Materialintegrierte Sensorische Systeme

Labor mit verteilten Sensornetzwerken

PD Stefan Bosse
Universät Bremen, FB Mathematik & Informatik
WS2018
2020-02-04
sbosse@uni-bremen.de

1. Inhalt

2. Überblick

Die Teilnahme an der Veranstaltung soll Studenten interdisziplinär einen praxisnahen und system-orientierten Zugang für die Modellierung, den Entwurf und die Anwendung von material-eingebetteten oder material-applizierten Sensorischen Systemen bieten, die aufgrund der technischen Realisierung und des Einsatzes spezielle Anforderungen an die Datenverarbeitung stellen und ein Verständnis des Gesamtsystems (inklusive Aspekte der Materialwissenschaften und Technologien) voraussetzen. Diese neuen Sensorischen Materialen finden z. B. in der Robotik (Kognition) oder in der Produktionstechnik für die Material- und Produktüberwachung Anwendung.

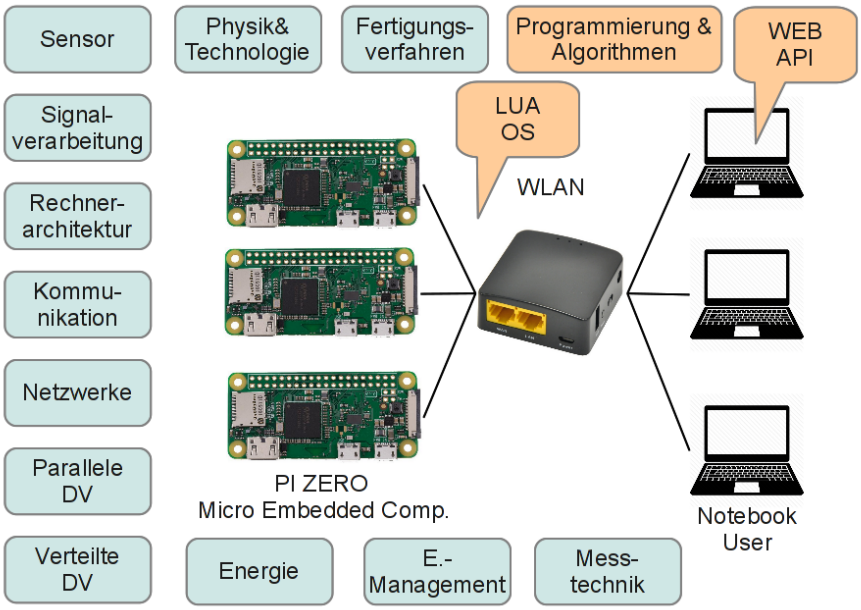
Sensorischen Materialien sind charakterisiert durch eine starke Kopplung von Sensorik, Datenverarbeitung, und Kommunikation und bestehen aus einem Trägerwerkstoff, der u. U. eine mechanisch tra-

gende Struktur darstellen kann, und aus eingebetteten Sensornetzwerken, die neben Sensoren auch Elektronik für die Sensorsignalverarbeitung, Datenverarbeitung, Kommunikation, und Kommunikations- und Energieversorgungsnetzwerke integrieren.

2.1. Grunddaten

VAK	04-326-FT-041
Dozent	PD Stefan Bosse, sbosse@uni-bremen.de
Kategorie	Kurs: Vorlesung und integrierte Übung
Umfang	4 SWS
Art	Master Ergänzung, Systems Engineering / Produktionstechnik
Profil	Fertigungstechnik
ECTS	6
Leistung	Übungsblätter, Mündliche Prüfung
Wann	Jedes Winter Semester, Mi. 14-17 Uhr
Wo	Rober Hooke Str. 5, Raum 1.17
Info	http://sun45.informatik.uni-bremen.de

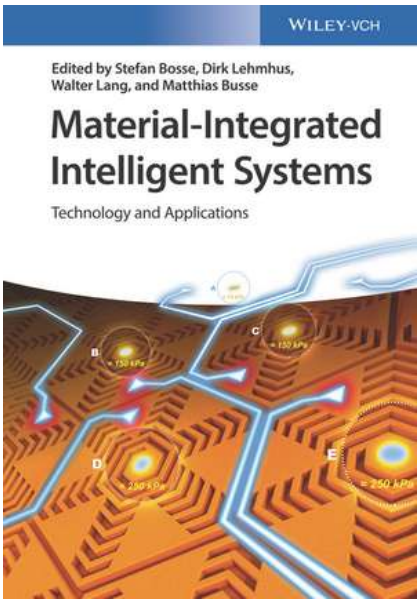
2.2. Inhalte und Themen



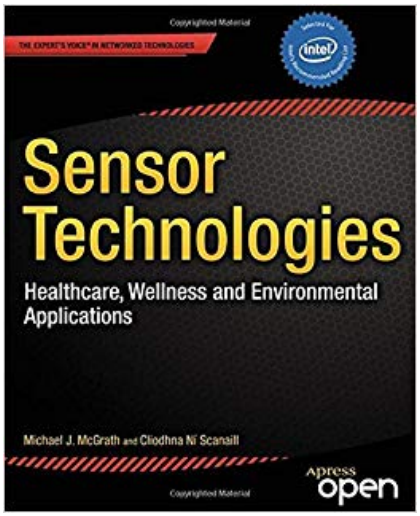
2.3. Literatur

➤ Empfohlene Literatur (neben Vorlesungsskript) zur Vertiefung

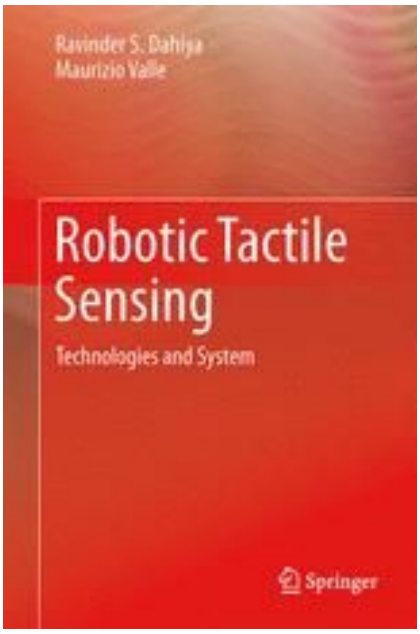
S. Bosse, D. Lehmhus, W. Lang, and M. Busse, Eds., **Material-Integrated Intelligent Systems - Technology and Applications**, 1. ed. Wiley VCH, 2018, p. 685.



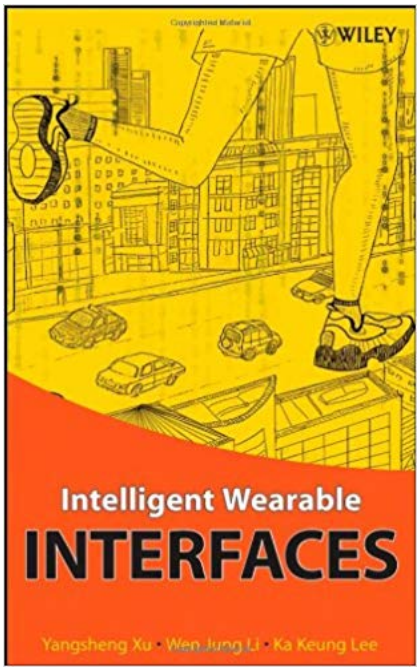
M. J. McGrath and C. N. Scanaill, **Sensor Technologies Healthcare, Wellness, and Environmental Applications**. Apress Open, 2014.



R. Dahiya and M. Valle, **Robotic tactile Sensing**. Springer, 2013.



Y. Xu, W. J. Li, and K. K. C. Lee, **Intelligent Wearable Interfaces**. Wiley, 2008.



2.4. Software

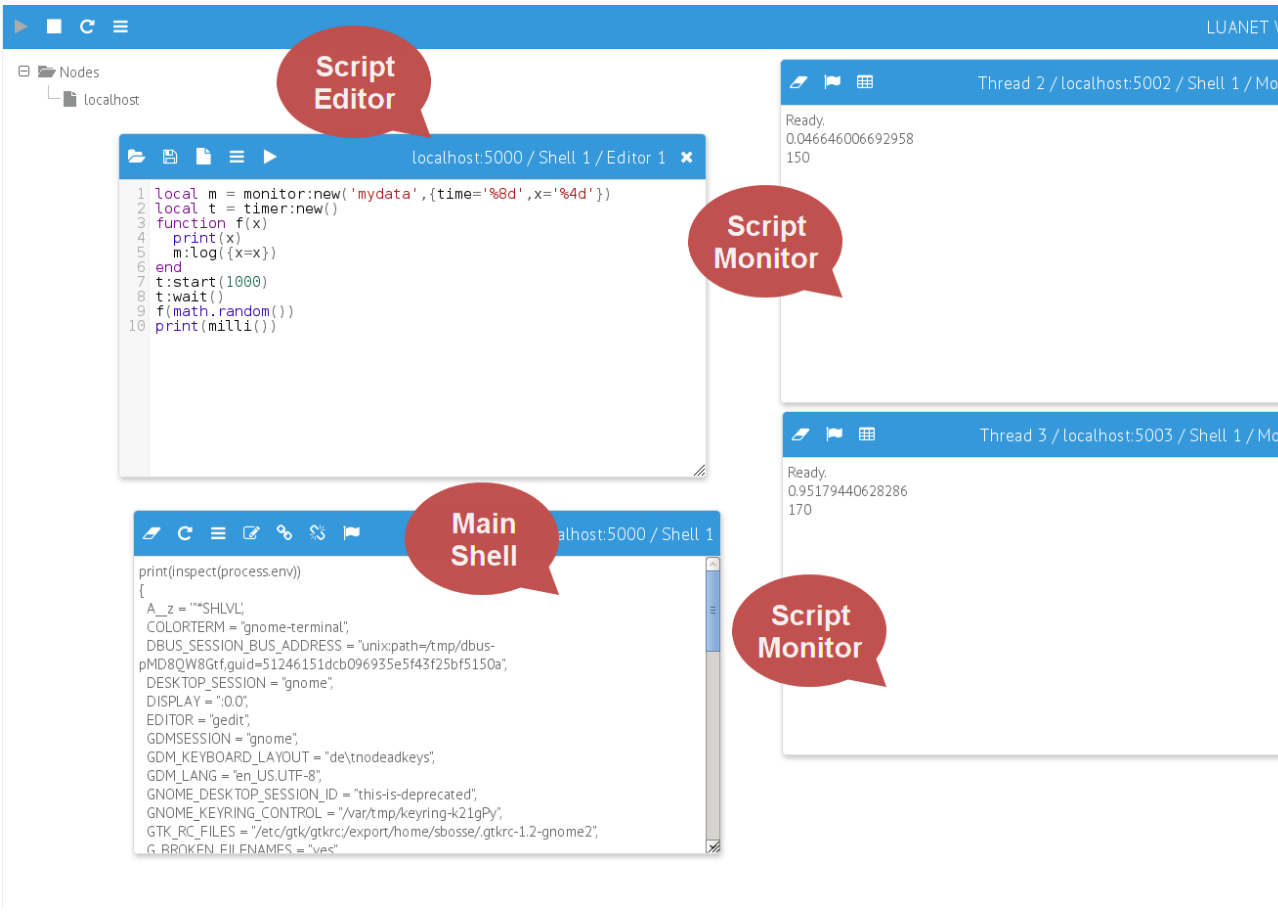
LuaOS

LuaOS: Lua Operating System

- ➤ LUAOS wird auf einem Rechner (Netzwerkknoten) durch eine Lua VM ausgeführt (*lvm*)
  - ❏ Raspberry Pi
  - ❏ Notebook
  - ❏ ..
- ➤ LUAOS:
  - ❏ Ausführung von Skripten in einem Sandkasten
  - ❏ Scheduling und Multithreading
  - ❏ Einfache Sensor und Geräte API

LuaWEB

- ➤ LUAOS bietet einen WEB basierten Zugriff auf Rechnerknoten
  - ❏ Skriptausführung über eine Shell mit Monitoring



2.5. Ziele

1. Die Studenten sollen ein Grundverständnis von material-integrierten intelligenten und sensorischen Systemen erwerben um wesentliche Fragestellungen beim Entwurf und der Nutzung dieser smarten Materialien beantworten zu können
2. Praktisch sollen Kenntnisse und Programmierfertigkeiten von verteilen Sensornetzwerken mit einem Laborprototyp erworben werden
3. Es soll interdisziplinäres Arbeiten und Forschen erlernt werden
4. Verständnis und Anwendung von Sensoren, Technologien, Elektronik, Messtechnik, Signalverarbeitung, Datenverarbeitung, Netzwerken, und Materialintegration in einem ganzheitlichen Ansatz
5. Zentrale Fragen der Materialintegration bezüglich Möglichkeiten und

Randbedingungen sollen beantwortet werden können.

3. Einführung

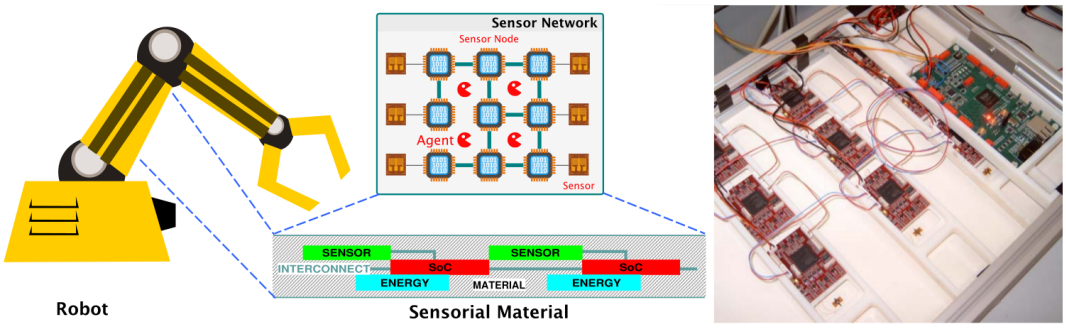
3.1. Materialintegrierte Sensorische Systeme

Material. Technische Strukturen, wie z. B. mechanische Strukturen in robotischen Systemen, e-Textilien, Windradflügeln, Flugzeugstrukturen, usw.

Sensor. Messung einer physikalischen Größe, Wandlung i.A. in elektrische Größe, Intrinsisch (von innen her, durch in der Sache liegende Anreize bedingt), Extrinsisch (von außen her angeregt), z.B. Dehnungssensorik

Sensorisches System. Zusammenschluss von Sensoren, Elektronik, Kommunikation und Energieversorgung in Sensornetzwerken

Sensorisches Material. Integration oder Applikation eines sensorischen Systems in Materialien oder Strukturen



3.2. Anwendungen in Strukturüberwachung

Structural Health Monitoring. Überwachung von technischen Strukturen auf Veränderungen und Beschädigungen (Abweichung vom Initial-Zustand)

Load Monitoring. Aufnahme von Belastungen und Verformungen von mechanischen Strukturen (für SHM oder korrigierende Steuerung von Robotern und Maschinen)

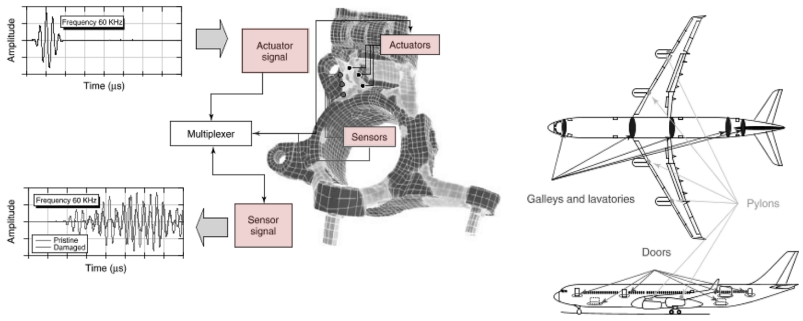


Figure 1. Links: Akustische Strukturüberwachung, Rechts: SHM im Flugzeug [Boller, ESHM, 2009]

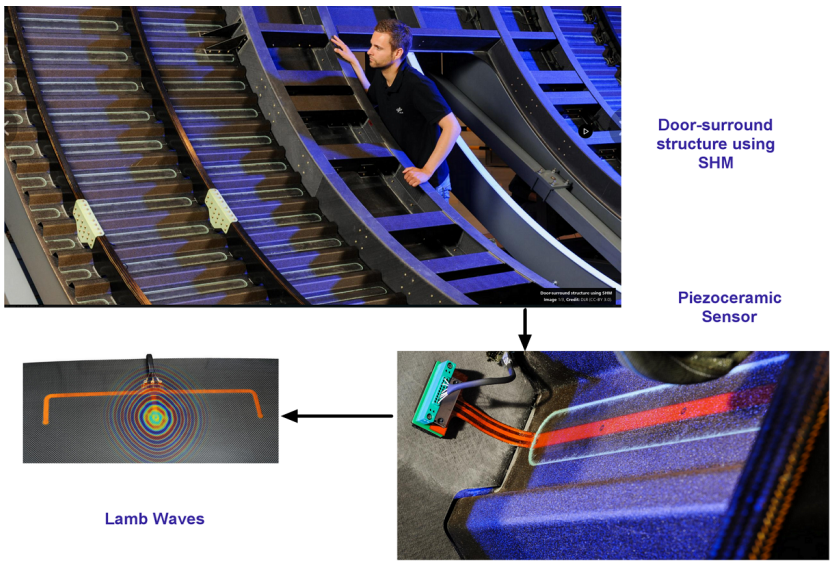


Figure 2. Beispiel: Sensorierung einer Türstruktur eines Flugzeugs [DLR]

3.3. Anwendungen in der Robotik

Sensorische Systeme in der Robotik

Extrinsische Perzeption: Haptik → Künstliche Haut

➤ Technische Umsetzung durch punktuelle Sensoren

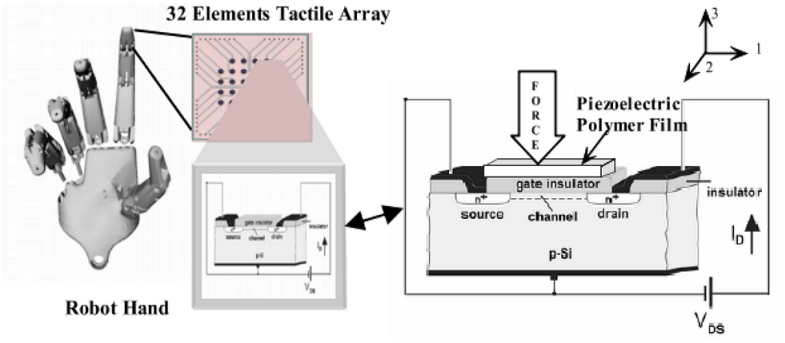


Figure 3. Beispiel Roboterhand [Dahiya et al., 2004]

➤ Technische Umsetzung durch Sensorarrays und resistive Dehnungs- oder Drucksensoren

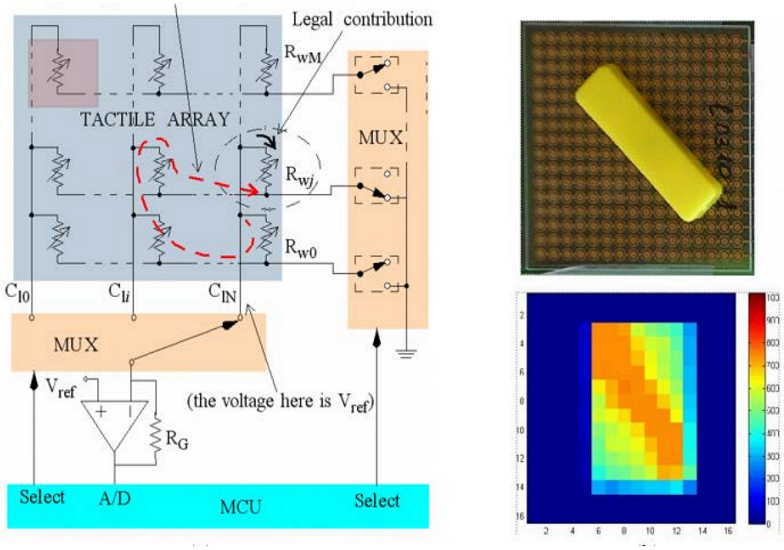


Figure 4. Prinzipieller Aufbau eines Sensorarrays [Vidal Verdu et al., 2011, Sensors]

➤ Technische Umsetzung durch starre Sensorfelder und getrennter Signalverrabeitung (1. Generation)

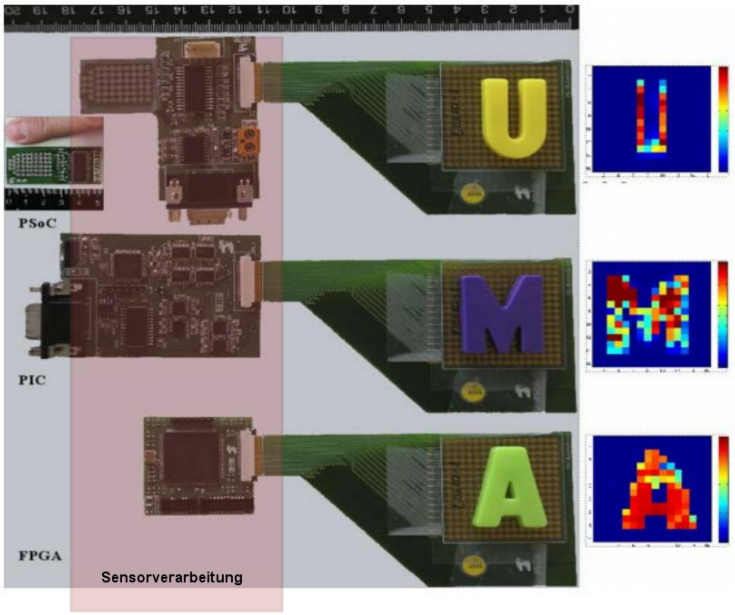


Figure 5. Künstliche Haut, technische Umsetzung und Beispiele für Messungen, 1. Generation [Vidal Verdu et al., 2011, Sensors]

➤ Technische Umsetzung durch flexible Patches und integrierter Elektronik (2. Generation)

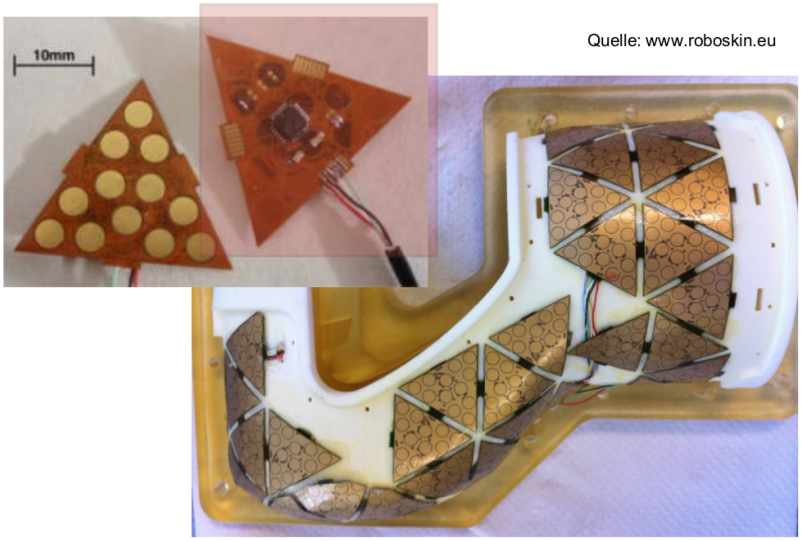


Figure 6. Künstliche Haut, technische Umsetzung 2. Generation [Vidal Verdu et al., 2011, Sensors]

3.4. Anforderungen, Entwurfsmethodiken, Test & Simulation, Normen

- ➤ Welche physikalische Größe soll gemessen werden?
- ➤ Welche Art von Sensor?
- ➤ Platzierung von Sensoren, Material-Applizierung versa Material-Integration?
- ➤ Fertigungstechnologien für Einbettung, el. Verbindung, Sensorik?
- ➤ Datenverarbeitung und Algorithmen - zentralistisch versa dezentral?
- ➤ Topologie Sensornetzwerk und Verbindungstechnologien?
- ➤ Simulation von mech. Strukturen (FEM), Sensornetzwerke, Kommunikation?

Die Entwurfsmethodik ist interdisziplinär und umfasst eine Vielzahl von Domänen

3.5. Komponenten und Domänen

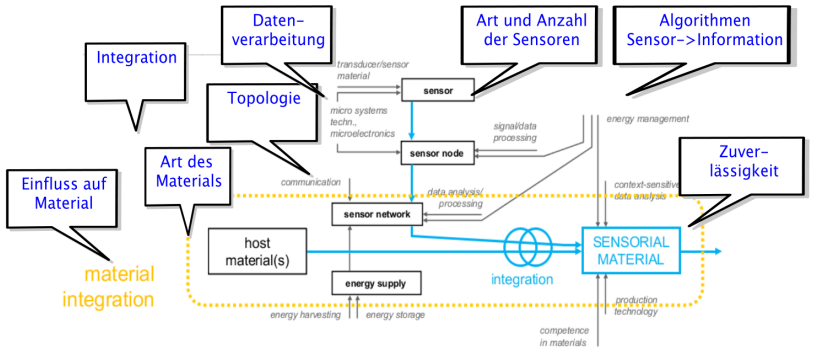


Figure 7. Sensorisches Material - Komponenten und Zusammenhänge [Lehmhus, 2013]

3.6. Grundlagen Messtechnik

1. Mathematische Grundlagen
2. Rauschen - nichts ist exakt → Physik & Statistik
3. Elektrische Größen → Elektrische Messtechnik
4. Optische Größen → Optische Messtechnik
5. Messfehler und Vertrauen → Fehlerklassen → Statistik
6. Einfluss von Messgerät auf Messung

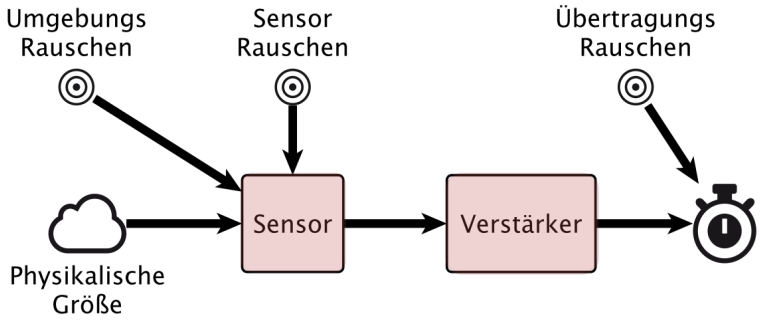


Figure 8. Instrumenten Modell [nach Webster, MISH, 1999]

3.7. Grundlagen Sensoren: Prinzipien, Einteilung, Materialien

1. Sensormodelle
2. Klassifizierung der Messgrößen
3. Klassifizierung der Messmethoden und Technologien
4. Resistive - Elektrogeneratorische - Kapazitive - Optische - Induktive Messwandler
5. Messung einer physikalischen Größe und Wandlung in elektrische Größe
6. Messung einer physikalischen Größe und Wandlung in optische Größe

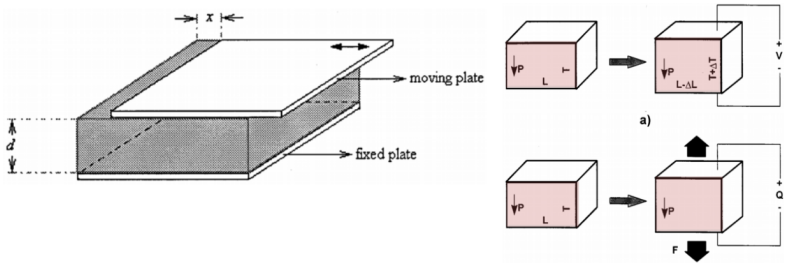


Figure 9. Kapazitiver Verschiebungssensor [Webster, MISH, 1999]

3.8. Grundlagen Elektronik und analog-digitale Signalverarbeitung

1. Operationsverstärker
2. Messbrücken, Verstärker
3. Signalverarbeitung und Filter
4. Analog-Digital und Digital-Analog Wandler

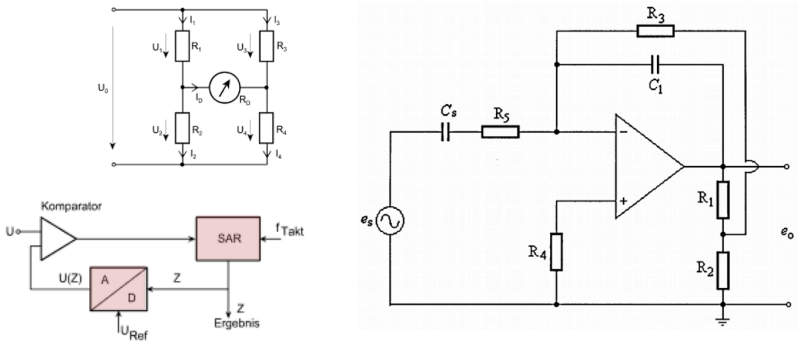


Figure 10. Messbrücke, SAR AD-Wandlung, Ladungsverstärker [Suchaneck, 2007, Webster, MISH, 1999]

3.9. Grundlagen Sensoren: Fertigungsverfahren und Technologien allgemein

1. Druckverfahren - Ätzverfahren - Sputtering- und Abscheidung - Mikrobearbeitung
2. Polymer Technologien
3. Silizium Technologien → Elektronische Schaltungen!
4. Metall Technologien
5. Sensor Materialien - das Material als Sensor

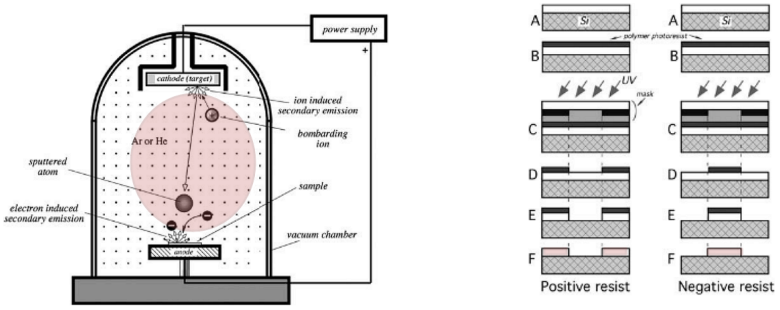


Figure 11. Links: Sputter-Verfahren, Rechts: Ätzverfahren [Fraden, 2010]

3.10. Sensornetzwerke: Grundlagen, Metriken, Entwicklung

1. Paradigmenwechsel: Zentrale Datenverarbeitung → Verteilte Datenverarbeitung → Robustheit
2. Komponenten eines Sensornetzwerks, Netzwerktopologien
3. Maßzahlen für verteilte und parallele Datenverarbeitung - Limitierungen

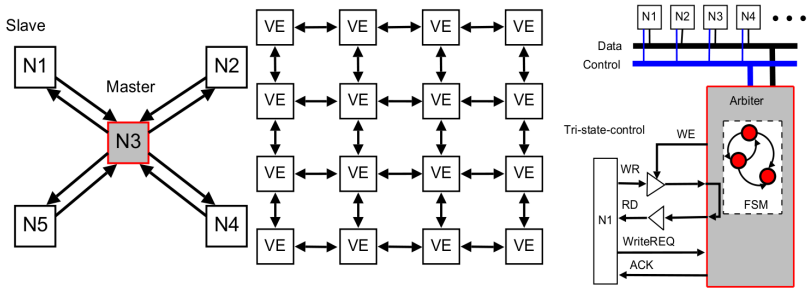


Figure 12. Stern-, Maschen- (Gitter), und Bustopologien

3.11. Eingebettete Systeme: Sensorknoten

1. Aufbau und Funktionsweise eines Sensorknotens - Eingebettetes System
2. Analoger Teil - Digitaler Teil - Technologien
3. Schnittstellen und Energieversorgung
4. DV Architekturen und Entwurf (Software, Hard- und Software, Hardware) → Parallele DV
5. Algorithmen für die Signalverarbeitung - Sensor Fusion - Echtzeitfähige Systeme
6. Betriebssysteme, Virualisierung und Virtuelle Maschinen

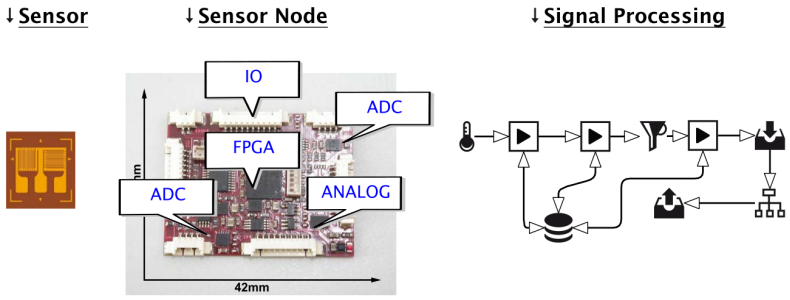


Figure 13. Sensor, Sensor Knoten (mittlere Integrations- und Miniaturisierungsstufe mit FPGA), Signalverarbeitungsfluss

3.12. Kommunikation in Sensornetzwerken

1. Nachrichtenbasierte Kommunikation - Protokolle - Protokollstack
2. Pfadfindung und Routing - Delta und Smart Delta Routing
3. Robustheit - Redundanz - Intelligente Pfadfindung
4. Übertragungstechnologien: Drahtlos versa drahtgebunden

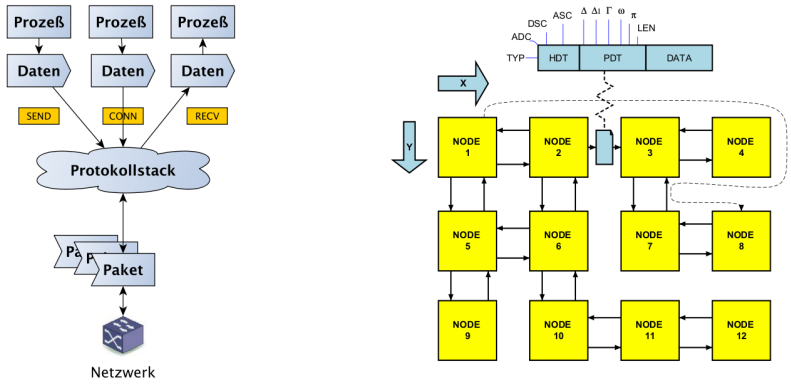


Figure 14. Links: Protokollstack, Rechts: Smartes Delta Routing in fehlerhaften Maschennetzwerken

3.13. Materialintegration

1. Host-Materialien - welche sind geeignet?
2. Komposit Technologie: | Material + Sensor + Elektronik + Energie + Material |

3. Verbindungstechnologien (elektrisch + optisch; Energie und Kommunikation)
4. Energieverteilung und Energiegewinnung (Energiemanagement), Flexible Elektronik

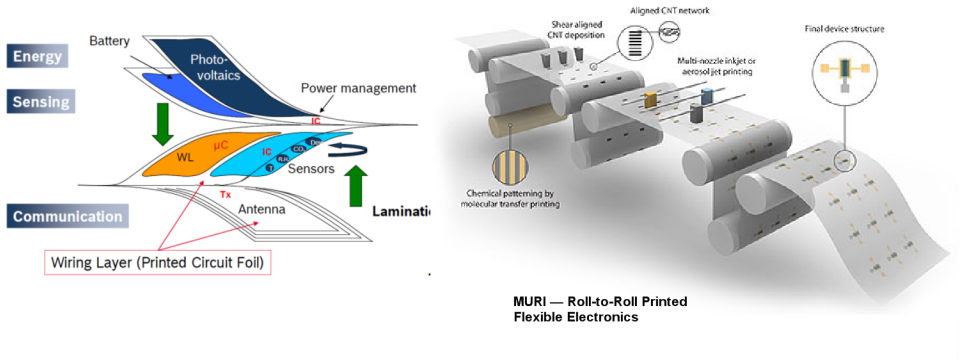
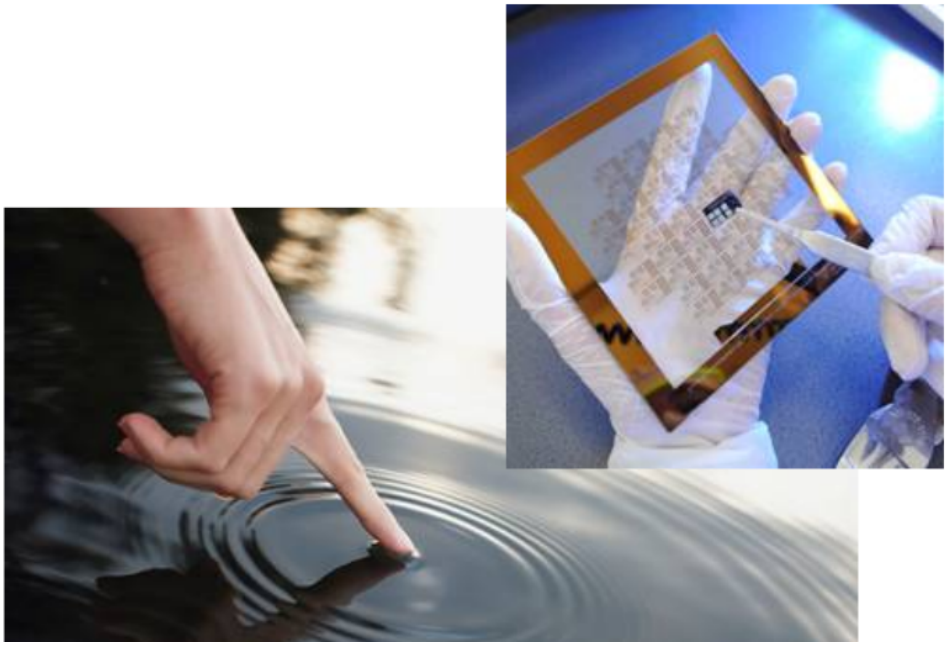


Figure 15. Links: Flexibles SM zwischen Folien, Rechts: Druckverfahren kombiniert mit Roll-to-roll Folierung

4. Sensorische Materialien

4.1. Vision Sensorische Materialien

Sensorische Materialien erfassen Daten über ihre Umgebung und/oder ihren eigenen Zustand. Sie verarbeiten diese Daten lokal und nutzen die gewonnenen Informationen intern oder kommunizieren sie nach außen.



4.2. Materialien, die fühlen können

- Taktile Wahrnehmung ist mehr als nur Sensorik → Funktionale Skalierung
- Zum Beispiel biologische "Mechanorezeptoren" in der menschlichen Haut viel komplexer als technische Drucksensoren.
- Neben der reinen Sensorfunktion gibt es eine Vorverarbeitung (Fusion) der rohen Sensordaten
- Multimodale Perzeption: Gleichzeitige Aufnahme verschiedener Messgrößen und Fusion

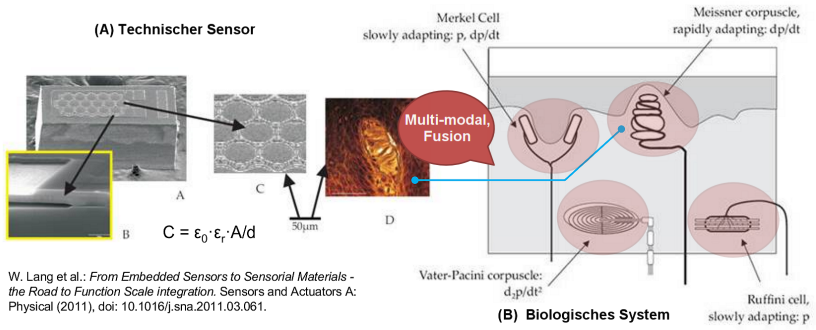


Figure 16. From Embedded Sensors to Sensorial Materials - the Road to Function Scale integration (Lang, 2011)

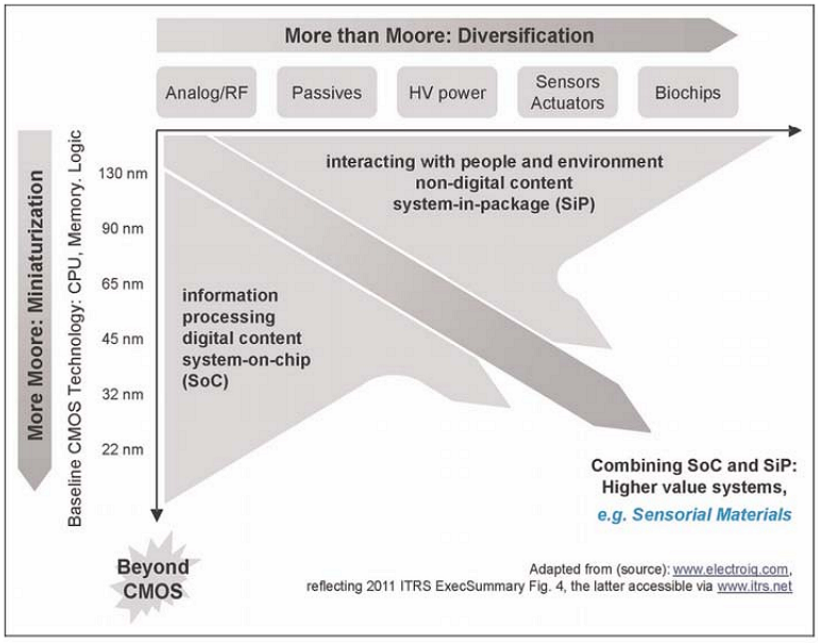


Figure 17. Roadmap to Function Scale Integration (Lang, 2011)

4.3. Werkstoffe, die ihren Zustand kennen

- ➤ Das Internet of Things, Ambient Intelligence und andere Anwendunsszenarien für „smart systems“ würden von materialintegrierten Sensorknoten und ebensolcher Datenverarbeitung profitieren.
- ➤ **Denn reine Oberflächenintegration kann es nicht gewesen sein!**

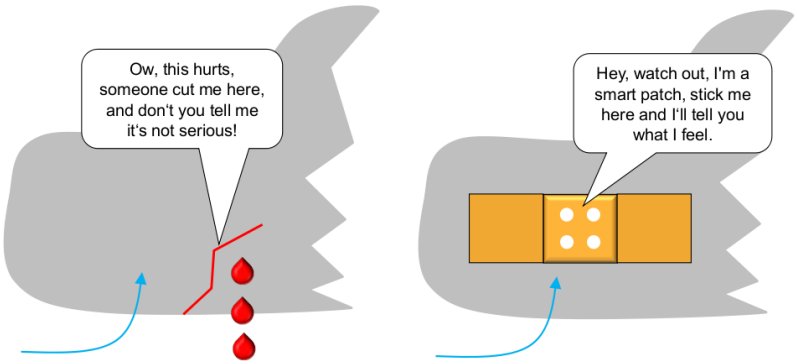
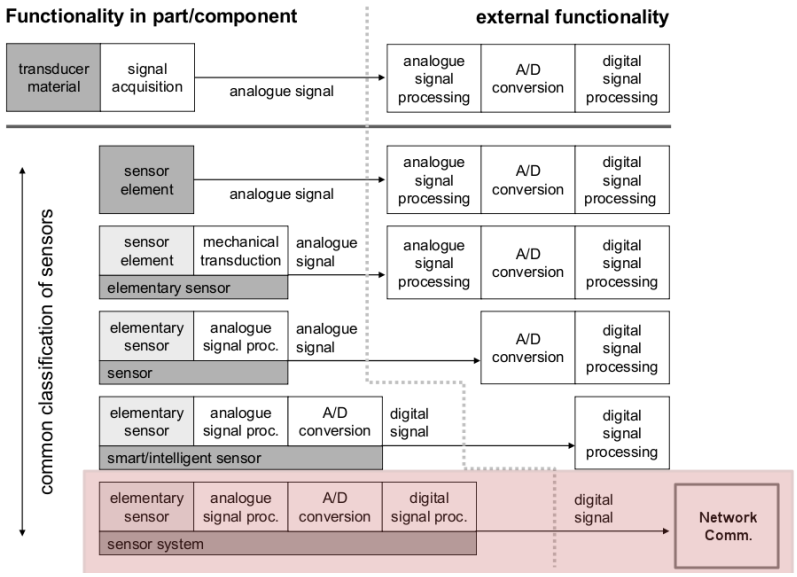


Figure 18. Neben der Perzeption sind Interpretation (des Zustands) und Reaktivität (Änderung des Zustands) wichtige Eigenschaften von smarten Materialien

4.4. Materialintegration

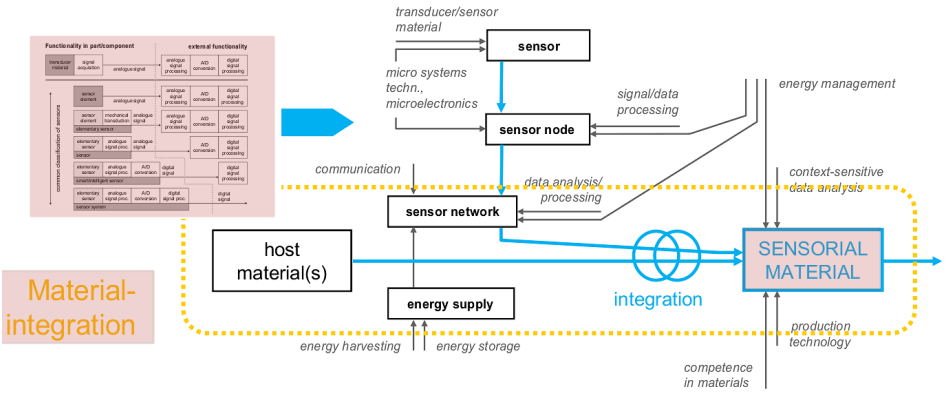
Ansätze

- ➤ Funktionalität ins Material verlagern ist wichtigster Schritt beim Entwurf von SM.
- ➤ Im Zuge der Entwicklung von einer sensorierten Struktur zur Materialintegration wird die Funktionalität des Sensorsystems Schritt für Schritt von der Außenwelt in das aufnehmende Material verlagert.



[Lee, S. H., 2010]

- Die Funktionalität ins Material verlagern bedeutet der Entwurf eines heterogenen Systems bestehend aus:
- Sensoren
- Sensorknoten
- Sensornetzwerk und Kommunikation
- Energieversorgung
- Energiemanagement
- Hostmaterial



[Lehmhus 2013, JIMS]

Integrationsgrad

Bedingt durch Anwendung und Technologie sind verschiedene Integrationsgrade von Sensornetzwerken möglich:

1. Externe Sensorik, externe Überwachung → Immer möglich, keine speziellen Technologien erforderlich
2. Oberflächenapplikation → Meistens möglich, erfordert allgemein verfügbare Klebetechnologien
3. Oberflächenintegration → Außenbereich eines Materials oder einer Struktur → Erfordert spezielle Fertigungstechnologien
4. Volumenintegration → Innenbereich eines Materials oder einer Struktur → Erfordert spezielle Fertigungs-, Sensor-, und Datenverarbeitungstechnologien
5. Intrinsische Materialeigenschaften → Sensormaterial

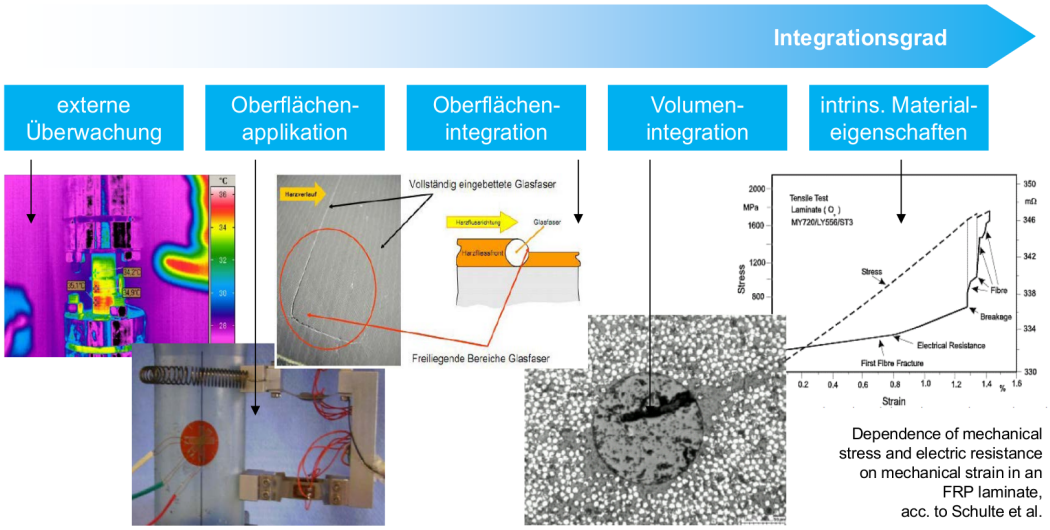


Figure 19. Verschiedene Integrationsgrade an Beispielen und Anwendungen

Intrinsische Materialeigenschaften

- ➤ Elektr. Leitfähigkeit von C-Fasern als Sensoreffekt
  - ❑ D.-Y. Song et al. , Materials Science and Engineering: A 456 (2007) 286-291
- ➤ Gefüllte Matrixwerkstoffe in Faserverbundkompositen (FVK)
  - ❑ Terfenol-D, P. Wierach et al.: Composites modified with Terfenol-D particles for stress detection. Euromat 2013, Sept. 8th-13th, 2013, Sevilla, Spain.
  - ❑ Graphen & C-Nanoröhren (CNT), J.-M. Park et al., Composites Part B: Engineering 39 (2008) 1170-1182.
  - ❑ etc.
- ➤ Magnetische Partikel als Zugabe in Mg
  - ❑ K.-H. Wu et al., J. of Magnetism and Magn. Mat. 322 (2010) 1134-1136.
- ➤ Martensitische Umwandlung von Stählen

- ❏ B.-A. Behrens , K. Weilandt: Materials Science & Technology 3 (2009) 1485-1496.

- ➤ Piezoelektrische Keramiken

Einschränkung

Nur der Sensor und damit nur ein Element der Messkette wird ersetzt!

Möglicher Vorteil

Wenn angepasste Ausleseverfahren z.B. mit räumlicher Fokussierung und damit hoher räumlicher Auflösung verfügbar wären.

Volumenintegration

Klassifikation nach folgender Metrik:

- ➤ Typ des aufnehmenden Materials
  - ❏ Metall, Leichtmetall, (thermoplastischer/duroplastischert) Kunststoff, Textil, faserverstärkter Kunststoff, …
- ➤ Verarbeitung des aufnehmenden Materials
  - ❏ Gießverfahren, Additive Manufacturing, Laminierverfahren, pulverbasierte Verfahren, …
- ➤ Konfiguration des Sensor-Matrix-Verbundes
  - ❏ Klassifikation nach Konnektivität der aufnehmenden Phase und der die Sensorknoten repräsentierenden Phase
- ➤ Sensortyp (nach Messprinzip)
  - ❏ Faseroptischer Sensor (FBG), Dehnungsmessstreifen (DMS), kapazitive Sensoren …
- ➤ Sensortyp (nach Messgröße)
  - ❏ Mechanischer (Dehnungs-) Sensor, Temperatursensor, …

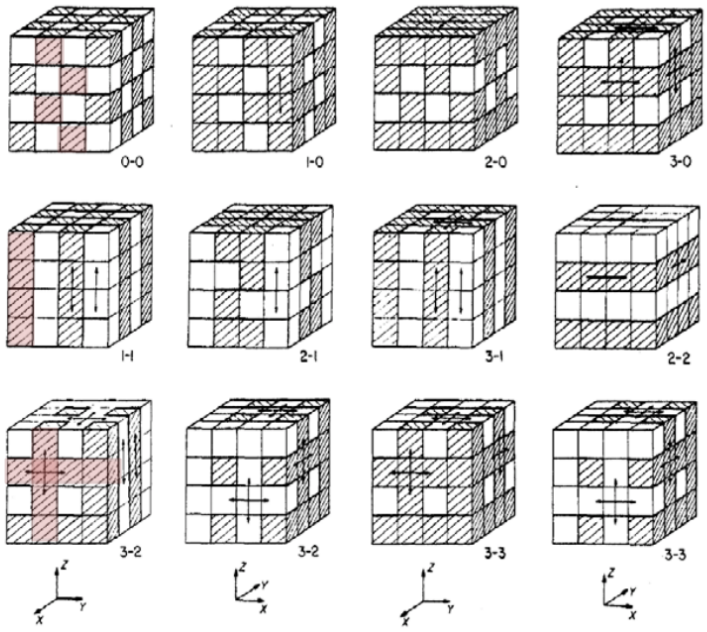


Figure 20. Klassifikation der Integration nach Newnham [R. E. Newnham, D. P. Skinner, L. E. Cross: Connectivity and piezoelectric-pyroelectric composites. Mat. Res. Bull. 13 (1978) 525-536.]

Metrik

➤ Einteilung nach Konnektivität: 1. Ziffer aufnehmendes Material, 2. Ziffer Sensorknoten

3-0

Individuelle Sensorknoten mit physischer Verbindung, mit **Knotengeometrie**:

➤ 0D ("smart dust", sehr geringe Abmessungen in der Größenordnung der eigenschaftsbestimmenden Bestandteile der Matrix)

➤ 1D (Draht/Faser)

➤ 2D (SmartPatch, Funktionales Netz begrenzter Größe)

➤ 3D (komplexe 3D-Knoten, die die 0D-Varianten in ihren Abmessungen um wenigstens eine Größenordnung übertreffen)

3-1

Sensorknoten aus kontinuierlichen, langen Drähten/Fasern oder 1D **Sensornetzwerke** in Faserform

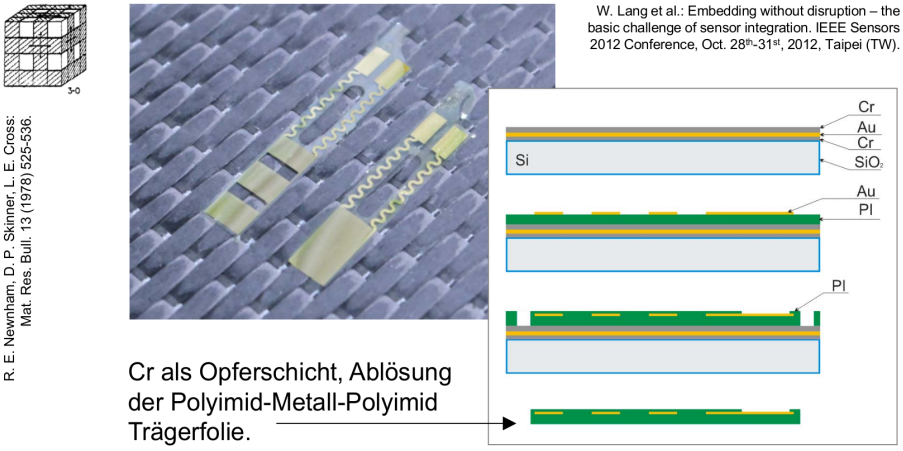
3-2

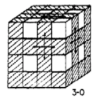
Groß- bzw. vollflächige Funktionale Netze oder Textilien, oder 2D **Sensornetzwerke**

2-2

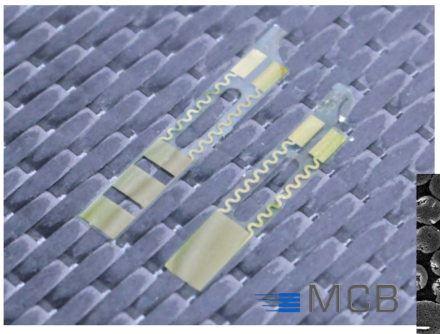
Schichtmaterial mit vollflächigen “smart layers” und 2D Sensornetzwerken

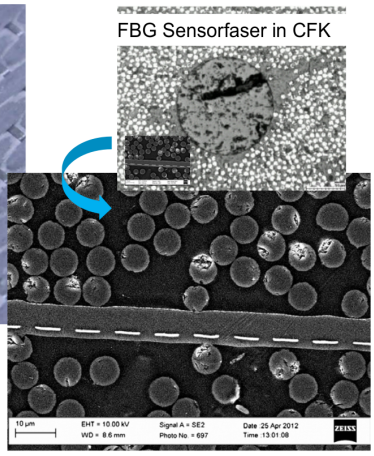
Konnektivität 3-0





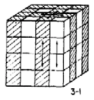
R. E. Newnham, D. P. Skinner, L. E. Cross:
Mat. Res. Bull. 13 (1978) 525-536.



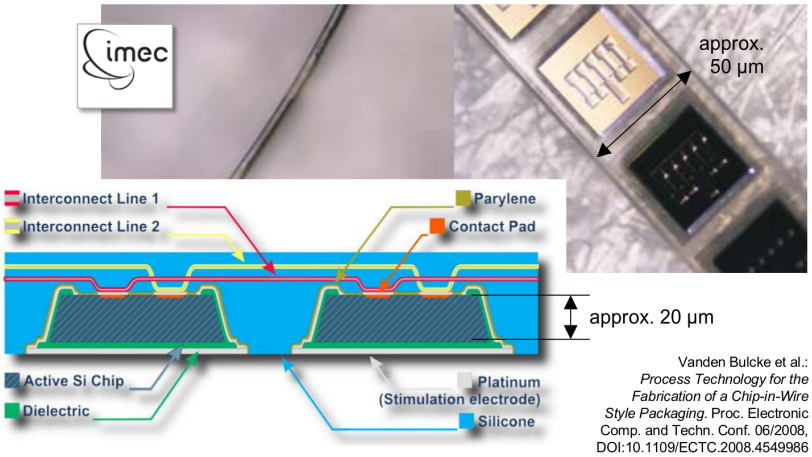


Das gezeigte Konzept repräsentiert allein den Sensor, keinerlei Signal- oder Datenverarbeitung.

Konnektivität 3-1

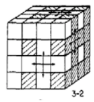


R. E. Newnham, D. P. Skinner, L. E. Cross:
Mat. Res. Bull. 13 (1978) 525-536.

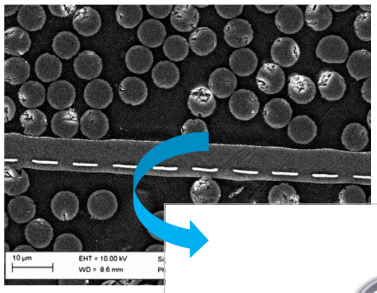


Vanden Bulcke et al.:
Process Technology for the Fabrication of a Chip-in-Wire Style Packaging. Proc. Electronic Comp. and Techn. Conf. 06/2008, DOI:10.1109/ECTC.2008.4549986

Konnektivität 3-2

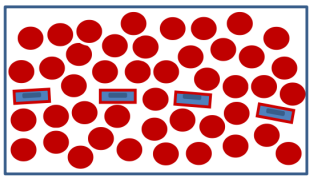


R. E. Newnham, D. P. Skinner, L. E. Cross: Mat. Res. Bull. 13 (1978) 525-536.

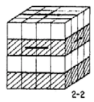


MCB

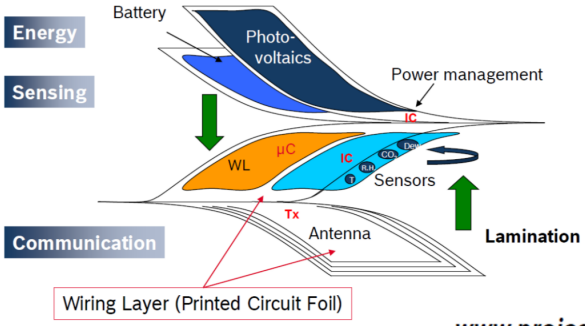
W. Lang et al.: *From embedded Sensors to Sensorial Materials […].* Sensors and Actuators A 171 (2011) 3– 11



Konnektivität 2-2



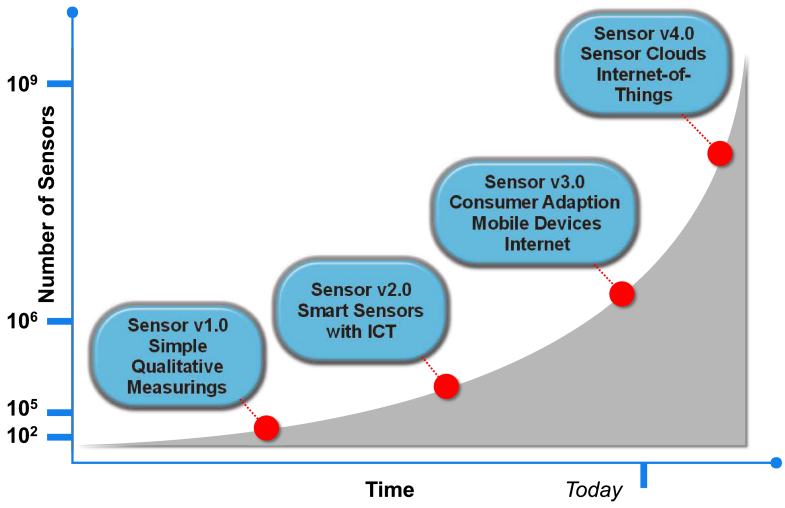
R. E. Newnham, D. P. Skinner, L. E. Cross: Mat. Res. Bull. 13 (1978) 525-536.



