



Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

MIN-Fakultät
Fachbereich Informatik



– VHDL-Einführung / HDL-Übersicht –

<https://tams.informatik.uni-hamburg.de>

Andreas Mäder



Universität Hamburg
Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik
Technische Aspekte Multimodaler Systeme

Oktober 2021



1. VHDL

- Konzepte
- sequenzieller Code
- konkurrenter Code
- Simulation
- VHDL Einheiten
- struktureller Code
- Entwurfsmethodik

2. Hardwarebeschreibungssprachen



VHDL

VHSIC Hardware Description Language
Very High Speed Integrated Circuit

- ▶ digitale Systeme
 - ▶ Modellierung/Beschreibung
 - ▶ Simulation
 - ▶ Dokumentation
- ▶ Komponenten
 - ▶ Standard ICs
 - ▶ anwendungsspezifische Schaltungen: ASICs, FPGAs
 - ▶ Systemumgebung: Protokolle, Software ...



► Abstraktion

- ▶ von der Spezifikation
- ▶ über die Implementation
- ▶ bis hin zum fertigen Entwurf
- ⇒ VHDL durchgängig einsetzbar
- ⇒ Simulation immer möglich
- Algorithmen und Protokolle
- Register-Transfer Modelle
- Netzliste mit Backannotation

Entwicklung

- ▶ 1983 vom DoD initiiert
- ▶ 1987 IEEE Standard IEEE 1076
- ▶ 2004 IEC Standard IEC 61691-1-1
- ▶ regelmäßige Überarbeitungen
VHDL'93, VHDL'02, VHDL'08, VHDL'19



Erweiterungen

- ▶ Modellierung und Zellbibliotheken IEC 61691-2, IEC 61691-5
- ▶ Hardwaresynthese IEC 61691-3-3, IEC 62050
- ▶ mathematische Typen und Funktionen IEC 61691-3-2
- ▶ analoge Modelle und Simulation IEC 61691-6

Links

- ▶ <https://tams.informatik.uni-hamburg.de/research/vlsi/vhdl>
- ▶ <https://www.vhdl-online.de>
- ▶ <https://www.itiv.kit.edu/english/721.php>
- ▶ <https://www.asic-world.com/vhdl>
- ▶ <https://en.wikipedia.org/wiki/VHDL>
- ▶ https://de.wikipedia.org/wiki/Very_High_Speed_Integrated_Circuit_Hardware_Description_Language



► Typen, Untertypen, Alias-Deklarationen

- > skalar integer, real, character, boolean, bit,
 Aufzählung
- > komplex line, string, bit_vector, Array, Record
- > Datei text, File
- > Zeiger Access
- strikte Typbindung
- Konvertierungsfunktionen

► Objekte constant, variable, file

► Operatoren

- 1 logisch and, or, nand, nor, xor, xnor
- 2 relational =, /=, <, <=, >, >=
- 3 schiebend sll, srl, sla, sra, rol, ror
- 4 additiv +, -, &
- 5 vorzeichen +, -
- 6 multiplikativ *, /, mod, rem
- 7 sonstig **, abs, not



► Anweisungen

- > Zuweisung `:=, <=`
- > Bedingung `if, case`
- > Schleifen `for, while, loop, exit, next`
- > Zusicherung `assert, report`
- > ...

► Sequenzielle Umgebungen

- > Prozesse `process`
- > Unterprogramme `procedure, function`
- lokale Gültigkeitsbereiche
- Deklarationsteil: definiert Typen, Objekte, Unterprogramme
Anweisungsteil: Codeanweisungen sequenziell ausführen

⇒ Imperative sequenzielle Programmiersprache (z.B. Pascal)

- Beliebige Programme *ohne Bezug zum Hardwareentwurf* möglich
- Beispiel: Datei einlesen, verlinkte Liste erzeugen ...



VHDL – sequenziell (cont.)

```
...
type      LIST_T;
type      LIST_PTR      is access LIST_T;
type      LIST_T        is record KEY    : integer;
                           LINK    : LIST_PTR;
                        end record LIST_T;

constant  INPUT_ID       : string    := "inFile.dat";
file      DATA_FILE      : text;
variable  DATA_LINE      : line;
variable  LIST_P, TEMP_P : LIST_PTR := null;

procedure READ_DATA is      -- Datei einlesen, Liste aufbauen
  variable KEY_VAL      : integer;
  variable FLAG          : boolean;
begin
  file_open (DATA_FILE, INPUT_ID, read_mode);
  L1: while not endfile(DATA_FILE) loop
    readline(DATA_FILE, DATA_LINE);
    L2: loop
      read(DATA_LINE, KEY_VAL, FLAG);
      if FLAG  then  TEMP_P := new LIST_T'(KEY_VAL, LIST_P);
                      LIST_P := TEMP_P;
                  else   next L1;
                  end if;
    end loop L2;
  end loop L1;
  file_close(DATA_FILE);
end procedure READ_DATA;
...
```



- ▶ ähnlich ADA'83
- ▶ Konkurrenter Code
 - > mehrere Prozesse
 - > Prozeduraufufe
 - > Signalzuweisung <=
 - bedingt <= ... when ...
 - selektiv with ... select ... <= ...
 - > Zusicherung assert
 - ▶ modelliert gleichzeitige Aktivität der Hardwareelemente
- ▶ Synchronisationsmechanismus für Programmlauf / Simulation
 - > Objekt signal
 - ▶ Signale verbinden konkurrent arbeitende „Teile“ miteinander
 - ▶ Entsprechung in Hardware: Leitung



- ▶ Semantik der Simulation im Standard definiert: **Simulationszyklus**
- ▶ konkurrent aktive Codefragmente
 - ▶ Prozesse + konkurrente Anweisungen + Instanzen (in Hierarchien)
 - ▶ durch Signale untereinander verbunden

„Wie werden die Codeteile durch einen sequenziellen Simulationsalgorithmus abgearbeitet?“

- ▶ Signaltreiber: Liste aus Wert-Zeit Paaren

s: integer	NOW	+5 ns	+12 ns	+15 ns	+21 ns	+27 ns	Zeitpunkt Wert
	2	7	3	12	8	-3	

- ▶ Simulationsereignis
 - ▶ Werteänderung eines Signals
 - ▶ (Re-) Aktivierung eines Prozesses nach Wartezeit

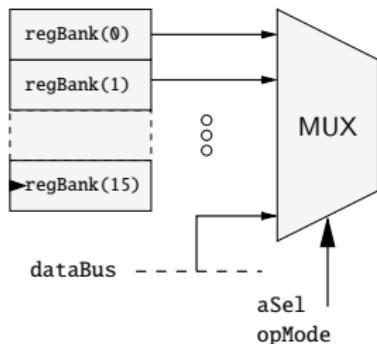
⇒ Ereignisgesteuerte Simulation

- ▶ theoretisches Modell
- ▶ veranschaulicht Semantik für den VHDL-Benutzer
- ▶ praktische Implementation durch Simulationsprogramme weicht (aus Performanzgründen) in der Regel davon ab
- ▶ vielfältige Optimierungsmöglichkeiten
 - ▶ compilierende Simulation
 - ▶ Ausnutzen von Datenabhängigkeiten
 - ▶ zyklusbasierte Simulation
 - ▶ Hardwareemulation
 - ▶ mixed-mode Simulation
 - ▶ ...

