

Ocenjevanje datoteke testnih vrednosti ALU za vse operacije

Ocenjevanje datoteke testnih vrednosti ALU za vse operacije	1
-- **** STUDENT: 64200100.....	3
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Upoštevati morate dejansko zakasnitev ALU (glej vrednost combinatorial path delay), ne wait for 1 ns!.....	3
-- **** STUDENT: 64200112.....	9
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Upoštevati morate dejansko zakasnitev ALU (glej vrednost combinatorial path delay), ne wait for 1 ns!.....	9
-- **** STUDENT: 64200163.....	14
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Manjka assert.	14
Upoštevati morate dejansko zakasnitev ALU (glej vrednost combinatorial path delay), ne wait for 1 ns!.....	14
Ta datoteka testnih vrednosti samo izračunava vse kombinacije – jih ne preverja v ločenem procesu in ne vrne napake ob različnem izidu (assert).	14
-- **** STUDENT: 64200238.....	16
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	16
-- **** STUDENT: 64200288.....	22
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Upoštevati morate dejansko zakasnitev ALU (glej vrednost combinatorial path delay), ne wait for 3 ns!.....	22
-- **** STUDENT: 64200385.....	28
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Upoštevati morate dejansko zakasnitev ALU (glej vrednost combinatorial path delay), ne wait for 1 ns!.....	28
-- **** STUDENT: 64210113.....	35
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Upoštevati morate dejansko zakasnitev ALU (glej vrednost combinatorial path delay), ne wait for 1 ns!.....	35
-- **** STUDENT: 64210382.....	43
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Upoštevati morate dejansko zakasnitev ALU (glej vrednost combinatorial path delay), ne wait for 1 ns!.....	43
-- **** STUDENT: 64210384.....	50
-- KOMENTARJI K OCENI NALOGE -- Matej Možek:	50
Povezava med alu_cla in testbench mora biti parametrizirana (manjka generic map (n=>n))	50
Upoštevati morate dejansko zakasnitev ALU (glej vrednost combinatorial path delay), ne wait for 100 ps!.....	50

-- **** STUDENT: 64210386.....	55
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Upoštevati morate dejansko zakasnitev ALU (glej vrednost combinatorial path delay), ne wait for 10 ns!.....	55
Povezava med alu_cla in testbench mora biti parametrizirana (manjka generic map (n=>n))	55
-- **** STUDENT: 64210445.....	63
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Upoštevati morate dejansko zakasnitev ALU (glej vrednost combinatorial path delay), ne wait for 1 ns!.....	63
Povezava med alu_cla in testbench mora biti parametrizirana (manjka generic map (n=>n))	63
-- **** STUDENT: 64210455.....	71
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Upoštevati morate dejansko zakasnitev ALU (glej vrednost combinatorial path delay), ne wait for 1 ns!.....	71
-- **** STUDENT: 64210457.....	76
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Upoštevati morate dejansko zakasnitev ALU (glej vrednost combinatorial path delay), ne wait for 1 ns!.....	76
Povezava med alu_cla in testbench mora biti parametrizirana (manjka generic map (n=>n))	76
-- **** STUDENT: 64240430.....	82
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Upoštevati morate dejansko zakasnitev ALU (glej vrednost combinatorial path delay), ne wait for 100 ps!.....	82
Povezava med alu_cla in testbench mora biti parametrizirana (manjka generic map (n=>n))	82
-- **** STUDENT: 64210132.....	87
-- KOMENTARJI K OCENI NALOGE Matej Možek: Povezava med alu_cla in testbench mora biti parametrizirana (manjka generic map (n=>n)).	87
Ideja datoteke testnih vrednosti je, da preleti celoten obseg števil x in y ter na drugačen način (torej z VHDL + operatorjem) preveri, če so razlike v izračunu strojne komponente in simulatorja. Koda datoteke testnih vrednosti je ista kot pri testiranju CLA seštevalnika, kar ni poanta naloge – testirati je treba operaciji seštevanja in odštevanja preko celotnega obsega operandov X in Y.	87
***** NASLEDNJIČ KODO NALOŽITE V USTREZEN RAZDELEK (OSTALO_x), KJER JE X ŠTEVILKA DOMAČE NALOGE *****	87

```

-- *****
-- **** STUDENT: 64200100
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Upoštevati morate dejansko zakasnitev ALU (glej vrednost combinatorial path delay), ne wait for 1 ns!
-- *****

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity alu_tb is
generic( n:natural:=8 );
end alu_tb;

architecture behavior of alu_tb is
component alu_cla
    generic( n: natural := 8 );
    port( M          : in    std_logic;
          F          : in    std_logic_vector( 2 downto 0 );
          X, Y       : in    std_logic_vector( n-1 downto 0 );
          S          : out   std_logic_vector( n-1 downto 0 );
          Negative, Cout, Overflow, Zero, Gout, Pout : out   std_logic );
end component;

signal      M: std_logic := '0';
signal      F: std_logic_vector( 2 downto 0 ):= ( others=>'0' );
signal      X: std_logic_vector( n-1 downto 0 ):= ( others=>'0' );
signal      Y: std_logic_vector( n-1 downto 0 ):= ( others=>'0' );

signal      S: std_logic_vector( n-1 downto 0 );
signal      Negative: std_logic;
signal      Cout: std_logic;
signal      Overflow: std_logic;
signal      Zero: std_logic;
signal      Gout: std_logic;
signal      Pout: std_logic;

signal      X8u,Y8u: integer :=0;
signal      Sprocs, xandy, xnandy, xory, xnory, xxory, xxnory : std_logic_vector( n-1 downto 0 ) := ( others => '0' );

```

```

signal      Negprocs, Overprocs, Zeroprocs : std_logic := '0';
constant    nic : std_logic_vector( n-1 downto 0 ) := ( others => '0' );

```

```

BEGIN

```

```

    uut: alu_cla
    GENERIC MAP ( n => n )
    PORT MAP (
        M => M,
        F => F,
        X => X,
        Y => Y,
        S => S,
        Negative => Negative,
        Cout => Cout,
        Overflow => Overflow,
        Zero => Zero,
        Gout => Gout,
        Pout => Pout );

```

```

    stimproc: process
    begin
        for i in -( 2**( n-1 ) ) to 2**( n-1 ) - 1 loop
            X8u    <= i;
            X      <= std_logic_vector( to_signed( i, n ) );
            for j in -( 2**( n-1 ) ) to 2**( n-1 ) - 1 loop
                Y8u    <= j;
                Y      <= std_logic_vector( to_signed( j, n ) );

```

```

            M      <= '0';
            F      <= "000";
            wait for 1 ns;
            assert( Sprocs = S );
            assert( Negprocs = Negative );
            assert( Overprocs = Overflow );
            assert( Zeroprocs = Zero );

```

```

            F      <= "001";
            wait for 1 ns;
            assert( Sprocs = S );
            assert( Negprocs = Negative );

```

```
assert( Overprocs = Overflow );
assert( Zeroprocs = Zero );
```

```
F      <= "010";
wait for 1 ns;
assert( Sprocs = S );
assert( Negprocs = Negative );
assert( Overprocs = Overflow );
assert( Zeroprocs = Zero );
```

```
F      <= "011";
wait for 1 ns;
assert( Sprocs = S );
assert( Negprocs = Negative );
assert( Overprocs = Overflow );
assert( Zeroprocs = Zero );
```

```
F      <= "100";
wait for 1 ns;
assert( Sprocs = S );
assert( Negprocs = Negative );
assert( Overprocs = Overflow );
assert( Zeroprocs = Zero );
```

```
F      <= "101";
wait for 1 ns;
assert( Sprocs = S );
assert( Negprocs = Negative );
assert( Zeroprocs = Zero );
```

```
F      <= "110";
wait for 1 ns;
assert( nic = S );
```

```
F      <= "111";
wait for 1 ns;
assert( nic = S );
```

```
M      <= '1';
```

```
F      <= "000";
```

```

wait for 1 ns;
assert( xandy = S );

F      <= "001";
wait for 1 ns;
assert( xnandy = S );

F      <= "010";
wait for 1 ns;
assert( xory = S );

F      <= "011";
wait for 1 ns;
assert( xnory = S );

F      <= "100";
wait for 1 ns;
assert( xxory = S );

F      <= "101";
wait for 1 ns;
assert( xxnory = S );

F      <= "110";
wait for 1 ns;
assert( X = S );
assert( Negprocs = Negative );
assert( Zeroprocs = Zero );

F      <= "111";
wait for 1 ns;
assert( Y = S );
assert( Negprocs = Negative );
assert( Zeroprocs = Zero );
end loop;
end loop;
wait;
end process;

procs: process ( X8u, Y8u, F )

```

```

variable SZ, XZ, YZ : std_logic_vector( n-1 downto 0 );
variable RZ : integer;
begin
XZ := std_logic_vector( to_signed( X8u, n ) );
YZ := std_logic_vector( to_signed( Y8u, n ) );
if F = "101" then
    SZ := not nic;
    Overprocs    <= '0';
else
    if F = "000" then
        RZ := X8u + Y8u;
    elsif F = "001" then
        RZ := X8u - Y8u;
    elsif F = "010" then
        RZ := X8u + 1;
    elsif F = "011" then
        RZ := X8u - 1;
    elsif F = "100" then
        RZ := X8u + X8u;
    elsif F = "110" then
        RZ := X8u;
    elsif F = "111" then
        RZ := Y8u;
    else
        RZ := 0;
    end if;

    if RZ > 2**( n-1 ) - 1 or RZ < -( 2**( n-1 ) ) then
        Overprocs    <= '1';
    else
        Overprocs    <= '0';
    end if;
    SZ := std_logic_vector( to_unsigned( RZ mod 2**n, n ) );
end if;
Sprocs <= SZ;
Negprocs    <= SZ( n-1 );
if SZ = nic then
    Zeroprocs    <= '1';
else
    Zeroprocs    <= '0';
end if;

```

```
end if;  
xandy <= XZ and YZ;  
xnandy <= XZ nand YZ;  
xory <= XZ or YZ;  
xnory <= XZ nor YZ;  
xxory <= XZ xor YZ;  
xxnory <= XZ xnor YZ;  
end process;  
END;
```



```

-- *****
-- **** STUDENT: 64200112
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Upoštevati morate dejansko zakasnitev ALU (glej vrednost combinatorial path delay), ne wait for 1 ns!
-- *****

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY alu_cla_tb IS
    GENERIC ( n : natural := 8 );
END alu_cla_tb;

ARCHITECTURE behavior OF alu_cla_tb IS
    COMPONENT alu_cla
        GENERIC ( n : natural := 8 );
        PORT (
            M : IN std_logic;
            F : IN std_logic_vector( 2 downto 0 );
            X : IN std_logic_vector( n-1 downto 0 );
            Y : IN std_logic_vector( n-1 downto 0 );
            S : OUT std_logic_vector( n-1 downto 0 );
            Negative : OUT std_logic;
            Cout : OUT std_logic;
            Overflow : OUT std_logic;
            Zero : OUT std_logic;
            Gout : OUT std_logic;
            Pout : OUT std_logic
        );
    END COMPONENT;

    SIGNAL      M : std_logic := '0';
    SIGNAL      F : std_logic_vector( 2 downto 0 ) := ( others => '0' );
    SIGNAL      X : std_logic_vector( n-1 downto 0 ) := ( others => '0' );
    SIGNAL      Y : std_logic_vector( n-1 downto 0 ) := ( others => '0' );

    SIGNAL      S : std_logic_vector( n-1 downto 0 );
    SIGNAL      Negative : std_logic;
    SIGNAL      Cout : std_logic;

```

```

SIGNAL      Overflow : std_logic;
SIGNAL      Zero : std_logic;
SIGNAL      Gout : std_logic;
SIGNAL      Pout : std_logic;

SIGNAL      X_int, Y_int : integer := 0;
SIGNAL      S_comp, X_and_Y, X_nand_Y, X_or_Y, X_nor_Y,
X_xor_Y, X_xnor_Y : std_logic_vector( n-1 downto 0 ) := ( others => '0' );
SIGNAL      Neg_comp, Over_comp, Zero_comp : std_logic := '0';

CONSTANT    zeros : std_logic_vector( n-1 downto 0 ) := ( others => '0' );

```

```
BEGIN
```

```

uut: alu_cla
GENERIC MAP ( n => n )
PORT MAP (
M => M,
F => F,
X => X,
Y => Y,
S => S,
Negative => Negative,
Cout => Cout,
Overflow => Overflow,
Zero => Zero,
Gout => Gout,
Pout => Pout
);

stim_proc: process
BEGIN
FOR i IN -( 2**( n-1 ) ) TO 2**( n-1 ) - 1 LOOP
X_int <= i;
X      <= std_logic_vector( to_signed( i, n ) );

FOR j IN -( 2**( n-1 ) ) TO 2**( n-1 ) - 1 LOOP
Y_int <= j;
Y      <= std_logic_vector( to_signed( j, n ) );

```

```

M    <= '0';
F    <= "000";
WAIT FOR 1 ns;
ASSERT( S_comp = S ) REPORT "0000 failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "001";
WAIT FOR 1 ns;
ASSERT( S_comp = S ) REPORT "0001 failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "010";
WAIT FOR 1 ns;
ASSERT( S_comp = S ) REPORT "0010 failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "011";
WAIT FOR 1 ns;
ASSERT( S_comp = S ) REPORT "0011 failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "100";
WAIT FOR 1 ns;
ASSERT( S_comp = S ) REPORT "0100 failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "101";
WAIT FOR 1 ns;
ASSERT( S_comp = S ) REPORT "0101 failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "110";
WAIT FOR 1 ns;
ASSERT( zeros = S ) REPORT "0110 failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "111";
WAIT FOR 1 ns;
ASSERT( zeros = S ) REPORT "0111 failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

M    <= '1';
F    <= "000";
WAIT FOR 1 ns;
ASSERT( X_and_Y = S ) REPORT "1000 failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "001";
WAIT FOR 1 ns;

