Laboratorijska vežba 4

Opis standardnih sekvencijalnih mreža korišćenjem VHDL jezika - 1

Memorijski elementi

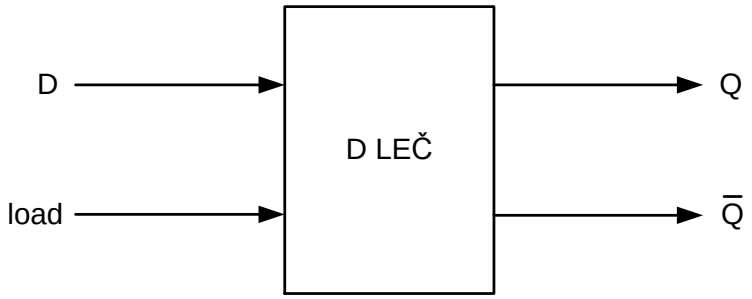
U prethodne dve vežbe bavili smo se kombinacionim mrežama kod kojih trenutno stanje na izlazima iz mreže zavise samo od trenutnih vrednosti ulaznih siglana. Postoji i druga grupa mreža kod kojih trenutno stanje na izlazima iz mreže ne zavisi samo od trenutnih vrednosti na ulazima mreže već i od prethodnog ponašanja mreže (unutrašnjeg stanja u kome se mreža trenutno nalazi). Ovakve mreže po pravilu uključuju memorijske elemente koji su u stanju da memorišu odgovarajuće logičke vrednosti. Trenutni sadržaj memorijskih elemenata definiše *trenutno (tekuće) stanje* u kome se mreža nalazi. Svake promena na ulazima u mrežu potencijlano može dovesti do promene stanja u kome se mreža nalazi. Logičke mreže koje sadrže memorijske elemente nazivaju se *sekvencijalne mreže*.

U zavisnosti od načina memorisanja logičkih vrednosti, svi memorijski elementi mogu se podeliti u dve velike grupe:

- lečeve (*latch*) - memorijske elemente kontrolisane nivoom signala
- flip-flopove (*flip-flop*) - memorijske elemente kontrolisane ivicom signala

Lečevi

Leč predstavlja memorijski element koji je kontrolisan nivoom odgovarajućeg ulaznog signala. Leč se realizuje pomoću odgovarajuće asinhrone kombinacione mreže koja obavezno sadrži povratne sprege od izlaza ka ulazima. Postoji veliki broj različitih tipova lečeva: SR leč, D leč, SR leč sa dozvolom upisa, D leč sa dozvolom upisa, itd. U praksi se najčešće koristi D leč sa dozvolom upisa. Interfejs D leča sa dozvolom upisa prikazan je na slici 1.



Slika 1. Interfejs D leča sa dozvolom upisa

D leč sa dozvolom upisa ima dva ulazna porta:

- *D* ulaz za podatke
- *Load* ulaz za dozvolu upisa

D leč sa dozvolom upisa u opštem slučaju ima i dva izlazna porta:

- *Q* izlaz za podatke

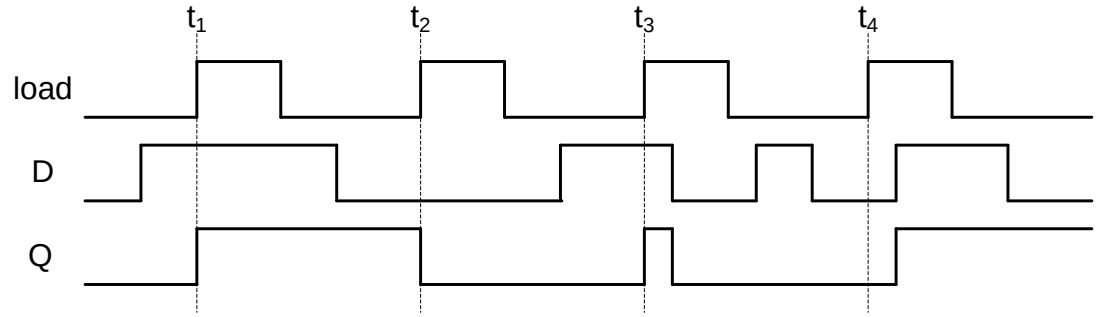
- $\overline{Q}$ invertovani izlaz za podatke

NAPOMENA: U praksi se najčešće koristi D leč koji ima samo jedan izlaz, izlaz Q.

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

Load	*D*	*Q*(t+1)	$\overline{Q}$(t+1)
0	0	*Q*(t)	$\overline{Q}$(t)
0	1	*Q*(t)	$\overline{Q}$(t)
1	0	0	1
1	1	1	0

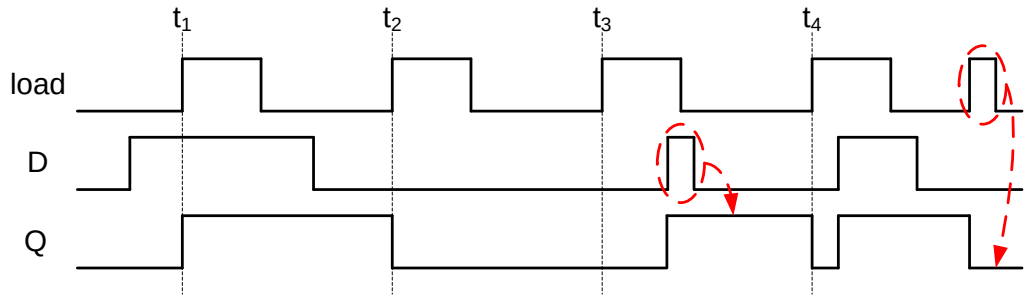
Kao što se može primetiti na osnovu prethodne tabele, sve dok je signal dozvole upisa *load* neaktivan (ima vrednost 0), D leč čuva prethodno upisanu vrednost i ignoriše vrednosti koje mu se trenutno nalaze na *D* ulazu. Tek kada signal dozvole upisa load postane aktivan (dobije vrednost 1), u D leč se upisuje trenutna vrednost koja se nalazi na *D* ulazu. Vremenski dijagram rada D leča sa dozvolom upisa prikazan je na slici 2.



Slika 2. Vremenski dijagram rada D leča sa dozvolom upisa

Ranije je rečeno da su lečevi memorijski elementi koji su kontrolisani nivoom ulaznog signala. Ovo se lepo može videti na vremenskom dijagramu sa slike 2. U trenutku t_3 *load* ulaz postaje 1, i obzirom da je trenutna vrednost *D* ulaza jednaka 1, *Q* izlaz leča dobija vrednost 1. Međutim, tokom perioda u kojem *load* ulaz ima vrednost 1, *D* ulaz menja svoju vrednost sa 1 na 0. S obzirom da je D leč kontrolisan nivoom signala *load*, sve dok je *load* ulaz ima vrednost 1 svaka promena na *D* ulazu će biti "upisivana" u leč, odnosno možemo reći da će leč "pratiti" promene na svom *D* ulazu. Nova vrednost D leča biće jednaka onoj vrednosti D ulaza koja je važila u trenutku kada se deaktivirao signal dozvole upisa *load*, u našem primeru to je vrednost 0. Svaka naredna promena na D ulazu biće ignorisana od strane leča ukoliko je signal dozvole upisa *load* neaktivan (ima vrednost 0), što se takođe može videti na slici 1.

Iako je mogućnost kontrole upisa nivoom signala ponekad poželjna, ovakvi elementi se moraju koristiti vrlo pažljivo. Razlog zbog čega je to tako leži u činjenici da će očekivano ponašanje leča biti kompromitovano ukoliko se na njegove ulaze dovedu signali na kojima postoje gličevi. Ova situacija je ilustrovana na slici 3, u slučaju prisustva gličeva na *load* i *D* ulazima.



Slika 3. Vremenski dijagram rada D leča u slučaju prisustva gličeva na *load* i *D* ulazima

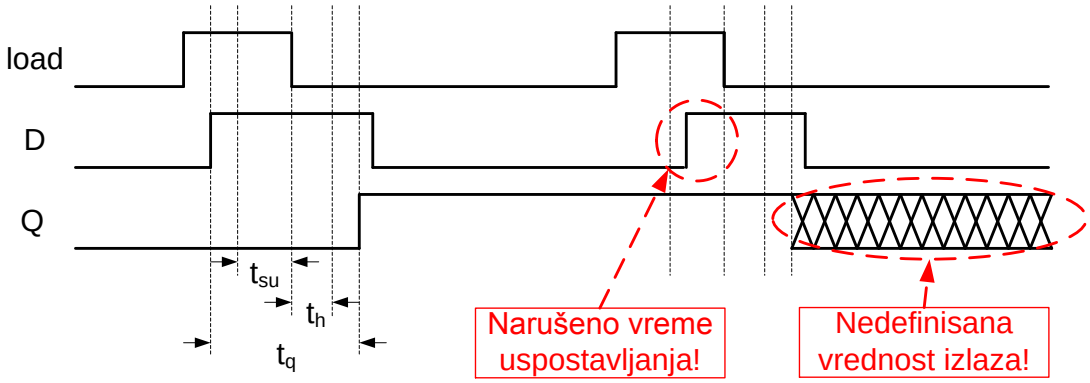
Pojava gliča na *D* ulazu u trenutku kada je signal dozvole upisa aktivan može dovesti do upisa nekorektne vrednosti u leč, kao što je ilustrovano na slici 3 u ciklusu upisa koji počinje u trenutku t_3. Slično, pojava gliča na signalu dozvole upisa, *load*, takođe može dovesti do upisa pogrešne vrednosti u leč, što se vidi na slici 3 u ciklusu upisa koji počinje u trenutku t_4.

