POKAZNA VEŽBA 2

VHDL opis digitalnih sistema

Potrebno predznanje

- Urađena pokazna vežba 1
- Digitalna logička kola
- Projektovanje digitalnog sistema uz pomoć logičke šeme pomoću Intel Quartus alata

Šta će biti naučeno tokom izrade vežbe?

Nakon urađene vežbe, bićete u mogućnosti da:

- Opišete proizvoljnu kombinacionu mrežu u VHDL-u
- Simulirate vaš sistem uz pomoć ModelSim-Altera alata

Apstrakt i motivacija

Implementacija složenih sistema koji se sastoje od više logičkih kola, ulaza i izlaza može oduzeti mnogo vremena. Pored opisivanja digitalnog sistema pomoću istinitosnih tablica, u ovoj vežbi ćete naučiti da opišete digitalni sistem na mnogo jednostavniji način koristeći VHDL jezik i njegove konstrukcije, dok ćemo alatu prepustiti da automatski formira istinitosne tablice i Bulove funkcije za naš sistem. VHDL jezik podržava konstrukcije koje će Vama, kao autoru digitalnog sistema, omogućiti da sistem opišete koncentrišući se na logiku ponašanja sistema, tj. šta želite da sistem ima na izlazu i pod kojim uslovima, a šablonski postupak izvođenja i minimizacije jednačina ostavite računaru. U ovoj vežbi ćete naučiti kako da simulirate ponašanje vašeg digitalnog sistema korišćenjem Intel Quartus alata, koji pored opisa digitalnog sistema omogućava i njegovu simulaciju, tj. proveru ispravnosti njegovog rada, kao i implementaciju na određenoj fizičkoj platformi.

TEORIJSKE OSNOVE

1. VHDL opis sistema sa digitalnim logičkim kolima

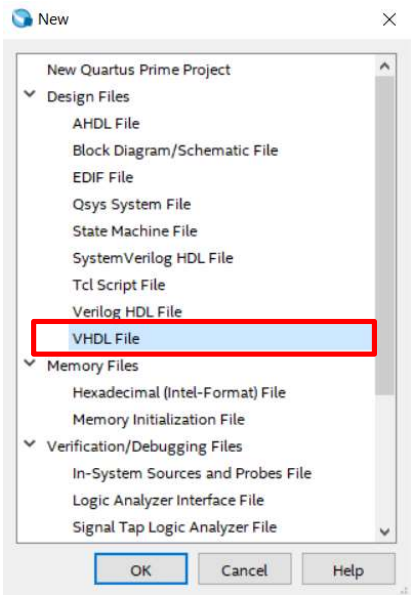
Projektovanje digitalnih sistema pomoću logičke šeme postaje mukotrpni posao kada su u pitanju složeni digitalni sistemi. Tokom rada na ovim vežbama, uskoro ćete primetiti da čak i nešto što se smatra jednostavnim komponentama digitalnih sistema (multiplekser ili sabirač, kao primer) zahteva implementaciju pomoću prilično složenih kombinacija logičkih kola, tj. funkcije koju ove komponente računaju su prilično složene Bulove funkcije.

Kako bi se olakšalo i ubrzalo projektovanje veoma složenih digitalnih sistema, osmišljeni su jezici za opis digitalnih sistema (**HDL** – ***Hardware Description Language***). Ovi jezici liče na programske jezike, kako bi njihovo usvajanje bilo lakše, no njihova namena je drugačija – oni služe da bi se lakše, brže i kraće opisala arhitektura digitalnog sistema. Opis digitalnog sistema u nekom od HDL jezika je ekvivalentan logičkoj šemi i predstavlja samo drugi način predstave istog digitalnog sistema. U ovom predmetu koristićemo **VHDL**, jezik za opis fizičke arhitekture. VHDL je HDL jezik namenjen za opis digitalnih sistema visoke integracije, iz čega je dobio i svoj naziv (VHDL – Very high scale integration circuit Hardware Description Language). Ostali često korišćeni jezici za opis fizičke arhitekture su Verilog, SystemVerilog i SystemC, ali njih nećemo koristiti u ovom predmetu.

Osnovna razlika između FPGA opisa digitalnih sistema i softverskog programiranja je način na koji se izvršavaju instrukcije. Instrukcije u softverskom programiranju (npr. C programski jezik) se izvršavaju **sekvencijalno** (uzastopno), dok se instrukcije u FPGA opisu digitalnih sistema (u našem slučaju VHDL) uglavnom **paralelno** izvršavaju (osim dodele vrednosti promenljivim unutar procesa u VHDL-u).

U prethodnim vežbama smo za opis digitalnog sistema koristili logičku šemu u ****.bdf*** datoteci. Kako bismo opisali novi digitalni sistem pomoću VHDL jezika, potrebno je napraviti novu datoteku tipa ****.vhd***.

Izborom opcije ***File->New*** u glavnom meniju, otvara se prozor za kreiranje nove datoteke (Slika 1-1) gde je potrebno izabrati opciju ***VHDL File*** i pritisnuti dugme OK. Nakon toga, otvara se nova datoteka kojoj je potrebno dati ime i sačuvati je u okviru projekta izborom ***File->Save As*** u glavnom meniju.



Slika 1-1. Kreiranje nove VHDL datoteke

Svaka VHDL datoteka se sastoji iz tri osnovna dela:

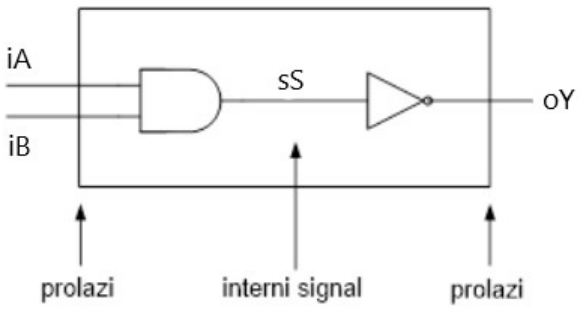
- **use** – uključuje potrebne biblioteke,
- **entity** – deklaracija entiteta onako kako se on vidi spolja, tj. naziv i svi ulazi i izlazi digitalnog sistema,
- **architecture** – definicija arhitekture digitalnog sistema, tj. komponenti i njihovog ponašanja.

Intel Quartus alat ne generiše automatski nijedan deo opisa sistema i svaki put je potrebno ručno pisati kod. Listing 1-1 prikazuje kako treba da izgleda osnovna struktura VHDL opisa digitalnog sistema.

Listing 1-1. Struktura VHDL opisa digitalnog sistema

```
library ieee;
use ieee.std_logic_1164.all;
use <libraryName>;

-- entitet sa deklaracijom svih ulaza i izlaza
entity <entityName> is
      port(
          <portName_1> : <portDirection_1> <portType_1>;
          <portName_2> : <portDirection_2> <portType_2>;
          -- ...
          <portName_N> : <portDirection_N> <portType_N>
          );
end entity;

-- arhitektura digitalnog sistema
architecture <architectureName> of <entityName> is
      -- interni signali vidljivi samo u arhitekturi
      signal <signalName> : <signalType>;
      -- ...
begin
      <architectureBody>
end architecture;
```

Ostale delove VHDL opisa ćemo objasniti kroz primer. Posmatrajmo sistem sa Slike 1-2 koji predstavlja digitalni sistem sastavljen od 2 logička kola: and i not.



Slika 1-2. Primer jednostavnog digitalnog Sistema

Entitet sistema je predstava sistema onako kako se on vidi **SPOLJA**. Posmatrajte sistem sa Slike 1-2 spolja, tako da nemate mogućnost da vidite unutrašnjost sistema (unutrašnjost pravougaonika). Šta vam je vidljivo? Ako „zatamnite“ unutrašnjost sistema, vidljivi će ostati jedino **ulazi i izlazi sistema**, tzv. **prolazi** (**ports**). Ulazi sistema su promenljive Bulovih funkcija koje sistem računa, a izlazi su rezultati Bulovih funkcija. Svaki pojedinačni izlaz predstavlja jednu Bulovu funkciju ulaza. Ukoliko sistem ima N izlaza i M ulaza, on računa N Bulovih funkcija od M promenljivih.

Digitalni sistem sa Slike 1-2 ima dva ulaza i jedan izlaz, što znači da sistem računa jednu Bulovu funkciju od dve promenljive. Prilikom opisa entiteta sistema, mora se definisati sledeće:

- ***entityName*** – jedinstveno ime entiteta (sistema),
- ***portName_1*** … ***portName_N*** – nazivi prolaza sistema,
- ***portDirection_1*** ... ***portDirection_N*** – smerovi prolaza sistema (**in** za ulaze i **out** za izlaze),
- ***portType_1*** … ***portType_N*** – tipovi signala prolaza sistema (std_logic, std_logic_vector).

Prilikom VHDL opisa digitalnog sistema, tip signala koji predstavlja jednu Bulovu promenljivu (signal koji uzima jednu od 2 vrednosti – 0 ili 1) se naziva **std_logic**. Ukoliko želite da više signala, odnosno žica u sistemu, imaju isto ime, taj skup žica se može definisati da bude tipa **std_logic_vector (N-1 downto 0)**, gde je N broj žica (bita) u signalu.

U našem primeru, dva ulaza su nazvana ***iA*** i ***iB***, tipa ***std_logic,*** a jedini izlaz je nazvan ***oY***, tipa ***std_logic***. VHDL konvencija davanja naziva signalima je da se ulazima dodeljuju nazivi sa malim početnim slovom ***i***, a izlazima nazivi sa malim početnim slovom ***o***.

Nakog definisanja entiteta sistema, definiše se **arhitektura** sistema. Arhitektura sistema predstavlja **UNUTRAŠNJOST** sistema, a njen opis podrazumeva opis ponašanja svih komponenti unutar sistema. Za kombinacione mreže, arhitektura predstavlja opis svih Bulovih funkcija koje kombinaciona mreža računa, tj. opis svih logičkih kola koji se nalaze unutar sistema.

Ukoliko pogledate ponovo Sliku 1-2, videćete da se unutar sistema pojavljuje signal koji se ne vidi spolja (***sS***). Takvi signali, koji se ne vide spolja, a postoje unutar sistema, nazivaju se **interni signali** i u VHDL opisu sistema bivaju definisani unutar arhitekture sistema. Interni signali se deklarišu pre ključne reči ***begin***, pomoću sintakse koja liči na deklaraciju prolaza (Listing 1-1).

```
signal <signalName> : <signalType>;
```

VHDL konvencija dodeljivanja naziva predlaže da se internim signalima dodeljuju nazivi sa malim početnim slovom ***s***. Vodite računa da sistem komunicira sa svojom okolinom isključivo preko prolaza (ulaza/izlaza), tj. **interni signali nisu vidljivi izvan sistema**.

Konačno, nakon ključne reči ***begin*** sledi opis arhitekture sistema, tj. logičkih kola koji se nalaze u sistemu. Prilikom opisa digitalnih sistema na nivou logičkih kola, unutar tela arhitekture treba navesti, redom, sva logička kola koja se nalaze unutar digitalnog sistema, koristeći logičke operatore: **not, and, or, xor, nand, nor, xnor**.

Operator dodele vrednosti izlaznom signalu iz logičkog kola je operator **<=**.

U nastavku je dat Listing 1-2, koji prikazuje opis digitalnog sistema sa Slike 1-2. U okviru entiteta prvo se deklarišu ulazi iA i iB, kao i izlaz oY digitalnog sistema sa Slike 1-2. Zatim se deklariše signal sS u okviru arhitekture sistema, a pre ključne reči ***begin***. Ovaj digitalni sistem koristi 2 logička kola za računanje izlaza Bulove funkcije: **AND** i **NOT** i njihova realizacija je prikazana unutar dela begin-end koji pripadaju arhitekturi sistema.

Listing 1-2. VHDL opis digitalnog sistema sa Slike 1-2

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

entity MyFirstDigitalSystem is
      port(
          iA : in std_logic;
          iB : in std_logic;
          oY : out std_logic
          );
end entity;

architecture Behavioral of MyFirstDigitalSystem is

      signal sS : std_logic;

begin
      sS <= iA and iB;
      oY <= not(sS);
end architecture;
```

Redosled navođenja logičkih kola nije bitan, pošto su dodele u VHDL jeziku jednostavan lingvistički način navođenja elemenata sistema, a ne stvarno „dodeljivanje“ vrednosti. Kao što nije bitno da li ćete na papiru, dok crtate šemu vašeg digitalnog sistema, prvo nacrtati NOT kolo sa desne strane, pa onda nacrtati AND kolo sa leve strane, ili obrnuto, tako ni redosled navođenja tih kola unutar VHDL opisa nije bitan. Dodele u VHDL jeziku ne treba posmatrati sekvencijalno, kao što bi ih posmatrali u programskim jezicima, već ih treba posmatrati kao **opis** komponenata koje uključujete u vaš digitalni sistem. Nazivi signala govore alatu kako ste vi povezali komponente koje navodite.

Naravno, VHDL omogućava da opišete više od jednog logičkog kola unutar jedne dodele, navodeći složeniju Bulovu funkciju u jednom redu. Sistem sa Slike 1-2 je moguće opisati i sledećom jednom dodelom, umesto dve:

```
oY <= not (iA and iB);
```

Zbog ovoga, prilikom opisa kombinacionih mreža, svaki izlaz se može opisati u jednom redu, kao Bulova funkcija ulaza sistema, pa se gubi potreba za internim signalima. Odluka o tome da li će vaš opis imati više jednostavnijih dodela ili manje složenijih je na vama, a ona bi trebala da ima optimalni odnos čitljivosti i veličine opisa.

2. Provera ispravnosti sistema – VHDL test bench

Nakon opisivanja digitalnog sistema i uspešne sinteze, sledi korak za proveru ispravnosti digitalnog sistema. To je moguće uraditi na dva načina – uz pomoć razvojne platforme i uz pomoć simulatora. U prethodnoj vežbi smo ispravnost sistema proveravali direktno na razvojnoj platformi MAX 10, gde smo ulaze i izlaze našeg dizajna povezali na fizičke komponente kao što su prekidači, diode i slično. Generalno je praksa da se u ranim fazama razvoja oslanjamo prvenstveno na simulator, a tek kasnije na testiranje na platformi kada je sistem uspešno simuliran. Za potrebe simulacije je potrebno kreirati posebnu VHDL datoteku koja se naziva **Test Bench**.

Test bench je digitalni sistem **BEZ PROLAZA** i služi isključivo za proveru rada nekog drugog digitalnog sistema. Zadatak test bencha je da **definiše sve ulaze** u sistem koje proverava i kupi vrednosti izlaza u svoje **interne signale**, kako bi oni mogli biti analizirani od strane alata za simulaciju.

Za razliku od opisa samog sistema (bilo da je u pitanju blok šema ili VHDL opis), test bench opisuje **sekvencu** vrednosti ulaznih signala, po čemu se razlikuje od samog opisa digitalnog sistema koji nema sekvencijalnost odnosno vremensku komponentu. Test bench uvodi **sekvencijalnost** u svoj opis zbog potrebe da se u njemu definiše **vremenski redosled signala**, odnosno da se jasno definiše kada se koja vrednost šalje na ulaz i koliko traju pauze između dve promene vrednosti na ulazu. Test bench biva korišten isključivo od strane **simulatora**, programskog alata za simulaciju rada digitalnog sistema, zbog čega i biva interpretiran na „softverski“ način, sekvencijalno.

Na Listingu 2-1 je prikazana opšta struktura VHDL opisa test bench-a.

Listing 2-1. Struktura VHDL opisa test bench-a

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

-- Prazan entitet za test bench
entity <entityName_tb> is
end entity;

-- arhitektura digitalnog sistema
architecture <architectureName_tb> of <entityName_tb> is
	-- definisati onoliko signala koliko ima entitet koji testiramo
	signal <signalName_1> : <signalType>;
	-- ...
	signal <signalName_N> : <signalType>;

	-- instanciranje komponente koju testiramo (entity postaje component)
	component <entityName>
		port(
			<portName_1> : <portDirection_1> <portType_1>;
			<portName_2> : <portDirection_2> <portType_2>;
			-- ...
			<portName_N> : <portDirection_N> <portType_N>
			);
	end component;

begin
	-- instanciranje komponente i internih signala test bench-a
	uut : <entityName> port map (
		<portName_1> => <signalName_1>,
		<portName_2> => <signalName_2>,
		-- ...,
		<portName_N> => <signalName_N>
	 );

	-- sekvencijalno zadavanje svih kombinacija ULAZNIH signala
	stimulus : process
		-- ...
	end process stimulus;
end architecture;
```

Svaka VHDL datoteka (pa i test bench) se sastoje iz tri dela: **use, entity i architecture**. Iako se test bench-evi suštinski razlikuju od VHDL opisa sistema, i dalje prate istovetnu strukturu.

Use deo se nalazi na početku svake VHDL datotete i nakon te ključne reči se navode sve biblioteke koje će biti korišćene. Za potrebe ovog semestra nama će biti dovoljne samo dve: biblioteka u kojoj se nalaze standardni logički tipovi i digitalna logička kola (ieee.std_logic_1164) i malo kasnije u semestru ćemo se upoznati i sa bibliotekom koja nam omogućuje neke osnovne aritmetičke operacije sa neoznačenim i sa označenim brojevima.

Entity deo definiše prolaze (ulaze i izlaze) sistema. Preko prolaza koji se nazivaju **port**-ovi, naš sistem može da komunicira sa spoljašnim svetom. Kada je reč o test bench-evima, entity deo će **UVEK BITI PRAZAN** iz razloga što je test bench u suštini digitalni sistem bez prolaza – on je krajnja tačka i neće se povezivati ni sa kakvim spoljašnim svetom.

Deo **architecture** opisuje "unutrašnost" Sistema, ali kod test bencha definišemo samo interne signale u koje ćemo mapirati ulaze i izlaze iz glavnog entiteta koji testiramo i koji smo opisali u prethodnoj VHDL datoteci.

Potrebno je definisati onoliko SIGNALA koliko ima PROLAZA u glavnom entitetu koji testiramo.

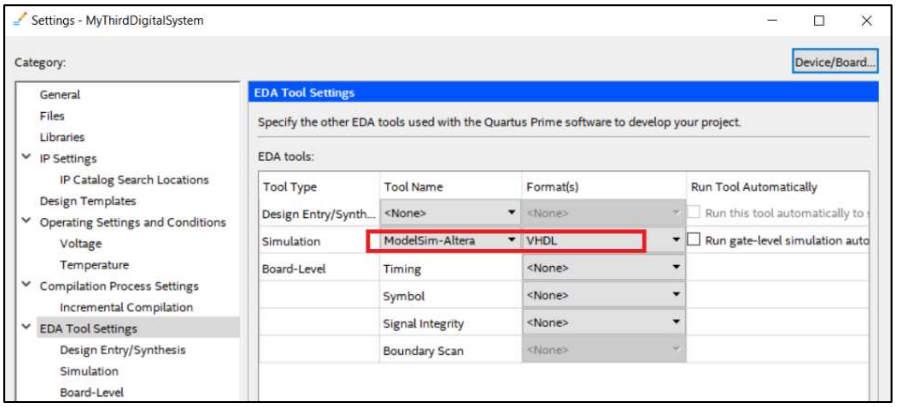
Takođe, u okviru arhitekture test bench-a moramo instancirati komponentu koju ćemo testirati i povezati njene prolaze sa signalima u test bench-u (**port map**). Prvo je potrebno uraditi deklaraciju komponente. **Komponenta** izgleda identično kao entity blok modula koji želite da testirate, samo je ključna reč **entity** zamenjena sa **component**. Komponenta se deklariše u sekciji pre ključne reči **begin**, a unutar arhitekture. Posle toga se glavni entitet mapira (povezuje) unutar arhitekture, ali posle **begin**. Prilikom mapiranja (povezivanja) prolaza i signala, potrebno je **sa leve strane navesti prolaze glavnog entiteta koji testiramo**, **a sa desne strane znaka => internne signale iz test bench arhitekture.**

PROLAZI GLAVNOG ENTITETA => SIGNALI TEST BENCH-a

Nakon instanciranja komponente i mapiranja, potrebno je zadati sekvencu ulaznih vrednosti sa kojima želimo da verifikujemo sistem i to ćemo navesti u **stimulus procesu**, nakon ključne reči **begin** od stimulus procesa.

Kako bismo verifikovali da naš sistem radi ispravno, poželjno je u test bench-u pokriti **SVE KOMBINACIJE** ulaza iz istinitosne tablice, a zatim u simulaciji proveriti da li smo dobili očekivani rezultat na izlaznim signalima za svaku od zadatih kombinacija ulaza.

Pre nego što napravimo test bench datoteku, potrebno je podesiti EDA alate u slučaju da ste taj korak preskočili tokom kreiranja projekta. Izborom opcije u glavnom meniju **Assignment->Settings->EDA Tool Settings** podesite deo za simulaciju kao na Slici 2-1. Na vežbama ćemo koristiti alat **ModelSim-Altera** i format **VHDL**.



Slika 2-1. Podešavanje simulatora

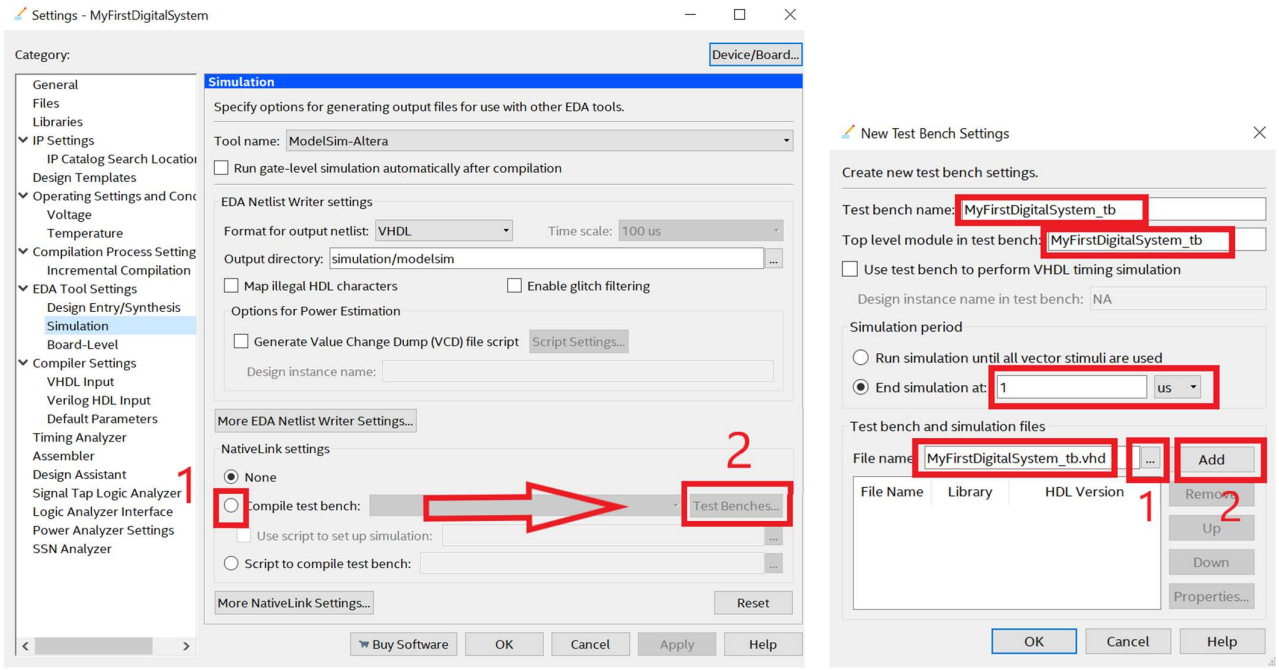
Simulator je sada podešen i potrebno je **ponovo prevesti ceo projekat**, kako bi se kreirao folder za simulaciju u koji ćemo dodati novu datoteku za testiranje. U slučaju da ste prilikom kreiranja projekta već podesili ovu stavku, automatski će biti kreiran folder za simulaciju.

Napravite novu ***.vhd*** datoteku, a zatim je sačuvajte kao ***_tb.vhd***, gde je * naziv entiteta koji se testira (u našem slučaju **MyFirstDigitalSystem**), a dodatak u nazivu **tb** označava da je to test bench datoteka.

Test datoteku je potrebno snimiti u projektni direktorijum (**File->Save**) i dodati je u projekat pomoću opcije menija **Project->Add/Remove Files in Project...**

Potrebno je dodati test bench datoteku i u simulaciju: **Assignment->Settings->EDA Tool Settings->Simulation** kao na Slici 2-2.

Izaberite dugme **Compile test bench**: a zatim pritisnite **Test Benches…** dugme i otvoriće se prozor za dodavanje nove datoteke, gde je potrebno uneti ime test datoteke (može biti više test bench datoteka), podesiti period simulacije na 1 µs, navesti putanju do test bench datoteke i pritisnuti **Add.**



Slika 2-2. Dodavanje nove test bench datoteke

Nakon kreiranja test datoteke i dodavanja u projekat i simulaciju, možemo početi sa pisanjem koda koji će testirati naš digitalni sistem. Kako bismo verifikovali da sistem radi ispravno, poželjno je u test bench-u pokriti sve moguće kombinacije ulaza (iA i iB) iz istinitosne tablice, a zatim u simulaciji proveriti da li smo dobili željeni rezultat (oY) za svaku od zadatih kombinacija ulaza. Sistem ima 2 ulaza i potrebno je verifikovati ukupno 2^2 tj. 4 kombinacije, tako da ćemo ispravnost sistema proveriti koristeći sledeću sekvencu ulaznih vrednosti:

- na početku, oba ulaza neka budu jednaka 0 i neka to stanje traje 100 ns,
- nakon toga, neka signal *iA* ima vrednost 0, a signal *iB* vrednost 1 i neka novo stanje traje 100 ns,
- nakon toga, neka signal *iA* ima vrednost 1, a signal *iB* vrednost 0 i neka novo stanje traje 100 ns,
- nakon toga, oba ulaza neka budu jednaka 1 i neka ovo stanje traje do kraja simulacije.

Na ovaj način smo pokrili sve kombinacije iz istinitosne tablice za ulaze iA i iB. Listing 2-2 prikazuje kako treba da izgleda test bench za verifikaciju našeg prvog digitalnog sistema.

Listing 2-2. Struktura VHDL test bench datoteke

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

-- entitet test bench-a je bez prolaza
entity MyFirstDigitalSystem_tb is
end MyFirstDigitalSystem_tb;

architecture Test_tb of MyFirstDigitalSystem_tb is

	-- broj signala = broj prolaza glavnog entiteta
	signal sA : std_logic;
	signal sB : std_logic;
	signal sY : std_logic;

	-- komponenta koju testiramo - MyFirstDigitalSystem
	component MyFirstDigitalSystem
	port(
		iA : in std_logic;
		iB : in std_logic;
		oY : out std_logic
	 );
	end component;

	begin
	-- mapiranje glavnih portova i signala test bench-a
	uut : MyFirstDigitalSystem port map (
		iA => sA,
		iB => sB,
		oY => sY
	 );

	-- zadavanje svih kombinacija ULAZNIH signala
	stimulus : process
	begin
		sA <= '0';
		sB <= '0';
		wait for 100 ns;
		sA <= '0';
		sB <= '1';
		wait for 100 ns;
		sA <= '1';
		sB <= '0';
		wait for 100 ns;
		sA <= '1';
		sB <= '1';
		wait;	-- wait „drži“ poslednje vrednosti do kraja simulacije
	end process stimulus;
end architecture;
```

Vremenska komponenta u opisu sekvence ulaznih vrednosti se opisuje koristeći **wait for** iskaz. Ovaj iskaz govori simulatoru koliko dugo da „drži“ vrednosti na ulazu do naredne promene.

Komentari u VHDL-u se pišu nakon dva minusa (--). Sve navedeno nakon toga alat ignoriše.

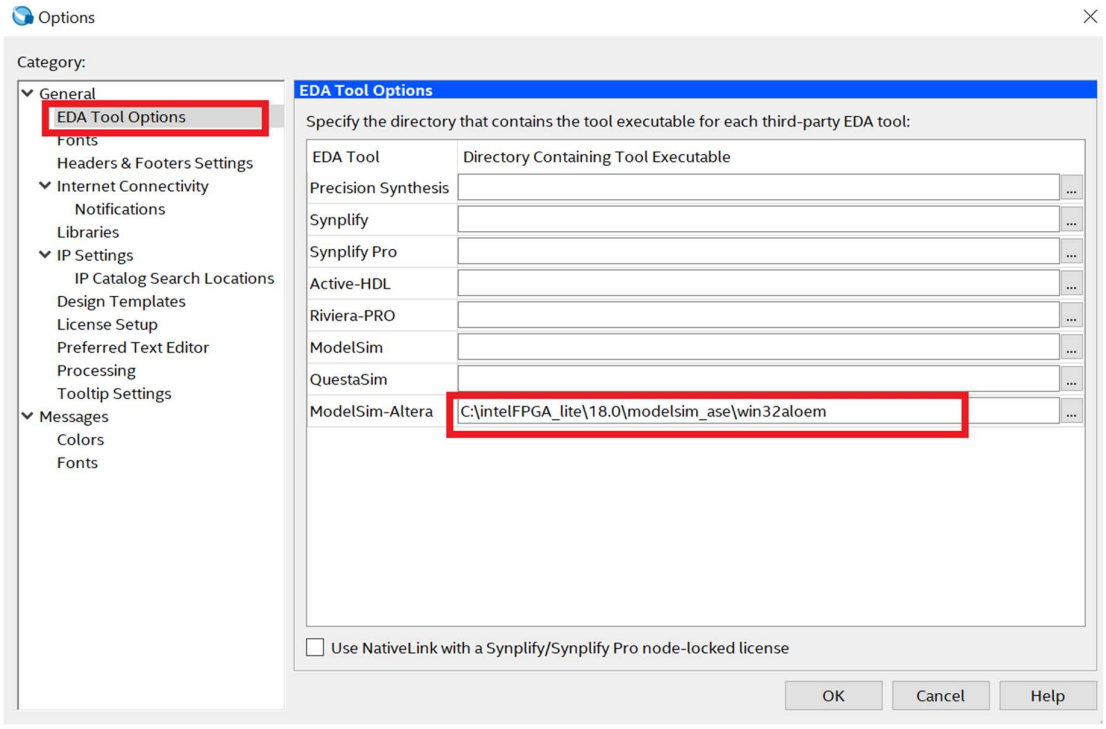
Na kraju stimulus procesa treba navesti iskaz **wait** jer on govori simulatoru da poslednje stanje ulaznih signala zadrži beskonačno, odnosno do kraja simulacije. Ukoliko ne postoji wait iskaz nakon poslednje zadate kombinacije ulaza, simulacija će stati istog trenutka i verovatno nećete biti u mogućnosti da vidite izlaz za poslednju kombinaciju ulaza. Ukoliko iza poslednje kombinacije ulaza stavite takođe iskaz **wait for 100 ns**, simulacija će se ponavljati do kraja isteka zadatog intervala.

Kada smo uspešno napisali i uvezali test bench, sledeći korak jeste pokretanje simulacije. Potrebno je sačuvati projekat i još jednom **pokrenuti proces sinteze**. Ukoliko dobijete poruku da je sinteza uspešno završena možemo (konačno) preći na pokretanje simulacije.

2.1. Pokretanje simulacije

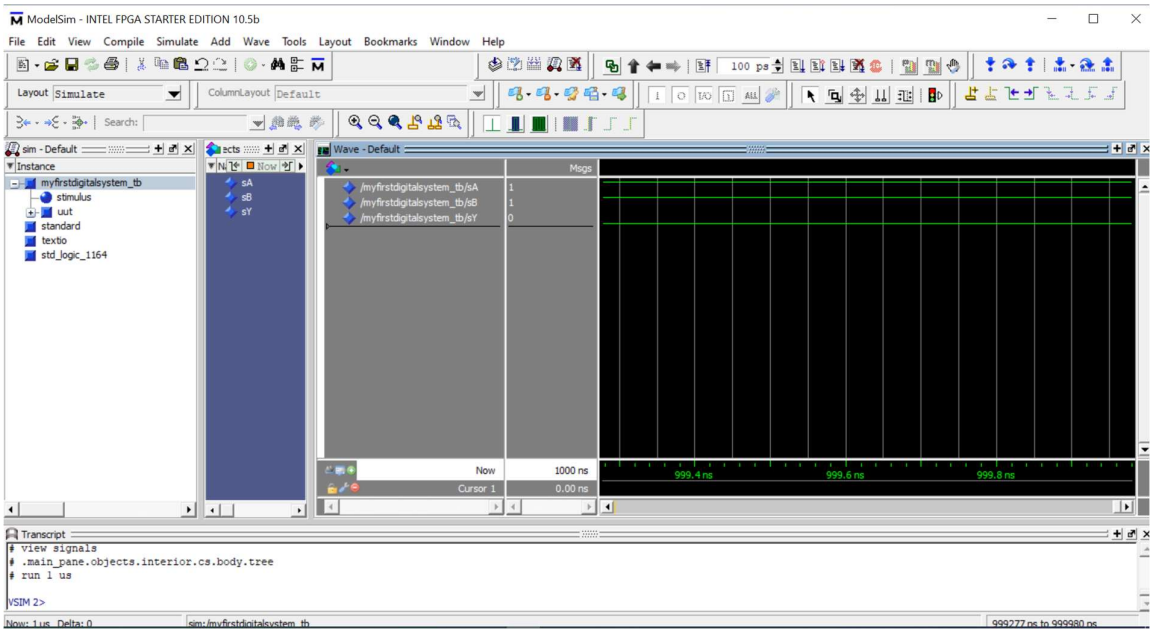
ModelSim-Altera simulator je sastavni deo Intel Quartus programskog paketa. Uz pomoć ovog programa moguće je izvršiti funkcionalnu proveru ispravnosti digitalnog sistema koji se projektuje.

U prethodnim koracima za pisanje test bench datoteke uradili smo delimično podešavanja i za simulator. Za ispravnu simulaciju, takođe je potrebno podesiti putanju do izvršne datoteke ModelSim-Altera simulatora izborom opcije glavnog menija **Tools->Options->General->EDA Tool Options** i u polju **ModelSim-Altera** podesiti putanju do direktorijuma sa izvršnom datotekom, kao na Slici 2-3.



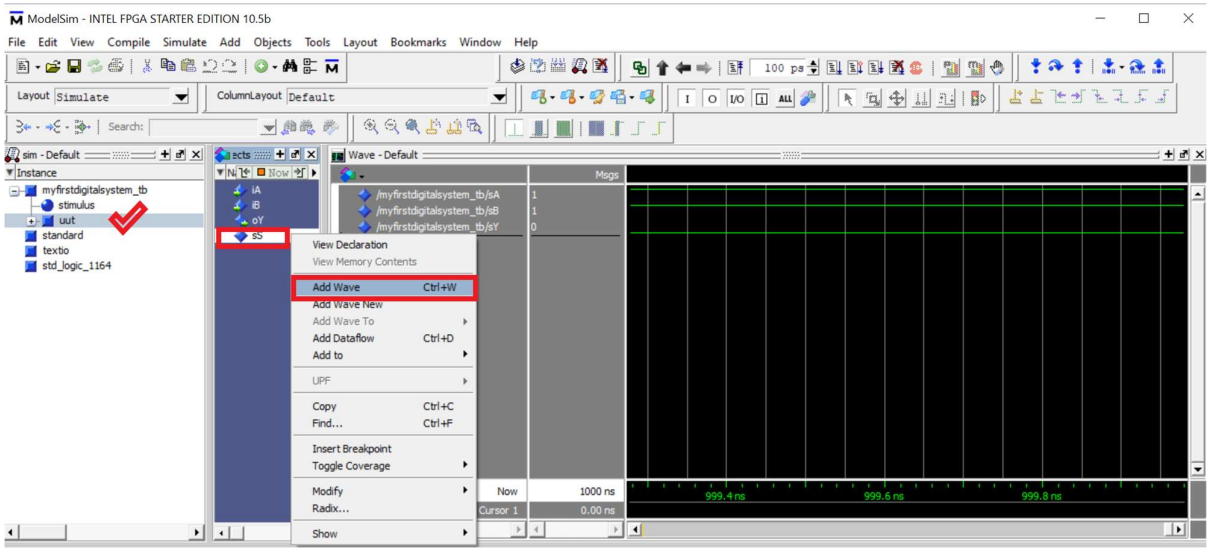
Slika 2-3. Podešavanje putanje do izvršne datoteke ModelSim-Altera simulatora

Sada možemo pokrenuti ModelSim-Altera simulator izborom sledeće opcije Tools menija u Intel Quartus alatu: **Tools->Run Simulation Tool->RTL Simulation** nakon čega će se prikazati prozor kao na Slici 2-4.



Slika 2-4. Izgled osnovnog prozora ModelSim-Altera simulatora

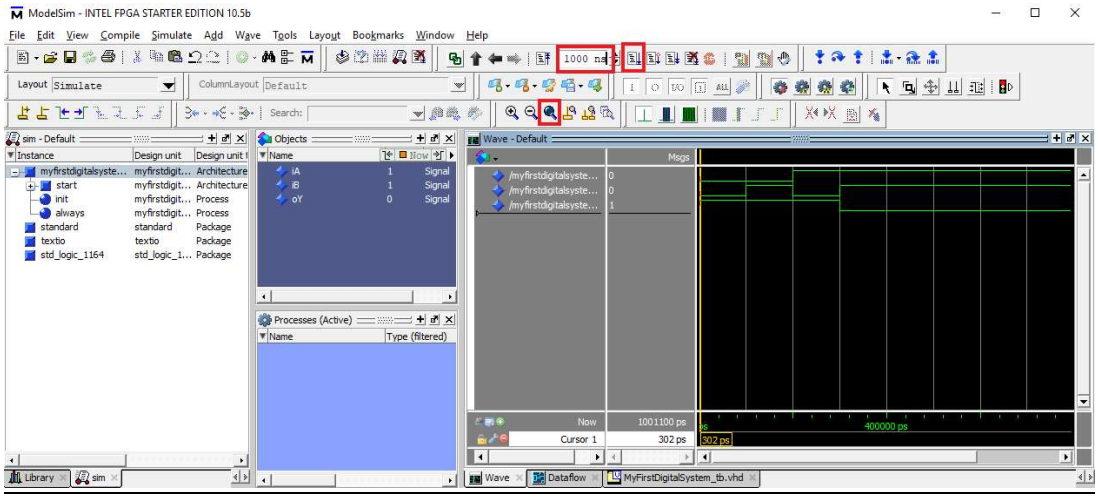
Nakon pokretanja simulatora, u delu za prikaz signala (prozor ***Wave***) biće prikazani svi prolazi glavnog entiteta (ulazi A i B i izlaz Y). Ukoliko želite da prikažete neki od internih signala, potrebno je izabrati objekat ***uut*** (*eng. Unit Under Testing)* je naziv koji smo dali prilikom mapiranja, ali može biti bilo koji naziv), kao na Slici 2-5. Nakon toga će se u plavom srednjem prozoru prikazati svi signali korišćeni u glavnom entitetu. Izborom željenog signala i pritiskom desnog klika miša, otvoriće se prozor gde je moguće izabrati opciju ***Add Wave***. Nakon ovoga, izabrani signal će biti prikazan u ***Wave*** prozoru sa ostalim signalima.



Slika 2-5. Dodavanje signala za iscrtavanje

Oblici naknadno uključenih signala će biti iscrtani samo u vremenskim trenucima simulacije nakon trenutka uključivanja signala u listu. Ukoliko želimo da omogućimo iscrtavanje oblika svih signala od početnog trenutka (i onih naknadno dodatih signala) neophodno je restartovati simulaciju i ponovo pokrenuti opciju **Run**.

U gornjem delu prozora sa ikonicama, pronađite polje koje definiše period signala (npr. 100 ps) i postavite na neku veću vrednost, npr. 1000 ns. Levo od perioda se nalazi ikonica za resetovanje prikaza svih signala. Desno od te polja za vreme je ikonica za pokretanje simulacije (Run). Pokrenite simulaciju i signali će se pojaviti u prozoru za iscrtavanje, kao na Slici 2-6. Svi signali bi trebali biti zelene boje. Ukoliko je neki signal crvene boje, to znači da jedan ili više ulaza nisu definisani dobro. Sada proverite da li su dobijeni očekivani rezultati uz pomoć istinitosne tablice.



Slika 2-6. Pokretanje simulacije

Grupa ikona na Slici 2-7 omogućava podešavanje prikaza signala kao i kontrolu toka izvršenja simulacije.

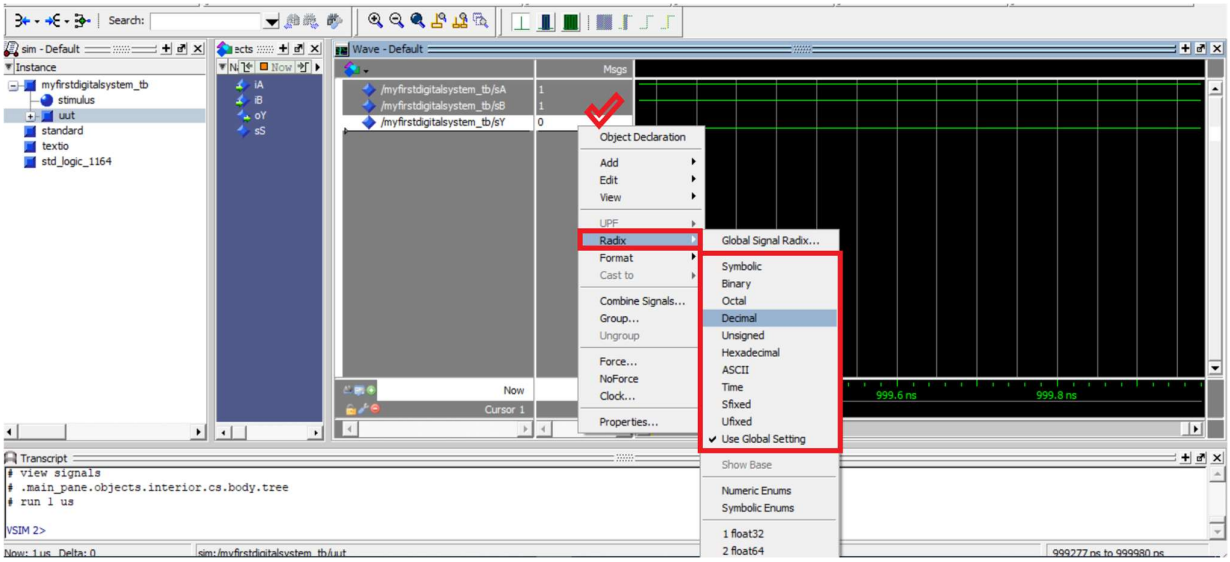
Slika 2-7. Ikonice za podešavanje signala i kontrolu izvršenja simulacije

Značenja pojedinih ikona su:

- **Zoom In** – povećanje nivoa uvećanja.
- **Zoom Out** – smanjenje nivoa uvećanja.
- **Zoom to Full View** – omogućava pregled kompletnog toka simulacije.
- **Find Previous Transition** – skok na prethodnu promenu signala.
- **Find Next Transition** – skok na narednu promenu signala.
- **Insert Cursor** – postavljenje pojedinačnog markera.
- **Delete Cursor** – uklanjanje pojedinačnog markera.
- **Restart** – restartovanje simulacije.
- **Run All** – pokretanje izvršenja simulacije na neodređen vremenski period.
- **Run**– pokretanje izvršenja simulacije na vremeski period definisan u susednom polju.
- 1000 ns **Run Length** – dužina trajanja simulacije
- **Step Into** – izvršenje simulacije korak po korak sa uvidom u progres izvršenja u HDL kodu.
- **Break** – zaustavljanje izvršenja simulacije.

Da bi simulacioni dijagram bio pregledniji, potrebno je pritisnuti na ikonu **Zoom to Full View**. Na početku, dijagram prikazuje sve signale i vektore u binarnim vrednostima. Radi lakše čitljivosti vektora, moguće je promeniti način ispisivanja njihovih vrednosti u neki drugi brojni sistem, npr. decimalni ili heksadecimalni – Slika 2-8. Desnim

klikom na željeni signal (ili više izabranih signala istovremeno), otvoriće se prozor sa opcijama. Izborom opcije ***Radix***, u novom prozoru će se pojaviti različite opcije za prikazivanje vrednosti signala (Binary, Octal, Decimal, Unsigned).



Slika 2-8. Promena načina ispisivanja vrednosti signala

Nakon uspešno završene simulacije, sledi korak provere ispravnosti sistema na razvojnoj platformi MAX 10. Na prethodnim vežbama smo se upoznali sa dodelom pinova, tako da je sada potrebno ponoviti taj korak i proveriti ispravnost rešenja na fizičkoj platformi.

Port	Smer	Komponenta	Naziv komponente	Location	I/O standard
iA	input	Levo dugme	i_pb_left	PIN_D1	3.3 V Schmitt Trigger
iB	input	Desno dugme	i_pb_right	PIN_E3	3.3 V Schmitt Trigger
oY	output	LED dioda 0	o_led[0]	PIN_A8	3.3-V LVTTL

3. VHDL – dodela vrednosti signalu

VHDL jezik omogućuje nekoliko načina za dodelu vrednosti signalu, koji su kraći i čitljiviji od opisa na nivou logičkih kola. Ovi opisi podsećaju na opis programske podrške pošto koriste iste ili slične konstrukcije.

U VHDL jeziku, signalu se vrednost može dodeliti na sledeće načine:

- Dodelom vrednosti signalu na nivou logičkih kola (and, or, not, ...),
- Dodelom vrednosti signalu pomoću ostalih VHDL operatora (opis signala pomoću logičkih operatora i operatora konkatenacije),
- Uslovnom dodelom vrednosti signalu,
- Pomoću kombinacionih procesa.

Kombinacione mreže su složeni digitalni sistemi kod kojih logičko stanje izlaza zavisi **samo od trenutnih vrednosti signala na ulazu**. Čim se vrednost signala na ulazu promeni, menja se i izlaz sistema. U ovoj vežbi su nam od interesa samo osnovna logička kola, kao osnovni predstavnici kombinacionih mreža, dok ćemo se u narednoj vežbi detaljno upoznati sa svim ostalim predstavnicima ovog tipa mreža i naučiti kako da ih opišemo na sve prethodno nabrojane načine za dodelu vrednosti.

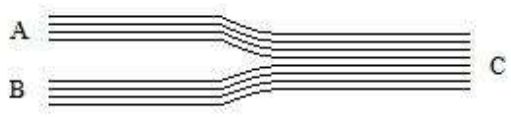
3.1. Dodela vrednosti pomoću operatora <=

Na nivou logičkih kola, vrednost signala se definiše i dodeljuje pomoću operatora <= navodeći logički izraz sa desne strane, koristeći VHDL logičke operatore **not**, **and**, **or**, **xor**, **nand**, **nor**, **xnor**,...

Prilikom definisanja (dodele) vrednosti nekom signalu, mogu se koristiti i ostali VHDL operatori. Trenutno je za nas najzanimljiviji operator za spajanje ili **konkatenaciju (&)** koji omogućuje da signalu dodelimo vrednost kombinacije nekih drugih signala. Na primer, ako pretpostavimo da je signal C 8-bitni, a signali A i B 4-bitni, tada ako napišemo sledeće:

```
C <= A & B;
```

signal C će dobiti vrednost koja se dobije spajanjem signala A i B. Viših 4 bita signala C će dobiti vrednost signala A, a nižih 4 bita signala C će dobiti vrednost signala B. **Konkatenacija** se može posmatrati i kao davanje novog imena skupu žica. Posmatrajući Sliku 3-1, ako signal A predstavlja gornje 4 žice, a signal B predstavlja donje 4 žice (svaka žica prenosi 1 bit, signali su 4-bitni), tada gornja konkatenacija znači da ovih 8 žica posmatramo kao jedinstveni signal C. Konkatenacija ne definiše nove komponente u digitalnom sistemu, već samo postojećim "žicama" iz drugih signala daje novo ime i posmatra kao novu celinu.



Slika 3-1. Konkatenacija signala A i B u signal C

Konkatenacija se može vršiti i nad delovima signala koristeći operator indeksiranja, kao i nad konstantama. Na primer, ako napišemo sledeće:

```
signal sA : std_logic_vector(3 downto 0) := „1010“;
signal sB : std_logic_vector(5 downto 0) := „011011“;
signal sC : std_logic_vector(7 downto 0);
-- ...
   sC <= '1' & sA(1 downto 0) & sB(4 downto 2) & "00";
```

Signal C je definisan kao 8-bitni signal i nakon izvršenih operacija konkatenacije će imati sledeću vrednosti:

```
C = „11011000“
```

Za kompletan spisak VHDL **aritmetičkih operatora** preporučujemo da konsultujete udžbenik i naredne vežbe, jer oni izlaze izvan opsega ove vežbe.

3.2. Uslovna dodela vrednosti

Uslovna dodela se koristi da bi se opisala kombinaciona logika tipa "IF-THEN-ELSE". Kako bi se skratio VHDL opis, VHDL ne zahteva da se ovakva logika opisuje na nivou logičkih kola, već za to postoji posebna konstrukcija prilikom dodele signala. Sintaksa uslovne dodele je data u Listingu 3-1.

Listing 3-1. Sintaksa uslovne dodele u VHDL-u

```
<signalName> <= <expression_1> when <condition_1> else
                <expression_2> when <condition_2> else
                -- ...
                <expression_N> when <condition_N> else
                <expression_Default>;
```

Signal `<signalName>` dobija vrednost `<expression_1>` ukoliko je ispunjen uslov `<condition_1>`, u suprotnom dobija vrednost `<expression_2>` ukoliko je ispunjen uslov `<condition_2>`, itd. Ukoliko nijedan od uslova nije ispunjen, signal dobija vrednost `<expression_Default>`.

NAPOMENA: Neophodno je definisati **podrazumevanu vrednost** i navesti je kao poslednju **BEZ** ključne reči **else**.

Za definisanje uslova mogu se koristiti sledeći VHDL **relacioni operatori**:

=	jednakost	<	manje od	<=	manje od ili jednako
/=	nejednakost	>	veće od	>=	veće od ili jednako

3.3. Kombinacioni procesi

Kombinaciona mreža se može opisati i pomoću **kombinacionog procesa**. Naredba **process** služi za ponavljanje niza naredbi u okviru procesa (tela procesa. Unutar zagrada u prvoj liniji procesa definiše se **lista osetljivosti** procesa, koja služi alatu za **simulaciju** kako bi znao kada komponenta (kombinaciona mreža) koja se opisuje procesom može da promeni izlaznu vrednost.

Sa svakom promenom argumenta iz liste osetljivosti izvršava se niz naredbi zadat u okviru tela procesa. Dakle, u listi osetljivosti kombinacionog procesa treba da budu navedeni **svi ulazi kombinacione mreže koji utiču na promenu izlaza u datom procesu**, odvojeni zarezom.

Listing 3-2. Sintaksa VHDL kombinacionog procesa

```
process (<sensitivityList>) begin
    <processBody>
end process;
```

Telo procesa sadrži opis kombinacione mreže. Svim izlazima kombinacione mreže treba da se dodeli vrednost. Za tu svrhu se može koristiti operator dodele, ali i uslovne VHDL konstrukcije kao što su **IF-ELSE** i **CASE, koje mogu da se koriste samo u okviru procesa**.

IF-ELSE naredba služi za sekvencijalno izršavanje naredbi kada je zadovoljen zadati uslov.

Listing 3-3. Sintaksa IF-ELSE strukture

```
process (<sensitivityList>) begin
      if (<condition_1>) then
            <statements>;
      elsif (<condition_2>) then
            <statements>;
      else // obavezno definisati podrazumevanu ELSE granu KOMBINACIONE MREŽE
            <statements>;
      end if;
end process;
```

CASE naredba se koristi za uslovno izvršavanje naredbi kada je zadovoljen unapred zadati uslov:

Listing 3-4. Sintaksa CASE strukture

```
process (<sensitivityList>) begin
      case (<signalName>) is
            when <value_1> => <statements>;
            when <value_2> => <statements>;
            -- ...
            when <value_N> => <statements>;
      // obavezno definisati podrazumevanu WHEN OTHERS granu KOMBINACIONE MREŽE
            when others => <statements>;
      end case;
end process;
```

VAŽNA NAPOMENA: Prilikom definisanja kombinacionih mreža pomoću procesa koji sadrže IF-ELSE i CASE, neophodno je definisati svaki izlaz procesa u svakoj grani IF i CASE strukture. U suprotnom kombinaciona mreža je nepotpuno definisana i rezultovaće lošim digitalnim sistemom. Takođe treba **izbegavati paralelne IF i CASE strukture**.

3.4. Koja je razlika ranijih načina opisivanja?

Naveli smo nekoliko pristupa za opisivanja kombinacionih mreža:

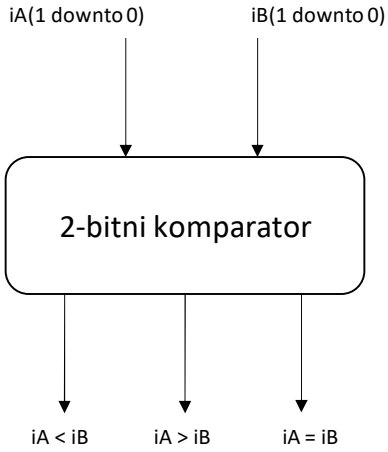
1. pomoću operatora dodele
2. pomoću uslovne dodele (što se još naziva i konkurentnim iskazom)
3. pomoću kombinacionih procesa

Između navedenih načina opisivanja kombinacionih mreža **NEMA RAZLIKE**, odnosno svi rezultuju istom kombinacionom mrežom. Procesi imaju više podržanih konstrukcija, a dodele su u najvećem broju slučajeva kraće i ne zahtevaju listu osetljivosti. Da li ćete koristiti dodele ili procese ostaje na vama.

ZADACI

4. VHDL implementacija digitalnog sistema

U ovom zadatku je potrebno implementirati dvobitni komparator. **Komparator** je jednostavna kombinaciona mreža za poređenje vrednosti ulaznih signala. U ovom zadaku, ulazni signali su **dvobitni**, dok izlaz sistema predstavljaju **3 jednobitna** signala koji se aktiviraju ukoliko je jedna vrednost manja, odnosno veća od druge, ili ukoliko su jednake.



Slika 4-1. Dvobitni komparator

Vaš zadatak je da implementirate 2-bitni komparator na sledeće načine, a zatim testirate njegovo ponašanje:

- opisom u VHDL-u na nivou logičkih kola,
- opisom u VHDL-u koristeći uslovnu dodelu,
- opisom u VHDL-u koristeći kombinacioni proces za IF-ELSE

PITANJE: Kako bi implementirali ovaj digitalni sistem crtanjem logičke šeme u blok dijagram datoteci?

REŠENJE:

Prvi korak u implementaciji svakog digitalnog sistema je analiza broja prolaza i pravljenje istinitosne tablice. Dvobitni komparator ima sledeće prolaze i u nastavku je prikazan opis entiteta u VHDL-u:

- 2 dvobitna ulaza opšte namene – ***iA*** i ***iB***
- 3 jednobitna izlaza – ***oLESS***, ***oGREAT***, ***oEQUAL***

```
entity Comparator is
      port(
            iA     : in std_logic_vector(1 downto 0);
            iB     : in std_logic_vector(1 downto 0);
            oLESS  : out std_logic;
            oGREAT : out std_logic;
            oEQUAL : out std_logic
    );
end entity;
```

Zatim sledi formiranje istinitosne tablice za komparator. S obzirom na to da imamo 2 ulazna signala širine 2 bita, imaćemo ukupno 16 ulaznih kombinacija (2^4) u istinitosnoj tablici.

Izlazni signali, koji pokazuju da li je vrednost signala A manja, veća ili jednaka vrednosti signala B, su jednobitni i biće postavljeni na vrednost 1 kada je odgovarajući uslov ispunjen. U nastavku sledi Tabela 4-1 koja predstavlja istinitosnu tablicu dvobitnog komparatora:

Tabela 4-1. Istinitosna tablica dvobitnog komparatora

A1	A0	B1	B0	A < B	A > B	A = B
0	0	0	0	0	0	1
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	1	0
0	1	0	1	0	0	1
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	0
1	0	1	0	0	0	1
1	0	1	1	1	0	0
1	1	0	0	0	1	0
1	1	0	1	0	1	0
1	1	1	0	0	1	0
1	1	1	1	0	0	1

Na osnovu istinitosne tablice će se izvesti logičke funkcije za opis sistema u VHDL-u na nivou logičkih kola:

$$oLESS = \overline{A1}\,\overline{A0}\,\overline{B1}\,B0 + \overline{A1}\,\overline{A0}\,B1\overline{B0} + \overline{A1}\,\overline{A0}\,B1B0 + \overline{A1}\,A0\,B1\,\overline{B0} + \overline{A1}\,A0\,B1B0 + A1\overline{A0}\,B1B0$$
$$oGREAT = \overline{A1}\,A0\,\overline{B1}\,\overline{B0} + A1\,\overline{A0}\,\overline{B1}\,\overline{B0} + A1\,\overline{A0}\,\overline{B1}\,B0 + A1A0\overline{B1}\,\overline{B0} + A1\,A0\overline{B1}B0 + A1\,A0\,B1\,\overline{B0}$$
$$oEQUAL = \overline{A1}\,\overline{A0}\,\overline{B1}\,\overline{B0} + \overline{A1}\,A0\,\overline{B1}\,B0 + A1\overline{A0}\,B1\overline{B0} + A1A0B1B0$$

Minimizacijom jednačina algebarskim putem ili korišćenjem Karnoovih mapa dobiju se sledeće jednačine:

$$oLESS = \overline{A1}\,B1 + \overline{A0}\,B1B0 + \overline{A1}\,\overline{A0}\,B0$$
$$oGREAT = A1\,\overline{B1} + A1A0\,\overline{B0} + A0\overline{B1}\,\overline{B0}$$
$$oEQUAL = \overline{(A1 \oplus B1)}\;\overline{(A0 \oplus B0)}$$

U nastavku sledi opis arhitekture komparatora u VHDL-u na **nivou logičkih kola**:

```
architecture Logicka_kola of Comparator is
begin
   oLESS <= (not(iA(1)) and iB(1)) or (not(iA(0)) and iB(1) and iB(0)) or
            (not(iA(1)) and not(iA(0)) and iB(0));
   oGREAT <=(iA(1) and not(iB(1))) or (iA(1) and iA(0) and not(iB(0))) or
            (iA(0) and not(iB(1)) and not(iB(0)));
   oEQUAL <=(not(iA(1) xor iB(1))) and (not(iA(0) xor iB(0)));

end architecture Logicka_kola;
```

Očevidno je da postoji velika verovatnoća za pravljenje i još teže otkrivanje grešaka ukoliko se sistem opisuje u VHDL-u na nivou logičkih kola, tako da je bilo koji od narednih načina opisivanja dosta jednostavniji i čitljiviji. U nastavku sledi opis arhitekture dvobitnog komparatora u VHDL-u pomoću **uslovne dodele**:

```
architecture Uslovna_dodela of Comparator is
begin
   oLESS <=   '1' when (iA < iB) else '0';
   oGREAT <=  '1' when (iA > iB) else '0';
   oEQUAL <=  '1' when (iA = iB) else '0';
end architecture Uslovna_dodela;
```

KOMBINACIONI proces IF-ELSE: mora imati definisanu **LISTU OSETLJIVOSTI**, gde se navode **SVI ULAZNI SIGNALI** koji utiču na promenu izlaznih signala tog procesa. Obavezno je definisati **SVAKI IZLAZ** u svakoj **iF-ELSE** grani. U nastavku je opisana arhitektura dvobitnog komparatora u VHDL-u pomoću **IF-ELSE procesa**:

```
architecture if_else of Comparator is
begin
   process (iA, iB) begin -- iA i iB utiču na promenu izlaza -> lista osetlj.
         if (iA < iB) then
               oLESS  <= '1';
               oGREAT <= '0';
               oEQUAL <= '0';
         elsif (iA > iB) then
               oLESS  <= '0';
               oGREAT <= '1';
               oEQUAL <= '0';
         else
               oLESS  <= '0';
               oGREAT <= '0';
               oEQUAL <= '1';
         end if;
   end process;
end architecture if_else;
```

KOMBINACIONI proces CASE: mora takođe imati definisanu **LISTU OSETLJIVOSTI**, gde se navode **SVI ULAZNI SIGNALI** koji utiču na promenu izlaznih signala tog procesa. Obavezno je definisati **SVAKI IZLAZ** u svakoj **CASE** grani, kao i podrazumevanu granu **WHEN OTHERS**. Ovaj način opisivanja nije pogodan za opisivanje komparatora, te stoga neće biti urađen.

PITANJE: Kako bi izgledao opis dvobitnog komparatora u VHDL-u pomoću CASE procesa?

U okviru jedne VHDL datoteke može da postoji samo **JEDAN** definisan **ENTITET**, ali on može imati **VIŠE** različitih **ARHITEKTURA**. Svaka arhitektura se definiše imenom i na taj način se razlikuju. Ukoliko u opisu sistema ima više različitih arhitektura jednog entiteta, testiranje u okviru test-bench datoteke zahteva mapiranje na sledeći način:

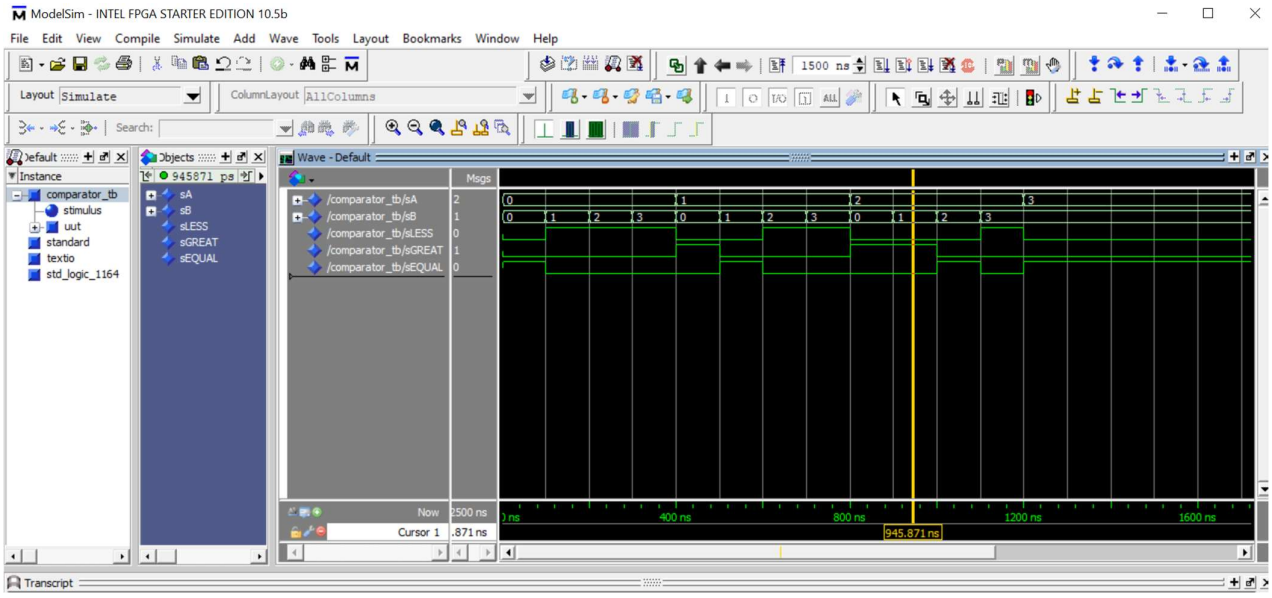
```
uut : entity work.<naziv_entiteta>(<naziv_arhitekture>)
   port map (
         -- mapiranje prolaza
    );
```

U nastavku je opisan test bench za dvobitni komparator koji testira arhitekturu opisanu u procesu IF-ELSE:

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

entity Comparator_tb is
end Comparator_tb;

architecture Test_tb of Comparator_tb is

	signal sA   : std_logic_vector(1 downto 0);
	signal sB   : std_logic_vector(1 downto 0);
	signal sLESS : std_logic;
	signal sGREAT : std_logic;
	signal sEQUAL : std_logic;

	component Comparator is
		port(
			iA     : in std_logic_vector(1 downto 0);
			iB     : in std_logic_vector(1 downto 0);
			oLESS  : out std_logic;
			oGREAT : out std_logic;
			oEQUAL : out std_logic
			);
	end component;
begin
	-- mapiranje komponente koja ima više različitih arhitektura
	uut : entity work.Comparator(if_else) port map (
		iA => sA,
		iB => sB,
		oLESS => sLESS,
		oGREAT => sGREAT,
		oEQUAL => sEQUAL
	 );

	-- zadavanje svih kombinacija ULAZNIH signala
	stimulus : process
	begin
		sA <= "00";
		sB <= "00";
		wait for 100 ns;
		sA <= "00";
		sB <= "01";
		-- TO DO: dodati sve kombinacije ulaznih signala A i B
		wait;
	end process stimulus;
end architecture;
```

Testirajte i ostale arhitekture koje su urađene u ovom zadatku i proverite njihovu ispravnost. Idealno bi bilo testirati sve kombinacije vrednosti ulaznih signala A i B (ukupno 16 kombinacija), tako da je potrebno dopuniti stimulus proces sa preostalim kombinacijama na označenom mestu u kodu sa **TO DO**.

U nastavku sledi prikaz izvršene simulacije dvobitnog komparatora:



Slika 4-2. Simulacija dvobitnog komparatora

Nakon uspešne simulacije, potrebno je testirati ovaj digitalni sistem na ploči MAX 10. U Tabeli 4-1 je dat predlog rasporeda pinova za ulazne i izlazne signale:

Port	Smer	Komponenta	Location	I/O standard
iA1	input	I_SW[3]	PIN_M1	3.3 V Schmitt Trigger
iA0	input	I_SW[2]	PIN_M2	3.3 V Schmitt Trigger
iB1	input	I_SW[1]	PIN_L3	3.3 V Schmitt Trigger
iB0	input	I_SW[0]	PIN_M3	3.3 V Schmitt Trigger
oLESS	output	o_sem_r	PIN_H13	3.3-V LVTTL
oGREAT	output	o_sem_y	PIN_E4	3.3-V LVTTL
oEQUAL	output	o_sem_g	PIN_B1	3.3-V LVTTL

ZAKLJUČAK

U ovoj vežbi ste naučili da opišete jednostavan digitalni sistem u jeziku za opis fizičke arhitekture VHDL. Takođe ste se upoznali sa osnovnom sintaksom VHDL-a, kao i sa različitim načinima dodele vrednosti signalu. Nakon završetka ove vežbe, trebali bi biti u stanju da implementirate jednostavan digitalni sistem uz pomoć VHDL-a, kao i da verifikujete opisani sistem simulacijom u ModelSim-Altera alatu i na platformi MAX 10.