Übungsaufgaben mit Lösungen zur 6. Auflage

Zu den einzelnen Kapiteln sind Übungsaufgaben angegeben. Einige enthalten die Lösung in Kurzform. Sie finden die ausführlichen Musterlösungen zu allen Aufgaben sowie VHDL-Dateien zum Downloaden unter

http://www.ecs.hs-osnabrueck.de/buch.html

Aufg.	Kap.	Thema
1, 2	2	Minimieren logischer Gleichungen
3, 4	2,4	Minimieren logischer Gleichungen [VHDL]* zu 4
5	2,4,5	Entwurf eines 2-Bit-Vergleichers [VHDL]*
6	2,4,5	Schaltnetz zur Wasserstandsregelung [VHDL]*
7	3	Widerstandsdimensionierung für Gatter mit offenem Kollektor
8	2,3	Ansteuerung von Leuchtdioden
9	4,5	VHDL-Entwurf eines Addierers – Test mit einer Testbench [VHDL]*
10	2,4,5	Darstellung von Hexadezimalziffern auf einer 7-Segmentanz. [VHDL]*
11	2,6	Zustands- und flankengesteuertes D-Flipflop
12	2,6	Analyse eines Schaltwerks mit D-Flipflops
13	6	Entwurf eines JK- und eines T-Flipflops mit Hilfe eines D-Flipflops
14	2,4,5,6	Steuerung einer Ampelanlage [VHDL]*
15	4,6	Testbench für einen synchronen Dualzähler [VHDL]*
16	4,6	VHDL-Entwurf des programmierbaren Synchronzählers 74163 [VHDL]*
17	2,4,6	Synchroner Modulo-5-Zähler [VHDL]*
18	2,4,6	Entwurf eines synchronen Schaltwerks (Moore-Automat) [VHDL]*
19	2,4,6	Entwurf eines synchronen Schaltwerks (Mealy-Automat) [VHDL]*
20	2,4,6	Entwurf eines synchronen Schaltwerks mit Registerausgabe [VHDL]*
21	4,7	Entwurf eines SRAMs 1Ki x 8 Bit [VHDL]*
22	1,2,7	Entwurf eines Speichersystems mit 8-Bit-Wortbreite
23	1,2,7	Speichersystem mit 16-Bit-Wortbreite

Anmerkung:

Abbildungen und Tabellen in den Übungsaufgaben werden aufgabenweise durchnummeriert und zusätzlich mit "Ü" gekennzeichnet, um Verwechslungen mit den Abbildungs- und Tabellennummern der Kap. 1 bis 9 und 11 zu vermeiden. Aufgaben, die mit [VHDL]* gekennzeichnet sind, enthalten entsprechende VHDL-Modelle. In den Übungsaufgaben Ü9, Ü15 und Ü21 sind ausführliche Beispiele mit VHDL-Modellen, die eine Testbench enthalten, dargestellt.

Aufgabe 1: Minimieren logischer Gleichungen

Gegeben ist folgende logische Gleichung:

$Y = A\,B\,C \vee \overline{A}\;\overline{B}\;\overline{C} \vee A\,B\,\overline{C} \vee A\;\overline{B}\;\overline{C}$

a) Vereinfachen Sie die logische Gleichung mit der Booleschen Algebra und geben Sie die negierte und nichtnegierte disjunktive Minimalform an.

b) Minimieren Sie die Gleichung mit dem KV-Diagramm und geben Sie die disjunktiven minimalen Gleichungen an. Entwerfen Sie die zugehörigen digitalen Schaltungen.

Lösung 1: nichtnegierte disjunktive Minimalform $Y = A \vee \overline{B}\;\overline{C}$

Lösung 2: negierte disjunktive Minimalform $Y = \overline{\overline{A}\;C \vee \overline{A}\,B}$

Aufgabe 2: Minimieren logischer Gleichungen

Gegeben ist folgende logische Gleichung mit den Zwischengrößen U und X.

$Y = A\,B\,U \vee \overline{A}\,X\,C \vee [\overline{A} \wedge (B \vee U)]$ mit $U = B\,\overline{C}$ und $X = B \vee \overline{A}\,C$

Gesucht sind die disjunktiven Minimalformen. Geben Sie die zugehörigen Schaltungen an. Vergleichen Sie die negierte mit der nichtnegierten Form.

Lösung 1: nichtnegierte disjunktive Minimalform: $Y = B\,\overline{C} \vee \overline{A}\;C$

Lösung 2: negierte disjunktive Minimalform: $Y = \overline{A\,C \vee \overline{B}\;\overline{C}}$

Aufgabe 3: Minimieren logischer Gleichungen

	X1	X2	X3	Y1	Y2
0	0	0	0	0	0
1	0	0	1	1	1
2	0	1	0	1	1
3	0	1	1	0	0
4	1	0	0	1	0
5	1	0	1	0	1
6	1	1	0	0	1
7	1	1	1	0	1

Tabelle Ü3.1 Wahrheitstabelle zur 3. Aufgabe

Gegeben ist die Wahrheitstabelle Tabelle Ü3.1 mit den Eingangsvariablen X1, X2, X3 und den Ausgangsvariablen Y1 und Y2.

3.1 Bestimmen Sie aus der Wahrheitstabelle die logischen Gleichungen unter Anwendung der disjunktiven Normalform.

3.2 Geben Sie die disjunktiven Minimalformen an.

Lösung zu 3.1. Disjunktive Normalformen:

Y1 = ¬X1¬X2 X3 ∨ ¬X1 X2¬X3 ∨ X1¬X2¬X3

Y2 = ¬X1¬X2 X3 ∨ ¬X1 X2¬X3 ∨ X1¬X2 X3 ∨ X1 X2¬X3 ∨ X1 X2 X3

Lösung zu 3.2 Disjunktive Minimalformen:

$Y1 = \neg X1 \neg X2\ X3 \vee \neg X1\ X2 \neg X3 \vee X1 \neg X2 \neg X3$ (nichtnegiert)

$\neg Y1 = \neg X1 \neg X2 \neg X3 \vee X1\ X2 \vee X2\ X3 \vee X1\ X3$ (negiert)

$Y2 = X2 \neg X3 \vee X1\ X3 \vee \neg X2\ X3$ (nichtnegiert)

alternativ: $Y2 = X2 \neg X3 \vee X1\ X2 \vee \neg X2\ X3$ (nichtnegiert)

$\neg Y2 = \neg X2 \neg X3 \vee \neg X1\ X2\ X3$ (negiert)

Aufgabe 4: Minimieren logischer Gleichungen [VHDL]*

Bestimmen Sie aus der Wahrheitstabelle (Tabelle Ü4.1) mit Hilfe des KV-Diagramms die negierten und nichtnegierten Minimalformen für Y1, Y2 und Y3.

Tabelle Ü4.1 Wahrheitstabelle zur 4. Aufgabe

	X1	X2	X3	X4	Y1	Y2	Y3
0	0	0	0	0	1	1	1
1	0	0	0	1	0	*	0
2	0	0	1	0	0	0	0
3	0	0	1	1	0	0	0
4	0	1	0	0	1	1	*
5	0	1	0	1	0	0	*
6	0	1	1	0	0	*	0
7	0	1	1	1	0	0	1
8	1	0	0	0	1	1	1
9	1	0	0	1	0	0	0
10	1	0	1	0	0	*	*
11	1	0	1	1	0	0	0
12	1	1	0	0	1	1	1
13	1	1	0	1	0	0	1
14	1	1	1	0	1	1	1
15	1	1	1	1	0	0	0

Lösung zu 4:

Nichtnegierte disjunktive Minimalformen:

$Y1 = \neg X3 \neg X4 \vee X1\ X2 \neg X4$

$Y2 = \neg X3 \neg X4 \vee X2 \neg X4$ alternativ: $Y2 = \neg X3 \neg X4 \vee X1 \neg X4$

$Y3 = X2 \neg X3 \vee \neg X3 \neg X4 \vee X1 \neg X4 \vee \neg X1\ X2\ X4$

Negierte disjunktive Minimalformen:

$\neg Y1 = X4 \vee \neg X1\ X3 \vee \neg X2\ X3$

$\neg Y2 = X4 \vee \neg X1\ X3$ alternativ: $\neg Y2 = X4 \vee \neg X2\ X3$

$\neg Y3 = \neg X2\ X4 \vee X1\ X3\ X4 \vee \neg X1\ X3 \neg X4$

VHDL-Modell: Entwurf zu vorgegebener Wahrheitstabelle Tab. Ü4.1

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

entity uebung_4 is port (
   x:       in std_logic_vector (1 to 4);   -- Vektor x mit den Elementen x(1), x(2), x(3), x(4)
   y:       out std_logic_vector (1 to 3)); -- Vektor y mit den Elementen y(1), y(2), y(3)
end uebung_4;

architecture verhalten of uebung_4 is begin
wahrheits_tab: process(x) begin
  case x is
   when "0000" => y <= "111";
   when "0001" => y <= "0-0";
   when "0010" => y <= "000";
   when "0011" => y <= "000";
   when "0100" => y <= "11-";
   when "0101" => y <= "00-";
   when "0110" => y <= "0-0";
   when "0111" => y <= "001";
   when "1000" => y <= "111";
   when "1001" => y <= "000";
   when "1010" => y <= "0--";
   when "1011" => y <= "000";
   when "1100" => y <= "111";
   when "1101" => y <= "001";
   when "1110" => y <= "111";
   when "1111" => y <= "000";
   when others => y <= "---";              -- alle anderen Faelle der neunwertigen Logik
   end case;
 end process wahrheits_tab;
end verhalten;
```

Aufgabe 5: Entwurf eines 2-Bit-Vergleichers [VHDL]*

Entwerfen Sie einen 2-Bit-Vergleicher. Es sollen die 2-Bit-Dualzahlen A und B in Betragsdarstellung miteinander verglichen und das Ergebnis des Vergleichs ausgegeben werden. Es soll gelten:

Vergleich	Ausgänge
A > B	YA = 1, YB = 0 und YG = 0
A < B	YA = 0, YB = 1 und YG = 0
A = B	YA = 0, YB = 0 und YG = 1

Stellen Sie die nichtnegierten und negierten disjunktiven Minimalformen für YA, YB und YG auf. Welche Gleichungen sind zur Realisierung der digitalen Schaltung nötig, wenn die Anzahl der UND-Verknüpfungen pro Gleichung minimal sein soll?

Lösung:

Nichtnegierte disjunktive Minimalformen:

YA = A1¬B1 ∨ A1 A0¬B0 ∨ A0¬B1¬B0

YB = ¬A1 B1 ∨ ¬A0 B1 B0 ∨ ¬A1¬A0 B0

YG = ¬A1¬A0¬B1¬B0 ∨ ¬A1 A0¬B1 B0 ∨ A1¬A0 B1¬B0 ∨ A1 A0 B1 B0

Negierte disjunktive Minimalformen:

¬YA = ¬A1 B1 ∨ ¬A0 B1 ∨ B1 B0 ∨ ¬A1¬A0 ∨ ¬A1 B0

¬YB = ¬B1¬B0 ∨ A1¬B1 ∨ A1¬B0 ∨ A1 A0 ∨ A0¬B1

¬YG = ¬A1 B1 ∨ A1¬B1 ∨ ¬A0 B0 ∨ A0¬B0

Es werden die nichtnegierten disjunktiven Minimalformen benutzt, da die Anzahl der Produktterme im Vergleich zu den negierten Minimalformen geringer ist.

VHDL-Modell: 2-Bit-Vergleicher

```
entity comp is port(
  a,b:     in bit_vector (1 downto 0);   -- 2 Eingangsvektoren mit je 2 Elementen
  ya,yb,yg:          out bit);           -- 3 Ausgaenge
end comp;

architecture verhalten of comp is begin
 vergleich: process (a,b) begin
    if a > b then ya <= '1'; yb <= '0'; yg <= '0';          -- Zahlenvergleich
    elsif a < b then ya <= '0'; yb <= '1'; yg <= '0';
    elsif a = b then ya <= '0'; yb <= '0'; yg <= '1';
    end if;
 end process vergleich;
end verhalten;
```

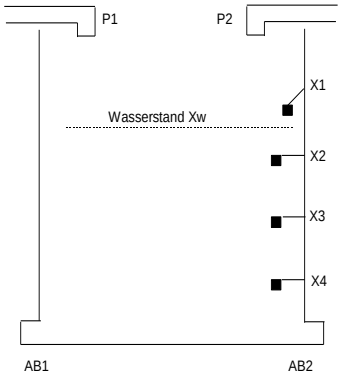
Aufgabe 6: Schaltnetz zur Wasserstandsregelung [VHDL]*

Gegeben ist ein Wasserbehälter mit den beiden Abflüssen AB1 und AB2. Der Behälter kann über zwei Pumpen P1 und P2 gespeist werden. Am Behälterrand sind 4 Schwimmer (X1, X2, X3 und X4) angeordnet, über die der Wasserstand Xw überwacht werden kann. Der Wasserstand im Behälter soll über steuerbare Ventile an den Pumpen und Abflüssen geregelt werden (Abb. Ü6.1).

P1 = 1:	Pumpe 1 eingeschaltet	P1 = 0:	Pumpe 1 ausgeschaltet
P2 = 1:	Pumpe 2 eingeschaltet	P2 = 0:	Pumpe 2 ausgeschaltet
AB1= 1:	AbFluss 1 geöffnet	AB1 = 0:	AbFluss 1 geschlossen
AB2 = 1:	AbFluss 2 geöffnet	AB2 = 0:	AbFluss 2 geschlossen

Falls eine unerlaubte Kombination der Eingangsvariablen auftritt, so soll eine Fehler-Variable F den Logik-Zustand "1" annehmen.

Geben Sie die minimalen disjunktiven Gleichungen für alle Ausgangsvariablen in negierter und nichtnegierter Form an. Welche Gleichungen sind günstiger, wenn die Anzahl der Produktterme als Kriterium gewählt wird?



Xw unterhalb von X4:
Beide Pumpen ein; AB1 und AB2 zu

Xw zwischen X4 und X3:
Beide Pumpen ein; AB1 auf, AB2 zu

Xw zwischen X3 und X2:
P1 ein, P2 aus; AB1 zu, AB2 auf

Xw zwischen X2 und X1:
P1 aus, P2 ein; AB1 und AB2 auf

Xw oberhalb von X1:
P1 und P2 aus; AB1 und AB2 auf

Abb. Ü6.1 Wasserbehälter zur Aufgabe 6

Lösung:

Nichtnegierte disjunktive Minimalform	Negierte disjunktive Minimalform
$P1 = \overline{X2}$	$P1 = \overline{X2}$
$P2 = \overline{X3} \vee \overline{X1}\,X2$	$P2 = \overline{X1 \vee \overline{X2}\,X3}$
$AB1 = \overline{X3}\,X4 \vee X2$	$AB1 = \overline{\overline{X2}\,X3 \vee \overline{X4}}$
$AB2 = X3$	$AB2 = \overline{\overline{X3}} = X3$
$F = X3\ \overline{X4} \vee X2\ \overline{X3} \vee X1\ \overline{X2}$	$F = \overline{\overline{X1}\ \overline{X2}\ \overline{X3} \vee X2\,X3\,X4 \vee \overline{X1}\,X3\,X4}$

Die Gleichungen in negierter und nichtnegierter Form sind gleich günstig, da sie die gleiche Anzahl an Produkttermen enthalten.

VHDL-Modell: Schaltnetz zur Wasserstandsregelung

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

entity wasser_beh is port(
  x:        in std_logic_vector (1 to 4); -- Eingangsvektor x: Elemente x(1), x(2), x(3), x(4)
  P1,P2,AB1,AB2,F:        out std_logic);       --5 Ausgaenge
end wasser_beh;

architecture verhalten of wasser_beh is begin
 schwimmer: process(x)
   begin
     case x is
          when "0000" =>                        -- Wasserstand unterhalb von x4
               P1 <= '1'; P2 <= '1'; AB1 <= '0'; AB2 <= '0'; F <= '0';
          when "0001" =>                        -- Wasserstand zwischen x4 und x3
               P1 <= '1'; P2 <= '1'; AB1 <= '1'; AB2 <= '0'; F <= '0';
          when "0011" =>                        -- Wasserstand zwischen x3 und x2
```

```
            P1 <= '1'; P2 <= '0'; AB1 <= '0'; AB2 <= '1'; F <= '0';
        when "0111" =>                  -- Wasserstand zwischen x2 und x1
            P1 <= '0'; P2 <= '1'; AB1 <= '1'; AB2 <= '1'; F <= '0';
        when "1111" =>                  -- Wasserstand oberhalb von x1
            P1 <= '0'; P2 <= '0'; AB1 <= '1'; AB2 <= '1'; F <= '0';
        when others =>                  -- Schwimmer defekt: Fehlermeldung
            P1 <= '-'; P2 <= '-'; AB1 <= '-'; AB2 <= '-'; F <= '1';
    end case;
  end process schwimmer;
end verhalten;
```

Aufgabe 7: Widerstandsdimensionierung für Gatter mit offenem Kollektor

Gegeben sei ein Gatter in Standard TTL mit Open-Kollektor-Ausgang und einem Pull-Up-Widerstand R_L an U_B = 5V. Der Ausgang werde mit n digitalen Eingängen (Standard TTL) belastet.

