Auswahl der Entwicklungsumgebung

Echtzeitsysteme 2 – Vorlesung/Übung

Fabian Scheler
Michael Stilkerich
Wolfgang Schröder-Preikschat

Lehrstuhl für Informatik IV
Verteilte Systeme und Betriebssysteme
Friedrich-Alexander Universität Erlangen-Nürnberg

http://www4.cs.fau.de/~{scheler,mike,wosch}
{scheler,mike,wosch}@cs.fau.de

Überblick

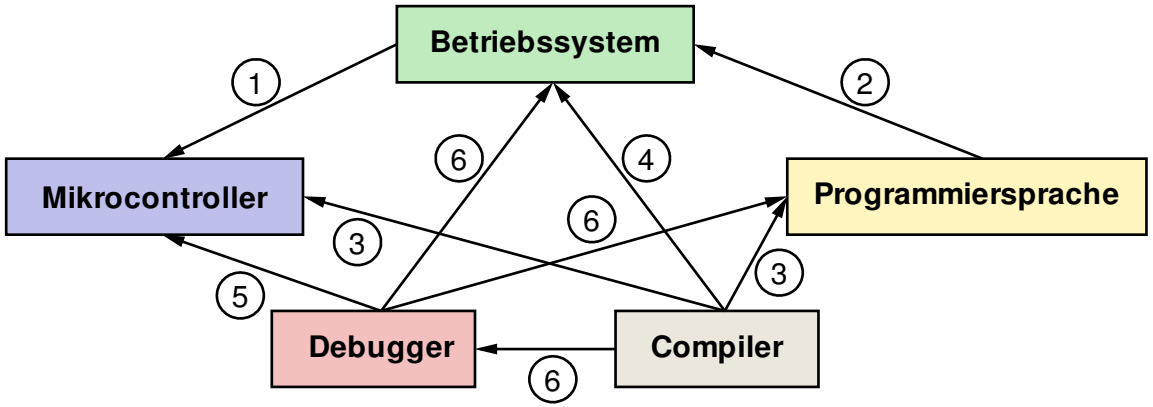
- Einleitung
- Ziele
- Entscheidungsfindung
  - Hardware, Mikrocontroller
  - Programmiersprache, Compiler
  - Betriebssystem

Einleitung

- Auswahl einer geeigneten Kombination aus
  - Mikrocontroller
  - Programmiersprache
  - Compiler
  - Betriebssystem
  - Debugger

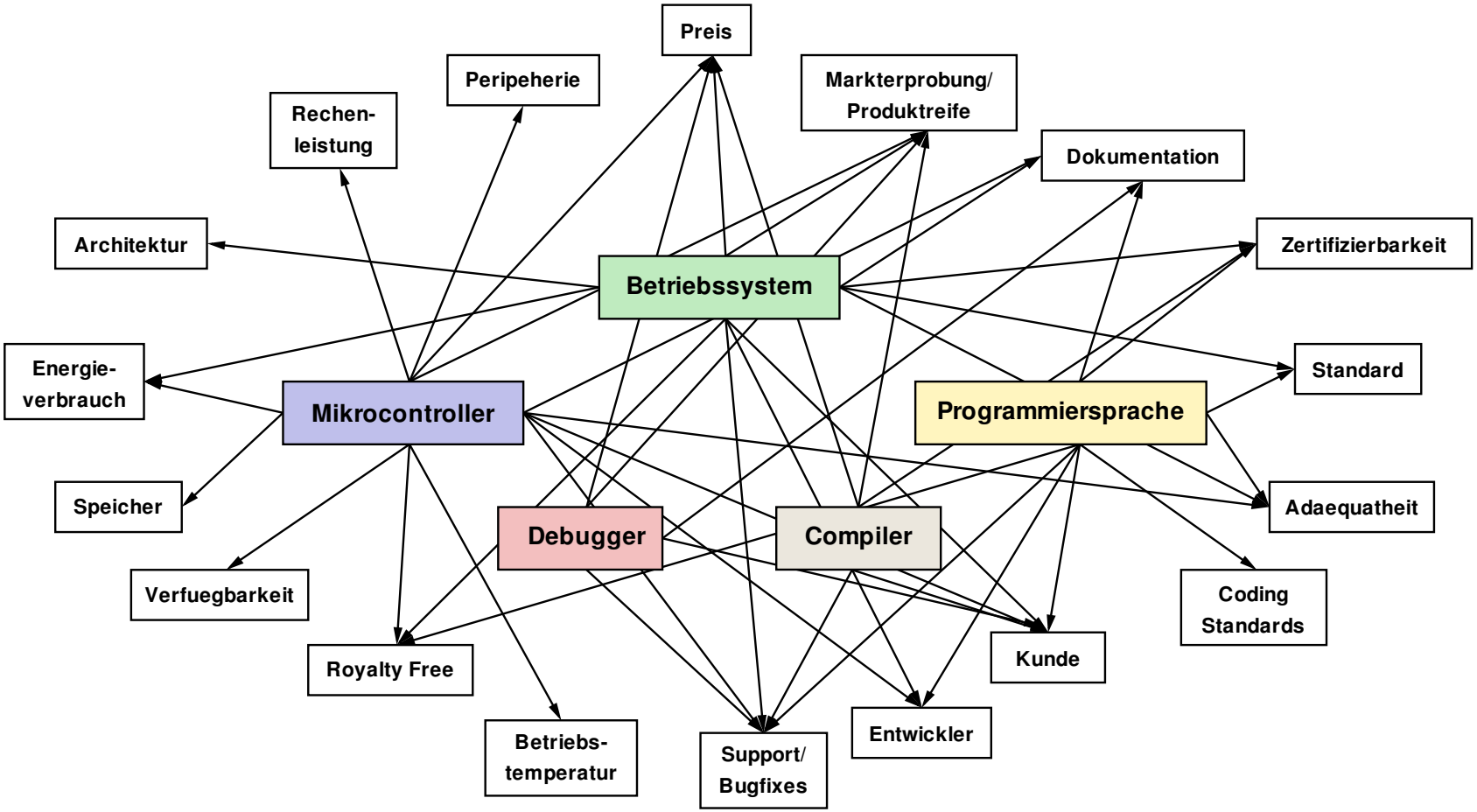
  ist schwierig
- auch (oder vor allem) ohne externe Randbedingungen
  - zwischen den einzelnen Komponenten existieren Abhängigkeiten
- externe Randbedingungen können die Auswahl
  - vereinfachen,
  - erschweren oder
  - unmöglich machen

