Verilog/VHDL

Mehdi Khayati Sarkandi – Uni Siegen

Verilog/VHDL

Hardware Description Language (HDL)

- Werkzeug zum Entwurf komplexer digitaler Schaltungen,
- zur Simulation des Systemverhaltens,
- zur Überprüfung auf korrekte Funktionsfähigkeit (Verifikation)

Verilog/VHDL

Hardware Description Language (HDL)

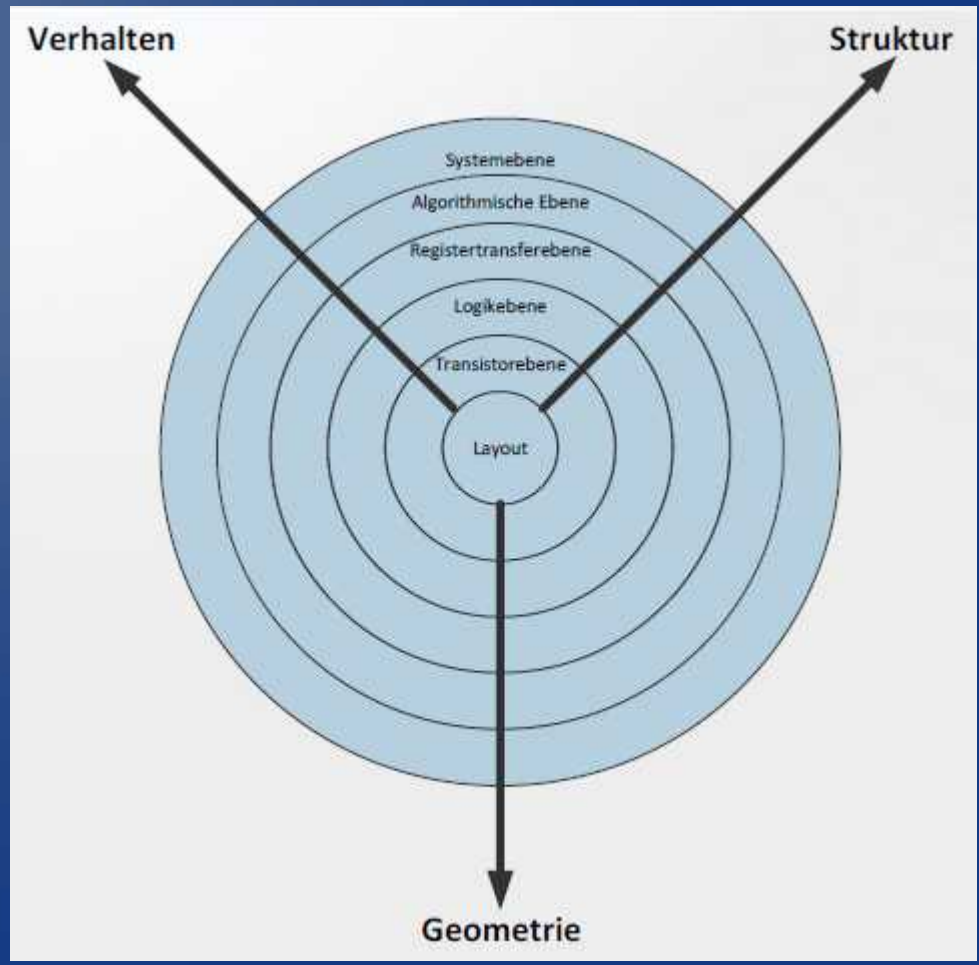
- textuell statt graphisch
- neben sequentieller auch **zeitlich *parallele*** Bearbeitung von Anweisungen
- Verilog und VHDL sind die zwei bekanntesten HDL

Verilog/VHDL

Designmodelle

Sichtweisen:

- Verhaltensmodell
- Strukturmodell
- Physikalisches Modell



Verilog/VHDL

Abstraktionsebenen:

	Verhaltensdesign	**Strukturdesign**	**Pysikalisches Design**
1.Systemebene	Systemspezifikation	CPUs, grobe Funktionsblöcke	Partitionierung
2.Algorithmische Ebene	Algorithmen	Subsysteme, Busse, Prozessor	**MCM** [Multi Chip Module]
3.Registertransferebene	Register-Transfers	Module, ALU, MUX, Register	ASICs, FPGAs
4.Logikebene	Boolesche Gleichung	Gatter, Flipflops	Zellen
5.Transistorebene (Schaltkreisebene)	Differentialgleichung, Transistorfunktion	Transistoren, Leitungsstücke	Rasterdiagramm für Transistorlayout
6.Layoutebene	Konkretes Layout mit Maskenebene		