4.5. Sensorevolution

Intelligente Sensoren und Erhöhung der Sensordichte

- ➤ Entwicklung von Sensoren mit integrierten Informations- und Kommunikationstechnologien
  - ❏ Generation 1: Diskrete Sensoren, getrennte Sensorverarbeitung
  - ❏ Generation 2: Integrierte Smarte Sensoren
  - ❏ Generation 3: Netzwerke und Internetkonnektivität
  - ❏ Generation 4: Sensorclouds, IoT, MIIS


Sensor v4.0
Sensor Clouds
Internet-of-
Things
Sensor v3.0
Consumer Adaption
Mobile Devices
Internet
Sensor v2.0
Smart Sensors
with ICT
Sensor v1.0
Simple
Qualitative
Measurings
10^9
10^6
10^5
10^2
Number of Sensors
Time
Today

5. Eingebettete Systeme und Datenverarbeitung

5.1. Rechnerarchitektur

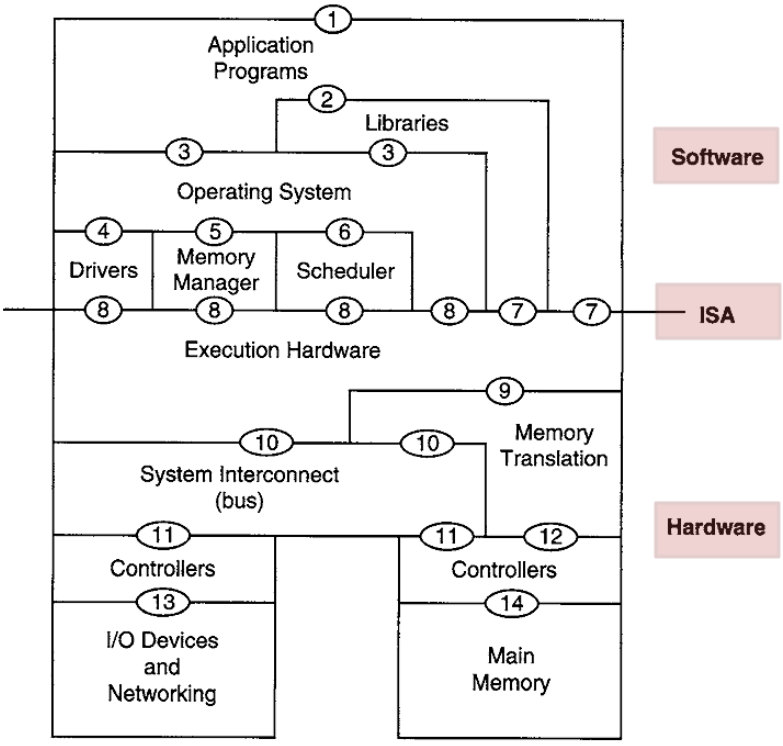


Figure 21. Computersystemarchitekturen. Implementierungsebenen kommunizieren vertikal über die gezeigten Schnittstellen. (Nach Glenford Myers, 1982) [A]

5.2. Materialinformatik

Bedeutung:

1. Rechnen in Materialien (Verteilte Datenverarbeitung in Materialien)
2. Rechnen von Materialien (Data Science, KI, .., um neue Materialien zu entwickeln)

➤ Normalerweise wird die Berechnung von Sensorerfassung und Steuerung getrennt

➤ Smarte Materialien stellen die enge Verbindung zwischen

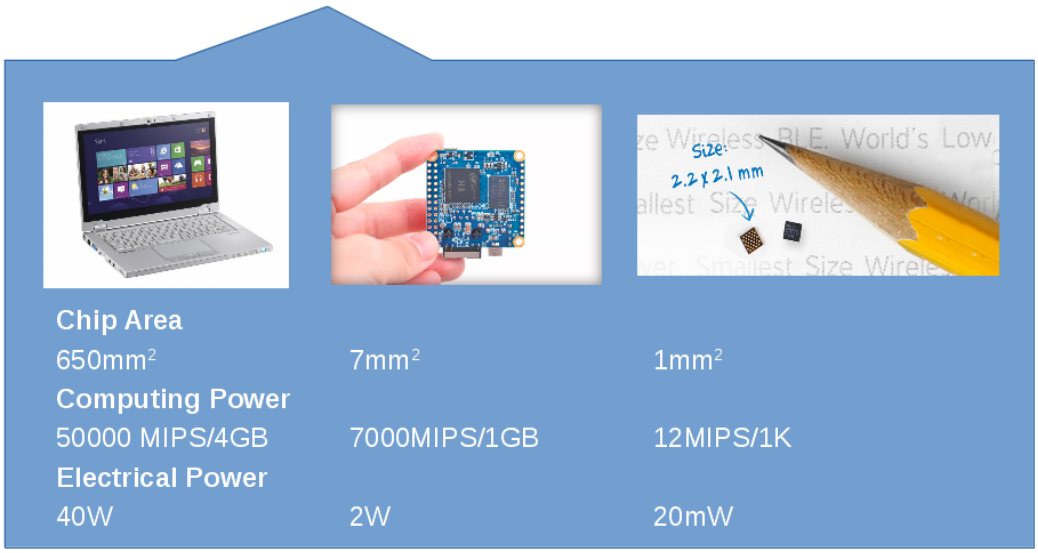
- ❏ Berechnung,
- ❏ Kommunikation,
- ❏ Sensorerfassung und
- ❏ Steuerung mit lose gekoppelten Nano-Computern dar.

➤ Algorithmische Skalierung und Verteilung sind erforderlich

- ❏ Verteilte Systeme können als eine große Maschine behandelt werden

➤ Mooresches Gesetz sagte starke Miniaturisierung voraus

➤ Jedoch die Informatik und deren Methoden konnten nicht immer folgen → Lücke zwischen Hardware und Software



Eine normalisierte Recheneffizienz eines Computers (nur unter Berücksichtigung der Datenverarbeitungseinheit) kann definiert werden durch:

$$\epsilon_C = \frac{C}{AP}$$

A

Chipfläche in mm^2

C

Rechenleistung in Mega Instructions per Second (MIPS) - oder besser in Kilo *Dhrystones/s* (KDS)!

P

Elektrische Leistungsaufnahme in W

Die Recheneffizienz kann verwendet werden, um verschiedene Computer und Geräte zu vergleichen, d.h. einen Skalierungsfaktor anzugeben:

$$s = \frac{\epsilon_1}{\epsilon_2}$$

➤ Neben der reinen Rechenleistung sind noch Speicher und Kommunikationsfähigkeit einer Rechneranlage wichtige Kenngrößen, so dass sich zwei weitere normierte Recheneffizienzen ergeben:

$$\epsilon_{CM} = \frac{CM}{AP}$$

$$\epsilon_{CMD} = \frac{CMD}{AP}$$

M

Gesamter Speicher in Mega Bytes (MB), beinhaltet RAM, ROM, Register

D

Gesamte Kommunikationsfähigkeit als Datendurchsatz (Mega Bit/s)!

Maßzahlen der Rechenleistung

➤ Einfachste Maßzahl ist die Anzahl der Integer- oder Fliesskommaoperationen pro Zeiteinheit (MIPS/FLOPS)

- ❏ Aber nur der eigentliche Kern des Mikroprozessors wird dabei erfasst - Speicher, Speicherhierarchie und Kommunikationsfähigkeit fehlen

➤ Ein Programm besteht i.A. aus den folgenden High-level Operationen:

- ❏ Berechnung (skalar, vektoriell)
- ❏ Speicherallokation, Objekterzeugung (Code und Daten)
- ❏ Funktionsaufrufe
- ❏ Erzeugung, Zugriff, und Freigabe verschiedener Objekte mit unterschiedlichen “Speicherabdruck”: Arrays, Strings, Records, Funktionen, Methodische Objekte mit Prototypen (Klassen)

⇒ **Dhrystone** Benchmark umfasst alle oben genannten Operationen!

Chip Fläche

Item Size in rbe	
1 register bit (**rbe**)	1.0 **rbe**
1 static RAM bit in an on-chip cache	0.6 **rbe**
1 DRAM bit	0.1 **rbe**
rbe corresponds to (in feature size: f)	1 $\mathbf{rbe} = 675f^2$
Item A corresponds to 1 mm^2 with $f = 1\mu$.	Size in A Units
$1A$ or about	= $f^2 \times 10^6$ (f in microns) ≈ 1481 **rbe**

Figure 22. Normierte Maßzahlen für die Chip Fläche (A/rbe) [H]

$1A$ or about	= $f^2 \times 10^6$ (f in microns) ≈ 1481 **rbe**
A simple integer file (1 read + 1 read/write) with 32 words of 32 bits/word or about	 = 1444 **rbe** $\approx 1A$ ($= 0.975A$)
A 4KB direct mapped cache or about	= 23,542 **rbe** $\approx 16A$
Generally a simple cache (whose tag and control bits are less than 1/5th the data bits) uses	 $= 4A/KB$

Figure 23. Normierte Maßzahlen für die Chip Fläche von Speicher (A/rbe) [H]

Simple processors (approximation)	
A 32-bit processor (no cache and no floating point)	$= 50A$
A 32-bit processor (no cache but includes 64-bit floating point)	$= 100A$
A 32b (signal)processor, as above, with vector facilities but no cache or vector memory	$= 200A$
Area for inter-unit latches, buses, control and clocking	allow an additional 50% of the processor area.

Figure 24. Normierte Maßzahlen für die Chip Fläche von Prozessoren (A/rbe) [H]

Xilinx FPGA	
A slice (2 LUTs + 2 FFs + MUX)	= 700 **rbe**
A configurable logic block (4 slices) Virtex 4	= 2,800 **rbe** $\approx 1.9A$
A 18Kb BlockRAM	= 12,600 **rbe** $\approx 8.7A$
An Embedded PPC405 core	$\approx 250A$

Figure 25. Normierte Maßzahlen für die Chip Fläche von FPGAs (A/rbe) [H]

Feature size (in micron)	Number of A per mm^2
1.000	1.00
0.350	8.16
0.130	59.17
0.090	123.46
0.065	236.69
0.045	493.93

Figure 26. Zusammenhang der Technologiegröße f (Fertigungsauflösung/Transistordimension) mit der normierten Fläche A [H]

➤ Hitachi entwickelte bereits 2006 einen 0.15 x 0.15 Millimeter großen, 7.5 μm dicken Microchip der:

- ❏ Drahtlose Kommunikation und Energieversorgung via RFID → (2.45 GHz Mikrowelle) ermöglichtet, einen
- ❏ 128 Bit ROM Speicher, und einen
- ❏ einfachen Mikroprozessor enthielt.

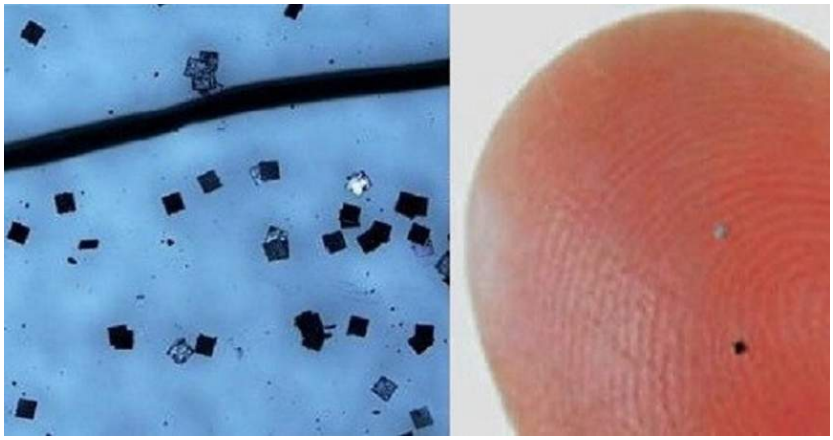


Figure 27. Es geht noch kleiner → Smart Dust reloaded (by Hitachi)

Aufgabe

1. Mache eigene Messungen mit dem *dhrstone* benchmark Test auf verschiedenen Rechnern und Virtuellen Maschinen.
2. Stelle eine Tabelle zusammen mit gängigen Computern (mobil, Desktop, Server, eingebettete Rechner, Nanorechner) mit den Daten MIPS/DPS, Speicher, Kommunikation (abgeschätzt) sowie Chip Fläche und el. Leistungsbedarf
3. Berechne die verschiedenen ϵ Parameter und vergleiche …

5.3. Entwurf Eingebetteter Systeme

Softwareentwurf

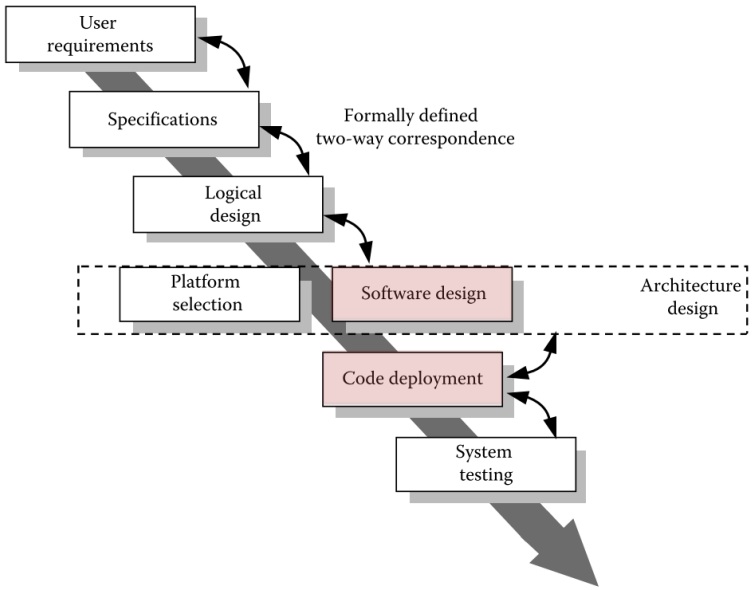


Figure 28. Entwurfsebenen von Eingebetteten Systemen auf Softwareebene

➤ Der Entwurf Eingebetteter Systeme geht immer mehr Richtung Hardware-Software Co-Design und System-on-Chip Architekturen

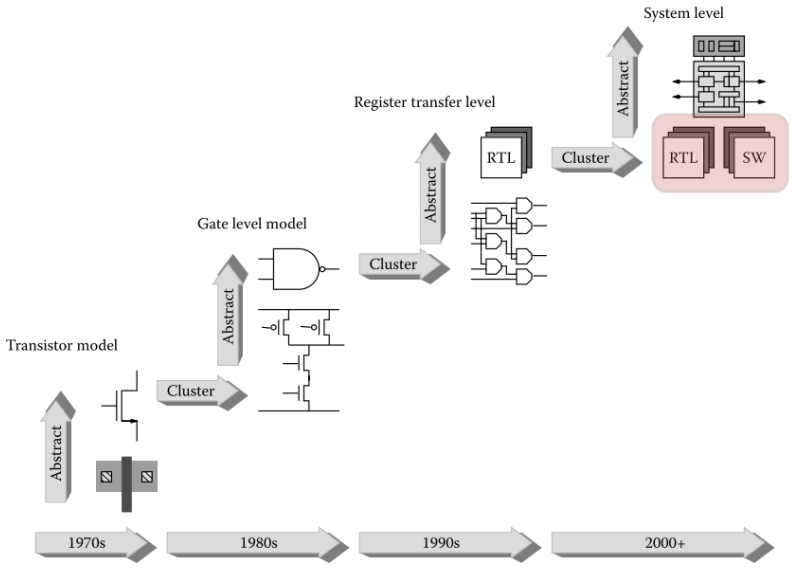


Figure 29. Entwurfsebenen von Eingebetteten Systemen auf Hardwareebene

5.4. Rechnersysteme

➤ Man unterscheidet folgende Kategorien von Rechnern:

A. Stationäre Geräte: Server

B. Stationäre Geräte: Desktop

C. Mobile Geräte: Notebook

D. Mobile Geräte: Smartphone

E. Stationäre und mobile Eingebettete Systeme: Einplatinencomputer und Microcontroller

F. System-on-Chip Geräte: Einchipcomputer und Microcontroller

5.5. Rechnertechnologien

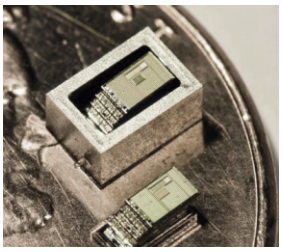
Existierende “Nano”-Computer:

➤ Smart Dust → Vision Millionen lose gekoppelter Nano-Computer

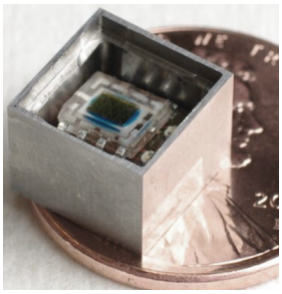
❑ Eingebettet in Materialien

- ❏ Auf Oberflächen verstreut
- ❏ Dispergiert in Flüssigkeiten, Folien, ..
- ❏ ungefähr 10mm^3 Volumen

Micro Mote M3



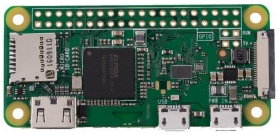
ELM System



Existierende Kleinstrechner für Sensornetzwerke mit Drahtloskommunikation (WLAN, Bluetooth)

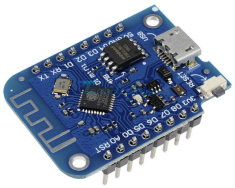
Raspberry PI Zero

- ➤ 32-bit RISC ARM Prozessor
- ➤ 512MB RAM, 16GB ROM
- ➤ 1W
- ➤ 70x30mm



ESP8266

- 32-bit RISC Tensilica Xtensa Diamond Standard
- 128kB RAM, 16MB ROM
- 0.15W
- 30x20 mm



	Micro Mote (M3)	**ELM System**	**Atmel Tiny 20**	**Freescale KL03**	**ARM Cortex Smart Phone**
Processor	Arm Cortex M0	C8051F990 (SL)	AVR	Arm Cotex M0+	Arm Cortex A9
Clock	740kHz max.	32kHz	-	48MHz	1GHz
CPU Chip Area	0.1mm^2	9mm^2	1mm^2	4mm^2	7mm^2/ROM
Sensors	Temperature		-	-	Temp, Light, Sound, Accel., Press., Magn.
Communication	900MHz radio, optical	optical	electrical	-	3G/4G, WLAN, USB, Blue-tooth, NFC
Harvester, Battery	Solar cell, Thin film	Solar cell, Coin	-	-	-
Power Consumption	70mW / CPU	160mW / CPU	20mW	3mW @ 48MHz	100mW avg.,
Manufacturing	180nm CMOS	-	-	-	40nm CMOS
Package	Wire bonded	Silicon Stack	PCB	Single Chip	Single Chip
Computing Eff. ϵ_C	**150**	**0.02**	**0.6**	**4.0**	**0.53**

Smart Dust

- ➤ Kommunikation: Optisch, Laser, LED
- ➤ Energieversorgung: Optisch, Fotozelle
- ➤ Energiespeicher: Dünnfilmbatterie
- ➤ Sensorik: Temperatur, Licht

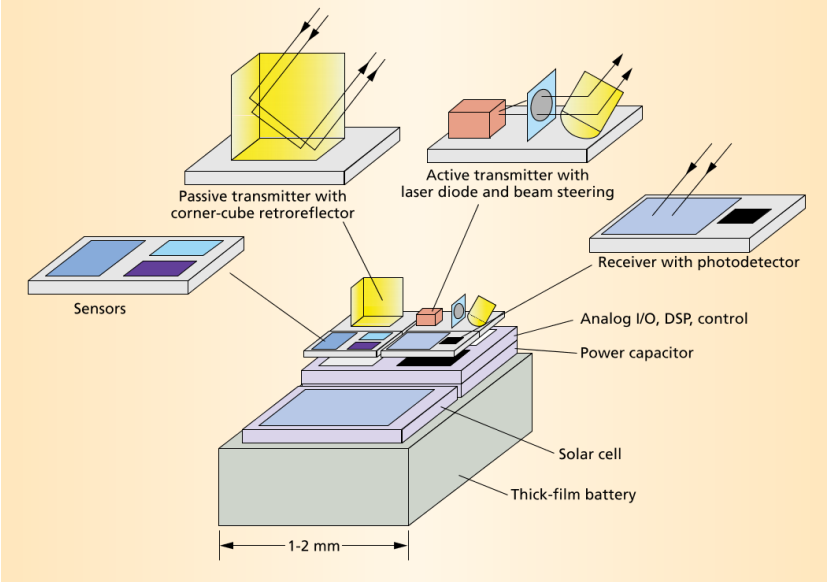


Figure 30. Aufbau einer Smart Dust Mote [Smart Dust: Warneke et al., 2001]

5.6. Rechnernetzwerke

Von großen Computern zu großen Netzwerken ...

- Ursprünglich wurden Software- und Betriebssysteme auf Computern mit hoher Rechenleistung und Speicherkapazität ausgeführt.
- Die Integration von Berechnung in technische Strukturen oder Geräte erfordert das Herunterskalieren von Algorithmen und Methoden in Richtung auf verteilte Verarbeitungsnetzwerke mit Plattformen mit geringen Ressourcen.

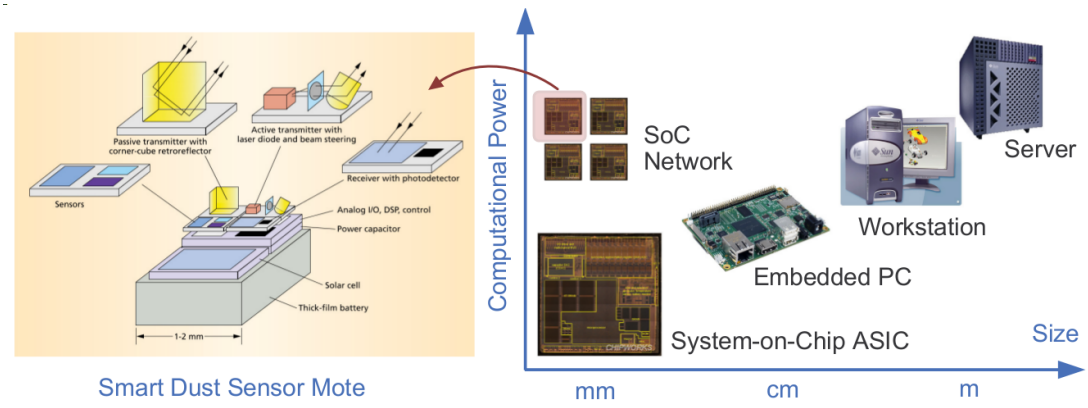


Figure 31. Skalierung und Miniaturisierung [Smart Dust: Warneke et al., 2001]

5.7. Rechnertypen

- ➤ Rechner und Mikroprozessoren unterscheiden sich u.A. durch die Art und Weise der Maschinenbefehlsausführung
- ➤ Man unterscheidet: Reduced Instruction Set Computer (RISC) und Complex Instruction Set Computer (CISC)
  - ❑ RISC: Maschinenbefehle arbeiten häufig mit Registern (Anzahl Register groß)
  - ❑ CISC: Maschinenbefehle arbeiten häufig mit dem Speicher (Anzahl der Register klein)
  - ❑ Eingebettete Systeme verwenden i.A. RISC (Aufgabe: Warum?)
- ➤ Es gibt Maschinenbefehle mit einer unterschiedlicher Anzahl von Operanden (0,1,2,3,..)
- ➤ Neben register- und speicherbasierten Rechnern gibt es reine stackbasierte Rechner (Nulloperandenbefehle)
  - ❑ Alle Operationen werden über den Stapelspeicher ausgeführt

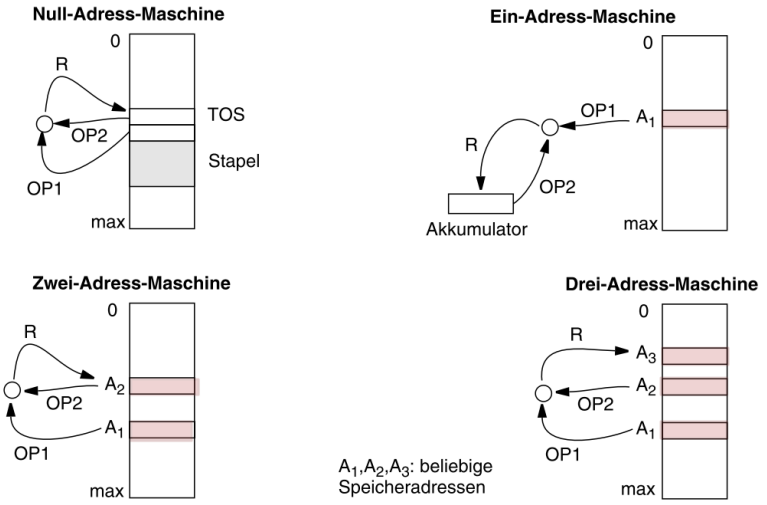


Figure 32. Rechnereinteilung nach Adressanzahl: gezählt wird die Anzahl Operanden eines dyadischen Operators (z.B. einer Addition) [C].

5.8. Programmierung

Randbedingungen

Die Programmierung eingebetteter Systeme stellt aufgrund folgender Randbedingung eine Herausforderung dar:

1. Geringe Rechnenleistung pro Netzknoten
2. Geringe Speicherkapazität von Daten
3. Bei energieautarken Systemen gibt es eingeschränkte Laufzeit
4. Geringe Kommunikationsleistung
5. Fehleranfälligkeit
6. Schlechte Wartbarkeit (Softwareupdates usw.)
7. Eingeschränkte Möglichkeiten der Fehlersuche

Programmiersprachen

➤ Assembler → Maximale Performanz, minimalste Zuverlässigkeit/Korrektheit (Fehler), minimalster Speicherbedarf, kein automatisches Speichermanagement

- C/C++ → Sehr gute Performanz, geringer Speicherbedarf, und mittlere Zuverlässigkeit/Korrektheit (Fehler), kein oder minimales automatisches Speichermanagement → Prozedurale Programmierung
- JAVA → Mittlere Performanz, hoher Speicherbedarf, gute Zuverlässigkeit/Korrekheit (Fehler), automatisches Speichermanagement → Objektorientierte Programmierung
- Skriptsprachen → Unterschiedliche Performanz, wird durch Interpreter bestimmt, automatisches Speichermanagement
  - Python (schlechte Performanz)
  - JavaScript (mittlere oder gute Performanz, evtl. hoher Speicherbedarf)
  - Lua (mittlere oder gute Performanz, niedriger Speicherbedarf)

Performanz und Speicherbedarf bei Skriptsprachen hängen vom Interpreter ab!

- JavaScript: Google V8 (Bytecode+Just-in-time native code Compiler), Jerryscript (Bytecode)
- Lua (Bytecode) und Luajit (Bytecode+Just-in-time native code Compiler)

Bytecode. Der Bytecode ist eine Sammlung von Befehlen für eine virtuelle Maschine.

- Bei der Kompilierung eines Quelltextes mancher Programmiersprachen oder Umgebungen, wie beispielsweise Java, wird nicht direkt Maschinencode, sondern ein Zwischencode, der Bytecode, erstellt.
- Dieser Code ist in der Regel unabhängig von realer Hardware. Er entsteht als Resultat einer semantischen Analyse des Quelltexts und ist im Vergleich zu diesem oft relativ kompakt und wesentlich effizienter interpretierbar als der originale Quelltext [Wikipedia].
- Reine Bytecode Interpreter haben geringe oder mittlere Performanz bei niedrigen Speicherbedarf
  - Der Bytecode wird entweder vollständig aus dem Quelltext beim Start eines Programms erzeugt, oder
  - Der Bytecode wird stückweise nach Bedarf aus dem Quelltext zur Laufzeit erzeugt
- Ein JIT Compiler übersetzt häufig vorkommende Bytecode Abschnitte zur Ausführungszeit in nativen Maschinencode

- ❏ Das ergibt erhöhte Ausführungsperformanz, benötigt aber mehr Speicher

➤ Vorteile von Skriptsprachen gegenüber kompilierten Programmen: Schneller Test, ausführliche und genaue Rückmeldung vom Interpreter bei Fehlern, bessere Laufzeitüberwachung von Fehlern, …

Ausführungsplattform für Eingebettete Systeme

➤ Daher ist eine zweistufige Architektur sinnvoll:

- ❏ Virtuelle Maschine programmiert in C (hohe Portabilität) und ggfs. Assembler, mit integrierten Compiler, die Bytecode ausführt und bei Bedarf Bytecode in nativen Maschinenkode umwandelt (JIT)
- ❏ Skriptsprache (in diesem Kurs Lua) die im Quelltext von der virtuellen Maschine ausgeführt wird

➤ Virtualisierung durch die VM von:

- ❏ Prozessor (Instruction Set Architecture, ISA)
- ❏ Speicher, automatisches Speichermanagement
- ❏ Ein- und Ausgabegeräten (Geräte)
- ❏ Kommunikation

➤ Warum nicht JAVA?

- ❏ Hoher Speicherbeadarf
- ❏ Nur objektorientierte Programmierung, nicht immer effizient
- ❏ Kein Interpreter: Compiler und VM sind getrennt

5.9. Virtualisierung

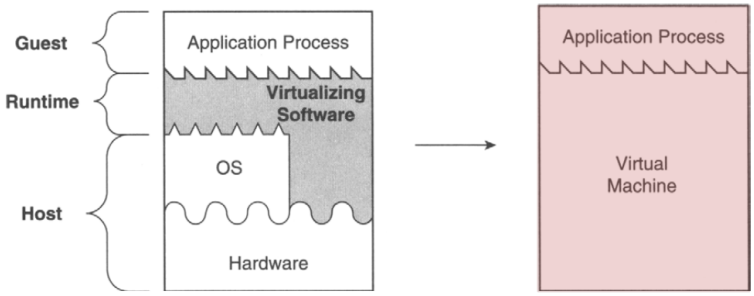
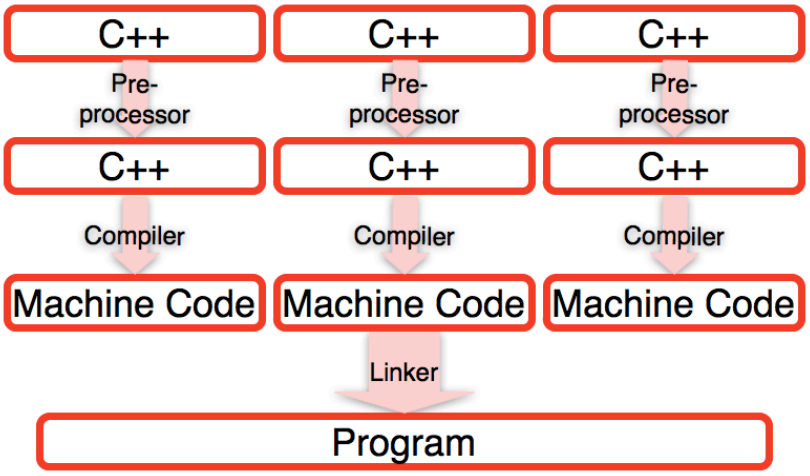


Figure 33. Eine prozessvirtuelle Maschine, die Programme ausführen kann, die für ein anderes Betriebssystem und eine andere ISA entwickelt wurden: Die Virtualisierungssoftware bildet eine Plattform auf eine andere ab, und übersetzt eine Reihe von Betriebssystem- und Benutzerebenenbefehlen. [A]

Compiler



[http://doc.gold.ac.uk]

Figure 34. Klassischer Softwareentwurf mit Compiler und Linker (C)

Interpeter

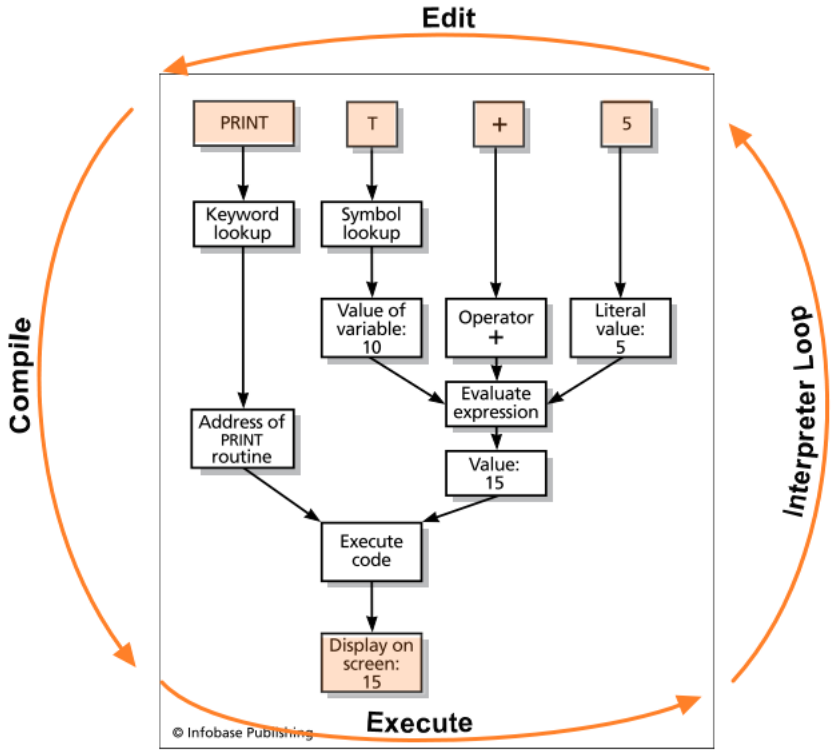
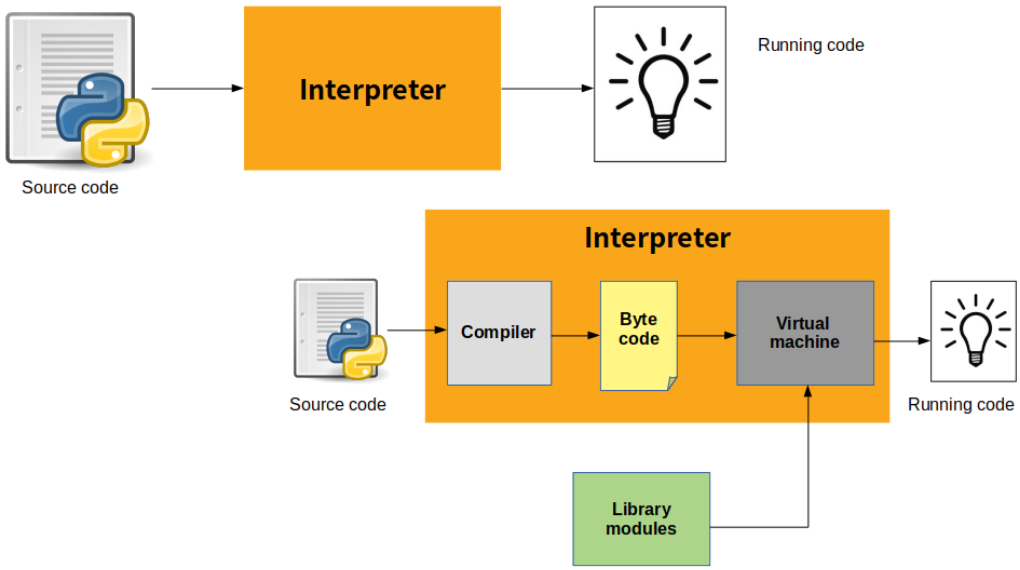


Figure 35. Edit - Compile - Execute Zyklus bei einem Interpreter

Bytecode Interpreter

➤ Die Übersetzung des Quelltextes in Bytecode kann vor und während der Ausführung des Programms erfolgen!

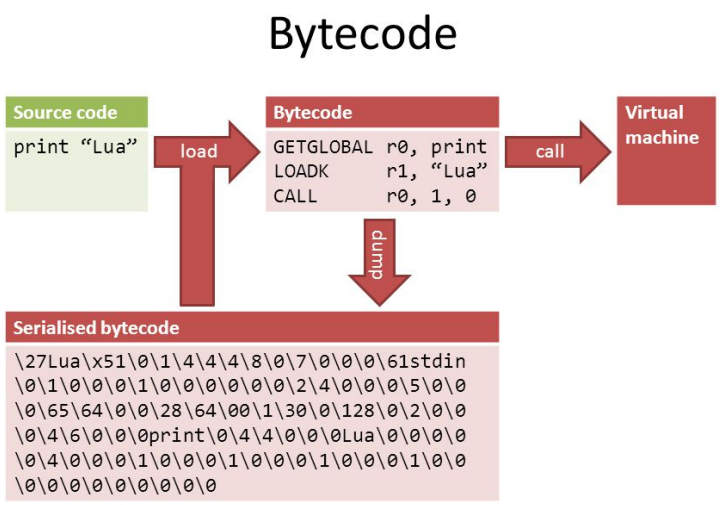


[Python]

Figure 36. Vergleich Interpreter mit Bytcode Compiler-Interpreter System

Serialisierung

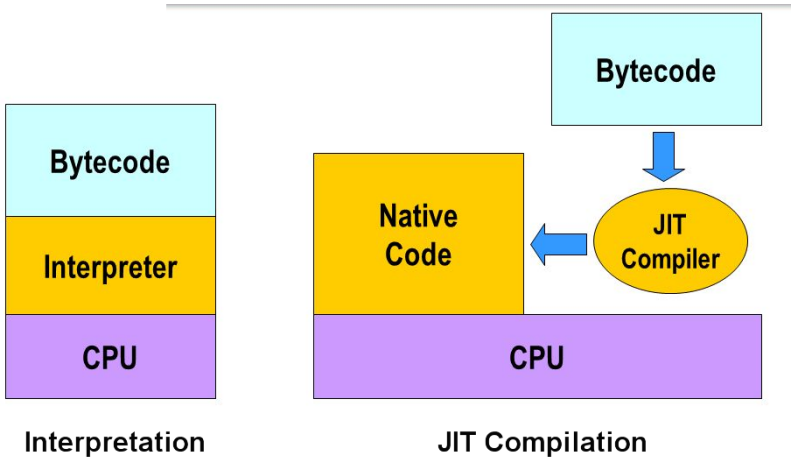
- Da Bytecode unabhängig von der Hostplattform sein sollte, kann Bytecode einfach von einer Maschine zu einer anderen übertragen und ausgeführt werden
- Dazu ist eine **Serialisierung** von Daten und Code erforderlich (Flache Liste von Bytes), mit anschließender **Deserialisierung** (Wiederherstellung der Daten- und Codestruktur)



[Peter Cawley]

JIT Compiler

➤ Neben der Bytecode Ausführung kann während der Laufzeit des Programms der Bytecode stückweise optimiert werden → Erzeugung von Maschinencode durch einen Just-in-Time Compiler (JIT)



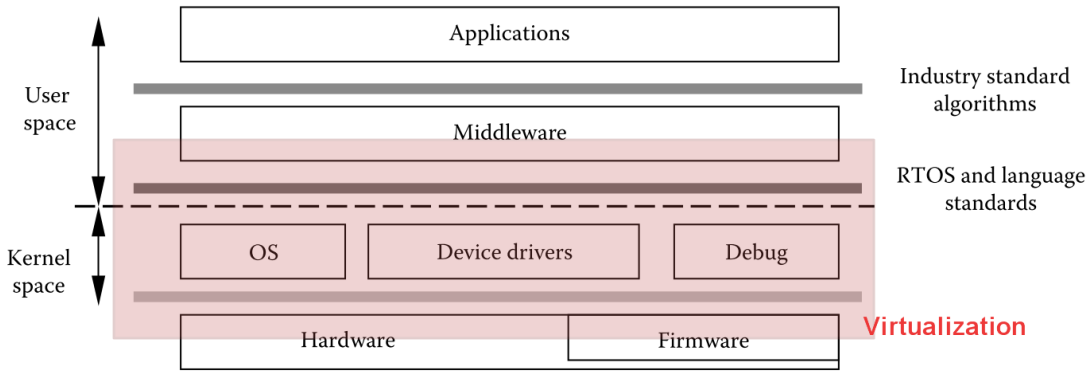
[King Fahd University of Petroleum and Minerals]

Figure 37. Vergleich Interpreter mit JIT Compiler-Interpreter Systeme

Aufgabe:

1. Überlegen Sie sich die Vor- und Nachteile von
   - ➤ Compilern
   - ➤ Interpretern
   - ➤ Bytecode Interpretern
   - ➤ JIT Compiler und Interpreter
2. Was sind die Bewertungskriterien?
3. Welche Ausführungsarchitekturen könnten für Eingebettete Systeme geeignet sein?

➤ Rechenleistung, Speicherbedarf, und Ein-Ausgabe Durchsatz/Latenz sind wichtige Metriken für den Einsatz in Eingebetteten Systemen



[B]

Figure 38. Typische Datenverarbeitungsebenen in Eingebetteten Systemen und Virtualisierung

Perfomanz von Interpretern

Dhrystone Benchmark (Kombination aus Berechnung, Objekten, Arrays, Strings, Funktionen)

VM/OS-ARCH	**Linux-i686**	**Linux-armv6l (PI-3)**	**Linux-armv6l (PI Zero)**
python2.7	105k/s	10k/s	4k/s
lua 5.1	140k/s	-	-
luajit 2.0.5X	660k/s	71k/s	40k/s
jerryscript 1.1.7X	45k/s	-	-
nodejs 4	6300k/s	350k/s	40k/s
C/gcc	18000k/s	?	?

Speicherbedarf von Interpretern

Dhrystone Benchmark (Kombination aus Berechnung, Objekten, Arrays, Strings, Funktionen)

VM/OS-ARCH	**Linux-i686**	**Linux-armv6l (PI-3)**	**Linux-armv6l (PI Zero)**
python2.7	-	-	-
lua 5.1	-	-	-
luajit 2.0.5X	2MB	-	-
jerryscript 1.1.7X	2MB		
nodejs 4	24MB	-	-
C/gcc	?	?	?

Automatisches Speichermanagement

- In C/C++ muss für jedes zur Laufzeit dynamisch erzeugte Datenobjekt (Array, String, Record, ..) immer explizit Speicherplatz im Hauptspeicher angefragt werden (`malloc`) und wieder frei gegeben werden wenn das Datenobjket nicht mehr benötigt wird (`free`)
- In Skriptsprachen gibt es i.A. ein automatisches Speichermanagement mit einem sog. Garbage Collector und Objektreferenzierung
- Jedes Datenobjekt welches verwendet wird (z.B. in Variablen oder Funktionen) besitzt eine Referenz

➤ Es muss eine Wurzeltabelle geben von der aus alle Referenzen auf verwendete Objekte auffindbar sind

Example 1. (*Beispiel von Objektreferenzierungen in Lua: Wo existieren Referenzen?*)

var $o_1 = \{ a = 1,\ b = \{1,2,3,4\} \}$
var $o_2 = \{ op = o_1,\ index = 2 \}$
function *show* () *print* $(o_2.op.a\ \}$ **end**; *show*()

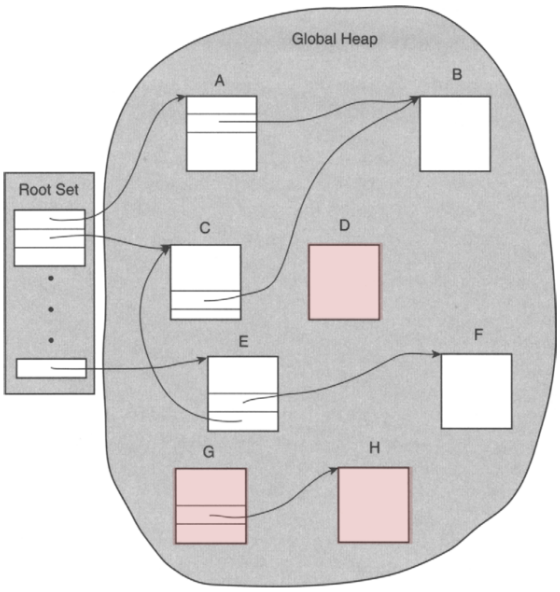


Figure 39. Verschiedene Objekte: Objekte die nicht mehr referenziert sind, werden freigegeben

Speicherarchitektur von Programmen

➤ **Heap**: Datenobjekte mit längerer Lebensdauer (Tabellen- oder Listenstruktur) → Benötigt Speichermanagement

➤ **Stack**: Datenobjekte mit kurzer Lebensdauer (Stapelstruktur) → Benötigt kein Speichermanagement

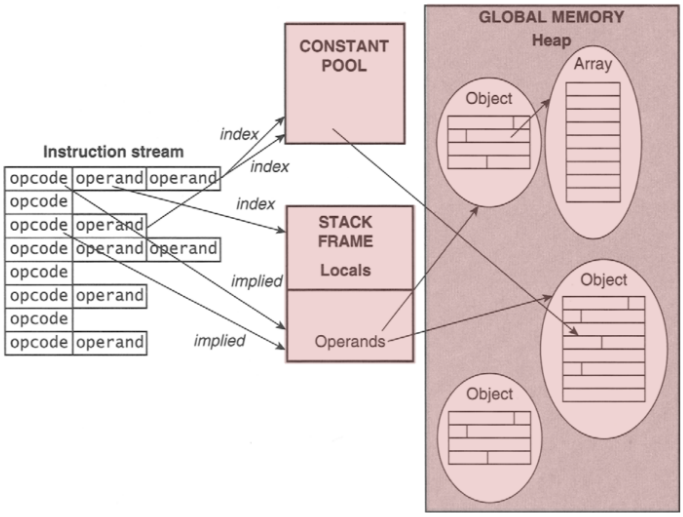
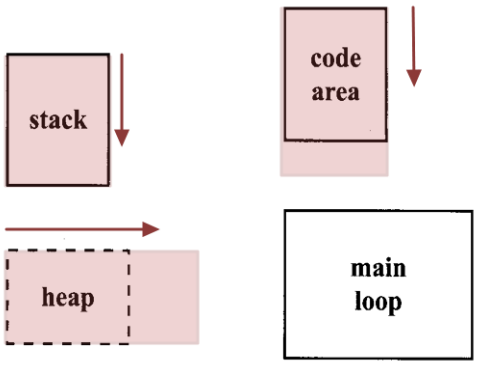


Figure 40. Speicherhierarchie von Programmen und Zusammenhang mit Maschinenanweisungen

Speicherorganisation von Virtuellen Maschinen



Speicher

- ➤ Die meisten Objekte in Programmiersprachen benötigen Speicher
  - ❑ Lokale, temporäre, und globale Variablen
  - ❑ Objekte

- ❏ Arrays
- ❏ Zeichenketten
- ❏ Funktionen (!)

➤ Der Kontext von Speicherobjekten ist wichtig:

- ❏ **Modul** → limitierte Programmsichtbarkeit
- ❏ **Lokal** → stark limitierte Programmsichtbarkeit
- ❏ **Global** → uneingeschränkte Sichtbarkeit

➤ Beispiele: Die globale Variable *Xglobal* ist überall, auch außerhalb des Definitionsmoduls, sichtbar, die Variable *xlokal* ist im gesamtem Modul und den Funktionen *f1* und *f2* sichtbar, wo hingegen *xlokal1* und *xlokal2* jeweils nur ihrem Funktionskontext sichtbar sind.

```
Xglobal = 1
local xloakal = 2
function f1 ()
  local xlokal1 = 3
end
function f2 ()
  local xlokal2 = 3
end
```

➤ Nur wenn Variablen bei der Ausführung sichtbar sind belegen sie Speicher!

➤ Ein Teil des Speicher ist vorbelegt:

- ❏ Globale Variablen
- ❏ Funktionen, ausführbarer Code, …

➤ Ein anderer Teil des Speichers wird zur Laufzeit auf dem Stack und Heap belegt und benötigt Verwaltung:

- ❏ Lokale und temporäre Variablen, ..
- ❏ Funktionen, ausführbarer Code, …

➤ Achtung: Bei Programmiersprachen bei denen Funktionen Werte erster Ordnung sind befindet sich auch Code im Heap oder auf dem Stack

Speicherverwaltung

Listen

Freie und belegte Speicherbereiche werden durch Listen (einfach- oder doppelt verkettet) verwaltet

Tabellen

Tabellen (Hashtabellen) werden zur Speicherverwaltung verwendet

Manuelle Verwaltung

Speicherbelegung explizit vom Programmierer und teils vom Compiler auszuführen

Vorteile

- ➤ Optimale Speicherbelegung (tatsächlich naiv und falsch) und Performanz/Effizienz

Nachteile

- ➤ Speicherlecks (nicht mehr benötigter Speicher wird nicht frei gegeben)
- ➤ Mehrfachfreigabe (Inkonsistenz der Speicherverwaltung)
- ➤ Benutzung von Speicherobjekten nach Freigabe

Automatische Verwaltung

Die Freigabe von Speicherbereichen wird automatisch durch einen **Garbage Collector** durchgeführt. Dieser bestimmt auch automatisch ob Speicherobjekte noch benötigt werden.

Vorteile

- ➤ Speicherlecks werden vermieden
- ➤ Keine Mehrfachfreigabe (Inkonsistenz der Speicherverwaltung)
- ➤ Keine Benutzung von Speicherobjekten nach Freigabe

Nachteile

- ➤ Nicht optimale Speicherbelegung und schlechtere Performanz/Effizienz

Garbage Collection

- ➤ Wie soll der GC heraus finden ob ein Objekt noch benötigt wird?

1. Referenzzähler wird jedem Objekt hinzugefügt und bei jeder neuen Referenzierung erhöht (Allokation). Ein Objekt mit Referenzwert Null kann frei gegeben werden.

2. Der GC rechnet Objektabhängkeiten aus und markiert Objekte die freigegeben werden können

➤ Beispiel von Referenzen und Sichtbarkeit (Programmkontext)

local $v_1 = \{ x = 100, y = 200 \}^1 \quad [o_1.ref = 1]$
local $v_2 = \{ pos = v_1,\ color = "red" \}^2\ [o_1.ref = 2,\ o_2.ref = 1]$
function $draw\ (v_3)$
  local $v_4 = \{ pos = o.pos,\ color = "white" \}^3\ [o_3.ref = 1,\ o_1.ref = 4]$
  local $v_5 = \{ pos = \{x = v_3.pos.x, y = v_3.pos.y\},\ color = "black" \}^4\ [o_4.ref = 1,\ o_1.ref = 5]$
  $draw(v_4)$
  $draw(v_5)$
  $[o_3.ref = 0,\ o_4.ref = 0,\ o_5.ref = 0,\ o_1.ref = 3,\ o_2.ref = 2]$
end
$draw(v_2)\ [o_2.ref = 2,\ o_1.ref = 3]$
$v_2 = nil \quad [o_2.ref = 0,\ o_1.ref = 1]$
$v_1 = nil \quad [o_1.ref = 0]$

GC: Referenzzählung

➤ Wenn ein Objekt erzeugt wird, wird der Referenzzähler auf Null gesetzt

➤ Jede weitere Referenzierung erhöht den Zähler um eins (also bereits das zuweisen eines Objektes an eine variable!)

➤ Wird eine Referenzierung vom Objekt entfernt (explizit → `=null`, oder implizit durch Entfernen des referenzierenden Objektes), dann wird der Zähler um eins erniedrigt

➤ Ist der Zähler wieder Null, dann wird der Speicher freigegeben

Nachteile

1. Zyklische Referenzen können nicht aufgelöst werden:

```
local o1 = { x=100,y=100 }
local o2 = { prev=o1 }
o1.next = o2
```

2. Daher werden u.U. nicht alle Objekte freigegeben → Speicherlecks

3. Der Referenzzähler belegt selber Speicher (gehört zum Objekt). Bei low-resource Plattformen wird z.B. eine Datenwortbreite von 16 Bit für den Zähler verwendet, und der maximale Referenzwert ist 65536 (jerryscript ist so ein Kandidat)!
4. Allokation ist langsamer (verlangsamt Zuweisung)

Vorteile

1. Einfach zu implementieren
2. Freigabe ist effizient (wenig Rechenaufwand)

GC: Mark & Sweep

➤ Bestimmte Objekte sind direkt zugänglich, und es muss herausgefunden werden welche Objekte erreichbar sind → Graphensuche

➤ Es gibt einen Stammsatz von Speicherplätzen im Programm, dessen Objekte bekanntermaßen erreichbar sind.

Mark-and-Sweep verläuft in zwei Phasen

Markierungsphase

➤ Erreichbare Objekte suchen:

- ❑ Der Stammsatz wird zu einer Arbeitsliste hinzugefügt
- ❑ Solange die Arbeitsliste nicht leer ist wird ein Objekt von der Liste entfernt. Wenn es nicht markiert ist, markiere es als erreichbar und alle Objekte die davon aus erreichbar sind werden zur Arbeitsliste hinzugefügt.

Sweepphase

➤ Für alle belegten Objekte:

- ❑ Ist das Objekt nicht markiert so lösche es (Speicher freigeben)
- ❑ Ist das Objekt markiert, lösche die Markierung

Vorteile

1. Auch zyklische Referenzen können über die Arbeitsliste aufgelöst werden
2. Alle Objekte können frei gegeben werden
3. Allokation schnell

Nachteile

1. Rechenintensiv und langsam!!

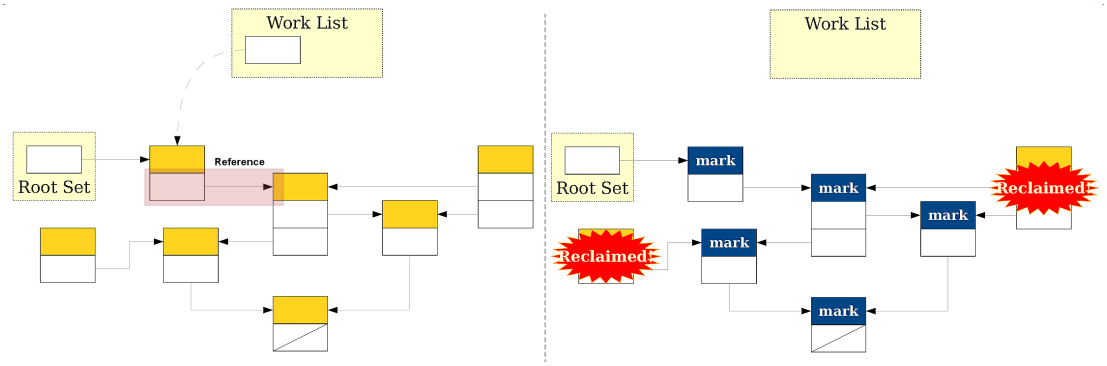


Figure 41. Beispiel eines M&S Durchlaufes mit anschließender Freigabe von nicht mehr erreichbaren == benötigten Objekten

5.10. Multitasking

➤ Auf einem Rechnerknoten gibt es eine Vielzahl von Aufgaben die bearbeitet werden müssen:

- ❏ Sensorverarbeitung
- ❏ Kommunikation
- ❏ Steuerung

➤ Zum Teil müssten Aufgaben parallel verarbeitet werden → Multitasking → Parallelisierung → Nur bedingt möglich → Task Scheduling

5.11. Echtzeitverarbeitung

➤ Echtzeitverarbeitung bedeutet die Ausführung eines Tasks innerhalb eines vorgegebenen Zeitintervalls $[t_0,t_1]$

- ❏ **Soft Realtime**: Zeitüberschreitung bei weicher Echtzeitanforderung → Tolerierbar
- ❏ **Hard Realtime**: Zeitüberschreitung bei harter Echtzeitanforderung → Systemfehler

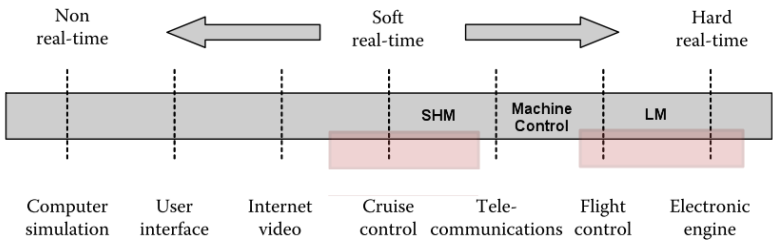


Figure 42. Echtzeitspektrum: Wiche und harte Echtzeitanforderungen bei unterschiedlichen Applikationen [B]

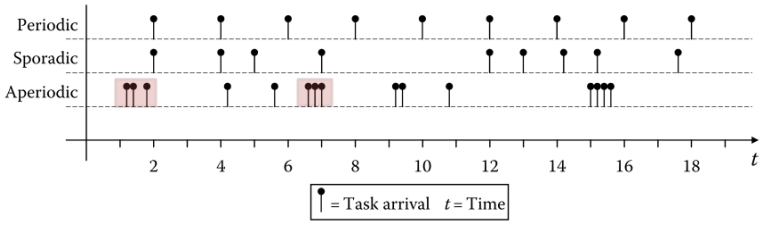


Figure 43. Periodische, sporadische, und aperiodische Tasks [B]

Preemption

- ➤ Wenn schon nicht alles Tasks gleichzeitig verarbeitet werden können, dann wenigstens die wichtigsten vorrangig ausführen
- ➤ Das erfordert aber:
  - ❑ Festlegung einer Priorität
  - ❑ Bei Echtzeitanforderungen die Unterbrechung von unterlegenen (kleiner Priorität) Tasks

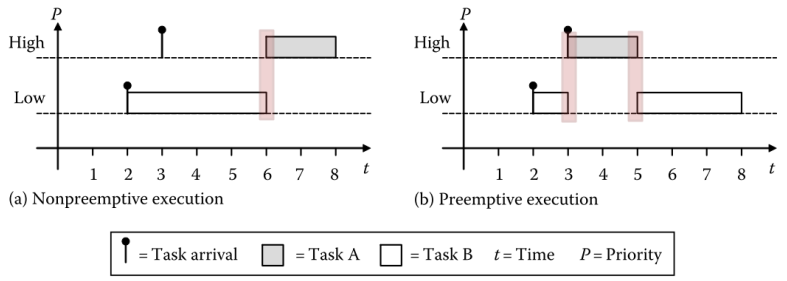


Figure 44. Unterschied preemptive und nicht preemptive Ausführung von Tasks [B]

5.12. Parallele Datenverarbeitung und Prozesse

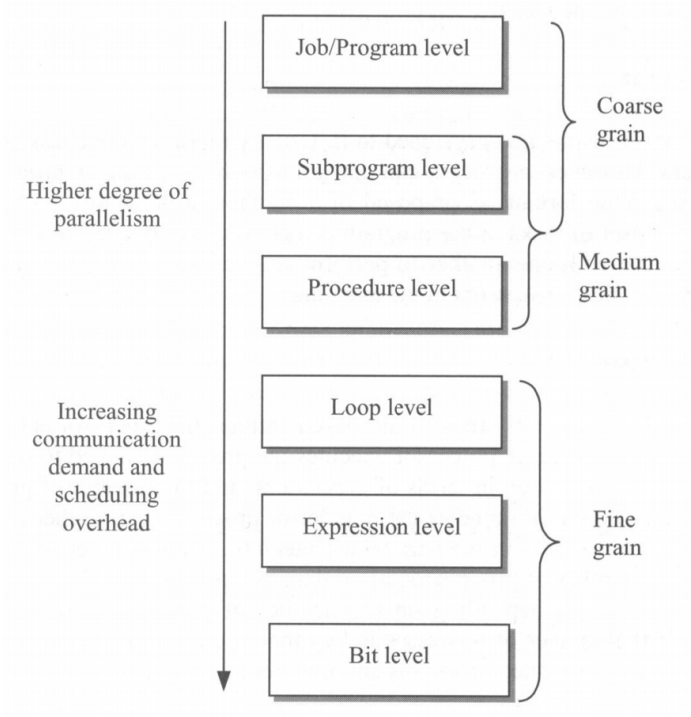


Figure 45. Ebenen der parallelen Datenverarbeitung

- Ein Prozess ist die dynamische Ausführung eines Programms
- Der Zustand (Mikrobene) eines Prozesses ist durch den Datenzustand aller (privaten) Prozessvariablen, globaler Variablen, und dem Kontrollzustand

(Instruktion) bestimmt

Man unterscheidet folgende Prozessklassen:

Task

Ein Task T ist die kleinste Einheit und i.A. ein Teil einer Berechnung und eines Prozesses. Man unterscheidet fein und grob granulierte Tasks. Wenige Anweisungen und wenig Rechenlast bedeuten feine Granularität. Parallele Datenverarbeitung wird durch konkurrierende Ausführung von Tasks auf verschiedenen Prozessoren erreicht.

Prozesse

Der eigentliche Prozesse ist von allen anderen Prozessen stark entkoppelt und gesichert (lose Kopplung). Die Erzeugung eines Prozesses ist rechenintensiv → schwergewichtige Prozesse. Prozesse können parallel ausgeführt werden.

Threads

Leichtgewichtige Subprozesse eines Prozesses mit mittlerer Kopplung (Datenaustausch zwischen Threads ist direkt möglich). Die Erzeugung eines Threads ist weniger rechen- und speicherintensiv. Threads können parallel ausgeführt werden.

Fibers

Sehr leichtgewichtige Prozesse mit starker Kopplung die in Threads oder Prozessen ausgeführt werden. Die Erzeugung von Fibers ist optimal. Fibers werden i.A. nicht parallel ausgeführt und daher sequenzielle und verschachtelt ausgeführt.

