J. Reichardt

Lehrbuch Digitaltechnik, 3. Auflage

Lösungen zu den Aufgaben (3.A_V2.1, 24.06.14)

1 Einleitung

A Aufgabe 1.1

a) s. Bild 1.1

b) s. Bild 1.2

c) s. die Aufzählungsliste in Kap 1.2

d) s. Kap. 1.2 letzter Absatz.

2 Modellierung digitaler Schaltungen

A Aufgabe 2.1

a) s. den Merksatz in Kap. 2.2

b) s. die einführenden Saätze in Kap. 2.3.1, 2.3.2 und 2.3.3

c) s. die Merksätze in Kap. 2.4.1 und 2.4.2

d) s. Erläuterungen zu Bild 2.2

e) s. Erläuterungen zu Bild 2.6

3 Boole´sche Algebra

A Aufgabe 3.1

C	B	A	$A \wedge B$	$A \wedge C$	$B \vee C$	$(A \wedge B) \vee (A \wedge C)$	$A \wedge (B \vee C)$
0	0	0	0	0	0	0	0

Aufgabenlösungen zu: Jürgen Reichardt, Lehrbuch Digitaltechnik 3. Auflage
ISBN 978-3-486-72765-4. Oldenbourg Wissenschaftsverlag, München 2013.

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

Die letzten beiden Spalte sind identisch. Dies bestätigt die Gültigkeit des Distributivgesetzes.

A Aufgabe 3.2

a) $A \vee (\overline{A} \wedge B) = (A \vee \overline{A}) \wedge (A \vee B) = 1 \wedge (A \vee B) = A \vee B$

b) $A \vee (A \wedge B) = (A \wedge 1) \vee (A \wedge B) = A \wedge (1 \vee B) = A \wedge 1 = A$

c) $(A \vee B) \wedge (\overline{A} \vee B) = (A \wedge \overline{A}) \vee B = 0 \vee B = B$

A Aufgabe 3.3

a) $(A \barwedge B) \barwedge C \neq A \barwedge (B \barwedge C)$

C	B	A	$A \barwedge B$	$(A \barwedge B) \barwedge C$	$B \barwedge C$	$A \barwedge (B \barwedge C)$
0	0	0	1	1	1	1
0	0	1	1	1	1	0
0	1	0	1	1	1	1
0	1	1	0	1	1	0
1	0	0	1	0	1	1
1	0	1	1	0	1	0
1	1	0	1	0	0	1
1	1	1	0	1	0	1

Die zweite und vierte Spalte der rechten Seite der Wahrheitstabelle sind unterschiedlich. Daher gilt das Assoziativgesetz nicht für die NAND-Verknüpfung.

b) $(A \barvee B) \barvee C \neq A \barvee (B \barvee C)$

C	B	A	$A \barvee B$	$(A \barvee B) \barvee C$	$B \barvee C$	$A \barvee (B \barvee C)$
0	0	0	1	0	1	0
0	0	1	0	1	1	0
0	1	0	0	1	0	1

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

Die zweite und vierte Spalte der rechten Seite der Wahrheitstabelle sind unterschiedlich. Daher gilt das Assoziativgesetz nicht für die NOR-Verknüpfung.

A Aufgabe 3.4

Die Lösungen werden für die XOR-Verknüpfung dargestellt. Die entsprechenden Lösungen für die XNOR-Verknüpfung ergeben sich durch analogen Lösungsansatz.

a) Distributivgesetz:

$(A \wedge B) \leftrightarrow (A \wedge C) = ((A \wedge B) \wedge (\overline{A \wedge C})) \vee ((\overline{A \wedge B}) \wedge (A \wedge C))$

$= ((\overline{A} \vee \overline{B}) \wedge A \wedge C) \vee (A \wedge B \wedge (\overline{A} \vee \overline{C}))$

$= (((\overline{A} \vee \overline{B}) \wedge A) \wedge C) \vee (B \wedge (A \wedge (\overline{A} \vee \overline{C})))$

$= (A \wedge \overline{B} \wedge C) \vee (B \wedge A \wedge \overline{C}) = A \wedge ((\overline{B} \wedge C) \vee (B \wedge \overline{C}))$

$= A \wedge (B \leftrightarrow C)$

b) Adsorptionsgesetz:

$A \wedge (\overline{A} \leftrightarrow B) = A \wedge ((A \wedge B) \vee (\overline{A} \wedge \overline{B}))$

$= (A \wedge A \wedge B) \vee (A \wedge \overline{A} \wedge \overline{B})$

$= (A \wedge B) \vee 0 = A \wedge B$

c) De Morgan'sches Gesetz:

$\neg (A \leftrightarrow B) = \neg ((A \wedge \overline{B}) \vee (\overline{A} \wedge B)) = \neg (A \wedge \overline{B}) \wedge \neg (\overline{A} \wedge B)$

$= (\overline{A} \vee B) \wedge (A \vee \overline{B}) = A \leftrightarrow B$ (POS des XNOR)

$= (\overline{A} \wedge A) \vee (\overline{A} \wedge \overline{B}) \vee (B \wedge A) \vee (B \wedge \overline{B})$ (Distributivgesetz)

$= 0 \vee (\overline{A} \wedge \overline{B}) \vee (B \wedge A) \vee 0 = \overline{A} \leftrightarrow \overline{B}$ (SOP des XNOR)

A Aufgabe 3.5

$(A \leftrightarrow B) \leftrightarrow C = A \leftrightarrow (B \leftrightarrow C)$

C	B	A	A↔ B	(A↔B) ↔ C	B↔C	A ↔ (B↔C)
0	0	0	0	0	0	0
0	0	1	1	1	0	1
0	1	0	1	1	1	1
0	1	1	0	0	1	0
1	0	0	0	1	1	1
1	0	1	1	0	1	0
1	1	0	1	0	0	0
1	1	1	0	1	0	1

Die zweite und vierte Spalte der rechten Seite der Wahrheitstabelle sind identisch. Daher gilt das Assoziativgesetz für die XOR-Verknüpfung.

A Aufgabe 3.6

a) Das „Ausklammern“ des UND- oder des ODER-Operators aus Produkt- bzw. Summentermen. Vgl. Kap. 3.4.4.

b) Die Inversion eines Produkt- bzw- eines Summenterms. Vgl. Kap. 3.4.5

c) Konjunktion: UND-Verknüpfung, Disjuntion: ODER-Verknüpfun (vgl. Kap. 3.3.2 und Kap. 3.3.3)

d) Produktterm = Minterm = UND-verknüpfter Ausdruck,

Summenterm = Maxterm = ODER-verknüpfter Ausdruck.

e) Disjunktive Normalform = Darstellung eines logischen Ausdrucks mit N Eingangssignalen durch ODER-Verknüpfung von Mintermen, die alle N Eingangssignale entweder invertiert oder nichtinvertiert enthalten (vgl. Kap. 3.6.1)..

Konjunktive Normalform = Darstellung eines logischen Ausdrucks mit N Eingangssignalen durch UND-Verknüpfung von Maxtermen, die alle N Eingangssignale entweder invertiert oder nichtinvertiert enthalten (vgl. Kap. 3.6.2).

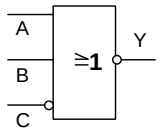
f) Durch doppelte Inversion des SOP-Ausdrucks. Durch Anwendung des de Morgan’schen Gesetzes wird die äußere ODER Vernüpfung in eine NAND-Funktion überführt (vgl. Bsp. d) in Kap. 3.5.2).

g) Mit einem XOR-Gatter mit 2 Eingängen, dazu wird z.B. das Eingangssignal B als Steuersignal interpretiert. Die Wahrheitstabelle des XOR-Gatters (vgl. z.B. Y_6 in Bild 3.3) zeigt, dass für B=0 das Signal A am Ausgang Y nicht invertiert erscheint und für B=1 invertiert.

h) Vgl. Tab. 3.4

i) Ein Multiplexer hat die Funktion eines Auswahlschalters (1 aus n) für digitale Signale. Vgl. den Text zu Bild 2.3.

A Aufgabe 3.7



C	B	A	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

A Aufgabe 3.8

a) $Y1 = (X1 \wedge X2 \wedge X3) \vee (\overline{X2} \wedge X3) = X3 \wedge ((X1 \wedge X2) \vee \overline{X2})$ (Distributivgesetz)

$= X3 \wedge (X1 \vee \overline{X2})$ (Adsorptionsgesetz)

$= (X3 \wedge X1) \vee (X3 \wedge \overline{X2})$ (Distributivgesetz)

b) $Y2 = (\overline{X1} \wedge \overline{X2} \wedge \overline{X3}) \vee (\overline{X1} \wedge X2 \wedge X3) \vee (X1 \wedge X2 \wedge X3) \vee$

$(X1 \wedge \overline{X2} \wedge \overline{X3}) \vee (X1 \wedge X2 \wedge \overline{X3}) \vee (\overline{X1} \wedge X2 \wedge \overline{X3})$

Durch Ausklammern von jeweils zwei UND-verknüpften Signalen aus den Klammern erhält man:

$Y2 = ((X2 \wedge X3) \vee (\overline{X1} \wedge X1)) \vee$

$((X2 \wedge \overline{X3}) \vee (\overline{X1} \wedge X1)) \vee$

$((\overline{X3} \wedge \overline{X2}) \vee (\overline{X1} \wedge X1)) = (X2 \wedge X3) \vee (X2 \wedge \overline{X3}) \vee (\overline{X3} \wedge \overline{X2})$

$Y2 = (X2 \wedge X3) \vee (\overline{X3} \wedge (X2 \vee \overline{X2}) = (X2 \wedge X3) \vee \overline{X3}$

$Y2 = X2 \vee \overline{X3}$ (Adsorptionsgesetz)

Durch die Methode der KV-Minimierung (vgl. Kap. 6) lässt sich dieses Ergebnis bestätigen.

A Aufgabe 3.9

$$Y = Y1 \vee Y2 = Y4 \vee Y3 \vee \neg (\overline{B} \wedge \overline{C}) = Y4 \vee Y3 \vee B \vee C$$

$$= (A \vee \overline{A}) \vee \neg (Y5 \wedge \overline{D}) \vee B \vee C = 1 \vee \neg (Y5 \wedge \overline{D}) \vee B \vee C = 1$$

Die Schaltung liefert also unabhängig von den Eingangssignalen immer eine 1.

A Aufgabe 3.10

Gewünschte Wahrheitstabelle:

S	A	Y
0	0	0
0	1	1
1	0	1
1	1	0

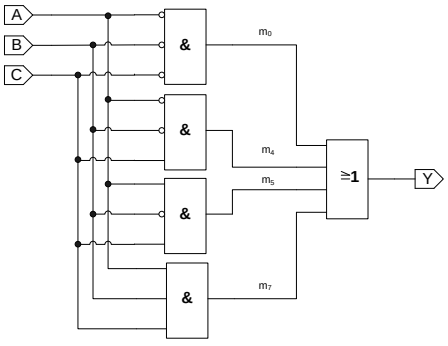
Die dazu gehörige DNF ist: $Y = (A \wedge \overline{S}) \vee (\overline{A} \wedge S)$.

Dies ist nach Tabelle 3.4 die SOP-Darstellung eines XOR-Gatters.

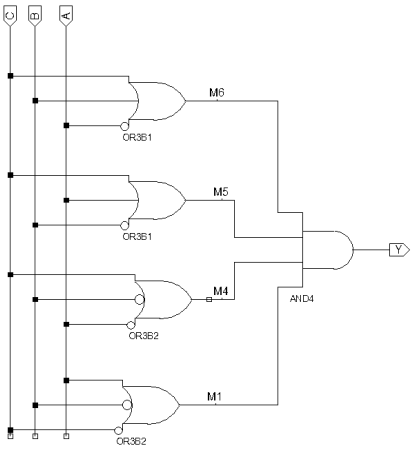
A Aufgabe 3.11

DNF : $Y = m_0 \vee m_4 \vee m_5 \vee m_7$ KNF : $Y = M_6 \wedge M_5 \wedge M_4 \wedge M_1$

Die DNF wird unter Verwendung der DIN-Symbole mit teilweise invertierten Eingängen dargestellt:



Die Darstellung der KNF verwendet US-Symbole, ebenfalls mit teilweise invertierten Eingängen:



A Aufgabe 3.12

C	B	A	Y1	Y2
0	0	0	1	1
0	0	1	1	1
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	0

DNF zu Y1: $Y1 = m_0 \vee m_1 \vee m_2 \vee m_4 \vee m_5 \vee m_6$

KNF zu Y1: $Y1 = M_4 \wedge M_0$

DNF zu Y2: $Y2 = m_0 \vee m_1 \vee m_2 \vee m_4 \vee m_6$

KNF zu Y2: $Y2 = M_4 \wedge M_2 \wedge M_0$

A Aufgabe 3.13

Die sich aus der Schaltung ergebende Wahrheitstabelle ist abgebildet. Die im Vergleich mit den gemessenen Werten korrekten Signale werden markiert:

C	B	A	U1	U2	U3	U4	U5	U6 = Y
0	0	0	1 ✓	1 ✓	1 ✓	0 ☠	0 ✓	0 ✓
0	0	1	0 ✓	1 ✓	1 ✓	1 ✓	0 ✓	0 ✓
0	1	0	1 ✓	1 ✓	1 ✓	0 ☠	0 ✓	0 ✓
0	1	1	0 ✓	1 ✓	1 ✓	1 ✓	0 ✓	0 ✓
1	0	0	1 ✓	1 ✓	0 ✓	0 ✓	0 ✓	0 ✓
1	0	1	0 ✓	1 ✓	0 ✓	1 ✓	0 ✓	0 ✓
1	1	0	1 ✓	0 ✓	0 ✓	0 ✓	1 ✓	1 ✓
1	1	1	0 ✓	0 ✓	0 ✓	1 ✓	0 ✓	0 ✓

Die Analyse ergibt, dass das ODER-Gatter U4 einen Fehler an dem mit U3 verbundenen Eingang hat: Dieser liegt dauerhaft auf 0 (Stuck-at-zero-Fehler). Am Ausgang der Schaltung wird dieser Fehler nicht bemerkt!

4 VHDL-Einführung I

A Aufgabe 4.1

Das VHDL-Modell verwendet eine selektive Signalzuweisung zur Darstellung der Wahrheitstabelle des Paritätsgenerators:

```
entity PARGEN is
   port( A, B, C : in bit   ;
         P_E : out bit );
end PARGEN;

architecture A of PARGEN is
signal YINT: bit_vector(2 downto 0);
begin
 YINT <= (C,B,A); -- Aggregat zur Buendelung von einzelnen Bits
 with YINT select
    P_E   <= '0' when "000",
             '1' when "001",
             '1' when "010",
             '0' when "011",
             '1' when "100",
             '0' when "101",
             '0' when "110",
             '1' when "111";
end A;
```

Aufgabenlösungen zu: Jürgen Reichardt, Lehrbuch Digitaltechnik 3. Auflage
ISBN 978-3-486-72765-4. Oldenbourg Wissenschaftsverlag, München 2013.

Alternativ wäre z.B. auch ein Datenflussmodell mit XOR-Gattern denkbar.

A Aufgabe 4.2

a) entity und architecture

b) Simulationssemantik: Nebenläufige Anweisungen werden in beliebiger Reihenflge (quasi gleichzeitg) ausgeführt.

Synthesesemantik: Nebenläufige Anweisungen entsprechen unterschiedliche Hardwareblöcken, die völlig unabhängig voneinander aktiv sind.

c) Unbedingte, bedingte und selektive (vgl. Kap. 4.3.3)

d) IN und OUT; bei bidirektionaler Datenübertragung ggf. auch INOUT (vgl. Tab. 4.1)

e) Mit einer selektiven Signalzuweisung

f) Stimulusgenerator, Device-Under-Test (DUT) und Response-Monitor (vgl. Bild 4.7).

g) Mit einer assert-Anweisung (vgl. Listing 4.7)

A Aufgabe 4.3

a) korrekt

b) korrekt

c) falsch: Ziffer am Anfang

d) falsch: enthält Sonderzeichen -

e) falsch: **`with`** ist VHDL-Schlüsselwort

A Aufgabe 4.4

```
entity AUFG_4_4 is
        port(E1, E2, E3: in bit;
        Y1, Y2: out bit);
end AUFG_4_4;
architecture VERHALTEN of AUFG_4_4 is
begin
        Y1 <=(E1 and E2) or E3;
        Y2 <=(E1 or E2) and E3;
end VERHALTEN;
```

A Aufgabe 4.5

```
entity AUFG_4_5 is
        port(E, S: in bit;
        Y: out bit);
end AUFG_4_5;
architecture VERHALTEN of AUFG_4_5 is
```

```
begin
        Y <= E when S = '0' else not E;
end VERHALTEN;
```

Das Implementierungsergebnis ist abhängig von der gewählten PLD-Technologie:

bei einem Coolrunner-II-CPLD wird: `Y<=((E AND NOT S) OR (NOT E AND S)`

bei einem XC9500-CPLD wird: `Y <= S XOR E.`

Man erkennt, dass in dem Coolrunner-II-CPLD die DNF des XOR-Gatters implementiert wird. im XC9500-CPLD wird hingegen das in diesen Bausteinen an den Ausgängen liegende XOR-Gatter verwendet.

A Aufgabe 4.6

```
entity AUFG_4_6 is
 port( S: in bit_vector(1 downto 0);
        A, B: in bit;
        Y: out bit);
end AUFG_4_6 ;
architecture VERHALTEN of AUFG_4_6 is
begin
        Y <= A and B when S="00"
        else A or B when S="01"
        else A nand B when S="10"
        else A nor B;
end VERHALTEN;
```

Das Implementierungsergebnis ergibt z.B. für einen Coolrunner-II-CPLD:

```
Y <= S(1)XOR ((B AND A) OR (B AND S(0)) OR (A AND S(0)))
```

5 Zahlensysteme in der Digitaltechnik

A Aufgabe 5.1

a) Byte = Eine Gruppe von 8 Bits, Nibble = Eine Gruppe von 4 Bits.

b) s. Merksatz vor Gl. (5.1)

c) Durch Inversion der einzelnen Bitstellen mt nachfolgender Addition von 1

d) -2_{10} = 0xE bzw 0xFE

e) Durch vorzeichengerechte Ergänzung (vgl. Tipp in Kap. 5.51)

f) Die Darstellung gebrochener rationaler Zahlen mit und ohne ganzzahligen Anteil (vgl. Gl. 5.9 bzw. Gl. 5.11).

g) Vorteil: Bei vorgegebener Bitbreite läßt sich in der Gleitkommadarstellung eine bessere Zahlenauflösung erreichen. Nachteil: Der Hardwareaufwand zur Implementierung arithmetischer Berechnungen ist signifikant größer.

A Aufgabe 5.2

a) $6B_{16}$

b) $14.D_{16}$

c) 331_8

d) 110111.010100_2

e) 1010111111111110_2

f) 101011100.00111000_2

A Aufgabe 5.3

a) 107_{10}

b) 20.8125_{10}

c) 217_{10}

d) 55.3125_{10}

e) 45054_{10}

f) 348.21875_{10}

A Aufgabe 5.4

a) 1111101_2

b) 6652_8

c) $5D2B_{16}$

d) 10402_5

e) $FE59_{16}$

A Aufgabe 5.5

a)

```
   110011
 + 11010
-------
   10010_   Carry
-------
  1001101
```

b)

```
  1100110
+ 1111001
-------
  1100000_   Carry
-------
 11011111
```

c)

```
   110011
 – 11010
-------
   11000_   Borrow
-------
   011001
```

d)

```
  1100110
– 1111001
-------
  1111001_   Borrow
-------
 11101101
```

Man beachte, dass diese Subtraktion zu einem negativen Ergebnis führt, denn es wird ein führendes Borrow-Bit benötigt.

A Aufgabe 5.6

Dezimal-zahl	Vorzeichen-Betrag	2er-Komplement
+25	0x19	0x19
+120	0x78	0x78
–42	0xAA	0xD6
–111	0xEF	0x91

A Aufgabe 5.7

a) 11010100 + 11101011 = $D4_{16}$ + EB_{16} = $1BF_{16}$

Beide Operanden sind negativ. Nach Streichung des führenden Bits ist das Ergebnis 0xBF ebenfalls negativ. Daher entsteht kein Überlauf.

b) 10111111 + 11011111 = BF_{16} + DF_{16} = $19E_{16}$

Beide Operanden sind negativ. Nach Streichung des führenden Bits ist das Ergebnis 0x9E ebenfalls negativ. Daher entsteht kein Überlauf.

c) 01011101 + 00110001 = $5D_{16} + 31_{16} = 8E_{16}$

Beide Operanden sind positiv, das führende Bit zeigt jedoch ein negatives Ergebnis an. Daher ist das Overflow-Bit zu setzen.

d) 01100001 + 00011111 = $61_{16} + 1F_{16} = 80_{16}$

Beide Operanden sind positiv, das führende Bit zeigt jedoch ein negatives Ergebnis an. Daher ist das Overflow-Bit zu setzen.

A Aufgabe 5.8

$12648430_{10} = C0FFEE_{16}$. Hoffentlich hilft er ☺.

A Aufgabe 5.9

a) 1.25 + 3.75 = 001.010 + 011.110 = 101.000. Das Ergebnis generiert einen Überlauf, da es im s2Q3-Format eine negative Zahl darstellt!

b) 3.75 + 0.875 = 011.110 + 000.111 = 100.101. Das Ergebnis generiert einen Überlauf, da es im s2Q3-Format eine negative Zahl darstellt!

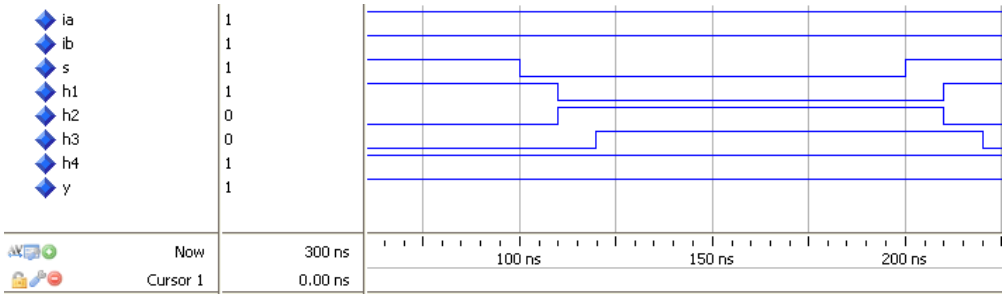
c) –3.75 + 1.0. Zunächst ist die 2er-Komplementdarstellung von –3.75 zu bestimmen: 100.010. Anschließend ist die Summe zu bilden: 100.010 + 001.000 = 101.010. Zur Überprüfung wird von diesem Ergebnis der Betrag gebildet, der das Resultat 2.75 ergibt.

d) 1.25 – 1.0. Zunächst ist die 2er-Komplementdarstellung von –1.0 zu bestimmen: 111.000. Die Subtraktion wird dann auf die Addition des 2er-Komplements zurückgeführt: 001.010 + 111.000 = 1000.010. Nach Streichung des führenden Bits ergibt sich das richtige Ergebnis: 0.25.

6 Logikminimierung

A Aufgabe 6.1

```
entity AUFGABE_6_1 is
        port( IA, IB, S: in bit;
        Y: out bit);
end AUFGABE_6_1 ;
architecture ARCH1 of AUFGABE_6_1  is
signal H1, H2, H3, H4 : bit; -- lokale Signale
begin
        H1 <= IB and S after 10 ns;
        H2 <= not S after 10 ns;
        H3 <= IA and H2 after 10 ns;
H4 <= IA and IB after 10 ns;
        Y <=  H1 or H3 or H4 after 10 ns;
end ARCH1;
```



A Aufgabe 6.2

a) Die DNF enthält in ihren Produkttermen alle Eingangssignale in invertierter oder nichtinvertierter Form, in der DMF sind in den Produkttermen möglicherweise einige Signale heraus optimiert worden.

b) Durch disjunktive Zusammenfassung der Einsen in einem KV-Diagramm (vgl. Merksatz zu Bild 6.7).

c) Unter Anwendung des Dualitätsprinzips durch disjunktive Zusammenfassung der Nullen in einem KV-Diagramm und anschliessende Inversion des Ausgangssignals (vgl. Beispiel 6.7).

d) Der Term enthält 3 Signale da durch den Viererblock 2 Signale heraus optimiert werden.

e) Vgl. den Merksatz zu Bild 6.4

f) Ein Output-Don't Care Term ist ein Minterm, dessen Ausgangssignal in der Anwendung entweder 0 oder 1 sein kann (Don't Care). Bei der KV-Minimierung kann dieser Term in die Primimplikanten mit einbeogen werden, um damit die Anzahl der Signale, die den Primimplikanten beschreiben, zu reduzieren.

h) An dem Auftreten von Schachbrettmusterstrukturen im KV-Diagramm.

i) Ein Strukturhazard bedeutet einen kurzzeitigen Signalwechsel eines eigentlich stabilen Signals (vgl. Bild 6.19). Ursache sind Laufzeitunterschiede in der kombinatorischen Logik (vgl. Bild 6.18). Strukturhazards lassen sich durch redundante Produktterme vermeiden.

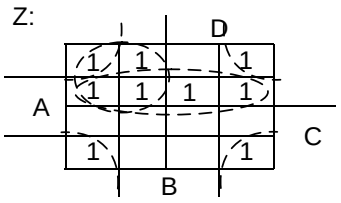
A Aufgabe 6.3

Zunächst wird die Wertigkeit der Eingangssignale festgelegt: Definiere A als MSB und D als LSB. Nun können die Minterme bestimmt werden, die zu den einzelnen Produkttermen der Bestimmungsgleichung von Z gehören:

$(\overline{A} \wedge B \wedge \overline{C} \wedge \overline{D}) = m_4$;

$(A \wedge \overline{C}) = m_8 \vee m_9 \vee m_{12} \vee m_{13}$;

$(\overline{A} \wedge \overline{B}) = m_0 \vee m_1 \vee m_2 \vee m_3$;

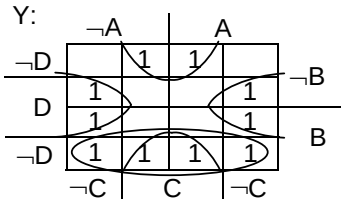


Nun werden die Kernimplikanten ODER-verknüpft. Man erkennt, dass sich nur die Breite des ersten Produktterms reduzieren lässt:

$Z = (\overline{C} \wedge \overline{D}) \vee (A \wedge \overline{C}) \vee (\overline{A} \wedge \overline{B})$

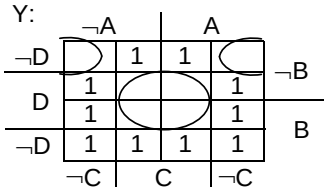
A Aufgabe 6.4

a) Bestimmung der DMF: Es werden drei Kernimplikanten gebildet:



$Y = (\overline{D} \wedge C) \vee (D \wedge \overline{C}) \vee (\overline{D} \wedge B)$

b) Bestimmung der KMF: Es werden zwei Kernimplikanten gebildet:

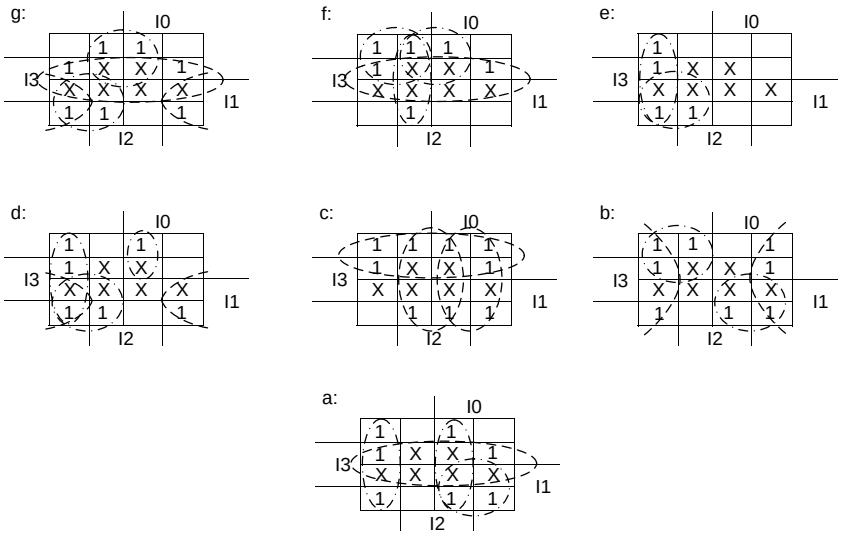


$\overline{Y} = (D \wedge C) \vee (\overline{D} \wedge \overline{C} \wedge \overline{B})$ => $Y = (\overline{D} \vee \overline{C}) \wedge (D \vee C \vee B)$

A Aufgabe 6.5

Zunächst wird eine Wahrheitstabelle für die einzelnen Segmente aufgestellt. Anschließend erfolgt die KV-Minimierung unter Berücksichtigung der Don't Care-Terme. Dabei sind mehrere Lösungen möglich. Die markierten Kernimplikanten wurden ausgewählt.

