

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 1

### *Literatur*

[PA02] P.J. Ashenden: The Designer's Guide to VHDL, Second Edition; Morgan Kaufmann Publishers, 2002, ISBN: 1-55860-674-2

[RS00] J. Reichardt, B. Schwarz: VHDL-Synthese, Entwurf digitaler Schaltungen und Systeme; Oldenbourg Verlag, 2000, ISBN: 3-486-25128-7

[CS01] C. Siemers: Hardwaremodellierung, Einführung in Simulation und Synthese von Hardware; HANSER Verlag, 2001; ISBN: 3-446-21361-9

[JB98] J. Bhasker: A VHDL Primer, Third Edition; Prentice Hall PTR, 1998, ISBN: 0130965758

[AG93] J.R. Armstrong, F.G. Gray: Structured Logic Design with VHDL; Prentice-Hall, 1993; ISBN: 0-13-885206-1

[AG00] J.R. Armstrong, F.G. Gray: VHDL Design Representation and Synthesis, 2<sup>nd</sup> ed.; Prentice-Hall, 2000; ISBN: 0-13-021670-4; vergriffen!!!

[CS99] C. Siemers: Prozessorbau, Eine konstruktive Einführung in das Hardware / Software-Interface; HANSER Verlag, 1999; ISBN: 3-446-19330-8

[SiDr02] A. Sikora, R. Drechsler: Software-Engineering und Hardware-Design, Eine systematische Einführung; HANSER Verlag, 2002; ISBN: 3-446-21861-0

[Wo01] W. Wolf: Computers as Components, Principles of Embedded Computing System Design; Morgan Kaufmann Publishers, 2001; ISBN: 1-55860-693-9

[BRJ99] G. Booch, J. Rumbaugh, I. Jacobson: Das UML-Benutzerhandbuch, 2. Auflage; Eddison-Wesley, 1999; ISBN: 3-8237-1486-0

[OF03] W. Oberschelp, G. Vossen: Rechneraufbau und Rechnerstrukturen; Oldenbourg Verlag, 2003; ISBN: 3-4862-7206-3

<b>Universität Duisburg - Essen</b>	<b>Dr.-Ing. R. Viga</b>	<b>Fak. f. Ing.-Wiss. EIT, EBS/VS</b>
<b>Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2</b>		<b>SEITE: 2</b>

### *Inhaltsübersicht*

- Grundlagen und Bedienung eines Entwicklungs- und Simulationssystems für VHDL
- VHDL Sprachgrundlagen mit einfachen Modellbeispielen aus der Digitaltechnik
- Verhaltens-orientierte Modelle
- Datenfluss-orientierte Modelle
- Struktur-orientierte Modelle
- Programmierung von Komplexlogiken mit VHDL

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 3

### ***Was ist VHDL?***

- V** - Very high speed integrated circuits
- H** - Hardware
- D** - Description
- L** - Language

### ***Der VHDL-Befehlssatz umfasst:***

- Sequenzielle Befehle
- Nebenläufige (konkurrente) Befehle
- Befehle zur Beschreibung von Netzlisten
- Befehle zur Beschreibung von Zeitabhängigkeiten
- Befehle zur Beschreibung von Signalverläufen (Waveforms)

<b>Universität Duisburg - Essen</b>	<b>Dr.-Ing. R. Viga</b>	<b>Fak. f. Ing.-Wiss. EIT, EBS/VS</b>
<b>Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2</b>		<b>SEITE: 4</b>

## ***Historische Entwicklung von VHDL***

### **1981**

VHSIC Programm des US DoD; verschiedene Chiplieferanten sollten einheitliche Beschreibungen ihrer Produkte vorweisen können

### **1983**

IBM, Texas Instruments und Intermetrics entwickeln zusammen VHDL

### **1985**

erste veröffentlichte Version von VHDL, V. 7.2

### **1986**

Übergabe von VHDL an den IEEE zur Standardisierung

### **12/1987**

IEEE Standard 1076-1987; Language Reference Manual (LRM)

### **1993**

Überarbeitete Version des Standards 1076-1993

Vereinheitlichung der Modellierung von Logiksignalen zur sog. 9-wertigen Logik (Real sind die Logiksignale nicht nur auf 0/1 bzw. false/true beschränkt!) mit dem IEEE STD\_LOGIC\_1164 Packet.

Seit 9/1988 verlangt das US Militär nach ihrer Standard MIL 454, dass alle Zulieferer ihre Produkte in VHDL auf der Verhaltens- und Strukturebene beschreiben und auch die Testsätze dafür, in VHDL beschrieben, beistellen.

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 5

## ***Was ist ein Modell?***

**UML**: unified modeling language [BRJ99] besagt:

*Ein Modell ist eine Vereinfachung der Realität.*

Unsere "Realität" hier sind Modelle von Systemen.

Systeme sind eine Gruppierung von kooperierenden, interagierenden Objekten, die als die Gesamtheit *System* gegenüber ihrer Umwelt ein Verhalten zeigen.

Ein Modell liefert die Entwürfe zu einem System.

Modelle können sowohl detailliertere Pläne umfassen als auch eher allgemeine Pläne, die einen Überblick über das zu betrachtende System gewähren.

Ein gutes Modell betrachtet die Elemente, die weitreichende Auswirkungen haben und übergeht die, die für das gegebene Abstraktionsniveau nicht relevant sind.

Jedes System kann unter verschiedenen Aspekten mit unterschiedlichen Modellen beschrieben werden, jedes Modell ist daher eine semantisch abgeschlossene Abstraktion des Systems.

Modelle können **strukturbezogen** sein, d.h. die Organisation des Systems steht im Vordergrund, oder sich auf das **Verhalten** beziehen, dann ist die Dynamik des Systems von Bedeutung.

*Wir bauen Modelle, um das zu entwickelnde System besser verstehen zu können.*

**UML** ist eine reine Beschreibungssprache, die der Spezifikation und Dokumentation dient. Die mit UML beschriebenen Modelle sind nicht simulierbar.

**VHDL** hat den Charakter einer Programmiersprache, d.h. VHDL liefert auch ausführbaren Code, oder besser formuliert Programmcode als Beschreibungsmodell, das anhand von Tests in Form von Stimulanzmustern (Eingaben) und Reaktionen darauf (Ausgaben) dynamisch ausführbar (simulierbar) ist.

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 6

## **Was ist Hard- und Software?**

In [CS01] finden wir dazu Folgendes:

**Hardware** ist die *physikalische Implementierung einer Funktionalität*.

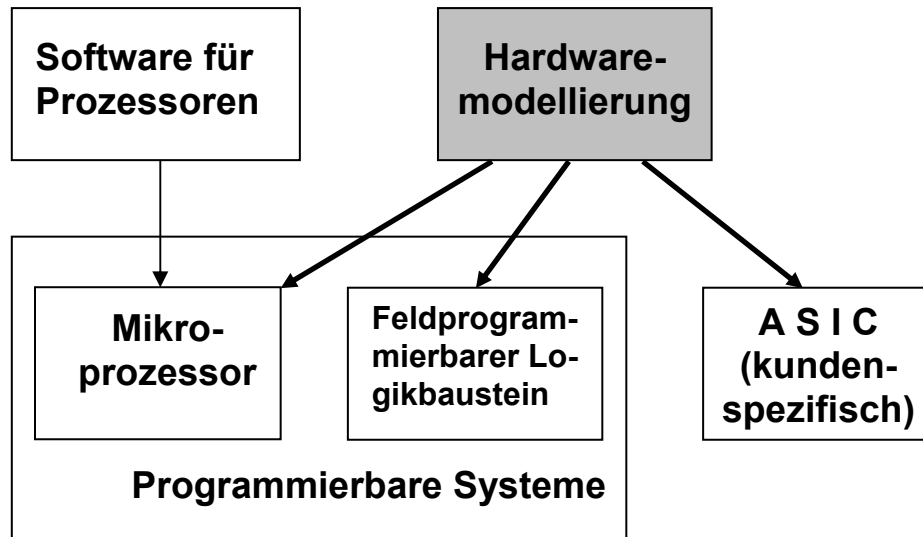
Wir wollen hier darunter mikroelektronische Systeme sehen, die eine gewünschte Funktionalität erfüllen. Weiter einschränkend sollen diese **digitale Schaltungen** sein, die wir in diesem Kurs beschreiben (modellieren) und simulieren wollen.

Die Funktionalität der Hardware kann weiter darauf beschränkt werden, dass sie einen **Befehlsumfang** implementiert, also einen sog. **Prozessor** oder in der technischen Informatik auch mit **Maschine** bezeichnet.

Zusammen mit einem **Speicher** und darin befindlicher **Programme als Abfolge solcher Befehle** kommt man dann zum Begriff der Software.

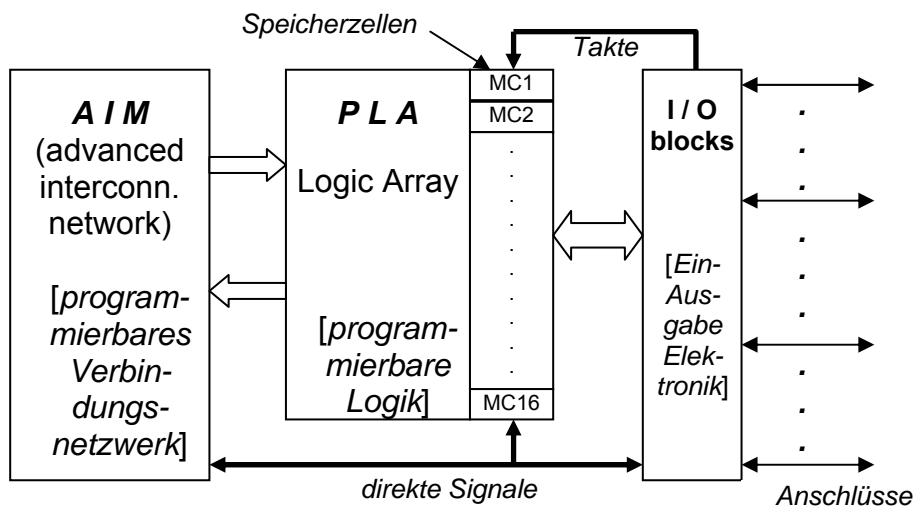
*Unter **Software** versteht man die logische Implementierung einer Funktionalität zur Ausführung durch ein mikroelektronisches System (Hardware, Prozessor). Im engeren Sinne ist darunter die Folge von Befehlen gefasst, die von dieser Hardware zur Verfügung gestellt und in einem strukturierten Ablauf gebracht wird.*

### Überblick Digitale Systeme

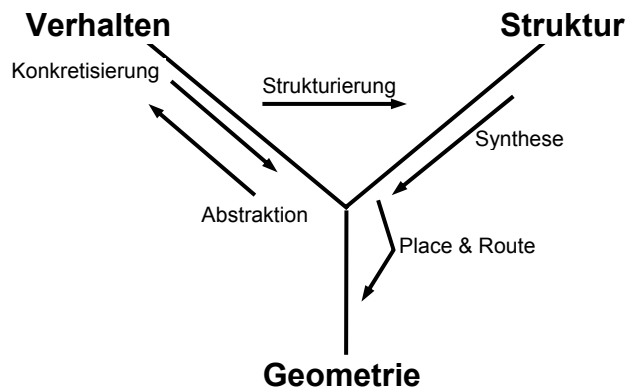
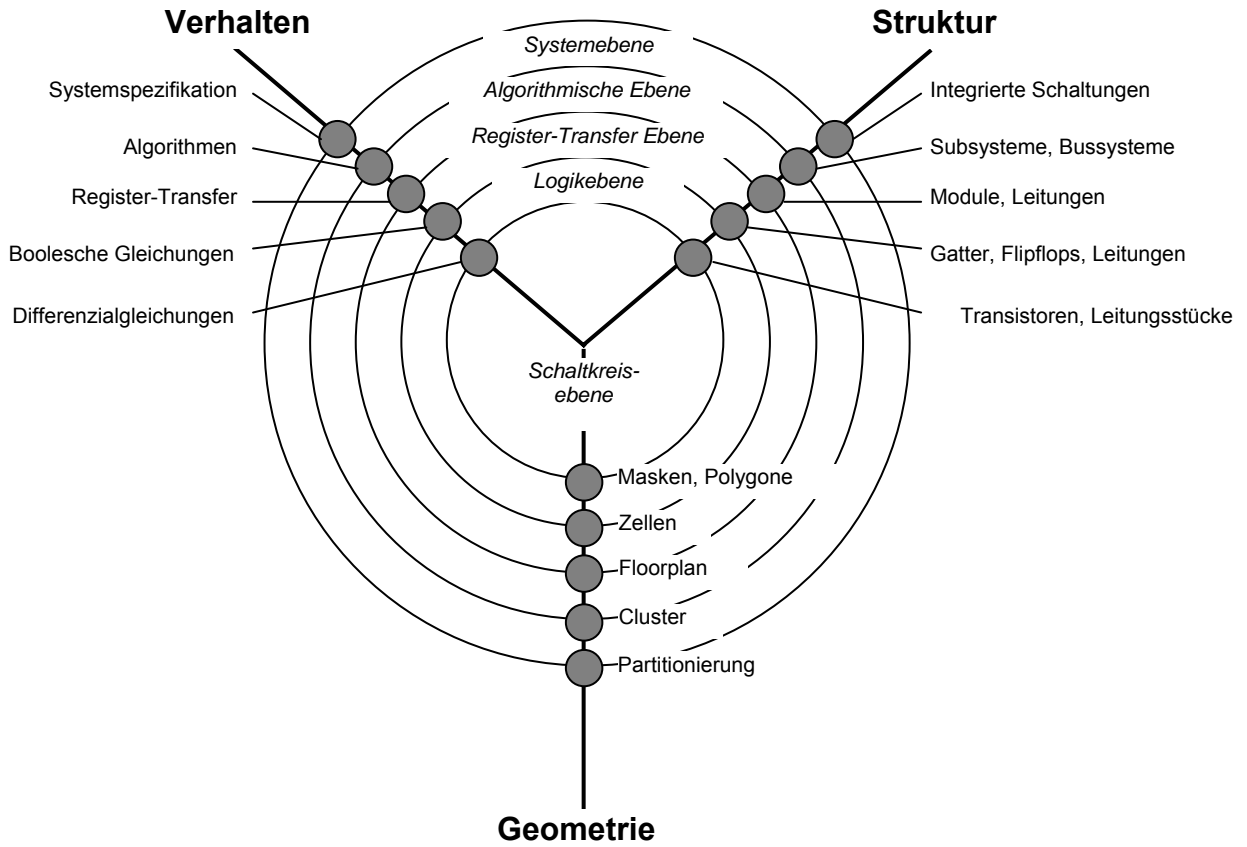


