

Architekturen und Entwurf von Rechnersystemen

Wintersemester 2017/2018



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Teil 0: Einleitung

Teil 2: Trends, Techniken und Werkzeuge





Organisatorisches

Dozent

- ▶ Andreas Koch
- ▶ Sprechzeiten: In der Regel Mi 11:00-11:30 Uhr, E101
- ▶ koch@esa.cs.tu-darmstadt.de

Wissenschaftlicher Mitarbeiter

- ▶ Jaco Hofmann
- ▶ Sprechzeiten: In der Regel Mi 14-15 Uhr, E106
- ▶ jah@esa.cs.tu-darmstadt.de

- ▶ Fabian Czappa
- ▶ Sprechstunde in der Regel
 - ▶ Montag 13:30 - 15:10
 - ▶ Donnerstag 9:50 - 11:30
- ▶ Raum E104

<http://www.esa.cs.tu-darmstadt.de>, Abschnitt “Lehre”

- ▶ Folien
- ▶ Vorlesungsaufzeichnungen
- ▶ Übungsaufgaben
- ▶ Ggf. zusätzliches Lehrmaterial

- ▶ Besteht im wesentlichen aus vier Teilen
 - ▶ 1. Teil: Einführung in Hardware-Beschreibungssprache Bluespec
 - ▶ 2. Teil: Trends, Techniken und Werkzeuge des Hardware-Entwurfs
 - ▶ 3. Teil: Rekonfigurierbare System-on-Chips am Beispiel des Xilinx Zynq 7000
 - ▶ 4. Teil: Entwurfsverfahren und Werkzeuge für Hardware-Rechenbeschleuniger
- ▶ DT/RO (inkl. Verilog!) ist **Voraussetzung** für AER
- ▶ Falls Kenntnisse fehlen bzw. vertieft werden sollen
 - ▶ Grundlagen Hardware-Design: DT/RO-Aufzeichnungen und Buch
 - ▶ Verilog: TGDI-Aufzeichnungen und Buch, **alte** CMS 2012-Aufzeichnungen



Grundlagen RO/DT + Reststoff

D.M. Harris und S.L. Harris: *Digital Design and Computer Architecture*, 2. Auflage, MKP, 2012

Erster Teil der Vorlesung

- ▶ Bluespec Online-Dokumentation der RBG unter `/usr/local/bluespec/doc/BSV`
- ▶ Insbesondere Einführungsbuch *Bluespec by Example*

Dritter Teil der Vorlesung

- ▶ L. Crockett, R. Elliot, Martin Enderwitz, Bob Stewart und D. Northcote: *The Zynq Book*, Strathclyde Academic Media, 2014
 - ▶ Frei verfügbar auf <http://www.zynqbook.com/>

Form: Integrierte Veranstaltung

- ▶ Dienstags 16:15 - 17:55 Uhr, S311/08
- ▶ Donnerstags 16:15 - 17:55 Uhr, S101/A01
- ▶ In der Regel wird die Vorlesung aufgezeichnet werden

Aufteilung

- ▶ Anfangs zwei Vorlesungen pro Woche (zur Bluespec-Einführung)
- ▶ Alle zwei Wochen ab dem 26.10.2017 Übungsvorstellung und Erläuterung
- ▶ Nach der Bluespec-Einführung wöchentliche Vorlesungen

Übungsaufgaben

- ▶ Freiwillige Aufgaben (auch keine Bonuspunkte!)
- ▶ Hilfe bei der Bearbeitung durch den Tutoren/Moodle
- ▶ Werden in Tafelübungen besprochen



- ▶ Klausur
- ▶ Derzeit geplant am 19.02.2018, 13:00-14:30
- ▶ 90 Minuten reine Schreibzeit

Verilog-Simulator und Signalverlaufs-Anzeige

- ▶ Open Source: Icarus Verilog `iverilog` und `gtkwave`
 - ▶ Auch in RBG Pool installiert
- ▶ Kostenlos, z.B.
 - ▶ Xilinx WebPack <http://www.xilinx.com>
 - ▶ Altera Quartus Web Edition <http://www.altera.com>

Bluespec-Compiler und Simulator

- ▶ Kommerzielles Werkzeug, **nicht** frei verfügbar
 - ▶ Auch keine "Studentenversion"
- ▶ 200 Lizenzen im RBG Pool installiert
 - ▶ Über SSH auch von außen zugänglich
 - ▶ Auch (bedingt) graphische Oberfläche
 - ▶ Konsolenbedienung wird aber für AER häufig ausreichend sein



Vorlesung benutzt das aus RO/DT bekannte interaktive Quizsystem

- ▶ Antworten via Browser
- ▶ Smartphone, Laptop, Tablet verbunden mit [eduroam](#) (WLAN der TU)

Test des Quizsystems . . .



Einführung in Bluespec



Bluespec hat einen eigenen Foliensatz!

Danach hier weiter zu

Trends, Techniken und Werkzeuge des Hardware-Entwurfs



Entwicklung der Mikroelektronik

Am Anfang war das Relais und die Röhre...

- ▶ 1947/48 Erfindung des Bipolartransistors durch Shockley, Bardeen, Braitain
- ▶ ab 1950 Grundlagenforschung auf dem Gebiet der Bipolarschaltungen
- ▶ ca. 1960 Entwicklung der Silizium-Planartechnik und erste *integrierte Schaltungen* mit ca. 10 Bauelementen
- ▶ ca. 1960 Konzeption des MOS-Transistors durch Khang und Atalla
- ▶ ca. 1970 erster Mikroprozessor
- ▶ und was waren bzw. sind die *Vorteile der integrierten Schaltungen*
 - ▶ Transistoren, Dioden und Widerstände sowie die Verbindung der Bausteine untereinander werden in einem gemeinsamen Herstellungsprozess in einem Silizium-Einkristall integriert.
 - ▶ Höhere Zuverlässigkeit, höhere Schaltgeschwindigkeiten, höhere Packungsdichte.

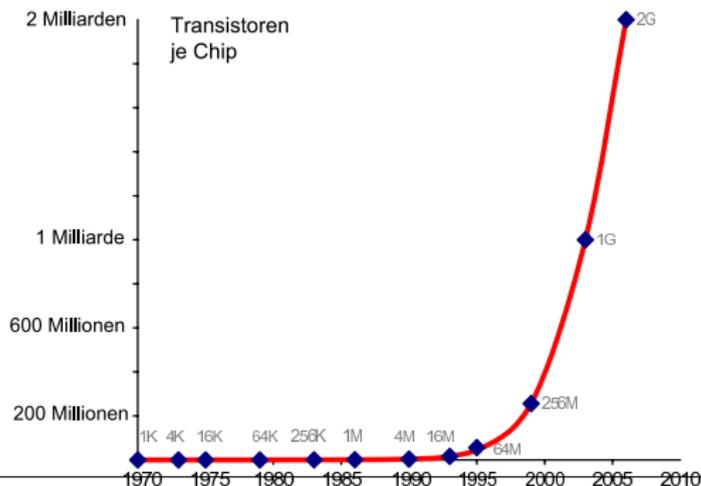
- ▶ Mikroelektronik umfaßt viele Gebiete:
 - ▶ Elektrotechnik: Transistorschaltungen/Differentialgleichungen
 - ▶ Physik, Verfahrenstechnik/Chemie
- ▶ Warum Mikroelektronik in der Informatik?
 - ▶ Grundverständnis nützlich
 - ▶ Einfluss auf Architekturen von Rechnersystemen
 - ▶ Zur Entwicklung von Mikroelektronik wird Software verwendet.
 - ▶ Die Anforderungen an den Entwurf werden mit Algorithmen gelöst.
 - ▶ Logikminimierung
 - ▶ Platzierung von Komponenten
 - ▶ Automatische Hardware-Erzeugung durch spezielle Compiler
 - ▶ ...

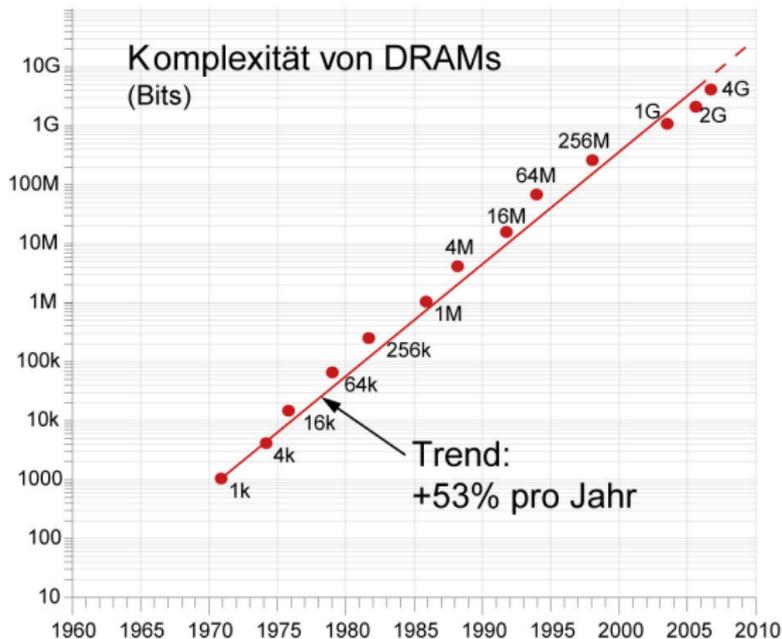
- ▶ Integrierte Schaltungen (ICs), oft auch nur als **Chips** bezeichnet.
- ▶ Chipmarkt: in 2004 Umsatz von 213 Milliarden USD weltweit
für 2015 Umsatz von 332 Milliarden USD
- ▶ Nahezu überall verbaut
 - ▶ Offensichtlich: Rechner
 - ▶ PC, Server, Supercomputer, ...
 - ▶ Versteckt: eingebettete Systeme
 - ▶ Autos, Fernseher, Herzschrittmacher, ...
- ▶ Alleine in Deutschland: 11,7 Milliarden EUR Chip-Umsatz (2015)
 - ▶ Bildet aber Basis für 50x größeren Markt
 - ▶ 3 Millionen Arbeitsplätze
- ▶ Sollte man sich auch als Informatiker genauer anschauen!
- ▶ Ursprung dieser Bedeutung?

