Einführung in Computer Microsystems
Sommersemester 2013
Block 1: Einleitung und Verilog Reloaded

TECHNISCHE UNIVERSITÄT DARMSTADT

Organisatorisches

Inhalt

TECHNISCHE UNIVERSITÄT DARMSTADT

- Weitgehend neu aufgebaute Vorlesung
  - Erste Hälfte: Vertiefung Verilog und Hardware-Entwurf
  - Zweite Hälfte: Einführung Bluespec SystemVerilog
- Änderung: Kaum noch Wiederholung aus TGDI
  - TGDI ist Voraussetzung für Kanonik CMS
- Falls Kenntnisse fehlen bzw. vertieft werden sollen
  - Grundlagen Hardware-Design: TGDI-Aufzeichnungen und Buch
  - Verilog: TGDI-Aufzeichnungen und Buch, alte CMS 2012-Aufzeichnungen

Hintergrundmaterial

Nur zur auszugsweisen Benutzung!

TECHNISCHE UNIVERSITÄT DARMSTADT

Grundlagen TGDI

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

Erster Teil der Vorlesung

Ciletti, Michael D.: *Starter's Guide to Verilog 2001*, Prentice Hall, 2004.
Ciletti, Michael D.: *Advanced Digital Design with the Verilog HDL*, 2. Auflage, Prentice Hall, 2010.
Katz, Randy H.: *Contemporary Logic Design*, Addison-Wesley Longman, 1994.

Zweiter Teil der Vorlesung

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

Organisation

TECHNISCHE UNIVERSITÄT DARMSTADT

Vorlesung

- Mittwoch, 9:50-11:20 Uhr, C 205
- Parallele Übertragung (solange nötig) nach C110 und C120
- In der Regel wird die Vorlesung aufgezeichnet werden

Tafelübung (“große Übung”)

- Mittwoch, 13:30-14:15 Uhr, C205

Freiwillige Kleingruppenübungen

- Verschiedene Termine
- Anmeldung über Moodle bis 24.04.2013, 18:00 Uhr

Software-Werkzeuge

TECHNISCHE UNIVERSITÄT DARMSTADT

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 häufig ausreichend sein

Weitere Lehrveranstaltungen im SoSe 2013

Bereich Technische Informatik

TECHNISCHE UNIVERSITÄT DARMSTADT

- Praktikum Adaptive Computersysteme (P4, 6CP)
- Praktikum Technische Informatik: Eingebettete Systeme (P4, 6CP)
- Seminar zur Technischen Informatik (S2, 3CP)

Informationen unter

`http://www.esa.cs.tu-darmstadt.de`, im Bereich ‘Lehre’

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Entwicklung der Mikroelektronik

Mikroelektronik - Meilensteine

TECHNISCHE UNIVERSITÄT DARMSTADT

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 Sizilium-Einkristall integriert.
  - Höhere Zuverlässigkeit, höhere Schaltgeschwindigkeiten, höhere Packungsdichte.

Mikroelektronik in CMS

TECHNISCHE UNIVERSITÄT DARMSTADT

- Mikroelektronik umfaßt viele Gebiete:
  - Elektrotechnik: Transistorschaltungen/Differentialgleichungen
  - Physik, Verfahrenstechnik/Chemie
- Warum Mikroelektronik in der Informatik?
  - Grundverständnis kann nicht schaden
  - Es geht um Computer Microsystems
  - Zur Entwicklung von Mikroelektronik wird Software verwendet.
  - Die Anforderungen an den Entwurf (z. B. Logikminimierung, Platzierung von Bauelementen) werden mit Algorithmen gelöst.

Bedeutung der Mikroelektronik

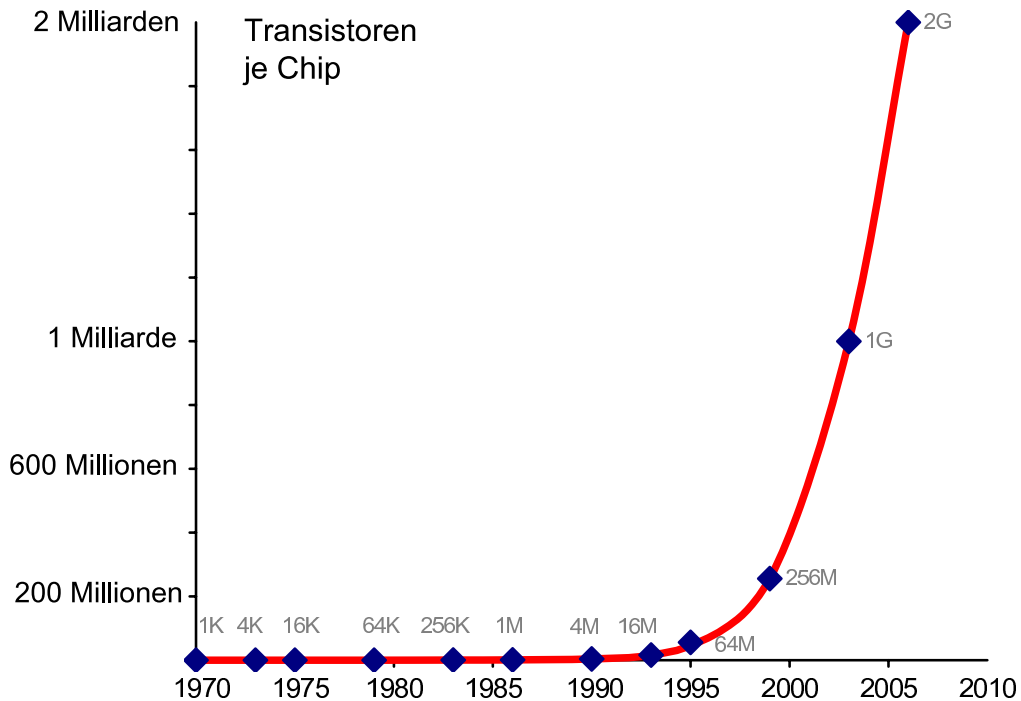
TECHNISCHE UNIVERSITÄT DARMSTADT

- Integrierte Schaltungen (ICs), oft auch nur als **Chips** bezeichnet.
- Chipmarkt: in 2004 Umsatz von 213 Milliarden USD weltweit
  für 2012 Umsatz von 323 Milliarden USD
- Nahezu überall verbaut
  - Offensichtlich: Rechner
    - PC, Server, Supercomputer, ...
  - Versteckt: eingebettete Systeme
    - Autos, Fernseher, Herzschrittmacher, ...
- Alleine in Deutschland: 9 Milliarden EUR Chip-Umsatz
  - 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

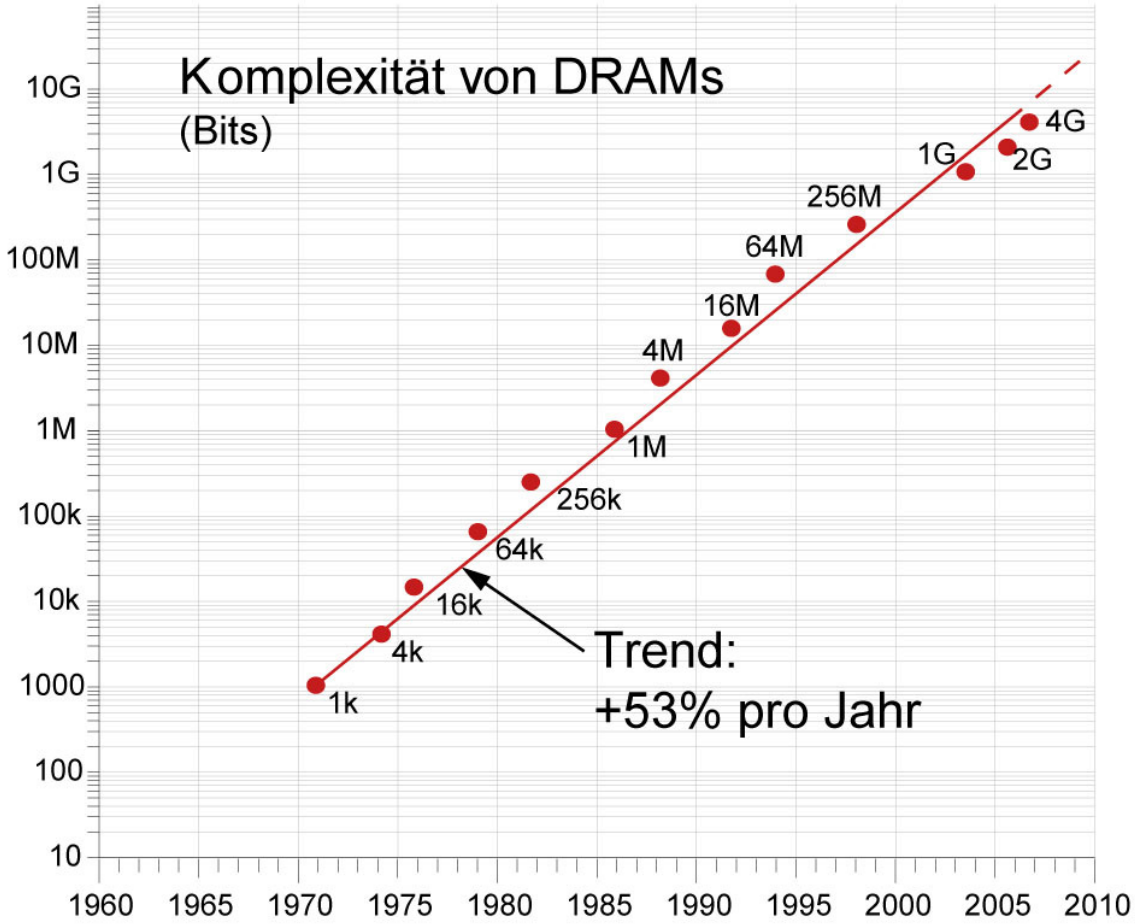
TECHNISCHE UNIVERSITÄT DARMSTADT

Alle 18-24 Monate verdoppelt sich die Anzahl der wirtschaftlichsten Transistoren auf einem Chip. Rekordhalter 2012 NVIDIA GK110: 7.1 Milliarden Transistoren



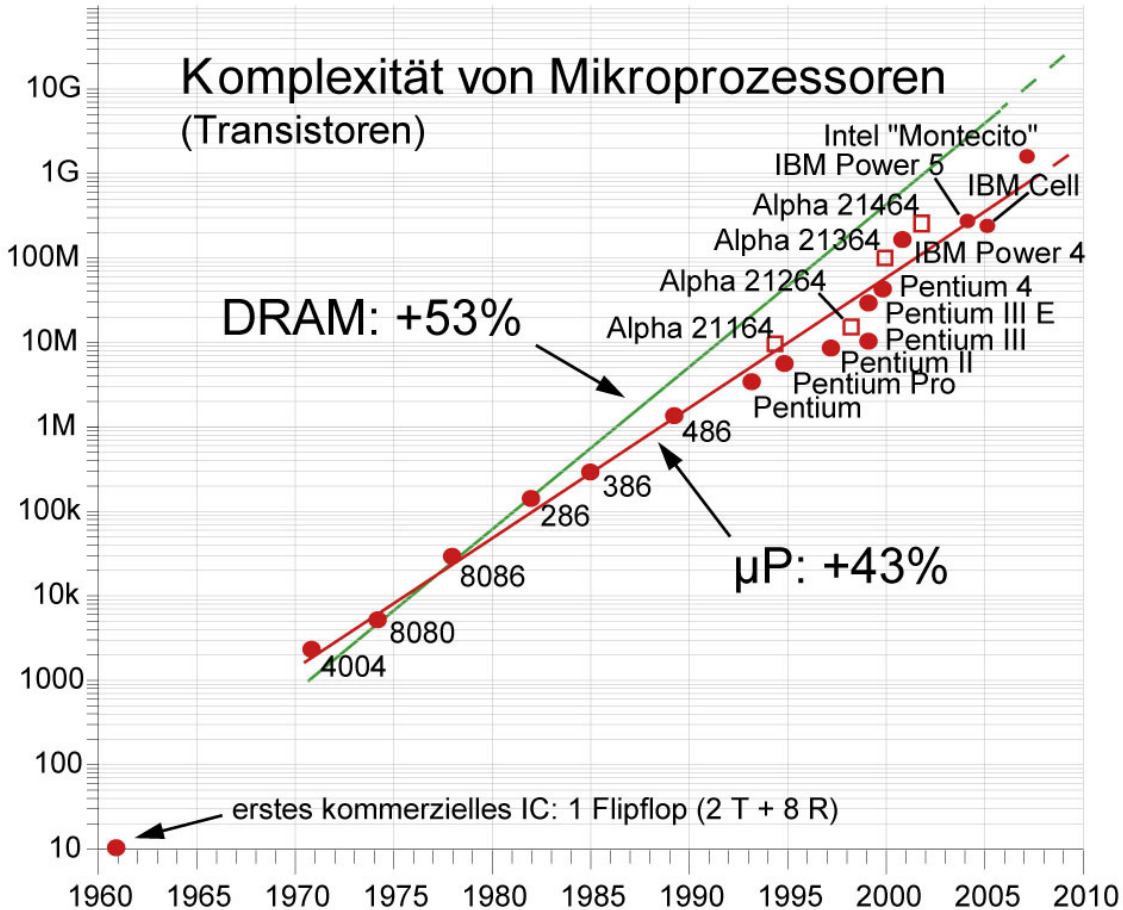
Speicherbausteine

TECHNISCHE UNIVERSITÄT DARMSTADT



Prozessoren

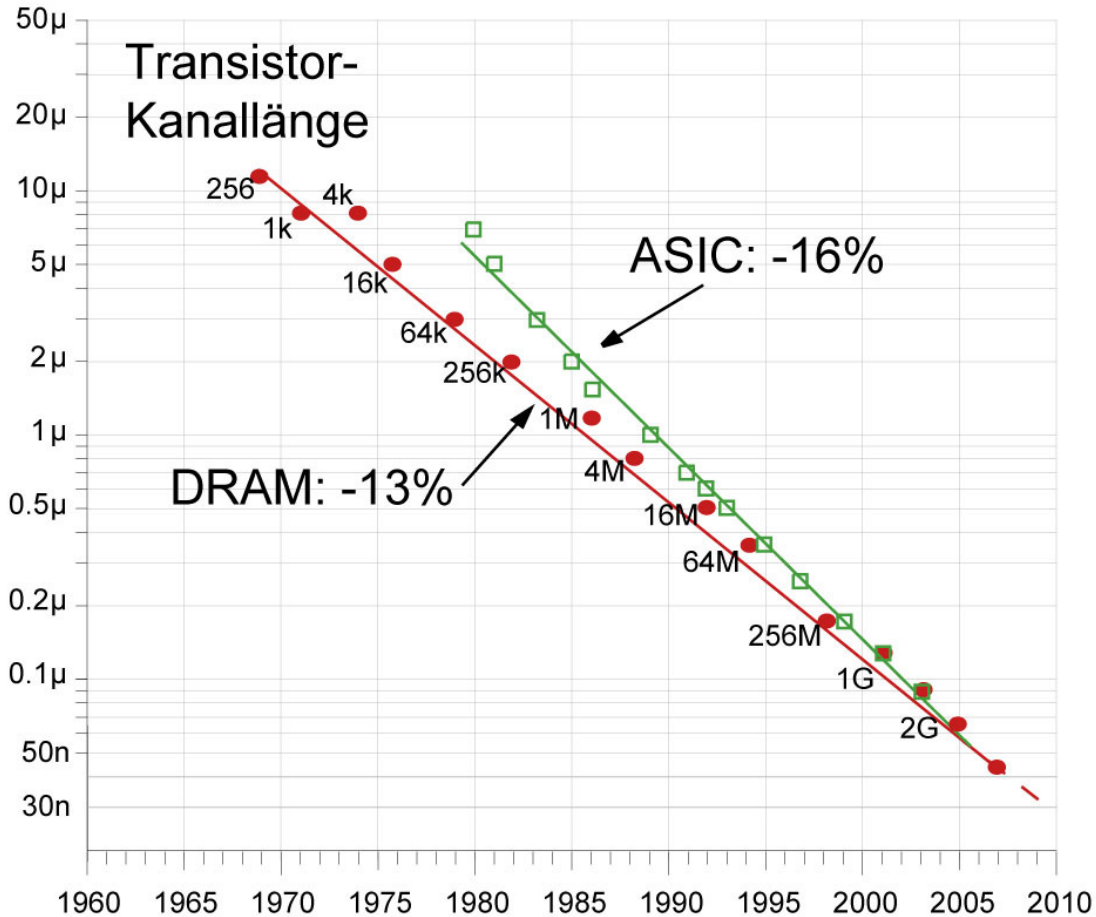
TECHNISCHE UNIVERSITÄT DARMSTADT



Verbesserung der Fertigungsprozesse

Transistor-Kanallängen

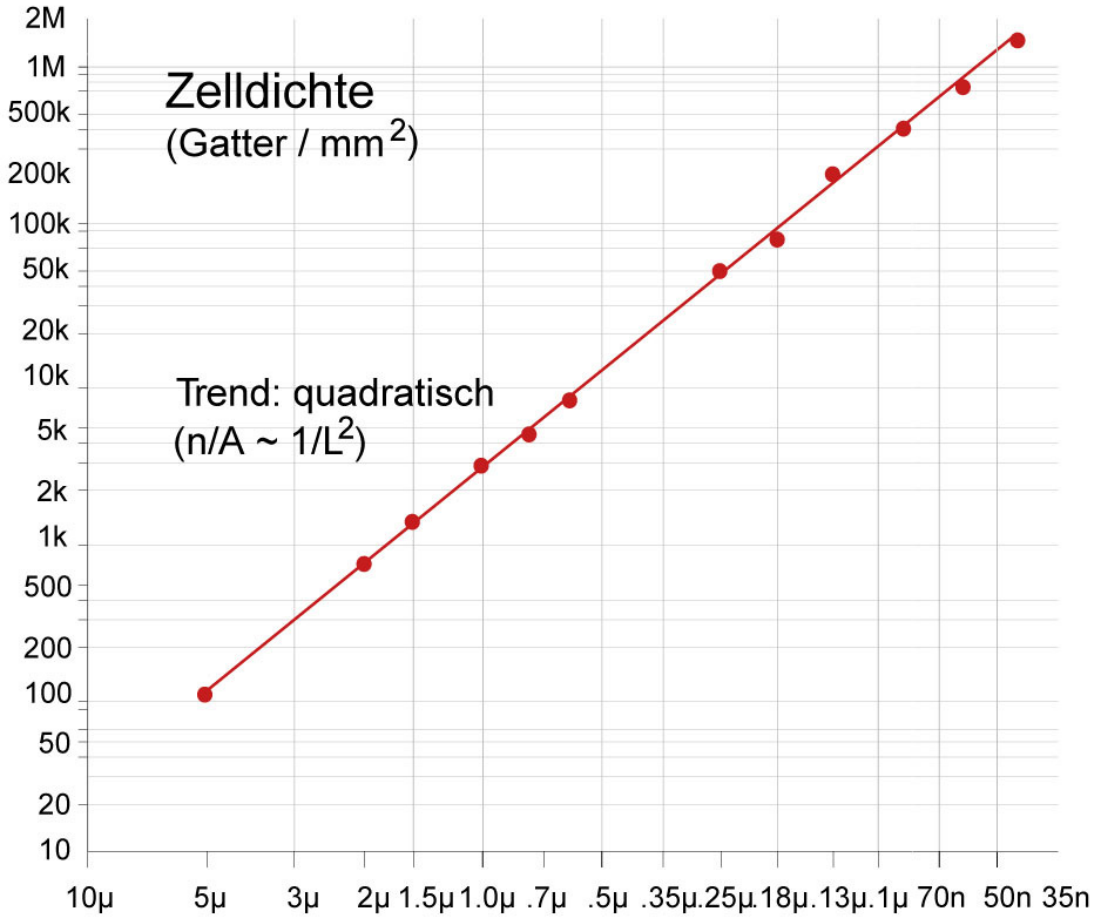
TECHNISCHE UNIVERSITÄT DARMSTADT



Verbesserung der Fertigungsprozesse

Auswirkungen - Transistoren schrumpfen um 13% jährlich

TECHNISCHE UNIVERSITÄT DARMSTADT

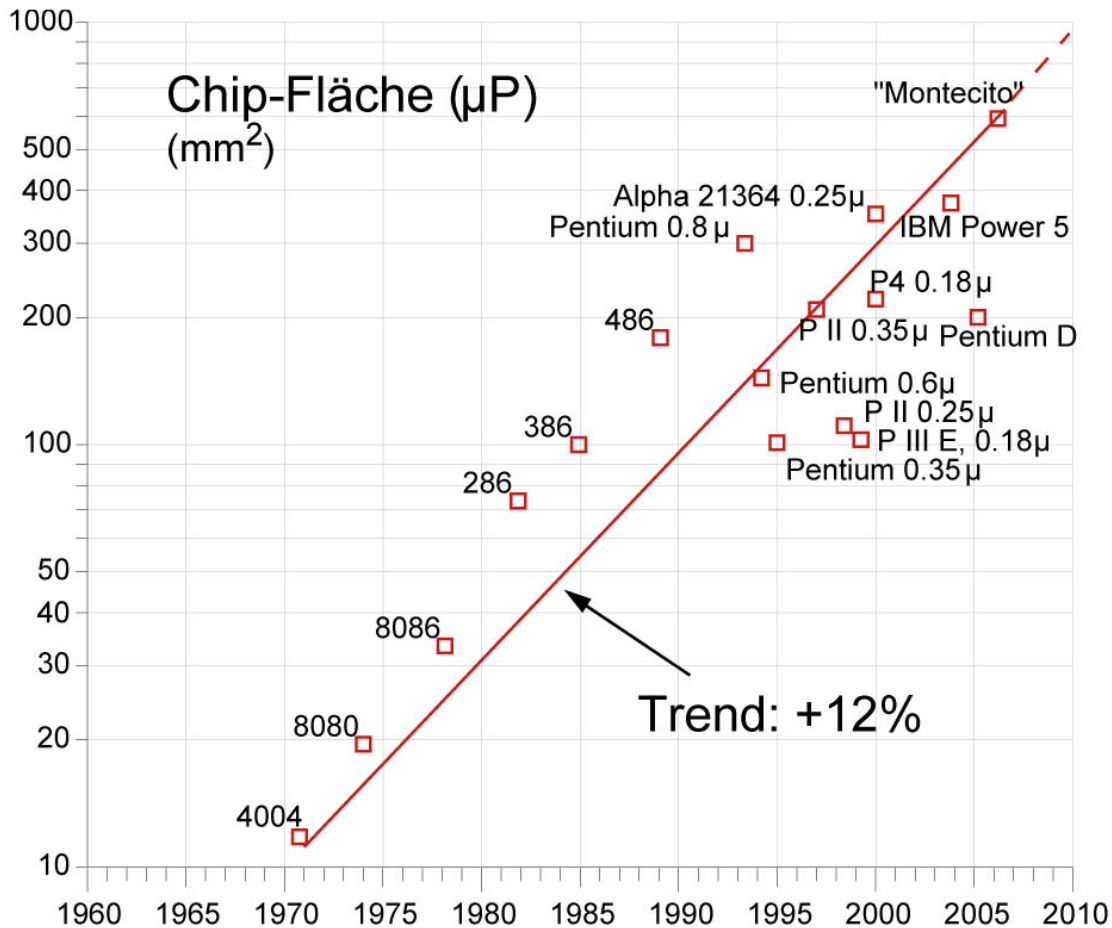


- pro Flächeneinheit $1/0,87^2 = 33\%$ mehr Elemente

Chip-Größen

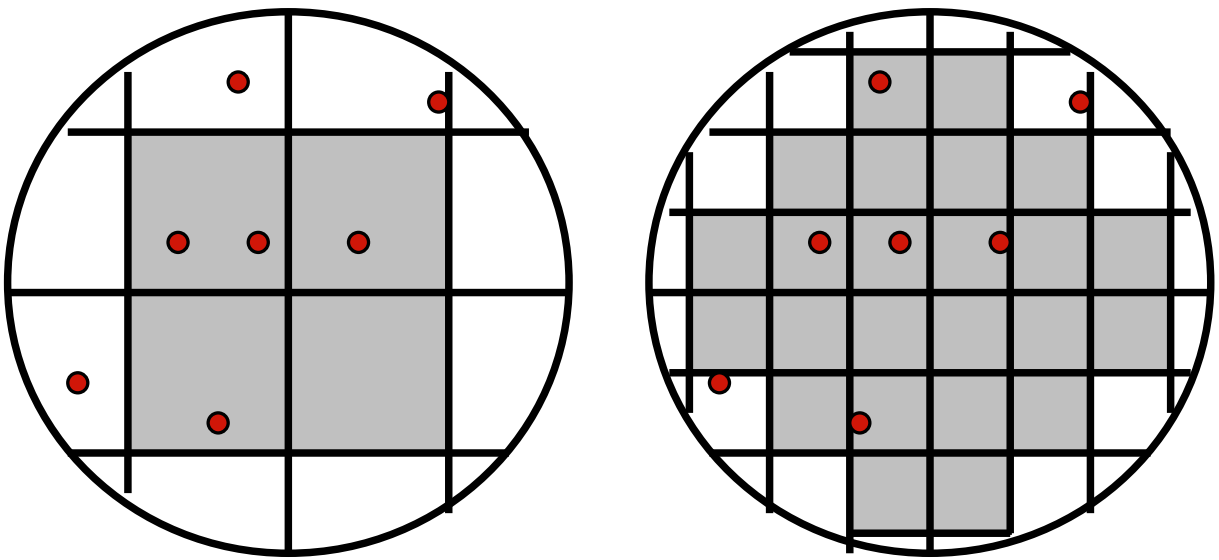
TECHNISCHE UNIVERSITÄT DARMSTADT

Nicht nur kleinere Strukturen, auch größere Chip-Flächen



Ausbeute
Effekte der Chip-Größe

TECHNISCHE UNIVERSITÄT DARMSTADT



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

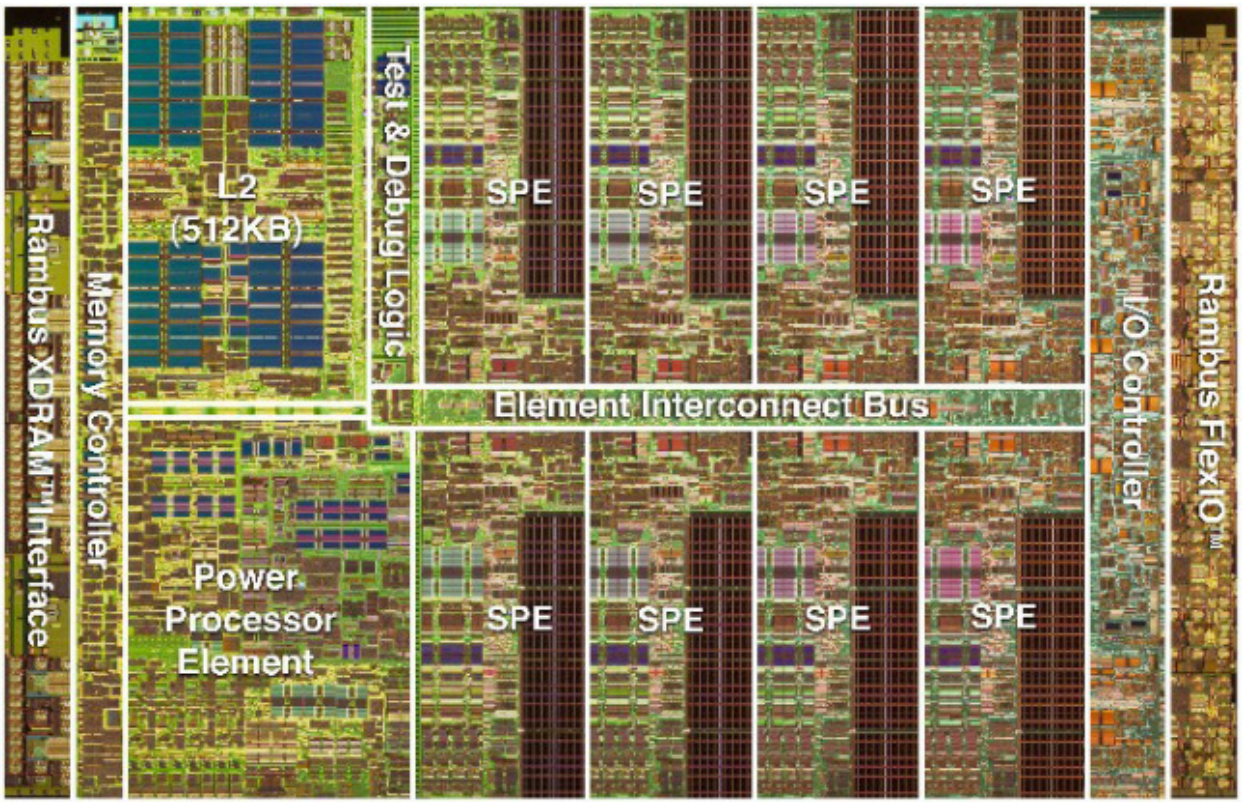
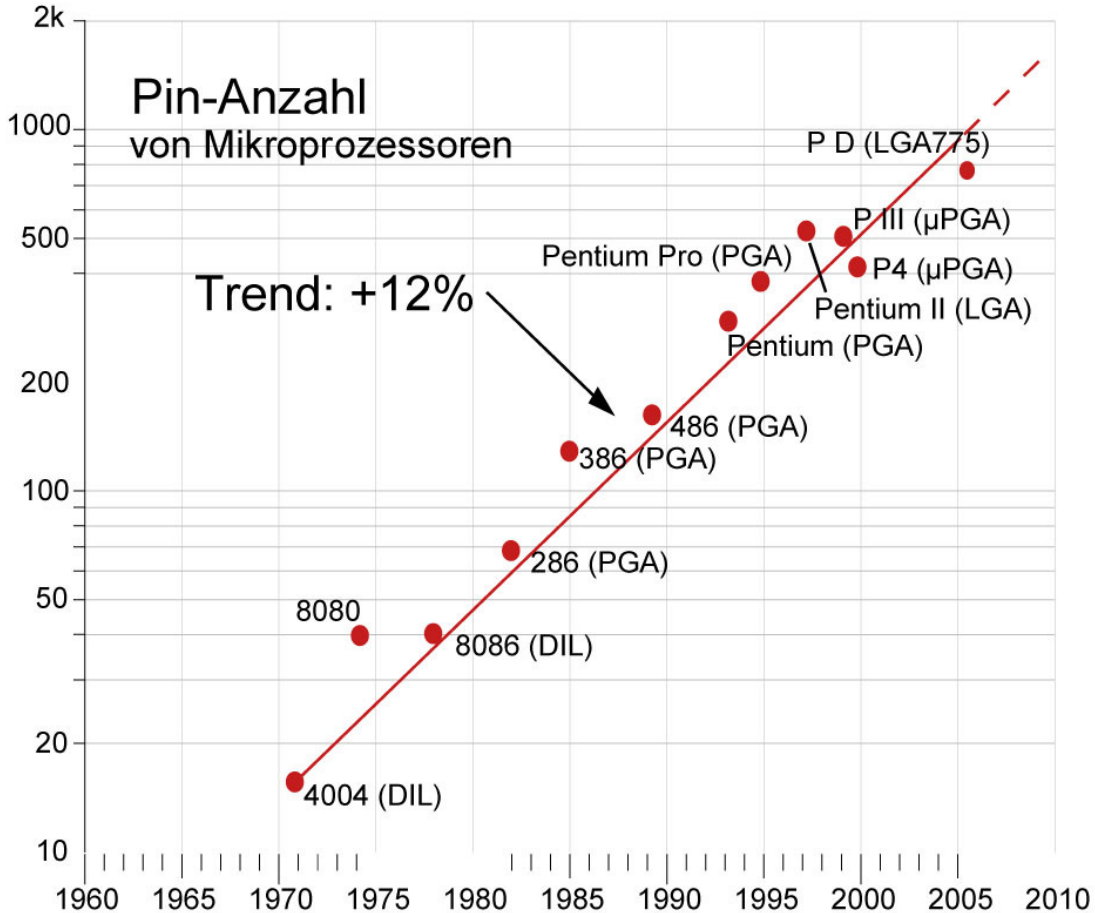


Figure : Layout (Die) des Cell-Prozessors, Quelle: IBM

In der PlayStation 3 werden nur sechs SPEs verwendet.