Dezimal-ziffer	I3	I2	I1	I0	g	f	e	d	c	b	a
0	0	0	0	0	0	1	1	1	1	1	1
1	0	0	0	1	0	0	0	0	1	1	0
2	0	0	1	0	1	0	1	1	0	1	1
3	0	0	1	1	1	0	0	1	1	1	1
4	0	1	0	0	1	1	0	0	1	1	0
5	0	1	0	1	1	1	0	1	1	0	1
6	0	1	1	0	1	1	1	1	1	0	0
7	0	1	1	1	0	0	0	0	1	1	1
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	0	0	1	1	1
-	1	0	1	0	x	x	x	x	x	x	x
-	1	0	1	1	x	x	x	x	x	x	x
-	1	1	0	0	x	x	x	x	x	x	x
-	1	1	0	1	x	x	x	x	x	x	x
-	1	1	1	0	x	x	x	x	x	x	x
-	1	1	1	1	x	x	x	x	x	x	x



Aufgabenlösungen zu: Jürgen Reichardt, Lehrbuch Digitaltechnik 3. Auflage
ISBN 978-3-486-72765-4. Oldenbourg Wissenschaftsverlag, München 2013.

$g = I3 \vee (I2 \wedge \overline{I1}) \vee (I1 \wedge \overline{I0}) \vee (\overline{I2} \wedge I1)$

$f = I3 \vee (I2 \wedge \overline{I0}) \vee (\overline{I1} \wedge \overline{I0}) \vee (I2 \wedge \overline{I1})$

$e = (\overline{I2} \wedge \overline{I0}) \vee (I1 \wedge \overline{I0})$

$d = (\overline{I2} \wedge \overline{I0}) \vee (I1 \wedge \overline{I0}) \vee (\overline{I2} \wedge I1) \vee (I2 \wedge \overline{I1} \wedge I0)$

$c = I2 \vee \overline{I1} \vee I0$

$b = \overline{I2} \vee (\overline{I1} \wedge \overline{I0}) \vee (I1 \wedge I0)$

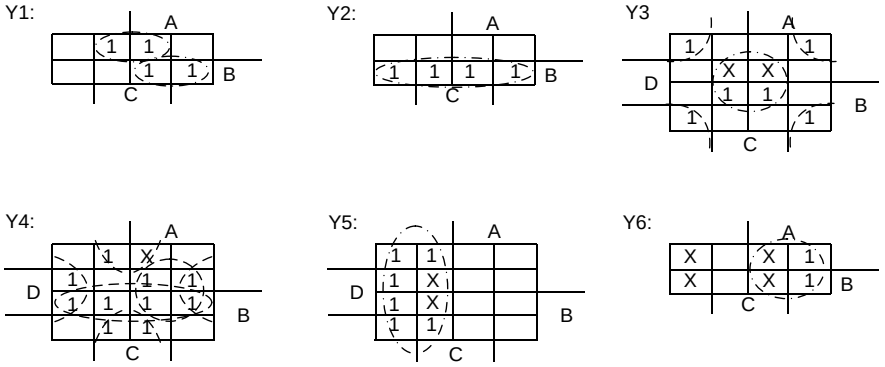
$a = I3 \vee (I1 \wedge I0) \vee (I2 \wedge I0) \vee (\overline{I2} \wedge \overline{I0})$

Im Falle einer Ansteuerung der Anzeige mit den Hexadezimalwerten 0xA … 0xF werden die folgenden logischen Werte anstatt der Don’t-Cares ausgegeben:

Hex.-zahl	I3	I2	I1	I0	g	f	e	d	c	b	a
0xA	1	0	1	0	1	1	1	1	0	1	1
0xB	1	0	1	1	1	1	0	1	1	1	1
0xC	1	1	0	0	1	1	0	0	1	1	1
0xD	1	1	0	1	1	1	0	1	1	0	1
0xE	1	1	1	0	1	1	1	1	1	0	1
0xF	1	1	1	1	1	1	0	0	1	1	1

A Aufgabe 6.6

Bei den Ausgangssignalen Y1, Y2 und Y6 sind die unteren acht Wahrheitstabelleneinträge Don't-Care. Für diese Signale können daher KV-Diagramme mit drei Eingangssignalen verwendet werden:



$Y1 = (\overline{B} \wedge C) \vee (A \wedge B)$; $Y2 = B$; $Y3 = (\overline{D} \wedge \overline{C}) \vee (D \wedge C) = D \leftrightarrow C$

$Y4 = (\overline{D} \wedge C) \vee (D \wedge \overline{C}) \vee (D \wedge A) \vee (D \wedge B) = (D \leftrightarrow C) \vee (D \wedge A) \vee (D \wedge B)$;

$Y5 = \overline{A}$; $Y6 = A$

A Aufgabe 6.7

Es lassen sich die folgenden konformen Terme identifizieren:

$F_1 = (A \vee B)$ und $F_2 = (C \wedge D)$.

Damit wird:

$F = F_1 \wedge F_2 \vee E$; $G = F_1 \wedge \overline{E}$; $H = F_2 \wedge E$

A Aufgabe 6.8

$Y = (A \wedge B \wedge C) \vee (A \wedge B \wedge D) \vee (\overline{A} \wedge \overline{C} \wedge \overline{D}) \vee (\overline{B} \wedge \overline{C} \wedge \overline{D})$

Die Anwendung des Distributivgesetzes liefert:

$Y = ((A \wedge B) \wedge (C \vee D)) \vee ((\overline{C} \wedge \overline{D}) \wedge (\overline{A} \vee \overline{B}))$

Mit dem de Morgan'schen Gesetz wird daraus:

$Y = ((A \wedge B) \wedge (C \vee D)) \vee (\neg(C \vee D) \wedge \neg(A \wedge B))$

Mit den Ausdrücken $F_1 = C \vee D$ und $F_2 = A \wedge B$ wird:

$F = (F_1 \wedge F_2) \vee (\overline{F_1} \wedge \overline{F_2}) = F_1 \leftrightarrow F_2$

7 VHDL-Einführung II

A Aufgabe 7.1

a) Ereignis-Warteschlange, Delta-Delay und Request-Update Prinzip (vgl. Kap. 7.3).

b) Alle Signale, die auf der rechten Seite einer Signalzuweisung stehen, sowie alle Signale, die in den Bedingungsausdrücken von if- oder case-Anweisungen stehen.

c) Ja, es handelt sich um eine sequenzielle Anweisung

d) Nein, selektive Signalzuweisungen dürfen nicht innerhalb von Prozessen verwendet werden.

e) über Signale

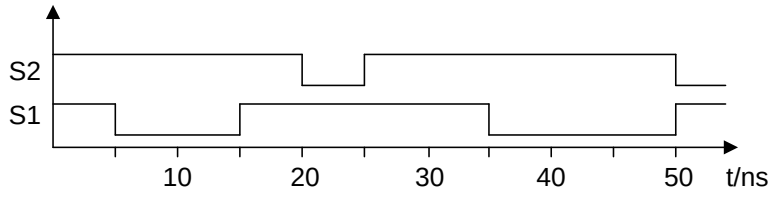
f) Sequenzielle Anweisungen sind, solche, die innerhalb von Prozessen sequentiell abgearbeitet werden. Nebenläufie Anweisungen sind äquivalent zu Prozessen und werden wie diese zeitgleich abgearbeitet.

g) Eine kombinatorische Schleife entsteht, wenn ein Ausgang eines kombinatorischen Logikblocks direkt auf dessen Eingang zurückgeführt wird. Zur Vermeidung ist es notwendig (aber nicht unbedingt hinreichend), dass Ausgangssignale eines kombinatorischen Prozesses nicht inder Senstivitätsliste des Prozesses stehen.

h) Am Ende des Prozesses

i) Variable sind nur innerhalb des Prozesses gültig, in dem sie deklariert sind. Die Zuweisung an Variablen erfolgt innerhalb des Prozesses sofort. Signale werden hingegen am Ende des Prozesses aktualisiert.

A Aufgabe 7.2



A Aufgabe 7.3

a) Ein Verhaltensmodell des Paritätscheckers kann z.B. wie folgt erstellt werden:

- Mit dem Koppelsignal CHK wird eine im Checker intern gebildete Parität der vier niederwertigen Bits gebildet (`case`-Anweisung im Prozesss CHECK).
- In einer nebenläufigen Anweisung erfolgt eine Äquivalenzüberprüfung des intern gebildeten Paritätsbits mit dem zusammen mit den Daten übertragenen Paritätsbit D(4). Dies erfordert ein XNOR-Gatter.

```
-- Paritaets Checker für gerade Parität
-------------------------------------------------
entity PAR_CHECK is
        port(D: in bit_vector(4 downto 0);
        OK: out bit);
end PAR_CHECK;

architecture VERHALTEN of PAR_CHECK is
signal CHK: bit;
begin
CHECK: process (D)
        begin
        case D(3 downto 0) is
                            when "0000" => CHK <= '0';
                            when "0011" => CHK <= '0';
                            when "0101" => CHK <= '0';
                            when "0110" => CHK <= '0';
                            when "1001" => CHK <= '0';
                            when "1010" => CHK <= '0';
                            when "1100" => CHK <= '0';
                            when "1111" => CHK <= '0';
                            when others => CHK <= '1';
        end case;
        end process CHECK;
        OK <= D(4) xnor CHK;
end VERHALTEN;
```

b) In der Testbench werden zunächst die Port-Signale des synthesefähigen Checkers als lokale Signale deklariert. Zusätzlich wird ein Signal TEST hinzugefügt, welches die Testnummer enthält. Zwei weitere Prozesse dienen als Stimulusgenerator bzw. zur Überprüfung der Ausgangssignale. Die durchnummerierten Tests sind so aufgebaut, dass für jedes Datenwort D(3..0) das Paritätsbit jeweils als 0 oder als 1 übertragen wird.

Eine Wahrheitstabelle mit 32 Zeilen dient zur Bestimmung des zu erwartenden Binärwertes des OK-Signals. Die zugehörigen Testnummern werden in zwei `assert`-Anweisungen ausgewertet. Bei der Auswertung wird darauf geachtet, dass der Test jeweils in der Mitte zwischen zwei unterschiedlichen D-Werten stattfindet.

```
-- Paritaets Checker für gerade Parität mit Testbench
-------------------------------------------------
entity AUFGABE_7_3 is
end AUFGABE_7_3;

architecture TESTBENCH of AUFGABE_7_3 is
signal D: bit_vector(4 downto 0);   -- Port-Eingangssignal
signal OK: bit;                     -- Port-Ausgangssignal
signal CHK: bit;                    -- Koppelsignal
signal TEST: integer range 1 to 32; -- Testbench Signal

begin
-------------------------------------------------------------
-- synthesefähige Prozesse
CHECK: process (D)
        begin
        case D(3 downto 0) is
                            when "0000" => CHK <= '0';
                            when "0011" => CHK <= '0';
                            when "0101" => CHK <= '0';
                            when "0110" => CHK <= '0';
```

```
                        when "1001" => CHK <= '0';
                        when "1010" => CHK <= '0';
                        when "1100" => CHK <= '0';
                        when "1111" => CHK <= '0';
                        when others => CHK <= '1';
        end case;
        end process CHECK;
        OK <= D(4) xnor CHK;
-------------------------------------------------------------
-- Testbench Prozesse
STIMULI: process
begin
D <= "00000"; TEST <= 1; wait for 100 ns;
D <= "10000"; TEST <= 2; wait for 100 ns;
D <= "00001"; TEST <= 3; wait for 100 ns;
D <= "10001"; TEST <= 4; wait for 100 ns;
D <= "00010"; TEST <= 5; wait for 100 ns;
D <= "10010"; TEST <= 6; wait for 100 ns;
D <= "00011"; TEST <= 7; wait for 100 ns;
D <= "10011"; TEST <= 8; wait for 100 ns;
D <= "00100"; TEST <= 9; wait for 100 ns;
D <= "10100"; TEST <= 10; wait for 100 ns;
D <= "00101"; TEST <= 11; wait for 100 ns;
D <= "10101"; TEST <= 12; wait for 100 ns;
D <= "00110"; TEST <= 13; wait for 100 ns;
D <= "10110"; TEST <= 14; wait for 100 ns;
D <= "00111"; TEST <= 15; wait for 100 ns;
D <= "10111"; TEST <= 16; wait for 100 ns;
D <= "01000"; TEST <= 17; wait for 100 ns;
D <= "11000"; TEST <= 18; wait for 100 ns;
D <= "01001"; TEST <= 19; wait for 100 ns;
D <= "11001"; TEST <= 20; wait for 100 ns;
D <= "01010"; TEST <= 21; wait for 100 ns;
D <= "11010"; TEST <= 22; wait for 100 ns;
D <= "01011"; TEST <= 23; wait for 100 ns;
D <= "11011"; TEST <= 24; wait for 100 ns;
D <= "01100"; TEST <= 25; wait for 100 ns;
D <= "11100"; TEST <= 26; wait for 100 ns;
D <= "01101"; TEST <= 27; wait for 100 ns;
D <= "11101"; TEST <= 28; wait for 100 ns;
D <= "01110"; TEST <= 29; wait for 100 ns;
D <= "11110"; TEST <= 30; wait for 100 ns;
D <= "01111"; TEST <= 31; wait for 100 ns;
D <= "11111"; TEST <= 32; wait for 100 ns;
end process STIMULI;

MONITOR: process
begin
  wait for 50 ns;
  case TEST is
    when 1|4|6|7|10|11|13|16|18|19|21|24|25|28|30|31
        => assert OK = '1' report "Uebertragungsfehler";
    when 2|3|5|8|9|12|14|15|17|20|22|23|26|27|29|32
        => assert OK = '0' report "Uebertragungsfehler";
  end case;
  wait for 50 ns;
end process MONITOR;

end TESTBENCH;
```

A Aufgabe 7.4

Der 1-zu-4-Demultiplexer wird mit Hilfe einer `case`-Anweisung realisiert. In der Testbench wird zunächst E = 1 nacheinander auf alle Ausgänge geschaltet und danach E = 0. Dies ergibt acht Tests, deren Ergebnisse in der Testbench überprüft werden.

```
-- 1-zu-4 Demultiplexer mit Testbench
-------------------------------------------------
entity AUFGABE_7_4 is
end AUFGABE_7_4;

architecture TESTBENCH of AUFGABE_7_4 is
signal E: bit;                        -- Daten Eingangssignal
signal SEL: bit_vector(1 downto 0);   -- Selektionssignal
signal Y: bit_vector(3 downto 0);     -- Daten Ausgangssignal
signal TEST: integer range 1 to 8;    -- Testbench Signal

begin
-------------------------------------------------------------
-- synthesefähige Prozesse
DMUX: process (E, SEL)
begin
   Y <= "0000";
        case SEL is
                  when "00" => Y(0)<= E;
                  when "01" => Y(1)<= E;
                  when "10" => Y(2)<= E;
                  when "11" => Y(3)<= E;
   end case;
end process DMUX;
-------------------------------------------------------------
-- Testbench Prozesse
STIMULI: process
begin
E <= '1';
TEST <= 1; SEL <= "00"; wait for 100 ns;
TEST <= 2; SEL <= "01"; wait for 100 ns;
TEST <= 3; SEL <= "10"; wait for 100 ns;
TEST <= 4; SEL <= "11"; wait for 100 ns;
E <= '0';
TEST <= 5; SEL <= "00"; wait for 100 ns;
TEST <= 6; SEL <= "01"; wait for 100 ns;
TEST <= 7; SEL <= "10"; wait for 100 ns;
TEST <= 8; SEL <= "11"; wait for 100 ns;
end process STIMULI;

MONITOR: process
begin
  wait for 50 ns;
  case TEST is
    when 1 => assert Y(0) = '1' report "Fehler Test Nr. 1";
    when 2 => assert Y(1) = '1' report "Fehler Test Nr. 2";
    when 3 => assert Y(2) = '1' report "Fehler Test Nr. 3";
    when 4 => assert Y(3) = '1' report "Fehler Test Nr. 4";
    when 5 => assert Y(0) = '0' report "Fehler Test Nr. 5";
    when 6 => assert Y(1) = '0' report "Fehler Test Nr. 6";
    when 7 => assert Y(2) = '0' report "Fehler Test Nr. 7";
    when 8 => assert Y(3) = '0' report "Fehler Test Nr. 8";
```

```
  end case;
  wait for 50 ns;
end process MONITOR;

end TESTBENCH;
```

A Aufgabe 7.5

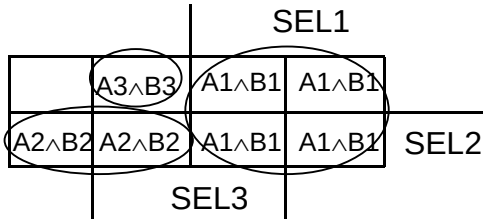
Der Codeumsetzer wird mit einer `case`-Anweisung realisiert. Die zu erstellende Testbench ähnelt der aus den Aufgaben 7.3 und 7.4.

```
-- Gray-Binär Codeumsetzer
-------------------------------------------------
entity AUFGABE_7_5 is
    port( G: in bit_vector(3 downto 0);  -- Daten Eingangssignal
          B: out bit_vector(3 downto 0)  -- Daten Ausgangssignal
    );
end AUFGABE_7_5;

architecture VERHALTEN of AUFGABE_7_5 is
begin
-------------------------------------------------------------
-- synthesefähiger Prozess
UMSETZER: process (G)
begin
        case G is
                  when x"0" => B <= x"0";
                  when x"1" => B <= x"1";
                  when x"2" => B <= x"3";
                  when x"3" => B <= x"2";
                  when x"4" => B <= x"7";
                  when x"5" => B <= x"6";
                  when x"6" => B <= x"4";
                  when x"7" => B <= x"5";
                  when x"8" => B <= x"F";
                  when x"9" => B <= x"E";
                  when x"A" => B <= x"C";
                  when x"B" => B <= x"D";
                  when x"C" => B <= x"8";
                  when x"D" => B <= x"9";
                  when x"E" => B <= x"B";
                  when x"F" => B <= x"A";
   end case;
end process UMSETZER;
end VERHALTEN;
```

A Aufgabe 7.6

```
architecture A2 of TEST is
begin
P1: process(A1, A2, A3, B1, B2, B3, SEL1, SEL2, SEL3)
begin
   Y <= '0';
        if SEL1 = '1' then Y <= A1 and B1;
        elsif SEL2 = '1' then Y <= A2 and B2;
        elsif SEL3 = '1' then Y <= A3 and B3;
   end if;
end process P1;
end A2;
```

SEL3	SEL2	SEL1	Y
0	0	0	0
0	0	1	A1∧B1
0	1	0	A2∧B2
0	1	1	A1∧B1
1	0	0	A3∧B3
1	0	1	A1∧B1
1	1	0	A2∧B2
1	1	1	A1∧B1



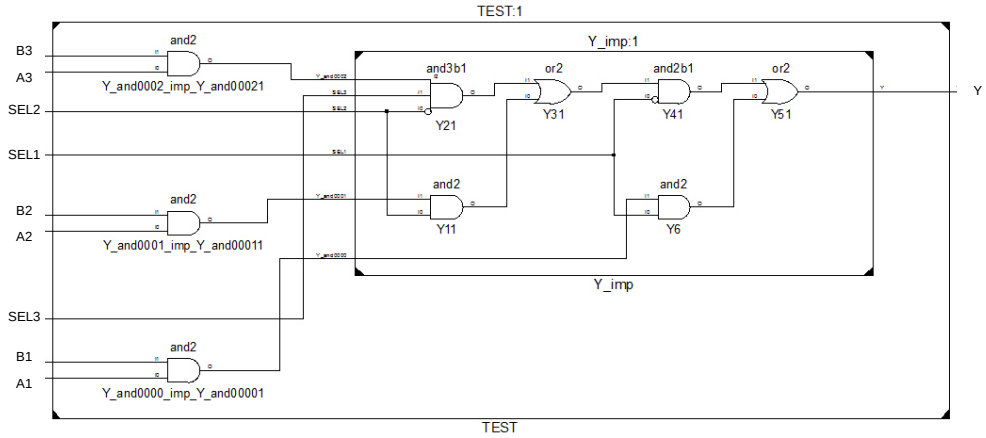
Synthetisierte logische Gleichungen für einen CPLD-Baustein (DMF):

Y <= ((SEL1 AND B1 AND A1)

OR (SEL2 AND B2 AND A2 AND NOT SEL1)

OR (NOT SEL2 AND NOT SEL1 AND SEL3 AND A3 AND B3));

RTL-Schematic aus ISE (V13.3): Der Schaltplan implementiert in dem Macro Y_imp_1 zwei 2-zu-1 Multiplexer in Reihe. Beide Multiplexer repräsentieren je einen if/else(if)-Zweig. Der Schaltplan stellt keine disjunktive Minimalform dar. Diese wird erst während des Design-Fittings erstellt.



8 Codes

A Aufgabe 8.1

a) s. Kap. 8.3

b) Die binäre Codierung von Dezimalzahlen. Tab. 8.2 zeigt eine Auswahl verschiedener BCD-Codes.

c) S. Merksatz und Beispiel 8.3

d) der Gray-Code, vgl. Beispiel.8.5

e) Indem zwischen je zwei benachbarten Codeworten die Anzahl unterschiedlicher Bitstellen bestimmt wird. Anhand der Hamming-Distanz eines Codes lässt sich erkennen, ob dieser für eine Fehlererkennung oder ggf. für eine Fehlerkorrektur geeignet ist.

f) Rekursive Reflexion (vgl. Tabelle 8.3)

g) Die Hamming-Distanz H muss mindestens 2 sein.

h) Falls bei der Addition einer einzelnen Bitstelle das Ergebnis größer als 9 ist, so muss in dieser Bitstelle zusätzlich +6 addiert werden (vgl. Tipp in Kap. 8.3).

A Aufgabe 8.2

b_3: 8	b_2: 4	b_1: 3	b_0: –2	Z
0	0	0	0	0
0	0	0	1	x
0	0	1	0	3
0	0	1	1	1
0	1	0	0	4
0	1	0	1	2
0	1	1	0	7
0	1	1	1	5
1	0	0	0	8
1	0	0	1	6
1	0	1	0	x
1	0	1	1	9
1	1	0	0	x
1	1	0	1	x
1	1	1	0	x
1	1	1	1	x

Aufgabenlösungen zu: Jürgen Reichardt, Lehrbuch Digitaltechnik 3. Auflage
ISBN 978-3-486-72765-4. Oldenbourg Wissenschaftsverlag, München 2013.

Dieser Code ist bewertbar. Er hat Hamming-Distanzen zwischen 1 und 4, er ist daher nicht stetig und besitzt eine Redundanz von 0.68 Bit.

A Aufgabe 8.3

a) 0011 0110 0101 0000

b) 0001 0010 0011 0100

c) 0001 0000 0000 0010

d) 0110 0111 1000 1001

A Aufgabe 8.4

a)

```
  0111
  0011
  0110   (+ 6₁₀ da Summe > 1001)
-------
 1 101_
  1___   (Carry)
-------
 1 0000
```

b)

```
  0111
  1001
  0110   (+ 6₁₀ da Summe > 1001)
-------
 1 111_  (Carry)
-------
 1 0110
```

c)

```
  1001
  1000
  0110   (+ 6₁₀ da Summe > 1001)
-------
 1 000_  (Carry)
-------
 1 0111
```

d)

```
   0110   1001
  +0110   0110
-------------
   0110   0110   (+ 6₁₀ da Summe > 1001)
-------------
   0101   110_
 1 ____          (Carry)
-------------
```

Aufgabenlösungen zu: Jürgen Reichardt, Lehrbuch Digitaltechnik 3. Auflage
ISBN 978-3-486-72765-4. Oldenbourg Wissenschaftsverlag, München 2013.

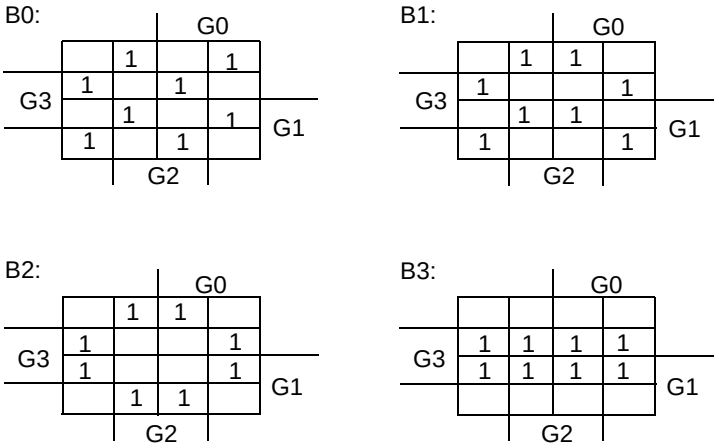
1 0011 0101

A Aufgabe 8.5

a) Der VHDL-Code entspricht dem der Aufgabe 7.5. Die Die Synthese und Implementierung in einen Coolrunner-II-CPLD ergibt die folgenden logischen Gleichungen:

```
B(3) <= G(3);
B(2) <= ((G(3) AND NOT G(2))
      OR (NOT G(3) AND G(2)));
B(1) <= G(1)
      XOR ((G(3) AND NOT G(2))
      OR (NOT G(3) AND G(2)));
B(0) <= G(1)
      XOR ((G(3) AND G(2) AND G(0))
      OR (G(3) AND NOT G(2) AND NOT G(0))
      OR (NOT G(3) AND G(2) AND NOT G(0))
      OR (NOT G(3) AND NOT G(2) AND G(0)));
```

b) Die KV-Diagramme werden aus der Wahrheitstabelle des Code-Umsetzers abgeleitet:



Daraus lässt sich unter Verwendung von XOR-Gattern ableiten:

B0 = G3 ↔ G2 ↔ G1 ↔ G0 (Schachbrettmuster von Einsen)

$B1 = (\overline{G3} \wedge G2 \wedge \overline{G1}) \vee (G3 \wedge \overline{G2} \wedge \overline{G1}) \vee (G3 \wedge G2 \wedge G1) \vee (\overline{G3} \wedge \overline{G2} \wedge G1)$

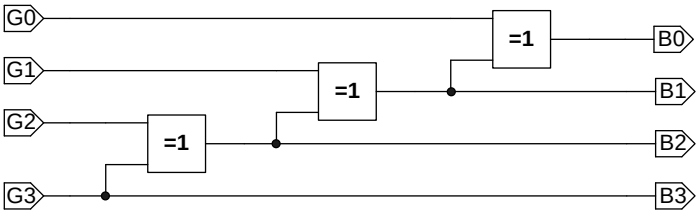
$= (\overline{G3} \wedge ((G2 \wedge \overline{G1}) \vee (\overline{G2} \wedge G1)) \vee (G3 \wedge ((G2 \wedge G1) \vee (\overline{G2} \wedge \overline{G1})))$

= G3 ↔ G2 ↔ G1

$B2 = (\overline{G3} \wedge G2) \vee (G3 \wedge \overline{G2}) = G3 \leftrightarrow G2$

$B3 = G3$

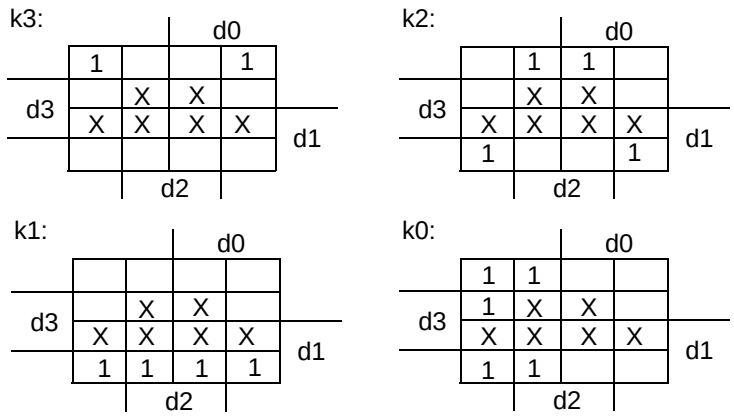
Dies entspricht der Lösung aus a). Unter Verwendung von XOR-Gattern ergibt sich daraus der folgende Schaltplan:



A Aufgabe 8.6

Zunächst ist die Wahrheitstabelle zu bestimmen:

d	d3	d2	d1	d0	k = 9 – d	k3	k2	k1	k0
0	0	0	0	0	9	1	0	0	1
1	0	0	0	1	8	1	0	0	0
2	0	0	1	0	7	0	1	1	1
3	0	0	1	1	6	0	1	1	0
4	0	1	0	0	5	0	1	0	1
5	0	1	0	1	4	0	1	0	0
6	0	1	1	0	3	0	0	1	1
7	0	1	1	1	2	0	0	1	0
8	1	0	0	0	1	0	0	0	1
9	1	0	0	1	0	0	0	0	0
-	1	0	1	X	-	X	X	X	X
-	1	0	X	X	-	X	X	X	X



Aus den KV-Diagrammen erhält man durch Zusammenfassung der Kernimplikanten:

$k3 = \overline{d3} \wedge \overline{d2} \wedge \overline{d1}$

$k2 = (\overline{d2} \wedge d1) \vee (d2 \wedge \overline{d1})$

$k1 = d1$

$k0 = \overline{d0}$

9 Physikalische Implementierung und Beschaltung von Logikgattern

A Aufgabe 9.1

a) 8 : Das Gatter besitzt 4 NMOS sowie 4 PMOS Transistoren.

b) Der Inverter benötigt nur 2 Transistoren, der nichtinvertierende CMOS-Treiber hingegen 4 da das Invertersignal durch einen zweiten Inverter geführt werden muss.

c) Weil dadurch die Stromaufnahme stark ansteigt (vgl. Bild 9.5)

d) Weil durch Leckströme der Eingangspegel in den verbotenen Bereich kommen kann und damit im Gatter größere statische Querströme fliesen können (vgl. Bild 9.5)

e) Nur Tris-State oder Open-Collector bzw. Open-Drain Ausgänge

f) Durch das Wired-AND Prinzip lässt sich leicht eine zweite Logikstufe aufbauen. Nachtielig ist dabei jedoch die lange L→H Schaltzeit der Wired-AND Stufe.

g) I) Das Gattersymbol ist durch ein Open-Drain Symbol (Raute mit Unterstrich) gekennzeichnet. II) das Gatter ist durch das Three-State Symbol (auf dem Kopf stehendes Dreieck) gekennzeichnet.

h) Die Wahrheitstabelle enthält Nullen und Einsen, die Arbeitstabelle die L- und H-Pegel. Aussagekräftiger ist die Arbeitstabelle da diese das physikalische Verhalten auch für L-aktive Logik beschreiben (vgl. Bild 9.9).

i) Das die in den Datenblättern angegebenen Arbeitstabellen verwendet werden. So hat z.B. ein „7400-NAND“ Baustein in negativer Logik die Funktion eines NOR-Gatters.

j) Ein Three-State Ausgang kann zusätzlich zu den Pegeln Low und High auch den Wert „hochohmig Z“ annehmen (vgl. Erläuterungen zu Tab. 9.3).

k) Unter Verwendung des Datentyps std_logic im Zusammenhang entweder mit einer bedingten nebenläufigen Anweisung oder einem Prozess mit if-Satatement (vgl. Listing 9.2)

l) Der Pull-Up-Widerstand ersetzt den PMOS-Transistor eines Push-Pull Ausgangs, für mehrere Ausgänge wird ein gemeinsamer Pull-Up-Widerstand verwendet.

