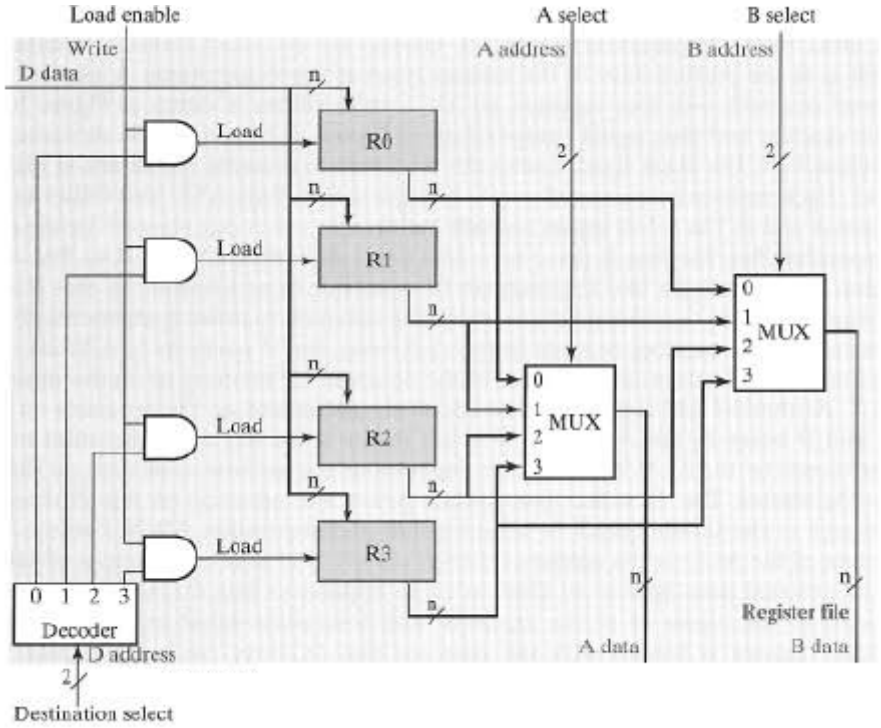
Polje registrov (ang. register file)

V VHDL programirajte arhitekturo splošne strukture polja registrov, ki omogoča vpisovanje polja registrov enake velikosti. Spodnja slika prikazuje organizacijo povezovanja komponent. Vhod v polje registrov je podatkovno vodilo (D), ki je priključeno na vhode za vzporedno nalaganje registrov R0 do R3. Vpisovanje v registre je izvedeno s signalom (LE) (ang. load enable). Vsebina se v registre vpiše, ko je LE='1'. Izbira registra, kamor se podatek z vodila (D) vpiše je izvedena z demultiplekserjem. Demultiplekserju nastavimo ciljni naslov (ang. destination select). Z izbiralnikoma vodil (ang. bus multiplexer) A in B določamo kateri register se pojavi na izhodnih vodilih A in B.



Slika 1: Izvedba splošnega polja štirih registrov (R0−R3).

Število registrov je (**nr_regs**), širina posameznega registra je (**reg_width**). Entiteta izdelane strukture **reg_file** ima priključke:

```
--    @Component name: reg_file
--    @Parameters:
--    argument 1: število registrov
--    argument 2: širina registra
--    argument 3: signal ure
--    argument 4: omogočanje vpisa (ang. load enable ), (aktiven '1')
--    argument 5: asinhrono brisanje vsebine registrov (ang. reset), (aktiven '0')
--    argument 6: izbira ciljnega registra (ang. destination)
--    argument 7: izbira registra na izhodnem vodilu A
--    argument 8: izbira registra na izhodnem vodilu B
--    argument 9: vhodno vodilo D
--    argument 10: izhodno vodilo A
--    argument 11: izhodno vodilo B
--    @Description:
--    Splošno polje registrov (ang. register file)
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;
```

Naloge:

1. Ustvarite VHDL datoteko **`reg_file_functions.vhd`** v kateri boste znotraj VHDL paketa (**PACKAGE**) programirali funkcije, potrebne za izvedbo sinteze splošnega polja registrov. V datoteki se nahaja definicija vhodnega tipa splošnega dvodimenzionalnega polja (**`muxnto1_bus_type`**). Podani so prototipi funkcij z vhodnimi parametri in opisom delovanja:

```
PACKAGE reg_file_functions IS
--    @Type name: splošni dvodimenzionalni tip za izbiralnik vodil (ang. bus multiplexer)
--    @Parameters:
--    argument 1: x dimenzija polja
--    argument 2: y dimenzija polja
--    @Description:
--    definicija tipa splošnega dvodimenzionalnega polja (x, y) bitov tipa STD_LOGIC
      Type muxnto1_bus_type is array (natural range <>, natural range <>) of STD_LOGIC;

--    @Type name: sizeof
-- @Parameters:
--    argument 1: x dimenzija polja
--    argument 2: y dimenzija polja
--    @Description:
--    definicija tipa splošnega dvodimenzionalnega polja (x, y) bitov tipa STD_LOGIC
      FUNCTION sizeof (a: NATURAL) RETURN NATURAL;

--    @Component name: dmuxnto1
--    @Parameters:
--    argument 1: število naslovov demultiplekserja
--    argument 2: podatkovni vhod demultiplekserja
--    argument 3: izhod demultiplekserja
--    @Description:
--    Splošni 1-na-2^n_addr demultiplekser
      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 name: dmuxnto1
--    @Parameters:
--    argument 1: velikost registra
--    argument 2: signal ure
--    argument 3: signal asinhronega brisanje (aktiven '0')
--    argument 4: serijski vhod za pomikanje desno
--    argument 5: serijski vhod za pomikanje levo
--    argument 6: vhod za vzporedno nalaganje
--    argument 6: izhod splošnega univerzalnega registra
--    @Description:
--    Splošni univerzalni register:
      -- S0 S1
      -- 1 1      : Vzporedno nalaganje x => Q
      -- 1 0      : Pomikanje levo  (v smeri od LSB do MSB)
      -- 0 1      : Pomikanje desno (v smeri od MSB do LSB)
      -- 0 0      : Držanje vsebine
      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 name: muxnto1
--    @Parameters:
--    argument 1: število naslovov multiplekserja
--    argument 2: podatkovni vhodi multiplekserja
--    argument 3: izhod demultiplekserja
--    @Description:
--    Splošni 1-na-2^n_addr multiplekser
      COMPONENT muxnto1 IS
            generic( n_addr: natural := 2 );
            PORT (s     : in  std_logic_vector(n_addr - 1 downto 0);
                  w     : in  std_logic_vector(2**n_addr - 1 downto 0);
                  f     : OUT STD_LOGIC
            );
      END COMPONENT;
```

```
--    @Component name: muxnto1_bus
--    @Parameters:
--    argument 1: število naslovov
--    argument 2: širina vodila
--    argument 3: izbiralni vhod
--    argument 3: vhodno polje vodil izbiralnika
--    argument 4: izbrano izhodno vodilo izbiralnika
--    @Description:
--    Splošni izbiralnik vodil (ang. bus multiplexer)
      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;

END reg_file_functions;

PACKAGE BODY reg_file_functions IS

--    @Function name: sizeof
      -- @Parameters:
      --    a: input number
      --    @Return:
      --    Number of bits required to encode a binary input number a
      FUNCTION sizeof (a: NATURAL) RETURN NATURAL IS
            VARIABLE aggregate      : NATURAL := a;
            VARIABLE return_val     : NATURAL := 0;
      BEGIN
            compute_sizeof:
            FOR i IN a DOWNTO 0 LOOP
                  IF aggregate > 0 THEN
                        return_val := return_val + 1; --increment number of encoding bits
                  END IF;
                  aggregate := aggregate / 2;   -- divide by base of 2
            END LOOP;
            RETURN return_val;
      END sizeof;

END reg_file_functions;
```

2. Ustvarite datoteko **dmuxnto1.vhd** v kateri programirajte entiteto in arhitekturo splošne strukture demultiplekserja (**1/n_addr**), ki glede na vrednost naslovnega vhoda (**s**) na izhodu (**f**) postavi ustrezno mesto (2^s) na vrednost podatkovnega vhoda (**w**). Ime entitete mora biti: **dmuxnto1**. Podana je entiteta strukture:

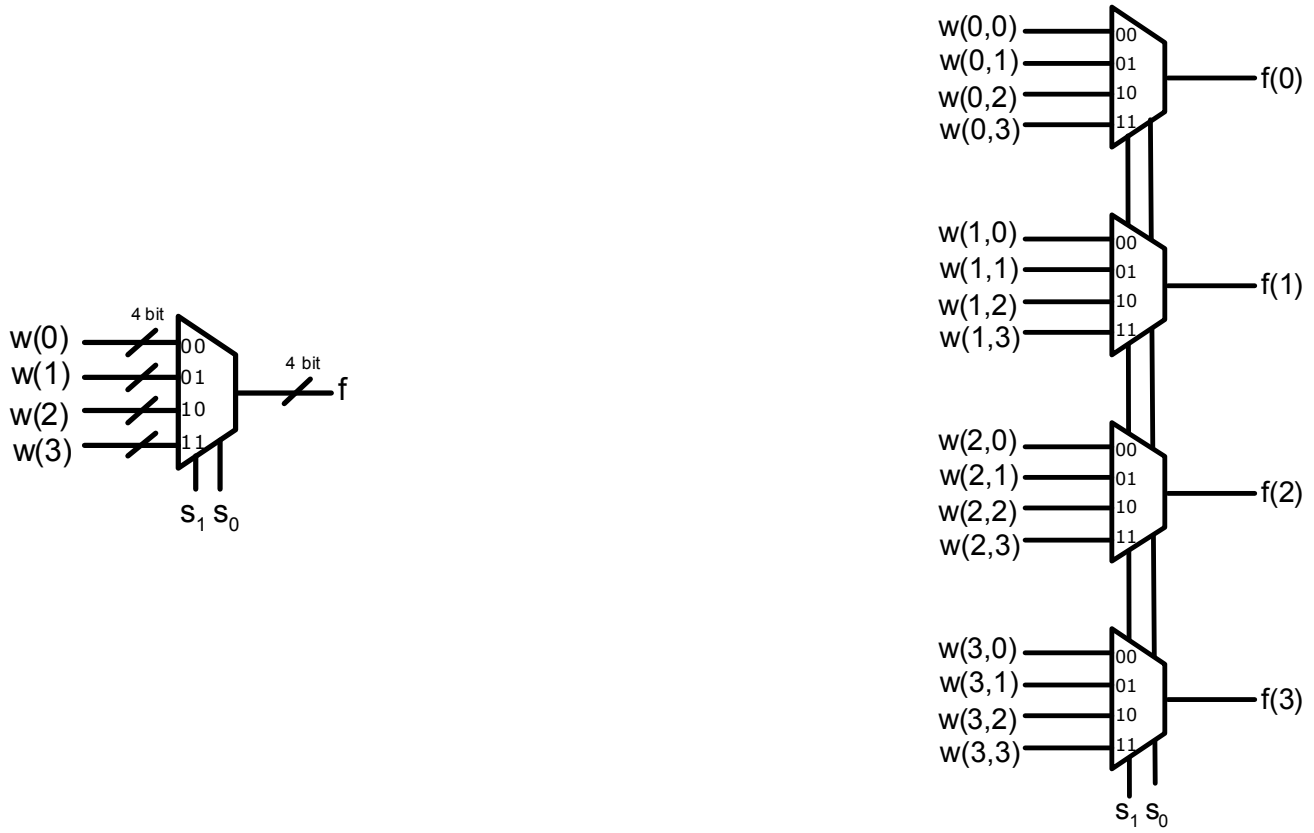
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.numeric_std.all;

ENTITY 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 dmuxnto1;
```

Zaradi standardizacije VHDL uporabljajte standardno knjižnico (**ieee.numeric_std.all**) in ne (**ieee.arith.all**). Programirajte procesni stavek, v katerem izhod f postavite na '1' z uporabo funkcije za pretvorbo vhoda (**s**) v neko celoštevilsko (**integer**) spremenljivko (**addr**):

