POKAZNA VEŽBA 2
Kombinacione mreže

Potrebno predznanje

- Urađena pokazna vežba 1
- Standardne kombinacione mreže (dekoderi, koderi, multiplekseri, demultiplekseri, sabirači, itd.)

Šta će biti naučeno tokom izrade vežbe?

Nakon urađene vežbe, bićete u mogućnosti da:

- Opišete proizvoljnu kombinacionu mrežu u VHDL-u koristeći opis na nivou logičkih kola
- Opišete proizvoljnu kombinacionu mrežu u VHDL-u koristeći uslovne dodele
- Opišete proizvoljnu kombinacionu mrežu u VHDL-u koristeći kombinacione procese
- Sintetišete i implementirate vaš sistem za FPGA integrisano kolo

Apstrakt i motivacija

Jedna od najjednostavnijih matematičkih operacija je sabiranje. Implementacija digitalnog sistema koji vrši sabiranje dva broja može biti veoma mukotrpna. Posmatrajmo sabiranje dva 4-bitna broja, tj. dva broja koja vrednosti uzimaju u opsegu od 0 do 15. Rezultat ovog sabiranja je najviše 5-bitni broj, što znači da digitalni sistem koji vrši sabiranje dva 4-bitna broja računa 5 Bulovih funkcija od 8 promenljivih. Tradicionalni način implementacije ovog sistema bi zahtevao puno vremena – istinitosna tablica bi imala 256 redova. A sve ovo zarad sabiranja brojeva od 0 do 15! Zamislite kolika je istinitosna tablica za sabiranje dva 64-bitna broja, što je standard u današnjim računarima... U ovoj vežbi naučićemo da opišemo digitalni sistem na mnogo, *mnogo* jednostavniji način koristeći VHDL jezik i njegove konstrukcije, a alatu ćemo pustiti da automatski formira istinitosne tablice i Bulove funkcije za naš sistem. Krajnji proizvod će biti isti sistem, opisan u vremenu koji je za više redova veličine manji od vremena potrebnog za tradicionalno projektovanje. Xilinx ISE alat će umesto vas da otkrije Bulove funkcije koje vaš sistem treba da računa i implementira ih, omogućavajući vama da se koncentrišete na projektovanje logike sistema, a ne na surov fizički posao pisanja istinitosne tablice i crtanja logičke šeme.

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. Ovo i nije začuđujuće, s obzirom da funkcije rade na prostoru od samo 2 vrednosti – ukoliko je skup vrednosti jednostavan, potrebna složenost se oslikava u funkcijama. Projektovanje čak i ovih jednostavnih komponenti na nivou logičkih kola postaje veoma mukotrpno i bilo bi ljudski nemoguće na ovom nivou projektovati složene digitalne sisteme kao što je procesor.

Kako bi se olakšalo i time 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 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 korišteni jezici za opis fizičke arhitekture su Verilog, SystemVerilog i SystemC, ali njih nećemo koristiti u ovom predmetu.

Osnovni delovi VHDL opisa digitalnih sistema su:

- `use` – nabrajanje biblioteka koje se koriste (Xilinx ISE to za naše potrebe popuni automatski),
- `entity` – deklaracija entiteta onako kako se on vidi spolja, tj. naziv i spisak ulaza i izlaza entiteta digitalnog sistema,
- `architecture` – definicija arhitekture digitalnog sistema, tj. komponenata sistema i Bulovih funkcija koje sistem računa.

Listing 1-1 prikazuje osnovnu strukturu VHDL opisa digitalnog sistema.

Alat koji ćemo koristiti na ovim vežbama (Xilinx ISE) automatski generiše use deo opisa sistema i veoma retko ćete morati nešto da dodajete u ovom delu. Biblioteke koje alat automatski uključuje sadrže potrebne jezičke konstrukcije za definiciju osnovnih tipova signala koje možete koristiti u vašim sistemima, tj. tip signala koji predstavlja Bulovu promenljivu.

