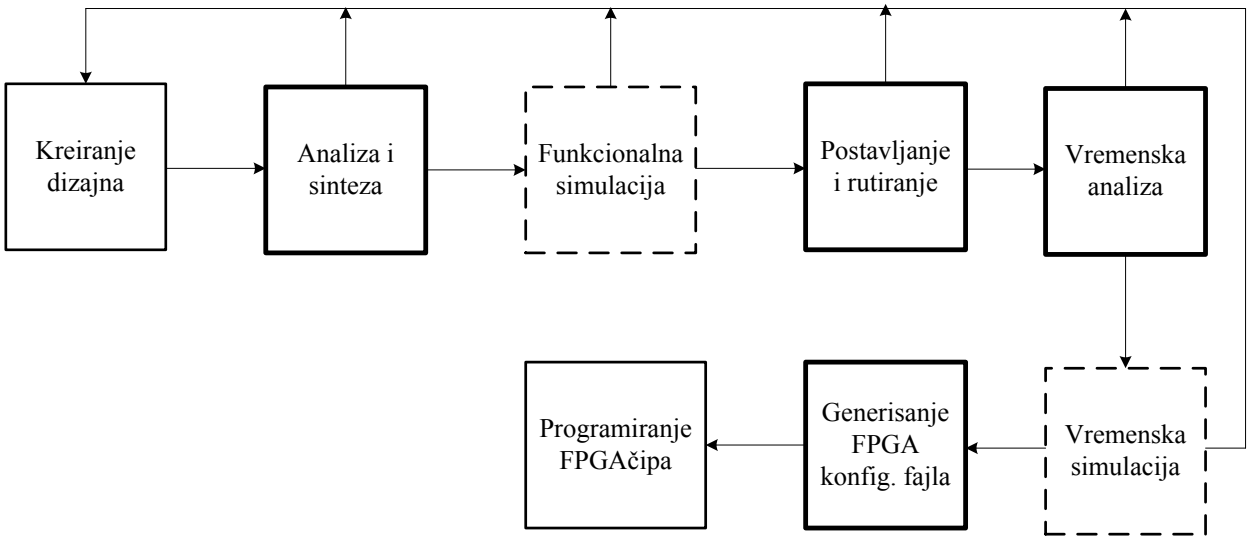
PROGRAMIRANJE KOMUNIKACIONOG HARDVERA
– Poglavlje 3 (deo 1) –

3 Razvojna okruženja

U okviru ovog poglavlja će biti izloženi osnovni principi projektovanja hardverskog dizajna, kompajliranja projektovanog dizajna i programiranja FPGA čipa kompajliranim dizajnom. Takođe će biti dat osnovni pregled funkcionalnosti i principa rada u razvojnim okruženjima dva najpoznatija proizvođača FPGA čipova Xilinx i Altera. U slučaju razvojnog okruženja namenjenog za razvoj dizajna za Altera FPGA čipove biće opisan Quartus 9.0 softverski paket, a u slučaju razvoja za Xilinx FPGA čipove biće opisan ISE 12.2 softverski paket. Oba paketa će biti opisana u potrebnoj meri tako da polaznici kursa mogu nakon završetka kursa da programiraju i veoma složene projekte. Naravno, zbog veličine i kompleksnosti navedenih softverskih paketa u ovim skriptama neće biti opisane sve mogućnosti ovih paketa.

3.1 Tok projekta hardverskog dizajna na FPGA čip

Kada se projektuje određeni hardverski dizajn za FPGA čip neophodno je slediti određeni redosled operacija u projektu. Naravno, prvo treba osmisliti rešenje i zatim ga programirati, na primer, upotrebom VHDL jezika. Potom treba kreirani dizajn kompajlirati, pri čemu se i samo kompajliranje sastoji od više etapa. Pri tome je neophodno izvršiti i testiranje dizajna radi potvrđivanja njegove ispravne funkcionalnosti. Na kraju kada smo se uverili da je dizajn korektan i izvršava željeni skup funkcija, treba izvršiti programiranje FPGA čipa tj. žargonski rečeno na njega treba spustiti realizovani dizajn.



Slika 3.1.1. – Tok projekta hardverskog dizajna FPGA čipa

Slika 3.1.1 precizno prikazuje tok projekta hardverskog dizajna namenjenog programiranju FPGA čipa. Prvi korak je kreiranje dizajna. Dizajn se može kreirati na više načina, upotrebom nekog od tekstualnih programskih jezika kao što su VHDL ili Verilog, ali i kreiranjem grafičkog (šematskog) opisa dizajna upotrebom grafičkih editora koja tipično nude sama razvojna okruženja (Quartus, ISE), ali i posebni alati poput OrCAD i sličnih. Kada se kreira dizajn, neophodno je kompajlirati dizajn da bi se on pretvorio u konfiguracioni fajl

pomoću koga će se izvršiti programiranje FPGA čipa. Delovi toka projekta koji su označeni podebljanim linijama predstavljaju etape u kompajliranju.