Scheduling

Die Ausführung von Prozessen (Threads) kann parallel auf mehreren Datenverarbeitungseinheiten (Prozessoren) geschehen → Parallele Datenverarbeitung

- Häufig ist aber die Anzahl der Prozess N größer als die Anzahl der Prozessoren P
- Dann müssen Prozesse die rechenbereit sind hintereinander ausgeführt werden → Scheduling
- Ein Prozess kann mittels der Operationen *suspend* und *resume* blockiert oder wieder lauffähig gemacht werden

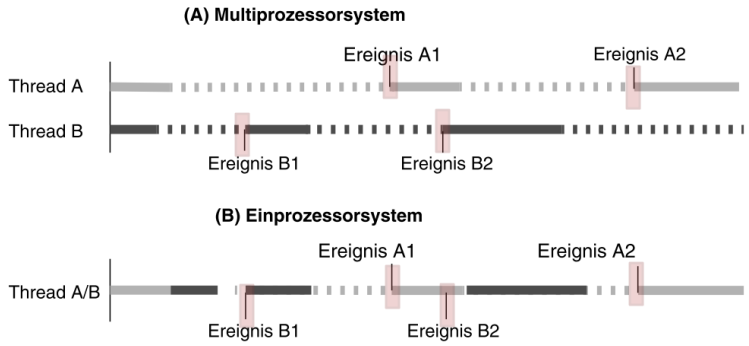


Figure 46. Grobe Sicht des zeitlichen Ablaufs (Ein- und Multiprozessorsystem) [C]

Sequenzielle und parallele Ausführung

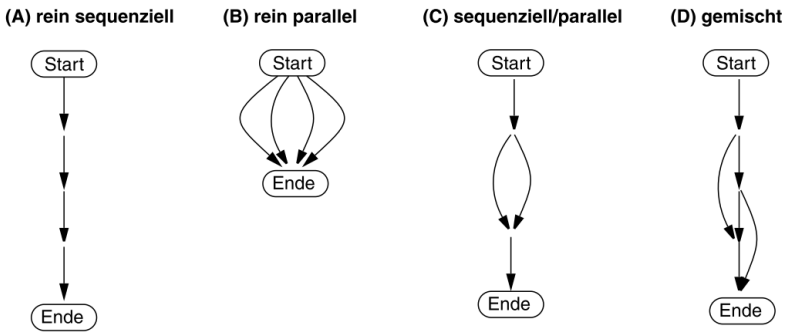


Figure 47. Darstellung paralleler Abläufe (Beispiele) [C]

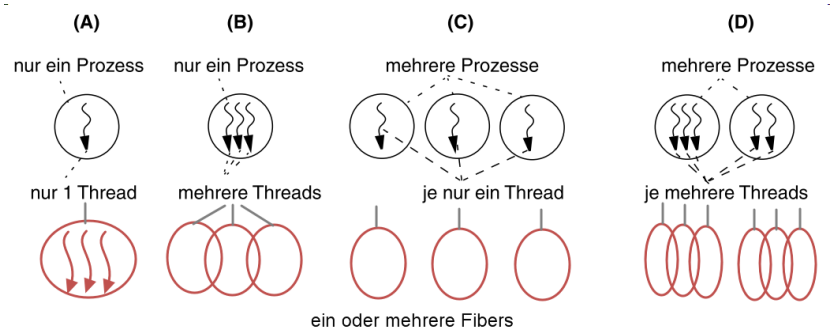


Figure 48. Varianten der Software-Parallelität mit Prozessen, Threads, und Fibers [C]

- Bei der parallelen (und verteilten) Datenverarbeitung soll es eine Beschleunigung der Rechenzeit im Vergleich zum rein sequenziellen Verarbeiten geben
- Aber es gibt auch in parallelen Systemen immer sequenzielle Anteile die die Beschleunigung S reduzieren

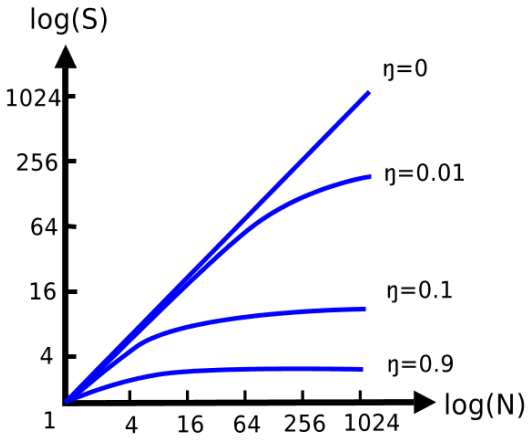


Figure 49. Begrenzung der maximalen Beschleunigung eines parallelen Systems bei einem bestimmten sequenziellen Anteil (in %): Amdahlsches Gesetz (N: Anzahl der Prozesse und Prozessoren)

Prozesszustände

Ein Prozess kann sich in verschiedenen Ausführungszuständen (Makroebene) befinden:

New

Ein Prozess wird erzeugt und der Code und Daten aus einer Datei gelesen

Ready/Ready Suspended

Der Prozess ist rechenbereit (ausführungsbereit), aber wird noch nicht vom Prozessor ausgeführt

Running

Der Prozess wird von einem Prozessor ausgeführt (Ein Prozessor oder Prozessorkern kann immer nur einen Prozess ausführen)

Waiting/Blocked/Suspended

Der Prozess wartet auf ein Ereignis (Daten, Zeitintervall, Ressource frei, usw.). Man unterscheidet u.U. Blockierung und Suspendierung (Details: Art und Weise der Blockierung)

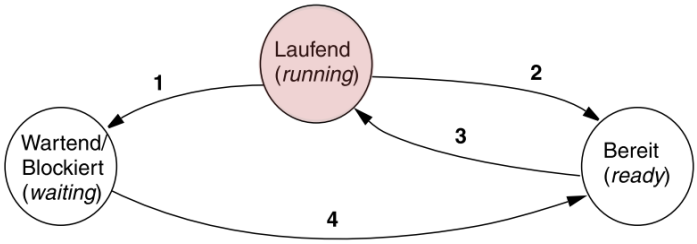


Figure 50. Grundmodell mit drei Zuständen [C]

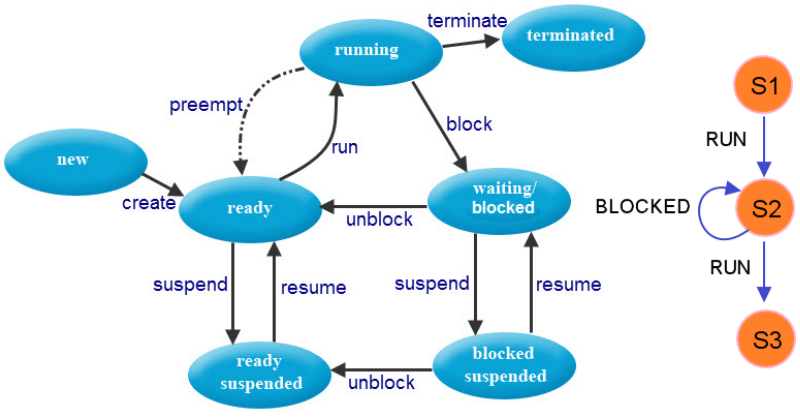


Figure 51. Übergangsdiagramm von Prozesszuständen (Details)

Prozessblockierung

- ➤ Ein rein sequenzielles Programm wird solange schrittweise ausgeführt bis die letzte Anweisung verarbeitet wurde
- ➤ Prozessblockierung bedeutet das Anhalten eines Prozesses an einer bestimmten Stelle im Programmfluss mit späterer Ausführung an genau dieser unterbrochenen Stelle!

5.13. Parallele Datenverarbeitung und VMs

- ➤ Da Virtuelle Maschine i.A. ein automatisches Speichermanagement besitzen ist schwierig oder ineffizient eine VM mit einem gemeinsamen Heap und VM Zustand zwischen Threads zu teilen
  - ❏ Gefahr der Dateninkonsistenz
  - ❏ Alternativ Schutz durch Mutualen Ausschluss bei jeglicher Art der Speicherverwaltung → Aber: Ständig wird Speicher verwaltet (in jeder Anweisung!)
- ➤ Daher wird folgendes parallele Ausführungsmodell bei VMs gewählt:
  - ❏ Jeder Thread führt eine eigen VM aus (mit eignen GC, Heap, und Zustand)
  - ❏ Kommunikation zwischen Threads über einen seriallisierten gemeinsamen Speicher (Store), über den Daten ausgetauscht werden können

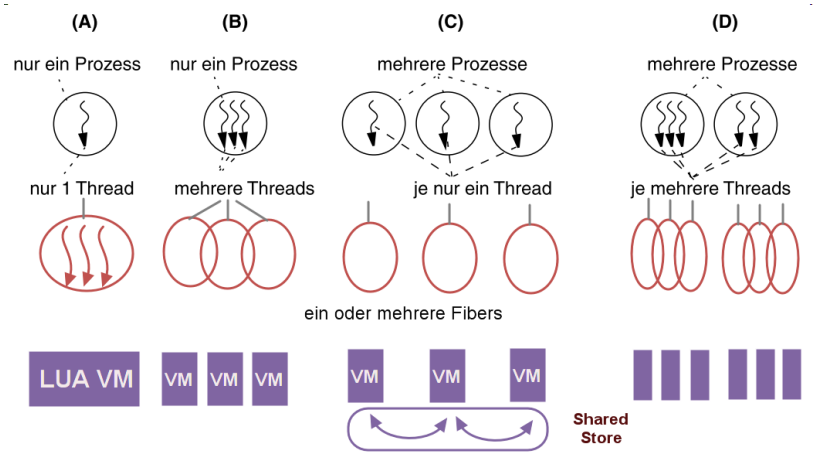


Figure 52. Das parallele Ausführungsmodell der LUAJIT UV VM (lvm)

5.14. Asynchrone Ereignisverarbeitung

- Neben der Parallelisierung der reinen Datenverarbeitung (Berechnung) kann auch eine Partitionierung von Ein- und Ausgabe bzw. der Ereignisverarbeitung erfolgen
- Bekanntes Beispiel: Geräteinterrupts mit einfachen Foreground-Background System mit zwei Prozessen:
  - P1: Hohe Priorität (Ereignisverarbeitung → Interrupthandler → Foreground Prozess)
  - P2: Niedrige Priorität (Berechnung → Hauptprogramm → Background Prozess) mit Preemption durch Ereignisverarbeitung
- Ein- und Ausgabeoperationen eines Programms können **synchron** (blockierend) oder **asynchron** (im Hintergrund verarbeitet und nicht blockierend) ausgeführt werden.

Beispiel JavaScript

- Synchrone Operationen liefern das Ergebnis in einer Datenanweisung zurück die solange blockiert wird bis das Ereignis eintritt(Ergebnisdaten verfügbar sind). Dabei werden zwei aufeinander folgende Operationen sequenziell ausgeführt, und eine folgende Berechnung (nicht ereignisabhängig) erste nach den Ereignissen ausgeführt:

```
x = IO1(arg1,arg2,..)
y = IO2(arg1,arg2,..)
z = f(x,y)
```

- Bei der asynchronen (nebenläufigen) Ausführung von ereignisabhängigen Operationen wird das Ergebnis über eine Callback Funktion verarbeitet. Die IO Operation blockiert nicht. Vorteil: Folgende Berechnungen können unmittelbar ausgeführt werden.

```
IO1(arg1,arg2,..,function (res) x=res end)
IO2(arg1,arg2,..,function (res) y=res end)
z = f(x,y) -- Problem?
```

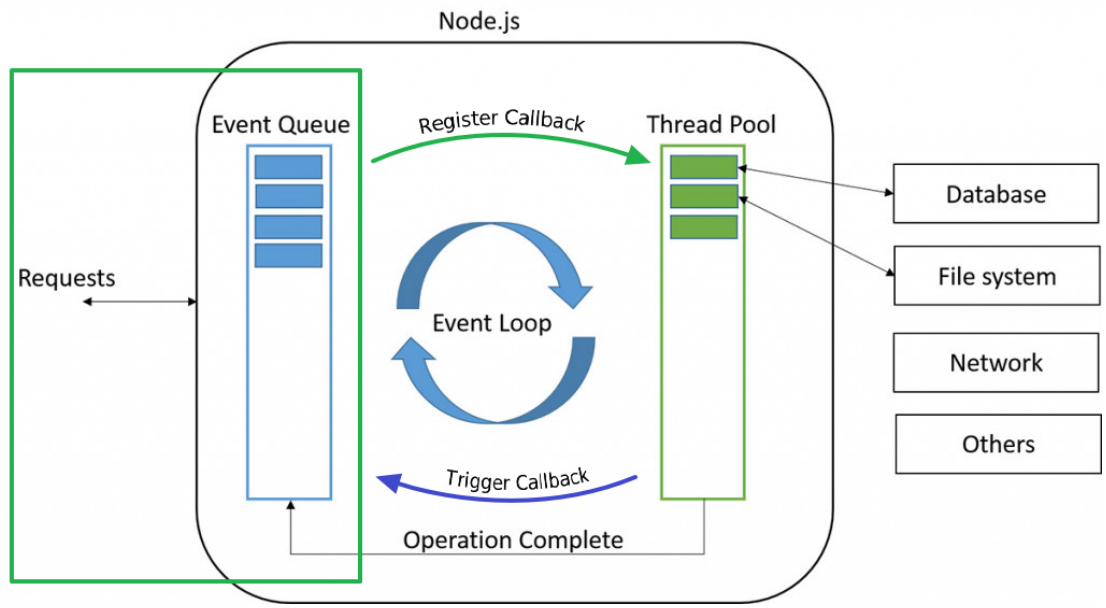


Figure 53. Ereignisbasierte Verarbeitung von asynchronen Operationen in node.js/jxcore: Ein JS Thread verbunden über die Eventloop mit N IO Threads

Synchronisation

- Asynchrone Ereignisverarbeitung mit preemptiven Verhalten benötigt *explizite* Synchronisation (Locks...) zur Atomarisierung von kritischen Bereichen
- Asynchrone Ereignisverarbeitung spaltet den Datenfluss auf und benötigt Daten- und Ergebnissynchronisation über Prädikatsfunktionen oder explizite Synchronisation:

```
local x,y,z
function P(f,x,y,z)
  if x ~= nil and y ~= nil then
    return f(x,y)
  else return z end
end
IO1(arg1,arg2,..,function (res)  x=res; z=P(f,x,y,z) end)
IO2(arg1,arg2,..,function (res)  y=res; z=P(f,x,y,z) end)
```

5.15. Digitale Signalverarbeitung

Digitalisierung als erste Stufe der Signalverarbeitung

➤ Die AD-Wandlung setzt i.A. ein Frequenzfilter (Tiefpass) am Eingang voraus, mit dem der zu erfassende Spektralbereich des Signals begrenzt wird. In der sog. Sample&Hold-Schaltung wird das analoge Signal zeitdiskretisiert, und anschließend mit dem eigentlichen AD-Wandler in einen diskreten i.A. binärkodierten Digitalwert umgesetzt.

Digitale Signalverarbeitung als zweite Stufe

➤ Die Eingangsdaten stehen als kontinuierlicher Datenstrom für die weitere Verarbeitung zur Verfügung. Ein Eingangsdatenstrom wird in einen Ausgangsdatenstrom transformiert.

Erzeugung analoger Signale als dritte Stufe

➤ Die DA-Wandlung kann nur ein quasi-analoges Signal (immer noch zeit- und wertdiskret!) erzeugen. Eine "Zeit- und Wertglättung" findet hier ebenfalls unter Verwendung eines Tiefpass-Frequenzfilters statt

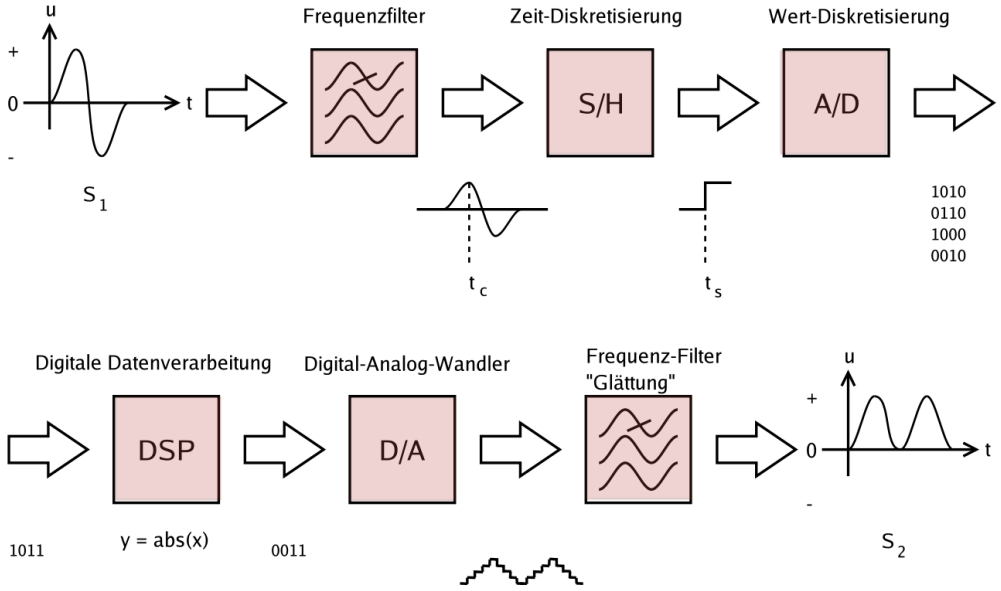


Figure 54. DSP Architektur

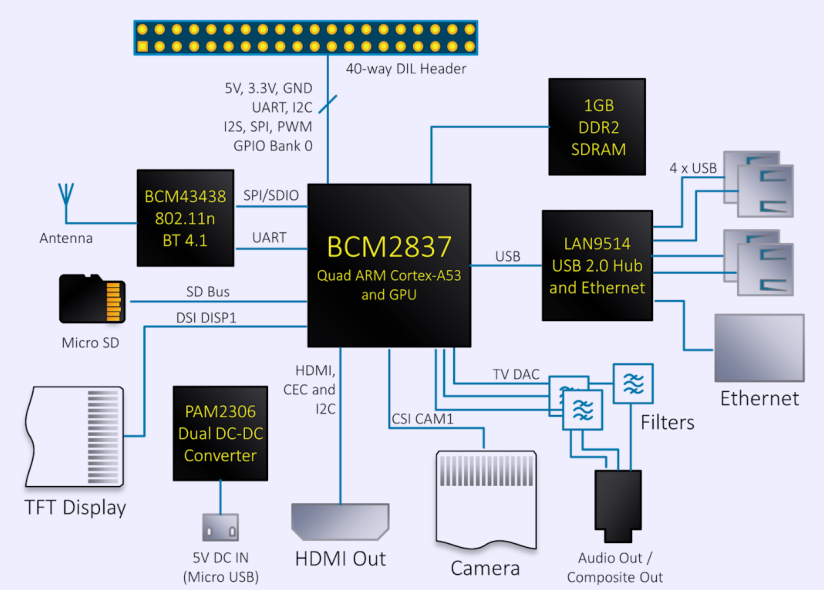
6. Architektur Eingebetteter Rechner

6.1. Eingebettetes System

Komponenten

- Prozessor, i.A. RISC Architektur,
- Nichtflüchtiger Programm- und Datenspeicher ROM (Read Only Memory), i.A. FLASH ROM
- Flüchtiger Programm- und Datenspeicher RAM (Random Access Memory), i.A. DRAM
- Timer und Clock Generator
- IO
  - Verdrahtete serielle Kommunikation mit UART / RS232
  - Verdrahtete serielle Kommunikation mit Ethernet
  - Drahtlose serielle Kommunikation mit Bluetooth (kurzreichweitig, P2P)
  - Drahtlose serielle Kommunikation mit WLAN (kurzreichweitig, P2M)
  - Drahtlose serielle Kommunikation mit WWAN (langreichweitig, P2M)
  - General Purpose IO (Digitale Ports)
  - Analoge Eingänge mit ADC
- Energieversorgung
  - Spannungskonverter
  - Spannungsüberwachung (Watchdog)
  - Energiegewinnung
  - Energiespeicher

Rapsberry PI



xdevs.com/article/rpi3_oc

Figure 55. Raspberry Pi 3 Rechnerarchitektur

Modellierung eines Sensorknotens

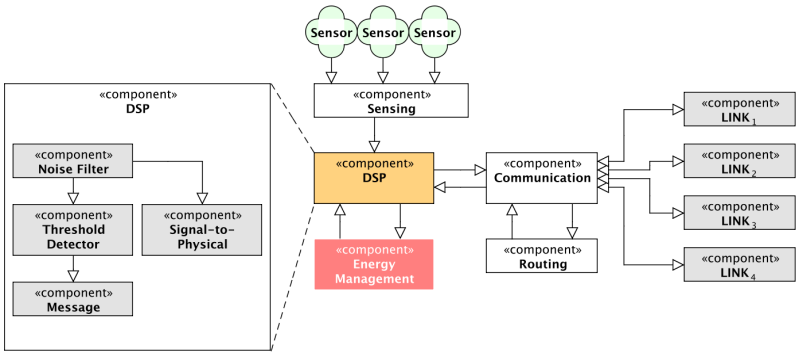


Figure 56. Komponenten eines Sensorknotens (Datenflussarchitektur)

➤ Sensormodell: Ein Sensorknoten ist mit verschiedenen Sensoren verbunden.

I. Dehnungsmessstreifen (elektrische Größe) II.Energiesensor (Ladezustand eines Energiespeichers Ladespannung)

➤ Energiemodell: Jeder Sensorknoten verfügt über einen Energiespeicher, der

I. Über eine zeitlich fluktuierende Energiequelle (Energy Harvester) mit Energie gespeist wird, und der

II. Durch Berechnungs- und Kommunikationaktivität des Sensorknotens entladen wird.

$$E(t) = E_0 + \sum_{\tau=0}^{t} e_{\text{harvest}}(\tau) + \sum_{\tau=0}^{t} e_{\text{computing}}(\tau)$$

$$e_{\text{computing}}(t) = \sum_{i=0}^{n} e_{\text{instruction}}(i)$$

➤ Die Bilanzierung der Energie kann vereinfacht werden: Jede elementare Berechnung des Sensorknotens benötigt die gleiche Energiemenge $e_{\text{instruction}}$.

➤ Energiemanagement muss verwendet werden, um mit der zur Verfügung stehenden Energie $E(t)$ die Aufgaben und Ziele 1. Lokal (Knotenebene) und 2. Global (Netzwerk) zu erfüllen.

Prozessor

Tasks und Prozesse sind softwarebasierte oder virtuelle Einheiten. Ein Prozessor, oder allgemein eine Datenverarbeitungseinheit (Processing Element PE) ist eine hardwarebasierte physikalische Ressource, deren Anzahl und Eigenschaften i.A. zur Laufzeit fest (statisch) sind. Ein oder mehrere Prozesse $\{P_1,P_2,..\}$ sowie Taks $\{T_1,T_2,..\}$ können temporal verteilt einem Prozessor PE zugeteilt und von ihm ausgeführt werden.

Programm

Schrittweise Ablaufvorschrift von Anweisungen $\{A_1,A_2,..\}$

Datenpfad

Der Datenpfad eines Programms stellt alle funktionalen Anteile (mit arithmetischen, relationalen, und booleschen Operation) und den Datenspeicher dar (Register, Speicherblöcke).

```
x := (a+b)/100;
y := f(a,b-1);
```

Kontrollpfad

Der Kontrollpfad eines Programms beschreibt und steuert die schrittweise Berechnung, die sich mit einem Kontrollflussgraphen bzw. Zustandsgraphen darstellen lässt. Der Programmfluss kann von der Entwicklung von Daten abhängen.

```
for i := 1 to 20 do x := x * i; done;
if x < 0 then x := 0; end;
```

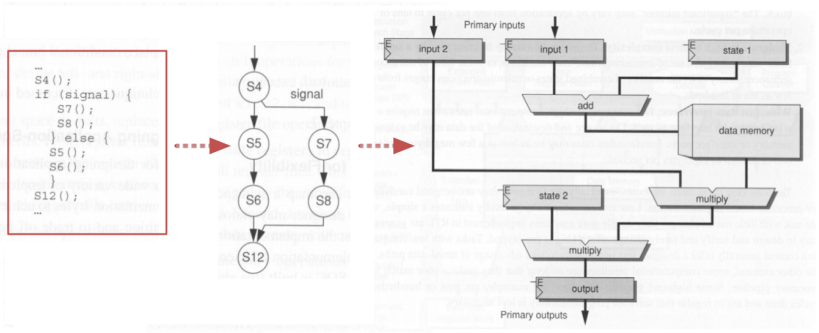


Figure 57. (Links) Programm (Mitte) Kontrollpfad als Zustandsdiagramm (Rechts) Datenpfad auf Registerebene

6.2. Rechnerarchitekturen

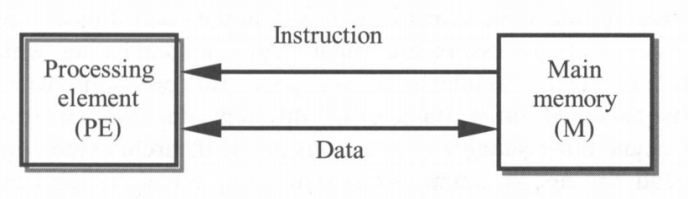
Datenverarbeitung kann prinzipiell

- programmgesteuert mit einem generischen oder anwendungsoptimierten Prozessor [dynamisch], oder
- anwendungsspezifisch [statisch] mit Register-Transfer Logik Architekturen implementiert werden. Hier wird die Maschine zur Entwicklungszeit festgelegt.

Klassifikation von Parallelrechnerarchitekturen kann basierend auf dem Flynn'schen Modell erfolgen. Klassifikation nach

- Datenstrom und
- Instruktionsstrom

zwischen Speicher- und Datenverarbeitungseinheiten.



[Tokhi,2003]

SISD: Single-Instruction Single-Data Stream

➤ Ein von-Neumann Rechner besteht aus einer zentralen PE, dem Mikroprozessor, einem für Daten und Instruktionen gemeinsamen Hauptspeicher, und Ein-/Ausgabeeinheiten.

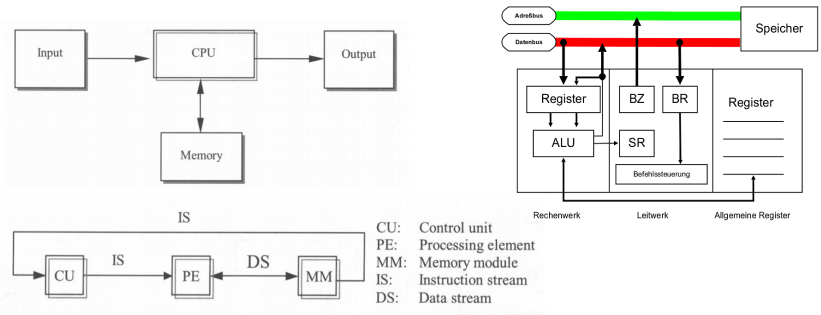


Figure 58. Von-Neumann Rechnerarchitektur und Flynns SISD Modell [PCRT-SPC,Tokhi et al.,2003]

➤ Es gibt nur einen Daten- und Instruktionsstrom, der zeitlich sequenziell ist (Temporales Multiplexing).

➤ Die Ausführung einer Maschinenanweisung ist in mehrere Teilschritte (Phasen) unterteilt:

1. Befehlsholphase
2. Dekodierungsphase
3. Operandenholphase
4. Ausführungsphase
5. Rückschreibephase
6. Neuadressierung

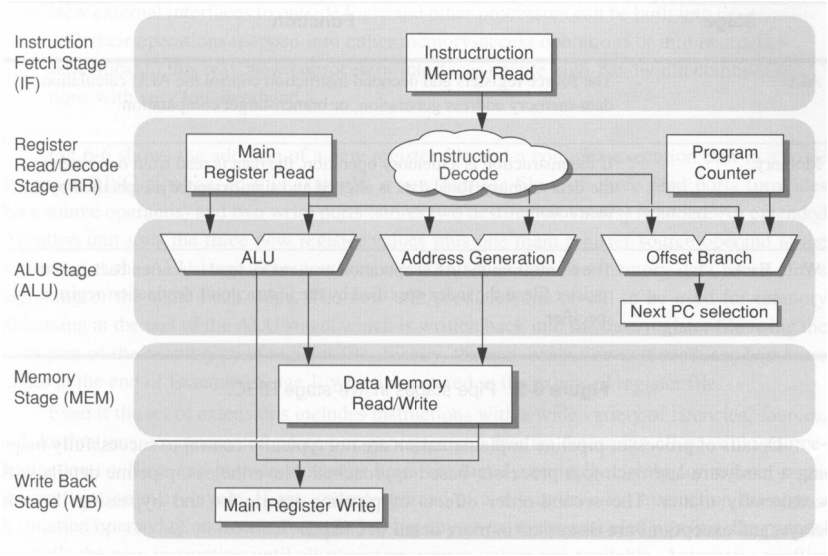


Figure 59. Ausführungsebenen (Stufen) in einem Mikroprozessor [EC-SOC,Rowen,2004]

Harvard Architektur

- ➤ Der von Daten- und Instruktionen gemeinsam geteilte Hauptspeicher ist der Flaschenhals bei der Programmausführung.
- ➤ Aufteilung von Daten und Programmcode in getrennten Speicher beschleunigt die Ausführung von Maschinenanweisungen.
- ➤ Das Leitwerk des Prozessors kann direkt mit dem Codespeicher verbunden werden!

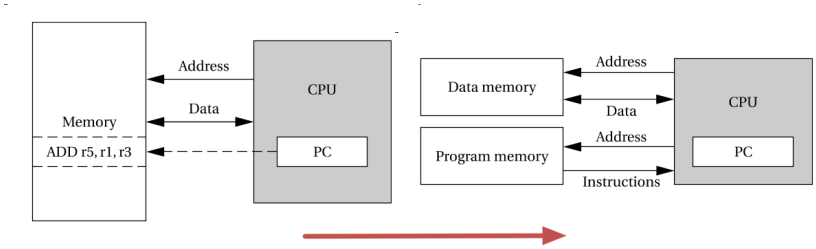


Figure 60. Übergang von zentralem Hauptspeicher zu N-Speicher System [CC,Wolf,2008]

Beispiel: ATMEG32 Mikrokontroller

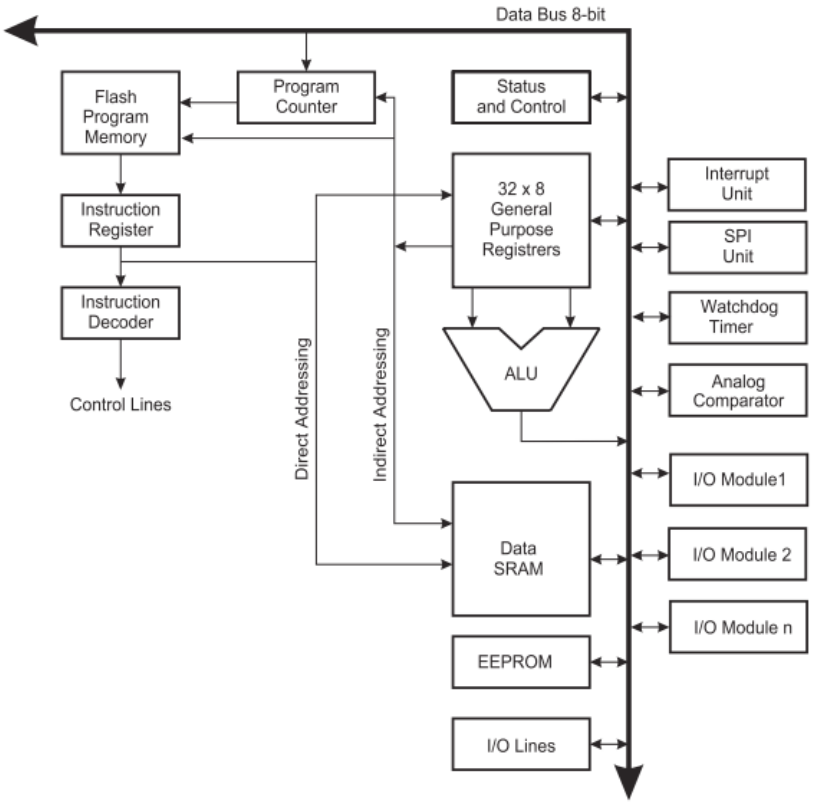


Figure 61. Rechnerarchitektur des Atmel ATMEGA 32 Mikrokontrollers mit Harvardarchitektur [Atmel]

SIMD: Single-Instruction Multiple-Data

- Multiprozessorarchitektur
- Datenverarbeitung (Berechnung) und Steuerung (Kontrollfluss) sind getrennt
- Es gibt einen Instruktionsstrom, aber mehrere Datenströme
- Es wird eine Instruktion auf verschiedenen Daten angewendet
- Anwendung: Verarbeitung von Vektoren und Matrizen, z.B. in der digitalen Bildverarbeitung

➤ RTL Architektur mit Zustandsautomat und RT Datenpfad gehört zur SIMD Klasse!

Bei Digitallogik und Registertransferarchitektur entspricht SIMD der Ausführung mehrerer Datenpfadoperationen (Berechnungen) in einem Zeitschritt

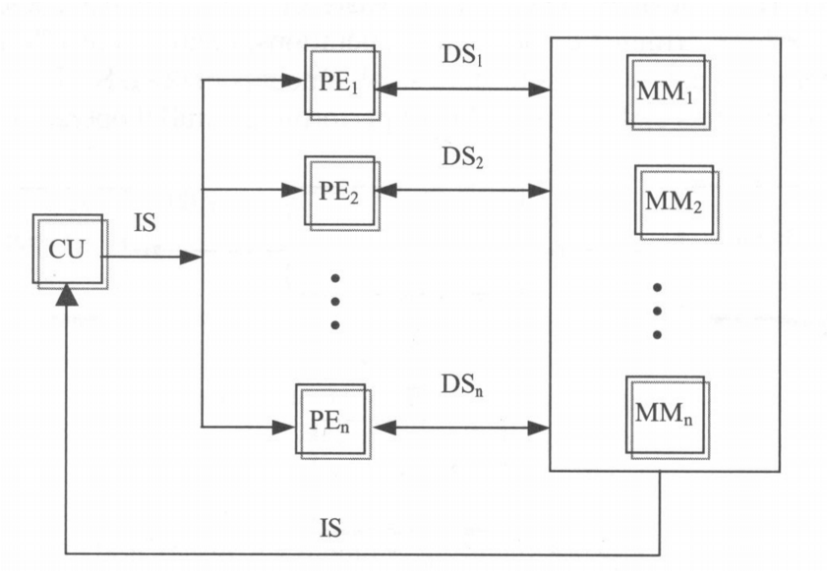


Figure 62. SIMD Model [PCRTSPC,Tokhi et al.,2003]

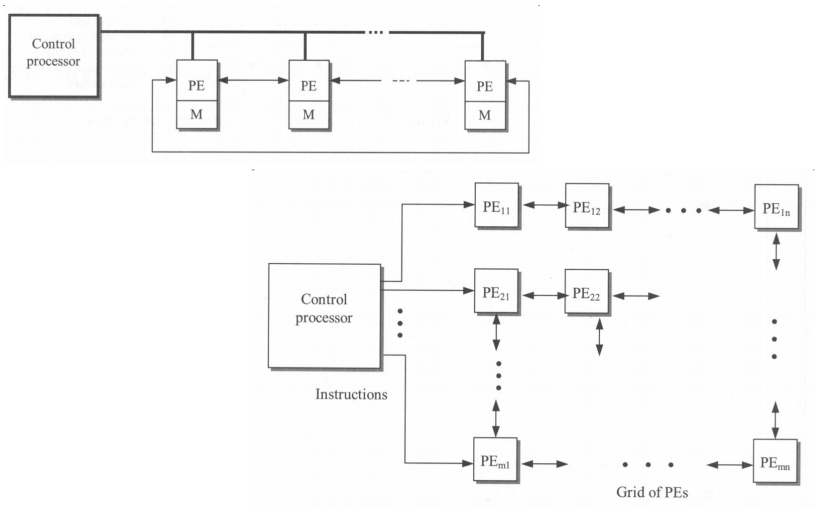


Figure 63. SIMD Architekturen: Vektor- und Arrayrechner [PCRTSPC,Tokhi et al.,2003]

MISD: Multiple-Instruction Single-Data

➤ Es gibt mehrere Kontroll- und Datenverarbeitungseinheiten, die mit mehreren Instruktionsströmen arbeiten und die jeweils mit verschiedenen Anweisungen auf den gleichen Daten operieren.

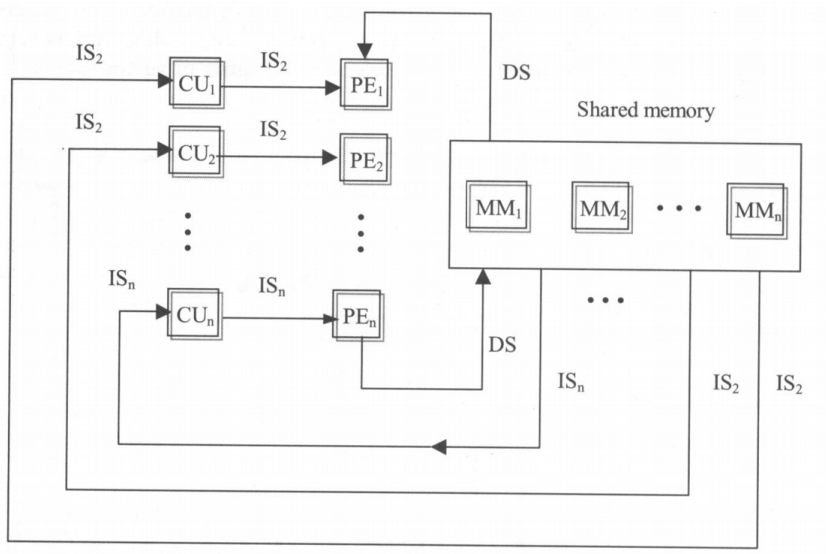


Figure 64. MISD Modell [PCRTSPC,Tokhi et al.,2003]

MIMD: Multiple-Instruction Multiple-Data

- ➤ Mehrere Kontroll- und Datenverarbeitungseinheiten können unabhängig verschiedene Anweisungen mit verschiedenen Daten bearbeiten → Shared-Memory Multiprozessor
- ➤ Alternative System-on-Chip Architektur: Multi-RTL mit multiplen Zustandsautomaten und RT Datenpfaden

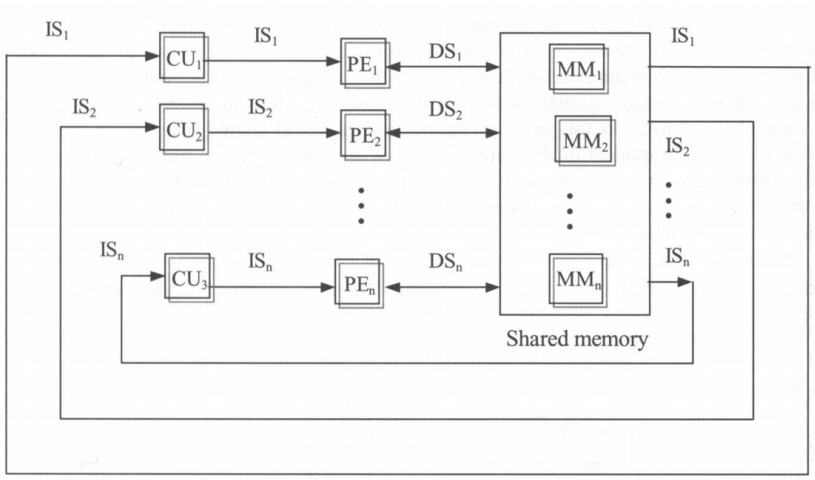


Figure 65. MIMD Modell [PCRTSPC,Tokhi et al.,2003]

Shared-Memory und Dsitributed-Memory Multiprozessor Architektur

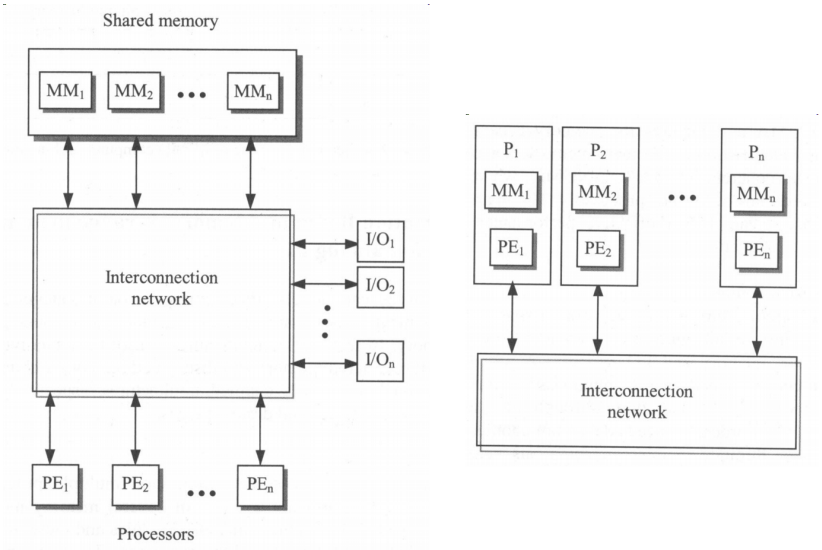


Figure 66. SM (stark gekoppelt) und DM Architektur (schwach gekoppelt) [Tokhi et al.,2003]

Applikationsspezifische Prozessoren

- ➤ Generische Mikroprozessoren sind limitiert durch
  - ❑ fehlende oder eingeschränkte Parallelität im Datenpfad - meist nur eine Berechnung, i.R. eine funktionale Operation mit maximal zwei Operanden, pro Bearbeitungszyklus möglich (Ausnahme bei Pipelining und replizierten ALUs)
  - ❑ eingeschränkte Unterstützung bei Parallelität im Kontrollpfad - nur Pipelining mit Ausführung von mehreren Anweisungen in verschiedenen Phasen/Modulen der Verarbeitungspipeline und einfaches Multi-Threading
  - ❑ Multiprozessor Betrieb ermöglicht nur grob garnulierte Parallelität
  - ❑ Shared-Memory als globales Objekt führt zu erhöhter sequenzieller Ausführung (Mutex Scheduler muss Wettbewerb auflösen).
  - ❑ zentralern Hauptspeicher.

Fazit: schlechte Performanz in Latenz und Datendurchsatz, nur grob granulierte Parallelität.

- ➤ Anwendungsspezifische Mikroprozessoren
  - ❑ erweitern generische Prozessoren mit zusätzlichen Registern, Maschinenbefehlen, und spezialisierten und optimierten Funktionseinheiten.
  - ❑ Es wird unterschieden: erweiterbar und rein applikationsspezifisch
- ➤ Vorteile erweiterbar:
  - ❑ Mikroprozessorkern und Software-Compiler existieren
- ➤ Nachteile erweiterbar:
  - ❑ Nur eingeschränkte Möglichkeiten der Erweiterung, keine Anpassung von Datenwortbreiten, Architekturanpassungen, etc.
- ➤ Vorteile appliaktionsspezifisch:
  - ❑ Verbesserte Möglichkeiten der Optimierung wie Datenwortbreite, Anzahl der Bussysteme, Kommunikationsstrukturen, Architektur, usw.
- ➤ Nachteile applikationsspezifisch:

- ❑ Es existieren keine getesteten oder verifizierten Prozessorkerne und Software-Compiler; Entwurfsprozeß daher fehleranfällig und deutlich langwieriger als bei erweiterbaren Mikroprozessoren

Erweiterbare Mikroprozessoren können bereits gefertigt sein, und sind mit (externen) Erweiterungsschnittstellen ausgestattet, um optimierte Funktionseinheiten wie z.B. Decoder, Encoder, oder kryptographische Funktionen zu implementieren.

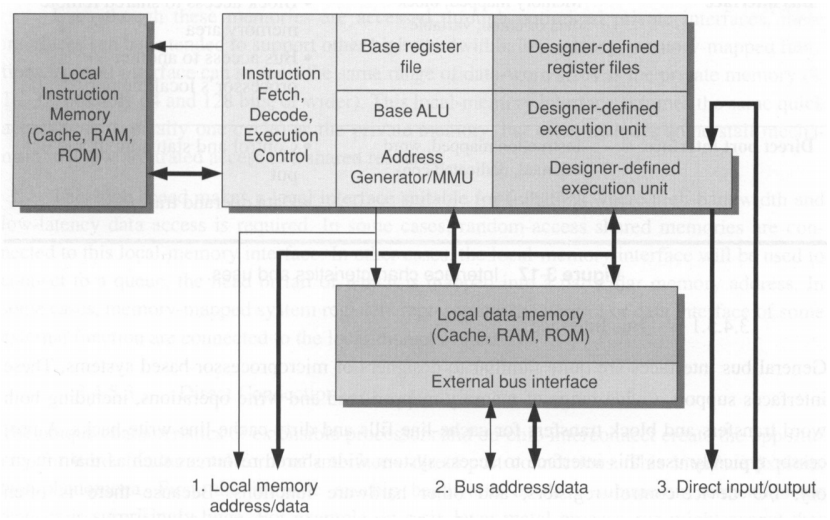


Figure 67. Schnittstellen zur applikationsspezifischen Erweiterung generischer Prozessoren [ECSOC,Rowen,2004]

System-on-Chip und Multiprozessor System-on-Chip Designs

- ➤ Parallelrechnerarchitekturen lassen sich auf einem einzigen Mikrochip als System-on-Chip Design (SoC) integrieren.
- ➤ Ein MPSoC Design besteht aus mehreren Prozessorkernen, Speicher und Speicherkontroller, einem on-Chip Verbindungsnetzwerk (Interconnect), und weiteren Peripheriekomponenten.

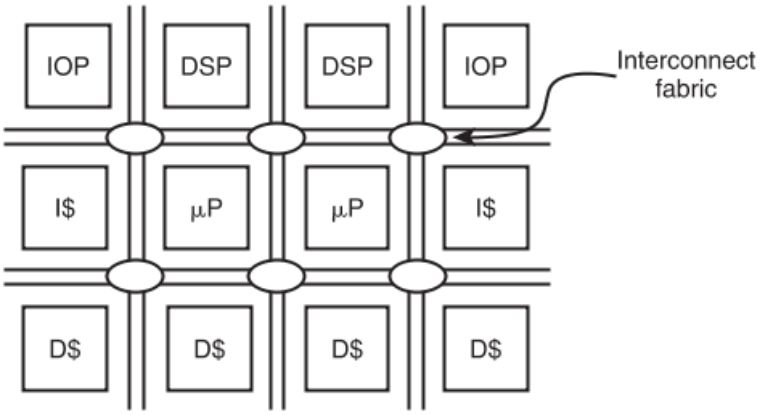


Figure 68. Modell MPSoC. I$, instruction cache; D$, data cache, IOP: input-output processor [MPSOC,Jerraya al.,2005]

6.3. Pipelines

Queues und Pipelines

Parallele Systeme die aus mehreren Prozessoren oder Verarbeitungseinheiten bestehen erfordern Datenaustausch und Synchronisation.

- Queues (First-In First-Out FIFO Spiecher) können zum synchronisierten Datenaustausch zwischen einem Datenproduzenten und Konsumenten (zwei Prozessoren, Prozesse, oder PEs) verwendet werden.
- Synchronisation:
  - Blockierung des **Produzenten** wenn die Queue vollständig gefüllt ist, und
  - Blockierung des **Kosumenten** wenn die Queue kein Datenwort enthält (leer ist).

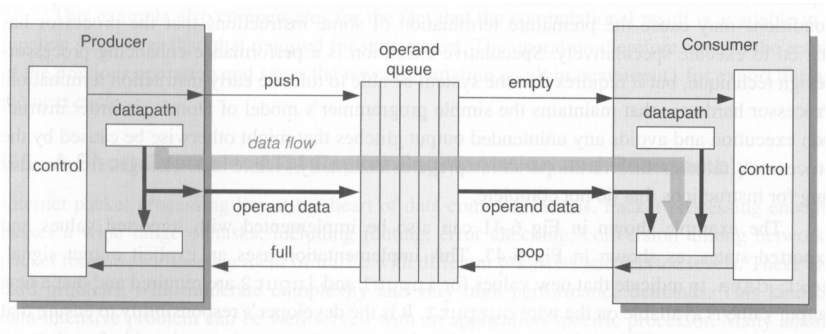


Figure 69. Queues zum synchronem Datenaustausch zwischen Verarbeitungseinheiten und Verwendung in Pipelines

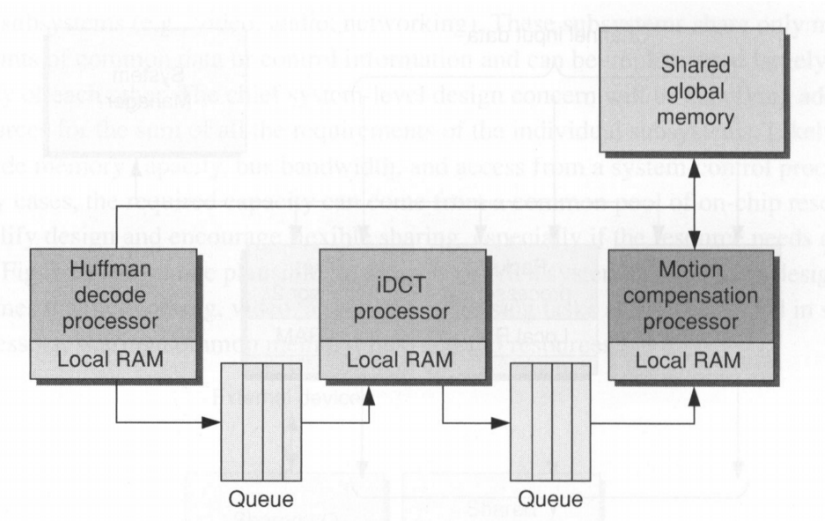


Figure 70. Verwendung von Queues in Pipelines zur Steuerung des Datenflusses

7. Betriebssysteme

7.1. Abstraktion von Rechnern

- ➤ Ein Betriebssystem soll wie eine virtuelle Maschine die technischen Einzelheiten von Hardware von der Software kapseln
- ➤ Ein Betriebssystem soll Abstraktionen und einen einheitlichen Zugriff einführen für
  - ❑ Speicher

- ❏ Prozessor
- ❏ Sicherheit
- ❏ Kommunikation
- ❏ Ein- und Ausgabegeräte
- ❏ Sensoren
- ❏ Aktoren
- ❏ Energie

➤ Betriebssysteme soll eine Kapselung von unterschiedlichen Modulen und Prozessen ermöglichen

- ❏ Ein Fehler in einem Modul oder Prozess A darf nicht andere Module beeinflussen
- ❏ Ein fehlerhaftes Modul oder Prozess sollte neu gestartet werden können ohne dass es zu einem Zusammenbruch des gesamten Systems kommt
- ❏ Ein Modul oder Prozess A darf nicht in der Lage sein ein anderes Modul oder Prozess B zu beeinflussen (Veränderung von privaten Speicher usw.)

➤ Einführung von Privilegien und Sicherheitsstufen mit unterschiedlicher Berechtigung

- ❏ Einschränkung des Speicherzugriffs
- ❏ Einschränkung des Befehlssatzes des Prozessors
- ❏ Einschränkung auf Zugriff von IO Geräten
- ❏ Einschränkung der Prozessausführungszeit

➤ Vergleich der Privilegienebenen eines RISC und eines CISC Prozessors

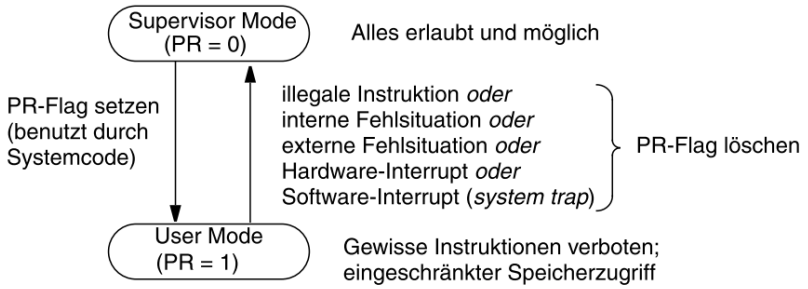


Figure 71. Privilegiensystem des PowerPC-Prozessors [C]

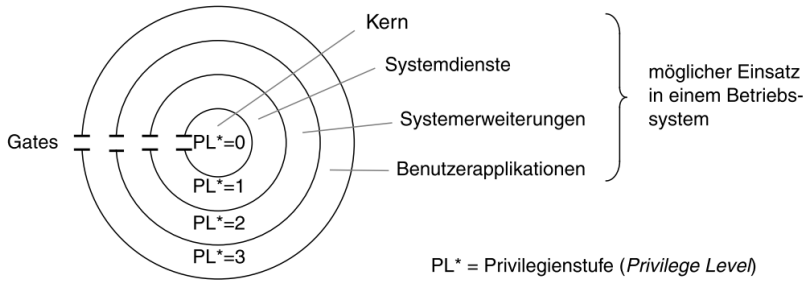


Figure 72. Privilegiensystem des Intel X86 Prozessors [C]

7.2. Betriebssysteme für Eingebettete Systeme

Anforderungen

- ➤ Leichtgewichtig (kleine Code Größe, geringer Speicherbedarf, geringer CPU Bedarf)
- ➤ Kurze Startzeiten
- ➤ Stabilität
- ➤ Effizienter Umgang mit knappen Ressourcen (CPU, Speicher, usw.)
- ➤ Power Management inkl. Sleep Mode (Abschaltung von Geräten)
- ➤ Sicherheit?

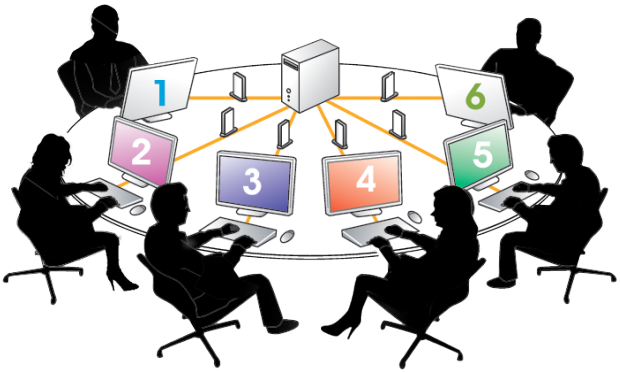
Beispiele

8. Dateisysteme

Fragestellung:
Was ist allen Dateisystemen gemeinsam?

8.1. Dateisysteme und Betriebssysteme

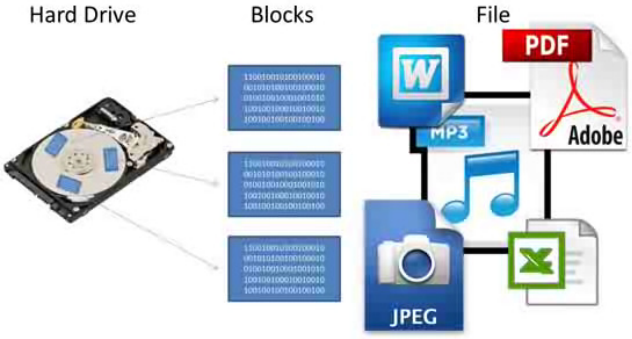
- Dateisysteme sind eng verknüpft mit Betriebssystemen:
  - MS Windows: `FAT` (File Allocation Table), `NTFS` (NT Filesystem)
  - Linux, Android ($\in$ UNIX): `EXT2`,`EXT3` (Extended Filesystem)
  - Solaris ($\in$ UNIX): `ZFS` (Zettabyte Filesystem)
  - Aber unabhängig (CD/DVD): `Joliet`, `RockRidge`, ..
- Ein Dateisystem stellt eine der wichtigen **Abstraktionsmechanismen** dar
- Eigenschaften der Dateisysteme spiegeln Betriebssystemklasse wieder:
  - Einzelnutzer System
  - Mehrbenutzer System
  - Server System
  - Mobiles System
  - Netzwerk System
  - Verteiltes System



8.2. Aufgaben der Dateisysteme

Aufgabe: Organisation und Ablage von beliebigen digitalen Objekten

- Speicherung der Objektdaten auf **Langzeitspeichern** (z.B. Festplatte)
- Die Objekte werden auf verschiedenen Abstraktionsebenen auf binäre **Daten** abgebildet
- Objekte: *Text, Musik, Video, Foto, Zeichnung, Programmcode, ..*
- Jedes Objekt ist durch einen Satz von Attributen gekennzeichnet: *Name, Größe, Art, Einordnung in Containern, ..*



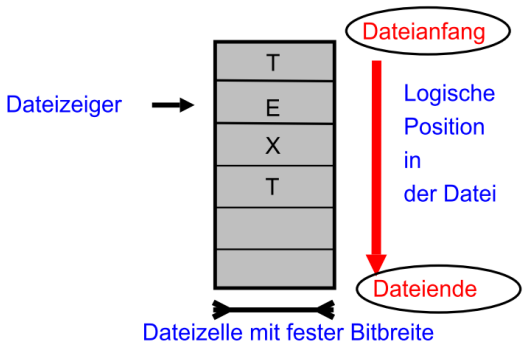
[A]

- Attribute sind **Metadaten**
- Organisation der Objekte in **Containern**
- Ein Objekt ist schließlich nur noch eine Nummer!

8.3. Dateien - Das Linearmodell

➤ Auf der Programmier- und Programmebene sind Dateien eine Folge von Bytes → Array/Tabellenstrukturierung

- ❏ Eine Datei kann gelesen (Lesezugriff) und verändert werden (Schreibzugriff)
- ❏ Es gibt einen Lese- und Schreibzeiger der eine Zelle des Arrays zum Lesen oder Schreiben auswählt




```
C:
fd=fopen("/tmp/x.dat",
   "r+");
n=fread(buf,1,10,fd);
n=fwrite(buf,1,10,fd);
fseek(fd,10,SEEK_SET);
fclose(fd);
```

Figure 73. Logischer Aufbaue einer Datei (links) und programmatischer Zugriff (rechts, C)

8.4. Dateien - Metadaten

➤ Mit einer Datei sind eine Vielzahl von Attribute verknüpft → Metadaten

- ❏ Name
- ❏ Größe (Länge des Dateninhalts)
- ❏ Datum (Erzeugung, Zugriff)
- ❏ Typ (Text, Binär, Programmcode, usw.)
- ❏ Zugriffsrechte (Operationen Lesen, Schreiben, Ausführen, usw.)
- ❏ Eigentümer
- ❏ Sperren (Modifikationsschutz bei Mehrfachzugriff)

➤ Welche Metadaten gespeichert werden hängt vom Dateisystem und Betriebssystem ab!

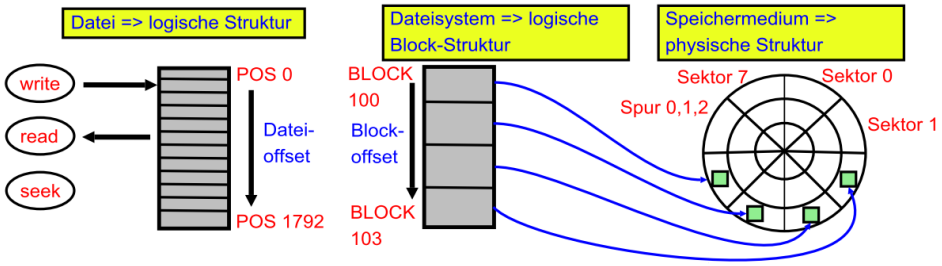
➤ Metadaten werden i.A. in Verzeichnisstrukturen untergebracht. Die Datei selber ist nur durch eine eindeutige Nummer referenziert.

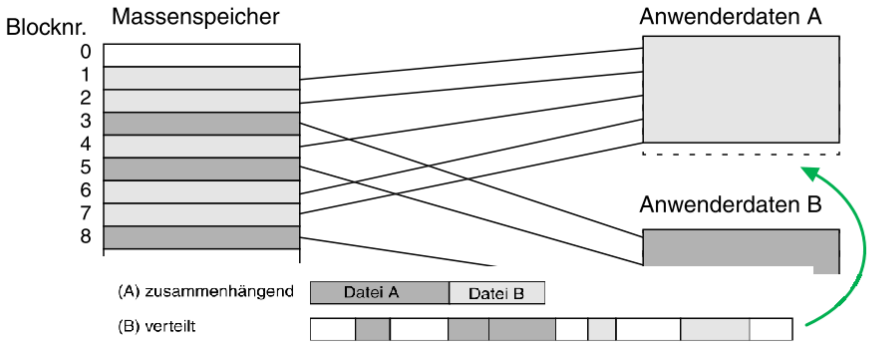
8.5. Physische und Logische Ebene

Blockstrukturierung

➤ Für die technische Speicherung (Festplatte, DVD, USB-FLASH) werden Dateisysteme und die Dateien in Blöcke gleicher Größe zerlegt

➤ Das ermöglich eine vereinfachte und effizientere Ausnutzung des Datenträgers (siehe verteilte Belegung)





8.6. Organistationsstrukturen

Verzeichnisse und Baumstruktur

➤ Die meisten Dateisyssteme organisieren Dateien in Baum- oder Graphenstrukturen

➤ Diese Graphen bestehen aus Knoten: **Verzeichnisse** (Ordner) die als Container für Dateien und weitere Unterverzeichnisse dienen

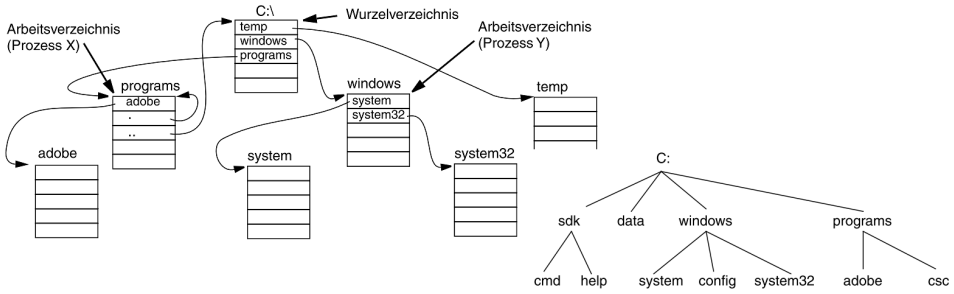


Figure 74. Beispiel von Verzeichnissen und eines Verzeichnisbaums

Pfade

➤ Einzelne Dateien und Unterverzeichnisse sind dabei eindeutig über einem **Pfad** gekennzeichnet.

➤ Ein Pfad beginnt im Wurzelknoten und geht entlang des hierarchischen Verzeichnisweges; **Wurzelknoten**: *Windows* → `C:\`, *Unix* → `/`

➤ Einzelne Elemente des Pfades werden durch ein **Trennzeichen** zusammengefügt: *Windows* →`\`, *Unix* → `/`

Beispiele für Pfade

```
Windows
Pfadname: \Programme\java\j2re1.4.2_06\bin\java.exe
Dateiname: java.exe
Verzeichnispfad: \Programme\java\j2re1.4.2_06\bin\
Verzeichnisname: bin
```

```
Unix
Pfadname: /usr/jre1.6.0_31/bin/java
Dateiname: java
Verzeichnispfad: /usr/jre1.6.0_31/bin
Verzeichnisname: bin
```

8.7. Aufbau und Eigenschaften von Dateisystemen

Fragestellung:
Wie werden die Daten nun konkret organisiert????