```

```

ASSERT( X_nand_Y = S ) REPORT "1001 failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F      <= "010";
WAIT FOR 1 ns;
ASSERT( X_or_Y = S ) REPORT "1010 failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F      <= "011";
WAIT FOR 1 ns;
ASSERT( X_nor_Y = S ) REPORT "1011 failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F      <= "100";
WAIT FOR 1 ns;
ASSERT( X_xor_Y = S ) REPORT "1100 failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F      <= "101";
WAIT FOR 1 ns;
ASSERT( X_xnor_Y = S ) REPORT "1101 failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F      <= "110";
WAIT FOR 1 ns;
ASSERT( X = S ) REPORT "1110 failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F      <= "111";
WAIT FOR 1 ns;
ASSERT( Y = S ) REPORT "1111 failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;
END LOOP;
END LOOP;
WAIT;
END PROCESS;

test_proc: process ( X_int, Y_int, F )
VARIABLE S_temp, X_temp, Y_temp : std_logic_vector( n-1 downto 0 );
VARIABLE Res_temp : integer;
BEGIN
X_temp := std_logic_vector( to_signed( X_int, n ) );
Y_temp := std_logic_vector( to_signed( Y_int, n ) );

IF F = "101" THEN
S_temp := NOT zeros;
Over_comp <= '0';

```

```

ELSE
CASE F IS
WHEN "000" => Res_temp := X_int + Y_int;
WHEN "001" => Res_temp := X_int - Y_int;
WHEN "010" => Res_temp := X_int + 1;
WHEN "011" => Res_temp := X_int - 1;
WHEN "100" => Res_temp := X_int + X_int;
WHEN "110" => Res_temp := X_int;
WHEN "111" => Res_temp := Y_int;
WHEN OTHERS => Res_temp := 0;
END CASE;

IF Res_temp > 2**( n-1 ) - 1 OR Res_temp < -( 2**( n-1 ) ) THEN
Over_comp   <= '1';
ELSE
Over_comp   <= '0';
END IF;

S_temp := std_logic_vector( to_unsigned( Res_temp MOD 2**n, n ) );
END IF;

S_comp    <= S_temp;
Neg_comp  <= S_temp( n-1 );

IF S_temp = zeros THEN
Zero_comp <= '1';
ELSE
Zero_comp <= '0';
END IF;

X_and_Y    <= X_temp AND Y_temp;
X_nand_Y   <= X_temp NAND Y_temp;
X_or_Y     <= X_temp OR Y_temp;
X_nor_Y    <= X_temp NOR Y_temp;
X_xor_Y    <= X_temp XOR Y_temp;
X_xnor_Y   <= X_temp XNOR Y_temp;
END PROCESS;

END;

```

```

-- *****
-- **** STUDENT: 64200163
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Manjka assert.
Upoštevati morate dejansko zakasnitev ALU (glej vrednost combinatorial path delay), ne wait for 1 ns!
Ta datoteka testnih vrednosti samo izračunava vse kombinacije – jih ne preverja v ločenem procesu in ne vrne napake ob
različnem izidu (assert).
-- *****

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity alu_cla_tb is
    generic( n: natural:= 8 );
end alu_cla_tb;

architecture behavior of alu_cla_tb is
    component alu_cla
        generic( n: natural:= 8 );
    port( M : in std_logic;
          F : in std_logic_vector( 2 downto 0 );
          X : in std_logic_vector( n-1 downto 0 );
          Y : in std_logic_vector( n-1 downto 0 );
          S : out std_logic_vector( n-1 downto 0 );
          Negative : out std_logic;
          Cout: out std_logic;
          Overflow: out std_logic;
          Zero: out std_logic;
          Gout: out std_logic;
          Pout: out std_logic
        );
    end component;

    signal M : std_logic := '1';
    signal F : std_logic_vector( 2 downto 0 ) := ( others => '0' );
    signal X : std_logic_vector( n-1 downto 0 ) := ( others => '0' );
    signal Y : std_logic_vector( n-1 downto 0 ) := ( others => '0' );
    signal S : std_logic_vector( n-1 downto 0 );
    signal Negative : std_logic;

```

```

signal          Cout : std_logic;
signal          Overflow : std_logic;
signal          Zero : std_logic;
signal          Gout : std_logic;
signal          Pout : std_logic;

begin
  uut: alu_cla
    generic map( n => n )
    port map(
      M => M,

      F => F,
      X => X,
      Y => Y,
      S => S,
      Negative => Negative,
      Cout => Cout,
      Overflow => Overflow,
      Zero => Zero,
      Gout => Gout,
      Pout => Pout
    );
  stim_proc: process
    begin
      for i in -( 2**( n-1 ) ) to ( 2**( n-1 ) )-1 loop
        X      <= std_logic_vector( to_signed( i, n ) );
        for j in -( 2**( n-1 ) ) to ( 2**( n-1 ) )-1 loop
          Y      <= std_logic_vector( to_signed( j, n ) );
          for k in 0 to 1 loop
            M      <= not M;
            for n in 0 to ( 2**( F'length )-1 ) loop
              wait for 1ns;
              F      <= std_logic_vector( unsigned( F ) + 1 );
            end loop;
          end loop;
        end loop;
      wait;
    end process;
end;

```

```

-- *****
-- **** STUDENT: 64200238
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY alu_cla_tb IS
    GENERIC ( n : natural := 8 );
END alu_cla_tb;

ARCHITECTURE behavior OF alu_cla_tb IS
    COMPONENT alu_cla
        GENERIC ( n : natural := 8 );
        PORT (
            M : IN std_logic;
            F : IN std_logic_vector( 2 downto 0 );
            X : IN std_logic_vector( n-1 downto 0 );
            Y : IN std_logic_vector( n-1 downto 0 );
            S : OUT std_logic_vector( n-1 downto 0 );
            Negative : OUT std_logic;
            Cout : OUT std_logic;
            Overflow : OUT std_logic;
            Zero : OUT std_logic;
            Gout : OUT std_logic;
            Pout : OUT std_logic
        );
    END COMPONENT;

    -- Inputs
    SIGNAL M : std_logic := '0';
    SIGNAL F : std_logic_vector( 2 downto 0 ) := ( others => '0' );
    SIGNAL X : std_logic_vector( n-1 downto 0 ) := ( others => '0' );
    SIGNAL Y : std_logic_vector( n-1 downto 0 ) := ( others => '0' );

    -- Outputs
    SIGNAL S : std_logic_vector( n-1 downto 0 );

```



```

SIGNAL      Negative : std_logic;
SIGNAL      Cout : std_logic;
SIGNAL      Overflow : std_logic;
SIGNAL      Zero : std_logic;
SIGNAL      Gout : std_logic;
SIGNAL      Pout : std_logic;

SIGNAL      X_int, Y_int : integer := 0;
SIGNAL      S_comp, X_and_Y, X_nand_Y, X_or_Y, X_nor_Y,
X_xor_Y, X_xnor_Y : std_logic_vector( n-1 downto 0 ) := ( others => '0' );
SIGNAL      Neg_comp, Over_comp, Zero_comp : std_logic := '0';

CONSTANT    zeros : std_logic_vector( n-1 downto 0 ) := ( others => '0' );

```

```
BEGIN
```

```

uut: alu_cla
GENERIC MAP ( n => n )
PORT MAP (
M => M,
F => F,
X => X,
Y => Y,
S => S,
Negative => Negative,
Cout => Cout,
Overflow => Overflow,
Zero => Zero,
Gout => Gout,
Pout => Pout
);

-- Stimulus process
stim_proc: process
BEGIN
FOR i IN -( 2**( n-1 ) ) TO 2**( n-1 ) - 1 LOOP
X_int <= i;
X      <= std_logic_vector( to_signed( i, n ) );

FOR j IN -( 2**( n-1 ) ) TO 2**( n-1 ) - 1 LOOP

```

```

Y_int <= j;
Y      <= std_logic_vector( to_signed( j, n ) );

    -- Arithmetic operations
M      <= '0';      -- Arithmetic mode
F      <= "000";    -- X + Y
WAIT FOR 1 ns;
ASSERT( S_comp = S ) REPORT "Sum failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F      <= "001";    -- X - Y
WAIT FOR 1 ns;
ASSERT( S_comp = S ) REPORT "Dif failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F      <= "010";    -- X + 1
WAIT FOR 1 ns;
ASSERT( S_comp = S ) REPORT "X+1 failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F      <= "011";    -- X - 1
WAIT FOR 1 ns;
ASSERT( S_comp = S ) REPORT "X-1 failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F      <= "100";    -- X + X
WAIT FOR 1 ns;
ASSERT( S_comp = S ) REPORT "X+X failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F      <= "101";    -- -1
WAIT FOR 1 ns;
ASSERT( S_comp = S ) REPORT "-1 failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F      <= "110";    -- Undefined operation
WAIT FOR 1 ns;
ASSERT( zeros = S ) REPORT "0110 failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F      <= "111";    -- Undefined operation
WAIT FOR 1 ns;
ASSERT( zeros = S ) REPORT "0111 failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

    -- Logical operations
M      <= '1';      -- Logic mode
F      <= "000";    -- X AND Y

```

```

WAIT FOR 1 ns;
ASSERT( X_and_Y = S ) REPORT "X and Y failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "001";    -- X NAND Y
WAIT FOR 1 ns;
ASSERT( X_nand_Y = S ) REPORT "X nand Y failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "010";    -- X OR Y
WAIT FOR 1 ns;
ASSERT( X_or_Y = S ) REPORT "X or Y failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "011";    -- X NOR Y
WAIT FOR 1 ns;
ASSERT( X_nor_Y = S ) REPORT "X nor Y failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "100";    -- X XOR Y
WAIT FOR 1 ns;
ASSERT( X_xor_Y = S ) REPORT "X xor Y failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "101";    -- X XNOR Y
WAIT FOR 1 ns;
ASSERT( X_xnor_Y = S ) REPORT "X xnor Y failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "110";    -- X
WAIT FOR 1 ns;
ASSERT( X = S ) REPORT "X failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "111";    -- Y
WAIT FOR 1 ns;
ASSERT( Y = S ) REPORT "Y failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;
END LOOP;
END LOOP;
WAIT;
END PROCESS;

bog_pomagaj: process ( X_int, Y_int, F )
VARIABLE S_temp, X_temp, Y_temp : std_logic_vector( n-1 downto 0 );
VARIABLE Res_temp : integer;
BEGIN
X_temp := std_logic_vector( to_signed( X_int, n ) );

```

```

Y_temp := std_logic_vector( to_signed( Y_int, n ) );

IF F = "101" THEN
S_temp := NOT zeros;
Over_comp    <= '0';
ELSE
CASE F IS
WHEN "000" => Res_temp := X_int + Y_int;
WHEN "001" => Res_temp := X_int - Y_int;
WHEN "010" => Res_temp := X_int + 1;
WHEN "011" => Res_temp := X_int - 1;
WHEN "100" => Res_temp := X_int + X_int;
WHEN "110" => Res_temp := X_int;
WHEN "111" => Res_temp := Y_int;
WHEN OTHERS => Res_temp := 0;
END CASE;

IF Res_temp > 2**( n-1 ) - 1 OR Res_temp < -( 2**( n-1 ) ) THEN
Over_comp    <= '1';
ELSE
Over_comp    <= '0';
END IF;

S_temp := std_logic_vector( to_unsigned( Res_temp MOD 2**n, n ) );
END IF;

S_comp    <= S_temp;
Neg_comp  <= S_temp( n-1 );

IF S_temp = zeros THEN
Zero_comp  <= '1';
ELSE
Zero_comp  <= '0';
END IF;

X_and_Y    <= X_temp AND Y_temp;
X_nand_Y   <= X_temp NAND Y_temp;
X_or_Y     <= X_temp OR Y_temp;
X_nor_Y    <= X_temp NOR Y_temp;
X_xor_Y    <= X_temp XOR Y_temp;

```

```
X_xnor_Y    <= X_temp XNOR Y_temp;  
END PROCESS;  
  
END;
```

```

-- *****
-- **** STUDENT: 64200288
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Upoštevati morate dejansko zakasnitev ALU (glej vrednost combinatorial path delay), ne wait for 3 ns!
-- *****

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY alu_cla_tb IS
    GENERIC ( n : natural := 8 );
END alu_cla_tb;

ARCHITECTURE behavior OF alu_cla_tb IS
    COMPONENT alu_cla
        GENERIC ( n : natural := 8 );
        PORT (
            M : IN std_logic;
            F : IN std_logic_vector( 2 downto 0 );
            X : IN std_logic_vector( n-1 downto 0 );
            Y : IN std_logic_vector( n-1 downto 0 );
            S : OUT std_logic_vector( n-1 downto 0 );
            Negative : OUT std_logic;
            Cout : OUT std_logic;
            Overflow : OUT std_logic;
            Zero : OUT std_logic;
            Gout : OUT std_logic;
            Pout : OUT std_logic
        );
    END COMPONENT;

    -- Inputs
    SIGNAL M : std_logic := '0';
    SIGNAL F : std_logic_vector( 2 downto 0 ) := ( others => '0' );
    SIGNAL X : std_logic_vector( n-1 downto 0 ) := ( others => '0' );
    SIGNAL Y : std_logic_vector( n-1 downto 0 ) := ( others => '0' );

    -- Outputs
    SIGNAL S : std_logic_vector( n-1 downto 0 );