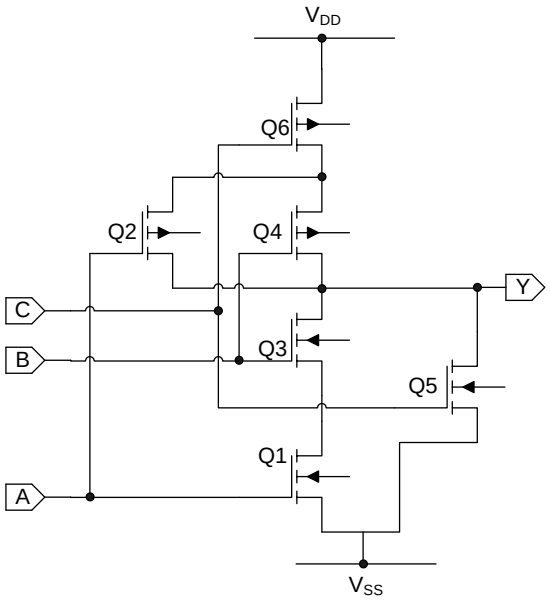
m) Der Datentyp std_logic ist aufgelöst. Das bedeutet, dass ein Signal durch mehrere Treiber (Prozesse bzw. nebenläufigen Anweisungen) definiert sein darf. Der Datentyp std_ulogic ist hingegen unaufgelöst, er darf nur durch einen Prozess oder nebenläufige Anweisung definiert sein.

n) Falls ein Signal versehentlich in mehreren Prozessen bzw. nebenläufigen Anweisungen eine Signalzuweisung erfährt, so fällt dieser Fehler im Compiler bzw. in einer VHDL-Simualtion häufiig nicht auf, sondern erst in der Regel spät bei der Hardware-Implementierung. Bei Verwendung eines aufgelösten Signals weist hingegen bereits der Simulations-Compiler auf diesen Fehler hin.

A Aufgabe 9.2

Die einzelnen Transistoren Qi sind wie folgt zusammenzuschalten:

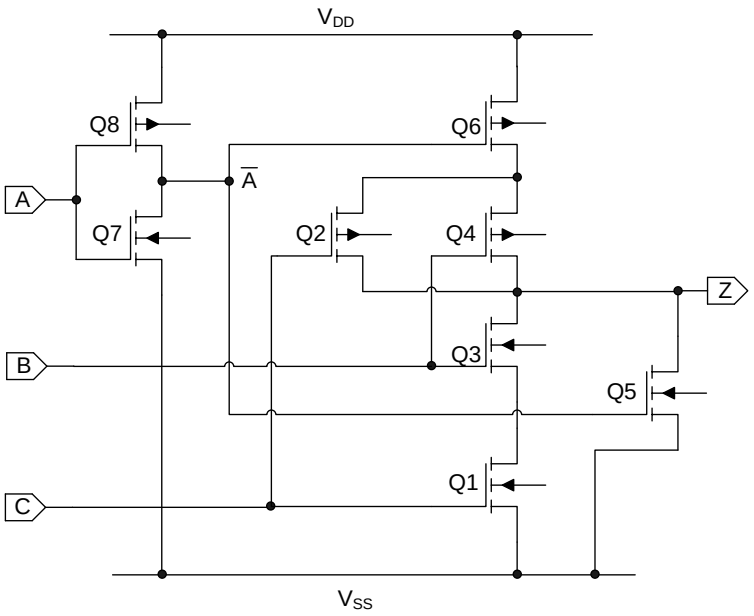
$Y = \neg(((Q1,Q2) \wedge (Q3,Q4)) \vee (Q5,Q6))$



A Aufgabe 9.3

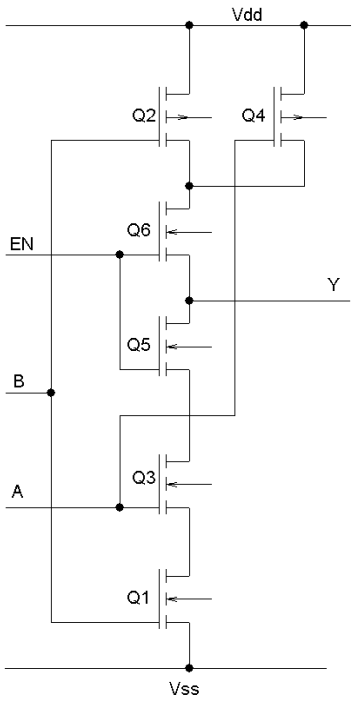
Das Signal A wird durch das Transistorpaar (Q7,Q8) invertiert und an die Gate-Eingänge des Paars (Q5,Q6) gelegt. Die zweite Stufe der Schaltung ist gegeben durch:

$Y = \neg ((Q5,Q6) \vee ((Q1,Q2) \wedge (Q3,Q4)))$



A Aufgabe 9.4

Die NAND-Funktion wird durch die Transistorpaare (Q1,Q2) und (Q3,Q4) aufgebaut. Mit den beiden NMOS-Transistoren Q5 und Q6 wird der Three-State-Ausgang realisiert.



Das synthesefähige VHDL-Modell des Gatters verwendet an den Eingängen den Datentyp `bit` und am Ausgang den Datentyp `std_logic`. Aus diesem Grunde muss die Funktion `To_stdlogic` verwendet werden.

```
-- NAND-Gatter mit Three-State-Ausgang
-------------------------------------------------
library ieee;
use ieee.std_logic_1164.all; -- für std_logic Datentyp erforderlich

entity AUFGABE_9_4 is
    port( A, B , EN: in bit;
          Y: out std_logic
    );
end AUFGABE_9_4;

architecture VERHALTEN of AUFGABE_9_4 is

function To_stdlogic ( b : bit) return std_logic is
begin
      case b is
```

```
            when '0' => return ('0');
            when '1' => return ('1');
            when others => return '0';
      end case;
end;

begin
-------------------------------------------------------------
-- synthesefähiger Prozess
P1: process (A, B, EN)
begin
   if EN='1' then
      Y <= To_stdlogic(A nand B);
   else
      Y <= 'Z';
  end if;
end process P1;
end VERHALTEN;
```

In einer Testbench, die die Wahrheitstabelle vollständig überprüft, können z.B. die folgenden Stimuliprozesse verwendet werden, die unabhängig von einander ein periodisch wechselndes Signal erzeugen:

```
P_EN: process
begin
   EN <='0'; wait for 50 ns;
   EN <='1'; wait for 50 ns;
end process P_EN;
P_A: process
begin
   EN <='0'; wait for 100 ns;
   EN <='1'; wait for 100 ns;
end process P_EN;
P_EN: process
begin
   EN <='0'; wait for 200 ns;
   EN <='1'; wait for 200 ns;
end process P_EN;
```

A Aufgabe 9.5

Die DNF des XNOR-Gatters ergibt sich aus der Inversion der DNF des XOR-Gatters:

$F2 = \neg((A \wedge \overline{B}) \vee (\overline{A} \wedge B))$

Nun wird das Adsorptionsgesetz $(A \wedge B) = A \wedge (\overline{A} \vee B)$ auf beide Terme angewendet:

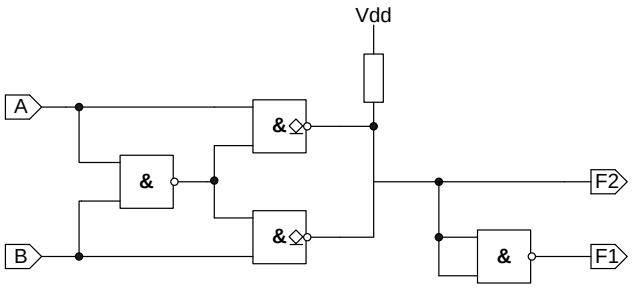
$F2 = \neg ((A \wedge (\overline{A} \vee \overline{B})) \vee (B \wedge (\overline{A} \vee \overline{B})))$

Die dreifache Anwendung des de Morgan'schen Gesetzes liefert:

$F2 = \neg (A \wedge (\overline{A \wedge B})) \wedge \neg (B \wedge (\overline{A \wedge B}))$

In der Schaltung wird der Term $\overline{A \wedge B}$ gemeinsam verwendet. Die beiden NAND-Verknüpfungen mit den Eingangssignalen A bzw. B werden durch Gatter mit Open-Drain-Ausgang realisiert, sodass

das äußere UND-Gatter durch ein Wired-AND gebildet wird. Der XOR-Ausgang wird durch ein weiteres NAND mit Push-Pull-Ausgang realisiert, welches als Inverter beschaltet ist.



10 Datenpfadkomponenten

A Aufgabe 10.1

a) Steuersignale kontrollieren die Datenpfadkomponenten, Statussignale sind (neben den Datenausgangssignalen) die Ausangssignale von Datenpfadkomponenten.

b) Vgl. Merksatz in Kap. 10.4

c) Die Modellierung eines Prioritätsencoders innerhalb eines Prozesses erfordert eine if-Anweisung mit ggf. weiteren elsif-Zweigen (vgl. Listing 10.3). Die gleiche Funktion hat auch eine bedingte nebenläufige Signalzuweisung.

d) Vgl. Merksatz zu Bild 10.4 in Kap 10.3. Ein Demultiplexer wird in VHDL mit einer case-Anweisung modelliert.

e) Vgl. Merksatz in Kap 10.6.

f) Zum Ripple-Carry Addierer s. Merksatz am Ende von Kap. 10.8.2. Zum Carry-Lookahead Addierer s. den 1. Absatz von Kap. 10.8.3 bzw. Bild 10.15. Vorteil der Carry-Lookahead Struktur ist die von der Bitbreite weitgehend unabhängige Ausführungszeit für die Addition. Nachteil ist der mit zunehmender Bitbreite schnell zunehmende Hardwareaufwand des Carry-Lookahead Generators.

g) Modellierung mit nebenläufigen Signalzuweisungen, Prozessen oder Komponenten (Strukturmodell).

h) Den Datentyp unsigned bzw. signed im Zusammenhang mit der Bibliothek ieee.numeric_std.

i) Das Ergebnis ist 6 Bit breit.

j) Es wird der => Operator verwendet, links davon stehen die lokalen Signale der Komponentendeklaration und rechts davon die aktuellen Signale der instanziierenden, übergeordneten entity (vgl. letzter Absatz in Kap. 10.7).

A Aufgabe 10.2

Die Testbench sieht acht Tests vor. In der vorgestellten Testbench wird der Wert der Quellen als konstant angenommen: QUELLEN = 0xA5. Da die Anzahl der Quellen doppelt so groß ist wie die Anzahl der Senken, wird jeder Senke zu zwei unterschiedlichen Zeitpunkten ein Quellwert zugewiesen:

```
-- Multiplexer Komponente
entity MUX is
   port(QUELLEN: in bit_vector(7 downto 0);
         SEL: in bit_vector(2 downto 0);
         ONE_WIRE: out bit);
end MUX;

architecture VERHALTEN of MUX is
begin
P1: process(QUELLEN, SEL)
begin
   case SEL is
      when "000" => ONE_WIRE <= QUELLEN(0) after 5 ns;
      when "001" => ONE_WIRE <= QUELLEN(1) after 5 ns;
      when "010" => ONE_WIRE <= QUELLEN(2) after 5 ns;
      when "011" => ONE_WIRE <= QUELLEN(3) after 5 ns;
      when "100" => ONE_WIRE <= QUELLEN(4) after 5 ns;
      when "101" => ONE_WIRE <= QUELLEN(5) after 5 ns;
      when "110" => ONE_WIRE <= QUELLEN(6) after 5 ns;
      when "111" => ONE_WIRE <= QUELLEN(7) after 5 ns;
   end case;
end process P1;
end VERHALTEN;
-----------------------------------------------------------
-- Demultiplexer Komponente
entity DMUX is
   port(
         ONE_WIRE: in bit;
         SEL: in bit_vector(1 downto 0);
         SENKEN: out bit_vector(3 downto 0)
         );
end DMUX;

architecture VERHALTEN of DMUX is
begin
P1: process(ONE_WIRE, SEL)
begin
   SENKEN <= "0000"; -- Default: alles 0
   case SEL is       -- ueberschreibe genau ein Bit
      when "00" => SENKEN(0) <= ONE_WIRE after 5 ns;
      when "01" => SENKEN(1) <= ONE_WIRE after 5 ns;
      when "10" => SENKEN(2) <= ONE_WIRE after 5 ns;
      when "11" => SENKEN(3) <= ONE_WIRE after 5 ns;
   end case;
end process P1;
end VERHALTEN;

-----------------------------------------------------------
-- Multiplexer/Demultiplxer System
entity AUFGABE_10_2 is
   port( QUELLEN : in bit_vector(7 downto 0);
```

```
         QUELLEN_SEL: in bit_vector(2 downto 0);
         SENKEN_SEL: in bit_vector(1 downto 0);

         SENKEN : out bit_vector(3 downto 0));
end AUFGABE_10_2 ;

architecture STRUKTUR of AUFGABE_10_2 is
component MUX is
   port( QUELLEN: in bit_vector(7 downto 0);
         SEL: in bit_vector(2 downto 0);
         ONE_WIRE: out bit);
end component;

component DMUX is
   port(
         ONE_WIRE: in bit;
         SEL: in bit_vector(1 downto 0);
         SENKEN: out bit_vector(3 downto 0)
         );
end component;
signal WIRE: bit;
begin
U1: MUX port map(QUELLEN, QUELLEN_SEL, WIRE);

U2: DMUX port map(WIRE, SENKEN_SEL, SENKEN);
end STRUKTUR;

-------------------------------------------------------
-- Testbench zum Multiplexer/Demultiplexer System

entity AUFGABE_10_2_TB is
end AUFGABE_10_2_TB;

architecture TESTBENCH of AUFGABE_10_2_TB is
signal QUELLEN: bit_vector(7 downto 0);
signal SENKEN: bit_vector(3 downto 0);
signal QUELLEN_SEL: bit_vector(2 downto 0);
signal SENKEN_SEL: bit_vector(1 downto 0);
signal TEST_NR: integer range 0 to 7;

component AUFGABE_10_2 is
   port( QUELLEN : in bit_vector(7 downto 0);
         QUELLEN_SEL: in bit_vector(2 downto 0);
         SENKEN_SEL: in bit_vector(1 downto 0);

         SENKEN : out bit_vector(3 downto 0));
end component;

begin

DUT: AUFGABE_10_2 port map(QUELLEN, QUELLEN_SEL, SENKEN_SEL, SENKEN);

-- konstante Quellensignale
QUELLEN <= "10100101";

-- Auswahl der Quellen und Senken
SEL_STIMULI: process
begin
   TEST_NR <= 0; QUELLEN_SEL<= "000"; SENKEN_SEL <= "00"; wait for 100 ns;
   TEST_NR <= 1; QUELLEN_SEL<= "001"; SENKEN_SEL <= "01"; wait for 100 ns;
```

```
   TEST_NR <= 2; QUELLEN_SEL<= "010"; SENKEN_SEL <= "10"; wait for 100 ns;
   TEST_NR <= 3; QUELLEN_SEL<= "011"; SENKEN_SEL <= "11"; wait for 100 ns;
   TEST_NR <= 4; QUELLEN_SEL<= "100"; SENKEN_SEL <= "00"; wait for 100 ns;
   TEST_NR <= 5; QUELLEN_SEL<= "101"; SENKEN_SEL <= "01"; wait for 100 ns;
   TEST_NR <= 6; QUELLEN_SEL<= "110"; SENKEN_SEL <= "10"; wait for 100 ns;
   TEST_NR <= 7; QUELLEN_SEL<= "111"; SENKEN_SEL <= "11"; wait for 100 ns;
end process SEL_STIMULI;

-- Response Monitor
MONITOR: process
begin
   wait for 50 ns; -- warte stabile Signale ab
   for I in 0 to 7 loop
      case TEST_NR is
         when 0 => assert SENKEN = "0001" report "Error: test 0";
         when 1 => assert SENKEN = "0000" report "Error: test 1";
         when 2 => assert SENKEN = "0100" report "Error: test 2";
         when 3 => assert SENKEN = "0000" report "Error: test 3";
         when 4 => assert SENKEN = "0000" report "Error: test 4";
         when 5 => assert SENKEN = "0010" report "Error: test 5";
         when 6 => assert SENKEN = "0000" report "Error: test 6";
         when 7 => assert SENKEN = "1000" report "Error: test 7";
      end case;
      wait for 100 ns;
   end loop;
end process MONITOR;
end TESTBENCH;
```

A Aufgabe 10.3

Das VHDL-Modell des 8-Bit-Addierers ist angelehnt an Listing 10.9:

```
entity VOLLADD is
       port (A, B, CI :in bit; SUM, CO :out bit);
end VOLLADD;

architecture VERHALTEN of VOLLADD is
begin
   SUM <= A xor B xor CI after 5 ns;
   CO <= (A and B) or (CI and (A xor B)) after 5 ns;
end VERHALTEN;

entity AUFGABE_10_3 is
       port (A, B :in bit_vector(7 downto 0);
             CI :in bit;
             SUM:out bit_vector(8 downto 0));
end AUFGABE_10_3 ;
architecture STRUKTUR of AUFGABE_10_3 is
component VOLLADD
       port (A, B, CI :in bit; SUM, CO :out bit);
end component;
signal CARRY: bit_vector(8 downto 0);

begin
CARRY(0) <= CI;

CARRY_CHAIN: for I in 0 to 7 generate
  VA: VOLLADD
   port map(A(I), B(I), CARRY(I), SUM(I), CARRY(I+1));
```

```
end generate CARRY_CHAIN;

SUM(8) <= CARRY(8);
end STRUKTUR;
```

Es sind verschiedene worst- und best-case-Szenarien denkbar: Im worst-case ergibt sich das endgültige Summationsergebnis nach Durchlaufen der kompletten Carry-Laufzeitkette. Z.B.:

- ausgehend von A = 0xFF, B = 0x00 und CI = 0 nach Wechsel auf CI=1
- ausgehend von A = 0xFF, B = 0x00 und CI = 1 nach Wechsel auf CI=0
-

Im best-case erhält man das endgültige Ergebnis nach nur einer Laufzeit durch den/die Volladdierer. Z.B:

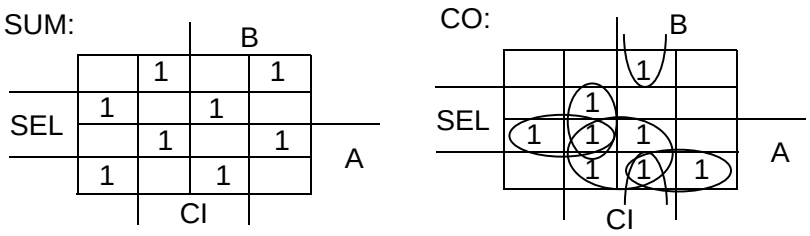
- ausgehend von A = 0x00, B = 0x00 und CI = 0 nach Wechsel auf A = 0xFF
- ausgehend von A = 0xFF, B = 0x00 und CI = 0 nach Wechsel auf A = 0x00

ISBN 978-3-486-72765-4. Oldenbourg Wissenschaftsverlag, München 2013.

A Aufgabe 10.4

In der Wahrheitstabelle zur Berechnung der Differenz A – B – CI ist zu beachten, dass das Eingangs- bzw. Ausgangs-Carry bei der Subtraktion (SEL = 1) L-aktiv ist!

SEL	CI	A	B	Berechnung	SUM	CO
0	0	0	0	0 + 0 + 0 = 0	0	0
0	0	0	1	0 + 0 + 1 = 1	1	0
0	0	1	0	0 + 1 + 0 = 1	1	0
0	0	1	1	0 + 1 + 1 = 2	0	1
0	1	0	0	1 + 0 + 0 = 1	1	0
0	1	0	1	1 + 0 + 1 = 2	0	1
0	1	1	0	1 + 1 + 0 = 2	0	1
0	1	1	1	1 + 1 + 1 = 3	1	1
1	0	0	0	0 – 0 – 1 = –1	1	0
1	0	0	1	0 – 1 – 1 = –2	0	0
1	0	1	0	1 – 0 – 1 = 0	0	1
1	0	1	1	1 – 1 – 1 = –1	1	0
1	1	0	0	0 – 0 – 0 = 0	0	1
1	1	0	1	0 – 1 – 0 = –1	1	0
1	1	1	0	1 – 0 – 0 = +1	1	1
1	1	1	1	1 – 1 – 0 = 0	0	1



SUM = SEL ↮ CI ↮ B ↮ A

CO = (CI ∧ A) ∨ ($\overline{\text{SEL}}$ ∧ A ∧ B) ∨ ($\overline{\text{SEL}}$ ∧ CI ∧ B) ∨ (SEL ∧ CI ∧ $\overline{\text{B}}$) ∨ (SEL ∧ A ∧ $\overline{\text{B}}$)

```
-- 1-Bit Addierer / Subtrahierer
entity AS_COMP is
       port( A, B, CIN, SEL: in bit;
             S, CO: out bit);
end AS_COMP;
```

```
architecture ARCH1 of AS_COMP is
begin
        S <= SEL xor CIN xor B xor A after 10 ns;
        CO <= (CIN and A) or
              (not SEL and B and A) or
              (not SEL and CIN and B) or
              (SEL and CIN and not B) or
              (SEL and not B and A) after 10 ns;
end ARCH1;
-----------------------------------------------
-- 4-Bit Ripple-Carry Addierer / Subtrahierer
entity AUFGABE_10_4 is
        port( A, B: in bit_vector(3 downto 0);
              CIN, SEL: in bit;
              S: out bit_vector(3 downto 0);
              COUT: out bit);
end AUFGABE_10_4;

architecture STRUKTUR of AUFGABE_10_4 is
component AS_COMP
        port( A, B, CIN, SEL: in bit;
              S, CO: out bit);
end component;

signal CARRY: bit_vector(4 downto 0);
begin
        CARRY(0) <= CIN;
VA:     for I in 0 to 3 generate
    AS:       AS_COMP
              port map (A(I), B(I), CARRY(I), SEL, S(I), CARRY(I+1));
        end generate VA;
        COUT <= CARRY(4);
end STRUKTUR;
```

Mit der nachfolgenden Macro-Datei für den VHDL-Simulator Model-Sim lässt sich das korrekte Verhalten des Addierers / Subtrahierers verifizieren:

```
restart
# Welche Fenster oeffnen?
view signals
view wave

# Default Radix, wichtig fuer Hex-Eingaben
radix hex

# Anzeigen aller Signale im Waves Fenster
# add wave sim:/<modulname>/*
add wave sim:/aufgabe_10_4/*

# kombinatorisches Verhalten:
force cin 0
force sel 0
# Additionen:
# 7+6 =D
force a 7
force b 6
run 50ns
```

```
# f+6=0x15
force a f
run 50ns

# 0+0=0
force a 0
force b 0
run 50ns

# f+0+cin=10 (Worst Case Ripple)
force a f
force cin 1
run 50ns

#Subtraktionen ohne Eingangsuebertrag:
force sel 1
# 7-6 =1
force a 7
force b 6
run 50ns

# f-6=9
force a f
run 50ns

# e-f-cin=-2 (=0xE)
force a e
force b f
force cin 0
run 50ns
```

A Aufgabe 10.5

Die Umsetzung der Zahl 5.75 ins 3Q2-Format liefert: 0101.11

Die Umsetzung der Zahl 1.9375 ins 1Q4-Format liefert: 01.1111

Die dezimale Multiplikation der Operanden liefert P = 11.140625.

```
101.11 * 1.1111
          11111
         11111
        11111
       00000
      11111____
     111011010_ (Übertrag in die nächst höhere Stufe)
     001101000_ (weitere Überträge in höhere Stufen)
_________________
     1011001001
```

Der Dualpunkt muss nach der sechsten Stelle von rechts gesetzt werden: P = 1011.001001.

Die Umsetzung dieser Binärzahl in das Dezimalsystem liefert das korrekte Ergebnis:

P = 8 + 2 + 1 + 0.125 + 0.015625 = 11.140625.

Damit bei der Addition kein Übertrag verloren geht und außerdem die volle numerische Auflösung erhalten bleibt, muss das Summationsergebnis neben dem Vorzeichenbit vier Vorkomma- und vier Nachkommastellen besitzen, es ist also neun Bit breit. Der Operand A ist im VHDL-Code also um ein Bit und der Operand B um drei Bit vorzeichengerecht zu ergänzen. Ferner müssen beim A-Operanden zwei Nullen angehängt werden, die die Nachkommastellen repräsentieren:

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

entity AUFGABE_10_5 is
end AUFGABE_10_5;

architecture ARCH of AUFGABE_10_5 is
signal A,B: std_logic_vector(5 downto 0);
signal ATEMP, BTEMP, SUM: std_logic_vector(8 downto 0);
begin
-- Zahlenbeispiel aus der Aufgabe:
A <= "010111"; B <= "011111";

-- Vor- und Nachkommaerweiterung:
ATEMP <= A(5) & A & "00";
BTEMP <= B(5) & B(5) & B(5) & B;
SUM <= ATEMP + BTEMP;
end ARCH;
```

Das Ergebnis dieser Berechnung ergibt ATEMP = 0x05C, BTEMP = 0x01F und SUM = 0x07B. Diese Summe entspricht im vorzeichenbehafteten 4Q4-Format dem korrekten Dezimalergebnis 7.6875.

A Aufgabe 10.6

Multiplikator					1	0	1	1	-8 + 2 + 1 = -5
Multiplikand					1	1	0	1	-8 + 4 + 1 = -3
			1	*1*	*1*	*1*	1	1	0	1	Multiplikand mit vorzeichengerechter Ergänzung auf 8 Bit
			1	*1*	*1*	1	1	0	1		
			0	*0*	0	0	0	0			
			0	0	0	1	1				2er-Komplement des Multiplikanden
		1	0	1	0	1	0	0	0	-	1. Carry
		1		1							2. Carry
~~1~~	~~0~~	0	0	0	0	1	1	1	1		

8 + 4 + 2 + 1 = 15

A Aufgabe 10.7

```
-- 4-Bit ALU
--------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ALU4 is
        port( A, B: in std_logic_vector(3 downto 0);      --4-Bit Operanden
              OPCODE: in bit_vector(1 downto 0);          --2-Bit OPCODE
              RESULT: out std_logic_vector(3 downto 0);   --4-Bit Ergebnis
              CFLAG, ZFLAG: out bit);                     --Carry/Zero Flag
end ALU4;

architecture ALGO of ALU4 is
begin
P1:     process( A,B,OPCODE )
-- 5-Bit Temporaere Variable
variable TEMP_RESULT, TEMPA, TEMPB: std_logic_vector(4 downto 0);
begin
        CFLAG <= '0';                                     -- Vorbelegung mit 0
        TEMPA := '0'& A;                                  -- MSB anfuegen
        TEMPB := '0'& B;                                  -- MSB anfuegen
        case OPCODE is
              when "00" => TEMP_RESULT := TEMPA + TEMPB;
                           if TEMP_RESULT(4)='1' then
                             CFLAG <= '1';
                           end if;
              when "01" => TEMP_RESULT := TEMPA - TEMPB;
                           if TEMP_RESULT(4)='1' then
                             CFLAG <= '1';
                           end if;
              when "10" =>      TEMP_RESULT := TEMPA or TEMPB;
              when "11" =>      TEMP_RESULT := TEMPA and TEMPB;
        end case;
        if TEMP_RESULT(3 downto 0) = "0000" then
              ZFLAG <= '1';
        else
              ZFLAG <= '0';
        end if;
        RESULT <= TEMP_RESULT(3 downto 0);
end process P1;
end ALGO;
```

11 Latches und Flipflops in synchronen Schaltungen

A Aufgabe 11.1

Der nachfolgende VHDL-Code enthält ein Modell des Scan-Flipflops, eine aus vier Elementen bestehende Scan-Flipflop-Kette sowie eine Testbench. In dieser wird zunächst der Normalbetrieb als Register und anschließend das Laden der Binärzahläquivalente 1 bis 5 überprüft. Die serielle Verbindung zwischen den einzelnen Flipflops wird durch den Signalvektor LINK gebildet, der mit Ein- und Ausgangssignalen aus fünf Elementen besteht.

