert Einsatz von Geräten verschiedener Lieferanten
    - ∘ netzweite Konsistenz der ID bei Sendern und Empfängern

- ⋆ NMT Netzwerk–Management
  - ∘ Initialisierung, Starten und Stoppen von Prozessen auf Knoten
  - ∘ Erkennen von Fehlerzuständen im System
  - ∘ Schreiben/Lesen von Parametern bei den Knoten bei der Initialisierung
- ⋆ LMT Layer Management
  - ∘ Setzen von Zeitparametern in Schicht 2
- Ein Knoten muß für CAL die Masterfunktion übernehmen und einige Identifier hierfür reservieren
- CMS–Dienste
  - ⋆ lokale Dienste
    vom CAL–Modul selbst ausgeführt



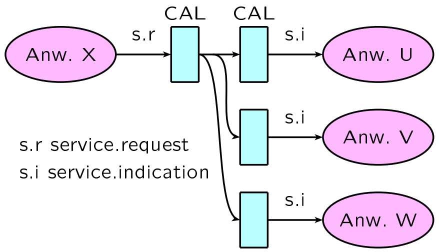
Anw. X = Anwendung X
s.r service.request

- ⋆ unangeforderte Dienste
  lokales CAL zeigt ein von ihm festgestelltes Ereignis an



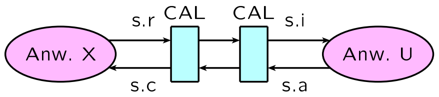
s.i service.indication

- ⋆ unbestätigte Dienste
  Dienstanforderung über den CAN–Bus verschickt an einen oder mehrere Server

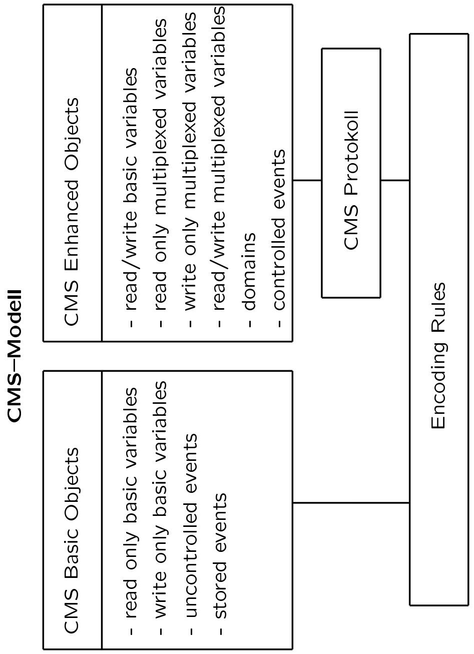


s.r service.request
s.i service.indication

- ⋆ bestätigte Dienste
  Dienst von genau einem Server angefordert; dieser meldet das Ergebnis der Dienstausführung zurück



s.r service.request
s.i service.indication
s.a service.response
s.c service.confirmation

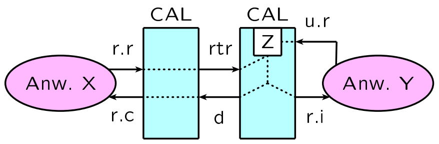


- CMS basic objects
  - ⋆ read only variable
    - ∘ kann vom Client nur gelesen werden
    - ∘ read.indication geht an Server-Prozeß
  - ⋆ write only variable
    - ∘ kann vom Client nur geschrieben werden
    - ∘ write.indication geht an Server-Prozeß
  - ⋆ uncontrolled event
    - ∘ Mitteilung, daß ein Ereignis eingetreten, wird spontan von einem Server an alle Clienten (bei Initialisierung definiert) geschickt
  - ⋆ stored event
    - ∘ Ereignis hat Eigenschaften einer read–only–Variable
    - ∘ zusätzlich kann es wie ein uncontrolled event mit dem Dienst "store and notify" vom Server beschrieben werden, wobei ebenfalls eine Nachricht an alle Clienten geschickt (mit dem Ereigniswert)

- Dienste für CMS basic objects
  - ⋆ Identifier der Nachricht bezeichnet eindeutig das basic object
  - ⋆ Dienste

basic object	Dienst	Diensttyp	Anforderer
write-only	write	unbestätigt	Client
read-only	update	lokal	Server
read-only	read	bestätigt	Client
uncon. event	notify	unbestätigt	Server
stored event	store	lokal	Server
stored event	store & notify	unbestätigt	Server
stored event	read	bestätigt	Client

  - ⋆ Zugriff auf die Variablen direkt vom CMS abgehandelt
  - ⋆ Beispiel:
    Übermittlung von Meßergebnissen bzw. Prozeßzuständen über Read–Only–Variable Z des Prozesses Y

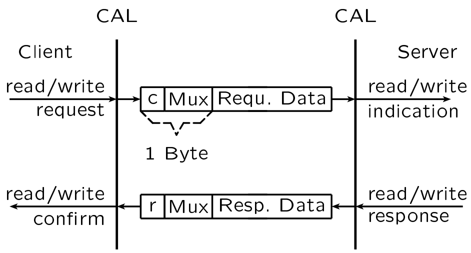


r.r read.request
rtr remote transmit request
u.r update.request
r.i read.indication
r.c read.confirmation
d CAN-Message with data

Erklärung:
Der Prozeß Y führt eine einfache Variable Z (z.B. die gemessene Temperatur in einem technischen Prozeß). Die Temperatur wird periodisch abgelesen und durch update.request wird die Variable Z aktualisiert. Der Prozeß Y kann die Variable Z jederzeit über einen Dienst read.request lesen. Der Prozeß X wird darüber informiert, daß die Variable gelesen wurde (read.indication).

- CMS enhanced objects (erweiterte Objekte)
  - ⋆ read–write basic variables
    - ∘ 1 Client und 1 Server
    - ∘ in CAN–Nachricht anzugeben, ob lesen oder schreiben
    - ∘ Dienste: read, write
  - ⋆ multiplexed variables
    - ∘ 1 Client und 1 Server
    - ∘ mehrere Variable zu einem CMS–Objekt zusammengefaßt
    - ∘ es gibt Strukturen und Felder
    - ∘ Dienste: read, write
  - ⋆ domains
    - ∘ Datenbereich > 8 Bytes, z.B. für Programme
    - ∘ Dienste (vom Client angefordert): upload, download, initiate, abort
  - ⋆ controlled events
    - ∘ Anzeige des Ereignisses kann freigegeben oder gesperrt werden
    - ∘ es gibt genau einen Server und genau einen Clienten
    - ∘ Anwendung : Synchronisation
    - ∘ Dienste: notify, control

- Beispiel: read–write–Anforderung für Multiplexvariable

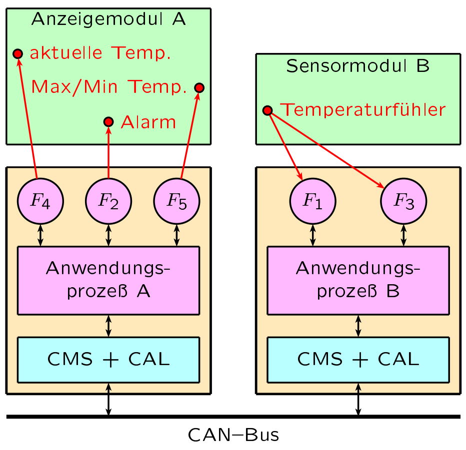


Erläuterung:

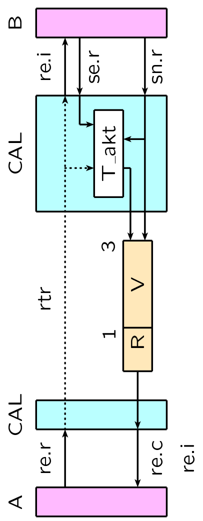
$$Mux = \begin{cases} 0 & \text{basic variable oder erstes Element einer multiplexed variable} \\ \neq 0 & \text{Auswahlindex multiplexed Variable } (\leq 128 \text{ Elemente}) \end{cases}$$

c: request code (0 write, 1 read)
r: result code (0 success, 1 failure)
Request Data: zu schreibende Daten bei c = 0
Response Data:
  geschriebene Daten bei c = 0, r = 0
  gelesene Daten bei c = 1, r = 0
  Fehlercode bei r = 1

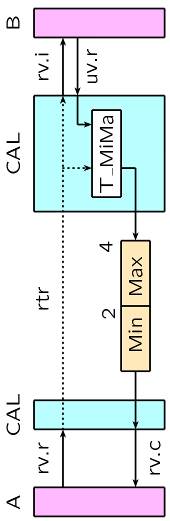
- großes Anwendungsbeispiel (aus Etschberger): Ablesen und überwachen von Temperaturen in einem Sensormodul sowie Anzeige durch einen Anzeigemodul



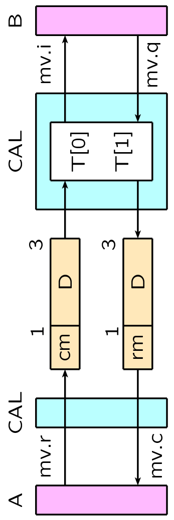
- ⋆ Aufgabe Anzeigemodul A
  - ∘ aktualisiert periodisch Temperaturanzeige (Funktion F4)
  - ∘ gibt Alarm bei Grenzwertüberschreitung (Funktion F2)
  - ∘ aktualisiert periodisch Anzeige der aufgetretenen Temperaturextrema (Funktion F5)
- ⋆ Aufgabe Sensormodul B
  - ∘ Beispiel für ein CMS-Profil
  - ∘ liest periodisch Temperatur und kontrolliert die Grenzwerte (Funktion 1)
  - ∘ bestimmt minimale und maximale aufgetretene Temperaturen (Funktion 3)
- ⋆ weitere Aufgabe von Anwendungsprozeß A
  - ∘ Einstellen der minimalen und maximalen Temperatur im Sensormodul



re.r read_event.request
rtr remote transmission request
re.i read_event.indication
se.r store_event.request
sn.r store_and_notify.request
re.c read_event.confirmation
R (in/out of) Range
V Value (Temp. Wert)
T_akt aktuelle Temperatur
Stored_Event
Struktur: R (1 Byte), V (2 Byte)



rv.r read_variable.request
rtr remote transmission request
rv.i read_variable.indication
uv.r update_variable.request
rv.c read_variable.confirmation
Min minimale Temperatur
Max maximale Temperatur
T_MiMa minimale und maximale Temperatur
Read_Only_Variable
Struktur: Min (2 Byte), Max (2 Byte)



mv.r	read/write_mux_variable.request
mv.i	read/write_mux_variable.indication
mv.q	read/write_mux_variable.response
mv.c	read/write_mux_variable.confirmation
cm	c (1 Bit), Mux (3 Bit)
rm	r (1 Bit), Mux (3 Bit)
D	Data
T	Temperatur
	Read_Write_Multiplexed_Variable
T[1]	Obergrenze Temperatur
T[0]	Untergrenze Temperatur

6.7 MAP, Überblick

- Manufacturing Automation Protocol (MAP)
- Entwicklung ab 1982 (General Motors)
- inzwischen genormt
- Probleme
  - ⋆ Breitband–Geräteanschlüsse (10 Mbps) (zu) teuer
  - ⋆ FULL–MAP–Protokoll (zu) umfangreich und (zu) langsam
  - ⋆ kein Massenmarkt, daher Komponenten (zu) teuer
- Schicht 1 (physical))
  - ⋆ Token-Bus (IEEE 802.4)
  - ⋆ FULL–MAP mit Breitbandtechnik (10 Mbps)
  - ⋆ Routers und Gateways zur Kommunikation mit Full–MAP–Segmenten nötig
  - ⋆ auch Basisband-Technik mit 5 Mbps

- Schicht 2 (data link)
  - ⋆ Logical Link Control (LLC, IEEE 802.2)
  - ⋆ 3 Diensttypen
    - ◦ unbestätigte Datagramme (Typ 1)
    - ◦ verbindungsorientierte Nachrichten (Typ 2)
    - ◦ bestätigte Datagramme (Typ 3)
  - ⋆ reduzierte Definitionen für die Steuerebenen (real time segments) wie MINI–MAP und EAP (enhanced architecture protocol)
- Schicht 3 (network)
  - ⋆ End System to Intermediate System Vermittlungsprotokoll (ES-IS, ISO 9542)
  - ⋆ Connectionless-Mode Network Service (CLNS, ISO 8473)
- Schicht 4 (transport)
  - ⋆ Transport Service Class 4 (ISO 8072)
  - ⋆ end-to-end Protokoll
  - ⋆ verbindungsorientiertes Protokoll

- Schicht 5 (session)
  - ⋆ Basic Connection Oriented Session Service Definition and Protocol (ISO 8326, ISO 8327)
- Schicht 6 (presentation)
  - ⋆ abstrakte Syntax und Transfersyntax (ASN.1)
  - ⋆ ISO 8822, ISO 8823
- Schicht 7 (application)
  - ⋆ Aufteilung in 7a und 7b
  - ⋆ Schicht 7a (ACSE)
    - ◦ ACSE Association Control Service Elements
    - ◦ ISO 8649, ISO 8650
    - ◦ allgemeine Dienste zum Verbindungsaufbau und zum Verbindungsabbau:
      Initiate, Conclude, Abort, Cancel, Reject
    - ◦ Namen/Adressen von Objekten in Schicht 7

- ⋆ Schicht 7b
  - ◦ MMS Manufacturing Message Specification (ISO 9506)
  - ◦ FTAM File Transfer Access and Management (ISO 8571)
  - ◦ NM Network Managment (ISO 9595)
  - ◦ DS Directory Services (ISO 9594)
  - ◦ MMS-Zusatzdefinition (bisher kein Standard) zur Einbindung der Dienstaufrufe in C (Parameter, Typen, Anzahlen) existiert
  - ◦ Begleitstandards (companion standards) für Roboter, NC-Maschinen und SPS

6.8 MMS bei MAP

- MMS Manufacturing Message Specification
- Client/Server–Konzept
- 16 Objekttypen mit Attributen
- 79 Dienste für Objekte (ohne FTAM)
- Modifikation der Objekte durch den Server über das Netz, durch den Client, durch autonome Übergänge
- Conformance Klassen
  - ⋆ wurden festgelegt, da volles MMS sehr komplex und einfache Geräte nur wenige Funktionen erbringen
  - ⋆ beschreiben die Dienste, die ein Gerät einer Klasse mindestens beherrschen muß
  - ⋆ beim Verbindungsaufbau werden Dienste und Parameter ausgehandelt, im Feldbereich i.a. nur bei System–Initialisierung
  - ⋆ einfachste Geräteklasse hat 5 Dienste

- Objektklasse Virtuelles Gerät (virtual manufacturing device, VMD)
  - ⋆ ein oder mehrere VMD's auf einem realen Gerät realisiert
  - ⋆ VMD's enthalten Objekte und Zustandsinformationen
  - ⋆ Ein VMD hat viele Objektattribute wie z.B.
    - ◦ Steuerprogramm–Name
    - ◦ Hersteller
    - ◦ Modell
    - ◦ Version
    - ◦ Liste akzeptierter Syntax
    - ◦ Status (voll, eingeschränkt)
    - ◦ physikalischer Status (ok, defekt)
    - ◦ Liste von enthaltenen Objektklassen

- Objektklasse Bereich (domain)
  - ⋆ Betriebsmittel
  - ⋆ meist Objekte, die Code und Daten enthalten können (Speicher)
  - ⋆ definieren den lokalen Gültigkeitsbereich der in ihnen enthaltenen Variablen
  - ⋆ Attribute z.B.
    - ◦ Name
    - ◦ Art (löschbar, sharable)
    - ◦ Liste enthaltener Variablen
    - ◦ Liste enthaltener PI
    - ◦ Zustand (loading, complete, in use u.a.)
  - ⋆ Dienste
    - ◦ Definition Bereich
    - ◦ Löschen Bereichen,
    - ◦ Übertragung in Bereich (upload/download)

- Objektklasse Abläufe (program invocation, PI)
  - sind die Bearbeiter–Aktivitäten des Servers (tasks, threads)
  - PI's auch dynamisch erzeugt und gelöscht
  - Attribute z.B.
    - Name
    - Art (löschbar, reentrant u.a.)
    - Zustand (idle,starting, running, stopped, resetting, unrunable u.a.)
    - Liste benutzter domains
    - Liste relevanter Ereignisse
  - Dienste
    - Definition PI
    - Löschen PI
    - Starten
    - Halten und Weiterführen (resume) der Programmausführung
    - Reset– und Kill–Funktionen

- Objektklasse Variable
  - Variable sind Objekte, deren Inhalt Werte sind
  - es gibt
    - unnamed variables
    - named variables
    - named typed variables
    - scattered access variables (Gruppe, nur Gesamtzugriff)
  - Zugriff durch den Client
  - Attribute z.B.
    - Zugriffsart
    - Zugriffsrechte
  - Dienste z.B.
    - lesen
    - schreiben

- Objektklasse Semaphor
  - Arten
    - Token–Semaphor (zählender Semaphor)
      Menge identischer Token, die frei oder belegt sind
    - Pool–Semaphor (binärer Semaphor)
      Token mit eindeutigem Namen
    - Semaphor–entry–Objekte
      Listenführung über "frei, belegt, wartend"; auch mit Zeitüberwachung und maximaler Belegungsdauer
  - Attribute z.B.
    - Name, Art, Anzahl, Status (frei/belegt)
    - Liste Anforderer/Beleger, Prioritäten
    - Reaktionen bei Zeitüberschreitung oder Verbindungsabbruch
  - Dienste z.B.
    - definieren, löschen
    - belegen, freigeben
    - Status abfragen
  - Steuerung anderer Dienstanforderungen in Abhängigkeit vom Semaphorzustand

- Objektklasse Ereignis (event)
  - Objekte
    - event condition (Ereignisbedingung)
      - lokale Bedingungen
      - Zustandsänderungen
      - vom Client ausgelöst
    - event–action (auszuführende Aktion)
    - Event–enrollment
      (Liste der Clients, die beim Eintreten verständigt werden)
  - Dienste z.B.
    - Definition
    - Ändern und Löschen von Ereignisbedingungen und Aktionen
    - auslösen von Ereignissen
    - Abfrage von Attributen
    - Benachrichtigung bei Eintreten von Ereignissen

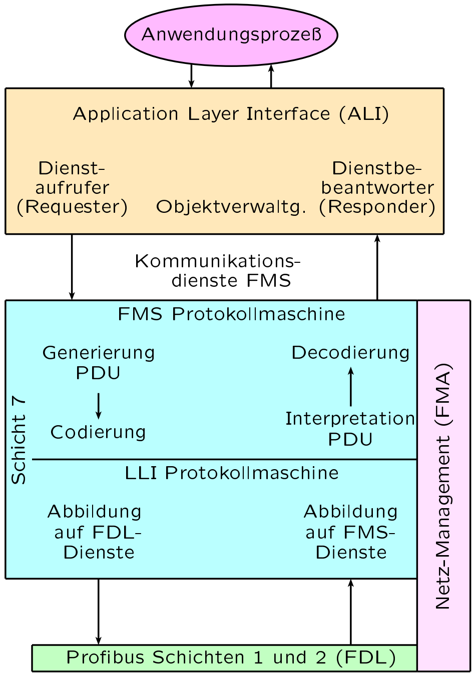
- Objektklasse Journal (journal)
  - Unterstützung bei der Erstellung von Logbüchern
  - Ordnung Informationen nach Zeitstempel oder Reihenfolge des Eintreffens
  - Dienste z.B.
    - definieren von Journalen
    - löschen von Journalen
    - Lesen/Schreiben von Journaleinträgen
    - Abfragen des Journalzustands

- Robot Control Companion Standards (RCCS)
  - ISO 9506
  - Definition abstraktes Modell eines Robotersystems
    - in MMS–Terminologie
    - als Zwischenschritt noch von MMS unabhängiges Modell mit roboterspezifischen, abstrakten Objekten und Attributen
    - hieraus dann konkrete MMS–Objekte und deren Namen
  - für die verschieden fähigen Robotersysteme wieder "conformance classes" festgelegt
  - Robotersysteme als VMD beschrieben mit zusätzlichen Attributen
  - die MMS–Dienste greifen auf die roboterspezifischen Objekte zu

  - jedoch die einzelnen Komponenten ohne Strukturierung
  - daher ein Vorschlag des deutschen NAM (Normenausschuß Machinenbau im DIN) zu ISO 9506 für eine hierarchische Strukturierung des physikalischen Robotersystems und (neu) die Einbeziehung der Fortbewegung mobiler Roboter
  - Robotersystem besteht aus:
    - einem oder mehreren Manipulatoren
    - Hilfseinrichtungen zur Steuerung des Robotersystems
    - einem Roboter–Steuerungs–System, das die Manipulatoren und Hilfseinrichtungen (intelligent) betreibt
    - den Roboter–Anwendungsprogrammen
    - ggf. dem Fahrwerk zur Mobilität

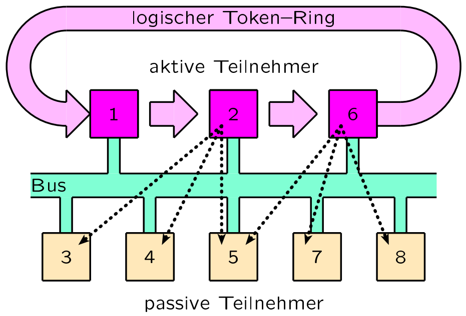
6.9 PROFI–Bus

- Process field bus (DIN 19245)
- deutsche Entwicklung ab 1987
- BMFT (FZI Karlsruhe) und 14 deutsche Industriepartner
- Master– und Slave– Geräte am Bus
- Norm definiert Schicht 1, 2 und 7 (3 bis 6 leer)
  - Anwendungsprozesse wenden sich über ALI (application layer interface) an die Schicht 7 (FMS field message specification, an MMS orientiert)
  - FMS verkehrt direkt mit Schicht 2 über LLI (lower layer interface)
- da FMS sehr umfangreich, PROFIBUS–DP als reduzierte Version
  - Es bleiben die Schichten 1 und 2, an Stelle von FMS treten einige wenige Funktionen und Betriebsarten
  - Anwenderprozesse wenden sich über DDLM (direct–data–link–mapper) direkt an Schicht 2



- Schicht 1
  - ⋆ RS–485 oder LWL
  - ⋆ Übertragung asynchron (NRZ) mit üblichen UART–Bausteinen: 11 Bit für 1 Byte (+ Start-, Paritäts-, Stopbit)
    UART= universal asynchronous receiver/transmitter
  - ⋆ Bitraten von 9.6 kbps bis 1.5 Mbps in Stufen
  - ⋆ maximal 32 Teilnehmer pro Segment
  - ⋆ mehrere Segmente über Repeater
  - ⋆ maximal 127 logische Teilnehmer
  - ⋆ zwei Prioritätsstufen

- Schicht 2
  - ⋆ Buszugriff
    - ◦ Master–Geräte (aktive Teilnehmer) im (logischen) Token–Bus–Ring
    - ◦ Slave–Geräte (passive Teilnehmer) senden auf Anforderung des Masters in dessen Buszugriffszyklus

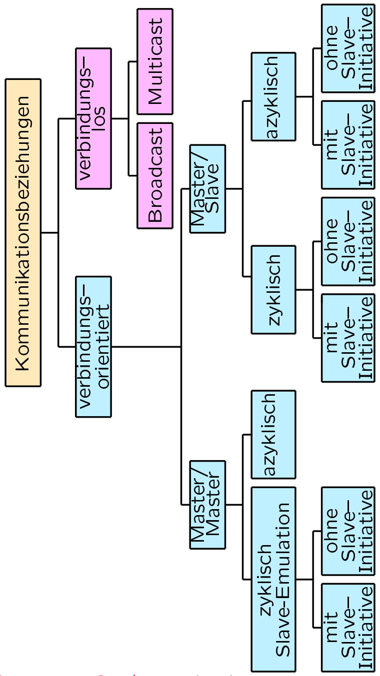


- ⋆ Nachrichtentypen
  - ◦ SDN Send data with no ACK (z.B. Rundruf)
  - ◦ Master/Master oder Master/Slave Nachrichten
    - ▷ SDA Send data with acknowledge
    - ▷ RDR Request data with reply
    - ▷ SDR Send and request data
  - ◦ Beschleunigung zyklischer Dienste durch "Polling–Liste" in LLI
    - ▷ CRDR Cyclic request data with reply
    - ▷ CSRD Cyclic send and request data

- ⋆ Rahmen (Telegramme)
  - ◦ Bezeichnungen für Felder

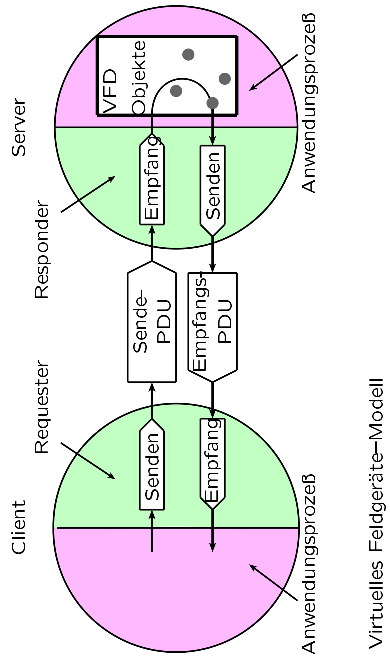
SD	Startzeichen (start delimiter) (vier Varianten SD1 bis SD4)
DA	Empfänger (destination address)
SA	Absender (source address)
ED	Endezeichen (end delimiter)
FC	Steuerzeichen (frame control) Aufruf, Quittung, Antwort, Priorität, Sequenznummer
LE	Länge (length)
FCS	Längsparität (frame–check sequence)

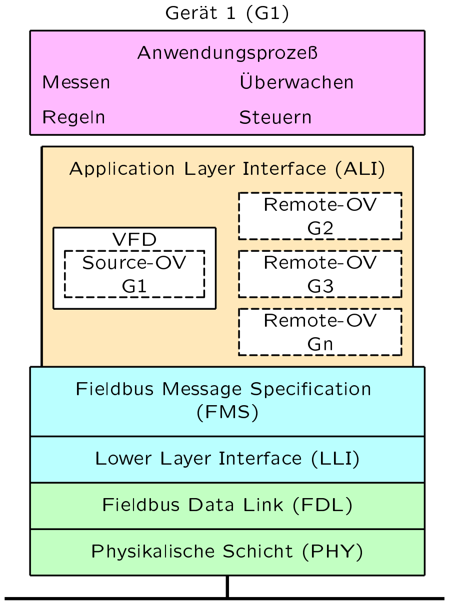
  - ◦ Kommando (6 Byte):
    SD1 DA SA FC FCS ED
  - ◦ Daten fester Länge (14 Byte):
    SD2 DA SA FC Daten (8 Byte) FCS ED
  - ◦ Daten, variabler Länge (max. 246 Byte):
    SD3 LE1 LE2 DA SA FC Daten FCS ED
  - ◦ Token: SD4 DA SA
  - ◦ Kurzquittung: SC



- ALI (application layer interface)
  - ⋆ Zur Kommunikation dienen logische Kanalnummern mit Eigenschaften, die bei Konfigurierung festgelegt
  - ⋆ Objekte
    - ◦ Objekte stehen im Anwendungsprozeß und sind über Prozeduranfrage von ALI aus zugänglich
    - ◦ statische Objekte
      - ▷ Variable (simple, array, record, list variable)
      - ▷ Bereich (domain)
      - ▷ Ereignis (event)
    - ◦ dynamische Objekte
      - ▷ Abläufe (program invocations)
    - ◦ Attribute z.B.
      - ▷ Index im Objektverzeichnis OV
      - ▷ Name
      - ▷ max. Größe in Bytes
      - ▷ phys. Adresse
      - ▷ Zustand
      - ▷ freie benutzerspezifische Attribute

- ⋆ Objektverzeichnisse (OV)
  - ◦ ALI enthält die Objektverzeichnisse (OV) der eigenen und ausschnittsweise der fremden Objekte und verwaltet sie
- ⋆ Dienste
  - ◦ Zugriff auf Variable (read/write)
  - ◦ Zugriff auf Bereiche (init, request, terminate up–/down–load)
  - ◦ Programm-Aufruf (create, delete, start, stop, resume, reset, kill)
  - ◦ Ereignisse (notification, alter–event–condition)
  - ◦ VFD-Verwaltung (für virtual field devices)
  - ◦ OV–Verwaltung
  - ◦ Kommunikationsüberwachung
- ⋆ Es werden VFD's (virtual field devices) definiert, die über ihre Objekte angesprochen werden





Schichtenmodell PROFI-Bus

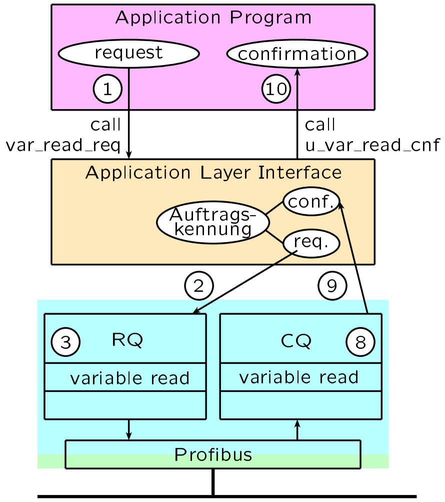
- Anwendungsbeispiel
  - ⋆ periodisches Messen Öltemperatur im Server
  - ⋆ Client liest aktuellen Wert
  - ⋆ Eintrag in OV

Index	Typ	Länge	Adresse	Symbol	...
		...			
112	7	2	6041	Öltemperatur	...
		...			

