Hardwarepraktikum WS 1997/98

Versuch 2

Kombinatorische Systeme I

Jan Horbach, 17518
Chris Hübsch, 17543
Lars Jordan, 17560

1. Aufgabe:

Gegenstand des Versuchs ist die BOOLEsche Funktion

$f = x_1 \oplus x_2 \oplus x_3 \oplus x_4$

Berechnen Sie eine Testfolge (minimaler Länge), die geeignet ist, alle SA0- und alle SA1-Fehler an den Eingängen und dem Ausgang einer technischen Realisierung der Funktion *f* zu erkennen.

Bestimmung der Testfolge:

Ein Stuck-at-Fehler ist ein Fehler, der so wirkt, als würde der Pegel an einem Ein- oder Ausgang unabhängig vom Testvektor fest auf 0 (SA0-Fehler) oder 1 (SA1-Fehler) liegen.
Es muß für jeden Fehler mindestens ein Testvektor bestimmt werden, der geeignet ist, diesen Fehler zu erkennen. Durch einen Soll-Ist-Vergleich kann dann geprüft werden, ob einer der Fehler, die durch diesen Testvektor erkannt werden, vorliegt.

Die Testvektoren müssen zwei Bedingungen genügen:

- Provokationsbedingung: Der Befehl muß provoziert werden, d.h. zum Nachweis eines SA0-Fehlers muß am Fehlerort eine 1 angelegt werden, um die Verfälschung zu 0 am Ausgang beobachten zu können, und zum Nachweis eines SA1-Fehlers muß analog der Pegel 0 am Fehlerort angelegt werden.
- Transportbedingung: Die Belegung der anderen Eingänge ist so zu wählen, daß der Pegel mindestens eines Ausgangs vom Pegel am Fehlerort abhängt, d.h. der Fehler am Fehlerort muß zum Ausgang „transportiert“ werden können.

Aus diesen Bedingungen läßt sich eine Berechnungsvorschrift der Testvektoren ableiten, die geeignet sind, Stuck-at-Fehler an einem Eingang x_i der technischen Realisierung einer BOOLEschen Funktion *f* zu erkennen:

SA0-Defekt an x_i:

$$T_{x_i \equiv 0} = x_i \wedge (f(x_1,\ldots,x_i,\ldots,x_k) \oplus f(x_1,\ldots,\overline{x_i},\ldots,x_k) = 1$$

SA1-Defekt an x_i:

$$T_{x_i \equiv 1} = \overline{x_i} \wedge (f(x_1,\ldots,x_i,\ldots,x_k) \oplus f(x_1,\ldots,\overline{x_i},\ldots,x_k) = 1$$

Da der Ausgang stets beobachtbar ist, entfällt für dort auftretende Fehler die Transportbedingung:

$$T_{f \equiv 0} = f = 1$$

$$T_{f \equiv 1} = \overline{f} = 1$$

Berechnung der Transportbedingungen:

$$f(x_1) \oplus f(\overline{x_1}) = (x_1 \oplus x_2 \oplus x_3 \oplus x_4) \oplus (\overline{x_1} \oplus x_2 \oplus x_3 \oplus x_4) = 1 \oplus 0 \oplus 0 \oplus 0 = 1$$
$$f(x_2) \oplus f(\overline{x_2}) = (x_1 \oplus x_2 \oplus x_3 \oplus x_4) \oplus (x_1 \oplus \overline{x_2} \oplus x_3 \oplus x_4) = 0 \oplus 1 \oplus 0 \oplus 0 = 1$$
$$f(x_3) \oplus f(\overline{x_3}) = (x_1 \oplus x_2 \oplus x_3 \oplus x_4) \oplus (x_1 \oplus x_2 \oplus \overline{x_3} \oplus x_4) = 0 \oplus 0 \oplus 1 \oplus 0 = 1$$
$$f(x_4) \oplus f(\overline{x_4}) = (x_1 \oplus x_2 \oplus x_3 \oplus x_4) \oplus (x_1 \oplus x_2 \oplus x_3 \oplus \overline{x_4}) = 0 \oplus 0 \oplus 0 \oplus 1 = 1$$

Die Transportbedingung ist also für alle x_i 1, d.h. *jede* Pegeländerung an einem Eingang ändert das Ergebnis am Ausgang, unabhängig von der Belegung der anderen Eingänge.