Kommunikation mit der Aussenwelt

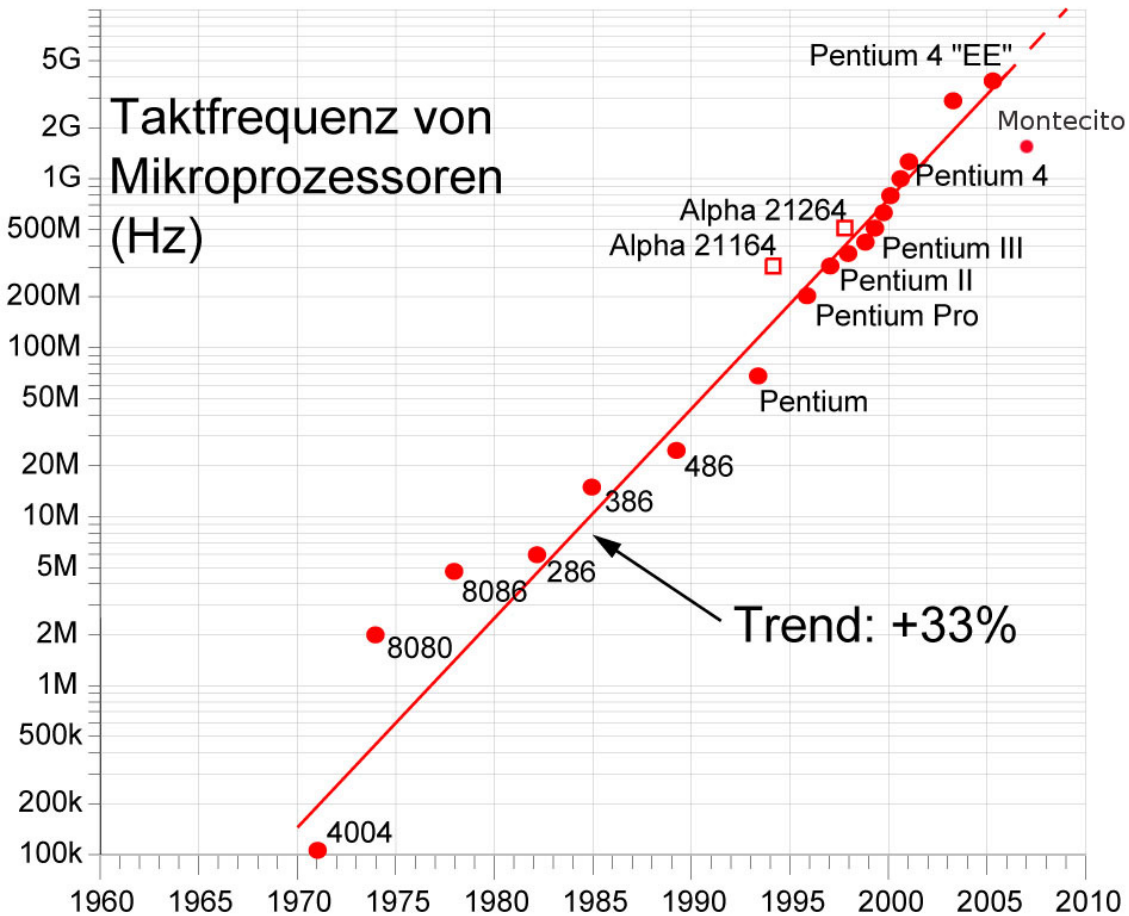
TECHNISCHE UNIVERSITÄT DARMSTADT



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

Taktfrequenz - Entwicklung

TECHNISCHE UNIVERSITÄT DARMSTADT



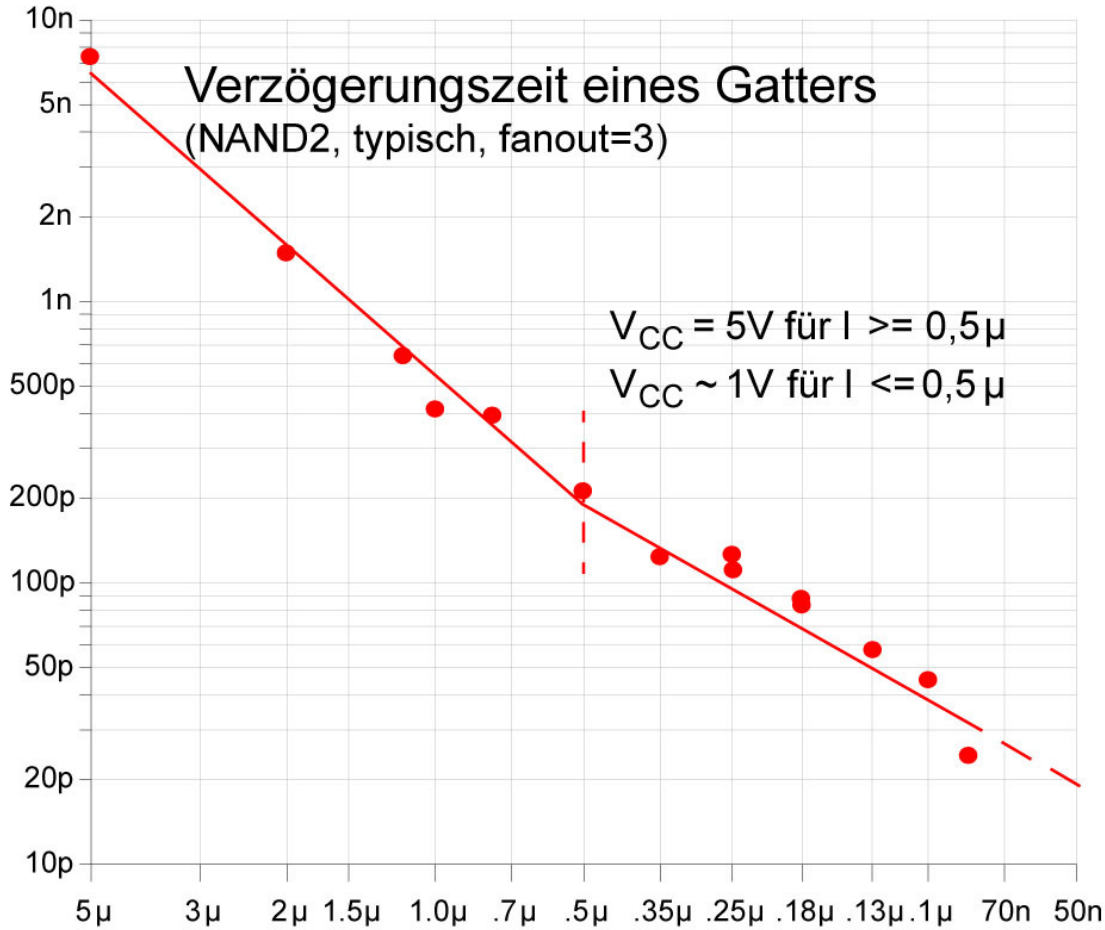
Taktfrequenz - Bedeutung und Grenzen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Leistungssteigerung wurde lange Zeit, durch erhöhen der Taktfrequenz erreicht. Aktuell liegt der Prozessortakt vieler Mikroprozessoren bei 3.x GHz.
  - 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: 2-12 CPU-Kerne von Intel und AMD verfügbar), Cell B. E. Prozessor von IBM
  - Massiv parallele Systeme mit mehreren tausend Prozessoren, Spezialarchitekturen z. B. Vektorrechner

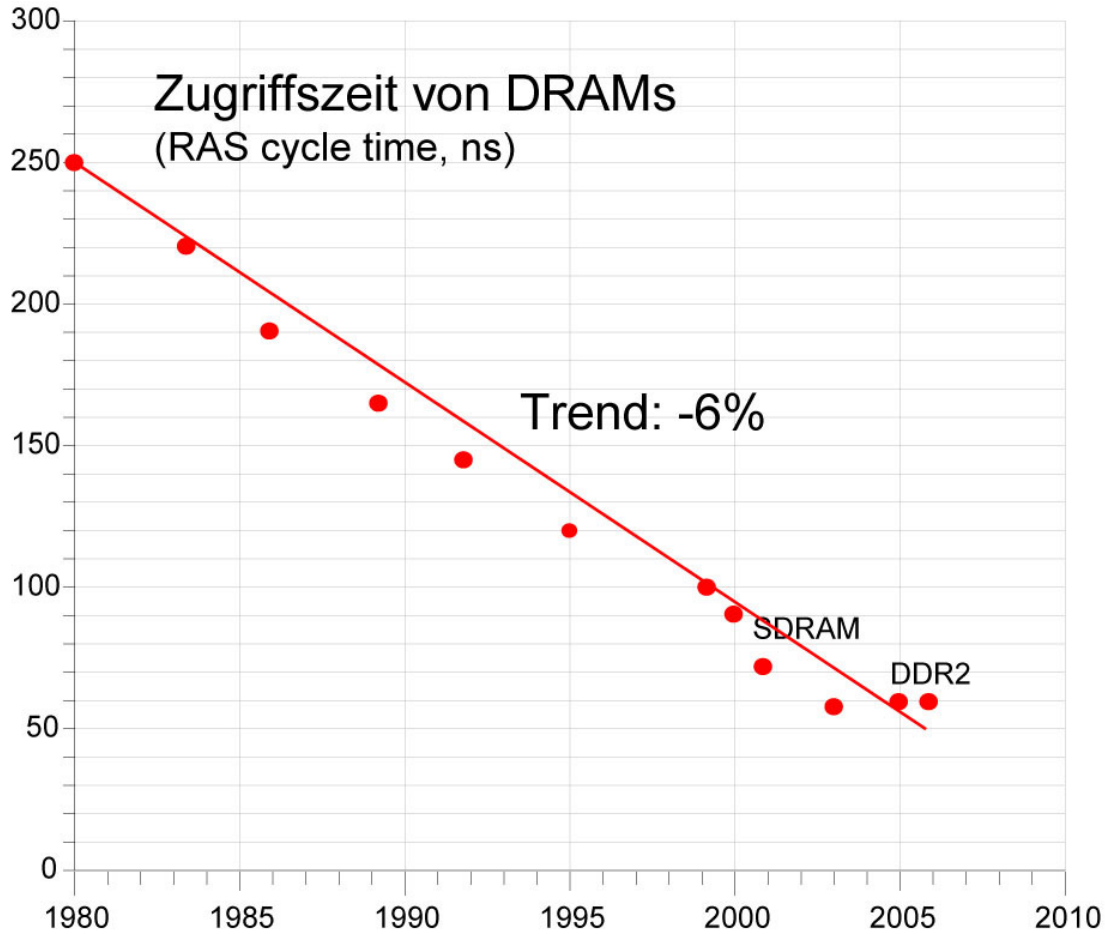
Gatter-Laufzeit

TECHNISCHE UNIVERSITÄT DARMSTADT



Speicher-Zugriffszeit

TECHNISCHE UNIVERSITÄT DARMSTADT



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

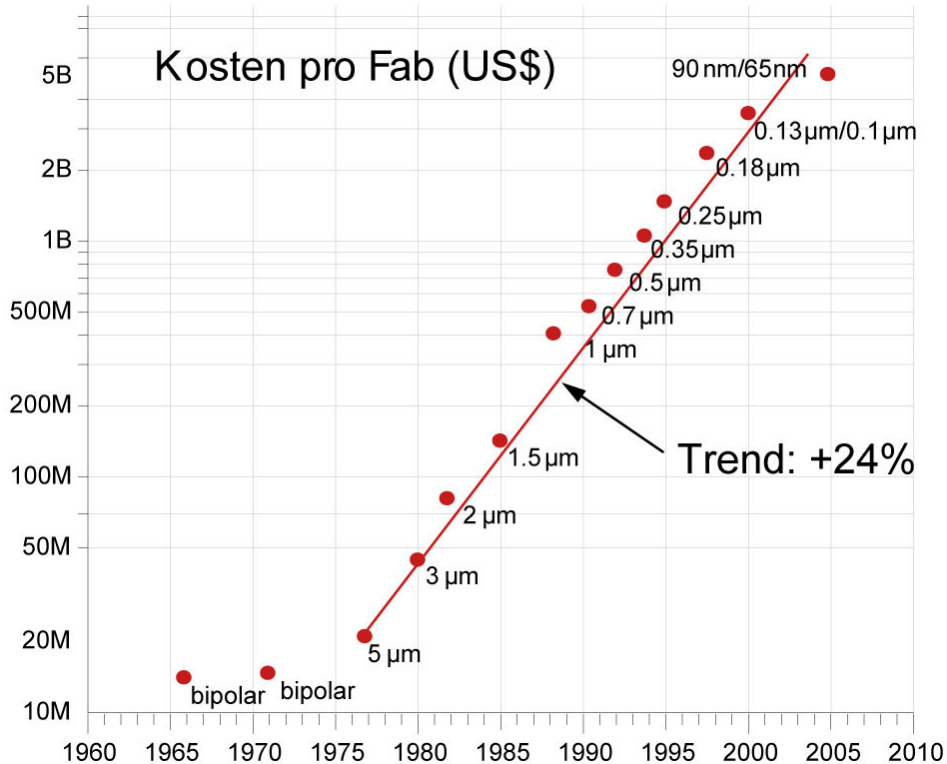
Zusammenfassung der technischen Aspekte

TECHNISCHE UNIVERSITÄT DARMSTADT

- +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!)

Kostenentwicklung I

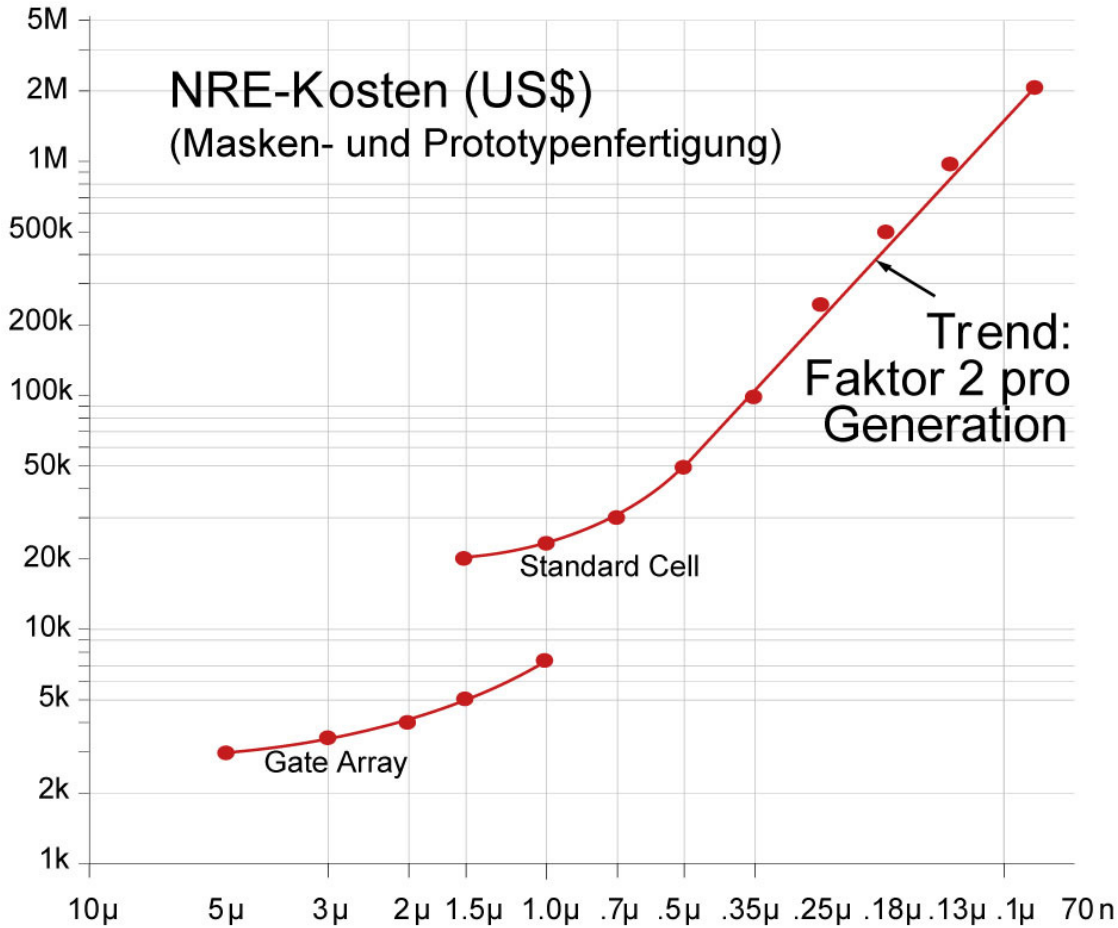
TECHNISCHE UNIVERSITÄT DARMSTADT



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

Kostenentwicklung II

TECHNISCHE UNIVERSITÄT DARMSTADT



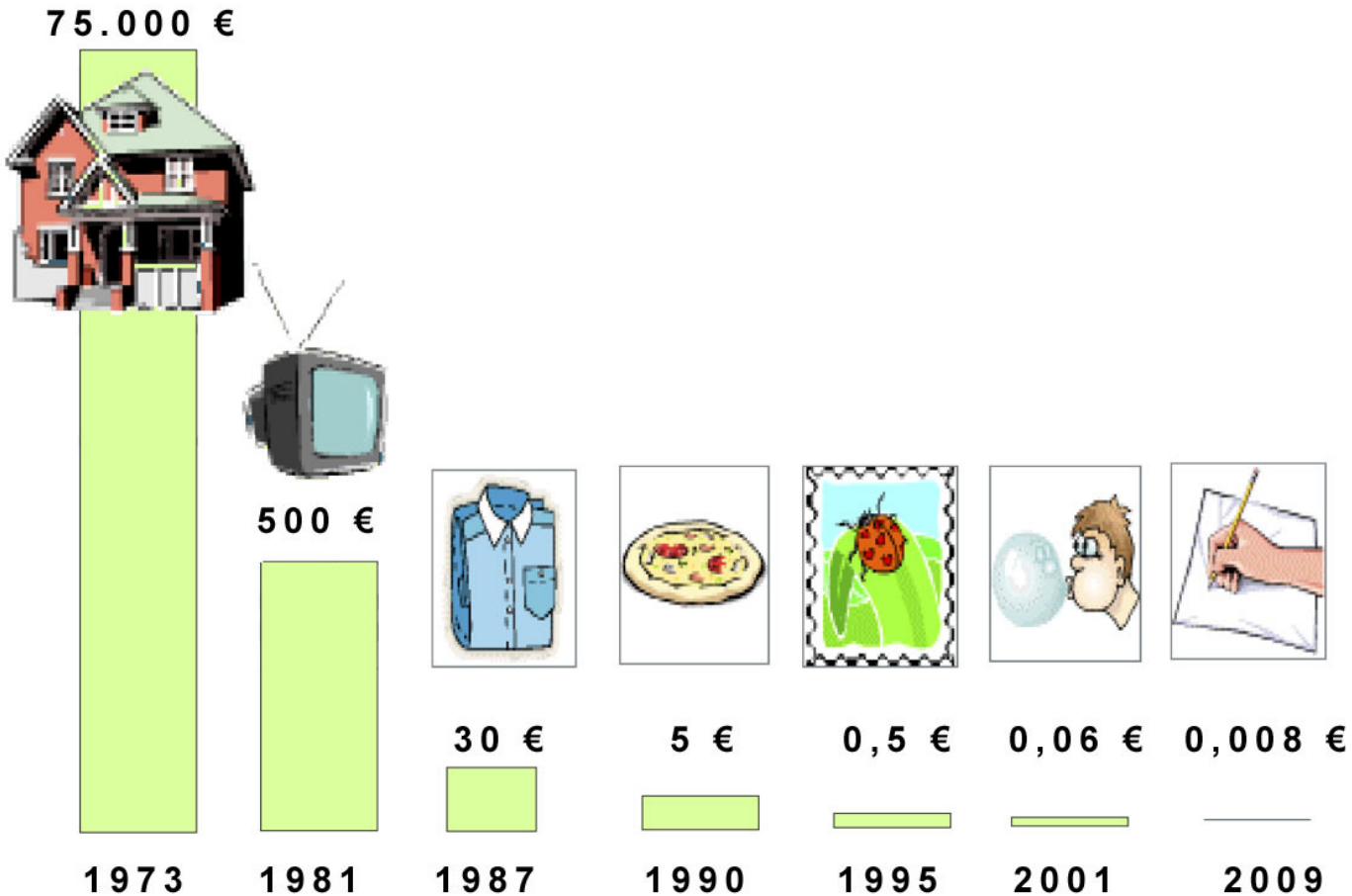
Einsparmöglichkeiten

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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 / mm^2
  - 65nm: EUR 7.500 / mm^2
  - 28nm: EUR 12.000 / mm^2
  - 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

TECHNISCHE UNIVERSITÄT DARMSTADT



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Hardware-Entwurfstechniken

Hardware-Entwurf

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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?

Lösungsansätze

TECHNISCHE UNIVERSITÄT DARMSTADT

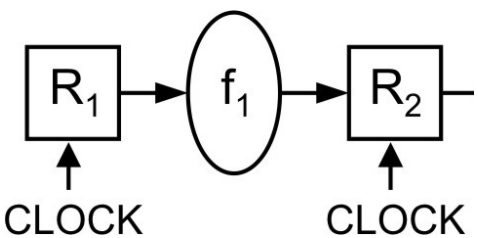
- Abstraktere Vorgehensweisen
- Beschreibe
  - ... nicht mehr einzelne Transistoren
  - ... sondern komplette Systeme
- Vergleichbar Software-Entwicklung
  - ... statt Assembler
  - ... Beschreibung von Systemen als interagierende Komponenten (service-oriented architectures)
- Mittel der Wahl: (Hardware)-Beschreibungssprachen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

Entwurfsebenen 1
Unterschiedlicher Abstraktionsgrad

TECHNISCHE UNIVERSITÄT DARMSTADT

- **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

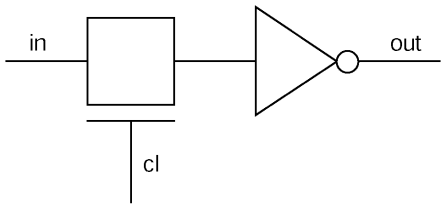
```
always @(posedge CLOCK)
   R2 <= f1(R1);
```



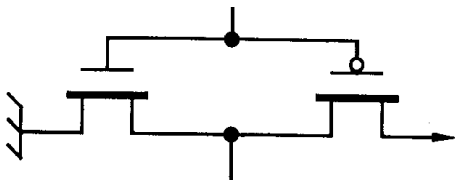
Entwurfsebenen 2
Unterschiedlicher Abstraktionsgrad

TECHNISCHE UNIVERSITÄT DARMSTADT

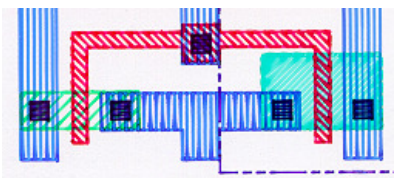
- **Logik- oder Gatterebene**
  Netze aus Gattern, Flip-Flops, etc.



- **Transistorebene**
  Elektrischer Schaltplan



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



Verhaltensebene

TECHNISCHE UNIVERSITÄT DARMSTADT

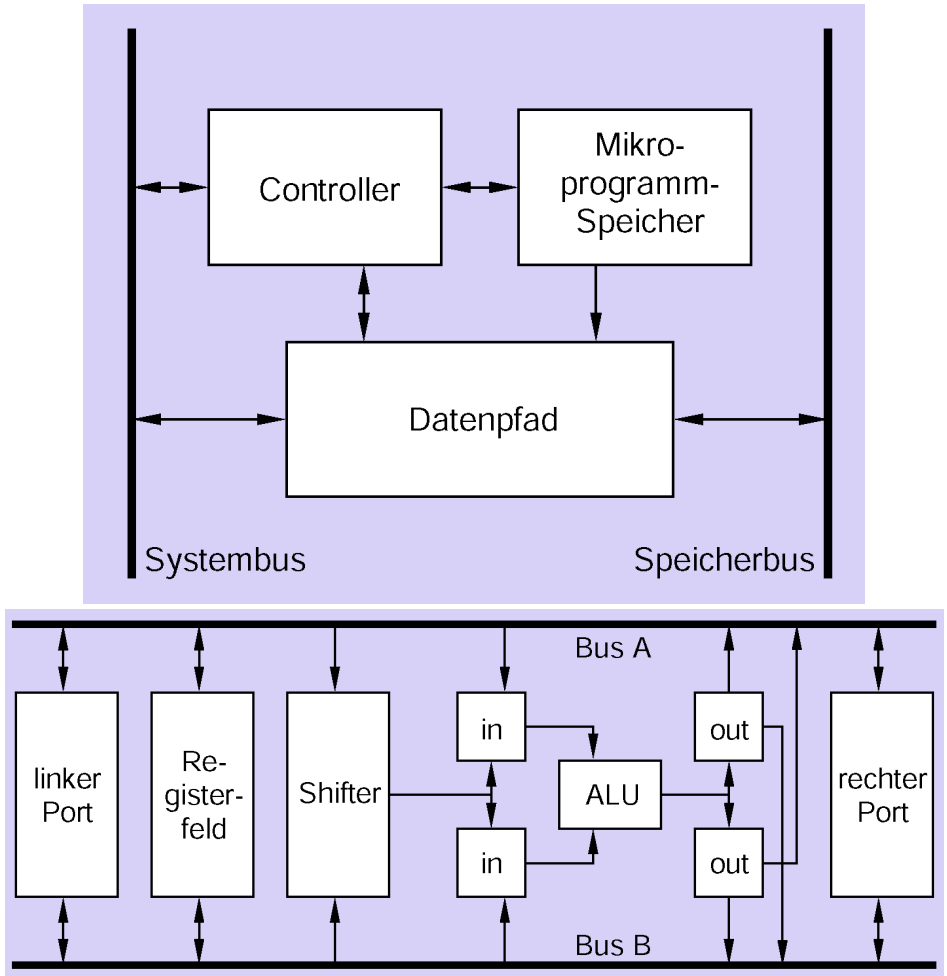
- Verilog HDL unterstützt auch die Beschreibung des Verhaltens.

```
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

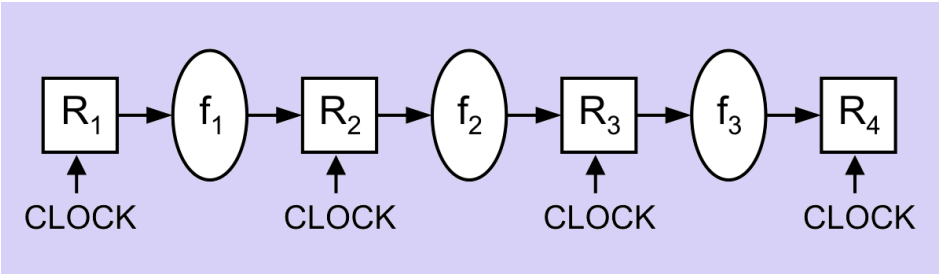
Systemebene
Komponenten eines Mikrokontrollers

TECHNISCHE UNIVERSITÄT DARMSTADT



Register-Transfer-Ebene (RTL)

TECHNISCHE UNIVERSITÄT DARMSTADT

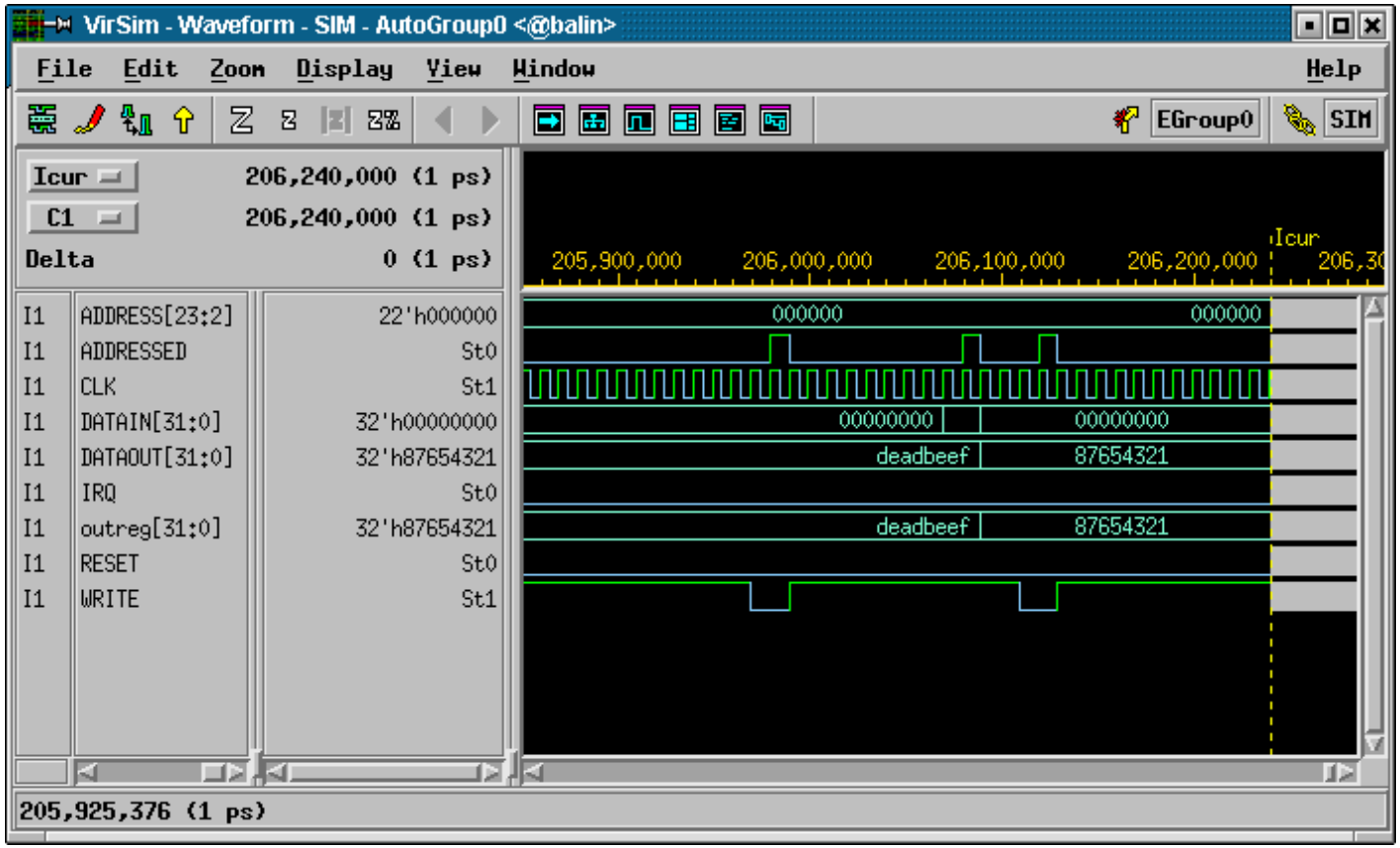



```
always @(posedge CLOCK) // mit jeder steigenden
 begin                  // Taktflanke
   R2 <= f1(R1);        // Register-Transfer von
   R3 <= f2(R2);        // Ri durch fi nach Ri+1
   R4 <= f3(R3);
 end
```

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

RTL-Simulation

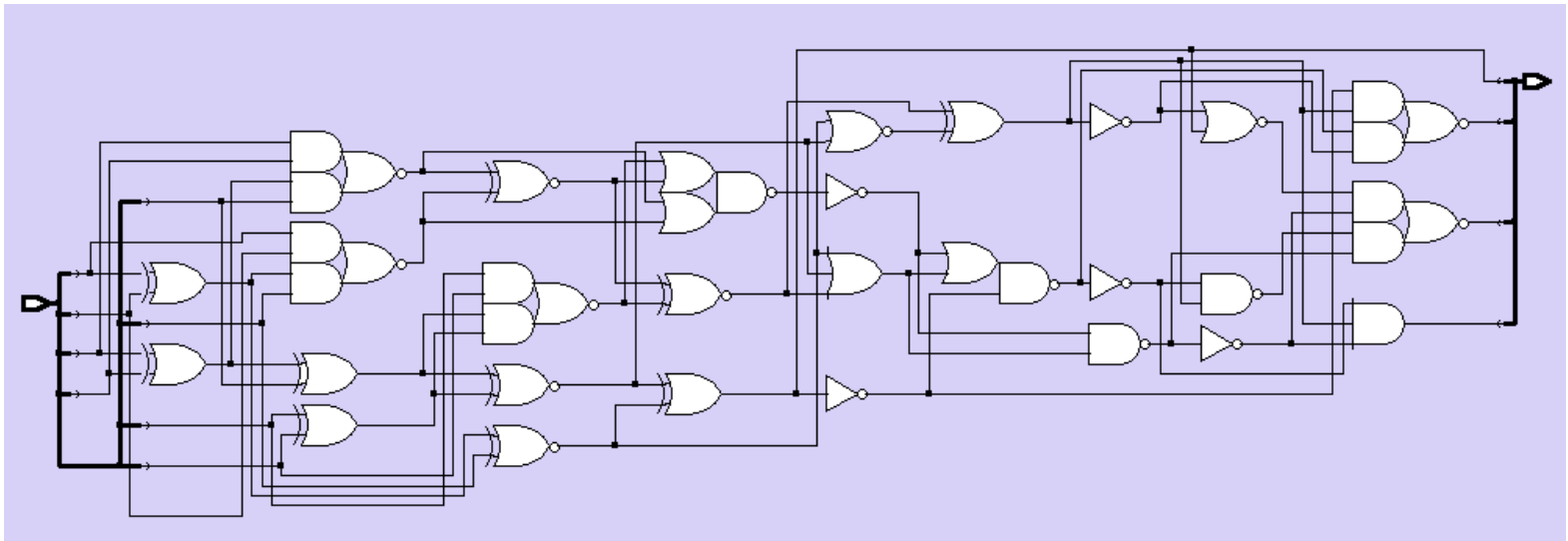
TECHNISCHE UNIVERSITÄT DARMSTADT



- Digitalsimulation noch ohne reale Verzögerungszeiten
- Alternativ auch Textausgabe möglich

Logik- oder Gatter-Ebene

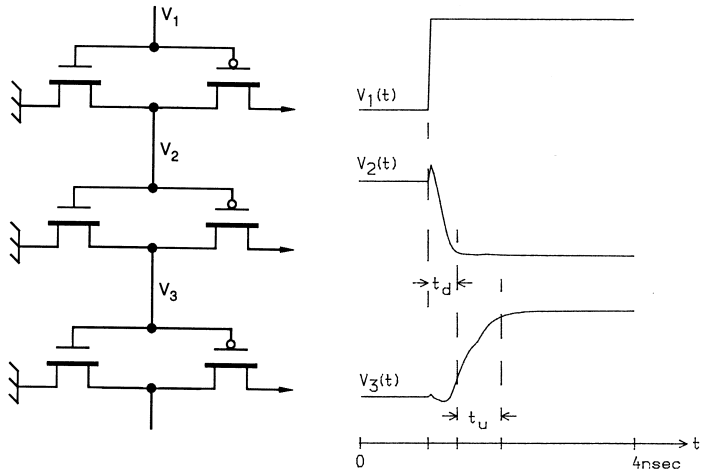
TECHNISCHE UNIVERSITÄT DARMSTADT



- 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, ...

Transistor- oder Schaltkreisebene

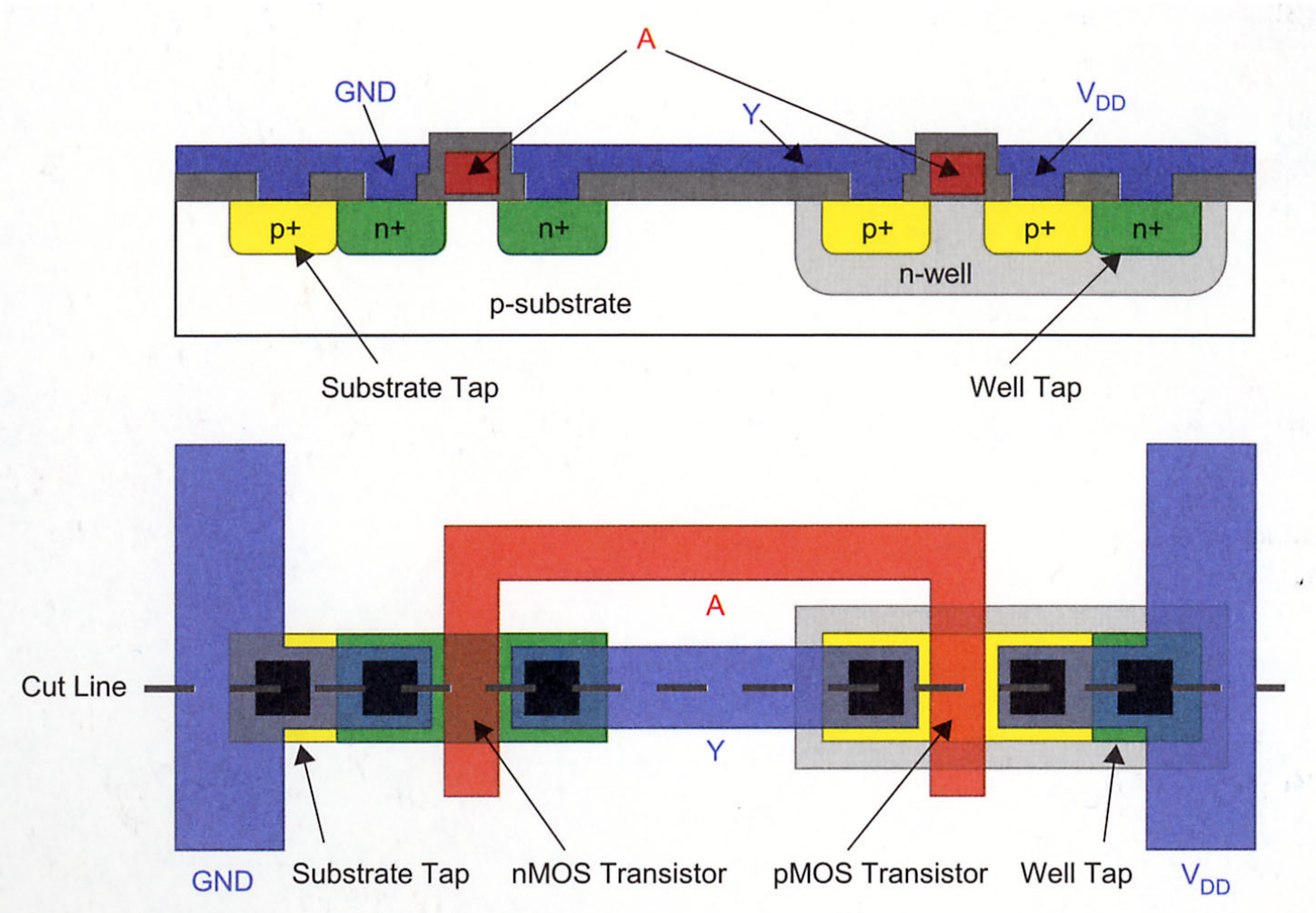
TECHNISCHE UNIVERSITÄT DARMSTADT



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

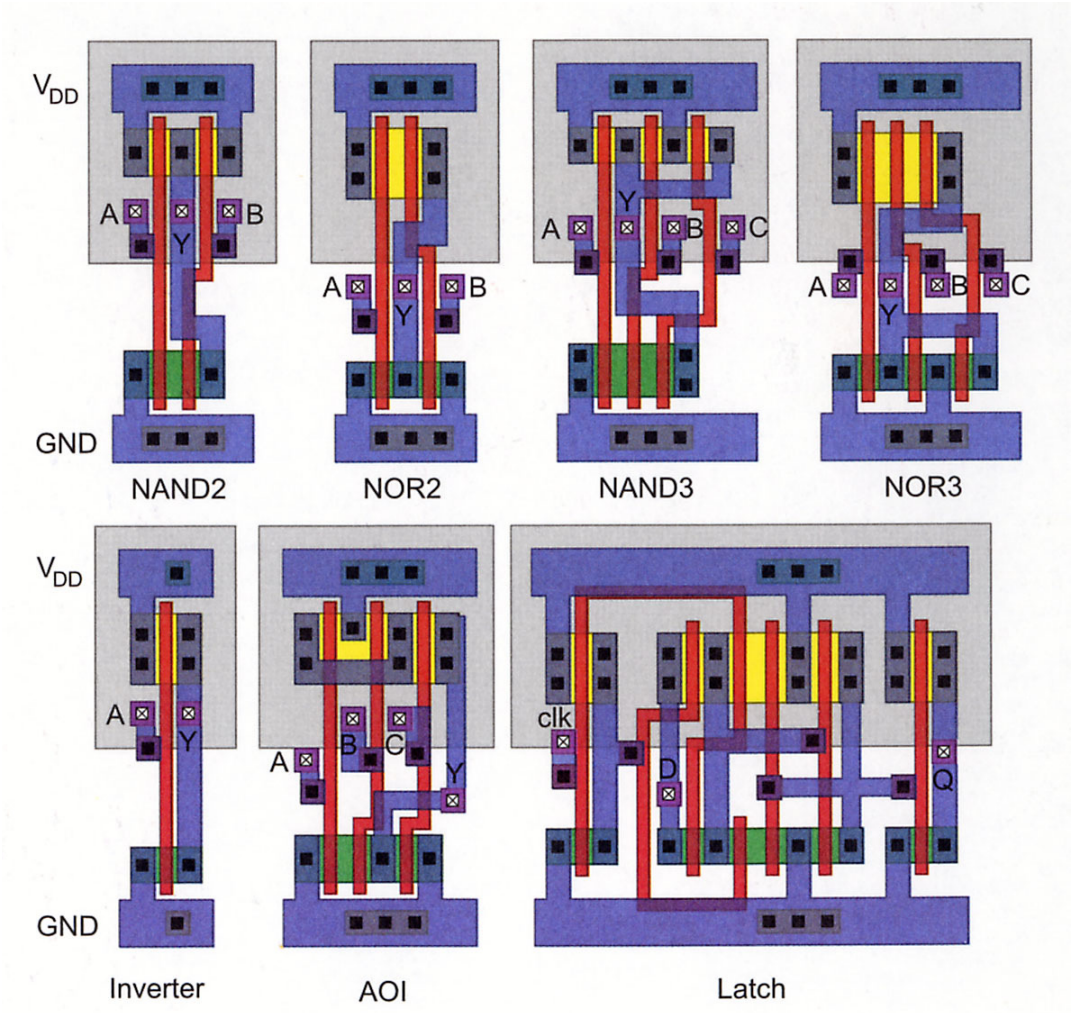
Layout-Ebene
Geometrien der Transistoren und Leitungen

TECHNISCHE UNIVERSITÄT DARMSTADT



Layout-Ebene
Einige Basisgatter (=Zellen)

TECHNISCHE UNIVERSITÄT DARMSTADT



Layout-Ebene

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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)

Fundamentale Entwurfsmethoden - Abstraktion

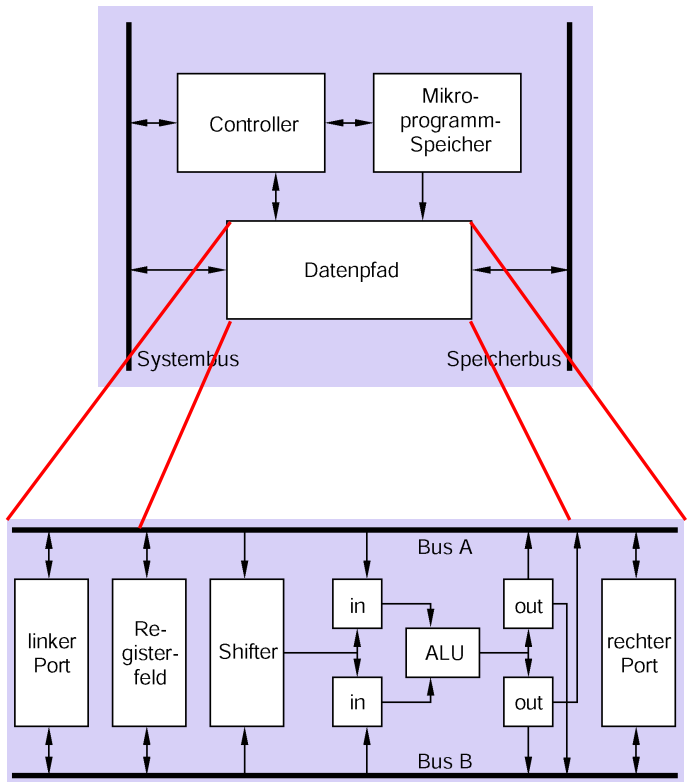
TECHNISCHE UNIVERSITÄT DARMSTADT

- 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

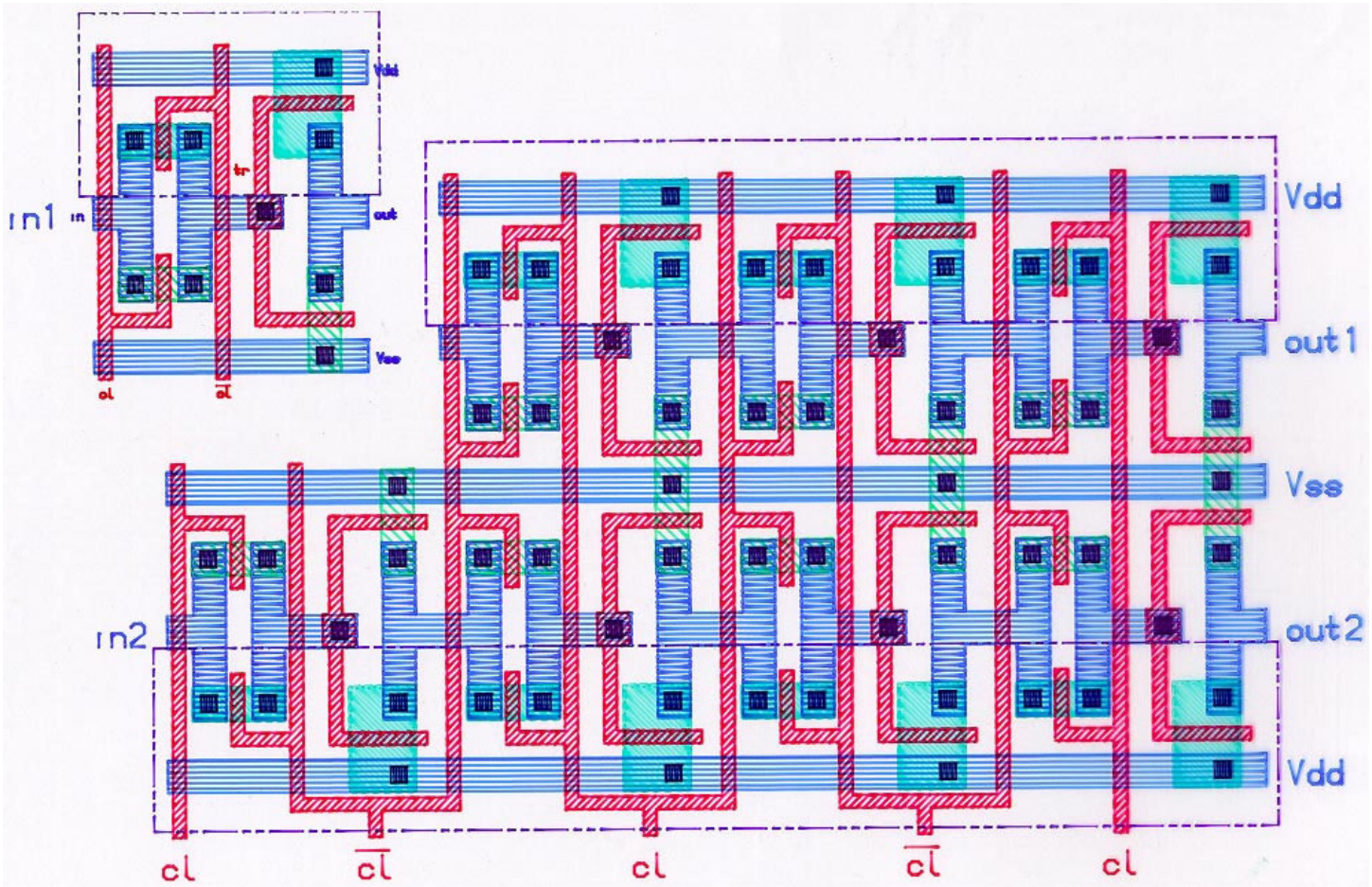
TECHNISCHE UNIVERSITÄT DARMSTADT

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



Reguläre Zerlegung
Gezielte Vervielfältigung von Komponenten

TECHNISCHE UNIVERSITÄT DARMSTADT



Übergeordnete Entwurfsmethoden und Entwurfsstrategien

TECHNISCHE UNIVERSITÄT DARMSTADT

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

Top-down-Entwurf 1

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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.

Top-down-Entwurf 2

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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.

Bottom-up-Entwurf

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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.

Meet-in-the-Middle-Entwurf

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Verilog: Kurzwiederholung

Beispiel: Einfache ALU

Verhaltensmodell

TECHNISCHE UNIVERSITÄT DARMSTADT

