Hardwarepraktikum WS 1998/99

Versuch 3

Kombinatorische Systeme II

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

Chemnitz, den 18. Januar 1999

Aufgabe 1

≫ Bestandteil der Versuchsanleitung ist ein Auszug aus dem Datenblatt des Schaltkreises SN 74 LS 181 – Arithmetic Logic Function / Function Generator. Analysieren Sie die Schaltung unter folgenden Einschränkungen:

- Active-High Data
- arithmetische Addition ($M = 0, S_O = 1, S_1 = 0, S_2 = 0, S_3 = 1$)

Anmerkung:
C_{-1} steht in dieser Aufgabe für das eingehende Carry, dieses ist in den anderen Aufgaben mit C_n bezeichnet!

Interne Signale

Aus dem "logic diagram" ergeben sich folgende Einzelsignale:

$$\begin{array}{ll} Z_7 = \overline{A_3 \wedge B_3} & Z_6 = \overline{A_3 \vee B_3} \\ Z_5 = \overline{A_2 \wedge B_2} & Z_4 = \overline{A_2 \vee B_2} \\ Z_3 = \overline{A_1 \wedge B_1} & Z_2 = \overline{A_1 \vee B_1} \\ Z_1 = \overline{A_0 \wedge B_0} & Z_0 = \overline{A_0 \vee B_0} \end{array}$$

Die Z–Signale stellen also effektiv die negierte Ausgabe der beiden Grundverknüpfungen UND und ODER für jedes Eingangspaar dar. Dabei werden die Z mit den ungeraden Indizes nur dann Low, wenn am entsprechenden Eingangspaar beide Eingänge auf $High$ liegen, also hier ein Carry generiert werden muss ("Generate"). Die Z mit den geraden Indizes hingegen werden immer dann Low, wenn mindestens ein Eingang eines Paares auf $High$ liegt ("Propagate").
Die W–Signale lassen sich nach dem Einsetzen der Z's vereinfachen und ergeben schließlich die Funktion der Summenbildung eines Halbadders für jeweils ein Eingangspaar.

$$\begin{aligned} W_3 &= Z_6 \not\leftrightarrow Z_7 \\ &= \overline{A_3 \vee B_3} \not\leftrightarrow \overline{A_3 \wedge B_3} \\ &= A_3 \not\leftrightarrow B_3 \end{aligned}$$

$$
\begin{aligned}
W_2 &= Z_4 \not\leftrightarrow Z_5 \\
&= \overline{A_2 \vee B_2} \not\leftrightarrow \overline{A_2 \wedge B_2} \\
&= A_2 \not\leftrightarrow B_2
\end{aligned}
$$

$$
\begin{aligned}
W_1 &= Z_2 \not\leftrightarrow Z_3 \\
&= \overline{A_1 \vee B_1} \not\leftrightarrow \overline{A_1 \wedge B_1} \\
&= A_1 \not\leftrightarrow B_1
\end{aligned}
$$

$$
\begin{aligned}
W_0 &= Z_0 \not\leftrightarrow Z_1 \\
&= \overline{A_0 \vee B_0} \not\leftrightarrow \overline{A_0 \wedge B_0} \\
&= A_0 \not\leftrightarrow B_0
\end{aligned}
$$

Die Einzel-Carrys $C_{0..2}$ lassen sich jeweils mit Hilfe des nächstniedrigen Carrys vereinfachen (Ausklammern und Substituieren); dies bedeutet, daß die ALU intern bereits mit Carry-Look-Ahead arbeitet.