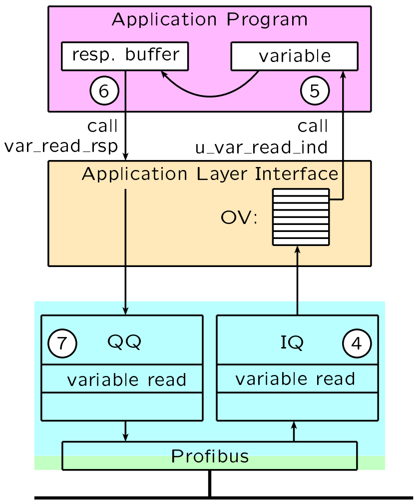
Anmerkungen:

- ∘ Typ 7 bedeutet INTEGER 16
- ∘ weitere Einträge z.B. Maßeinheit
- ⋆ nachfolgend Blockdiagramm für Client und Server
- ⋆ die ausgeführten Schritte beim Lesen durch den Client sind fortlaufend numeriert
- ⋆ Warteschlangen zur besseren Entkopplung der Prozesse

CQ confirmation queue
IQ indication queue
QQ response queue
RQ request queue



Client



Server

6.10 SERCOS-Bus

- serielles Echtzeit–Communications–System (SERCOS)
- entwickelt von Bosch und VdW
- Einsatz vor allem für Steuerung von Werkzeugmaschinen-Antrieben und Robotergelenken
- Lichtwellenleiter
- NRZI-Code mit Bitstuffing nach 5 mal "1"
- Ziel:
  präzise äquidistante Zeitraster der Übertragung
- Zeitraster z.B. 62μsec, 125μsec, usw. bis max. 65 msec
- eine Master-Steuerung ist mit (vielen) Slaves (z.B.Antrieben) in einem LWL–Ring verbunden
- bei Fehlern wird nicht wiederholt, sondern auf den nächsten Takt gewartet
- bei Mehrfachfehler: Abschalten

- 3 Typen von Rahmen ("Telegrammen") zur Aktualisierung der Ist–und Soll–Werte
  - ⋆ Master–Synchronisationstelegramme (SYNCH)
    Master–Station teilt Zeitbasis bezogen auf Zyklusbeginn mit (Rundspruch an alle Slaves)

SYNCH	...	Slave 1	...	Slave n	...	MT

  - ⋆ Antriebs–Telegramme (SLAVE)
    jeder Slave sendet pro Zyklus zu dem ihm zugeteilten Zeitpunkt die angeforderten Daten an den Master

SLAVE	SD	Adr	Daten (z.B. 32 Bit)	CRC	ED

  - ⋆ Master–Datentelegramm (MT)
    nach jedem Antriebs–Telegramm sendet der Master Daten (je 32 Bit) für alle Slaves

MT	SD	Adr	Daten für n Slaves (n$*$32 Bit)	CRC	ED

- Start– und Ende–Delimiter SD und ED sind das 01111110-Flag des HDLC–Protokolls
- Interpretation der Daten (z.B.Namen, Funktion) durch Id–Nummern (umfangreicher vorgegebener Katalog) festgelegt (z.B. bedeutet 36 einen Geschwindigkeits–Sollwert)
- viele Nummern frei für spezielle Anwendungen; dies vermindert aber die Portabilität der Programme

6.11 INTERBUS–S

- INTERBUS–S –Protokoll dient zur schnellen zeitäquidistanten Übertragung bei vielen Teilnehmern im Sensor–Aktor–Bereich
- eine Master–Steuerung und n Teilnehmer (Slaves) im Ring angeordnet
- die Daten aller Slaves werden ausgehend vom Master pro Zeitzyklus in einem Rahmen durchgeschoben (wie ein großes Schieberegister); die Slaves erhalten dabei aus dem Puffer des Masters die Daten (Sollwerte) und senden gleichzeitig ihre Ist– und Statuswerte
- die Daten werden in einem Rahmen (Summenrahmenverfahren) gesendet; es gibt keine Adressen, die Position im Rahmen gibt die Teilnehmer–Nummer

Bits	16	16	16	⋯	16	16	16
Frame	loopcheck	T_1	T_2	⋯	T_n	CRC	ED

- Ablauf
  - ⋆ jeder Slave hat ein Schieberegister (1 bis 4 Byte), in das er am Zyklusanfang seine Daten schreibt
  - ⋆ der Master beginnt seinen Puffer zu leeren, jeder Slave schiebt nun die Daten im Ring unbesehen weiter, bis auf die für ihn bestimmten
  - ⋆ letztere bringt er in sein Schieberegister und sendet dafür seine Daten
  - ⋆ am Ende hat jeder Slave in seinem Schieberegister die Daten des Masters und der Master in seinem Puffer die Daten aller Slaves
  - ⋆ beispielsweise Zykluszeit 3 msec bei 48 Slaves mit 16–Bit–Daten
- gelegentliche azyklische Daten (sogenannte Bedarfsdaten wie Parameter, Steueranweisungen) müssen in übergeordnetem Protokoll in 2–Byte–Pakete zerlegt und in freien Zeitschlitzen der Station gesendet werden

- viele Kennzahlen (analog wie bei SERCOS-Bus) erlauben eine einheitliche Interpretation
- Bewertung
  - ⋆ gut geeignet für Sensor–/Aktorebene (wenig Overhead, äquidistant)
  - ⋆ starre Stationszahl: Probleme beim Zu– und Abschalten von Teilnehmern

6.12 Busse zur Gebäude–Leittechnik

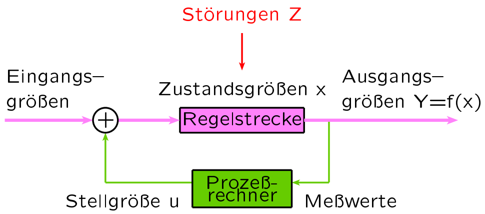
- Ziel
  Vernetzung von gemeinsamen und privaten Funktionen in Häusern, Schulen, Banken, Krankenhäusern, Industriegebäuden
- Milliardenmarkt für Vernetzung und Geräteprodukte
- derzeitige Vorschläge ähneln Sensor–Aktor–Bussen
- Beispiele u.a.
  - ⋆ EIB European Installation Bus
    unterstützt z.B. von Bosch, Siemens
    - ◦ twisted pair, powerline
      Funk, Infrarot, EIB.net
    - ◦ ISO/IEC 8802-2 Logical Link Layer
    - ◦ kurze Telegramme mit bis zu 14 Bytes Daten
    - ◦ 2000 Produkte verfügbar
    - ◦ Installationen mit bis zu 60000 devices
  - ⋆ LON Local Operating Network
    spezielle "Neuron"–Chips der Firma Echolon mit 3 CPU's (2 für Protokoll, 1 für Anwendungs–SW)

7 Regelungstechnik

7.1 Einführung

- Die Führung technischer Prozesse kombiniert Planung, Steuerung und Regelung
  - ⋆ Steuerung
    Einwirkung mittels Stellgrößen in eine gewünschte vorausgeplante Richtung.
    Beispiel: Heizung um 6:30 Uhr einschalten.
  - ⋆ Regelung
    Überwachung und Minimierung der Abweichungen zwischen Ist– und Sollwerten nach gegebenen Kriterien.
    Beispiel: Kesseltemperatur auf 43 Grad halten.
- System als Schachtelung von Regelschleifen abgestufter Reaktionsfähigkeit betrachtbar
  - ⋆ höhere Ebenen bzw. äußere Schleifen
    - ◦ Zeitanforderungen unkritisch (bis zu Tagen)
    - ◦ relativ große Abweichungen erlaubt

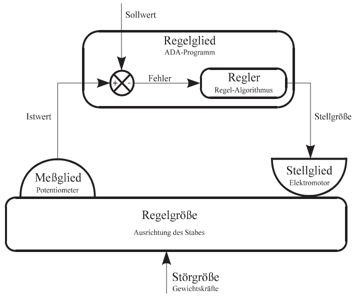
    - ◦ Realisierung in Software
    - ◦ Beispiel: Bestände in Materiallager
  - ⋆ niedere Ebenen bzw. innere Schleifen
    - ◦ kritische Zeitanforderungen (z.B. msec)
    - ◦ nur geringe Abweichungen erlaubt
    - ◦ Realisierung in HW
    - ◦ Beispiel: Temperatur auf 0.1 Grad halten
- Beispiel: Der Prozeßrechner als Regler



- Ziele
  - ⋆ Stellgröße u so wählen, daß Istwerte Y mit den vorgegebenen Sollwerten möglichst gut übereinstimmen
  - ⋆ gezielte Nachführung bei Störungen und neuen Vorgaben

- Mögliche Kriterien
  - ⋆ Abweichung möglichst schnell gegen Null
  - ⋆ Abweichung überschreitet nicht eine bestimmte Bandbreite
  - ⋆ kein Schwingen
  - ⋆ Integral über den Abweichungsbetrag wird über die Zeit minimiert
  - ⋆ energiesparende Nachführung
  - ⋆ technisch gut realisierbar (z.B. bei Temperaturkonstanz keine überdimensionale Heiz- und Kühlleistung gegenüber im Mittel benötigte)

7.2 Einfache Regler

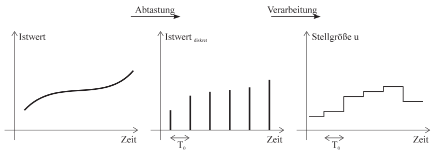


7.2.1 Der P-Regler

- einfachster Regler
- proportionales Verhalten
- Die Gleichung eines P-Reglers

$$u(t) = K * e(t)$$

- Diskretisierung: kontinuierliche Istwerte der Regelgröße abgetastet und digitalisiert.
- diskrete Istwerte, die nach der Amplitude und nach der Zeit quantisiert sind.
- Abtastung periodisch mit der Abtastzeit T_0
- berechnete Stellgröße u ist über den Zeitraum T_0 konstant



$$u(k) = K * e(k)$$

mit $u(k)$ Stellgröße zum Zeitpunkt k
K Verstärkungsfaktor
$e(k)$ Fehler = (Sollwert-Istwert) zum Zeitpunkt k

Vorteile:

- Einfache Implentierung möglich, da der Algorithmus nur den momentanen Fehler $e(k)$ benötigt und keine äquidistanten Zeitpunkte k erfordert
- Einfache Parameter-Bestimmung

Nachteil:

- Störgrößen werden nicht restlos ausgeregelt, es bleibt immer die sogenannte P-Abweichung

7.2.2 Der PI-Regler

- P-Regler mit überlagertem Integral-Anteil
- Gleichung eines PI-Reglers für kontinuierliche Signale:

$$u(t) = K * (e(t) + \frac{1}{T_I}\int_0^t e(\tau)d\tau)$$

nach DIN 19226 mit T_I Integrierzeit (Nachstellzeit)

- für digitale Regelung Gleichung diskretisieren
- für kleine Abtastzeiten T_0 die kontinuierliche Integration durch Rechteck-Integration annähern
- diskretisierte PI-Gleichung:

$$u(k) = K * (e(k) + \frac{T_0}{T_I}\sum_{i=0}^{k-1} e(i))$$

Vorteil:

- Durch I-Anteil keine bleibende Regelabweichung

Nachteile:

- Zeitaufwendige Parameter-Bestimmung
- Einschwingen auf Sollwert
- zeitlich äquidistanten Abtastung notwendig (bei Feldbussen z.B. mit SERCOS, INTERBUS–S garantiert)

7.2.3 Der PID-Regler

- PI-Regler mit überlagertem Differential-Anteil
- Gleichung eines PI-Reglers für kontinuierliche Signale:

$$u(t) = K * (e(t) + \frac{1}{T_I}\int_0^t e(\tau)d\tau + T_D\frac{de(t)}{dt})$$

nach DIN 19226 mit T_D Differenzierzeit (Vorhaltzeit)

- Um den D-Anteil zu diskretisieren, ersetzt man den Differentialquotienten durch eine Differenz erster Ordnung.
- diskretisierte PID-Gleichung („Stellungsform“):

$$u(k) = K*(e(k)+\frac{T_0}{T_I}\sum_{i=0}^{k-1} e(i)+\frac{T_D}{T_0}(e(k)-e(k-1)))$$

- Subtrahiert man von dieser Gleichung $u(k-1)$ erhält man die „Geschwindigkeitsform“

$$u(k) = u(k-1) + K*(e(k)-e(k-1)+\frac{T_0}{T_I}e(k-1)$$
$$+\frac{T_D}{T_0}(e(k) - 2e(k-1) + e(k-2)))$$

Vorteile:

- Durch I-Anteil keine bleibende Regelabweichung
- Schnellen Fehlerveränderungen wird durch den D-Anteil schnell entgegengewirkt

Nachteile:

- Sehr zeitaufwendige Parameter-Bestimmung
- Einschwingen auf Sollwert
- Neigung zur Instabilität

7.2.4 Auslegung der Parameter

Bei allen erwähnten Reglern ergibt sich das Problem, die Parameter der Regelgleichung (K, T_I bzw. T_D) so zu bestimmen, daß

- keine kritischen Schwingungen (Instabilität) auftreten
- der Regelkreis auch nach jeder Störung stabil bleibt
- möglichst wenig Schwingungen beim Einregeln auf den Sollwert auftreten
- Totzeiten des Stellvorgangs, Rechenzeiten und Bandbreite der Stellgrößen berücksichtigt werden
- Auslegung mittels quantitativer Berechnung eines Modells des Systems (Impulsantwort, Laplace-Transformation)
- experimentelle Verfahren zur Parameter-Optimierung (Takahashi)

7.3 Explizites Rechenbeispiel für PID-Regler

- Erläuterungen
  - ⋆ Berechnung mit dem Algebra-System MapleV
  - ⋆ Eingaben entsprechen einer Programmiersprache, wobei Wertezuweisungen wie in Pascal mit := bezeichnet
  - ⋆ Eingabezeilen in roter Schrift mit > eingeleitet
  - ⋆ Ausgabe in blauer Schrift
  - ⋆ Kommentare in schwarzer Schrift
- DGL 2. Ordnung als Regelstrecke

Protokoll der MAPLE-Sitzung:

PID-Regler fuer ein System, das durch eine lineare DGL 2. Ordnung beschrieben wird

eine zusätzlich benötigte Standardprozedur

```
> readlib(unassign);
```

proc() ... end

1. Differentialgleichung

u(t) ist die Stellgroesse des Reglers und k die Rueckfuehrkonstante (Einfluss der Stellgroesse auf den Prozess), xsoll ist der Sollwert

```
> dgl1:=diff(diff(x(t),t),t)+a*diff(x(t),t)
   +b*x(t)+c+k*u(t)= 0;
```

$$dgl1 := \left(\frac{\partial^2}{\partial t^2}x(t)\right) + a\left(\frac{\partial}{\partial t}x(t)\right) + b\,x(t) + c + k\,u(t) = 0$$

```
> u(t):=kp*(x(t)-xsoll)+kd*diff(x(t),t)
   +ki*int(x(z)-xsoll,z=0..t);
```

$$u(t) := kp\,(x(t) - xsoll) + kd\left(\frac{\partial}{\partial t}x(t)\right) + ki\int_0^t x(z) - xsoll\,dz$$

Als neue Variable wird die Groesse y(t) = x(t)-xsoll eingefuehrt

```
> x(t):=y(t)+xsoll;
```

$$x(t) := y(t) + xsoll$$

dann ergibt sich die DGL

```
> simplify(dgl1);
```

$$\left(\frac{\partial^2}{\partial t^2}\mathrm{y}(t)\right)+a\left(\frac{\partial}{\partial t}\mathrm{y}(t)\right)+b\,\mathrm{y}(t)+b\,xsoll+c+k\,kp\,\mathrm{y}(t)+k\,kd\left(\frac{\partial}{\partial t}\mathrm{y}(t)\right)+k\,ki\int_0^t \mathrm{x}(z)-xsoll\,dz=0$$

fasst man gleiche Terme zusammen und setzt

```
> abk:= A=a+k*kd, B=b+k*kp,
   C=c+b*xsoll, KI=k*ki;
```

$$abk := A=a+k\,kd,\ B=b+k\,kp,\ C=c+b\,xsoll,\ KI=k\,ki$$

dann ergibt sich die DGL

```
> dgl2:=diff(diff(y(t),t),t)+A*diff(y(t),t)
   +B*y(t)+C+KI*int(y(z),z=0..t)= 0;
```

$$dgl2 := \left(\frac{\partial^2}{\partial t^2}\mathrm{y}(t)\right)+A\left(\frac{\partial}{\partial t}\mathrm{y}(t)\right)+B\,\mathrm{y}(t)+C+KI\int_0^t \mathrm{y}(z)\,dz=0$$

Anmerkung: A und B koennen durch kd und kp beeinflusst werden, C ist durch den Prozess vorgegeben, KI ist nur bei einem Integralteil des Reglers ungleich Null

2. Loesung bei KI = 0

```
> KI:=0;
```

$$KI := 0$$

```
> LOES1:= dsolve(dgl2,y(t));
```

$$LOES1 := \mathrm{y}(t) = -\frac{C}{B}+\_C1\,e^{\left(-1/2\left(A-\sqrt{A^2-4B}\right)t\right)}+\_C2\,e^{\left(-1/2\left(A+\sqrt{A^2-4B}\right)t\right)}$$

Eigenschaften: _C1 und _C2 sind freie Parameter, die z.B. aus den Anfangswerten bestimmt werden.

Fuer eine stationaere Loesung muss gelten: A positiv (je groesser A ist, desto schneller konvergiert die Loesung) B so waehlen, dass die

Wurzel nahe Null ist, jedenfalls deutlich kleiner als A (falls A*A-4*B negativ , dann wird die Wurzel imaginaer, es kommt zu Schwingungen)

Schlussfolgerung: Da zwei Werte eingestellt werden muessen, ist der gewuenschte Verlauf nur mit einem PD-Regler erreichbar. Ein P-Regler nimmt keinen Einfluss auf den A-Wert. Eine Regelung waere dann nur fuer die Prozesse moeglich, bei denen A schon einen geeigneten Wert hat

3. Beispiele fuer KI = 0

```
> _C1:= 1; _C2:= 1;
```

$$\_C1 := 1$$

$$\_C2 := 1$$

```
> C:= 1;
```

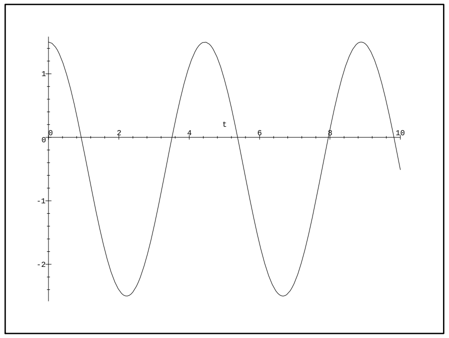
$$C := 1$$

```
> B:= 2;
```

$$B := 2$$

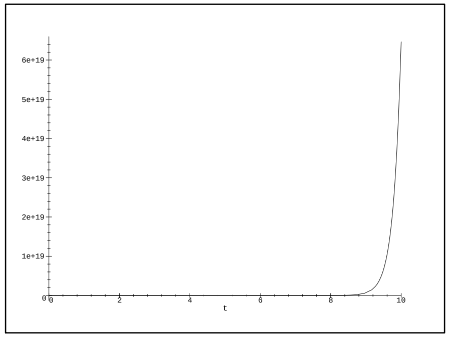

```
> A:=0;plot(op(2,LOES1),t=0..10);
```

$$A := 0$$



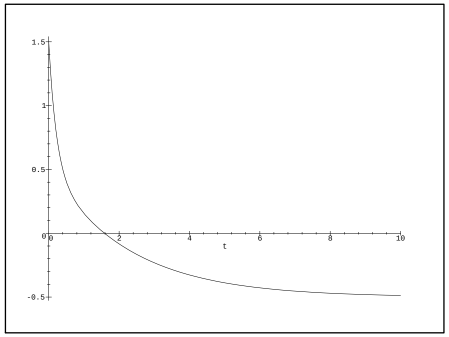

```
> A:=-5;plot(op(2,LOES1),t=0..10);
```

$$A := -5$$




```
> A:=5;plot(op(2,LOES1),t=0..10);
```

$$A := 5$$



in diesem Fall gibt es eine stationaere Loesung y → -C/B = -0.5 die stationaere Loesung ist aber nicht der Sollwert!!

4. Gleichungen fuer KI ungleich Null

```
> unassign('KI','A','B','C');
```

Die Differentialgleichung kann dann ebenfalls mit dem Ansatz y(t)=f(C)*exp(-lambda*t) geloest werden. Es ergibt sich:

```
> LOES2:= dsolve(dgl2,y(t),laplace);
```

$$LOES2 := \mathrm{y}(t) = \sum_{\_r=\%1}\left(\_r^2\,\mathrm{y}(0)+\mathrm{D}(y)(0)\,\_r+A\,\_r\,\mathrm{y}(0)-C\right)e^{(\_r\,t)}\Big/\left(3\,\_r^2+2\,A\,\_r+B\right)$$

$$\%1 := \mathrm{RootOf}(\_Z^3+A\,\_Z^2+B\,\_Z+KI)$$

Man sieht, dass eine Summe ueber e-Funktionen vorliegt, wobei _r die Wurzeln der folgenden Gleichung gl2 sind:

```
> gl2:= z^3+A*z^2+B*z+KI=0;
```

$$gl2 := z^3+A\,z^2+B\,z+KI=0$$

A, B und KI muessen so bestimmt werden, dass der Realteil dieser Wurzeln immer groesser Null ist, dann konvergiert die Loesung immer gegen Null, d.h. den Sollwert fuer den Prozess, da y(t)=x(t)-xsoll. In die Wurzeln geht die Konstante C nicht ein. Aus den gewuenschten Wurzeln lassen sich rueckwaerts kd, kp und ki berechnen.

```
> solve(gl2,z);
```

$$\%1^{1/3}-\%2-\frac{1}{3}A,\ -\frac{1}{2}\%1^{1/3}+\frac{1}{2}\%2-\frac{1}{3}A+\frac{1}{2}I\sqrt{3}\left(\%1^{1/3}+\%2\right),\ -\frac{1}{2}\%1^{1/3}+\frac{1}{2}\%2-\frac{1}{3}A-\frac{1}{2}I\sqrt{3}\left(\%1^{1/3}+\%2\right)$$

$$\%1 := \frac{1}{6}B\,A-\frac{1}{2}KI-\frac{1}{27}A^3+\frac{1}{18}\left(12\,B^3-3\,B^2A^2-54\,B\,A\,KI+81\,KI^2+12\,KI\,A^3\right)^{1/2}$$

$$\%2 := \frac{\frac{1}{3}B-\frac{1}{9}A^2}{\%1^{1/3}}$$

Die Wurzeln haben also die Form rw0, rw±I*iw

Man kann die drei Werte rw0, rw und iw vorgeben, daraus A, B und KI bestimmen und daraus wiederum kd, kp und ki.

5. Beispiele fuer KI ungleich Null

wir setzen (ohne prinzipielle Einschraenkungen der Loesungen)

```
> y(0):=0; D(y)(0):=0; C:=1;
```

$$y(0) := 0$$

$$D(y)(0) := 0$$

$$C := 1$$

```
> A:=2; B:= 3; KI:=2; evalf(solve(gl2,z));
```

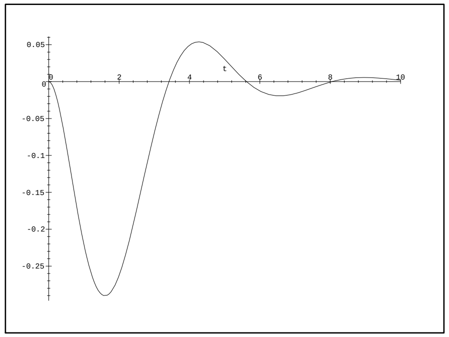
$$A := 2$$

$$B := 3$$

$$KI := 2$$

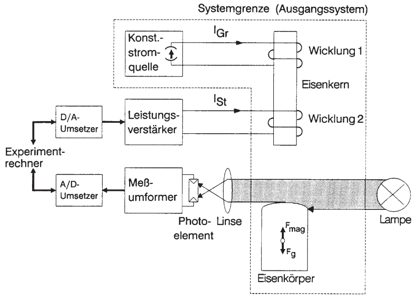
$$-1., -.5000000000 + 1.322875656\,I, -.5000000000 - 1.322875656\,I$$


```
> plot(op(2,LOES2),t=0..10);
```



Dies ist also ein Satz von Parametern mit stationaerer Loesung

7.3.1 Beispiel: Magnetisch aufgehängter Schgebekörper



Systemgleichung:

$$m \cdot \ddot{x} = F_{mag} - F_g;$$

Entwicklung um den Betriebspunkt:

$$m \cdot \ddot{x} = I_{Gr} + f_1 I_{St} + f_2 x - mg;$$

$$m \cdot \ddot{x} + p_1 \cdot \dot{x} + p_2 \cdot x = k I_{St}$$

Explizite Berechnung der Lösung mit Maple V

7.4 Lineare Rückführregelung

- zu regelnder Prozeß durch gekoppeltes System linearer zeitinvarianter homogener Differentialgleichungen beschrieben
  - ⋆ ohne Regelkreis

$$dx/dt = A * x$$

wobei

x: (n)-Vektor der Zustandsvariablen
A: $(n \times n)$-Matrix (konstant)

  - ⋆ mit Regelkreis

$$dx/dt = A * x + B * u$$

wobei

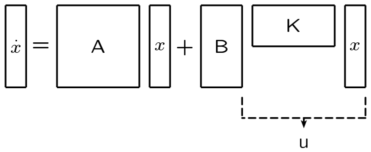
B: $(n \times m)$-Matrix der Stellkoeffizienten (gegeben durch techn. Prozeß)
u: (m)-Vektor der Stellgrößen

  - ⋆ Lineare Rückführregelung

$$u = -K * x$$

wobei

K: $(m \times n)$-Matrix von zu bestimmenden Koeffizienten



- Matrix K so bestimmen, daß Regelungsziel und Nebenbedingungen erfüllt
- Durch Regelung ergibt sich die DGL

$$dx/dt = Ax + B(-Kx) = (A - BK)x$$

- Differentialgleichung hat also durch Einfluß des Reglers neue Eigenwerte
- Lösungsansatz bei linearen DGL

$$x(t) = x0 * e^{\lambda * t}$$

$x0$ ist ein konstanter Vektor, der von den Anfangswerten abhängt

λ ist aus der obigen DGL durch Einsetzen des Lösungsansatzes zu bestimmen

- Lösungsschritte
  - ⋆ eingesetzt

$$\lambda * x0 * e^{\lambda t} = (A - BK) * x0 * e^{\lambda t}$$

  - ⋆ daraus lineares Gleichungssystem für x0

$$(A - BK - \lambda I) * x0 = 0$$

  - ⋆ Lösungen für $x0 \neq 0$ nur, falls Determinante des Gleichungssystems Null, also Forderung:

$$|(A - BK - \lambda I)| = 0$$

  - ⋆ Daraus Bestimmung der Eigenwerte λ_i, mit $i = 1, \ldots, n$
  - ⋆ λ_i i.a. komplexe Zahl, bestehend aus Real- und Imaginärteil
- Lösung
  - ⋆ Durch Addition der verschiedenen Teil-Lösungen

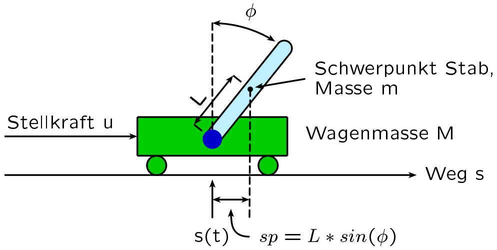
$$x(t) = \sum_{i=1}^{n} x0_i * e^{\lambda_i * t}$$

  - ⋆ $x0_i$ aus den Anfangswerten

- ⋆ Fall 1: $\exists i : \Re(\lambda_i) > 0$
  keine stabile (stationäre) Lösung
- ⋆ Fall 2: $\forall i : \Re(\lambda_i) < 0$:
  stabil, stationäre Lösung
- ⋆ Fall 3: $\forall i : \Re(\lambda_i) = 0$:
  periodische Schwingungen
- ⋆ Spezialfall: $\forall i : \Re(\lambda_i) < 0 \wedge \Im(\lambda_i) = 0$:
  stabil, asymptotische Näherung an den stationären Zustand
- Forderungen an Regler
  - ⋆ K so bestimmen, daß Fall 2 vorliegt
  - ⋆ $|\Re(\lambda_i)|$ sollte möglichst groß sein
  - ⋆ $|\Im(\lambda_i)|$ sollte möglichst klein sein
  - ⋆ die Lösung sollte auch noch bei kleinen Störgrößen Z stabil bleiben

7.5 Anwendungsbeispiel: Balancierter Stab

- Klassisches Beispiel zur Demonstration von Regelungsalgorithmen



- Ausgangsgleichungen

$$(M + m) * \ddot{s} = \text{<Kräfte>}$$

$$\theta * \ddot{\phi} = \text{<Drehmomente>}$$

wobei $\theta = m * L^2/3$ das Trägheitsmoment des Stabes ist

Es ergeben sich 4 Gleichungen (nach Ackermann)

(1) Horizontalbewegung des Stabschwerpunktes

$$m * \frac{d^2}{dt^2}(s + L * sin\phi) = H$$

(2) Vertikalbewegung des Stabschwerpunktes

$$m * \frac{d^2}{dt^2}(L * cos\phi) = V - m * g$$

(3) Drehbewegung des Stabs um seinen Schwerpunkt

$$\theta * \frac{d^2\phi}{dt^2} = V * L * sin\phi - H * L * cos\phi$$

(4) Horizontalbewegung des Wagens

$$M * \frac{d^2 s}{dt^2} = u - H$$

für die 4 Unbekannten

- ⋆ Horizontalkraft H des Wagens auf den Stab
- ⋆ Vertikalkraft V des Wagens auf den Stab
- ⋆ Beschleunigung $\ddot{s}$
- ⋆ Winkelbeschleunigung $\ddot{\phi}$

- Durch Linearisierung der Winkelfunktionen (kleine ϕ bzw. $\dot{\phi}$) erhält man ein lineares DGL–System. Die Koeffizienten a_{21}, a_{41}, b_2 und b_4 sind Funktionen von L, M und m.

