CPE 426/526
Chapter 10 - Modeling for Synthesis

Dr. Rhonda Kay Gaede

UAH
10.1 Behavioral Model Development

- Ideally, we create a behavioral model, validate it and then go directly to synthesis. However, the behavioral model may not be synthesizable. So, we find ourselves in the situation shown below.

```
System Specification

<table>
<thead>
<tr>
<th>Creation of Behavioral Model (Model 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Validation &amp; Simulation</td>
</tr>
<tr>
<td>Style Correct?</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

Electrical and Computer Engineering
10.1 Behavioral Model Development - Creation of the Initial Model

- Two types of graphics-based tools
  - ______________________
    - The graphical representation is dictated by the _______________ area.
    - Examples from the worlds of control and DSP are given in the book.
    - Some of these tools generate C, VHDL or Verilog.
  - ______________________
    - Many tools are available that allow entry of models from
      _______________, _______________ or ________________.
      - Modeler’s Assistant is a tool developed at Virginia Tech.

- Modeling and Model Efficiency
  - Simulation - a study shows 3x advantage with hand coding.
  - Synthesis - a study shows 20 % advantage with hand coding

- The type of tool used depends on
  - ______________________
  - ______________________
  - ______________________
  - ______________________
10.2 The Semantics of Simulation and Synthesis

- Simulation semantics have a standard
- Synthesis semantics are tool-specific, though a standardization effort is underway
- **Bad Synthesis**
- **Style Example:**

```vhdl
entity DECIMATE is
generic (DEC: INTEGER);
port ( RX,IX: in REAL:=0.0;
    N: in REAL:=0.0;
    RY,IY: out REAL:=0.0);
end DECIMATE;
architecture BEHAV of DECIMATE is
begin
    process
        variable cnt: integer :=0;
    begin
        wait on N;
        cnt := integer(N) rem DEC;
        if cnt = 0 then
            RY <= RX;
            IY <= IX;
        end if;
    end process;
end BEHAV;
```
library IEEE;
use IEEE.std_logic_1164.all;
entity PAR_TO_SER is
port(START,SHCLK: in STD_LOGIC;
    PAR_IN: in STD_LOGIC_VECTOR(7 downto 0);
    SO: out STD_LOGIC);
end PAR_TO_SER;

architecture ALG1 of PAR_TO_SER is
begin
    P1:process(START,SHCLK)
        variable COUNT: INTEGER range 7 downto -1 := 0;
        variable DONE: BOOLEAN;
        begin
            if START = '1' then
                COUNT := 7; DONE := FALSE;
            elsif SHCLK'EVENT and SHCLK = '1' then
                if DONE = FALSE then
                    SO <= PAR_IN(COUNT); COUNT := COUNT - 1;
                    end if;
                    if COUNT < 0 then DONE := TRUE;
                    else DONE := FALSE;
                    end if;
                    end if;
                    end process;
    end ALG1;

Good Synthesis
Style Example:
library ieee;
use IEEE.std_logic_1164.all;
entity PAR_TO_SER_SCHED is
generic (PERIOD: TIME);
port (START: in STD_LOGIC; PAR_IN: in STD_LOGIC_VECTOR(7 downto 0);
     SO: out STD_LOGIC);
end PAR_TO_SER_SCHED;
architecture ALG2 of PAR_TO_SER_SCHED is
begin
    P1: process (START)
        variable COUNT: INTEGER;
    begin
        if START = '1' then
            COUNT := 7;
            while COUNT >= 0 loop
                SO <= transport PAR_IN (COUNT) after (7 - COUNT) * PERIOD;
                COUNT := COUNT - 1;
            end loop;
        end if;
    end process;
end ALG2;

Another Bad Synthesis Style Example:

The simulation results are the same as the previous model.
10.2 The Semantics of Simulation and Synthesis

- Correct and Incorrect Simulation Models

library ieee;
use ieee.std_logic_1164.all;

dentity T_FF is
port(RESET,T,CLK: in STD_LOGIC; QOUT: out STD_LOGIC);
end T_FF;
architecture ALG of T_FF is
signal Q: STD_LOGIC;
begin
process(RESET,T,CLK)
begin
if (RESET = '1') then
 Q  <= '0';
elsif (CLK'EVENT and CLK = '1') then
 if T = '1' then
 Q  <=  not Q ;
 end if;
end if;
end process;
QOUT <= Q;
end ALG;

library ieee;
use ieee.std_logic_1164.all;

dentity T_FF2 is
port(RESET,T,CLK: in STD_LOGIC; QOUT: out STD_LOGIC);
end T_FF2;
architecture ALG of T_FF2 is
signal Q: STD_LOGIC;
begin
process(RESET,T,CLK)
begin
if (RESET = '1') then
 Q  <= '0';
elsif (CLK'EVENT and CLK = '1') then
 if T = '1' then
 Q  <=  not Q ;
 end if;
end if;QOUT <= Q;
end process;
end ALG;
10.2 The Semantics of Simulation and Synthesis

Delay in Models
All one can expect from a synthesis tool is functionality, the delay comes from the FPGA or ASIC library layout taken into consideration.

Data Types
Stick to std_logic(_vector), bit and constrained integers.
10.3 Modeling Sequential Behavior

Sequential behavior is inferred from basic VHDL language constructs

Three main concerns

1. identifying the clocking mechanism
2. identifying registered elements
3. initializing the circuit

combine (1) and (3)

