Solutions - Homework 2

(Due date: October 7th @ 7:30 pm)

Presentation and clarity are very important! Show your procedure!

PROBLEM 1 (20 PTS)

- Convert the following signed fixed point numbers in format [16 8] to the dual fixed point format 16_8_3. If more bits are required, you are allowed to use the format 17_8_3.

FX	FA.09	09.3E	09.FA	8A.91	80.AE	81.E4	8A.12	AB.CE
DFX								

✓ FA.09:
1111 1010.0000 1001 ⇒ To DFX 16_8_3 (*num*0): 0111101000001001 = 7A09

✓ 09.3E:
0000 1001.0011 1110 ⇒ To DFX 16_8_3 (*num*0): 0000100100111110 = 093E

✓ 09.FA:
0000 1001.1111 1010 ⇒ To DFX 16_8_3 (*num*0): 0000100111111010 = 09FA

✓ 8A.91:
1000 1010.1001 0001 ⇒ To DFX 16_8_3 (*num*0): 0000101010010001 = not a *num*0!
⇒ To DFX 16_8_3 (*num*1): 1111110001010100 = FC54

✓ 80.AE:
1000 0000.1010 1110 ⇒ To DFX 16_8_3 (*num*0): 0000000010101110 = not a *num*0!
⇒ To DFX 16_8_3 (*num*1): 1111110000000101 = FC05

✓ 81.E4:
1000 0001.1110 0100 ⇒ To DFX 16_8_3 (*num*0): 0000000111100100 = not a *num*0!
⇒ To DFX 16_8_3 (*num*1): 1111110000001111 = FC0F

✓ 8A.12:
1000 1010.0001 0010 ⇒ To DFX 16_8_3 (*num*0): 0000101000010010 = not a *num*0!
⇒ To DFX 16_8_3 (*num*1): 1111110001010000 = FC50

✓ AB.CE:
1010 1011.1100 1110 ⇒ To DFX 16_8_3 (*num*0): 0010101111001110 = not a *num*0!
⇒ To DFX 16_8_3 (*num*1): 1111110101011110 = FD5E

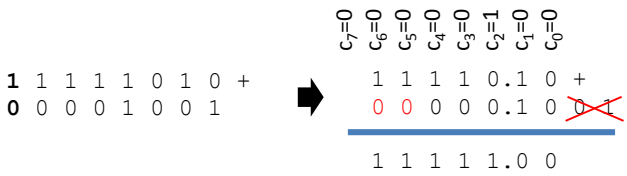
PROBLEM 2 (30 PTS)

- Calculate the result of the following operations where the numbers are represented in dual fixed-point arithmetic. Note that the results must be in the same format. Include an overflow bit when necessary.

DFX Format: 8_4_2	Result	overflow		Result	overflow
FA+09			EB+A3		
FB-90			C0+C2		
43+7A			F6+34		

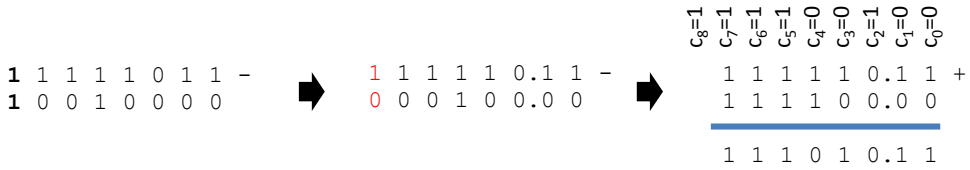
DFX Format 16_8_4	Result	overflow		Result	overflow
FA2A+0A09			F939-0932		
C000+F1C3			F343-6A99		
FFF0-081B			BEEF-FADE		

✓ FA+09:



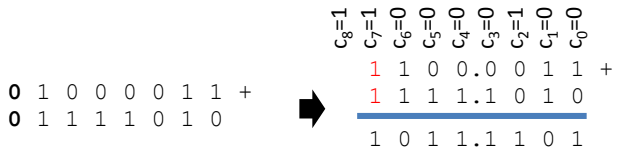
11111.00 ⇒ To DFX 8_4_2 (*num*0): **0**111.0000 = 70 Overflow = 0