```

```

SIGNAL      Negative : std_logic;
SIGNAL      Cout : std_logic;
SIGNAL      Overflow : std_logic;
SIGNAL      Zero : std_logic;
SIGNAL      Gout : std_logic;
SIGNAL      Pout : std_logic;

SIGNAL      X_integer, Y_integer : integer := 0;
SIGNAL      S_computed, xandy, xnandy, xory, xnory,
xxory, xxnory : std_logic_vector( n-1 downto 0 ) := ( others => '0' );
SIGNAL      Neg_comp, OverflowDetect, Zero_comp : std_logic := '0';

CONSTANT    zeros : std_logic_vector( n-1 downto 0 ) := ( others => '0' );

```

```
BEGIN
```

```

uut: alu_cla
GENERIC MAP ( n => n )
PORT MAP (
M => M,
F => F,
X => X,
Y => Y,
S => S,
Negative => Negative,
Cout => Cout,
Overflow => Overflow,
Zero => Zero,
Gout => Gout,
Pout => Pout
);

-- Stimulus process
stim_proc: process
BEGIN
FOR i IN -( 2**( n-1 ) ) TO 2**( n-1 ) - 1 LOOP
X_integer <= i;
X <= std_logic_vector( to_signed( i, n ) );

FOR j IN -( 2**( n-1 ) ) TO 2**( n-1 ) - 1 LOOP

```

```

Y_integer    <= j;
Y            <= std_logic_vector( to_signed( j, n ) );

    -- Aritmeticni operatorji
M    <= '0';    -- Arithmetic mode
F    <= "000";    -- X + Y
WAIT FOR 3 ns;
ASSERT( S_computed = S ) REPORT "Sum failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "001";    -- X - Y
WAIT FOR 3 ns;
ASSERT( S_computed = S ) REPORT "Dif failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "010";    -- X + 1
WAIT FOR 3 ns;
ASSERT( S_computed = S ) REPORT "X+1 failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "011";    -- X - 1
WAIT FOR 3 ns;
ASSERT( S_computed = S ) REPORT "X-1 failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "100";    -- X + X
WAIT FOR 3 ns;
ASSERT( S_computed = S ) REPORT "X+X failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "101";    -- -1
WAIT FOR 3 ns;
ASSERT( S_computed = S ) REPORT "-1 failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "110";    -- Undefined operation
WAIT FOR 3 ns;
ASSERT( zeros = S ) REPORT "0110 failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "111";    -- Undefined operation
WAIT FOR 3 ns;
ASSERT( zeros = S ) REPORT "0111 failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

    -- Logicni operatorji
M    <= '1';    -- Logic mode
F    <= "000";    -- X AND Y

```



```

WAIT FOR 3 ns;
ASSERT( xandy = S ) REPORT "X and Y failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "001";    -- X NAND Y
WAIT FOR 3 ns;
ASSERT( xnandy = S ) REPORT "X nand Y failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "010";    -- X OR Y
WAIT FOR 3 ns;
ASSERT( xory = S ) REPORT "X or Y failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "011";    -- X NOR Y
WAIT FOR 3 ns;
ASSERT( xnory = S ) REPORT "X nor Y failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "100";    -- X XOR Y
WAIT FOR 3 ns;
ASSERT( xxory = S ) REPORT "X xor Y failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "101";    -- X XNOR Y
WAIT FOR 3 ns;
ASSERT( xxnory = S ) REPORT "X xnor Y failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "110";    -- X
WAIT FOR 3 ns;
ASSERT( X = S ) REPORT "X failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;

F    <= "111";    -- Y
WAIT FOR 3 ns;
ASSERT( Y = S ) REPORT "Y failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) SEVERITY ERROR;
END LOOP;
END LOOP;
WAIT;
END PROCESS;

myprocess: process ( X_int, Y_integer, F )
VARIABLE S_temporary, X_temporary, Y_temporary : std_logic_vector( n-1 downto 0 );
VARIABLE Rezultat_temporary : integer;
BEGIN
X_temporary := std_logic_vector( to_signed( X_int, n ) );

```

```

Y_temporary := std_logic_vector( to_signed( Y_int, n ) );

IF F = "101" THEN
S_temporary := NOT zeros;
OverflowDetect    <= '0';
ELSE

CASE F IS
WHEN "000" => Rezultat_temporary := X_integer + Y_integer;
WHEN "001" => Rezultat_temporary := X_integer - Y_integer;
WHEN "010" => Rezultat_temporary := X_integer + 1;
WHEN "011" => Rezultat_temporary := X_integer - 1;
WHEN "100" => Rezultat_temporary := X_integer + X_integer;
WHEN "110" => Rezultat_temporary := X_integer;
WHEN "111" => Rezultat_temporary := Y_integer;
WHEN OTHERS => Rezultat_temporary := 0;
END CASE;

IF Rezultat_temporary > 2** ( n-1 ) - 1 OR Rezultat_temporary < -( 2** ( n-1 ) ) THEN
OverflowDetect    <= '1';
ELSE
OverflowDetect    <= '0';
END IF;

S_temporary := std_logic_vector( to_unsigned( Rezultat_temporary MOD 2**n, n ) );
END IF;

S_computed    <= S_temporary;
Neg_comp      <= S_temporary( n-1 );

IF S_temporary = zeros THEN
Zero_comp    <= '1';
ELSE
Zero_comp    <= '0';
END IF;

xandy <= X_temporary AND Y_temporary;

xnandy      <= X_temporary NAND Y_temporary;

```

```
xory  <= X_temporary OR Y_temporary;  
xnory <= X_temporary NOR Y_temporary;  
xxory <= X_temporary XOR Y_temporary;  
xxnory      <= X_temporary XNOR Y_temporary;  
END PROCESS;  
END;
```

```

-- *****
-- **** STUDENT: 64200385
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Upoštevati morate dejansko zakasnitev ALU (glej vrednost combinatorial path delay), ne wait for 1 ns!
-- *****

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY alu_cla_tb IS
GENERIC( n: natural := 8 );
END alu_cla_tb;

ARCHITECTURE behavior OF alu_cla_tb IS
  COMPONENT alu_cla GENERIC( n: natural := 8 );
  PORT(
    M : IN std_logic;
    F : IN std_logic_vector( 2 downto 0 );
    X : IN std_logic_vector( n-1 downto 0 );
    Y : IN std_logic_vector( n-1 downto 0 );
    S : OUT std_logic_vector( n-1 downto 0 );
    Negative : OUT std_logic;
    Cout : OUT std_logic;
    Overflow : OUT std_logic;
    Zero : OUT std_logic;
    Gout : OUT std_logic;
    Pout : OUT std_logic
  );
  END COMPONENT;

  -- Inputs
  signal M : std_logic := '0';
  signal F : std_logic_vector( 2 downto 0 ) := ( others => '0' );
  signal X : std_logic_vector( n-1 downto 0 ) := ( others => '0' );
  signal Y : std_logic_vector( n-1 downto 0 ) := ( others => '0' );

  -- Outputs
  signal S : std_logic_vector( n-1 downto 0 );
  signal Negative : std_logic;

```

```

signal      Cout : std_logic;
signal      Overflow : std_logic;
signal      Zero : std_logic;
signal      Gout : std_logic;
signal      Pout : std_logic;
    signal   X_int, Y_int : integer := 0;
    signal   S_comp, X_and_Y, X_nand_Y, X_or_Y, X_nor_Y, X_xor_Y, X_xnor_Y : std_logic_vector( n-1 downto 0 )
:= ( others => '0' );
    signal   Neg_comp, Over_comp, Zero_comp : std_logic := '0';
    constant zeros : std_logic_vector( n-1 downto 0 ) := ( others => '0' );

```

BEGIN

```

    uut: alu_cla
        GENERIC MAP ( n => n )
        PORT MAP (
M => M,
F => F,
X => X,
Y => Y,
S => S,
Negative => Negative,
Cout => Cout,
Overflow => Overflow,
Zero => Zero,
Gout => Gout,
Pout => Pout
);

```

```

    stim_proc: process
begin

```

```

    for i in -( 2**( n-1 ) ) to 2**( n-1 ) - 1 loop
        X_int <= i;
        X      <= std_logic_vector( to_signed( i, n ) );
        for j in -( 2**( n-1 ) ) to 2**( n-1 ) - 1 loop
            Y_int <= j;
            Y      <= std_logic_vector( to_signed( j, n ) );

            M      <= '0';
            F      <= "000";
            wait for 1 ns;

```

```

        assert( S_comp = S ) report "Sum fail => X:" & integer'image( i ) & " Y:" & integer'image( j ) severity
error;
        assert( Neg_comp = Negative ) report "Sum N fail => X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
        assert( Over_comp = Overflow ) report "Sum V fail => X:" & integer'image( i ) & " Y:" & integer'image( j
) severity error;
        assert( Zero_comp = Zero ) report "Sum Z fail => X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;

        F      <= "001";
        wait for 1 ns;
        assert( S_comp = S ) report "Dif fail => X:" & integer'image( i ) & " Y:" & integer'image( j ) severity error;
        assert( Neg_comp = Negative ) report "Dif N fail => X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
        assert( Over_comp = Overflow ) report "Dif V fail => X:" & integer'image( i ) & " Y:" & integer'image( j
) severity error;
        assert( Zero_comp = Zero ) report "Dif Z fail => X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;

        F      <= "010";
        wait for 1 ns;
        assert( S_comp = S ) report "X+1 fail => X:" & integer'image( i ) & " Y:" & integer'image( j ) severity
error;
        assert( Neg_comp = Negative ) report "X+1 N fail => X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
        assert( Over_comp = Overflow ) report "X+1 V fail => X:" & integer'image( i ) & " Y:" & integer'image( j
) severity error;
        assert( Zero_comp = Zero ) report "X+1 Z fail => X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;

        F      <= "011";
        wait for 1 ns;
        assert( S_comp = S ) report "X-1 fail => X:" & integer'image( i ) & " Y:" & integer'image( j ) severity
error;
        assert( Neg_comp = Negative ) report "X-1 N fail => X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
        assert( Over_comp = Overflow ) report "X-1 V fail => X:" & integer'image( i ) & " Y:" & integer'image( j
) severity error;
        assert( Zero_comp = Zero ) report "X-1 Z fail => X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;

```

```

F      <= "100";
wait for 1 ns;
assert( S_comp = S ) report "X+X fail => X:" & integer'image( i ) & " Y:" & integer'image( j ) severity
error;
    assert( Neg_comp = Negative ) report "X+X N fail => X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
    assert( Over_comp = Overflow ) report "X+X V fail => X:" & integer'image( i ) & " Y:" & integer'image( j
) severity error;
    assert( Zero_comp = Zero ) report "X+X Z fail => X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;

F      <= "101";
wait for 1 ns;
assert( S_comp = S ) report "-1 fail => X:" & integer'image( i ) & " Y:" & integer'image( j ) severity
error;
    assert( Neg_comp = Negative ) report "-1 N fail => X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
    assert( Zero_comp = Zero ) report "-1 Z fail => X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;

F      <= "110";    -- undefined
wait for 1 ns;
assert( zeros = S ) report "0110 fail => X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;

F      <= "111";    -- undefined
wait for 1 ns;
assert( zeros = S ) report "0111 fail => X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;

M      <= '1';
F      <= "000";
wait for 1 ns;
assert( X_and_Y = S ) report "X and Y fail => X:" & integer'image( i ) & " Y:" & integer'image( j
) severity error;

F      <= "001";
wait for 1 ns;

```

```

        assert( X_nand_Y = S ) report "X nand Y fail => X:" & integer'image( i ) & " Y:" & integer'image(
j ) severity error;

        F      <= "010";
        wait for 1 ns;
        assert( X_or_Y = S ) report "X or Y fail => X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;

        F      <= "011";
        wait for 1 ns;
        assert( X_nor_Y = S ) report "X nor Y fail => X:" & integer'image( i ) & " Y:" & integer'image( j
) severity error;

        F      <= "100";
        wait for 1 ns;
        assert( X_xor_Y = S ) report "X xor Y fail => X:" & integer'image( i ) & " Y:" & integer'image( j
) severity error;

        F      <= "101";
        wait for 1 ns;
        assert( X_xnor_Y = S ) report "X xnor Y fail => X:" & integer'image( i ) & " Y:" & integer'image(
j ) severity error;

        F      <= "110";
        wait for 1 ns;
        assert( X = S ) report "X fail => X:" & integer'image( i ) & " Y:" & integer'image( j ) severity error;
        assert( Neg_comp = Negative ) report "X N fail => X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
        assert( Zero_comp = Zero ) report "X Z fail => X:" & integer'image( i ) & " Y:" & integer'image( j
) severity error;

        F      <= "111";
        wait for 1 ns;
        assert( Y = S ) report "Y fail => X:" & integer'image( i ) & " Y:" & integer'image( j ) severity error;
        assert( Neg_comp = Negative ) report "Y N fail => X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;
        assert( Zero_comp = Zero ) report "Y Z fail => X:" & integer'image( i ) & " Y:" & integer'image( j
) severity error;
        end loop;
    end loop;

```



```

    wait;
end process;
comp_proc: process ( X_int, Y_int, F )
variable S_temp, X_temp, Y_temp : std_logic_vector( n-1 downto 0 );
variable Res_temp : integer;
begin
x_temp := std_logic_vector( to_signed( X_int, n ) );
y_temp := std_logic_vector( to_signed( Y_int, n ) );
    if F = "101" then
        S_temp := not zeros;
        Over_comp    <= '0';
    else
        if F = "000" then
            Res_temp := X_int + Y_int;
        elsif F = "001" then
            Res_temp := X_int - Y_int;
        elsif F = "010" then
            Res_temp := X_int + 1;
        elsif F = "011" then
            Res_temp := X_int - 1;
        elsif F = "100" then
            Res_temp := X_int + X_int;
        elsif F = "110" then
            Res_temp := X_int;
        elsif F = "111" then
            Res_temp := Y_int;
        else
            Res_temp := 0;
        end if;
        if Res_temp > 2**( n-1 ) - 1 or Res_temp < -( 2**( n-1 ) ) then
            Over_comp    <= '1';
        else
            Over_comp    <= '0';
        end if;
        S_temp := std_logic_vector( to_unsigned( Res_temp mod 2**n, n ) );
    end if;
    S_comp <= S_temp;
    Neg_comp    <= S_temp( n-1 );

    if S_temp = zeros then

