FIFO vmesnik (first in - first out buffer)

V VHDL programirajte arhitekturo splošne strukture FIFO (ang. first in - first out) vmesnika.
Entiteta izdelane strukture **fifo** ima priključke:

```
entity fifo is
      generic(    fifo_width: natural := 4;      -- dolžina vhodnega podatka
                  fifo_size: natural := 8);      -- število hranjenih podatkov
      PORT (      clk,          -- signal ure (spremembe na sprednjo fronto)
                  nCLR,         -- signal za asinhrono brisanje (vsebina FIFO gre na 0)
                  nEnable,      --   signal za omogočanje FIFO ('0'-> omogočen vpis, '1' -> ohranja stanje)
                  LOAD : IN std_logic;     -- signal za omogočanje nalaganja ('1'-> fifo_in se vpiše)
                  fifo_in : in std_logic_vector(fifo_width - 1 downto 0);      -- vhodni podatek
                  fifo_out : out std_logic_vector(fifo_width - 1 downto 0)     -- izhodni podatek
            );
end fifo;
```

Naloge:

1. V arhivu predloge naloge se nahaja datoteka **dff.vhd**. V to datoteko kopirajte entiteto in različne arhitekture D flip-flopa (**dff**) v spodnji tabeli. Ime entitete mora biti **dff**. Podana je entiteta strukture:

   ```
   ENTITY dff IS
   PORT (      D, -- vhod D flip-flopa
               clk, -- signal ure (prožen na sprednjo fronto)
               nPRESET, -- signal za asinhrono postavljanje (nPRESET = '0' se postavi Q=>'1')
               nCLEAR : IN STD_LOGIC; -- signal za asinhrono brisanje (nCLEAR = '0' se postavi Q=>'0')
               Q : OUT STD_LOGIC); -- izhod D flip-flopa
   END dff;
   ```

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY dff IS
PORT ( D,                          -- vhod D flip-flopa
       clk,                        -- signal ure (prožen na sprednjo fronto)
       nPRESET,                    -- signal za postavljanje (nPRESET = '0' se postavi Q=>'1')
       nCLEAR : IN STD_LOGIC;      -- signal za brisanje (nCLEAR = '0' se postavi Q=>'0')
       Q      : OUT STD_LOGIC);    -- izhod D flip-flopa
END dff;

-- VHDL klice primitivov FDC, FDP, FDS, FDR si oglejte na:
--     https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/7series_hdl.pdf
-- Opise primitivov FDC, FDP, FDS, FDR si oglejte na:
--     https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/7series_scm.pdf

ARCHITECTURE dff_asyn_preset_clr OF dff IS
BEGIN
--WARNING:Xst:3001 - This design contains one or more registers or latches with an active asynchronous set and asynchronous reset.
-- While this circuit can be built, it creates a sub-optimal implementation in terms of area, power and performance.
-- For a more optimal implementation Xilinx highly recommends one of the following:
--1) Remove either the set or reset from all registers and latches if not needed for required functionality
--2) Modify the code in order to produce a synchronous set and/or reset (both is preferred)

PROCESS ( clk, nPRESET, nCLEAR )
       BEGIN
              IF (nPRESET = '0') THEN
                     Q <= '1';     --asinhrono postavljanje izhoda (preset)
              ELSIF (nCLEAR = '0') THEN
                     Q <= '0';     --asinhrono brisanje izhoda (clear)
              ELSIF rising_edge(clk) THEN
                     Q <= D;
              END IF;
       END PROCESS;
END dff_asyn_preset_clr;
```

```
ARCHITECTURE dff_fdc OF dff IS
BEGIN

PROCESS ( clk, nCLEAR )
	BEGIN
		IF (nCLEAR = '0') THEN
			Q <= '0';	--asinhrono brisanje izhoda (clear) - rezultat je primitiv FDC
		ELSIF rising_edge(clk) THEN
			Q <= D;
		END IF;
	END PROCESS;
END dff_fdc;

ARCHITECTURE dff_fdp OF dff IS
BEGIN

PROCESS ( clk, nPRESET )
	BEGIN
		IF (nPRESET = '0') THEN
			Q <= '1';	--asinhrono postavljanje izhoda (preset) - rezultat je primitiv FDP
		ELSIF rising_edge(clk) THEN
			Q <= D;
		END IF;
	END PROCESS;
END dff_fdp;

