Hardwarepraktikum WS 1998/99

Versuch 2

Kombinatorische Systeme I

Gruppe 01:
Karsten Petersen, 19164
Marion Riedel, 19259
Tino Schwarze, 19304

Chemnitz, den 18. Dezember 1998

Aufgabe 1

≫ 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 $f = x_1 \not\sim x_2 \not\sim x_3 \not\sim x_4$ zu erkennen.

Transportbedingungen

Zur Berechnung der Transportbedingungen wird die zu testende Eingangsvariable negiert in f eingesetzt und das Ergebnis mit der originalen Funktion f mittels XOR verknüpft:

$$\begin{aligned} f(\overline{x_1}) \not\sim f(x_1) &= (\overline{x_1} \not\sim x_2 \not\sim x_3 \not\sim x_4) \not\sim (x_1 \not\sim x_2 \not\sim x_3 \not\sim x_4) \\ &= 1 \not\sim 0 \not\sim 0 \not\sim 0 \\ &= 1 \end{aligned}$$

Es ist offensichtlich, daß sich aufgrund der inneren Symmetrie von f für jede Transportbedingung das Ergebnis 1 einstellen wird.

Testvektoren

Zur Berechnung der Testvektoren werden nun die zu testende Eingangsvariable und die zugehörige Transportbedingung mittels UND verknüpft und dann alle Eingangsvektoren gesucht, die zum Ergebnis 1 führen würden:

$$\begin{aligned} T_{x_1 \equiv 0} &= x_1 \wedge 1 \stackrel{!}{=} 1 \\ \Rightarrow (x_1, x_2, x_3, x_4) &= (1,0,0,0), (1,0,0,1), (1,0,1,0), (1,0,1,1), \\ &\quad (1,1,0,0), (1,1,0,1), (1,1,1,0), (1,1,1,1) \end{aligned}$$

Die Berechnung der restlichen Testvektoren für die Eingänge $x_i, (i = 2..4)$ erfolgt äquivalent und ist aufgrund der einfachen Struktur von f trivial.

Für den Ausgang f gilt:

$$\begin{aligned} T_{f \equiv 0} &= f \wedge 1 \stackrel{!}{=} 1 \\ T_{f \equiv 1} &= \overline{f} \wedge 1 \stackrel{!}{=} 1 \end{aligned}$$

Umgangssprachlich ausgedrückt heißt das, daß im ersten Fall alle Vektoren mit ungerader Anzahl (bzw. mit gerader Anzahl im zweiten Fall) Einsen als Testvektoren für einen Fehler an f geeignet sind.

Die so berechneten Testvektoren wurden nun in folgende Tabelle eingetragen:

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

In den gesuchten Testfolgen muß für jeden betrachteten Fehler wenigstens ein Testvektor enthalten sein, der diesen Fehler erkennt. Wie aus der Tabelle leicht ersichtlich ist, genügen zwei beliebige komplementäre Testvektoren für die Eingänge plus ein dritter Vektor für die verbleibende Ausgangsbelegung. (Zwei komplementäre Vektoren haben immer den selben Ausgabewert, daher muß zur Erkennung eines Stuck-At Fehlers am Ausgang selbst noch ein Vektor hinzugezogen werden, der einen anderen Ausgangswert hat.) Wir wählen die Vektoren (x_1, x_2, x_3, x_4):

$$\begin{array}{ll} 1. & (0,0,0,0) \\ 2. & (1,1,1,1) \\ 3. & (0,0,0,1) \end{array}$$

Die Testvektoren $(0,0,0,0)$, $(0,0,0,1)$ und $(1,1,1,1)$ ergeben somit eine mögliche Testfolge minimaler Länge, mittels derer wir die Schaltungen der nachfolgenden Aufgaben untersuchen werden.