Interne Abhängigkeiten



1) Gibt es Betriebssystem für den µController, welche Eigenheiten des µController werden unterstützt
2) Bietet das Betriebssystem eine passende Schnittstelle für eine Programmiersprache
3) Gibt es einen Compiler der eine best. Programmiersprache für einen bestimmten µController übersetzt
4) Kann ich das Betriebssystem mit einem best. Compiler übersetzen?
5) Gibt es für den µController passende Debugging-Tools?
6) Unterstützt der Debugger die Ausgabe des Compilers, das Betriebssystem und die Programmiersprache?

Externe Abhängigkeiten



Auswahlmöglichkeiten - µController

- dies ist nur ein Bruchteil

Auswahlmöglichkeiten - OS

- dies ist nur ein Bruchteil

Ziele ...

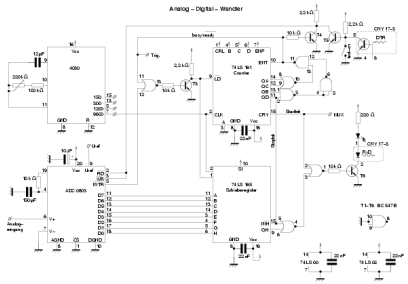
- dieses Auswahlprozesses: Optimierung gewisser Metriken
  - Performanz (Durchsatz, mittlere Antwortzeit, ...)
  - Kosten (pro Stück, nonrecurring engineering costs (NRE))
  - Energieverbrauch
  - Time-to-Market
  - Safety
  - ...

- dieser Vorlesung
  - Skizze einer Entscheidungshilfe
  - Aufzeigen einiger relevanter Eigenschaften

Hardware / µController

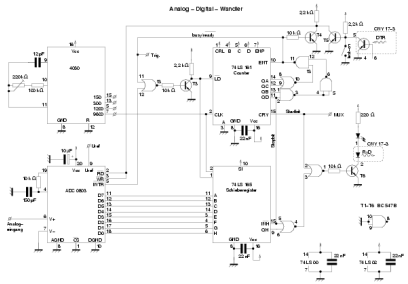
- Prozessortechnologie:
  - Vielzweck-Processor vs. Spezialzweck-Processor
- Speicher:
  - schreibbar vs. nicht schreibbar
  - flüchtig vs. nicht flüchtig
- Speicherarchitektur:
  - von Neumann vs. Harvard
- Instruction-Set Architecture:
  - CISC vs. RISC

Prozessortechnologie



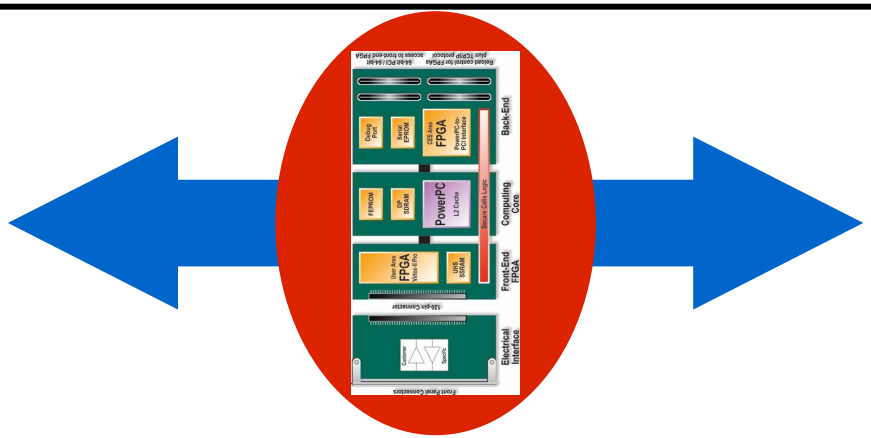
- Vielzweck-Prozessor
  - Standardprozessor, z.B. Pentium
  - große Teile des Systems entstehen in Software
  + hohe Verfügbarkeit, kurze Time-to-Market
  + geringe NRE-Kosten
  + hohe Flexibilität
  ~ Stückkosten: gering – hoch
  ~ Performanz: gering – hoch
  - Energieverbrauch
  - Chipgröße