Prvi korak u kompajliranju je izvšavanje analize i sinteze. U ovom koraku se prvo proverava sintaksa unetog dizajna. Na primer, ako je dizajn kreiran upotrebom VHDL jezika, proverava se da li kod zadovoljava sva sintaksna pravila VHDL jezika. Ukoliko se detektuju greške, kompajliranje se zaustavlja i korisnik se obaveštava o grešci zbog koje je zaustavljeno kompajliranje. Pored sintaksnih grešaka, mogu da postoje i logičke greške poput dodeljivanja više vrednosti istom signalu unutar na primer konkurentnog koda u VHDL jeziku. Bilo kog tipa da je greška, kompajliranje će biti zaustavljeno i korisnik će biti obavešten o tipu greške koja je uočena. Tada se korisnik vraća na prvi korak, a to je kreiranje dizajna, koje se u ovom slučaju sastoji od korigovanja greške u dizajnu. Tipično, razvojno okruženje tj. kompajler ne obaveštava odmah o svim greškama koje postoje u dizajnu, već samo na one na koje je naišao do momenta zaustavljanja kompajliranja, što znači da možda postoje i druge greške u dizajnu osim onih o kojima je korisnik obavešten. Na ove greške će se naići nakon što korisnik koriguje do tada otkrivene greške i ponovo pokrene kompajliranje dizajna. Ukoliko nema grešaka, kompajler će izvršiti analizu dizajna, tako što će raščlaniti dizajn na osnovne primitivne delove (flip-flopove, osnovne logičke funkcije (I, ILI, NE i dr.)...). Analiza je pored, naravno, otkrivanja grešaka, pre svega bitna zbog toga što mnogi metodi dizajniranja poput, na primer, VHDL jezika omogućavaju opisivanje ponašanja dizajna, a ne samo strukture dizajna. Otuda je analiza bitna jer ona iz opisa ponašanja izvodi strukturu dizajna. Nakon analize dizajna, kompajler će izvršiti sintezu dizajna koja rezultuje opisom dizajna u vidu mreže primitivnih komponenti koje su međusobno povezane linijama ili magistralama. Rezultat sinteze je tzv. netlista koja predstavlja mrežu linija koje povezuju primitivne elemente dizajna. Ovim je u potpunosti opisan realizovani hardverski dizajn. Ono što je takođe važno napomenuti je da se u procesu sinteze obavlja i optimizacija, koja pokušava da sažme određene delove dizajna ukoliko je to moguće, ili koja formira dizajn tako da kombinacione logike imaju što manja kašnjenja i sl. Tipično, razvojna okruženja nude podešavanje da se dizajn optimizuje po kašnjenju, po resursima ili balansiranje ova dva navedena kriterijuma. U slučaju optimizacije po kašnjenju cilj sinteze je da kreira dizajn tako da kašnjenje najduže putanje u dizajnu između neka dva međusobno povezana flip-flopa (registra) ili flip-flopa i pina FPGA čipa bude minimizovano. Ovde neki delovi dizajna mogu biti namerno duplirani u procesu sinteze da bi se ostvarilo minimalno kašnjenje. U slučaju optimizacije po prostoru cilj sinteze je da kreira dizajn tako da on zauzima što manje resursa.

Čak i ako se proces analize i sinteze završi uspešno, to još ne znači da je dizajn korektan. Naime, možda postoje i neke logičke greške u dizajnu koje utiču na to da projektovani dizajn pogrešno izvršava svoju funkciju. Banalan primer je da ukoliko smo želeli da formiramo I kolo, a greškom smo upotrebili ILI kolo, analiza i sinteza će uspešno da se izvrše, ali realizovani dizajn neće obavljati funkciju koju želimo. Stoga je potrebno izvršiti postupak funkcionalne simulacije. Ovaj postupak nije deo kompajliranja i nije obavezan, ali ipak ga treba uraditi jer on verifikuje ispravnost rada dizajna u idealnim uslovima kada sve linije i elementi u dizajnu nemaju kašnjenje (odnosno kašnjenje je nula). U okviru funkcionalne analize vrši se stimulisanje dizajna dobijenog postupkom sinteze odgovarajućim stimulansima. U stvari, na ulaze dizajna se dovode odgovarajuće vrednosti kojima se želi stimulisati rad dizajna u različitim situacijama. Na primer, ako bi testirali dizajn dvoulaznog I kola, na ulaz bi trebali dovesti sve moguće kombinacije vrednosti ulaza i posmatrati izlaz za svaku kombinaciju i porediti sa tačnim vrednostima. Ukoliko se simulirane vrednosti na izlazu poklapaju sa tačnim (očekivanim) vrednostima, onda smo sigurni da je dizajn funkcionalano ispravan, odnosno da u idealnim

uslovima kada nema drugih uticaja on radi kako treba, tj. logički je ispravan. Naravno, u slučaju složenog dizajna sa velikim brojem ulaza i izlaza, nije moguće pokriti sve kombinacije. Ali, tada treba osmisliti testove kojim će se pokriti većina situacija. Na primer, ako dizajnirani blok treba da vrši testiranje ispravnosti IP zaglavlja datagrama, onda ti testovi treba da pokriju sve kombinacije mogućih tipova grešaka, kao i slučajeve ispravnih zaglavlja. U praksi često tim koji radi dizajn i tim koji testira dizajn nisu isti, jer tim koji radi dizajn zna kako dizajn funkcioniše i to neretko dovodi do toga da se testovi nekih situacija svesno ili nesvesno preskoče. Naravno, tokom razvoja dizajna, tim koji radi taj razvoj mora da izvrši osnovna testiranja, ali finalnu verziju dizajna uglavnom testira drugi tim. Ukoliko se uoče greške u funkcionalnoj simulaciji, ponovo se vraćamo na prvi korak gde se sada koriguju otkrivene logičke greške.