8.8. UFS (Unix Filesystem)

Struktur

- Dateien werden in Blöcken im **Blockbereich** gespeichert
- Die Belegung kann verteilt sein
- Das Auffinden von Dateiblöcken geschieht durch Index Blöcke (I-nodes) im **Indexbereich**
- Der Superblock enthält alle relevanten Parameter des Dateisystems (Bereichsgrößen, Datum, Name, Blockgröße, ..)

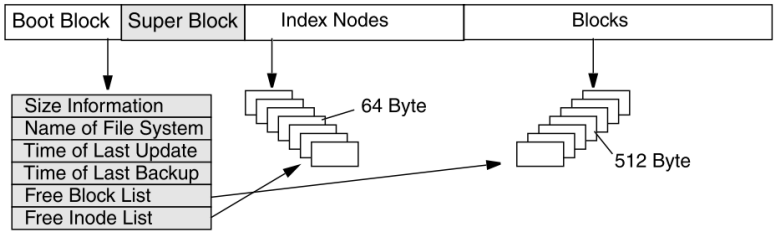


Figure 75. Allgemeine Datenträgerunterteilung des UFS.

I-Nodes

- Eine I-Node ist eine Tabelle mit Blockreferenzen die zu einer Datei gehören.
- Eine I-Node hat eine feste Größe → Begrenzte Anzahl von Datenblockreferenzen!
- Daher Verkettung von vielen I-Nodes für große Dateien

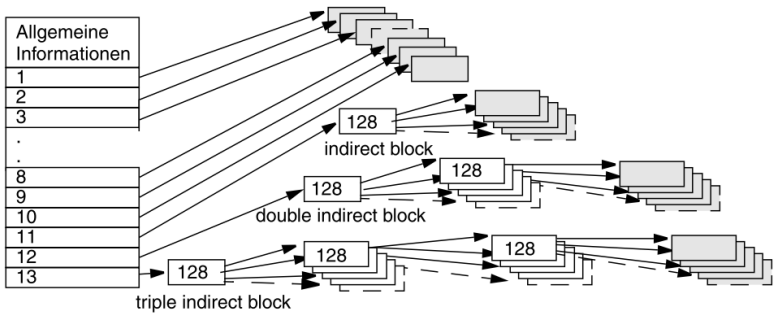


Figure 76. Vernetzung und Tabellenstruktur von I-Nodes

8.9. FAT (File Allocation Table)

➤ Es gibt nur eine einzige globale Tabelle für alle Dateien mit Blockreferenzen die zu einer Datei gehören.

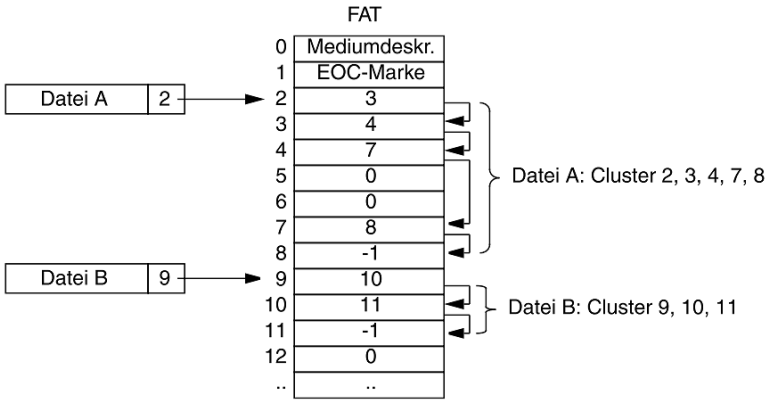


Figure 77. Funktionsweise der FAT-Belegungstabelle

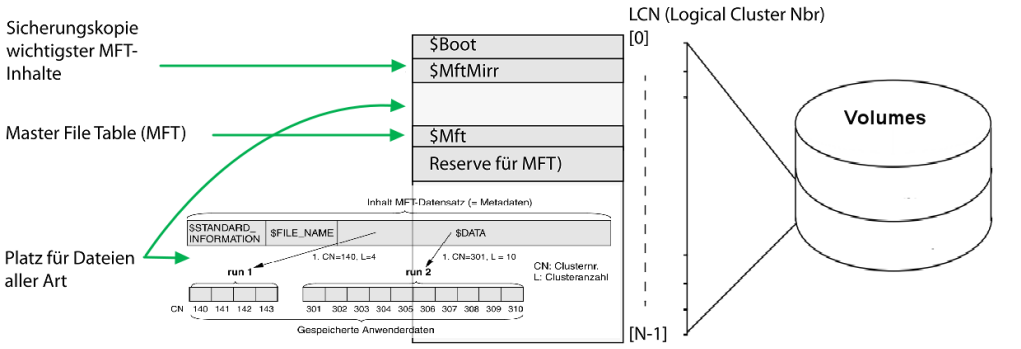
8.10. NTFS (New Technology Filesystem)

➤ Keine Trennung zwischen Index- und Datenbereichen

➤ Das Auffinden von Dateiblöcken geschieht mit Bereichsbäumen (B-trees) → komplex in der Implementierung, aber schnelle Suche!

➤ Alles ist eine Datei, auch Verzeichnisse → sehr viele Metadateien

- Es gibt eine erweiterbare globale Tabelle für alle Metadateien und Dateien: Masterfile Table MFT → Dateisystemjournal
- Eigenschaften: *Wiederherstellbarkeit*, *Zugriffsrechteverwaltung*, *Datenkomprimierung*, *Datenverschlüsselung*



8.11. Metadaten und Eigenschaften

Aufagbe!
(1) Windows: File Explorer öffnen → Nach C:\Windows\System32 wechseln → Detailinformation von Datei cmd.exe
(2) Unix: Terminal öffnen → cd /usr/bin → ls -l sh

Attribut	**DOS/WIN98 FAT**	**Windows NTFS**	**Unix EXT2/UFS**
Dateiname	+	+[1]	+
Namenslänge	11/255[3]	255	255
Case Sens.\UTF	-\-	+\+	+\?
Eigentümer	-	+	+
Erzeugungszeit	+	+	-/+
Zugriffszeit	-	+	+
Modifikationszeit	-	+	+
POSIX Rechte[2]	-	-	+
Links	-	+	+
Max. Dateien	65k-270M[3]/65k[4]	4G/4G[4]	4G/32k[4]
Dateigröße	<4GB[3]	>1TB	>16GB/512GB

[1] Primäres Attribut (direkt gespeichert) [2] Read/Write/Execute: Owner/Group/Others
[3] Abhängig von FAT Version und ggf. Volumegröße [4] Pro Verzeichnis

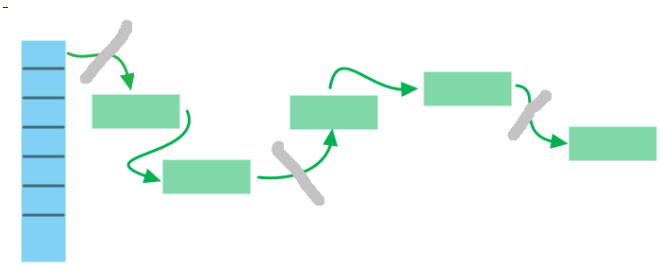
8.12. Fehler und Robustheit

Dateien werden bei den meisten Dateisystemen mit verketteten Listen gespeichert.

Speichern einer Datei bedeutet: 1. Datenblöcke speichern und verketten; 2. Verzeichnis aktualisieren; 3. I-Node Tabelle und Superblock aktualisieren.

Fehler

- ➤ Ist die Kette an einer Stelle beschädigt ist die ganze Datei fehlerhaft und inkorrekt.
- ➤ Wird das Verzeichnis nicht aktualisiert sind die Daten nicht auffindbar
- ➤ Ist das Verzeichnis ebenfalls eine Kette kann bei einer Beschädigung das gesamte Verzeichnis unbrauchbar werden
- ➤ Ist der Superblock beschädigt ist das gesamte Dateisystem unbrauchbar



8.13. Journaling, Replikation und Redundanz

- Um nach einem Crash das Dateisystem wieder schnell und konsistent herstellen zu können wird bei vielen modernen Dateisystemen ein **Journal** geführt.
- **Replikation** und Synchronisation von Superblöcken (der Zugang zum Dateisystem)
- **Replikation** von Dateisystemen (Synchronisierte Kopien auf mehreren physischen Datenträgern)
- **Backups**!!!
- Aufbau des Dateisystems selber kann verbessert werden um Inkonsistenzen zu vermeiden!
  KISS: Keep It Simple and Safe!
- Auswahl des richtigen Dateisystems!

8.14. Dateisysteme für Eingebettete Systeme

Anforderungen

- Robustheit
  - Bei plötzlichen Abbruch durch z.B. Energiemangel
  - Störeinflüsse (elektromagnetisch, Schwankungen der Versorgungsspannung)
- Energieeffizienz
  - Minimierung der Schreib- und Lesezugriffe (administrativer Overhead!)

- ❏ Minimierung des Ressourcenbedarfs (Minimierung der administrativen Daten wie I-Nodes)

➤ Anpassung an physische Speichermedien

- ❏ Flash EEPROM Speicher hat begrenzte Anzahl von Schreibzyklen (1000-100000)
- ❏ Anpassung der Blockgröße und Schreiben nur von vollständigen Blöcken → Flash EEPROM kann keine einzelnen Speicherzellen löschen (zurücksetzen), nur ganze Blöcke

➤ Journaling ja! Aber: siehe vorherigen Punkt → Besser dann nicht!

➤ Redundanz (Replikation von kritischer Bereichen)

8.15. Zusammenfassung

➤ Dateisysteme sind wesentlicher Bestandteil von Betriebssystemen: Service

➤ Dateisysteme speichern Dateien und bieten eine Organisationsstruktur mit Verzeichnissen und Bäumen

➤ Es gibt unterschiedliche Repräsentation von Dateien: Linearmodell; Blockmodell; Physisches Modell (Datenträger)

➤ Es gibt eine große Vielzahl von Dateisystemen die sich unterscheiden durch:

- ❏ Speicherung der Dateien (Größe, Verteilung, Effizienz, Performanz)
- ❏ Verwaltung von Verzeichnissen
- ❏ Metadaten (Dateiattribute)

➤ Die Dateisysteme von UNIX und Windows unterscheiden sich deutlich auf der Implementierungsebene; Der Service ist aber ähnlich!

➤ Verkettung und Indextabellen sind wesentliche Datenstrukturierung

9. Programmierung von Eingebetteten Systemen

9.1. Programmierkonzepte

Funktionale Programmierung

➤ Das Programm besteht aus einer Menge von Funktionen

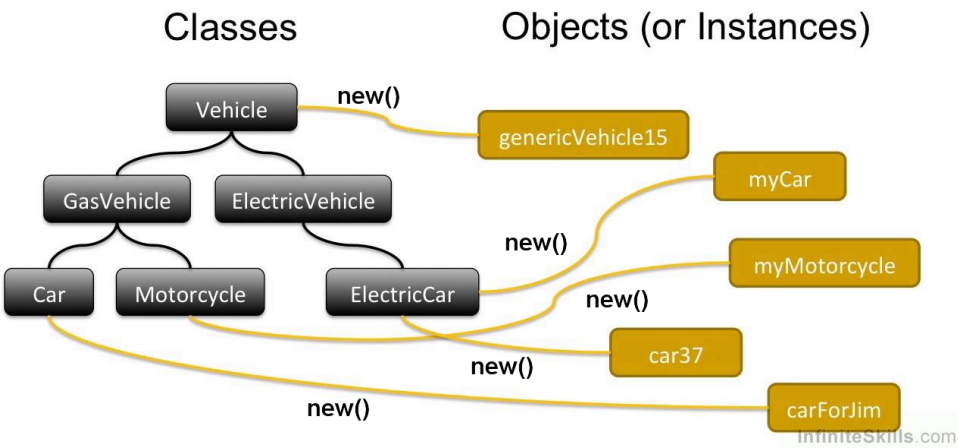
- Drei Konzepte: Funktionsdefinition, Funktionsapplikation, Funktionskomposition
- Eine Funktion berechnet ein Ergebnis von Eingabedaten (Funktionsparameter)
- Es gibt keinen Zustand und keine Speichervariablen

Prozedurale Programmierung

- Funktionen können neben Berechnung auch Aktionen ausführen → Ein- und Ausgabe
- Es gibt einen Zustand und ein Speichermodell
- Es gibt Funktionen (s.o.) und Prozeduren → Kein Rückgabewert, nur Aktion
- Das Programm wird in eine Menge von Funktionen und Prozeduren zerlegt
- Es gibt Datenstrukturen: Arrays (einsortig), Records (mehrsortig)
- Daten werden durch Funktionen und Prozeduren verändert

Objektorientierte Programmierung

- Objekte bestehen aus Daten und Methoden die die Daten verändern (Schreiben) oder weiter geben können (Lesen)
- Objekte werden von Klassen instanziiert
- Eine Klasse definiert die Daten und Methoden
- Klassen können wie Funktionen zusammengesetzt werden
  - Vererbung
  - Superklassen



Datentypen

- Daten (Werte) besitzen in Programmen bestimmte Datentypen:
  - Zahlen (Ganzzahl, reelle Zahl, usw.)
  - Boolesche Werte (true,false)
  - Arrays (von Werten)
  - Textzeichen (char)
  - Zeichenketten (strings, Arrays von Textzeichen)
  - Records (Prozedurale Datenstrukturen)
  - Objekte (Objektorientierte Datenstrukturen mit Methodenzugriff auf Daten)
  - Funktionen
- Kerndatentypen sind in Lua:
  - number (i.A. Fliesskommazahl, 8 Byte)
  - string (Zeichenkette)
  - table (Universelle Tabellenstruktur für Arrays, Records, und Objects)
  - boolean
  - function

- ❏ userdata (nur über definierte Operationen Zugriff möglich)

Werte

- ➤ Zahlen (number): `0, 1, -1, 1.23, ..`
- ➤ Boolesche Werte: `false`, `true`
- ➤ Zeichenketten: `"text"` oder `'text'`
- ➤ Datenstrukturen: `{a=1,b=2, ..}`
- ➤ Arrays: `{1,2,3,4..}`
- ➤ In Lua sind Funktionen Werte erster Ordnung: `function () .. end`

Variablen

- ➤ Eine Variable ist gekennzeichnet durch:
  - ❏ Name (Identifizierer)
  - ❏ Wert (kann undefiniert sein)
  - ❏ Typ (kann polymorph sein)
- ➤ In Lua kann eine Variable dynamisch zur Laufzeit Werte mit unterschiedlichen Datentypen speichern → Polymorphe Variablen und dynamische Typisierung!

Definition 1. (*Definition einer globalen und loakelen Variable*)

var *name* [= *value*]
local var *name* [= *value*]
local var $name_1, name_2, .. = val_1, val_2, ..$

Funktionen

- ➤ Eine Funktion ist gekennzeichnet durch
  - ❏ Name (Identifizierer, optional!)
  - ❏ Parameter (keine, eine, oder mehrere Funktionsvariablen)
  - ❏ Anweisungen (lokale Variablendefinitionen, Berechnung, `return`, Kontrollanweisungen)

- ➤ Funktionen können in Ausdrücken aufgerufen werden (Funktionsapplikation)
- ➤ In Funktionen können weitere Funktionen ausgeführt werden (Funktionskomposition)
- ➤ Funktionen (Prozeduren) können als Anweisung ausgeführt werden
- ➤ In Lua kann eine Funktion auch ohne Namen anonym als Wert definiert werden!
- ➤ Funktionen geben Werte durch die `return v` Anweisung zurück

Definition 2. (*Definition und Applikation von Funktionen*)

function *name* (*parameter*$_1$, *parameter*$_2$, ..)
  Anweisung
  Anweisung
  ..
  [**return** *value*]
end
local function .. *end*
var *name* = **function** (..) .. *end*
name (*value*$_1$, *value*$_2$, ..) -- *Prozeduraler Aufruf*
x = *name* (*value*$_1$, *value*$_2$, ..) -- *Funktionaler Aufruf*

Tabellen

- ➤ In Lua gibt es keine Unterscheidung zwischen Arrays, Datenstrukturen, und Objekten!
- ➤ Es gibt nur Tabellen die aus key-value Paaren bestehen (Schlüssel und Wert) → Hashtabellen
- ➤ Auf einen Tabelleneintrag kann mittes der Schlüsselreferenzierung `t.key` zugegriffen werden (in Ausdrücken lesend oder in datenanweisungen schreibend)
- ➤ Die Definition von Records und Arrays ist sehr ähnlich
- ➤ Bei Arrays ist der Schlüssel aber ein automatisch erzeugter Index 1,2,3, .. → Erstes Element hat immer den Index 1! → Die Länge eines Arrays kann über die `#`*array* Operation ermittelt werden
- ➤ Da Records und Arrays tatsächlich Hashtabellen sind, und der Schlüssel immer eine Zeichenkette ist (auch Index 1!), sind folgende Schreibweisen isomorph: `t.key == t["key"]`

Definition 3. (*Definition und Applikation von Tabellen*)

```
var s = {   -- Record
  a1 = val1,
  a2 = val2,
  ..
}
var a = {   -- Array
  val1,
  val2,
  ..
}
s.a1 = s.a2
a[1] = a[2]
s["a1"] = s["a2"]
```

Anweisungen

➤ Programmanweisungen in Lua:

- ❏ Datenanweisung (Wertzuweisung an Variablen)
- ❏ Bedingte Verzweigung
- ❏ Prozeduraufruf
- ❏ Schleifen

Definition 4. (*Datenanweisung*)

```
variable = expression
```

Bedingte Verzweigungen

Definition 5. (*Bedingte Verzweigung*)

```
if condition then anweisungen_true end
if condition then anweisungen_true
else anweisungen_false end
if .. elseif .. elseif .. else .. end
```

Definition 6. (*Mehrfachauswahl (Extension, kein Standard)*)

case x **of**
  v_1 => *anweisungen* **end**
  v_2 => *anweisungen* **end**
  $v_3, v_4, ..$ => *anweisungen* **end**
  else => *anweisungen* **end**
end

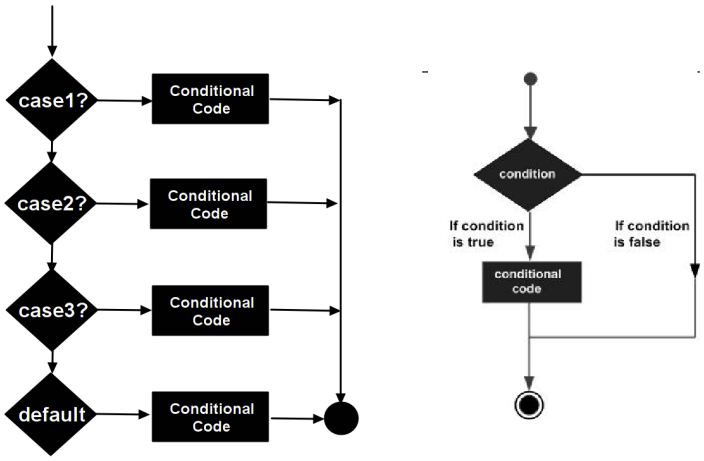


Figure 78. Kontrollfluss von bedingten Verzweigungen

Schleifen

Definition 7. (*Zählschleife*)

for *index* = a, b [, *step*] **do**
  anweisungen
end

Definition 8. (*Bedingte Schleifen*)

while *condition* **do**
  anweisungen
end
repeat
  anweisungen
until *condition*

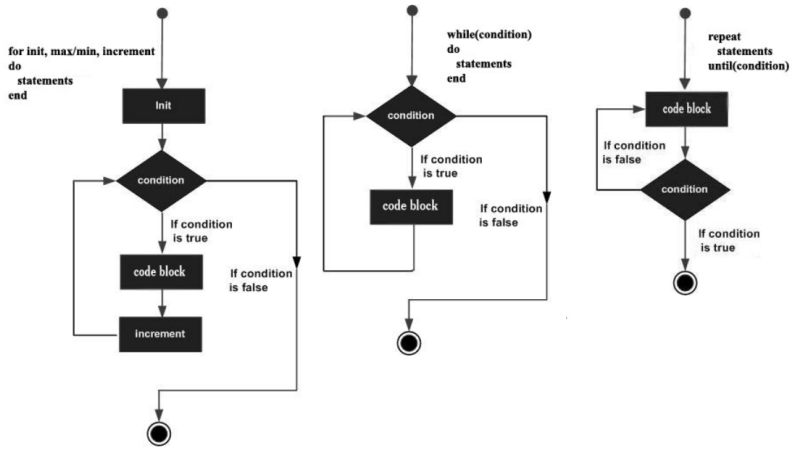


Figure 79. Kontrollfluss von Schleifen in Lua (for, while, repeat)

Turtle Grafik

- ➤ Um das Programmieren zu erlernen ist eine grafische Visualisierung von Programmabläufen und deren Ausgabe sinnvoll um Programmierkonzepte zu erlernen
- ➤ Die Grafik ist das Rechenergebnis!
- ➤ Die Turtle Grafik hat einfache Befehle, die eine symbolische Schildkröte als Dreieck benutzt um auf einer zweidimensionalen Fläche zu zeichnen.
  - ❑ Das Dreieck hat eine Ausrichtung die in Grad angegeben ist [0,360]
- ➤ Dabei kann die Position des Dreiecks absolut (*posn*) oder relativ (*jump*,*move*) verändert werden.
  - ❑ Die relativen Verschiebungen des Dreiecks werden in der aktuellen Richtung des Dreiecks ausgeführt
  - ❑ Die Ausrichtung des Dreiecks kann relativ durch *turn* verändert werden

Definition 9. (*Turtle Grafik*)

move($delta : number$)
turn($degree$)
jump($delta$)
posn($x : number, y : number$)
pnsz($width : number$)
pncl($color : string$)
crcl(x, y, r)
oval(x, y, w, h)

Example 2. (*Iteratives Zeichnen einer Blume*)

```
require "turtle"
for i=1,12 do
  for j=1,12 do
    move(50)
    turn(30)
  end
  turn(30)
  wait(0.1)
end
```

Objekte und Klassen

- Beben der reinen prozeduralen Programmierung mit Funktionen und Records (Tables) kann in Lua auch auf einfache Weise objektorientiert programmiert werden
- Dabei werden Tabellen wieder verwendet um Objekte und Klassen zu implementieren
- Jedoch ist die Unterstützung von Lua nur gering → Verwendung der *class* Bibliothek, die in der *pervasives* Bibliothek enthalten ist (wird von *lvm* direkt geladen)
- Bei der Instanziierung von Objekten werden i.A. Daten des Objekts mit Parameterwerten initialisiert → Definition einer Initialisierungsfunktion erforderlich!
- Ein Objekt kann sich selbst mit *self* referenzieren

Definition 10. (*Definition von Klassen und Instanziierung von Objekten*)

require(*'class'*)
geo = **class**()
function *geo* : *init*(p_1, p_2, p_3) **self**.$a = p_1$; **self**.$b = p_2$; .. *end*
function *geo* : $meth_1$() **self**.xx = .. **end**
function *geo* : $meth_2$() .. **end**
o = *geo* : **new**(v_1, v_2, v_3)
..

Ein Objekt einer Klasse kann sich selber über die Variable *self* referenzieren.

- Neben dem Punktoperator, mit dem allgemein ein Attribut eines Objektes ausgewählt werden kann, kann der Doppelpunktoperator verwendet werden, um Methodenaufrufe automatisch mit der Selbstreferenz auszuführen.
- Bei der Verwendung des Punktoperators muss als erstes Argument immer das Objekt selber übergeben werden.

```
c.m(c,v1,v2,..)
-- Analog kurze Schreibweise
c:m(v1,v2,..)
```

1. Eine neue leere Klasse mit der *class* Funktion erzeugen

```
c = class()
```

2. Eine Initialisierungsfunktion für diese Klasse definieren. Die Initialisierungsfunktion wird bei jedem instanziierten Objekt einmalig aufgerufen.

```
function c:init (p1,p2,..) .. end
```

3. Weitere Methoden der Klasse definieren:

```
function c:m (p1,p2,..) .. end
```

4. Objekete mit der *new* Methode erzeugen

```
local o = c:new(v1,v2,..)
```

10. LUAOS

10.1. Konzept

LUAOS: Lua Operating System

- ➤ LUAOS wird auf einem Rechner (Netzwerkknoten) durch eine Lua VM ausgeführt (*lvm*)
- ➤ LUAOS bietet einen WEB basierten Zugriff auf Rechnerknoten
  - ❏ Skriptausführung über eine Shell mit Monitoring
- ➤ LUAOS:
  - ❏ Ausführung von Skripten in einem Sandkasten
  - ❏ Scheduling und Multithreading
  - ❏ Einfache Sensor und Geräte API
- ➤ Ein Skript wird in einem eigenen gekapselten Thread ausgeführt (mit jeweils eigenen Ausführungszustand und VM)
  - ❏ Ein Skript kann weitere Fibers erzeugen
  - ❏ Ein Skript hat einem maximale durchgehende Laufzeit von einer Sekunde (danach wird das Skript abgebrochen)
  - ❏ Die Ausführung muss daher durch blockierende IO, der *yield*, oder der *sleep* Funktionen unterbrochen werden

10.2. WEB Interface

- ➤ Das LUAOS ist über ein WEB Interface erreichbar

Figure 80. Typische Schnappschuss der LUAOS WEBAPI mit Netzwerkübersicht (links oben) und verschiedenen Fenstern

10.3. LUAOS Architektur

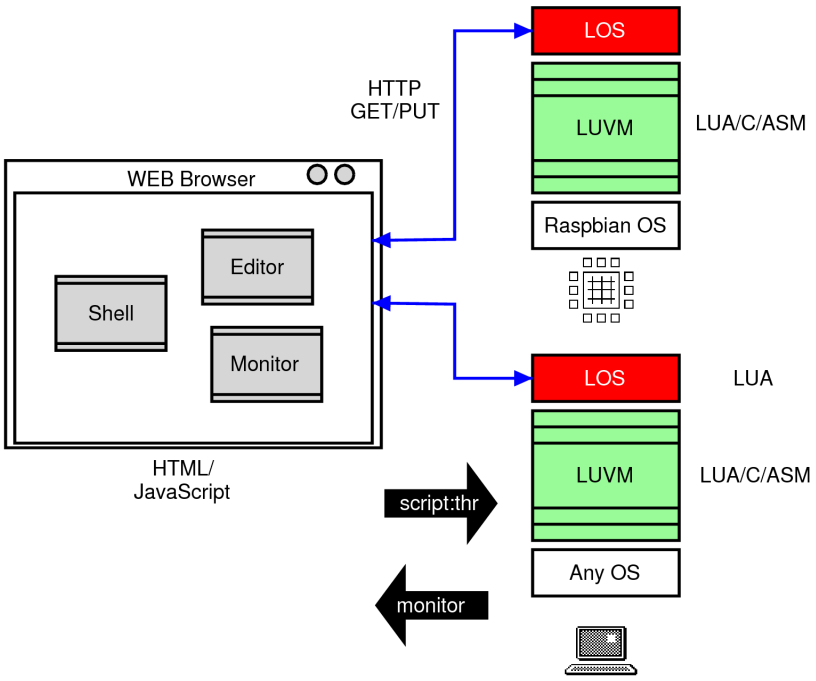


Figure 81. Verteiltes LUAOS System mit WEB API

10.4. LUVM Architektur

➤ Die LUAOS VM (LUVM) besteht aus:

- ❏ LUA Parser → Skripttext wird direkt in Bytecode Instruktionen übersetzt (kein Parserbaum und AST)
- ❏ LUA Bytecode Interpreter (BC Loop ist in Assembler programmiert!)
- ❏ LUA JIT Compiler der zur Laufzeit BC in nativen Maschinencode übersetzt
- ❏ LUA-C API für native Erweiterungen
- ❏ Automatisches Speichermanagement mit Garabage Collector (Nicht inkrementeller Mark & Sweep Algorithmus)
- ❏ Asynchrone IO mit einer UV Loop (libuv, wird auch in node.js verwendet)

- ❏ LUV: LUA-UV API Interface
- ❏ LUV Module: Thread, Fiber, State, Filesystem (FS), Network (Net), Timer, Process, Pipe, Codec (Serialisierung)

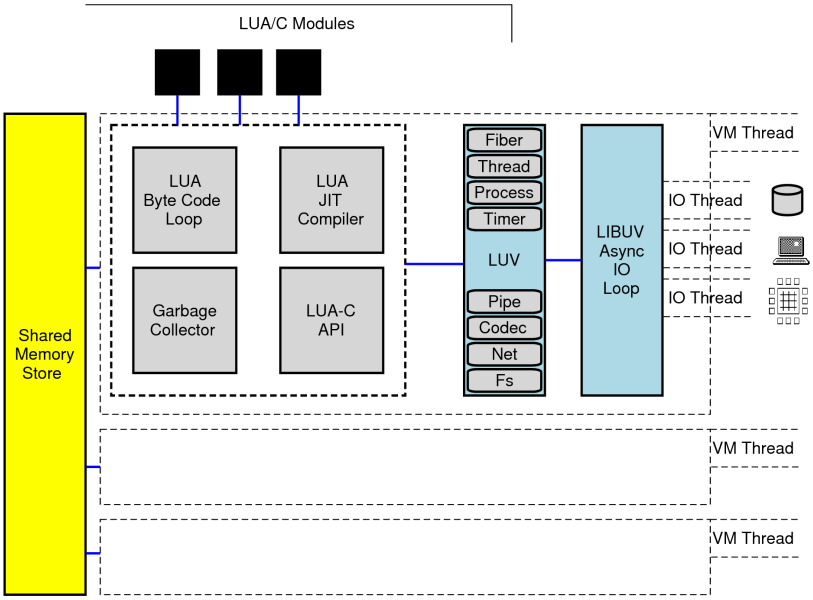


Figure 82. Aufbau der LUAOS VM

10.5. Skript API

function clock() → number
Gibt die aktuelle Systemzeit in Millisekunden zurück

function print(...)
Ausgabe von Text oder Werten auf der virtuellen Konsole

function sleep(millisec)
Der aufrufende Prozess wird für die angegebene Zeit blockiert

function yield()
Blockierender Aufruf des Schedulers und Abgabe der Ausführung (an andere Prozesse) mit späterer Fortführung

Fiber

- ➤ Leichtgewichtige Tasks (werden geschachtelt sequenziell ausgeführt)

function fiber.create (function(*args...*), *args...*) → *fiber*

Erzeugt einen neuen leichtgewichtigen Prozess. Der Prozess ist sofort ausführungsbereit.

function *fiber*:join()

Der aufrufende Prozess wird blockiert bis der Subprozess terminiert ist

HTTP

➤ HTTP Server und Klienten

function http:new() → *http*

Erzeugt eine neue HTTP Instanz

function *http*:get(url, function (data,error))

Ausführung einer HTTP GET Anfrage. Das Ergebnis (Fehler oder Daten) wird durch eine asynchrone Rückruffunktion bearbeitet.

function *http*:put(url, data, function (data,error))

Ausführung einer HTTP OUT Anfrage. Das Ergebnis (Fehler oder Daten) wird durch eine asynchrone Rückruffunktion bearbeitet.

function *http*:service(ipport, function (url,address,params,body) → string)

Installation eines HTTP Service (IP Port *ipport*). Eingehende GET/PUT Anfragen werden durch eine Rückruffunktion bearbeitet. Das Ergebnis der Anfrage wird von dieser Funktion als Text zurück gegeben.

HTML

➤ Ein HTML Text Formattierer

function html:new() → *html*

Erzeugt eine neue HTML Formattierungssnstanz

function *html*:html(*head*:*html* string,*body*: *html* string) → string

Erzeugt den Rahmen einer HTML Seite

function *html*:head() → string

Erzeugt den Kopf einer HTML Seite

function *html*:body(*html* string table) → string

Erzeugt den Inhalt einer HTML Seite

function *html*:table(*head* table, *body* table table) → string Weitere Funktionen: `h1, h2, h3, ol, ul, li, dl, dt, dd`

Timer

➤ Intervalltimer die in Fibers und Skripten benutzt werden können um periodisch Tasks auszuführen

function timer:new() → *timer*
Erzeugt einen neuen Timer.

function *timer*:start(interval:number,period:number)
Startet den Timer mit dem angegebenen Intervall und der Periodenzeit (oder 0). Nach Ablauf des Intervalls wird ein Ereignis signalisiert, auf welches mit der *await* Operation gewartet werden kann.

function *timer*:stop()
Stoppt den Timer. Es werden keine weiteren Ereignisse mehr signalisiert.

function *timer*:wait()
Auf ein Timerereignis warten. Der aufrufende Prozess wird blockiert.

Sensor

➤ Zugriff auf und Implementierung von Sensoren

function sensor:all() → table
Gibt eine Liste alle verfügbaren Sensoren zurück (Liste der Namen)

function sensor:new(sensorclass:string,...) → *sensor*
Erzeugt eine neue Sensorinstanz vom Typ *sensorclass*. Mögliche Klassen sind: “temperature”, “cpu”.

function *sensor*:read() → number|string|table|nil
Gibt den aktuellen Sensorwert zurück.

function *sensor*:calibrate?(...)
Kalibriert den Sensor oder die Sensorfunktion.

Actor

function actor:all() → table
Gibt eine Liste alle verfügbaren Aktuatoren zurück (Liste der Namen)

function actor:new(actorclass:string,...) → *actor*

Erzeugt eine neue Aktuatorinstanz vom Typ *actorclass*. Mögliche Klassen sind: “dotdisp””.

function *actor*:reset(...)

Setzt den Aktuator in einen definierten Anfangszustanf.

function *actor*:set(...)

Setzt eine Steuervariable mit einem neuen Aktuatorwert.

function *actor*:get?(string) → number|string|table|nil

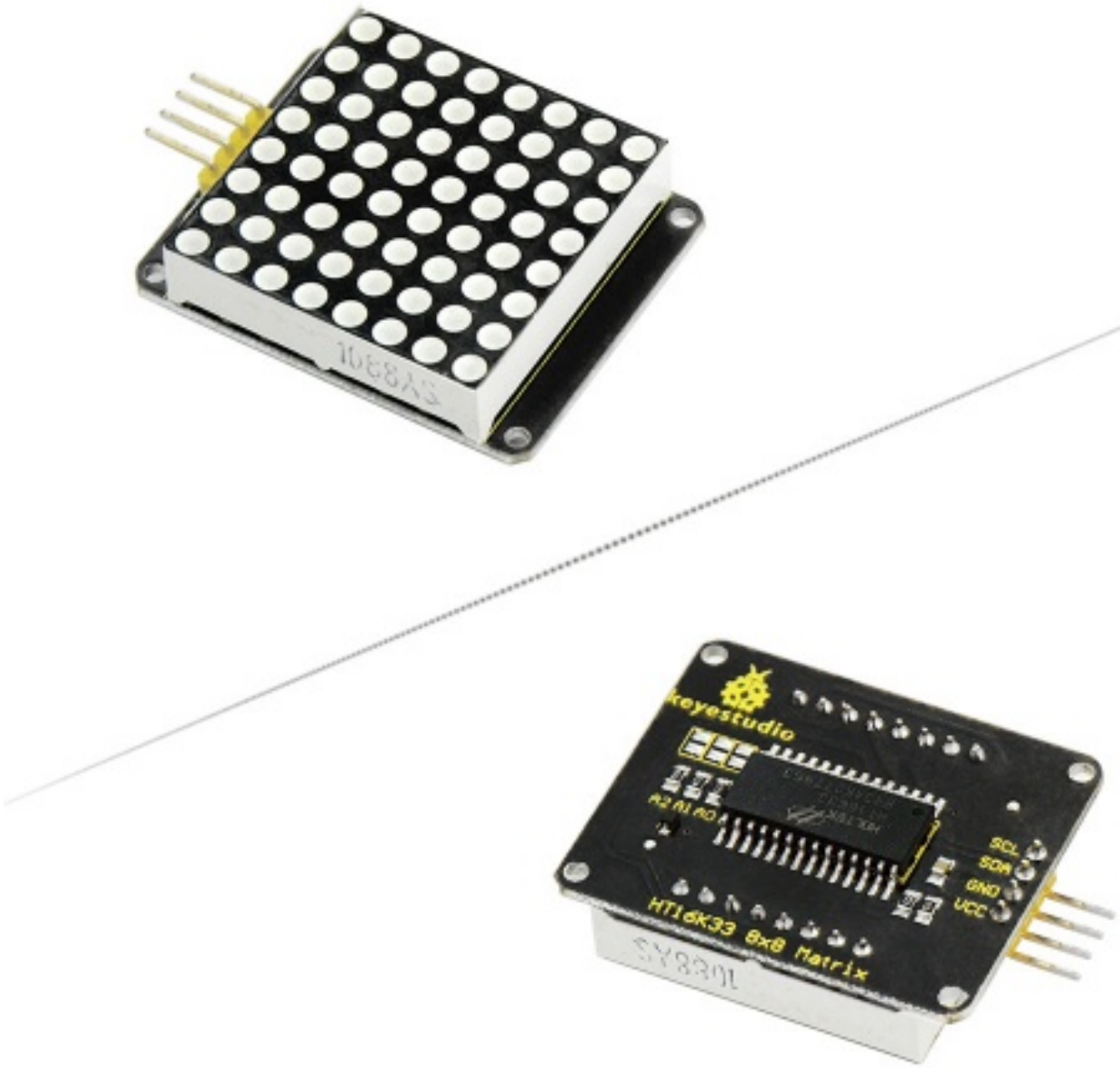
Gibt den Wert einer Aktuatorvariable zurück.

function *actor*:control?(...)

Ausführung eines Steuerkommandos.

Punktmatrix Ausgabegerät

- 8 rows and 8 cols LED matrix
- Driven by HT16K33 chip
- Access to I2C communication pins


keyestudio
A2 A1 A0
SCL
SDA
GND
VCC
HT16K33 8x8 Matrix


Drive Chip HT 16K33
keyestudio
A2 A1 A0
SCL
SDA
GND
VCC
HT16K33 8x8 Matrix
I2C communicati
(SCL)
I2C communicati
(SDA)
GND (ground)
power positive (5V)
8*8 LED dot display

Beispiele

```
-- Actor DOTS Matrix
local dot = actor:new('dots')
dot:reset()
-- dot:set(on,row,col)
dot:set(1,4,4)
sleep(1000)
dot:set(0,4,4)

-- Sensor CPU Temperatur
local s = sensor:new('temperature')
print(s:read())
```

10.6. Asynchrone Ereignisverarbeitung

Synchrone Verarbeitung

Die aufrufende Operation, z.B. das Lesen aus einer Datei, wird solange "blockiert"/wartet (genauer die Prozessausführung), bis das Ereignis eingetreten ist, d.h. im Beispiel die Daten gelesen wurde.

➤ Eine synchrone Operation (Prozedur) gibt das Ergebnis der Operation direkt zurück

Asynchrone Verarbeitung

Die aufrufende Operation wartet nicht auf das Eintreten des Ereignisses (keine Blockierung), und der Prozesse fährt unmittelbar danach in der Ausführung folgender Operationen fort.

➤ Jetzt wird eine Rückruffunktion verwendet, um beim Eintreten des Ereignisses die Daten (oder das Ereignis) verarbeiten zu können.

➤ In JavaScript werden i.A. alle Ein- und Ausgabeoperation asynchron ausgeführt. In Lua gibt es eine ausgewogene Mischung, bzw. die synchrone Variante ist vorherrschend.

Definition 11. (*Synchrone Ereignisverabeitung*)

$$result = ioop_1^{synchron}(arg_1, arg_2, ..)$$
$$result = ioop_2^{synchron}(arg_1, arg_2, ..)$$
$$..$$

$Process{=}ioop_1{}^{\text{synchron}}{\rightarrow}ioop_2{}^{\text{synchron}}{\rightarrow}next$

Definition 12. (*Asynchrone Ereignisverabeitung*)

$ioop_1^{asynchron}(arg_1, arg_2, .., \textbf{function}_1\ (p_1, p_2, ..)\ processing\ \textbf{end})$
$ioop_2^{asynchron}(arg_1, arg_2, .., \textbf{function}_2\ (p_1, p_2, ..)\ processing\ \textbf{end})$
..

$Process{=}ioop_1{}^{\text{asynchron}}{\rightarrow}ioop_2{}^{\text{asynchron}}{\rightarrow}next..{\rightarrow}function_1{\rightarrow}function_2\ |$
$ioop_1{}^{\text{asynchron}}{\rightarrow}ioop_2{}^{\text{asynchron}}{\rightarrow}next..{\rightarrow}function_2{\rightarrow}function_1\ |$
$ioop_1{}^{\text{asynchron}}{\rightarrow}function_1{\rightarrow}ioop_2{}^{\text{synchron}}{\rightarrow}next..\ |\ ..$

Beispiele in Lua

➤ Die *sleep* Operation kann verwendet um den aktuellen Programmfluss für eine bestimmte Zeit (Millisekunden) anzuhalten

Example 3. (*Synchrone Verzögerung einer Serviceschleife*)

```
local s = sensor:new('temperature')
local doit = true
while doit do
  local v = s:read()
  print('The current temperature is '..v)
  sleep(1000)
end
```

➤ Alternativ kann auch ein Timer erzeugt werden. Nach dem Start des Timers (mittels der *start* Methode und der Angabe des Zeitintervalls bis ein Timerereignis ausgelöst wird und dem Zeitintervall zwischen zwei Ereignissen) kann mittels der *wait* Methode der Prozessfluss angehalten werden bis ein Timerereignis eintritt

Example 4. (*Synchrone Verzögerung einer Serviceschleife mit einem Timer*)

```
local s = sensor:new('temperature')
local t = timer:new()
local doit = true
t:start(1000,1000)
while doit do
  local v = s:read()
  print('The current temperature is '..v)
  t:wait()
end
```

Example 5. (*Periodische asynchrone Ausführung einer Serviceschleife*)

```
local s = sensor:new('temperature')
setInterval(1000, function ()
  local v = s:read()
  print('The current temperature is '..v)
end)
```

Example 6. (*Asynchrone Implementierung eines HTTP WEB Servers*)

```
local H = html:new({})
local s = sensor:new("temperature")
http:service(8080, function (req, url, remote, params)
  return H:html(H:head(), H:body({
     H:h1("Temperature"),
     H:p(s:read())
   }))
end)
```

Aufgabe

1. Was ist der Nachteil bei der Verwendung eines Timers mit explizitem Warten auf das Ereignis im Vergleich zur *sleep* Operation oder der asynchronen Ausführung mittels *setInterval*?
2. Was sind Vorteile und Nachteile der synchronen gegenüber der asynchronen Ereignis- und Datenverarbeitung?

11. Sensorische Systeme

11.1. Sensoraggregation

In sensorischen Systemen werden Sensordaten in verschiedenen Ebenen verarbeitet:

Vertikale Ebenen

Perzeption

Hier findet die Akquisition der rohen Sensordaten statt. Die Sensoren sind räumlich verteilt und werden lokal vorverarbeitet.

Aggregation

Einzelne Sensordaten werden zeitlich und räumlich zusammengeführt und gesammelt (Sensorfusion)

Applikation

Die gesammelten Daten werden nutzbar gemacht: Weitere Datenverarbeitung, Aufbereitung, Eigenschaftsselektion, Informationsgewinnung, Visualisierung

Horizontale Ebenen

➤ Die horizontalen Ebenen durchziehen alle vertikalen Ebenen:

1. Sicherheit
2. Datenverarbeitung
3. Kommunikation
4. Datenspeicherung
5. Nachrichtenvermittlung
6. Management

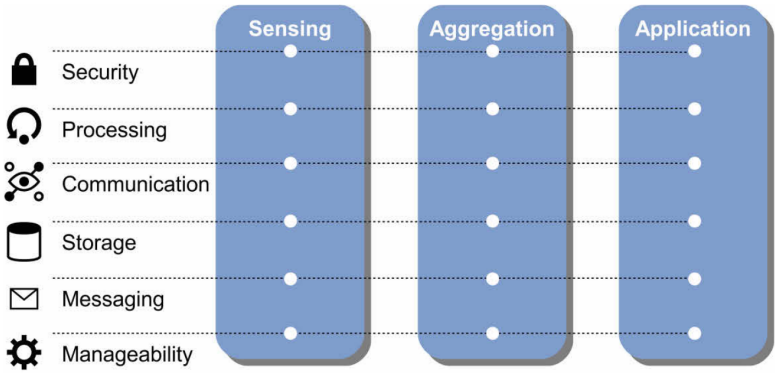


Figure 83. Grundlegender Zusammenhang der horizontalen und vertikalen Ebenen in Sensorischen Systemen

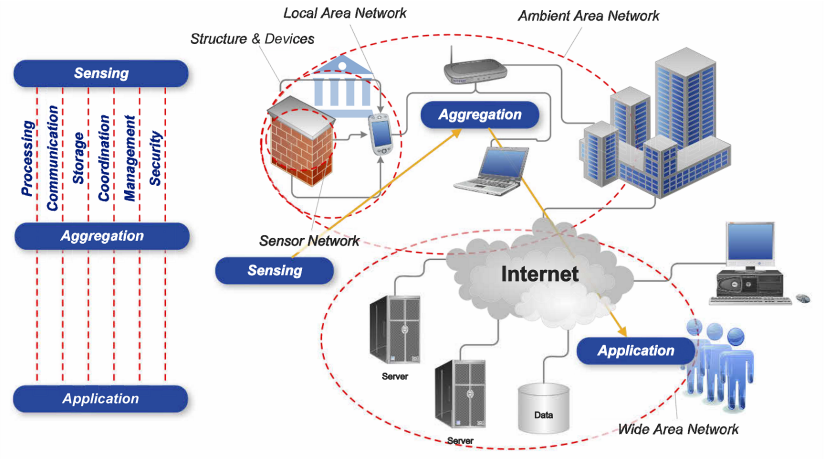


Figure 84. Räumliche Abbildung der vertikalen Ebenen auf Cloud Computing

12. Messtechnik

Elektrische Messtechnik, Elektronik, und Signalverarbeitung

12.1. Messung elektrischer Größen

- Elektrische Größen: 1. zeitlich konstant → Gleich..., 2. zeitlich veränderlich → Wechsel...
- Elektrische Größen lassen sich meist nicht direkt messen - Wandlung in eine Spannung oder einen Strom erforderlich

Größe	Zusammenhang
Spannung U	$U=RI$, $u(t)=ri(t)$
Strom I - der elektrische Strom I ist der Quotient aus der ladungsmenge δQ, die während der Zeit δt durch einen elektrischen Leiter fließt.	$I=U/R$, $i(t)=\delta Q/\delta t$, $I=Q/T$
Widerstand R	$R=U/I$
Leistung P	$P=UI$, $p=\delta W/\delta t$
Ladung Q - elektrische Spannung U ist der Quotient aus der zur Verschiebung der Ladung Q erforderlichen Arbeit W	$Q=W/U$, $U=W/Q$
Frequenz f - reziprok zur Periodendauer T einer Schwingung	$f=1/T=\omega/2\pi$

Figure 85. Elektrische Größen und deren Zusammenhänge

➤ Messtechnik wird häufig Spannungen und Ströme an verschiedenen Punkten in der Signalverarbeitung erfassen

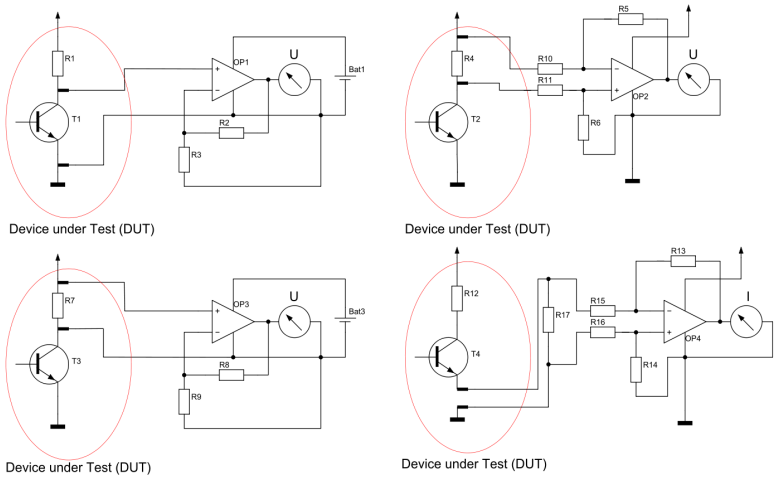


Figure 86. Messung von Spannung und Strom: Potentialfrei und potentialgekoppelt

➤ Wechselspannungen und Wechselströme müssen zeitaufgelöst gemessen werden → Oszilloskop

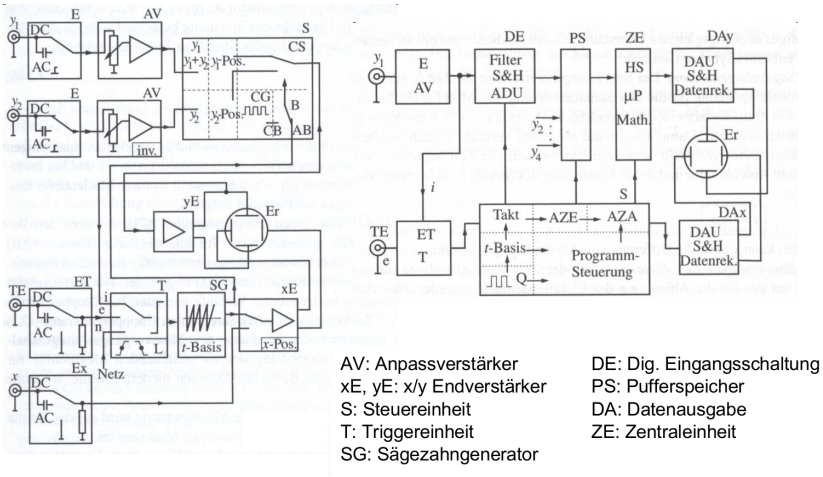


Figure 87. Messung von zeitlichen Verläufen mit einem Oszilloskop (links: analog, rechts: digital) [TDM, Hoffmann, 2002]

12.2. Operationsverstärker

- ➤ Basis vieler Signalaufbereitungs- und Messmodule ist der Operationsverstärker
- ➤ Der Operationsverstärker ist ein linearer Differenzverstärker mit sehr großer Leerlaufverstärkung (ohne weitere Beschaltung) V_0 ≈ 1000-100000
- ➤ Der Operationsverstärker besitzt zwei Eingänge und einen Ausgang:

  (+) Nicht invertierender Eingang, Spannung U_p
  (-) Invertierender Eingang, Spannung U_n

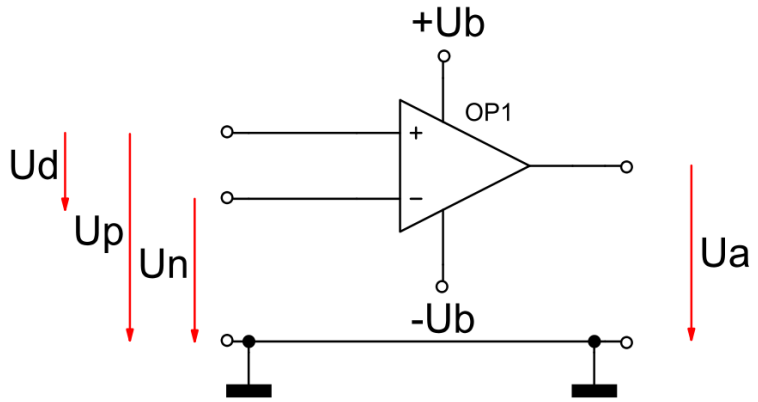


Figure 88. Schaltbild des Operationsverstärkers

- Die Ausgangsspannung U_a ist die Differenz U_d der beiden Eingangsspannungen U_p und U_n (relativ zum Bezugspotential) multipliziert mit der Verstärkung V_0:

$$U_a = V_0(U_p - U_n) - U_0 = V_0 U_d - U_0$$

- Die Ausgangsspannung U_a kann Werte im Bereich $[-\epsilon U_b, +\epsilon U_b]$ annehmen; dabei liegt der Wert ϵ i.A. im Bereich [0.7,0.99] und ist durch die interne Schaltung des OPs vorgegeben (Werte $\epsilon > 0.95$: sog. Rail-to-Rail Verstärker).
- Es gibt eine Eingangsoffsetspannung U_0, die zu einer Verschiebung der Kennlinie des OPs führt. Es gilt: $U_a{=}0$ wenn $U_d{=}U_0$.

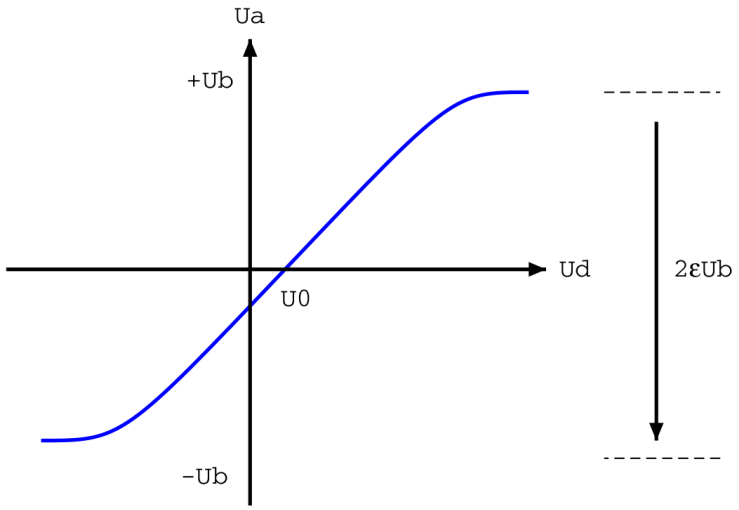


Figure 89. Kennlinie eines Operationsverstärkers

➤ Ursache für Offsetspannung, Sättigung und Nichtlineratität der Kennlinie liegen in der internen Transistorschaltung begründet (Transistor ist ein nicht-lineares Übertragungselement)

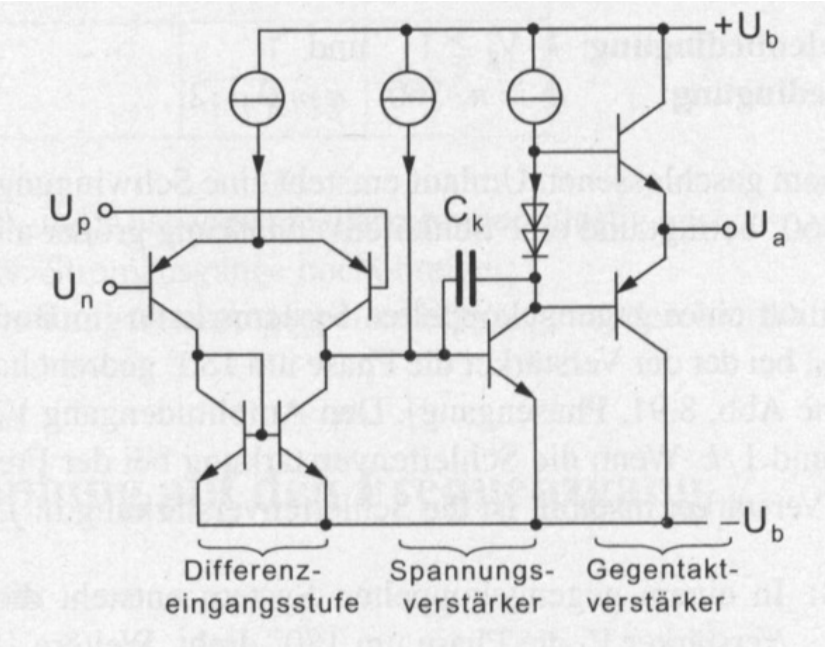


Figure 90. Vereinfachte Transistorschaltung eines Operationsverstärkers [TDE, Kories,2004]

Rückkopplung

Rückkopplung: Das Ausgangssignal einer Schaltung wird auf den Eingang zurückgeführt. Man unterscheidet:

- Gegenkopplung: Ein Teil des Ausgangssignals wird vom Eingangssignal subtrahiert
- Mitkopplung: Ein Teil des Ausgangssignals wird zum Eingangssignal addiert

Kopplungsfaktor k: Anteil des rückgekoppelten Signals

- Verstärkung des rückgekoppelten Systems: $V = V_0/(1 + k * V_0)$

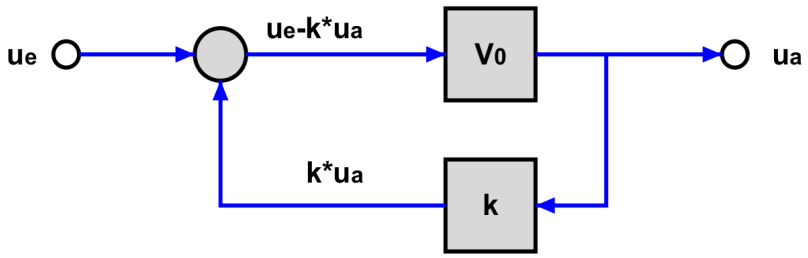


Figure 91. Gegengekoppeltes System mit einer Leerlaufverstärkung V_0

Analog Computer und Schaltungen

A. Impedanzwandler
B. Nichtinvertierender Verstärker
C. Invertierender Verstärker
D. Addierer
E. Subtrahierer
F. Instrumentenverstärker
G. Spannungsgesteuerte Stromquellen
H. Integrator
I. Differenzierer
J. Schmitt-Trigger
K. Multivibrator
L. Sägezahn-Generator
M. Pulsweitenmodulator
N. Aktive Filter
O. Multiplizierer

12.3. Impedanzwandler

➤ Der Impedanzwandler wird in Spannungs-Spannungs-Gegenkopplung mit einer Gegenkopplung von k=1 betrieben. Der Gegenkopplungsgrad ist $g = 1 + kV_0 \approx V_0$

➤ Die Übertragungsfunktion lautet:

$$U_a = U_e \frac{V_0}{1 + kV_0} \approx U_e$$

➤ Der Eingangswiderstand ist sehr hoch, der Ausgangswiderstand niedrig → Impedanzwandlung

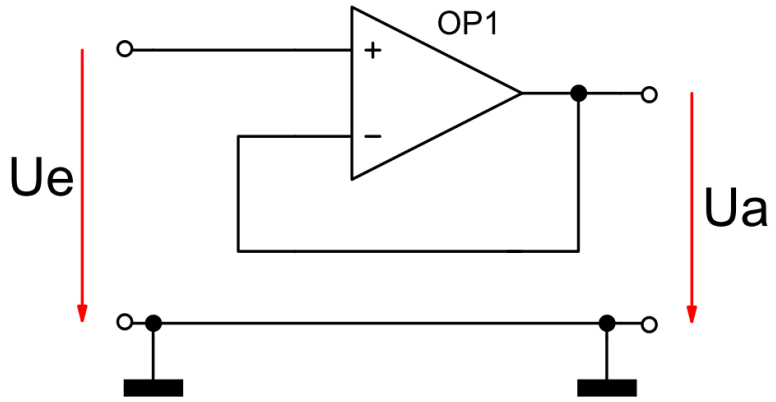


Figure 92. Schaltung des Impedanzwandlers

12.4. Nichtinvertierender Verstärker

➤ Dieser Verstärker wird in Spannungs-Spannungs-Gegenkopplung mit einer Gegenkopplung von $k = R_2/(R_1 + R_2)$ betrieben. Der Gegenkopplungsgrad ist $g = 1 + kV_0$

➤ Die Übertragungsfunktion lautet:

$$U_a = U_e \frac{V_0}{1 + kV_0} \approx U_e(1 + \frac{R_1}{R_2})$$

➤ Der Eingangswiderstand ist sehr hoch ($\rightarrow\infty$), der Ausgangswiderstand niedrig (≈ 0 Ω) → Impedanzwandlung + Spannungsverstärkung

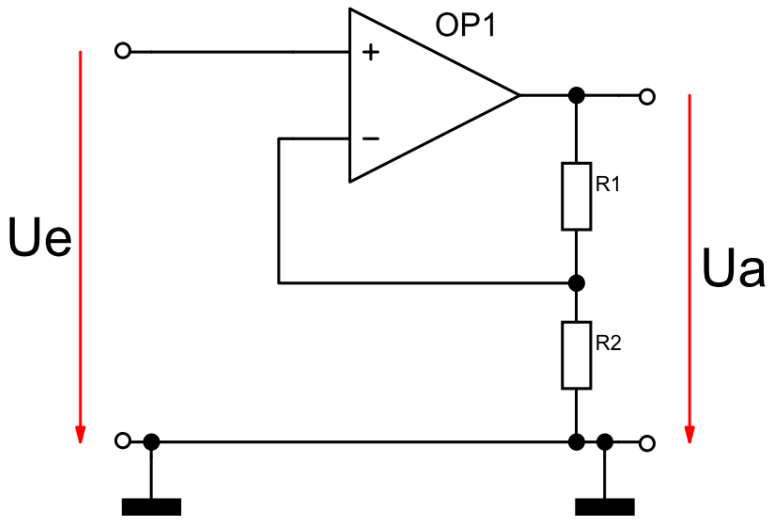


Figure 93. Schaltung des nichtinvertierenderen Verstärkers

12.5. Invertierender Verstärker

- Dieser Verstärker wird in Spannungs-Strom-Gegenkopplung mit einer Gegenkopplung von $k = 1/R_2$ betrieben. Der Gegenkopplungsgrad ist $g = 1 + kV_0$
- Die Übertragungsfunktion lautet:

$$U_a = U_e \frac{V_0}{1 + kV_0} \approx U_e(-\frac{R_2}{R_1})$$

- Der Eingangswiderstand ist gleich R_1, der Ausgangswiderstand sehr niedrig (≈ 0 Ω) $\rightarrow$ Impedanzwandlung + Spannungsverstärkung, die Polarität ändert sich

