Synthese Eingebetteter Systeme

Wintersemester 2012/13

1 – Einführung: Entwurf und Synthese

Michael Engel
Informatik 12
TU Dortmund

2012/10/17

Überblick

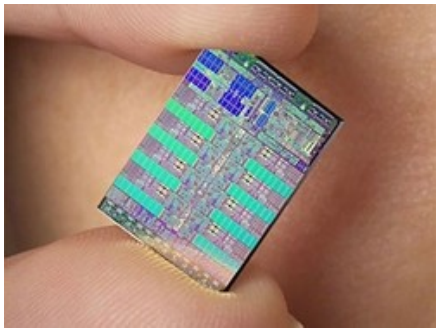
- Komplexität aktueller eingebetteter Systeme
- Electronic System Level Modeling (ESL)
- Beschreibungsebenen für Eingebettete Systeme
  - Entwurfsfluss eingebetteter Systeme
  - Vom Transistor zum Verhalten
  - Vor- und Nachteile verschiedener Abstraktionsebenen
- Synthese
  - Begriffsdefinition
  - High-Level-Synthese
  - Entwurfsfluss eingebetteter Systeme
  - Beispiele

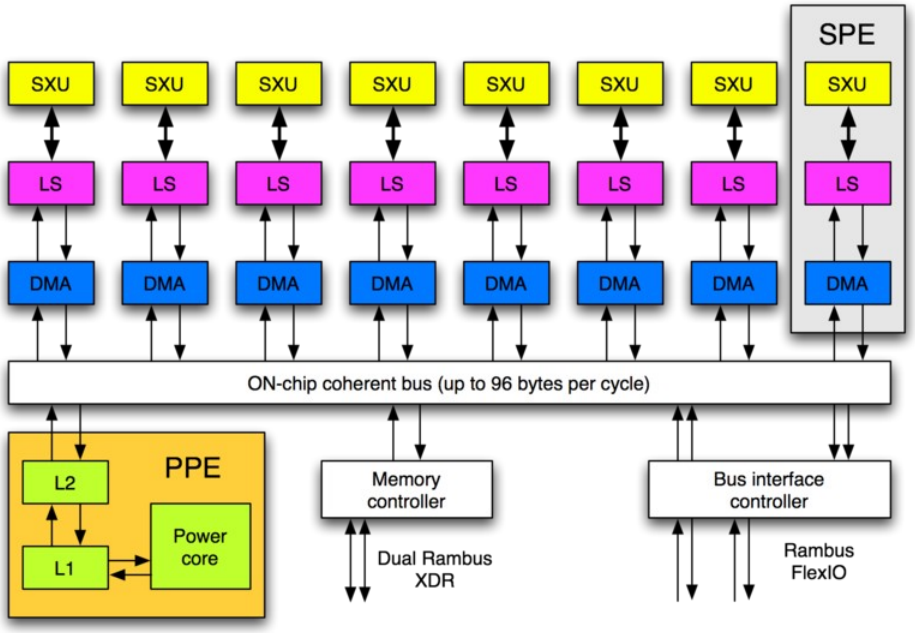
Komplexität Eingebetteter Systeme

- Systems-on-Chip (SoC)
  - Basis aktueller eingebetteter Systeme
  - Integration kompletter Systeme auf einem Baustein
- Heute: Multiprozessor SoCs (MPSoCs)
  - Bis zu 188 Prozessor-Cores auf einem Chip
    - TILERA/Cisco METRO-Chip
  - Zusätzlich: Peripherie, Busse, Speicher
  - Selbst einfache Systeme sind oft MPSoCs
- Steigende Komplexität von SoCs
  - Längere Entwurfsdauer, aber kürzere Produktzyklen
  - Steigende Kosten der Entwicklung

MPSoC-Beispiel: CELL

- IBM/Toshiba/Sony CELL-Prozessor
  - Entwicklungsstart: März 2001
  - Entwicklungsdauer: ca. 4 Jahre
  - > 400 beteiligte Entwickler
  - Entwicklungskosten:
    400 Millionen US$