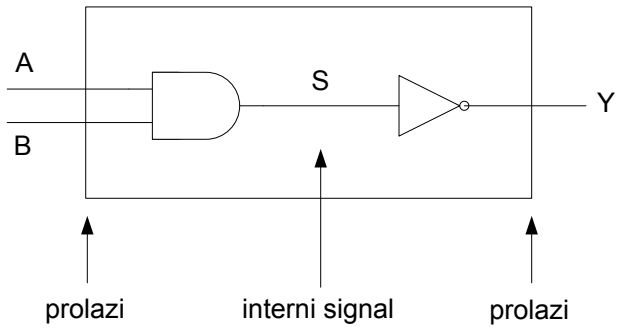
Ostale delove VHDL opisa ćemo objasniti kroz primer. Posmatrajmo sistem sa Slike 1-1 koji predstavlja digitalni sistem sastavljen od 2 logička kola.

Listing 1-1. Struktura VHDL opisa digitalnog sistema

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

entity <entityName> is
    port ( <portName1> : <portDirection1> <portType1>;
           <portName2> : <portDirection2> <portType2>;
           ...
           <portNameN> : <portDirectionN> <portTypeN> );
end <entityName>;

architecture <architectureName> of <entityName> is
    signal <signalName> : <signalType>;
    ...
begin
    <architectureBody>
end <architectureName>;
```



Slika 1-1. Primer jednostavnog digitalnog sistema

Entitet sistema je predstava sistema onako kako se on vidi spolja. Posmatrajte sistem sa slike 2-1 spolja, tako da nemate mogućnost da vidite unutrašnjost sistema. 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 1-1 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),
- *portName1* .. *portNameN* – nazivi prolaza sistema,
- *portDirection1* .. *portDirectionN* – smerovi prolaza sistema (**in** za ulaze i **out** za izlaze),
- *portType1* .. *portTypeN* – tipovi signala prolaza sistema.

Prilikom VHDL opisa digitalnog sistema, tip signala koji predstavlja jednu Bulovu promenljivu (signal koji uzima jednu od 2 vrednosti) se naziva **std_logic**. Ukoliko želite da više signala, odn. 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, u poslednjem delu VHDL opisa definiše se arhitektura sistema. Arhitektura sistema predstavlja unutrašnjost sistema, a njen opis podrazumeva opis ponašanja svih komponenata 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 1-1, videćete da se unutar sistema pojavljuje signal koji se ne vidi spolja. 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). VHDL konvencija dodeljivanja naziva predlaže da se internim signala dodeljuju nazivi sa malim početnim slovom *s*. Vodite računa da sistem komunicira sa svojom okolinom isključivo preko prolaza (ulaza/izlaza), odn. 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 signalu na izlazu iz logičke kapije je operator **<=**.

Listing 1-2 prikazuje opis sigitalnog sistema sa Slike 1-1.

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

```
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 MyFirstDigitalSystem;

architecture Behavioral of MyFirstDigitalSystem is
    signal sS : STD_LOGIC;
begin
    sS <= iA and iB;
    oY <= not(sS);
end Behavioral;
```

Na sličan način mogu se opisati i složenije kombinacione mreže, jednostavnim navođenjem svih logičkih kola koja se nalaze u sistemu, po jedno logičko kolo za svaku elementarnu Bulovu funkciju unutar sistema.

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 1-1 je moguće opisati i sledećom jednom dodelom, umesto dve:

```
Y <= not(A and B);
```

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 na tome da li će vas opis imati više jednostavnijih dodela ili manje složenijih je na vama, a ona bi trebala da ima optimalni odnos čitljivosti i veličine opisa.

2. Verifikacija rada digitalnih sistema – VHDL 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 ulaz u sistem koji proverava i kupi vrednosti izlaza u svoje interne signale, kako bi oni mogli biti analizirani od strane alata za simulaciju.

Pisanje većeg dela test bencha je izvan opsega ove vežbe i mi ćemo koristiti automatski generisan test bench od strane alata Xilinx ISE u kome radimo. Jedini deo test bencha koji ćete vi pisati je **stimulus**, tj. sekvenca vrednosti ulaznih signala sa kojom želite da proverite vaš sistem.

Za razliku od opisa samog sistema, test bench opisuje **sekvencu** vrednosti ulaznih signala, čime se poprilično razlikuje od samog opisa digitalnog sistema koji nema sekvencijalnost. Test bench uvodi sekvencijalnost u svoj opis zbog potrebe da se u njemu definiše **vremenski redosled signala**, odn. 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.

Kao primer, napišimo test bench za sistem sa slike 1-1. Pretpostavimo da želimo da proverimo sistem 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.

Operator dodele u VHDL-u može da se iskoristi za dodelu vrednosti signalu unutar stimulus dela test bencha. Konstante vrednosti (literali) tipa *std_logic* se navode između apostrofa, dok se konstantne vrednosti tipa *std_logic_vector* navode između navodnika.

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.

Sekvenca ulaznih vrednosti sa kojima želimo da proverimo sistem treba biti navedena unutar **stimulus procesa** u VHDL test benchu. Nakon što alat automatski napravi strukturu test bencha, vaše je samo da unutar stimulus procesa opišete sekvencu ulaza za proveru. Listing 2-1 prikazuje primer opisa ranije navedene sekvence za sistem sa slike 1-1.

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 stanja ulaznih signala zadrži beskonačno, odn. do kraja simulacije. U suprotnom, simulacija će stati istog trenutka i verovatno nećete biti u mogućnosti da vidite izlaz za poslednju kombinaciju ulaza.

Listing 2-1. Sekvenca ulaza za proveru digitalnog sistema

```
-- stimulus process --

process
begin

	iA <= '0';
	iB <= '0';
	wait for 100 ns;

	iA <= '0';
	iB <= '1';
	wait for 100 ns;

	iA <= '1';
	iB <= '0';
	wait for 100 ns;

	iA <= '1';
	iB <= '1';

	wait;
end process;
```

3. VHDL opis kombinacionih mreža

VHDL jezik omogućuje nekoliko načina za opis kombinacionih mreža, koji su kraći i čitljiviji od opisa na nivou logičkih kola. Ovi opisi podsećaju na opis programske podrške pošto koriste iste ili slične konstrukte. Kombinacione mreže se u VHDL jeziku mogu opisati sledećim načinima:

- Dodelom vrednosti signalu na nivou logičkih kola (način koji smo koristili u prethodnim vežbama),
- Dodelom vrednosti signalu pomoću ostalih VHDL operatora,
- Uslovnom dodelom vrednosti signalu,
- Pomoću kombinacionih procesa.

3.1. Dodela vrednosti

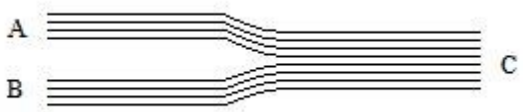
Signalu se u VHDL jeziku vrednost može dodeliti pomoću operatora **<= .**

Na nivou logičkih kola vrednost signala se definiše navodeći logički izraz sa desne strane, koristeći VHDL logičke operatore **not**, **and**, **or**, **xor**, **nand**, **nor**, **xnor**, kao što smo radili u prethodnim vežbama.

Prilikom definisanja (dodele) vrednosti nekom signalu, mogu se koristiti i ostali VHDL operatori. Trenutno je za nas najzanimljiviji operator **konkatenacije (&).** Operator konkatenacije 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:

```
C <= '1' & A(1 downto 0) & B(3 downto 2) & "00";
```

definišemo signal C širine 7 bita, i to jednog bita vrednosti konstante '1', dva bita iz signala A, dva bita iz signala B i dva bita vrednosti konstante '0'.

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> <= <expression1> when <condition1> else
                <expression2> when <condition2> else
                ...
                <expressionN> when <conditionN> else
                <expressionDefault>;
```

Signal <signalName> dobija vrednost <expression1> ukoliko je ispunjen uslov <condition1>, u suprotnom dobija vrednost <expression2> ukoliko je ispunjen uslov <condition2>, itd. Ukoliko nijedan od uslova nije ispunjen, signal dobija vrednost <expressionDefault>.

