Digitaltechnik
Wintersemester 2017/2018
10. Vorlesung

TECHNISCHE UNIVERSITÄT DARMSTADT

Inhalt

TECHNISCHE UNIVERSITÄT DARMSTADT

Einleitung

TECHNISCHE UNIVERSITÄT DARMSTADT

Organisatorisches

TECHNISCHE UNIVERSITÄT DARMSTADT

- Zusammenlegung von Übungsgruppen ab KW 51
  - G01 $\rightarrow$ G07
    - Mo 08:00-09:40 S202/C205
    - Thomas Kampa
    - Roland Schurig
  - G22 $\rightarrow$ G02
    - Mo 15:20-17:00 S311/006
    - Timo Henz
    - Moritz Nottebaum

Organisatorisches

TECHNISCHE UNIVERSITÄT DARMSTADT

- Zusammenlegung von Übungsgruppen ab KW 51
  - G01 $\rightarrow$ G07
    - Mo 08:00-09:40 S202/C205
    - Thomas Kampa
    - Roland Schurig
  - G22 $\rightarrow$ G02
    - Mo 15:20-17:00 S311/006
    - Timo Henz
    - Moritz Nottebaum
- Hausaufgaben fürs neue Jahr: SystemVerilog Tools ausprobieren
  - Ende KW 51 im Moodle

Rückblick auf die letzten Vorlesungen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Sequentielle Logik
  - Sequentielle Schaltungen
  - Speicherelemente
  - Synchrone sequentielle Logik
- Endliche Zustandsautomaten
  - Konzept, Notationen und Anwendungsbeispiele
  - Moore vs. Mealy
  - Zerlegen von Zustandsautomaten
- Zeitverhalten synchroner sequentieller Schaltungen
- Parallelität

Harris 2013
Kap. 3.1 - 3.6

Wiederholung: Parallelität
Nochmal Plätzchen backen

TECHNISCHE UNIVERSITÄT DARMSTADT

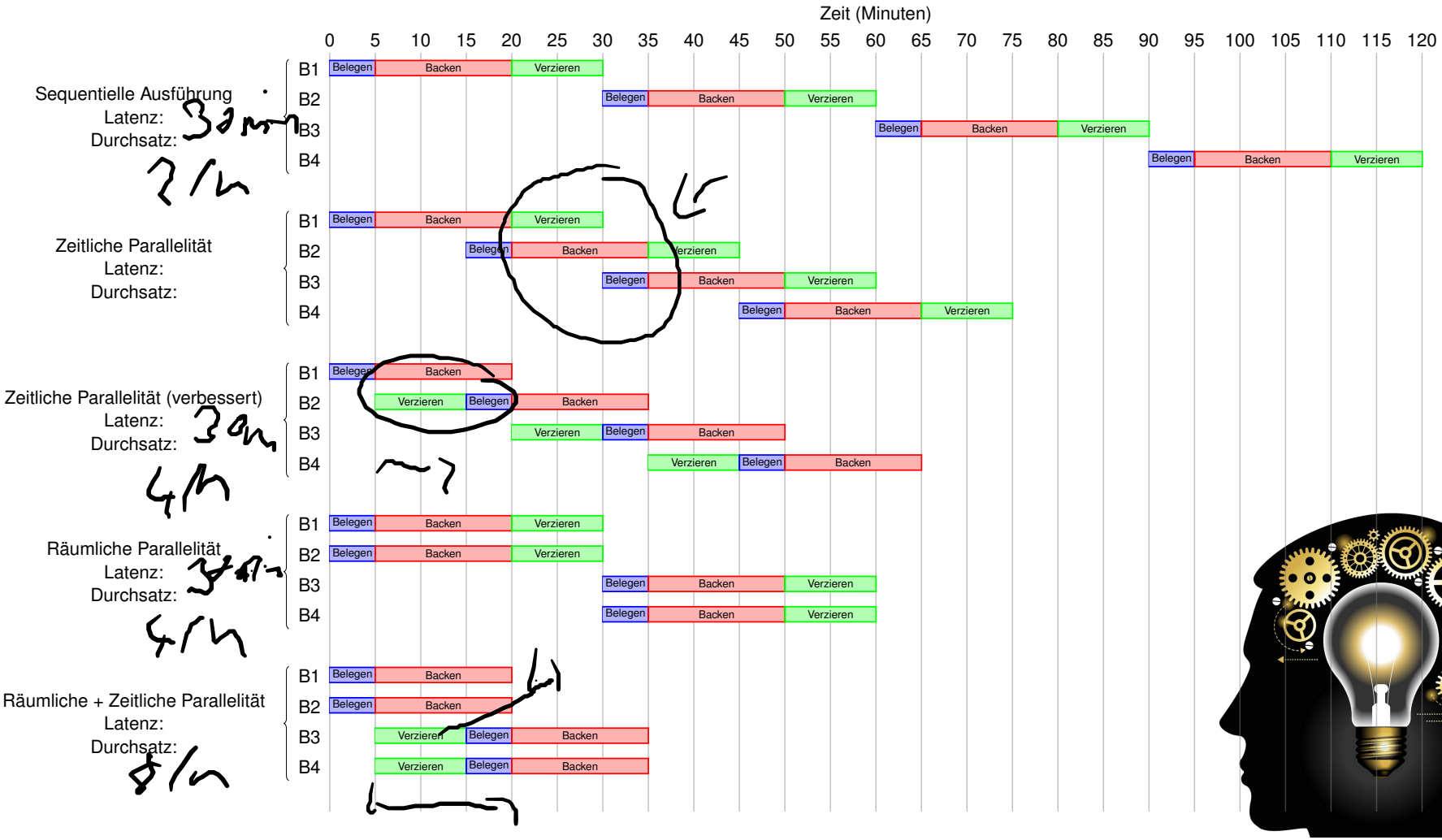
- Annahmen:
  - genug Teig ist fertig
  - 5 Minuten zum Belegen eines Bleches
  - 15 Minuten Backzeit
  - 10 Minuten verzieren