Die bei fehlerfreier Schaltung erwarteten Ergebnisse (Soll-Pegel) lauten:

Testvektor	Soll-Pegel
0000	0
0001	1
1111	0

Aufgabe 2

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

Für die Realisierung der Schaltung belegen wir die Eingänge s_i, $(i = 0..4)$ folgendermaßen:

$$\begin{aligned} s_0 &= x_1 \\ s_1 &= x_2 \\ s_2 &= x_3 \\ s_3 &= x_4 \end{aligned}$$

Ein Koeffizientenvergleich der ausmultiplizierten Funktion f mit der allgemeinen Schaltfunktion des 16-zu-1-Multiplexers liefert demnach folgende Belegung des Eingangs-Vektors $d = (d_0, d_1..d_{15})$:

$$d = 0110100110010110$$

Daraus ergibt sich dann die Schaltung in Abbildung 1.

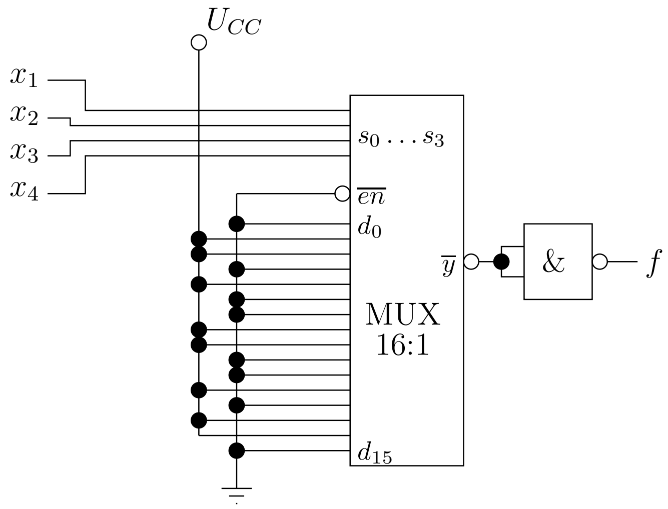

Abbildung 1: Realisierung der Funktion f mit 16:1 Multiplexer

Die `VHDL`-Beschreibung der Schaltung befindet sich im Anhang.

Durchführung

Beim Anlegen unserer Testfolge an obige Schaltung erhielten wir folgende Ergebnisse (Ist-Pegel):

Testvektor	Soll-Pegel	Ist-Pegel
0000	0	0
0001	1	1
1111	0	0

Zur Untersuchung der Vollständigkeit unserer Testfolge negierten wir in jedem Testvektor einzeln und nacheinander jedes der vier Bits (entspricht einem erzwungenen Stuck-at-Fehler). Wie erwartet negierte sich bei *jeder* Änderung auch der Ausgang f. Deshalb kann man nun aussagen, daß die Testfolge vollständig in bezug auf die betrachteten Fehler ist.

Der Versuch verlief somit erfolgreich.

Aufgabe 3

≫ 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 Aufgabe 1 an die Schaltung an und prüfen Sie die Vollständigkeit der Testfolge in bezug auf die betrachteten Fehler.

Der 16-zu-1-Multiplexer läßt sich mit Hilfe zweier 8-zu-1-Multiplexer und eines 2-zu-1-Multiplexers aufbauen.
Ein 2:1-MUX liefert die Schaltfunktion

$$f = \overline{s} \wedge d_0 \;\vee\; s \wedge d_1.$$

Mit Hilfe der Morganschen Gesetze kann man die ODER-Verknüpfung ersetzen:

$$f = \overline{\overline{\overline{s} \wedge d_0} \wedge \overline{s \wedge d_1}}$$

und erhält einen 2:1-MUX, der sich ausschließlich mit NAND-Gattern realisieren läßt.