$$
\begin{aligned}
C_2 &= \overline{(\overline{C_{-1}} \wedge Z_1 \wedge Z_3 \wedge Z_5) \vee (Z_0 \wedge Z_3 \wedge Z_5) \vee (Z_2 \wedge Z_5) \vee Z_4} \\
&= \overline{(((\overline{C_{-1}} \wedge Z_1) \vee Z_0) \wedge Z_3 \vee Z_2) \wedge Z_5 \vee Z_4} \\
&= \overline{(\overline{C_1} \wedge Z_5) \vee Z_4} \\
&= \overline{\overline{(C_1 \vee \overline{Z_5})} \vee Z_4} \\
&= (C_1 \vee \overline{Z_5}) \wedge \overline{Z_4} \\
&= C_1\overline{Z_4} \vee \overline{Z_5 Z_4} \\
&= C_1\overline{Z_4} \vee \overline{Z_5}
\end{aligned}
$$

$$
\begin{aligned}
C_1 &= \overline{(\overline{C_{-1}} \wedge Z_1 \wedge Z_3) \vee (Z_0 \wedge Z_3) \vee Z_2} \\
&= \overline{((\overline{C_{-1}} \wedge Z_1) \vee Z_0) \wedge Z_3 \vee Z_2} \\
&= \overline{(\overline{C_0} \wedge Z_3) \vee Z_2} \\
&= \overline{\overline{(C_0 \vee \overline{Z_3})} \vee Z_2} \\
&= (C_0 \vee \overline{Z_3}) \wedge \overline{Z_2} \\
&= C_0\overline{Z_2} \vee \overline{Z_3 Z_2} \\
&= C_0\overline{Z_2} \vee \overline{Z_3}
\end{aligned}
$$

$$
\begin{aligned}
C_0 &= \overline{(\overline{\overline{C_{-1}} \wedge Z_1}) \vee Z_0} \\
&= \overline{\overline{(C_{-1} \vee \overline{Z_1})} \vee Z_0} \\
&= (C_{-1} \vee \overline{Z_1}) \wedge \overline{Z_0} \\
&= C_{-1}\overline{Z_0} \vee \overline{Z_1 Z_0} \\
&= C_{-1}\overline{Z_0} \vee \overline{Z_1}
\end{aligned}
$$

$$
\begin{aligned}
\overline{C_{n+4}} &= \overline{C_{-1}} Z_1 Z_3 Z_5 Z_7 \vee Z_0 Z_3 Z_5 Z_7 \vee Z_2 Z_5 Z_7 \vee Z_4 Z_7 \vee Z_6 \\
&= (((\overline{C_{-1}} Z_1 \vee Z_0) Z_3 \vee Z_2) Z_5 \vee Z_4) Z_7 \vee Z_6 \\
&= (\overline{C_2} \wedge Z_7) \vee Z_6 \\
&= \overline{(C_2 \vee \overline{Z_7})} \vee Z_6 \\
&= \overline{(C_2 \vee \overline{Z_7}) \wedge \overline{Z_6}} \\
&= \overline{C_2 \overline{Z_6} \vee \overline{Z_7 Z_6}} \\
&= \overline{C_2 \overline{Z_6} \vee \overline{Z_7}}
\end{aligned}
$$

Der Endzustand der einzelnen C's entspricht damit Formel (10) aus dem Script.
Für die Ausgänge $F_{0..3}$ werden nun die Halbadder-Summen über dem entsprechenden Eingangspaar mit dem nächstniedrigen Carry verknüpft. Dies entspricht Formel (11) aus dem Skript.

$$
\begin{aligned}
F_3 &= W_3 \not\vee C_2 \\
F_2 &= W_2 \not\vee C_1 \\
F_1 &= W_1 \not\vee C_0 \\
F_0 &= W_0 \not\vee C_{-1}
\end{aligned}
$$

Es bleiben abschließend noch die speziellen Ausgangssignale:
X verknüpft hier die ungeraden Z ("Generate") so, daß X praktisch das globale "Generate" der ALU bildet. Y ist dann das globale "Propagate".

$$
\begin{aligned}
X &= \overline{Z_1 \wedge Z_3 \wedge Z_5 \wedge Z_7} \\
Y &= \overline{Z_6 \vee (Z_4 \wedge Z_7) \vee (Z_2 \wedge Z_5 \wedge Z_7) \vee (Z_0 \wedge Z_3 \wedge Z_5 \wedge Z_7)}
\end{aligned}
$$

Damit ist die Analyse der Schaltung abgeschlossen.