ARCHITECTURE dff_fds OF dff IS
BEGIN
PROCESS ( clk, nPRESET )
	BEGIN
		IF rising_edge(clk) THEN
			IF (nPRESET = '0') THEN
				Q <= '1';	--sinhrono postavljanje izhoda (set) - rezultat je primitiv FDS
			ELSE
				Q <= D;
			END IF;
		END IF;
	END PROCESS;
END dff_fds;
```

```
ARCHITECTURE dff_fdr OF dff IS
BEGIN

PROCESS ( clk, nCLEAR )
	BEGIN
		IF rising_edge(clk) THEN
			IF (nCLEAR = '0') THEN
				Q <= '0';	--sinhrono brisanje izhoda (reset) - rezultat je primitiv FDR
			ELSE
				Q <= D;
			END IF;
		END IF;
	END PROCESS;
END dff_fdr;

ARCHITECTURE dff_syn_set_rst OF dff IS
BEGIN
-- rezultat sinteze je FDR, vhod in reset priključèka sta izvedena and vrati z LUT2
PROCESS ( clk, nPRESET, nCLEAR )
	BEGIN
		IF rising_edge(clk) THEN
			IF (nPRESET = '0') THEN
				Q <= '1';	--sinhrono postavljanje izhoda (preset)
			ELSIF (nCLEAR = '0') THEN
				Q <= '0';	--sinhrono brisanje izhoda (clear)
			ELSE
				Q <= D;
			END IF;
		END IF;
END PROCESS;
END dff_syn_set_rst;
```

V zgornji tabeli so prikazani različni načini sinhronega in asinhronega postavljanja in brisanja izhoda D-FF. Določeni (`dff_asyn_preset_clr`) NISO primerni za uporabo v Xilinx FPGA vezjih, saj nimajo ustreznega primitiva za realizacijo D-FF. Določeni so počasni, ker zahtevajo sinhrono izvedbo brisanja in postavljanja (`dff_syn_set_rst`). V nadaljevanju zato uporabljajte samo tiste izvedbe, ki se sintetizirajo v enega izmed primitivov, ki so na voljo.

2. Preverite tehnološke izvedbe posamezne arhitekture iz prejšnje tabele izvedb D-FF. Če ima entiteta več arhitektur, lahko ciljno arhitekturo, ki jo želite ***implementirati***, zapišete/kopirate na konec datoteke. Xilinx ISE upošteva zadnjo navedeno arhitekturo v VHD datoteki. Oglejte si poročilo sinteze in opozorila sintetizatorja, glede na izdelano vrsto D flip-flopa.

   Nato izdelajte datoteko testnih vrednosti (**`dff_tb.tbw`**) in s simulacijo pravilnost delovanja za vpis podatka '0' in '1'. Če ima entiteta več arhitektur, lahko arhitekturo, ki jo želite ***simulirati***, izberete v simulacijski datoteki s konfiguracijskim stavkom FOR USE (**`for all: ime_komponenete use entity work.ime_komponenete(ime_arhitekture);`**). Stavek FOR USE postavite med IS in BEGIN datoteke testnih vrednosti.

3. V arhivu predloge naloge se nahaja datoteka **`muxnto1.vhd`**. V to datoteko kopirajte entiteto in arhitekturo enote podanega splošnega n-naslovnega izbiralnika (multiplekserja). Ime entitete je: **`muxnto1`**. V podani entiteti je prikazana uporaba operatorja potenciranja (2^{n_addr}) za splošne strukture. Izdelani izbiralnik je (**`n_addr`**) naslovni, torej ima 2^{n_addr} vhodov.
   Ne pozabite vključiti knjižnice (**`ieee.numeric_std.all`**), sicer funkcija (**`to_integer`**) ne deluje.

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

ENTITY muxnto1 IS
generic( n_addr: natural := 2 ); -- število naslovov izbiralnika
PORT (      s : in std_logic_vector(n_addr - 1 downto 0); -- vektor naslovov
            w : in std_logic_vector(2**n_addr - 1 downto 0); --vektor podatkovnih vhodov
            f : OUT STD_LOGIC -- izhod izbiralnika
);
END muxnto1;
ARCHITECTURE ideal OF muxnto1 IS
BEGIN
      f <= w(to_integer(unsigned(s)));
END ideal;
```