Berechnung der Mengen der Testvektoren:

Da die Transportbedingung für alle Eingänge 1 ist, vereinfachen sich die allgemeinen Gleichungen zu:

$$T_{x_i \equiv 0} = x_i \wedge 1 = 1 \Rightarrow x_i = 1$$
$$T_{x_i \equiv 1} = \overline{x_i} \wedge 1 = 1 \Rightarrow x_i = 0$$
$$T_{f \equiv 0} = (x_1 \oplus x_2 \oplus x_3 \oplus x_4) = 1 \Rightarrow s.Tabelle$$
$$T_{f \equiv 1} = \overline{(x_1 \oplus x_2 \oplus x_3 \oplus x_4)} = 1 \Rightarrow s.Tabelle$$

Die Testvektoren lassen sich nun in einer Tabelle zusammenfassen. *TV* steht dabei für Testvektor und $\delta(TV)$ für dessen Dezimaläquivalent. In die Matrix ist das Attribut $e_{i,j}$ mit $i=\delta(TV)$ und $j \in$ *Menge der betrachteten Fehler* wie folgt eingetragen:

$$e_{i,j} = \begin{cases} 0, \text{ falls der Testvektor i nicht geeignet ist, den Fehler j zu erkennen} \\ 1, \text{ falls der Testvektor i geeignet ist, den Fehler j zu erkennen} \end{cases}$$

$\delta(TV)$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Fehler \ *TV*	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
$x_1 \equiv 0$									1	1	1	1	1	1	1	1
$x_1 \equiv 1$	1	1	1	1	1	1	1	1								
$x_2 \equiv 0$					1	1	1	1					1	1	1	1
$x_2 \equiv 1$	1	1	1	1					1	1	1	1				
$x_3 \equiv 0$			1	1			1	1			1	1			1	1
$x_3 \equiv 1$	1	1			1	1			1	1			1	1		
$x_4 \equiv 0$		1		1		1		1		1		1		1		1
$x_4 \equiv 1$	1		1		1		1		1		1		1		1	
$f \equiv 0$		1	1		1			1	1			1		1	1	
$f \equiv 1$	1			1		1	1			1	1		1			1

Nun muß die kürzeste Folge von Testvektoren bestimmt werden, die alle betrachteten Fehler erkennt, d.h. in der Testfolge muß für jeden Fehler ein Testvektor enthalten sein, der diesen Fehler erkennen kann. Damit gilt:

$$\bigwedge_j (\bigvee_i e_{i,j}) = 1$$

Für die konkrete Aufgabe folgt daraus (der 2. Index *j* ist der Einfachheit halber weggelassen):

$$(e_8 \vee e_9 \vee e_{10} \vee e_{11} \vee e_{12} \vee e_{13} \vee e_{14} \vee e_{15}) \wedge$$
$$(e_0 \vee e_1 \vee e_2 \vee e_3 \vee e_4 \vee e_5 \vee e_6 \vee e_7) \wedge$$
$$(e_4 \vee e_5 \vee e_6 \vee e_7 \vee e_{12} \vee e_{13} \vee e_{14} \vee e_{15}) \wedge$$
$$(e_0 \vee e_1 \vee e_2 \vee e_3 \vee e_8 \vee e_9 \vee e_{10} \vee e_{15}) \wedge$$
$$(e_2 \vee e_3 \vee e_6 \vee e_7 \vee e_{10} \vee e_{11} \vee e_{14} \vee e_{15}) \wedge$$
$$(e_0 \vee e_1 \vee e_4 \vee e_5 \vee e_8 \vee e_9 \vee e_{12} \vee e_{13}) \wedge$$
$$(e_1 \vee e_3 \vee e_5 \vee e_7 \vee e_9 \vee e_{11} \vee e_{13} \vee e_{15}) \wedge$$
$$(e_0 \vee e_2 \vee e_4 \vee e_6 \vee e_8 \vee e_{10} \vee e_{12} \vee e_{14}) \wedge$$
$$(e_1 \vee e_2 \vee e_4 \vee e_7 \vee e_8 \vee e_{11} \vee e_{13} \vee e_{14}) \wedge$$
$$(e_0 \vee e_3 \vee e_5 \vee e_6 \vee e_9 \vee e_{10} \vee e_{12} \vee e_{15}) = 1$$

Ein Auflösen und Ausmultiplizieren dieses Ausdrucks wäre ziemlich kompliziert und aufwendig. Wenn man jedoch die Tabelle genauer betrachtet, stellt man fest, daß man bei den Eingängen durch ein „Falten“ und „Umklappen“ der Tabelle zwischen *i=7* und *i=8* alle Felder der Tabelle abdecken kann. Das bedeutet, daß sich schon durch 2 Testvektoren alle Fehler erkennen ließen. Aber offensichtlich genügen die $e_{i,j}$ für den Ausgang nicht dieser Bedingung, so daß man einen dritten Testvektor benötigt, der den noch ausstehenden Fehler erkennt.

