

-- **** STUDENT: 64000225.....	5
-- KOMENTARJI K OCENI NALOGE   -- Matej Možek: Iz simulacije sledi, da operacija POP ne deluje pravilno za prvovpisani podatek (oz. zadnji prebrani podatek), ker je v pogoju za izhodni 3-stanjski vmesnik postavljeno EMPTY='0'. Zastavica EMPTY se postavi takoj ko preberemo zadnji podatek iz sklada (ki pa je še vedno veljaven – ni enak Z). Pogoj za 3-stanjski vmesnik torej ni odvisen od EMPTY.....	5
-- **** STUDENT: 64190088.....	8
-- KOMENTARJI K OCENI NALOGE   -- Matej Možek: EMPTY zastavica ne deluje pravilno – po brisanju (nCLR='0') je sklad prazen. FULL zastavica ne deluje pravilno – ko je sklad poln, se postavi (do sem je ok), potem pa mora to vrednost ohranjati, dokler ni operacije pop (tu je napaka). POP operacija ne deluje pravilno – izhod sklada je vedno 0. ....	8
Opozorila sintetizatorja: .....	8
WARNING:HDLCompiler:746 - "lifo.vhd" Line 71: Range is empty (null range).....	8
Napisali ste: 0 downto (lifo_width - 1), verjetno ste mislili obratno: 0 to (lifo_width - 1). ....	8
Pri primerjavi std_logic_vector tipa z integer konstanto morate uporabiti unsigned: unsigned(lifo_ctr) = lifo_size. ....	8
-- **** STUDENT: 64200100.....	11
-- KOMENTARJI K OCENI NALOGE   -- Matej Možek: Iz simulacije sledi, da operacija POP ne deluje pravilno za prvovpisani podatek (oz. zadnji prebrani podatek), ker je v pogoju za izhodni 3-stanjski vmesnik postavljeno EMPTY='0'. Zastavica EMPTY se postavi takoj ko preberemo zadnji podatek iz sklada (ki pa je še vedno veljaven – ni enak Z). Pogoj za 3-stanjski vmesnik torej ni odvisen od EMPTY.....	11
-- ***** .....	11
-- **** STUDENT: 64200112.....	14
-- KOMENTARJI K OCENI NALOGE   -- Matej Možek: Ni pripomb.....	14
-- **** STUDENT: 64200163.....	17
-- KOMENTARJI K OCENI NALOGE   -- Matej Možek: Izhod sklada se mora pojaviti TAKOJ, ko se postavi POP operacija. Problem je v zapisu izhoda za data:           17	
data <= lifo_out when lifo_mode = "00010" or lifo_mode = "00011" else (others => 'Z'); .....	17
Iz zgornjega sledi, da lifo_out postane izhod, če sklad ni poln – dejansko je branje sklada čisto neodvisno od zastavice FULL (važno samo, da ni prazen). .....	17
Pravilno bi se glasilo: data <= lifo_out when (nEnable = '0' and POP = '1') else (others => 'Z'); .....	17
Sklad lahko beremo, če je omogočen in če je operacija POP. Če je sklad prazen (EMPTY), potem ostaja na izhodu zadnja vpisana vrednost (uporabnik pač lahko bere prazen sklad – ni sicer verjetno, ampak se zgodi). ....	17
-- **** STUDENT: 64200238.....	20
-- KOMENTARJI K OCENI NALOGE   -- Matej Možek: Iz simulacije sledi, da operacija POP ne deluje pravilno za prvovpisani podatek (oz. zadnji prebrani podatek), ker je v pogoju za izhodni 3-stanjski vmesnik postavljeno EMPTY='0'.	

Zastavica EMPTY se postavi takoj ko preberemo zadnji podatek iz sklada (ki pa je še vedno veljaven – ni enak Z). Pogoj za 3-stanjski vmesnik torej ni odvisen od EMPTY.....	20
-- ***** .....	20
-- ***** STUDENT: 64200288.....	24
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni rezultatov simulacije, so pa napake sintetizatorja: .....	24
ERROR:HDLCompiler:580 - "lifo.vhd" Line 126: Mismatch on label ; expected ideal.....	24
Ime arhitekture se mora ujemati na začetku in na koncu. Na začetku imate “improved”, na koncu imate “ideal”. ☺ .....	24
Po tej spremembi simulacija deluje pravilno. ....	24
-- ***** STUDENT: 64200296.....	28
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Iz simulacije sledi, da operacija POP ne deluje pravilno za prvovpisani podatek (oz. zadnji prebrani podatek), ker je v pogoju za izhodni 3-stanjski vmesnik postavljeno EMPTY='0'. Zastavica EMPTY se postavi takoj ko preberemo zadnji podatek iz sklada (ki pa je še vedno veljaven – ni enak Z). Pogoj za 3-stanjski vmesnik torej ni odvisen od EMPTY.....	28
-- ***** .....	28
-- ***** STUDENT: 64200385.....	31
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	31
-- ***** STUDENT: 64210113.....	34
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Uporabljate stare Synopsis knjižnice (STD_LOGIC_ARITH), česar v predlogah že ni kar nekaj let. Iz simulacije sledi, da operacija POP ne deluje pravilno za prvovpisani podatek (oz. zadnji prebrani podatek), ker je v pogoju za izhodni 3-stanjski vmesnik postavljeno EMPTY='0'. Zastavica EMPTY se postavi takoj ko preberemo zadnji podatek iz sklada (ki pa je še vedno veljaven – ni enak Z). Pogoj za 3-stanjski vmesnik torej ni odvisen od EMPTY.	34
-- ***** .....	34
-- ***** STUDENT: 64210132.....	37
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni rezultatov simulacije zaradi napak sintetizatorja: .....	37
ERROR:HDLCompiler:40 - "lifo.vhd" Line 76: dff is not a component .....	37
Pri poimenovanju neke komponente v povezovalnem stavku lahko uporabite katerokoli ime **RAZEN** imena te komponente: .....	37
Namesto: dff: tff port map( ... naredite d_ff: dff port map(.....	37
ERROR:HDLCompiler:69 - "lifo.vhd" Line 83: <full_sig> is not declared. ....	37
Uporabljate signal: lifo_mode <= PUSH & POP & FULL_sig & EMPTY_sig & nEnable; .....	37
Pri čemer imate deklarirana svoja signala: signal FULL_s, EMPTY_s : std_logic;.....	37
Po teh spremembah simulacija deluje pravilno. ....	37
Uporabljate stare Synopsis knjižnice (STD_LOGIC_ARITH), česar v predlogah že ni kar nekaj let. ....	37

-- **** STUDENT: 64210290.....	40
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb.....	40
-- **** STUDENT: 64210382.....	44
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Uporabljate stare Synopsis knjižnice (STD_LOGIC_ARITH), česar v predlogah že ni kar nekaj let. Iz simulacije sledi, da operacija POP ne deluje pravilno za prvovpisani podatek (oz. zadnji prebrani podatek), ker je v pogoju za izhodni 3-stanjski vmesnik postavljeno EMPTY='0'. Zastavica EMPTY se postavi takoj ko preberemo zadnji podatek iz sklada (ki pa je še vedno veljaven – ni enak Z). Pogoj za 3-stanjski vmesnik torej ni odvisen od EMPTY.	44
-- ***** .....	44
-- **** STUDENT: 64210384.....	47
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Iz simulacije sledi, da operacija POP ne deluje pravilno za prvovpisani podatek (oz. zadnji prebrani podatek), ker je v pogoju za izhodni 3-stanjski vmesnik postavljeno EMPTY='0'. Zastavica EMPTY se postavi takoj ko preberemo zadnji podatek iz sklada (ki pa je še vedno veljaven – ni enak Z). Pogoj za 3-stanjski vmesnik torej ni odvisen od EMPTY.....	47
-- ***** .....	47
-- **** STUDENT: 64210386.....	50
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Iz simulacije sledi, da operacija POP ne deluje pravilno za prvovpisani podatek (oz. zadnji prebrani podatek), ker je v pogoju za izhodni 3-stanjski vmesnik postavljeno EMPTY='0'. Zastavica EMPTY se postavi takoj ko preberemo zadnji podatek iz sklada (ki pa je še vedno veljaven – ni enak Z). Pogoj za 3-stanjski vmesnik torej ni odvisen od EMPTY.....	50
-- ***** .....	50
-- **** STUDENT: 64210445.....	53
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Iz simulacije sledi, da operacija POP ne deluje pravilno za prvovpisani podatek (oz. zadnji prebrani podatek), ker je v pogoju za izhodni 3-stanjski vmesnik postavljeno EMPTY='0'. Zastavica EMPTY se postavi takoj ko preberemo zadnji podatek iz sklada (ki pa je še vedno veljaven – ni enak Z). Pogoj za 3-stanjski vmesnik torej ni odvisen od EMPTY.....	53
-- ***** .....	53
-- **** STUDENT: 64210455.....	57
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Naloga ni programirana. ....	57
-- **** STUDENT: 64210457.....	59
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Uporabljate stare Synopsis knjižnice (STD_LOGIC_ARITH), česar v predlogah že ni kar nekaj let. Iz simulacije sledi, da operacija POP ne deluje pravilno za prvovpisani podatek (oz. zadnji prebrani podatek), ker je v pogoju za izhodni 3-stanjski vmesnik postavljeno EMPTY='0'. Zastavica EMPTY se postavi takoj	

ko preberemo zadnji podatek iz sklada (ki pa je še vedno veljaven – ni enak Z). Pogoji za 3-stanjski vmesnik torej ni odvisen od EMPTY.

59

-- *****	59
-- ***** STUDENT: 64240430	62
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Iz simulacije sledi, da operacija POP ne deluje pravilno za prvovpisani podatek (oz. zadnji prebrani podatek), ker je v pogoju za izhodni 3-stanjski vmesnik postavljeno EMPTY='0'. Zastavica EMPTY se postavi takoj ko preberemo zadnji podatek iz sklada (ki pa je še vedno veljaven – ni enak Z). Pogoji za 3-stanjski vmesnik torej ni odvisen od EMPTY	62
-- *****	62
-- ***** PREDLOGA VAJE	65
-- KOMENTARJI K OCENI NALOGE -- Matej Možek: Ni pripomb	65

```

-- *****
-- **** STUDENT: 64000225
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Iz simulacije sledi, da operacija POP ne deluje pravilno za prvovpisani podatek (oz. zadnji prebrani
podatek), ker je v pogoju za izhodni 3-stanjski vmesnik postavljeno EMPTY='0'. Zastavica EMPTY se postavi takoj ko
preberemo zadnji podatek iz sklada (ki pa je še vedno veljaven – ni enak Z). Pogoju za 3-stanjski vmesnik torej ni
odvisen od EMPTY.
-- *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use IEEE.math_real.ALL;

entity lifo is
    generic(
        lifo_width: natural := 4; -- velikost podatka sklada
        lifo_size: natural := 8 ); -- velikost sklada
    PORT (
        clk,      -- signal      ure
        nCLR,     -- signal      asinhronnega brisanja vsebine sklada ( aktiven '0' )
        nEnable,  -- signal      omogočanja sklada ( aktiven '0', sicer drži vsebino )
        PUSH,     -- operacija vpisa na sklad ( aktiven '1' )
        POP       : IN std_logic; -- operacija branja s sklada ( aktiven '1' )
        data      : inout std_logic_vector( lifo_width - 1 downto 0 ); -- tristanjski izhod sklada
        FULL,     -- izhod, ki postane '1', ko je sklad poln
        EMPTY     : OUT std_logic      -- izhod, ki postane '0', ko je sklad prazen
    );
end lifo;

architecture ideal of lifo is

    type shift_reg_array_type is array ( lifo_width - 1 downto 0 ) of std_logic_vector( lifo_size - 1 downto 0 );
    signal    lifo_content : shift_reg_array_type;
    signal    lifo_out      : std_logic_vector( lifo_width - 1 downto 0 );
    signal    s: std_logic_vector( 1 downto 0 ):= ( others => '0' );
    signal    lifo_mode     : std_logic_vector( 4 downto 0 );

    constant  zeros : std_logic_vector( lifo_size - 1 downto 0 ) := ( others => '0' );
    constant  highz : std_logic_vector( lifo_width - 1 downto 0 ) := ( others => 'Z' );