Dieser 2:1-MUX dient nun dazu, anhand einer Eingangsbelegung zu entscheiden, welcher der beiden 8:1-MUX „aktuell“ ist, d.h. welcher in diesem Fall für den Ausgang relevant ist. Da bei den 8:1-MUX nur drei Adreßeingänge zur

Verfügung stehen, belegen wir diese jeweils mit $s_0 = x_2$, $s_1 = x_3$, $s_2 = x_4$. Das verbleibende x_1 bildet damit das Entscheidungskriterium für den 2:1-MUX: ist es gleich null, wird der obere (siehe Schaltung) MUX benutzt, anderenfalls der untere. Die Beschaltung der Dateneingänge d ist äquivalent zu denen in Aufgabe 2, nur bekommt jetzt der obere MUX $d_0..d_7$ und der untere $d_8..d_{15}$ aus Aufgabe 2.

Daraus ergibt sich dann die Schaltung in Abbildung 2.

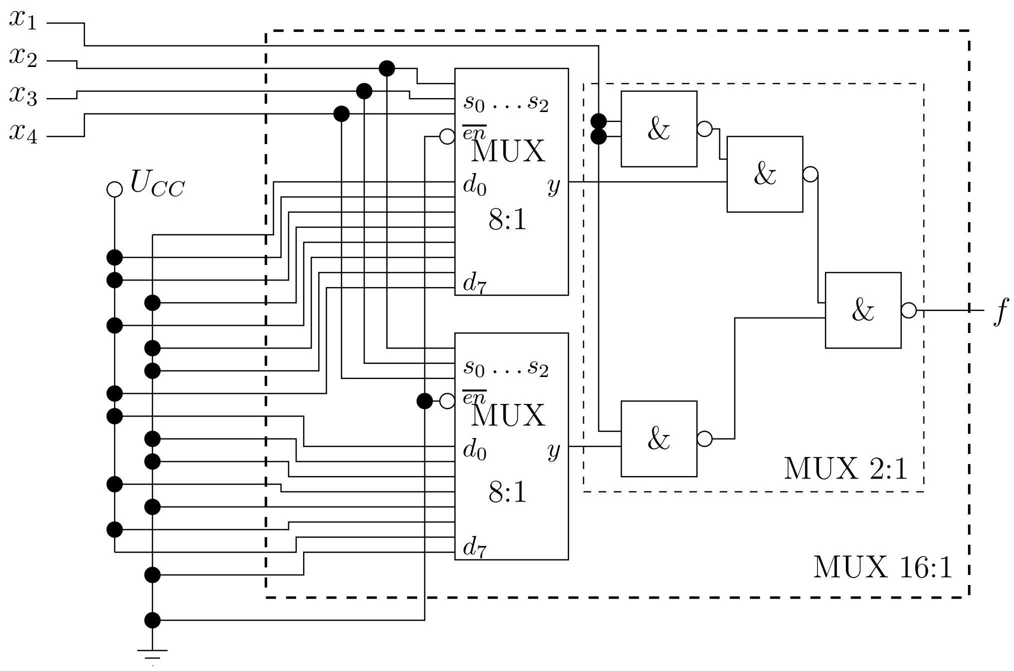

Abbildung 2: Realisierung der Funktion f mit zwei 8:1 Multiplexern

Die VHDL-Beschreibung der Schaltung befindet sich im Anhang.

Durchführung

Die Ist-Pegel, die sich einstellten, entsprachen genau denen von Aufgabe 2 bzw. den Soll-Pegeln. Auch die Ergebnisse beim Testen auf Vollständigkeit (Verfahren in Aufgabe 2 beschrieben) glichen sich genau.

Der Versuch verlief erfolgreich.

Aufgabe 4

≫ Realisieren Sie die Funktion f mit nur einem 8-zu-1-Multiplexer (und einem Negator). Legen Sie die Testfolge aus Aufgabe 1 an die Schaltung an und prüfen Sie die Vollständigkeit der Testfolge in bezug auf die betrachteten Fehler.

