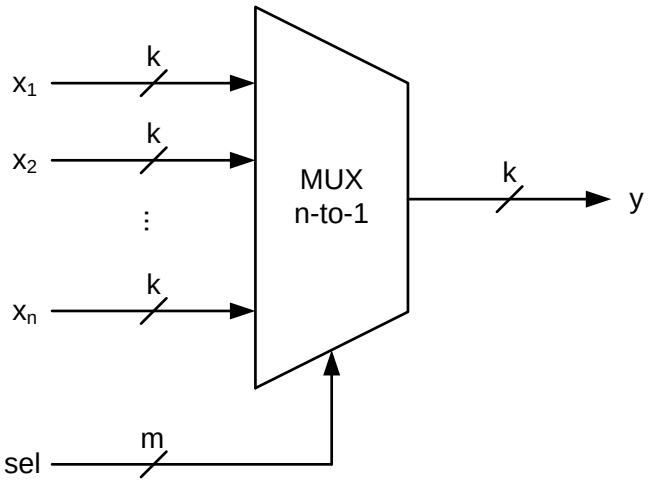
Laboratorijska vežba 2

Opis standardnih kombinacionih mreža korišćenjem VHDL jezika - 1

Multiplekser

Multiplekser je kombinaciona mreža sa *n* ulaza za podatke, x_1, x_2, ..., x_n, jednog selekcionog ulaza, *sel*, i jednim izlazom za podatke, *y*. Širina svakog od *n* ulaza za podatke je jednaka i iznosi *k* bita. Širina izlaza za podatke mora biti jednaka širini ulaza za podatke, *k* bita. Širina selekcionog ulaza jednaka je *m* bita i mora biti dovoljno velika da omogući jednoznačnu selekciju svakog od raspoloživih *n* ulaza za podatke. Interfejs generičkog multipleksera prikazan je na slici 1.



Slika 1.Interfejs generičkog multipleksera

U zavisnosti od vrednosti selekcionog ulaznog porta (*sel*) na izlaz multipleksera *y* prosleđuje se odgovarajuća vrednost koja se u tom trenutku nalazi na ulazu x_i čiji indeks *i* odgovara tekućoj numeričkoj vrednosti selekcionog ulaza.

Multiplekser najčešće ima broj ulaza za podatke koji je jednak stepenu broja 2, $n=2^m$. U tom slučaju je širina selekcionog ulaza jednaka *m*. U opštem slučaju, multiplekser sa n ulaza zahteva selekcioni ulaz širine $m = \lceil log_2 n \rceil$ bita.

VHDL modeli multipleksera

Multiplekser se u VHDL-u može modelovati na razne načine. Da bi smo ih ilustrovali koristićemo 1-bitni 2-na-1 multiplekser. Prilikom razvoja VHDL modela prvi korak je da se deklariše odgovarajući entitet, korišćenjem VHDL *entity* deklaracije. U našem primeru 1-bitni multiplekser 2-na-1 ima sledeće portove:

- dva 1-bitna ulazna porta za podatke, x_1 i x_2
- jedan 1-bitni selekcioni ulaz, *sel*
- jedan 1-bitni izlaz za podatke, *y*

Odogovarajuća *entity* deklaracija 1-bitnog multipleksera 2-na-1 ima sledeći izgled

```
library ieee;
use ieee.std_logic_1164.all;

entity mux2na1 is
        port (x1:       in std_logic;   -- ulazni port podataka 1
              x2:       in std_logic;   -- ulazni port podataka 2
              sel:      in std_logic;   -- selekcioni ulaz
              y:        out std_logic);-- izlazni port podataka
end entity mux2na1;
```

U ovom primeru odlučili smo se za korišćenje *std_logic* tipa podataka za sve portove entiteta mux2na1. Kao što je ranije rečeno, *std_logic* tip je preporučeni tip podataka za modelovanje električnih signala unutar VHDL modela. Umesto ovog tipa podataka mogli smo na primer koristiti i tip *bit*. Ukoliko bi smo se odlučili za korišćenje tipa bit za modelovanje ulaznih i izlaznih portova multipleksera, odgovarajuća entity deklaracija bi imala sledeći izgled.

```
entity mux2na1 is
        port (x1:       in bit;         -- ulazni port podataka 1
              x2:       in bit;         -- ulazni port podataka 2
              sel:      in bit;         -- selekcioni ulaz
              y:        out bit);       -- izlazni port podataka
end entity mux2na1;
```

Nakon što smo opisali interfejs multiplekera, potrebno je modelovati njegovu funkcionalnost. Za ovo je potrebno napisati odgovarajuće arhitekturno telo. Obzirom da se multiplekser može modelovati na razne načine, možemo napisati veliki broj različitih arhitekturnih tela.

