

# Digitaltechnik

## Wintersemester 2017/2018

### 10. Vorlesung



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT





1. Einleitung
2. Historie von Hardwarebeschreibungssprachen
3. SystemVerilog für kombinatorische Logik
4. SystemVerilog Modulhierarchie
5. Zusammenfassung

# Einleitung



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

0111100111100011010001011001011011111111100  
01111101010101010000101010000101010101010  
0110110001101010111011100000010001100101  
1010100110101011110111000001010100110111  
1101011101010010001010111011111000010001  
011110100101111100100101000000101111110101  
0010110101110101011110000010010100011000  
01111100110100100010001000101110101110111  
1110001001000000111010111010100110100011  
1000000100110111101100011011101010000101  
0011100111000100001101110110100001100000  
1111001110010110011101110010111000011010  
1101110111100110001000011110111100000111  
0010011110011010010111000110111010101010  
1110110001001111010010110100011101100100  
1001100110111011100100000110001011111011



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



- ▶ Zusammenlegung von Übungsgruppen ab KW 51
  - ▶ G01 → G07
    - ▶ Mo 08:00-09:40 S202/C205
    - ▶ Thomas Kampa
    - ▶ Roland Schurig
  - ▶ G22 → 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
- ▶ Zeitverhalte 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

### ⇒ Durchsatz steigern mit

- zeitlicher Parallelität
- räumlicher Parallelität

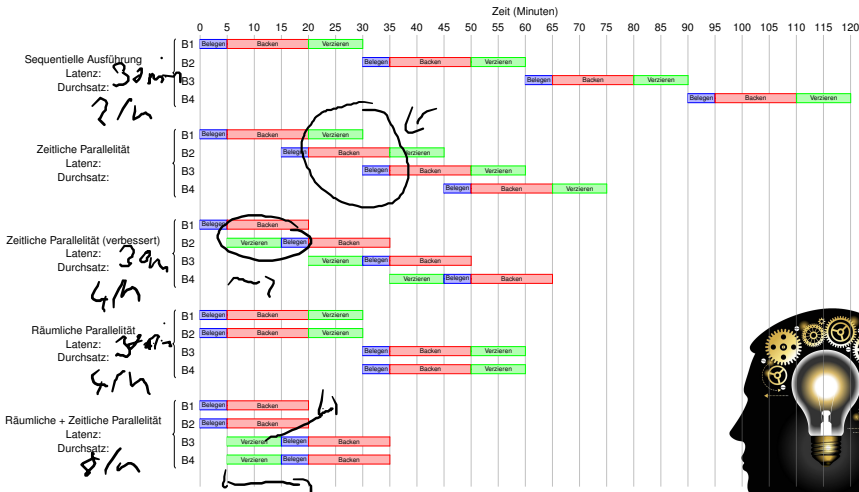


# Wiederholung: Parallelität

## Nochmal Plätzchen backen



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



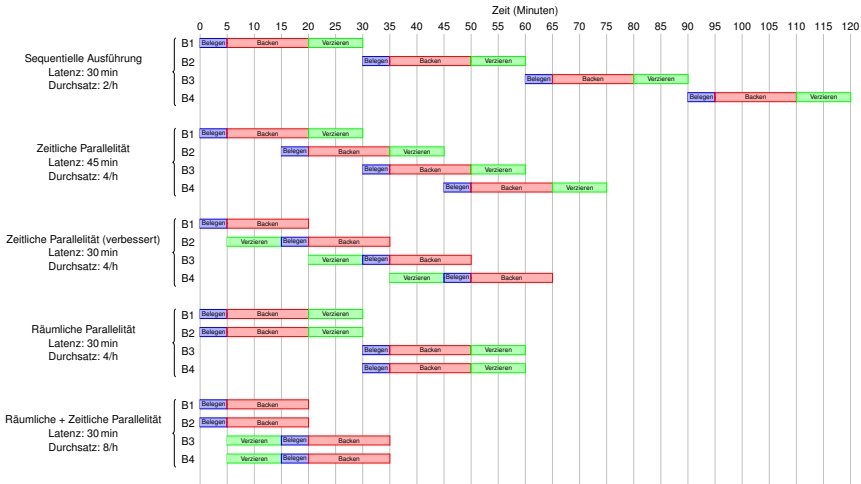


# Wiederholung: Parallelität

## Nochmal Plätzchen backen



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT



# Wiederholung: Schichtenmodell



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Anwendungs- software	Programme
Betriebs- systeme	Gerätetreiber
Architektur	Befehle Register
Mikro- architektur	Datenpfade Steuerung
Logik	Addierer Speicher
Digital- schaltungen	UND Gatter Inverter
Analog- schaltungen	Verstärker Filter
Bauteile	Transistoren Dioden
Physik	Elektronen