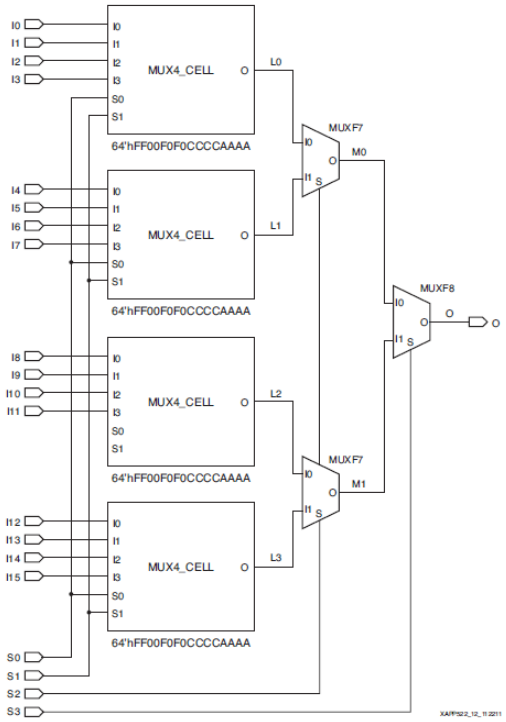
   Za posplošitev je v deklaraciji entitete parameter **`generic(n_addr: natural :=2);`** ki podaja število naslovov izbiralnika.

4. Izdelajte datoteko testnih vrednosti (**`muxnto1_tb.tbw`**) in s simulacijo preverite pravilnost delovanja za 2-naslovni izbiralnik (MUX 4/1). Če entiteto implementirate, se v sporočilih pojavijo opozorila:

   ```
   at 0 ps, Instance /muxnto1_tb/UUT/ : Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
   ```

   Opozorila izvirajo iz sintetizatorja, ker se argument funkcije (**`to_integer`**) ne ovrednoti takoj, zato sintetizator privzame vrednost (`0`).

Postavite parameter **`(n_addr:=4)`** in si oglejte nastalo tehnološko shemo na primeru vezja Artix7.



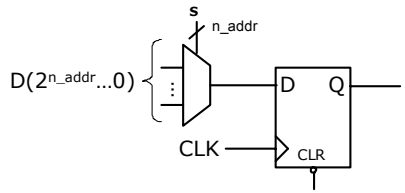
Slika 1: Izvedba izbiralnika 16/1 v vezju Artix 7.

Na levi sliki je prikazan detajl izvedbe izbiralnika 16/1, ki je sintetiziran s štirimi LUT6 tabelami, ki realizirajo izbiralnike 4/1. Izhodi teh izbiralnikov 4/1 so povezani preko kaskade dveh izbiralnikov 2/1 MUXF7 in končnega izbiralnika 2/1 MUXF8. Izbiralnik 16/1 je največ, kar lahko izvedemo znotraj ene rezine sliceL v vezju Artix7. Večje izbiralnike realiziramo kaskadno z uporabo več rezin sliceL.

Izbiralnik 4/1 je izveden z eno LUT6 tabelo, ki je inicializirana z: `0xFF00F0F0CCCCAAAA`. Pri tem sta naslovna vhoda na priključkih I_4, I_5, podatkovni vhodi pa so 00: I_0, 01: I_1, 10: I_2, 11:I_3.

Podrobnejši opis povezovanja in sinteze splošnih N/1 izbiralnikov v FPGA vezjih.

5. V arhivu predloge naloge se nahaja datoteka **`muxdff.vhd`**. V tej datoteki programirajte strukturo univerzalnega logičnega modula (ULM) na sliki 1.



Slika 2: Splošna ULM struktura, sestavljena iz parametriziranega izbiralnika N/1 in D−FF.

