Die hohe Einsicht wohnt nicht in den einzelnen Kammern, sondern im Gefüge der Welt.

Ernst Jünger

Entwerfen heißt, eine bestimmte Aufgabe mit zuhandenen technischen Mitteln auf wirtschaftlich und anwendungspraktisch sinnvolle Weise zu lösen.

1. die Aufgabe

- Realzeitraster: in welcher Zeit ist es zu tun?
- Funktionalität: was ist zu tun? (die auszuführenden Informationswandlungen).

2. die Voraussetzungen und Nebenbedingungen

- Zuhandenheit,
- Infrastruktur,
- Kosten,
- Zeitbedarf (Einordnung in eine Terminplanung, Time to Market),
- Akzeptanz.

Das Realzeitraster

- in welcher Zeit ist dieAufgabe zu erledigen?
- einmalig oder zyklisch?
- Latenzzeit: von der Anforderung bis zum Beginn der Reaktion.

Ziel:
Durchführung der geforderten Informationswandlungen in der vorgegebenen Zeit t mit kostenoptimalen Mitteln (Kosten über alles = über den gesamten Lebenszyklus = Total Cost of Ownership).

Je geringer die Zeitvorgabe t und je größer die Kompliziertheit bzw. Komplexität, um so größer der Aufwand

Komplexität und Kompliziertheit sind keine Wechselworte:

Komplexität beschreibt Ressorcen-Anforderungen des Algorithmus in Abhängigkeit von der Problemgröße. Beschreibung hat die Form O (n) (Order of... = Größenordnung von...). *Komplex* = mehr als linear mit der Problemgröße wachsende Anforderungen (Verarbeitungsleistung, Speicherplatz).

Kompliziertheit = Spitzfindigkeit, Verwickeltheit, Vielfalt der Funktionen. *Kompliziert* = verwickelt (sophisticated). Läßt sich näherungsweise durch Umfang der Problembeschreibung (= funktionelle Spezifikation) kennzeichnen.

Nutzung zuhandener Mittel und Entwicklungstechnologien

- t groß: das kostengünstigste zuhandene Mittel aussuchen: reicht es aus?
- t extrem klein: ist die Aufgabe überhaupt realisierbar? - Es kommen nur schaltungstechnische Lösungen in Frage. Erfinderische Bemühungen gelegentlich notwendig.

Zeitraster	Beispiele	technische Mittel
10 ns	Befehlsausführung in Prozessoren, Speicheransteuerung, Videodarstellung	Hardware, Gatter-Ebene (auch: Transistor-Ebene)
100 ns	Gerätesteuerung	Hardware auf Gatter-Ebene, State Machines, Mikroprogrammierung
1 µs	Gerätesteuerung	Hardware auf Gatter-Ebene, State Machines, Mikroprogrammierung, Maschinenprogrammierung (was sich mit 10...50 Maschinenbefehlen erledigen läßt)
1 ms	Gerätesteuerung, Regelungsaufgaben (Closed Loop)	Realzeit-Software (was sich mit wenigen tausend Maschinenbefehlen erledigen läßt)
10 ms	Gerätesteuerung, Regelungsaufgaben (Closed Loop), Bedienung/Anzeige	Realzeit-Software
100 ms	Regelungsaufgaben (Closed Loop), Bedienung/Anzeige, Prozeßsteuerung, Informationsbeschaffung (Retrival)	komplexe Realzeit-Software, Vernetzung
1 s	Prozeßsteuerung, Informationsbeschaffung (Retrival)	Realzeit- bzw. beliebige Software (je nach Hardware-Plattform)
kommt nicht drauf an	Fertigungssteuerung, allgemeine Verwaltung	beliebige Software (auch Windows etc.), Vernetzung (incl. Internet)

Zuhandenheit

- Technologien,
- Beschreibungsmittel,
- Fachwissen (State of he Art), gedankliche Ansätze (Paradigmata), Problemverständnis,
- Verifizierungsmittel:
  - Testen,
  - Simulation,
  - Prüfen/Messen,
  - Fehlersuchen (Debugging).

Was man nicht prüfen kann (Funktionsnachweis), kann man eigentlich gar nicht realisieren.

Ebenen der Zuhandenheit:

- Basistechnologien (Si, GaAs usw.),
- Fertigungstechnologien (z. B. 0,35 µm CMOS),
- Bauelemente - fertige Schaltkreise, auch programmierbare:
  - Gatter-Ebene
  - Register-Transfer-Ebene
  - Funktionsblöcke (Building Blocks)
- Hardware-Plattformen,
- Entwicklungsumgebungen,
- Systemumgebungen.

