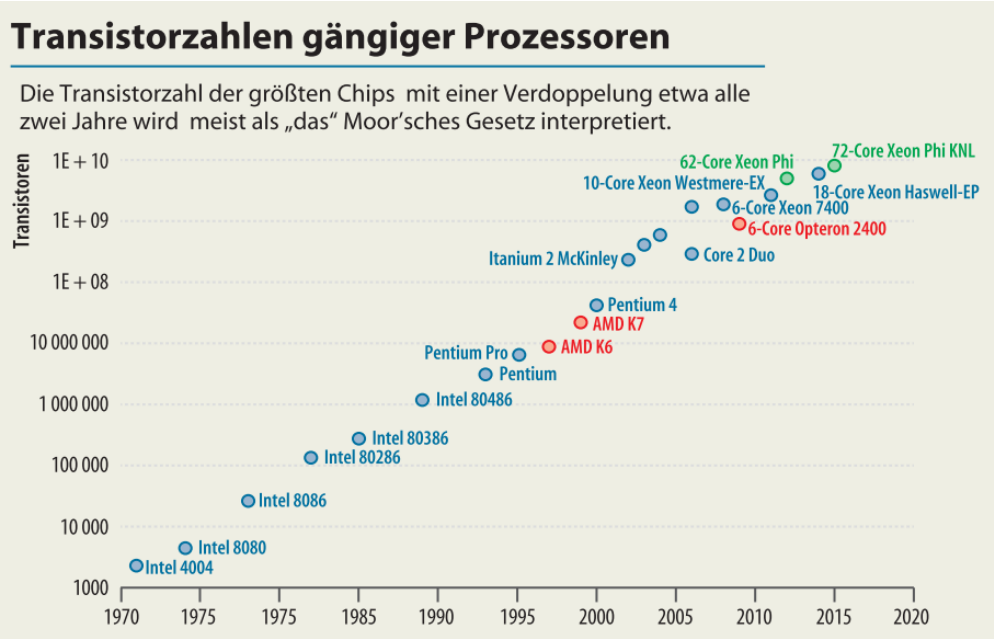
OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

Grundlagen VHDL für Kombinatorische Logik (VHDLcomb)

Motivation - Entwicklung der Chipkomplexität (Moore's Law)



- **Designer Produktivität**
  1970: Transistor Layout (Polygone); 1980: Transistor Schaltplan / Boolsche Gleichungen; 1990: Gatter Schaltplan; 2000: Hardwarebeschreibung

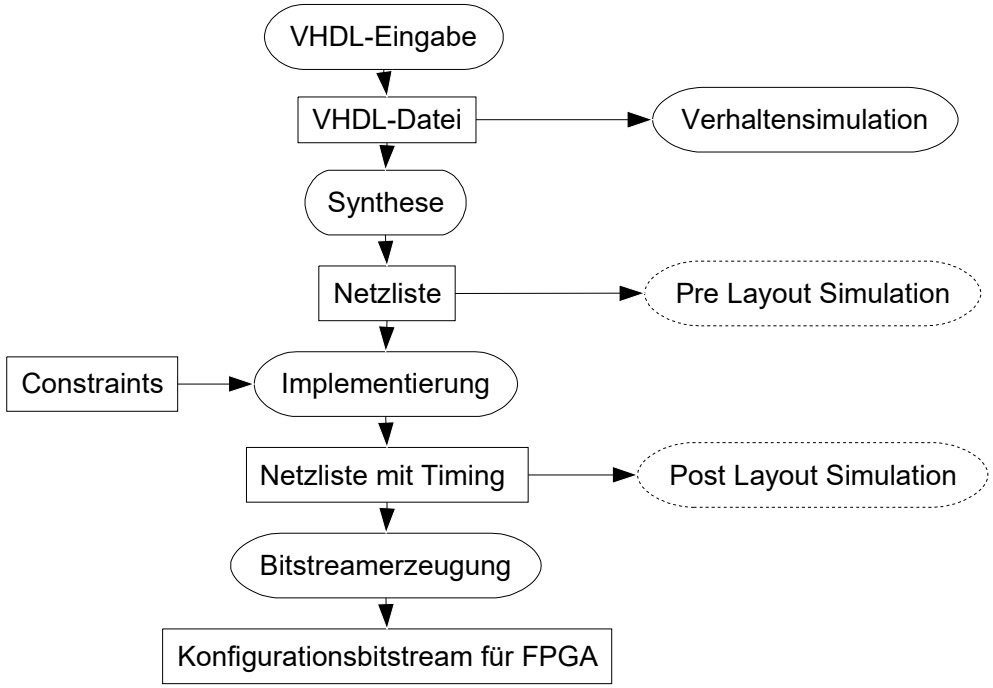
Motivation - Warum Hardwarebeschreibungssprachen?

- Vorteile von Hardwarebeschreibung
  - höhere Abstraktion -> kürzere Entwicklungszeiten/ geringere Kosten
  - einheitliche Modellierung für Spezifikation, Simulation und Synthese -> Reuse

- zwei etablierte Sprachen:
  - Verilog (Verbreitung: mehr USA)
  - VHDL (Verbreitung: mehr Europa; Verwendung in USA zunehmend)
    - Very High Speed Integrated Circuit Hardware Description Language
    - standardisiert (Simulation IEEE 1076-1987, IEEE 1076-1993, IEEE 1076-2008;
      Synthese IEEE 1076-1999 basierend auf IEEE 1076-1987(VHDL87))

- Konzentration auf VHDL (in dieser Veranstaltung)