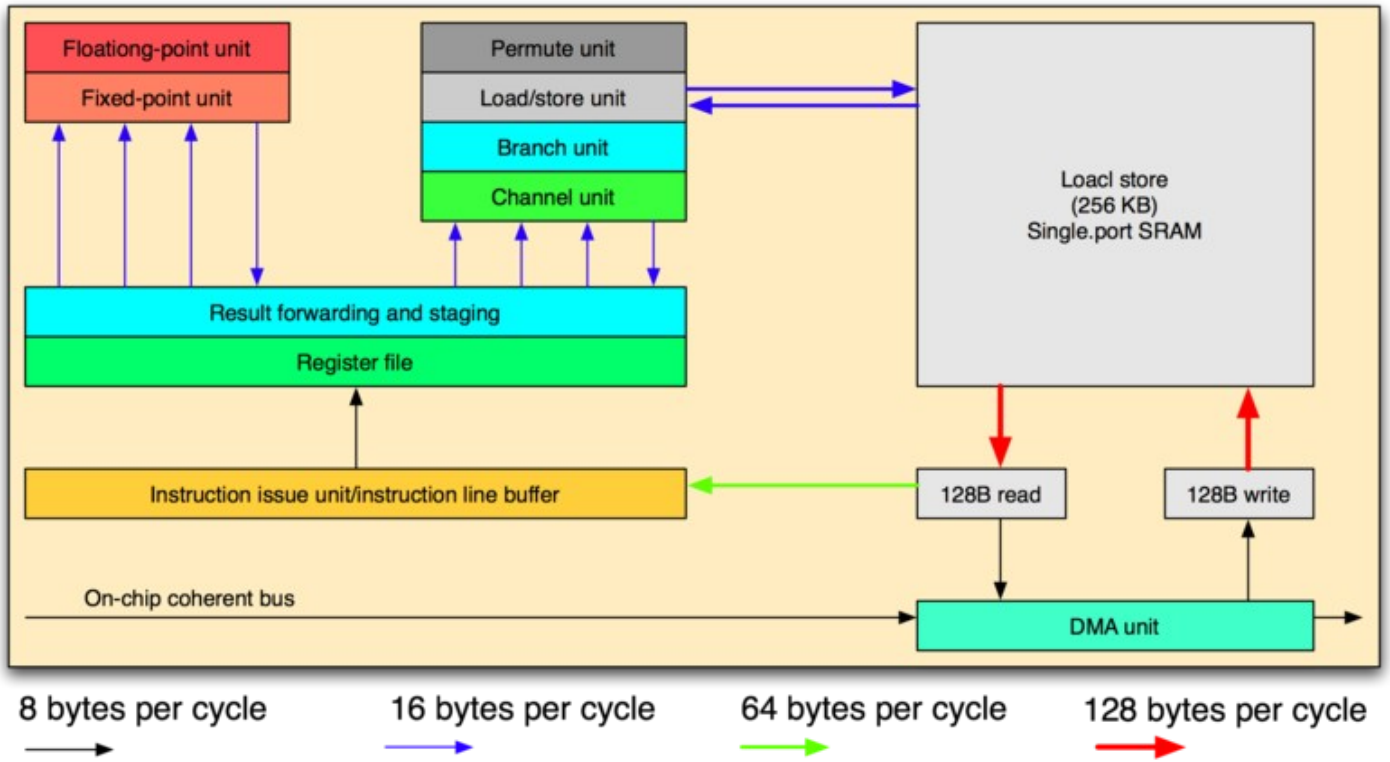
CELL: Details

Technische Daten des CELL:

- **241 Millionen Transistoren** auf 235mm^2 Chipfläche (90nm), 3.2 GHz
- Acht Synergistic Processing Elements (SPE)
  - Je eine Recheneinheit (ALU) mit vierfachem SIMD
  - 128 Register, die jeweils 128 Bit groß sind
  - Memory Flow Controller (MFC): DMA-Transfers
  - Lokaler Speicher von 256 kB
  - Je **21 Millionen Transistoren**, davon ~2/3 für RAM
- Ein PowerPC Processing Element (PPE)
  - 64-Bit-PowerPC-Architektur von IBM
  - In-Order-Ausführung, zwei Threads gleichzeitig
  - 512 kB L2-Cache
- Element Interconnect Bus (EIB)
  - Ringstruktur
  - Bis zu 96 Byte pro Takt übertragbar