Wir haben die Testvektoren „0111“ (*i=7*) und „1000“ (*i=8*) ausgewählt, die sämtliche Stuck-at-Fehler an den Eingängen sowie den SA0-Fehler an *f* erkennen, und zusätzlich noch den Testvektor „1001“ (*i=9*), der geeignet ist, den übrigbleibenden SA1-Fehler an *f* zu erkennen.

Eine mögliche Testfolge der minimalen Länge 3 lautet also:

{0111, 1000, 1001}

Die erwarteten Pegel am Ausgang betragen für die ersten beiden Testvektoren 1, für den letzten Testvektor 0. Falls man einen Eingang ändert, muß sich auch das Ergebnis ändern. Zur Simulierung eines SA0-Fehlers kann man also den Eingangspegel von 1 auf 0 verfälschen, und wenn sich dann der Ausgangspegel ändert, bedeutet das, daß der Testvektor so einen SA0-Fehler an dem Eingang tatsächlich erkennen kann. Bei einem SA1-Fehler verfährt man analog, indem man bewußt den Pegel von 0 auf 1 verfälscht. Zusammenfassend: Der Ausgang darf nur bei dem angelegten Testvektor den erwarteten Pegel führen, bei einer bewußten Eingangsänderung (was einem Stuck-at-Fehler entsprechen würde) muß sich der Ausgangspegel ändern.

2. Aufgabe:

Realisieren Sie die Funktion *f* mit einem 16-zu-1-Multiplexer. Legen Sie die Testfolge aus 1. an die Schaltung an und prüfen Sie die Vollständigkeit der Testfolge in bezug auf die betrachteten Fehler.

Vorüberlegung:

Die zu realisierende Funktion
$f = x_1 \oplus x_2 \oplus x_3 \oplus x_4$
liefert, ausmultipliziert:
$\bar{x}_1\bar{x}_2\bar{x}_3x_4 \vee \bar{x}_1\bar{x}_2x_3\bar{x}_4 \vee \bar{x}_1x_2\bar{x}_3\bar{x}_4 \vee \bar{x}_1x_2x_3x_4 \vee x_1\bar{x}_2\bar{x}_3\bar{x}_4 \vee x_1\bar{x}_2x_3x_4 \vee x_1x_2\bar{x}_3x_4 \vee x_1x_2x_3\bar{x}_4$

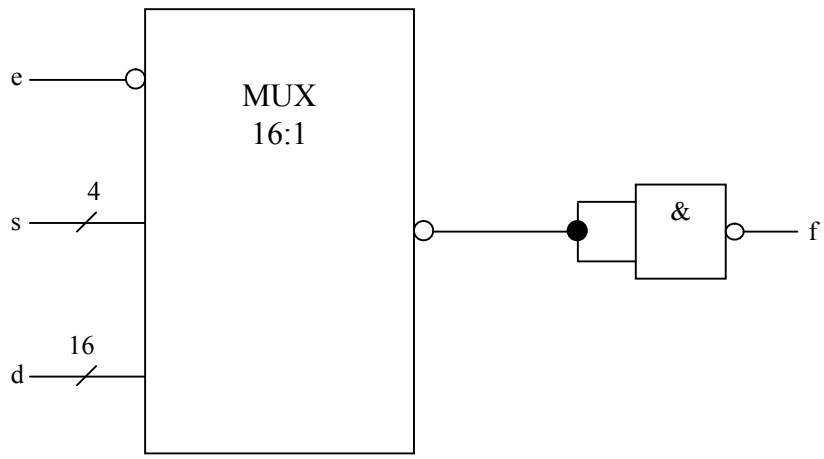
Der Multiplexer besitzt folgende allgemeine Schaltfunktion:

$$\bar{s}_3\bar{s}_2\bar{s}_1\bar{s}_0d_0 \vee \bar{s}_3\bar{s}_2\bar{s}_1s_0d_1 \vee \bar{s}_3\bar{s}_2s_1\bar{s}_0d_2 \vee \bar{s}_3\bar{s}_2s_1s_0d_3 \vee$$
$$\bar{s}_3s_2\bar{s}_1\bar{s}_0d_4 \vee \bar{s}_3s_2\bar{s}_1s_0d_5 \vee \bar{s}_3s_2s_1\bar{s}_0d_6 \vee \bar{s}_3s_2s_1s_0d_7 \vee$$
$$s_3\bar{s}_2\bar{s}_1\bar{s}_0d_8 \vee s_3\bar{s}_2\bar{s}_1s_0d_9 \vee s_3\bar{s}_2s_1\bar{s}_0d_{10} \vee s_3\bar{s}_2s_1s_0d_{11} \vee$$
$$s_3s_2\bar{s}_1\bar{s}_0d_{12} \vee s_3s_2\bar{s}_1s_0d_{13} \vee s_3s_2s_1\bar{s}_0d_{14} \vee s_3s_2s_1s_0d_{15}$$

Setzt man nun $s_3=x_1$, $s_2=x_2$, $s_1=x_3$ und $s_0=x_4$, liefert ein Koeffizientenvergleich folgende Werte für die Dateneingänge *d*:

$d_0=0$, $d_1=1$, $d_2=1$, $d_3=0$, $d_4=1$, $d_5=0$, $d_6=0$, $d_7=1$,
$d_8=1$, $d_9=0$, $d_{10}=0$, $d_{11}=1$, $d_{12}=0$, $d_{13}=1$, $d_{14}=1$, $d_{15}=0$

Die Werte für *s* und *d* legt man an den Multiplexer an, enable *e* wird mit 0 beschaltet, da es negiert vorliegt, und der Ausgang wird durch ein NAND negiert, damit man den Pegel direkt mit den anderen zwei Realisierungsmöglichkeiten vergleichen kann:



Im Anhang befindet sich auch eine VHDL-Beschreibung dieser Schaltung.

Durchführung:

Legt man die oben hergeleiteten Werte und Testvektoren an die Schaltung an, erhält man genau die erwarteten Ergebnisse. Ändert man nun jeweils einen Eingang eines Testvektors auf den entgegengesetzten Wert (was einem Stuck-at-Fehler auf diesem Wert entspricht), ändert sich auch das Ergebnis, d.h. der betrachtete Testvektor wäre in der Lage, diesen Fehler zu erkennen:

TV	mit „Fehler“	Ergebnis	TV	mit „Fehler“	Ergebnis	TV	mit „Fehler“	Ergebnis
0111		1	1000		1	1001		0
	*1*111	*0*		*0*000	*0*		*0*001	*1*
	0*0*11	*0*		1*1*00	*0*		1*1*01	*1*
	01*0*1	*0*		10*1*0	*0*		10*1*1	*1*
	011*0*	*0*		100*1*	*0*		100*0*	*1*

Da das für alle Testvektoren und Fehler der Fall war und außerdem auch ein SA0- bzw. SA1-Fehler am Ausgang sofort festgestellt werden kann, ist das System von Testvektoren vollständig in bezug auf die betrachteten Fehler. Es reicht also aus, nur diese drei Testvektoren an die Schaltung anzulegen, um deren korrekte Realisierung (d.h. ohne Einfach-Stuck-at-Fehler) festzustellen.

3. Aufgabe:

Realisieren Sie die Funktion *f* mit einem 16-zu-1-Multiplexer. Der 16-zu-1-Multiplexer ist aus zwei 8-zu-1-Multiplexern (und einigen Grundgattern) zusammenzusetzen. Legen Sie die Testfolge aus 1. an die Schaltung an und prüfen Sie die Vollständigkeit der Testfolge in bezug auf die betrachteten Fehler.

Vorüberlegung:

Die Werte für *s* und *d* sind denen aus Aufgabe 2 identisch, da sich der Multiplexer nach außen hin genauso verhalten soll. Enable *e* wird wieder mit 0 beschaltet, der Ausgang braucht allerdings nicht noch einmal negiert zu werden, da er schon in der „normalen“ Form vorliegt.

Die Dateneingänge *d* müssen nun auf die beiden Multiplexer aufgeteilt werden (siehe Schaltung), und die Adreßeingänge *s* müssen mit drei der x_i belegt werden. Wir haben dazu x_1, x_2 und x_3 ausgewählt; x_4 geht dann in die Grundgatter mit ein.