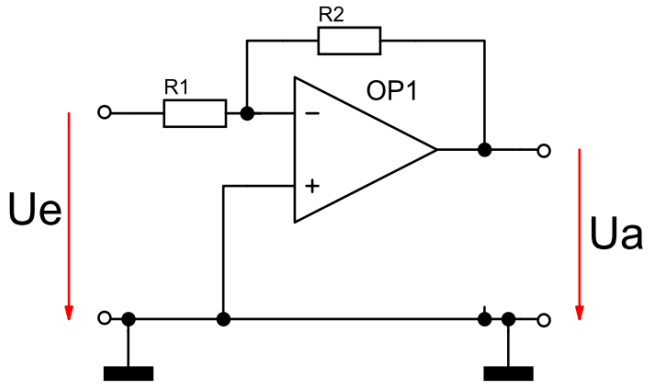


Figure 94. Schaltung des invertierenderen Verstärkers

12.6. Addierer

➤ Dieser Verstärker wird in Spannungs-Strom-Gegenkopplung betrieben. Die Übertragungsfunktion lautet:

$$U_a = -\sum_{i-1}^{n} I_i R_G = -\left(U_{e1}\frac{R_G}{R_1} + U_{e2}\frac{R_G}{R_2} + .. + U_{en}\frac{R_G}{R_n} \right)$$

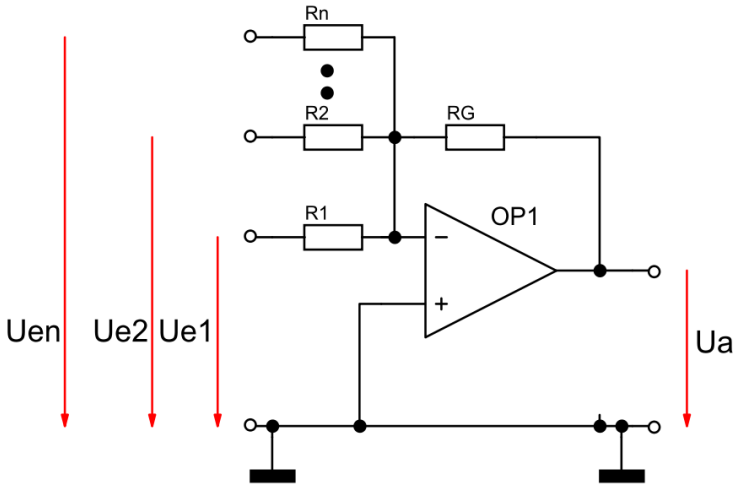


Figure 95. Schaltung Addierer

12.7. Subtrahierer

➤ Dieser Verstärker wird in Spannungs-Strom-Gegenkopplung betrieben. Die Übertragungsfunktion lautet:

$$U_a = (U_{e1} - U_{e2})\frac{R_2}{R_1}$$

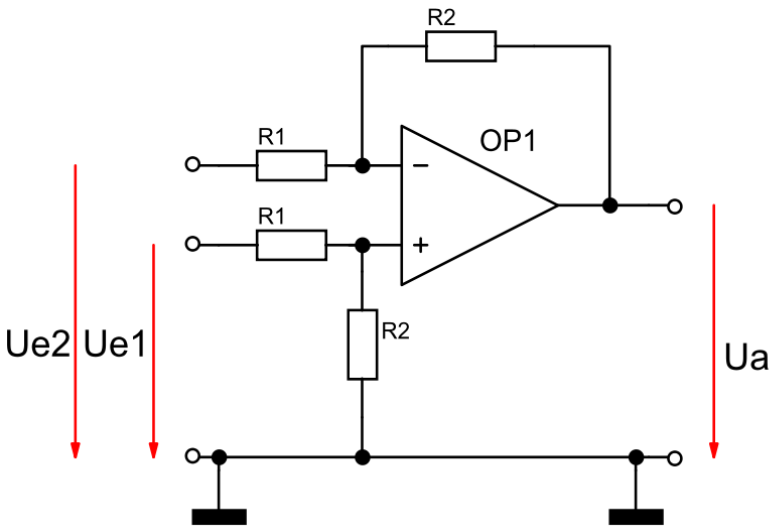


Figure 96. Schaltung des Subtrahierers

12.8. Instrumentenverstärker

➤ Dieser Verstärker misst die Differenz zweier Eingangsspannungen U_{e1} und U_{e2}, wobei jeder Kanal den gleichen sehr hohen Eingangswiderstand besitzt! Die Übertragungsfunktion lautet:

$$U_a = (U_{e1} - U_{e2})(1 + 2\frac{R_2}{R_1})$$

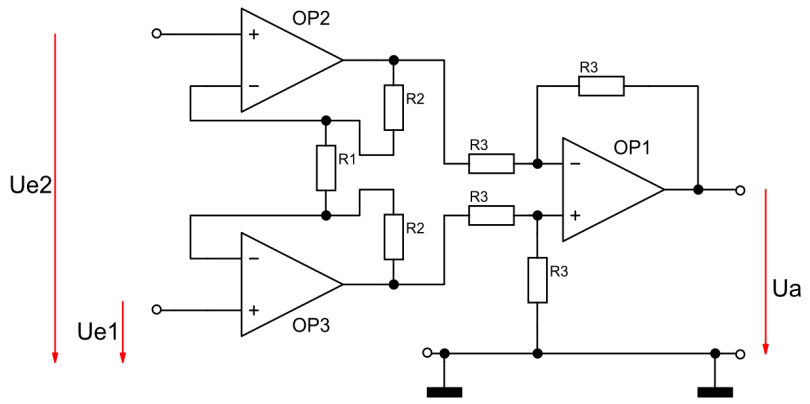


Figure 97. Schaltung des Instrumentenverstärkers

12.9. Integrator

- Dieser Verstärker arbeitet wie ein invertierender Verstärker und lädt einen im Gegenkopplungszweig vorhandenen Kondensator C um. Die Ausgangsspannung ist daher das Integral der Eingangsspannung. (Offsetspannung führt immer zu $\rightarrow$ U_a=± U~b+!)
- Die Übertragungsfunktion lautet:

$$I_e = U_e/R \Rightarrow U_a = -\frac{1}{RC}\int U_e dt$$

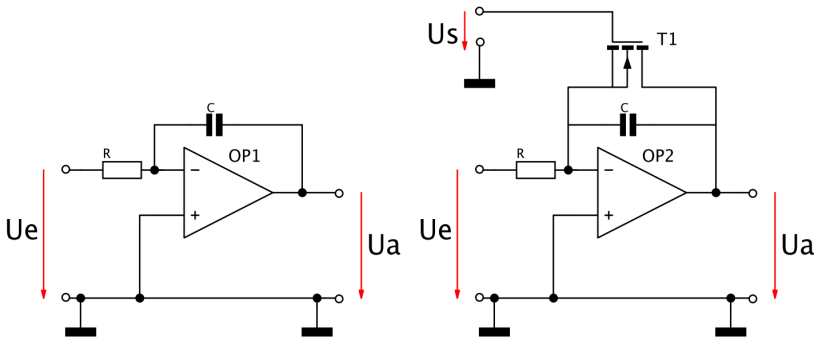


Figure 98. Schaltung Integrator (links) und mit Rücksetzschaltung (rechts)

12.10. Differenzierer

➤ Dieser Verstärker arbeitet wie ein invertierender Verstärker. Die Ausgangsspannung ist die zeitliche Ableitung der Eingangsspannung. Die Übertragungsfunktion lautet:

$$I_e = C\frac{dU_e}{dt} \Rightarrow U_a = -RC\frac{dU_e}{dt}$$

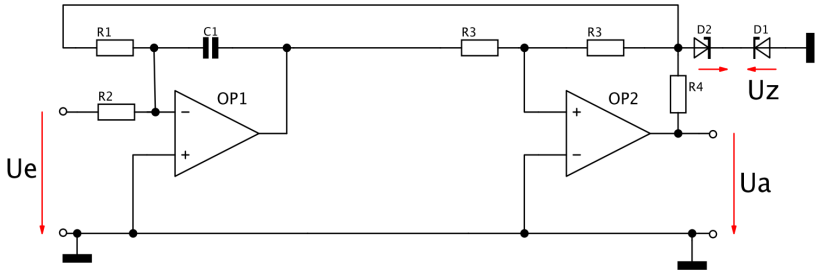


Figure 99. Schaltung des Differenzierers

12.11. Pulsweitenmodulator

➤ Kombination aus einem Differenzierer und Schmitt-Trigger

➤ Eine Eingangsspannung U_e wird in ein (digitales) Rechtecksignal mit fester Periodendauer T und der Eingangsspannung proportionalen Tastzeit t_1 umgesetzt.

$$\frac{t_1}{T} = 1/2\left(1 - \frac{U_e}{U_Z - 0.7\text{V}}\frac{R_1}{R_2}\right)$$

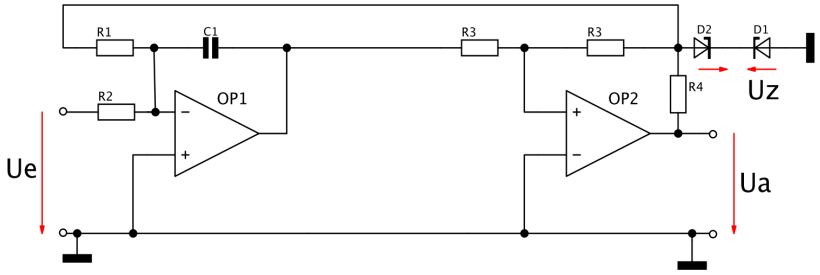


Figure 100. Schaltung Pulsweitenmodulator

12.12. Aktives Filter: Tiefpass 1. Ordnung

- Ein Tiefpass besteht aus einem RC Glied. Dieses RC-Glied kann entweder passiv sein, mit einem nachgeschalteten Verstärker, oder sich aktiv im Rückkopplungszweig eines invertierenden Verstärkers befinden.
- Die Übertragungsfunktion eines Tiefpasses besitzt in Abhängigkeit der Frequenz eines Signals ab einer Grenzfrequenz eine zunehmende Dämpfung (Hochpass: umgekehrt).
- Grenzfrequenz: $f_{-3db}=1/(2\pi R_2 C_1)$

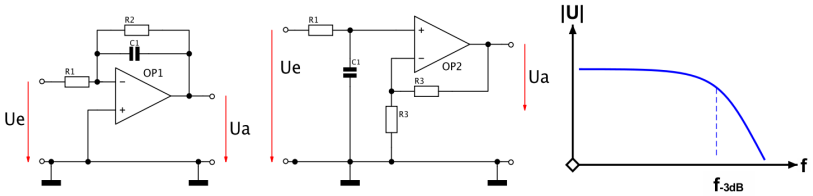


Figure 101. Schaltung Tiefpass 1. Ordnung (links invertierend, rechts nicht invertierend)

12.13. Multiplizierer

- Multiplikation und Division werden auf Addition und Subtraktion von Logarithmen zurückgeführt → Bipolare Transistoren besitzen eine logarithmische Übertragungsfunktion (Kollektorstrom in Abhängigkeit vom Basisstrom).

$$\frac{xy}{z} = e^{\ln(x)+\ln(y)-\ln(z)}$$

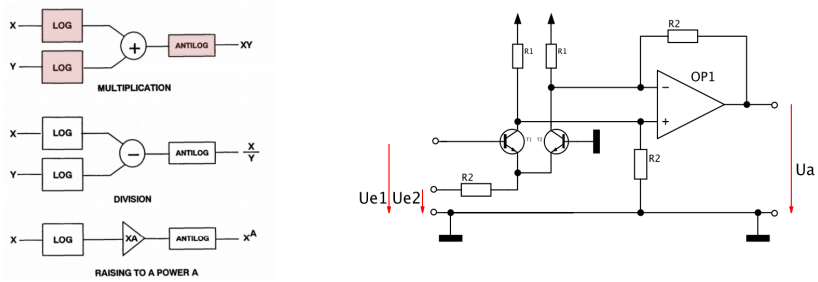


Figure 102. Prinzip Multiplizierer (links) und Schaltung Steilheitsmultiplizierer (rechts)

12.14. Digital-Analog Wandler

- Die Aufgabe eines DA-Wandlers besteht darin, eine Zahl Z kodiert mit Bitvektoren (i.A. binärgewichtete Kodierung) in eine dazu proportionale Spannung U_s umzuwandeln.
- Es gibt verschiedene Prinzipien:

  I. Parallelverfahren

  II. Wägeverfahren

  III. Zählverfahren

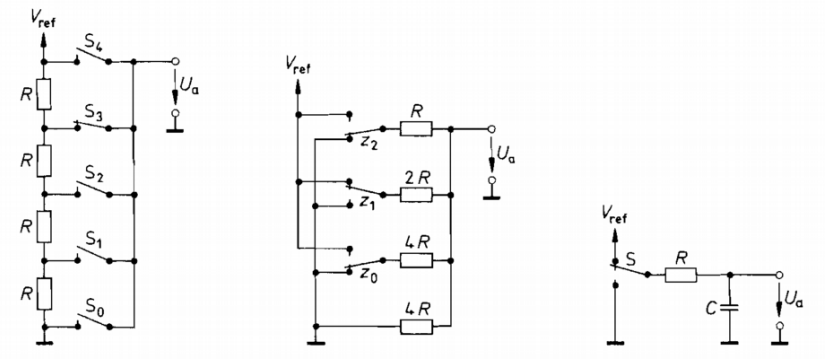


Figure 103. vlr.: Parallelverfahren, Wägeverfahren, Zählverfahren [HST, Tietze, 2002]

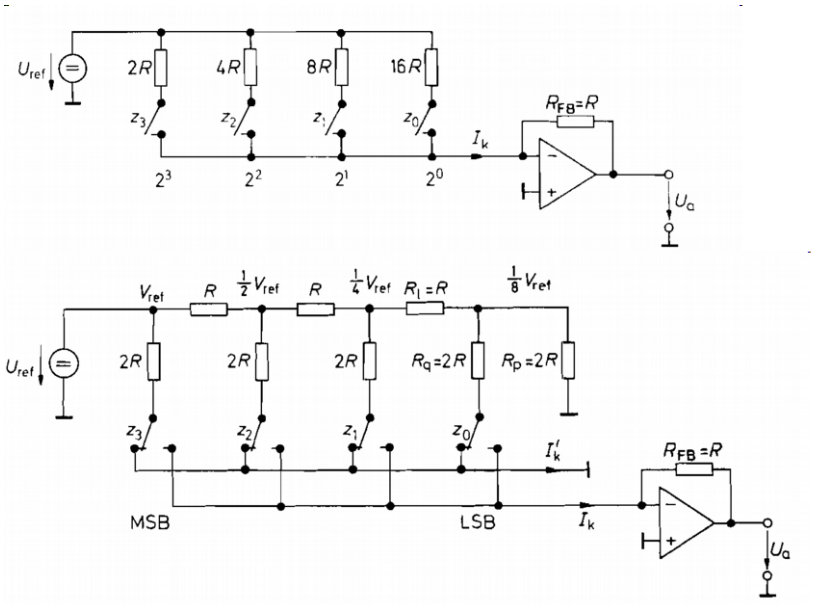


Figure 104. DA Prinzip der Summation gewichteter Ströme u. Leiternetzwerk [HST, Tietze, 200]

12.15. Analog-Digital Wandler

➤ Ein AD-Wandler ist in drei Stufen unterteilt:

I. Der Sampler führt eine Diskretisierung in der Zeitdimension durch,

II. Der Quantisierer führt eine Diskretisierung in der Wertdimension durch, derart, dass ein quantisierter Wert einem Wertintervall q(n-Δ) $\leq$ q(n) $<$ q(n+Δ), mit $\Delta/2$ als Auflösung des Quantisierers, entspricht, und

III. Einem Kodierer, der das quantisierte Signal in einen Digitalwert kodiert, i.A. Kodierung nach dem Dualzahlensystem.

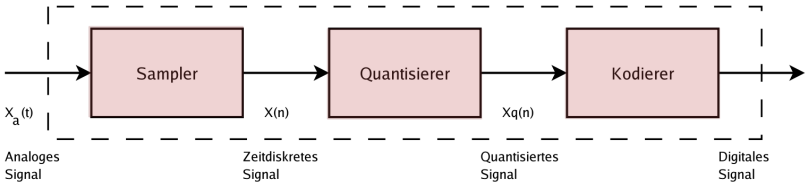
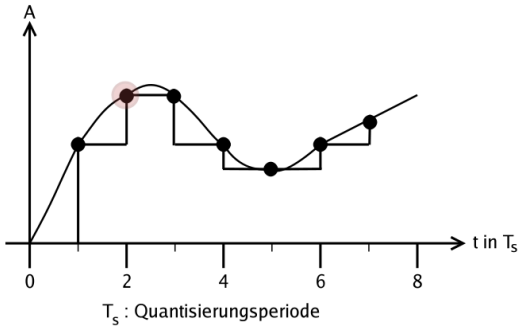


Figure 105. Aufbau eines ADC

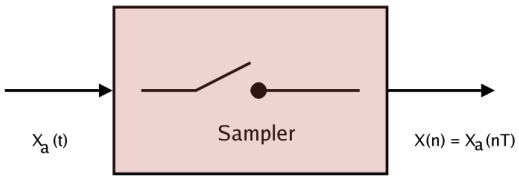
Diskretisierung

➤ Die Diskretisierung eines analogen Signals bedeutet eine Diskretisierung im Zeit- und Signalraum



Sampler

➤ Der Sampler führt die Diskretisierung im Zeitraum aus



Sampling Theorem

➤ Im allgemeinen wird in der digitalen Signalverarbeitung ein analoges Signal periodisch mit einer festen Sample-Frequenz abgetastet.

➤ Jedes elektrische Wechselsignal ist aus einer Überlagerung von verschiedenen Sinus- und Cosinusschwingungen zusammengesetzt, die sich unterscheiden nach:

a. Frequenz

b. Amplitude

c. Phase (zueinander)

$$s(t) = a_0 + \sum a_k \cos(2\pi k f_0 t) + \sum b_k \sin(2\pi k f_0 t)$$

➤ Das Sampling-Theorem besagt, dass das zu digitalisierende Signal nur ein Frequenzspektrum bis zu einer maximalen Frequenz besitzen darf:

$$f_{sample} > 2 f_{signal}$$

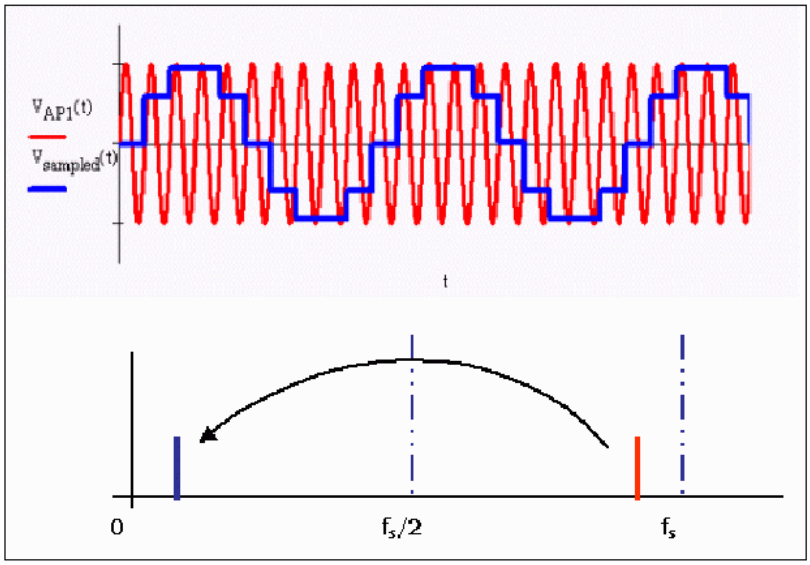


Figure 106. Signalfrequenzen oberhalb $f_{\text{sample}}/2$ werden zu niedrigeren Frequenzen gespiegelt und erzeugen Fälschungen (Aliasing)

➤ Ähnlich wie bei den DA-Wandlern gibt es bei den AD-Wandlern unterschiedliche Verfahren wo einige DA- Wandler verwenden um einen Spannungswert zu approximieren.

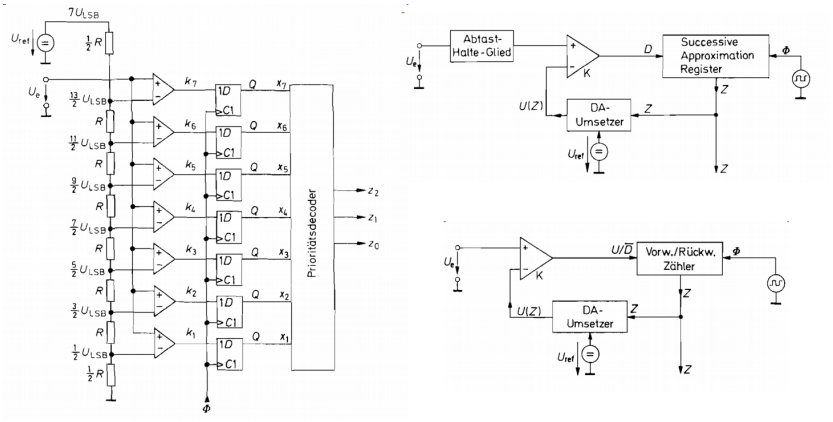


Figure 107. Verschiedene AD Verfahren (Parallel-, Wäge- und Zählverfahren) [HST, Tietze, 2002]

12.16. Zooming ADC

- Resistive Sensoren, z.B. Dehnungsensoren, liefern eine nur kleine relative Änderung ihres Widerstandes in der Größenordnung von 1% resultierend von einer Änderung der Dehnung im gesamten Arbeitsbereich des Sensors.
- Annahme: nur ein unkalibrierter und unkompensierter resistiver Sensor
- Ein Fensterverfahren kann verwendet werden um einen solchen Sensor an das Messsystem anzupassen → hohe Auflösung und Nutzung des vollen Bereichs

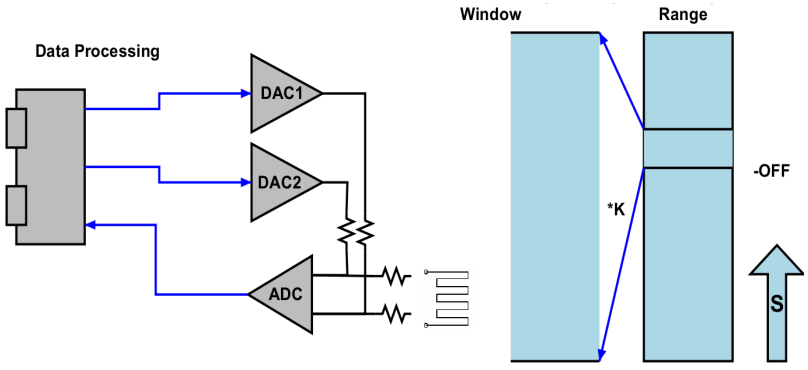


Figure 108. Fensterverfahren (zooming in region of interest)

- Ein parametrisierbarer Bereichsausschnitt des Eingangssignals wird auf das volle digitale Werteintervall abgebildet:

$$W(s) = k(s - off)$$

Algorithm 1. (*Autokalibration mit sukzessiver Approximation*)

$sar \rightarrow DIGITALRANGE/2$
$DAC_1 \rightarrow GAIN_0$
$DAC_2 \rightarrow 0$
WHILE $sar <> 0$ **DO**
  IF $ADC > DIGITALRANGE/2$ **THEN**
    $DAC_2 \rightarrow DAC_2 + sar$
  ELSE
    $DAC_2 \rightarrow DAC_2 - sar$
  END
  $sar \rightarrow shift_{right}(sar, 1)$
DONE
$off \rightarrow DAC_2 - DAC_1$

12.17. Messbrücken

➤ Betriebsschaltung für Druck-, Kraft- und Dehnungssensoren: Wheastone Messbrücke

$$U_a = 2\left(1 + \frac{R_2}{R_1}\right)(V_1 - V_2) + V_N$$

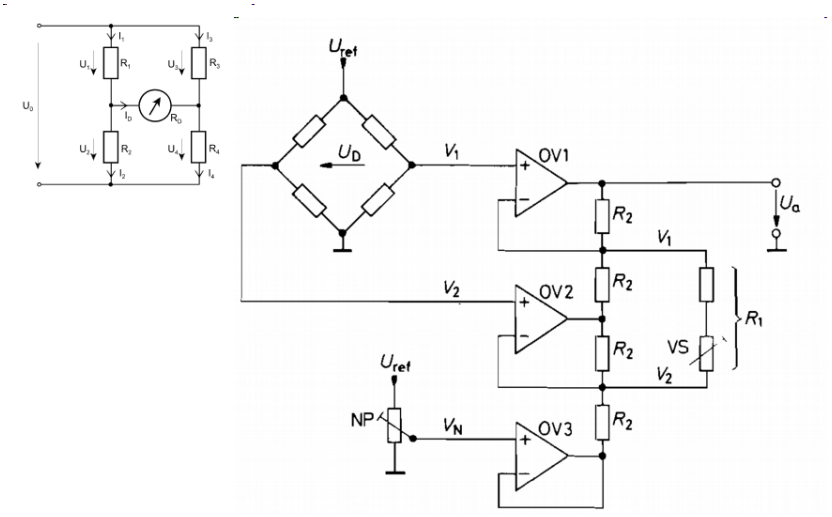


Figure 109. Messbrückenschaltung

13. Messdatenauswertung

13.1. Das Instrumentenmodell

➤ Der Sensor hat die Aufgabe eine physikalische Variable X in eine Signalvariable (i.A. elektrisch) S zu wandeln.

Physikalische Var. X	**Signalvariable S**
Kraft (Dehnung)	Widerstand
Länge	Strom
Temperatur	Spannung
Druck	Kapazität

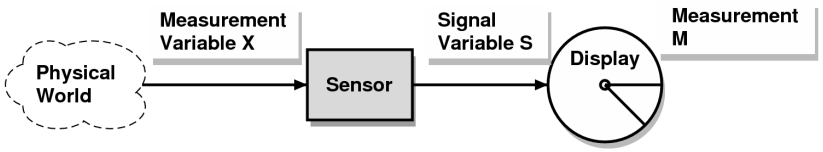


Figure 110. Einfaches Instrumentenmodell: Sensor und Display

➤ Das Signal S kann dargestellt, aufgenommen oder weiter verarbeitet werden.

➤ Das Signal S ist eine analoge Größe. Ein Signal besteht aus einem Nutz- und einem Rauschanteil, der immer vorhanden ist, und eine Störung der Messung darstellt.

*Analoge (physikalische) Signale: Analoge Signale sind zeit- und wertkontinuierlich, d.h. man findet in einem beliebig kleinen Intervall [*a,b*] immer eine Zahl* c *für die gilt:* $a < c < b$*. Ein analoges Signal besitzt aufgrund physikalischer Vorgänge die Eigenschaft keinen exakten und zeitlich konstanten Wert zu besitzten, sondern setzt sich zusam- men aus einer Überlagerung mit einem stochastischen Rauschsignal.*

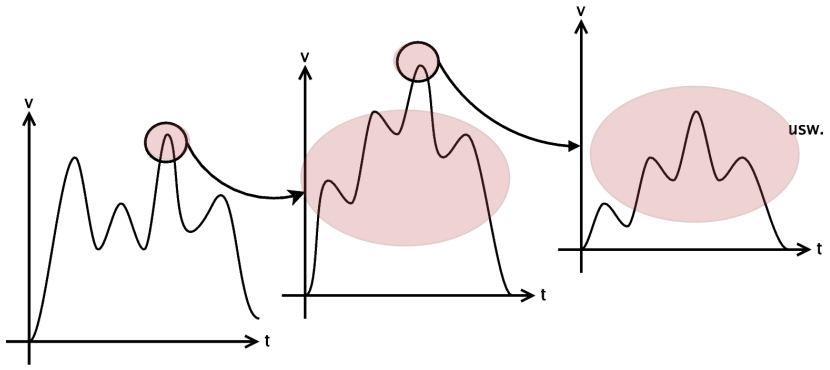


Figure 111. Zoom eines analogen Signals in Zeit- und Wertdimension (vereinfacht)

➤ Der Zusammenhang zwischen dem Signal S und der physikalischen Messgröße X ist durch eine Funktion K gegeben (Übertragunsfunktion). Sie ist i.A. nicht linear und unbekannt, und wird meistens durch Kalibrierung ermittelt und kann mit einem Polynom m-ten Grades angenähert werden (Kalibrierungsfunktion)

$$K(X) \ : \ X \rightarrow S \ , \ K(x) \approx \sum a_n x^n$$

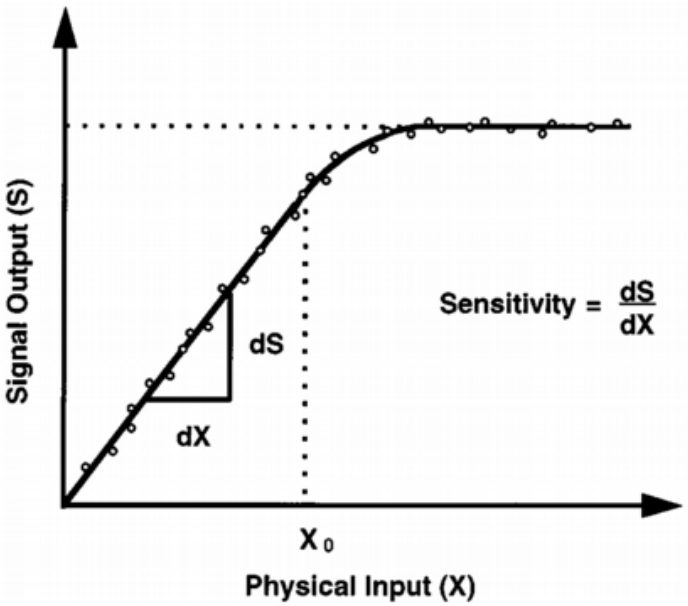


Figure 112. Die Kalibrierungsfunktion wird i. A. aus Messungen und Regression (polynomielle Anpassung) abgeleitet. [Webster, MISH, 1999]

- In der digitalen Signalverarbeitung findet eine Wandlung der analogen Signalgröße S in ein zeit- und wertdiskreten digitalen Wert D statt.
- Elektrische Signale Se werden i. A. mit einem elektrischen Signalverstärker vergrößert bevor die eigentliche Analog-Digital Wandlung stattfindet.
- Ein zeitlich variierendes Sensorsignal $S^{\sim}$ kann aus einer Überlagerung von sinusförmigen Grundschwingungen mit verschiedenen Frequenzen, Amplituden, und relativen Phasen gebildet werden. Die Messung von Wechselsignalen erfordert i.A. ein Frequenzfilter (Tiefpassübertragung).

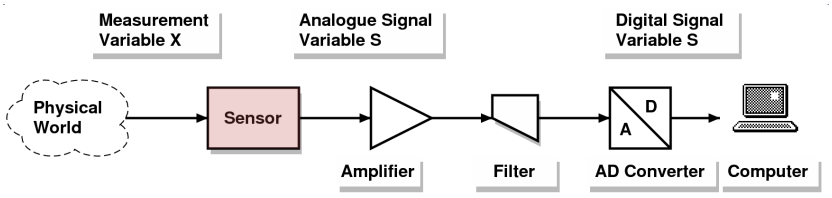


Figure 113. Digitales Signalverarbeitungssystem

Aktive und passive Sensoren

- Passive Sensoren werden nur durch physikalische Vorgänge der Umgebung angetrieben, die auch ohne den Sensor stattfinden.
- Aktive Sensoren erzeugen einen Stimulus, d. h. es findet eine Anregung der Umgebung statt, und es wird die Reaktion der Umgebung auf diese Anregung gemessen.

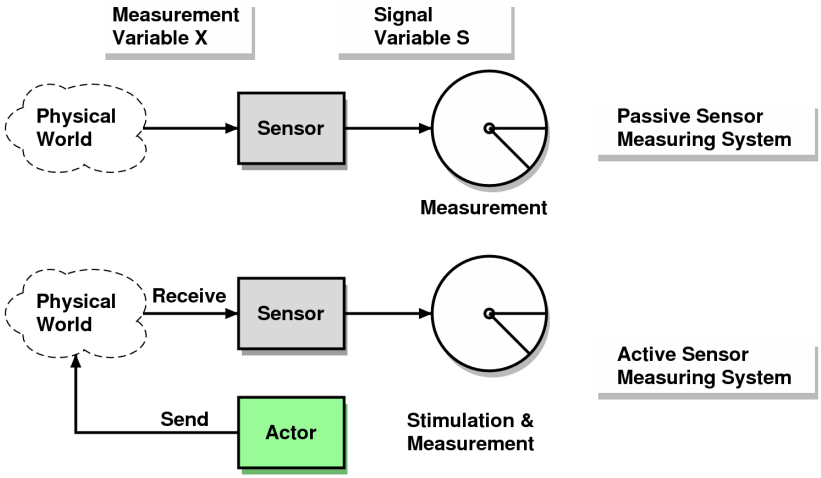


Figure 114. Aktive und passive Messverfahren.

13.2. Messfehler und Vertrauen

- Die Messgrößen können statisch (zeitlich konstant) oder dynamisch (zeitlich veränderlich) sein. Die Wandlung dieser Messgrößen ergeben dann entsprechend Gleich- und Wechselsignale.
- Auch eine prinzipiell zeitlich unveränderliche Messgröße (bezogen auf die Messung in einem vorgegeben Zeitinterval) erzeugt kein konstantes Signal. Ursache: Rauschen
- Wiederholt man daher eine Messung N-mal unter gleichen Bedingungen, so wird man eine Reihe von verschiedenen Messwerten {s1,s2,...,sn} erhalten.
- Es gibt systematische und zufällige Fehler bei der Messung, die sich überlagern.

Systematische Abweichung (systematischer Fehler):

- Abweichung wird durch den Sensor verursacht
- z.B.: falsche Eichung, dauernd vorhandene Störungen wie Reibung
- lässt sich nur durch sorgfältiges Untersuchen der Fehlerquelle beseitigen

Zufällige Abweichung (zufälliger oder statistischer Fehler):

- Abweichung wird durch unvermeidbare, regellose Störungen verursacht
- bei wiederholter Messung weichen Einzelergebnisse voneinander ab
- Einzelergebnisse schwanken um einen Mittelwert

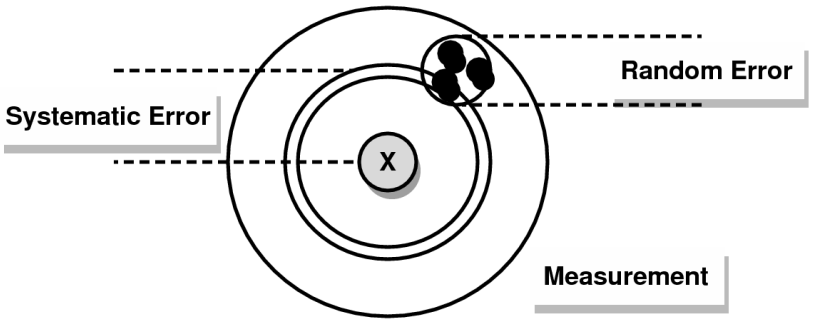


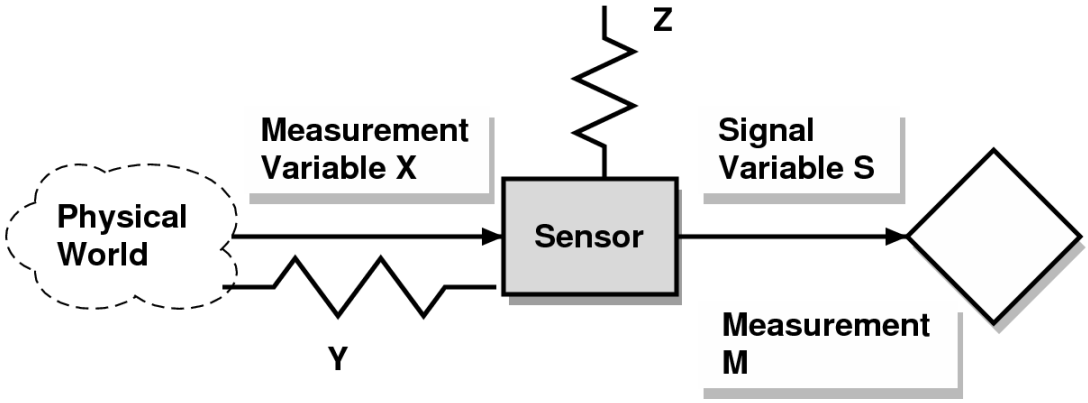
Figure 115. Offset und Präzision bei der Messung einer Variable X

Systematische Fehler

- Eine Messgröße X ist meistens durch störende Messgrößen Y,Z,... usw. überlagert:

$$K(X,Y,Z) \; : \; X \times Y \times Z \rightarrow S, K(x,y,z) \approx \sum_{n=0}^{m} a_n x^n + \sum_{n=0}^{m} b_n y^n + \sum_{n=0}^{m} c_n z^n$$

- So kann z.B. bei einer Messung einer Kraft oder einer Dehnung die umgebende Temperatur T oder Strukturschwingungen Einfluss auf den Sensor und dessen Übertragungsfunktion und somit auf das Messsignal S haben.



- Systematische Fehler verfälschen die Kalibrierungsfunktion (z. B. bei Geraden den Offset und Steigung). Sind sie bekannt, können sie kompensiert (rausgerechnet) werden.
- Systematische Fehler können aber auch während der Signalverarbeitung entstehen, so z. B. Offsetspannungen und zeitlicher Drift von Parametern (Verstärkungsfaktor).

Zufällige Fehler - Streuung

- Zufällige Fehler beeinflussen die Genauigkeit einer Messung (Rauschen).
- Wiederholt man eine Messung einer Größe X die durch reine zufälligen Fehler verfälscht wird, so ist die Häufigkeitsverteilung der Messwerte $S=\{s_1,s_2,...,s_n\}$ um einen Mittelwert $\bar{S}$ durch eine Gaussverteilung gegeben (dabei muss die Anzahl der Messungen N groß sein).

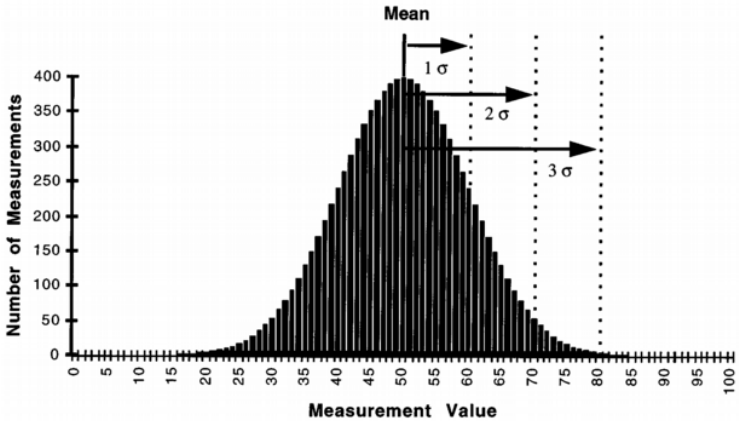


Figure 116. Häufigkeitsverteilung nach Gauss von Messwerten um einen Mittelwert

➤ Der Mittelwert S repräsentiert die Abschätzung des wahren/wirklichen Wertes Σ der Messgröße X (oder S):

$$\bar{S} = \frac{1}{N} \sum_{i=1}^{N} s_i$$

➤ Die Standardabweichung ist ein Maß für die Zuverlässigkeit (Präzision) der einzelnen Messwerte einer Messreihe $\{s_1,s_2,...,s_n\}$:

$$\sigma = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} (s_i - \bar{S})^2}$$

Eine Vergrößerung der Anzahl N der Messungen (unter gleichen Bedingungen!) führt zu einer Verbesserung des Mittelwertes $\bar{S}$ (Grenzfall $N \to \infty$), nicht aber zu einer wesentlichen Verkleinerung der Standardabweichung σ, da die Genauigkeit nicht steigt!

➤ Der wirkliche Mittelwert Σ ist nicht bekannt (nur im Grenzfall $N \to \infty$ ist $\bar{S}=\Sigma$) - es gibt aber ein Vertrauensintervall mit einer Wahrscheinlichkeit P dass dieser darin enthalten ist:

$\Sigma \in [\bar{S}\text{-}\sigma,\bar{S}\text{+}\sigma]$ mit 68.3%,
$\Sigma \in [\bar{S}\text{-}2\sigma,\bar{S}\text{+}2\sigma]$ mit 95.4%,
$\Sigma \in [\bar{S}\text{-}3\sigma,\bar{S}\text{+}3\sigma]$ mit 99.73%

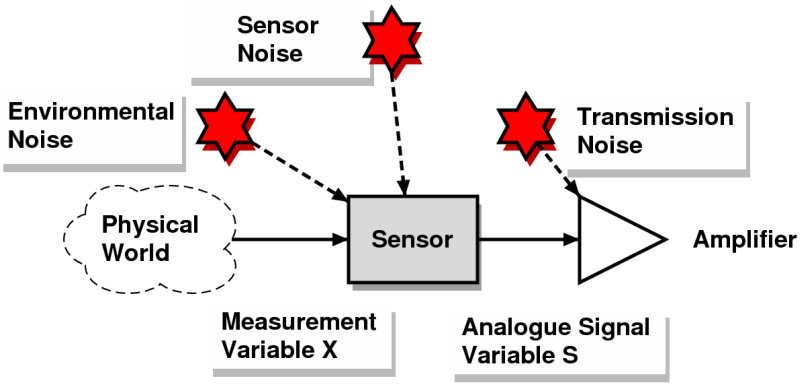
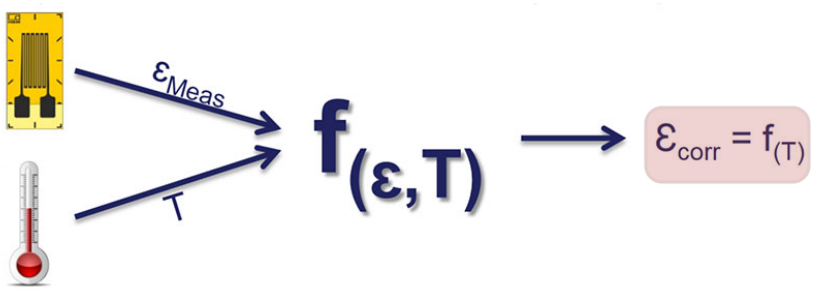


Figure 117. Rauschquellen bei einer Messung

13.3. Sensorfusion

Sensor Fusion: viele Sensoren helfen viel?

- Die Genauigkeit einer Messung bzw. das Vertrauen in einen Messwert lässt sich durch Zusammenschluss und Korrelation mehrerer verschiedener Sensoren R_1,R_2, ... und Sensorsignale erhöhen s_{r1},s_{r2},.. $\Rightarrow$ **Sensordatenfusion**
- So kann z.B. gleichzeitig die Dehnung mit einem Dehnungssensor (Wandlung in elektrische Widerstandsänderung) und die Temperatur mit einem Temperaturssensor gemessen werden, um auch systematischer Fehler durch Temperaturdrift der Übertragungsfunktion kompensieren zu können.
- Durch die Spannungsversorgung und ein Stromfluss I=U/R erwärmt sich ein Dehnungsmessstreifen gegenüber dem Messkörper. Je nach Wärmeleitfähigkeit des Messkörpers wird die Wärmeleistung mehr oder weniger an den Messkörper abgegeben. Bei schlecht wärmeleitenden Messkörpern kann es somit zu einem Temperaturunterschied zwischen Messkörper und Dehnungsmessstreifen kommen.



http://www.hbm.com

Figure 118. Gleichzeitige Messung von Temperatur und Dehnung führt zu einer Korrekturfunktion

➤ Relative Widerstandsänderung des Dehnungsmesstreifens mit Temperaturkorrektur:

$$\varepsilon = \frac{\Delta l}{l_0} = \frac{\Delta R}{k_S R_0},$$
$$k_S(T) \approx k_0 + bT + cT^2 + dT^3 + ..$$

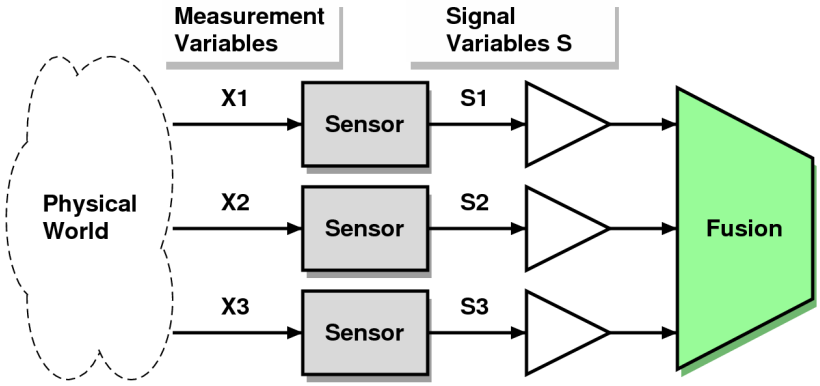


Figure 119. Multisensorfusion

Klassifizierung Fusion

Redundanz

Mehrere Sensoren messen parallel die gleiche Messgröße (Eigenschaft), z. B. mehrere Temperatursensoren messen die Temperatur eines Körpers

Vielfältigkeit

Mehrere Sensoren messen verschiedene aber korrelierte Messgrößen, z. B. gleichzeitige Messung von Temperatur, Druck und Feuchtigkeit.

Bereich

Mehrere Sensoren messen die gleiche Meßgröße aber in verschiedenen Messbereichen, z. B. mehrer Temperatursensoren messen am gleichen Ort verschiedene Temperaturbereiche.

Zeit

Aktuelle Messungen von Signalen von Sensoren werden zeitlich mit historischen Informationen korreliert, z. B. von einer früheren Kalibration.

Sensor System Konfigurationen

Komplementär

Die Sensoren sind unabhängig voneinander und vervollständigen Informationen

Konkurrierend

Jeder Sensor liefert unabhängig eine Messung der gleichen Messgröße

Kooperativ

Sensoren liefern zusammen Informationen die einzelnen nicht verfügbar wären

Schätzung (Estimation)

- Mit Schätzungstechniken lässt sich die Präzision von Messungen schlechter Qualität verbessern:
- Mittelwertbildung (Minderung der statistischen Fluktuation)
- Tiefpassfilter
- Wiener Filter
- Kalmann Filter
- Modelbasierte Filter

14. Sensoren

Ziele

- ➤ Verständnins der Definition und Klassifikation von Sensoren, Fokus mechanische Sensoren
- ➤ Rolle im Material-integrierten Sensorsystem erkennen
- ➤ Fähigkeit zur Abgrenzung: Material- vs. Struktureffekte
- ➤ Sensormaterialien und direkt materialbasierte Sensoren
- ➤ Mikrosystemtechnische Sensoren (Struktureffekte)
- ➤ Anwendung von Optische Sensoren

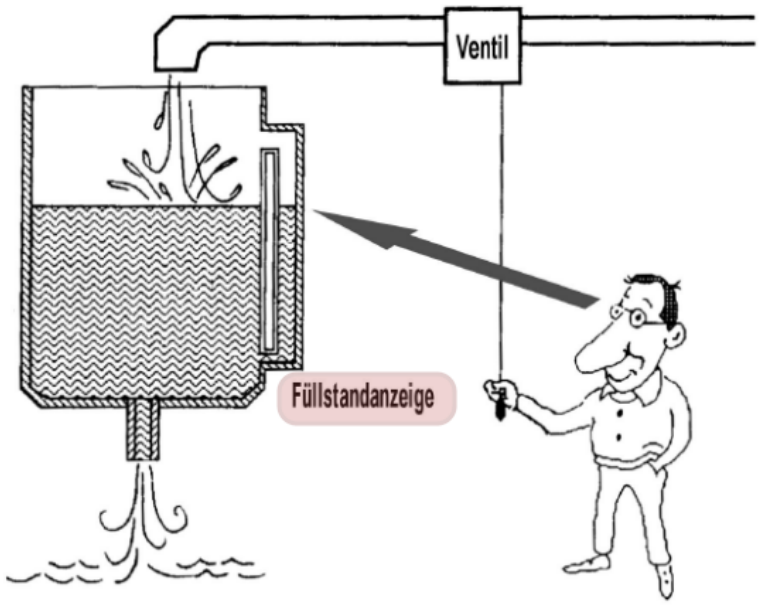
14.1. Definition Sensor

Was ist ein Sensor? [D]

> *Ein Sensor ist eine Einheit, die ein Signal oder einen Stimulus empfängt und darauf reagiert.*

"Natürlicher Sensor"

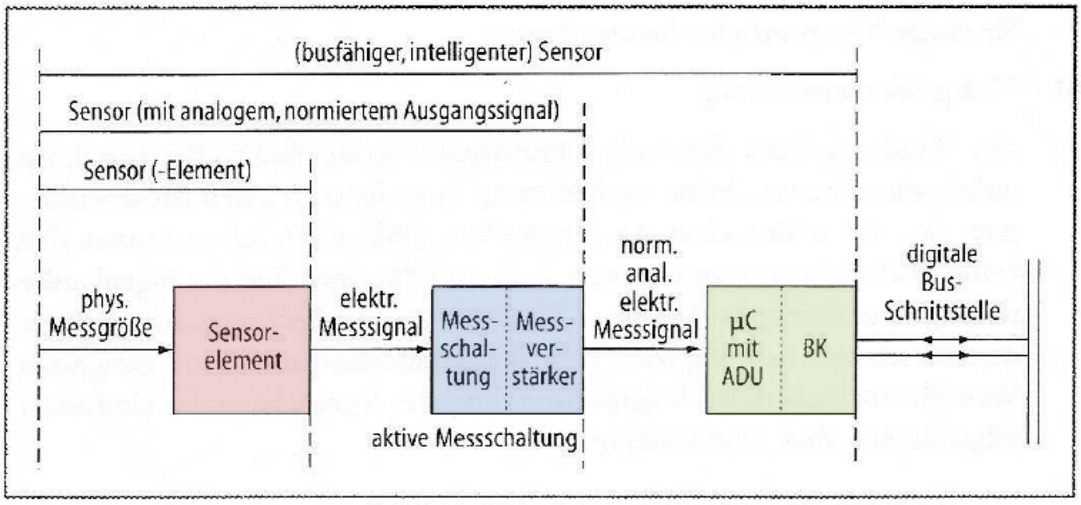
Der Sensor besteht aus zwei Teilen: (1) Füllstandanzeige (2) Menschliches Auge, das ein Signal an das Gehirn sendet.



[D]

Ein physikalischer Sensor ist eine Einheit, die ein Signal oder einen Stimulus empfängt und darauf mit einem elektrischen Signal reagiert.

Elemente, die eine im Allgemeinen nichtelektrische Messgröße in ein elektrisches Ausgangssignal umwandeln, heißen Sensoren. Dabei kann eine aktive nachgeschaltete analoge und digitale Sensorsignalverabeitung erfolgen.



[Weinrich, Grundlagen und Messprinzipien der Sensorik, Universität Hamburg]

➤ Veränderung und Unschärfe des Sensorbegriffs: Im ursprünglichen Sinne ist der Sensor nur der Meßfühler/Aufnehmer, heute oft alles, was im Gehäuse der "Sensoreinheit" mit verpackt ist.

Stimulus

➤ Ein Stimulus ist eine Größe, Eigenschaft oder Beschaffenheit, die wahrgenommen und in ein elektrisches Signal umgewandelt wird.

Ein- und Ausgangssignal

Eingangssignal

Ein Sensor wandelt ein (generell) nicht-elektrisches Signal in ein elektrisches um.

Ausgangssignal

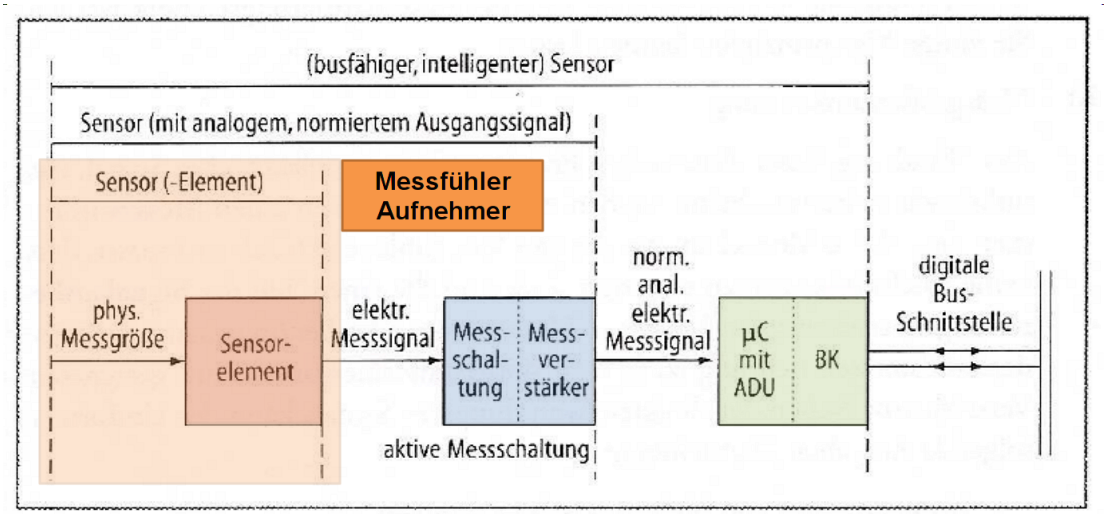
Das Ausgangssignal kann eine Spannung, ein Strom oder eine Ladung sein.

Messgrößen

Es kann weiter unterscheidbar sein durch Amplitude, Frequenz oder Phase.

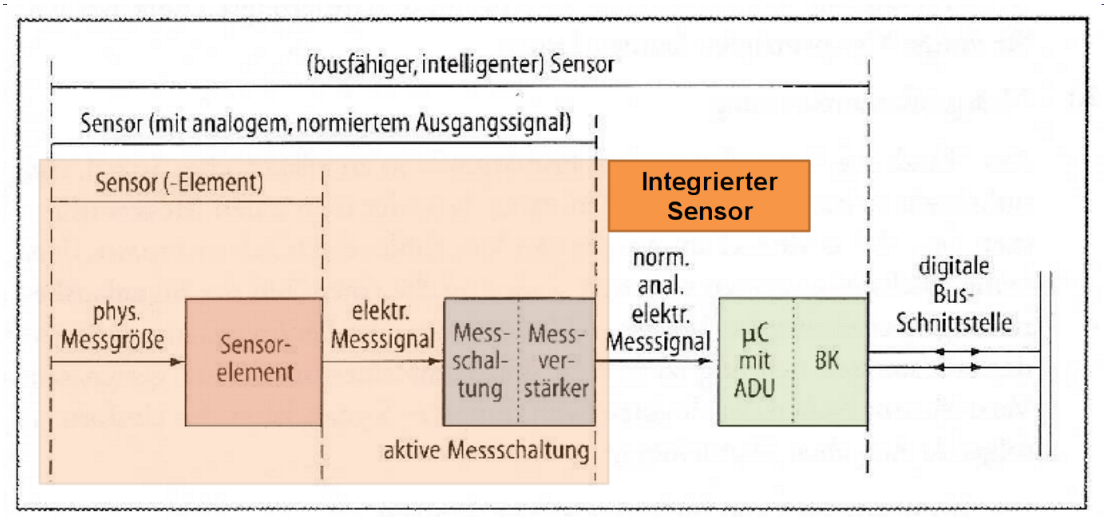
Sensorelement

➤ Ein Sensorelement (einfacher Sensor) oder Messfühler wandelt die Messgröße in eine primäre elektrische Größe um.



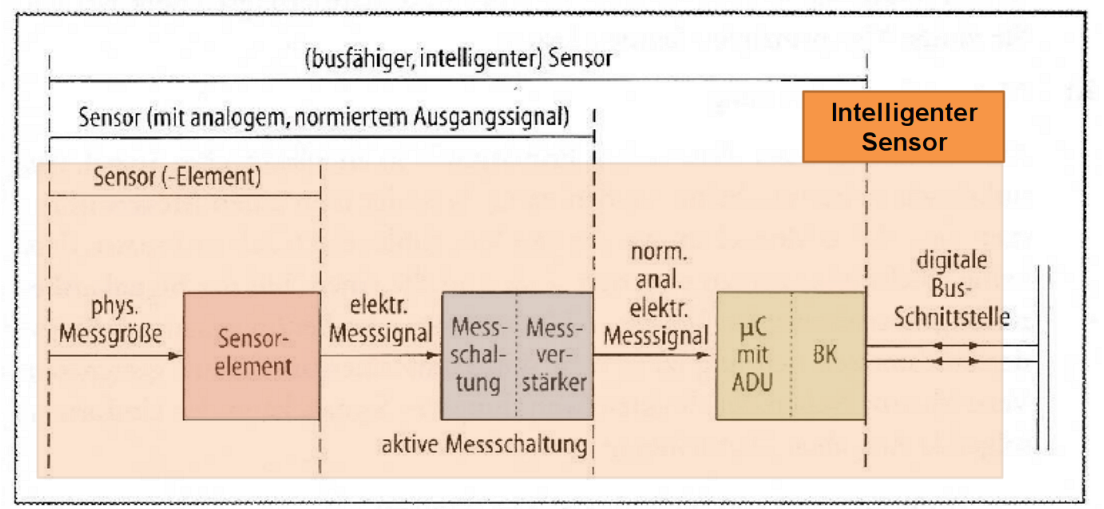
Integrierter Sensor

➤ Ein integrierter Sensor setzt die Messgröße in ein standardisiertes Signal um.



Intelligenter Sensor

➤ Ein intelligenter Sensor ist ein integrierter Sensor mit rechner- gesteuerter Auswertung und digitalisierter Ausgabe.



Sensormodell

Der Sensor als “Black Box”

Fasst man den Sensor als “Black Box” auf, kann er unabhängig von Prinzip und Aufbau u. a. über folgende Merkmale beschrieben werden:

- Transferfunktion
- Messbereichsumfang
- Ausgabebereich
- Genauigkeit
- Kalibrierungsfehler
- Hysterese
- Sättigung
- Wiederholgenauigkeit
- Verlässlichkeit
- dynamische Eigenschaften

14.2. Sensormetrik

Metrikklassen

Einteilung nach

- Art der Messgröße/des Stimulus
- Art der Erfassung der Messgröße/Messprinzip
- Art der Umwandlung von der Messgröße zum Ausgangssignal
- Material des Sensors
- Einsatzgebiet des Sensors
- Eigenschaften, Spezifikationen, Parametern (Empfindlichkeit etc.)

Einteilung nach Messgröße

- Wegsensoren
- Dehnungssensoren
- Beschleunigungssensoren
- Kraftsensoren

- Gassensoren
- Feuchtesensoren
- Temperatursensoren
- Lichtsensoren
- Magnet(feld)sensoren
- etc.

Einteilung nach Messprinzip

- resistive Sensoren (primäre elektrische Größe: Widerstand)
- induktive Sensoren (Induktivität)
- kapazitive Sensoren (Kapazität)
- piezoelektrische Sensoren (Ladung)
- thermoelektrische Sensoren (Spannung)
- optische Sensoren (z. B. Lichtintensität, weitere Wandlung erf.)
- etc.

Sensortypen

Ein System kann verschiedene Sensortypen beinhalten:

Extrinsisch
: Ermitteln von Informationen über die Systemumgebung

Intrinsisch
: Ermitteln von Informationen über den internen Systemzustand

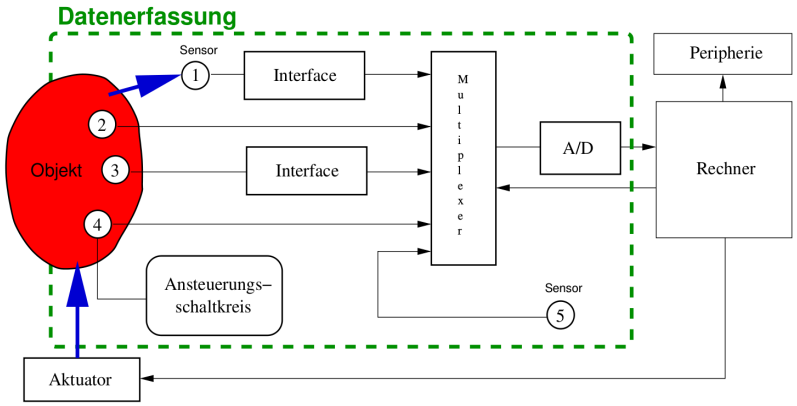
Aktive Sensoren
: Erzeugen aufgrund des Messprinzips ein elektrisches Signal (z.B. Thermoelement, Lichtsensor), d.h. variieren elektrisches Signal bei Veränderung des Stimulus

Passive Sensoren
: Enthalten passive Bauteile, deren Parameter durch die Messgröße verändert werden, d.h. bei Veränderung des Stimulus (z.B. resistive Thermometer oder Dehnungsmessstreifen)

Messverfahren

Die Differenzierung nach aktiven und passiven Sensoren kann nach dem ***Energiebedarf*** der Sensoren erfolgen:

- ➤ Passive Sensoren modulieren ein Signal unter dem Einfluss der Messgröße und benötigen daher Hilfsenergie.
- ➤ Aktive Sensoren erzeugen aufgrund des Messprinzips direkt ein elektrisches Signal.
- ➤ Aktive Sensoren können häufig in Umkehrung des Messeffektes auch als Aktoren genutzt werden. Sie liefern dafür häufig lediglich bei einer Änderung der Messgröße ein Signal (Ausnahme u. a. Thermoelem.).
- ➤ Teilweise wird die Perspektive auch umgekehrt - Sensorelemente, die keine Energiezufuhr benötigen, werden dann als "passiv" bezeichnet.



Sensortypen:	
1.: extrinsisch, passiv	2. und 3.: intrinsisch, passiv
4.: intrinsisch, aktiv	5.: intrinsisch (in der Datenerfassung), passiv

Figure 120. Aktive, passive, intrinsisische, und extrinsische Sensoren in einem Messsystem [D]

14.3. Sensormodell

Eigenschaften von Sensoren

- ➤ Ein Eingangssignal muss eventuell mehrmals konvertiert werden, bis der Sensor ein elektrisches Ausgangssignal ausgibt.