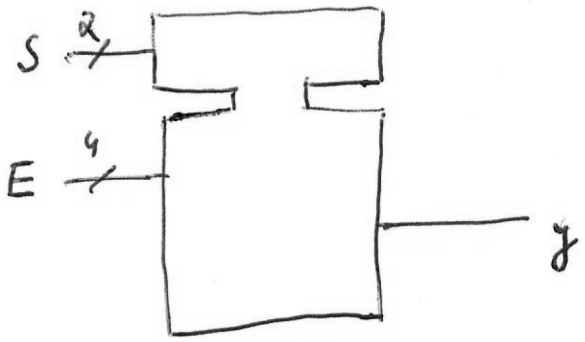
Motivation - VLSI Designflow (Beispiel FPGA)

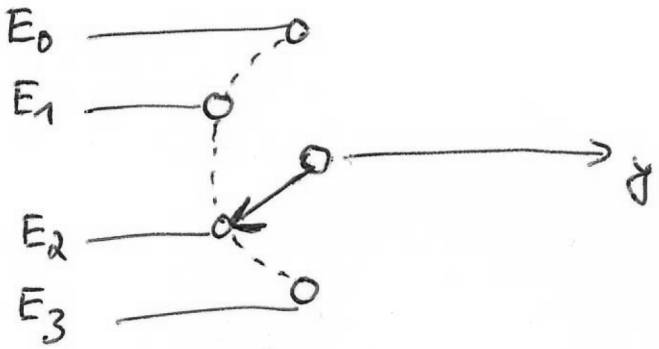


VLSI= Very Large Scale Integration

VHDL-Grundlagen
Enitity/Architecture

- Aufbau einer VHDL-Datei (*.vhd) :
  - **entity**
    - Analogie Chip: Chip-Gehäuse / Schnittstelle nach außen
    - Analogie Platine: Steckverbindungen / Schnittstelle nach außen
  - **architecture**
    - Analogie Chip: Innenleben / Funktionalität
    - Analogie Platine: Aufbau der Platine
- Beispiel: Multiplexer





VHDL-Grundlagen - Entity/Architecture

- Allg. Syntax Entity/Architecture

```
entity <entityname> is
  [generic <Parameter-Dekl.>]
  port(<Dekl. der I/Os>);
end <entityname>;

architecture <archname> of <entityname> is
  [<Architecture-Deklarationen>]
begin
 {<VHDL-Anweisungen>;}
end <archname>;



<> = Platzhalter für Bezeichner
[] = optional
{} = beliebig oft
|  = alternativ
```

- Beispiel Multiplexer

```
entity MUX4 is
  port( S: in bit_vector (1 downto 0);
        E: in bit_vector (3 downto 0);
        Y: out bit);
end MUX4;

architecture BEHAV of MUX4 is
-- comment
begin
  with S select
  Y<=  E(0) when "00",
       E(1) when "01",
       E(2) when "10",
       E(3) when "11";
end BEHAV;
```

VHDL-Grundlagen - Konventionen

- VHDL-Dateien sollten denselben Namen wie die entity tragen
- VHDL-Code ist generell nicht ”case sensitive“
- Bezeichner müssen mit Buchstaben beginnen
  - nachfolgende Zeichen können auch „Zahlen“ und „_“ sein
- Signalzuweisung erfolgt mit Operator „<= “
  - Typ bit erfolgt in Klammerung mit Apostroph ‘’: X<=‘0‘;
  - Typ bit_vector erfolgt in Anführungszeichen “”: X<=“0101“;
  - Zuweisung des gleichen Werts an alle Bitstellen eines Busses: X<=(others=>‘0‘);
- VHDL-Anweisungen werden mit „;“ abgeschlossen
- Kommentare sind Zeilenkommentare und beginnen mit „--“

VHDL-Grundlagen - Konventionen

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

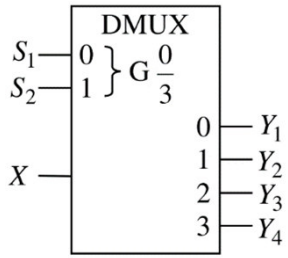
VHDL-Grundlagen - Portdekl. (N)

- Portmodi:
  - in
    - Eingangssignal; nur auf der rechten Seite einer Signal-zuweisung oder in einer Abfrage
  - out
    - Ausgangssignal; nur auf der linken Seite einer Signalzu-weisung stehen
  - buffer
    - sowohl auf der rechten als auch auf der linken Seite einer Signalzuweisung;
      Statt buffer sollte aber lieber ein internes Signal verwendet werden, dass dann dem Ausgang zugewiesen wird !
  - inout
    - bidirektionales Signal (in Verbindung mit Datentyp std_logic)

```
entity MUX4 is
  port( S: in bit_vector (1 downto 0);
        E: in bit_vector (3 downto 0);
        Y: out bit);
end MUX4;
architecture BEHAV of MUX4 is
-- comment
begin
  with S select
  Y<=  E(0) when "00",
       E(1) when "01",
       E(2) when "10",
       E(3) when "11";
end BEHAV;
```

VHDL-Grundlagen - Portdeklarationen - Übung

- Beschreiben Sie die entity folgendes Schaltsymbols in VHDL
- Verwenden Sie für S und Y Busse



VHDL-Grundlagen - Selektive und bedingte Signalzuweisung

- Allg. Syntax
- selektiv

```
[<label_1>:]
with <steuersig> select
  <signal> <= <sigwert_1> when <Bedingung_1>,
              <sigwert_2> when <Bedingung_2>,
              ...
              [<sigwert_n> when others];
```

Anmerkung:
Es müssen alle möglichen Werte des <steuersig> in expliziten when-Zweigen berücksichtigt werden (alterantiv auch „when others"-Zweig)

- bedingt

```
[<label_2>:]
<signal> <= <sigwert_1> when <Bedingung_1> else
            [<sigwert_2> when <Bedingung_2> else]
              ...
            <sigwert_n>;
```