Ukoliko se u sistemu koriste lečevi, da bi se obezbedio ispravan i pouzdan rad sistema, mora se garantovati da će svi signali povezani na ulazne portove biti bez gličeva. Kao što ćemo uskoro videti, ovaj zahtev nije neophodan ukoliko koristimo drugu vrstu memorijskih elemenata, koji su kontrolisani ivicom signala, flip flopove.

Generisanje signala bez gličeva jako je teško i po pravilu zahteva primenu posebnih tehnika projektovanja koje povećavaju složenost projektovanih mreža i usporavaju njihov rad. Ovo je razlog zašto se prilikom projektovanja sistema u kojima ima potrebe za korišćenjem memorijskih elemenata ne preporučuje korišćenje lečeva.

U prethodnim analizama nismo uzimali u obzir konačnu brzinu prostiranja signala kroz elektronske komponente od kojih je sastavljen leč. Iako je ovakvo idealizovano modelovanje pogodno u nekim slučajevima, prilikom razvoja sistema koji će se na kraju i implementirati pomoću konkretnog digitalnog elektronskog kola kašnjenja signala kroz sistem, usled konačne brzine prostiranja, moraju se uzeti u obzir. Na primer, prethodno pominjani D leč radiće ispravno samo ukoliko se signal povezan na *D* ulaz ne menja oko trenutka kada se signal na *load* ulazu menja sa 1 na 0. Promena signala na D ulazu leča tokom ovog „prozora“ dovešće do pojave nepredvidivih rezultata na *Q* izlazu D leča. Zbog toga dizajner mora da obezbedi da mreža koja generiše *D* signal generiše signal koji će biti stabilan tokom ovog kritičnog perioda.

Minimalni vremenski interval tokom kojega signal na *D* ulazu u leč mora biti stabilan pre nailaska opadajuće ivice *load* signala naziva se ***vreme uspostavljanja* (*setup time*), t_{su}**. Minimalni vremenski interval tokom kojega signal na D ulazu u leč mora biti stabilan nakon nailaska opadajuće ivice *load* signala naziva se **vreme držanja (hold time), t_h**. Poslednji vremenski parametera koji opisuje rad leča, **jeste kašnjenje na izlazu *Q*, t_q**, koje se definiše kao maksimalni vremenski interval koji mora da protekne od nailaska opadajuće ivice *load* signala nakon kojega *Q* izlaz dobija novu stabilnu vrednost. Slika 4 ilustruje vremenske karakteristike D leča.



Slika 4. Vremenske karakteristike D leča

VHDL modeli D leča sa dozvolom upisa

Kao što se može videti sa slike 1, D leč sa dozvolom upisa ima sledeće portove:

- 1-bitni ulazni port za podatke, *D*
- 1-bitni ulazni port za kontrolu upisa, *load*
- 1-bitni izlazni port za podatke, *Q*

Odogovarajuća *entity* deklaracija D leča sa dozvolom upisa ima sledeći izgled

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

entity d_latch is
        port (d:        in std_logic;   -- ulazni port podataka
              load:     in std_logic;   -- ulazni port dozvole upisa
              q:        out std_logic); -- izlazni port podataka 1
end entity d_latch;
```

Nakon što smo opisali interfejs D leča sa dozvolom upisa, potrebno je modelovati njegovu funkcionalnost. Za ovo je potrebno napisati odgovarajuće arhitekturno telo.

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

Bihevijani modeli

Varijanta 1: Model D leča dozvolom upisa baziran na korišćenju konkurentne naredbe uslovne dodele vrednosti signalu

```
architecture beh1 of d_latch is
begin
        q <= d when load = '1';
end architecture beh1;
```

Varijanta 2: Model D leča sa dozvolom upisa baziran na korišćenju *process* i *if* naredbe

```
architecture beh2 of d_latch is
begin
        dlatch: process (d, load) is
        begin
                if (load = '1') then
                        q <= d;
                end if;
        end process;
end architecture beh2;
```

NAPOMENA: Priloženi modeli pokazuju da se leč modeluje korišćenjem nepotpunih naredbi uslovne dodele signalu, kod kojih nije specificirana nova vrednost koju signal treba da dobije za svaki od mogućih ishoda uslova koji se testira. Međutim, ovakav način modelovanja u praksi često rezultuje u neželjenom uvođenju leča u sistem koji se modeluje. Često se prilikom pisanja modela kobinacionih mreža, koji uključuju korišćenje *if* naredbi, ne definišu nove vrednosti signala kojima se manipuliše unutar *if* naredbe za svaki od mogućih ishoda testa unutar *if* naredbe. Ukoliko se izvrši automatska sinteza ovako napisanog modela rezultat će biti pojava neželjenih lečeva.

Kombinovanjem entity deklaracije i odgovarajućeg arhitekturnog tela dobijamo kompletan model D leča sa dozvolom upisa. Na primer, dole je prikazan model D leča sa dozvolom upisa koji se bazira na funkcionalnom modelu koji koristi *process* naredbu.

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

entity d_latch is
        port (d:        in std_logic;   -- ulazni port podataka
              load:     in std_logic;   -- ulazni port dozvole upisa
              q:        out std_logic);-- izlazni port podataka 1
end entity d_latch;

architecture beh2 of d_latch is
begin
        dlatch: process (d, load) is
        begin
                if (load = '1') then
                        q <= d;
                end if;
        end process;

end architecture beh2;
```

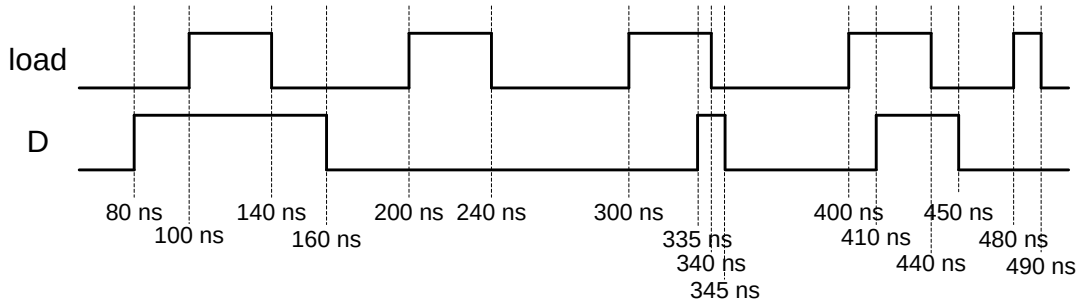
Verifikaciono okruženje

Primer jednog mogućeg testbenča koji se može iskoristiti za verifikaciju D leča sa dozvolom upisa prikazan je u nastavku.

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

entity d_latch_tb is
end entity d_latch_tb;


architecture beh of d_latch_tb is
   -- Deklaracija komponente VHDL modula koji se verifikuje (DUV)
   -- U ovom slučaju je to D latch sa dozvolom upisa
   component d_latch is
        port (d:      in std_logic;   -- ulazni port podataka
              load:   in std_logic;   -- ulazni port dozvole upisa
              q:      out std_logic);-- izlazni port podataka 1
   end component d_latch;

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

begin
        -- Komponenta koja se verifikuje
        duv: d_latch
           port map (
              d => d_s,
              load => load_s,
              q => q_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
                load_s <= '0', '1' after 100 ns, '0' after 140 ns,
                                '1' after 200 ns, '0' after 240 ns,
                                '1' after 300 ns, '0' after 340 ns,
                                '1' after 400 ns, '0' after 440 ns,
                                '1' after 480 ns, '0' after 490 ns;

                d_s <= '0', '1' after 80 ns, '0' after 160 ns,
                             '1' after 335 ns, '0' after 345 ns,
                             '1' after 410 ns, '0' after 450 ns;

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

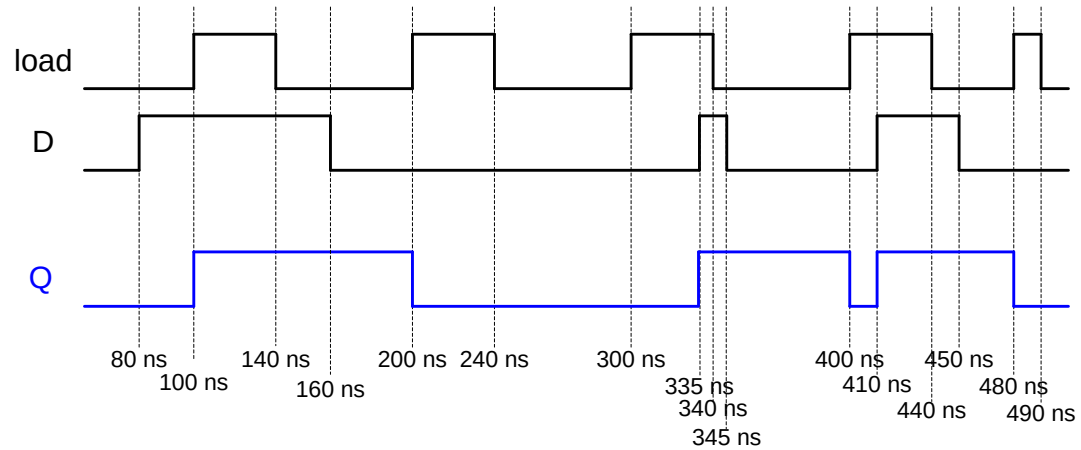
Prikazani testbenč instancionira komponentu D leč i koristeći *stim_gen* proces generiše povorku ulaznih signala koja može da se iskoristi za proveru ispravnog rada napisanog modela D leča.