Vrednosti sa desne strane se mogu definisati pomoću VHDL operatora kao u prethodnoj sekciji, a za definisanje uslova mogu se koristiti VHDL **relacioni operatori**:

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

3.3. Kombinacioni procesi

Sem pomoću operatora dodele vrednosti, kojim se definiše vrednost izlaza kombinacione mreže na osnovu vrednosti ulaza, kombinaciona mreža se može opisati i pomoću **kombinacionog procesa**. Listing 3-2 prikazuje sintaksu za VHDL opis kombinacione mreže pomoću procesa.

Listing 3-2. Sintaksa VHDL kombinacionog procesa

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

Unutar zagrada u prvoj liniji procesa definiše se **lista osetljivosti** procesa. Lista osetljivosti služi alatu za simulaciju kako bi znao kada komponenta (kombinaciona mreža) koja se opisuje procesom može da promeni izlaznu vrednost. Prema osobinama kombinacione mreže, promena izlazne vrednosti se može desiti prilikom promene bilo kog od ulaza, pošto izlaz uvek predstavlja funkciju trenutnih vrednosti ulaza. Dakle, u listi osetljivosti kombinacionog procesa treba da budu navedeni **svi ulazi** kombinacione mreže koja se procesom opisuje, odvojeni zarezom.

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 VHDL konstrukcije koje se mogu koristiti samo u procesima, a najčešće korištene su uslovne konstrukcije IF i CASE. Listing 3-3 i Listing 3-4 prikazuju sintaksu ovih konstrukcija.

Listing 3-3. Sintaksa IF strukture

```
if (<condition1>) then
    <statements>
elsif (<condition2>) then
    <statements>
else
    <statements>
end if;
```

Listing 3-4. Sintaksa CASE strukture

```
case (<signalName>) is
    when <value1> => <statements>
    when <value2> => <statements>
    ...
    when <valueN> => <statements>
    when others => <statements>
end case;
```

IF i CASE struktura imaju istu logiku kao i u programskim jezicima. Samo iskazi koji su u delu strukture za koju je uslov zadovoljen će definisati ponašanje izlaza kombinacione mreže.

Prilikom definisanja kombinacionih mreža pomoću procesa koji sadrže IF i CASE, neophodno je definisati svaki izlaz u svakoj grani IF i CASE strukture. U suprotnom kombinaciona mreža je nedovoljno definisana i rezultovaće lošim digitalnim sistemom. Takođe treba izbegavati paralelne IF i CASE strukture pošto svaki izlaz definiše jedna logička funkcija, a svaka IF i CASE struktura predstavlja različitu logičku funkciju.

Za spisak ostalih struktura koje se mogu naći u telu procesa preporučujemo da konsultujete udžbenik i internet, pošto oni izlaze van opsega ove vežbe.

3.4. Koja je razlika ranijih načina opisivanja?

Naveli smo dva pristupa opisivanju kombinacionih mreža – pomoću operatora dodele (što se još naziva i konkurentnim iskazom) i pomoću kombinacionih procesa. Između navedenih načina opisivanja kombinacionih mreža **nema razlike**, odn. oba načina 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 i vašim preferencama.

