

64-210 Eingebettete Systeme

-VHDL-Einführung / HDL-Übersicht-

<http://tams.informatik.uni-hamburg.de/lectures/2013ss/vorlesung/es>

Andreas Mäder



Universität Hamburg
Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik

Technische Aspekte Multimodaler Systeme

Sommersemester 2013



Gliederung

1. VHDL

Konzepte

sequenzieller Code

konkurrenenter Code

Simulation

VHDL Einheiten

struktureller Code

Entwurfsmethodik

2. Hardwarebeschreibungssprachen

VHDL

VHDL

VHSIC **H**ardware **D**escription **L**anguage
Very **H**igh **S**peed **I**ntegrated **C**ircuit

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



VHDL (cont.)

▶ 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 08
- ▶ 2004 IEC Standard IEC 61691-1-1 11
- ▶ regelmäßige Überarbeitungen: VHDL'93, VHDL'02, VHDL'08, VHDL'11



VHDL (cont.)

Erweiterungen

- ▶ Hardwaremodellierung/Zellbibliotheken
IEC 61691-2 01, IEC 61691-5 04
- ▶ Hardwaresynthese
IEC 61691-3-3 01, IEC 62050 04
- ▶ mathematische Typen und Funktionen
IEC 61691-3-2 01
- ▶ analoge Modelle und Simulation
IEC 61691-6 09



VHDL (cont.)

Links

- ▶ <http://tams.informatik.uni-hamburg.de/research/vlsi/vhdl>
- ▶ <http://www.vhdl.org>
- ▶ <http://www.vhdl-online.de>
- ▶ <http://en.wikipedia.org/wiki/VHDL>
- ▶ http://de.wikipedia.org/wiki/Very_High_Speed_Integrated_Circuit_Hardware_Description_Language
- ▶ <http://www.asic-world.com/vhdl>



VHDL – sequenziell

- ▶ 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

VHDL – sequenziell (cont.)

▶ 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 |
| > | ... | |

VHDL – sequenziell (cont.)

▶ 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...

```

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

VHDL – sequenziell (cont.)

```

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;
  ...
    
```

VHDL – konkurrent

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



VHDL – Simulation

- ▶ 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?“

VHDL – Simulation (cont.)

- ▶ Signaltreiber: Liste aus Wert-Zeit Paaren

S: integer ←

NOW	+5 ns	+12 ns	+15 ns	+21 ns	+27 ns
2	7	3	12	8	-3

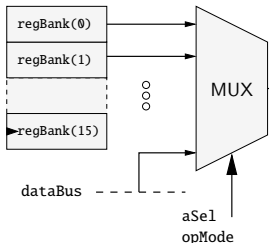
Zeitpunkt
Wert

- ▶ 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 in der Regel davon ab
 - ▶ Optimierungsmöglichkeiten:
Datenabhängigkeiten, zyklenbasierte Simulation ...

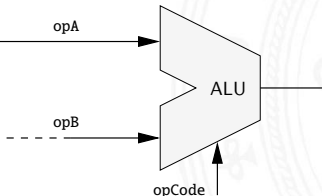
Ereignisgesteuerte Simulation

Beispiel: Datenpfad

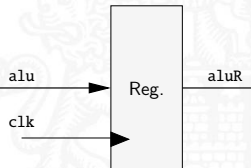
```
opA <= regBank(aSel)
      when opMode=regM else
      dataBus;
```



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



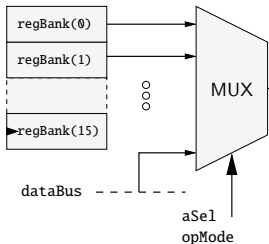
```
regP: process (clk) is
begin
  if rising_edge(clk) then
    aluR <= alu;
  end if;
end process regP;
```



Ereignisgesteuerte Simulation – Zyklus 1

1. Simulationsereignis

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



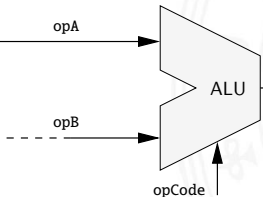
Ereignisliste

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

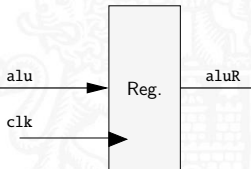
```