Stim_gen proces generiše vremenske oblike za ukupno dva signala, *d_s* i *load_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 5.



Slika 5. Generisani talasni oblici na dva ulazna signala D leča sa dozvolom upisa

Nakon što se izvrši simulacija kombinovanog rada testbenča i modela D leča sa dozvolom upisa talasni oblik izlaznog signala podataka *q_s* trebalo bi da izgleda kao na slici 6.



Slika 6. Generisani talasni oblici na dva ulazna signala D leča sa dozvolom upisa 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 D leča sa dozvolom upisa.

Zadaci za vežbu

Zadatak 1:

Napisati VHDL model D leča sa dozvolom upisa koji će modelovati konačnu brzinu prostiranja signala kroz leč. Vremenske karakteristike D leča koji je potrebno modelovati imaju sledeće vrednosti:

- vreme uspostavljanja, t_{su} = 2 ns
- vreme držanja, t_h = 1.5 ns
- kašnjenje na izlazu, t_q = 3 ns

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

Zadatak 2:

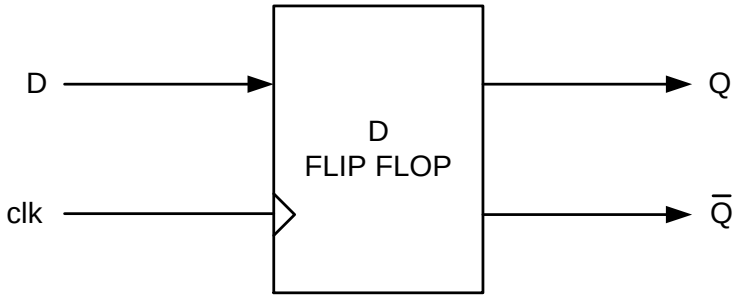
Proširiti VHDL model D leča sa dozvolom upisa iz zadatka 1 tako da uključi:

- vremenske provere zadovoljenosti vremena uspostavljanja i držanja na *D* ulazu leča 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.

Flip flopovi

Flip flop je drugi tip memorijskog elementa kod kojega je upis novog podatka kontrolisan ivicom odgovarajućeg kontrolnog signala. Kao i kod lečeva, postoji veliki broj različitih tipova flip flopova: SR flip flop, D flip flop, JK flip flop, T flip flop, itd. U praksi se najčešće koristi D flip flop. Interfejs D flip flopa prikazan je na slici 7.



Slika 7. Interfejs D flip flopa

D flip flop ima dva ulazna porta:

- *D* ulaz za podatke
- *clk* ulaz za dozvolu upisa

D flip flop u opštem slučaju ima i dva izlazna porta:

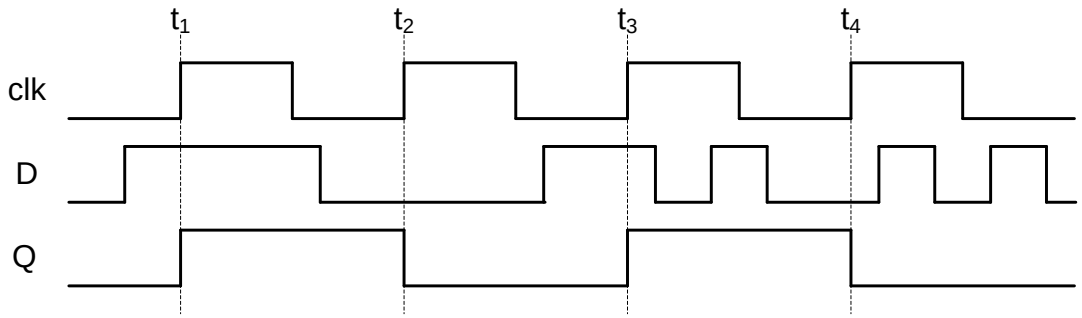
- *Q* izlaz za podatke
- $\overline{Q}$ invertovani izlaz za podatke

NAPOMENA: U praksi se najčešće koristi D flip flop koji ima samo jedan izlaz, izlaz Q.

Funkcionalnost D flip flopa kod koga se novi podataka upisuje ne rastuću ivicu *clk* signala može se prikazati sledećom tabelom.

clk	*D*	*Q*(t+1)	$\overline{Q}$(t+1)
0	x	*Q*(t)	$\overline{Q}$(t)
1	x	*Q*(t)	$\overline{Q}$(t)
↓	x	*Q*(t)	$\overline{Q}$(t)
↑	0	0	1
↑	1	1	0

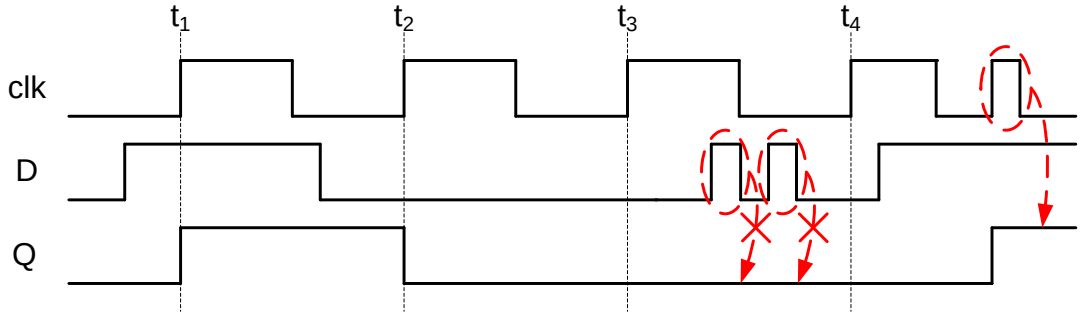
Kao što se može primetiti na osnovu prethodne tabele, sve dok se na *clk* ulazu ne pojavi rastuća iviva, D flip flop čuva prethodno upisanu vrednost i ignoriše vrednosti koje mu se trenutno nalaze na *D* ulazu. Tek kada se na *clk* ulazu pojavi rastuća ivica, u D flip flop se upisuje trenutna vrednost koja se nalazi na *D* ulazu. Vremenski dijagram rada D flip flopa, aktivnog na rastuću ivicu *clk* signala, prikazan je na slici 8.



Slika 8. Vremenski dijagram rada D flip flopa aktivnog na rastuću ivicu *clk* signala

Nailaskom rastuće ivice *clk* signala, trenutna vrednost *D* ulaza upisuje se u flip flop i pojavljuje se na *Q* izlazu (to je vrednost 1 u trenutku t_1, vrednost 0 u trenutku t_2, itd.). Sve promene na *D* ulazu koje se dešavaju između dve rastuće ivice *clk* signala se ignorišu, kao što je slučaj sa promenama *D* ulaza koje se pojavljuju nakon nailaska rastućih ivica u trenucima t_3 i t_4.

Za razliku od leča, flip flop zahteva da samo *clk* signal bude bez gličeva. Gličevi koji se pojavljuju na *D* ulazu će biti ignorisani od strane flip flopa. Ova situacija je ilustrovana na slici 9.



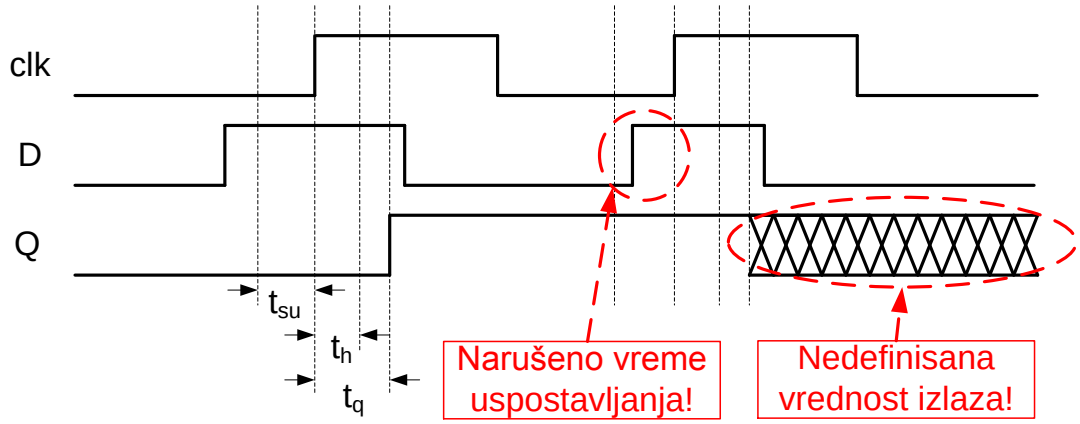
Slika 9. Vremenski dijagram rada D flip flopova u slučaju prisustva gličeva na *clk* i *D* ulazima

Pojava gliča na *D* ulazu u trenutku kada na *clk* signalu nema rastuće ivice, kao što je ilustrovano na slici 9 u ciklusu upisa koji počinje u trenutku t_3, biva „ignorisana“ od strane D flip flopa i ne prenosi se na *Q* izlaz. Međutim, pojava gliča na *clk* signalu dovodi do upisa pogrešne vrednosti u flip flop, što se vidi na slici 9 u ciklusu upisa koji počinje u trenutku t_4.