$\Rightarrow$ Durchsatz steigern mit

- zeitlicher Parallelität
- räumlicher Parallelität

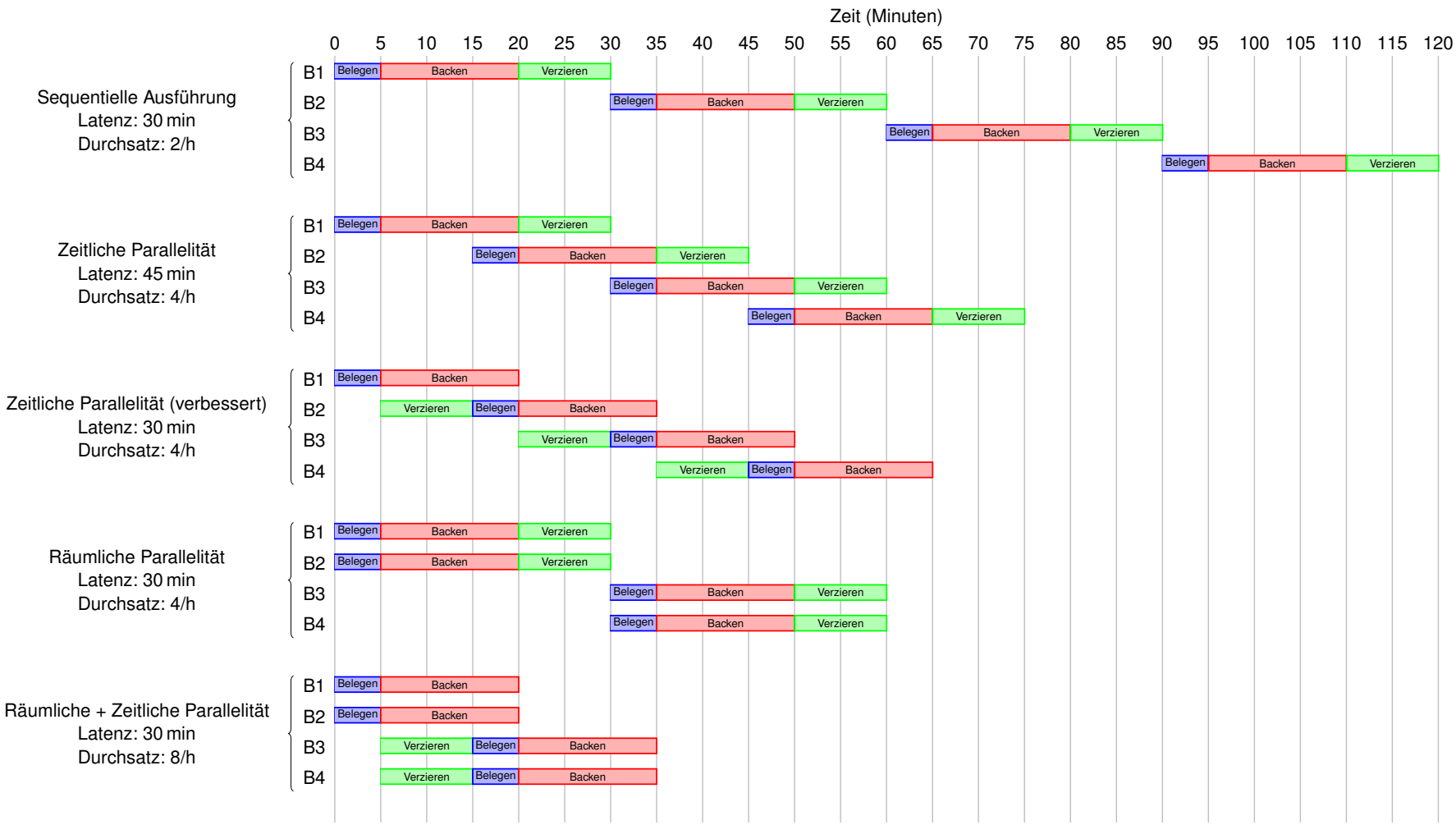
Wiederholung: Parallelität
Nochmal Plätzchen backen

TECHNISCHE UNIVERSITÄT DARMSTADT



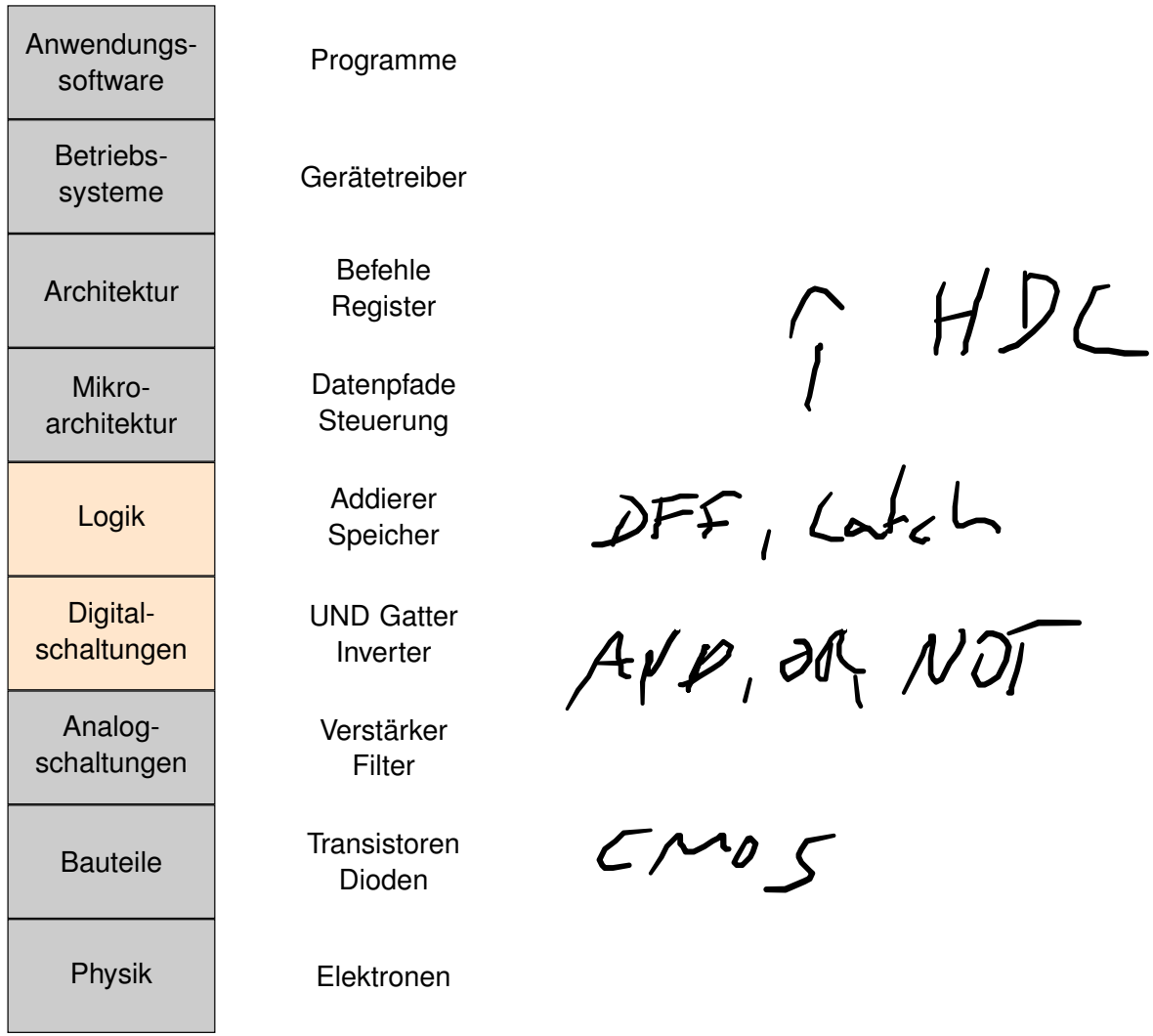
Wiederholung: Parallelität
Nochmal Plätzchen backen