Randbedingungen:

- Kollektorströme des Ausgangstransistors:
  - Reststrom des gesperrten Transistors : I_{CRest} = 250 µA
  - Max. Kollektorstrom des leitenden Transistors: I_{Cmax} = 16mA
- Eingangsströme der digitalen Eingänge:
  - H-Pegel: I_{IH} = 40 µA
  - L-Pegel: I_{IL} = -1,6 mA

7.1 Gesucht ist der maximale (R_{Lmax}) und minimale Wert (R_{Lmin}) des Pull-Up-Widerstands für n=5.

7.2 Wie viele Gattereingänge n dürfen den Ausgang maximal belasten?

7.3 Erweitern Sie die Schaltung auf k kollektorseitig verbundene Ausgangstransistoren. Geben Sie für den Fall k=3 und n=5 den Wert für R_{Lmax} und R_{Lmin} an.

Lösung:

7.1 $R_{Lmax} = (U_{CC} - U_{OHmin}) / (n\, I_{IH} + I_{CRest}) = 5{,}77\ k\Omega$

$R_{Lmin} = (U_{CC} - U_{OLmax}) / (n\, I_{IL} + I_{Cmax}) = 575\ \Omega$

7.2 n = 9

7.3 $R_{Lmax} = (U_{CC} - U_{OHmin}) / (5I_{IH} + 3I_{CRest}) = 2{,}74\ k\Omega$

$R_{Lmin} = (U_{CC} - U_{OLmax}) / (5I_{IL} + I_{Cmax}) = 575\ \Omega$

Aufgabe 8: Ansteuerung von Leuchtdioden

Zur Anzeige der Logik-Pegel werden in der Digitaltechnik häufig Leuchtdioden (LEDs) eingesetzt, da sie wenig Strom aufnehmen und wenig Platz benötigen. Für die Ansteuerung der LEDs in TTL-Technik können z.B. Gatter mit Gegentakt-Endstufe (Abb. Ü8.1 a und b) oder mit Open-Kollektor-Ausgang (Abb. Ü8.1 c) eingesetzt werden. Der erforderliche LED-Strom wird mit Hilfe eines Vorwiderstandes eingestellt. Ergänzen Sie beiden Schaltungen und dimensionieren Sie die Vorwiderstände.

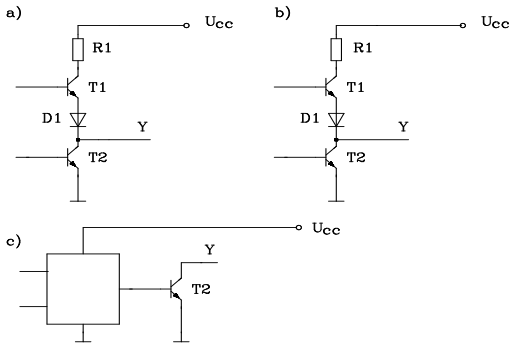


Abb Ü8.1 Vorgegebene Schaltungen zu Aufgabe 8

Anleitung:

Eine rote Leuchtdiode (GaAsP rot) leuchtet bei einem Strom $I_D \approx 10$ mA so hell, dass sie als Indikator zur Anzeige des Logik-Pegels eingesetzt werden kann. Die Spannung an der Leuchtdiode beträgt hierfür etwa 1,7 V. Ist die an der LED anliegende Spannung deutlich kleiner (< 1,5 V), bleibt die Leuchtdiode dunkel. Falls die in Abb. Ü8.1 eingesetzten Transistoren durchgeschaltet sind, beträgt die Kollektor-Emitter-Restspannung U_{CEsat} = 0,3 V. An der Siliziumdiode fällt im durchgeschalteten Zustand eine Spannung von 0,6 V ab. Der Kollektorwiderstand R1 in Schaltung a) und b) hat einen Wert von 130 Ω. Es gilt: U_{CC} = 5 V. I_{CRest} ist vernachlässigt.

Lösung:

Fall a: $R_a = \dfrac{U_{CC} - I_D R_1 - U_{CEsat} - U_{D1} - U_{D2}}{I_D} = 110\ \Omega$

Fall b: $R_b = \dfrac{U_{CC} - U_{CEsat} - U_{D2}}{I_D} = 300\ \Omega$ (gewählt 270Ω)

Fall c: $R_c = R_b = \dfrac{U_{CC} - U_{CEsat} - U_{D2}}{I_D} = 300\ \Omega$ (gewählt 270Ω)

Ein Vergleich der drei Treiberschaltungen zeigt, dass im Fall a) die im treibenden Gatter entstehende Verlustleistung am größten ist. Sie beträgt für die oben angegebenen Werte 22 mW. Für die beiden anderen Fälle b) und c) ist die im trei-

benden Ausgangstransistor entstehende Verlustleistung mit 3 mW verhältnismäßig klein.

$P_{Va} = I_D^2 * R1 + (U_{CEsat} + U_{D1})I_D = 13\ mW + 9\ mW = 22\ mW$

$P_{Vb} = P_{Vc} = I_D * U_{CEsat} = 3\ mW$

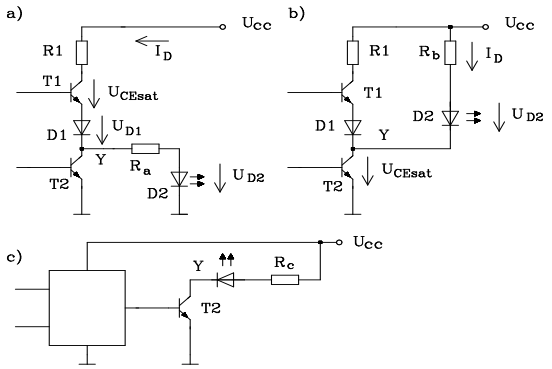


Abb Ü8.2 Treiberschaltungen zu Aufgabe 8 mit den erforderlichen Widerständen und LEDs

Aufgabe 9: VHDL-Entwurf eines Addierers – Test mit einer Testbench [VHDL]*

Entwerfen Sie einen 6-Bit-Ripple-Carry-Addierer und testen Sie ihn mit einer Testbench unter Einsatz von Testvektoren.

Anleitung: Deklarieren Sie die Komponenten für einen Volladdierer und einen parametrisierbaren n-Bit-Ripple-Carry-Addierer (s. Kap. 5.3). Legen Sie die Komponenten in einem Package ab. Danach entwerfen Sie eine Testbench mit Testvektoren (Kap. 4.9.2).

Lösung:

```
-- adder_pack.vhd
-- Volladdierer: full_adder und n-Bit-Addierer: ripadd_n
package adder_pack is
  component full_adder -- Componenten-Deklaration des 1-Bit-Volladdierers
     port (c_in: in bit;            -- Uebertrag-Eingang
           a1,b1:      in  bit; -- Bitstellen der Zahlen a und b
           sum1:       out bit; -- Summe
           c_out:      out bit); -- Uebertrag-Ausgang
  end component;

  component ripadd_n   -- Componenten-Deklaration des n-Bit-Addierers
     generic (n: integer := 8);
     port (ci:          in bit;
        a,b:            in bit_vector(n-1 downto 0);-- zu addierende Zahlen
```

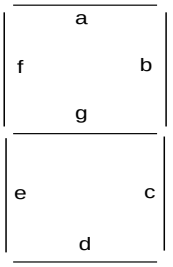
```
        summe : out bit_vector (n-1 downto 0);
        co: out bit);
  end component;
end adder_pack;

-- Modell des 1-Bit-Volladdierers
entity full_adder is
     port (c_in: in bit;            -- Uebertrag-Eingang
          a1,b1:        in  bit; -- Bitstellen der Zahlen a und b
          sum1:         out bit; -- Summe
          c_out:        out bit); -- Uebertrag- Ausgang
end full_adder;
architecture logik_full_adder of full_adder is
constant tpd1: time := 10 ns;              -- Verzögerungszeiten fuer die Simulation
constant tpd2: time := 8 ns;
begin
     sum1 <= a1 xor (b1 xor c_in) after tpd1;
     c_out <= ((a1 xor b1) and c_in) or (a1 and b1) after tpd2;
end logik_full_adder;

-- Modell des n-Bit-Addierers
use work.adder_pack.all;
entity ripadd_n is
     generic (n: integer := 8);
     port (ci:          in bit;
        a,b:            in bit_vector(n-1 downto 0);-- zu addierende Zahlen
        summe : out bit_vector (n-1 downto 0);
        co: out bit);
end ripadd_n;
architecture adder_n of ripadd_n is
signal c: bit_vector (n downto 0);         -- Zwischengroessen fuer Netzliste
begin
     c(0) <= ci;
     add: for i in 0 to n-1 generate       -- Instanziierung mit Generate-Anweisung
     addi: full_adder port map(c(i), a(i), b(i), summe(i), c(i+1));
     end generate;
     co <= c(n);
end adder_n;
-- Testbench fuer 6-Bit-Addierer
-- Datum: 19.02.2003
use work.adder_pack.all;
entity tb_add6 is
end tb_add6;
architecture auto_test of tb_add6 is
     signal ci,co,c_in,c_o: bit;
     signal a,b,summe,ergebnis: bit_vector(5 downto 0);
type test_vektor is record
     c_in: bit;
     a: bit_vector(5 downto 0);
     b: bit_vector(5 downto 0);
     ergebnis: bit_vector(5 downto 0);
```

```
    c_o: bit;
end record;

type test_vektor_array is array (natural range <>) of test_vektor;
constant test_feld: test_vektor_array:=(
(c_in => '1',a => "100010",b => "000000",ergebnis => "100011", c_o => '0'),
(c_in => '0',a => "111110",b => "000011",ergebnis => "000001", c_o => '1'),
(c_in => '1',a => "101000",b => "111111",ergebnis => "101000", c_o => '1'),
(c_in => '0',a => "100010",b => "000000",ergebnis => "100001", c_o => '0'),  -- Fehler
(c_in => '1',a => "101000",b => "000011",ergebnis => "100111", c_o => '1')   -- Fehler
);

begin
dut: ripadd_n                    -- instanziieren der Werte
    generic map (n => 6)
    port map (ci,a,b,summe,co);
testen: process        -- Testvektor zuweisen
    variable vektor: test_vektor;
    variable fehler: boolean := false;
begin
    for i in test_feld'range loop
        vektor := test_feld(i);
        ci <= vektor.c_in;
        a <= vektor.a;
        b <= vektor.b;
    wait for 100 ns;
    if summe /= vektor.ergebnis or co /= vektor.c_o then
        assert false
        report "Addierer defekt";
        fehler := true;
    end if;
end loop;

    assert not fehler
    report "Gesamttest ist fehlerhaft"
    severity note;
    assert fehler
    report "Gesamttest ist o.k"
    severity note;
    wait;
end process;
end auto_test;
```

Ausführung mit ModelSim (Xilinx):

```
 run -all
 # ** Error: Addierer defekt
 #    Time: 400 ns  Iteration: 0  Instance: /tb_add6
 # ** Error: Addierer defekt
 #    Time: 500 ns  Iteration: 0  Instance: /tb_add6
 # ** Note: Gesamttest ist fehlerhaft
 #    Time: 500 ns  Iteration: 0  Instance: /tb_add6
```

Aufgabe 10: Darstellung von Hexadezimalziffern auf einer 7-Segment-Anzeige [VHDL]*



7-Segment-Anzeige

Abb. Ü10.1 7-Segmentanzeige

Gegeben ist eine 7-Segmentanzeige mit den Segmenten a,b,c,d,e,f und g. Die Leuchtsegmente werden so kombiniert, dass eine Dezimalziffer oder ein Sonderzeichen dargestellt werden kann.

Mittels der skizzierten 7-Segmentanzeige sollen die in Tabelle Ü10.1 gegebenen hexadezimalen Ziffern in Abhängigkeit der Eingangsvariablen X1,X2,X3 und X4 dargestellt werden. Entwerfen Sie dafür eine minimale Schaltung und geben Sie die logischen Gleichungen in disjunktiver Minimalform an. Es gilt folgende Zuordnung: Ein Segment der Anzeige leuchtet, wenn die zugehörige Steuervariable (Sa, Sb, Sc, Sd, Se, Sf, Sg) den Logik-Zustand 1 annimmt.