✓ FB-90:



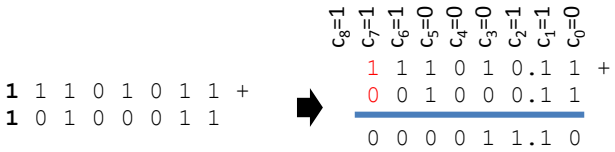
111010.11 ⇒ To DFX 8_4_2 (*num*0): **0**0101100 = not a *num*0!
⇒ To DFX 8_4_2 (*num*1): **1**1101011 = EB Overflow = 0

✓ 43+7A:



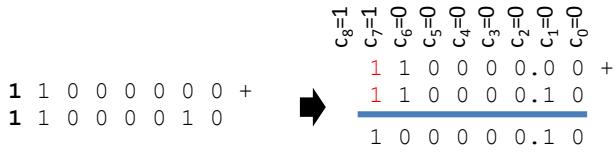
1011.1101 ⇒ To DFX 8_4_2 (*num*0): **0**0111101 = not a *num*0!
⇒ To DFX 8_4_2 (*num*1): **1**1101111 = EF Overflow = 0

✓ EB+A3:



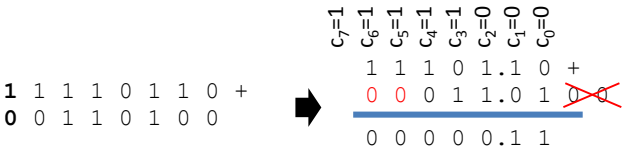
000011.10 ⇒ To DFX 8_4_2 (*num*0): **0**0111000 = 38 Overflow = 0

✓ C0+C2:



100000.10 ⇒ To DFX 8_4_2 (*num*0): **0**0001000 = not a *num*0!
⇒ To DFX 8_4_2 (*num*1): **1**0000010 = not a *num*1! Overflow = 1

✓ F6+34:



00000.11 ⇒ To DFX 8_4_2 (*num*0): **0**0001100 = 0C Overflow = 0

✓ FA2A+0A09:

```
1111101000101010 +      1 1 1 1 1 0 1 0 0 0 1 0.1 0 1 0 +
0000101000001001   ➡    0 0 0 0 0 0 0 1 0 1 0.0 0 0 0 1 0 1 0
                        ---------------------------------------
                        1 1 1 1 1 0 1 0 1 1 0 0.1 0 1 0
```

11110101100.1010 ⇒ To DFX 16_8_4 (*num*0): 0010110010100000 = not a *num*0!
⇒ To DFX 16_8_4 (*num*1): 1111101011001010 = FACA Overflow = 0

✓ C000+F1C3:

```
1100000000000000 +      1 1 0 0 0 0 0 0 0 0 0 0.0 0 0 0 +
1111000111000011   ➡    1 1 1 1 0 0 0 1 1 1 0 0.0 0 1 1
                        ---------------------------------------
                        1 0 1 1 0 0 0 1 1 1 0 0.0 0 1 1
```

101100011100.0011 ⇒ To DFX 16_8_4 (*num*0): 0001110000110000 = not a *num*0!
⇒ To DFX 16_8_4 (*num*1): 1011000111000011 = not a *num*1! Overflow = 1

✓ FFF0-081B:

```
1111111111110000 -   ➡  11111111111.0000 -      ➡  1 1 1 1 1 1 1 1 1 1 1.0 0 0 0 +
0000100000011011        00000001000.0001 1011      1 1 1 1 1 1 1 0 1 1 1.1 1 1 1
                                                   ---------------------------------
                                                   1 1 1 1 1 1 1 0 1 1 0.1 1 1 1
```