```

```

constant    ud_ctr_size : integer := integer( ceil( log2( real( lifo_size + 1 ) ) ) );
constant    ud_ctr_zero : std_logic_vector( ud_ctr_size - 1 downto 0 ) := ( others => '0' );

signal      ud_ctr_mode : std_logic_vector( 1 downto 0 ):= ( others => '0' );
signal      ud_ctr_out  : std_logic_vector( ud_ctr_size-1 downto 0 );

signal      FULL_sig, EMPTY_sig : std_logic;

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 dff IS
  PORT ( D, clk, nPRESET, nCLEAR : IN STD_LOGIC;
        Q : OUT STD_LOGIC );
END COMPONENT;

COMPONENT ud_counter is
  generic( ctr_size: natural := 6 );
  PORT (
    clk,    -- signal      ure
    nCLR,   -- signal      za brisanje števca ( aktiven '0' )
    D_nU,   -- signal      za smer štetja števca ( naraščajoče -> '0' )
    EN      : IN std_logic;  -- signal      za omogočanje štetja ( aktiven '1' )
    RCO : out std_logic;     -- izhodni prenos na naslednjo stopnjo ( Ripple carry out )
    Q : out std_logic_vector( ctr_size - 1 downto 0 )  -- izhodno štetje števca
  );
end COMPONENT;

begin

  lifo: for i in 0 to lifo_width-1 generate
    sr: shift_reg
      generic map ( reg_size => lifo_size )
      port map (

```

```

        clk => clk, nCLR => nCLR, sr_in => '0',          sl_in => data( i ),          s => s, x =>
zeros,        Q => lifo_content( i )
    );

    buf: dff
    port map (
        D => lifo_content( i )( 0 ), clk => clk, nPRESET => '1', nCLEAR => nCLR,          Q =>
lifo_out( i )
    );
end generate;

lifo_mode    <= PUSH & POP & FULL_sig & EMPTY_sig & nEnable;
with lifo_mode select s    <=
    "10" when "10000" | "10010",    "01" when "01000" | "01100",    "00" when others;

data    <= lifo_out when lifo_mode = "01000" or lifo_mode ="01100" else highz;

sc: ud_counter
generic map ( ctr_size => ud_ctr_size )
port map (
    clk => clk, nCLR => nCLR,          D_nU => ud_ctr_mode( 0 ), EN => ud_ctr_mode( 1 ),    Q => ud_ctr_out
);

with lifo_mode select ud_ctr_mode    <=
    "10" when "10000" | "10010",    "11" when "01000" | "01100",    "00" when others;

FULL_sig    <= '1' when unsigned( ud_ctr_out ) = lifo_size else '0';
EMPTY_sig    <= '1' when unsigned( ud_ctr_out ) = 0 else '0';

FULL    <= FULL_sig;
EMPTY    <= EMPTY_sig;

end ideal;

```

```

-- *****
-- **** STUDENT: 64190088
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: EMPTY zastavica ne deluje pravilno – po brisanju (nCLR='0') je sklad prazen. FULL zastavica ne deluje
pravilno – ko je sklad poln, se postavi (do sem je ok), potem pa mora to vrednost ohranjati, dokler ni operacije pop
(tu je napaka). POP operacija ne deluje pravilno – izhod sklada je vedno 0.
Opozorila sintetizatorja:
WARNING:HDLCompiler:746 - "lifo.vhd" Line 71: Range is empty (null range)
Napisali ste: 0 downto (lifo_width - 1), verjetno ste mislili obratno: 0 to (lifo_width - 1).
Pri primerjavi std_logic_vector tipa z integer konstanto morate uporabiti unsigned: unsigned(lifo_ctr) = lifo_size.
-- *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use IEEE.math_real.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity lifo is
    generic(
        lifo_width: natural := 4; -- velikost podatka sklada
        lifo_size: natural := 8 ); -- velikost sklada
    PORT (
        clk, -- signal ure
        nCLR, -- signal asinhronnega brisanja vsebine sklada ( aktiven '0' )
        nEnable, -- signal omogočanja sklada ( aktiven '0', sicer drži vsebino )
        PUSH, -- operacija vpisa na sklad ( aktiven '1' )
        POP : IN std_logic; -- operacija branja s sklada ( aktiven '1' )
        data : inout std_logic_vector( lifo_width - 1 downto 0 ); -- tristanjski izhod sklada
        FULL, -- izhod, ki postane '1', ko je sklad poln
        EMPTY : OUT std_logic -- izhod, ki postane '0', ko je sklad prazen
    );
end lifo;

architecture ideal of lifo is

    constant zeroes : std_logic_vector( lifo_size - 1 downto 0 ) := ( others => '0' );
    signal lifo_out : std_logic_vector( lifo_width - 1 downto 0 );
    signal lifo_mode : std_logic_vector( 4 downto 0 );

    signal s, ud_ctr_mode : std_logic_vector( 1 downto 0 ) := ( others => '0' );
    type shift_reg_array_type is array ( lifo_width - 1 downto 0 ) of std_logic_vector( lifo_size - 1 downto 0 );

```



```

signal      Q : shift_reg_array_type;
constant    ud_ctr_size : integer := integer( ceil( log2( real( lifo_size + 1 ) ) ) );

constant    lifo_empty : integer := 0; -- konstanta "prazen sklad"
signal      lifo_ctr : std_logic_vector( ud_ctr_size - 1 downto 0 ); -- skladovni števec
signal      FULL_sig, EMPTY_sig : std_logic; -- vmesna signala za FULL in EMPTY

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 dff IS
    PORT ( D, clk, nPRESET, nCLEAR : IN STD_LOGIC;
           Q : OUT STD_LOGIC );
END COMPONENT;

COMPONENT ud_counter is
    generic( ctr_size: natural := 6 );
    PORT ( clk, -- signal ure
           nCLR, -- signal za brisanje števca ( aktiven '0' )
           D_nU, -- signal za smer štetja števca ( naraščajoče -> '0' )
           EN : IN std_logic; -- signal za omogočanje štetja ( aktiven '1' )
           RCO : out std_logic; -- izhodni prenos na naslednjo stopnjo ( Ripple carry out )
           Q : out std_logic_vector( ctr_size - 1 downto 0 ) -- izhodno štetje števca
         );
end COMPONENT;

begin

    lifo_mode    <= nEnable & PUSH & POP & FULL_sig & EMPTY_sig;

    with lifo_mode select s    <= "10" when "01000" | "01001",          "01" when "00100" | "00110",          "00"
when others;

    data    <= lifo_out when lifo_mode = "00100" or lifo_mode = "00110" else ( others => 'Z' );

```

```

zanka_for_gen: for i in 0 downto ( lifo_width - 1 ) generate

    U0: shift_reg generic map( reg_size => lifo_size ) port map(
        clk => clk, nCLR => nCLR, sr_in => '0', sl_in => data( i ), s => s, x => zeroes, Q => Q( i ) );

    BUFi: dff port map ( Q( i )( 0 ), clk, '1', nCLR, lifo_out( i ) );

end generate;

with lifo_mode select ud_ctr_mode      <=      "10" when "01000" | "01001",          "11" when "00100" |
"00110",          "00" when others;

U2: ud_counter generic map ( ctr_size => ud_ctr_size ) port map (
    clk => clk, nCLR => nCLR, D_nU => ud_ctr_mode( 0 ), EN => ud_ctr_mode( 1 ), Q => lifo_ctr );

FULL_sig      <= '1' when data = lifo_size else '0';
EMPTY_sig     <= '1' when data = lifo_empty else '0';
FULL          <= FULL_sig;
EMPTY         <= EMPTY_sig;

end ideal;

```

```

-- *****
-- **** STUDENT: 64200100
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Iz simulacije sledi, da operacija POP ne deluje pravilno za prvovpisani podatek (oz. zadnji prebrani
podatek), ker je v pogoju za izhodni 3-stanjski vmesnik postavljeno EMPTY='0'. Zastavica EMPTY se postavi takoj ko
preberemo zadnji podatek iz sklada (ki pa je še vedno veljaven – ni enak Z). Pogoju za 3-stanjski vmesnik torej ni
odvisen od EMPTY.
-- *****

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use IEEE.math_real.ALL;

entity lifo is
    generic(
        lifo_width: natural := 4; -- velikost podatka sklada
        lifo_size: natural := 8 ); -- velikost sklada
    PORT (
        clk,      -- signal      ure
        nCLR,     -- signal      asinhronnega brisanja vsebine sklada ( aktiven '0' )
        nEnable,  -- signal      omogoanja sklada ( aktiven '0', sicer dri vsebino )
        PUSH,     -- operacija vpisa na sklad ( aktiven '1' )
        POP       : IN std_logic; -- operacija branja s sklada ( aktiven '1' )
        data      : inout std_logic_vector( lifo_width - 1 downto 0 ); -- tristanjski izhod sklada
        FULL,     -- izhod, ki postane '1', ko je sklad poln
        EMPTY     : OUT std_logic -- izhod, ki postane '0', ko je sklad prazen
    );
end lifo;

architecture ideal of lifo is

    constant zeroes : std_logic_vector( lifo_size - 1 downto 0 ) := ( others => '0' );
    signal lifo_out : std_logic_vector( lifo_width - 1 downto 0 );
    signal lifo_mode : std_logic_vector( 4 downto 0 );

    signal s, ud_ctr_mode : std_logic_vector( 1 downto 0 ) := ( others => '0' );
    type shift_reg_array_type is array ( lifo_width - 1 downto 0 ) of std_logic_vector( lifo_size - 1 downto 0 );
    signal Q : shift_reg_array_type;
    constant ud_ctr_size : integer := integer( ceil( log2( real( lifo_size + 1 ) ) ) );

    constant lifo_empty : integer := 0; -- konstanta "prazen sklad"
    signal lifo_ctr : std_logic_vector( ud_ctr_size - 1 downto 0 ); -- skladovni tevec

```

```

constant    visZ : std_logic_vector( lifo_width - 1 downto 0 ) := ( others => 'Z' );

signal      FULL_sig, EMPTY_sig : std_logic; -- vmesna signala za FULL in EMPTY

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 dff IS
    PORT ( D, clk, nPRESET, nCLEAR : IN STD_LOGIC;
           Q : OUT STD_LOGIC );
END COMPONENT;

COMPONENT ud_counter is
    generic( ctr_size: natural := 6 );
    PORT (
        clk,    -- signal      ure
        nCLR,   -- signal      za brisanje tevca ( aktiven '0' )
        D_nU,   -- signal      za smer tetja tevca ( naraajoe -> '0' )
        EN      : IN std_logic; -- signal      za omogoanje tetja ( aktiven '1' )
        RCO : out std_logic;    -- izhodni prenos na naslednjo stopnjo ( Ripple carry out )
        Q : out std_logic_vector( ctr_size - 1 downto 0 ) );
end COMPONENT;

begin
lifo: for i in 0 to lifo_width-1 generate
sr: shift_reg
generic map ( reg_size => lifo_size )
port map (
clk => clk, nCLR => nCLR, sr_in => '0',
sl_in => data( i ),
s => s, x => zeroes,
Q => Q( i ) );

buf: dff
port map (
D => Q( i )( 0 ), clk => clk, nPRESET => '1', nCLEAR => nCLR,

```

```

Q => lifo_out( i );
end generate;

lifo_mode    <= PUSH & POP & FULL_sig & EMPTY_sig & nEnable;
with lifo_mode select s    <=
"10" when "10000" | "10010",
"01" when "01000" | "01100",
"00" when others;

data    <= lifo_out when lifo_mode = "01000" or lifo_mode = "01100" else visZ;

sc: ud_counter
generic map ( ctr_size => ud_ctr_size )
port map (
clk => clk,
nCLR => nCLR,
D_nU => ud_ctr_mode( 0 ),
EN => ud_ctr_mode( 1 ),
Q => lifo_ctr );

with lifo_mode select ud_ctr_mode    <=
"10" when "10000" | "10010",
"11" when "01000" | "01100",
"00" when others;

FULL_sig    <= '1' when unsigned( lifo_ctr ) = lifo_size else '0';
EMPTY_sig    <= '1' when unsigned( lifo_ctr ) = 0 else '0';

FULL    <= FULL_sig;
EMPTY    <= EMPTY_sig;
end ideal;

```

```

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

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use IEEE.math_real.ALL;

entity lifo is
    generic(
        lifo_width: natural := 4; -- velikost podatka sklada
        lifo_size: natural := 8 ); -- velikost sklada
    PORT (
        clk, -- signal ure
        nCLR, -- signal asinhronega brisanja vsebine sklada ( aktiven '0' )
        nEnable, -- signal omogoanja sklada ( aktiven '0', sicer dri vsebino )
        PUSH, -- operacija vpisa na sklad ( aktiven '1' )
        POP : IN std_logic; -- operacija branja s sklada ( aktiven '1' )
        data : inout std_logic_vector( lifo_width - 1 downto 0 ); -- tristanjski izhod sklada
        FULL, -- izhod, ki postane '1', ko je sklad poln
        EMPTY : OUT std_logic -- izhod, ki postane '0', ko je sklad prazen
    );
end lifo;

architecture ideal of lifo is

    constant zeroes : std_logic_vector( lifo_size - 1 downto 0 ) := ( others => '0' );
    signal lifo_out : std_logic_vector( lifo_width - 1 downto 0 );
    signal lifo_mode : std_logic_vector( 4 downto 0 );

    signal s, ud_ctr_mode : std_logic_vector( 1 downto 0 ) := ( others => '0' );
    type shift_reg_array_type is array ( lifo_width - 1 downto 0 ) of std_logic_vector( lifo_size - 1 downto 0 );
    signal Q : shift_reg_array_type;
    constant ud_ctr_size : integer := integer( ceil( log2( real( lifo_size + 1 ) ) ) );

    constant lifo_empty : integer := 0; -- konstanta "prazen sklad"
    signal lifo_ctr : std_logic_vector( ud_ctr_size - 1 downto 0 ); -- skladovni tevec
    signal FULL_sig, EMPTY_sig : std_logic; -- vmesna signala za FULL in EMPTY