Komponenti D-FF in izbiralnika N/1 povežite neposredno (ne s povezovalnim stavkom) po podani entiteti:

```
ENTITY muxdff IS
generic( n_addr: natural := 2 );       -- stevilo naslovov ULM strukture
PORT (   S : in std_logic_vector(n_addr - 1 downto 0);
         D : in std_logic_vector(2**n_addr - 1 downto 0); -- vektor vhodov ULM strukture
         clk, -- signal ure (prožen na sprednjo fronto)
         nCLEAR : IN std_logic; -- signal za asinhrono brisanje (nCLEAR = '0' se postavi Q=>'0')
         Q : out std_logic -- izhod ULM strukture
);
END muxdff;
```

V podani entiteti smo ***odstranili*** signal **`nPreset`**, saj v Xilinx FPGA ne moremo učinkovito realizirati asinhronega postavljanja in brisanja naenkrat (glej opozorilo v tabeli realizacij D-FF). Realiziramo lahko samo enega - odločimo se za brisanje (**`nCLEAR`**). Iz nabora primitivov v danem FPGA vezju izberemo tak D-FF, ki ima asinhrono brisanje - to je primitiv FDC. Pred uporabo FDC moramo vključiti knjižnico UNISIM (`Library UNISIM;use UNISIM.vcomponents.all;`). Način klicanja je:

```
      U1: FDC
            generic map ( INIT => '0')      -- Initial value of register ('0' or '1')
            port map (   Q => Q,            -- Data output
                         C => clk,          -- Clock input
                         CLR => nCLEAR,     -- Asynchronous clear input
                         D => muxout);      -- Data input
```

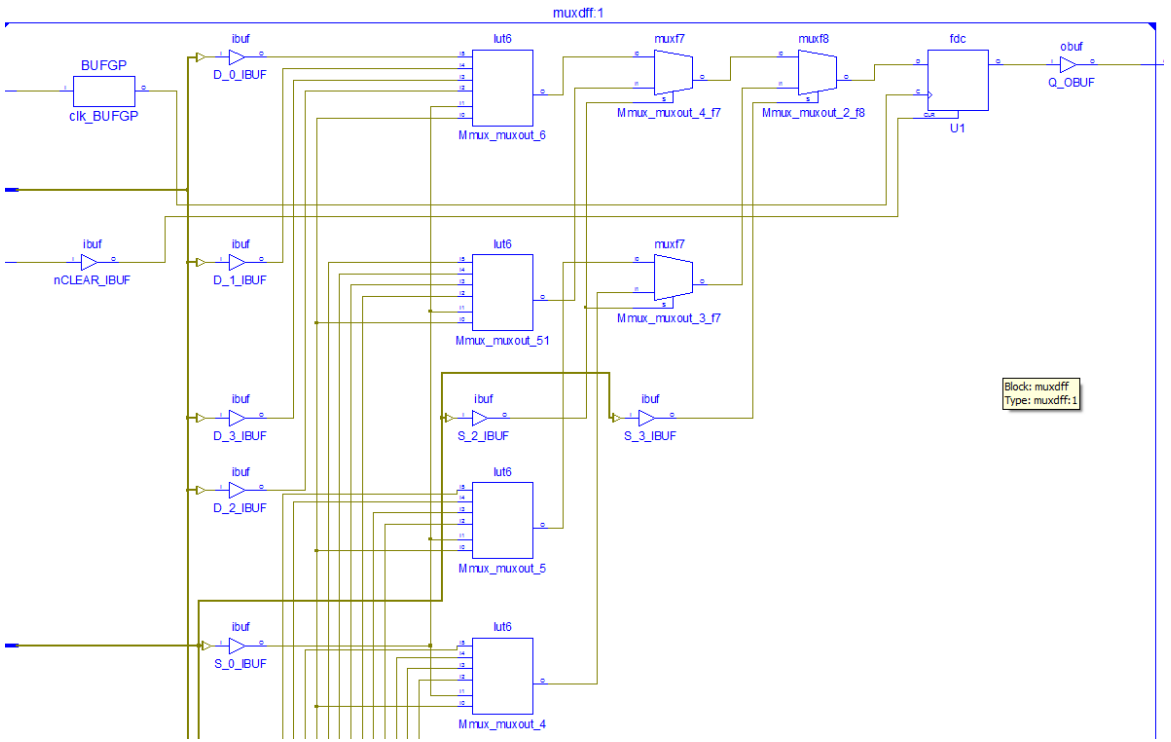
Postavite parameter velikosti izbiralnika `(n_addr:=4)` in si oglejte porabo osnovnih elementov (ang. basic elements BEL) v poročilu sintetizatorja na primeru vezja Artix7. Spartan 6 nima elementa MUXF8, zato bo poročilo nekoliko drugačno.

```
Primitive and Black Box Usage:
------------------------------
# BELS                               : 7
#      LUT6                          : 4
#      MUXF7                         : 2
#      MUXF8                         : 1
# FlipFlops/Latches                  : 1
#      FDC                           : 1
# Clock Buffers                      : 1
#      BUFGP                         : 1
# IO Buffers                         : 22
#      IBUF                          : 21
#      OBUF                          : 1

Device utilization summary:
---------------------------

Selected Device : 7a100tcsg324-3

Slice Logic Utilization:
 Number of Slice LUTs:  4  out of  63400
0%
    Number used as Logic: 4  out of  63400
0%
```