Tabelle Ü10.1 Zuordnung zwischen den Eingangsvariablen und den Hex-Ziffern

X1	X2	X3	X4	Zeichen	Sa Sb Sc Sd Se Sf Sg
0	0	0	0	0	
0	0	0	1	1	
0	0	1	0	2	
0	0	1	1	3	
0	1	0	0	4	
0	1	0	1	5	
0	1	1	0	6	
0	1	1	1	7	
1	0	0	0	8	
1	0	0	1	9	
1	0	1	0	A	
1	0	1	1	b	
1	1	0	0	C	
1	1	0	1	d	
1	1	1	0	E	
1	1	1	1	F	

Lösung:

Nichtnegierte disjunktive Minimalform:

$Sa = X1\ \overline{X4} \vee X2\ X3 \vee \overline{X2}\ \overline{X4} \vee \overline{X1}\ X3 \vee X1\ \overline{X2}\ \overline{X3} \vee \overline{X1}\ X2\ X4$

$Sb = \overline{X1}\ \overline{X2} \vee \overline{X2}\ \overline{X4} \vee \overline{X1}\ \overline{X3}\ \overline{X4} \vee X1\ \overline{X3}\ X4 \vee \overline{X1}\ X3\ X4$

$Sc = X1\ \overline{X2} \vee \overline{X1}\ X2 \vee \overline{X1}\ \overline{X3} \vee \overline{X1}\ X4 \vee \overline{X3}\ X4$

$Sd = X1\ \overline{X3} \vee X2\ \overline{X3}\ X4 \vee X2\ X3\ \overline{X4} \vee \overline{X2}\ X3\ X4 \vee \overline{X1}\ \overline{X2}\ \overline{X4}$

$Se = \overline{X2}\ \overline{X4} \vee X3\ \overline{X4} \vee X1\ X3 \vee X1\ X2$

$Sf = \overline{X3}\ \overline{X4}\ v\ X2\ \overline{X4}\ v\ X1\ X3\ v\ X1\ \overline{X2}\ v\ \overline{X1}\ X2\ \overline{X3}$

$Sg = X1\ X4\ v\ \overline{X2}\ X3\ v\ X3\ \overline{X4}\ v\ X1\ \overline{X2}\ v\ \overline{X1}\ X2\ \overline{X3}$

VHDL-Modell: Hexadezimalziffern auf 7-Segment

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

entity uebung_10 is port(
   x:                    in std_logic_vector (1 to 4);
   sa,sb,sc,sd,se,sf,sg:  out std_logic);
end uebung_10;

architecture verhalten of uebung_10 is
  signal y: std_logic_vector (6 downto 0);          -- y ist Zwischengroesse
begin

  wahrheits_tab: process(x) begin
    case x is
      when "0000" => y <= "1111110";
      when "0001" => y <= "0110000";
      when "0010" => y <= "1101101";
      when "0011" => y <= "1111001";
      when "0100" => y <= "0110011";
      when "0101" => y <= "1011011";
      when "0110" => y <= "1011111";
      when "0111" => y <= "1110000";
      when "1000" => y <= "1111111";
      when "1001" => y <= "1111011";
      when "1010" => y <= "1110111";
      when "1011" => y <= "0011111";
      when "1100" => y <= "1001110";
      when "1101" => y <= "0111101";
      when "1110" => y <= "1001111";
      when "1111" => y <= "1000111";
      when others => y <= "-------";                -- neunwertige Logik
   end case;

end process wahrheits_tab;
   sa <= y(6); sb <= y(5); sc <= y(4); sd <= y(3); -- nebenlaeufige Anweisungen
   se <= y(2); sf <= y(1); sg <= y(0);
end verhalten;
```

Aufgabe 11: Zustands- und flankengesteuertes D-Flipflop

Ein zustands- und ein positiv flankengesteuertes D-Flipflop werden von einem Signal X am D-Eingang und Φ am Takteingang angesteuert. Der Ausgang des zustandsgesteuerten D-Flipflops sei Qz und der des flankengesteuerten Qf. Vervollständigen Sie den skizzierten Signalzeitplan und geben Sie Unterschiede der beiden Flipflops an.

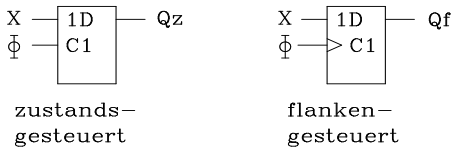


Abb. Ü11.1 Ansteuerung der beiden D-Flipflops

Anmerkung: Zu Beginn der Betrachtungen seien beide Flipflops zurückgesetzt. Die Verzögerungszeiten der Flipflops seien vernachlässigbar.

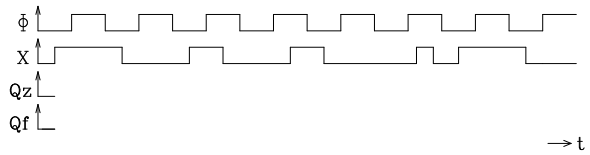


Abb. Ü11.2 Vorgegebener Signalverlauf des Taktes und Eingangssignals

Lösung:

Ein zustandsgesteuertes D-Flipflop ist für Φ = 1 transparent, d. h. eine Änderung am D-Eingang wirkt sich direkt am Ausgang aus (Verzögerungszeiten seien vernachlässigbar). Während der Takt Φ im Logik-Zustand 0 ist, speichert das zustandsgesteuerte D-Flipflop den Wert.

Dagegen übernimmt das flankengesteuerte D-Flipflop den Logik-Zustand am D-Eingang mit der positiven Taktflanke und speichert ihn bis zur nächstfolgenden positiven Taktflanke.

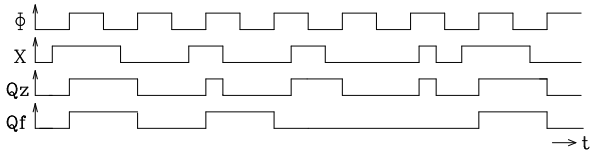


Abb. Ü11.3 Vollständiger Signalzeitplan zu Aufgabe 11

Aufgabe 12: Analyse eines Schaltwerks mit D-Flipflops

In einer digitalen Schaltung sollen drei flankengesteuerte D-Flipflops eingesetzt werden.

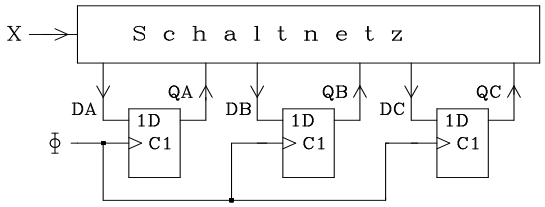


Abb. Ü12.1 Blockschaltbild zu Aufgabe 12

Die logischen Gleichungen für die D-Eingänge lauten:

$DA = \overline{QA}$

$DB = \overline{X}\,\overline{QC}\,\overline{QB}\,QA \vee \overline{X}\,QB\,\overline{QA} \vee X\,QB\,QA \vee X\,QC\,\overline{QA}$

$DC = X\,\overline{QC}\,\overline{QB}\,\overline{QA} \vee \overline{X}\,QB\,QA \vee \overline{X}\,QC\,\overline{QA} \vee X\,QC\,QA$

QA, QB, QC sind die Ausgänge der drei D-Flipflops und X ist eine Eingangsvariable (Abb. Ü12.1 und Ü12.2). Die Ausgänge QA, QB und QC haben die Wertigkeit 2^0, 2^1 und 2^2. Vervollständigen Sie den in Abb. Ü12.2 gegebenen Signalzeitplan und charakterisieren Sie die Schaltung.

Für t = 0 sollen die drei D-Flipflops rückgesetzt sein.

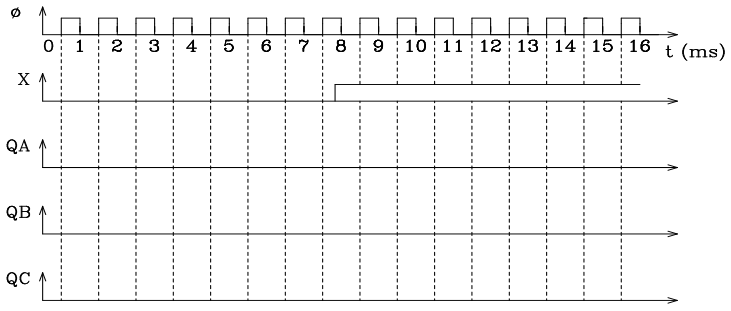


Abb. Ü12.2 Signalzeitplan zu Aufgabe 12

Lösung:

Das erste D-Flipflop mit dem Ausgang QA ist als Frequenzteiler (1:2) geschaltet. Der zeitliche Verlauf der Ausgangsgröße QA kann unabhängig von X und den Ausgängen QB und QC bestimmt werden (Abb. Ü12.3). Da QB und QC von X, QA, QB und QC abhängen, kann der zeitliche Verlauf dieser beiden Flipflopausgänge nur für eine Taktperiode zwischen zwei benachbarten positiven Taktflanken

bestimmt werden. Dann werden die neu ermittelten Logik-Zustände wieder in die Gleichungen eingesetzt und der Signalverlauf für die nächste Periode gewonnen, usw. (Abb. Ü12.3).

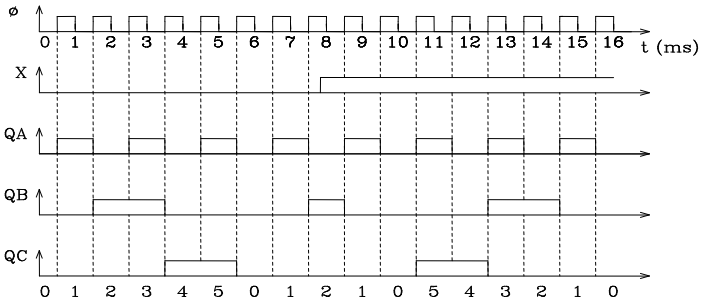


Abb. Ü12.3 Vollständiger Signalzeitplan zu Aufgabe 12

Die Schaltung stellt einen synchronen Vorwärts-/Rückwärts-Modulo-6-Zähler dar, der über das Eingangssignal X in der Zählrichtung umgeschaltet werden kann. Der Zähler zählt für X = 0 vorwärts und für X = 1 rückwärts.

13: Entwurf eines JK- und eines T-Flipflops mit Hilfe eines D-Flipflops

Entwerfen Sie mit Hilfe eines flankengesteuerten D-Flipflops und eines Schaltnetzes ein flankengesteuertes JK- und T-Flipflop.

Lösung: Es gelten folgende Übergangsbedingungen:

D-Flipflop	JK-Flipflop	T-Flipflop
$Q^* = D$ (Gl.1)	$Q^* = J\,\overline{Q} \vee \overline{K}\,Q$ (Gl.2)	$Q^* = T\,\overline{Q} \vee \overline{T}\,Q$ (Gl.3)

a) Entwurf des JK-Flipflops: Gleichsetzen der Übergangsbedingungen nach Gl.1 und Gl.2: $D = J\,\overline{Q} \vee \overline{K}\,Q$

b) Entwurf des T-Flipflops: Gleichsetzen von Gl.1 und Gl.3: $D = T\,\overline{Q} \vee \overline{T}\,Q$

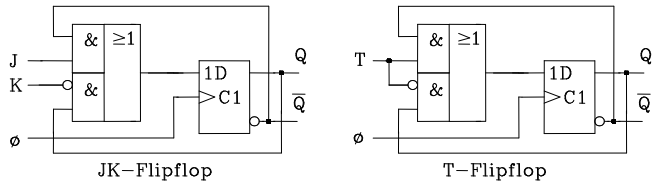


Abb. Ü13.1 Realisierung eines JK- und eines T-Flipflops mit Hilfe eines D-Flipflops

Aufgabe 14: Steuerung einer Ampelanlage [VHDL]*

Eine Straßenkreuzung mit Haupt- und Nebenstrecke soll von einer Ampelanlage gesteuert werden. Für die Lichtsignale der Hauptstrecke soll gelten: 14 s grün, 2 s gelb, 14 s rot, 2 s rotgelb, usw.. Für die Lichtsignale der Nebenstrecke soll gelten: 6 s grün, 2 s gelb, 22 s rot, 2 s rotgelb, usw.. Beim Umschalten sollen sich die Rotphasen beider Ampelanlagen für 2 s überlappen. Daraus folgt, dass innerhalb eines Ampelzyklus beide Ampelanlagen zweimal für je 2 s gleichzeitig rot geschaltet sind.

Zum Entwurf werde ein Zähler eingesetzt. Dessen Ausgänge sollen über ein Schaltnetz die sechs Lampen der beiden Ampelanlagen ansteuern.

14.1 Geben Sie den Zählertyp und die erforderliche Taktfrequenz an.

14.2 Geben Sie in Form einer Wahrheitstabelle die Abhängigkeit der sechs Steuervariablen vom Zählerstand an, beginnen Sie beim Zählerstand 0 mit der Grünphase der Hauptstrecke.

14.3 Stellen Sie die logischen Gleichungen für die sechs Steuervariablen auf. Zum Entwurf soll ein PAL mit nichtnegierten Ausgängen eingesetzt werden. Minimieren Sie die Gleichungen, so dass die Anzahl der erforderlichen Produktterme (UND-Verknüpfungen) des PALs minimal wird.

Lösung zu 14.1:

Da ein Lichtsignalzyklus der Haupt- und Nebenstrecke 32 s benötigt und die kleinste Zeiteinheit 2 s beträgt, ist eine Zählerkapazität von 32 s/2 s = 16 nötig. Gewählt wird ein asynchroner Vorwärts-Dualzähler mit einer Taktfrequenz von 0,5 Hz. Der Zähler kann asynchron über ¬R rückgesetzt werden (Abb Ü14.3).

Lösung zu 14.2:

Tabelle Ü14.1 Wahrheitstabelle mit den geforderten Steuervariablen

	QD	QC	QB	QA	GRÜN_H	ROT_H	GELB_H	GRÜN_N	ROT_N	GELB_N
0	0	0	0	0	1	0	0	0	1	0
1	0	0	0	1	1	0	0	0	1	0
2	0	0	1	0	1	0	0	0	1	0
3	0	0	1	1	1	0	0	0	1	0
4	0	1	0	0	1	0	0	0	1	0
5	0	1	0	1	1	0	0	0	1	0
6	0	1	1	0	1	0	0	0	1	0
7	0	1	1	1	0	0	1	0	1	0
8	1	0	0	0	0	1	0	0	1	0
9	1	0	0	1	0	1	0	0	1	1
10	1	0	1	0	0	1	0	1	0	0
11	1	0	1	1	0	1	0	1	0	0
12	1	1	0	0	0	1	0	1	0	0
13	1	1	0	1	0	1	0	0	0	1
14	1	1	1	0	0	1	0	0	1	0
15	1	1	1	1	0	1	1	0	1	0

Lösung zu 14.3:

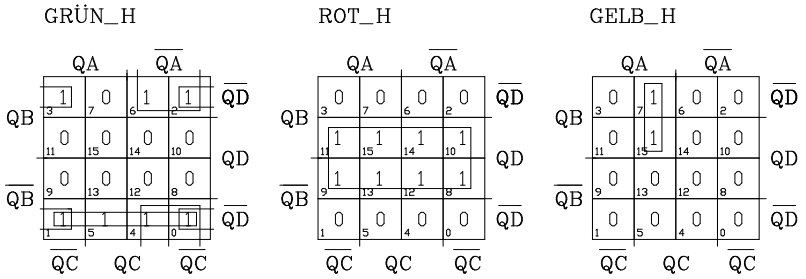


Abb. Ü14.1 KV-Diagramme zur Minimierung der logischen Gleichungen GRÜN_H, ROT_H und GELB_H

GRÜN_H = (0)∨(1)∨(2)∨(3)∨(4)∨(5)∨(6) = $\overline{QD}\ \overline{QB} \vee \overline{QD}\ \overline{QC} \vee \overline{QD}\ \overline{QA}$

ROT_H = (8)∨(9)∨(10)∨(11)∨(12)∨(13)∨(14)∨(15) = QD

GELB_H = (7)∨(15) = QC QB QA

GRÜN_N = (10)∨(11)∨(12) = QD QB $\overline{QC}$ ∨ QD QC $\overline{QB}\ \overline{QA}$

ROT_N = (0)∨(1)∨(2)∨(3)∨(4)∨(5)∨(6)∨(7)∨(8)∨(9)∨(14)∨(15) = $\overline{QD} \vee \overline{QC}\ \overline{QB} \vee$ QC QB

GELB_N = (9)∨(13) = QD $\overline{QB}$ QA

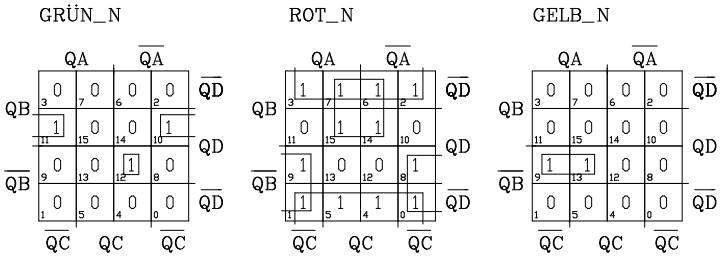


Abb. Ü14.2 KV-Diagramme zur Minimierung der logischen Gleichungen GRÜN_N, ROT_N und GELB_N

In Abb. Ü14.3 ist das Blockschaltbild der Ampelsteuerung abgebildet. Der eingesetzte Zähler kann über ¬R = 0 rückgesetzt werden. Dadurch wird die Grünphase für die Hauptstrecke eingeleitet.

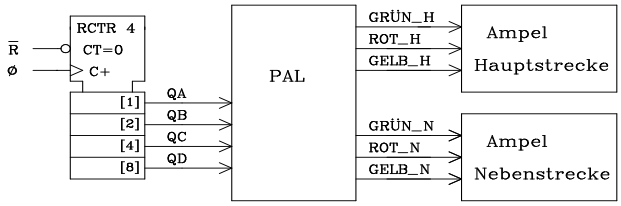


Abb. Ü14.3: Blockschaltbild der Ampelsteuerung

VHDL-Modell: Ampelanlage

