

-- **** STUDENT: 64000225.....	3
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	3
-- **** STUDENT: 64190088.....	6
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	6
-- **** STUDENT: 64200100.....	9
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	9
-- **** STUDENT: 64200112.....	12
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	12
-- **** STUDENT: 64200163.....	15
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	15
-- **** STUDENT: 64200238.....	18
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	18
-- **** STUDENT: 64200288.....	22
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	22
-- **** STUDENT: 64200296.....	25
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	25
-- **** STUDENT: 64200385.....	29
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	29
-- **** STUDENT: 64210113.....	32
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	32
-- **** STUDENT: 64210290.....	35
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	35
-- **** STUDENT: 64210382.....	39
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	39
-- **** STUDENT: 64210384.....	42
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	42
-- **** STUDENT: 64210386.....	46
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	46
-- **** STUDENT: 64210445.....	49
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	49
-- **** STUDENT: 64210455.....	52
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	52
-- **** STUDENT: 64210457.....	56
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	56

-- **** STUDENT: 64240430.....	59
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	59
-- **** PREDLOGA VAJE	62
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	62

```

-- *****
-- **** STUDENT: 64000225
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
use ieee.math_real.all;

entity reg_file is
    generic(
        nr_regs          : natural := 4;
        reg_width        : natural := 8 );
    PORT ( clk,          -- clock input
        LE              : in std_logic;    -- Load enable input      ( active '1' )
        nRST            : in std_logic;    -- reset input              ( active '0' )
        dest_select,    -- register number destination select input
        A_select,       -- A, B bus destination select input
        B_select        : in  std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        D               : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus input
        A, B            : out std_logic_vector( reg_width - 1 downto 0 )    -- A, B bus output
    );
end reg_file;

architecture NDV of reg_file is

    COMPONENT dmuxnto1 IS
        generic(
            n_addr: natural := 2 );
        PORT (
            s : in  std_logic_vector( n_addr - 1 downto 0 );
            w : in  STD_LOGIC;
            f : OUT  std_logic_vector( 2*n_addr - 1 downto 0 )
        );
    END COMPONENT;

    COMPONENT muxnto1_bus IS
        generic(
            n_addr      : INTEGER := 2;
            bus_width    : INTEGER := 8 );
        PORT (
            s            : IN      std_logic_vector( n_addr - 1 downto 0 );

```

```

        w          :      IN          muxnto1_bus_type( 2**n_addr - 1 DOWNT0 0, bus_width - 1
DOWNT0 0 );
        f          :      OUT      std_logic_vector( bus_width - 1 downto 0 )
    );
END COMPONENT;

COMPONENT shift_reg is
    generic( reg_size: natural := 4 );
    PORT ( clk, nCLR, sr_in, sl_in : IN std_logic;
        s : in std_logic_vector( 1 downto 0 );
        x : in std_logic_vector( reg_size - 1 downto 0 );
        Q : out std_logic_vector( reg_size - 1 downto 0 )
    );
end COMPONENT;

constant    nr_regs_size : integer := integer( ceil( log2( real( nr_regs - 1 ) ) ) );
signal      load_enable_sig      : std_logic_vector( nr_regs - 1 DOWNT0 0 ) := ( others => '0' );
SIGNAL      reg_file              : muxnto1_bus_type( nr_regs - 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others
=> ( others => '0' ) );

type reg_file_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
signal      reg_file_array : reg_file_array_type := ( others => ( others => '0' ) );

type reg_mode_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( 1 downto 0 );
signal      reg_mode_sig      : reg_mode_array_type := ( others => ( others => '0' ) );

begin

    dmux: dmuxnto1
    generic map ( n_addr => nr_regs_size )
    port map(
        w => LE, s => dest_select, f => load_enable_sig
    );

    amux: muxnto1_bus
    generic map ( n_addr => nr_regs_size, bus_width => reg_width )
    port map (
        s => A_select, w => reg_file, f => A
    );

```

```

bmux: muxnto1_bus
generic map ( n_addr => nr_regs_size, bus_width => reg_width )
port map (
    s => B_select, w => reg_file, f => B
);

regs: for i in 0 to nr_regs - 1 generate
    reg: shift_reg
        generic map ( reg_size => reg_width )
        port map (
            s => reg_mode_sig( i ), x => D, Q => reg_file_array( i ),      clk => clk, nCLR => nRST, sr_in
=> '0', sl_in => '0'
        );
    end generate;

process ( reg_file_array ) is
    variable reg_file_col : std_logic_vector( reg_width - 1 downto 0 );
begin
    for idx in 0 to nr_regs-1 loop
        reg_file_col := reg_file_array( idx );
        for jdx in 0 to reg_width-1 loop
            reg_file( idx, jdx )      <= reg_file_col( jdx );
        end loop;
    end loop;
end process;

process ( load_enable_sig ) is
begin
    for i in 0 to nr_regs-1 loop
        reg_mode_sig( i )  <= load_enable_sig( i ) & load_enable_sig( i );
    end loop;
end process;

end NDV;

```

```

-- *****
-- **** STUDENT: 64190088
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
use ieee.math_real.all;

entity reg_file is
    generic(
        nr_regs          : natural := 4;
        reg_width        : natural := 8 );
    PORT ( clk,          -- clock input
        LE               : in std_logic;    -- Load enable input      ( active '1' )
        nRST             : in std_logic;    -- reset input                ( active '0' )
        dest_select,     -- register number destination select input
        A_select,        -- A, B bus destination select input
        B_select         : in  std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        D               : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus input
        A, B            : out std_logic_vector( reg_width - 1 downto 0 )    -- A, B bus output
    );
end reg_file;

architecture NDV of reg_file is

    component dmuxnto1 IS
        generic(
            n_addr: natural := 2 );
        PORT (
            s : in      std_logic_vector( n_addr - 1 downto 0 );
            w : in      STD_LOGIC;
            f : OUT     std_logic_vector( 2*n_addr - 1 downto 0 )
        );
    END component;

    component muxnto1_bus IS
        generic(
            n_addr      :      INTEGER := 2;
            bus_width   :      INTEGER := 8 );
        PORT (
            s           :      IN      std_logic_vector( n_addr - 1 downto 0 );

```

```

        w          :      IN          muxnto1_bus_type( 2**n_addr - 1 DOWNT0 0, bus_width - 1
DOWNT0 0 );
        f          :      OUT      std_logic_vector( bus_width - 1 downto 0 )
    );
END component;

component shift_reg is
    generic( reg_size: natural := 4 );
    PORT ( clk, nCLR, sr_in, sl_in : IN std_logic;
        s : in std_logic_vector( 1 downto 0 );
        x : in std_logic_vector( reg_size - 1 downto 0 );
        Q : out std_logic_vector( reg_size - 1 downto 0 )
    );
end component;

constant    nr_regs_size : integer := integer( ceil( log2( real( nr_regs - 1 ) ) ) );
signal      load_enable_sig      : std_logic_vector( nr_regs - 1 DOWNT0 0 ) := ( others => '0' );
SIGNAL      reg_file              : muxnto1_bus_type( nr_regs - 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others
=> ( others => '0' ) );

type reg_file_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
signal      reg_file_array : reg_file_array_type := ( others => ( others => '0' ) );

type reg_mode_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( 1 downto 0 );
signal      reg_mode_sig      : reg_mode_array_type := ( others => ( others => '0' ) );

begin

    u0: dmuxnto1
        generic map( n_addr => nr_regs_size )
        port map( w => LE, s => dest_select, f => load_enable_sig );

    u1: muxnto1_bus
        generic map( n_addr => nr_regs_size, bus_width => reg_width )
        port map( s => A_select, w => reg_file, f => A );

    u2: muxnto1_bus
        generic map( n_addr => nr_regs_size, bus_width => reg_width )
        port map( s => B_select, w => reg_file, f => B );

```

```

for_gen: for i in 0 to nr_regs - 1 generate
    u3: shift_reg
        generic map( reg_size => reg_width )
        port map
            ( clk => clk, nCLR => nRST, sr_in => '0', sl_in => '0', s => reg_mode_sig( i ), x => D, Q =>
reg_file_array( i ) );
    end generate;

process( reg_file_array, reg_file )
    variable reg_file_col : std_logic_vector( reg_width - 1 downto 0 );
    begin
        for i in nr_regs-1 downto 0 loop
            reg_file_col := reg_file_array( i );
            for j in reg_width-1 downto 0 loop
                reg_file( i, j ) <= reg_file_col( j );
            end loop;
        end loop;
    end process;

process( load_enable_sig )
    begin
        for i in 0 to nr_regs-1 loop
            reg_mode_sig( i ) <= load_enable_sig( i ) & load_enable_sig( i );
        end loop;
    end process;

end NDV;

