VHDL Modeling for Synthesis

Design Examples – Multiplier
(from Smith text, Chapter 10)
4-bit multiplier (hierarchical design) (Smith Chap. 10.2)
1-bit full adder

entity Full_Adder is
  generic (TS : TIME := 0.11 ns; TC : TIME := 0.1 ns);
  port (X, Y, Cin: in BIT;
        Cout, Sum: out BIT);
end Full_Adder;

architecture Behave of Full_Adder is
begin
  Sum  <= X xor Y xor Cin after TS;
  Cout <= (X and Y) or (X and Cin) or (Y and Cin) after TC;
end;
8-bit adder - entity

-- Cascade 8 1-bit adders for 8-bit adder
entity Adder8 is
port (A, B: in BIT_VECTOR(7 downto 0);
    Cin: in BIT;
    Cout: out BIT;
    Sum: out BIT_VECTOR(7 downto 0));
end Adder8;
architecture Structure of Adder8 is
  component Full_Adder  -- declare component to be used
    port (X,Y,Cin: in BIT;
        Cout, Sum: out BIT);
  end component;
  signal C: BIT_VECTOR(7 downto 0);

begin
  Stages: for i in 7 downto 0 generate
    LowBit: if i = 0 generate
      FA: Full_Adder port map (A(0),B(0),Cin,C(0),Sum(0));
    end generate;

    OtherBits: if i /= 0 generate
      FA: Full_Adder port map (A(i),B(i),C(i-1),C(i),Sum(i));
    end generate;
  end generate;
  Cout <= C(7);
end;
entity DFFClr is
  generic (TRQ : TIME := 2 ns; TCQ : TIME := 2 ns);
  port (CLR, CLK, D : in BIT;
        Q, QB : out BIT);
end;

architecture Behave of DFFClr is
  signal Qi : BIT;
begin
  QB <= not Qi;
  Q <= Qi;
  process (CLR, CLK) begin
    if CLR = '1' then
      Qi <= '0' after TRQ;
    elsif CLK'EVENT and CLK = '1' then
      Qi <= D after TCQ;
    end if;
  end process;
end;
entity Register8 is
  port (D : in BIT_VECTOR(7 downto 0);
        Clk, Clr: in BIT ;
        Q : out BIT_VECTOR(7 downto 0));
end;

architecture Structure of Register8 is
  component DFFClr
    port (Clr, Clk, D : in BIT; Q, QB : out BIT);
  end component;
  begin
    STAGES: for i in 7 downto 0 generate
      FF: DFFClr port map (Clr, Clk, D(i), Q(i), open);
    end generate;
  end;
entity Mux8 is
  generic (TPD : TIME := 1 ns);
  port (A, B : in BIT_VECTOR (7 downto 0);
    Sel : in BIT := '0';
    Y : out BIT_VECTOR (7 downto 0));
End Mux8;

architecture Behave of Mux8 is begin
  Y <= A after TPD when Sel = '1' else B after TPD;
end;
entity AllZero is
  generic (TPD : TIME := 1 ns);
  port (X : in BIT_VECTOR; F : out BIT );
end;

architecture Behave of AllZero is
begin
  process (X) begin
    F <= '1' after TPD;
    for j in X'RANGE loop
      if X(j) = '1' then F <= '0' after TPD; end if;
    end loop;
  end process;
end;
-- Variable-width shift register

entity ShiftN is
    generic (TCQ : TIME := 0.3 ns; TLQ : TIME := 0.5 ns;
             TSQ : TIME := 0.7 ns);
    port(CLK, CLR, LD, SH, DIR: in BIT;
         D: in BIT_VECTOR; Q: out BIT_VECTOR);
begin
    assert (D'LENGTH <= Q'LENGTH)
    report "D wider than output Q" severity Failure;
end ShiftN;

architecture Behave of ShiftN is
begin
    Shift: process (CLR, CLK)
        subtype InB is NATURAL range D'LENGTH-1 downto 0;
        subtype OutB is NATURAL range Q'LENGTH-1 downto 0;
        variable St: BIT_VECTOR(OutB);
begin
  if CLR = '1' then                   -- clear function
    St := (others => '0');
    Q <= St after TCQ;
  elsif CLK'EVENT and CLK='1' then
    if LD = '1' then                  -- load function
      St := (others => '0');
      St(InB) := D;
      Q <= St after TLQ;
    elsif SH = '1' then              -- shift function
      case DIR is
        when '0' => St := '0' & St(St'LEFT downto 1);
        when '1' => St := St(St'LEFT-1 downto 0) & '0';
      end case;
      Q <= St after TSQ;
    end if;
  end if;
end process;
end;
Multiplier control algorithm
-- Controller State Machine

entity SM_1 is
  generic (TPD : TIME := 1 ns);
  port(Start, Clk, LSB, Stop, Reset: in BIT;
       Init, Shift, Add, Done : out BIT);
end;

architecture Moore of SM_1 is
  type STATETYPE is (I, C, A, S, E);
  signal State: STATETYPE;
begin
  -- Outputs are functions of state ("Moore" model)
  Init <= '1' after TPD when State = I else '0' after TPD;
  Add <= '1' after TPD when State = A else '0' after TPD;
  Shift <= '1' after TPD when State = S else '0' after TPD;
  Done <= '1' after TPD when State = E else '0' after TPD;
end;
-- State transitions on rising clock edge
process (CLK, Reset) begin
    if Reset = '1' then
        State <= E;
    elsif CLK'EVENT and CLK = '1' then
        case State is
            when I => State <= C;
            when C => if LSB = '1' then State <= A;
                elsif Stop = '0' then State <= S;
                else State <= E;
            end if;
            when A => State <= S;
            when S => State <= C;
            when E => if Start = '1' then State <= I;
        end case;
    end if;
end process;
end;
Top-level of the design
-- Top level of the design
entity Mult8 is
  port (A, B: in BIT_VECTOR(3 downto 0);
       Start, CLK, Reset: in BIT;
       Result: out BIT_VECTOR(7 downto 0);
       Done: out BIT);
end Mult8;

architecture Structure of Mult8 is
  use work.Mult_Components.all;  -- defined later
  signal SRA, SRB, ADDout, MUXout, REGout: BIT_VECTOR(7 downto 0);
  signal Zero, Init, Shift, Add, Low: BIT := '0';
  signal High: BIT := '1';
  signal F, OFL, REGclr: BIT;
begin