Paradigmata:

- kombinatorischw Zuordnung, Table Lookup
- State Machines (abstrakte Automaten)
- Sequencer
- Controller
- universelle Prozessoren
- spezielle Prozessoren
- Signalverarbeitung analog (= Regelungstechnik, rückgekoppelte Systeme, Laplace-Transformation)
- Signalverarbeiung digital (z-Transformation)
- neuronal
- Fuzzy
- herkömmliche Programmierung (Programmablauf, Flowchart- bzw. if-then-goto-Paradigma)
- strukturiert
- Multitasking
- ereignisgesteuert
- objektorientiert
- relational (Datenbasis; Memory Based Reasoning)
- regelbasiert (Expertensysteme, Prolog usw.)
- Schaltplan
- Zustandsgraph
- Entscheidungstabelle
- Petri-Netz
- Boolesche Gleichung
- Kontaktplan (= herkömmliche Steuerungstechnik)

Ungewöhnliche Kombinationen = Erfindung (Pat Es gibt keinen Universalalgorithmus des Erfindens, wohl aber ein Methodenwissen
einschließlich einer gut gefüllten Trickkiste.

Vorgehen: Durchmustern der Wissensbasis, Zusammenstellen geeigneter Kombinationen.

Auf welcher Ebene der Zuhandenheit aufsetzen?
Abhängig von Realzeitraster-Vorgabe und deshalb fließendes Ziel in Abhängigkeit vom Stand der Technik:

- < 1 ns: Basis- (Halbleiter-) Technologie,
- wenige ns: Fertigungstechnologie (zuhandene Halbleiter-Technologien + Integrationsgrad + Kombinationen (Mixed Signal, Logik + DRAM usw.),
- < 1 µs: Bauelemenetechnologie (zuhandene Schaltkreise),
- vom ms-Bereich an: zuhandene Funktionseinheiten...fertige Systemumgebungen

Kostenoptimierung = nicht zuviel Overkill für die Problemlösung = gerade soviel, um Reserven für Änderungen zu haben (den gesamten Lebenszyklus bedenken!).

Software oder Hardware?
Im strengen Sinne gibt es die Alternative nicht, da jede Software eine Hardwareplattform braucht, um laufen zu können.

Beides zusammen = Kostenoptimum unter Erfüllung der vorgegebenen Realzeitanforderungen.

Stand der Technik:

- Hohe Anforderungen an die Funktionalität.
- Programmier-Paradigma hat sich soweit durchgesetzt, daß praktisch jedem Lösungsansatz für auch nur halbwegs komplexe funktionelle Anforderungen irgendeine Form der Programmierbarkeit zugrunde gelegt wird - kaum noch jemand baut hochkomplizierte Spezialschaltungen, die nicht irgendwie programmierbar bzw. durch Programmierung steuerbar (parametrisierbar) sind.
- Es geh also in der Praxis nicht um den extremen Gegensatz: fertige Systemumgebung vs. Einzweck-Spezialschaltung - sondern um kostenoptimierte Verbundlösungen.

    - Ausnutzung zuhandener Prozessoren - Auswahl, Gestaltung der Systemumgebung, Zusatzbeschaltung, Implementierung vorteilhafter Programmier-Paradigmata (Multitasking, Ereignissteuerung, State Machine usw.), Mehrprozessorkonfigurationen.
    - Gestaltung neuartiger Prozessoren.
    - programmierbare, funktionsvariable Hardware.

Das Entscheidungsproblem (Hardware vs. Software) stellt sich in der Praxis:

- zur Erfüllung extremer Relazeitanforderungen,
- zur Erfüllung gegebener Anforderungen auf kostengünstigere Weise (Funktionalität implementieren, die bisher im gegebenen Kostenrahmen nicht implementierbar war),
- zum “entwicklungsmäßig schneller sein” (Time to Market).

Weshalb ersetzen wir Hardware durch Software?

- der herkömmliche Trend seit den 60er Jahren,
- Rückgriff auf ein höheres Niveau der Zuhandenheit,
- Zuhandenheit hochleistungsfähiger Technologien (Taktfrequenzen),
- Verkürzung der Entwicklungszeit ("es ist nur zu programmieren" = Time to Market),
- Verringerung der Hardwarekosten (Controller usw.),
- Beherrschung von Kompliziertheit (durch Zerlegen in handliche Programmstücke),
- Ersparen der Hardware-Entwicklung (zuhandene funktionstüchtige Plattformen),
- kurze Änderungszyklen,
- flexible Entwicklungswerkzeuge,
- funktionelle Flexibilität allgemein (man darf programmieren - es läßt sich alles programmieren),
- Software kann nicht ausfallen (verschleißen) -- wohl aber Fehler enthalten (und auch veralten -- wenn es keine HardwarePlatform mehr gibt, auf der sie laufen könnte).

Weshalb ersetzen wir Software durch Hardware?