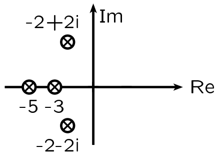
$$\frac{d}{dt}\begin{pmatrix}\phi\\ \dot{\phi}\\ s\\ \dot{s}\end{pmatrix} = \begin{pmatrix}0 & 1 & 0 & 0\\ a_{21} & 0 & 0 & 0\\ 0 & 0 & 0 & 1\\ a_{41} & 0 & 0 & 0\end{pmatrix} * \begin{pmatrix}\phi\\ \dot{\phi}\\ s\\ \dot{s}\end{pmatrix} + \begin{pmatrix}0\\ b_2\\ 0\\ b_4\end{pmatrix} * u$$

- Regelungsziel
  ϕ, $\dot{\phi}$ möglichst schnell oder mit kleiner Kraft/Arbeit gegen Null führen
- gemessen wird nur s und ϕ; $\dot{s}$ und $\dot{\phi}$ werden berechnet (sog. "Beobachter")
- Zahlenwerte für die Beispiele
  M: 0.981 kg
  m: 0.08 kg
  L: 0.312 m
- Ergebnis ohne Regler, d.h. $\dot{x} = A * x + Z$
  - ⋆ $\lambda_1 = 0,\ \lambda_2 = 0,\ \lambda_3 = -5,\ \lambda_4 = +5$
  - ⋆ Stab bleibt bei Z=0 senkrecht, falls dies Ausgangslage
  - ⋆ bei Störungen (Z ≠ 0) kippt Stab (ϕ wächst unbegrenzt wegen Linearisierung)

- Ergebnis mit PD-Regler
  - ⋆ Anmerkung: K hat im Beispiel vier Komponenten, da es eine PD-Regelung realisiert und sowohl den Winkel als auch den Ort korrigiert, also

$$u = K1 * \phi + K2 * \dot{\phi} + K3 * s + K4 * \dot{s}$$

  - ⋆ K so bestimmen, daß
    - ∘ Prozeß stabil, d.h. daß die λ_i die gewünschten Werte haben
    - ∘ u technisch sinnvoll machbar ist
  - ⋆ gewünschte Lage der λ_i im Beispiel

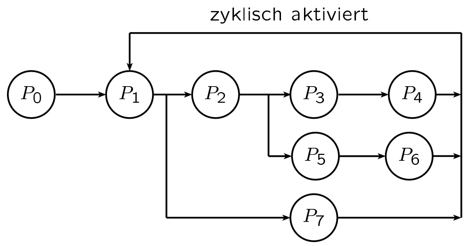


  - ⋆ dann ergibt sich

$$K = (35,39; 7,174; 5,088; 5,258)$$

  - ⋆ K so wählen, daß auch bei kleinen Störungen Z die λ (ungefähr) im gewünschten Bereich liegen

- Prozesse zur Regelung des balancierten Stabs
  - ⋆ P0: System initialisieren; neue Vorgaben vom Leitstand
  - ⋆ P1: Meßwerte (s und ϕ) einlesen; zyklische Aktivierung (Zykluszeit t_m)
  - ⋆ P2: Beobachterprozeß; berechnet $\dot{s}$ und $\dot{\phi}$
  - ⋆ P3: Stellgröße u berechnen
  - ⋆ P4: u ausgeben
  - ⋆ P5:ϕ und $\dot{\phi}$ für Bildschirm (Prozeßwarte) aufbereiten
  - ⋆ P6: Bildschirmausgabe
  - ⋆ P7: Regelungsprotokoll führen
  - ⋆ Präzedenzgraph



7.6 Maple-Sheet für balancierten Stab

Maple: Inverses Pendel (balancierter Stab) nach Ackermann

1. Die DGL

Horizontalbewegung des Stabschwerpunktes

```
> gl01:= m*diff(diff(s(t)+L*Sphi(t),t),t) =H;
```

$$gl01 := m\left(\left(\frac{\partial^2}{\partial t^2}\mathrm{s}(t)\right) + L\left(\frac{\partial^2}{\partial t^2}\mathrm{Sphi}(t)\right)\right) = H$$

Vertikalbewegung des Stabschwerpunktes

```
> gl02:=m*diff(diff(L*Cphi(t),t),t) = V - m*g;
```

$$gl02 := m\,L\left(\frac{\partial^2}{\partial t^2}\mathrm{Cphi}(t)\right) = V - m\,g$$

Drehbewegung des Stabs um seinen Schwerpunkt

```
> gl03:=theta*diff(diff(phi(t),t),t) -V*L*Sphi(t)+
  H*L*Cphi(t)=0;
```

$$gl03 := \theta\left(\frac{\partial^2}{\partial t^2}\phi(t)\right) - V\,L\,\mathrm{Sphi}(t) + H\,L\,\mathrm{Cphi}(t) = 0$$

Horizontalbewegung des Wagens

```
> gl04:=M*diff(diff(s(t),t),t) - u+H=0;
```

$$gl04 := M\left(\frac{\partial^2}{\partial t^2}\mathrm{s}(t)\right) - u + H = 0$$

Traegheitsmoment

```
> theta:=m*L*L/3;
```

$$\theta := \frac{1}{3}m\,L^2$$

```
> Cphi(t) := cos(phi(t));
```

$$\mathrm{Cphi}(t) := \cos(\phi(t))$$

```
> Sphi(t):=sin(phi(t));
```

$$\mathrm{Sphi}(t) := \sin(\phi(t))$$

gegeben: m (Masse Stab), M (Masse Wagen), L (Laenge Stab) unbekannt: s(t), phi(t), H, V

2. Vereinfachte Differentialgleichungen

Naeherung

```
> Cphi(t):=1;
```

$$\mathrm{Cphi}(t) := 1$$


```
> Sphi(t):= phi(t);
```

$$\mathrm{Sphi}(t) := \phi(t)$$

damit wird:

```
> gl01;
```

$$m\left(\left(\frac{\partial^2}{\partial t^2}\mathrm{s}(t)\right) + L\left(\frac{\partial^2}{\partial t^2}\phi(t)\right)\right) = H$$

```
> gl02;
```

$$0 = V - m\,g$$

```
> gl03;
```

$$\frac{1}{3}m\,L^2\left(\frac{\partial^2}{\partial t^2}\phi(t)\right) - V\,L\,\phi(t) + H\,L = 0$$

```
> gl04;
```

$$M\left(\frac{\partial^2}{\partial t^2}\mathrm{s}(t)\right) - u + H = 0$$

wir loesen jetzt gl01 nach H und gl02 nach V auf:

```
> H:=solve(gl01,H);
```

$$H := m\left(\frac{\partial^2}{\partial t^2}\mathrm{s}(t)\right) + m\,L\left(\frac{\partial^2}{\partial t^2}\phi(t)\right)$$

```
> V:=solve(gl02,V);
```

$$V := m\,g$$

eingesetzt in gl03 und gl04 ergibt sich:

```
> gl031:=simplify(gl03);
```

$$gl031 := \frac{4}{3}m\,L^2\left(\frac{\partial^2}{\partial t^2}\phi(t)\right) - m\,g\,L\,\phi(t) + L\,m\left(\frac{\partial^2}{\partial t^2}\mathrm{s}(t)\right) = 0$$

```
> gl041:=simplify(gl04);
```

$$gl041 := M\left(\frac{\partial^2}{\partial t^2}\mathrm{s}(t)\right) - u + m\left(\frac{\partial^2}{\partial t^2}\mathrm{s}(t)\right) + m\,L\left(\frac{\partial^2}{\partial t^2}\phi(t)\right) = 0$$

gl031 und gl041 sind die beiden DGL fuer die beiden Unbekannten s(t) und phi(t)

3. Reglergleichung

> u:= KP*phi(t)+KD*diff(phi(t),t);

$$u := KP\,\phi(t) + KD\left(\frac{\partial}{\partial t}\phi(t)\right)$$

damit wird:

> gl032:=simplify(gl031);

$$gl032 := \frac{4}{3} m L^2 \left(\frac{\partial^2}{\partial t^2}\phi(t)\right) - m\,g\,L\,\phi(t) + L\,m\left(\frac{\partial^2}{\partial t^2}s(t)\right) = 0$$

> gl042:=simplify(gl041);

$$gl042 := M\left(\frac{\partial^2}{\partial t^2}s(t)\right) - KP\,\phi(t) - KD\left(\frac{\partial}{\partial t}\phi(t)\right) + m\left(\frac{\partial^2}{\partial t^2}s(t)\right) + m\,L\left(\frac{\partial^2}{\partial t^2}\phi(t)\right) = 0$$

4. Loesung

Loesungsansatz

> phi(t):=cp*exp(lambda*t);

$$\phi(t) := cp\,e^{(\lambda t)}$$

> s(t):=cs*exp(lambda*t);

$$s(t) := cs\,e^{(\lambda t)}$$

eingesetzt in die beiden Gleichungen und dividiert durch exp(lambda*t) ergibt sich ein homogenes Gleichungssystem fuer die Unbekannten cp und cs

> gl033:=simplify(op(1,gl032)/exp(lambda*t))=0;

$$gl033 := \frac{1}{3} L\,m\left(4\,L\,cp\,\lambda^2 - 3\,g\,cp + 3\,cs\,\lambda^2\right) = 0$$

> gl043:=simplify(op(1,gl042)/exp(lambda*t))=0;

$$gl043 := M\,cs\,\lambda^2 - KP\,cp - KD\,cp\,\lambda + m\,cs\,\lambda^2 + L\,m\,cp\,\lambda^2 = 0$$

homogene Gleichung fuer cp und cs, nur Loesung, falls Determinante 0 in diesem Fall sind die Loesungen voneinander abhaengig

> solve(gl033,cp);

$$-3\,\frac{cs\,\lambda^2}{4\,L\,\lambda^2 - 3\,g}$$

Prozedur dafuer

> Cp:=proc(lam,cs) global L,g;

> -3*cs*lam*lam/(4*L*lam*lam-3*g);

> end;

```
Cp := proc(lam,cs)
global L,g;
 -3*cs*lam^2/(4*L*lam^2-3*g)
 end
```


die Matrixelemente der Gleichungen lauten

> a11:= L*m/3*(4*L*lambda*lambda-3*g);

$$a11 := \frac{1}{3} L\,m\left(4\,L\,\lambda^2 - 3\,g\right)$$

> a12:=L*m/3*3*lambda*lambda;

$$a12 := L\,m\,\lambda^2$$

> a21:=-KP-KD*lambda+L*m*lambda*lambda;

$$a21 := -KP - KD\,\lambda + L\,m\,\lambda^2$$

> a22:=M*lambda*lambda+m*lambda*lambda;

$$a22 := M\,\lambda^2 + m\,\lambda^2$$

damit ergibt sich die Determinante

> detgl:=collect(a11*a22-a12*a21=0,lambda);

$$detgl := \left(\frac{4}{3} m\,L^2\,(M+m) - L^2\,m^2\right)\lambda^4 + L\,m\,KD\,\lambda^3 + (-m\,g\,L\,(M+m) + L\,m\,KP)\,\lambda^2 = 0$$

Loesungen sind

> lam:=solve(detgl,lambda);

$$lam := 0, 0, \frac{1}{2}\left(-L\,m\,KD + \frac{1}{3}(9\,L^2\,m^2\,KD^2 + 48\,m^2\,L^3\,M^2\,g + 60\,m^3\,L^3\,M\,g - 48\,m^2\,L^3\,M\,KP + 12\,L^3\,m^4\,g - 12\,L^3\,m^3\,KP)^{1/2}\right)\Big/\left(\frac{4}{3}m\,L^2\,M + \frac{1}{3}L^2\,m^2\right),$$
$$\frac{1}{2}\left(-L\,m\,KD - \frac{1}{3}(9\,L^2\,m^2\,KD^2 + 48\,m^2\,L^3\,M^2\,g + 60\,m^3\,L^3\,M\,g - 48\,m^2\,L^3\,M\,KP + 12\,L^3\,m^4\,g - 12\,L^3\,m^3\,KP)^{1/2}\right)\Big/\left(\frac{4}{3}m\,L^2\,M + \frac{1}{3}L^2\,m^2\right)$$

> lambda1:=lam[1];

$$\lambda 1 := 0$$

> lambda2:=lam[2];

$$\lambda 2 := 0$$

> lambda3:=simplify(lam[3]);

$$\lambda 3 := -\frac{1}{2}\left(3\,L\,m\,KD - \sqrt{3}(L^2\,m^2(3\,KD^2 + 16\,L\,M^2\,g + 20\,m\,g\,L\,M - 16\,L\,M\,KP + 4\,m^2\,g\,L - 4\,L\,m\,KP))^{1/2}\right)\Big/(m\,L^2\,(4\,M + m))$$

> lambda4:=simplify(lam[4]);

$$\lambda 4 := -\frac{1}{2}\left(3\,L\,m\,KD + \sqrt{3}(L^2\,m^2(3\,KD^2 + 16\,L\,M^2\,g + 20\,m\,g\,L\,M - 16\,L\,M\,KP + 4\,m^2\,g\,L - 4\,L\,m\,KP))^{1/2}\right)\Big/(m\,L^2\,(4\,M + m))$$

die Gesamtloesung setzt sich aus den Teilloesungen zusammen

winkel berechnet phi(t), abhaengig von weiteren frei waehlbaren Parametern

```
> winkel:=proc(t,Cs1,Cs2,Cs3,kp,kd)
   global KP,KD,tit;
    local wu,rwu,iwu;
> KP:=kp;
> KD:=kd;
> wu:=evalf(lambda3);
> rwu:=Re(wu);
> iwu:=Im(wu);
> tit:=cat(`phi(t) fuer KP=`,convert(KP,string),
   ` KD=`,convert(KD,string),
   ` rwu=`,convert(rwu,string),
   ` iwu=`,convert(iwu,string));
> Re(evalf(Cp(lambda1,Cs1)*exp(lambda1*t)+
   Cp(lambda2,Cs2)*exp(lambda2*t)+
   Cp(lambda3,Cs3)*exp(lambda3*t)+
   Cp(lambda4,Cs1)*exp(lambda4*t)));
> end;
```



```
winkel :=
 proc(t,Cs1,Cs2,Cs3,kp,kd)
 local wu,rwu,iwu;
 global KP,KD,tit;
     KP := kp;
     KD := kd;
     wu := evalf(lambda3);
     rwu := Re(wu);
     iwu := Im(wu);
     tit := ...
     Re(evalf(Cp(lambda1,Cs1)*
       exp(lambda1*t)+ ...
 end
```

5. Berechnung der Werte KP und KD fuer vorgegebene Wurzeln rw ± I*iw

```
> lambda3;
```

$$-\frac{1}{2}\left(3\,L\,m\,KD-\sqrt{3}(L^2\,m^2(3\,KD^2+16\,L\,M^2\,g+20\,m\,g\,L\,M-16\,L\,M\,KP+4\,m^2\,g\,L-4\,L\,m\,KP))^{1/2}\right)\Big/(m\,L^2\,(4\,M+m))$$

```
> realteil:=rw=-1/2*3*L*m*KD/
  (m*L*L*(4*M+m));
```

$$realteil := rw = -\frac{3}{2}\frac{KD}{L\,(4\,M+m)}$$

```
> kd:=solve(realteil,KD);
```

$$kd := -\frac{2}{3}\,rw\,L\,(4\,M+m)$$


```
> imteil:= iw*iw=-1/4*3*L*L*m*m*(
  3*KD*KD+16*L*M*M*g+20*m*g*L*M-
  16*L*M*KP+4*m*m*g*L-4*L*m*KP)/
  (m*L*L*(4*M+m))^2;
```

$$imteil := iw^2 = -\frac{3}{4}(3\,KD^2+16\,L\,M^2\,g+20\,m\,g\,L\,M-16\,L\,M\,KP+4\,m^2\,g\,L-4\,L\,m\,KP)\Big/\left(L^2\,(4\,M+m)^2\right)$$

```
> kp:=solve(imteil,KP);
```

$$kp := -\left(iw^2+\frac{9}{4}\frac{KD^2}{L^2\,(4\,M+m)^2}+12\frac{M^2\,g}{L\,(4\,M+m)^2}+15\frac{m\,g\,M}{L\,(4\,M+m)^2}+3\frac{m^2\,g}{L\,(4\,M+m)^2}\right)\Big/\left(-12\frac{M}{L\,(4\,M+m)^2}-3\frac{m}{L\,(4\,M+m)^2}\right)$$

6. Beispiele

zunaechst eine Berechnung von phi(t) bei verschiedenen Parametern, u.a. dem Realteil und dem Imaginaerteil der Wurzeln (lambda)

```
> winkel2:=proc(t,Cs1,Cs2,Cs3,rwi,iwi)
   global rw,iw,KP,KD; local lkp,lkd;
> rw:=rwi;
> iw:=iwi;
> lkd:=kd;
> KD:=lkd;
> lkp:=kp;
> winkel(t,Cs1,Cs2,Cs3,lkp,lkd);
> end;
```



```
winkel2 := proc(t,Cs1,Cs2,Cs3,rwi,iwi)
    local lkp,lkd;
    global rw,iw,KP,KD;
        rw := rwi;
        iw := iwi;
        lkd := kd;
        KD := lkd;
        lkp := kp;
        winkel(t,Cs1,Cs2,Cs3,lkp,lkd)
end
```

jetzt feste Zahlenwerte

```
> M:=0.981;
```

$$M := .981$$

```
> m:=0.08;
```

$$m := .08$$


```
> L:=0.312;
```

$$L := .312$$

```
> g:=9.81;
```

$$g := 9.81$$

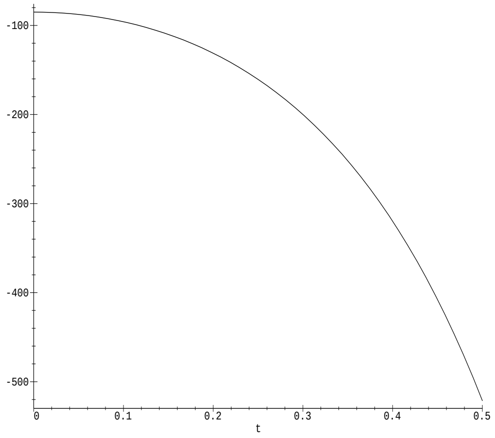
Ausgabeprozedur

```
> aus:=proc(ta,te,cs1,cs2,cs3,lrw,liw);
> print(plot(winkel2(t,cs1,cs2,cs3,lrw,liw),
   t=ta..te,title=tit,axes=NORMAL));
> end;
```

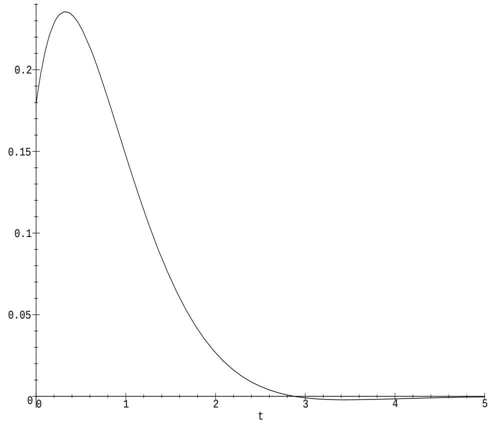
```
aus :=
proc(ta,te,cs1,cs2,cs3,lrw,liw)
    print(
    plot(winkel2(t,cs1,cs2,cs3,lrw,liw),
         t = ta .. te,title = tit,
         axes = NORMAL)
    )
end
```


Ausgaben fuer ausgewaehlte Faelle

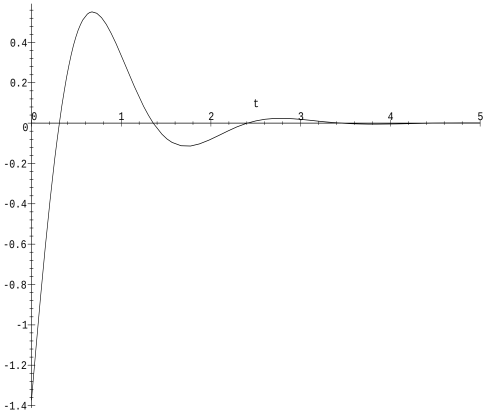
```
> plot(winkel(t,1,1,1,0,0),t=0..1,title=tit,
   axes=NORMAL);
> plot(winkel2(t,1,1,1,-1.5,2),t=0..0.5,title=tit,
   axes=NORMAL);
> aus(0, 5, 1, 1, 1, -1.5, 1.0);
> aus(0, 5, 1, 1, 1, -1.5, 2.0);
> aus(0, 5, 1, 1, 1, -1.5, 3.0);
> aus(0, 5, 1, 1, 1, -3.0, 1.0);
> aus(0, 5, 1, 1, 1, -3.0, 20.0);
```

- $\phi(t)$ für ungeregelten balancierten Stab
  - ⋆ nicht stabil
  - ⋆ wegen Linearisierung ist ϕ nicht begrenzt



- $\phi(t)$ für geregelten balancierten Stab
  - ⋆ wurzeln: 0, -1.5 ± 1*I
  - ⋆ stabil, Überschwingen
  - ⋆ wegen kleinem Imaginärteil der Wurzel lange Schwingungsperiode



- $\phi(t)$ für geregelten balancierten Stab
  - ⋆ wurzeln: 0, -1.5 ± 3*I
  - ⋆ stabil, Überschwingen
  - ⋆ wegen größerem Imaginärteil der Wurzel kürzere Schwingungsperiode

7.7 Einführung in die Fuzzy-Logik

- Definition
  Eine unscharfe Menge (fuzzy set) $\tilde{A}$ über der scharfen (crisp) Menge X wird durch eine Zugehörigkeitsfunktion $\mu(x, X, \tilde{A})$ charakterisiert, die jedem $x \in X$ eine reelle Zahl aus [0, 1] zuordnet: die Wahrscheinlichkeit, mit der das Element der unscharfen Menge angehört
- Unscharfe Menge

$$\tilde{A} = \left\{ \left(x_1, \mu(x_1, X, \tilde{A})\right), \ \ldots, \ \left(x_n, \mu(x_n, X, \tilde{A})\right) \right\}$$

- Scharfe Menge $\widetilde{X}$

$$\mu(x, X, \widetilde{X}) = 1 \qquad \forall x \in X$$

- Trägermenge (support) von $\tilde{A}$

$$supp(\tilde{A}) \equiv \left\{ x \in X \mid \mu(x, X, \tilde{A}) > 0 \right\}$$

- α–Schnitt(α–level–set, α–cut)

$$\tilde{A}_\alpha = \left\{ \left(x, \mu(x, X, \tilde{A})\right) \mid x \in X \wedge \mu(x, X, \tilde{A}) \geq \alpha \right\}$$

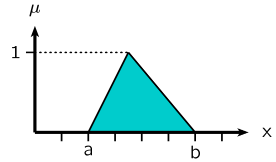
- gekappte unscharfe Menge $\tilde{A} \uparrow \alpha$
  Sei $\tilde{A}$ über der Scharfen Menge X definiert durch $\mu(x, X, \tilde{A})$, dann gilt für alle $x \in X$

$$\mu(x, X, \tilde{A} \uparrow \alpha) =$$

$$\begin{cases} \mu(x, X, \tilde{A}), & falls \quad \mu(x, X, \tilde{A}) \leq \alpha \\ \alpha, & falls \quad \mu(x, X, \tilde{A}) > \alpha \end{cases}$$

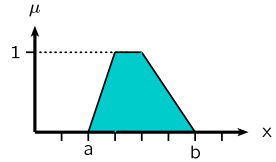
- Ein wesentlicher Teil des Entwurfsprozesses liegt in der Festlegung der Form und Werte der Zugehörigkeitsfunktionen μ

- Programme bieten einfache Formen an, z.B.
  - ⋆ Dreieck



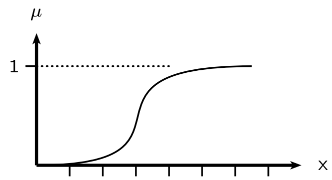
häufig gleichschenklig

  - ⋆ Trapez

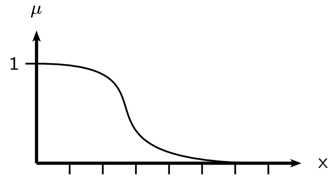


meist symmetrisch

- ⋆ "S"-Funktion



- ⋆ "Z"-Funktion



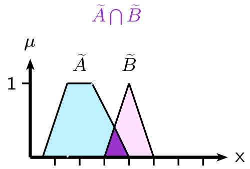
- ⋆ Üblich sind auch Polygonzug, Glockenkurve, $\sin^2$ usw.

- Operationen
  - ⋆ Benötigt werden Durchschnitt, Vereinigung, Komplement der unscharfen Mengen
  - ⋆ Definition dieser Operationen frei an Anwendung anpaßbar, solange für $\mu = 0$ und $\mu = 1$ die üblichen Logikoperationen gelten bzw. allgemeiner bestimmte Normen (t–Norm, s–Norm) erfüllt sind
- Übliche Operationen für die Regelungstechnik
  - ⋆ Durchschnitt

$$\tilde{C} = \tilde{A} \cap \tilde{B}$$

wobei

$$\mu(x, X, \tilde{C}) = min\left(\mu(x, X, \tilde{A}), \mu(x, X, \tilde{B})\right)$$

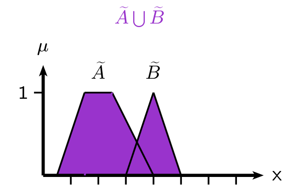


- ⋆ Vereinigung

$$\tilde{C} = \tilde{A} \cup \tilde{B}$$

wobei

$$\mu(x, X, \tilde{C}) = max\left(\mu(x, X, \tilde{A}), \mu(x, X, \tilde{B})\right)$$



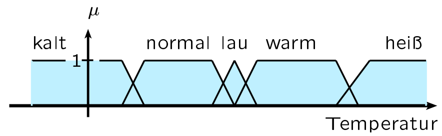
- ⋆ Komplement

$$\tilde{C} = \overline{\tilde{A}}$$

wobei

$$\mu(x, X, \tilde{C}) = \left(1 - \mu(x, X, \tilde{A})\right) \quad \forall x \in X$$

- Linguistische Variable
  - ⋆ Eine linguistische Variable (L.V.) ist charakterisiert durch ihren Namen v und ihre Werte (Ausprägungen)
  - ⋆ der L.V. ist eine scharfe Menge X zugeordnet
  - ⋆ Menge der Werte (linguistische Terme, L.T.) von v ist Termset $\mathcal{T}(v)$
  - ⋆ jeder linguistische Term ist eine unscharfe Menge definiert über der scharfen Menge X
  - ⋆ Beispiel Temperatur
    - ∘ L.V. ist Temperatur
    - ∘ zugehörige scharfe Menge ist Menge der Temperaturwerte
    - ∘ zugehörige L.T. sind: kalt, normal, lau, warm, heiß
    - ∘ Zugehörigkeitsfunktionen



7.8 Fuzzy–Regelung

- weit verbreitet
  - ⋆ insbesondere in Konsumgütern, wie
  - ⋆ Waschmaschinen
  - ⋆ Fotoapparate
- Warum?
  - ⋆ mathematisches Modell für Prozeß nicht vorhanden, zu schwierig
  - ⋆ Fuzzy–Regelung stellt nur geringe mathematische Anforderungen
- Der dem Menschen (aus Erfahrung) bekannte Regelungsvorgang wird in WENN–DANN–Regeln mit linguistischen Variablen formuliert
- Prämissen und Konklusion werden "unscharf" ausgewertet

- Entwurfschritte

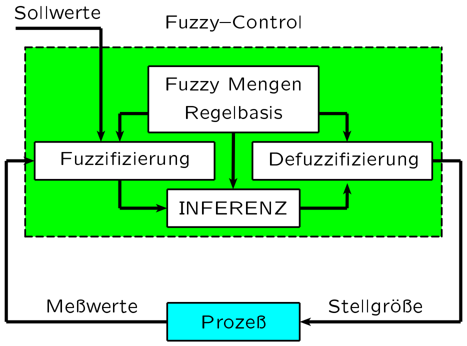
(1) Fuzzifizierung der Meßwerte: Festlegung der linguistischen Variablen und ihrer Terme mit entsprechenden Zugehörigkeitsfunktionen

(2) Erstellen der Regelbasis (Expertenwissen)

(3) Auswahl geeigneter Inferenzoperationen

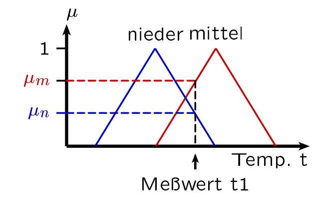
(4) Defuzzifizierung: Berechnung der scharfen Stellgrößen

- Struktur Fuzzy-Regler



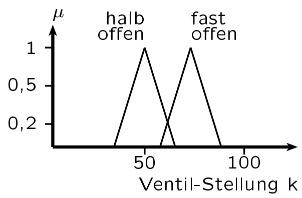
- Die einzelnen Entwurfsschritte am Beispiel erklärt
  - ⋆ nach H.-H. Bothe: Fuzzy–Logic. Springer 1993
  - ⋆ Aufgabe: Kühlventilstellung regeln
  - ⋆ Meßwert: Temperatur t1
  - ⋆ Stellwert: Stellung k_s des Kühlventils
- zu (1): Prinzip Fuzzifizierung
  - ⋆ jeder Meßwert stammt aus einer scharfen Menge (L.V.)
  - ⋆ zu allen linguistischen Variablen sind unscharfe Mengen (L.T.) definiert (menschliche Erfahrung, "trial and error")
  - ⋆ für jeden Meßwert werden die Zugehörigkeitswahrscheinlichkeiten zu den L.T. ermittelt
- zu (1): Beispiel Fuzzifizierung
  - ⋆ Temperatur t1 gemessen
  - ⋆ es gibt zwei scharfe Mengen (L.V.)
    - ∘ Temperatur T
    - ∘ Stellung K des Kühlventils

- ⋆ unscharfe Mengen der L.V. Temperatur T (unvollständig)



wobei
$\mu_n = \mu(t1, T, \widetilde{nieder})$
$\mu_m = \mu(t1, T, \widetilde{mittel})$

- ⋆ unscharfe Mengen der L.V. Ventil-Stellung K (unvollständig)



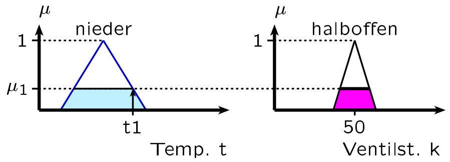
- zu (2): Prinzip Regelbasis
  - ⋆ Elementarbedingung der Form $(V = \tilde{A})$, wobei V eine L.V. (Gleichheitszeichen ist als "ist" zu lesen)
  - ⋆ Verknüpfung der Elementarbedingungen mit AND, OR, NOT
  - ⋆ Aktion der Form $(W = \tilde{B})$, wobei W eine L.V.
  - ⋆ Form der Regel j z.B:
    IF $(V_1 = \tilde{A}_{1j})$ AND
    $(V_2 = \tilde{A}_{2j})$ OR
    $(V_3 = \tilde{A}_{3j})$
    THEN $W = \tilde{B}_j$
  - ⋆ V_1, V_2, V_3, W sind L.V.
  - ⋆ $\tilde{A}_{1j}$, $\tilde{A}_{2j}$, $\tilde{A}_{3j}$, $\tilde{B}_j$ sind L.T. der entsprechenden L.V.
- zu (2): Beispiel Regelbasis
  - ⋆ R1: if (T=$\widetilde{nieder}$) THEN K = $\widetilde{halboffen}$
  - ⋆ R2: if (T=$\widetilde{mittel}$) THEN K = $\widetilde{fastoffen}$

- zu (3): Prinzip Inferenz
  - ⋆ für jede Elementarbedingung $(X = \tilde{A})$ wird bei gemessenem Wert $x \in X$ berechnet $\mu(x, X, \tilde{A})$
  - ⋆ Verknüpfung Elementarbedingungen durch spezifische Strategien
  - ⋆ Verknüpfung in Regelungstechnik nach Mamdami:

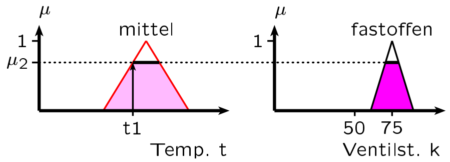
AND	als Durchschnitt (Minimum μ–Werte)
OR	als Vereinigung (Maximum μ–Werte)
NOT	als Komplement $(1 - \mu)$

  - ⋆ damit μ_j für Regel j aus Bedingungsteil berechenbar
  - ⋆ μ_j beschränkt den μ–Wert der Konklusions–Variablen $\tilde{B}_j$ der rechten Seite; es wird berechnet die gekappte unscharfe Menge $\tilde{B}_j \uparrow \mu_j$

- zu (3): Beispiel Inferenz
  - ⋆ R1 liefert $\mu_1 = \mu(t1, T, \widetilde{nieder})$ und daraus $\widetilde{halboffen} \uparrow \mu_1$



  - ⋆ R2 liefert $\mu_2 = \mu(t1, T, \widetilde{mittel})$ und daraus $\widetilde{fastoffen} \uparrow \mu_2$



- zu (4): Prinzip Defuzzifizierung
  - ⋆ aus den gekappten unscharfen Mengen bezüglich einer linguistischen Variablen wird die Vereinigungsmenge gebildet
  - ⋆ mehrere Verfahren zur Berechnung scharfer Stellwerte aus gekappten unscharfen Mengen aus Konklusionen in Schritt 3
    - ∘ Mittelwertbildung (mean of maxima)
    - ∘ Mittelwert der Maximalwerte
    - ∘ Schwerpunktbildung der Fläche (center of area/gravity)
    - ∘ gewichtete Schwerpunktbildung der Fläche, falls die Regeln gewichtet werden