```
module alu (
  input  wire  [2:0] OPCODE,
  input  wire [31:0] A,
                     B,
  output reg  [31:0] RESULT
);


`define ADD      3'b000  // 0      // nur zur Übung:
`define MUL     'b001    // 1      // Konstanten auf
`define AND      3'o2    // 2      // verschiedene Arten
`define LOGAND 3'h3      // 3
`define MOD      4       // 4
`define SHL      3'b101  // 5


  always @ (*)
    case (OPCODE)
      `ADD:    RESULT = A + B;
      `MUL:    RESULT = A * B;
      `AND:    RESULT = A & B;
      `LOGAND: RESULT = A && B;
      `MOD:    RESULT = A % B;
      `SHL:    RESULT = A << B;
      default: $display ("Unimplemented Opcode: %d!", OPCODE);
    endcase

endmodule
```

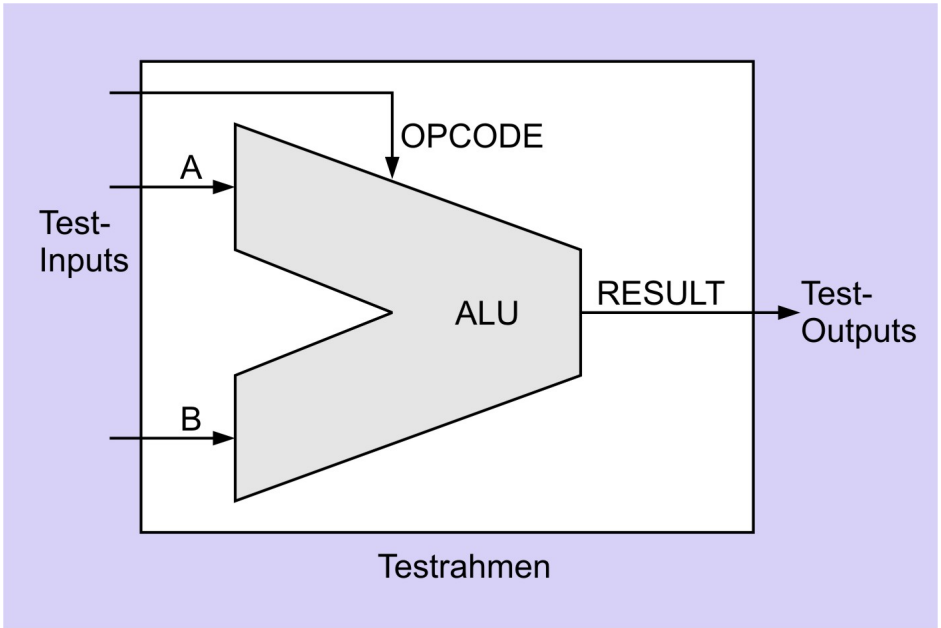
Wie ausprobieren?

TECHNISCHE UNIVERSITÄT DARMSTADT

- Modul `alu` macht freiwillig überhaupt nichts
- Der Simulator prüft quasi nur die Syntax
- Lösung:
  - Von außen Daten an Moduleingänge anlegen
  - Sogenannte Stimuli
  - Dann beobachten, wie sich Modulausgänge verhalten
- Analog zu Unit Tests im Software-Bereich
  - JUnit etc.

Testrahmen

TECHNISCHE UNIVERSITÄT DARMSTADT



- Saubere Trennung von
  - Prüfling (device under test, DUT)
  - Erzeugung von Eingabedaten
  - Auswertung der Ausgabedaten

Testrahmen für die einfache ALU

TECHNISCHE UNIVERSITÄT DARMSTADT

```
module test;

  reg     [2:0] OPCODE; // Zuweisungsziele für Eingabedaten (Variablen)
  reg    [31:0] A,
                B;
  wire   [31:0] RESULT;  // Stück Draht (zum Lesen der Ausgabe)

`define ADD         0
`define MUL         1
`define AND         2
`define LOGAND      3
`define MOD         4
`define SHL         5

  alu AluDUT (OPCODE, A, B, RESULT); // ALU-Instanz

   initial  begin                          // Test-Inputs
     $display ("Simulation beginnt...");
     OPCODE = `ADD; A = 3; B = 2; #1; // <- Zeit vergehen lassen
     OPCODE = `SHL; A = 3; B = 2; #1;
     $display ("Simulation endet.");
     $finish;
   end

   always @ (RESULT)                    // Test-Outputs
     $display ("OPCODE = %d, A = %d, B = %d: RESULT = %d",
       OPCODE, A[5:0], B[5:0], RESULT[5:0]);

endmodule
```

```
Simulation beginnt...
OPCODE =  0, A =   3, B =   2: RESULT =   5
OPCODE =  5, A =   3, B =   2: RESULT =  12
Simulation endet.
```

Verhalten

TECHNISCHE UNIVERSITÄT DARMSTADT

- Abbildung von Eingaben auf Ausgaben
- “was”, nicht “wie”
- Realisierung nicht von außen sichtbar (black box)
- Zur Modellierung reicht häufig ein einzelner `always`-Block

Struktur

TECHNISCHE UNIVERSITÄT DARMSTADT

- Beschreibe Einheit als
  - Untereinheiten
  - Verbindungen
- Im Extremfall
  - Keine `always` oder `initial`-Blöcke
  - Nur Modulinstanziierungen

Exkurs: Verbindungen über Port-Namen

TECHNISCHE UNIVERSITÄT DARMSTADT

Verbindung über Port-Reihenfolge

```
module topmod;
  wire [4:0] v;
  wire a,b,c,w;

  modB b1 (v[0], v[3], w, v[4]);
endmodule

module modB (wa, wb, c, d);
  inout wa, wb;
  input c, d;

  ...
endmodule
```

Verbindungen über Port-Namen

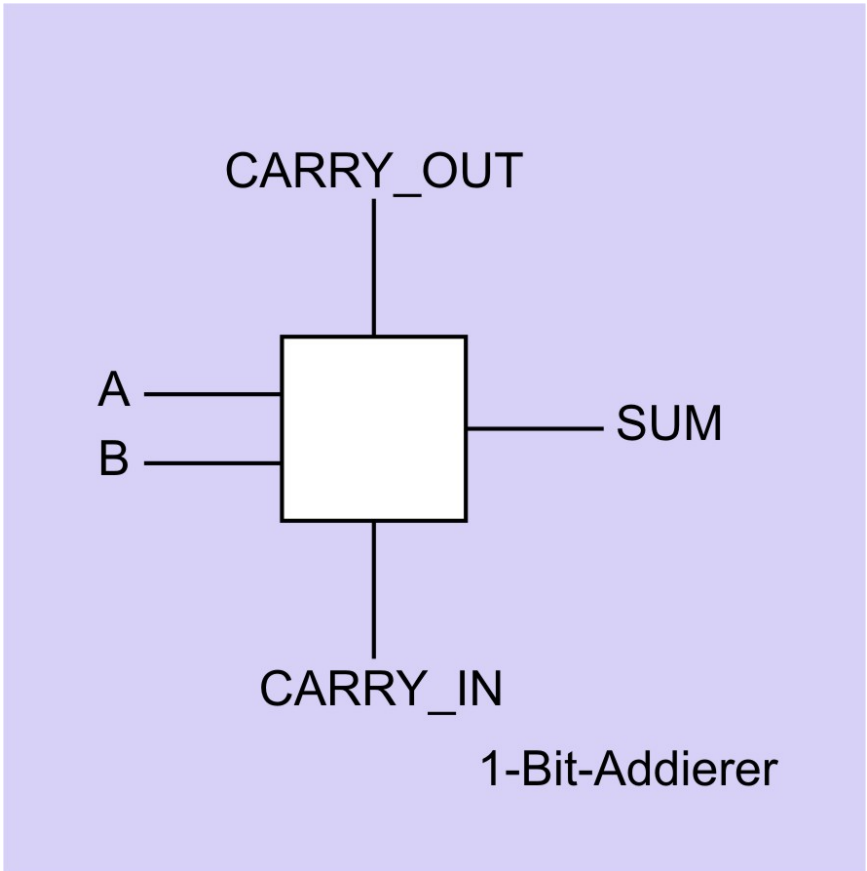
```
module topmod;
  wire [4:0] v;
  wire a,b,c,w;

  modB b1 (.wb(v[3]),.wa(v[0]),.d(v[4]),.c(w));
endmodule
...
```

- Unterschiedliche Reihenfolge
- Nichtangeschlossene Ports

In den Folien aus Platzgründen üblicherweise nicht gemacht, aber sinnvoll in der Praxis!

Beispiel: 1b-Addierer

TECHNISCHE UNIVERSITÄT DARMSTADT

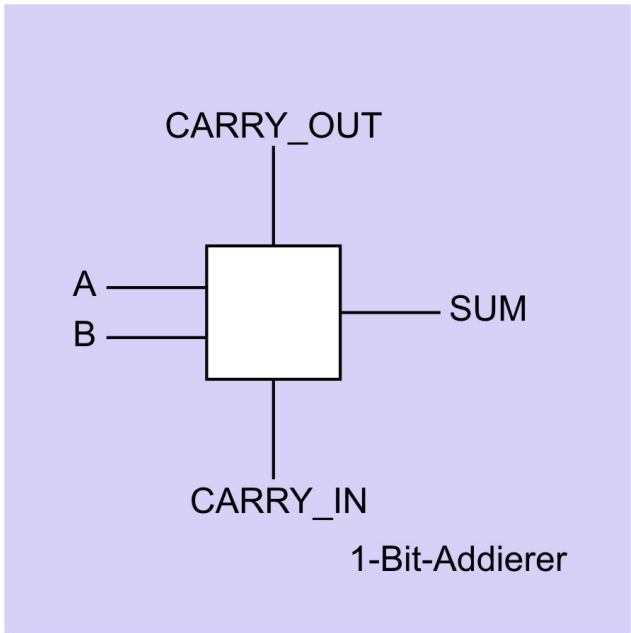


Verhaltensbeschreibung des 1b-Addierers

Konkreter Aufbau aus Gattern interessiert hier nicht

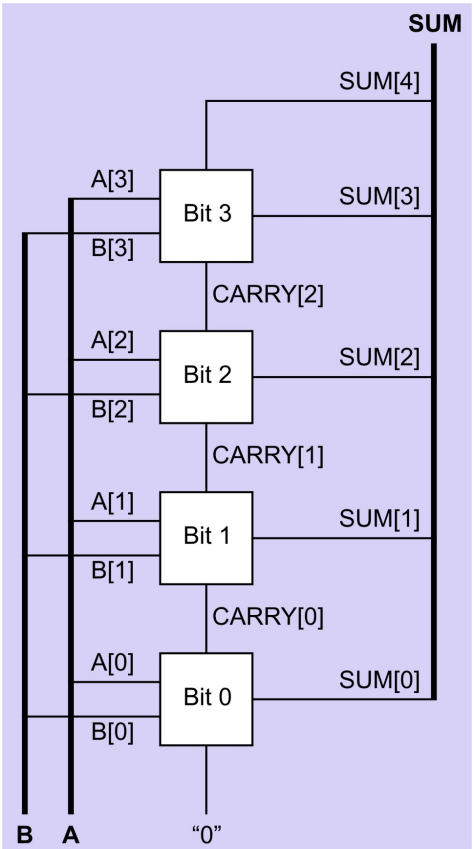
TECHNISCHE UNIVERSITÄT DARMSTADT

```
module one_bit_adder(
  input       A,        // 1-Bit-Wires per Default
              B,
              CARRY_IN,
  output reg  SUM,
              CARRY_OUT
);

  // Verhalten des one_bit_adder
  always @ (*)
    {CARRY_OUT, SUM} = A + B + CARRY_IN;

endmodule // one_bit_adder
```



Struktur eines 4b-Addierers in Ripple-Carry-Technik

Aufgebaut aus 1b-Addierern




```
module four_bit_adder(
  input  wire [3:0]  A,
                     B,
  output wire [4:0]  SUM
);


  wire [2:0]  CARRY;


  // Struktur des four_bit_adder
  one_bit_adder Bit0 (A[0],  B[0],       1'b0, SUM[0], CARRY[0]);
  one_bit_adder Bit1 (A[1],  B[1],  CARRY[0], SUM[1], CARRY[1]);
  one_bit_adder Bit2 (A[2],  B[2],  CARRY[1], SUM[2], CARRY[2]);
  one_bit_adder Bit3 (A[3],  B[3],  CARRY[2], SUM[3], SUM[4] );

endmodule // four_bit_adder
```

Synthese von regulären Strukturen

Makros

TECHNISCHE UNIVERSITÄT DARMSTADT

- Ähnlich C-Präprozessor
- Simple Textersetzung, keine Typprüfung ($\longrightarrow$ Bluespec)
- Über Modulgrenzen hinweg gültig bis zum Programmende

```
module module_1;

‘define TEXT "Hallo"
‘define TIMES 3

  reg [2:0] COUNTER;

  initial
    for (COUNTER = 1; COUNTER <= ‘TIMES; COUNTER = COUNTER + 1)
      $display (‘TEXT);

endmodule


module module_2;
  reg [2:0] COUNTER;

  initial
    for (COUNTER = 1; COUNTER <= ‘TIMES; COUNTER = COUNTER + 1)
      $display (‘TEXT);

endmodule
```

Modulparameter mit `parameter` und `defparam`

TECHNISCHE UNIVERSITÄT DARMSTADT

- Übergebe Konstanten in eine Modulinstanz
  - `parameter` bei der Moduldefinition
  - `defparam` bei der Instanziierung

```
module counter #(
  parameter Width = 8
) (
  input  wire                  CLOCK,
  output reg    [Width-1:0]    COUNT
);

   initial
     COUNT = 0;
  always @(posedge CLOCK)
     COUNT = COUNT + 1;
endmodule // counter

module main;
  defparam Counter1.Width = 3; // Parameter explizit definiert
  wire [Counter1.Width-1:0]  C1;
  wire [3:0]                 C2;
  reg                        CLOCK;
 ...
// Takterzeugung & $display C1, C2 weggelassen
 ...
  counter      Counter1(CLOCK, C1);
  counter #(4) Counter2(CLOCK, C2); // Parameter bei Instanzierung

endmodule // main
```

```
  0: C1=0 C2= 0
 10: C1=1 C2= 1
 30: C1=2 C2= 2
 50: C1=3 C2= 3
 70: C1=4 C2= 4
 90: C1=5 C2= 5
110: C1=6 C2= 6
130: C1=7 C2= 7
150: C1=0 C2= 8
170: C1=1 C2= 9
190: C1=2 C2=10
210: C1=3 C2=11
230: C1=4 C2=12
250: C1=5 C2=13
270: C1=6 C2=14
290: C1=7 C2=15
310: C1=0 C2= 0
```

Lokale Parameter mit `localparam`

TECHNISCHE UNIVERSITÄT DARMSTADT

- Falls Parameter nicht von außen überschrieben werden dürfen
- Sonst Verhalten wie `parameter`

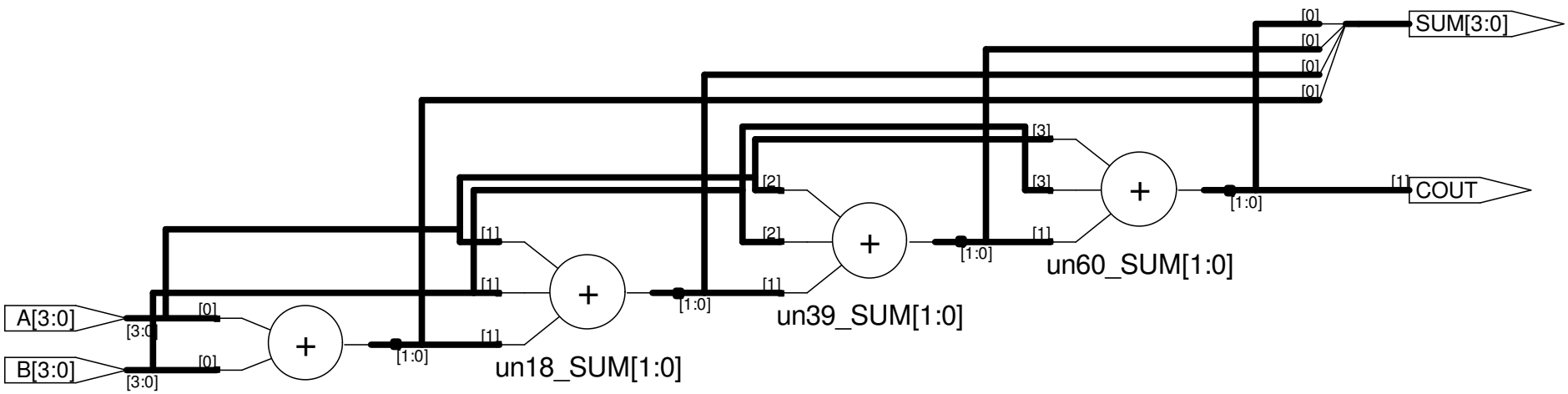
```
module RAM
 #(parameter ASIZE=10, DSIZE=8)
  (inout [DSIZE-1:0] data,
   input [ASIZE-1:0] addr,
   input             en, rw_n);

   // Speichertiefe ist 2**(ASIZE)
   localparam MEM_DEPTH = 1<<ASIZE;

   reg [DSIZE-1:0] mem [0:MEM_DEPTH-1];

    ...

endmodule
```

Synthese von `for`-Anweisungen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Nicht als sequentielle Schleife
  - Wie in normaler Programmiersprache
- Stattdessen: Räumlich "ausgerollt"
  - Parallele Abarbeitung

```
module unrolled_for #(parameter WIDTH=4)
                     (input        [(WIDTH-1):0] A, B,
                      output reg [(WIDTH-1):0] SUM,
                      output reg                COUT);
integer I;
reg C;

always @(*) begin
  C = 0;
  for (I = 0; I < WIDTH; I = I + 1) begin
    {C, SUM[I]} = A[I] + B[I] + C;
  end
  COUT = C;
end
```



`generate` / `genvar` mit Instanziierungen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
module top_pads2 (pdata, paddr, ...);
  input [15:0]  pdata;                    // pad data bus
  inout [31:0]  paddr;                    // pad addr bus
  wire  [15:0]  data;                     // data bus
  wire  [31:0]  addr;                     // addr bus
  wire          wr;                       // Schreibsignal (gibt addr auf paddr-Pads aus)
   ...
  genvar i;
   ...                                                  // Erzeugt Instanznamen
                                                        //   dat[0].i1 bis dat[15].i1
  generate for (i=0; i<16; i=i+1) begin: dat
    IBUF i1 (.O(data[i]),  .I(pdata[i]));  end
  endgenerate

  generate for (i=0; i<32; i=i+1) begin: adr            // Erzeugt Instanznamen
    BIDIR b1 (.N2(addr[i]),  .N1(paddr[i]),  .WR(wr)); end   //  adr[0].b1 to adr[31].b1
  endgenerate
endmodule
```

Geht aber noch einfacher ...

TECHNISCHE UNIVERSITÄT DARMSTADT

```
module top_pads3 (pdata, paddr, pctl1, pctl2,    pctl3,  pclk);
  input [15:0]  pdata;                          // pad data bus
  inout [31:0]  paddr;                          // pad addr bus
  wire  [15:0]  data;                           // data bus
  wire  [31:0]  addr;                           // addr bus
  wire          wr;                             // Schreibsignal (gibt addr auf paddr-Pads aus)
                                                //  Array-Instanznamen
                                                //  i[15] bis i[0]
  IBUF  i[15:0]  (.O(data), .pI(pdata));
  BIDIR b[31:0] (.N2(addr), .pN1(paddr), .WR(wr));          // Array-Instanznamen
                                                //  b[31] bis b[0]
endmodule
```

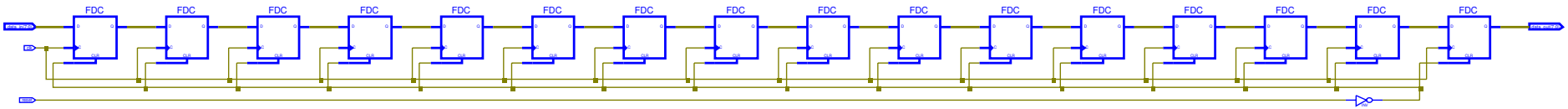
`generate` / `genvar`-Anweisung mit `assign` und `always`

TECHNISCHE UNIVERSITÄT DARMSTADT

```
module generated_arraya_pipeline(data_out,data_in,clk,reset);
 parameter width = 8;
 parameter length = 16;
 output [width-1:0] data_out;
 input  [width-1:0] data_in;
 input clk,reset;
 reg [width-1:0] pipe [0:length-1];
 wire [width-1:0] d_in [0:length-1];
 assign d_in[0] = data_in;
 assign data_out = pipe[length-1];
 generate
  genvar k;
  for (k=1;k<=length-1;k=k+1) begin: W
   assign d_in[k] = pipe[k-1]; end
 endgenerate
 generate
  genvar j;
  for (j=0;j<=length-1;j=j+1)
   begin: stage
    always @(posedge clk or negedge reset) begin
     if (reset == 0) pipe[j] <= 0; else pipe[j] <= d_in[j]; end
   end
 endgenerate
endmodule
```

Ergebnis der Generierung

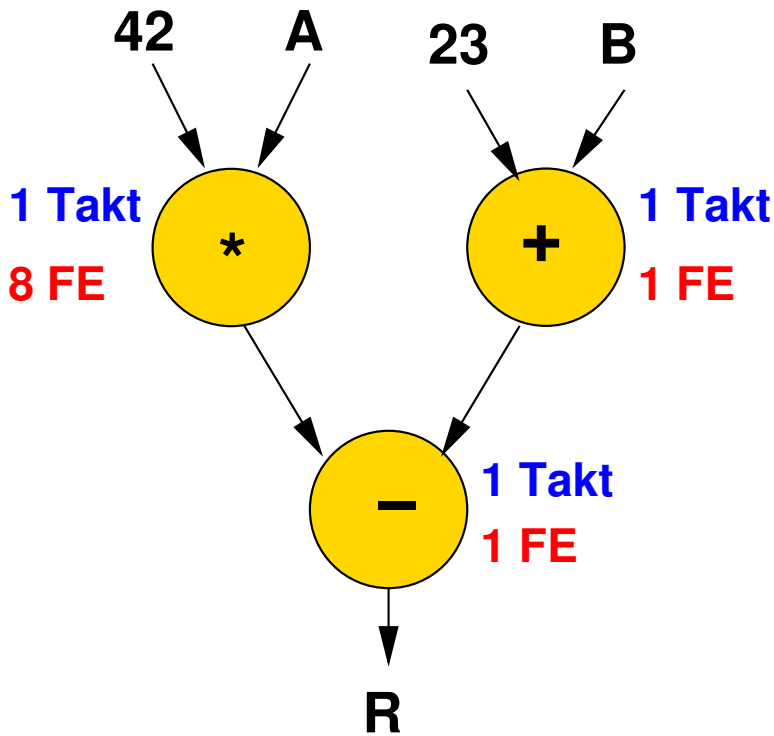
TECHNISCHE UNIVERSITÄT DARMSTADT



Pipelines in RTL

Pipeline mit parallelen Operatoren

Berechne $R = 42 \cdot A - (23 + B)$



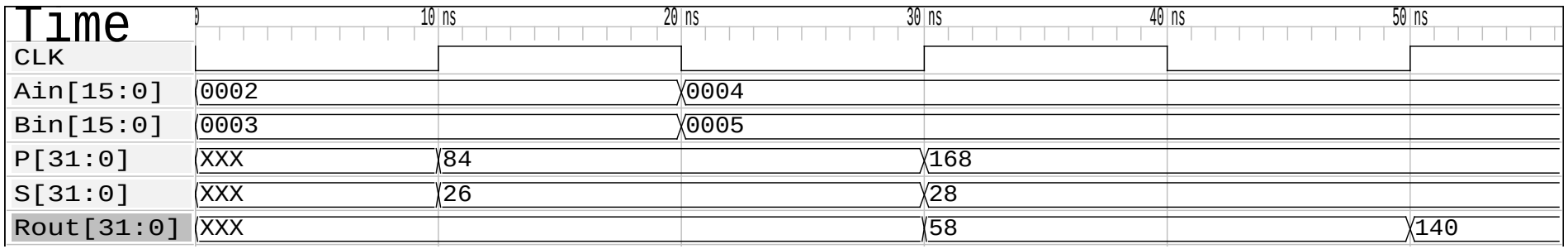

```
module compute
  (input              CLK,
   input       [15:0] A, B,
   output reg  [31:0] R);

  reg [31:0] P, S;

  always @(posedge CLK) begin
    P <= 42 * A;
    S <= 23 + B;
    R <= P - S;
  end

endmodule
```

- FE = Flächeneinheit
- Durchsatz: 1 Datum pro Takt
- Latenz: 2 Takte
- Fläche: 10 FE

Pipeline mit parallelen Operatoren

Signalverlaufsdiagramm

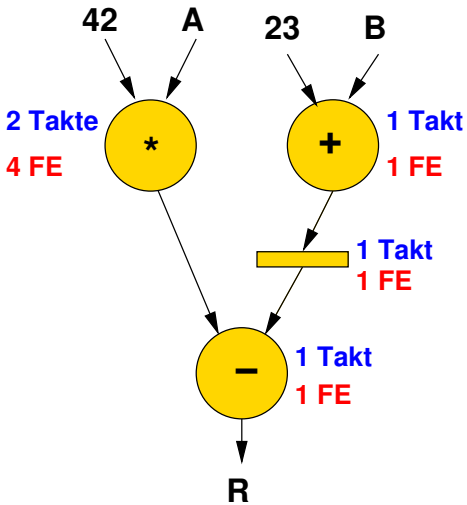
TECHNISCHE UNIVERSITÄT DARMSTADT



Pipeline mit parallelen Operatoren

Berechne $R = 42 \cdot A - (23 + B)$

TECHNISCHE UNIVERSITÄT DARMSTADT



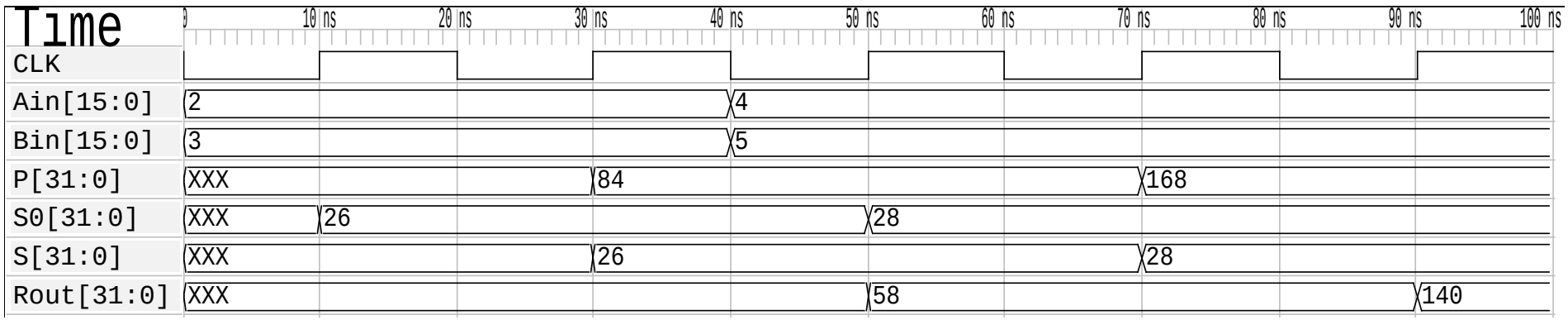

```
module compute
  (input               CLK,
   input       [15:0]  A, B,
   output reg  [31:0]  R);

  reg  [31:0]  S, S0;
  wire [31:0]  P;

  // Langsamer und kleiner Multiplizierer
  mul2 MUL2 (CLK, A, 42, P);

  always @(posedge CLK) begin
    S0 <= 23 + B;
    S <= S0;       // Verzögerungsregister
    R <= P - S;
  end

endmodule
```

- Multiplizierer nun langsamer und kleiner
  - Braucht zwei Takte
  - Verzögerungsregister zum Synchronisieren
- Durchsatz: 1 Datum pro 2 Takte
- Latenz: 3 Takte
- Fläche: 7 FE

Pipeline mit parallelen Operatoren

Signalverlaufsdiagramm

TECHNISCHE UNIVERSITÄT DARMSTADT



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Verilog: Details und fortgeschrittene Techniken

Verfeinerung: Vorzeichenbehaftung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Ohne Angaben: Vorzeichenlos
- Vorzeichenbehaftete Zahlen durch Schlüsselwort `signed`
  - `wire signed [7:0] op1`
  - `reg  signed [3:0] op2`
- Konstanten durch `s` vor Kennung für Basis
  - `4'she`: 4b breit, vorzeichenbehaftet, hexadezimal, Wert -2
  - `4'he`: 4b breit, vorzeichenlos, hexadezimal, Wert 14

Vorsicht beim Rechnen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Nur wenn alle Teile eines Ausdrucks `signed` sind, ist Ergebnis `signed`
- Wenn auch nur ein Teil `unsigned` ist, wird Ergebnis `unsigned`
- Unabhängig von Vorzeichenbehaftung des Zuweisungsziels
- Ergebnis wird abhängig von seiner Vorzeichenbehaftung auf Breite von Ziel aufgefüllt
  - Bei `unsigned`: Mit Nullbits
  - Bei `signed`: Durch Vorzeichenerweiterung
    - *sign extension*, TGDI

Beispiel: Vorzeichen- und Breitenerweiterung

TECHNISCHE UNIVERSITÄT DARMSTADT

```
module sign_test;
    reg          [2:0]  u1 =  1;       // bitmuster 001 = 1
    reg          [2:0]  u2 = -2;       // bitmuster 110 = 6
    reg signed   [2:0]  s1 =  1;       // bitmuster 001 = 1
    reg signed   [2:0]  s2 = -2;       // bitmuster 110 = -2
    reg          [4:0]  u;
    reg signed   [4:0]  s;
    reg          [4:0]  u3 = 4'he;     // bitmuster 01110 = 14
    reg signed   [4:0]  s3 = 4'she;    // bitmuster 11110 = -2

     initial  begin
       u = u1 + u2; s = s1 + s2;
       $display("u=u1+u2=%b+%b=%b␣s=s1+s2=%b+%b=%b", u1, u2, u, s1, s2, s);

       u = u1 + s2; s = s1 + u2;
       $display("u=u1+s2=%b+%b=%b␣s=s1+u1=%b+%b=%b", u1, s2, u, s1, u2, s);

       u = s1 + s2; s = u1 + u2;
       $display("u=s1+s2=%b+%b=%b␣s=u1+u2=%b+%b=%b", s1, s2, u, u1, u2, s);

       u = u3 + u1; s = s3 + s1;
       $display("u=u3+u1=%b+%b=%b␣s=s3+s
    end
endmodule
```

```
u=u1+u2=001+110=00111   s=s1+s2=001+110=11111
u=u1+s2=001+110=00111   s=s1+u2=001+110=00111
u=s1+s2=001+110=11111   s=u1+u2=001+110=00111
u=u3+u1=01110+001=01111 s=s3+s1=11110+001=11111
```

Typkonvertierungen

TECHNISCHE UNIVERSITÄT DARMSTADT

Implizite Konvertierung in vorzeichenlosen Typ

- Anwendung des Extraktionsoperators [*msb*:*lsb*]
- Auch bei Angabe des gesamten Wortes

```
reg signed [7:0] DATA;
...  = DATA[7:0];
```

ist die rechte Seite immer vorzeichenlos

Explizite

- `$signed(`*v*`)` konvertiert *v* in vorzeichenbehafteten Typ
- `$unsigned(`*v*`)` konvertiert *v* in vorzeichenlosen Typ

Was ist mit `integer`?

TECHNISCHE UNIVERSITÄT DARMSTADT

- Nicht für Synthese verwenden!
- Nur ungenau definiert
  - 32b oder 64b vorzeichenbehaftete Zahl
  - Hängt von CAD-Werkzeugen ab!
- Aber nützlich für
  - Simulation
  - Schleifenzähler für `for` etc.

Felder von Variablen

TECHNISCHE UNIVERSITÄT DARMSTADT

- `reg A[1:1000]` oder `reg A[1000:1]`
  - Feld von 1000 Variablen, jede 1b breit
  - `RESULT = A[500]`
- `reg [15:0] B [1:1000]`
  - Feld von 1000 Variablen, jede 16b breit
  - `RESULT = A[500][8]`
- `reg [15:0] B [1:100][1:200][1:300]`
  - Feld von 6.000.000 Variablen, jede 16b breit
  - `RESULT = A[99][156][223][7]`

Verilog Sprachstandard

TECHNISCHE UNIVERSITÄT DARMSTADT

Wenn man es genau wissen möchte:

- Sprache: Standard IEEE 1364-2005 “Verilog Language Reference Manual”
- Syntheseregeln: Standard IEEE 1364.1 / IEC 62142-2005 “Verilog register transfer level synthesis”

Aus dem TU Darmstadt-Netz (ggf. via VPN) über ULB aus der IEEE Literaturdatenbank Xplore abrufbar.