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

```



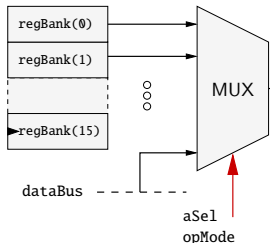
```
regP: process (clk) is
begin
  if rising_edge(clk) then
    aluR <= alu;
  end if;
end process regP;
```



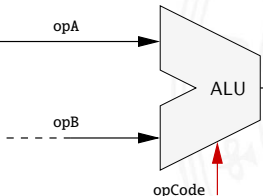
Ereignisgesteuerte Simulation – Zyklus 1

1. Simulationsereignis
2. Prozessaktivierung

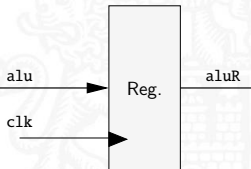
```
opA <= regBank(aSel)
      when opMode=regM else
      dataBus;
```



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



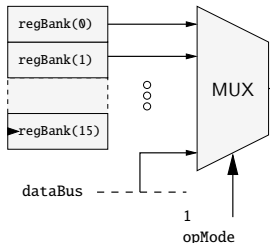
```
regP: process (clk) is
begin
  if rising_edge(clk) then
    aluR <= alu;
  end if;
end process regP;
```



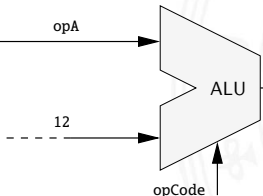
Ereignisgesteuerte Simulation – Zyklus 1

1. Simulationsereignis
2. Prozessaktivierung
3. Aktualisierung der Signaltreiber

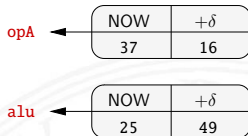
```
opA <= regBank(aSel)
      when opMode=regM else
      dataBus;
```



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



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



Ereignisgesteuerte Simulation – Zyklus 2

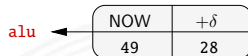
Zeitschritt: $+\delta$

Simulationsereignisse

```

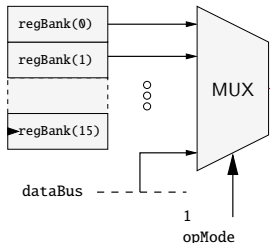
+δ      opA  16
        alu  49
+10 ns  clk  '1'
...
    
```

Signaltreiber



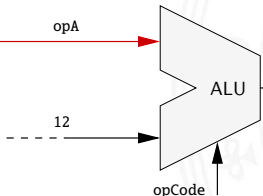
```

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



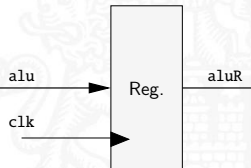
```

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



```

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



Ereignisgesteuerte Simulation – Zyklus 3

Zeitschritt: +10 ns

Simulationsereignisse

+10 ns clk '1'

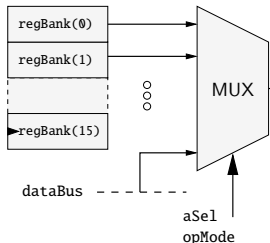
...

Signaltreiber

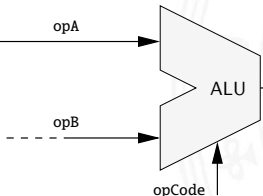
NOW	+ δ
25	28

aluR ←

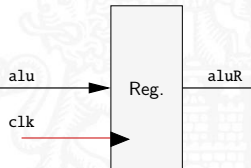
```
opA <= regBank(aSel)
      when opMode=regM else
      dataBus;
```



```
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;
```