Bei dieser Belegung gilt folgende Schaltbelegungstabelle:

x_1	x_2	x_3	x_4	f_1 (MUX1)	f_2 (MUX2)	f
0	0	0	0	1	0	0
0	0	0	1	1	0	1
0	0	1	0	0	1	1
0	0	1	1	0	1	0
0	1	0	0	0	1	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	1
1	0	0	0	0	1	1
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	1
1	1	1	0	0	1	1
1	1	1	1	0	1	0

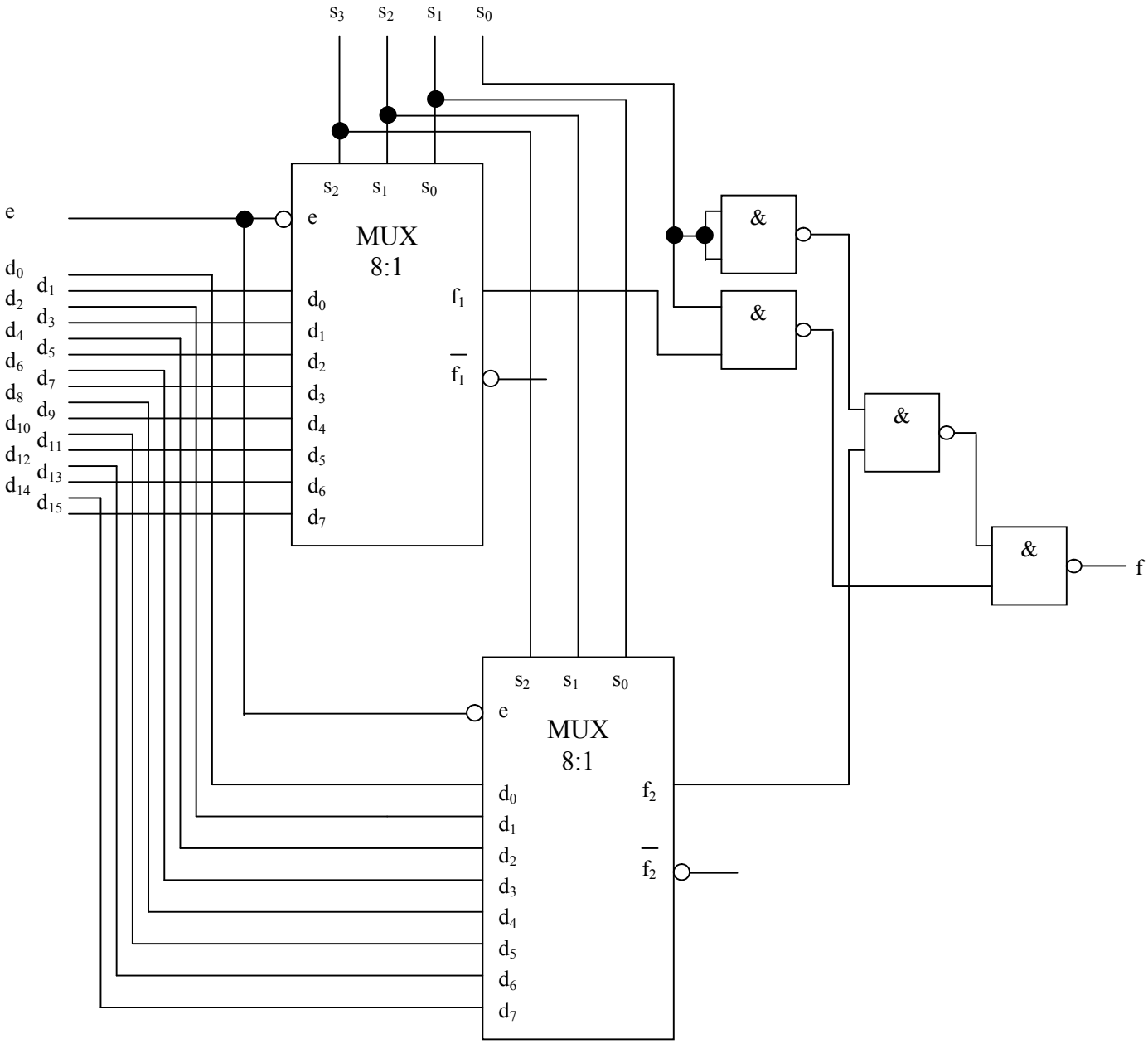
Aus dieser Tabelle läßt sich folgende Formel für *f* ableiten:

$$f = (x_4 \wedge f_1) \vee (\overline{x_4} \wedge f_2)$$

Da wir jedoch nur NAND-Gatter zur Verfügung haben, muß die Formel erst durch Anwendung der DE MORGANschen Gesetze so umgewandelt werden, dass nur noch AND und NOT vorkommen. Diese Formel lautet dann:

$$f = \overline{\overline{x_4 \wedge f_1} \wedge \overline{\overline{x_4} \wedge f_2}}$$

Somit benötigen wir 4 NAND-Gatter mit jeweils 2 Eingängen. Der 16-zu-1-Multiplexer kann dann also folgendermaßen durch zwei 8-zu-1-Multiplexer realisiert werden:



Im Anhang befindet sich auch eine VHDL-Beschreibung dieser Schaltung.