NAPOMENA: U narednim arhitekturnim telima pretpostavka je da su svi ulazni i izlazni portovi tipa *std_logic*.

Bihevijani modeli

Varijanta 1: Model multipleksera baziran na korišćenju konkurentne naredbe uslovne dodele vrednosti signalu

```
architecture beh1 of mux2na1 is
begin
        y <= x1 when sel = '0' else
             x2;
end architecture beh1;
```

Varijanta 2: Model multipleksera baziran na korišćenju konkurentne naredbe selektovane dodele vrednosti signalu

```
architecture beh2 of mux2na1 is
begin
        with sel select
                y <= x1 when ‘0’
                     x2 when others;
end architecture beh2;
```

Varijanta 3: Model multipleksera baziran na korišćenju *process* i *if* naredbe

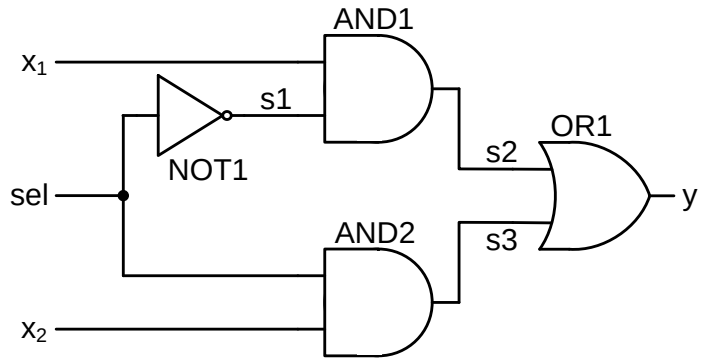
```
architecture beh3 of mux2na1 is
begin
        mux: process (x1, x2, sel) is
        begin
                if (sel = '0') then
                        y <= x1;
                else
                        y <= x2;
                end if;
        end process;
end architecture beh3;
```

Varijanta 4: Model multipleksera baziran na korišćenju *process* i *case* naredbe

```
architecture beh4 of mux2na1 is
begin
        mux: process (x1, x2, sel) is
        begin
                case sel is
                        when '0' =>
                                y <= x1;
                        when others =>
                                y <= x2;
                end case;
        end process;
end architecture beh4;
```

Strukturni model

Sve prethodne varijante VHDL modela 1-bitnog multipleksera 2-na-1 pripadaju klasi bihevijalnih (funkcionalnih) modela. Moguće je napisati i strukturni model multipleksera koji bi funkciju multipleksera opisao implicitno, modelujući njegovu unutrašnju strukturu korišćenjem standardnih logičkih kapija. Jedan mogući strukturni model 1-bitnog multipleksera 2-na-1 prikazan je na slici 2.



Slika 2.Strukturni model 1-bitnog multipleksera 2-na-1

VHDL omogućava pisanje i strukturnih modela. VHDL model prikazan u nastavku predstavlja model strukture sa slike 2 napisan korišćenjem VHDL jezika.

```
architecture struct of mux2na1 is
        signal s1, s2, s3: std_logic;
begin
        not1: entity work.not_gate(beh)
                port map (in1 => sel,
                          out1 => s1);

        and1: entity work.and_gate(beh)
                port map (in1 => x1,
                          in2 => s1,
                          out1 => s2);

        and2: entity work.and_gate(beh)
                port map (in1 => sel,
                          in2 => x2,
                          out1 => s3);

        or1:   entity work.or_gate(beh)
                port map (in1 => s2,
                          in2 => s3,
                          out1 => y);
end architecture struct;
```

NAPOMENA: Prethodni strukturni model 1-bitnog multipleksera 2-na-1 nije kompletan. Da bi se model kompletirao neophodno je obezbediti VHDL modele invertora kao i dvoulaznih I i ILI kola.

Kombinovanjem entity deklaracije i odgovarajućeg arhitekturnog tela dobijamo kompletan model 1-bitnog multipleksera 2-na-1. Na primer, dole je prikazan model 1-bitnog multipleksera 2-na-1 koji se bazira na funkcionalnom modelu koji koristi *process* i *case* naredbe.

```
library ieee;
use ieee.std_logic_1164.all;

entity mux2na1 is
        port (x1:       in std_logic;   -- ulazni port podataka 1
              x2:       in std_logic;   -- ulazni port podataka 2
              sel:      in std_logic;   -- selekcioni ulaz
              y:        out std_logic); -- izlazni port podataka
end entity mux2na1;


architecture beh4 of mux2na1 is
begin
        mux: process (x1, x2, sel) is
        begin
                case sel is
                        when '0' =>
                                y <= x1;
                        when others =>
                                y <= x2;
                end case;
        end process;
end architecture beh4;
```

Verifikaciono okruženje

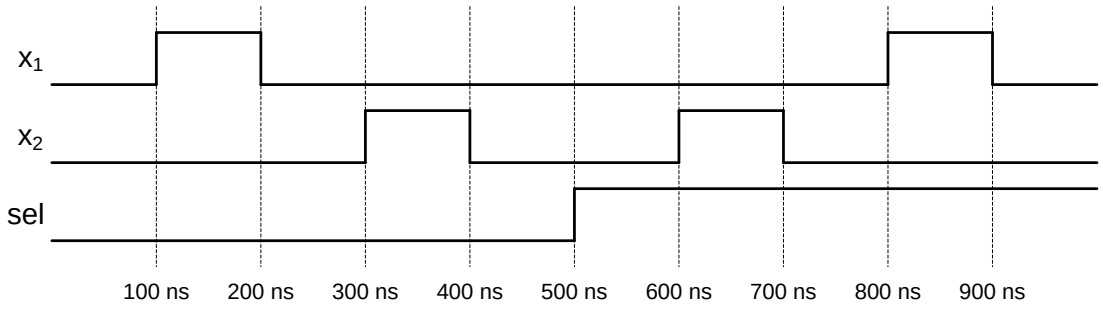
Nakon što smo napisali odgovarajući VHDL model neophodno je izvršiti njegovu verifikaciju kako bi smo utvrdili da li razvijeni model poseduje željenu funkcionalnost. Ovo se postiže simuliranjem VHDL modula korišćenjem odgovarajućeg verifikacionog okruženja. Verifikaciono okruženje sastoji se iz jednog ili više dodatnih VHDL modula čija je isključiva namena verifikacija funkcionalnosti nekog drugog VHDL modula. Verifikaciona okruženja po pravilu su vrlo kompleksna, te metodologija njihovog razvoja izlazi iz okvira ovog kursa. Na ovom mestu upoznaćemo se sa najjednostavnijim tipom verifikacionog okruženja koje se sastoji samo od jednog dodatnog VHDL modula, koji se zove testbenč (*testbench*). Primer jednog mogućeg testbenča za 1-bitni multiplekser 2-na-1 prikazan je u nastavku.

```
library ieee;
use ieee.std_logic_1164.all;

entity mux2na1_tb is
end entity mux2na1_tb;


architecture beh of mux2na1_tb is
   -- Deklaracija komponente VHDL modula koji se verifikuje (DUV)
   -- U ovom slučaju je to multiplekser 2-na-1
   component mux2na1 is
           port (x1:    in std_logic;   -- ulazni port podataka 1
                 x2:    in std_logic;   -- ulazni port podataka 2
                 sel:   in std_logic;   -- selekcioni ulaz
```

```
            y:      out std_logic);-- izlazni port podataka
    end component mux2na1;

    -- Deklaracija unutrašnjih signala potrebnih za povezivanje stimulus
    -- generatora sa ulazima DUV-a
    signal x1_s, x2_s, sel_s: std_logic;
    signal y_s: std_logic;

begin
        -- Komponenta koja se verifikuje
        duv: mux2na1
           port map (
              x1 => x1_s,
              x2 => x2_s,
              sel => sel_s,
              y => y_s);

        -- Stimulus generator koji generise potrebne vrednosti na
        -- ulaznim portovima DUV-a na osnovu kojih ce biti moguce
        -- proveriti da li DUV implementira potrebnu funkcionalnost
        stim_gen: process
        begin
                x1_s <= '0', '1' after 100 ns, '0' after 200 ns,
                        '1' after 800 ns, '0' after 900 ns;

                x2_s <= '0', '1' after 300 ns, '0' after 400 ns,
                        '1' after 600 ns, '0' after 700 ns;

                sel_s <= '0', '1' after 500 ns;

                wait;
        end process;
end architecture beh;
```

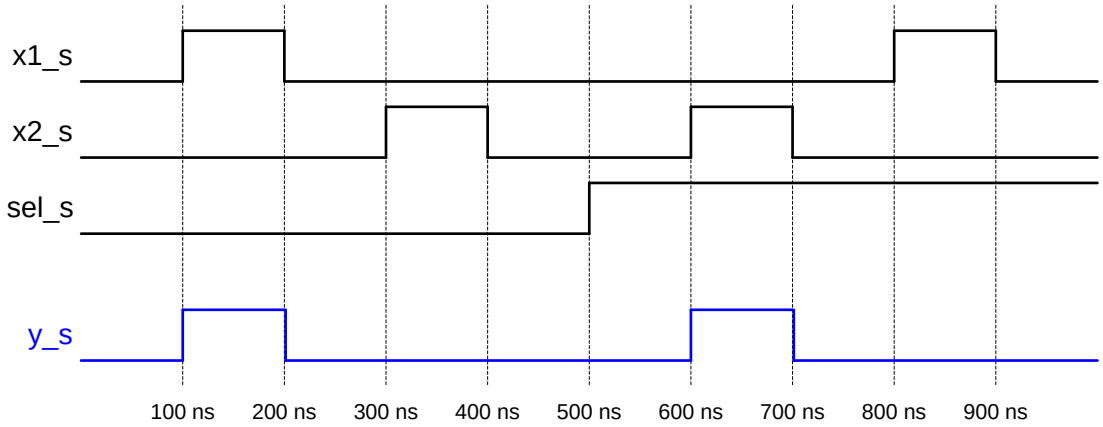
Prikazani testbenč instancionira komponentu 1-bitnog multipleksera 2-na-1 i koristeći *stim_gen* proces generiše povorku ulaznih signala koja može da se iskoristi za proveru ispravnog rada napisanog modela multipleksera.

Stim_gen proces generiše vremenske oblike za ukupno tri signala, *x1_s*, *x2_s* i *sel_s*, koristeći tri sekvencijalne naredbe dodele vrednosti signalu. Talasni oblici koji će biti generisani pomoću ove tri naredbe dodele vrednosti signalu prikazani su na slici 3.



Slika 3.Generisani talasni oblici na tri ulazna signala 1-bitnog multipleksera 2-na-1

Nakon što se izvrši simulacija kombinovanog rada testbenča i modela multipleksera talasni oblik izlaznog signala podataka multipleksera *y_s*, trebalo bi da izgleda kao na slici 4.



Slika 4.Generisani talasni oblici na tri ulazna signala 1-bitnog multipleksera 2-na-1 zajedno sa talasnim oblikom izlaznog signala podataka *y_s*, koji je dobijen kao rezultat simulacije

NAPOMENA: Razvijeno verifikaciono okruženje je univerzalno i može se koristiti za funkcionalnu verifikaciju bilo kojeg od predloženih modela 1-bitnog multipleksera 2-na-1.

Zadaci za vežbu

Zadatak 1:

Napisati potrebne VHDL modele invertora i dvoulaznih I i ILI logičkih kapija tako da se kompletira strukturni model 1-bitnog multipleksera 2-na-1. Izvršiti funkcionalnu verifikaciju razvijenog modela.

Zadatak 2:

Napisati bihevijanlni i strukturni VHDL model 1-bitnog multipleksera 4-na-1. Za razvijeni VHDL model napisati testbenč koji se može koristiti za proveru ispravnosti napisanih modela.

Zadatak 3:

Napisati bihevijalni VHDL model 16-bitnog multipleksera 4-na-1. Napisati testbenč koji se može iskoristiti za funkcionalnu verifikaciju razvijenog VHDL modela.

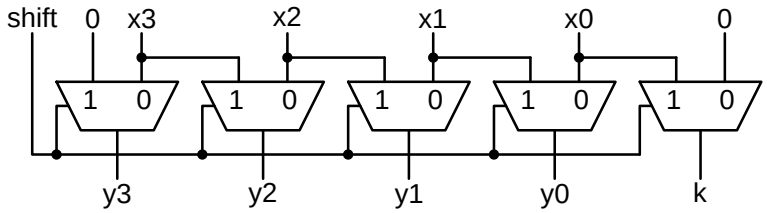
Zadatak 4:

Napisati strukturni VHDL model 1-bitnog multipleksera 8-na-1 koji će biti baziran na korišćenju odgovarajućeg broja 1-bitnih multipleksera 4-na-1 i potrebnih elementarnih logičkih kapija. Za razvijeni model napisati odgovarajući testbenč

pomoću kojega će biti moguće izvršiti proveru funkcionalnosti VHDL modela multipleksera 8-na-1.

Zadatak 5:

U digitalnim sistemima često je neophodno izvršiti pomeranje višebitnog vektora za jednu ili više pozicija u levo ili desno. Digitalno električno kolo koje može da izvrši pomeranje 4-bitnog vektora $X=x_3x_2x_1x_0$ za jedno mesto u desno, kada je ulazni port *shift* postavljen na 1, prikazano je na slici 5.



Slika 5.Kolo za pomeranje 4-bitnog vektora za jedno mesto u desno

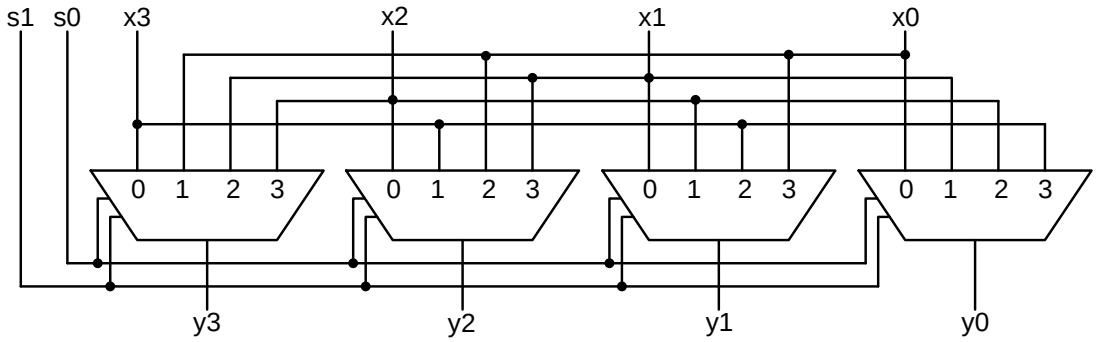
Kada je ulazni port *shift*=1, na izlaznim portovima y_i pojavljuju se sledeće vrednosti: *y3*=0, *y2=x3*, *y1=x2*, *y0=x1*, a na izlaznom portu *k* vrednost *k=x0*. U slučaju da je *shift*=0, na izlaznim portovima y_i pojavljuju se sledeće vrednosti: *y3*=x3, *y2=x2*, *y1=x1*, *y0=x0*, a na izlaznom portu *k* vrednost *k=0*.

a) Napisati strukturni VHDL model kola za pomeranje 4-bitnog vektora za jedno mesto u desno koristeći sliku 5. Za napisani VHDL model napisati testbenč koji se može iskoristiti za njegovu funkcionalnu verifikaciju.
b) Napisati bihevijalni VHDL model kola za pomeranje 4-bitnog vektora za jedno mesto u desno. Iskoristiti testbenč napisan pod a) za proveru ispravnosti rada bihevijalnog modela. Uporediti složenost strukturnog i bihevijalnog modela? Koji od njih je jednostavniji? Komentarisati dobijene rezultate.

Zadatak 6:

Kolo iz zadatka 4 pomera ulazni vektor za jedan bit u desno. Prazno mesto koje nastaje kao rezultat pomeranja popunjava se sa vrednošću 0. U praksi se često javlja potreba za mogućnošću pomeranja ulaznog vektora za više od jednog bita u istom trenutku. Prazna mesta koja pri tome nastaju se ili popunjavaju sa 0 ili sa bitovima koji su „ispali“. U ovom drugom slučaju kolo zapravo vrši rotaciju bitova ulaznog vektora za specificirani broj bita. Ovakvo kolo naziva se *Barrel Shifter*.

Barrel Shifter koji vrši rotaciju 4-bitnog ulaznog vektora za 0, 1, 2 ili 3 pozicije prikazan je na slici 6.



Slika 6. 4-bitni *Barrel shifter*

Funkcionalnost 4-bitnog Barrel shifter-a prikazana je u tabeli 1.

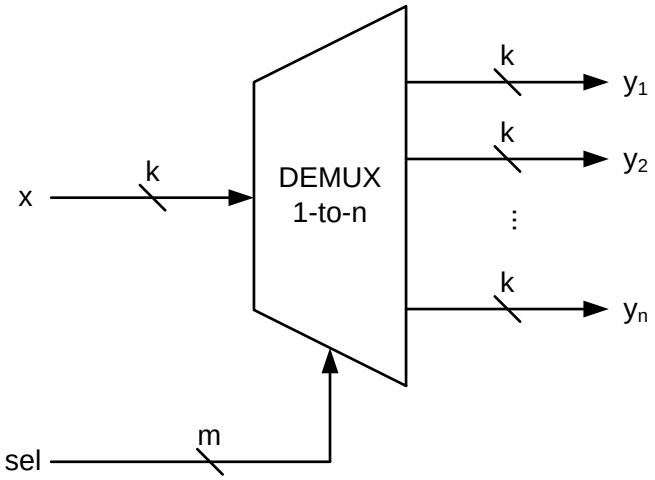
s1	s0	y3	y2	y1	y0
0	0	x3	x2	x1	x0
0	1	x0	x3	x2	x1
1	0	x1	x0	x3	x2
1	1	x2	x1	x0	x3

Tabela 1. Funkcionalnost 4-bitnog *Barrel shifter*-a

a) Napisati strukturni VHDL model 4-bitnog *Barrel shifter*-a koristeći sliku 6. Za napisani VHDL model napisati testbenč koji se može iskoristiti za njegovu funkcionalnu verifikaciju.
b) Napisati bihevijalni VHDL model 4-bitnog *Barrel shifter*-a. Iskoristiti testbenč napisan pod a) za proveru ispravnosti rada bihevijalnog modela. Uporediti složenost strukturnog i bihevijalnog modela? Koji od njih je jednostavniji? Komentarisati dobijene rezultate.

Demultiplekser

Demultiplekser je kombinaciona mreža koja realizuje funkciju koja je suprotna od funkcije multipleksera. Demultiplekser je kombinaciona mreža sa jednim ulazom za podatke, *x*, jednim selekcionim ulazom, *sel*, i *n* izlaza za podatke, y_1, y_2, ..., y_n. Širina ulaza za podatke je jednaka i iznosi *k* bita. Širina svakog od *n* izlaza za podatke mora biti jednaka širini ulaza za podatke, *k* bita. Širina selekcionog ulaza jednaka je *m* bita i mora biti dovoljno velika da omogući jednoznačnu selekciju svakog od raspoloživih *n* izlaza za podatke. Interfejs generičkog demultipleksera prikazan je na slici 7.



Slika 7.Interfejs generičkog demultipleksera

U zavisnosti od vrednosti selekcionog ulaznog porta (*sel*) na tačno jedan izlaz demultipleksera y_i, čiji indeks *i* odgovara tekućoj numeričkoj vrednosti selekcionog ulaza, prosleđuje se odgovarajuća vrednost koja se u tom trenutku nalazi na ulazu *x*.

Demultiplekser najčešće ima broj izlaza za podatke koji je jednak stepenu broja 2, $n=2^m$. U tom slučaju je širina selekcionog ulaza jednaka *m*. U opštem slučaju, demultiplekser sa n izlaza zahteva selekcioni ulaz širine $m = \lceil log_2 n \rceil$ bita.

VHDL modeli demultipleksera

Kao i u slučaju multipleksera i demultiplekser se u VHDL-u može modelovati na razne načine. Da bi smo ih ilustrovali koristićemo 1-bitni 1-na-2 demultiplekser koji ima sledeće portove:

- jedan 1-bitni ulaz za podatke, *x*
- jedan 1-bitni selekcioni ulaz, *sel*
- dva 1-bitna izlazna porta za podatke, y_1 i y_2

Odogovarajuća *entity* deklaracija 1-bitnog demultipleksera 1-na-2 ima sledeći izgled

```
library ieee;
use ieee.std_logic_1164.all;

entity demux1na2 is
        port (x:        in std_logic;     -- ulazni port podataka
              sel:      in std_logic;     -- selekcioni ulaz
              y1:       out std_logic;    -- izlazni port podataka 1
              y2:       out std_logic);   -- izlazni port podataka 2
end entity demux1na2;
```

Ukoliko bi smo se odlučili za korišćenje tipa bit za modelovanje ulaznih i izlaznih portova multipleksera, odgovarajuća entity deklaracija bi imala sledeći izgled.

```
entity demux1na2 is
        port (x:        in bit;     -- ulazni port podataka
              sel:      in bit;     -- selekcioni ulaz
              y1:       out bit;    -- izlazni port podataka 1
              y2:       out bit);   -- izlazni port podataka 2
end entity demux1na2;
```

Nakon što smo opisali interfejs demultiplekera, potrebno je modelovati njegovu funkcionalnost. Za ovo je potrebno napisati odgovarajuće arhitekturno telo. Obzirom da se demultiplekser može modelovati na razne načine, možemo napisati veliki broj različitih arhitekturnih tela.

NAPOMENA: U narednim arhitekturnim telima pretpostavka je da su svi ulazni i izlazni portovi tipa *std_logic*.

Bihevijani modeli

Varijanta 1: Model demultipleksera baziran na korišćenju konkurentne naredbe uslovne dodele vrednosti signalu

```
architecture beh1 of demux1na2 is
begin
        y1 <= x when sel = '0' else
              '0';

        y2 <= x when sel = '1' else
              '0';
end architecture beh1;
```

Varijanta 2: Model demultipleksera baziran na korišćenju konkurentne naredbe selektovane dodele vrednosti signalu

```
architecture beh2 of demux1na2 is
begin
        with sel select
                y1 <= x when ‘0’
                        ‘0’ when others;

        with sel select
                y2 <= x when ‘1’
                        ‘0’ when others;
end architecture beh2;
```

Varijanta 3: Model demultipleksera baziran na korišćenju *process* i *if* naredbe

```
architecture beh3 of demux1na2 is
begin
        mux: process (x, sel) is
        begin
                if (sel = '0') then
                        y1 <= x;
                        y2 <= '0';
                else
                        y1 <= '0';
                        y2 <= '0';
                end if;
        end process;
end architecture beh3;
```

Varijanta 4: Model demultipleksera baziran na korišćenju *process* i *case* naredbe

```
architecture beh4 of demux1na2 is
begin
        mux: process (x sel) is
        begin
                case sel is
                        when '0' =>
                                y1 <= x;
                                y2 <= '0';
                        when others =>
                                y <= '0';
                                y2 <= x;
                end case;
        end process;
end architecture beh4;
```

