VHDL-SIMULATION UND -SYNTHESE (8.A)

Jürgen Reichardt, Bernd Schwarz

Lösungen zu den Übungsaufgaben

(Stand: 12.08.2020)

Die in diesem Abschnitt angegebenen VHDL-Codes zu den Entwurfsaufgaben stellen nur exemplarische Musterlösungen dar. Naturgemäß existiert bei Verwendung unterschiedlicher Syntaxkonstrukte eine Vielfalt gleichwertiger Lösungen. **In jedem Fall wird, auch wenn dies im Buch nicht explizit angegeben ist, empfohlen Syntax und Semantik der selbst entworfenen Codes mit einem VHDL-Simulator zu überprüfen.**

Lösungen zu den Aufgaben in Kap. 2.7

2.1

a) Hilfe	Korrekt
b) help	Korrekt
c) 2ter_Versuch	Falsch: Ziffer am Anfang
d) Case	Falsch: VHDL-Schlüsselwort
e) Zweiter_Versuch	Korrekt
f) Dieser_Bezeichner_ist_lang_aber_ist_er_auch_gueltig	Korrekt, wird aber bei der Synthese abgeschnitten

2.2

a) Mein_Name	, MeinName	: Unterschiedlich
b) nummer	, NUMMER	: Nicht unterschiedlich
c) Nummer	, Nummern	: Unterschiedlich
d) two	, too	: Unterschiedlich

2.3

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

2.4

```
-- 8 zu 1 Decoder
---------------------------
entity DECODER1X8 is
       port(  S: in bit_vector(2 downto 0);
              Y: out bit_vector (7 downto 0));
end DECODER1X8;
architecture VERHALTEN1 of DECODER1X8 is
begin
       with S select
       Y <=   "00000001" when "000",
              "00000010" when "001",
              "00000100" when "010",
              "00001000" when "011",
              "00010000" when "100",
              "00100000" when "101",
              "01000000" when "110",
              "10000000" when "111";
end VERHALTEN1;
architecture VERHALTEN2 of DECODER1X8 is
begin
       Y <=   "00000001" when S="000" else
              "00000010" when S="001" else
              "00000100" when S="010" else
              "00001000" when S="011" else
              "00010000" when S="100" else
              "00100000" when S="101" else
              "01000000" when S="110" else
              "10000000" ;
end VERHALTEN2;
```

2.5

```
-- Programmierbare kombinatorische Logik
------------------------------------------------------------
entity LOGIK1 is
       port(  E, S: in bit;
              Y: out bit);
end LOGIK1;
architecture VERHALTEN of LOGIK1 is
begin
       with S select
       Y <=   E when '0',
              not E when '1';
end VERHALTEN;
```

Synthetisierte Gleichung:

$Y = E \leftarrow\!+\!\rightarrow S;$

2.6

```
-- Look-Up Tabelle
------------------------------------------------------------
entity LUT is
       port( S: in bit_vector(1 downto 0);
              A, B: in bit;
              Y: out bit);
end LUT;
architecture VERHALTEN of LUT 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;
```

Synthetisierte Gleichungen:

$n0b = B \vee A;$

n0c = ¬B ∨ ¬A;

Y = (S_1 ∧ ¬S_0 ∧ n0c) ∨ (¬S_1 ∧ S_0 ∧ n0b) ∨ (¬S_1 ∧ ¬n0c) ∨ (S_1 ∧ ¬n0b);

Lösungen zu den Aufgaben in Kap. 3.8

3.1

a) B"1010_0101_1001"	Länge: 12, Wert: A59\h = 2649
b) B"1001-0011"	Falsch: Enthält Minuszeichen
c) b"1111_000"	Länge 7,Wert: 78\h = 120
d) "11110000"	Länge 8,Wert: F0\h = 240
e) x"B5_CD"	Länge 16, Wert: B5CD\h = 46541
f) X"3HA4"	Falsch: Enthält Zeichen „H“
g) o"123"	Länge 9,Wert: 53\h = 83
h) O"123_678"	Falsch: Zeichen „8“ ist keine erlaubte Oktalzahl

3.2

```
-- FPGA-CLB
---------------------------------
entity FPGA_CLB is
       port( A, B, CLK, RESET : in bit;
              S: in bit_vector(2 downto 0);
              Y: out bit);
end FPGA_CLB;

architecture VERHALTEN of FPGA_CLB is
signal TEMP0, TEMP1: bit;
begin
-- Look-Up-Tabelle als Prozess
LUT: process(A, B, S(1 downto 0))
       begin
              case S(1 downto 0) is
                     when "00" => TEMP0 <= A and B;
                     when "01" => TEMP0 <= A or B;
                     when "10" => TEMP0 <= not(A and B);
                     when "11" => TEMP0 <= not (A or B);
       end case;
end process LUT;

-- Flipflop als Prozess
FF: process( CLK, RESET)
       begin
              if RESET='1' then
                     TEMP1 <= '0';
              elsif CLK'event and CLK='1' then
                     TEMP1 <= TEMP0;
       end if;
end process FF;

-- Multiplexer als nebenlaeufige Anweisung
with S(2) select Y <= TEMP0 when '0', TEMP1 when '1';
end VERHALTEN;
```

3.3

```
-- Halbsubtrahierer mit Wahrheitstabelle
-- B  A  I Y COUT
-- ------I-------
-- 0  0  I 0  0
-- 0  1  I 1  0
-- 1  0  I 1  1
-- 1  1  I 0  0
-------------------------------------------------------
entity HALBSUB is
       port( B, A: in bit;
              Y, COUT: out bit);
end HALBSUB;
architecture TABELLE of HALBSUB is
```

```
signal TEMP_IN, TEMP_OUT: bit_vector(1 downto 0);   -- Temp-Signale
begin
       TEMP_IN <= B & A;             -- Temporaeres Eingangssignal
       Y <= TEMP_OUT(1);             -- Ausgangssignal
       COUT <= TEMP_OUT(0);          -- Ausgangssignal
       with TEMP_IN select
       TEMP_OUT <=    "00" when "00",
                      "10" when "01",
                      "11" when "10",
                      "00" when "11";
end TABELLE;
```