TECHNISCHE UNIVERSITÄT DARMSTADT



Wiederholung: Schichtenmodell

TECHNISCHE UNIVERSITÄT DARMSTADT



Überblick der heutigen Vorlesung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Historie von Hardwarebeschreibungssprachen
- SystemVerilog für kombinatorische Logik
- SystemVerilog Modulhierarchie

Harris 2013
Kap. 4.1-3.3
Seite 167 - 190

Historie von Hardwarebeschreibungssprachen

TECHNISCHE UNIVERSITÄT DARMSTADT

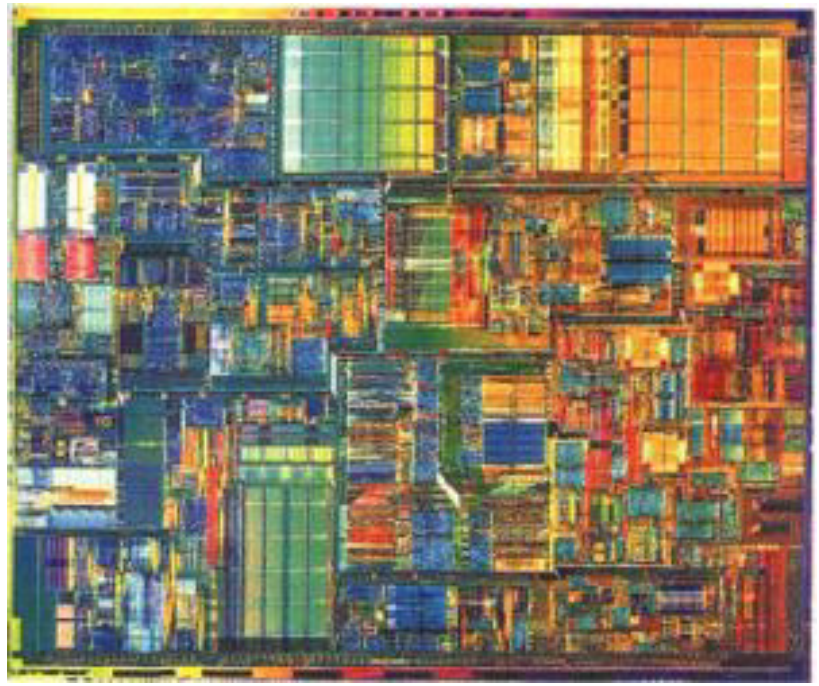
Notwendigkeit von HDLs
Hardware Description Language

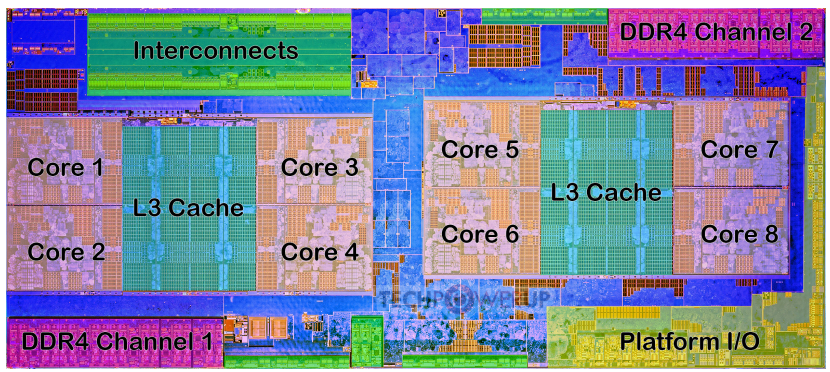
TECHNISCHE UNIVERSITÄT DARMSTADT

- Komplexität technischer Systeme steigt ständig (vgl. Moores Gesetz)
  - 2000: Intel Pentium 4:
    $42 \cdot 10^6$ Transistoren auf 217 mm^2
  - 2017: AMD Ryzen:
    $4{,}8 \cdot 10^9$ Transistoren auf 192 mm^2

$\Rightarrow$ ohne rechnergestützte Hilfsmittel nicht zu beherrschen

$\Rightarrow$ Hardware-Beschreibungssprachen zum Beherschen von Komplexität





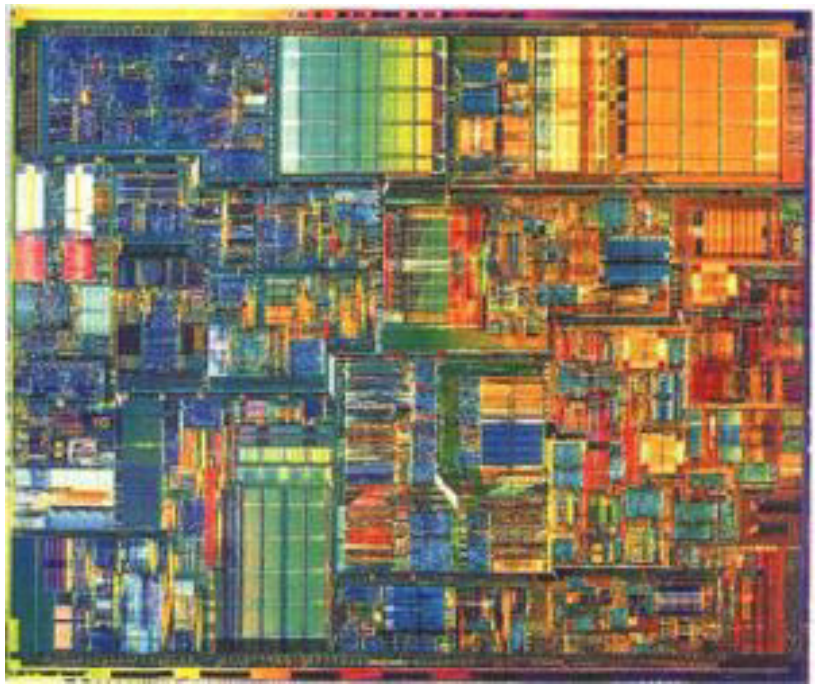
Notwendigkeit von HDLs
Hardware Description Language