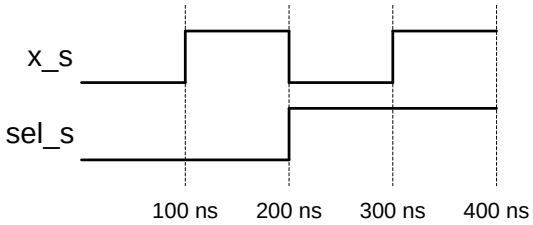
Verifikaciono okruženje

Primer jednog mogućeg testbenča za 1-bitni demultiplekser 1-na-2 prikazan je u nastavku.

```
library ieee;
use ieee.std_logic_1164.all;

entity demux1na2_tb is
end entity demux1na2_tb;


architecture beh of demux1na2_tb is
   -- Deklaracija komponente VHDL modula koji se verifikuje (DUV)
   -- U ovom slučaju je to demultiplekser 1-na-2
   component demux1na2 is
         port (x:      in std_logic;     -- ulazni port podataka
               sel:    in std_logic;     -- selekcioni ulaz
               y1:     out std_logic;    -- izlazni port podataka 1
               y2:     out std_logic);   -- izlazni port podataka 2
   end component demux1na2;

   -- Deklaracija unutrašnjih signala potrebnih za povezivanje stimulus
   -- generatora sa ulazima DUV-a
   signal x_s, sel_s: std_logic;
   signal y1_s, y2_s: std_logic;

begin
         -- Komponenta koja se verifikuje
         duv: demux1na2
            port map (
               x => x_s,
               sel => sel_s,
               y1 => y1_s,
               y2 => y2_s);

         -- Stimulus generator koji generise potrebne vrednosti na
         -- ulaznim portovima DUV-a na osnovu kojih ce biti moguce
         -- proveriti da li DUV implementira potrebnu funkcionalnost
         stim_gen: process
         begin
                  x_s <= '0', '1' after 100 ns, '0' after 200 ns,
                           '1' after 300 ns;

                  sel_s <= '0', '1' after 200 ns;

                  wait;
         end process;
end architecture beh;
```

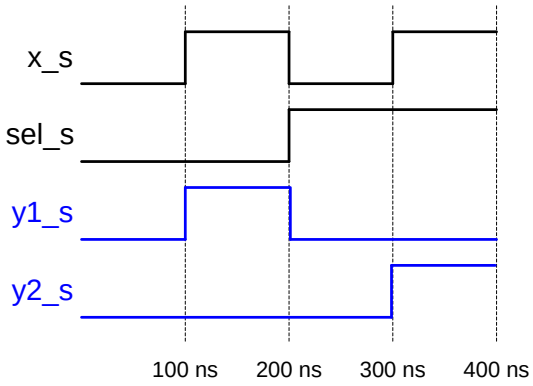
Prikazani testbenč instancionira komponentu 1-bitnog demultipleksera 1-na-2 i koristeći *stim_gen* proces generiše povorku ulaznih signala koja može da se iskoristi za proveru ispravnog rada napisanog modela demultipleksera.

Stim_gen proces generiše vremenske oblike za dva signala, *x_s*, i *sel_s*, koristeći dve sekvencijalne naredbe dodele vrednosti signalu. Talasni oblici koji će biti generisani pomoću ove dve naredbe dodele vrednosti signalu prikazani su na slici 3.



Slika 8.Generisani talasni oblici dva ulazna signala 1-bitnog demultipleksera 1-na-2

Nakon što se izvrši simulacija kombinovanog rada testbenča i modela demultipleksera talasni oblici izlaznih signala podataka demultipleksera *y1_s* i *y2_s* trebalo bi da izgledaju kao na slici 9.



Slika 9.Generisani talasni oblici na dva ulazna signala 1-bitnog demultipleksera 1-na-2 zajedno sa talasnim oblikom izlaznih signala podataka *y1_s* i *y2_s* koji su dobijeni kao rezultat simulacije

NAPOMENA: Razvijeno verifikaciono okruženje je univerzalno i može se koristiti za funkcionalnu verifikaciju bilo kojeg od predloženih modela 1-bitnog demultipleksera 1-na-2.

Zadaci za vežbu

Zadatak 1:

Nacrtati strukturni model 1-bitnog demultipleksera 1-na-2 baziran na korišćenju standardnih logičkih kapija. Na osnovu razvijenog strukturnog modela napisati odgovarajući strukturni VHDL model. Izvršiti funkcionalnu verifikaciju razvijenog VHDL modela.

Zadatak 2:

Napisati VHDL model 1-bitnog demultipleksera 1-na-4 koji će pored ulaznog porta za podatke, ulaznog selekcionog porta i odgovarajućeg broja izlaznih portova za podatke imati i dodatni port dozvole rada, *en*. Kada je vrednost porta dozvole rada *en* jednaka 0, demultiplekser treba da je neaktivan, odnosno na svim izlaznim portovima za podatke potrebno je da se nalaze vrednosti 0, bez obzira na trenutnu vrednost ulaznog porta za podatke. Kada je vrednost porta *en* jednaka 1, demultiplekser treba da obavlja svoju normalnu funkciju. Napisati odgovarajući testbenč pomoću kojega je moguće izvršiti funkcionalnu verifikaciju razvijenog VHDL modela.

Zadatak 3:

Napisati VHDL model 16-bitnog demultipleksera 1-na-8. Napisati testbenč koji se može iskoristiti za funkcionalnu verifikaciju razvijenog VHDL modela.