↑ HDL

DFF, Latch

AND, OR, NOT

CMOS

# 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

1110101010000001010101010110101100100101  
00111100111111011001101001110111001110001  
0011011101001100110110011111110010001101  
1010001110110000110101100010000011110000  
1011000100000110101111101000101011000101  
1011001011101010110000111101001010000010  
1001010000101010001111010010111000111111  
10111100111010010000101111111010101001  
111010001011101111011111110000100001000  
0110101000111000001110110010010001001101  
1101010011111000010011101111010110011111  
1000000100001100010010001010110000100110  
101111011010101000110101010100000011100000  
0110110110110110110010101101011010101111  
1100010001000000100011110110001100101110  
1110100010110001000010010111110100111101

# Notwendigkeit von HDLs

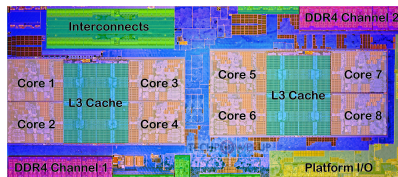
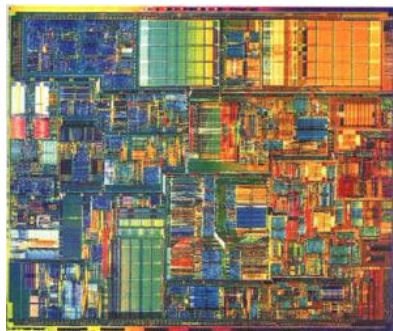
## Hardware Description Language

- ▶ Komplexität technischer Systeme steigt ständig (vgl. Moores Gesetz)

- ▶ 2000: Intel Pentium 4:  
 $42 \cdot 10^6$  Transistoren auf  $217 \text{ mm}^2$
- ▶ 2017: AMD Ryzen:  
 $4,8 \cdot 10^9$  Transistoren auf  $192 \text{ mm}^2$

⇒ ohne rechnergestützte Hilfsmittel nicht zu beherrschen

⇒ Hardware-Beschreibungssprachen zum Beherrschen von Komplexität



# Notwendigkeit von HDLs

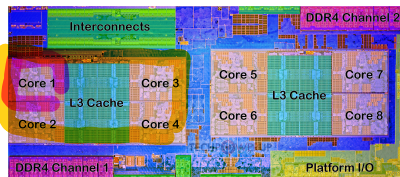
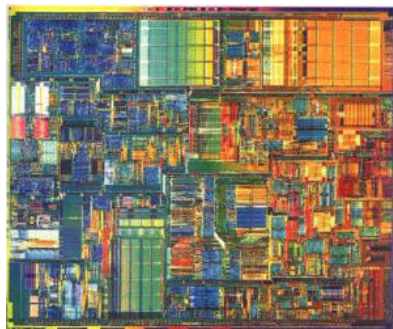
## *Hardware Description Language*

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

⇒ ohne rechnergestützte Hilfsmittel nicht zu beherrschen

⇒ Hardware-Beschreibungssprachen zum Beherrschen von Komplexität

- ▶ Hierarchie
- ▶ Modularität
- ▶ Regularität





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

# Robert Piloty, 1924 - 2013



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

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





# Robert Piloty, 1924 - 2013



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- ▶ Maßgeblich an Einführung/Entwicklung des Informatikstudiums beteiligt
  - ▶ Forschung an
    - ▶ programmgesteuerten Rechenanlagen (PERM)
    - ▶ rechnergestützter 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
- ⇒ 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
- ⇒ 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
  - ⇒ 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
  - ⇒ 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 Scala (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 Scala (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
- ⇒ Schritt von Beschreibung zur Ausführung (Semantic Gap) wird immer größer



- ▶ Simulation des funktionalen/zeitlichen Verhaltens der beschriebenen Schaltung
  - ▶ berechnete Ausgaben zu vorgegebenen Eingaben werden auf Korrektheit geprüft
  - ⇒ Fehlersuche einfacher (billiger) als in realer Hardware