- Beispiel Multiplexer

```
entity MUX4 is
  port( S: in bit_vector (1 downto 0);
        E: in bit_vector (3 downto 0);
        Y: out bit);
end MUX4;
```

- selektiv

```
architecture BEHAV of MUX4 is
begin
  with S select
  Y<=  E(0) when "00",
       E(1) when "01",
       E(2) when "10",
       E(3) when "11";
end BEHAV;
```

- bedingt

```
architecture BEHAV of MUX4 is
begin
  Y<=  E(0) when S="00"else
       E(1) when S="01"else
       E(2) when S="10"else
       E(3);
end BEHAV;
```

VHDL-Grundlagen - Selektive und bedingte Signalzuweisung - Übung

- Modellieren Sie einen Codeumwandler, der 3bit-Binärzahlen in Gray-Code wandelt
- Verwenden Sie keine Operatoren sondern setzen Sie die Wertetabelle mit „with select“ um

Nr	x3	x2	x1	y3	y2	y1
0	0	0	0	0	0	0
1	0	0	1	0	0	1
2	0	1	0	0	1	1
3	0	1	1	0	1	0
4	1	0	0	1	1	0
5	1	0	1	1	1	1
6	1	1	0	1	0	1
7	1	1	1	1	0	0

```
entity BIN2GRAY is
  port(
    BIN: in bit_vector(2 downto 0);
    GRAY: out bit_vector(2 downto 0)
);
end BIN2GRAY;
--…
```

VHDL-Grundlagen - Lokale Signale

- Vorstellung:
  - Analogie Chip/Platine: Signal=Leitung
- Zweck:
  - lokale Signale dienen zum Informationsaustausch innerhalb einer „architecture“

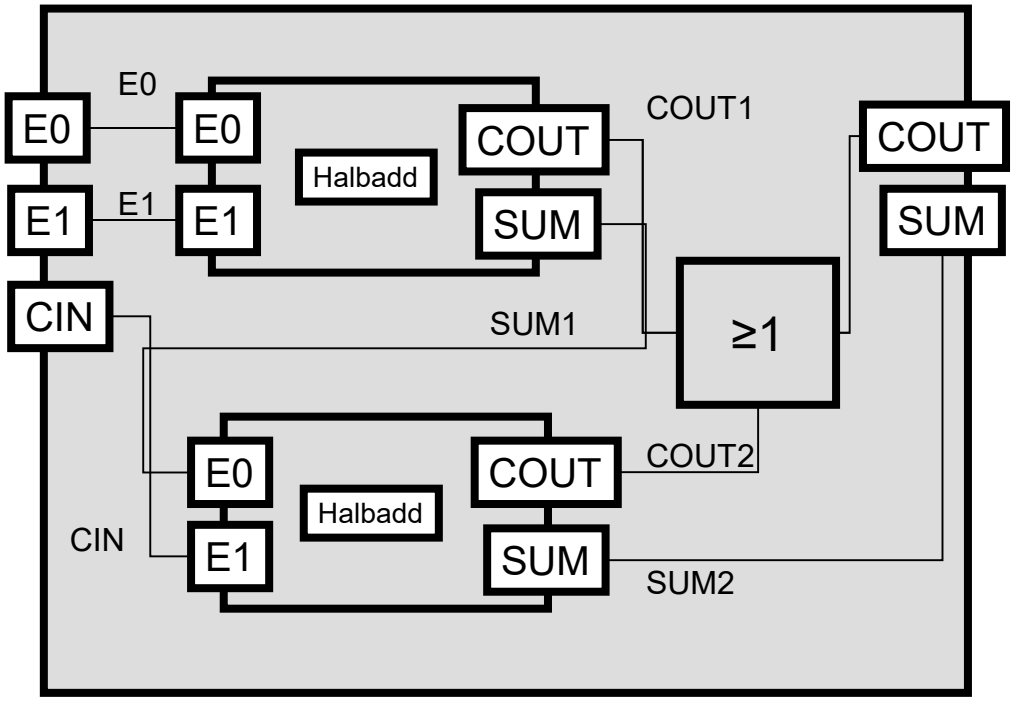
- Syntax:
  - Deklaration innerhalb „architecure“ **vor** „**begin**“ mit **Keyword** „**signal**“

```
architecture ... of ... is
signal A: bit;
signal B: bit_vector (3 downto 0);
signal C: bit_vector (2 downto 0):=”000“; --default value
...
begin
...
end ...;
```

  - Achtung! Nur eine Quelle darf einem Signal einen Wert zuweisen!

VHDL-Grundlagen - Struktur-Modellierung (hierarchische Modellierung)

- Beispiel: Volladdierer aus zwei Halbaddierern

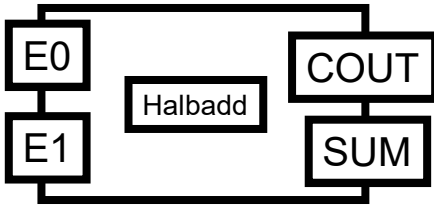



```
Halbadd.(wiederh.)
E0 E1 COUT SUM
0  0   0    0
0  1   0    1
1  0   0    1
1  1   1    0

-> SUM <= E0 xor E1;
-> COUT <= E0 and E1;


 Volladd.
 =2 kaskadierte Halbadd.
```

VHDL-Grundlagen - Struktur-Modellierung (hierarchische Modellierung)

Übung

- Beschreiben Sie einen Halbaddierer mit VHDL (halfadd.vhd)
- …




```
Halbadd.(wiederh.)
E0  E1  COUT  SUM
0   0    0    0
0   1    0    1
1   0    0    1
1   1    1    0

-> SUM <= E0 xor E1;
-> COUT <= E0 and E1;
```

OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG

VHDL-Grundlagen - Struktur-Modellierung (hierarchische Modellierung)

- Allg. Syntax

```
--in architecture vor begin (Deklaration)
...
component <compentityname>
  [generic (<Dekl. v. Parametern>)];
  port(<Dekl. der I/Os>);
end component;
--in architecture nach begin (Instanzierung)
...
<label>: <compentityname>
  [generic map (<Parameterzuord.>)];
  port map(<Signalzuord.>); --(*)
end component;

(*) 2 Varianten für die Signalzuordnung:
1. port map (<componentsig_1> => <toplevelsig_1>,
          ...,
          <componentsig_n> => <toplevelsig_n>);
2. port map (<toplevelsig_1>, ...,
             <toplevelsign>);

!! nicht angeschlossene Signale:
Inputsignal: <insigname> => (others=>'0')
Outputsignal: <outsigname> => open
```

- Beispiel Volladdierer

```
--1. Variante:
entity FULLADD is
  port( E0, E1, CIN: in bit;
        SUM, COUT: out bit);
end FULLADD;

architecture STRUCT of FULLADD is
signal SUM1, SUM2, COUT1, COUT2: bit;
component HALFADD
  port( E0, E1: in bit;
        SUM, COUT: out bit);
end component;
begin
U1: HALFADD port map(E0 => E0, E1 => E1, SUM =>
            SUM1, COUT => COUT1);
U2: HALFADD port map(E0 => SUM1, E1 => CIN,
            SUM => SUM2, COUT => COUT2);
    COUT <= COUT1 or COUT2;
    SUM <= SUM2;
end STRUCT;

--2. Variante:
...
U1: HALFADD port map(E0, E1, SUM1, COUT1);
U2: HALFADD port map(SUM1, CIN, SUM2, COUT2);
...
```

VHDL-Grundlagen - Wie testet man eine digitale Schaltung?

- Früher: Signalgenerator + Logikanalyser/ Oszilloskop
  - Erzeugen Eingangssignale; Vergleich der Ausgangssignale mit Erwartung
- Heute: TESTBENCH = VHDL-Modul, das Stimuli für Test/Simulation dieser Hardwarebeschreibung erzeugt
  - Erzeugen Eingangssignale; Vergleich der Ausgangssignale mit Erwartung
- Syntax:

```
entity TB is
end TB;

architecture TESTBENCH of TB is
component <componentname> --Entity-Name des zu testenden Modul
          <Port-Liste> --IDENTISCH zur Port-Entity des zu testenden Moduls
end component;
signal <Signalliste>; --noch einmal alle Component-Signale
begin
  [<label>]: <componentname> port map(<sigzuord>);
  --Stimuli-Signalzuweisungen
end TESTBENCH;
```

OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG

VHDL-Grundlagen - Einfache Testbenches: Beispiel Multiplexer

- Testbench

```
entity TB_MUX4 is
end TB_MUX4;

architecture TESTBENCH of TB_MUX4 is
  component MUX4
    port( S: in bit_vector (1 downto 0);
          E: in bit_vector (3 downto 0);
          Y: out bit);
  end component;
  signal TB_S: bit_vector (1 downto 0);
  signal TB_E: bit_vector (3 downto 0);
  signal TB_Y: bit;

begin
  DUT: MUX4 port map(S=>TB_S, E=>TB_E,
                     Y=>TB_Y);

  TB_E<="0101"; -- static input
  TB_S<="00", "01" after 10 ns,
        "10" after 20 ns, "11" after 30 ns;
end TESTBENCH;
```

- Hardwarebeschreibung

```
entity MUX4 is
  port( S: in bit_vector (1 downto 0);
        E: in bit_vector (3 downto 0);
        Y: out bit);
end MUX4;

architecture BEHAV of MUX4 is
begin
  with S select
  Y<=  E(0) when "00",
       E(1) when "01",
       E(2) when "10",
       E(3) when "11";
end BEHAV;
```

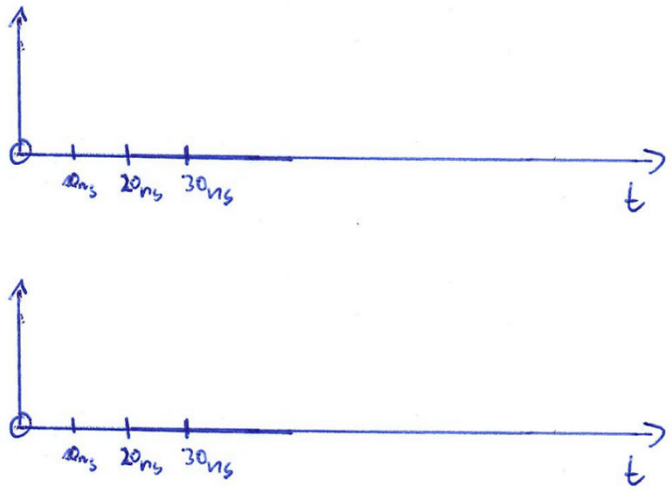
Anmerkungen

1. **Keyword „after"** darf nur im Testbench verwendet werden **!! (NICHT SYNTHETISIERBAR)**
2. zwischen Zeitwert und Zeiteinheit **MUSS** Leerzeichen sein !!