```
addr := to_integer(unsigned(s));
```

Definirajte vmesni signal (**f_sig**), ki mu priredite številko izhoda (**2**addr**) z uporabo funkcije za pretvorbo celoštevilske spremenljivke v tip (**std_logic_vector**). Dobljeni signal kombinacijsko priredite izhodu (**f**) znotraj izbiralnika: Če je podatkovni vhod postavljen (**w = '1'**), potem izhod (**f**) postane enak (**f_sig**), sicer je izhod splošna konstanta nič (**zeroes**). Splošno konstanto nič definirajte podobno kot pri prejšnjih nalogah.

3. Ustvarite datoteko **`muxnto1_bus.vhd`** v kateri programirajte entiteto in arhitekturo splošne strukture izbiralnika vodil (ang. bus multiplexer), ki izbira med (2^{n_addr}) vodili glede na vrednost naslovnega vhoda (**`s`**). Na izhodu (**`f`**) postavi izbrano vodilo ($2^{s}-1$). Velikost vodila je (**`bus_width`**). Za ilustracijo delovanja komponente si oglejte nalogo 5 v zbirki nalog na domači strani predmeta. Slika 2 prikazuje povezovanje štirih dvonaslovnih izbiralnikov, ki tvorijo izbiralnik vodil, ki lahko izbira med štirimi 4−bitnimi vodili.



Slika 2: Izvedba 4 naslovnega izbiralnika 4 bitnih vodil (ang. bus multiplexer).

Ime entitete mora biti **muxnto1_bus**. Podana je entiteta strukture:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.numeric_std.all;
USE work.reg_file_functions.all;

ENTITY 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 muxnto1_bus;
```

Vhodno polje vodil (**w**) je splošno dvodimenzionalno polje elementov tipa (**std_logic**), definirano v paketni datoteki (**reg_file_functions**). To polje moramo uporabljati, saj v entiteti uporaba operatorja (**array**) ni dovoljena. Izhod izbiralnika vodil je tipa (**std_logic_vector**), zato morate najprej splošno dvodimenzionalno polje tipa (**muxnto1_bus_type**), pretvoriti v polje (**array**) vektorjev tipa (**std_logic_vector**). To storite tako, da definirate pretvorjeni signal:

```
type mux_addr_type is array (bus_width - 1 downto 0) of std_logic_vector(2**n_addr - 1 downto 0);
signal mux_addr : mux_addr_type := (others =>(others => '0'));
```

Samo pretvorbo izvajate znotraj procesnega stavka, v dveh gnezdenih (**FOR ... LOOP**) zankah. Prvi indeks zanke teče po stolpcih, drug po vrsticah polja (**w**). Za hranjenje stolpca dvodimenzionalnega polja uvedite spremenljivko:

```
variable mux_addr_col : std_logic_vector(2**n_addr - 1 downto 0);
```

Dobljeni signal (**mux_addr**) povežite z uporabo povezovalnega stavka nad komponentami splošnega izbiralnika ($2^{n\_addr}/1$), ki smo ga izdelali v eni prejšnjih nalog (**muxnto1.vhd**). Datoteka (**muxnto1.vhd**) je priložena predlogi domače naloge.

Sestavite datoteko testnih vrednosti (**muxnto1_bus.tbw**) in s simulacijo preverite pravilnost delovanja izbiralnika vodil za vse vhodne kombinacije naslova (**s**).

4. Ustvarite datoteko **reg_file.vhd** v kateri programirajte entiteto in arhitekturo splošnega polja registrov. Ime entitete mora biti: **reg_file**. Podana je entiteta strukture:

```
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;
```

Definirajte konstanto (**nr_regs_size**), ki predstavlja število bitov, potrebnih za zapis števila (**nr_regs**). Definirajte signal (**reg_file**) dvodimenzionalnega polja elementov polja (**std_logic**) in signala (**reg_file_array, reg_mode_sig**) polja (**array**) vektorjev tipa (**std_logic_vector**). Signal (**reg_file**) služi povezovanju na vhode splošnih izbiralnikov vodil po sliki 1, na signal (**reg_file_array**) se povežejo izhodi univerzalnih registrov (**Q**). Signal (**reg_file_array**) nato pretvorimo *obratno* v signal (**reg_file**) z uporabo gnezdenih (**FOR ... LOOP**) zank. Signal (**reg_mode_sig**) povežete na krmilne vhode univerzalnih registrov (**S**).