```

```

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 dff IS
    PORT ( D, clk, nPRESET, nCLEAR : IN STD_LOGIC;
           Q : OUT STD_LOGIC );
END COMPONENT;

COMPONENT ud_counter is
    generic( ctr_size: natural := 6 );
    PORT (
        clk,      -- signal      ure
        nCLR,     -- signal      za brisanje tevca ( aktiven '0' )
        D_nU,     -- signal      za smer tetja tevca ( naraajoe -> '0' )
        EN       : IN std_logic; -- signal      za omogoanje tetja ( aktiven '1' )
        RCO      : out std_logic; -- izhodni prenos na naslednjo stopnjo ( Ripple carry out )
        Q : out std_logic_vector( ctr_size - 1 downto 0 ) -- izhodno tetje tevca
    );
end COMPONENT;

begin

    U0 : for i in 0 to lifo_width - 1 generate
        U1 : shift_reg
            generic map( reg_size => lifo_size )
            port map( clk => clk,          nCLR => nCLR,          sr_in => '0',          sl_in =>
data( i ),          s => s,          x => zeroes,          Q => Q( i )
            );

        U2 : dff
            port map( D => Q( i )( 0 ),          clk => clk,          nPRESET => '1',          nCLEAR =>
nCLR,          Q => lifo_out( i )
            );

    end generate;

```

```

lifo_mode    <= nEnable & PUSH & POP & FULL_sig & EMPTY_sig;

with lifo_mode select
    s    <=    "10" when "01000" | "01001",          "01" when "00100" | "00110",          "00" when
others;

data    <= lifo_out when ( POP='1' and nEnable='0' ) else ( others => 'Z' );

with lifo_mode select
    ud_ctr_mode    <=    "10" when "01000" | "01001",          "11" when "00100" | "00110",          "00"
when others;

U3 : ud_counter
    generic map( ctr_size => ud_ctr_size )
    port map( clk => clk,          nCLR => nCLR,          D_nU => ud_ctr_mode( 0 ),          EN =>
ud_ctr_mode( 1 ),          Q => lifo_ctr
    );

FULL_sig    <= '1' when unsigned( lifo_ctr )=lifo_size else '0';
EMPTY_sig    <= '1' when unsigned( lifo_ctr )=lifo_empty else '0';
FULL    <= FULL_sig;
EMPTY    <= EMPTY_sig;

end ideal;

```



```

-- *****
-- **** STUDENT: 64200163
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Izhod sklada se mora pojaviti TAKOJ, ko se postavi POP operacija. Problem je v zapisu izhoda za data:
data <= lifo_out when lifo_mode = "00010" or lifo_mode = "00011" else (others => 'Z');
Iz zgornjega sledi, da lifo_out postane izhod, če sklad ni poln – dejansko je branje sklada čisto neodvisno od
zastavice FULL (važno samo, da ni prazen).
Pravilno bi se glasilo: data <= lifo_out when (nEnable = '0' and POP = '1') else (others => 'Z');
Sklad lahko beremo, če je omogočen in če je operacija POP. Če je sklad praze (EMPTY), potem ostaja na izhodu zadnja
vpisana vrednost (uporabnik pač lahko bere prazen sklad – ni sicer verjetno, ampak se zgodi).
-- *****

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.numeric_std.all;
use IEEE.math_real.all;

entity lifo is
    generic( lifo_width: natural := 4;
             lifo_size: natural := 8 );
    port( clk,      nCLR,      nEnable,      PUSH,      POP      : in std_logic;
          data : inout std_logic_vector( lifo_width - 1 downto 0 );
          FULL,      EMPTY : out std_logic
    );
end lifo;

architecture ideal of lifo is
    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 dff is
        port( D, clk, nPRESET, nCLEAR : in std_logic;
              Q : out std_logic
        );

```

```

end component;
component ud_counter is
    generic( ctr_size : natural := 6 );
    port( clk,
          nCLR,
          D_nU,
          EN : in std_logic;
          RCO : out std_logic;
          Q : out std_logic_vector( ctr_size - 1 downto 0 )
    );
end component;

constant zeroes : std_logic_vector( lifo_size - 1 downto 0 ) := ( others => '0' );
constant ud_ctr_size : integer := integer( ceil( log2( real( lifo_size + 1 ) ) ) );
constant lifo_empty : integer := 0;

type shift_reg_array_type is array ( lifo_width - 1 downto 0 ) of std_logic_vector( lifo_size - 1 downto 0 );
signal Q : shift_reg_array_type;

signal lifo_mode : std_logic_vector( 4 downto 0 );
signal lifo_ctr : std_logic_vector( ud_ctr_size - 1 downto 0 );
signal lifo_out : std_logic_vector( lifo_width - 1 downto 0 );
signal s, ud_ctr_mode : std_logic_vector( 1 downto 0 ) := ( others => '0' );
signal FULL_sig, EMPTY_sig : std_logic;

begin
lifo_assemble: for i in 0 to lifo_width - 1 generate
    u1: shift_reg
        generic map( reg_size => lifo_size )
        port map(
            clk => clk,          nCLR => nCLR,          sr_in => '0',          sl_in => data( i ),
s => s,          x => zeroes,          Q => Q( i )
        );
    BUFi: dff
        port map(
            D => Q( i )( 0 ),          clk => clk,          nPRESET => '1',          nCLEAR => nCLR,
Q => lifo_out( i )
        );
end generate;
lifo_mode    <= nEnable & FULL_sig & PUSH & POP & EMPTY_sig;

```

```

        with lifo_mode select s    <= "10" when "00100" | "00101",          "01" when "00010" | "01010",          "00"
when others;
    data    <= lifo_out when lifo_mode = "00010" or lifo_mode = "00011" else ( others => 'Z' );
    ctr: ud_counter
        generic map( ctr_size => ud_ctr_size )
        port map( clk => clk,          nCLR => nCLR,          D_nU => ud_ctr_mode( 0 ),          EN    =>
ud_ctr_mode( 1 ),          Q => lifo_ctr
        );
    with lifo_mode select ud_ctr_mode    <= "10" when "00100" | "00101",          "11" when "00010" | "01010",
    "00" when others;
    FULL_sig    <= '1' when unsigned( lifo_ctr ) = lifo_size else '0';
    EMPTY_sig    <= '1' when unsigned( lifo_ctr ) = lifo_empty else '0';
    FULL    <= FULL_sig;
    EMPTY    <= EMPTY_sig;
end ideal;

```

```

-- *****
-- **** STUDENT: 64200238
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Iz simulacije sledi, da operacija POP ne deluje pravilno za prvovpisani podatek (oz. zadnji prebrani
podatek), ker je v pogoju za izhodni 3-stanjski vmesnik postavljeno EMPTY='0'. Zastavica EMPTY se postavi takoj ko
preberemo zadnji podatek iz sklada (ki pa je še vedno veljaven – ni enak Z). Pogoju za 3-stanjski vmesnik torej ni
odvisen od EMPTY.
-- *****

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use IEEE.math_real.ALL;

entity lifo is
  generic(
    lifo_width: natural := 4;
    lifo_size: natural := 8
  );
  PORT(
    clk : in std_logic;
    nCLR : in std_logic;
    nEnable : in std_logic;
    PUSH : in std_logic;
    POP : in std_logic;
    data : inout std_logic_vector( lifo_width - 1 downto 0 );
    FULL : out std_logic;
    EMPTY : out std_logic
  );
end lifo;

architecture ideal of lifo is

  COMPONENT shift_reg is
    generic( reg_size: natural := 4 );
    PORT(
      clk : in std_logic;
      nCLR : in std_logic;
      sr_in : in std_logic;
      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 dff IS
PORT(
D : in STD_LOGIC;
clk : in STD_LOGIC;
nPRESET : in STD_LOGIC;
nCLEAR : in STD_LOGIC;
Q : out STD_LOGIC
);
END COMPONENT;

```

```

COMPONENT ud_counter is
generic( ctr_size: natural := 6 );
PORT(
clk : in std_logic;
nCLR : in std_logic;
D_nU : in std_logic;
EN : in std_logic;
RCO : out std_logic;
Q : out std_logic_vector( ctr_size - 1 downto 0 )
);
end COMPONENT;

```

```

constant zeroes : std_logic_vector( lifo_size - 1 downto 0 ) := ( others => '0' );
signal lifo_out : std_logic_vector( lifo_width - 1 downto 0 );
constant z : std_logic_vector( lifo_width - 1 downto 0 ) := ( others => 'Z' );
signal lifo_mode : std_logic_vector( 4 downto 0 );
signal s, ud_ctr_mode : std_logic_vector( 1 downto 0 ) := ( others => '0' );
type shift_reg_array_type is array( lifo_width - 1 downto 0 ) of std_logic_vector( lifo_size - 1 downto 0 );
signal Q : shift_reg_array_type;
constant ud_ctr_size : integer := integer( ceil( log2( real( lifo_size + 1 ) ) ) );
constant lifo_empty : integer := 0;
signal lifo_ctr : std_logic_vector( ud_ctr_size - 1 downto 0 );
signal FULL_sig, EMPTY_sig : std_logic;

```

```

begin
    anarhija: for i in 0 to lifo_width-1 generate
        anarhija2: shift_reg
            generic map ( reg_size => lifo_size )
    PORT map (
        clk => clk,
        nCLR => nCLR,
        sr_in => '0',
        sl_in => data( i ),
        s => s,
        x => zeroes,
        Q => Q( i )
    );

    BUFI: dff port map (
        D => Q( i )( 0 ),
        clk => clk,
        nPRESET => '1',
        nCLEAR => nCLR,
        Q => lifo_out( i )
    );
end generate anarhija;

lifo_mode    <= PUSH & POP & FULL_sig & EMPTY_sig & nEnable;
with lifo_mode select s    <=
    "01" when "01000" | "01100",
    "10" when "10000" | "10010",
    "00" when others;

lenin: ud_counter
generic map( ctr_size => ud_ctr_size )
port map (
    clk => clk,
    nCLR => nCLR,
    D_nU => ud_ctr_mode( 0 ),
    EN => ud_ctr_mode( 1 ),
    Q => lifo_ctr
);

with lifo_mode select ud_ctr_mode    <=

```

```
"11" when "01000" | "01100",  
"10" when "10000" | "10010",  
"00" when others;  
  
data <= lifo_out when lifo_mode = "01000" or lifo_mode = "01100" else z;  
FULL_sig <= '1' when unsigned( lifo_ctr ) = lifo_size else '0';  
EMPTY_sig <= '1' when unsigned( lifo_ctr ) = lifo_empty else '0';  
FULL <= FULL_sig;  
EMPTY <= EMPTY_sig;  
  
end ideal;
```

```

-- *****
-- **** STUDENT: 64200288
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni rezultatov simulacije, so pa napake sintetizatorja:
ERROR:HDLCompiler:580 - "lifo.vhd" Line 126: Mismatch on label ; expected ideal
Ime arhitekture se mora ujemati na začetku in na koncu. Na začetku imate "improved", na koncu imate "ideal". ☺
Po tej spremembi simulacija deluje pravilno.
-- *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use IEEE.math_real.ALL;

entity lifo is
  generic(
    lifo_width: natural := 4;
    lifo_size: natural := 8
  );
  PORT(
    clk : in std_logic;
    nCLR : in std_logic;
    nEnable : in std_logic;
    PUSH : in std_logic;
    POP : in std_logic;
    data : inout std_logic_vector( lifo_width - 1 downto 0 );
    FULL : out std_logic;
    EMPTY : out std_logic
  );
end lifo;

architecture ideal of lifo is

  COMPONENT shift_reg is
    generic( reg_size: natural := 4 );
    PORT(
      clk : in std_logic;
      nCLR : in std_logic;
      sr_in : in std_logic;
      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 dff IS
PORT(
D : in STD_LOGIC;
clk : in STD_LOGIC;
nPRESET : in STD_LOGIC;
nCLEAR : in STD_LOGIC;
Q : out STD_LOGIC
);
END COMPONENT;

```

```

COMPONENT ud_counter is
generic( ctr_size: natural := 6 );
PORT(
clk : in std_logic;
nCLR : in std_logic;
D_nU : in std_logic;
EN : in std_logic;
RCO : out std_logic;
Q : out std_logic_vector( ctr_size - 1 downto 0 )
);
end COMPONENT;