Slika 3: Izvedba ULM z 16/1 izbiralnikom v enem sliceL v Artix7.

6. V arhivu predloge naloge se nahaja datoteka **shift_reg.vhd**. V tej datoteki z uporabo povezovanja več (**reg_size**) ULM struktur realizirajte arhitekturo pomikalnega registra, ki deluje podobno kot TTL pomikalni register 74194:

```
entity shift_reg is
            generic( reg_size: natural := 4);   --velikost registra
            PORT (clk, -- signal ure (prozen na sprednjo fronto)
                  nCLR, -- signal za asinhrono brisanje (vsebina registra gre na 0)
                  sr_in,      -- zaporedni vhod za pomikanje desno (takrat gre sr_in->MSB)
                  sl_in : IN std_logic;   -- zaporedni vhod pri pomikanju levo (takrat gre sl_in->LSB)
                  s : in std_logic_vector(1 downto 0);      -- izbira operacije pomikalnega registra
                  x : in std_logic_vector(reg_size - 1 downto 0); -- vhod za vzporedno nalaganje
                  Q : out std_logic_vector(reg_size - 1 downto 0) -- vzporedni izhod registra
                  );
end shift_reg;
```

Deklarirajte komponento izdelanega ULM modula. Pomikalni register ima 2-bitni funkcijski vhod **s**, ki določa operacijo pomikalnega registra:

```
--   Postavitev mest v pomikalnem registru je:
--   (MSB mesto registra je fizično skrajno levo, LSB mesto je fizično skrajno desno)
-- S1 S0
-- 1  1    : Vzporedno nalaganje x => Q
-- 1  0    : Pomikanje levo     (v smeri od LSB do MSB, takrat gre sl_in->LSB)
-- 0  1    : Pomikanje desno    (v smeri od MSB do LSB, takrat gre sr_in->MSB)
-- 0  0    : Držanje vsebine
```

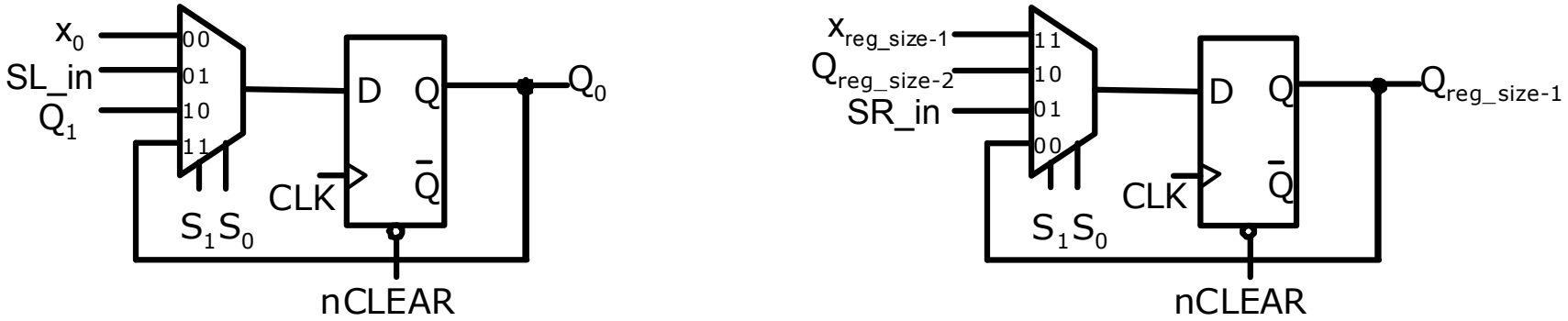
7. Povežite **reg_size** komponent ULM modula s podano entiteto pomikalnega registra z uporabo povezovalnega (**PORT MAP**) stavka znotraj FOR ... GENERATE stavka. Pri povezovanju definirajte vmesni signal (**D**), katerega *tip* je dvodimenzionalno polje:

```
type dff_mux_in is array (reg_size - 1 downto 0) of std_logic_vector(3 downto 0);
```