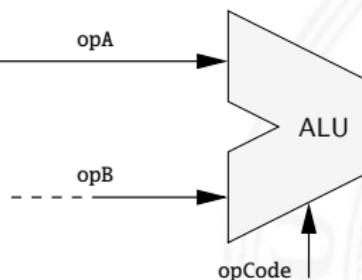
Ereignisgesteuerte Simulation

Beispiel: Datenpfad

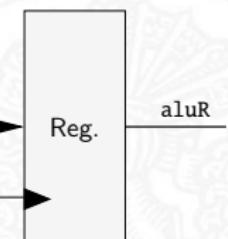
```
opA <= regBank(aSel)
      when opMode=regM else
      dataBus;
```



```
with opCode select
alu <= opA + opB when opcAdd,
opA - opB when opcSub,
opA and opB when opcAnd,
...
```



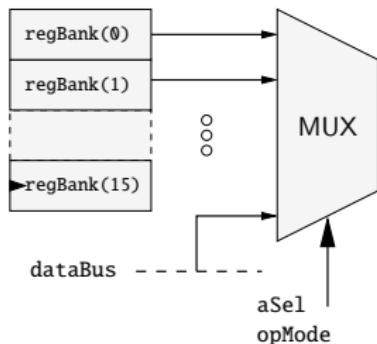
```
regP: process (clk) is
begin
  if rising_edge(clk) then
    aluR <= alu;
  end if;
end process regP;
```



Ereignisgesteuerte Simulation – Zyklus 1

1. Simulationsereignis

```
opA <= regBank(aSel)
      when opMode=regM else
      dataBus;
```



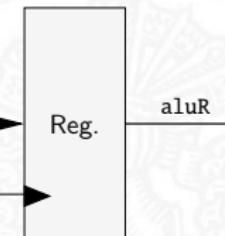
Ereignisliste

NOW	aSel	1
	opMode	regM
	opCode	opcAdd
+10 ns	clk	'1'

...

with opCode select
alu <= opA + opB when opcAdd,
opA - opB when opcSub,
opA and opB when opcAnd,
...

```
regP: process (clk) is
begin
  if rising_edge(clk) then
    aluR <= alu;
  end if;
end process regP;
```



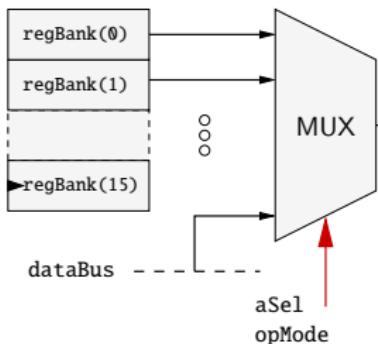
Ereignisgesteuerte Simulation – Zyklus 1

VHDL - Simulation

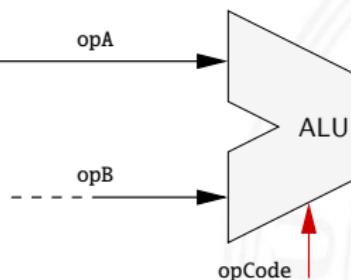
VHDL-Einführung

1. Simulationsereignis
2. Prozessaktivierung

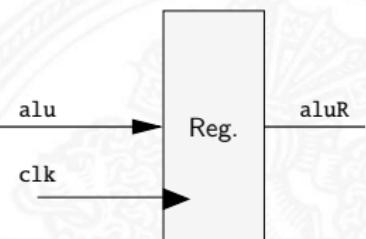
```
opA <= regBank(aSel)
      when opMode=regM else
      dataBus;
```



```
with opCode select
alu <= opA + opB when opcAdd,
opA - opB when opcSub,
opA and opB when opcAnd,
...
```



```
regP: process (clk) is
begin
  if rising_edge(clk) then
    aluR <= alu;
  end if;
end process regP;
```



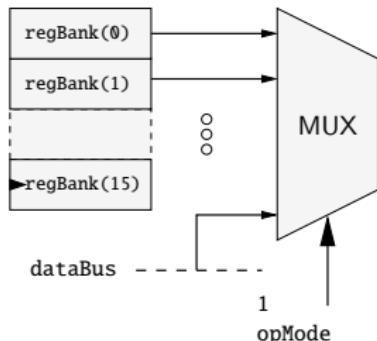
Ereignisgesteuerte Simulation – Zyklus 1

VHDL - Simulation

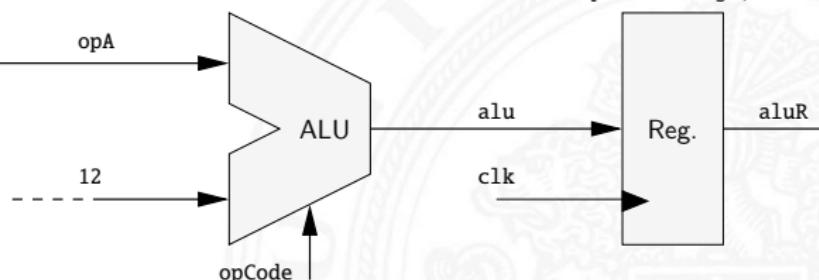
VHDL-Einführung

1. Simulationsereignis
2. Prozessaktivierung
3. Aktualisierung der Signaltreiber

```
opA <= regBank(aSel)
      when opMode=regM else
      dataBus;
```



```
with opCode select
  alu <= opA + opB when opcAdd,
  opA - opB when opcSub,
  opA and opB when opcAnd,
  ...
```



	NOW	$+\delta$
opA	37	16

	NOW	$+\delta$
alu	25	49

```
regP: process (clk) is
begin
  if rising_edge(clk) then
    aluR <= alu;
  end if;
end process regP;
```

Ereignisgesteuerte Simulation – Zyklus 2

VHDL - Simulation

VHDL-Einführung

Zeitschritt: $+\delta$

Simulationereignisse

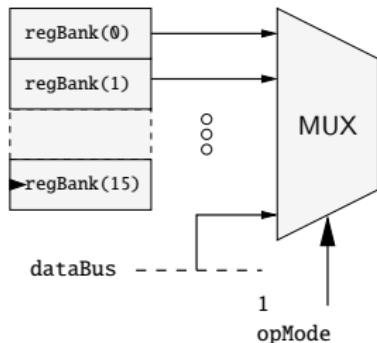
$+\delta$	opA	16
	alu	49
$+10\text{ ns}$	clk	'1'

alu

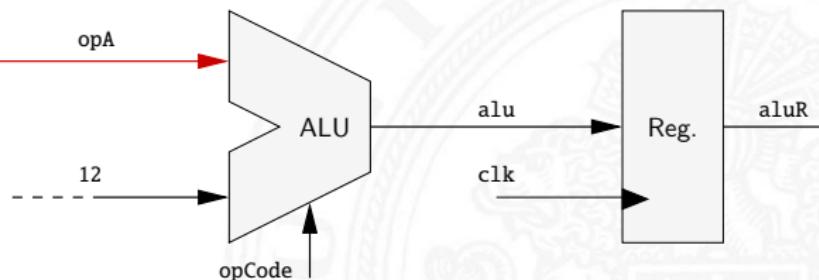
Signaltreiber

NOW	$+\delta$
49	28

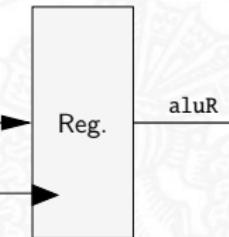
```
opA <= regBank(aSel)
      when opMode=regM else
      dataBus;
```



```
with opCode select
  alu <= opA + opB when opcAdd,
  opA - opB when opcSub,
  opA and opB when opcAnd,
  ...
```



```
regP: process (clk) is
begin
  if rising_edge(clk) then
    aluR <= alu;
  end if;
end process regP;
```