```

```

constant nicle : std_logic_vector( lifo_size - 1 downto 0 ) := ( others => '0' );
signal lifo_out : std_logic_vector( lifo_width - 1 downto 0 );
constant z_state : std_logic_vector( lifo_width - 1 downto 0 ) := ( others => 'Z' );
signal lifo_mode : std_logic_vector( 4 downto 0 );
signal s, ud_ctr_mode : std_logic_vector( 1 downto 0 ) := ( others => '0' );
type shift_reg_array_type is array( lifo_width - 1 downto 0 ) of std_logic_vector( lifo_size - 1 downto 0 );
signal Q : shift_reg_array_type;
constant ud_ctr_size : integer := integer( ceil( log2( real( lifo_size + 1 ) ) ) );
constant lifo_empty : integer := 0;
signal lifo_ctr : std_logic_vector( ud_ctr_size - 1 downto 0 );
signal FULL_sig, EMPTY_sig : std_logic;

```

```
begin
```

```
reg_block: for i in 0 to lifo_width-1 generate  
  reg_inst: shift_reg  
  generic map ( reg_size => lifo_size )  
  port map (  
    clk => clk,  
    nCLR => nCLR,  
    sr_in => '0',  
    sl_in => data( i ),  
    s => s,  
    x => nicle,  
    Q => Q( i )  
  );
```

```
buf_inst: dff port map (  
  D => Q( i )( 0 ),  
  clk => clk,  
  nPRESET => '1',  
  nCLEAR => nCLR,  
  Q => lifo_out( i )  
);  
end generate reg_block;
```

```
lifo_mode    <= PUSH & POP & FULL_sig & EMPTY_sig & nEnable;
```

```
with lifo_mode select s <=  
  "10" when "10000" | "10010",  
  "01" when "01000" | "01100",  
  "00" when others;
```

```
data <= lifo_out when lifo_mode = "01000" or lifo_mode = "01100" else z_state;
```

```
counter_inst: ud_counter  
generic map ( ctr_size => ud_ctr_size )  
port map (  
  clk => clk,  
  nCLR => nCLR,  
  D_nU => ud_ctr_mode( 0 ),  
  EN => ud_ctr_mode( 1 ),
```

```

Q => lifo_ctr
);

with lifo_mode select ud_ctr_mode      <=
"10" when "10000" | "10010",
"11" when "01000" | "01100",
"00" when others;

FULL_sig      <= '1' when unsigned( lifo_ctr ) = lifo_size else '0';
EMPTY_sig     <= '1' when unsigned( lifo_ctr ) = 0 else '0';

FULL <= FULL_sig;
EMPTY <= EMPTY_sig;

end improved;

```

```

-- *****
-- **** STUDENT: 64200296
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Iz simulacije sledi, da operacija POP ne deluje pravilno za prvovpisani podatek (oz. zadnji prebrani
podatek), ker je v pogoju za izhodni 3-stanjski vmesnik postavljeno EMPTY='0'. Zastavica EMPTY se postavi takoj ko
preberemo zadnji podatek iz sklada (ki pa je še vedno veljaven – ni enak Z). Pogoju za 3-stanjski vmesnik torej ni
odvisen od EMPTY.
-- *****

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use IEEE.math_real.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity lifo is
    generic(
        lifo_width: natural := 4; -- velikost podatka sklada
        lifo_size: natural := 8 ); -- velikost sklada
    PORT (
        clk, -- signal ure
        nCLR, -- signal asinhronnega brisanja vsebine sklada ( aktiven '0' )
        nEnable, -- signal omogoanja sklada ( aktiven '0', sicer dri vsebino )
        PUSH, -- operacija vpisa na sklad ( aktiven '1' )
        POP : IN std_logic; -- operacija branja s sklada ( aktiven '1' )
        data : inout std_logic_vector( lifo_width - 1 downto 0 ); -- tristanjski izhod sklada
        FULL, -- izhod, ki postane '1', ko je sklad poln
        EMPTY : OUT std_logic -- izhod, ki postane '0', ko je sklad prazen
    );
end lifo;

architecture ideal of lifo is

    constant zeroes : std_logic_vector( lifo_size - 1 downto 0 ) := ( others => '0' );
    signal lifo_out : std_logic_vector( lifo_width - 1 downto 0 );
    signal lifo_mode : std_logic_vector( 4 downto 0 );

    signal s, ud_ctr_mode : std_logic_vector( 1 downto 0 ) := ( others => '0' );
    type shift_reg_array_type is array ( lifo_width - 1 downto 0 ) of std_logic_vector( lifo_size - 1 downto 0 );
    signal Q : shift_reg_array_type;
    constant ud_ctr_size : integer := integer( ceil( log2( real( lifo_size + 1 ) ) ) );

```

```

constant    lifo_empty : integer := 0; -- konstanta "prazen sklad"
signal      lifo_ctr : std_logic_vector( ud_ctr_size - 1 downto 0 ); -- skladovni tevec
signal      FULL_sig, EMPTY_sig : std_logic; -- vmesna signala za FULL in EMPTY

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 dff IS
    PORT ( D, clk, nPRESET, nCLEAR : IN STD_LOGIC;
           Q : OUT STD_LOGIC );
END COMPONENT;

COMPONENT ud_counter is
    generic( ctr_size: natural := 6 );
    PORT (
        clk,    -- signal      ure
        nCLR,   -- signal      za brisanje tevca ( aktiven '0' )
        D_nU,   -- signal      za smer tetja tevca ( naraajoe -> '0' )
        EN      : IN std_logic; -- signal      za omogoanje tetja ( aktiven '1' )
        RCO : out std_logic;    -- izhodni prenos na naslednjo stopnjo ( Ripple carry out )
        Q : out std_logic_vector( ctr_size - 1 downto 0 ) -- izhodno tetje tevca
    );
end COMPONENT;

begin

    lifo_mode    <= push & empty_sig & nenable & full_sig & pop;
    with lifo_mode select s    <=
        "10" when "10000" | "11000",    "01" when "00001" | "00011",    "00" when others;

    data    <= lifo_out when ( s="01" ) else ( others=>'Z' );

    u1:for i in 0 to lifo_width-1 generate
        reg: shift_reg
            generic map( reg_size=>lifo_size )

```

```

s=>s,      port map( clk=>clk,      nCLR=>nCLR,      sr_in=>'0',      sl_in=>data( i ),
            x=>zeroes,      Q=>Q( i )
            );

            bufer: dff
            port map( D=>Q( i )( 0 ),      clk=>clk,      nPRESET=>'1',      nCLEAR=>nCLR,
Q=>lifo_out( i )
            );

            end generate;

with lifo_mode select ud_ctr_mode      <=
    "10" when "10000" | "11000",      "11" when "00001" | "00011",      "00" when others;

count: ud_counter
    generic map( ctr_size=>ud_ctr_size )
    port map( clk=>clk,      nCLR=>nCLR,      D_nU=>ud_ctr_mode( 0 ),      EN=>ud_ctr_mode( 1
),      RCO=>open,      Q=>lifo_ctr
    );

empty_sig      <= '1' when lifo_ctr=lifo_empty else '0';
empty      <= empty_sig;
full_sig      <= '1' when lifo_ctr=lifo_size else '0';
FULL      <= full_sig;
end ideal;

```

```

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

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use IEEE.math_real.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity lifo is
    generic(
        lifo_width: natural := 4; -- velikost podatka sklada
        lifo_size: natural := 8 ); -- velikost sklada
    PORT (
        clk, -- signal ure
        nCLR, -- signal asinhronnega brisanja vsebine sklada ( aktiven '0' )
        nEnable, -- signal omogočanja sklada ( aktiven '0', sicer drži vsebino )
        PUSH, -- operacija vpisa na sklad ( aktiven '1' )
        POP : IN std_logic; -- operacija branja s sklada ( aktiven '1' )
        data : inout std_logic_vector( lifo_width - 1 downto 0 ); -- tristanjski izhod sklada
        FULL, -- izhod, ki postane '1', ko je sklad poln
        EMPTY : OUT std_logic -- izhod, ki postane '0', ko je sklad prazen
    );
end lifo;

architecture ideal of lifo is

    constant zeros : std_logic_vector( lifo_size - 1 downto 0 ) := ( others => '0' );
    signal lifo_out : std_logic_vector( lifo_width - 1 downto 0 );
    signal lifo_mode : std_logic_vector( 4 downto 0 );

    signal s, ud_ctr_mode : std_logic_vector( 1 downto 0 ) := ( others => '0' );
    type shift_reg_array_type is array ( lifo_width - 1 downto 0 ) of std_logic_vector( lifo_size - 1 downto 0 );
    signal Q_lf: shift_reg_array_type;
    constant ud_ctr_size : integer := integer( ceil( log2( real( lifo_size + 1 ) ) ) ); -- ceil( ) zaokroži
navzgor

    constant lifo_empty : integer := 0; -- konstanta "prazen sklad"
    signal lifo_ctr : std_logic_vector( ud_ctr_size - 1 downto 0 ); -- skladovni števec

```

```

signal      FULL_sig, EMPTY_sig : std_logic; -- vmesna signala za FULL in EMPTY

constant    hi_z : std_logic_vector( lifo_width - 1 downto 0 ) := ( others => 'Z' );

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 ); -- 2, ker so samo štirje mode-i
        x : in std_logic_vector( reg_size - 1 downto 0 );
        Q : out std_logic_vector( reg_size - 1 downto 0 )
        );
end COMPONENT;

COMPONENT dff IS
  PORT ( D, clk, nPRESET, nCLEAR : IN STD_LOGIC;
        Q : OUT STD_LOGIC );
END COMPONENT;

COMPONENT ud_counter is
  generic( ctr_size: natural := 6 );
  PORT ( clk, -- signal ure
        nCLR, -- signal za brisanje števca ( aktiven '0' )
        D_nU, -- signal za smer štetja števca ( naraščajoče -> '0' )
        EN : IN std_logic; -- signal za omogočanje štetja ( aktiven '1' )
        RCO : out std_logic; -- izhodni prenos na naslednjo stopnjo ( Ripple carry out )
        Q : out std_logic_vector( ctr_size - 1 downto 0 ) -- izhodno štetje števca
        );
end COMPONENT;

begin
  lf: for i in 0 to lifo_width-1 generate
    U2: shift_reg
      generic map ( reg_size => lifo_size )
      port map (
        clk => clk,
        nCLR => nCLR,
        sr_in => '0',
        sl_in => data( i ),
        s => s,
        x => zeros,
        Q => Q_lf( i )
      );
  end generate
end;

```



```

        BUFi: dff
            port map (
                D => Q_lf( i )( 0 ),
clk => clk,
nPRESET => '1',
nCLEAR => nCLR,          Q => lifo_out( i )
            );
        end generate;

        lifo_mode    <= PUSH & POP & nEnable & FULL_sig & EMPTY_sig;
        with lifo_mode select
s    <=
            "10" when "10000" | "10001",    "01" when "01000" | "01010",    "00" when others;

        data    <= lifo_out when
lifo_mode = "01000" or lifo_mode = "01010" or lifo_mode = "01001"
        else hi_z;

        with lifo_mode select
ud_ctr_mode <=
            "01" when "10000" | "10001",    "11" when "01000" | "01010",    "00" when others;

        U3: ud_counter
            generic map ( ctr_size => ud_ctr_size )
            port map (
                clk => clk,  nCLR => nCLR,          D_nU => ud_ctr_mode( 1 ), EN => ud_ctr_mode( 0 ),
RCO => open,          Q => lifo_ctr
            );

        FULL_sig    <= '1' when unsigned( lifo_ctr ) = lifo_size else '0';
        EMPTY_sig   <= '1' when unsigned( lifo_ctr ) = 0 else '0';

        FULL    <= FULL_sig;
        EMPTY   <= EMPTY_sig;

    end ideal;

```

```

-- *****
-- **** STUDENT: 64210113
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Uporabljate stare Synopsis knjižnice (STD_LOGIC_ARITH), česar v predlogah že ni kar nekaj let. Iz
simulacije sledi, da operacija POP ne deluje pravilno za prvovpisani podatek (oz. zadnji prebrani podatek), ker je v
pogoju za izhodni 3-stanjski vmesnik postavljeno EMPTY='0'. Zastavica EMPTY se postavi takoj ko preberemo zadnji
podatek iz sklada (ki pa je še vedno veljaven – ni enak Z). Pogoju za 3-stanjski vmesnik torej ni odvisen od EMPTY.

-- *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.math_real.ALL;

entity lifo is
    generic(
        lifo_width: natural := 4; -- velikost podatka sklada
        lifo_size: natural := 8 ); -- velikost sklada
    PORT (
        clk,      -- signal      ure
        nCLR,     -- signal      asinhronnega brisanja vsebine sklada ( aktiven '0' )
        nEnable,  -- signal      omogočanja sklada ( aktiven '0', sicer drži vsebino )
        PUSH,     -- operacija vpisa na sklad ( aktiven '1' )
        POP       : IN std_logic; -- operacija branja s sklada ( aktiven '1' )
        data : inout std_logic_vector( lifo_width - 1 downto 0 ); -- tristanjski izhod sklada
        FULL,    -- izhod, ki postane '1', ko je sklad poln
        EMPTY   : OUT std_logic      -- izhod, ki postane '0', ko je sklad prazen
    );
end lifo;

architecture ndv of lifo is
    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 ud_counter
generic( ctr_size: natural := 6 );
PORT(
    clk : IN std_logic;
    nCLR : IN std_logic;
    D_nU : IN std_logic;
    EN : IN std_logic;
    RCO : OUT std_logic;
    Q : OUT std_logic_vector( ctr_size - 1 downto 0 )
);
END COMPONENT;