```
library ieee; use ieee.std_logic_1164.all; use work.std_arith.all;
entity ampel is port (
        clk, r_neg: in  std_logic;          -- r_neg = reset nicht
          gruen_h, rot_h, gelb_h,gruen_n, rot_n, gelb_n:   out std_logic);
end ampel;

architecture verhalten of ampel is      -- Architektur mit 2 Prozessen
        signal zaehl: std_logic_vector(3 downto 0);       -- interner Zaehler

begin
zaehler: process (r_neg, clk)               -- Prozess zur Modellierung des Zaehlers

begin
        if r_neg='0' then  zaehl <= "0000";      -- asynchroner Reset
        elsif (clk'event and clk='1') then  zaehl <= zaehl +1;
        end if;
end process zaehler;
ausgabe: process (zaehl)                    -- Prozess fuer die Ausgabe
```

```
begin
        if (zaehl < "0111")   then    gruen_h <= '1'; rot_h <= '0'; gelb_h <= '0';
                                      gruen_n <= '0'; rot_n <= '1'; gelb_n <= '0';
        elsif (zaehl = "0111") then   gruen_h <= '0'; rot_h <= '0'; gelb_h <= '1';
                                      gruen_n <= '0'; rot_n <= '1'; gelb_n <= '0';
        elsif (zaehl = "1000") then   gruen_h <= '0'; rot_h <= '1'; gelb_h <= '0';
                                      gruen_n <= '0'; rot_n <= '1'; gelb_n <= '0';
        elsif (zaehl = "1001") then   gruen_h <= '0'; rot_h <= '1'; gelb_h <= '0';
                                      gruen_n <= '0'; rot_n <= '1'; gelb_n <= '1';
        elsif (zaehl > "1001" and zaehl < "1101") then
                                      gruen_h <= '0'; rot_h <= '1'; gelb_h <= '0';
                                      gruen_n <= '1'; rot_n <= '0'; gelb_n <= '0';
        elsif (zaehl = "1101") then   gruen_h <= '0'; rot_h <= '1'; gelb_h <= '0';
                                      gruen_n <= '0'; rot_n <= '0'; gelb_n <= '1';
        elsif (zaehl = "1110") then   gruen_h <= '0'; rot_h <= '1'; gelb_h <= '0';
                                      gruen_n <= '0'; rot_n <= '1'; gelb_n <= '0';
        elsif (zaehl = "1111") then   gruen_h <= '0'; rot_h <= '1'; gelb_h <= '1';
                                      gruen_n <= '0'; rot_n <= '1'; gelb_n <= '0';
        end if;

end process ausgabe;
end verhalten;
```

Aufgabe 15: Testbench für einen synchronen Dualzähler *[VHDL]**

Entwerfen Sie eine Testbench für einen synchronen 4-Bit Dualzähler mit asynchronem Reset. Verwenden Sie sowohl eine Testbench mit Testvektoren als auch eine Testbench mit Ein- und Ausgabedatei.

Lösung:

Im VHDL-Entwurf wird an einer Stelle ein Fehler eingebaut und die Reaktion auf diesen Fehler wird dokumentiert.

1. Testbench mit Testvektoren (Kap. 4.9.2)

```
-- zaehler_pack.vhd
-- Package zaehler_pack mit der Komponente bin_4
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned."+";        -- Erweiterung des Operators "+" auf Vektoren

package zaehler_pack is
component bin_4 port (
    clk, reset:       in std_logic;
    q:     buffer std_logic_vector(3 downto 0));
end component;
end zaehler_pack;

-- 4-Bit-Dualzaehler mit asynchronem Reset
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned."+";
```

```
entity bin_4 is port (
     clk, reset:         in std_logic;
     q:      buffer std_logic_vector(3 downto 0));
end bin_4;
architecture archcounter of bin_4 is
begin
     zaehler: process (reset, clk)
     begin
       if reset = '0' then
             q <=  "0000";
       elsif (clk'event and clk = '0') then  -- negative Flanke ist aktiv
             q <= q + 1;
             if q >= "0110" then q <= "0000";          -- eingebauter Fehler "modulo-7-zaehler"
             end if;
       end if;
     end process zaehler;
end archcounter;

-- test_bench_zaehl.vhd
-- Testbench fuer 4-Bit-Dualzaehler mit asynchronem Reset
library ieee;
use ieee.std_logic_1164.all;
use work.zaehler_pack.all;

entity test_bench_zaehl is
end test_bench_zaehl;

architecture auto_test of test_bench_zaehl is
     signal clk,reset: std_logic;
     signal q: std_logic_vector(3 downto 0);
type test_vektor is record
     clk: std_logic;
     reset: std_logic;
     q: std_logic_vector(3 downto 0);
end record;
type test_vektor_array is array (natural range <>) of test_vektor;
constant test_feld: test_vektor_array:=(
     (clk => '0',reset => '0',q => "0000"),              -- neg. Taktflanke aktiv
     (clk => '1',reset => '1',q => "0000"),
     (clk => '0',reset => '1',q => "0001"),
     (clk => '1',reset => '1',q => "0001"),
     (clk => '0',reset => '1',q => "0010"),
     (clk => '1',reset => '1',q => "0010"),
     (clk => '0',reset => '1',q => "0011"),
     (clk => '1',reset => '1',q => "0011"),
     (clk => '0',reset => '1',q => "0100"),
     (clk => '1',reset => '1',q => "0100"),
     (clk => '0',reset => '1',q => "0101"),
     (clk => '1',reset => '1',q => "0101"),
     (clk => '0',reset => '1',q => "0110"),
     (clk => '1',reset => '1',q => "0110"),
     (clk => '0',reset => '1',q => "0111"),
     (clk => '1',reset => '1',q => "0111"),
     (clk => '0',reset => '1',q => "1000"),
```

```
    (clk => '1',reset => '1',q => "1000")
);
begin      -- instanziieren der component bin_4
dut: bin_4 port map (clk, reset, q);
testen: process       -- Testvektor zuweisen und pruefen
    variable vektor: test_vektor;
    variable fehler: boolean := false;
begin
    for i in test_feld'range loop
    vektor := test_feld(i);
    clk <= vektor.clk;
    reset <= vektor.reset;
    wait for 20 ns;  -- Vergleich nach 20ns
    if q /= vektor.q then
        assert false
        report "Zaehler reagiert falsch"; -- zeitabhängige Fehlermeldung
        fehler := true;
    end if;
    wait for 30 ns;  -- Taktperiode = 100 ns = 2 x (20+30)ns
end loop;
-- Ausgabe eines Reports nach Testende
    assert not fehler
    report "Gesamttest ist fehlerhaft"
    severity note;
    assert fehler
    report "Gesamttest ist o.K"
    severity note;
    wait;
end process testen;
end auto_test;
```

Fehlerprotokoll des Compiler ModelSim:

```
# ** Error: Zaehler reagiert falsch
#    Time: 720 ns  Iteration: 0  Instance: /test_bench_zaehl
# ** Error: Zaehler reagiert falsch
#    Time: 770 ns  Iteration: 0  Instance: /test_bench_zaehl
# ** Error: Zaehler reagiert falsch
#    Time: 820 ns  Iteration: 0  Instance: /test_bench_zaehl
# ** Error: Zaehler reagiert falsch
#    Time: 870 ns  Iteration: 0  Instance: /test_bench_zaehl
# ** Note: Gesamttest ist fehlerhaft
#    Time: 900 ns  Iteration: 0  Instance: /test_bench_zaehl
```

Nachdem der Fehler beseitigt ist, wird erneut compiliert und mit dem Run-Kommando ausgeführt:

Protokoll des Compiler ModelSim mit fehlerfreiem VHDL-Modell:

```
run -all
# ** Note: Gesamttest ist o.K
#    Time: 900 ns  Iteration: 0  Instance: /test_bench_zaehl
```

2. Testbench mit Ein- und Ausgabedatei

Beim zweiten Test wird - wie in Kap. 4.9.3 erläutert - die Eingabe der Stimuli über eine formatierte Textdatei vorgenommen. In gleicher Weise werden die Testergebnisse formatiert in einer Ausgabedatei abgelegt. Im folgenden Beispiel wird vorausgesetzt, dass die compilierten Packages zaehler_pack und text_io in der Work-Library gespeichert sind.

```
-- tb_zaehl_io.vhd
-- Testbench mit Ein- und Ausgabedatei

library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;
use work.text_io_pack.all;     -- siehe Kap. 4.9.3
use work.zaehler_pack.all;
entity tb_textio is                -- Testbench
end tb_textio;

architecture auto_test of tb_textio is
    signal clk,reset: std_logic;
    signal q: std_logic_vector(3 downto 0);
begin
dut: bin_4 port map (clk, reset, q);
testen: process
    file eingabe_datei: text is in "zaehl_ein.txt";          -- Stimuli-Eingabe
    file ausgabe_datei: text is out "zaehl_aus.txt";         -- Ergebnis-Ausgabe
    variable zeile_ein,zeile_aus: line;
    variable v_clk: std_logic;
    variable v_reset: std_logic;
    variable v_q: std_logic_vector(3 downto 0);
    variable aus_q: std_logic_vector(3 downto 0);
    variable fehler: boolean := false;
    variable gut: boolean;
    variable char: character;
    variable fehler_aus: string(1 to 4) := "nein";
    constant abstand_2: string(1 to 2) := "  ";
    constant abstand_3: string(1 to 3) := "   ";
    constant abstand_4: string(1 to 4) := "    ";
-- Ueberschrift der Ausgabedatei
    constant ueber: string(1 to 29) := "clk reset   q    qsoll Fehler";
begin
        write(zeile_aus, ueber);        -- Ueberschriftausgabe
        writeline(ausgabe_datei, zeile_aus); -- Ausgabe der Zeile
        writeline(ausgabe_datei, zeile_aus); -- Ausgabe einer Leerzeile

zeile_loop:          while not endfile(eingabe_datei) loop
    readline (eingabe_datei,zeile_ein); -- Zeile einlesen
    -- Zeile auswerten: Uebergabe an Signale
    -- ueberspringe Zeile, falls Zeichen kein Tabulator
    read (zeile_ein,char,gut);
    if not gut or char /= HT then next;
    end if;
    assert gut
```

```
    report "Fehler beim Lesen"
    severity note;
    read (zeile_ein,v_clk,gut);
    next when not gut;
    read (zeile_ein,char);
    read (zeile_ein,v_reset,gut);
    next when not gut;
    read (zeile_ein,char);
    read (zeile_ein,v_q,gut);
    next when not gut;
    clk <= v_clk;   -- Typ-Konvertierung: variable --> signal
    reset <= v_reset;
    wait for 20 ns;
-- ueberpruefen des Ergebnisses
    aus_q := q;
    if aus_q /= v_q then
        assert false
        report "Zaehler reagiert falsch";
        fehler := true;
        fehler_aus := " ja ";
    end if;
-- formatierte Ausgabe
    write(zeile_aus, ' ');
    write(zeile_aus, v_clk);
    write(zeile_aus, abstand_4);
    write(zeile_aus, v_reset);
    write(zeile_aus, abstand_3);
    write(zeile_aus, aus_q);
    write(zeile_aus, abstand_3);
    write(zeile_aus, v_q);
    write(zeile_aus, abstand_3);
    write(zeile_aus, fehler_aus);
    writeline(ausgabe_datei, zeile_aus);
        fehler_aus := "nein";
    wait for 30 ns;  -- Taktperiode = 100 ns = 2 x (20+30)ns
end loop zeile_loop;
-- Ausgabe eines Reports
    assert not fehler
    report "Gesamtbewertung: Zaehler ist fehlerhaft"
    severity note;
    assert fehler
    report "Gesamtbewertung: Zaehler ist o.K"
    severity note;
    wait;
end process testen;
end auto_test;
```

Fehlerprotokoll ModelSim: (s. o)

Tabelle Ü15.1 Ein- und Ausgabedatei, die innerhalb der Testbench verwendet werden

eingabe_datei: zaehl_ein.txt	ausgabe_datei: zaehl_aus.txt				
4-Bit-Binärzähler mit asnchronem Reset	clk	reset	q	qsoll	Fehler
0 0 0000	0	0	0000	0000	nein
1 1 0000	1	1	0000	0000	nein
0 1 0001	0	1	0001	0001	nein
1 1 0001	1	1	0001	0001	nein
0 1 0010	0	1	0010	0010	nein
1 1 0010	1	1	0010	0010	nein
0 1 0011	0	1	0011	0011	nein
1 1 0011	1	1	0011	0011	nein
0 1 0100	0	1	0100	0100	nein
1 1 0100	1	1	0100	0100	nein
0 1 0101	0	1	0101	0101	nein
1 1 0101	1	1	0101	0101	nein
0 1 0110	0	1	0110	0110	nein
1 1 0110	1	1	0110	0110	nein
0 1 0111	0	1	0000	0111	ja
1 1 0111	1	1	0000	0111	ja
0 1 1000	0	1	0001	1000	ja
1 1 1000	1	1	0001	1000	ja

Aufgabe 16: VHDL-Entwurf des programmierbaren Synchronzählers 74163 [VHDL]*

Entwerfen Sie das VHDL-Modell des programmierbaren synchronen 4-Bit-Dualzählers 74163. Eine detaillierte Beschreibung des Zählertyps finden Sie in Kapitel 6.2.2.1.

Lösung:

```
-- Ladbarer 4-Bit-Binaerzaehler: Typ SN74163
-- Steuereingaenge: clr, load, ent, enp
-- Takteingang clk
-- Ladeeingaenge: d (4-Bit-Vektor)
-- Ausgaenge: q (4-Bit-Vektor) und Uebertrag rco

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.''+'';

entity SN74163 is port (
      clr, load, ent, enp:         in std_logic;
    clk:   in std_logic;
    d: in std_logic_vector(3 downto 0);
    q:      buffer std_logic_vector(3 downto 0);
    rco: out std_logic);
end SN74163 ;
```

