Laboratorijska vežba 5

Opis standardnih sekvencijalnih mreža korišćenjem VHDL jezika - 2

Registri

Flip flop je u stanju da skladišti samo jedan bit informacije. Kada se *n* flip flopova posmatra kao celina koja je u stanju da skladišti *n* bita informacije, tada taj skup flip flopova nazivamo *registrom*.

U zavisnosti od toga kako se podaci upisuju i čitaju iz registra, kao i da li se eventualno vrše neke transformacije nad njima, postoji veliki broj različitih tipova registara:

- *n*-bitni registar sa paralelnim upisom i paralelnim čitanjem
- *n*-bitni registar sa serijskim upisom i paralelnim čitanjem
- *n*-bitni registar sa paralelnim upisom i serijskim čitanjem
- *n*-bitni registar sa serijskim upisom i serijskim čitanjem
- *n*-bitni pomerački registar
- *n*-bitni registar sa rotacijom

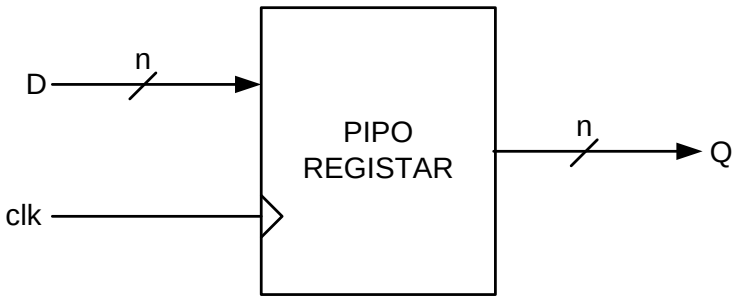
Pored ulaznih i izlaznih portova za podatke, registri mogu posedovati i čitav niz dodatnih portova:

- ulazni port dozvole rada registra - *en* (*enable*)
- *clock enable* (*ce*) signal za selekciju rastućih, odnosno opadajućih, ivica *clk* ulaza na koje registar treba da reaguje
- ulazni port dozvole upisa novog sadržaja u registar - *load* ili *we* (*write enable*)
- ulazni port dozvole čitanja trenutnog sadržaja iz registra - *read* ili *re* (*read enable*)
- ulazni portove za inicijalizaciju sadržaja registra - *reset, set, clear, preset*

Većina navedenih dodatnih portova su sinhroni (osim *clear* i *preset* portova koji su asinhroni), međutim mogu postojati asinhrone verzije ovih portova.

Registri sa paralelnim upisom i paralelnim čitanjem

Registri sa paralelnim upisom i paralelnim čitanjem (PIPO) su najčešće korišćeni tip ragistara. Osnovni interfejs *n* bitnog registra sa paralelnim upisom i čitanjem prikazan je na slici 1.



Slika 1. Interfejs *n* bitnog registra sa paralelnim upisom i čitanjem

N bitni registar sa paralelnim upisom i čitanjem ima dva ulazna porta:

- *D* ulaz za podatke, širine *n* bita
- *clk* ulaz za sinhronizaciju rada registra

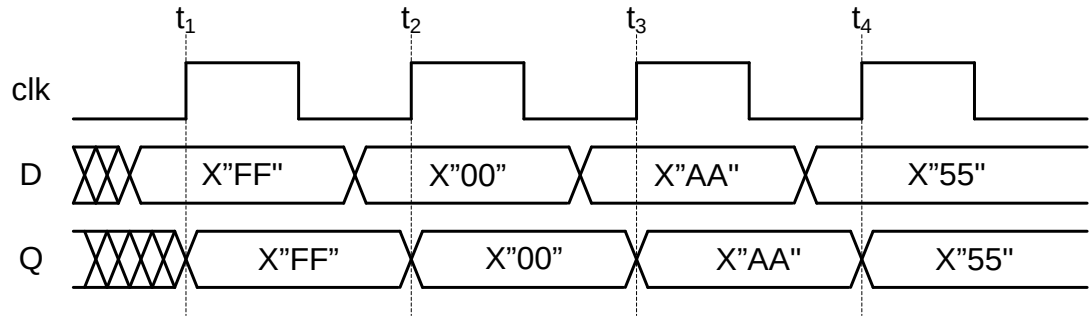
N bitni registar sa paralelnim upisom i čitanjem ima jedan izlazni port:

- *Q* izlaz za podatke, širine *n* bita

Funkcionalnost registra sa paralelnim upisom i čitanjem može se prikazati sledećom tabelom.

clk	*D*	*Q*(t+1)
0	x	*Q*(t)
1	x	*Q*(t)
↓	x	*Q*(t)
↑	*data*	*data*

Kao što se može primetiti na osnovu prethodne tabele, sve dok se na clk ulazu ne pojavi rastuća ivica registar čuva prethodno upisanu vrednost i ignoriše vrednosti koje mu se trenutno nalaze na *D* ulazu. Tek kada se na *clk* ulazu pojavi "sinhronizacioni događaj" (rastuća ili opadajuća ivica), u registar se upisuje trenutna vrednost koja se nalazi na *D* ulazu. Bitno je naglasiti da se ovaj upis obavlja paralelno, u svaki od unutrašnjih flip flopova koji čine registar se istovremeno upisuje odgovarajući bit ulaznog porta *D*. Takođe, podaci koji su trenutno upisani u unutrašnje flip flopove se istovremeno (paralelno) pojavljuju na izlaznom portu *Q*. Zbog toga se ovaj registar i zove registar sa paralelnim upisom i čitanjem podataka. Na primer, vremenski dijagram rada 8-bitnog registra sa paralelnim upisom i čitanjem prikazan je na slici 2.



Slika 2. Vremenski dijagram rada 8-bitnog registra sa paralelnim upisom i čitanjem

VHDL modeli n-bitnog registra sa paralelnim upisom i čitanjem

Da bi smo ilustrovali različite načine modelovanja n-bitnog registra sa paralelnim upisom i čitanjem, koristićemo 4-bitni registar. *Entity* deklaracija 4-bitnog registra sa paralelnim upisom i čitanjem ima sledeći izgled

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

entity reg4 is
   port (clk: in std_logic;
         d:   in std_logic_vector(3 downto 0);  -- ulazni port podataka
         q:   out std_logic_vector(3 downto 0) -- izlazni port podataka
        );
end entity reg4;
```

Nakon što smo opisali interfejs 4-bitnog registra sa paralelnim upisom i čitanjem, potrebno je modelovati njegovu funkcionalnost. Za ovo je potrebno napisati odgovarajuće arhitekturno telo.

Bihevijani model

Varijanta 1: Model 4-bitnog registra sa paralelnim upisom i čitanjem baziran na korišćenju *process* i *if* naredbe

```
architecture beh1 of reg4 is
begin
        reg: process (clk) is
        begin
                if (clk'event and clk = '1') then
                        q <= d;
                end if;
        end process;
end architecture beh1;
```

Kombinovanjem entity deklaracije i odgovarajućeg arhitekturnog tela dobijamo kompletan model 4-bitnog registra sa paralelnim upisom i čitanjem.

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

entity reg4 is
   port (clk: in std_logic;
         d:   in std_logic_vector(3 downto 0);  -- ulazni port podataka
         q:   out std_logic_vector(3 downto 0) -- izlazni port podataka
        );
end entity reg4;
```

```
architecture beh1 of reg4 is
begin
        reg: process (clk) is
        begin
                if (clk'event and clk = '1') then
                        q <= d;
                end if;
        end process;
end architecture beh1;
```

Strukturni model

Za 4-bitni registar sa paralelnim upisom i čitanjem moguće je razviti i strukturni model. Ovaj model koristi komponentu D flip flopa koja se zatim instancionira 4 puta u okviru arhitekturnog tela, kao što je prikazano u modelu u nastavku.

```
architecture struct of reg4 is
   component d_ff is
      port (clk: in std_logic;    -- ulazni port dozvole upisa
            d:   in std_logic;    -- ulazni port podataka
            q:   out std_logic);-- izlazni port podataka 1
   end component d_ff;
begin
   bit0: d_ff
            port map (
               clk => clk,
               d => d(0),
               q => q(0));

   bit1: d_ff
            port map (
               clk => clk,
               d => d(1),
               q => q(1));

   bit2: d_ff
            port map (
               clk => clk,
               d => d(2),
               q => q(2));

   bit3: d_ff
            port map (
               clk => clk,
               d => d(3),
               q => q(3));
end architecture struct;
```

NAPOMENA: Prikazani bihevijalni i strukturni modeli 4-bitnog registra ilustruju moć bihevijalnog stila modelovanja. Ne samo što je bihevijalni model znatno kompaktniji od odgovarajućeg strukturnog modela, već je iz njega jasno vidljiva

funkcionalnost modula koji se modeluje što nije slučaj sa struturnim modelom. Pored toga, bihevijalni model je univerzalan u smislu da predstavlja model registra sa paralelnim upisom i čitanjem za proizvoljnu širinu ulaznog i izlaznog porta za podatke. Ovo nije slučaju sa strukturnim modelom. Na primer, u slučaju da nam je neophodan model 8-bitnog registra sa paralelnim upisom i čitanjem ako bismo koristili bihevijalni model jedin neophodna izmena bi bila u deklaraciji portova *d* i *q*, koji bi se morali deklarisati kao portovi tipa *std_logic_vector(7 downto 0)*. U slučaju strukturnog modela pored ovoga bilo bi neophodno i dodati još četiri naredbe instancioniranja *d_ff* komponente sa odgovarajućim povezivanjem portova. VHDL obezbeđuje način da se napišu i generički modeli iterativnih struktura, kao što je slučaj sa registrima, korišćenjem *generate* naredbi. Jedna od narednih vežbi posvećena je pisanju parametrizovanih, generičkih VHDL modela.

Verifikaciono okruženje

Primer jednog mogućeg testbenča koji se može iskoristiti za verifikaciju 4-bitnog registra sa paralelnim upisom i čitanjem prikazan je u nastavku.

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

entity reg4_tb is
end entity reg4_tb;


architecture beh of reg4_tb is
   -- Deklaracija komponente VHDL modula koji se verifikuje (DUV)
   -- U ovom slučaju je to 4-bitni registar
   component reg4 is
      port (clk: in std_logic;
            d:   in std_logic_vector(3 downto 0);
            q:   out std_logic_vector(3 downto 0));
   end component reg4;

   -- Deklaracija unutrašnjih signala potrebnih za povezivanje stimulus
   -- generatora sa ulazima DUV-a
   signal clk_s: std_logic;
   signal d_s: std_logic_vector(3 downto 0);
   signal q_s: std_logic_vector(3 downto 0);

begin
        -- Komponenta koja se verifikuje
        duv: reg4
           port map (
              clk => clk_s,
              d => d_s,
              q => q_s);

        -- Klok generator koji generise periodicni clk_s signal koji
        -- ce se koristiti za aktiviranje registra
        clk_gen: process
        begin
```

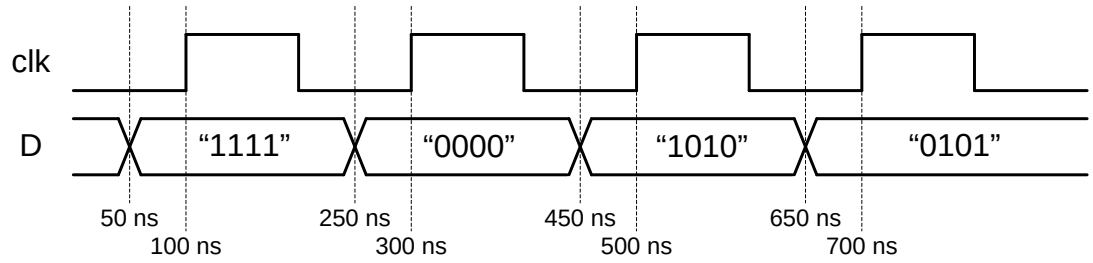
```
            clk_s <= '0', '1' after 100 ns;
            wait for 200 ns;
        end process;

        -- 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
           d_s <= "0000", "0000" after 80 ns, "0000" after 160 ns,
                          "0000" after 335 ns, "0000" after 345 ns;

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

Prikazani testbenč instancionira komponentu 4-bitnog registra i koristeći *clk_gen* i *stim_gen* procese generiše povorku ulaznih signala koja može da se iskoristi za proveru ispravnog rada napisanog modela registra.

Clk_gen proces generiše periodičnu povorku impulsa na signalu *clk_s* koji se dovodi na *clk* ulaz registra.

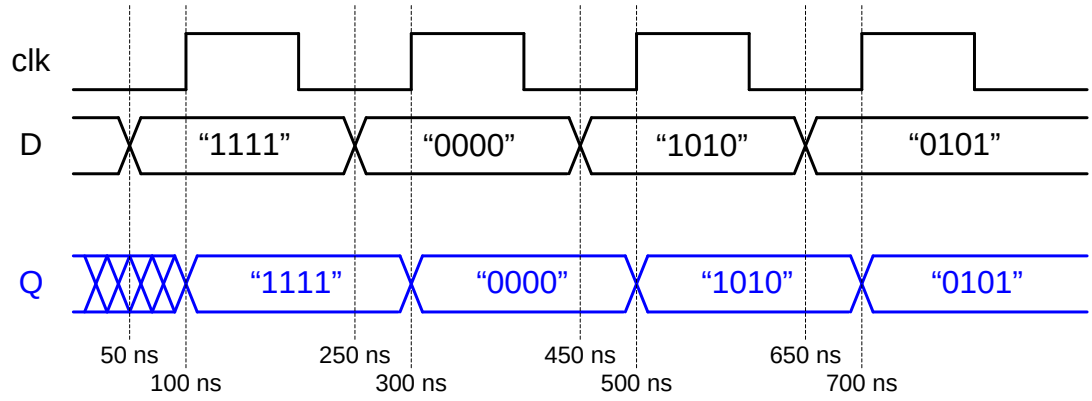
Stim_gen proces generiše vremenske oblike za signal *d_s* koristeći jednu naredbu sekvencijalne dodele vrednosti signalu. Obzirom da je reč o vektoru koji se sastoji od 4 bita, generisane vrednosti su vektor *std_logic* vrednosti.

Talasni oblici koji će biti generisani pomoću ove dve process naredbe prikazani su na slici 5.



Slika 5. Generisani talasni oblici na dva ulazna signala 4-bitnog registra

Nakon što se izvrši simulacija kombinovanog rada testbenča i modela 4-bitnog registra sa paralelnim upisom i čitanjem talasni oblik izlaznog signala podataka *q_s* trebalo bi da izgleda kao na slici 6.



Slika 6. Generisani talasni oblici na dva ulazna 4-bitnog registra sa paralelnim upisom i čitanjem zajedno sa talasnim oblicima izlaznong signala podataka *q_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 4-bitnog registra sa paralelnim upisom i čitanjem.

Zadaci za vežbu

Zadatak 1:

Napisati VHDL model 4-bitnog registra sa paralelnim upisom i čitanjem koji će modelovati konačnu brzinu prostiranja signala kroz registar. Vremenske karakteristike registra koji je potrebno modelovati imaju sledeće vrednosti:

- vreme uspostavljanja na ulazu *D*, t_{su} = 0.5 ns
- vreme držanja na ulazu *D*, t_h = 0.5 ns
- kašnjenje na izlazu *Q*, t_q = 1.3 ns

Za tako napisani model registra razviti i potrebno verifikaciono okruženje koje će moći da se iskoristi za njegovu funkcionalnu verifikaciju.

Zadatak 2:

Proširiti VHDL model 4-bitnog registra sa paralelnim upisom i čitanjem iz zadatka 1 tako da uključi:

- vremenske provere zadovoljenosti vremena uspostavljanja i držanja na *D* ulazu i signalizaciju ukoliko neki od ovih uslova nije ispunjen; za implementaciju ovih provera koristiti *assert* VHDL naredbu
- modelovanje pojave nepoznatog stanja na izlazu *Q*, u slučaju da na *D* ulazu nisu zadovoljena vremena uspostavljanja ili držanja; za nepoznatu vrednost koristiti ‘X’ vrednost *std_logic* tipa

Za razvijeni VHDL model napisati odgovarajući testbenč pomoću kojega ća biti moguće izvršiti funkcionalnu verifikaciju modela.

Zadatak 3:
Napisati VHDL model 4-bitnog registra sa paralelnim upisom i čitanjem koji će pored ulanzog porta za podatke imati i sledeće ulazne portove:

- *clock enable* (*ce*) signal za selekciju rastućih ivica *clk* ulaza na koje registar treba da reaguje
- ulazni portove za inicijalizaciju sadržaja registra – *reset*

Za razvijeni VHDL model napisai odgovarajući testbenč pomoću kojega će biti moguće izvršiti funkcionalnu verifikaciju modela.

Zadatak 4:
Napisati VHDL model 4-bitnog registra sa paralelnim upisom i čitanjem koji će pored ulanzog porta za podatke imati i sledeće ulazne portove:

- *clock enable* (*ce*) signal za selekciju rastućih ivica *clk* ulaza na koje registar treba da reaguje
- ulazni port dozvole upisa novog sadržaja u registar - *we* (*write enable*)
- ulazni portove za inicijalizaciju sadržaja registra – *set*

Za razvijeni VHDL model napisai odgovarajući testbenč pomoću kojega će biti moguće izvršiti funkcionalnu verifikaciju modela.

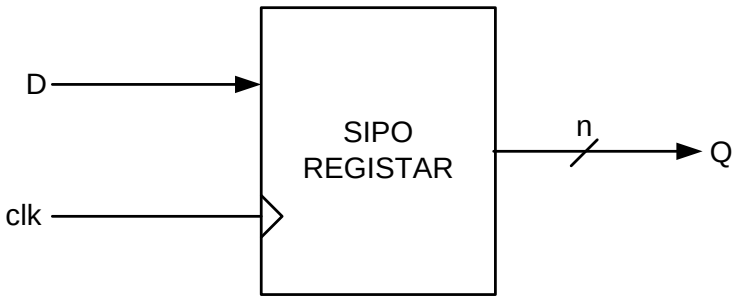
Zadatak 5:
Napisati VHDL model 4-bitnog registra sa paralelnim upisom i čitanjem koji će pored ulanzog porta za podatke imati i sledeće ulazne portove:

- *clock enable* (*ce*) signal za selekciju rastućih ivica *clk* ulaza na koje registar treba da reaguje
- ulazni port dozvole upisa novog sadržaja u registar - *we* (*write enable*)
- ulazni port dozvole čitanja trenutnog sadržaja iz registra - *read* ili *re* (*read enable*). Kada je *re* port jednak 0 na Q izlazu registra treba da stoji vrednost “0000”, a ako je *re* port jednak 1 na Q izlazu registra treba da stoji tekuća vrednost koja se nalazi smeštena u registru.
- ulazni portove za inicijalizaciju sadržaja registra – *clear* i *preset*

Za razvijeni VHDL model napisai odgovarajući testbenč pomoću kojega će biti moguće izvršiti funkcionalnu verifikaciju modela.

Registri sa serijskim upisom i paralelnim čitanjem

Registri sa serijskim upisom i paralelnim čitanjem (SIPO) poseduju 1-bitni ulazni port podataka i *n*-bitni izlazni port podataka. Podaci se u registar upisuju serijski, bit po bit, a trenutni sadržaj registra dostupan je u svakom trenutku na paralelnom izlaznom portu. Ovi registri se obično koriste za proces konverzije iz serijskog tipa u paralelni tip reprezentacije podataka unutar sistema. Obično se mogu pronaći u sistemima za prijem podataka koji se prenose preko serijskog linka. Osnovni interfejs *n*-bitnog registra sa serijskim upisom i paralelnim čitanjem prikazan je na slici 7.



Slika 7. Interfejs *n*-bitnog registra sa serijskim upisom i paralelnim čitanjem

N bitni registar sa serijskim upisom i paralelnim čitanjem ima dva ulazna porta:

- *D* ulaz za podatke, širine 1 bita
- *clk* ulaz za sinhronizaciju rada registra

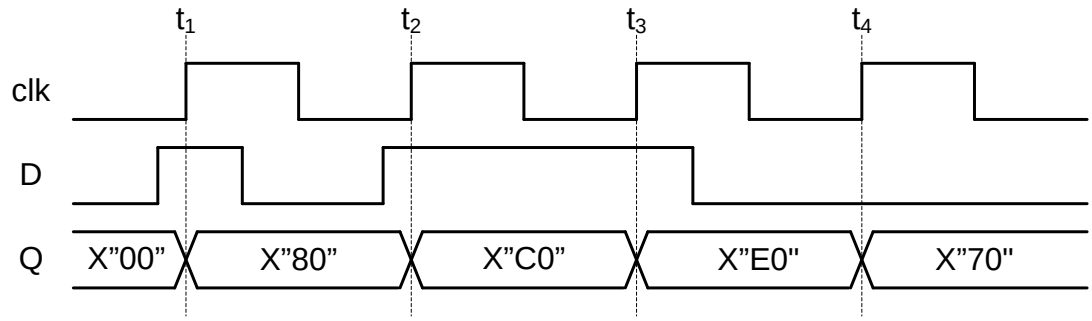
N bitni registar sa serijskim upisom i paralelnim čitanjem ima jedan izlazni port:

- *Q* izlaz za podatke, širine *n* bita

Funkcionalnost registra sa serijskim upisom i paralelnim čitanjem može se prikazati sledećom tabelom.

clk	*D*	*Q*(t+1)
0	x	*Q*(t)
1	x	*Q*(t)
↓	x	*Q*(t)
↑	*data*	*Q*(t)(n-1:0)&*data*, ili *data*&*Q*(t)(n:1)

Kao što se može primetiti na osnovu prethodne tabele, sve dok se na *clk* ulazu ne pojavi rastuća ivica registar čuva prethodno upisanu vrednost i ignoriše vrednosti koje mu se trenutno nalaze na *D* ulazu. Tek kada se na *clk* ulazu pojavi "sinhronizacioni događaj" (rastuća ili opadajuća ivica), u registar se upisuje trenutna vrednost koja se nalazi na *D* ulazu. Bitno je naglasiti da se ovaj upis obavlja na jedan od dva načina, u zavisnosti da li je ulazni port *D* povezan na bit najmanje ili najveće težine unutar pomeračkog registra. U prvom slučaju novi sadržaj registra dobija se tako što se trenutni sadržaj registra pomeri za jedno mesto u levo, a na upražnjeno mesto se upiše trenutna vrednost *D* porta. U drugom slučaju, trenutni sadržaj registra pomeri se za jedno mesto u desno, a na upražnjeno mesto se upiše trenutna vrednost *D* porta. Novoupisana vrednost se istovremeno (u paralelnom formatu) pojavljuje i na izlaznom portu *Q*. Na primer, vremenski dijagram rada 8-bitnog registra sa serijskim upisom i paralelnim čitanjem, pri čemu se serijski podaci upisuju na bit najveće težine prikazan je na slici 8.



Slika 8. Vremenski dijagram rada 8-bitnog registra sa serijskim upisom i paralelnim čitanjem, pri čemu se serijski podaci upisuju na bit najveće težine

VHDL modeli n-bitnog registra sa serijskim upisom i paralelnim čitanjem

Da bi smo ilustrovali različite načine modelovanja n-bitnog registra sa serijskim upisom i paralelnim čitanjem, koristićemo 4-bitni registar kod kojega se serijski podaci upisuju u bit najveće težine. *Entity* deklaracija 4-bitnog registra sa serijskim upisom i paralelnim čitanjem ima sledeći izgled

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

entity sipo_reg4 is
   port (clk: in std_logic;
         d:    in std_logic;  -- ulazni port podataka
         q:    out std_logic_vector(3 downto 0) -- izlazni port podataka
        );
end entity sipo_reg4;
```

Nakon što smo opisali interfejs 4-bitnog registra sa serijskim upisom i paralelnim čitanjem, potrebno je modelovati njegovu funkcionalnost. Za ovo je potrebno napisati odgovarajuće arhitekturno telo.

Bihevijani model

Varijanta 1: Model 4-bitnog registra sa serijskim upisom i paralelnim čitanjem baziran na korišćenju *process* i *if* naredbe i unutrašnjeg signala

```
architecture beh1 of sipo_reg4 is
        signal q_s: std_logic_vector(3 downto 0);
begin
        reg: process (clk) is
        begin
                if (clk’event and clk = '1') then
                        q_s <= d&q_s(3 downto 1);
                end if;
        end process;

        q <= q_s;
```

```
end architecture beh1;
```

NAPOMENA: Prilikom modelovanja 4-bitnog registra sa serijskim upisom i paralelnim čitanjem morali smo iskoristiti unutrašnji signal, *q_s*, jer se unutar procesa *reg*, prilikom upisivanja nove vrednosti u registar koristi tekuća vrednost. Zbog ovoga ne bi bilo moguće koristiti port *q*, jer je on deklarisan kao izlazni. Izlaznim portovima moguće je samo dodeljivati vrednosti a ne i čitati. Isto tako bi bilo pogrešno ukoliko bi se port *q* deklarisao kao ulazno/izlazni port, jer bi to narušilo specifikaciju 4-bitnog registra koja ne sadrži niti jedan ulazno/izlazni port.

Varijanta 2: Model 4-bitnog registra sa serijskim upisom i paralelnim čitanjem baziran na korišćenju *process* i *if* naredbe i unutrašnje promenljive

```
architecture beh2 of sipo_reg4 is
begin
        reg: process (clk) is
                variable q_v: std_logic_vector(3 downto 0);
        begin
                if (clk'event and clk = '1') then
                        q_v := d&q_s(3 downto 1);
                        q <= q_v;
                end if;
        end process;
end architecture beh2;
```

Kombinovanjem entity deklaracije i odgovarajućeg arhitekturnog tela dobijamo kompletan model 4-bitnog registra sa serijskim upisom i paralelnim čitanjem.

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

entity sipo_reg4 is
   port (clk: in std_logic;
         d:    in std_logic;  -- ulazni port podataka
         q:    out std_logic_vector(3 downto 0) -- izlazni port podataka
        );
end entity sipo_reg4;


architecture beh2 of sipo_reg4 is
begin
        reg: process (clk) is
                variable q_v: std_logic_vector(3 downto 0);
        begin
                if (clk'event and clk = '1') then
                        q_v := d&q_s(3 downto 1);
                        q <= q_v;
                end if;
        end process;
end architecture beh2;
```

Strukturni model

Za 4-bitni registar sa serijskim upisom i paralelnim čitanjem moguće je razviti i strukturni model. Ovaj model koristi komponentu D flip flopa koja se zatim instancionira 4 puta u okviru arhitekturnog tela, kao što je prikazano u modelu u nastavku.

```
architecture struct of sipo_reg4 is
   component d_ff is
      port (clk: in std_logic;    -- ulazni port dozvole upisa
            d:   in std_logic;    -- ulazni port podataka
            q:   out std_logic);-- izlazni port podataka
   end component d_ff;

   signal q_s: std_logic_vector(3 downto 0);
begin
   bit3: d_ff
            port map (
               clk => clk,
               d => d,
               q => q_s(3));

   bit2: d_ff
            port map (
               clk => clk,
               d => q_s (3),
               q => q_s (2));

   bit1: d_ff
            port map (
               clk => clk,
               d => q_s(2),
               q => q_s(1));

   bit0: d_ff
            port map (
               clk => clk,
               d => q_s(1),
               q => q_s(0));

   q <= q_s;
end architecture struct;
```

Verifikaciono okruženje

Primer jednog mogućeg testbenča koji se može iskoristiti za verifikaciju 4-bitnog registra sa serijskim upisom i paralelnim čitanjem prikazan je u nastavku.

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

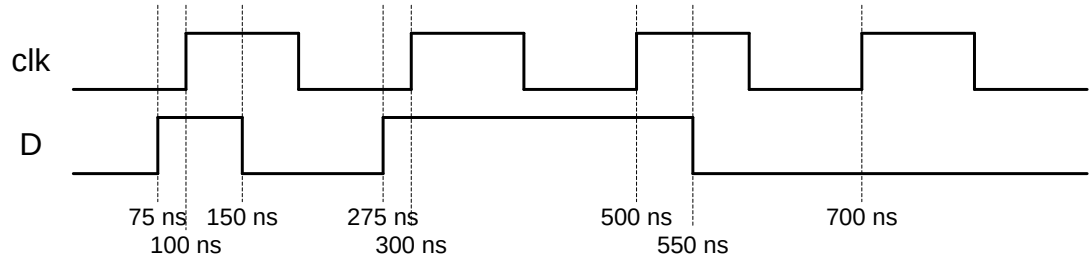
```
entity sipo_reg4_tb is
end entity sipo_ reg4_tb;

architecture beh of sipo_reg4_tb is
   -- Deklaracija komponente VHDL modula koji se verifikuje (DUV)
   -- U ovom slučaju je to 4-bitni registar sa serijskim ulazom i
   -- paralelnim izlazom
   component sipo_reg4 is
      port (clk: in std_logic;
            d:    in std_logic;
            q:    out std_logic_vector(3 downto 0));
   end component sipo_reg4;

   -- Deklaracija unutrašnjih signala potrebnih za povezivanje stimulus
   -- generatora sa ulazima DUV-a
   signal clk_s: std_logic;
   signal d_s: std_logic;
   signal q_s: std_logic_vector(3 downto 0);

begin
        -- Komponenta koja se verifikuje
        duv: sipo_reg4
           port map (
              clk => clk_s,
              d => d_s,
              q => q_s);

        -- Klok generator koji generise periodicni clk_s signal koji
        -- ce se koristiti za aktiviranje registra
        clk_gen: process
        begin
                clk_s <= '0', '1' after 100 ns;
                wait for 200 ns;
        end process;

        -- 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
                d_s <= '0', '1' after 75 ns, '0' after 150 ns,
                              '1' after 275 ns, '0' after 550 ns;

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

Prikazani testbenč instancionira komponentu 4-bitnog registra i koristeći *clk_gen* i *stim_gen* procese generiše povorku ulaznih signala koja može da se iskoristi za proveru ispravnog rada napisanog modela registra.

Clk_gen proces generiše periodičnu povorku impulsa na signalu *clk_s* koji se dovodi na *clk* ulaz registra.

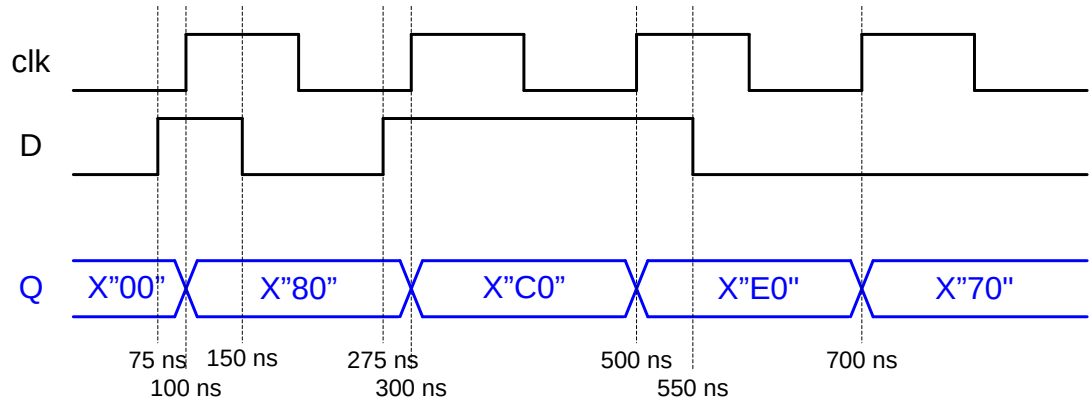
Stim_gen proces generiše vremenske oblike za signal *d_s* koristeći jednu naredbu sekvencijalne dodele vrednosti signalu.

Talasni oblici koji će biti generisani pomoću ove dve process naredbe prikazani su na slici 9.



Slika 9. Generisani talasni oblici na dva ulazna signala 4-bitnog registra

Nakon što se izvrši simulacija kombinovanog rada testbenča i modela 4-bitnog registra sa serijskim upisom i paralelnim čitanjem, talasni oblik izlaznog signala podataka *q_s* trebalo bi da izgleda kao na slici 10.



Slika 10. Generisani talasni oblici na dva ulazna 4-bitnog registra sa serijskim upisaom i paralelnim čitanjem zajedno sa talasnim oblicima izlaznong signala podataka *q_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 4-bitnog registra sa serijskim upisom i paralelnim čitanjem.

Zadaci za vežbu

Zadatak 1:

Napisati VHDL model 4-bitnog registra sa serijskim upisom i paralelnim čitanjem koji će pored ulanzog porta za podatke imati i sledeće ulazne portove:

- *clock enable* (*ce*) signal za selekciju rastućih ivica *clk* ulaza na koje registar treba da reaguje
- ulazni portove za inicijalizaciju sadržaja registra – *reset*

Za razvijeni VHDL model napisai odgovarajući testbenč pomoću kojega će biti moguće izvršiti funkcionalnu verifikaciju modela.

Zadatak 2:

Napisati VHDL model 4-bitnog registra sa serijskim upisom i paralelnim čitanjem koji će pored ulanzog porta za podatke imati i sledeće ulazne portove:

- *clock enable* (*ce*) signal za selekciju rastućih ivica *clk* ulaza na koje registar treba da reaguje
- ulazni port dozvole upisa novog sadržaja u registar - *we* (*write enable*)
- ulazni portove za inicijalizaciju sadržaja registra – *set*

Za razvijeni VHDL model napisai odgovarajući testbenč pomoću kojega će biti moguće izvršiti funkcionalnu verifikaciju modela.

Zadatak 3:

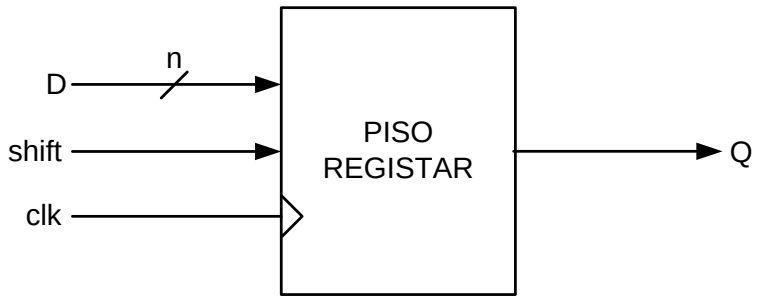
Napisati VHDL model 4-bitnog registra sa serijskim upisom i paralelnim čitanjem koji će pored ulanzog porta za podatke imati i sledeće ulazne portove:

- *clock enable* (*ce*) signal za selekciju rastućih ivica *clk* ulaza na koje registar treba da reaguje
- ulazni port dozvole upisa novog sadržaja u registar - *we* (*write enable*)
- ulazni port dozvole čitanja trenutnog sadržaja iz registra - *read* ili *re* (*read enable*). Kada je *re* port jednak 0 na *Q* izlazu registra treba da stoji vrednost "0000", a ako je *re* port jednak 1 na *Q* izlazu registra treba da stoji tekuća vrednost koja se nalazi smeštena u registru.
- ulazni portove za inicijalizaciju sadržaja registra – *clear* i *preset*

Za razvijeni VHDL model napisai odgovarajući testbenč pomoću kojega će biti moguće izvršiti funkcionalnu verifikaciju modela.

Registri sa paralelnim upisom i serijskim čitanjem

Registri sa paralelnim upisom i serijskim čitanjem (PISO) poseduju 1-bitni ulazni port podataka i *n*-bitni izlazni port podataka. Podaci se u registar upisuju paralelno, a sadržaj registra čita se serijski, preko 1-bitnog izlaznog porta. Ovi registri se obično koriste za proces konverzije iz paralelnog tipa u serijski tip reprezentacije podataka unutar sistema. Obično se mogu pronaći u sistemima za predaju podataka koji se prenose preko serijskog linka. Osnovni interfejs *n*-bitnog registra sa paralelnim upisom i serijskim čitanjem prikazan je na slici 11.



Slika 11. Interfejs *n*-bitnog registra sa paralelnim upisom i serijskim čitanjem

N bitni registar sa serijskim upisom i paralelnim čitanjem ima dva ulazna porta:

- *D* ulaz za podatke, širine *n* bita
- *shift* kontrolnog ulaza, pomoću kojega se vrši čitanje upisanog sadržaja
- *clk* ulaz za sinhronizaciju rada registra

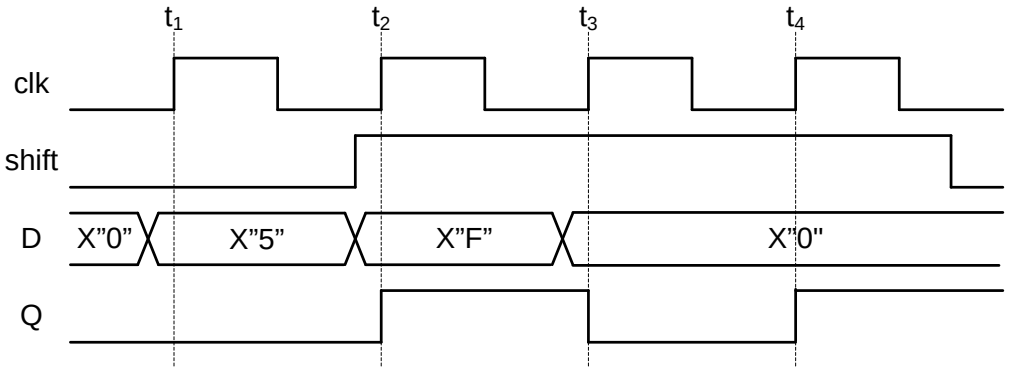
N bitni registar sa paralelnim upisom i serijskim čitanjem ima jedan izlazni port:

- *Q* izlaz za podatke, širine 1 bita

Funkcionalnost D leča sa dozvolom upisa može se prikazati sledećom tabelom.

clk	*shift*	*D*	Q	*s*(t+1)
0	x	x	*s*(t)(0)	*s*(t)
1	x	x	*s*(t)(0)	*s*(t)
↓	x	x	*s*(t)(0)	*s*(t)
↑	0	*data*	*s*(t)(0)	*data*
↑	1	x	*s*(t)(0)	‘0’&*s*(t)(n-1:1)

Kao što se može primetiti na osnovu prethodne tabele, sve dok se na *clk* ulazu ne pojavi rastuća ivica registar čuva prethodno upisanu vrednost i ignoriše vrednosti koje mu se trenutno nalaze na *D* ulazu. Tek kada se na *clk* ulazu pojavi "sinhronizacioni događaj" (rastuća ili opadajuća ivica), registar će promeniti svoje stanje u zavisnosti od trenutne vrednosti na ulaznom portu *shift*. Ukoliko je vrednost *shift* porta jednaka 0 u registar (predstavljen signalom *s*(t) u tabeli gore) se upisuje trenutna vrednost koja se nalazi na *D* ulazu. Ukoliko je vrednost *shift* ulaza jednaka 1, vrši se pomeranje tekućeg sadržaja registra za jedno mesto u desno, pri čemu se vrednost bita najmanje težine pojavljuje na izlaznom portu *Q*. Na primer, vremenski dijagram rada 4-bitnog registra sa paralelnim upisom upisom i serijskim čitanjem prikazan je na slici 12.



Slika 12. Vremenski dijagram rada 4-bitnog registra sa paralelnim upisom i serijskim čitanjem

VHDL modeli n-bitnog registra sa paralelnim upisom i serijskim čitanjem

Da bi smo ilustrovali različite načine modelovanja n-bitnog registra sa paralelnim upisom i serijskim čitanjem, koristićemo 4-bitni registar. *Entity* deklaracija 4-bitnog registra sa paralelnim upisom i serijskim čitanjem ima sledeći izgled

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

entity piso_reg4 is
   port (clk: in std_logic;
         d:    in std_logic_vector(3 downto 0);  -- ulazni port podataka
         q:    out std_logic -- izlazni port podataka
        );
end entity piso_reg4;
```

Nakon što smo opisali interfejs 4-bitnog registra sa paralelnim upisom i serijskim čitanjem, potrebno je modelovati njegovu funkcionalnost. Za ovo je potrebno napisati odgovarajuće arhitekturno telo.

Bihevijani model

Varijanta 1: Model 4-bitnog registra sa paralelnim upisom i serijskim čitanjem baziran na korišćenju *process* i *if* naredbe i unutrašnjeg signala

```
architecture beh1 of piso_reg4 is
        signal q_s: std_logic_vector(3 downto 0);
begin
        reg: process (clk) is
        begin
                if (clk'event and clk = '1') then
                        if (shift = '0') then
                                q_s <= d;
                        else
                                q <= q_s(0);
                                q_s <= '0'&q_s(3 downto 1);
```

```
                end if;
            end if;
        end process;

end architecture beh1;
```

Varijanta 2: Model 4-bitnog registra sa paralelnim upisom i serijskim čitanjem baziran na korišćenju *process* i *if* naredbe i unutrašnje promenljive

```
architecture beh2 of piso_reg4 is
begin
        reg: process (clk) is
                variable q_v: std_logic_vector(3 downto 0);
        begin
                if (clk'event and clk = '1') then
                        if (shift = '0') then
                                q_v := d;
                        else
                                q <= q_v(0);
                                q_v := '0'&q_v(3 downto 1);
                        end if;
                end if;
        end process;
end architecture beh2;
```

Kombinovanjem entity deklaracije i odgovarajućeg arhitekturnog tela dobijamo kompletan model 4-bitnog registra sa serijskim upisom i paralelnim čitanjem.

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

entity piso_reg4 is
   port (clk: in std_logic;
         d:    in std_logic_vector(3 downto 0);  -- ulazni port podataka
         q:    out std_logic -- izlazni port podataka
        );
end entity piso_reg4;


architecture beh1 of piso_reg4 is
        signal q_s: std_logic_vector(3 downto 0);
begin
        reg: process (clk) is
        begin
                if (clk'event and clk = '1') then
                        if (shift = '0') then
                                q_s <= d;
                        else
                                q <= q_s(0);
                                q_s <= '0'&q_s(3 downto 1);
                        end if;
```

```
            end if;
        end process;

end architecture beh1;
```

Verifikaciono okruženje

Primer jednog mogućeg testbenča koji se može iskoristiti za verifikaciju 4-bitnog registra sa paralelnim upisom i serijskim čitanjem prikazan je u nastavku.

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

entity piso_reg4_tb is
end entity piso_ reg4_tb;


architecture beh of piso_reg4_tb is
   -- Deklaracija komponente VHDL modula koji se verifikuje (DUV)
   -- U ovom slučaju je to 4-bitni registar sa paralelnim ulazom i
   -- serijskim izlazom
   component piso_reg4 is
      port (clk: in std_logic;
            d:    in std_logic_vector(3 downto 0);
            q:    out std_logic
           );
   end component piso_reg4;

   -- Deklaracija unutrašnjih signala potrebnih za povezivanje stimulus
   -- generatora sa ulazima DUV-a
   signal clk_s: std_logic;
   signal shift_s: std_logic;
   signal d_s: std_logic_vector(3 downto 0);
   signal q_s: std_logic;

begin
        -- Komponenta koja se verifikuje
        duv: piso_reg4
           port map (
              clk => clk_s,
              d => d_s,
              q => q_s);

        -- Klok generator koji generise periodicni clk_s signal koji
        -- ce se koristiti za aktiviranje registra
        clk_gen: process
        begin
                clk_s <= '0', '1' after 100 ns;
                wait for 200 ns;
        end process;
```

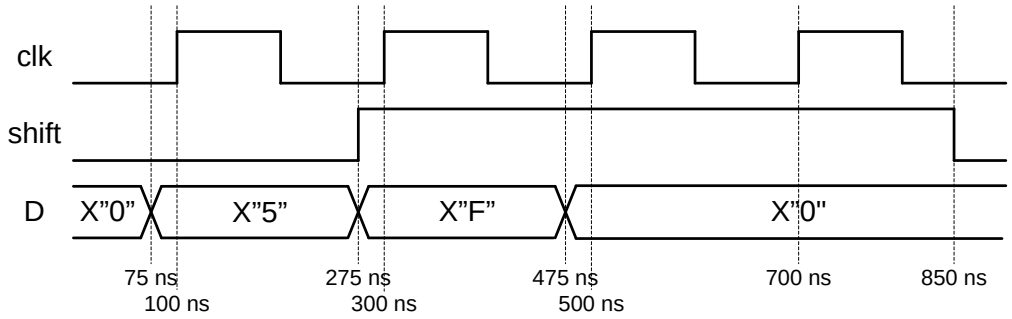
```
        -- Stimulus generator koji generise potrebne vrednosti na
        -- ulaznim portovima DUV-a na osnovu kojih ce biti moguce
        -- proveriti da li DUV implementira potrebnu funkcionalnost
        stim_gen: process
        begin
                shift_s <= '0', '1' after 275 ns, '0' after 850 ns;

                d_s <= X"0", X"5" after 75 ns, X"F" after 275 ns,
                                X"0" after 475 ns;


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

Prikazani testbenč instancionira komponentu 4-bitnog registra i koristeći *clk_gen* i *stim_gen* procese generiše povorku ulaznih signala koja može da se iskoristi za proveru ispravnog rada napisanog modela registra.

Clk_gen proces generiše periodičnu povorku impulsa na signalu *clk_s* koji se dovodi na *clk* ulaz registra.

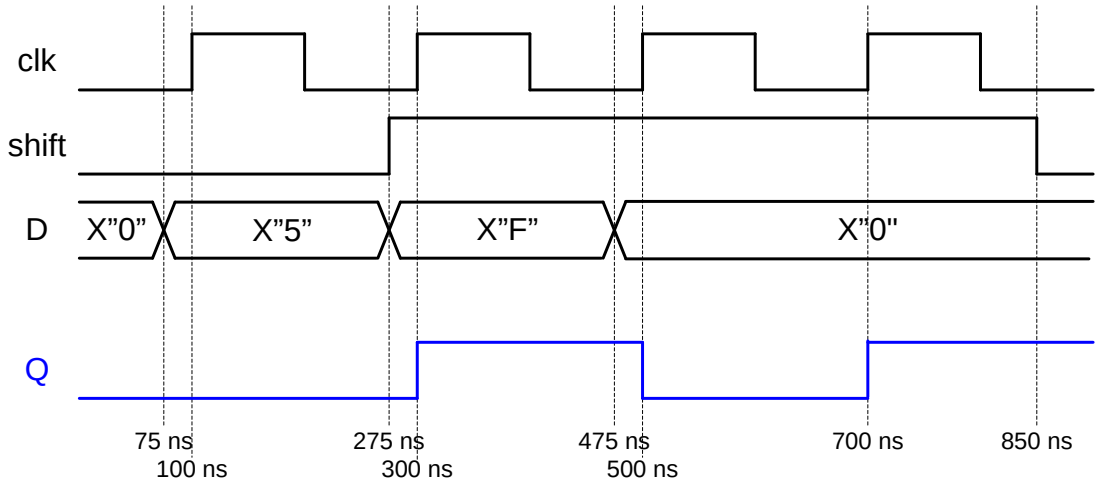
Stim_gen proces generiše vremenske oblike za signale *shift_s* i *d_s* koristeći dve naredbe sekvencijalne dodele vrednosti signalu.

Talasni oblici koji će biti generisani pomoću ove dve *process* naredbe prikazani su na slici 13.



Slika 13. Generisani talasni oblici na tri ulazna signala PISO registra

Nakon što se izvrši simulacija kombinovanog rada testbenča i modela 4-bitnog registra sa paralelnim upisom i serijskim čitanjem, talasni oblik izlaznog signala podataka *q_s* trebalo bi da izgleda kao na slici 14.



Slika 14. Generisani talasni oblici na tri ulazna signala 4-bitnog registra sa paralelnim upisaom i serijskim čitanjem zajedno sa talasnim oblicima izlaznog signala podataka *q_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 4-bitnog registra sa paralelnim upisom i serijskim čitanjem.

Zadaci za vežbu

Zadatak 1:

Napisati VHDL model 4-bitnog registra sa paralelnim upisom i serijskim čitanjem koji će pored ulanzog porta za podatke imati i sledeće ulazne portove:

- *clock enable* (*ce*) signal za selekciju rastućih ivica *clk* ulaza na koje registar treba da reaguje
- ulazni portove za inicijalizaciju sadržaja registra – *reset*

Za razvijeni VHDL model napisai odgovarajući testbenč pomoću kojega će biti moguće izvršiti funkcionalnu verifikaciju modela.

Zadatak 2:

Napisati VHDL model 4-bitnog registra sa paralelnim upisom i serijskim čitanjem koji će pored ulanzog porta za podatke imati i sledeće ulazne portove:

- *clock enable* (*ce*) signal za selekciju rastućih ivica *clk* ulaza na koje registar treba da reaguje
- ulazni port dozvole upisa novog sadržaja u registar - *we* (*write enable*)
- ulazni portove za inicijalizaciju sadržaja registra – *set*

Za razvijeni VHDL model napisai odgovarajući testbenč pomoću kojega će biti moguće izvršiti funkcionalnu verifikaciju modela.

Zadatak 3:

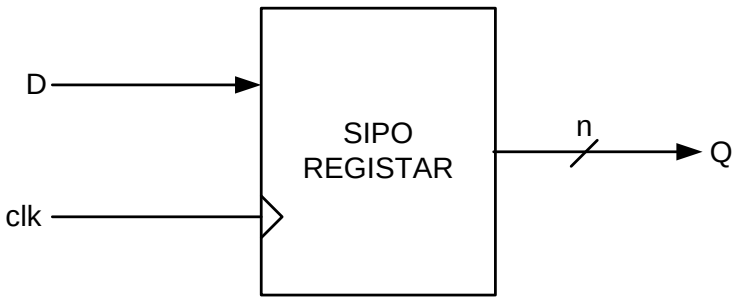
Napisati VHDL model 4-bitnog registra sa paralelnim upisom i serijskim čitanjem koji će pored ulanzog porta za podatke imati i sledeće ulazne portove:

- *clock enable* (*ce*) signal za selekciju rastućih ivica *clk* ulaza na koje registar treba da reaguje
- ulazni port dozvole upisa novog sadržaja u registar - *we* (*write enable*)
- ulazni portove za inicijalizaciju sadržaja registra – *clear* i *preset*

Za razvijeni VHDL model napisai odgovarajući testbenč pomoću kojega će biti moguće izvršiti funkcionalnu verifikaciju modela.

Registri sa serijskim upisom i čitanjem

Registri sa serijskim upisom i čitanjem (SISO) poseduju 1-bitni ulazni port podataka i 1-bitni izlazni port podataka. Podaci se u registar upisuju serijski, bit po bit, i čitaju se serisjki, preko izlaznog porta. Obzirom da je registar veličine *n* bita, upisani podatak pojaviće se na izlazu registra nakon *n* taktova. Ovi registri se obično koriste kao linije za kašnjenjne unutar složenijih sistema. Osnovni interfejs *n*-bitnog registra sa serijskim upisom i čitanjem prikazan je na slici 15.



Slika 15. Interfejs *n*-bitnog registra sa serijskim upisom i čitanjem

N-bitni registar sa serijskim upisom i čitanjem ima dva ulazna porta:

- *D* ulaz za podatke, širine 1 bita
- *clk* ulaz za sinhronizaciju rada registra

N-bitni registar sa serijskim upisom i čitanjem ima jedan izlazni port:

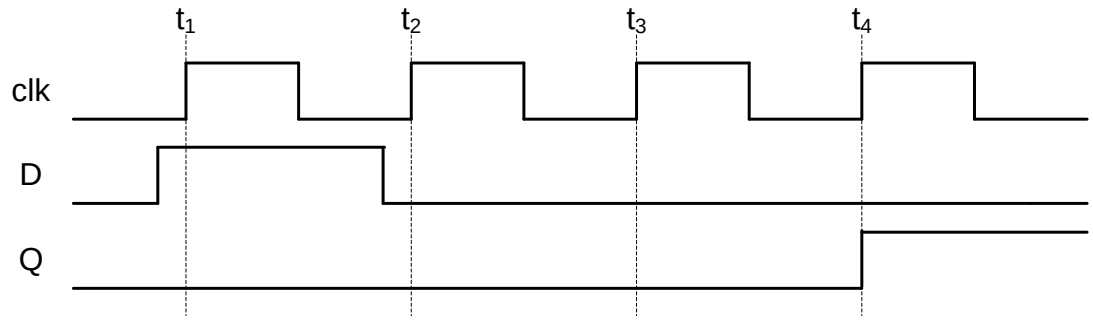
- *Q* izlaz za podatke, širine 1 bita

Funkcionalnost registra sa serijskim upisom i čitanjem može se prikazati sledećom tabelom.

clk	*D*	*Q*	*s*(t+1)
0	x	*s*(t)(0) ili *s*(t)(*n*-1)	*s*(t)
1	x	*s*(t)(0) ili *s*(t)(*n*-1)	*s*(t)
↓	x	*s*(t)(0) ili *s*(t)(*n*-1)	*s*(t)
↑	*data*	*s*(t)(0) ili *s*(t)(*n*-1)	*data*&*s*(t)(*n*-1:1) ili *s*(t)(*n*-2:0)&*data*

Kao što se može primetiti na osnovu prethodne tabele, sve dok se na *clk* ulazu ne pojavi rastuća ivica registar čuva prethodno upisanu vrednost i ignoriše vrednosti koje mu se trenutno nalaze na *D* ulazu. Tek kada se na *clk* ulazu pojavi "sinhronizacioni

događaj" (rastuća ili opadajuća ivica), u registar se upisuje trenutna vrednost koja se nalazi na *D* ulazu. Bitno je naglasiti da se ovaj upis obavlja na jedan od dva načina, u zavisnosti da li je ulazni port *D* povezan na bit najmanje ili najveće težine unutar pomeračkog registra. U prvom slučaju novi sadržaj registra dobija se tako što se trenutni sadržaj registra pomeri za jedno mesto u levo, a na upražnjeno mesto se upiše trenutna vrednost *D* porta. U drugom slučaju, trenutni sadržaj registra pomeri se za jedno mesto u desno, a na upražnjeno mesto se upiše trenutna vrednost *D* porta. Na izlaznom portu *Q* pojavljuje se vrednost bita najmanje ili najveće težine, u zavisnosti od toga gde je povezan ulazni port *D*. Na primer, vremenski dijagram rada 4-bitnog registra sa serijskim upisom i čitanjem, pri čemu se serijski podaci upisuju na bit najveće težine prikazan je na slici 16.



Slika 16. Vremenski dijagram rada 4-bitnog registra sa serijskim upisom i čitanjem, pri čemu se serijski podaci upisuju na bit najveće težine

VHDL modeli n-bitnog registra sa serijskim upisom i čitanjem

Da bi smo ilustrovali različite načine modelovanja n-bitnog registra sa serijskim upisom i čitanjem, koristićemo 4-bitni registar kod kojega se serijski podaci upisuju u bit najveće težine. *Entity* deklaracija 4-bitnog registra sa serijskim upisom i čitanjem ima sledeći izgled

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

entity siso_reg4 is
   port (clk: in std_logic;
         d:   in std_logic;  -- ulazni port podataka
         q:   out std_logic -- izlazni port podataka
        );
end entity siso_reg4;
```

Nakon što smo opisali interfejs 4-bitnog registra sa serijskim upisom i čitanjem, potrebno je modelovati njegovu funkcionalnost. Za ovo je potrebno napisati odgovarajuće arhitekturno telo.

Bihevijani model

Varijanta 1: Model 4-bitnog registra sa serijskim upisom i čitanjem baziran na korišćenju *process* i *if* naredbe i unutrašnjeg signala

```
architecture beh1 of siso_reg4 is
        signal q_s: std_logic_vector(3 downto 0);
begin
        reg: process (clk) is
        begin
                if (clk'event and clk = '1') then
                        q_s <= d&q_s(3 downto 1);
                end if;
        end process;

        q <= q_s(0);
end architecture beh1;
```

Varijanta 2: Model 4-bitnog registra sa serijskim upisom i čitanjem baziran na korišćenju *process* i *if* naredbe i unutrašnje promenljive

```
architecture beh2 of siso_reg4 is
begin
        reg: process (clk) is
                variable q_v: std_logic_vector(3 downto 0);
        begin
                if (clk'event and clk = '1') then
                        q_v := d&q_s(3 downto 1);
                        q <= q_v(0);
                end if;
        end process;
end architecture beh2;
```

Kombinovanjem entity deklaracije i odgovarajućeg arhitekturnog tela dobijamo kompletan model 4-bitnog registra sa serijskim upisom i čitanjem.

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

entity siso_reg4 is
   port (clk: in std_logic;
         d:    in std_logic;  -- ulazni port podataka
         q:    out std_logic -- izlazni port podataka
        );
end entity siso_reg4;

architecture beh2 of siso_reg4 is
begin
        reg: process (clk) is
                variable q_v: std_logic_vector(3 downto 0);
        begin
                if (clk'event and clk = '1') then
                        q_v := d&q_s(3 downto 1);
                        q <= q_v(0);
```

```
            end if;
        end process;
end architecture beh2;
```

Strukturni model

Za 4-bitni registar sa serijskim upisom i čitanjem moguće je razviti i strukturni model. Ovaj model koristi komponentu D flip flopa koja se zatim instancionira 4 puta u okviru arhitekturnog tela, kao što je prikazano u modelu u nastavku.

```
architecture struct of siso_reg4 is
   component d_ff is
      port (clk: in std_logic;    -- ulazni port dozvole upisa
            d:   in std_logic;    -- ulazni port podataka
            q:   out std_logic);-- izlazni port podataka
   end component d_ff;

   signal q_s: std_logic_vector(3 downto 0);
begin
   bit3: d_ff
         port map (
            clk => clk,
            d => d,
            q => q_s(3));

   bit2: d_ff
         port map (
            clk => clk,
            d => q_s (3),
            q => q_s (2));

   bit1: d_ff
         port map (
            clk => clk,
            d => q_s(2),
            q => q_s(1));

   bit0: d_ff
         port map (
            clk => clk,
            d => q_s(1),
            q => q_s(0));

   q <= q_s(0);
end architecture struct;
```

Verifikaciono okruženje

Primer jednog mogućeg testbenča koji se može iskoristiti za verifikaciju 4-bitnog registra sa serijskim upisom i čitanjem prikazan je u nastavku.

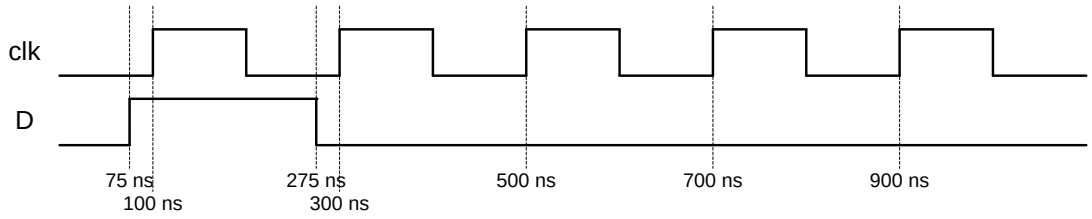
```
library ieee;
use ieee.std_logic_1164.all;

entity siso_reg4_tb is
end entity siso_ reg4_tb;


architecture beh of sipo_reg4_tb is
   -- Deklaracija komponente VHDL modula koji se verifikuje (DUV)
   -- U ovom slučaju je to 4-bitni registar sa serijskim ulazom i
   -- paralelnim izlazom
   component siso_reg4 is
      port (clk: in std_logic;
            d:    in std_logic;
            q:    out std_logic);
   end component siso_reg4;

   -- Deklaracija unutrašnjih signala potrebnih za povezivanje stimulus
   -- generatora sa ulazima DUV-a
   signal clk_s: std_logic;
   signal d_s: std_logic;
   signal q_s: std_logic;

begin
        -- Komponenta koja se verifikuje
        duv: siso_reg4
           port map (
              clk => clk_s,
              d => d_s,
              q => q_s);

        -- Klok generator koji generise periodicni clk_s signal koji
        -- ce se koristiti za aktiviranje registra
        clk_gen: process
        begin
                clk_s <= '0', '1' after 100 ns;
                wait for 200 ns;
        end process;

        -- 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
                d_s <= '0', '1' after 75 ns, '0' after 275 ns;

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

Prikazani testbenč instancionira komponentu 4-bitnog registra i koristeći *clk_gen* i *stim_gen* procese generiše povorku ulaznih signala koja može da se iskoristi za proveru ispravnog rada napisanog modela registra.

Clk_gen proces generiše periodičnu povorku impulsa na signalu *clk_s* koji se dovodi na *clk* ulaz registra.

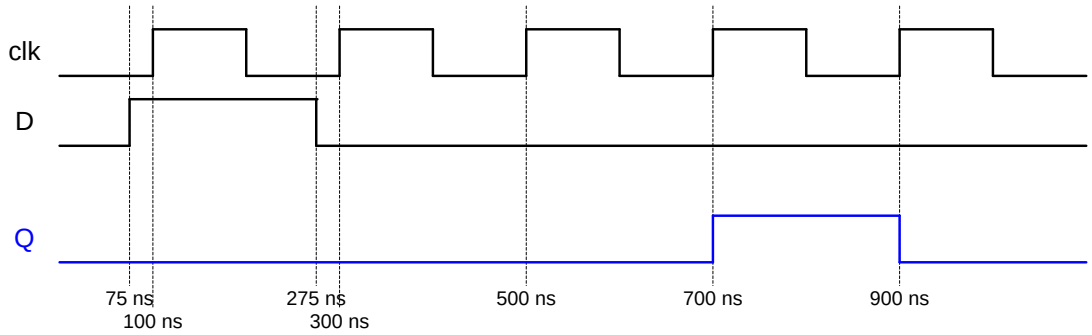
Stim_gen proces generiše vremenske oblike za signal *d_s* koristeći jednu naredbu sekvencijalne dodele vrednosti signalu.

Talasni oblici koji će biti generisani pomoću ove dve process naredbe prikazani su na slici 17.



Slika 17. Generisani talasni oblici na dva ulazna signala 4-bitnog registra sa serijskim upisom i čitanjem

Nakon što se izvrši simulacija kombinovanog rada testbenča i modela 4-bitnog registra sa serijskim upisom i paralelnim čitanjem, talasni oblik izlaznog signala podataka *q_s* trebalo bi da izgleda kao na slici 18.



Slika 18. Generisani talasni oblici na dva ulazna 4-bitnog registra sa serijskim upisom i čitanjem zajedno sa talasnim oblicima izlaznog signala podataka *q_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 4-bitnog registra sa serijskim upisom i čitanjem.

Zadaci za vežbu

Zadatak 1:

Napisati VHDL model 4-bitnog registra sa serijskim upisom i čitanjem koji će pored ulanzog porta za podatke imati i sledeće ulazne portove:

- *clock enable* (*ce*) signal za selekciju rastućih ivica *clk* ulaza na koje registar treba da reaguje
- ulazni portove za inicijalizaciju sadržaja registra – *reset*

Za razvijeni VHDL model napisai odgovarajući testbenč pomoću kojega će biti moguće izvršiti funkcionalnu verifikaciju modela.

Zadatak 2:

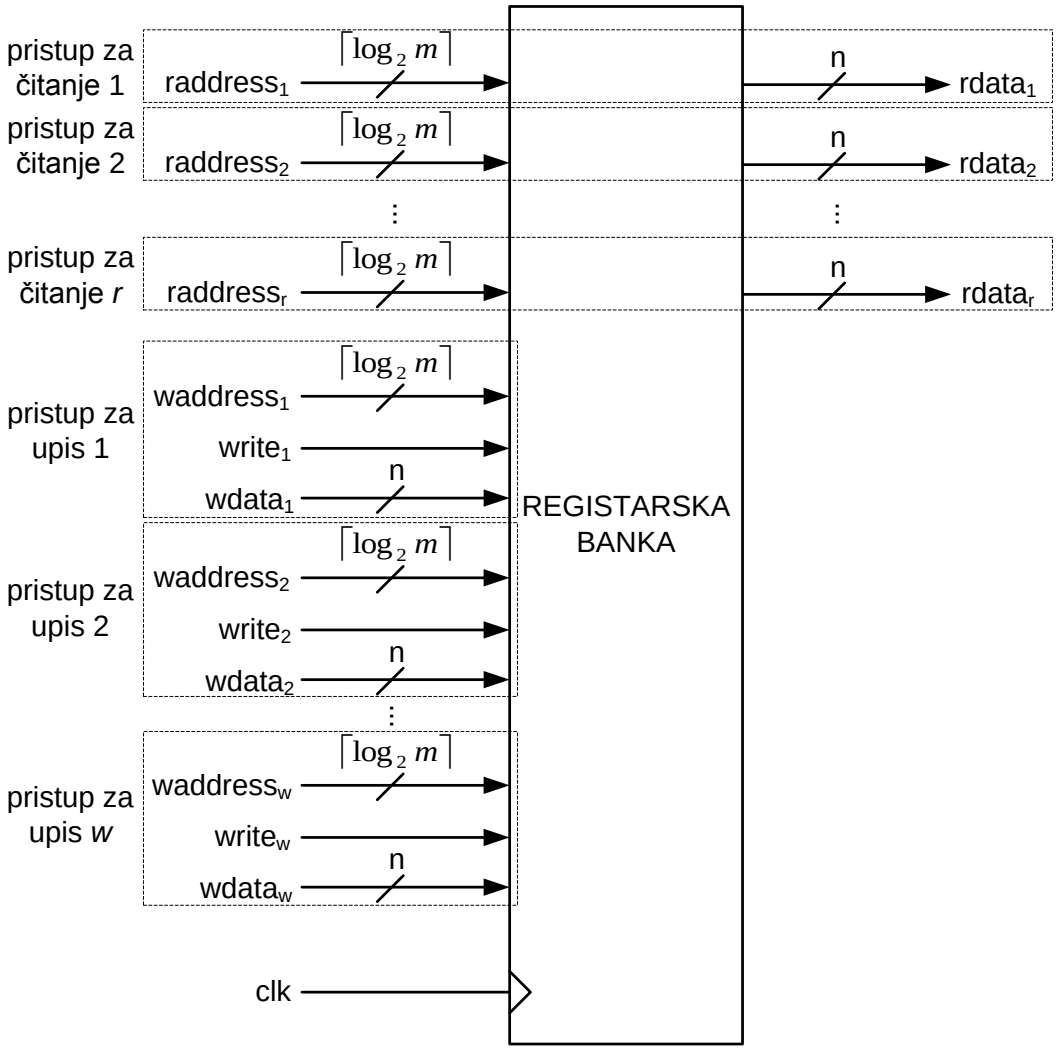
Napisati VHDL model 4-bitnog registra sa serijskim upisom i čitanjem koji će pored ulanzog porta za podatke imati i sledeće ulazne portove:

- *clock enable* (*ce*) signal za selekciju rastućih ivica *clk* ulaza na koje registar treba da reaguje
- ulazni port dozvole upisa novog sadržaja u registar - *we* (*write enable*)
- ulazni portove za inicijalizaciju sadržaja registra – *set*

Za razvijeni VHDL model napisai odgovarajući testbenč pomoću kojega će biti moguće izvršiti funkcionalnu verifikaciju modela.

Registarske banke

Registarska banka objedinjuje skup registara sa paralelnim upisaom i čitanjem unutar jedne komponente. Registarska banka po pravilu obezbeđuje veći broj pristupa za čitanje i upis sadržaja u selektovane registra unutar banke. Na ovaj način omogućen je istovremeni pristup većem broju registara unutar registarske banke. Ovaj istovremeni pristup je od velikog značaja prilikom projektovanja mikroprocesora, tako da se registarske banke najčešće koriste unutar mikroprocesora za realizaciju skup registara opšte namene. Osnovni interfejs registarske banke sastavljene od *m n*-bitnih registara sa paralelnim upisom i čitanjem prikazan je na slici 19.



Slika 19. Interfejs registarske banke sastavljene od *m n*-bitnih registara sa paralelnim upisom i čitanjem, *r* pristupa za čitanje podataka i *w* pristupa za upis podataka

Registarska banka u opštem slučaju poseduje sledeće portove:

- *r* pristupa za čitanje podataka upisanih u registre,
- *w* pristupa za upis novih podataka u registre unutar banke
- *clk* ulaz za sinhronizaciju rada registarske banke
- portove za inicijalizaciju sadržaja registara (*reset*, *set*, *clear*, *preset*)

Svaki od pristupa za čitanje podataka sastoji se iz sledećih portova:

- *raddress*, ulazne adresne magistrale koja služi za adresiranje registra čiji sadržaj se želi pročitati; moguće vrednosti adresa su iz opsega [0 *m*-1] celih brojeva
- *rdata*, izlazne magistrale podataka preko koje se dobija trenutni sadržaj adresiranog registra

Svaki od pristupa za upis podataka sastoji se iz sledećih portova:

- *waddress*, ulazne adresne magistrale koja služi za adresiranje registra u koji želimo upisati novi sadržaj; moguće vrednosti adresa su iz opsega [0 *m*-1] celih brojeva
- *wdata*, ulazne magistrale podataka preko koje se prosleđuje podatak koji je potrebno upisti u adresirani registar
- *write*, ulaznog porta dozvole upisa novog podatka u adresirani registar

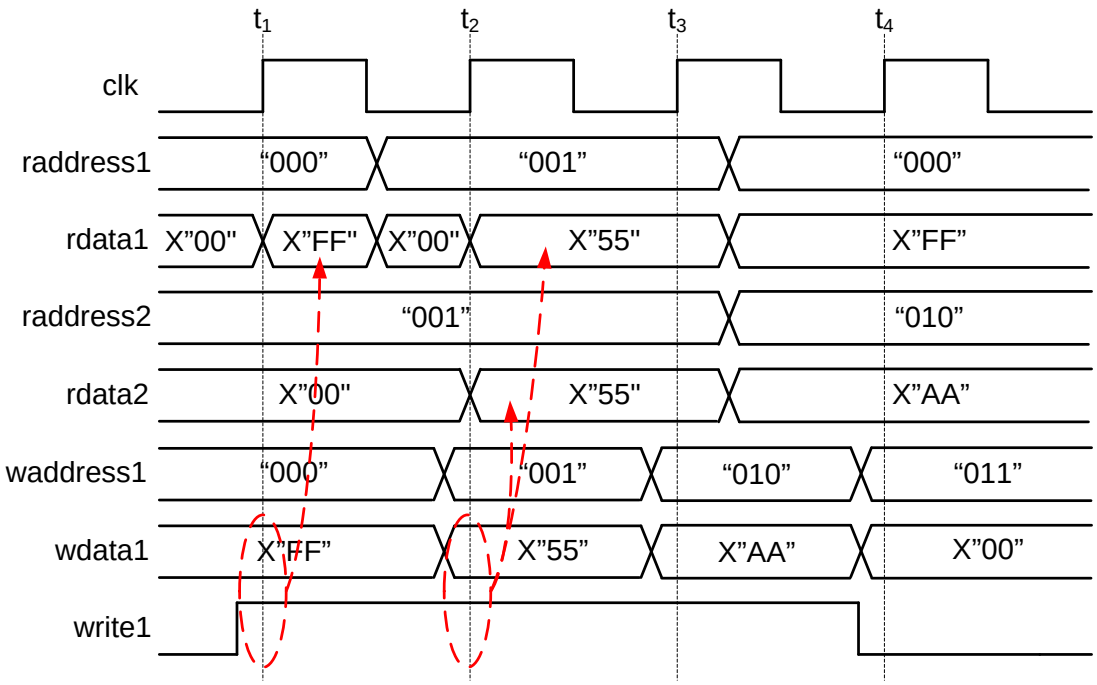
Kao što je već rečeno, registarska banka dozvoljava istovremeni pristup većem broju registara unutar banke radi izvršavanja operacija čitanja ili upisa sadržaja. Preko svakog od *r* pristupa za čitanje može se adresirati željeni registar unutar banke (postavljajući njegovu adresu na odgovarajući *raddress* ulazni port), a njegov sadržaj pojaviće se na odgovarajućem *rdata* izlaznom portu.

U zavisnosti od trenutka pojavljivanja ovog sadržaja na *rdata* portu razlikujemo dva načina čitanja podataka iz registarske banke:

- **sinhrono čitanje** - kod kojega se sadržaj adresiranog registra pojavlje u rdata portu prilikom nailaska sledeće rastuće ivice clk signala,
- **asinhrono čitanje** - kod kojega se sadržaj adresiranog registra pojavljuje odmah nakon stabilizacije nove adrese na raddress portu.

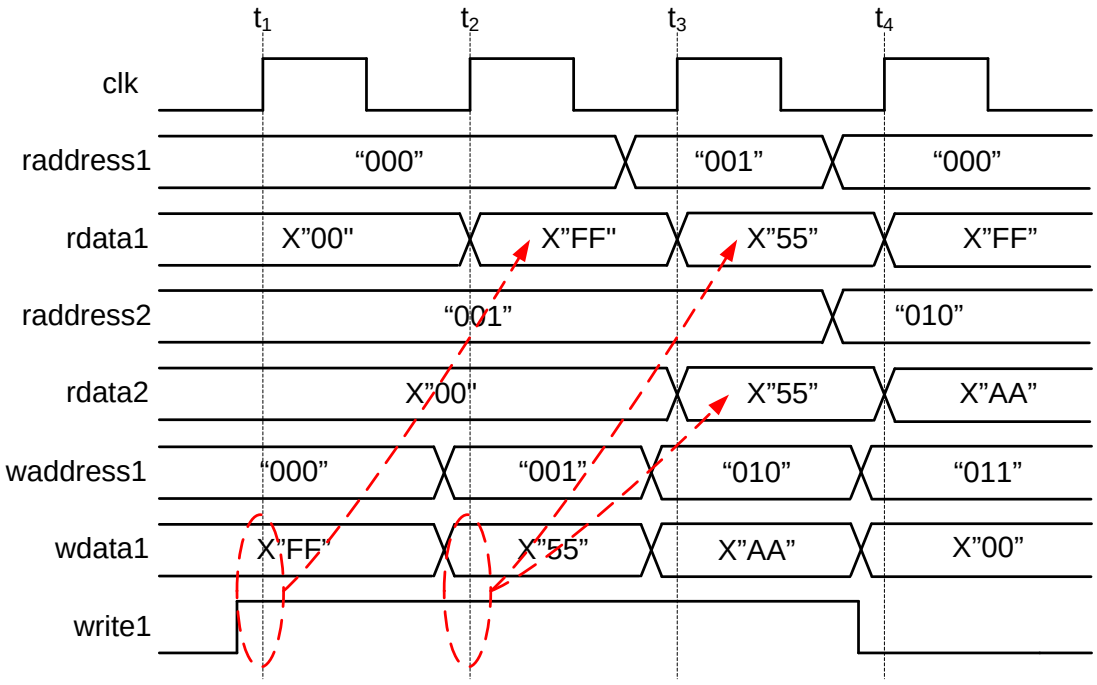
Upis podataka u registarsku banku vrši se preko *w* pristupa za upis. Postavljanjem adrese registra u koji želimo upisati podatak na *waddress* ulazni port, adresira se željeni registar unutar banke. Podatak koji se želi upisati u adresirani registar postavlja se na *wdata* ulazni port. Sam upis podatka je uvek sinhroni sa *clk* signalom, i aktivira se kada se *write* ulazni port postavi na vrednost 1.

Na primer, vremenski dijagram rada 8-bitne registarske banke sa asinhronim čitanjem, sastavljene od 8 registara, sa 2 kanala za čitanje i 1 kanalom za upis prikazan je na slici 20.



Slika 20. Vremenski dijagram rada 8-bitne registarske banke sa asinhronim čitanjem, sastavljene od 8 registara, sa 2 kanala za čitanje i 1 kanalom za upis

Vremenski dijagram rada iste 8-bitne registarske banke, ali ovaj put sa sinhronim čitanjem, sastavljene od 8 registara, sa 2 kanala za čitanje i 1 kanalom za upis prikazan je na slici 21.



Slika 21. Vremenski dijagram rada 8-bitne registarske banke sa sinhronim čitanjem, sastavljene od 8 registara, sa 2 kanala za čitanje i 1 kanalom za upis

Kao što se sa slika 20 i 21 može videti, razlika između banki sa asinhronim i sinhronim čitanjem ogleda se u brzini odziva banke prilikom operacije čitanja. U slučaju asinhronog čitanja, željeni podatak dostupan je u istom taktu kada se inicira

proces čitanja, dok je kod sinhronog čitanja željeni podatak dostupan tek u sledećem taktu. Registarske banke sa sinhronim čitanjem unose kašnjenje prilikom operacije čitanja od 1 periode *clk* takt signala.

VHDL modeli registarske banke

Da bi smo ilustrovali različite načine modelovanja registrske banke, koristićemo dve verzije 8-bitne registarske banke sastavljene od 8 registara sa 2 kanala za čitanje i 1 kanalom za upis:

- verzija 1 – banka sa asinhronim čitanjem,
- verzija 2 – banka sa sinhronim čitanjem.

U oba slučaja e*ntity* deklaracija registarske banke ima sledeći izgled

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

entity reg_bank is
   port (clk:          in std_logic;

         -- kanal za citanje 1
         raddress1: in std_logic_vector(2 downto 0);
         rdata1:      out std_logic_vector(7 downto 0);

         -- kanal za citanje 2
         raddress2: in std_logic_vector(2 downto 0);
         rdata2:      out std_logic_vector(7 downto 0);

         -- kanal za upis 1
         waddress1: in std_logic_vector(2 downto 0);
         wdata1:     in std_logic_vector(7 downto 0);
         write1:      in std_logic
        );
end entity reg_bank;
```

Nakon što smo opisali interfejs registarske banke, potrebno je modelovati njenu funkcionalnost. Za ovo je potrebno napisati odgovarajuće arhitekturno telo.

Bihevijani model

Varijanta 1: Model registarske banke sa asinhronim čitanjem

```
architecture beh1 of reg_bank is
   type reg_file_t is array (0 to 7)
                        of std_logic_vector(7 downto 0);
   signal reg_file_s: reg_file_t;
begin
   -- proces koji modeluje upis u registarsku banku
   write_reg_file: process (clk) is
```

```
   begin
      if (clk'event and clk = '1') then
         if (write1 = '1') then
            reg_file_s(conv_integer(waddress1)) <= wdata1;
         end if;
      end if;
   end process;

   -- asinhrono citanje iz registarske banke
   rdata1 <= reg_file_s(conv_integer(raddress1));
   rdata2 <= reg_file_s(conv_integer(raddress2));
end architecture beh1;
```

Varijanta 2: Model registarske banke sa sinhronim čitanjem

```
architecture beh2 of reg_bank is
   type reg_file_t is array (0 to 7)
                        of std_logic_vector(7 downto 0);
   signal reg_file_s: reg_file_t;
begin
   -- proces koji modeluje upis u registarsku banku
   write_reg_file: process (clk) is
   begin
      if (clk'event and clk = '1') then
         if (write1 = '1') then
            reg_file_s(conv_integer(waddress1)) <= wdata1;
         end if;
      end if;
   end process;

   -- proces koji modeluje sinhrono citanje iz registarske banke
   read_reg_file: process (clk) is
   begin
      if (clk'event and clk = '1') then
         rdata1 <= reg_file_s(conv_integer(raddress1));
         rdata2 <= reg_file_s(conv_integer(raddress2));
      end if;
   end process;
end architecture beh2;
```

Kombinovanjem entity deklaracije i odgovarajućeg arhitekturnog tela dobijamo kompletan model registarske banke.

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


entity reg_bank is
   port (clk:        in std_logic;
```

```
        -- kanal za citanje 1
        raddress1: in std_logic_vector(2 downto 0);
        rdata1:    out std_logic_vector(7 downto 0);

        -- kanal za citanje 2
        raddress2: in std_logic_vector(2 downto 0);
        rdata2:    out std_logic_vector(7 downto 0);

        -- kanal za upis 1
        waddress1: in std_logic_vector(2 downto 0);
        wdata1:    in std_logic_vector(7 downto 0);
        write1:    in std_logic
       );
end entity reg_bank;


architecture beh1 of reg_bank is
   type reg_file_t is array (0 to 7)
                      of std_logic_vector(7 downto 0);
   signal reg_file_s: reg_file_t;
begin
   -- proces koji modeluje upis u registarsku banku
   write_reg_file: process (clk) is
   begin
      if (clk'event and clk = '1') then
         if (write1 = '1') then
            reg_file_s(conv_integer(waddress1)) <= wdata1;
         end if;
      end if;
   end process;

   -- asinhrono citanje iz registarske banke
   rdata1 <= reg_file_s(conv_integer(raddress1));
   rdata2 <= reg_file_s(conv_integer(raddress2));
end architecture beh1;
```

Verifikaciono okruženje

Primer jednog mogućeg testbenča koji se može iskoristiti za verifikaciju registarske banke prikazan je u nastavku.

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

entity reg_bank_tb is
end entity reg_bank_tb;


architecture beh of reg_bank_tb is
   -- Deklaracija unutrašnjih signala potrebnih za povezivanje stimulus
   -- generatora sa ulazima DUV-a
```

```
    signal clk_s: std_logic;
    signal raddress1_s, raddress2_s: std_logic_vector(2 downto 0);
    signal rdata1_s, rdata2_s: std_logic_vector(7 downto 0);
    signal waddress1_s: std_logic_vector(2 downto 0);
    signal wdata1_s: std_logic_vector(7 downto 0);
    signal write1_s: std_logic;

begin
        -- Komponenta koja se verifikuje
        duv: entity work.reg_bank(beh1)
            port map (
                clk => clk_s,
                -- kanal 1 za citanje
                raddress1 => raddress1_s,
                rdata1 => rdata1_s,

                -- kanal 2 za citanje
                raddress2 => raddress2_s,
                rdata2 => rdata2_s,

                -- kanal 1 za upis
                waddress1 => waddress1_s,
                wdata1 => wdata1_s,
                write1 => write1_s);

        -- Klok generator koji generise periodicni clk_s signal koji
        -- ce se koristiti za aktiviranje registra
        clk_gen: process
        begin
                clk_s <= '0', '1' after 100 ns;
                wait for 200 ns;
        end process;

        -- 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
            -- Inicijalizacija
            raddress1_s <= (others => '0');
            raddress2_s <= (others => '0');
            waddress1_s <= (others => '0');
            wdata1_s <= (others => '0');
            write1_s <= '0';

            wait for 200 ns;
            -- Upisimo prvo podatke u registarsku banku
            write1_s <= '1';
            for i in 0 to 7 loop
                waddress1_s <= conv_std_logic_vector(i, 3);
                wdata1_s <= conv_std_logic_vector(2*i+1, 8);
                wait for 200 ns;
            end loop;
```

```
            write1_s <= '0';
            wait for 200 ns;

            -- Procitajmo podatke koristeci oba porta za citanje
            for i in 0 to 7 loop
               raddress1_s <= conv_std_logic_vector(i, 3);
               raddress2_s <= conv_std_logic_vector(7-i, 3);
               wait for 200 ns;
            end loop;

            wait for 200 ns;

            -- Istovremeno citanje i upis u registarsku banku
            write1_s <= '1';
            for i in 0 to 7 loop
               waddress1_s <= conv_std_logic_vector(i, 3);
               wdata1_s <= conv_std_logic_vector(2*i+2, 8);
               raddress1_s <= conv_std_logic_vector(i, 3);
               raddress2_s <= conv_std_logic_vector(7-i, 3);
               wait for 200 ns;
            end loop;

            write1_s <= '0';
            wait;
         end process;
end architecture beh;
```

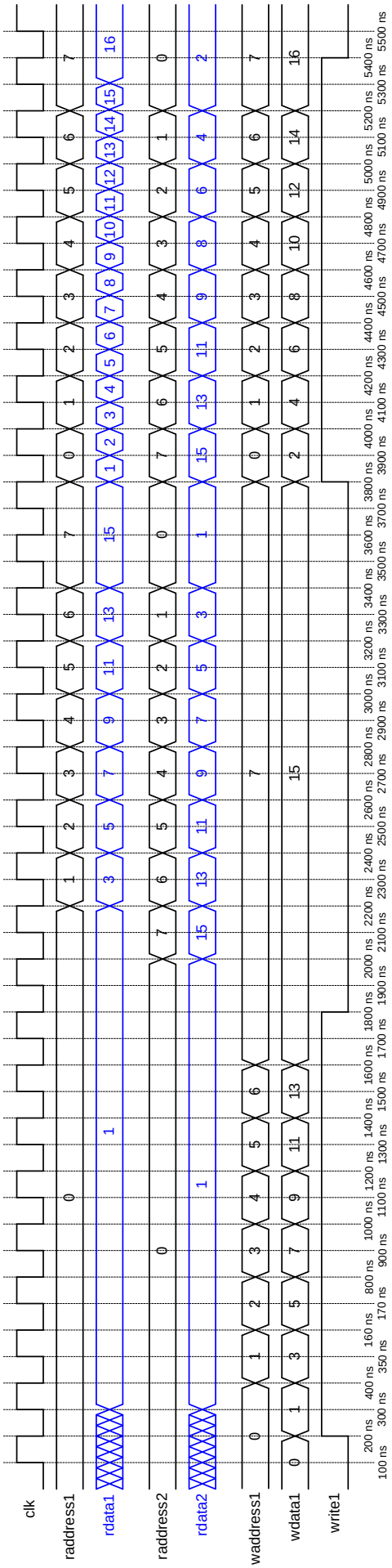
Prikazani testbenč instancionira komponentu registarske banke i koristeći *clk_gen* i *stim_gen* procese generiše povorku ulaznih signala koja može da se iskoristi za proveru ispravnog rada napisanog modela registarske banke.

Clk_gen proces generiše periodičnu povorku impulsa na signalu *clk_s* koji se dovodi na *clk* ulaz registarske banke.

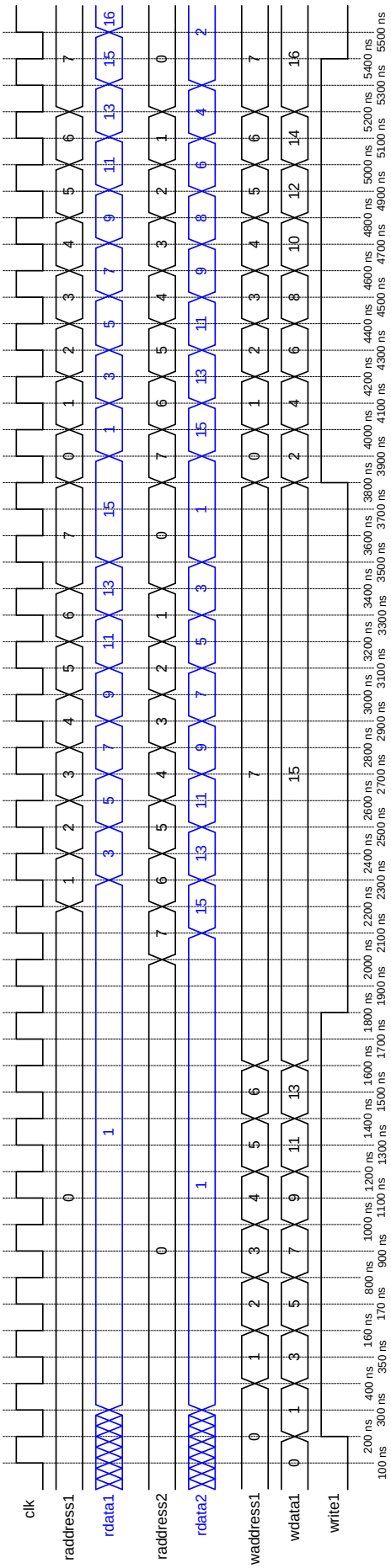
Stim_gen proces generiše vremenske oblike za adresne magistrale pristupa za čitanje i upis podataka (*raddress1_s*, *raddress2_s* i *waddress1_s* signali). Takođe, proces generiše i potrebne vremenske oblike za magistralu podataka pristupa za upis podataka (*wdata1_s*), kao i za signal dozvole upisa, *write1_s*.

Talasni oblici koji će biti generisani pomoću ove dve process naredbe, kao i signali koji su rezultat simulacije, u sluačaju verifikacije registarske banke sa asinhronim čitanjem, prikazani su na slici 22.

On što je posebno potrebno naglasiti na slici 22, jeste asinhrono čitanje podataka iz registrske banke. Cim se uspostavi nova vrednost na adresnoj magistrali odgovarajućeg pristupa za čitanje, podaci iz selektovanih registara odmah postaju dostupni preko izlaznih magistrala za podatake (*rdata1_s* i *rdata2_s*). Takođe, obzirom da je čitanje asinhrono, svaka promena sadržaja adresiranog registra odmah postaje vidljiva na odgovarajućem izlaznom portu podataka (situacija počevši od vremena 3800 ns na *rdata1_s* i *rdata2_s* signalima).



Slika 22. Rezultat simulacije rada registrarske banke sa asinhronim čitanjem



Slika 23. Rezultat simulacije rada registrarske banke sa sinhronim čitanjem

Na slici 23 prikazani su rezultati simulacije rada registraske banke sa sinhronim čitanjem podataka. Upoređivanjem slika 22 i 23 možemo uočiti da se na slici 23 nove vrednosti na izlaznim portovima podataka (*rdata1_s* i *rdata2_s* signali) pojavljuju uvek na rastuću ivicu *clk* signala. Ovo je upravo ono što smo i mogli očekivati jer je ovaj put reč o registarskoj banci sa sinhronim čitanjem podataka, što znači da se izlazni portovi podataka užuriraju tek nailaskom rastuće ivice *clk* signala. Zbog ovakvog načina rada, takođe možemo primetiti da izlazni podaci „kasne“ u odnosu na izlazne podatke kod registarske banke sa asinhronim čitanjem. Ovo je eksperimentalna potvrda činjenice da je odziv registarskih banki sa sinhronim čitanjem sporiji od odziva registarskih banki sa sinhronim odzivom. Kod registarskih banki sa asinhronim odzivom pročitani podaci su na raspolaganju u istom taktu kada je generisana i adresa registra koji se želi pročitati, dok su u slučaju registarskih banki sa sinhronim čitanjem podaci dostupni tek u narednom periodu klok signala, *clk*. Ovo je takođe jasno vidljivo na slikama 22 i 23.

Zadaci za vežbu

Zadatak 1:

Napisati VHDL model registarske banke sa sledećim karakteristikama:

- broj registara u banci, $m = 32$
- broj pristupa za upis, $w = 2$
- broj pristupa za čitanje, $r = 2$
- tip čitanja podataka iz registarske banke – asinhrono
- sinhroni *reset* ulaz
- sinhroni *clock enable* ulaz
- sinhroni *enable* ulaz

Za razvijeni VHDL model napisai odgovarajući testbenč pomoću kojega će biti moguće izvršiti funkcionalnu verifikaciju modela.