-- Internal signals
REGclr <= Init or Reset;
Result <= REGout;

-- Instantiate components
SR1: ShiftN port map(CLK=>CLK,CLR=>Reset,LD=>Init, SH=>Shift,
  DIR=>Low ,D=>A,Q=>SRA);
SR2: ShiftN port map(CLK=>CLK,CLR=>Reset,LD=>Init, SH=>Shift,
  DIR=>High,D=>B,Q=>SRB);
Z1:  AllZero port map(X=>SRA,F=>Zero);
A1:  Adder8 port map(A=>SRB,B=>REGout,Cin=>Low, Cout=>OFL,
  Sum=>ADDout);
M1:  Mux8 port map(A=>ADDout,B=>REGout,Sel=>Add,Y=>MUXout);
R1:  Register8 port map(D=>MUXout,Q=>REGout,Clk=>CLK,Clr=>REGclr);
F1:  SM_1 port map(Start,CLK,SRA(0),Zero,Reset,Init,Shift,Add,Done);
end;
package Mult_Components is
  component Mux8
    port (A,B:BIT_VECTOR(7 downto 0); Sel:BIT; Y:out BIT_VECTOR(7 downto 0));
  end component;
  component AllZero
    port (X : BIT_VECTOR; F:out BIT );
  end component;
  component Adder8
    port (A,B:BIT_VECTOR(7 downto 0); Cin:BIT; Cout:out BIT;
         Sum:out BIT_VECTOR(7 downto 0));
  end component;
  component Register8
    port (D:BIT_VECTOR(7 downto 0); Clk,Clr:BIT;
         Q:out BIT_VECTOR(7 downto 0));
  end component;
  component ShiftN
    port (CLK,CLR,LD,SH,DIR:BIT; D:BIT_VECTOR; Q:out BIT_VECTOR);
  end component;
  component SM_1
    port (Start,CLK,LSB,Stop,Reset:BIT; Init,Shift,Add,Done:out BIT);
  end component;
end;
-- For test: generate clock with specified high/low times
package Clock_Utils is
    procedure Clock (signal C: out Bit; HT, LT:TIME);
end Clock_Utils;

package body Clock_Utils is
    procedure Clock (signal C: out Bit; HT, LT:TIME) is
        begin
            loop
                C<='1' after LT,
                    '0' after LT + HT; -- waveform
                wait for LT + HT;
            end loop;
        end;
end Clock_Utils;
package body Utils is

-- convert number to an array of bits
function Convert (N, L: NATURAL) return BIT_VECTOR is
    variable T: BIT_VECTOR(L-1 downto 0);
    variable V: NATURAL := N;
    begin
        for i in T'RIGHT to T'LEFT loop
            T(i) := BIT'VAL(V mod 2);
            V := V/2;
        end loop;
        return T;
    end;
end;
-- convert array of bits to a number
function Convert (B: BIT_VECTOR) return NATURAL is
  variable T:BIT_VECTOR(B'LENGTH-1 downto 0) := B;
  variable V:NATURAL:= 0;
begin
  for i in T'RIGHT to T'LEFT loop
    if T(i) = '1' then V:= V + (2**i);
  end if;
  end loop;
  return V;
end;
end Utils;
-- Test bench to exercise multiplier model

entity Test_Mult8_1 is end; -- runs forever, use break!!

architecture Structure of Test_Mult8_1 is
  use Work_Utils.all;
  use Work_Clock_Utils.all;
  component Mult8
    port (A, B : BIT_VECTOR(3 downto 0); Start, CLK, Reset : BIT;
          Result : out BIT_VECTOR(7 downto 0); Done : out BIT);
  end component;
  signal A, B : BIT_VECTOR(3 downto 0);
  signal Start, Done : BIT := '0';
  signal CLK, Reset : BIT;
  signal Result : BIT_VECTOR(7 downto 0);
  signal DA, DB, DR : INTEGER range 0 to 255;
begin
  C: Clock(CLK, 10 ns, 10 ns);
  UUT: Mult8 port map (A, B, Start, CLK, Reset, Result, Done);
  DR <= Convert(Result);
  Reset <= '1', '0' after 1 ns;
process begin
  for i in 1 to 3 loop
    for j in 4 to 7 loop
      DA <= i; DB <= j;
      A<=Convert(i,A'Length); B<=Convert(j,B'Length);
      wait until CLK'EVENT and CLK='1'; wait for 1 ns;
      Start <= '1', '0' after 20 ns;
      wait until Done = '1';
      wait until CLK'EVENT and CLK='1';
    end loop; end loop;
  for i in 0 to 1 loop
    for j in 0 to 15 loop
      DA <= i; DB <= j;
      A<=Convert(i,A'Length); B<=Convert(j,B'Length);
      wait until CLK'EVENT and CLK='1'; wait for 1 ns;
      Start <= '1', '0' after 20 ns;
      wait until Done = '1';
      wait until CLK'EVENT and CLK='1';
    end loop; end loop;
  wait;
end process;