```
-- Scan-FF
entity SCAN_FF is
   port( CLK : in bit;
         D, TE, TI: in bit;
         Q : out bit);
end SCAN_FF;

architecture VERHALTEN of SCAN_FF is
begin
P1: process(CLK)
begin
   if CLK='1' and CLK'event then
      if TE ='0' then
         Q <= D after 5 ns;
      else
         Q <= TI after 5 ns;
  end if;
   end if;
end process P1;
end VERHALTEN;
-------------------------------------------------------
-- Scan-Kette
entity AUFGABE_11_1 is
   port( CLK: in bit;
         D : in bit_vector(3 downto 0);
         TE, TI: in bit;
         Q: out bit_vector(3 downto 0);
         Y1: out bit);
end AUFGABE_11_1;

architecture STRUKTUR of AUFGABE_11_1 is
component SCAN_FF is
   port( CLK : in bit;
         D, TE, TI: in bit;
         Q : out bit);
end component;

signal LINK: bit_vector(4 downto 0);

begin
LINK(0) <= TI;
CHAIN: for I in 0 to 3 generate
  FF: SCAN_FF port map(CLK, D(I), TE, LINK(I), LINK(I+1));
```

Aufgabenlösungen zu: Jürgen Reichardt, Lehrbuch Digitaltechnik 3. Auflage
ISBN 978-3-486-72765-4. Oldenbourg Wissenschaftsverlag, München 2013.

```
end generate CHAIN;
Q <= LINK(4 downto 1);
Y1 <= LINK(4);
end STRUKTUR;
-------------------------------------------------------
-- Testbench zur Scan-Kette
entity AUFGABE_11_1_tb is
end AUFGABE_11_1_tb;

architecture TESTBENCH of AUFGABE_11_1_tb is
signal CLK, TI, TE, Y1 : bit;
signal D, Q: bit_vector(3 downto 0);
component AUFGABE_11_1 is
   port( CLK: in bit;
         D : in bit_vector(3 downto 0);
         TE, TI: in bit;
         Q: out bit_vector(3 downto 0);
         Y1: out bit);
end component;

begin
DUT: AUFGABE_11_1 port map (CLK, D, TE, TI, Q, Y1);

CLKGEN: process -- 10-MHZ Taktgenerator
begin
   CLK <= '1'; wait for 50 ns;
   CLK <= '0'; wait for 50 ns;
end process CLKGEN;

D <= "1010"; -- konstanter Dateneingang

STIMGEN: process
begin
   wait for 50 ns; -- warte auf fallende Flanke
   TI <= '0'; TE <='0'; wait for 100 ns; -- Betriebsmodus

   TI <= '0'; TE <='1'; wait for 100 ns; -- Scan-Modus lade 0001
   TI <= '0'; TE <='1'; wait for 100 ns;
   TI <= '0'; TE <='1'; wait for 100 ns;
   TI <= '1'; TE <='1'; wait for 100 ns;

   TI <= '0'; TE <='1'; wait for 100 ns; -- Scan-Modus lade 0002
   TI <= '0'; TE <='1'; wait for 100 ns;
   TI <= '1'; TE <='1'; wait for 100 ns;
   TI <= '0'; TE <='1'; wait for 100 ns;

   TI <= '0'; TE <='1'; wait for 100 ns; -- Scan-Modus lade 0003
   TI <= '0'; TE <='1'; wait for 100 ns;
   TI <= '1'; TE <='1'; wait for 100 ns;
   TI <= '1'; TE <='1'; wait for 100 ns;

   TI <= '0'; TE <='1'; wait for 100 ns; -- Scan-Modus lade 0004
   TI <= '1'; TE <='1'; wait for 100 ns;
   TI <= '0'; TE <='1'; wait for 100 ns;
   TI <= '0'; TE <='1'; wait for 100 ns;

   TI <= '0'; TE <='1'; wait for 100 ns; -- Scan-Modus lade 0005
   TI <= '1'; TE <='1'; wait for 100 ns;
   TI <= '0'; TE <='1'; wait for 100 ns;
   TI <= '1'; TE <='1'; wait for 100 ns;
```

```
end process STIMGEN;

MONITOR: process
begin
   wait for 125 ns; -- t = 125 ns:
   assert Q = "1010" report "Fehler im Betriebsmodus";

   wait for 400 ns; -- t = 525 ns:
   assert Q = "0001" report "Fehler im Scanmodus 1";

   wait for 400 ns; -- t = 925 ns:
   assert Q = "0010" report "Fehler im Scanmodus 2";

   wait for 400 ns; -- t = 1325 ns:
   assert Q = "0011" report "Fehler im Scanmodus 3";

   wait for 400 ns; -- t = 1725 ns:
   assert Q = "0100" report "Fehler im Scanmodus 4";

   wait for 400 ns; -- t = 2125 ns:
   assert Q = "0101" report "Fehler im Scanmodus 5";
end process MONITOR;
end TESTBENCH;
```

A Aufgabe 11.2

Der nachfolgende VHDL-Code enthält die beiden Architekturen DATENFLUSS und VERHALTEN. In der Testbench AUFGABE_11_2_TB werden beide Architekturen jeweils einmal instanziiert (DUT_0 und DUT_1). Aus diesem Grunde muss die Testbench zwei Konfigurationsanweisungen enthalten, die spezifizieren, welche der Architekturen jeweils verwendet werden soll: **for** <component_name> **use entity** <entity_name(<architecture_name>).

Bei der Komponenteninstanziierung ist spezifiziert, dass die Datenfluss-Architektur das Signal Q1 und die Verhaltens-Architektur das Signal Q2 beschreibt:

```
-- Zwei JK-FF Architekturen
entity AUFGABE_11_2 is
   port( CLK, RESET, J, K : in bit;
         Q : out bit);
end AUFGABE_11_2;

architecture DATENFLUSS of AUFGABE_11_2 is
signal QTEMP: bit;
begin
P1: process(CLK, RESET)
begin
   if RESET = '1' then
      QTEMP <= '0' after 5 ns;
   elsif CLK='1' and CLK'event then
      QTEMP <= (J and not QTEMP) or (not K and QTEMP) after 5 ns;
   end if;
end process P1;
Q <= QTEMP;
end DATENFLUSS;
```

```
architecture VERHALTEN of AUFGABE_11_2 is
signal QTEMP: bit;
begin
P1: process(CLK, RESET)
begin
   if RESET = '1' then
      QTEMP <= '0' after 5 ns;
   elsif CLK='1' and CLK'event then
--      QTEMP <= (J and not QTEMP) or (not K and QTEMP) after 5 ns;
      if J='0' and K='1' then QTEMP <= '0' after 5 ns;
      elsif J='1' and K='0' then QTEMP <= '1' after 5 ns;
      elsif J='1' and K='1' then QTEMP <= not QTEMP after 5 ns;
      end if;
   end if;
end process P1;
Q <= QTEMP;
end VERHALTEN;
------------------------------------------------------------
-- Testbench zum JKFF
entity AUFGABE_11_2_tb is
end AUFGABE_11_2_tb;

architecture TESTBENCH of AUFGABE_11_2_tb is
signal CLK, RESET, J, K, Q1, Q2 : bit;
component AUFGABE_11_2 is
   port( CLK, RESET, J, K : in bit;
         Q : out bit);
end component;

-- Komponentenkonfigurationen:
for DUT_0: AUFGABE_11_2 use entity WORK.AUFGABE_11_2(DATENFLUSS);
for DUT_1: AUFGABE_11_2 use entity WORK.AUFGABE_11_2(VERHALTEN);

begin
DUT_0: AUFGABE_11_2 port map (CLK, RESET, J, K, Q1);
DUT_1: AUFGABE_11_2 port map (CLK, RESET, J, K, Q2);

CLKGEN: process -- 10-MHZ Taktgenerator
begin
   CLK <= '1'; wait for 50 ns;
   CLK <= '0'; wait for 50 ns;
end process CLKGEN;

RESET <= '0', '1' after 10 ns, '0' after 33 ns;

STIMGEN: process
begin
   wait for 50 ns; -- warte auf fallende Flanke
   J <= '1'; K <= '0'; wait for 100 ns; -- Setzen
   J <= '0'; K <= '0'; wait for 100 ns; -- Speichern
   J <= '0'; K <= '1'; wait for 100 ns; -- Loeschen
   J <= '1'; K <= '1'; wait for 100 ns; -- Toggeln
end process STIMGEN;

MONITOR: process
begin
   wait for 125 ns; -- nach der 1. Flanke:
   assert Q1 = '1' report "DUT_0: Fehler beim Setzen";
   assert Q2 = '1' report "DUT_1: Fehler beim Setzen";
```

```
    wait for 100 ns; -- nach der 2. Flanke:
    assert Q1 = '1' report "DUT_0: Fehler beim Speichern";
    assert Q2 = '1' report "DUT_1: Fehler beim Speichern";

    wait for 100 ns; -- nach der 3. Flanke:
    assert Q1 = '0' report "DUT_0: Fehler beim Loeschen";
    assert Q2 = '0' report "DUT_1: Fehler beim Loeschen";

    wait for 100 ns; -- nach der 4. Flanke:
    assert Q1 = '1' report "DUT_0: Fehler beim Toggeln";
    assert Q2 = '1' report "DUT_1: Fehler beim Toggeln";
end process MONITOR;
end TESTBENCH;
```

A Aufgabe 11.3

a) Latches sind taktzustandsgesteuert, Flipflops hingegen taktflankengesteuert.

b) Durch Kreuzkopplung zweier NOR- oder NAND-Gatter (s. Bilder 11.1 bzw. 11.5).

c) Das aus NOR-Gattern aufgebaute RS-Latch gerät ins Schwingen (vgl. Abschnitt **6.** In Bild 11.2)

d) s. Gl. (11.1)

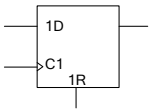
e) Die Synthesetabelle beschreibt, wie die Schaltung anzusteuern ist, um aus einem bestimmten Zustand heraus eine bestimmte Funktion zu erreichen. Die Arbeitstabelle beschreibt hingegen, wie die Schaltung in einem bestimmten Zustand reagiert, wenn sie mit den verschiedenen Eingangssignalkombinationen angesteuert wird.

f) s. Merksatz in Kap. 11.3 unterhalb von Gl. (11.2)

g) s. Merksatz zu Listing 11.5.

h) s. Merksatz zu Listing 11.5.

i)



j) Der asynchrone Reset muss vor der Taktflankenabfrage abgefragt werden, der Freigabeeingang erfordert eine zusätzliche if-Bedingung innerhalb des taktsynchronen Rahmens:

```
P1: process(CLK, RESET)
begin
  if RESET = '1' then Q <= '0';        -- async. Reset
       elsif (CLK='1' and CLK'event) then   -- steigende Flanke
       if ENABLE = '1' then Q <= D;      -- Abfrage des Freigabesignals
       end if;
       end if;
end process P1;
```

k) Vgl. Merksatz zu Bild 11.12

l) Zur Vermeidung von Setup- oder Hold-Time Fehlern wird empfohlen, die Tetbenchsignale während der fallenden Flanke zu ändern.

m) Vgl. die Erläuterungen vor Bild 11.12

n) Die Flipflops eines synchronen Systems erhalten ihr Taktsignal nicht alle zum gleichen Zeitpunkt. Die Verzögerung zwischen der langsamsten und der schnellsten Taktansteuerung wird als Clock-Skew (Taktverzögerung) bezeichnet.

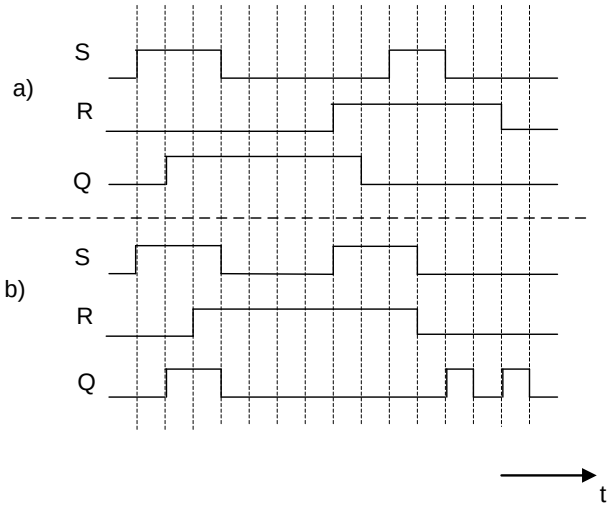
o) Die Taktreserve ist die Differenz zwischen der Taktperiodendauer und der für das Durchlaufen des kritischen Pfades erforderlichen Zeit.

p) Der kritischste Pfad ist der längste in einem synchronen System vorhandene Laufzeitpfad zwischen zwei Flanken (vgl. Bild 11.24).

q) Ein Zweispeicher-Flipflop (Master-Slave Flipflop) hat die Eigenschaft, dass es Eingangsignale bei steigender Flanke übernimmt aber bei fallender Flanke ausgibt.

A Aufgabe 11.4

In der Teilaufgabe a) wird das R-S-Latch nach der irregulären Ansteuerung zurückgesetzt, in der Teilaufgabe b) wird nach der irregulären Ansteuerung versucht, zu Speichern. Dies führt zu einem Schwingen des Ausgangssignals.



A Aufgabe 11.5

Als Entprellschaltung kann ein RS-Latch mit L-aktiven Eingängen verwendet werden, welches aus rückgekoppelten NAND-Gattern aufgebaut ist: Der Eingang A wird an $\overline{R}$ und der Eingang B an $\overline{S}$ angeschlossen. Zum Zeitpunkt t = t1 wird mit A = 1 und B = 1 zunächst *Speichern* gefordert.

Das prellende Signal B = 0 führt zum *Setzen* des Flipflops. Im weiteren Verlauf des Prellens wird dieser Zustand nur gespeichert. Das Ausgangssignal ist also nach dem *Setzen* sofort stabil. Das stabile Signal B = 0 hält das RS-Latch im Zustand *Setzen*. Zum Zeitpunkt t = t2 wird zunächst B = 1. Das entspricht dem *Speichern*. Danach wird A = 0 und führt zum *Löschen* des RS-Latches. Im weiteren Verlauf des Prellens wird zwischen *Speichern* und *Löschen* gewechselt, was den Signalausgang des Latches aber nicht ändert. Die Simulation der Testbench zum VHDL-Code demonstriert das korrekte Verhalten der Schaltung:

```
-- Entprellschaltung mit RS-Latch
entity AUFGABE_11_5 is
   port( N_R, N_S : in bit;
         N_Q, Q : out bit);
end AUFGABE_11_5;

architecture DATENFLUSS of AUFGABE_11_5 is
signal Q_TEMP, NQ_TEMP: bit;
begin
   Q_TEMP <= N_S nand NQ_TEMP after 10 ns;
   NQ_TEMP <= N_R nand Q_TEMP after 10 ns;
   Q <= Q_TEMP;
   N_Q <= NQ_TEMP;
end DATENFLUSS;
-------------------------------------------------------
-- Testbench zur Entprellschaltung
entity AUFGABE_11_5_tb is
end AUFGABE_11_5_tb;

architecture TESTBENCH of AUFGABE_11_5_tb is
signal A, B, A1, B1 : bit;
component AUFGABE_11_5 is
   port( N_R, N_S : in bit;
         N_Q, Q : out bit);
end component;

begin
DUT: AUFGABE_11_5 port map (A, B, A1, B1);
STIMGEN: process
begin
   A <= '0'; B <= '1'; wait for 100 ns; -- Schalterzustand AUS
   A <= '1'; B <= '1'; wait for 100 ns; -- Schalter prellt beim Einschalten
   A <= '1'; B <= '0'; wait for 100 ns; -- Schalter prellt beim Einschalten
   A <= '1'; B <= '1'; wait for 100 ns; -- Schalter prellt beim Einschalten
   A <= '1'; B <= '0'; wait for 100 ns; -- Schalter prellt beim Einschalten
   A <= '1'; B <= '1'; wait for 100 ns; -- Schalter prellt beim Einschalten
   A <= '1'; B <= '0'; wait for 100 ns; -- Schalter prellt beim Einschalten
   A <= '1'; B <= '1'; wait for 100 ns; -- Schalter prellt beim Einschalten
   A <= '1'; B <= '0'; wait for 300 ns; -- Schalterzustand EIN
   A <= '1'; B <= '1'; wait for 100 ns; -- Schalter prellt beim Ausschalten
   A <= '0'; B <= '1'; wait for 100 ns; -- Schalter prellt beim Ausschalten
   A <= '1'; B <= '1'; wait for 100 ns; -- Schalter prellt beim Ausschalten
   A <= '0'; B <= '1'; wait for 100 ns; -- Schalter prellt beim Ausschalten
   A <= '1'; B <= '1'; wait for 100 ns; -- Schalter prellt beim Ausschalten
   A <= '0'; B <= '1'; wait for 100 ns; -- Schalter prellt beim Ausschalten
   A <= '1'; B <= '1'; wait for 100 ns; -- Schalter prellt beim Ausschalten
   A <= '0'; B <= '1'; wait for 300 ns; -- Schalterzustand AUS
end process STIMGEN;
end TESTBENCH;
```

A Aufgabe 11.6

```
1   entity TEST is
2     port(  CLK, RESET: in bit;
3            A, B, SEL: in bit;
4            ERG1, ERG2: out bit);
5   end TEST;
6  ----------------------------------------
7   architecture FEHLER_A of TEST is
8   variable TEMP: bit;
9   begin
10  P1:       process(CLK, RESET)
11    begin
12            if RESET='1' then
13                   TEMP :='0';
14            elsif CLK'event and CLK='0' then
15                   TEMP:= A;
16            end if;
17            ERG1 <= '0' when SEL='0' else TEMP;
18    end process P1;
19
20  P2:       process(TEMP, A, B)
21    begin
22            if SEL ='0' then
23                   ERG2 <= TEMP;
24            else
25                   ERG2 <= A and B;
26            end if;
27    end process;
28  end FEHLER_A;
29 ------------------------------------------------------------------
30  architecture FEHLER_B of TEST is
31  begin
32  P1:       process(CLK, RESET)
33    variable TEMP: bit;
34    begin
35            if RESET='1' then
36                   TEMP :='0';
37            elsif CLK'event and CLK='0' then
38                   TEMP:= A;
39            end if;
40    end process P1;
41
42  P2:       process(TEMP, A, B)
43    begin
44            if SEL ='0' then
45                   ERG2 <= TEMP;
46            else
47                   ERG2 <= A and B;
48            end if;
49    end process;
50    if SEL ='0' then
51            ERG1 <= '0';
52    else
53            ERG1 <= TEMP;
54    end if;
55 end FEHLER_B;
```

Der VHDL-Code enthält die folgenden Fehler:

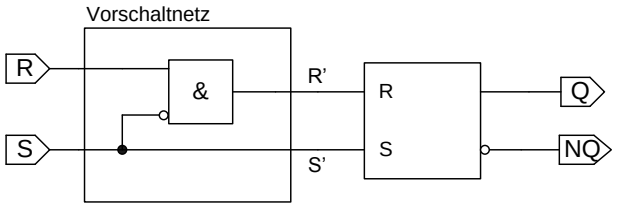
- Zeile 8: Deklaration einer Variablen außerhalb eines Prozesses.
- Zeile 17: Verbotene nebenläufige Signalzuweisung innerhalb eines Prozesses.
- Zeile 20: Die Variable TEMP befindet sich in der Sensitivityliste des Prozess P2, hier dürfen nur Signale stehen.
- Zeile 42: TEMP ist in P1 der Architektur FEHLER_B als Variable deklariert und daher außerhalb dieses Prozesses ungültig. Daher darf sich TEMP nicht in der Sensitivityliste von P2 befinden.
- Zeile 45: TEMP ist nicht deklariert (s.o.).
- Zeile 50: Es wird eine sequenzielle Anweisung (`if`) außerhalb eines Prozesses verwendet.
- Zeile 53: Verwendung eines nicht deklarierten Signals.
- Zeile 53: Zuweisung eines zweiten Treibersignals an das nicht aufgelöste Signal ERG1. Mehrere Zuweisungen an das gleiche Signal sind nur bei aufgelösten Datentypen (z.B. `std_logic`) erlaubt.

A Aufgabe 11.7

Das Vorschaltnetz für ein aus NOR-Gattern aufgebautes RS-Latch muss die folgende Funktion erfüllen:

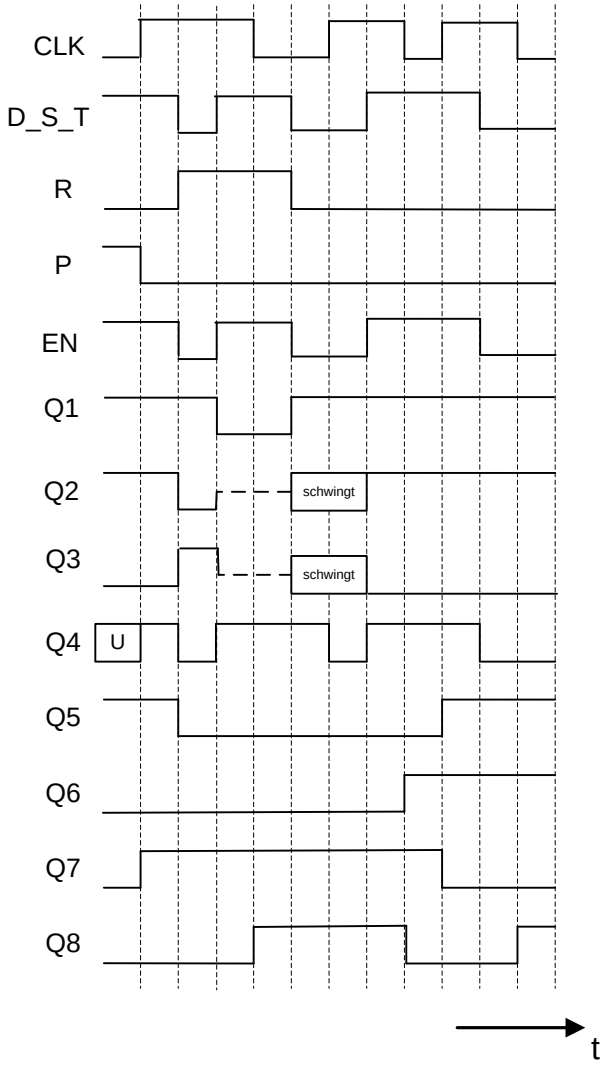
R	S	R'	S'
0	0	0	0
0	1	0	1
1	0	1	0
1	1	**0**	**1**

Daraus erhält man die Gleichungen S' = S und R' = $\overline{S} \wedge R$, Die gesamte Schaltung zeigt das nachfolgende Bild:



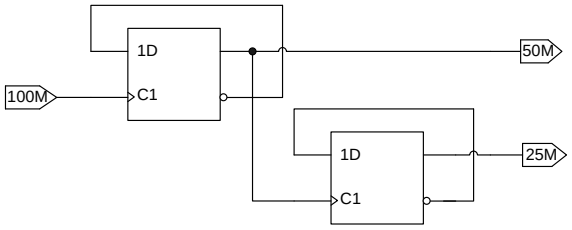
A Aufgabe 11.8 und Aufgabe 11.9

Das nachfolgende Impulsdiagramm zeigt den Signalverlauf zu den Aufgaben 11.8 und 11.9:



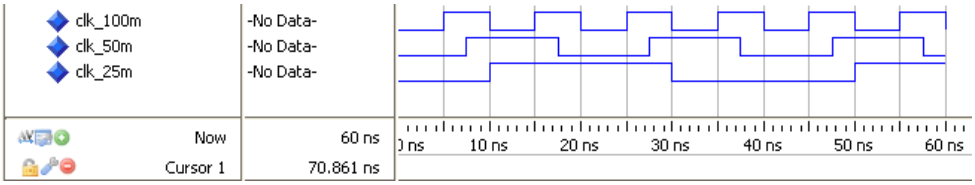
A Aufgabe 11.10

Als Taktteiler durch zwei können D-Flipflops verwendet werden, deren Ausgänge invertiert auf den D-Eingang gelegt werden. Die entsprechende Schaltung zeigt das folgende Bild:




```
entity AUFGABE_11_10 is
      port(   CLK_100M: in bit;
              CLK_50M, CLK_25M: out bit);
end AUFGABE_11_10;
-------------------------------------------
architecture VERHALTEN of AUFGABE_11_10 is
signal CLK_50M_TEMP, CLK_25M_TEMP: bit;
begin
P1:   process(CLK_100M)
      begin
         if CLK_100M'event and CLK_100M='1' then
                  CLK_50M_TEMP <= not CLK_50M_TEMP after 2500 ps;
         end if;
      end process P1;
P2:   process(CLK_50M_TEMP)
      begin
         if CLK_50M_TEMP'event and CLK_50M_TEMP='1' then
                  CLK_25M_TEMP <= not CLK_25M_TEMP after 2500 ps;
         end if;
      end process P2;
CLK_50M <= CLK_50M_TEMP;
CLK_25M <= CLK_25M_TEMP;
end VERHALTEN;
-------------------------------------------
-- Testbench zu Aufgabe 11.10
entity AUFGABE_11_10_TB is
end AUFGABE_11_10_TB;

architecture TESTBENCH of AUFGABE_11_10_TB is
signal CLK_100M, CLK_50M, CLK_25M: bit;

component AUFGABE_11_10 is
      port(   CLK_100M: in bit;
              CLK_50M, CLK_25M: out bit);
end component;

begin
DUT: AUFGABE_11_10 port map (CLK_100M, CLK_50M, CLK_25M);
P1: process
begin
   CLK_100M <= '0'; wait for 5 ns;
   CLK_100M <= '1'; wait for 5 ns;
end process P1;
end TESTBENCH;
```

Die Simulation zeigt, dass die Ausgangssignale der beiden Taktteilerflipflops asynchron zum Eingangstakt sind. Bei der Realisierung eines vollständig synchronen Systems wird daher empfohlen, zur Ansteuerung der Flipflops in den langsameren Taktdomänen die Ausgangssignale CLK_50M und CLK_25M auf die Enable-Eingänge der Flipflops zu legen und an die Takteingänge den 100-MHz-Systemtakt CLK_100M anzuschließen.



12 Entwurf synchroner Zustandsautomaten

A Aufgabe 12.1

a) Bein einem Moore-Automaten kann sich ein Ausgangssignal nur nach einer zuvor erfolgten Zustandsänderung (Taktflanke) ändern, bei einem Mealy-Automaten kann sich ein Ausgangssignal zusätzlich auch dann ändern, wenn sich ein Eingangssignal geändert hat.

b) Mealy- und Moore-Automaten besitzen neben einem taktflankengesteuerten Zustandsregister jeweils ein Übergangs- sowie ein Ausgangsschaltnetz. Bei Medwedew-Automaten fehlt das Ausgangsschaltnetz (vgl. Bilder 12.3a, 12.8a und 12.11)

c) Bei Moore-Zuständen befinden sich die Ausgnagssignalwerte in der untern Hälfte des Zustandskreise, bei einem Mealy-Automaten befinden sich die Ausgangssignalwerte hingegen hinter einem Schrägstrich am Übergangspfeil.

d) Die Zuordnung von Bitkombinationen zu den einzelnen Zuständen

e) Eine Kombination von Zustandsbits, die innerhalb des Automaten keinem Zustand zugeordnet ist. Beim Automatenentwurf muss analysiert werden, wie der Automat reagiert, wenn er unerwartet in einen Pseudozustand gerät und anschliessend neue Eingangssignale angelegt werden.

f) Aufspaltung all derer Mealy-Zustände, die unterschiedliche Ausgangssignalkombinationen zur Folge haben können, in entsprechend viele Moore-Zustände.

g) Trennung von Zustandsregister und Schaltnetzen in zwei unterschiedliche Prozesse. Abbildung des Zustandsdiagramms in dem kombinatorischen Prozess. Dessen aktuelle Zustände werden in einer case-Anweisung analysiert. In jedem Zustand werden die Eingangssignalwerte durch eine if-then-else Anweisung analysiert und die Folgezustände bzw. Ausgangssignale zugewiesen (vgl. z.B. Listing 12.3).

h) Bei einem Moore-Automaten erfolgt die Zuweisung der Ausgangssignale ausserhalb der if-Bedingungen, die die Eingangssignale abprüfen, und zwar sinnvollerweise vor der if-Bedingung.

A Aufgabe 12.2

Die Folgezustandstabelle zu Bild 12.14 lautet:

Zustand S1 S0	Eingabe E1 E0	Folgezustand $S1^+$ $S0^+$	Ausgabe A
0 0	0 0	0 0	0
	0 1	0 1	0
	1 0	0 0	0
	1 1	0 0	0
0 1	0 0	0 0	0
	0 1	0 1	0
	1 0	0 0	0
	1 1	1 1	0
1 0	0 0	X X	X
	0 1	X X	X
	1 0	X X	X
	1 1	X X	X
1 1	0 0	0 0	0
	0 1	0 1	0
	1 0	0 0	1
	1 1	0 0	0

Mit der KV-Minimierungsmethode enthält man daraus die folgenden logischen Gleichungen:

$S1^+ = \overline{S1} \wedge S0 \wedge E1 \wedge E0$; $S0^+ = (\overline{E1} \wedge E0) \vee (\overline{S1} \wedge S0 \wedge E0)$; $A = S1 \wedge E1 \wedge \overline{E0}$

Eine Pseudozustandsanalyse ergibt die folgenden Übergänge aus dem Pseudozustand (1 0):

E1 E0	$S1^+$ $S0^+$	A
0 0	0 0	0
0 1	0 1	0
1 0	0 0	1
1 1	0 0	0

Durch den Eingangssignalvektor (1 0) wird also im Pseudozustand ein fehlerhaftes A = 1 erzeugt.

A Aufgabe 12.3

Für den Automaten wird eine binäre Zustandscodierung gewählt:

Aufgabenlösungen zu: Jürgen Reichardt, Lehrbuch Digitaltechnik 3. Auflage
ISBN 978-3-486-72765-4. Oldenbourg Wissenschaftsverlag, München 2013.

	S1 S0
Z0	0 0
Z1	0 1
Z2	1 0
Z3	1 1

Daraus erhält man die nachfolgende Folgezustandstabelle:

Zustand S1 S0	Eingabe E1 E0	Folgezustand $S1^+$ $S0^+$	Ausgabe A
0 0	0 0	0 0	0
	0 1	0 1	
	1 0	0 0	
	1 1	0 0	
0 1	0 0	0 0	0
	0 1	0 1	
	1 0	0 0	
	1 1	1 0	
1 0	0 0	0 0	0
	0 1	0 1	
	1 0	1 1	
	1 1	0 0	
1 1	0 0	0 0	1
	0 1	0 1	
	1 0	0 0	
	1 1	0 0	

Ergebnis einer KV-Minimierung sind die logischen Gleichungen:

$S1^+ = (\overline{S1} \wedge S0 \wedge E1 \wedge E0) \vee (S1 \wedge \overline{S0} \wedge E1 \wedge \overline{E0})$

$S0^+ = (\overline{E1} \wedge E0) \vee (S1 \wedge \overline{S0} \wedge E1 \wedge \overline{E0})$

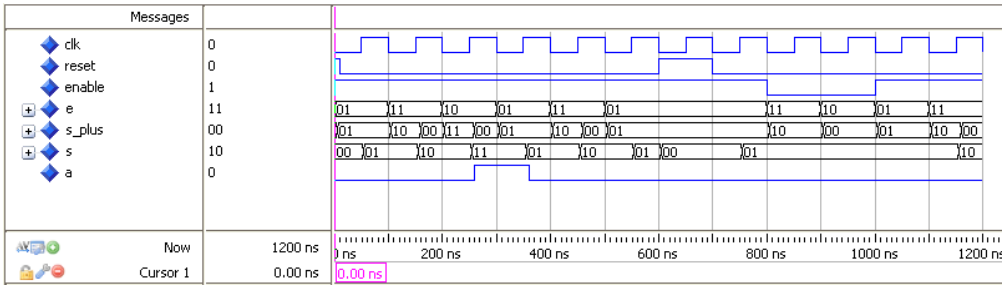
$A = S0 \wedge S1$

Diese Gleichungen finden sich im Prozess NETZE des Datenflussmodells zum Automaten:

```
-- Sequenzerkennung aus Listing 12.3
-- Realisierung als Datenflussmodell
architecture DATENFLUSS of SEQUENZ_ERK is
signal S,S_PLUS: bit_vector(1 downto 0) ; -- Zustandsbits
begin
REG: process(CLK, RESET) -- Zustandsaktualisierung
```

```
begin
      if  RESET = '1' then S <= "00" after 5 ns;
      elsif CLK = '1' and CLK'event then
            if ENABLE = '1' then
                  S <= S_PLUS after 5 ns;
            end if;
      end if;
end process REG;

NETZE: process(E, S)  -- Kombinatorische Logik
begin
   S_PLUS(1) <= (E(0) and E(1) and S(0) and not S(1) ) or
                (not E(0) and E(1) and not S(0) and S(1)) after 5 ns;
   S_PLUS(0) <= (E(0) and not E(1)) or
                (not E(0) and E(1) and not S(0) and S(1)) after 5 ns;
   A <= S(0) and S(1) after 5 ns;
end process NETZE;
end DATENFLUSS;
```

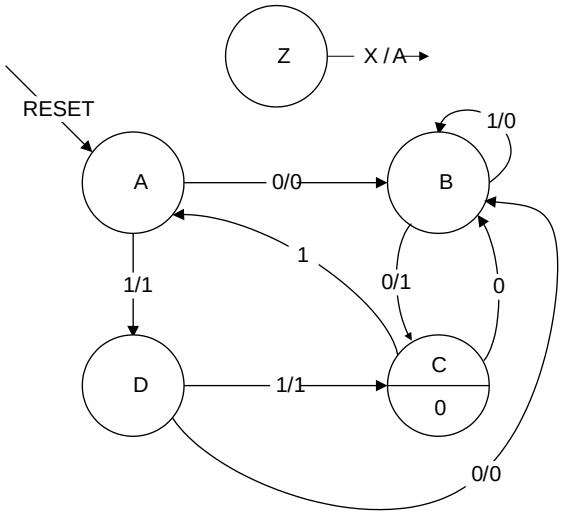
Die Simulation des Automaten zeigt das nachfolgende Bild:



Im Unterschied zu Bild 12.13 wird für den Zustand und den Folgezustand kein symbolischer Zustandstyp verwendet, sondern ein 2-Bit Zustandstyp gemäß der oben gewählten Zustandscodierung. Ansonsten ist das Simulationsergebnis identisch mit dem aus Bild 12.13.

A Aufgabe 12.4

Das Zustandsdiagramm zeigt nachfolgendes Bild:



Für den Automaten wird eine Zustandscodierung im Gray-Code gewählt:

	S1 S0
A	0 0
B	0 1
C	1 1
D	1 0

Daraus erhält man die nachfolgende Folgezustandstabelle:

Zustand S1 S0	Eingabe X	Folgezustand $S1^+$ $S0^+$	Ausgabe A
0 0	0	0 1	0
0 0	1	1 0	1
0 1	0	1 1	1
0 1	1	0 1	0
1 0	0	0 1	0
1 0	1	1 1	1
1 1	0	0 1	0
1 1	1	0 0	0

Als Ergebnis einer KV-Minimierung ergeben sich die logischen Gleichungen:

$S1^+ = (X \wedge \overline{S0}) \vee (\overline{X} \wedge S0 \wedge \overline{S1})$

$S0^{+} = \overline{X} \vee (\overline{S0} \wedge S1) \vee (S0 \wedge \overline{S1})$

$A = (X \wedge \overline{S0}) \vee (\overline{X} \wedge S0 \wedge \overline{S1})$

Das nachfolgende VHDL-Modell dieses Automaten enthält ein Verhaltensmodell sowie ein Datenflussmodell des Automaten. Außerdem ist eine Testbench angegeben, in der durch eine VHDL-Konfigurationsanweisung jeweils eine Komponente (DUT_0, bzw. DUT_1) dieser Modelle instanziiert wird.

In dem Verhaltensmodell wird als Default-Folgezustand ZB und als Default-Ausgangssignal A = 0 gewählt, da diese Werte im Zustandsdiagramm dominieren und somit eine Reduzierung des Schreibaufwands im Code erlauben.

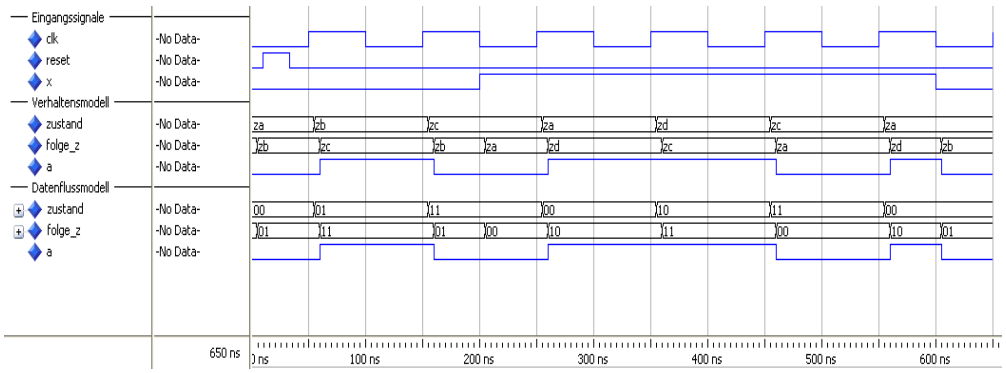
Mit dem in der Testbench angegebenen Stimulus-Prozess wird durch sechs exemplarische Testvektoren die Zustandsfolge A, B, C, A, D, C, A durchlaufen. Im Monitor-Prozess wird die Korrektheit der Ausgangssignale A_0 bzw. A_1 in diesen Zuständen jeweils kurz vor der nächsten steigenden Taktflanke überprüft.

```
entity AUFGABE_12_4 is
port( CLK, RESET     : in  bit;
            X : in bit;
            A : out bit
     );
end AUFGABE_12_4;
-----------------------------------------------------------------------
-- Realisierung als Verhaltensmodell
architecture VERHALTEN of AUFGABE_12_4 is

type ZUSTAENDE is (ZA, ZB, ZC, ZD);
signal ZUSTAND, FOLGE_Z: ZUSTAENDE;

begin
REG: process(CLK, RESET) -- Zustandsaktualisierung
begin
      if  RESET = '1' then ZUSTAND <= ZA after 5 ns;
      elsif CLK = '1' and CLK'event then
         ZUSTAND <= FOLGE_Z after 5 ns;
      end if;
end process REG;
NETZE: process(X, ZUSTAND)
begin
   FOLGE_Z <= ZB after 5 ns; -- Default-Folgezustand
   A <= '0' after 5 ns;      -- Default Ausgangssignal
   case ZUSTAND is
      when ZA => if X = '1' then
                    FOLGE_Z <= ZD after 5 ns;
                    A <= '1' after 5 ns;
                 end if;
      when ZB => if X = '0' then
                    FOLGE_Z <= ZC after 5 ns;
                    A <= '1' after 5 ns;
                 end if;
      when ZC => if X = '1' then
                    FOLGE_Z <= ZA after 5 ns;
                 end if;
```

```
      when ZD => if X = '1' then
                    FOLGE_Z <= ZC after 5 ns;
                    A <= '1' after 5 ns;
                 end if;
   end case;
end process NETZE;
end VERHALTEN;
-----------------------------------------------------------------------
-- Realisierung als Datenflussmodell
architecture DATENFLUSS of AUFGABE_12_4 is

-- Verwende zur Zustandscodierung 2-Bit-Konstanten:
constant ZA : bit_vector(1 downto 0) := "00";
constant ZB : bit_vector(1 downto 0) := "01";
constant ZC : bit_vector(1 downto 0) := "11";
constant ZD : bit_vector(1 downto 0) := "10";

signal ZUSTAND, FOLGE_Z: bit_vector(1 downto 0);

begin
REG: process(CLK, RESET) -- Zustandsaktualisierung
begin
      if  RESET = '1' then ZUSTAND <= ZA after 5 ns;
      elsif CLK = '1' and CLK'event then
         ZUSTAND <= FOLGE_Z after 5 ns;
      end if;
end process REG;
NETZE: process(X, ZUSTAND)
begin
   FOLGE_Z(1) <= ((X and not ZUSTAND(0))
                 or (not X and ZUSTAND(0) and not ZUSTAND(1))) after 5 ns;
   FOLGE_Z(0) <= (not X
                 or (not ZUSTAND(0) and ZUSTAND(1))
                 or (ZUSTAND(0) and not ZUSTAND(1))) after 5 ns;
   A          <=((X and not ZUSTAND(0))
                 or (not X and ZUSTAND(0) and not ZUSTAND(1))) after 5 ns;
end process NETZE;
end DATENFLUSS;
-----------------------------------------------------------------------------
-
entity AUFGABE_12_4_tb is
end AUFGABE_12_4_tb;

architecture TESTBENCH of AUFGABE_12_4_tb is

-- Komponentendeklaration:
component AUFGABE_12_4 is
port( CLK, RESET    : in  bit;
            X : in bit;
            A : out bit );
end component;

-- Komponentenkonfigurationen:
for DUT0: AUFGABE_12_4 use entity work.AUFGABE_12_4(VERHALTEN);
for DUT1: AUFGABE_12_4 use entity work.AUFGABE_12_4(DATENFLUSS);

signal CLK, RESET, X, A_0, A_1 : bit;
signal TEST_NR: integer range 0 to 5;

begin
```

```
-- je eine Verhaltens- und Datenflussmodellinstanz:
DUT0: AUFGABE_12_4 port map(CLK, RESET, X, A_0);
DUT1: AUFGABE_12_4 port map(CLK, RESET, X, A_1);

TAKTGEN: process                -- Taktgenerator
begin
   CLK <= '0'; wait for 50 ns;
   CLK <= '1'; wait for 50 ns;
end process TAKTGEN;

RESET<= '0', '1' after 10 ns, '0' after 33 ns; -- asynchroner Reset

STIMULI: process
begin
   TEST_NR <= 0; X <= '0'; wait for 100 ns;
   TEST_NR <= 1; X <= '0'; wait for 100 ns;
   TEST_NR <= 2; X <= '1'; wait for 100 ns;
   TEST_NR <= 3; X <= '1'; wait for 100 ns;
   TEST_NR <= 4; X <= '1'; wait for 100 ns;
   TEST_NR <= 5; X <= '1'; wait for 100 ns;
end process STIMULI;

MONITOR: process
begin
   wait for 45 ns; -- kurz vor der steigenden Flanke
   for I in 0 to 5 loop
     case TEST_NR is
       when 0 => assert A_0 = '0' report "DUT0:Ausgangssignalfehler TEST 0";
                 assert A_1 = '0' report "DUT1:Ausgangssignalfehler TEST 0";
       when 1 => assert A_0 = '1' report "DUT0:Ausgangssignalfehler TEST 1";
                 assert A_1 = '1' report "DUT1:Ausgangssignalfehler TEST 1";
       when 2 => assert A_0 = '0' report "DUT0:Ausgangssignalfehler TEST 2";
                 assert A_1 = '0' report "DUT1:Ausgangssignalfehler TEST 2";
       when 3 => assert A_0 = '1' report "DUT0:Ausgangssignalfehler TEST 3";
                 assert A_1 = '1' report "DUT1:Ausgangssignalfehler TEST 3";
       when 4 => assert A_0 = '1' report "DUT0:Ausgangssignalfehler TEST 4";
                 assert A_1 = '1' report "DUT1:Ausgangssignalfehler TEST 4";
       when 5 => assert A_0 = '0' report "DUT0:Ausgangssignalfehler TEST 5";
                 assert A_1 = '0' report "DUT1:Ausgangssignalfehler TEST 5";
       end case;
       wait for 100 ns; -- warte bis kurz vor der naechsten Flanke
   end loop;
end process MONITOR;

end TESTBENCH;
```

Das nachfolgende Simulationsergebnis zeigt das gewünschte Verhalten für das Verhaltens- sowie für das Datenflussmodell:



A Aufgabe 12.5

Für den Moore-Automaten mit vier Zuständen wird eine binäre Zustandscodierung gewählt:

	S1 S0
ZA	0 0
ZB	0 1
ZC	1 0
ZD	1 1

Daraus erhält man die nachfolgende Folgezustandstabelle:

Zustand S1 S0	Eingabe X	Folgezustand $S1^+$ $S0^+$	Ausgabe Y
0 0	0	0 1	0
0 0	1	1 1	
0 1	0	1 0	0
0 1	1	0 1	
1 0	0	0 1	1
1 0	1	0 0	
1 1	0	0 1	1
1 1	1	1 0	

Als Ergebnis einer KV-Minimierung ergeben sich die logischen Gleichungen:

$S1^{+} = (X \wedge \overline{S0} \wedge \overline{S1}) \vee (X \wedge S0 \wedge S1) \vee (\overline{X} \wedge S0 \wedge \overline{S1})$

$S0^{+} = (X \wedge \overline{S1}) \vee (\overline{X} \wedge \overline{S0}) \vee (\overline{X} \wedge S0 \wedge S1)$

$Y = S1$

In dem Verhaltensmodell wird als Default-Folgezustand ZB und als Default-Ausgangssignal Y = 0 gewählt, da diese Werte im Zustandsdiagramm dominieren und somit eine Reduzierung des Schreibaufwands im Code erlauben.

Die im Testbench-Code angegebenen Stimuli wurden exemplarisch so gewählt, dass die Zustandsfolge ZA, ZB, ZB, ZC, ZA, ZD durchlaufen wird.

```
entity AUFGABE_12_5 is
port( CLK, RESET     : in  bit;
            X : in bit;
            Y : out bit
     );
end AUFGABE_12_5;
-----------------------------------------------------------------------
-- Realisierung als Verhaltensmodell
architecture VERHALTEN of AUFGABE_12_5 is

type ZUSTAENDE is (ZA, ZB, ZC, ZD);
signal ZUSTAND, FOLGE_Z: ZUSTAENDE;

begin
REG: process(CLK, RESET) -- Zustandsaktualisierung
begin
      if  RESET = '1' then ZUSTAND <= ZA after 5 ns;
      elsif CLK = '1' and CLK'event then
         ZUSTAND <= FOLGE_Z after 5 ns;
      end if;
end process REG;
NETZE: process(X, ZUSTAND)
begin
   FOLGE_Z <= ZB after 5 ns; -- Default-Folgezustand
   Y <= '0' after 5 ns;      -- Default Ausgangssignal
   case ZUSTAND is
      when ZA => if X = '1' then
                    FOLGE_Z <= ZD after 5 ns;
                 end if;
      when ZB => if X = '0' then
                    FOLGE_Z <= ZC after 5 ns;
                 end if;
      when ZC => Y <= '1';
                 if X = '1' then
                    FOLGE_Z <= ZA after 5 ns;
                 end if;
      when ZD => Y <= '1';
                 if X = '1' then
                    FOLGE_Z <= ZC after 5 ns;
                 end if;
   end case;
end process NETZE;
end VERHALTEN;
```

```
-------------------------------------------------------------------------------
entity AUFGABE_12_5_tb is
end AUFGABE_12_5_tb;

architecture TESTBENCH of AUFGABE_12_5_tb is

-- Komponentendeklaration:
component AUFGABE_12_5 is
port( CLK, RESET     : in  bit;
            X : in bit;
            Y : out bit );
end component;

signal CLK, RESET, X, Y : bit;
signal TEST_NR: integer range 0 to 5;

begin
-- je eine Verhaltens- und Datenflussmodellinstanz:
DUT: AUFGABE_12_5 port map(CLK, RESET, X, Y);

TAKTGEN: process                -- Taktgenerator
begin
   CLK <= '0'; wait for 50 ns;
   CLK <= '1'; wait for 50 ns;
end process TAKTGEN;

RESET<= '0', '1' after 10 ns, '0' after 33 ns; -- asynchroner Reset

STIMULI: process
begin
   TEST_NR <= 0; X <= '0'; wait for 100 ns;
   TEST_NR <= 1; X <= '1'; wait for 100 ns;
   TEST_NR <= 2; X <= '0'; wait for 100 ns;
   TEST_NR <= 3; X <= '1'; wait for 100 ns;
   TEST_NR <= 4; X <= '1'; wait for 100 ns;
   TEST_NR <= 5; X <= '1'; wait for 100 ns;
end process STIMULI;

MONITOR: process
begin
   wait for 45 ns; -- kurz vor der steigenden Flanke
   for I in 0 to 5 loop
       case TEST_NR is
          when 0 => assert Y = '0' report "Reset-Fehler: kein Zustand ZA";
          when 1 => assert Y = '0' report "Fehler: kein Zustand ZB";
          when 2 => assert Y = '0' report "Fehler: kein Zustand ZB";
          when 3 => assert Y = '1' report "Fehler: kein Zustand ZC";
          when 4 => assert Y = '0' report "Fehler: kein Zustand ZA";
          when 5 => assert Y = '1' report "Fehler: kein Zustand ZD";
       end case;
       wait for 100 ns; -- warte bis kurz vor der naechsten Flanke
   end loop;
end process MONITOR;

end TESTBENCH;
```

13 Entwurf von Synchronzählern

A Aufgabe 13.1

a) Alle im Zähler verwendeten Flipflops werden mit dem gleichen Taktsignal aktualisiert.

b) I) Dies sind gesteuerte Eingangs- bzw. Ausgangssignalfunktionen

II) Dies sind steuernde Eingangssignale.

c) I) eine UND-Abhängigkeit, d.h. das Signal hat die Funktion eines Tors (Gate), d.h. das steuernde Signal muss den logischen Wert 1 haben, um die zugehörige Gatterfunktion zu erreichen.

II) Die Auswahl einer bestimmten Betriebsart (Modus)

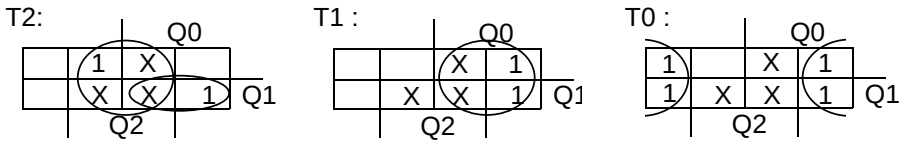
d) Es sind 2 Prozesse erforderlich, ein getakteter für das Zählregister, ein zweiter zur kombinatorischen Auswertung des Zählzustands (vgl. z.B. Bild 13.14).

e) Entweder vollständig seriell oder durch Carry-Lookahead Kaskadierung (vgl. Bild 13.17).

f) Vorteilhaft sind T-Flipflops da diese für T=1 automatisch ihren logischen Wert invertieren (toggeln), so wie es in einem Zählzyklus erforderlich ist.

A Aufgabe 13.2

Ausgehend von der Tabelle 13.2 erhält man die folgenden KV-Diagramme bzw. logischen Gleichungen:



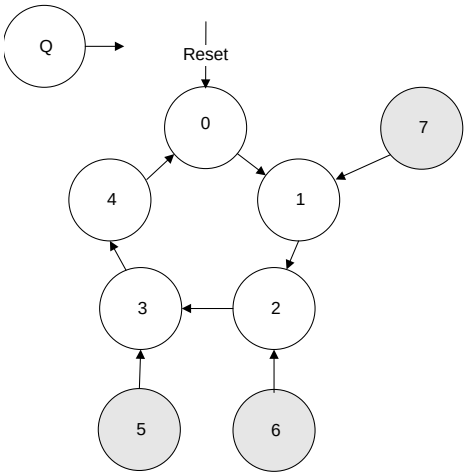
$T2 = Q2 \vee (Q1 \wedge Q0)$ $T1 = Q0$ $T0 = \overline{Q2}$

Aus den KV-Diagrammen lassen sich auch die Pseudozustände bestimmen:

Z	Q2 Q1 Q0	T2 T1 T0	$Q2^+$ $Q1^+$ $Q0^+$	Z^+
5	1 0 1	1 1 0	0 1 1	3
6	1 1 0	1 0 0	0 1 0	2
7	1 1 1	1 1 0	0 0 1	1

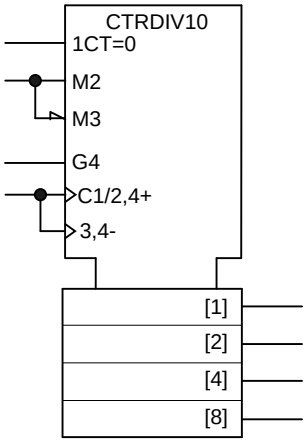
Damit erhält man das vollständige Zustandsdiagramm für den mit T-Flipflops aufgebauten mod-5-Zähler:

Aufgabenlösungen zu: Jürgen Reichardt, Lehrbuch Digitaltechnik 3. Auflage
ISBN 978-3-486-72765-4. Oldenbourg Wissenschaftsverlag, München 2013.



A Aufgabe 13.3

Die doppelte Funktion des Taktes (Vorwärts- bzw. Rückwärtszählen) wird im Schaltsymbol durch zwei Eingänge dargestellt:

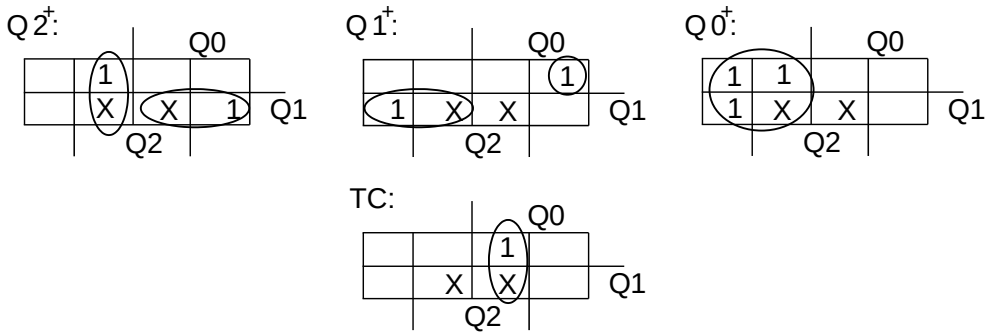


A Aufgabe 13.4

Für den mod-6-Zähler dient die folgende Zustandsfolgetabelle, die Folgezustandsspalten für die Realisierung mit D- und T-Flipflops enthält:

Q2	Q1	Q0	$Q2^+$	$Q1^+$	$Q0^+$	TC	T2	T1	T0
0	0	0	0	0	1	0	0	0	1
0	0	1	0	1	0	0	0	1	1
0	1	0	0	1	1	0	0	0	1
0	1	1	1	0	0	0	1	1	1

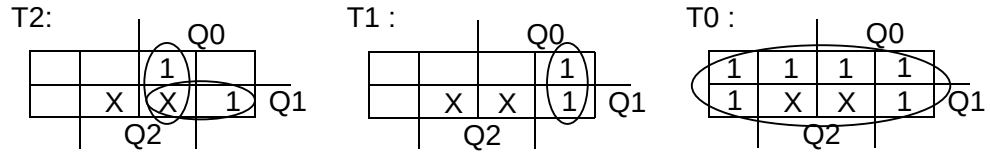
1	0	0	1	0	1	0	0	0	1
1	0	1	0	0	0	1	1	0	1
1	1	0	X	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X	X



$Q2^+ = (Q2 \wedge \overline{Q0}) \vee (Q1 \wedge Q0)$; $Q1^+ = (Q1 \wedge \overline{Q0}) \vee (\overline{Q2} \wedge \overline{Q1} \wedge Q0)$; $Q0^+ = \overline{Q0}$

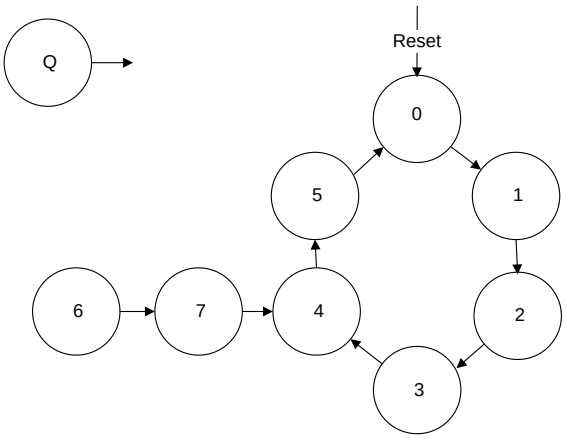
$TC = Q0 \wedge Q2$

Die Minimierung der T-Flipflop-Übergangsfunktionen ergibt:



$T2 = (Q0 \wedge Q2) \vee (Q0 \wedge Q1)$; $T1 = Q0 \wedge \overline{Q2}$; $T0 = 1$

Die Pseudozustandsanalyse für die Realisierung mit D-Flipflops ergibt das folgende komplette Zustandsdiagramm:



Aus einem Pseudozustand kehrt der Zähler also spätestens nach zwei Takten in den Zählzyklus zurück.

Das nachfolgende VHDL-Modell verwendet einen asynchronen Reset:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;           -- Erlaubt Addition von Vektoren
entity  AUFGABE_13_4 is
  port (RESET,CLK  : in BIT;
        Q: out BIT_VECTOR(2 downto 0);
        TC: out BIT);
end AUFGABE_13_4 ;
architecture VERHALTEN of AUFGABE_13_4 is
signal QINT: STD_LOGIC_VECTOR(2 downto 0);
begin

SYN_COUNT:process (CLK, RESET)             -- Takt und asynchrone Eingaenge
begin
   if RESET = '1' then
      QINT <= (others=>'0') after 10 ns;
   elsif CLK='1' and CLK'event then
      if QINT = 5 then
         QINT <= (others => '0') after 10 ns;
      else
         QINT <= QINT + 1 after 10 ns;
      end if;
   end if;
end process SYN_COUNT;
Q <= To_bitvector(QINT);                   -- Typ Umwandlung

-- Ausgangsschaltnetz:
TC <= '1' after 10 ns when (QINT = 5) else '0' after 10 ns;
end VERHALTEN;
```