```

```

COMPONENT dff IS
PORT ( D, clk, nPRESET, nCLEAR : IN STD_LOGIC;
        Q : OUT STD_LOGIC );
END COMPONENT;

```

```

type shift_reg_array_type is array ( lifo_width - 1 downto 0 ) of std_logic_vector( lifo_size - 1 downto 0 );
signal    lifo_out : std_logic_vector( lifo_width - 1 downto 0 );
signal    lifo_mode : std_logic_vector( 4 downto 0 );
signal    lifo_Q : shift_reg_array_type;
signal    s: std_logic_vector( 1 downto 0 );
constant  zero : std_logic_vector( lifo_size - 1 downto 0 ) := ( others => '0' );
constant  Z : std_logic_vector( lifo_width - 1 downto 0 ) := ( others => 'Z' );
constant  ud_ctr_size : integer := integer( ceil( log2( real( lifo_size + 1 ) ) ) );
signal    ud_ctr_mode : std_logic_vector( 1 downto 0 );
signal    ud_output : std_logic_vector( ud_ctr_size-1 downto 0 );
signal    emptyS : std_logic;
signal    fullS : std_logic;

```

begin

```

LMap: for i in 0 to lifo_width-1 generate
    SM: shift_reg
        generic map ( reg_size => lifo_size )
        port map (
            clk => clk,          nCLR => nCLR,          sr_in => '0',          sl_in => data( i ),
            s => s,              x => zero,             Q => lifo_Q( i )
        );
end generate

```

```

        buf: dff
        port map (
            D => lifo_Q( i )( 0 ),          clk => clk,          nPRESET => '1',          nCLEAR => nCLR,
            Q => lifo_out( i )
        );

    end generate;

    lifo_mode    <= PUSH & POP & fullS & emptyS & nEnable;

    with lifo_mode select s    <= "10" when "10000" | "10010",          "01" when "01000" | "01100",          "00"
when others;

    data    <= lifo_out when lifo_mode = "01000" or lifo_mode ="01100"
            else Z;

    SU: ud_counter
    generic map ( ctr_size => ud_ctr_size )
    port map (
        clk => clk,          nCLR => nCLR,          D_nU => ud_ctr_mode( 0 ),          EN => ud_ctr_mode( 1
    ),
        Q => ud_output
    );

    with lifo_mode select ud_ctr_mode    <= "10" when "10000" | "10010",          "11" when "01000" | "01100",
    "00" when others;

    fullS    <= '1' when unsigned( ud_output ) = lifo_size else '0';
    emptyS    <= '1' when unsigned( ud_output ) = 0 else '0';
    FULL    <= fullS;
    EMPTY    <= emptyS;

end ndv;

```

```

-- *****
-- **** STUDENT: 64210132
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni rezultatov simulacije zaradi napak sintetizatorja:
ERROR:HDLCompiler:40 - "lifo.vhd" Line 76: dff is not a component
Pri poimenovanju neke komponente v povezovalnem stavku lahko uporabite katerokoli ime **RAZEN** imena te komponente:
Namesto: dff: tff port map(      ... naredite d_ff: dff port map(
ERROR:HDLCompiler:69 - "lifo.vhd" Line 83: <full_sig> is not declared.
Uporablajte signal: lifo_mode <= PUSH & POP & FULL_sig & EMPTY_sig & nEnable;
Pri čemer imate deklarirana svoja signala: signal FULL_s, EMPTY_s : std_logic;
Po teh spremembah simulacija deluje pravilno.
Uporablajte stare Synopsis knjižnice (STD_LOGIC_ARITH), česar v predlogah že ni kar nekaj let.
-- *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.math_real.ALL;

entity lifo is
    generic(
        lifo_width: natural := 4; -- velikost podatka sklada
        lifo_size: natural := 8 ); -- velikost sklada
    PORT (
        clk,      -- signal      ure
        nCLR,      -- signal      asinhronnega brisanja vsebine sklada ( aktiven '0' )
        nEnable,   -- signal      omogočanja sklada ( aktiven '0', sicer drži vsebino )
        PUSH,      -- operacija vpisa na sklad ( aktiven '1' )
        POP        : IN std_logic;      -- operacija branja s sklada ( aktiven '1' )
        data : inout std_logic_vector( lifo_width - 1 downto 0 ); -- tristanjski izhod sklada
        FULL,      -- izhod, ki postane '1', ko je sklad poln
        EMPTY      : OUT std_logic      -- izhod, ki postane '0', ko je sklad prazen
    );
end lifo;

architecture ndv of lifo is
    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 ud_counter
generic( ctr_size: natural := 6 );
PORT(

```

```

    clk : IN std_logic;
    nCLR : IN std_logic;
    D_nU : IN std_logic;
    EN : IN std_logic;
    RCO : OUT std_logic;
    Q : OUT std_logic_vector( ctr_size - 1 downto 0 )

```

```

);
END COMPONENT;

```

```

COMPONENT dff IS
PORT ( D, clk, nPRESET, nCLEAR : IN STD_LOGIC;
        Q : OUT STD_LOGIC );
END COMPONENT;

```

```

type shift_reg_array_type is array ( lifo_width - 1 downto 0 ) of std_logic_vector( lifo_size - 1 downto 0 );
signal    lifo_content : shift_reg_array_type;
signal    lifo_out : std_logic_vector( lifo_width - 1 downto 0 );
signal    s: std_logic_vector( 1 downto 0 );
signal    lifo_mode : std_logic_vector( 4 downto 0 );

```

```

constant    zeros : std_logic_vector( lifo_size - 1 downto 0 ) := ( others => '0' );
constant    highz : std_logic_vector( lifo_width - 1 downto 0 ) := ( others => 'Z' );

```

```

constant    ud_ctr_size : integer := integer( ceil( log2( real( lifo_size + 1 ) ) ) );
constant    ud_ctr_zero : std_logic_vector( ud_ctr_size - 1 downto 0 ) := ( others => '0' );

```

```

signal    ud_ctr_mode : std_logic_vector( 1 downto 0 );
signal    ud_ctr_out : std_logic_vector( ud_ctr_size-1 downto 0 );

```

```

signal    FULL_s, EMPTY_s : std_logic;

```

```

begin

```

```

    lifo: for i in 0 to lifo_width-1 generate

```

```

        sr: shift_reg
        generic map ( reg_size => lifo_size )
        port map (
            clk => clk, nCLR => nCLR, sr_in => '0',          sl_in => data( i ),          s => s, x =>
zeros,          Q => lifo_content( i )
        );

        dff: dff
        port map (
            D => lifo_content( i )( 0 ), clk => clk, nPRESET => '1', nCLEAR => nCLR,          Q =>
lifo_out( i )
        );
    end generate;

    lifo_mode    <= PUSH & POP & FULL_sig & EMPTY_sig & nEnable;
    with lifo_mode select s    <=
        "10" when "10000" | "10010",    "01" when "01000" | "01100",    "00" when others;

    data    <= lifo_out when lifo_mode = "01000" or lifo_mode = "01100" else highz;

    count: ud_counter
    generic map ( ctr_size => ud_ctr_size )
    port map (
        clk => clk, nCLR => nCLR,          D_nU => ud_ctr_mode( 0 ), EN => ud_ctr_mode( 1 ),          Q => ud_ctr_out
    );

    with lifo_mode select ud_ctr_mode    <=
        "10" when "10000" | "10010",    "11" when "01000" | "01100",    "00" when others;

    FULL_s <= '1' when unsigned( ud_ctr_out ) = lifo_size else '0';
    EMPTY_s    <= '1' when unsigned( ud_ctr_out ) = 0 else '0';

    FULL    <= FULL_s;
    EMPTY    <= EMPTY_s;

end ndv;

```

```

-- *****
-- **** STUDENT: 64210290
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use IEEE.math_real.ALL;

ENTITY lifo IS
  GENERIC(
    lifo_width: NATURAL:= 4;  -- velikost podatka sklada
    lifo_size: NATURAL := 8   -- velikost sklada
  );
  PORT(
    clk,    -- signal      ure
    nCLR,   -- signal      asinhronnega brisanja vsebine sklada ( aktiven '0' )
    nEnable, -- signal      omogočanja sklada ( aktiven '0', sicer drži vsebino )
    PUSH,   -- operacija vpisa na sklad ( aktiven '1' )
    POP : IN STD_LOGIC; -- operacija branja s sklada ( aktiven '1' )
    data : INOUT STD_LOGIC_VECTOR( lifo_width - 1 DOWNT0 0 ); -- tristanjski izhod sklada
    FULL,  -- izhod, ki postane '1', ko je sklad poln
    EMPTY : OUT STD_LOGIC      -- izhod, ki postane '0', ko je sklad prazen
  );
END lifo;

architecture ideal of lifo is

  constant zeroes : std_logic_vector( lifo_size - 1 downto 0 ) := ( others => '0' );
  signal lifo_out : std_logic_vector( lifo_width - 1 downto 0 );
  signal lifo_mode : std_logic_vector( 4 downto 0 );

  signal s, ud_ctr_mode : std_logic_vector( 1 downto 0 ) := ( others => '0' );
  type shift_reg_array_type is array ( lifo_width - 1 downto 0 ) of std_logic_vector( lifo_size - 1 downto 0 );
  signal Q : shift_reg_array_type;
  constant ud_ctr_size : integer := integer( ceil( log2( real( lifo_size + 1 ) ) ) );

  constant lifo_empty : integer := 0; -- konstanta "prazen sklad"

```



```

signal      lifo_ctr : std_logic_vector( ud_ctr_size - 1 downto 0 );  -- skladovni števec
signal      FULL_sig, EMPTY_sig : std_logic; -- vmesna signala za FULL in EMPTY

component shift_reg is
    generic(
        reg_size: natural
    );
    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 dff is
    port(
        D, clk, nPRESET, nCLEAR : in std_logic;
        Q : out std_logic
    );
end component;

component ud_counter is
    generic(
        ctr_size: natural := 6
    );
    port(
        clk,      -- signal      ure
        nCLR,     -- signal      za brisanje števca ( aktiven '0' )
        D_nU,     -- signal      za smer štetja števca ( naraščajoče -> '0' )
        EN       : in std_logic;  -- signal      za omogočanje štetja ( aktiven '1' )
        RCO      : out std_logic;  -- izhodni prenos na naslednjo stopnjo ( Ripple carry out )
        Q : out std_logic_vector( ctr_size - 1 downto 0 )  -- izhodno štetje števca
    );
end component;

begin

lifo : for idw in 0 to lifo_width-1 generate
    u0: shift_reg

```

```

        generic map(
            reg_size => lifo_size
        )
        port map(
            clk => clk,          nCLR => nCLR,          sr_in => '0',          sl_in => data( idw ),
s => s,          x => zeroes,          Q => Q( idw )
        );
        buf0 : dff
            port map(
                clk => clk,          nCLEAR => nCLR,          nPRESET => '1',          D => Q( idw )( 0 ),
Q => lifo_out( idw )
            );
end generate lifo;

lifo_mode    <= PUSH&POP&FULL_sig&EMPTY_sig&nENABLE;

with lifo_mode select s    <=
    "10" when "10000" | "10010",    "01" when "01000" | "01100",    "00" when others;

data    <= lifo_out when ( not( PUSH ) and POP and not( nENABLE ) )='1' else ( others=>'Z' );

with lifo_mode select ud_ctr_mode    <=
    "01" when "10000" | "10010",    "11" when "01000" | "01100",    "00" when others;

FULL_sig    <= '1' when unsigned( lifo_ctr )=lifo_size else '0';
EMPTY_sig    <= '1' when unsigned( lifo_ctr )=lifo_empty else '0';
FULL    <= FULL_sig;
EMPTY    <= EMPTY_sig;

c1 : ud_counter
    generic map(
        ctr_size => ud_ctr_size
    )
    port map(
        clk => clk,          nCLR => nCLR,          D_nU => ud_ctr_mode( 1 ),          EN => ud_ctr_mode( 0
    ),          RCO => open,          Q => lifo_ctr
    );

PROCESS( Q ) -- process for logging the value of FIFO
    function array_of_slv_to_string( Q : shift_reg_array_type ) return string is