Der 16:1-MUX wird in diesem Fall als 8:1-MUX beschaltet, indem die Eingänge s_3 und $d_8 \dots d_{15}$ auf L-Pegel gezogen werden. Wir können also nur drei Adreßeingänge benutzen, die mit $s_0 = x_1$, $s_1 = x_2$, $s_2 = x_3$ belegt werden; s_3 wird mit 0 belegt. Durch Koeffizientenvergleich mit der Schaltfunktion des 8:1-MUX ergibt sich dann für die Beschaltung der Dateneingänge:

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

Das negierte x_4 erhalten wir mittels eines NAND-Gatters.

Es ergibt sich die Schaltung in Abbildung 3.

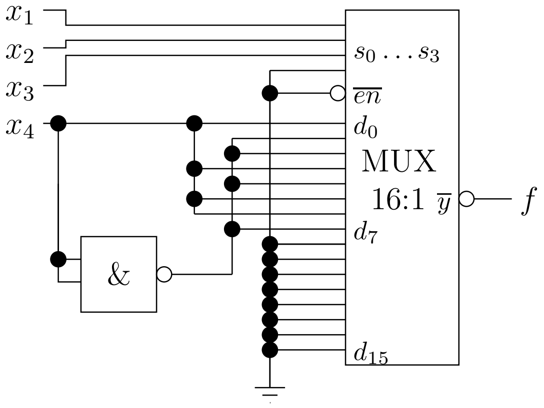

Abbildung 3: Realisierung der Funktion f mit 8:1 Multiplexer und Negator

Durchführung

Wir erhielten dieselben Ergebnisse wie in Aufgabe 2 und 3. Die Schaltung ist also korrekt und die Testfolge in bezug auf die betrachteten Fehler vollständig.

Auch dieser Versuch verlief erfolgreich.

