

Digitaltechnik

Wintersemester 2021/2022

10. Vorlesung












TECHNISCHE
UNIVERSITÄT
DARMSTADT





Umfrage zur letzten Woche

1. Hardwarebeschreibungssprachen
2. SystemVerilog für kombinatorische Logik
3. SystemVerilog Modulhierarchie
4. Zusammenfassung

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

Überblick der heutigen Vorlesung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ Hardwarebeschreibungssprachen
- ▶ SystemVerilog für kombinatorische Logik
- ▶ SystemVerilog Modulhierarchie



Harris 2013/2016
Kap. 4.1 - 4.3

Agenda












TECHNISCHE
UNIVERSITÄT
DARMSTADT

1. Hardwarebeschreibungssprachen

2. SystemVerilog für kombinatorische Logik

3. SystemVerilog Modulhierarchie

4. Zusammenfassung

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

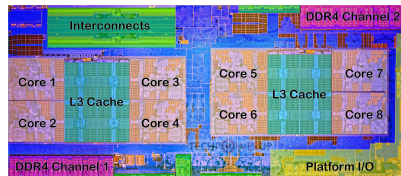
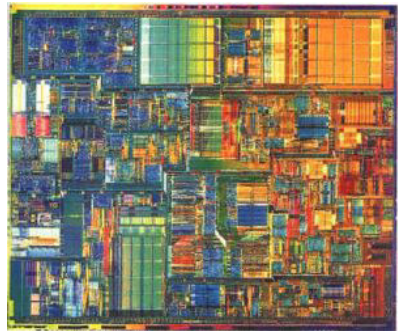
Notwendigkeit von HDLs

Hardware Description Language

- ▶ Komplexität technischer Systeme steigt ständig (vgl. Moore'sches 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

- ⇒ ohne rechnergestützte Hilfsmittel nicht zu beherrschen
- ⇒ Hardwarebeschreibungssprachen 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



- ▶ **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)



- ▶ 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
 - ▶ 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
- ▶ *Synthese* übersetzt Hardware-Beschreibungen in *Netzliste*
- ▶ Netzliste
 - ▶ beschreibt die Schaltungselemente (Logikgatter) und die Verbindungsknoten
 - ▶ entspricht Registertransferebene
 - ▶ kann auf Gatter-Bibliothek einer konkreten Zielarchitektur abgebildet werden (Technology-Mapping)
 - ▶ wenige CMOS-Basisgatter für Application-Specific Integrated Circuits (ASICs)
 - ▶ wenige kleine Lookup-Tabellen für Field-Programmable Gate Arrays (FPGAs)
- ▶ **WICHTIG:** für effiziente Hardware-Beschreibung muss HDL-Programmierer *immer* die Zielarchitektur im Auge behalten

Agenda



TECHNISCHE
UNIVERSITÄT
DARMSTADT

1. Hardwarebeschreibungssprachen

2. SystemVerilog für kombinatorische Logik

3. SystemVerilog Modulhierarchie

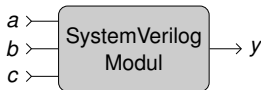
4. Zusammenfassung

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

Wiederholung: SystemVerilog Module



- ▶ Ein *Modul* beschreibt wie eine Aufgabe (Berechnung) durchgeführt wird
 - ▶ Ähnlich einer *Funktion* in Programmiersprachen
 - ▶ Schnittstellenbeschreibung:
 - ▶ Eingänge
 - ▶ Ausgänge
 - ▶ (Parameter)
 - ▶ zwei Arten von Modul-Beschreibungen:
 - ▶ Struktur: Wie ist die Schaltung aus (Sub-)Modulen aufgebaut?
 - ▶ Verhalten: Was tut die Schaltung?
- ⇒ strukturelle Modul-Hierarchie mit Verhaltensbeschreibung auf unterster Ebene



Beispiel für Verhaltensbeschreibung



comb/example.sv

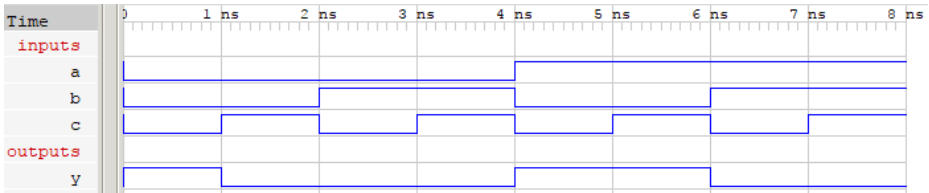
```
1 module example(input logic a, b, c, output logic y);  
2  
3     assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b & c;  
4  
5 endmodule
```