Opisani *tip signala* predstavlja dvodimenzionalno polje, ki je urejeno kot enodimenzionalno polje (**reg_size**) 4-bitnih elementov tipa (**std_logic_vector**).
Vsak vhod ULM se veže na naslednje mesto glede na funkcijo, ki jo pomikalni register opravlja. Zgled povezovanja ULM v 4-bitni pomikalni register je naloga 31 v Zbirki rešenih nalog pri predmetu NDV.

8. LSB mesto pomikalnega registra sestavite z uporabo operatorja sestavljanja (**&**) kot je prikazano po spodnjem zgledu:

```
D(0) <=     x(0)  &     -- (operacija 11) - nalaganje vhoda x(LSB)
            sl_in &     -- (operacija 10) - pomik levo (sl vhod gre v mesto LSB)
            Q_sig(1) &  -- (operacija 01) - pomik desno (mesto 1 gre v mesto LSB)
            Q_sig(0);   -- (operacija 00) - držanje vsebine LSB
```



Slika 4: Povezovanje ULM na LSB mesto (levo) in MSB mesto (desno) pomikalnega registra.

9. Podobno zapišite določitveni izraz za MSB mesto po zgornji sliki.
   Za preostala mesta (`1 ... reg_size - 2`) definirajte zanko **`for ... generate`** v kateri se nahaja povezovalni stavek:

```
U0: muxdff generic map (n_addr => 2) port map ( S, D(i), clk, nCLR, Q_sig(i));
```

   V povezovalnem stavku postavite signal **`nPRESET='1'`**. Parameter števila naslovov za izbiralnik ULM enote (**`n_addr`**) postavite na 2, saj ima register 4 operacije.