Ereignisgesteuerte Simulation – Zyklus 3

VHDL - Simulation

VHDL-Einführung

Zeitschritt: +10 ns

Simulationereignisse

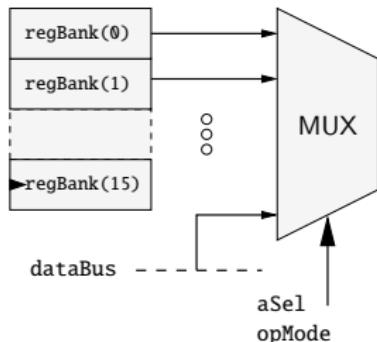
+10 ns clk '1'

...

Signaltreiber

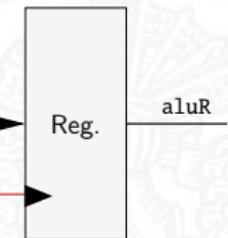
NOW	+ δ
25	28

```
opA <= regBank(aSel)
      when opMode=regM else
      dataBus;
```



```
with opCode select
alu <= opA + opB when opcAdd,
opA - opB when opcSub,
opA and opB when opcAnd,
...
```

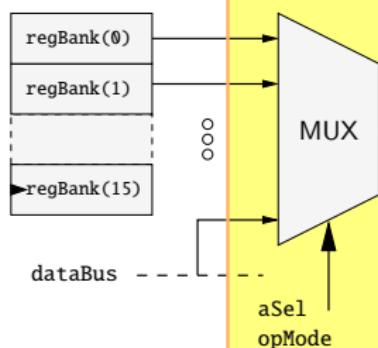
```
regP: process (clk) is
begin
    if rising_edge(clk) then
        aluR <= alu;
    end if;
end process regP;
```



Zyklenbasierte Simulation

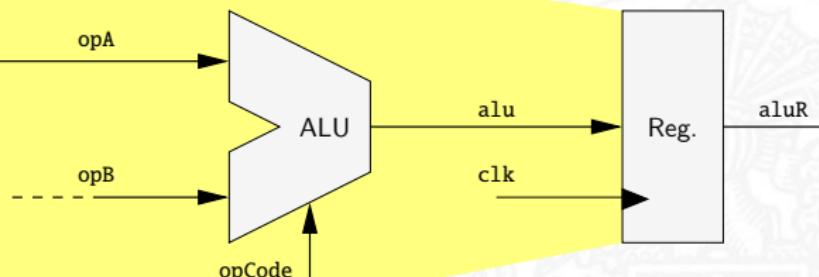
- Diskretes Zeitraster: Takt bei Register-Transfer Code
- In jedem Zyklus werden alle Beschreibungen simuliert
- Sequenzialisierung der Berechnung entsprechend den Datenabhängigkeiten

```
opA <= regBank(aSel)
      when opMode=regM else
      dataBus;
```



```
with opCode select
alu <= opA + opB when opcAdd,
opA - opB when opcSub,
opA and opB when opcAnd,
```

...



Semantik der Simulation

► Kausalität

1. Simulationereignis Simulationszyklus_n
2. Aktivierung des konkurrenten Codes
3. Signalzuweisungen in der Abarbeitung
-
4. Erneute Signaländerung Simulationszyklus_{n+1}

► Trennung der Zyklen

- ⇒ Simulation ist *unabhängig von der sequenziellen Abarbeitungsreihenfolge* durch den Simulator
- ⇒ auch bei *mehreren Events* in einem Zyklus,
bzw. bei *mehrfachen Codeaktivierungen* pro Event



Prozesse / Umgebungen von sequenziellem Code

- ▶ ständig aktiv ⇒ Endlosschleife

1. Sensitiv zu Signalen

- ▶ Aktivierung, bei Ereignis eines Signals
- ▶ Abarbeitung aller Anweisungen bis zum Prozessende

```
ALU_P: process (A, B, ADD_SUB) is
begin
    if ADD_SUB  then  X <= A + B;
                      else  X <= A - B;
    end if;
end process ALU_P;
```

2. explizite wait-Anweisungen

- ▶ Warten bis Bedingung erfüllt ist
- ▶ Abarbeitung aller Anweisungen bis zum nächsten wait
- ▶ Prozessende wird „umlaufen“ (Ende einer Schleife)

VHDL – Simulationszyklus (cont.)

Beispiel: Erzeuger / Verbraucher

```
...
signal C_READY, P_READY : boolean      -- Semaphore
signal CHANNEL           : ...          -- Kanal

PRODUCER_P: process is                  -- Erzeuger
begin
    P_READY    <= false;
    wait until C_READY;
    CHANNEL    <= ...
    P_READY    <= true;
    wait until not C_READY;
end process PRODUCER_P;

CONSUMER_P: process is                  -- Verbraucher
begin
    C_READY    <= true;
    wait until P_READY;
    C_READY    <= false;
    ...        <= CHANNEL;            -- verarbeitet Werte
    wait until not P_READY;
end process CONSUMER_P;
```



VHDL – Simulationszyklus (cont.)

Signalzuweisungen im sequenziellen Kontext

- ▶ sequenzieller Code wird nach der Aktivierung bis zum Prozessende / zum wait abgearbeitet
- ▶ Signalzuweisungen werden erst in folgenden Simulationszyklen wirksam, frühestens im nächsten δ -Zyklus
- ⇒ eigene Zuweisungen sind im sequenziellen Kontext des Prozesses nicht sichtbar

```
process ...
...
if SWAP = '1' then    -- Werte tauschen
  B <= A;           -- B = 'altes' A
  A <= B;           -- A = 'altes' B
end if;
```

```
process ...
...
NUM <= 5;           -- Zuweisung
...
if NUM > 0 then    -- ggf. /= 5 !!!
  ...
end if;
```



- ▶ Strukturbeschreibungen / Hierarchie

- > Instanzen component configuration
- > Schnittstellen entity
- > Versionen und Alternativen (*exploring the design-space*)
 architecture configuration

- ▶ Management von Entwürfen

- > Bibliotheken library
- > Code-Reuse package
- ▶ VHDL-Erweiterungen: Datentypen, Funktionen ...
- ▶ Gatterbibliotheken
- ▶ spezifisch für EDA-Tools (**Electronic Design Automation**)
- ▶ eigene Erweiterungen, firmeninterne Standards ...