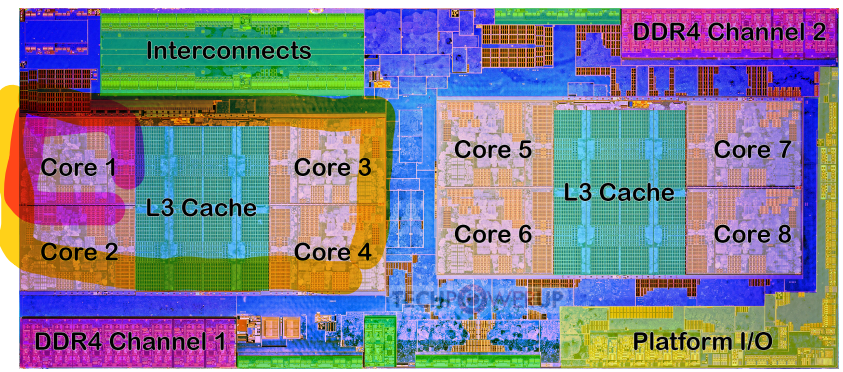
TECHNISCHE UNIVERSITÄT DARMSTADT

- Komplexität technischer Systeme steigt ständig (vgl. Moores Gesetz)
  - 2000: Intel Pentium 4:
    $42 \cdot 10^6$ Transistoren auf 217 mm^2
  - 2017: AMD Ryzen:
    $4{,}8 \cdot 10^9$ Transistoren auf 192 mm^2

$\Rightarrow$ ohne rechnergestützte Hilfsmittel nicht zu beherrschen

$\Rightarrow$ Hardware-Beschreibungssprachen zum Beherschen von Komplexität
  - Hierarchie
  - Modularität
  - Regularität





Anwendung und Entwicklung der HDLs

TECHNISCHE UNIVERSITÄT DARMSTADT

- seit Beginn der Rechnerentwicklung:
  - Suche nach verständlichen und einheitlichen Beschreibungssprachen für
    - Designspezifikation
    - Simulation
    - Verifikation
    - Dokumentation
  - $\Rightarrow$ nutzt auch der Kommunikation zwischen Entwicklern
- zunächst Hochsprachen (bspw. Pascal, LISP, Petri-Netze) zur Hardware-Beschreibung eingesetzt
- 1960/70: Register-Transfersprachen
  - Daten*transfer* zwischen *Register*n durch kombinatorische Operatoren
  - $\Rightarrow$ synchrone sequentielle Schaltungen als Abstraktionslevel

Robert Piloty, 1924 - 2013

TECHNISCHE UNIVERSITÄT DARMSTADT

- Maßgeblich an Einführung/Entwicklung des Informatikstudiums beteiligt
- Forschung an
  - programmgesteuerten Rechenanlagen (PERM)
  - rechnergestützer Schaltungsentwurf

Robert Piloty, 1924 - 2013

TECHNISCHE UNIVERSITÄT DARMSTADT

- Maßgeblich an Einführung/Entwicklung des Informatikstudiums beteiligt
- Forschung an
  - programmgesteuerten Rechenanlagen (PERM)
  - rechnergestützer Schaltungsentwurf
- RTS 1a (Register Transfer System Language)
  - an TH Darmstadt entwickelt
  - entstand aus praktischer Erfahrung
  - sollte Fehler früherer Ansätze vermeiden (bspw. zu hohes Abstraktionsniveau)
  - sollte leicht zu lernen und zu lehren sein
  - sollte verschiedene Entwurfsmethoden und Entwurfsebenen abdecken
  - einfache syntaktische und semantische Regeln
  - $\Rightarrow$ jede gültige Hardwarebeschreibung in RTS 1a soll auch realisierbar sein

Beispiel für RTS 1a Beschreibung

Quelle: Computer Aids for VLSI Circuits, 1981

TECHNISCHE UNIVERSITÄT DARMSTADT

example.rts

```
1  -INPUTTERMINAL- START, MRE[1:4], MDE[1:4];
2  -REGISTER- AK[1:4], MR[1:4], MD[1:4], CC[1:4], ST[1:2];
3  -CASE- ST
4  :0: -IF- START -THEN-
5        AK <= #0B4, MR <= MRE, MD <= MDE, CC <= #12D4, ST <= #1B2 -FI-
6  :1: -IF- MR[4] -THEN-
7        AK <= ADD(AK, MD), ST <= #2D2
8      -ELSE-
9        (AK,MR) <= RSH(#0, (AK,MR)), CC <= INC(CC),
10       -IF- EQ(CC, #15D4) -THEN- ST <= #0B2 -ELSE- ST <= #1B2 -FI-
11     -FI-
12 :2: (AK,MR) <= RSH(#0, (AK,MR)), CC <= INC(CC),
13     -IF- EQ(CC, #15D4) -THEN- ST <= #0B2 -ELSE- ST <= #1B2 -FI-
14 -ESAC-
15 -FINIS
```

1983 - Geburtsstunde wichtiger HDL Standards

TECHNISCHE UNIVERSITÄT DARMSTADT

- Consensus Language (CONLAN)
  - allgemeine, erweiterbare Sprache
  - sollte den akademischen „Wildwuchs“ in geordnete Bahnen lenken

  $\Rightarrow$ Akzeptanz von HDLs in Industrie fördern

1983 - Geburtsstunde wichtiger HDL Standards

TECHNISCHE UNIVERSITÄT DARMSTADT

- Consensus Language (CONLAN)
  - allgemeine, erweiterbare Sprache
  - sollte den akademischen „Wildwuchs“ in geordnete Bahnen lenken

  $\Rightarrow$ Akzeptanz von HDLs in Industrie fördern

- Very High-Speed Integrated Circuits Hardware Description Language (VHDL)
  - vom US Department of Defense maßgeblich gefördert
  - IEEE Standard 1076 (1987, 1993, 2002, 2008)
  - Erweiterung:
    - 1998: VHDL-AMS (Analog and Mixed-Signal)

1983 - Geburtsstunde wichtiger HDL Standards

TECHNISCHE UNIVERSITÄT DARMSTADT

- Consensus Language (CONLAN)
  - allgemeine, erweiterbare Sprache
  - sollte den akademischen „Wildwuchs“ in geordnete Bahnen lenken

  $\Rightarrow$ Akzeptanz von HDLs in Industrie fördern

- Very High-Speed Integrated Circuits Hardware Description Language (VHDL)
  - vom US Department of Defense maßgeblich gefördert
  - IEEE Standard 1076 (1987, 1993, 2002, 2008)
  - Erweiterung:
    - 1998: VHDL-AMS (Analog and Mixed-Signal)

- Verilog HDL
  - von Gateway Design Automation (Cadence) zur Simulation entwickelt
  - IEEE Standard 1364 (1995, 2001)
  - Erweiterung:
    - 1998: Verilog-AMS (Analog and Mixed-Signal)
    - 2002: SystemVerilog (Verifikation)

Aktueller Tendenz: Anstieg des Abstraktionslevels

TECHNISCHE UNIVERSITÄT DARMSTADT

- SystemC
  - C++ Klassenbibliothek
  - erlaubt besonders schnelle Simulation

Aktueller Tendenz: Anstieg des Abstraktionslevels

TECHNISCHE UNIVERSITÄT DARMSTADT

- SystemC
  - C++ Klassenbibliothek
  - erlaubt besonders schnelle Simulation
- Constructing Hardware in a Scala Embedded Language (Chisel)
  - von UC Berkeley
  - durch Einbettung in Skala (funktionales Java) sehr flexibel