- Im folgenden Abschnitt wird der Sensor als 'Black Box' betrachtet.
- Es interessiert uns im Folgenden nur die Beziehung zwischen Eingangs- und Ausgangssignal.

Transferfunktion

- Jeder Sensor besitzt eine ideale bzw. theoretische Beziehung zwischen Eingangs- und Ausgangssignal.
- Das Ausgangssignal S repräsentiert dabei den wahren Wert des Eingangssignals s.
- Die ideale Beziehung zwischen Eingangs- und Ausgangssignal eines Sensors wird beschrieben durch die Transferfunktion $S = f(s)$.

Transferfunktion

- Lineare Transferfunktion: $S = a + bs$
- Logarithmische Transferfunktion: $S = a + k \ln s$
- Exponentiale Transferfunktion: $S = a \cdot e^{ks}$
- beliebige Polynome höherer Ordnung: $S = a_0 + a_1 s^1 + a_2 s^2$..mit
- k ist eine Konstante
- a ist das Ausgangssignal bei einem Eingangssignal von 0
- b ist die Steigung
- b wird in diesem Zusammenhang oft als **Sensitivität** bezeichnet.

Sensitivität

Für nicht-lineare Transferfunktionen ist die Sensitivität für jeden Eingangswert s_i wie folgt definiert:

$$b = \frac{dS(s_i)}{ds}$$

Approximation einer Transferfunktion

- Einige nicht-lineare Transferfunktionen sind linear in einem eingeschränkten Bereich.

- Nicht-lineare Transferfunktionen können durch mehrere lineare Funktionen approximiert werden.
- Die Differenz zwischen wahrem und linear approximiertem Ausgangssignal sollte unter einem zu spezifizierenden Limit liegen.

Mehrdimensionale Transferfunktionen

- Die Transferfunktion kann von mehr als einem Stimulus abhängen.
  - Beispiel: Infrarot-Wärmestrahlungssensor (Stefan-Boltzmann Gesetz)

$$U = G(T_b^4 - T_s^4)$$

mit

- G: Konstante
- T_b: absolute Temperatur des gemessenen Objektes
- T_s: absolute Temperatur der Sensoroberfläche
- U: Ausgangsspannung
- Sensitivität in Bezug auf die Temperatur des gemessenen Objektes:

$$b = \frac{dU}{dT_b} = 4GT_b^3$$

Messbereichsumfang

- Der dynamische Bereich eines Stimulus, der von einem Sensor erfasst wird, wird Messbereichsumfang (engl. Span oder Full Scale Input) genannt.
  - beziffert den kleinsten und höchsten für einen Sensor zulässigen Stimuluswert
  - größere Stimuli können den Sensor beschädigen
  - kleiner oder größere Stimuli können zu einer Sättigung des Ausgangssignals führen

Ausgabebereich

➤ Der Ausgabebereich (endl. Full Scale Output) eines Sensors ist das Intervall zwischen dem Ausgangssignal bei kleinstem und größtem angelegten Stimulus.

Reale Transferfunktion

➤ Im Vergleich zur idealen Transferfunktion sind reale Sensoren immer ungenau.

➤ Die Transferfunktion eines realen Sensors heißt daher: reale Transferfunktion.

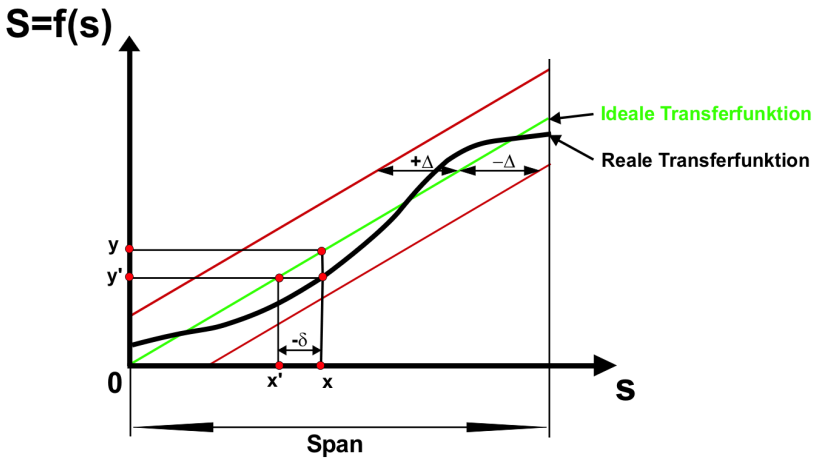


Figure 121. Abweichung der realen Transferfunktion von der idealisierten Modellfunktion innerhalb von Ober- und Untergrenzen [D]

Kalibration

➤ Kalibration ist eine Korrektur der Transferfunktion (Anpassung der Parametrisierung)

➤ Kalibration soll die reale Transferfunktion der idealen Transferfunktion anpassen

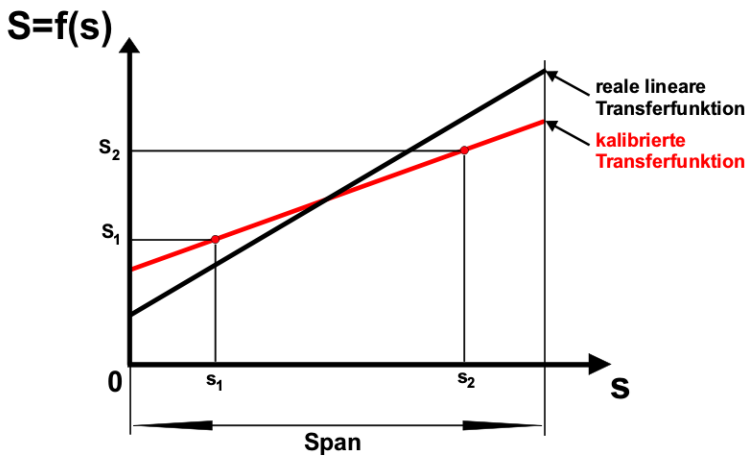


Figure 122. Vergleich der realen und kalibrierten Transferfunktion eines Sensors

Hysterese

- Hysterese beschreibt ein zeit- und historienabhängiges Verhalten der Transferfunktion.
- D.h. der Stimulus wird erst in eine Richtung verändert (vergrößert) und dann wieder in die andere Richtung auf den ursprünglichen Wert geändert (verkleinert) . Dabei ist das Ausgangssignal des Sensors aber nicht mehr gleich.

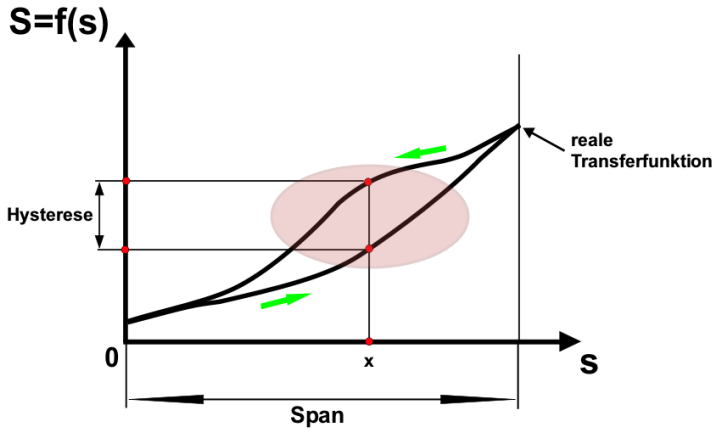


Figure 123. Hysteresverhalten der Sensortransferfunktion [D]

Sättigung

- ➤ Fast jeder Sensor hat Arbeitsbereichsgrenzen.
- ➤ Viele Sensoren haben eine lineare Transferfunktion, . . .
  - ❑ aber: Ab einem bestimmten Stimuluswert wird nicht mehr die gewünschten Ausgabe erzeugt.
  - ❑ Es tritt eine Sättigung der realen Transferfunktion ein.

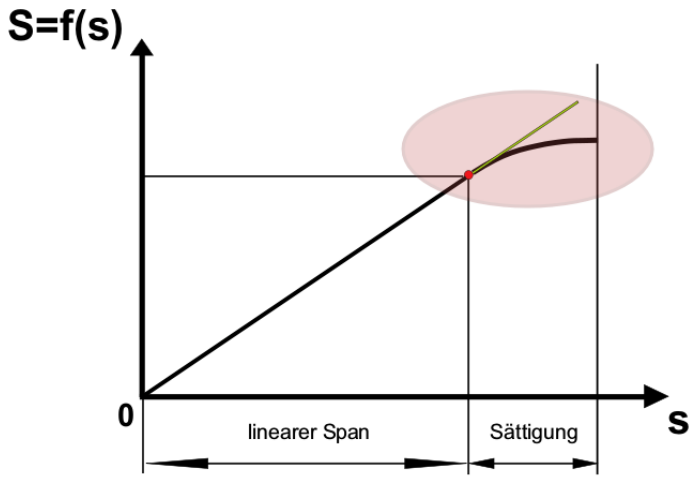
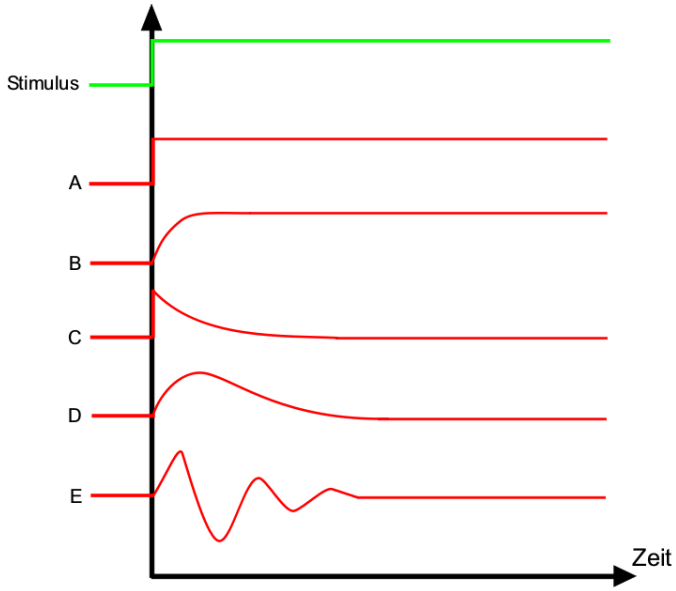


Figure 124. Sättigung der Transferfunktion

Antwortverhalten

→ Zeitliches Verhalten

- ➤ Es kann eine verzögerte Stimulusantwort auftreten
- ➤ Es kann zeitliches Über- und Unterschwingen auftreten
- ➤ Es kann Oszillation auftreten



Umwelteinflüsse

- minimal und maximal zulässige Umgebungstemperatur
- minimal und maximal zulässige Luftfeuchtigkeit
- Kurz- und Langzeitstabilität (Drift) (Hilfe bei Langzeitdrift: Pre-aging erhöht Stabilität)
- statische und dynamische Änderungen von elektromagnetischen Feldern, Gravitationskräften, Vibrationen, Strahlung , etc.
- Selbsterwärmung z.B. durch Stromfluss
- Mechanischer Stress im Material durch Integration!
- Energieversorgung

14.4. SHM: Materialintegrierte Dehnungssensoren

Fibre-Bragg-Gratings (FBG)

- FBGs können in Materialien eingebettet werden (Verbindung)
- Es tritt eine Gitterkon.- und Wellenlängenänderung im reflektierten Spektrum aufgrund einer mechanischen Dehnung der Faser auf

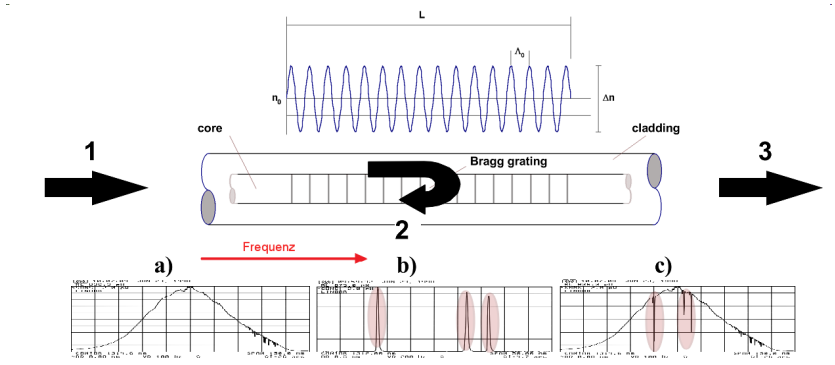
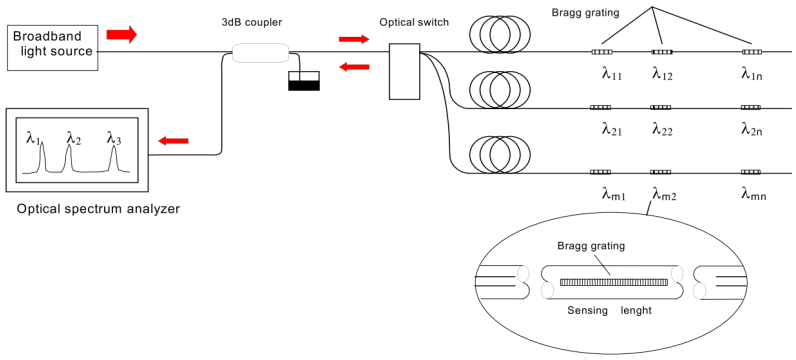


Figure 125. Schema eines Faser-Bragg-Gitter- und Funktionsprinzips. a) Intensitätsspektrum einer in die Faser eingebrachten Breitbandquelle. b) Spektren werden von drei Faser-Bragg-Gittern reflektiert. c) Transmissionsspektrum nach Passieren der drei Bragg-Gitter [E]

- ➤ Es können mehrere Sensoren örtlich getrennt in Materialien und Strukturen verlegt werden
- ➤ Die Signale können einzelnen über einen Multiplexer ausgewertet werden oder man verwendet verschiedene FBG die sich in der Gitterkonstanten unterscheiden
- ➤ Eine Lichtfaser kann dabei zudem aus mehereren hintereinander angeordneten FBG bestehen die ebenfalls unterschiedliche Gitterkonstanten und somit Inteferenzfrequenzen besitzen → quasi-gleichzeitge Auswertung mehrerer Frequenzen

Multiplexing und Multifrequenzfasern [E]



]]

- ➤ Die relative Wellenlängenänderung $\delta\lambda$ eines FBG durch axiale Dehnung ϵ

ist gegeben durch:

$$\frac{\delta\lambda}{\lambda_0} = k_\varepsilon \Delta\varepsilon + k_T \Delta T$$

➤ Dabei gibt ebenso eine Temperaturabhängigkeit T die rechnerisch kompensiert werden muss → Sensorfusion

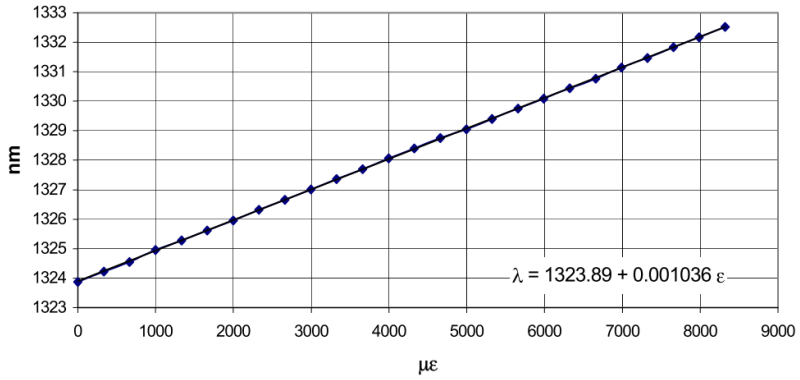


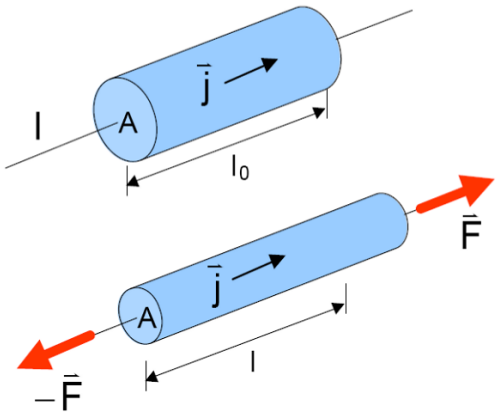
Figure 126. Beispiel einer Wellenlängenverschiebung durch Dehnung [E]

14.5. Piezoresistivität

➤ Piezoresistivität ist die durch Dehnung verursachte Änderung des elektrischen Widerstands

➤ Beispiel: Dehnungsmessstreifen

Widerstandsänderung infolge mechanischer Belastung (passiver, modulierender Effekt).



Quelle: Vorlesung IZFM, Universität Stuttgart

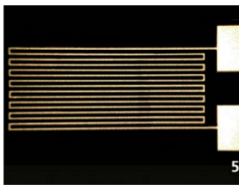
$$R = \rho \cdot \frac{l_0}{A_0}$$

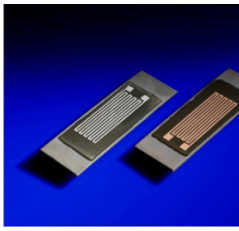
$$\sigma = E \cdot \frac{\Delta l}{l}$$

$$R_F = \rho \cdot \frac{l_F}{A_F}$$

$$\Delta R / R = K \, \Delta l / l$$

k-Faktor: Empfindlichkeit des DMS





Gedruckte Dehnungsmess
Fraunhofer IFAM

Fraunh

- Dabei gibt es verschiedene Beiträge (Terme) zu relativen Widerstandsänderung durch:
  - Änderung des spezifischen WIderstandes
  - Änderung der Länge
  - Änderung des Querschnitts (Fläche)

$$\frac{\Delta R}{R} = \frac{1}{R} \cdot \left[\frac{\partial R}{\partial \rho} \cdot \Delta\rho + \frac{\partial R}{\partial l} \cdot \Delta l + \frac{\partial R}{\partial r} \cdot \Delta r \right] = \frac{\Delta\rho}{\rho} + \frac{\Delta l}{l} - \frac{2 \cdot \Delta r}{r} = \left[\frac{\Delta\rho/\rho}{\varepsilon} + 1 + 2 \cdot \mu \right] \cdot \varepsilon = k \cdot \varepsilon$$

Beitrag der Radius- bzw. Querschnittsänderung

Beitrag der Längenänderung

Beitrag der Änderung des spez. Widerstandes

Längendehnung	$\varepsilon = \Delta l / l$
Dehungsempfindlichkeit	k
Poisson-Zahl	$\mu = -\frac{\Delta r/r}{\Delta l/l}$

Metallische Dehnungsmessstreifen (DMS)

- Widerstandsänderung dominiert durch Geometrieänderung.
- gute Linearität in einem weiten Temperaturbereich (-50 ... 200°C, Pt-DMS bis ca. 1000°C)
- relativ große Dehnungen (ca. 0,5 %) zulässig
- geringe Empfindlichkeit (k-Faktoren: Konstantan 2.05, Ni80Cr20 2.2, Pt92W8 4.0, Pt 6.0)
- S-T-C: Self-temperature-compensated, d. h. Auswahl des DMS-Materials für einen best. Werkstoff des Messobjektes zur Kompensation unterschiedlicher Wärmeausdehnungskoeffizienten durch Temperaturabhängigkeit von

Halbleiter-DMS

- Widerstandsänderung dominiert durch Änderung des spezifischen Widerstandes ρ: Änderung der Besetzungswahrscheinlichkeit/dichte von Valenz- und Leitungsbändern und der Ladungsträgerbeweglichkeit durch mechanische Spannung,
- zusätzliche Abhängigkeit der Änderung von ρ von der kristallographischen Orientierung (u. a. bei Si).
- höhere Temperaturempfindlichkeit als bei vielen Metall-DMS

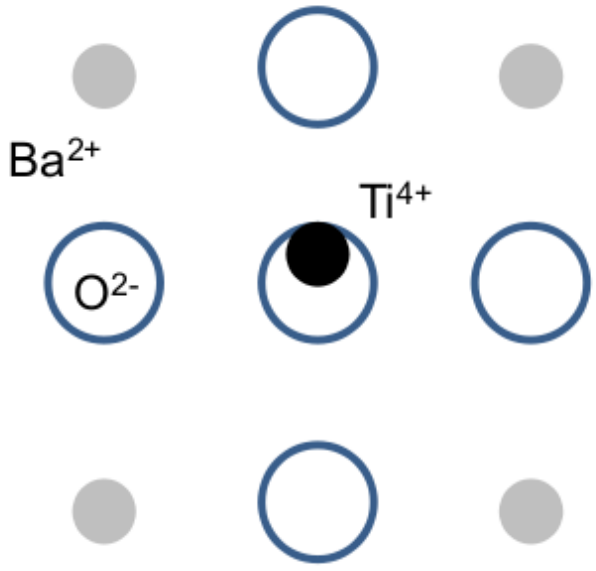
- relativ geringe Dehnungen (ca. 0,1 %) zulässig
- hohe Empfindlichkeit (k-Faktoren: Si B-dot./p 80…190, Si P-dot./n -25…-100)

14.6. Piezoelektrizität

- Piezoelektrizität bedeutet dass bei einer Dehnung bzw. Verformung ein elektrisches Feld an der Oberfläche aufgebaut wird

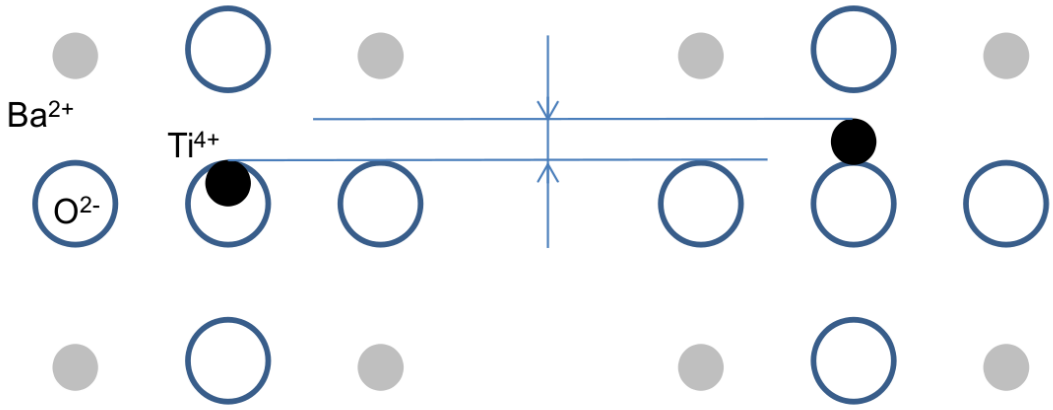
Der Begriff Piezoelektrizität beschreibt die Ausbildung eines elektrischen Feldes in einem Material durch mechanische Belastung. Der Effekt ist umkehrbar, d.h. aus einem äußeren elektrischen Feld folgt eine mechanische Dehnung des Materials. Basis ist der Aufbau des Materials aus Ionen oder polaren Molekülen und die Kristallstruktur, die nicht zentrosymmetrisch sein darf.

Klassisches Materialbeispiel ist Bariumtitanat BaTiO3. In der Elementarzelle kann das Ti4+-Ion eine von 6 verschiedenen Positionen minimaler Energie annehmen, die alle nicht zentrisch sind und damit zu einem Dipol-Charakter der Elementarzelle führen.



- Oberhalb der Curie-Temperatur sind die Dipolmomente der einzelnen Elementarzellen zufällig verteilt.

- Unterhalb der Curie-Temperatur (BaTiO3 ca. 380 K) treten die Dipole in Wechselwirkung, richten sich aneinander aus und nehmen dabei bevorzugte Orientierungen ein:
  - Damit kommt es zur Ausbildung von Domänen im Kristall, die aber noch zufällig verteilt sind und sich bezogen auf den Gesamtkristall ausgleichen.
- Wird das Material einem äußeren elektrischen Feld ausgesetzt, wachsen die Domänen, deren Dipole parallel zum Feld ausgerichtet sind, auf Kosten der anderen: Das Material wird polarisiert.
- Wird ein derart polarisiertes piezoelektrisches Material einer mechanischen Belastung ausgesetzt, verändert sich die azentrische Lage der Ionen in den Elementarzellen
  - Wenn diese in Folge einer vorherigen Polarisierung parallel gerichtet sind, geschieht dies in allen Elementarzellen in gleicher Richtung und damit verstärkt.
- Dies führt zu einem messbaren elektrischen Spannungsausschlag, dessen Höhe von der der mechanischen Belastung abhängt.



14.7. Ferroika

- Ferroika sind Materialien, die unterhalb der sog. Curie-Temperatur spontane Ordnungszustände mit langer Reichweite zeigen (Ausbildung einer Domänenstruktur).

➤ Die Bereichsgrenzen/Domänenwände sind dabei durch äußere Einflüsse veränderbar.

Ferromagnetismus (Einflussgröße magnetisches Feld)

Ausrichtung der magnetischen Momente, u. a. Fe, Ni, Co.

Ferroelektrizität (elektr. Feld)

Ausrichtung elektrischer Dipolmomente, u. a. piezoelektrische Materialien wie BaTiO3, PZT.

Ferroelastizität (mech. Spannung)

Übereinstimmende kristallographische Orientierung in durch Zwillingsgrenzen begrenzten Domänen, makroskopische Dehnung als Folge einer Gleichrichtung über den gesamten Kristall, mechanisch induzierte martensitische Phasenumwandlungen, z. B. in Formgedächtnislegierungen.

14.8. Piezoelektrizität vs. Ferroelektrizität

➤ Piezoelektrizität benennt die Verknüpfung zwischen elektrischem Feld und mechanischer Dehnung. Besondere Piezoelektrika sind Pyroelektrika, die bei Temperaturänderung Ladungstrennung zeigen.

➤ Technisch für Aktor-/Sensoranwendungen interessante Piezoelektrika sind gewöhnlich Ferroelektrika, eine Untergruppe der Pyroelektrika, die als einzige die parallele Ausrichtung der Domänen mittels eines elektrischen Feldes erlaubt.

❏ Beispiel: Bariumtitanat BaTiO3, Blei-Zirkon-Titanat Pb(ZrxTi1-x)O3 (PZT)

14.9. Wandler- und verknüpfte Effekte

➤ Grundsätzlich ist der Zustand eines Kristalls in Bezug auf seine thermischen, elastischen und elektrischen Eigenschaften durch die Angabe je einer (entsprechenden) Zustandsgröße bestimmt [F]:

- ❑ Die **thermische Zustandsgröße** ist ein Skalar, die
- ❑ **elektrische Größe** ein Vektor [3 Vektorkoordinaten], die
- ❑ **elastische Größe** ein symmetrischer Tensor zweiter Stufe (6 Tensorkoordinaten, d.h. 10 unabhängige Vari- ablen zur vollst. Beschreibung).

Heckmann-Diagramm

➤ Darstellung linearer Beziehungen (über Materialkonstanten) zwischen

- ❑ mechanischen,
- ❑ elektrischen und
- ❑ thermischen Zustandsgrößen in dielektrischen Materialien.

➤ Äußeres Dreieck, Ecken: Intensive, d. h. materialmengenunabhängige Zustandsgrößen.

➤ Inneres Dreieck, Ecken: Extensive, d. h. materialmengenabhängige Zustandsgrößen.

➤ Markierte Verbindungen: Thermische, dielektrische und elastische Haupteffekte.

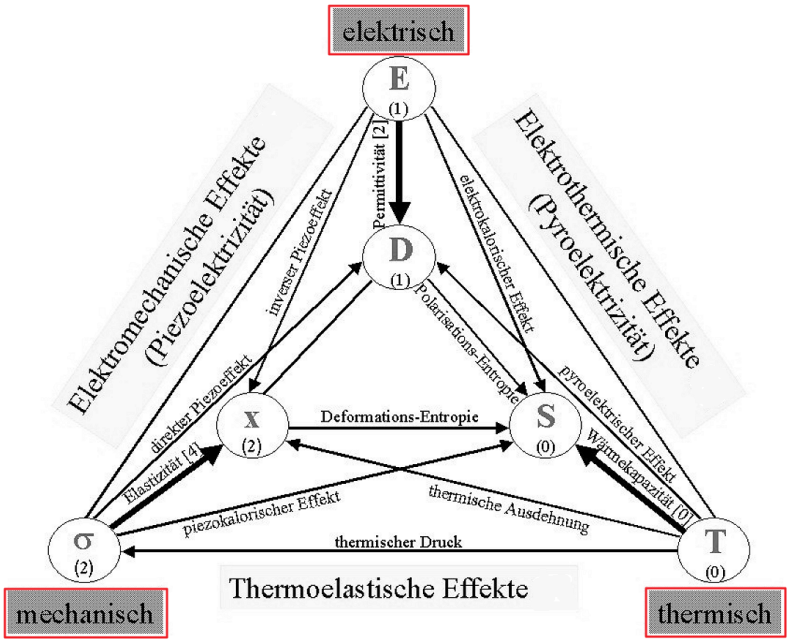


Figure 127. Heckmann-Diagramm in der Darstellung nach [Nye, 1985; Sutter, 2005]

Linearisiertes Heckmann-Diagramm

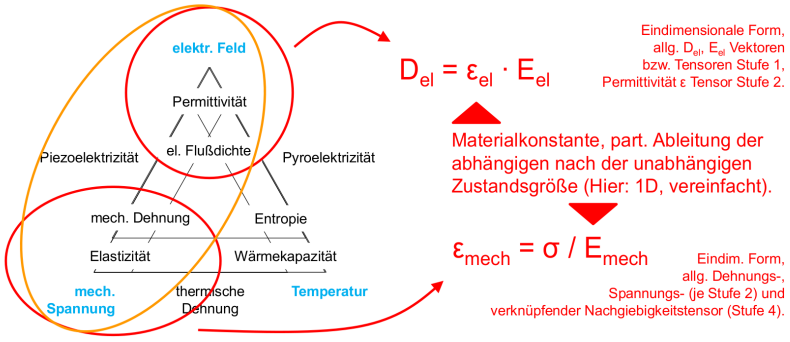


Figure 128. Heckmann-Diagramm: Beispiele für Haupteffekte, linearisierte Form.

Piezoelektrischer Effekt im Heckmann-Diagramm

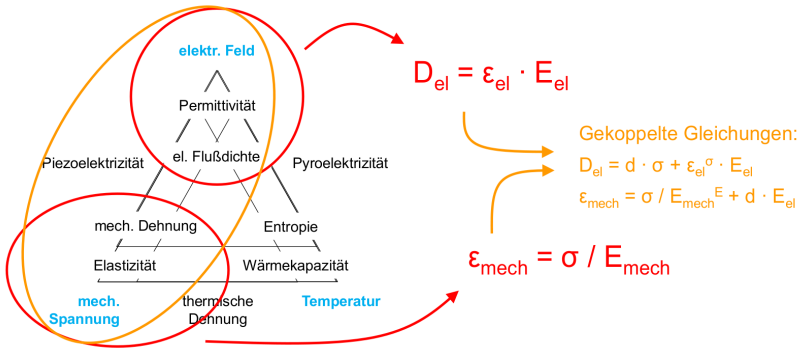


Figure 129. Heckmann-Diagramm: Piezoelektrischer Effekt.

Magnetische Effekte im Heckmann-Diagramm

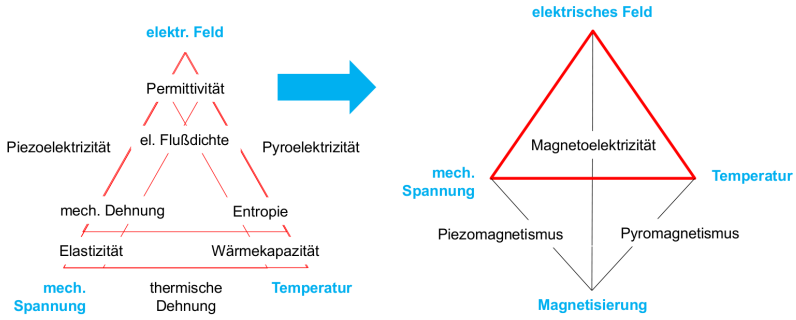


Figure 130. Heckmann-Diagramm: Erweiterung um magnetische Effekte.

Eigenschaftsverknüpfungen

➤ Verknüpfung in der Form: $Response = f(Input)$.

Input	Charge / Current	Magnetization	Strain	Temperature	Light
Electric Field	Permittivity Conductivity	Electromagnet. Effect	Converse Piezo-electric Effect	Electrocaloric Effect	Electrooptic Effect
Magnetic Field	Magnetoelectric Effect	Permeability	Magnetostriction	Magnetocaloric Effect	Magnetooptic Effect
Stress	Piezoelectric Effect	Piezomagnetic Effect	Elastic Constants	Friction	Photoelastic Effect
Heat	Pyroelectric Effect	Curie-Weiss Effect	Thermal Expansion	Specific Heat	Thermo-luminescence
Light	Photovoltaic Effect	(Absorption + Pyroelectric)	Photostriction	Absorption + Curie Weiss	Refractive Index

Figure 131. Die Darstellung entspricht dem Heckmann-Diagramm erweitert um magnetische/optische Effekte, aber ohne Transportphänomene. Diagonale analog Haupteffekten im Heckmann-Diagramm [Uchino, Giniewicz, 2003]

Transportphänomene

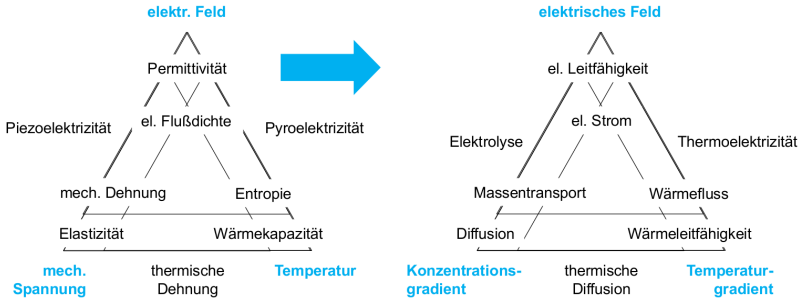


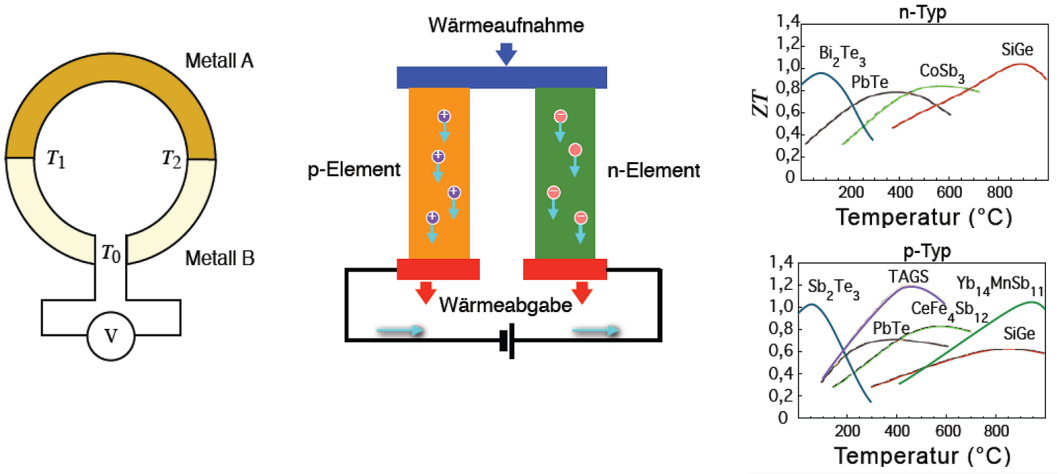
Figure 132. Heckmann-Diagramm: Analoge Darstellung für Transportphänomene.

14.10. Thermoelektrizität

Seebeck-Effekt

- Spannungsabfall in einem Stromkreis aus zwei unterschiedlichen elektrischen Leitern bei Vorliegen einer Temperaturdifferenz zwischen den Kontaktstellen.
- Umkehrbarer Effekt, nutzbar zur Energiegewinnung aus Temperaturdifferenzen oder zur aktiven Kühlung (Peltier-Elem.)

➤ Wirkungsgrade werden ausgedrückt durch Gütefaktor ZT, derzeit noch beschränkt.



[Leipner, Martin-Luther-Universität Halle-Wittenberg]

➤ Höhe des Spannungsabfalls U ist abhängig von Temperaturdifferenz T_2-T_1 und Seebeck-Koeffizient S

➤ Der thermoelektrischer Gütefaktor ZT als Ausdruck für den Wirkungsgrad ist abhängig vom der mittleren Arbeitstemperatur T, der elektischen leitfähigkeit σ, und der Wärmeleitfähigkeit λ:

$$U = \int_{T_1}^{T_2} S dT, ZT = \frac{S^2\sigma}{\lambda} T$$

➤ Effiziente thermoelektrische Materialien erfordern hohe elektrische und geringe thermische Leitfähigkeit – scheinbarer Widerspruch zu Wiedemann-Franz-Gesetz.

- ❏ Konstituiert einen linearen Zusammenhang zwischen elektrischer und thermischer Leitfähigkeit;
- ❏ Allerdings nur für den elektronischen Anteil;
- ❏ Der phononische Anteil kann unabhängig von der elektrischen Leitfähigkeit beeinflusst werden.

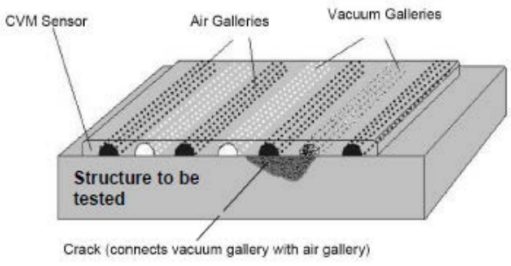
14.11. Material- vs. Struktureffekte

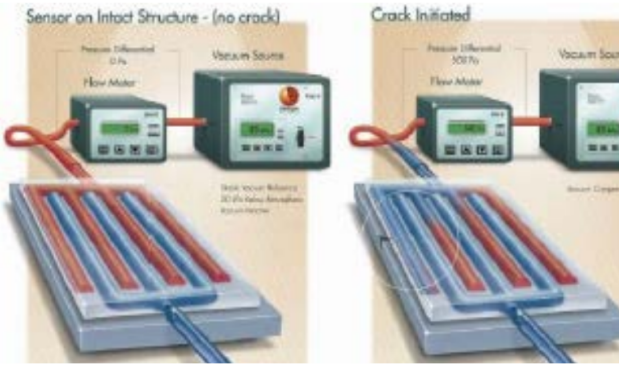
"Binäre Sensoren": Reißdraht

- ➤ Reißdrähte sind die einfachste Form einer Schadenssensorik im Bereich Strukturüberwachung:
  - ❑ Es handelt sich um elektrische Leiter, die an kritischen Bereichen einer mechanisch belasteten Struktur angebracht sind und die bei Auftreten eines Schadens (Rissinitiierung, Überdehnung o. ä.) reißen.
- ➤ Damit wird die elektrische Verbindung getrennt, der Sensor liefert mithin eine binäre Information, je nach Auslegung z.B. hinsichtlich Überschreitung einer Belastungsgrenze.

Comparative Vaccuum Monitoring

- ➤ Der Sensor wird in der Regel auf eine Strukturoberfläche aufgebracht, die im rissgefährdeten Bereich liegt.
  - ❑ Ein Riss, der z.B. durch Vakuum- und Referenzdruck-Galerien verläuft, führt dazu, dass in den ersteren kein Vakuum mehr gehalten wird.

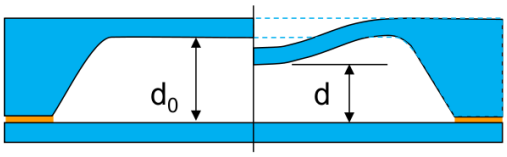




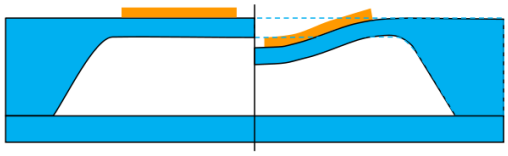
Drucksensoren

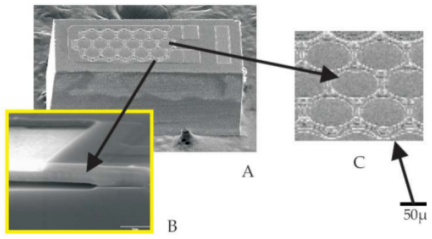
1. Drucksensorik mit einem Sensoreffekt als Struktureigenschaft (MEMS)

Kapazitiv: $C = \varepsilon_0 \cdot \varepsilon_r \cdot A/d$



Membran & Dehnungssenorik:





- piezoresistive (DMS bzw. Dehnungs-messstreifen) oder
- piezoelektrische Dehnungssensoren

→ Sensormaterialien

➤ Prinzipiell materialunabhängige Eigenschaften einer Struktur:

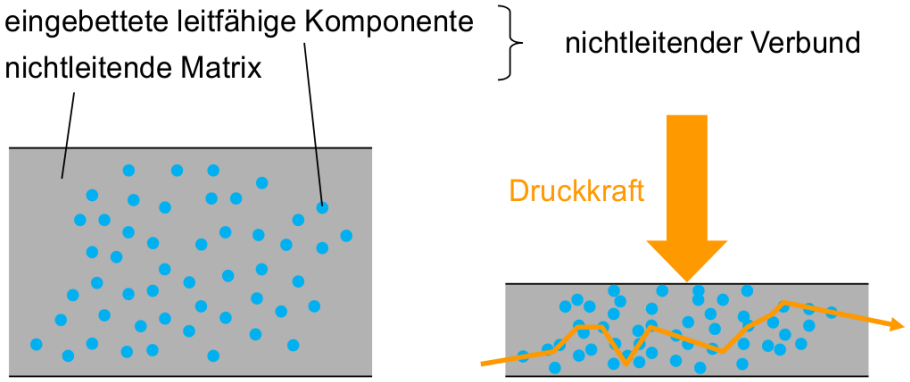
- ❏ kapazitiver Drucksensor

➤ Intrinsische Materialeigenschaft des Sensormaterials:

- ❏ klassischer piezoresistiver Effekt, z. B. in Halbleiterwerkstoffen
- ❏ piezoelektrischer Effekt

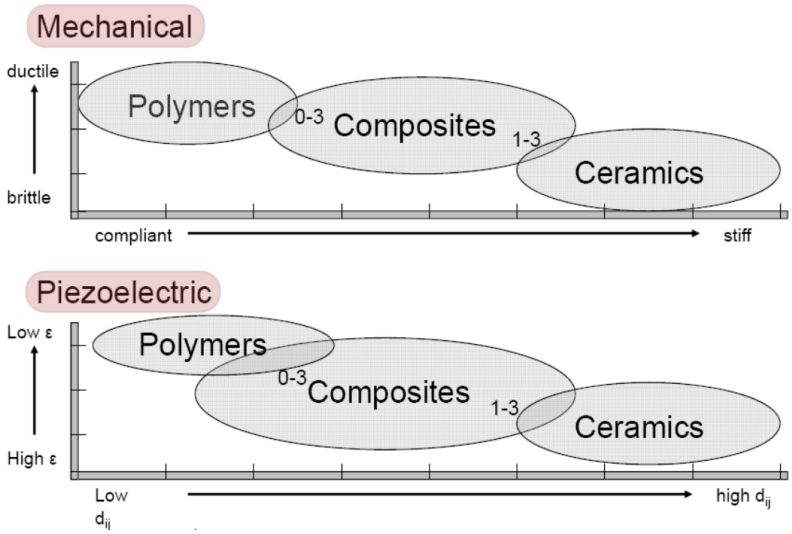
2. Drucksensorik: Sensoreffekt durch Strukturierung

➤ Verbundwerkstoffe als Sensormaterialien und Verbundstruktur als Basis von Sensoreffekten (Perkolation)



➤ Verbundwerkstoffe als Sensormaterialien und Verbundstruktur zur Anpassung sekundärer Eigenschaften an Messaufgabe und -umgebung

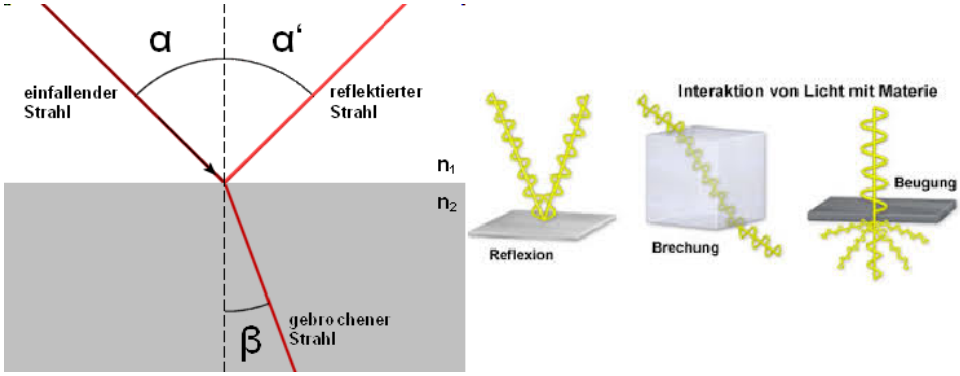
- ❏ Beispiel piezoelektrische Polymermatrix-Verbunde;
- ❏ Erhöhte Duktilität gegenüber Keramik;
- ❏ Höhere Temperaturstabilität, Feuchteresistenz und Steifigkeit gegenüber Polymeren.



[Zwaag et al., 2010]

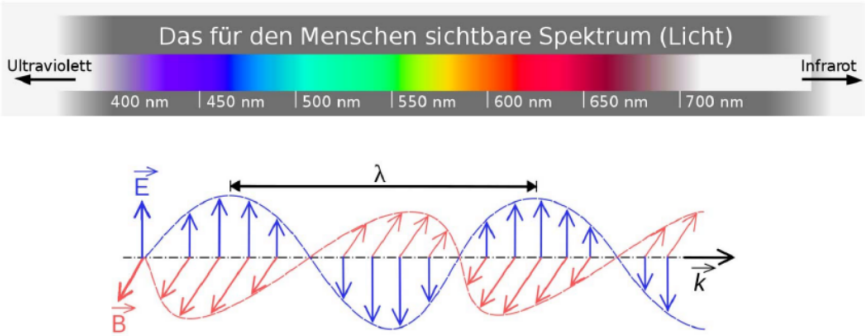
14.12. Optische Sensoren

- ➤ Prinzip: Nutzung einer Beeinflussung des Lichts
- ➤ Voraussetzung: Kontrolle über den Weg des Lichts.
- ➤ Phäomene u. a.:
  - ❏ Brechung: Richtungsänderung an Grenzflächen
  - ❏ Reflexion: Spiegelung an Grenzflächen
  - ❏ Beugung: Interferenzeffekte



[Wikipedia]

- ➤ Nutzung einer Beeinflussung des Lichts als Sensor:
  1. Intensität
  2. Farbe (Wellenlänge λ)
  3. Polarisation (Schwingungsrichtung)



[Wikipedia]

Lichtwellenleiter

- ➤ Häugigster mechano-optischer Sensor ist der Lichtwellenleiter (Glasfaser)
- ➤ Ein Lichtwellenleiters besteht aus unterschiedlichen Bereichen:
  - ❏ Mantel und
  - ❏ Kern, mit jeweils unterschiedlichen optischen Eigenschaften (Brechungsidnex)

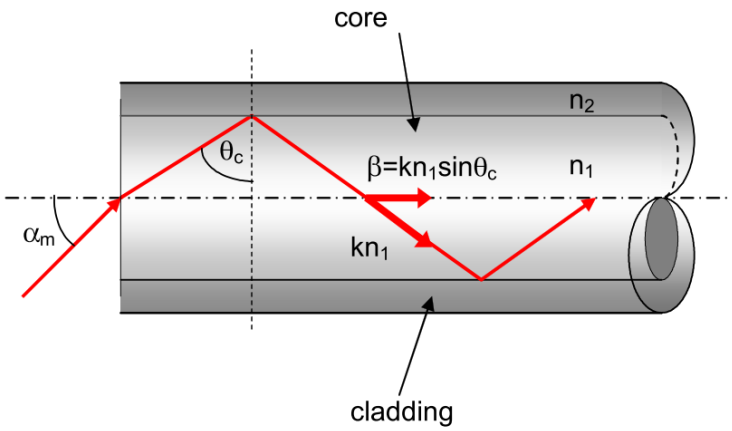


Figure 133. Schematischer Aufbau eines Lichtwellenleiters mit Mantel und Kern [E]

Intensitätsbasierte Sensoren

➤ Abstände von LWL können durch Intensitätsänderung detektiert werden

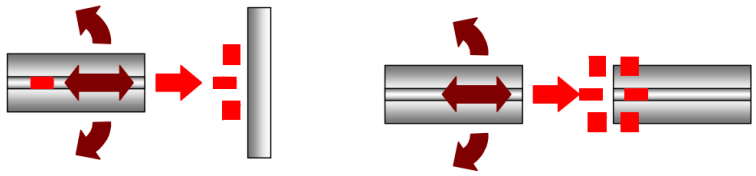
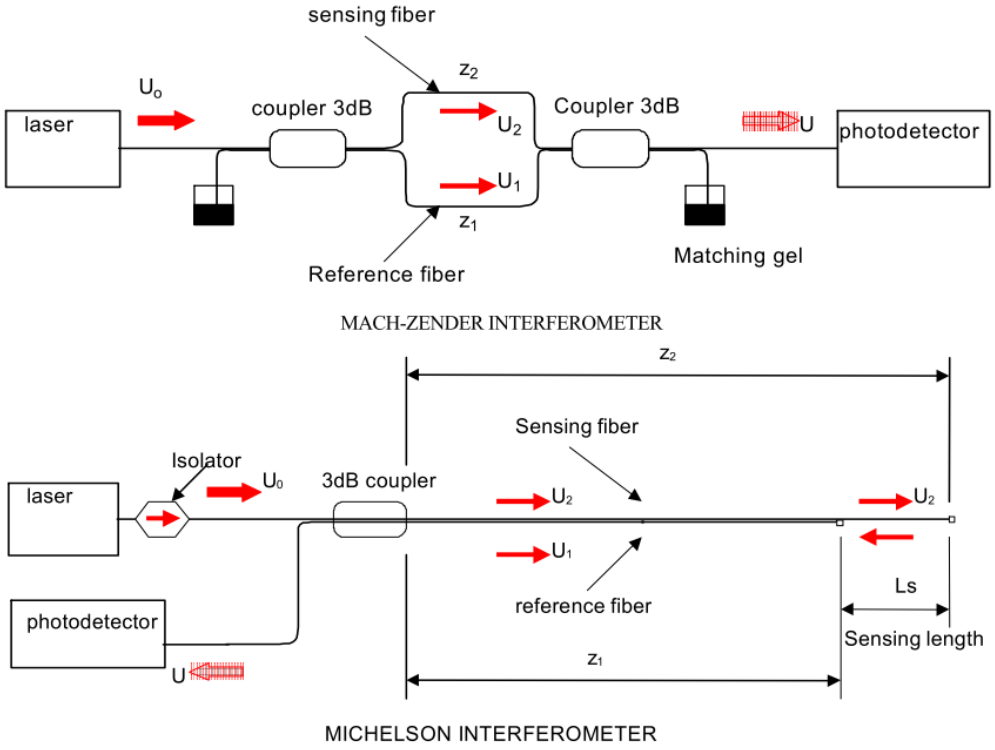


Figure 134. Faseroptischer Näherungssensor [E]

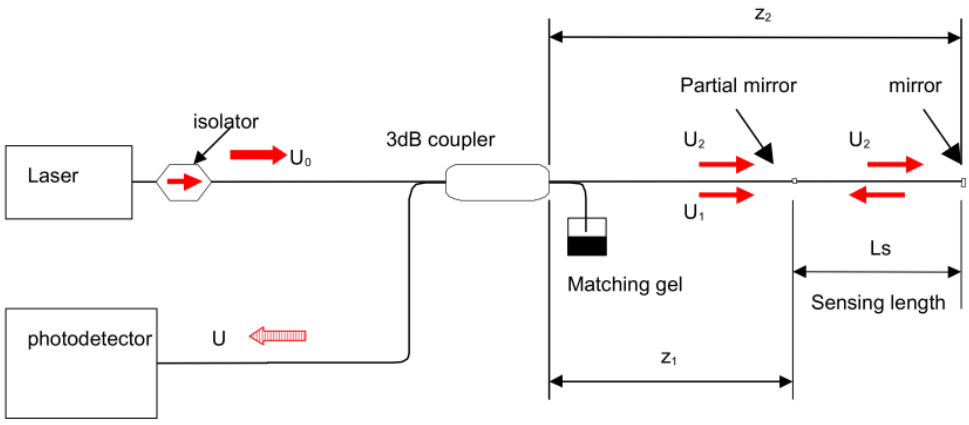
Phasenmodulierte Sensoren

➤ Basieren auf Interferenzeigenschaften des Lichtes → Interferometer

➤ Interferrenz ist eine ortsaufgelöste Eigenschaft → mechanischer Sensor



[E]

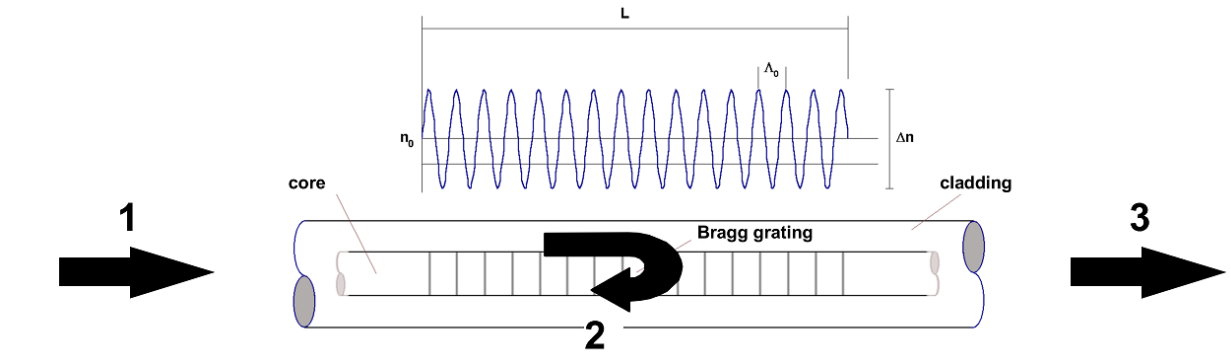


FABRY-PEROT INTERFEROMETER

[E]

Wellenlängenbasierte Sensoren

- ➤ Fasergitter (FBG)
  - ❑ Die Gitter können mittels Laserstrukturiering in die Faser eingebschrieben werden
  - ❑ Das Gitter ist i.A. auf einen kleinen Abschnitt zwecks Lokalisierung der Dehnung begrenzt
  - ❑ Es können eine Vielzahl von Gitter entlang der Faser in Reihe angeordnet werden, die sich durch ihre Gitterkonstante und somit die Resonanzfrequenzen unterscheiden → Multispektrale Messung an mehreren Orten gleichzeitg



15. Sensornetzwerke

15.1. Verteilte Sensornetzwerke

(Distributed Sensor Networks DSN)

- Ein verteiltes Sensornetzwerk besteht aus einer Menge von Sensor Knoten $N=\{n_1,n_2,...\}$, die in einem Kommunikationsnetzwerk miteinander verbunden sind.
- Ein Sensorknoten besteht aus:
  - Einer Menge von Sensoren $S=\{s_1,s_2,...\}$, wenigstens einem Sensor s (inkl. virtuellen),
  - Analoger Signalverarbeitung inkl. ADC $A=\{a_1,a_2,...\}$
  - Digitale Signal- und Datenverarbeitungeinheiten $P=\{p_1,p_2,..\}$ und Speicher $M=\{m_1,..\}$
  - Kommunikationsmodule $C=\{c_1,c_2,...\}$,
  - Energieversorgung und Managementmodule $E=\{e_1,e_2,..\}$

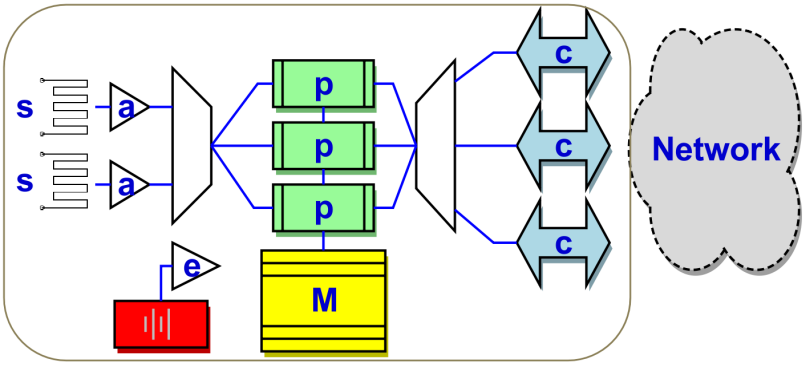


Figure 135. Aufbau und Konnektivität eines Sensorknotens

- Verteilte Sensornetzwerke unterscheiden sich von klasschischer Datenverarbeitung
- DSN können räumlich und zeitlich verteilt sein - ohne zentralen Rechner
- Das Netzwerk selber ist der Rechner - eine virtuelle Maschine

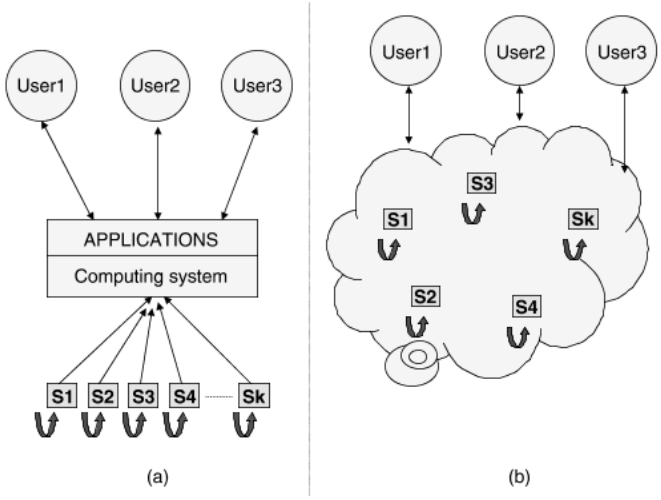


Figure 136. Von zentraler Nutzer-Applikations-orientierter (a) zu dezentraler Sensor-orientierter Datenverarbeitung (b) [G]

Anforderungen an und Fähigkeiten von DSN

- niedrige Kosten
- zuverlässig und robust gegen Störungen (Kommunikation, Energie, Ausfall von Knoten)

- Autonomie eines Sensorknotens
- schnelle Antwortzeit (Latenz)
- Eignung für echtzeitfähige datenstrombasierte Verarbeitung
- lange Betriebszeiträume
- Kombination von verschiedenen senorischen Größen zu neuen Informationen, die durch einen einzlnen Sensor bzw. Sensorknoten nicht verfügbar wären → Daten- und Sensorfusion
- Erhöhter Datendurchsatz durch nebenläufige Datenverarbeitung der einzelnen Sensorknoten
- Redundanz und Fehlertoleranz durch komplimentäre Gruppenbildung und Wettbewerb sowie Kooperation von Sensorknoten
- Zeitliche Synchronität → Synchronisation von Uhren

15.2. Taxonomie von DSN Architekturen

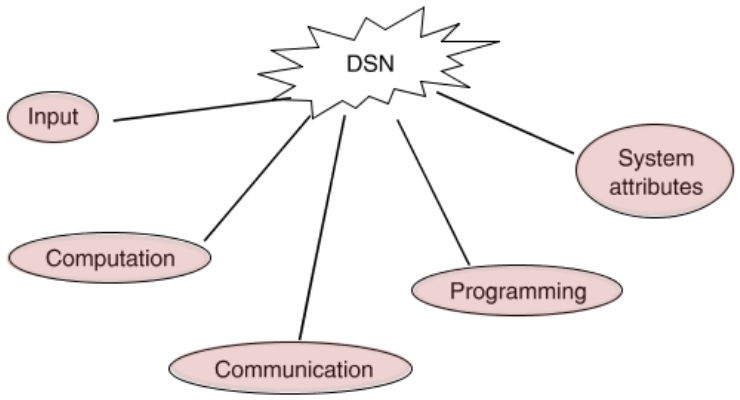


Figure 137. Wichtige Aspekte eines DSN [G]

- Die Hauptaufgabe eines DSN ist die Sammlung und Verarbeitung sensorischer Daten
- Eingabe, Berechnung, Kommunikation, und Programmierung müssen zusammenhängend und nicht unabhängig betrachtet werden à Entwurf eines Systems
- Für jeden dieser Bestandteile gibt es unterschiedliche Teilaspekte und Variationen, die die Struktur und die Performant des gesamten DSN bestimmen.

➤ Es wird zwischen Funktion und Implementierung unterschieden:

Funktion

Die Funktion beschreibt die grundlegenden Operationen und Fähigkeiten des DSN.

Implementierung

Die Implementierung beschreibt die Methoden mit denen die Funktionen erfüllt werden können.

