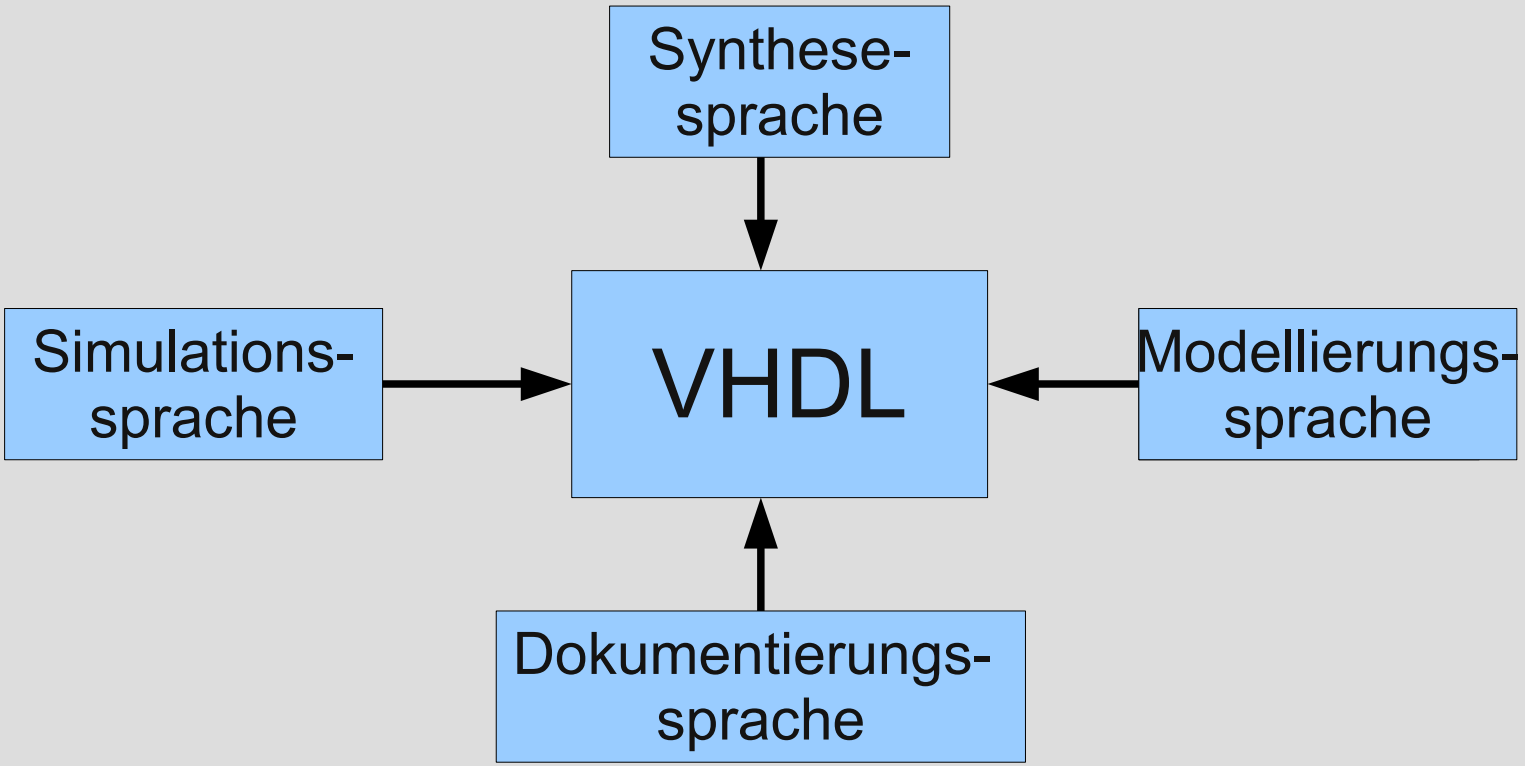
Einführung in VHDL

- Digitale Systeme haben immer größere Bedeutung erlangt. Komplexität wurde dabei immer größer, sodass die Entwicklung digitaler Systeme zu weiten Teilen nur noch mit Computerunterstützung durchführbar ist.
- Zur Beschreibung von Hardware existieren *Hardwarebeschreibungssprachen,* wie etwa VHDL, ABEL, Verilog u.a., die Architektur, Struktur und Verhalten von digitalen Systemen beschreiben.