11111110110.1111 ⇒ To DFX 16_8_4 (*num*0): 0111011011110000 = 76F0 Overflow = 0

✓ F939-0932:

```
1111100100111001 -   ➡  11110010011.1001 -     ➡  1 1 1 1 0 0 1 0 0 1 1.1 0 0 1 +
0000100100110010        00000001001.0011 0010      1 1 1 1 1 1 1 0 1 1 0.1 1 0 1
                                                   ---------------------------------
                                                   1 1 1 1 0 0 0 1 0 1 0.0 1 1 0
```

11110001010.0110 ⇒ To DFX 16_8_4 (*num*0): 0000101001100000 = not a *num*0!
⇒ To DFX 16_8_4 (*num*1): 1111100010100110 = F8A6 Overflow = 0

✓ F343-6A99:

```
1111001101000011 -   ➡  11100110100.0011 -     ➡  1 1 1 0 0 1 1 0 1 0 0.0 0 1 1 +
0110101010011001        11111101010.1001 1001      0 0 0 0 0 0 1 0 1 0 1.0 1 1 1
                                                   ---------------------------------
                                                   1 1 1 0 1 0 0 1 0 0 1.1 0 1 0
```

11101001001.1010 ⇒ To DFX 16_8_4 (*num*0): 0100100110100000 = not a *num*0!
⇒ To DFX 16_8_4 (*num*1): 1111010010011010 = F49A Overflow = 0

✓ BEEF-FADE:

```
1011111011101111 -   ➡  001111101110.1111 -   ➡  0 0 1 1 1 1 1 0 1 1 1 0.1 1 1 1 +
1111101011011110        111110101101.1110          0 0 0 0 0 1 0 1 0 0 1 0.0 0 1 0
                                                   ---------------------------------
                                                   0 1 0 0 0 1 0 0 0 0 0 1.0 0 0 1
```

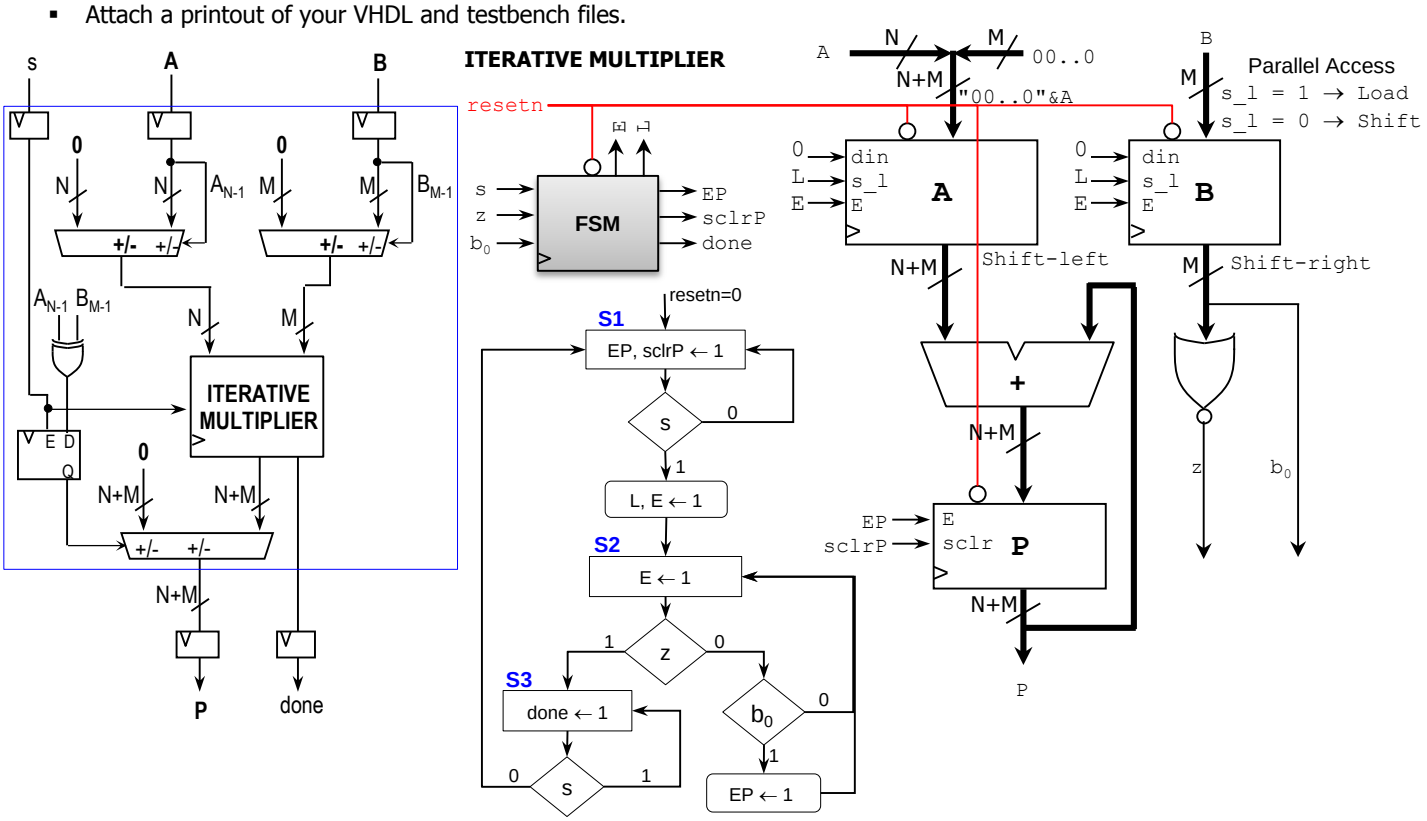
010001000001.0001 ⇒ To DFX 16_8_4 (*num*0): 0100000100010000 = not a *num*0!
⇒ To DFX 16_8_4 (*num*1): 1100010000010001 = not a *num*1! Overflow = 1