Durchführung:

Die Ergebnisse entsprechen genau denen von Aufgabe 2. Also ist das System von Testvektoren vollständig in bezug auf die betrachteten Fehler und die Schaltung ist korrekt.

4. Aufgabe:

Realisieren Sie die Funktion *f* mit nur einem 8-zu-1-Multiplexer (16-zu-1-Multiplexer mit $s_3=0$, $d_8..d_{15}$ nicht belegt) und einem Negator. Legen Sie die Testfolge aus 1. an die Schaltung an und prüfen Sie die Vollständigkeit der Testfolge in bezug auf die betrachteten Fehler.

Vorüberlegung:

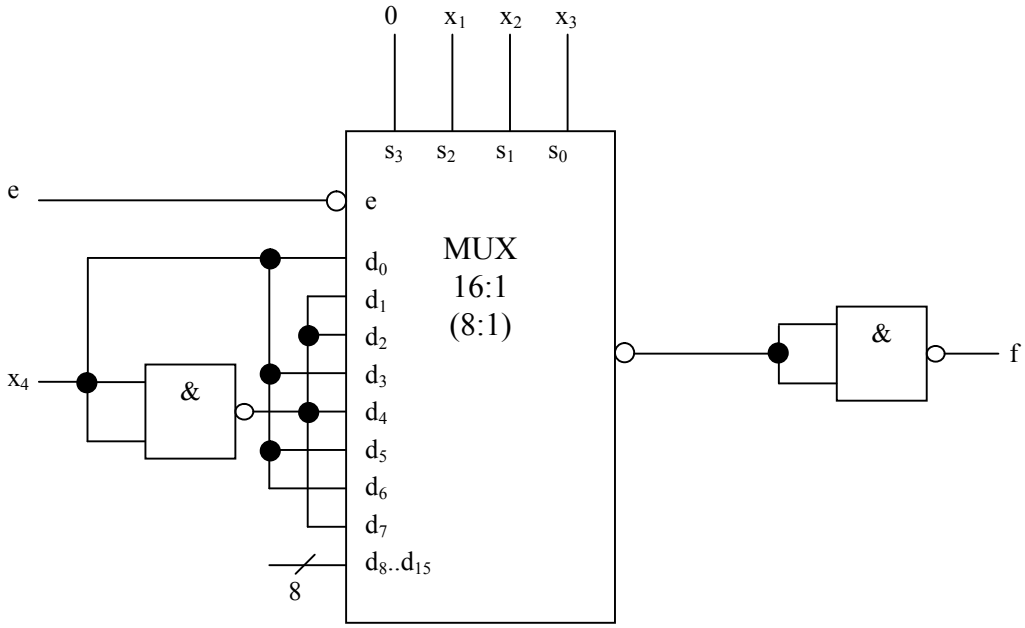
Dieser Multiplexer besitzt folgende allgemeine Schaltfunktion:

$$\bar{s}_2\bar{s}_1\bar{s}_0 d_0 \vee \bar{s}_2\bar{s}_1 s_0 d_1 \vee \bar{s}_2 s_1\bar{s}_0 d_2 \vee \bar{s}_2 s_1 s_0 d_3 \vee s_2\bar{s}_1\bar{s}_0 d_4 \vee s_2\bar{s}_1 s_0 d_5 \vee s_2 s_1\bar{s}_0 d_6 \vee s_2 s_1 s_0 d_7$$

Setzt man $s_2=x_1$, $s_1=x_2$ und $s_0=x_3$, liefert ein Koeffizientenvergleich folgende Werte für die Dateneingänge *d*:

$$d_0 = x_4, \quad d_1 = \overline{x_4}, \quad d_2 = \overline{x_4}, \quad d_3 = x_4, \quad d_4 = \overline{x_4}, \quad d_5 = x_4, \quad d_6 = x_4, \quad d_7 = \overline{x_4}$$

Die Werte für *s* und *d* legt man wieder an den Multiplexer an, enable *e* wird wiederum mit 0 beschaltet, und der Ausgang muß diesmal auch wieder durch ein NAND negiert werden, damit man den Pegel direkt mit den anderen zwei Realisierungsmöglichkeiten vergleichen kann:



Im Anhang befindet sich auch eine VHDL-Beschreibung dieser Schaltung.

