

Simulation von VHDL Modellen (1)

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

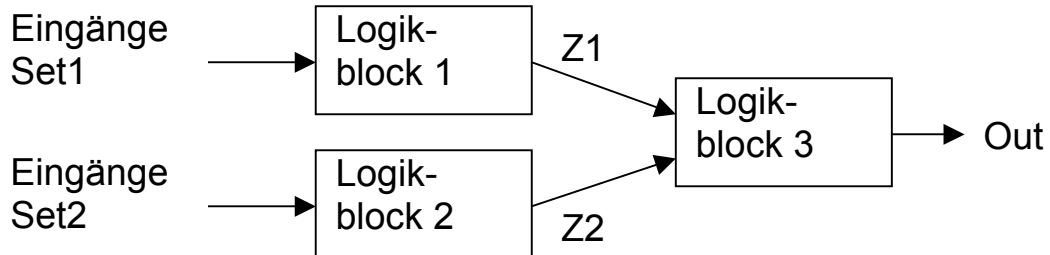
(3)	<code>AV := X * Y;</code>	-- direkte Variablenzuweisung
(4)	<code>BV := AV + Z;</code>	-- direkte Variablenzuweisung

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

```

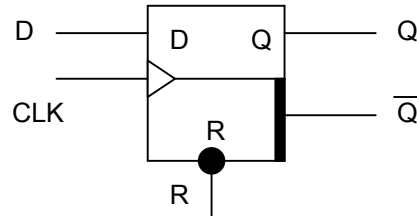
```

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

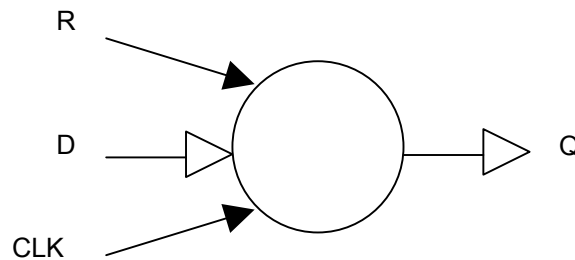
```

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

Digitales Schaltelement D-Flipflop



Schaltsymbol nach DIN



Grafische Repräsentation mit Signalcharakterisierung

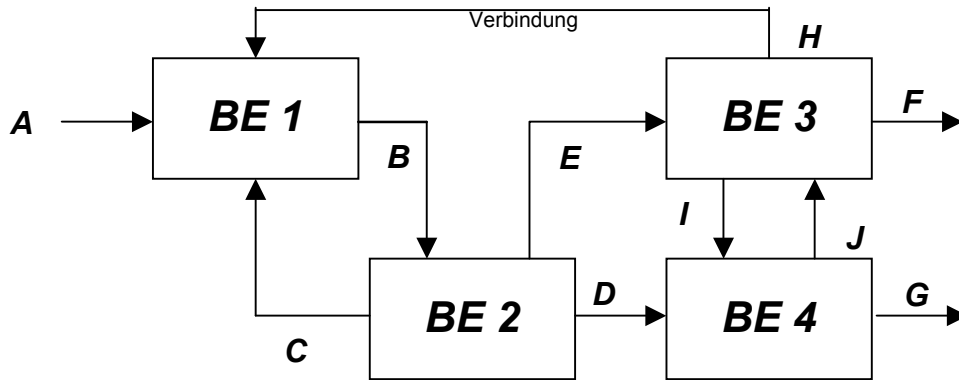
```

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;