PROBLEM 3 (50)

- Design the following signed multiplier circuit ($N = 12, M = 8$). Use the structural description in VHDL. Create a different VHDL file for each circuit (FSM, registers, adder/subtractors).
- Create a testbench to test the following cases. Complete the table.

A	FED	48A	78C	78D
B	FC	FE	F4	61
P	0004C	FF6EC	FA570	2DC6D

- Attach a printout of your VHDL and testbench files.



✓ **VHDL Code**: Top file (Signed Iterative Multiplier)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity iter_sign_mult is
     generic (N: INTEGER:= 12;
              M: INTEGER:= 8);
     port (clock, resetn, s: in std_logic;
           A: in std_logic_vector (N-1 downto 0);
           B: in std_logic_vector (M-1 downto 0);
           P: out std_logic_vector (N+M -1 downto 0);
          done: out std_logic);
end iter_sign_mult;

architecture Behavioral of iter_sign_mult is

    component iter_mult
        generic (N: INTEGER:= 12;
                 M: INTEGER:= 8);
        port (clock, resetn, s: in std_logic;
              dA: in std_logic_vector (N-1 downto 0);
              dB: in std_logic_vector (M-1 downto 0);
              P: out std_logic_vector (N+M -1 downto 0);
              done: out std_logic);
    end component;

     component my_addsub
            generic (N: INTEGER:= 4);
```

```
            port(   addsub   : in std_logic;
                    x, y     : in std_logic_vector (N-1 downto 0);
                    s        : out std_logic_vector (N-1 downto 0);
                    overflow : out std_logic;
                    cout     : out std_logic);
    end component;

    component my_rege
            generic (N: INTEGER:= 4);
            port ( clock, resetn: in std_logic;
                    E, sclr: in std_logic; -- sclr: Synchronous clear
                    D: in std_logic_vector (N-1 downto 0);
                    Q: out std_logic_vector (N-1 downto 0));
    end component;

    component dffe
        Port ( d : in  STD_LOGIC;
               clrn: in std_logic:= '1';
               prn: in std_logic:= '1';
               clk : in  STD_LOGIC;
               ena: in std_logic;
               q : out  STD_LOGIC);
    end component;

    signal Aq, As: std_logic_vector (N-1 downto 0);
    signal Bq, Bs: std_logic_vector (M-1 downto 0);
    signal sq, sqq, aux, donep: std_logic;
    signal Pt, dP: std_logic_vector (N+M -1 downto 0);

begin

ds: dffe port map (d => s, clrn => resetn, prn => '1' , clk => clock, ena => '1', q => sq);