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



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

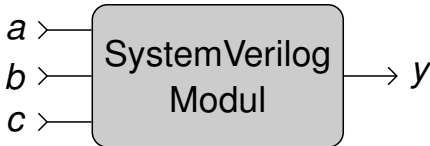
# SystemVerilog für kombinatorische Logik



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

1011010010110000110111011101001010010111  
1101000010001101101100011101110100101110  
1101101111010111001001101000010100110011  
00101110011000110111001011110011011110011  
0001111000010001011001110001000011011001  
0101100111111111000100100011011100001100  
1110000111001100110011110011100110001111  
10110100111100110000011001110001101101010  
00110010111111101111101110101011110000001  
0100010110010011011101100001010110010000  
0010101101011111010010000111101100111000  
1010110101001001010010111100011010111111  
1001110010010011100011010110111001101000  
0011100111010101100101100010101001100101  
0110001010011110101110010101101000101011  
0010000000111101101101011001001000111101

- ▶ Schnittstellenbeschreibung:
    - ▶ Eingänge
    - ▶ Ausgänge
    - ▶ (Parameter)
  - ▶ zwei Arten von Modul-Beschreibungen:
    - ▶ Struktur: Wie ist die Schaltung aus (Unter-)Modulen aufgebaut?
    - ▶ Verhalten: Was tut die Schaltung?
- ⇒ strukturelle Modul-Hierarchie mit Verhaltensbeschreibung auf unterster Ebene



# Beispiel für Verhaltensbeschreibung



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





- ▶ Unterscheidet Groß- und Kleinschreibung
  - ▶ bspw. `reset`  $\neq$  `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

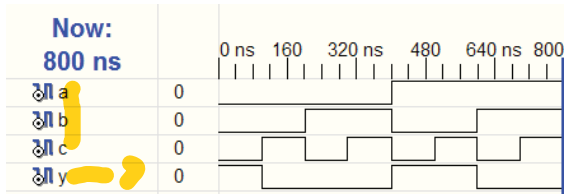


TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

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
```

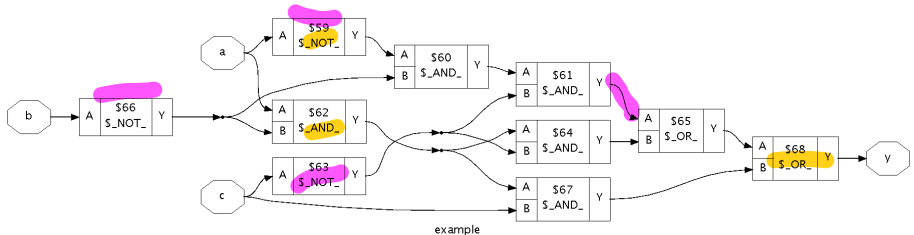
## Signalverlaufsdigramm (waves):



# Synthese von Verhaltensbeschreibungen

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
```



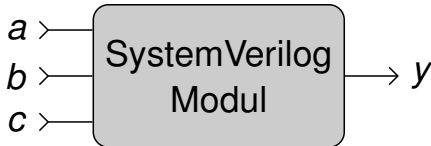
# SystemVerilog Modulhierarchie



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

1010111010011101001110000001111001010010  
1111011001010011110000100100001111110010  
1000111011110100111100111101100001010101  
1110010100110011110010101110111011110101  
0110001110100111111010101100011000100011  
0110111001110101001111011011011110100101  
0110011111011101000110110101011011100110  
111100011001011000010001000101000011111  
101101010101010000010111000000100101111010  
0100000100100010011111011101111011110101  
0100111001001001100010101010110111011110  
11010100011000110000000000001111111010101  
1010000010010101100011000111111111100101  
0101010110010111010010010010010100001100  
0001010011001001100111011111110100100010  
1101100100111011010101011000100001100101

- ▶ Schnittstellenbeschreibung:
    - ▶ Eingänge
    - ▶ Ausgänge
    - ▶ (Parameter)
  - ▶ zwei Arten von Modul-Beschreibungen:
    - ▶ **Struktur**: Wie ist die Schaltung aus (Unter-)Modulen aufgebaut?
    - ▶ Verhalten: Was tut die Schaltung?
- ⇒ strukturelle Modul-Hierarchie mit Verhaltensbeschreibung auf unterster Ebene



# Strukturelle Beschreibung: Modulinstantiierung



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

⇒ absolute Portzuweisung per Namen übersichtlicher (selbstdokumentiert)

