Lösungshinweise zum Aufgabenblatt 3: VHDL 2

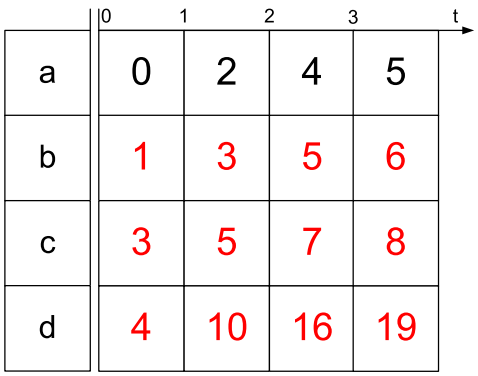
Ziel: VHDL-Programme lesen und verstehen, Zeitmodell in VHDL

Zu Aufgabe 1: Timing

a) Bei der Initialisierung, und später bei jeder Änderung von Signal a, wird der Prozess einmal durchlaufen. Aufgrund der Zuweisung an b wird er jedoch noch einmal durchlaufen bevor er sich dann stabilisiert (keine Änderungen an den Signalen a und b). Da der zweite Durchlauf durch die Änderung an Signal b ausgelöst wird, erhält b in diesem entsprechend den neuen Wert aus dem ersten Durchlauf.

$$\begin{aligned} b_{neu} &= a_{neu} + 1; \\ c &= b_{neu} + 2 = a_{neu} + 3; \\ d &= a_{neu} + b_{neu} + c = \ldots = 3 \cdot a_{neu} + 4; \end{aligned}$$

```
architecture A of Aufgabe1 is
  signal a, b: integer := 0;
begin
  a <= 2 after 1 ns, 4 after 2 ns,
       5 after 3 ns;
  process (a, b)
    variable c, d: integer := 0;
  begin
    b <= a + 1;
    c := b + 2;
    d := a + b + c;
  end process;
end A;
```



b) Der Prozess wir aufgrund der Zuweisung an b noch einmal durchlaufen. Dann ist a und b allerdings stabil und kein weiterer Durchlauf findet statt. Im zweiten Prozessdurchlauf hat b bereits den neuen Wert aus dem ersten Durchlauf. Die Zuweisung ($b <= a - c + 3$) an das Signal b hat keine Auswirkung auf die Werte und kann bei der Betrachtung ignoriert werden. Denn, sofort danach wird b mit der neuen Zuweisung ($b <= a + c$) überschrieben, und bei ($d <= b - 3$) ist b_{alt} von Interesse!

$$\begin{aligned} \mathbf{b_{neu}} &= a_{neu} + c \\ &= \mathbf{2 \cdot a_{neu} + 1} \\ \\ \mathbf{d} &= b_{neu} - 3 \\ &= \mathbf{2 \cdot a_{neu} - 2} \end{aligned}$$

$$\begin{aligned} \mathbf{c} &= a_{neu} - b_{neu} - d \\ &= a_{neu} - b_{neu} - (b_{neu} - 3) \\ &= a_{neu} - 2 \cdot b_{neu} + 3 \\ &= a_{neu} - 2 \cdot (2 \cdot a_{neu} + 1) + 3 \\ &= \mathbf{-3 \cdot a_{neu} + 1} \end{aligned}$$

```
architecture B of Aufgabe1 is
  signal a, b : integer := 0;
begin
  a <= 13 after 1 ns, 7 after 2 ns;
  process (a, b)
    variable c, d : integer := 0;
    begin
      c := 1 + a ;
      b <= a - c + 3 ;
      d := b - 3 ;
      b <= a + c ;
      c := a - b - d ;
  end process;
end B;
```