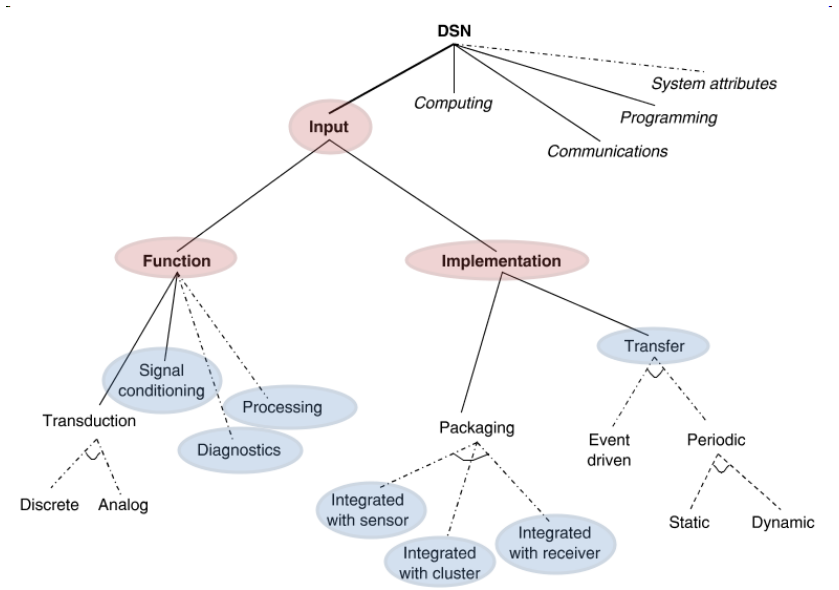


Figure 138. Taxonomie der Eingabe-Aspekete von DSN [G]

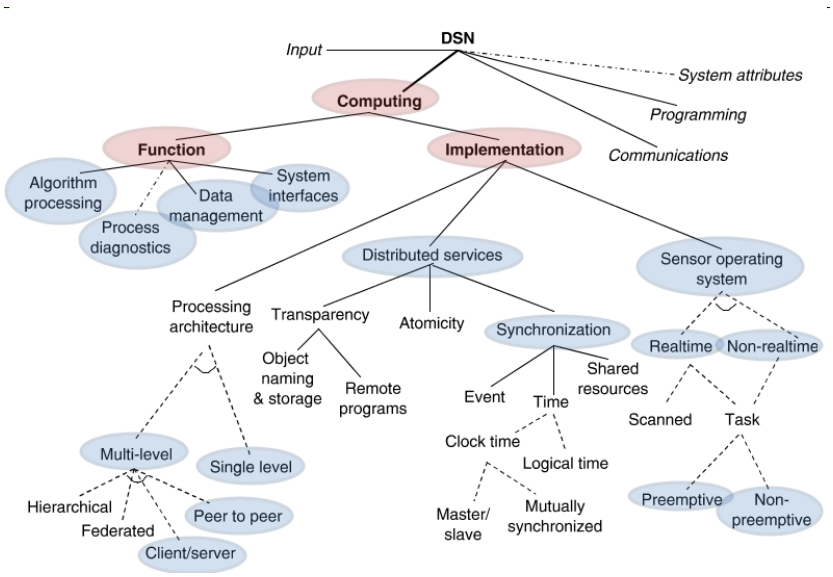


Figure 139. Taxonomie der Berechnung von DSN [G]

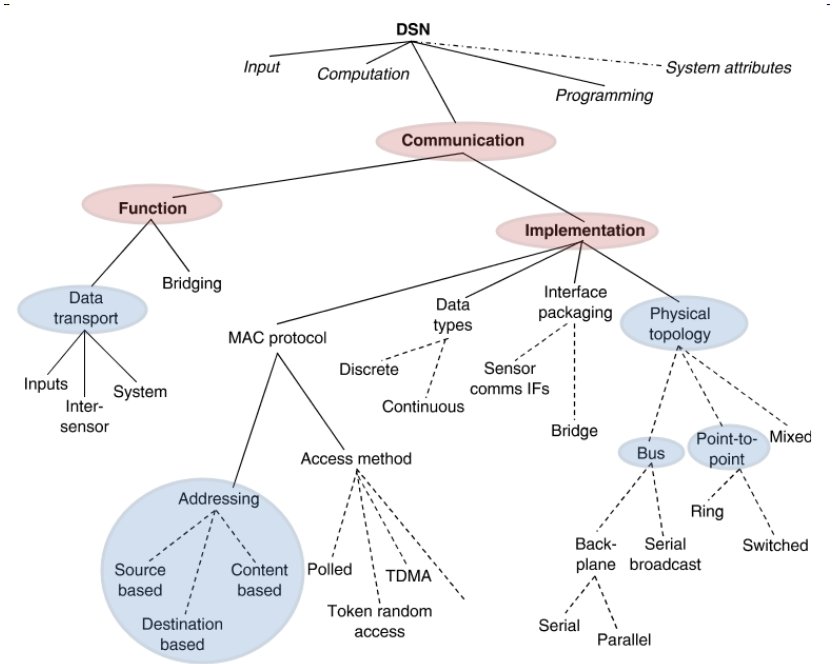


Figure 140. Taxonomie der Kommunikation in DSN [G]

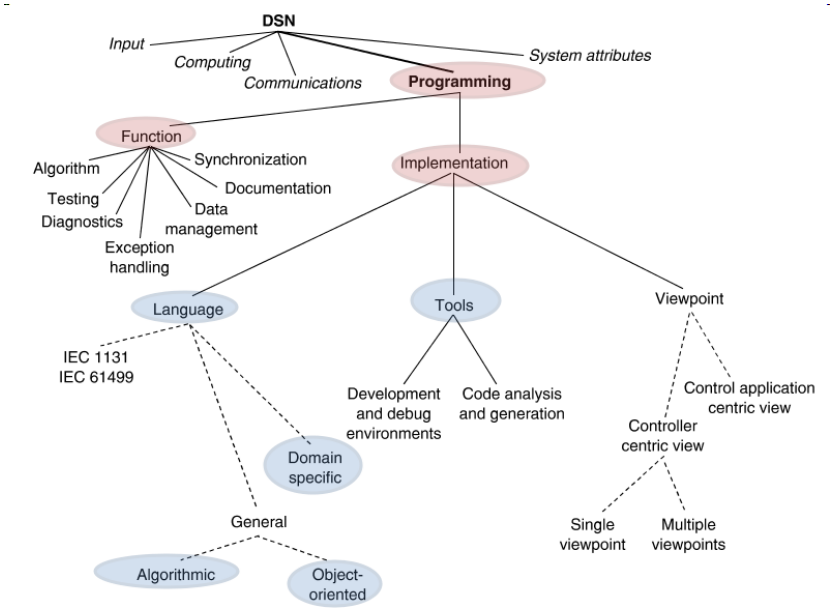


Figure 141. Taxonomie der Programmierung für DSN [G]

15.3. Netzwerkarchitekturen und Topologien

Definition 13. (*Kommunikationsnetzwerk*)
Ein Netzwerk G besteht aus Knoten $N=\{n_1,n_2,...\}$ und Verbindungen (Kanten) $C=\{c_1,c_2,..\}$, die die Knoten untereinander verbinden. Das Netzwerk kann als ein Graph $G(N,C)$ beschrieben werden, der i.A. zyklisch und gerichtet ist.

Parameter und Metriken von Netzwerken

Knotenzahl *N*

Gesamtzahl der Knoten, die ein Netzwerk oder ein Subnetzwerk bilden

Knotengrad (Konnektivität) *K*

Jeder Knoten hat mindestens eine, maximal K Verbindungen zu Nachbarknoten

Skalierbarkeit

Lässt sich das Netzwerk für belibiege Anzahl N von Knoten erweitern?

Routing

Strategie für die Weiterleitung und Zustellung von Daten von einem Senderknotenns n_s zu einem oder mehreren Empfängerknoten $\{n_{e1},n_{e2},\ldots\}$

- ➤ Unicast: nur ein Empfänger
- ➤ Multicast: eine Gruppe von ausgezeichneten Empfängern
- ➤ Broadcast: alle Knoten (in einem Bereich) sind Empfänger

Ausdehnung D

Maximaler Abstand (Distanz) zwischen zwei Knoten

Kosten O

Anzahl der Kanten NC

Effizienz η

Anzahl der Kanten NC im Verhältnis zur Anzahl der Knoten N

Latenz τ

Verzögerung T bei der Zustellung von Daten durch Vermittlung (normiert und vereinfacht in Anzahl von Zwischenknoten)

Durchsatz B

Datendurchsatz (Bandbreite)

Nachrichtenbasierte Vermittlungsnetzwerke

- ➤ Ein Knoten eines Netzwerkes ist
  1. Quelle von Nachrichten, die Daten kapseln,
  2. Empfänger von Nachrichten, und
  3. Vermittler (Router) von Nachrichten (Zwischenknoten)

Geschaltete Netzwerke

- ➤ Direkte Datenvermittlung, ein Knoten eines Netzwerkes ist
  1. Quelle von Daten, und
  2. Empfänger von Daten

15.4. Dynamische Verbindungsnetzwerke

Kreuzmatrixschalter

➤ Direkte Schaltung von Verbindungen ermöglicht die Übertragung von einem Sender- zu einem Empfängerknoten.

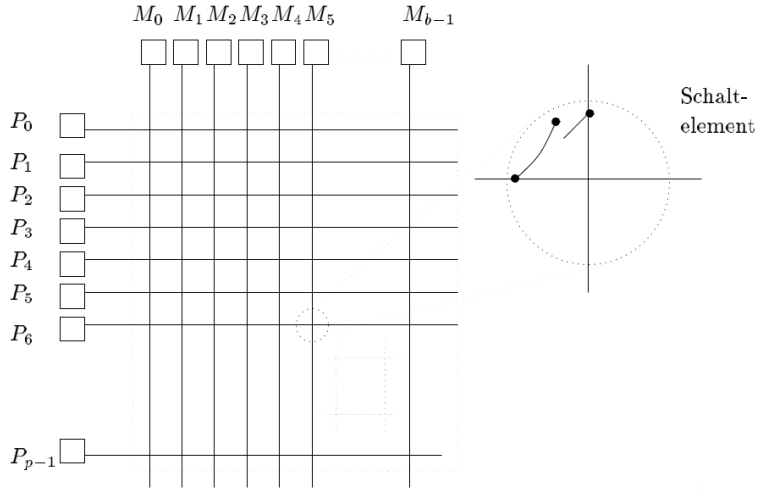


Figure 142. Vollständiger Kreuzschalter (Crossbar Switch) mit Schaltelementen [PA, Vornberger, 1998]

Parameter

➤ Knotenzahl: N

➤ Knotengrad: 1

➤ Skalierbar: Ja

➤ Routing: Nicht erforderlich - konfliktfrei

➤ Ausdehnung: 1

➤ Kosten: N^2

Vorteile

➤ Jeder Knoten kann mit jedem anderen Knoten jederzeit verbunden (geschaltet) werden → Dynamische Konnektivität des Netzwerkes

➤ Ausdehnung optimal klein (und Latenz minimal)

- ➤ Keine Nachrichtenkapselung und kein Kommunikationsprotokoll erforderlich
- ➤ Skalierung bezüglich Leistung ist gut

Nachteile

- ➤ Hohe Kosten und hoher Hardware-Aufwand, der quadratisch mit der Anzahl der Knoten wächst
- ➤ Skalierung bezüglich Kosten ist schlecht
- ➤ Fehlende Synchronisation zwischen Sender und Empfänger

Busbasierte Verbindung

- ➤ Alle Knoten nutzen eine gemeinsame Kommunikationsverbindung
- ➤ Bus-basierte Netzwerke skalieren bezüglich Kosten gut, aber nicht bezüglich der Leistung (Flaschenhals!)

Parameter

- ➤ Knotenzahl: N
- ➤ Knotengrad: 1
- ➤ Skalierbar: Jein
- ➤ Routing: Nicht erforderlich - aber nicht konfliktfrei
- ➤ Ausdehnung: 1
- ➤ Kosten: N (1)

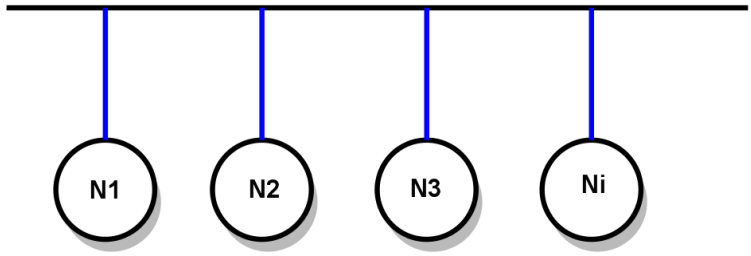


Figure 143. Topologie des busbasierten Netzwerkes

Mehrstufen (Multistage) Verbindungsnetzwerke

- ➤ Mehrstufiger Aufbau des Netzwerkes
- ➤ Kompromiss bezüglich Skalierung zwischen Kreuzschaltern und Bussystemen

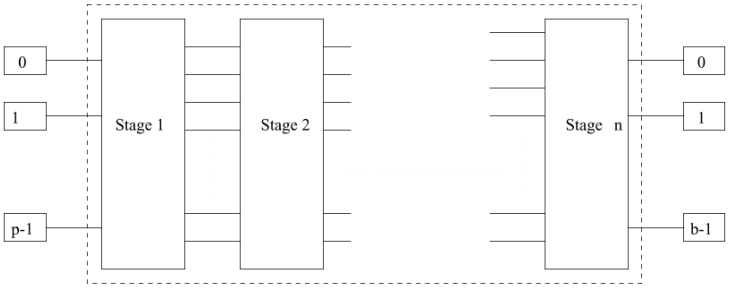


Figure 144. Aufbau eines Mehrstufen Netzwerks (n-stufig) [PA, Vornberger, 1998]

Parameter

- ➤ Knotenzahl: N
- ➤ Knotengrad: 1
- ➤ Skalierbar: Ja
- ➤ Routing: Nicht erforderlich - aber nicht konfliktfrei!
- ➤ Ausdehnung: 1 (oder n - Stufenzahl)

➤ Kosten: $N\log_2 N$

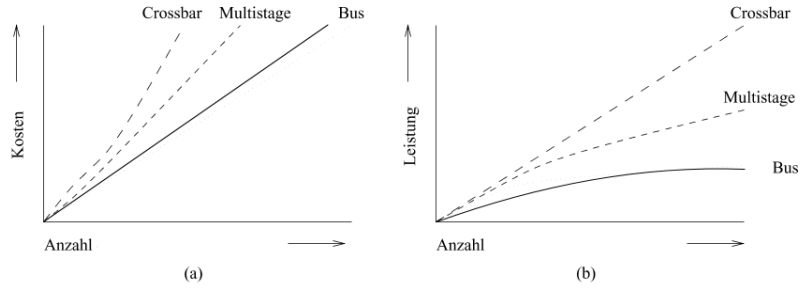


Figure 145. Skalierung von Kosten und Leistung der verschiedenen dyn. Verbindungsnetze [PA, Vornberger, 1998]

Mehrstufiges Permutations-Netzwerk

➤ Jede Stufe besteht aus Binärschaltern, die entweder durchgeschaltet oder gekreuzt geschaltet sind.

➤ Die Ausgänge einer Stufe k werden über ein Permutationsverfahren mit den Eingängen der nächsten Stufe k+1 verbunden.

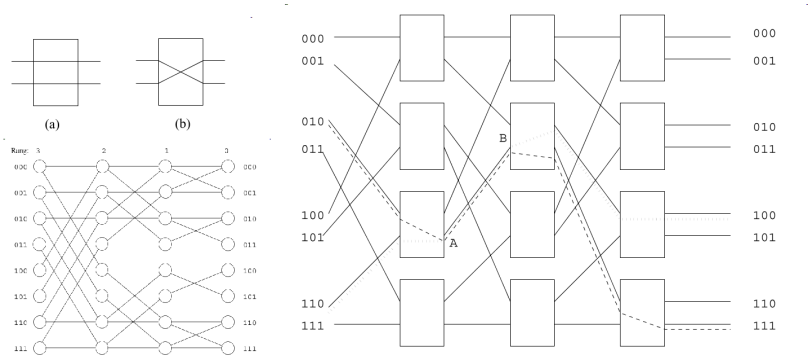


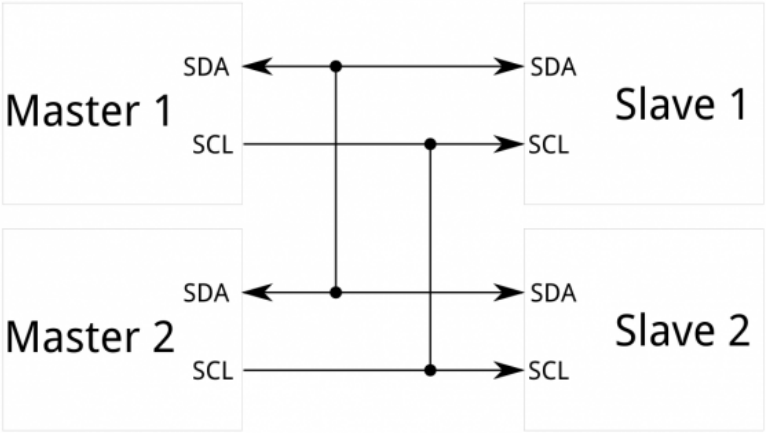
Figure 146. Vollständiges nicht konfliktfreies Mehrstufennetzwerk [PA, Vornberger, 1998]

I2C

I2C: Inter-Integrated Circuit

➤ Für I2C sind nur zwei Drähte erforderlich, z. B. asynchrone serielle Kabel.

- Diese beiden Drähte können jedoch bis zu 1008 Slave-Geräte unterstützen.
- Im Gegensatz zu SPI kann I2C ein Multi-Master-System unterstützen, sodass mehr als ein Master mit allen Geräten am Bus kommunizieren kann (obwohl die Master-Geräte nicht über den Bus miteinander kommunizieren können und sich über die Busleitungen abwechseln müssen).
- Die Datenraten liegen zwischen asynchroner serieller und SPI. Die meisten I2C-Geräte können mit 100 kHz oder 400 kHz kommunizieren.
- Mit I2C ist ein gewisser Overhead verbunden. Für jeweils 8 zu sendende Datenbits muss ein zusätzliches Bit Metadaten (das "ACK / NACK" -Bit) übertragen werden.
- Die zur Implementierung von I2C erforderliche Hardware ist komplexer als SPI, jedoch weniger als asynchrone serielle Schnittstellen (UART). I2C kann einfach in Software implementiert werden.



[learn.sparkfun.com]

Figure 147. Generelle Netzwerkstruktur

- Das Inter-Integrated Circuit (I2C) -Protokoll ist ein Protokoll, mit dem mehrere integrierte digitale Slave-Schaltkreise ("Chips") mit einem oder mehreren "Master"-Chips kommunizieren können.
- Es ist nur für die Kommunikation über kurze Entfernungen innerhalb eines einzelnen Geräts vorgesehen.
- Wie bei asynchronen seriellen Schnittstellen (wie RS-232 oder UARTs) sind für den Informationsaustausch nur zwei Signalleitungen erforderlich.

Protokoll

- ➤ Die Kommunikation über I2C ist komplexer als bei einer UART- oder SPI-Lösung. Die Signalisierung muss einem bestimmten Protokoll entsprechen, damit die Geräte am Bus diesen als gültige I2C-Kommunikation erkennen können.

- ➤ Nachrichten werden in zwei Arten von Frames unterteilt:

  - ❏ Ein Adress-Frame, in dem der Master den Slave angibt, an den die Nachricht gesendet wird, und
  - ❏ Einen oder mehrere Daten-Frames, bei denen es sich um 8-Bit-Datennachrichten handelt, die von Master zu Slave oder umgekehrt weitergeleitet werden.

- ➤ Die Daten werden auf die SDA-Leitung gelegt, nachdem die SCL-Leitung 0-Pegel hat, und werden abgetastet, nachdem der SCL-Leitung von 0- nach 1-Pegel übergeht.

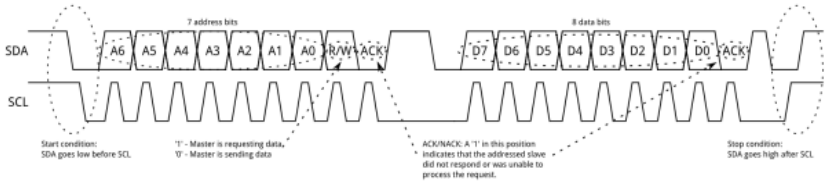


Figure 148. Generelles I2C Kommunikationsprotokoll

15.5. Statische Verbindungsnetzwerke

- ➤ Nachrichtenbasierte Kommunikation, d.h. die zu übertragenen Daten werden in Nachrichten gekapselt
- ➤ In den meisten Topologien ist die Vermittlung von Nachrichten durch andere Knoten entlang eines Pfades ($S \rightarrow n_{i1} \rightarrow n_{i2} \rightarrow .. \rightarrow\rightarrow E$) erforderlich

Sternnetzwerk

- ➤ Master-Slave Netzwerk Hierarchie
- ➤ Ein ausgezeichneter Knoten ist Master, der mit jedem anderen Knoten (Slave) verbunden ist
- ➤ Kommunikation findet immer über den Master (=Router) statt

Parameter

- Knotenzahl: N
- Knotengrad: 1 (Slave), N-1 (Master)
- Skalierbar: Ja
- Routing: erforderlich - wähle Ziel in zwei Schritten über Master
- Ausdehnung: 2
- Geschlossen: nein
- Kosten: N-1

Vollständig verbundenes Maschennetzwerk

- Es gibt keinen ausgezeichneten Master, alle Knoten sind mit jedem anderen verbunden

Parameter

- Knotenzahl: N
- Knotengrad: N-1
- Skalierbar: Ja
- Routing: (nicht) erforderlich - wähle Ziel in einem Schritt
- Ausdehnung: 1
- Geschlossen: ja
- Kosten: $N(N\text{-}1)$ ≈ N^2

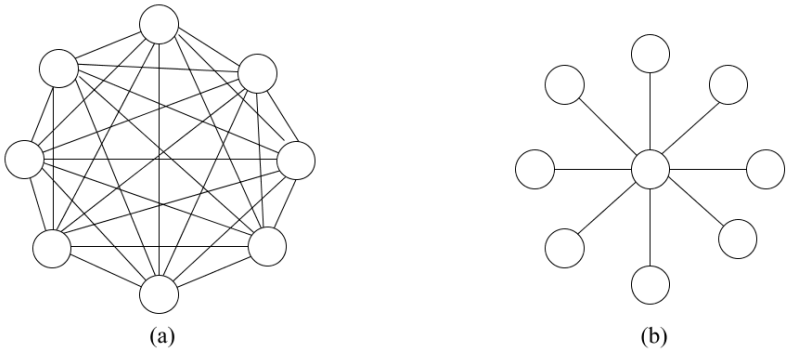


Figure 149. Topologie Vollständig verbundenes Maschen- und Sternnetzwerk [PA, Vornberger, 1998]

Binärer Baum

Parameter

- Knotenzahl: N
- Knotengrad: 3
- Skalierbar: Ja
- Routing: erforderlich - laufe vom Start aufwärts zum gemeinsamen Vorfahren und dann abwärts zum Ziel
- Ausdehnung: $2k$ (k - Höhe des Baums)
- Geschlossen: nein
- Kosten: $N\log_2 N$

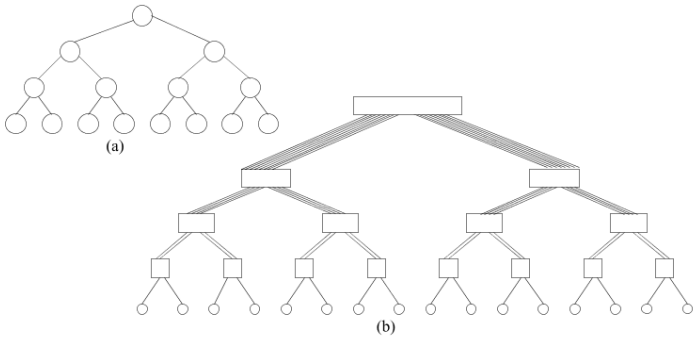


Figure 150. Topologie Binärbaum Netzwerk [PA, Vornberger, 1998]

Lineares Array - Kette - und Ring

Parameter

- Knotenzahl: N
- Knotengrad: 2
- Skalierbar: Ja
- Routing: erforderlich - wähle Richtung und laufe in diese Richtung
- Ausdehnung: N-1 (Kette), $N/2$ (Ring)
- Geschlossen: nein (Kette), ja (Ring)
- Kosten: N-1 (Kette), N (Ring)

Figure 151. Kette und Ring (technologisch schwierig)

2D Gitter

- ökonomisch und techn. geeignet für material-integrierte verdrahtete Sensornetzwerke

Parameter

- Knotenzahl: N
- Knotengrad: 4
- Skalierbar: Ja
- Routing: erforderlich (Δ-Distanz Routing)
- Ausdehnung: 2($\sqrt{N}$-1)(ohne wraparound), 1+($\sqrt{N}$-1) (mit wraparound)
- Geschlossen: nein (ohne wraparound), ja (mit wraparound)
- Kosten: 2(N-$\sqrt{N}$) (ohne wraparound)
- Routing: Wandere horizontal (erste Dimension) bis zur Zielspalte, dann wandere vertikal bis zur Zielzeile (zweite Dimension) $\rightarrow$ Δ-Distanz Routing

Algorithm 2.

TYPE $DIM = \{1, 2, ..., m\}$ $\Delta_i : i-th\ component\ of\ \Delta$, $\tilde{0} = (0, 0, ..)$
TYPE $DIR = \{NORTH, SOUTH, EAST, WEST, UP, DOWN, ...\}$
Numerical mapping of directions
DEF dir = **function**(i) $\rightarrow$
  match i **with**
  $... -2 \rightarrow NORTH \mid -1 \rightarrow WEST \mid 1 \rightarrow EAST \mid 2 \rightarrow SOUTH$ ...
DEF $route_{xy}$ = **function** $(\Delta, \Delta^0, M) \rightarrow$
  if $\Delta \neq \tilde{0}$ **then**
    for *first* $i \in DIM$ **with** $\Delta_i \neq 0$ **do**
      if $\Delta_i > 0$ **then**
        $moveto(dir(i))$; $route_{xy}(\Delta with\ \Delta_i - 1, \Delta^0, M)$
      else if $\Delta_i < 0$ **then**
        $moveto(dir(-i))$; $route_{xy}(\Delta with\ \Delta_i + 1, \Delta^0, M)$
  else $deliver(M)$

3D Gitter

- ökonomisch und techn. geeignet für material-integrierte verdrahtete Sensornetzwerke

Parameter

- Knotenzahl: N

- Knotengrad: 6
- Skalierbar: Ja
- Routing: erforderlich (Δ-Distanz Routing)
- Ausdehnung: 3*(√3N-1)
- Kosten: 3(N-√3N)

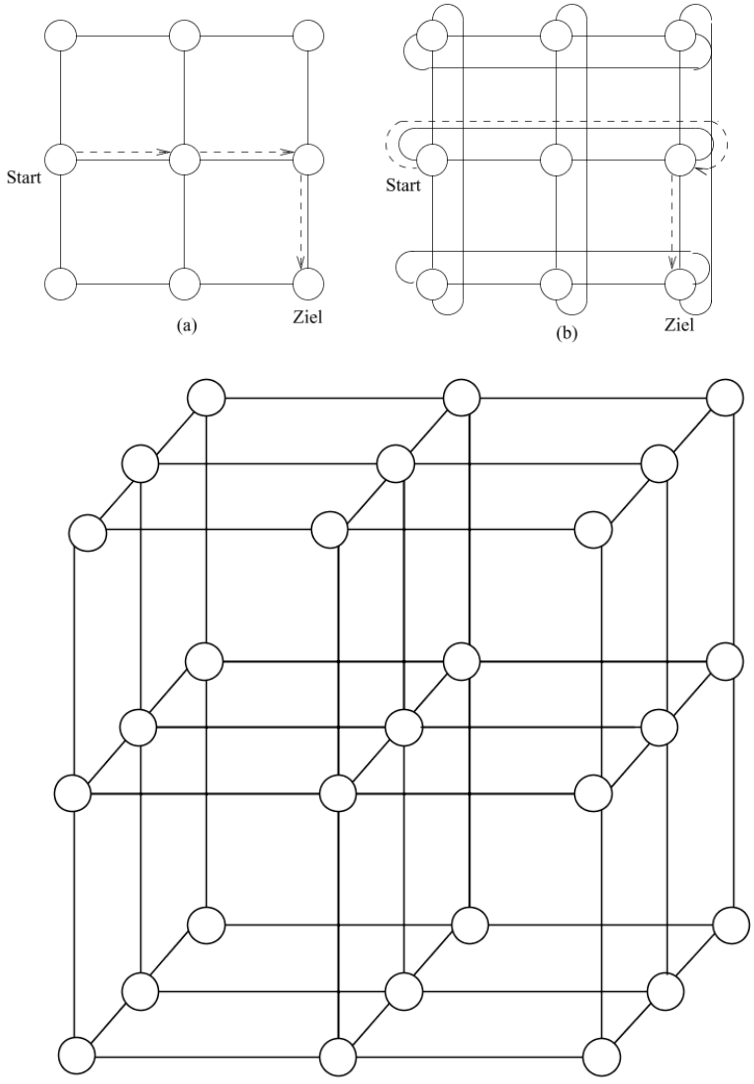


Figure 152. 2D und 3D Gitter und Torus (technologisch schwierig)

15.6. Ad-hoc Netzwerke:

- ➤ Ad-hoc Netzwerke deren Struktur und Verbindungen sich erst zur Laufzeit bilden.
- ➤ Dabei kann sich die Struktur (Netzwerktopologie) dynamisch ändern.
- ➤ Es können Knoten und Verbindungen zwischen Knoten hinzukommen und verschwinden.
- ➤ Selbstorganisation und Selbstkonfiguration können wesentliche Prinzipien wärend der Bildung und danach sein.

Sensorknoten (aka motes)

- ➤ Übliches Designziel: klein und gunstig
- ➤ Mikroprozessor
  - ❑ wenige MHz bis einige hundert MHz
  - ❑ energiesparende Zustande
- ➤ Speicher
  - ❑ haufig nur wenige KB Arbeitsspeicher
  - ❑ teils zusatzlich Flashspeicher fur Code und Daten
- ➤ Sensorik
  - ❑ Temperatur, Licht, Druck, Beschleunigung, Ultraschall uvm.
- ➤ Drahtlose Kommunikation
  - ❑ Reichweiten von wenigen Metern bis einige hundert Meter
  - ❑ Erste Standards, z. B. IEEE 802.15.4 (WPAN) und ZigBee
- ➤ Energieversorgung
  - ❑ heute meist Batterien
  - ❑ in Zukunft: Solarzellen und anderes Energy Harvesting

Ausbringung

- ➤ Keine zentrale Infrastruktur

- ❏ Initiale Position haufig unbekannt
- ❏ Multihop-Kommunikation

➤ Aufgaben

- ❏ Erkennen / Melden von Ereignissen
- ❏ Beantworten von Anfragen

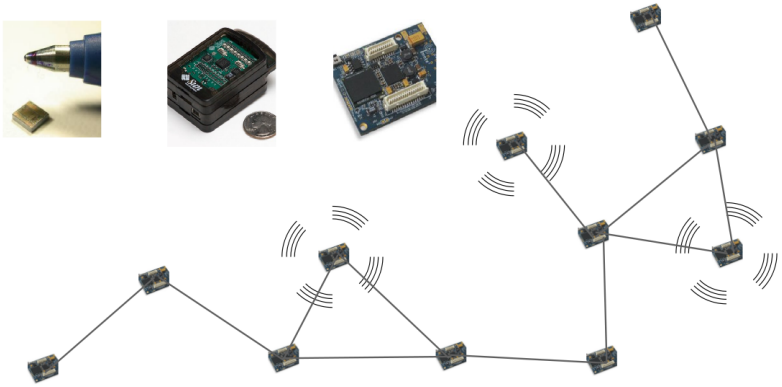


Figure 153. (Links) Technologien von Sensorknoten für Ad-hoc Netzwerk (Rechts) Dynamische Netzwerkstrukturen [KIT]

Herausforderungen

➤ Geringes Energiebudget

- ❏ geringe Leistung (jahre-)lange Bereitschaft
- ❏ Lastbalancierung, Aufteilung von Aufgaben

➤ Geringer Speicher, geringe Rechenleistung

- ❏ verteilte Losungen komplexe Selbstorganisation

➤ Viele Sensorknoten

- ❏ Algorithmen müssen gut mit Größe des Netzes skalieren

➤ Geometrie

- ❏ enger Zusammenhang zwischen Positionen und Daten/Aufgaben

- ❏ gemeinsame Nutzung und Störung der Funkkanäle¨

➤ Redundanz, Unzuverlässigkeit und Dynamik

- ❏ Knoten können ausfallen, sich bewegen
- ❏ viele Knoten können Aufgaben redundant ausführen

Anwendungen

➤ Umweltüberwachung

- ❏ Waldbrände, Gletscherbewegungen
- ❏ Bewässerung von Agrarflachen
- ❏ Schadstoffe (chemisch, radioaktiv)

➤ Beobachtung von Tieren

- ❏ Überwachung von Brutstätten, Bewegungen
- ❏ Intelligente Zäune

➤ Internet of Things

- ❏ Smart Cities, Smart Homes
- ❏ Vernetzung alltäglicher Produkte

➤ Medizinische Überwachung

- ❏ Patienten im Krankenhaus

➤ Sicherheit

- ❏ Grenzüberwachung
- ❏ Überwachung von Gebäuden

➤ Verkehrsoptimierung

- ❏ Stauwarnungen

➤ Materialintegrierte Sensorsysteme

- ❏ Strukturüberwachung

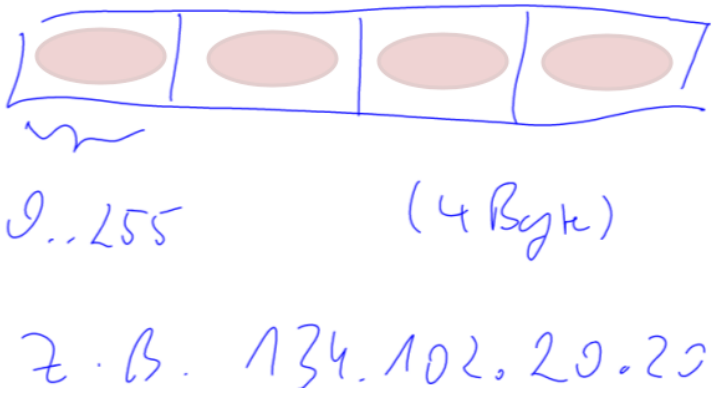
- ❏ Smart Things
- ❏ Artificial Skin
- ❏ E-Textiles

15.7. IP Kommunikation

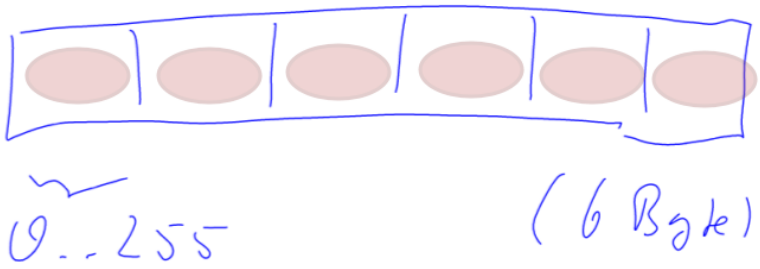
➤ Internet Kommunikation mit dem Internet Protocol (IP)

- ❏ Netzwerkknoten benötigen eindeutige IP Adresse (nicht für MISS geeignet!)

IP-4



IP-6



➤ Das Internet ist ein Netz aus Netzen (hierarchisch und verteilt)

- ❏ Netzwerkadressen werden in räumlichen Domains zusammengefasst (IP4)
- ❏ Mobile Geräte benötigen daher lokale IP Adressen (u.U. wechselnd)
- ❏ IP6 ordnet Geräten eindeutige Adressen zu (mobil)

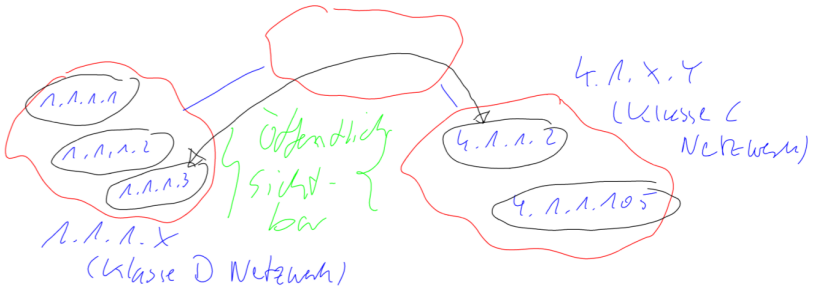


Figure 154. Domainen im Internet und Netze aus Netzen

- ➤ Private Netzwerke und die immer größer werdende Anzahl von Geräten erfordern bei IP4 NAT (nicht bei IP6)
- ➤ NAT: Netwrok Address Translation

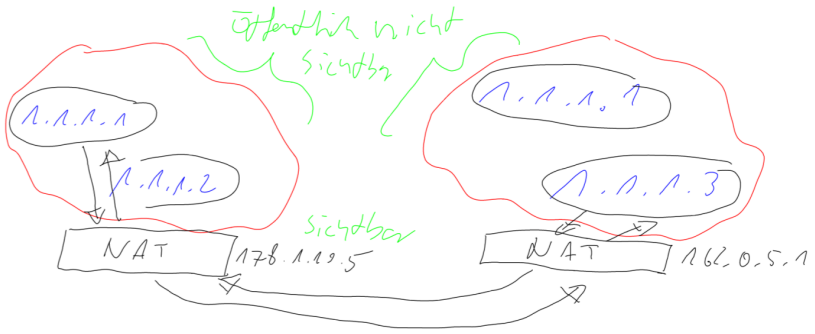


Figure 155. Kommunikation zwichen privaten IP Netzwerken über NAT

15.8. Remote Procedure Call

- ➤ Der RPC dient der Klienten-Server Kommunikation, wobei anders als bei HTTP Servern die Rolle von Klient und Server beliebig getauscht werden kann
- ➤ Es gibt drei Operationen:

Get Request (port)

Der Serverprozess gibt die Verarbeitung von RPC Anfragen auf einem bestimmten Port bekannt.

PutReply (client)

Der Serverprozess sendet eine Antwort auf eine RPC Anfrage

Transaction (port,data)

Der Klientenprozess sendet eine RPC Anfrage an einen Server (port) und wartet auf die Antwort

- ➤ RPC ist Interprozesskommunikation!
- ➤ Neben Datenübertragung dient RPC zur Ausführung von Prozeduren (Funktionen) mit Daten!

LUAOS

```
rpc = rpc:new()
```

rpc:getreq

`function (host:string,port:number,handler: function (data:table)` → `table|nil)`
Ein Server wird gestartet. Die benutzerdefinierte Handlerfunktion wird bei jedem Request aufgerufen und muss einen Datenrekord zurückgegben (oder nil)

rpc:putrep

`function (client:userdata, data:table)`
Der Server sendet die Rückantwort (implizit in *rpc:getreq* enthalten)

rpc:trans

`function (host:string,port:number,data:table)` → `error:nil|string,table|nil`
Ein Servertransktion durch einen Klienten.

Example 7. (*LUAOS Serverprozess*)

```
local rpc = rpc:new()
rpc:getreq('127.0.0.1',12345, function (request)
  local reply = {}
  print(request)
  case request.cmd of
    "square" => y=request.x*request.x;  reply = { status = "OK", y = y } end
    "sqrt"  => y=math.sqrt(request.x); reply = { status = "OK", y = y } end
    else reply = {status = "EINVALIDMCD" } end
  end
  return reply
end)
```

Example 8. (*LUAOS Klientenanfrage*)

```
local rpc = rpc:new()
local err,reply = rpc:trans('127.0.0.1',12345,
  {
    cmd='square',
    x=math.random()
  })
if err then print(err) else print(reply) end
```

16. Energie und Energiemanagement

16.1. Energiequellen

Physikalischen Größe und Materialien/Komponenten:

- Wärme: Thermoelektrika
- Schwingung: u. a. Piezoelektrika
- Bewegung: u. a. Piezoelektrika, Generatoren
- Elektromagnetische Wellen: Antennen
- Licht: Solarzellen

Natürliche und technische Quellen:

- Abwärme von Maschinen versa Körperwärme des Menschen

- ➤ Schwingung/Bewegung von Maschinenelementen versa Bewegung des Menschen/von Tieren
- ➤ Beleuchtung in Innenräumen versa Sonnenlicht etc.

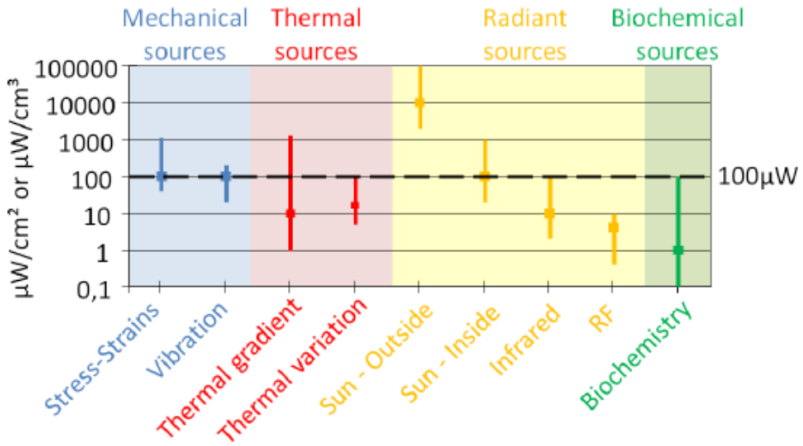


Figure 156. Energieernte und zu erwartende Ausbeute (Leistungsdichte) bei verschiedenen Energiequellen [S. Boisseau, 2012]

Angebot

Source	Source power	Harvested power
Ambient light		
Indoor	0.1 mW/cm^2	10 µW/cm^2
Outdoor	100 mW/cm^2	10 mW/cm^2
Vibration/motion		
Human	0.5 m @ 1 Hz 1 m/s^2 @ 50 Hz	4 µW/cm^2
Industrial	1 m @ 5 Hz 10 m/s^2 @ 1 kHz	100 µW/cm^2
Thermal energy		
Human	20 mW/cm^2	30 µW/cm^2
Industrial	100 mW/cm^2	1–10 mW/cm^2
RF		
Cell phone	0.3 µW/cm^2	0.1 µW/cm^2

[R. J. M. Vullers, 2009]

Nachfrage

	Device type	
	Power consumption	Energy autonomy
Smartphone	1 W	5 h
MP3 player	50 mW	15 h
Hearing aid	1 mW	5 days
Wireless sensor node	100 µW	Lifetime
Cardiac pacemaker	50 µW	7 years
Quartz watch	5 µW	5 years

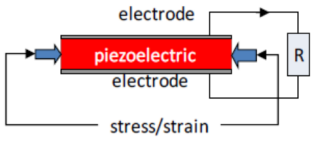
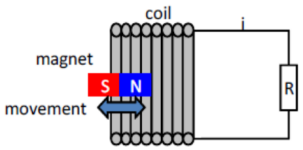
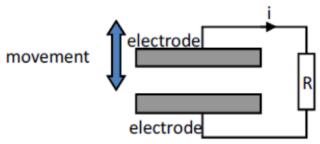
[C. O. Mathuna, 2008]

Grundprinzip: Methoden der Umsetzung

Mechanisches System

System aus Feder, Masse und Dämpfer.

1. Piezoelektrische Konverter
2. Elektromagnetische Konverter
3. Elektrostatische Konverter

Piezoelectric converters	Electromagnetic converters	Electrostatic converters
Use of piezoelectric materials	Use of Lenz's law	Use of a variable capacitor structure
		

[S.Boisseau, 2012]

Thermische Systeme

Ausnutzung des Seebeck Effekts, d.h. Spannungsabfall in einem Stromkreis aus zwei unterschiedlichen elektrischen Leitern bei Vorliegen einer Temperaturdifferenz zwischen den Kontaktstellen

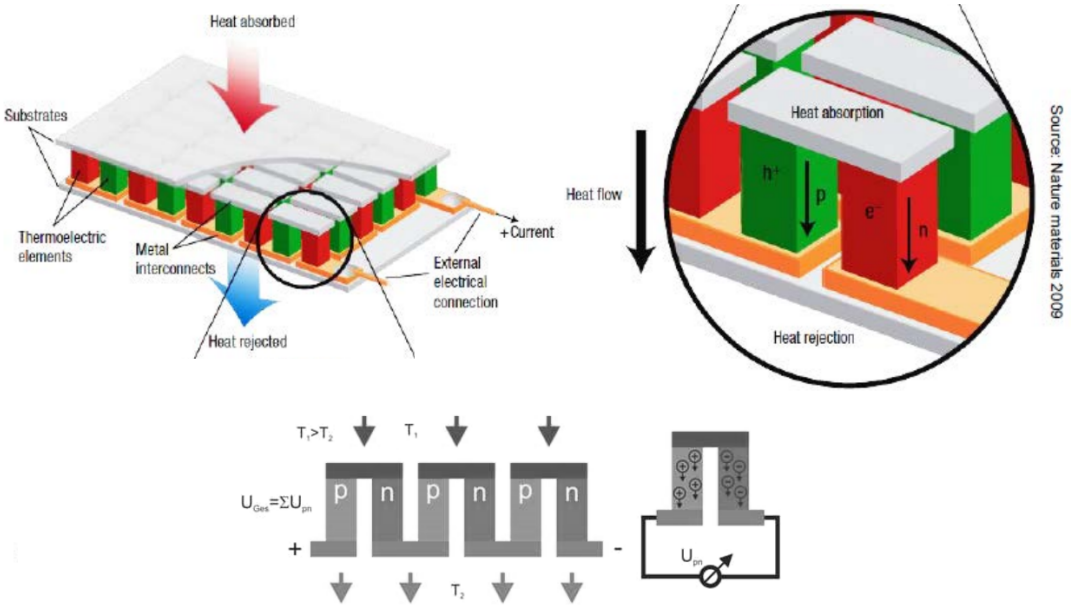


Figure 157. Typischer Aufbau eines Thermogenerators aus Einzelelementen (n- und p-Halbleiter) für Temperaturdifferenz senkrecht zur Fläche des Thermogenerators

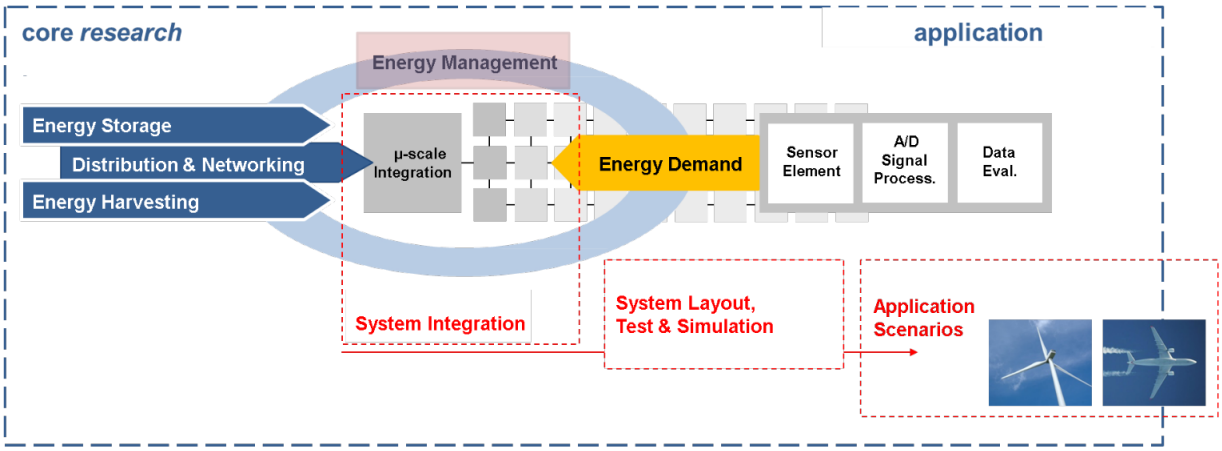
Optische Systeme

Ausnutzung des fotoelektrischen Effekts (Solarzellen, Fotodioden)

16.2. Energiemanagement

- ➤ Energieernte muss eingebettet in das Gesamtsystem betrachtet werden.
- ➤ Energiemanagement findet auf verschiedenen Ebenen statt:
  - ❑ Systemebene
  - ❑ Netzwerkebene
  - ❑ Sensorknotenebene
  - ❑ Gerätebene (Mikroprozessor, IO Prozessoren, ..)

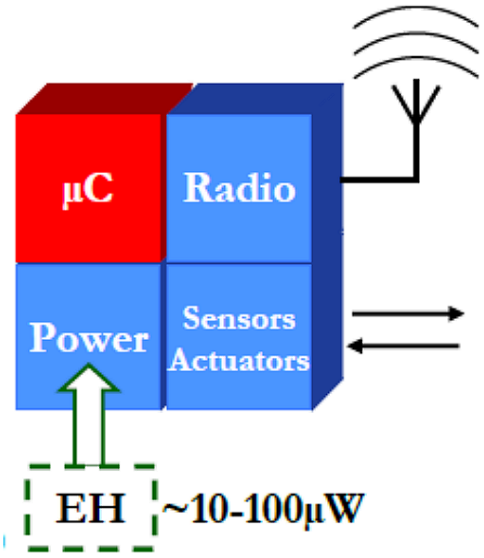
❑ Sensorebene



Verbraucher

Energieverbraucher in einem materialintegrierten Sensornetzwerk:

- ➤ Signalverarbeitung analog
- ➤ Signalverarbeitung digital
- ➤ Datenverarbeitung
- ➤ Kommunikation innerhalb des Netzwerks und nach außen
- ➤ Verluste/begrenzte Wirkungsgrade: Leitung, Speicherung etc.



[S.Boisseau, 2012]

Energiemodell

➤ Zeitliche Komponente: Sowohl Energieverbrauch als auch Energiegewinnung sind Funktionen der Zeit

Energiemodell: Jeder Sensorknoten verfügt über einen Energiespeicher, der

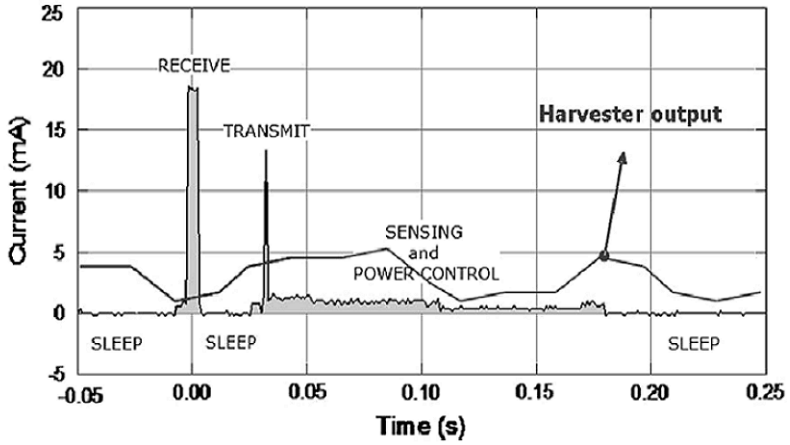
I. Über eine zeitlich fluktuierende Energiequelle (Energy Harvester) mit Energie gespeist wird, und der

II. Durch Berechnungs- und Kommunikationaktivität (Digitale Signalverarbeitung) des Sensorknotens entladen wird,

III. Und analoge Signalverarbeitung führt zu einer Entladung des Speichers.

$$E(t) = E_0 + \sum_{\tau=0}^{t} e_{harvest}(\tau) - \sum_{\tau=0}^{t} e_{computing}(\tau) - \sum_{\tau=0}^{t} e_{analog}(\tau)$$

$$e_{computing}(t) = \sum_{i=0}^{n} e_{instruction}(i)$$



[R. J. M. Vullers, 2009]

Figure 158. Zeitliche Abweichungen zwischen Verfügbarkeit von/Bedarf an Energie: Speicher erforderlich.

- ➤ Maßnahmen zur Minimierung des Energieverbrauchs in Sensornetzwerken:
  - ❑ Erfassung und Verarbeitung von Messwerten nicht kontinuierlich, sondern in bestimmten Zeitintervallen.
  - ❑ Verkürzung der Zeitintervalle bei kritischen Betriebszuständen.
  - ❑ Normalbetrieb mit reduzierter Sensoranzahl, aktivieren weitere Sensoren bei kritischen Betriebszuständen.
  - ❑ Datenverarbeitung in Abhängigkeit vom Betriebszustand - Normalbetrieb mit energetisch vorteilhaften Algorithmen auf Kosten der Genauigkeit, bei kritischen Betriebszsutänden Wechsel zu energetisch aufwändigeren, aber numerisch genaueren Algorithmen.

Algorithmische Selektion

Beispiel für die Auswirkungen einer an die Energiesituation angepassten Wahl des Algorithmus zur lokalen Datenverarbeitung in den Knoten eines Sensornetzwerkes ist ein PID-Regler.

- ➤ Leistungsbedarf von Digitalelektronik hängt von der mittleren Schaltaktivität von Logikgattern pro Zeiteinheit ab
  - ❑ Parasitäre Kapazitäten und Verlustwiderstande sind Ursachen

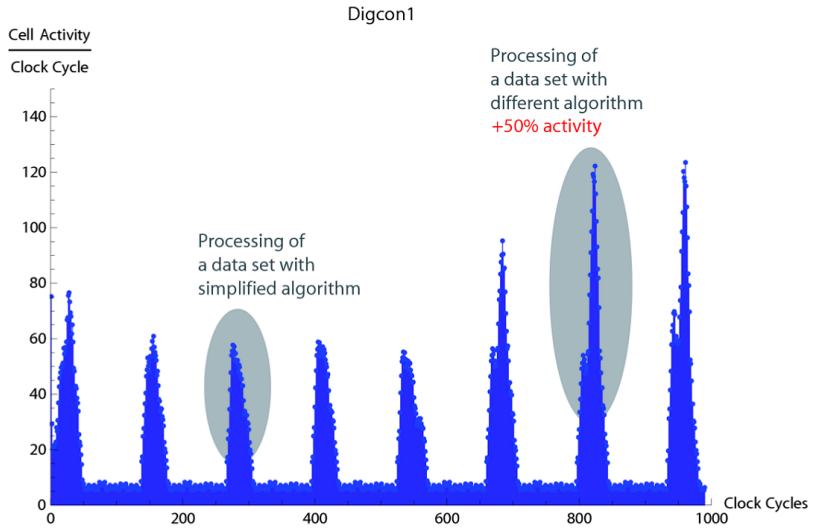


Figure 159. Schaltaktivität eine Digitallogikschlatkreises (PID Regler) in Abhängigkeit von der Datenverarbeitung (zeitlich) und der Auswahl der Algorithmen (Links) P-Regler (Rechts) PID-Regler

16.3. Energiespeicher

Energie wird i.A. in Form von elektrischer Energie gespeichert.

- ➤ Batterien
- ➤ Kondensatoren (Supercaps)

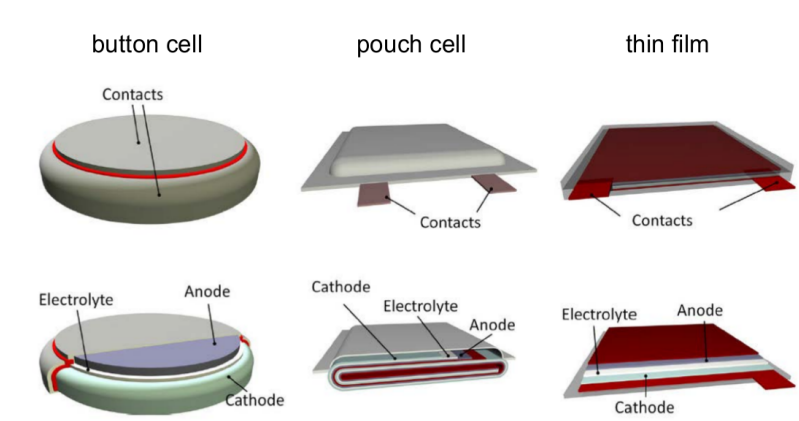


Figure 160. Verschiedene Batterieformen

17. Fertigungstechnik

17.1. Fertigungsverfahren

- Auf der Ebene des Sensorelementes sind Verfahren für
  - Materialaufbringung/-abscheidung,
  - Materialabtrag und
  - Strukturierung erforderlich
- Verfahren sind zunächst unabhängig davon, ob dem Sensorelement ein struktur- oder materialbasierter Effekt zugrunde liegt, bezüglich:
  - Kontaktierung,
  - Zuleitungen,
  - Elektroden,
  - definierte Größen aktiver Komponenten
- Bei Betrachtung integrierter Sensoren kommt der Bereich Aufbau- und Verbindungstechnik/Packaging mit weiteren typischen Fertigungsprozessen hinzu
- System-in-Foil-Technologien für die Materialintegration besonders geeignet

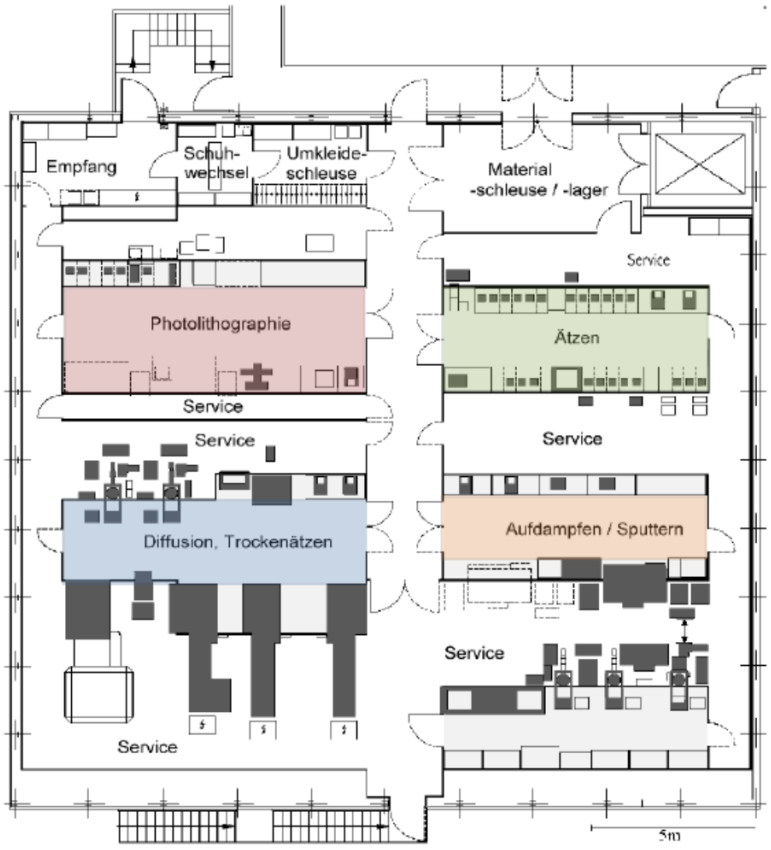
Mikrosystemtechnische Sensoren

Mikrosystemtechnische Sensoren integrieren mikroelektronische und senorische/aktorische Komponenten und nutzen dafür (Fertigungs-)Methoden der Mikroelektronik und Halbleitertechnologie.

- Der klassische Werkstoff (als Substrat und/oder als Funktionsmaterial) der Mikrosystemtechnik ist **Silizium**.
- Andere Materialien können z. B. durch die bereits erwähnten Abscheideverfahren eingebracht werden und das Spektrum der umsetzbaren Funktionalitäten und Eigenschaftsprofile erweitern.
- Aufgrund der geringen Strukturgrößen erfordern Prozesse der Mikrosystemtechnik wie der Mikroelektronik in der Regel Reinräume.