- zu (4): Formeln
  - ⋆ Bezeichnungen:
    x ein Wert des Ausgabebereichs (Stellwert)
    $\mu(x)$ Zugehörigkeitfunktion zu der oben genannten Vereinigungsmenge
    g_j Gewicht der Regel j
    μ_j Ergebnis der linken Seite der Regel j
    $F(\widetilde{A})$ Fläche der Zugehörigkeitsfunktion der unscharfen Menge
  - ⋆ x-Koordinate x_s des Schwerpunkts

$$x_s = \int x\,\mu(x)\,dx \;/\; \int \mu(x)\,dx$$

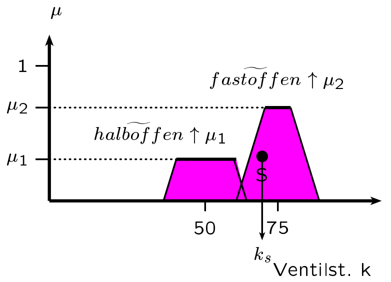
  - ⋆ Schwerpunkt bei Massenpunkten i mit Fläche F_i und x-Koordinate x_{s_i} des Schwerpunkts

$$x_s = \sum_i x_{s_i} F_i \;/\; \sum_i F_i$$

  - ⋆ x-Koordinate x_s^* des Schwerpunkts bei gewichteten Regeln j = 1, 2, ... , n

$$x_s^* = \sum_{j=1}^{n} x_{s_j} F(\widetilde{B_j} \uparrow \mu_j)\mu_j g_j \;/\; \sum_{j=1}^{n} F(\widetilde{B_j} \uparrow \mu_j)\mu_j g_j$$

- zu (4): Beispiel Defuzzifizierung
  - ⋆ Schwerpunktbildung der gekappten unscharfen Mengen $\widetilde{halboffen} \uparrow \mu_1$ und $\widetilde{fastoffen} \uparrow \mu_2$
  - ⋆ Schwerpunkt sei $S = (k_s, \mu_s)$
  - ⋆ Ventil ist auf k_s einzustellen



- Hinweis: Es gibt adaptive Fuzzy-Regelungen
  - ⋆ Adaption der Gewichte der Regeln
  - ⋆ Veränderung der Neigung der Zugehörigkeitsfunktionen der linguistischen Terme innerhalb vorgegebener Grenzen

7.9 Fuzzy–Regelung eines aufrechten (inversen) Pendels (balancierter Stab)

- klassische schwierige Regelungsaufgabe
- in der Literatur mehrere Beschreibungen mittels Fuzzy–Regelung
- Linguistische Variable
  - ⋆ Winkel ϕ
    mit L.T.: neg, zero, pos
  - ⋆ Winkelbeschleunigung $\dot{\phi}$
    mit L.T.: neg, zero, pos
  - ⋆ Stellkraft U
    mit L.T.: grossneg, neg, zero, pos, grosspos

- 9 Regeln durch AND–Verknüpfung von ϕ und $\dot{\phi}$
  z.B.
  IF (ϕ = neg) AND ($\dot{\phi}$ = pos) THEN U = pos
- Satz von Regeln durch Matrixdarstellung übersichtlicher

$\dot{\phi}$ \ ϕ	neg	zero	pos
neg	grosspos	pos	neg/zero
zero	pos	zero	neg
pos	pos/zero	neg	grossneg

7.10 Fuzzy–Regler als Chips

- Übliche Aufgaben benötigen oft nur Regler mit 2 Eingängen (x und dx/dt) und einer Ausgangsgröße (2 bis 4 msec)XS
- Hierfür spezielle Chips am Markt
- Probleme heute teilweise noch im Preis, da die Chips im Masseneinsatz (z.B. Kfz) sehr billig sein müssen (weniger als DM 10.–)
- zeitkritische Fuzzy–Regelungen benötigen Hochleistungschips als Co-Prozessoren
  - ⋆ Hochleistung durch interne parallele Regelauswertung
  - ⋆ bis zu je 8 Ein– und Ausgangsgrößen
  - ⋆ bis zu 16 Punkten pro Zugehörigkeitsfunktion (max. 7 Zugehörigkeits–Funktionen pro Variable)
  - ⋆ 25 bis zu 256 Regeln
  - ⋆ Auswertungszeiten im Bereich von 0,1 bis 50 msec

- Vorreiter 1989: Togai FC–110
- Hochleistungs-Fuzzychips, z.B.
  OMRON FP-3000, SIEMENS SAE 81C99
- OMRON FP-3000
  - ⋆ 8 Eingänge
  - ⋆ 8 Ausgänge
  - ⋆ maximal 96 Regeln
  - ⋆ 3 msec für Auswertung von 96 Regeln
- SAE 81C99
  - ⋆ 256 Eingänge
  - ⋆ 64 Ausgänge
  - ⋆ maximal 16384 Regeln
  - ⋆ 7,9 Mio Regeln je Sekunde
    etwa 1,3 μsec je Regel

8 Echtzeitplanung (real-time scheduling)

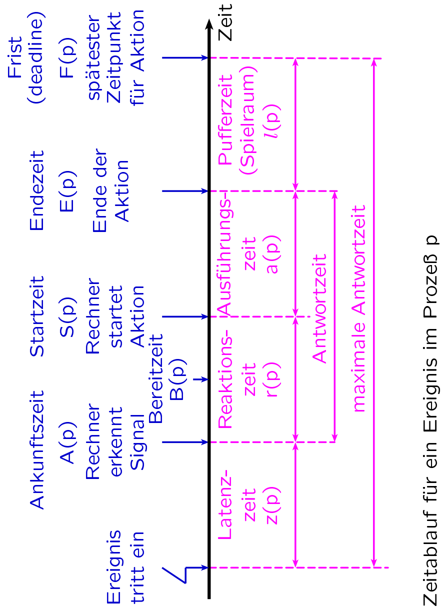
8.1 Grundbegriffe

Die Ablaufgeschwindigkeit der physikalischen Vorgänge im technischen Prozeß bestimmt notwendige Reaktionszeiten des Rechners und legt Antwortzeiten fest.

Es treten perodische und spontane Anforderungen (Zeitpunkte, Signale, Alarme), harte und weiche Zeitbedingungen auf.

Planung für Prozessoren und Betriebsmittel

- statische/dynamische Planung
- explizit/implizite Planung
- offline/online Planung



- Anmerkungen
  - Großbuchstaben: Zeitpunkte
  - Kleinbuchstaben: Zeitdauern
  - Reaktionszeit und Ausführungszeit abhängig von der Umgebung, z.B. von anderen wichtigerern Prozessen
  - Latenzzeit:
    u.a. Verzögerungen durch lokale Hardware, Übertragung über Netze
  - Bereitzeit:
    Der Prozeß darf vor seiner Bereitzeit nicht gestartet werden, da z.B. dann erst benötigte Daten verfügbar; in der Regel $B(p) = A(p)$

- Forderungen an Echtzeitplanung
  - a priori zeigen, daß stets Zeitrandbedingungen erfüllt,
    d.h. $\forall p : E(p) \leq F(p)$
  - Einplanung nicht nur CPU, sondern auch andere Betriebsmittel (in Vorlesung nur CPU berücksichtigt)
  - Zeitbedingungen einhalten bzgl. parallelen Vorgängen (Rechenprozessen)
  - Berücksichtigung von Kausalzusammenhängen (Synchronisation, Vorranggraphen, Präzedenzsysteme)
  - Einhalten von Zeitbedingungen bei der Verteilung von Information (Kommunikation, gemeinsames Führungsziel)
  - präemptive Prozessorzuteilung
- Nicht trivial, da technische Systeme vielfach komplex und schwer überschaubar

8.2 Dimensionierung der Rechnerleistung

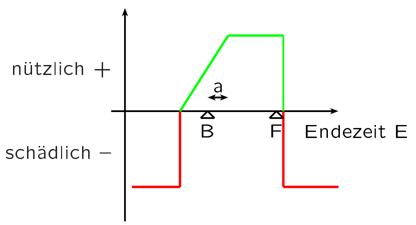
- Aufstellen "worst case"–Analyse
  - Bedarf aus bekannten periodischen Anforderungen
  - Bedarf aus erwarteten spontanen Anforderungen
  - Zuschlag 100% oder mehr zum Auffangen von Lastspitzen.
- Unterschied zu konventionellen Betriebszielen
  - keine maximale Auslastung des Prozessors,
  - keine Durchsatzoptimierung,
  - Abläufe sollten determiniert abschätzbar sein.

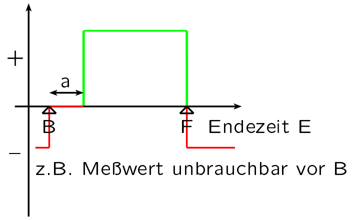
8.3 Schadenstaxonomie

- Programmablaufsteuerung muß Zeitbedingungen (Antwortzeiten) des technischen Prozesses berücksichtigen, damit kein Schaden entsteht.
- Harte Zeitbedingungen
  - msec, sec, aber nicht min oder Std.
  - Übertretung verursacht materiellen, wirtschaftlichen und/oder Personenschaden
  - Beispiele:
    - falsches Mischungsverhältnis ergibt Fehlproduktion
    - falsche Steuerung führt zu Maschinenschaden
    - energiesparende, jedoch instabile Fluglagen gefährden Flug
    - Landung Space–Shuttle
    - Roboter im Gefahrenbereich

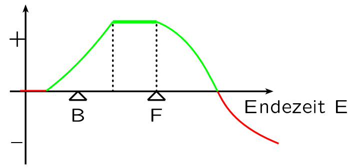
- Weiche Zeitbedingungen
  - Übertretung führt zu Störungen
  - das Führungsziel wird nicht erreicht
  - Verlust der Optimalität, Schaden tolerierbar
  - Beispiele:
    - Verspätete Produktfertigstellung
    - Höherer Energieverbrauch
    - tolerierbare Fehlfarben
    - tolerierbare Maßabweichungen
- Abstufung des Schadens nach Nutzenfunktionen

(1) Beispiele für harte Randbedingungen



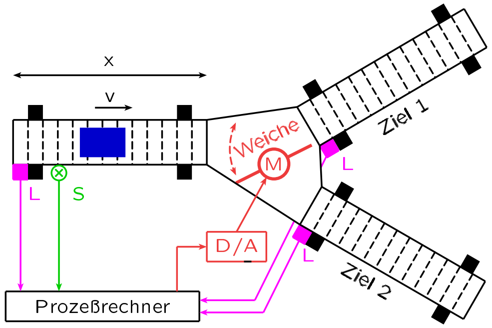


(2) Beispiel für weiche Randbedingungen



8.4 Echtzeitanforderung am Beispiel Paketverteilungsanlage

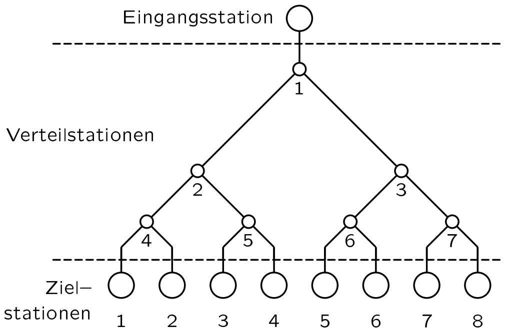
- Verteilstation



L	Lichtschranke
M	Motor für Weiche
S	Strichcode-Leser
v	Geschwindigkeit des Förderbands
x	Abstand Ankunftssignalgeber–Weiche
t_W	Zeitdauer zum Umstellen der Weiche

- Es treten asynchrone Ereignisse auf
- Ablauf
  - Rechner erhält Signal beim Eintreffen
  - Rechner liest und verarbeitet Strichcode
  - Rechner berechnet Weg (Ziel 1 oder Ziel 2)
  - Rechner gibt Signal an Weichenmotor (nur falls neuer Weg!)
  - Rechner reagiert nur beim Eintreffen von Signalen (ereignisgesteuert)
- von der Lichtschranke bis zur Weiche benötigt Paket die Zeit x/v
- Rechner muß spätestens t_W Zeiteinheiten bevor Paket an Weiche ist dem Weichenmotor das Stellsignal senden
- damit maximale Antwortzeit $\triangle t_{max} = x/v - t_W$

- Schema Gesamtanlage (Version 1)

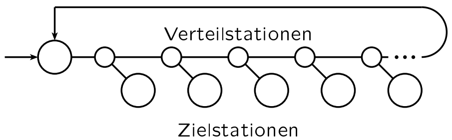


  - ⋆ 8 Zielstationen
  - ⋆ 7 Verteilstationen
  - ⋆ mehrere Pakete gleichzeitig
  - ⋆ Strichcode-Leser nur an Eingangsstation
  - ⋆ Rechner führt Buch, wo welches Paket aufgrund Signale der Lichtschranken und veranlaßt Weichenumstellungen (fehleranfällig, erschwerter Wiederanlauf)

⋆ Signalein–/ausgänge:

1	Leser Strichcode bei Eingangsstation
15	Lichtschranken
7	Signale für Motor nach links
7	Signale für Motor nach rechts
14	Kontakte für Ist-Weichenstellungen (links/rechts)
44	Signale

- Schema Gesamtanlage (Version 2)



- Exkurs zur Anlagenplanung
  - ⋆ Welche Konfiguration ist besser?
  - ⋆ Kriterien?
  - ⋆ Alternativen mit Einfluß örtlicher Verteilung?
- Exkurs zur Ausfallsicherheit
  - ⋆ Bei Ausfallsicherheit mindestens doppelt so viele Signale $\rightarrow$ 88 Signale
  - ⋆ Check vor Zielstation, ggf. zurück zu Eingangsstation
  - ⋆ Wiederanlauf nach Fehlern (überall Lesestationen?, alles zurück?)
  - ⋆ Erkennen Stau
  - ⋆ alternative Wege bei Störungen

- Kugelfallversuch
  einfaches Beispiel mit harten Zeitanforderungen

8.5 Weitere Bezeichnungen

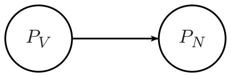
- Algorithmen zur korrekten Zuteilung von Rechenleistung bei mehreren zu bearbeitenden Aktionen (Rechenprozessen) $P_1, P_2, \ldots, P_n$ müssen für jeden Prozeß P_i beinhalten:
  - ⋆ $F(p_i)$
    spätester Zeitpunkt (Frist) für Antwort an technischen Prozeß
  - ⋆ $B(p_i)$
    Bereitzeitpunkt (frühester Startzeitpunkt)
  - ⋆ $a(pi)$
    Gesamtausführungszeit
- Weitere wichtige Größen, die von der aktuellen Zeit t abhängig sind:
  - ⋆ $a(p_i, t)$
    Restausführungszeit des Prozesses P_i zur Zeit t; wobei
    $a(p_i, B(p_i)) = a(p_i)$
    $a(p_i, F(p_i)) = 0$, falls Prozeß zeitgerecht beendet

- ⋆ $\triangle t(p_i, t)$
  Zeitspanne bis zur Frist des Prozesses P_i; wobei
  $\triangle t(p_i, t) = F(p_i) - t$
- ⋆ $l(p_i, t)$
  Zur Zeit t noch verfügbarer Spielraum für Prozeß P_i:
  $l(p_i, t) = \triangle t(p_i, t) - a(p_i, t)$
  falls:
  $l(p_i, t) < 0 \rightarrow P_i$ kann nicht rechtzeitig fertiggestellt werden
  Echtzeitforderung verletzt!

- Anmerkungen
  - ⋆ Laufzeit $a(pi)$ von Programmen bei Echtzeitumgebung bekannt, da Aufgaben und Aufgabenprofile festgelegt (Analyse des Codes, Messungen)
  - ⋆ ggf. Präzedenzen der Aktionen P_i beachten (z.B. P_i muß vor P_j ausgeführt werden)

8.6 Präzedenzsysteme

- Folgen von untereinander abhängigen Aktionen (z.B. "pipes")
- Bereitzeiten, Antwortzeiten oder Fristen ergeben sich evtl. erst aus vorherigem Ablauf
- Beispiel für Graph-Darstellung



Vorgänger Nachfolger

  - ⋆ Es gelte:

$$
\begin{aligned}
a(p_V) &= 1 & a(p_N) &= 3\\
F(p_V) &= 3 & F(p_N) &= 5\\
B(p_V) &= 0 & B(p_N) &= 0\\
\rightarrow \triangle t(p_V, 0) &= 3 & \rightarrow \triangle t(p_N, 0) &= 5
\end{aligned}
$$

  - ⋆ Falls von P_V Frist $F(p_V)$ gerade noch eingehalten wird, kann der Nachfolgeprozeß nicht rechtzeitig fertig werden

- Bei der Planung mit Präzedenzsystemen muß also berücksichtigt werden, daß auch Folgeprozesse noch rechtzeitig fertig werden

- Verfahren hierzu:
  Es wird ein normalisiertes Präzedenzsystem PS' anstelle des ursprünglichen Präzedenzsystems PS eingeführt
- Normalisierung (Durchführung zur Zeit t)
  - ⋆ $a'(p_i) = a(p_i)$ und $a'(p_i, t) = a(p_i, t)$
  - ⋆

$$
F'(p_i, t) =
\begin{cases}
F(p_i) & \text{falls } \mathcal{N}(p_i) = \emptyset\\
min\left(F(p_i),\ \underset{q \in \mathcal{N}(p_i)}{min}\ (F'(q) - a'(q, t))\right) & \text{sonst}
\end{cases}
$$

  - ⋆ wobei $\mathcal{N}(p_i)$ die Menge der unmittelbaren Nachfolger von p_i
  - ⋆ rekursive Berechnung, beginnend mit Prozessen ohne Nachfolger im Präzedenzgraphen
  - ⋆ Anmerkungen zu $B'(p_i)$:
    - ◦ hängen die Bereitzeiten von externen Vorgängen ab, die unabhängig vom Scheduling der Prozesse sind, dann gilt $B'(p_i) = B(p_i)$

    - ◦ hängen die Bereitzeiten von der Beendigung vorheriger Prozesse ab, dann ist $B'(p_i)$ abhängig vom konkreten Scheduling.
  - ⋆ ähnlich wie in der Netzplantechnik lassen sich früheste und späteste Start-Termine für die Prozesse angeben. Diese sind aber für das Scheduling ohne Bedeutung, da ja die konkreten Präzedenzen berücksichtigt werden müssen.
- Präzedenzsystem nur dann planbar, falls zugehöriges normalisiertes PS' planbar, d.h. es dafür einen zulässigen Plan gibt

8.7 Pläne

- Phasen der Planung:

(1) Test auf Einplanbarkeit (feasibility check)
(2) Planberechnung (schedule construction)
(3) Umsetzung auf Zuteilung im Betriebssystem (dispatching)

- Gesucht:
  Plan mit aktueller Start– und Endezeit für jede Aktion P_i
- Darstellung z.B. als nach der Zeit geordnete Liste von Tupeln (P_i, S(P_i), E(P_i), BM); wobei BM die Menge der benötigten Btriebsmittel (kann entfallen, wenn immer nur eines geplant wird)
- Falls Prozeß unterbrochen wird: mehrere Tupel je P_i mit Anfangs- und Endzeitpunkten von aktiven Intervallen
- Zulässiger Plan
  Ein Plan, bei dem alle Prozesse einer Prozeßmenge eingeplant werden und dabei keine Präzedenzrestriktionen und keine Zeitanforderungen verletzt werden

- Optimales Planungsverfahren
  Verfahren ist optimal, wenn es für jede Prozeßmenge unter den gegebenen Randbedingungen einen zulässigen Plan findet, falls ein solcher existiert
- statische Planung
  - ⋆ erfordert Kenntnis aller Prozesse P_i und von $B(p_i)$, $a(p_i)$ und $F(p_i)$
  - ⋆ abgeschlossenes System
  - ⋆ exakte Algorithmen gehören in diese Gruppe
- dynamische Planung
  - ⋆ unvorhergesehen eintreffende Aktionen (Prozesse)
  - ⋆ unbekannte Prozeßparameter
  - ⋆ keine Gesamtoptimierung
- Grundstrategien zur Prozeßbearbeitung
  - ⋆ Präemptives (unterbrechbares) Abarbeiten
    - ◦ Aktionen (Prozesse) haben Prioritäten
    - ◦ Prioritäten statisch oder dynamisch berechnet

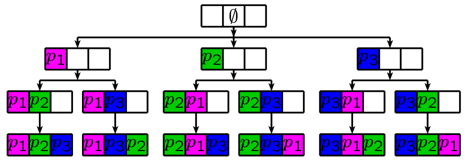
- ◦ Ausführung einer Aktion wird sofort unterbrochen, sobald Aktion mit höherer Priorität eintrifft
- ◦ unterbrochene Aktion wird an der Unterbrechungsstelle fortgesetzt, sobald keine Aktion höherer Priorität ansteht
- ◦ typisch für Echtzeitaufgaben (mit Ausnahme von Programmteilen, die zur Sicherung der Datenkonsistenz nicht unterbrochen werden dürfen)
- ◦ Nachteil: häufiges Umschalten reduziert Leistung

⋆ Nichtpräemptives (ununterbrechbares) Abarbeiten

- ◦ Eine begonnene Aktion wird beendet, selbst wenn während der Ausführung Aktionen höherer Dringlichkeit eintreffen
- ◦ z.B. Schreiben in Druckerpuffer, Schließen eines Ventils
- ◦ Nachteil: evtl. Versagen (zu hohe Reaktionszeit) des Systems beim Eintreffen "unvorhergesehener" Anforderungen

8.8 Planen durch Suchen

- Zunächst ununterbrechbare Aktionen vorausgesetzt
- exakte Planung über Durchsuchen des Lösungsraums
- Beispiel:
  - ⋆ n=3 Prozesse P_1, P_2, P_3 und 1 CPU
  - ⋆ Suchbaum:



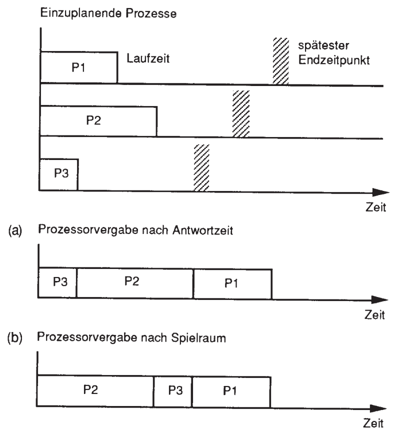
- n! (hier 6) Permutationen zu bewerten
- Bei n zu planenden Prozessen und einem Mehrfachbetriebsmittel aus m Einheiten hat ein Knoten $\binom{n}{m} = \frac{n!}{m!(n-m)!}$ Unterknoten

- wegen Komplexität normalerweise nicht einsetzbar
- leichte Verbesserung der Komplexität durch
  - ⋆ Abbrechen der Bildung einer Sequenz, wenn bei einer der Aktionen eine Zeitüberschreitung
  - ⋆ Heuristiken, z.B. vorsortieren nach $B(p_i)$
- viele Algorithmen z.B. im Buch: J.Blazewicz,et al.: Scheduling in Computer and Manufacturing Systems. 2.Auflage, Springer 1994
- Bei unterbrechbaren Aktionen führt Granulariät der Zeitintervalle mit Präzedenzen der Aktionen zu System mit ununterbrechbaren Aktionen
- Weitere Probleme treten durch Synchronisation der Aktionen und die Konkurrenz um weitere Betriebsmittel auf (siehe z.B. J.Xu, D.L.Parnas: Scheduling Processes with release times, deadlines, precedence and exclusion relations. IEEE Trans. on SW Eng., Vol.16, 1990, S.360).

8.9 Zeitplanung bei Einprozessor–Systemen

- Einziges verplantes Betriebsmittel ist 1 CPU
- Kriterien zur Prüfung auf Einhaltbarkeit aller Zeitbedingungen sind relativ einfach
  - ⋆ n Prozesse
  - ⋆ geordnet nach Fristen:
    $F(p_i) < F(p_{i+1}) \quad (i = 1 \ldots, n-1)$
  - ⋆ gleiche Bereitzeiten $B(p_i) = 0$
  - ⋆ es muß notwendig gelten:
    $\forall i : \triangle t(p_i, t) \geq \sum_{k=1}^{i} a(p_k, t) \quad (i = 1, \ldots, n)$
- gesucht: Strategien zur dynamisch korrekten Ordnung der Prozesse
- S1: Einplanung nach Fristen
  next deadline scheduling
  der Prozeß mit der nächsten Frist F erhält als nächster die CPU
- S2: Einplanung nach kleinstem Spielraum
  least slack time scheduling
  least laxity scheduling
  der noch nicht beendete Prozeß p mit dem momentan kleinsten Spielraum l(p, t) erhält als nächster die CPU

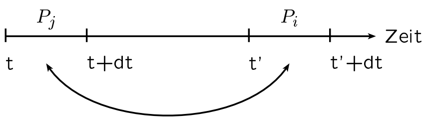
- ⋆ $l(p_j, t) \;=\; F(p_j) - t - a(p_j, t)$
- ⋆ solange p_i bearbeitet wird, ist $l(p_i, t)$ konstant; für alle wartenden p_j nimmt $l(p_i, t)$ linear in t ab.
- ⋆ erkennt früher als S1, wenn Zeitbedingungen nicht eingehalten werden können



- Voraussetzung: Einprozessorsystem, gleiche Bereitzeiten,
  Satz: Die Zuteilungsstrategien S1 und S2 sind optimal
- Beweis für Optimalität von S1
  - ⋆ Beweisidee: Tausch in existierendem Plan
  - ⋆ zulässiger Plan PlanZ sei bekannt
  - ⋆ PlanF sei Plan nach Fristen
  - ⋆ Prozesse seien nach Fristen geordnet:
    $F(p_i) \leq F(p_{i+1})$
  - ⋆ pz(Plan, t) liefert den Index des Prozesses, der im genannten Plan zur Zeit t aktiv ist
  - ⋆ PlanZ wird schrittweise in PlanF überführt
  - ⋆ PlanZ(t) ist der bis zur Zeit t modifizierte Plan
  - ⋆ PlanZ(0)=PlanZ, d.h. Ausgangssituation
  - ⋆ Annahme: PlanZ(t) und PlanF seien bis zum Zeitpunkt t identisch
  - ⋆ zum Zeitpunkt t gilt:
    i = pz (PlanF, t)
    j = pz (PlanZ(t), t)
    i ≠ j (nach Voraussetzung)
  - ⋆ betrachtet Zeitintervall dt

⋆ Fall $i < j$:

- $F(p_i) \le F(p_j)$
  wegen Ordnung d. Prozesse
- $t + dt \le F(p_j)$
  da PlanZ zulässiger Plan
- da p_i in PlanF im Intervall (t, t+dt) rechnet und die Pläne bis zur Zeit t identisch sind, kann p_i auch in PlanZ noch nicht beendet sein, also gilt:
  $\exists t' \ge t + dt \ : \ i = pz(PlanZ(t), t') = pz(PlanZ(t), t' + dt)$
  $t' + dt \le F(p_i) \le F(p_j)$
- Übergang von Planz(t) zu PlanZ(t+dt) erfolgt durch zeitliches Tauschen der Aktivitätsphase von p_i mit der von p_j



- Zeitbedingungen sind dadurch nicht verletzt

⋆ Fall $i > j$:

- rechnet in PlanF zur Zeit t der Prozeß p_i, dann sind alle Prozesse p_j mit $j < i$ bereits beendet
- PlanF und PlanZ(t) stimmen bis zur Zeit t überein
- der Fall $i > j$ kann also gar nicht auftreten

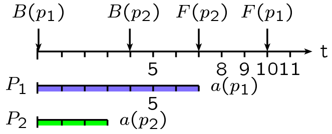
⋆ Damit ist gezeigt: Jeder zulässige Plan kann ohne Verletzung der Fristen in den durch die Fristenplanung erzeugten Plan überführt werden

⋆ Fristenplanung ist also bei Einprozessorsystemen optimal, egal ob präemptiv oder nichtpräemptiv

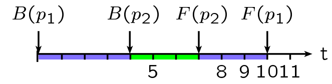
- Beweis für Optimalität von S2
  Beweisidee wie für S1

- Beispiel, daß $B(p_i) = 0$ notwendig
  S1 und S2 liefern unzulässige Pläne, falls dies nicht der Fall (vgl. folgende Seite)
- allerdings Modifikation möglich, so daß korrekte Pläne auch in diesem Fall:
  ⋆ präemptive Strategie
  ⋆ Neuplanung bei jeder Bereitzeit
  ⋆ Einplanung nur derjenigen Prozesse, deren Bereitzeit erreicht ist
  ⋆ entspricht Neuplanung, wenn ein Prozeß aktiv wird
  ⋆ ev. Zeitscheiben für Prozesse mit gleichem Spielraum

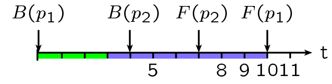
Lastprofil:



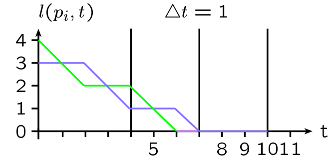
korrekter Plan:

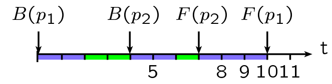


Fristenplanung:



kleinster Spielraum:

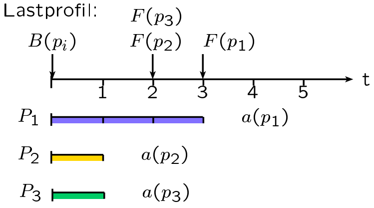




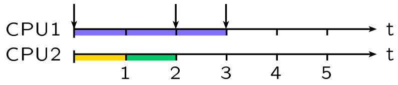
8.10 Zeitplanung bei Mehrprozessor–Systemen

- gehört zur Klasse der Zuteilungsverfahren für homogene, austauschbare Betriebsmittel
- S1 nicht optimal, egal ob präemptiv oder nichtpräemptive Strategie
- S2 geht nur, falls alle Bereitzeitpunkte $B(p_i)$ gleich
- korrekte Zuteilungsalgorithmen erfordern das Abarbeiten von Suchbäumen mit NP–Aufwand oder geeignete Heuristiken
- Buch: J. Blazewicz, et al.
- Gegenbeispiel zu S1 ist optimal
  ⋆ präemptiv
  ⋆ 2 Prozessoren CPU_1 und CPU_2
  ⋆ 3 Prozesse p_1, p_2, p_3 mit $B(p_i) = 0$
  ⋆ S1 genügt nicht! P1 zu spät beendet
  ⋆ S2 liefert die korrekte Zuteilung, da alle Bereitzeiten gleich

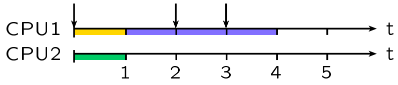
Lastprofil:



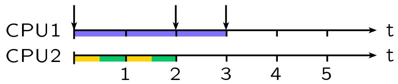
korrekter Plan:



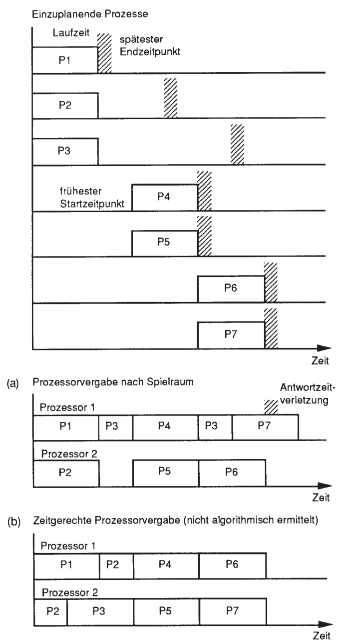
Fristenplanung S1:



Spielraumplanung S2: $\triangle t = .5$



- Nachweis, daß S2 optimal bei gleichen Bereitzeiten aller Prozesse durch Betrachtung der Auslastung. Es wird immer am Prozeß mit geringstem Spielraum gearbeitet, d.h. wenn dort Zeitüberschreitung auftritt, dann auch, falls noch Prozesse mit größerem Spielraum eingeschoben würden.
- Beispiel, daß S2 bei unterschiedlichen Bereitzeiten versagt



- Beweis, daß jede Zuteilungsstrategie versagt, wenn
  - ⋆ unterschiedliche $B_{(pi)}$ und
  - ⋆ die $B_{(pi)}$ nicht vorab bekannt und
  - ⋆ nicht alle Prozesse unter Berücksichtigung ihrer $B_{(pi)}$ zum Zeitpunkt $t = 0$ geplant
  - ⋆ d.h. es versagt auch S2 (kürzester Spielraum) bei unterschiedlichen $B_{(pi)}$
  - ⋆ Vorgehensidee:
    - ∘ n CPU's und n-2 Prozesse ohne Spielraum, diese muessen sofort rechnen, da sonst Fristüberschreitung
    - ∘ damit Problem auf 2 CPU's reduziert
    - ∘ drei weitere Prozesse vorhandenen und einzuplanen
    - ∘ Reihenfolge von Strategie abhängig, alle Reihenfolgen zu betrachten
    - ∘ später treffen dann noch Prozesse ein, die solche Fristen haben, daß auf jeden Fall Fristenüberschreitung auftritt
    - ∘ aber zulässiger Plan existent, wenn alle Prozesse von Anfang an bekannt

⋆ Zahlenwerte:
- ∘ 3 Prozeße p_1, p_2 und p_2 vorhanden

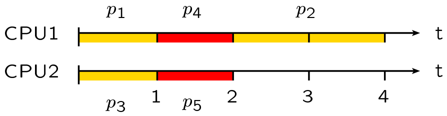
i	$B(P_i)$	$a(P_i)$	$F(P_i)$
1	0	1	1
2	0	2	4
3	0	1	2

- ∘ p_1 ohne Spielraum, rechnet auf CPU1
- ∘ Es gibt jetzt je nach Strategie zwei Fälle zu betrachten: p_2 oder p_3 an CPU2

⋆ Fall 1: p_2 an CPU2
- ∘ zur Zeit t=1 muß p_3 begonnen werden
- ∘ zur Zeit t=1 treffen zwei Aufträge ein

i	$B(P_i)$	$a(P_i)$	$F(P_i)$
4	1	1	2
5	1	1	2

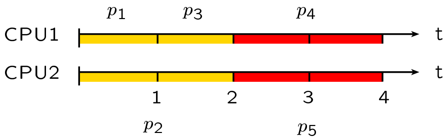
- ∘ damit müssen drei Aufträge ohne Spielraum bearbeitet werden; geht nicht
- ∘ es gibt aber korrekten Plan:



⋆ Fall 2: p_3 an CPU2
- ∘ zur Zeit t=1 sind p_1 und p_3 beendet
- ∘ zur Zeit t=1 wird p_2 begonnen
- ∘ zur Zeit t=2 treffen 2 Aufträge ein

i	$B(P_i)$	$a(P_i)$	$F(P_i)$
4	2	2	4
5	2	2	4

- ∘ es müssen noch 5 Zeiteinheiten Rechnerleistung aufgebracht werden
- ∘ es sind aber nur 4 vorhanden bis zu den Fristen bei Zeit 4
- ∘ Plan geht nicht
- ∘ es gibt aber korrekten Plan:

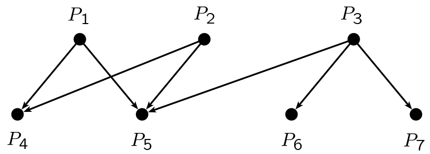


8.11 Anomalien bei nichtpräemptiven Systemen

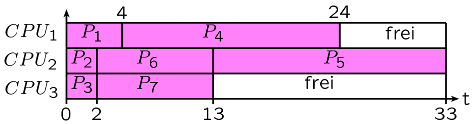
- Bei den plausiblen Strategien S1 oder S2 können sog. Anomalien auftreten, z.B.:
  - ⋆ zeitweise freiwilliges Ruhenlassen eines Prozessors kann die Gesamtzeit verkürzen
  - ⋆ der Einsatz eines zusätzlichen Prozessors verlängert die Gesamtzeit.
- Beispiel aus Coffman/Denning: Operating Systems Theory
  - ⋆ 3 gleichartige Prozessoren CPU_1, CPU_2, CPU_3
  - ⋆ Präzedenzgraph für 7 Aktionen $p_1, p_2, \ldots, p_7$
  - ⋆ vorgegebene Laufzeiten $a(p_i)$

i	$a(p_i)$
1	4
2	2
3	2
4	20
5	20
6	11
7	11

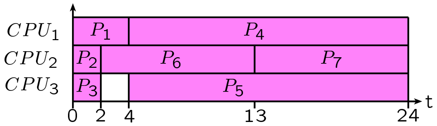
⋆ Präzedenzgraph:



- ⋆ p_1, p_2, p_3 sofort startbar
- ⋆ plausible Zuteilungsstrategie: Starten einer Aktion, sobald Vorgänger fertig und eine CPU frei; Gesamtzeit = 33



- ⋆ Optimale Zuteilung; Gesamtzeit = 24

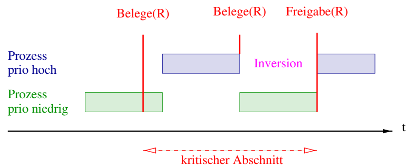


8.12 Zuteilung nach Prioritätszahlen

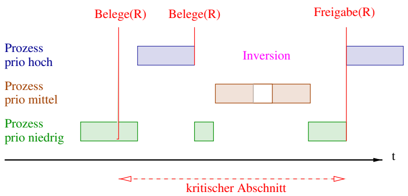
- die einfachen Strategien S1 und S2 werden in der Praxis auch bei Einprozessorsystemen nicht explizit angewandt, da
  - ⋆ kein abgeschlossenes System der Aktionen (Alarme, Interrupts erfordern dynamische Planung)
  - ⋆ Bereitzeitpunkt nur bei zyklischen und Termin–Prozessen bekannt,
  - ⋆ Laufzeiten nicht exakt bekannt, von Daten und Umgebung abhängig
  - ⋆ Synchronisation, Kommunikation und gemeinsame Betriebsmittel verletzen die Forderung nach Unabhängigkeit der Aktionen
- bei großen Systemen Zuteilung nach festen Prioritätszahlen
- Priorität aus Wichtigkeit für den technischen Prozeß und aus Abschätzungen der aktuellen Fristen oder Spielräume
- bei gleicher Priorität meist FIFO–Strategie

- ausreichende Leistungsreserve der CPU mittels worst-case-Betrachtungen
- Prioritätszahlen sind meist vom Typ nat im Bereich [0..255]
- je größer die Zahl, desto höher die Dringlichkeit (z.B. LYNXOS); oft auch umgekehrt (z.B. PEARL, UNIX)
- Einteilung in Prioritätsgruppen, je nach Härte der Zeitbedingungen
- Wann wird umgeplant ?
  - ⋆ Prioritätsinkonsistenz möglichst kurz
  - ⋆ beim Beenden einer Aktion
  - ⋆ beim Übergang einer Aktion in Wartezustand.
  - ⋆ beim Eintreffen einer neuen Anforderung (neuer Prozeß wird aktiv)
  - ⋆ nach bestimmten Zeitintervallen Überprüfung der Situation (z.B. bei Spielraumplanung)

- Behinderung wichtiger Prozesse durch unwichtige darf in Echtzeitsystemen nicht auftreten, also
  - ⋆ Prioritätssreihenfolge bei allen Anforderungen von Betriebsmitteln (CPU, Semaphore, Netzkommunikation, Puffer, Peripherie),
    d.h. Vordrängen in allen Warteschlangen
- Prioritätsinversion (priority inversion) Ein Prozess mit niedriger Priorität blockiert einen Prozess mit höherer Priorität
  - ⋆ begrenzte Inversion (bounded inversion) Die Inversion ist durch die Dauer des kritischen Abschnitts beschränkt



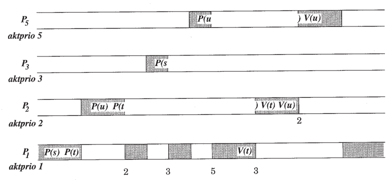
- ⋆ unbegrenzte Inversion (unbounded inversion) Der kritische Abschnitt wird durch weitere Prozesse auf unbestimmte Zeit blockiert



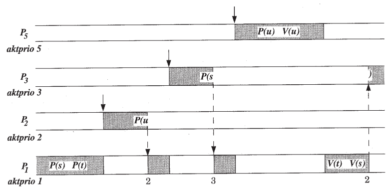
- Prioritätsvererbung (priority inheritance)
  - ⋆ Der Prozess erbt die höhere Priorität des Prozesses, solange dieser das gemeinsame Betriebsmittel blockiert.
  - ⋆ Verhindert unbegrenzte Blockierung
  - ⋆ Die Dauer der Blockierung wird auf die Dauer des kritischen Abschnitts begrenzt
  - ⋆ Die Blockierungen werden hintereinander gereiht (Blockierungsketten)
  - ⋆ Es verhindert keine deadlocks

- Prioritätsgrenzen (priority ceiling)
  - ⋆ Jedes Betriebsmittel (Semaphor) s erhält eine Prioritätsgrenze $ceil(s)$ =Maximum der Prioritäten der Prozesse, die auf s zugreifen
  - ⋆ Der Prozess p darf ein BM nur blockieren, wenn er von keinem anderen Prozess, der andere BM besitzt, verzögert werden kann
  - ⋆ aktuelle Prioritätsgrenze für Prozess p $aktceil(p) = max\{ceil(s) | s \in lockedsem\}$ $lockedsem$ = Menge aller blockierter BM
  - ⋆ Prozess p darf Betriebsmittel s benutzen, wenn $aktprio(p) > aktceil(p)$
  - ⋆ andernfalls gibt es genau einen Prozeß, der s besitzt. Die Priorität dieses Prozesses wird auf $aktprio(p)$ gesetzt
  - ⋆ Blockierung nur für die Dauer eines kritischen Abschnitts
  - ⋆ Verhindert deadlocks
  - ⋆ schwieriger zu realisieren, zusätzlicher Prozesszustand

- ⋆ vereinfachtes Protokoll: Immediate priority ceiling Prozesse, die BM s belegen, erhalten die Priorität $ceil(s)$
- Beispiel zur Prioritätsvererbung



- gleiches Beispiel mit Prioritätsgrenzen

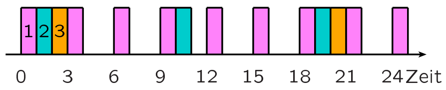


- Ein reales Beispiel für das Inversionsproblem: Mars Pathfinder

8.13 Zeitplanung periodischer Prozesse

- Bezeichnungen
  - ⋆ Prozesse werden periodisch aktiviert
  - ⋆ n Prozesse p_i, mit $1 \leq i \leq n$
  - ⋆ Periode $T(p_i)$
  - ⋆ Ausführungszeit $a(p_i)$
  - ⋆ $B(p_i)$ Bereitzeit relativ zum Beginn einer Periode
  - ⋆ $F(p_i)$ Frist relativ zum Beginn einer Periode
- Beispiel: Meßwerterfassung
  - ⋆ 3 Prozesse p_i
  - ⋆ jeweils Werte einlesen, umrechnen, skalieren, auf Platte speichern
  - ⋆ Kenngrößen der Prozesse:

i	$T(p_i)$	$B(p_i)$	$a(p_i)$	$F(p_i)$
1	3	0	1	3
2	9	0	1	9
3	18	0	1	18



- Einplanbarkeit aufgrund Last
  - ⋆ Auslastung durch Prozeß P_i:

$$\varrho_i = \frac{a(p_i)}{T(p_i)}$$

  - ⋆ Gesamte Auslastung :

$$\varrho = \sum_{i=1}^{n} \varrho_i = \sum_{i=1}^{n} \frac{a(p_i)}{T(p_i)}$$

  - ⋆ Bei m Einheiten eines BM gilt: $\varrho \leq m$ notwendig für Zeiteinhaltung, aber nicht hinreichend!
- Einplanung nach Fristen
  - ⋆ bei 1 BM, $\varrho \leq 1$ und $T(p_i) = F(p_i)$ für alle i ist Einplanung nach Fristen optimal
  - ⋆ Beweisidee: vor dem Verletzen einer Frist ist das BM nie unbeschäftigt

- Einplanung nach Raten rate–monotonic scheduling, smallest period–first
  - ⋆ Rate $R_i = 1/T(p_i)$
  - ⋆ hohe Rate = hohe Priorität
  - ⋆ Prozesse mit hohen Raten zuerst bedient
  - ⋆ Ratenplanung benutzt also statische Prioritäten
  - ⋆ optimal, falls eine Lösung mit statischen Prioritäten existiert
  - ⋆ Verfahren mit dynamischen Prioritäten können aber evtl. bessere Ergebnisse liefern
  - ⋆ Einplanbarkeitstest 1 für Ratenplanung
    - ∘ Bei n Aktionen ist Ratenplanung sicher erfolgreich,falls

$$\varrho \leq \varrho_{max} = n * \left(2^{1/n} - 1\right)$$

$$\lim_{n\to\infty} \varrho_{max} = ln2 \approx 0.69$$

    - ∘ Beweis (Liu, Layland 1973) durch "worstcase"–Analyse

- ∘ Achtung! Das heißt aber nicht, daß Ratenplanung für Systeme mit $\varrho > \varrho_{max}$ nicht doch Lösungen findet, nur ist dies nicht garantiert.
- ∘ Für Spezialfälle (Perioden gleich oder Perioden Teiler einer größeren) gibt es günstigere Abschätzungen
- ⋆ Einplanbarkeitstest 2 für Ratenplanung
  - ∘ Für Satz periodischer Aktionen P_i gibt Ratenplanung einen erfolgreichen Plan, falls unter der Annahme gleichzeitigen Starts (= worst case) alle Aktionen innerhalb ihrer ersten Periodenzeit $T(p_i)$ beendet werden
  - ∘ Test ist auch bei grosser Anzahl Prozesse durch Simulation effizient ausführbar, falls die Perioden nicht teilerfremd
  - ∘ Beweisidee: Gleichzeitiger Start ist worst case-Situation, da alle höherprioren Aktionen eine Ausführung von P_i verzögern

8.14 Vergleich der Planungsverfahren

- Einordnung der eingeführten Verfahren
- für Einprozesorsystem (Ausnahme angegeben)
- und statische Planung

Strategie	präemptiv	nichtpräemptiv
Suchen		optimale Pläne O(n!), NP-vollständig
Fristen Spielraum	optimal	optimal bei gleicher Bereitzeit
Spielraum (Mehrpr.)	optimal bei gleicher Bereitzeit	NP-vollständig Anomalien
Monotone Raten	optimal bei beschränkter Auslastung	

9 Echtzeit-Betriebssysteme

Wichtige Eigenschaften gegenüber
Universalrechnern herausgegriffen

9.1 Anforderungen

- stabiler Betrieb rund um die Uhr
- definierte Reaktionszeiten
- parallele Prozesse
- Unterstützung von Mehrprozessorsystemen
- schneller Prozeßwechsel (geringer Prozeßkontext)
- Unterbrechungshierarchie
- Unterbrechungsbehandlungen in Benutzerprozessen (Unterbrechungs–Antwortprogramme)
- Prioritäten für Prozesse
- Scheduling streng präemptiv nach Prioritäten mit FIFO bei gleicher Priorität
- Warteschlangen im System nach Prioritäten
- Prioritätsvererbung

- umfangreiche Zeitdienste
  - ⋆ absolute und relative Uhren
  - ⋆ Weckdienste (delay)
  - ⋆ Prozesse einplanen mit Zykluszeit
- Zeitüberwachung aller Wartezustände (timeout)
- Prozeß–Management (Task–M.)
  - ⋆ erzeugen, starten, löschen
  - ⋆ anhalten, wiederaufnehmen; abhängig von Zeit-punkten, -intervallen, anderen Ereignissen, periodisch
- Synchronisation und Kommunikation
  Semaphor, Botschaften (n:m), Ereignisse, Monitore (protected objects), Rendezvous (jeweils einfache Konzepte, mit Prioritäten)
- einfaches Speicher–Management
- keine virtuelle Adressierung
- wichtige Prozesse speicherresident

- Direkter Ein–/Ausgabeverkehr mit Peripherie
  - ⋆ vielfältigste Peripherie
  - ⋆ direkter Zugriff auf Hardware-Adressen und -Register durch den Benutzer
  - ⋆ Treiber in Benutzerprozessen möglichst schnell und einfach zu implementieren
    dynamisches Binden an Systemkern
  - ⋆ direkte Nutzung DMA
  - ⋆ keine mehrfachen Puffer: direkt vom Benutzerpuffer auf das Gerät
- Einfachste Dateistrukturen
- Protokoll für Feldbus oder LAN–Bus, möglichst hardware-unterstützt
- statt Programmierschnittstelle ein Bediensystem (Kommandos, Zustandsübersicht, Messungen)
- Generierung maßgeschneiderter BS aus Bibliotheken (besonders vorteilhaft für Masseneinsatz in eingebetteten Systemen)

9.2 Beispiele für Produkte

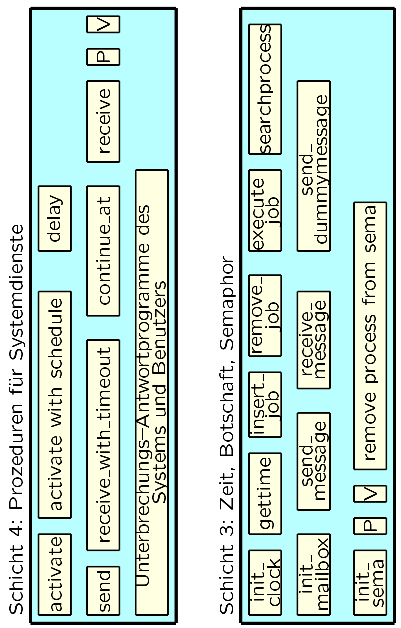
- unvollständige Liste!
- viele herstellerspezifische (ad-hoc) Lösungen für bestimmte Produktbereiche
- Einsatzgebiete
  - ⋆ Industrie-Steuerungen
  - ⋆ Prozeßvisualisierung
  - ⋆ Informationssysteme
  - ⋆ Mobile Computer
  - ⋆ Consumer electronics
  - ⋆ Multimedia-Anwendungen
  - ⋆ Hausgeräte
  - ⋆ Unterhaltungsindustrie
  - ⋆ Kommunikationssysteme
  - ⋆ Handys
  - ⋆ eingebettete Systeme, u.a. Automobil
- Tendenz zu universelleren (firmen- und produktübergreifenden) Echtzeitsystemen
  - ⋆ geringere Entwicklungs- und Wartungskosten
  - ⋆ Kompatibilität zu anderen

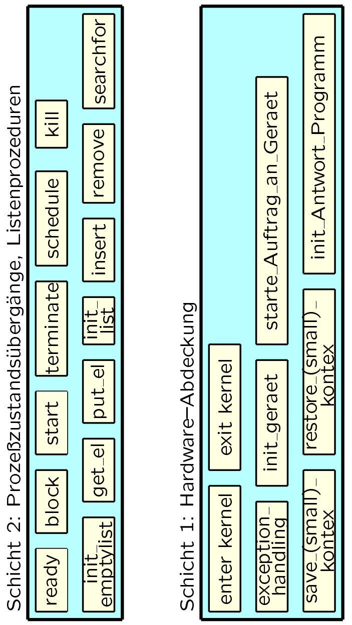
  - ⋆ geringerer Schulungsaufwand
  - ⋆ weit höhere Funktionalität und Qualität
- Schnittstelle typischerweise als 1:1-Systemaufrufe in der Programmiersprache (z.B. Fortran, Pascal, C, C++, Java)
- Schnittstelle variiert bei den Realzeitsystemen beträchtlich (wenig portabel)
- VRTX (32)
  VERSATILE REALTIME EXECUTIVE ("Vertex") von READY SYSTEMS für Motorola, Intel u. a.
- VRTXmc von Microtec Research
- VME$_{exec}$
  von Motorola für VME-Bus-Umgebung
- OS-9 von Dr. Keil für Motorola, Intel
- FLEXOS
  von DIGITAL RESEARCH für PC
- OSEK/VDX (Automobilhersteller)
- RMOS3 bei Siemens Automation & Drives
- QNX-RTOS von QNX Software Systems für PC

- VxWorks von Wind River für alle
- LynxOS
  von Lynx Real-Time Systems für Motorola, Intel, Sparc
  (Einsatz im Prozeßrechner–Praktikum)
- Windows-CE und Windows NT Embedded von Microsoft für eingebettete Systeme
  - ⋆ in D u.a. Partnerschaft mit Siemens dafür
- RT-Linux in vielen Varianten

9.3 Schichtenmodell

- erste Spezifikation in European Workshop on Industrial Computer Systems (EWICS) TC8 on Real-Time Operating Systems (1976-1979)
- für ein verteiltes Mehrprozessor-Echtzeit-Betriebssystem
  level 1: Context Switching,
  Basic Synchronisation,
  List Handling
  level 2: Dispatcher Primitives
  level 3: Synchronization,
  Task Management
- Implementierung als Diplomarbeit 1981 auf Mehrrechnersystem (drei LSI11) am Lehrstuhl
- in den folgenden Skizzen:
  - ⋆ Komponenten unvollständig
  - ⋆ Komponenten durch einen typischen Dienst charakterisiert
  - ⋆ Beispiel, kein reales System als Modell

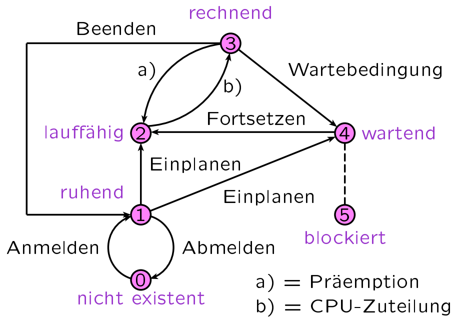




9.4 Rechenprozesse (tasks, threads)

- Gleichzeitige physikalische Vorgänge erfordern parallele Aktionen im Mehrprogramm-, Mehrprozessor- oder Mehrrechnerbetrieb
- Prozeßrechner–Betriebssysteme müssen mehrere Prozesse präemptiv (unterbrechbar) nach Prioritäten steuern
- Unterbrechungs–Antwortprogramme und Betriebssystem–Kerndienste sind spezielle Abläufe, die ebenfalls wie Prozesse unterbrechbar
- Leichtgewichtsprozesse (threads)
  - ⋆ ohne eigenen Prozeßadreßraum
  - ⋆ laufen im Prozeßadreßraum eines (Schwergewichts-)Prozesses
  - ⋆ Prozeßwechsel billig
  - ⋆ Erzeugen threads billig
  - ⋆ schneller Zugriff auf gemeinsame Daten im gemeinsamen Adreßraum

- Zustandsdiagramm



- ruhender (inaktiver) Rechenprozeß (Zustand 1): keine Einplanung im BS, aber angemeldet und in den Listen des BS geführt; kein Auftrag zur Abwicklung
- Rechenprozeß aktiv in Zuständen > 1; Prozeß gestartet bzw. Start im BS eingeplant
- Zyklische Prozesse sind dauernd aktiv

- Zustand lauffähig (ready):
  Alle Voraussetzungen für Start bzw. Fortsetzung erfüllt;
  wartend auf Prozessorzuteilung
- Zustand wartend (waiting):
  Alle Betriebsmittel außer CPU zugeteilt, aber Prozeß wartet auf Eintreten einer Bedingung (z.B. Ablauf einer Wartezeit, Eintreffen eines Interrupts, Eingang einer Botschaft, Freigabeoperation (V-Operation) eines anderen Prozesses
- Zustand blockiert (blocked):
  Dem Rechenprozeß fehlt ein vom BS verwaltetes Betriebsmittel, z.B. der Arbeitsspeicher (nur für Hintergrundaufgaben, zeitkritische Prozesse dauernd aktiv und speicherresident)

- Prozeßaktivierung
  - ⋆ Variante 1:
    - ◦ Hauptprozess (maintask) wird gestartet
    - ◦ Prozesse ruhend und bei Bedarf durch Betriebssystemaufrufe aus anderen Prozessen aktiviert
    - ◦ für Meß- und Steuerungsaufgaben mit wechselnden Aktionen
  - ⋆ Variante 2:
    - ◦ alle Prozesse einmalig bei der Initialisierung aktiviert
    - ◦ keine dynamisch erzeugten Prozesse
    - ◦ Prozesse dauernd aktiv mit internem Zyklus von Arbeitsphasen und von Wartephasen auf Nachrichten
    - ◦ für Dienstprozesse (server), die auf Aufträge anderer Prozesse warten
  - ⋆ Variante 3 (in ADA):
    - ◦ Task wird innerhalb Blockstruktur deklariert und beim Durchlaufen der Deklaration automatisch gestartet
    - ◦ aber auch Taskvariable zur dynamischen Erzeugung

⋆ Beispiel zu Variante 1: (PEARL; Process and Experiment Automation Realtime Language)

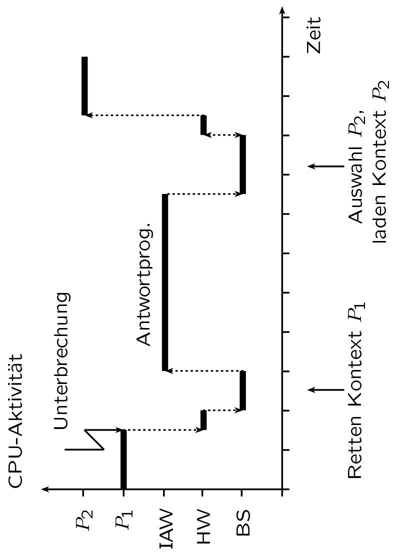
```
TASK master_1;
  <Deklarationen>
  <Initialisierung von Daten und Geraeten>
  ...
  ACTIVATE P1;
  AT 7:00 ACTIVATE P2;
  AT 12:00 ALL 1 min ACTIVATE P3
    UNTIL 16:30;
  ...
  LOOP
    RECEIVE botschaft FROM ANY;
    CASE botschaft.typ OF
      b1: <Fall Prozeß Pi beendet> ;
      b2: <Fall Neuplanung erforderlich> ;
      b3: <Fall Fehler in Pi aufgetreten> ;
    END CASE;
  END LOOP;
END master_1;
TASK Pi;
  <Deklarationen>
  ...
  SEND <botschaft>  TO master_1;
    /* z.B. Botschaft "b1" */
END Pi;
```


⋆ Beispiel zu Variante 2:

```
TASK master_2;
  <Deklarationen>
  <Initialisierung von Daten und Geraeten>
  ...
  SEND <starte>  TO P1;
  SEND <starte>  TO P2;
  ...
  LOOP
    RECEIVE botschaft FROM P1 OR P2;
    CASE botschaft.typ OF
      b1: <Fall Ergebnis von Pi> ;
      b2: <Fall Fehler in Pi> ;
    END CASE;
  END LOOP;
END master_2;
TASK Pi;
  <Deklarationen>
  <Initialisierungen>
  LOOP
    RECEIVE <starte>  FROM ANY;
    Auftrag aus starte ausführen;
      /* jetzt Antwort an Auftraggeber senden */
    SEND <ergebnis>  TO <client> ;
  END LOOP;
END Pi;
```


9.5 Unterbrechungs–Antwortprogramme

- Unterbrechungssignale (Interrupts)
  - ⋆ von Geräten, Uhr, Signalgebern usw.
  - ⋆ haben Unterbrechungspriorität
  - ⋆ können Rechenprozesse niederer Priorität unterbrechen
- Unterbrechungsvektor
  - ⋆ im Arbeitsspeicher des BS
  - ⋆ für jede Unterbrechung dort Anfangsadresse des Antwortprogramms und Prozessorstatuswort
- Unterbrechung
  - ⋆ Kontext des laufenden Prozesses P_i retten
  - ⋆ Anfangsadresse Antwortprogramm und Prozessorstatuswort aus Unterbrechungsvektor holen
  - ⋆ Unterbrechungs–Antwortprogramm aufrufen
  - ⋆ unterbrechbar durch höherpriore Abläufe
  - ⋆ nach Rückkehr Kontext von P_i restaurieren
  - ⋆ P_i oder höherprioren Prozeß P_j fortsetzen
  - ⋆ unterbrechbar durch höherpriore Abläufe



- Fünf Klassen von Unterbrechnungsantwortprogrammen
  - a Mini-UAP
    keine Interaktion mit Betriebssystem
    nur Zugriff auf E/A-Register
  - b UAP mit nichtblockierendem Systemaufruf
    geeigneter Mechanismus zum Umschalten in Betriebssystemadreßraum (`enter kernel`)
    Dienst nicht an jeder Stelle ausführbar
  - c Kombination aus a und b
    In der Praxis oft mit kurzen ununterbrechbaren UAPs und Systemdiensten
  - d wie c, aber mit Pufferung der Systemaufrufe,
    falls BS gerade ununterbrechbar
  - e Maximallösung
    direkter Start einer Task
    sehr mächtig, aber langsam
- Zu Unterbrechungen gehören auch "Software–Interrupts"
  - ⋆ (Ausnahmesignale, traps, exceptions)
  - ⋆ durch Anweisungen des BS oder der Benutzerprogramme ausgelöst

- Sprachliche Mittel zur Anbindung von Unterbrechungs–Antwortprogrammen
  - ⋆ Taskstart:
    WHEN <Unterbrechungssignals> DO
    ACTIVATE <Taskname> ;
  - ⋆ Antwortprozedur:
    ON <Unterbrechungssignals> DO
    <Statement-Sequenz>
    <setze unterbrochenen Prozeß fort >
    END ON
  - ⋆ <Name des Unterbrechungssignals> muß Compiler/Binder auf die Hardware–Interrupt–Nummer abbilden
  - ⋆ Adresse des Antwortprogramms wird dynamisch im Unterbrechungs–Vektor eingetragen
  - ⋆ Antwortprogramm muß bei dieser Lösung statisch im Speicher bleiben.
  - ⋆ Eine saubere sprachliche Einbindung ist schwierig (Gültigkeitsbereiche, Rechte), daher existieren viele verschiedene Lösungen