Aufgabe 2

≫ Entwerfen und realisieren Sie einen 8-Bit-Ripple-Carry-Adder aus zwei Gruppen zu je vier Bit. Ein- und auslaufender Übertrag sollen wie die A_i und B_i high-active sein.

Für die Realisierung der Schaltung sind uns zwei ALU's vorgegeben, die jeweils die Addition zweier 4-Bitvektoren mit ein- und auslaufendem Übertrag realisieren.
Um eine Addition zweier 8-Bitvektoren A und B zu realisieren, belegen wir die Eingänge der ersten ALU mit den vier jeweils unteren Bits von A und B und dem Übertrag C_n. Die zweite ALU bekommt als Eingangswerte dann die vier oberen Bits der Vektoren und als Übertrag den Carry-Ausgang der ersten ALU. Die Ausgänge F_i der beiden ALU's und der (negierte) Carry-Ausgang der zweiten ALU bilden das Additionsergebnis.
(Die Negatoren für die Carrys sind aufgrund der ALU-Vorgabe notwendig.)

Daraus ergibt sich die Schaltung in Abbildung 1.

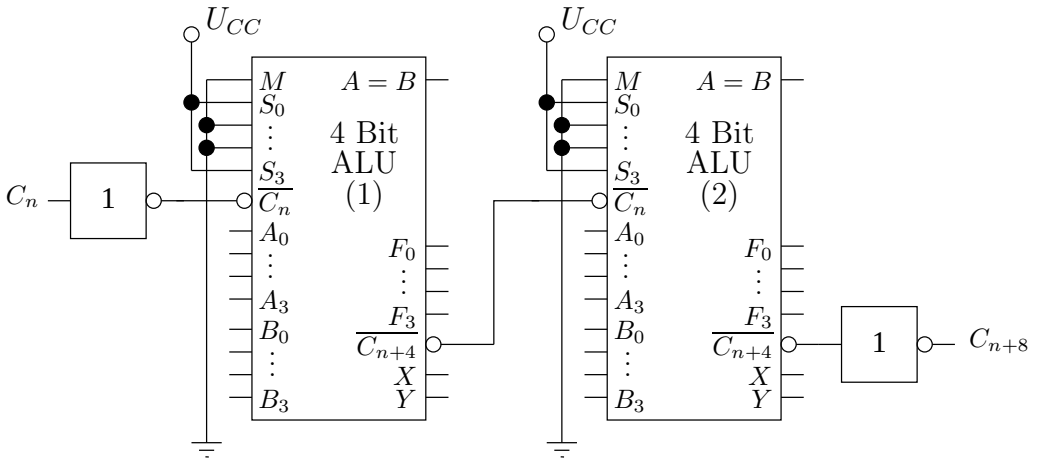

Abbildung 1: 8-Bit-Ripple-Carry-Adder aus zwei Gruppen zu vier Bit

Eine `VHDL`-Beschreibung der Schaltung sowie die Ergebnisse der Simulation befinden sich im Anhang.

Durchführung

Die Addition verschiedener Testaufgaben funktionierte fehlerlos.

Der Versuch verlief erfolgreich.

Aufgabe 3

≫ Entwerfen und realisieren Sie eine CLA-Logik, die den Übertrag für den 8-Bit-Adder vorrausschauend erzeugt. Verwenden Sie dazu die von den 4-Bit-ALUs bereitgestellten Signale X und Y.

Der vorgegebene Schaltkreis SN 74 LS 182 realisiert eine CLA-Logik für eine in vier Gruppen aufgeteilte Addition. Da wir jedoch nur zwei Gruppen haben, vereinfacht sich die Schaltung. Wir benötigen zur Analyse also – umgangssprachlich ausgedrückt – nur die untere Hälfte des abgedruckten "logic diagram". Relevant für uns ist nur der Ausgang C_{n+y} (Pin (11)), da wir nur das Carry für die gesamte Addition (also C_{n+8}) vorausschauend bestimmen sollen. Die CLA-Bestimmung von C_{n+4} (Pin (12) $\hat{=}$ C_{n+x}) können wir uns also sparen.