```
architecture arch_bin4 of SN74163 is

constant tco: time := 8 ns;
constant tpd: time := 10 ns;

begin
zaehlen:  process (clk)
    begin
     if (clk'event and clk = '1') then
            if clr = '0' then        -- synchroner Reset
             q <= (others => '0') after tco;
            elsif load = '0' then q <= d after tco;
             elsif (ent = '1' and enp = '1') then -- zaehlt, falls beide enable aktiv
             q <= q + 1 after tco;
            end if;
     end if;
  end process zaehlen;

ueberlauf: rco <= '1' after tpd when q = 15 and ent = '1' else
     '0' after tpd;
end arch_bin4;
```

Aufgabe 17: Synchroner Modulo-5-Zähler [VHDL]*

Geben Sie zu den drei Unterpunkten jeweils die nichtnegierten disjunktiven Minimalformen an.

17.1 Entwerfen Sie mit Hilfe von D-Flipflops einen synchronen Modulo-5 Zähler. Der Zähler soll vorwärts zählen.
17.2 Entwerfen Sie mit Hilfe von D-Flipflops einen synchronen Modulo-5 Zähler. Der Zähler soll rückwärts zählen.
17.3 Entwerfen Sie mit Hilfe von D-Flipflops einen umschaltbaren synchronen Modulo-5 Zähler. Mit UM = 0 soll vorwärts und mit UM = 1 soll rückwärts gezählt werden.

Lösung:

17.1 Vorwärtszähler

Tabelle Ü17.1 Wahrheitstabelle für den synchronen Modulo-5-Vorwärtszähler

QC	QB	QA	Zahl	QC^*	QB^*	QA^*
0	0	0	0	0	0	1
0	0	1	1	0	1	0
0	1	0	2	0	1	1
0	1	1	3	1	0	0
1	0	0	4	0	0	0
1	0	1	5	*	*	*
1	1	0	6	*	*	*
1	1	1	7	*	*	*

Übergangsbedingung für das D-Flipflop: D = Q*

$$DA = \mathrm{QA}^* = (0) \vee (2) = \overline{QA}\,\overline{QC}$$

$$DB = \mathrm{QB}^* = (1) \vee (2) = QA\overline{QB} \vee \overline{QA}QB$$

$$DC = \mathrm{QC}^* = (3) = QAQB$$

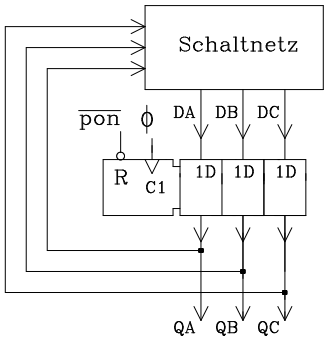


Abb. Ü17.1 Schaltung des Modulo-5-Vorwärtszählers

17.2 Rückwärtszähler

Tabelle Ü17.2 Wahrheitstabelle für den synchronen Modulo-5-Rückwärtszähler

QC	QB	QA	Zahl	QC*	QB*	QA*
0	0	0	0	1	0	0
0	0	1	1	0	0	0
0	1	0	2	0	0	1
0	1	1	3	0	1	0
1	0	0	4	0	1	1
1	0	1	5	*	*	*
1	1	0	6	*	*	*
1	1	1	7	*	*	*

Übergangsbedingung für das D-Flipflop: D = Q*

$$DA = \mathrm{QA}^* = (2) \vee (4) = QC \vee \overline{QA}QB$$

$$DB = \mathrm{QB}^* = (3) \vee (4) = QC \vee QAQB$$

$$DC = \mathrm{QC}^* = (0) = \overline{QA}\,\overline{QB}\,\overline{QC}$$

Es gilt die Schaltung nach Abb. Ü17.1.

17.3 Umschaltbarer Vor-/Rückwärtszähler

Für die Umschaltvariable UM wird in der Wahrheitstabelle eine zusätzliche Eingangsgröße berücksichtigt.

Übergangsbedingung für das D-Flipflop: D = Q*

$$DA = \mathrm{QA}^* = (0) \vee (2) \vee (10) \vee (12) = QC\ UM\ \vee\ \overline{QA}\,QB\ \vee\ \overline{QA}\,\overline{QC}\,\overline{UM}$$

$$DB = \mathrm{QB}^* = (1) \vee (2) \vee (11) \vee (12) = QC\ UM \vee QA\ QBUM \vee \overline{QA}\ QB\,\overline{UM} \vee QA\,\overline{QB}\,\overline{UM}$$

$$DC = \mathrm{QC}^* = (3) \vee (8) = QAQB\,\overline{UM} \vee\ \overline{QA}\,\overline{QB}\,\overline{QC}\ UM$$

Tabelle Ü17.3 Wahrheitstabelle für Aufgabe 17.3

UM	QC	QB	QA	Zahl	QC*	QB*	QA*
0	0	0	0	0	0	0	1
0	0	0	1	1	0	1	0
0	0	1	0	2	0	1	1
0	0	1	1	3	1	0	0
0	1	0	0	4	0	0	0
0	1	0	1	5	*	*	*
0	1	1	0	6	*	*	*
0	1	1	1	7	*	*	*
1	0	0	0	8	1	0	0
1	0	0	1	9	0	0	0
1	0	1	0	10	0	0	1
1	0	1	1	11	0	1	0
1	1	0	0	12	0	1	1
1	1	0	1	13	*	*	*
1	1	1	0	14	*	*	*
1	1	1	1	15	*	*	*

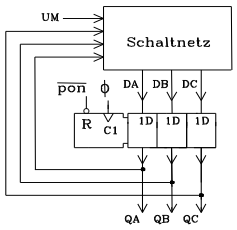


Abb. Ü17.2 Schaltung des umschaltbaren Modulo-5-Vorwärts-/Rückwärtszählers

VHDL-Modell: Umschaltbarer Modulo-5-Zähler

```
library ieee;
use ieee.std_logic_1164.all;
entity mod_5_um is port (
        clk, pon, um: in std_logic;
        q: out std_logic_vector (2 downto 0));
end mod_5_um;
architecture verhalt_mod_5 of mod_5_um is  -- Realisierung mit einer Zustandsmaschine
        type zustand_type is (S0, S1, S2, S3, S4);
```

```
        signal zustand: zustand_type;
begin
zustand_machine: process (clk, pon)                    --Registerausgabe
begin
        if pon = '0' then q <= "000"; zustand <= S0;    -- Anfangszustand
          elsif clk'event and clk = '1' then
            case zustand is                    -- Zustandsfolge ist von "um" abhaengig
                when S0 =>
                        if um='1' then zustand <= S4; q <= "100";
                        elsif um='0' then zustand <= S1; q <= "001";
                        end if;
                when S1 =>
                        if um='1' then zustand <= S0; q <= "000";
                        elsif um='0' then  zustand <= S2; q <= "010";
                        end if;
                when S2 =>
                        if um='1' then zustand <= S1; q <= "001";
                        elsif um='0' then zustand <= S3; q <= "011";
                        end if;
                when S3 =>
                        if um='1' then zustand <= S2; q <= "010";
                        elsif um='0' then zustand <= S4; q <= "100";
                        end if;
                when S4 =>
                        if um='1' then zustand <= S3; q <= "011";
                        elsif um='0' then zustand <= S0; q <= "100";
                        end if;
                when others => null;
            end case;
        end if;

end process zustand_machine;
end verhalt_mod_5;
```

Aufgabe 18: Entwurf eines synchronen Schaltwerks (Moore-Automat) [VHDL]*

Gegeben ist ein Takt ϕ mit der Periodendauer T_p = 1µs und ein zum Takt asynchrones Eingangssignal X1 (Abb. Ü18.1). Die Eingangsimpulse sind breiter als T_p und der Abstand zwischen den Eingangsimpulsen, gemessen von negativer Flanke bis zur nächstfolgenden positiven, ist größer als 3 T_p.

Entwerfen Sie einen digitalen Differenzierer, der sowohl nach der positiven als auch nach der negativen Flanke eines Eingangsimpulses je einen zur positiven Taktflanke synchronen Impuls der Breite T_p ausgibt. Auch als Reaktion auf schmale Eingangsimpulse sollen an Y1 zwei Impulse in einem Abstand der Taktperiode T_p ausgegeben werden. Die synchrone Ausgabe an Y1 soll so schnell wie möglich erfolgen (Abb. Ü18.1).

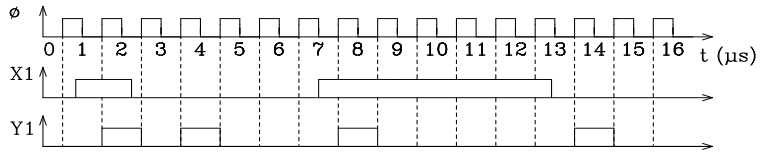


Abb. Ü18.1 Signalzeitplan des digitalen Differenzierers

Anleitung:

Setzen Sie zur Lösung der Aufgabe ein synchrones Schaltwerk vom Typ Moore-Automat ein. Stellen Sie das entsprechende Zustandsdiagramm und die Zustandsfolgetabelle auf und reduzieren Sie die Zustände soweit wie möglich. Entwerfen Sie anhand der Zustandsfolgetabelle ein synchrones Schaltwerk mit D-Flipflops als Zustandsvariablenspeicher.

Lösung:

Zur Lösung der Aufgabe wird zunächst ein Zustandsdiagramm entworfen. Anhand der Aufgabenstellung wird schrittweise das Zustandsdiagramm a) in Abb. Ü18.2 entwickelt. Nach dem Einschalten der Versorgungsspannung (pon) wird der Anfangszustand 0 erreicht. In diesem Zustand gibt das Schaltwerk am Ausgang Y1 den Logik-Zustand 0 aus und wartet (Warteschleife für ¬X1 = 1) bis das Eingangssignal X1 den Logik-Zustand 1 annimmt. Für X1 = 1 erfolgt mit der nächsten positiven Taktflanke der Übergang in den Zustand 1.

Im Zustand 1 wird an Y1 für eine Taktperiode der Logik-Zustand 1 ausgegeben. Mit der nächsten positiven Taktflanke erfolgt für ¬X1 = 1 (kurzer Eingangsimpuls) der Übergang nach Zustand 4, während für X1 = 1 (breiter Impuls) der Folgezustand 2 erreicht wird. In beiden Zuständen (2 und 4) wird Y1 = 0 (¬Y1 = 1) ausgegeben.

Von Zustand 4 ausgehend wird für X1 = 0 (¬X1 = 1) zunächst der Zustand 5 erreicht, in dem Y1 = 1 wird (zweiter Ausgabeimpuls) und anschließend der Anfangszustand 0. Sowohl im Zustand 4 als auch im Zustand 5 kann aufgrund der Randbedingung "Minimaler Abstand zwischen zwei Eingangsimpulsen ist größer als 3 T_p" das Eingangssignal X1 nicht 1 werden.

Falls ein breiter Eingangsimpuls vorliegt, wartet das Schaltwerk im Zustand 2 das Impulsende (Warteschleife für X1 = 1) ab und geht dann für X1 = 0 (¬X1 = 1) mit der nächsten positiven Flanke in den Zustand 3 über. Im Zustand 3 wird für eine Taktperiode Y1 = 1 ausgegeben, und anschießend erfolgt der Übergang in den Anfangszustand 0. Aufgrund des minimalen Abstands der Eingangsimpulse von größer 3 T_p kann im Zustand 3 die Eingangsvariable X1 nicht "1" werden.

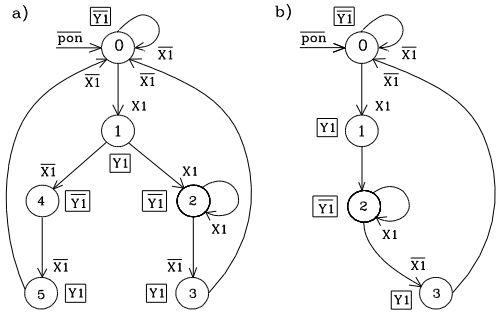


Abb. Ü18.2 Zustandsdiagramm des digitalen Differenzierers

Die zu dem Zustandsdiagramm in Abb. Ü18.2 a) zugehörige Zustandsfolgetabelle ist in Tabelle Ü18.1 a) dargestellt. Die Zustände 3 und 5 in Tabelle Ü18.1 a) sind äquivalent und werden zu dem Zustand 3 zusammengefasst. Wenn man die Zustandsnummer 5 durch 3 ersetzt, erkennt man, dass die Zustände 2 und 4 ebenfalls äquivalent sind. Sie werden zu dem Zustand 2 zusammengefasst. Die sich daraus ergebende reduzierte Zustandsfolgetabelle ist in Tabelle Ü18.1 b) abgebildet und das zugehörige Zustandsdiagramm ist in Abb. Ü18.2 b) skizziert.

Tabelle Ü18.1 Gegenüberstellung der nichtreduzierten (links) und der reduzierten Zustandsfolgetabelle (rechts)

Eingang	Zustand		Ausgang
m	m	m+1	m
X	Z	Z^*	Y
0	0	0	0
1	0	1	0
0	1	4	1
1	1	2	1
0	2	3	0
1	2	2	0
0	3	0	1
1	3	*	1
0	4	5	0
1	4	*	0
0	5	0	1
1	5	*	1

Eingang	Zustand		Ausgang
m	m	m+1	m
X	Z	Z^*	Y
0	0	0	0
1	0	1	0
0	1	2	1
1	1	2	1
0	2	3	0
1	2	2	0
0	3	0	1
1	3	*	1

Tabelle Ü18.2 Ausführliche Form der reduzierten Zustandsfolgetabelle

Eing. (dez.)	**Zustand (dez.)**		**Ausg. (dez.)**		**Eingangs-variablen**	**Zustands-variablen**				**Augangs-variablen**
m	m	m+1	m		m	m		m+1		m
X	Z	Z^*	Y		X1	Z1	Z2	$Z1^*$	$Z2^*$	Y1
0	0	0	0		0	0	0	0	0	0
1	0	1	0		1	0	0	0	1	0
0	1	2	1		0	0	1	1	0	1
1	1	2	1		1	0	1	1	0	1
0	2	3	0		0	1	0	1	1	0
1	2	2	0		1	1	0	1	0	0
0	3	0	1		0	1	1	0	0	1
1	3	*	1		1	1	1	*	*	1

Anhand der ausführlichen Zustandsfolgetabelle in Tabelle Ü18.2 werden mit Hilfe der KV-Diagramme (Abb. Ü18.3) die Gleichungen D1 und D2 für die D-Flipflops sowie die Ausgangsgleichung Y1 bestimmt.

Für die KV-Diagramme gilt folgende Zuordnung: X1: 2^2 Z1: 2^1 Z2: 2^0

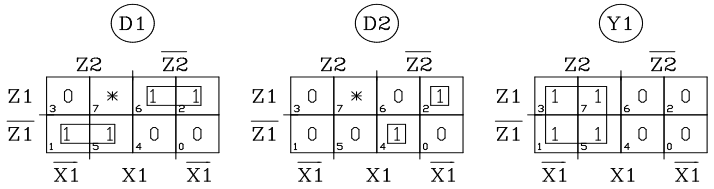


Abb. Ü18.3 KV-Diagramme für D1, D2 und Y1

$Z1^* = D1 = (1) \vee (5) \vee (2) \vee (6) = Z1\ \overline{Z2} \vee \overline{Z1}\ Z2$

$Z2^* = D2 = (4) \vee (2) = X1\ \overline{Z1}\ \overline{Z2} \vee \overline{X1}\ Z1\ \overline{Z2}$

$Y1 = (1) \vee (5) \vee (3) \vee (7) = Z2$

Das gesuchte Schaltwerk ist in Abb. Ü18.4 abgebildet. Für die technische Realisierung kann ein PAL mit Registerausgang eingesetzt werden. In diesem Fall wird zur Lösung der Aufgabe nur ein integrierter Baustein benötigt.

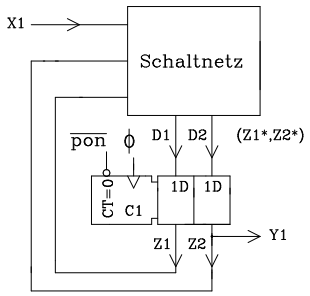


Abb. Ü18.4 Schaltung des digitalen Differenzierers

Das Schaltwerk in Abb. Ü18.4 ist ein Moore-Automat mit synchroner Ausgabe Y1 = Z2. Die im Zustandsdiagramm (Abb. Ü18.2) gekennzeichnete Möglichkeit, über einen Einschaltimpuls (¬pon) den Anfangszustand 0 zu erreichen, wird im Schaltwerk über den negierten Rücksetzeingang am D-Register mit ¬pon = 0 realisiert.

VHDL-Modell: Digitaler Differenzierer mit zwei Ausgabeimpulsen

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

entity differ_2 is port (
  clk, x1, pon:      in std_logic;
  y1:                out std_logic);
end differ_2;

architecture differ_2_arch of differ_2 is
  type zustand_type is (S0, S1, S2, S3);
  signal zustand: zustand_type;
begin
zustand_machine: process (clk, pon)
begin
  if pon='0' then zustand <= S0;        -- mit pon = 0 in den Anfangszustand
  elsif clk'event and clk = '1' then
        case zustand is
              when S0 =>                                  -- Zustand 0 (S0)
                    if x1='1' then zustand <= S1;
                    elsif x1='0' then zustand <= S0;
                    end if;
              when S1 =>                                  -- Zustand 1 (S1)
                    zustand <= S2;
              when S2 =>                                  -- Zustand 2 (S2)
                    if x1='1' then zustand <= S2;
                    elsif x1='0' then zustand <= S3;
                    end if;
              when S3 =>                                  -- Zustand 3 (S3)
                    if x1='0' then zustand <= S0;
```

```
                    end if;
        end case;
  end if;
end process;

y1_zuweisung:                    -- Signal-Zuweisung fuer kombinatorische Ausgabe
y1 <= '1' when (zustand = S1 or zustand = S3)
        else '0';
end differ_2_arch;
```