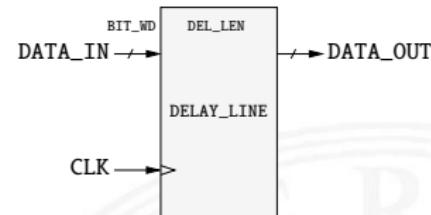
VHDL – Entity

- ▶ Beschreibung der Schnittstelle „black-box“
- > mit Parametern **generic**
- > mit Ein- / Ausgängen **port**

parametrierbare Verzögerungsleitung

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity DELAY_LINE is
generic(BIT_WD      : integer range 2 to 64 := 16;
        DEL_LEN     : integer range 2 to 16 := 16);
port ( CLK         : in  std_logic;
        DATA_IN     : in  signed(BIT_WD-1 downto 0);
        DATA_OUT    : out signed(BIT_WD-1 downto 0));
end entity DELAY_LINE;
```



- ▶ Implementation einer Entity
- ▶ mehrere Architekturen möglich ⇒ Alternativen
- ▶ Deklarationsteil: Typen, Signale, Unterprogramme,
Komponenten, Konfigurationen
Anweisungsteil: Prozesse, konkurrente Anweisungen, Instanzen

```
architecture BEHAVIOR of DELAY_LINE is
  type    DEL_ARRAY_TY is array (1 to DEL_LEN) of signed(BIT_WD-1 downto 0);
  signal   DEL_ARRAY      : DEL_ARRAY_TY;
begin
  DATA_OUT      <= DEL_ARRAY(DEL_LEN);
  REG_P: process (CLK) is
  begin
    if rising_edge(CLK) then
      DEL_ARRAY <= DATA_IN & DEL_ARRAY(1 to DEL_LEN-1);
    end if;
  end process REG_P;
end architecture BEHAVIOR;
```

- ▶ Hierarchie
 - ▶ funktionale Gliederung des Entwurfs
 - ▶ repräsentiert Abstraktion
 - ▶ Instanziierung von Komponenten
 1. Komponentendeklaration **component**
 - ▶ im lokalen Kontext
 - ▶ in externen Packages
 2. Instanz im Code verwenden
 - ▶ im Anweisungsteil der Architecture
 - ▶ Mapping von Ein- und Ausgängen / Generic-Parametern
 3. Bindung: Komponente \Leftrightarrow Entity+Architecture
 - ▶ lokal im Deklarationsteil **for ... use ... configuration**
 - ▶ als eigene VHDL-Einheit

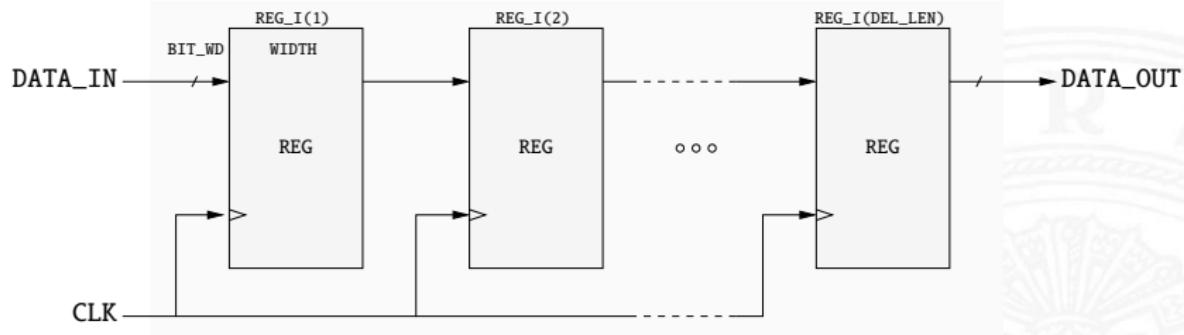
VHDL – strukturell (cont.)

- ▶ Komponente: lokale Zwischenstufe im Bindungsprozess
 - ▶ andere Bezeichner, Schnittstellen (Ports und Generics)
 - ▶ bei Bibliothekselementen wichtig
 - ▶ 2-stufige Abbildung
 1. Instanz in Architektur \Leftrightarrow Komponente
 2. Komponente \Leftrightarrow Entity+Architecture
 - ▶ „Default“-Konfiguration
 - ▶ gleiche Bezeichner und Deklaration
 - ▶ zuletzt (zeitlich) analysierte Architektur
- ▶ strukturierende Anweisungen
 - ▶ Gruppierung **block**
 - ▶ konkurrenten Code (Prozesse, Anweisungen, Instanzen ...)
 - ▶ bedingt ausführen **if ... generate ...**
 - ▶ wiederholt ausführen **for ... generate ...**

VHDL – strukturell (cont.)

Beispiel als Strukturbeschreibung von Registern: REG

- Die Register sind als eigene VHDL-Entities / -Architekturen woanders definiert



VHDL – strukturell (cont.)

```
architecture STRUCTURE of DELAY_LINE is
  component REG is
    generic( WIDTH      : integer range 2 to 64); -- 1. Komponentendeklaration
    port   ( CLK        : in std_logic;
             D_IN       : in signed(WIDTH-1 downto 0);
             D_OUT      : out signed(WIDTH-1 downto 0));
  end component REG;
  type    DEL_REG_TY is array (0 to DEL_LEN) of signed(BIT_WD-1 downto 0);
  signal  DEL_REG     : DEL_REG_TY;
begin
  DEL_REG(0)    <= DATA_IN;
  DATA_OUT      <= DEL_REG(DEL_LEN);
  GEN_I: for I in 1 to DEL_LEN generate
    REG_I: REG generic map (WIDTH => BIT_WD)      -- 2. Instanziierung
           port map (CLK, DEL_REG(I-1), DEL_REG(I));
  end generate GEN_I;
end architecture STRUCTURE;
```

Konfigurationen

1. Auswahl der Architektur durch eindeutigen Bezeichner

```
configuration DL_BEHAVIOR of DELAY_LINE is
  for BEHAVIOR
  end for;
end configuration DL_BEHAVIOR;
```

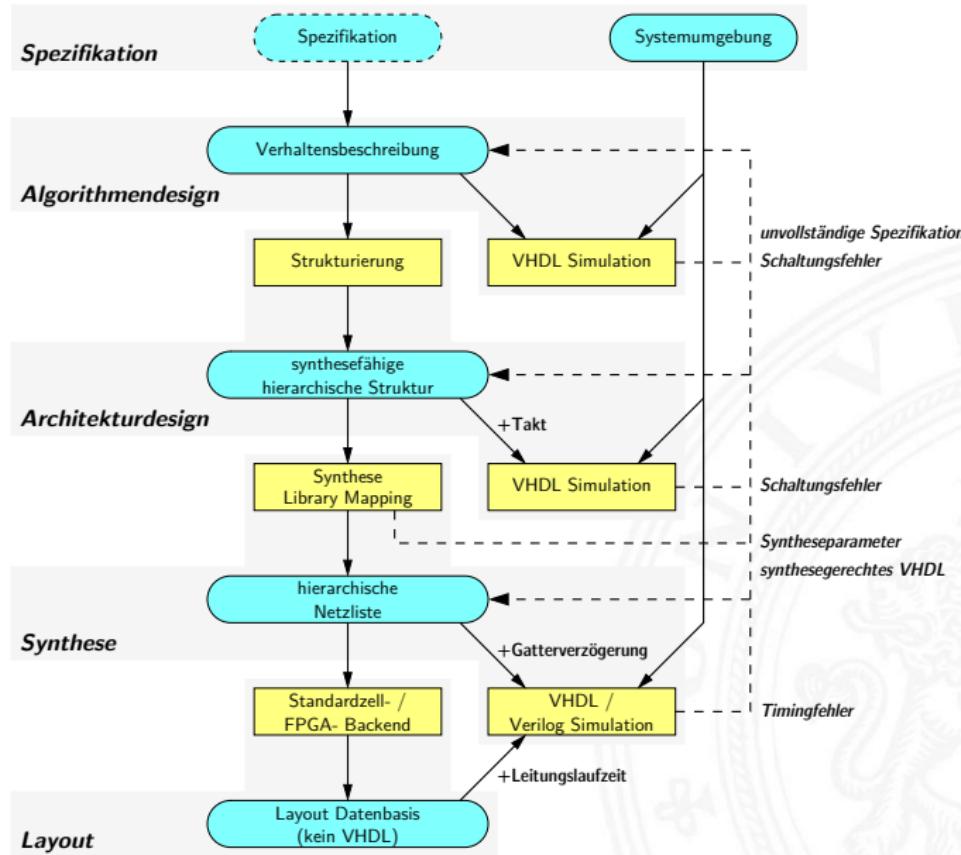
2. Bindung von Instanzen in der Hierarchie