```



```

-- *****
-- **** STUDENT: 64200100
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
use ieee.math_real.all;

entity reg_file is
    generic(
        nr_regs          : natural := 4;
        reg_width        : natural := 8 );
    PORT ( clk,          -- clock input
        LE               : in std_logic;    -- Load enable input      ( active '1' )
        nRST             : in std_logic;    -- reset input                ( active '0' )
        dest_select,     -- register number destination select input
        A_select,        -- A, B bus destination select input
        B_select         : in  std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        D               : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus input
        A, B            : out std_logic_vector( reg_width - 1 downto 0 )    -- A, B bus output
    );
end reg_file;

architecture NDV of reg_file is

    constant nr_regs_size : integer := integer( ceil( log2( real( nr_regs - 1 ) ) ) );
    signal    load_enable_sig : std_logic_vector( nr_regs - 1 DOWNT0 0 ) := ( others => '0' );
    SIGNAL     reg_file       : muxnto1_bus_type( nr_regs - 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others
=> ( others => '0' ) );

    type reg_file_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
    signal    reg_file_array : reg_file_array_type := ( others => ( others => '0' ) );

    type reg_mode_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( 1 downto 0 );
    signal    reg_mode_sig   : reg_mode_array_type := ( others => ( others => '0' ) );

    COMPONENT dmuxnto1 IS
    generic( n_addr: natural:=2 );

```

```

PORT ( s:in STD_LOGIC_VECTOR( n_addr-1 downto 0 );
w:in STD_LOGIC;
f:OUT STD_LOGIC_VECTOR( 2**n_addr-1 downto 0 ) );
END COMPONENT;

```

```

COMPONENT muxnto1_bus IS
generic( n_addr:INTEGER:=2;
bus_width:INTEGER:= 8 );
PORT ( s:IN std_logic_vector( n_addr-1 downto 0 );
w:IN muxnto1_bus_type( 2**n_addr-1 DOWNT0 0, bus_width-1 DOWNT0 0 );
f:OUT std_logic_vector( bus_width-1 downto 0 ) );
END COMPONENT;

```

```

COMPONENT shift_reg is
generic( reg_size: natural:=4 );
PORT ( clk, nCLR, sr_in, sl_in : IN std_logic;
s : in std_logic_vector( 1 downto 0 );
x : in std_logic_vector( reg_size-1 downto 0 );
Q : out std_logic_vector( reg_size-1 downto 0 ) );
end COMPONENT;

```

```

begin
dmux: dmuxnto1
generic map ( n_addr => nr_regs_size )
port map( w => LE,s => dest_select,f => load_enable_sig );

```

```

A_BUS_MUX: muxnto1_bus
generic map ( n_addr => nr_regs_size, bus_width => reg_width )
port map ( s => A_select,w => reg_file,f => A );

```

```

B_BUS_MUX: muxnto1_bus
generic map ( n_addr => nr_regs_size, bus_width => reg_width )
port map ( s => B_select,w => reg_file,f => B );

```

```

registrs: for i in 0 to nr_regs - 1 generate
regs1: shift_reg
generic map ( reg_size => reg_width )
port map ( s => reg_mode_sig( i ),x => D, Q => reg_file_array( i ),clk => clk,nCLR => nRST,sr_in => '0',sl_in => '0'
);
end generate;

```

```

process ( reg_file_array ) is
variable reg_file_col : std_logic_vector( reg_width-1 downto 0 );
begin
for i in 0 to nr_regs-1 loop
reg_file_col:= reg_file_array( i );
for j in 0 to reg_width-1 loop
reg_file( i, j ) <= reg_file_col( j );
end loop;
end loop;
end process;
process ( load_enable_sig ) is
begin
for i in 0 to nr_regs-1 loop
reg_mode_sig( i ) <= load_enable_sig( i ) & load_enable_sig( i );
end loop;
end process;
end NDV;

```

```

-- *****
-- **** STUDENT: 64200112
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
use ieee.math_real.all;

entity reg_file is
    generic(
        nr_regs          : natural := 4;
        reg_width        : natural := 8 );
    PORT ( clk,          -- clock input
        LE               : in std_logic;    -- Load enable input      ( active '1' )
        nRST             : in std_logic;    -- reset input                ( active '0' )
        dest_select,     -- register number destination select input
        A_select,        -- A, B bus destination select input
        B_select         : in  std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        D               : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus input
        A, B            : out std_logic_vector( reg_width - 1 downto 0 )    -- A, B bus output
    );
end reg_file;

architecture NDV of reg_file is

    constant nr_regs_size : integer := integer( ceil( log2( real( nr_regs - 1 ) ) ) );
    signal    load_enable_sig : std_logic_vector( nr_regs - 1 DOWNT0 0 ) := ( others => '0' );
    SIGNAL     reg_file      : muxnto1_bus_type( nr_regs - 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others
=> ( others => '0' ) );

    type reg_file_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
    signal    reg_file_array : reg_file_array_type := ( others => ( others => '0' ) );

    type reg_mode_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( 1 downto 0 );
    signal    reg_mode_sig   : reg_mode_array_type := ( others => ( others => '0' ) );

    COMPONENT dmuxnto1 IS
        generic(
            n_addr: natural := 2 );

```

```

        PORT (
            s : in      std_logic_vector( n_addr - 1 downto 0 );
            w : in      STD_LOGIC;
            f : OUT     std_logic_vector( 2**n_addr - 1 downto 0 )
        );
END COMPONENT;

COMPONENT muxnto1_bus IS
    generic(
        n_addr      :    INTEGER := 2;
        bus_width   :    INTEGER := 8 );
    PORT (
        s           :    IN      std_logic_vector( n_addr - 1 downto 0 );
        w           :    IN      muxnto1_bus_type( 2**n_addr - 1 DOWNT0 0, bus_width - 1
DOWNT0 0 );
        f           :    OUT     std_logic_vector( bus_width - 1 downto 0 )
    );
END COMPONENT;

COMPONENT shift_reg is
    generic( reg_size: natural := 4 );
    PORT ( clk, nCLR, sr_in, sl_in : IN std_logic;
        s : in std_logic_vector( 1 downto 0 );
        x : in std_logic_vector( reg_size - 1 downto 0 );
        Q : out std_logic_vector( reg_size - 1 downto 0 )
    );
END COMPONENT;

begin

    U0 : dmuxnto1
        generic map( n_addr => nr_regs_size )
        port map( w => LE,          s => dest_select,          f => load_enable_sig
        );

    U1A : muxnto1_bus
        generic map( n_addr => nr_regs_size,          bus_width => reg_width )
        port map( s => A_select,          w => reg_file,          f => A
        );

    U1B : muxnto1_bus
        generic map( n_addr => nr_regs_size,          bus_width => reg_width )
        port map( s => B_select,          w => reg_file,          f => B
        );

```

```

    );

U2 : for i in 0 to nr_regs - 1 generate
    U3 : shift_reg
        generic map( reg_size => reg_width )
        port map( s => reg_mode_sig( i ),          x => D,          Q => reg_file_array( i ),
clk => clk,          nCLR => nRST,          sr_in => '0',          sl_in => '0'
        );
    end generate;

P0 : process( reg_file_array ) is
    variable reg_file_col : std_logic_vector( reg_width - 1 downto 0 );
begin
    for i in 0 to nr_regs - 1 loop
        reg_file_col := reg_file_array( i );
        for j in 0 to reg_width - 1 loop
            reg_file( i, j ) <= reg_file_col( j );
        end loop;
    end loop;
end process;

P1 : process( load_enable_sig ) is
begin
    for i in 0 to nr_regs - 1 loop
        reg_mode_sig( i ) <= load_enable_sig( i ) & load_enable_sig( i );
    end loop;
end process;

end NDV;

```

```

-- *****
-- **** STUDENT: 64200163
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

library ieee;
use ieee.std_logic_1164.all;
use work.reg_file_functions.all;
use ieee.math_real.all;

entity reg_file is
    generic( nr_regs : natural := 4;
             reg_width : natural := 8 );
    port( clk,      LE : in std_logic;
          nRST : in std_logic;
          dest_select,      A_select,
          B_select : in std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
          D : in      std_logic_vector( reg_width - 1 downto 0 );
          A, B : out std_logic_vector( reg_width - 1 downto 0 )
    );
end reg_file;

architecture NDV of reg_file is
    component dmuxnto1 is
        generic( n_addr: natural := 2 );
        port( s : in std_logic_vector( n_addr - 1 downto 0 );
              w : in std_logic;
              f : out std_logic_vector( 2*n_addr - 1 downto 0 )
        );
    end component;
    component muxnto1_bus is
        generic( n_addr : integer := 2;
                 bus_width : integer := 8 );
        port( s : in std_logic_vector( n_addr - 1 downto 0 );
              w : in muxnto1_bus_type( 2*n_addr - 1 downto 0, bus_width - 1 downto 0 );
              f : out std_logic_vector( bus_width - 1 downto 0 )
        );
    end component;
    component shift_reg is