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Modellierung von Parallelität

`always` und `initial`-Blöcke

TECHNISCHE UNIVERSITÄT DARMSTADT

- Werden zueinander parallel ausgeführt (in beliebiger Reihenfolge)
- Werden im Inneren i.d.R. sequenziell ausgeführt
- Ausführung erfolgt ohne Unterbrechung
  - Falls keine Zeitkontrollanweisungen (`#` etc.) auftreten
- Eintrittsbedingungen mit `@` (gelesen: *at*)
- `always @(COUNTER)`: Bei Änderungen von `COUNTER`
- `always @(*)`: alle Lesevariablen eines Blockes
- Faustregel
  - `always`-Blöcke in Schaltungsteilen (synthetisierbar)
  - `initial`-Blöcke in Testmodulen (nicht synthetisierbar)

♯ *n*: *n* Zeiteinheiten warten

TECHNISCHE UNIVERSITÄT DARMSTADT

- Explizite Modellierung von Zeit
- Andere parallele Prozesse laufen weiter

```
module time_delay;
  reg DATA;

  always @(DATA)
    $display ("Zeit:␣%2.0f,␣DATA␣=␣%d", $time, DATA);

  initial  begin
    DATA = 0;
    #10;
    DATA = #10 1;
    #10;
  end

endmodule
```

```
Zeit:  0, DATA = 0
Zeit: 20, DATA = 1
```

Sieht einfach aus, Gemeinheiten liegen tiefer

TECHNISCHE UNIVERSITÄT DARMSTADT

```
module time_delay;
  reg DATA;

  always @(DATA)
    $display ("Zeit: %2.0f, DATA = %d", $time, DATA);

  initial  begin
    DATA = 0;
    DATA = 1;
  end

endmodule
```

```
Zeit:  0, DATA = 1
```

- Transition `DATA` $0 \rightarrow 1$ nicht sichtbar für `always`-Block
  - `initial`-Block läuft atomar ab

Extremfall

TECHNISCHE UNIVERSITÄT DARMSTADT

```
module time_delay;
  reg DATA;

  always @(DATA)
    $display ("Zeit:␣%2.0f,␣DATA␣=␣%d", $time, DATA);

  initial  begin
    DATA = 0;
    #0;
    DATA = 1;
  end

endmodule
```

```
Zeit:   0, DATA = 0
Zeit:   0, DATA = 1
```

- Transition nun sichtbar
  - `#` unterbricht Ausführung von `initial`-Block
  - Erlaubt Reaktion durch `always`-Block
  - Es vergeht aber keine Zeit!

Diskussion von ♯

Bisher im wesentlichen Trickserei

TECHNISCHE UNIVERSITÄT DARMSTADT

- # lässt sich nicht synthetisieren
- Hat nur Effekte während der Simulation
- Dort benutzt zur Erzeugung von Testsignalen
- Kenntnisse aber manchmal bei Fehlersuche nützlich

```
module gen_clock;
  reg CLOCK;

  always @(CLOCK)
    $display ("Zeit: %2.0f, CLOCK = %d", $time, CLOCK);

  always begin
    CLOCK = 0;
    #10;
    CLOCK = 1;
    #10;
  end

endmodule
```

```
Zeit:  0, CLOCK = 0
Zeit: 10, CLOCK = 1
Zeit: 20, CLOCK = 0
Zeit: 30, CLOCK = 1
Zeit: 40, CLOCK = 0
Zeit: 50, CLOCK = 1
... bis zum Stromausfall
```

Blockende Zuweisung =

TECHNISCHE UNIVERSITÄT DARMSTADT

- Wird immer zusammenhängend ausgeführt
- Auch wenn sie eine Zeitkontrolle #*n* enthält
- Nachfolgende Anweisungen starten erst nach Ende der blockenden Zuweisung
- Ablauf der blockenden Zuweisung
  1. Lese aktuelle Werte von Variablen und werte Ausdruck auf rechter Seite aus
  2. Warte evtl. mit # die angegebene Zeit ab
  3. Übernehme Wert in Zuweisungsziel auf linker Seite
  4. Mache mit nächster Anweisung weiter
- Benutzung
  - Zur Erzeugung von Stimuli in Simulation
  - In rein kombinatorischen Blöcken in der Synthese
    - Ohne `always @(posedge ...)`

Nichtblockende Zuweisung <=

TECHNISCHE UNIVERSITÄT DARMSTADT

- Wird immer in zwei Phasen getrennt ausgeführt
- Ablauf der nichtblockenden Zuweisung
  1. Lese aktuelle Werte von Variablen und werte Ausdruck auf rechter Seite aus, merke Ergebnis
  2. Mache sofort mit nächster Anweisung im Block weiter
  3. Am Ende des Blockes
     - Übernehme gemerkte Werte in Zuweisungsziele auf linker Seite
     - Falls Zeitkontrolle: Verzögere obige Zuweisung auch noch (benutzen wir aber nicht!)
- Benutzung
  - In allen sequentiellen Blöcken während der Synthese

Niemals = und <= an eine Variable in einem Block mischen!

Beispiel: Blockende Zuweisungen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
module blocking_1;

  reg A, B;

  always @(A,B)
    $display("A=%b␣B=%b", A, B);

   initial  begin
    A = 0;
    B = 1;

    A = B;
    B = A;
  end

endmodule
```

A=1 B=1

Ausführung nacheinander.

Beispiel: Nichtblockende Zuweisungen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
module blocking_2;

  reg A, B;

  always @(A,B)
    $display("A=%b␣B=%b", A, B);

   initial  begin
    A = 0;
    B = 1;

    A <= B;
    B <= A;
  end

endmodule
```

A=1 B=0

Getrennte Ausführung von Lesen und Schreiben.

Beispiel: Zeitverhalten

Bei blockenden und nicht-blockenden Zuweisungen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
module blocking_3;

  reg A, B, C, D, E, F;

  // blockende Zuweisungen
  initial  begin
    A = #10 1;
    B = #2 0;
    C = #4 1;
  end

  // nichtblockende Zuweisungen
  initial  begin
    D <= #10 1;
    E <= #2 0;
    F <= #4 1;
  end

  always @(A,B,C,D,E,F)
    $display (
      "t=%2.0f   A=%b B=%b C=%b   D=%b E=%b F=%b",
      $time, A, B, C, D, E, F);

endmodule
```

```
t= 0   A=x B=x C=x   D=x E=x F=x
t= 2   A=x B=x C=x   D=x E=0 F=x
t= 4   A=x B=x C=x   D=x E=0 F=1
t=10   A=1 B=x C=x   D=1 E=0 F=1
t=12   A=1 B=0 C=x   D=1 E=0 F=1
t=16   A=1 B=0 C=1   D=1 E=0 F=1
```

Zeiteinheiten mit `timescale`

TECHNISCHE UNIVERSITÄT DARMSTADT

- Was bedeutet `#1` überhaupt?
  - Sekunden? Stunden? Wochen?
- Zuordnung durch ‘`timescale`-Direktive
  - Am Anfang des Verilog-Modells
- Zwei Parameter
  1. Maß für 1 Zeiteinheit
     - 1, 10, 100
     - Einheit s, ms, us, ns, ps, oder fs
  2. Auflösung der Simulation
     - 1, 10, 100
     - Einheit s, ms, us, ns, ps, oder fs
     - Muß kleiner gleich Zeiteinheit sein!
     - Genauer $\rightarrow$ langsamer
- Bei RTL-Simulation nicht so kritisch
- Bei uns oft ausreichend:
  - `‘timescale 1 ns / 1 ns`
  - `‘timescale 1 ns / 10 ps`

Simulation von Parallelität

Parallelität

TECHNISCHE UNIVERSITÄT DARMSTADT

- In konventionellen Programmiersprachen wie z.B. Pascal, C
  - Anweisungen werden der Reihe nach bearbeitet
  - Programmzähler zeigt auf aktuelle Anweisung
  - Es gibt nur einen Kontrollfluß
- In HDLs und realen Schaltungen
  - Alle Komponenten arbeiten parallel
  - Z.B. kann eine Taktflanke eine Vielzahl von gleichzeitigen Aktionen auslösen
    - Modelliert durch parallele `always`-Blöcke

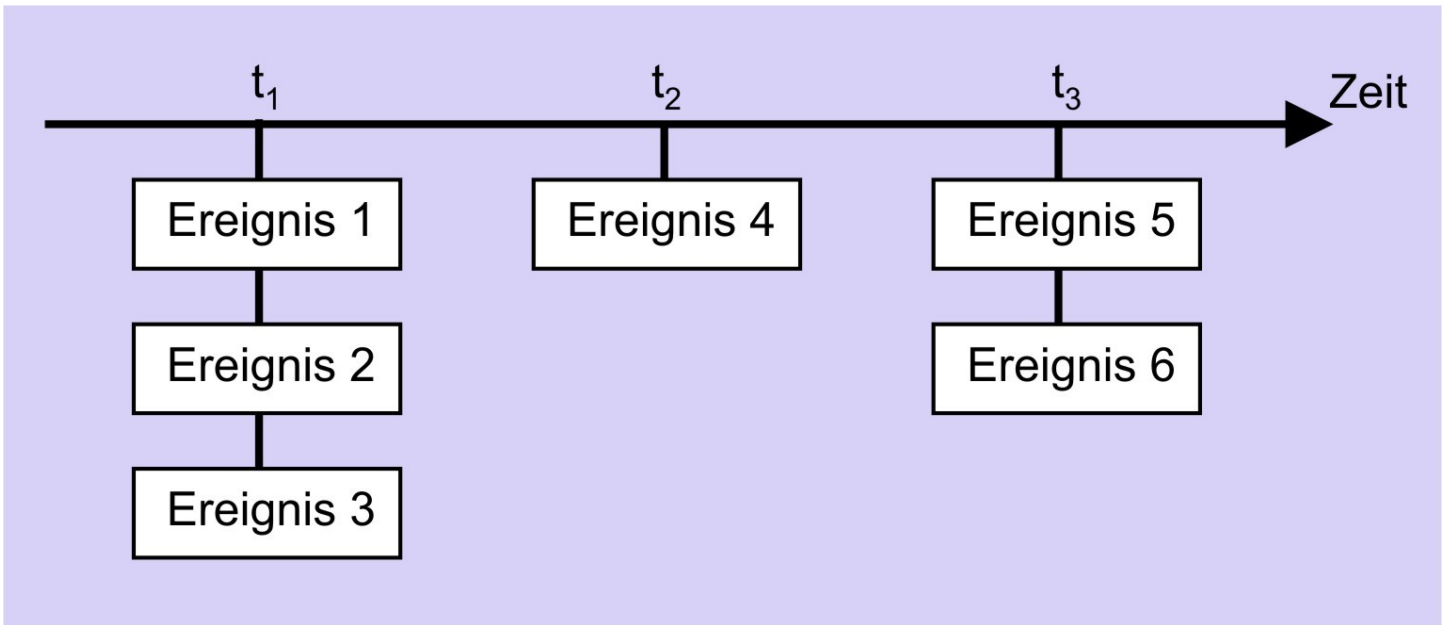
Modellierung von Parallelität in Verilog

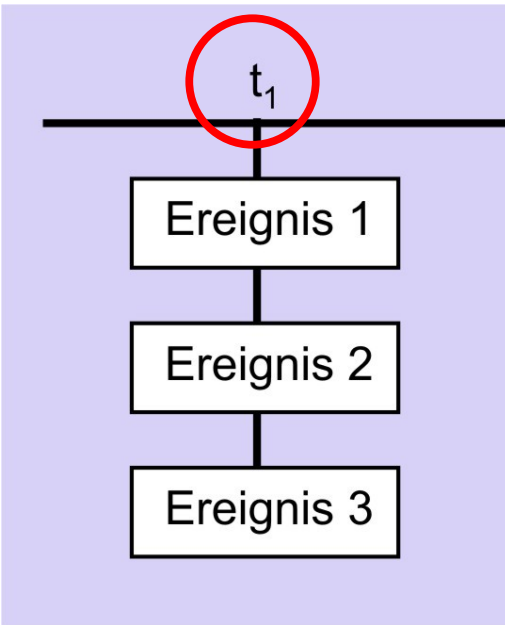
TECHNISCHE UNIVERSITÄT DARMSTADT

- Instanzen von Modulen
- `always`- und `initial`-Blöcke
- ständige Zuweisungen (*continuous assignments*)
- nichtblockende Zuweisungen
- Mischformen

Ereignisgesteuerte Simulation der Parallelität

TECHNISCHE UNIVERSITÄT DARMSTADT





- globale Simulations-Zeitpunkte t_1, t_2, ...
- ein oder mehrere Ereignisse sollen jeweils parallel ausgeführt werden
- Ereignis-Scheduler wählt eines zufällig aus
- wenn bei t_1 nichts mehr zu tun, gehe zu t_2 weiter

Parallelität

TECHNISCHE UNIVERSITÄT DARMSTADT

- Kein Verlass auf bestimmte Reihenfolge
  - Kann zwischen Simulatoren variieren
  - Kann auch durch Simulationsoptionen beeinflußt werden
- parallel = nicht-deterministisch
  - ein richtiges Ergebnis garantiert nicht allgemeine Korrektheit
  - exponenziell viele Ergebnisse möglich
- Unwägbarkeiten können durch Entwurfsstile reduziert werden
  - Synchrone Register-Transfer-Logik
  - Designer legt Zeitablauf explizit im Modell fest
  - Unterschiedliche Ereignisse finden in unterschiedlichen Takten statt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Systematischer Hardware-Entwurf

Systematischer Entwurf

TECHNISCHE UNIVERSITÄT DARMSTADT

- Verschiedenste Techniken
- Abstraktion
  - Modell
  - Darstellungen
- Schrittweise Verfeinerung

Steuerwerk / Datenpfad - Modell

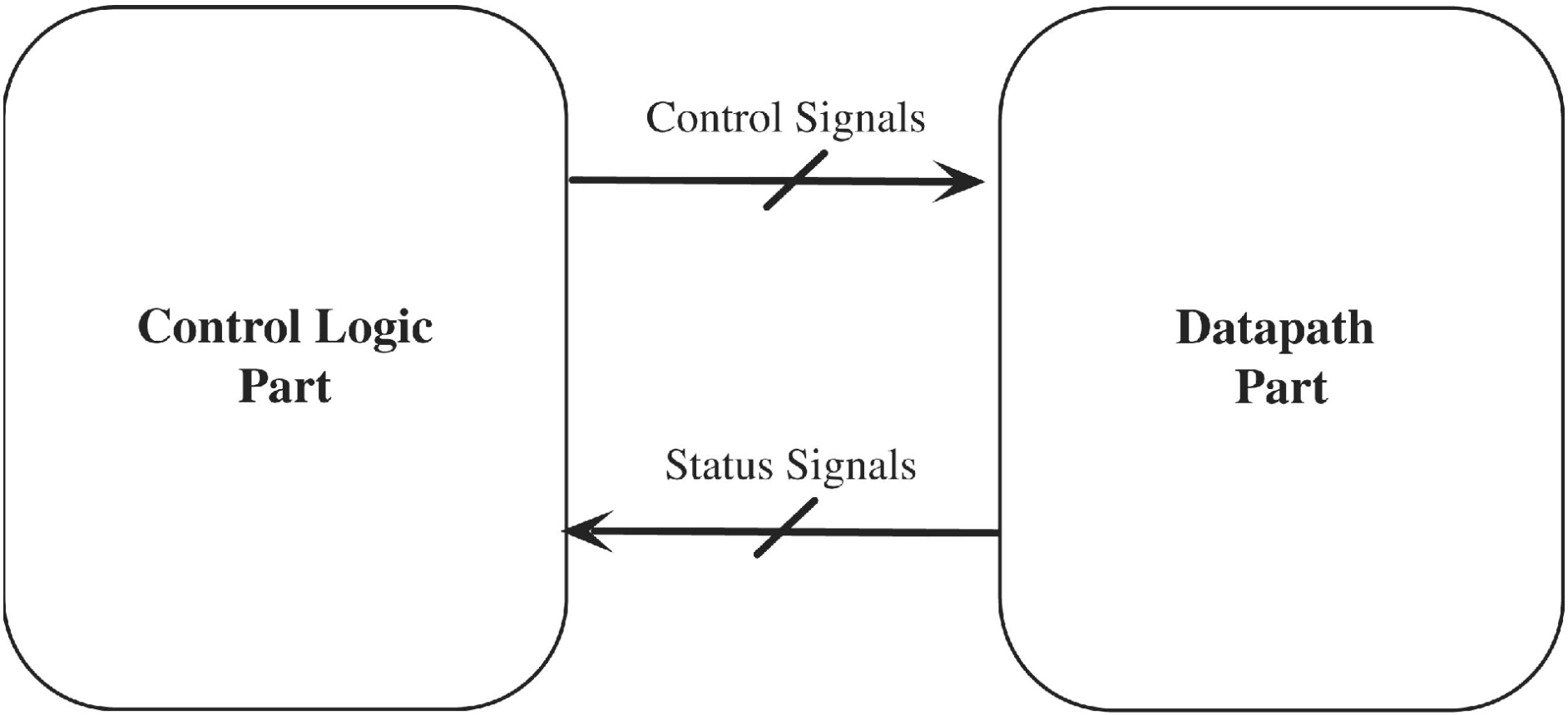
TECHNISCHE UNIVERSITÄT DARMSTADT

- Kernidee: Trennung von
  - Steuerung von Abläufen (*controller*)
  - Datenverarbeitung (*datapath*)
- In TGDI bereits eingeführt als, z.B. bei MIPS-Mikroarchitekturen
  - Steuerwerk
  - Datenpfad
- Hier: Verfeinerung und Umsetzung in Verilog

Steuerwerk / Datenpfad - Modell

Kurze Wiederholung

TECHNISCHE UNIVERSITÄT DARMSTADT

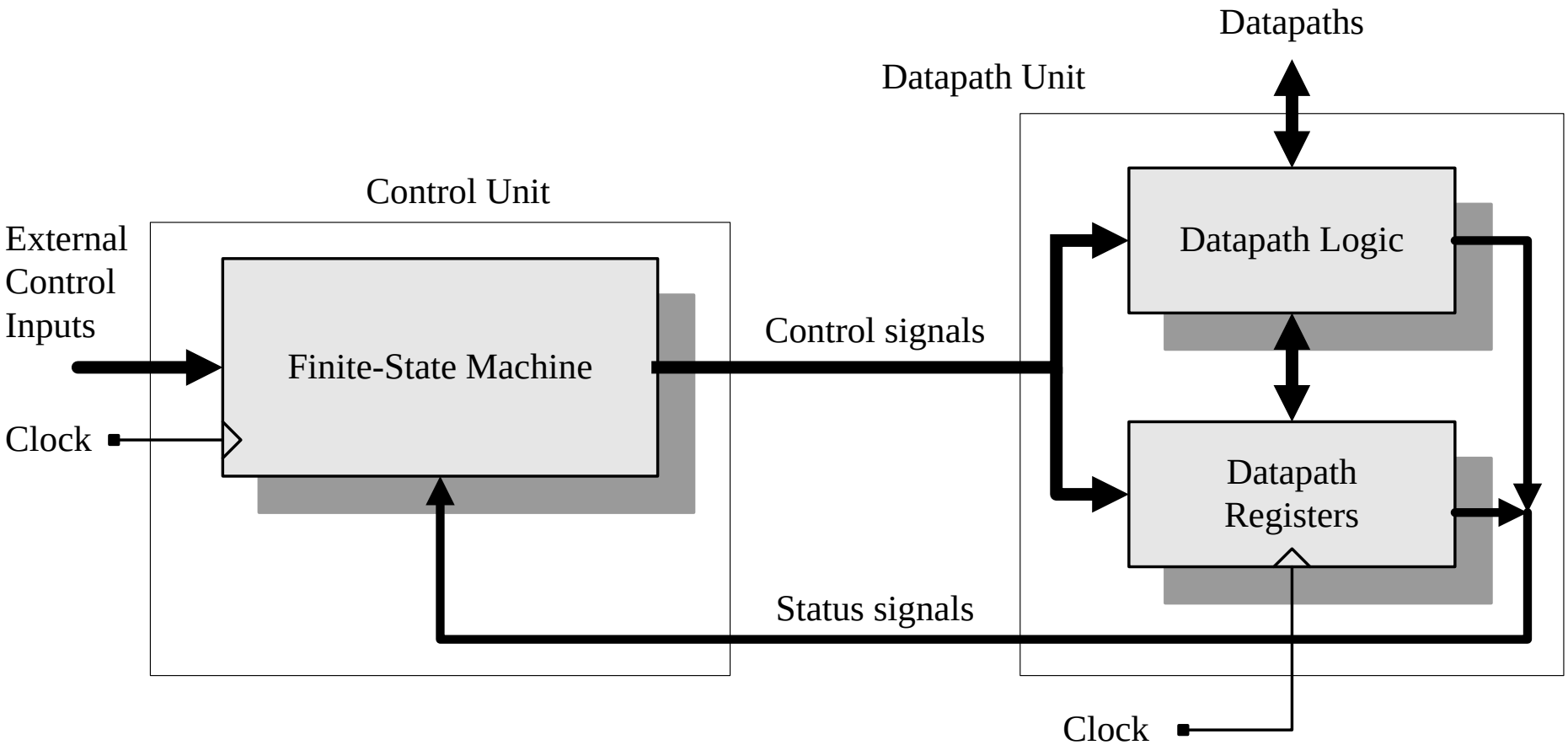


Quelle: Lee, Fig 5.1

Steuerwerk / Datenpfad - Modell

Genauerer Blick

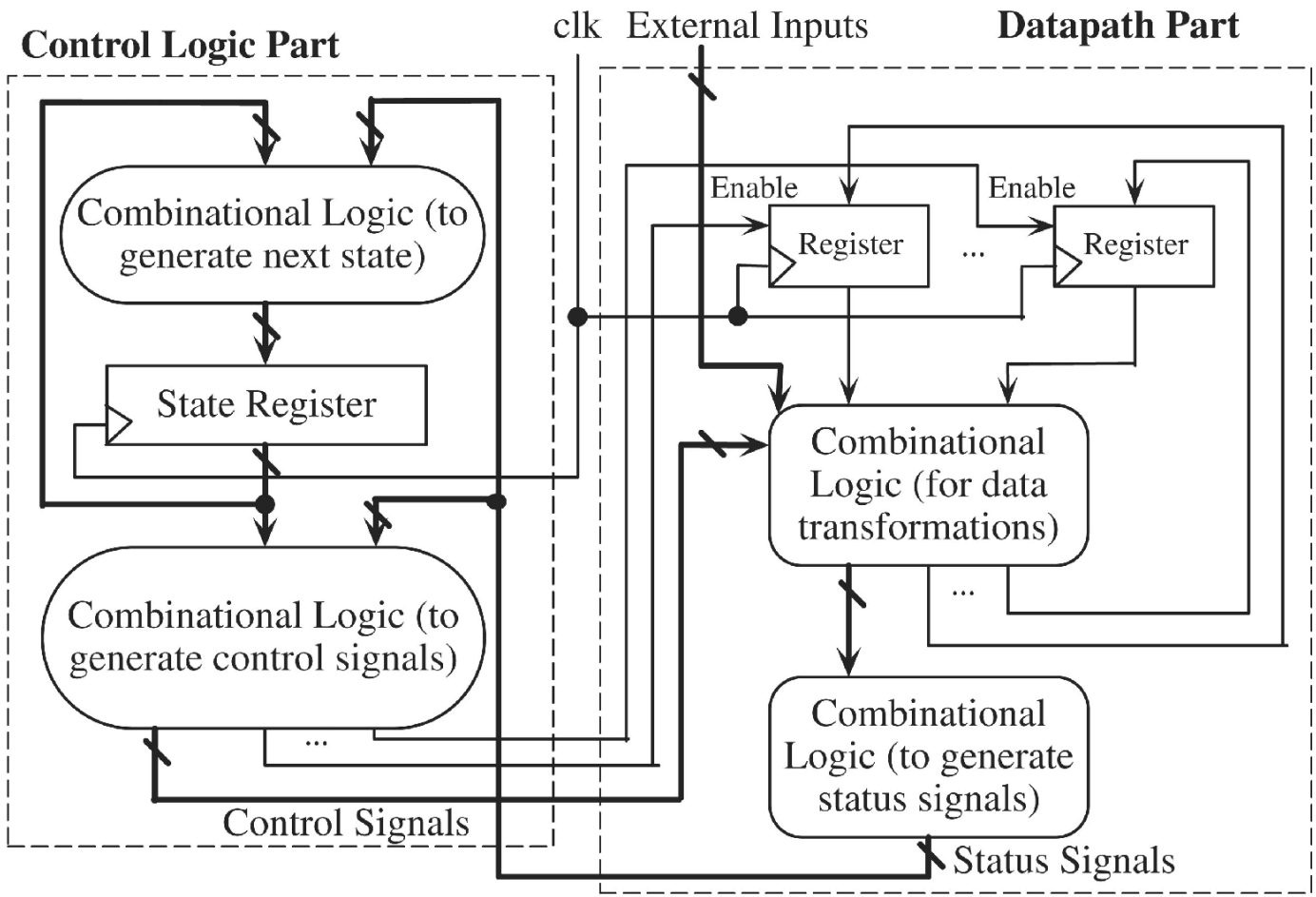
TECHNISCHE UNIVERSITÄT DARMSTADT



Quelle: Ciletti, Fig 7.1

Steuerwerk / Datenpfad-Modell

Noch genauerer Blick



Quelle: Lee, Fig 5.2

Diskussion

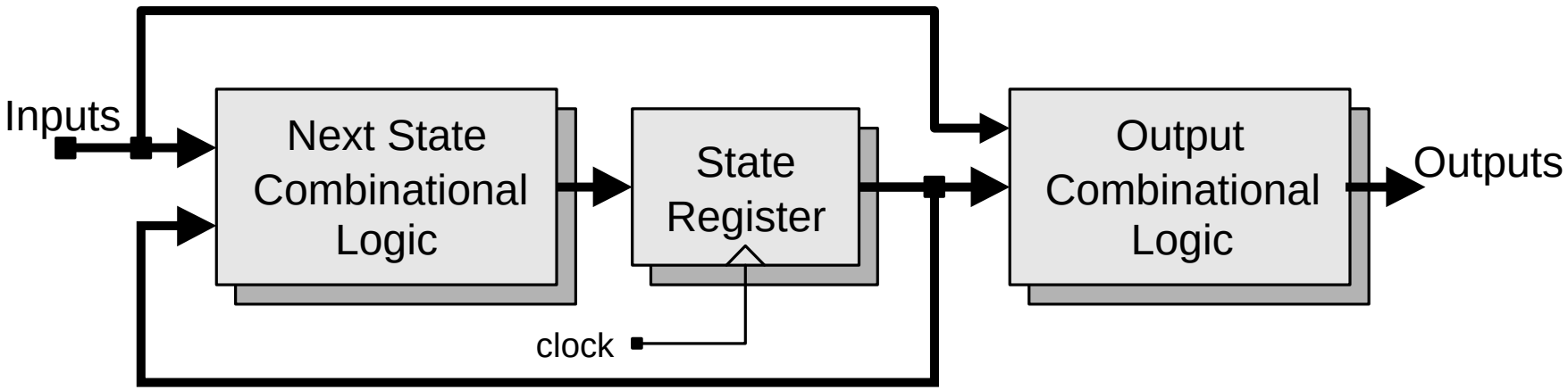
TECHNISCHE UNIVERSITÄT DARMSTADT

- Steuerwerk: Steuert Abläufe
  - Zustandsautomaten
  - Mealy / Moore / Mischformen
- Datenpfad: Manipuliert Daten
  - Speichern
    - Register
    - Echte Speicher (RAM, ROM)
  - Operationen
    - Arithmetisch
    - Logisch
  - Weiterleiten
    - Multiplexer
    - Tri-State-Busse

Exkurs: Zustandsautomaten in Hardware

Mealy-Automat

TECHNISCHE UNIVERSITÄT DARMSTADT



Ausgänge abhängig von

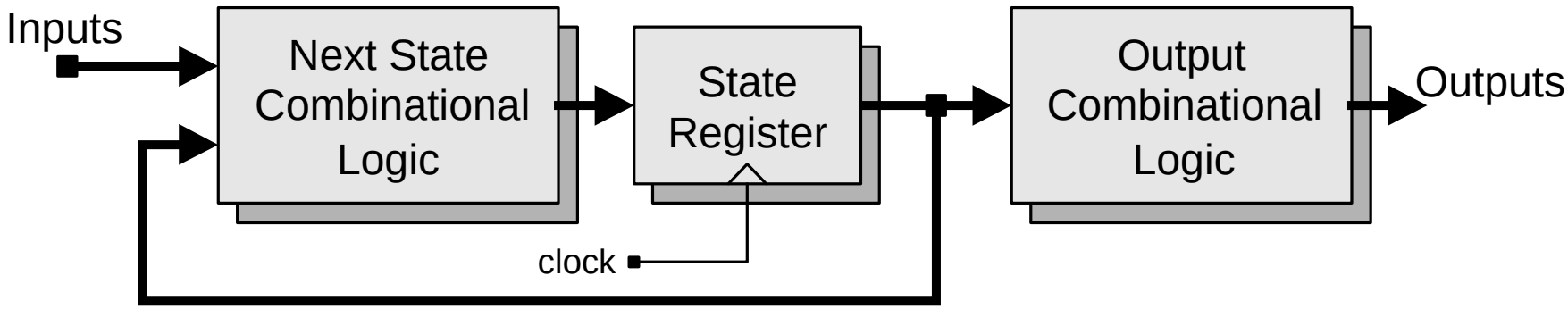
- aktuellem Zustand
- aktuellen Eingangssignalen

Quelle: Ciletti, Fig 6.44a

Exkurs: Zustandsautomat in Hardware

Moore-Automat

TECHNISCHE UNIVERSITÄT DARMSTADT



Ausgänge abhängig

- Nur von aktuellem Zustand

Quelle: Ciletti, Fig 6.44b

Beispiele für Kommunikation

Zwischen Steuerwerk und Datenpfad

TECHNISCHE UNIVERSITÄT DARMSTADT

Steuersignale von Steuerwerk zum Datenpfad

- “Übernehme neuen Wert in Register”
- “Schreibe Daten in Speicher”
- “Leite Daten auf 4. Eingang weiter”
- “Führe Additions-Operation aus”

Statussignale von Datenpfad zum Steuerwerk

- “Ergebnis ist Null”
- “Ergebnis ist ungerade”
- “Beide Werte sind gleich”
- “1. Wert ist größer als 2. Wert”

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Systematische Konstruktion

Umsetzung von Algorithmen in Hardware

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Beschreibe Algorithmus in Pseudo-Code
   - Wie beim Programmieren von Software
2. Schreibe Pseudo-Code in RTL-Beschreibung um
   - Keine `for`, `while`-Schleifen, Prozeduraufrufe
   - Aber Sprünge und `if/then/else` sind zugelassen!
   - Nur noch Konstrukte vergleichbar synthetisierbarem Verilog
   - Aber hier noch kein Verilog selbst erforderlich
3. Entwerfe Datenpfad-Struktur
   - Basierend auf Operationen in RTL-Beschreibung
4. Entwerfe Zustandsmaschine
   - auf Basis der RTL-Beschreibung
5. Realisiere Logik für Zustandsmaschine
   - Kann von Logiksynthese übernommen werden
   - Schauen wir uns hier aber genauer an

Beispiel: Fakultätsberechnung

Pseudo-Code

TECHNISCHE UNIVERSITÄT DARMSTADT

Annahmen

- Eingabe Zahl 0...7 (vorzeichenlos, 3b)
- Ausgabe ist 16b breit
- Signal `start`=1 startet Rechnung
- Signal `done`=1 zeigt Abschluss der Rechnung an

```
fact[15:0] := 1;
done := 0;

FOR count := 2 TO n[2:0] DO
  fact := fact * count;

done := 1;
```

RTL-Beschreibung 1

TECHNISCHE UNIVERSITÄT DARMSTADT

- Schleifen auflösen
  - In Bedingung und Sprung
- Aufteilen der Rechnung in Einzelschritte
  - Zunächst vergleichbar Assembler-Anweisungen

```
1: fact  := 1;
2: done  := 0;
3: count := 2;
4: IF (count <= n) THEN BEGIN
5:   fact  := fact * count;
6:   count := count + 1;
7:   GOTO 4;
8: END
9: done  := 1;
```

RTL-Beschreibung 2

Weitere Verfeinerung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Hardware rechnet parallel
- Alle parallel ausführbaren Operationen in einem Schritt

```
1: fact  := 1;
   done  := 0;
   count := 2;
2: IF (count <= n) THEN BEGIN
     fact  := fact * count;
     count := count + 1;
     GOTO 2;
   END
3: done  := 1;
```

Jetzt parallele Operationen in einem Schritt

Diskussion: Parallele Operationen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Alle Operationen in einem Schritt rechnen mit gleichen Variablenwerten
- Zuweisungen werden erst im nächsten Schritt sichtbar

```
1: fact  := 1;
   done  := 0;
   count := 2;
2: IF (count <= n) THEN BEGIN
     fact  := fact * count;
     count := count + 1;
     GOTO 2;
   END
3: done  := 1;
```

```
1: fact  := 1;
   done  := 0;
   count := 2;
2: IF (count <= n) THEN BEGIN
     count := count + 1;
     fact  := fact * count;
     GOTO 2;
   END
3: done  := 1;
```

➥Gleiches Ergebnis!

Datenpfad aus RTL-Beschreibung ableiten 1

TECHNISCHE UNIVERSITÄT DARMSTADT

```
1: fact  := 1; // [15:0]
   done  := 0;
   count := 2; // [2:0]
2: IF (count <= n) THEN BEGIN
     fact  := fact * count;
     count := count + 1;
     GOTO 2;
   END
3: done  := 1;
```

Datenpfad aus RTL-Beschreibung ableiten 2

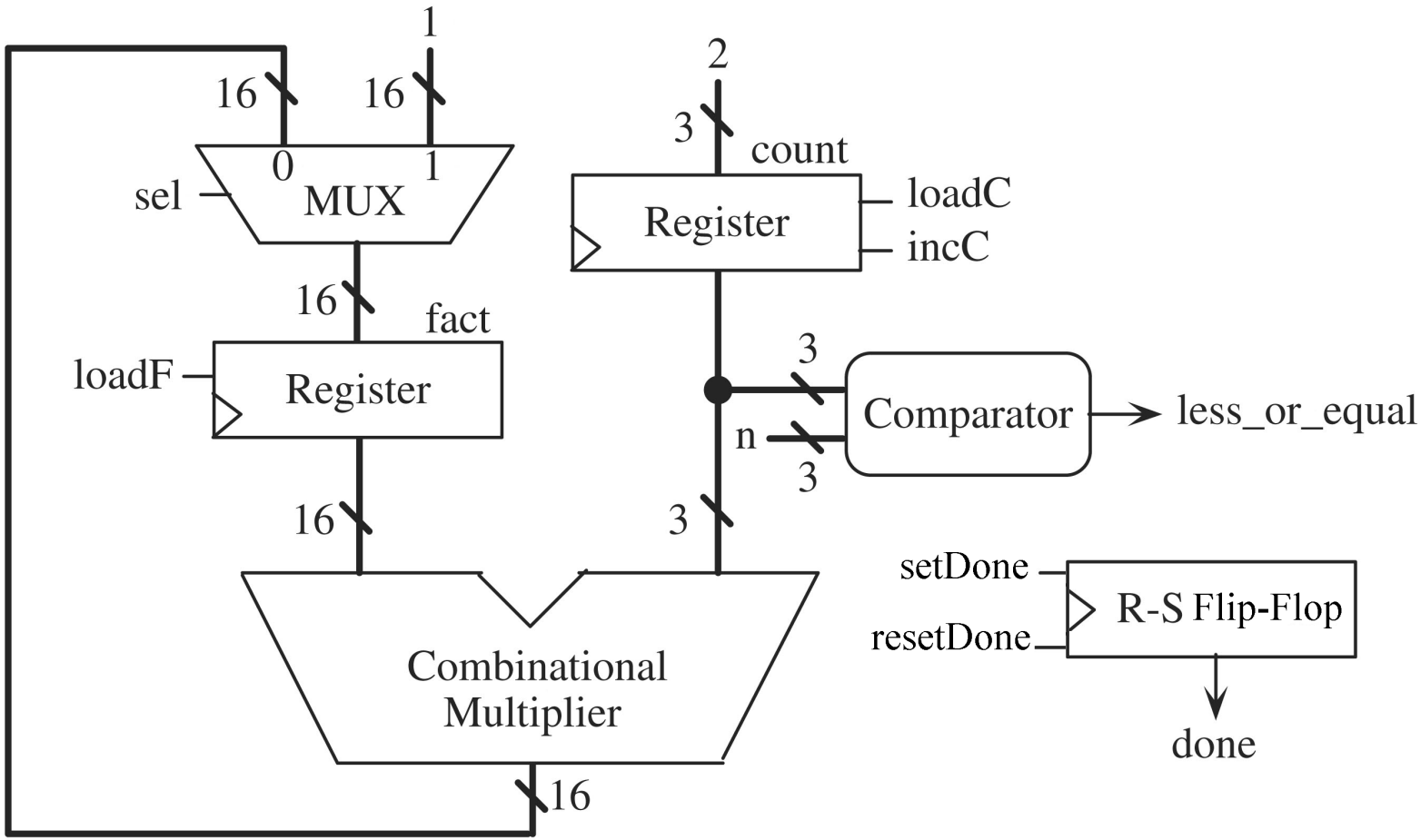
TECHNISCHE UNIVERSITÄT DARMSTADT

1. Variablen werden Register
   - Ggf. Spezialregister ausnutzen
   - Zähler bei inkrementieren/dekrementieren
   - Schieberegister bei verdoppeln/halbieren
2. Variablen mit mehreren Quellen für Werte
   - Multiplexer oder Tri-State-Busse am Registereingang
   - Wählt aktuelle Quelle aus
3. Operatoren werden arithmetische/logische Blöcke
4. Steuersignale bestimmen, Beispiele:
   - Wann übernimmt Register neuen Wert?
   - Soll Zähler diesen Takt zählen?
   - Welcher Mux-Eingang soll auf den Ausgang gelegt werden?
5. Statussignale bestimmen, Beispiel:
   - Was war das Ergebnis eines Vergleichs?

Datenpfad

Eine Möglichkeit aus vielen!

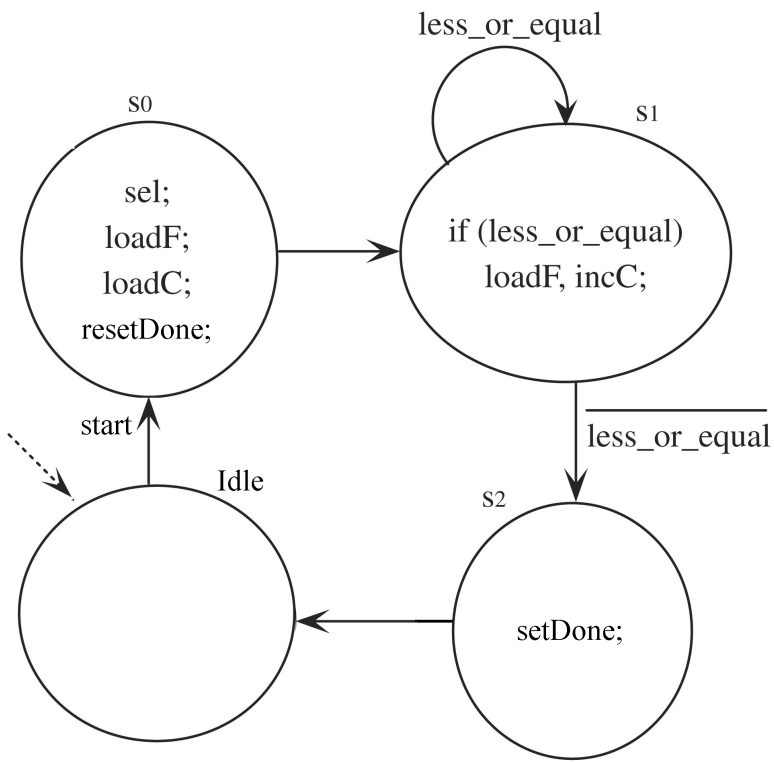
TECHNISCHE UNIVERSITÄT DARMSTADT



Zustandsautomat als Zustandsübergangsgraph

***state transition graph* (STG)**

TECHNISCHE UNIVERSITÄT DARMSTADT



- Steuersignale in Zuständen
  - Nicht aufgeführte Signale $\rightarrow$ deaktiviert
- Statussignale
  - Zur Berechnung von Ausgangssignalen in Mealy-Zustand
  - An Übergängen: Boolesche Verknüpfung von Statussignalen
    - Und nichts anderes!

Verilog-Realisierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Nun Verilog-Modell formulierbar
- Saubere Trennung von Steuerwerk und Datenpfad
  - In eigene Module
- Im Steuerwerk saubere Trennung von
  - Kombinatorischen Rechnungen
  - Speicherelementen
  - Register-Transfer-Logik

Hauptmodul des Fakultätsrechners

TECHNISCHE UNIVERSITÄT DARMSTADT