Die Analyse des "logic diagram" ergibt:

$$\overline{C_{n+8}} = \overline{X_1 Y_1 \vee Y_0 Y_1 X_0 \vee C_n Y_0 Y_1}$$

Die X_i und Y_i werden von den ALU's geliefert, die genauen Formeln der inneren Berechnung stehen in Aufgabe 1.
Da wir für die Realisierung der Schaltung nur NAND-Gatter zur Verfügung haben, formen wir die Funktion durch eine einfache Negation der gesamten Gleichung und Anwendung der Morganschen Gesetze um und erhalten für unsere CLA:

$$C_{n+8} = \overline{\overline{X_1 Y_1} \wedge \overline{Y_0 Y_1 X_0} \wedge \overline{C_n Y_0 Y_1}}$$

Abbildung 2 zeigt den Schaltplan.

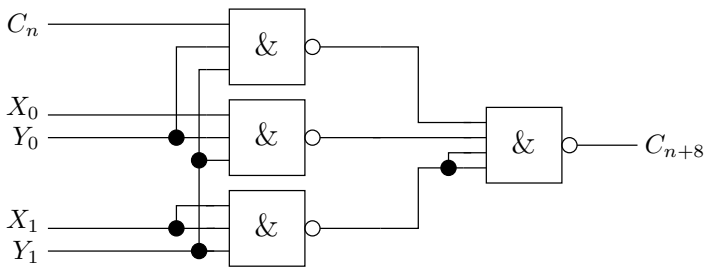

Abbildung 2: CLA-Logik für den 8-Bit-Ripple-Carry-Adder

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

Durchführung

Das durch die CLA-Logik bestimmte Carry und der Carry-Ausgang der zweiten ALU (endgültiger Übertrag der 8-Bit-Addition) stimmten stets überein.

Der Versuch verlief erfolgreich.