### Prinzipschaltbild eines CPLD Bausteins

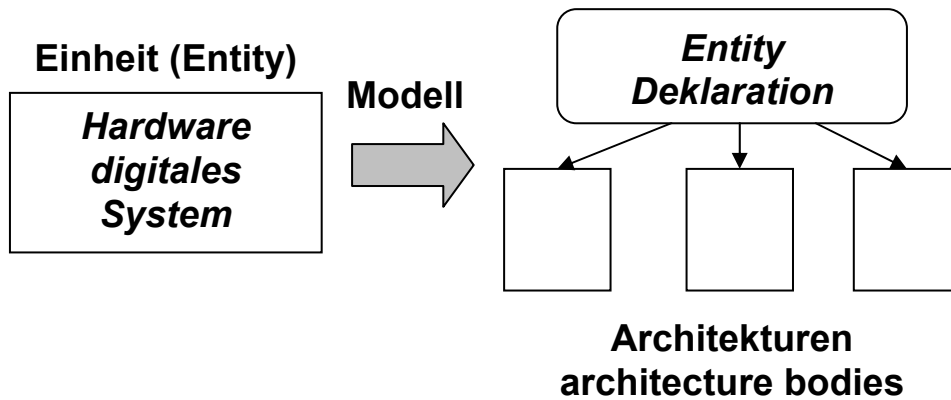
(complex programmable logic device)



**Y-Diagramm nach Gajski; aus [CS01]**



## Aufbau von VHDL-Beschreibungen



### Allgemeiner Aufbau der Entity Deklaration

```
entity entity_name is
[ generic (par_1 {, par_k}: type_name [:= def_val]
           {; further_gen_decl} ); ]
[ port ( { port_il {, port_ik}: IN type_name [:= def_val]; }
         { port_ol {, port_ok}: OUT type_name [:= def_val]; }
         { port_iol {, port_iok}: INOUT type_name [:= def_val]; }
         ); ]
end [entity] [entity_name];
```

### Allgemeiner Aufbau einer Architektur

```
architecture arch_name of entity_name is
-- lokaler Deklarationsteil:
-- USE-Anweisungen
-- Typen, Untertypen,
-- Konstante, Variable, Signale
-- Files, Komponenten,
-- Unterprogramme, Attribute, Konfigurationen
-- Definitionen von Unterprogrammen, Attributen,
-- Shared Variables (global)
begin
-- nebenläufige Anweisungen zur Modellierung
-- (strukturelle ~, Verhaltensbeschreibung)
end [architecture] [arch_name];
```

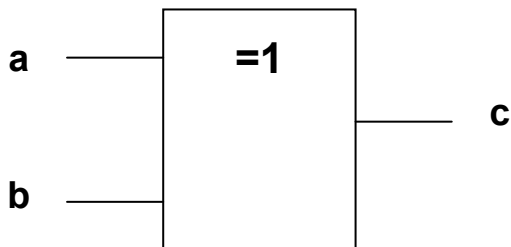
### Beispiel: XOR - Gatter

```
entity xor_gate is
  port (a, b : in boolean;    -- bit
        c   : out boolean);  -- bit
end xor_gate;
```

### Beispiel: XOR - Funktion

```
architecture Specification of xor_gate is
begin
  c <= a xor b;
end architecture Specification;
```

**XOR – Gatter**



**Wahrheitstabelle**

a	b	c
0	0	0
0	1	1
1	0	1
1	1	0

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 11

### ***Beschreibung in sequenzieller Form (Prozess)***

```

architecture Algorithmic of xor_gate is
begin
  process(a, b)
  begin
    if a /= b then
      c <= '1';           -- c <= true;
    else c <= '0';       -- c <= false;
    end if;
  end process;
end architecture Algorithmic;

```

### ***Beschreibung der booleschen Gleichung***

```

architecture BooleGln of xor_gate is
begin
  c <= (a and (not b)) or ((not a) and b);
end BooleGln;

```

### ***Beschreibung mit Signalzuweisungen (concurrent) nach Auflösung der booleschen Gleichung***

```

architecture Datenfluss of xor_gate is
  signal x, y: bit;
begin
  c <= x or y;
  x <= a and not b;
  y <= not a and b;
end architecture Datenfluss;

```

### Beschreibung der Wahrheitstafel

```

architecture WHT of xor_gate is
begin
  c <=
    '0' when (a = '0') and (b = '0') else
    '1' when (a = '0') and (b = '1') else
    '1' when (a = '1') and (b = '0') else
    '0' when (a = '1') and (b = '1') else
    '0';
end architecture WHT;

```

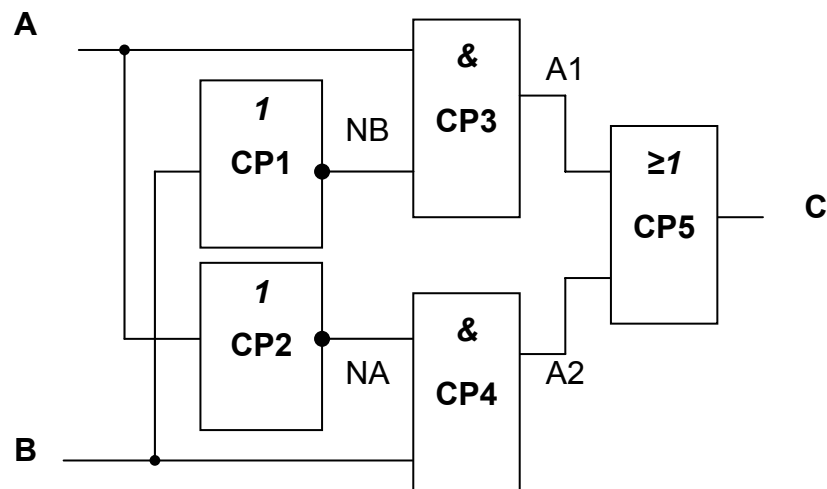
### Beschreibung als gesteuerter Inverter für die Komplementbildung

```

architecture Arithmetic of xor_gate is
begin
  c <= b when (a = '0') else not b;
end architecture Arithmetic;

```

### XOR als Schaltnetz



<b>Universität Duisburg - Essen</b>	<b>Dr.-Ing. R. Viga</b>	<b>Fak. f. Ing.-Wiss. EIT, EBS/VS</b>
<b>Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2</b>		<b>SEITE: 13</b>

## ***XOR Strukturmodell***

```

entity inverter is
  port(A: in bit; B: out bit);
end inverter;

architecture basic of inverter is
begin
  B <= not A;
end architecture basic;

-----

entity and2 is
  port(A, B: in bit; C: out bit);
end and2;

architecture basic of and2 is
begin
  C <= A and B;
end architecture basic;

-----

entity or2 is
  port(A, B: in bit; C: out bit);
end or2;

architecture basic of or2 is
begin
  C <= A or B;
end architecture basic;

-----

entity xor_gate is
  port(A, B : in bit;
        C      : out bit);
end xor_gate;

architecture Structure of xor_gate is
  signal NA, NB, A1, A2: bit;
begin
  CP1: entity work.inverter(basic)
    port map(A, NA);
  CP2: entity work.inverter(basic)
    port map(B, NB);
  CP3: entity work.and2(basic)
    port map(A, NB, A1);
  CP4: entity work.and2(basic)
    port map(NA, B, A2);
  CP5: entity work.or2(basic)
    port map(A1, A2, C);
end architecture Structure

```

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 14

### ***Beispiel: Gemischte Verhaltens- und Strukturbeschreibung***

```

entity multiplier is
  port (Clk, Reset: in bit;
         Multiplicand, Multilplier: in integer;
         Product: out integer);
end multiplier;

architecture mixed of multiplier is
  signal partial_product, full_product: integer;
  signal arith_control, result_en, mult_bit, mult_load: bit

begin  -- mixed
  arith_unit: entity work.shift_adder(behavior)
    port map(addend => multiplicand, augend => full_product,
             sum => partial_product,
             add_control => arith_control);
  result: entity work.reg(behavior)
    port map(d => partial_product, q => full_product,
             en => result_en, reset => reset);
  multiplier_sr: entity work.shift_reg(behavior)
    port map(d => multiplier, q => mult_bit,
             load => mult_load, clk => Clk);
  product <= full_product;

  control_section: process is
    -- variable declarations
    -- .....
  begin  -- control section
    -- sequential staments to assign values to control signals
    -- .....
    wait on Clk, Reset;
  end process control_section;

end architecture mixed;

```

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 15

### Testen von VHDL Modellen (Template 1)

```

entity xor_gate is
--  port (a, b : in bit;
--        c    : out bit);
end xor_gate;

architecture Specification of xor_gate is
    signal a, b: bit;          -- hinzufügen
begin
    ----- Testmuster -----
    a <= '0',
        '1' after 20 ns,
        '0' after 40 ns,
        '1' after 60 ns;

    b <= '0',
        '0' after 20 ns,
        '1' after 40 ns,
        '1' after 60 ns;
    ----- Testmuster -----

    c <= a xor b;
end architecture Specification;

```

- Portdeklaration in entity auskommentieren und als lokale Signalliste vereinbaren
- Testmuster in die Architektur einfügen

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 16

## *Testen von VHDL Modellen (Template 2)*

```

entity test is
end test;

use work.all -- (1)

architecture behavior of TEST is
    signal a, b, c: bit; -- (5)

    component xorg is -- (2)
        port (a, b : in bit;
              c : out bit);
    end component;
    for all: xorg use entity work.xor_gate(Specification); -- (3)
    --for all: xorg use entity work.xor_gate(Algorithmic);
    -- .....

begin
    a <= '0',
        '1' after 20 ns,
        '0' after 40 ns,
        '1' after 60 ns;

    b <= '0',
        '0' after 20 ns,
        '1' after 40 ns,
        '1' after 60 ns;

    DUT: xorg port map(a, b, c); -- (4)

end behavior;

```

- Eigene Testentity in der die zu testende Schaltung als Komponente eingefügt wird
- Tests werden mit Signalzuweisungen beschrieben
- Portmap der Testkomponente verbindet die Signale zu den Eingängen

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 17

### *Testen von VHDL Modellen (Template 3)*

```

--... wie Template 2
begin

  process is

    a <= '0'; b <= '0';
    wait for 20 ns;

    a <= '1'; b <= '0';
    wait for 20 ns;

    a <= '0'; b <= '1';
    wait for 20 ns;

    a <= '1'; b <= '1';
    wait for 20 ns;

  end process;

  DUT: xorg port map(a, b, c);           -- (4)

end behavior;

```

- Generell wie Template 2, doch
- Testmuster werden über einen eigenen Prozess erzeugt

### Simulation von VHDL Modellen (1)

(1)	<code>AS &lt;= X * Y after 2 ns;</code>	-- verzögerte Signalzuweisung
(2)	<code>BS &lt;= AS + Z after 2 ns;</code>	-- verzögerte Signalzuweisung

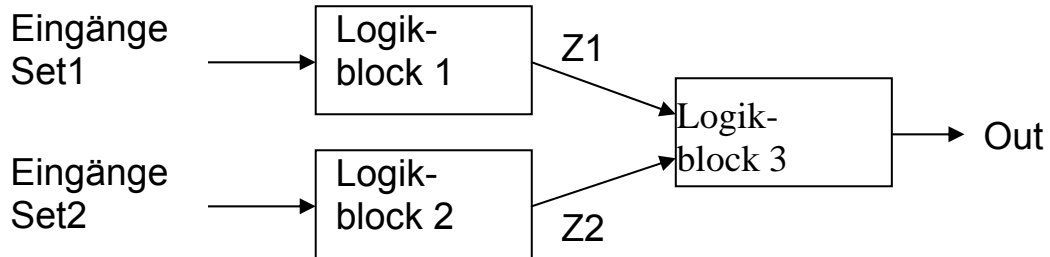
(3)	<code>AV := X * Y;</code>	-- direkte Variablenzuweisung
(4)	<code>BV := AV + Z;</code>	-- direkte Variablenzuweisung

Zeit	Init	t1	t1+2	t1+4	t1+6
X	1	4	5	5	3
Y	2	2	2	3	2
Z	0	3	2	2	2

Zeit	Init	t1	t1+2	t1+4	t1+6
X	1	4	5	5	3
Y	2	2	2	3	2
AS	2	2	8	10	15
Z	0	3	2	2	2
BS	2	2	5	10	12

Zeit	Init	t1	t1+2	t1+4	t1+6
X	1	4	5	5	3
Y	2	2	2	3	2
AV	2	8	10	15	6
Z	0	3	2	2	2
BV	2	11	12	17	8

### Simulation von VHDL Modellen (2)



**Schaltung aus Logikblöcken beliebiger Funktion**

```

Logic_Block1:
  process (X1, X2, X3) is
    variable Yint: bit;
  begin
    Yint := X1 and X2;
    Z1 <= Yint or X3 after 30 ns;
  end process Logic_Block1;
  
```

**Prozessbeschreibung (Beispiel) für Logikblock 1**

### *Simulation von VHDL Modellen (3)*

```

AS <= X * Y;           -- Zuweisung S1
BS <= AS + Z;         -- Zuweisung S2