Nakon što smo utvrdili funkcionalnom simulacijom da je dizajn ispravan u idealnim uslovima, kreće se na sledeći korak kompajliranja, a to je postavljanje i rutiranje. Tek u ovom koraku se koristi informacija o samom FPGA čipu. U prethodnim koracima nije vođeno računa koji FPGA čip će biti programiran realizovanim dizajnom, a ovde je to esencijalno. U okviru postavljanja dizajna na čip vrši se raspoređivanje elemenata dizajna dobijenih procesom sinteze na CLB blokove i/ili posebne interne delove (interne memorije, množači, PLL-ovi) FPGA čipa. U procesu rutiranja se vrši povezivanje postavljenih delova preko internih magistrala FPGA čipa tako da se ostvare što manja kašnjenja. U okviru ovog dela se radi složen proces optimizacije koja za cilj ima da što optimalnije rasporedi dizajn na dostupne resurse FPGA čipa i da se delovi dizajna povežu na što je moguće efikasniji način. Ova problematika je veoma složena naročito u slučaju velikog dizajna koji zauzima velike resurse, tako da pored razvojnih okruženja koja nude sami proizvođači FPGA čipova, postoje i posebne kompanije koje nude svoja tzv. '*third-party*' rešenja koja ostvaruju bolja konačna rešenja postavljanja i rutiranja. Naravno, ukoliko je dizajn prevelik da stane na čip, kompajler će prijaviti grešku i zaustaviti dalje kompajliranje. Tada, se opet vraćamo na prvi korak gde korigujemo dizajn. Dizajn korigujemo ili tako što ga smanjujemo (optimalnijim rešenjima ili izbacivanjem nekih funkcija) ili tako što biramo FPGA čip sa većom količinom resursa na raspolaganju.

Nakon što je dizajn postavljen na FPGA čip i izvršeno je međusobno povezivanje svih delova dizajna, vrši se vremenska analiza dizajna. Vremenska analiza uzima u obzir realne karakteristike izabranog FPGA čipa i vrši analizu svih putanja i linija u dizajnu u cilju određivanja najgorih slučajeva. Kao rezultat se dobija maksimalna frekvencija koja se može koristiti u sekvencijalnoj logici, kao i maksimalno kašnjenje sa stanovišta svih kombinacionih logika. Ukoliko vremenska analiza pokaže da dizajn ne zadovoljava željene vremenske kriterijume (na primer, maksimalna frekvencija dizajna je manja od one koju smo nameravali da koristimo), ponovo se vraćamo na prvi korak i vršimo korekcije dizajna.

Vremenska simulacija je slična funkcionalnoj analizi, samo što umesto idealnih uslova, u obzir se uzimaju realni uslovi, odnosno karakteristike izabranog FPGA čipa. Principi simulacije su identični onima iz funkcionalne simulacije, sa osnovnom razlikom što se sada u simulaciji uzimaju u obzir realni uslovi rada dizajna. U principu ova simulacija nije od velikog značaja, jer na osnovu funkcionalne simulacije znamo da li je dizajn logički ispravan ili ne, a na osnovu vremenske analize znamo da li su vremenski uslovi dizajna ispunjeni ili ne.

Kada je dizajn napokon verifikovan kao ispravan, ostaje poslednji korak kompajliranja, a to je generisanje konfiguracionog fajla kojim će se programirati FPGA čip. Format ovog fajla zavisi od samih proizvođača. Nakon formiranja ovog fajla, vrši se programiranje FPGA čipa. Programiranje se može izvršiti i sa samog računara, a takođe i sam konfiguracioni fajl se može

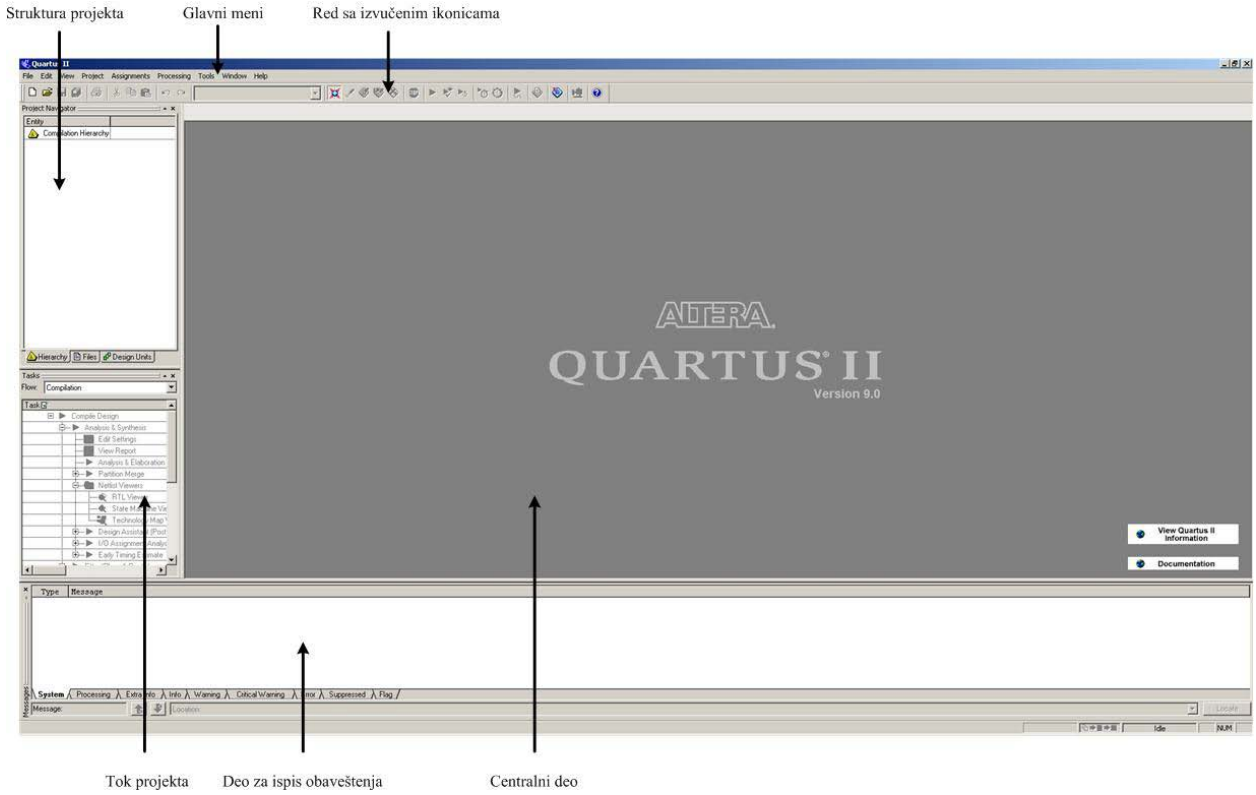
smestiti u PROM memoriji iz koje će, po dobijanju napajanja, FPGA čip povući svoju konfiguraciju tj. programirati se.