Verilog/VHDL

Unterschiede und Gemeinsamkeiten von VHDL/Verilog

- VHDL wie Ada und Pascal, Verilog wie C
- Verilog: einfachere Struktur
- Verilog ist case-sensitive, VHDL nicht
- Bibliotheksmanagement fehlt bei Verilog
- VHDL hat keine automatische Typkonvertierung im Gegensatz zu Verilog
- VHDL: benutzerdefinierte Datentypen
- Verilog: benutzerdefinierte Gatter/Module, Primitive

Modulbeschreibung Verilog:

module module-Name (Liste von Ein- und Ausgängen)

Programm-Code; // body

endmodule

Modulbeschreibung VHDL:

entity <Entityname> **is**

port(<Deklaration der Ein- und Ausgänge>**)**;
[< Entitydeklarationen >;]

end <Entityname>;

architecture <Architekturname> **of** <Entityname> **is**
[< Architekturdeklarationen >;]

begin

{<VHDL-Anweisungen>;}

end <Architekturname>;

Verilog/VHDL

Einige Aspekte der Verilog-Sprache

- Zahlen: abc = 4'b0110 n;
- Operatoren: Arithmetische, Logische, Bitweise, Reduktions-, Bedingte, Vergleich-, Auswahl-, Schleifen-Operatoren

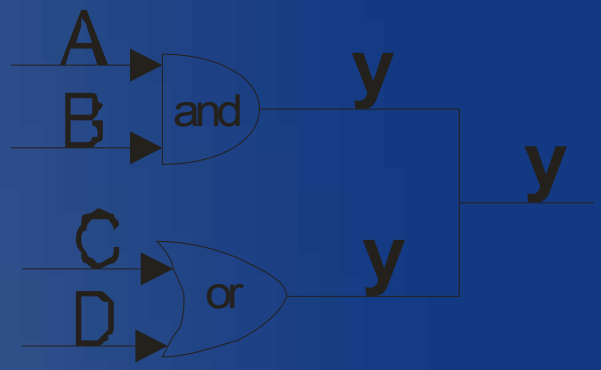
  z. B. Reduktion: a= 3'b101; &a ist gleich 1'b0
  z. B. Vergleich: (3'b01x === 3b'01x) liefert wahr,
  (3'b01z === 3'b01x) liefert falsch.

Verilog/VHDL

- Datentypen:

  - net-Typ: **wire**, **wor**, **wand** etc.

    repräsentiert elektrische Signale zwischen den Komponenten



  - Registertyp: **reg** etc.

    stellt einen Speicher dar

Verilog/VHDL

- Prozeduralblöcke: (Verhaltensmodell)

  wie herkömmliche Programmiersprachen,
  mit „**begin**“ und „**end**“

  - blockierende und nicht-blockierende Zuweisung

```
begin
    a1 <= x;
    a2 <= a1;
    a3 <= a2;
    y <= a3;  // y bekommt den Wert von a3!
end
```

```
begin
    a1 = x;
    a2 = a1;
    a3 = a2;
    y = a3;  // y bekommt den Wert von x!
end
```

Verilog/VHDL

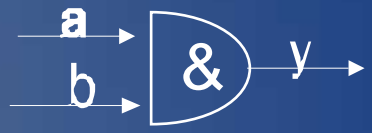
- **initial**-Prozeduralblock, der nur einmal am Anfang der Simulation ausgeführt wird
- **always**-Prozeduralblöcke, die am Anfang des Programmcodes ***parallel*** laufen
  - endlose Schleifen,
  - nur die Zuweisung von „reg“-Typen
  - Änderung abhängig von der Sensitivitätsliste (kombinatorisch, sequentiell)

```
always @ (Ereignis_liste)
    begin
        Anweisungen_1;
        y = a+b;
        Anweisungen_n;
    end
```