3.4

```
-- Code-Umsetzer Gray-Code-> Binaercode
---------------------------------------
entity GRAY is
       port( G: in bit_vector(3 downto 0);
             B: out bit_vector(3 downto 0));
end GRAY;
architecture VERHALTEN of GRAY is
begin
P1: process( G )
       begin
             case G is
                    when x"0" => B <= x"0";
                    when x"1" => B <= x"1";
                    when x"3" => B <= x"2";
                    when x"2" => B <= x"3";
                    when x"6" => B <= x"4";
                    when x"7" => B <= x"5";
                    when x"5" => B <= x"6";
                    when x"4" => B <= x"7";
                    when x"C" => B <= x"8";
                    when x"D" => B <= x"9";
                    when x"F" => B <= x"A";
                    when x"E" => B <= x"B";
                    when x"A" => B <= x"C";
                    when x"B" => B <= x"D";
                    when x"9" => B <= x"E";
                    when x"8" => B <= x"F";
             end case;
       end process P1;
end VERHALTEN;
```

3.5

```
-- Paritaetschecker fuer gerade Paritaet
----------------------------------------
entity PAR_CHECK is
       generic( N : integer :=4);
       port(  D: in bit_vector(N downto 0);
              OK: out bit);
end PAR_CHECK;
architecture VERHALTEN of PAR_CHECK is
begin
PARGEN: process( D )
       variable PAR: boolean;
       begin
             PAR:= false;                          -- Var. initialisieren
             for I in N-1 downto 0 loop            -- Alle Bits ausser MSB
                    if D(I) = '1' then             -- Falls '1'
                           PAR := not PAR;         -- Toggeln
                    end if;
             end loop;
             if ((PAR and D(N)='1') or
                      (not PAR and D(N)='0'))      -- mit MSB vergleichen
             then
                    OK <= '1';
             else
                    OK <= '0';
             end if;
       end process PARGEN;
end VERHALTEN;
```

3.6

```
-- Testbench fuer
-- 3-Bit Johnson-Zähler mit 1:2 u. 1:6 Frequenzteiler
entity TEST_TEIL is
  port ( COUNT: out bit_vector (2 downto 0);-- Johnson-Zaehlbits
          FTEIL: out bit_vector (1 downto 0) );     -- MSB: 2:1, LSB: 6:1
end TEST_TEIL;
architecture ARCH1 of TEST_TEIL is
signal CLK, RESET: bit;              -- lokale Signale
signal TEMP: bit_vector (2 downto 0);
begin
---------------------------
RESET <= '1', '0' after 100 ns;      -- nebenlaeuf. Reset-Impuls am Anfang
---------------------------
TAKT: process                        -- Taktprozess
begin
       CLK <='0';
       wait for 100 ns;
       CLK <= '1';
       wait for 100 ns;
end process TAKT;
---------------------------
P1: process (CLK, RESET)             -- Johnson-Zaehler
begin
   if RESET='1' then
TEMP <= "000";
FTEIL <= "11";
   elsif CLK='1' and CLK'event then
       case TEMP is
              when "000"  => TEMP <= "001";
                             FTEIL <= "01";
              when "001"  => TEMP <= "011";
                             FTEIL <= "10";
              when "011"  => TEMP <= "111";
                             FTEIL <= "00";
              when "111"  => TEMP <= "110";
                             FTEIL <= "10";
              when "110"  => TEMP <= "100";
                             FTEIL <= "00";
              when "100"  => TEMP <= "000";
                             FTEIL <= "11";
              when others => TEMP <= "111";
                             FTEIL <= "11";
     end case;
    end if;
end process P1;
COUNT <= TEMP;
end ARCH1;
```

3.7

```
-- Flipflop-Varianten
entity FF_TEST is
  port ( CLK, RESET, PRESET, ENABLE, DATEN: in bit;
          Q_A, Q_B, Q_C: out bit);
end FF_TEST;
architecture VERHALTEN of FF_TEST is
begin
A:     process(CLK, RESET, PRESET)
       begin
              if RESET='1' then
                     Q_A <= '0';
              elsif PRESET='1' then
                     Q_A <= '1';
              elsif CLK'event and CLK='0' then
                     Q_A <= DATEN;
              end if;
       end process A;
B:     process(CLK, RESET)
       begin
              if RESET='1' then
                     Q_B <= '0';
              elsif CLK'event and CLK='0' then
                     if PRESET='1' then
                            Q_B <= '1';
```

```
                        else
                                Q_B <= DATEN;
                        end if;
                end if;
        end process B;
C:      process(CLK, RESET)
        begin
                if RESET='1' then
                        Q_C <= '0';
                elsif CLK'event and CLK='0' then
                        if PRESET='1' then
                                Q_C <= '1';
                        elsif ENABLE='0' then
                                Q_C <= DATEN;
                        end if;
                end if;
        end process C;
end VERHALTEN;
```

3.8

```
-- 4-Bit Schieberegister mit Parallelausgang
-------------------------------------------
entity SREG4BIT is
        port( DIN, CLK, RESET: in bit;
              DOUT: out bit_vector(3 downto 0));
end SREG4BIT;
architecture VERHALTEN of SREG4BIT is
signal Q_INT: bit_vector(3 downto 0);
begin
P1:     process(CLK, RESET)
        begin
                if RESET='1' then
                        Q_INT <="0000";               -- Loeschen
                elsif CLK='1' and CLK'event then      -- Schieben
                        Q_INT(3) <= Q_INT(2);
                        Q_INT(2) <= Q_INT(1);
                        Q_INT(1) <= Q_INT(0);
                        Q_INT(0) <= DIN;
                end if;
        end process P1;
DOUT <= Q_INT;
end VERHALTEN;
```

3.9

```
-- Primzahlgenerator
entity PRIM_GEN is
  port ( CLK, RESET: in bit;
           Q: out bit_vector(3 downto 0));
end PRIM_GEN;
architecture VERHALTEN of PRIM_GEN is
signal Q_INT: bit_vector(3 downto 0);
begin
P1:     process(CLK, RESET)
        begin
                if RESET='1' then
                        Q_INT <= x"1";
                elsif CLK'event and CLK='0' then
                        case Q_INT is
                                when x"1" => Q_INT <= x"2";
                                when x"2" => Q_INT <= x"3";
                                when x"3" => Q_INT <= x"5";
                                when x"5" => Q_INT <= x"7";
                                when x"7" => Q_INT <= x"B";
                                when x"B" => Q_INT <= x"D";
                                when x"D" => Q_INT <= x"1";
                                when others => Q_INT <= x"1"; -- Reset Zustand
                        end case;
                end if;
        end process P1;
        Q <= Q_INT;
end VERHALTEN;
```

3.10

FEHLER_A:

- Die Variable TEMP wurde unzulässigerweise in der `architecture` deklariert.
- Der Prozess P1 enthält eine unzulässige nebenläufige, bedingte Signalzuweisung.
- Die Variable wird unzulässigerweise außerhalb des taktsynchronen Umfelds gelesen.
- Das Signal SEL befindet sich nicht in den Empfindlichkeitslisten beider Prozesse.
- Im Prozess P2 darf keine Variable in der Empfindlichkeitsliste stehen.