```

```

        use Std.TextIO.all;
        variable bv: bit_vector( Q( Q'left )'range ) := to_bitvector( Q( Q'left ) );
        variable lp: line;
    begin
        for i in Q'RANGE loop      -- scroll the array of std_logic_vectors
            bv := to_bitvector( Q( i ) );    -- convert to printable value
            write( lp, bv );    -- append dynamic string
            write( lp, HT );    -- insert horizontal tab
        end loop;
        return lp.all;    -- return the concatenated string
    end;
BEGIN
    report array_of_slv_to_string( Q );    -- write internal FIFO contents
END PROCESS;

end ideal;

```

```

-- *****
-- **** STUDENT: 64210382
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Uporabljate stare Synopsis knjižnice (STD_LOGIC_ARITH), česar v predlogah že ni kar nekaj let. Iz
simulacije sledi, da operacija POP ne deluje pravilno za prvovpisani podatek (oz. zadnji prebrani podatek), ker je v
pogoju za izhodni 3-stanjski vmesnik postavljeno EMPTY='0'. Zastavica EMPTY se postavi takoj ko preberemo zadnji
podatek iz sklada (ki pa je še vedno veljaven – ni enak Z). Pogoju za 3-stanjski vmesnik torej ni odvisen od EMPTY.
-- *****

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

entity lifo is
    generic(
        lifo_width: natural := 4; -- velikost podatka sklada
        lifo_size: natural := 8 ); -- velikost sklada
    PORT (
        clk,      -- signal      ure
        nCLR,      -- signal      asinhronnega brisanja vsebine sklada ( aktiven '0' )
        nEnable,    -- signal      omogočanja sklada ( aktiven '0', sicer drži vsebino )
        PUSH,      -- operacija vpisa na sklad ( aktiven '1' )
        POP       : IN std_logic;    -- operacija branja s sklada ( aktiven '1' )
        data : inout std_logic_vector( lifo_width - 1 downto 0 ); -- tristanjski izhod sklada
        FULL,      -- izhod, ki postane '1', ko je sklad poln
        EMPTY : OUT std_logic      -- izhod, ki postane '0', ko je sklad prazen
    );
end lifo;

architecture ndv of lifo is
    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 ud_counter

```

```

generic( ctr_size: natural := 6 );
PORT(
    clk : IN std_logic;
    nCLR : IN std_logic;
    D_nU : IN std_logic;
    EN : IN std_logic;
    RCO : OUT std_logic;
    Q : OUT std_logic_vector( ctr_size - 1 downto 0 )
);
END COMPONENT;

COMPONENT dff IS
PORT ( D, clk, nPRESET, nCLEAR : IN STD_LOGIC;
        Q : OUT STD_LOGIC );
END COMPONENT;

type shift_reg_array_type is array ( lifo_width - 1 downto 0 ) of std_logic_vector( lifo_size - 1 downto 0 );
signal    lifo_content : shift_reg_array_type;
signal    lifo_out : std_logic_vector( lifo_width - 1 downto 0 );
signal    s: std_logic_vector( 1 downto 0 );
signal    lifo_mode : std_logic_vector( 4 downto 0 );

constant  zeros : std_logic_vector( lifo_size - 1 downto 0 ) := ( others => '0' );
constant  highz : std_logic_vector( lifo_width - 1 downto 0 ) := ( others => 'Z' );

constant  ud_ctr_size : integer := integer( ceil( log2( real( lifo_size + 1 ) ) ) );
constant  ud_ctr_zero : std_logic_vector( ud_ctr_size - 1 downto 0 ) := ( others => '0' );

signal    ud_ctr_mode : std_logic_vector( 1 downto 0 );
signal    ud_ctr_out : std_logic_vector( ud_ctr_size-1 downto 0 );

signal    FULL_sig, EMPTY_sig : std_logic;

begin
lifo: for i in 0 to lifo_width-1 generate
    sr: shift_reg
        generic map ( reg_size => lifo_size )
        port map (
            clk => clk, nCLR => nCLR, sr_in => '0',          sl_in => data( i ),          s => s, x =>
zeros,          Q => lifo_content( i )

```

```

    );

    buf: dff
    port map (
        D => lifo_content( i )( 0 ), clk => clk, nPRESET => '1', nCLEAR => nCLR,      Q =>
lifo_out( i )
    );
end generate;

lifo_mode    <= PUSH & POP & FULL_sig & EMPTY_sig & nEnable;
with lifo_mode select s    <=
    "10" when "10000" | "10010",      "01" when "01000" | "01100",      "00" when others;

data    <= lifo_out when lifo_mode = "01000" or lifo_mode ="01100" else highz;

sc: ud_counter
generic map ( ctr_size => ud_ctr_size )
port map (
    clk => clk,  nCLR => nCLR,      D_nU => ud_ctr_mode( 0 ), EN => ud_ctr_mode( 1 ),  Q => ud_ctr_out
);

with lifo_mode select ud_ctr_mode    <=
    "10" when "10000" | "10010",      "11" when "01000" | "01100",      "00" when others;

FULL_sig    <= '1' when unsigned( ud_ctr_out ) = lifo_size else '0';
EMPTY_sig   <= '1' when unsigned( ud_ctr_out ) = 0 else '0';

FULL    <= FULL_sig;
EMPTY   <= EMPTY_sig;

end ndv;

```

```

-- *****
-- **** STUDENT: 64210384
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Iz simulacije sledi, da operacija POP ne deluje pravilno za prvovpisani podatek (oz. zadnji prebrani
podatek), ker je v pogoju za izhodni 3-stanjski vmesnik postavljeno EMPTY='0'. Zastavica EMPTY se postavi takoj ko
preberemo zadnji podatek iz sklada (ki pa je še vedno veljaven – ni enak Z). Pogoj za 3-stanjski vmesnik torej ni
odvisen od EMPTY.
-- *****

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use IEEE.math_real.ALL;

entity lifo is
    generic(
        lifo_width: natural := 4; -- velikost podatka sklada
        lifo_size: natural := 8 ); -- velikost sklada
    PORT (
        clk,      -- signal      ure
        nCLR,     -- signal      asinhronnega brisanja vsebine sklada ( aktiven '0' )
        nEnable,  -- signal      omogočanja sklada ( aktiven '0', sicer drži vsebino )
        PUSH,     -- operacija vpisa na sklad ( aktiven '1' )
        POP       : IN std_logic;    -- operacija branja s sklada ( aktiven '1' )
        data      : inout std_logic_vector( lifo_width - 1 downto 0 ); -- tristanjski izhod sklada
        FULL,     -- izhod, ki postane '1', ko je sklad poln
        EMPTY    : OUT std_logic    -- izhod, ki postane '0', ko je sklad prazen
    );
end lifo;

architecture ideal of lifo is

    constant zeroes : std_logic_vector( lifo_size - 1 downto 0 ) := ( others => '0' );
    signal lifo_out : std_logic_vector( lifo_width - 1 downto 0 );
    signal lifo_mode : std_logic_vector( 4 downto 0 );

    signal s, ud_ctr_mode : std_logic_vector( 1 downto 0 ) := ( others => '0' );
    type shift_reg_array_type is array ( lifo_width - 1 downto 0 ) of std_logic_vector( lifo_size - 1 downto 0 );
    signal Q : shift_reg_array_type;
    constant ud_ctr_size : integer := integer( ceil( log2( real( lifo_size + 1 ) ) ) );

    constant lifo_empty : integer := 0;          -- konstanta "prazen sklad"

```

```

signal      lifo_ctr : std_logic_vector( ud_ctr_size - 1 downto 0 );  -- skladovni števec
signal      FULL_sig, EMPTY_sig : std_logic; -- vmesna signala za FULL in EMPTY

signal      Q_ud : std_logic_vector( ud_ctr_size - 1 downto 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 dff IS
  PORT ( D, clk, nPRESET, nCLEAR : IN STD_LOGIC;
        Q : OUT STD_LOGIC );
END COMPONENT;

COMPONENT ud_counter is
  generic( ctr_size: natural := 6 );
  PORT ( clk, -- signal ure
        nCLR, -- signal za brisanje števca ( aktiven '0' )
        D_nU, -- signal za smer štetja števca ( naraščajoče -> '0' )
        EN : IN std_logic; -- signal za omogočanje štetja ( aktiven '1' )
        RCO : out std_logic; -- izhodni prenos na naslednjo stopnjo ( Ripple carry out )
        Q : out std_logic_vector( ctr_size - 1 downto 0 ) -- izhodno štetje števca
      );
end COMPONENT;

begin
  povezava:
    for i in 0 to lifo_width - 1 generate
      blok: shift_reg
        generic map( reg_size => lifo_size )
        port map(
          clk => clk,          nCLR => nCLR,          sr_in => '0',          sl_in => data( i ),
          s => s,              x => zeroes,          Q => Q( i )
        );
    end generate
end begin

```



```

        pomnjenje: dff
            port map(
                D => Q( i )( Q'right ),          clk => clk,          nPRESET => '1',          nCLEAR => nCLR,
                Q => lifo_out( i )
            );
        end generate;

        lifo_mode    <= nEnable & POP & PUSH & FULL_sig & EMPTY_sig;

        with lifo_mode select s    <= "10" when "00100" | "00101",          "01" when "01000" | "01010",          "00"
when others;
        with lifo_mode select data<= lifo_out when "01000" | "01010",          ( others => 'Z' ) when others;

        stevec: ud_counter
            generic map( ctr_size => ud_ctr_size )
            port map(
                clk => clk,          nCLR => nCLR,          D_nU => ud_ctr_mode( ud_ctr_mode'right ),
                EN => ud_ctr_mode( ud_ctr_mode'left ),          RCO => open,          Q => Q_ud
            );

        with lifo_mode select ud_ctr_mode    <= "11" when "01000" | "01010",          "10" when "00100" | "00101",
        "00" when others;
        EMPTY_sig    <= '1' when unsigned( Q_ud ) = 0 else '0';
        FULL_sig      <= '1' when unsigned( Q_ud ) = lifo_size else '0';

        EMPTY <= EMPTY_sig;
        FULL  <= FULL_sig;

    end ideal;

```

```

-- *****
-- **** STUDENT: 64210386
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Iz simulacije sledi, da operacija POP ne deluje pravilno za prvovpisani podatek (oz. zadnji prebrani
podatek), ker je v pogoju za izhodni 3-stanjski vmesnik postavljeno EMPTY='0'. Zastavica EMPTY se postavi takoj ko
preberemo zadnji podatek iz sklada (ki pa je še vedno veljaven – ni enak Z). Pogoj za 3-stanjski vmesnik torej ni
odvisen od EMPTY.
-- *****

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use IEEE.math_real.ALL;

entity lifo is
    generic(
        lifo_width: natural := 4; -- velikost podatka sklada
        lifo_size: natural := 8 ); -- velikost sklada
    PORT (
        clk,      -- signal      ure
        nCLR,     -- signal      asinhronega brisanja vsebine sklada ( aktiven '0' )
        nEnable,  -- signal      omogoanja sklada ( aktiven '0', sicer dri vsebino )
        PUSH,     -- operacija vpisa na sklad ( aktiven '1' )
        POP       : IN std_logic; -- operacija branja s sklada ( aktiven '1' )
        data      : inout std_logic_vector( lifo_width - 1 downto 0 ); -- tristanjski izhod sklada
        FULL,     -- izhod, ki postane '1', ko je sklad poln
        EMPTY     : OUT std_logic -- izhod, ki postane '0', ko je sklad prazen
    );
end lifo;

architecture ideal of lifo is

    constant z : std_logic_vector( lifo_width - 1 downto 0 ) := ( others => 'Z' );
    constant zeroes : std_logic_vector( lifo_size - 1 downto 0 ) := ( others => '0' );
    signal lifo_out : std_logic_vector( lifo_width - 1 downto 0 );
    signal lifo_mode : std_logic_vector( 4 downto 0 );

    signal s, ud_ctr_mode : std_logic_vector( 1 downto 0 ) := ( others => '0' );
    type shift_reg_array_type is array ( lifo_width - 1 downto 0 ) of std_logic_vector( lifo_size - 1 downto 0 );
    signal Q : shift_reg_array_type;
    constant ud_ctr_size : integer := integer( ceil( log2( real( lifo_size + 1 ) ) ) );
    signal ud_out : std_logic_vector( ud_ctr_size-1 downto 0 );