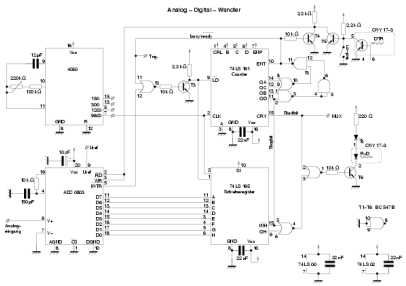
Prozessortechnologie



- Spezialzweck-Prozessor
  - erfüllt nur eine Aufgabe, z.B. JPEG Codec
  - wenig Software zur Realisierung des Systems erforderlich
  + geringe Größe
  + geringer Energieverbrauch
  ~ Stückkosten: gering – hoch
  - hohe NRE-Kosten
  - geringe Flexibilität
  - lange Entwicklungszeit

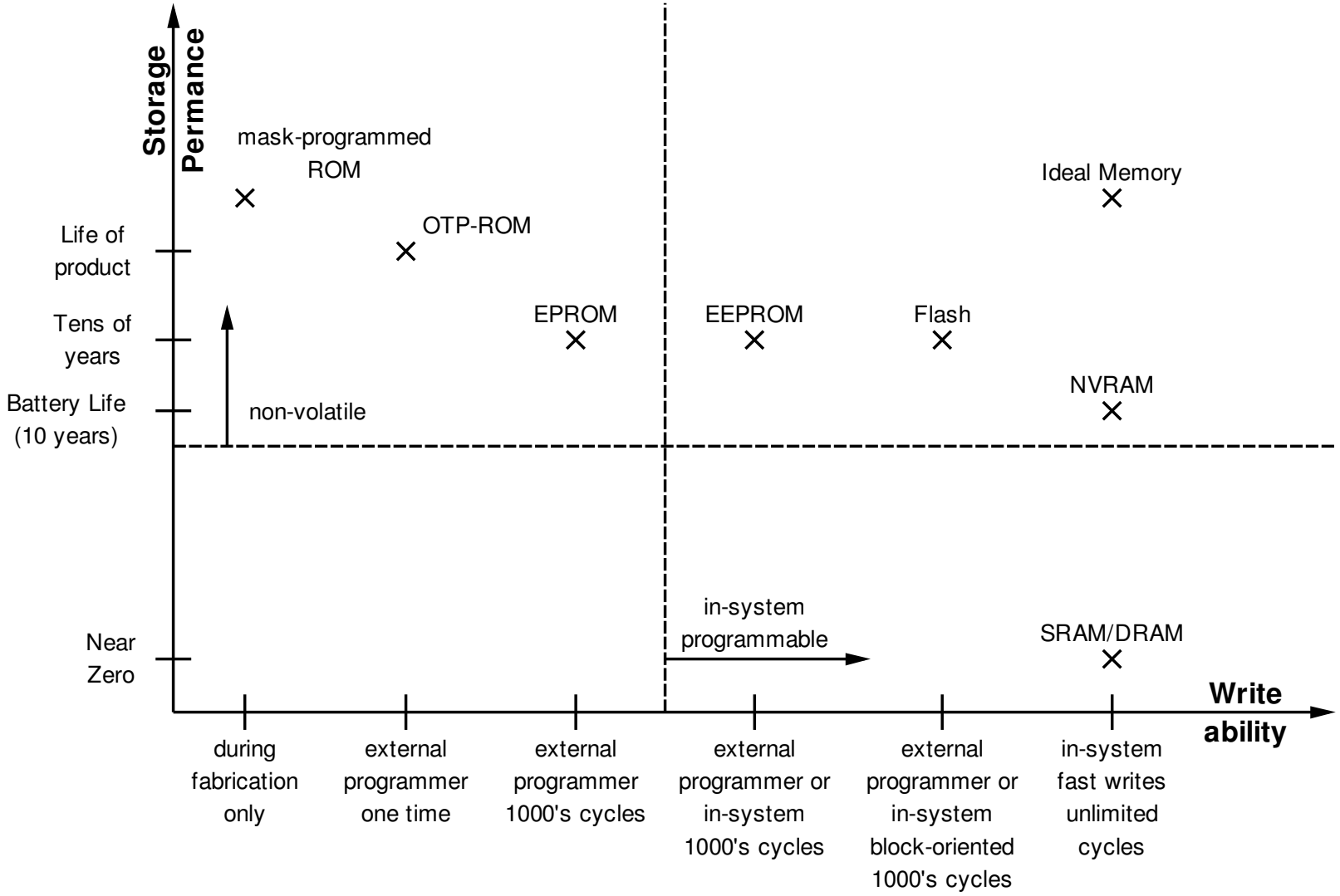
Prozessortechnologie





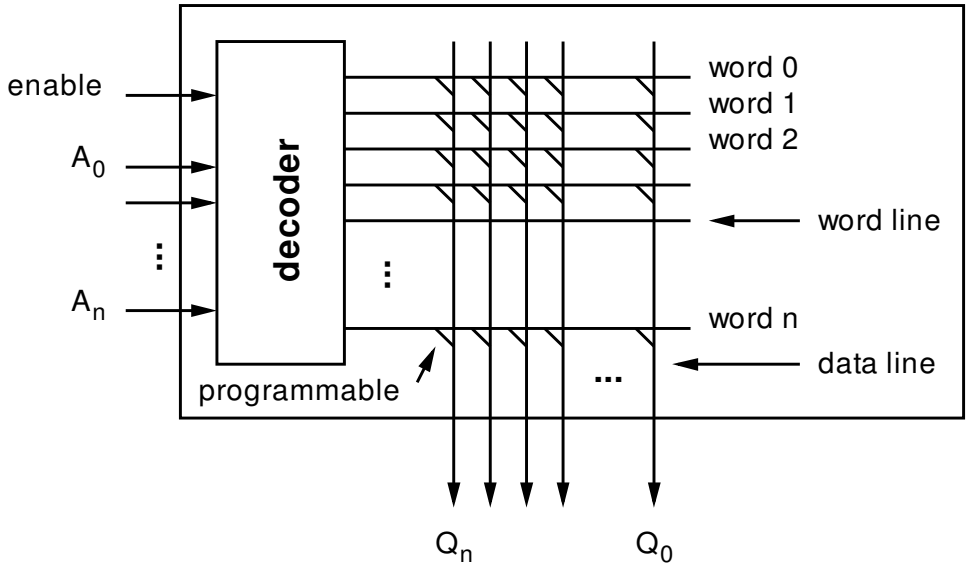
- Kompromiss: anwendungsspezifischer Prozessor (ASIP)
  - optimierte Prozessoren (z.B. µController, DSP)
  - Kombinationen aus Prozessoren und FPGA / ASIC
  - \+ Performanz
