Univerzitet u Novom Sadu
Fakultet tehničkih nauka
Odsek za energetiku, elektroniku i telekomunikacije
Katedra za elektroniku

Beleške sa predavanja iz predmeta:

UVOD U MIKRORAČUNARSKU ELEKTRONIKU

(VHDL)

Pripremio: prof. dr Veljko Malbaša
Novi Sad, 2007. godine

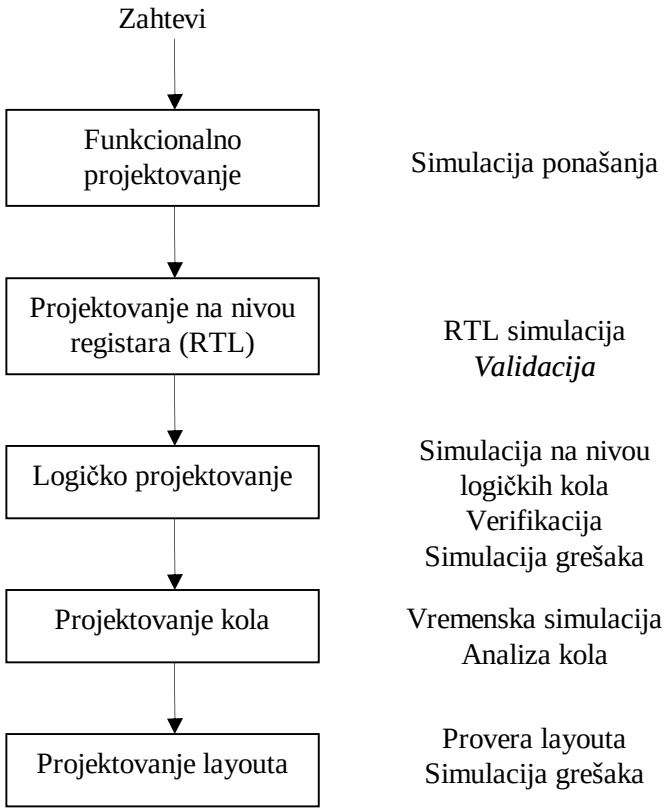
1. Uvod

Jezici za opis hardvera (*Hardware Description Languages - HDL)* koriste se za opis, modeliranje i projektovanje digitalnih sistema. HDL model može korisno da služi za verifikaciju i simulaciju digitalnog sistema.

VHDL (skraćenica od *Very High Speed Integrated CircuitsHDL)* posebna je verzija HDL-a koja se pojavila 1985. godine na inicijativu i uz finansiranje Ministarstva odbrane SAD. Profesionalna organizacija inženjera elektronike i elektrotehnike IEEE je 1987. godine objavila standard IEEE 1076-1987 i kasnije standard IEEE 1076-1993 za jezik VHDL. Prošle decenije napravljen je standard IEEE 1164 za softverske pakete koji sadrže tipove podataka, funkcije i procedure koji se koriste u modelima urađenim primenom jezika VHDL.

2.1 Projektovanje digitalnih sistema

Slika 1.1 prikazuje jedan pristup projektovanju digitalnih sistema koji počinje definisanjem zahteva koje sistem treba da ispuni, posle koga sledi niz projektantskih koraka koji se završavaju specifikacijom za direktnu implementaciju digitalnog sistema.



Slika 1.1: Postupak projektovanja digitalnih sistema

Specifikacija funkcionalnih zahteva obuhvata operacije koje sistem treba da izvršava, interfejs digitalnog sistema sa spoljnim jedinicama i ograničenja u pogledu brzine rada, cene, dimenzija i disipacije.

Na osnovu specifikacije zahteva radi se preliminarno funkcionalno projektovanje i simulacija ponašanja u kojoj se proverava funkcionalnost modela i zadovoljenje zadatih ograničenja.

Cilj projektovanja na nivou registara je da funkcionalni model prevede u strukturu koja se sastoji od komponenata kao što su registri, memorije, aritmetičko logička i upravljačka jedinica. Model na ovom nivou apstrakcije često se naziva RTL *(Register Transfer Logic)* model ditigalnog sistema. Jedan od važnih zadataka projektanta u ovom koraku je pokazivanje da je model na nivou registara ekvivalentan funkcionalnom modelu digitalnog sistema.

Daljim rafiniranjem RTL modela i razradom detalja dobija se digitalni sistem koji se sastoji iz logičkih kola. Naravno, u ovom koraku takođe treba pokazati da je dobijeni model ekvivalentan modelu na nivou registara. Na ovom nivou može da se radi simulacija modela sa ciljem da se pokaže da digitalnih sistem radi prema zadatoj početnoj specificikaciji.

Pošto VHDL model može da se koristi za implementaciju na programabilnim kolima tipa FPGA ili za neposrednu izradu specijalizovanih integrisanih kola *(Application Specific Integrated Circuit - ASIC)*, poslednja dva koraka zavise od izbora tehnologije. Na primer, ako je implementacija u FPGA, onda se generiše mikrokod koji se koristi za programiranje FPGA kola.

Značajna prednost primene VHDL-a je što razvijeni modeli mogu da se koriste za otkrivanje grešaka u projektovanju i za verifikaciju modela digitalnog sistema. Iskustvo pokazuje da je u postupku projektovanja veoma važno greške otkriti što je moguće ranije u ciklusu projektovanja. Greška koja se otkrije u ranoj fazi projektovanja značajno je lakša za otklanjanje i manje skupa od greške koja se otkrije u kasnijoj fazi projektovanja. Pokazuje se da VHDL model omogućava rano otkrivanje i ispravljanje grešaka što sa druge strane skraćuje vreme projektovanja digitalnog sistema.

Pošto se u postupku projektovanja obavlja rafinacija jednog modela u drugi, zadatak projektanta je da pokaže da je jedan model digitalnog sistema ekvivalentan prethodnom modelu na višem nivou apstrakcije. Naravno, digitalni sistem predstavljen je modelima na različitim nivoima apstrakcije i cilj je pokazati da su modeli međusobno ekvivalentni i naravno da zadovoljavaju zadatu početnu specifikaciju. Pored verifikacije funkcionalnosti, VHDL model može da se koristi za verifikaciju zadatih ograničenja, na primer u pogledu brzine rada digitalnog sistema.

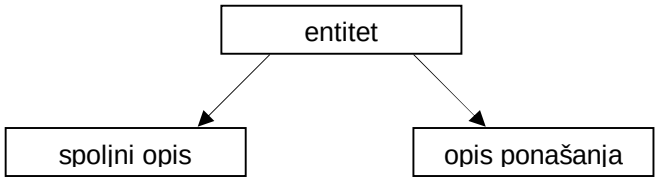
Konačno, treba napomenuti da VHDL podržava ponovno korišćenje prethodno razvijenih i verifikovanih modela što značajno doprinosi smanjenju vremena i cene razvoja digitalnih sistema.

2.2 Osnovni pojmovi

Osnovni programski objekti u VHDL modelima su signali koji se pojavljuju na pristupnim tačkama digitalnog sistema ili na internim provodnicima koji spajaju komponente digitalnih sistema. U VHDL modelu mora biti deklarisan tip signala. Tip signala može biti, na primer bit ili niz bitova.

Za razliku od promenljivih u programskim jezicima koji mogu da imaju u svakom trenutku samo jednu vrednost, signali pored vrednosti imaju i vreme dodele te vrednosti signalu, obeleženo parom *(vrednost, vreme).* Signal zadržava dodeljenu vrednost sve dok mu se ne dodeli neka druga vrednost u nekom kasnijem trenutku. Važna osobina signala je da mogu imati nekoliko parova *(vrednost, vreme)* koji predstavljaju sekvencu vrednosti koju će signal dobiti u navedenim vremenskim trenucima. Sekvenca vrednosti dodeljenih signalu u toku vremena je vremenski dijagram signala.

U postupku modeliranja primenom VHDL-a, digitalni sistem obično se dekomponuje na jednostavnije komponente koje se nazivaju entiteti. Izbor entiteta zavisi od primene i izbora projektanta. Na primer, entitet može bit multiplekser, integrisano kolo ili komponenta kao što je tranzistor. Opis entitea u VHDL-u sastoji se iz dva dela: opis pristupnih tačaka (spoljnjih priključaka) i opis ponašanja, Slika 1.2.

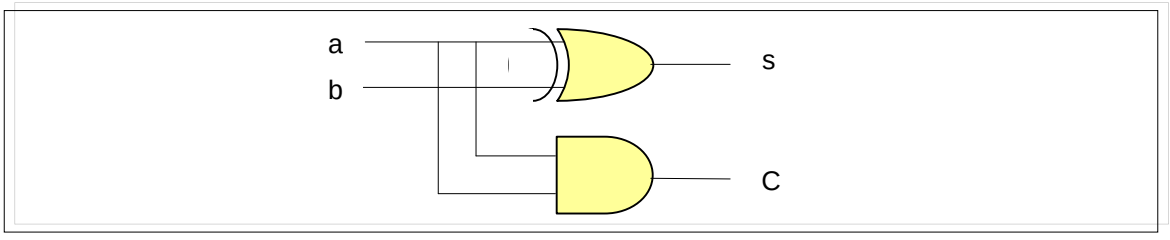


Slika 1.2: Opis entiteta sastoji se iz spoljnjeg opisa i opisa ponašanja

2.3 Spoljni opis entiteta

Pristupne tačke entiteta nazivaju se *portovi*. Preko portova se prenose signali koji mogu biti ulazni, izglazni ili bidirekcioni (ulazno-izlazni). Spoljni opis entiteta obuhvata deklaraciju signala koji se prenose preko portova.

Slika 1.3 prikazuje logičku šemu polusabirača sa ulaznim pristupnim tačkama preko kojih se prenose ulazni signali *a* i *b*, i izlaznim pristupnim tačkama za izlazne signala *s* (suma) i *C* (prenos).



Slika 1.3: Logična šema polusabirača

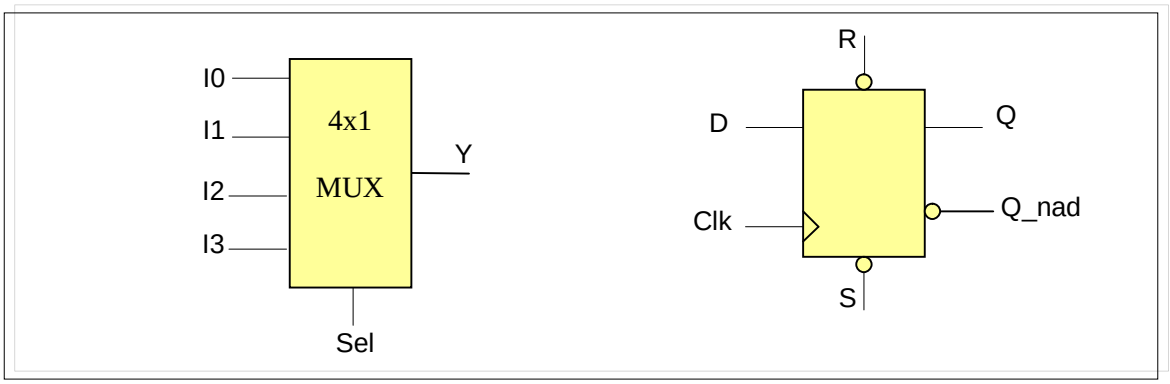
Spoljni opis entiteta zadaje se deklaracijom koja se naziva '*entity*'. Spoljni opis polusabirača sa Slika 1.3 ima sledeći izgled:

```
entity polusabirac is
port (a, b: in bit;
      s, C: out bit);
end polusabirac;
```

Zadebljanim (masnim) slovima označene su ključne reči (koje ne mogu da se koriste za nazive signala). Naziv entiteta je *'polusabirac'*. Ulaznim portovima pridruženi su signali *a* i *b*, koji su deklarisani kao tipovi *'bit'*, što znači da predstavljaju po jedan bit koji može imati vrednosti *1* ili *0*. U IEEE 1164 standardu, umesto tipa signala *'bit'* koristi se *'std_ulogic'* tako da prethodna deklaracija ima sledeći izgled:

```
entity polusabirac is
port (a, b: in std_ulogic;
      s, C: out std_ulogic);
end polusabirac;
```

Za niz bita koristi se tip signala `'bit_vector'` odnosno `'std_ulogic_vector'`.



Slika 1.4: Grafički simboli multiplsera (levo) D flip-flopa (desno)

Slika 1.4 (levo) prikazuje grafički simbol multipleksera sa 4 ulazna signala (obeležena sa *I0* do *I3*), jednim izlaznim signalom *Y* i signalom za selekciju *Sel*. Spoljni opis ovog multipleksera ima sledeći oblik:

```
entity multiplekser is
port (I0, I1, I2, I3 : in std_ulogic_vector(7 downto 0);
                Sel: in std_ulogic_vector(1 downto 0);
                  Y: out std_ulogic_vector(7 downto 0));
end multiplekser;
```

Slika 1.4 (desno) prikazuje grafički simbol D flip-flopa sa ulaznim signalima *D, Clk, R* i *S* i izlaznim signalima *Q* i *Q_nad*. Spoljni opis D flip-flopa ima sledeći oblik:

```
entity D_flip_flop is
port (D, Clk, R, S : in std_ulogic;
           Q, Qbar: out std_ulogic);
end multiplekser;
```

2.4 Opis ponašanja

Interno ponašanje (funkcionalnost) entiteta opisuje se primenom strukture koja se naziva *'**architecture**'*. Ponašanje digitalnog sistema može da se opiše na različite načine, na primer primenom tablice istinitosti, Bulovim funkcijama ili opisom strukture (kako su komponenete entiteta međusobno povezane).

VHDL mora da obuhvati sledeće elemente ponašanja entiteta:

- događaje (promene signala),
- kašnjenja i
- konkurentno izvršavanja operacija.

Ponašanje polusabirača može se opisati sledećim modulom kome je dodeljeno ime *'ponasanje'*.

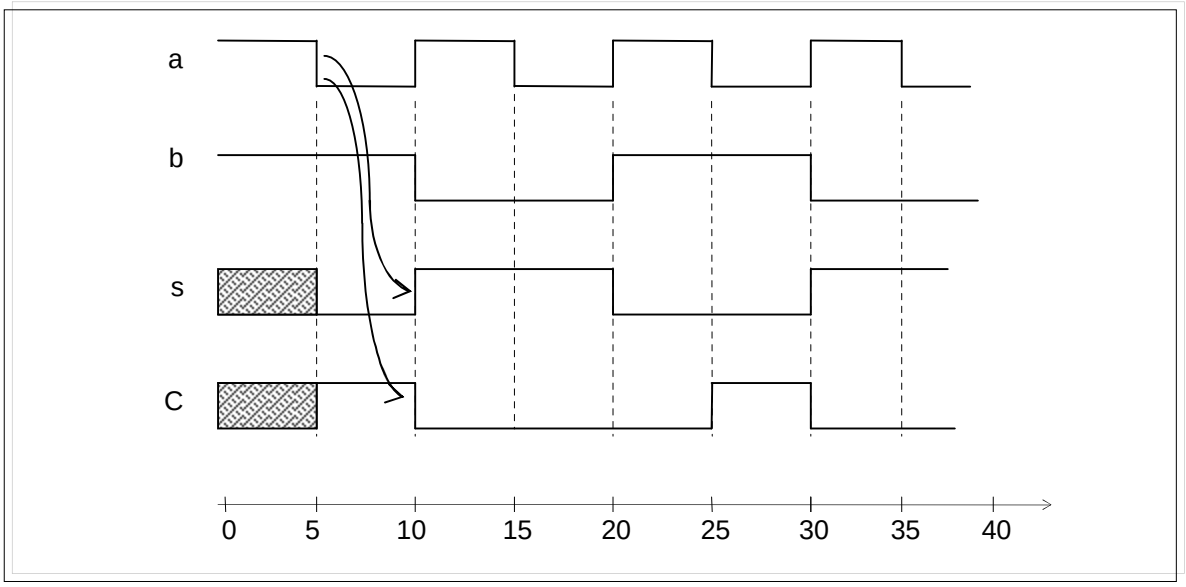
```
architecture ponasanje of polusabirac is
begin
  s <= (a xor b) after 5 ns;
  C <= (a and b) after 5 ns;
end ponasanje;
```

Operator '<=' je operator dodele vrednosti signalu. Svaka naredba određuje kako se na osnovu ulaznih signala izračunava vrednost izlaznih signala. Kašnjenje je definisano ključnom reči *'after'*.

Naredbe se izvršavaju konkurentno, što znači da njihov redosled nije značajan. Na primer, naredbe za izračunavanje izlaza *s* i *C* mogu da zamene mesta, ali će opis ponašanja ostati isti. Kompletan opis polusabirača u VHDL-u ima sledeći izgled:

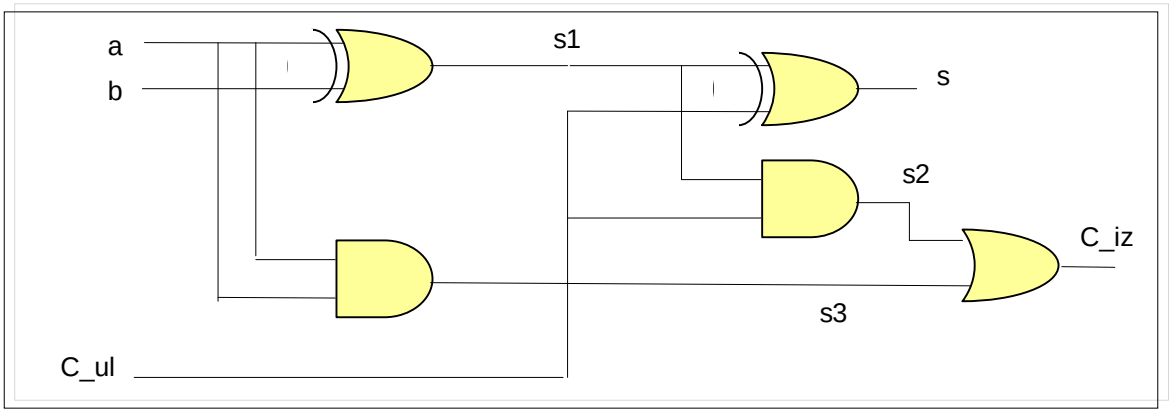
```
library IEEE;
use IEEE.std_logic_1164.all;

entity polusabirac is
port (a, b: in std_ulogic;
      s, C: out std_ulogic);
end polusabirac;

architecture ponasanje of polusabirac is
begin
  s <= (a xor b) after 5 ns;
  C <= (a and b) after 5 ns;
end ponasanje;
```

Slika 1.5 prikazuje vremenski dijagram simulacije ponašanja polusabirača opisanog prethodnim VHDL modelom za neke ulazne signale.



Slika 1.5: Vremenski dijagram simulacije rada polusabirača za neke ulazne signale

Slika 1.6 prikazuje logičku šemu punog sabirača sa tri ulazna signala, *a*, *b* i *C_u* (ulazni prenos) i dva izlazna signala *s* (suma) i *C* (prenos).



Slika 1.6: Logička šema punog sabirača

U punom sabiraču, pored ulaznih (*a, b, C_ul*) i izlaznih (*s, C_iz*) signala, pojavljuju se unutrašnji signali (*s1, s2* i *s3*). Naravno, u složenijim kolima može biti veći broj unutrašnjih signala. Pošto se u entitetu deklarišu samo ulazni i izlazni signali, u opisu ponašanja koristi se naredba *'signal'* za deklarisanje unutrašnjih signala *s1, s2* i *s3*. VHDL model punog sabirača sa slike ima sledeći izgled:

```
library IEEE;
use IEEE.std_logic_1164.all;

entity punisabirac is
port (a, b, C_in: in std_ulogic;
      s, C_iz: out std_ulogic);
end punisabirac;
```

```
architecture ponasanje_puni of punisabirac is

signal s1, s2, s3: std_ulogic;
constant gate_delay: Time := 5 ns;

begin
   s1 <=    (a  xor b)   after gate_delay;
   s2 <= (C_ul  and  s1)  after gate_delay;
   s3 <=    (a  and  b)   after gate_delay;
   s  <=   (s1 xor  C_ul) after gate_delay;
   C_iz <= (s2  or s3)    after gate_delay;
end ponasanje_puni;
```

naredbe za deklaraciju unutrašnjih signala

telo opisa ponašanja

Sve naredbe u telu opisa ponašanja kod simulacije se izvršavaju konkurentno.

Konstante se deklarišu i mogu bili određenog tipa, na primer ***Time***. Konstante moraju imati vrednost pre početka simulacije i ne mogu da menjaju vrednost u toku simulacije.

Simlacija se obavlja u dva koraka koji se ponavljaju:

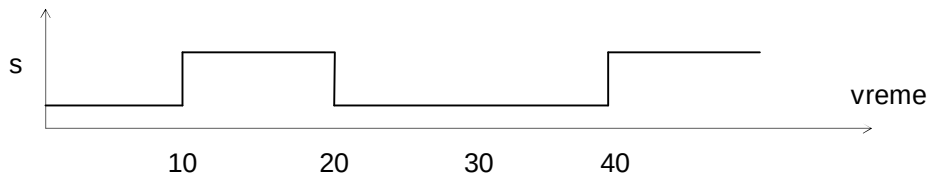
- U prvom koraku izvršavaju se desne strane naredbi dodele kod kojih postoji promena vrednosti signala. Na osnovu rezultata izvršavanja i na osnovu kašnjenja određuju se promene signala na levoj strani i vreme u kome ove promene treba da se dese.
- Vreme se inkrementira na vrednost koja odgovara prvoj promeni signala koja je ranije utvrđena.

2.5 Vremenski dijagram signala

Signal može da ima jednu ili više promena koje su određene trenucima u kojima se dešavaju promene. Na primer, promena signala u toku vremena može se zadati na sledeći način:

```
s <= '0', '1' after 10 ns, '0' after 20 ns , '1' after 40 ns;
```

Ovoj naredbi odgovara sledeći vremenski dijagram u toku simulacije:



Slika 1.7: Vremenski dijagram signala iz primera

2.6 Uslovna dodela vrednosti signalu

U primeru 4x1 multipleksera sa 8-bitnim signalima, izlaz multipleksera jednak je jednom od ulaza, u zavisnosti od vrednosti signala za selekciju *s1* i *s0*. VHDL model ovog multipleksera ima sledeći izgled.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity mux4 is
port (In0,In1, In2, In3: in std_logic_vector (7 downto 0);
      S0, S1: in std_logic;
      Z: out std_logic_vector (7 downto 0);
end mux4;

architecture ponasanje of mux4 is
begin
  Z <= In0 after 5 ns when S0 = '0' and S1 = '0' else
       In1 after 5 ns when S0 = '0' and S1 = '1' else
       In2 after 5 ns when S0 = '1' and S1 = '0' else
       In3 after 5 ns when S0 = '1' and S1 = '1' else
       "00000000" after 5 ns ;
end ponasanje;
```

U ovom primeru promena signala na ulazima *In0* do *In3* i/ili promena signala za selekciju *S0* i *S1* mogu da dovedu do promene izlaznog signala. Zbog toga kod bilo koje promene ovih signala moraju da se izračunavaju sve četiri uslovne naredbe.

Treba obratiti pažnju da u ovom slučaju redosled navođenja naredbi ima značaja jer se izvršava prva naredba kod koje je uslov tačan. U datom primeru vidi se da zbog prirode signala za selekciju samo jedan uslov može biti tačan, tako da u ovom posebnom slučaju redosled navođenja naredbi ne utiče na dodelu vrednosti izlaznom signalu.

2.7 Selekcija

U nekim primerima neophodno je izabrati jednu od mogućih opcija. Na primer, u digitalnom sistemu koji se sastoji od više registara, ulazna adresa određuje registar iz koga se čita sadržaj i dodeljuje izlaznom signalu.

Naredba za selekciju radi slično kao i uslovna naredba, ali važna razlika je da se svi uslovi izračunavaju i samo jedan od njih može biti tačan. Dalje, uslovi moraju pokriti sve moguće slučajeve, odnosno ne sme se dozvoliti da neka kombinacija ulaznih siganala, koji se koriste u izračunavanju uslova, nema odgovarajuću naredbu dodele vrednosti izlaznom signalu.

Kao primer uzmimo digitalni sistem sa 4 registra (*reg0* do *reg3*) iz kojih se istovremeno može čitati na dva izlazna porta *reg_out_1* i *reg_out_2*. Adresa *adr1* koristi se za izbor registra koji se čita na portu *reg_out_1*, a adresa *adr2* za izbor registra na portu *reg_out_2*. Međutim, da bi ilustrovali pokrivanje svih mogućih slučajeva, pretpostavimo da adresni signali *addr1* i *addr2* imaju po tri bita. Sledeći VHDL model opisuje ponašanje ovakvog digitalnog sistema.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity reg4 is
port (adr1, adr2: in std_logic_vector (2 downto 0);
      reg_out_1, reg_out_2: out std_logic_vector(31 downto 0);
end reg4;

architecture ponasanje of reg4 is
signal reg0, reg2: std_logic_vector (31 downto 0) =
          to_stdlogicvector (x"12345678");
signal reg1, reg3: std_logic_vector (31 downto 0) =
          to_stdlogicvector (x"9abcdef0");

begin
 with adr1 select
  reg_out_1 <= reg0 after 5 ns when "000",
               reg1 after 5 ns when "001",
               reg2 after 5 ns when "010",
               reg3 after 5 ns when "011",
               reg3 after 5 ns when other;

 with adr2 select
  reg_out_2 <= reg0 after 5 ns when "000",
               reg1 after 5 ns when "001",
               reg2 after 5 ns when "010",
               reg3 after 5 ns when "011",
               reg3 after 5 ns when other;
end ponasanje;
```

Treba primetiti da se kod deklaracije registari istovremeno inicijalizuju na početne vrednosti. Pošto su početne vrednosti date u obliku heksadecimalnih brojeva, funkcija *to_stdlogicvector( )* koristi se za pretvaranje heksadecimalnih brojeva u signal tipa *std_logic_vector*. Ova funkvija nalazi se u biblioteci `std_logic_1164.`

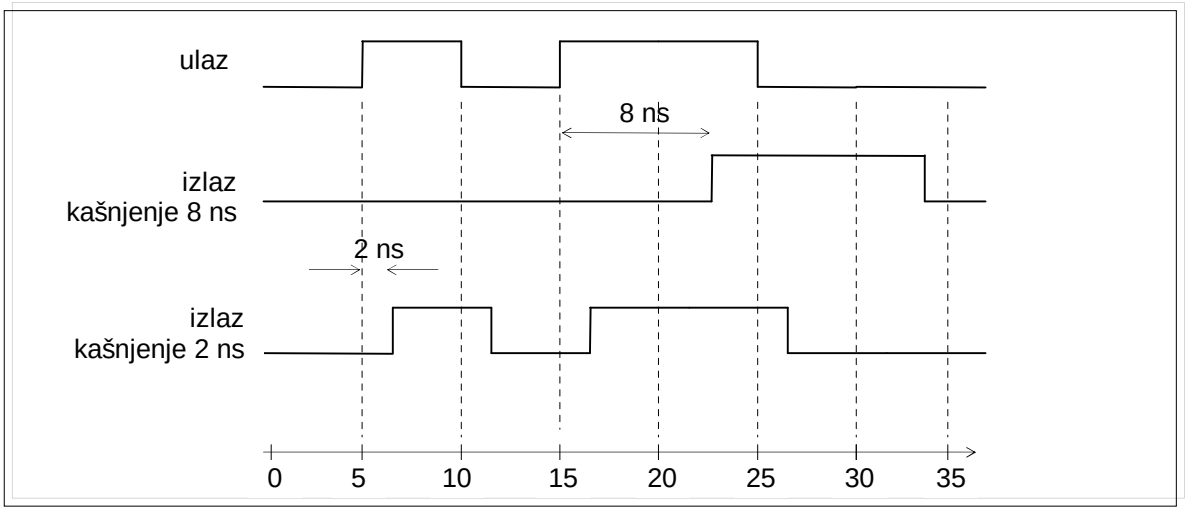
2.8 Kašnjenja

VHDL modelira tri vrste kašnjenja: inercijalno, transportno i delta kašnjenje.

Inercijalno kašnjenje

Kod inercijalnog kašenjenja signal na ulazu logičkog kola mora da traje duže od nekog minimalnog vremena da bi se pojavio na izlazu logičkog kola. Signali koji su kraći od minimalnog trajanja,

nemaju uticaj na izlaz logičkog kola. Obično se za minimalno vreme uzima vreme kašnjenja ili propagacije signala kroz logičko kolo. Sledeći vremenski dijagram signala opisuje signale na ulazu i izlazu logičkog kola u slučajevima da kolo ima kašnjenje 8 ns i 2 ns.



Slika 1.8: Ilustracija inercijalnog kašnjenja

VHDL dozvoljava da se u naredbi eksplicitno navede minimalno vreme trajanja signala:

*sum <= **reject** 2 **ns inertial** (a **and** b) **after** 5 **ns;***

Transportno kašnjenje

Komponente sa transportnim kašnjenjem prenose signal bilo kojeg trajanja (širine) sa ulaza na izlaz, zakašnjene za specificirani vremenski interval. Primer ovakvih komponenata su na primer provodnici sa konačnim prostiranjem signala.

*sum <= **transport** (a **and** b) **after** 5 **ns;***

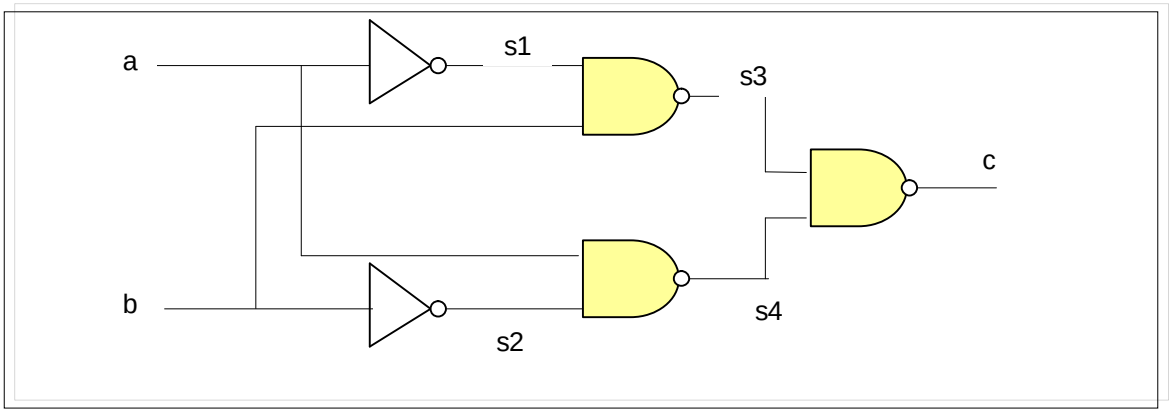
Ukoliko nije navedena ključna reč '*inertial*' niti '*transport*' onda se podrazumeva da je kašnjenje inercijalnog tipa.

Delta kašnjenje

U nekim slučajevima nismo zainteresovani da eksplicitno navodimo kašnjenje. Na primer, kašnjenje ne mora da se navodi ako programski alat već ima u svojoj biblioteci opise komponenata sa već ugrađenim kašnjenjem. Ili, u postupku sinteze, projektanta ne interesuje fizička realizacija i ponašanje sistema u vremenu, već pre svega funkcionalnost projektovanog digitalnog sistema. Primer naredbe bez navođenja kašnjenja:

*sum <= (a **and** b);*

Problem koji je neophodno rešiti kod modela bez kašnjenja je ispravan redosled dodeljivanja vrednosti signalima. Problem simulacije sistema bez kašnjenja pokazan je na primeru sledećeg sistema.



Slika 1.9: Primer za delta kašnjenje

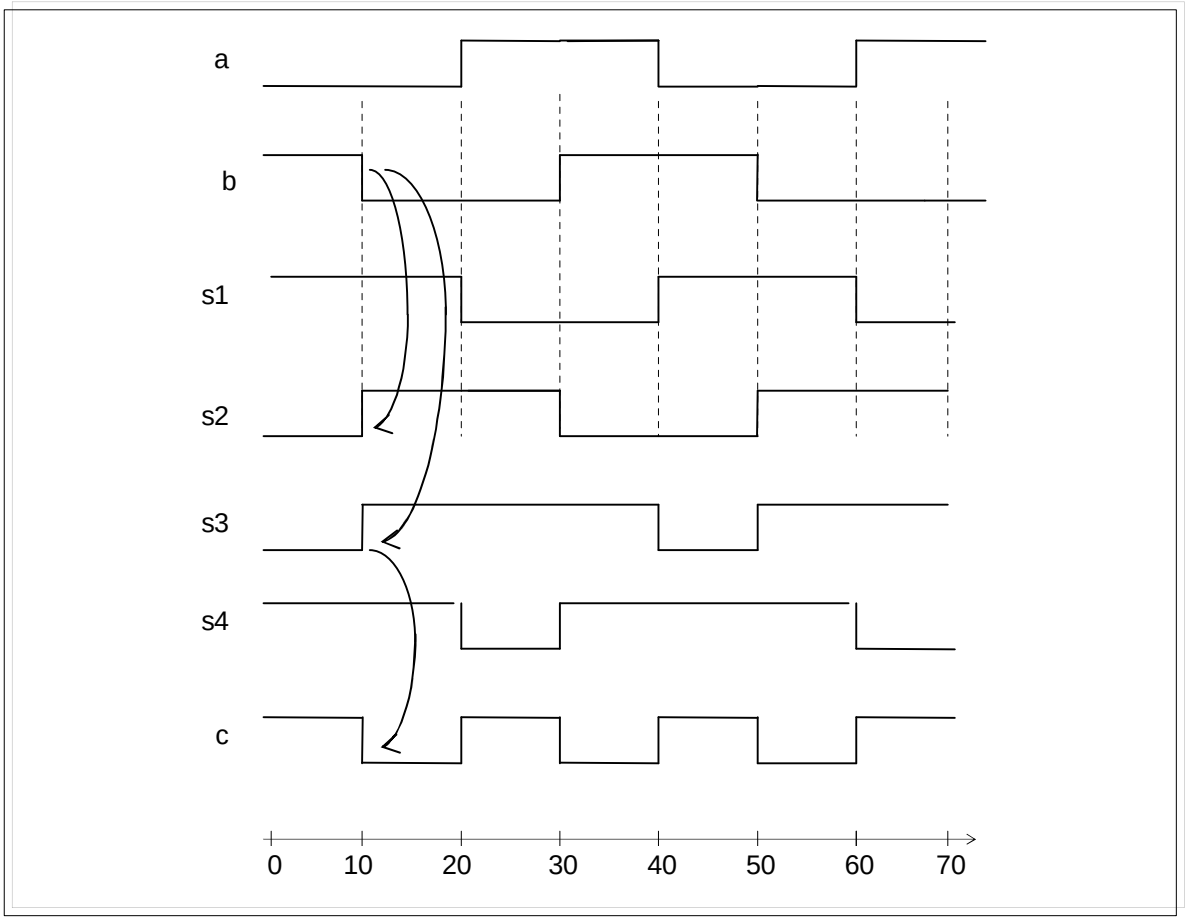
Digitalni sistem sa slike opisan je sledećim VHDL modelom u kome nije navedeno kašnjenje.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity delta_kas is
port (a, b: in std_logic;
          c: out std_logic);
end delta_kas;

architecture ponasanje of delta_kas is

signal s1, s2, s3, s4: std_logic := '0';

begin
   s1 <= not a;
   s2 <= not b;
   s3 <= not (s1 and b);
   s4 <= not (s2 and a);
   z  <= not (s3 and s4);
end ponasanje;
```

Da bi se odredio pravilan redosled dodele vrednosti signalima, ako nije navedeno kašnjenje, onda simulator podrazumeva da postoji beskonačno kratko kašnjenje, označeno sa ***'after 0 ns'*** :

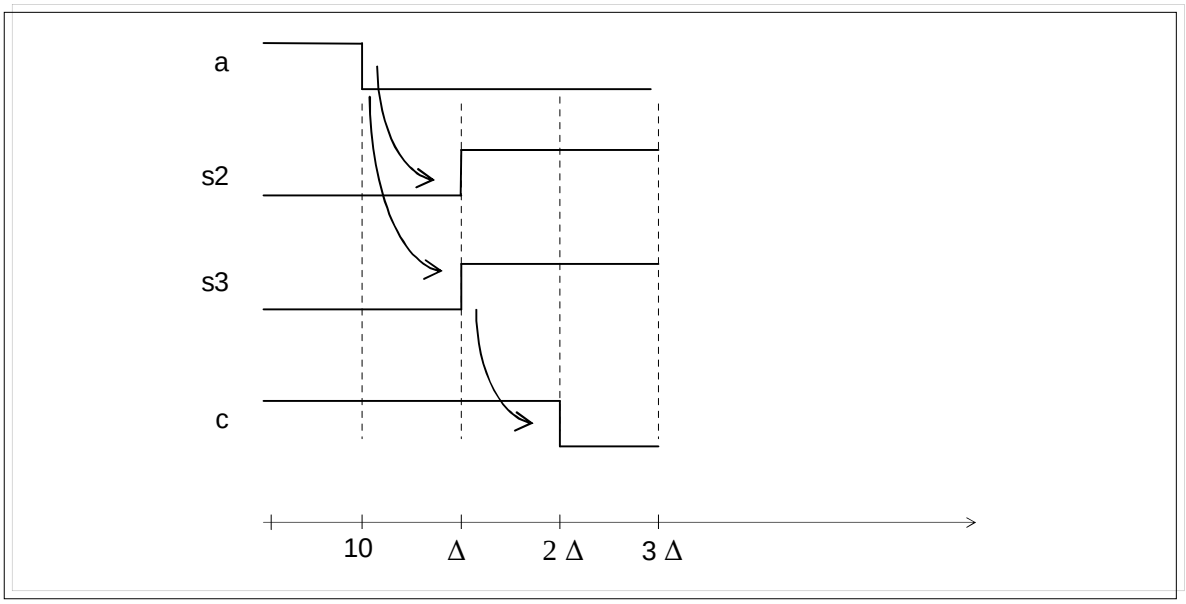
```
   s1 <= not a 'after 0 ns;
```

Uvođenjem beskonačno kratkog kašnjenja, redosled dodele vrednosti signalima obavlja se na isti način kao i kod inercijalnog ili transportnog kašnjenja.



Slika 1.10: Vremenski dijagram kod delta kašnjenja

Način simulacije promene signala označenog na prethodnom vremenskom dijagramu prikazan je na dijagramu sa Slika 1.11.



Slika 1.11: Primer simulacije promene signala kod delta kašnjenja

2. Modeliranje ponašanja digitalnih sistema

U ovom poglavlju izloženo je modeliranje ponašanja digitalnih sistema koji ne mogu da se opišu kao jednostavni elementi sa kašnjenjem. Uveden je pojam procesa koji omogućava opis ponašanja složenih komponenata digitalnih sistema.

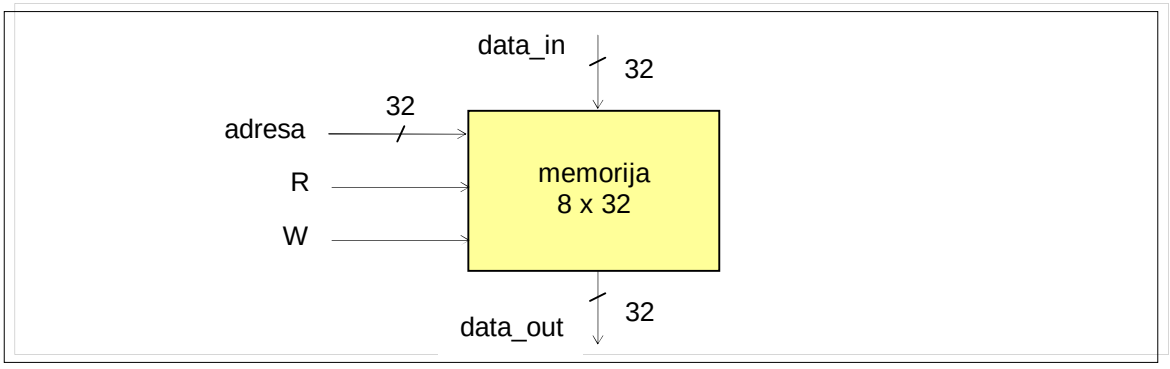
2.1 Proces

Proces omogućava da se komponenta digitalnog sistems modelira kao jedna naredba dodeljivanja vrednosti signalu ili signalima, pri čemu mogu da se koriste standardne naredbe programskog jezika kao što je C ili Pascal. Proces je grupa naredbi koje se sekvencijalno izvršavaju bez kašnjenja.

Primena procesa ilustrovana je na primeru modeliranja memorije sa 8 memorijskih lokacija, pri čemu svaka lokacija ima 32 bita. Na osnovu adresnih signala (*adresa*) memorija obavlja operaciju čitanja (ako je aktivan upravljački signal *R*) ili upisa (aktivan upravljački signal *W*). Operacije memorije opisane su sledećim RTL naredbama:

R: *data_out* ← *M[adresa]*
W: *M[adresa]* ← *data_in*

Spoljni signali memorije prikazani su na Slika 2.1.



Slika 2.1: Spoljni signali memorije 8x32 bita

Naredbe VHDL-a, koje su do sada objašnjene, nisu pogodne za modeliranje ponašanja memorije sa navedenim operacijama.

VHDL model memorije, sa vremenom pristupa 10 ns, predstavljen je sledećim programom.

```
library IEEE;
```

```
use IEEE.std_logic_1164.all;
use WORK.std_logic_arith.all;

entity memorija is
port (adresa, data_in: in std_logic_vector (31 downto 0);
                  R, W: in std_logic;
          data_out: out std_logic_vector (31 downto 0));
end memorija;

architecture ponasanje of memorija is
type reg_niz is array (0 to 7) of std_logic_vector (31 downto 0);

begin

mem_proces: process ( R, W)
variable data_mem: reg_niz := (
  to_stdlogicvector (X"00000000"),
  to_stdlogicvector (X"00000000"),
  to_stdlogicvector (X"00000000"),
  to_stdlogicvector (X"00000000"),
  to_stdlogicvector (X"00000000"),
  to_stdlogicvector (X"00000000"),
  to_stdlogicvector (X"00000000"),
  to_stdlogicvector (X"00000000"));
variable adr: integer;

begin
adr := to_integer (adresa (2 downto 0));

if W = '1' then
   data_mem (adr) := data_in;

elsif R = '1' then
  data_out <= data_mem (adr) after 10 ns;
endif;
end process mem_proces;
end ponasanje;
```

deklarativni deo procesa: definicija promenljivih i konstanti

telo procesa

Memorija je modelirana kao niz od osam 32-bitnih reči. Pošto se niz indeksira celim brojem *(integer)* funkcija *to_integer( )* prevodi tip podatka *std_logic_vector*u celi broj.

Naredbe procesa izvršavaju se sekvencijalno, pri čemu se dodela vrednosti promenljivima (označena simbolom :=) odvija bez kašnjenja. U okviru procesa mogu se izvršiti naredba dodele vrednosti signalima koji su definisani izvan procesa. U odnosu na vreme simulacije, vreme izvršenja procesa je jednako nuli, a po završetku procesa signali dobijaju nove vrednosti posle navedenog kašnjenja. Proces može da se posmatra i kao jedna složena naredba dodele vrednosti signalu.

U deklaraciji procesa navodi se se lista ulaznih sinala, koja se naziva 'lista osetljivosti'. Proces se izvršava kada nastupti promena vrednosti signala u listi osetljivosti.

2.2 Konkurentni procesi

VHDL program može imati više procesa koji se konkurentno izvršavaju. Sledeći primer sadrži dva procesa koji realizuju dva izlaza polusabirača.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity polusabirac is
port (a, b: in std_logic;
       suma, prenos: out std_logic);
end  polusabirac;

architecture ponasanje of polusabirac is
begin

suma_proces: process ( a, b)
        begin
          if (a = b)  then
             suma  <= '0'  after 5 ns;
          else
             suma  <= (a or b)  after 5 ns;
          end if;
end process  suma_proces;

prenos_proces: process ( a, b)
        case  a is
          when '0' =>
             prenos  <= a  after 5 ns;

          when '1' =>
             prenos  <= b  after 5 ns;

          when others =>
             prenos  <= 'X'  after 5 ns;
        end case
end process  prenos_proces;
end  ponasanje;
```

Primer koji je bio na kolokvijumu 11. januara 2007

Napisati VHDL model ponašanja dekodera 2/4 sa 2 ulazna signala *x1* i *x0* i izlaznim signalima *d3* do *d0*. Ponašanje dektodera modelirati procesom koji treba da se izvrši kada nastupi promena u nekom od ulaznih signala. Pretpostaviti da je kašnjenje izlaznih signala dekodera 5 ns u odnosu na ulazne signale. Predvidti ulazni signal *D*, kada je *D = 0*, svi izlazi su na 0, a kada je *D= 1*, onda je selektovani izlaz na 1, a ostali izlazi na 0. Predvideti ulazni signal dozvole D, tako da kada je D = 0, svi izlazni signali su 0, a kada je D=1, selektovani izlazni signal je na 1, a ostali signali isu na 0..

D	x1	x0	d3	d2	d1	d0
0	-	-	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

```
library IEEE;
use IEEE.std_logic_1164.all;

entity 2_na_4_dekoder is
port ( D, x1, x0: in std_logic;
       d3, d2, d1, d0: out std_logic);
end  2_na_4_dekoder;

architecture ponasanje of 2_na_4_dekoder is
begin

process (x1, x0)
begin
 if D = '1' then
       d0 <= ( not x1) and ( not x0) after 5 ns;
       d1 <= ( not x1) and       x0  after 5 ns;
       d2 <=       x1  and ( not x0) after 5 ns;
       d3 <=       x1  and       x0  after 5 ns;

 else
       d0 <= '0' after 5 ns;
       d1 <= '0' after 5 ns;
       d2 <= '0' after 5 ns;
       d3 <= '0' after 5 ns;

end process;
end  ponasanje;
```

kraj primera

2.3 Naredbe čekanja

Modeli u prethodnim primerima rade tako što se procesi izvršavaju kada nastupi promena u signalima iz liste osetljivosti. Ukoliko proces treba da se izvrši ako se ispune neki drugi uslovi, na primer kada istekne predviđeno vreme, onda se koristi naredba ***wait***.

Naredba ***wait*** može da se koristi za navođenje uslova pod kojim se izvršava proces:

```
wait for period_vremena;
wait on signal;
wait until uslov;
```

Naredba ***wait for*** *period_vremena*; suspenduje proces dok ne istekne navedeni period vremena. Naredba ***wait on*** *signal*; suspenduje proces dok se ne dogodi promena vrednosti signala koji je naveden. Umesto jednog signala može se navesti lista signala i proces je suspendovan do promene jednog ili više signala u listi. Naredba ***wait until*** *uslov*; suspenduje proces dok navedeni uslov ne bude tačan.

Signalima su pridruženi atributi. Na primer, atribut ***event*** označava da se desila promena signala. Predikat *clock'**event*** je tačan ukoliko se dogodila promena signala *clock* u poslednjem simulacionom ciklusu. Ovaj predikat može da se koristi kod čekanja na promenu signala *clock* sa logičke 0 na logičku 1:

```
wait until (clock'event and clock = '1');
```

Pošto promena može da bude sa 0 na 1 ili sa 1 na 0, mora da se koristi provera da li je signal promenjen tako da je nova vrednost signala jednaka logičkoj 1. Ista naredba može se napisati primenom funkcije za detekciju prednje ivice signala:

```
wait until rising_edge (clock);
```

Sledeći VHDL program modelira ponašanje D flip-flopa koji okida na prednjoj ivici sinhronizacionog signala.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity D_flip_flop is
port (D, clock: in std_logic;
      Q, Q_nad: out std_logic);
end D_flip_flop;

architecture ponasanje of D_flip_flop is
begin

D_proces: process
begin
    wait until (clock'event and clock = '1');
        Q <= D after 5 ns;
        Q_nad <= not D after 5 ns;
end process D_proces;
```

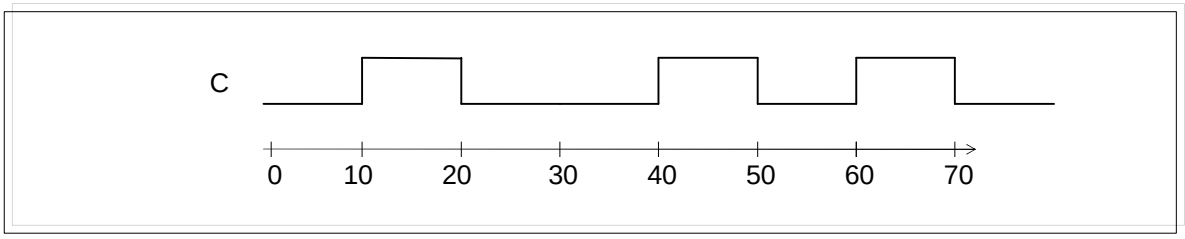
Primetiti da nema liste osetljivosti!

Primetiti da nema nema memorije jer signal zadržava vrednost do sledeće promene!

```
end ponasanje;
```

Naredba ***wait for*** *period_vremena*; često se koristi za beskonačno ponavljanje procesa. Sledeći model pokazuje generisanje talasnog oblika izlaznog signala prikazanog na Slika 2.2.

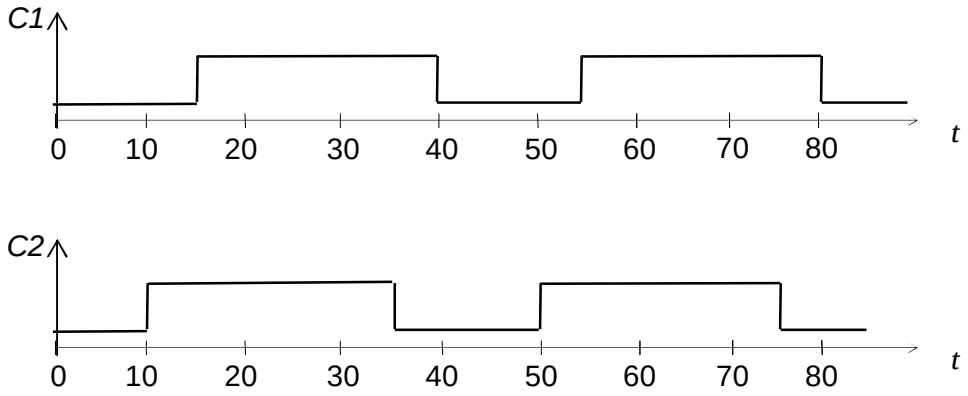
```
library IEEE;
use IEEE.std_logic_1164.all;

entity talasni_oblik_1 is
port (clock: out std_logic);
end talasni_oblik_1;

architecture ponasanje of talasni_oblik_1 is
begin
   process
   begin
      C <= '0', '1' after 10 ns, '0' after 20 ns, '1' after 40 ns;
      wait for 50 ns;
   end process;
end ponasanje;
```



Slika 2.2: Talasni oblik izlaznog signala

<u>Primer</u>: Kolokvijum održan 11. januara 2007. godine

Napisati VHDL model koji u toku simulacije na izlaznom signalima *C1* i *C2* generiše beskonačne povorke impulsa prikazane na dijagramu.



Rešenje

```
library IEEE;
use IEEE.std_logic_1164.all;

entity talasni_oblik is
port (C1: out std_logic);
end talasni_oblik;

architecture ponasanje of talasni_oblik is
begin
   process
   begin
      C1 <= '0', '1' after 15 ns;
      C2 <= '0', '1' after 10 ns, '0' after 35 ns;
      wait for 40 ns;
   end process;
end ponasanje;
```

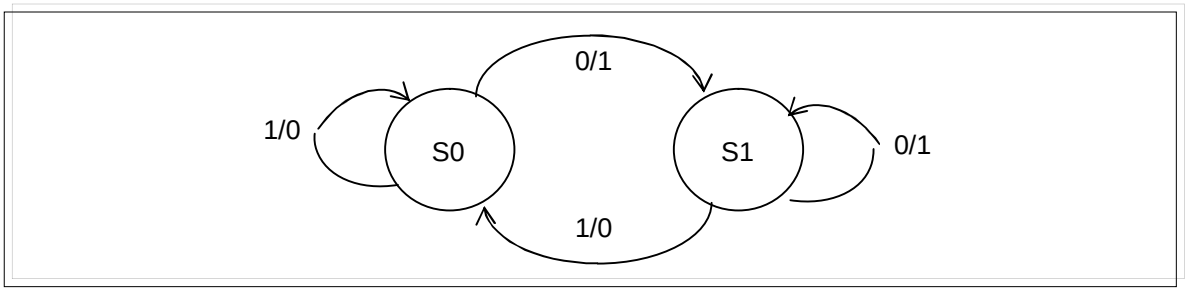
kraj primera

2.4 Modeliranje sekvencijalnih kola

Izlaz sekvencijalnih kola zavisi od trenutnih vrednosti ulaznih signala i stanja memorijskih elemenata kola. Kod sinhronih sekvencijalnih kola stanje može da se menja samo u dozvoljenim vremenskim trenucima određenim sinhronizacionih signalom.

Sekvencijalno kolo određeno je sa dve funkcije: funkcija prelaza određuje stanje kola u narednom vremenskom intervalu, a funkcija izlaza određuje izlaz kola. U opštem slučaju ulazi obe funkcije su stanje i ulazi sekvencijalnog kola.

Sekvencijalno kolo dekomponuje se u dve celine: kombinaciona mreža koja realizuje funkcije prelaza i izlaza i memorijski deo koji pamti stanje kola. Ove dve celine mogu se realizovati sa dva procesa.



Slika 2.3: Graf stanja sekvencijalnog kola

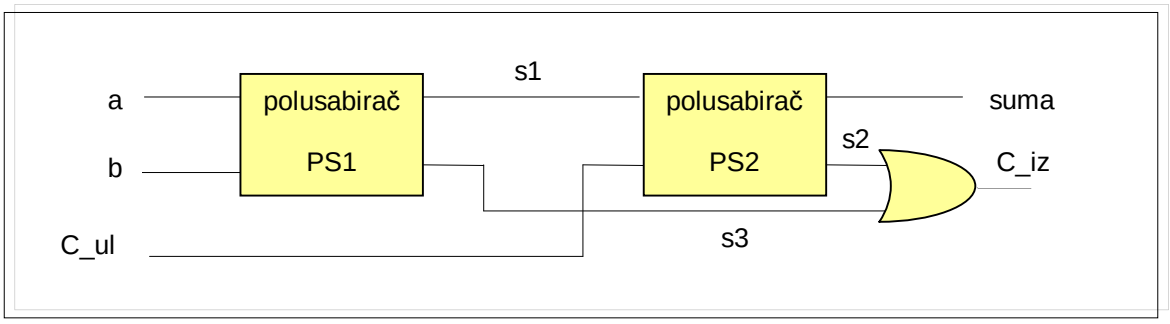
```
library IEEE;
use IEEE.std_logic_1164.all;

entity automat  is
port  (reset, clock,  x:  in std_logic;
                      y:  out std_logic);
end   automat;

architecture ponasanje  of  automat is
type stanje is (stanje0, stanje1);
signal  stanje_automata, naredno_stanje: stanje := stanje0;

begin

komb_proces: process (stanje_automata, x)
begin
    case stanje_automata is
    when stanje0  =>
        if  x = '0' then
            naredno_stanje <= stanje1;
            y <= '1';
        else
            naredno_stanje <= stanje0;
            y <= 'o';
        end if;


    when stanje1  =>
        if  x = '1' then
            naredno_stanje <= stanje0;
            y <= '0';
        else
            naredno_stanje <= stanje1;
            y <= '1';
        end if;
    end case;
end process  komb_proces;

mem_proces: process
begin
    wait until  clock'event  and clock = '1';
    if  reset = '1' then
        stanje_automata <= stanje'left;
    else
        stanje_automata <= naredno_stanje;
    end if;
end process  mem_proces;

end  ponasanje;
```

3. Modeliranje strukture digitalnih sistema

U ovom poglavlju izložena je primena VHDL-a u modeliranju strukture digitalnih sistema. Pod strukturom se podrazumeva skup komponenata i način na koji su te komponente međusobno povezane. Model koji je izložen u ovom poglavlju opisuje način povezivanja, pri čemu se ne daje opis funkcionalnosti komponenata sistema.

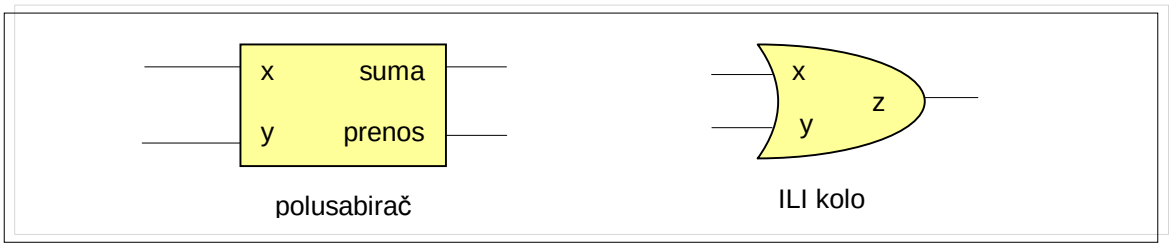
2.5 Opis strukture

Kod opisa strukture digitalnog sistema navode se pojedine komponente, njihovi spoljni (pristupni) signali i način međubosnog povezivanja pristupnih signala. Na Slika 3.1 prikazana je struktura punog sabirača koji se sastoji od dva polusabirača (označeni sa *PS1* i *PS2*) i jednog *ILI* kola.



Slika 3.1: Struktura punog sabirača

Slika 3.2 prikazuje komponente od kojih je sastavljen puni sabirač. Svaka komponenta predstavljena je svojim pristupnim signalima i njihovim nazivima.



Slika 3.2: Komponente od kojih je sastavljen puni sabirač

VHDL model koji opisuje strukturu digitalnog sistema mora da obuhvati sledeće:

- listu komponenti od kojih je sastavljen sistem,

- definiciju pristupnih signala koji se koriste kod povezivanja komponenti i
- jedinstveno označavanje i time korišćenje višestrukih pojavljivanja iste komponente.

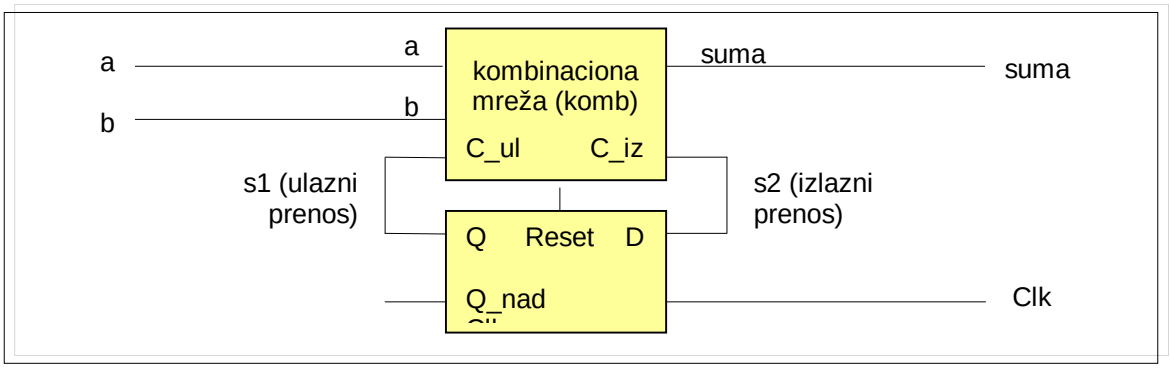
Struktura punog sabirača sa Slika 3.1 može se opisati sledećim VHDL modelom.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity puni_sabirac is
port ( a, b, C_ul: in std_logic;
       suma, C_iz: out std_logic);
end puni_sabirac;

architecture struktura of puni_sabirac is

component polusabirac
port (x, y: in std_logic;
      suma, prenos: out std_logic);
end component;

component ili_kolo
port (x, y: in std_logic;
      z: out std_logic);
end component;

signal s1,s2, s3: std_logic;
begin
  PS1: polusabirac port map ( x => a, y => b,     suma => s1,     prenos => s3);
  PS2: polusabirac port map ( x => s1, y => C_ul, suma => suma, prenos => s2);
  ILI1: ili_kolo port map ( x => s2, y => s3, z => C_iz);
end struktura;
```

U deklarativnom delu opisa arhitekture navode se sve komponente i signali koji će se koristiti u opisu strukture. Iz navedenog primera vidi se da deklarisane komponente mogu da se pojavljuju jedan ili više puta u telu strukturnog dela modela.

Labele odvojene dvotačkom, u primeru *PS1, PS2* i *ILI1* označavaju naredbe koje određuju kako su pristupne tačke svake komponente povezane sa signalima i pristupnim tačkama entiteta. Naravno, ponašanje komponenti mora biti opisano u nekom drugom delu VHDL modela digitalnog sistema.

Kao drugi primer uzmimo serijski sabirač, Slika 3.3. Na ulaze *a* i *b* dovode se serijski biti dva sabirka, od najmanje značajnog bita prema najznačajnijem. U svakoj periodi sinhronizacionog signala kombinaciona mreža računa jedan bit sume i izlazni prenos koji se pamti u D flip-flopu i sabira sa narednim parom bitova sabiraka.



Slika 3.3: Blok šema serijskog sabirača

Struktura serijskog sabirača sa Slika 3.3 opisana je sledećim VHDL modelom.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity serijski_sabirac is
port ( a, b, Clk, Reset: in std_logic;
                    suma: out std_logic);
end serijski_sabirac;

architecture struktura of serijski_sabirac is

component komb
port ( a, b, C_ul: in std_logic;
       suma, C_iz: out std_logic);
end component;

component D_flip_flop
port (Clk,Reset, D: in std_logic;
      Q,, Q_nad: out std_logic);
end component;

signal s1,s2: std_logic;

begin
  C: komb port map ( a => a, b => b, C_ul => s1, suma => suma);
  DFF: D_flip_flop port map ( Clk => Clk, Reset => Reset, D => s2, Q => s1,
                              Q_nad => open);
end struktura;
```

Napisati strukturni VHDL model 3/8 dekodera koji kao komponente koristi 2/4 dekoder (primer prikazan u odeljku 2.2). Ulazi 3/8 dekoder su *(x2, x1, x0)*, a izlazi (*d7* do *d0)*.

Rešenje

Bez upotrebe STD_LOGIC_VECTOR tipa

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity dekoder3_8 is
Port(
      x0    : in  STD_LOGIC;
      x1    : in  STD_LOGIC;
      x2    : in  STD_LOGIC;
      d0    : out STD_LOGIC;
      d1    : out STD_LOGIC;
      d2    : out STD_LOGIC;
      d3    : out STD_LOGIC;
      d4    : out STD_LOGIC;
      d5    : out STD_LOGIC;
      d6    : out STD_LOGIC;
      d7    : out STD_LOGIC);
end entity dekoder3_8;

architecture structural of dekoder3_8 is
component dekoder2_4 is
Port(
      en    : in  STD_LOGIC;
      x0    : in  STD_LOGIC;
      x1    : in  STD_LOGIC;
      d0    : out STD_LOGIC;
      d1    : out STD_LOGIC;
      d2    : out STD_LOGIC;
      d3    : out STD_LOGIC);
end component;

component logic_not is
Port(
      ul    : in  STD_LOGIC;
      izl   : out STD_LOGIC);
end component;

signal s1 : STD_LOGIC;

begin
      NOT1 : logic_not port map(ul => x2, izl => s1);
      DEC1 : dekoder2_4 port map (en => x2, x1 => x1, x0 => x0,
                                  d0 => d4, d1 => d5, d2 => d6, d3 => d7);
      DEC2 : dekoder2_4 port map (en => s1, x1 => x1, x0 => x0,
                                  d0 => d0, d1 => d1, d2 => d2, d3 => d3);
end architecture structural;
```

Sa STD_LOGIC_VECTOR tipom

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity dekoder3_8 is
Port(
      x     : in  STD_LOGIC_VECTOR (2 downto 0);
      d     : out  STD_LOGIC_VECTOR (7 downto 0));
end entity dekoder3_8;

architecture structural of dekoder3_8 is
component dekoder2_4 is
Port(
      en    : in  STD_LOGIC;
      x     : in  STD_LOGIC_VECTOR (1 downto 0);
      d     : out  STD_LOGIC_VECTOR (3 downto 0));
end component;

component logic_not is
Port(
      ul    : in  STD_LOGIC;
      izl   : out  STD_LOGIC);
end component;

signal s1 : STD_LOGIC;

begin
      NOT1 : logic_not port map(ul => x(2), izl => s1);
      DEC1 : dekoder2_4 port map (en => x(2), x => x(1 downto 0),
                                              d => d(7 downto 4));
      DEC2 : dekoder2_4 port map (en => s1, x => x(1 downto 0),
                                              d => d(3 downto 0));
end architecture structural;
```

kraj primera

2.6 Modeli sa promenljivim parametrima

VHDL dozvoljava da se prave opšti modeli u koje mogu kasnije da se unose vrednosti parametara. Na primer, model komponente sa parametrom za kašnjenje može da se koristi u drugim modelima pri čemu se dodeljuje vrednost parametru za kašnjenje. Drugi primer je registar sa parametrom koji predstavlja broj bita. Isti model može da se koristi za registre sa različitim brojem bita tako što se dodeljuju različite vrednosti parametru.

Sledeći VHDL model opisuje ekskluzivno ILI kolo sa parametrom *'kasnjenje'* koji može da se menja u zavisnosti od kašnjenja kola koje se primenjuje u realizaciji digitalnog sistema. Ključna reč ***generic*** koristi se za deklaraciju parametra koji je promenljiv.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity xor_gen is
generic (kasnjenje: Time := 2 ns);
port ( x, y: in std_logic;
       z: out std_logic);
end xor_gen;

architecture ponasanje of xor_gen is
begin
 z < = (x xor y) after kasnjenje);
end ponasanje;
```

Vrednost generičkog parametra *kasnjenje* zadaje se kod korišćenja modela. Na primer, sledeći deo VHDL modela polusabirača koristi prethodnu ekskluzivno ILI komponentu i zadaje vrednost generičkog parametra *kasnjenje*.

```
architecture gen_kasnjenje of polusabirac is

component xor_gen
generic (kasnjenje: Time);
port ( x, y : in std_logic;
       z: out std_logic);
end component;

component and_gen
generic (kasnjenje: Time);
port ( x, y : in std_logic;
       z: out std_logic);
end component;

signal s1,s2: std_logic;

begin
  EX1: xor_gen generic map ( kasnjenje => 6 ns)
```

```
                port map ( x => a,  y  => b,  z  => suma);
  AND1: and_gen  generic map ( kasnjenje => 3 ns)
                port map ( x => a,  y  => b,  z  => C_iz);
end  gen_kasnjenje;
```

2.7 Konfiguracije

Kao što je u prethodnim odeljcima pokazano, strukturni model sadrži komponente i veze između pristupnih tačaka komponeti, pristupnih tačaka sistema i internih signala. Po pravilu, entitet iz radnog direktorijuma, koji ima isti naziv kao komponenta, koristi se za simulaciju ponašanja komponenti strukturnog modela. Ukoliko entitet ima više različitih pridruženih arhitektura, onda se za simulaciju koristi arhitektura koja je poslednja prevedena (kompajlirana).

Projektant može u VHDL programu da navede biblioteku (direktorijum) u kojoj je par entitet-arhitektura sa opisom ponašanja koji treba koristiti u toku simulacije. U sledećem primeru pokazan je izbor parova entitet-arhitektura za polusabirače koji se koriste kao komponente u strukturnom opisu punog sabirača.

```
library IEEE;
library LOWPOWER;
use IEEE.std_logic_1164.all;

entity puni_sabirac is
port  ( a, b,  C_ul:  in std_logic;
          suma, C_iz:  out std_logic);
end  puni_sabirac;

architecture struktura  of  puni_sabirac is

component polusabirac
port  (x, y:  in std_logic
       suma, prenos:  out std_logic);
end  component;

component  ili_kolo
port  (x, y:  in std_logic;
          z:  out std_logic);
end  component;

signal s1,s2, s3:  std_logic;

-- navodjenje konfiguracije
for PS1: polusabirac  use entity WORK. polusabirac (ponasanje);
for PS2: polusabirac  use entity WORK. polusabirac (struktura);
for ILI1: ili_kolo  use entity LOWPOWER.lp_or_2 (ponasanje)
  generic map (gate_delay => gate_delay)
  port map (I1 => x, I2 => y, O1 => z);
```

```
begin
  PS1: polusabirac port map ( x => a,  y => b,      suma => s1,     prenos => s3);
  PS2: polusabirac port map ( x => s1, y => C_ul, suma => suma, prenos => s2);
  ILI1: ili_kolo  port map ( x => s2, y => s3, z  => C_iz);
end  struktura;
```

U ovom primeru, za polusabirače *PS1* i *PS2* koristi se isti entitet *(polusabirac)* koji je u radnom direktorijumu *WORK*. Arhitektura koja je pridružena entitetu navedena je u zagradi posle naziva entiteta. Treba primetiti da se za polusabirače *PS1* i *PS2* koriste različite arhitekture, jedna koja opisuje ponašanje *(ponasanje)* i druga koja opisuje strukturu *(strutkura)* polusabiraca.

Komponenti može da se pridruži entitet sa različitim imenom, kao što je to u prethodnom primeru slučaj sa komponentom *ILI1:ili_kolo*. U ovom slučaju mora da se navede mapiranje generičkih parametara (naredba ***generic map***) i mapiranje pristupnih tačaka (naredba ***port map***) između entiteta i komponente.

Konfiguracija, odnosno navođenje parova entitet-arhitektura koji opisuju komponente, može da se navede u posebnom modulu, određenom ključnom reči ***configuration***. Sledeći modul pokazuje konfiguraciju za puni sabirač iz prethodnog primera.

```
configuration konfiguracija1 of puni_sabirac is

for struktura

  for PS1: polusabirac  use entity WORK. polusabirac (ponasanje);
  end for;

  for PS2: polusabirac  use entity WORK. polusabirac (struktura);
  end for;

  for ILI1: ili_kolo  use entity LOWPOWER.lp_or_2 (ponasanje)
    generic map (gate_delay => gate_delay)
    port map (I1 => x, I2 => y, O1 => z);
   end for;

end for;
end  konfiguracija1;
```

2.8 Funkcije

Funkcije u VHDL-u vraćaju izračunati rezultat na osnovu vrednosti ulaznih parametara. Deklaracija funkcije ima sledeći izgled:

```
function rising_edge ( signal clock: in std_logic)  return boolean;
```

Deklaracija funkcije sadrži ime funkcije, formalne (ulazne) parametre i tip rezultata koji se dobija izračunavanjem na osnovu vrednosti ulaznih parametara. Funkcija može da ima samo ulazne parametre, čije vrednosti funkcija ne može da menja. Struktura funkcije ima sledeći izgled:

```
function prednja_ivica ( signal clock: in std_logic)  return boolean is
  -- deklaracije
  --
begin
 -- telo funkcije
 --
 return ( izraz)
end prednja_ivica;
```

Poziv funkcije sadrži stvarne parametre čije vrednosti zamenjuju formalne parametre. Tipovi podataka stvarnih i formalnih parametara moraju da se podudaraju. Na primer, u pozivu funkcije *prednja_ivica* kao parametar može da se koristi signal koji se zove *ulaz1*:

```
prednja_ivica ( ulaz1);
```

Simulaciono vreme izvršavanja funkcije je jednako nuli. Za razliku od procesa, u telu funkcije i u procedurama koje funkcija poziva ne mogu da se koristi naredbe ***wait***.

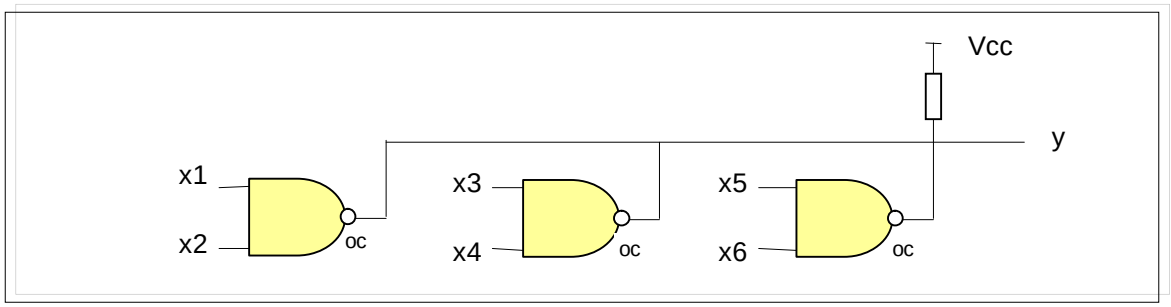
Sledeći VHDL program pokazuje primenu funkcije za detekciju prednje ivice u modelu ponašanja D flip-flopa koji menja stanje na prednjoj ivici sinhronizacionog signala *Clk*.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity D_flip_flop is
port (D, Clk: in std_logic;
      Q, Q_nad: out std_logic);
end  D_flip_flop ;

architecture ponasanje of D_flip_flop is

function prednja_ivica (signal clock: std_logic)  return boolean is
  variable  ivica: boolean := FALSE;
  begin
     ivica := (clock = '1'   and clock'event):
     return (ivica );
end  prednja_ivica;

begin
  Dff_izlaz: process
  begin
    wait until  (prednja_ivica (Clk));

    Q  < =  D  after 5  ns;
```

```
    Q_nad <= not D after 5 ns;

  end process Dff_izlaz;
end ponasanje;
```

2.9 Funkcije za određivanje signala na zajedničkim izlazima

Funkcije se posebno koriste u rešavanju problema određivanja vrednosti signala na zajedničkim izlazima dva ili više logičkih kola. Na primer, u digitalnom sistemu prikazanom na Slika 3.4, izlazi logičkih kola sa otvorenim kolektorima kratko su spojeni tako da realizuju ožičenu logičku I operaciju. U ovakvim primerima vrednost izlaza zavisi od izlaza dva ili više logičkih kola i moraju na poseban način da se izračunavaju.



Slika 3.4: Izlazi logičkih kola sa otvorenim kolektorom spojeni u zajednički izlaz

Naravno, ne smeju se međusobno kratko spajati izlazi logičkih kola koja na svojim izlazima daju isključivo ili logičku 1 ili logičku 0. Kratko spajanje izlaza dozvoljeno je samo za logička kola koja imaju posebnu konfiguraciju izlaznog stepena, na primer izlaz sa otvorenim kolektorom ili izlaz koji može biti u stanju visoke impendanse. Iz ovih primera jasno je da VHDL mora da ima mogućnost predstavljanja signala koji osim vrednosti logičke *0* i logičke *1*, mogu da imaju i neke druge vrednosti.

U IEEE 1164 standardu tip signala *std_ulogic* deklarisan je kao nabrojivi tip na sledeći način:

```
type std_ulogic is
        ( 'U',    -- neinicijalizovan
          'X',    -- nepoznat
          '0',    -- logička 0
          '1',    -- logička 1
          'Z',    -- visoka impendansa
          'W',    -- nepoznat, male snage
          'L',    -- logička 0, male snage
          'H',    -- logička 1, male snage
          '-'     -- svejedno
        );
```

Rezolucija je postupak utvrđivanja vrednosti signala na zajedničkoj liniji na koju su dovedeni izlazi više logičkih kola. Algoritam za računanje vrednosti signala realizuje se rezolucionom funkcijom. Jedan od načina zadavanja algoritma za rezoluciju je preko tabele u kojoj su navedene sve mogućnosti koji mogu da imaju izlazi logičkih kola koji su kratko spojeni i vrednosti zajedničkog izlaza u svakoj od mogućih kombinacija.

Sledeća tabela predstavlja algoritam za rezoluciju u slučaju kada se vežu izlazi *y1* i *y2* dvaju logičkih kola koji imaju tip signala *std_ulogic*. Kolone predstavljaju moguće vrednosti izlaza logičkog kola *y1*, vrste moguće vrednosti izlaza *y2*, a u presecima vrsta i kolona date su vrednosti signala na zajedničkoj liniji.

y2 \ y1	U	X	0	1	Z	W	L	H	–
U	U	U	U	U	U	U	U	U	U
X	U	X	X	X	X	X	X	X	X
0	U	X	0	X	0	0	0	0	X
1	U	X	X	1	1	1	1	1	X
Z	U	X	0	1	Z	W	L	H	X
W	U	X	0	1	W	W	W	W	X
L	U	X	0	1	L	W	L	W	X
H	U	X	0	1	H	W	W	H	X
–	U	X	X	X	X	X	X	X	X

Slika 3.5: Tabela za rezoluciju dva kratko spojena signala *y1* i *y2* tipa *std_ulogic*

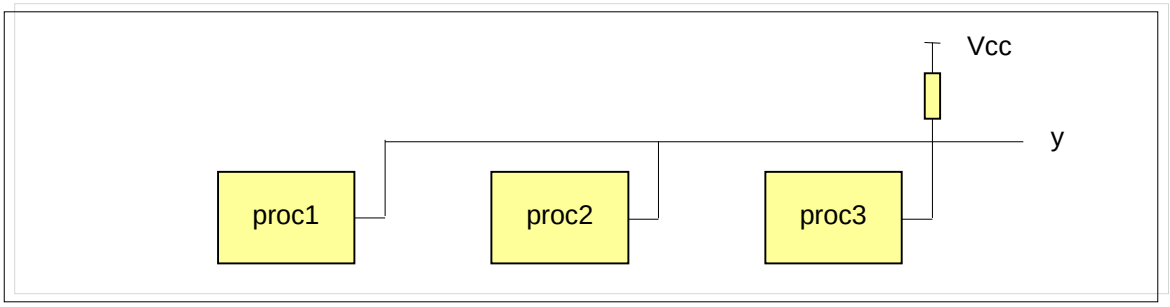
Na osnovu tabele možemo napisati funkciju (na primer koja se zove ***resolved***) koja implementira rezoluciju dva ili više signala tipa *std_ulogic*. Deklaracija funkcije *resolved* ima sledeći izgled:

function *resolved (s : std_ulogic_vector)* ***return*** *std_ulogic;*

Na osnovu ove funkcije možemo deklarisati novi tip signala *std_logic* koji može da ima 9 vrednosti kao i nabrojani tip signala *std_ulogic* ali može da podržava kratko spojene izlazne signale više logičkih kola. Deklaracija ima sledeći izgled:

subtype *std_logic* ***is*** *resolved std_ulogic ;*

Deklaracija ***subtype*** označava da signal može da ima samo vrednosti koje su obuhvaćene tipom signala *std_ulogic* i da postoji funkcija *resolved* koja se koristi za rezoluciju ovog tipa signala. Svaki put kada u toku simulacije signal tipa *std_logic* na zajedničkoj liniji treba da dobije novu vrednost, poziva se funkcija *resolved* koja proverava vrednosti svih signala vezanih na zajedničku liniju i određuje vrednost izlaznog signala.



Slika 3.6: Primer tri podsistema sa ožičenom logičkom I operacijom na izlazima

Na Slika 3.6 prikazana su tri podsistema čiji izlazi imaju otvoreni kolektor, tako da kad se vežu na zajedničku liniju na izlazu *y* implementiraju ožičenu logičku I operaciju. Pretpostavimo da je ponašanje podsistema opisano procesima *proc1, proc2* i *proc3.* Sledeći VHDL program pokazuje kako se koristi funkcija *resolved* za određivanje vrednosti izlaznog signala *y*.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity tri_podsistema is
end tri_podsistema;

architecture ponasanje of tri_podsistema is

function oziceno_I (s_bus: std_ulogic_vector) return std_ulogic is
  begin
     for i in s_bus'range loop
        if s_bus (i) = '0' then
        return '0';
        end if;
     end loop;
   return '1';
   end oziceno_I;

subtype ozicena_I_logika is oziceno_I std_ulogic;

signal zajednicki_izlaz: ozicena_I_logika;


begin

proc1: process
begin
   --
   zajednicki_izlaz: <= '1' after 5 ns;
   --
   end process proc1;

proc2: process
begin
   --
   zajednicki_izlaz: <= '0' after 5 ns;
```

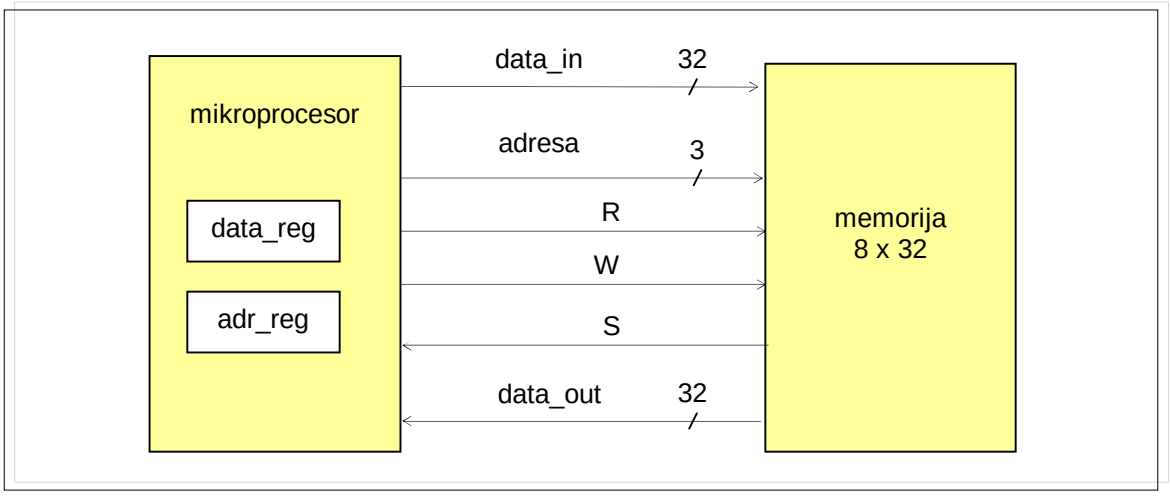
```
    --
    end process  proc2;

proc3: process
begin
    --
    zajednicki_izlaz: <= '1' after 5 ns;
    --
    end process  proc3;

end  ponasanje;
```

2.10 Procedure

Procedure su potprogrami koji mogu da menjaju vrednost jednog ili više parametara. Deklaracija procedure ima sledeći izgled:

```
procedure  ime ( signal  x: in std_logic, signal  y: in std_logic,
                 signal  z: out  std_log, signal  w: out  std_logic );
```

Ako se ne navede klasa parametara, onda se podrazumeva da je klasa ulaznih parametara ***constant*** a klasa izlaznih parametara ***variable***.



Slika 3.7: Primer sistema sa mikroprocesorom i memorijom

Primena procedure ilustrovaće se na primeru sistema koji se sastoji od mikroprocesora i memorije, Slika 3.7. Pretpostavićemo da mikroprocesor može da obavi dve operacije, čitanje iz adresirane memorijske lokacije i upis u adresiranu memorijsku lokaciju. Ove dve operacije realizizovaće dve procedure sa nazivima *citanje* i *upis*. VHDL model mikorprocesora sa operacijama čitanja i upisa ima sledeći izgled.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity  mikroprocesor is
```

```
port ( data_in: out  std_logic_vector (31 downto 0);
         adresa: out  std_logic_vector (2  downto 0);
           R, W : out std_logic;
       data_out: in std_logic_vector (31 downto 0);
              S : in std_logic);
end  mikroprocesor;

architecture ponasanje  of  mikroprocesor is
procedure citanje( signal  adr_reg: in std_logic_vector (2 downto 0);
                   signal  R: out std_logic;
                   signal  S: in std_logic;
                   signal  adresa: out  std_logic_vector (2  downto 0);
                   signal  data_reg: out  std_logic_vector (31  downto 0)) is

begin
   adresa <= adr_reg;
  R <= '1';
   wait until  S = '1';
  data_reg <= data_out;
  R <= '0';
   end   citanje;

procedure upis( signal  adr_reg: in std_logic_vector (2 downto 0);
                   signal  data_reg: in  std_logic_vector (31  downto 0);
                   signal  adresa: out  std_logic_vector (2  downto 0);
                   signal  W: out std_logic;
                        -- signal S je spoljni signal entiteta i vidljiv je iz procedure!
                   signal  data_in: out  std_logic_vector (31  downto 0)) is
begin
   adresa <= adr_reg;
  W <= '1';
   wait until  S = '1';
  data_in  <=  data_reg;
  W <= '0';
   end   upis;

--
-- ovde treba da se stave deklaracije signala
--

begin   -- opis ponasanja mikroprocesora
process
begin
   --
   --
   end process;

process
begin
   --
   --
```

```
    end process;

end ponasanje;
```

Treba napomenuti da signali ne mogu da se deklarišu unutar procedure, ali mogu da se proslede proceduri kao ulazni parametri (tipa ***in***). Signali koje procedura modifikuje su tipa ***out***. Zanimljivo je da procedura može da koristi ili dodeljuje vrednosti signalima koji nisu deklarisani u listi parametara. U navedenom primeru procedura koristi vrednost signala *S* koji je naveden u spoljnjem opisu entiteta *mikroprocesor*. Izmena vrednosti signala koji nisu deklarisani je sporedni efekat *(side effect* izvršavanja procedure i ne preporučuje se jer često dovodi do grešaka koje se teško otkrivaju.

2.11 Konkurentno i sekvencijalno izvršavanje procedura

U zavisnosti od načina pozivanja, procedura može da se izvršava konkurentno ili sekvencijalno. Procedura koja se nalazi u telu opisa arhitekture izvršava se konkurentno kada se promeni vrednost ulaznog signala. U ovom slučaju, u listi parametara ne može da bude promenljiva, jer promenljive mogu da postoje samo unutar procesa.

Konkurentno izvršavanje procedure ilustrovano je na primeru strukturnog modela serijskog sabirača, koji je obrađen u odeljku 2.5. VHDL program je izmenjen tako što je komponenta koja opisuje D flip flop zamenjena procedurom sa istim imenom *(D_flip_flop)*. Procedura se izvršava konkurentno sa komponentom *comb* kada se desi promena u ulaznim signalima (signali *Clk*, *Reset* i *D*, koji su deklarisanim kao ***in***). Promena vrednosti izlaza (signali *Q* i *Q_nad*) događa se na prednjoj ivici signala *Clk*.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity serijski_sabirac is
port  ( a, b, Clk, Reset: in std_logic;
                          suma:  out std_logic);
end  serijski_sabirac;

architecture struktura of  serijski_sabirac is

component   komb
port  ( a, b, C_ul: in std_logic;
          suma, C_iz:  out std_logic);
end  component;

procedure D_flip_flop ( signal D, Clk, Reset : in std_logic;
                        signal Q,, Q_nad:  out std_logic) is

begin
   if  Reset = '0' then
          Q  <= '0'  after 5 ns;
      Q_nad <= '1'  after 5 ns;
   elsif (Clk'event  and Clk = '1') then
          Q  <= D  after 5 ns;
```

```
        Q_nad <= (not D)  after 5 ns;
    end if;
end  D_flip_flop;

signal s1,s2: std_logic;

begin
   C:  komb  port map ( a => a, b  => b,  C_ul => s1,  suma  =>  suma,  C_iz => s2 );
    D_flip_flop ( Clk => Clk  Reset => Reset, D => s2, Q => s1,  Q_nad  => open );
end  struktura;
```

Treba primetiti da se u pozivu procedure formalni parametri ne podrazumevaju na osnovu redosleda u definiciji procedure, nego se formalni parametri zamenjuju stvarnim parametrima na osnovu imena formalnih parametara.

Procedura koja je unutar procesa izvršava se sekvencijalno.

2.12 Pakovanja

Pošto VHDL programi mogu da budu veoma složeni, pogodno je definicije tipa podataka, funkcije i procedure staviti u pakovanja *(packages*) koji mogu da se koriste u različitim VHDL modelima. Deklaracija pakovanja sadrži listu funkcija i procedura koja definiše način njihovog formalnog korišćenja.

Sledeći primer prikazuje deo pakovanja *std_logic_1164.vhd* koji sadrži deklaracije novog tipa podataka i skupa funkcija nad uvedenim tipom podataka.

```
package std_logic_1164  is

type std_ulogic is
        ( 'U',    -- neinicijalizovan
          'X',    -- nepoznat
          '0',    -- logička 0
          '1',    -- logička 1
          'Z',    -- visoka impendansa
          'W',    -- nepoznat, male snage
          'L',    -- logička 0, male snage
          'H',    -- logička 1, male snage
          '-'     -- svejedno
        );

type std_ulogic_vector is array  ( natural range  <>) of std_ulogic;

function  resolved ( s : std_ulogic_vector) return std_ulogic;

subtype  std_logic is  resolved  std_ulogic ;

type std_logic_vector is array  ( natural range  <>) of std_logic;
```

```
function  "and" (l,rb : std_logic) return std_logic;    -- operator overloading!
function  "and" (l,rb : std_ulogic) return std_ulogic;

-- i tako dalje...

end  std_logic_1164  ;
```

Tip podataka '***natural***' je celobrojni pozitivan broj, koji je ograničen. Oznaka '<>' omogućava promenljivu vrednost indeksa u nizu, odnosno računanje gornje vrednosti indeksa u trenutku kreiranja objekta navedenog tipa. Kod poziva funkcije, proverava se opseg stvarnog parametra i gornja granica koristi umesto onake '<>'.

Pored deklaracije, pakovanje mora da sadrži i implementaciju funkcija i procedura. Telo pakovanja ima sledeći izgled.

```
package body novo_ pakovanje is
--
-- definicija tipova podataka, funkcija i procedura
--
end  std_logic_1164  ;
```

Pakovanja se prevode i smeštaju u biblioteke, a da bi se koristile ime pakovanja navodi se iza ključne reči ***use***.