Verilog/VHDL

- Verilog-Primitive: vordefinierte Module

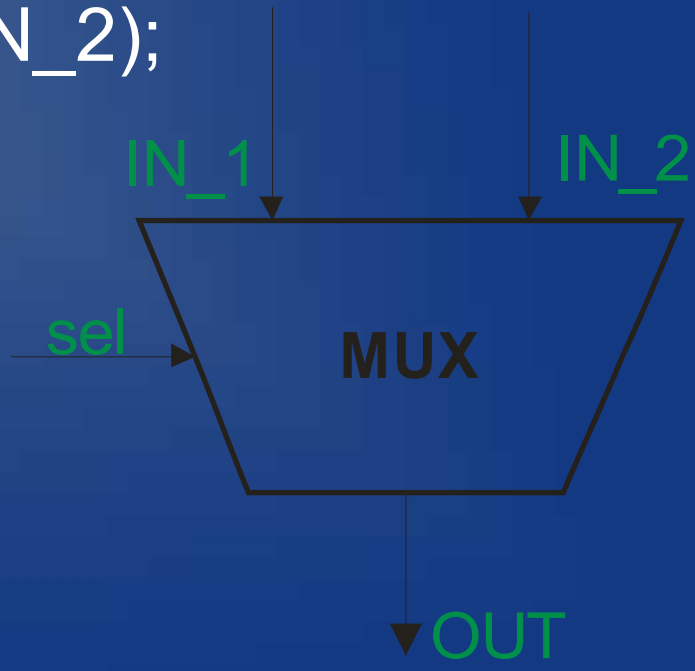
  z. B.: **and** (y, a, b);



- user-definierte Module: Anwendung durch Instanzen

  z. B.: **MUX** M1 (OUT, sel, IN_1, IN_2);

```
module MUX ( IN_1, IN_2, sel, OUT);
      input [2:0] IN_1, IN_2;
      input sel;
      output [2:0] OUT;
      wire OUT;
      assign OUT = (sel) ? IN_1 : IN_2;
endmodule
```

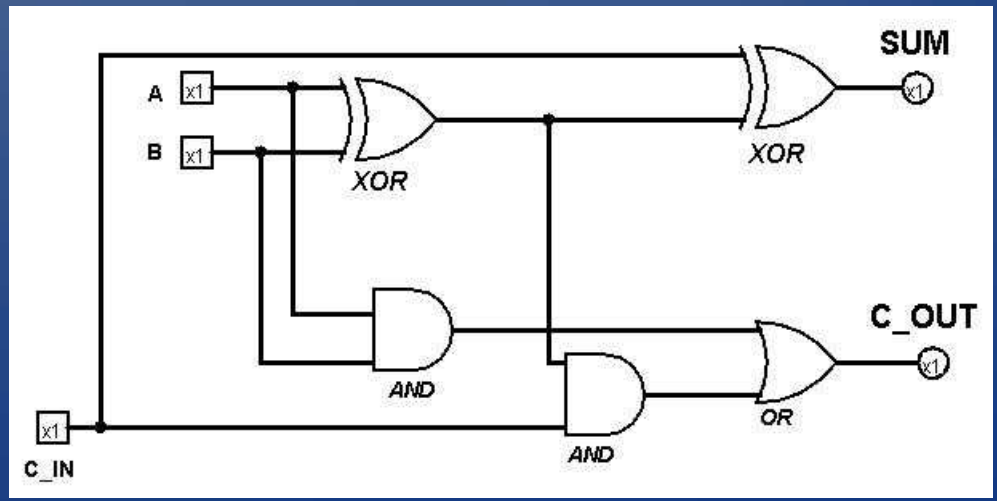


Verilog/VHDL

Volladdierer

```
entity VOLLADD is
    port (  A, B, C_IN: in bit;
            SUM, C_OUT: out bit);
end VOLLADD
architecture VERHALTEN of VOLLADD is
begin
    SUM <= A xor B xor C_IN;
    C_OUT <= (A and B) or (C_IN and (A xor B));
end VERHALTEN
```

```
module VOLLADD (SUM, C_OUT, A, B, C_IN);
  input  A, B, C_IN;
  output SUM, C_OUT;
    always @ (A, B, C_IN)
      begin
        SUM = A^B^C_IN;
        C_OUT = (A & B) | (B & C_IN) | (A & C_IN);
      end
endmodule
```



Verilog/VHDL

- Verifikation

  - Überprüfung der Funktionsfähigkeit
  - Überprüfung der Ausführungszeit
  - Testbench: mit initialisierten Anfangswerten, kann dokumentiert oder gespeichert werden
  - Signalanalyseprogramm

Verilog/VHDL

Synthesewerkzeuge

- **Logiksynthese** beinhaltet Analyse, Übersetzung in logische Operatoren und Speicher und die anschließende Eliminierung von Redundanzen

  → Gatternetzliste mit logischen Funktionen der Gatter und deren Signalverzögerung (logische Gatter, FlipFlop etc.).
- **Layoutsynthese** (Place & Route) bildet mit Hilfe von Synthesewerkzeugen und der Verwendung von Bibliotheks- und Technologie-Informationen eine Gatternetzliste in einem geometrischen Layout auf dem Chip ab.

Verilog/VHDL

Vielen Dank für

Ihre Aufmerksamkeit!