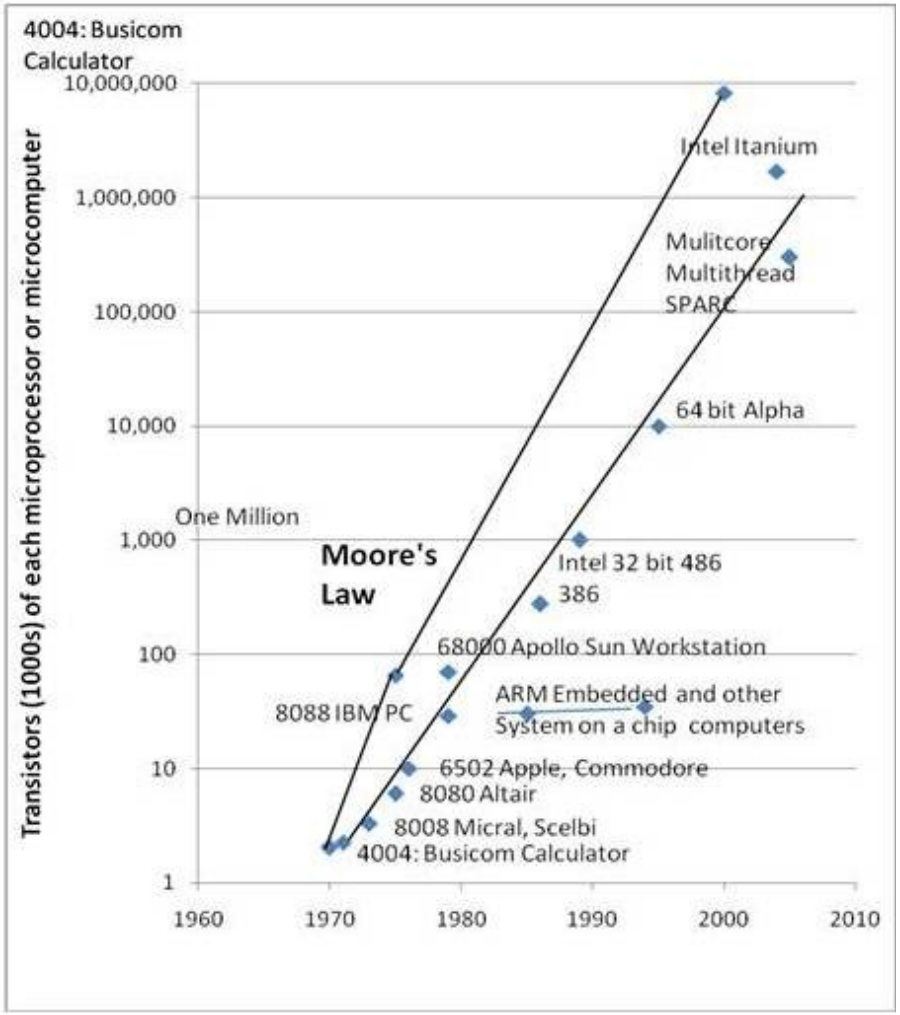
CELL: Details (2)

- Struktur der CELL SPUs:



Entwicklung der Komplexität

- Komplexität von Chips, Systemen und Software steigt immens
  - Moore‘s Law: doppelte Anzahl Transistoren pro Chip alle 18 Monate
- Anzahl Chips/System sinkt
  - Aber: Anzahl benötigter Verbindungen steigt!
  - 6502: 40 Pins (1976)
  - Intel i7: 1366 Pins (2011)
- Softwarekomplexität
  - Apple][: 20 kB ROM
  - Windows 7: ... ☺



http://www.ieee.org/portal/cms_docs_sscs/sscs/08Fall/BellFig6.JPG

Electronic System Level Modeling (ESL)

- **Neue, effizientere Entwurfsmethoden erforderlich**
- ESL: “Concurrent design of hardware and software”
- Zwei Entwurfsebenen:
- *Verhaltensebene*
  - ESL-Entwurf *vor* Hardware/Software-Partitionierung
- *Architekturebene*
  - ESL-Entwurf *nach* Hardware/Software-Partitionierung
  - Architekturentwurf
    - Entwurf in einer ESL-Sprache
    - Verwendet **Synthese**technologie, um RTL-Beschreibung zu erzeugen
  - Plattform-basierter Entwurf
    - Entwurf basierend auf ESL-Modell, verwendet existierende Plattform und **Abbildung**stechniken zur Erzeugung einer RTL-Beschreibung
  - Softwarespezifikation, -entwurf und Implementierung

Synthese

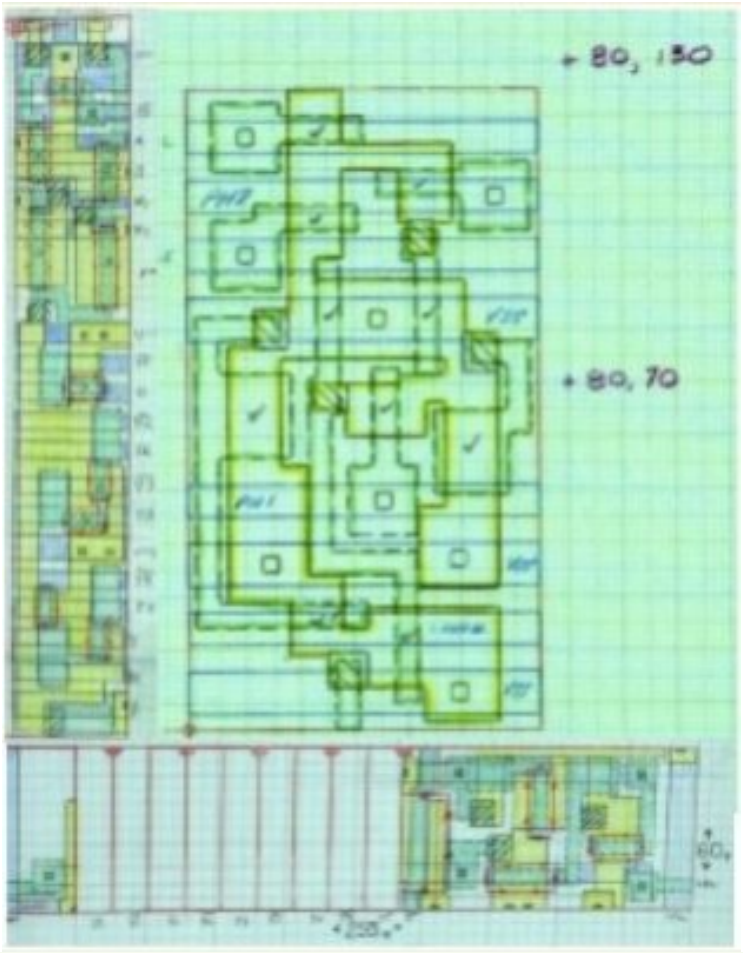
- Definitionen [2]:
  - **„Design synthesis can be defined as the transformation of a design to a level of lower abstraction.“**
- Präzisierung:
  - „Transformation of a design from a point in the functional domain to one in the structural domain“
- **High-Level-Synthese** (HLS)
- **Logiksynthese**
  - „maps RTL-descriptions onto a specific structure suitable for the target architecture: **low-level synthesis**“