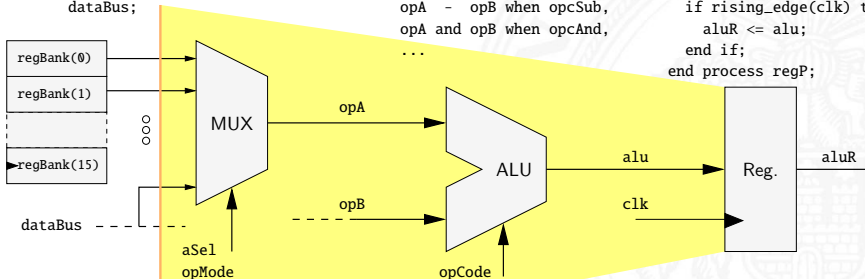
Zyklusbasierte Simulation

- ▶ 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,
      ...
```

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





VHDL – Simulation

Semantik der Simulation

► Kausalität

1. Simulationsereignis
2. Aktivierung des konkurrenten Codes
3. Signalzuweisungen in der Abarbeitung

Simulationszyklus_{*n*}

-
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

VHDL – Simulation (cont.)

Prozesse / Umgebungen von sequenziellem Code

- ▶ ständig aktiv \Rightarrow 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 – Simulation (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 <= ...                               -- generiert Werte
    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 – Simulation (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 !!!
...
    
```



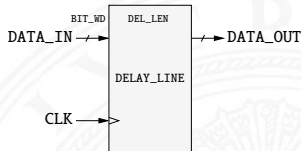

Entwurfsspezifische Eigenschaften

- ▶ 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 (**E**lectronic **D**esign **A**utomation)
 - ▶ 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;
    
```

VHDL – Architecture

- ▶ Implementation einer Entity
- ▶ mehrere Architekturen möglich \Rightarrow Alternativen
- ▶ Deklarationsteil: Typen, Signale, Unterprogramme, Komponenten, Konfigurationen
- Anweisungsteil: Prozesse, konkurrenente 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;
    
```

VHDL – strukturell

▶ Hierarchie

- ▶ funktionale Gliederung des Entwurfs
- ▶ repräsentiert Abstraktion

▶ Instanziierung von Komponenten

1. Komponentendeklaration

- ▶ im lokalen Kontext
- ▶ in externen Packages

`component`

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
- ▶ als eigene VHDL-Einheit

`for ... use ...
configuration`



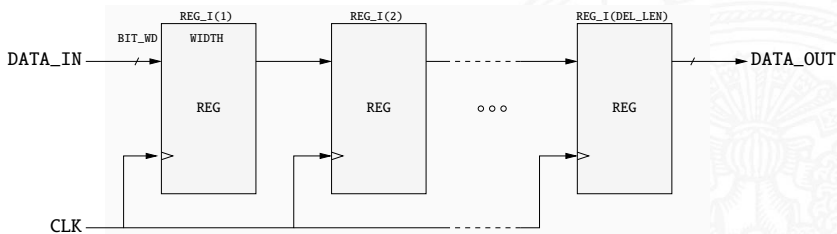
VHDL – strukturell (cont.)

- ▶ 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                                -- 1. Komponentendeklaration
  generic( WIDTH           : integer range 2 to 64);
  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;
    
```



VHDL – strukturell (cont.)

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;
    
```



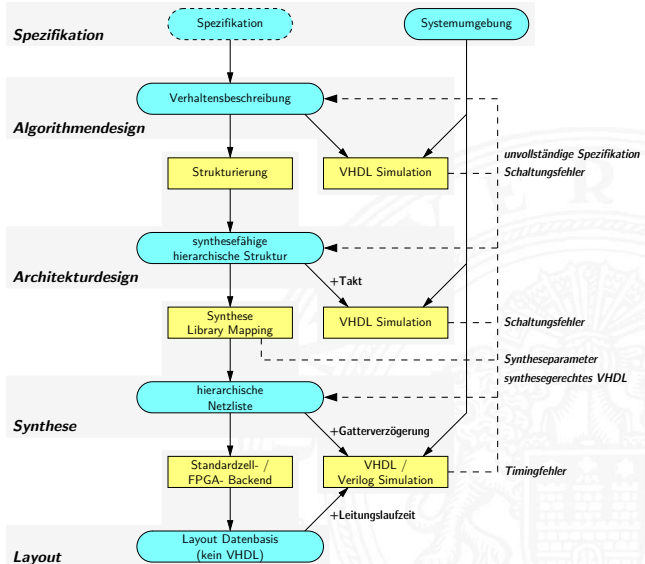

VHDL – Entwurfsmethodik

- ▶ 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





VHDL – Entwurfsmethodik (cont.)