- ein (wieder mal) neuerer Trend,
- ermöglicht durch hochintegrierte programmierbare Schaltkreise (FPGAs, CPLDs),
- mehr Leistung,
- Kontrolle über den gesamten Lebenszyklus (nicht mehr auf Zulieferungen, fremde Standards und frende Schutzrechte angewiesen sein),
- Kostenoptimierung = Verbilligung der Hardware-Plattform, kein Software-Overhead (besserer Wirkungsgrad),
- Lösung der Aufgabe mit weniger Silizium, Strom, EMV (langsamst-mögliche Taktierung),
- besseres Realzeitverhalten,
- kürzere Latenzzeiten,
- echter Parallelismus (von vornherein = gemäß Struktur des Problems - der dem Problem überhaupt inhärente Parallelismus könnte voll ausgenutzt werden),
- unmittelbare (evidente) Vergegenständlichung der Informationswandlungen (kein Zwang zum Programmieren (= unangemessen, unübersichtlich, "semantische Lücke" zwischen Problembeschreibung und Programmablauf, zuviel Overhead).

Stoff im 1. Semester: *was* ist zu entwerfen?

Klassische Digitaltechnik

- Intuitives und systematisches Entwerfen am Beispiel (SERDES)
- Register-Tansfer-Entwuf und State Machines
- Funktionprinzipien programmgesteuerter Automaten

Problemlösung mit Mikrocontrollern

- Grundlagen der Anwendungstechnik. Mikrocontroller-Familien
- Grundlagen der Schaltungstechnik
- Speichersubsysteme
- E-A-Subsysteme
- Grundlagen der Programmorganisation. Elementare Programmiertechniken
- Beschleunigungsvorkehrungen

Stoff im 2. Semester: *wie* ist zu entwerfen?

Wird vor allem anhand von Beispielen geübt.

- Problemlösung mit Mikrocontrollern
- Problemlösung mit programmmierbaren Schaltkreisen
- Die Hardware-Beschreibungssprache VHDL (Einführung)

Mikrocontroller + Hardware

Schwerpunkte

- Takt
- Rücksetzen

Jeweils gemäß Datenblatt/Applikationshandbuch vorgehen. Achtung: es kann sein, daß andere Schaltungsteile kritischere Anforderungen (Takttoleranzen, Dauer des Rücksetzens und Rücksetz-Schwelwerte (V_{CC}) haben!

E-A-Ankopplung

- Strombelastung
- kapazitive Last
- Derating beachten
- Treibfähigkeit beachten

Richtungsssteuerung
Vorsicht vor Buskonflikten beim Einschalten und im Betrieb!

Hochohmige (floating) Busleitungen
Vermeiden. (Pull-up-Widerstände, Bushalteschaltungen, Park-Hardware)

Port-Belegung

- Im Problemfall Ports stets für die höher integrierte Funktion im µC ausnutzen,dafür die einfachen I/O-Funktionen auslagern.

ungenutzte Ports:
- als Ausgänge und mit Festwert belegen (nur, wenn garantiert unbeschaltet)
- als Eingänge mit externem Pull-up
- als Eingänge und internen Pullup scharf machen (oder lassen)
- NIE als wirklich offene Eingänge (falls nicht ausdrücklich im Datenblatt spezifiziert)

Bus-Erweiterung mit Pufferstufen und Registern

- Buskoppelstufen und Register
- Abfrage: Multiplexer
- Universal-Busschaltkreise ausnutzen
- Boundary Scan
- einfache Schiebe-Interfaces

Auf *asynchrone Eingänge* achten (Metastabilität). Ggf. über D-FF-Register synchronisieren.

Taktierung bzw. Output Enable von außen - nur selten vorhanden (“echte” Tristate-Ports, um µC seinerseits als Slave an einen Bus zu hängen)

Bei Anschaltung von Bustreibern *Buskonflikte* vermeiden (Richtungssteuerung).

Speicheranschaltung

dafür vorgesehene Prozessoren

- ausgebauter Speicherbus (H8/300H)
- Multiplex-Speicherbus (PIC, 8051 usw.)

sonst: Ports für Anschaltung ausnutzen. Es gibt auch spezielle kleine RAMs (Siemens).

Adreßdecodierung
Alias-Adressen: nie für Programmiertricks ausnutzen!

Adreßerweiterung
Bankregister (oder Segment-RAM) und Bank-Organisation. Heutzutage: einfachste Hardwarelösung.

Externe Beschleunigungsmaßnahmen

- Adresse für Ausgabezwecke ausnutzen
- Parallelisierung von Speicher- und Interfacezugriffen
- Speicher breiter gestalten, als Art Mikroprogrammspeicher betrieben (µC übernimmt Sequencing)
- Aufschalten von Daten auf Teile des Datenbus
- externe Steuer- bzw. Zwischenspeicher (adressierbar oder FIFO)

Einfache Multiprozessorkopplungen

- Direktverbindungen
- Dual-Port-RAMs
- einfache Bussysteme
- Schiebering-Strukturen