- ▶ module Beginn der 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 der Schnittstellenbeschreibung

comb/example.sv

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

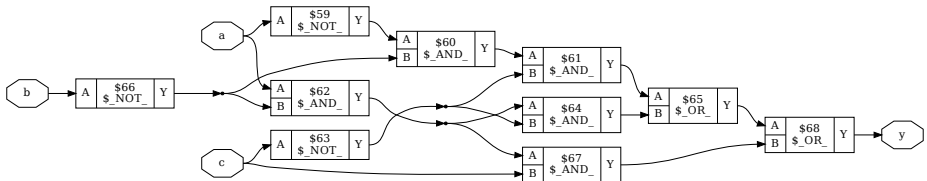
Plot mit Open-Source Tools Icarus Verilog + GTKWave



comb/example.sv

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

Plot mit Open-Source Tools YoSys + GraphViz





- ▶ Unterscheidet Groß- und Kleinschreibung
 - ▶ bspw. `reset` \neq `Reset`
- ▶ Bezeichner für Modul- und Signalnamen dürfen nicht mit Ziffern anfangen
 - ▶ bspw. `2mux` ungültig
- ▶ Anzahl von Leerzeichen, Leerzeilen und Tabulatoren irrelevant
- ▶ Kommentare:
 - ▶ *// Kommentar bis zum Ende der Zeile*
 - ▶ */* Kommentar über
mehrere Zeilen */*



Pause & Umfrage bis hier

Agenda












TECHNISCHE
UNIVERSITÄT
DARMSTADT

1. Hardwarebeschreibungssprachen

2. SystemVerilog für kombinatorische Logik

3. SystemVerilog Modulhierarchie

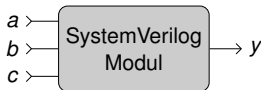
4. Zusammenfassung

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

Wiederholung: SystemVerilog Module



- ▶ Ein *Modul* beschreibt wie eine Aufgabe (Berechnung) durchgeführt wird
 - ▶ Ähnlich einer *Funktion* in Programmiersprachen
 - ▶ Schnittstellenbeschreibung:
 - ▶ Eingänge
 - ▶ Ausgänge
 - ▶ (Parameter)
 - ▶ zwei Arten von Modul-Beschreibungen:
 - ▶ Struktur: Wie ist die Schaltung aus (Sub-)Modulen aufgebaut?
 - ▶ Verhalten: Was tut die Schaltung?
- ⇒ strukturelle Modul-Hierarchie mit Verhaltensbeschreibung auf unterster Ebene



Strukturelle Beschreibung: Modulinstantiierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

comb/and3.sv

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

comb/inv.sv

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

comb/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



comb/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
```

comb/nand3_named.sv

```
1 module nand3_named(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 (selbstdokumentierend)

Bitweise Verknüpfungsoperatoren



comb/gates.sv

```
1 module gates (input logic [3:0] a, b, // 4 bit Vektoren
2               output logic [3:0] y1,y2,y3,y4,y5);
3
4   /* Fünf unterschiedliche Logikgatter
5      mit zwei Eingängen, jeweils 4 bit Vektoren */
6   assign y1 = a & b; // AND
7   assign y2 = a | b; // OR
8   assign y3 = a ^ b; // XOR
9   assign y4 = ~(a & b); // NAND
10  assign y5 = ~(a | b); // NOR
11
12 endmodule
```

Reduktionsoperatoren (unär)



comb/and8.sv

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

► analog:

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

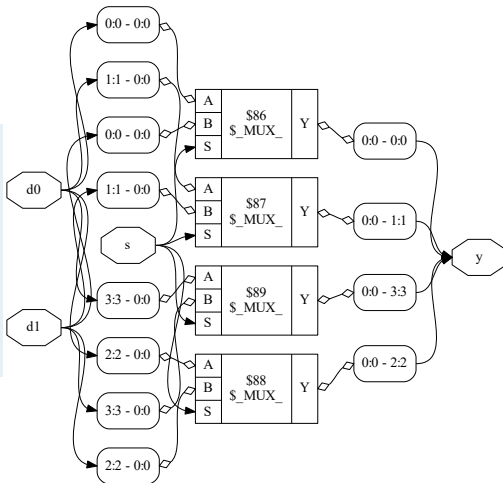
Bedingte Zuweisung (ternär) und deren Syntheseresultat



TECHNISCHE
UNIVERSITÄT
DARMSTADT

comb/mux2x4.sv

```
1 module mux2x4
2   (input logic [3:0] d0,d1,
3    input logic      s,
4    output logic [3:0] y);
5
6   /* if s=1 then y=d1;
7      else y=d0; */
8   assign y = s ? d1 : d0;
9
10  endmodule
```