- ▶ 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



Gliederung

1. VHDL
2. Hardwarebeschreibungssprachen

VHDL-AMS

Verilog / SystemVerilog

SystemC

Beispiele



VHDL-AMS

VHDL-AMS – **A**nalog and **M**ixed **S**ignal

- ▶ Analoge VHDL Erweiterung IEC 61691-6 09
- ▶ 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 ↔ kontinuierlich) für das Zeit- und Wertemodell
 - ▶ Wann wird der Solver aufgerufen?



VHDL-AMS (cont.)

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
 - ▶ ...



VHDL-AMS (cont.)

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

VHDL-AMS (cont.)

► 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

Beispiele

► Diode: charakteristische Gleichungen

$$i_d = i_s \cdot (e^{(v_d - r_s \cdot i_d) / n \cdot v_t} - 1)$$

$$i_c = \frac{d}{dt} (tt \cdot i_d - 2 \cdot c_j \cdot \sqrt{v_j^2 - v_j \cdot v_d})$$

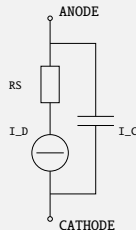
VHDL-AMS (cont.)

```

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, CJO, 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 * CJO * sqrt(VJ**2 - VJ*V_D));
    I_C == Q_C'dot;
end architecture LEVEL0;
    
```



VHDL-AMS (cont.)

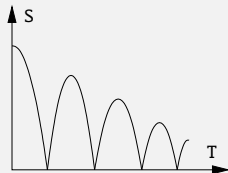
► hüpfender Ball

```

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;
    
```



Verilog / SystemVerilog

- ▶ 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 06
 - ▶ 2005 aktuelle Erweiterung: SystemVerilog IEEE 1800 09



Verilog / SystemVerilog (cont.)

Links

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

Verilog / SystemVerilog (cont.)

Vorteile

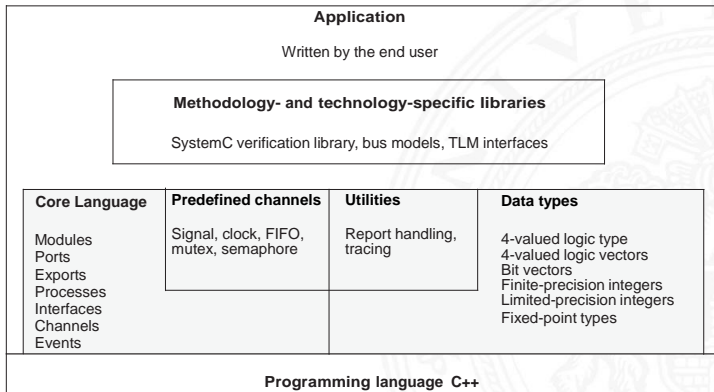
- ▶ sehr kompakter Code, wird gerne als Netzlistenformat genutzt
 - häufig geäußerte Kritik an VHDL: „deklarativer Overhead“ –
- ▶ Simulation
 - ▶ sehr schnell: älter als VHDL → Algorithmen besser optimiert
 - ▶ „golden Simulation“: finale Simulation(en) der Netzliste mit extrahierten Leitungslaufzeiten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

(System)Verilog oder VHDL

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

SystemC

- ▶ C++ basiert: Klassenbibliotheken mit
 - ▶ hardware-nahen Datentypen
 - ▶ Simulatorkern





SystemC (cont.)

- ▶ 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 – **O**pen **S**ystem**C** Initiative
 - ▶ 2005 IEEE/IEC Standard
 - ▶ Schrittweise Entwicklung

IEEE 1666 12



SystemC (cont.)