3.2 Quartus II softverski paket

Quartus II softverski paket predstavlja razvojno okruženje za FPGA čipove kompanije Altera. Trenutno poslednja verzija je 12.1, koja je izdata u novembru 2012. U okviru ovih skripti biće opisana verzija 9.0 softverskog paketa Quartus II. Opis razvojnog okruženja će pratiti kreiranje jednog konkretnog projekta, formiranje sastavnih delova projekta i na kraju kompajliranje kreiranog projekta. Na ovaj način će na praktičnom primeru biti prikazani osnovni delovi Quartus paketa i načini njihovog korišćenja da bi se kreirala jedna hardverska implementacija koja treba da se spusti na FPGA čip.

3.2.1. Glavni prozor Quartus aplikacije

Po pokretanju Quartus softvera, dobija se sledeći prikaz prozora u kome je otvorena Quartus aplikacija (slika 3.2.1.1.).



Slika 3.2.1.1. – Glavni prozor Quartus aplikacije

Glavni prozor sadrži glavni meni, red sa izvučenim ikonicama, struktura projekta, tok projekta, centralni deo i deo za ispis obaveštenja. Glavni meni grupiše sve opcije u srodne grupe i preko njega se može pristupiti svim opcijama koje nudi Quartus. Grupe opcija su raspoređene u **File**, **Edit**, **View**, **Project**, **Assignments**, **Processing**, **Tools**, **Window** i **Help**. **File** grupa opcija sadrži osnovne operacije za kreiranje/otvaranje/snimanje fajlova i projekata. **Edit** grupa opcija se

odnosi na opcije koje se koriste u editovanju otvorenih fajlova poput kopiranja, *paste* opcije, *undo* opcije i dr. **View** grupa opcija daje mogućnost podešavanja prozora, na primer, prikaz na celom ekranu, otvaranje tzv. *utility* prozora i dr. **Project** grupa opcija se koristi za menadžment projekta, gde se kreiraju revizije projekta, dodaju novi fajlovi u projekat i dr. **Assignments** grupa opcija omogućava podešavanje parametara projekta, poput izbora FPGA čipa, definisanja na koje pinove FPGA čipa treba da se vežu portovi dizajna, definisanja parametara kompajliranja i dr. **Processing** grupa opcija pokreće potpuno ili delimično kompajliranje i posebne alate poput analize potrošnje snage realizovanog dizajna, simulacije i dr. **Tools** grupa opcija omogućava aktivaciju posebnih alata poput programera koji programira FGPA čip, alati za uvid u vizuelni prikaz implementiranog dizajna na FPGA čipu, alati za pokretanje simulatora drugih kompanija i dr. **Window** grupa opcija služi za menadžment prozora, na primer da se neki prozor odvoji od glavnog prozora u poseban prozor i sl. **Help** grupa opcija kao što i sam naziv kaže se koristi za nalaženje pomoći u vidu dodatne dokumentacije.

Red sa izvučenim ikonicama sadrži najvažnije opcije iz glavnog menija u vidu ikonica čime se omogućava brz pristup najvažnijim opcijama. Neke od najvažnijih opcija koje su stavljene u ovaj red po difoltu su otvaranje i snimanje fajlova, pokretanje potpunog kompajliranja ili samo prve etape kompajliranja, pokretanje vremenske analize i dr. Pokretanjem pokazivača iznad neke od ikonica u oblačiću se ispisuje naziv dotične ikonice, odnosno opcije koju dotična ikonica predstavlja. Mogu se dodavati i druge ikonice, pri čemu u slučaju većeg broja ikonica, kreiraće se više redova koji sadrže ikonice.

Struktura projekta je deo glavnog prozora u kome se nalaze informacije o strukturi projekta. Postoje tri taba za ovaj deo prozora **Hierarchy**, **Files** i **Design Units**. **Hieararchy** prikazuje hijerarhiju projekta, odnosno relacije između pojedinih delova projekta. Na ovaj način se jednostavno dobija uvid šta je na vrhu hijerarhije i za svaki dizajnirani deo se vidi ko mu je roditelj, a ko su mu deca. U slučaju VHDL baziranog projekta se na jednostavan način može uvideti u kom od VHDL entiteta su korišćene komponente i koje su tačno komponente korišćene (na osnovu toga ko su deca), takođe se za svaki entitet može videti gde je on uključen kao komponenta (na osnovu toga ko je roditelj). **Files** daje pregled svih fajlova koji su uključeni u projekat. U slučaju da neki fajl tj. hardverski deo koji on definiše nije deo hijerarhijske strukture, to znači da taj fajl neće biti korišćen u kompajliranju i da je najverovatnije ili greškom uključen u projekat i treba ga izbrisati ili postoji greška u dizajniranju ako je dotični fajl trebao da bude uključen u hijerarhijsku strukturu. **Design Units** daje pregled korisničkih paketa, kao i entiteta i njihovih arhitektura u slučaju VHDL koda, odnosno može se reći da se ovde daje pregled realizovanih hardverskih celina u projektu.