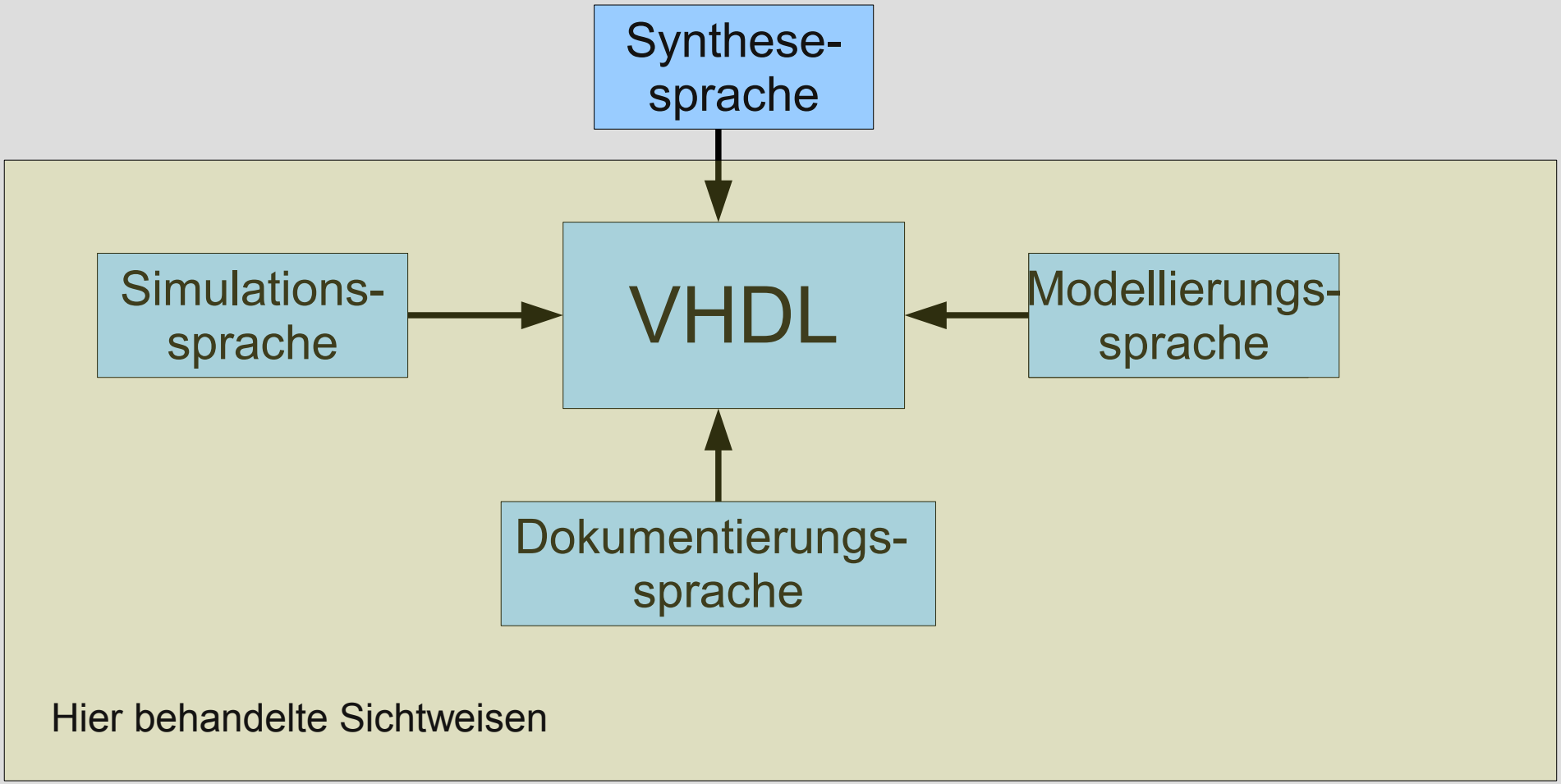
Einführung in VHDL (2)

- **V**ery High Speed Integrated Circuit **H**ardware **D**escription **L**anguage
- Spezifikation: Anfang der 80er Jahre
- Standardisierung: 1987, 1993 von IEEE
- Wichtige Schritte des Hardwaredesign-Prozesses, die unterstützt werden:
  - Simulation und Verifikation des modellierten Systems
  - Optimierung
  - Synthese der beschriebenen Hardware für eine bestimmte Technologie

Einführung in VHDL (3)



Einführung in VHDL (3)



Einführung in VHDL (4)

- Im Gegensatz zu „normalen“ Programmiersprachen wie C++ entsteht durch die Compilierung von VHDL-Programmen kein direkt ausführbarer Code.
- Eine VHDL-Beschreibung ist nur im Zusammenhang mit einem Logiksimulator (=Testumgebung) verifizierbar.