Reinräume

- ➤ Wegen der Feinheit der Strukturen ist Schutz vor Verunreinigungen unerlässlich!
- ➤ Ein ruhig sitzender Mensch erzeugt ca. 300000 Partikel/Minute.
- ➤ Verschiedene Arbeitsschritte in der Mikrosystemfertigung bedürfen verschiedene abgetrennte Bereiche



Silizium

Silizium als zentraler Werkstoff

- ➤ Verfügbarkeit: Silizium gibt es wie Sand am Meer
- ➤ Einkristallin und in höchster Reinheit herstellbar

- Halbleiter, p- und n-Dotierung möglich mit Bor respektive Phosphor oder Arsen, Leitfähigkeitseinstellung
- Oxidation: Ausbildung von Isolationsschichten & Diffusionssperren
- Chemisch inert gegenüber vielen Materialien
- Mit alkalischen Medien anisotrop ätzbar
- Hohe Wärmeleitfähigkeit
- Geringe thermische Ausdehnung

Wafer-basierte Prozesse

- Verarbeitung von Silizium geht einher mit Wafer-basierten Prozessen.
- Dies bedeutet u. a. Batch-Prozesse sowie eine in erster Näherung zweidimensional orientierte Vorgehensweise bei der Herstellung von sensorischen, aktorischen oder elektronischen Komponenten.
- Die Vergrößerung der Wafer-Fläche bietet Kostenvorteile und hängt ab vom Durchmesser der prozessstabil herstellbaren Si-Einkristalle

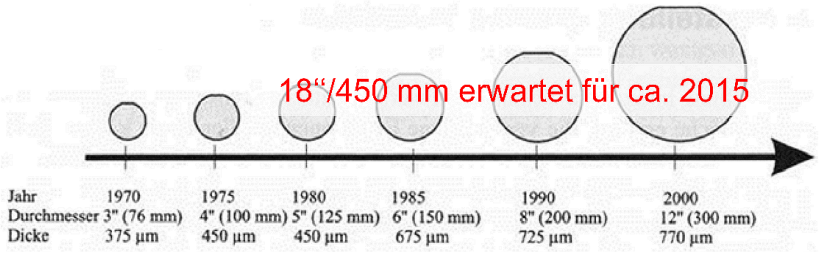


Figure 161. Historische Vergrößerung des Waferdurchemessers [Prof. Vellekoop, IMSAS]

Siliziumprozess

1. Herstellung des reinen Siliziums (Kristallzucht)
2. Photolitographische Strukturierung
3. Ätzen (Abtragen)
4. Aufdampfen und Sputtern (Auftragen)
5. Schneiden
6. (Verpacken)

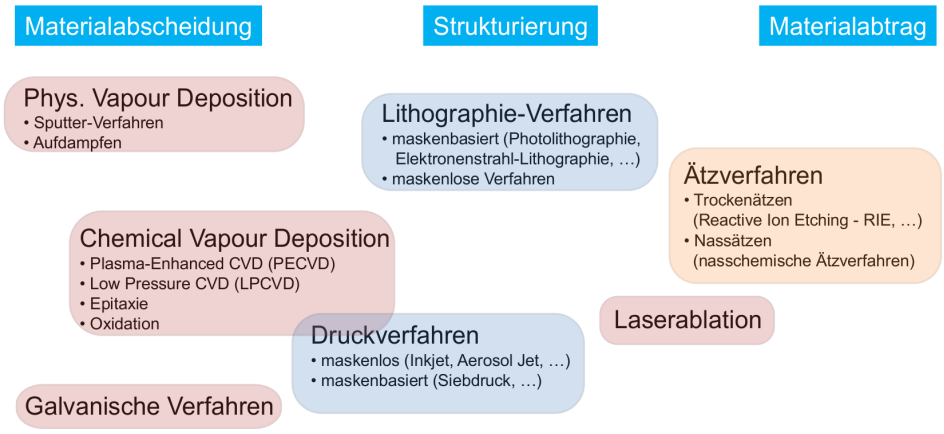


Figure 162. Klassifizierung der Fertigungsverfahren

17.2. Photolithographie

- ➤ Photolithographie ist die zweidimensionale Strukturierung eines Photolacks (Photoresist) über eine Quelle:
  - ❏ gleichmäßige Belichtung durch eine Maske, deren Struktur auf den Photoresist projiziert wird.
- ➤ Je nach Reaktion auf die Belichtung wird zwischen Positiv- und Negativresist differenziert.
  - ❏ Die Bezeichnung Positiv- bzw. Negativresist kann bezogen auf die final erzeugte Struktur verstanden werden.
  - ❏ Bei Positivresist sind die erzeugten Öffnungen im Resist ein (positives) Abbild der Öffnungen der Maske, ebenso die in Folgeschritten erzeugten Strukturen.
- ➤ Belichtetet Bereiche müssen durch chemischen Entwicklungsvorgang abgetragen werden
- ➤ Dann kann der eigentliche Prozessschritt erfolgen (Auftrag/Abtrag)
- ➤ Der Photolack muss nach dem eigentlichen Prozessschritt wieder entfernt werden (Stripping)

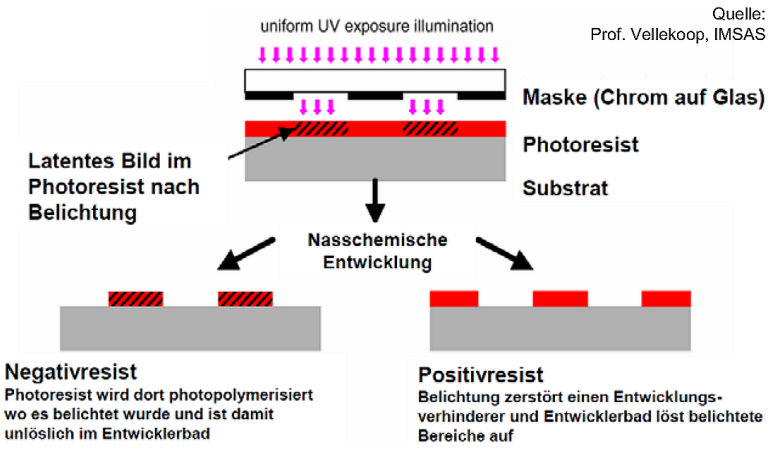


Figure 163. Vergleich (Links) Negativresistverfahren (Rechts) Positivresistverfahren

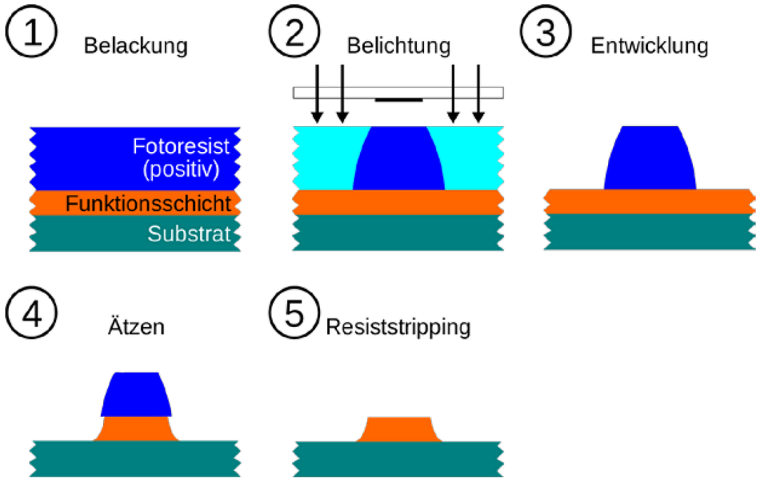
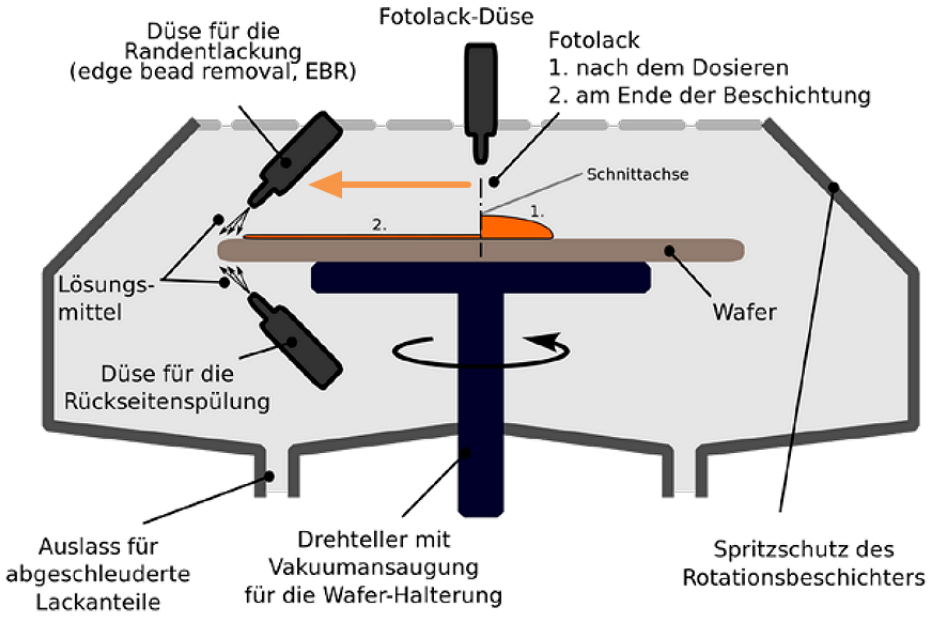


Figure 164. Prozessschritte der Photolithographie [www.wikipedia.org]

Substratbeschichtung

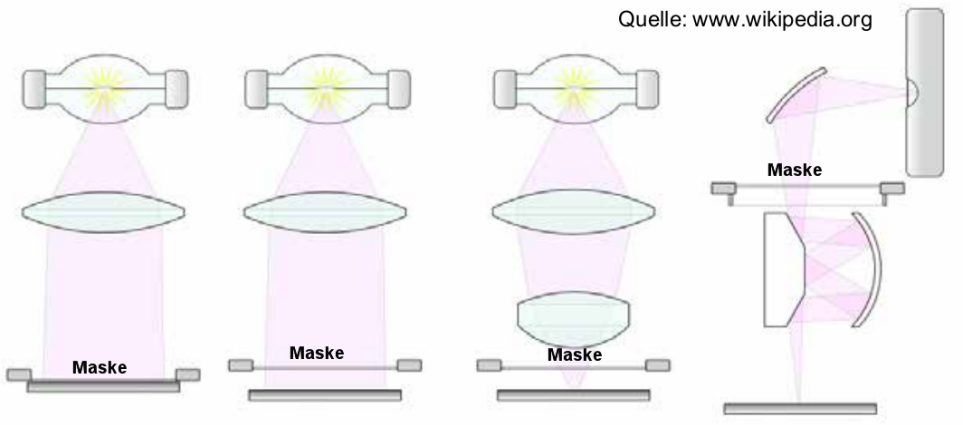
- Das Aufbringen des Photoresist erfolgt in der Regel über Spin Coating.
- Aus einer Düse wird der Photolack auf den rotierenden Wafer aufgebracht und verteilt sich von innen nach außen



[www.wikipedia.org]

Belichtung

- Kontaktbelichtung und Proximitybelichtung (ca. 10-50 μm Abstand Maske-Substrat) bilden Maskenstrukturen in Originalgröße auf das Substrat ab.
- Projektionsbelichtung erlaubt bei Abbildungsmaßstäben von ca. 1:4-5 gröbere, damit kostengünstigere Masken für identische Strukturgrößen.
- Nachteil: Schrittweises Verfahren erforderlich, da das Substrat nicht in einem Schritt komplett belichtet wird.

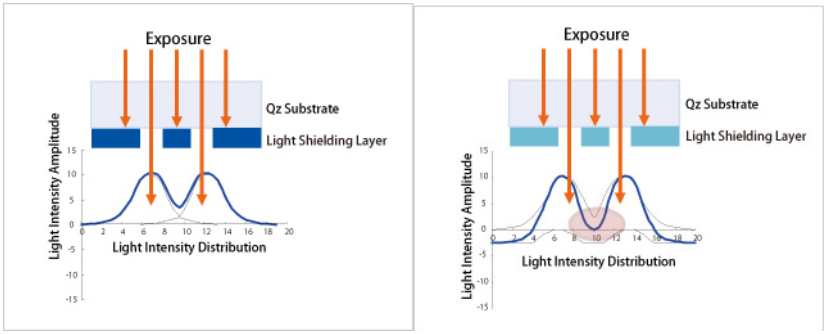


Binärmasken

- Einfachste Variante sind **binäre Masken**, bestehend aus einem Glassubstrat, auf dem eine Chrom- oder eine opake Molybdänsilizid- Schicht aufgebracht ist (Bezeichnungen COG, OMOG).
- Die Cr- bzw. MoSi-Schicht ist undurchlässig für das zur Belichtung verwendete Licht. Nach einer Strukturierung, die die Beschichtung lokal entfernt, kann das Licht durch das freigelegte Glas auf das mit Photolack beschichteten Substrat fallen, verbleibendes Cr bzw. MoSi beschattet das Substrat in den übrigen Bereichen (Binärmaske).
- Die Strukturierung der Cr-/MoSi-Schicht erfolgt in der Regel ebenfalls über Lithographie, meist mittels maskenloser, **direktschreibender** Verfahren wie Laser- /→Elektronenstrahl-Lithographie.

Phasenmasken

- Bei Phasenmasken erfolgt die Abbildung einer Struktur auf dem Substrat nicht über eine Abdeckung/Abschattung, sondern über Interferenz in Folge von Phasenverschiebungen.
- Die Phasenverschiebungen werden eingestellt, indem z. B. das transparente Substrat der Maske am entsprechenden Ort angeätzt wird.
- Die Differenz der im Material zurückzulegenden Strecken bewirkt im Zusammenspiel mit der verringerten Wellenlänge/Phasengeschwindigkeit die genutzt Phasenverschiebung.



[www.toppan.co.jp]

Figure 165. Vergleich (Links) Binärmaske (Rechts) Phasenmaske

Auflösung

➤ Die Wellenlänge des verwendeten Lichts ist ein bestimmender Faktor für die minimal herstellbaren Strukturgrößen (minimum feature size/ critical dimension CD, depth of field DOF, NA: Numerische Apertur, λ: Wellenlänge):

$$CD = k_1 \frac{\lambda}{NA}$$
$$DOF = k_2 \frac{\lambda}{NA^2}$$

➤ Optisch dichteres Medium (z. B. Wasser) erlaubt anderen Strahlengang zwischen letzter Linse und Substrat, möglicher Nachteil: Einfluss auf DOF $\geq$ Einfluss auf CD.

17.3. Elektronenstrahl-Lithographie

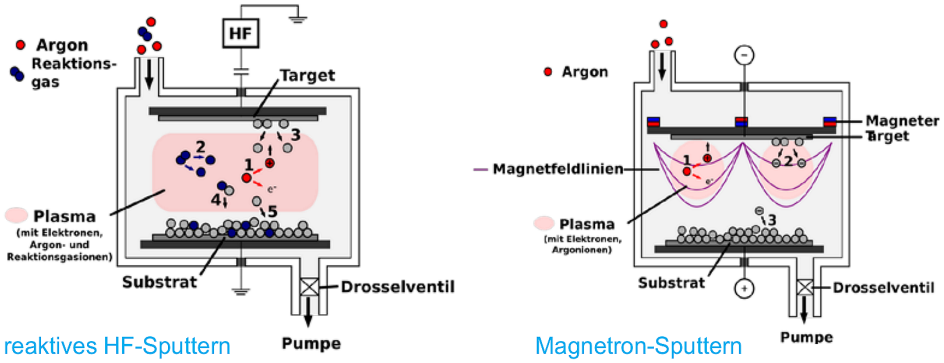
➤ Die Elektronenstrahl-Lithographie verwendet anders als die Photolithographie kein Licht, sondern einen Elektronenstrahl zur Belichtung einer dem Photoresist entsprechenden Beschichtung.

➤ Es existieren maskenbasierte Verfahren analog der Photolithographie wie auch maskenlose direct write-Verfahren.

➤ Die minimale erzielbaren Strukturgrößen sind in der Regel kleiner als bei der Photolithographie.

17.4. Sputter-Verfahren

➤ Beim Sputtern werden durch Beschuss mit energiereichen Ionen (z. B. Argon-Ionen) aus einem Festkörper, dem Target, Atome herausgerissen, die dann auf einem Substrat abgeschieden werden können.



[www.widkipedia.org]

17.5. Epitaxie

➤ Epitaxie beschreibt das gerichtete Aufwachsen einer Kristallschicht auf einem geeigneten Substrat des gleichen (Homoepitaxie) oder eines anderen Materials (Heteroepitaxie).

➤ Epitaxie-Verfahren lassen sich einteilen in

- ❏ Flüssigphasenepitaxie
- ❏ chemische Gasphasenepitaxie (CVD-Verfahren)
- ❏ physikalische Gasphasenepitaxie (PVD-Verfahren)

➤ Silizium-Schichten werden in der Regel mit chemischen Gasphasen-Epitaxie-Verfahren hergestellt, Anwendungsbeispiele sind ICs, die nach dem SOI (Silicon-on-Insulator)-Prinzip aufgebaut sind.

➤ In der erzeugten Si-Schicht werden die Funktionselemente der ICs durch **Dotierung** realisiert.

➤ **Dotierung** bedeutet das Einbringen von Fremdatomen in ein Substrat zur Änderung der elektrischen Eigenschaften

Silizium

- ➤ Erwärmung des Substrats auf ca. 600-1200°C im Vakuum
- ➤ Einbringen von Wasserstoff und gasförmigen Si-Verbindungen in die Vakuumkammer, z. B. Silan, Dichlorsilan, Trichlorsilan.
- ➤ Thermische Zersetzung der Si-Verbindungen in der Umgebung des Substrats, Abscheidung von Si-Atomen auf dessen Oberfläche.
- ➤ Schichtenwachstum in der Ebene energetisch bevorzugt, bis eine komplette Schicht abgeschieden ist.
- ➤ Dotierung kann parallel durchgeführt werden, etwa durch Zugabe gasförmiger Borverbindungen (Diboran, p-Dotierungen) bzw. Phosphor- und Arsenverbindungen (Phosphin, Arsin, n-Dotierung).

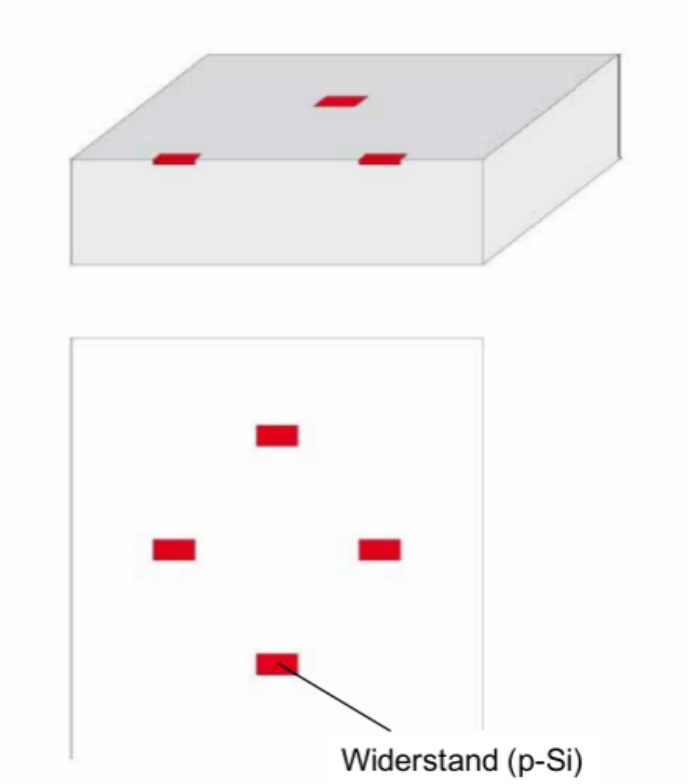
17.6. Drucksensoren

Si-basierte Sensorik/MEMS

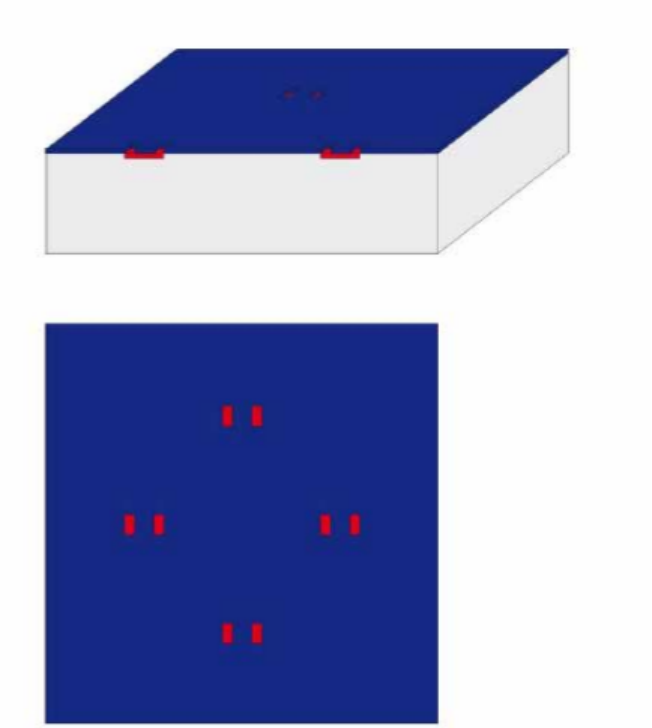
Fertigungsschritte

Piezoresistiver Si-Drucksensor: Herstellung der Funktionsbereiche (Resisitive Sensoren der Wheatstone Messbrücke)

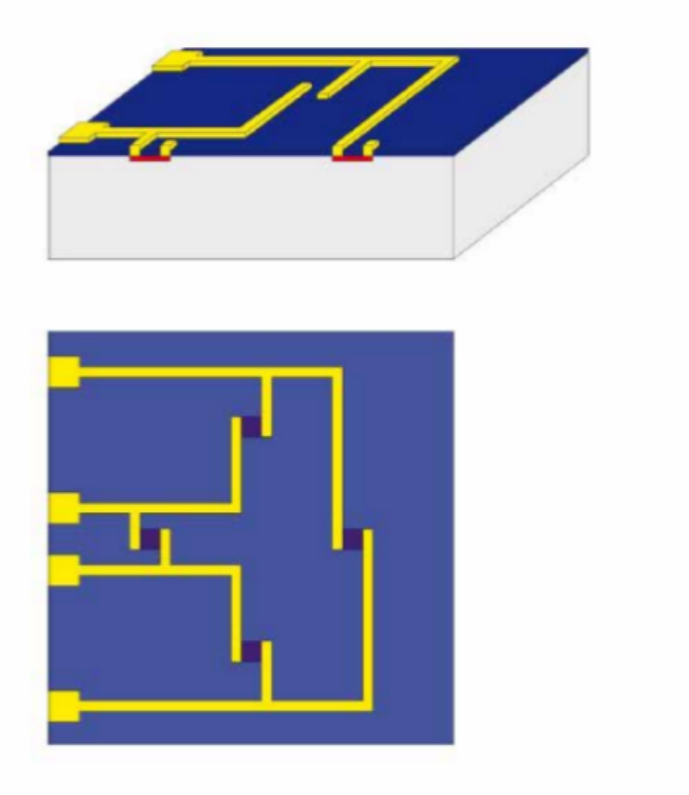
- ➤ Aufbringen des Photoresist z. B. mittels Spin Coating
- ➤ Photolithographie: Belichtung durch Maske
- ➤ Lokale Entfernung des Resist
- ➤ Dotierung in den freigelegten Bereichen über Diffusion (der einfachere, aber schlechter kontrollierbare Prozess) oder Ionenimplantation
- ➤ Entfernen des restlichen Resists



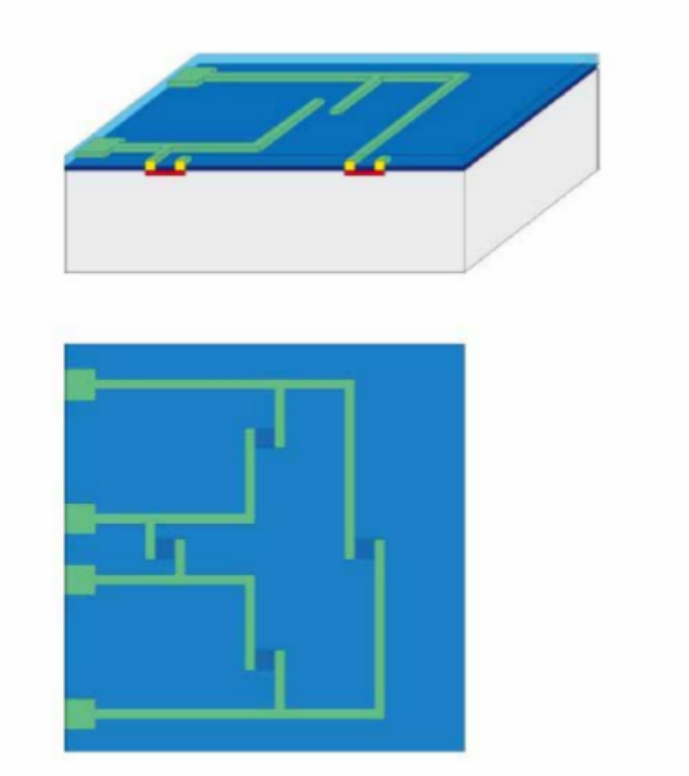
- Abscheiden einer SiO_2- und einer Si_3N_4-Schicht mittels Low Pressure CVD (LPCVD): Isolationsschicht zur Aufbringung der Leiterbahnen
- Photolithographie: Kontaktierung der Widerstände
- Trockenätzen der Si_3N_4-Schicht mit RIE, Nassätzen der SiO_2-Schicht mit Flusssäure (HF)



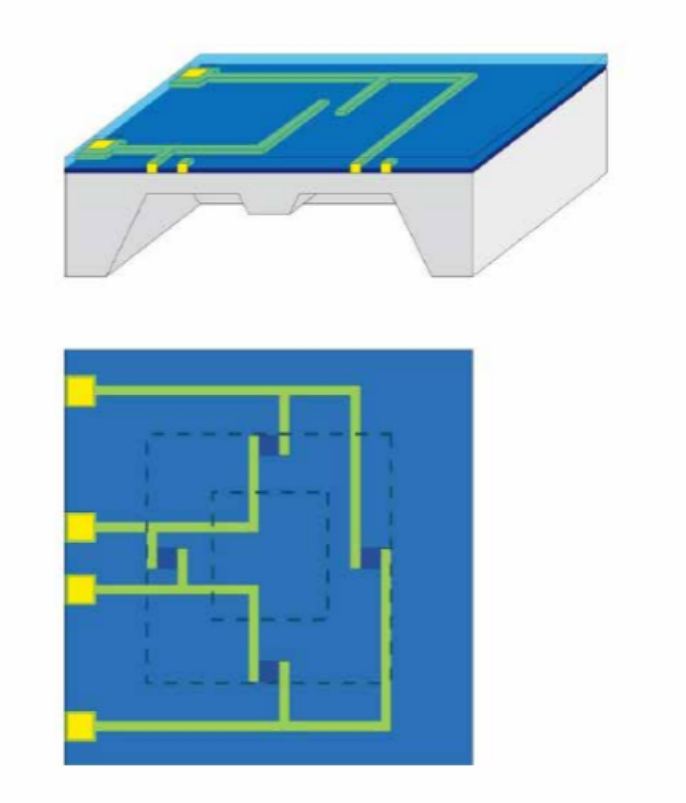
- Abscheiden einer Al-Schicht als Basis für die Leiterbahnen durch Aufdampfen (PVD)
- Strukturierung der Leiterbahnen über Photolithographie
- Entfernung des überschüssigen Aluminiums mittels Nassätzen



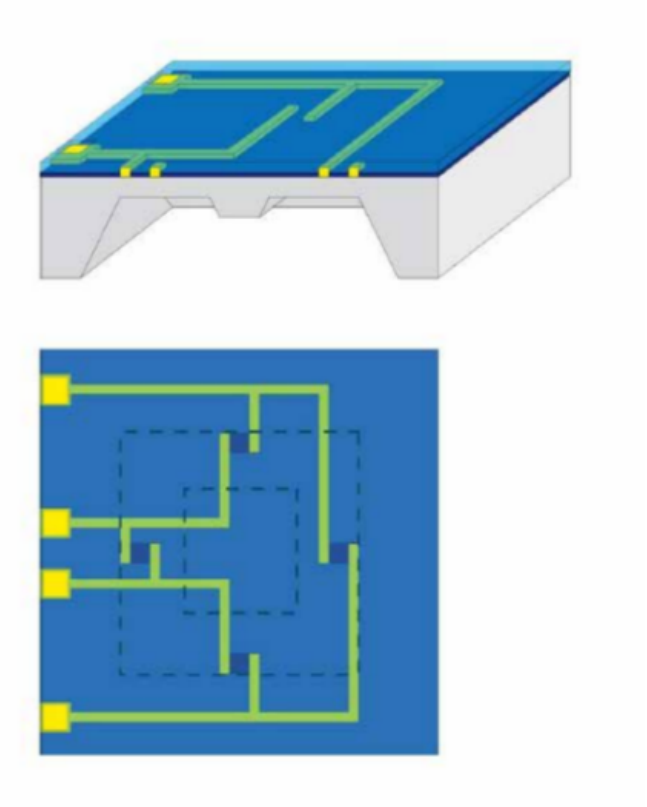
- Aufbringen einer Isolations- und Passivierungsschicht auf Si_3N_4-Basis mittels Plasma-Enhanced CVD (PECVD)
- LPCVD in diesem Schritt nicht möglich wegen Temperaturempfindlichkeit des Al



- ➤ Aufbringen einer rückseitigen Nitridschicht, Vorbereitung als Ätzmaske
- ➤ Lithographie, Trockenätzen der Si_3N_4-Schicht, Nassätzen der SiO_2-Schicht
- ➤ Nassätzen des Siliziums in KOH



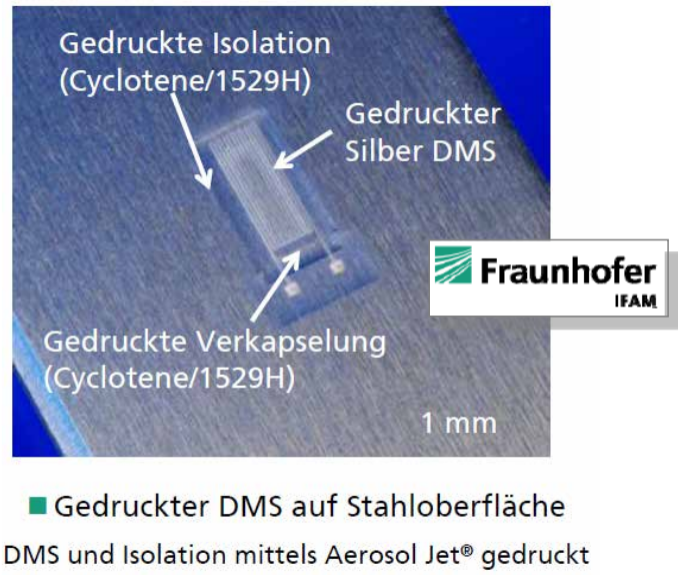
- Öffnen der Passivierungsschicht auf der Vorderseite zur elektrischen Kontaktierung der Sensoren
- Photolithographie, Trockenätzen der Si_3N_4-Schicht
- Alle beschriebenen Prozesse finden auf Wafer-Ebene statt, das heißt, es werden mehrere Chips gleichzeitig verarbeitet.



17.7. Druckverfahren

Gedruckte Sensorik

- DMS: Herstellung über **maskenloses Drucken**
- Erforderliche Komponenten:
  - Isolationsschicht,
  - DMS-Struktur, Verkapselung zum Schutz vor Umwelteinflüssen, mechanischer Beschädigung, etc.
- Druckverfahren hier:
  - Aerosol Jet-Druck,
  - alternatives maskenloses Verfahren: Inkjet-Druck



Aerosol Jet-Druck

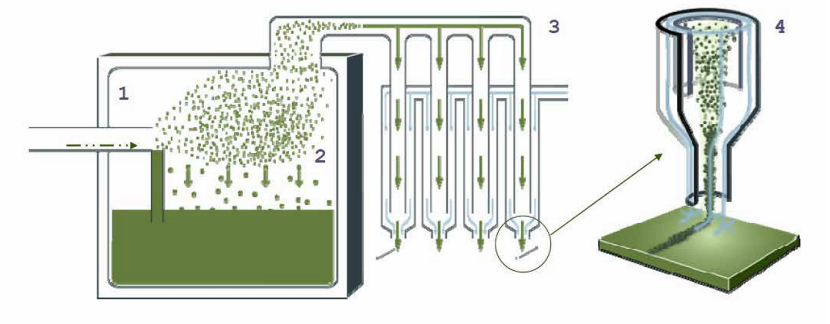


Figure 166. Schematische Darstellung Aerosol Jet-Druck (1) Gaseinleitung (2) Aerosolherstellung (3) Düsenverteilung (4) Aufbringung der Tröpfchen [Neotech Services]

Inkjet-Druck

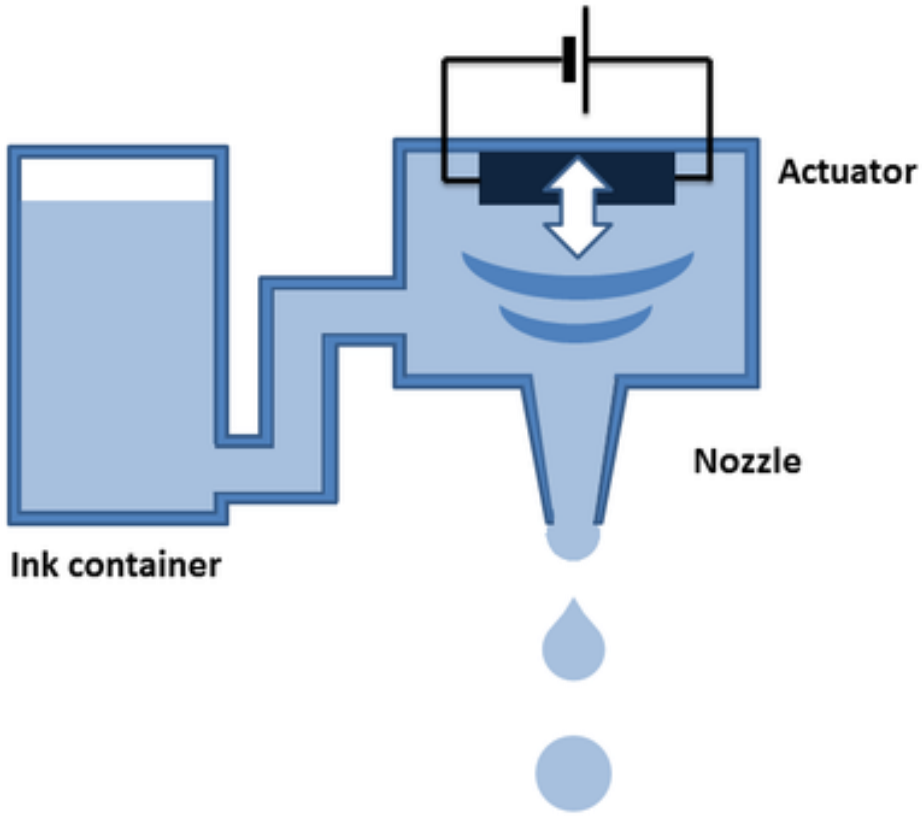
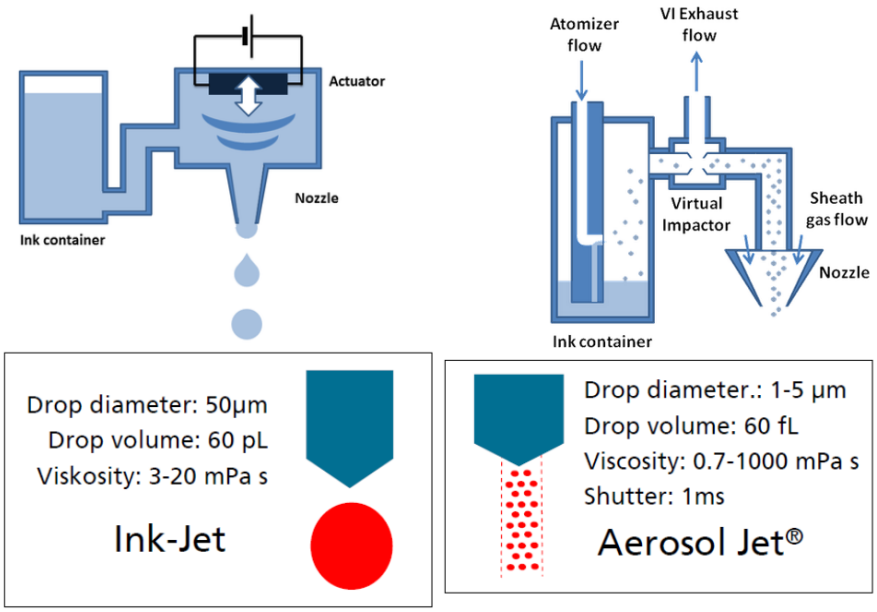


Figure 167. Schematische Darstellung Tintenstrahl(Inkjet)-Druck: Die Tintentropfen werden durch Ultraschallwellen direkt durch die Düse aufgetragen

➤ Vergleich der beiden Druckverfahren hinsichtlich:

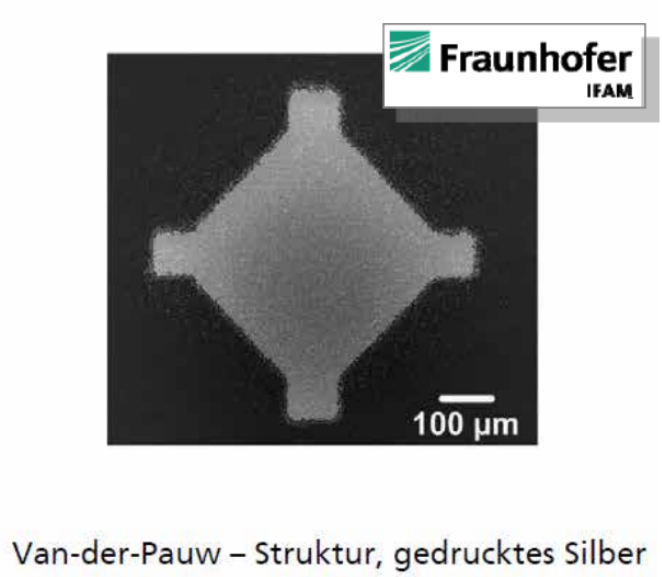
- ❏ Erzielbare minimale und maximale Strukturgröße
- ❏ Viskosität
- ❏ Art/Material der Partikel/Tinte



[iiw.kuleuven.be]

Sintern

- Sintern (Verschmelzung) der gedruckten Struktur ist nach dem Druckvorgang erforderlich!
- Beispiel Silbertinte:
- Ofensinterung unter H_2, N_2, Argon, … (bis 1700°C)
- Lasersintern (lokaler Energieeintrag)
- Mikrowellensintern
- UV Curing
- Sintern unter Druck (mechanisch)
- elektrisch Sintern
- Ziele: Entfernung organischer Bestandteile, Erhöhung der elektrischen Leitfähigkeit, Erhöhung der mechanischen Stabilität.



Van-der-Pauw – Struktur, gedrucktes Silber

17.8. Chipdünnung

- ➤ Silizium ist spröde und bruchempfindlich
  - ❑ Kann nicht nennenswert gebogen und nur wenig gedehnt werden bevor Bruch auftritt
- ➤ Aber: Die eigentliche funktionale Schicht eines Siliziumhalbleiters ist nur wenige Mikrometer dick!
  - ❑ Daher: Entfernung des Substrats und Reduktion auf funktionale Schicht
- ➤ Dünne Schichten sind wenigstens biegbar - Dehnung aber weiterhin ein Problem!

17.9. Zusammenfassung

- ➤ Da viele Sensoren als MEMS aufgebaut sind, werden vielfach für die Fertigung die Verfahren der Mikrosystemtechnik und der Mikroelektronik eingesetzt
- ➤ Silizium ist aus diesem Grund ein verbreiteter Substrat-, aber auch Funktionswerkstoff.

- ➤ Weitere Funktionalitäten können über Struktuierungsverfahren und/ oder über Integration weiterer Materialien eingebracht werden
  - ❑ Hierfür stehen z. B. PVD, CVD oder Druckverfahren zur Verfügung.
- ➤ Maskenlose Druckverfahren bieten hohe Flexibilität, Rolle-zu-Rolle-Verfahren hohe Produktivität auch für large area-Systeme, aber nicht die hohen Integrationsgrade der Siliziumtechnologie.

---

18. Materialintegration

18.1. Herausforderungen

Die vielen Probleme der Materialintegration:

Zentrale Herausforderung während der Fertigung von Materialien/Strukturen mit eingebetteten Sensorsystemen

- ➤ Mechanische Stabilität → Toleranz von mechanischer Belastung in Produktion und Betrieb
- ➤ Thermische Stabilität → Toleranz von thermischer Belastungen in Produktion und Betrieb

Funktionale Herausforderung

- ➤ Verträglichkeit mit der Matrix
  - ❑ Mechanische Anpassung: Anpassung an Steifigkeit, Festigkeit etc. des aufnehmenden Materials, Grenzflächeneigenschaften, Erzeugung innerer Spannungskonzentrationen, Eigenspannungen …
  - ❑ Thermische Anpassung: Anpassung des thermischen Ausdehnungskoeffizienten an das aufnehmende Material, Eigenspannungen …
  - ❑ Chemische und sonstige Verträglichkeit: Reaktion auf äußere Einflüsse wie Feuchtigkeit, spezielle chemische Einflüsse im Betrieb etc.

18.2. Vom Sensor zum System

Integration

Vorteile der Integration:

- ➤ Viel Funktion auf wenig Raum!
- ➤ Plug and Play: Einfachere Handhabung
  - ❑ Signal(vor-)verarbeitung und automatischer Kalibration, z.B. normierte Kennlinie, u. a. Drucksensor Streuung Si-basierter Drucksensor: ±1%, als integrierter Sensor: ±0,1%.
- ➤ EMV (Elektromagnetische Verträglichkeit): Kurze Signalwege für das ursprüngliche Sensorsignal, geringere Störanfälligkeit.
  - ❑ Stimmt das wirklich? Digitalelektronik und Wechselstromkreise induzieren unmittelbar Störungen in das Sensorsignal und sind hauptsächlich Störquelle.
- ➤ Herstellungskosten sind geringer
- ➤ Zuverlässigkeit: Systeme fallen an Schnittstellen aus

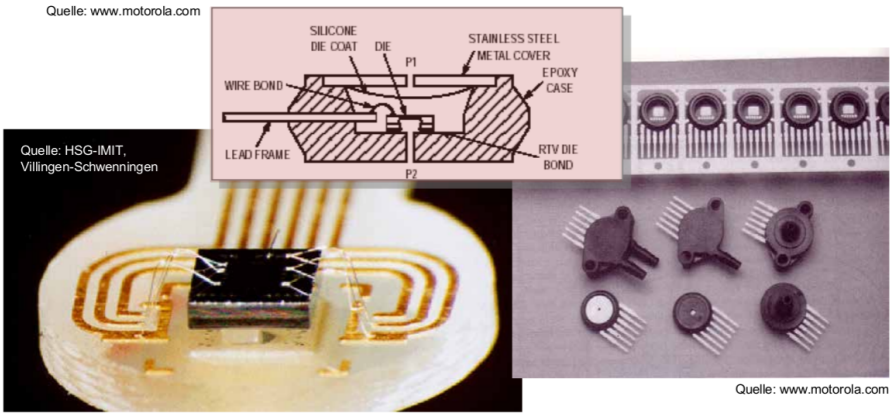


Figure 168. Packaging und Integration von Elektronik und Sensor in einem Gehäuse

18.3. Hybride Integration

Vorteile der hybriden Integration:

- ➤ Reduzierter Entwicklungsaufwand
- ➤ Mehr Freiheit und Flexibilität in der Auslegung/Gestaltung
- ➤ Möglichkeit der Reaktion (Produktanpassung) auf unabhängige Generationswechsel-Zyklen von Hauptkomponenten wie Datenverar-

beitung und Sensor (i. d. R. Generationswechsel beim Sensorelement langsamer)

- ➤ Prozessausbeute insgesamt größer wg. unabhängiger Auswahl der Komponenten

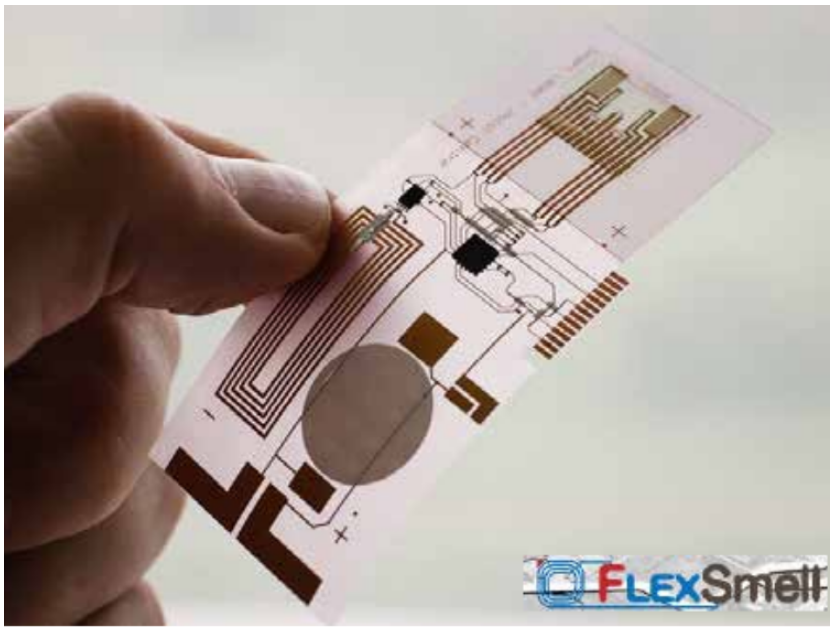
Nachteile der hybriden Integration

- ➤ Optimale Miniaturisierung und Funktionalität nicht erreichbar
- ➤ Erhöhte Fehleranfälligkeit durch Schnittstellen (mechanisch, elektrisch)

Folienintegration

"SmartLabel": Chemische bzw. Gassensorik inkl. Auswertung und Kommunikation.

- ➤ Folienintegration (System-in-Foil)
- ➤ Komponenten u. a. Sensoren, Antenne, Leiterbahnen, Energiespeicher, "naked die"- IC.
- ➤ Flexibilität durch Mikrochipdünnung



[Holst Centre, Eindhoven (NL)]

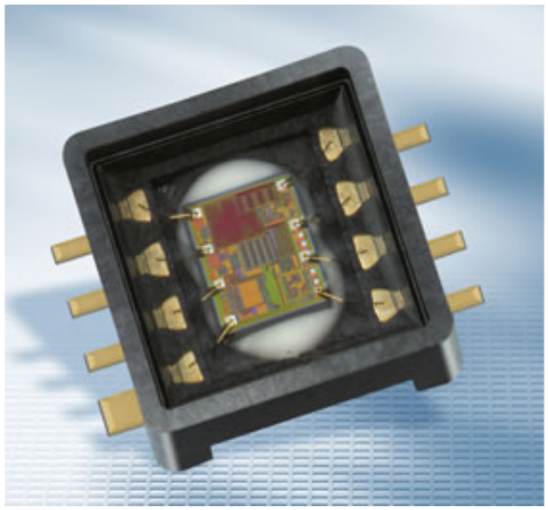
18.4. Monolithische Integration (System-on-Chip)

System-on-Chip (SoC) Technologie:

- ➤ Hochkomplexe, aber für große Serien auch hocheffiziente Fertigungsprozesse (vgl. CMOS-Prozess)
- ➤ Einschränkung des nutzbaren Technologiespektrums für den Bereich Sensorik auf (z. B.) CMOS-kompatible Prozesse
- ➤ Hohe Zuverlässigkeit (vgl. Aufbau- und Verbindungstechnik als häufigste Fehlerquelle)
- ➤ Kurze Signalwege zum Ort der Datenverarbeitung, geringe parasitäre Kapazitäten
- ➤ Integration analog/digital auf einem Chip u. U. problematisch]]

Beispiel Infineon KP125 Luftdrucksensor

- ➤ kapazitives Messprinzip
- ➤ Sensor hergestellt mittels surface micromachining
- ➤ monolithische Integration mit Signalkonditionierungs IC
- ➤ Herstellung im BiCMOS-Prozess
- ➤ kostengünstig dank großer Stückzahlen
- ➤ Ausgabe des Sensors: Analoge Spannung! ADC-DSP-DAC Kette!



[www.infineon.com/sensors]

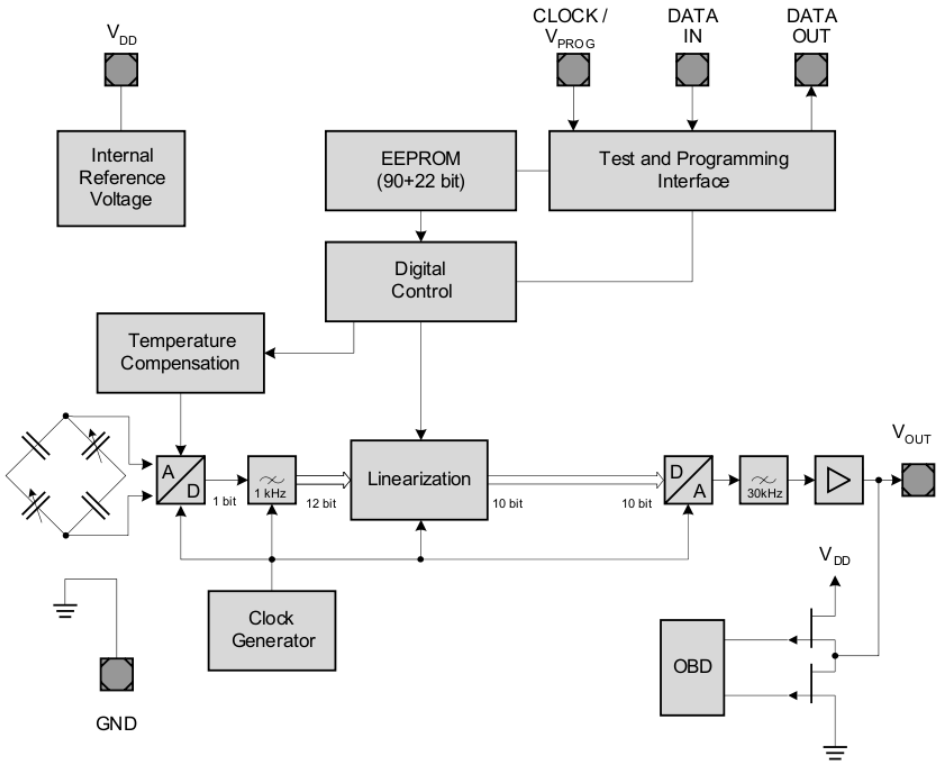


Figure 169. Analoge und Digitale Signalverabeitung in KP125 Drucksensor [www.infineon.com/sensors]

18.5. Gedruckte Elektronik und Sensorik

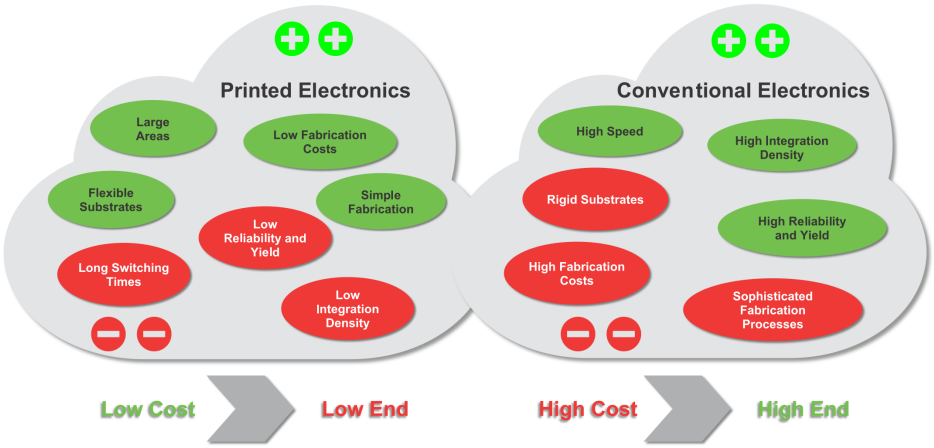


Figure 170. Vor- und Nachteile von Druckverfahren bei elektronischen Schaltungen

18.6. System-in/on-Foil (SiF)

➤ System-in-Foil technologien bieten ein hohes Potenzial für die flächige (2D) Materialintegration

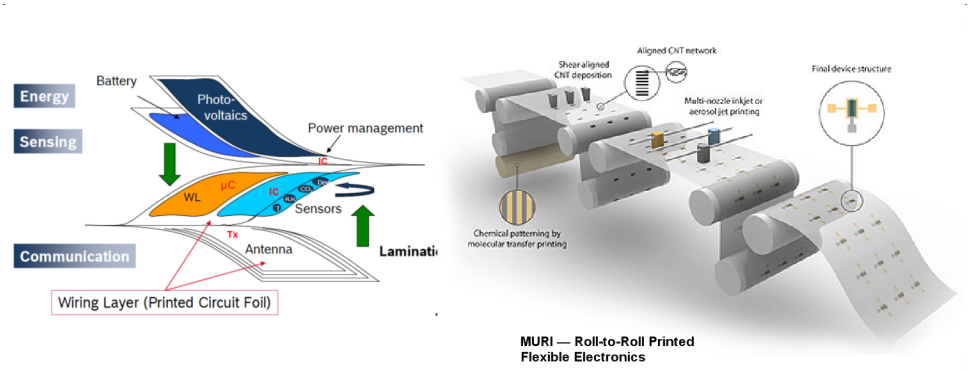


Figure 171. Integration von Sensoren, Elektronik, Energiegewinnung, Energiespeicherung, und Verbindung eingebettete zwischen Folien

Systemkomponenten

- ➤ Eine oder mehrere Verdrahtungsebenen
- ➤ Folienkomponenten: Sensoren, Batterien, Photovoltaische Zellen
- ➤ Integrierte Elektronik (gedünnt!)
- ➤ Passive Bauelemente (Widerstände, Kondensatoren, Induktivitäten)

Integrationsgrade

Homogene Integration

Das sensorische Material wird zusammen mit dem Funktionsmaterial/der Funktionsstruktur hergestellt (ein Prozess)

Heterogene Integration

Das sensorische Material wird es am Ende des Herstellungsprozesses des Trägermaterials/der Trägerstruktur aufgebracht (zwei Prozesse)

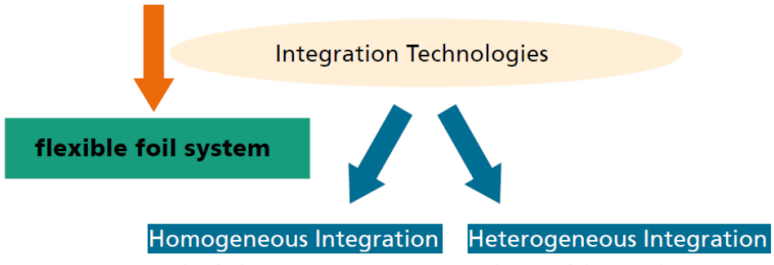


Figure 172. Die verschiedenen Integrationsgrade beim Folienverfahren im Vergleich

Kontaktierung

- ➤ Neben der Herstellung der Leiterbahnen ist die Kontaktierung der Bauelemente zentrales Problem

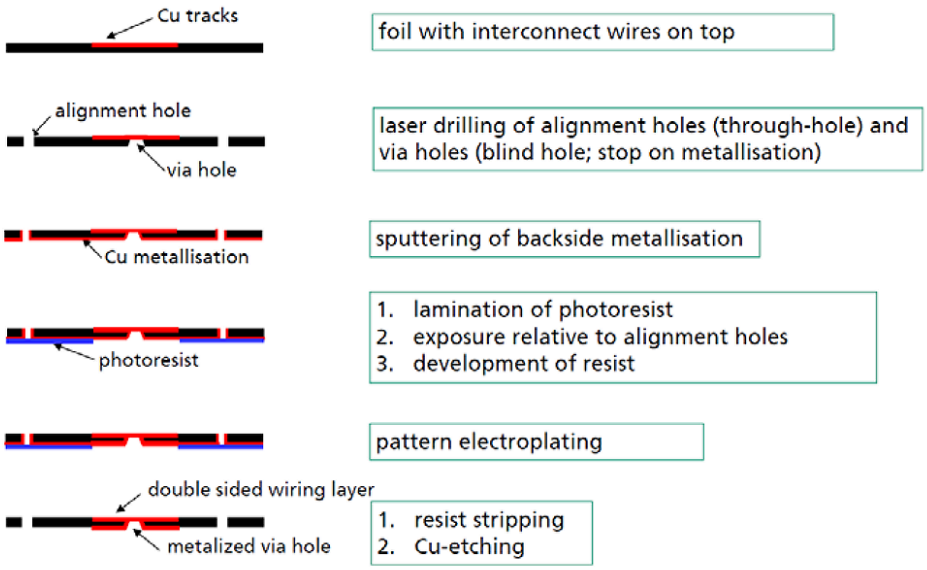
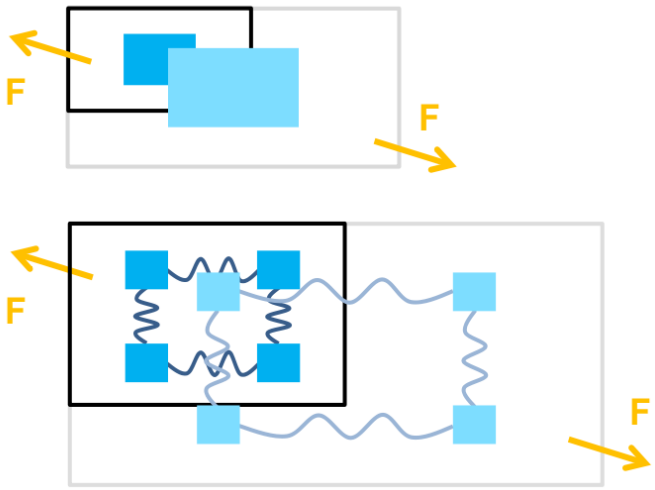


Figure 173. Folie mit beidseitigen Leiterbahnen: Kontaktierung in einzelnen Prozessschritten (aufwendig!!)

18.7. Mechanische Anpassung

Grundprinzipien, Last/Dehnung in der Ebene.

- ➤ Neutrale Ebene
- ➤ Dehnbare Komponenten
- ➤ Kapselung: Dehnbare Verbindungen zwischen herkömmlichen Komponenten, die starre Inseln innerhalb der Gesamtstruktur darstellen



➤ Eliminierung der mechanischen Belastung durch Positionierung in der bezogen auf Biegung “neutralen Ebene”:

❑ Unwirksam bei Belastung in der Strukturebene oder Druckbelastung dazu.

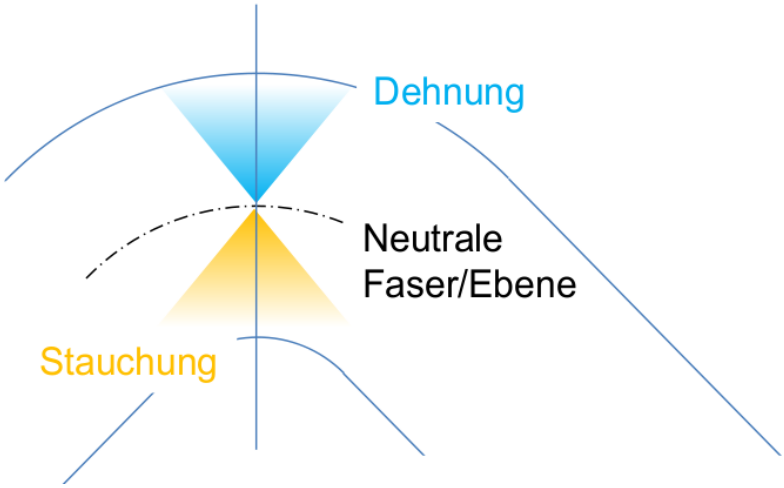


Figure 174. Starre Strukturen sollten in der neutralen Ebene eines Materials integriert werden. Aber: Z.B. Dehnungssensorik muss an die Randflächen!

➤ Viele integrierbare Strukturen und Materialien besitzen nur geringe Dehn- und Biegbarkeit

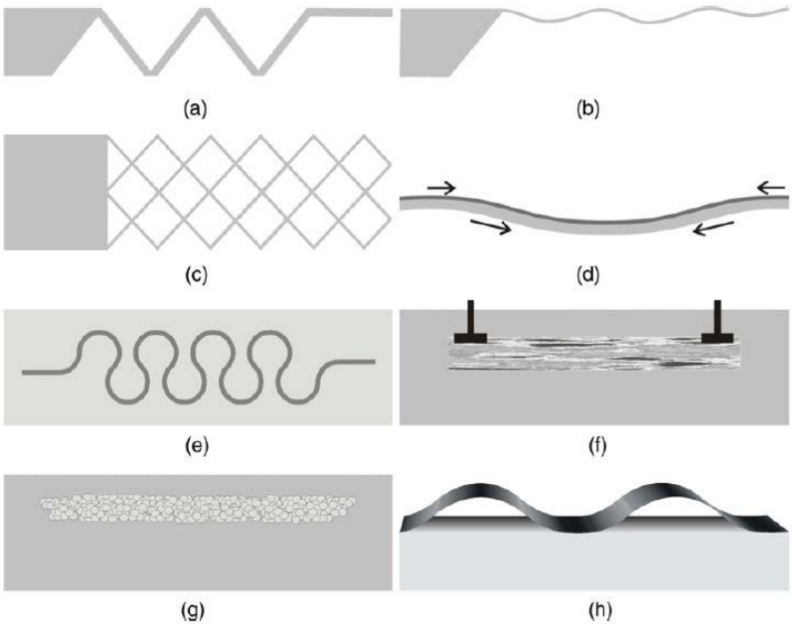
➤ Um Sensornetzwerke in Materialien und Verbundwerkstoffe integrieren zu

können bedarf es einer mechanischen Anpassung

➤ Komplexe Geometrien führen zu komplexer Verformung auch bei Belastung in der Ebene.

➤ Flexibilität als Dehnbarkeit über verteilte Strukturen:

a. Akkordeonstruktur
b. Wellenförmige Membranstruktur
c. Netzstruktur
d. Schichtstruktur
e. Hufeisen-Geometrie
f. Implantate
g. Flüssige Einbettung (Partikel)
h. Wellenförmige Struktur



18.8. Thermische Anpassung

➤ Der Temperaturausdehnungskoeffizient α ([10-6/K]) spielt bei der Kombination von Materialien und der Integration eine wesentliche Rolle

- Abweichungen der thermischen Ausdehnungskoeffizienten zwischen eingebetteten Komponenten und aufnehmendem Material führt zu mechanischen Spannungen:
  - Bei ursprünglich spannungsfreiem Zustand als Folge von Temperaturänderungen im Betrieb.
  - Im Verlauf der Herstellung bei Abkühlung von einem Prozessschritt, der mit erhöhter Temperatur einhergeht ggf. auch in Form von Eigenspannungen, die im Betrieb den lastinduzierten Spannungen überlagert sind.

Daher sollten Verbundwerkstoffe und integerierte Komponenten ähnliche Temperaturausdehnungskoeffizient besitzen!

19. Referenzen

Autoren

Die Werkstoff- und Fertigungsmodule und deren Illustrationen stammen von Dirk Lehmhus (Universität Bremen, ZWE ISIS)

Bücher

A.]ames E. Smith and R. Nair, Virtual Machines - Versatile Platforms for Systems and Processes. Elsevier Inc., 2005.

B. R. Zurawski, Ed., Embedded systems handbook : Embedded systems design and verification. CRC Press, 2009.

C. Eduard Glatz, Betriebssysteme - Grundlagen, Konzepte, Systemprogrammierung. dpunkt, 2015.

D. Vorlesung: Angewandte Sensorik, J. Zhang, Universität Hamburg, 2003

E. D. Balageas, C. Fritzen, and A. Güemes, Structural health monitoring. ISTE, 2006.

F. L. J. Gauckler, K. Conder Skript zur Vorlesung Ceramics II, ETH Zürich

G. Chapmann, Distributed Sesnor networks (DSN), 2005

H. Michael J. Flynn Wayne Luk, Computer System Design: System-on-Chip, Wiley, 2011

Veröffentlichungen