FEHLER_B:

- Die im Prozess P1 deklarierte Variable TEMP ist im Prozess P2 unbekannt.
- In der Empfindlichkeitsliste von Prozess P2 fehlt SEL und TEMP darf nicht enthalten sein.
- Die `architecture` enthält eine sequentielle `if`-Anweisung außerhalb eines Prozesses.

3.11

Das Ausgangssignal ist konstant '0'.
Begründung: SIG (und damit SIGOUT) wird mit '0' initialisiert. Während CLK='1' ist, durchläuft der Simulator zwar die unbedingte Anweisung SIG <= '1'. Diese würde allerdings, falls dann noch gültig, erst am Ende des Prozesses ausgeführt. In der `if`-Abfrage wird somit nicht der `then`- sondern der `else`-Zweig durchlaufen. In diesem Zweig wird SIG aus der Antivalenz von CLK und SIG gebildet, diese ist aber '0', da CLK='1' und SIG='0' ist. Somit bleibt unverändert SIG='0'.

3.12

```
entity TEST is
port( CLK, A1, A2, A3: in bit;
       S: buffer bit_vector(3 downto 0));
end TEST;
architecture UEBUNG of TEST is
begin
P0:    process (A1, A2, A3)
       variable VAR: bit;
       begin
              if A1 ='1' then
                     VAR := A2 and A3;     -- Latch oder komb. Logik
                     S(0) <= VAR;          -- Latch
              end if;
       end process P0;
P1:    process (A1)
       variable VAR: bit;
       begin
              VAR:='0';
              S(1)<='0';
              if A1 ='1' then
                     VAR := VAR or A2;     -- komb. Logik
                     S(1) <= VAR and A3;   -- komb. Logik
              end if;
       end process P1;
P2:    process (CLK)
       variable VAR: bit;
       begin
              if CLK'event and CLK='1' then
                     VAR := VAR and A1;    -- Flipflop
                     S(2) <= VAR and A3;   -- Flipflop
              end if;
       end process P2;
P3:    process (CLK)
       variable VAR: bit;
       begin
              if CLK'event and CLK='1' then
                     VAR := S(3) and A1;   -- komb. Logik
                     S(3) <= VAR;          -- Flipflop
       end if;
       end process P3;
end UEBUNG;
```

Lösungen zu den Aufgaben in Kap. 4.6

4.1

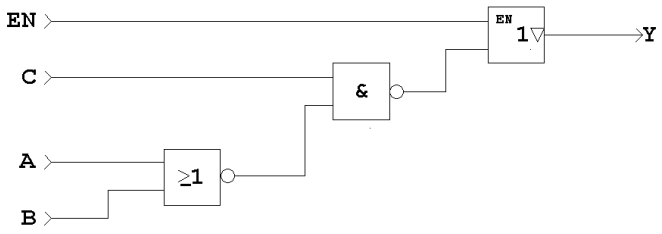
a) vgl. Tabelle 4-1

b) vgl. Kap. 4.1 und Tabelle 5-3

4.2

```
-- Tri-State Logik
---------------------------------
library ieee;
use ieee.std_logic_1164.all;
entity TS_LOGIK is
       port( EN, A, B, C : in bit;
             Y: out std_ulogic);
end TS_LOGIK;
architecture VERHALTEN of TS_LOGIK is
begin
       process(EN, A, B, C)
       begin
             if EN='1' then
                    Y <=   to_stdulogic(( A and B and C) or
                           ( A or B or not C));
             else
                    Y <= 'Z';
             end if;
       end process;
end VERHALTEN;
```

Die Synthese erfordert zwei Gatter sowie einen Tri-State-Treiber:



4.3

Der Prozess wird aktiviert, sofern sich eins der Eingangssignale ändert. Es werden zunächst alle Signale hochohmig gesetzt. In einer geschachtelten `if`-Abfrage wird zunächst das Freigabesignal abgefragt. Wenn dieses '1' ist, so wird der zum Wert des Signals SEL gehörige Ausgang mit dem Eingang DIN verbunden und somit der Z-Wert überschrieben.

```
-- 1 zu 4 Demultiplexer
-------------------------------------
library ieee;
use ieee.std_logic_1164.all;
entity DEMUX is
       port( DIN, EN: in std_ulogic;
             SEL: in bit_vector(1 downto 0);
             DOUT0, DOUT1, DOUT2, DOUT3: out std_ulogic);
end DEMUX;
architecture VERHALTEN of DEMUX is
begin
P1:    process(SEL, DIN, EN)
       begin
             DOUT0<='Z'; DOUT1<='Z'; DOUT2<='Z'; DOUT3<='Z'; -- Alle Z
             if EN = '0' then                        -- Falls Enable
                    if SEL = "00" then               -- jeweiligen
                           DOUT0 <= DIN;             -- Ausgang
                    elsif SEL= "01" then             -- durchschalten
                           DOUT1 <= DIN;
                    elsif SEL= "10" then
                           DOUT2 <= DIN;
```

```
                        else
                                DOUT3 <= DIN;
                        end if;
                end if;
        end process P1;
end VERHALTEN;
```