Einführung in VHDL (5)

- Enthält neben den aus „normalen“ Programmiersprachen bekannten Datentypen, Kontrollstrukturen, Prozeduren, ... auch einige Konzepte, welche die die speziellen Eigenschaften von Hardware reflektieren
- Syntax ist stark an *Pascal* angelehnt.

Einführung in VHDL (6)

- Beispiel (Verwendung wie „normale“ PS (nicht synthetisierbar)):

```
 --  Hello world program.
use std.textio.all; --  Imports the standard textio package.

--  Defines a design entity, without any ports.
entity hello_world is
end hello_world;

architecture behaviour of hello_world is
begin
   process
      variable l : line;
   begin
      l := new String'("Hello world!");
      writeline (output, l);
      wait;
   end process;
end behaviour;
```

Aufbau von VHDL-Beschreibungen

- Einführung
  - Reales System verfügt üblicherweise über Schnittstellen und eine interne Realisierung, in der Verbindungen und Komponenten (z.B. logische Gatter, FlipFlops usw. enthalten sind.
  - Schnittstelle: ***entity***
  - Interne Realisierung: ***architecture***

Aufbau von VHDL-Beschreibungen

- Einführung
  - Schnittstelle: ***entity***
  - Interne Realisierung: ***architecture***

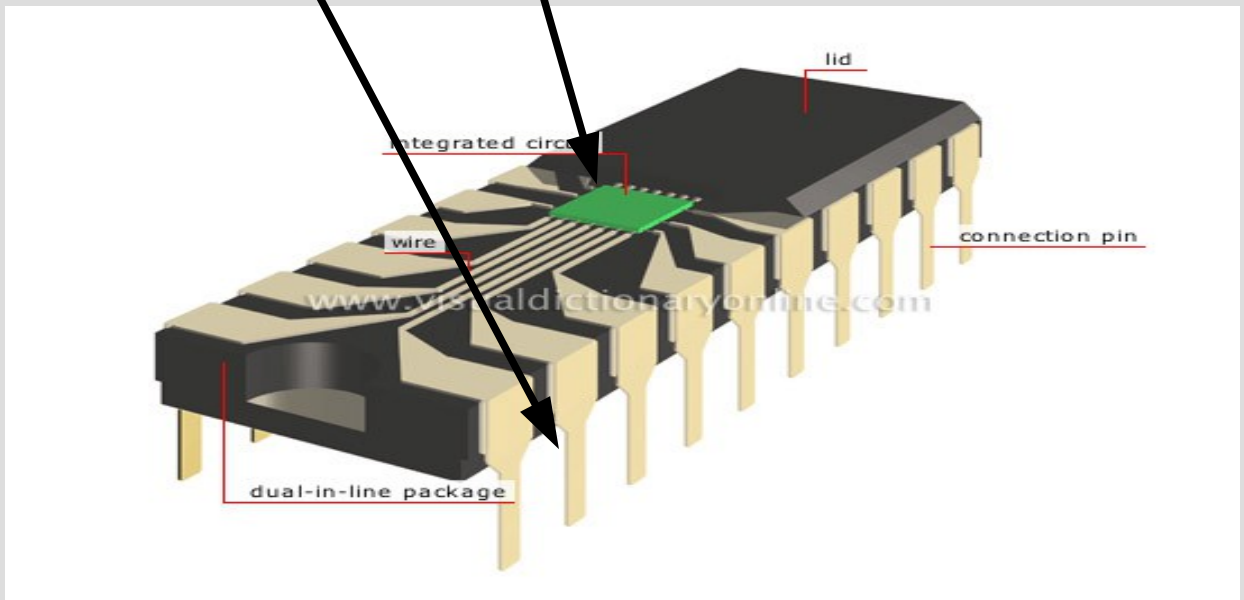
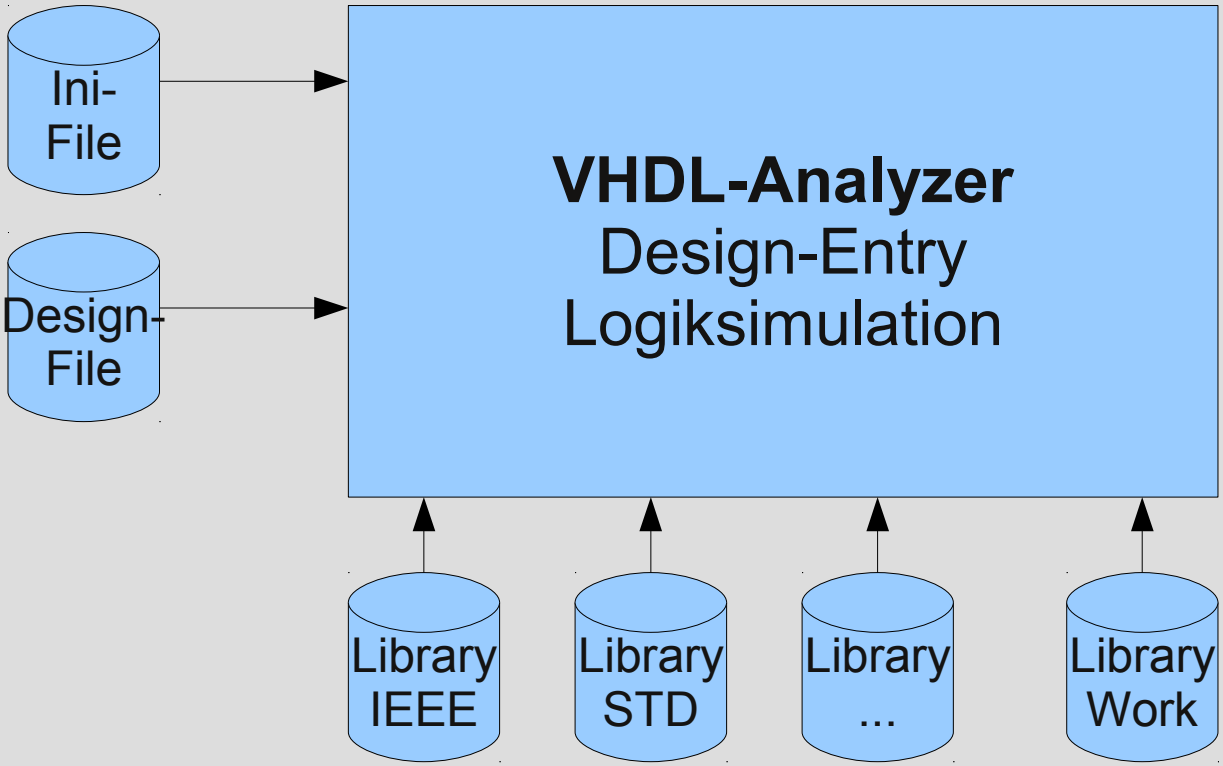
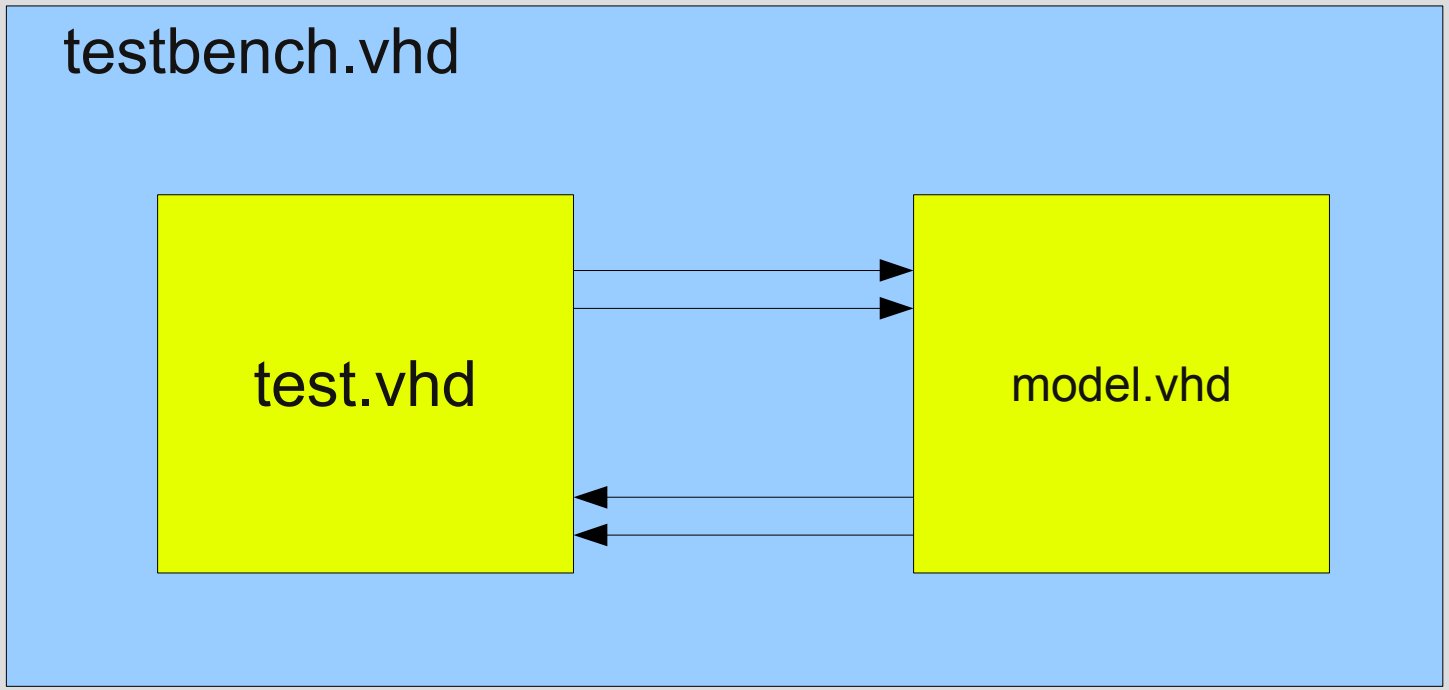


Bild: http://elektroniki.blogspot.com/2010/11/history-of-integrated-circuit-aka.html

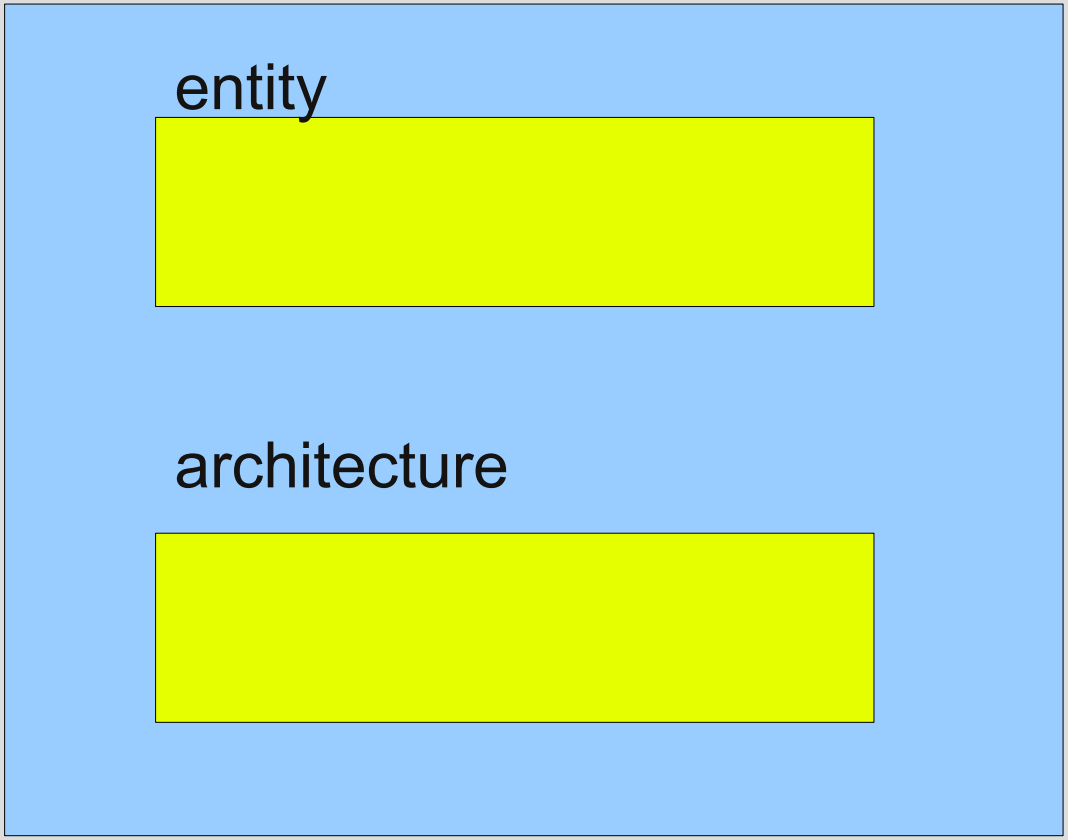
Aufbau von VHDL-Beschreibungen (2)