	0	1	2	t
a	0	13	7	
b	1	27	15	
c	1	-38	-20	
d	-2	24	12	

Zu Aufgabe 2: Gatterschaltung

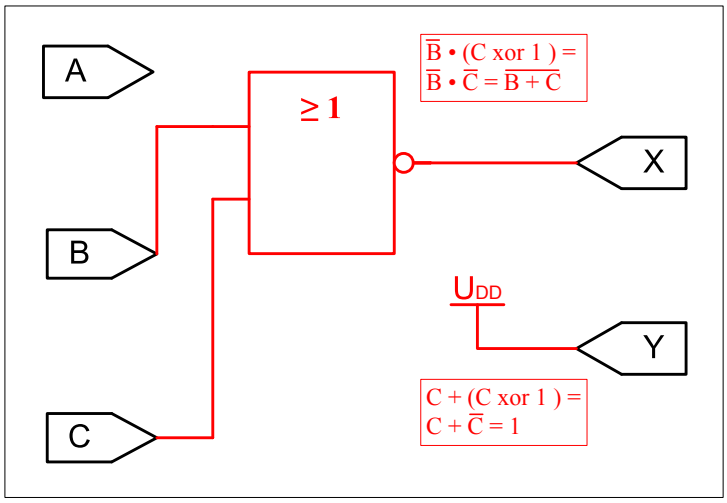
a) Der Prozess P hat, zusätzlich zu den Eingangssignalen A, B *und* C, ein lokales Signal D in der *Sensitivliste*. Eine Änderung an einem der Eingangssignale führt zu einem zusätzlichen Durchlauf des Prozesses, da dabei eine Wertänderung des Signals D stattfindet. Danach stabilisieren sich die Werte bis zu nächsten Änderung an A, B *oder* C.
Bei jedem Durchlauf des Prozesses wird die „alte“ Zuweisung (D <= A and B and C) durch die „neue“ (D <= C xor 1) überschrieben. Dies führt dazu, dass D immer den Wert (C xor 1) zugewiesen bekommt.

$$
\begin{aligned}
\mathbf{X_{neu}} &= \overline{B}_{neu} \cdot D_{neu} \\
&= \overline{B}_{neu} \cdot (C_{neu} \text{ xor } 1) \\
&= \overline{B}_{neu} \cdot \left((\overline{C}_{neu} \cdot 1) + (C_{neu} \cdot \overline{1})\right) \\
&= \overline{B}_{neu} \cdot \overline{C}_{neu} \\
&= \overline{\mathbf{B_{neu}} + \mathbf{C_{neu}}}
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{Y_{neu}} &= C_{neu} + D_{neu} \\
&= C_{neu} + (C_{neu} \text{ xor } 1) \\
&= C_{neu} + \left((\overline{C}_{neu} \cdot 1) + (C_{neu} \cdot \overline{1})\right) \\
&= C_{neu} + \overline{C}_{neu} \\
&= \mathbf{1}
\end{aligned}
$$

```
entity C_ent is
  port( A, B, C  : in   std_logic;
        X, Y     : out std_logic );
end C_ent;

architecture C of C_ent is
  signal D: std_logic;
begin
  P : process (A, B, C, D)
  begin
    D <= A and B and C;
    X <= (not B) and D;
    D <= C xor 1;
    Y <= C or D;
  end process;
end C;
```



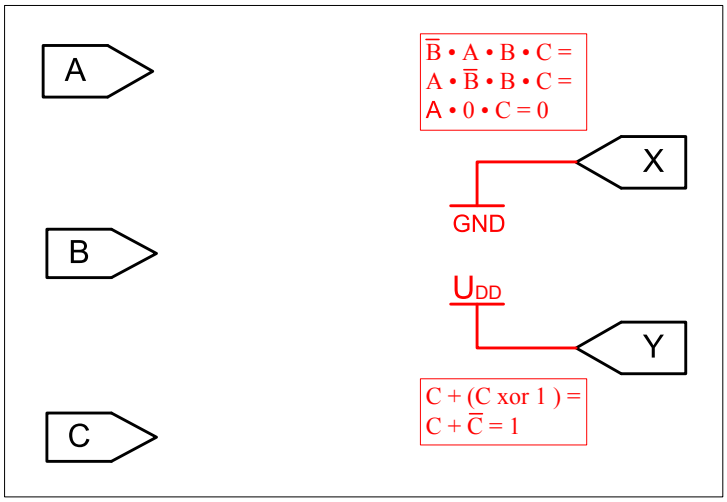
b) Im Prozess R ist D als lokale **Variable** deklariert. Das führt zu sofortigen Sichtbarkeit des an D neu zugewiesenen Wertes. Somit werden, nach einer Signaländerung an A, B oder C, an die Ausgangssignale X und Y jeweils verschiedene Werte zugewiesen.