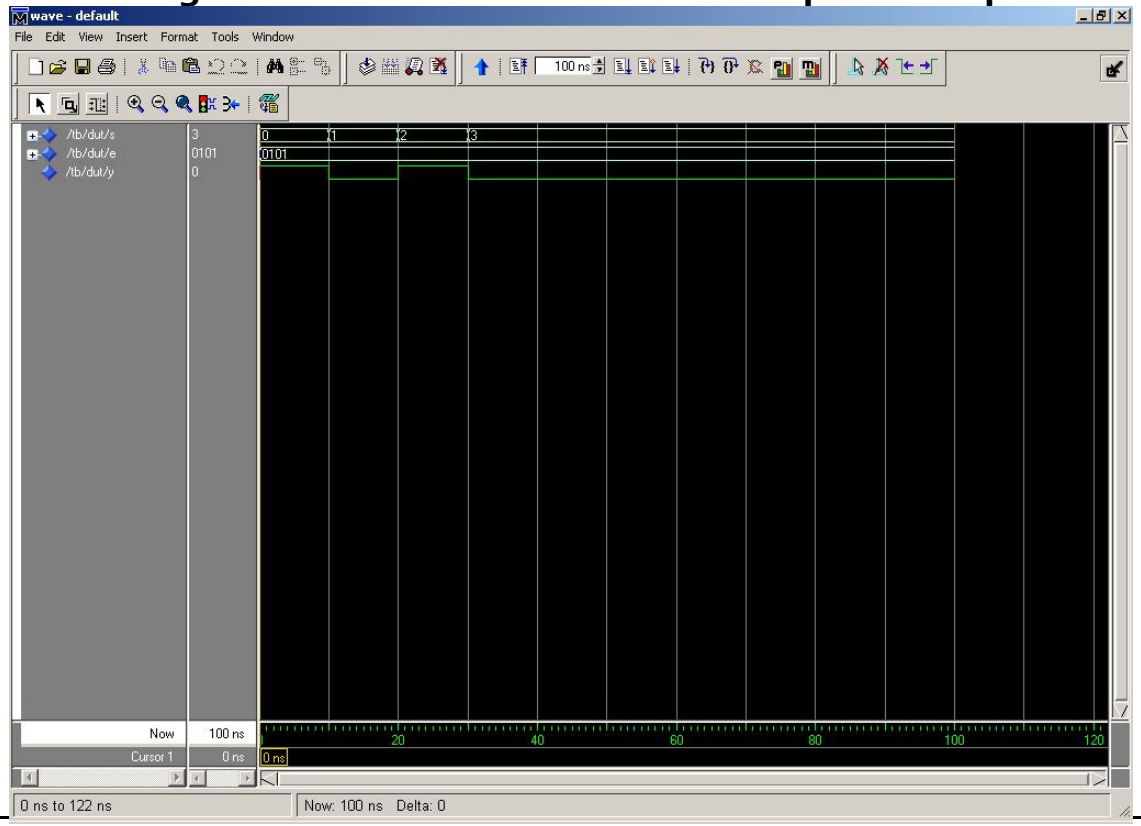
VHDL-Grundlagen - Einfache Testbenches - Intrepretation - Übung

- Skizzieren Sie die Resultierende Waveform des Testbenches

```
TB_E<="0101";
TB_S<="00", "01" after 10 ns,
            "10" after 20 ns,
            "11" after 30 ns;
```



VHDL-Grundlagen - Einfache Testbenches: Beispiel Multiplexer



Teil 2

VHDL-Grundlagen - Auswahl wichtiger Datentypen

Datentyp	Zuläsige Werte	Bemerkung
bit, bit_vector	'0', '1'	Initialisierung '0'
std_ulogic, std_ulogic_vector	U', 'X', '0','1','Z', 'w', 'L', 'H', '-'	Initialisierung 'U'
std_logic, std_logic_vector	U', 'X', '0','1','Z', 'w', 'L', 'H', '-'	Initialisierung 'U'
integer	-21478364 bis 2147483647	
natural	0 bis 2147483647	
positive	1 bis 2147483647	

```
ERLÄUTERUNG DER STD(U)_LOGIC WERTE
U    = Simulator: unbekannt, nicht initialisiert?
X    = Simulator: Buskonflikt 0, 1
0    = Logikzustand 0
1    = Logikzustand 1
Z    = high Impedanz (Z); Tri-State
W    = Simulator Buskonflikt L, H
L    = 0 bei „Open Emitter“
H    = 1 Bei „Open Collector“
'-' = don’t care
```

VHDL-Grundlagen - Spezielle Hinweise für std_(u)logic

- Unterschied zw. std_ulogic und std_logic:
  - beide können tri-state Zustände modellieren
  - std_logic kann Buskonflikte lösen; (u = unresolved)
  - aber: versehentliche Mehrfachzuweisungen sind nicht mehr erkennbar
- std_(u)logic zwingt den Entwickler zur sauberen Initialisierung

  std_(u)logic benötigt die Einbindung einer speziellen Library:

```
library ieee;
use ieee.std_logic_1164.all; --use ieee.std_ulogic_1164.all;

entity <entityname> is
...
```

- std_(u)logic_vector und bit_vector sind ineinander konvertierbar:

```
 --Beispiel:
signal a: bit_vector(1 downto 0):="10";
signal s: std_logic_vector (1 downto 0):="01";
s<= to_stdlogicvector(a);
a<= to_bitvector(s);
```

VHDL-Grundlagen - Beispiel Multiplexer in std_logic

- std_logic

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

entity MUX4 is
  port( S: in std_logic_vector (1 downto 0);
        E: in std_logic_vector (3 downto 0);
        Y: out std_logic);
end MUX4;

architecture BEHAV of MUX4 is
begin
  with S select
  Y<=  E(0) when "00",
       E(1) when "01",
       E(2) when "10",
       E(3) when others; --for all other
                         --combinations!!!
end BEHAV;
```

- bit_vector