Aktueller Tendenz: Anstieg des Abstraktionslevels

TECHNISCHE UNIVERSITÄT DARMSTADT

- SystemC
  - C++ Klassenbibliothek
  - erlaubt besonders schnelle Simulation
- Constructing Hardware in a Scala Embedded Language (Chisel)
  - von UC Berkeley
  - durch Einbettung in Skala (funktionales Java) sehr flexibel
- BlueSpec-Verilog (BSV)
  - vom MIT, aber inzwischen kommerzialisiert
  - erbt Abstraktionsniveau von funktionalem Haskell

Aktueller Tendenz: Anstieg des Abstraktionslevels

TECHNISCHE UNIVERSITÄT DARMSTADT

- SystemC
  - C++ Klassenbibliothek
  - erlaubt besonders schnelle Simulation
- Constructing Hardware in a Scala Embedded Language (Chisel)
  - von UC Berkeley
  - durch Einbettung in Skala (funktionales Java) sehr flexibel
- BlueSpec-Verilog (BSV)
  - vom MIT, aber inzwischen kommerzialisiert
  - erbt Abstraktionsniveau von funktionalem Haskell
- High-Level-Synthese: low-level Verilog/VHDL aus abstrakten Anwendungsbeschreibungen (bspw. in C, Java, Matlab) erzeugen

Aktueller Tendenz: Anstieg des Abstraktionslevels

TECHNISCHE UNIVERSITÄT DARMSTADT

- SystemC
  - C++ Klassenbibliothek
  - erlaubt besonders schnelle Simulation
- Constructing Hardware in a Scala Embedded Language (Chisel)
  - von UC Berkeley
  - durch Einbettung in Skala (funktionales Java) sehr flexibel
- BlueSpec-Verilog (BSV)
  - vom MIT, aber inzwischen kommerzialisiert
  - erbt Abstraktionsniveau von funktionalem Haskell
- High-Level-Synthese: low-level Verilog/VHDL aus abstrakten Anwendungsbeschreibungen (bspw. in C, Java, Matlab) erzeugen

$\Rightarrow$ Schritt von Beschreibung zur Ausführung (Semantic Gap) wird immer größer

Von HDL zu Logikgattern

TECHNISCHE UNIVERSITÄT DARMSTADT

- Simulation des funktionalen/zeitlichen Verhaltens der beschriebenen Schaltung
  - berechnete Ausgaben zu vorgegebenen Eingaben werden auf Korrektheit geprüft

  $\Rightarrow$ Fehlersuche einfacher (billiger) als in realer Hardware

Von HDL zu Logikgattern

TECHNISCHE UNIVERSITÄT DARMSTADT

- Simulation des funktionalen/zeitlichen Verhaltens der beschriebenen Schaltung
  - berechnete Ausgaben zu vorgegebenen Eingaben werden auf Korrektheit geprüft

  $\Rightarrow$ Fehlersuche einfacher (billiger) als in realer Hardware

- Synthese übersetzt Hardware-Beschreibungen in Netzlisten
  - Schaltungselemente (Logikgatter) + Verbindungsknoten
  - entspricht Registertransferebene
  - kann auf Gatter-Bibliothek einer konkreten Zielarchitektur abgebildet werden (Technology-Mapping)
    - wenige CMOS-Basisgatter für Application-Specific Integrated Circuits (ASICs)
    - kleine Lookup-Tabellen für Field-Programmable Gate Arrays (FPGAs)

Von HDL zu Logikgattern

TECHNISCHE UNIVERSITÄT DARMSTADT

- Simulation des funktionalen/zeitlichen Verhaltens der beschriebenen Schaltung
  - berechnete Ausgaben zu vorgegebenen Eingaben werden auf Korrektheit geprüft

  $\Rightarrow$ Fehlersuche einfacher (billiger) als in realer Hardware

- Synthese übersetzt Hardware-Beschreibungen in Netzlisten
  - Schaltungselemente (Logikgatter) + Verbindungsknoten
  - entspricht Registertransferebene
  - kann auf Gatter-Bibliothek einer konkreten Zielarchitektur abgebildet werden (Technology-Mapping)
    - wenige CMOS-Basisgatter für Application-Specific Integrated Circuits (ASICs)
    - kleine Lookup-Tabellen für Field-Programmable Gate Arrays (FPGAs)

- WICHTIG: für effiziente Hardware-Beschreibung muss HDL-Programmierer *immer* die Zielarchitektur im Auge behalten

SystemVerilog für kombinatorische Logik

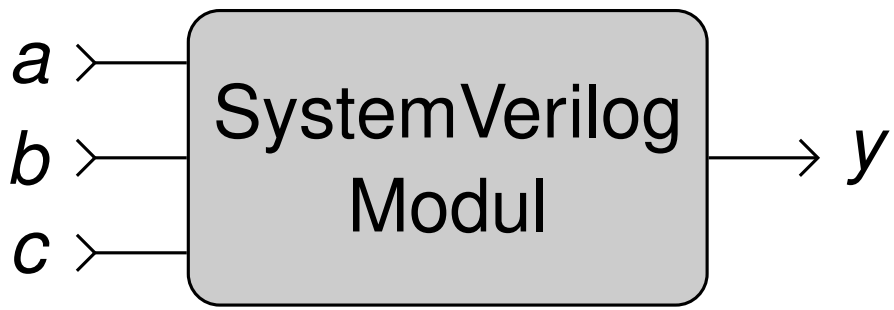
TECHNISCHE UNIVERSITÄT DARMSTADT

SystemVerilog Module

TECHNISCHE UNIVERSITÄT DARMSTADT

- Schnittstellenbeschreibung:
  - Eingänge
  - Ausgänge
  - (Parameter)
- zwei Arten von Modul-Beschreibungen:
  - Struktur: Wie ist die Schaltung aus (Unter-)Modulen aufgebaut?
  - Verhalten: Was tut die Schaltung?

$\Rightarrow$ strukturelle Modul-Hierarchie mit Verhaltensbeschreibung auf unterster Ebene



Beispiel für Verhaltensbeschreibung

TECHNISCHE UNIVERSITÄT DARMSTADT

example.sv

```
1  module example(input  logic a, b, c,
2                 output logic y);
3
4  assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b & c;
5
6  endmodule
```

- **module** Beginn Schnittstellenbeschreibung
- example Modulname
- **input**, **output** Port-Richtung
- **logic** Port-Datentyp
- a,b,c,y Port-Namen
- **assign** (kombinatorische) Signalzuweisung
- ~& | (kombinatorische) Operatoren (NOT, AND, OR)
- **endmodule** Ende Schnittstellenbeschreibung

SystemVerilog Syntax

TECHNISCHE UNIVERSITÄT DARMSTADT

- Unterscheidet Groß- und Kleinschreibung
  - bspw. reset ≠ Reset
- Bezeichner für Module/Signale Namen dürfen nicht mit Ziffern anfangen
  - bspw. 2mux ungültig
- Anzahl von Leerzeichen, Leerzeilen und Tabulatoren irrelevant
- Kommentare:
  - // *bis zum Ende der Zeile*
  - /* *über mehrere Zeilen* */