```
module fac (
  input          CLK, RESET,
  input          START,
  input    [2:0] N,
  output [15:0] FACT,
  output         DONE
);

  facdp   FACDP (CLK, RESET, SEL, LOADF, LOADC, INCC, RDONE, SDONE,
                 N, LEQ, DONE, FACT);
  facfsm FACFSM(CLK, RESET, START, LEQ,
                 SEL, LOADF, LOADC, INCC, RDONE, SDONE);

endmodule
```

Saubere Trennung von Steuerwerk und Datenpfad

Datenpfad: Schnittstelle und Verwaltung

TECHNISCHE UNIVERSITÄT DARMSTADT

```
module facdp (
  input              CLK, RESET, SEL, LOADF, LOADC, INCC, RDONE, SDONE,
  input       [2:0]  N,
  output             LEQ,
  output reg         DONE,
  output reg [15:0]  FACT
);

  reg  [2:0]  COUNT;

  always @(posedge CLK, posedge RESET) begin
    if  (RESET) begin
      FACT <= 0;
      COUNT <= 0;
      DONE <= 0;
    end else  ...
```

Datenpfad: Aktiver Teil

TECHNISCHE UNIVERSITÄT DARMSTADT

```
... end else begin

    if (LOADF)                    // Behandlung von fact
      FACT <= (SEL) ? 1 : (FACT * COUNT);

    if (LOADC)                    // Behandlung von count
      COUNT <= 2;
    else if (INCC)
      COUNT <= COUNT + 1;

    case ({SDONE,RDONE})          // Behandlung von done
      2'b10: DONE <= 1;
      2'b01: DONE <= 0;
    endcase

  end
end

// Statussignal less_or_equal für Steuerwerk
assign LEQ = (COUNT <= N);

endmodule
```

Steuerwerk: Schnittstelle und Verwaltung

TECHNISCHE UNIVERSITÄT DARMSTADT

```
module facfsm (
  input       CLK, RESET, START, LEQ,
  output reg SEL, LOADF, LOADC, INCC, RDONE, SDONE
);

  localparam IDLE = 0;
  localparam S0 = 1;
  localparam S1 = 2;
  localparam S2 = 3;
  reg  [1:0]  STATE, NEXTSTATE;

   ...
```

Steuerwerk: Aktiver Teil

TECHNISCHE UNIVERSITÄT DARMSTADT

```
always @(STATE,START,LEQ) begin

  SEL = 0; LOADF = 0; LOADC = 0; INCC = 0; RDONE = 0;SDONE = 0; // Latches vermeiden
  NEXTSTATE = IDLE;

  case (STATE)
    IDLE:   if (START)                  // Auf Startsignal warten
              NEXTSTATE = S0;
    S0:     begin                       // Datenpfad initialisieren
              SEL = 1; LOADF = 1; LOADC = 1; RDONE = 1;
              NEXTSTATE = S1;
            end
    S1:     if (LEQ) begin              // Schleife count <= n
              LOADF = 1; INCC = 1;
              NEXTSTATE = S1;
            end else
              NEXTSTATE = S2;
    S2:     begin                       // Ende der Berechnung anzeigen
              SDONE = 1;
              NEXTSTATE = IDLE;
            end
  endcase
end

always @(posedge CLK, posedge RESET) begin // Neuen Zustand übernehmen
  if (RESET)    STATE <= IDLE;
  else          STATE <= NEXTSTATE;
end
```

Testrahmen: Kopf und Verwaltung

TECHNISCHE UNIVERSITÄT DARMSTADT

```
module tb_fac;

reg          CLK;
reg          RESET;
reg          START;
reg    [2:0] N;
wire  [15:0] FACT;
wire         DONE;

  // Unit-under-Test instantiieren
  fac FAC(CLK, RESET, START, N, FACT, DONE);

  // Takt erzeugen
  always begin
    CLK = 0;
    #5;
    CLK = 1;
    #5;
  end
...
```

Testrahmen: Aktiver Teil

TECHNISCHE UNIVERSITÄT DARMSTADT

```
initial  begin
  $monitor("%t␣START=%d␣N=%d␣DONE=%d␣FACT=%d", $time, START, N, DONE, FACT);

  @(negedge CLK);       // Reset der Schaltung
  RESET = 1;
  @(negedge CLK);
  RESET = 0;

  @(negedge CLK);       // Berechne 3! = 6
  N = 3;
  START = 1;
  @(negedge CLK);
  START = 0;
  @(negedge CLK);
  while (!DONE)
    @(posedge CLK);

  @(negedge CLK);       // Berechne 5! = 120
  N = 5;
  START = 1;
  @(negedge CLK);
  START = 0;
  @(negedge CLK);
  while (!DONE)
    @(posedge CLK);

  $finish;
end
```

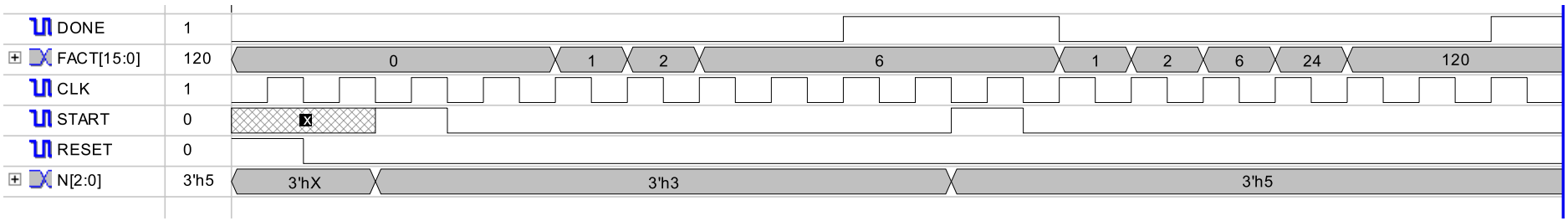
Simulationsergebnisse

TECHNISCHE UNIVERSITÄT DARMSTADT

```
  0 START=x N=x DONE=x FACT=   x
 10 START=x N=x DONE=0 FACT=   0
 30 START=1 N=3 DONE=0 FACT=   0
 40 START=0 N=3 DONE=0 FACT=   0
 45 START=0 N=3 DONE=0 FACT=   1
 55 START=0 N=3 DONE=0 FACT=   2
 65 START=0 N=3 DONE=0 FACT=   6
 85 START=0 N=3 DONE=1 FACT=   6
100 START=1 N=5 DONE=1 FACT=   6
110 START=0 N=5 DONE=1 FACT=   6
115 START=0 N=5 DONE=0 FACT=   1
125 START=0 N=5 DONE=0 FACT=   2
135 START=0 N=5 DONE=0 FACT=   6
145 START=0 N=5 DONE=0 FACT=  24
155 START=0 N=5 DONE=0 FACT= 120
175 START=0 N=5 DONE=1 FACT= 120
```

Signalverlaufsdiagramm

TECHNISCHE UNIVERSITÄT DARMSTADT



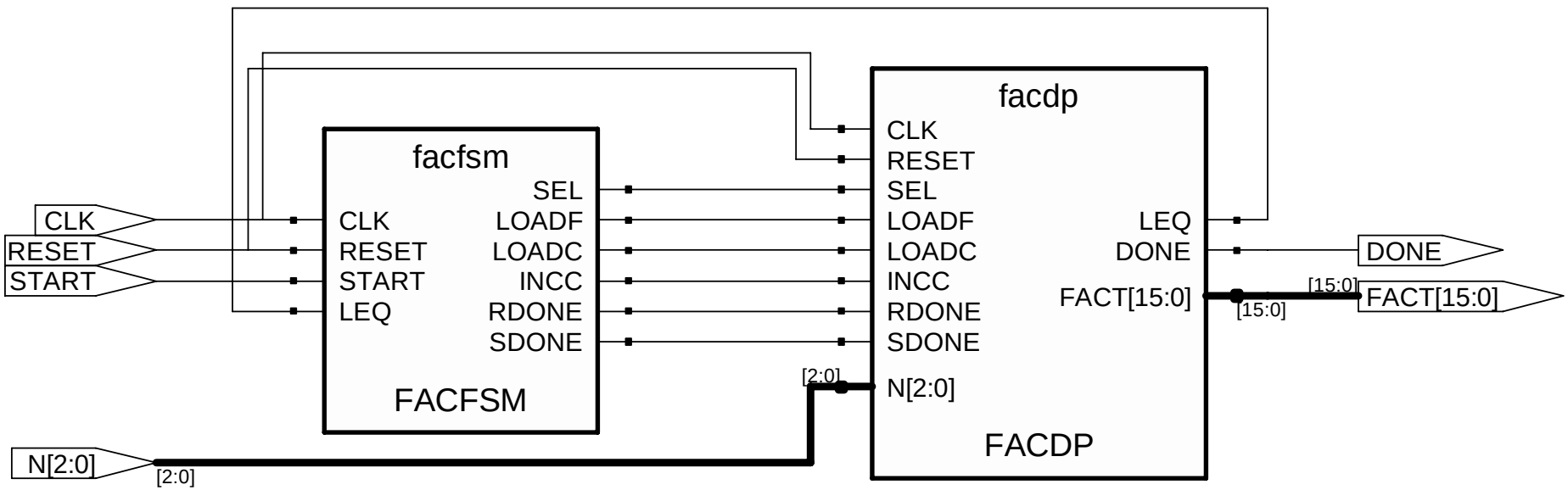
Beachte Verzögerung bei zweiter Berechnung 5!

`START=1` $\longrightarrow$ `RDONE=1` $\longrightarrow$ `DONE=0`

Alternative: Anderes Protokoll mit `DONE=1` nur für einen Takt

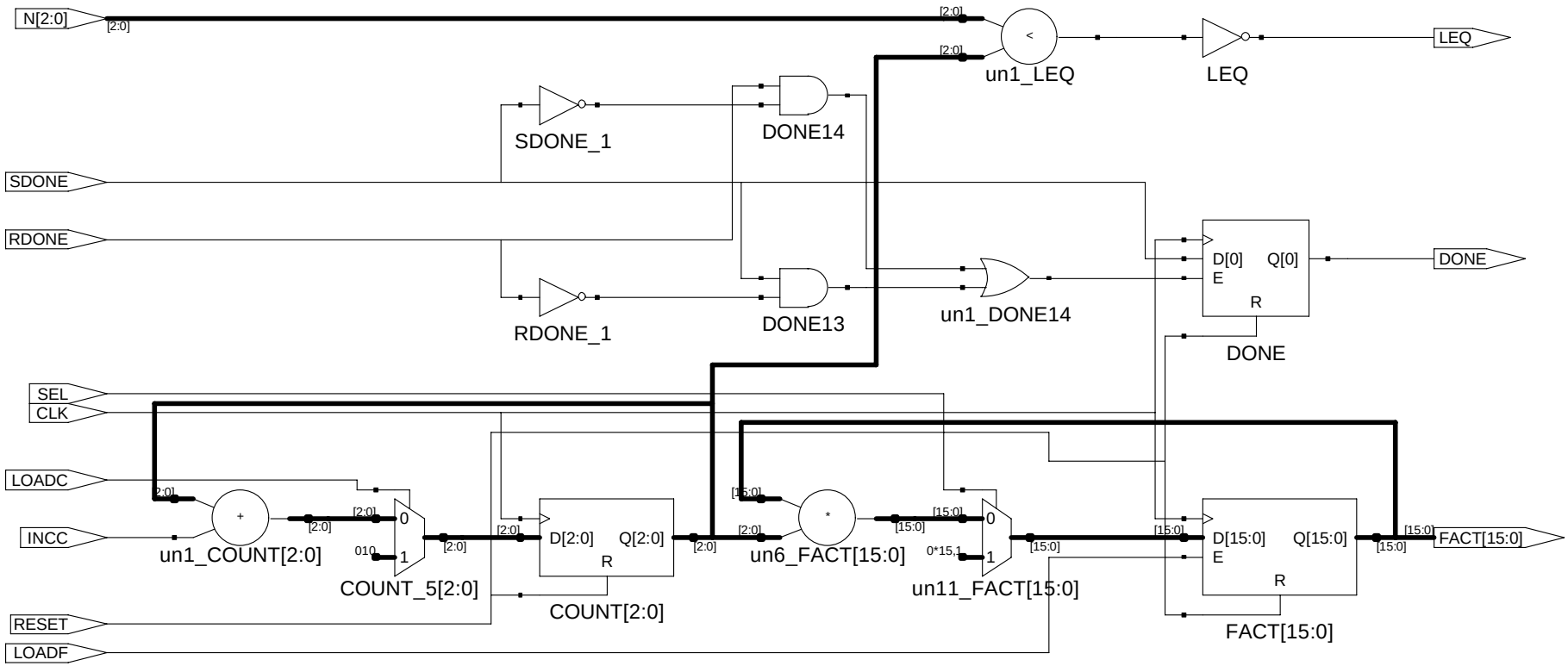
Syntheseergebnisse: Hauptmodul

TECHNISCHE UNIVERSITÄT DARMSTADT



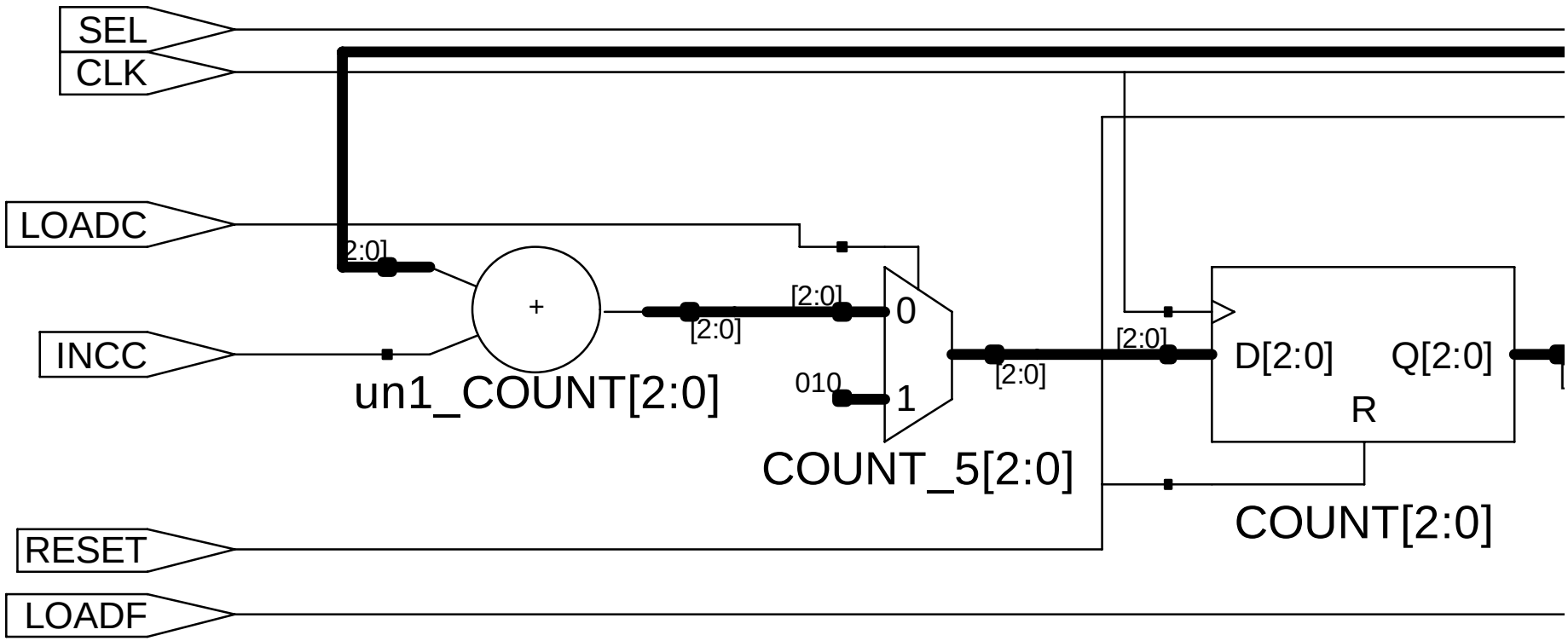
Syntheseergebnisse: Datenpfad

TECHNISCHE UNIVERSITÄT DARMSTADT



Syntheseergebnisse: Datenpfad

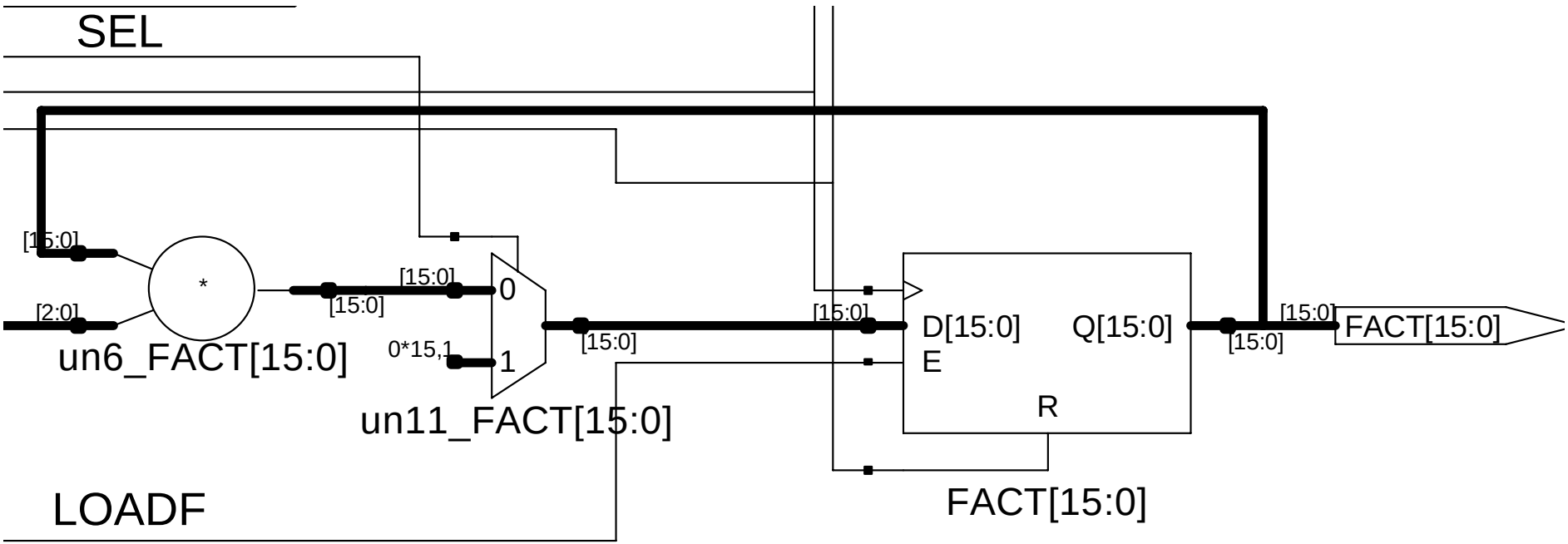
Schleifenzähler



Syntheseergebnisse: Datenpfad

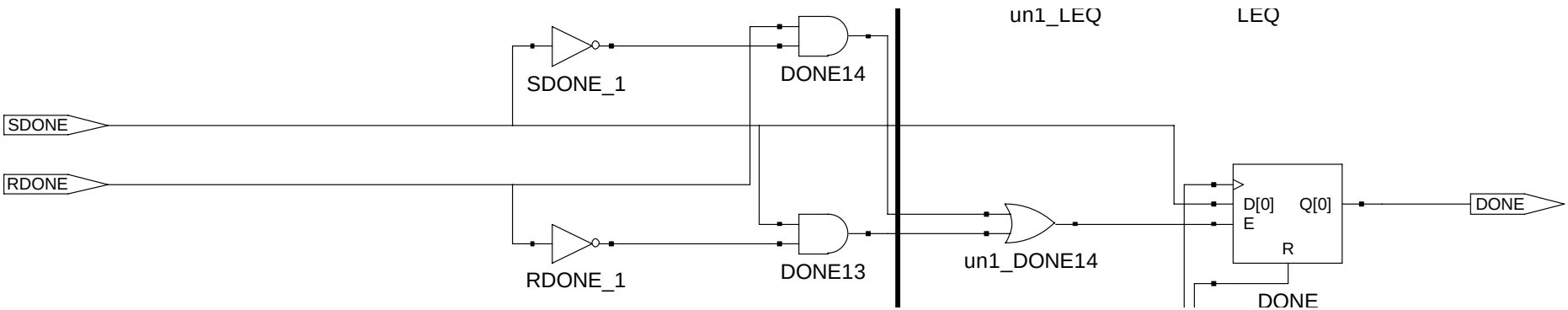
Produktberechnung

TECHNISCHE UNIVERSITÄT DARMSTADT



Syntheseergebnisse: Datenpfad

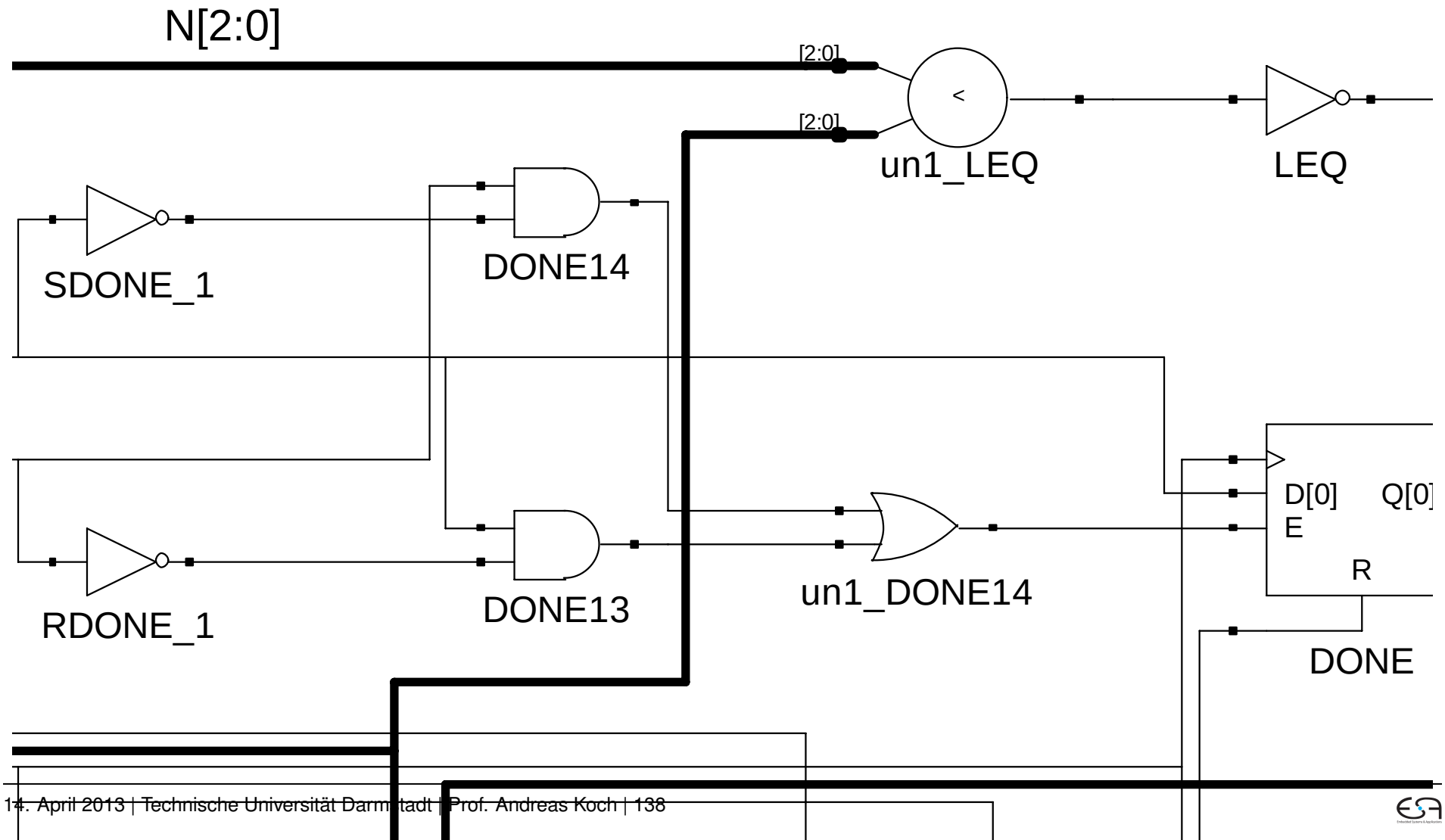
Berechnung des DONE-Signals



Syntheseergebnisse: Datenpfad

Berechnung des `LEQ`-Signals

TECHNISCHE UNIVERSITÄT DARMSTADT



Zwischenstand nach 1. Block der Vorlesung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Überblick über Verilog
  - Simulation
  - Logiksynthese
- Weitergehende Optimierungen
  - Auch jenseits der Logiksynthese
- Systematischer Schaltungsentwurf

Alles aber immer noch sehr nah an der digitalen synchronen Logik.
➥Geht das nicht auch abstrakter?

Bluespec SystemVerilog $\longrightarrow$ 2. Block

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Einführung in BlueSpec

Synthese

Logiksynthese

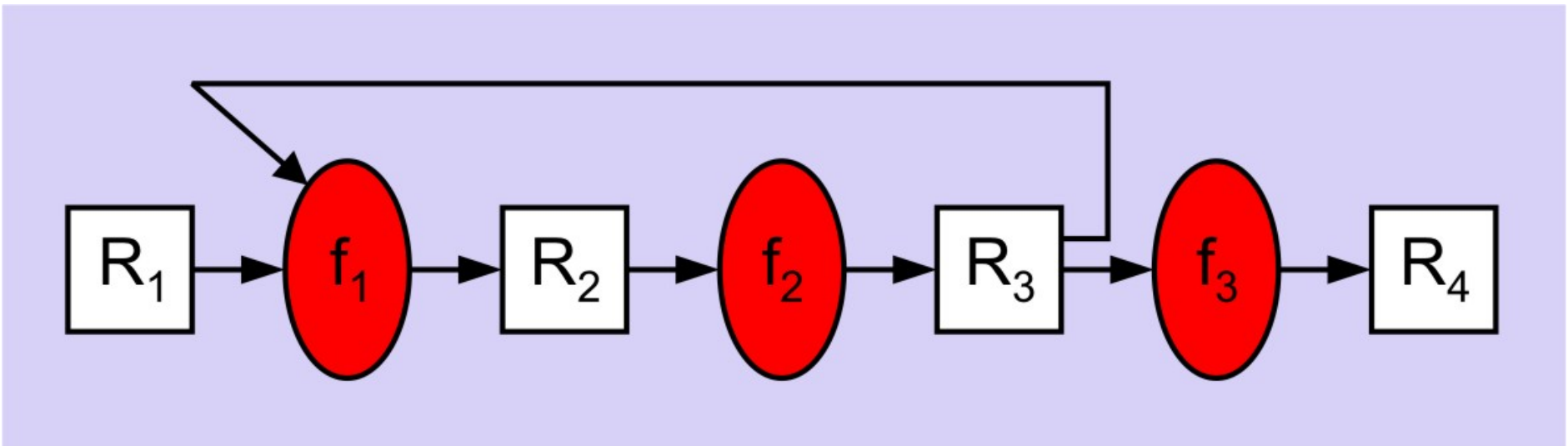
TECHNISCHE UNIVERSITÄT DARMSTADT

- Abbildung von RTL-Modell auf Gattermodell
  - Register-Transfer-Ebene auf Logikebene
- Wichtige Hersteller von Entwurfswerkzeugen
  - Für ASICs: Synopsys, Cadence
  - Für FPGAs: Synopsys, Mentor Graphics
  - Gibt aber auch noch diverse andere Anbieter

Logiksynthese

TECHNISCHE UNIVERSITÄT DARMSTADT

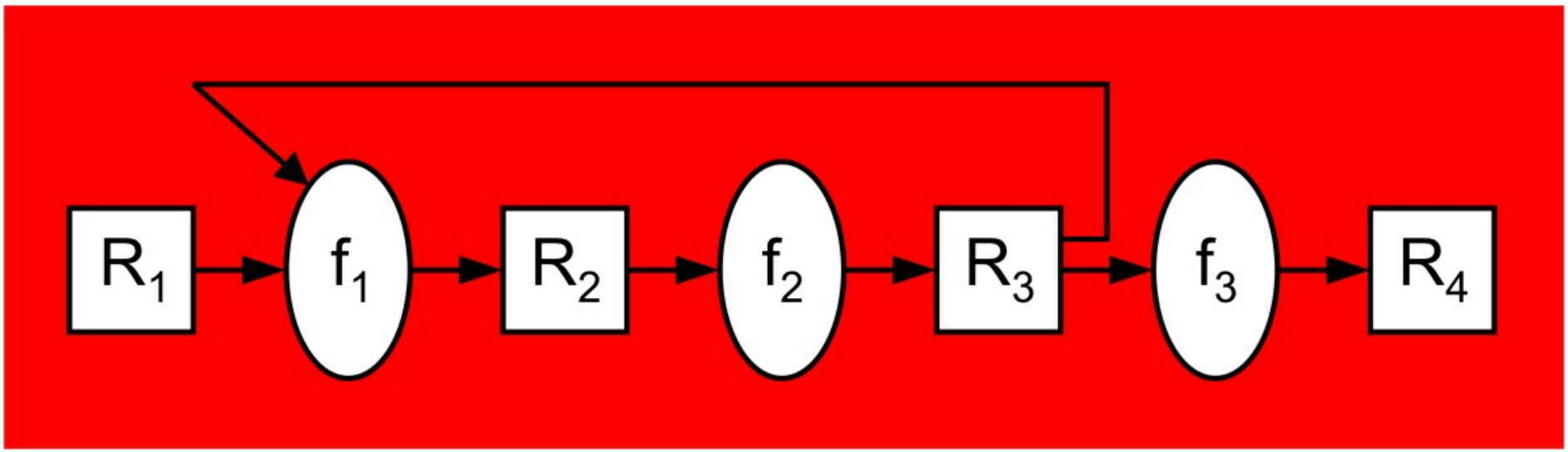
Optimiert im wesentlichen Logik zwischen getakteten Registern



Vergleich: High-Level-Synthese

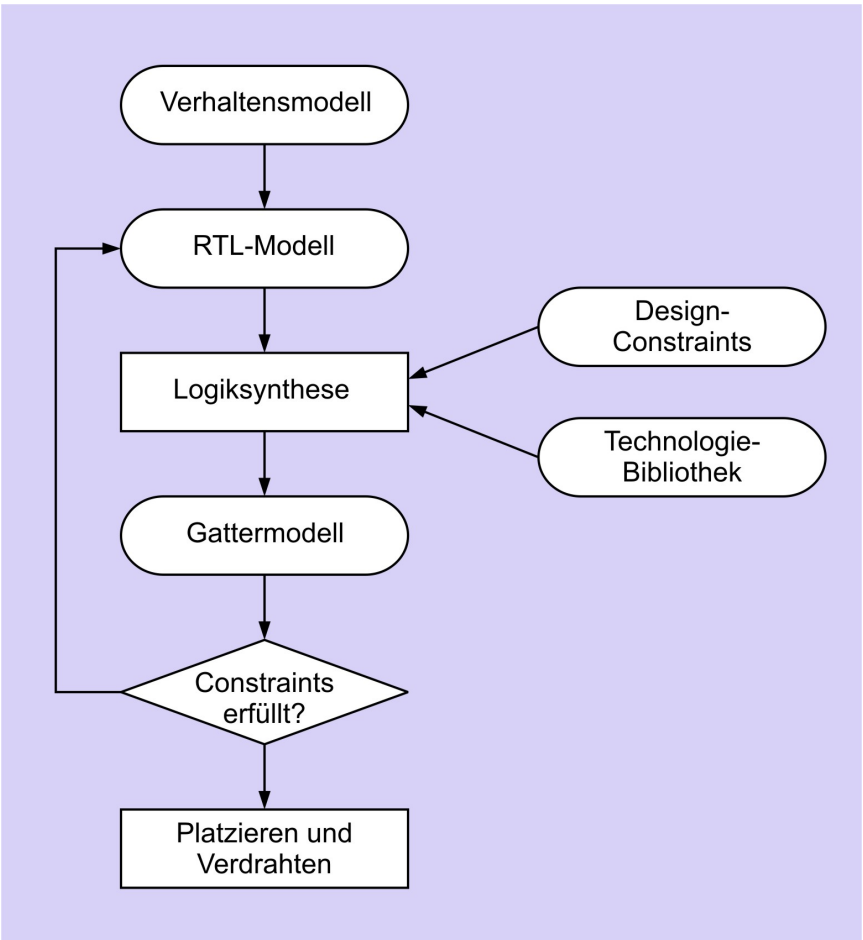
TECHNISCHE UNIVERSITÄT DARMSTADT

Beginnt oberhalb von RTL und optimiert über Taktgrenzen hinweg



Noch experimentell, nur eingeschränkte praktische Bedeutung

Ablauf der Logiksynthese



- RTL-Modell in Verilog
- Design-Constraints
  - Wie schnell?
  - Wie groß?
  - (Wieviel Energie?)
- Zieltechnologie
  - AND, OR
  - Addierer, Flip-Flops
  - Abbildung auf LUTs
  - Genaue Laufzeiten
  - Genaue Flächenangaben

Vorteile der Logiksynthese

TECHNISCHE UNIVERSITÄT DARMSTADT

- Kürzere Entwurfszeit
- Weniger fehleranfällig
- Anforderungen an Zeit und Fläche aufstellbar
- Portabilität zwischen verschiedenen Chip-Herstellern
- Leichtere Exploration des Entwurfsraumes
  - Wieviel langsamer, wenn 25% kleiner?
- Einheitlicher Entwurfsstil bei Team-Arbeit
- Leichtere Wiederverwendung von (Teil-)Entwürfen

Wichtigste synthetisierbare Verilog-Konstrukte

TECHNISCHE UNIVERSITÄT DARMSTADT

Signale und Variablen	`wire, reg`

prozedural	`always, begin, end,`
	`if, else,`
	`case,`
	`function, task, =, <=`

Struktur	`module,`
	`input, inout, output,`
	`parameter,`
	
	`assign`

Eingeschränkt: `for`

Einige nichtsynthetisierbare Verilog-Konstrukte

TECHNISCHE UNIVERSITÄT DARMSTADT

- initial: Stattdessen explizites Reset-Verhalten beschreiben
- Zeitkontrolle: `#` und `@` innerhalb von Block
  - Alle Zeitverzögerungen aus Beschreibung der Zieltechnologie

➥Prä- und Post-Synthese-Simulationen können differieren

Synthetisierbare Operatoren

TECHNISCHE UNIVERSITÄT DARMSTADT

arithmetisch	`*, /, +, -, %`
logisch	`!, &&, ||`
bit-weise	`~, &, |, ^, ^~, ~^`
Reduktion	`&, ~&, |, ~|, ^, ^~, ~^`
Relation	`>, <, >=, <=`
Gleichheit	`==, !=` aber kein `===` und `!==` mit `x` und `z`
Shift	`>>, <<, <<<, >>>`
Konkatenation	`{ }`
bedingt	`?:`

Syntheseergebnisse

Vorgehen

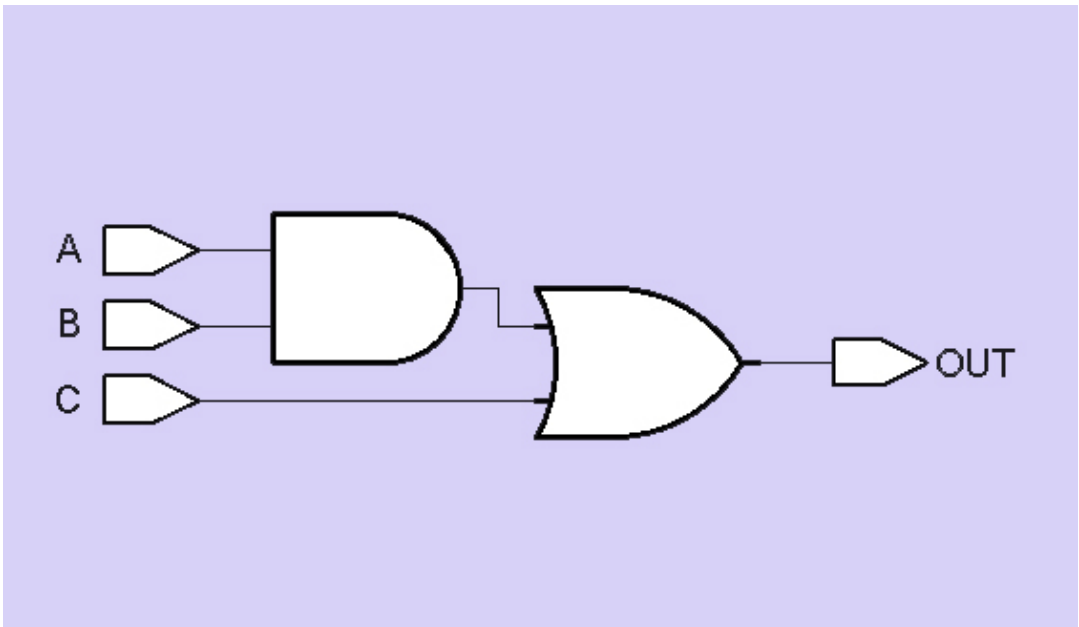
TECHNISCHE UNIVERSITÄT DARMSTADT

- Zunächst Abbildung auf allgemeines Gattermodell
- Noch weitgehend ohne Berücksichtigung der Zieltechnologie
- Reine Zwischendarstellung

Synthese einer `assign`-Anweisung

TECHNISCHE UNIVERSITÄT DARMSTADT

assign OUT = (A & B) | C;



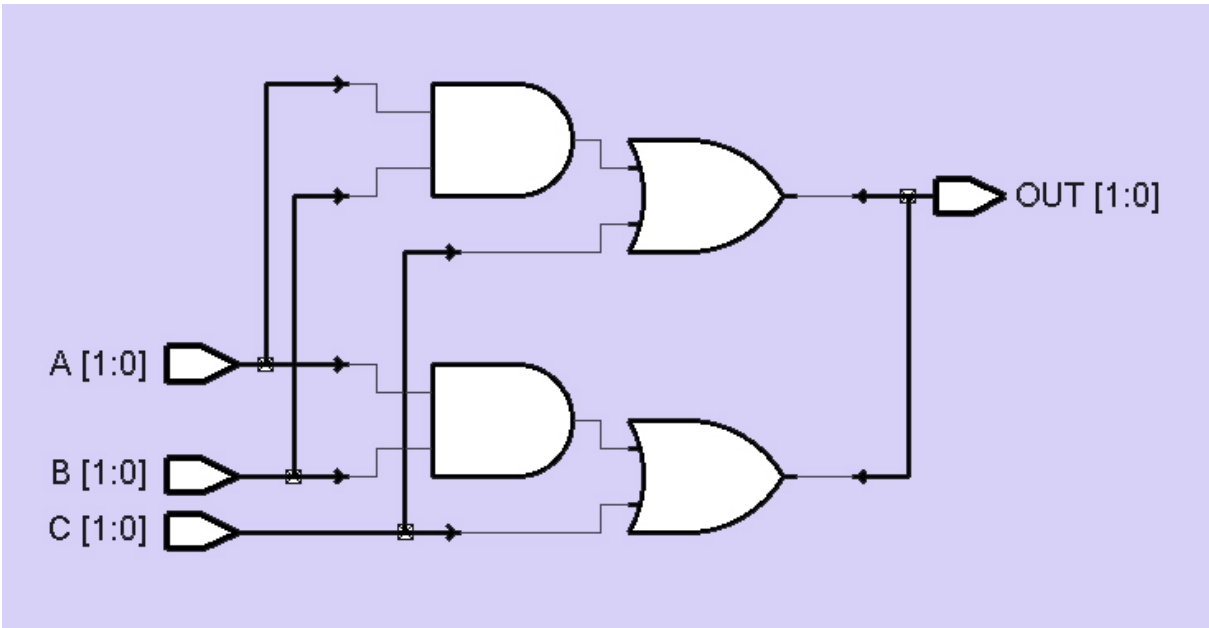
Annahme hier: Alle Signale 1b breit

Synthese einer `assign`-Anweisung

Hier: 2b breite Signale

TECHNISCHE UNIVERSITÄT DARMSTADT

assign OUT = (A & B) | C;

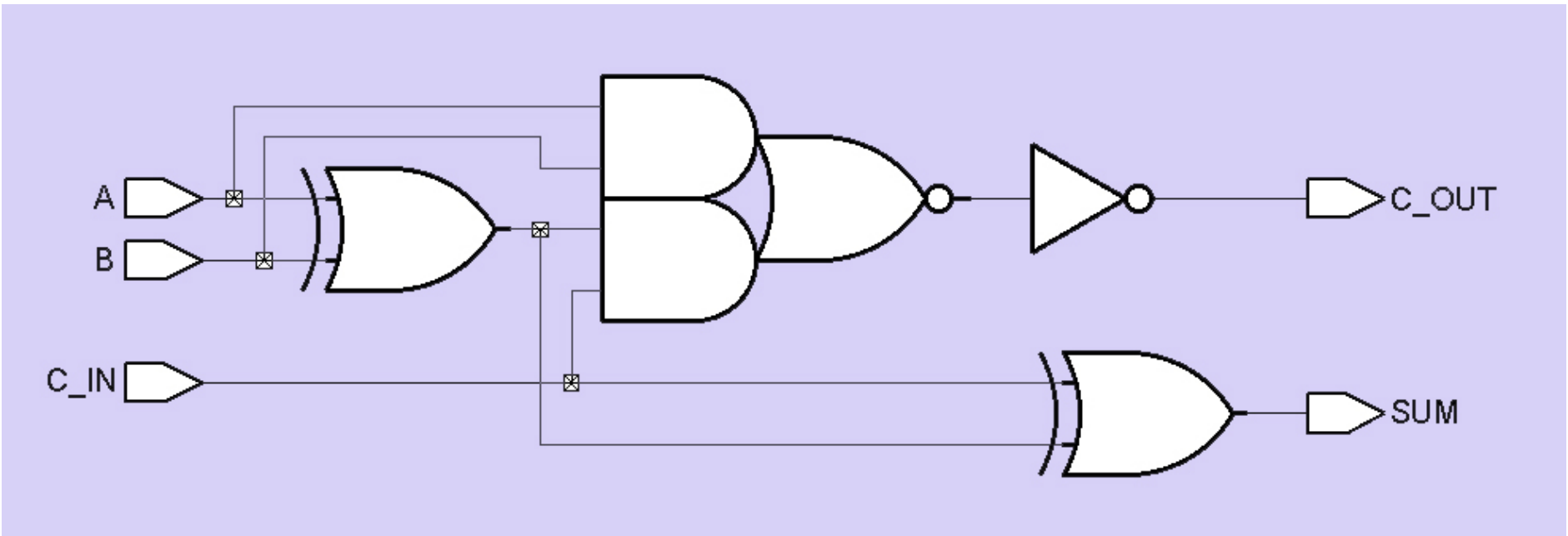


Synthese einer `assign`-Anweisung

TECHNISCHE UNIVERSITÄT DARMSTADT

assign {C_OUT, SUM} = A + B + C_IN

1b-Volladdierer



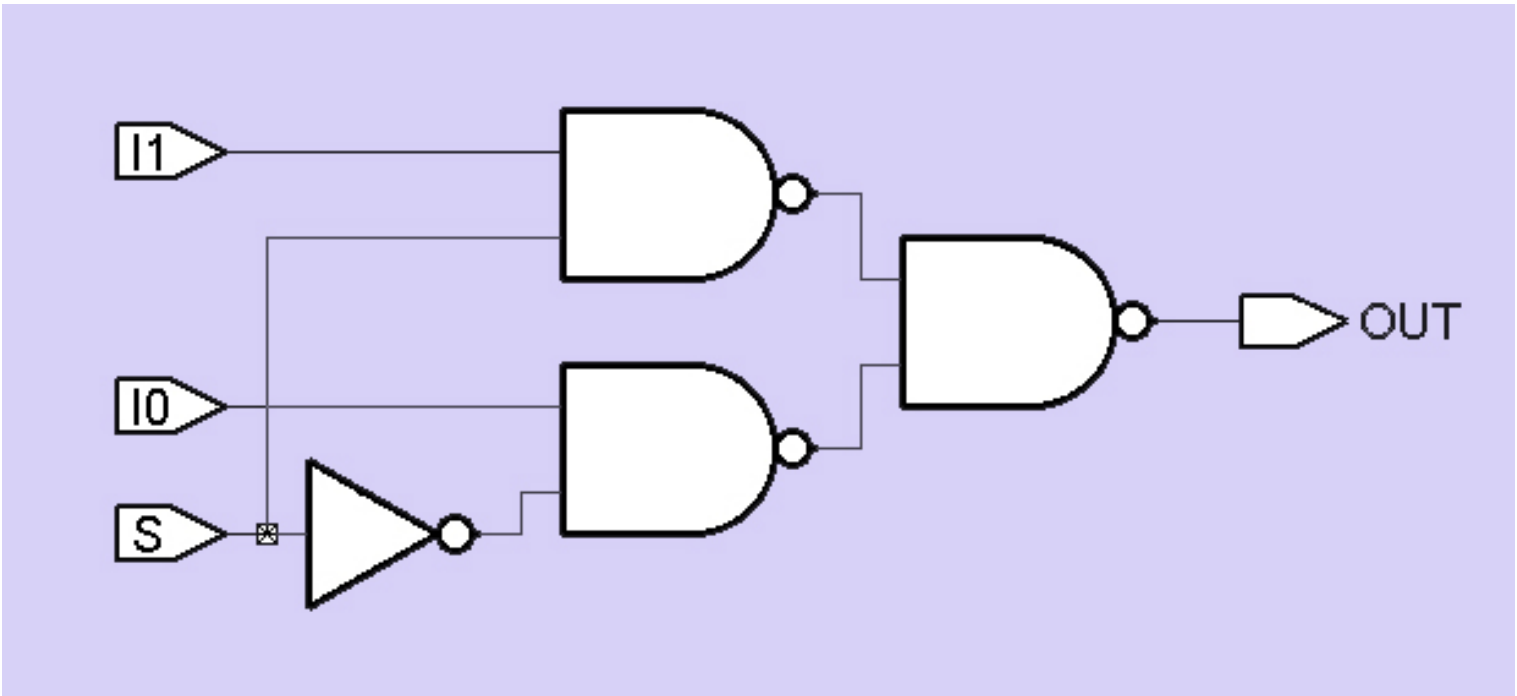
Merkwürdiges Gatter in der Mitte: AND-OR-INVERT (AOI),
sehr effizient in ASIC-Technologie realisierbar

Synthese einer `assign`-Anweisung

TECHNISCHE UNIVERSITÄT DARMSTADT

assign OUT = (S) ? I1 : I0

Multiplexer



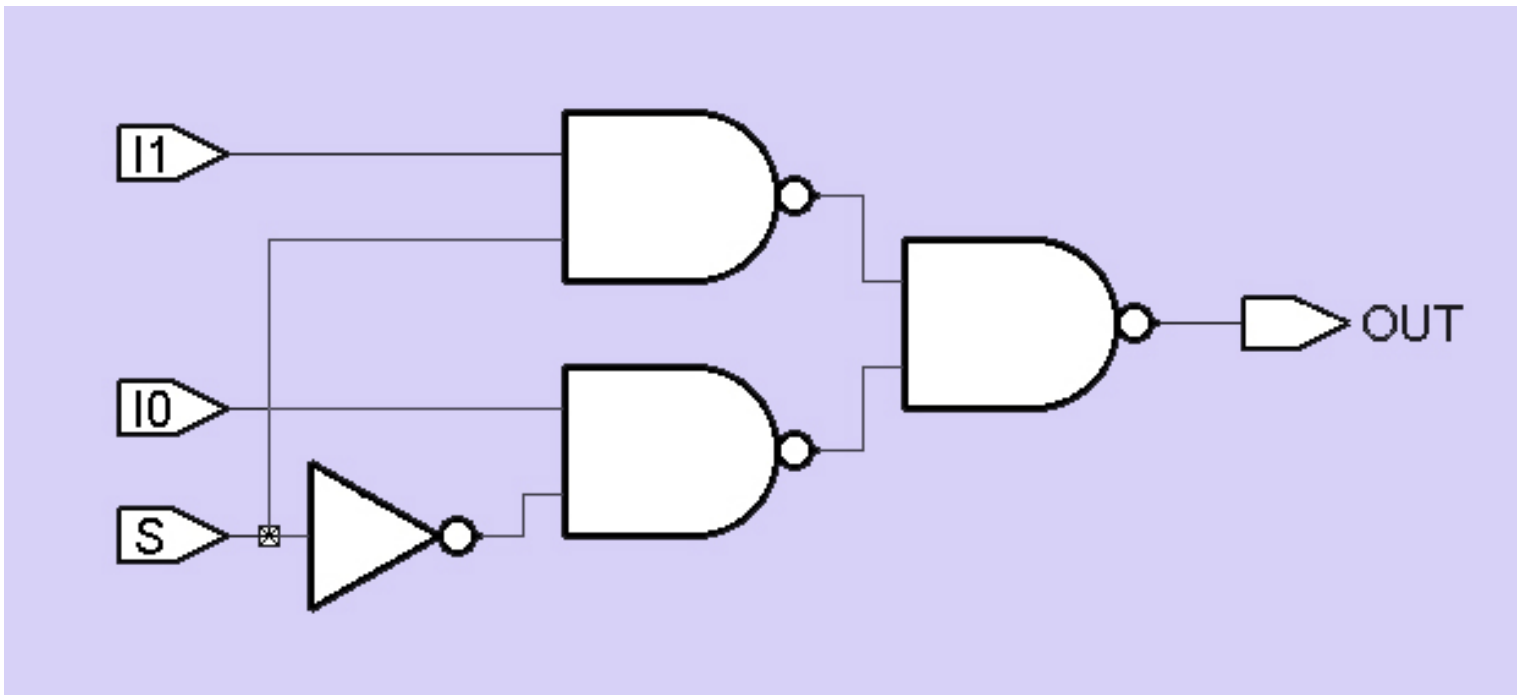
Synthese von `if/else` und `case`

TECHNISCHE UNIVERSITÄT DARMSTADT