# Bitweise Verknüpfungsoperatoren



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 Eingängen, 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

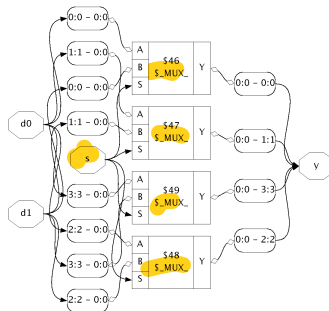
# Bedingte Zuweisung (ternär) und deren Syntheseergebnis



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

mux2.sv

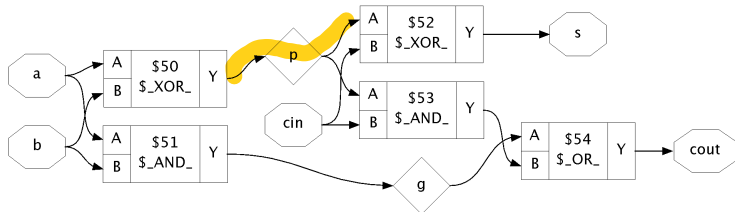
```
1 module mux2(input logic [3:0] d0, d1,  
2             input logic      s,  
3             output logic [3:0] y);  
4     assign y = s ? d1 : d0;  
5 endmodule
```



# Interne Verbindungsknoten (Signale)

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

- ▶  $\sim$  NOT (höchste Präzedenz)
- ▶  $*$ ,  $/$ ,  $\%$  Multiplikation, Division, Modulo
- ▶  $+$ ,  $-$  Addition, Subtraktion
- ▶  $\ll$ ,  $\gg$  logischer Shift
- ▶  $\lll$ ,  $\ggg$  arithmetischer Shift
- ▶  $<$ ,  $\leq$ ,  $>$ ,  $\geq$  Vergleich
- ▶  $==$ ,  $!=$  gleich, ungleich
- ▶  $\&$ ,  $\sim\&$  AND, NAND
- ▶  $\wedge$ ,  $\sim\wedge$  XOR, XNOR
- ▶  $|$ ,  $\sim|$  OR, NOR
- ▶  $?:$  ternärer Operator (niedrigste Präzedenz)

# Syntax für numerische Literale



- ▶ Syntax:  $\langle N \rangle' \langle B \rangle \langle \text{wert} \rangle$ 
  - ▶  $\langle N \rangle$  = Bitbreite
  - ▶  $\langle B \rangle$  = 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

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
```

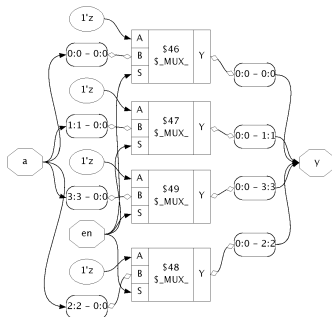
# Hochohmiger Ausgang (Z) und dessen *falsche* Synthese



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

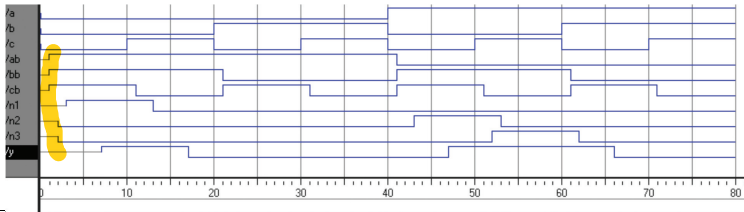


# Verzögerungen: # Zeiteinheiten



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

0010000010000000101011001011100100100010  
111001010101110011111101101110101011010  
10011000100010001001011001100000110101011  
01001110000101011110110100111100001100010  
0010111011100000111001000000110011000011011  
01110111110010001101110111001011001100011  
10111001101010010000001011101110100001100  
01100011011001100100101011111000010100  
11111011001111011001101001101010000011001  
000001010110000010011000000000100011000111  
111011101000001001011111001000001100100100  
111110001101111001000000100000100000000110  
1111101000111000011000001001001110111101101  
100010001001101100001101011101001011111101  
01110100110010010011101001010101111111101  
000001010000001011100100010101111110100010



- ▶ Historie von Hardwarebeschreibungssprachen
- ▶ SystemVerilog für kombinatorische Logik
- ▶ SystemVerilog Modulkonstruktion
  
- ▶ Nächste Vorlesung behandelt
  - ▶ SystemVerilog für sequentielle Logik