Anhang A: VHDL-Beschreibungen

```
entity ls181 is          -- SN74LS181
       port ( m      :in bit;          -- logische vs. arithmetische Fkt.
              s      :in bit_vector (3 downto 0); -- Funktionsselektor
              Cn     :in bit;          -- carry-in
              a      :in bit_vector (3 downto 0); -- A
              b      :in bit_vector (3 downto 0); -- B
              eq     :out bit;         -- A=B
              f      :out bit_vector (3 downto 0); -- f(s,a,b)
              Cnp4   :out bit;         -- out carry (Cn+4)
              X      :out bit;         -- carry propagate inputs
              Y      :out bit);        -- carry generate inputs
end ls181;


architecture behaviour of ls181 is

  signal z0,z1,z2,z3,z4,z5,z6,z7 :bit;
  signal w3,w2,w1,w0 :bit;
  signal c2,c1,c0,cm1 :bit;
  signal Y_tmp :bit;
  signal f_tmp :bit_vector (3 downto 0);

begin
  -- die Schaltfunktionen wurden aus dem Datenblatt uebertragen
  z7 <= (b(3) and a(3) and s(3)) nor (not b(3) and a(3) and s(2));
  z6 <= not ((not b(3) and s(1)) or (b(3) and s(0)) or a(3));

  z5 <= (b(2) and a(2) and s(3)) nor (not b(2) and a(2) and s(2));
  z4 <= not ((not b(2) and s(1)) or (b(2) and s(0)) or a(2));

  z3 <= (b(1) and a(1) and s(3)) nor (not b(1) and a(1) and s(2));
  z2 <= not ((not b(1) and s(1)) or (b(1) and s(0)) or a(1));

  z1 <= (b(0) and a(0) and s(3)) nor (not b(0) and a(0) and s(2));
  z0 <= not ((not b(0) and s(1)) or (b(0) and s(0)) or a(0));

  w3 <= z6 xor z7;
  w2 <= z4 xor z5;
  w1 <= z2 xor z3;
  w0 <= z0 xor z1;

  Y_tmp <= not (z6 or (z7 and z4) or (z7 and z5 and z2) or (z7 and z5 and z3 and z0));
  Y <= Y_tmp;

  X <= not (z7 and z5 and z3 and z1);
  Cnp4 <= not Y_tmp or (z7 and z5 and z3 and z1 and Cn);
```

```
  c2 <= not ((Cn and z1 and z3 and z5 and (not m))
        or (z3 and z5 and z0 and (not m))
        or (z5 and z2 and (not m))                                    50
        or (z4 and (not m)));
  c1 <= not ((Cn and z1 and z3 and (not m))
        or (z3 and z0 and (not m))
        or (z2 and (not m)));
  c0 <= (Cn and z1 and (not m)) nor (z0 and (not m));
  cm1 <= (not m) nand Cn;

  f_tmp(3) <= w3 xor c2;
  f_tmp(2) <= w2 xor c1;
  f_tmp(1) <= w1 xor c0;                                              60
  f_tmp(0) <= w0 xor cm1;
  f <= f_tmp;
  eq <= f_tmp(3) and f_tmp(2) and f_tmp(1) and f_tmp(0);
end behaviour;


entity alu_8bit is      -- 8-Bit ALU mit ein- und ausgehendem Carry
      port   (      Cn      :in bit;
                    a       :in bit_vector(7 downto 0);
                    b       :in bit_vector(7 downto 0);               70
                    f       :out bit_vector(7 downto 0);
                    Cnp8    :out bit);
end alu_8bit;


architecture structure of alu_8bit is
-- Realisierung der 8-Bit-ALU mit 2x LS181

  component ls181
      port ( m      :in bit;        -- logische vs. arithmetische Fkt.  80
             s      :in bit_vector (3 downto 0); -- Funktionsselektor
             Cn     :in bit;        -- carry-in
             a      :in bit_vector (3 downto 0); -- A
             b      :in bit_vector (3 downto 0); -- B
             eq     :out bit;       -- A=B
             f      :out bit_vector (3 downto 0); -- f(s,a,b)
             Cnp4   :out bit;       -- out carry (Cn+4)
             X      :out bit;       -- carry propagate inputs
             Y      :out bit);      -- carry generate inputs
  end component;                                                      90

  for all: ls181 use entity work.ls181(behaviour);

  signal s_const :bit_vector (3 downto 0);
  signal not_Cn :bit;
```

```
  signal cnp4 :bit;     -- Uebertrag aus 1. ALU
  signal not_Cnp8 :bit;

  signal X0, X1, Y0, Y1 :bit; -- propagate/generate Leitungen
                              -- (hier ungenutzt)                    100

  signal eq1, eq2 :bit; -- (unbenutzte) A=B-Ausganege
  signal zero :bit;

begin
  zero <= '0';
  s_const <= "1001"; -- feste Belegung der Function-Select-Eingaenge

  not_Cn <= not Cn;
                                                                     110
  alu1: ls181 port map (zero, s_const, not_Cn, a(3 downto 0), b(3 downto 0),
                        eq1, f(3 downto 0), cnp4, X0, Y0);
  alu2: ls181 port map (zero, s_const, Cnp4, a(7 downto 4), b(7 downto 4),
                        eq2, f(7 downto 4), not_Cnp8, X1, Y1);

  Cnp8 <= not not_Cnp8;
end structure;


entity CLA is                                                        120
        port   (        Cn      :in bit;        -- eingehendes carry
                        X0,Y0   :in bit;        -- X,Y vom 1. Adder