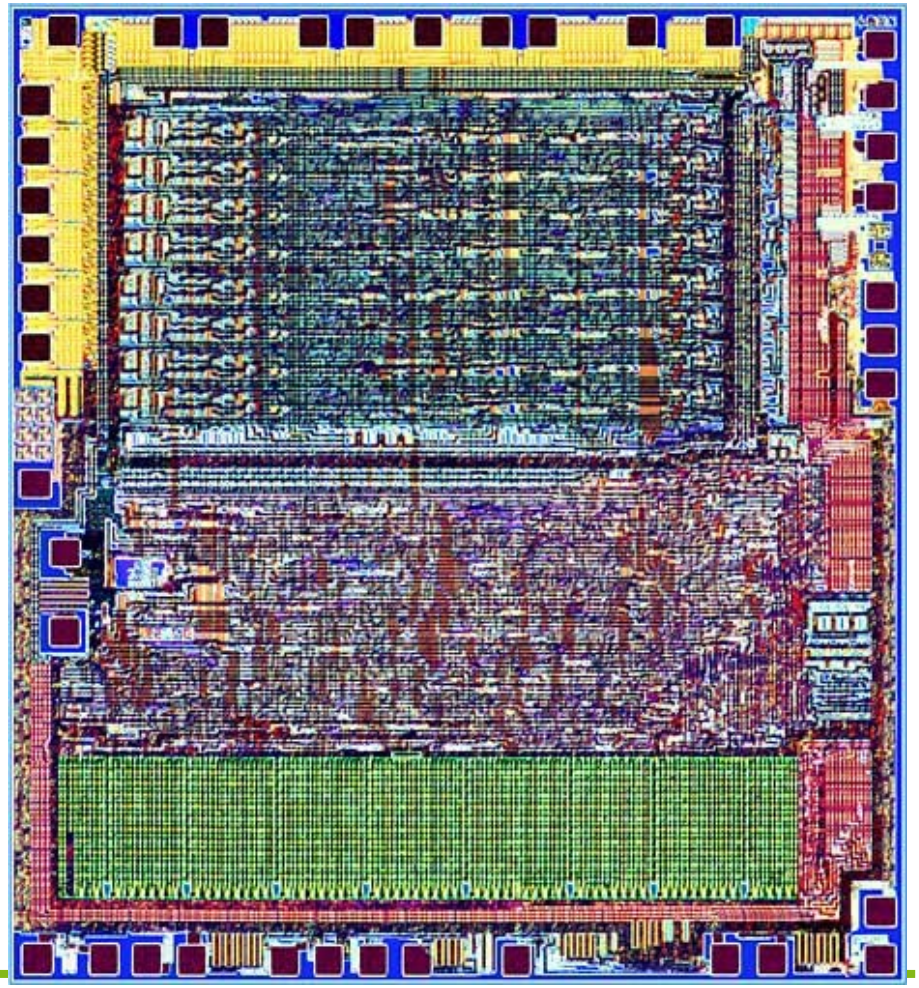
Low-Level-Synthese: Historie

- Anfänge der Mikroprozessoren in den 60er Jahren: Fairchild, TI
- Bis Ende der 70er Jahre: kaum Software-Unterstützung
  - Layouts für Chips auf Transistorebene manuell
    - Komplexität: einige 1000 Transistoren
    - MOS6502 CPU: 3520 Tr. (1976)
  - Leiterplattenlayout für Systeme manuell
    - Komplexität: 10–100 ICs
    - z.B. Apple][: ca. 100 ICs (1977)
- Erste CAD-Werkzeuge
  - Leiterplattenlayout
- Hardwarebeschreibungssprachen
  - MIMOLA (DO), Karl–3 (KL)