Ukoliko se u sistemu koriste flip flopovi, da bi se obezbedio ispravan i pouzdan rad sistema, mora se garantovati da će svi signali povezani na *clk* ulazne portove biti bez gličeva. Signali koji se povezuju na ulaze podataka flip flopova mogu sadržati gličeve, jedino se oni ne smeju pojavljivati u trenucima nailaska aktivnih ivica *clk* signala. U praksi se teži da se što više smanji broj različitih *clk* signala. Ono čemu se teži je da se u sistemima koristi samo jedan *clk* signal, koji je povezan na *clk* ulaze svih flip flopova u sistemu. U ovom slučaju se mora obezbediti da samo taj jedan, globani sinhronizacioni signal, nema gličeve, što je znatno lakši zadatak od slučaja kada se koriste lečevi.

Obzirom da je rad flip flopa „sinhronizovan“ sa signalom koji se dovodi na *clk* ulazni port flip flopa, sistemi koji kao memorijske elemente koriste flip flopove nazivaju se i *sinhroni sekvencijalni sistemi*.

Kao i u slučaju lečeva, i prilikom rada sa flip flopovima moraju se poštovati odgovarajući vremenski zahtevi kako bi se obezbedio pravilan rad flip flopa. Minimalni vremenski interval tokom kojega signal na *D* ulazu u flip flop mora biti stabilan pre nailaska opadajuće aktivne ivice (rastuće ili opadajuće) *clk* signala naziva se ***vreme uspostavljanja* (*setup time*), t_{su}**. Minimalni vremenski interval tokom kojega signal na D ulazu u flip flop mora biti stabilan nakon nailaska aktivne ivice (rastuće ili opadajuće) clk signala naziva se **vreme držanja (hold time), t_h**. Poslednji vremenski parametera koji opisuje rad flip flopa, jeste **kašnjenje na izlazu *Q*, t_{cq}**, koje se definiše kao maksimalni vremenski interval koji mora da protekne od nailaska aktivne ivice *clk* signala nakon kojega *Q* izlaz dobija novu stabilnu vrednost. Slika 10 ilustruje vremenske karakteristike D flip flopa.



Slika 10. Vremenske karakteristike D flip flopa

VHDL modeli D flip flopa

Kao što se može videti sa slike 7, D flip flop ima sledeće portove:

- 1-bitni ulazni port za podatke, *D*
- 1-bitni ulazni port za kontrolu upisa, *clk*
- 1-bitni izlazni port za podatke, *Q*

Odogovarajuća *entity* deklaracija D flip flopa sa dozvolom upisa ima sledeći izgled

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

entity 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 entity d_ff;
```

Nakon što smo opisali interfejs D flip flopa, potrebno je modelovati njegovu funkcionalnost. Za ovo je potrebno napisati odgovarajuće arhitekturno telo.

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

Bihevijani modeli

Varijanta 1: Model D flip flopa, aktivnog na rastućoj ivici clk signala, baziran na korišćenju *process* i *if* naredbe

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

NAPOMENA: Prikazani VHDL model D flip flopa koristi *if* naredbu u okviru procesa za utvrđivanje trenutka nailaska rastuće ivice na *clk* ulaznom portu. Kada se detektuje rastuća ivica, trenutna vrednost *D* ulaznog porta se prosleđuje na izlaz *Q*. U svim ostalim slučajevima ne dolazi do promene vrednosti na *Q* izlazu, što možemo tumačiti kao čuvanje „stare" vrednosti u flip flopu, koja je bila upisana u njega u nekom prethodnom trenutku. Uslov koji se proverava u okviru *if* naredbe sastoji se i dva dela. Prvi deo proverava da li se u tekućem simulacionom trenutku desio događaj na ulaznom portu *clk*, odnosno da li je u tekućem simulacionom trenutku port *clk* promenio svoju vrednost. Drugi deo uslova proverava da li je tekuća vrednost ulaznog porta *clk* jednaka 1. Tek ako su oba ova uslova zadovoljena ispunjen je uslov *if* naredbe i pristupa se izvršavanju naredbe dodele signalu *q*. Ako malo pažljivije analiziramo situaciju koja može dovesti do ispunjenja *if* uslova možemo zaključiti da on može da bude ispunjen samo ako se u tekućem simulacionom trenutku na ulaznom portu *clk* pojavila rastuća ivica. U ovom slučaju je prethodna vrednost porta *clk* morala biti jednaka 0, pa se u tekućem simulacionom trenutku pojavljuje događaj na portu *clk*, a tekuća vrednost porta *clk* je jednaka 1, tako da je zadovoljen i drugi deo uslova unutar *if* naredbe. Ovakav način modelovanja testa za detekciju pojave rastuće ivice na odgovarajućem signalu predstavlja standardni način modelovanja i često se sreće u VHDL modelima.

Varijanta 2: Model D flip flopa, aktivnog na rastućoj ivici clk signala, baziran na korišćenju *process* i *if* naredbe

```
architecture beh2 of d_ff is
begin
        dff: process (clk) is
        begin
                if (rising_edge(clk)) then
                        q <= d;
                end if;
        end process;
end architecture beh2;
```

NAPOMENA: U ovom VHDL modelu prikazan je drugi način zapisivanja uslova za proveru postojanja rastuće ivice na portu *clk*, koji se bazira na korišćenju funkcije

rising_edge(s). Ova funkcija je definisana u okviru *std_logic_1164* paketa i vraća vrednost *true* ukoliko se na signalu *s* pojavila rastuća ivica. U protivnom funkcija vraća vrednost *false*. U slučaju kada *clk* port može da uzima samo vrednosti 0 ili 1 ova funkcija ima isto ponašanje kao i složeni uslov clk'**event and** clk = '1'.

Varijanta 3: Model D flip flopa, aktivnog na opadajućoj ivici clk signala, baziran na korišćenju *process* i *if* naredbe

```
architecture beh3 of d_ff is
begin
        dff: process (clk) is
        begin
                if (clk'event and clk = '0') then
                        q <= d;
                end if;
        end process;
end architecture beh3;
```

NAPOMENA: Iako se u praksi uglavnom sreću flip flopovi koji su osetljivi na rastuću ivicu clk signala, postoje i flip flopovi koji su osetljivi na opadajuću ivicu *clk* signala. VHDL model D flip flopa osetljivog na opadajuću ivicu *clk* signala prikazan je gore. U ovom modelu koristi se složeni uslov, slično modelu iz varijante 1, sa jednom malom ali suštinskom promenom. Uslov se opet sastoji iz dva dela: prvog koji proverava da li se na portu *clk* desio događaj u tekućem simulacionom trenutku i drugog u kojem se proverava da li je tekuća vrednost *clk* porta jednaka 0. Ovaj test ekvivalentat je pitanju „da li se na clk portu pojavila opadajuća ivica".

Varijanta 4: Model D flip flopa, aktivnog na opadajućoj ivici clk signala, baziran na korišćenju *process* i *if* naredbe

```
architecture beh4 of d_ff is
begin
        dff: process (clk) is
        begin
                if (falling_edge(clk)) then
                        q <= d;
                end if;
        end process;
end architecture beh4;
```

NAPOMENA: U okviru *std_logic_1164* paketa definisana je i funkcija *falling_edge(s)*. Ova funkcija vraća vrednost *true* ako se na signalu *s* pojavila opadajuća ivica, u protivnom vraća vrednost *false*. Ova funkcija se može iskoristiti prilikom pisanja VHDL modela flip flopa osetljivog na opadajuću ivicu *clk* signala, kao što je pokazano u modelu iznad.

Kombinovanjem entity deklaracije i odgovarajućeg arhitekturnog tela dobijamo kompletan model D flip flopa. Na primer, dole je prikazan model D flip flopa, osetljivog na rastuću ivicu clk signala, koji se bazira na funkcionalnom modelu koji koristi *process* naredbu.

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

entity 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 entity d_ff;

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

Verifikaciono okruženje

Primer jednog mogućeg testbenča koji se može iskoristiti za verifikaciju D flip flopa, osetljivog na rastuću ivicu *clk* signala, prikazan je u nastavku.

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

entity d_ff_tb is
end entity d_ff_tb;

architecture beh of d_ff_tb is
   -- Deklaracija komponente VHDL modula koji se verifikuje (DUV)
   -- U ovom slučaju je to D flip flop
   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;

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

begin
        -- Komponenta koja se verifikuje
```