```

```
        Zero_comp    <= '1';
    else
        Zero_comp    <= '0';
    end if;
    X_and_Y          <= X_temp and Y_temp;
    X_nand_Y         <= X_temp nand Y_temp;
    X_or_Y <= X_temp or Y_temp;
    X_nor_Y          <= X_temp nor Y_temp;
    X_xor_Y          <= X_temp xor Y_temp;
    X_xnor_Y         <= X_temp xnor Y_temp;
end process;
```

```
END;
```

```

-- *****
-- **** STUDENT: 64210113
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Upoštevati morate dejansko zakasnitev ALU (glej vrednost combinatorial path delay), ne wait for 1 ns!
-- *****

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity alu_cla_tb is
    generic ( n: natural := 8 );
end alu_cla_tb;

architecture behavior of alu_cla_tb is
    component alu_cla
        generic ( n: natural := 8 );
        port (
            M : in std_logic;
            F : in std_logic_vector( 2 downto 0 );
            X : in std_logic_vector( n-1 downto 0 );
            Y : in std_logic_vector( n-1 downto 0 );
            S : out std_logic_vector( n-1 downto 0 );
            Negative : out std_logic;
            Cout : out std_logic;
            Overflow : out std_logic;
            Zero : out std_logic;
            Gout : out std_logic;
            Pout : out std_logic
        );
    end component;

    signal M : std_logic := '0';
    signal F : std_logic_vector( 2 downto 0 ) := ( others => '0' );
    signal X : std_logic_vector( n-1 downto 0 ) := ( others => '0' );
    signal Y : std_logic_vector( n-1 downto 0 ) := ( others => '0' );

    signal S : std_logic_vector( n-1 downto 0 );
    signal Negative : std_logic;
    signal Cout : std_logic;

```

```

signal      Overflow : std_logic;
signal      Zero : std_logic;
signal      Gout : std_logic;
signal      Pout : std_logic;

signal      X_val, Y_val : integer := 0;
signal      S_result, X_and_Y, X_nand_Y, X_or_Y, X_nor_Y, X_xor_Y, X_xnor_Y : std_logic_vector( n-1 downto 0
):= ( others=> '0' );
signal      Neg_result, Overflow_result, Zero_result : std_logic := '0';
constant    zero_val : std_logic_vector( n-1 downto 0 ):= ( others=> '0' );

begin

 uut: alu_cla
   generic map ( n => n )
   port map(
     M => M,          F => F,          X => X,          Y => Y,          S => S,
     Negative => Negative, Cout => Cout,          Overflow => Overflow, Zero => Zero,
     Gout => Gout,      Pout => Pout
   );

 stimulus_process: process
 begin
   for i in -( 2**( n-1 ) ) to ( 2**( n-1 ) ) - 1 loop
     X   <= std_logic_vector( to_signed( i, n ) );
     X_val <= i;

     for j in -( 2**( n-1 ) ) to ( 2**( n-1 ) ) - 1 loop
       Y   <= std_logic_vector( to_signed( j, n ) );
       Y_val <= j;

       M   <= '0';      -- Arithmetics mode

       -- X + Y

       F   <= "000";
       wait for 1 ns;
       assert( S_result = S )
       report "Sestevanje error.x:" & integer'image( i ) & "y: " & integer'image( j )
       severity error;
       assert( Neg_result = Negative )

```

```

report "Ses n error. x:" & integer'image( i ) & "y:" & integer'image( j )
severity error;
assert( Overflow_result = Overflow )
report "ses v error. x:" & integer'image( i ) & "y:" & integer'image( j )
severity error;
assert( Zero_result = Zero )
report "Ses z error. x:" & integer'image( i ) & "y:" & integer'image( j )
severity error;

-- X - Y
F      <= "001";
wait for 1 ns;
assert( S_result = S )
report "Odstevanje error.x: " & integer'image( i ) & "y: " & integer'image( j )
severity error;
assert( Neg_result = Negative )
report "Odstevanje n error. X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
assert( Overflow_result = Overflow )
report "Odstevanje v error. X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
assert( Zero_result = Zero )
report "Odstevanje z error. X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;

-- X + 1
F      <= "010";
wait for 1 ns;
assert( S_result = S )
report "x + 1 error.x: " & integer'image( i ) & "y " & integer'image( j )
severity error;
assert( Neg_result = Negative )
report "x+1 n error. x:" & integer'image( i ) & "y:" & integer'image( j )
severity error;
assert( Overflow_result = Overflow )
report "x+1 v errorr x:." & integer'image( i ) & "y:" & integer'image( j )
severity error;
assert( Zero_result = Zero )
report "x+1 z error. x:" & integer'image( i ) & "y" & integer'image( j )
severity error;

```

-- x - 1

```
F      <= "011";
wait for 1 ns;
assert( S_result = S )
report "x - 1 error.x: " & integer'image( i ) & "y: " & integer'image( j )
severity error;
assert( Neg_result = Negative )
report "x-1 n error. X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
assert( Overflow_result = Overflow )
report "x-1 v error. X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
assert( Zero_result = Zero )
report "x-1 z error. X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
```

-- x+x

```
F      <= "100";
wait for 1 ns;
assert( S_result = S )
report "x + x error.x: " & integer'image( i ) & "y: " & integer'image( j )
severity error;
assert( Neg_result = Negative )
report "X+X N error. X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
assert( Overflow_result = Overflow )
report "X+X V error. X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
assert( Zero_result = Zero )
report "X+X Z error. X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
```

-- -1

```
F      <= "101";
wait for 1 ns;
assert( S_result = S )
report "negacija error.x: " & integer'image( i ) & "y: " & integer'image( j )
severity error;
assert( Neg_result = Negative )
```

```

report "-1 N failed. X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
assert( Zero_result = Zero )
report "-1 Z failed. X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;

-- nedefinirano
F      <= "110";
wait for 1 ns;
assert( zero_val = S )
report "Nedefiniran operator error.x: " & integer'image( i ) & "y: " & integer'image( j )
severity error;

-- nedefinirano
F      <= "111";
wait for 1 ns;
assert( zero_val = S )
REPORT "Nedefiniran operator error.x: " & integer'image( i ) & " y: " & integer'image( j )
severity error;

M      <= '1';      -- Logical mode

-- X and Y
F      <= "000";
wait for 1 ns;
assert( X_and_Y = S )
report "and error. X: " & integer'image( i ) & "y: " & integer'image( j )
severity error;

-- X nand Y
F      <= "001";
wait for 1 ns;
assert( X_nand_Y = S )
report "nand error. X: " & integer'image( i ) & "y: " & integer'image( j )
severity error;

-- X or Y
F      <= "010";
wait for 1 ns;
assert( X_or_Y = S )

```

```

report "or error. X: " & integer'image( i ) & "y: " & integer'image( j )
severity error;

-- X nor Y
F      <= "011";
wait for 1 ns;
assert( X_nor_Y = S )
report "nor error. X: " & integer'image( i ) & " Y: " & integer'image( j )
severity error;

-- X Xxor Y
F      <= "100";
wait for 1 ns;
assert( X_xor_Y = S )
report "xor error. X: " & integer'image( i ) & "y: " & integer'image( j )
severity error;

-- X xnor Y
F      <= "101";
wait for 1 ns;
assert( X_xnor_Y = S )
report "xnor error. X: " & integer'image( i ) & "y: " & integer'image( j )
severity error;

-- X
F      <= "110";
wait for 1 ns;
assert( X = S )
report "x error. X: " & integer'image( i ) & "y: " & integer'image( j )
severity error;
assert( Neg_result = Negative )
report "X N failed. X:" & integer'image( i ) & " y:" & integer'image( j )
severity error;
assert( Zero_result = Zero )
report "X Z failed. X:" & integer'image( i ) & " y:" & integer'image( j )
severity error;

-- Y
F      <= "111";
wait for 1 ns;

```



```

        assert( Y = S )
        report "x error. X: " & integer'image( i ) & "y: " & integer'image( j )
        severity error;
        assert( Neg_result = Negative )
        report "Y N failed. X:" & integer'image( i ) & " y:" & integer'image( j )
        severity error;
        assert( Zero_result = Zero )
        report "Y Z failed. X:" & integer'image( i ) & " y:" & integer'image( j )
        severity error;

    end loop;

end loop;
wait;
end process;

calculation_process: process ( X_val, Y_val, F )
    variable Sdel, Xdel, Ydel : std_logic_vector( n-1 downto 0 );
    variable Rez : integer;

begin

    Xdel:= std_logic_vector( to_signed( X_val,n ) );
    Ydel:= std_logic_vector( to_signed( Y_val,n ) );

    if F = "101" then Sdel:= not zero_val;
        Overflow_result    <= '0';
    else
        case F is
            when "000" => Rez := X_val + Y_val;
            when "001" => Rez := X_val - Y_val;
            when "010" => Rez := X_val + 1;
            when "011" => Rez := X_val - 1;
            when "100" => Rez := X_val + X_val;
            when "110" => Rez := X_val;
            when "111" => Rez := Y_val;
            when others => Rez := 0;
        end case;

    if Rez > 2**( n-1 ) - 1 or Rez < -( 2**( n-1 ) ) then
        Overflow_result    <= '1';
    end if;
end process;

```

```

else
    Overflow_result    <= '0';
end if;
    Sdel := std_logic_vector( to_unsigned( Rez mod 2**n, n ) );
end if;

    S_result    <= Sdel;
    Neg_result  <= Sdel( n-1 );

if Sdel = zero_val then
    Zero_result  <= '1';
else Zero_result  <= '0';
end if;

    X_nand_Y    <= Xdel nand Ydel;
    X_and_Y     <= Xdel and Ydel;
    X_xor_Y     <= Xdel xor Ydel;
    X_xnor_Y    <= Xdel xnor Ydel;
    X_or_Y      <= Xdel or Ydel;
    X_nor_Y     <= Xdel nor Ydel;

    end process;

end;

```

```

-- *****
-- **** STUDENT: 64210382
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Upoštevati morate dejansko zakasnitev ALU (glej vrednost combinatorial path delay), ne wait for 1 ns!
-- *****

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY alu_cla_tb IS
    GENERIC( n: natural := 8 );
END alu_cla_tb;

ARCHITECTURE behavior OF alu_cla_tb IS
    COMPONENT alu_cla
        GENERIC( n: natural := 8 );
    PORT(
        M : IN std_logic;
        F : IN std_logic_vector( 2 downto 0 );
        X : IN std_logic_vector( n-1 downto 0 );
        Y : IN std_logic_vector( n-1 downto 0 );
        S : OUT std_logic_vector( n-1 downto 0 );
        Negative : OUT std_logic;
        Cout : OUT std_logic;
        Overflow : OUT std_logic;
        Zero : OUT std_logic;
        Gout : OUT std_logic;
        Pout : OUT std_logic
    );
END COMPONENT;

-- Inputs
signal M : std_logic := '0';
signal F : std_logic_vector( 2 downto 0 ) := ( others => '0' );
signal X : std_logic_vector( n-1 downto 0 ) := ( others => '0' );
signal Y : std_logic_vector( n-1 downto 0 ) := ( others => '0' );

-- Outputs
signal S : std_logic_vector( n-1 downto 0 );

```

```

signal      Negative : std_logic;
signal      Cout     : std_logic;
signal      Overflow  : std_logic;
signal      Zero      : std_logic;
signal      Gout      : std_logic;
signal      Pout      : std_logic;

      signal      X_int, Y_int : integer := 0;
      signal      S_comp, X_and_Y,      X_nand_Y, X_or_Y, X_nor_Y,      X_xor_Y, X_xnor_Y :
std_logic_vector( n-1 downto 0 ) := ( others => '0' );
      signal      Neg_comp, Over_comp, Zero_comp : std_logic := '0';

      constant    zeros : std_logic_vector( n-1 downto 0 ) := ( others => '0' );

BEGIN
  uut: alu_cla
    GENERIC MAP ( n => n )
    PORT MAP (
M => M,
F => F,
X => X,
Y => Y,
S => S,
Negative => Negative,
Cout => Cout,
Overflow => Overflow,
Zero => Zero,
Gout => Gout,
Pout => Pout
);

  stim_proc: process
  begin
    for i in -( 2**( n-1 ) ) to 2**( n-1 ) - 1 loop
      X_int <= i;
      X      <= std_logic_vector( to_signed( i, n ) );
      for j in -( 2**( n-1 ) ) to 2**( n-1 ) - 1 loop
        Y_int <= j;
        Y      <= std_logic_vector( to_signed( j, n ) );

```

```

M      <= '0';      -- Aritmetièni naèin
F      <= "000";    -- X + Y
wait for 1 ns;
assert( S_comp = S ) report "Sum failed. X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
    assert( Neg_comp = Negative ) report "Sum N failed. X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;
    assert( Over_comp = Overflow ) report "Sum V failed. X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;
    assert( Zero_comp = Zero ) report "Sum Z failed. X:" & integer'image( i ) & " Y:" & integer'image(
j ) severity error;

F      <= "001";    -- X - Y
wait for 1 ns;
assert( S_comp = S ) report "Dif failed. X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
    assert( Neg_comp = Negative ) report "Dif N failed. X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;
    assert( Over_comp = Overflow ) report "Dif V failed. X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;
    assert( Zero_comp = Zero ) report "Dif Z failed. X:" & integer'image( i ) & " Y:" & integer'image(
j ) severity error;

F      <= "010";    -- X + 1
wait for 1 ns;
assert( S_comp = S ) report "X+1 failed. X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
    assert( Neg_comp = Negative ) report "X+1 N failed. X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;
    assert( Over_comp = Overflow ) report "X+1 V failed. X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;
    assert( Zero_comp = Zero ) report "X+1 Z failed. X:" & integer'image( i ) & " Y:" & integer'image(
j ) severity error;

F      <= "011";    -- X - 1
wait for 1 ns;
assert( S_comp = S ) report "X-1 failed. X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
    assert( Neg_comp = Negative ) report "X-1 N failed. X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;