```
configuration DL_STRUCTURE of DELAY_LINE is
  for STRUCTURE
    for GEN_I
      for all: REG use entity work.REG(BEHAVIOR);
      end for;
    end for;
  end for;
end configuration DL_STRUCTURE;
```



- ▶ VHDL Abstraktion: von Algorithmen- bis zur Gatterebene
 - ⇒ eine Sprache als Ein- und Ausgabe der Synthese
- ▶ Synthese: ab der RT-Ebene
 - ▶ Abbildung von Register-Transfer Beschreibungen auf Gatternetzlisten
 - ▶ erzeugt neue Architektur, Entity bleibt
- ▶ High-Level Synthese
 - ▶ eingeschränkter „Suchraum“, feste Zielarchitektur
 - ▶ spezielle Anwendungsfälle
- ▶ Simulation
 - ▶ System-/Testumgebung als VHDL-Verhaltensmodell
 - ▶ Simulation der Netzliste durch Austausch der Architektur

VHDL-basierter Entwurfsablauf



- ## ► Modellierung der Systemumgebung

- ▶ Simulation auf allen Abstraktionsebenen

Algorithmenebene Auswahl verschiedener Algorithmen
keine Zeitmodelle

RT-Ebene Datenabhängigkeiten: Ein-/Ausgabe (Protokolle)
Taktbasierte Simulation

Gatterebene + Gatterverzögerungen

Layout + Leitungslaufzeiten

- #### ► Synthese ab der Register-Transfer Ebene



1. VHDL
2. Hardwarebeschreibungssprachen

VHDL-AMS

Verilog / SystemVerilog

SystemC

Beispiele



VHDL-AMS – Analog and Mixed Signal

- ▶ Analoge VHDL Erweiterung IEC 61691-6
- ▶ VHDL Obermenge: digitale Systeme werden wie in „normalem“ VHDL modelliert
- ▶ „mixed-signal“: gemischte analog-digitale Systeme
„multi-domain“: elektrische + nicht-elektrische Systeme
- ▶ analoge Modellierung: Differentialgleichungssysteme deren freie Variablen durch einen Lösungsalgorithmus („analog Solver“) bestimmt werden
- ▶ analoge Erweiterung der Simulationssemantik
 - ▶ Anpassung der Paradigmen (diskret \leftrightarrow kontinuierlich) für das Zeit- und Wertemodell
 - ▶ Wann wird der Solver aufgerufen?



Anwendungen

- ▶ analoge Systeme
- ▶ mikromechanische Systeme
- ▶ mechanische Komponenten
- ▶ Modellierung der Schnittstellen zu mechanischen Komponenten
- ▶ Beispiel: Ansteuerung eines Motors, Simulation von
 - ▶ analogem Treiber
 - ▶ elektromagnetischem Verhalten des Motors
 - ▶ Massenträgheit und Last
 - ▶ ...



Erweiterungen

► Datentypen

- > **nature**: zur Modellierung verschiedener Domänen

```
subtype VOLTAGE is real tolerance "TOL_VOLTAGE";
subtype CURRENT is real tolerance "TOL_CURRENT";

nature ELECTRICAL is      VOLTAGE across
                        CURRENT through
                        GROUND reference;
```

► Objekte/Ports

- > **terminal**: Referenzpunkt

Knoten eines elektrischen Netzes, Ort im mechanischen System

- > **quantity**: Werte die (zeitkontinuierlich) berechnet werden
Variablen des Gleichungssystems



► Anweisungen

- > simultane Anweisung: Beschreibung einer Gleichung, partiell definierte Systeme (if, case-Fälle) sind möglich
- > break: steuert die Zusammenstellung der Gleichungssysteme des „analog Solvers“ beispielsweise bei Diskontinuitäten
- > procedural: analoges Äquivalent zum process



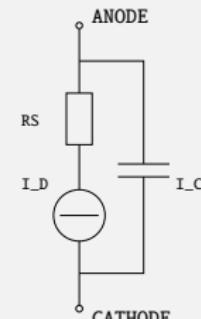
► Diode: charakteristische Gleichungen

$$\begin{aligned} i_d &= i_s \cdot (e^{(v_d - rs \cdot i_d) / n \cdot vt} - 1) \\ i_c &= \frac{d}{dt} (tt \cdot i_d - 2 \cdot c_j \cdot \sqrt{vj^2 - vj \cdot v_d}) \end{aligned}$$

```
library ieee, AMS_LIB;
use ieee.math_real.all;
use AMS_LIB.ELEC_ENV.all;

entity DIODE is
generic (
    ISS           : real := 1.0e-14;
    N            : real := 1.0;
    TT, CJ0, VJ, RS   : real := 0.0);
port
    ( terminal ANODE, CATHODE : ELECTRICAL);
end entity DIODE;

architecture LEVEL0 of DIODE is
    quantity V_D across I_D, I_C through ANODE to CATHODE;
    quantity Q_C : CHARGE;
    constant VT : real := 0.0258;
begin
    I_D == ISS * (exp((V_D-RS*I_D)/(N*VT)) - 1.0);
    Q_C == (TT * I_D) - (2.0 * CJ0 * sqrt(VJ**2 - VJ*V_D));
    I_C == Q_C'dot;
end architecture LEVEL0;
```



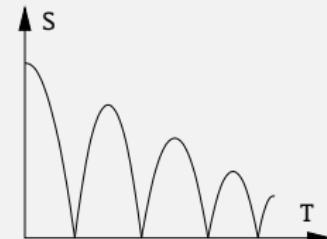
VHDL-AMS – Beispiele (cont.)

► hüpfender Ball