Aufgabe 19: Entwurf eines synchronen Schaltwerks (Mealy-Automat) [VHDL]*

Gegeben ist ein Takt ϕ mit der Periodendauer T_p = 1µs und ein zum Takt asynchrones Eingangssignal X1 (Abb. Ü19.1). Die Eingangsimpulse sind breiter als T_p und der Abstand zwischen den Eingangsimpulsen, gemessen von negativer Flanke bis zur nächstfolgenden positiven, ist größer als 3 T_p.

Entwerfen Sie einen digitalen Differenzierer, der unmittelbar nach der positiven

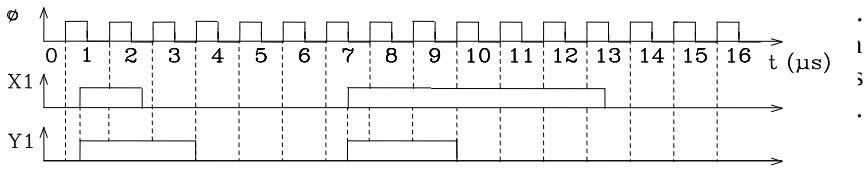


Abb. Ü19.1 Signalzeitplan des digitalen Differenzierers

Anleitung:

Setzen Sie zur Lösung der Aufgabe ein synchrones Schaltwerk vom Typ Mealy-Automat ein. Stellen Sie das entsprechende Zustandsdiagramm und die Zustandsfolgetabelle auf und reduzieren Sie die Zustände soweit wie möglich. Entwerfen Sie anhand der Zustandsfolgetabelle ein synchrones Schaltwerk mit D-Flipflops als Zustandsvariablenspeicher.

Lösung:

Zur Lösung der Aufgabe wird zunächst ein Zustandsdiagramm entworfen. Anhand der Aufgabenstellung wird schrittweise das Zustandsdiagramm a) in Abb. Ü19.2 entwickelt. Nach dem Einschalten der Versorgungsspannung (pon) wird der Anfangszustand 0 erreicht. In diesem Zustand gibt das Schaltwerk am Ausgang Y1 für X1 = 0 den Logik-Zustand 0 und für X1 = 1 den Logik-Zustand 1 aus. So-

lange X1 = 0 ist, bleibt das Schaltwerk in einer Warteschleife. Für X1 = 1 erfolgt mit der nächsten positiven Taktflanke der Übergang in den Zustand 1.

Im Zustand 1 wird an Y1 für eine Taktperiode der Logik-Zustand 1 ausgegeben. Mit der nächsten positiven Taktflanke erfolgt für ¬X1 = 1 (kurzer Eingangsimpuls) der Übergang nach Zustand 4, während für X1 = 1 (breiter Impuls) der Folgezustand 2 erreicht wird. In beiden Zuständen (2 und 4) wird Y1 = 1 ausgegeben. Da im Zustand 0 für X1 = 1 und in zwei aufeinander folgenden Zuständen (1 und 2 bzw. 1 und 4) am Ausgang Y1 = 1 ausgegeben wird, sind die Ausgangsimpulse breiter als 2 T_p und schmaler als 3 T_p .

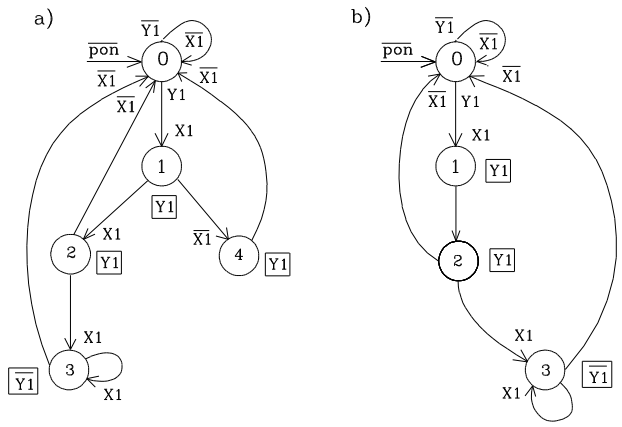


Abb. Ü19.2 Zustandsdiagramm des digitalen Differenzierers

Vom Zustand 4 ausgehend wird für X1 = 0 (¬X1 = 1) der Anfangszustand 0 erreicht. Im Zustand 4 kann aufgrund der Randbedingung „Minimaler Abstand zwischen zwei Eingangsimpulsen ist größer als 3 T_p“ das Eingangssignal X1 nicht 1 werden. Im Zustand 2 verzweigt sich der Signalfluss. Für X1 = 0 (¬X1 = 1) wird der Anfangszustand und für X1 = 1 der Zustand 3 erreicht.

Im Zustand 3 wartet das Schaltwerk das Impulsende (Warteschleife für X1 = 1) ab und geht dann für X1 = 0 (¬X1 = 1) mit der nächsten positiven Flanke in den Zustand 0 über. Im Zustand 3 wird Y1 = 0 (¬Y1 = 1) ausgegeben.

Die zu dem Zustandsdiagramm in Abb. Ü19.2 a) zugehörige Zustandsfolgetabelle ist in Tabelle Ü19.1 a) dargestellt.

Die Zustände 2 und 4 in Tabelle Ü19.1 a) sind äquivalent und werden zu dem Zustand 2 zusammengefasst. Die sich daraus ergebende reduzierte Zustandsfolgetabelle ist in Tabelle Ü19.1 b) abgebildet und das zugehörige Zustandsdiagramm ist in Abb. Ü19.2 b) skizziert.

Tabelle Ü19.1 Gegenüberstellung der nichtreduzierten und der reduzierten Zustandsfolgetabelle

a) Zustandsfolgetabelle

Eingang	**Zustand**		**Ausgang**
m	m	m+1	m
X	Z	Z*	Y
0	0	0	0
1	0	1	1
0	1	4	1
1	1	2	1
0	2	0	1
1	2	3	1
0	3	0	0
1	3	3	0
0	4	0	1
1	4	*	1

b) Reduzierte Zustandsfolgetabelle

Eingang	**Zustand**		**Ausgang**
m	m	m+1	m
X	Z	Z*	Y
0	0	0	0
1	0	1	1
0	1	2	1
1	1	2	1
0	2	0	1
1	2	3	1
0	3	0	0
1	3	3	0

Tabelle Ü19.2 Ausführliche Form der reduzierten Zustandsfolgetabelle

Eing. (dez.)	**Zustand (dez.)**		**Ausg. (dez.)**		**Eigangs-variablen**	**Zustands-variablen**				**Augangs-variablen**
m	m	m+1	m		m	m		m+1		m
X	Z	Z^*	Y		X1	Z1	Z2	$Z1^*$	$Z2^*$	Y1
0	0	0	0		0	0	0	0	0	0
1	0	1	1		1	0	0	0	1	1
0	1	2	1		0	0	1	1	0	1
1	1	2	1		1	0	1	1	0	1
0	2	0	1		0	1	0	0	0	1
1	2	3	1		1	1	0	1	1	1
0	3	0	0		0	1	1	0	0	0
1	3	3	0		1	1	1	1	1	0

Anhand der ausführlichen Zustandsfolgetabelle in Tabelle Ü19.2 werden mit Hilfe der KV-Diagramme (Abb. Ü19.3) die Gleichungen D1 und D2 für die D-Flipflops sowie die Ausgangsgleichung Y1 bestimmt.

Für die KV-Diagramme gilt folgende Zuordnung: 2^2 2^1 2^0
X1 Z1 Z2

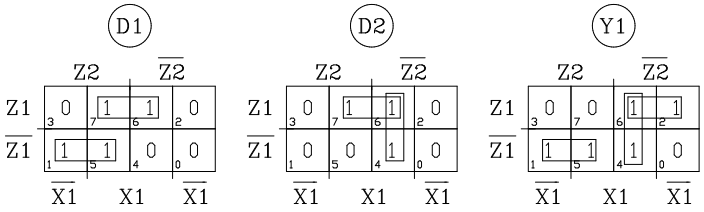


Abb. Ü19.3 KV-Diagramme für D1, D2 und Y1

Minimale Gleichungen für Z1*, Z2* und Y1:

$Z1^* = D1 = (1) \vee (5) \vee (6) \vee (7) = \overline{Z1}\ Z2 \vee X1\ Z1$

$Z2^* = D2 = (4) \vee (6) \vee (7) = X1\ \overline{Z2} \vee X1\ Z1$

$Y1 = (4) \vee (1) \vee (5) \vee (2) \vee (6) = Z1\ \overline{Z2} \vee \overline{Z1}\ Z2 \vee X1\ \overline{Z2}$

Das gesuchte Schaltwerk ist in Abb. Ü19.4 abgebildet. Für die technische Realisierung kann ein PAL mit zusätzlichen Registerausgängen eingesetzt werden. In diesem Fall wird zur Lösung der Aufgabe nur ein integrierter Baustein benötigt.

Das Schaltwerk in Abb. Ü19.4 ist ein Mealy-Automat mit asynchroner Ausgabe am Ausgang Y1. Die im Zustandsdiagramm (Abb. Ü19.2) markierte Möglichkeit, über einen Einschaltimpuls (¬pon) den Anfangszustand 0 zu erreichen, wird im Schaltwerk über den negierten Rücksetzeingang am D-Register mit ¬pon = 0 realisiert.

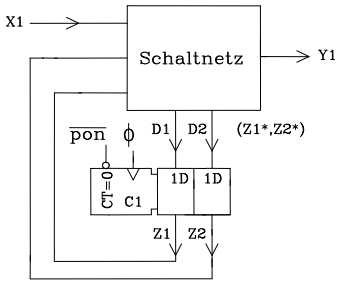


Abb. Ü19.4 Schaltung des digitalen Differenzierers

VHDL-Modell: Digitaler Differenzierer als Mealy-Automat

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

entity diff_mealy is port (
  clk, pon, x1:      in std_logic;
  y1:                out std_logic);
end diff_mealy;
```

```
architecture diff_mealy_arch of diff_mealy is
  type zustand_type is (S0, S1, S2, S3);
  signal zustand: zustand_type;
begin
zustand_machine: process (clk, pon)
begin
  if pon='0' then zustand <= S0;          -- pon = 0 --> Anfangszustand
  elsif clk'event and clk = '1' then
          case zustand is
                  when S0 =>
                          if x1='1' then zustand <= S1;
                          elsif x1='0' then zustand <= S0;
                          end if;
                  when S1 =>
                          zustand <= S2;
                  when S2 =>
                          if x1='0' then zustand <= S0;
                          elsif x1='1' then zustand <= S3;
                          end if;
                  when S3 =>
                          if x1='1' then zustand <= S3;
                          elsif x1='0' then zustand <= S0;
                          end if;
          end case;
  end if;
end process;

y1_assignment:                  -- Kombinatorische Ausgabe fuer Mealy-Automat
y1 <= '1' when (zustand = S0 and x1='1') else
     '0' when (zustand = S0 and x1='0') else
     '1' when (zustand = S1) else
     '1' when (zustand = S2)
     else '0';