```

```

        assert( Over_comp = Overflow ) report "X-1 V failed. X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;
        assert( Zero_comp = Zero ) report "X-1 Z failed. X:" & integer'image( i ) & " Y:" & integer'image(
j ) severity error;

        F      <= "100";    -- X + X
        wait for 1 ns;
        assert( S_comp = S ) report "X+X failed. X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
        assert( Neg_comp = Negative ) report "X+X N failed. X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;
        assert( Over_comp = Overflow ) report "X+X V failed. X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;
        assert( Zero_comp = Zero ) report "X+X Z failed. X:" & integer'image( i ) & " Y:" & integer'image(
j ) severity error;

        F      <= "101";    -- -1
        wait for 1 ns;
        assert( S_comp = S ) report "-1 failed. X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
        assert( Neg_comp = Negative ) report "-1 N failed. X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;
        assert( Zero_comp = Zero ) report "-1 Z failed. X:" & integer'image( i ) & " Y:" & integer'image(
j ) severity error;

        F      <= "110";    -- undefined operation
        wait for 1 ns;
        assert( zeros = S ) report "0110 failed. X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;

        F      <= "111";    -- undefined operation
        wait for 1 ns;
        assert( zeros = S ) report "0111 failed. X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;

        M      <= '1';      -- Logièni naèin
        F      <= "000";    -- X and Y
        wait for 1 ns;
        assert( X_and_Y = S ) report "X and Y failed. X:" & integer'image( i ) & " Y:" & integer'image( j
) severity error;

```

```

F      <= "001";    -- X nand Y
wait for 1 ns;
assert( X_nand_Y = S ) report "X nand Y failed. X:" & integer'image( i ) & " Y:" & integer'image(
j ) severity error;

F      <= "010";    -- X or Y
wait for 1 ns;
assert( X_or_Y = S ) report "X or Y failed. X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;

F      <= "011";    -- X nor Y
wait for 1 ns;
assert( X_nor_Y = S ) report "X nor Y failed. X:" & integer'image( i ) & " Y:" & integer'image( j
) severity error;

F      <= "100";    -- X xor Y
wait for 1 ns;
assert( X_xor_Y = S ) report "X xor Y failed. X:" & integer'image( i ) & " Y:" & integer'image( j
) severity error;

F      <= "101";    -- X xnor Y
wait for 1 ns;
assert( X_xnor_Y = S ) report "X xnor Y failed. X:" & integer'image( i ) & " Y:" & integer'image(
j ) severity error;

F      <= "110";    -- X
wait for 1 ns;
assert( X = S ) report "X failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) severity
error;
assert( Neg_comp = Negative ) report "X N failed. X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;
assert( Zero_comp = Zero ) report "X Z failed. X:" & integer'image( i ) & " Y:" & integer'image( j
) severity error;

F      <= "111";    -- Y
wait for 1 ns;
assert( Y = S ) report "Y failed. X:" & integer'image( i ) & " Y:" & integer'image( j ) severity
error;

```

```

        assert( Neg_comp = Negative ) report "Y N failed. X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;
        assert( Zero_comp = Zero ) report "Y Z failed. X:" & integer'image( i ) & " Y:" & integer'image( j
) severity error;
        end loop;
    end loop;
wait;
end process;

-- Calculate sum for checking the adder result
comp_proc: process ( X_int, Y_int, F )
variable S_temp, X_temp, Y_temp : std_logic_vector( n-1 downto 0 );
variable Res_temp : integer;
begin
    X_temp := std_logic_vector( to_signed( X_int, n ) );
    Y_temp := std_logic_vector( to_signed( Y_int, n ) );

    if F = "101" then
        S_temp := not zeros;
        Over_comp <= '0';
    else
        if F = "000" then
            Res_temp := X_int + Y_int;
        elsif F = "001" then
            Res_temp := X_int - Y_int;
        elsif F = "010" then
            Res_temp := X_int + 1;
        elsif F = "011" then
            Res_temp := X_int - 1;
        elsif F = "100" then
            Res_temp := X_int + X_int;
        elsif F = "110" then
            Res_temp := X_int;
        elsif F = "111" then
            Res_temp := Y_int;
        else
            Res_temp := 0;
        end if;

        if Res_temp > 2** ( n-1 ) - 1 or Res_temp < -( 2** ( n-1 ) ) then

```



```
        Over_comp    <= '1';
    else
        Over_comp    <= '0';
    end if;
    S_temp := std_logic_vector( to_unsigned( Res_temp mod 2**n, n ) );
end if;
```

```
S_comp <= S_temp;
Neg_comp    <= S_temp( n-1 );
```

```
if S_temp = zeros then
    Zero_comp    <= '1';
else
    Zero_comp    <= '0';
end if;
```

```
X_and_Y      <= X_temp and Y_temp;
X_nand_Y     <= X_temp nand Y_temp;
X_or_Y <= X_temp or Y_temp;
X_nor_Y      <= X_temp nor Y_temp;
X_xor_Y      <= X_temp xor Y_temp;
X_xnor_Y     <= X_temp xnor Y_temp;
```

```
end process;
```

```
END;
```

```

-- *****
-- **** STUDENT: 64210384
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek:
Povezava med alu_cla in testbench mora biti parametrizirana (manjka generic map (n=>n))
Upoštevati morate dejansko zakasnitev ALU (glej vrednost combinatorial path delay), ne wait for 100 ps!
-- *****

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity alu_cla_tb is
    generic ( n : natural := 8 );
end alu_cla_tb;

architecture dn2 of alu_cla_tb is

    COMPONENT alu_cla
    PORT(
        M : IN std_logic;
        F : IN std_logic_vector( 2 downto 0 );
        X : IN std_logic_vector( n-1 downto 0 );
        Y : IN std_logic_vector( n-1 downto 0 );
        S : OUT std_logic_vector( n-1 downto 0 );
        Negative : OUT std_logic;
        Cout : OUT std_logic;
        Overflow : OUT std_logic;
        Zero : OUT std_logic;
        Gout : OUT std_logic;
        Pout : OUT std_logic
    );
    END COMPONENT;

    -- Inputs
    signal M : std_logic := '0';
    signal F : std_logic_vector( 2 downto 0 ) := ( others => '0' );
    signal X : std_logic_vector( n-1 downto 0 ) := ( others => '0' );
    signal Y : std_logic_vector( n-1 downto 0 ) := ( others => '0' );

```

```

-- Outputs
signal      S : std_logic_vector( n-1 downto 0 );
signal      Negative : std_logic;
signal      Cout : std_logic;
signal      Overflow : std_logic;
signal      Zero : std_logic;
signal      Gout : std_logic;
signal      Pout : std_logic;

      signal      clock : std_logic;
constant    clock_period : time := 100 ps;

      signal      helper_i, helper_j : integer range -2**( n-1 ) to 2**( n-1 )-1;
      signal      rezultat : integer range -2**n to 2**n-1;

      signal      neg, z, v : std_logic;
      signal      izhod : std_logic_vector( n-1 downto 0 );

begin

uut: alu_cla
  port map(
    M => M,      F => F,      x => x,      y => y,      s => s,      negative => negative,      cout => cout,
    overflow => overflow,      zero => zero,      gout => gout,      pout => pout
  );

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

stimulus_process: process
begin
z1:  for a in 0 to 1 loop
      for b in 0 to 7 loop
        for i in -2**( n-1 ) to 2**( n-1 )-1 loop
          for j in -2**( n-1 ) to 2**( n-1 )-1 loop

```

```

        F      <= std_logic_vector( to_unsigned( b, 3 ) );
        x      <= std_logic_vector( to_signed( i, n ) );
        y      <= std_logic_vector( to_signed( j, n ) );
        helper_i    <= i;
        helper_j    <= j;
        wait for clock_period;
        assert ( s = izhod )
        report "Test ( code " & integer'image( a ) & integer'image( b ) &
            " ) failed for x = " & integer'image( i ) & " and y = " &
            integer'image( j ) & "." severity error;
        exit z1 when ( s /= izhod );
        if M = '0' then
            assert ( negative = neg )
            report "Negativity test ( code " & integer'image( a ) & integer'image( b ) &
                " ) failed for x = " & integer'image( i ) & " and y = " &
                integer'image( j ) & "." severity error;
            assert ( zero = z )
            report "Zero test ( code " & integer'image( a ) & integer'image( b ) &
                " ) failed for x = " & integer'image( i ) & " and y = " &
                integer'image( j ) & "." severity error;
            assert ( overflow = v )
            report "Overflow test ( code " & integer'image( a ) & integer'image( b ) &
                " ) failed for x = " & integer'image( i ) & " and y = " &
                integer'image( j ) & "." severity error;
            exit z1 when ( negative /= neg or zero /= z or overflow /= v );
        end if;
    end loop;
end loop;
end loop;
end loop;
M      <= not M;
end loop;
wait;
end process;

```

```

posodobitev_process: process ( helper_i, helper_j, izhod )
    variable rezultat : integer range -2**n to 2**n-1;  -- zakasnitev 1 clock periode ( !! )
    variable helper_x, helper_y, helper_izhod : std_logic_vector( n-1 downto 0 );
begin
    if M = '0' then
        case F is

```

```

when "000" => rezultat := helper_i + helper_j;
when "001" => rezultat := helper_i - helper_j;
when "010" => rezultat := helper_i + 1;
when "011" => rezultat := helper_i - 1;
when "100" => rezultat := helper_i + helper_i;
when "101" => rezultat := -1;
when others => rezultat := 0;
end case;
if ( rezultat >= -2**( n-1 ) and rezultat < 0 )
or ( rezultat >= 2**( n-1 ) and rezultat < 2**n ) then
neg    <= '1';
else
neg    <= '0';
end if;
if rezultat mod 2**n = 0 then
z      <= '1';
else
z      <= '0';
end if;
if rezultat < -2**( n-1 ) or rezultat >= 2**( n-1 ) then
v      <= '1';
else
v      <= '0';
end if;
helper_izhod := std_logic_vector( to_unsigned( rezultat mod 2**n, n ) );
else
helper_x := std_logic_vector( to_signed( helper_i, n ) );
helper_y := std_logic_vector( to_signed( helper_j, n ) );
case F is
when "000" => helper_izhod := helper_x and helper_y;
when "001" => helper_izhod := helper_x nand helper_y;
when "010" => helper_izhod := helper_x or helper_y;
when "011" => helper_izhod := helper_x nor helper_y;
when "100" => helper_izhod := helper_x xor helper_y;
when "101" => helper_izhod := helper_x xnor helper_y;
when "110" => helper_izhod := helper_x;
when "111" => helper_izhod := helper_y;
when others => report "ne bo.";
end case;
end if;

```

```
        izhod <= helper_izhod;  
    end process;  
end;
```

```

-- *****
-- **** STUDENT: 64210386
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Upoštevati morate dejansko zakasnitev ALU (glej vrednost combinatorial path delay), ne wait for 10 ns!
Povezava med alu_cla in testbench mora biti parametrizirana (manjka generic map (n=>n))
-- *****
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

-- Uncomment the following Library declaration if using
-- arithmetic functions with Signed or Unsigned values
-- USE ieee.numeric_std.ALL;

ENTITY alu_cla_tb IS
    GENERIC( n: natural := 8 );
END alu_cla_tb;

ARCHITECTURE behavior OF alu_cla_tb IS

    -- Component Declaration for the Unit Under Test ( UUT )

    COMPONENT alu_cla
    PORT(
        M : IN std_logic;
        F : IN std_logic_vector( 2 downto 0 );
        X : IN std_logic_vector( 7 downto 0 );
        Y : IN std_logic_vector( 7 downto 0 );
        S : OUT std_logic_vector( 7 downto 0 );
        Negative : OUT std_logic;
        Cout : OUT std_logic;
        Overflow : OUT std_logic;
        Zero : OUT std_logic;
        Gout : OUT std_logic;
        Pout : OUT std_logic
    );
    END COMPONENT;

    -- Inputs

```

```

signal      M : std_logic := '0';
signal      F : std_logic_vector( 2 downto 0 ) := ( others => '0' );
signal      X : std_logic_vector( n-1 downto 0 ) := ( others => '0' );
signal      Y : std_logic_vector( n-1 downto 0 ) := ( others => '0' );

-- Outputs
signal      S : std_logic_vector( n-1 downto 0 );
signal      Negative : std_logic;
signal      Cout : std_logic;
signal      Overflow : std_logic;
signal      Zero : std_logic;
signal      Gout : std_logic;
signal      Pout : std_logic;
-- No clocks detected in port list. Replace <clock> below with
-- appropriate port name

constant    clock_period : time := 10 ns;

signal      Xi, Yi : integer := 0;
signal      S_test, XandY,          XnandY, XorY, XnorY,          XxorY, XxnorY : std_logic_vector( n-1
downto 0 ) := ( others => '0' );
signal      N_test, OF_test, Z_test : std_logic := '0';

constant    nicle : std_logic_vector( n-1 downto 0 ) := ( others => '0' );

BEGIN

-- Instantiate the Unit Under Test ( UUT )
 uut: alu_cla PORT MAP (
M => M,
F => F,
X => X,
Y => Y,
S => S,
Negative => Negative,
Cout => Cout,
Overflow => Overflow,
Zero => Zero,
Gout => Gout,
Pout => Pout