```

```

        generic( reg_size : natural := 4 );
        port( clk,          nCLR,          sr_in,          sl_in : in std_logic;
              s : in std_logic_vector( 1 downto 0 );
              x : in std_logic_vector( reg_size - 1 downto 0 );
              Q : out std_logic_vector( reg_size - 1 downto 0 )
        );
    end component;
    constant nr_regs_size : integer := integer( ceil( log2( real( nr_regs - 1 ) ) ) );
    signal load_enable_sig : std_logic_vector( nr_regs - 1 downto 0 ) := ( others => '0' );
    signal reg_file : muxnto1_bus_type( nr_regs - 1 downto 0, reg_width - 1 downto 0 ) := ( others => (
others => '0' ) );
    type reg_file_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
    signal reg_file_array : reg_file_array_type := ( others => ( others => '0' ) );
    type reg_mode_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( 1 downto 0 );
    signal reg_mode_sig : reg_mode_array_type := ( others => ( others => '0' ) );
begin
    dmux: dmuxnto1
        generic map( n_addr => nr_regs_size )
        port map(
            w => LE,          s => dest_select,          f => load_enable_sig
        );
    amux_bus: muxnto1_bus
        generic map( n_addr => nr_regs_size,          bus_width => reg_width )
        port map (
            s => A_select,          w => reg_file,          f => A
        );
    bmux_bus: muxnto1_bus
        generic map( n_addr => nr_regs_size,          bus_width => reg_width )
        port map(
            s => B_select,          w => reg_file,          f => B
        );
    registers: for i in 0 to nr_regs - 1 generate
        reg: shift_reg
            generic map( reg_size => reg_width )
            port map(
                s => reg_mode_sig( i ),          x => D,          Q => reg_file_array( i ),          clk => clk,
nCLR => nRST,          sr_in => '0',          sl_in => '0'
            );
    end generate;
    process( reg_file_array ) is

```



```

variable reg_file_col : std_logic_vector( reg_width - 1 downto 0 );
begin
    for i in 0 to nr_regs - 1 loop
        reg_file_col := reg_file_array( i );
        for j in 0 to reg_width - 1 loop
            reg_file( i, j ) <= reg_file_col( j );
        end loop;
    end loop;
end process;
process( load_enable_sig, LE ) is
begin
    for i in 0 to nr_regs - 1 loop
        reg_mode_sig( i )( 0 ) <= load_enable_sig( i ) and LE;
        reg_mode_sig( i )( 1 ) <= load_enable_sig( i ) and LE;
    end loop;
end process;
end NDV;

```

```

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

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
use ieee.math_real.all;

entity reg_file is
    generic(
        nr_regs          : natural := 4;
        reg_width        : natural := 8 );
    PORT ( clk,          -- clock input
        LE               : in std_logic;    -- Load enable input      ( active '1' )
        nRST             : in std_logic;    -- reset input                ( active '0' )
        dest_select,     -- register number destination select input
        A_select,        -- A, B bus destination select input
        B_select         : in  std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        D               : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus input
        A, B            : out std_logic_vector( reg_width - 1 downto 0 )    -- A, B bus output
    );
end reg_file;

architecture NDV of reg_file is

    constant nr_regs_size : integer := integer( ceil( log2( real( nr_regs - 1 ) ) ) );
    signal    load_enable_sig : std_logic_vector( nr_regs - 1 DOWNT0 0 ) := ( others => '0' );
    SIGNAL     reg_file      : muxnto1_bus_type( nr_regs - 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others
=> ( others => '0' ) );

    type reg_file_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
    signal    reg_file_array : reg_file_array_type := ( others => ( others => '0' ) );

    type reg_mode_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( 1 downto 0 );
    signal    reg_mode_sig   : reg_mode_array_type := ( others => ( others => '0' ) );

    component shift_reg is
        generic( reg_size: natural := 4 );

```

```

        PORT ( clk, nCLR, sr_in, sl_in : IN std_logic;
              s : in std_logic_vector( 1 downto 0 );
              x : in std_logic_vector( reg_size - 1 downto 0 );
              Q : out std_logic_vector( reg_size - 1 downto 0 )
              );
end component;

component dmuxnto1 IS
    generic(
        n_addr: natural := 2 );
    PORT (
        s : in std_logic_vector( n_addr - 1 downto 0 );
        w : in STD_LOGIC;
        f : OUT std_logic_vector( 2**n_addr - 1 downto 0 )
        );
END component;

component muxnto1_bus IS
    generic(
        n_addr      : INTEGER := 2;
        bus_width    : INTEGER := 8 );
    PORT (
        s            : IN std_logic_vector( n_addr - 1 downto 0 );
        w : IN muxnto1_bus_type( 2**n_addr - 1 DOWNT0 0, bus_width - 1 DOWNT0 0 );
        f            : OUT std_logic_vector( bus_width - 1 downto 0 )
        );
END component;

BEGIN
    -- Zanka za dostop do vrstic `reg_file_array`

    PROCESS( reg_file_array ) is
        VARIABLE temp_stolpci : std_logic_vector( reg_width - 1 DOWNT0 0 );
        BEGIN

            FOR i IN 0 TO nr_regs - 1 LOOP

                temp_stolpci := reg_file_array( i );

                FOR j IN 0 TO reg_width - 1 LOOP
                    reg_file( i,j ) <= temp_stolpci( j );
                END LOOP;
            END LOOP;
        END PROCESS;

```

```

U1: for i in 0 to nr_regs - 1 generate

    U1_1: shift_reg
        generic map ( reg_size => reg_width )
        port map (
            s => reg_mode_sig( i ), x => D, Q => reg_file_array( i ),      clk => clk, nCLR => nRST, sr_in
=> '0', sl_in => '0'
        );
    end generate;

U2: dmuxnto1
generic map ( n_addr => nr_regs_size )
port map(
    w => LE, s => dest_select, f => load_enable_sig
);

U3: muxnto1_bus
generic map ( n_addr => nr_regs_size, bus_width => reg_width )
port map (
    s => A_select, w => reg_file, f => A
);

U4: muxnto1_bus
generic map ( n_addr => nr_regs_size, bus_width => reg_width )
port map (
    s => B_select, w => reg_file, f => B
);

process( load_enable_sig ) is
begin
    for i in 0 to nr_regs-1 loop
        reg_mode_sig( i )  <= load_enable_sig( i ) & load_enable_sig( i );
    end loop;
end process;

```

```
end NDV;
```

```

-- *****
-- **** STUDENT: 64200288
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
use ieee.math_real.all;

entity reg_file is
    generic(
        nr_regs          : natural := 4;
        reg_width        : natural := 8 );
    PORT ( clk,          -- clock input
        LE               : in std_logic;    -- Load enable input      ( active '1' )
        nRST             : in std_logic;    -- reset input                ( active '0' )
        dest_select,     -- register number destination select input
        A_select,        -- A, B bus destination select input
        B_select         : in  std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        D               : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus input
        A, B            : out std_logic_vector( reg_width - 1 downto 0 )    -- A, B bus output
    );
end reg_file;

architecture NDV of reg_file is

    COMPONENT dmuxnto1 IS
        generic(
            n_addr: natural := 2 );
        PORT (
            s : in      std_logic_vector( n_addr - 1 downto 0 );
            w : in      STD_LOGIC;
            f : OUT     std_logic_vector( 2*n_addr - 1 downto 0 )
        );
    END COMPONENT;

    COMPONENT muxnto1_bus IS
        generic(
            n_addr      :      INTEGER := 2;
            bus_width   :      INTEGER := 8 );
        PORT (
            s           :      IN      std_logic_vector( n_addr - 1 downto 0 );

```

```

        w          :      IN          muxnto1_bus_type( 2**n_addr - 1 DOWNT0 0, bus_width - 1
DOWNT0 0 );
        f          :      OUT      std_logic_vector( bus_width - 1 downto 0 )
    );
END COMPONENT;

COMPONENT shift_reg is
    generic( reg_size: natural := 4 );
    PORT ( clk, nCLR, sr_in, sl_in : IN std_logic;
        s : in std_logic_vector( 1 downto 0 );
        x : in std_logic_vector( reg_size - 1 downto 0 );
        Q : out std_logic_vector( reg_size - 1 downto 0 )
    );
end COMPONENT;

constant    nr_regs_size : integer := integer( ceil( log2( real( nr_regs - 1 ) ) ) );
signal      load_enable_sig      : std_logic_vector( nr_regs - 1 DOWNT0 0 ) := ( others => '0' );
SIGNAL      reg_file              : muxnto1_bus_type( nr_regs - 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others
=> ( others => '0' ) );

type reg_file_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
signal      reg_file_array : reg_file_array_type := ( others => ( others => '0' ) );

type reg_mode_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( 1 downto 0 );
signal      reg_mode_sig      : reg_mode_array_type := ( others => ( others => '0' ) );

begin

    MuxB: muxnto1_bus
    generic map ( n_addr => nr_regs_size, bus_width => reg_width )
    port map (
        s => B_select,      w => reg_file,      f => B
    );

    MuxA: muxnto1_bus
    generic map ( n_addr => nr_regs_size, bus_width => reg_width )
    port map (
        s => A_select,      w => reg_file,      f => A
    );