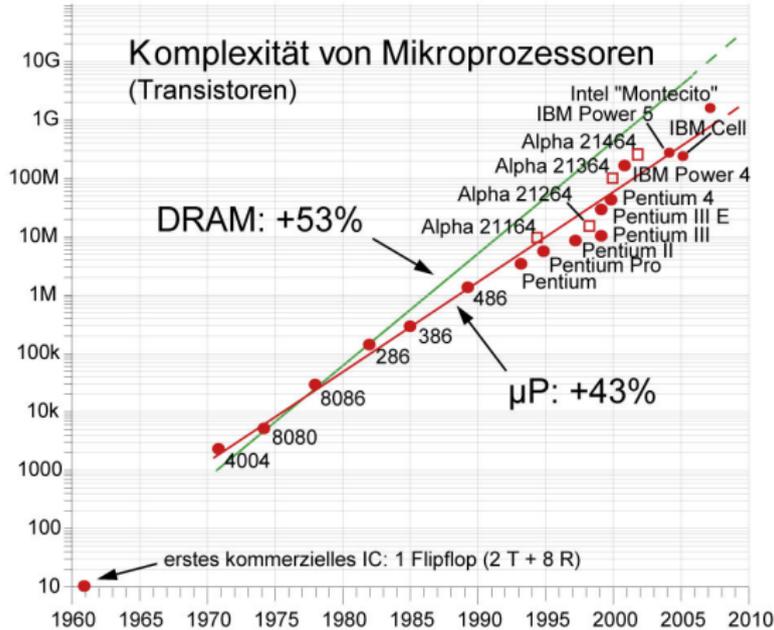
Moore's Gesetz - Exponentielles Wachstum

Alle 18-24 Monate verdoppelt sich die Anzahl der wirtschaftlichsten Transistoren auf einem Chip.

Rekordhalter 2016: Altera Stratix 10 FPGA mit 30 Milliarden Transistoren

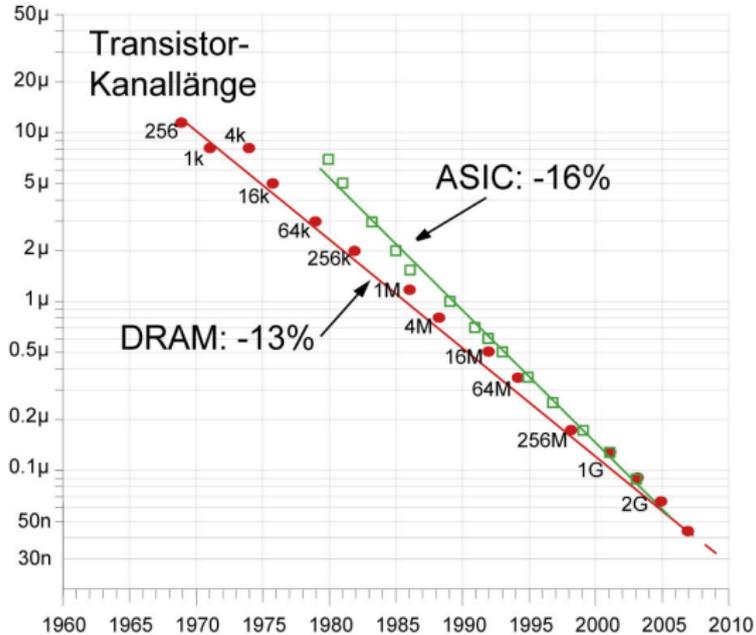






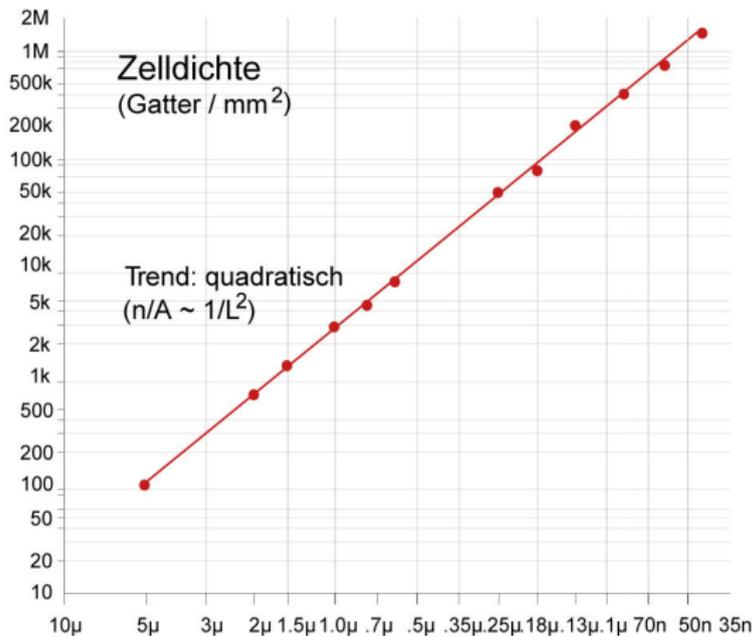
Verbesserung der Fertigungsprozesse

Transistor-Kanallängen



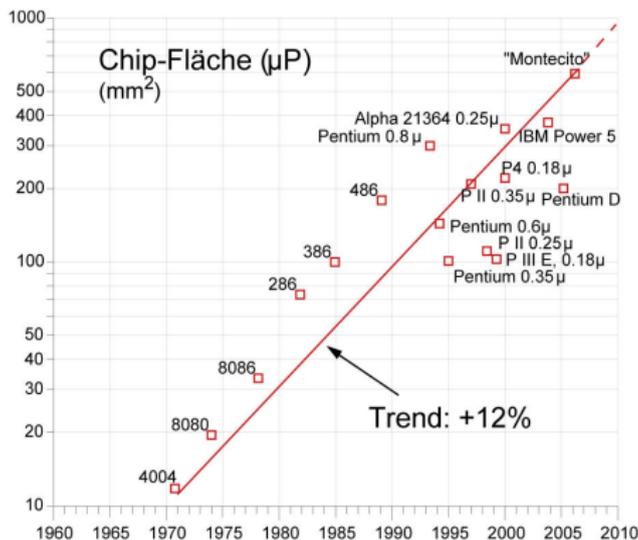
Verbesserung der Fertigungsprozesse

Auswirkungen - Transistoren schrumpfen um 13% jährlich



► pro Flächeneinheit $1/0,87^2 = 33\%$ mehr Elemente

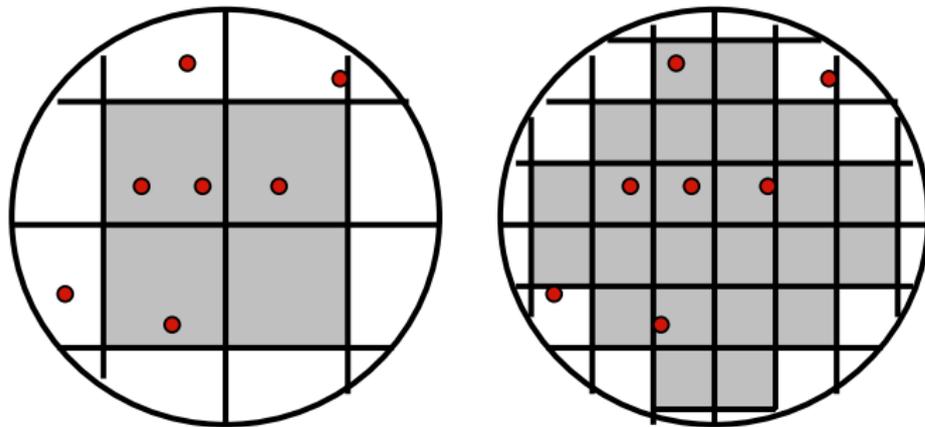
Nicht nur kleinere Strukturen, auch größere Chip-Flächen



Beispiel 2016: Nvidia GP100 "Pascal" GPU mit 610 mm^2

Ausbeute

Effekte der Chip-Größe



Heute zuverlässig erreichbar: Nur ca. 1 Fehler pro cm^2 .

Quelle: Quelle: J. Rabaey, Digital Integrated Circuits

Beispiel Cell-Prozessor - Layout

Verwendet unter anderem in PlayStation 3



TECHNISCHE
UNIVERSITÄT
DARMSTADT

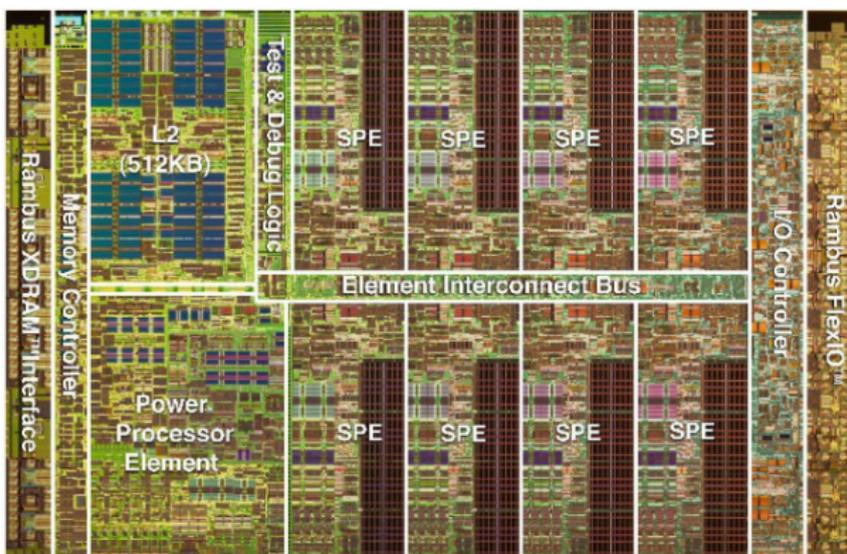
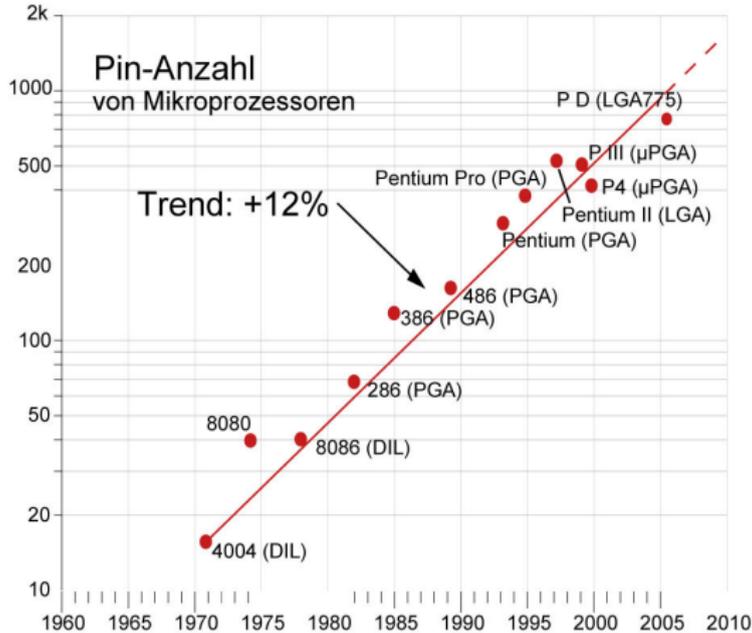


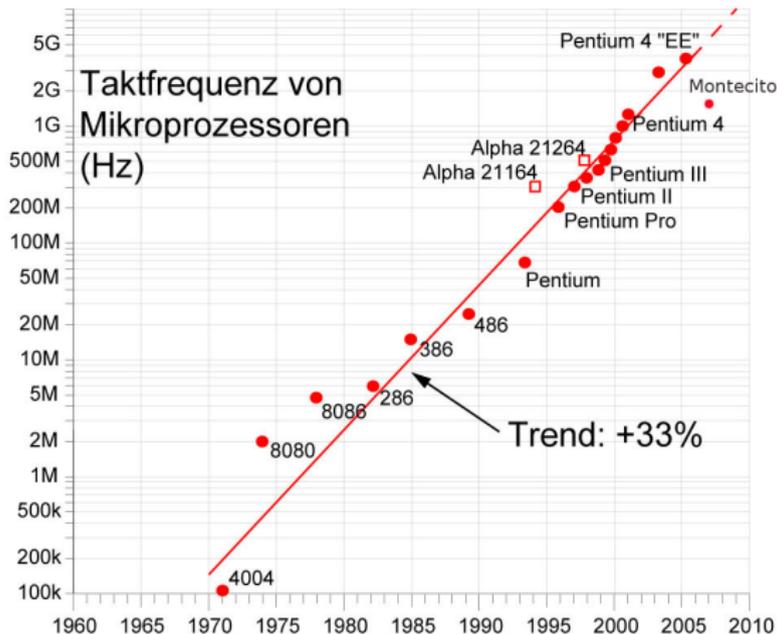
Figure: Layout (Die) des Cell-Prozessors, Quelle: IBM

In der PlayStation 3 werden nur sechs SPEs verwendet.



Problem: Komplexität (+53% p. a.) wächst stärker als Kommunikationsmöglichkeit

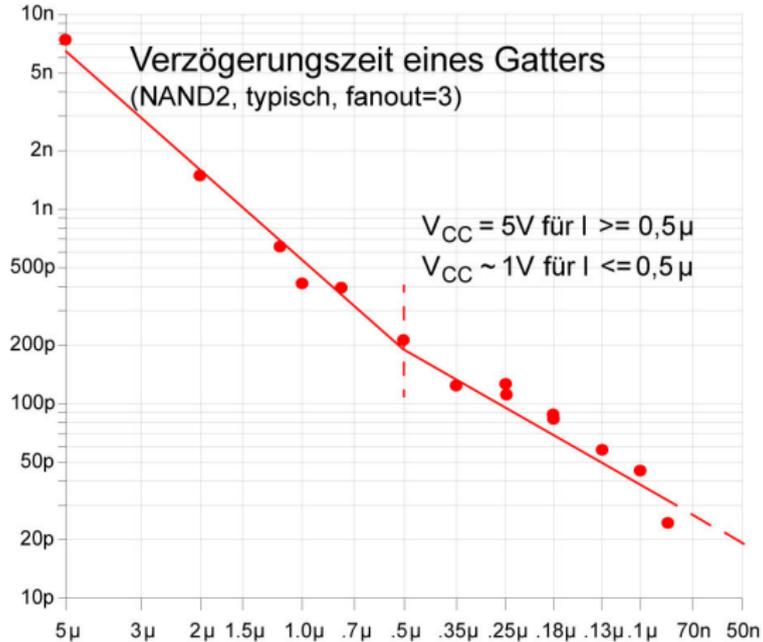
Taktfrequenz - Entwicklung

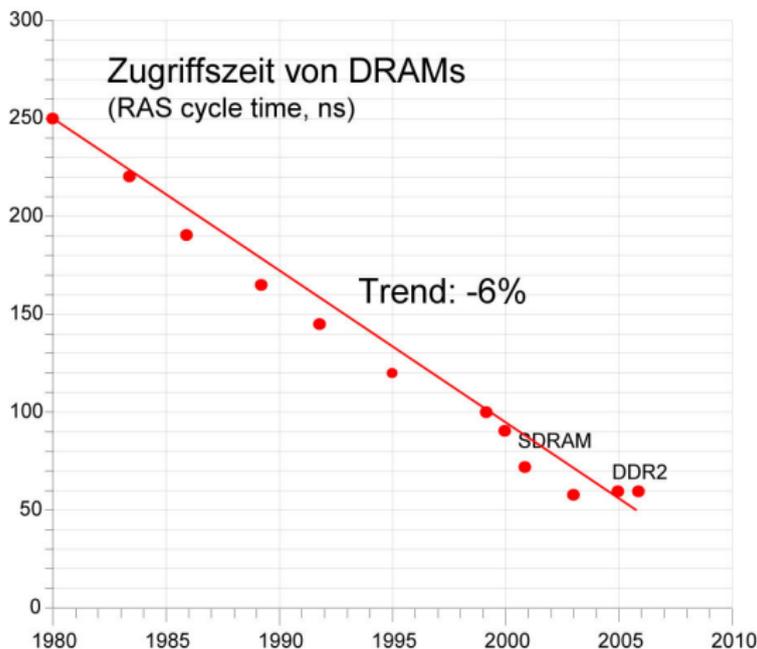


Aktuell (2016): IBM z13 CPU mit 5 GHz



- ▶ Leistungssteigerung wurde lange Zeit durch Erhöhen der Taktfrequenz erreicht.
- ▶ Aktuell liegt der Prozessortakt vieler Mikroprozessoren bei 3...4 GHz.
- ▶ Aber nicht immer höhere Taktfrequenz → schnellerer Prozessor
 - ▶ Intel Pentium 4 EE, 3,8 GHz Takt, 11,5 SPECint2006
 - ▶ Intel Montecito 9050, 1,6 GHz Takt, 14,5 SPECint2006
- ▶ Bedingt durch die Technologie (CMOS-Technologie) steigt der Leistungsumsatz der Prozessoren mit dem Takt ($P \approx U^2 \cdot f \cdot C_L$).
- ▶ Die entstehende Wärme ist nur mit großem Aufwand abzutransportieren.
- ▶ Parallelrechner:
 - ▶ Integration mehrerer CPUs auf einem Chip (aktuell: 2-60 CPU-Kerne von Intel/AMD/IBM verfügbar)
 - ▶ Massiv parallele Systeme mit mehreren tausend Prozessoren, Spezialarchitekturen z. B. Vektorrechner

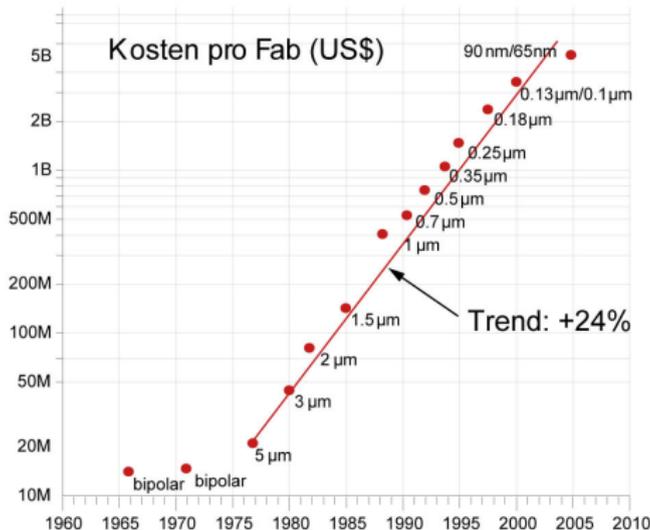




Gatterlaufzeiten liegen im ps-Bereich. Hauptspeicher ist vergleichsweise langsam.

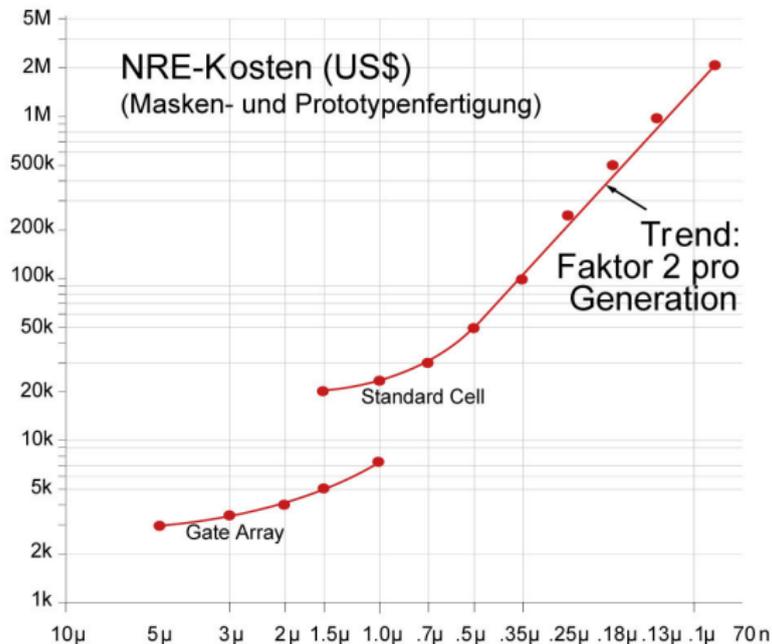


- ▶ +53% p. a. Chip-Komplexität
 - ▶ Zahl der Transistoren, Speichergröße
- ▶ +33% p. a. Packungsdichte
 - ▶ Elemente/Flächeneinheit
- ▶ +33% p. a. Taktfrequenz
- ▶ +12% p. a. Chip-Fläche
- ▶ +12% p. a. mehr Pins (Flaschenhals!)
- ▶ -6% p. a. Speicherzugriffszeit (Flaschenhals!)



- ▶ In zehn Jahren haben sich die Kosten für eine Fabrik fast verzehnfacht.
- ▶ Aktueller Stand 22nm Technologie: Ca. USD 10 Milliarden pro Fabrik

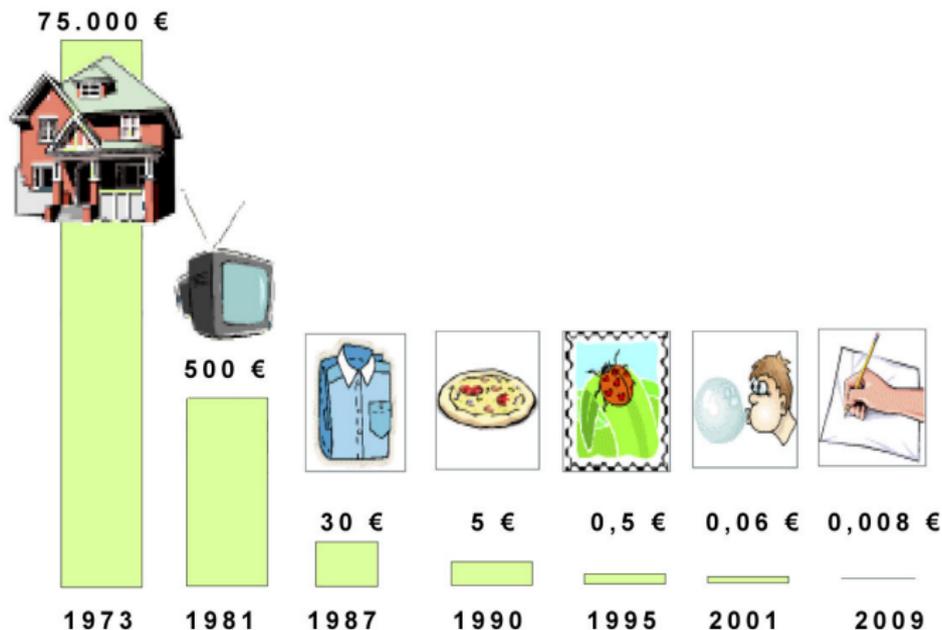
Kostenentwicklung II



- ▶ Fertigung auf älterer Technologie
 - ▶ 180nm ... 110nm *sehr* weit verbreitet
- ▶ Multi-Projekt-Chips
 - ▶ Sonderangebote für EU Forschung und Lehre, z.B. von ST Micro
 - ▶ 130nm: EUR 2.200 / mm²
 - ▶ 65nm: EUR 7.500 / mm²
 - ▶ 28nm: EUR 12.000 / mm²
 - ▶ Für 15 Chips, Gehäuse kosten extra (EUR 30-200 je Stück)
- ▶ Programmierbare/konfigurierbare Schaltungen
 - ▶ Keine photochemische Chip-Fertigung mehr erforderlich
 - ▶ Beispiele sind: PALs, PLAs, FPGAs
 - ▶ FPGAs werden gerne zum Rapid-Prototyping eingesetzt.

Kostensenkung durch Massenfertigung

Kosten für 1 Mb DRAM





Hardware-Entwurfstechniken

- ▶ Vergleichbar einem Puzzle mit einer Milliarde Teile
- ▶ Zusammensetzen unter hohem Zeitdruck
 - ▶ Time-to-Market (TTM)
- ▶ Ein einziger Fehler kann Millionen USD kosten
 - ▶ Erneute Chip-Fabrikation (“re-spin”)
 - ▶ Intel Pentium Bug (FDIV): deutlicher Gewinneinbruch

Wie bekommt man die Komplexität eines solchen Entwurfs mit 100 Millionen Transistoren (und mehr) in den Griff?



- ▶ Abstraktere Vorgehensweisen
- ▶ Beschreibe
 - ▶ ... nicht mehr einzelne Transistoren
 - ▶ ... sondern komplette Systeme
- ▶ Vergleichbar Software-Entwicklung
 - ▶ ... statt Assembler
 - ▶ ... Beschreibung von Systemen als interagierende Komponenten (service-oriented architectures)
- ▶ Mittel der Wahl: (Hardware)-Beschreibungssprachen
 - ▶ Sehr abstrakt: MATLAB/Simulink (Signalverarbeitung)
 - ▶ Abstrakt: SystemC
 - ▶ Noch recht hardware-nah: Verilog HDL - VHDL → Bluespec SystemVerilog
- ▶ Entkoppeln von Entwurf und technischer Realisierung
- ▶ Umsetzung idealerweise automatisch (Synthese)
 - ▶ Klappt aber noch nicht immer!

Entwurfsebenen 1

Unterschiedlicher Abstraktionsgrad

▶ **Verhaltensebene**

Was soll passieren? Realisierung bleibt offen.

$$y = f(x)$$

▶ **Systemebene**

Grobe Aufteilung von Struktur, Zeit, Daten und Kommunikation

CPU, FPGA, DRAM,

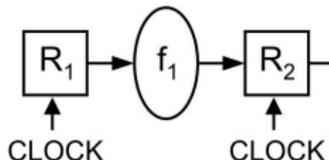
4 Busse, 32b Integer

▶ **Register-Transfer-Ebene**

Synchron, getaktet

always @(posedge CLOCK)

```
R2 <= f1(R1);
```

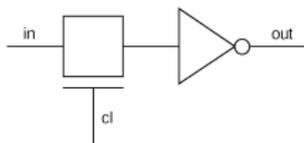


Entwurfsebenen 2

Unterschiedlicher Abstraktionsgrad

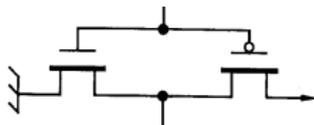
► Logik- oder Gatterebene

Netze aus Gattern, Flip-Flops, etc.



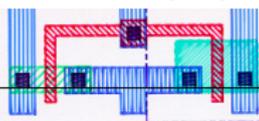
► Transistorebene

Elektrischer Schaltplan



► Layoutebene

Maßstabsgetreue geometrische Anordnung des Chips mit verschiedenen Schichten (3D)



- ▶ Verilog HDL unterstützt auch die Beschreibung des Verhaltens.

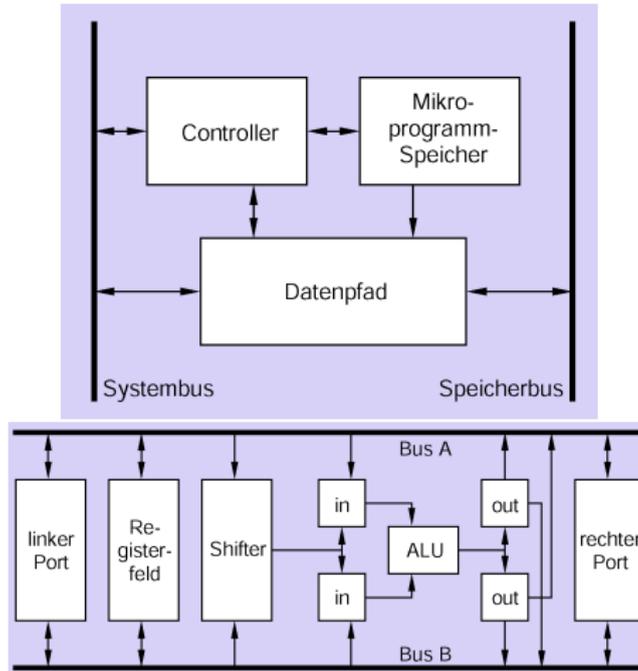
```
module MULT_SCALED2(  
  input wire [15:0] a, b,  
  output wire [32:0] prod  
);
```

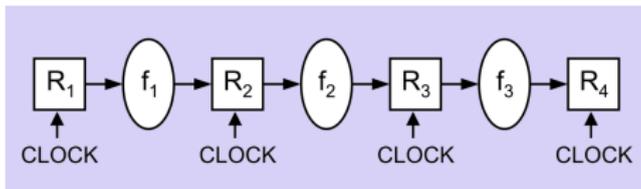
```
  assign prod = a * b * 2;
```

```
endmodule
```

- ▶ Keine Angaben über
 - ▶ Art des Multiplizierers (seriell, parallel, seriell/parallel)
 - ▶ Zeitverhalten

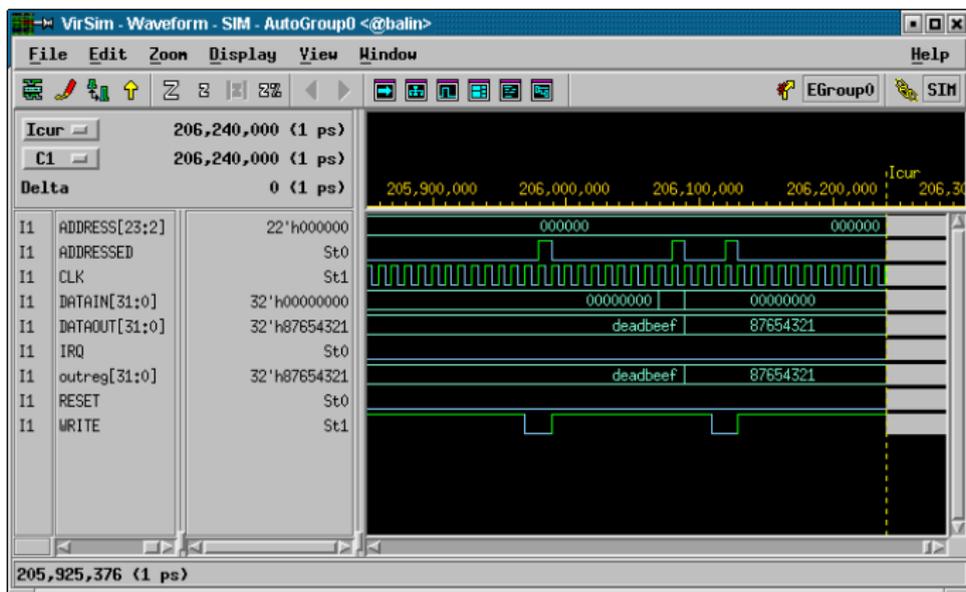
Systemebene Komponenten eines Mikrokontrollers



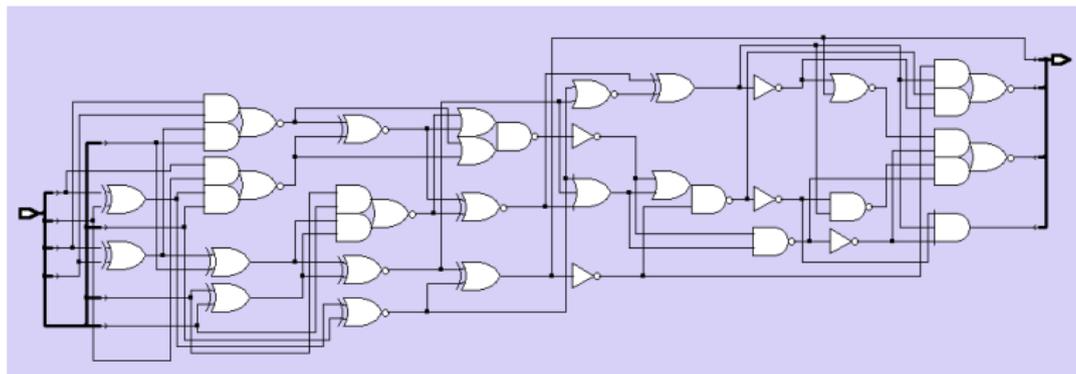


```
always @(posedge CLOCK) // mit jeder steigenden
begin                // Taktflanke
    R2 <= f1(R1);    // Register-Transfer von
    R3 <= f2(R2);    // Ri durch fi nach Ri+1
    R4 <= f3(R3);
end
```

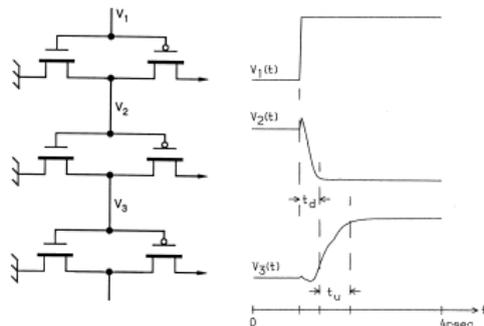
- ▶ Sehr wichtige Entwurfsebene
- ▶ Fließbandverarbeitung (Pipelines)/Automatennetze
- ▶ Effiziente Umsetzung in Hardware automatisch möglich



- ▶ Digitalsimulation noch ohne reale Verzögerungszeiten
- ▶ Alternativ auch Textausgabe möglich

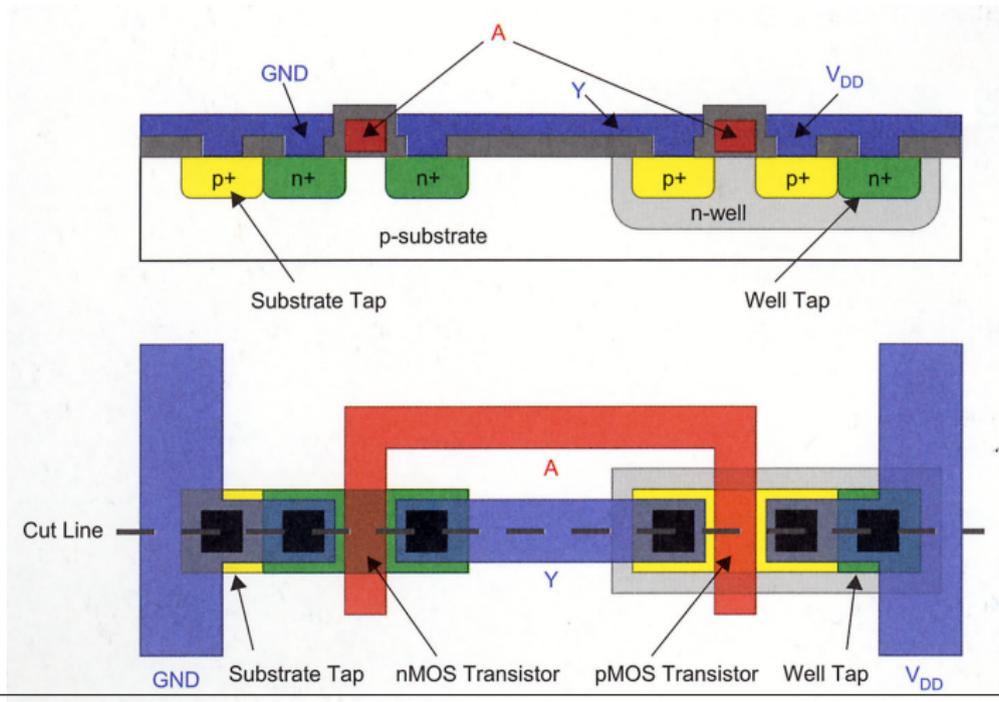


- ▶ Ergebnis der Logiksynthese
- ▶ Generische AND, NOR, Flip-Flops, etc.
- ▶ Zeitverhalten abschätzbar (aber noch sehr ungenau)
- ▶ Noch keine endgültige Hardware-Beschreibung
 - ▶ Hängt von konkreter Zieltechnologie ab
 - ▶ ASIC, FPGA, Gate-Array, ...



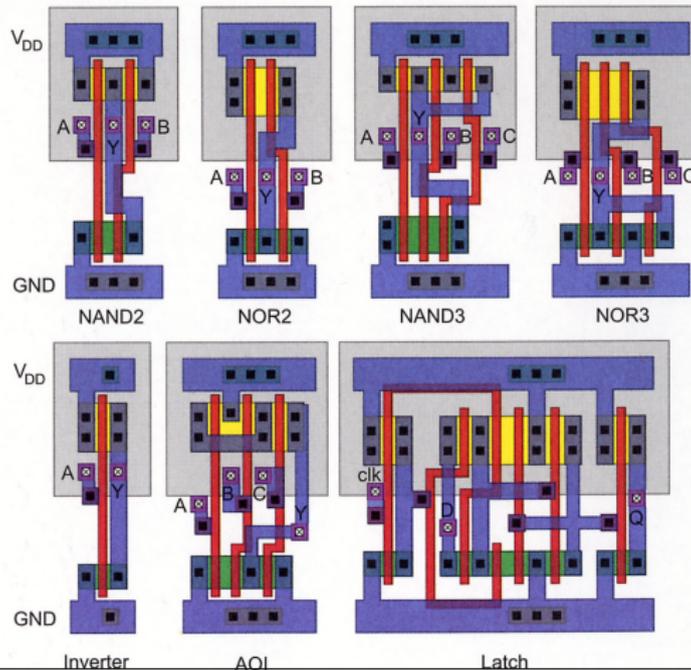
- ▶ Schaltpläne aus Transistoren, Widerständen, etc.
- ▶ Beim Digitalschaltungsentwurf verborgen
- ▶ Ausnahmen
 - ▶ Analoge Teilschaltungen
 - ▶ Full-Custom-Entwurf
- ▶ Analogsimulation mit Schaltzeiten

Layout-Ebene Geometrien der Transistoren und Leitungen



Layout-Ebene

Einige Basisgatter (=Zellen)



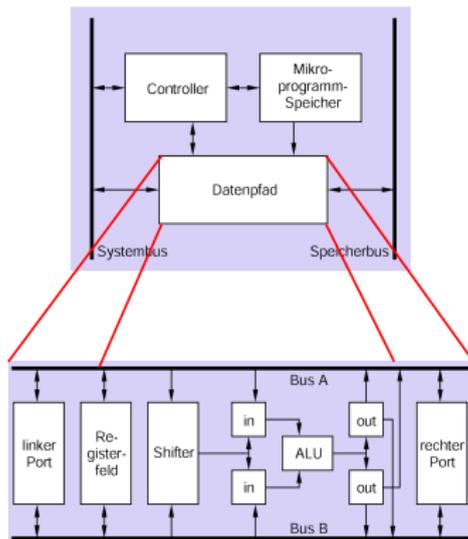


- ▶ Maßstabsgetreue Darstellung des endgültigen Chips
- ▶ Ergebnis von Platzieren und Verdrahten
 - ▶ Transistoren und Leitungen als Polygone
 - ▶ Abmessungen haben Einfluß auf elektrische Eigenschaften
 - ▶ Nun genaues Zeitverhalten bekannt
 - ▶ Weitergabe an Halbleiterhersteller (Tape-Out)

- ▶ Weglassen für die *aktuelle* Beschreibung *unwichtiger* Details
- ▶ Arbeiten auf unterschiedlichen Ebenen
 - ▶ Von ungenau bis sehr genau
 - ▶ Verhaltensebene, . . . , Layout-Ebene
- ▶ Beispiel: Entwurf eines MP3-Encoder-Chips
 - ▶ Manuell von funktionaler bis RTL-Ebene
 - ▶ Andere Schritte i. d. R. automatisch
- ▶ Beispiel: Entwurf von Empfangsteil für UMTS-Handy
 - ▶ Manuell von funktionaler bis Layout-Ebene
 - ▶ Sehr komplizierte Chips
 - ▶ Entwurf erfordert hochspezialisierte Kenntnisse.
 - ▶ Hochfrequenztechnik-Schaltungen sind schwierig zu modellieren.

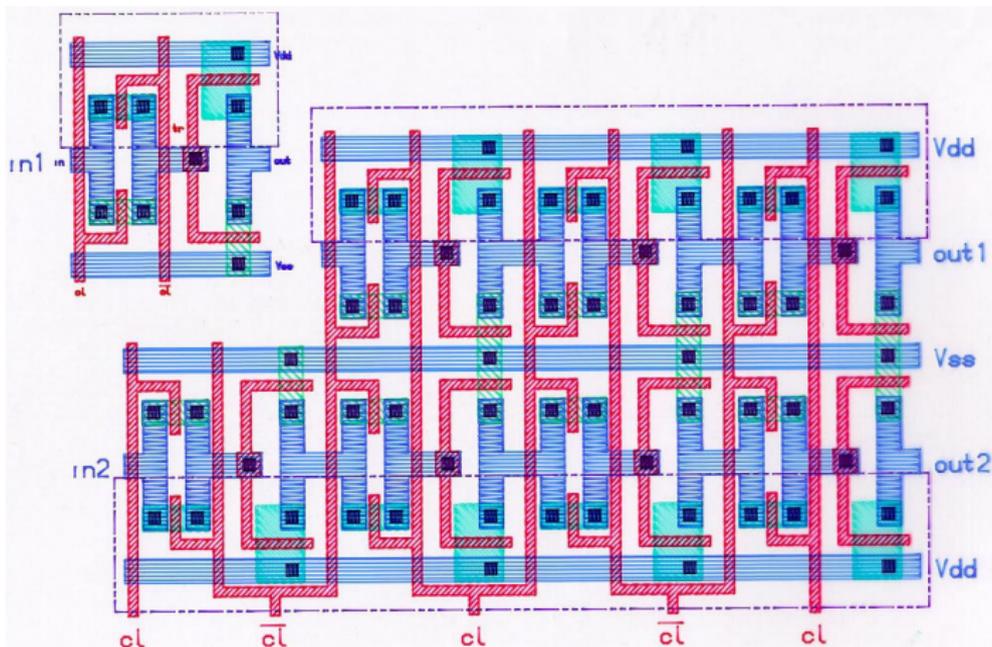
Hierarchische Zerlegung - Aufteilen eines Problems in kleinere Unterprobleme

- ▶ Alte Idee: divide et impera (Philip II, -381 ... -335)
- ▶ Auch rekursiv anwendbar
- ▶ Damit entsteht eine Hierarchie von Zerlegungen



Reguläre Zerlegung

Gezielte Vervielfältigung von Komponenten



- ▶ Top-down-Entwurf (Detaillierungsprozeß)
- ▶ Bottom-up-Entwurf (Kompositionsprozeß)
- ▶ Meet-in-the-Middle-Entwurf
- ▶ Diese Methoden existieren auch im Softwareentwurf.