```



```

);

-- Stimulus process
stim_proc: process
begin

for i in -( 2**( n-1 ) ) to 2**( n-1 ) - 1 loop
    Xi    <= i;
    X     <= std_logic_vector( to_signed( i, n ) );
    for u in -( 2**( n-1 ) ) to 2**( n-1 ) - 1 loop
        Yi    <= u;
        Y     <= std_logic_vector( to_signed( u, n ) );

        M     <= '0';
        F     <= "000";
        wait for clock_period;

        assert( S_test = S ) report "SUM failed: X:" & integer'image( i ) & " Y:" & integer'image( u )
severity error;
        assert( N_test = Negative ) report "SUM N failed: X:" & integer'image( i ) & " Y:" &
integer'image( u ) severity error;
        assert( OF_test = Overflow ) report "SUM V failed: X:" & integer'image( i ) & " Y:" &
integer'image( u ) severity error;
        assert( Z_test = Zero ) report "SUM Z failed: X:" & integer'image( i ) & " Y:" & integer'image( u
) severity error;

        F     <= "001";
        wait for clock_period;

        assert( S_test = S ) report "DIF failed: X:" & integer'image( i ) & " Y:" & integer'image( u )
severity error;
        assert( N_test = Negative ) report "DIF N failed: X:" & integer'image( i ) & " Y:" &
integer'image( u ) severity error;
        assert( OF_test = Overflow ) report "DIF V failed: X:" & integer'image( i ) & " Y:" &
integer'image( u ) severity error;
        assert( Z_test = Zero ) report "DIF Z failed: X:" & integer'image( i ) & " Y:" & integer'image( u
) severity error;

        F     <= "010";
        wait for clock_period;

```

```

        assert( S_test = S ) report "X+1 failed: X:" & integer'image( i ) & " Y:" & integer'image( u )
severity error;
        assert( N_test = Negative ) report "X+1 N failed: X:" & integer'image( i ) & " Y:" &
integer'image( u ) severity error;
        assert( OF_test = Overflow ) report "X+1 V failed: X:" & integer'image( i ) & " Y:" &
integer'image( u ) severity error;
        assert( Z_test = Zero ) report "X+1 Z failed: X:" & integer'image( i ) & " Y:" & integer'image( u
) severity error;

        F      <= "011";
        wait for clock_period;

        assert( S_test = S ) report "X-1 failed: X:" & integer'image( i ) & " Y:" & integer'image( u )
severity error;
        assert( N_test = Negative ) report "X-1 N failed: X:" & integer'image( i ) & " Y:" &
integer'image( u ) severity error;
        assert( OF_test = Overflow ) report "X-1 V failed: X:" & integer'image( i ) & " Y:" &
integer'image( u ) severity error;
        assert( Z_test = Zero ) report "X-1 Z failed: X:" & integer'image( i ) & " Y:" & integer'image( u
) severity error;

        F      <= "100";
        wait for clock_period;

        assert( S_test = S ) report "X+X failed: X:" & integer'image( i ) & " Y:" & integer'image( u )
severity error;
        assert( N_test = Negative ) report "X+X N failed: X:" & integer'image( i ) & " Y:" &
integer'image( u ) severity error;
        assert( OF_test = Overflow ) report "X+X V failed: X:" & integer'image( i ) & " Y:" &
integer'image( u ) severity error;
        assert( Z_test = Zero ) report "X+X Z failed: X:" & integer'image( i ) & " Y:" & integer'image( u
) severity error;

        F      <= "101";
        wait for clock_period;

        assert( S_test = S ) report "-1 failed: X:" & integer'image( i ) & " Y:" & integer'image( u )
severity error;

```

```

    assert( N_test = Negative ) report "-1 N failed: X:" & integer'image( i ) & " Y:" & integer'image(
u ) severity error;
    assert( Z_test = Zero ) report "-1 Z failed: X:" & integer'image( i ) & " Y:" & integer'image( u )
severity error;

    F      <= "110";
    wait for clock_period;

    assert( nicle = S ) report "0110 failed: X:" & integer'image( i ) & " Y:" & integer'image( u )
severity error;

    F      <= "111";
    wait for clock_period;

    assert( nicle = S ) report "0111 failed: X:" & integer'image( i ) & " Y:" & integer'image( u )
severity error;

    M      <= '1';
    F      <= "000";
    wait for clock_period;

    assert( XandY = S ) report "X AND Y failed: X:" & integer'image( i ) & " Y:" & integer'image( u )
severity error;

    F      <= "001";
    wait for clock_period;

    assert( XnandY = S ) report "X nand Y failed: X:" & integer'image( i ) & " Y:" & integer'image( u
) severity error;

    F      <= "010";
    wait for clock_period;

    assert( XorY = S ) report "X or Y failed: X:" & integer'image( i ) & " Y:" & integer'image( u )
severity error;

    F      <= "011";
    wait for clock_period;

```

```

severity error;

    assert( XnorY = S ) report "X nor Y failed: X:" & integer'image( i ) & " Y:" & integer'image( u )

F      <= "100";
wait for clock_period;

severity error;

    assert( XxorY = S ) report "X xor Y failed: X:" & integer'image( i ) & " Y:" & integer'image( u )

F      <= "101";
wait for clock_period;

severity error;

    assert( XxnorY = S ) report "X xnor Y failed: X:" & integer'image( i ) & " Y:" & integer'image( u )

F      <= "110";
wait for clock_period;

severity error;

    assert( X = S ) report "X failed: X:" & integer'image( i ) & " Y:" & integer'image( u ) severity
u ) severity error;
    assert( N_test = Negative ) report "X N failed: X:" & integer'image( i ) & " Y:" & integer'image(
severity error;
    assert( Z_test = Zero ) report "X Z failed: X:" & integer'image( i ) & " Y:" & integer'image( u )

F      <= "111";
wait for clock_period;

severity error;

    assert( Y = S ) report "Y failed: X:" & integer'image( i ) & " Y:" & integer'image( u ) severity
u ) severity error;
    assert( N_test = Negative ) report "Y N failed: X:" & integer'image( i ) & " Y:" & integer'image(
severity error;
    assert( Z_test = Zero ) report "Y Z failed: X:" & integer'image( i ) & " Y:" & integer'image( u )

    end loop;
end loop;

wait;
end process;

```

```

test_proc: process ( Xi, Yi, F )
variable St, Xt, Yt : std_logic_vector( n-1 downto 0 );
variable Restmp : integer;
begin

Xt := std_logic_vector( to_signed( Xi, n ) );
Yt := std_logic_vector( to_signed( Yi, n ) );

    if F = "101" then
        St := not nicle;
        OF_test      <= '0';
    else
        if F = "000" then
            Restmp := Xi + Yi;
        elsif F = "001" then
            Restmp := Xi - Yi;
        elsif F = "010" then
            Restmp := Xi + 1;
        elsif F = "011" then
            Restmp := Xi - 1;
        elsif F = "100" then
            Restmp := Xi + Xi;
        elsif F = "110" then
            Restmp := Xi;
        elsif F = "111" then
            Restmp := Yi;
        else
            Restmp := 0;
        end if;

        if Restmp > 2**( n-1 ) - 1 or Restmp < -( 2**( n-1 ) ) then
            OF_test      <= '1';
        else
            OF_test      <= '0';
        end if;

        St := std_logic_vector( to_unsigned( Restmp mod 2**n, n ) );
    end if;

S_test <= St;

```

```
N_test <= St( n-1 );

if St = nicle then
    Z_test <= '1';
else
    Z_test <= '0';
end if;

XandY <= Xt and Yt;
XnandY <= Xt nand Yt;
XorY <= Xt or Yt;
XnorY <= Xt nor Yt;
XxorY <= Xt xor Yt;
XxnorY <= Xt xnor Yt;

end process;

END;
```

```

-- *****
-- **** STUDENT: 64210445
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Upoštevati morate dejansko zakasnitev ALU (glej vrednost combinatorial path delay), ne wait for 1 ns!
Povezava med alu_cla in testbench mora biti parametrizirana (manjka generic map (n=>n))
-- *****
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY alu_cla_tb IS
    GENERIC( n: natural := 8 );
END alu_cla_tb;

ARCHITECTURE behavior OF alu_cla_tb IS

    -- Component Declaration for the Unit Under Test ( UUT )

    COMPONENT alu_cla
    PORT(
        M : IN std_logic;
        F : IN std_logic_vector( 2 downto 0 );
        X : IN std_logic_vector( 7 downto 0 );
        Y : IN std_logic_vector( 7 downto 0 );
        S : OUT std_logic_vector( 7 downto 0 );
        Negative : OUT std_logic;
        Cout : OUT std_logic;
        Overflow : OUT std_logic;
        Zero : OUT std_logic;
        Gout : OUT std_logic;
        Pout : OUT std_logic
    );
    END COMPONENT;

    -- Inputs
    signal M : std_logic := '0';
    signal F : std_logic_vector( 2 downto 0 ) := ( others => '0' );
    signal X : std_logic_vector( 7 downto 0 ) := ( others => '0' );
    signal Y : std_logic_vector( 7 downto 0 ) := ( others => '0' );

```

```

-- Outputs
signal      S : std_logic_vector( 7 downto 0 );
signal      Negative : std_logic;
signal      Cout : std_logic;
signal      Overflow : std_logic;
signal      Zero : std_logic;
signal      Gout : std_logic;
signal      Pout : std_logic;
-- No clocks detected in port list. Replace <clock> below with
-- appropriate port name

constant    period : time := 1 ns;

signal      Xval, Yval : integer := 0;
signal      S_test, XandY,          XnandY, XorY, XnorY,          XxorY, XxnorY : std_logic_vector( n-1
downto 0 ) := ( others => '0' );
signal      N_test, OF_test, Z_test : std_logic := '0';

constant    zeros : std_logic_vector( n-1 downto 0 ) := ( others => '0' );

BEGIN

-- Instantiate the Unit Under Test ( UUT )
 uut: alu_cla PORT MAP (
M => M,
F => F,
X => X,
Y => Y,
S => S,
Negative => Negative,
Cout => Cout,
Overflow => Overflow,
Zero => Zero,
Gout => Gout,
Pout => Pout
);

-- Stimulus process
stim_proc: process

```


begin

```
    for i in -( 2**( n-1 ) ) to 2**( n-1 ) - 1 loop
        Xval  <= i;
        X      <= std_logic_vector( to_signed( i, n ) );
        for j in -( 2**( n-1 ) ) to 2**( n-1 ) - 1 loop
            Yval  <= j;
            Y      <= std_logic_vector( to_signed( j, n ) );

            M      <= '0';
            F      <= "000";
            wait for period;

            assert( S_test = S ) report "SUM failed at: X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
            assert( N_test = Negative ) report "SUM N failed at: X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;
            assert( OF_test = Overflow ) report "SUM V failed at: X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;
            assert( Z_test = Zero ) report "SUM Z failed at: X:" & integer'image( i ) & " Y:" & integer'image(
j ) severity error;

            F      <= "001";
            wait for period;

            assert( S_test = S ) report "DIF failed at: X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
            assert( N_test = Negative ) report "DIF N failed at: X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;
            assert( OF_test = Overflow ) report "DIF V failed at: X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;
            assert( Z_test = Zero ) report "DIF Z failed at: X:" & integer'image( i ) & " Y:" & integer'image(
j ) severity error;

            F      <= "010";
            wait for period;

            assert( S_test = S ) report "X+1 failed at: X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
```

```

        assert( N_test = Negative ) report "X+1 N failed at: X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;
        assert( OF_test = Overflow ) report "X+1 V failed at: X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;
        assert( Z_test = Zero ) report "X+1 Z failed at: X:" & integer'image( i ) & " Y:" & integer'image(
j ) severity error;

        F      <= "011";
        wait for period;

        assert( S_test = S ) report "X-1 failed at: X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
        assert( N_test = Negative ) report "X-1 N failed at: X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;
        assert( OF_test = Overflow ) report "X-1 V failed at: X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;
        assert( Z_test = Zero ) report "X-1 Z failed at: X:" & integer'image( i ) & " Y:" & integer'image(
j ) severity error;

        F      <= "100";
        wait for period;

        assert( S_test = S ) report "X+X failed at: X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
        assert( N_test = Negative ) report "X+X N failed at: X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;
        assert( OF_test = Overflow ) report "X+X V failed at: X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;
        assert( Z_test = Zero ) report "X+X Z failed at: X:" & integer'image( i ) & " Y:" & integer'image(
j ) severity error;

        F      <= "101";
        wait for period;

        assert( S_test = S ) report "-1 failed at: X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
        assert( N_test = Negative ) report "-1 N failed at: X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;
        assert( Z_test = Zero ) report "-1 Z failed at: X:" & integer'image( i ) & " Y:" & integer'image(
j ) severity error;

```

```

F      <= "110";
wait for period;

severity error;

assert( zeros = S ) report "0110 failed at: X:" & integer'image( i ) & " Y:" & integer'image( j )

F      <= "111";
wait for period;

severity error;

assert( zeros = S ) report "0111 failed at: X:" & integer'image( i ) & " Y:" & integer'image( j )

M      <= '1';
F      <= "000";
wait for period;

assert( XandY = S ) report "X AND Y failed at: X:" & integer'image( i ) & " Y:" & integer'image( j
) severity error;

F      <= "001";
wait for period;

assert( XnandY = S ) report "X nand Y failed at: X:" & integer'image( i ) & " Y:" & integer'image(
j ) severity error;

F      <= "010";
wait for period;

severity error;

assert( XorY = S ) report "X or Y failed at: X:" & integer'image( i ) & " Y:" & integer'image( j )

F      <= "011";
wait for period;

assert( XnorY = S ) report "X nor Y failed at: X:" & integer'image( i ) & " Y:" & integer'image( j
) severity error;

F      <= "100";
wait for period;