```
library ieee, AMS_LIB;
use ieee.math_real.all;
use AMS_LIB.MECH_ENV.all;

entity BOUNCER is
generic ( S_INI      : DISPLACEMENT := 10.0;
           V_INI      : VELOCITY      := 0.0);
end entity BOUNCER;

architecture BALL of BOUNCER is
  quantity S      : DISPLACEMENT := S_INI;
  quantity V      : VELOCITY     := V_INI;
  constant G      : REAL         := 9.81;
  constant AIR_RES : REAL        := 0.1;
begin
  break V => -V      when not S'above(0.0);
  S'dot == V;
  if V > 0.0  use  V'dot == -G - V**2 * AIR_RES;
    else  V'dot == -G + V**2 * AIR_RES;
  end use;
end architecture BALL;
```





- ▶ Hardwarebeschreibungssprache
 - ▶ Verhaltensbeschreibung (auf Gatter- und RT-Ebene)
SystemVerilog auch auf höheren Ebenen
 - ▶ Strukturbeschreibung, Hierarchien
- ▶ Entwicklung
 - ▶ 1985 ursprünglich proprietäre Sprache / Simulator
 - ▶ 1995 IEEE/IEC Standard, regelmäßige Überarbeitungen IEEE 1364
 - ▶ 2005 aktuelle Erweiterung: SystemVerilog IEEE 1800

Vorteile

- ▶ sehr kompakter Code, wird gerne als Netzlistenformat genutzt
 - häufig geäußerte Kritik an VHDL: „deklarativer Overhead“ –
- ▶ Simulation
 - ▶ sehr schnell: älter als VHDL → Algorithmen besser optimiert
 - ▶ „golden Simulation“: finale Simulation(en) der Netzliste mit extrahierten Leitungslaufzeiten vor der Fertigung
 - ⇒ inzwischen einheitliche Simulatorengines



(System)Verilog oder VHDL

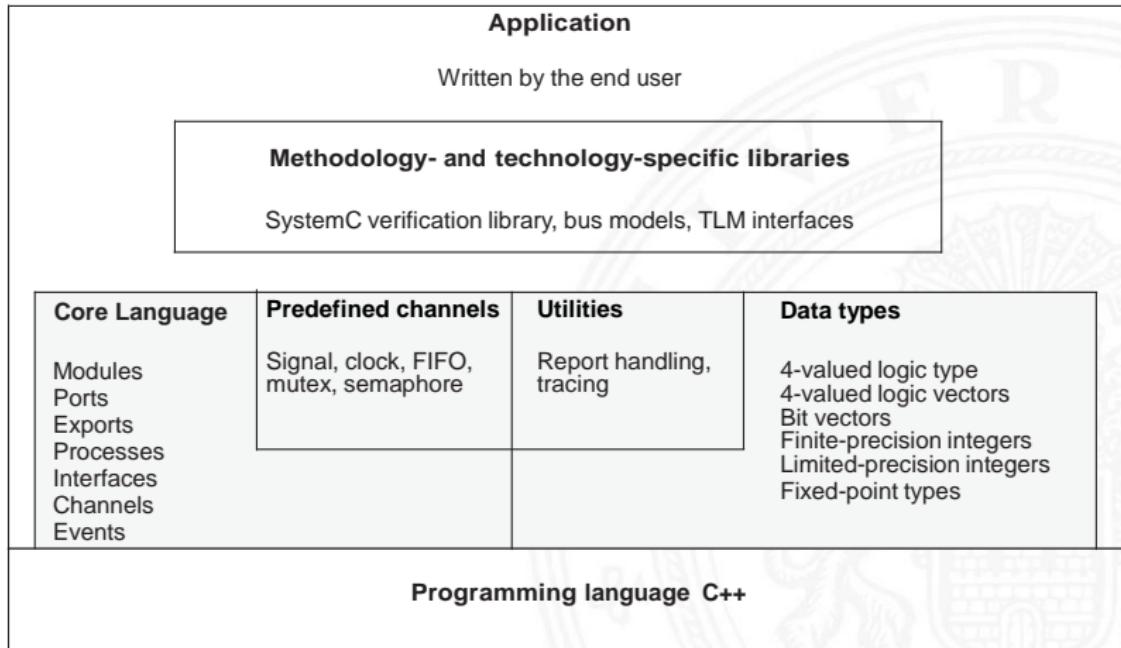
- ▶ kein Unterschied bei den Modellierungsmöglichkeiten
- ▶ oft werden Komponenten beider HDLs gemeinsam eingesetzt
- ▶ EDA-Werkzeuge: Synthesis, Simulation
 - ▶ ein internes Datenformat
 - ▶ unterschiedliche „front-Ends“

Links

- ▶ <https://accellera.org>
- ▶ <https://www.eda-twiki.org>
- ▶ <https://www.asic-world.com/systemverilog>
- ▶ <https://www.asic-world.com/verilog>
- ▶ <https://electrosofts.com/systemverilog>
- ▶ <https://electrosofts.com/verilog>
- ▶ <https://en.wikipedia.org/wiki/SystemVerilog>
- ▶ <https://en.wikipedia.org/wiki/Verilog>



- ▶ C++ basiert: Klassenbibliotheken mit
 - ▶ hardware-nahen Datentypen
 - ▶ Simulatorkern
- ⇒ *Executable ist der Simulator*





- ▶ Semantische Erweiterungen für den Hardwareentwurf
 - ▶ Konkurrente Modelle: Prozesse
 - ▶ Zeitbegriff: Clocks
 - ▶ Reaktivität (Ereignisse in VHDL): Waiting, Watching ...
 - ▶ Kommunikationsmechanismen: Signale, Protokolle
 - ▶ Hierarchie: Module, Ports ...
 - ▶ ...
- ▶ Entwicklung
 - ▶ 2000 OSCI – **Open SystemC Initiative**
 - ▶ 2005 IEEE/IEC Standard
 - ▶ Schrittweise Entwicklung

IEEE 1666

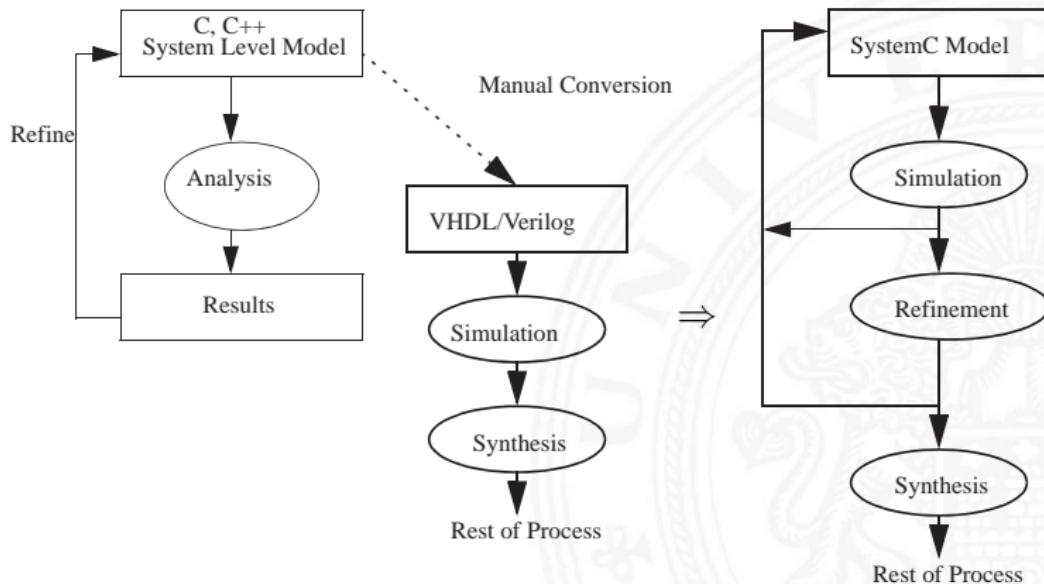
Links

- ▶ <https://accellera.org>
- ▶ <https://www.asic-world.com/systemc>
- ▶ <https://en.wikipedia.org/wiki/SystemC>



Idee/Vorteile

- ▶ Hard- und Software gemeinsam beschreiben:
Hardware-Software Co-Design
- ⇒ durchgängiger, Werkzeug-unterstützter Entwurfsablauf





- ⇒ höhere Abstraktionsebenen
- + C/C++ Infrastruktur nutzen: Compiler, Debugger ...
- + Know-How nutzen:
jeder Softwareentwerfer kann damit auch „Hardware machen“
- + Einfache Integration von eigenem Code und Erweiterungen

Praxis

- + Ersatz für (proprietäre) High-Level Simulation
- macht den C++ Programmierer nicht zum Hardwaredesigner
- noch mehr Deklarationsoverhead als bei VHDL
- Unterstützung durch EDA-Werkzeuge

Kunst des Hardwareentwurfs

Guten, synthesefähigen Code zu erstellen

... gilt für alle HDLs

Beispiele

D-Flipflop mit asynchronem Reset

► VHDL: dff.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity DFF is
port ( CLOCK      : in  std_logic;
       RESET      : in  std_logic;
       DIN        : in  std_logic;
       DOUT       : out std_logic);
end entity DFF;

architecture BEHAV of DFF is
begin
  DFF_P: process (RESET, CLOCK) is
  begin
    if RESET = '1' then
      DOUT <= '0';
    elsif rising_edge(CLOCK) then
      DOUT <= DIN;
    end if;
  end process DFF_P;
end architecture BEHAV;
```