```

```

MuxD: dmuxnto1
generic map ( n_addr => nr_regs_size )
port map(
    w => LE,      s => dest_select,  f => load_enable_sig
);

Regstr: for i in 0 to nr_regs - 1 generate
    reg: shift_reg
        generic map ( reg_size => reg_width )
        port map (
            s => reg_mode_sig( i ),      x => D,      Q => reg_file_array( i ),      clk => clk,
nCLR => nRST,      sr_in => '0',      sl_in => '0'
        );
end generate;

process ( reg_file_array ) is
    variable reg_file_col : std_logic_vector( reg_width - 1 downto 0 );
begin
    for A in 0 to nr_regs-1 loop
        reg_file_col := reg_file_array( A );
        for B in 0 to reg_width-1 loop
            reg_file( A, B ) <= reg_file_col( B );
        end loop;
    end loop;
end process;

process ( load_enable_sig ) is
begin
    for z in 0 to nr_regs-1 loop
        reg_mode_sig( z ) <= load_enable_sig( z ) & load_enable_sig( z );
    end loop;
end process;

end NDV;

```



```

-- *****
-- **** STUDENT: 64200296
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
use ieee.math_real.all;

entity reg_file is
    generic(
        nr_regs          : natural := 4;
        reg_width        : natural := 8 );
    PORT ( clk,          -- clock input
        LE               : in std_logic;    -- Load enable input      ( active '1' )
        nRST             : in std_logic;    -- reset input                ( active '0' )
        dest_select,     -- register number destination select input
        A_select,        -- A, B bus destination select input
        B_select         : in  std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        D               : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus input
        A, B            : out std_logic_vector( reg_width - 1 downto 0 )    -- A, B bus output
    );
end reg_file;

architecture NDV of reg_file is

    constant nr_regs_size : integer := integer( ceil( log2( real( nr_regs - 1 ) ) ) );
    signal    load_enable_sig : std_logic_vector( nr_regs - 1 DOWNT0 0 ) := ( others => '0' );
    SIGNAL     reg_file      : muxnto1_bus_type( nr_regs - 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others
=> ( others => '0' ) );

    type reg_file_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
    signal    reg_file_array : reg_file_array_type := ( others => ( others => '0' ) );

    type reg_mode_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( 1 downto 0 );
    signal    reg_mode_sig   : reg_mode_array_type := ( others => ( others => '0' ) );

    COMPONENT dmuxnto1 IS
        generic(
            n_addr: natural := 2 );

```

```

        PORT (
            s : in      std_logic_vector( n_addr - 1 downto 0 );
            w : in      STD_LOGIC;
            f : OUT     std_logic_vector( 2**n_addr - 1 downto 0 )
        );
END COMPONENT;

COMPONENT muxnto1_bus IS
    generic(
        n_addr      : INTEGER := 2;
        bus_width   : INTEGER := 8 );
    PORT ( s
        : IN      std_logic_vector( n_addr - 1 downto 0 );
        w
        : IN      muxnto1_bus_type( 2**n_addr - 1 DOWNT0 0, bus_width - 1
DOWNT0 0 );
        f
        : OUT     std_logic_vector( bus_width - 1 downto 0 )
    );
END COMPONENT;

COMPONENT shift_reg is
    generic( reg_size: natural := 4 );
    PORT ( clk, nCLR, sr_in, sl_in : IN std_logic;
        s : in std_logic_vector( 1 downto 0 );
        x : in std_logic_vector( reg_size - 1 downto 0 );
        Q : out std_logic_vector( reg_size - 1 downto 0 )
    );
END COMPONENT;

begin

    dmuxnto1_inst: dmuxnto1
        generic map ( n_addr => nr_regs_size )
        port map(
            w => LE,          s => dest_select,          f => load_enable_sig
        );

    A_BUS_MUX: muxnto1_bus
    generic map (
        n_addr => nr_regs_size,
        bus_width => reg_width
    )
    port map (
        w => reg_file,

```

```

s => A_select,
f => A
);

B_BUS_MUX : muxnto1_bus
generic map (
n_addr => nr_regs_size,
bus_width => reg_width
)
port map (
w => reg_file,
s => B_select,
f => B
);

process ( reg_file_array ) is
variable reg_file_col : std_logic_vector( reg_width - 1 downto 0 );
begin
for i in 0 to nr_regs-1 loop
reg_file_col := reg_file_array( i );
for j in 0 to reg_width-1 loop
reg_file( i, j ) <= reg_file_col( j );
end loop;
end loop;
end process;

-- Mode signal process
mode_sig_proc : process( load_enable_sig )
begin
for i in 0 to nr_regs - 1 loop
reg_mode_sig( i ) <= load_enable_sig( i ) & load_enable_sig( i );
end loop;
end process;

gen_shift_regs: for i in 0 to nr_regs - 1 generate
shift_reg_inst: shift_reg
generic map (
reg_size => reg_width
)
port map (

```

```
clk => clk,  
nCLR => nRST,  
sr_in => '0',          sl_in => '0',  
s => reg_mode_sig( i ),  
x => D,  
Q => reg_file_array( i )  
);  
end generate;  
  
    end NDV;
```

```

-- *****
-- **** STUDENT: 64200385
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
use ieee.math_real.all;

entity reg_file is
    generic(
        nr_regs          : natural := 4;
        reg_width        : natural := 8 );
    PORT ( clk,          -- clock input
        LE               : in std_logic;    -- Load enable input      ( active '1' )
        nRST             : in std_logic;    -- reset input                ( active '0' )
        dest_select,     -- register number destination select input
        A_select,        -- A, B bus destination select input
        B_select         : in  std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        D               : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus input
        A, B            : out std_logic_vector( reg_width - 1 downto 0 )    -- A, B bus output
    );
end reg_file;

architecture NDV of reg_file is

    constant nr_regs_size : integer := integer( ceil( log2( real( nr_regs - 1 ) ) ) );
    signal    load_enable_sig : std_logic_vector( nr_regs - 1 DOWNT0 0 ) := ( others => '0' );
    SIGNAL     reg_file      : muxnto1_bus_type( nr_regs - 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others
=> ( others => '0' ) );

    type reg_file_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
    signal    reg_file_array : reg_file_array_type := ( others => ( others => '0' ) );

    type reg_mode_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( 1 downto 0 );
    signal    reg_mode_sig   : reg_mode_array_type := ( others => ( others => '0' ) );

    component dmuxnto1 is
        generic(
            n_addr: natural := 2 );

```

```

        PORT (
            s : in      std_logic_vector( n_addr - 1 downto 0 );
            w : in      STD_LOGIC;
            f : OUT     std_logic_vector( 2**n_addr - 1 downto 0 )
        );
end component;

component muxnto1_bus is
    generic(
        n_addr      :    INTEGER := 2;
        bus_width   :    INTEGER := 8 );
    PORT (
        s           :    IN      std_logic_vector( n_addr - 1 downto 0 );
        w           :    IN      muxnto1_bus_type( 2**n_addr - 1 DOWNT0 0, bus_width - 1
DOWNT0 0 );
        f           :    OUT     std_logic_vector( bus_width - 1 downto 0 )
    );
end component;

component shift_reg is
    generic( reg_size: natural := 4 );
    PORT ( clk, nCLR, sr_in, sl_in : IN std_logic;
        s : in std_logic_vector( 1 downto 0 );
        x : in std_logic_vector( reg_size - 1 downto 0 );
        Q : out std_logic_vector( reg_size - 1 downto 0 )
    );
end component;

begin

U1: dmuxnto1
    generic map( n_addr => nr_regs_size )
    port map ( w => LE,
        s => dest_select,
        f => load_enable_sig );

U2: muxnto1_bus
    generic map( n_addr => nr_regs_size,
        bus_width => reg_width )
    port map ( s => A_select,
        w => reg_file,
        f => A );

```

```

U3: muxnto1_bus
generic map( n_addr => nr_regs_size,
bus_width => reg_width )
port map ( s => B_select,
w => reg_file,
f => B );

sr: for i in 0 to nr_regs - 1 generate
U4: shift_reg
generic map( reg_size => reg_width )
port map ( clk => clk,
nCLR => nRST,
sr_in => '0',
sl_in => '0',
s => reg_mode_sig( i ),
x => D,
Q => reg_file_array( i ) );
end generate;

process( reg_file_array ) is
variable reg_file_col : std_logic_vector( reg_width - 1 downto 0 );
begin
for i in 0 to nr_regs - 1 loop
reg_file_col := reg_file_array( i );
for j in 0 to reg_width - 1 loop
reg_file( i,j ) <= reg_file_col( j );
end loop;
end loop;
end process;

process( load_enable_sig ) is
begin
for i in 0 to nr_regs - 1 loop
reg_mode_sig( i ) <= load_enable_sig( i ) & load_enable_sig( i );
end loop;
end process;

end NDV;