Physikalisches Layout: Transistor-Layout-Ebene

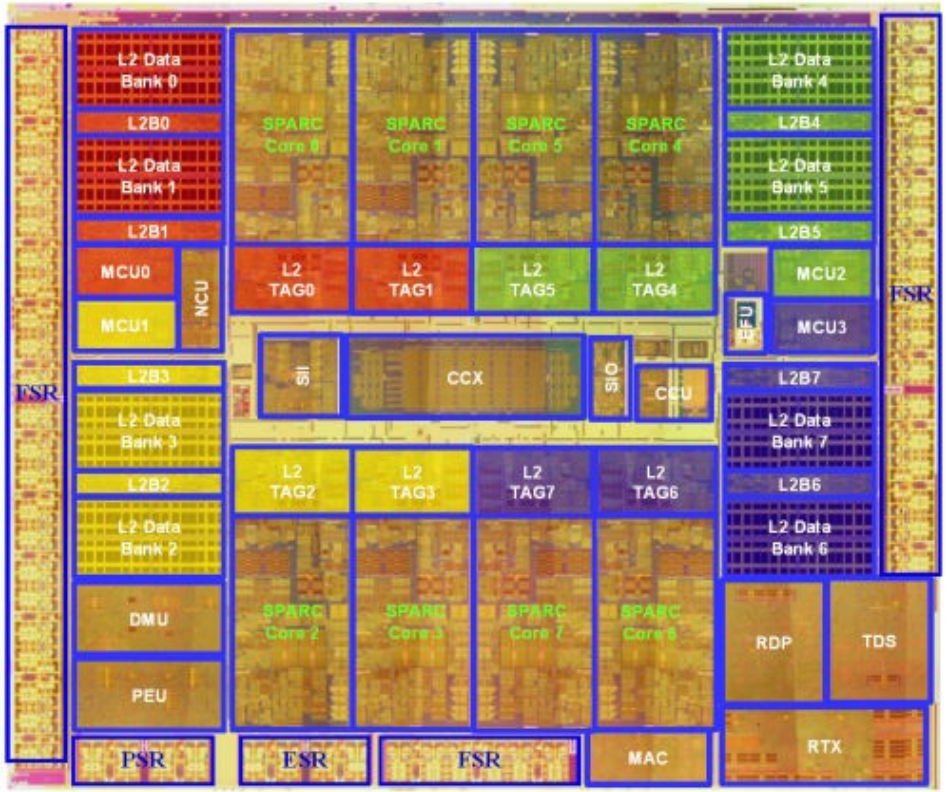
- Geometrische Anordnung der Transistoren auf einem Chip