4.4

```
-- PLD-Ausgangszelle
---------------------------------
library ieee;
use ieee.std_logic_1164.all;
entity PLD_CELL is
        port( PROD_TERM, CLK, RESET : in bit;
                S: in bit_vector(2 downto 0);
                IO_PAD: inout std_logic;
                FEEDBACK: out bit);
end PLD_CELL;
architecture VERHALTEN of PLD_CELL is
signal TEMP0, TEMP1: bit;
signal TEMP2_V: bit_vector(0 downto 0);
signal FEEDBACK_V: bit_vector(0 downto 0);
signal IO_PAD_V: std_logic_vector(0 downto 0);

begin
-- Polaritaetsumkehr durch Antivalenz-Gatter:
TEMP0 <= PROD_TERM xor S(2);

-- Flipflop als Prozess
FF: process( CLK, RESET)
        begin
                if RESET='1' then
                        TEMP1 <= '0';
                elsif CLK'event and CLK='1' then
                        TEMP1 <= TEMP0;
        end if;
end process FF;

-- Multiplexer als nebenlaeufige Anweisung
with S(1) select TEMP2_V(0)<= TEMP0 when '0', TEMP1 when '1';

-- Tri-State Treiber
TS: process(TEMP2_V(0), S(0))
        begin
                if S(0)='1' then
                        IO_PAD_V <= To_StdLogicVector(TEMP2_V);
                else
                        IO_PAD_V(0) <='Z';
                end if;
        end process;

-- IO-Pad als Ausgang:
IO_PAD <= IO_PAD_V(0);

-- IO-PAD als Eingang:
FEEDBACK_V <= To_bitvector(IO_PAD_V);
FEEDBACK <= FEEDBACK_V(0);
end VERHALTEN;
```

4.5

```
-- Code-Umsetzer in den Aiken Code
-----------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
entity AIKEN is
        port( E: in bit_vector(3 downto 0);
                Y: out std_ulogic_vector(3 downto 0));
end AIKEN;
architecture VERHALTEN of AIKEN is
begin
P1:     process(E)
        begin
                case E is
                when "0000" => Y <= "0000";
                when "0001" => Y <= "0001";
```

```
                when "0010" => Y <= "0010";
                when "0011" => Y <= "0011";
                when "0100" => Y <= "0100";
                when "0101" => Y <= "1011";
                when "0110" => Y <= "1100";
                when "0111" => Y <= "1101";
                when "1000" => Y <= "1110";
                when "1001" => Y <= "1111";
                when others => Y <= "----";
                end case;
        end process P1;
end VERHALTEN;
```

4.6

```
-- Wahrheitstab. mit Don't-Care Ein- und Ausgaengen
-------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
entity WAHRTAB is
        port( E: in bit_vector(2 downto 0);
                Y: out std_ulogic_vector(2 downto 0));
end WAHRTAB;
architecture VERHALTEN of WAHRTAB is
begin
P1:     process(E)
        begin
                case E is
                when "000" | "001" => Y <= "11-";
                when "010" => Y <= "101";
                when "011" => Y <= "100";
                when "100" | "101" => Y <= "01-";
                when "110" => Y <= "001";
                when "111" => Y <= "000";
                end case;
        end process P1;
end VERHALTEN;
```

Wenn der Quellcode zu drei Invertern (je einer für die drei Eingänge) synthetisiert wurde und keine anderen Gatter verwendet werden, so hat eine Minimierung stattgefunden.

Lösungen zu den Aufgaben in Kap. 5.9

5.1

```
-- Multiplikation zweier 3 Bit Zahlen
-------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;           -- IEEE
-- use ieee.std_logic_signed.all;   -- Synopsys/Viewlogic
-- Bei Verwendung der std_logic_signed Bibliothek muessen anstatt
-- der "signed bzw. unsigned" Signale "std_logic_vector" Signale verwendet
-- werden!

entity MULT is
        port( A, B: in signed(2 downto 0);
              Y: out signed(5 downto 0));
end MULT;
architecture VERHALTEN of MULT is
begin
        Y <= A * B;
end VERHALTEN;
```

Syntheseergebnis: Es wird 3x3-Bit-Multiplizierer-Funktionsblock generiert.

5.2

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

```
use ieee.numeric_std.all;            -- IEEE
-- use ieee.std_logic_signed.all;    -- Synopsys/Viewlogic
-- Bei Verwendung der std_logic_signed Bibliothek muessen anstatt
-- der "signed bzw. unsigned" Signale "std_logic_vector" Signale verwendet
-- werden!
entity ABSWERT is
       port( A: in signed(4 downto 0);
             Y: out unsigned(4 downto 0));
end ABSWERT;
architecture VERHALTEN of ABSWERT is
begin
       Y <= unsigned(abs(A));
end VERHALTEN;
```

Einige Synthesewerkzeuge setzen die Absolutwertbildung recht aufwendig um. Z.B. generiert Aurora der Fa. Viewlogic einen 3-Bit-Subtrahierer in Zusammenhang mit einem Schaltnetz. Grundlage für die einfachen Lösungen ist die Tatsache, dass das Zweierkomplement aus der Inversion aller Bitstellen mit anschließender Addition von '1' gebildet wird [90].

5.3

```
-- 4-Bit ALU
-------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;                          -- IEEE1076.3
-- use ieee.std_logic_unsigned.all;                -- Viewlogic / Synopsys
entity ALU4 is
       port( A, B: in unsigned(3 downto 0);        --4-Bit Operanden
             OPCODE: in bit_vector(1 downto 0);    --2-Bit OPCODE
             RESULT: out unsigned(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: unsigned(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;
```

5.4

```
-- N-Bit Vorwärts-/Rückwärtszähler
---------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;            -- IEEE
--use ieee.std_logic_signed.all;     -- Synopsys / Viewlogic

entity VRZLR is
       generic( N: natural:=4);
```

```
        port( CLK, RESET, UND: in bit;
              Q: out unsigned(N-1 downto 0));
end VRZLR;

architecture VERHALTEN of VRZLR is
signal Q_INT : unsigned(N-1 downto 0);       -- internes Signal
begin
P1:     process(CLK, RESET)
        begin
                if RESET='1' then                       -- Asynchron
                        Q_INT <= (others=>'0');         -- loeschen
                elsif CLK='1' and CLK'event then        -- Synchron zaehlen
                        if UND ='1' then
                                Q_INT <= Q_INT+1;       -- Aufwaerts
                        else
                                Q_INT <= Q_INT-1;       -- Abwaerts
                        end if;
                end if;
        end process P1;
        Q <= Q_INT;                                     -- Ausgangssignal
end VERHALTEN;
```

5.5

```
-- gesteuerter Zaehler
---------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;            -- IEEE
--use ieee.std_logic_signed.all;     -- Synopsys / Viewlogic
entity ZLR1 is
        port( CLK, RESET: in bit;
              MODE: in  bit_vector(1 downto 0);
              Q: out unsigned(3 downto 0));
end ZLR1;

architecture VERHALTEN of ZLR1 is
signal Q_INT : unsigned(3 downto 0);         -- internes Signal
begin
P1:     process(CLK)
        begin
                if CLK='1' and CLK'event then        -- Synchron
                        if RESET='1' then
                                Q_INT <= (others=>'0');-- loeschen
                        elsif MODE = "00" then
                                Q_INT <= Q_INT+1;    -- um eins
                        elsif MODE = "01" then
                                Q_INT <= Q_INT+2;    -- um zwei
                        elsif MODE = "10" then
                                Q_INT <= Q_INT+3;    -- um drei
                        elsif MODE = "11" then
                                Q_INT <= Q_INT+4;    -- um vier
                        end if;
                end if;
        end process P1;
        Q <= Q_INT;                                  -- Ausgangssignal
end VERHALTEN;
```