9.6 UNIX für Echtzeitaufgaben?

- Warum UNIX
  - ⋆ Bei Cross–Entwicklungen möchte man nicht zwei Betriebssysteme kennen müssen
  - ⋆ Problem 1: UNIX nicht echtzeitfähig
  - ⋆ Problem 2: Bei kleinen Echtzeitanwendungen (embedded systems) fehlt notwendige HW (Platte, MMU)
- 1969 UNIX als Mehrbenutzersystem bei BELL durch Ken Thompson und Dennis Ritchie entwickelt
- Später viele Versionen, u.a. System V
- SVID: UNIX SYSTEM V Interface Definition
- seit 1984 Standardisierungen der Schnittstellen
  - ⋆ X/OPEN
    einheitliche Systemumgebung auf Basis SVID
  - ⋆ IEEE 1003
    Schnittstellen auf Basis SVID, u.a.
    - ∘ POSIX (Portable Operating System UNIX)
    - ∘ Realtime facilities (IEEE 1003.1b)

- Was fehlt UNIX für Echtzeitverhalten?
  - ⋆ Zeit nicht deterministisch projektierbar
    - ∘ System zu langsam (lange ununterbrechbare Phasen)
    - ∘ 100 – 1000 msec Reaktionszeit auch bei nur einem Benutzerprozeß
    - ∘ großer Prozeßkontext
    - ∘ keine "echten" Prioritäten
    - ∘ Zeitscheibentechnik
    - ∘ keine hauptspeicherresidenten Tasks
    - ∘ Kern nicht reentrant
  - ⋆ E/A- Zugang nur über vorhandene Treiber und Systempuffer
  - ⋆ Dateistruktur zu allgemein
  - ⋆ viele Kopiervorgänge zwischen Puffern
  - ⋆ keine geeignete Prozeßkommunikation:
    - ∘ Pipes (mit Kopieren verbunden),
    - ∘ Signale (unsicher);
    - ∘ Semaphor zu langsam implementiert
  - ⋆ SVID–Dienste meist zu allgemein und deshalb langsam

- Fünf Vorgehensweisen zur Abhilfe:

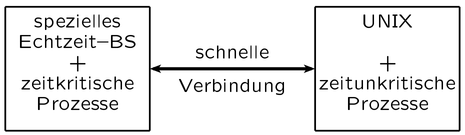
(1) Modifikation des Kerns, u.a.
  - ⋆ Prioritäten
  - ⋆ unterbrechbare Stellen im Kern

(2) Implementierung eines echtzeitfähigen Minikernels
Linux-Betriebsystemkern nur eine Echtzeittask mit der niedrigsten Priorität

(3) neuer Kern bei gleichen Aufrufen, z.B. LynxOS

(4) SVID oder POSIX auf geeignetem Echtzeit–Betriebssystem aufsetzen

(5) heterogenes Mehrrechnersystem



- POSIX Real–Time Extensions (IEEE 1003.1b)
  - ⋆ Prioritäten für Prozesse
  - ⋆ Präemption
  - ⋆ schneller Kontextwechsel
  - ⋆ Speicherfixierung aktiver Prozesse
  - ⋆ asynchrone E/A (Direkt–DMA von Gerät zu Prozeß)
  - ⋆ feineres Zeitraster (Uhren, Wecker)
  - ⋆ "Contiguous"–Dateien mit festen Plattenbereichen
  - ⋆ einfache Blockverwaltung bei Plattenspeicher
  - ⋆ Hilfsmittel für gemeinsame Datenbereiche
- Ausbauklassen (IEEE 1003.13)
  AEP (application environment profiles)

(1) Minimal real–time system for embedded systems (realtime kernel, threads)

(2) Real–time controller system (E/A, Dateien im Speicher, Ausnahmen)

(3) Dedicated real–time system (Prozesse, evtl. MMU)

(4) Multi–purpose real–time system (Vollausbau, Massenspeicher, Netz, FTAM)

9.7 RT-Linux

Grundidee:

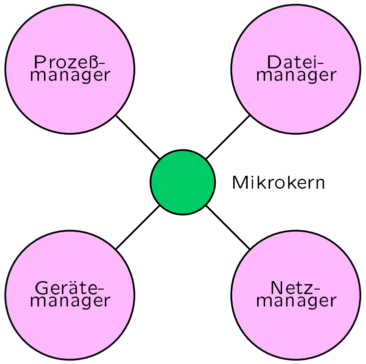
- Implementierung eines Echtzeitkerns
- Linux-Kern läuft als Task mit kleinster Priorität
- Kommunikation der Echtzeittasks mit Linux Prozessen über FIFO-Puffer

Unter `http://www.rtlinux.org/documents.html` finden sich detaillierte Beschreibungen zur Architektur von RTLinux

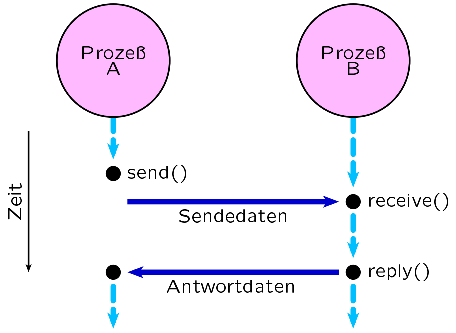
- The RT-Linux approach to hard real-time. Victor Yodaiken, Oct 97. This is the white paper on the first page of this site.
- Real-Time Linux. Barabanov and Yodaiken, Linux Journal, Mar. 96. A good overview.
- A Linux-based Real-Time Operating System. Michael Barabanov, M.S. Thesis, June 97. The most detailed explanation to date.
- Real-time Linux Applications and Design. Yodaiken, Usenix presentation[date?]. An overview in outline form (slides).

9.8 QNX realtime OS

- besteht aus Mikrokern, System- und Anwenderprozessen
- basiert auf Standard-PC-Hardware
- frei verfügbar unter `http://get.qnx.de/`
- Systemprozesse (außerhalb Mikrokern)
  - ⋆ Prozeßmanager
  - ⋆ Dateimanager
  - ⋆ Gerätemanager
  - ⋆ Netzmanager



- Mikrokern
  - ⋆ klein (auch für eingebettete System im ROM)
  - ⋆ skalierbar
  - ⋆ schnell (Kontextwechsel einige Mikrosekunden)
  - ⋆ kleiner BS-Overhead
  - ⋆ zuverlässig
- nachrichtenbasierte Interprozeßkommunikation



Anmerkung: receive nach FIFO oder nach Priorität

- Multitasking
- Multithreading
- präemptive RK-Vergabe nach Prioritäten
- Aufgaben Mikrokern
  - ⋆ Erzeugung Prozesse (soweit nicht im Prozeßmanager)
  - ⋆ Speichermanagement
  - ⋆ Zeit-, Wecker-, Uhrenverwaltung
  - ⋆ Interprozeßkommunikation
  - ⋆ Behandlung Unterbrechungen und Signale
  - ⋆ RK-Vergabe
- transparentes verteiltes System
  - ⋆ starten von Prozessen an beliebigem Ort des Netzes
  - ⋆ neue Prozesse erben Umgebung des Vaters, u.a.:
    - ∘ geöffnete Dateien
    - ∘ aktueller Pfad
    - ∘ Dateideskriptoren
    - ∘ Benutzer-ID

- POSIX-Eigenschaften, u.a.:
  - ⋆ POSIX.1b Uhren und Wecker
    - ∘ mehrere Wecker je Prozeß
    - ∘ Auflösung in Nanosekunden
    - ∘ flexible Weckersteuerung: Wecker arbeiten synchron oder asynchron; einmalig oder periodisch wiederholend
  - ⋆ voll geschachtelte Unterbrechungen
  - ⋆ dynamisch zuordenbare Unterbrechungsbehandlungen
  - ⋆ flexible Primitive für einen gemeinsamen Speicher
  - ⋆ eingebaute Testprimitive zur Fehlerlokalisierung irgendwo im Netz
  - ⋆ durch Benutzer konfigurierbare Grenzen für Ressourcen
  - ⋆ netzwerkuniverselle Namen

- ⋆ POSIX.1b Realzeit Scheduling
  - ∘ 32 Prioritäten
  - ∘ präemptiv, RK-Vergabe gemäß Priorität
  - ∘ Strategien (prozeßspezifisch wählbar)
    - ▹ grundsätzlich bei allen Strategien Präemption durch höherprioren Prozeß überlagert
    - ▹ FIFO (first in first out)
    - ▹ Zeitscheiben (round robin) Zeitscheibe (50 ms)
    - ▹ adaptive RK-Zuteilung Zeitscheibenverfahren; am Ende einer Zeitscheibe wird die Priorität eines Prozesses um 1 erniedrigt, falls sie nicht schon erniedrigt wurde; blockiert der Prozeß erhält er seine urpsrüngliche Priorität wieder
- ⋆ Server können ihre Priorität abhängig von ihren Aufträgen wechseln

- Dateisysteme
  - ⋆ mehrere verschiedene Dateisysteme können zur gleichen Zeit laufen
  - ⋆ POSIX-Dateisystem
  - ⋆ DOS-Dateisystem
  - ⋆ Netz-Dateisystem SMB (Server Message Block) für Windows NT, Windows 95, Windows for Workgroups, LAN Manager
  - ⋆ Netz-Dateisystem NFS
  - ⋆ CD-ROM-Dateisystem
  - ⋆ elementares Block-Dateisystem
- Fehlertoleranz und Flexibilität des Netzes
  - ⋆ dynamisches Zufügen, Entfernen, Anhalten und Fortsetzen von Knoten ohne explizite Rekonfiguration durch den Benutzer
  - ⋆ automatische Erkennung von Netzänderungen (Hinzufügen, Entfernen von LANs)
- Konfigurierung von QNX gemäß Bedarf der Anwendung

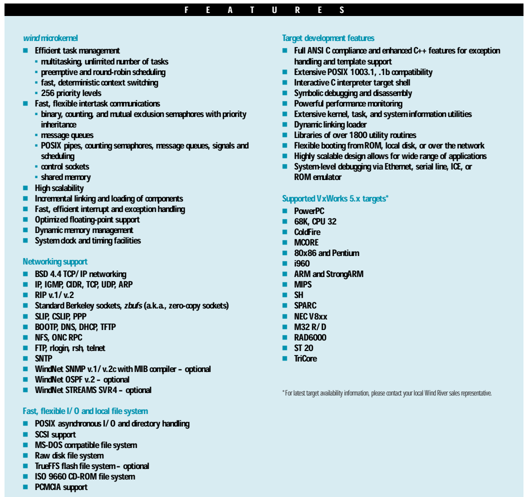
9.9 LynxOS

- Hard Real-Time Performance
- POSIX 1003 konform
- 256 Prioritäten für kernel threads
- präemptiver EZ-BS-Kern
- schneller Kontextwechsel, deterministische Blockierungszeiten
- Priority inheritance
- Semaphore, mutex, events, message queues
- clocks und timer
- UNIX-Dateisystem und Netzkommunikation
- skalierbar

Unter
`http://www.lynuxworks.com/products/whatislos.html`
finden sich detaillierte Beschreibungen zur Architektur und zur Leistungsfähigkeit von LynxOS

9.10 VxWorks 5.4

- charakteristische Eigenschaften



F E A T U R E S

wind microkernel
- Efficient task management
  - multitasking, unlimited number of tasks
  - preemptive and round-robin scheduling
  - fast, deterministic context switching
  - 256 priority levels
- Fast, flexible intertask communications
  - binary, counting, and mutual exclusion semaphores with priority inheritance
  - message queues
  - POSIX pipes, counting semaphores, message queues, signals and scheduling
  - control sockets
  - shared memory
- High scalability
- Incremental linking and loading of components
- Fast, efficient interrupt and exception handling
- Optimized floating-point support
- Dynamic memory management
- System clock and timing facilities

Networking support
- BSD 4.4 TCP/IP networking
- IP, IGMP, CIDR, TCP, UDP, ARP
- RIP v.1/v.2
- Standard Berkeley sockets, *zbufs* (a.k.a., zero-copy sockets)
- SLIP, CSLIP, PPP
- BOOTP, DNS, DHCP, TFTP
- NFS, ONC RPC
- FTP, rlogin, rsh, telnet
- SNTP
- WindNet SNMP v.1/v.2c with MIB compiler – optional
- WindNet OSPF v.2 – optional
- WindNet STREAMS SVR4 – optional

Fast, flexible I/O and local file system
- POSIX asynchronous I/O and directory handling
- SCSI support
- MS-DOS compatible file system
- Raw disk file system
- TrueFFS flash file system – optional
- ISO 9660 CD-ROM file system
- PCMCIA support

Target development features
- Full ANSI C compliance and enhanced C++ features for exception handling and template support
- Extensive POSIX 1003.1, .1b compatibility
- Interactive C interpreter target shell
- Symbolic debugging and disassembly
- Powerful performance monitoring
- Extensive kernel, task, and system information utilities
- Dynamic linking loader
- Libraries of over 1800 utility routines
- Flexible booting from ROM, local disk, or over the network
- Highly scalable design allows for wide range of applications
- System-level debugging via Ethernet, serial line, ICE, or ROM emulator

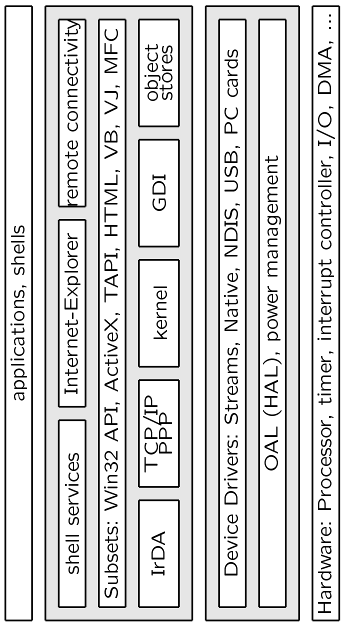
Supported VxWorks 5.x targets*
- PowerPC
- 68K, CPU 32
- ColdFire
- MCORE
- 80x86 and Pentium
- i960
- ARM and StrongARM
- MIPS
- SH
- SPARC
- NEC V8xx
- M32 R/D
- RAD6000
- ST 20
- TriCore

*For latest target availability information, please contact your local Wind River sales representative.

Unter `http://www.windriver.com/products/html/vxwks54.html` finden sich detaillierte Beschreibungen zur Architektur von VxWorks und der Entwicklungsumgebung Tornado

9.11 Windows-CE

- modular, schichtenorientiert
- Baukastensystem
- skalierbar (ca. 200 KB bis 8MB)
- versch. Hardware-Plattformen
- 32 bit
- MS-Windows-kompatibel (Teilmenge Win32 API)
- teilweise NT-Schnittstellen
- viele Kommunikationsprotokolle
- deterministisch
- derzeit noch "weiches" Realzeitverhalten
- 32 (Schwergewichts-) Prozesse
- dazu threads in den einzelnen Prozessen
- virtuelles Speichermanagement
- COM (Component Object Model) und ActiveX
- Spiegelung Dateisystem möglich
- integriertes Energie-Management
- erster Einsatz 1996 (handheld computer)



Schichten und Komponenten Windows CE

10 Echtzeitsprachen

10.1 Bemerkungen zur Historie

- 1960–1970
  - ⋆ Assemblerprogramme, da Speicher teuer
  - ⋆ frühe Ansätze zur Vereinheitlichung in Kernphysik: CAMAC–Geräteschnittstelle
  - ⋆ in den Progammen wurde "optimiert" ⇒ jedes Bit genutzt
- ab ca. 1966
  - ⋆ erster Einsatz von höheren Sprachen, z.B.
    - ∘ CORAL und RTL/2 (England)
    - ∘ ALGOL 60
    - ∘ FORTRAN IV (meistens)
  - ⋆ Prozeduraufrufe für Betriebssystem-Echtzeit-Dienste
  - ⋆ Probleme
    - ∘ viel Wissen über BS nötig
    - ∘ wenig portabel
    - ∘ keine semantische Prüfung der Parameter durch den Übersetzer, da keine speziellen Datentypen für z.B. task, semaphor, clock
    - ∘ schwierige Fehlersuche

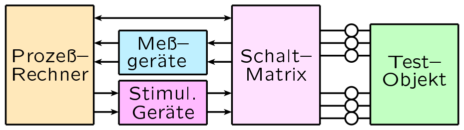
- ab 1970: erste Echtzeitsprachen
  - ⋆ nationale und internationale Normungen
    - ∘ PEARL (Deutschland) (Process and Experiment Automation Realtime Language)
    - ∘ HAL/S (USA)
    - ∘ PROCOL (Japan)
    - ∘ RT-FORTRAN
    - ∘ RT-BASIC
  - ⋆ Datentypen z.B.
    - ∘ task
    - ∘ clock
    - ∘ duration
    - ∘ sema
    - ∘ interrupt

    mit den zugehörigen Operationen in die Sprache integriert
  - ⋆ einheitliche Anweisungen für E/A und die Beschreibung von Datenwegen
  - ⋆ Vorteile
    - ∘ benutzerfreundliche Sprachelemente

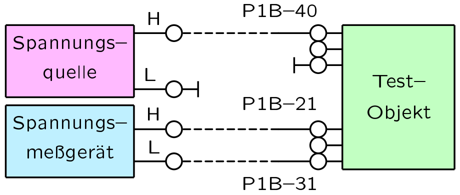
    - ∘ Semantik der Paramter bei BS–Aufrufen vom Übersetzer/Binder prüfbar
    - ∘ weitgehend portabel
  - ⋆ Nachteile
    - ∘ eigenes umfangreiches BS nötig (Speicher, BS–Overhead, Software–Kosten)
    - ∘ vorhandenes universelles BS wird angepaßt → Gefahren: ineffizient, überflüssige Teile im BS, Supersets der Sprache gefährden Portabilität
- ab 1978 universelle Echtzeitsprachen
  - ⋆ universelle sichere hohe Sprachkonzepte für alle Anwendungsbereiche
  - ⋆ Standardisierung, insbesondere DoD (ADA)
  - ⋆ Datentypen (z.B. task, duration, interrupt) oder systemabhängige Parameter in sprachlich sauberer Weise mittels Moduln / packages eingebunden
  - ⋆ Beispiele
    - ∘ ADA83, ADA95
    - ∘ CHILL
    - ∘ MODULA–2
    - ∘ PEARL, PEARL 90, Mehrrechner–PEARL

- heute
  - ⋆ eher Tendenz statt spezieller Echtzeitsprache, eine universelle Sprache (z.B. C, C++ oder JAVA) mit Bibliotheksprozeduren für Realzeitdienste anzureichern
  - ⋆ (herstellerspezifische) Speziallösungen für eingeschränkte Anwendungsbereiche, z.B.
    - ∘ Prüfwesen
    - ∘ Standard–Regelungsaufgaben
    - ∘ Fördertechnik
    - ∘ Visualisierung
    - ∘ Telefonanlagen
  - ⋆ meist als Progammbibliotheken
  - ⋆ Beispiele
    - ∘ SPS-Programmierung
    - ∘ ATLAS (Abbreviated Test Language for All Systems) für Prüfsysteme, insbesondere Flugzeug–Elektronik
    - ∘ ESTEREL und andere synchrone Sprachen (Automaten)
    - ∘ Sprachen zur Spezifikation von einfachen Echtzeitsystemen

- Anwendungsbeispiel Atlas
  - ⋆ Blockschaltbild Testsystem



  - ⋆ Prüfung Elektronik–Gerät



    - ∘ Verbale Testvorschrift

Lege 28V = an P1B–40
OK, falls 30mV zw. P1B-21 und P1B-31

- ∘ ATLAS–Programmausschnitt

```
100 302 APPLY, SOURCE, DC SIGNAL,
        VOLTAGE 28 ERRLMT + - 0.5V,
        CNX HI P1B-40 LO COMMON $
100 304 VERIFY, (VOLTAGE), DC SIGNAL,
        NOM 0.000V LL-0.030V UL 0.030V,
        CNX HI P1B-21 LO P1B-31 $
100 306 GOTO, STEP 105 084, IF NOGO $
        ...
105 084 PRINT, MESSAGE, 'FEHLER' $
```

10.2 Anforderungen an Echtzeitsprachen

- projektierbares Zeitverhalten
- wenig (kein) Overhead durch Laufzeitsystem
- effiziente Erstellung großer komplexer Systeme mit hohen Sicherheitsanforderungen (z.B. in Luft– und Raumfahrt, Verkehr, Medizin)
- Lesbarkeit, Übersichtlichkeit, Einfachheit durch wenige Konzepte
- Modularisierung und strenge Typprüfung als Voraussetzungen zur frühen Fehlererkennung durch Übersetzer, Binder und Laufzeitsystem
- überprüfbare Schnittstellen(beschreibungen) der Moduln
- unabhängiges Implementieren, Übersetzen, Testen von Moduln durch verschiedene Personen
- Daten- und Methodenkapselung in Moduln

- spezielle Datentypen, z.B.
  - ⋆ process (task, thread)
  - ⋆ exception
  - ⋆ semaphore, event
  - ⋆ bit, byte (auf jeden Fall Einzelbits setzen/löschen, z.B. in Statusregistern)
  - ⋆ clock, time, duration
- Zeitdienste
- Zeitüberwachung aller Wartezustände
- Prioritäten
- Ansprechen von Speicheradressen, z.B. für "memory mapped" E/A, Adreßrechnung
- Aktivierung von Prozessen
  - ⋆ sofort
  - ⋆ zu bestimmten Zeitpunkten
  - ⋆ zyklisch
  - ⋆ in bestimmten Zeitabständen innerhalb eines Zeitintervalls
  - ⋆ bei bestimmten Ereignissen
- einfache Dateistrukturen (z.B. fester Speicherbereich)

- Synchronisation (auch in verteilten Systemen)
- schnelle Kommunikation
- vielseitige Protokolle zur Kommunikation (Feldbus-Protokolle, LAN-Protokolle, V24 usw.
- Interrupt– und Ausnahme–Behandlung, insbesondere benutzerdefinierte Unterbrechungs–Antwortprogramme
- Unterstützung der vielseitigen Peripherie und der Coprozessoren
- Definition virtueller Geräteklassen mit einheitlichen Schnittstellen
- einheitliches E/A–Konzept für Standard– und Prozeß–E/A, einschließlich Beschreibung der Datenwege
  - ⋆ Beispiel:
Endlagenschalter Nr. 53 ist verbunden mit Eingang 5 einer Digitaleingabe auf Steckplatz Nr. 8
PEARL:
Datenweg: schalter53 $\rightarrow$ DIGIN $*8*5$
READ schaltzustand FROM schalter53;
IF schaltzustand = '1'B THEN ...;

  - ⋆ Übersetzer muß Namen der Datenwege kennen und die korrekten Treiberprogramme mit Parametern einsetzen
  - ⋆ bei Änderung des Datenweges (z.B. Umstecken) nur Wegbeschreibung aufdatieren; Programm nicht ändern, aber neu übersetzen (neue Treiber bzw. geänderte Parameter)
- Evtl. Funktionen für verteilte Systeme, z.B
  - ⋆ Operationen auf Daten auf anderen Rechnern im Netz
  - ⋆ Objektverwaltung im Netz (aktuelle Verzeichnisse der Namen mit Attributen, lokale Teilkopien)
  - ⋆ Konfigurierungs–Funktionen zur Zuordnung der Programm–Moduln zu Rechnern
  - ⋆ (automatische) Rekonfiguration in Fehlersituationen
  - ⋆ Synchronisierung Uhren

- Anmerkung:
Viele Fragen bei der Sprachdefinition sind immer noch offen, z.B.
  - ⋆ Bei Ausnahmen: Wo wird nach der Behandlung fortgesetzt?
  - ⋆ Soll es untergeordnete Tasks (Hierachien) geben?
  - ⋆ Darf eine Task neu gestartet werden, falls die vorhergehende Inkarnation noch aktiv ist?
  - ⋆ Darf eine Task eine andere löschen und wenn ja, in welchem Zustand ?

  Beispiel für mögliche Fehlersituation:

P1:		P2:
öffneVentil (Nr53);		...
delay (2 sec);	←	TERMINATE (P1);
schließeVentil (Nr53);		...

10.3 Modul–Strukturen

- Moduln enthalten:
  - ⋆ Daten
  - ⋆ Typdefinitionen
  - ⋆ lokale und/oder globale Prozeduren
  - ⋆ Tasks
- evtl. Konfigurierungseinheiten in verteilten Systemen
- ermöglichen Datenkapselung (objektorientierte Konzepte mit Vererbung)
- Modularten
  - ⋆ Funktionsmodul (Bibliotheksmodul)
nur Funktionen und Prozeduren im Modul
  - ⋆ Datenmodul
nur permanente Objekte oder globale Systemdaten im Modul
  - ⋆ Objektmodul
(Abstrakte Datenstruktur, Rechenstruktur)
Daten und zugehörige Operationen im Modul; von außen sichtbare Schnittstelle ist definiert (Geheimnisprinzip)

  - ⋆ Klassenmodul (Modul als Schema)
(classes, templates, generic packages)
erst durch Instantiierung (constructors) entstehen Exemplare
objektorientierte Programmierung
- Sprachen trennen die Modulbeschreibung in
  - ⋆ Schnittstellenbeschreibung
    - ◦ Definitionsteil, Kopf, head, interface
    - ◦ Deklaration exportierter Objekte (Typen, Daten, Funktionen)
    - ◦ bei Prozeduren: Spezifikation Typ aller Parameter!
    - ◦ Spezifikation importierter Objekte
    - ◦ "opaque type"–Objekte, private Objekte
Datentyp im Modul definiert, aber von außen nur der Name sichtbar
  - ⋆ Implementierungsteil
    - ◦ Rumpf, body
    - ◦ Realisierung der exportierten Objekte
    - ◦ Realisierung der internen Objekte
    - ◦ Initialisierung der Objekte
- möglichst Erkennung aller Schnittstellenfehler beim Übersetzen oder spätestens beim Binden

10.4 Programmstruktur und Moduln in Ada

- Freie Ada-Compiler für Windows und Linux unter
`http://www.adahome.com/Resources/Compilers`
  - ⋆ ObjectAda Compiler von Aonix
  - ⋆ GNAT: The GNU NYU Ada 9X Translator
- Programm strukturiert in
  - ⋆ Blöcke
  - ⋆ Unterprogramme
  - ⋆ Pakete
  - ⋆ Tasks
- beliebige Schachtelung
- Hauptprogramm ist eine Prozedur (main task)
- Übersetzungseinheiten
  - ⋆ Bibliothekseinheiten (Bibl–E)
    - ◦ Unterprogramm–Vereinbarung
    - ◦ Unterprogramm–Rumpf
    - ◦ Paketvereinbarung
    - ◦ Vereinbarung einer generischen Einheit

- ⋆ Sekundär–Einheiten
  - ∘ Unterprogramm– oder Paketrumpf
- Pakete und Tasks müssen, Unterprogramme können in Spezifikation (Kopf, head) und Rumpf (body) aufgeteilt werden
- im Kopf stehen die sichtbaren (Export–) Objekte
- die Angabe private spezifiziert die opaken Objekte, deren Realisierung verborgen (nicht ausnutzbar) bleibt
- Anweisungsteil des Pakets wird einmalig beim Abarbeiten der Paketdeklaration ausgeführt
- Syntax Paket

```
PACKAGE <name> IS
   <sichtbare Vereinbarungen> ;
   [PRIVATE <Vereinbarungen> ;]
END <name>

PACKAGE BODY <name> IS
   <lokale Vereinbarungen> ;
BEGIN
   <Anweisungen>
   [EXCEPTION <Ausnahmebehandlung> ]
END <name> ;
```


- Spezifikation muß vor Rumpf übersetzt werden
  - ⋆ fehlender Prozedurrumpf kann durch Stummelrumpf (stub unit) ersetzt werden. Beispiel:

```
PACKAGE BODY master IS
  ...
  PROCEDURE q (x, y: REAL) IS
    SEPARATE;
  ...
END master;
```

  - ⋆ und getrennt unter Angabe des Pakets, in dem sich der Stummelrumpf befindet, übersetzt werden. Beispiel:

```
SEPARATE (master)
PROCEDURE q (x, y: REAL) IS
  BEGIN
  ...
  END q;
```


- Private Typen
  - ⋆ Realisierung privater Typen kann zwar gesehen (z.B. wegen Speicherbedarf), aber nicht geändert oder benutzt werden
  - ⋆ Typ im Rumpf muß mit Spezifikation im Kopf übereinstimmen
  - ⋆ Änderung der Spezifikation erfordert Neuübersetzung aller abhängigen Bibliothekseinheiten bzw. benutzenden Programme
  - ⋆ in Spezifikation eines Pakets
    `TYPE <name> IS [LIMITED] PRIVATE`
  - ⋆ im Rumpf dann
    `PRIVATE <Spezifikation von name>`
  - ⋆ ohne LIMITED: außerhalb muß Zuweisung und Vergleich (= oder ≠) für diesen Typ existieren
  - ⋆ mit LIMITED: Zuweisung und Vergleich durch eigene Paketfunktionen

- Beispiel

```
PACKAGE complex_numbers IS
  TYPE compl IS PRIVATE;
  i: CONSTANT compl;
FUNCTION + (x, y: compl) RETURN compl;
FUNCTION – (x, y: compl) RETURN compl;
PRIVATE
  TYPE compl IS
    RECORD
      re, im: FLOAT;
    END RECORD;
  i: CONSTANT compl := (0.0, 1.0);
END complex_numbers;

PACKAGE BODY complex_numbers IS
...
FUNCTION + (x, y: compl) RETURN compl IS
  BEGIN ... END +
FUNCTION – (x, y: compl) RETURN compl IS
  BEGIN ... END –
END complex_numbers;
```



```
Aufruf in Block, z.B.:
DECLARE
WITH complex_numbers;
USE complex_numbers;
u,v: compl;
BEGIN
u:= (2.5, 1.0);
v:= i;
u:= u + v;
END
```

- Anmerkungen
  - ⋆ WITH <Bibliothekseinheit>
    - ∘ Benutzung Bibliothekseinheit bei Übersetzung
    - ∘ Bibliothekseinheit muß bei der Übersetzung schon bekannt sein (mindestens mit ihrem Definitionsteil)
    - ∘ zyklische WITH–Relationen daher nicht möglich
  - ⋆ USE <Bibliothekseinheit>
    - ∘ im Rumpf kann der qualifizierende Paketname entfallen

- Generische Einheiten
  - ⋆ Unterprogramme oder Pakete
  - ⋆ dienen als Programmschablonen zur Mehrfachverwendung von Algorithmen
  - ⋆ freie Parameter (generic parameters) nach dem Wortsymbol GENERIC
  - ⋆ Parameter sind Objekte und Objekttypen
  - ⋆ freie Parameter bei Übersetzung der Ausprägung im Rumpf textuell durch die aktuellen Parameter ersetzt (entspricht templates in C++)
  - ⋆ Ausprägung (Instantiierung) wird durch NEW und Belegung der freien Parameter deklariert
  - ⋆ Beispiel: Generisches Paket zur Erzeugung von Ringpuffern verschiedener Länge und mit Elementen verschiedener Art


```
GENERIC
TYPE sometype IS PRIVATE;
PACKAGE queue_handling IS
TYPE queue (maxlength: NATURAL)
  IS PRIVATE;
PROCEDURE enqueue (q: IN OUT queue;
  elem: IN sometype);
  ...
PROCEDURE dequeue ...
FUNCTION nonempty ...
FUNCTION nonfull ...
PRIVATE
SUBTYPE index
IS CARDINAL RANGE 0..1000;
TYPE queue (maxlength: NATURAL) IS
RECORD
head: index := 0;
tail: index := 0;
q: ARRAY (0..maxlength-1) OF sometype;
END RECORD;
END queue_handling;
```



```
PACKAGE BODY queue_handling IS
...
END queue_handling;
```