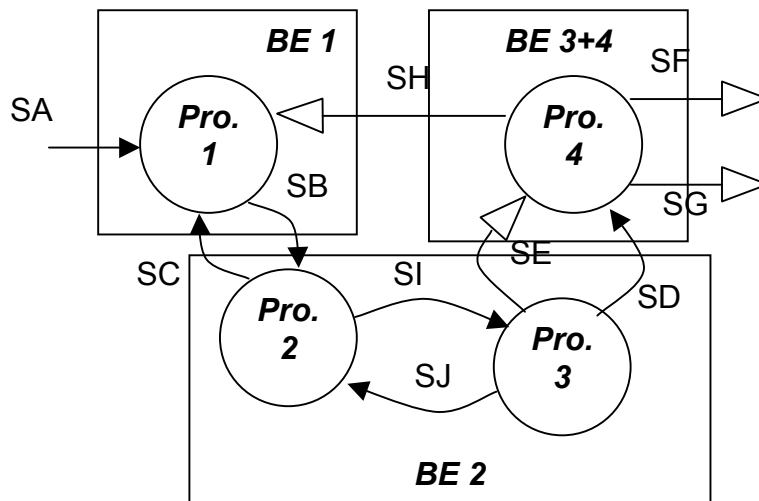
```

VHDL Modell

Schaltungsmodell aus Prozessen

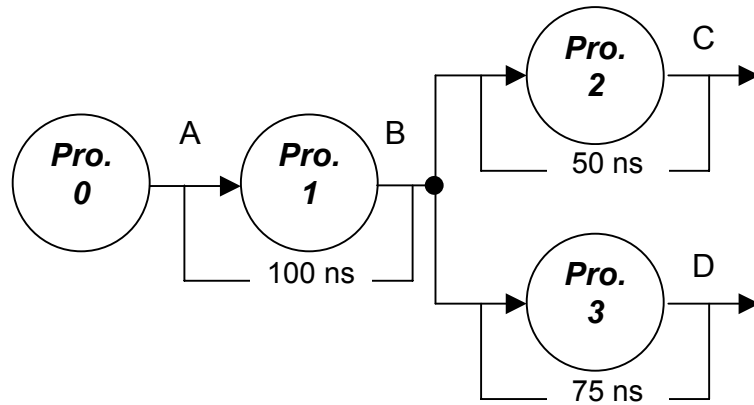


Digitalschaltung aus Bauelementen BE i



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. H.-D. Hümmer	Fak. 5 IIMT, IT/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)

Universität Duisburg - Essen	Dr.-Ing. H.-D. Hümmer	Fak. 5 IIMT, IT/VS
Hilfsblatt zur Vorlesung: Rechnergesteuerte Systeme 2		SEITE: 25

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;

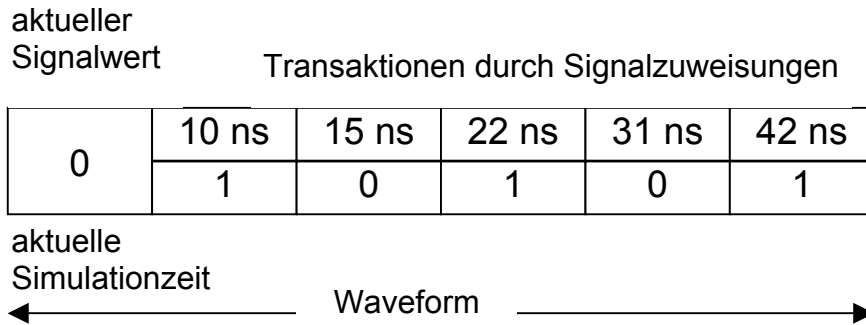
```