```
entity MUX4 is
  port( S: in bit_vector (1 downto 0);
        E: in bit_vector (3 downto 0);
        Y: out bit);
end MUX4;

architecture BEHAV of MUX4 is
begin
  with S select
  Y<=  E(0) when "00",
       E(1) when "01",
       E(2) when "10",
       E(3) when "11";
end BEHAV;
```

VHDL-Grundlagen - std logic - Übung

- Modellieren Sie einen BIN-to-Gray-Code-Umwandler mit std-logic
- Verwenden Sie keine Operatoren sondern setzen Sie die Wertetabelle mit „with select“ um
- Verwenden Sie einen 3bit Eingangsbus names BIN und einen 3bit Ausgangsbus names GRAY
- Schreiben Sie zweispaltig für gute Platzausnutzung

Nr	x3	x2	x1	y		
0	0	0	0	0	0	0
1	0	0	1	0	0	1
2	0	1	0	0	1	1
3	0	1	1	0	1	0
4	1	0	0	1	1	0
5	1	0	1	1	1	1
6	1	1	0	1	0	1
7	1	1	1	1	0	0

VHDL-Grundlagen - Operatoren I/V

- 1. Logische Operatoren:
  - and, or, nand, nor, xor, xnor, not
- 2. Vergleichs-Operatoren:
  - =, /=, <=, >=, <, >
- 3. Shift-Operatoren:
  - sll, srl, sla, sra, rol, ror
- 4. Arithmetische Operatoren:
  - +, -, *, /, mod, rem, a**b(exponent), abs

VHDL-Grundlagen - Operatoren II/V

- Bibliothek std (automatisch eingebunden!!!):

-bit: and, or, nand, nor, xor, xnor, not, =, /=, <, <= , >, >=

-bit_vector: and, or, nand, nor, xor, xnor, not, sll, srl, sla, sra, rol, ror, =, /=, <, <= , >, >=

-integer: =, /=, <, <= >, >=, +, -, *, / , mod, rem, **, abs

-real: =, /=, <, <= >, >=, +, -, *, / , **, abs

- Bibliothek numeric_bit:

-bit: and, or, nand, nor, xor, xnor, not, =, /=, <, <= , >, >=, +, -

-unsigned/signed: and, or, nand, nor, xor, xnor, not, =, /=, <, <= , >, >=, sll, srl, sla, sra, rol, ror, +, -, *, /, rem, mod

-Konverter-Funktionen:
to_integer(<(un)signed>), to_(un)signed(<integer>,<natural>)

- Bibliothek std_logic_1164:

-std_logic: and, nand, or, nor, xor, xnor, not, sll, srl, rol, ror

-Konverter-Funktionen:
To_bit(<std_ulogic>), To_bitvector(<std_ulogic_vector>),
To_StdLogicVector(<bit_vector>)

VHDL-Grundlagen - Operatoren III/V

- Bibliothek numeric_std:

-unsigned/signed: and, or, nand, nor, xor, xnor, not, =, /=, <, <= , >, >=,
shift_left, shift_right, rotate_left, rotate_right, +, -, *, /,
rem, mod

-Konverter-Funktionen:
to_integer(<(un)signed>), to_(un)signed(<integer>,<natural>),
to_(un)signed(<std_logic_vector), to_std_logic_vector(<(un)signed>)

-Casts (nur zwischen artverwanten Typen möglich, z.B. vector -> vector); nicht vector-> int):
std_logic_vector(<(un)signed>), (un)signed (<std_logic_vector>)

VHDL-Grundlagen - Operatoren IV/V

- Bibliothek std_logic_arith (Synopsis-Lib):

-signed/unsigned: =, /=, <, <=, >, >=, +, -, *, abs

-Konverter-Funktionen:
  CONV_INTEGER(<(un)signed>|<std_ulogic>),
  CONV_(UN)SIGNED(<integer>|<(un)signed>|<std_ulogic>),
  CONV_STD_LOGIC_VECTOR(<integer>|<(un)signed>|<std_ulogic>)

- Bibliothek std_logic_unsigned bzw. std_logic_signed (Synopsis-Lib)

-std_logic_vector: =, /=, <, <=, >, >=, +, -, *

-Konverter-Funktionen:
  CONV_INTEGER(<std_logic_vector>)

- ACHTUNG: VHDL 87 nicht unterstützt:
  -Bibliothek std: xnor, sll, srl, sla, sra, rol, ror;
  -Bibliothek std_logic: xnor

VHDL-Grundlagen - Operatoren V/V

Empfehlung für Bibliotheken:

- Xilinx verwendet:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_(un)signed.all;
```

- Am sinnvollsten zu verwenden:

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

Besondere Operatoren

Eine Besonderheit bilden noch die sog. „Verkettungsoperatoren" „&" und „|":

-mit „&" lassen sich bspsweise Signale und auch Vektoren zu Vektoren zusammenfassen:

```
a <= '1';
b <= "01";
c <= a & b; --="101"
```

-mit „|" können bspsweise in selektiven Signalzuweisungen Bedingungen vereinfacht werden:

```
with b select
  a<= '0' when "00",
      '1' when "01",
      '1' when "10",
      '1' when "11";
alternativ:
 with b select
  a<= '0' when "00",
      '1' when "01" | "10" |"11";
```

VHDL-Grundlagen - Operatoren - Boolsche' Gleichungen - Übung

- Setzen sie folgende bool'sche Gleichungen in ein gemeinsames VHDL-Modul mit dem Namen boolequ
- Y1 = (E1 ∧ E2) ∨ E3 und Y2 = (E1 ∨ E2) ∧ E3

VHDL-Grundlagen - process

- bisher verwendete nebenläufige Konstrukte erlauben nur stark eingeschränkte Modellierung
- process= grundlegenstes VHDL-Syntaxkonstrukt
  - alle Prozesse in einer architecture werden nebenläufig/ gleichzeitig ausgeführt
  - innerhalb eines Prozesses werden Anweisungen sequentiell ausgeführt
  - -> Modellierung komplexerer Funktionalität möglich
  - “when-else”- und “when-select”-Konstrukte sind eigentlich spezielle Prozesse

VHDL-Grundlagen - process Varianten

- Prozesse mit Empfindlichkeitsliste
  - werden ausgeführt/aktiviert, wenn sich ein Signalwert in der Empfindlichkeitsliste ändert (event)
- Prozesse ohne Empfindlichkeitsliste
  - werden bis zur nächsten wait-Anweisung (wait until, wait for) ausgeführt
  - sind nur synthetisierbar:
    - wenn NUR EINE “wait until”-Anweisung verwendet wird und diese am Anfang steht
    - „wait for“-Anweisungen sind generell nicht synthetisierbar
  - -> Empfehlung: Verwendung von Prozessen mit Empfindlichkeitsliste

VHDL-Grundlagen - process Syntax

- Syntax

```
[<processlabel>:] process [(<Empf.liste>)]
  <Deklarationsteil>
begin
  {<sequentielle Anweisungen>;}
end process [<processlabel>]

Anmerkungen:
<Empf.liste>
  Signale bei deren Änderung(event) der
  Prozess ausgeführt/aktiviert wird

<Deklarationsteil>:
  hier z.B. lokale Variablen und Signale
  deklarieren

<sequentielle Anweisungen>:
  z.B.
  case-Anweisung
  if-Anweisung
  for loop
  while loop
```

- Beispiel Multiplexer

```
entity MUX4 is
  port( S: in bit_vector (1 downto 0);
        E: in bit_vector (3 downto 0);
        Y: out bit);
end MUX4;

architecture BEHAV of MUX4 is
begin
MUXPROC: process(S,E)
  begin
    case S is
          when "00" => Y<= E(0);
          when "01" => Y<= E(1);
          when "10" => Y<= E(2);
          when "11" => Y<= E(3);
    end case;
  end process MUXPROC;
end BEHAV;
```

VHDL-Grundlagen - Sequentielle Statements (case)

- Syntax

```
case <steuersig> is
  when <Bedingung_1> => {<Anweisung>;}
  [when <Bedingung_2> => {<Anweisung>;}]
  ...
  [when others => {<Anweisung>;}]
end case;
```

Anmerkung:
Es müssen alle möglichen Werte des <steuersig> in expliziten when-Zweigen berücksichtigt werden
(alterantiv auch „when others"-Zweig)

- Beispiel Multiplexer

```
entity MUX4 is
  port( S: in bit_vector (1 downto 0);
        E: in bit_vector (3 downto 0);
        Y: out bit);
end MUX4;

architecture BEHAV of MUX4 is
begin
MUXPROC: process(S,E)
  begin
    case S is
          when "00" => Y<= E(0);
          when "01" => Y<= E(1);
          when "10" => Y<= E(2);
          when "11" => Y<= E(3);
    end case;
  end process MUXPROC;
end BEHAV;
```

VHDL-Grundlagen - Sequentielle Statements (if)

- Syntax

```
if <Bedingung_1> then
  {<Anweisung>;}
[elsif <Bedingung_2> then
  {<Anweisung>;}
elsif
  ...]
[else
 {<Anweisung>;}]
end if;

Anmerkung:
!!! Schreibweise „elsif“ und „end if“
  einprägen
```

- Beispiel Multiplexer

```
entity MUX4 is
  port( S: in bit_vector (1 downto 0);
        E: in bit_vector (3 downto 0);
        Y: out bit);
end MUX4;

architecture BEHAV of MUX4 is
begin
MUXPROC: process(S,E)
  begin
    if S="00“ then
       Y<= E(0);
    elsif S="01“ then
       Y<= E(1);
    elsif S="10“ then
       Y<= E(2);
    else
       Y<= E(3);
    end if;
  end process MUXPROC;
end BEHAV;
```

OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG

VHDL-Grundlagen - std logic - Übung

- Modellieren Sie einen BIN-to-Gray-Code-Umwandler mit std-logic
- Verwenden Sie keine Operatoren sondern setzen Sie die Wertetabelle mit „case“ um
- Verwenden Sie einen 3bit Eingangsbus names BIN und einen 3bit Ausgangsbus names GRAY
- Schreiben Sie zweispaltig für gute Platzausnutzung

Nr	x3	x2	x1	y		
0	0	0	0	0	0	0
1	0	0	1	0	0	1
2	0	1	0	0	1	1
3	0	1	1	0	1	0
4	1	0	0	1	1	0
5	1	0	1	1	1	1
6	1	1	0	1	0	1
7	1	1	1	1	0	0

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

entity BIN2GRAY is
  port(
     NUM: in  std_logic_vector(2 downto 0);
     GRAY: out std_logic_vector(2 downto 0)
  );
end BIN2GRAY;
architecture BEHAV of BIN2GRAY is
begin
```

VHDL-Grundlagen - „variable“ vs. “signal” in „process“

- Variable wird Wert sofort zugewiesen;
  Signal wird Wert erst bei Prozessende (oder bei wait-Anweisung) zugewiesen
- Variable nur innerhalb process; wenn Wert außerhalb weiter verwendet werden soll, muss dieser einem lokalen Signal der architecture zugewiesen werden
- Zuweisungsoperator für Variable „:=„

```
architecture BEHAV of TEST is
signal SIG: bit;
begin
PROC: process(CLK)
  begin
    SIG <='1';
    ...
        if SIG = '1' then -- !! Only executed in
                          -- 2nd execution
          SIG <= '0';
          ...
        else
          ...
        end if;
  end process PROC;
end BEHAV;
```