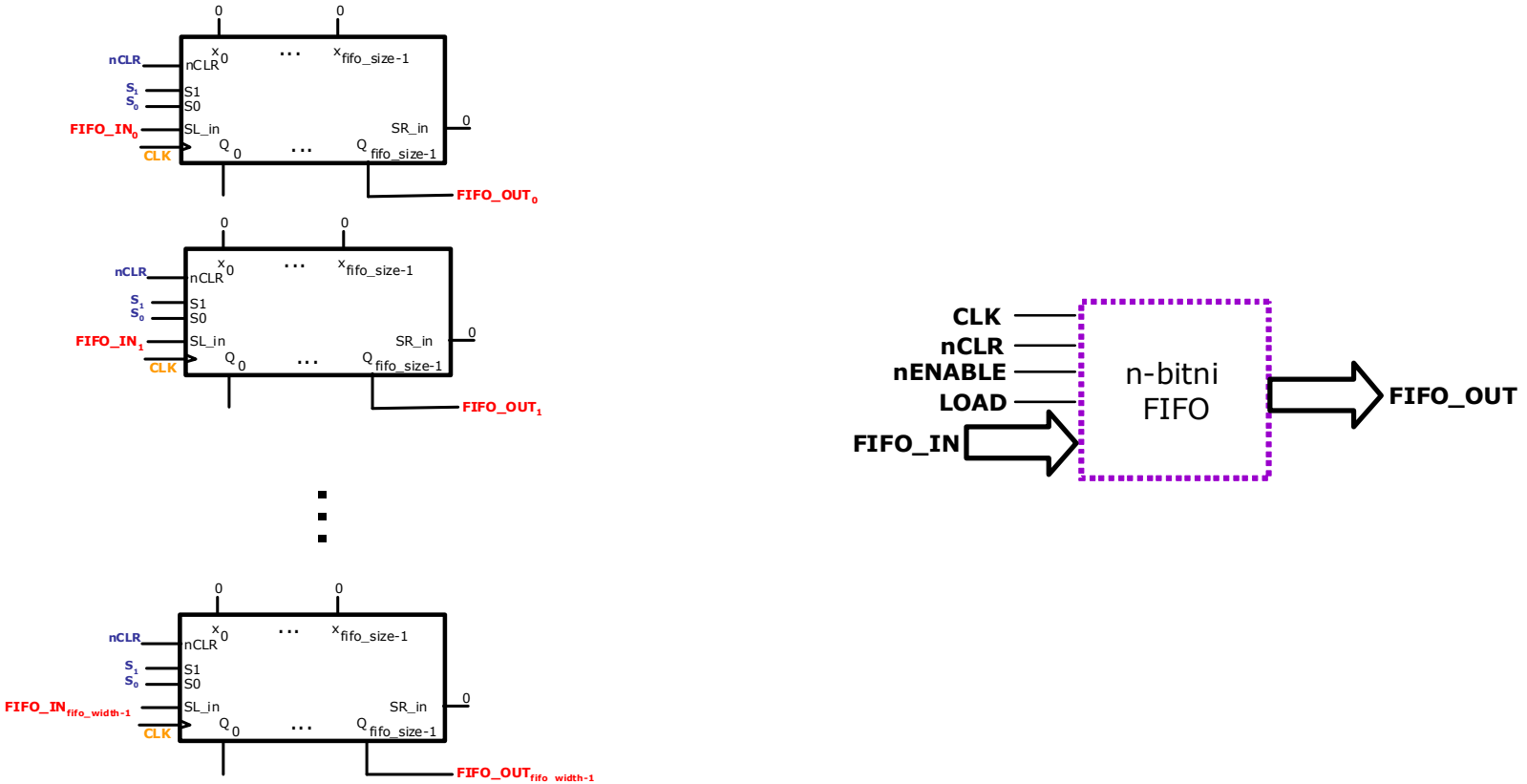
10. Izdelajte datoteko testnih vrednosti (**`shift_reg_tb.tbw`**) in s simulacijo preverite pravilnost delovanja vseh operacij pomikalnega registra (pomik levo in desno, držanje vsebine, nalaganje vsebine).

11. V arhivu predloge naloge se nahaja datoteka **`fifo.vhd`**. V tej datoteki z uporabo povezovanja več (**`fifo_width`**) izdelanih pomikalnih registrov realizirajte arhitekturo FIFO vmesnika:

```
entity fifo is
      generic(    fifo_width: natural := 4;      -- dolžina vhodnega podatka
                  fifo_size: natural := 8);      -- število hranjenih podatkov
      PORT (      clk,          -- signal ure (spremembe na sprednjo fronto)
                  nCLR,         -- signal za asinhrono brisanje (vsebina FIFO gre na 0)
                  nEnable,      --   signal za omogočanje FIFO ('0'-> omogočen vpis, '1' -> ohranja stanje)
                  LOAD : IN std_logic;    -- signal za omogočanje nalaganja ('1'-> fifo_in se vpiše)
                  fifo_in : in std_logic_vector(fifo_width - 1 downto 0);      -- vhodni podatek
                  fifo_out : out std_logic_vector(fifo_width - 1 downto 0)     -- izhodni podatek
            );
end fifo;
```

Deklarirajte komponento izdelanega pomikalnega registra. Od vseh operacij pomikalnega registra sta za FIFO vmesnik uporabni samo pomik levo (**`'10'`**) in ohranjanje vsebine (**`'00'`**). FIFO vmesnik bo vpisal nov podatek (pomakne vsebino eno mesto levo), ko je omogočen (**`nEnable = '0'`**) *in* ko je signal za nalaganje aktiven (**`LOAD = '1'`**), sicer vsebino ohranja. Krmilni signal za način delovanja pomikalnih registrov (**`s`**) zapišite z uporabo (**`when ... else`**) stavka ali kot logično funkcijo (**`and, or, not ...`**).