```

```

constant    lifo_empty : integer := 0; -- konstanta "prazen sklad"
signal      lifo_ctr : std_logic_vector( ud_ctr_size - 1 downto 0 ); -- skladovni tevec
signal      FULL_sig, EMPTY_sig : std_logic; -- vmesna signala za FULL in EMPTY

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 dff IS
    PORT ( D, clk, nPRESET, nCLEAR : IN STD_LOGIC;
           Q : OUT STD_LOGIC );
END COMPONENT;

COMPONENT ud_counter is
    generic( ctr_size: natural := 6 );
    PORT (
        clk,    -- signal      ure
        nCLR,   -- signal      za brisanje tevca ( aktiven '0' )
        D_nU,   -- signal      za smer tetja tevca ( naraajoe -> '0' )
        EN      : IN std_logic; -- signal      za omogoanje tetja ( aktiven '1' )
        RCO : out std_logic;    -- izhodni prenos na naslednjo stopnjo ( Ripple carry out )
        Q : out std_logic_vector( ctr_size - 1 downto 0 ) -- izhodno tetje tevca
    );
end COMPONENT;

begin

    lifo: for i in 0 to lifo_width-1 generate
        sreg: shift_reg
            generic map ( reg_size => lifo_size )
            port map (
                clk => clk,          nCLR => nCLR,          sr_in => '0',          sl_in => data( i ),
                s => s,              x => zeroes,          Q => Q( i )
            );
    end generate;

```

```

buff: dff
    port map (
        D => Q( i )( 0 ),          clk => clk,          nPRESET => '1',          nCLEAR => nCLR,
        Q => lifo_out( i )
    );
end generate;

lifo_mode    <= PUSH & POP & FULL_sig & EMPTY_sig & nEnable;

with lifo_mode select s    <=
    "10" when "10000" | "10010",    "01" when "01000" | "01100",    "00" when others;

data    <= lifo_out when lifo_mode = "01000" or lifo_mode ="01100" else z;

sc: ud_counter
    generic map ( ctr_size => ud_ctr_size )
    port map (
        clk => clk,          nCLR => nCLR,          D_nU => ud_ctr_mode( 0 ),          EN => ud_ctr_mode( 1
    ),
        Q => ud_out
    );

with lifo_mode select ud_ctr_mode    <=
    "10" when "10000" | "10010",    "11" when "01000" | "01100",    "00" when others;

FULL_sig    <= '1' when unsigned( ud_out ) = lifo_size else '0';
EMPTY_sig    <= '1' when unsigned( ud_out ) = 0 else '0';

FULL    <= FULL_sig;
EMPTY    <= EMPTY_sig;

end ideal;

```

```

-- *****
-- **** STUDENT: 64210445
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Iz simulacije sledi, da operacija POP ne deluje pravilno za prvovpisani podatek (oz. zadnji prebrani
podatek), ker je v pogoju za izhodni 3-stanjski vmesnik postavljeno EMPTY='0'. Zastavica EMPTY se postavi takoj ko
preberemo zadnji podatek iz sklada (ki pa je še vedno veljaven – ni enak Z). Pogoju za 3-stanjski vmesnik torej ni
odvisen od EMPTY.
-- *****

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use IEEE.math_real.ALL;

entity lifo is
  generic(
    lifo_width: natural := 4; -- velikost podatka sklada
    lifo_size: natural := 8); -- velikost sklada
  PORT ( clk,          -- signal ure
        nCLR,          -- signal asinhronnega brisanja vsebine sklada (aktiven '0')
        nEnable,       -- signal omogočanja sklada (aktiven '0', sicer drži vsebino)
        PUSH,          -- operacija vpisa na sklad (aktiven '1')
        POP : IN std_logic; -- operacija branja s sklada (aktiven '1')
        data : inout std_logic_vector(lifo_width - 1 downto 0); --tristanjski izhod sklada
        FULL,          -- izhod, ki postane '1', ko je sklad poln
        EMPTY : OUT std_logic      -- izhod, ki postane '0', ko je sklad prazen
  );
end lifo;

architecture ideal of lifo is

  constant zeroes : std_logic_vector(lifo_size - 1 downto 0) := (others => '0');
  signal lifo_out : std_logic_vector(lifo_width - 1 downto 0);
  signal lifo_mode : std_logic_vector(4 downto 0);

  signal s, ud_ctr_mode : std_logic_vector(1 downto 0) := (others => '0');
  type shift_reg_array_type is array (lifo_width - 1 downto 0) of std_logic_vector(lifo_size - 1 downto 0);
  signal Q : shift_reg_array_type;
  constant ud_ctr_size : integer := integer(ceil(log2(real(lifo_size + 1))));

  constant lifo_empty : integer := 0; -- konstanta "prazen sklad"

```

```

signal lifo_ctr : std_logic_vector(ud_ctr_size - 1 downto 0);    -- skladovni števec
signal FULL_sig, EMPTY_sig : std_logic; --vmesna signala za FULL in EMPTY

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 dff IS
    PORT ( D, clk, nPRESET, nCLEAR : IN STD_LOGIC;
          Q : OUT STD_LOGIC);
END COMPONENT;

COMPONENT ud_counter is
    generic( ctr_size: natural := 6);
    PORT ( clk,                                -- signal ure
          nCLR,                                -- signal za brisanje števca (aktiven '0')
          D_nU,                                -- signal za smer štetja števca (naraščajoče -> '0')
          EN : IN std_logic;                   -- signal za omogočanje štetja (aktiven '1')
          RCO : out std_logic;                  -- izhodni prenos na naslednjo stopnjo (Ripple carry out)
          Q : out std_logic_vector(ctr_size - 1 downto 0)    -- izhodno štetje števca
    );
end COMPONENT;

constant hz : std_logic_vector(lifo_width - 1 downto 0) := (others => 'Z');
signal ud_ctr_out : std_logic_vector(ud_ctr_size-1 downto 0);

begin

    lifo: for i in 0 to lifo_width-1 generate
        sr: shift_reg
            generic map (reg_size => lifo_size)
            port map (
                clk => clk,
                nCLR => nCLR,
                sr_in => '0',

```

```

        sl_in => data(i),
        s => s,
        x => zeroes,
        Q => Q(i)
    );

    buf: dff
    port map (
        D => Q(i)(0),
        clk => clk,
        nPRESET => '1',
        nCLEAR => nCLR,
        Q => lifo_out(i)
    );
end generate;

lifo_mode <= PUSH & POP & FULL_sig & EMPTY_sig & nEnable;
with lifo_mode select s <= "10" when "10000" | "10010",
    "01" when "01000" | "01100",
    "00" when others;

data <= lifo_out when lifo_mode = "01000" or lifo_mode = "01100" else hz;

sc: ud_counter
generic map (ctr_size => ud_ctr_size)
port map (
    clk => clk,
    nCLR => nCLR,
    D_nU => ud_ctr_mode(0),
    EN => ud_ctr_mode(1),
    Q => ud_ctr_out
);

with lifo_mode select ud_ctr_mode <=
    "10" when "10000" | "10010",
    "11" when "01000" | "01100",
    "00" when others;

FULL_sig <= '1' when unsigned(ud_ctr_out) = lifo_size else '0';
EMPTY_sig <= '1' when unsigned(ud_ctr_out) = 0 else '0';

```

```
FULL <= FULL_sig;  
EMPTY <= EMPTY_sig;
```

```
end ideal;
```



```

-- *****
-- **** STUDENT: 64210455
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Naloga ni programirana.
-- *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use IEEE.math_real.ALL;

entity lifo is
    generic(
        lifo_width: natural := 4; -- velikost podatka sklada
        lifo_size: natural := 8 ); -- velikost sklada
    PORT (
        clk, -- signal ure
        nCLR, -- signal asinhronega brisanja vsebine sklada ( aktiven '0' )
        nEnable, -- signal omogočanja sklada ( aktiven '0', sicer drži vsebino )
        PUSH, -- operacija vpisa na sklad ( aktiven '1' )
        POP : IN std_logic; -- operacija branja s sklada ( aktiven '1' )
        data : inout std_logic_vector( lifo_width - 1 downto 0 ); -- tristanjski izhod sklada
        FULL, -- izhod, ki postane '1', ko je sklad poln
        EMPTY : OUT std_logic -- izhod, ki postane '0', ko je sklad prazen
    );
end lifo;

architecture ideal of lifo is

    constant zeroes : std_logic_vector( lifo_size - 1 downto 0 ) := ( others => '0' );
    signal lifo_out : std_logic_vector( lifo_width - 1 downto 0 );
    signal lifo_mode : std_logic_vector( 4 downto 0 );

    signal s, ud_ctr_mode : std_logic_vector( 1 downto 0 ) := ( others => '0' );
    type shift_reg_array_type is array ( lifo_width - 1 downto 0 ) of std_logic_vector( lifo_size - 1 downto 0 );
    signal Q : shift_reg_array_type;
    constant ud_ctr_size : integer := integer( ceil( log2( real( lifo_size + 1 ) ) ) );

    constant lifo_empty : integer := 0; -- konstanta "prazen sklad"
    signal lifo_ctr : std_logic_vector( ud_ctr_size - 1 downto 0 ); -- skladovni števec
    signal FULL_sig, EMPTY_sig : std_logic; -- vmesna signala za FULL in EMPTY

```

```

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 dff IS
    PORT ( D, clk, nPRESET, nCLEAR : IN STD_LOGIC;
           Q : OUT STD_LOGIC );
END COMPONENT;

COMPONENT ud_counter is
    generic( ctr_size: natural := 6 );
    PORT (
        clk,      -- signal      ure
        nCLR,     -- signal      za brisanje števca ( aktiven '0' )
        D_nU,     -- signal      za smer štetja števca ( naraščajoče -> '0' )
        EN        : IN std_logic; -- signal      za omogočanje štetja ( aktiven '1' )
        RCO : out std_logic;      -- izhodni prenos na naslednjo stopnjo ( Ripple carry out )
        Q : out std_logic_vector( ctr_size - 1 downto 0 ) -- izhodno štetje števca
    );
end COMPONENT;

begin

end ideal;

```

```

-- *****
-- **** STUDENT: 64210457
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Uporabljate stare Synopsis knjižnice (STD_LOGIC_ARITH), česar v predlogah že ni kar nekaj let. Iz
simulacije sledi, da operacija POP ne deluje pravilno za prvovpisani podatek (oz. zadnji prebrani podatek), ker je v
pogoju za izhodni 3-stanjski vmesnik postavljeno EMPTY='0'. Zastavica EMPTY se postavi takoj ko preberemo zadnji
podatek iz sklada (ki pa je še vedno veljaven – ni enak Z). Pogoju za 3-stanjski vmesnik torej ni odvisen od EMPTY.
-- *****

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

entity lifo is
    generic(
        lifo_width: natural := 4; -- velikost podatka sklada
        lifo_size: natural := 8 ); -- velikost sklada
    PORT (
        clk, -- signal ure
        nCLR, -- signal asinhronnega brisanja vsebine sklada ( aktiven '0' )
        nEnable, -- signal omogočanja sklada ( aktiven '0', sicer drži vsebino )
        PUSH, -- operacija vpisa na sklad ( aktiven '1' )
        POP : IN std_logic; -- operacija branja s sklada ( aktiven '1' )
        data : inout std_logic_vector( lifo_width - 1 downto 0 ); -- tristanjski izhod sklada
        FULL, -- izhod, ki postane '1', ko je sklad poln
        EMPTY : OUT std_logic -- izhod, ki postane '0', ko je sklad prazen
    );
end lifo;

architecture ndv of lifo is
    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 ud_counter

```

```

generic( ctr_size: natural := 6 );
PORT(
    clk : IN std_logic;
    nCLR : IN std_logic;
    D_nU : IN std_logic;
    EN : IN std_logic;
    RCO : OUT std_logic;
    Q : OUT std_logic_vector( ctr_size - 1 downto 0 )
);
END COMPONENT;

COMPONENT dff IS
PORT ( D, clk, nPRESET, nCLEAR : IN STD_LOGIC;
        Q : OUT STD_LOGIC );
END COMPONENT;

type shift_reg_array_type is array ( lifo_width - 1 downto 0 ) of std_logic_vector( lifo_size - 1 downto 0 );
signal    lifo_content : shift_reg_array_type;
signal    lifo_out : std_logic_vector( lifo_width - 1 downto 0 );
signal    s: std_logic_vector( 1 downto 0 );
signal    lifo_mode : std_logic_vector( 4 downto 0 );

constant  zeros : std_logic_vector( lifo_size - 1 downto 0 ) := ( others => '0' );
constant  highz : std_logic_vector( lifo_width - 1 downto 0 ) := ( others => 'Z' );

constant  ud_ctr_size : integer := integer( ceil( log2( real( lifo_size + 1 ) ) ) );
constant  ud_ctr_zero : std_logic_vector( ud_ctr_size - 1 downto 0 ) := ( others => '0' );

signal    ud_ctr_mode : std_logic_vector( 1 downto 0 );
signal    ud_ctr_out : std_logic_vector( ud_ctr_size-1 downto 0 );

signal    FULL_sig, EMPTY_sig : std_logic;

begin
lifo: for i in 0 to lifo_width-1 generate
    sr: shift_reg
        generic map ( reg_size => lifo_size )
        port map (
            clk => clk, nCLR => nCLR, sr_in => '0',                sl_in => data( i ),                s => s, x =>
zeros,                Q => lifo_content( i )