  - \+ Energieverbrauch
  - \- hohe NRE
  - \- hohe Entwicklungszeit
  - \- Unterstützung durch Entwicklungswerkzeuge

Speicher


Storage Permance
mask-programmed ROM
Ideal Memory
OTP-ROM
Life of product
Tens of years
EPROM
EEPROM
Flash
NVRAM
Battery Life (10 years)
non-volatile
in-system programmable
SRAM/DRAM
Near Zero
Write ability
during fabrication only
external programmer one time
external programmer 1000's cycles
external programmer or in-system 1000's cycles
external programmer or in-system block-oriented 1000's cycles
in-system fast writes unlimited cycles

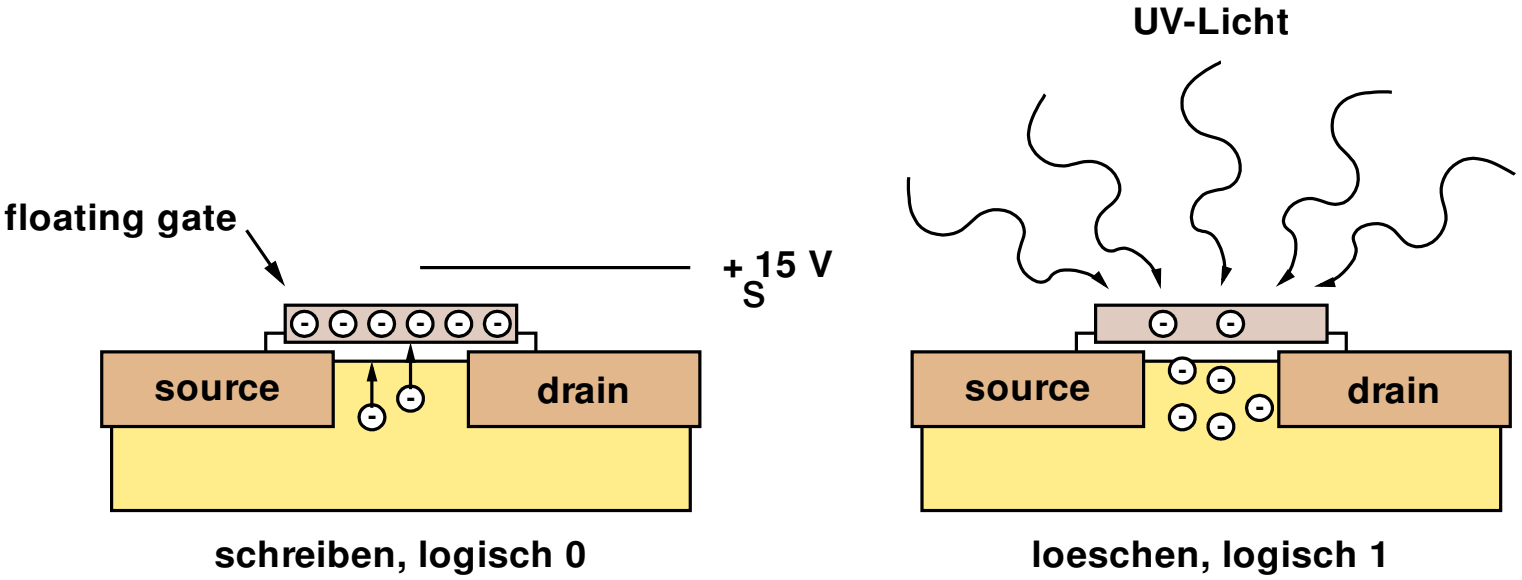
Speichertypen (1)

- mask-programmed ROM
  - Programmierung durcht entsprechende Belichtung des Siliziums
  - sehr unflexibel, hohe NRE-Kosten
  - sehr stabil
- one-time programmable (OTP) ROM
  - externes Programmiergerät
  - *fusible link* – Programmierung durch Durchtrennen von Sicherungen
  - sehr geringe Stückkosten
  - sehr stabil
  - unflexibel



Speichertypen (2)

- erasable programmable ROM (EPROM)
  - externes Programmiergerät (*floating gate*)
  - löschbar durch Bestrahlung mit UV-Licht (ca. 5 – 30 min)
  - anfällig für radioaktive Strahlung und elektrische Felder
  - werden selten in der Produktion eingesetzt



Speichertypen (3)

- electrically erasable programmable ROM (EEPROM)
  - externes Programmiergerät, in-system
  - elektronisch schreib- und löschbar (wortweise)
  - ansonsten wie EPROM
- Flash memory
  - elektronisch schreib- und löschbar (blockweise)
  - größere Bereiche können schneller geschrieben/gelöscht werden als bei EEPROM
  - ansonsten wie EPROM

Speichertypen (4)

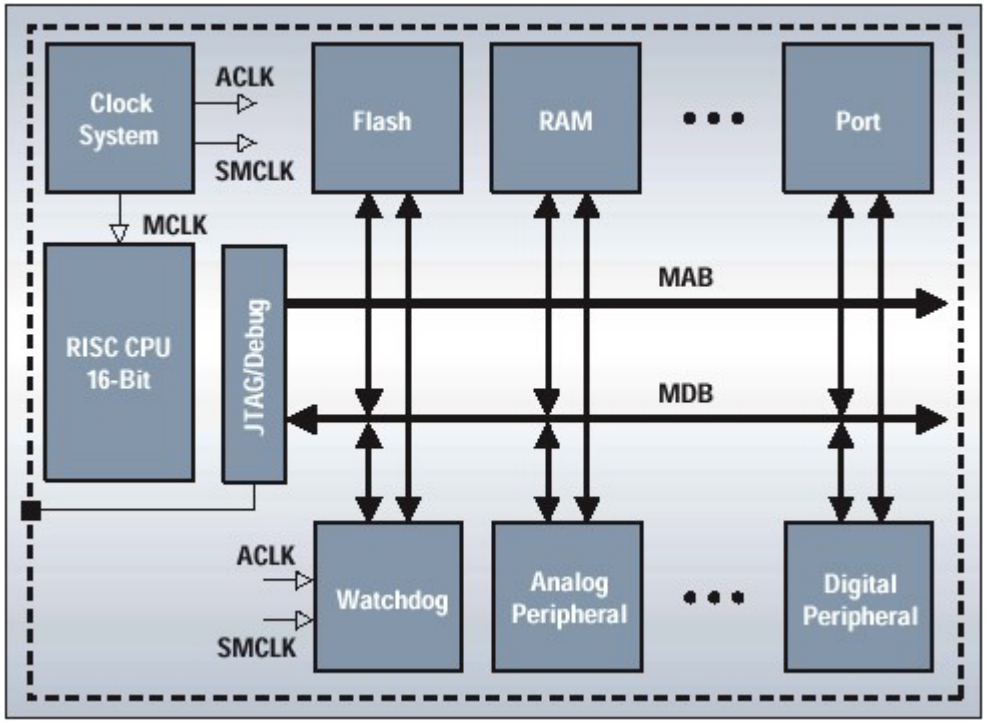
- static RAM (SRAM)
  - ein Flip-Flop je Bit: 6 Transistoren
  - kein Refresh notwendig: static RAM
- dynamic RAM (DRAM)
  - ein Transistor + ein Kondensator pro Bit: kompakter als SRAM
  - Speicher verliert Inhalt: Refresh notwendig
- pseudo static RAM (PSRAM)
  - DRAM mit integriertem Refresh-Controller
- nonvolatile RAM (NVRAM)
  - batterie-gestütztes RAM
- Forschung: MRAM bzw. FeRAM

Speicherarchitektur

- zwei verschiedene Anwendungsgebiete
  - Datenverarbeitung
  - mathematische Berechnungen
- Datenverabeitung
  - Beispiele: Textverarbeitung, Betriebssystem, Datenbanken
  - Hauptaufgaben: Daten verschieden, Bedingungen prüfen
  - Charakteristik: tendenziell sequentiell
- mathematische Berechnungen
  - Beispiele: digitale Signalbearbeitung, Simulation
  - Hauptaufgaben: Addition, Multiplikation
  - Charakteristik: tendenziell parallel