Aufbau von VHDL-Beschreibungen (3)



Aufbau von VHDL-Beschreibungen (4)



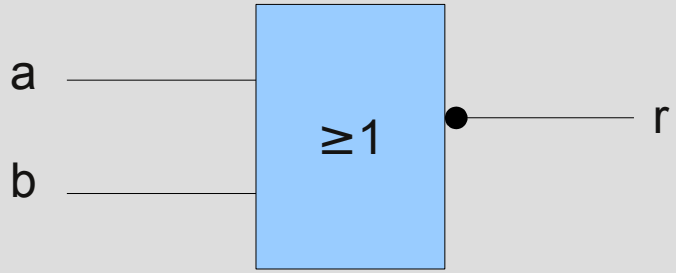
Minimalstruktur eines VHDL-Modells

Aufbau von VHDL-Beschreibungen (5)

- Schnittstellenbeschreibung
  - Schlüsselwort ***entity***
  - Spezifikation der Ein- und Ausgänge des Systems durch das Schlüsselwort ***port***
  - Daneben können Konstanten, sogenannte *Generics* definiert werden, die zum Zeitpunkt der Übersetzung der VHDL-Quelldatei Einfluss auf bestimmte Architekturparameter nehmen (Busbreite, Verzögerungszeiten,...)

Aufbau von VHDL-Beschreibungen (6)

- Beispiel:
  - Zweieingängiges NOR-Gatter.




```
entity mnor is
    port
    (
        a, b:   in bit;
        r:      out bit;
    );
end mnor2;
```

Aufbau von VHDL-Beschreibungen (7)

- Als Typ ist neben bit oder bit_vector u.a. der Typ ***std_logic*** oder ***std_logic_vector*** möglich.
- Der Typ std_logic erfordert das Einbinden der Library IEEE vor der Entity-Deklaration durch:

```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
```

Aufbau von VHDL-Beschreibungen (8)

- Nach IEEE 1164 kann ein Signal folgende Zustände annehmen:
  - U : Uninitialized
  - X : Forcing Unknown
  - 0 : Forcing 0
  - 1 : Forcing 1
  - Z : High Impedance
  - W : Weak Unknown
  - L : Weak 0
  - H : Weak 1
  - D : Don‘t Care

Aufbau von VHDL-Beschreibungen (9)

- Mit std_logic wird ein Logik-Signal als 9-wertig definiert. Diese 9-wertige Logik ist heute faktisch Industrie-Standard. Durch das Einbinden der Library IEEE.std_logic_1164.all stehen bestimmte Funktionen zur Verfügung; das gilt auch für die Library
  - ***IEEE.std_logic_unsigned.all***.

Aufbau von VHDL-Beschreibungen (9a)

std_logic_1164 NOT

U	X	0	1	Z	W	L	H	D
U	X	1	0	X	X	1	0	x

Aufbau von VHDL-Beschreibungen (9b)

std_logic_1164 AND

	U	X	0	1	Z	W	L	H	D
U	U	U	0	U	U	U	0	U	U
X	U	X	0	X	X	X	0	X	X
0	0	0	0	0	0	0	0	0	0
1	U	X	O	1	X	X	0	1	X
Z	U	X	0	X	X	X	0	X	X
W	U	X	0	X	X	X	0	X	X
L	0	0	0	0	0	0	0	0	0
H	U	X	0	1	X	X	0	1	X
D	U	X	0	X	X	X	0	X	X

Aufbau von VHDL-Beschreibungen (9c)

std_logic_1164 OR

	U	X	0	1	Z	W	L	H	D
U	U	U	U	1	U	U	U	1	U
X	U	X	X	1	X	X	X	1	X
0	U	X	0	1	X	X	0	1	X
1	1	1	1	1	1	1	1	1	1
Z	U	X	X	1	X	X	X	1	X
W	U	X	X	1	X	X	X	1	X
L	U	X	0	1	X	X	0	1	X
H	1	1	1	1	1	1	1	1	1
D	U	X	X	1	X	X	X	1	X

Aufbau von VHDL-Beschreibungen (9)

- Im weiteren soll aber nur die zweiwertige Logik (Datentyp bit oder bit_vector) betrachtet werden.