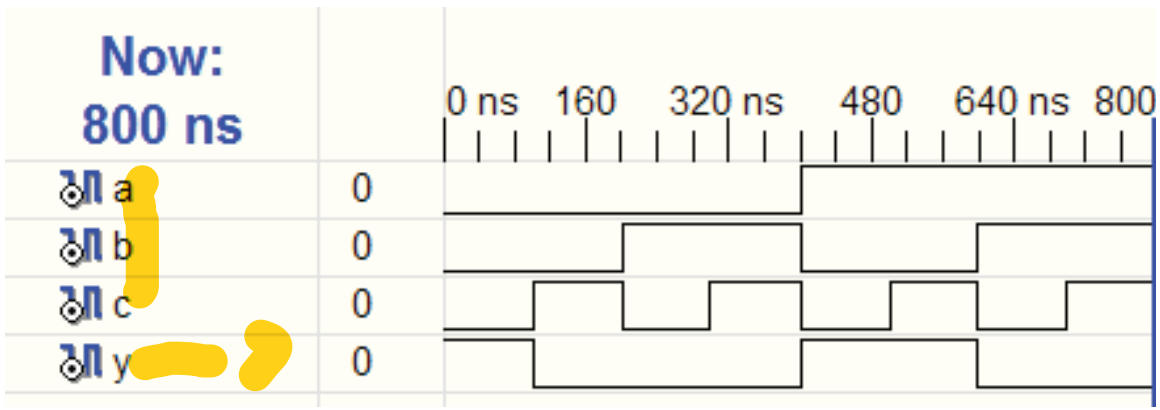
Simulation von Verhaltensbeschreibungen

TECHNISCHE UNIVERSITÄT DARMSTADT

example.sv

```
1  module example(input  logic a, b, c,
2                 output logic y);
3
4  assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b & c;
5
6  endmodule
```

Signalverlaufsdiagramm (*waves*):

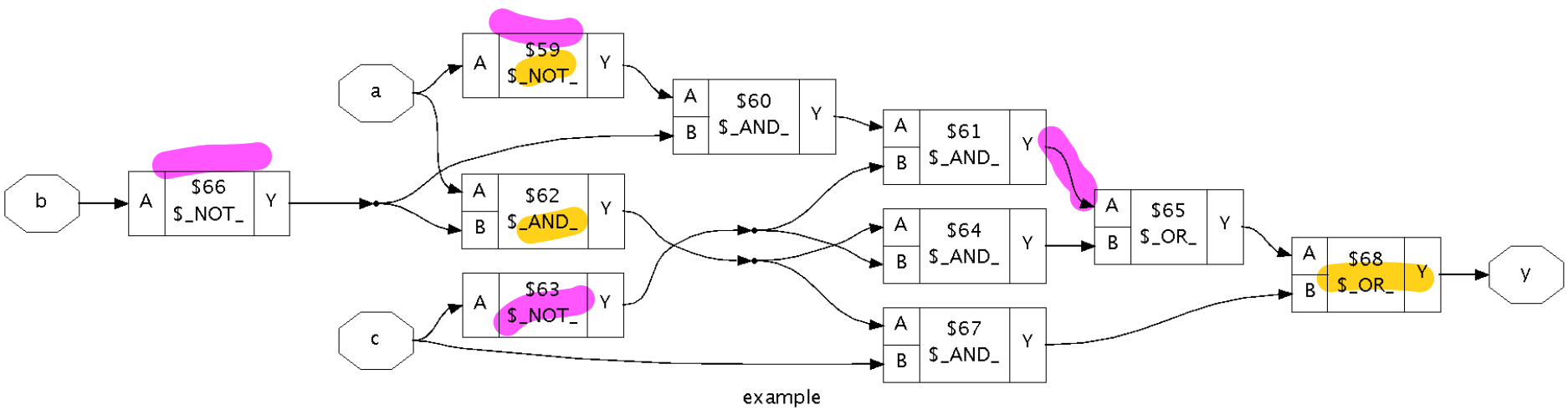


Synthese von Verhaltensbeschreibungen

TECHNISCHE UNIVERSITÄT DARMSTADT

example.sv

```
1  module example(input  logic a, b, c,
2                 output logic y);
3
4  assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b & c;
5
6  endmodule
```



example

SystemVerilog Modulhierarchie

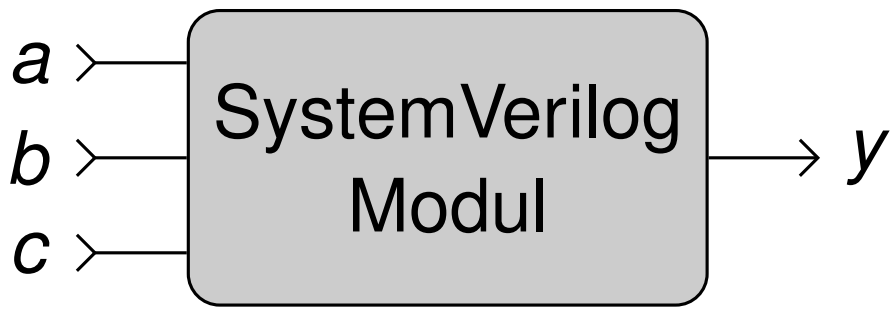
TECHNISCHE UNIVERSITÄT DARMSTADT

SystemVerilog Module

TECHNISCHE UNIVERSITÄT DARMSTADT

- Schnittstellenbeschreibung:
  - Eingänge
  - Ausgänge
  - (Parameter)
- zwei Arten von Modul-Beschreibungen:
  - Struktur: Wie ist die Schaltung aus (Unter-)Modulen aufgebaut?
  - Verhalten: Was tut die Schaltung?

$\Rightarrow$ strukturelle Modul-Hierarchie mit Verhaltensbeschreibung auf unterster Ebene



Strukturelle Beschreibung: Modulinstanziierung

TECHNISCHE UNIVERSITÄT DARMSTADT

and3.sv

```
1 module and3(input logic a, b, c, output logic y);
2   assign y = a & b & c;
3 endmodule
```

inv.sv

```
1 module inv(input logic a, output logic y);
2   assign y = ~a;
3 endmodule
```

nand3.sv

```
1 module nand3 (input logic d, e, f, output logic w);
2   logic s;                    //internes Signal für Modulverbindung
3   and3 andgate(d, e, f, s);   //Instanz von and3 namens andgate
4   inv  inverter(s, w);        //Instanz von inv namens inverter
5 endmodule
```

Strukturelle Beschreibung: Portzuweisung nach Position oder Namen

TECHNISCHE UNIVERSITÄT DARMSTADT

nand3.sv

```
1 module nand3 (input logic d, e, f, output logic w);
2   logic s;                    //internes Signal für Modulverbindung
3   and3 andgate(d, e, f, s);//Instanz von and3 namens andgate
4   inv  inverter(s, w);        //Instanz von inv namens inverter
5 endmodule
```

nand3.named.sv

```
1 module nand3 (input logic d, e, f, output logic w);
2   logic s;
3   and3 andgate(.a(d), .b(e), .c(f), .y(s));
4   inv  inverter(.a(s), .y(w));
5 endmodule
```