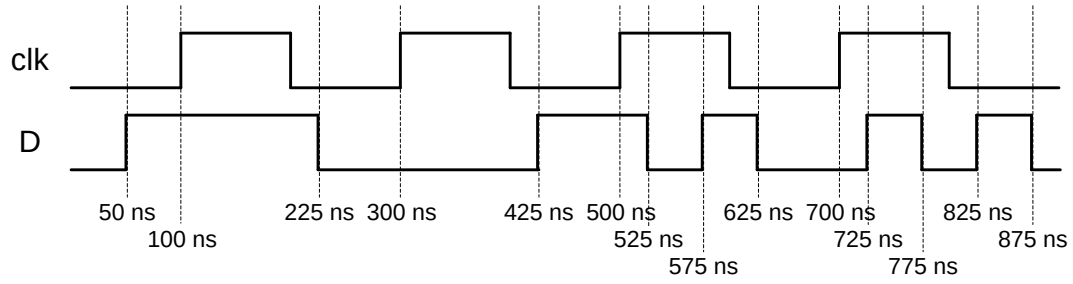
```
        duv: d_ff
           port map (
              clk => clk_s,
              d => d_s,
              q => q_s);

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

        -- Stimulus generator koji generise potrebne vrednosti na
        -- D ulaznom portu DUV-a na osnovu kojih ce biti moguce
        -- proveriti da li DUV implementira potrebnu funkcionalnost
        stim_gen: process
        begin
                d_s <= '0', '1' after 50 ns, '0' after 225 ns,
                                '1' after 425 ns, '0' after 525 ns,
                                '1' after 575 ns, '0' after 625 ns,
                                '1' after 725 ns, '0' after 775 ns,
                                '1' after 825 ns, '0' after 875 ns;

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

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

Clk_gen proces generiše periodičnu povorku pravougaonih impulsa na signalu *clk_s*. Perioda generisanog signala iznosi 200 ns. Svaki put kada se proces aktivira on planira dve transakcije na signalu *clk_s*, postavljajući ga na vrednost 0 u tekućem simulacionom trenutku, a zatim na vrednost 1 nakon 100 ns. Nakon izvršavanja naredbe dodele vrednosti signalu, proces se deaktivira u trajanju od 200 ns pomoću *wait* naredbe. Nakon što protekne 200 ns, *clk_gen* proces se opet aktivira i planira naredne dve transakcije na *clk_s* signalu (prva koja *clk_s* postavlja na 0 i druga koja *clk_s* postavlja na 1), pre nego što se opet deaktivira u trajanju od 200 ns. Ovaj postupak se zatim periodično ponavlja sve do završetka simulacije, a za rezultat ima generisani periodični signal na signalu *clk_s*.

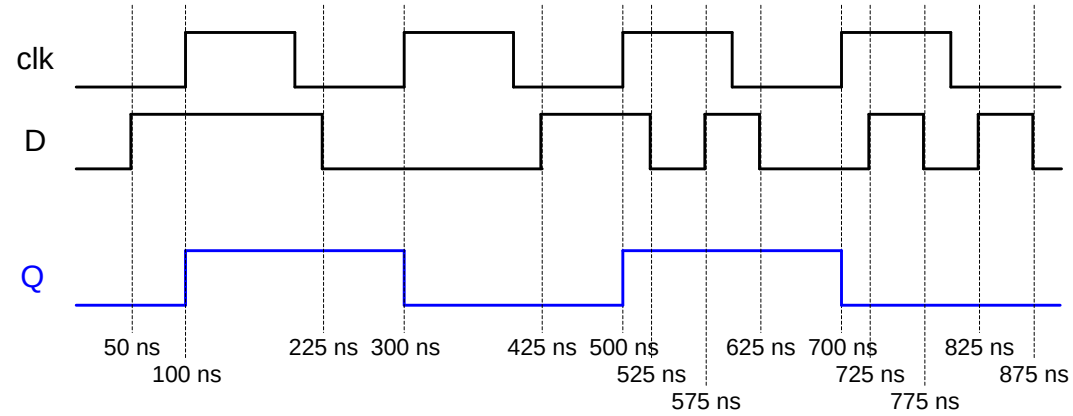
Stim_gen proces generiše vremenske oblike za signal *d_s* koristeći jednu naredbu sekvencijalne dodele vrednosti signalu.

Talasni oblici koji će biti generisani pomoću ove dve process naredbe prikazani su na slici 11.



Slika 11. Generisani talasni oblici na dva ulazna signala D flip flopa

Nakon što se izvrši simulacija kombinovanog rada testbenča i modela D flip flopa talasni oblik izlaznog signala podataka *q_s* trebalo bi da izgleda kao na slici 12.



Slika 12. Generisani talasni oblici na dva ulazna signala D flip flopa 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 D flip flopa koji su aktivni na rastuću ivicu *clk* porta.

Zadaci za vežbu

Zadatak 1:

Napisati bihevijalni VHDL model SR flip flopa koji će biti osetljiv na opadajuću ivicu *clk* ulaznog porta. Za razvijeni VHDL model napisati testbenč koji se može koristiti za funkcionalnu verifikaciju.

Zadatak 2:

Napisati bihevijalni VHDL model JK flip flopa koji će biti osetljiv na rastuću ivicu *clk* ulaznog porta. Za razvijeni VHDL model napisati testbenč koji se može koristiti za funkcionalnu verifikaciju.

Zadatak 3:

Napisati VHDL model D flip flopa osetljivog na rastuću ivicu *clk* porta koji će modelovati konačnu brzinu prostiranja signala kroz flip flop. Vremenske karakteristike D flip flopa koji je potrebno modelovati imaju sledeće vrednosti:

- vreme uspostavljanja, t_{su} = 2 ns
- vreme držanja, t_h = 1.5 ns
- kašnjenje na izlazu, t_{cq} = 3 ns

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

Zadatak 4:

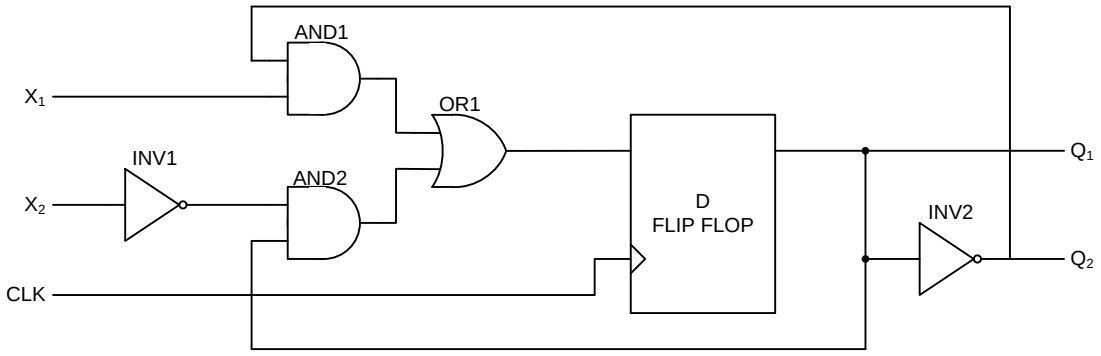
Proširiti VHDL model D flip flopa iz zadatka 3 tako da uključi:

- vremenske provere zadovoljenosti vremena uspostavljanja i držanja na *D* ulazu flip flopa 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 5:

Napisati strukturni VHDL model logičke mreže prikazane na slici 13. Za razvijeni VHDL model napisati odgovarajući testbenč pomoću kojega će biti moguće izvršiti funkcionalnu verifikaciju razvijenog modela i utvrditi koji tip flip flopa je realizovan pomoću mreže sa slike 13.



Slika 13. Logička mreža koja realizuje nepoznati tip flip flopa

Modelovanje sinhronih i asinhronih ulaza u sekvencijalnim mrežama

Ranije smo videli da u radu flip flopa centralno mesto zauzima jedan, sinhronizacioni, signal koji se dovodi na *clk* ulazni port svakog flip flopa. Promene na ostalim ulaznim portovima se ignorišu sve do trenutka nailaska „sinhronizacionog događaja“ na *clk* ulaznom portu, rastuće ili opadajuće ivice. Tek u tom trenutku ažurira se stanje flip flopa i eventualno dešavaju promene unutar sistema. Ulazni portovi koji se analiziraju samo u trenutku nailaska „sinhronizacionog događaja“ nazivaju se *sinhroni ulazni portovi*. Međutim, postoji i druga klasa ulaznih portova, koji se analiziraju nezavisno od bilo kakvog sinhronišućeg signala, koji se nazivaju *asinhroni ulazni portovi.*

U opštem slučaju, unutar jednog sinhronog sekvencijalnog sistema mogu postojati i sinhroni i asinhroni ulazni portovi. U nastavku će biti prikazan pravilan način modelovanja ovakvih portova pomoću unutar VHDL modela.

Na primer, ukoliko posmatramo D flip flop, kao najjednostavnijeg predstavnika sinhronog sekvencijalnog sistema, *D* ulazni port pripada klasi *sinhronih ulaznih portova*, ili kraće *sinhronih ulaza*. Vrednost *D* ulaznog porta se analizira i koristi jedino u trenucima nailaska „sinhronizacionog događaja“, rastuće ili opadajuće ivice signala povezanog na *clk* ulazni port D flip flopa. U svim ostalim trenucima vrednost *D* ulaznog porta se ignoriše i bilo kakve promene na njemu nemaju nikakvog uticaja na stanje i ponašanje D flip flopa. Upravo is tog razloga kažemo da je *D* ulazni port *„sinhronizovan“* sa *clk* ulaznim portom, odnosno sa signalom koji je povezan na *clk* ulazni port.

Ova situacija jasno je vidljiva i unutar VHDL modela D flip flopa, prikazanog ranije, ponovo prikazanog u nastavku.

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

entity 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 entity d_ff;


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

Ako pažljivije analiziramo VHDL model možemo videti da se D ulazni port koristi jedino u sekvencijalnoj naredbi dodele vrednosti signalu (u našem slučaju izlaznom portu *q*), koja se pak nalazi unutar *if* naredbe u kojoj se proverava da li je nastupio „sinhronizacioni događaj“, u našem slučaju nailazak rastuće ivice na *clk* ulaznom portu. Sve dok se ne desi ovaj događaj, vrednost *D* ulaznog porta se ne koristi unutar modela flip flopa. Odavde možemo izvesti prvo pravilo prilikom modelovanja

sinhronih ulaza: **sihroni ulazni portovi smeju se koristiti samo u delovima modela koji se nalaze unutar *if* naredbe koja testira nailazak „sinhronizacionog događaja“ na *clk* ulazu**!

Dalje, vidimo da se *D* ulazni port ne nalazi unutar liste osetljivosti procesa koji modeluje ponašanje D flip flopa. Na prvi pogled ovo se može činiti kao greška, ali zapravo nije. Obzirom da se vrednost sinhronog ulaza koristi samo nakon što se identifikuje postojanje „sinhronizacionog događaja“, proces ne treba da bude osetljiv na ovaj tip ulaza, jer sve ostale promene na sinhronom ulazu će ionako biti ignorisane od strane sistema koji modelujemo. Ukoliko bi smo sinhroni ulaz uvrstili u listu osetljivosti smo bi smo usporili izvršavanje modela. U ovom slučaju model bi se aktivirao svaki put kada se desi događaj na sinhronom ulazu. Međutim, ukoliko se u tom trenutku nije desio i „sinhronizacioni događaj“, događaj na sinhronom ulazu je irelevantan. Na primer, ukoliko bi proces unutar našeg modela D flip flopa uključivao i *D* ulazni port u listi osetljivost, on bi se aktivirao svaki put kada bi se desio događaj na *D* ulazu. Nakon aktiviranja, proces bi počeo sa izvršavanjem naredbi iz svog tela. Prva naredba na koju bi naišao bila bi *if* naredba koja proverava da li se u tekućem simulacionom trenutu pojavila rastuća ivica na *clk* ulaznom portu. Ukoliko se rastuća ivica nije pojavila uslov iz *if* naredbe ne bi bio zadovoljen tako da bi se preskočilo izvršavanje naredbe dodele nove vrednosti izlaznom portu *q*, proces bi došao do svog kraja i deaktivirao se. U ovom slučaju se proces potpuno bespotrebno aktivirao, obzirom da nismo ažurirali ni jedan od signala/portova entiteta čija se funkcionalnost modeluje pomoću procesa. Jedina situacija koja može dovesti do potencijalne promene vrednosti izlaza *q* je trenutak kada se na *clk* ulazu pojavi rastuća ivica, odnosno da se desi događaj na *clk* ulazu. Zbog toga se *clk* ulaz mora naći u listi osetljivosti, ali je *D* ulaz suvišan.

Nakon ovog primera može izvesti drugo pravilo u modelovanju sinhronih ulaza: **sinhroni ulazni portovi ne smeju se naći unutar liste osetljivosti procesa koji modeluju ponašanje sistema**!

Koja pravila važe prilikom modelavanja druge klase ulaznih portova sinhronih sekvencijalnih sistema, asinhronih ulaznih portova?

Obzirom da se vrednost asinhronih ulaza može koristiti u bilo kom trenutku unutar modela, nezavisno od stanja *clk* ulaznog porta, važe sledeća dva pravila:

1. **Asinhroni ulazi se koriste u delovima VHDL modela koji se nalaze izvan *if* naredbe koja testira nailazak „sinhronizacionog događaja“ na *clk* ulazu.**
2. **Asinhroni ulazi moraju biti navedeni u listi osetljivosti procesa koji modeluje ponašanje sinhronog sekvencijalnog sistema.**

U sledećem VHDL „kosturu“ jasno su označene sekcije u kojima se potrebno koristiti sinhrone, a u kojim asinhrone ulazne portove.