Beispielarchitekturen mit Prozessen

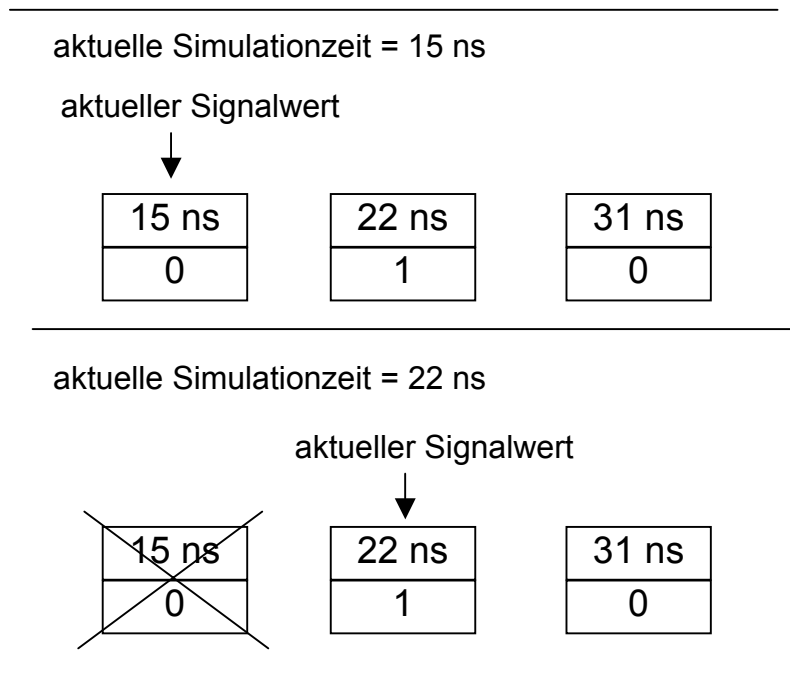
Zeit (ns)	X	Y5	Z5	Y5A	Z5A
0	'0'	'0'	'0'	'0'	'0'
+1		'0'	'0'	'0'	'0'
1	'1'				
+1		'1'	'0'	'1'	'0'
+2				'1'	'1'
2					
3					
4	'0'				
+1		'0'	'1'	'0'	'1'
+2				'0'	'0'
5					
6					

Reaktion der vier Beispielarchitekturen

Signaltreiber und Simulation



Signaltreiber eines Prozesses

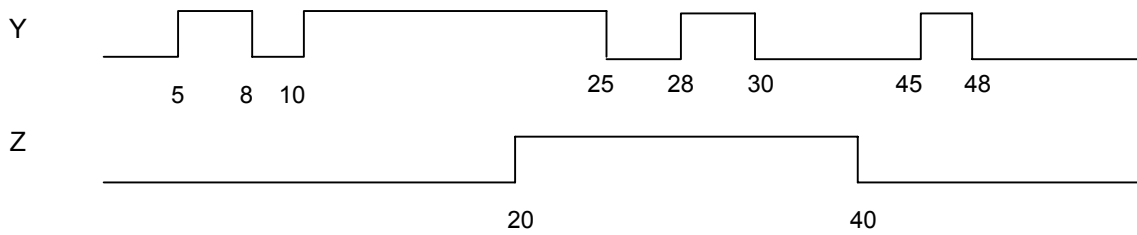


Fortschreiten der Simulationszeit

Verzögerungsmechanismen

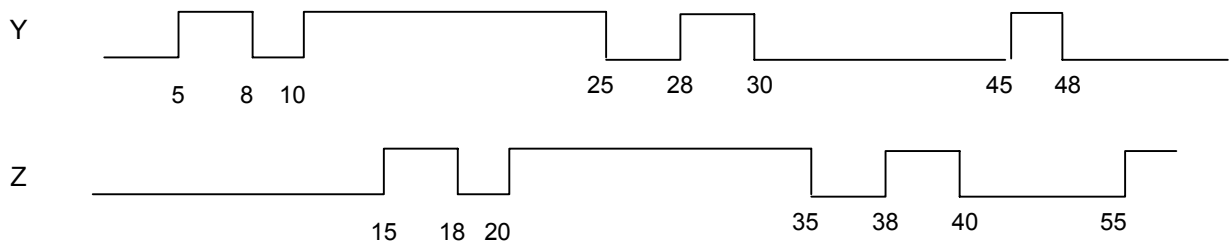
Inertial Delay

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	20 T + 22ns	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	11 T + 10ns	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. H.-D. Hümmer	Fak. 5 IIMT, IT/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. H.-D. Hümmer	Fak. 5 IIMT, IT/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;

```