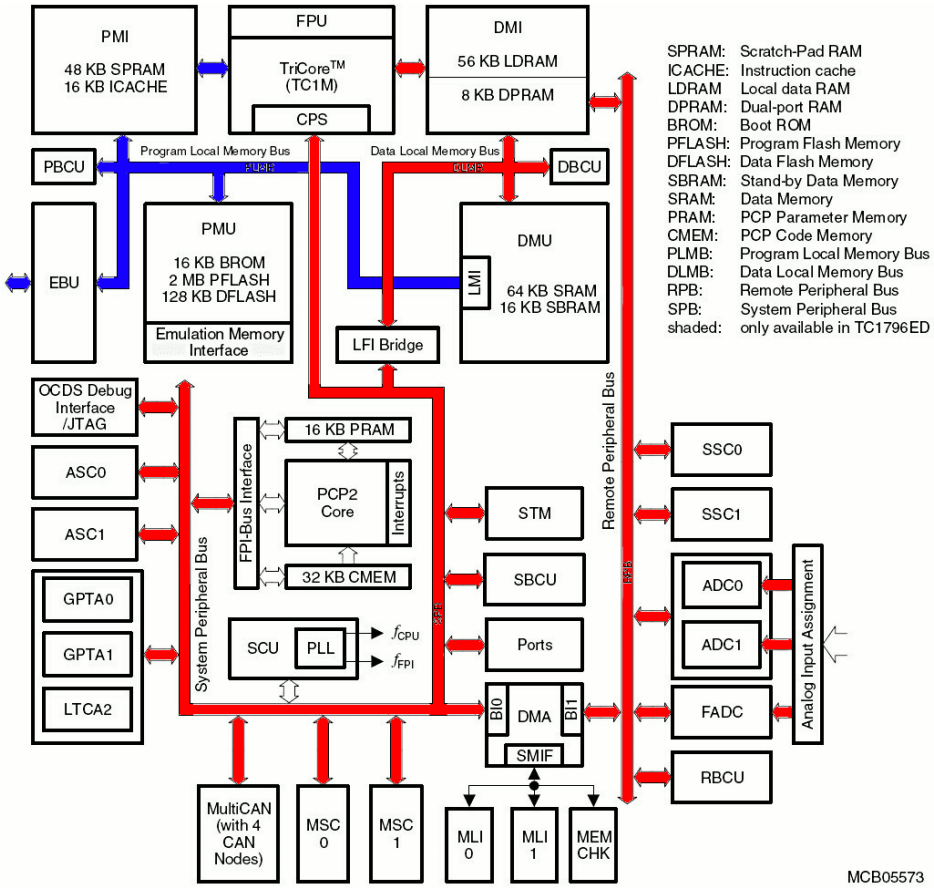
von Neumann Architektur

- gemeinsamer Speicherbereich
  - Daten und Programm finden sich im selben Speicher
- gemeinsamer Bus
  - Daten und Programm gehen über denselben Bus
  - → Flaschenhals!!!
- Vorteil: billiger
- Beispiel: MSP430



Harvard Architektur

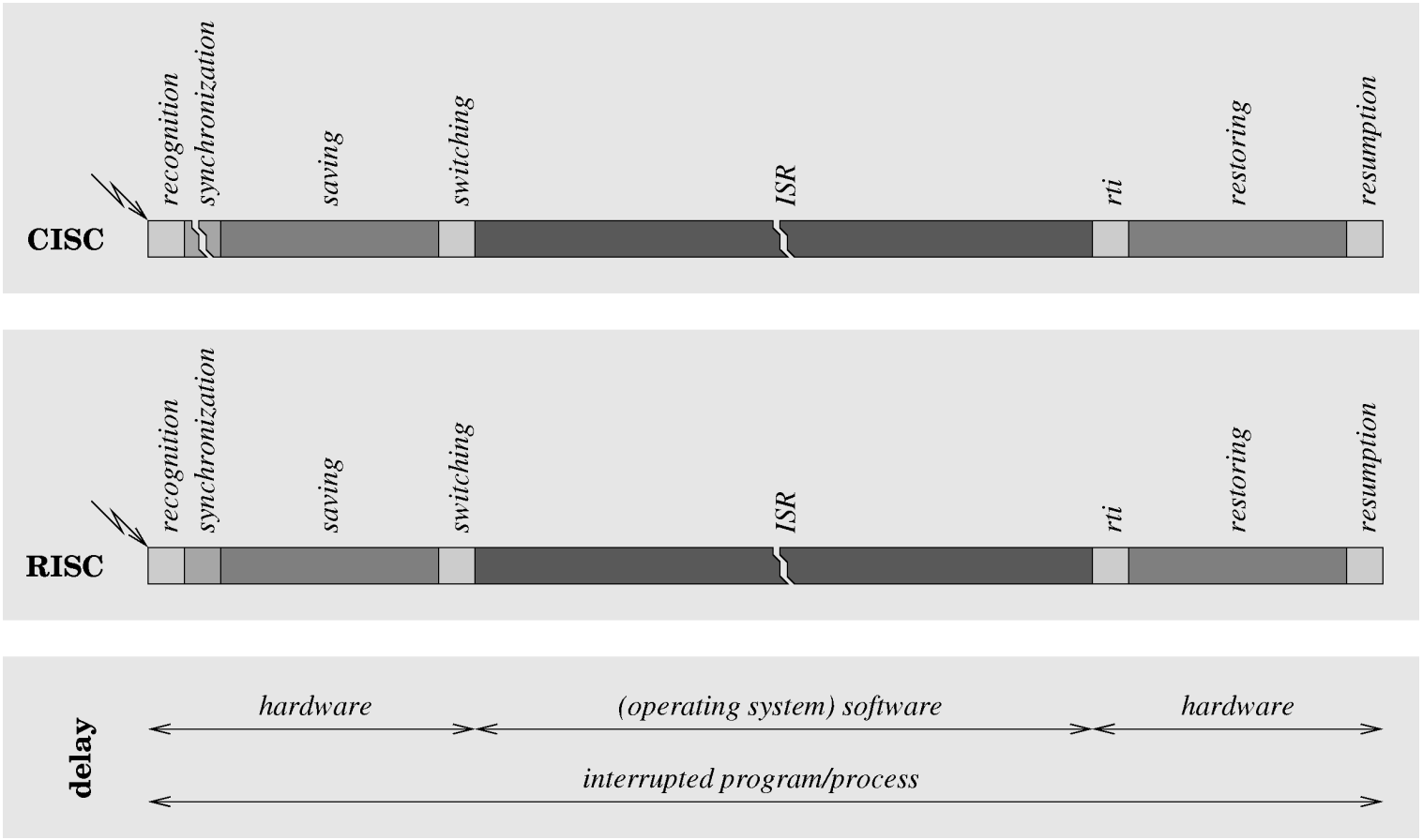
- getrennte Speicherbereiche
  - Daten und Programm finden sich in verschiedenen Speicherbereichen
  - teilweise: getrennte Register
- getrennte Busse
  - Daten und Programm gehen über verschiedene Busse
  - → weniger Konkurrenz um den Bus zwischen Daten und Programm
- Beispiel: TriCore