```

```

        assert( XxorY = S ) report "X xor Y failed at: X:" & integer'image( i ) & " Y:" & integer'image( j
) severity error;

        F      <= "101";
        wait for period;

        assert( XxnorY = S ) report "X xnor Y failed at: X:" & integer'image( i ) & " Y:" & integer'image(
j ) severity error;

        F      <= "110";
        wait for period;

        assert( X = S ) report "X failed at: X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
        assert( N_test = Negative ) report "X N failed at: X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;
        assert( Z_test = Zero ) report "X Z failed at: X:" & integer'image( i ) & " Y:" & integer'image( j
) severity error;

        F      <= "111";
        wait for period;

        assert( Y = S ) report "Y failed at: X:" & integer'image( i ) & " Y:" & integer'image( j )
severity error;
        assert( N_test = Negative ) report "Y N failed at: X:" & integer'image( i ) & " Y:" &
integer'image( j ) severity error;
        assert( Z_test = Zero ) report "Y Z failed at: X:" & integer'image( i ) & " Y:" & integer'image( j
) severity error;
        end loop;
    end loop;

wait;
end process;

test_proc: process ( Xval, Yval, F )
variable Stmp, Xtmp, Ytmp : std_logic_vector( n-1 downto 0 );
variable Restmp : integer;
begin

```

```

Xtmp := std_logic_vector( to_signed( Xval, n ) );
Ytmp := std_logic_vector( to_signed( Yval, n ) );

if F = "101" then
    Stmp := not zeros;
    OF_test    <= '0';
else
    if F = "000" then
        Restmp := Xval + Yval;
    elsif F = "001" then
        Restmp := Xval - Yval;
    elsif F = "010" then
        Restmp := Xval + 1;
    elsif F = "011" then
        Restmp := Xval - 1;
    elsif F = "100" then
        Restmp := Xval + Xval;
    elsif F = "110" then
        Restmp := Xval;
    elsif F = "111" then
        Restmp := Yval;
    else
        Restmp := 0;
    end if;

    if Restmp > 2**( n-1 ) - 1 or Restmp < -( 2**( n-1 ) ) then
        OF_test    <= '1';
    else
        OF_test    <= '0';
    end if;

    Stmp := std_logic_vector( to_unsigned( Restmp mod 2**n, n ) );
end if;

S_test <= Stmp;
N_test <= Stmp( n-1 );

if Stmp = zeros then
    Z_test <= '1';
else

```

```
        Z_test <= '0';  
    end if;
```

```
    XandY  <= Xtmp and Ytmp;  
    XnandY <= Xtmp nand Ytmp;  
    XorY   <= Xtmp or Ytmp;  
    XnorY  <= Xtmp nor Ytmp;  
    XxorY  <= Xtmp xor Ytmp;  
    XxnorY <= Xtmp xnor Ytmp;
```

```
end process;
```

```
END;
```

```

-- *****
-- **** STUDENT: 64210455
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Upoštevati morate dejansko zakasnitev ALU (glej vrednost combinatorial path delay), ne wait for 1 ns!
-- *****

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY alu_cla_tb IS
    GENERIC( n: natural := 8 );
END alu_cla_tb;

ARCHITECTURE behavior OF alu_cla_tb IS
    COMPONENT alu_cla
        GENERIC( n: natural := 8 );
        PORT(
            M : IN std_logic;
            F : IN std_logic_vector( 2 downto 0 );
            X : IN std_logic_vector( n-1 downto 0 );
            Y : IN std_logic_vector( n-1 downto 0 );
            S : OUT std_logic_vector( n-1 downto 0 );
            Negative : OUT std_logic;
            Cout : OUT std_logic;
            Overflow : OUT std_logic;
            Zero : OUT std_logic;
            Gout : OUT std_logic;
            Pout : OUT std_logic
        );
    END COMPONENT;

    -- Vhodi
    signal M : std_logic := '0';
    signal F : std_logic_vector( 2 downto 0 ) := ( others => '0' );
    signal X : std_logic_vector( n-1 downto 0 ) := ( others => '0' );
    signal Y : std_logic_vector( n-1 downto 0 ) := ( others => '0' );

    -- Izhodi
    signal S : std_logic_vector( n-1 downto 0 );

```

```

signal      Negative : std_logic;
signal      Cout     : std_logic;
signal      Overflow  : std_logic;
signal      Zero      : std_logic;
signal      Gout      : std_logic;
signal      Pout      : std_logic;

-- Vmesni signali
signal      X_int, Y_int : integer := 0;
signal      S_expected : std_logic_vector( n-1 downto 0 ) := ( others => '0' );
signal      Neg_expected, Overflow_expected, Zero_expected : std_logic := '0';

constant    zeros : std_logic_vector( n-1 downto 0 ) := ( others => '0' );

```

BEGIN

```

uut: alu_cla
GENERIC MAP ( n => n )
PORT MAP (
M => M,
F => F,
X => X,
Y => Y,
S => S,
Negative => Negative,
Cout => Cout,
Overflow => Overflow,
Zero => Zero,
Gout => Gout,
Pout => Pout
);

stim_proc: process
begin
for i in -( 2**( n-1 ) ) to 2**( n-1 ) - 1 loop
X_int <= i;
X      <= std_logic_vector( to_signed( i, n ) );
for j in -( 2**( n-1 ) ) to 2**( n-1 ) - 1 loop
Y_int <= j;
Y      <= std_logic_vector( to_signed( j, n ) );

```



```

-- Aritmetične operacije
M    <= '0';

F    <= "000";    -- X + Y
wait for 1 ns;
assert( S_expected = S ) report "Error in ADD. X: " & integer'image( i ) & ", Y: " & integer'image( j ) severity
error;

F    <= "001";    -- X - Y
wait for 1 ns;
assert( S_expected = S ) report "Error in SUB. X: " & integer'image( i ) & ", Y: " & integer'image( j ) severity
error;

F    <= "010";    -- X + 1
wait for 1 ns;
assert( S_expected = S ) report "Error in INC. X: " & integer'image( i ) severity error;

F    <= "011";    -- X - 1
wait for 1 ns;
assert( S_expected = S ) report "Error in DEC. X: " & integer'image( i ) severity error;

F    <= "100";    -- X + X
wait for 1 ns;
assert( S_expected = S ) report "Error in DOUBLE. X: " & integer'image( i ) severity error;

F    <= "101";    -- Constant -1
wait for 1 ns;
assert( S = not zeros ) report "Error in CONST. X: " & integer'image( i ) severity error;

F    <= "110";    -- Undefined
wait for 1 ns;
assert( S = zeros ) report "Error in UNDEF1. X: " & integer'image( i ) & ", Y: " & integer'image( j ) severity error;

F    <= "111";    -- Undefined
wait for 1 ns;
assert( S = zeros ) report "Error in UNDEF2. X: " & integer'image( i ) & ", Y: " & integer'image( j ) severity error;

-- Logicne operacije
M    <= '1';

```

```

F      <= "000";    -- X and Y
wait for 1 ns;
assert( S = ( X and Y ) ) report "Error in AND. X: " & integer'image( i ) & ", Y: " & integer'image( j ) severity
error;

```

```

F      <= "001";    -- X nand Y
wait for 1 ns;
assert( S = ( X nand Y ) ) report "Error in NAND. X: " & integer'image( i ) & ", Y: " & integer'image( j ) severity
error;

```

```

F      <= "010";    -- X or Y
wait for 1 ns;
assert( S = ( X or Y ) ) report "Error in OR. X: " & integer'image( i ) & ", Y: " & integer'image( j ) severity
error;

```

```

F      <= "011";    -- X nor Y
wait for 1 ns;
assert( S = ( X nor Y ) ) report "Error in NOR. X: " & integer'image( i ) & ", Y: " & integer'image( j ) severity
error;

```

```

F      <= "100";    -- X xor Y
wait for 1 ns;
assert( S = ( X xor Y ) ) report "Error in XOR. X: " & integer'image( i ) & ", Y: " & integer'image( j ) severity
error;

```

```

F      <= "101";    -- X xnor Y
wait for 1 ns;
assert( S = ( X xnor Y ) ) report "Error in XNOR. X: " & integer'image( i ) & ", Y: " & integer'image( j ) severity
error;

```

```

F      <= "110";    -- X
wait for 1 ns;
assert( S = X ) report "Error in PASS X. X: " & integer'image( i ) severity error;

```

```

F      <= "111";    -- Y
wait for 1 ns;
assert( S = Y ) report "Error in PASS Y. Y: " & integer'image( j ) severity error;
end loop;
end loop;

```

```
wait;  
end process;  
  
END behavior;
```

```

-- *****
-- **** STUDENT: 64210457
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Upoštevati morate dejansko zakasnitev ALU (glej vrednost combinatorial path delay), ne wait for 1 ns!
Povezava med alu_cla in testbench mora biti parametrizirana (manjka generic map (n=>n))
-- *****
-- test of all alu operations
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY alu_cla_tb IS
GENERIC( n: natural := 8 );
END alu_cla_tb;

ARCHITECTURE behavior OF alu_cla_tb IS

    -- Component Declaration for the Unit Under Test ( UUT )

    COMPONENT alu_cla
    PORT(
        M : IN std_logic;
        F : IN std_logic_vector( 2 downto 0 );
        X : IN std_logic_vector( n-1 downto 0 );
        Y : IN std_logic_vector( n-1 downto 0 );
        S : OUT std_logic_vector( n-1 downto 0 );
        Negative : OUT std_logic;
        Cout : OUT std_logic;
        Overflow : OUT std_logic;
        Zero : OUT std_logic;
        Gout : OUT std_logic;
        Pout : OUT std_logic
    );
    END COMPONENT;

    -- Inputs
    signal M : std_logic := '0';
    signal F : std_logic_vector( 2 downto 0 ) := ( others => '0' );
    signal X : std_logic_vector( n-1 downto 0 ) := ( others => '0' );

```

```

signal          Y : std_logic_vector( n-1 downto 0 ) := ( others => '0' );

-- Outputs
signal          S : std_logic_vector( n-1 downto 0 );
signal          Negative : std_logic;
signal          Cout : std_logic;
signal          Overflow : std_logic;
signal          Zero : std_logic;
signal          Gout : std_logic;
signal          Pout : std_logic;

-- test signals
signal          X_n, Y_n : integer := 0;
signal          S_check, X_nand_Y, X_or_Y,          X_nor_Y, X_and_Y,          X_xor_Y, X_xnor_Y : std_logic_vector(
n-1 downto 0 ) := ( others => '0' );
signal          Neg_check, Over_check, Zero_check : std_logic := '0';

constant        zeros : std_logic_vector( n-1 downto 0 ) := ( others => '0' );

-- Procedure for assertions / for evaluating the alu
procedure run_assertions(
expected_S : in std_logic_vector( n-1 downto 0 );
expected_Neg : in std_logic;
expected_Over : in std_logic;
expected_Zero : in std_logic;
actual_S : in std_logic_vector( n-1 downto 0 );
actual_Neg : in std_logic;
actual_Over : in std_logic;
actual_Zero : in std_logic;
error_message : string
) is
begin
-- wait for 1 ns;
assert ( actual_S = expected_S ) report error_message & " - S mismatch" severity error;
    if ( expected_Neg /= '0' and expected_Over /= '0' and expected_Zero /= '0' ) then
        assert ( actual_Neg = expected_Neg ) report error_message & " - Negative mismatch" severity
error;
        assert ( actual_Over = expected_Over ) report error_message & " - Overflow mismatch" severity
error;
        assert ( actual_Zero = expected_Zero ) report error_message & " - Zero mismatch" severity error;

```

```

        end if;
    end procedure;

BEGIN

    -- Instantiate the Unit Under Test ( UUT )
    uut: alu_cla PORT MAP (
        M => M,
        F => F,
        X => X,
        Y => Y,
        S => S,
        Negative => Negative,
        Cout => Cout,
        Overflow => Overflow,
        Zero => Zero,
        Gout => Gout,
        Pout => Pout
    );

    -- Stimulus process
    stim_proc: process
    begin
        -- Loop through all values of X and Y in the 2's complement range
        for i in -( 2**( n-1 ) ) to 2**( n-1 ) - 1 loop
            X_n <= i;
            X    <= std_logic_vector( to_signed( i, n ) );
            for j in -( 2**( n-1 ) ) to 2**( n-1 ) - 1 loop
                Y_n <= j;
                Y    <= std_logic_vector( to_signed( j, n ) );
                wait for 1 ns;
                M    <= '0';      -- AritmetiÄni naÄin
                F    <= "000";

                -- X + Y
                wait for 1 ns;
                run_assertions( S_check, Neg_check, Over_check, Zero_check, S, Negative, Overflow, Zero, "Y failed. X:" &
integer'image( i ) & " Y:" & integer'image( j ) );
                F    <= "001";

                -- X - Y
                wait for 1 ns;
            end loop
        end loop
    end
end process;