Tok projekta predstavlja deo glavnog prozora u kome se prati tok kompajliranja. Ovde postoji padajući meni sa opcijama da se prati samo kompajliranje dizajna, da se radi rana vremenska analiza sa sintezom, da se prati pun dizajn kod koga se pored celokupnog toka kompajliranja dodaju i dodatne opcije poput kreiranja projekta, kreiranja novog fajla i sl. koje su u principu dostupne i iz glavnog menija. Najčešće se koristi opcija koja je po difoltu postavljena (**Compilation**), a koja omogućava praćenje samo celokupnog toka kompajliranja. Postavljanjem ove opcije se jasno mogu videti etape u procesu kompajliranja koje odgovaraju onima navedenim u sekciji 3.1.

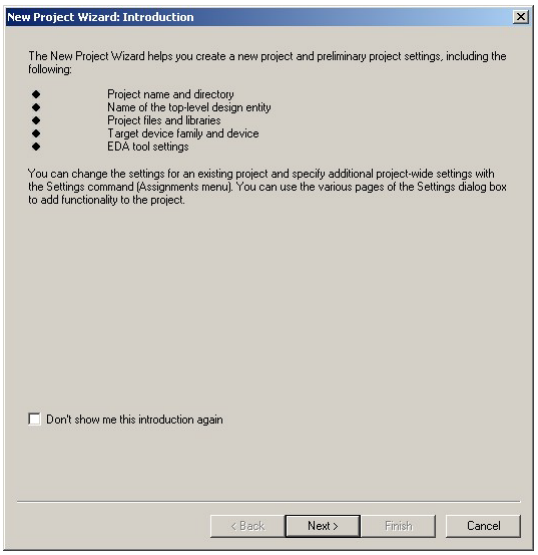
Centralni deo prozora se koristi za otvaranje tekućih fajlova u kome se može vršiti editovanje fajlova, čitanje izveštaja, vizuelni prikaz raspoređivanja dizajna na čip i sl. Na primer,

u ovom delu se otvaraju VHDL fajlovi, tj. preciznije otvara se editor koji omogućava editovanje VHDL fajlova.

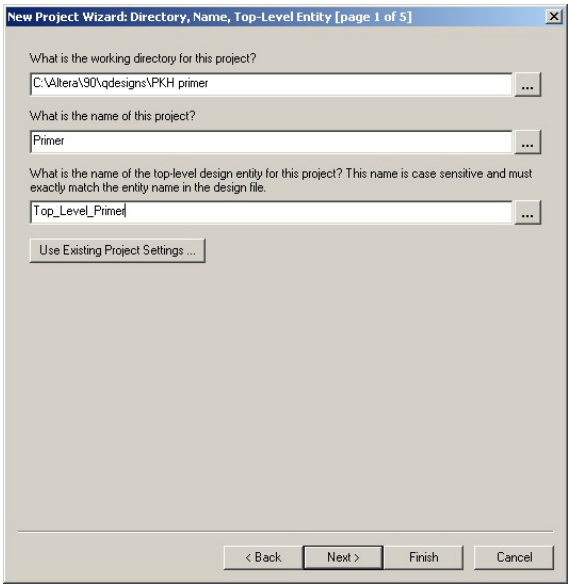
Deo za ispis obaveštenja se koristi za ispis raznih obaveštenja, prevashodno onih koji se odnose na tok kompajliranja. Tu se prikazuju obaveštenja o greškama, o toku aktiviranog kompajliranja, upozorenja i sl.

3.2.2. Kreiranje projekta

Izaberimo opciju **File->New Project Wizard**. Pokretanjem ove opcije otvaramo dijalog za kreiranje novog projekta. Napomena: na slikama su prikazani podaci koje treba unositi prilikom kreiranja projekta primera koji se koristi u ovoj sekciji za opis razvojnog okruženja. Prvi prozor dijaloga je prikazan na slici 3.2.2.1. Ovaj prozor samo daje osnovne informacije šta se podešava u projektu. Postoji opcija kojom se može ovaj prozor isključiti za buduća kreiranja projekta.