Instruction Set Architecture

- Complex Instruction Set Computing (CISC)
  - mächtige Befehle, umfangreicher Befehlssatz
  - zahlreiche Adressierungsarten
  - Konsequenzen
    - \+ kompakter Programmcode
    - ~ Befehle belegen unterschiedlich viel Speicher
    - \- Befehle dauern unterschiedlich lange
- Reduced Instruction Set Computing (RISC)
  - relativ einfache Befehle, überschaubarer Befehlssatz
  - LOAD / STORE Architektur
  - Konsequenzen
    - \+ Befehle dauern alle gleich lange
    - ~ Befehle belegen alle gleich viel Speicher
    - \- größerer Programmcode
- einzelne Befehle sind i.d.R. nicht unterbrechbar

ISA - Interruptlatenz



weitere Entscheidungsgrundlagen

- IC-Technologie:
  - full custom / VLSI vs. CPLD / FPGA
- Kommunikationssysteme:
  - time-triggered vs. event-triggered

Betriebssystem

- Grundsätzliche Frage
  - Common of the shelf (COTS) RTOS oder Eigenbau?
- Eigenbau
  + Ressourcen-schonend
  + exakt auf das System zugeschnitten
  - hoher Entwicklungsaufwand, hohe NRE
- COTS RTOS
  + geringer Entwicklungsaufwand, geringe NRE
  + Flexibilität
  - Overhead
  - höhere Stückkosten

Auswahl eines COTS RTOS

- grundsätzlicher Ansatz (Laplante)
  - wähle Kriterien c_1 bis c_n, mit $0 \leq c_i \leq 1$; $1 \leq i \leq n$
  - wähle Gewichte w_1 bis w_n, mit $0 \leq w_i \leq 1$; $1 \leq i \leq n$
  - 0 – nicht erfüllt bzw. nicht relevant,
    1 – komplett erfüllt bzw. sehr relevant
  - Bewertung des RTOS: $\sum_{i=1}^{n} c_i \cdot w_i$
- Problematik: verlässliche Information finden
  - Angaben der Hersteller sind oft für Marketingzwecke *frisiert*
  - Erfahrungsberichte anderer Anwender: aufwendige Recherche
- die Wahl des richtigen Betriebssystems ist entscheidend und sollte nicht zu einer *make-or-break* Entscheidung werden!

RTOS: Kriterien (1)

- minimale Interrupt-Latenz, c_1

  Zeit vom Auftreten einer Unterbrechung bis zur Ausführung der ersten Anweisung der benutzerdefinierten Behandlungsroutine

- Anzahl der untersützten Fäden, c_2

  Man benötigt eine gewisse Anzahl von Fäden, um alle Ereignisbehandlungen abbilden zu können. Die Anzahl der Fäden wird dabei nicht nur durch das Betriebssystem limitiert, sondern auch durch den zur Verfügung stehenden Speicher.

- Speicherbedarf des Betriebssystems, c_3

  Der Speicherbedarf des Betriebssystems allein, etwaige Anwendungsprogramme spielen keine Rolle!

RTOS: Kriterien (2)

- zur Auswahl stehende Scheduling-Verfahren, c_4

  Verschiedene Scheduling-Verfahren erlauben es, das System besser an die Anwendung anzupassen

- zur Auswahl stehende IPC-Verfahren, c_5

  Manche IPC-Mechanismen sind zur Formulierung gewisser Probleme einfach besser geeignet als andere, es erleichtert dem Benutzer die Arbeit, wenn er einen passenden Mechanismus auswählen kann.

- Customer-Support, c_6

  Das beste Betriebssystem ist nutzlos, wenn man es nicht benutzen kann - Hotline, Schulungen, Mailinglisten, Dokumentation ...

RTOS: Kriterien (3)

- zur Verfügung stehende Anwendungen, c_7

  Existieren Anwendungen oder Bibliotheken, die ich in meinem System wiederverwenden kann? Dies ist besonders im Echtzeitumfeld schwierig.