```

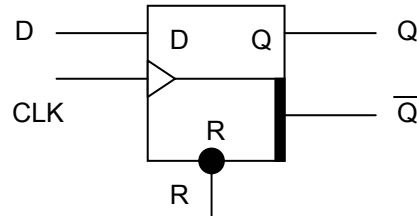
```

process (X,Y) is      -- gleiches Verhalten wie S1
begin
  AS <= X * Y;
end process;

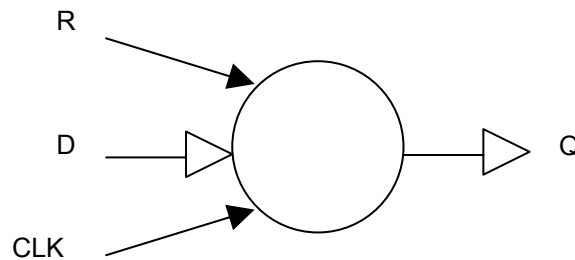
```

<b>Zeit</b>	<b>Init</b>	<b>t1</b>	<b>t1+delta</b>	<b>t1+2* delta</b>
<b>X</b>	<b>1</b>	<b>4</b>	<b>4</b>	<b>4</b>
<b>Y</b>	<b>2</b>	<b>2</b>	<b>2</b>	<b>2</b>
<b>AS</b>	<b>2</b>	<b>2</b>	<b>8</b>	<b>8</b>
<b>Z</b>	<b>0</b>	<b>3</b>	<b>3</b>	<b>3</b>
<b>BS</b>	<b>2</b>	<b>2</b>	<b>5</b>	<b>11</b>

## *Digitales Schaltelement D-Flipflop*



**Schaltsymbol nach DIN**



### **Grafische Repräsentation mit Signalcharakterisierung**

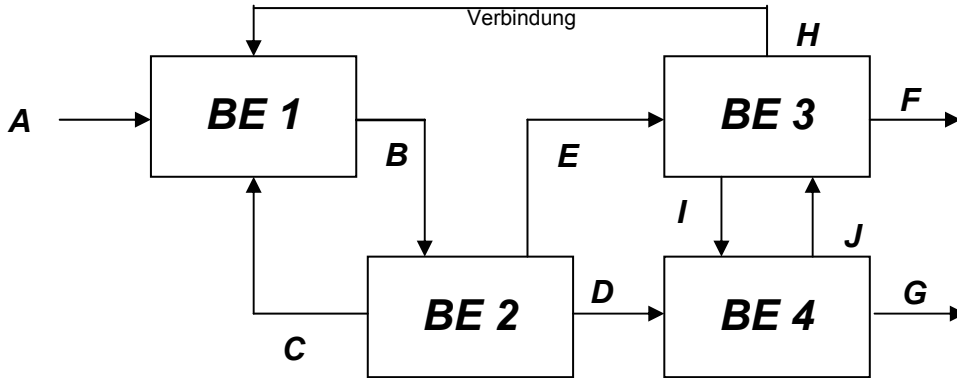
```

D-Flipflop:
process (CLK,R) is
begin
  if R='0' then
    Q <= '0';
  elsif CLK'event and CLK='1' then
    Q <= D;
  end if;
end process;

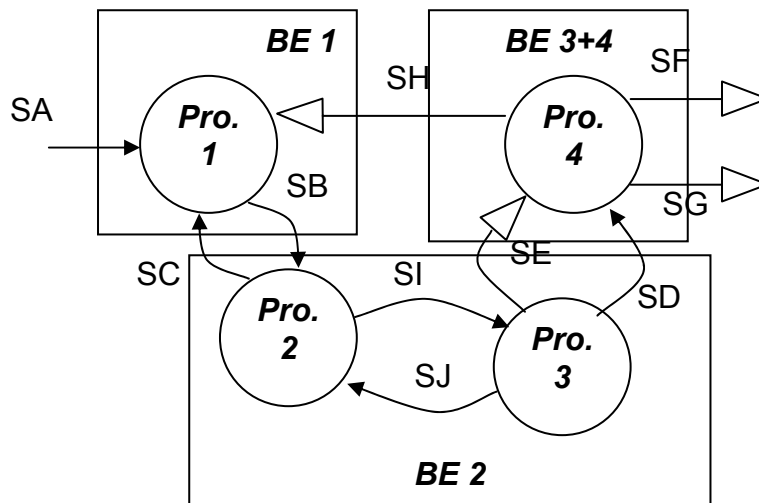
```

### **VHDL Modell**

**Schaltungsmodell aus Prozessen**

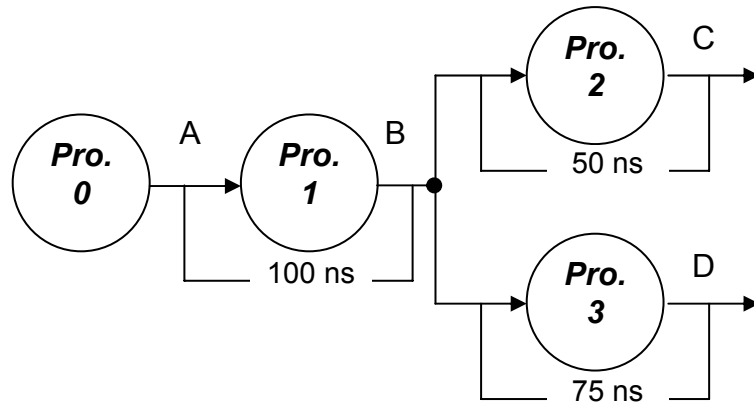


**Digitalschaltung aus Bauelementen BE i**



**Mögliches Prozessmodell mit Überdeckungen**

**Prozess-Simulation (1)**



**Prozessbeispiel**

Zeit	Trans- aktion
0	A, 1
...	...
100	B, 1
...	...
150	C, 0
...	...
175	D, 1

**Transaktionen über die Zeit**

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 24

### ***Prozess-Simulation (2)***

1. Setze die Simulationszeit auf den Zeitpunkt der nächsten Eintragung in der Transaktionsliste. Gibt es keine weiteren Eintragungen ist die Simulation beendet.
2. Aktiviere alle Prozesse, auf die die eingetragenen Signale mit Events triggern.
3. Bearbeite alle aktivierten Prozesse und trage ggf. in die Transaktionsliste ein; gehe zu Schritt 1.

### ***Einfacher Simulationszyklus***

1. Setze die Simulationszeit auf den Zeitpunkt der nächsten Eintragung in der Transaktionsliste. Gibt es keine weiteren Eintragungen ist die Simulation beendet, sonst weiter bei Schritt 2.
2. Starte einen neuen Simulationszyklus *ohne die Simulationszeit zu erhöhen*. Aktiviere alle Prozesse, die von den Signalen mit Events getriggert werden.
3. Bearbeite alle aktivierten Prozesse und trage ggf. in die Transaktionsliste ein. Einige der neuen Einträge können DeltaT beinhalten!
4. Gibt es neue Events an Signalen durch Zuweisungen mit DeltaT über Schritt 3, gehe zu Schritt 2, ansonsten weiter bei Schritt 1.

### ***Simulationszyklus mit DeltaT (Nullzeit-Zyklen)***

<b>Universität Duisburg - Essen</b>	<b>Dr.-Ing. R. Viga</b>	<b>Fak. f. Ing.-Wiss. EIT, EBS/VS</b>
<b>Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2</b>		<b>SEITE: 25</b>

### ***Prozess-Simulation (3)***

```

entity BUFF is
  port(X: in bit; Z: out bit);
end BUFF;

-----
architecture ONE of BUFF is
begin
  process (X) is
    variable Y1: bit;
  begin
    Y1 := X;
    Z <= Y1 after 1 ns;
  end process;
end architecture ONE;

-----
architecture TWO of BUFF is
  signal Y2: bit;
begin
  Y2 <= X;
  Z <= Y2;
end architecture TWO;

-----
architecture THREE of BUFF is
  signal Y3: bit;
begin
  Y3 <= X;
  Z <= Y3 after 1 ns;
end architecture THREE;

-----
architecture FOUR of BUFF is
  signal Y4: bit;
begin
  Y4 <= X after 1 ns;
  Z <= Y4 after 1 ns;
end architecture FOUR;

```

### ***Beispiele verschiedener Architekturen***

**Prozess-Simulation (4)**

Zeit (ns)	X	Z1	Y2	Z2	Y3	Z3	Y4	Z4
0	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'
+1			'0'	'0'	'0'			
1	'1'	'0'				'0'	'0'	'0'
+1			'1'		'1'			
+2				'1'				
2		'1'				'1'	'1'	
3								'1'
4	'0'							
+1			'0'		'0'			
+2				'0'				
5		'0'				'0'	'0'	
6								'0'

**Reaktion der vier Beispielarchitekturen**

### Prozess-Simulation (5)

```

architecture FIVE of BUFF is
  signal Y5: bit;
begin
  process (X) is
  begin
    Y5 <= X;
    Z <= Y5;
  end process;
end architecture FIVE;

-----
architecture FIVE_A of BUFF is
  signal Y5: bit;
begin
  process (X, Y5) is
  begin
    Y5 <= X;
    Z <= Y5;
  end process;
end architecture FIVE_A;

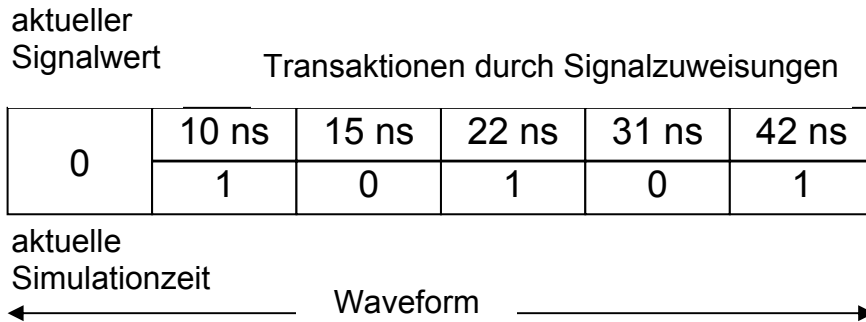
```

### Beispielarchitekturen mit Prozessen

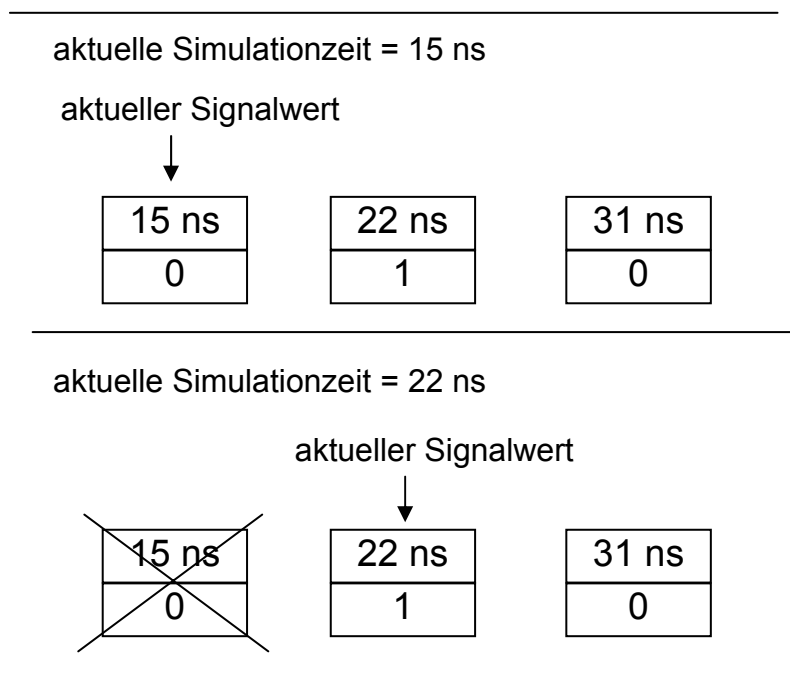
Zeit (ns)	X	Y5	Z5	Y5A	Z5A
0	'0'	'0'	'0'	'0'	'0'
+1		'0'	'0'	'0'	'0'
1	'1'				
+1		'1'	'0'	'1'	'0'
+2				'1'	'1'
2					
3					
4	'0'				
+1		'0'	'1'	'0'	'1'
+2				'0'	'0'
5					
6					

### Reaktion der vier Beispielarchitekturen

### Signaltreiber und Simulation



### Signaltreiber eines Prozesses

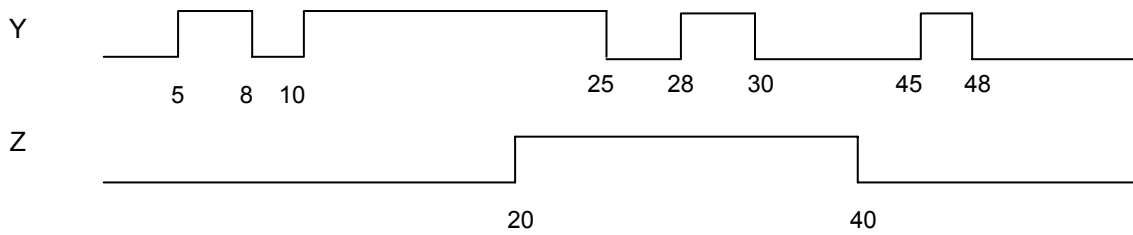


### Fortschreiten der Simulationszeit

## Verzögerungsmechanismen

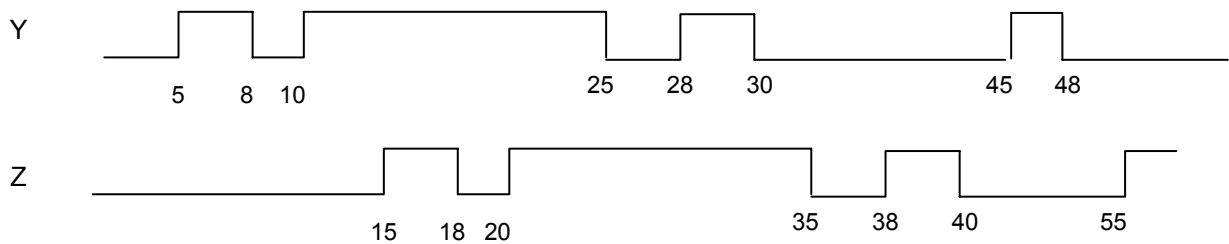
### *Inertial Delay*

**Z <= reject 4 ns inertial Y after 10 ns;**



### *Transport Delay*

**Z <= transport Y after 10 ns;**



### Treiber Update (1)

Z <= 3 **after** 5 ns, 21 **after** 10 ns, 14 **after** 17 ns;

Z <=

curr   now	3   T + 5ns	21   T + 10ns	14   T + 17ns
------------	-------------	---------------	---------------

---

Z <= **transport** 11 **after** 10 ns;

Z <= **transport** 20 **after** 22 ns;

Z <= **transport** 35 **after** 18 ns;

Z <=

curr   now	11   T + 10ns	20   T + 22ns
------------	---------------	---------------

Z <=

curr   now	11   T + 10ns	<del>20   T + 22ns</del>	35   T + 18ns
------------	---------------	--------------------------	---------------

Z <= 11 **after** 10 ns;

-- inertial is default

Z <= **reject** 15 ns **inertial** 22 **after** 20 ns;

Z <= 33 **after** 15 ns;

Z <=

curr   now	<del>11   T + 10ns</del>	22   T + 20ns
------------	--------------------------	---------------

**Löschen, da 10ns  
zwischen 5ns und 20ns !**

## *Treiber Update (2)*

### *Regeln zum Aufbau eines Treibers:*

1. Alle Transaktionen im Treiber, die zu oder nach der Delayzeit der ersten neuen Transaktionszeit auftreten, werden gelöscht, wie im Transport-Fall.
2. Alle neuen Transaktionen werden am Ende des Treibers angefügt.
3. alle Transaktionen, die in den Zeitraum (neue Transaktionszeit – Rejectzeit) bis zur neuen Transaktionszeit fallen und einen unterschiedlichen Wert aufweisen sind zu löschen.