```

```

-- *****
-- **** STUDENT: 64210113
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
use ieee.math_real.all;

entity reg_file is
    generic(
        nr_regs          : natural := 4;
        reg_width         : natural := 8 );
    PORT ( clk,          -- clock input
        LE               : in std_logic;    -- Load enable input      ( active '1' )
        nRST             : in std_logic;    -- reset input                ( active '0' )
        dest_select, -- register number destination select input
        A_select,     -- A, B bus destination select input
        B_select      : in  std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        D             : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus input
        A, B          : out std_logic_vector( reg_width - 1 downto 0 )    -- A, B bus output
    );
end reg_file;

architecture NDV of reg_file is

    COMPONENT muxnto1_bus IS
        generic(
            n_addr      : INTEGER := 2;
            bus_width    : INTEGER := 8 );
        PORT (
            s            : IN      std_logic_vector( n_addr - 1 downto 0 );
            w            : IN      muxnto1_bus_type( 2*n_addr - 1 DOWNT0 0, bus_width - 1
DOWNT0 0 );
            f            : OUT     std_logic_vector( bus_width - 1 downto 0 )
        );
    END COMPONENT;

    COMPONENT dmuxnto1 IS
        generic(
            n_addr: natural := 2 );
        PORT (
            s : in      std_logic_vector( n_addr - 1 downto 0 );

```



```

        w : in      STD_LOGIC;
        f : OUT     std_logic_vector( 2**n_addr - 1 downto 0 )
    );
END COMPONENT;

COMPONENT shift_reg is
    generic( reg_size: natural := 4 );
    PORT ( clk, nCLR, sr_in, sl_in : IN std_logic;
          s : in std_logic_vector( 1 downto 0 );
          x : in std_logic_vector( reg_size - 1 downto 0 );
          Q : out std_logic_vector( reg_size - 1 downto 0 )
    );
end COMPONENT;

constant    nr_regs_size : integer := integer( ceil( log2( real( nr_regs - 1 ) ) ) );
signal      load_enable_sig : std_logic_vector( nr_regs - 1 DOWNT0 0 ) := ( others => '0' );
SIGNAL      reg_file
=> ( others => '0' ) ) : muxnto1_bus_type( nr_regs - 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others
=> ( others => '0' ) );

type reg_file_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
signal      reg_file_array : reg_file_array_type := ( others => ( others => '0' ) );

type reg_mode_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( 1 downto 0 );
signal      reg_mode_sig : reg_mode_array_type := ( others => ( others => '0' ) );

begin

    shtReg: for i in 0 to nr_regs - 1 generate
        reg: shift_reg
            generic map ( reg_size => reg_width )
            port map (
                s => reg_mode_sig( i ),    x => D,        Q => reg_file_array( i ), clk => clk,    nCLR => nRST,        sr_in
=> '0',    sl_in => '0'
            );

        dmux: dmuxnto1
            generic map ( n_addr => nr_regs_size )
            port map(
                w => LE,        s => dest_select,    f => load_enable_sig
            );
    end generate;
end;

```

```

muxx: muxnto1_bus
generic map ( n_addr => nr_regs_size, bus_width => reg_width )
port map (
    s => A_select,      w => reg_file,      f => A
);

```

```

muxxx: muxnto1_bus
generic map ( n_addr => nr_regs_size, bus_width => reg_width )
port map (
    s => B_select,      w => reg_file,      f => B
);

```

```

end generate;

```

```

    process ( reg_file_array ) is
        variable reg_file_col : std_logic_vector( reg_width - 1 downto 0 );

```

```

    begin
        for i in 0 to nr_regs-1 loop
            reg_file_col := reg_file_array( i );

            for j in 0 to reg_width-1 loop
                reg_file( i, j ) <= reg_file_col( j );
            end loop;
        end loop;
    end process;

```

```

    process ( load_enable_sig ) is
    begin

        for i in 0 to nr_regs-1 loop
            reg_mode_sig( i ) <= load_enable_sig( i )&load_enable_sig( i );
        end loop;

```

```

    end process;

```

```

end NDV;

```

```

-- *****
-- **** STUDENT: 64210290
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
USE ieee.math_real.all;

ENTITY reg_file IS
  GENERIC(
    nr_regs : NATURAL := 4;
    reg_width : NATURAL := 8
  );
  PORT(
    clk,    -- clock input
    LE      : IN STD_LOGIC;    -- Load enable input      ( active '1' )
    nRST    : IN STD_LOGIC;    -- reset input          ( active '0' )
    dest_select, -- register number destination select input
    A_select,   -- A, B bus destination select input
    B_select    : IN STD_LOGIC_VECTOR( sizeof( nr_regs - 1 ) - 1 downto 0 );
    D           : IN STD_LOGIC_VECTOR( reg_width - 1 downto 0 );    -- data input bus input
    A, B       : OUT STD_LOGIC_VECTOR( reg_width - 1 downto 0 )    -- A, B bus output
  );
END reg_file;

ARCHITECTURE NDV OF reg_file IS

  component dmuxnto1 is
    generic(
      n_addr: natural
    );
    port(
      s : in std_logic_vector( n_addr - 1 downto 0 );
      w : in std_logic;
      f : out std_logic_vector( 2*n_addr - 1 downto 0 )
    );
  end component;

```

```

component shift_reg is
    generic(
        reg_size: natural := 4
    );
    port(
        clk, nCLR, sr_in, sl_in : IN std_logic;
        s : in std_logic_vector( 1 downto 0 );
        x : in std_logic_vector( reg_size - 1 downto 0 );
        Q : out std_logic_vector( reg_size - 1 downto 0 )
    );
end component;

component muxnto1_bus is
    generic(
        n_addr : integer;
        bus_width : integer
    );
    port(
        s : in std_logic_vector( n_addr - 1 downto 0 );
        w : in muxnto1_bus_type( 2**n_addr - 1 downto 0, bus_width - 1 downto 0 );
        f : out std_logic_vector( bus_width - 1 downto 0 )
    );
end component;

constant nr_regs_size : integer := integer( ceil( log2( real( nr_regs - 1 ) ) ) );
signal load_enable_sig : std_logic_vector( nr_regs - 1 DOWNT0 0 ) := ( others => '0' );
signal reg_file
    : muxnto1_bus_type( nr_regs - 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := (
others => ( others => '0' ) );

type reg_file_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
signal reg_file_array : reg_file_array_type := ( others => ( others => '0' ) );

type reg_mode_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( 1 downto 0 );
signal reg_mode_sig : reg_mode_array_type := ( others => ( others => '0' ) );

BEGIN

dmux : dmuxnto1
    generic map(

```

```

        n_addr => nr_regs_size
    )
    port map(
        s => dest_select,          w => LE,          f => load_enable_sig
    );

mux_entry_wires : process( reg_file_array )
begin
    for hor in 0 to reg_width-1 loop
        for ver in 0 to nr_regs-1 loop
            reg_file( ver,hor ) <= reg_file_array( ver )( hor );
        end loop;
    end loop;
end process;

R_gen : for i in 0 to nr_regs-1 generate
    R : shift_reg
        generic map(
            reg_size => reg_width
        )
        port map(
            clk => clk,          nCLR => nRST,          sr_in => '0',          sl_in => '0',
s => reg_mode_sig( i ),          x => D,          Q => reg_file_array( i )
        );
end generate R_gen;

load_gen : process( LE,load_enable_sig )
begin
    for i in 0 to nr_regs-1 loop
        reg_mode_sig( i )( 0 )    <= load_enable_sig( i ) and LE;
        reg_mode_sig( i )( 1 )    <= load_enable_sig( i ) and LE;
    end loop;
end process;

A_bus_mux : muxnto1_bus
    generic map(
        n_addr => nr_regs_size,          bus_width => reg_width
    )
    port map(
        s => A_select,          w => reg_file,          f => A

```

```
    );  
    B_bus_mux : muxnto1_bus  
        generic map(  
            n_addr => nr_regs_size,          bus_width => reg_width  
        )  
        port map(  
            s => B_select,          w => reg_file,          f => B  
        );  
END NDV;
```

```

-- *****
-- **** STUDENT: 64210382
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
use ieee.math_real.all;

entity reg_file is
    generic(
        nr_regs          : natural := 4;
        reg_width        : natural := 8 );
    PORT ( clk,          -- clock input
        LE               : in std_logic;    -- Load enable input      ( active '1' )
        nRST             : in std_logic;    -- reset input                ( active '0' )
        dest_select,     -- register number destination select input
        A_select,        -- A, B bus destination select input
        B_select         : in  std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        D               : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus input
        A, B            : out std_logic_vector( reg_width - 1 downto 0 )    -- A, B bus output
    );
end reg_file;

architecture NDV of reg_file is

    constant nr_regs_size : integer := integer( ceil( log2( real( nr_regs - 1 ) ) ) );
    signal    load_enable_sig : std_logic_vector( nr_regs - 1 DOWNT0 0 ) := ( others => '0' );
    SIGNAL     reg_file      : muxnto1_bus_type( nr_regs - 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others
=> ( others => '0' ) );

    type reg_file_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
    signal    reg_file_array : reg_file_array_type := ( others => ( others => '0' ) );

    type reg_mode_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( 1 downto 0 );
    signal    reg_mode_sig   : reg_mode_array_type := ( others => ( others => '0' ) );

begin