- unterstützte Architekturen bzw. Peripherie, c_8

  Gibt es Portierungen dieses Betriebssystems für verschiedene Prozessortypen, welche Peripheriegeräte werden standardmäßig unterstützt? Wie viel Handarbeit ist notwendig, damit das Betriebssystem die Anwendung vollständig unterstützt?

- ist das Betriebssystem quelloffen, c_9

  Ist der Quelltext des Betriebssystems für den Entwickler les- oder sogar veränderbar, um Vorgänge im Betriebssystem besser nachvollziehen zu können oder anpassen zu können? Wird das Betriebssystem als Binärabbild ausgeliefert?

RTOS: Kriterien (4)

- Kontextwechsel, c_{10}

  Wie viel Zeit benötigt das Betriebssystem um den Wechsel des laufenden Fadens durchzuführen?

- durch das Betriebssystem verursachte Kosten, c_{11}

  Zu diesen Kosten zählen sowohl Ausgaben, um Arbeitsplätze mit den entsprechenden Entwicklungswerkzeugen auszustatten, als auch später in der Produktion des Systems anfallende Lizenzkosten.

- zur Verfügung stehenden Entwicklungswerkzeuge, c_{12}

  Gibt es Werkzeuge, die den Entwickler beim Umgang mit dem Betriebssystem unterstützen? Implementiert das Betriebssystem einen Standard und ist es gegen anderen Implementierungen austauschbar?

RTOS: ProOSEK/time vs. eCos

Kriterium			ProOSEK	eCos
Nr.	Beschreibung	Gewichtung		
1	Interruptlatenz	1	1	0,7
2	Anzahl der Fäden	1	1	1
3	Speicherbedarf	1	1	0,6
4	Scheduling	0,8	1	1
5	IPC	1	1	1
6	Support	1	0,6	1
7	Anwendungen	0,5	0	1
8	Portabilität	0,1	0,7	1
9	Quellcode	0,1	0	1
10	Kontextwechsel	0,5	1	0,75
11	Kosten	0,1	0	1
12	Standards	0,1	1	1
		Ergebnis	5,97	6,19

Programmiersprache

„Misuse of the underlying programming language can be the single greates source of performance deterioration and missed deadlines in real-time systems.“

Phillip A. Laplante

Eignung einer Programmiersprache

- Metriken (informell) nach Cardelli:
  - *Economy of Execution*
    Wie schnell wird ein Programm ausgeführt? Ist die Programmiersprache inhärent mit einem Performanznachteil verbunden?
  - *Economy of Compilation*
    Wie schnell kann man ein Programm übersetzen?
  - *Economy of Small-Scale Development*
    Wie schwierig ist der Umgang mit der Programmierprache für einen einzelnen Programmierer?
  - *Economy of Large-Scale Development*
    Wie schwierig ist der Umgang mit der Programmiersprache für Team von Programmierern?
  - *Economy of Language Features*
    Wie schwer ist es, die Programmiersprache zu erlernen?
- zusätzlich in Echtzeitsysteme: **Analysierbarkeit**

Assemblersprachen

- Beispiele: je nach Prozessor
- \+ direkte Kontrolle über die Hardware
- \- kaum Abstraktionsmechanismen
- \- schwierig, fehlerbehaftet, aufwendig
- \- unportabel

➔ Assembler nur im Ausnahmefall verwenden!

Prozedurale Sprachen

- Beispiele: C, Fortran, Ada95, Modula-2
- \+ Strukturierung der Programme in Unterprogramme
- \+ Typisierung / abstrakte Datentypen
- \+ Ausnahmebehandlung
- \+ dynamische Speicherverwaltung
- ~ unterschiedlich stark ausgeprägte Modulkonzepte
- \- Kapselung

➔ in Echtzeitsystemen am häufigsten Anzutreffen

Objektorientierte Sprachen

- Beispiele: C++, C#, Java, Ada95
- statisch übersetzt vs. dynamisch ausgeführt
- \+ Kapselung und Information Hiding
- ~ Vererbung und Polymorphismus
- ~ Ausnahmebehandlung
- ~ Garbage Collection

➔ in Echtzeitsystemen (noch) selten eingesetzt, auch, weil fähige (noch) Entwickler fehlen

Metriken

		prozedural			objekt-orien-tiert		
	Assembler	C	Fortran	Ada95	C++	Ada95	Java
Execution	++	++	++	++	+	o	o
Compilation	++	++	++	++	+	+	+
Small-Scale Development	--	++	+	++	+	+	+
Large-Scale Development	--	o	-	+	++	++	++
Language Features	++	+	++	+	--	-	--
Analysierbarkeit	++	-	++	+	--	o	--

Zusammenfassung

- Auswahl der geeigneten Entwicklungsumgebung ist schwierig
  - es existieren eine Fülle interner und externer Abhängigkeiten
- Hardware
  - Vielzweck-Prozessor vs. Spezialzweck-Prozessor
  - diverse Speichertypen
  - von Neumann vs. Harvard
  - RISC vs. CISC
- Betriebsystem
  - Eigenbau vs. COTS
  - schwierige Informationsbeschaffung
  - Metrik: Gewichtung einer Menge von Kriterien
- Programmiersprachen
  - Assembler, prozedurale Sprachen, objekt-orientierte Sprachen
  - Metriken nach Cardelli