end diff_mealy_arch;
```

Aufgabe 20: Entwurf eines synchronen Schaltwerks mit Registerausgabe [VHDL]*

Für die Steuerung der Datenübergabe von einem Rechner an zwei Messgeräte soll ein synchrones Schaltwerk entworfen werden. Es werden nacheinander Datenwörter gesendet, die von den Messgeräten empfangen und quittiert werden. Ein neues Datenwort wird mit der positiven Flanke am Ausgang Y1 des Schaltwerks gesendet und bleibt bis zur negativen Flanke von Y1 gültig. Der Ausgang Y1 soll solange im 1-Zustand bleiben, bis beide Messgeräte das Datenwort empfangen und und durch einen 1-Impuls quittiert haben. Nach einer kurzen Pause wird anschließend das nächste Datenwort mit einem neuen 1-Impuls am Ausgang Y1 übergeben (s. Abb. Ü20.1).

Es gilt:

a) Die Übernahme der Daten am Messgerät ist mit der negativen Flanke des Quittungssignals X1 bzw. X2 abgeschlossen.
b) Impulsbreite von X1 bzw. X2 ist größer als 1,2 µs und kleiner als 10 µs.
c) Impulspause am Ausgang Y1 ist größer als 0,5µs.

Zur Lösung der Aufgabenstellung soll ein synchroner Moore-Automat mit Registerausgabe eingesetzt werden. Bestimmen Sie die erforderliche Taktfrequenz und begründen Sie Ihre Wahl. Geben Sie das Zustandsdiagramm und die Zustandsfolgetabelle in ausführlicher Form an. Reduzieren Sie die Anzahl der Zustände, falls es möglich ist. Die digitale Schaltung ist *nicht* erforderlich. Geben Sie ein geeignetes VHDL-Modell an, das eine Registerausgabe ermöglicht.

Anm.: Es ist sichergestellt, dass stets beide Messgeräte Quittungssignale senden.

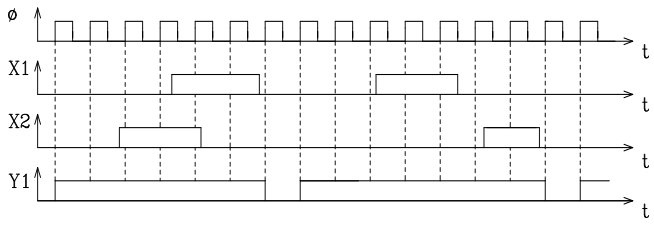


Abb. Ü20.1 Beispiel für den Signalverlauf am Ein- und am Ausgang des Schaltwerks

Lösung:

Es soll gelten: $0{,}5\mu s \le T_\Phi \le 1.2\mu s$, damit ein Impuls einerseits sicher erfasst wird und andererseits die Impulspause an Y groß genug ist. Gewählt wird $T_\Phi = 1\mu s$. Y1* an ein Register (z.B. D-Flipflop) angeschlossen und der entsprechende Wert wird mit der nächsten aktiven Taktflanke als Y1 am Registerausgang ausgegeben.

Tabelle Ü20.1 Zustandsfolgetabelle für einen Moore-Atomaten mit Registerausgabe

X1	**X2**	**Z1**	**Z2**	**Z3**	**Z1***	**Z2***	**Z3***	**Y1***
0	0	0	0	0	0	0	1	1
0	1	0	0	0	*	*	*	1
1	0	0	0	0	*	*	*	1
1	1	0	0	0	*	*	*	1
0	0	0	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
1	0	0	0	1	0	1	0	1
1	1	0	0	1	0	1	1	1
0	0	0	1	0	0	1	0	1

0	1	0	1	0	0	1	1	1
1	0	0	1	0	0	1	0	1
1	1	0	1	0	0	1	1	1
0	0	0	1	1	0	0	0	0
0	1	0	1	1	0	1	1	1
1	0	0	1	1	0	1	1	1
1	1	0	1	1	0	1	1	1
0	0	1	0	0	1	0	0	1
0	1	1	0	0	1	0	0	1
1	0	1	0	0	0	1	1	1
1	1	1	0	0	0	1	1	1

VHDL-Modell: Moore-Automat mit Registerausgabe

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

entity daten_ueber is port (
        pon,clk:   in std_logic;
        x:                 in std_logic_vector (1 to 2);
        y1:                out std_logic);
end daten_ueber;

architecture arch_ueber of daten_ueber is
        type states is (S0,S1,S2,S3,S4); -- Typdeklaration
        signal zustand: states;
begin
reg_aus: process(clk, pon)
  begin
        if pon='1'then                          -- pon = 1 ---> Anfangszustand
          zustand <= S0;
          y1 <= '0';
        elsif (clk'event and clk = '1') then
                y1 <= '1';              -- y1 = 1 ist die Voreinstellung
                case zustand is                 -- wenn IST-Zustand gleich ...
                   when S0 =>
                        zustand <= S1;
                   when S1 =>
                     case x is                  -- Folgezustand ist abhaengig von x
                        when "00" => zustand <= S1;
                        when "01" => zustand <= S4;
                        when "10" => zustand <= S2;
                        when others => zustand <= S3;
                     end case;
                   when S2 =>
                      if (x = "00" or x = "10") then  -- X=0 oder X=2
                          zustand <= S2;
                      else                              -- X=1 oder X=3
                          zustand <= S3;
                      end if;
                   when S3 =>
                       if x="00" then
                          zustand <= S0;
                          y1 <= '0';            -- Y1=0 gilt fuer den Zustand S0
```

```
                    else
                        zustand <= S3;
                    end if;
                when S4 =>
                    if (x ="00" or x = "01")  then
                        zustand <= S4;
                    else
                        zustand <= S3;
                    end if;
            end case;
        end if;
end process reg_aus;
end arch_ueber;
```

Aufgabe 21: Entwurf eines SRAMs 1Ki x 8 Bit (VHDL-Modell mit Testbench) [VHDL]*

Entwerfen Sie ein VHDL-Modell mit Testbench für ein SRAM der Speicherkapazität 1Ki x 8 Bit. Alle Ein- und Ausgänge sollen vom Datentyp „std_logic“ bzw. „std_logic_vector“ sein. Das Modell soll erprobt werden mit einer Testbench, die mit Ein- und Ausgabedateien arbeitet.

Anleitung:

Das Modell für den Speicher soll parametrisierbar sein, so dass eine Anpassung an andere Speicherkapazitäten leicht möglich ist. In der Eingabedatei sollen die unterschiedlichen Betriebszustände des Speichers (Schreiben, Lesen, Datenbus hoch-ohmig) berücksichtigt werden. Der Zugriff auf den Speicher wird gesteuert über

- nwe = write enable, low aktiv
- noe = output enable, low aktiv
- ncs = chip select, low aktiv

Lösung:

Das VHDL-Modell wird als Komponente in dem Package „ram_pack“ abgelegt und steht nach der Compilierung in der Library work zur Verfügung.

VHDL-Modelleines SRAMs mit Testbench:

```
-- ram_pack.vhd

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;

package ram_pack is

component ram
    generic (n: integer := 10;          -- n = Anzahl der Adressbits
     k: integer := 8);                  -- k = Anzahl der Datenbits
```

```
port (      nwe, noe, ncs: in std_logic; -- Steuersignale
     adresse: in std_logic_vector (n - 1 downto 0);          -- Adresse
     daten: inout std_logic_vector(k - 1 downto 0));         -- Daten
end component;

end ram_pack;
```

```
-- RAM mit nKi x iBit

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;

entity ram is
generic (n: integer := 10;
      k: integer := 8);
port (
     nwe, noe, ncs:  in std_logic;
     adresse:        in std_logic_vector (n - 1 downto 0);
     daten:          inout std_logic_vector(k - 1 downto 0));
end ram;
```

```
architecture arch_ram of ram is
--Std_ToInt  32 Bits maximal
-- Typkonvertierung: std_logic_vector → integer

function Std_ToInt (std: std_logic_vector) return integer is
     variable result, abit: integer := 0;
     variable count: integer := 0;
begin
     bits: for i in std'low to std'high loop
     abit := 0;
          if (std(i) = '1') then              -- alle Werte ausser '1'  → abit := 0
                    abit := 2**(i-std'low);
          end if;
                    result := result + abit;
          count := count + 1;
          exit bits when count = 32;     -- 32 Bits maximal
     end loop bits;
return (result);
end Std_ToInt;

     type ram_matrix is array (natural range 0 to 2**n - 1) of std_logic_vector(k-1 downto
     0);
     signal mem: ram_matrix := (others => (others => '0')); -- alle Speicherplaetze auf 0 set-
     zen
     signal add:       natural;

begin
zugriff: process (ncs,adresse)
     constant tzu: time := 50 ns;
```

```
begin
     add <= Std_ToInt(adresse);
     if ncs = '1' then
              daten <= "ZZZZZZZZ" after tzu;
  elsif ncs='0' and nwe = '0' then
            mem(add) <= daten after tzu; -- Daten speichern
  elsif ncs='0' and nwe = '1'and noe = '0' then
            daten <= mem(add) after tzu; -- Daten lesen
  end if;
end process zugriff;
end arch_ram;

-- tb_ram_io.vhd
-- Testbench fuer RAM
library ieee;
use ieee.std_logic_1164.all;

-- use ieee.std_logic_unsigned.all;
use std.textio.all;
use work.text_io_pack.all;
use work.ram_pack.all;

entity tb_ram is
 generic (n: integer := 10;
       k: integer := 8);
end tb_ram;

architecture auto_ram of tb_ram is
-- variable n: natural;
-- variable k: natural;
signal nwe,noe,ncs: std_logic;
signal adresse: std_logic_vector(n-1 downto 0);
signal daten: std_logic_vector(k-1 downto 0);

begin
dut: ram  -- design under test
     generic map (10,8)
     port map (nwe,noe,ncs,adresse,daten);

testen: process
     file eingabe_datei: text is in "ram_ein.txt";    -- Eingabedatei
     file ausgabe_datei: text is out "ram_aus.txt"; -- Ausgabedatei
     variable zeile_ein, zeile_aus: line;
     variable v_nwe,v_noe,v_ncs: std_logic;
     variable v_adresse: std_logic_vector(n-1 downto 0);
     variable v_daten, aus_daten, soll_daten: std_logic_vector(k-1 downto 0);
     variable fehler: boolean := false;

     variable gut: boolean;
     variable char: character;
     variable fehler_aus: string(1 to 4) := "nein";
     constant abstand_2: string(1 to 2) := "  ";
     constant abstand_3: string(1 to 3) := "   ";
     constant abstand_4: string(1 to 4) := "    ";
```

```
-- Ueberschrift der Ausgabedatei
    constant ueber: string(1 to 53) := "nwe noe ncs   adresse   daten   daten_soll  Fehler";
begin
          write(zeile_aus, ueber);        -- Ueberschriftausgabe
          writeline(ausgabe_datei, zeile_aus); -- Ausgabe der Zeile
          writeline(ausgabe_datei, zeile_aus); -- Ausgabe einer Leerzeile

zeile_loop:          while not endfile(eingabe_datei) loop
    readline (eingabe_datei,zeile_ein); -- Zeile einlesen
    -- Zeile auswerten: Uebergabe an Signale
    -- ueberspringe Zeile, falls Zeichen kein Tabulator
    read (zeile_ein,char,gut);
    if not gut or char /= HT then next;
    end if;
    assert gut
    report "Fehler beim Lesen"
    severity note;
    read (zeile_ein,v_nwe,gut);          -- write enable
    next when not gut;
    read (zeile_ein,char);
    read (zeile_ein,v_noe,gut);          -- output enable
    next when not gut;
    read (zeile_ein,char);
    read (zeile_ein,v_ncs,gut);                  -- chip select
    next when not gut;
    read (zeile_ein,char);
    read (zeile_ein,v_adresse,gut);              -- Adresse
    next when not gut;
    read (zeile_ein,char);
    read (zeile_ein,v_daten,gut);                        -- Daten
    next when not gut;
    read (zeile_ein,char);
    read (zeile_ein,soll_daten,gut);                     -- Vergleichsdaten
    next when not gut;
    nwe <= v_nwe;-- Typ-Konvertierung: variable --> signal
    noe <= v_noe;
    ncs <= v_ncs;
    adresse <= v_adresse;
    daten <= v_daten;
    wait for 70 ns;

-- ueberpruefen des Ergebnisses
    aus_daten := daten;
    if (ncs='1' and aus_daten /= "ZZZZZZZZ") or
          (ncs='0' and nwe='1' and noe='0' and aus_daten /= soll_daten) then
          assert false
          report "RAM reagiert falsch";
          fehler := true;
          fehler_aus := " ja ";
    end if;
-- formatierte Ausgabe
```

```
    write(zeile_aus, ' ');
    write(zeile_aus, v_nwe);
    write(zeile_aus, abstand_3);
    write(zeile_aus, v_noe);
    write(zeile_aus, abstand_3);
    write(zeile_aus, v_ncs);
    write(zeile_aus, abstand_3);
    write(zeile_aus, v_adresse);
    write(zeile_aus, abstand_2);
    write(zeile_aus, aus_daten);
    write(zeile_aus, abstand_3);
    write(zeile_aus, soll_daten);
    write(zeile_aus, abstand_3);
    write(zeile_aus, fehler_aus);
    writeline(ausgabe_datei, zeile_aus);
          fehler_aus := "nein";
    wait for 30 ns;  -- Taktperiode = 100ns
end loop zeile_loop;

-- Ausgabe eines Reports
    assert not fehler
    report "Gesamtbewertung: RAM ist fehlerhaft"
    severity note;
    assert fehler
    report "Gesamtbewertung: RAM ist o.K"
    severity note;
    wait;
end process testen;
end auto_ram;
```

Report des Entwicklungstools ModeSim:

```
run -all
# ** Note: Gesamtbewertung: RAM ist o.K
#    Time: 1200 ns  Iteration: 0  Instance: /tb_ram
```

Eingabedatei

```
Eingabedatei RAM: 1 Ki x 8 Bit
Schreiben: Adresse 15: 10101010 und 1023: 00000011
   1 1 1 0000001111 ZZZZZZZZ ZZZZZZZZ
   0 1 0 0000001111 10101010 10101010
   1 1 1 1111111111 ZZZZZZZZ ZZZZZZZZ
   0 1 0 1111111111 00000011 00000011

Lesen: Adresse 15 und 1023
   1 1 1 0000001111 ZZZZZZZZ ZZZZZZZZ
   1 0 0 0000001111 ZZZZZZZZ 10101010
   1 1 1 1111111111 ZZZZZZZZ ZZZZZZZZ
   1 0 0 1111111111 ZZZZZZZZ 00000011

Lesen: Adresse 0 und 1 Default: 0
   1 1 1 0000000000 ZZZZZZZZ ZZZZZZZZ
   1 0 0 0000000000 ZZZZZZZZ 00000000
   1 1 1 0000000001 ZZZZZZZZ ZZZZZZZZ
   1 0 0 0000000001 ZZZZZZZZ 00000000
```

Ausgabedatei

```
nwe noe ncs    adresse    daten    daten_soll  Fehler

 1   1   1   0000001111  ZZZZZZZZ   ZZZZZZZZ   nein
 0   1   0   0000001111  10101010   10101010   nein
 1   1   1   1111111111  ZZZZZZZZ   ZZZZZZZZ   nein
 0   1   0   1111111111  00000011   00000011   nein
 1   1   1   0000001111  ZZZZZZZZ   ZZZZZZZZ   nein
 1   0   0   0000001111  10101010   10101010   nein
 1   1   1   1111111111  ZZZZZZZZ   ZZZZZZZZ   nein
 1   0   0   1111111111  00000011   00000011   nein
 1   1   1   0000000000  ZZZZZZZZ   ZZZZZZZZ   nein
 1   0   0   0000000000  00000000   00000000   nein
 1   1   1   0000000001  ZZZZZZZZ   ZZZZZZZZ   nein
 1   0   0   0000000001  00000000   00000000   nein
```

Aufgabe 22: Entwurf eines Speichersystems mit 8-Bit-Wortbreite

Entwerfen Sie ein digitales Speichersystem mit folgenden Eigenschaften:

- 16 Adressleitungen und 8 Datenleitungen (Wortbreite: 8 Bit)
- Adressbereich des RAMs: 0 ... 3FFFH (hexadezimal)
- Adressbereich des ROMs: 8000H ... BFFFH (hexadezimal)

Zur Verfügung stehen statische RAMs der Speicherkapazität 4Ki x 8 Bit und EPROMs der Speicherkapazität 8Ki x 8 Bit. Beide Speichertypen haben je einen ¬CS- und einen ¬OE-Anschluß. Die RAMs haben zusätzlich noch einen Schreibeingang ¬WE.

22.1 Geben Sie ein Blockschaltbild des gesamten Speichersystems an.

22.2 Geben Sie die Gleichungen für vollständige und unvollständige Decodierung an. Nennen Sie Vor- und Nachteile der beiden Decodierungsarten.

22.3 Im RAM-Bereich soll nun wahlweise der Speicher im Adressbereich von 3800H ... 3FFFH per Schalter gesperrt werden können. Geben Sie für den Fall der vollständigen Decodierung die Änderung der Gleichungen und die entsprechende Schaltung an.

Lösung zu 22.1:

Die Adressen eines Speicherbereichs werden häufig als Hexadezimalzahlen angegeben, während die Speicherkapazität der Schreib-/Lese- und Festwertspeicher mit Potenzzahlen zur Basis 2 bezeichnet wird. Aus dem Grund werden in einer Tabelle die für die Aufgabe wichtigen Umrechnungen zwischen Potenzzahlen zur Basis 2, Hexadezimalzahlen und Dezimalzahlen angegeben.