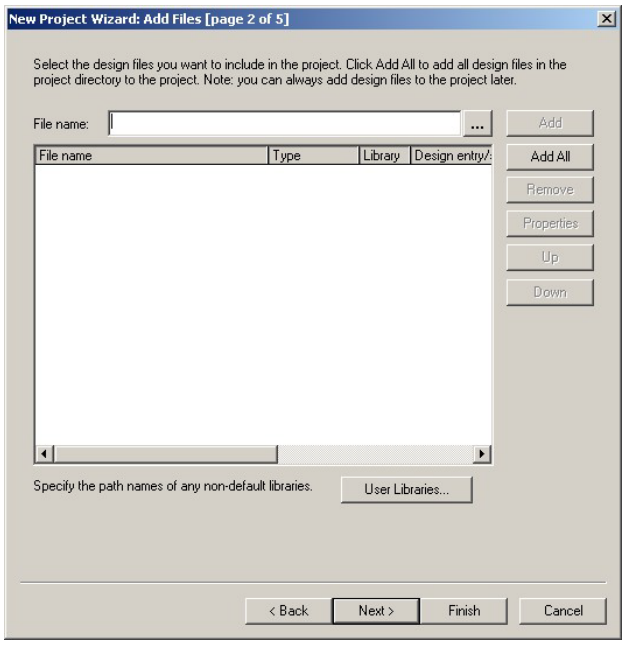


Slika 3.2.2.1. – Početni prozor dijaloga za kreiranje novog projekta



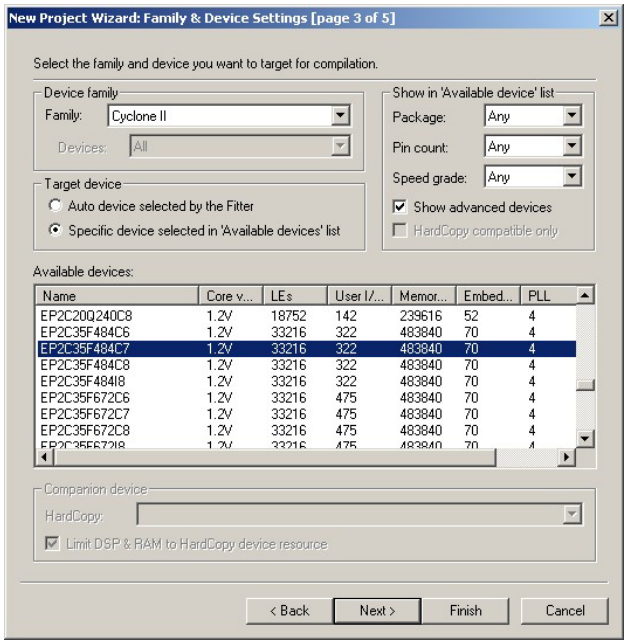
Slika 3.2.2.2. – Izbor lokacije projekta i naziva projekta i top level entiteta

Sledeći prozor (slika 3.2.2.2) daje mogućnost izbora lokacije projekta, kao i naziva projekta i entiteta koji će se naći na vrhu hijerarhije projekta (tzv. top level entitet). I nakon kreiranja projekta postoji mogućnost promene top level entiteta. Takođe, projekat se relativno adresira pa je moguće folder celokupnog projekta pomerati po želji. Takođe treba primetiti da se uvek možemo vratiti unazad tokom kreiranja projekta na neki od prethodnih prozora.



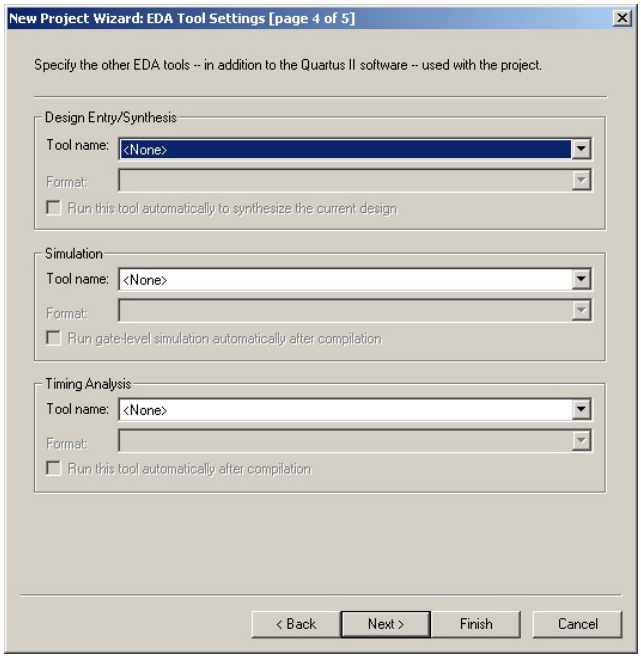
Slika 3.2.2.3. – Dodavanje fajlova u projekat

Sledeći prozor (slika 3.2.2.3) daje mogućnost dodavanja fajlova u projekat, ako već postoje neki fajlovi koji će biti deo projekta. Naravno, uvek je moguće dodati nove fajlove i kasnije nakon kreiranja projekta. Takođe se ovde mogu dodati i nestandardne biblioteke (sopstvene ili od drugih dizajnera).



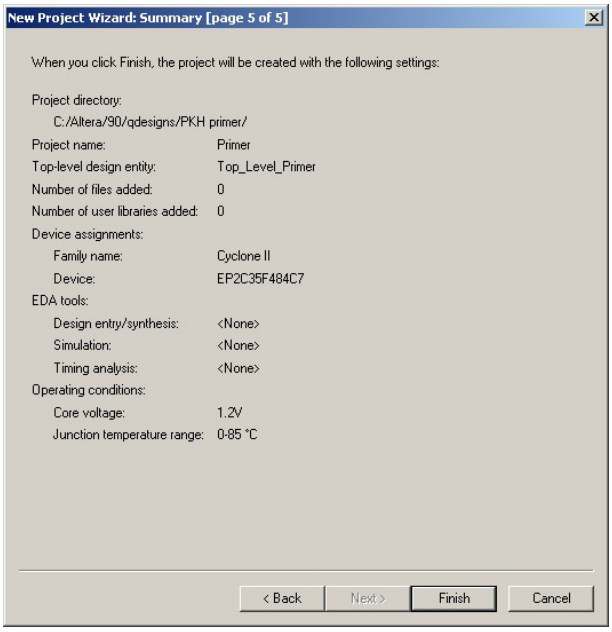
Slika 3.2.2.4. – Izbor FPGA čipa

Na sledećem prozoru (slika 3.2.2.4) se vrši izbor FPGA čipa. Naknadno, posle kreiranja projekta, je moguće izvršiti izbor nekog drugog čipa ukoliko se odlučimo da spustimo dizajn na neki drugi čip. Čip se bira prvo izborom familije u gornjem levom uglu, nakon čega se u donjoj polovini prozora dobija spisak čipova izabrane familije. Moguće je smanjiti broj prikazanih čipova, tako što se u desnom gornjem uglu odaberu podaci o konkretnom pakovanju čipa, broju pinova čipa i brzinskoj klasi čipa.