### *Beispiel*

Y <= **transport X after 2 ns**;  
Z <= X **after 2 ns**;

X <=	0   0ns	1   1ns	0   2ns	1   7ns	0   12ns
------	---------	---------	---------	---------	----------

Ereignis	SimZeit	Y Treiber	Z-Treiber
	t = 0	<0>	<0>
ja (X = 1)	t = 1	<0> <1   3>	<0> <1   3>
ja (X = 0)	t = 2	<0> <1   3> <0   4>	<0> <1   3> <0   4>
nein	t = 3	<1> <0   4>	-----
nein	t = 4	<0>	<0>
ja (X = 1)	t = 7	<0> <1   9>	<0> <1   9>
nein	t = 9	<1>	<1>
ja (X = 0)	t = 12	<1> <0   14>	<1> <0   14>
nein	t = 14	<0>	<0>

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 32

## ***Wait Anweisungen (1)***

### ***Erzeugen eines Taktsignals***

```
clock_gen: process (clk) is
begin
  if clk = '0' then
    clk <= '1' after T, '0' after 2*T;
  end if;
end process clock_gen;
```

### ***ist eine Konvention für den folgenden Prozess mit gleichem Verhalten***

```
clock_gen: process is
begin
  if clk = '0' then
    clk <= '1' after T, '0' after 2*T;
  end if;
  wait on clk;
end process clock_gen;
```

### ***Umgehung des if-then Konstrukts***

```
clock_gen: process is
begin
  clk <= '1' after T, '0' after 2*T;
  wait until clk = '0';
end process clock_gen;
```

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 33

## *Wait Anweisungen (2)*

### *clock\_gen Prozess mit Timeout*

```
clock_gen: process is
begin
  clk <= '1' after T, '0' after 2*T;
  wait for 2*T;
end process clock_gen;
```

### *Prozess mit Timeout DeltaT*

```
wait0: process is
begin
  wait on data;
  siga <= data;
  wait for 0 ns;
  sigb <= siga;
end process wait0;
```

### *Anregung von Einprozess-Modellen*

```
entity quicksort is
end quicksort;

architecture behavior of quicksort is
-- signal start: bit:= '0';
begin
-- start <= '1' after 10 ns;    -- if sensitivity list
-- process (start) is        -- if sensitivity list
  process is
  begin
    ....
--   wait on start;          -- erase if sensitivity
    wait;                   -- erase if sensitivity
  end process;
end architecture behavior;
```

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 34

## *Lexikalische Elemente (1)*

Lexikalische Elemente sind:

- besondere **Begrenzungszeichen** (delimiter),
- **Bezeichner** (identifier),
- **Kommentare** (comment),
- **Zeichen** (character literal),
- **Zeichenketten** (string literal),
- **Bitzeichenketten** (bit string literal) und
- **abstrakte Zahlen** (abstract literal).

Lexikalische Elemente werden in der Sprache getrennt durch **Separatoren**:

- Leerzeichen,
- Tabulator und
- Zeilenende-Zeichen.

### ***Begrenzungszeichen (delimiter)***

- für Operationen (Operatoren),
- als Begrenzer für bestimmte Teilkonstrukte der Sprache und
- zur Interpunktion.

" # & ' ( ) \* + - , . / : ; < = > [ ] |

### ***Paare von Sonderzeichen***

=> \*\* := /= >= <= <>

<= steht für den Vergleich: kleiner oder gleich in Ausdrücken  
ist aber auch das Zuweisungssymbol für Signale

Begrenzungszeichen können Separatoren ersetzen, z.B.

( **A + B** ) kann auch geschrieben werden (**A+B**), also ohne Leerzeichen zwischen Operanden und Operator (Begrenzerfunktion von +).

## *Lexikalische Elemente (2)*

### **Bezeichner (identifizier)**

benennen Benutzer-**Objekte** in VHDL, wie z.B. **Variable, Signale, Prozesse** u.s.w.  
für die Sprache selbst belegte Bezeichner sind **reservierte Worte**

- sie dürfen nur Groß- und Kleinbuchstaben, Zahlen und den Unterstrich "\_" enthalten,
- sie müssen mit einem Buchstaben beginnen,
- sie dürfen *nicht* mit einem Unterstrich beginnen oder enden und
- sie dürfen *keine zwei aufeinander folgenden* Unterstriche beinhalten.

erlaubte Bezeichner:

**COUNT** ist identisch **count**;    **last\_value**,    **h3Z25**,    **Date\_2305**

unerlaubter Bezeichner:

<b>last@value</b>	-- illegales Zeichen
<b>5bit_counter</b>	-- beginnt mit numerischen Zeichen
<b><u>A0</u></b>	-- beginnt mit Underline
<b>A0_</b>	-- endet mit Underline
<b>clock__pulse</b>	-- zwei Underlines
<b>end</b>	-- reserviertes Wort ist nicht erlaubt

**reservierte Worte** in VHDL sind:

<b>abs</b>	<b>disconnect</b>	<b>label</b>	<b>package</b>	<b>sla</b>
<b>access</b>	<b>downto</b>	<b>library</b>	<b>port</b>	<b>sll</b>
<b>after</b>	<b>else</b>	<b>linkage</b>	<b>postponed</b>	<b>sra</b>
<b>alias</b>	<b>elsif</b>	<b>literal</b>	<b>procedure</b>	<b>srl</b>
<b>all</b>	<b>end</b>	<b>loop</b>	<b>process</b>	<b>subtype</b>
<b>and</b>	<b>entity</b>	<b>map</b>	<b>protected</b>	<b>then</b>
<b>architecture</b>	<b>exit</b>	<b>mod</b>	<b>pure</b>	<b>to</b>
<b>array</b>	<b>file</b>	<b>nand</b>	<b>range</b>	<b>transport</b>
<b>assert</b>	<b>for</b>	<b>new</b>	<b>record</b>	<b>type</b>
<b>attribute</b>	<b>function</b>	<b>next</b>	<b>register</b>	<b>unaffected</b>
<b>begin</b>	<b>generate</b>	<b>nor</b>	<b>reject</b>	<b>units</b>
<b>block</b>	<b>generic</b>	<b>not</b>	<b>rem</b>	<b>until</b>
<b>body</b>	<b>group</b>	<b>null</b>	<b>report</b>	<b>use</b>
<b>buffer</b>	<b>guarded</b>	<b>of</b>	<b>return</b>	<b>variable</b>
<b>bus</b>	<b>if</b>	<b>on</b>	<b>rol</b>	<b>wait</b>
<b>case</b>	<b>impure</b>	<b>open</b>	<b>ror</b>	<b>when</b>
<b>component</b>	<b>in</b>	<b>or</b>	<b>select</b>	<b>while</b>
<b>configuration</b>	<b>inertial</b>	<b>others</b>	<b>severity</b>	<b>with</b>
<b>constant</b>	<b>inout</b>	<b>out</b>	<b>shared</b>	<b>xnor</b>
	<b>is</b>		<b>signal</b>	<b>xor</b>

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 36

### ***Lexikalische Elemente (3)***

#### ***Kommentare (comment)***

beginnen mit dem Doppelminus "--" und enden am Zeilenende:

-- **es folgt eine Variablenzuweisung**  
C := A + B;      -- **dies ist eine gültige Variablenzuweisung**

#### ***Zeichen (character literal)***

werden in VHDL in Hochkommas eingerahmt:

Großbuchstaben: 'A', 'B', ..., 'Z';	Kleinbuchstaben: 'a', 'b', ..., 'z';
Ziffern: '0', '1', ..., '9';	Sonderzeichen: '!', '\$', '&', ...;
Hochkomma: "" und	Leerzeichen: ' '.

#### ***Zeichenketten (string literal)***

bestehen aus einer Folge von Einzelzeichen und werden in Anführungszeichen angegeben.

"Eine Zeichenkette"      oder auch      "any printing chars (&%@^\*)",  
"0011ZZZ"                      oder auch      "", der leere String.

Zeichenketten, die nicht in eine Zeile passen, können auch mit dem *Kettungsoperator für Strings*, dem "&" aneinander gekettet werden, wie z.B.

**"Eine Zeichenkette, die nicht in eine Zeile passt" &  
"kann mit dem Operator & gekettet werden"**

um die zwei Teilstrings zu einem zu binden.

***Im Praktikumsversuch 2*** wird der Datentyp ***bit\_vector*** verwendet, der einen String darstellt (ein String ist in VHDL ein eindimensionales Feld, ein Vektor). Das Verschieben eines 8 stelligen strings **d** wird durch die Verkettung des ***character\_literal*** '0' mit dem String (bit\_vector) von links nach rechts ohne die letzte Stelle erreicht:

**d := '0' & d(7 downto 1);**      -- Verkettung von Zeichenketten mit &

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 37

## *Lexikalische Elemente (4)*

### *Bitzeichenketten (bit string literal)*

geben Zahlenwerte an.

- **binäre Angaben** mit **B"..."**,
- **oktale Angaben** mit **O"..."** oder
- **hexadezimale Angaben** mit **X"..."**.

Entsprechend sind die in Bitzeichenketten zugelassenen Zeichen

- bei **B** nur **0** oder **1**,
- bei **O** sind es die Zeichen **0, 1, 2, 3, 4, 5, 6, 7** und
- bei **X** die Zeichen **0, 1, ..., 9, A, B, C, D, E, F**.

Bitketten ermöglichen z.B. Speicherinhalte nicht immer binär sondern auch gekürzt als Oktal- oder Hexadezimalwert anzugeben. Das Resultat ist in jedem Fall ein als Bitstring angegebenes Bitmuster.

**B"0100011"**, oder **B"10"**, oder auch **b"1111\_0010\_1010\_0101"**, oder **B""**.

**Oktal angegebene Strings** fassen 3-bit Gruppen zusammen, wie

**O"372"** ist äquivalent zu **B"011\_111\_010"**, oder auch **o"00"**, das äquivalent **B"000\_000"** ist.

**Hexadezimal angegebene Strings** fassen 4-bit Gruppen zusammen, wie

**X"FA"** äquivalent zu **B"1111\_1010"**, oder auch **x"0d"** äquivalent zu **B"0000\_1101"**.

Man beachte, dass **O"372"** nicht äquivalent zu **X"FA"** ist, da das erste Literal 9 Bitstellen anspricht, der letzte jedoch nur 8!!!

### *Zahlenangaben (abstract literal)*

wir können zwischen **Ganzzahlen (Integer)** und **gebrochen rationalen Zahlen (Real)** unterscheiden.

#### *Integer (integer literal)*

besteht aus Stellen (**digits**) ohne einen Dezimalpunkt.  
Dezimale Integer-Literale sind z.B. **23 0 146**.

#### *Reals (real literal)*

haben immer einen Dezimalpunkt, durch den sie sich von den Integer Literalen unterscheiden.

Dezimale Real Literale sind z.B. **23.1 0.0 3.14159**.

Ohne Beweis sollte klar sein, dass die maschinelle Repräsentation solcher Rationalzahlen auf Grund der Stellenbeschränkung nur eine Näherung darstellen kann.

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 38

### ***Lexikalische Elemente (5)***

***abstract\_literal*** ist einerseits ***decimal\_literal*** und andererseits ***based\_literal***. Beide Literale können auch in der sog. **E-Notation** erscheinen. Die Zahl hinter dem "E" oder dem "e" bezeichnet die Anzahl der Positionen, die der Stellenpunkt **nach links (negativ)** oder **nach rechts** zu verschieben ist.

***Beispiele dezimaler Literale:***

Integerzahlen:

<b>46E5</b>	entspricht der Ganzzahl	4600000,
<b>1E+12</b>	ist entsprechend	10 <sup>12</sup> und
<b>19e00</b>	ist gleich	19.

Realzahlen in E-Notation:

<b>1.234E09</b>	ist	1.234*10 <sup>9</sup> ,
<b>98.6E+21</b>	ist	0.986*10 <sup>23</sup> und
<b>34.0e-08</b>	ist	0.00000034.