ZADACI

4. Implementacija našeg prvog digitalnih sistema

Pre nego što počnemo da implementiramo složenije kombinacione mreže, upoznaćemo se sa procedurom implementacije digitalnih sistema za FPGA integrisana kola. E2LP platforma, koju ćemo koristiti na ovim vežbama, kao sastavni deo sadrži FPGA integrisano kolo koje se može konfigurisati da se ponaša kao proizvoljan digitalni sistem. Xilinx ISE alat omogućava da se opisani digitalni sistem VHDL jezikom implementira za određeno FPGA integrisano kolo, odn. da se generiše konfiguraciona datoteka sa kojom može da se izvrši konfiguracija FPGA integrisanog kola kako bi se on ponašao kao željeni projektovani digitalni sistem.

Za početak, otvorite projekat iz prethodne vežbe, vaš prvi digitalni sistem, dvostrukim klikom na datoteku *MyFirstDigitalSystem.xise*. Ovime ćete otvoriti Xilinx ISE alat i odgovarajući projekat.

Ovog puta treba da izaberete pogled za implementaciju sistema (View: Implementation). Pogled birate na vrhu Design prozora. Pogled za implementaciju biramo zbog toga što će nam u njemu biti date komande za implementaciju sistema za FPGA. Primetite da u ovom pogledu nisu prokazani VHDL testbench-evi jer oni ne služe implementaciji nego simulaciji sistema.

Xilinx ISE alat vrši implementaciju sistema u tri koraka:

- sinteza (*synthesis*)
- implementacija (*implementation*)
- generisanje datoteke za konfiguraciju FPGA (*generate programming file*).

U donjem delu Design prozora, pretpostavljajući da je u gornjem selektovana odgovarajuća VHDL datoteka sa opisom sistema, biće prikazane gore navedene opcije.

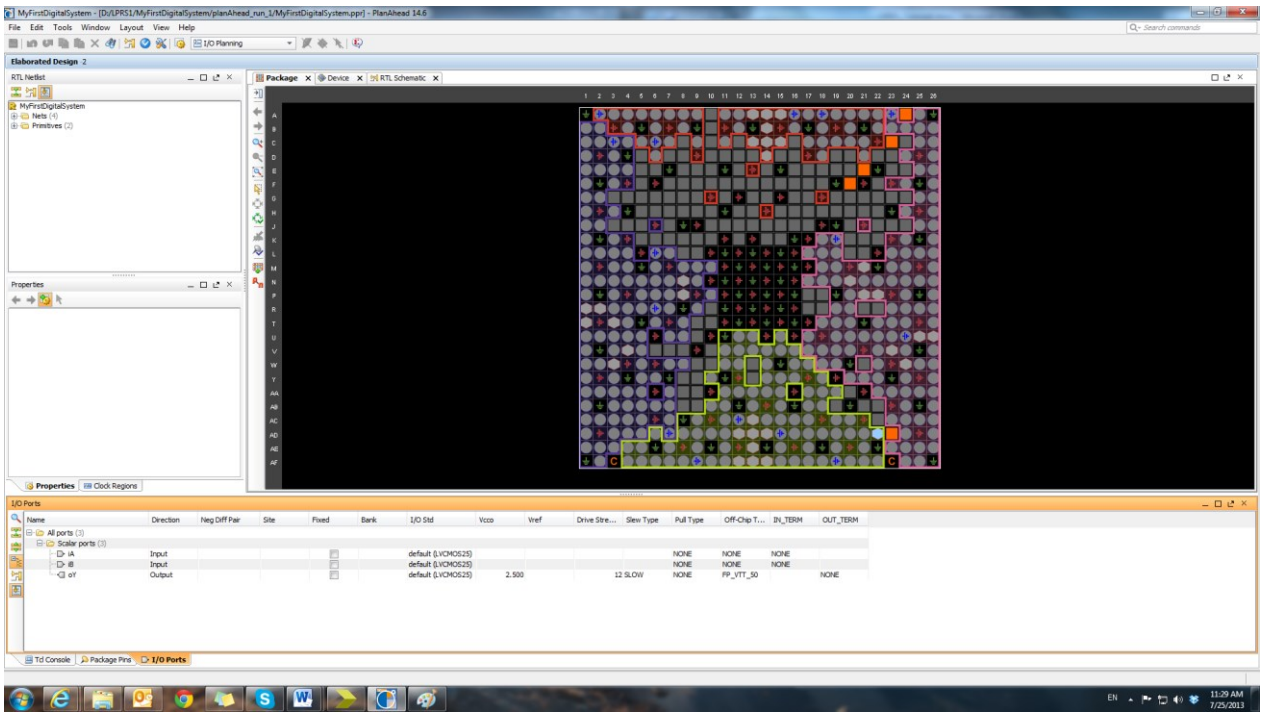
Za početak, pokrenite sintezu sistema klikom na **Synthesize – XST**. Ova opcija pokreće alat za sintezu vašeg sistema i ukoliko sinteza bude uspešna, pojaviće se zeleni checkmark potvrde. Ukoliko u vašem opisu sistema postoje greške, alat će ih prikazati u konzolnom prozoru.