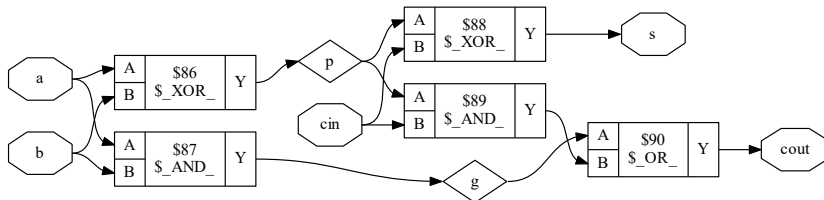


Interne Verbindungsknoten (Signale)



comb/fulladder.sv

```
1 module fulladder(input logic a, b, cin,  
2                  output logic s, cout);  
3  
4 logic p, g; // interne Verbindungsknoten  
5 assign p = a ^ b;  
6 assign g = a & b;  
7 assign s = p ^ cin;  
8 assign cout = g | (p & cin);  
9  
10 endmodule
```



Bindung von Operatoren (Präzedenz)



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- ▶ `[]` Zugriff auf Vektorelement (höchste Präzedenz)
- ▶ `~, !, &, |, ^` unäre Operatoren: NOT, Negation, Reduktion
- ▶ `*, /, %` Multiplikation, Division, Modulo
- ▶ `+, -` Addition, Subtraktion
- ▶ `<<, >>, <<<, >>>` logischer und arithmetischer Shift
- ▶ `<, <=, >, >=` Vergleich
- ▶ `==, !=` gleich, ungleich
- ▶ `&, ~&` bitweises AND, NAND
- ▶ `^, ~^` bitweises XOR, XNOR
- ▶ `|, ~|` bitweises OR, NOR
- ▶ `&&` logisches AND (Vektoren sind genau dann wahr,
- ▶ `||` logisches OR wenn wenigstens ein Bit 1 ist)
- ▶ `?:` ternärer Operator
- ▶ `{ }` Konkatenation (niedrigste Präzedenz)

- ▶ Syntax: `<N>'<wert>`
 - ▶ `<N>` = Bitbreite
 - ▶ `` = Basis (d,b,o,h)
 - ▶ beide Angaben optional (default: 32'd)
 - ▶ Werte werden mit führenden 0en bis zur Bitbreite aufgefüllt
 - ▶ 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



comb/concat.sv

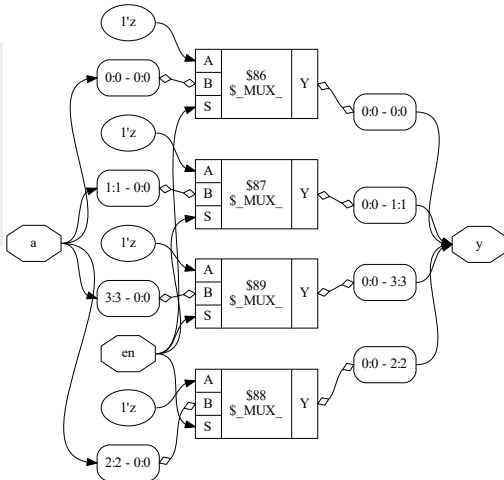
```
1 module concat(input logic [2:0] a, b, output logic [11:0] y);  
2  
3 // y = a[2] a[1] b[0] b[0] b[0] a[0] 1 0 0 0 1 0  
4 assign y = {a[2:1], {3{b[0]}}, a[0], 6'b100010};  
5  
6 endmodule
```

Hochohmiger Ausgang (Z) und dessen *falsche* Synthese



comb/tristate.sv

```
1 module tristate
2   (input  logic [3:0] a,
3    input  logic      en,
4    output logic [3:0] y);
5
6   assign y = en ? a : 4'bz;
7
8 endmodule
```

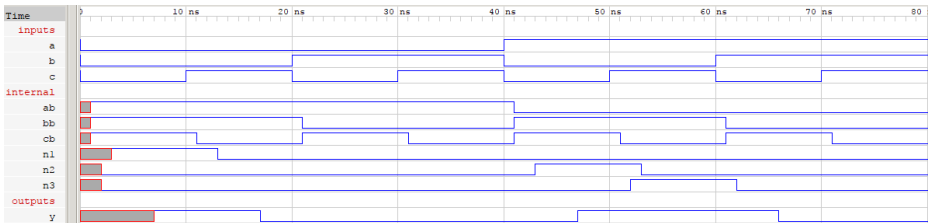


Verzögerungen: # Zeiteinheiten



comb/example_delay.sv

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



Agenda












TECHNISCHE
UNIVERSITÄT
DARMSTADT

1. Hardwarebeschreibungssprachen

2. SystemVerilog für kombinatorische Logik

3. SystemVerilog Modulhierarchie

4. Zusammenfassung

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



- ▶ Hardwarebeschreibungssprachen
- ▶ SystemVerilog für kombinatorische Logik
- ▶ SystemVerilog Modulhierarchie

- ▶ Nächste Vorlesung behandelt
 - ▶ SystemVerilog für sequentielle Logik