$$
\begin{aligned}
\mathbf{X_{neu}} &= \overline{B}_{neu} \cdot D \\
&= \overline{B}_{neu} \cdot (A_{neu} \cdot B_{neu} \cdot C_{neu}) \\
&= \underbrace{\overline{B}_{neu} \cdot B_{neu}}_{=0} \cdot A_{neu} \cdot C_{neu} \\
&= \underbrace{0 \cdot A_{neu} \cdot C_{neu}}_{=0} \\
&= \mathbf{0}
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{Y_{neu}} &= C_{neu} + D \\
&= C_{neu} + (C_{neu} \text{ xor } 1) \\
&= C_{neu} + \left((\overline{C}_{neu} \cdot 1) + (C_{neu} \cdot \overline{1})\right) \\
&= C_{neu} + \overline{C}_{neu} \\
&= \mathbf{1}
\end{aligned}
$$

```
entity D_ent is
  port( A, B, C : in  std_logic;
            X, Y : out std_logic );
end D_ent;

architecture D of D_ent is
begin
  R: process (A, B, C)
    variable D: std_logic;
    begin
      D := A and B and C;
      X <= (not B) and D;
      D := C xor 1;
      Y <= C or D;
  end process;
end D;
```



Zu Aufgabe 3: Automat

Nachfolgend ist ein Automat in Form eines Prozesses realisiert. Die *synchrone* Zustandsfortschaltung erfolgt über den Takt „CLK“. Zusätzlich verfügt der Automat über das asynchrone Reset „Res“.
Der Automat beschreibt das so genannten „fully interlocked Protocol“ der Sender-Seite. Nachdem die Daten auf den Bus gelegt werden, informiert der Sender den Empfänger über das Signal *„Rdy“ (= data ready)* darüber und wartet auf die (Empfangs-)Bestätigung *(„Ack“ = 1)*. Hat der Empfänger den Empfang bestätigt, so setzt der Sender *„Rdy“* wieder auf null und wartet wieder auf eine Bestätigung („Ack“ = 0). Anschließend wechselt er in den Ausgangszustand zurück.

```
Entity AUTOMAT is
    Port ( CLK  : in std_logic;
           Res  : in std_logic;
           Ack  : in std_logic;
           Data : out std_logic_vector(7 downto 0);
           Rdy  : out std_logic );
end AUTOMAT;

architecture BEHAVIORAL of AUTOMAT is

type state is (S1, S2, S3);
signal CS: state;
signal DataToSend: std_logic_vector(7 downto 0);
-- interne, zu versendende Daten
begin

   SendData : Process (CLK, Res, Ack)
   begin
      Rdy <= '0';
      if (Res = '1') then
         CS <= S1;
      elsif (CLK'event and CLK = '1') then
        case CS is
          when S1 =>
            Data <= DataToSend;
            CS <= S2;
          when S2 =>
            Rdy <= '1';
            if (Ack  = '1') then
              CS <= S3;
            else
              CS <= S2;
            end if;
          when S3 =>
            if (Ack  = '0') then
              CS <= S1;
            else
              CS <= S3;
            end if;
        end case;
      end if;
   end process;

end BEHAVIORAL;
```