```

```

dmux: dmuxnto1
generic map ( n_addr => nr_regs_size )
port map(
    w => LE, s => dest_select, f => load_enable_sig
);

amux: muxnto1_bus
generic map ( n_addr => nr_regs_size, bus_width => reg_width )
port map (
    s => A_select, w => reg_file, f => A
);

bmux: muxnto1_bus
generic map ( n_addr => nr_regs_size, bus_width => reg_width )
port map (
    s => B_select, w => reg_file, f => B
);

regs: for i in 0 to nr_regs - 1 generate
    reg: shift_reg
        generic map ( reg_size => reg_width )
        port map (
            s => reg_mode_sig( i ), x => D, Q => reg_file_array( i ),
=> '0', sl_in => '0'
        );
    end generate;

process ( reg_file_array ) is
    variable reg_file_col : std_logic_vector( reg_width - 1 downto 0 );
begin
    for idx in 0 to nr_regs-1 loop
        reg_file_col := reg_file_array( idx );
        for jdx in 0 to reg_width-1 loop
            reg_file( idx, jdx )      <= reg_file_col( jdx );
        end loop;
    end loop;
end process;

process ( load_enable_sig ) is
begin

```



```
        for i in 0 to nr_regs-1 loop
            reg_mode_sig( i )  <= load_enable_sig( i ) & load_enable_sig( i );
        end loop;
    end process;

end NDV;
```

```

-- *****
-- **** STUDENT: 64210384
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
use ieee.math_real.all;

entity reg_file is
    generic(
        nr_regs          : natural := 4;
        reg_width        : natural := 8 );
    PORT ( clk,          -- clock input
          LE,            -- Load enable input      ( active '1' )
          nRST           : in std_logic;          -- reset input      ( active '0' )
          dest_select,    -- register number destination select input
          A_select,       -- A, B bus destination select input
          B_select        : in  std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
          D               : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus input
          A, B            : out std_logic_vector( reg_width - 1 downto 0 )    -- A, B bus output
    );
end reg_file;

architecture NDV of reg_file is

    component dmuxnto1
    generic ( n_addr : natural := 2 );
    port(
        s : in std_logic_vector( n_addr - 1 downto 0 );
        w : in std_logic;
        f : out      std_logic_vector( 2*n_addr - 1 downto 0 )
    );
end component;

    component muxnto1_bus
    generic(
        n_addr : integer := 2;
        bus_width : integer := 8

```

```

    );
port(
    s : in std_logic_vector( n_addr - 1 downto 0 );
    w : in muxnto1_bus_type( 2**n_addr - 1 downto 0, bus_width - 1 downto 0 );
    f : out std_logic_vector( bus_width - 1 downto 0 )
);
end component;

component shift_reg
generic ( reg_size : natural := 4 );
port(
    clk, nCLR, sr_in, sl_in : in std_logic;
    s : in std_logic_vector( 1 downto 0 );
    x : in std_logic_vector( reg_size - 1 downto 0 );
    Q : out std_logic_vector( reg_size - 1 downto 0 )
);
end component;

constant    nr_regs_size : integer := integer( ceil( log2( real( nr_regs - 1 ) ) ) );
signal      load_enable_sig : std_logic_vector( nr_regs - 1 DOWNT0 0 ) := ( others => '0' );
SIGNAL      reg_file : muxnto1_bus_type( nr_regs - 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others
=> ( others => '0' ) );

type reg_file_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
signal      reg_file_array : reg_file_array_type := ( others => ( others => '0' ) );

type reg_mode_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( 1 downto 0 );
signal      reg_mode_sig : reg_mode_array_type := ( others => ( others => '0' ) );

begin
    dmux: dmuxnto1
        generic map ( n_addr => nr_regs_size )
        port map(
            s => dest_select,          w => LE,          f => load_enable_sig
        );

    mux_bus_A: muxnto1_bus
    generic map(
        n_addr => nr_regs_size,    bus_width => reg_width
    )

```

```

port map(
    s => A_select,      w => reg_file,      f => A
);

mux_bus_B: muxnto1_bus
generic map(
    n_addr => nr_regs_size,  bus_width => reg_width
)
port map(
    s => B_select,      w => reg_file,      f => B
);

povezava:
for i in 0 to nr_regs - 1 generate
    blok: shift_reg
        generic map ( reg_size => reg_width )
        port map(
            clk => clk,      nCLR => nRST,      sr_in => '0',      sl_in => '0',
s => reg_mode_sig( i ),      x => D,      Q => reg_file_array( i )
        );
end generate;

process ( reg_file_array )
    variable reg_file_col : std_logic_vector( reg_width - 1 downto 0 );
begin
    for i in 0 to nr_regs - 1 loop
        reg_file_col := reg_file_array( i );
        for j in 0 to reg_width - 1 loop
            reg_file( i,j ) <= reg_file_col( j );
        end loop;
    end loop;
end process;

process ( load_enable_sig )
begin
    for i in 0 to nr_regs - 1 loop
        reg_mode_sig( i ) <= ( others => load_enable_sig( i ) );
    end loop;
end process;

```

```
end NDV;
```

```

-- *****
-- **** STUDENT: 64210386
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
use ieee.math_real.all;

entity reg_file is
    generic(
        nr_regs          : natural := 4;
        reg_width        : natural := 8 );
    PORT ( clk,          -- clock input
        LE               : in std_logic;    -- Load enable input      ( active '1' )
        nRST             : in std_logic;    -- reset input                ( active '0' )
        dest_select,     -- register number destination select input
        A_select,        -- A, B bus destination select input
        B_select         : in  std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        D               : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus input
        A, B            : out std_logic_vector( reg_width - 1 downto 0 )    -- A, B bus output
    );
end reg_file;

architecture NDV of reg_file is

    component dmuxnto1 IS
        generic(
            n_addr: natural := 2 );
        PORT (
            s : in      std_logic_vector( n_addr - 1 downto 0 );
            w : in      STD_LOGIC;
            f : OUT     std_logic_vector( 2*n_addr - 1 downto 0 )
        );
    END component;

    component muxnto1_bus IS
        generic(
            n_addr      :      INTEGER := 2;
            bus_width   :      INTEGER := 8 );
        PORT (
            s           :      IN      std_logic_vector( n_addr - 1 downto 0 );

```

```

        w          :      IN          muxnto1_bus_type( 2**n_addr - 1 DOWNT0 0, bus_width - 1
DOWNT0 0 );
        f          :      OUT      std_logic_vector( bus_width - 1 downto 0 )
    );
END component;

component shift_reg is
    generic( reg_size: natural := 4 );
    PORT ( clk, nCLR, sr_in, sl_in : IN std_logic;
        s : in std_logic_vector( 1 downto 0 );
        x : in std_logic_vector( reg_size - 1 downto 0 );
        Q : out std_logic_vector( reg_size - 1 downto 0 )
    );
end component;

constant    nr_regs_size : integer := integer( ceil( log2( real( nr_regs - 1 ) ) ) );
signal      load_enable_sig      : std_logic_vector( nr_regs - 1 DOWNT0 0 ) := ( others => '0' );
SIGNAL      reg_file              : muxnto1_bus_type( nr_regs - 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others
=> ( others => '0' ) );

type reg_file_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
signal      reg_file_array : reg_file_array_type := ( others => ( others => '0' ) );

type reg_mode_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( 1 downto 0 );
signal      reg_mode_sig      : reg_mode_array_type := ( others => ( others => '0' ) );

begin

    demux: dmuxnto1
    generic map ( n_addr => nr_regs_size )
    port map(
        s => dest_select,          w => LE,          f => load_enable_sig
    );

    mux1: muxnto1_bus
    generic map( n_addr => nr_regs_size, bus_width => reg_width )
    port map (
        s => A_select,          w => reg_file,          f => A
    );

```

```

mux2: muxnto1_bus
generic map( n_addr => nr_regs_size, bus_width => reg_width )
port map (
    s => B_select,          w => reg_file,          f => B
);

registers: for i in 0 to nr_regs - 1 generate
    reg: shift_reg
        generic map( reg_size => reg_width )
        port map (
            clk => clk,          nCLR => nRST,          sr_in => '0',          sl_in => '0',
s => reg_mode_sig( i ),          x => D,          Q => reg_file_array( i )
        );
    end generate;

process ( reg_file_array ) is
    variable reg_file_col : std_logic_vector( reg_width - 1 downto 0 );
begin
    for i in 0 to nr_regs - 1 loop
        reg_file_col := reg_file_array( i );
        for j in 0 to reg_width-1 loop
            reg_file( i, j ) <= reg_file_col( j );
        end loop;
    end loop;
end process;

process ( load_enable_sig ) is
begin
    for i in 0 to nr_regs-1 loop
        reg_mode_sig( i ) <= load_enable_sig( i ) & load_enable_sig( i );
    end loop;
end process;

end NDV;