5.6

```
-- Programmierbarer Frequenzteiler 2..15
-----------------------------------------
entity TEILER is
        port( CLK, RESET: in bit;
              N: in integer range 2 to 15;
              TC: out bit);
end TEILER;

architecture VERHALTEN of TEILER is
signal Q: integer range 0 to 15;
begin
ZLR:    process(CLK, RESET)
        begin
                if RESET='1' then Q <= 0;
                elsif CLK='1' and CLK'event then
                        if Q >= N-1 then
                                Q <= 0;         -- Ruecksetzen
```

```
                        TC <= '1';      -- Ausgangssignal setzen
                    else
                        Q <= Q + 1;     -- Zaehlen
                        TC <= '0';      -- Ausgangssignal loeschen
                    end if;
                end if;
        end process ZLR;
end VERHALTEN;
```

5.7

```
-- Ladbarer N-Bit Zaehler mit Enable
-------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;           -- IEEE
entity GEN_CTR is
        generic( BITS: natural := 4);
        port(   CLK, RESET, LOAD, ENABLE: in bit;
                D: in unsigned (BITS-1 downto 0);
                Q: out unsigned (BITS-1 downto 0));
end GEN_CTR;

architecture VERHALTEN of GEN_CTR is
signal QINT: unsigned(BITS-1 downto 0);
begin
CTR:    process (CLK, RESET)
        begin
                if RESET ='1' then
                        QINT <= (others => '0');
                elsif CLK='1' and CLK'event then
                        if ENABLE='1' then
                                if LOAD = '1' then
                                        QINT <= D;
                                else
                                        QINT <= QINT + 1;
                                end if;
                        end if;
                end if;
        end process CTR;
        Q <= QINT;              -- Kopie an den Ausgang
end VERHALTEN;
```

5.8

a) 199	Korrekt
b) 8#AFFE#	Falsch, da HEX-Konstanten bei Oktaler Basis
c) 2#1011_1010#	Korrekt
d) 16#ABCD#	Korrekt
e) 5#224_33#	Korrekt

5.9

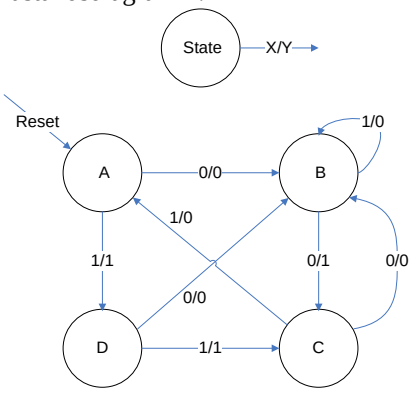
```
-- Pixeladressierung
--------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.mumeric_std.all;
entity PIX_ADR is
        port( CLK, RESET: in bit;
                Z_IND, SP_IND: in bit_vector(6 downto 0);
                RNW: in bit;
                D_IN: in std_logic;
                D_OUT: out std_logic);
end PIX_ADR;

architecture VERHALTEN of PIX_ADR is
signal MEM: std_logic_vector(0 to 16383);
begin
P1:     process(CLK, RESET)
        variable Z_IND_VAR, SP_IND_VAR: integer range 0 to 127;
```

```
        variable INDEX: integer range 0 to 16383;
        begin
          if RESET='1'then
                MEM <= (others=>'0');
          elsif CLK'event and CLK='1' then
                Z_IND_VAR := To_integer(unsigned(To_StdLogicVector(Z_IND)));
                SP_IND_VAR := To_integer(unsigned(To_StdLogicVector(SP_IND)));
                INDEX := 128 * Z_IND_VAR + SP_IND_VAR; --Index
                if RNW='1' then
                        D_OUT <= MEM(INDEX);   -- Speicher lesen
                else
                        MEM(INDEX) <= D_IN;    -- Speicher schreiben
                end if;
        end if;
end process P1;
end VERHALTEN;
```

Lösungen zu den Aufgaben in Kap. 7.6

Hinweis: Einige Synthesewerkzeuge unterstützen keine Parameter vom Typ `time`, sodass die Zeiten direkt in die Signalzuweisungen einzutragen sind, wenn sie vom Synthesewerkzeug überlesen werden sollen.

7.1
Zustandsdiagramm:




```
-- FSM 2 Prozesse:
entity FSM_2_ME is
generic(        TD1 : time := 10 ns;
                TD2 : time := 20 ns);
port(   CLK, RESET, ENABLE    : in  bit;       -- sekundäre Eingangssignale
        X               : in  bit;
        Y                  : out bit );
end FSM_2_ME;

architecture SEQUENZ of FSM_2_ME is
type ZUSTAENDE is (ZA, ZB, ZC, ZD);          -- Aufzählungstyp
attribute ENUM_ENCODING: STRING;
attribute ENUM_ENCODING of ZUSTAENDE: type is "00 01 11 10";
signal ZUSTAND,FOLGE_Z: ZUSTAENDE ;
begin
Z_SPEICHER: process(CLK, RESET) -- Zustandsaktualisierung
   begin
        if  RESET = '1'  then
                        ZUSTAND <= ZA after TD1;
        elsif CLK = '1' and CLK'event then
                        if ENABLE = '1' then
                                ZUSTAND <= FOLGE_Z after TD1;
                        end if;
        end if;
end process Z_SPEICHER;
UE_AUS_SN: process(X, ZUSTAND)-- Folgezustands- u. Ausgangsberechnung
   begin
        FOLGE_Z <= ZB after TD2;       -- Defaultzuweisung
        Y <= '0' after TD2;
        case ZUSTAND is
                        when ZA =>      if    X = '1' then
                                        FOLGE_Z<= ZD after TD2;
```

```
                                    Y <= '1'after TD2;
                                    end if;
                        when ZB =>  if    X= '0' then
                                    FOLGE_Z<= ZC after TD2;
                                    Y <= '1'after TD2;
                                    end if;
                        when ZC =>  if    X= '1' then
                                    FOLGE_Z<= ZA after TD2;
                                    end if;
                        when ZD =>  if    X= '1' then
                                    FOLGE_Z<= ZC after TD2;
                                    Y <= '1'after TD2;
                                    end if;
        end case;
end process UE_AUS_SN;
end SEQUENZ;
```