- 10 bis 100 ports pro Modul nicht unüblich

$\Rightarrow$ absolute Portzuweisung per Namen übersichtlicher (selbstdokumentiert)

Bitweise Verknüpfungsoperatoren

TECHNISCHE UNIVERSITÄT DARMSTADT

gates.sv

```
1  module gates (input  logic [3:0] a, b,
2                output logic [3:0] y1,y2,y3,y4,y5);
3     /* Fünf unterschiedliche Logikgatter
4        mit zwei Eingaengen, jeweils 4b Busse */
5     assign y1 =    a & b;  // AND
6     assign y2 =    a | b;  // OR
7     assign y3 =    a ^ b;  // XOR
8     assign y4 = ~(a & b); // NAND
9     assign y5 = ~(a | b); // NOR
10 endmodule
```

Reduktionsoperatoren (unär)

TECHNISCHE UNIVERSITÄT DARMSTADT

and8.sv

```
1 module and8 (input logic [7:0] a, output logic y);
2     assign y = &a;
3 // Abkürzung für
4 // assign y = a[7] & a[6] & a[5] & a[4] &
5 //              a[3] & a[2] & a[1] & a[0];
6 endmodule
```

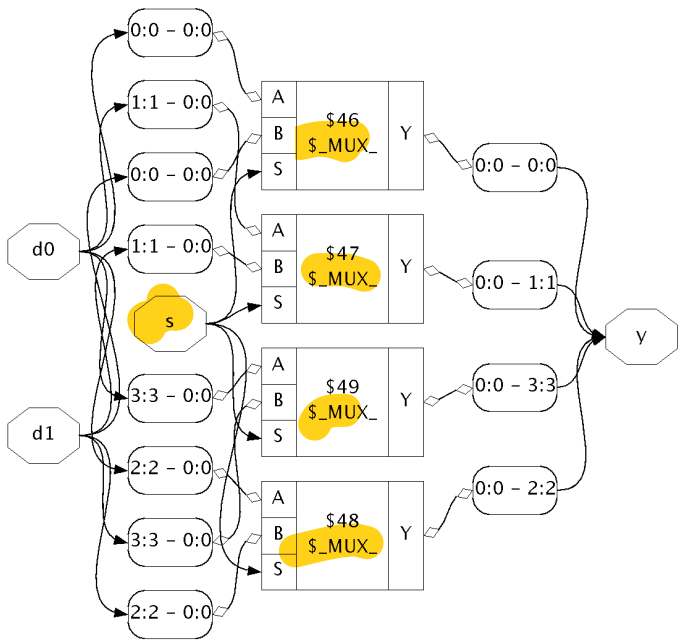
- analog:
  - | OR
  - ^ XOR
  - ~| NOR
  - ~& NAND
  - ~^ XNOR

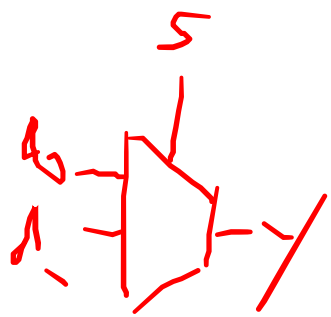
Bedingte Zuweisung (ternär) und deren Syntheseergebnis

TECHNISCHE UNIVERSITÄT DARMSTADT

mux2.sv

```
1  module mux2(input  logic [3:0] d0, d1,
2              input  logic       s,
3              output logic [3:0] y);
4    assign y = s ? d1 : d0;
5  endmodule
```



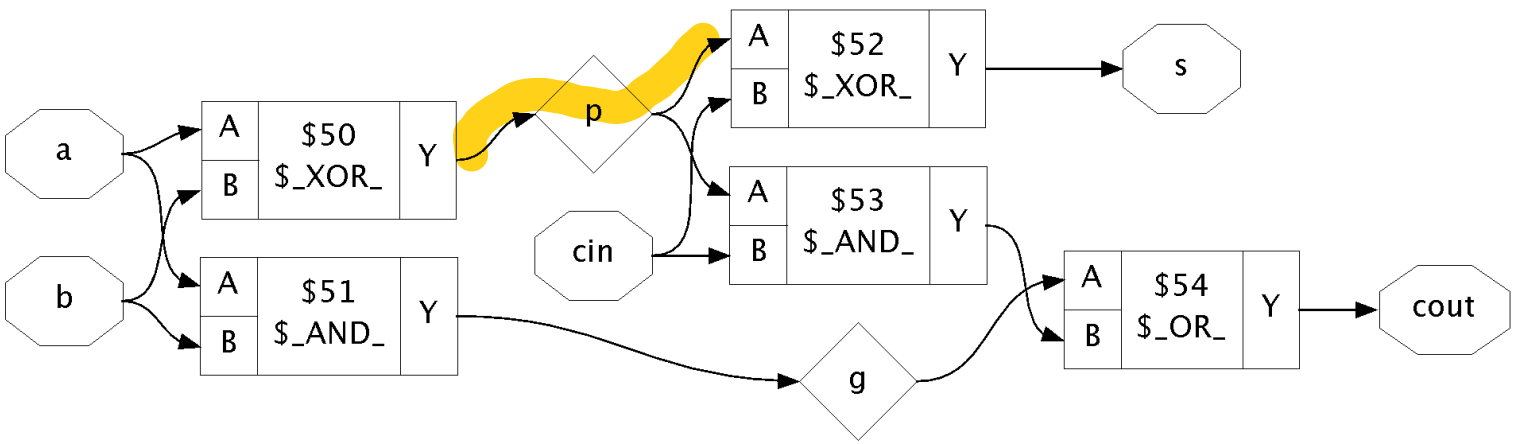


Interne Verbindungsknoten (Signale)

TECHNISCHE UNIVERSITÄT DARMSTADT

fulladder.sv

```
1  module fulladder(input  logic a, b, cin,
2                   output logic s, cout);
3    logic  p, g;    // interne Verbindungsknoten
4    assign p = a ^ b;
5    assign g = a & b;
6    assign s = p ^ cin;
7    assign cout = g | (p & cin);
8  endmodule
```



Bindung von Operatoren (Präzedenz)

TECHNISCHE UNIVERSITÄT DARMSTADT

∼	NOT (höchste Präzedenz)
*, /, %	Multiplikation, Division, Modulo
+, −	Addition, Subtraktion
<<, >>	logischer Shift
<<<, >>>	arithmetischer Shift
<, <=, >, >=	Vergleich
==, !=	gleich, ungleich
&, ∼&	AND, NAND
^, ∼^	XOR, XNOR
|, ∼|	OR, NOR
?:	ternärer Operator (niedrigste Präzedenz)