```



```

-- *****
-- **** STUDENT: 64210445
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
use ieee.math_real.all;

entity reg_file is
    generic(
        nr_regs          : natural := 4;
        reg_width        : natural := 8 );
    PORT ( clk,          -- clock input
        LE               : in std_logic;    -- Load enable input      ( active '1' )
        nRST             : in std_logic;    -- reset input                ( active '0' )
        dest_select,     -- register number destination select input
        A_select,        -- A, B bus destination select input
        B_select         : in  std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        D               : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus input
        A, B            : out std_logic_vector( reg_width - 1 downto 0 )    -- A, B bus output
    );
end reg_file;

architecture NDV of reg_file is

    constant nr_regs_size : integer := integer( ceil( log2( real( nr_regs - 1 ) ) ) );
    signal    load_enable_sig : std_logic_vector( nr_regs - 1 DOWNT0 0 ) := ( others => '0' );
    SIGNAL     reg_file      : muxnto1_bus_type( nr_regs - 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others
=> ( others => '0' ) );

    type reg_file_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
    signal    reg_file_array : reg_file_array_type := ( others => ( others => '0' ) );

    type reg_mode_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( 1 downto 0 );
    signal    reg_mode_sig   : reg_mode_array_type := ( others => ( others => '0' ) );

begin

```

```

dmux: dmuxnto1
generic map ( n_addr => nr_regs_size )
port map( w => LE,          s => dest_select,          f => load_enable_sig );

amux: muxnto1_bus
generic map ( n_addr => nr_regs_size, bus_width => reg_width )
port map( s => A_select,          w => reg_file,          f => A );

bmux: muxnto1_bus
generic map ( n_addr => nr_regs_size, bus_width => reg_width )
port map( s => B_select,          w => reg_file,          f => B );

regs: for i in 0 to nr_regs - 1 generate
    reg: shift_reg
        generic map ( reg_size => reg_width )
        port map( s => reg_mode_sig( i ),          x => D,          Q => reg_file_array( i ),          clk =>
clk,          nCLR => nRST,          sr_in => '0',          sl_in => '0' );
    end generate;

process ( reg_file_array ) is
    variable reg_file_col : std_logic_vector( reg_width - 1 downto 0 );
begin

    for idx in 0 to nr_regs-1 loop
        reg_file_col := reg_file_array( idx );
        for jdx in 0 to reg_width-1 loop
            reg_file( idx, jdx )      <= reg_file_col( jdx );
        end loop;
    end loop;

end process;

process ( load_enable_sig ) is
begin

    for i in 0 to nr_regs-1 loop
        reg_mode_sig( i )  <= load_enable_sig( i ) & load_enable_sig( i );
    end loop;

end process;

```

```
end NDV;
```

```

-- *****
-- **** STUDENT: 64210455
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
use ieee.math_real.all;

entity reg_file is
  generic(
    nr_regs : natural := 4;
    reg_width : natural := 8
  );
  PORT (
    clk : in std_logic;      -- clock input
    LE : in std_logic; -- Load enable input ( active '1' )
    nRST : in std_logic;    -- reset input ( active '0' )
    dest_select : in std_logic_vector( integer( ceil( log2( real( nr_regs ) ) ) ) - 1 downto 0 ); -- register
    destination_select
    A_select : in std_logic_vector( integer( ceil( log2( real( nr_regs ) ) ) ) - 1 downto 0 ); -- bus A select
    B_select : in std_logic_vector( integer( ceil( log2( real( nr_regs ) ) ) ) - 1 downto 0 ); -- bus B select
    D : in std_logic_vector( reg_width - 1 downto 0 ); -- data input bus
    A, B : out std_logic_vector( reg_width - 1 downto 0 ) -- output buses A and B
  );
end reg_file;

architecture behavior of reg_file is

  COMPONENT dmuxnto1 IS
    generic( n_addr: natural := 2 );
    PORT (
      s : in std_logic_vector( n_addr - 1 downto 0 );
      w : in std_logic;
      f : out std_logic_vector( 2*n_addr - 1 downto 0 )
    );
  END COMPONENT;

```

```

COMPONENT muxnto1_bus IS
generic(
n_addr : INTEGER := 2;
bus_width: INTEGER := 8
);
PORT (
s : IN std_logic_vector( n_addr - 1 downto 0 );
w : IN muxnto1_bus_type( 2*n_addr - 1 DOWNTO 0, bus_width - 1 DOWNTO 0 );
f : OUT std_logic_vector( bus_width - 1 downto 0 )
);
END COMPONENT;

COMPONENT shift_reg IS
generic( reg_size : natural := 4 );
PORT (
clk, nCLR, sr_in, sl_in : IN std_logic;
s : in std_logic_vector( 1 downto 0 );
x : in std_logic_vector( reg_size - 1 downto 0 );
Q : out std_logic_vector( reg_size - 1 downto 0 )
);
END COMPONENT;

constant    nr_regs_size : integer := integer( ceil( log2( real( nr_regs ) ) ) );
signal      load_enable_sig : std_logic_vector( nr_regs - 1 downto 0 ) := ( others => '0' );
signal      reg_file : muxnto1_bus_type( nr_regs - 1 downto 0, reg_width - 1 downto 0 ) := ( others => (
others => '0' ) );

type reg_file_2d_array is array ( nr_regs - 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
signal      reg_array : reg_file_2d_array := ( others => ( others => '0' ) );

type reg_mode_array is array ( nr_regs - 1 downto 0 ) of std_logic_vector( 1 downto 0 );
signal      reg_mode_sig : reg_mode_array := ( others => ( others => '0' ) );

begin

demux: dmuxnto1
generic map ( n_addr => nr_regs_size )
port map (
w => LE,
s => dest_select,

```

```

f => load_enable_sig
);

a_mux: muxnto1_bus
generic map ( n_addr => nr_regs_size, bus_width => reg_width )
port map (
s => A_select,
w => reg_file,
f => A
);

b_mux: muxnto1_bus
generic map ( n_addr => nr_regs_size, bus_width => reg_width )
port map (
s => B_select,
w => reg_file,
f => B
);

registers: for i in 0 to nr_regs - 1 generate
reg_unit: shift_reg
generic map ( reg_size => reg_width )
port map (
clk => clk,
nCLR => nRST,
s => reg_mode_sig( i ),
x => D,
Q => reg_array( i ),
sr_in => '0',
sl_in => '0'
);
end generate;

process ( reg_array ) is
variable temp_col : std_logic_vector( reg_width - 1 downto 0 );
begin
for i in 0 to nr_regs - 1 loop
temp_col := reg_array( i );
for j in 0 to reg_width - 1 loop
reg_file( i, j ) <= temp_col( j );

```

```
end loop;  
end loop;  
end process;  
  
process ( load_enable_sig ) is  
begin  
for i in 0 to nr_regs - 1 loop  
reg_mode_sig( i ) <= load_enable_sig( i ) & load_enable_sig( i );  
end loop;  
end process;  
  
end behavior;
```

```

-- *****
-- **** STUDENT: 64210457
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
use ieee.math_real.all;

entity reg_file is
    generic(
        nr_regs          : natural := 4;
        reg_width        : natural := 8 );
    PORT ( clk,          -- clock input
        LE               : in std_logic;    -- Load enable input      ( active '1' )
        nRST             : in std_logic;    -- reset input                ( active '0' )
        dest_select,     -- register number destination select input
        A_select,        -- A, B bus destination select input
        B_select         : in  std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        D               : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus input
        A, B            : out std_logic_vector( reg_width - 1 downto 0 )    -- A, B bus output
    );
end reg_file;

architecture NDV of reg_file is

    COMPONENT dmuxnto1 IS
        generic(
            n_addr: natural := 2 );
        PORT (
            s : in      std_logic_vector( n_addr - 1 downto 0 );
            w : in      STD_LOGIC;
            f : OUT     std_logic_vector( 2*n_addr - 1 downto 0 )
        );
    END COMPONENT;

    COMPONENT muxnto1_bus IS
        generic(
            n_addr      :      INTEGER := 2;
            bus_width   :      INTEGER := 8 );
        PORT (
            s           :      IN      std_logic_vector( n_addr - 1 downto 0 );

```



```

        w          :      IN          muxnto1_bus_type( 2**n_addr - 1 DOWNT0 0, bus_width - 1
DOWNT0 0 );
        f          :      OUT      std_logic_vector( bus_width - 1 downto 0 )
    );
END COMPONENT;

COMPONENT shift_reg is
    generic( reg_size: natural := 4 );
    PORT ( clk, nCLR, sr_in, sl_in : IN std_logic;
        s : in std_logic_vector( 1 downto 0 );
        x : in std_logic_vector( reg_size - 1 downto 0 );
        Q : out std_logic_vector( reg_size - 1 downto 0 )
    );
end COMPONENT;

constant    nr_regs_size : integer := integer( ceil( log2( real( nr_regs - 1 ) ) ) );
signal      load_enable_sig      : std_logic_vector( nr_regs - 1 DOWNT0 0 ) := ( others => '0' );
SIGNAL      reg_file              : muxnto1_bus_type( nr_regs - 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others
=> ( others => '0' ) );

type reg_file_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
signal      reg_file_array : reg_file_array_type := ( others => ( others => '0' ) );

type reg_mode_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( 1 downto 0 );
signal      reg_mode_sig      : reg_mode_array_type := ( others => ( others => '0' ) );

begin

    dmux: dmuxnto1
    generic map ( n_addr => nr_regs_size )
    port map(
        w => LE, s => dest_select, f => load_enable_sig
    );

    amux: muxnto1_bus
    generic map ( n_addr => nr_regs_size, bus_width => reg_width )
    port map (
        s => A_select, w => reg_file, f => A
    );