Links

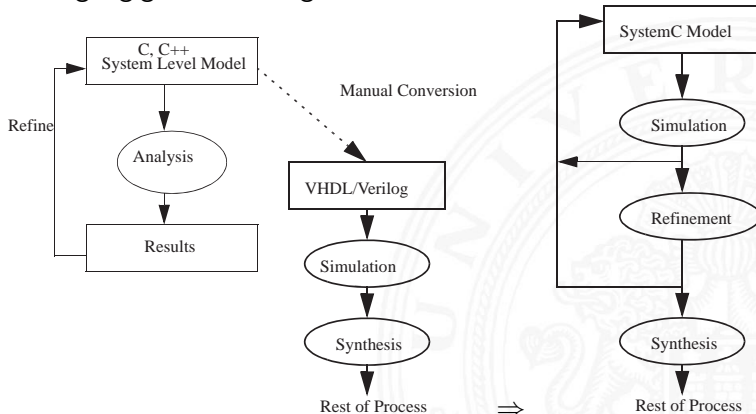
- ▶ <http://www.systemc.org>
- ▶ <http://en.wikipedia.org/wiki/SystemC>
- ▶ <http://www.asic-world.com/systemc>
- ▶ <http://www-ti.informatik.uni-tuebingen.de/~systemc>

Idee/Vorteile

- ▶ Hard- und Software gemeinsam beschreiben:
Hardware-Software Co-Design
- ⇒ höhere Abstraktionsebenen

SystemC (cont.)

⇒ durchgängiger, Werkzeug-unterstützter Entwurfsablauf





SystemC (cont.)

- ▶ C/C++ Infrastruktur nutzen: Compiler, Debugger. . .
- ▶ Know-How nutzen:
jeder Softwareentwickler 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.)

► 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.)

► 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 (cont.)

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.)

```

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.)

► 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++;
        end
    endmodule
    
```




Beispiele (cont.)

► SystemC: counter.h counter.cc

```

#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.)

```

#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;
}
    
```



Literaturliste

- [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



Literaturliste (cont.)

- [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.
Morgan Kaufmann Publishers Inc.;
San Mateo, CA, 2008.
ISBN 978-0-12-088785-9



Literaturliste (cont.)

[Bha99] Jayaram Bhasker:

A VHDL primer.

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



Literaturliste (cont.)

- [G⁺02] Thorsten Grötter [u. a.]:
System Design with SystemC.
Kluwer Academic Publishers; Boston, MA, 2002.
ISBN 1-4020-7072-1
- [PT97] David Pellerin, Douglas Taylor:
VHDL Made Easy!
Prentice-Hall, Inc.; Englewood Cliffs, NJ, 1997.
ISBN 0-13-650763-8



IEEE / IEC Standards

[IEEE 1076 08] *Standard 1076-2008;*

IEEE Standard VHDL Language Reference Manual.

Institute of Electrical and Electronics Engineers, Inc.;

New York, NY, 2009.

ISBN 978-0-7381-5801-3

[IEC 61691-1-1 11] *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.)

[IEEE 1076.1 07] *Standard 1076.1-2007; IEEE Standard VHDL Analog and Mixed-Signal Extensions.*

Institute of Electrical and Electronics Engineers, Inc.;
New York, NY, 2007.
ISBN 0-7381-5627-2

[IEC 61691-6 09] *IEC 61691-6; IEEE 1076.1-2009 – Behavioural languages - Part 6: VHDL Analog and Mixed-Signal Extensions.*

Institute of Electrical and Electronics Engineers, Inc.;
New York, NY, 2009.
ISBN 978-0-7381-6283-6



IEEE / IEC Standards (cont.)

[IEEE 1076.2 96] *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 01] *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.)

[IEEE 1076.3 97] *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 01] *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 / IEC Standards (cont.)

[IEEE 1076.4 01] *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 04] *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.)

[IEEE 1076.6 99] *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 04] *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 93] *Standard 1164-1993; IEEE Standard Multivalued 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 01] *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.)

[IEEE 1364 06] *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 2004] *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.)

[IEEE 1364.1 02] *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 05] *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 09] *IEEE 1800-2009;
Standard for SystemVerilog - Unified Hardware Design,
Specification and Verification Language.*

Institute of Electrical and Electronics Engineers, Inc.;
New York, NY, 2009.

ISBN 978-0-7381-6129-7

[IEC 62530 07] *IEC 62530 Edition 1.0 2007-11; IEEE 1800 –
Standard for SystemVerilog - Unified Hardware Design,
Specification and Verification Language.*

International Electrotechnical Commission; Genf, 2007.

ISBN 2-8318-9349-6



IEEE / IEC Standards (cont.)

[IEEE 1666 12] *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 09] *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