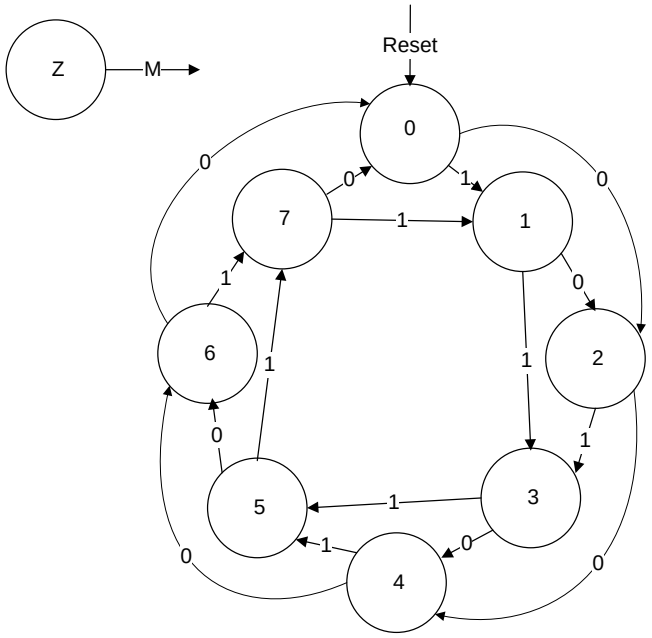
Für die Simulation mir ModelSim wurde die nachfolgende Macro-Datei entworfen:

```
restart
radix hex
add wave sim:/aufgabe_13_4/reset
```

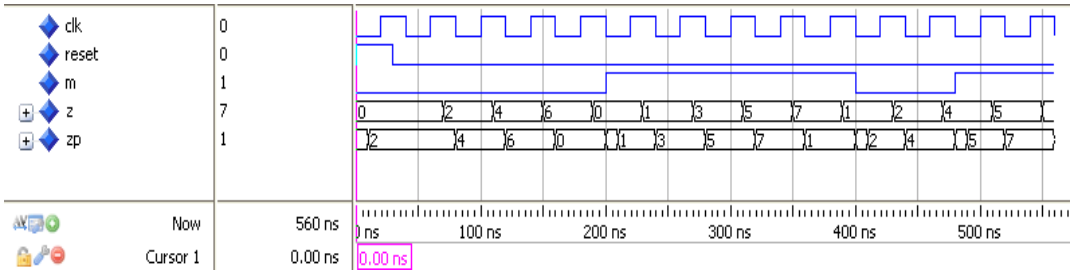
```
add wave sim:/aufgabe_13_4/clk
add wave sim:/aufgabe_13_4/qint
add wave sim:/aufgabe_13_4/q
add wave sim:/aufgabe_13_4/tc

force clk 0 0, 1 50ns -r 100ns
force reset  1 0, 0 80ns
run 1000ns
```

A Aufgabe 13.5

Das Zustandsdiagramm für den Zähler ist:



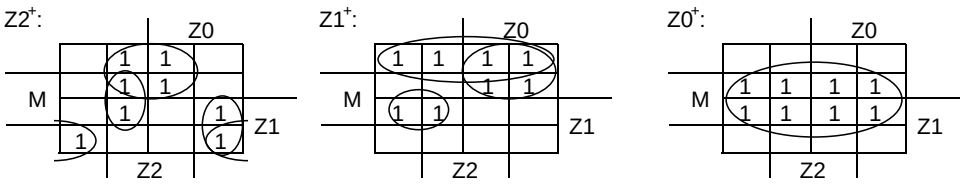
Das vervollständigte Impulsdiagramm zu Bild 13.17 zeigt nachfolgendes Bild:



Die Folgezustandstabelle wird gleich für D- und T-Flipflops angegeben:

M	Z2 Z1 Z0	Z^+	$Z2^+$ $Z1^+$ $Z0^+$	T2 T1 T0
0	0 0 0	2	0 1 0	0 1 0
0	0 0 1	2	0 1 0	0 1 1
0	0 1 0	4	1 0 0	1 1 0
0	0 1 1	4	1 0 0	1 1 1
0	1 0 0	6	1 1 0	0 1 0
0	1 0 1	6	1 1 0	0 1 1
0	1 1 0	0	0 0 0	1 1 0
0	1 1 1	0	0 0 0	1 1 1
1	0 0 0	1	0 0 1	0 0 1
1	0 0 1	3	0 1 1	0 1 0
1	0 1 0	3	0 1 1	0 0 1
1	0 1 1	5	1 0 1	1 1 0
1	1 0 0	5	1 0 1	0 0 1
1	1 0 1	7	1 1 1	0 1 0
1	1 1 0	7	1 1 1	0 0 1
1	1 1 1	1	0 0 1	1 1 0

Die Implementierung mit D-Flipflops ergibt:



$Z2^+ = (Z2 \wedge \overline{Z1}) \vee (M \wedge Z2 \wedge \overline{Z0}) \vee (\overline{Z2} \wedge Z1 \wedge Z0) \vee (\overline{M} \wedge \overline{Z2} \wedge Z1)$

$Z1^+ = (\overline{M} \wedge \overline{Z1}) \vee (\overline{Z1} \wedge \overline{Z0}) \vee (M \wedge Z1 \wedge \overline{Z0})$

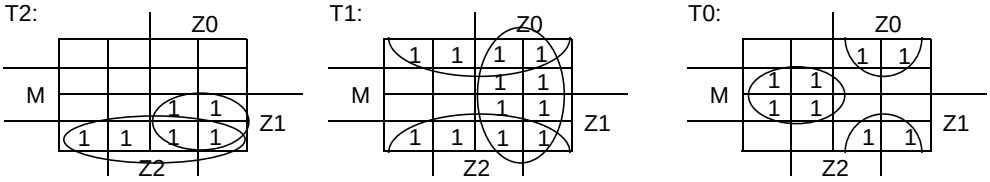
$Z0^+ = M$

Unter der Annahme, dass in einer CMOS-Technologie jeder Gattereingang zwei Transistoren erfordert, ergibt sich der Hardwareaufwand wie folgt:

- 3-fach-UND-Gatter: $4 \Leftrightarrow 6 \cdot 4 = 24$ Transistoren
- 2-fach-UND-Gatter: $3 \Leftrightarrow 4 \cdot 3 = 12$ Transistoren
- 4-fach-ODER-Gatter: $1 \Leftrightarrow 8 \cdot 1 = 8$ Transistoren
- 3-fach-ODER-Gatter: $1 \Leftrightarrow 8 \cdot 1 = 8$ Transistoren

Dies sind in der Summe 50 Transistoren in der Folgezustandslogik.

Für die Realisierung mit T-Flipflops gilt:



$T2 = (Z1 \wedge Z0) \vee (\overline{M} \wedge Z1)$

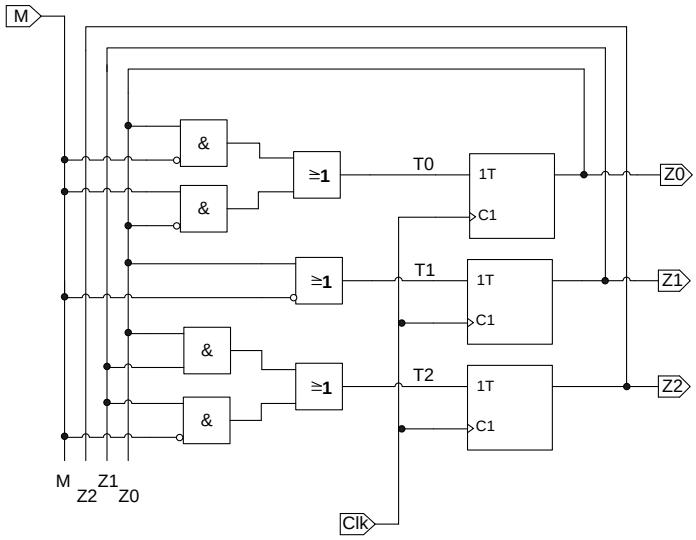
$T1 = \overline{M} \vee Z0$

$T0 = (\overline{M} \wedge Z0) \vee (M \wedge \overline{Z0})$

Der Hardwareaufwand dieser Lösung ist:

- 2-fach-UND-Gatter: $4 \Leftrightarrow 4 \cdot 4 = 16$ Transistoren
- 2-fach-ODER-Gatter: $3 \Leftrightarrow 4 \cdot 3 = 12$ Transistoren

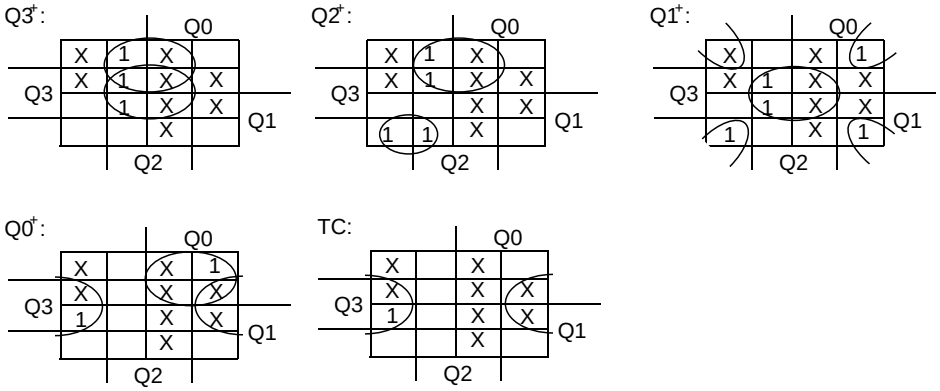
Also in der Summe 28 Transistoren und damit etwas mehr als die Hälfte der Transistoren, die für die D-Flipflop-Lösung benötigt wird. Die komplette Schaltung ist nachfolgend dargestellt:



A Aufgabe 13.6

Folgezustandstabelle:

Q	Q3	Q2	Q1	Q0	Q^+	$Q3^+$	$Q2^+$	$Q1^+$	$Q0^+$	TC
-	0	0	0	0	-	X	X	X	X	X
0	0	0	0	1	1	0	0	1	1	0
2	0	0	1	0	3	0	1	1	0	0
1	0	0	1	1	2	0	0	1	0	0
4	0	1	0	0	5	1	1	0	0	0
-	0	1	0	1	-	X	X	X	X	X
3	0	1	1	0	4	0	1	0	0	0
-	0	1	1	1	-	X	X	X	X	X
-	1	0	0	0	-	X	X	X	X	X
-	1	0	0	1	-	X	X	X	X	X
7	1	0	1	0	0	0	0	0	1	1
-	1	0	1	1	-	X	X	X	X	X
5	1	1	0	0	6	1	1	1	0	0
-	1	1	0	1	-	X	X	X	X	X
6	1	1	1	0	7	1	0	1	0	0
-	1	1	1	1	1	X	X	X	X	X



$Q3^+ = (Q2 \wedge \overline{Q1}) \vee (Q3 \wedge Q2)$

$Q2^+ = (Q2 \wedge \overline{Q1}) \vee (\overline{Q3} \wedge Q1 \wedge \overline{Q0})$

$Q1^+ = (Q2 \wedge Q3) \vee (\overline{Q2} \wedge \overline{Q3})$

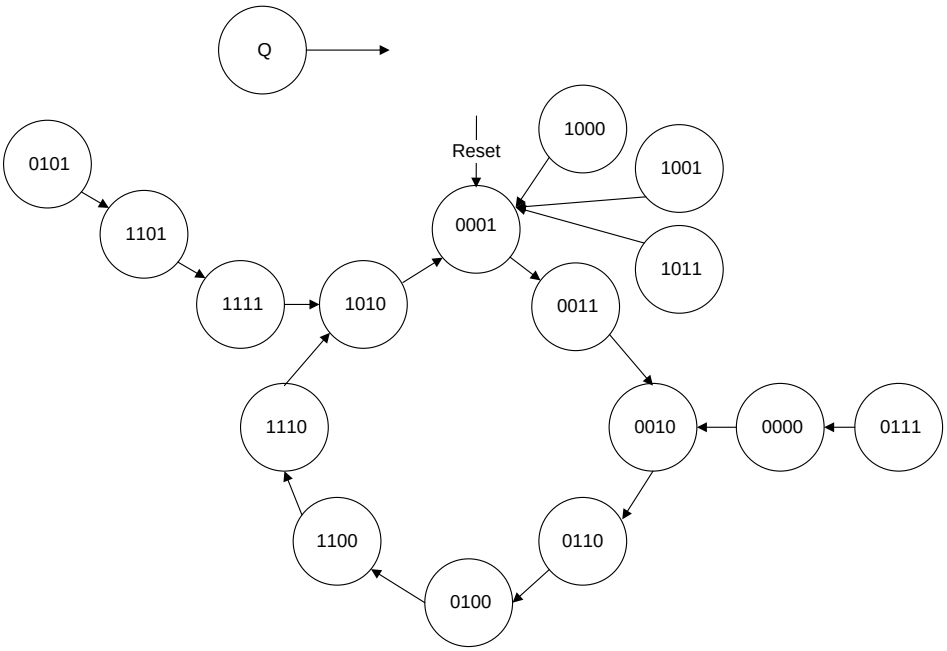
$Q0^+ = (Q3 \wedge \overline{Q2}) \vee (\overline{Q1} \wedge Q0)$

$TC = Q3 \wedge \overline{Q2}$

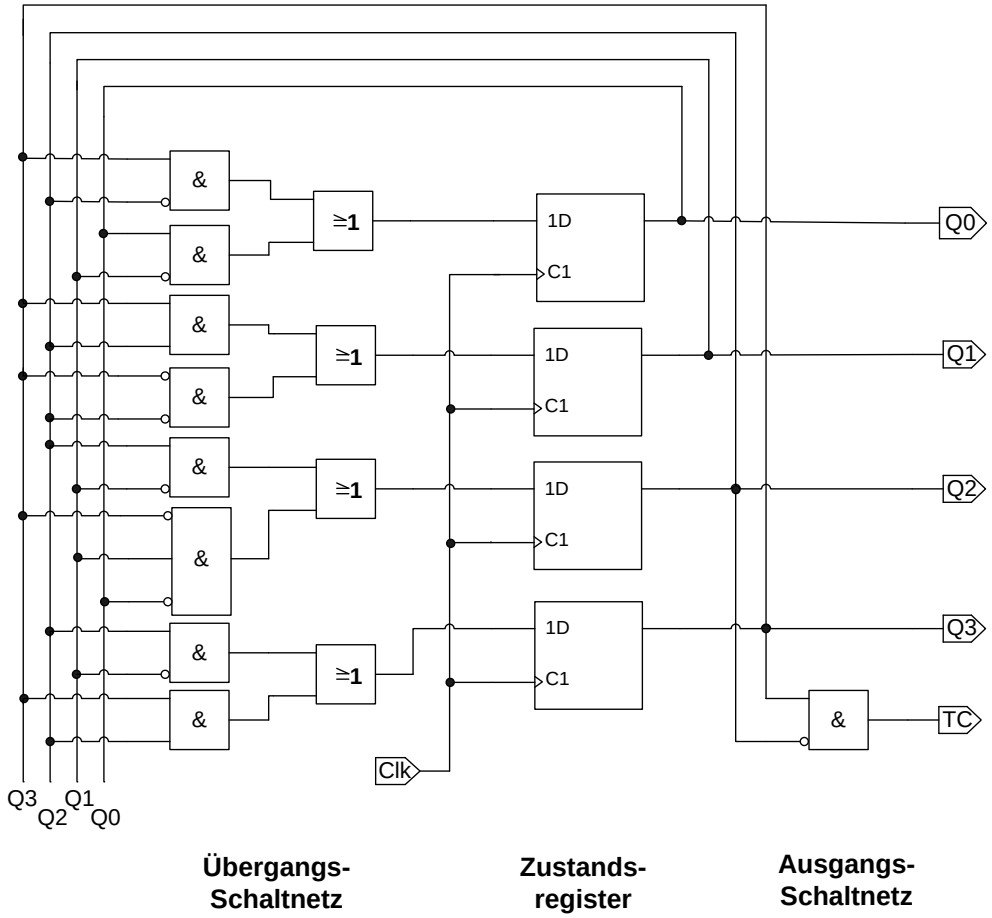
Aus dem KV-Diagramm ergeben sich die folgenden Folgezustände zu den Pseudozuständen:

Q	Q3	Q2	Q1	Q0	$Q3^+$	$Q2^+$	$Q1^+$	$Q0^+$	TC
m_0	0	0	0	0	0	0	1	0	0
m_5	0	1	0	1	1	1	0	1	0
m_7	0	1	1	1	0	0	0	0	0
m_8	1	0	0	0	0	0	0	1	1
m_9	1	0	0	1	0	0	0	1	1
m_{11}	1	0	1	1	0	0	0	1	1
m_{13}	1	1	0	1	1	1	1	1	0
m_{15}	1	1	1	1	1	0	1	0	0

Das vollständige Zustandsdiagramm zeigt die nachfolgende Abbildung. Es ist erkennbar, dass der Zähler nach spätestens drei Taktzyklen in den normalen Ablauf zurückkehrt:



Den Schaltungsaufbau mit der Kennzeichnung der Moore-Automatenelemente zeigt das nachfolgende Bild.



A Aufgabe 13.7

Das VHDL-Verhaltensmodell bildet die Folgezustandstabelle in Form einer case-Anweisung innerhalb eines getakteten Prozesses ab. In diesem Code ist der Folgezustand aller Pseudozustände der Reset-Zustand „00000“.

```
library IEEE;
entity  AUFGABE_13_7 is
  port (RESET, CLK, EN : in BIT;
        Q_OUT: out bit_vector(4 downto 0));
end AUFGABE_13_7;
```

```
architecture VERHALTEN of AUFGABE_13_7 is
signal Q: bit_vector(4 downto 0);
begin

P1:process (CLK, RESET)           -- Takt und asynchrone Eingaenge
begin
   if RESET = '1' then
      Q <= (others=>'0') after 10 ns;
   elsif CLK='1' and CLK'event then
      if EN = '1' then
         case Q is                 -- Zustandsfolgetabelle
            when "00000" => Q <= "00001" after 10 ns;
            when "00001" => Q <= "00011" after 10 ns;
            when "00011" => Q <= "00111" after 10 ns;
            when "00111" => Q <= "01111" after 10 ns;
            when "01111" => Q <= "11111" after 10 ns;
            when "11111" => Q <= "11110" after 10 ns;
            when "11110" => Q <= "11100" after 10 ns;
            when "11100" => Q <= "00000" after 10 ns; -- Ruecksprung
            when others =>  Q <= "00000" after 10 ns; -- Pseudozustände
         end case;
      end if;
   end if;
end process P1;
Q_OUT <= Q;
end VERHALTEN;
```

A Aufgabe 13.8

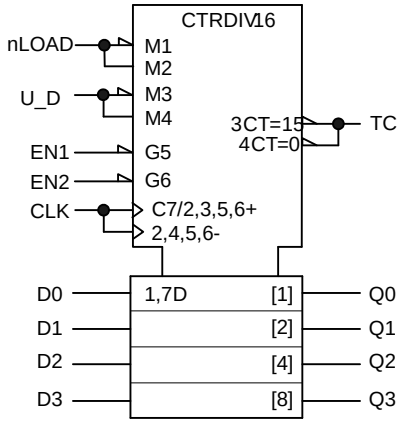
Der VHDL-Code verwendet zur Parametrierung der Bitbreite den `generic`-Parameter BITS, der auf vier Bit voreingestellt ist.:

```
entity AUFGABE_13_7 is
      generic( BITS: natural := 4);
      port(   CLK, RESET, NLOAD, ENT, ENP: in bit;
              D: in std_logic_vector (BITS-1 downto 0);
              Q: out std_logic_vector (BITS-1 downto 0);
              TC: out bit);
end AUFGABE_13_7;

architecture VERHALTEN of AUFGABE_13_7 is
signal QINT: std_logic_vector(BITS-1 downto 0);
begin
      -- Zaehlerprozess
CTR:  process (CLK, RESET)
      begin
              if RESET ='0' then
                      QINT <= (others =>'0');
              elsif CLK='1' and CLK'event then
                      if NLOAD = '0' then
                              QINT <= D;
                      elsif ENT='1' and ENP='1' then
                              QINT <= QINT + 1;
                      end if;
              end if;
      end process CTR;
      -- Overflow nebenlaeufig
      TC <= '1' when (QINT = 2**BITS-1 and ENT='1') else '0';
      -- Ausgangssignalzuweisung
```

```
      Q <= QINT;
end VERHALTEN;
```

A Aufgabe 13.9

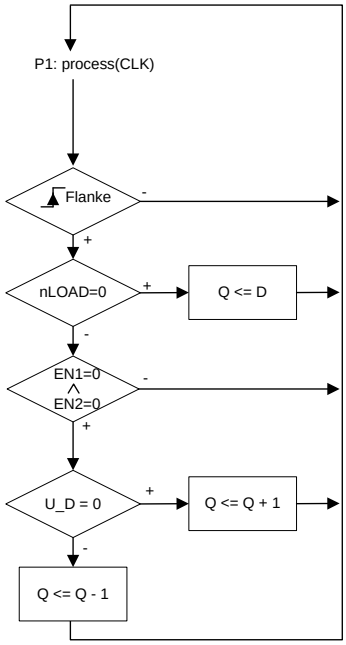
Das Schaltsymbol mit einer Bezeichnung der Ein- und Ausgänge zeigt das nachfolgende Bild:

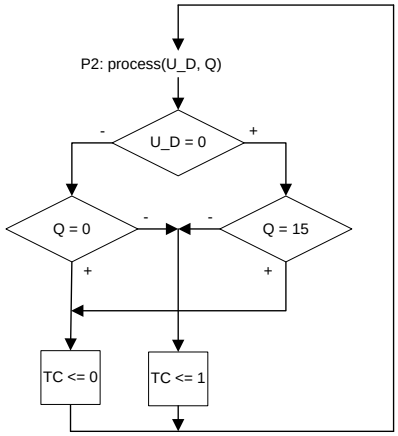


Dem Schaltsymbol sind die folgenden Funktionen zu entnehmen:

- M1: Betriebsart Taktsynchrones Laden (1,7D), falls nLOAD = 0
- M2: Betriebsart Taktsynchrones Zählen, falls nLOAD = 1
- M3: Betriebsart Vorwärts-Zählen, falls U_D = 0
- M4: Betriebsart Rückwärts-Zählen, falls U_D = 1
- G5: L-aktiver Freigabeeingang EN1 (Vorwärtszählen)
- G6: L-aktiver Freigabeeingang EN2 (Rückwärtszählen)
- C7: Kennzeichnung des Signals CLK als Taktsignal
- 3CT = 15: L-aktiver Übertrag TC im Vorwärtszählbetrieb falls Zählerstand = 15
- 4CT = 0: L-aktiver Übertrag TC im Rückwärtszählbetrieb falls Zählerstand = 0

Die Flussdiagramme für den getakteten, sowie den ungetakteten Prozess sind:



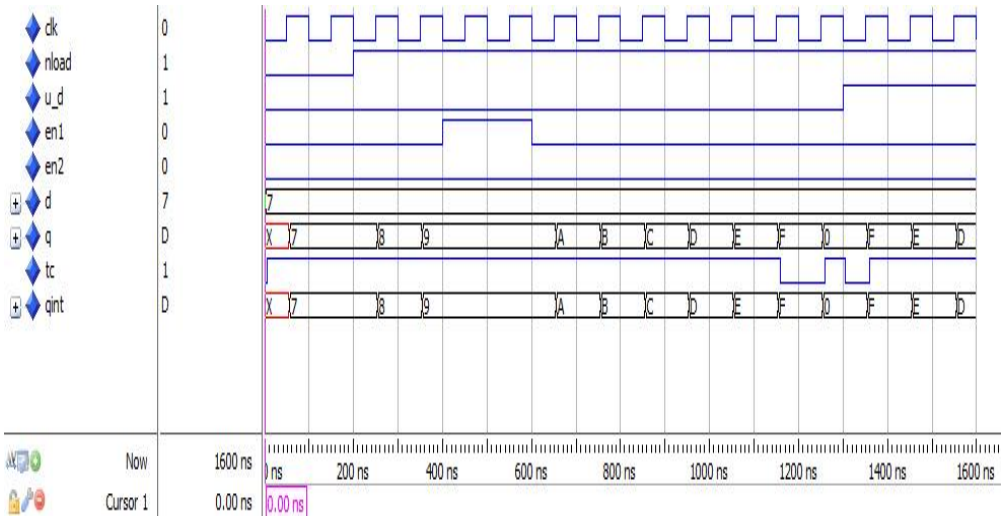


Im nachfolgenden VHDL-Code sind diese Flussdiagramme abgebildet:

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

entity AUFGABE_13_9 is
  port(CLK, NLOAD, U_D, EN1, EN2: in bit;
       D: in unsigned(3 downto 0);
       Q: out unsigned(3 downto 0);
       TC: out bit);
end AUFGABE_13_9;

architecture VERHALTEN of AUFGABE_13_9 is

signal QINT: unsigned(3 downto 0);

begin
P1: process (CLK) -- getakteter Prozess
    begin
          if CLK='1' and CLK'event then
             if NLOAD = '0' then
                QINT <= D after 5 ns;
        elsif (EN1='0' and EN2='0') then
           if U_D='0' then
              QINT <= QINT + 1 after 5 ns;
           else
                    QINT <= QINT - 1 after 5 ns;
```

```
               end if;
            end if;
         end if;
end process P1;
Q <= QINT;            -- Ausgabe des Binärwertes nebenlaeufig

P2: process(U_D, QINT)     -- ungetakteter Prozess
    begin
      if U_D = '0' then
         if QINT=15 then
            TC <= '0' after 5 ns; -- L aktiv
         else
            TC <= '1' after 5 ns;
         end if;
      else
         if QINT=0 then
            TC <= '0' after 5 ns; -- L aktiv
         else
            TC <= '1' after 5 ns;
         end if;
      end if;
end process P2;
end VERHALTEN;
```

Die Simulation bestätigt das erwartete Verhalten eines Vorwärts-/Rückwärtszählers:



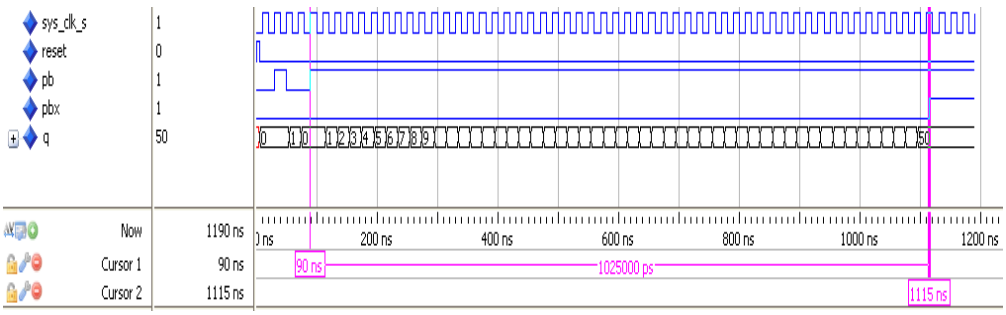
A Aufgabe 13.10

Die Idee der Entprellschaltung besteht darin, das manuelle Taktsignal PB (Push-Button) mit dem 50-MHz-Systemtakt SYS_CLK_S abzutasten. Nur wenn ein Zähler für mindestens 50 Abtastwerte (also für 1 µs) eine logische 1 erkannt hat, wird das Ausgangssignal PBX taktsynchron gesetzt. Wenn hingegen beim Abtasten eine logische 0 erkannt wird, so wird der Zähler zurückgesetzt. Zum Erreichen des Zahlenwertes 50 muss ein 6-Bit-Zähler verwendet werden. Zur Initialisierung des Zählers wird ein asynchroner Reset verwendet