- Im Rumpf befinden sich die Prozeduren enqueue, dequeue, nonfull, nonempty. Es gibt keinen ausführbaren Teil im Rumpf.
- Ausprägung, z.B. eines Blocks

```
DECLARE
PACKAGE int_queue
IS NEW queue_handling (INTEGER);
USE int_queue;
longqueue: queue (200);
shortqueue: queue (10);
BEGIN
IF nonfull(shortqueue) THEN ...;
...
END int_queue;
```

10.5 Objektorientierung in ADA95

- ADA83 hat strenges Typkonzept mit enger Datenkapselung
- zur besseren Wiederverwendbarkeit von Paketen durch Erweiterung der Funktionen wird in ADA95 Objektorientierung mit Vererbung hinzugefügt
- Objektorientierung basiert in ADA 95 auf der Typerweiterung von RECORD (TAGGED), ähnlich STRUCT in C++
- Beispiel:

```
TYPE messwert IS TAGGED
  RECORD
    m1, m2 : INTEGER;
  END RECORD;
```

Erweiterung um Zeitstempel in anderem Paket z.B.

```
TYPE zeitmesswert
    IS NEW messwert WITH
  RECORD
    ankunft: time;
  END RECORD;
```


- abgeleitete Typen erben die Operationen des Vorgängers, können diese umdefinieren und neue hinzufügen (Typhierarchie mit Einfachvererbung)
- in ADA95 trägt die gesamte Ableitungshierarchie (Grundtyp mit Ableitungen) die Bezeichnung "Klasse"
- Zum Grundtyp T existiert der Klassentyp T'CLASS (polymorpher Typ), der als formaler Parameter beim Überladen von Unterprogramm-Namen dienen kann
- beim Aufruf mit einem aktuellem Parameter aus der Ableitungshierarchie wird dynamisch die passende Operation verwendet
- dieses Vorgehen heißt in ADA95 dispatching (spätes Binden)

- Abstrakte Typen und Unterprogramme
  - ⋆ auf oberstem Niveau können abstrakte Typen (dh. ohne Angabe des Inhalts) und abstrakte Unterprogramme (ohne Angabe eines Rumpfes) definiert werden
  - ⋆ in der Ableitungshierarchie kann dieser Grundstock von allgemeinen Typen und Operationen auf verschiedene Weise realisiert werden
  - ⋆ der Anwender sucht sich unter den Varianten die für seine Aufgabe geeignete Implementierung aus
  - ⋆ Beispiel
    - ◦ abstrakter Typ: menge
    - ◦ abstrakte Operationen: einfügen. löschen, istenthalten, Vereinigung, Durchschnitt
    - ◦ als Implementierung kann der abgeleitete Typ menge in verschiedener Weise angeboten werden, z.B. als
      - ▷ BitSET (bei kleiner Grundmenge)
      - ▷ Feld
      - ▷ als geordnete einfach verkettete Liste

- Hierarchische Bibliotheken
  - ⋆ bei Wiederverwendung vorhandener Pakete sollen Erweiterungen nicht zur Neuübersetzung aller bisherigen Benutzer des Grundpakets führen, da sie ja die neuen Operationen nicht verwenden
  - ⋆ Erweiterung darf also nicht in Grundpaket eingehen
  - ⋆ ADA95 führt aus diesem Grund das Konzept von Kind–Paketen (child packages) ein und erlaubt die Definition eines Baums von Paket–Spezifikationen (hierarchical libraries)
  - ⋆ ein Kind–Paket erbt vom Eltern–Paket alle sichtbaren Operationen und darf im Rumpf alle privaten Spezifikationen des Elternpakets verwenden
  - ⋆ durch Weitervererbung wird eine Paket–Hierarchie aufgebaut
  - ⋆ Kind–Paket enthält bei Spezifikation den (qualifizierenden) Namen des Elternpakets

- ⋆ Beispiel: Paket Für Komplexe Zahlen wird um Darstellung in Polarkoordinaten erweitert

```
PACKAGE complex_numbers IS
  TYPE compl IS PRIVATE;
  FUNCTION + (x, y: compl) RETURN compl;
  PRIVATE
    TYPE compl IS
      RECORD
      re, im: FLOAT;
      END RECORD;
END complex_numbers;
PACKAGE complex_numbers.polar IS
  TYPE polar IS PRIVATE;
  FUNCTION abs (x: compl) RETURN FLOAT;
  FUNCTION arc (x: compl) RETURN FLOAT;
  FUNCTION polar2compl (r, theta: FLOAT)
      RETURN compl;
  PRIVATE
    TYPE polar IS
      RECORD
      r, theta: FLOAT;
      END RECORD;
END complex_numbers.polar;
```


- ⋆ Kindpakete können PRIVATE spezifiziert werden
- ⋆ ADA95 gibt Regeln, die die Beziehungen von Eltern–Kind–Paketen und zwischen Geschwister–Paketen beschreiben, um zu verhindern, daß in untergeordneten Paketen durch Umdefinition PRIVAT–Spezifikationen sichtbar gemacht werden können
- ⋆ Beispiele zu den Regeln
  - ◦ Kontext–Umgebung des Elternpakets gilt auch für Nachkommen
  - ◦ WITH–Benutzung eines Kindpaket umfaßt auch WITH für alle Vorfahren
  - ◦ Kind benötigt keine WITH–Beziehung zu seinen Eltern, jedoch zu den Geschwistern
  - ◦ sichtbares Kind kann kein PRIVATE–Geschwisterpaket verwenden
  - ◦ PRIVATE–Teil und Rumpf jedes Kindes kann PRIVAT–Teil seiner Eltern (und aller Vorfahren) benutzen

10.6 Tasks in ADA

- für Tasks (Prozesse) Datentyp TASK
- Deklaration in Prozeduren oder Paketen
- auch Komponenten von Feldern oder Records
- auch aktuelle Parameter
- Tasks werden automatisch beim Abarbeiten der Deklaration aktiv, aber erst am Ende dieses Deklarationssteils gestartet
- Tasks terminieren automatisch beim Erreichen des Blockendes, falls sie nicht auf des Ende von untergeordneten Tasks warten müssen
- Block wird erst verlassen, wenn alle in ihm vereinbarten Tasks beendet sind
- keine Parameter beim Start einer Task
- bei Deklaration ist Trennung in Spezifikationsteil und Rumpf obligatorisch
- Spezifikationsteil darf nur Eingänge (ENTRY) für Aufrufe von Rendezvous–Anweisungen aus anderen Tasks deklarieren mit Angabe der formalen Parameter der zugehörigen ACCEPT–Anweisung im Rumpf

- Syntax

```
TASK [TYPE] name IS
  ENTRY ename (<formale Parameter> );
  ENTRY ...
END name;

TASK BODY name IS
  <deklarationen>
BEGIN
  <Anweisungen>
  [EXCEPTION [<exception_handler> ]]
END name;
```

bei Anweisungen u.a.:

```
ACCEPT ename (<formale Parameter> ) DO
  ...
END ename;
```

- es gibt nur die Operation ABORT zum Datentyp TASK (gewaltsames Beenden)
- durch die Blockstruktur können Taskaufrufe geschachtelt auftreten. Die umfassende Task wird durch implizite Synchronisation des Betriebssystems erst beendet, wenn alle in ihr deklarierten und damit gestarteten Tasks (subtasks) beendet sind (Ausnahme: Tasks die durch Zeiger–Kreierung aktiviert wurden).

10.7 Partitionen in ADA 95

- ADA 95–Programm kann aus mehreren Partitionen (partitions) bestehen
- entsprechen einem ADA 83–Programm
- haben einen eigenen Adreßraum
- können Tasks enthalten
- Programm kann aus Partitionen auf verschiedenen Rechnern zusammengesetzt sein
- active partitions enthalten tasks und main ()
- passive partitions enthalten nur Daten und/oder Unterprogramme
- Partition wird erst beendet, wenn alle ihre Tasks beendet
- werden von außen oder durch eine sog. Environment–Task angestoßen, bei deren Abarbeitung die in ihr enthaltene main–Prozedur aufgerufen wird
- Partition kann mit einer anderen mittels RPC (remote procedure call) oder über gemeinsame Daten einer dritten Partition kommunizieren

10.8 Synchronisation

- Prozesse, die an gemeinsamen Aufgaben kooperieren, müssen sich koordinieren (Synchronisation) und über Prozeßgrenzen (Adreßraum, Rechnernetz) Daten austauschen (Kommunikation)
- Synchronisation erforderlich wegen
  - ⋆ Konkurrenz und Konsistenz bei Benutzung exklusiver Betriebsmittel
  - ⋆ Einhaltung von Reihenfolgen (Produzent vor Konsument)
  - ⋆ Auftraggeber/Auftragnehmer–Diensten (client server)
- Synchronisations–Operationen legen nicht unbedingt die Reihenfolge der Zuteilung fest (faire oder unfaire (prioritätengebundene) Zuteilung)
- logische Synchronisation:
  es wird feste zeitliche Reihenfolge von Aktionen in verschiedenen Prozessen erzwungen, z.B. Prozeß P_i erst fortsetzen, wenn P_j beendet

- betriebsmittelorientierte Synchronisation: Operationen auf gemeinsamen Objekten (Variablen, Betriebsmittel) dürfen von verschiedenen Prozessen aus nur mit Einschränkungen erfolgen, z.B. nicht gleichzeitig (wechselseitiger Ausschluß, mutual exclusion)
- Lösungen (vgl. Grundvorlesungen) u.a. durch Semaphore (Dijkstra 68) mit den unteilbare Operationen P (sema) und V (sema)
- Spezielle Echtzeitforderungen
  - ⋆ Eine V–Operation wählt (unfair) den Prozeß höchster Priorität aus der Warteschlange zu diesem Semaphor
  - ⋆ Zeitüberwachung des Wartezustandes
  - ⋆ sichere implizit arbeitende Sprachkonstrukte
  - ⋆ bei Fehlerabruch im kritischen Bereich keine Blockade wartender Prozesse
  - ⋆ Prioritätsvererbung

- Rendezvous in ADA83
  - ⋆ Synchrone Kommunikation
  - ⋆ Definition eines Eingangs in einer Task
  - ⋆ ACCEPT-Anweisung zu den ENTRYs in dieser Task
  - ⋆ Aufruf dieses Eingangs aus einer andern Task wie ein Prozduraufruf mit Parametern
  - ⋆ sowohl aufrufende als auch aufgerufene Task warten, bis Anweisungen im ACCEPT ausgeführt sind
  - ⋆ alternatives Warten durch SELECT mit guards (WHEN)
  - ⋆ watchdog möglich

```
GENERIC
  TYPE item IS PRIVATE
PACKAGE readerwriter IS
  PROCEDURE read(x: OUT item);
  PROCEDURE write(x: IN item);
END;

PACKAGE BODY readerwriter IS
  V: item;
```



```
TASK control IS
  ENTRY start;
  ENTRY stop;
  ENTRY write(x:item);
END;

TASK BODY control IS
  readers: integer :=0;
BEGIN
  ACCEPT write(x:IN item) DO
    v := x;
  END;
  LOOP
    SELECT
      WHEN write'count=0 =>
      ACCEPT start;
      readers:=readers+1;
    OR
      ACCEPT stop;
      readers:=readers-1;
    OR
      WHEN readers=0 =>
      ACCEPT write(x:IN item) DO
        v := x;
      END;
```



```
      END SELECT;
    END LOOP;
  END control;

  PROCEDURE read(x:OUT item) IS
  BEGIN
    control.start; x:=v; control.stop;
  END read;

  PROCEDURE write(x:IN item) IS
  BEGIN
    control.write(x);
  END write;

END readerwriter
```

Beispiel für Zeitbedingungen im rendezvous:

```
SELECT
  ACCEPT this(···) DO
    ···
  END;
OR
  ACCEPT that(···) DO
    ···
  END;
OR
  DELAY 10.0;
    ··· – time out statements
END SELECT;
```


- Protected type in ADA 95
  - ⋆ ADA 83 ohne direkte Operationen für exklusive Programmteile. Semaphore durch eigenen Prozeß darzustellen, wobei durch synchrone Kommunikation die Exklusivität gesichert wird. Dieses Vorgehen über Tasks ist ineffizient
  - ⋆ daher in ADA 95 als Typ und Deklaration PROTECTED [TYPE]
  - ⋆ Objekte dieses Typs beinhalten (zusammengesetzte) Typen und Daten sowie die benötigten Operationen (Funktionen, Prozeduren, ENTRY)
  - ⋆ Laufzeitsystem sichert, daß PROTECTED PROCEDURES nur exklusiv ausgeführt werden
  - ⋆ Priority Inheritance hier möglich: Angabe von PRAGMA Locking_Policy (Ceiling_Locking)
  - ⋆ beim Auftreten einer Ausnahme im kritischen Bereich wird der Block verlassen und die Belegung automatisch aufgehoben, sodaß keine Blockade anderer erfolgen kan

- ⋆ auf PROTECTED FUNCTION kann gleichzeitig mehrfach lesend zugegriffen werden (nur IN–Parameter), da read–only–Eigenschaft ohne Nebeneffekte
- ⋆ Vorgehen ähnelt Monitoren
- ⋆ PROTECTED PROCEDURES in ADA 95 als Interrupt–Antwortprogramme
- ⋆ Definition wie Pakete mit sichtbarem und privatem Teil
- ⋆ Beispiel

```
PROTECTED TYPE datensatz IS
  PROCEDURE schreibe (n: IN index;
                      wert: item);
  FUNCTION lese (n: index) RETURN item;
PRIVATE
  tabelle: itemarray(index);
END datensatz;

PROTECTED BODY datensatz IS
  PROCEDURE schreibe (n: IN index;
                      wert: item);
    BEGIN ... END schreibe;
  FUNCTION lese (n: index) RETURN item;
    BEGIN ... END lese;
END datensatz;
```

10.9 Ausnahmen (exceptions)

- Ausnahmen sind Unterbrechungssignale aus dem aktuellen Programmablauf
- Beispiele
  - ⋆ arithmetische Alarme
  - ⋆ Speicherschutzalarme
  - ⋆ Fehlermeldungen vom Laufzeitsystem wie z.B. Indexgrenzenüberschreitung
- Echtzeitsysteme erfordern zumindest eine saubere, abschließende Behandlung und im Notfall den Übergang in einen "fail–safe–Zustand" des technischen Prozesses
- Sprachmittel für spezifische Behandlung auf Benutzerebene notwendig
- Benutzer kann in der Sprache Ausnahmen selbst (logisch) definieren und auslösen; nützlich, um bei bestimmten Ereignissen, die das Programm entdeckt, gezielt den sequentiellen Programmablauf zu durchbrechen

- problematisch ist Festlegung der Fortsetzungsstelle nach Ausführung (abhängig von der Block– und Modulhierarchie)
  - ⋆ Variante Terminierung
    - verursachende Programmstelle wird verlassen
    - nach Setzen von Zustandsinformation wird in die nächsthöhere Instanz zurückgekehrt (z.B. zur Aufrufstelle der verursachenden Prozedur)
    - die Ausnahme–Behandlung kann mehrfach nach oben weitergereicht werden bis zu einer abschließenden Beantwortung
  - ⋆ Variante Reparatur und Wiederholung
    - nach Reparatur wird an einer definierten Stelle fortgesetzt
    - bei einer Wiederholung kann ein anderer Algorithmus gleicher Funktion vorgesehen werden (partielle Diversität)

- PEARL
  - ⋆ SIGNAL
  - ⋆ alle SIGNAL–Namen vordefiniert und im SYSTEM–Teil beschrieben
  - ⋆ neben EOF, OVERFLOW u.a. sind implementierungsabhängige Namen für Laufzeitfehler vorgesehen, z.B. geplanter Startzeitpunkt schon verstrichen, Operationen an nicht (mehr) existierenden Task
  - ⋆ ON–Anweisung zur Beantwortung
  - ⋆ ON signalname DO <anweisungen>
  - ⋆ ON signalname DO SYS für Default–Behandlung
  - ⋆ beim Verlassen des definierenden Blocks wird die übergeordnete ON–Anweisung aktiv bzw. beim Fehlen die SYS–Reaktion

- ADA
  - ⋆ EXCEPTION
  - ⋆ Ausnahmen können in Anweisungen, bei Deklarationen und im Rendezvous auftreten
  - ⋆ Benutzer kann Ausnahmen selbst definieren

    ```
    ausn3: EXCEPTION;
    ```

  - ⋆ vor- oder benutzerdefinierte Ausnahmen durch RAISE auszulösen

    ```
    RAISE ausn3;
    ```

  - ⋆ Anweisungen zur Behandlung sind am Ende eines Rahmens (Block, UP–Rumpf, TASK, PACKAGE) anzugeben
  - ⋆ nach dem Eintreten werden die Anweisungen abgearbeitet und der Rahmen verlassen
  - ⋆ falls keine Behandlung angegeben, wird die Ausnahme an den umgebenden Rahmen weitergereicht (exception propagation), bis eine Behandlung oder letzlich ein Programmabbruch erfolgt

- ⋆ Syntax

```
EXCEPTION
   WHEN exceptionname =>
         <Anweisungsfolge> ;
   ...
   WHEN OTHERS =>
         <Anweisungsfolge> ;
- - Ende des Rahmens
```

- ⋆ an Stelle des Namens der Ausnahme auch mit | getrennte Liste von Namen möglich
- ⋆ OTHER am Ende der WHEN–Folge
- ⋆ es gibt viele vordefinierte Ausnahmen wie
  - constraint_error
  - numeric_error
  - programm_error
  - storage_error
  - tasking_error

10.10 Unterbrechungs–Antwortprogramme

- Unterbrechung (interrupt) kommt von CPU, Peripherie oder Prozessen
- Unterbrechung durchbricht sequentiellen Programmablauf
- Formulierung von Unterbrechungs–Antwortprogrammen auf Sprachebene ist wichtige, effizient zu realisierende Forderung der Echtzeitprogrammierung
- selbst wenn vorgefertigte Gerätetreiber viele Interrupts vom Benutzer fernhalten, bleiben Interrupts zur Prozeßführung, die nur spezifisch behandelbar
- Antwortprogramme fest (statisch) oder kurzzeitig (dynamisch) zugeordnet
- default–Behandlung durch Betriebssystem

- ADA 83
  - ⋆ Antwortprogramm wird als Rendezvous (Kommunikation) mittels ACCEPT–Anweisung in einer Task ausgeführt
  - ⋆ zugehörige ENTRY–Spezifikation im Task–Kopf enthält Interrupt–Identifikator
  - ⋆ Interrupt wird vom Laufzeitsystem in ENTRY–Aufruf umgesetzt
  - ⋆ Nachteile: ineffizient und verzögert durch Taskwechsel

- ⋆ Beispiel

```
TASK intantwort IS
   ENTRY alarm;
   FOR alarm USE AT 8♯ 220♯;
      - - Interrupt 144
   PRAGMA PRIORITY (120)
END intantwort;

TASK BODY intantwort IS
   ...
      - - Deklarationen
BEGIN
   LOOP
      ACCEPT alarm DO
         ...
      END;
   END LOOP;
END intantwort;
```

- ADA 95
  - Nachteile des Konzepts von ADA 83 durch Verwendung von PROTECTED PROCEDURES als Antwortprogramme weitgehend vermieden
  - mit priority_ceiling_locking werden derartige Prozeduren beim Auftreten des Interrupts exklusiv mit genügend hoher Priorität ausgeführt
  - Gesamtprioritätsbereich hängt vom Betriebssystem ab
  - Prioritäten für Tasks mindestens 30 Werte
  - höhere Prioritätsklasse für Interrupts
  - Zuordnung zu Signalen statisch oder dynamisch
  - Identifikatoren der Unterbrechungen vom Hersteller im Paket ADA.Interrupt.Names

- Beispiel statische Zuordnung

```
PROTECTED PROCEDURE alarm IS
  PROCEDURE response;
    - - muß parameterlos sein
  PRAGMA
    attach_handler (response, Alarm_Id);
END alarm;
PROTECTED PROCEDURE BODY alarm IS
  PROCEDURE response IS
    ...
  END response;
END alarm;
```

- Beispiel dynamische Zuordnung

```
PROTECTED PROCEDURE alarm IS
  PROCEDURE response;
    - - muß parameterlos sein
  PRAGMA interrupt_handler (response);
END alarm;

PROTECTED PROCEDURE BODY alarm IS
  - - wie oben

 - - späterer Prozeduraufruf:
attach_handler (alarm.response, Alarm_Id);
```


10.11 Real-Time JAVA

- Real-Time Specification for Java (RTSJ)
- ausgearbeitet durch Real-Time for Java Expert Group (RTJEG)
- first release update Juni 2000
- Wesentliche Konzepte:
  - präemptiver Scheduler mit mindestens 28 Prioritäten
  - Prioritätsvererbung
  - no heap realtime thread
  - abstract class: memory area mit physical, immortal and scoped memory
  - Monitor, no wait queues, event, timer
  - asynchronous interrupt exception
- PDF-Dokument mit Syntax und Semantik unter http://www.rtj.org/

10.12 Multi-Agent Distributed Real-Time System

- Eigenentwicklung am Lehrstuhl
- Echtzeit Programmierung mit verteilten Agenten
  - Top Down Zerlegung der Anwendung in autonome Agenten
  - zustandsbasiertes regelorientiertes Programmiermodell
  - Spezifikation der Aufträge (contracts), die Agent annimmt
  - Möglichkeit der Verhandlung (contract net protocol)
  - Verteilung Agenten auf heterogene Rechner
  - Entwurf unabhängig von Hardware
  - Transparenz der Kommunikation
  - Isolation der Ein/Ausgabe in Subagenten
  - bestimmbares Zeitverhalten der Agenten
  - zeitkritische Agenten in eigene Hardware

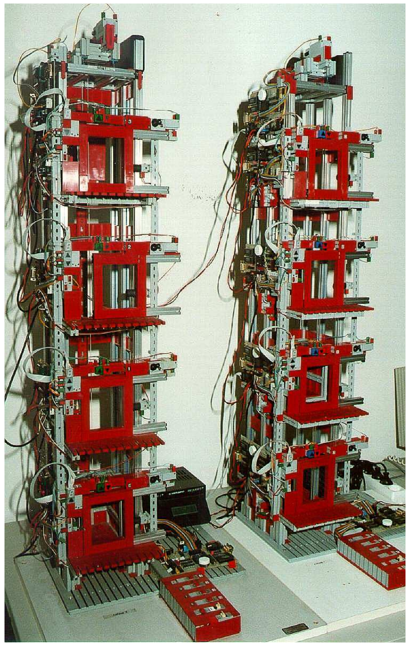
- Sprachdefinition
  - globale Deklaration der verwendeten Zielrechner (**targetdecl**)
  - globale Deklaration der verwendeten Kommunikationsverbindungen (**conndecl**)
  - Vollständige Deklaration eines Agenten
    - Zielrechner (**target**)
    - Instantiierung der Subagenten (**decls**)
    - Deklaration der aufrufbaren Aufträge (**contracts**) mit
      - Name und Parameter
      - Kostenfunktion für Verhandlung (**cost**)
      - Ausführungsteil (**do**)
    - Aktionsteil (**states**)

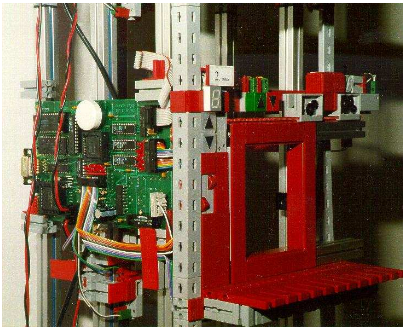
Beispiele für Subagenten:

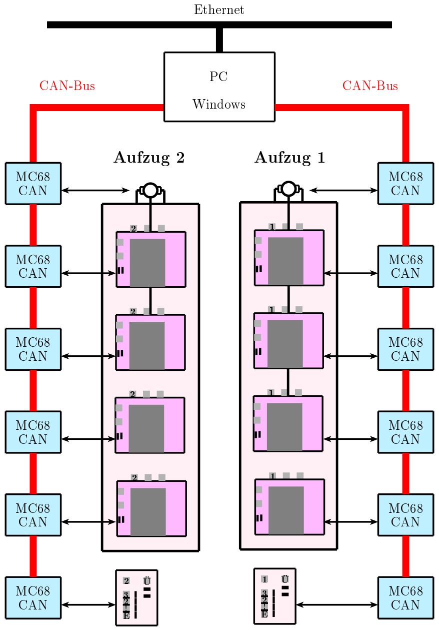
Definition		**contracts**
VARINT	<Name>	.set, .put
ARRAYINT	<Name> (<Dimension>)	.set, .put
DIGIN	<Name> (<Kanal> ,<id-on> , <id-off>)	.on, .off .<id-on> , .<id-off>
DIGOUT	<Name> (<Kanal> ,<id-on> , <id-off>)	.on, .off .<id-on> , .<id-off> .toggle(t_on,t_off)
ANAIN	<Name> (<Kanal>)	.get
ANAOUT	<Name> (<Kanal>)	.put(<value>)
TIMER	<Name>	.restart, .tout(<delay>)
CFUNC	<Bib-Name>	.<C-subroutine-id> (...)
PID	<Name> (<ad-in> ,<da-out> , <kp> ,<ki> ,<kd>)	.setpoint(<value>)

Anmerkungen:
der PID-Regler hat nur eine Eingangs- und eine Ausgangsgröße

- Aktionsteil eines Agenten
  - endlicher Automat mit Zuständen
  - zyklische Bearbeitung der Aktionen des aktiven Zustands
  - jede Aktion durch Bedingung gebunden (guard ⇒)
  - Aktion wird ausgeführt, wenn Bedingung erfüllt
  - Anweisungen einer Aktion:
    - Aufruf eines Auftrags eines Agenten oder Subagenten
    - Zustandsübergang (**newstate**)
    - arithmetische und logische Ausdrücke
  - Spezielle Bedingung: **once**
  - Spezielle Zustände: **initial** , **shutdown**
- Prototypische Anwendung: Doppelaufzug







- Beispiel Türagent

```
targetdecl mc68hc11 mc12;
conndecl mc12 CAN to mc_elev CAN;
agent door2 target mc12;
decls
DigOut motor(c0, on, off);
DigOut direction(c2, close, open);
DigIn open_key(d1, off, on);
DigIn closed_key(d4, off, on);
DigIn light_barrier(d0, interrupted, ok);
Timer delay;
contracts
open do newstate opening;
states
closed/shutdown:
once ⇒ motor.off;
opening:
once ⇒ {direction.open; motor.on;}
open_key.on ⇒ newstate waiting;
waiting:
once ⇒ {motor.off; delay.restart(5000);}
delay.tout ⇒ newstate closing;
closing/initial:
once ⇒ {direction.close; motor.on;}
closed_key.on
⇒ {elevator1.start; newstate closed;}
light_barrier.interrupted
⇒ {motor.off; newstate opening;}
endagent;
```


- Beipiel für Verhandlung

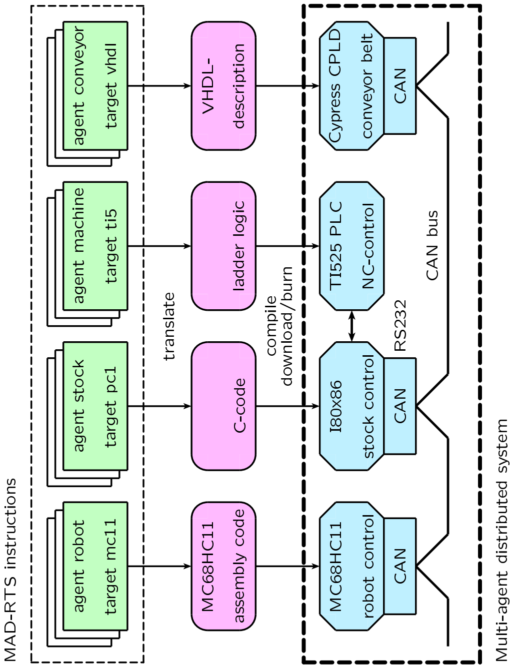
a) Aufzugsagent

```
targetdecl mc68hc11 mc_elev;
agent elevator1 target mc1;
decls
varinteger aktfloor;
arraybool(4) stop;
contracts
get (integer floor)
cost ((instate busy)*100+
abs(floor-aktfloor)*20);
do stop(floor).set(true);
states
...
busy:
...
endagent;
```

b) Auftrag, um einen der beiden Aufzüge nach Stockwerk 2 zu holen

```
elevator1 | elevator2.get(2)
```


- Implementierung
  - ⋆ Übersetzer
    - ∘ Syntaxbeschreibung in YACC
    - ∘ Lexikalische Analyse in FLEX
    - ∘ Syntaktische Analyse in BISON
    - ∘ Erzeugung einer Zwischensprache
    - ∘ Implementiert in C++
    - ∘ auf PC unter MSDOS/Windows
  - ⋆ Code Generatoren und Laufzeitsystem für
    - ∘ Intel 80x86
    - ∘ Microcontroller MC68HC11
    - ∘ Speicherprogrammierbare Steuerung (SPS) TI500 Serie
    - ∘ Programmable Logic Device (PLD) über VHDL
  - ⋆ Kommunikation
    - ∘ Direkter Aufruf der Aufträge von Agenten auf dem gleichen Targetrechner
    - ∘ Automatische Übermittlung der Aufträge über CAN-Bus bzw. serielle Schnittstelle



- Vorteile von MAD-RTS
  - ⋆ problemorientierte Spezifikation
  - ⋆ TopDown Entwurf unabhängig von der Verteilung auf Rechner
  - ⋆ transparente Kommunikation
  - ⋆ deterministische Antwortzeiten
  - ⋆ Separierung von zeitkritischen Modulen in Agenten auf dediziertem Prozessor
  - ⋆ verzögerungsfreie Reaktion durch Hardware-Agenten (PLD)
  - ⋆ nicht zeitkritische Agenten mit Verhandlungsprotokoll
  - ⋆ tragfähiges Konzept von Signalebene bis Verhandlungsebene
  - ⋆ selbstdokumentierende strukturierte Programme

10.13 Synchrone Programmiersprachen: ESTEREL

- G. Berry, Frankreich, ab 1984 entwickelt
- andere Sprachen
  - ⋆ Lustre
  - ⋆ Signal
  - ⋆ Statecharts
- ESTEREL-Modell für reaktive Systeme
  - ⋆ Schnittstellenmodul zur Umgebung
  - ⋆ Kernmodul mit logischer Ablaufsteuerung
  - ⋆ Datenmodul (data handling layer bei Berry)
- ESTEREL ist Programm-Generator zur Erzeugung Kernmodul
- Schnittstellen- und Datenmoduln in konventionellen Programmiersprachen geschrieben
- Ziel: deterministische Abläufe

- Arten der Synchronität
  - ⋆ asynchrone Systeme
  - ⋆ zeit-synchron
    - gemeinsamer Zeittakt der Komponenten
    - alle Komp. machen gleichzeitig einen Schritt (im Takt)
  - ⋆ EA-synchron
    - Ankunft des Eingangssignals und Senden des Ausgangssignals als Ergebnis im gleichen Takt ⇒ als gleichzeitig zu betrachten
  - ⋆ nachrichten-synchron
    - wie Ada
    - Nachrichtenübertragung erst, wenn Sender sendebereit und Empfänger empfangsbereit
    - der Sender oder der Empfänger wartet bis der andere bereit
- ESTEREL ist
  - ⋆ zeit-synchron
  - ⋆ EA-synchron

- synchrone Systeme in ESTEREL
  - ⋆ alle Reaktionen der Prozesse erfolgen sofort ohne Zeitverzug
  - ⋆ alle Aktionen haben die Bearbeitungszeit 0 z.B. Befehle, Nachrichtenaustausch, Abläufe in Prozessen
  - ⋆ ⇒ alle Prozesse haben dasselbe Bild über den Zustand des Systems
  - ⋆ ⇒ der Zustand der Umgebung ändert sich während einer Aktion nicht
  - ⋆ Befehle benötigen nur Zeit, wenn dies explizit angegeben wird, z.B.