Man kann in VHDL auch die Zahlen in verschiedenen Basis-Systemen als sog. ***based\_literal*** angeben. Die Basis muss zwischen 2 und 16 liegen!

<b>2#11111101#</b>	(Dualzahl)
= <b>8#375#</b>	(Oktalzahl)
= <b>16#0fd# = 16#FD#</b>	(Hexadezimalzahl).

Dies gilt auch für Real-Zahlen. Als Beispiel ist **dezimal 0.5** gleich

**2#0.100# (Dualzahl) = 8#0.4# (Oktalzahl) = 12#0.6# (Basis 12 Zahl)**

Auch die E-Notation gilt:

**2#1#E10 = 2<sup>10</sup>,      16#4#E2 = 4\*16<sup>2</sup>      10#1024#E+00 = 1024.**

Man beachte, dass die Zahlenangabe hinter dem „E“ eine dezimale ist.

Aus Gründen der besseren Lesbarkeit kann man auch Gruppen von Ziffern mit dem Unterstrich "\_" abtrennen, ähnlich der gewohnten Tausender-Trennung im Dezimalen.

**123\_456 ,      3.141\_592\_6 ,      2#1111\_1100\_0000\_0000#.**

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 39

## ***Erweiterte Backus-Naur Form (EBNF)***

Produktionsregel für syntaktische Einheit ***Variablenzuweisung***:

**variable\_assignment\_statement** ::= target := expression;

**::=** bedeutet „ist definiert zu“;

**target** Bezeichner hier für eine Variable,

**:=** Begrenzer, hier mit der Bedeutung „wird zugewiesen“ und

**expression** (Ausdruck) weitere Einheit; wieder in EBNF beschrieben.

Die wait-Anweisung ist in der EBNF im VHDL LRM beschrieben:

**wait\_statement** ::= **wait** [sensitivity\_clause] [condition\_clause] [timeout\_clause];

Dem Schlüsselwort **wait** folgen drei geklammerte Klauseln.

[ ... ] bedeutet, dass der Inhalt der Klammer optional ist, d.h. auch fehlen darf.

Weiter abgeleitet ergibt sich für die erste Klammer:

**sensitivity\_clause** ::= **on** sensitivity\_list

Dem Schlüsselwort **on** folgt eine weitere Einheit (Sensitäts-Signalliste), die weiter abzuleiten ist:

**sensitivity\_list** ::= *signal\_name* { , *signal\_name* }

Die syntaktische Einheit *signal\_name* steht in der geschweiften Klammer nach einem Begrenzer Kommabegrenzer.

{ ... } bedeutet, dass der Inhalt ***nicht oder beliebig oft*** wiederholt vorkommen darf.

Die Einheit ***Signalname*** ist gegeben zu:

**signal\_name** ::= identifier

womit die Regel-Ableitung an einem Terminal ***identifier*** (Bezeichner, s.o.) endet.

Die Bedingungsklausel ist wie folgt abzuleiten:

**condition\_clause** ::= **until** condition

**condition** ::= *boolean\_expression*

Eine Bedingung nach dem Schlüsselwort ***until*** ist somit ein boolescher Ausdruck, also einer der zu ***true*** oder ***false*** evaluierbar ist.

Die Zeitklausel beginnt mit dem Schlüsselwort ***for*** und verlangt einen Ausdruck, der in einen ***Zeitwert*** evaluiert werden soll:

**timeout\_clause** ::= **for** *time\_expression*

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 40

## **Zuweisungen**

Man erkennt, dass der Ausdruck (expression) formal eine sehr komplexe syntaktische Einheit ist und teilweise **semantische Details** (in Kursivschrift angedeutet) genauere Aussagen treffen.

Wir wollen etwas abweichen vom LRM unter Wertbestimmungs-Ausdruck (*value\_expression*) verstehen:

**value\_expression** ::= *numerical\_expression* | *boolean\_expression*

Die obige Zeitwert-Ausdruck (*time\_expression*) ist eine besondere Wertbestimmung.

Unter numerischer Wertbestimmung

**numerical\_expression** ::= [ + | - ] *num\_term* { ( + | - | ) *num\_term* }  
**num\_term** ::= [abs] *numerical\_factor* { ( \* | / | mod | rem ) *numerical\_factor* }  
**numerical\_factor** ::= constant | variable | signal      -- als numerischen Typ

Die Wertbestimmung eines booleschen Typs (true | false):

**condition** ::= *boolean\_expression*  
**boolean\_expression** ::= [not] *boolean\_factor* { ( and | or | .... ) [not] *boolean\_factor* }  
**boolean\_factor** ::= constant | variable | signal      -- als boolean Typ

Die Bedingung ist also auch ein *boolean\_expression*!

## **Variablenzuweisung (wie oben)**

**variable\_assignment\_statement** ::= target := expression;

## **Signalzuweisung (sequenziell)**

**signal\_assignment\_statement** ::= target <= {delay\_mechanism} waveform;  
**delay\_mechanism** ::= transport | [reject *time\_expression*] inertial  
**waveform** ::= waveform\_element { , waveform\_element }  
**waveform\_element** ::= *value\_expression* [after *time\_expression*]

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 41

### ***Prozessdeklaration***

```

process_statement ::=
[ process_label :]
  process [ (sensitivity_list) ] [is]
    process_declarative_part
  begin
    process_statement_part
  end process [ process_label ];

process_statement_part ::= { sequential_statement }

```

### ***Architekturdeklaration***

```

architecture_body ::=
  architecture identifier of entity_name is
    architecture_declarative_part
  begin
    architecture_statement_part
  end [architecture] [ architecture_name ];

architecture_statement_part ::= { concurrent_statement }

```

### ***Blockdeklaration***

```

block_statement ::=
[ block_label :]
  block [ (guard_expression) ] [is]
    block_header
    block_declarative_part
  begin
    block_statement_part
  end block [ block_label ];

block_statement_part ::= { concurrent_statement }

```

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 42

### ***Befehlsfolgen in Prozessen***

```

sequence_of_statements ::= { sequential_statement }
sequential_statement ::=
    wait_statement
    | assertion_statement
    | report_statement
    | signal_assignment_statement
    | variable_assignment_statement
    | procedure_call_statement
    | if_statement
    | case_statement
    | loop_statement
    | next_statement
    | exit_statement
    | return_statement
    | null_statement

```

### ***nebenläufige Befehle in Architektur und in Blöcken***

```

block_statement_part ::= { concurrent_statement }
concurrent_statement ::=
    block_statement
    | process_statement
    | concurrent_procedure_call_statement
    | concurrent_assertion_statement
    | concurrent_signal_assignment_statement
    | component_instantiation_statement
    | generate_statement

```

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 43

## ***Aus-/Einschluss bestimmter Signalzuweisungen in Blöcken (guarded blocks)***

Guard ist ein logisches Signal (true, false) und hat steuernde Funktion für entsprechend gekennzeichnete Signalzuweisungen.

- ist der **Guard = true**, dann werden alle Guarded-Signalzuweisungen im Block ausgeführt, wenn sich die Zuweisungswerte ändern,
- ist der **Guard = false**, werden diese Signalzuweisungen deaktiviert und
- **wechselt der Guard von false nach true**, werden die Signalzuweisungen auf jeden Fall ausgeführt.

### ***Guarded Block Konstrukt***

```

GaBlo:
  block (guard-expression)
    signal sig: bit;
  begin
    sig <= guarded waveform-elements;
  end block GaBlo;

```

### ***vergleichbares Prozess-Konstrukt***

```

architecture .....
  signal sig: bit;
  signal guard: boolean;
begin
  ....
  guard <= guard-expression;

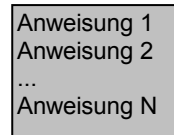
  GaBlo:
  process (guard, signals in waveform-elements) is
  begin
    if guard then
      sig <= waveform-elements;
    end if;
  end process GaBlo;
end architecture ...;

```

## Steuerung des Programmflusses

Struktogramm-Elemente zur Darstellung des Steuerflusses bei sequenzieller Bearbeitung mit den entsprechenden Sprachkonstrukten.

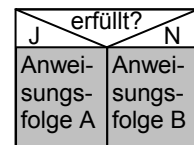
**sequence\_of\_statements ::= { sequential\_statement }**



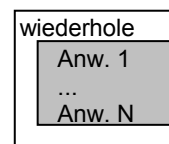
### 1. Sequenz

**sequential\_statement ::=**

- wait\_statement
- | assertion\_statement
- | report\_statement
- | signal\_assignment\_statement
- | variable\_assignment\_statement
- | procedure\_call\_statement
- | if\_statement
- | case\_statement
- | loop\_statement
- | next\_statement
- | exit\_statement
- | return\_statement
- | null\_statement



### 2. Selektion



### 3. Iteration

### IF-Anweisungen (Alternative)

EBNF Produktionsregel:

```

if_statement ::=
  [ if_label : ]
  if condition then
    sequence_of_statements
  { elsif condition then
    sequence_of_statements }
  [ else
    sequence_of_statements ]
  end if [ if_label ];

```

Im Versuch 2 ist **d** deklariert als 8-stelligen Bitvektor in der umgekehrten Stellenfolge  
variable **d** : bit\_vector(7 downto 0);

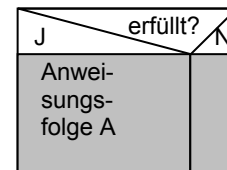
Das einfachste *if* Konstrukt ohne *elsif* oder *else* löst die Abfrage, ob das niedrigstwertige Bit 0 gesetzt ('1') ist:

*einfaches if Konstrukt*

```

if (d and "00000001") = "00000001" then
  n := n + 1;
end if;

```

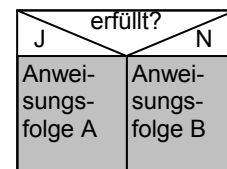


*if-then-else Konstrukt*

```

if (d and "00001111") = "00000000" then
  no1_lh <= false;
else
  no1_lh <= true;
end if;

```

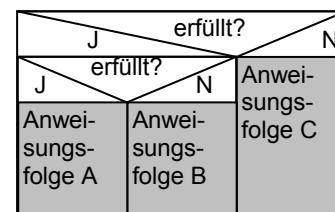


*if Schachtelung*

```

if (d and "00001111") /= "00000000" then
  no1_lh <= false;
  if (d and "00000011") = "00000000" then
    no1_lq <= true;
  else
    no1_lq <= false;
  end if;
else
  no1_lh <= true;
  no1_lq <= true;
end if;

```



Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 46

**Vergleich der Alternativen im sequenziellen Fluss und in nebenläufigen Signalzuweisungen:**

*if- elsif Konstrukt*

```

if en = '0' then
  result <= 0;
elsif op = yes then
  result <= input_0;
else
  result <= input_1;
end if;

```

**EBNF der nebenläufigen Signalzuweisung**

```

concurrent_signal_assignment_statement ::=
  [ label : ] conditional _signal_assignment
  | [ label : ] selected _signal_assignment

conditional_signal_assignment ::=
  target <= options conditional_waveforms;

options ::= [ guarded ] [ delay_mechanism ]

conditional_waveforms ::=
  { waveform when condition else }
  waveform [ when condition ]

```

Setzen wir das obige *elsif*-Konstrukt um in eine solche Signalzuweisung mit gleichem Verhalten, so folgt:

```

result <= 0 when en = '0' else
  input_0 when op = yes else
  input_1;

```

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 47

## Case-Anweisungen (Selektion, Fallunterscheidung)

EBNF:

```

case_statement ::=
  [ case_label : ]
  case expression is
    case_statement_alternative
    { case_statement_alternative }
  end case [ case_label ];

case_statement_alternative ::=
  when choices =>
    sequence_of_statements

choices ::=
  expression
  | discrete range
  | element_name
  | others

```

Beispiel:

```

case state is
  when 0           => act := 0;
  when 1 to 7     => act := 1;
  when 15 downto 8 => act := 2;
  when others    => act := 3;
end case;

```

Die Auswahl **choices** muss alle auftretenden Fälle abdecken! Daher wird **others** benötigt und ggf. auch das **null** Statement.

```

type state is (idle, active, waiting, undefined);

```

.....

```

case state is
  when active   => act := 0;
  when waiting  => act := 1;
  when others  => act := 3;
end case;

```

```

case state is
  when active   => act := 0;
  when waiting  => act := 1;
  when others  => null;
end case;

```

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 48

**Vergleich der Selektion im sequenziellen Fluss und in nebenläufigen Signalzuweisungen:**

*In einem Prozess z.B.:*

```

case d is
  when "00000000" => intsig <= 0;
  when "00000001" => intsig <= 1;
  when "00000010" => intsig <= 2;
  when "00000011" => intsig <= 3;
  .....
  when others => intsig <= 255;
end case;

```

*EBNF (2. Teil der nebenläufigen Signalzuweisung):*

```

selected_signal_assignment ::=
  with expression select
    target <= options selected_waveforms;

selected_waveform ::=
  { waveform when choices, }
  waveform when choices

```

*obiges Case-Konstrukt umgesetzt ergibt:*

```

with d select
  intsig <= 0 when "00000000",
  intsig <= 1 when "00000001",
  intsig <= 2 when "00000010",
  intsig <= 3 when "00000011",
  .....
  intsig <= 255 when others;

```

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 49

## LOOP-Anweisung (Programmschleife)

EBNF der Schleife:

```

loop_statement ::=
  [ loop_label:]
  [ iteration_scheme ] loop
  sequence_of_statements
end loop [ loop_label ];

```

```

iteration_scheme ::=
  while condition
  | for loop_parameter_specification

```

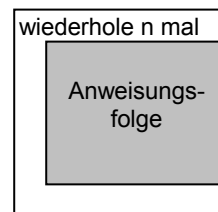
## FOR-LOOP-Anweisung (Zählschleife)

Die *for*-Schleife ist ein Konstrukt aus mehreren möglichen Schleifenkonstrukten:

```

for-loop_Anweisung ::=
  [ loop_Bezeichnung:]
  for Zähler in diskreter_Bereich loop
  sequence_of_statements
end loop [ loop_Bezeichnung ];

```



Strukturblock (Struktogramm, rechts) ist die grafische Darstellung dieses Konstrukts. Der *Zähler* ist eine *implizit vereinbarte* Variable zum Zählen der Durchläufe.

