VHDL Modeling for Synthesis

Multiplier Design (Nelson model)
System Example: 8x8 multiplier
Multiply Algorithm

A <- 0
M <- Multiplicand
Q <- Multiplier
CNT <- 0

A <- A + M \times Q(0)
A:Q <- right shift
CNT <- CNT + 1

CNT = 8

No Yes

INIT

ADD

1

SHIFT

0

START

DONE <- 1

HALT
Multiplier – Top Level

entity multiplier is
port (MLIER: in std_logic_vector(7 downto 0);
      MCAND: in std_logic_vector(7 downto 0);
      PRODUCT: out std_logic_vector(15 downto 0);
      CLOCK: in std_logic;
      START: in std_logic;
      RESET: in std_logic;
      DONE: out std_logic);
end multiplier;

architecture structure of multiplier is

use work.mult_components.all; -- component declarations

-- internal signals to interconnect components
signal MR, QR, AR: std_logic_vector(7 downto 0);
signal AD: std_logic_vector(8 downto 0);
signal LoadA, LoadM, LoadQ, ShiftAQ, ClearA: std_logic;
Multiplier – Top Level (continued)

begin

PRODUCT <= AR & QR; -- 16-bit product from A and Q registers

M: mreg port map (MCAND, MR, LoadM, CLOCK);
Q: Qreg port map (MPLIER, QR, AR(0), LoadQ, ShiftAQ, CLOCK);
A: areg port map (AD, AR, LoadA, ShiftAQ, ClearA, CLOCK);
ADR: adder port map (AR, MR, AD);
C: mctrl port map (RESET, START, CLOCK, QR(0), LoadA, LoadM, LoadQ,
                  ShiftAQ, ClearA, DONE);

end;
Multiplicand Register (mreg)

-- simple parallel-load register
library ieee;
use ieee.std_logic_1164.all;
entity mreg is
    port  (  Min:  in  std_logic_vector(7 downto 0);
    Mout: out std_logic_vector(7 downto 0);
    Load: in  std_logic;   -- parallel load only
    Clk:  in  std_logic);   -- system clock
end mreg;

architecture comp of mreg is
begin
    process (Clk)   -- wait for change in Load
    begin
        if (rising_edge(Clk)) then
            if Load = '1' then
                Mout <= Min;  -- parallel load
            end if;
        end if;
    end process;
end;
Accumulator Register (areg)

-- shift register with clear and parallel load
library ieee;
use ieee.std_logic_1164.all;

entity Areg is
  port ( Ain: in    std_logic_vector(8 downto 0);
         Aout: out std_logic_vector(7 downto 0);
         Load: in  std_logic; -- load entire register
         Shift: in std_logic; -- shift right
         Clear: in std_logic; -- clear register
         Clk:   in std_logic); -- clock
end Areg;

architecture comp of areg is
  signal A: std_logic_vector(8 downto 0); -- internal state

(continue next slide)
Accumulator Register (areg)

begin
  Aout <= A(7 downto 0);  -- low 8 internal bits to outputs

process (Clear, Clk)  -- wait for event
begin
  if Clear = '1' then
    A <= "000000000";  -- clear register
  elsif rising_edge(Clk) then
    if Load = '1' then
      A <= Ain;        -- parallel load
    elsif Shift = '1' then
      A <= '0' & A(8 downto 1);  -- right shift
    end if;
  end if;
end process;
end;
Multiplier/Product Register (Qreg)

-- shift register with parallel load

library ieee;
use ieee.std_logic_1164.all;

entity Qreg is
  port ( Qin: in std_logic_vector(7 downto 0);
        Qout: out std_logic_vector(7 downto 0);
        SerIn: in std_logic;  -- serial input for shift
        Load: in std_logic;  -- parallel load
        Shift: in std_logic;  -- right shift
        Clk: in std_logic);  -- clock
end Qreg;

architecture comp of qreg is
  signal Q: std_logic_vector(7 downto 0);  -- internal storage