```
library ieee;
```

```
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity AUFGABE_13_10 is
  port(       SYS_CLK_S, RESET, PB: in bit;
            PBX: out bit);
end AUFGABE_13_10;

architecture VERHALTEN of AUFGABE_13_10 is

signal Q: std_logic_vector (5 downto 0);

begin
P1: process (SYS_CLK_S, RESET) -- getakteter Prozess
    begin
         if RESET = '1' then Q <=(others=>'0') after 5 ns;
         elsif SYS_CLK_S='1' and SYS_CLK_S'event then
            if PB = '0' then                    -- Schalter hat geprellt
               Q <= (others=>'0') after 5 ns; -- loesche Zaehler
               PBX <= '0';                      -- manuelles Taktsignal
         elsif Q = 50 then                   -- falls ausreichend lange stabil
            PBX <= '1' after 5 ns;          -- setze manuelles Taktsignal
            else
               Q <= Q + 1 after 5 ns;          -- sonst zaehle weiter
            end if;
         end if;
end process P1;
end VERHALTEN;
```

Das nachfolgend dargestellte Simulationsergebnis zeigt, dass es etwas mehr als 1 µs nach dem letzten Prellen des Tasters dauert, bis das Ausgangssignal PBX gesetzt wird.



14 Schieberegister

A Aufgabe 14.1

a) Ein Schieberegister mit Parallel-Ausgangsregister.

b) Der Johnson-Zähler besitzt 32 Zustände, ein Binärzähler mit gleicher Anzahl von Zustandsbits hingegen 65536.

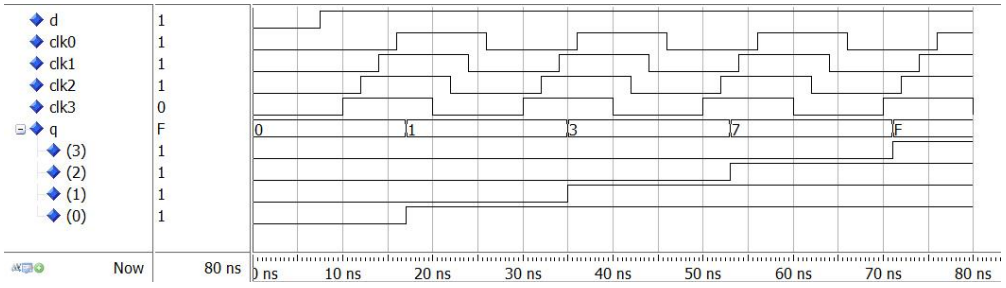
c) Das Übergangsschaltnetz ist sehr einfach aufgebaut womit ein Johnson-Zähler deutlich schneller getaktet werden kann als ein Binärzähler.

d) Vgl. Kap. 14.2: `Q <= SERIAL_IN & Q(MAXINDEX-1 downto 1);`

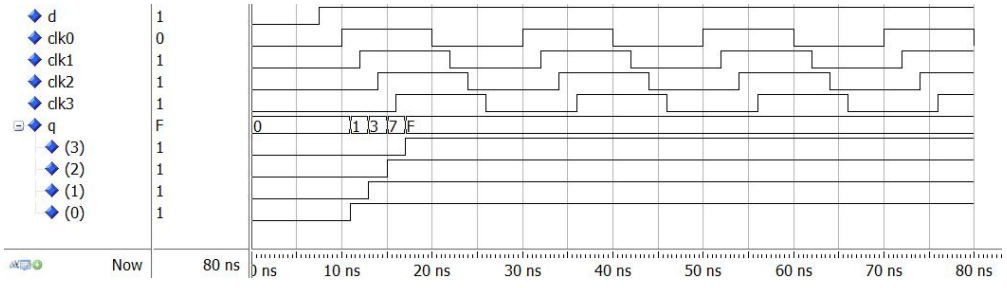
e) Diese Schaltungen besitzen ein Schieberegister, in das eine logische Linearkombination mehrerer Schieberegisterbits hineingetaktet wird.

f) Die nachfolgenden Bilder zeigen das Verhalten für negative und positive Taktverzögerung um jeweils ± 2ns bei einer Flipflopverzögerung t_{pd}(FF) von jeweils 1 ns:

Taktverzögerung negativ: Das Schieberegisterverhalten ist prinzipiell korrekt, wenn auch an den einzelnen Ausgängen leicht verzögert.



Taktverzögerung positiv: Das Schieberegisterverhalten ist fehlerhaft weil das Ergebnis 0xF schon mit der ersten Taktflanke erreicht wird. Grund dafür ist die Tatsache, dass das jeweils folgende Flipflop sein Taktsignal erst erhält, nachdem der Ausgang des davor befindlichen Flipflops seinen Wert bereits geändert hat. Die Flipflopkette wirkt damit quasi transparent.



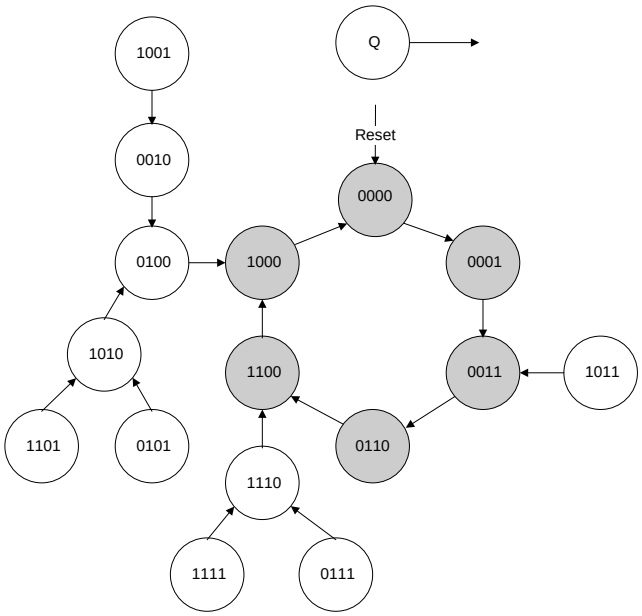
A Aufgabe 14.2

Die Zustandsfolgetabelle ist nachfolgend abgedruckt:

Q3	Q2	Q1	Q0	SERIAL_IN	$Q3^+$	$Q2^+$	$Q1^+$	$Q0^+$
0	0	0	0	1	0	0	0	1
0	0	0	1	1	0	0	1	1
0	0	1	0	0	0	1	0	0
0	0	1	1	0	0	1	1	0
0	1	0	0	0	1	0	0	0
0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	1	0	0
0	1	1	1	0	1	1	1	0
1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1	0
1	0	1	0	0	0	1	0	0
1	0	1	1	0	0	1	1	0
1	1	0	0	0	1	0	0	0
1	1	0	1	0	1	0	1	0
1	1	1	0	0	1	1	0	0
1	1	1	1	0	1	1	1	0

Aufgabenlösungen zu: Jürgen Reichardt, Lehrbuch Digitaltechnik 3. Auflage
ISBN 978-3-486-72765-4. Oldenbourg Wissenschaftsverlag, München 2013.

Daraus ergibt sich das folgende Zustandsdiagramm, in dem die zyklisch durchlaufenen Zählzustände grau markiert sind:

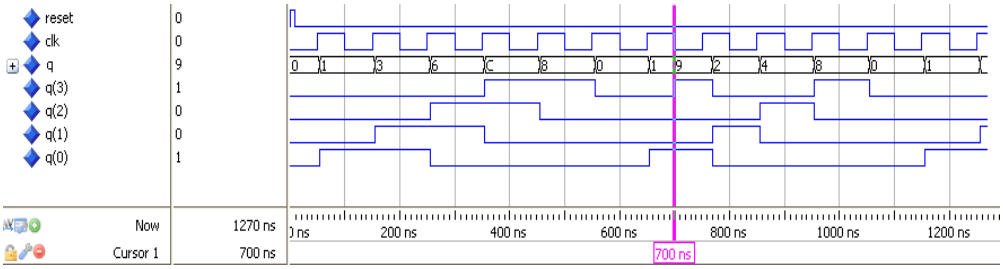


Der VHDL-Code ergibt sich in Anlehnung an Listing 14.3:

```
entity AUFGABE_14_2 is
   port( CLK, RESET : in bit;
         Q : out bit_vector(3 downto 0)
         );
end AUFGABE_14_2;
architecture VERHALTEN of AUFGABE_14_2 is
signal QINT: bit_vector(3 downto 0);
signal SERIAL_IN : bit;

begin
P1: process(CLK, RESET)
begin
   if RESET = '1' then
      QINT <= (others=>'0') after 5 ns;
   elsif CLK='1' and CLK'event then
      QINT <= QINT(2 downto 0) & SERIAL_IN after 5 ns;
   end if;
end process P1;

SERIAL_IN <= not (QINT(3) or QINT(2) or QINT(1));

Q <= QINT;   -- Kopie als Ausgangssignal
end VERHALTEN;
```

Mit der nachfolgenden ModelSim-Macro-Datei wird zunächst ein zyklischer Durchlauf und bei t = 700 ns eine Pseudozustandsstörung mit QINT = 9 simuliert. Das Simulationsergebnis zeigt, dass der Zähler nach vier Taktzyklen in den zyklischen Ablauf zurückkehrt.

```
restart
radix hex
add wave /aufgabe_14_2/reset
add wave /aufgabe_14_2/clk
add wave /aufgabe_14_2/q
add wave /aufgabe_14_2/q(3)
add wave /aufgabe_14_2/q(2)
add wave /aufgabe_14_2/q(1)
add wave /aufgabe_14_2/q(0)

force CLK 0 0 , 1 50ns -r 100ns
force RESET 1, 0 @10ns
run 700ns

# Hazard Simulation:
force qint 9
run 70ns
noforce qint
run 500 ns
```



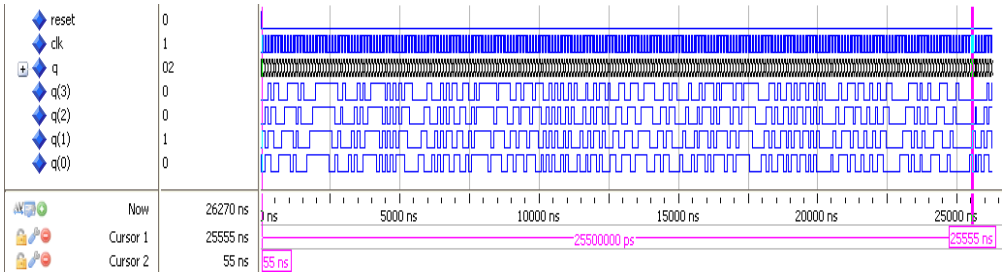
A Aufgabe 14.3

Der VHDL-Code mit Links-Schieberegister ergibt sich in Anlehnung an Listing 14.5 unter Berücksichtigung der Rückkopplungsgleichungen aus Tabelle 14.3 wie folgt:

```
entity AUFGABE_14_3 is
   port( CLK, RESET : in bit;
         Q : out bit_vector(7 downto 0)
         );
end AUFGABE_14_3;
architecture VERHALTEN of AUFGABE_14_3 is
signal QINT: bit_vector(7 downto 0);
signal SERIAL_IN : bit;

begin
P1: process(CLK, RESET)
begin
   if RESET = '1' then
      QINT <= x"01" after 5 ns;
   elsif CLK='1' and CLK'event then
      if QINT = x"00" then
         QINT <= x"01" after 5 ns;
      else
         QINT <= QINT(6 downto 0) & SERIAL_IN after 5 ns;
  end if;
```

```
   end if;
end process P1;

SERIAL_IN <= QINT(7) xor QINT(3) xor QINT(2) xor QINT(1);

Q <= QINT;    -- Kopie als Ausgangssignal
end VERHALTEN;
```

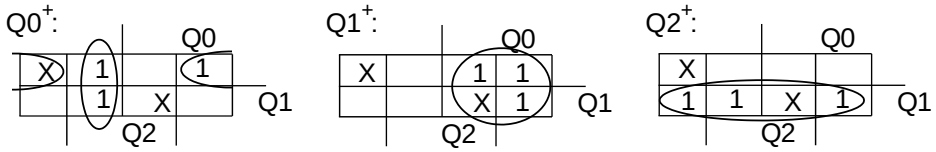
Die Simulation einer kompletten Zufallszahlensequenz zeigt mit den Markierungen, dass sich z.B. die Pseudozufallszahl 0x2 nach 255 Takten wiederholt.



A Aufgabe 14.4

Die gewünschte Bitfolge am Ausgang besteht aus sechs Elementen. Die Folge lässt sich also mit einem 3-Bit-Schieberegister Q2, Q1, Q0 erreichen. Nachfolgend wird eine Zustandsfolgetabelle so konstruiert, dass zunächst die gewünschte Bitfolge als Q2-Bit dargestellt wird. Unter Berücksichtigung der Funktionalität des Links-Schieberegisters können nun zunächst in den ersten zwei Zeilen das Q1-Bit und nachfolgend das Q0-Bit der ersten Zeile ergänzt werden. Entsprechend verfährt man nun mit vier weiteren Zeilen. Nachfolgend müssen in der Folgezustandsspalte die Werte der Zustandsbits der jeweils nachfolgenden Zeile eingetragen werden. Da es sich um ein periodisches Signalmuster handelt, müssen in der 6. Zeile als Folgezustandsbits die Zustandsbits der ersten Zeile eingetragen werden. Da die Ausgangsbitfolge nur aus sechs Bit besteht, können den letzten beiden Zeilen der Tabelle Don't-Care-Folgezustände zugeordnet werden. In dieser Tabelle sind die Zeilen, also die Minterme, natürlich nicht in der gewohnten aufsteigenden Reihenfolge angegeben daher wird links eine Mintermspalte ergänzt, die für die Eintragung in die KV-Diagramme benötigt wird.

Minterm	Q2 Q1 Q0	$Q2^+$ $Q1^+$ $Q0^+$
m_2	0 1 0	1 0 0
m_4	1 0 0	0 0 1
m_1	0 0 1	0 1 1
m_3	0 1 1	1 1 0
m_6	1 1 0	1 0 1
m_5	1 0 1	0 1 0
m_0	0 0 0	X X X
m_7	1 1 1	X X X

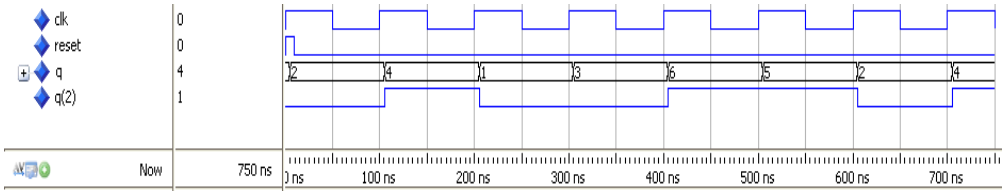


Die KV-Minimierung der Folgezustandsbits ergibt die folgenden logischen Gleichungen:

$Q0^+ = (\overline{Q0} \wedge Q2) \vee (\overline{Q1} \wedge \overline{Q2})$; $Q1^+ = Q0$; $Q2^+ = Q1$

Diese logischen Gleichungen werden im VHDL-Prozess verwendet. Die Simulation zeigt die gewünschte periodische Ausgangssignalfolge 010011 des Zustandsbits Q2.

```
entity AUFGABE_14_4 is
   port( CLK, RESET : in bit;
         Q : out bit_vector(2 downto 0));
end AUFGABE_14_4;

architecture VERHALTEN of AUFGABE_14_4 is
signal Q_INT: bit_vector(2 downto 0);
begin
P1: process(CLK, RESET)
begin
   if RESET ='1' then
     Q_INT <= "010" after 5 ns;     -- Reset Wert
   elsif CLK='1' and CLK'event then -- ansteigende Signalflanke
     Q_INT(0) <= (not Q_INT(0) and Q_INT(2)) or
                 (not Q_INT(1) and not Q_INT(2))after 5 ns; -- Linear Feed-
back
     Q_INT(1) <= Q_INT(0) after 5 ns; -- SRG
     Q_INT(2) <= Q_INT(1) after 5 ns; -- SRG
   end if;
end process P1;
Q <= Q_INT;
end VERHALTEN;
```



A Aufgabe 14.5

Eine von zwei zyklische Zustandsfolgen aus acht Zuständen, die die Anfordeung erfüllt ist (vgl. Bild 14.16):

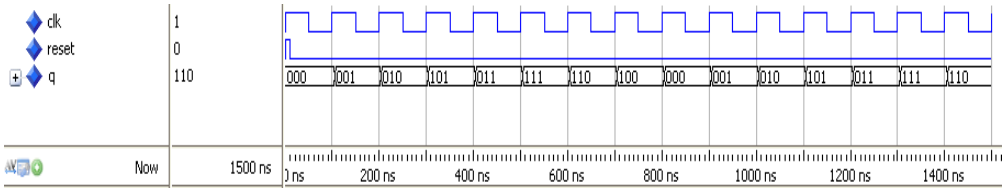
000, 001, 010, 101, 011, 111, 110, 100, …

Als Lösung wird zunächst die linke Seite einer Folgezustandstabelle konstruiert, wobei ähnlich wie in Aufgabe 14.4 zu berücksichtigen ist, dass die Minterme nicht in aufsteigender Reihenfolge angegeben sind. Anschließend muss der Wert des Schiebeeingangsbits SE = $Q0^+$ so angegeben werden, dass sich der jeweils folgende Zustand ergibt:

Minterm	Q2 Q1 Q0	SE = $Q0^+$
m_0	0 0 0	1
m_1	0 0 1	0
m_2	0 1 0	1
m_5	1 0 1	1
m_3	0 1 1	1
m_7	1 1 1	0
m_6	1 1 0	0
m_4	1 0 0	0

Die KV-Minimierung liefert $SE = Q0^+ = (\overline{Q0} \wedge \overline{Q2}) \vee (Q1 \wedge \overline{Q2}) \vee (Q0 \wedge \overline{Q1} \wedge Q2)$

Die Verifikation erfolgt durch eine VHDL-Simulation in Anlehnung an Aufgabe 14.5 mit einer angepassten Übergangsschaltnetzfunktion:



A Aufgabe 14.6

Grundlage für die Implementierung des Produktes P = A · B ist der nachfolgende Pseudocode:

```
while (S = 0) { }; -- warte auf Startsignal
P=0;
load A;
load B;
while B != 0 do {
   if B(0)=1 then
      P = P + A;
   end if;
   A = A << 1;
   B = B >> 1;
}
```

Die Elemente des Datenpfads (vgl. Bild 10.1) sind demnach:

- ein Ausgangsregister P mit doppelter Bitbreite NN
- ein ladbares Links-Schieberegister
- ein ladbares Rechts-Schieberegister
- ein Addierer für doppelte Bitbreite
- ein Multiplexer mit dem entschieden wird, ob das Ausgangsregister P mit Nullen initialisiert werden soll, oder ob die Partialprodukte addiert werden sollen.
- ein Komparator, der eine NULL-Erkennung durchführt

In dem nachfolgend vorgestellten VHDL-Code erfolgt jede Iteration der `while`-Schleife sowie die beiden Schiebeoperationen taktsynchron. Der Addierer und der Multiplexer arbeiten hingegen kombinatorisch.

Hinzu kommt ein Steuerwerk (vgl. Bild 10.1), in dem die Steuerung der Datenpfadkomponenten erfolgt:

- Eingangssignale des Automaten (Statussignale):
  - S: Startsignal für eine neue Berechnung
  - LA: Laden eines neuen Operanden A
  - LB: Laden eines neuen Operanden B
  - B_INT(0): Das jeweils niederwerige Bit des B-Operanden
  - Z: der geschobene B-Operand enthält nur Nullen
- Ausgangssignale des Automaten (Steuersignale):
  - EA: Freigabe zum Laden oder Schieben des A-Operanden
  - EB: Freigabe zum Laden oder Schieben des B-Operanden
  - EP: Freigabe zum Laden des Ergebnisregisters
  - PSEL: Steuerung des Multiplexers zum Initialisieren des Ergebnisregisters
  - DONE: Quittierungssignal zum Erkennen des Berechnungsendes
- Automatenzustände:
  - S1: Laden der A- und B-Operanden, Löschen des Ergebnisregisters
  - S2: Akkumulation der Partialprodukte, Schieben der A- und B-Operanden
  - S3: Beendigung der Berechnung, Setzen des Quittierungssignals

Aufgabenlösungen zu: Jürgen Reichardt, Lehrbuch Digitaltechnik 3. Auflage
ISBN 978-3-486-72765-4. Oldenbourg Wissenschaftsverlag, München 2013.

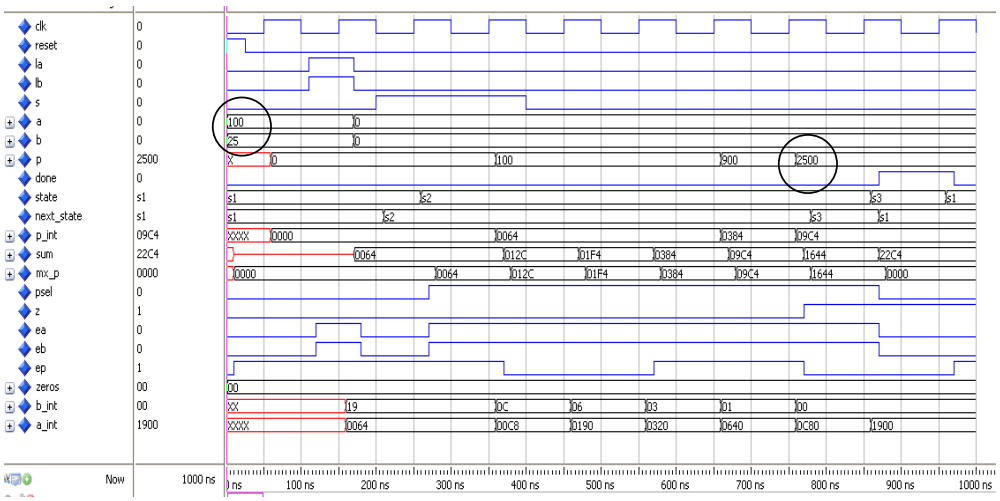
Der VHDL-Code für Operanden der Bitbreite N ist nachfolgend abgedruckt:

```
-- NxN serieller Multiplizierer
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity AUFGABE_14_6 is
generic( N:integer:=8; NN:integer:=16); -- Bitbreite, doppelte Breite
port( CLK, RESET     : in bit;
      LA, LB, S: in bit;                    -- Ladesignale, Startsignal
      A,B: in std_logic_vector(N-1 downto 0); -- N-Bit Operanden
      P: out std_logic_vector(NN-1 downto 0); -- Ergebnis, doppelte Breite
      DONE: out bit);                       -- Statussignal
end AUFGABE_14_6;
architecture VERHALTEN of AUFGABE_14_6  is
type STATE_TYPE is (S1, S2, S3);            -- Zustandstyp
signal STATE, NEXT_STATE: STATE_TYPE;       -- Zustandssignale
signal P_INT, SUM, MX_P: std_logic_vector(NN-1 downto 0); -- Datenpfadsig
                                                          nale
signal PSEL, Z, EA, EB, EP: bit;            -- Steuer- und Statussignale
signal ZEROS: std_logic_vector(N-1 downto 0);
signal B_INT: std_logic_vector(N-1 downto 0); -- internes Schieberegister
signal A_INT: std_logic_vector(NN-1 downto 0);-- internes Schieberegister
begin
-- ======================== Datenpfad ==========================
ZEROS <= (others =>'0');
SLREG: process(CLK)                         -- Links-Schieberegister
begin
      if CLK='1'and CLK'event then
            if EA ='1' then
                  if LA = '1' then          -- synchrones Laden
                        A_INT <= ZEROS & A after 10 ns;
                  else                      -- Links schieben
                        A_INT <= A_INT(NN-2 downto 0) & '0' after 10 ns;
                  end if;
            end if;
      end if;
end process SLREG;
SRREG: process(CLK)                         -- Rechts-Schieberegister
begin
      if CLK='1'and CLK'event then
            if EA ='1' then
                  if LB = '1' then          -- synchrones Laden
                        B_INT <= B after 10 ns;
                  else                      -- Rechts schieben
                        B_INT <= '0' & B_INT(N-1 downto 1) after 10 ns;
                  end if;
            end if;
      end if;
end process SRREG;

MUX: process(SUM, PSEL)                     -- Multiplexer
begin
      if PSEL = '0' then                    -- Initialisierung
            MX_P <= (others=>'0') after 10 ns;
      else
            MX_P <= SUM after 10 ns;
      end if;
end process MUX;
```

```
P_REG: process(CLK)                          -- Ausgangsregister
begin
      if CLK='1' and CLK'event then
            if EP='1' then
                  P_INT <= MX_P after 10 ns;
            end if;
      end if;
end process P_REG;

NULL_ERK: process(B_INT, ZEROS)              -- Null Erkennung
begin
      if B_INT = ZEROS then
            Z <= '1' after 10 ns;
      else
            Z <= '0' after 10 ns;
      end if;
end process NULL_ERK;

-- nebenlaeufige Anweisungen:
SUM <= A_INT + P_INT after 10 ns;            -- Addierer
P <= P_INT;                                  -- Vermeidung des buffer-ports

-- ================ Steuerpfad ========================
FSM_REGS: process(CLK, RESET)                        -- Zustandsregister
begin
      if RESET = '1' then
            STATE <= S1 after 10 ns;
      elsif CLK='1' and CLK'event then
            STATE<= NEXT_STATE after 10 ns;
      end if;
end process FSM_REGS;

FSM_TRANS: process(STATE, S, Z, LA, LB, B_INT(0))
begin
   EA<='0' after 10 ns;      --Defaults Outputs zur Vermeidung von Latches
   EB<='0' after 10 ns;
   EP<='0' after 10 ns;
   DONE<='0' after 10 ns;
   PSEL<='0' after 10 ns;
     case STATE is                           --Zustandsabfrage
      when S1 =>
            EP <='1' after 10 ns;          --Moore-Ausgaenge
            if S='0' then
               NEXT_STATE <= S1 after 10 ns;
               if LA='1' then
                    EA <= '1' after 10 ns;
                end if;                     --Mealy Signal
               if LB='1' then
                    EB <= '1' after 10 ns;
                end if;                     --Mealy Signal
            else
               NEXT_STATE <= S2 after 10 ns;
            end if;
      when S2 =>
            EA <= '1' after 10 ns;
            EB <= '1' after 10 ns;
            PSEL <= '1' after 10 ns;
            if Z='0' then
               NEXT_STATE <= S2 after 10 ns;
               if B_INT(0) = '1' then
```

```
                    EP <= '1' after 10 ns;
                end if;
            else
                NEXT_STATE <= S3 after 10 ns;
            end if;
        when S3 =>
            DONE <= '1' after 10 ns;
            if S='1' then
                NEXT_STATE <= S3 after 10 ns;
            else
                NEXT_STATE <= S1 after 10 ns;
            end if;
    end case;
end process FSM_TRANS;
end VERHALTEN;
```

Die Verifikation erfolgt exemplarisch über die Simulation von $100_{10} \cdot 25_{10} = 2500_{10}$. Diese zeigt die einzelnen Datenpfad-, Status- und Steuersignale.



A Aufgabe 14.7

Die Lösungsidee besteht darin, das Eingangssignal mit dem Systemtakt SYS_CLK_S abzutasten und die Abtastwerte in ein Schieberegister ausreichender Tiefe zu laden. Wenn dieses ausschließlich Einsen enthält, so hat der Schalter aufgehört zu prellen.

Bei einem Systemtakt von 50 MHz muss das Schieberegister aus mindestens 50 D-Flipflops bestehen, wenn sichergestellt ist, dass das Prellen nicht länger als 1 µs dauert. Das Schieberegister wird im taktsynchronen Prozess P1 modelliert und der Komparatorprozess P2 vergleicht den Inhalt des Schieberegsters kombinatorisch mit der Konstante ALL_ONES:

```
entity AUFGABE_14_7 is
port( SYS_CLK_S      : in bit;
      PB: in bit;                          -- Manuelles Signal vom Taster
      PBX: out bit);                       -- entprelltes Tastersignal
end AUFGABE_14_7;

architecture VERHALTEN of AUFGABE_14_7  is
constant ALL_ONES: bit_vector(49 downto 0) := (others=>'1');
signal SRG: bit_vector(49 downto 0);       -- internes Schieberegister

begin
P1: process(SYS_CLK_S)                      -- Rechts-Schieberegister
begin
   if SYS_CLK_S='1'and SYS_CLK_S'event then
      SRG <= PB & SRG(49 downto 1) after 2 ns;
   end if;
end process P1;

P2: process(SRG)                    -- Komparator kombinatorisch
begin
   if SRG = ALL_ONES then
      PBX <= '1' after 2 ns;
   else
      PBX <= '0' after 2 ns;
   end if;
end process P2;
end VERHALTEN;
```

Für diese Schieberegisterlösung werden 50 D-Flipflops sowie ein 50-fach-UND-Gatter benötigt. Im Vergleich dazu erfordert die Zählerlösung aus der Aufgabe 13.10 nur 6 D-Flipflops sowie einen 6-Bit-Addierer.

15 Synchronisation digitaler Systeme

A Aufgabe 15.1

a) Vgl. die einführenden Texte in Kap. 15.2.1 und Kap.

b) Vgl. die Beschreibung unter Bild 15.5. Voraussetzung für eine korrekte Funktion der Schaltung ist, dass das asynchrone Eingangssignal mindestens für eine Taktperiode anliegt (Vgl. den Merksatz in Kap. 15.3.1).

c) Beim synchronen Datenaustausch arbeiten Sender und Empfänger mit der gleichen Taktquelle, beim asynchronen Datenaustausch sind hingegen die Taktquellen von Sender und Empfänger unkorreliert.

d) Vgl. die Beschreibung unter Bild 15.10.

A Aufgabe 15.2

Die Gl. 15.1 wird zunächst nach t_R aufgelöst:

$$t_R = \tau \cdot \ln(MTBF \cdot T_0 \cdot f_{Clk} \cdot f_{Data})$$

Nach Einsetzen der angegebenen Takt- und Datenfrequenzen f_{Clk}=100 MHz und f_{Data} = 1 MHz bzw. MTBF = 10^4 Jahre = $3.1536 \cdot 10^{11}$ s sowie der Flipflop Parameter τ = 0.17 ns und $T_0 = 9.6 \cdot 10^{-18}$ s erhält man eine Erholungszeit t_R = 3.32 ns.

Die zur Verfügung stehende Periodendauer ist $T_{Clk} = 1/f_{Clk}$ ist 10 ns.

Diese teilt sich auf in $T_{Clk} = t_R + t_{ÜSN} + t_S$.

Daraus erhält man mit t_S = 4 ns für die maximal erlaubte Verzögerung im Übergangsschaltnetz $t_{ÜSN}$ = 2.68 ns.

A Aufgabe 15.3

Die einzuhaltende Zeit zwischen zwei Synchronisationsfehlern ist MTBF = 10^6 Jahre = $3.1536 \cdot 10^{13}$s
Die Gl. 15.1 wird nun nach τ aufgelöst:

$$\tau = \frac{t_R}{\ln(MTBF \cdot T_0 \cdot f_{Clk} \cdot f_{Data})}$$

Als Erholungszeit steht die Periodendauer des Systemtakts abzüglich der Setup-Zeit zur Verfügung: t_R = (2.5 -0.1) ns = 2.4 ns. Einsetzen der MTBF sowie der Parameter $f_{Clk} = 400 \cdot 10^6$ Hz, $f_{Data} = 100 \cdot 10^6$ Hz und T0= $1 \cdot 10^{-18}$ s erfordert, dass für die einzusetzende Technologie $\tau < 8.6 \cdot 10^{-2}$ ns spezifiziert sein muss.

Aufgabenlösungen zu: Jürgen Reichardt, Lehrbuch Digitaltechnik 3. Auflage
ISBN 978-3-486-72765-4. Oldenbourg Wissenschaftsverlag, München 2013.

A Aufgabe 15.4

In dem nachfolgend dargestellten VHDL-Code zu Bild 15.6 wird zusätzlich zur üblichen Modellierung von D-Flipflos die Setup-Zeit überprüft: Falls sich das Eingangssignal dieser Flipflops innerhalb einer Zeitspanne von 5 ns vor der steigenden Flanke ändert, wird der Flipflop-Ausgang undefiniert 'U'. Dies erfordert den Datentyp std_ulogic.