Aufbau von VHDL-Beschreibungen (10)

- Architekturbeschreibung
  - Eigentliche Realisierung der Entity
  - Schlüsselwort ***architecture***
  - Zu jeder Architektur gehört eine Entity, eine Entity kann aber mehrere Realisierungen haben (siehe später)
  - Beispiel (**NOR-Gatter**)

```
architecture a of mnor2 is
        begin
                r <= not(a or b);
        end architecture a;
```

Aufbau von VHDL-Beschreibungen (11)

- Modellierungssichtweisen
  - Verhaltensmodellierung
  - Datenflussmodellierung
  - Strukturmodellierung

Aufbau von VHDL-Beschreibungen (11)

- Modellierungssichtweisen
  - Verhaltensmodellierung
  - Datenflussmodellierung
  - Strukturmodellierung
  - Beispiel: 2-bit Volladdierer

```
entity fulladder is
        port
        (
            a,b,cin:    in bit;
            r, cout:    out bit
        );
    end fulladder;
```

Verhaltensmodellierung

- Beschreibbar mit
  - Signalzuweisungsbefehlen;
  - Prozessen;
  - ‘Concurrent’ - Prozeduraufrufen;
  - ‘Concurrent’ - Regelüberprüfungen.

Verhaltensmodellierung (2)

Verhaltensmodell Volladder mit Signalzuweisungen

```
architecture behavioral of adder is
begin
l1:   r <= '0' when (a = '0' and b = '0' and cin = '0') else
          '1' when (a = '0' and b = '0' and cin = '1') else
          '1' when (a = '0' and b = '1' and cin = '0') else
          '0' when (a = '0' and b = '1' and cin = '1') else
          '1' when (a = '1' and b = '0' and cin = '0') else
          '0' when (a = '1' and b = '0' and cin = '1') else
          '0' when (a = '1' and b = '1' and cin = '0') else
          '1' when (a = '1' and b = '1' and cin = '1');
l2:   cout <= '0' when (a = '0' and b = '0' and cin = '0') else
          '0' when (a = '0' and b = '0' and cin = '1') else
          '0' when (a = '0' and b = '1' and cin = '0') else
          '1' when (a = '0' and b = '1' and cin = '1') else
          '0' when (a = '1' and b = '0' and cin = '0') else
          '1' when (a = '1' and b = '0' and cin = '1') else
          '1' when (a = '1' and b = '1' and cin = '0') else
          '1' when (a = '1' and b = '1' and cin = '1');
end architecture behavioral;
```

Verhaltensmodellierung (3)

Verhaltensmodell D-FF mit Prozessen

```
entity d_ffbit is
        port( CLK, D, R : in bit;
                        Q          : inout bit);
end d_ffbit;

architecture A2_DFF of d_ffbit is
begin
        P1: process( CLK, R )
                begin
                if R = '1' then
                        Q <= '0'after 5 ns;
                        elsif( CLK'event and CLK = '1' ) then
                                Q <= D after 5 ns;
                        -- else Q <= Q;  Kommentar
                end if;
               end process;
end A2_DFF;
```

Verhaltensmodellierung (3)

Verhaltensmodell D-FF mit Prozessen

```
entity d_ffbit is
        port( CLK, D, R : in bit;
                        Q         : inout bit);
end d_ffbit;

architecture A2_DFF of d_ffbit is
begin
        P1: process( CLK, R )
                begin
                if R = '1' then
                        Q <= '0'after 5 ns;
                        elsif( CLK'event and CLK = '1' ) then
                                Q <= D after 5 ns;
                        -- else Q <= Q;  Kommentar
                end if;
                end process;
end A2_DFF;
```

Der Process wird von Ereignissen der Signale, die in der „Sensitivity-List“ (CLK,R) enthalten sind, getriggert. Jedes Ereignis der „Sensitivity“-List verursacht die Ausführung des Prozesses.

Datenflussmodellierung

- Datenflussmodellierung

```
architecture dataflow of adder is
begin
        r <= cin xor (a xor b);
        cout <= (a and b) or (a and cin)
xor (b and cin);
end architecture dataflow;
```

Strukturmodellierung

- Ein Strukturmodell nutzt zur Realisierung der logischen Funktion bereits in VHDL beschriebeneTeilkomponenten (z. B. Realisierung eines Mikroprozessors durch Synthese aus ALU, Internal Cache, Register).
- Strukturmodelle beschreiben, wie die einzelnen Teilkomponenten untereinander zu verbinden sind.