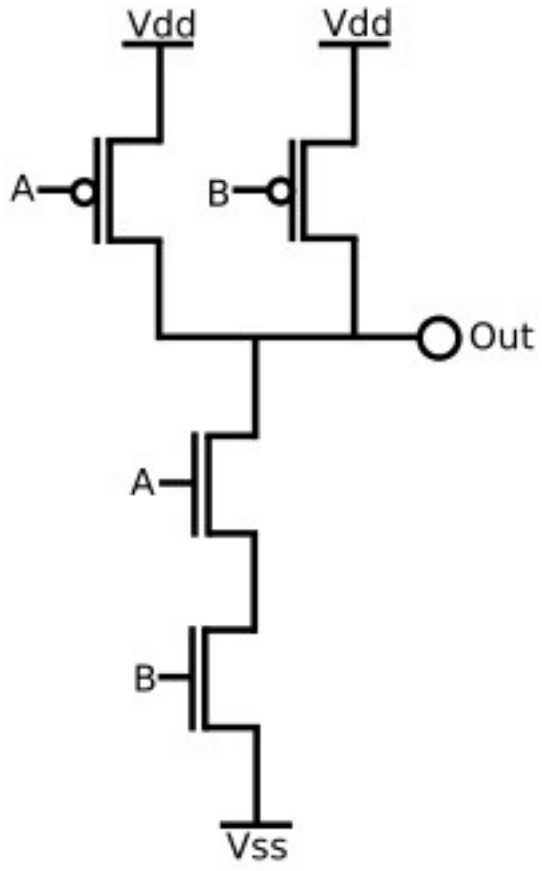
Physikalisches Layout: Höhere Ebenen

- Cell, Module, Floorplans, Physical Partitions

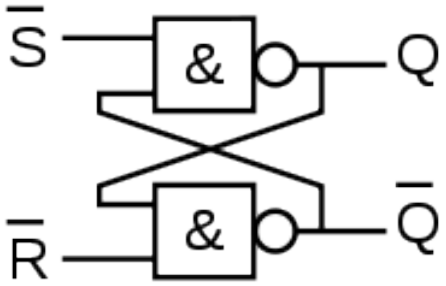


Strukturelles Layout: Transistoren und Gatter

- Verschaltung einzelner Transistoren zu Logikeinheiten

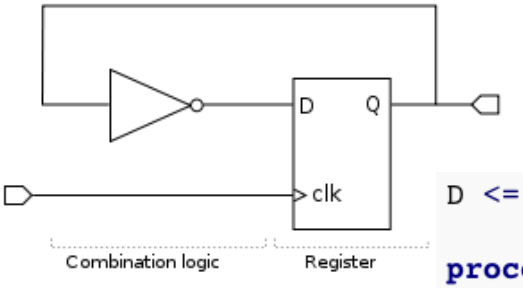


- Verschaltung einzelner Gatter und Flip-Flops zu Automaten und Logik



Registertransfer, Strukturelles Layout

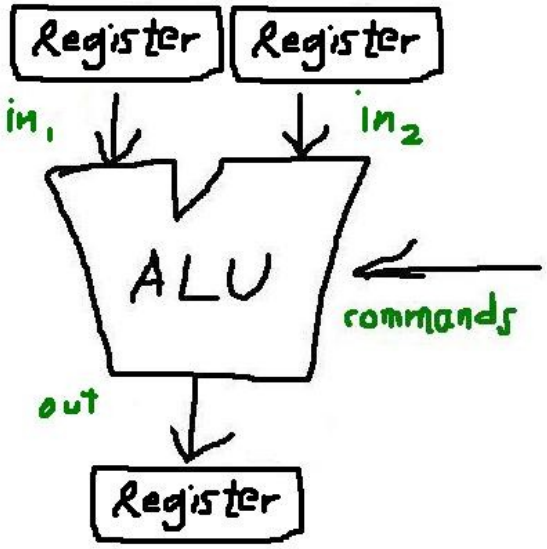
- Registertransfer
  - Synchrone Schaltungen (Schaltwerke)
  - Kombinatorische Logik + Register
  - Hardwarebeschreibungs-sprache: VHDL, Verilog




```
D <= not Q;

process(clk)
begin
    if rising_edge(clk) then
        Q <= D;
    end if;
end process;
```