    ```
    await 30 MILLISECOND
    ```

    (wartezustand dauert exakt 30 ms)
- die Anweisung

```
every 1000 MILLISECOND do
  emit SECOND
end
```

ergibt in einem synchronen System exakt alle 1000 ms das Signal SECOND, aber nicht in einem asynchronen System

- Anmerkung: MILLISECOND ist ein Signal, das mit einem Uhrentakt verbunden ist, eine entsprechende Anweisung kann mit jedem Signal geschrieben werden, z.B.

```
every 60 MILLIMETER do
  emit CHECK
end
```

- Wie realistisch ist die Annahme?
  - ⋆ Bearbeitungsschritt = alle Operationen, die beim Zustandsübergang ausgeführt werden müssen (Reaktion auf das Eingangssignal)
  - ⋆ synchrones System entspricht Ablauf in einem sequentiellen Automaten
  - ⋆ ⇒ keine Unterbrechung von Bearbeitungsschritten
  - ⋆ ⇒ nicht alle realen Systeme so modellierbar
  - ⋆ ⇒ nur für bestimmte reaktive Systeme anwendbar, nämlich solche bei denen
    - die Arbeit des Automaten getaktet werden kann und Takt länger als längster Bearbeitungsschritt (Takt bestimmt Reaktionsfähigkeit)
    - sichergestellt ist, daß Abstand zwischen externen Signalen, die zum Zustandsübergang führen, wesentlich größer als jeder Berarbeitungsschritt (z.B. Stoppuhr, Verkaufsautomat)

- ESTEREL-Sprachelemente
  - ⋆ normale imperative Standard-Anweisungen
    - Zuweisung
    - Erzeugung Signale
    - Bedingungen
    - Schleifen
    - Exception-Behandlung
    - parallele Abläufe
  - ⋆ (temporale) Anweisungen mit Signalen
    - await <event> do
    - do <anweisungen> watching <event>
    - loop <anweisungen> each <event>
  - ⋆ Beispiel

    ```
    do
      every STEP do
        emit JUMP
      end
    watching 100 METER
    ```


- Signal
  - ⋆ broadcast zeitlos an alle Prozesse und die Umgebung
  - ⋆ Signal enthält auch noch Wert
  - ⋆ alle gleichbenannten Signale, die zur gleichen Zeit gesendet werden, werden in ein Signal verschmolzen
  - ⋆ Kombination der Werte der Einzelsignale spezifizierbar, z.B. Liste, Addition usw.
- parallele Anweisungen
  - ⋆ sequentielle Ausführung der folgenden Anweisung durch ; gekennzeichnet (sequencing operator)
  - ⋆ parallele Ausführung mit der folgenden Anweisung durch || gekennzeichnet (parallel operator)
  - ⋆ der Operator ; hat Vorrang vor ||
- formale Semantik vorhanden
- Übersetzung ESTEREL-Programme in sequentielle Automaten, die dann Kernmodul bilden

- Beispiel nach Berry

```
module P:
input I;
output O;
signal S in
  emit O;
  loop
      await I;
      await I do emit S end
    ||
      await S do emit O end
  end
end.
```

daraus wird folgender Automat erzeugt (das interne Signal S verschwindet)

```
state 0:
  ε → output O; goto 1
  I → output O; goto 1
state 1:
  ε → goto 1
  I → goto 2
state 2:
  ε → goto 2
  I → output O; goto 1
```


- Beispiel Stoppuhr

```
module STOP_WATCH:
  input START_STOP, HS;
  output TIME(integer);
  var AKT := 0 : integer in
    loop       % stopped state
      emit TIME(AKT);
      await START_STOP;
      do     % running state
        every HS do
          AKT := AKT + 1;
          emit TIME(AKT);
        end
      upto START_STOP
    end
end.
```

- Erweiterung der Stoppuhr durch reset

```
module STOP_WATCH_RESET:
  input START_STOP, HS, RESET;
  output TIME(integer);
  loop
    run STOP_WATCH
  each RESET
end.
```


11 Zuverlässigkeit und Sicherheit

- Echtzeitsysteme stellen erhöhte Anforderungen an Zuverlässigkeit und Sicherheit zur Vermeidung von wirtschaftlichem Schaden und Personengefährdung
- Neben der Perfektionierung von Maßnahmen zur Fehlervermeidung werden Fehlertoleranzeigenschaften gefordert

11.1 Definitionen

- Zuverlässigkeit
  (dependability, aber auch reliability)
  Fähigkeit eines technischen Systems, die vorgesehenen Aufgaben unter bestimmten Betriebsbedingungen während der vorgegebenen Zeitspanne zu erfüllen

- Sicherheit
  (safety)
  Fähigkeit eines technischen Systens, Gefährdung und Schaden in bestimmten Betriebszuständen zu vermeiden
- Fehler (Defekt, error, fault)
  Nichterfüllung vorgegebener Anforderungen (Fehler als Zustand)
- Ausfall (failure)
  Ereignis, das Übergang vom fehlerfreien in fehlerbehafteten Zustand bewirkt
- Fehler transient, permanent
- Zuverlässigkeit durch Fehler, dh. Fehlzustände (errors, faults) oder Funktionsausfälle (faults, failures) beeinträchtigt
- Überlebenswahrscheinlichkeit R(t)
  (reliability)
  als Maßzahl ist die Wahrscheinlichkeit, daß die zunächst funktionsfähige Einheit im Zeitraum 0 bis t ohne Reparaturen funktionsfähig bleibt
  Komplement: Ausfallwahrscheinlichkeit

- MTBF (mean time between failures)
  mittlerer Ausfallabstand = $\frac{\Sigma Betriebszeiten}{\Sigma Ausfaelle}$
  Ausfallrate = $\frac{1}{MTBF}$
- MTTR (mean time to repair)
  mittlere Instandsetzungsdauer
  Instandsetzungsrate = $\frac{1}{MTTR}$
- Verfügbarkeit (availability)
  Wahrscheinlichkeit, daß eine reparierbare Einheit zum Zeitpunkt t funktionsfähig ist
  p = $\frac{MTBF}{MTBF+MTTR}$
- Erhöhung der Zuverlässigkeit
  Maßnahmen gegen Fehler aus dem System selbst durch Erhöhung der Verfügbarkeit
- Erhöhung der Sicherheit
  Neben Erhöhung der Zuverlässigkeit noch Maßnahmen gegen gefährliche Auswirkungen; Verminderung des Restrisikos; Prüfdokumente als Nachweis gefordert
- Fehlertoleranz
  Fähigkeit eines Systems, trotz Auftretens einer begrenzten Anzahl von Fehlern seine Funktionen zu erfüllen; wichtig für hochzuverlässige und wartungsarme (–freie) Systeme.

- Verfügbarkeit vs. Sicherheit:
  - ⋆ Auto, das niemals fährt, ist sicher (es geht keine Gefahr von ihm aus), es ist aber nicht brauchbar
  - ⋆ Auto, das immer fahren kann (auch wenn alle Bremsen defekt), hat höchste Verfügbarkeit, ist aber nicht sicher
- Sicherheit muß folgende Fehlerquellen berücksichtigen
  - ⋆ Spezifikationsfehler
  - ⋆ Implementierungsfehler
  - ⋆ Ausfälle Hardware, Geräte usw.
  - ⋆ Bedienungsfehler
  - ⋆ unerwartete Einflüße oder Störungen durch die Umgebung

- Bausteine
  - ⋆ Erkennen von Fehlern, z.B. durch Plausibilität, Redundanz
  - ⋆ Maskieren von Fehlern durch Redundanz
  - ⋆ Begrenzung der Ausbreitung von Fehlern
  - ⋆ Behandlung von Fehlern
  - ⋆ fail-save-Techniken (HW und SW), z.B.
    - ◦ Ventil schließt automatisch bei Stromausfall oder Ausbleiben von Anweisungen des Prozeßrechners
    - ◦ Relais fällt durch Federkraft ab
    - ◦ Überdrucksicherheitsventil

11.2 Fehlertolerante Systeme

(fehlertolerierende Systeme, fault tolerant systems)

- trotz beliebiger (oder nur begrenzter) Anzahl von gleichzeitig auftretenden Fehlern wird (evtl. reduzierte) Funktion erfüllt
- Fehlertoleranzverfahren gibt an, welche und wieviele gleichzeitig auftretende Fehler durch entsprechende Maßnahmen kompensiert werden können
- Fehlermodell beschreibt
  - ⋆ Menge der Komponenten und Funktionen des Systems
  - ⋆ Menge der Fehlermöglichkeiten
  - ⋆ beschreibt Menge der tolerierten Fehler aus der Menge der Fehlermöglichkeiten
  - ⋆ von Fehler betroffene Komponenten und deren Beeinträchtigung
  - ⋆ Problem: An alle Fehler gedacht?
  - ⋆ Problem: Fehler der Umgebung auch einbezogen?

  - ⋆ Problem: Wie wahrscheinlich sind die Fehler?
  - ⋆ Problem: Soll man den Fehler berücksichtigen oder nicht?
  - ⋆ je nach Parameter: Optimistische oder pessimistische Modelle
- Arten von Fehlermodellen
  - ⋆ Komponenten-Liste
    - ◦ Annahme: jeder Fehler einer Komponente ergibt Systemfehler
    - ◦ also Wahrscheinlichkeit, daß System intakt, gleich Wahrscheinlichkeit, daß keine Komponente ausgefallen
    - ◦ System toleriert also keine Fehler
  - ⋆ Kombinatorische Modelle
    - ◦ Beispiel: Fehlerbaum
    - ◦ Attributierung mit der Wahrscheinlichkeit des Zustandes
    - ◦ Erweiterung: explizite Formel zur Verknüpfung der Folgeknoten, um z.B. "beliebige 2 von 3 Knoten dürfen ausfallen", einfach darstellen zu können
  - ⋆ Automaten, Markoff-Modelle

    - ◦ dynamisches Modell
    - ◦ enthält u.a. auch Reparaturen, Rekonfigurationen
    - ◦ relevante Systemzustände werden mit ihren Übergängen modelliert
    - ◦ Systemzustände sind gegeben durch die Zustände der Einzelkomponenten, die ausfallen können
    - ◦ der Systemzustand enthält auch die Angabe, ob das Gesamtsystem arbeitsfähig ist oder nicht
    - ◦ Übergangswahrscheinlichkeiten festzulegen
    - ◦ Zeitdauern, z.B. für Reparaturen, festzulegen
    - ◦ Auswertung numerisch oder simulativ
    - ◦ Ergebnis z.B. Wahrscheinlichkeit, daß System in einem arbeitsfähigen Zustand ist

- Grundanforderungen beim Auftreten von Fehlern
  - ⋆ "Überleben" beim Ausfall der 1. Komponente
  - ⋆ stoßfreier Übergang (evtl. mit Leistungsreduzierung)
  - ⋆ automatisches Neukonfigurieren
- Fehlertolerante Systeme sind normalerweise redundante Systeme, d.h. zusätzliche Komponenten bzw. Funktionen eingesetzt, dh. mehr technische Mittel als für das funktionierende System allein benötigt würden
- Arten der Redundanz

(1) Strukturelle Redundanz
  - ⋆ zusätzliche, im fehlerfreien Fall entbehrliche (Hardware-)Komponenten

(2) Funktionelle Redundanz
  - ⋆ Umschalten auf gleichartige Ersatzkomponente
  - ⋆ Diversität: verschieden implementierte Komponenten oder Funktionen

- ⋆ Zusatzinformationen zur Wiederherstellung der Konsistenz

(3) Zeitredundanz
- ⋆ Wiederholung
- ⋆ Schwache Auslastung erlaubt zusätzliche Aktionen in einem funktionell redundanten System

(4) statische Redundanz: die zusätzliche Komponente ist dauernd aktiv

(5) dynamische Redundanz: die redundanten Komponenten werden erst im Fehlerfall aktiviert (auch ggf. Abwerfen von Last oder unwichtigeren Aufgaben)

- Umfang der notwendigen Fehlertoleranz anwendungsabhängig
- beliebig hohe Fehlertoleranz = beliebig hohe Kosten
- Zuverlässigkeits- und Sicherheitsanalyse mit Kosten-Nutzen-Abschätzung erforderlich

11.3 Prüfung und Qualitätssicherung von Steuerungssoftware

- Kategorien von Normen
  - ⋆ Begriffsnormen
  - ⋆ Produktnormen
  - ⋆ Prozeßnormen (Verfahren und Vorgehensweisen z.B. bei der Erstellung von Software)
  - ⋆ Verfahrensnormen (Regeln für die Überprüfung der Einhaltung einer bestimmten Norm)
- Normen und Richtlinien zur Qualitätssicherung von Steuerungssoftware
  - ⋆ Begriffsnormen
    - ◦ Software-Qualitätsmerkmale DIN 6627
    - ◦ Qualitätssicherung DIN ISO 8402
    - ◦ Begriffe der Sicherheitstechnik DIN VDE 31000-2
  - ⋆ Produktnormen
    - ◦ Software-Produkt DIN ISO IEC 12119
    - ◦ Benutzerdokumentation DIN 66230

- ⋆ Prozeßnormen für Software-Lebenszyklus
  - ◦ Sicherheitssoftware IEC CD 1508-1 bis IEC CD 1508-7, insbesondere IEC CD 1508-3
  - ◦ Steuerungssoftware DIN V VDE 0801
  - ◦ Kerntechnik DIN IEC 880
  - ◦ Luftfahrt RTCA/DO 178B
- ⋆ Prozeßnormen für Qualitätsmanagement
  - ◦ Entwicklung DIN EN ISO 9001
    gilt universell für materielle Güter, Dienstleistungen, Software
    enthält organisatorische, administrative und technische Bestandteile eines Qualitätsmanagements
    nur Vorschrift, wann was durchzuführen, aber nicht wie (Werkzeuge, Methoden)
  - ◦ Software DIN ISO 9000-3
- ⋆ Verfahrensnormen
  - ◦ Audits DIN ISO 10011

11.4 Fehlererkennung und –behandlung

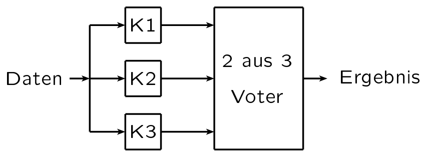
- Erkennung Ziel: Offenlegung und Lokalisierung mit kurzer Latenzzeit
- Fehlererkennung
  - ⋆ Plausibilitätskontrollen
  - ⋆ Redundanz
- Fehlerausgrenzung
  - ⋆ fehlerhafte Teile werden entfernt
  - ⋆ problematisch ist Zusammenhang mit anderen Komponenten
  - ⋆ Rekonfigurierung ersetzt Funktionen durch neue fehlerfreie Komponenten
- Fehlerkorrektur
  - ⋆ Komponente wird in einen fehlerfreien Zustand überführt
  - ⋆ Vorwärtsbehebung:
    Erstellen eines konsistenten Zustands aus der Momentansituation, z.B. durch Ausnahmebehandlung

- ⋆ Rückwärtsbehebung:
  Rücksetzen ("roll back") auf einen fehlerfreien Zustand der Vergangenheit (Rücksetzpunkte, checkpoints, recovery points). Problem: Domino–Effekt mit mehrfachem Rücksetzen
- Fehlerkompensierung
  - ⋆ verhindert die Ausbreitung der negativen Fehlerwirkung
  - ⋆ Fehlzustand bleibt und muß später behoben werden
- Begrenzung der Fehlerausbreitung
  - ⋆ Fehler soll sich nicht auf andere Komponenten ausbreiten, bevor er erkannt wird
  - ⋆ In einem hierarchischen Software–Schichtenmodell sollten Fehler, die in einer Schicht auftreten, nicht in höhere Schichten aufsteigen

11.5 Hardware–Redundanz

- zusätzliche parallele Komponenten
- Statische Redundanz
  - ⋆ mitlaufende Reserve (hot stand-by)
    - ◦ redundante, aktive (mitlaufende) Komponenten
    - ◦ bei Ausfall wird sofort umgeschaltet
    - ◦ Beispiel: Doppelrechnersystem
      2 unabhängige Rechner erfassen mit unabhängigen Geräten die gleichen Daten, jedoch steuert nur einer. Bei einem Fehler wird die Steuerung sofort auf den 2. Rechner geschaltet.
  - ⋆ Fehlermaskierende Struktur
    - ◦ N Komponenten arbeiten parallel (N=2n+1, n=1,2)
    - ◦ mit 1 Voter
    - ◦ selten Voter wieder redundant
    - ◦ Ergebnisse werden votiert (Mehrheitsergebnis gilt)

- ◦ Beispiel:TMR Triple Modular Redundancy



K1, K2, K3 sind Hardwarekomponenten, die ausfallen können
der Voter ist gegenüber den Komponenten Ki hoch zuverlässig

- ⋆ Vorteil: Sofortiges Weiterarbeiten bei Ausfall
- ⋆ Probleme: Kosten, synchrones Arbeiten der Systeme

- Dynamische Redundanz (cold standby, kalte Reserve)
  - ⋆ erhöht die Verfügbarkeit
  - ⋆ passive Reservekomponenten werden nach Ausfall zugeschaltet
  - ⋆ ersetzen nach einiger Zeit ausgefallene Komponente
  - ⋆ Vorteil: kostengünstig
  - ⋆ Nachteil: Trägheit, Reduktion der Gesamtleistung im Fehlerfall
- Hybride Redundanz
  - ⋆ z.B. statisch redundantes 2 aus 3–System
  - ⋆ und 2 passive Komponenten mit Rekonfigurations–Steuergerät

11.6 Zuverlässige Software

- Software enthält inhärent Fehler (kein fail safe–Übergang):
  - (1) Spezifikationsfehler
    die Aufgabe ist unvollständig, nicht eindeutig oder falsch beschrieben
  - (2) Entwurfsfehler
    der Entwurf erfüllt nicht die Spezifikation bzw. ist in sich widersprüchlich oder falsch
  - (3) Implementierungsfehler
    Abweichung vom Entwurf, logische Fehler
  - (4) Werkzeugfehler
    Übersetzer, Binder, Betriebssystem oder die Hardware enthalten Fehler
  - (5) Betriebsfehler
    Fehlbedienung, Fehler in der Dokumentation, Viren
- Produktion zuverlässiger Software
  - ⋆ straffstes Projektmanagement mit gemeinsamer Team–Datenbank
  - ⋆ soweit möglich formale Spezifikation (Problem u.a. Beschreibung Zeitverhalten)

- ⋆ formale Beschreibungstechnik für Teile
- ⋆ umfangreiche Dokumentation
- ⋆ sauberer, robuster Entwurf, u.a. mit Fehlererkennung, Fehlerlokalisierung, Verhinderung der Fehlerausbreitung
- ⋆ Methodiken zur Programmanalyse
- ⋆ Testmethodik
- ⋆ Code-Review
- ⋆ Entwicklung ggf. nach ISO 9000 mit SW-Zertifizierung, TÜV-Abnahme
- ⋆ Einsatz einer Spezifikationssprache, z.B. RT-UML

11.7 Software-Redundanz

(1) Benutzung redundanter Hardware

- Rechner verschiedener Hersteller
- gleiche Algorithmen auf verschiedenen Rechnern compiliert
- parallele Ausführung der gleichen Algorithmen auf verschiedenen Rechnern
- Votieren der Ergebnisse
- Beispiel: Projekt FUTURE

(2) Voll diversitäre Software

- Realisierung in unterschiedlichen Firmen
- Realisierung auf unterschiedlichen Rechnern
- unterschiedliche, unabhängige Teams (mindestens 2)
- unterschiedliche Spezifikation
- unterschiedliche Algorithmen
- unterschiedliche Entwicklungen, Realisierungen
- paralleler Ablauf auf verschiedenen Rechnern
- Akzeptanzkriterium oder Votieren der Ergebnisse

- Alternativ: Vergleich und Abstimmung bei den Entwicklungsschritten und Einsatz nur eines Systems

(3) Partiell diversitäre Software

- Einschränkungen in obiger Liste
- z.B. nur verschiedene Algorithmen: Ergebnis eines Algorithmus wird an Hand Akzeptanzkriterium geprüft; bei Versagen erfolgt Wiederholung mit anderem Algorithmus
- Beispiel: Schreiben eines Datenblocks

```
ENSURE <Auftragswarteschlange konsistent>
     /* Akzeptanzbedingung */
BY
     /* 1. Algorithmus */
  Auftrag optimal in Warteschlange einordnen;
ELSE BY
     /* 2. Algorithmus */
  direkt in vorgegebenen Bereich schreiben;
ELSE
  Fehlermeldung Auftrag nicht ausgeführt;
```


- Erfahrungsbeispiel "PODS" (projekt on diverse software)
  - ⋆ P. Bishop et al. in IEEE Trans. on SW–Engineering, Vol SE-12, S. 929-940, 1986
  - ⋆ Aufgabe: kritisches Steuerprogramm beim Hochfahren eines Generators
  - ⋆ Ausführung durch drei Teams in England, Norwegen, Finnland
  - ⋆ beobachtete Phasen
    - ◦ 1 Kundenspezifikation
    - ◦ 3 unabhängige Herstellerspezifikationen
    - ◦ 3 Entwürfe
    - ◦ 3 Codierungen
    - ◦ 3 Abnahmen (2472 Tests)
    - ◦ Votieren der 3 Programme (665 288 Tests)
  - ⋆ Spezifikationsmethode:

    Team 1: informell
    Team 2: Spez. Sprache
    Team 3: Spez. Sprache

- ⋆ verwendete Sprache:

  Team 1: FORTRAN
  Team 2: Assembler
  Team 3: FORTRAN

- ⋆ Algorithmus:

  Team 1: Polynom–Algorithmus
  Team 2: Tabellen
  Team 3: Tabellen

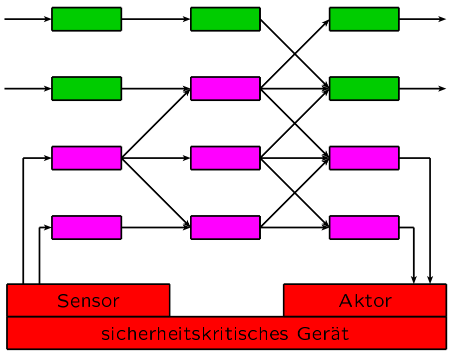
- ⋆ Ergebnis des Votierens:
  - ◦ 7 Fehler entdeckt
  - ◦ davon 6 aus Kundenspezifikation (2 falsche Angaben, 4 mehrdeutige Angaben)
  - ◦ davon 3 in gefährlichen Situationen
  - ◦ davon 2 Fehler doppelt, d.h. sie hätten sich durchgesetzt
  - ◦ kein Fehler bei der Codierung

- ⋆ Ergebnis der Untersuchung
  - ◦ ermöglicht evtl. Erkennung mehrdeutiger Kundenspezifikationen
  - ◦ erlaubt viele Tests mit Datensätzen aus Zufallszahlen im zulässigen Zahlbereich ohne das Ergebnis im voraus kennen zu müssen
  - ◦ wenig Unterschied von informeller und formaler Spezifikation (vielleicht war Projekt zu klein)
  - ◦ nach Abnahme keine Fehler in der Implementierung
  - ◦ Anzahl Fehler je Programmzeile unabhängig von der Programmiersprache
  - ◦ Laufzeit: Assembler 3 mal schneller als FORTRAN
  - ◦ kein Fehler im Compiler entdeckt
  - ◦ Restfehler aus ungenauer Kundenspezifikation
  - ◦ Kosten: etwa 2,25-fache von einem Projekt (366 Std. Kundenspezifikation, 1892 Std. für 3 Hersteller)

11.8 Maßnahmen zur Sicherheit

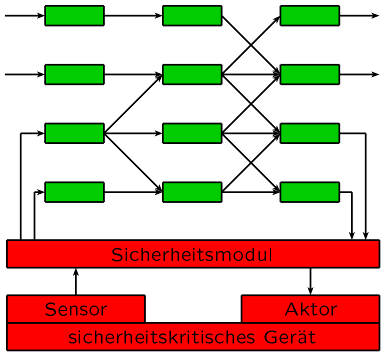
- Sicherheitsnachweise durch Dokumente und Prüfungen für TÜV–Abnahme erforderlich
- ausreichende Redundanz zur Fehlertoleranz oder Fehlererkennung
- möglichst einfache, stark modularisierte Systeme, da i.d.R.
  - ⋆ ein Versagen jeder Komponente das Versagen des Gesamtsystems bedeuten kann
  - ⋆ die Entwicklung unübersichtlicher
  - ⋆ Anzahl der Schnittstellen (Schwachstellen und Fehlerquellen) steigt
- Trennung von Sicherheit und Funktion als Entwurfsprinzip (sicherheitsorientierte Modularisierung)
  - ⋆ neben üblicher Modularisierung
  - ⋆ sicherheitsrelevante Funktionalität von restlicher Funktionalität getrennt und in wenige sichere Module gekapselt

⋆ Negativbeispiel (nach S. Montenegro)



- für Sicherheit kritischer Modul
- unkritischer Modul
- kritisch gewordener unkritischer Modul

⋆ empfohlene sicherheitsorientierte Modularisierung
- ∘ Sicherheitsmodul hat vollständige Kontrolle über Anlage
- ∘ Funktionsmodul kann Befehle nur über Sicherheitsmodul abgeben
- ∘ Sicherheitsmodul prüft Sicherheit jedes Befehls bevor es ihn weitergibt



- für Sicherheit kritischer Modul
- unkritischer Modul

- "fail–safe"–Übergänge in ungefährliche Zustände vorsehen (z.B. sicheres schadenfreies Abschalten)
  - ⋆ Hardware–Einrichtungen für fail–safe–Übergänge, die nicht von der Software gesteuert werden (z.B. Grenzwertüberschreitung durch Schalter verhindern)
  - ⋆ benutzen von physikalischen Eigenschaften zum Übergang in einen sicheren Zustand. Beispiele: Relais fällt im Fehlerfall auf Ausschalten, Eisenbahnsignal fällt auf Halt,
  - ⋆ Senden aktiver Signale im korrekten Ablauf (z.B. periodisch "Weitermachen" an Maschine); beim Ausbleiben eines solchen Signals wird abgeschaltet

- Probleme bei Fehlermeldungen
  - ⋆ man verläßt sich blind auf das Frühwarnsystem und wird nachlässig ⇒ Unfall, wenn gefährliche Situation nicht gemeldet wird, z.B. wegen technischen Fehlers (was zu erwarten ist)
  - ⋆ Frühwarnsystem zu empfindlich, meldet Gefahren, die keine sind ⇒ Warnungen werden ignoriert ⇒ Unfall, wenn Fehlermeldung berechtigt war
- Sehr wichtig: Gestaltung der Mensch-Maschine-Schnittstelle
  - ⋆ benutzerfreundliche Bedienung, jedoch Prüfungen zur Verhinderung inkorrekter Eingaben
  - ⋆ am Leitstand geeignete ermüdungsarme Darstellungen für den menschlichen Überwacher
  - ⋆ vollständige, verständliche und begründete Alarmmeldungen
  - ⋆ Alarmmeldungen können nicht zu schnell beseitigt werden, damit sorgfältig reagiert wird

⋆ Bedienbarkeit der Anlage im Normal- und Notbetrieb sicherstellen

⋆ Negativbeispiel (Montenegro)
- ∘ Notfall; aber kein großes Problem: Um Anlage in einen sicheren Zustand zu bringen, nur einige kritische Werte abzulesen und entsprechend gegenzusteuern
- ∘ kritisch erst durch gutgemeinte Umstände:
  - ▷ Sirene jault in voller Lautstärke
  - ▷ rote Lichter blinken
  - ▷ Leute rennen herum und schreien verschiedene Anweisungen
  - ▷ Rechner ist in Notbetrieb und erlaubt den üblichen Zugriff auf die kritischen Werte nicht mehr
  - ▷ Wo normalerweise die benötigten Daten erscheinen, rauschen Fehlermeldungen und Warnungen über den Bildschirm. Sie sollten alle diese Meldungen lesen, um wissen zu können, was die jetzige Lage ist.

  - ▷ Situation war im Training nicht erwähnt. Sie müssen zuerst in einem alten Manual nachsehen. Was da beschrieben ist, entspricht nicht dem, was vor Ihren Augen passiert.
  - ▷ Mit Glück schaffen Sie es, die Werte, die Sie brauchen, zu bekommen.
  - ▷ Sollwerte in verschiedenen Fenstern einzugeben, bei denen Texte und Überschriften blinken, damit Sie ihre Wichtigkeit erkennen.
  - ▷ Um die Anlage manuell zu steuern, müssen Sie noch ein Paßwort eingeben, das vor kurzem geändert wurde, ohne daß Sie etwas davon wußten.
  - ▷ Nun müssen Sie nur noch mit zitternder Hand die Maus pixelgenau positionieren, um die Sollwerte eingeben zu können.
- ∘ … leider zeitlich nicht mehr geschafft ⇒ menschliches Versagen

Meldung SZ v. 2.2.1999

Qualmender Aschenbecher löst 40minütigen Alarm aus

Düsseldorf (dpa) – Ein brennendes Taschentuch hat im Düsseldorfer Flughafen zu einem 40minütigen Fehlalarm geführt. So lange lief im Terminal C eine automatische Evakuierungsdurchsage, weil das Papiertaschentuch in einem Aschenbecher Feuer gefangen hatte. Mitarbeiter beruhigten die Passagiere im Terminal mit Megaphonen. Die Fluggäste hätten die Panne eher humorvoll aufgenommen, berichtete der Sprecher des Flughafens. Panik sei nicht ausgebrochen. „Wir mußten den Rechner der Brandmeldeanlage runterfahren und neu starten.“ Eine Entwarnung durch Lautsprecherdurchsagen sei nicht möglich gewesen, weil die Evakuierungsaufforderung keine anderen Ansagen zuließe. Bei einem Brand im Gebäude des Düsseldorfer Flughafens im April 1996 waren 17 Menschen getötet worden.