                        X1,Y1   :in bit;        -- X,Y vom 2. Adder
                        Cnp8    :out bit        -- resultierendes Carry
                );
end CLA;


architecture behaviour of CLA is
                                                                     130
begin
  Cnp8 <= not ((not (Y0 and Y1 and Cn)) and
               (not (Y0 and Y1 and X0)) and
               (not (X1 and Y1)));
end behaviour;


entity TB_alu_8bit is
        -- Testbench fuer CLA
        port   (        Cn      :in bit;                             140
                        a       :in bit_vector(7 downto 0);
                        b       :in bit_vector(7 downto 0);
                        f       :out bit_vector(7 downto 0);
                        Cnp8    :out bit;       -- Carry von 2. ALU
```

```
                    Ccla    :out bit        -- Carry von CLA
                );
end TB_alu_8bit;


architecture structure of TB_alu_8bit is                                    150
-- Realisierung der 8-Bit-ALU mit 2x LS181
-- und extra CLA

  component ls181
        port ( m      :in bit;          -- logische vs. arithmetische Fkt.
               s      :in bit_vector (3 downto 0); -- Funktionsselektor
               Cn     :in bit;          -- carry-in
               a      :in bit_vector (3 downto 0); -- A
               b      :in bit_vector (3 downto 0); -- B
               eq     :out bit;         -- A=B                             160
               f      :out bit_vector (3 downto 0); -- f(s,a,b)
               Cnp4   :out bit;         -- out carry (Cn+4)
               X      :out bit;         -- carry propagate inputs
               Y      :out bit);        -- carry generate inputs
  end component;

  component CLA
        port   (       Cn     :in bit;          -- eingehendes carry
                       X0,Y0  :in bit;          -- X,Y vom 1. Adder
                       X1,Y1  :in bit;          -- X,Y vom 2. Adder        170
                       Cnp8   :out bit         -- resultierendes Carry
               );
  end component;

  for all: ls181 use entity work.ls181(behaviour);
  for all: CLA use entity work.CLA(behaviour);

  signal s_const :bit_vector (3 downto 0);
  signal not_Cn :bit;
  signal cnp4 :bit;     -- Uebertrag aus 1. ALU                             180
  signal not_Cnp8 :bit;

  signal X0, X1, Y0, Y1 :bit; -- propagate/generate Leitungen
                              -- (hier ungenutzt)

  signal eq1, eq2 :bit; -- (unbenutzte) A=B-Ausganege
  signal zero :bit;

begin
  zero <= '0';                                                              190
  s_const <= "1001"; -- feste Belegung der Function-Select-Eingaenge

  not_Cn <= not Cn;
```

```
  alu1: ls181 port map (zero, s_const, not_Cn, a(3 downto 0), b(3 downto 0),
                        eq1, f(3 downto 0), cnp4, X0, Y0);
  alu2: ls181 port map (zero, s_const, Cnp4, a(7 downto 4), b(7 downto 4),
                        eq2, f(7 downto 4), not_Cnp8, X1, Y1);
  cla1: cla port map (Cn, X0,Y0,X1,Y1, Ccla);
                                                                        200
  Cnp8 <= not not_Cnp8;
end structure;
```

Anhang B: Simulations-Ergebnisse

```
                /       /       / /       / /
                T       T       T T       T T
                B       B       B B       B B
                -       -       - -       - -
                a       a       a a       a a
                l       l       l l       l l
                u       u       u u       u u
                -       -       - -       - -
                8       8       8 8       8 8
                b       b       b b       b b
                i       i       i i       i i
                t       t       t t       t t
                /       /       / /       / /
                a       b       C f       C C
                (       (       n (       n c
                7       7       . 7       p l
                :       :       . :       8 a
                0       0       . 0       . .
                )       )       . )       . .
                ======== ======== - ======== - -
 0              00000000 00000000 0 00000000 0 0
10              00000000 00000000 1 00000001 0 0
20              00000001 00000000 0 00000001 0 0
30              00000000 10000000 0 10000000 0 0
40              00000000 10000000 1 10000001 0 0
50              10000000 10000000 0 00000000 1 1
60              10000000 01111111 0 11111111 0 0
70              10000000 01111111 1 00000000 1 1
80              00000001 00000001 1 00000011 0 0
```