Ukoliko nema grešaka, sledeći korak je dodela pinova FPGA vašim ulazima i izlazima. Zbog čega ovo radimo? FPGA integrisano kolo je povezano sa ostatkom E2LP platforme preko svojih nožica, odn. pinova, koje možemo posmatrati kao ulaze i izlaze samog FPGA integrisanog kola. Svaki pin je povezan na neku od komponenti E2LP platforme, npr. na prekidače, tastere, LED-ove, LCD itd. Kada želimo da implementiramo sistem za FPGA, mi moramo alatu naglasiti na koji pin na poveže koji ulaz i izlaz našeg sistema. Izbor pinova vršimo tako da naš sistem bude proverljiv na platformi, odn. ulaze ćemo povezati na neki od prekidača ili tastera, a izlaze najčešće na neke od dioda. Ako naš sistem radi sa LCD ekranom ili nekim drugom složenijom periferijom koja postoji na platformi (npr. sa HDMI izlazom ili audio sistemom), onda ulaze i izlaze sistema treba povezati na odgovarajuće pinove koji su na platformi fizički povezani sa odgovarajućom periferijom.

Za primer našeg prvog digitalnog sistema, ulaze ćemo povezati na prekidače, a izlaze na LED-ove, prema sledećoj tabeli:

Port	Smer	Komponenta	FPGA pin
iA	in	SW0	W19
iB	in	SW1	Y24
iC	in	SW2	K19
oY	out	LED0	N24

Dodela pinova ulazima i izlazima sistema se vrši u alatu Xilinx PlanAhead. Ovaj alat može biti pokrenut iz Xilinx ISE-a. U donjem delu Design prozora pronađite opciju **User Constraints**, kliknite na plus pored i izaberite opciju **I/O Pin Planning (PlanAhead) – Pre-Synthesis**. Alat izgleda kao na slici 4-1.



Slika 4-1. Xilinx PlanAhead

Glavni deo prozora ovog alata predstavlja 2D pregled pinova FPGA. A kako alat zna za koji FPGA mi projektujemo sistem pa nam je sada dao odgovarajući raspored pinova? Sećate li se koraka prilikom pravljenja projekat koji smo u prošloj vežbi preskočili? U tom koraku se alatu definiše za koji FPGA projektujemo sistem, a tu informaciju alat koristi prilikom implementacije sistema, a i konkretno u ovom slučaju prilikom dodele pinova našem sistemu.

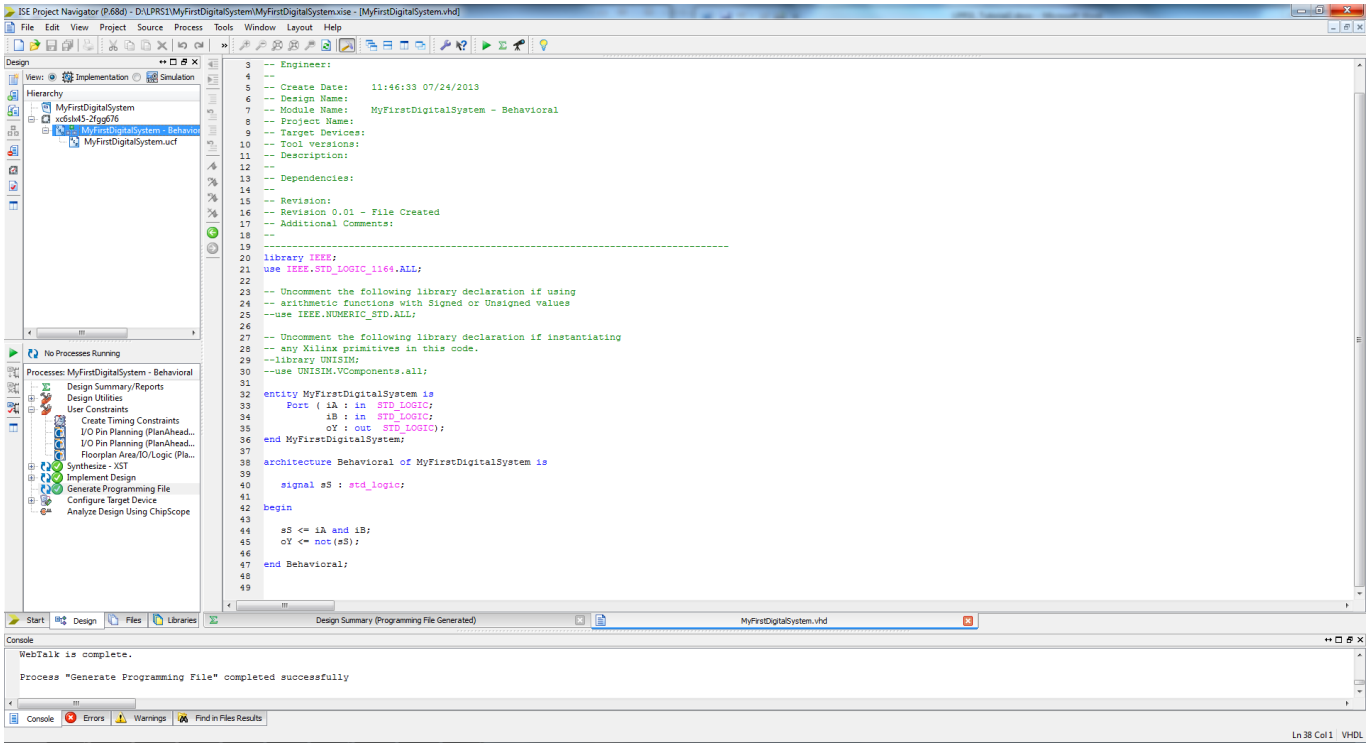
U donjem delu PlanAhead glavnog prozora možete dodeliti pinove FPGA vašim ulazima i izlazima. Ukoliko izaberete *I/O Ports* tab, pojaviće se lista svih ulaza i izlaza sistema. Skalarni ulazi i izlazi sistema su grupisani odvojeno od vektorskih. Za svaki ulaz i izlaz treba u koloni **Site** napisati sa kojim pinom ga povezujete. Nakon dodele svih pinova, kliknite na *Save* dugme i izađite iz alata.