(continue next slide)
Multiplier/Product Register (Qreg)

begin

Qout <= Q; -- drive output from internal storage

process (Clk) -- wait for clock event
begin

if rising_edge(clk) then
if Load = '1' then
  Q <= Qin; -- load Q
elsif Shift = '1' then
  Q <= SerIn & Q(7 to 1); -- shift Q right
end if;
end if;
end process;
end;
8-bit adder (behavioral)

library ieee;
use ieee.std_logic_1164.all;
entity adder is
  port( X,Y: in  std_logic_vector(7 downto 0);
       Z:     out std_logic_vector(8 downto 0));
end adder;
-- 8th output bit is for carry

architecture comp of adder is
begin
  Z <= std_logic_vector(unsigned('0' & X) + unsigned('0' & Y));
end;
library ieee;
use ieee.std_logic_1164.all;

entity mctrl is
  port (Reset:  in std_logic; -- asynchronous reset
        Start:    in std_logic; -- start pulse
        Clock:    in std_logic; -- clock input
        Q0:       in std_logic; -- LSB of multiplier
        LoadA:    out std_logic; -- load A register from adder
        LoadM:    out std_logic; -- load M register
        LoadQ:    out std_logic; -- load Q register
        ShiftAQ:  out std_logic; -- shift A & Q registers
        ClearA:   out std_logic; -- clear A register
        DONE:     out std_logic); -- external DONE signal
end mctrl;
Multiplier Controller - Architecture

architecture comp of mctrl is
  type states is (INIT, ADD, SHIFT, HALT);
  signal State: States := HALT; -- state of the controller
  signal CNT8: std_logic; -- 1 for 8 iterations
begin
  -- decode state variable for outputs
  ClearA <= '1' when State = INIT else '0';
  LoadM <= '1' when State = INIT else '0';
  LoadQ <= '1' when State = INIT else '0';
  LoadA <= '1' when (State = ADD) and (Q0 = '1') else '0';
  ShiftAQ <= '1' when State = SHIFT else '0';
  DONE <= '1' when State = Halt else '0';
Controller – State transition process

process (Clock) -- implement state machine state transitions
Begin
    if Reset = '1' then
        State <= HALT;
    elsif rising_edge(Clock = '1') then
        case State is
            when HALT => if Start = '1' then -- wait for start pulse
                State <= INIT;
                end if;
            when INIT  => State <= ADD; -- Add A+M if Q0 = 1
            when ADD => State <= SHIFT; -- Shift A:Q
            when Shift => if CNT8 = '1' then -- Shift accumulator/multiplier
                State <= HALT;
                else
                    State <= ADD;
                    end if;
            end case;
        end if;
    end process;
Controller – Iteration counter

process (Clock)
  variable count: integer range 0 to 7;
begin
  if rising_edge(Clock) then
    case State is
      when INIT => count := 0;
      when ADD => if (count = 7) then
        count := 0;
        else
        count := count + 1;
        end if;
      when others => count := count;
    end case;
  end if;
  if (count = 0) then
    CNT8 <= '1';
  else
    CNT8 <= '0';
  end if;
end process;
library ieee;
use ieee.std_logic_1164.all;
package mult_components is
    component Areg
        port (    Ain: in std_logic_vector(8 downto 0);
                  Aout: out std_logic_vector(7 downto 0);
                  Load: in std_logic;  -- load register
                  Shift: in std_logic; -- shift right
                  Clear: in std_logic; -- clear register
                  Clk: in std_logic); -- clock
    end component;
    component Qreg
        port (     Qin: in std_logic_vector(7 downto 0);
                       Qout: out std_logic_vector(7 downto 0);
                       SerIn: in std_logic; -- serial input for shift
                       Load: in std_logic; -- parallel load
                       Shift: in std_logic; -- right shift
                       Clk: in std_logic); -- system clock
    end component;
    component adder
        port(              X,Y: in std_logic_vector(7 downto 0);
                          Z: out std_logic_vector(8 downto 0));
    end component;
Components package

cOMPONENT mreg
  port (Min: in std_logic_vector(7 downto 0);
        Mout: out std_logic_vector(7 downto 0);
        Load: in std_logic;  -- parallel load only
        Clk: in std_logic);  -- system clock
end component;

COMPONENT mctrl
  port (Reset: in std_logic;
        Start: in std_logic;
        Clock: in std_logic;
        Q0: in std_logic;
        LoadA: out std_logic;
        LoadM: out std_logic;
        LoadQ: out std_logic;
        ShiftAQ: out std_logic;
        ClearA: out std_logic;
        Done: out std_logic);
end component;
end package;