- Komplexe Komponenten
  - z.B. ALUs, Speicher



Beschreibungsabstraktionen

- Von den Anforderungen zum Chip
- Gajskis Y-Chart

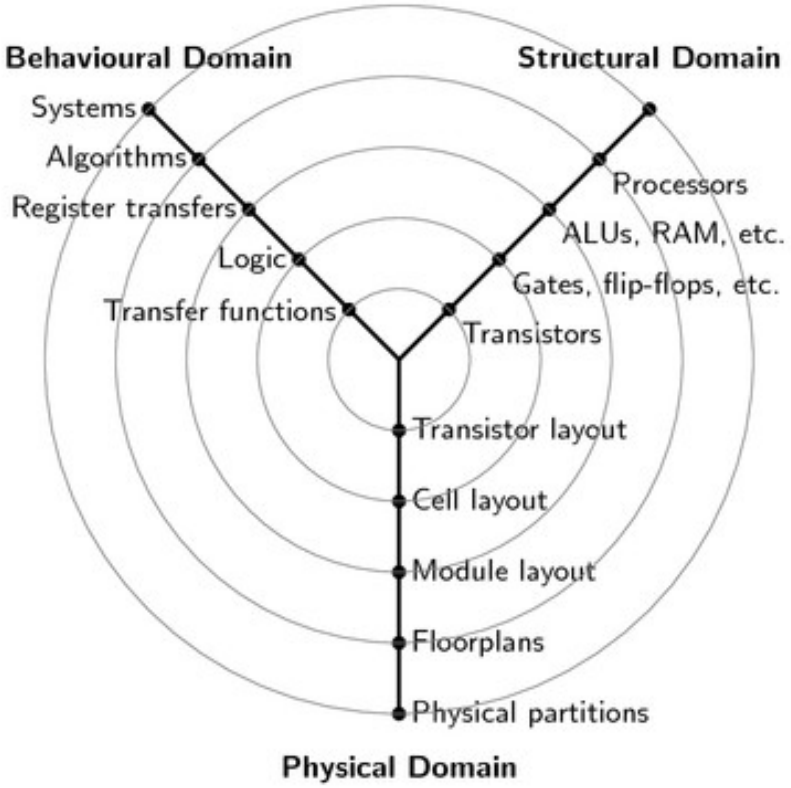


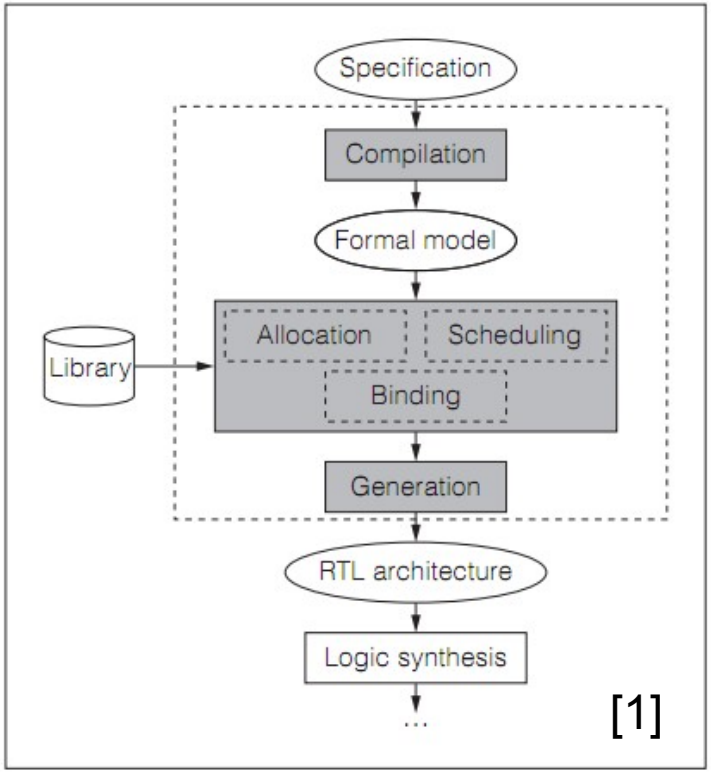
Figure 1: Gajski-Kuhn Y-chart

High-Level-Synthese

- Herkömmliche Entwurfsmethoden
  - Zu aufwendig, langwierig und unflexibel
  - Schlechte Wiederverwendbarkeit
- Lösung: High-Level-Synthese (HLS) [1]
  - „High-level Synthesis, also known as architectural or behavioural synthesis, allows a design to be specified functionally at the **level of algorithms** and then be converted using a set of constraints into the RTL's structural connection of functional units and registers. During this process both the **temporal** and **spatial** locations of each function must be determined, i.e. during which clock cycles will a function occur and what physical functional unit will it be bound to."
- Temporale Zuordnung: *Scheduling*
- Räumliche Zuordnung: *Mapping*

High-Level-Synthese: Entwurfsfluss

- Vorhanden:
  - High-Level-Beschreibung der Anwendung (C/C++/SystemC)
  - Bibliothek von RTL-Komponenten
  - Entwurfsconstraints
- HLS-Syntheseschritte:
  - Übersetzung der Spezifikation
  - Allokation von HW-Ressourcen
    - Funktionseinheiten, Speicher, Busse...
  - Scheduling von Operationen
  - Zuordnung von Operationen zu Funktionseinheiten
  - Zuordnung von Variablen zu Speichern
  - Zuordnung von Transfers zu Bussen
  - Erzeugung der Beschreibung der RTL-Architektur

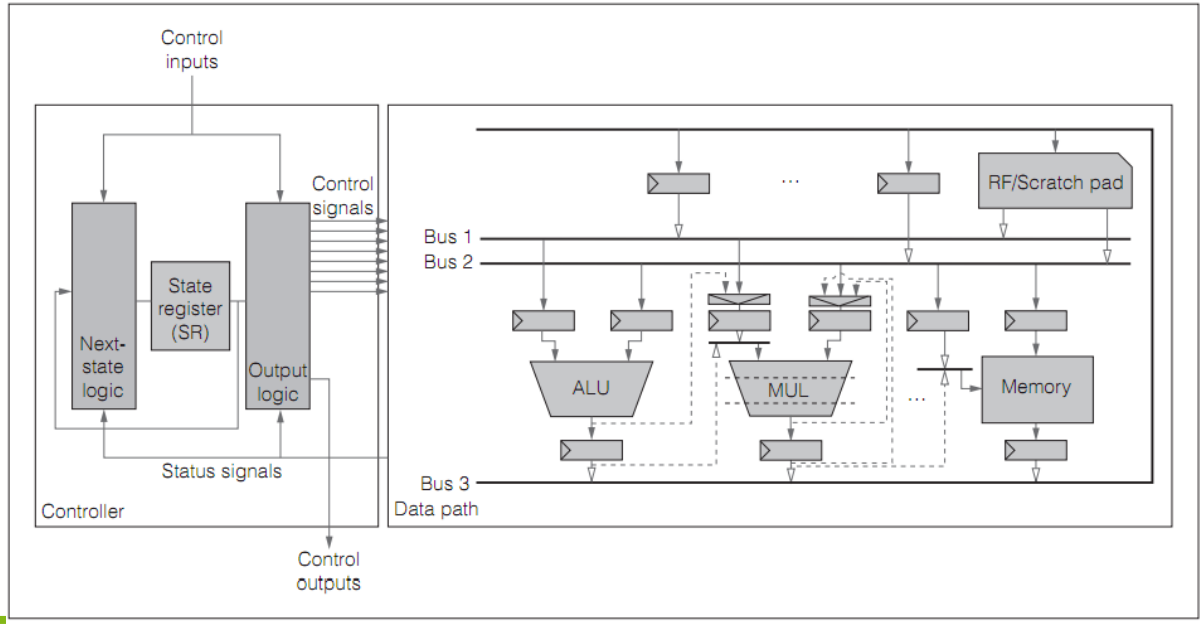


[1]