Beispiele (cont.)

D-Flipflop mit asynchronem Reset

► Verilog: dff.v

```
module dff (clock, reset, din, dout);
  input clock, reset, din;
  output dout;

  reg dout;

  always @(posedge clock or reset)
  begin
    if (reset)
      dout = 1'b0;
    else
      dout = din;
  end
endmodule
```



Beispiele (cont.)

D-Flipflop mit asynchronem Reset

► SystemC: dff.h

```
#include "systemc.h"

SC_MODULE(dff)
{
    sc_in<bool>    clock;
    sc_in<bool>    reset;
    sc_in<bool>    din;
    sc_out<bool>   dout;

    void do_ff()
    { if (reset)
        dout = false;
      else if (clock.event())
        dout = din;
    };

    SC_CTOR(dff)
    {
        SC_METHOD(do_ff);
        sensitive(reset);
        sensitive_pos(clock);
    }
};
```

Beispiele

8-bit Zähler, synchron rücksetz- und ladbar

► VHDL: counter.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity COUNTER is
port ( CLOCK      : in  std_logic;
       LOAD       : in  std_logic;
       CLEAR      : in  std_logic;
       DIN        : in  unsigned (7 downto 0);
       DOUT       : out unsigned (7 downto 0));
end entity COUNTER;
```

Beispiele (cont.)

8-bit Zähler, synchron rücksetz- und ladbar

```
architecture BEHAV of COUNTER is
    signal CNTVAL : unsigned (7 downto 0);
begin
    CNT_P: process (CLOCK) is
    begin
        if rising_edge(CLOCK) then
            if CLEAR = '1' then
                CNTVAL <= (others=>'0');
            elsif load = '1' then
                CNTVAL <= DIN;
            else
                CNTVAL <= CNTVAL + 1;
            end if;
        end if;
    end process CNT_P;

    DOUT <= CNTVAL;
end architecture BEHAV;
```

Beispiele (cont.)

8-bit Zähler, synchron rücksetz- und ladbar

► Verilog: counter.v

```
module counter(clock, load, clear, din, dout);
    input          clock, load, clear;
    input [0:7]     din;
    output [0:7]    dout;

    wire [0:7]      dout;
    reg  [0:7]      cntval;

    assign dout = cntval;

    always @(posedge clock)
    begin
        if (clear)
            cntval = 0;
        else if (load)
            cntval = din;
        else
            cntval = cntval + 1;
    end
endmodule
```



Beispiele (cont.)

8-bit Zähler, synchron rücksetz- und ladbar

► SystemC: zwei Dateien

counter.h

```
#include "systemc.h"

SC_MODULE(counter)
{ sc_in<bool>          clock;
  sc_in<bool>          load;
  sc_in<bool>          clear;
  sc_in<sc_uint<8>>    din;
  sc_out<sc_uint<8>>   dout;

  int cntval;
  void onetwothree();

  SC_CTOR(counter)
  { SC_METHOD(onetwothree);
    sensitive_pos (clock);
  }
};
```

Beispiele (cont.)

8-bit Zähler, synchron rücksetz- und ladbar

counter.cc

```
#include "counter.h"