Slika 3.2.2.5. – Izbor alata drugih proizvođača

Na sledećem prozoru (slika 3.2.2.5) se biraju eventualno dodatni alati drugih proizvođača, ako su oni na raspolaganju. Ovi alati treba da omoguće bolje perfomanse u sintezi, fleksibilniju i kvalitetniju simulaciju i kvalitetniju vremensku analizu.

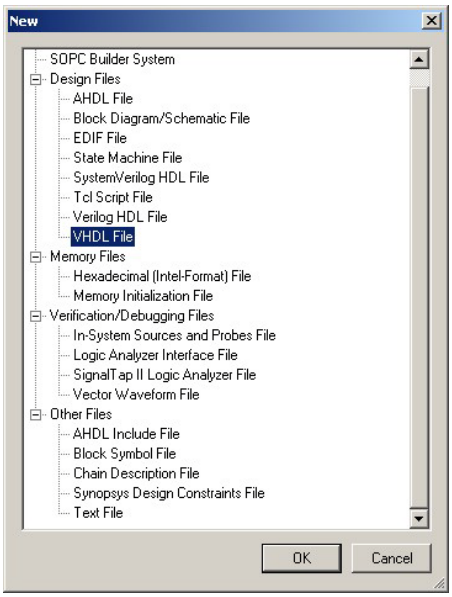


Slika 3.2.2.6. – Poslednji ekran dijaloga

Poslednji ekran dijaloga (slika 3.2.2.6) daje rezime svih prethodno odabranih podešavanja tokom dijaloga. Klikom na **Finish** dugme se izvršava kreiranje projekta sa podešavanjima zadatim tokom dijaloga. Projektni fajl ima ekstenziju .qpf, i dvostrukim klikom miša na ovaj fajl će se otvoriti dotični projekat u Quartus razvojnom okruženju. Naravno, otvaranjem ovog fajla iz Quartus razvojnog okruženja će se takođe otvoriti dotični projekat.

3.2.3. Unos dizajna

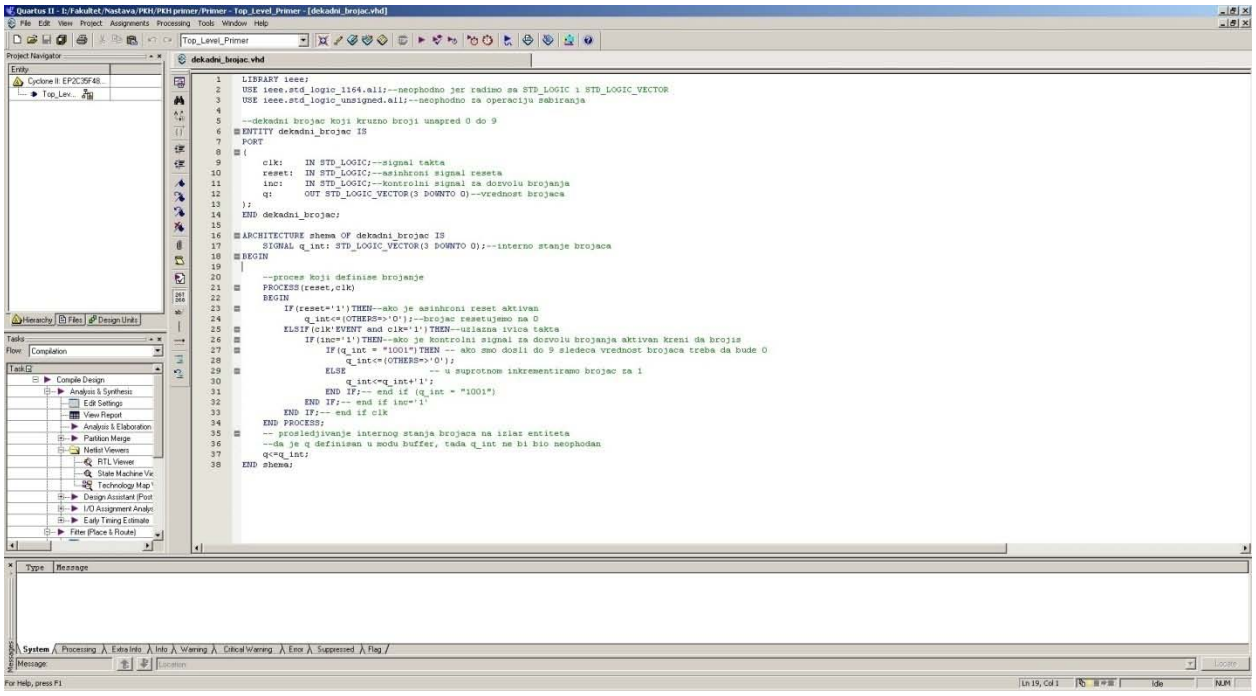
Nakon kreiranja projekta je potrebno kreirati i sam dizajn, na primer kreirati jedan ili više VHDL fajlova koji opisuju dizajn koji realizujemo tj. implementiramo. Izborom opcije **File-> New** se otvara novi prozor u kom se nudi izbor tipa fajla koji želimo da kreiramo (slika 3.2.3.1).