Syntax für numerische Literale

TECHNISCHE UNIVERSITÄT DARMSTADT

- Syntax: <N>'<B><wert>
  - <N> = Bitbreite
  - <B> = Basis (d,b,o,h)
  - beide Angaben optional (default: 32'd)
  - Unterstriche als optische Trenner möglich (werden ignoriert)

Literal	Bitbreite	Basis	Dezimal	Binär
3'b101	3	binär	5	101
'b11	32	binär	3	0000...0000011
8'b11	8	binär	3	00000011
8'b1010_1011	8	binär	171	10101011
3'd6	3	dezimal	6	110
6'o42	6	oktal	34	100010
8'hAB	8	hexadezimal	171	10101011
42	32	dezimal	42	0000...0101010

Konkatenation

TECHNISCHE UNIVERSITÄT DARMSTADT

concat.sv

```
1 module concat(input logic    [2:0] a, b,
2               output logic [11:0] y);
3   assign y = {a[2:1], {3{b[0]}}, a[0], 6'b100010};
4 // entspricht
5 // y = a[2] a[1] b[0] b[0] b[0] a[0] 1 0 0 0 1 0
6 endmodule
```

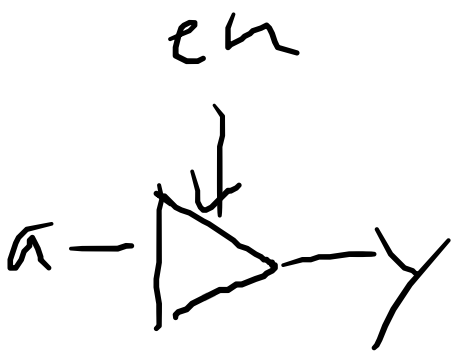
Hochohmiger Ausgang (Z) und dessen *falsche* Synthese

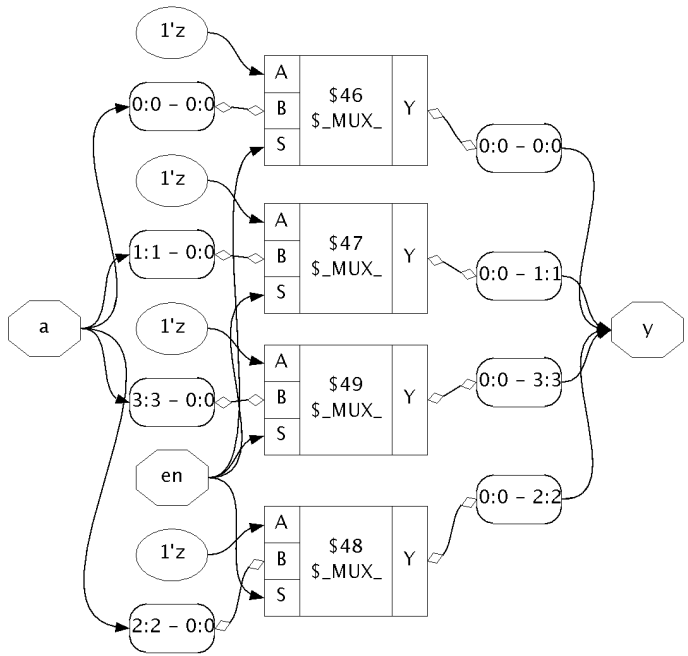
TECHNISCHE UNIVERSITÄT DARMSTADT

tristate.sv

```
1 module tristate(input  logic [3:0] a,
2                 input  logic       en,
3                 output logic [3:0] y);
4   assign y = en ? a : 4'bz;
5 endmodule
```

- Z darf *nur an Ausgängen* verwendet werden
- für interne aber trotzdem Signale simulierbar



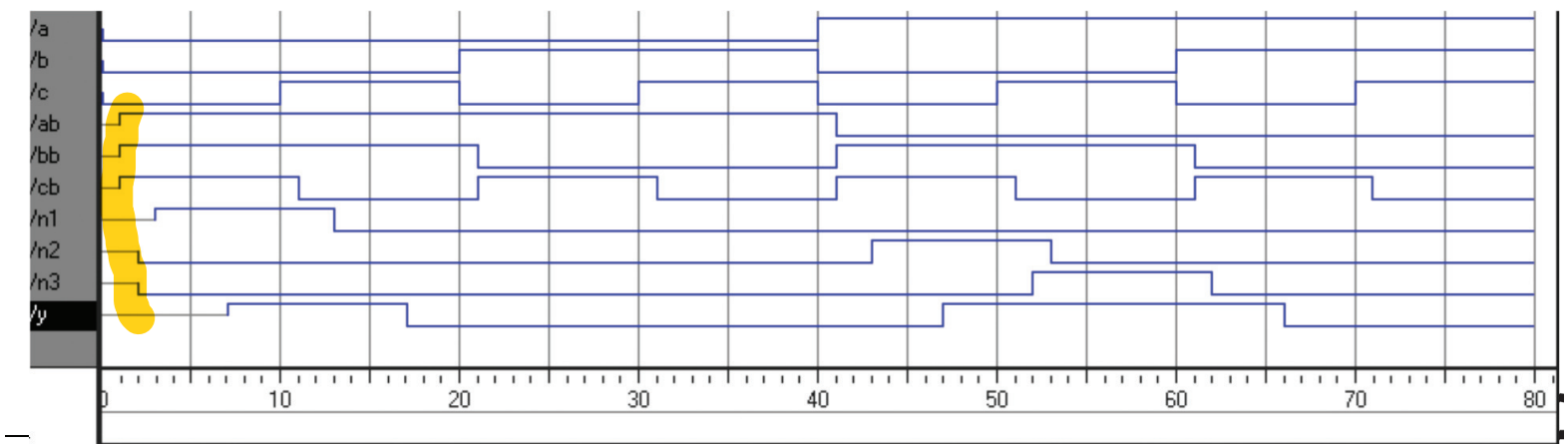


Verzögerungen: # Zeiteinheiten

TECHNISCHE UNIVERSITÄT DARMSTADT

example.delay.sv

```
1  `timescale 1ns / 10ps
2  module example(input  logic a, b, c,
3                 output logic y);
4    logic ab, bb, cb, n1, n2, n3;
5    assign #1 {ab, bb, cb} = ~{a, b, c};
6    assign #2 n1 = ab & bb & cb;
7    assign #2 n2 = a & bb & cb;
8    assign #2 n3 = a & bb & c;
9    assign #4 y = n1 | n2 | n3;
10 endmodule
```



Zusammenfassung

TECHNISCHE UNIVERSITÄT DARMSTADT

Zusammenfassung und Ausblick

TECHNISCHE UNIVERSITÄT DARMSTADT

- Historie von Hardwarebeschreibungssprachen
- SystemVerilog für kombinatorische Logik
- SystemVerilog Modulhierarchie

- Nächste Vorlesung behandelt
  - SystemVerilog für sequentielle Logik