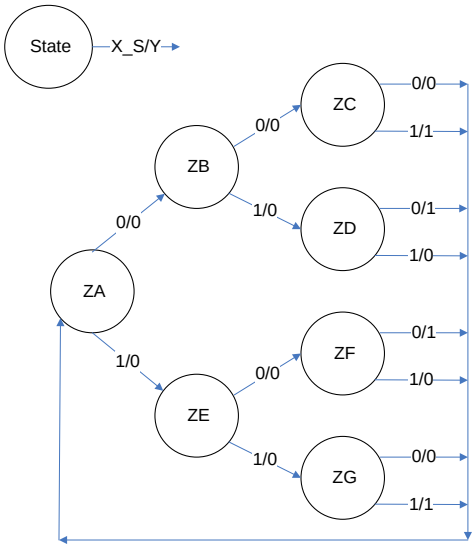
7.2

```
-- FSM 3 Prozesse:
entity FSM_3_MO is
generic(       TD1 : time := 10 ns;  -- D-FF Laufzeit
               TD2 : time := 20 ns;  -- Schaltnetzlaufzeit
               TD3 : time := 30 ns); -- Kette aus TD1 und TD2
port(  CLK, RESET, ENABLE   : in  bit;      -- sekundäre Eingangssignale
       X            : in  bit;
       Y              : out bit );
end FSM_3_MO;

architecture SEQUENZ of FSM_3_MO is
type ZUSTAENDE is (ZA, ZB, ZC, ZD);         -- Aufzählungstyp
attribute ENUM_ENCODING: STRING;
attribute ENUM_ENCODING of ZUSTAENDE: type is "00 01 11 10";
signal ZUSTAND,FOLGE_Z: ZUSTAENDE ;
signal X_S: bit;                    -- Synchronisiertes Eingangssignal
begin
SYNC: process(CLK, RESET)
   begin
       if RESET = '1'        then
              X_S <= '0' after TD1;
              Y   <= '0' after TD1;
       elsif CLK = '1' and CLK'event then
              X_S <= X after TD1;   -- Eingangssignalsynchronisation
              Y <= '0' after TD3;   -- Default-Zuweisung
              case FOLGE_Z is
                     when ZD =>  Y <= '1' after TD3;-- Unabhängig von X_S
                     when ZC =>  Y <= '1' after TD3;
                     when others => null;
              end case;
       end if;
   end process SYNC;
Z_SPEICHER: process(CLK, RESET) -- Zustandsaktualisierung
   begin
       if  RESET = '1'  then
                    ZUSTAND <= ZA after TD1;
       elsif CLK = '1' and CLK'event then
                    if ENABLE = '1' then
                           ZUSTAND <= FOLGE_Z after TD1;
                    end if;
       end if;
end process Z_SPEICHER;
UE_SN: process(X_S, ZUSTAND)-- Folgezustandsberechnung
   begin
       FOLGE_Z <= ZB after TD2;     -- Defaultzuweisung
       case ZUSTAND is
                    when ZA =>  if    X_S = '1' then
                                FOLGE_Z<= ZD after TD2;
                                end if;
                    when ZB =>  if    X_S= '0' then
                                FOLGE_Z<= ZC after TD2;
                                end if;
                    when ZC =>  if    X_S= '1' then
                                FOLGE_Z<= ZA after TD2;
                                end if;
                    when ZD =>  if    X_S= '1' then
```

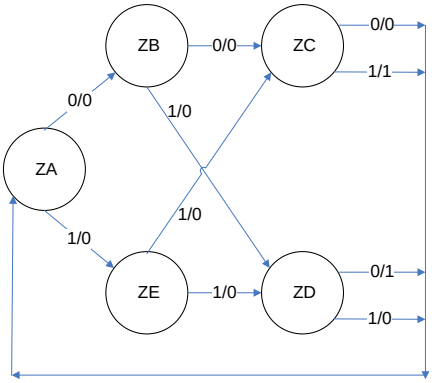
```
                                    FOLGE_Z<= ZC after TD2;
                                    end if;
        end case;
end process UE_SN;
end SEQUENZ;
```

7.3

Ursprüngliches Zustandsdiagramm mit 7 Zuständen:



Minimiertes Zustandsdiagramm: Die Zustände ZC und ZG sowie ZD und ZF können zusammengefasst werden, da sie gleiches Ausgangssignal- und Folgezustandsverhalten haben .




```
-- FSM 3 Prozesse: Paritäts-Checker
entity FSM_CHECK is
generic(       TD1 : time := 10 ns;  -- D-FF Laufzeit
               TD2 : time := 20 ns); -- Schaltnetzlaufzeit
port(  CLK, RESET, ENABLE    : in  bit;     -- sekundäre Eingangssignale
       X                     : in  bit;
       Y_S                   : out bit );
end FSM_CHECK;

architecture SEQUENZ of FSM_CHECK is
type ZUSTAENDE is (ZA, ZB, ZC, ZD, ZE, ZF, ZG, ZH); -- Aufzählungstyp
attribute ENUM_ENCODING: STRING;
attribute ENUM_ENCODING of ZUSTAENDE: type is "000 100 110 111 101 001 010 011";
signal ZUSTAND,FOLGE_Z: ZUSTAENDE:= ZA; -- Power-up of State Register
attribute SAFE_RECOVERY_STATE: STRING;
attribute SAFE_RECOVERY_STATE of ZUSTAND: signal is "ZA";
signal X_S: bit;                        -- Synchronisiertes Eingangssignal
signal Y: bit;
begin

SYNC: process(CLK, RESET)
   begin
        if RESET = '1'          then
               X_S     <= '0' after TD1;
               Y_S     <= '0' after TD1;
        elsif CLK = '1' and CLK'event then
               X_S     <=  X  after TD1; -- Eingangssignalsynchronisation
               Y_S     <=  Y  after TD1; -- Ausgangssignalsynchronisation
        end if;
   end process SYNC;
Z_SPEICHER: process(CLK, RESET) -- Zustandsaktualisierung
   begin
        if  RESET = '1'  then
                       ZUSTAND <= ZA after TD1;
        elsif CLK = '1' and CLK'event then
                       if ENABLE = '1' then
                              ZUSTAND <= FOLGE_Z after TD1;
                       end if;
```

```
       end if;
end process Z_SPEICHER;
UE_AUS_SN: process(X_S, ZUSTAND)-- Folgezustands- u. Ausgangsberechnung
   begin
       FOLGE_Z <= ZA after TD2;      -- Defaultzuweisung
       Y <= '0' after TD2;
       case ZUSTAND is
                     when ZA =>     if      X_S = '0' then
                                            FOLGE_Z<= ZB after TD2;
                                    else
                                            FOLGE_Z<= ZE after TD2;
                                    end if;
                     when ZB =>     if      X_S= '0' then
                                            FOLGE_Z<= ZC after TD2;
                                    else
                                            FOLGE_Z<= ZD after TD2;
                                    end if;
                     when ZC =>     if      X_S= '1' then
                                            y <= '1' after TD2;
                                    end if;
                     when ZD =>     if      X_S= '0' then
                                            y <= '1' after TD2;
                                    end if;
                     when ZE =>     if      X_S= '0' then
                                            FOLGE_Z<= ZD after TD2;
                                    else
                                            FOLGE_Z<= ZC after TD2;
                                    end if;
                     when others => null; -- für ZF, ZG u. ZH
        end case;
end process UE_AUS_SN;
end SEQUENZ;
```

Lösungen zu den Aufgaben in Kap. 8.5

8.1

```
--Johnson-Zähler mit Schieberegister SRG_4.
entity SRG_4 is
port(  RESET, CLK     : in bit; -- D_PL     : in bit_vector (3 downto 0);
       Q              : out bit_vector (3 downto 0));
end SRG_4;

----- als Komponente verwendet: ------------
entity D_FF is
       port(  RESET, CLK    : in  bit;
              D_IN          : in  bit;
              Q_OUT            : out bit );
end D_FF;

architecture VERHALTEN of D_FF is
begin
process(CLK, RESET) -- Zustandsaktualisierung
       begin
       if RESET = '1' then
              Q_OUT <= '0';
       elsif CLK = '1' and CLK'event then
              Q_OUT <= D_IN ;
       end if;
end process;
end VERHALTEN;
-----------------------------------------
architecture STRUK of SRG_4 is
signal SER_IN   : bit;
signal Q_INT : bit_vector(3 downto 0);
for all: D_FF use entity WORK.D_FF(VERHALTEN);
begin
SER_IN <= not Q_INT(3);
Q <= Q_INT;
I_0: entity work.D_FF
     port map(RESET => RESET, CLK => CLK, D_IN => SER_IN, Q_OUT => Q_INT(0));
I_1: entity work.D_FF
     port map(RESET => RESET, CLK => CLK, D_IN => Q_INT(0), Q_OUT => Q_INT(1));
I_2: entity work.D_FF
     port map(RESET => RESET, CLK => CLK, D_IN => Q_INT(1), Q_OUT => Q_INT(2));
```

```
I_3: entity work.D_FF
     port map(RESET => RESET, CLK => CLK, D_IN => Q_INT(2), Q_OUT => Q_INT(3));
end STRUK;
```

8.2

```
--Johnson-Zähler mit Schieberegister SRG_N.
entity SRG_N is
generic(N : positive := 4);
port(  RESET, CLK     : in bit;
       D_PL           : in bit_vector (N - 1 downto 0);
       Q              : out bit_vector (N - 1 downto 0));
end SRG_N;
----- als Komponenten verwendet: ----------
entity KOR_SN is
       port(  I  : in  bit_vector (1 downto 0);
              A  : out bit_vector (1 downto 0));
end KOR_SN;

architecture VERHALTEN of KOR_SN is
begin
A(0) <= not I(1);
A(1) <= I(0) NOR I(1);
end VERHALTEN;

entity D_FF_L is
       port(  RESET, CLK, LOAD, D_IN, D_L   : in  bit;
              Q_OUT                         : out bit );
end D_FF_L;

architecture VERHALTEN of D_FF_L is
begin
process(CLK, RESET) -- Zustandsaktualisierung
       begin
       if RESET = '1' then
              Q_OUT <= '0';
       elsif CLK = '1' and CLK'event then
              if LOAD = '1' then
                     Q_OUT <= D_L;
              else
                     Q_OUT <= D_IN;
              end if;
       end if;
end process;
end VERHALTEN;
-----------------------------------------
architecture STRUK of SRG_N is
component D_FF_L
port(  RESET, CLK, LOAD, D_IN, D_L   : in  bit;
              Q_OUT                         : out bit );
end component;
component KOR_SN
port(  I  : in  bit_vector (1 downto 0);
       A  : out bit_vector (1 downto 0));
end component;
signal SER_I, L_INT   : bit; -- Serieller Eingang, Ladefunktion
signal Q_INT : bit_vector(N downto 0);
for all: D_FF_L use entity WORK.D_FF_L(VERHALTEN);
for I_0: KOR_SN use entity WORK.KOR_SN(VERHALTEN);
begin
Q <= Q_INT;
I_0:   KOR_SN port map(I(0) => Q_INT(0), I(1) => Q_INT(N - 1),A(0) => SER_I, A(1) => L_INT );
I_K:   for K in 0 to N - 1 generate
       I_A:   if K = 0 generate
              I_D:   D_FF_L
              port map(RESET => RESET,   CLK => CLK, LOAD => L_INT, D_IN => SER_I, D_L =>
D_PL(K), Q_OUT => Q_INT(K));
              end generate I_A;
       I_R:   if K > 0 generate
              I_D: D_FF_L
              port map(RESET, CLK, L_INT, Q_INT(K - 1), D_PL(K), Q_INT(K));
              end generate I_R;
       end generate I_K;
end STRUK;
```

8.3

Hinweis: Einige Synthesewerkzeuge unterstützen keine Parameter vom Typ `time`, sodass die Zeiten direkt in die Signalzuweisungen einzutragen sind, wenn sie vom Synthesewerkzeug überlesen werden sollen.

```
-- 4-Bit Volladdierer Strukturmodell
entity ADD_S1 is
       generic(WB : POSITIVE :=4;    -- Wortbreite
                DEL: TIME     := 15 ns);
       port( A, B: in  bit_vector(WB-1 downto 0);
             SUM : out  bit_vector(WB   downto 0));
end ADD_S1;

architecture STRUKTUR of ADD_S1 is
signal RIPPLE_CARRY: bit_vector (WB-2 downto 0);
signal ZERO     : bit ;
component VOLL_ADD    -- Komponenten-Deklaration: verwendeter IC-Typ
       generic( DELAY : TIME);
       port(  C_IN, IN1, IN2 : in  bit ;
              S, C_OUT              : out bit );
end component;
              -- Konfiguration: Board wird mit Chip bestückt
for all: VOLL_ADD use entity WORK.VOLL_ADD(VERHALTEN);
begin         -- Instanziierung der Komponente: Adaptersockel-Platzierung
ZERO <= '0';
VA_LSB: VOLL_ADD
              generic map(DELAY => DEL)
              port map(      C_IN           => ZERO,
                             IN1            => A(0),
                             IN2            => B(0),
                             S              => SUM(0),
                             C_OUT   => RIPPLE_CARRY(0));
VA_K: for N in 1 to  WB-2 generate
       VA_I: VOLL_ADD
              generic map(DELAY => DEL)
              port map(RIPPLE_CARRY(N-1), A(N), B(N), SUM(N), RIPPLE_CARRY(N));
       end generate;
VA_MSB: VOLL_ADD
              generic map(DELAY => DEL)
              port map(RIPPLE_CARRY(WB-2), A(WB-1), B(WB-1), SUM(WB-1), SUM(WB));
end STRUKTUR;

-- Volladdierer mit Wahrheitstafel: Aufgabe 7.3
-- Aggregat mit BIT-Elementen zur Array-Bildung
entity VOLL_ADD is
       generic( DELAY : TIME := 15 ns);
       port( C_IN,IN1 ,IN2: in  BIT;
              S, C_OUT     : out BIT );
end VOLL_ADD;

architecture VERHALTEN of VOLL_ADD is
begin
process(C_IN,IN1,IN2)
subtype BV3_TYPE is BIT_VECTOR (2 downto 0);
begin
-- Der Type-Qualifier BIT_VECTOR' kennzeichnet den Typ des Aggregates
-- auf der rechten Seite.
       case    BV3_TYPE'(C_IN,IN1,IN2) is    --Aggregat nur mit Type-Qualif.
              when "001"  => (C_OUT,S) <= BIT_VECTOR'("01")after DELAY;
              when "010"  => (C_OUT,S) <= BIT_VECTOR'("01")after DELAY;
              when "011"  => (C_OUT,S) <= BIT_VECTOR'("10")after DELAY;
              when "100"  => (C_OUT,S) <= BIT_VECTOR'("01")after DELAY;
              when "101"  => (C_OUT,S) <= BIT_VECTOR'("10")after DELAY;
              when "110"  => (C_OUT,S) <= BIT_VECTOR'("10")after DELAY;
              when "111"  => (C_OUT,S) <= BIT_VECTOR'("11")after DELAY;
              when others => (C_OUT,S) <= BIT_VECTOR'("00")after DELAY;
-- In den Ausgangszuordnungen wird der Type-Qualifier erforderlich, damit
-- der BIT_VECTOR-String von einem Character-String unterschieden wird.
       end case;
end process;
end VERHALTEN;
```

8.4

```
-- 4-Bit Volladdierer Strukturmodell
entity ADD_S2 is
```

```
        generic(WB : POSITIVE :=4;    -- Wortbreite
                DEL: TIME     := 15 ns);
        port( A, B: in  bit_vector(WB-1 downto 0);
              SUM : out  bit_vector(WB   downto 0));
end ADD_S2;

architecture STRUK_HL of ADD_S2 is
signal RIPPLE_CARRY: bit_vector (WB-2 downto 0);
signal ZERO     : bit ;
component V_ADD       -- Komponenten-Deklaration: verwendeter IC-Typ
        generic( DELAY : TIME);
        port(  C_IN, IN1, IN2 : in  bit ;
               S, C_OUT              : out bit );
end component;
               -- Konfiguration: Board wird mit Chip bestückt
for all: V_ADD use entity WORK.V_ADD(VERHALTEN);
begin          -- Instanziierung der Komponente: Adaptersockel-Platzierung
ZERO <= '0';
VA_LSB: V_ADD
               generic map(DELAY => DEL)
               port map(      C_IN           => ZERO,
                              IN1            => A(0),
                              IN2            => B(0),
                              S              => SUM(0),
                              C_OUT   => RIPPLE_CARRY(0));
VA_K: for N in 1 to  WB-2 generate
        VA_I: V_ADD
               generic map(DELAY => DEL)
               port map(RIPPLE_CARRY(N-1), A(N), B(N), SUM(N), RIPPLE_CARRY(N));
        end generate;
VA_MSB: V_ADD
               generic map(DELAY => DEL)
               port map(RIPPLE_CARRY(WB-2), A(WB-1), B(WB-1), SUM(WB-1), SUM(WB));
end STRUK_HL;

-- Volladdierer Strukturmodell
-- Aufgabe 7.4
entity V_ADD is
        generic(DELAY : TIME := 30 ns);
        port( C_IN,IN1 ,IN2: in  bit;
              S, C_OUT     : out bit );
end V_ADD;

architecture STRUK_LL of V_ADD is
signal PROP, GEN : bit;  -- Propagate, Generate
signal ZERO      : bit;
component H_ADD
generic(TD : TIME);
port(X1, X2, X3: in bit;
     Q1, Q2:     out bit);
end component;
for all: H_ADD use entity WORK.H_ADD(VERHALTEN);
begin
ZERO <= '0';
C1: H_ADD
generic map(TD => DELAY)
port map(X1 => IN1, X2 => IN2, X3 => ZERO,
         Q1 => PROP, Q2 => GEN);
C2: H_ADD
generic map(TD => DELAY)
port map(X1 => PROP, X2 => C_IN, X3 => GEN,
          Q1 => S, Q2 => C_OUT);
end STRUK_LL;

-- Halbaddierer Verhaltensmodell
entity H_ADD is
        generic(TD : TIME := 20 ns);
        port( X1, X2, X3: in  bit;
              Q1, Q2    : out bit );
-- Q1 : S , PROP; Q2 : C, GEN
end H_ADD;
architecture VERHALTEN of H_ADD is
begin
Q1 <= X1 xor X2 after TD;
Q2 <= (X1 and X2) or X3 after TD;
```

```
end VERHALTEN;
```