```vhdl
if (RESET = '1') then
elsif (CLK'event and CLK = '1') then
end if;
```
Wait statements can give us _________ behavior, but not ________________ behavior.

Consider a circuit which toggles an output when the input has the same value for three consecutive clock periods.

entity EQDET is
port(I,CLK: in STD_LOGIC;  TEQDET: inout STD_LOGIC :'0');
end EQDET;

architecture ALG of EQDET is
begin
process
variable EQ,IBK1,IBK2: STD_LOGIC;
begin
wait until (CLK'EVENT and CLK = '1');
if(IBK1 = IBK2) and (IBK2 = I) then
   EQ := '1';
else
   EQ := '0';
end if;
TEQDET <= (EQ xor TEQDET);
IBK2 := IBK1;
IBK1 := I;
end process;
end ALG;
10.3 Modeling Sequential Behavior

entity EQDETR is
  port (RESET, I, CLK : in std_logic; TEQDET : inout std_logic);
end EQDETR;

architecture ALG of EQDETR is
begin
  process(RESET,CLK)
  variable EQ,IBK1,IBK2: STD_LOGIC;
  begin
    if (RESET = '1') then
      IBK1 := '0';
      IBK2 := '0';
      TEQDET <= '0';
    elsif (CLK'EVENT and CLK = '1') then
      if (IBK1 = I) and (IBK1 = IBK2) then
        EQ := '1';
      else
        EQ := '0';
      end if;
      TEQDET <= (EQ xor TEQDET);
      IBK2 := IBK1;
      IBK1 := I;
    end if;
  end process;
end ALG;
10.3 Modeling Sequential Behavior - Synthesis Result for Equality Detector
10.3 Modeling Sequential Behavior -
Using Signals instead of Variables

entity EQDETR is
  port(RESET, I, CLK: in STD_LOGIC; TEQDET: inout STD_LOGIC := '0');
end EQDETR;

architecture ALGS of EQDETR is
  signal IBK1, IBK2: STD_LOGIC;
begin
  process(RESET, CLK)
  begin
    variable EQ: STD_LOGIC;
    begin
      if (RESET = '1') then
        IBK1 <= '0';
        IBK2 <= '0';
        TEQDET <= '0';
      elsif (CLK'EVENT and CLK = '1') then
        if (IBK1 = I) and (IBK1 = IBK2) then EQ := '1';
        else EQ := '0';
        end if;
        TEQDET <= (EQ xor TEQDET);
        IBK1 <= I;
        IBK2 <= IBK1;
      end if;
    end process;
  end ALGS;
architecture FSM of EQDETR is
begin
  P1: process (RESET, CLK)
  type STATE_TYPE is (S0, S1, S2);
  variable STATE: STATE_TYPE; variable IBK1: STD_LOGIC;
  begin
    if RESET = '1' then
      STATE := S0; IBK1 := '0'; TEQDET <= '0';
    elsif (CLK'EVENT and CLK = '1') then
      case (STATE) is
        when S0 => STATE := S1;
           IBK1 := I;
        when S1 => if (IBK1 = I) then STATE := S2;
                   else STATE := S1;
                   end if;
                     IBK1 := I;
        when S2 => if (IBK1 = I) then STATE := S2;
                     TEQDET <= not TEQDET;
                   else STATE := S1;
                     end if;
                     IBK1 := I;
      end case;
    end if;
  end process;
end FSM;
10.3 Modeling Sequential Behavior - Finite State Machine Synthesis Output
10.3 Modeling Sequential Behavior - Synthesis Guidelines

- Don’t give initializations in declarations.
- Don’t use unconstrained integers.
- Let the synthesis tool do the encoding.
- If you need RAM, or other large, customized blocks, instantiate them.
- Don’t use a construct that a technology doesn’t have
10.4 Modeling Combinational Circuits for Synthesis

Consider the ones counter first introduced in chapter 3 again.

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

entity ONES_CNT is
port (A: in STD_LOGIC_VECTOR(2 downto 0);
     C: out STD_LOGIC_VECTOR(1 downto 0));
end ONES_CNT;

architecture DATA_FLOW of ONES_CNT is
begin
  C(1) <= (A(1) and A(0)) or (A(2) and A(0))
           or (A(2) and A(1));
  C(0) <= (A(2) and not A(1) and not A(0))
           or (not A(2) and not A(1) and A(0))
           or (A(2) and A(1) and A(0))
           or (not A(2) and A(1) and not A(0));
end DATA_FLOW;
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

architecture MUX of ONES_CNT is
begin
process(A)
begin
  case A is
    when "000" => C<= "00";
    when "001"|"010"|"100" => C<= "01";
    when "011"|"101"|"110" => C<= "10";
    when "111" => C<= "11";
    when others => null;
  end case;
end process;
end MUX;
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

architecture ROM of ONES_CNT is
begin
    process (A)
    type ROM_TABLE is array( 0 to 7) of STD_LOGIC_VECTOR(1 downto 0);
    constant ROM: ROM_TABLE :=
        (('0','0'),('0','1'),('0','1'),('1','0'),
        ('0','1'),('1','0'),('1','0'),('1','1'));
    begin
        C <= ROM(CONV_INTEGER(A));
    end process;
end ROM;
10.4 Modeling Combinational Circuits for Synthesis - Synthesis Output
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

architecture ALGORITHMIC of ONES_CNT is
begin
P1:process(A)
variable NUM: INTEGER range 0 to 3;
begin
NUM := 0;
for I in 0 to 2 loop
if A(I) = '1' then
NUM := NUM + 1;
end if;
end loop;
case NUM is
when 0 => C <= "00";
when 1 => C <= "01";
when 2 => C <= "10";
when 3 => C <= "11";
end case;
end process P1;
end ALGORITHMIC;
10.4 Modeling Combinational Circuits for Synthesis

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

architecture PROC of ONES_CNT is
  procedure ONES_CNT_PROC(X : in STD_LOGIC_VECTOR;  Z_SIZE:INTEGER;
    signal Z : out STD_LOGIC_VECTOR) is
    variable RES:  STD_LOGIC_VECTOR(Z_SIZE-1 downto 0);
  begin
    RES := CONV_STD_LOGIC_VECTOR(0,Z_SIZE);
    for I in  X'RANGE loop
      if (X(I) = '1') then
        RES := unsigned(RES) + 1;
      end if;
    end loop;
    Z <= RES;
  end ONES_CNT_PROC;
begin
  ONES_CNT_PROC(A,2,C);
end PROC;
```
10.4 Modeling Combinational Circuits for Synthesis - First Step of Synthesis
10.4 Modeling Combinational Circuits for Synthesis - After Optimization
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;