```

```

run_assertions( S_check, Neg_check, Over_check, Zero_check, S, Negative, Overflow, Zero, "X - Y failed" );
    F    <= "010";
    -- X + 1
        wait for 1 ns;
run_assertions( S_check, Neg_check, Over_check, Zero_check, S, Negative, Overflow, Zero, "X + 1 failed" );
    F    <= "011";
    -- X - 1
        wait for 1 ns;
run_assertions( S_check, Neg_check, Over_check, Zero_check, S, Negative, Overflow, Zero, "X - 1 failed" );
    F    <= "100";
    -- X + X
        wait for 1 ns;
run_assertions( S_check, Neg_check, Over_check, Zero_check, S, Negative, Overflow, Zero, "X + X failed" );
    F    <= "101";
    -- -1
        wait for 1 ns;
run_assertions( S_check, Neg_check, Over_check, Zero_check, S, Negative, Overflow, Zero, "-1 failed" );
--
    F    <= "110";
    -- Undefined operations
        wait for 1 ns;
run_assertions( zeros, '0', '0', '0', S, Negative, Overflow, Zero, "Undefined 110 failed" );
    F    <= "111";
        wait for 1 ns;
run_assertions( zeros, '0', '0', '0', S, Negative, Overflow, Zero, "Undefined 111 failed" );

    M    <= '1';        -- LogiÄni naÄin
    F    <= "000";        -- X and Y
        wait for 1 ns;
run_assertions( X_and_Y, '0', '0', '0', S, Negative, Overflow, Zero, "X AND Y failed" );
F    <= "001";-- X NAND Y
        wait for 1 ns;
run_assertions( X_nand_Y, '0', '0', '0', S, Negative, Overflow, Zero, "X NAND Y failed" );
F    <= "010";-- X OR Y
        wait for 1 ns;
run_assertions( X_or_Y, '0', '0', '0', S, Negative, Overflow, Zero, "X OR Y failed" );
F    <= "011";-- X NOR Y
        wait for 1 ns;
run_assertions( X_nor_Y, '0', '0', '0', S, Negative, Overflow, Zero, "X NOR Y failed" );
F    <= "100";-- X XOR Y

```

```

        wait for 1 ns;
run_assertions( X_xor_Y, '0', '0', '0', S, Negative, Overflow, Zero, "X XOR Y failed" );
F    <= "101";-- X XNOR Y
        wait for 1 ns;
        run_assertions( X_xnor_Y, '0', '0', '0', S, Negative, Overflow, Zero, "X XNOR Y failed" );

        F    <= "110";-- X
        wait for 1 ns;
run_assertions( X, Neg_check, Over_check, Zero_check, S, Negative, Overflow, Zero, "X pass-through failed" );
        F    <= "111";-- Y
        wait for 1 ns;
run_assertions( Y, Neg_check, Over_check, Zero_check, S, Negative, Overflow, Zero, "Y pass-through failed" );

        end loop;
end loop;
wait;
end process;

check_proc: process ( X_n, Y_n, F )    -- actual values
variable S_temp, X_temp, Y_temp : std_logic_vector( n-1 downto 0 );
variable Res_temp : integer;
begin
    X_temp := std_logic_vector( to_signed( X_n, n ) );
    Y_temp := std_logic_vector( to_signed( Y_n, n ) );

    -- calculating results to check
    if F = "101" then
        S_temp := not zeros;
        Over_check    <= '0';
    else if F = "000" then
        Res_temp := X_n + Y_n;
    elsif F = "001" then
        Res_temp := X_n - Y_n;
    elsif F = "010" then
        Res_temp := X_n + 1;
    elsif F = "011" then
        Res_temp := X_n - 1;
    elsif F = "100" then
        Res_temp := X_n + X_n;
    elsif F = "110" then

```



```

        Res_temp := X_n;
    elsif F = "111" then
        Res_temp := Y_n;
    else
        Res_temp := 0;
    end if;

-- check weather result is in range or there is overflow
    if Res_temp > 2**( n-1 ) - 1 or Res_temp < -( 2**( n-1 ) ) then
        Over_check    <= '1';
    else
        Over_check    <= '0';
    end if;

-- write result to binary to s_comp
        S_temp := std_logic_vector( to_unsigned( Res_temp mod 2**n, n ) );
    end if;

-- output the results to our signals
        S_check    <= S_temp;
        Neg_check  <= S_temp( n-1 );

-- check if result is zero to set zero bit
    if S_temp = zeros then
        Zero_check    <= '1';
    else
        Zero_check    <= '0';
    end if;

-- Logical operations connectin to signals for acctual values
        X_and_Y      <= X_temp and Y_temp;
        X_nand_Y     <= X_temp nand Y_temp;
        X_or_Y <= X_temp or Y_temp;
        X_nor_Y      <= X_temp nor Y_temp;
        X_xor_Y      <= X_temp xor Y_temp;
        X_xnor_Y     <= X_temp xnor Y_temp;

end process;

END;
```

```

-- *****
-- **** STUDENT: 64240430
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Upoštevati morate dejansko zakasnitev ALU (glej vrednost combinatorial path delay), ne wait for 100
ps!
Povezava med alu_cla in testbench mora biti parametrizirana (manjka generic map (n=>n))
-- *****
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity alu_cla_tb is
end alu_cla_tb;

architecture dn2 of alu_cla_tb is

constant      n : natural := 8;

-- Component Declaration for the Unit Under Test ( UUT )

COMPONENT alu_cla
PORT(
M : IN std_logic;
F : IN std_logic_vector( 2 downto 0 );
X : IN std_logic_vector( n-1 downto 0 );
Y : IN std_logic_vector( n-1 downto 0 );
S : OUT std_logic_vector( n-1 downto 0 );
Negative : OUT std_logic;
Cout : OUT std_logic;
Overflow : OUT std_logic;
Zero : OUT std_logic;
Gout : OUT std_logic;
Pout : OUT std_logic
);
END COMPONENT;

-- Inputs
signal      M : std_logic := '0';
signal      F : std_logic_vector( 2 downto 0 ) := ( others => '0' );

```

```

signal      X : std_logic_vector( n-1 downto 0 ) := ( others => '0' );
signal      Y : std_logic_vector( n-1 downto 0 ) := ( others => '0' );

-- Outputs
signal      S : std_logic_vector( n-1 downto 0 );
signal      Negative : std_logic;
signal      Cout : std_logic;
signal      Overflow : std_logic;
signal      Zero : std_logic;
signal      Gout : std_logic;
signal      Pout : std_logic;

      signal      clock : std_logic;
constant    clock_period : time := 100 ps;

      signal      helper_i, helper_j : integer range -2** ( n-1 ) to 2** ( n-1 )-1;
      signal      rezultat : integer range -2**n to 2**n-1;

      signal      neg, z, v : std_logic;
      signal      izhod : std_logic_vector( n-1 downto 0 );

begin

uut: alu_cla
  port map(
    M => M,      F => F,      x => x,      y => y,      s => s,      negative => negative,      cout => cout,
    overflow => overflow,      zero => zero,      gout => gout,      pout => pout
  );

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

stimulus_process: process
begin
  z1: for a in 0 to 1 loop

```

```

    for b in 0 to 7 loop
for i in -2**( n-1 ) to 2**( n-1 )-1 loop
for j in -2**( n-1 ) to 2**( n-1 )-1 loop
    F    <= std_logic_vector( to_unsigned( b, 3 ) );
x      <= std_logic_vector( to_signed( i, n ) );
y      <= std_logic_vector( to_signed( j, n ) );
helper_i    <= i;
helper_j    <= j;
wait for clock_period;
assert( s = izhod )
    report "Test failed for values: X = " &integer'image( a )&
        ", Y = " & integer'image( b ) &
        ", M = " & integer'image( i ) &
        ", F = " & integer'image( j ) & "."
severity error;

exit z1 when ( s /= izhod );

IF M = '0' then
assert( negative = neg )
    report "Negativity test failed for values: X = " &integer'image( a )&
        ", Y = " & integer'image( b ) &
        ", M = " & integer'image( i ) &
        ", F = " & integer'image( j ) & "."
severity error;

assert( zero = z )
    report "Zero test failed for values: X = " &integer'image( a )&
        ", Y = " & integer'image( b ) &
        ", M = " & integer'image( i ) &
        ", F = " & integer'image( j ) & "."
severity error;

assert( overflow = v )
    report "Overflow test failed for values: X = " &integer'image( a )&
        ", Y = " & integer'image( b ) &
        ", M = " & integer'image( i ) &
        ", F = " & integer'image( j ) & "."
severity error;
exit z1 when ( negative /= neg or zero /= z or overflow /= v );
end if;

```

```

        end loop;
    end loop;
    end loop;
    M      <= not M;
end loop;
wait;
end process;

```

```

posodobitev_process: process ( helper_i, helper_j, izhod )

```

```

    variable rezultat : integer range -2**n to 2**n-1;
    variable x_temp, y_temp, izhod_temp : std_logic_vector( n-1 downto 0 );
begin
    if M = '0' then
        case F is
            when "000" => rezultat := helper_i + helper_j;
            when "001" => rezultat := helper_i - helper_j;
            when "010" => rezultat := helper_i + 1;
            when "011" => rezultat := helper_i - 1;
            when "100" => rezultat := helper_i + helper_i;
            when "101" => rezultat := -1;
            when others => rezultat := 0;
        end case;
        if ( rezultat >= -2**( n-1 ) and rezultat < 0 )
        or ( rezultat >= 2**( n-1 ) and rezultat < 2**n ) then
            neg    <= '1';
        else
            neg    <= '0';
        end if;
        if rezultat mod 2**n = 0 then
            z      <= '1';
        else
            z      <= '0';
        end if;
        if rezultat < -2**( n-1 ) or rezultat >= 2**( n-1 ) then
            v      <= '1';
        else
            v      <= '0';
        end if;
        izhod_temp := std_logic_vector( to_unsigned( rezultat mod 2**n, n ) );
    end if;
end process;

```

```

else
    x_temp := std_logic_vector( to_signed( helper_i, n ) );
    y_temp := std_logic_vector( to_signed( helper_j, n ) );
    case F is
        when "000" => izhod_temp := x_temp and y_temp;
        when "001" => izhod_temp := x_temp nand y_temp;
        when "010" => izhod_temp := x_temp or y_temp;
        when "011" => izhod_temp := x_temp nor y_temp;
        when "100" => izhod_temp := x_temp xor y_temp;
        when "101" => izhod_temp := x_temp xnor y_temp;
        when "110" => izhod_temp := x_temp;
        when "111" => izhod_temp := y_temp;
        when others => report "ne bo.";
    end case;
end if;
izhod <= izhod_temp;
end process;
end;

```

```

-- *****
-- **** STUDENT: 64210132
-- *****
-- KOMENTARJI K OCENI NALOGE
Matej Možek: Povezava med alu_cla in testbench mora biti parametrizirana (manjka generic map (n=>n)).
Ideja datoteke testnih vrednosti je, da preleti celoten obseg števil x in y ter na drugačen način (torej z VHDL +
operatorjem) preveri, če so razlike v izračunu strojne komponente in simulatorja. Koda datoteke testnih vrednosti je
ista kot pri testiranju CLA seštevalnika, kar ni poanta naloge - testirati je treba operaciji seštevanja in odštevanja
preko celotnega obsega operandov X in Y.
***** NASLEDNJIČ KODO NALOŽITE V USTREZEN RAZDELEK (OSTALO_x), KJER JE X ŠTEVILKA DOMAČE NALOGE *****
-- *****
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY alu_tb IS
generic( n: natural := 8 );
END alu_tb;

ARCHITECTURE sim OF alu_tb IS

    component alu_cla
        generic( n: natural := 8 );
        port( M          : in      std_logic;    --naèin delovanja ('0' => aritmetièni, '1' => logièni)

              F          : in      std_logic_vector(2 downto 0);    -- funkcijski vhod za operacije
              X, Y      : in      std_logic_vector(n-1 downto 0);
              S          : out     std_logic_vector(n-1 downto 0);
              Negative, Cout, Overflow, Zero, Gout, Pout : out     std_logic );
    end component;

    signal M: std_logic := '0';
    signal F: std_logic_vector(2 downto 0) := (others => '0');
    signal X, Y: std_logic_vector(n-1 downto 0) := (others => '0');
    signal S: std_logic_vector(n-1 downto 0) := (others => '0');
    signal Negative, Cout, Overflow, Zero, Gout, Pout: std_logic := '0';

BEGIN

    uut: alu_cla port map (

```

```
F => F,  
M => M,  
X => X,  
Y => Y,  
S => S,  
Cout => Cout,  
Gout => Gout,  
Pout => Pout,  
Negative => Negative,  
Overflow => Overflow,  
Zero => Zero  
);
```

```
stim_proc: process  
begin
```

```
X <= x"AA";  
Y <= x"55";  
M <= '1';  
F <= "001";  
wait for 100ns;  
F <= "000";  
wait for 100ns;  
F <= "001";  
wait for 100ns;  
F <= "010";  
wait for 100ns;  
F <= "011";  
wait for 100ns;  
F <= "100";  
wait for 100ns;  
F <= "101";  
wait for 100ns;  
F <= "110";  
wait for 100ns;  
F <= "111";  
Y <= x"AA";  
wait for 100ns;  
M <= '0';  
F <= "000";
```



```
X <= x"0E";
Y <= x"F2";

wait for 100ns;
X <= x"0E";
Y <= x"F1";

wait for 100ns;
X <= x"0E";
Y <= x"7F";

wait for 100ns;
X <= x"F2";
Y <= x"7F";

wait for 100ns;
F(0) <= '1';
X <= x"0E";
Y <= x"0E";

wait for 100ns;
X <= x"0E";
Y <= x"0F";

wait for 100ns;
X <= x"0E";
Y <= x"81";

wait for 100ns;
X <= x"F2";
Y <= x"7F";

wait for 100ns;
X <= x"7F";
Y <= x"7F";
F <= "010";
```

```
wait for 100ns;  
X <= x"7F";  
Y <= x"7F";  
F <= "010";
```

```
wait for 100ns;  
X <= x"80";  
Y <= x"7F";  
F <= "011";
```

```
wait for 100ns;  
X <= x"7F";  
Y <= x"7F";  
F <= "100";
```

```
wait for 100ns;  
X <= x"80";  
Y <= x"7F";  
F <= "100";
```

```
wait for 100ns;  
X <= x"80";  
Y <= x"7F";  
F <= "110";
```

```
wait for 100ns;  
X <= x"80";  
Y <= x"7F";  
F <= "101";
```

```
wait;  
end process;
```

```
END sim;
```