Rezultat PlanAhead alata je datoteka sa ekstenzijom *.ucf* koja predstavlja datoteku korisničkih ograničenja sistema (*User Constraints File*). Ova datoteka definiše alatu neka ograničenja prilikom implementacije sistema, a jedan od primera je i dodela pinova ulazima i izlazima. Dodela pinova se smatra ograničenjem zato što, ukoliko vi ne navedete pinove za svaki ulaz i izlaz, alat će nasumično izabrati pinove za sve ulaze i izlaze za koje vi pin niste

eksplicitno izabrali. Datoteka se pojavljuje u gornjem delu Design prozora i vi je odatle možete otvoriti i menjati. Sintaksa dodele pinova je jednostavna, i možete je menjati a i formirati ručno. Pin se dodeljuje na sledeći način:

```
NET "<portName>" LOC = <pinName>;
```

Nakon dodele pinova vašim prolazima sistema, kliknite na opciju **Implement Design**. Ova opcija pokreće prevođenje vašeg opisa, mapiranje na elemente unutar FPGA i tzv. rutiranje signala unutar FPGA. Poslednji korak je klik na **Generate Programming File** čiji je rezultat datoteka sa ekstenzijom *.bit* koja se koristi za konfiguraciju FPGA. Ovi koraci se ne moraju pokretati odvojeno, klikom na poslednji korak, svaki prethodni koji nije urađen biće pokrenut automatski.



Slika 4-2. Glavni prozor Xilinx ISE alata nakon uspešne implementacije sistema

Nakon toga se može izvršiti sama konfiguracija E2LP platforme pomoću dobijene *.bit* datoteke.

5. Multiplekser

Multiplekser je kombinaciona mreža koja koja implementira uslovnu logiku – vrednost izlaza multipleksera je jednaka vrednosti jednog od njegovih ulaza. Koji ulaz se prosleđuje na izlaz zavisi od stanja selekcionih bita.

Vaš zadatak je da implementirate 4x1 multiplekser (4 ulaza opšte namene, 2 selekciona ulaza i 1 izlaz) na sledeće načine:

- definisanjem tabele istinitosti, Bulove jednačine izlaza i crtanjem logičke šeme
- opisom u VHDL-u na nivou logičkih kola
- opisom u VHDL-u koristeći uslovnu dodelu
- opisom u VHDL-u koristeći kombinacioni proces.

6. Primena multipleksera u složenijem sistemu

Vaš naredni zadatak je da implementirate složeniji digitalni sistem, sa dva 4-bit ulaza (iA i iB), jednim selekcionim ulazom od 2 bita (iSEL) i jednim 4-bit izlazom (oC), tako da se sistem ponaša prema sledećoj jednačini:

$$oC = \begin{cases} iA & if\ iSEL = \text{"00"} \\ iB & if\ iSEL = \text{"01"} \\ iA + iB & if\ iSEL = \text{"10"} \\ iA + 1 & if\ iSEL = \text{"11"} \end{cases}$$

Primetite da ovaj sistem sadrži 10 ulaza i 4 izlaza, što znači da on računa 4 Bulove funkcije od 10 promenljivih. Tradicionalni pristup projektovanju digitalnih sistema kroz istinitosne tablice i crtanjem logičke šeme bi trajao veoma dugo, a VHDL nam omogućuje opis istog sistema u samo nekoliko linija!

Ali nikada ne treba zaboraviti da ono što mi opisujemo VHDL-om jesu logička kola i jesu Bulove funkcije izvedene iz istinitosne tablice! VHDL je samo drugi (jednostavniji) način da se opiše isti sistem.

Prilikom implementacije sistema povežite ulaze iA i iB na prekidače, selekcione ulaze na tastere, a izlaz na LED-ove po izboru.

ZAKLJUČAK

U ovoj vežbi naučili smo da opisujemo kombinacione mreže na intuitivniji način nego što su istinitosne tablice ili Bulove funkcije. Koristeći uslovne dodele i kombinacione procese moguće je opisati veoma složenu Bulovu funkciju na kraći i razumljiviji način. Nakon kompletiranja ove vežbe, trebali bi biti u stanju da opišete proizvoljno složenu kombinacionu mrežu koristeći VHDL konstrukcije, ali i tradicionalno, pomoću Bulovih funkcija.