entity SIMP_ADD is
port(A,B: in STD_LOGIC_VECTOR(3 downto 0); CIN: in STD_LOGIC;
     C: out STD_LOGIC_VECTOR(3 downto 0); CAR_OUT: out STD_LOGIC);
end SIMP_ADD;

architecture ALG of SIMP_ADD is
begin
  P1: process(A,B,CIN)
  variable  PADDED_CIN: STD_LOGIC_VECTOR(3 downto 0);
  variable A_UNSIGNED: UNSIGNED(3 downto 0);
  variable C_UNSIGNED: UNSIGNED(4 downto 0);
begin
    A_UNSIGNED := UNSIGNED(A);
    PADDED_CIN := "000" & CIN;
    C_UNSIGNED := CONV_UNSIGNED(A_UNSIGNED,5) + UNSIGNED(B) + UNSIGNED(PADDED_CIN);
    C <= STD_LOGIC_VECTOR(C_UNSIGNED(3 downto 0));
    CAR_OUT <= C_UNSIGNED(4);
  end process;
end ALG;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;

entity SIMP_ADD_TRY_A is
port(A,B: in STD_LOGIC_VECTOR(3 downto 0);
        CIN: in STD_LOGIC;
        C: out STD_LOGIC_VECTOR(3 downto 0);CAR_OUT: out STD_LOGIC);
end SIMP_ADD_TRY_A;

architecture ALG of SIMP_ADD_TRY_A is
begin
    P1:process(A,B,CIN)
    variable  C_INTERNAL : STD_LOGIC_VECTOR(4 downto 0);
    begin
        C_INTERNAL := '0'&A + B + CIN;
        C <= C_INTERNAL(3 downto 0);
        CAR_OUT <= C_INTERNAL(4);
    end process;
end ALG;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;

entity SIMP_ADD_TRY_B is
port(A,B: in integer range 0 to 15;
   CIN: in integer range 0 to 1;
   C: out integer range 0 to 31);
end SIMP_ADD_TRY_B;

architecture ALG of SIMP_ADD_TRY_B is
begin
   C <= A + B + CIN;
end ALG;
10.4 Modeling Combinational Circuits for Synthesis - Arithmetic Circuits

-- simp_add_b (epf10k10qc208-3)
LIBRARY alt_vtl;
USE alt_vtl.VCOMPONENTS.all;
ENTITY simp_add_b IS
   PORT (
      A : IN std_logic_vector(3 downto 0);
      B : IN std_logic_vector(3 downto 0);
      c : OUT std_logic_vector(4 downto 0);
      cin : IN std_logic);
END simp_add_b;
ARCHITECTURE epf10k10qc208_a3 OF simp_add_b IS

SIGNAL gnd : std_logic;
SIGNAL vcc : std_logic;
SIGNAL n_28, n_29, n_30, n_31, n_32, c2_lc_aOUT, n_34, n_35, n_36, n_37, n_38,
      n_39, c3_lc_aOUT, n_41, n_42, n_43, n_44, n_45, n_46, ix72_aOUT,
      n_48, n_49, n_50, n_51, n_52, n_53, c0_lc_aOUT, n_55, n_56, n_57,
      n_58, n_59, n_60, c1_lc_aOUT, n_62, c_add_0_nx25_carry_aOUT, ...
BEGIN
   gnd <= '0';
   vcc <= '1';
   TRIBUF_2: TRIBUF
      PORT MAP (IN1 => n_28, OE => vcc, Y => c(2));
   TRIBUF_4: TRIBUF
      PORT MAP (IN1 => n_35, OE => vcc, Y => c(3));
   TRIBUF_6: TRIBUF
      PORT MAP (IN1 => n_42, OE => vcc, Y => c(4));
10.4 Hierarchical Arithmetic Circuit: Synthesis of Hierarchical Circuits

Decimal to binary conversion is an important logic function in digital circuits. Normal representation of decimal numbers is

\[ N = D_3 \times 10^3 + D_2 \times 10^2 + D_1 \times 10^1 + D_0 \times 10^0 \]

Rewriting it according to Horner

\[ N = ((D_3 \times 10 + D_2) \times 10 + D_1) \times 10 + D_0 \]
10.4 Hierarchical Arithmetic Circuit: BCD to Binary Converter

use IEEE.STD_LOGIC_arith.all;

entity MADD is
generic(IN_WIDTH: NATURAL := 4);
port(DI: in STD_LOGIC_VECTOR(IN_WIDTH-1 downto 0);
    DJ: in STD_LOGIC_VECTOR(3 downto 0);
    MSUM: out STD_LOGIC_VECTOR(IN_WIDTH+3 downto 0));
end MADD;

architecture ALG of MADD is
begin
  P1: process(DI,DJ)
  variable MSUM_US: UNSIGNED(IN_WIDTH+3 downto 0);
  variable PROD:UNSIGNED(2*IN_WIDTH-1 downto 0);
  begin
    PROD := UNSIGNED(DI)*CONV_UNSIGNED(10,IN_WIDTH);
    MSUM_US := PROD(IN_WIDTH+3 downto 0)+ UNSIGNED(DJ);
    MSUM <= STD_LOGIC_VECTOR(MSUM_US);
  end process;
end ALG;
10.4 Hierarchical Arithmetic Circuit: BCD to Binary Converter

entity BCDCONV is
  port(D0,D1,D2,D3: in STD_LOGIC_VECTOR(3 downto 0);
       BIN_OUT: out STD_LOGIC_VECTOR(15 downto 0));
end BCDCONV;
architecture STRUCTURAL of BCDCONV is
  component MADD
    generic(IN_WIDTH: NATURAL := 4);
    port(DI: in STD_LOGIC_VECTOR(IN_WIDTH-1 downto 0);
         DJ: in STD_LOGIC_VECTOR(3 downto 0);
         MSUM: out STD_LOGIC_VECTOR(IN_WIDTH+3 downto 0));
  end component;
  signal MSUM2: STD_LOGIC_VECTOR(7 downto 0);
  signal MSUM1: STD_LOGIC_VECTOR(11 downto 0);
  begin
    C1: MADD
      generic map(4) port map(D3,D2,MSUM2);
    C2: MADD
      generic map(8) port map(MSUM2,D1,MSUM1);
    C3: MADD
      generic map(12) port map(MSUM1,D0,BIN_OUT);
  end STRUCTURAL;
10.4 Hierarchical Arithmetic Circuit: Synthesis of Hierarchical Circuits

Compile -
****************************************
Report: hierarchy Design: BCDCONV
****************************************

BCDCONV
MADD_IN_WIDTH4
  AN2
  EN
  EON1
  IV
  MADD_IN_WIDTH4_DW02_mult_4_4_0
    AN2
    EO
    FA1A
    MADD_IN_WIDTH4_DW01_add_6_0
      AN2
      IV
      ND2
      OR2
    ND2
MADD_IN_WIDTH8
  AN2
  AN3
  AO2

lsi_10k
lsi_10k
lsi_10k
lsi_10k
lsi_10k
lsi_10k
lsi_10k
lsi_10k
lsi_10k
lsi_10k
lsi_10k
lsi_10k
10.4 Hierarchical Arithmetic Circuit: Synthesis of Hierarchical Circuits

Compile -ungroup_all -

************
Report : hierarchy
Design : BCDCONV
Version: 2001.08-SP1
Date   : Mon Mar 29 16:54:01 2004
************

BCDCONV
AN2      lsi_10k
EO       lsi_10k
FA1A     lsi_10k
IV       lsi_10k
ND2      lsi_10k
NR2      lsi_10k
OR2      lsi_10k
10.4 Hierarchical Arithmetic Circuit: Synthesis of Hierarchical Circuits

- Uniquify allows each instance to be compiled independently, could affect result if boundary optimization is enabled
- After uniquifying, compilation can proceed __________ or____________
- For ____________ approach
  - Subdesigns are compiled, taking into account the environment surrounding the subdesign
  - Subdesigns are recompiled as necessary to meet overall design constraints
- In the ____________ approach
  - Compile each instance
  - Compile top-level without touching other levels
10.5 Inferred Latches and Don’t Cares

entity INFERRED is
  port(IN_DAT, IN_EN: in STD_LOGIC; SEL: in STD_LOGIC_VECTOR(1 downto 0);
       A_LATCHED, A_COMB, B_LATCHED, B_COMB_0, B_COMB_1, B_COMB_2: out STD_LOGIC);
--pragma dc_script_begin
--set_flatten true
--pragma dc_script_end
end INFERRED;
architecture ALG of INFERRED is
begin
  P_A_LATCHED: process(IN_DAT, IN_EN)
  begin
    if IN_EN = '1' then A_LATCHED <= IN_DAT; end if;
  end process;
  P_A_COMB: process(IN_DAT, IN_EN)
  begin
    if IN_EN = '1' then A_COMB <= IN_DAT;
    else A_COMB <= '0';
    end if;
  end process;
  P_B_LATCHED: process(IN_DAT, SEL)
  begin
    case SEL is
    when "00" => B_LATCHED <= IN_DAT;
    when "01" => B_LATCHED <= not IN_DAT;
    when "10" => B_LATCHED <= '0';
    when "11" => null;
    when others => null;
    end case;
  end process;
end ALG;
10.5 Inferred Latches and Don’t Cares

P_B_COMB_0: process(IN_DAT,SEL)
begin
  case SEL is
    when "00" => B_COMB_0 <= IN_DAT;
    when "01" => B_COMB_0 <= not IN_DAT;
    when "10" => B_COMB_0 <= '0';
    when "11" => B_COMB_0 <= '1';
    when others => null; end case;
end process;
P_B_COMB_1: process(IN_DAT,SEL)
begin
  B_COMB_1 <= '1';
  case SEL is
    when "00" => B_COMB_1 <= IN_DAT;
    when "01" => B_COMB_1 <= not IN_DAT;
    when "10" => B_COMB_1 <= '0';
    when "11" => null;
    when others => null; end case;
end process;
P_B_COMB_2: process(IN_DAT,SEL)
begin
  case SEL is
    when "00" => B_COMB_2 <= IN_DAT;
    when "01" => B_COMB_2 <= not IN_DAT;
    when "10" => B_COMB_2 <= '0';
    when "11" => B_COMB_2 <= '-';
    when others => null; end case;
end process;
end ALG;
10.5 Inferred Latches and Don’t Cares
10.5 Inferred Latches and Don’t Cares

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity FUNCS is
port(X: in STD_LOGIC_VECTOR(2 downto 0); Z1,Z2: out STD_LOGIC);
--pragma dc_script_begin
--set_flatten true
--pragma dc_script_end
end FUNCS;

architecture ROM of FUNCS is
type ROM_1D is array(0 to 7) of STD_LOGIC;
begins
FULLY_SPECIFIED: process(X)
constant ROM1: ROM_1D:= "01101000";
begins
Z1 <=ROM1(CONV_INTEGER(X));
end process;
PARTIALLY_SPECIFIED: process(X)
constant ROM2: ROM_1D:= "01101--0";
begins
Z2 <=ROM2(CONV_INTEGER(X));
end process;
end ROM;
10.5 Inferred Latches and Don’t Cares
10.6 Tristate Circuits - VHDL Model

To infer a tristate circuit, the synthesis tool wants to see

```
entity TRISTATE is
port(A,B,ENA,ENB: in STD_LOGIC; BUS_SIG: out STD_LOGIC);
end TRISTATE;

architecture   ALG of  TRISTATE is
begin
PROCA: process(A,ENA)
 begin
 if (ENA = '1') then  BUS_SIG <= A;
 else  BUS_SIG <= 'Z';
 end if;
end process;
PROCB: process(B,ENB)
 begin
 if (ENB = '1') then BUS_SIG <= B;
 else   BUS_SIG <= 'Z';
 end if;
end process;
end   ALG;
```
10.6 Tristate Circuits - Synthesized Tristate Circuit
10.7 Shared Resources - Multiplex, then Add or Add, then Multiplex?

```vhdl
entity SHARE_1 is
    port(A,B,C,D: in STD_LOGIC_VECTOR(3 downto 0); SEL_CD: in STD_LOGIC;
        F: out STD_LOGIC_VECTOR(3 downto 0));
end SHARE_1;
architecture DF of SHARE_1 is
    signal MUX1,MUX2: SIGNED(3 downto 0);
    signal SUM: SIGNED(3 downto 0);
    begin
        MUX1 <= SIGNED(A) when SEL_CD = '0' else SIGNED(C);
        MUX2 <= SIGNED(B) when SEL_CD = '0' else SIGNED(D);
        SUM <= MUX1 + MUX2;
        F <= STD_LOGIC_VECTOR(SUM);
    end DF;
```
10.7 Shared Resources - Multiplex, then Add or Add, then Multiplex?

entity SHARE_2 is
  port(A,B,C,D: in STD_LOGIC_VECTOR(3 downto 0); SEL_CD: in STD_LOGIC;
  F: out STD_LOGIC_VECTOR(3 downto 0));
end SHARE_2;

architecture DF of SHARE_2 is
  signal ADD1,ADD2:SIGNED(3 downto 0);
  begin
    ADD1 <= SIGNED(A) + SIGNED(B);
    ADD2 <= SIGNED(C) + SIGNED(D);
    F <= STD_LOGIC_VECTOR(ADD1) when SEL_CD = '0'
        else STD_LOGIC_VECTOR(ADD2);
  end DF;
10.8 Flattening and Structuring - Three Level Circuit Flattened

entity FLAT_LOG_FUNC is
port(A, B, C, D: in STD_LOGIC; F: out STD_LOGIC);
--pragma dc_script_begin
--set_flatten true
--pragma dc_script_end
end FLAT_LOG_FUNC;

architecture DF of FLAT_LOGFUNC is
  signal S1, S2: STD_LOGIC;
begin
  S1 <= A or B;
  S2 <= S1 and C;
  F <= S2 and D;
end DF;
10.8 Flattening and Structuring - Two Level Circuit Structured

entity STRUC_LOG_FUNC is
  port(A,B,C,D: in STD_LOGIC; F1,F2: out STD_LOGIC);
--pragma dc_script_begin
--set_structure -timing true
--pragma dc_script_end
end STRUC_LOG_FUNC;

architecture DF of STRUC_LOG_FUNC is
begin
  F1 <= (A and B) or (A and D);
  F2 <= (B and C) or (C and D);
end DF;