```
if  (S) OUT = I1;
else    OUT = I0;
```

```
case (S)
   0: OUT = I0;
   1: OUT = I1;
endcase
```

Multiplexer



TECHNISCHE
UNIVERSITÄT
DARMSTADT

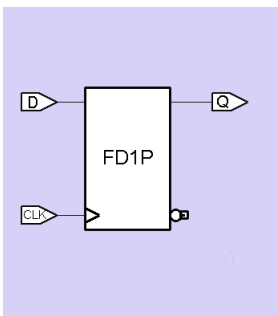
Synthese von Speicherelementen

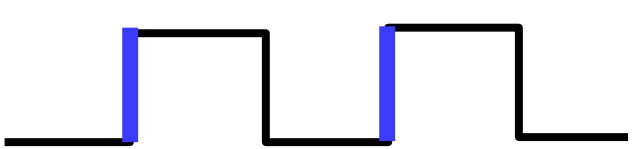
Synthese von Speicherelementen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
always @ (posedge CLK)
  Q <= D;
```

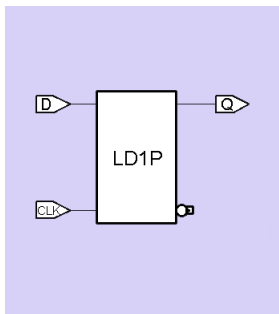
Vorderflankengesteuertes Flip-Flop

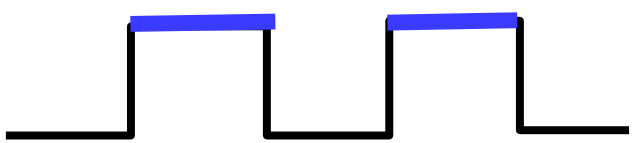




```
always @ (CLK or D)
  if  (CLK) Q = D;
```

Pegelgesteuertes Latch

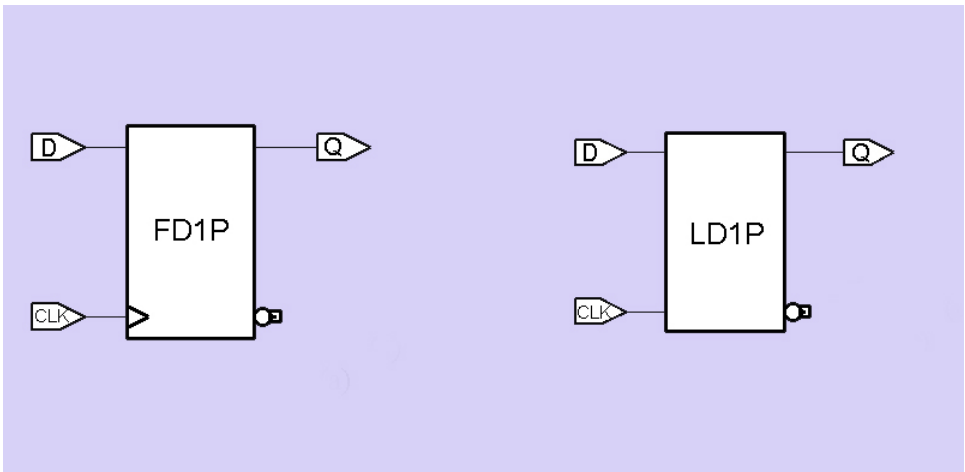




Wichtig: Unterschiede

TECHNISCHE UNIVERSITÄT DARMSTADT

- Hardware-Register
  - (flankengesteuertes) Flip-Flop
  - (pegelgesteuertes) Latch



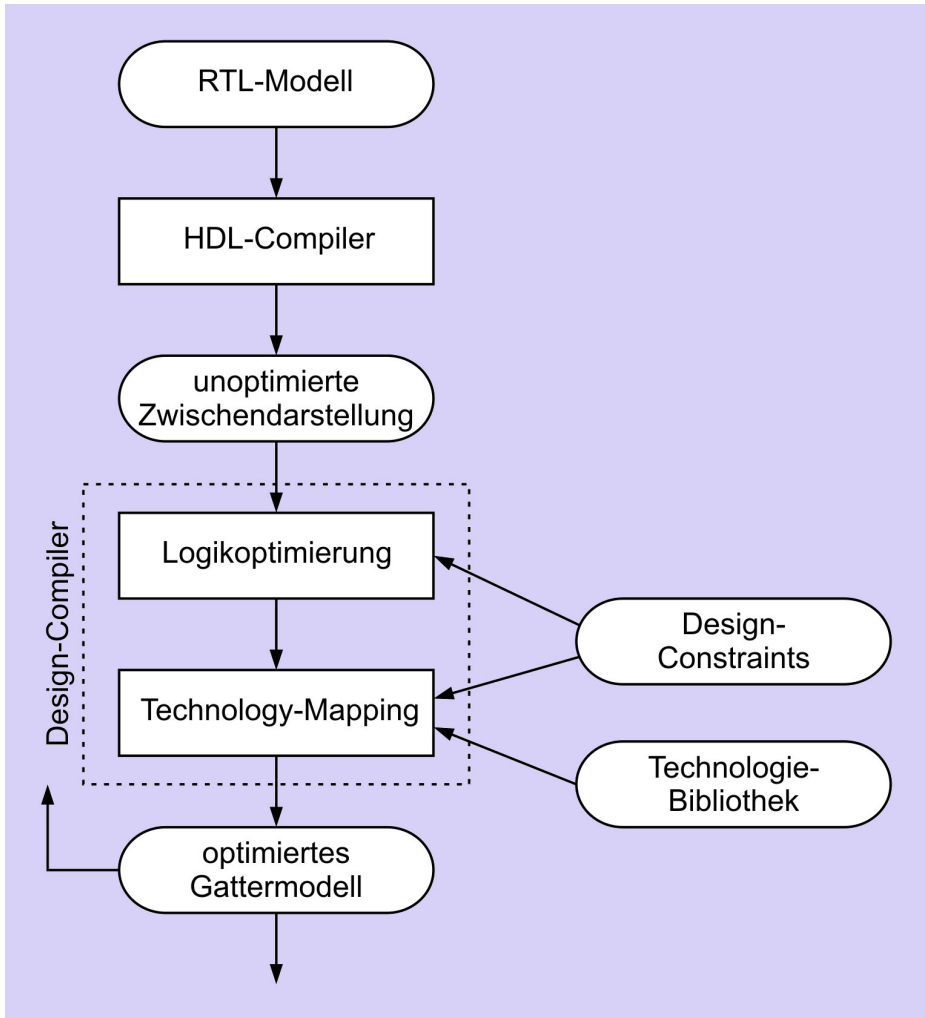
- Verilog-Datentyp `reg`

➥Nicht identisch

Verfeinerter Ablauf der Synthese

Verfeinerter Entwurfsablauf der Synthese

TECHNISCHE UNIVERSITÄT DARMSTADT

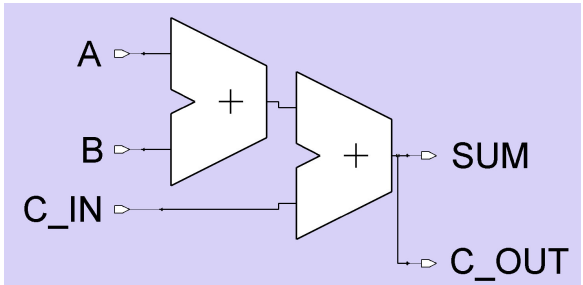


Syntheseablauf: Technologieabbildung

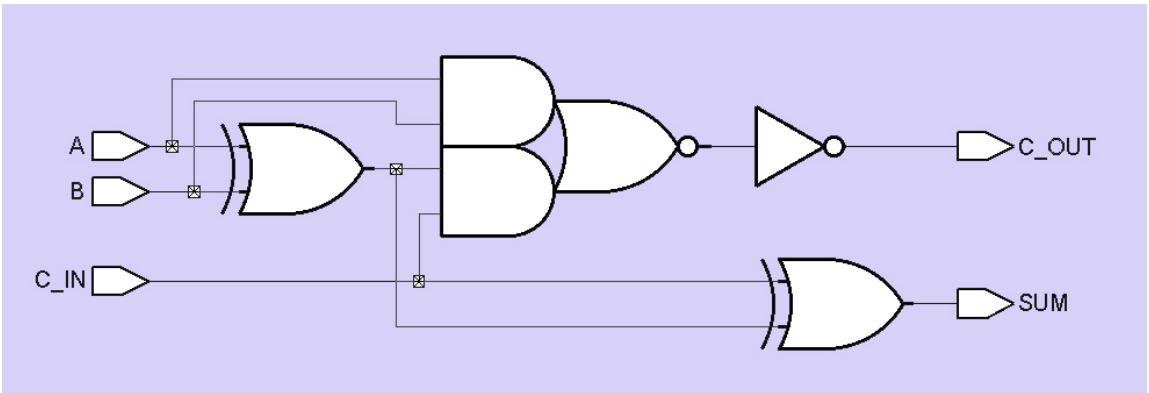
technology mapping

TECHNISCHE UNIVERSITÄT DARMSTADT

Unoptimierte Zwischendarstellung eines 1b-Addierers



Ergebnis für ASIC-Zielbibliothek (LSI 10K)

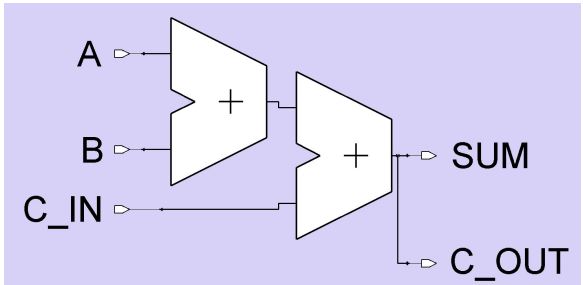


Syntheseablauf: Technologieabbildung

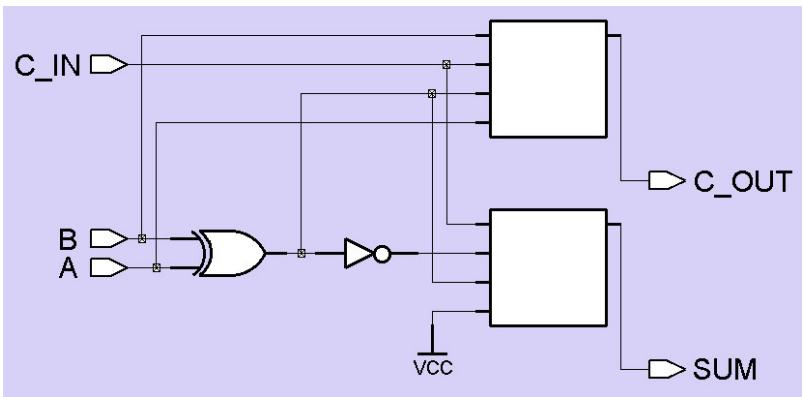
technology mapping

TECHNISCHE UNIVERSITÄT DARMSTADT

Unoptimierte Zwischendarstellung eines 1b-Addierers



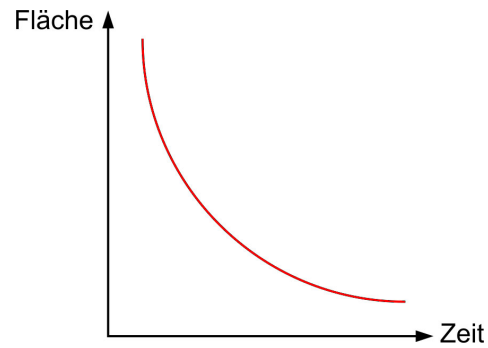
Ergebnis für FPGA (Xilinx XC4000)



Design-Constraints

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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

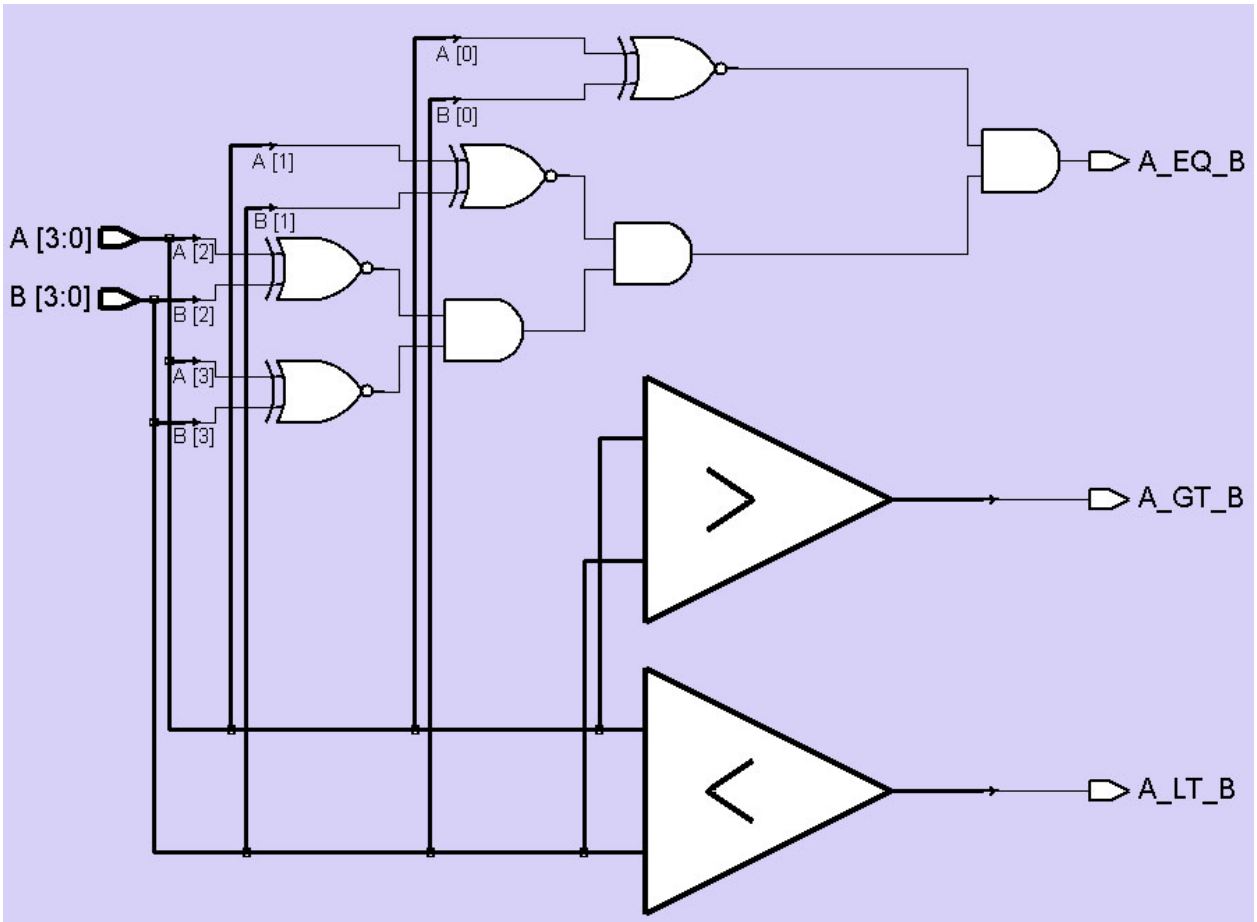
TECHNISCHE UNIVERSITÄT DARMSTADT

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

```
// Vergleicher
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

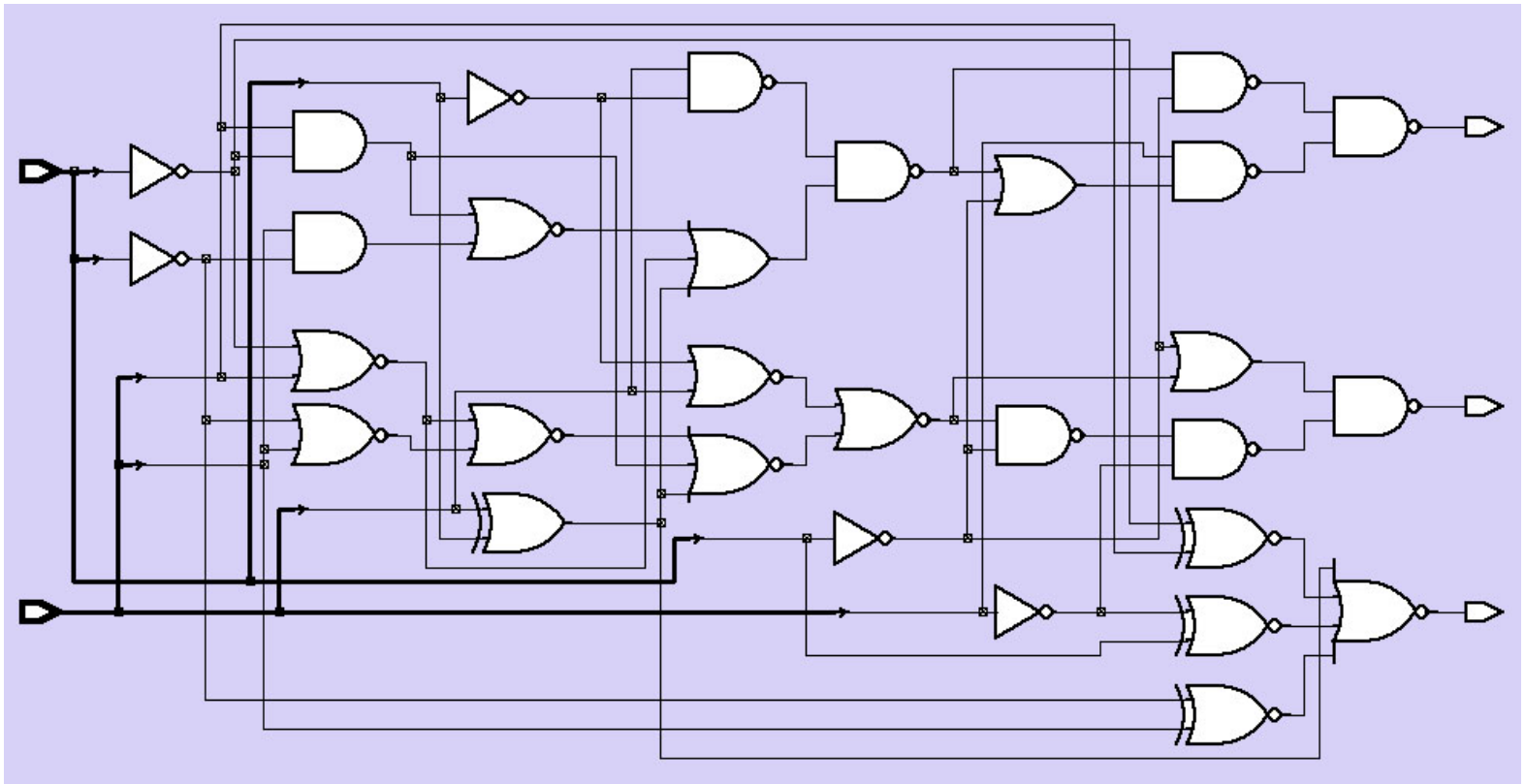
TECHNISCHE UNIVERSITÄT DARMSTADT



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

Benutzt um 1993

TECHNISCHE UNIVERSITÄT DARMSTADT

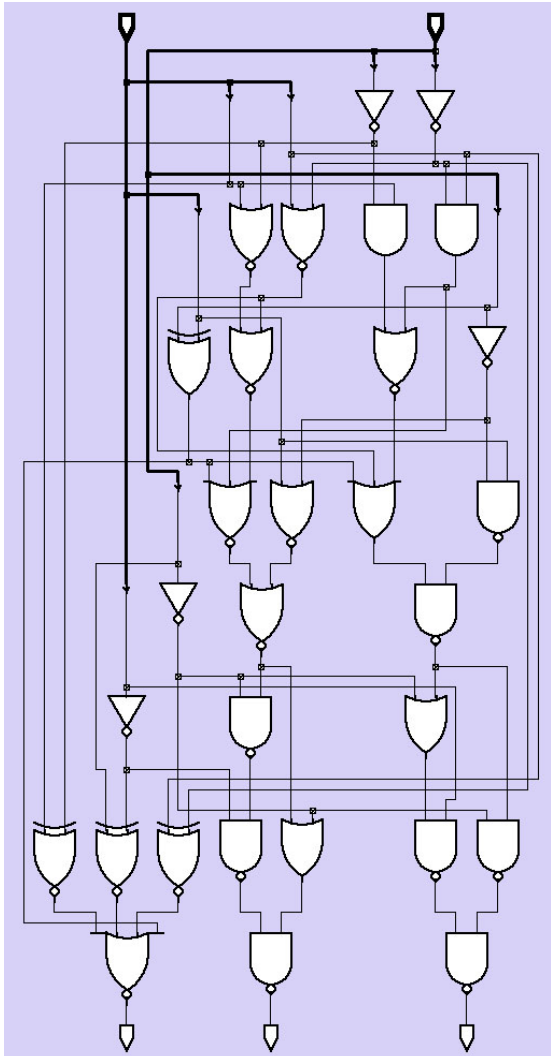


Gatternetzliste des 4b-Vergleichers

Als strukturelles Verilog

TECHNISCHE UNIVERSITÄT DARMSTADT

```
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
```



Schlüssellochblick in Technologie-Bibliothek

an2: AND-Gatter mit zwei Eingängen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
// 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 KCMS 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
```

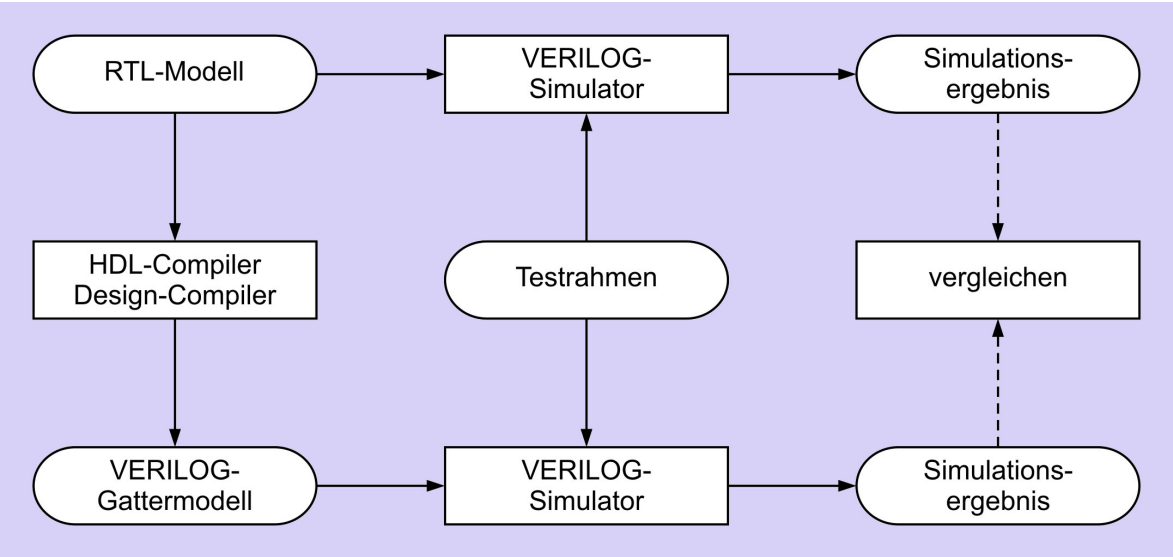
TECHNISCHE
UNIVERSITÄT
DARMSTADT

Verifikation

Verifikation

Verhält sich synthetisierte Schaltung noch so wie Simulations-modell?