```
model: process (clk,  ... samo asinhroni ulazi ... ) is
        begin
                ...
                Deo u kojem se koriste asinhroni ulazi
                ...
                if (clk'event and clk = '1') then
                        ...
                        Deo u kojem se koriste sinhroni ulazi
                        ...
                end if;
                ...
                Deo u kojem se koriste asinhroni ulazi
                ...
        end process;
```

Ilustrujmo korišćenje ovih pravila kroz par primera. Kao što je poznato, obzirom da nije moguće unapred predvideti u kom stanju će se naći flip flop nakon dovođenja napona napajanja, pre početka rada potrebno ga je inicijalizovati na neku poznatu početnu vrednost. U ovu svrhu flip flopovi obično imaju još neke dodatne ulaze koji se koriste u ovu svrhu. Standardni ulazi flip flopa koji se koriste za njegovu inicijalizaciju su:

1. **sinhroni *reset* ulaz** – ukoliko je ovaj ulaz na 1 prilikom nailaska „sinhronizacionog događaja“ na *clk* ulazu (rastuće ili opadajuće ivice, u zavisnosti od tipa flip flopa) stanje flip flopa će biti postavljeno na vrednost 0.
2. **sinhroni *set* ulaz** – ukoliko je ovaj ulaz na 1 prilikom nailaska „sinhronizacionog događaja“ na *clk* ulazu (rastuće ili opadajuće ivice, u zavisnosti od tipa flip flopa) stanje flip flopa će biti postavljeno na vrednost 1.
3. **asinhroni *clear* ulaz** – ukoliko je ovaj ulaz na 1, bez obzira da li postoji „sinhronizacioni događaj“ na *clk* ulazu (rastuća ili opadajuće ivica, u zavisnosti od tipa flip flopa) stanje flip flopa će biti postavljeno na vrednost 0.
4. **asinhroni *preset* ulaz** – ukoliko je ovaj ulaz na 1, bez obzira da li postoji „sinhronizacioni događaj“ na *clk* ulazu (rastuća ili opadajuće ivica, u zavisnosti od tipa flip flopa) stanje flip flopa će biti postavljeno na vrednost 1.

VHDL model D flip flopa sa sinhronim reset ulazom

VHDL model D flip flopa sa sinhronim *reset* ulazom prikazan je u nastavku.

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

entity d_ff_r is
        port (clk:      in std_logic;   -- ulazni port dozvole upisa
              reset:    in std_logic;   -- sinhroni reset ulaz
              d:        in std_logic;   -- ulazni port podataka
              q:        out std_logic);-- izlazni port podataka 1
```

```
end entity d_ff_r;

architecture beh1 of d_ff_r is
begin
        dff: process (clk) is
        begin
                if (clk'event and clk = '1') then
                        if (reset = '1') then
                                q <= '0';
                        else
                                q <= d;
                        end if;
                end if;
        end process;
end architecture beh1;
```

Može se primetiti da u okviru *entity* deklaracije postoji dodatni port, *reset*, koji služi za sinhronu inicijalizaciju D flip flopa. Obzirom da je reč o sinhronom ulazu, *reset* ulazni port nije naveden u listi osetljivosti *dff* procesa. Takođe, analizom tela *dff* procesa može se primetiti da se *reset* port koristi samo unutar *if* naredbe koja testira prisustvo rastuće ivice *clk* ulaza. Na osnovu ove analize možemo zaključiti da su ispoštovana oba pravila korišćenja sinhronih ulaznih portova unutar VHDL modela sinhronih sekvencijalnih mreža. Sam način korišćenja *reset* ulaznog porta govori nam da je reč o reset ulazu, jer se sadržaj flip flopa postavlja na vrednost 0 svaki put kada je *reset* ulaz aktivan, a detektovano je prisustvo rastuće ivice na *clk* ulazu.

VHDL model D flip flopa sa asinhronim preset ulazom

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

entity d_ff_p is
        port (clk:      in std_logic;   -- ulazni port dozvole upisa
              preset:   in std_logic;   -- asinhroni preset ulaz
              d:        in std_logic;   -- ulazni port podataka
              q:        out std_logic);-- izlazni port podataka 1
end entity d_ff_p;

architecture beh1 of d_ff_p is
begin
        dff: process (clk, preset) is
        begin
                if (preset = '1') then
                        q <= '1';
                else
                        if (clk'event and clk = '1') then
                                q <= d;
                        end if;
                end if;
        end process;
```

```
end architecture beh1;
```

Slično kao i u primeru VHDL modela D flip flopa sa sinhronim reset ulazom, *entity* deklaracija VHDL modela D flip flopa sa asinhronim preset ulazom sadrži dodatni port, *preset*, koji služi za asinhronu inicijalizaciju D flip flopa. Obzirom da je reč o asinhronom ulazu, *preset* ulazni port je naveden u listi osetljivosti *dff* procesa. Takođe, analizom tela *dff* procesa može se primetiti da se *preset* port koristi samo izvan *if* naredbe koja testira prisustvo rastuće ivice *clk* ulaza. Na osnovu ove analize možemo zaključiti da su ispoštovana oba pravila korišćenja asinhronih ulaznih portova unutar VHDL modela sinhronih sekvencijalnih mreža. Sam način korišćenja *preset* ulaznog porta govori nam da je reč o asinhronom preset ulazu, jer se sadržaj flip flopa postavlja na vrednost 1 svaki put kada je *reset* ulaz aktivan, bez obzira da li je detektovano prisustvo rastuće ivice na *clk* ulazu.

Verifikaciono okruženje

Primer jednog mogućeg testbenča koji se može iskoristiti za verifikaciju D flip flopa osetljivog na rastuću ivicu *clk* signala, sa sinhronim reset ulazom, prikazan je u nastavku.

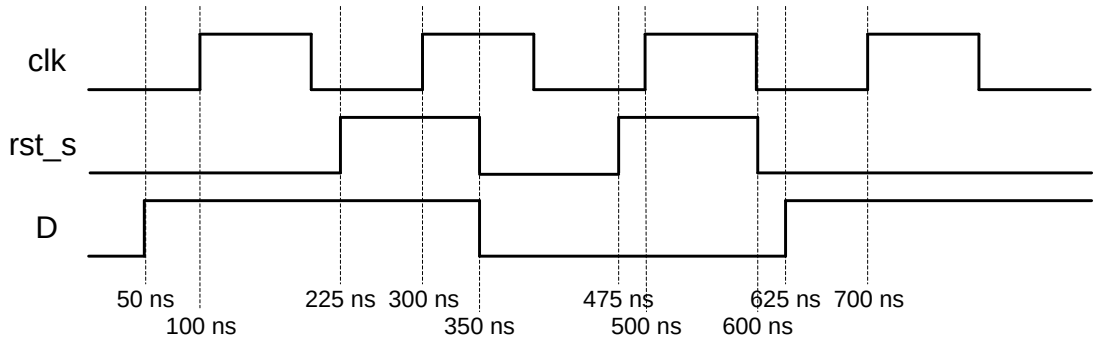
```
library ieee;
use ieee.std_logic_1164.all;

entity d_ff_r_tb is
end entity d_ff_r_tb;


architecture beh of d_ff_r_tb is
   -- Deklaracija komponente VHDL modula koji se verifikuje (DUV)
   -- U ovom slučaju je to D flip flop
   component d_ff_r is
         port (clk:    in std_logic;    -- ulazni port dozvole upisa
               reset:  in std_logic;    -- sinhroni reset ulaz
               d:      in std_logic;    -- ulazni port podataka
               q:      out std_logic);-- izlazni port podataka 1
   end component d_ff_r;

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

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

        -- Klok generator koji generise periodicni clk_s signal koji
```

```
        -- ce se koristiti za aktiviranje D flip flopa
        clk_gen: process
        begin
                clk_s <= '0', '1' after 100 ns;
                wait for 200 ns;
        end process;

        -- Stimulus generator koji generise potrebne vrednosti na
        -- reset i D ulaznim portovima DUV-a na osnovu kojih ce biti
        -- moguce proveriti da li DUV implementira potrebnu
        -- funkcionalnost
        stim_gen: process
        begin
                rst_s <= '0', '1' after 225 ns, '0' after 350 ns,
                                '1' after 475 ns, '0' after 600 ns;

                d_s <= '0', '1' after 50 ns, '0' after 350 ns,
                                '1' after 625 ns;

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

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

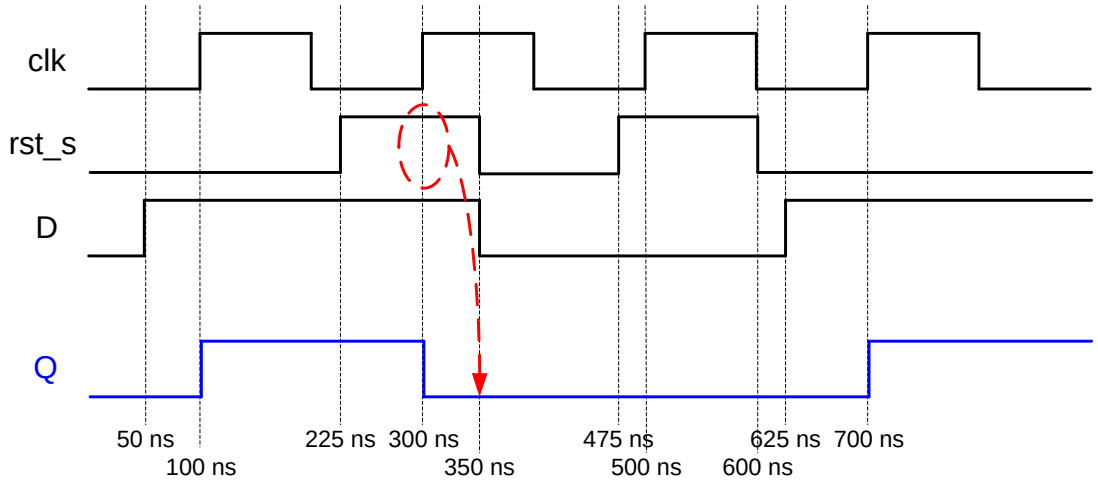
Clk_gen proces generiše periodičnu povorku pravougaonih impulsa na signalu *clk_s*. *Stim_gen* proces generiše vremenske oblike za signale *rst_s* i *d_s* koristeći dve naredbe sekvencijalne dodele vrednosti signalu.

Talasni oblici koji će biti generisani pomoću ove dve process naredbe prikazani su na slici 14.



Slika 14. Generisani talasni oblici na tri ulazna porta D flip flopa

Nakon što se izvrši simulacija kombinovanog rada testbenča i modela D flip flopa sa sinhronim resetom, talasni oblik izlaznog signala podataka *q_s* trebalo bi da izgleda kao na slici 15.



Slika 15. Generisani talasni oblici na tri ulazna porta D flip flopa zajedno sa talasnim oblicima izlaznog signala podataka *q_s* koji je dobijen kao rezultat simulacije

Zadaci za vežbu

Zadatak 1:

Napisati testbenč koji se može iskoristiti za verifikaciju VHDL modela D flip flopa sa asinhronim preset ulazom, koji je prikazan ranije.

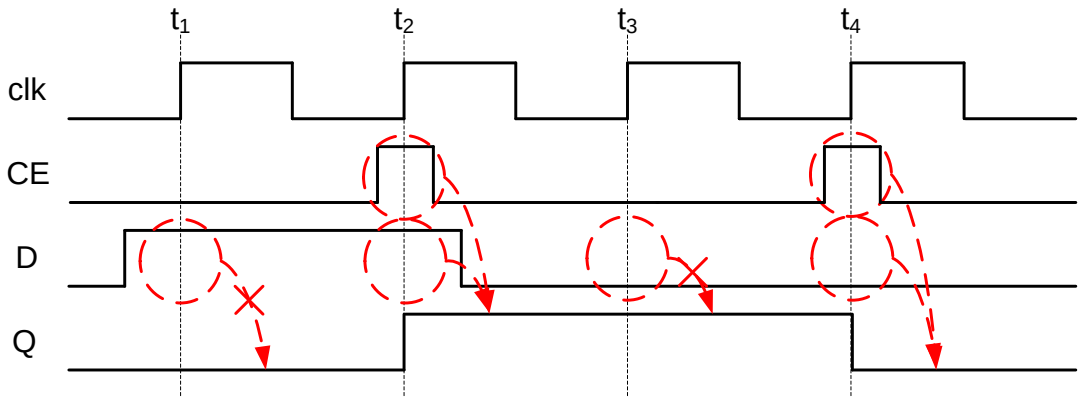
Zadatak 2:

Napisati VHDL modele i odgovarajuće testbenčeve za sledeće vrste D flip flopova:

a) D flip flop sa sinhronim *set* ulazom
b) D flip flop sa asinhronim *clear* ulazom
c) D flip flop sa asinhronim *clear* i *preset* ulazima
d) D flip flop sa sinhronim *set* i *reset* ulazima

Modelovanje *clock enable* ulaza u sekvencijalnim mrežama

Posebno značajan sinhroni ulazni port koji se često sreće prilikom rada sa sinhronim sekvencijalnim sistemima je *clock enable* (CE) ulazni port. CE port omogućava dizajneru da za svaki flip flop pojedinačno specificira na koju od rastućih (ili opadajućih) ivica *clk* porta da se aktivira. Ranije smo već rekli da je flip flop memorijski element koji je osetljiv na rastuću ili opadajuću ivicu *clk* ulaznog porta. Bez postojanja CE ulaza flip flop bi bio aktivan prilikom nailaska svake rastuće/opadajuće ivice na *clk* ulazu. Iako je ovo ponekad željeno ponašanje, u praksi se često javlja potreba za selekcijom rastućih/opadajućih ivica na koje flip flop treba da se aktivira. Ova mogućnost nam dozvoljava da projektujemo sisteme koji će koristiti jedan, zajednički, globalni sinhronizacioni signal, klok ili takt, ali će svaki od flip flopova unutar sistema moći da radi na različitim brzinama jer će biti aktiviran samo na odabrane rastuće/opadajuće ivice klok signala. Slika 16 ilustruje način korišćenja CE ulaznog porta za specifikaciju na kojim rastućim ivicama D flip flop treba da bude aktivan.



Slika 16. Selekcija rastućih ivica *clk* ulaznog porta na kojima će D flip flop biti aktivan korišćenjem CE ulaza

Sa slike 16 može se videti da je upis novih podataka u D flip flop sa *CE* ulazom kontrolisan ne samo pojavom rastuće ivice na *clk* ulazu već i istovremenim prisustvom visokog logičkom nivoa na *CE* ulazu. U periodi takta koja počinje u trenutku t_1 iako jeu trenutku nailaska rastuće ivice na *clk* ulazu vrednost *D* ulaza na 1, ona se neće upisati u D flip flop jer u tom trenutku *CE* ulaz ima vrednost 0. Slična situacija dešava se i u trećoj periodi takta, koja počinje u trenutku t_3, samo što se ovaj put u flip flop ne upisuje vrednost 0, koja bi inače trebala da se upiše. Upis u flip flop dozvoljen je samo na onim rastućim ivicama *clk* ulaza za koje je vrednost *CE* ulaza jednaka 1. Na slici 16 to je slučaj u drugoj i četvrtoj periodi takta.

VHDL model D flip flopa sa *clock enable* ulazom

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

entity d_ff_ce is
        port (clk:      in std_logic;    -- ulazni port dozvole upisa
              ce:       in std_logic;    -- clock enable ulaz
              d:        in std_logic;    -- ulazni port podataka
              q:        out std_logic);-- izlazni port podataka 1
end entity d_ff_ce;

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

Entity deklaracija VHDL modela D flip flopa sa *clock enable* ulazom sadrži dodatni port, *ce*, koji služi za selekciju na koju od rastućih ivica *clk* porta flip flop treba da se aktivira. Obzirom da je reč o sinhronom ulazu, *ce* ulazni port nije naveden u listi osetljivosti *dffce* procesa. Takođe, analizom tela *dffce* procesa može se primetiti da se *ce* ulazni port koristi unutar *if* naredbe koja testira prisustvo rastuće ivice *clk* ulaza. Kada se detektuje prisustvo rastuće ivice na *clk* ulazu, proverava se da li je *ce* ulaz jednak 1. Tek ako je i *ce* ulaz na visokom logičkom nivou vrši se upis novog sadržaja u D flip flop.

Verifikaciono okruženje

Primer jednog mogućeg testbenča koji se može iskoristiti za verifikaciju D flip flopa osetljivog na rastuću ivicu *clk* signala, sa *clock enable* ulazom, prikazan je u nastavku.

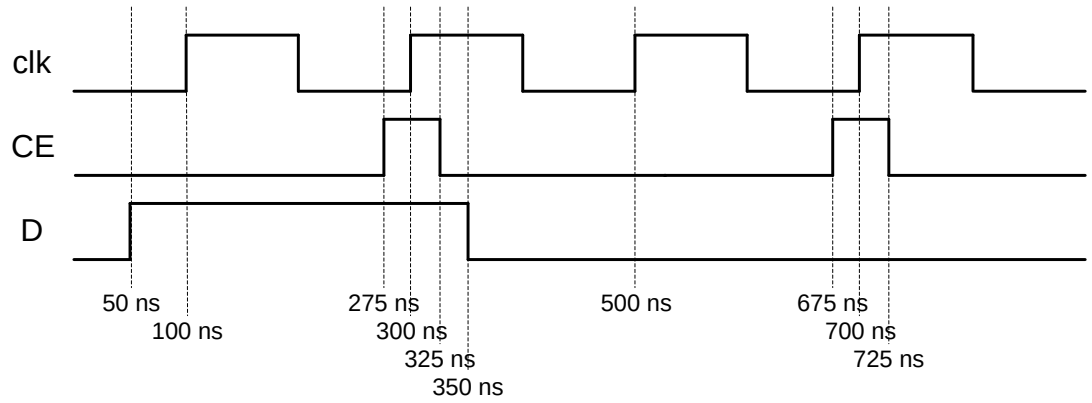
```
library ieee;
use ieee.std_logic_1164.all;

entity d_ff_ce_tb is
end entity d_ff_ce_tb;


architecture beh of d_ff_ce_tb is
   -- Deklaracija komponente VHDL modula koji se verifikuje (DUV)
   -- U ovom slučaju je to D flip flop
   component d_ff_ce is
         port (clk:    in std_logic;    -- ulazni port dozvole upisa
               ce:     in std_logic;    -- clock enable ulaz
               d:      in std_logic;    -- ulazni port podataka
               q:      out std_logic);-- izlazni port podataka 1
   end component d_ff_ce;

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

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

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

```
        -- Stimulus generator koji generise potrebne vrednosti na
        -- reset i D ulaznim portovima DUV-a na osnovu kojih ce biti
        -- moguce proveriti da li DUV implementira potrebnu
        -- funkcionalnost
        stim_gen: process
        begin
                ce_s <= '0', '1' after 275 ns, '0' after 325 ns,
                                '1' after 675 ns, '0' after 725 ns;

                d_s <= '0', '1' after 50 ns, '0' after 350 ns;

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

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

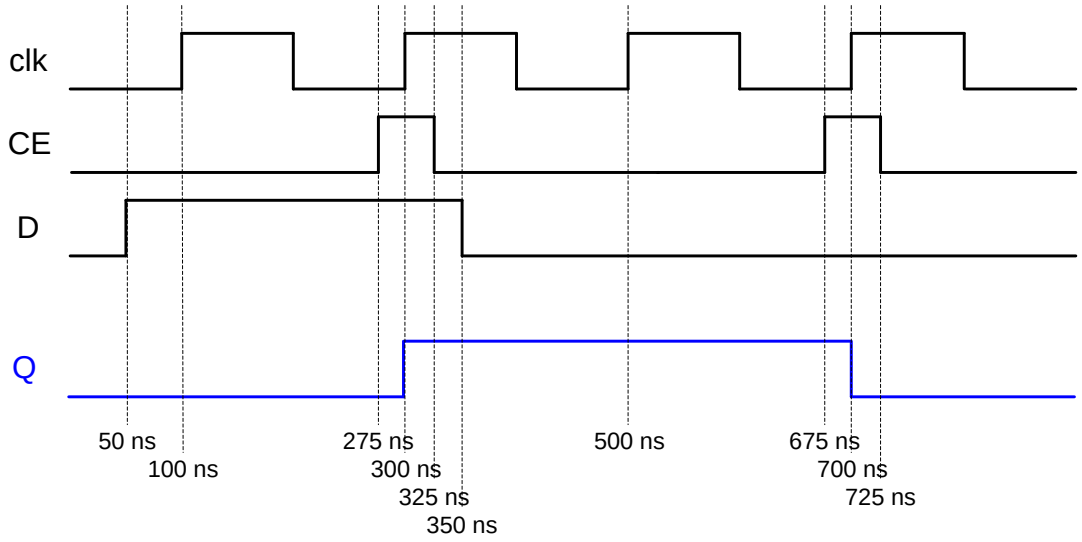
Clk_gen proces generiše periodičnu povorku pravougaonih impulsa na signalu *clk_s*. *Stim_gen* proces generiše vremenske oblike za signale *ce_s* i *d_s* koristeći dve naredbe sekvencijalne dodele vrednosti signalu.

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



Slika 17. Generisani talasni oblici na tri ulazna porta D flip flopa

Nakon što se izvrši simulacija kombinovanog rada testbenča i modela D flip flopa sa *clock enable* ulazom, talasni oblik izlaznog signala podataka *q_s* trebalo bi da izgleda kao na slici 18.



Slika 18. Generisani talasni oblici na tri ulazna porta D flip flopa zajedno sa talasnim oblicima izlaznog signala podataka *q_s* koji je dobijen kao rezultat simulacije

Zadaci za vežbu

Zadatak 1:

Napisati VHDL modele i odgovarajuće testbenčeve za sledeće vrste D flip flopova:

a) D flip flop sa *clock enable* ulazom i asinhronim *clear* i *preset* ulazima

b) D flip flop sa *clock enable* ulazom i sinhronim *reset* i *set* ulazima