-- Register A:
rA: my_rege generic map (N => N)
    port map (clock => clock, resetn => resetn, E => '1', sclr => '0', D => A, Q => Aq);

-- Register B:
rB: my_rege generic map (N => M)
    port map (clock => clock, resetn => resetn, E => '1', sclr => '0', D => B, Q => Bq);

aux <= Aq(N-1) xor Bq(M-1);
db: dffe port map (d => aux, clrn => resetn, prn => '1' , clk => clock, ena => sq, q => sqq);

--   Adder/Subtractor for A:
ga: my_addsub generic map (N => N)
    port map (addsub => Aq(N-1), x => (others => '0'), y => Aq, s => As);

--   Adder/Subtractor for B:
gb: my_addsub generic map (N => M)
    port map (addsub => Bq(M-1), x => (others => '0'), y => Bq, s => Bs);

ma: iter_mult generic map (N=>N, M=>M)
    port map (clock => clock, resetn => resetn, s => sq, dA => As, dB => Bs, P => Pt, done => donep);

-- Adder:
gP: my_addsub generic map (N => N+M)
    port map (addsub => sqq, x => (others => '0'), y => Pt, s => dP);

-- Register P:
rP: my_rege generic map (N => N+M)
    port map (clock => clock, resetn => resetn, E => '1', sclr => '0', D => dP, Q => P);

dd: dffe port map (d => donep, clrn => resetn, prn => '1' , clk => clock, ena => '1', q => done);

end Behavioral;
```

- ✓ **VHDL Code**: Iterative Multiplier

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity iter_mult is
    generic (N: INTEGER:= 12;
             M: INTEGER:= 8);
```

```
     port (clock, resetn, s: in std_logic;
           dA: in std_logic_vector (N-1 downto 0);
           dB: in std_logic_vector (M-1 downto 0);
           P: out std_logic_vector (N+M -1 downto 0);
           done: out std_logic);
end iter_mult;

architecture Behavioral of iter_mult is

     component my_pashiftreg
            generic (N: INTEGER:= 4;
                     DIR: STRING:= "LEFT");
            port ( clock, resetn: in std_logic;
                   din, E, s_l: in std_logic; -- din: shiftin input
                   D: in std_logic_vector (N-1 downto 0);
                   Q: out std_logic_vector (N-1 downto 0);
                   shiftout: out std_logic);
     end component;

     component my_addsub
            generic (N: INTEGER:= 4);
            port(   addsub   : in std_logic;
                   x, y     : in std_logic_vector (N-1 downto 0);
                   s        : out std_logic_vector (N-1 downto 0);
                   overflow : out std_logic;
                   cout     : out std_logic);
     end component;

     component my_rege
            generic (N: INTEGER:= 4);
            port ( clock, resetn: in std_logic;
                   E, sclr: in std_logic; -- sclr: Synchronous clear
                   D: in std_logic_vector (N-1 downto 0);
                   Q: out std_logic_vector (N-1 downto 0));
     end component;

     type state is (S1, S2, S3);
     signal y: state;
     signal LA, LB, EA, EB, EP, sclrP, z: std_logic;
     signal B: std_logic_vector (M-1 downto 0);
     signal A, dAx, dP, Pt: std_logic_vector (N+M -1 downto 0);

begin

rA: my_pashiftreg generic map (N => N+M, DIR => "LEFT")
    port map (clock => clock, resetn => resetn, din => '0', s_l => LA, E => EA, D => dAx, Q => A);
    dAx (M+N -1 downto N) <= (others =>'0');  dAx (N-1 downto 0) <= dA;

rB: my_pashiftreg generic map (N => M, DIR => "RIGHT")
    port map (clock => clock, resetn => resetn, din => '0', s_l => LB, E => EB, D => dB, Q => B);

-- n-bit NOR gate:
     process (B)
            variable result_or: std_logic;
     begin
            result_or:= '0';
            for i in B'range loop -- 'range: iterates through all bits in 'A'
               result_or := result_or or B(i);
            end loop;
            z <= not (result_or);
     end process;

--    Adder:
ga: my_addsub generic map (N => N+M)
    port map (addsub => '0', x => A, y => Pt, s => dP);

-- Register P:
rP: my_rege generic map (N => N+M)
    port map (clock => clock, resetn => resetn, E => EP, sclr => sclrP, D => dP, Q => Pt);

P <= Pt;

-- FSM:
     Transitions: process (resetn, clock, s, z, B(0))
     begin
```

```
            if resetn = '0' then -- asynchronous signal
                   y <= S1; -- if resetn asserted, go to initial state: S1
            elsif (clock'event and clock = '1') then
                   case y is
                          when S1 => if s = '1' then y <= S2; else y <= S1; end if;
                          when S2 => if z = '1' then y <= S3; else y <= S2; end if;
                          when S3 => if s = '1' then y <= S3; else y <= S1; end if;
                   end case;
            end if;
     end process;

     Outputs: process (y, s,z,B(0))
     begin
        -- Initialization of output signals
            sclrP <= '0'; EP <= '0'; LA <= '0'; LB <= '0'; EA <= '0'; EB <= '0'; done <= '0';
            case y is
                   when S1 =>
                          sclrP <= '1'; EP <= '1';
                          if s = '1' then
                                 LA <= '1'; EA <= '1';  LB <= '1'; EB <= '1';
                          end if;

                   when S2 =>
                          EA <= '1'; EB <= '1';
                          if z = '0' then
                                 if B(0) = '1' then EP <= '1'; end if;
                          end if;

                   when S3 =>  done <= '1';
            end case;
     end process;

end Behavioral;
```