- ▶ Der Schaltungsentwickler beginnt mit seinem Wissen über die zu realisierende Gesamtfunktion
- ▶ Danach Zerlegung in kleinere Teilfunktionen (Teileinheiten, Komponenten) und ein Verbindungsnetz (Verbindung der Schnittstellen durch Signale)
- ▶ Kriterien: z.B. Kosten, Geschwindigkeit oder Chipfläche
- ▶ Der Prozeß endet, wenn Basis-Funktionen verwendet werden können.

- ▶ Der Top-down-Entwurf setzt voraus, dass sich Teilsysteme zunächst abstrakt, nämlich als „black boxes“ beschreiben lassen.
- ▶ Die nach außen hin sichtbare Funktionalität der Teileinheiten muß abstrakt definierbar sein, ohne dass auf die Details ihrer internen Realisierung näher eingegangen werden muß.
- ▶ Damit ist es möglich, das Zusammenwirken von Systemkomponenten zu evaluieren bevor diese vollständig auf Logik-Ebene entworfen wurden.

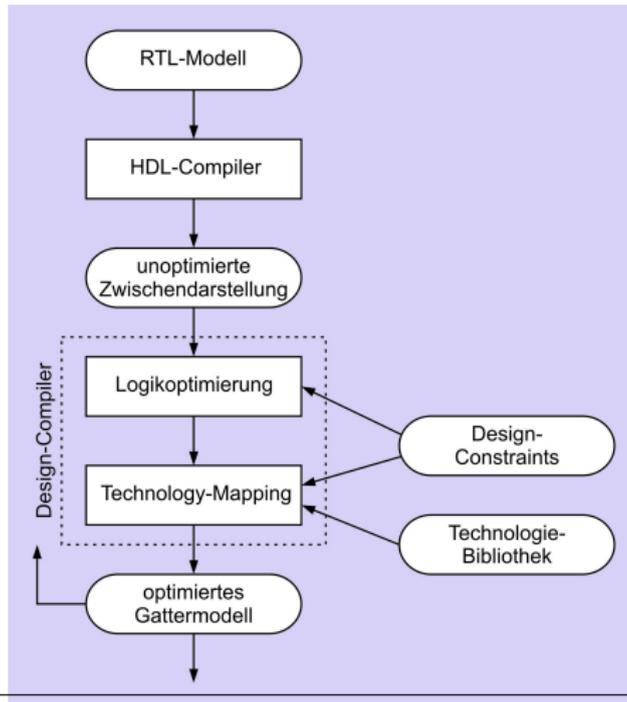
- ▶ Hier wird von den verfügbaren Primitiven (z. B. TTL-Gatter) ausgegangen, welche in einer Bibliothek abgelegt sind.
- ▶ Aus den Bibliothekselementen werden komplexere Komponenten gebildet, die ihrerseits auf der nächst höheren Ebene als (elementare) Bausteine eingesetzt werden können.

- ▶ Der Meet-in-the-middle-Entwurf vereinigt die Top-down- und die Bottom-up-Vorgehensweise
- ▶ Man beachtet beim Vorgehen von der einen Seite des Entwurfsprozesses (z. B. Top) die Auswirkungen an der anderen



Verfeinerter Ablauf der Synthese

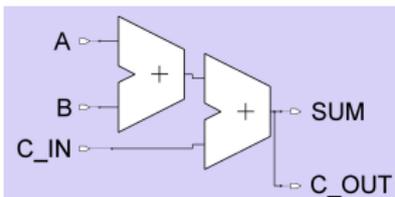
Verfeinerter Entwurfsablauf der Synthese



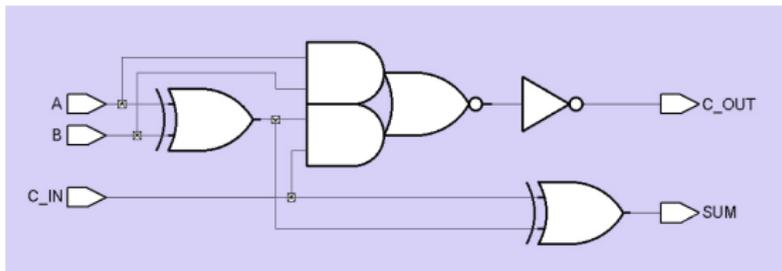
Syntheseablauf: Technologieabbildung

technology mapping

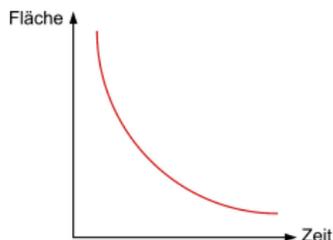
Unoptimierte Zwischendarstellung eines 1b-Addierers



Ergebnis für ASIC-Zielbibliothek (LSI 10K)



- ▶ Zeit
 - ▶ Timing-Analyse
 - ▶ Geschätzt nach Synthese (ohne Verdrahtungsverzögerung)
 - ▶ Exakt nach Platzieren und Verdrahten
 - ▶ ➡ Layout-Ebene
- ▶ Fläche
 - ▶ Geschätzt nach Synthese (ohne Verdrahtungsfläche!)
 - ▶ Exakt nach Platzieren und Verdrahten
- ▶ Elektrische Leistungsaufnahme
 - ▶ Simulation auf Layout-Ebene
 - ▶ Bestimmung von Umschaltfrequenzen (*toggle frequencies*) von Signalen



Beispiel: 4b-Vergleicher

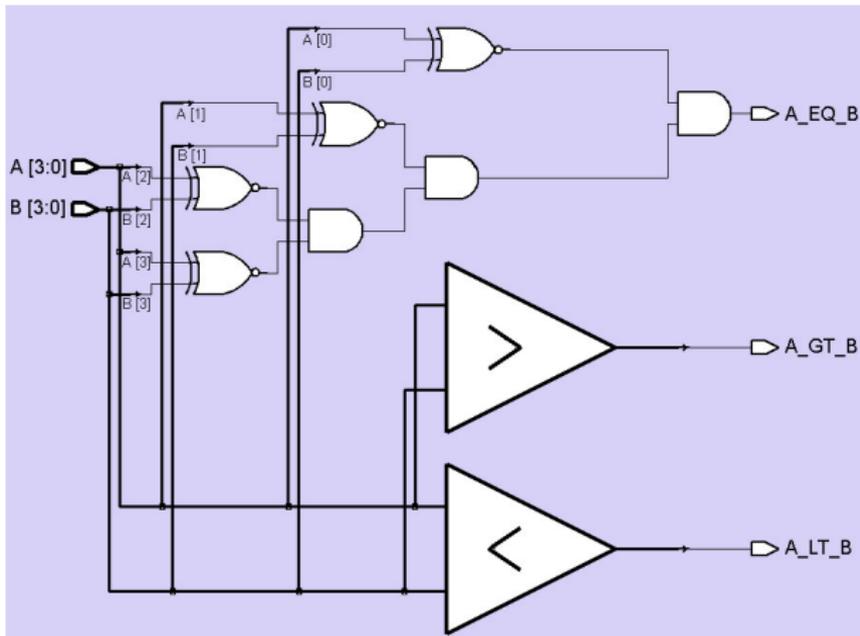
- ▶ Größenvergleich von zwei 4b breiten Eingabewerten A und B
- ▶ Bestimmt Flags für $<$, $>$, und $=$
- ▶ Erstes Ziel: Möglichst schnelle Schaltung, Fläche egal

```
// Vergleich
module mag_comp (
  input wire [3:0] A, B,
  output wire      A_GT_B, A_LT_B, A_EQ_B);

assign A_GT_B = (A > B);    // A groesser B
assign A_LT_B = (A < B);    // A kleiner B
assign A_EQ_B = (A == B);   // A gleich B

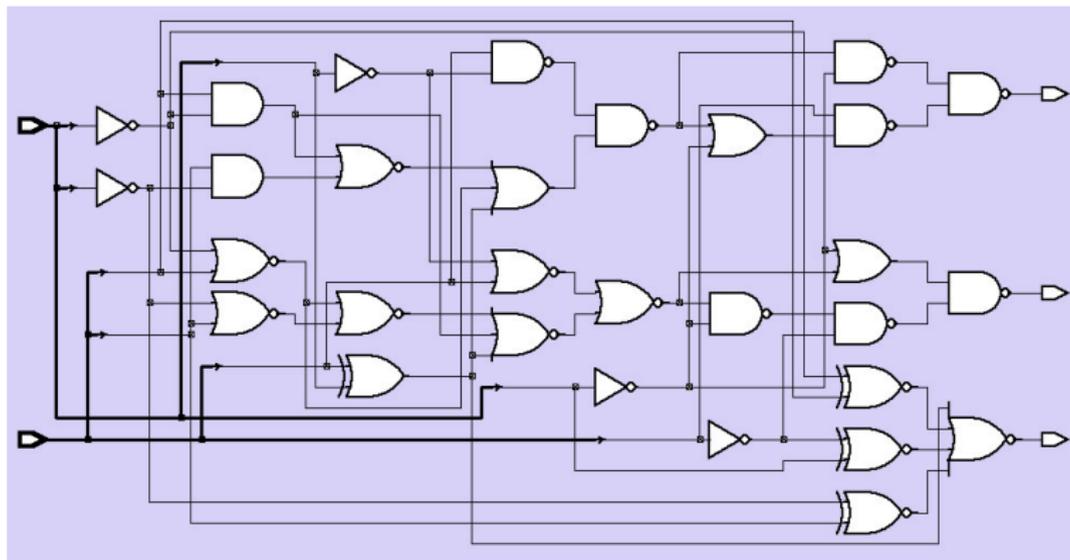
endmodule
```