Durchführung:

Auch diese Ergebnisse entsprechen genau denen von Aufgabe 2. Also ist das System von Testvektoren vollständig in bezug auf die betrachteten Fehler und auch diese Schaltung ist korrekt.

Anhang – VHDL-Beschreibung:

```
-- Funktionspackage
package func is
  function bin2nat (bits : bit_vector) return natural;
end func;

package body func is
  function bin2nat (bits : bit_vector) return natural is
    variable result : natural := 0;
  begin
    for i in bits'range loop
      result := result * 2 + bit'pos(bits(i));
    end loop;
    return result;
  end bin2nat;
end func;

--------------------------------------------------------------------

-- 2er-NAND
entity nand2 is
  port (x1,x2 : in bit;
        y : out bit);
end nand2;

architecture dataflow of nand2 is
begin
  y <= not(x1 and x2);
end dataflow;

-- 8:1-Multiplexer
use work.func.all;
entity mux8 is
  port (ne   : in  bit;                        -- enable negiert
        s    : in  bit_vector (2 downto 0);    -- Adresseingang
        d    : in  bit_vector (0 to 7);        -- Dateneingang
        f,nf : out bit);                       -- Ausgang (+negiert)
end mux8;

architecture verh of mux8 is
signal temp : bit;
begin
  process (ne,s,d,temp)
  begin
    temp <= d(bin2nat(s)) and (not ne);
    f  <= temp;
    nf <= not temp;
  end process;
end verh;

-- 16:1-Multiplexer
use work.func.all;
entity mux16 is
  port (ne : in  bit;                          -- enable negiert
        s  : in  bit_vector (3 downto 0);      -- Adresseingang
        d  : in  bit_vector (0 to 15);         -- Dateneingang
        nf : out bit);                         -- Ausgang negiert
end mux16;
```

```
architecture verh of mux16 is
begin
  process (ne,s,d)
  begin
    nf <= not (d(bin2nat(s)) and (not ne));
  end process;
end verh;

--------------------------------------------------------------------------

-- Schaltung aus Aufgabe 2
entity schaltung1 is
  port (ne : in bit;                   -- Anlegen von 0 (not enable)
        x1,x2,x3,x4 : in bit;          -- Anlegen der Testvektoren
        d : in bit_vector (0 to 15);   -- Anlegen von 0110100110010110
        f : out bit);                  -- Ausgang
end schaltung1;

architecture struc of schaltung1 is
  component mux16
    port (ne : in  bit;                         -- enable negiert
          s  : in  bit_vector (3 downto 0);     -- Adresseingang
          d  : in  bit_vector (0 to 15);        -- Dateneingang
          nf : out bit);                        -- Ausgang negiert
  end component;
  component nand2
    port (x1,x2 : in bit;
          y : out bit);
  end component;
  signal adr : bit_vector (3 downto 0);
  signal nf  : bit;
  for all: mux16 use entity work.mux16 (verh);
  for all: nand2 use entity work.nand2 (dataflow);
begin
  adr <= x1&x2&x3&x4;
  mux_1:  mux16 port map (ne,adr,d,nf);
  nand_1: nand2 port map (nf,nf,f);
end struc;

-- Schaltung aus Aufgabe 3
entity schaltung2 is
  port (ne : in bit;                   -- Anlegen von 0 (not enable)
        x1,x2,x3,x4 : in bit;          -- Anlegen der Testvektoren
        d : in bit_vector (0 to 15);   -- Anlegen von 0110100110010110
        f : out bit);                  -- Ausgang
end schaltung2;

architecture struc of schaltung2 is
  component mux8
    port (ne   : in  bit;                         -- enable negiert
          s    : in  bit_vector (2 downto 0);     -- Adresseingang
          d    : in  bit_vector (0 to 7);         -- Dateneingang
          f,nf : out bit);                        -- Ausgang (+negiert)
  end component;
  component nand2
    port (x1,x2 : in bit;
          y : out bit);
  end component;
  signal m1,m2,nm1,nm2 : bit;
  signal n1,n2,n3,n4   : bit;
  signal eing1,eing2   : bit_vector (0 to 7);