✓ **VHDL Code**: Parallel Access Shift Register

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity my_pashiftreg is
   generic (N: INTEGER:= 4;
            DIR: STRING:= "LEFT");
     port ( clock, resetn: in std_logic;
            din, E, s_l: in std_logic; -- din: shiftin input
            D: in std_logic_vector (N-1 downto 0);
            Q: out std_logic_vector (N-1 downto 0);
           shiftout: out std_logic);
end my_pashiftreg;

architecture Behavioral of my_pashiftreg is
     signal Qt: std_logic_vector (N-1 downto 0);
begin
     process (resetn, clock)
     begin
            if resetn = '0' then Qt <= (others => '0');
            elsif (clock'event and clock = '1') then
                   if E = '1' then
                          if s_l = '1' then Qt <= D;
                          else
                                 if DIR = "LEFT" then
                                    Qt(0) <= din;
                                    for i in 1 to N-1 loop
                                      Qt(i) <= Qt(i-1);
                                    end loop;
                                 elsif DIR = "RIGHT" then
                                    Qt(N-1) <= din;
                                    for i in 0 to N-2 loop
                                      Qt(i) <= Qt(i+1);
                                    end loop;
                                 end if;
                          end if;
                   end if;
            end if;

     end process;
```

```
        Q <= Qt;

        gl: if DIR = "LEFT" generate
                shiftout <= Qt(N-1);
            end generate;
        gr: if DIR = "RIGHT" generate
                shiftout <= Qt(0);
            end generate;

    end Behavioral;
```

✓ **VHDL Code**: Adder/Subtractor

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity my_addsub is
     generic (N: INTEGER:= 4);
     port(   addsub   : in std_logic; -- addsub = 0 (add), addsub = 1 (sub)
             x, y     : in std_logic_vector (N-1 downto 0);
             s        : out std_logic_vector (N-1 downto 0);
             overflow : out std_logic;
             cout     : out std_logic);
end my_addsub;

architecture structure of my_addsub is
     component fulladd
             port( cin, x, y : in std_logic;
                   s, cout   : out std_logic);
     end component;
     signal c: std_logic_vector (N downto 0);
     signal yx: std_logic_vector (N-1 downto 0);

begin
      c(0) <= addsub; cout <= c(N);  overflow <= c(N) xor c(N-1);
     gi: for i in 0 to N-1 generate
             yx(i) <= y(i) xor addsub;
             fi: fulladd port map (cin => c(i), x => x(i), y => yx(i), s => s(i), cout => c(i+1));
         end generate;
end structure;
```