Strukturmodellierung (2)

```
architecture structural of adder is
  component mxor2
      port
      (
            a1, a2: in bit;
            c:      out bit
      );
  end component;
  component mand2
      port
      (
            a1,a2:  in bit;
            c:      out bit
      );
  end component;
  component mor3
      port
      (
            a1,a2,a3:     in bit;
            c:            out bit
      );
  end component;
```

Strukturmodellierung (2)

```
    signal t1,t2,t3,t4:     bit;

begin
    p1:     mxor2 port map(a,b,t1);
    p2:     mxor2 port map(t1,cin,r);
    p3:     mand2 port map (a,b,t2);
    p4:     mand2 port map (a,cin,t3);
    p5:     mand2 port map (b,cin,t4);
    p6:     mor3 port map (t2,t3,t4,cout);
end architecture structural;
```

Signalzuweisung

- Eine Signalzuweisung (SIGNAL Assignment) ist ein ‘Concurrent’ - Statement; sie ist das wichtigste ‘Behavioral’ - Statement von VHDL. Signale übertragen Informationen zwischen Prozessen.
- ***Syntax: signal_name <= wert [after time expression];***

Signalzuweisung

- Eine Signalzuweisung (SIGNAL Assignment) ist ein ‘Concurrent’ - Statement; sie ist das wichtigste ‘Behavioral’ - Statement von VHDL. Signale übertragen Informationen zwischen Prozessen.
- Die Signalzuweisung ändert links nur dann den Wert, wenn rechts ein Ereignis auftritt. Der zugewiesene Wert ist ein Folgeereignis. Im Beispiel mand2.vhdl (Verhaltensbeschreibung) wird eine Architecture mit Concurrent Signal Assignment verwendet. Die Signalverzögerung ist im Beispiel konstant 2 ns (fiktive Annahme).

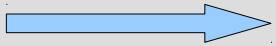
Signalzuweisung (2)

```
entity mand2 is
port
  ( a, b: in bit;
     r:     out bit  );
end mand2;

architecture behav of
mand2 is
begin
   r <= (a and b)  after 2 ns;
end;
```

Testbench

- Um einen Entwurf zu simulieren, muss er mit wohldefinierten Stimuli versorgt werden



- Notwendigkeit einer Testbench (Testumgebung)

- Es werden zu definierten Eingangswerten die erwarteten Ausgangswerte angegeben.

Testbench (2)

```
entity adder_tb is
end adder_tb;
architecture behav of adder_tb is
        component adder
          port (i0, i1 : in bit; ci : in bit; s : out bit; co : out bit);
        end component;
        for adder_0: adder use entity work.adder;
        signal i0, i1, ci, s, co : bit;
     begin
        adder_0: adder port map (i0 => i0, i1 => i1, ci => ci, s =>
s, co => co);
       -- siehe nächste Folie
end behav;
```

Testbench (3)

```
    process
        type pattern_type is record
           i0, i1, ci : bit;
           s, co : bit;
        end record;
        type pattern_array is array (natural range <>) of
pattern_type;
        constant patterns : pattern_array :=
          (('0', '0', '0', '0', '0'),
           ('0', '0', '1', '1', '0'),
           ('0', '1', '0', '1', '0'),
           ('0', '1', '1', '0', '1'),
           ('1', '0', '0', '1', '0'),
           ('1', '0', '1', '0', '1'),
           ('1', '1', '0', '0', '1'),
           ('1', '1', '1', '1', '1'));
```

Testbench (4)

```
  begin
      for i in patterns'range loop
         i0 <= patterns(i).i0;
         i1 <= patterns(i).i1;
         ci <= patterns(i).ci;
         wait for 1 ns;
         assert s = patterns(i).s
            report "bad sum value" severity error;
         assert co = patterns(i).co
            report "bad carray out value" severity error;
      end loop;
      assert false report "end of test" severity note;
      wait;
end process;
```

GHDL

- OpenSource – Implementierung von VHDL
- ghdl.free.fr
- Besonderheit gegenüber anderen Systemen:
  - Sourcekode muss compiliert werden
  - ghdl -a sourcefilename.vhdl
  - ghdl -e entityname
  - ghdl -r entityname
- Signalviewer:
  - GTKwave (siehe auch ghdl.free.fr)
  - ghdl -r entityname --vcd=output[.vcd]