Tabelle Ü22.1 Umrechnungen zwischen Potenzzahlen zur Basis 2, Hexadezimalzahlen und Dezimalzahlen

Potenzzahl zur Basis 2	Abkürzung	Hexadezimal-zahl	Dezimalzahl
2^{10}	1Ki	400H	1024
2^{11}	2Ki	800H	2048
2^{12}	4Ki	1000H	4096
2^{13}	8Ki	2000H	8192
2^{14}	16Ki	4000H	16384
2^{15}	32Ki	8000H	32768

Für die erforderliche RAM-Speicherkapazität von 4000H x 8 Bit werden nach Tabelle Ü22.1 vier statische RAMs der Kapazität 4Ki x 8 Bit benötigt. Der geforderte Festwertspeicher von ebenfalls 4000H x 8 Bit lässt sich mit zwei EPROMs des zur Verfügung stehenden Typs (Speicherkapazität = 8Ki x 8 Bit) realisieren. In Abb. Ü22.1 ist das geforderte Blockschaltbild dargestellt.

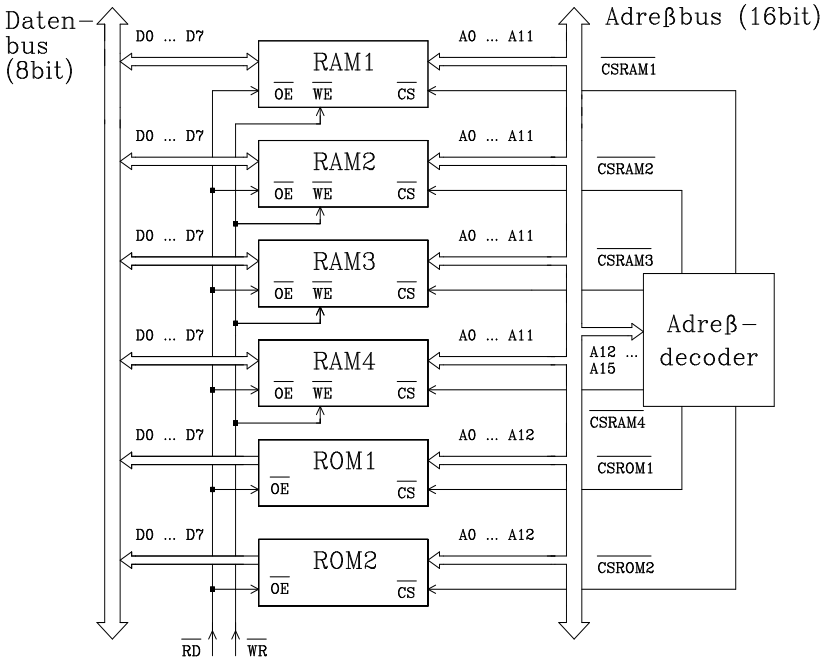


Abb. Ü22.1 Blockschaltbild des gesuchten Speichersystems

Lösung zu 22.2:

In einer Tabelle (Tabelle Ü22.2) ist die Aufteilung des gesamten Adressbereichs dargestellt. Für spätere Erweiterungen sind noch zwei Bereiche reserviert. Mit Hilfe der angegebenen Anfangs- und Endadressen der einzelnen Speicherbausteine lassen sich die Gleichungen für vollständige und unvollständige Adressdecodierung aufstellen.

Bei der vollständigen Adressdecodierung werden alle im System zur Verfügung stehenden Adressbits, die nicht schon am Speicherbaustein angeschlossen sind, zur Decodierung mit herangezogen. Für die unvollständige Adressdecodierung werden nur die zur Unterscheidung der einzelnen Speicherbausteine unbedingt notwendigen Adressbits zur Decodierung verwendet.

Der Hardwareaufwand ist bei der vollständigen Adressdecodierung größer als bei der unvollständigen. Vorteilhaft ist bei der vollständigen Adressdecodierung die eindeutige Beziehung zwischen Adresse und Speicherplatz, während bei der unvollständigen ein Speicherplatz unter mehreren Adressen angesprochen werden kann. Eine Speichererweiterung ist bei vollständiger Adressdecodierung leicht möglich, während bei unvollständiger alle Gleichungen überprüft und ggf. korrigiert werden müssen.

Tabelle Ü22.2 Speicherbelegungsplan für die gestellte Aufgabe

Speicher-Bereich	**Adresse (hex.)**	**Adresse (binär)**							**Speichertyp**	**¬Chip Select-Signal**
		A15	A14	A13	A12	A11	....	A0		
Anfang	0000	0	0	0	0	0	....	0	RAM1	¬CSRAM1
Ende	0FFF	0	0	0	0	1	....	1		
Anfang	1000	0	0	0	1	0	....	0	RAM2	¬CSRAM2
Ende	1FFF	0	0	0	1	1	....	1		
Anfang	2000	0	0	1	0	0	....	0	RAM3	¬CSRAM3
Ende	2FFF	0	0	1	0	1	....	1		
Anfang	3000	0	0	1	1	0	....	0	RAM4	¬CSRAM4
Ende	3FFF	0	0	1	1	1	....	1		
Anfang	4000	0	1	0	0	0	....	0	nichtbelegt	
Ende	7FFF	0	1	1	1	1	....	1		
Anfang	8000	1	0	0	0	0	....	0	ROM1	¬CSROM1
Ende	9FFF	1	0	0	1	1	....	1		
Anfang	A000	1	0	1	0	0	....	0	ROM2	¬CSROM2
Ende	BFFF	1	0	1	1	1	....	1		
Anfang	C000	1	1	0	0	0	....	0	nichtbelegt	
Ende	FFFF	1	1	1	1	1	....	1		

Tabelle Ü22.3: Gleichungen für die vollständige und unvollständige Adressdecodierung

Vollständige Adressdecodierung	*Unvollständige Adressdecodierung*
$\overline{CSRAM1} = \overline{\overline{A15}\,\overline{A14}\,\overline{A13}\,\overline{A12}}$	$\overline{CSRAM1} = \overline{\overline{A15}\,\overline{A13}\,\overline{A12}}$
$\overline{CSRAM2} = \overline{\overline{A15}\,\overline{A14}\,\overline{A13}\,A12}$	$\overline{CSRAM2} = \overline{\overline{A15}\,\overline{A13}\,A12}$
$\overline{CSRAM3} = \overline{\overline{A15}\,\overline{A14}\,A13\,\overline{A12}}$	$\overline{CSRAM3} = \overline{\overline{A15}\,A13\,\overline{A12}}$
$\overline{CSRAM4} = \overline{\overline{A15}\,\overline{A14}\,A13\,A12}$	$\overline{CSRAM4} = \overline{\overline{A15}\,A13\,A12}$
$\overline{CSROM1} = \overline{A15\,\overline{A14}\,\overline{A13}}$	$\overline{CSROM1} = \overline{A15\,\overline{A13}}$
$\overline{CSROM2} = \overline{A15\,\overline{A14}\,A13}$	$\overline{CSROM2} = \overline{A15\,A13}$

Lösung zu 22.3: Der zu sperrende Adressbereich liegt vollständig im Bereich des Bausteins RAM4. Folglich muß auch nur die Gleichung ¬CSRAM4 geändert werden. In die Adressdecodierung wird nun zusätzlich das Adressbit A11 und der vom Schalter einstellbare Logik-Zustand S einbezogen.

Es gilt folgende Zuordnung (Abb. Ü22.2):

Schalter geöffnet (S = 1): Speicher RAM4 kann im Adressbereich von 3000H bis 3FFFH angesprochen werden.

Schalter geschlossen (S = 0): Speicher RAM4 kann nur im Adressbereich von 3000H bis 37FFH angesprochen werden.

$$\overline{CSRAM4^*} = \overline{A15}\;\overline{A14}\,A13\,A12\,A11\;S \vee \overline{A15}\;\overline{A14}\,A13\,A12\,\overline{A11}\;S \vee \overline{A15}\;\overline{A14}\,A13\,A12\,\overline{A11}\;\overline{S}$$

$$\overline{CSRAM4^*} = \overline{A15}\;\overline{A14}\,A13\,A12\,S\,(A11 \vee \overline{A11}) \vee \overline{A15}\;\overline{A14}\,A13\;A12\;\overline{A11}\;(S \vee \overline{S})$$

$$\overline{CSRAM4^*} = \overline{A15}\;\overline{A14}\,A13\;A12\;S \vee \overline{A15}\;\overline{A14}\,A13\,A13\;\overline{A11}$$

Negiert man beide Seiten, so erhält man:

$$CSRAM4^* = \overline{\overline{A15}\;\overline{A14}\,A13\,A12\,S \vee \overline{A15}\;\overline{A14}\,A13\,A12\,\overline{A11}}$$

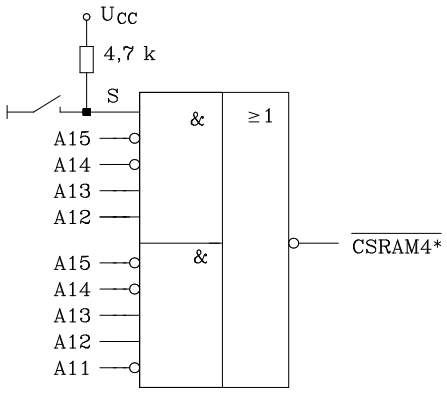


Abb. Ü22.2 Mit Hilfe eines Schalters lässt sich der Speicher RAM4 im Adressbereich 3800H bis 3FFFH sperren.

Aufgabe 23: Speichersystem mit 16-Bit-Datenbus

Entwerfen Sie ein Speichersystem mit den angegebenen Speicherkapazitäten.

- Gesamter RAM-Bereich: 128Ki * 16 Bit
- Gesamter Festwertspeicher-Bereich: 256Ki * 16 Bit

Es gelten folgende Randbedingungen:

- 20-Bit-Adressbus und 16-Bit-Datenbus
- Anfangsadresse RAM-Bereich: 00000H
- Anfangsadresse ROM-Bereich: 80000H

Zur Verfügung stehen folgende Speicherbausteine:

Statische RAMs der Kapazität 32Ki * 8 Bit und EPROMs mit 128Ki * 8 Bit

Gesucht sind die Gleichungen für die vollständige und unvollständige Adressdecodierung.

Zusatzfrage:

Wie lauten die Gleichungen für die vollständige Adressdekodierung, falls die Anfangsadresse für den ROM-Bereich 88000H ist?

Lösung:

Es werden 8 SRAMs der Kapazität 32Ki· 8 Bit und 4 EPROMs mit einer Kapazität 128Ki· 8 Bit gewählt.

Tabelle 23.1 Speicherbelegungsplan

Adresse (hex.)	**Adresse (binär)**								**Chip Select**
	A19	A18	A17	A16	A15	A14…A0	D15…D8	D7…D0	
00000 07FFF	0 0	0 0	0 0	0 0	0 0	0.....0 1.....1	RAM 2	RAM 1	$\overline{\text{CSRAM1-2}}$
08000 0FFFF	0 0	0 0	0 0	0 0	1 1	0.....0 1.....1	RAM 4	RAM 3	$\overline{\text{CSRAM3-4}}$
10000 17FFF	0 0	0 0	0 0	1 1	0 0	0.....0 1.....1	RAM 6	RAM 5	$\overline{\text{CSRAM5-6}}$
18000 1FFFF	0 0	0 0	0 0	1 1	1 1	0.....0 1.....1	RAM 8	RAM 7	$\overline{\text{CSRAM7-8}}$
20000 7FFFF	0 0	0 1	1 1	0.....0 1.....1			frei	frei	
80000 9FFFF	1 1	0 0	0 0	0.....0 1.....1			ROM 2	ROM 1	$\overline{\text{CSROM1-2}}$
A0000 BFFFF	1 1	0 0	1 1	0.....0 1.....1			ROM 4	ROM 3	$\overline{\text{CSROM3-4}}$
C0000 FFFFF	1 1	1 1	0 1	0.....0 1.....1			frei	frei	

Vollständige Adressdecodierung

$$\overline{CSRAM1-2} = \overline{\overline{A19}\,\overline{A18}\,\overline{A17}\,\overline{A16}\,\overline{A15}}$$

$$\overline{CSRAM3-4} = \overline{\overline{A19}\,\overline{A18}\,\overline{A17}\,\overline{A16}\,A15}$$

$$\overline{CSRAM5-6} = \overline{\overline{A19}\,\overline{A18}\,\overline{A17}\,A16\,\overline{A15}}$$

$$\overline{CSRAM7-8} = \overline{\overline{A19}\,\overline{A18}\,\overline{A17}\,A16\,A15}$$

$$\overline{CSROM1-2} = \overline{A19\,\overline{A18}\,\overline{A17}}$$

$$\overline{CSROM3-4} = \overline{A19\,\overline{A18}\,A17}$$

Unvollständige Adressdecodierung

$$\overline{CSRAM1-2} = \overline{\overline{A19}\,\overline{A16}\,\overline{A15}}$$

$$\overline{CSRAM3-4} = \overline{\overline{A19}\,\overline{A16}\,A15}$$

$$\overline{CSRAM5-6} = \overline{\overline{A19}\,A16\,\overline{A15}}$$

$$\overline{CSRAM7-8} = \overline{\overline{A19}\,A16\,A15}$$

$$\overline{CSROM1-2} = \overline{A19\,\overline{A17}}$$

$$\overline{CSROM3-4} = \overline{A19\,A17}$$

Zur Zusatzfrage:

Adresse	**Adresse (binär)**								**Chip Select-Signal**
(hex.)	A19	A18	A17	A16	A15	A14…A0	D15…D8	D7…D0	
88000 A7FFF	1 1	0 0	0 1	0 0	1 0	0.....0 1.....1	ROM 2	ROM 1	$\overline{\text{CSROM1-2}}$
A8000 C7FFF	1 1	0 1	1 0	0 0	1 0	0.....0 1.....1	ROM 4	ROM 3	$\overline{\text{CSROM3-4}}$

Vollständige Adressdekodierung:

$$\overline{CSROM1-2} = \overline{A19\,\overline{A18}\,\overline{A17}\,\overline{A16}\,A15 \vee A19\,\overline{A18}\,\overline{A17}\,A16\,\overline{A15} \vee \vee A19\,\overline{A18}\,\overline{A17}\,A16\,A15 \vee A19\,\overline{A18}\,A17\,\overline{A16}\,\overline{A15}}$$

$$\overline{CSROM3-4} = \overline{A19\,\overline{A18}\,A17\,\overline{A16}\,A15 \vee A19\,\overline{A18}\,A17\,A16\,\overline{A15} \vee \vee A19\,\overline{A18}\,A17\,A16\,A15 \vee A19\,A18\,\overline{A17}\,\overline{A16}\,\overline{A15}}$$

Die Gleichungen lassen sich wie folgt vereinfachen:

$$\overline{CSROM1-2} = \overline{A19\,\overline{A18}\,\overline{A17}\,A15 \vee A19\,\overline{A18}\,\overline{A17}\,A16 \vee A19\,\overline{A18}\,A17\,\overline{A16}\,\overline{A15}}$$

$$\overline{CSROM3-4} = \overline{A19\,\overline{A18}\,A17A15 \vee A19\,\overline{A18}\,A17\,A16 \vee A19A18\,\overline{A17}\,\overline{A16}\,\overline{A15}}$$