Ausführungsplattform: RTL-Ebene

RTL-Architektur: Registertransfer-Komponenten

- Controller
  - Endlicher Automat (FSM), steuert Datenpfad durch Signale
- Datenpfad
  - Speicherelemente (Register, Registerbänke, Speicher
  - Funktionale Einheiten (ALUs, Multiplizierer, Schieberegister)
  - Verbindungen
    - Busse
    - Multiplexer
  - Mehrere Instanzen
    - Verbindung: Busse



Unterschiede in RTL-Beschreibungen

RTL-Ausgabemodell der HLS abhängig von Zuordnungsentscheidungen

- Unterschiedliche Detaillierung der erzeugten RTL-Beschreibung
- Beispiel:
  - Übersetzung der High-Level-Instruktion
    a = b + c
    ausgeführt im Zustand *n*
  - Unterschiedliche Detaillierung der Zuordnung von Komponenten
  - Fehlende Zuordnungen müssen von der Logik-(low-level-)Synthese übernommen werden
    - Ebenso die zugehörige Optimierung

Without any binding:

```
state (n): a = b + c;
go to state (n + 1);
```

With storage binding:

```
state (n): RF(1) = RF(3) + RF(4);
go to state (n + 1);
```

With functional-unit binding:

```
state (n): a = ALU1 (+, b, c);
go to state (n + 1);
```

With storage and functional-unit binding:

```
state (n): RF(1)=ALU1 (+, RF(3), RF(4));
go to state (n + 1);
```

With storage, functional-unit, and connectivity binding:

```
state (n): Bus1 = RF(3); Bus2 = RF(4);
Bus3 = ALU1 (+, Bus1, Bus2);
RF(1) = Bus3;
go to state (n + 1);
```

Vorteile der High-Level-Synthese

- Kompakte Beschreibung auf abstrakter Ebene
- Schnellere Entwicklungszeit
  - Erleichtert *Prototyping* und *Design Space Exploration*
- Höhere Abstraktionsebene für Hardwarebeschreibung
  - Ermöglicht Architekturentscheidungen auf Systemebene
  - Wiederverwendung für verschiedene Technologien

"What are the 2 biggest reasons to use High Level Synthesis?"

Faster time to RTL :	######################## 64%
Faster verification time :	################## 49%
Fewer engineering resources :	########### 31%
Fewer Bugs :	####### 19%
RTL better than hand-coded :	##### 14%
Faster ECO implementation :	### 8%
Better product differentiation :	### 7%
Other :	## 4%

[3]

High-Level-Synthese: Erfahrungswerte

- Ist HLS heute schon sinnvoll einsetzbar?
  - Vergleich HDL–SystemC

Block A: AutoESL results vs. hand-coded (both results after running through Synopsys Design Compiler + Talus P&R)

	AutoPilot 1st result	AutoPilot final result
Area	2% larger	1% smaller
Power	2% higher power	12% lower
Latency	40% higher latency	6% lower

Original design:	800 lines of ANSI C code
Hand-coded design:	4000 lines of VHDL RTL code
AutoESL-generated design:	100,000 lines of Verilog RTL code

[4]

Lines of C code:	20
Lines of Catapult output VHDL:	450
Lines of manual VHDL:	60

	Area (k GE)	Power (mW)	Throughput rate (Mbps)
Hand coded VHDL	13	32	100
Catapult C	12.8	31	100

[5]

Zusammenfassung

- Komplexere elektronische Systeme
  - ...erfordern Entwurf auf Systemebene
  - ...benötigen neue Entwurfsmethoden und –werkzeuge
  - ...profitieren von höheren Abstraktionsebenen
- Vorlesung SeS: zwei wichtige Entwurfsansätze
  - High-Level-Synthese: SystemC
  - Wiederverwendung von IP: Mapping auf Multicores
- Weitere Themen
  - Exkurs: „Traditionelle“ Hardwarebeschreibungssprachen
  - FPGAs: Strukturen und Funktionsweise
  - Prozessorbeschreibungssprachen: ArchC

Literatur

1. Philippe Coussy, Daniel Gajski, Michael Meredith, and Andres Takach.
   An Introduction to High-Level Synthesis
   IEEE Journal on Design and Test, Vol. 26 Issue 4, July 2009
   http://portal.acm.org/citation.cfm?id=1608648
2. Brian Bailey and Grant Martin.
   ESL Models and their Application: Electronic System Level Design and Verification in Practice
   Springer-Verlag, 1st edition, 2009
3. Umfrage unter Kunden von Mentor Graphics:
   http://www.deepchip.com/items/0479-04.html
4. http://www.deepchip.com/items/0485-04.html
5. http://www.deepchip.com/items/0485-08.html