Zwischendarstellung des 4b-Vergleichers



4b-Vergleicher in LSI Logic 10K ASIC- Technologie

Benutzt um 1993

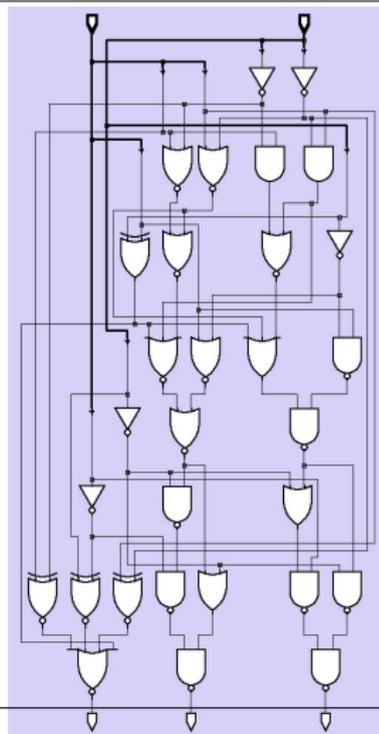


Gatternetzliste des 4b-Vergleichers

Als **strukturelles** Verilog



```
module mag_comp_gate (  
  input wire [3:0] A,  
         B,  
  output wire A_GT_B, A_LT_B, A_EQ_B);  
  
  wire  
  N107, N108, N109, N110, N111, N112, N113, N114, N115,  
  N116, N117, N118, N119, N120, N121, N122, N123, N124,  
  N125, N126, N127, N128, N129, N130, N131, N132, N133;  
  
  nr4 U89 (.A(N107), .B(N108), .C(N109), .D(N110), .Z(A_EQ_B));  
  nd2 U90 (.A(N111), .B(N112), .Z(A_GT_B));  
  nd2 U91 (.A(N113), .B(N114), .Z(A_LT_B));  
  iv U92 (.A(B[0]), .Z(N115));  
  iv U93 (.A(B[1]), .Z(N116));  
  iv U94 (.A(B[2]), .Z(N117));  
  nr2 U95 (.A(N119), .B(N120), .Z(N118));  
  iv U96 (.A(B[3]), .Z(N121));  
  nd2 U97 (.A(N123), .B(N124), .Z(N122));  
  iv U98 (.A(A[3]), .Z(N125));  
  en U99 (.A(N116), .B(A[1]), .Z(N108));  
  en U100 (.A(N125), .B(B[3]), .Z(N109));  
  en U101 (.A(N115), .B(A[0]), .Z(N110));  
  an2 U102 (.A(A[1]), .B(N116), .Z(N126));  
  nr2 U103 (.A(N116), .B(A[1]), .Z(N127));  
  nr2 U104 (.A(N115), .B(A[0]), .Z(N128));  
  nr2 U105 (.A(N127), .B(N128), .Z(N129));  
  nr3 U106 (.A(N129), .B(N126), .C(N107), .Z(N120));  
  nr2 U107 (.A(N117), .B(A[2]), .Z(N119));  
  nd2 U108 (.A(N118), .B(N121), .Z(N130));  
  nd2 U109 (.A(N130), .B(N125), .Z(N114));  
  or2 U110 (.A(N121), .B(N118), .Z(N113));  
  an2 U111 (.A(A[0]), .B(N115), .Z(N131));  
  
endmodule
```



endmodule



```
// 2-Input-AND-Gatter
```

```
// physikalische Zeiteinheit / Simulationsschrittweite
```

```
'timescale 100 ps / 10 ps
```

```
'celldefine
```

```
module an2 (Z, A, B);
```

```
    output Z;
```

```
    input  A, B;
```

```
// Instanz einer VERILOG-Primitive (in DT/AER nicht weiter behandelt!)
```

```
and And1 (Z, A, B);
```

```
// Verzögerungszeiten fuer steigende und fallende Flanken
```

```
specify
```

```
    (A *> Z) = (1, 1);
```

```
    (B *> Z) = (1, 1);
```

```
endspecify
```

```
endmodule
```

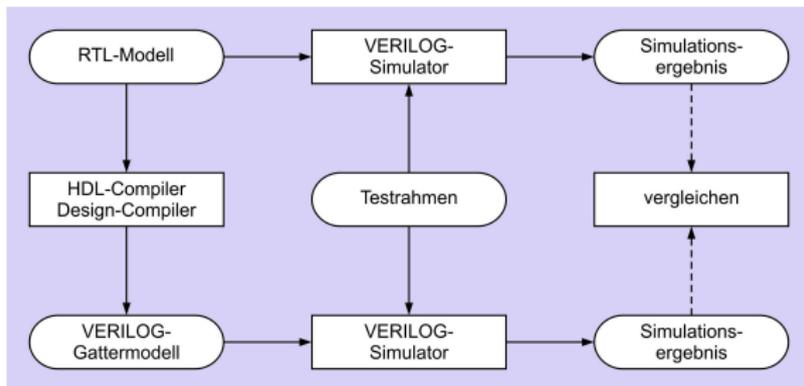
```
'endcelldefine
```



Verifikation

Verifikation

Verhält sich synthetisierte Schaltung noch so wie Simulationsmodell?



- ▶ Prä-Synthese ./ Post-Synthese-Simulation
- ▶ Gleiche Testdaten
 - ▶ Bei Post-Synthese aber genauere Tests möglich
 - ▶ Hier nun **genauer**es Zeitverhalten
- ▶ **Differenzen** in Ergebnissen durch anderes Zeitverhalten
- ▶ Vergleich etwas aufwendiger

Test des 4b-Vergleichers: Prä-Synthese



Testrahmen

```
module stimulus;

reg [3:0] A, B;
wire A_GT_B, A_LT_B, A_EQ_B;

// Instanz des Vergleichers
mag_comp Mag_comp (A, B, A_GT_B, A_LT_B, A_EQ_B);

initial
$monitor ($time,
    "_A=%d,_B=%d,_A_GT_B=%b,_A_LT_B=%b,_A_EQ_B=%b",
    A, B, A_GT_B, A_LT_B, A_EQ_B);

// Testmuster
initial begin
    A = 10; B = 9;
    #10 A = 14; B = 15;
    #10 A = 0; B = 0;
    #10 A = 8; B = 12;
    #10 A = 6; B = 14;
    #10 A = 14; B = 14;
end

endmodule
```

```
0 A=10, B= 9, A_GT_B=1, A_LT_B=0, A_EQ_B=0
10 A=14, B=15, A_GT_B=0, A_LT_B=1, A_EQ_B=0
20 A= 0, B= 0, A_GT_B=0, A_LT_B=0, A_EQ_B=1
30 A= 8, B=12, A_GT_B=0, A_LT_B=1, A_EQ_B=0
40 A= 6, B=14, A_GT_B=0, A_LT_B=1, A_EQ_B=0
50 A=14, B=14, A_GT_B=0, A_LT_B=0, A_EQ_B=1
```

Prä-Synthese-Ergebnis

Vergleich: Prä-Synthese mit Post-Synthese

Gleiche Stimuli wie eben



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Prä-Synthese

0 A=10, B= 9, A_GT_B=1, A_LT_B=0, A_EQ_B=0
10 A=14, B=15, A_GT_B=0, A_LT_B=1, A_EQ_B=0
20 A= 0, B= 0, A_GT_B=0, A_LT_B=0, A_EQ_B=1
30 A= 8, B=12, A_GT_B=0, A_LT_B=1, A_EQ_B=0
40 A= 6, B=14, A_GT_B=0, A_LT_B=1, A_EQ_B=0
50 A=14, B=14, A_GT_B=0, A_LT_B=0, A_EQ_B=1

Post-Synthese

0 A=10, B= 9, A_GT_B=x, A_LT_B=x, A_EQ_B=x
3 A=10, B= 9, A_GT_B=x, A_LT_B=x, A_EQ_B=0
6 A=10, B= 9, A_GT_B=x, A_LT_B=0, A_EQ_B=0
8 A=10, B= 9, A_GT_B=1, A_LT_B=0, A_EQ_B=0
10 A=14, B=15, A_GT_B=1, A_LT_B=0, A_EQ_B=0
16 A=14, B=15, A_GT_B=1, A_LT_B=1, A_EQ_B=0
18 A=14, B=15, A_GT_B=0, A_LT_B=1, A_EQ_B=0
20 A= 0, B= 0, A_GT_B=0, A_LT_B=1, A_EQ_B=0
23 A= 0, B= 0, A_GT_B=0, A_LT_B=1, A_EQ_B=1
28 A= 0, B= 0, A_GT_B=0, A_LT_B=0, A_EQ_B=1
30 A= 8, B=12, A_GT_B=0, A_LT_B=0, A_EQ_B=1
32 A= 8, B=12, A_GT_B=1, A_LT_B=0, A_EQ_B=0
34 A= 8, B=12, A_GT_B=0, A_LT_B=0, A_EQ_B=0
35 A= 8, B=12, A_GT_B=0, A_LT_B=1, A_EQ_B=0
40 A= 6, B=14, A_GT_B=0, A_LT_B=1, A_EQ_B=0
50 A=14, B=14, A_GT_B=0, A_LT_B=1, A_EQ_B=0
53 A=14, B=14, A_GT_B=0, A_LT_B=0, A_EQ_B=1

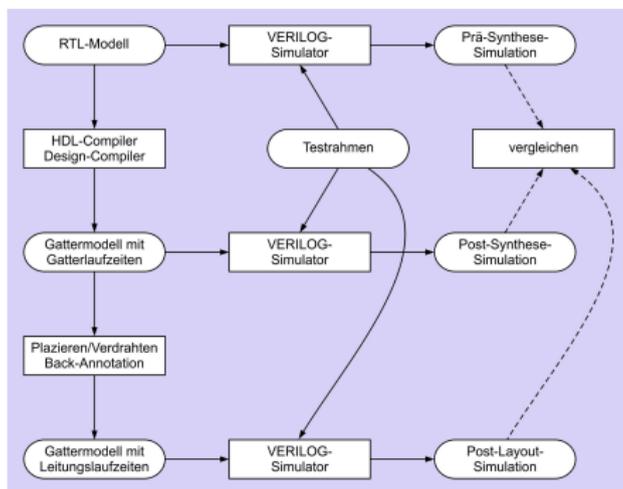
Annahme: Jedes Gatter hat 1
Zeiteinheit **Verzögerung**



- ▶ Unterschiedlich **viele** Ergebnisse
- ▶ Verschiedene **Werte**
- ▶ Unterschiedliche **Zeiten**
 - ▶ Vorher **gar keine** ausser den im Testrahmen
- ▶ Manche Ergebnisse schlicht **falsch** (z.B. bei $t=32$)
- ▶ Interpretation nötig
“Wenn man lange genug wartet, ist das Ergebnis richtig”!
- ▶ Was ist “**lange genug**”?
- ▶ Antwort: **Kritischer Pfad** (TGDI)
- ▶ Damit passender Takt für RTL wählbar **zwischen**
 - ▶ Eingangsregistern
 - ▶ Ausgangsregistern