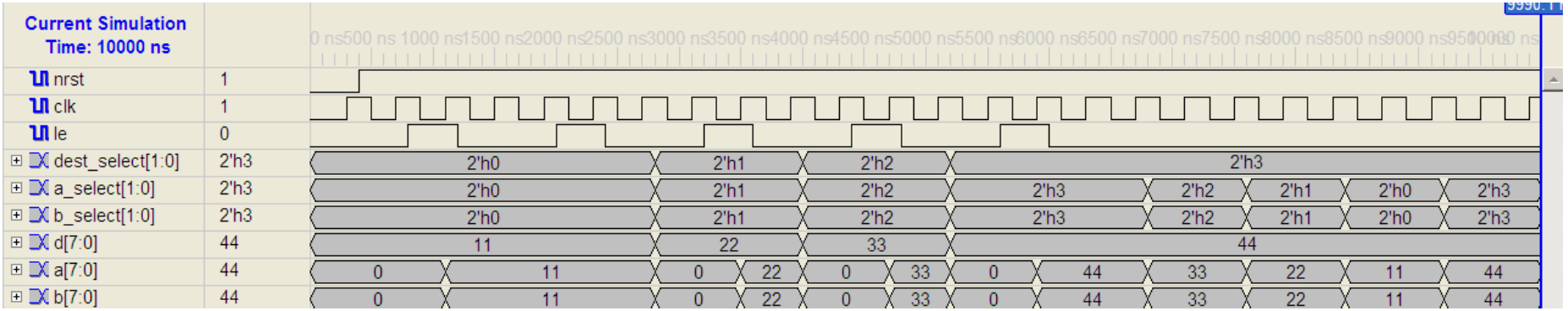
Pri povezovanju splošnega demultiplekserja (**dmuxnto1**), povežete signal (**LE**) na podatkovni vhod (**W**). Krmilni vhod (**S**) povežete na izbiro ciljnega registra (**dest_select**), izhod demultiplekserja (**F**) je povezan na vektorski signal (**load_enable_sig**).

Pri povezovanju splošnega izbiralnika vodil (**muxnto1_bus**) za vodilo A (**A_BUS_MUX**), povežete signal (**LE**) na podatkovni vhod (**W**). Krmilni vhod (**S**) povežete na izbiro ciljnega registra (**A_select**), polje (**W**) na signal (**reg_file**), izhod demultiplekserja (**F**) je povezan na izhodno vodilo (**A**). Podobna logika povezovanja velja za splošni izbiralnik vodil (**muxnto1_bus**) za vodilo B (**B_BUS_MUX**).

Pri povezovanju univerzalnih registrov (**shift_reg**), povežete v (**for...generate**) zanki signal (**LE**) na podatkovni vhod (**W**). Krmilni vhod (**S**) povežete na izbiro ciljnega registra (**reg_mode_sig**), vhod za vzporedno nalaganje (**X**) na vhodno vodilo (**D**), izhod registra (**Q**) je povezan na signal (**reg_file_array**). V

datoteki programirajte pretvorbo signala (**reg_file**) dvodimenzionalnega polja elementov polja (**std_logic**) v signal (**reg_file_array**) polja (**array**) vektorjev tipa (**std_logic_vector**) z uporabo procesnega stavka in gnezdenih (**FOR ... LOOP**) zank. Podobno kot pri prejšnji pretvorbi definirajte vmesno spremenljivko `variable reg_file_col : std_logic_vector(reg_width - 1 downto 0);`
v katero povežete vrednosti elementov posameznega stolpca signala (**reg_file_array**).
Definirajte poseben procesni stavek v katerem v (**FOR ... LOOP**) zanki povežete krmilni signal (**reg_mode_sig**) s signalom (**load_enable_sig**).

Za preverjanje uporabite priloženo VHDL datoteko testnih vrednosti (**reg_file_tb.vhd**) in s simulacijo preverite pravilnost delovanja seštevanja za opisane signale. Slika 3 prikazuje rezultat simulacije vpisa konstant (11, 22, 33, 44) v registre (R0, R1, R2, R3) po vrsti. Registri se ob ponastavitvi (**nRST = '0'**) zbrišejo, zato je njihova vsebina v simulaciji enaka (**x"00"**). Podatek se z vhodnega vodila (**D**) vpiše ob aktivnem robu signala ure, ko velja (**LE = '1'**).



Slika 3: Rezultat simulacije delovanje vpisa konstant (11, 22, 33, 44) v registre (R0, R1, R2, R3).