12. Za povezovanje (**`fifo_width`**) registrov (**`shift_reg`**) definirajte zanko **`for ... generate`** v kateri se nahaja povezovalni stavek:
    `port map (clk, nCLR, '0' , fifo_in(i), s, zeroes, Q(i));`
    V povezovalnem stavku postavite zaporedni vhod za pomik desno **`sr_in='0'`**, na vhod za vzporedno nalaganje pa priključite splošno konstanto nič (**`zeroes`**). Tako konstanto ste definirali pri domači nalogi aritmetično-logične enote. Poleg povezovalnega stavka znotraj **`for ... generate`** zanke tvorite tudi elemente izhoda FIFO strukture (**`fifo_out`**). Do posameznega elementa dvodimenzionalnega polja dostopate z definiranjem vseh indeksov polja elementa:
    `fifo_out(i) <= Q(i)(fifo_size - 1);`
    S prikazanim izrazom povežemo i-ti element izhoda FIFO strukture z MSB elementom i-tega pomikalnega registra v povezovalnem stavku.



Slika 5: Povezovalna shema pomikalnih registrov v strukturo FIFO vmesnika (levo) in končna entiteta FIFO strukture (desno).

13. Izdelajte datoteko testnih vrednosti (**`fifo_tb.tbw`**) in s simulacijo preverite pravilnost delovanja 4-bitnega FIFO vmesnika velikosti 8 zlogov (brisanje, omogočanje, vpis vsebine in pomikanje vsebine v FIFO vmesniku).

Razhroščevanje vsebine FIFO strukture:

Če želite sproti izpisovati vrednost dvodimenzionalnega polja (**Q**), ki je organizirano kot enodimenzionalno polje elementov (std_logic_vector) v oknu simulatorja iSim, v datoteko (**fifo.vhd**) vstavite spodnji procesni stavek.

```
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;
```

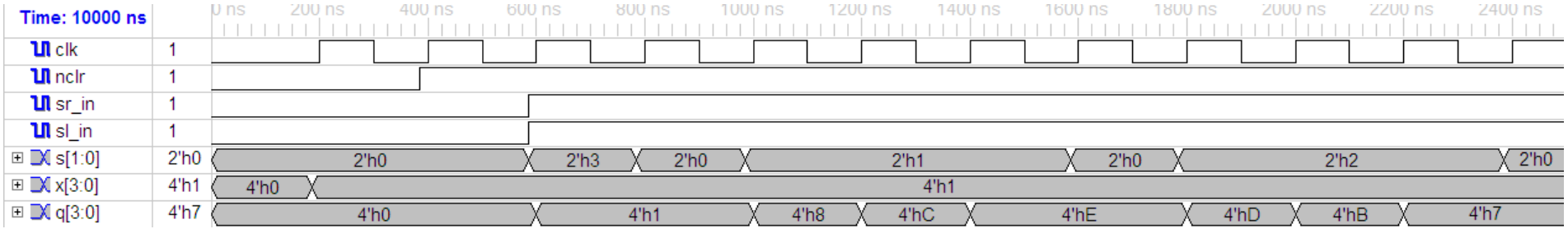
Izraz (report) bo v konzolnem oknu simulatorja iSim izpisoval vrednosti spremenljivk, vedno ko se spremeni vrednost spremenljivke (**Q**) v spodnji obliki:

```
at 1 us(2): Note: 0000    0000   0000   0000   0000   0000   0100   0001    (/fifo_tb/UUT/\l1(1)\/U0/).
```

V zgornji obliki je najprej povzet absolutni čas simulacije, nato sledi vsebina spremenljivke Q, ki je izpisana s TAB ločilom. Na koncu je podana enota, ki je dani izpis sprožila - torej v datoteki **fifo_tb**, enota **UUT**, (**for - generate**) zanka z imenom **l1** v prvi iteraciji (1) - instanca vezja U0. Če sledite po navodilih vezja nazaj, lahko ugotovite, da izpis izvira iz enote (**U0: muxdff**).

Rezultati simulacij:

Simulacija delovanja pomikalnega registra:



Simulacija delovanja FIFO vmesnika: