Fakultet elektrotehnike i računarstva
Zavod za elektroniku, mikroelektroniku,
računalne i inteligentne sustave

DIGITALNA LOGIKA

UPUTA ZA ČETVRTU LABORATORIJSKU VJEŽBU

Marko Čupić

Zagreb, 2008.

1. Zadatak

Ova laboratorijska vježba služi za upoznavanje s jezikom za specifikaciju sklopova VHDL. Na laboratorijskim vježbama koristi se razvojno okruženje WebISE, besplatna verzija profesionalnih alata koje razvija i prodaje firma Xilinx. To okruženje namijenjeno je razvoju sklopova temeljenih na porodicama sklopova FPGA i CPLD. Kako bi se izbjegla prevelika količina materijala unutar jedne pripreme VHDL nije opisan detaljno, već se potrebni dijelovi mogu pronaći u [1, 2].

Projektiranje nekog sklopa sastoji se od niza koraka od kojih je svaki idući korak na nižoj razini apstrakcije, tj. bliži je samom sklopovlju. U nastavku je dan redoslijed po kojemu se obavlja projektiranje, a masnim slovima označeni su pojmovi specifični za modeliranje sklopova:

1. *Opis funkcionalnosti sklopa*

   U ovom koraku opisuje se funkcionalnost sklopa koju želimo postići. Sklop je moguće opisati na dva načina. Prvi način – sličan programiranju, tj. izradi programa – opisivanje je ponašanja sklopa. U tom slučaju govorimo o **funkcijskom (**odnosno **ponašajnom) modelu**. Druga mogućnost je da opišemo sastavne elemente od kojih se sklop sastoji i način međusobnog povezivanja elemenata (što smo radili na drugoj laboratorijskoj vježbi). U ovom slučaju govorimo o **strukturnom modelu**. Elementi koje koristimo prilikom strukturnog opisivanja sklopa nazivaju se komponente. Komponente su tipično sklopovi nešto jednostavnije funkcionalnosti čijim se povezivanjem postiže tražena funkcionalnost sklopa koji razvijamo. Komponente su također opisane ili ponašajno ili strukturno. U slučaju strukturnog modela bitno je uočiti razliku između **komponente** i **primjerka** (**instance) komponente**. U nekom strukturnom dizajnu koristimo komponente (procesore, ALU, itd.), a kada upotrijebimo komponentu i povežemo je sa ostatkom dizajna tada govorimo o **primjerku komponente**. Evo jednostavne ilustracije na primjeru računala. Ako govorimo o procesoru AMD Athlon 64 X2 6000+, 3000 Mhz, govorimo o komponenti. No, na matičnoj ploči tih procesora može biti jedan ili više. Ako imamo više takvih procesora, svaki je spojen na druge vodiče na matičnoj ploči, pa govorimo o više primjeraka te komponente. Ta razlika slična je razlici između razreda i objekta u objektno orijentiranom programiranju.

2. *Simulacija sklopa*

   Nakon što je sklop opisan potrebno je provjeriti ispravnost opisa što se obavlja simulacijom. Ulaz simulatora je niz ispitnih uzoraka koje simulator postavlja na ulaze sklopa koji se simulira te se računa odziv, tj. izlaz. Na osnovu izlaza može se zaključiti je li sklop ispravno opisan. Često je osim ispitnih uzoraka potrebno napraviti i cjelokupnu ispitnu okolinu. Npr. ako želimo testirati procesor potrebno mu je dodati RAM i ROM kako bi se on mogao testirati, te u tom slučaju govorimo o ispitnoj okolini koja se sastoji od RAM-a i ROM-a i ispitivane komponente – procesora. Okoline za razvoj sklopova uvijek omogućavaju izvođenje više vrsta simulacija. Najjednostavnija je *simulacija ponašajnog modela* (Behavioral model simulation), kod koje se simulira isključivo logički rad sklopa, i gdje se pretpostavlja da svi korišteni elementi rade beskonačno brzo. Najsloženija je *simulacija nakon postavljanja i povezivanja* u ciljnu tehnologiju (Post place and route simulation) gdje se u obzir uzimaju svojstva realnih sklopova koji se koriste kako bi se ostvario opisani sklop (primjerice, koliko iznosi kašnjenje logičkih sklopova i sl).

3. *Implementacija sklopa u ciljnoj tehnologiji*

   U ovom koraku opisani sklop prevodi se u željenu tehnologiju (FPGA, CPLD, ASIC, itd.). To se često izvodi automatizirano budući da se radi o dosta kompleksnom koraku. Podaci ovog koraka mogu se koristiti u prethodnom koraku kako bi se poboljšala točnost simulacije. Recimo, u ovom

koraku saznajemo kašnjenja na sklopu koja se u koraku 2. nisu uzimala u obzir, ali koja mogu utjecati na dizajn.

4. *Izrada prototipa*

   Usprkos simulaciji još nije sigurno hoće li sklop nakon proizvodnje ispravno raditi. Naime, prilikom simulacije nikada nije moguće uzeti baš sve u obzir. Stoga je potrebno proizvesti probne primjerke (prototipove) i testirati ih kako bi se utvrdila njihova ispravnost. U slučaju FPGA i CPLD uređaja (vrste programirljivih uređaja koje se obrađuju na kolegiju Digitalna logika) izrada probnih primjeraka svodi se na programiranje sklopova, dok se u slučaju ASIC-a tvornici šalju podaci dobiveni u koraku 3 na osnovi kojih tvornica izrađuje čipove.

Iako su prethodni koraci predstavljeni slijedno, u stvarnosti se radi o *iterativnom* procesu. Ove laboratorijske vježbe pokrivaju prva dva koraka, modeliranje sklopa i simulaciju. Za modeliranje se upotrebljava jezik VHDL (eng. VHSIC Hardware Description Language, VHSIC – Very High Speed Integrated Circuit).

Ova vježba sastoji se od dvije inačice, od kojih svaki student rješava samo jednu.

Odabir inačice

Studenti čija je suma *predzadnje dvije* znamenke JMBAG-a parna (uključuje i 0) rade inačicu 1; studenti čija je suma *predzadnje dvije* znamenka JMBAG-a neparna rade inačicu 2. Dakle, ako je JMBAG=ABCDEFG**HI**J, razmatra se parnost sume H+I. Primjer:

JMBAG	***Inačica***
0036123**45**6	Inačica 2
0036234**46**7	Inačica 1
0036345**67**8	Inačica 2
0036678**00**1	Inačica 1

Inačica 1

U [2] počev od stranice 167 nalazi se nekoliko VHDL modela multipleksora 2/1. Jedan od tih modela je uz malu modifikaciju prepisan u nastavku (dodan je ulaz za omogućavanje te kašnjenje izlaza).

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

ENTITY mux21 IS PORT (
       E: IN Std_Logic;
       S: IN Std_Logic;
       D0: IN Std_Logic;
       D1: IN Std_Logic;
       Y: OUT Std_Logic);
end mux21;

ARCHITECTURE arch OF mux21 IS

BEGIN
  PROCESS (E, S, D0, D1)
  BEGIN
    IF E='1' THEN
      CASE S IS
        WHEN '0' => Y <= D0 AFTER 10 ns;
        WHEN '1' => Y <= D1 AFTER 10 ns;
        WHEN OTHERS => Y <= '0' AFTER 10 ns;
      END CASE;
    ELSE
      Y <= '0' AFTER 10 ns;
    END IF;
  END PROCESS;

END arch;
```

Prepišite ovaj model, napravite ispitnu okolinu i ispitajte njegov rad izvođenjem *simulacije ponašajnog modela*. Kako ovaj sklop ima 4 ulaza (E, S, D0 te D1), obavite ispitivanje za sve kombinacije ulaza (0000, 0001, ..., 1111 upravo navedenim radosljedom varijabli; svaku kombinaciju držite na ulazu 100 ns). Utvrdite koliko iznosi kašnjenje ulaza D0. Kako ćete to utvrditi?

Uporabom opisanog multipleksora 2/1 napišite strukturni model multipleksora 16/1, realiziranog kao multipleksorsko stablo. Sučelje tog sklopa treba sadržavati signale *D* (std_logic_vector), *S* (std_logic_vector), *E* (std_logic) te *f* (std_logic), tim poretkom. Svi vektori neka budu oblika (0 TO *n*), gdje je *n* gornja granica. Ispitajte rad tog sklopa na nekoliko ispitnih uzoraka. Prilikom provjere rezultata simulacije obratite pažnju na način imenovanja signala! Možete li objasniti od kojih se dijelova sastoji ime signala?

Konačno, uporabom opisanog multipleksora 16/1 napišite strukturni model sklopa koji ostvaruje funkciju f(A,B,C,D)=f_n, te ispitajte rad tog sklopa za sve kombinacije ulaza (od 0000 do 1111;

svaku kombinaciju zadržite 100 ns). Sučelje ovog sklopa treba sadržavati samo 5 signala: *A*, *B*, *C*, *D* te *f*, tim poretkom. Točna definicija funkcije f_n zadana je na kraju ovog dokumenta i ovisi o Vašem JMBAG-u.

Inačica 2

U [2] na stranicama 169-172 nalazi se nekoliko VHDL modela dekodera. Model dekodera 1/2 s ulazom za omogućavanje je uz malu modifikaciju prepisan u nastavku (dodano je kašnjenje izlaza).

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

ENTITY dek12e IS PORT (
       e : IN Std_Logic;
       a : IN Std_Logic;
       y0 : OUT Std_Logic;
       y1 : OUT Std_Logic);
end dek12e;

ARCHITECTURE arch OF dek12e IS

BEGIN

  y0 <= E AND NOT A AFTER 10 ns;
  y1 <= E AND A AFTER 10 ns;

END arch;
```

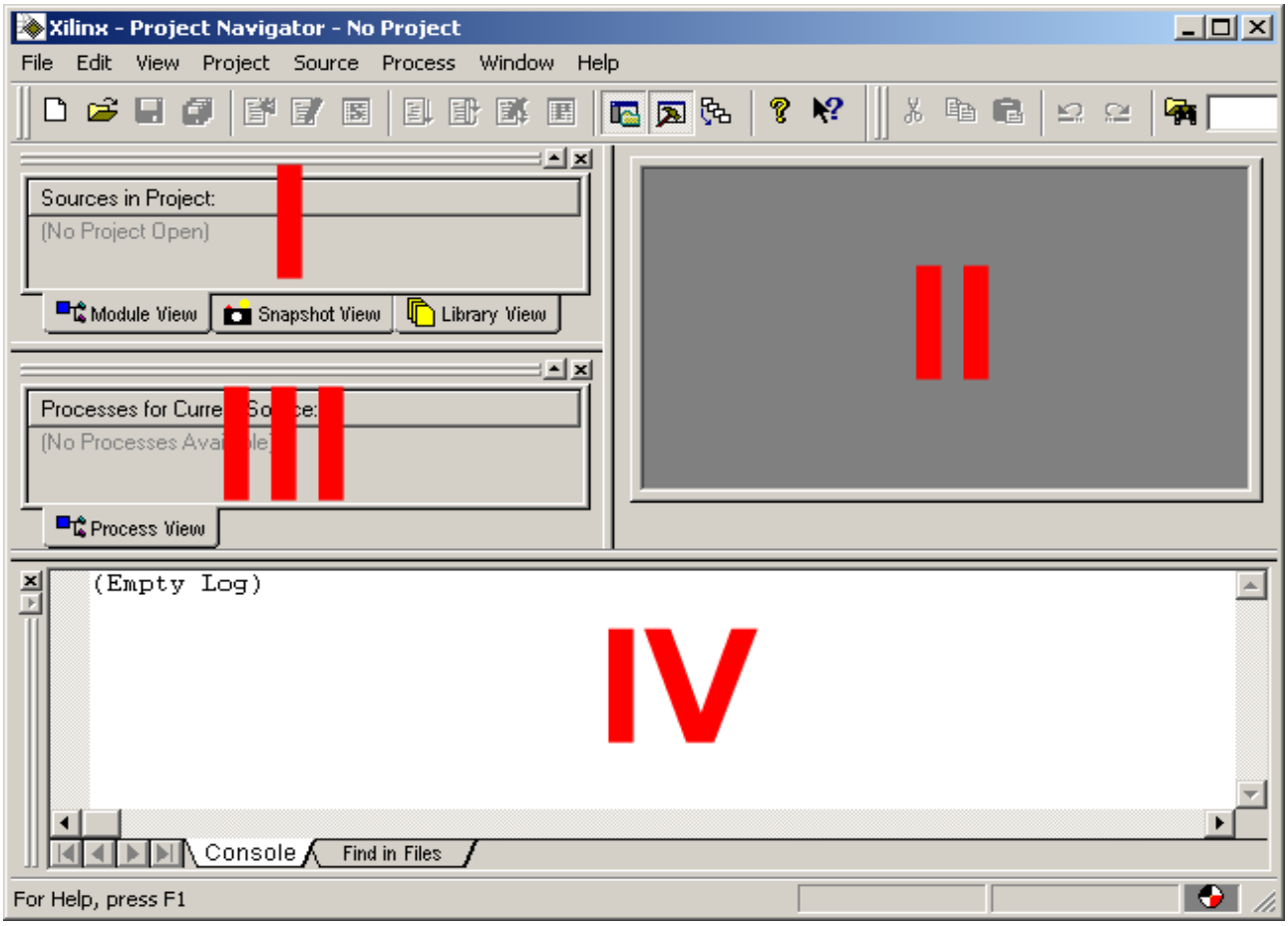
Prepišite ovaj model, napravite ispitnu okolinu i ispitajte njegov rad izvođenjem *simulacije ponašajnog modela*. Kako ovaj sklop ima 2 ulaza (E, A), obavite ispitivanje za sve kombinacije ulaza (00, 01, ..., 11 upravo navedenim redosljedom varijabli; svaku kombinaciju držite na ulazu 100 ns). Utvrdite koliko iznosi kašnjenje ulaza A. Kako ćete to utvrditi?

Uporabom opisanog dekodera 1/2 napišite strukturni model dekodera 4/16 s ulazom za omogućavanje, realiziranog kao dekodersko stablo. Sučelje tog sklopa treba sadržavati signale *E* (std_logic), *A* (std_logic_vector) te *Y* (std_logic_vector), tim poretkom. Svi vektori neka budu oblika (0 TO *n*), gdje je *n* gornja granica. Ispitajte rad tog sklopa na nekoliko ispitnih uzoraka. Prilikom provjere rezultata simulacije obratite pažnju na način imenovanja signala! Možete li objasniti od kojih se dijelova sastoji ime signala?

Konačno, uporabom opisanog dekodera 4/16 napišite strukturni model sklopa koji ostvaruje funkciju f(A,B,C,D)=f_n, te ispitajte rad tog sklopa za sve kombinacije ulaza (od 0000 do 1111; svaku kombinaciju držite na ulazu 100 ns). Sučelje ovog sklopa treba sadržavati samo 5 signala: *A*, *B*, *C*, *D* te *f*, tim poretkom. Točna definicija funkcije f_n zadana je na kraju ovog dokumenta i ovisi o Vašem JMBAG-u.

2. Upute za obavljanje zadatka

Pokrenite program Xilinx WebISE (odnosno Project Navigator). U nastavku ove upute prikazani su primjeri programa inačice 5. Ako se na laboratorijskim vježbama koristi neka druga inačica, moguće su manje razlike. Nakon pokretanja programa pojavljuje se radna površina programa čiji je izgled prikazan na slici 1.



Slika 1. Radna površina WebISE alata

Radna površina sastoji se od 4 dijela označena na slici rimskim brojevima. U nastavku je kratki opis svakog od tih dijelova:

I. *Sources in Project* prikazuje sve datoteke koje čine cjelokupni projekt/dizajn.

II. Prozor u kojemu se otvaraju prozori s VHDL kodom modela, izvještajima iz pojedinih faza dizajna, itd.

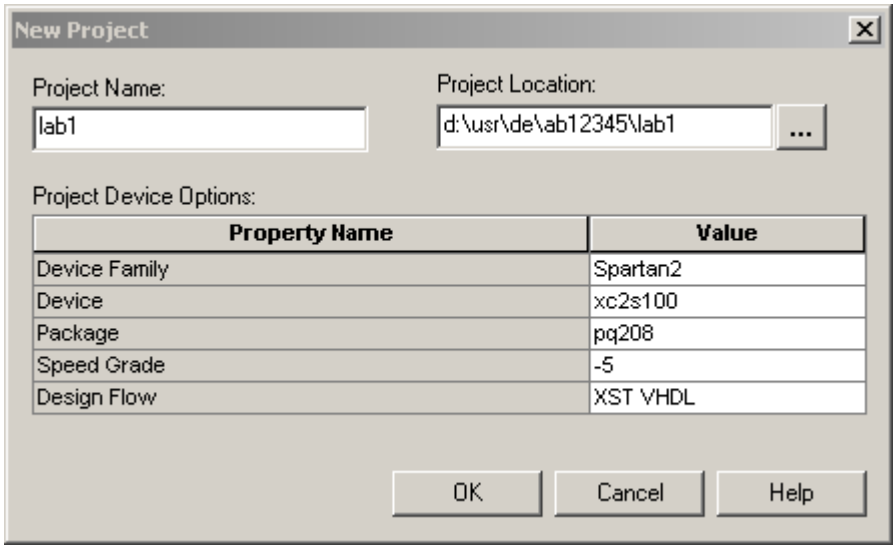
III. *Processes for Current Source* – prozor koji prikazuje trenutno raspoložive operacije. Sadržaj tog prozora ovisi o odabranoj datoteci u prozoru I. Odabiranje neke od opcija u ovom prozoru obavlja se dvostrukim klikom lijeve tipke miša.

IV. Statusni prozor (*Console, Find in Files*) u kojemu se ispisuju naredbe koje se pokreću tijekom rada programa te prikazuju njihovi izlazi. Primjerice, prilikom sintaksne provjere VHDL modela u ovom se prozoru ispisuju eventualne poruke o pogreškama.

Odabir opcije unutar prozora s raspoloživim procesima (prozor III) uglavnom uzrokuje pokretanje nekog procesa, pregledavanje rezultata nekog prethodno obavljenog procesa ili pokretanje nekog vanjskog programa. Nadalje, procesi su međusobno ovisni i ta ovisnost se proteže naniže. Primjerice, da bi se pristupilo implementaciji potrebno je prvo obaviti sintezu. U slučaju preskakanja pojedinih koraka program automatski pokreće sve prethodne procese o kojima ovisi odabrani proces. Rezultati izvršavanja pojedinih procesa dani su simbolički s lijeve strane naziva samog procesa. Mogu se pojaviti tri simbola. Zelena kvačica označava da je proces u potpunosti uspio. Žuti uskličnik označava uspjeh ali upozorava na određene – nekritične – probleme. Konačno, crveni križić označava da je došlo do pogreške koja je onemogućila nastavak izvršavanja procesa. Detalji o rezultatima izvršavanja svih procesa mogu se naći u odgovarajućim izvještajima.

2.1. Otvaranje novog projekta

Svaki dizajn sastoji se od jedne ili više datoteka koje su grupirane u *projekt*. Same datoteke mogu sadržavati dokumentaciju, modele sklopa, ispitne uzorke, itd. Prilikom izrade novog dizajna prvo je potrebno stvoriti novi projekt. Novi se projekt započinje odabirom opcije **`File → New Project...`** nakon čega se otvara dijalog koji je prikazan na slici 2. U taj dijalog upisuju se temeljni podaci o projektu i to naziv projekta, direktorij na disku u koji se smještaju sve datoteke projekta, arhitektura FPGA/CPLD na kojoj se bazira projekt, tj. u kojoj će projekt biti izveden, te način opisa samog dizajna. Opis dizajna može biti u jezicima VHDL, Verilog ili EDIF. Na ovim je laboratorijskim vježbama ciljni FPGA sklop Spartan2, a jezik u kojemu se opisuje dizajn je VHDL. Ime projekta je proizvoljno, a sam projekt potrebno je smjestiti u direktorij koji će asistent reći tijekom same vježbe.



Slika 2. *Dijalog za kreiranje novog projekta*

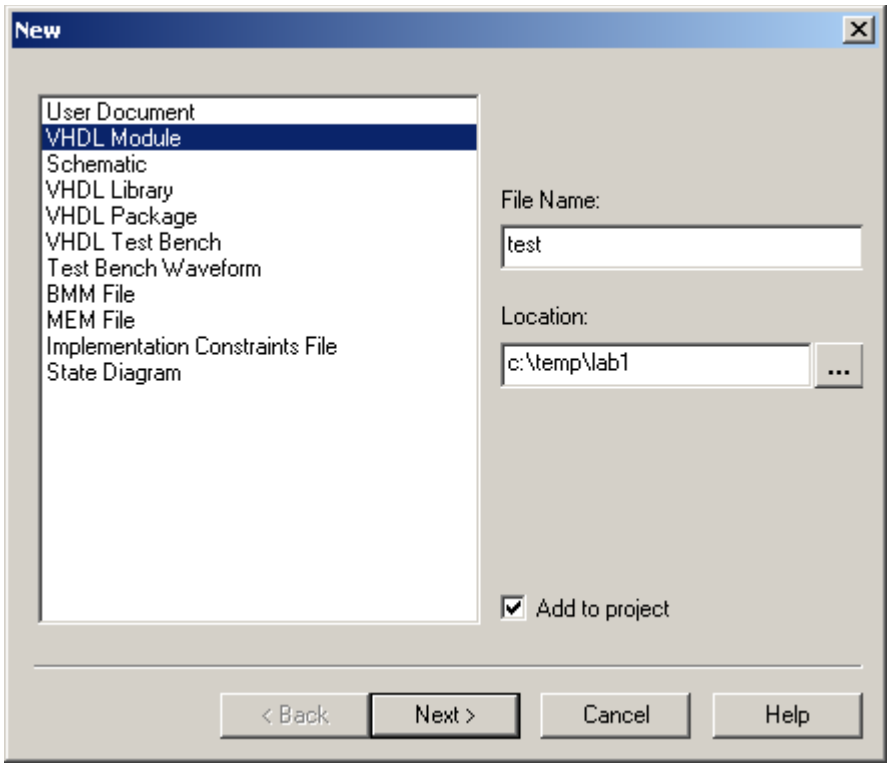
Nakon upisa navedenih podataka u integriranoj okolini otvoren je novi projekt koji u tom trenutku ne sadrži nikakve dodatne elemente.

U slučaju otvaranja već postojećeg projekta bira se opcija **`File → Open Project...`** nakon čega se dobija standardni Windows dijalog za otvaranje datoteka. Datoteka u kojoj se nalazi definirani projekt ima ekstenziju `npl`, a ime joj je identično imenu projekta.

2.2. Dodavanje VHDL modela projektu

Nakon što je projekt otvoren potrebno je dodati model sklopa. To se čini odabirom opcije **`Project → New Source...`**. čime se otvara dijalog koji traži da se dodatno definiraju podaci o datoteci/modulu koji želimo dodati. Dodatni podaci su tip datoteke, odnosno njen sadržaj, naziv datoteke i mjesto na disku gdje će datoteka biti pohranjena. Na slici 1.3. prikazan je dijalog.

U ovom slučaju želimo dodati novi VHDL model, pa je prema tome potrebno odabrati opciju `VHDL Module`, a u polje **`File Name:`** potrebno je upisati naziv datoteke bez ikakve ekstenzije. Za naziv datoteke obično se uzima ime modela (u ovim uputama to je `test`, dok ćete Vi navesti ime modela koji opisujete). Nakon upisa tih podataka treba odabrati tipku **`Next`** dva puta, a potom tipku **`Finish`**. U tom trenutku projektu je dodan novi modul, koji je istovremeno prikazan u novootvorenom prozoru. Sadržaj tog modula čini kostur VHDL modela koji je potrebno dopuniti željenim modelom. Stoga upišite VHDL opis sklopa koji radite.



Slika 3. Dijalog za definiranje novog VHDL modula u projektu

Prilikom upisa VHDL modela vrlo je korisna opcija uz pomoć koje se provjerava ispravnost sintakse upisanog VHDL modela. Ta opcija se dobija tako da se u prozoru *Sources in Project* odabere željeni modul, te se potom u prozoru *Processes for Current Source* odabere opcija **`Synthesize → Check syntax`**. Tijek provjere može se pratiti u najnižem, statusnom prozoru. Linije u kojima se korisnik obavještava o postojanju greške započinju sa rječju `Error`. U toj liniji opisana je greška i (približno) mjesto na kojoj je ona otkrivena. Na grešku u izvornom kodu moguće je otići tako da se dva puta klikne na samu liniju (ne na crveni znak sa strane!). Tijekom provjere ispisuju se i upozorenja za koja vrijedi isto pravilo kao i za greške, tj. dvostrukim klikom na liniju prelazi se u kod i to upravo na mjesto gdje je upozorenje generirano. Linija koja sadrži upozorenje počinje rječju *Warning*! Provjera sintakse ne pokreće simulaciju!

2.3. Jezik VHDL

VHDL je jezik za opis sklopovlja. Razvoj VHDL-a potaknulo je i financiralo američko Ministarstvo obrane tijekom 80-tih godina prošlog stoljeća. Jezik se temelji na Ada programskom jeziku. VHDL

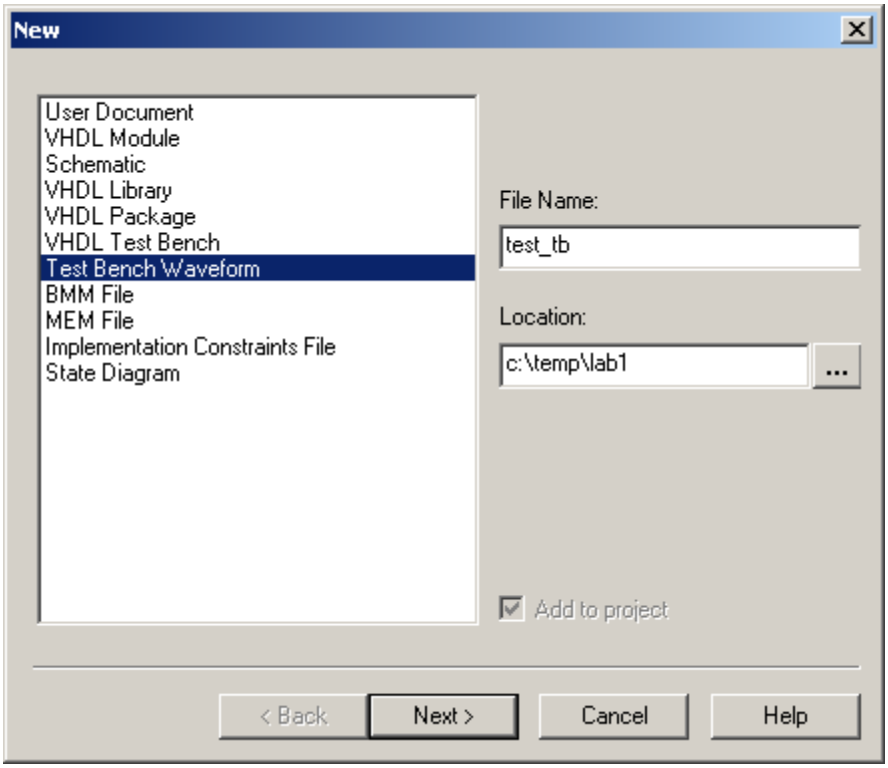
jezik formom podsjeća na Pascal. Sam VHDL dosta je kompleksan i nudi mnoštvo mogućnosti, ali se u okviru predmeta Digitalna logika obrađuje samo jedan njegov manji podskup.

VHDL je strogo tipiziran, te se ne razlikuju mala i velika slova. Komentari započinju znakovima – (dvije crtice) i protežu se do kraja linije. Svaki VHDL model – tj. sklop – sastoji se od točno jednog bloka ENTITY te od jednog ili više blokova ARCHITECTURE. Unutar bloka ENTITY opisuje se *sučelje* sklopa, dok se u bloku ARCHITECTURE opisuje *ponašanje* ili *struktura* sklopa.

Iako je VHDL dosta sličan "uobičajenim" programskim jezicima postoje i neke suštinske razlike specifične za jezike za opis sklopovlja. Prva od njih je *paralelno* izvršavanje svih izraza u VHDL-u. To znači da ako se u ARCHITECTURE dijelu između BEGIN i END upišu dva izraza **nije definiran** njihov redoslijed izvršavanja! Kako bi se postiglo slijedno izvođenje koristi se blok PROCESS. Sve što je unutar jednog bloka PROCESS izvršava se redoslijedom kojim je napisano. Druga karakteristika specifična za VHDL je postojanje tzv. **signala**. Signali su dosta slični varijablama ali ponešto se razlikuju u ponašanju; primjerice signali mogu imati kašnjenja, tj. nakon dodijeljivanja vrijednosti signalu on tu vrijednost poprima nakon određenog vremena za razliku od varijabli koje vrijednosti poprimaju odmah! Kako se načelno obavlja simulacija sklopova opisanih jezikom VHDL potrebno je proučiti u [2], zadaci 16.1 do 16.4, stranice 481 do 499.

2.4. Izrada simulacijskih ispitnih uzoraka

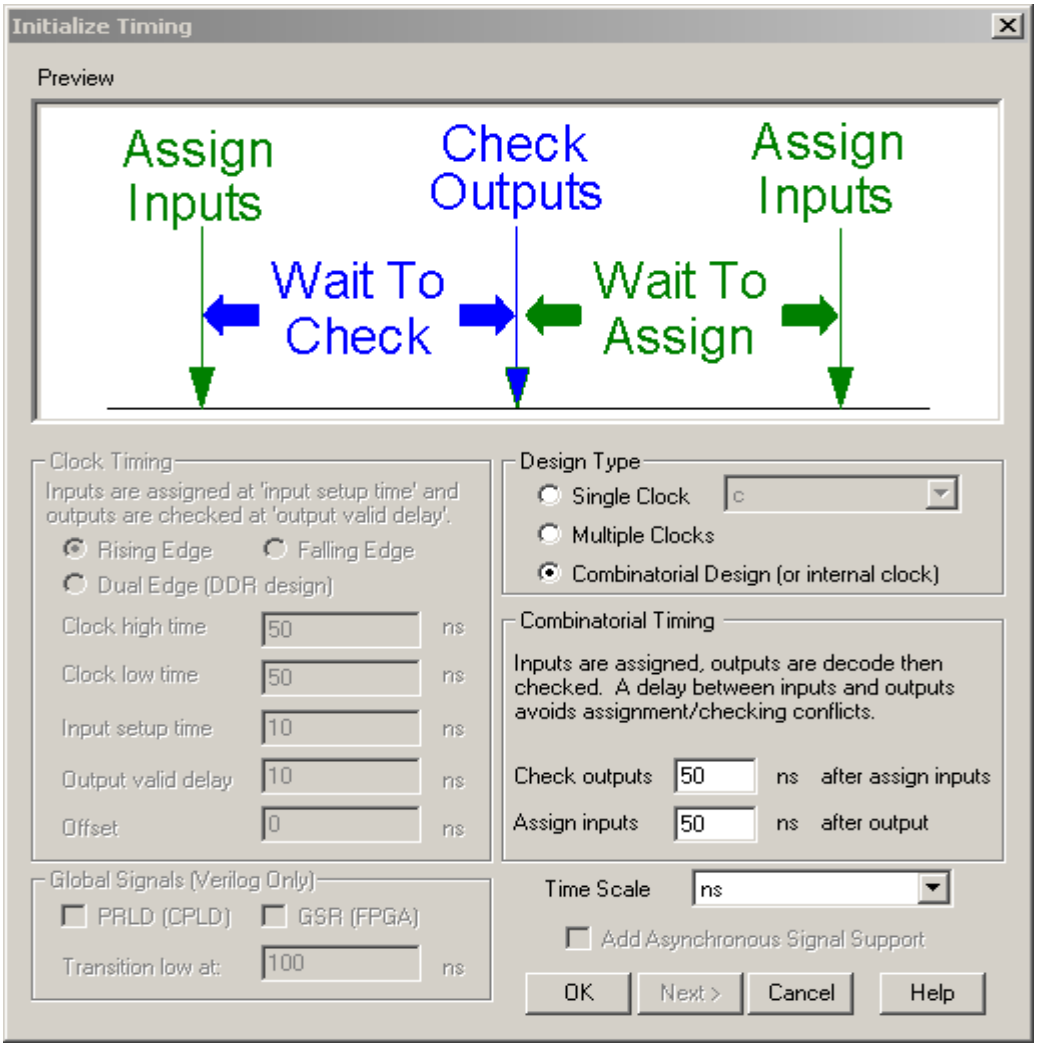
Nakon upisivanja opisa modela, idući korak je generiranje ispitnih uzoraka (odnosno, stvaranje ispitne okoline) kako bi se provjerila ispravnost upisanog modela. To se obavlja upotrebom programa **HDL Bencher**.



Slika 4. *Dodavanje modula s testnim valnim oblicima*

Generiranje ispitnih uzoraka, tj. ispitne okoline, započinje dodavanjem novog modula u projekt, pa je prema tome potrebno odabrati opciju **`Project → New Source...`**. U odnosu na prethodni slučaj, sada je prilikom određivanja vrste modula potrebno odabrati opciju **`Test Bench Waveform`**, a kao naziv modula preporučljivo je dati identično ime kao i ispitivanom modulu uz dodatak `_tb` (od **t**est **b**ench). Za dani VHDL primjer naziv test modula bio bi **test_tb** (slika 4).

Nakon odabira opcije **`Next`**, pojavljuje se drugi dijalog unutar kojega se odabire modul koji će se ispitivati, odnosno za koji je potrebno izraditi ispitnu okolinu. U našem slučaju postoji samo jedan VHDL modul pa se može ponovno odabrati tipka **`Next`** nakon čega se dobija sažetak odabranih vrijednosti. Da bi se završio proces dodavanja novog modula potrebno je odabrati tipku **`Finish`**. Odmah nakon toga automatski se pokreće HDL Bencher te se pojavljuje dijalog uz pomoć kojega se definiraju vrijednosti vremenskog takta te dodatnih parametara kao što trajanje visokog i niskog nivoa, vremena postavljanja i maksimalna vremena kašnjenja. Dijalog je prikazan na slici 5. Ponuđene vrijednosti su u ovom slučaju prihvatljive te ih je samo potrebno potvrditi odabirom tipke **`OK`**. Nakon potvrde vrijednosti takta vremenskog vođenja otvaraju se dva prozora. U gornjem se definiraju valni oblici ulaznih signala, dok se u donjem prozoru nalazi VHDL model sklopa koji se ispituje.



Slika 5. *Dijalog za definiranje vremenskih parametara simulacije sklopa*

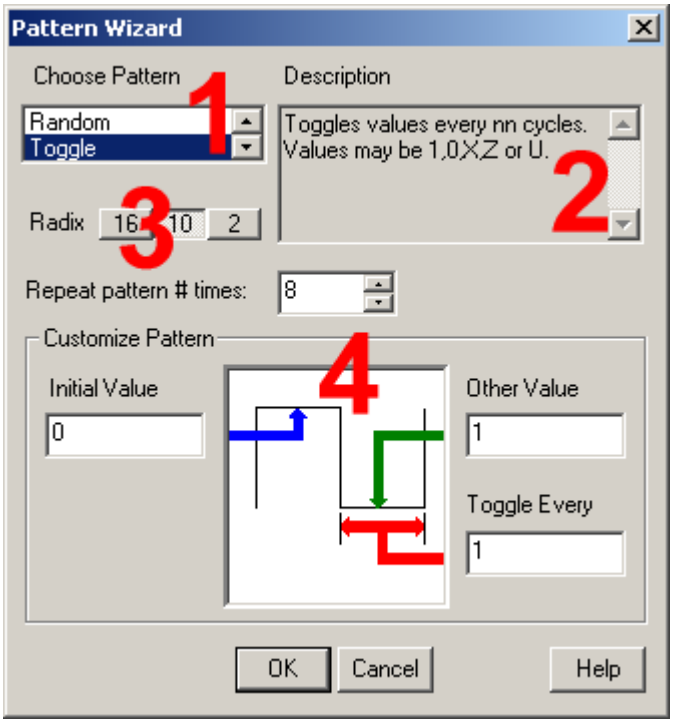
Promjena vrijednosti jednobitnih signala obavlja se tako da se pokazivač (miš) postavi na vrijednost signala (tj. na liniju koja predstavlja vrijednost signala) te se potom jednim klikom na lijevu tipku miša mijenja vrijednost u suprotnu – ako je bila 0 tada postaje 1 i obratno, ako je vrijednost bila 1 mijenja se u 0. Ta promjena vrijedi od odabranog trenutka pa do kraja simulacije, a moguće ju je identičnim postupkom na dijagramu promijeniti u nekom kasnijem trenutku.

Ako je potrebno u određenom trenutku nekom višebitnom signalu upisati vrijednost to se čini dvostrukim klikom pri čemu pokazivač miša mora pokazivati na ciklus unutar kojega želimo provesti promjenu, tj. na mjesto na signalu na kojemu želimo promjenu. Nakon dvostrukog klika pojavljuje se polje za upis vrijednosti i tipka odmah pored tog polja s nazivom «`Pattern`». U polje

je moguće upisati vrijednost čija baza ovisi o odabranoj vrijednosti u traci sa alatima. Upisana vrijednost proširuje se na desno i vrijedi u budućnosti dok se eksplicitno ne promjeni na nekom drugom mjestu.

Osim jednostavnih promjena valnih signala, te upisa konkretnih vrijednosti, moguće je definirati specifične promjene samog signala. Dijalog unutar kojega se definiraju specifični valni oblici dobiva se dvostrukim klikom na signal na kojemu se želi definirati neki niz; nakon pojave polja za upis vrijednosti potrebno je kliknuti na tipku Pattern. Sam dijalog prikazan je na slici 6 i sastoji se od sljedećih dijelova:

1. odabir vrste valnog oblika koji se želi definirati,
2. opis odabranog valnog oblika u polju 1,
3. baza za upisane vrijednosti (vrijedi samo za višebitne signale) te
4. parametri odabranog uzorka; sadržaj tog polja ovisi o odabranom valnom obliku.



Slika 6. *Dijalog za definiranje parametara valnih oblika signala*

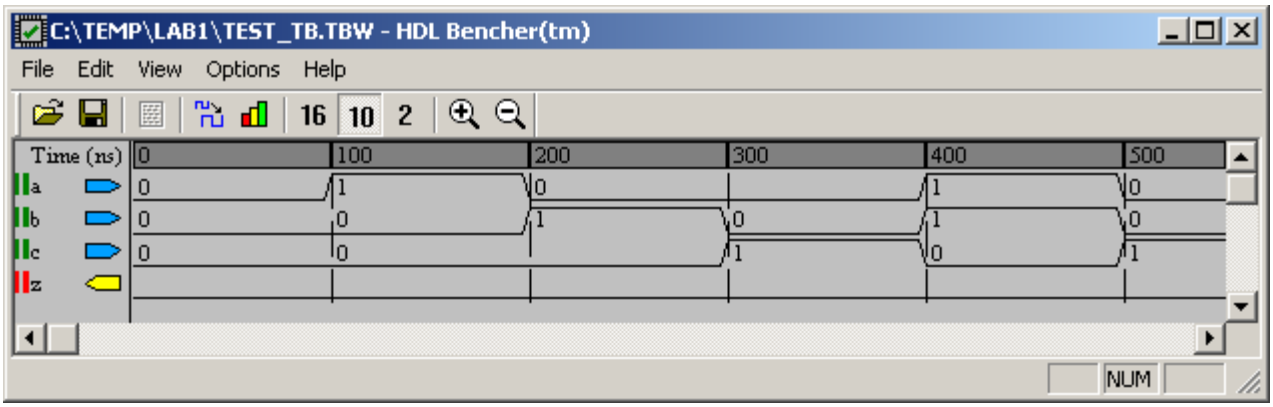
Novootvoreni dijalog omogućava definiranje sljedećih valnih oblika za jednobitne signale:

1. impuls,
2. slučajna vrijednost signala te
3. stalna promjena signala (engl. toggle).

Za višebitne signale moguće je definiranje sljedećih valnih oblika:

1. alterniranje između dvije vrijednosti,
2. brojanje unaprijed/unatrag,
3. slučajna vrijednost signala te
4. posmak vrijednosti nalijevo/nadesno.

Na slici 7 prikazan je primjer valnih oblika za sklop test. Vi ćete na ovom mjestu definirati valne oblike za sklop koji ispitujete.



Slika 7. Valni oblici za simulaciju zadanog VHDL modela sklopa

Prilikom napuštanja `HDL Bencher`-a, u slučaju da to nije prethodno definirano, program traži odabir broja ciklusa takta vremenskog vođenja kroz koji će se obaviti simulacija.

Obavezno zatvorite HDL Bencher prije povratka u integrirano razvojno okruženje! Povratkom u integrirano okruženje moguće je vidjeti generirani VHDL modul koji se koristi za testiranje. To se postiže odabirom modula s valnim oblikom u prozoru *Sources in Project*, nakon čega se u prozoru *Processes for Current Source* odabere opcija **`View Behavioral Testbench`**. Treba primjetiti da je taj model mješavina strukturnog i ponašajnog VHDL-a, tj. u njemu se koriste komponente, ali se također opisuje ponašanje sklopa.

2.5. Simulacija VHDL modela

Nakon što je upisan VHDL model i definirana testna okolina pristupa se simulaciji tijekom koje se provjerava je li model ispravan. Simulacija se pokreće odabirom ispitnog modula unutar prozora *Sources in Project* – modula načinjenog tijekom prethodnog koraka – te potom odabirom opcije *Simulate Behavioral VHDL Model* u prozoru *Processes for Current Source*. Time se pokreće program **ModelSim** koji otvara sljedeća četiri prozora – u zagradama su navedena imena prozora:

1. Glavni prozor unutar kojega se zadaju naredbe programu (`ModelSim XE/Starter 5.6a - Custom Xilinx Version`),
2. prozor unutar kojega su navedeni signali u modelu čija vrijednost se promatra tijekom simulacije (`signals`),
3. prozor sa strukturom cijelog dizajna (`structure`) i

4. prozor s prikazanim valnim oblicima tijekom simulacije (`wave - default`).

ModelSim odmah pokreće simulaciju, a rezultati su prikazani u prozoru s valnim oblicima simulacije. Nakon obavljene simulacije može se odabrati prozor s valnim oblicima kako bi se analizom dobivenih rezultata odredilo da li je model ispravan. Početno je povećanje u tom prozoru dosta veliko te ga je potrebno smanjiti kako bi se vidjeli valni oblici. Radi lakšeg snalaženja po valnim oblicima mogu se koristiti opcije zumiranja, jedan ili više pokazivača – koji su u stvari okomite crte, te još neke opcije u traci s alatima kao što je pronalaženje bridova odabranog signala.

Nakon analize valnih oblika potrebno je zatvoriti ModelSim te se time vraća se u WebISE okruženje gdje se dalje postupa u ovisnosti o rezultatima analize valnih oblika. Ako je analizom utvrđeno da se sklop ponaša ispravno može se pristupiti sintezi, a u suprotnom, ako nisu dobri valni oblici, potrebno je odrediti grešku u modelu, ispraviti model te ponovno pokrenuti simulaciju.

2.6. Mjerenje vremenskih intervala

ModelSim omogućava jednostavno mjerenje trajanja intervala upotrebom dva pokazivača (okomite crte). U ovom odjeljku pojašnjen je način na koji se to izvodi. Svi potrebni alati koji se koriste u mjerenju dostupni su preko izbornika.

Inicijalno je u prozoru s vremenskim signalima aktivan samo jedan pokazivač (vertikalna crta) te se upotrebom miša taj pokazivač može pozicionirati na proizvoljno mjesto u vremenskom dijagramu. Dodavanje novog pokazivača obavlja se odabirom opcije **`Insert → Cursor`**. U tom trenutku pojavljuje se novi pokazivač koji postaje aktivan (puna debela crta) dok se prethodni pokazivač deaktivira (isprekidana tanka crta). Na dnu se nalazi ispisana razlika u vremenu između dva pokazivača.

Radi olakšanog pozicioniranja pokazivača na prijelaze nekog signala (recimo prijelaz iz 0 u 1) postoje u traci s alatima opcije *Find Next Transition* i *Find Previous Transition*. Postupak kojim se pokazivač postavlja na prijelaz je sljedeći:

1. Pokazivač se postavi neposredno prije željenog prijelaza

2. Klikne se na ikonu *Find Next Transition* u traci sa alatima

2.7. Pregledavanje internih signala

Osim ulaznih i izlaznih signala moguće je vidjeti i sve interne signale u dizajnu. Interni signali su svi signali koji se ne pojavljuju u bloku ENTITY. Kako bi se u prozor s valnim oblicima dodali svi signali u dizajnu potrebno je u prozoru *signals* odabrati opciju **`Add → Wave → Signals in Design`**, nakon čeka se u prozoru s valnim oblicima pojavljuju svi signali u dizajnu. Uklanjanje nekog signala je jednostavno i svodi se na njegovo selektiranje te pritisak na tipku Del. U slučaju da su signali dodani nakon završene simulacije njihova vrijednost nije zapamćena te je potrebno ponovno pokrenuti simulaciju. Postupak je sljedeći: u glavnom prozoru simulatora (ModelSim XE) se otipka naredba `restart`, te se potom u dijalogu koji se pojavi pritisne OK. Nakon toga se izvrši naredba `run -all`.

Imena signala daju se na osnovi komponenti unutar kojih se nalaze. Tako npr. signal `reset` koji je definiran unutar primjerka komponente `controller` ima puno ime `controller/reset`. Na izlaznom testu (na kraju 4. vježbe) može se pojaviti pitanje u vezi imena primjerka koju je dobio ispitivani VHDL model, te je potrebno obratiti pažnju na ime!

2.8. Sinteza

Nakon što je napisan ispravan model, pristupa se sintezi. Pokretanje sinteze je jednostavno i svodi se na odabir modula koji je potrebno sintetizirati u prozoru *Sources in Project*, te potom odabir opcije *Synthesize* u prozoru *Processes for Current Source*.

2.9. Implementacija dizajna

Nakon sinteze pristupa se implementaciji. Implementacija, u slučaju FPGA uređaja, uključuje prevođenje, preslikavanje te razmještaj i povezivanje. Sva tri koraka moguće je pokrenuti odabirom opcije **`Implement Design`** u prozoru *Processes for Current Source*. U slučaju da implementacija prođe bez problema moguće je vidjeti izvještaj svakog od pojedinih koraka odabirom odgovarajućih opcija. U zadatku laboratorijske vježbe traži se određivanje zakašnjenja izlaznih signala u odnosu na ulazne nakon razmještanja i povezivanja komponenti. Da bi se odredila kašnjenja potrebno je pokrenuti simulaciju modela nakon razmještanja i povezivanja komponenti. To se postiže ponovnim odabirom ispitnog modula u prozoru *Sources in Project*, te potom odabirom opcije **`Simulate Post-Place & Route VHDL Model`** u prozoru *Processes for Current Source.*

3. VHDL

U ovoj laboratorijskoj vježbi po prvi se puta susrećemo s jezikom VHDL. Uvod u ovaj jezik napravljen je kroz predavanja. Stoga se u nastavku pažnja posvećuje nekim specifičnostima, poput rada s višebtnim signalima kao i operacije koje se mogu vršiti nad njima. Osim signala i varijable mogu biti višebitne te sve rečeno za signale vrijedi i za njih. Nakon toga opisani su `IF` i `CASE` izrazi te strukturni VHDL.

3.1. Višebitni signali i varijable

Deklaracija višebitnih signala slična je deklaraciji jednobitnih, ali se umjesto `std_logic` mora upotrijebiti `std_logic_vector` te se također mora definirati raspon. U općem slučaju deklaracija višebitnih signala obavlja se na sljedeći način:

```
SIGNAL imesignala: std_logic_vector (donja_granica TO gornja_granica);
```

odnosno sa:

```
SIGNAL imesignala: std_logic_vector (gornja_granica DOWNTO donja_granica);
```

U oba slučaja prilikom definiranja intervala, s lijeve strane riječi `TO`, odnosno `DOWNTO`, nalazi se indeks težeg bita, dok se sa desne strane nalazi indeks najmanje značajnog bita. Umjesto riječi `imesignala` treba staviti naziv signala koji se deklarira. U slučaju da je istovremeno potrebno deklarirati više signala imena se razdvajaju zarezom.

Primjeri deklariranja višebitnih signala su:

```
-- Deklaracija internog 10-bitnog signala bit 9 je bit najveće težine
SIGNAL bus10: std_logic_vector (9 downto 0);

-- Deklaracija tri trobitna
SIGNAL sigA, sigB, sigC: std_logic_vector (2 downto 0);

-- Deklaracija ulaznog 8-bitnog signala. Bit 0 je bit najveće težine
inbus: IN std_logic_vector (0 to 8);
```

U nastavku ove pripreme se **podrazumijeva** da su svi signali deklarirani tako da se veći indeks predstavlja teži bit. Primjerice, `sigA(3)` je teži bit od bita `sigA(0)` nekog višebitnog signala `sigA`.

Dodjeljivanje vrijednosti višebitnim signalima obavlja se upotrebom standardnog operatora za dodjeljivanje vrijednosti (<=), a konstantne – višebitne – vrijednosti se moraju upisivati unutar dvostrukih navodnika (″). U dodjeljivaju je također moguće korištenje raspona. Slijedi niz primjera koji ilustrira sve do sada rečeno:

```
-- Inicijalizacija 3-bitnog signala kojemu je bit 2 bit najveće težine
sigA <= "011";

-- Prethodno dodjeljivanje je ekvivalentno sljedećem:
sigA (2 downto 0) <= "011";

-- Dodjeljivanje 3-bitnog signala sigB signalu sigA širine 6-bita
-- (sigA), pri čemu se ta 3 bita smještaju u bitove, 5, 4 i 3
sigA (5 downto 3) <= sigB;
```

U slučaju da je potrebno spajanje dva ili više signala u šire signale koristi se operator konkatenacije (`&`). Primjer:

```
-- Spajanje dva 3-bitna signala (sigB i sigC) u jedan 6-bitni signal
-- te dodjeljivanje te vrijednosti signalu sigA. sigB se smješta u
-- više bitove signala sigA, a sigC u niže bitove signala sigA.
sigA <= sigB & sigC;
```

Osim operatora konkatenacije za spajanje signala moguće je koristiti i zagradu, npr.:

```
-- Spajanje tri 3-bitna signala (sigB, sigC i sigD) u jedan 9-bitni signal
-- te dodjeljivanje te vrijednosti signalu sigA. sigB se smješta u
-- više bitove signala sigA, a sigD u niže bitove signala sigA.
sigA <= (sigB, sigC, sigD);
```

3.2. Izrazi IF i CASE

U ovom poglavlju opisani su IF i CASE izrazi. Budući da su to izrazi koji se nalaze u gotovo svakom programskom jeziku danas u upotrebi nije puno prostora posvećeno pojašnjavanju njihova rada, ali su naglašene specifičnosti vezane uz VHDL.

Sintaksa IF izraza je:

```
IF uvjet THEN
    blok VHDL izraza
{ ELSIF uvjet THEN
    blok VHDL izraza }
[ ELSE
    blok VHDL izraza ]
END IF;
```

U prethodnom kodu `uvjet` je potrebno zamijeniti odgovarajućim VHDL izrazima provjere, dok je `blok VHDL izraza` niz izraza koji se simulira ako je odgovarajući uvjet točan, odnosno – u slučaju ELSE izraza – ako ni jedan uvjet nije točan. Unutar vitičastih zagrada označeni su izrazi koji se ne moraju pojaviti, mogu se pojaviti jednom ili proizvoljno puno puta. U uglatim zagradama naznačeni su izrazi koji se mogu pojaviti maksimalno jednom, tj. moguće ih je izostaviti. Prilikom konstruiranja uvjeta mogu se koristiti sljedeći operatori:

Operator	*Značenje*	*Primjer*
=	Provjera jednakosti	sigA = “1010”, sigB = '1'
/=	Provjera nejednakosti	sigA /= sigB
and	Logička I operacija	sigA = '1' AND sigB = '0'
or	Logička ILI operacija	
not	Negacija	

Primjer IF izraza:

```
-- Ako je signal e aktivan (u stanju logičke jedinice) tada u
-- ovisnosti o vrijednosti signala sel propusti na izlaz signale
-- a odnosno b, u suprotnom na izlaz postavi logičku nulu.
IF (e = '1') THEN
    IF (sel = '1' AND e /= '0') THEN
        c <= a;
    ELSE
        c <= b;
    END IF;
ELSE
    c <= '0';
END IF;
```

U slučaju višestrukih provjera izraz IF može postati nepregledan. U tom slučaju koristi se izraz CASE. Sintaksa tog izraza je:

```
CASE izraz IS
    WHEN uvjet =>
        blok VHDL izraza
  { WHEN uvjet =>
        blok VHDL izraza }
  [ WHEN OTHERS =>
        blok VHDL izraza ]
END CASE;
```

Opet, kao i u slučaju IF naredbe, su sa uglatim i vitičastim zagradama označeni dijelovi koji se mogu ponavljati proizvoljan broj puta (vitičaste zagrade), odnosno maksimalno jednom (uglate zagrade). Umjesto izraza može stajati varijabla ili signal čiju vrijednost provjeravamo, dok se umjesto uvjeta postavljaju određene kombinacije vrijednosti koje želimo obraditi. Izraz WHEN OTHERS je dio koji se izvršava u slučaju da niti jedan od prethodnih uvjeta nije zadovoljio, slično kao ELSE u slučaju IF naredbe.

Kod IF naredbe potrebno je pripaziti da se **prilikom modeliranja kombinacijskih sklopova uključe sve moguće kombinacije**, jer u suprotnom se nakon sinteze može dobiti sekvencijalni sklop, tj. sklop koji sadrži memorijske elemente. To konkretno znači da je prilikom pisanja modela kombinacijskog sklopa obavezno navesti ELSE izraz s odgovarajućim blokom naredbi, bez obzira što je taj dio u sintaksi naveden kao neobavezan.

Kod **CASE** naredbe je **obavezno obuhvatiti sve** moguće slučajeve koje izraz može poprimiti. To znači da se mora ili navesti potreban broj *WHEN uvjet*-a (pa *WHEN OTHERS* ne pisati), ili navesti nedovoljan broj *WHEN uvjet*-a i zatim kroz *WHEN OTHERS* pokriti preostale slučajeve. Evo primjera za razmisliti: definiran je dvobitni signal: SIGNAL sig: std_logic_vector(1 downto 0). Ukoliko želimo poduzeti odgovarajuću akciju za svaku moguću kombinaciju vrijednosti tog dvobitnog signala (dakle, pokriti je jednim *WHEN uvjet*), koliko ukupno *WHEN uvjet* izraza trebamo? (Hint: puno više od 4!)

3.3. Stukturni VHDL

Strukturni VHDL obrađivan je na predavanjima. Ovdje su kompletnosti radi – uz nove – ponovljeni i neki već poznati elementi. Prilikom korištenja strukturnog VHDL-a potrebno je razumijeti razliku između komponente, tj. njene deklaracije, te instanciranja komponente – što će biti pojašnjeno u nastavku.

Strukturni VHDL za razliku od ponašajnog opisuje od kojih jednostavnijih komponenti se sklop sastoji i na koji način su te komponente međusobno povezane. Opisi pojedinih komponenti mogu biti ponašajni, ali također mogu biti i strukturni.

Nakon ključne riječi `ARCHITECTURE`, a prije ključne riječi `BEGIN`, moraju se deklarirati i svi interni signali. Interni signali služe za međusobno povezivanje komponenti. Kao pravilo može se uzeti da sve linije na nekoj shemi koje povezuju izlaze jedne komponente sa ulazom neke druge komponente treba deklarirati kao interni signal.

Komponente se upotrebljavaju u strukturnom opisu sklopa instanciranjem. Instanciranje znači konkretnu upotrebu neke unaprijed deklarirane komponente. Prema tome, kada se govori o instanci komponente govori se o konkretnom elementu u dizajnu, a kada se govori o komponenti misli se na sve elemente u dizajnu sa istim sučeljem i ponašanjem. Instanciranje i povezivanje komponenti obavlja se u bloku `ARCHITECTURE` između odgovarajućih `BEGIN` i `END` rezerviranih riječi. Sintaksa za instanciranje je sljedeća:

```
ime_instance: ENTITY work.naziv_sklopa
              PORT MAP (povezivanje ulaza i izlaza);
```

Umjesto `ime_instance` potrebno je navesti neko proizvoljno i jedinstveno ime koje će dobiti instanca komponente. Umjesto `naziv_sklopa` potrebno je staviti ime komponente koja se upotrebljava. Konačno, u zagradama nakon `PORT MAP` rezerviranih riječi obavlja povezivanje internih signala i ulaznih i izlaznih signala sklopa kojeg modeliramo sa ulazima i izlazima komponente. Povezivanje se može provesti po položaju ili može biti eksplicitno.

Detaljnije se to može pokazati na primjeru. Ako koristimo komponentu čije je sučelje:

```
ENTITY sklopI IS
    PORT (a, b: IN std_logic; z: OUT std_logic);
END ENTITY;
```

Tada tu komponentu možemo instancirati na sljedeći način:

```
i1: ENTITY work.sklopI PORT MAP (s1, s2, s3);
```

Instanca je nazvana `i1`. Signal `s1` povezuje se na `a` ulaz komponente, signal `s2` na `b` ulaz komponente, te signal `s3` na `z` izlaz komponente. U tom primjeru upotrebljen je implicitan način povezivanja, odnosno povezivanje putem pozicije. Drugim riječima, kako su navođeni interni signali tako su uparivani sa ulazima i izlazima komponente navedenima u deklaraciji komponente. Osim implicitnog, moguće je i eksplicitno povezivanje, odnosno povezivanje putem imena. Primjer eksplicitnog povezivanja:

```
i2: ENTITY work.sklopI PORT MAP (z => s3, b => s2, a => s1);
```

U tom primjeru postignut je isti efekt kao i kod implicitnog povezivanja, tj. signal `s3` spojen je na izlaz `z` sklopa, signal `s2` spojen je na ulaz `b` sklopa, te je signal `s1` spojen na ulaz `a` sklopa `sklopI`. U ovom slučaju nije bilo nužno navoditi signale po redoslijedu kako su navođeni prilikom deklaracije komponente.