TECHNISCHE UNIVERSITÄT DARMSTADT



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

Test des 4b-Vergleichers: Prä-Synthese

TECHNISCHE UNIVERSITÄT DARMSTADT

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

Diskussion: Prä- ./. Post-Synthese-Simulation

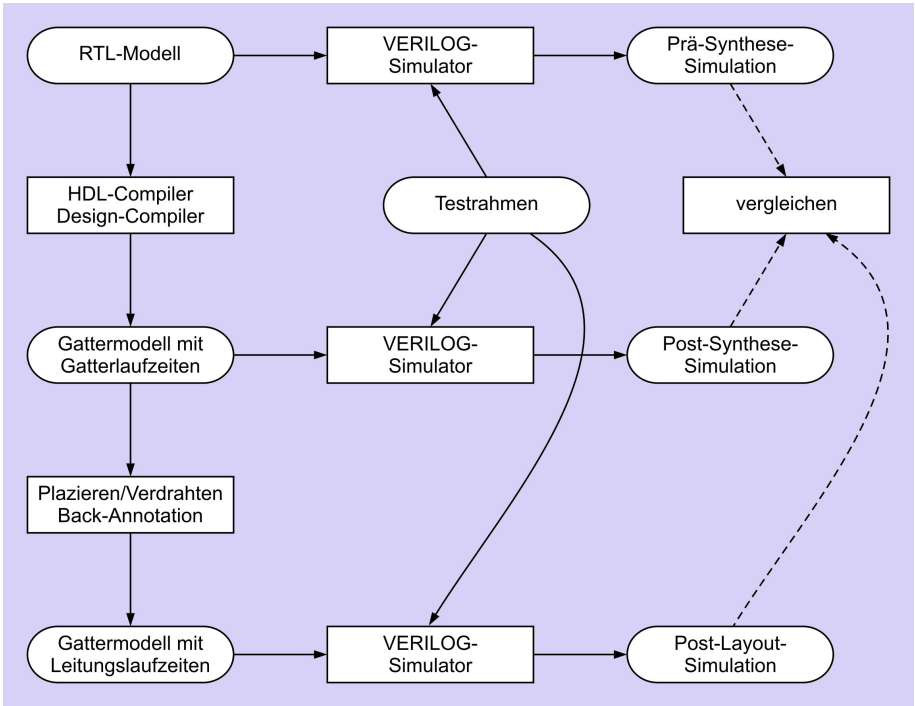
TECHNISCHE UNIVERSITÄT DARMSTADT

- 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

TECHNISCHE UNIVERSITÄT DARMSTADT



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

Synthesebeispiel: Zero-Counter

Synthesebeispiel: Zero-Counter

TECHNISCHE UNIVERSITÄT DARMSTADT

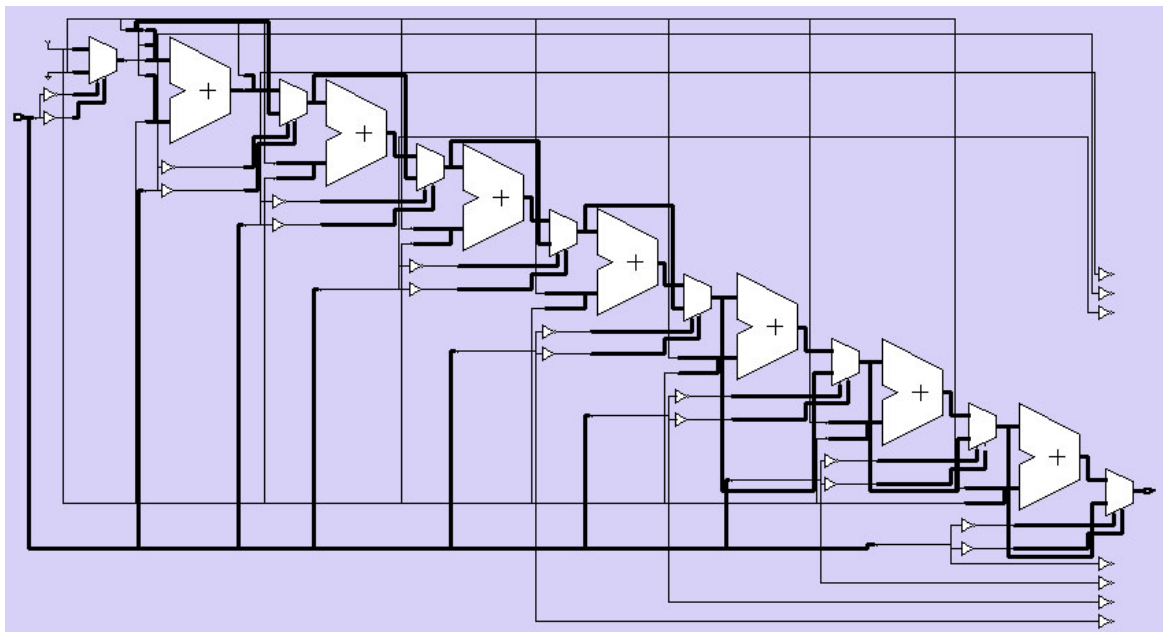
- Spezifikation
  - Eingabe ist ein 8b Datenwort
  - Ausgabe soll sein die Anzahl der Null-Bits in der Eingabe
- Genauer betrachtet
  - RTL-Modell kann Synthese-Ergebnis direkt beeinflussen
  - Wirkung von Design-Constraints
- Nicht mehr so relevant
  - Konkrete Umsetzung in Gatter-Modell
  - Bei größeren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

Zero-Counter: Intuitive Lösung

TECHNISCHE UNIVERSITÄT DARMSTADT

```
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

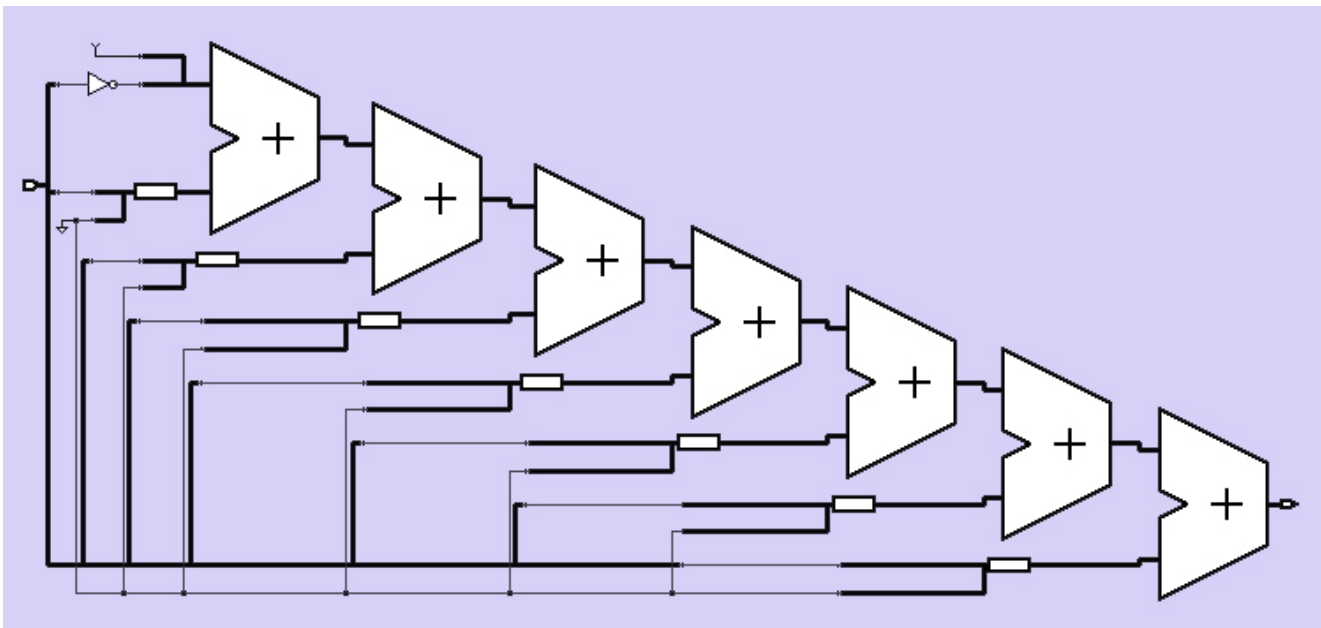


Zero-Counter: Vereinfachte Lösung

TECHNISCHE UNIVERSITÄT DARMSTADT

```
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

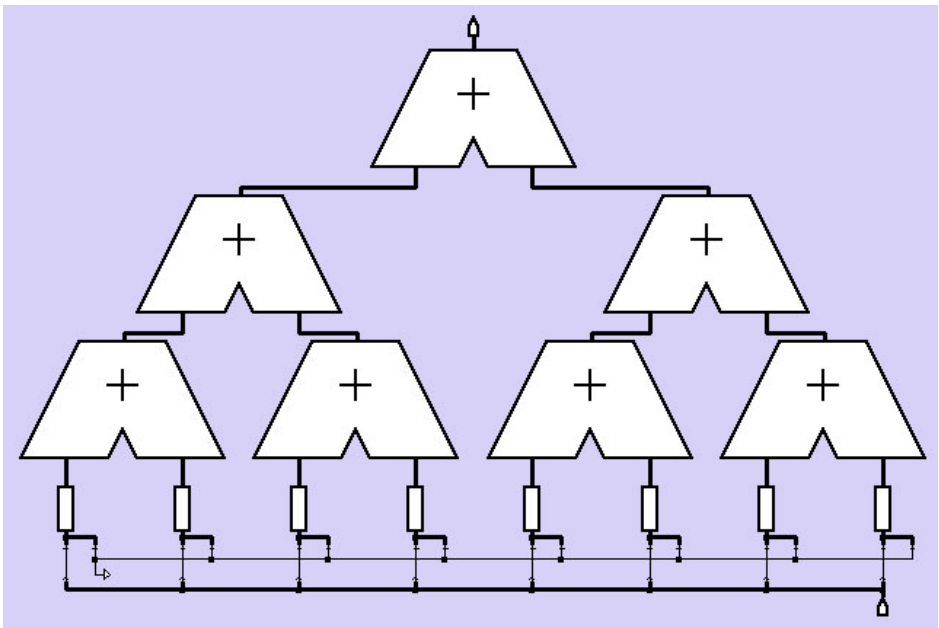


Zero-Counter: Schlaue Lösung

TECHNISCHE UNIVERSITÄT DARMSTADT

```
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



Einfluß von Design-Constraints

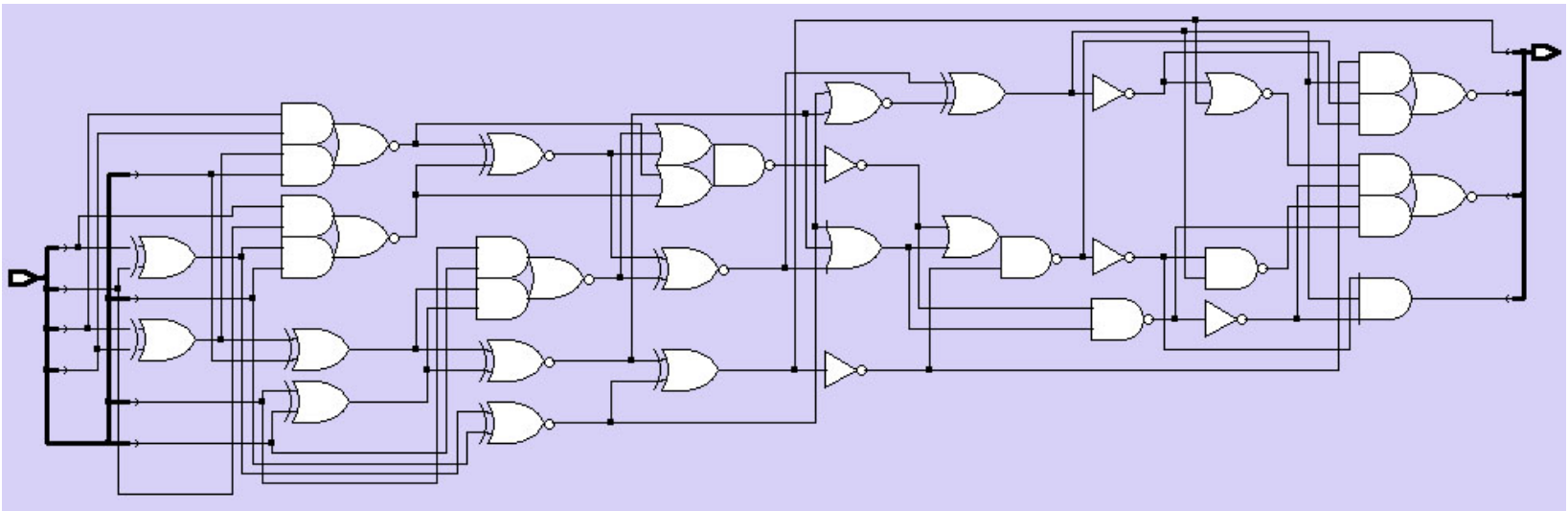
TECHNISCHE UNIVERSITÄT DARMSTADT

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

Optimierung auf Fläche

8b-Zero-Counter

TECHNISCHE UNIVERSITÄT DARMSTADT

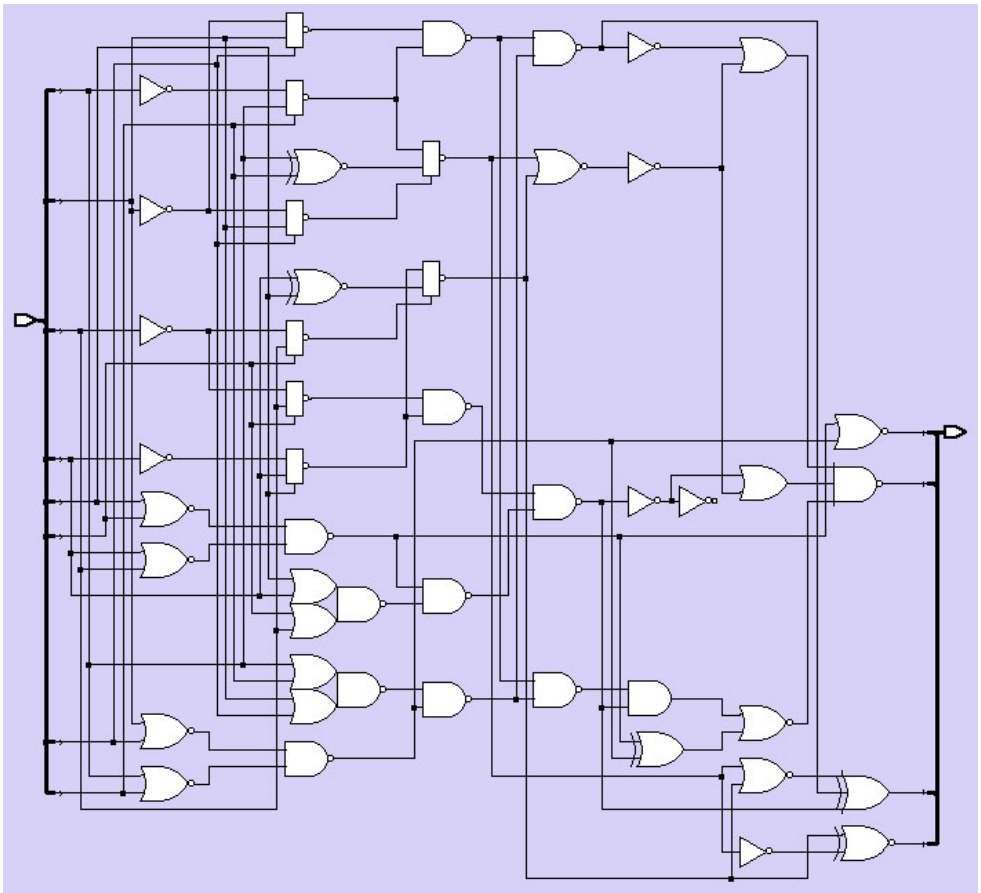


52 Gatter, 17.8ns Verzögerung

Optimierung auf Geschwindigkeit

8b-Zero-Counter

TECHNISCHE UNIVERSITÄT DARMSTADT



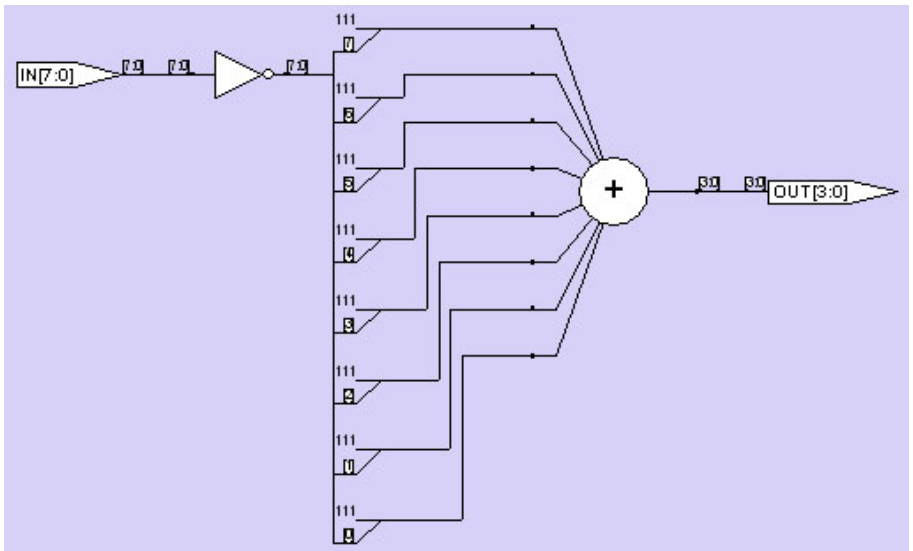
99 Gatter, 8.2ns Verzögerung

Zero-Counter: Noch bessere Lösung

TECHNISCHE UNIVERSITÄT DARMSTADT

```
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 einzelnem 8b-Addierer



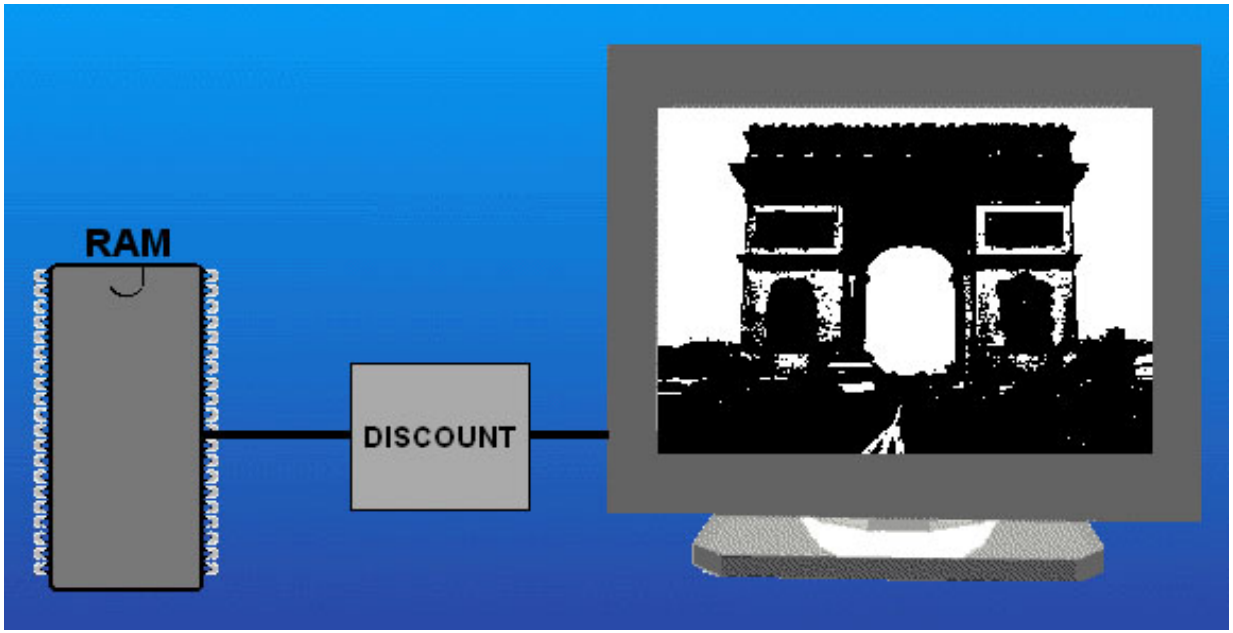
TECHNISCHE
UNIVERSITÄT
DARMSTADT

On-Chip/Off-Chip Speicher

Beispiel: Display-Controller DISCOUNT

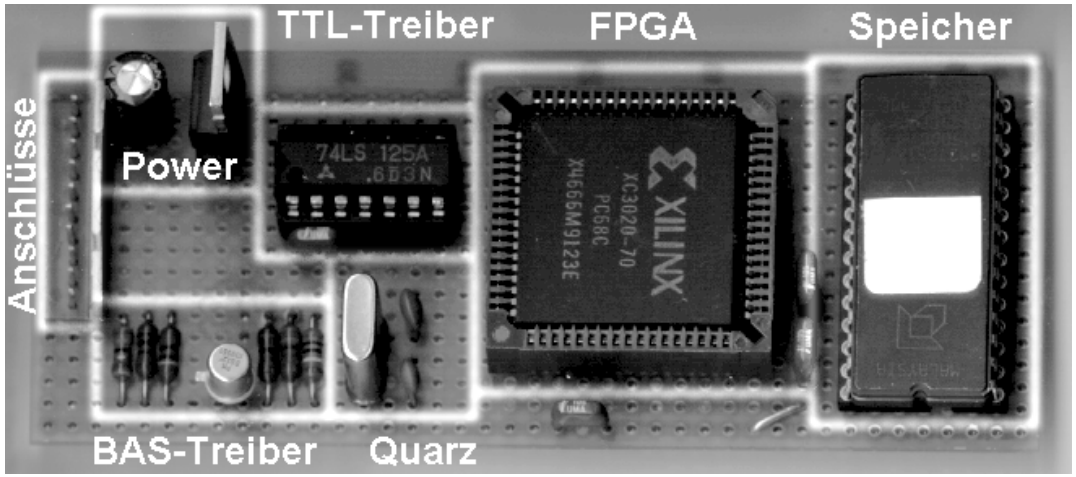
Genauer in CMS 2012 diskutiert

TECHNISCHE UNIVERSITÄT DARMSTADT



- Ausgabe von Schwarzweißbildern auf Monitor
- Genauer:
  - Bild ist in Speicher (15.360 Bytes) abgelegt
  - DISCOUNT generiert daraus Videosignale

Bildspeicher



- Bisherige Annahme: Externer ROM-Chip
- Nur einfaches Verhalten simuliert

```
// Bildspeicherdaten auslesen: Hier kombinatorisches ROM!
always @(ADDR) begin
  DATA = MEM[ADDR];
end
```

- Nun Idee: ROM direkt in DISCOUNT integrieren

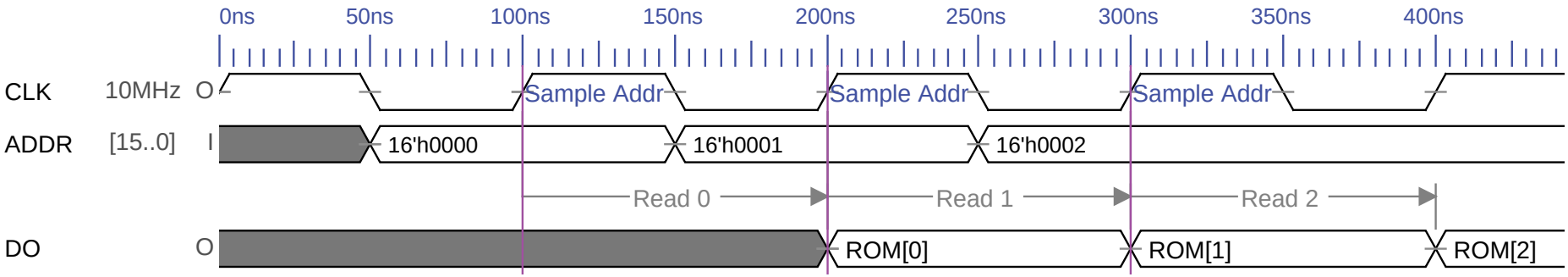
Schrittweises Vorgehen

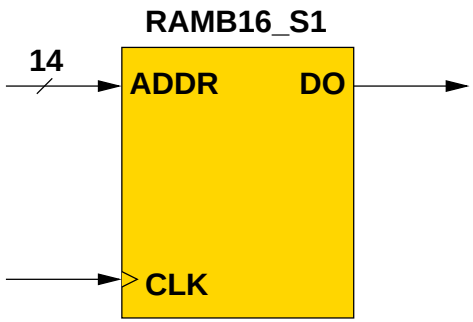
Zunächst nur in Simulation behandeln

TECHNISCHE UNIVERSITÄT DARMSTADT

- Feld aus 15.360*8 = 122.880 Registern?
- Nein, viel zu ineffzient
- Ersetze Pseudo-ROM durch echte Hardware-Modelle
- Werden vom FPGA/ASIC-Hersteller zur Verfügung gestellt
- Unsere Anforderungen
  - Platz für 15.360 Bytes
  - 14b Adressen
  - 8b Datenausgang
- Verfügbar auf Ziel-FPGA
  - Ramblöcke mit Platz für je 16.384 Bits
  - 14b Adressen
  - 1b Datenausgang
- Passt nicht ganz . . .

Trotzdem genauer anschauen: RAMB16_S1





- Adressen werden zur steigenden Flanke ausgewertet
  - Unterschied zum kombinatorischen Pseudo-ROM
- Ausgang `DO` liefert adressierte Daten nach nächster steigender Flanke

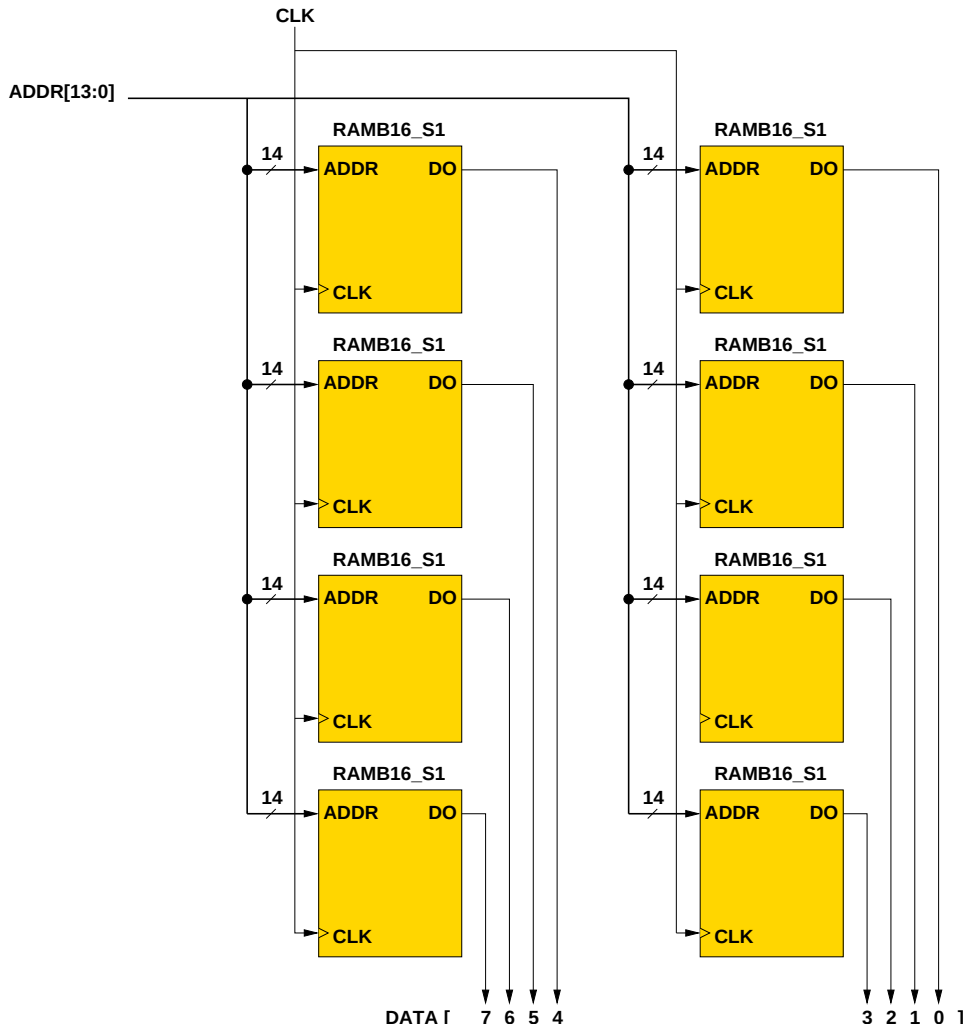
Details

TECHNISCHE UNIVERSITÄT DARMSTADT

- Ein ROM-Block namens RAMB16_S1 ??
- Der Block kann wohl noch mehr
- Hier wirklich nur als ROM benutzt
- Dazu verschiedene zusätzliche Steuersignale auf feste Werte legen
  - `EN=1'b1`, Block ist immer aktiv
  - `WE=1'b0`, niemals schreiben (soll ja ein ROM sein!)
  - `SSR=1'b0`, kein gesonderter Reset-Wert (Daten stehen im ROM)
- Aber nach wie vor
  - Ausgang `DO` ist nur 1b breit, wir brauchen aber 8b

Verschaltung zum einem 16K x 8b ROM

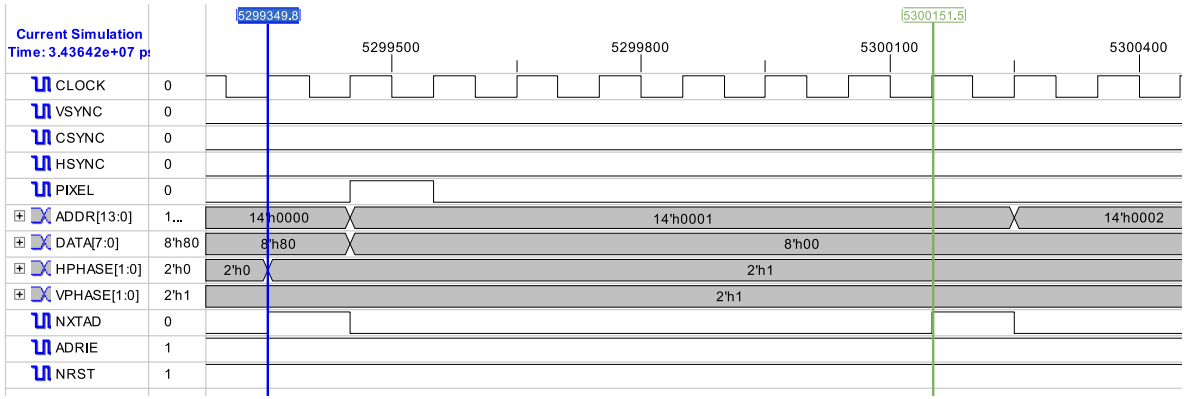
TECHNISCHE UNIVERSITÄT DARMSTADT



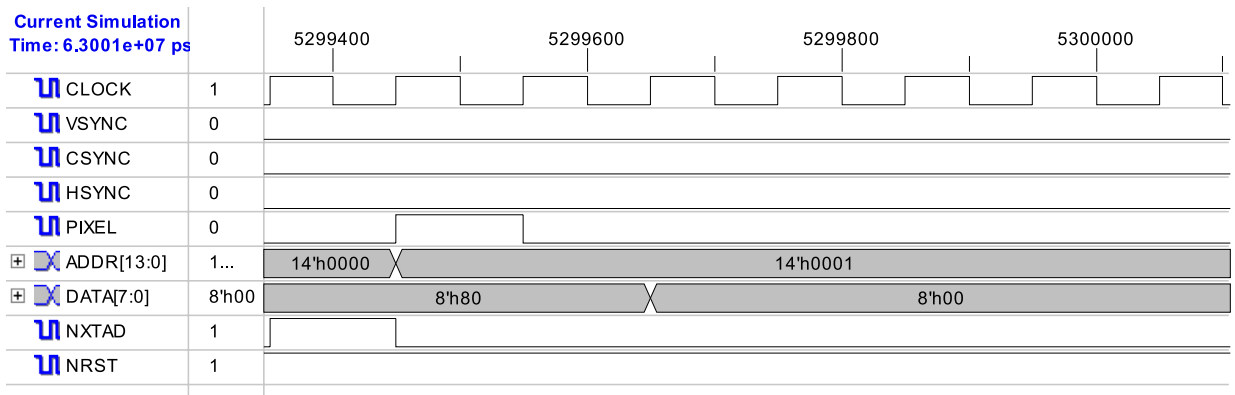
Zeitverhalten Pseudo-ROM ./. echtes ROM

TECHNISCHE UNIVERSITÄT DARMSTADT

Pseudo-ROM: Änderungen an ADDR ändern sofort DATA



Echtes ROM: Änderungen an ADDR ändern DATA verzögert



Einbindung des echten ROMs in Verilog

Kapseln innerhalb eines eigenen Moduls

TECHNISCHE UNIVERSITÄT DARMSTADT

```
// Bildspeicher aus Xilinx-ROM-Blöcken

`include "discount_defs.v"

module rom(
  input          CLK, // Clock
  input          NRST, // Reset#
  input  [`Asz-1:0] ADDR, // 14-bit Address Input
  output [`Bsz-1:0] DATA // 8-bit Data Output
);

  integer i;
```

Erstellen des Feldes aus acht `RAMB16_S1`

Statt Copy & Paste schlaueres Konstrukt

TECHNISCHE UNIVERSITÄT DARMSTADT

```
// Instanziere 8 RAMB16_S1 namens INST[0].ROMBLOCK ... INST[7].ROMBLOCK
generate
  genvar j;
    for (j = 0; j < 8; j = j + 1) begin: INST
      RAMB16_S1 ROMBLOCK(
                         .DO(DATA[j]), // bitweiser Anschluß
                         .ADDR(ADDR),
                         .CLK(CLK),
                         .EN(1'b1),    // Enable, immer aktiv
                         .WE(1'b0),    // Nie schreiben
                         .SSR(1'b0)    // ROM nicht zuruecksetzen
                        )
    end
endgenerate
```

- Erzeugt genau das erforderliche Feld
- Präfix (hier `INST`) vermeidet Namenskonflikte
- Nicht durch `for` ersetzbar
  - `for` kann keine Instanziierungen vornehmen
  - Sondern enthält nur prozeduralen Code

Modellierung von Speichern

RAM Speicher

TECHNISCHE UNIVERSITÄT DARMSTADT

- RAM (*random access memory*) ist i.d.R. flüchtig
- Ausnahme: *non-volatile RAM* (NVRAM), z.B. FeRAM, MRAM, PCRAM, ...
  - nicht: Flash, EPROM, EEPROM: Schreiben dauert deutlich länger als lesen
  - bei Flash: 0.5 s (NAND Flash), 1 s (NOR Flash) pro 128 KB Block
- Unterschiedliche Arten von flüchtigen RAM Speichern ($\rightarrow$ TGDI)
  - Statisches RAM (SRAM)
  - Dynamisches RAM (DRAM)

RAM Varianten

TECHNISCHE UNIVERSITÄT DARMSTADT

- asynchrone und synchrone RAMs
  - SRAM vs. SSRAM
  - DRAM vs. SDRAM)
- *double data-rate* (DDR)
  - Daten werden bei steigender und fallender Taktflanke übertragen
  - z.B. DDR3-SDRAM
- *quad data-rate* (QDR)
  - Daten werden bei steigender und fallender Taktflanke übertragen
  - ... und jeweils hälftig *zwischen* den Taktflanken (90° phasenverschoben)
  - z.B. QDR-SSRAM

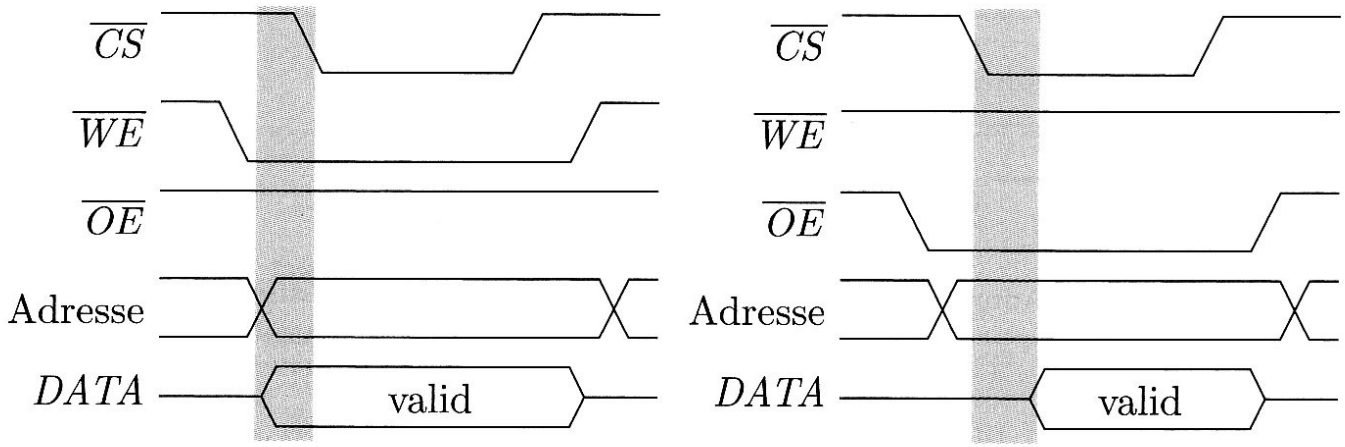
Speicherbus-Protokoll für externes SRAM

Asynchron

TECHNISCHE UNIVERSITÄT DARMSTADT

- Adressleitungen wählen Speicherzellen aus
- Datenleitungen
- Chipauswahlleitung $\overline{CS}$ (*chip select*)
- Schreibleitung $\overline{WE}$ (*write enable*)
- Leseleitung $\overline{OE}$ (*output enable*)
- $\overline{CS}$, $\overline{WE}$ und $\overline{OE}$
  - active-low, angedeutet durch die Negierung der Signalnamen
  - auch geschrieben als `nCS`, `nWE`, `nOE`
- Bei $\overline{CS}$ = 1: Ausgabeleitungen (DATA) hochohmig (Z)
  - $\overline{WE}$ und $\overline{OE}$ werden ignoriert
- Steuerung des Chips im wesentlichen über $\overline{CS}$
  - Siehe `Select` bei Adressdekodierung (kommt noch ...)