✓ **VHDL Code**: D-type Flip Flop

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity dffe is
    Port ( d : in  STD_LOGIC;
           clrn: in std_logic:= '1';
           prn: in std_logic:= '1';
           clk : in  STD_LOGIC;
           ena: in std_logic;
           q : out  STD_LOGIC);
end dffe;

architecture behaviour of dffe is

begin
     process (clk, ena, prn, clrn)
     begin
       if clrn = '0' then q <= '0';
       elsif prn = '0' then q <= '1';
       elsif (clk'event and clk='1') then
         if ena = '1' then
           q <= d;
         end if;
       end if;
     end process;

end behaviour;
```

✓ **VHDL Code**: N-bit Register

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- N-bit Register
-- E = '1', sclr = '0' --> Q <= D
-- E = '1', sclr = '1' --> Q is cleared (0)
entity my_rege is
   generic (N: INTEGER:= 4);
    port ( clock, resetn: in std_logic;
           E, sclr: in std_logic;
           D: in std_logic_vector (N-1 downto 0);
           Q: out std_logic_vector (N-1 downto 0));
end my_rege;

architecture Behavioral of my_rege is
    signal Qt: std_logic_vector (N-1 downto 0);
begin
    process (resetn, clock)
    begin
            if resetn = '0' then
              Qt <= (others => '0');
            elsif (clock'event and clock = '1') then
                    if E = '1' then
                      if sclr = '1' then
                         Qt <= (others => '0');
                      else
                         Qt <= D;
                      end if;
                    end if;
            end if;

    end process;
    Q <= Qt;
end Behavioral;
```

✓ **VHDL Testbench:**

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY tb_iter_sign_mult IS
    generic (N: INTEGER:= 12;
             M: INTEGER:= 8);
END tb_iter_sign_mult;

ARCHITECTURE behavior OF tb_iter_sign_mult IS

    -- Component Declaration for the Unit Under Test (UUT)
    component iter_sign_mult
        port (clock, resetn, s: in std_logic;
              A: in std_logic_vector (N-1 downto 0);
              B: in std_logic_vector (M-1 downto 0);
              P: out std_logic_vector (N+M -1 downto 0);
              done: out std_logic);
    end component;

   --Inputs
   signal clock : std_logic := '0';
   signal resetn : std_logic := '0';
   signal s : std_logic := '0';
   signal A : std_logic_vector(N-1 downto 0) := (others => '0');
   signal B : std_logic_vector(M-1 downto 0) := (others => '0');

    --Outputs
   signal P : std_logic_vector(N+M-1 downto 0);
   signal done : std_logic;

   -- Clock period definitions
   constant clock_period : time := 10 ns;

BEGIN
    -- Instantiate the Unit Under Test (UUT)
   uut: iter_sign_mult PORT MAP (clock => clock, resetn => resetn, s => s, A => A, B => B, P => P,
          done => done);

   -- Clock process definitions
   clock_process :process
   begin
            clock <= '0'; wait for clock_period/2;
            clock <= '1'; wait for clock_period/2;
   end process;

   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns; resetn <= '1';

      -- insert stimulus here
       A <= x"FED"; B <= x"FC"; s <= '1'; wait for clock_period;
       s <= '0';

       wait for clock_period*(N+5);
       A <= x"48A"; B <= x"FE"; s <= '1'; wait for clock_period;
       s <= '0';

       wait for clock_period*(N+5);
       A <= x"78C"; B <= x"F4"; s <= '1'; wait for clock_period;
       s <= '0';

        wait for clock_period*(N+5);
       A <= x"78D"; B <= x"61"; s <= '1'; wait for clock_period;
       s <= '0';

      wait;
   end process;

END;
```