Tijekom spajanja komponenti potrebno je kombiniranje signala i dodavanje invertora. Pri tome je potrebno obratiti pažnju na nekoliko pravila koja treba poštivati. Unutar PORT MAP izraza na mjestu gdje se očekuje ulazni jednobitni signal, dopušteno je staviti konstantu ('0' ili '1'), varijablu

(s1) ili komplement varijable (not s1). Nije dopušteno obavljati složene Booleove funkcije (and, or, ...). Time je dopušteno npr:

```
i1: ENTITY work.sklopI PORT MAP ('0', NOT s2, s3);
```

ali sljedeći izraz nije dopušten:

```
i2: ENTITY work.sklopI PORT MAP (NOT s1 OR s2, s2, s3);
```

Na mjesta na kojima se očekuje višebitni signal, može se dovesti odgovarajući višebitni interni signal, njegov dio ili pak agregacija signala, uz ograničenja koja će biti navedena u nastavku. Evo primjera: neka je definiran sklop sklopA čije je sučelje sljedeće:

```
ENTITY sklopA IS PORT (
  ulaz1: std_logic_vector(3 downto 0);
  ulaz2: std_logic_vector(0 to 1);
  izlaz: OUT std_logic);
END sklopA;
```

Prilikom instanciranja komponente na mjestu višebitnih ulaznih signala mogu doći agregacije konstanti, na oba načina (koristeći zagrade ili operator &). Npr. ispravno je:

```
i3: ENTITY work.sklopA PORT MAP ( ('1','0','1','1'), '0'&'1', y);
```

Međutim, agregacija unutar PORT MAP izraza ne smije sadržavati signale (niti operacije sa signalima). Dakle, sljedeće je pogrešno (ako je a jednobitni signal):

```
i3: ENTITY work.sklopI PORT MAP ( ('1',a,'1','1'), '0'&'1', y);
```

Ovakav izraz može (a i ne mora) preživjeti prevođenje, međutim, pokretanje simulacije neće uspjeti. Da bismo riješili ovaj problem, mogu se definirati dva interna signala (4-bitni int1 i 2-bitni int2), pa možemo pisati:

```
int1 <= (a, not a, not a or b, '1');
int2 <= not a & (not b or c);
i3: ENTITY work.sklopI PORT MAP ( int1, int2, y);
```

Uočite kod izvedbe agregacije operatorom & da je desni izraz stavljen u zagrade – u suprotnom se događa problem s prioritetima i prevođenje puca. Kada se agregacije pridjeljuju internim signalima (dakle nisu dio PORT MAP izraza), one mogu sadržavati i signale i složene izraze nad signalima.

Alternativno ovom pristupu, možemo definirati jedan veliki interni signal (6-bitni interni), i zatim njegove dijelove koristiti za potrebne višebitne ulaze:

```
interni <= (a, not a, not a or b, '1', not a, not b or c);
i3: sklopI PORT MAP ( interni(0 to 3), interni(4 to 5), y);
```

4. Priprema

Na laboratorijsku vježbu potrebno donijeti na papiru, pisano vlastitom rukom (dakle, ne fotokopiju, ispis s računala ili slično), samostalno izrađenu pripremu koja sadrži:

1. VHDL modele 3 tražena sklopa (definirana na početku ove upute, ovisno o Vašem JMBAG-u),

2. nacrtanu shemu multipleksorskog odnosno dekoderskog stabla (ovisno o inačici koju radite) iz koje je jasno vidljivo kako je nastao odgovarajući strukturni model,

3. nacrtan vremenski dijagram koji prikazuje očekivani rezultat simulacije sva tri sklopa, te

4. tablicu kombinacija funkcije f.

Pri tome, svaka stranica treba u gornjem desnom uglu sadržavati kemijskom olovkom napisano ime, prezime i JMBAG studenta. Alternativno, umjesto točke 1 (VHDL napisan rukom na papiru) moguće je kompletnu vježbu prije dolaska na sam termin vježbe napraviti kroz vhdllab koristeći Vaše korisničko ime i zaporku s Ferka (*naglasak na čitavu vježbu zajedno s ispitnim sklopovima i simulacijama; u tom slučaju asistent će pogledati vježbu koju ste tako napravili*). Točke 2 – 4 potrebno je riješiti na papiru neovisno o izvedbi točke 1.

5. Primjer pitanja

U nastavku je dan primjer pitanja koja se mogu pojaviti na izlaznom testu. Navedena su samo pitanja bez ponuđenih odgovora:

- Koju funkciju obavlja multipleksor?
- Koju funkciju obavlja dekoder?
- Kako se gradi multipleksorsko stablo?
- Kako se gradi dekodersko stablo?
- Kako se u VHDL-u označavaju komentari?
- Postoji li kašnjenje u simulaciji ponašajnog modela?
- Koje je puno ime nekog signala prilikom izvođenja simulacije?
- Koliko iznosi vrijeme kašnjenja sklopa?
- Koja je razlika između simulacije ponašajnog modela, i simulacije nakon smještaja u ciljnu arhitekturu?
- Koliko podatkovnih ulaza ima multipleksor koji ima 3 adresna ulaza?
- Kako se definiraju višebitni signali?
- Ispravna uporaba CASE naredbe prilikom modeliranja kombinacijskih sklopova.
- Način stvaranja primjeraka komponenti u strukturnom modeliranju?
- Agregacija signala.
- Povezivanje po imenu / povezivanje po poziciji.
- Kako operatori I, ILI i NE djeluju nad argumentima čije vrijednosti nisu samo 0 i 1, već uključuju i vrijednost U (neinicijalizirano)? Primjerice, znamo da je NOT 1 = 0; a koliko iznosi NOT U? Prikažite djelovanja svih navedenih operatora tablično (također i za Ex-ILI)!
- Prilikom strukturnog opisa nekog sklopa, taj sklop opisujemo koristeći komponente. Svaka od tih komponenti može biti opisana opet strukturno ili ponašajno. Ako se tako spuštamo do dna, kako je opisana ona najjednostavnija komponenta – strukturno ili ponašajno? Objasnite!

6. Zadane funkcije za inačicu 1

Funkciju odabirete tako da uzmete predzadnje dvije znamenke JMBAG-a (ne sumu!) i potom primjenite operator modulo 20. Znači, ako je JMBAG=ABCDEFG**HI**J, birate N = (HI) % 20. Primjerice, student čiji je JMBAG 0012345678 računa 67%20=7.

N	***f=suma_minterma(...)***
0	8, 10, 13, 14
1	5, 6, 8, 9, 10, 11
2	4, 6, 7, 8, 9, 14, 15
3	4, 5, 6, 7, 11, 13
4	3, 6, 7, 9, 12, 14, 15
5	3, 5, 7, 10, 13, 15
6	3, 4, 6, 8, 9, 14
7	3, 4, 5, 6, 7, 8, 11, 12, 14
8	2, 8, 9, 11, 12, 13, 15
9	2, 5, 7, 11, 12, 13, 15
10	2, 4, 6, 7, 11, 12, 13, 15
11	2, 4, 5, 7, 10, 11, 13, 14, 15
12	2, 3, 6, 7, 8, 10, 12, 13
13	2, 3, 5, 7, 8, 11, 12
14	2, 3, 4, 9, 12, 13, 14
15	2, 3, 4, 5, 7, 8, 9, 10, 13, 14
16	1, 6, 8, 9, 11, 12, 13
17	1, 5, 6, 7, 9, 10, 14, 15
18	1, 4, 6, 9, 11, 13
19	1, 4, 5, 7, 10, 14, 15

7. Zadane funkcije za inačicu 2

Funkciju odabirete tako da uzmete predzadnje dvije znamenke JMBAG-a (ne sumu!) i potom primjenite operator modulo 20. Znači, ako je JMBAG=ABCDEFG**HI**J, birate N = (HI) % 20. Primjerice, student čiji je JMBAG 0012345678 računa 67%20=7.

N	*f=suma_minterma(...)*
0	0, 6, 7, 8, 9, 12, 14
1	0, 5, 8, 9, 12, 13
2	0, 4, 6, 7, 8, 9, 11, 14
3	0, 4, 5, 6, 9, 10, 11
4	0, 3, 7, 10, 11, 12, 13, 14
5	0, 3, 5, 6, 10, 11, 15
6	0, 3, 4, 7, 9, 11, 12, 13
7	0, 3, 4, 5, 7, 8, 10, 11, 13, 14
8	0, 2, 10, 11, 12, 13, 15
9	0, 2, 5, 7, 10, 11, 13, 14, 15
10	0, 2, 4, 8, 10, 11, 12, 15
11	0, 2, 4, 5, 7, 8, 10, 11, 13, 15
12	0, 2, 3, 7, 8, 9, 12
13	0, 2, 3, 5, 6, 7, 9, 10, 11, 12
14	0, 2, 3, 4, 7, 8, 9, 11, 12, 13
15	0, 2, 3, 4, 5, 6, 7, 8, 9, 13
16	0, 1, 6, 7, 8, 9, 12, 14
17	0, 1, 5, 6, 8, 9, 12, 13, 14
18	0, 1, 4, 6, 7, 8, 10, 14
19	0, 1, 4, 5, 8, 9, 13, 15

Literatura:

[1] Peruško, Glavinić: *Digitalni sustavi*. Školska knjiga, 2005.

[2] Čupić, *Digitalna elektronika i Digitalna logika. Zbirka riješenih zadataka*. Kigen, 2006.