Zwei verschiedene Variablen **a**; **a** ist am Ende des Prozesse gleich 10!

```

example: process is
  variable a, b: integer;
begin
  a := 10;
  for a in 10 downto 5 loop
    b := a;
  end loop;
end process;

```

Aufzählung als Schleifenzähler; **a** ist nur innerhalb der Schleife definiert!

```

type state is (initial, idle, active, error);
variable b: state;
.....
for a in state loop
  b := a;
end loop;

```

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 50

### Zählschleife aus Praktikumsversuch

Im Versuch 2 wollen wir die Anzahl der zu '1' gesetzten Bits im Bitvektor **d** mit einer Integer Variablen **n** zählen, die vorher zu Null gesetzt war.

Das For-Schleifenkonstrukt in Verbindung mit den oben angegebenen zusätzlich benötigten Konstrukten IF und der Verschiebeoperation lösen das Problem:

```

for k in 1 to 8 loop           -- for Schleife
  if (d and "00000001") = "00000001" then
    n := n + 1;                   -- zähle die '1' en
  end if;
  d := '0' & d(7 downto 1);      -- verschiebe nach rechts
end loop;

```

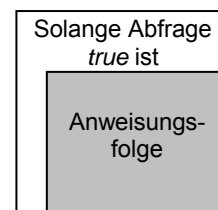
### WHILE-LOOP (abweisende Schleife)

Schleife kann auch umgangen werden (abweisen, wenn Abfrage false); *EBNF*:

```

while-loop_Anweisung ::=
  [ loop_identifizier:]
  while Abfrage loop
    sequence_of_statements
  end loop [ loop_ identifizier ];

```



Beispiel: am Ende des Prozesses ist  $a = 4$  und  $b = 5$ !

```

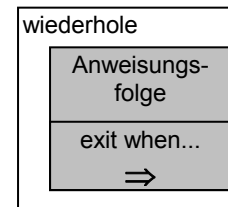
alt_example: process is
  variable a, b: integer;
begin
  a := 10;
  while a >= 5 loop
    b := a;
    a := a - 1;
  end loop;
end process;

```

### ***(REPEAT-) LOOP (allgemeine und annehmende Schleife)***

Die *EBNF* der allgemeinen Schleife, die prinzipiell endlos ist, aber mit der speziellen *exit*-Anweisung abgebrochen werden kann:

```
loop_Anweisung ::=
  [ loop_identifizier:]
loop
  sequence_of_statements
end loop [ loop_identifizier ];
```



Mit den folgenden Sonderanweisungen kann man das Schleifenkonstrukt steuern:

```
exit_statement ::= [ label :] exit [ loop_label ] [ when condition ];
next_statement ::= [ label :] next [ loop_label ] [ when condition ];
```

*Beispiel annehmende Schleife (repeat-until) im Vergleich zu oben:*

```
neu_example: process is
  variable a, b: integer;
begin
  a := 10;
  loop
    b := a;
    a := a - 1;
    exit when a < 5
  end loop;
end process;
```

*Beispiel mit next-Anweisung; b nimmt nur gerade Werte von a an:*

```
variable a, b: integer;
.....
a := 0;
loop
  a := a + 1;
  exit when a > 16;
  next when a mod 2 = 0;
  b := a;
end loop;
```

<b>Universität Duisburg - Essen</b>	<b>Dr.-Ing. R. Viga</b>	<b>Fak. f. Ing.-Wiss. EIT, EBS/VS</b>
<b>Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2</b>		<b>SEITE: 52</b>

*Weitere Beispiele mit Aufzählungen:*

```

type state is (initial, idle, active, error);
variable a, b: state;
.....
a := state'left;
loop
  exit when a = state'right;
  b := a;
  a := state'succ(a);
end loop;

```

```

type state is (initial, idle, active, error);
variable a, b: state;
.....
a := state'right;
loop
  b := a;
  a := state'pred(a);
  if a = state'left then
    b := a;
    exit;
  end if;
end loop;

```

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 53

### *Unterprogramme (Procedure und Function)*

```
subprogram_body ::=
  subprogram_specification is
    subprogram_declarative_part
  begin
    subprogram_statement_part
  end [ subprogram_kind ] [ designator ];
```

```
subprogram_specification ::=
  procedure identifier [ ( formal_parameter_list ) ]
  | function designator [ ( formal_parameter_list ) ]
  return type_mark
```

*EBNF der Prozedur-Deklaration:*

```
procedure_subprogram_body ::=
  procedure identifier [ ( formal_parameter_list ) ] is
    subprogram_declarative_part
  begin
    subprogram_statement_part
  end [ procedure ] [ identifier ];
```

Die **Schnittstelle** zu rufenden Programmen ist der Name **identifier** und die Formal-Parameterliste:

```
formal_parameter_list ::= parameter_interface_list
interface_list ::= interface_element { ; interface_element }
interface_element ::= identifier_list : mode type_name
mode ::= in | out | inout
```

*EBNF der Funktions-Deklaration:*

```
function_subprogram_body ::=
  function designator [ ( formal_parameter_list ) ]
    return type_mark is
    subprogram_declarative_part
  begin
    subprogram_statement_part
  end [ function ] [ identifier ];
```

Im **subprogram\_statement\_part** muss mindestens ein **return expression**;

vom Typ **type\_mark** angegeben werden.

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 54

### ***EBNF zum Prozeduraufruf***

```

procedure_call ::= procedure_name [ ( actual_parameter_part ) ]
actual_parameter_part ::= parameter_association_list
association_list ::= association_element { , association_element }
association element ::= [ formal_part => ] actual_part

```

### ***Beispiel-Vereinbarungen***

```

procedure myproc (a, b, c : in integer; y : out boolean; z : inout positive) is
  ....
begin
  ....
end procedure;

function myfunct (a, b, c : in positive) return boolean is
  ....
begin
  ....
  return true;
  ....
end function;

```

### ***Spezifikationen zu den Beispielen***

```

procedure myproc (a, b, c : in integer; y : out boolean; z : inout positive);
function myfunct (a, b, c : in positive) return boolean;

```

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 55

### *Erlaubte Aufrufe zu den Beispielen*

**myproc** (av, bv, cv, yb, zvar);      **myproc** (33, bv, 55, yb, zvar);  
**myproc** (16\*35, bv, cv, yb, zvar);      **myproc** (33, b, c, y, z);

### *Nicht erlaubte Aufrufe zu den Beispielen*

**myproc** (av, bv, cv, true, zvar);      **myproc** (33, bv, -55, yb, 25);  
**myproc** (true, bv, cv, 20, zvar);      **myproc** (33, b, c, yb, -77);

### *Positionsunabhängige Aktualparameter-Übergabe*

**myproc** (a => av, b => bv, c => cv, y => yb, z => zvar);  
**myproc** (b => bv, z => zvar, c => cv, y => yb, a => av);

### *Prozedurbeispiel: Rotation von Bitvektoren*

```
-- vorher war deklariert
subtype myvector is bit_vector(7 downto 0);
variable d : myvector;
.....
procedure rotate (d : inout myvector; nb : in natural; dir : in bit) is
  variable merk : bit;
begin
  for i in 0 to nb loop
    if dir = '0' then
      merk := d(0);
      d := merk & d(7 downto 1);
    else
      merk := d(7);
      d := d(6 downto 0) & merk;
    end if;
  end loop;
end procedure;
```

rotate(d, 3, '0') - rotiere d um 3 Stellen nach rechts

rotate(d, 1, '1') - rotiere d um 1 Stelle nach links

rotate(d=>d, dir=>'0', nb=>0) - rotiere d keine Stelle

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 56

### ***EBNF zum Funktionsaufruf***

**function\_call** ::= *function\_name* [ ( actual\_parameter\_part ) ]

### ***Funktionsaufruf***

**y := myfunct** (av, bv, cv);                      **y := myfunct** (33, bv, -55);  
**y := myfunct** (b, c, 20);                         **y := myfunct** (33, 44, -77);

### ***Funktionsbeispiel: Wertermittlung für Bitvektor als Dualzahl interpretiert***

```

architecture base of convert is
  subtype myvector is bit_vector(7 downto 0);
  signal a: integer;
  signal b: myvector;

  function value(vect : myvector) return integer is
    variable val: integer := 0;
  begin
    for i in 0 to 7 loop
      val := val + vect(i) * 2**i;
    end loop;
    return val;
  end function;

  begin
    b <= "00001111", "00100001" after 2 ns,
        "00111111" after 3 ns, "11111111" after 4 ns;

    a <= value(b);

  end base;

```

<b>Universität Duisburg - Essen</b>	<b>Dr.-Ing. R. Viga</b>	<b>Fak. f. Ing.-Wiss. EIT, EBS/VS</b>
<b>Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2</b>		<b>SEITE: 57</b>

*Sichtbarkeit von Vereinbarungen*

```

architecture arch of ent is
  type t is ...;
  signal s : t;

  procedure p1 (...) is
    variable v1 : t;
  begin
    v1 := s;
  end procedure p1;

begin -- arch

  proc1:
  process is
    variable v2 : t;

    procedure p2 (...) is
      variable v3 : t;
    begin
      p1 (v2, v3, ...);
    end procedure p2;

  begin -- proc1
    p2 (v2, ...);
  end process proc1;

  proc2:
  process is
    ....
  begin -- proc2
    p1 (...);
  end process p2;

end architecture arch;

```

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 58

### *Überdeckung von Vereinbarungen in Schachtelungen*

```
procedure p1 is
  variable v : integer;

  procedure p2 is
    variable v : integer;
  begin -- p2
    .....
    v := v + 1;
    .....
    p1.v := v + 33;    -- visibility by selection
    .....
  end procedure p2;

begin -- p1
  .....
  v := 2 * v;
  .....
end procedure p1;
```

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 59

**Programmbeispiel:  
Sichtbarkeit von Parametern; sequenzieller Prozeduraufruf**

```

entity visibility_1 is
end visibility_1;

architecture algo of visibility_1 is
begin

    visibility:
    process is
        variable A, B : integer := 200;
        variable Var_Out : integer;
        constant Fak : integer := 55;

        procedure InnerA (Ab, Ac : in Positive; Re : inout integer) is
            variable Var_Out : positive;
            variable Var_In : integer := 333;
            begin -- InnerA
                Var_Out := Ab * Ac + Re;
                Re := Var_Out + Var_In;
            end InnerA;

            function InnerB (Bb, Bc : in natural) return integer is
            -- procedure InnerB (Bb, Bc : in natural) is
            -- variable Var_Out : positive;
            -- variable A : positive;
            begin -- InnerB
                Var_Out := Bb * Bc;
                A := Bb + Bc;
                return A;
            end InnerB;

            -- variable Var_Out : integer;
            begin -- of process
                Var_Out := 30;
                InnerA (50, 70, Var_Out);
                -- InnerB (11 * 34, Fak); -- procedure call
                B := InnerB (11 * 34, Fak); -- function call
                wait;
            end process;
        end architecture algo;

```

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 60

### Nebenläufiger Prozeduraufruf

Vereinbarung und Aufruf einer Prozedur in der Architektur, z.B.

```
mypro(s1, s2, val1);
```

ist im Verhalten identisch zu dem des Prozesses mit sequenziellem Prozeduraufruf:

```
process is
begin
  mypro (s1, s2, val1);
  wait on s1, s2;
end process;
```

### Programmbeispiel (Rahmenvorgabe): Sichtbarkeit von Parametern; nebenläufiger Prozeduraufruf

```
entity visibility_2 is
end visibility_2;

architecture algo of visibility_2 is
  signal as, bs;

  procedure outer(signal as, bs: inout integer) is
    -- Deklarationen und Schachtelungen
    begin -- outer
      ....
    end procedure outer;

begin -- architecture
  outer(as, bs);
end architecture algo;
```

**Programmbeispiel:  
Zweiphasentakt mit nebenläufigen Prozeduraufrufen**

```

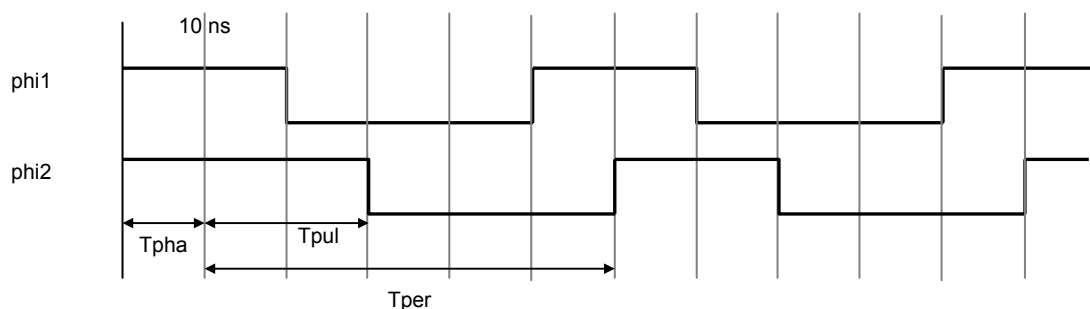
entity proc2clk is
end proc2clk;
architecture algo of proc2clk is
  signal phi1, phi2 : bit;

  procedure generate_clock(signal clk : out bit;
                           constant Tper, Tpul, Tpha: in time) is
  begin -- of procedure generate_clock
    wait for Tpha;
    loop
      clk <= '1', '0' after Tpul;
      wait for Tper;
    end loop;
  end generate_clock;

begin -- architecture
  gen_phi1: generate_clock(phi1, Tper => 50 ns, Tpul => 20 ns, Tpha => 0 ns);
  gen_phi2: generate_clock(phi2, Tper => 50 ns, Tpul => 20 ns, Tpha => 10 ns);
end architecture algo;

```

**Erzeugt Zweiphasentakt mit**  
**Tpha: Phasenverschub**  
**Tpul: Aktivzeit (Signal = '1')**  
**Tper: Periodendauer**



Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 62

## Package Konzept

### Package Vereinbarung zur Zusammenfassung von Modellteilen:

```

package_declaration ::=
  package identifier is
    { package_declarative_item }
  end [ package ] [ identifier ];

```

### Package Rumpf für Prozeduren und Funktionen, wenn solche in der Vereinbarung sind:

```

package_body ::=
  package body identifier is
    { package_body_declarative_item }
  end [ package ] [ identifier ];

```

Packages sind in Bibliotheken gespeichert.  
Standard Bibliotheken sind STD und WORK, die implizit referiert sind.  
Referieren von Bibliotheken mit der library Klausel, z.B. greift

```
library ieee;
```

auf die Bibliothek *ieee* zu.