```

```

    );

    buf: dff
    port map (
        D => lifo_content( i )( 0 ), clk => clk, nPRESET => '1', nCLEAR => nCLR,      Q =>
lifo_out( i )
    );
end generate;

lifo_mode    <= PUSH & POP & FULL_sig & EMPTY_sig & nEnable;
with lifo_mode select s    <=
    "10" when "10000" | "10010",      "01" when "01000" | "01100",      "00" when others;

data    <= lifo_out when lifo_mode = "01000" or lifo_mode ="01100" else highz;

sc: ud_counter
generic map ( ctr_size => ud_ctr_size )
port map (
    clk => clk,  nCLR => nCLR,      D_nU => ud_ctr_mode( 0 ), EN => ud_ctr_mode( 1 ),  Q => ud_ctr_out
);

with lifo_mode select ud_ctr_mode    <=
    "10" when "10000" | "10010",      "11" when "01000" | "01100",      "00" when others;

FULL_sig    <= '1' when unsigned( ud_ctr_out ) = lifo_size else '0';
EMPTY_sig   <= '1' when unsigned( ud_ctr_out ) = 0 else '0';

FULL    <= FULL_sig;
EMPTY   <= EMPTY_sig;

end ndv;

```

```

-- *****
-- **** STUDENT: 64240430
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Iz simulacije sledi, da operacija POP ne deluje pravilno za prvovpisani podatek (oz. zadnji prebrani
podatek), ker je v pogoju za izhodni 3-stanjski vmesnik postavljeno EMPTY='0'. Zastavica EMPTY se postavi takoj ko
preberemo zadnji podatek iz sklada (ki pa je še vedno veljaven – ni enak Z). Pogoj za 3-stanjski vmesnik torej ni
odvisen od EMPTY.
-- *****

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use IEEE.math_real.ALL;

entity lifo is
    generic(
        lifo_width: natural := 4; -- velikost podatka sklada
        lifo_size: natural := 8 ); -- velikost sklada
    PORT (
        clk, -- signal ure
        nCLR, -- signal asinhronega brisanja vsebine sklada ( aktiven '0' )
        nEnable, -- signal omogočanja sklada ( aktiven '0', sicer drži vsebino )
        PUSH, -- operacija vpisa na sklad ( aktiven '1' )
        POP : IN std_logic; -- operacija branja s sklada ( aktiven '1' )
        data : inout std_logic_vector( lifo_width - 1 downto 0 ); -- tristanjski izhod sklada
        FULL, -- izhod, ki postane '1', ko je sklad poln
        EMPTY : OUT std_logic -- izhod, ki postane '0', ko je sklad prazen
    );
end lifo;

architecture ideal of lifo is

    COMPONENT dff IS
        PORT ( D, clk, nPRESET, nCLEAR : IN STD_LOGIC;
              Q : OUT STD_LOGIC );
    END COMPONENT;

    COMPONENT ud_counter is
        generic( ctr_size: natural := 6 );
        PORT (
            clk, -- signal ure
            nCLR, -- signal za brisanje števca ( aktiven '0' )
            D_nU, -- signal za smer štetja števca ( naraščajoče -> '0' )

```

```

        EN      : IN std_logic;      -- signal          za omogočanje štetja ( aktiven '1' )
        RCO : out std_logic;          -- izhodni prenos na naslednjo stopnjo ( Ripple carry out )
        Q : out std_logic_vector( ctr_size - 1 downto 0 )  -- izhodno štetje števca
    );
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 zeroes : std_logic_vector( lifo_size - 1 downto 0 ) := ( others => '0' );--
signal lifo_out : std_logic_vector( lifo_width - 1 downto 0 );
signal lifo_mode : std_logic_vector( 4 downto 0 );

signal s, ud_ctr_mode : std_logic_vector( 1 downto 0 ) := ( others => '0' );
type shift_reg_array_type is array ( lifo_width - 1 downto 0 ) of std_logic_vector( lifo_size - 1 downto 0 );
signal Q_sig : shift_reg_array_type;
constant ud_ctr_size : integer := integer( ceil( log2( real( lifo_size + 1 ) ) ) );--

constant lifo_empty : integer := 0;-- konstanta "prazen sklad"
signal lifo_ctr : std_logic_vector( ud_ctr_size - 1 downto 0 ); -- skladovni števec
signal FULL_sig, EMPTY_sig : std_logic; -- vmesna signala za FULL in EMPTY

signal ud_ctr_out : std_logic_vector( ud_ctr_size - 1 downto 0 );

begin

    pov1:
        for i in 0 to lifo_width - 1 generate
            shift: shift_reg
                generic map( reg_size => lifo_size )
                port map(
                    clk => clk,          nCLR => nCLR,          sr_in => '0',          sl_in => data( i ),
                    s => s,          x => zeroes,          Q => Q_sig( i )
                );
        end generate
    end pov1;

```

```

        d_ff: dff
        port map(
            D => Q_sig( i )( 0 ),          clk => clk,          nPRESET => '1',          nCLEAR => nCLR,
Q => lifo_out( i )
        );
    end generate;

    lifo_mode    <= nEnable & POP & PUSH & FULL_sig & EMPTY_sig;

    with lifo_mode select s    <= "10" when "00100" | "00101",  "01" when "01000" | "01010",          "00" when
others;

    with lifo_mode select data<= lifo_out when "01000" | "01010",          ( others => 'Z' ) when others;

    stevec: ud_counter
        generic map( ctr_size => ud_ctr_size )
        port map (
            clk => clk,          nCLR => nCLR,          D_nU => ud_ctr_mode( 0 ),          EN => ud_ctr_mode( 1
),          RCO => open,          Q => ud_ctr_out );

    with lifo_mode select ud_ctr_mode    <= "11" when "01000" | "01010",          "10" when "00100" | "00101",
    "00" when others;

    FULL_sig    <= '1' when unsigned( ud_ctr_out ) = lifo_size else '0';
    EMPTY_sig   <= '1' when unsigned( ud_ctr_out ) = 0 else '0';

    EMPTY <= EMPTY_sig;
    FULL  <= FULL_sig;
end ideal;

```



```

-- *****
-- **** PREDLOGA VAJE
-- *****
-- KOMENTARJI K OCENI NALOGE
-- Matej Možek: Ni pripomb
-- *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use IEEE.math_real.ALL;

entity lifo is
    generic(
        lifo_width: natural := 4; -- velikost podatka sklada
        lifo_size: natural := 8 ); -- velikost sklada
    PORT (
        clk, -- signal ure
        nCLR, -- signal asinhronega brisanja vsebine sklada ( aktiven '0' )
        nEnable, -- signal omogočanja sklada ( aktiven '0', sicer drži vsebino )
        PUSH, -- operacija vpisa na sklad ( aktiven '1' )
        POP : IN std_logic; -- operacija branja s sklada ( aktiven '1' )
        data : inout std_logic_vector( lifo_width - 1 downto 0 ); -- tristanjski izhod sklada
        FULL, -- izhod, ki postane '1', ko je sklad poln
        EMPTY : OUT std_logic -- izhod, ki postane '0', ko je sklad prazen
    );
end lifo;

architecture ideal of lifo is

    constant zeroes : std_logic_vector( lifo_size - 1 downto 0 ) := ( others => '0' );
    signal lifo_out : std_logic_vector( lifo_width - 1 downto 0 );
    signal lifo_mode : std_logic_vector( 4 downto 0 );

    signal s, ud_ctr_mode : std_logic_vector( 1 downto 0 ) := ( others => '0' );
    type shift_reg_array_type is array ( lifo_width - 1 downto 0 ) of std_logic_vector( lifo_size - 1 downto 0 );
    signal Q : shift_reg_array_type;

    -- Skladovni števec šteje število podatkov, ki so vpisani na sklado, torej vrednost
    -- števca niè pomeni "prazen sklad", ena pomeni ena vrednost na sklado itd,
    -- vrednost lifo_size vrednost pomeni - "sklad poln", zato moramo predvideti
    -- za števec še eno mesto več za stanje "prazen sklad", kar pomeni skupno
    -- število stanj števca ( lifo_size + 1 )

```

```

constant    ud_ctr_size : integer := integer( ceil( log2( real( lifo_size + 1 ) ) ) );

constant    lifo_empty : integer := 0; -- konstanta "prazen sklad"
signal      lifo_ctr : std_logic_vector( ud_ctr_size - 1 downto 0 ); -- skladovni števec
signal      FULL_sig, EMPTY_sig : std_logic; -- vmesna signala za FULL in EMPTY

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 dff IS
    PORT ( D, clk, nPRESET, nCLEAR : IN STD_LOGIC;
           Q : OUT STD_LOGIC );
END COMPONENT;

COMPONENT ud_counter is
    generic( ctr_size: natural := 6 );
    PORT (
        clk,    -- signal      ure
        nCLR,    -- signal      za brisanje števca ( aktiven '0' )
        D_nU,    -- signal      za smer štetja števca ( naraščajoče -> '0' )
        EN       : IN std_logic; -- signal      za omogočanje štetja ( aktiven '1' )
        RCO : out std_logic;      -- izhodni prenos na naslednjo stopnjo ( Ripple carry out )
        Q : out std_logic_vector( ctr_size - 1 downto 0 ) -- izhodno štetje števca
    );
end COMPONENT;

begin

    lifo_mode    <= nEnable & FULL_sig & EMPTY_sig & PUSH & POP;

    WITH ( lifo_mode ) SELECT
        s    <=    "10"  WHEN "00010" | "00110", -- sklad omogočen, ni poln, ni ali je prazen, operacija
PUSH ( pomik levo )
        "01"  WHEN "00001" | "01001", -- sklad omogočen, je ali pa ni poln, ni prazen, operacija POP (
pomik desno )

```

```

        "00" WHEN OTHERS; -- sicer sklad drži vsebino ( registri ohranjajo vsebino )

L1: FOR i IN 0 TO lifo_width - 1 GENERATE
    LIFO_REG: shift_reg generic map ( reg_size => lifo_size )
    port map ( clk, nCLR, '0', data( i ), s, zeroes, Q( i ) );

    -- LSB element polja Q je izhod sklada - mora biti "double buffered", -- sicer se zadnji vpisan podatek *tako*
    -- pojavi na izhodu sklada, -- kar pomeni, da bo ob naslednjem ciklu signala ure na izhodu že
    -- predzadnji vpisan podatek ( glej spremno sliko k besedilu naloge ).
    BUFI: dff port map ( Q( i )( 0 ), clk, '1', nCLR, lifo_out( i ) );

END GENERATE;

-- ud_ctr_mode je dvobitni signal sestavljen iz En in D_nU signalov števca
WITH ( lifo_mode ) SELECT
    ud_ctr_mode <= "10" WHEN "00010" | "00110", -- sklad omogočen, ni poln, ni ali je prazen,
operacija PUSH ( pomik levo )
    "11" WHEN "00001" | "01001", -- sklad omogočen, je ali pa ni poln, ni prazen, operacija POP (
pomik desno )
    "00" WHEN OTHERS; -- sicer drži vsebino ( ohrani skladovni števec )

LIFO_COUNTER: ud_counter
generic map( ctr_size => ud_ctr_size )
port map ( clk => clk, -- signal ure
nCLR => nCLR, -- signal za brisanje števca ( aktiven '0' )
D_nU => ud_ctr_mode( 0 ), -- signal za smer štetja števca ( dol -> '0' )
EN => ud_ctr_mode( 1 ), -- signal za omogočanje štetja ( aktiven '1' )
Q => lifo_ctr-- izhodno štetje - vrednost skladovnega kazalca
);

-- če je vrednost skladovnega kazalca enaka lifo_size, postane FULL = '1'
-- torej ni več možno vpisovanje na sklad brez izgube vsebine, kar predstavlja
-- način delovanja s preprečitvijo izgube vsebine sklada.
-- Drug način delovanja je, da skladovni števec ne šteje več, ko enkrat doseže
-- lifo_size, podatki pa se še vedno vpisujejo v sklad:
-- Način delovanja pri katerem je najbolj svež podatek obenem najbolj pomemben.
-- Način delovanja sklada določa ciljna aplikacija.
FULL_sig <= '1' when ( unsigned( lifo_ctr ) = lifo_size ) else '0';
FULL <= FULL_sig;

```

```
-- če je vrednost skladovnega kazalca enaka nič, postane EMPTY = '1'
-- torej je vpisovanje na sklad možno
EMPTY_sig    <= '1' when ( unsigned( lifo_ctr ) = lifo_empty ) else '0';
EMPTY <= EMPTY_sig;

-- tristanjski vmesnik za branje s sklada ( POP )
data    <= lifo_out when ( nEnable = '0' and POP = '1' ) else ( others => 'Z' );

end ideal;
```