```

```

bmux: muxnto1_bus
generic map ( n_addr => nr_regs_size, bus_width => reg_width )
port map (
    s => B_select, w => reg_file, f => B
);

regs: for i in 0 to nr_regs - 1 generate
    reg: shift_reg
        generic map ( reg_size => reg_width )
        port map (
            s => reg_mode_sig( i ), x => D, Q => reg_file_array( i ),      clk => clk, nCLR => nRST, sr_in
=> '0', sl_in => '0'
        );
    end generate;

process ( reg_file_array ) is
    variable reg_file_col : std_logic_vector( reg_width - 1 downto 0 );
begin
    for idx in 0 to nr_regs-1 loop
        reg_file_col := reg_file_array( idx );
        for jdx in 0 to reg_width-1 loop
            reg_file( idx, jdx )      <= reg_file_col( jdx );
        end loop;
    end loop;
end process;

process ( load_enable_sig ) is
begin
    for i in 0 to nr_regs-1 loop
        reg_mode_sig( i )  <= load_enable_sig( i ) & load_enable_sig( i );
    end loop;
end process;

end NDV;

```

```

-- *****
-- **** STUDENT: 64240430
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
use ieee.math_real.all;

entity reg_file is
    generic(
        nr_regs          : natural := 4;
        reg_width        : natural := 8 );
    PORT ( clk,          -- clock input
        LE               : in std_logic;    -- Load enable input      ( active '1' )
        nRST             : in std_logic;    -- reset input                ( active '0' )
        dest_select,     -- register number destination select input
        A_select,        -- A, B bus destination select input
        B_select         : in  std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
        D               : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus input
        A, B            : out std_logic_vector( reg_width - 1 downto 0 )    -- A, B bus output
    );
end reg_file;

architecture NDV of reg_file is

    component dmuxnto1
        generic ( n_addr : natural := 2 );
        port(
            s : in std_logic_vector( n_addr - 1 downto 0 );
            w : in std_logic;
            f : out std_logic_vector( 2*n_addr - 1 downto 0 )
        );
    end component;

    component muxnto1_bus
        generic(
            n_addr : integer := 2;
            bus_width : integer := 8

```

```

    );
    port(
        s : in std_logic_vector( n_addr - 1 downto 0 );
        w : in muxnto1_bus_type( 2**n_addr - 1 downto 0, bus_width - 1 downto 0 );
        f : out std_logic_vector( bus_width - 1 downto 0 )
    );
end component;

component shift_reg
    generic ( reg_size : natural := 4 );
    port(
        clk, nCLR, sr_in, sl_in : in std_logic;
        s : in std_logic_vector( 1 downto 0 );
        x : in std_logic_vector( reg_size - 1 downto 0 );
        Q : out std_logic_vector( reg_size - 1 downto 0 )
    );
end component;

constant    nr_regs_size : integer := integer( ceil( log2( real( nr_regs - 1 ) ) ) );
signal      load_enable_sig : std_logic_vector( nr_regs - 1 DOWNT0 0 ) := ( others => '0' );
SIGNAL      reg_file : muxnto1_bus_type( nr_regs - 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others
=> ( others => '0' ) );

type reg_file_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
signal      reg_file_array : reg_file_array_type := ( others => ( others => '0' ) );

type reg_mode_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( 1 downto 0 );
signal      reg_mode_sig : reg_mode_array_type := ( others => ( others => '0' ) );

begin
    -- program
    dmux: dmuxnto1
        generic map ( n_addr => nr_regs_size )
        port map ( w => LE, s => dest_select, f => load_enable_sig
        );

    muxnto1_bus_A: muxnto1_bus
        generic map ( n_addr => nr_regs_size, bus_width => reg_width
        )
        port map ( s => A_select, w => reg_file, f => A

```

```

    );

muxnto1_bus_B: muxnto1_bus
    generic map ( n_addr => nr_regs_size,          bus_width => reg_width
    )
    port map ( s => B_select,          w => reg_file,          f => B
    );

pov1:
for i in 0 to nr_regs - 1 generate
    reg: shift_reg
        generic map ( reg_size => reg_width )
        port map ( clk => clk,          nCLR => nRST,          sr_in => '0',          sl_in =>
'0',          s => reg_mode_sig( i ),          x => D,          Q => reg_file_array( i )
        );
end generate;

process ( reg_file_array )
    variable reg_file_col : std_logic_vector( reg_width - 1 downto 0 );
begin
    for i in 0 to nr_regs - 1 loop
        reg_file_col := reg_file_array( i );
        for j in 0 to reg_width - 1 loop
            reg_file( i, j ) <= reg_file_col( j );
        end loop;
    end loop;
end process;

process ( load_enable_sig )
begin
    for i in 0 to nr_regs - 1 loop
        reg_mode_sig( i ) <= ( others => load_enable_sig( i ) );
    end loop;
end process;

end NDV;

```

```

-- *****
-- **** PREDLOGA VAJE
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE work.reg_file_functions.all;
use ieee.math_real.all;

entity reg_file is
    generic( nr_regs          : natural := 4;
             reg_width       : natural := 8 );
    PORT ( clk,  -- clock input
          LE      : in std_logic;    -- Load enable input      ( active '1' )
          nRST    : in std_logic;    -- reset input          ( active '0' )
          dest_select, -- register number destination select input
          A_select,   -- A, B bus destination select input
          B_select    : in  std_logic_vector( sizeof( nr_regs - 1 ) - 1 downto 0 );
          D            : in  std_logic_vector( reg_width - 1 downto 0 );    -- data input bus input
          A, B        : out std_logic_vector( reg_width - 1 downto 0 )    -- A, B bus output
          );
end reg_file;

architecture NDV of reg_file is

    constant nr_regs_size : integer := integer( ceil( log2( real( nr_regs - 1 ) ) ) );
    signal    load_enable_sig : std_logic_vector( nr_regs - 1 DOWNT0 0 ) := ( others => '0' );
    SIGNAL     reg_file       : muxnto1_bus_type( nr_regs - 1 DOWNT0 0, reg_width - 1 DOWNT0 0 ) := ( others
=> ( others => '0' ) );

    type reg_file_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( reg_width - 1 downto 0 );
    signal    reg_file_array : reg_file_array_type := ( others => ( others => '0' ) );

    type reg_mode_array_type is array ( nr_regs - 1 downto 0 ) of std_logic_vector( 1 downto 0 );
    signal    reg_mode_sig   : reg_mode_array_type := ( others => ( others => '0' ) );

begin

```

```

REG_DMUX: dmuxnto1
    generic map (      n_addr => nr_regs_size )
    port map (      s => dest_select,      w => LE,      f => load_enable_sig
    );

A_BUS_MUX: muxnto1_bus
    generic map (      n_addr=> nr_regs_size,      bus_width => reg_width )
    PORT MAP (      s => A_select,      w => reg_file,      f => A
    );

B_BUS_MUX: muxnto1_bus
    generic map (      n_addr=> nr_regs_size,      bus_width => reg_width )
    PORT MAP (      s => B_select,      w => reg_file,      f => B
    );

reg_file_process: PROCESS( reg_file_array, reg_file )
variable reg_file_col : std_logic_vector( reg_width - 1 downto 0 );
BEGIN
    FOR i IN 0 TO nr_regs - 1 LOOP
        reg_file_col := reg_file_array( i );
        FOR j IN 0 TO reg_width - 1 LOOP
            reg_file( i, j ) <= reg_file_col( j );
        END LOOP;

    END LOOP;
END PROCESS;

reg_mode_process: PROCESS( load_enable_sig )
BEGIN
    FOR r in nr_regs - 1 downto 0 LOOP
-- S0 S1      |      load_enable_sig( r )      Operation
-- 1 1      |      1      : Load x => Q
-- 0 0      |      0      : Hold contents
        reg_mode_sig( r ) <= ( others => load_enable_sig( r ) );
    END LOOP;
END PROCESS;

-- Loop through all registers in register file
regloop : for r in nr_regs - 1 downto 0 generate

```

```

    Regi: shift_reg    generic map ( reg_size => reg_width )
                        PORT map (   clk => clk,          nCLR => nRST,          sr_in => '0',          sl_in => '0',
s => reg_mode_sig( r ),          x => D,          Q => reg_file_array( r )
                        );
end generate;

end NDV;

```