Diese enthält unter anderem ein Package ***std\_logic\_1164***.  
In diesem Package ist u.a. ein Datentyp ***std\_logic*** definiert.

Ein Variablenvereinbarung könnte wie folgt sein:

```
variable carry: ieee.std_logic_1164.std_logic;
```

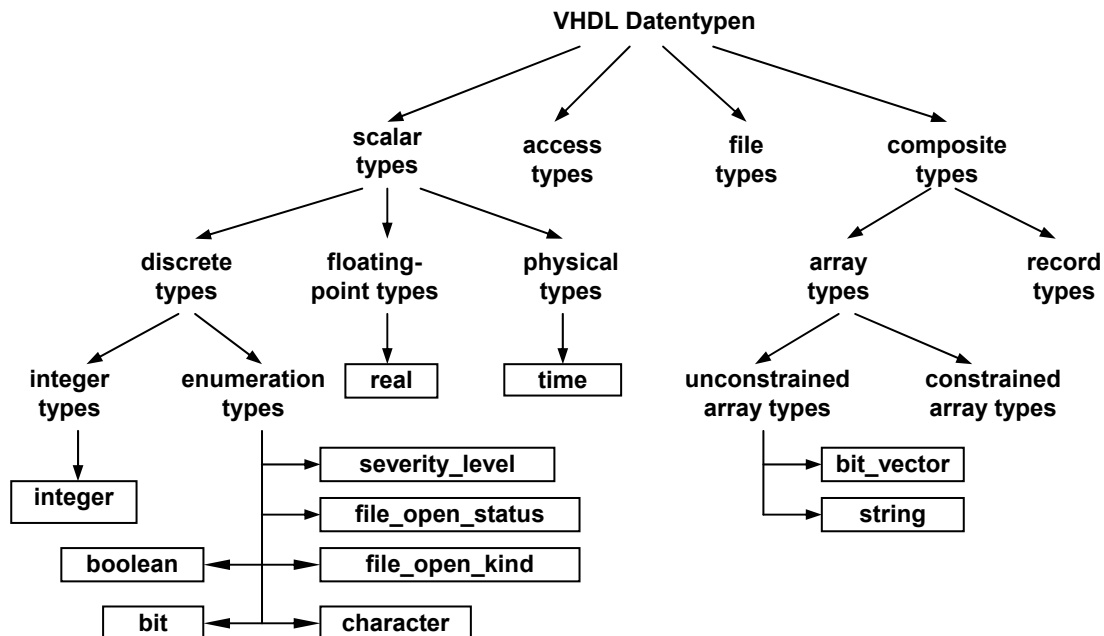
Zur Abkürzung der gesamten Referenz gibt es die ***use*** Klausel

```

library ieee;
use ieee.std_logic_1164.all;  -- use ieee.std_logic_1164.std_logic
.....
variable carry: std_logic;
.....

```

## Datentypen Übersicht



Alle eingerahmten Terminale sind im Standard-Package vordefiniert.

VHDL unterscheidet die Datenobjektklassen **constant**, **variable**, **signal** und **file**.

Entsprechend vereinbaren wir:

**constant** Name : *Typ\_Bezeichnung*; z.B. integer  
**variable** Name : *Typ\_Bezeichnung*; z.B. boolean  
**signal** Name : *Typ\_Bezeichnung*; z.B. bit\_vector(7 downto 0)

Die **file** Vereinbarung ist etwas anders (später).

### Eigene Typvereinbarungen:

Typ\_Vereinbarung ::= **type** Bezeichner **is** Typ\_Definition;

Ein Aufzählungstyp ist z.B.

**type** MeinTyp **is range** 1 **to** 100;

Eine Variablenvereinbarung von diesem Typ ist z.B.

**variable** MyVar : MeinTyp;

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 64

Das Schlüsselwort **range** bezeichnet eine begrenzte, ganzzahlige Aufzählung.

### Typenkonvertierung (Type Casting) an Beispielen

#### Typendefinition 1:

```
type MeinTyp is range 1 to 100;
type DeinTyp is range 3 to 95;
```

#### Variablendefinition 1:

```
VarA: MeinTyp
VarB: DeinTyp
```

#### Konvertierung bei der Zuweisung:

```
VarA:=MeinTyp (VarB)
```

#### Typdefinition 2:

```
type apples is range 0 to 100
type oranges is range 0 to 100
```

#### Variablendefinition 2:

```
A : oranges
B,C: apples
```

#### Gemischte Zuweisung:

```
C:= apples (A) + B
```

Trotz nominal gleicher Typendefinition von *apples* und *oranges* muss eine Typkonvertierung stattfinden.

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 65

## Verträgliche Untertypen (subtypes)

### Untertypenvereinbarung:

```
Subtyp_Vereinbarung ::= subtype Bezeichner is Subtyp_Indikator;  
Subtyp_Indikator ::= Basistyp [range Ausdruck (to | downto) Ausdruck]
```

### Alternative zu Typdefinition 1:

```
type MeinTyp is range 1 to 100;  
subtype DeinTyp is MeinTyp range 3 to 95;
```

Seien VarA und VarB definiert wie zuvor, dann ist für folgende Zuweisung eine Typkonvertierung nicht erforderlich:

```
VarA := VarB
```

Die Werteinhaltung wird von der Arbeitsumgebung kontrolliert. Bei Über-/Unterschreitung der Wertgrenzen erfolgt eine entsprechende Ausnahmebehandlung (exception Handling).

## Definition zusätzlicher Typen in Packages

### Beispiel einer Package-Definition

```
package int_types is  
    type byte is range 0 to 255;  
end package int_types;
```

### Nutzung des Package-Typs *byte*

```
use work.int-types.all  
entity small_adder is  
    port (a; b; in byte; s: out byte);  
end entity small_adder;
```

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 66

## Integer-Typen

### Wertebereich von Integer:

$$-(2^{31}-1) \leq x \leq + (2^{31} -1) \quad \text{entspricht 32-bit Darstellung}$$

### Wertbeschränkung (range constraint):

Integer\_Typdefinition ::= **range** Ausdruck (**to** | **downto**) Ausdruck

### Variable Wertgrenzen:

**constant** no\_of\_bits: integer:= 32;  
**type** bit\_index: **is range** 0 **to** no\_of\_bits -1;

### Operationen mit Integer\_Typen:

<b>+</b>	Addition	<b>-</b>	Subtraktion
<b>*</b>	Multiplikation	<b>/</b>	Division
<b>mod</b>	Modulo	<b>rem</b>	Restklasse
<b>abs</b>	Absolutbetrag	<b>**</b>	Potenzierung

### Unterscheidung von *mod* und *rem* an Beispielen:

A **rem** B = A – (A/B) \* B    Ersatzrechenvorschrift

$$\begin{array}{ll} 5 \text{ rem } 3 = 2 & (-5) \text{ rem } 3 = -2 \\ 5 \text{ rem } (-3) = 2 & (-5) \text{ rem } (-3) = -2 \end{array}$$

A **mod** B = A – B \* n    Ersatzrechenvorschrift

$$\begin{array}{ll} 5 \text{ mod } 3 = 2 & (-5) \text{ mod } 3 = 2 \\ 5 \text{ mod } (-3) = -2 & (-5) \text{ mod } (-3) = -2 \end{array}$$

Das Vorzeichen des Ergebnisses einer **rem**-Operation entspricht dem des Dividenten.

Das Vorzeichen des Ergebnisses einer **mod**-Operation entspricht dem des Moduls.

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 67

## Gleitkomma-Typen

### Wertebereich von real:

$-1,8 \cdot 10^{308} \leq x \leq +1,8 \cdot 10^{308}$       entspricht 64-bit Darstellung  
 mit 15-stelliger Dezimalgenauigkeit      (nach IEEE 754 oder IEEE 854)

### Wertbeschränkung (range constraint):

Gleitkomma-Typdefinition ::= range Ausdruck (**to** | **downto**) Ausdruck

Bsp.: **type** input\_level **is range** -10.0 **to** +10.0;  
**type** probability **is range** 0.0 **to** 1.0;

Initialwert von Variablen dieses Typs ist der zu Anfang stehende Wert der **range**-Deklaration.

## Physikalische Typen

In VHDL sind Zahlen mit physikalischer Dimension (z.B. Länge, Masse, Zeit, Widerstand, Spannung etc.) beschreibbar.

### Vereinbarung physikalischer Typen mit *units*

Physikalische\_Typdefinition ::=  
**range** Ausdruck (**to** | **downto**) Ausdruck  
**units**  
 Bezeichner;  
 {Bezeichner = Physikalisches\_Literal }  
**end units** [Bezeichner]  
 Physikalisches\_Literal ::= [Dezimales\_Literal | Basistyp\_Literal ] *units\_Name*

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 68

## Beispiele für units-Definitionen

### Widerstände:

```

type resistance is range 0 to 1E9
  units
    ohm;
  end units resistance;
  ⋮
variable R1, R2, R3: resistance;
  ⋮
R1:= 5 ohm; R2:= 22 ohm; R3:= 471_000 ohm;

```

### Längen (mit sekundären Einheiten):

```

type length is range 0 to 1E9
  units
    um;                -- primary unit: micron
    mm = 1000 um;     -- metric units
    m = 1000 mm;
    inch = 25400 um;  -- Englisch units
    foot = 12 inch;
  end units length;
  ⋮
variable M1, M2, M3: length;
  ⋮
M1:= 23mm; M2:= 2 foot; M3:= 9 inch;

-- auch Bruchzahlen möglich

M1:= 0.1 inch; M2:= 2.54 mm; M3:= 2,540526 mm;

```

Alle drei Variablen enthalten nach der Zuweisung den Wert *2540 um*. Bei M3 werden alle Stellen die *um*-Bruchteile ergeben abgeschnitten.

### Gemischte Verwendung:

```

5mm * 6 = 30 mm;           gemischt mit integer
18 kohm / 2.0 = 9 kohm;   gemischt mit real
39 mm / 22 um = 1500;     gleiche Typen
abs (-2 foot) = 2 foot;  als Argument in Ausdrücken

```

```

-- phystyp.vhd
-- physikalische Datentypen
entity phystyp is
end phystyp;

architecture algo of phystyp is
  type length is range 0 to 1E9
    units
      um; -- primary unit: micron
      mm = 1000 um; -- metric units
      cm = 10 mm;
      m = 1000 mm;
      inch = 25400 um; -- English units
      foot = 12 inch;
    end units length;

  type speed is range 0 to 1000
    units
      msec;
    end units speed;

  function veloc(s: in length; t: time) return speed is
    variable a, b, c: integer;
    -- variable c: real;
  begin
    a := s / 1 m;
    b := t / 1 sec;
    c := a / b;
    return c * 1 msec;
  end;

begin -- architecture

  process is
    variable M1, M2, M3: length := 0 m;
    variable X: integer;
    variable S: speed;
    variable T : time := 1 sec;
  begin
    M1 := 23 m; M2 := 200 foot; M3 := 9 inch;
    S := veloc(M2, T);

    wait;
  end process;
end algo;

```

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 70

## Beispiele für units-Definitionen

### Zeiten (Vordefiniert):

```

type time is range implementierungsabhängig
  units
    fs;           -- femtosecond
    ps = 1000 fs; -- picoseconds
    ns = 1000 ps; -- nanoseconds
    us = 1000 ns; -- microseconds
    ms = 1000 us; -- milliseconds
    sec = 1000 ms; -- seconds
    min = 60 sec; -- minutes
    hr = 60 min;  -- hours
  end units time;

```

## Aufzählungstypen

### Definition von Aufzählungstypen:

```

Aufzählung_Typdefinition ::=
( (Bezeichner | Character_Literal) [{,...} ] )

```

- Mindestens ein Element
- Element ist Bezeichner oder Buchstabe (character)

### Anwendungsbeispiele:

```

type alu_function is (disable, pass, add, subtract, multiply, divide);
type octal_digit is ('0', '1', '2', '3', '4', '5', '6', '7');
  ⋮
variable alu_op: alu_function;
variable last_digit: octal_digit:= '0';
  ⋮
alu_op:= subtract;
last_digit:= '7';

```

Elemente in unterschiedlichen Aufzählungen können gleiche Bezeichner haben (sog. Overloading).

```

Type logic_level is (unknown, low, undriven, high);
variable control: logic_level;
type water_level is (dangerously_low, low, ok);
variable water_sensor: water_level;
  ⋮
control:= low;
water_sensor:= low;

```

### Zeichentyp (character):

Character ist ein Aufzählungstyp von 256 ASCII-Zeichen des ISO 8859 Latin-1 Alphabets.

<b>type</b> character is	( nul,	soh,	stx,	etx,	eot,	enq,	ack,	bel,
	bs,	hat,	lf,	vt,	ff,	cr,	so,	si,
	dle,	dc1,	dc2,	dc3,	dc4,	nak,	syn,	etb,
	can,	em,	sub,	esc,	fsp,	gsp,	rsp,	usp,
	' ',	' ',	'"',	'#',	'\$',	'%',	'&',	'"',
	'(',	')',	'*',	'+',	':',	':',	':',	'/',
	'0',	'1',	'2',	'3',	'4',	'5',	'6',	'7',
	'8',	'9',	':',	':',	'<',	'=',	'>',	'?',
	'@',	'A',	'B',	'C',	'D',	'E',	'F',	'G',
	'H',	'I',	'J',	'K',	'L',	'M',	'N',	'O',
	'P',	'Q',	'R',	'S',	'T',	'U',	'V',	'W',
	'X',	'Y',	'Z',	'[',	'\',	']',	'^',	'_',
	'"',	'a',	'b',	'c',	'd',	'e',	'f',	'g',
	'h',	'i',	'j',	'k',	'l',	'm',	'n',	'o',
	'p',	'q',	'r',	's',	't',	'u',	'v',	'w',
	'x',	'y',	'z',	'{',	' ',	'}',	'~',	'del',
	c128,	c129,	c130,	c131,	c132,	c133,	c134,	c135,
	c136,	c137,	c138,	c139,	c140,	c141,	c142,	c143,
	c144,	c145,	c146,	c147,	c148,	c149,	c150,	c151,
	c152,	c153,	c154,	c155,	c156,	c157,	c158,	c159,
	' ',	'i',	'ç',	'£',	'¤',	'¥',	'¦',	'§',
	'"',	'©',	'ª',	'«',	'¬',	'¯',	'®',	'¸',
	'º',	'±',	'²',	'³',	'´',	'µ',	'¶',	'·',
	'¸',	'¹',	'º',	'»',	'¼',	'½',	'¾',	'¿',
	'À',	'Á',	'Â',	'Ã',	'Ä',	'Å',	'Æ',	'Ç',
	'È',	'É',	'Ê',	'Ë',	'Ì',	'Í',	'Î',	'Ï',
	'Ð',	'Ñ',	'Ò',	'Ó',	'Ô',	'Õ',	'Ö',	'×',
	'Ø',	'Ù',	'Ú',	'Û',	'Ü',	'Ý',	'Þ',	'ß',
	'à',	'á',	'â',	'ã',	'ä',	'å',	'æ',	'ç',
	'è',	'é',	'ê',	'ë',	'ì',	'í',	'î',	'ï',
	ð,	ñ,	ò,	ó,	ô,	õ,	ö,	÷,
	ø,	ù,	ú,	û,	ü,	ý,	þ,	ÿ);

- nul – usp : Steuerzeichen (nicht druckbar)
- ' ' – 'del' : reguläre druckbare Zeichen
- c128 – c159 : Zeichen ohne Bedeutung
- ' ' – 'ÿ' : Länderspezifische Sonderzeichen

### Anwendungsbeispiel:

```
variable cmd_char, terminator: character;
...
cmd_char:= 'p'; terminator:= cr;
```

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 72

### Boolean-Typ:

Boolean ist ein Aufzählungstyp mit zwei Bezeichnern.

**type** boolean **is** (false, true);

Boolean-Typ ist Grundlage für:

- Abfragen      "="    Gleichheit      "/="    Ungleichheit
- "<"    kleiner            "<="    kleiner oder gleich
- ">"    größer            ">="    größer oder gleich

- logische Verknüpfungen

**and, or, nand, nor, xor, not**

Bedingte Ausdrücke werden von links nach rechts evaluiert, Bsp.:

(b/=0) **and** (a/b > 1)

  |                    |  
false                x (wird nicht mehr evaluiert)

### Bit-Typ:

Bit ist ein Aufzählungstyp mit zwei Charactern.

**type** bit **is** ('0', '1');

Für Bit-Typ gelten gleiche Operationen wie beim Boolean-Typ, Bsp.:

'0' **and** '1' = '0',      '1' **xor** '1' = '0'

### Standard-Logik-Typ:

Standard-Logik ist ein Aufzählungstyp mit 9 Charactern für das 9-wertige Logikmodell nach IEEE-1164 Standard.

Zur Nutzung müssen die Bibliothek **ieee** und das Package **std\_logic\_1164** referenziert werden.

```
type std_ulogic is ('U', -- uninitialized; nicht initialisiertes Signal
                   'X', -- forcing unknown; mehr als ein Treiber an einem Signal
                   '0', -- forcing zero; wie bittyp '0'
                   '1', -- forcing one; wie bittyp '1'
                   'Z', -- high impedance; Tristate
                   'W', -- weak unknown; Konflikt zwischen 'L' und 'H'
                   'L', -- weak zero; offener Ausgang nach '0'
                   'H', -- weak one; offener Ausgang nach '1'
                   '-'); -- don't care; frei wählbares Signal
```



Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 74

### Beispieltyp tristate\_logic:

```

entity resolute is
end resolute;

architecture basic of resolute is
  type tristate_logic is ('0', '1', 'Z', 'X');
  type tristate_logic_array is array (integer range 0 to 2) of tristate_logic;

  function myres(inp: tristate_logic_array) return tristate_logic is
    variable x: tristate_logic := 'Z';
  begin
    for i in inp'range loop
      if x /= inp(i) then -- wenn gleich keine Änderung x
        if x = 'Z' then -- wenn x = 'Z' übernehme jeden Wert
          x := inp(i);
        elsif x /= inp(i) then
          x := 'X';
          exit;
        else x := inp(i);
        end if;
      end if;
    end loop;
    return x;
  end function;

  signal d : myres tristate_logic;

begin -- architecture
  d <= '0', 'Z' after 10 ns;
  d <= '0', '0' after 5 ns;
  d <= '0', '1' after 5 ns;

end basic;

```

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 75

## Attribute

Attribute kennzeichnen Eigenschaften von Variablen, Typen, Feldern etc.

### Für Skalare Datentypen:

<b>T'left</b>	erstes (linkes) Element von T
<b>T'right</b>	letztes (rechtes) Element von T
<b>T'low</b>	kleinster Wert von T
<b>T'high</b>	größter Wert von T
<b>T'ascending</b>	<b>true</b> , wenn T in aufsteigender Folge, <b>false</b> sonst

### Für Elemente skalarer Datentypen:

<b>T'image(x)</b>	ein String, der den Variablenwert <b>x</b> vom Typ T repräsentiert
<b>T'value(s)</b>	ein Wert vom Typ T, der von einem String <b>s</b> repräsentiert wird

### Für diskrete und physikalische skalare Typen:

<b>T'pred(x)</b>	Vorgänger von <b>x</b>
<b>T'succ(x)</b>	Nachfolger von <b>x</b>
<b>T'leftof(x)</b>	Vorgänger von <b>x</b>
<b>T'rightof(x)</b>	Nachfolger von <b>x</b>
<b>T'pos(s)</b>	die Positionnummer von <b>x</b> in der Aufzählung
<b>T'val(x)</b>	das Aufzählungselement auf der Position <b>x</b>

### Anwendungsbeispiel:

```
type resistance is range 0 to 1E9
units
  ohm;
  kohm = 1_000 ohm;
  Mohm = 1_000 kohm;
end units resistance;
```

```
type set_index_range is range 21 downto 11;
type logic_level is (unknown, low, undriven, high);
```

### für diesen Typen gilt:

resistance'left = 0 ohm	resistance'right = 1E9 ohm
resistance'low = 0 ohm	resistance'high = 1e9 ohm
resistance'ascending = true	
resistance'image(2 kohm) = "2000 ohm"	resistance'value("5 Mohm") = 5000000 ohm
set_index_range'left = 21	set_index_range'right = 11
set_index_range'low = 11	set_index_range'high = 21
set_index_range'ascending = false	
set_index_range'image(14) = "14"	set_index_range'value("20") = 20
logic_level'left = unknown	logic_level'right = high
logic_level'low = unknown	logic_level'high = high
logic_level'ascending = true	
logic_level'image(undriven) = "undriven"	logic_level'value("Low") = low

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 76

## Filetypen

### Filetyp-Deklaration:

```
Filetyp_Definition ::= file of Objekt_Typ
type integer_file is file of integer;
```

### File-Objekt-Deklaration:

```
File_Vereinbarung ::=
file Bezeichner {...}: Filetyp [[ open file_open_kind_Ausdruck] is Filename];
```

### Vordefinierte Aufzählungstypen für File:

```
type file_open_kind is (read_mode, write_mode, append_mode);
type file_open_status is (open_ok, status_error, failure);
```

## Kompositorische Datentypen

### Felder:

Felder sind eine Anreihung von Objekten gleichen Typs, wobei jedes Objekt eine  feste Position besitzt und über einen Index bezeichnet wird.

```
Feldtyp_Definition ::= array (diskreter_Bereich {...}) of Objekt_Typ
```

### Beispiele für Feldanwendungen:

```
type word is array (0 to 31) of bit;           -- big-endian-Format
type word is array (31 downto 0) of bit;       -- little-endian-Format
```

```
type controller_state is (initial, idle, active, error);
type state_counts is array (idle to error) of natural;
```

```
subtype coeff_ram_address is Integer range 0 to 63;
type coeff_array is array (coeff_ram_address) of real;
```

```
variable buffer_register, date_register: word;
variable counters: state_counts;
variable coeff: coeff_array;
```

### Mögliche Zuweisungen:

```
coeff (0) := 0.0;           -- Real-Wert an ersten Index
counters (active) := counters (active) +1;
data_register := buffer_register; -- Zuweisung ganzer Felder
```

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 77

### Mehrdimensionale Felder

```

type symbol is ('a', 't', 'd', 'h', digit, cr, error);
type state is range 0 to 6;
type transition_matrix is array (state, symbol) of state;
⋮
variable transition_table: transition_matrix;
⋮
transition-table (5, 'd') := 3;

```

### Aggregate

Die Festlegung von Initialwerten für Felder wird als Aggregat bezeichnet.

#### Bei eindimensionalen Feldern:

```

type point is array (1 to 3) of real;
constant origin: point := (0.0, 0.0, 0.0);
variable view_point: point := (10.0, 20.0, 0.0);

```

#### Bei mehrdimensionalen Feldern:

```

A2: array (1..3, 1..3) of float := ((1.0, 0.0, 0.0), (0.0, 0.0, 0.0), (0.0, 0.0, 0.0));

```

oder alternativ abkürzend:

```

A2: array (1..3, 1..3) of float := ((1.0, others => 0.0), (others => 0.0), (others => 0.0));

```

Beide Beispiele nutzen direkte Feld-Vereinbarung ohne vorherige Typ-Definition.

#### Auswahl von Feldobjekten über Index-Namen:

```

Type symbol is ('a', 't', 'd', 'h', digit, cr, error);
type state is range 0 to 6;
type transition_matrix is array (state, symbol) of state;
constant next_state : transition_matrix :=
  ( 0 => ('a' => 1, others => 6),
    1 => ('t' => 2, others => 6),
    2 => ('d' => 1, 'h' => 5, others => 6),
    3 => (digit => 4, others => 6),
    4 => (digit => 4, cr => 0, others => 6),
    5 => (cr => 0, others => 6),
    6 => (cr => 0, others => 6);

```

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 78

## Feld-Attribute

### Nutzbare Feld-Attribute eines Feldes **A** mit Dimension **N**:

<b>A'left(N)</b>	linke Komponente von <b>A</b> der Dimension <b>N</b>
<b>A'right(N)</b>	rechte Komponente von <b>A</b> der Dimension <b>N</b>
<b>A'low(N)</b>	kleinste Komponente von <b>A</b> der Dimension <b>N</b>
<b>A'high(N)</b>	größte Komponente von <b>A</b> der Dimension <b>N</b>
<b>A'range(N)</b>	Index Bereich von <b>A</b> der Dimension <b>N</b>
<b>A'reverse_range(N)</b>	umgekehrter Index Bereich von <b>A</b> der Dimension <b>N</b>
<b>A'length(N)</b>	Länge des Index Bereichs von <b>A</b> der Dimension <b>N</b>
<b>A'ascending(N)</b>	<b>true</b> , wenn Index Bereich von <b>A</b> der Dimension <b>N</b> in aufsteigender Folge, <b>false</b> sonst

Bei eindimensionalen Feldern kann die Angabe der Dimension **N** entfallen.

### Anwendungsbeispiele von Feld-Attributen:

**type A is array (1 to 4, 31 downto 0) of boolean;**

<b>A'left(1) = 1</b>	<b>A'low(1) = 1</b>
<b>A'right(2) = 0</b>	<b>A'high(2) = 31</b>
<b>A'range(1) ist 1 to 4</b>	<b>A'reverse_range(2) ist 0 to 31</b>
<b>A'length(1) = 4</b>	<b>A'length(2) = 32</b>
<b>A'ascending(1) = true</b>	<b>A'ascending(2) = false</b>

## Eingeschränkte und uneingeschränkte Feldtypen

VHDL erlaubt die Definition von Feldtypen mit unbegrenzter Größe (unconstrained). Die Größeneinschränkung wird nachträglich

- implizit durch Anführen eines Aggregates oder
- explizit durch Untertypen der definierten Index-Typen erreicht.

### Anwendungsbeispiele uneingeschränkter Feldtypen:

```
type sample is array (natural range <>) of integer;
variable short_sample: sample(0 to 63);           -- explizit
constant x_sample: sample := (1=>0, 3=>4, 2=>7, 4=>2);  -- implizit
```

Universität Duisburg - Essen	Dr.-Ing. R. Viga	Fak. f. Ing.-Wiss. EIT, EBS/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 79

## Strings

Zeichen folgen werden als Strings bezeichnet und stellen einen uneingeschränkten Feldtyp dar.

**type string is array** (positive **range** <>) **of** character;

### Anwendung von Strings:

**constant** LCD\_length: positive := 20;

**subtype** LCD\_string **is** string(1 **to** LCD\_length);

LCD\_string:= "Hello World";

## Bitvektoren

Zeichenfolgen aus dem Zeichenvorrat '0', '1' werden als Bitvektoren bezeichnet und stellen einen uneingeschränkten Feldtyp dar.

**type** bit\_vector **is array** (natural **range** <>) **of** bit;

### Anwendung von Bitvektoren:

**subtype** byte **is** bit\_vector(7 **downto** 0); -- Stellenbeschränkung auf 8 Bit