Signalverlauf: Zugriff auf asynchrones SRAM



a. Schreibzyklus

b. Lesezyklus

- Verzögerungszeiten des Speichers durch Schattierung dargestellt
- Schreiben: Adressen und Daten müssen vor der Übernahme *stabil* anliegen
  - Ähnlich Setup- und Hold-Zeiten, siehe TGDI
- Lesen: Nach Änderung von $\overline{CS}$ Verzögerung, bis gültigen Daten anliegen
- Auslesen des Speichers durch Setzen von $\overline{OE}$
- Alle Feinheiten in Datenblättern der Hersteller

Speicherbus-Protokoll für externes DRAM

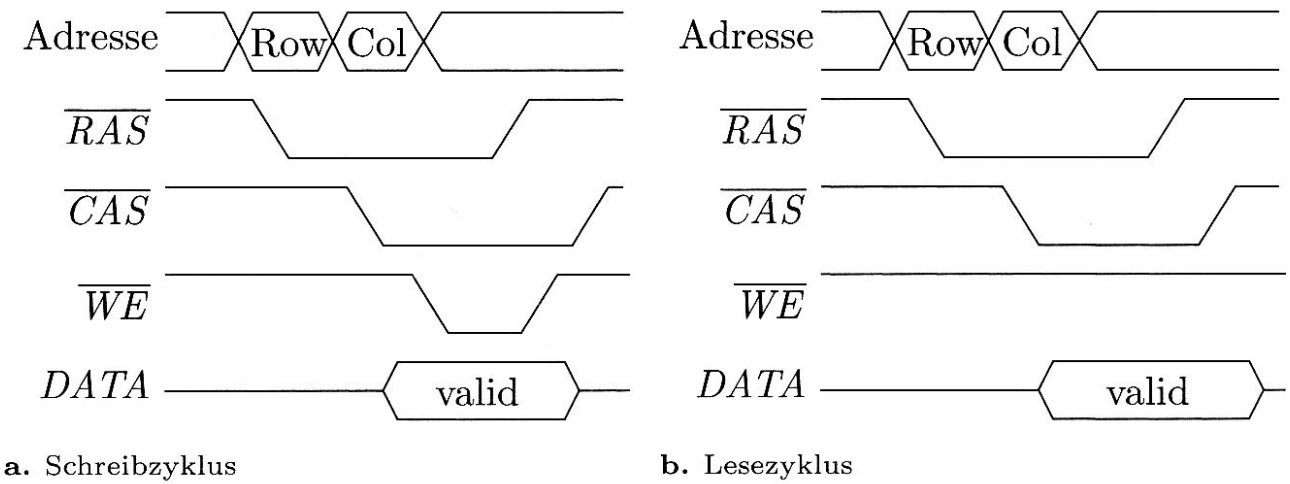
asynchron, stark vereinfacht

TECHNISCHE UNIVERSITÄT DARMSTADT

- DRAMs haben große Speicherkapazität (4 Gb = 1 G x 4 b)
- Zusammenfassung von Adressleitungen (hier: 30)
  - Spart Pins!
- Adressdaten aufteilen in Reihen- und Spaltenadressen
  - ggf. auch noch Bank-Adresse (hier nicht im Detail behandelt)
  - im 4 Gb Beispiel 16b Reihe, 11b Spalte, 3b Bank
  - $= 2^{16} \cdot 2^{11} \cdot 2^{3} = 2^{30} = 1$ G
- Reihen- und Spaltenadressen dann nacheinander übertragen
  - über den gleichen 16b Adressbus
  - Zeitmultiplex-Verfahren
- Auswahl der Art der Teiladresse
  - Row-Address-Strobe ($\overline{RAS}$) für Zeilenadresse
  - Column-Address-Strobe ($\overline{CAS}$) für Spaltenadresse

Signalverlauf: DRAM Zugriff

asynchron



a. Schreibzyklus **b.** Lesezyklus

- Hier stark vereinfacht, alle Feinheiten in Datenblättern der Hersteller

Verilog-Modell: ROM

TECHNISCHE UNIVERSITÄT DARMSTADT

```
module rom16x4 (ROM_data, ROM_addr);
 output [3:0]  ROM_data;
 input  [3:0]  ROM_addr;
 reg    [3:0]  ROM [15:0];

 assign ROM_data = ROM[ROM_addr];

 // für Simulation
  initial  $readmemh("ROM-2b-Adder.txt",ROM,0,15);
endmodule
```

angelehnt an *Ciletti, Michael D.: Advanced Digital Design with the Verilog HDL. Prentice Hall, 2003*, S. 424

Verilog-Modell: Externer SRAM-Chip

TECHNISCHE UNIVERSITÄT DARMSTADT

```
module sram16x8 (input [3:0] address,
                 input nCS, nWE, nOE,
                 inout  [7:0]  data);

 reg  [7:0]  memory [15:0]; // 16 Zellen, 8 Bit breit

 assign data = (!nCS && !nOE) ? memory[address] : 8'bZ;

 always @(nCS or nWE)
   if   (!nCS && !nWE) memory [address] = data;

endmodule
```

angelehnt an *Biere, Kroening, Weissenbacher, Wintersteiger: Digitaltechnik - Eine praxisnahe Einführung. Springer, 2008*, S. 184

Verilog-Modell: Externer DRAM-Chip

asynchron, stark vereinfacht

TECHNISCHE UNIVERSITÄT DARMSTADT

```
module dram256x8 (input [3:0] address,
                  input nRAS, nCAS, nWE, nOE,
                  inout [7:0] data);

 reg [7:0] memory [15:0][15:0]; // 16x16 Zellen, 8 Bit breit
 reg [3:0] row, column;

 assign data = (!nOE) ? memory[row][column] : 8'bZ;

 always @(negedge nRAS)
  row <= address;

 always @(negedge nCAS)
  column <= address;

 always @(negedge nWE)
  memory [row][column] <= data;

endmodule
```

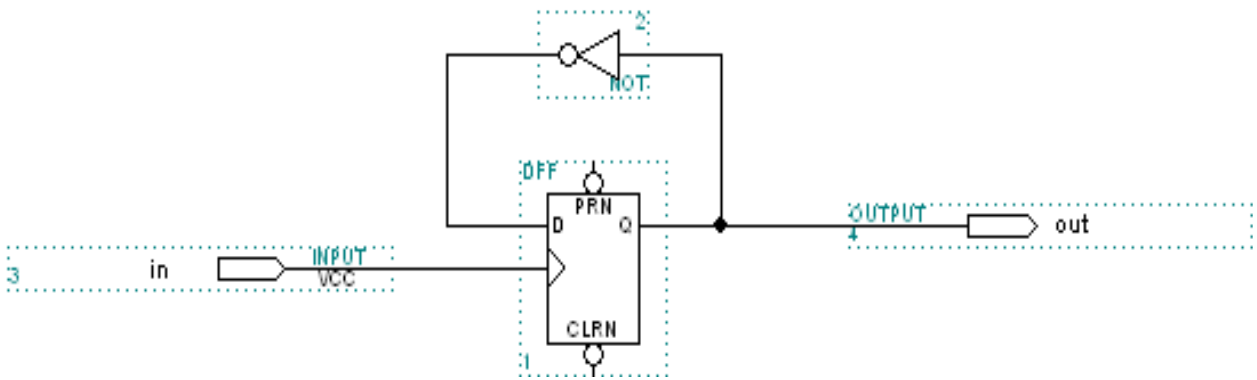
angelehnt an *Biere, Kroening, Weissenbacher, Wintersteiger: Digitaltechnik - Eine praxisnahe Einführung. Springer, 2008*, S. 185

Takterzeugung

Takterzeugung und -teilung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Takterzeugung erfolgt physikalisch z. B. durch einen Quarz
- Beispiel: Quarz hat eine Taktfrequenz von 64 *MHz*
- Was, wenn Chip nur bei 32 *MHz* läuft?
- Wie erreicht man eine Taktteilung?
- Z. B. rückgekoppeltes Flip-Flop



Komplexere Taktteilung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Quarz hat eine Taktfrequenz von 64 *MHz*
- Taktfrequenz z. B. auf 125 *kHz* teilen?
- Naive Lösung: neun rückgekoppelte Flip-Flops in Reihe schalten
- Unpraktikabel: lange Durchlaufzeit, Resourcenverschwendung an CLBs
- Bessere Lösung: Zähler mit verschiedenen Abgriffen

Taktteilung durch Zähler

TECHNISCHE UNIVERSITÄT DARMSTADT

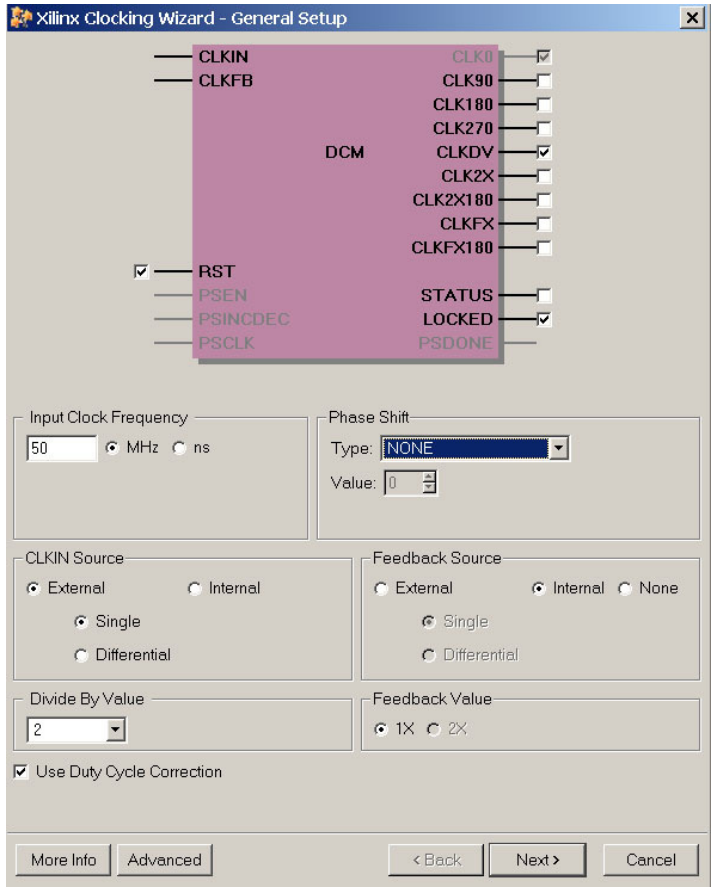
- Verschiedene Taktfrequenzen ergeben sich durch unterschiedliche Abgriffe
- Jeweils geteilt durch entsprechende Zweierpotenz

```
module taktteiler(clkin, clkout1, clkout2, clkout3);

  input clkin;
  output clkout1, clkout2, clkout3;

  reg [24:0] counter;

  assign clkout1 = counter[24];
  assign clkout2 = counter[23];
  assign clkout3 = counter[18];

  always @(posedge clkin) begin
    counter <= counter + 1;
  end

endmodule
```

Andere Teilungsverhältnisse

Digital Clock Manager (DCM) oder Phase-Locked Loop (PLL)

TECHNISCHE UNIVERSITÄT DARMSTADT



Damit z.B. auch Ausgangsfrequenz = 5 / 3 der Eingangsfrequenz

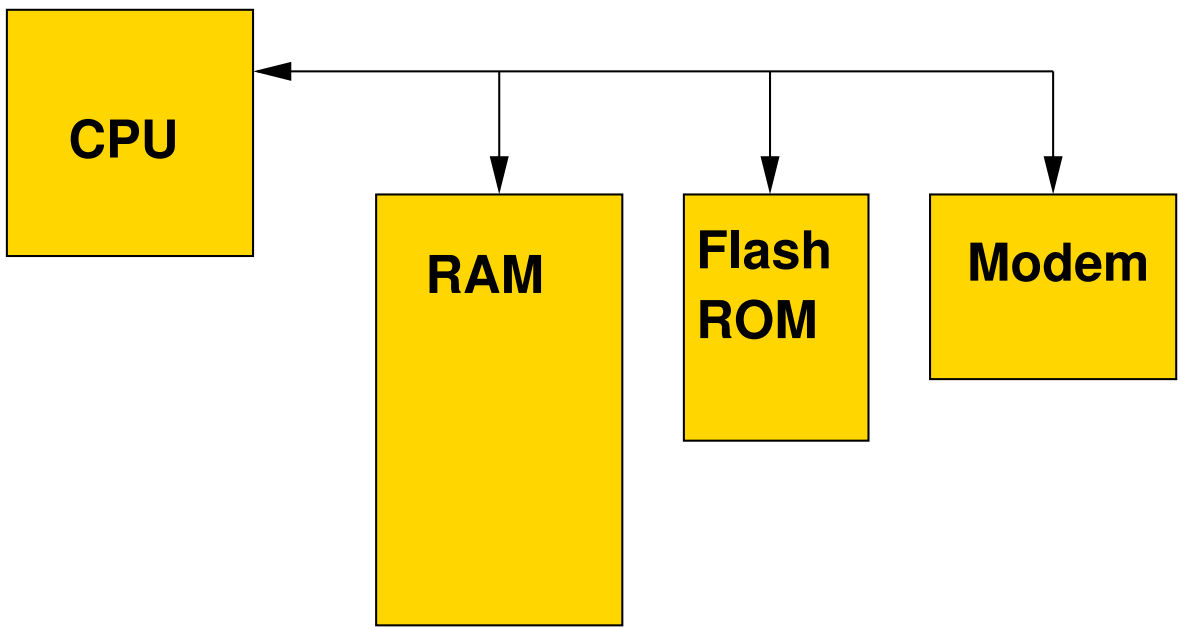
TECHNISCHE
UNIVERSITÄT
DARMSTADT

Kommunikation und Adressierung

Hardware-System

TECHNISCHE UNIVERSITÄT DARMSTADT

- Verschiedene Untereinheiten
- Kommunikation untereinander
- Auf verschiedene technische Weisen realisierbar



Häufig verwendet: Busse

TECHNISCHE UNIVERSITÄT DARMSTADT

- Gemeinsame Verbindungen zwischen Komponenten
- Maximal ein Initiator / Master gleichzeitig
  - Veranlasst Aktivitäten auf Bus
- Ein oder mehrere Targets / Slaves
  - Reagieren auf Aktivitäten auf dem Bus
- Grundlegende Transaktionen auf dem Bus
  - Initiator fordert Daten von Target an: Lesezugriff
  - Initiator überträgt Daten zum Target: Schreibzugriff
- Busorganisation in einfachen Systemen
  - Nur ein Master, häufig die CPU
- In leistungsstärkeren Systemen
  - Mehrere Master (*multi-master* Ansatz, DMA)
  - Beispiele: Gb/s-Ethernet, Festplatten-Controller, Graphikprozessor, USB 2.0...
  - Hier nicht weiter behandelt

Idee von Bussen

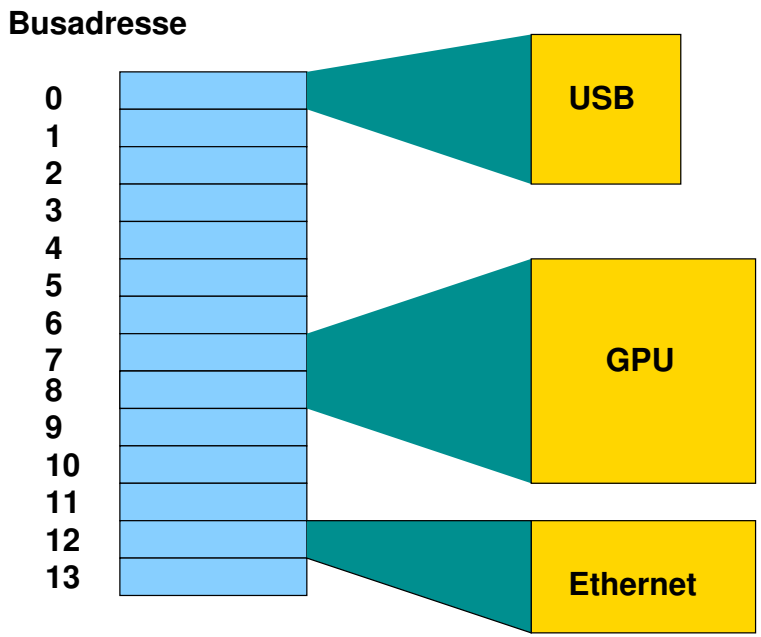
TECHNISCHE UNIVERSITÄT DARMSTADT

- Gemeinsam genutztes Medium
- Vorteile
  - Einfache Realisierung
  - Wenig Chip-Fläche
- Nachteile
  - Nur eine Verbindung gleichzeitig
    - Vereinfacht, kann etwas verbessert werden
    - $\rightarrow$ Disconnect/Reconnect, Split Transaction, ...
  - Probleme bei Bussen
    - Wie Bus vergeben, wenn mehrere Initiator/Master gleichzeitig Zugriff benötigen? $\rightarrow$ nicht weiter behandelt
    - Wie Target gezielt ansprechen? $\rightarrow$ kommt jetzt

Busadressen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Vergabe von Adressen für Teilnehmer an Bus
- Können auch sehr große Bereiche sein
  - Beispiel: Moderne GPU braucht ca. 2 GB an Adressraum



Buszugriff von CPU als Master

TECHNISCHE UNIVERSITÄT DARMSTADT

- CPU kennt schon Adressen
- Nämlich für den Speicher
- Wie damit Buszugriffe realisieren?
- Zwei Alternativen

1. Getrennter Adressbereich für Bus

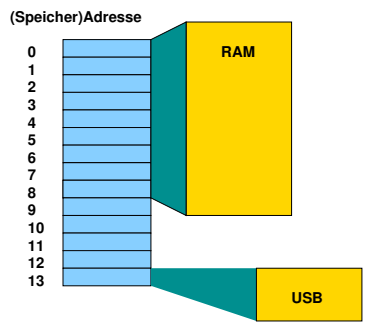
TECHNISCHE UNIVERSITÄT DARMSTADT

- Häufig Ein-/Ausgabe-Adressbereich genannt
- Verwendet z.B. bei älteren X86-Architekturen
- Spezielle Instruktionen für Ein-/Ausgabe
- Interpretieren angegebene Adresse immer als Busadresse
- Beispiel: `outb %al, 0x60`
  - Schreibt Inhalt des `AL`-Registers (1 Byte) an Busadresse 0x60
- Beispiel: `inb 0x71, %al`
  - Liest 1 Byte von Busadresse 0x71 in `AL` Register
- Beispiele für die klassische PC-Architektur
  - Tastatur liegt auf 0x60 und 0x64
  - 1. Serielle Schnittstelle liegt auf 0x3f8-3ff
  - 1. Festplatten-Controller liegt auf 0x1f0-1f7 und 0x3f6-3f7

2. Gemeinsamer Adressbereich

TECHNISCHE UNIVERSITÄT DARMSTADT

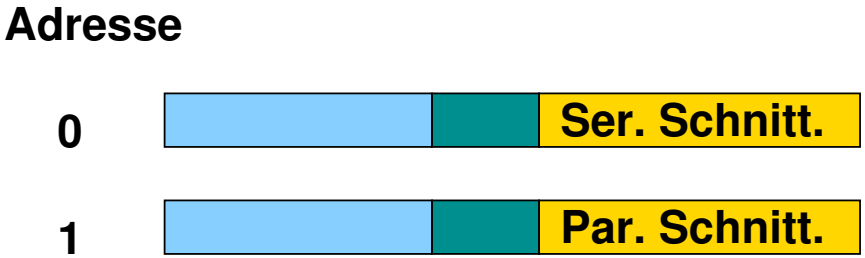
- Nachteil des getrennten Adressbereichs
  - Zusätzliche Instruktionen
  - Ungünstig, Instruktionsbits sind rares Gut
- Anderer Ansatz: Blende Busadressen in normalen Adressraum ein: memory mapped I/O
  - So bei den meisten anderen Prozessoren
  - Auch bei X86 möglich, heute überwiegend verwendet
- Zugriff nun mit ganz normalen Lade/Speicherbefehlen
  - `lw $t0, 4`: Lade 32b Wort aus RAM-Speicher
  - `sb $t1, 13`: Gebe Byte auf USB-Schnittstelle aus

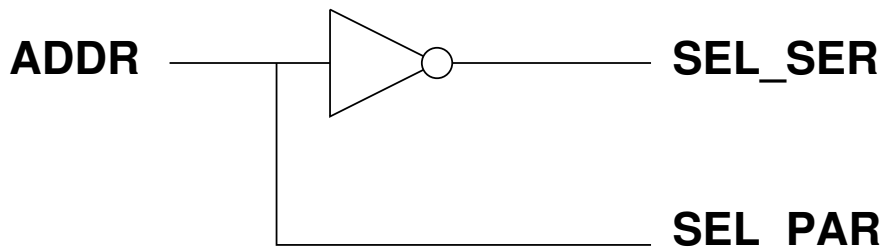


Adressdekodierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Problem: Wie erkennen Busteilnehmer, ob sie angesprochen werden?
  - Also Target / Slave eines Zugriffs sind
- Gängige Lösung
  - Select-Signal pro Teilnehmer
  - Wird aktiviert, wenn Teilnehmer vom Initiator/Master angesprochen wird
- Adressdekodierung übersetzt Busadressen in Select-Signale
- Triviales Beispiel: Zwei Teilnehmer auf Bus
  - Auf Adressen 0 und 1





Komplizierteres Szenario

Annahme: 64KB Adressraum

TECHNISCHE UNIVERSITÄT DARMSTADT

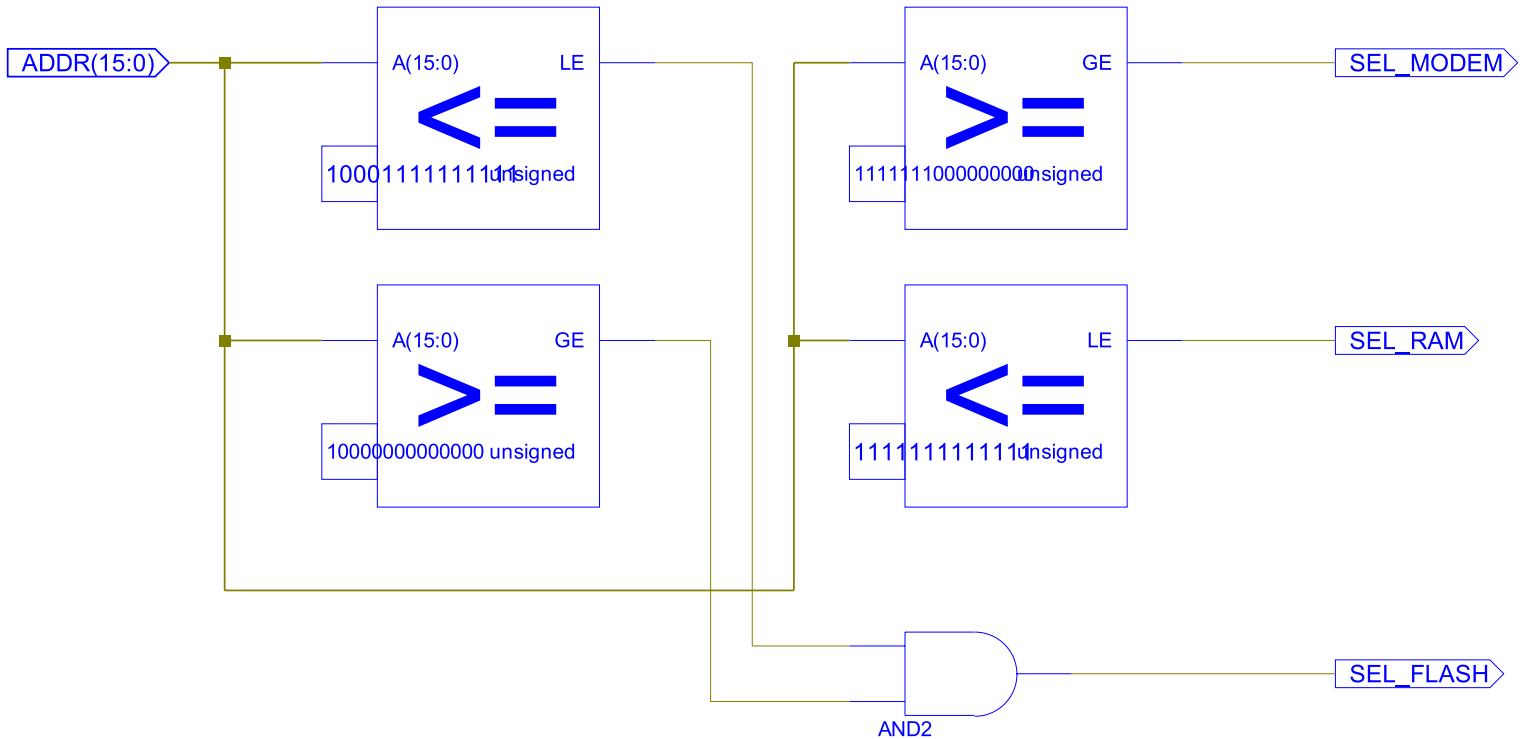
Startaddresse	Endaddresse	Teilnehmer
0x0000	0x1FFF	RAM 8KB
0x2000	0x23FF	Flash-Speicher 1KB
0xFE00	0xFFFF	Modem 512B

Wie realisierbar? Erster Versuch:

```
module decoder1 (
  input [15:0]  ADDR,
  output        SEL_RAM, SEL_FLASH, SEL_MODEM
);

  assign SEL_RAM    = (ADDR >= 16'h0000 && ADDR <= 16'h1FFF);
  assign SEL_FLASH  = (ADDR >= 16'h2000 && ADDR <= 16'h23FF);
  assign SEL_MODEM  = (ADDR >= 16'hFE00 && ADDR <= 16'hFFFF);

endmodule
```

Diskussion: Syntheseergebnis

TECHNISCHE UNIVERSITÄT DARMSTADT



- Optimiert: Immerhin nur vier statt sechs Vergleicher
- Jeder einzelne Vergleicher aber groß und langsam!

Neue Überlegung

TECHNISCHE UNIVERSITÄT DARMSTADT

Müssen wir immer alle Bits vergleichen?

Startaddresse	Endaddresse	Teilnehmer
16'b0000_0000_0000_0000	16'b0001_1111_1111_1111	RAM 8KB
16'b0010_0000_0000_0000	16'b0010_0011_1111_1111	Flash-Speicher 1KB
16'b1111_1110_0000_0000	16'b1111_1111_1111_1111	Modem 512B

- Nein, versuche nur Startadressen eindeutig zu unterscheiden
- Mit möglichst wenigen Bits!
- Beginne mit höchstwertigen (=linken) Bits
  - Vermeide so falsche Erkennung von Adressbereichen
  - Gegenbeispiel: Verwende Bit 9, um Modem (=1) und Flash (=0) auseinanderzuhalten
  - Klappt nicht: Bei Zugriff auf RAM kann Bit 9 auch 1 werden!
  - Bereiche dürfen sich nicht überlappen
  - Niemals gleichzeitig mehr als ein Select-Signal aktiv

Nächster Versuch ...

Konstruiere Entscheidungsbaum von links nach rechts

TECHNISCHE UNIVERSITÄT DARMSTADT

Startaddresse	Endaddresse	Teilnehmer
16'b0000_0000_0000_0000	16'b0001_1111_1111_1111	RAM 8KB
16'b0010_0000_0000_0000	16'b0010_0011_1111_1111	Flash-Speicher 1KB
16'b1111_1110_0000_0000	16'b1111_1111_1111_1111	Modem 512B

- `ADDR[15]==1` $\rightarrow$ Zugriff auf Modem
- `ADDR[15]==0 && ADDR[13]==0` $\rightarrow$ Zugriff auf RAM
- `ADDR[15]==0 && ADDR[13]==1` $\rightarrow$ Zugriff auf Flash

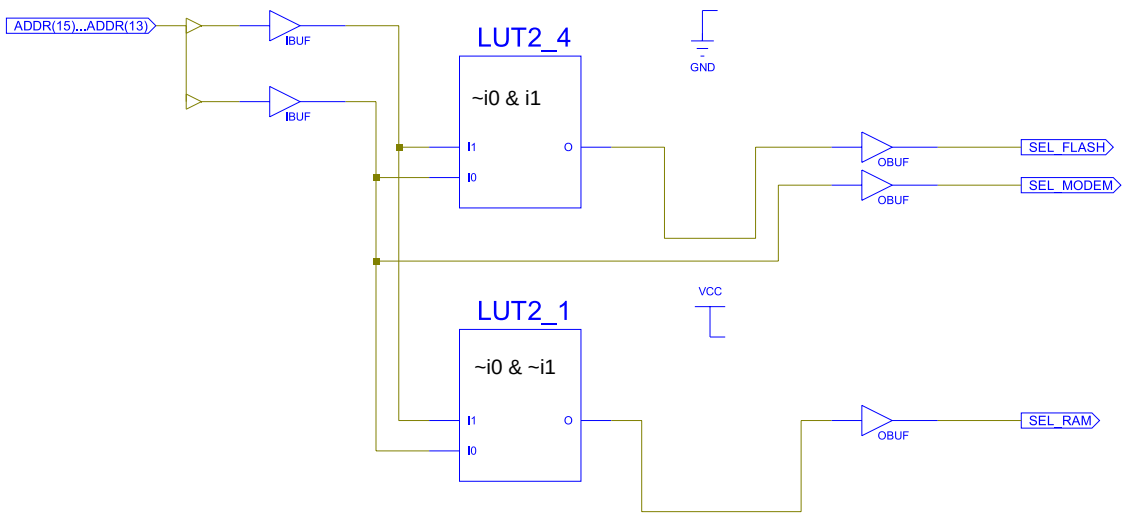
Diskussion

- Überlappungsfrei
- Select-Signal wird korrekt 1 bei Zugriff auf Startaddresse und folgende Adressen
- Aber was ist mit der Endaddresse? $\rightarrow$ später!

Verilog-Implementierung

TECHNISCHE UNIVERSITÄT DARMSTADT

```
module decoder2 (
  input [15:0]  ADDR,
  output        SEL_RAM, SEL_FLASH, SEL_MODEM
);

assign SEL_RAM     = ~ADDR[15] & ~ADDR[13];
assign SEL_FLASH   = ~ADDR[15] &  ADDR[13];
assign SEL_MODEM =  ADDR[15];

endmodule
```



Bluespec Implementierung

TECHNISCHE UNIVERSITÄT DARMSTADT

```
interface Decoder2;
  method Bool isEnabledRAM    (Bit#(16) addr);
  method Bool isEnabledFLASH  (Bit#(16) addr);
  method Bool isEnabledMODEM  (Bit#(16) addr);
endinterface

module mkDecoder (Decoder2);

  method Bool isEnabledRAM (Bit#(16) addr);
    return (addr[15] == 0 && addr[13] == 0);
  endmethod

  method Bool isEnabledFLASH (Bit#(16) addr);
    return (addr[15] == 0 && addr[13] == 1);
  endmethod

  method Bool isEnabledMODEM (Bit#(16) addr);
    return (addr[15] == 1);
  endmethod

endmodule
```

```
module top (Empty);

  Decoder2 d      <- mkDecoder;
  Reg#(Bit#(16)) a <- mkReg(0);

  rule isRAM if (d.isEnabledRAM(a));
    $display("RAM a=%4x",a);
  endrule

  rule isFlash  if  (d.isEnabledFLASH(a));
    $display("Flash a=%4x",a);
  endrule

  rule isModem if (d.isEnabledMODEM(a));
    $display("Modem a=%4x",a);
  endrule

  rule genaddress;
    a <= a + 8192;
    if  (a[15:14] == 1)
      $finish;
   endrule
endmodule
```

Dekodierung innerhalb eines Teilnehmers

Einfaches Beispiel 1KB Flash-ROM, organisiert als 1Kx8b

TECHNISCHE UNIVERSITÄT DARMSTADT

```
module rom1kx8 (
  input         SELECT,
  input  [9:0]  ADDR,
  output [7:0]  DATA
);

  reg  [7:0]  MEM [0:1023]

  assign DATA = (SELECT) ? MEM[ADDR] : 8'bz;

  initial  begin              // einige Beispieldaten eintragen
    MEM[0]    = 8'h42;
    MEM[1]    = 8'h23;
     ...
    MEM[1022] = 8'h20;
    MEM[1023] = 8'h07;
  end

endmodule
```

Dekodierung innerhalb eines Teilnehmers

Komplexeres Beispiel: Modem-Steuerregister

TECHNISCHE UNIVERSITÄT DARMSTADT

```
module modem (
  input         CLOCK,
  input         SELECT,
  input  [8:0]  ADDR,
  input         WRITE,
  inout  [7:0]  DATA
);

  reg [7:0] baudrate;
  reg [1:0]  parity;
  reg [7:0] inchar, outchar;

  assign DATA =  (~SELECT | WRITE) ? 8'bz :
                 ((ADDR==0) ? baudrate :
                  (ADDR==1) ? {6'b0,parity} :
                  (ADDR==2) ? inchar // <-- ADDR=2 liest Zeichen
                  : 8'h42);          // <-- Default-Wert für Debugging

  always @(posedge CLOCK) begin
    if (SELECT & WRITE)
      case (ADDR)
        0 : baudrate  <= DATA;
        1 : parity    <= DATA[1:0];
        2 : outchar   <= DATA;          // <-- ADDR=2 schreibt Zeichen
      endcase
  end

endmodule
```

Anschluss an Bus

Am Beispiel des 1KB Flash-ROMs

TECHNISCHE UNIVERSITÄT DARMSTADT

```
module mysystem;
 ...

  wire [15:0]  ADDR;
  wire  [7:0]  DATA;
  wire         SEL_RAM, SEL_FLASH, SEL_MODEM;

  // Adressdecoder
  decoder2  DECODER  (ADDR, SEL_RAM, SEL_FLASH, SEL_MODEM);

  // Flash-ROM
  rom1kx8   FLASH    (SEL_FLASH, ADDR[9:0], DATA);

 ...
endmodule
```

Verhalten des Flash-ROMs

TECHNISCHE UNIVERSITÄT DARMSTADT

Startaddresse	Endaddresse	Teilnehmer
16'b0000_0000_0000_0000	16'b0001_1111_1111_1111	RAM 8KB
16'b0010_0000_0000_0000	16'b0010_0011_1111_1111	Flash-Speicher 1KB
16'b1111_1110_0000_0000	16'b1111_1111_1111_1111	Modem 512B

Zugriff auf Busadresse	Flash Select	Zugriff auf ROM-Adresse	Ausgabe-datum	
16'h0000	0	10'h000	8'bz	Zugriff auf RAM-Bereich
16'hFE00	0	10'h200	8'bz	Zugriff auf Modem-Bereich
16'h2000	1	10'h000	8'h42	Zugriffe auf Flash-Bereich
16'h2001	1	10'h001	8'h23	
16'h23FE	1	10'h3FE	8'h20	
16'h23FF	1	10'h3FF	8'h07	
16'h2400	1	10'h000	8'h42	Hinter Flash-Ende!
16'h2401	1	10'h001	8'h23	
16'h27FE	1	10'h3FE	8'h20	
16'h27FF	1	10'h3FF	8'h07	
16'h2800	1	10'h000	8'h42	
16'h2801	1	10'h001	8'h23	

...

➥Der Flash-Bereich wiederholt sich!

Memory Aliasing

TECHNISCHE UNIVERSITÄT DARMSTADT

- Speicherbereich wiederholt sich
- Es sind aber immer die gleichen Daten
- Sichtbarkeit der gleichen *lokalen* Adressen an unterschiedlichen *Busadressen*: Aliasing
- Schadet in vielen Fällen nicht
  - Manchmal schon: Hack der Microsoft XBOX
  - Schreibzugriffe auf 80008008 werden abgefangen
  - Aber Fehler im Adressdecoder der Southbridge
  - Adressen sind auch aliased als 80008x08
  - Zugriffe erlaubt für $x \neq 0$
  - So Zugriff auf “geheimen” Boot-Code möglich
  - $\rightarrow$ “MIST Premature Unmap Attack ”
- Erlaubt aber sehr einfache und schnelle Adressdekodierung

Zusammenfassung Adressdekodierung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Zeige Busteilnehmern durch Select-Signal an, wenn sie angesprochen werden
- Select-Signale werden aus Busadressen erzeugt
- Adressbereiche müssen überlappungsfrei sein
- Gleichzeitig darf maximal ein Select-Signal aktiv sein
- Erstelle Adressdekodierlogik durch Aufbau eines Entscheidungsbaumes
  - Von höherwertigen zu niederwertigen Adressbits
  - Erfülle Anforderungen dabei durch Auswertung von möglichst wenigen Adressbits
- Dabei darf derselbe Adressbereich i.d.R. mehrfach auftauchen (aliasing)
- Er muß aber mindestens an den spezifizierten Adressen erreichbar sein