  signal adr           : bit_vector (2 downto 0);
  for all: mux8  use entity work.mux8  (verh);
  for all: nand2 use entity work.nand2 (dataflow);
```

```
begin
  teiler: process (d)  -- Aufteilung der Dateneingaenge
  begin
    for i in 0 to 7 loop
      eing1(i) <= d(i*2+1);
      eing2(i) <= d(i*2);
    end loop;
  end process;
  adr <= x1&x2&x3;
  mux_1:  mux8  port map (ne,adr,eing1,m1,nm1);
  mux_2:  mux8  port map (ne,adr,eing2,m2,nm2);
  nand_1: nand2 port map (x4,x4,n1);
  nand_2: nand2 port map (x4,m1,n2);
  nand_3: nand2 port map (n1,m2,n3);
  nand_4: nand2 port map (n2,n3,f);
end struc;

-- Schaltung aus Aufgabe 4
entity schaltung3 is
  port (ne : in bit;                  -- Anlegen von 0 (not enable)
        x1,x2,x3,x4 : in bit;         -- Anlegen der Testvektoren
        f : out bit);                 -- Ausgang
end schaltung3;

architecture struc of schaltung3 is
  component mux16  -- nur als 8:1-MUX beschaltet
    port (ne : in  bit;                         -- enable negiert
          s  : in  bit_vector (3 downto 0);     -- Adresseingang
          d  : in  bit_vector (0 to 15);        -- Dateneingang
          nf : out bit);                        -- Ausgang negiert
  end component;
  component nand2
    port (x1,x2 : in bit;
          y : out bit);
  end component;
  signal nx4,nf : bit;
  signal e8to15 : bit_vector (8 to 15) := "00000000";  -- nicht beschaltet
  signal zero   : bit := '0';                          -- S3 = 0
  signal eing   : bit_vector (0 to 15);          -- nur Bits 0-7 gebraucht
  signal adr    : bit_vector (3 downto 0);       -- nur Bits 2-0 gebraucht
  for all: mux16 use entity work.mux16 (verh);
  for all: nand2 use entity work.nand2 (dataflow);
begin
  nand_1: nand2 port map (x4,x4,nx4);   -- Signal x4 ausserhalb des MUX
negiert
  eing <= x4&nx4&nx4&x4&nx4&x4&x4&nx4&e8to15;
  adr <= zero&x1&x2&x3;
  mux_1:  mux16 port map (ne,adr,eing,nf);
  nand_2: nand2 port map (nf,nf,f);
end struc;

--------------------------------------------------------------------

-- Testbench zum Vergleich aller 3 Realisierungen
entity test is
end test;

architecture test of test is
  component schaltung1
    port (ne : in bit;                  -- Anlegen von 0 (not enable)
          x1,x2,x3,x4 : in bit;         -- Anlegen der Testvektoren
          d : in bit_vector (0 to 15);  -- Anlegen von 0110100110010110
          f : out bit);                 -- Ausgang
  end component;
```

```
  component schaltung2
    port (ne : in bit;                    -- Anlegen von 0 (not enable)
          x1,x2,x3,x4 : in bit;           -- Anlegen der Testvektoren
          d : in bit_vector (0 to 15);    -- Anlegen von 0110100110010110
          f : out bit);                   -- Ausgang
  end component;
  component schaltung3
    port (ne : in bit;                    -- Anlegen von 0 (not enable)
          x1,x2,x3,x4 : in bit;           -- Anlegen der Testvektoren
          f : out bit);                   -- Ausgang
  end component;
  signal ne          : bit := '0';  -- not enable
  signal x1,x2,x3,x4 : bit;         -- Testvektoren
  signal d           : bit_vector(0 to 15) := "0110100110010110";
  signal f1,f2,f3    : bit;         -- Ausgaenge
  for all: schaltung1 use entity work.schaltung1 (struc);
  for all: schaltung2 use entity work.schaltung2 (struc);
  for all: schaltung3 use entity work.schaltung3 (struc);
begin
  x1 <= '0', '1' after 100 ns;
  x2 <= '1', '0' after 100 ns;
  x3 <= '1', '0' after 100 ns;
  x4 <= '1', '0' after 100 ns, '1' after 200 ns;
  schaltung_1: schaltung1 port map (ne,x1,x2,x3,x4,d,f1);
  schaltung_2: schaltung2 port map (ne,x1,x2,x3,x4,d,f2);
  schaltung_3: schaltung3 port map (ne,x1,x2,x3,x4,f3);
end test;
```