Weitere Verfeinerung der Verifikation

Nun bis auf Layout-Ebene



- ▶ Post-Layout-Simulation schliesst ein
 - ▶ Gatterverzögerungen
 - ▶ Leitungsverzögerung
 - ▶ Kann umfassen: Widerstände, Kapazitäten, Induktivitäten



Synthesebeispiel: Zero-Counter

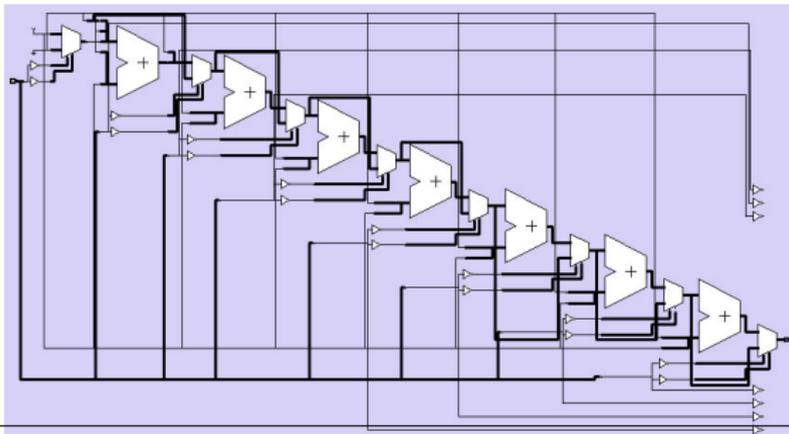


- ▶ Spezifikation
 - ▶ **Eingabe** ist ein 8b Datenwort
 - ▶ **Ausgabe** soll sein die Anzahl der Null-Bits in der Eingabe
- ▶ Genauer betrachtet
 - ▶ RTL-Modell kann Synthese-Ergebnis **direkt** beeinflussen
 - ▶ Wirkung von **Design-Constraints**
- ▶ Nicht mehr so relevant
 - ▶ **Konkrete** Umsetzung in Gatter-Modell
 - ▶ Bei größeren Schaltungen oft schlicht zu **unübersichtlich**

Zero-Counter: Intuitive Lösung

```
module count(  
  input wire [7:0] IN,  
  output reg [3:0] OUT);  
  
  integer I;  
  
  always @(IN) begin  
    OUT = 0;  
    for (I=0; I<=7; I=I+1)  
      if (IN[I]==0) OUT = OUT + 1;  
  end  
endmodule
```

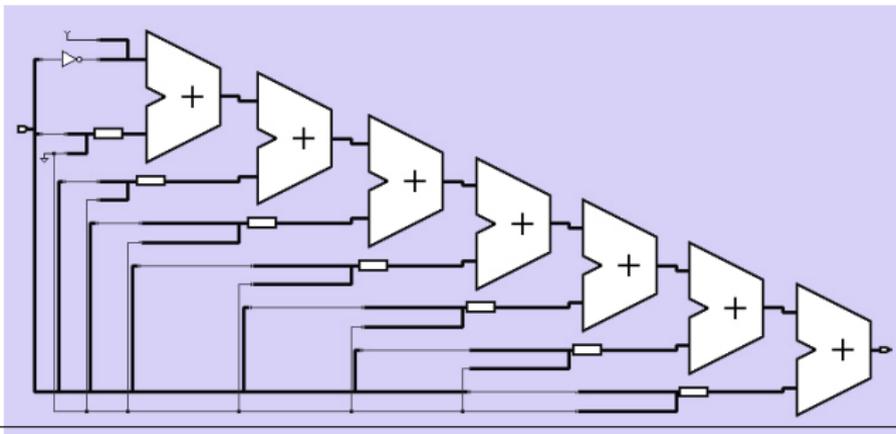
- ▶ for-Schleife wird räumlich abgerollt
- ▶ Addierer-Kaskade
- ▶ Multiplexer wählen bei jedem Bit, ob addiert wird



Zero-Counter: Vereinfachte Lösung

```
module count(  
  input wire [7:0] IN,  
  output reg [3:0] OUT);  
  
  integer i;  
  
  always @(IN) begin  
    OUT = ~IN[0];  
    for (i=1; i<8; i=i+1)  
      OUT = OUT + ~IN[i];  
  end  
endmodule
```

- ▶ for-Schleife wird räumlich abgerollt
- ▶ Addierer-Kaskade
- ▶ Multiplexer entfallen, Bits werden direkt aufaddiert



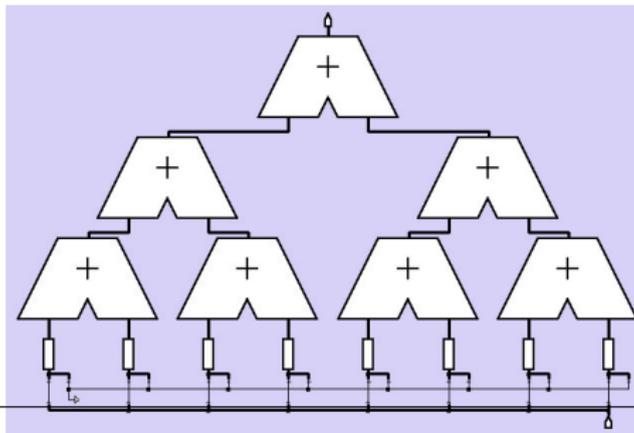
Zero-Counter: Schlaue Lösung

```
module count(  
  input wire [7:0] IN,  
  output reg [3:0] OUT);
```

```
always @(IN)  
  OUT = ((~IN[0]+~IN[1]) + (~IN[2]+~IN[3]))  
        + ((~IN[4]+~IN[5]) + (~IN[6]+~IN[7]));
```

```
endmodule
```

- ▶ Bits werden direkt aufaddiert
- ▶ Jetzt aber **hierarchische** Klammerung
- ▶ Damit **parallele** Berechnung
- ▶ Addierer-**Baum**

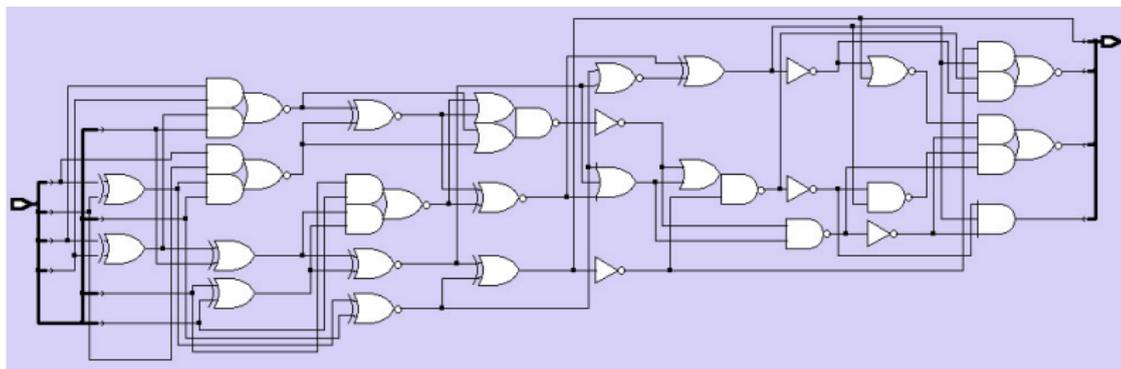




- ▶ Festlegen unterschiedlicher **Optimierungsziele**
- ▶ Üblich
 - ▶ Flächenbedarf
 - ▶ Geschwindigkeit (niedrige Verzögerung)
- ▶ Noch seltener
 - ▶ Energieverbrauch
 - ▶ Ausfallsicherheit

Optimierung auf Fläche

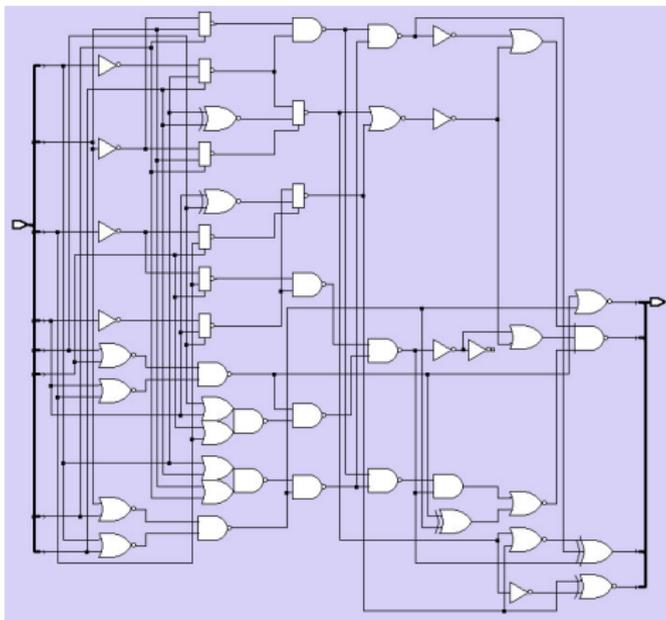
8b-Zero-Counter



52 Gatter, 17.8ns Verzögerung

Optimierung auf Geschwindigkeit

8b-Zero-Counter

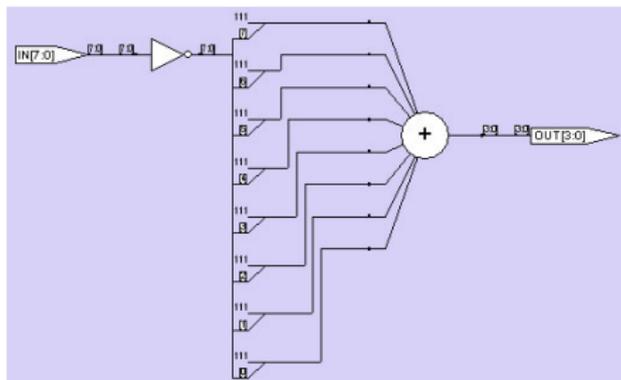


99 Gatter, 8.2ns Verzögerung

Zero-Counter: Noch bessere Lösung

```
module count(  
  input wire [7:0] IN,  
  output reg [3:0] OUT);  
  
  always @(IN)  
    OUT = ((~IN[0]+~IN[1]) + (~IN[2]+~IN[3]))  
          + ((~IN[4]+~IN[5]) + (~IN[6]+~IN[7]));  
  
endmodule
```

- ▶ Schlaueres Synthesewerkzeug
- ▶ Erkennt **Natur** der Berechnung
- ▶ Addiert alle Bits **gleichzeitig** mit
einem 8b-Addierer





Rekonfigurierbare System-on-Chips



Wird auf eigenem Foliensatz behandelt ...