```
-- VHDL-Modell eines DFF mit Setup-Time Check
-- Synchronisations Schaltung fuer kurze Pulse (3 DFFs)
library ieee;
use ieee.std_logic_1164.all;

entity Aufgabe_15_4 is
port(  CLK, INP : in std_logic;
       QOUT: out std_logic);
end Aufgabe_15_4;

architecture VERHALTEN of Aufgabe_15_4 is
signal Q_INT1, Q_INT2, Q: std_logic :='0'; --init all FFs
begin

DFF1: process(INP, Q)
begin
       if (Q = '1' and INP = '0')  then --reset by sync. output and E=0
              Q_INT1 <= '0' after 10 ns;
       elsif INP='1' and INP'event then
              Q_INT1 <= '1' after 10 ns;
       end if;
end process DFF1;

DFF2: process(CLK)
begin
       if CLK='1' and CLK'event then
              if Q_INT1'stable(5 ns) then
                     Q_INT2 <= Q_INT1 after 10 ns;
              else
                     Q_INT2 <= 'U' after 1 ns;
              end if;
       end if;
end process DFF2;

DFF3: process(CLK)
begin
       if CLK='1' and CLK'event then
              if Q_INT2'stable(5 ns) then
                     Q <= Q_INT2 after 10 ns;
              else
                     Q <= 'U' after 1 ns;
              end if;
       end if;
end process DFF3;

QOUT <= Q;
end VERHALTEN;
```

16 Digitale Halbleiterspeicher

A Aufgabe 16.1

a) RAMs können genau so schnell und beleibig häufig gelesen wie beschrieben werden; ROMs können nur gelesen werden, die Programmierung erfolgt bei der Herstellung; PROMs können gelesen werden, eine Programmierung ist deutlich langsamer und nur mit endlicher Häufigkeit möglich.

b) Die Tatsache, dass die ROM-Werte beim Chip-Hersteller durch Definition der Verdrahtungsmasken definiert wird.

c) Vgl. die Beschreibung zu Bild 16.5.

e) Vgl. die jewieligen Merksätze in Kap. 16.3.3 und Kap. 16.3.4

e) Mit einer Konstantendefinition für ein Feld (Array) sowie einem indizierten Adresszugriff auf ein Element dieses Feldes (vgl. Listing 16.1).

f) In einer SRAM-Speicherzelle wird die Bit-Information in einem rückgekoppelten CMOS-Inverterpaar abgelegt (vgl. Bild 16.6), in einer DRAM-Speicherzelle wird die Bit-Information hingegen in einem Kondensator (vgl. Bild 16.9).

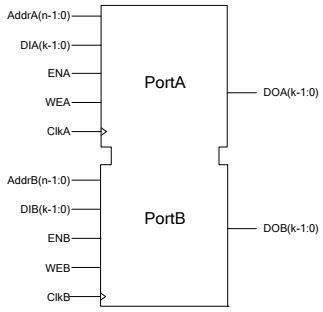
g) Die Adressen und Eingangsdaten des Speichers müssen synchron eingelesen und die Ausgangsdaten ebenfalls über Flipflops geführt werden.

h) Die Adressansteuerung erfolgt über die Spalten- und Zeilenadresssignale CAS und RAS (vgl. Bild 16.10).

i) Bei einem Dual-Port-RAM gibt es zwei Ports zur Ansteuerung der RAM-Inhalte. Jeder Port besitzt einen eigenen Adress- und Dateneingang sowie einen Datenausgang, sowie meist auch einen eigenen Takteingang (vgl. Bild 16.13).

j) Der Betrieb eines DDR-RAMs erfordert spezielle Flipflop-Schaltungen, die die Datenübernahme mit beiden Taktflanken erlauben.

k)

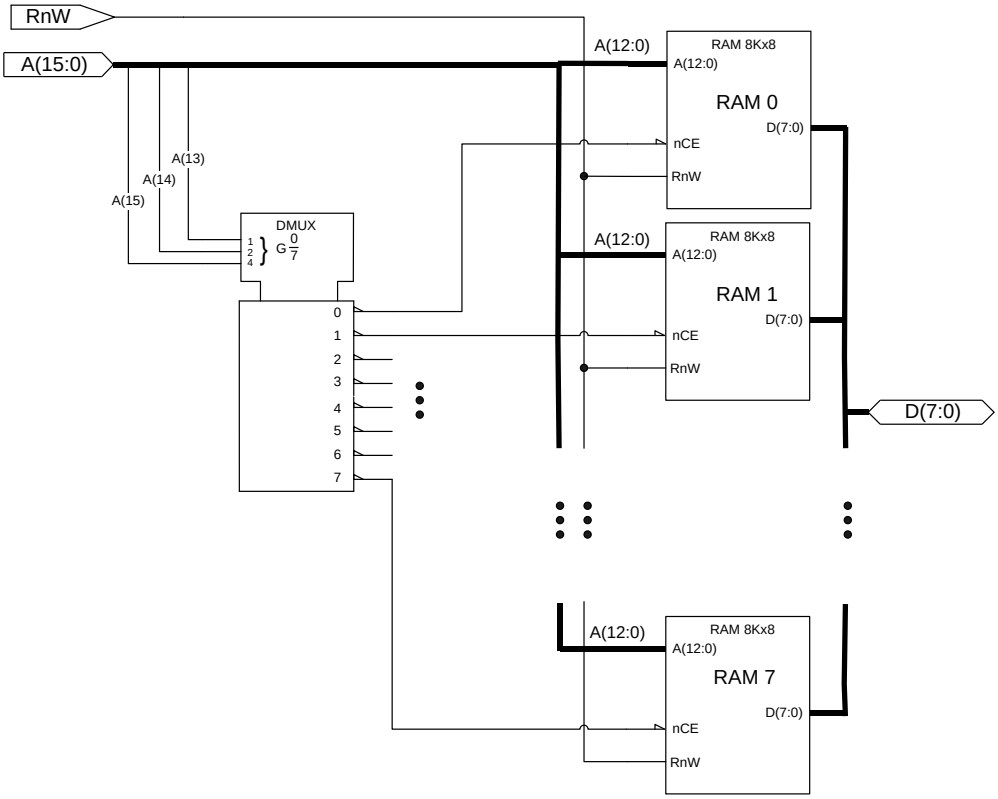


l) Zusätzlich zum FPGA-Block-RAM werden benötigt (vgl. Bild 16.18): Zwei (getaktete) Adresszähler sowie ein kombinatorischer Logikblock, der die Adressen zur Analyse des Füllzustands auswertet.

A Aufgabe 16.2

a) Es wird ein 3-zu-8-Demultiplexer benötigt, dessen Steuereingänge mit den Adressleitungen A(15) ... A(13) belegt werden und der die (in der Regel Low-aktiven) Freigabesignale für die RAM-Speicherbausteine erzeugt.

b) Die Schaltung zeigt das nachfolgende Bild:



c) 0xAFFE = 1010 1111 1111 1111 1111

Die obersten drei Bit 101 werden zur Auswahl des RAM-Bausteins verwendet. Daher befindet sich die Adresse im RAM 5.

A Aufgabe 16.3

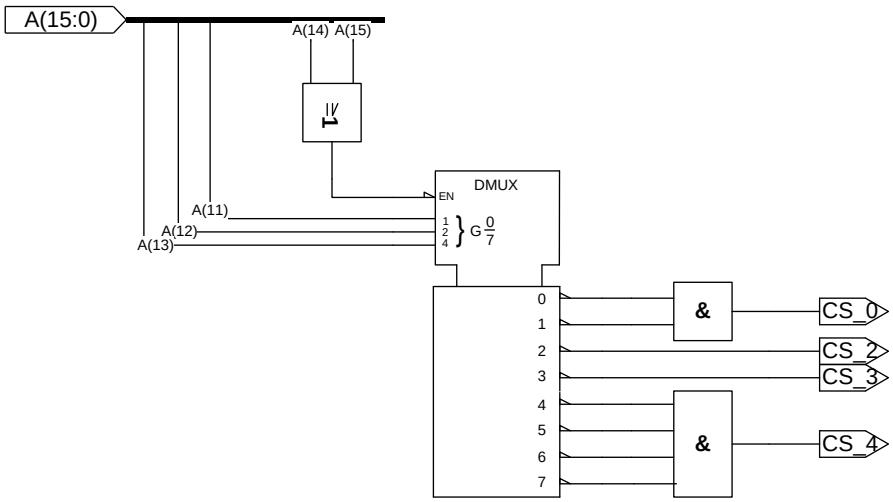
a) Die Speicheraufteilung (Memory-Map) ist nachfolgend dargestellt. Die jeweiligen Speicherauswahlsignale (Chip-Select CS_i) sind mit angegeben:

0x0000 0x0FFF	**4 kB-ROM** **(CS_0)**
0x1000	**2 kB-RAM**

0x17FF	**(CS_2)**
0x1800 0x1FFF	**2 kB-RAM** **(CS_3)**
0x2000 0x3FFF	**8 kB-RAM** **(CS_4)**
0x4000 0x7FFF	**nicht** **verwendet**

b) Einer der Speicherbausteine soll nur angesprochen werden wenn A(15) und A(14) Null sind. Als Freigabesignal dient daher die ODER-Verknüpfung dieser Adresssignale. Als Selektionseingang für den Decoder dienen die Adresssignale A(13), A(12) und A(11).

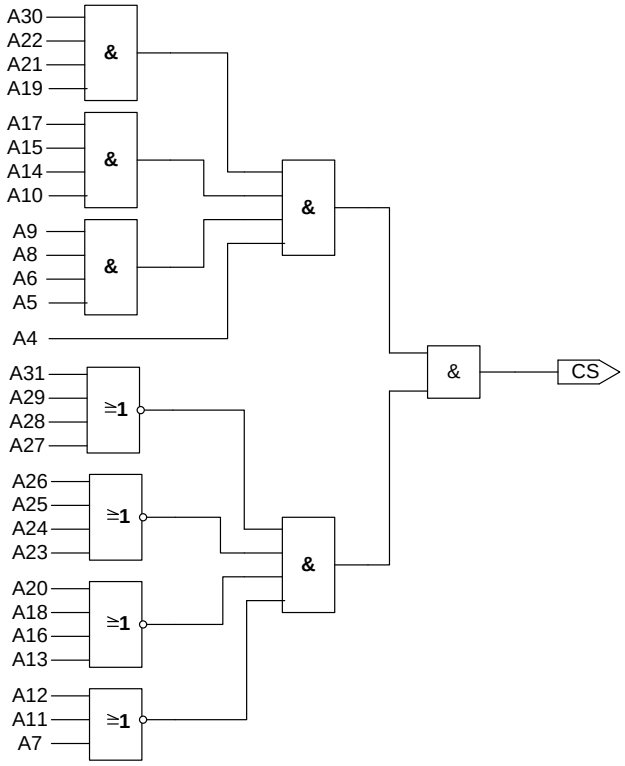
c) Das CS_0-Signal soll L-aktiv werden, wenn einer der Dekoderausgänge DX_0 **oder** der Ausgang DX_1 Null wird. Da es sich hier um L-aktive Signale handelt, wird ein UND-Gatter benötigt. Entsprechendes gilt für die Beschaltung von CS_4. Hier müssen allerdings vier der Decoderausgänge UND-verknüpft werden.



A Aufgabe 16.4

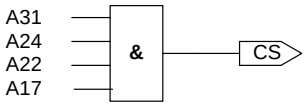
a) Die 32-Bit-Basisadresse ist mit 0x406AC770 vorgegeben. Dies entspricht der dualen Adresse (A31 downto 0): 0100 0000 0110 1010 1100 0111 0111 0000

Die darin enthaltenen Einsen müssen zu einer Decodierlogik UND-verknüpft werden. Zusätzlich müssen alle Nullen der Adressleitungen A31 … A7 NOR-verknüpft werden. Die niederwertigen vier Bit werden nicht decodiert, da sie zum Adressraum des Ethernet-Interfaces gehören. Da nur Gatter mit maximal vier Eingängen zur Verfügung stehen, ergibt sich eine 3-stufige Logik:



b) Sofern nicht zu viele Peripheriegeräte angeschlossen werden müssen, ist es günstiger den Adressraum des Ethernet-Interfaces auf z.B. 64 kB zu vergrößern (obwohl darin weiterhin nur 16 Register benötigt werden). Dadurch müssen die niederwertigen 16 Bit nicht decodiert werden. In diesem Fall sind nur 16 Adresseingänge zusammenzufassen, was mit einer 2-stufigen und damit schnelleren Logik möglich ist.

Mit dem Konzept einer nichtvollständigen Adressdecodierung kann sogar eine einstufige Logik erreicht werden. So würde z.B. die Basisadresse 0x81420000 mit der nachfolgenden Logik decodiert werden:



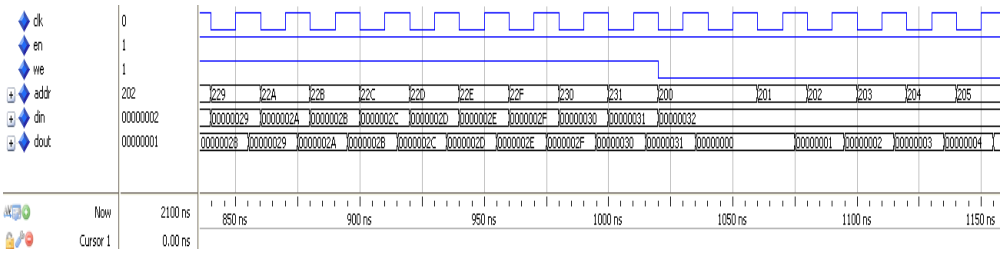
A Aufgabe 16.5

In dem Testbenchprozess werden, beginnend ab Adresse 0x200 zunächst 50 32-Bit-Daten geschrieben: Beginnend bei Null werden die Daten bei jeder Adresse inkrementiert. Anschliessend werden diese Daten in einer Schleife wieder sukzessive ausgelesen und diese mit dem Laufindex der Schleife verglichen. Fehler werden durch eine `assert`-Anweisung angezeigt. Die Simulation zeigt den Übergang vom Schreiben der letzten Daten zum Lesen der ersten Daten. Die gesamte Testbench ist mit den Pragmas `synthesis off/on` von der Synthese ausgeschlossen.

```
-- Aufgabe 15.5
```

```
-- 1kx32 BRAM-Speicher mit testbench

-- synthesis off
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity AUFGABE_15_5_TB is
end AUFGABE_15_5_TB;

architecture TESTBENCH of AUFGABE_15_5_TB is
component AUFGABE_15_5
port( CLK  : in bit;
      EN   : in bit;
      WE   : in bit;
      ADDR : in std_logic_vector( 11 downto 0);
         DIN  : in std_logic_vector(31 downto 0);
         DOUT : out std_logic_vector(31 downto 0));
end component;
signal CLK, EN, WE : bit;
signal ADDR : std_logic_vector(11 downto 0);
signal DIN, DOUT : std_logic_vector(31 downto 0);

begin
PCLK: process
begin
   CLK <= '0'; wait for 10 ns;
   CLK <= '1'; wait for 10 ns;
end process PCLK;

PADDR: process
begin
 wait for 20 ns; -- warte auf erste fallende Flanke

 EN <= '1';
 WE <= '1';      -- schreibe 50 Zahlen
 ADDR <= x"200";
 DIN <= x"00000000";
 for I in 0 to 49 loop
   wait for 20 ns; --warte eine Periode
   DIN <= DIN + 1; --weise neue Werte zu
   ADDR <= ADDR + 1;
 end loop;

 WE <= '0';      -- lese 50 Zahlen
 ADDR <= x"200"; -- setze Adresse zurück
 wait for 20 ns;

 for I in 0 to 49 loop
   wait for 20 ns; --warte eine Periode
   -- prüfe die Korrektheit der Daten
   assert conv_integer(DOUT) = I report "Error reading data";
   ADDR <= ADDR + 1;
 end loop;
end process PADDR;

U1: AUFGABE_15_5   -- BRAM Komponenteninstanz
   port map(CLK, EN, WE, ADDR, DIN, DOUT);
end TESTBENCH;
```

```
-- synthesis on
-----------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity AUFGABE_15_5 is
port( CLK  : in bit;
      EN   : in bit;
      WE   : in bit;
      ADDR : in std_logic_vector( 11 downto 0);
      DIN  : in std_logic_vector(31 downto 0);
      DOUT : out std_logic_vector(31 downto 0));
end AUFGABE_15_5;

architecture VERHALTEN of AUFGABE_15_5  is
type BRAM_TYPE is array(0 to 1023) of std_logic_vector(31 downto 0);
signal BRAM : BRAM_TYPE;

begin
P1: process(CLK)              -- BRAM im write-first Modus
begin
   if CLK='1'and CLK'event then
      if EN ='1' then
         if WE ='1' then
            BRAM(conv_integer(ADDR)) <= DIN after 5 ns;
            DOUT <= DIN after 5 ns;
         else
            DOUT <= BRAM(conv_integer(ADDR)) after 5 ns;
         end if;
      end if;
   end if;
end process P1;
end VERHALTEN;
```



A Aufgabe 16.6

Das FIFO wurde gemäß Bild 15.18 konstruiert. Es besteht aus einem Dual-Ported Block-RAM, zwei Adresszählern und einem Prozess zur Erzeugung der Statussignale.

In der Testbench existiert neben den beiden Taktgeneratoren und einem Reset-Prozess ein Producer-Prozess, in dem sukzessive die Daten erzeugt werden, die ins FIFO geschrieben werden: Beginnend bei 255_{10} werden diese bei jeder Schreiboperation dekrementiert. Als Handshake-Signale werden das Write-Enable-Signal WREN gesetzt, wenn das FIFO nicht voll ist und das Read-Enable-Signal, wenn das FIFO nicht leer ist.

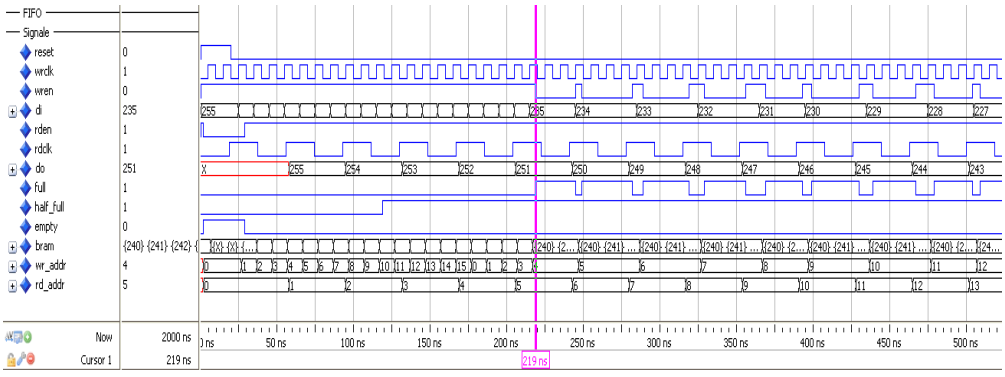
Die Simulation zeigt den Fall b), in dem der Schreibtakt größer als der Lesetakt ist: Zum Zeitpunkt t = 219 ns ist das FIFO gefüllt und das Schreiben wird durch den langsameren Lesetakt ausgebremst.

Ähnlich wie in Aufgabe 15.5 ist die gesamte Testbench als nicht-synthesefähig markiert.

```
-- Aufgabe 15.6
-- FIFO 16x8 mit Testbench

-- synthesis off
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity AUFGABE_15_6_TB is
end AUFGABE_15_6_TB;

architecture TESTBENCH of AUFGABE_15_6_TB is
component FIFO_16x8 is
port( RESET : in bit;
      WRCLK : in bit;
      WREN  : in bit;
      DI    : in std_logic_vector(7 downto 0);
      RDEN  : in bit;
      RDCLK : in bit;
      DO    : out std_logic_vector(7 downto 0);
      FULL  : out bit;
      HALF_FULL : out bit;
      EMPTY : out bit);
end component;
constant DEL_27MHZ : time := 18.5185185 ns;
constant DEL_100MHZ : time := 5 ns;
signal RESET : bit;
signal WRCLK, RDCLK, WREN, RDEN : bit;
signal FULL, HALF_FULL, EMPTY : bit;
signal DI, DO : std_logic_vector(7 downto 0);

begin
RESETP: process -- async. Reset
begin
 RESET <= '1'; wait for 20 ns;
 RESET <= '0'; wait;
end process RESETP;

WRCLKP: process -- Schreibtakt
begin
   WRCLK <= '0'; wait for DEL_100MHZ;
   WRCLK <= '1'; wait for DEL_100MHZ;
end process WRCLKP;

RDCLKP: process -- Lesetakt
begin
   RDCLK <= '0'; wait for DEL_27MHZ;
   RDCLK <= '1'; wait for DEL_27MHZ;
end process RDCLKP;

WREN <= '1' when FULL = '0' else '0';
RDEN <= '1' when EMPTY = '0' else '0';
```

```
PRODUCER: process(WRCLK, RESET)
begin
 if RESET = '1' then
    DI <= x"FF";
 elsif WRCLK='1' and WRCLK'event then
   if FULL ='0' then -- wenn FIFO nicht voll
      DI <= DI - 1;  -- dekrementiere Daten
   end if;
 end if;
end process PRODUCER;

U1: FIFO_16x8   -- FIFO Komponenteninstanz
   port map(RESET, WRCLK, WREN, DI,
            RDEN, RDCLK, DO,
            FULL, HALF_FULL, EMPTY);
end TESTBENCH;

-- synthesis on
-----------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity FIFO_16x8 is
port( RESET :in bit;
      WRCLK : in bit;
      WREN  : in bit;
      DI    : in std_logic_vector(7 downto 0);
      RDEN  : in bit;
      RDCLK : in bit;
      DO    : out std_logic_vector(7 downto 0);
      FULL  : out bit;
      HALF_FULL : out bit;
      EMPTY : out bit);
end FIFO_16x8;

architecture VERHALTEN of FIFO_16x8 is
type BRAM_TYPE is array(15 downto 0) of std_logic_vector(7 downto 0);
signal BRAM : BRAM_TYPE;
signal WR_ADDR : std_logic_vector(3 downto 0);
signal RD_ADDR : std_logic_vector(3 downto 0);

begin
PWRITE: process(WRCLK)              -- Schreiben auf PortA
begin
   if WRCLK='1'and WRCLK'event then
      if WREN ='1' then
         BRAM(conv_integer(WR_ADDR(3 downto 0))) <= DI after 2 ns;
       end if;
   end if;
end process PWRITE;

PREAD: process(RDCLK)               -- Lesen von PortB
begin
   if RDCLK='1'and RDCLK'event then
      if RDEN ='1' then
         DO <= BRAM(conv_integer(RD_ADDR(3 downto 0))) after 2 ns;
      end if;
   end if;
end process PREAD;
```

```
WR_ADDRP: process(WRCLK, RESET)
begin
   if RESET = '1' then
      WR_ADDR <= (others => '0') after 2 ns;
    elsif WRCLK='1' and WRCLK'event then
      if WREN = '1' then
        WR_ADDR <= WR_ADDR + 1 after 2 ns;
      end if;
   end if;
end process WR_ADDRP;

RD_ADDRP: process(RDCLK, RESET)
begin
   if RESET = '1' then
      RD_ADDR <= (others => '0') after 2 ns;
   elsif RDCLK='1' and RDCLK'event then
      if RDEN = '1' then
         RD_ADDR <= RD_ADDR + 1 after 2 ns;
      end if;
   end if;
end process RD_ADDRP;

FLAGS: process(WR_ADDR, RD_ADDR)
begin
   if (WR_ADDR-RD_ADDR) = 15 then
      FULL <= '1' after 2 ns;
   else
      FULL <= '0' after 2 ns;
   end if;
   if (WR_ADDR-RD_ADDR) >= 8 then
      HALF_FULL <= '1' after 2 ns;
   else
      HALF_FULL <= '0' after 2 ns;
   end if;
   if WR_ADDR = RD_ADDR then
      EMPTY <= '1' after 2 ns;
   else
      EMPTY <= '0' after 2 ns;
   end if;
end process FLAGS;
end VERHALTEN;
```



17 Programmierbare Logik

A Aufgabe 17.1

a) Diese Bausteine unterscheiden sich bezüglich ihrer Programmierbarkeit darin, in welcher der beiden UND- und ODER-Logikmatrizen sie sich programmieren lassen (vgl. Bild 17.2, Bild 17.5 und Bild 17.6).

b) Da ROMs alle Minterme verwenden, ist für die Implementierung in ROMs keine Minimierung der logischen Gleichungen erforderlich. Bei der Implementierung in einem PLA sollte eine Multi-Output Minimierung stattfinden, während die Implementierung in einem PAL nur eine individuelle Minimierung der einzelnen Ausgangssignale erfordert.

c) Die ODER-Matrix eines 1k x 16-ROMs besitzt 2^{10} = 1024 Eingänge.

d) Die ODER-Matrix des 8 x 20 x 4-PALs besitzt nur 20 Eingänge.

e) Durch eine Rückführung von Ausgangssignalen in die Produkttermmatrix.

f) Die wesentlichen Blöcke der Coolrunner-II CPLDs sind: Input-/Output Schnittstellenblöcke, PLA-Strukturen (Function Blocks), eine Verdrahtungsmatrix (AIM) sowie ein JTAG-Programmierinterface (vgl. Bild 17.12).

g) SRAM-basierte FPGAS müssen nach dem Einschalten für die jeweilige Aufgabenstellung konfiguriert werden. Dies ist für Antifuse-FPGAs nicht erforderlich. Daher müssen auf Platinen mit SRAM-basierten FPGAs entweder Programmierschnittstellen oder aber Boot-(E)EPROMs vorgesehen werden.

h) Mit Hilfe von Lookup-Tabellen (LUTs, vgl. Kap. 3.7.2).

i) Die Tatsache, dass die Verdrahtung zu nahen und entfernteren CLBs über unterschiedliche Leitungsressourcen erfolgt (vgl. Bild 17.20).

j) Die Taktsignale werden über spezielle, besonders starke Taktsignalverstärker (BUFG) in den FPGA eingespeist sowie innerhalb des FPGAs über einen H-Taktbaum verteilt (vgl. Bild 17.22).

k) Zusätzlich zu den kombinatorischen und getakteten Logikstrukturen besitzen Platform-FPGAs je nach Typ weitere Strukturen, mit denen sich komplexe Hardware-Software Systeme realisieren lassen: Block-RAM-Speicher, Multiplizierer bzw. DSP-Hardwaremodule, Schnelle serielle Interfaces, Ethernet-Schnittstellen sowie komplette Prozessoren (z.B. ARM).

Aufgabenlösungen zu: Jürgen Reichardt, Lehrbuch Digitaltechnik 3. Auflage
ISBN 978-3-486-72765-4. Oldenbourg Wissenschaftsverlag, München 2013.