```
architecture BEHAV of TEST is
signal SIG: bit;
begin
PROC: process(CLK)
  variable SIGVAR : bit;
  begin
    SIGVAR :='1';
    ...
        if SIGVAR = '1' then
          SIGVAR:='0';
          ...
        else
          ...
        end if;
    SIG <= SIGVAR;
  end process PROC;
end BEHAV;
```

OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG

VHDL-Grundlagen - Sequentielle Statements (for-Schleifen) (R)

- for (Anz. Schleifendurchläufe zur Laufzeit fest)

```
[<label>:]
for <Schleifenspezifikation> loop
    {<Anweisungen>;}
end loop;

Anmerkungen:
<Schleifenspezifikation>:
<Schleifenindex> in <U_Grenze> to <O_Grenze>
<Schleifenindex> in <O_Grenze> downto <U_Grenze>

Zum vorzeitigen Verlassen der Schleife
exit when <Bedingung>
Zum vorzeitigen Beenden eines
Schleifendurchlaufs
next when <Bedingung>
```

- Beispiel Paritätsgenerator(4bit)

```
entity PAR is
  port (D: in bit_vector (3 downto 0);
        PARBIT: out bit);
end PAR;

architecture BEHAV of PAR is
  begin
  PARPROC: process (D)
     variable PARBITVAR: bit;
     begin
     -- 0 even; 1 odd
       PARBITVAR :='0';
       for I in 3 downto 0 loop
         if D(I) ='1' then
            PARBITVAR := not PARBITVAR;
         end if;
       end loop;
     PARBIT <= PARBITVAR;
  end process PARPROC;
end BEHAV;
```

OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG

VHDL-Grundlagen - Sequentielle Statements (while-Schleifen) (R)

- while (Abbruchbedingung)

```
[<label>:]
while <Schleifenspezifikation> loop
    {<Anweisungen>;}
end loop;

Anmerkungen:
Zum vorzeitigen Verlassen der Schleife
exit when <Bedingung>
Zum vorzeitigen Beenden eines Schleifendurchl.
next when <Bedingung>
```

- Beispiel ser. Paritätsgenerator

```
entity PAR is
  port (CLK, DIN, SA, SP : in bit;
        DOUT: out bit);
end PAR;

architecture BEHAV of PAR is
  begin
  PARPROC: process
    variable PARBITVAR: bit;
    begin
      wait until (CLK'event and CLK='1');
      if SA='1' then
        PARBITVAR :='0';
        while SP='0' loop
          DOUT <= DIN;
          if DIN ='1' then
            PARBITVAR := not PARBITVAR;
          end if;
          wait until (CLK'event and CLK='1');
        end loop;
        DOUT <= PARBITVAR;
      end if;
  end process PARPROC;
end BEHAV;
```

VHDL-Grundlagen - process Zusammenfassung (wichtigste Folie!!!)

- process= wichtigstes VHDL-Syntaxkonstrukt
  - alle **Prozesse** in einer architecture werden **nebenläufig/ gleichzeitig** ausgeführt
  - **innerhalb** eines Prozesses werden **Anweisungen sequentiell** ausgeführt
  - **!!!Achtung Besonderheiten!!!**
    - Signalwertänderung erfolgt immer erst am Prozessende (oder bei wait-Anweisung)
    - -> nur die letzte Signalzuweisung ist entscheidend
    - -> wenn geänderte Werte innerhalb der Sequenz sofort zur Verfügung stehen müssen, müssen statt Signalen Variablen verwendet werden
    - -> wenn die Variablenwerte außerhalb weiter verwendet werden, muss am Ende des Prozesses die Variable einem lokalen Signal der architecture zugewiesen werden
    - Austausch zwischen Prozessen erfolgt mit den lokalen Signalen der architecture
      (bidirektionale Kommunikation 2 Signale!!!)
    - Innerhalb von Prozessen sind **nur sequentielle** Anweisungen und unbedingte Signalzuweisungen erlaubt (NICHT: with select und when else)

VHDL-Grundlagen - Modellierung kombinatorischer Logikelemente

- mit nebenläufigen Signalzuweisungen
- mit Process:
  - in Empfindlichkeitsliste: alle Signale, die auf der rechten Seite einer Zuweisung oder in einer Abfrage stehen
- Anmerkung:
  - Einem Signal muss in allen Verzweigungen ein Wert zugewiesen werden, da sonst ein Latch generiert wird (vgl. spätere Folien)

- Beispiel Multiplexer

```
entity MUX4 is
  port( S: in bit_vector (1 downto 0);
        E: in bit_vector (3 downto 0);
        Y: out bit);
end MUX4;
architecture BEHAV of MUX4 is
begin
  with S select
  Y<=  E(0) when "00",
       E(1) when "01",
       E(2) when "10",
       E(3) when "11";
end BEHAV;
...
architecture BEHAV of MUX4 is
begin
MUXPROC: process(S,E)
  begin
    case S is
           when "00" => Y<= E(0);
           when "01" => Y<= E(1);
           when "10" => Y<= E(2);
           when "11" => Y<= E(3);
    end case;
  end process MUXPROC;
end BEHAV;
...
```

VHDL-Grundlagen - Häufige Probleme bei kombinatorischer Logik

- Latches (unerwünschtes Speicherverhalten)
  - Ursache:
    - Signal/Variable wird nicht in allen Verzweigungen ein Wert zugewiesen
    - Signal/Variable wird in einer Verzweigung vor Wertzuweisung gelesen
  - Lösung:
    - Zuweisung eines Defaultwerts für alle Signale/Variable vor einer Programm-Verzweigung
      (vgl. Folien Statemachines)

- Unvollständige Empfindlichkeitsliste
  - Abweichungen von Simulation (Speicherverhalten) und synthetisierter Logik
    - Synthese ergänzt Empfindlichkeitsliste automatisch / Simulator nicht