void counter::onetwothree()
{ if (clear)
    cntval = 0;
  else if (load)
    cntval = din.read();           // read for type conversion from input port
  else
    cntval++;
}
dout = cntval;
}
```



- [AL08] Peter J. Ashenden, Jim Lewis:
VHDL-2008: just the new stuff.
Morgan Kaufmann Publishers Inc.;
San Mateo, CA, 2008.
ISBN 978-0-12-374249-0
- [APT02] Peter J. Ashenden, Gregory D. Peterson,
Darrell A. Teegarden:
The System Designer's Guide to VHDL-AMS.
Morgan Kaufmann Publishers Inc.;
San Mateo, CA, 2002.
ISBN 1-55860-749-8



[Ash07] Peter J. Ashenden:

Digital Design – An Embedded Systems Approach using VHDL.

Morgan Kaufmann Publishers Inc.;

San Mateo, CA, 2007.

ISBN 978–0–12–369528–4

[Ash08] Peter J. Ashenden:

The Designer's Guide to VHDL.

3rd ed.; Morgan Kaufmann Publishers Inc.;

San Mateo, CA, 2008.

ISBN 978–0–12–088785–9



Literaturliste (cont.)

- [Bha99] Jayaram Bhasker:
A VHDL primer.
3rd ed.; Prentice-Hall, Inc.;
Englewood Cliffs, NJ, 1999.
ISBN 0-13-096575-8
- [Bha02] Jayaram Bhasker:
A SystemC primer.
Star Galaxy Publishing; Allentown, PA, 2002.
ISBN 0-9650391-8-8
- [G⁺02] Thorsten Grötker [u. a.]:
System Design with SystemC.
Kluwer Academic Publishers; Boston, MA, 2002.
ISBN 1-4020-7072-1



Literaturliste (cont.)

[H⁺00] Ulrich Heinkel [u. a.]:

The VHDL Reference – A Practical Guide to Computer-Aided Integrated Circuit Design including VHDL-AMS.

John Wiley & Sons; New York, NY, 2000.
ISBN 0–471–89972–0

[Jas16] Ricardo Jasinski:

Effective Coding with VHDL – Principles and Best Practice.

The MIT Press; Cambridge, MA, 2016.
ISBN 978–0262034227



Literaturliste (cont.)

- [MR13] Paul Molitor, Jörg Ritter:
Kompaktkurs VHDL.
Oldenbourg; München, 2013.
ISBN 978-3-486-71292-6
- [PT97] David Pellerin, Douglas Taylor:
VHDL Made Easy!
Prentice-Hall, Inc.; Englewood Cliffs, NJ, 1997.
ISBN 0-13-650763-8
- [RS20] Jürgen Reichardt, Bernd Schwarz:
VHDL-Simulation und -Synthese
Achte; De Gruyter Oldenbourg; Berlin, 2020.
ISBN 978-3-11-067345-6



[IEEE 1076] *Standard 1076-2019;*

IEEE Standard VHDL Language Reference Manual.

Institute of Electrical and Electronics Engineers, Inc.;

New York, NY, 2019.

ISBN 978-1-5044-6135-1

[IEC 61691-1-1] *IEC 61691-1-1-2011;*

IEEE Behavioural languages - Part 1-1:

VHDL Language Reference Manual.

International Electrotechnical Commission; Genf, 2011.

ISBN 978-0-7381-6605-6

IEEE / IEC Standards (cont.)

Literaturliste

VHDL-Einführung

[IEEE 1076.1] *Standard 1076.1-2017; IEEE Standard VHDL Analog and Mixed-Signal Extensions.*

Institute of Electrical and Electronics Engineers, Inc.;
New York, NY, 2017.
ISBN 978-1-5044-4267-1

[IEC 61691-6] *IEC 61691-6 Ed. 2.0 2021-06;
IEEE 1076.1 – Behavioural languages - Part 6:
VHDL Analog and Mixed-Signal Extensions.*

International Electrotechnical Commission; Genf, 2021.
ISBN 978-2-8322-9830-5



IEEE / IEC Standards (cont.)

Literaturliste

VHDL-Einführung

[IEEE 1076.2] *Standard 1076.2-1996;*

IEEE Standard VHDL Mathematical Packages.

Institute of Electrical and Electronics Engineers, Inc.;

New York, NY, 1996.

ISBN 0-7381-0988-6

[IEC 61691-3-2] *IEC 61691-3-2 First edition 2001-06; Behavioural*

languages - Part 3-2: Mathematical Operation in VHDL.

International Electrotechnical Commission; Genf, 2001.

ISBN 0-580-39086-1



IEEE / IEC Standards (cont.)

Literaturliste

VHDL-Einführung

[IEEE 1076.3] *Standard 1076.3-1997;*

IEEE Standard VHDL Synthesis Packages.

Institute of Electrical and Electronics Engineers, Inc.;

New York, NY, 1997.

ISBN 1-5593-7923-5

[IEC 61691-3-3] *IEC 61691-3-3 First edition 2001-06; Behavioural*

languages - Part 3-3: Synthesis in VHDL.

International Electrotechnical Commission; Genf, 2001.

ISBN 0-580-39087-X



[IEEE 1076.4] *Standard 1076.4-2000; IEEE Standard VITAL ASIC (Application Specific Integrated Circuit) Modeling Specification 2001.*

Institute of Electrical and Electronics Engineers, Inc.;
New York, NY, 2001.
ISBN 0-7381-2691-0

[IEC 61691-5] *IEC 61691-5 First edition 2004-10; IEEE 1076.4 – Behavioural languages - Part 5: VITAL ASIC (Application Specific Integrated Circuit) Modeling Specification.*

International Electrotechnical Commission; Genf, 2004.
ISBN 2-8318-7684-2



IEEE / IEC Standards (cont.)

Literaturliste

VHDL-Einführung

[IEEE 1076.6] *Standard 1076.6-1999; IEEE Standard for VHDL Register Transfer Level (RTL) Synthesis.*

Institute of Electrical and Electronics Engineers, Inc.;
New York, NY, 1999.
ISBN 0-7381-1819-2

[IEC 62050] *IEC 62050 First edition 2005-07;
IEEE 1076.6 – IEEE Standard for VHDL
Register Transfer Level (RTL) Synthesis.*

International Electrotechnical Commission; Genf, 2004.
ISBN 0-7381-4065-1

IEEE / IEC Standards (cont.)

[IEEE 1164] *Standard 1164-1993; IEEE Standard Multivalue Logic System for VHDL Model Interoperability.*

Institute of Electrical and Electronics Engineers, Inc.;
New York, NY, 1993.
ISBN 1-55937-299-0 – withdrawn

[IEC 61691-2] *IEC 61691-2 First edition 2001-06;
Behavioural languages - Part 2: VHDL Multilogic
System for Model Interoperability.*

International Electrotechnical Commission; Genf, 2001.
ISBN 0-580-39266-X



IEEE / IEC Standards (cont.)

Literaturliste

VHDL-Einführung

[IEEE 1364] *Standard 1364-2005; IEEE Standard for Verilog Hardware Description Language.*

Institute of Electrical and Electronics Engineers, Inc.;
New York, NY, 2006.
ISBN 0-7381-8450-4

[IEC 61691-4] *IEC 61691-4 First edition 2004-10;
IEEE 1364 – Behavioural languages - Part 4:
Verilog Hardware Description Language.*

International Electrotechnical Commission; Genf, 2004.
ISBN 2-8318-7675-3



IEEE / IEC Standards (cont.)

Literaturliste

VHDL-Einführung

[IEEE 1364.1] *Standard 1364.1-2002; IEEE Standard for Verilog Register Transfer Level Synthesis.*

Institute of Electrical and Electronics Engineers, Inc.;
New York, NY, 2002.
ISBN 0-7381-3501-1

[IEC 62142] *IEC 62142 First edition 2005-06; IEEE 1364.1 – Verilog Register Transfer Level Synthesis.*

International Electrotechnical Commission; Genf, 2005.
ISBN 2-8318-8036-X

IEEE / IEC Standards (cont.)

[IEEE 1800] *IEEE 1800-2017;*

*Standard for SystemVerilog - Unified Hardware Design,
Specification and Verification Language.*

Institute of Electrical and Electronics Engineers, Inc.;

New York, NY, 2018.

ISBN 978-1-5044-4509-2

[IEC 62530] *IEC 62530 Edition 2.0 2011-05; IEEE 1800 –*

SystemVerilog - Unified Hardware Design,

Specification and Verification Language.

International Electrotechnical Commission; Genf, 2011.

ISBN 978–2–88912–450–3



[IEEE 1666] *Standard 1666-2011; IEEE Standard for Standard SystemC Language Reference Manual.*

Institute of Electrical and Electronics Engineers, Inc.;
New York, NY, 2012.
ISBN 978-0-7381-6801-2

[IEC 61691-7] *IEC 61691-7 Edition 1.0 2009-12;
IEEE 1666 – Behavioural languages - Part 7:
SystemC Language Reference Manual.*

International Electrotechnical Commission; Genf, 2009.
ISBN 978-0-7381-6284-3

[IEEE 1666.1] *Standard 1666.1-2016; IEEE Standard for Standard SystemC Analog/Mixed-Signal Extensions Language Reference Manual.*

Institute of Electrical and Electronics Engineers, Inc.;
New York, NY, 2016.
ISBN 978-1-5044-0746-5

[IEC 61691-8] *IEC 61691-8 Ed. 1.0 2021-07;
IEEE 1666.1 – Behavioural languages - Part 8:
Standard SystemC Analog/Mixed-Signal Extensions Language Reference Manual.*

International Electrotechnical Commission; Genf, 2021.
ISBN 978-2-8322-9951-7