Anhang: VHDL-Beschreibungen

```
entity mux16_inv is              -- invertierender 1:16-MUX
  port (s :in bit_vector(3 downto 0);  -- adress select
        d :in bit_vector(0 to 15);     -- data
        q :out bit                     -- output
       );
end mux16_inv;


architecture behaviour of mux16_inv is
  -- Verhaltensbeschreibung des invertierenden 1:16 MUX

begin
  process (s, d)
    variable i :integer;
    variable sel :integer;

  begin
    sel:=0;
    for i in s'range loop
      sel:=sel*2;

      if (s(i)='1') then
        sel:=sel+1;
      end if ;

    end loop;

    q <= not d(sel);
  end process;
end behaviour;


entity mux8 is         -- 1:8 MUX
        port ( s :in bit_vector(2 downto 0);  -- adress select
               d :in bit_vector(0 to 7);      -- data
               q :out bit);                   -- output
end mux8;


architecture behaviour of mux8 is
        -- Verhaltensbeschreibung des 1:8 MUX
begin
  process (s,d)
    variable i :integer;
    variable sel :integer;
```

```
  begin
    sel:=0;
    for i in s'range loop
      sel:=sel*2;

      if (s(i)='1') then
        sel:=sel+1;
      end if ;
    end loop;

    q <= d(sel);
  end process;
end behaviour;


entity notand is          -- ein NAND mit 2 Eing"angen
        port ( a, b :in bit;          -- input
               y    :out bit);        -- a nand b
end notand;


architecture behaviour of notand is
        -- Verhaltensbeschreibung des NAND
begin
  y <= a nand b;
end behaviour;


entity f is               -- die Realisierung der Funktion f
        port ( x :in bit_vector (4 downto 1); -- x4...x1
               y :out bit);
end f;


architecture behaviour of f is
        -- Direkte Beschreibung der Funktion f
begin
  y <= x(1) xor x(2) xor x(3) xor x(4);
end behaviour;


architecture structure_1 of f is       -- Aufgabe 2

   component mux16 -- verwendet einen inv. 1:16 MUX
      port (s : in bit_vector(3 downto 0);  -- adress select
            d : in bit_vector(0 to 15);     -- data
            q : out bit                     -- output
           );
   end component;
```

```
  component notand
    port (a, b :in bit;
         y :out bit);
  end component;

  signal tmp_d :bit_vector (0 to 15);
  signal mux_out :bit;

  for m: mux16 use entity work.mux16_inv(behaviour);
  for n: notand use entity work.notand(behaviour);

begin
  tmp_d <= "0110100110010110";          -- Eingangsbelegung des MUX
  m: mux16 port map (x, tmp_d, mux_out);
  n: notand port map (mux_out, mux_out, y);
end structure_1;


architecture structure_2 of f is     -- Aufgabe 2
      -- Realisierung der Funktion f mit 2x 8:1 MUX + 4 NAND

  component mux8
    port (s : in bit_vector(2 downto 0);  -- adress select
         d : in bit_vector(0 to 7);       -- data
         q : out bit                      -- output
        );
  end component;

  component notand
    port (a, b :in bit;
         y    :out bit);
  end component;

  signal tmp_d1 :bit_vector (0 to 7);        -- input fuer die MUXe
  signal tmp_d2 :bit_vector (0 to 7);        -- direktes portmap geht nicht

  signal mux1_out, mux2_out :bit;

  signal nand1_out, nand2_out, not_x1 :bit;

  for all: mux8 use entity work.mux8(behaviour);
  for all: notand use entity work.notand(behaviour);

begin
  tmp_d1 <= "01101001";          -- Eingangsbelegung der MUXe
  tmp_d2 <= "10010110";          -- ("Verdrahtung" der Booleschen Funktion)

  m1: mux8 port map (x(4 downto 2), tmp_d1, mux1_out);
```

```
  m2: mux8 port map (x(4 downto 2), tmp_d2, mux2_out);

  n1: notand port map (x(1), x(1), not_x1);
  n2: notand port map (mux2_out, x(1), nand1_out);
  n3: notand port map (mux1_out, not_x1, nand2_out);
  n4: notand port map (nand1_out, nand2_out, y);
end structure_2;


architecture structure_3 of f is      -- Aufgabe 3
      -- Realisierung der Funktion f mit nur einem 1:8 MUX

  component mux8
    port (s : in bit_vector(2 downto 0);  -- adress select
          d : in bit_vector(0 to 7);      -- data
          q : out bit                     -- output
         );
  end component;

  component notand
    port (a, b :in bit;
          y    :out bit);
  end component;

  signal not_x4 :bit;
  signal mux_in :bit_vector (0 to 7);

  for m: mux8 use entity work.mux8(behaviour);
  for n: notand use entity work.notand(behaviour);

begin
  -- x4 negieren
  n: notand port map (x(4), x(4), not_x4);

  -- Eingangsbelegung erzeugen
  mux_in <= x(4) & not_x4 & not_x4 & x(4) & not_x4 & x(4) & x(4) & not_x4;

  -- und an den MUX anlegen
  m: mux8 port map (x(3 downto 1), mux_in, y);
end structure_3;


entity TB_f is             -- Testbench fuer die verschiedenen
                           -- Implementationen von f
      port ( x :in bit_vector (3 downto 0); -- Eingangsvektor
             y :out bit_vector (0 to 3);    -- Ausgaben der Implementationen
             ok :out bit);                  -- Ergebnis okay?
end TB_f;
```

```
architecture structure of TB_f is
        -- Realisierung der Testbench fuer f

   component f
        port (x :in bit_vector (3 downto 0);
              y :out bit);
   end component;

   signal results :bit_vector (0 to 3);

   for f0: f use entity work.f(behaviour);
   for f1: f use entity work.f(structure_1);
   for f2: f use entity work.f(structure_2);
   for f3: f use entity work.f(structure_3);

begin
   f0: f port map (x, results(0));
   f1: f port map (x, results(1));
   f2: f port map (x, results(2));
   f3: f port map (x, results(3));

   -- Vergleichen der Resultate
   ok <= (results(0) and results(1) and results(2) and results(3)) or
         ((not results(0)) and (not results(1)) and (not results(2)) and
          (not results(3)));

   -- Ausgabe aller Resultate
   y <= results;
end structure;
```