Slika 3.2.3.1. – Izbor tipa novog fajla

Kao što se i sa slike 3.2.3.1 vidi, postoji više tipova fajlova preko kojih se može uneti dizajn poput šematskog tj. grafičkog unosa dizajna (**Schematic File**), unosa dizajna pomoću Verilog ili VHDL jezika i dr. Ovde se takođe kreiraju fajlovi za inicijalizaciju sadržaja interne memorije ako se ona koristi u dizajnu, kao i za kreiranje dizajna koji se koristi u simulaciji i verifikaciji dizajna. Pošto se u okviru kursa predmeta uči VHDL jezik, u okviru ovih skripti će biti razmatran unos dizajna pomoću VHDL jezika. Očigledno, tada ćemo da izaberemo opciju **VHDL File**, kojom će se u centralnom delu glavnog prozora otvoriti editor u okviru koga unosimo sadržaj VHDL dizajna (slika 3.2.3.2). Nakon unosa dizajna neophodno je snimiti fajl, pri čemu se kreirani fajl po difoltu dodaje u projekat, sem ako se ne isključi čekirana opcija **Add file to current project** u dnu prozora dijaloga za snimanje fajla u okviru koga se zadaje i naziv fajla i lokacija gde će se snimiti fajl. Na samim vežbama u kursu će ukratko biti prikazan i šematski (grafički) unos dizajna, radi šireg prikaza mogućnosti razvojnog okruženja. Međutim, s obzirom da šematski unos dizajna nije zgodan u slučaju složenijih implementacija jer zahteva veliko vreme kreiranja takvog dizajna, a takođe nije pogodan ni za efikasno ažuriranje, ne preporučuje se upotreba ovog načina unosa dizajna. Pored standardnih opcija koje se često koriste prilikom editovanja teksta poput *copy*, *paste*, *cut*, *find* i *replace* opcija, na raspolaganju je i velik broj drugih opcija među kojima su i sledeće korisne opcije: ubacivanje VHDL templejta,

komentarisanje/otkomentarisanje selektovanih linija, provera ispravnosti sintakse unetog koda, prikaz rednih brojeva linija koda. Po difoltu je uključen prikaz rednih brojeva linija koda.

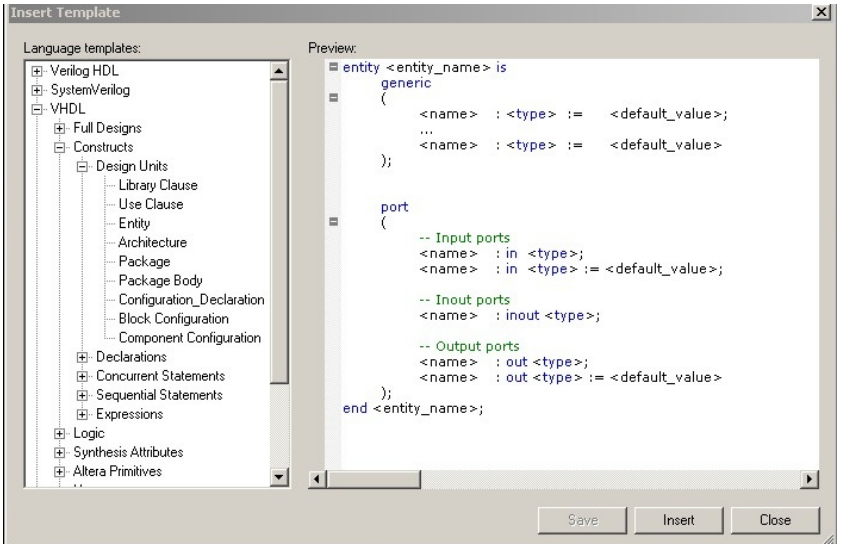


Slika 3.2.3.2. – Editor u centralnom delu glavnog prozora

Slika 3.2.3.3. – Korisne opcije editora

Navedene korisne opcije su dostupne u koloni ikonica na levoj strani editora. Kolona ikonica sa obeleženim korisnim opcijama je prikazana na slici 3.2.3.3. Većina ovih opcija je dostupna i u okviru glavnog menija. Ubacivanje templejta je dostupno u vidu opcije **Edit-> Insert Template**, analiza dizajna (provera ispravnosti sintakse unetog dizajna) je dostupna u vidu opcije **Processing->Analyze Current File**, aktivacija/deaktivacija prikaza rednih brojeva linija koda je dostupna u vidu opcije **View->Show Line Numbers**. Takođe, desnim klikom miša unutar editora se dobija meni u okviru koga se nalaze i opcije za komentarisanje i

otkomentarisanje selektovanih linija koda, kao i za ubacivanje templejta. Ubacivanje templejta omogućava ubacivanje kostura različitih konstrukcija i deklaracija, kako VHDL koda, tako i drugih jezika poput Veriloga. Aktiviranjem opcije ubacivanja templejta se otvara prozor za izbor templejta koji želimo ubaciti, pri čemu su templejti organizovani u logične celine. Prozor za izbor templejta je prikazan na slici 3.2.3.4 i na njemu je prikazan izbor templejta za deklaraciju entiteta u VHDL jeziku.



Slika 3.2.3.4. – Prozor za izbor templejta

Kreirajmo tri VHDL fajla koji će predstavljati deo projekta. Kod fajla **dekadni_brojac.vhd** je:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;--neophodno jer radimo sa STD_LOGIC i STD_LOGIC_VECTOR
USE ieee.std_logic_unsigned.all;--neophodno za operaciju sabiranja

--dekadni brojac koji kruzno broji unapred 0 do 9
ENTITY dekadni_brojac IS
PORT
(
	clk:	IN STD_LOGIC;--signal takta
	reset:	IN STD_LOGIC;--asinhroni signal reseta
	inc:	IN STD_LOGIC;--kontrolni signal za dozvolu brojanja
	q:	OUT STD_LOGIC_VECTOR(3 DOWNTO 0)--vrednost brojaca
);
END dekadni_brojac;

ARCHITECTURE shema OF dekadni_brojac IS
	SIGNAL q_int: STD_LOGIC_VECTOR(3 DOWNTO 0);--interno stanje brojaca
BEGIN
	--proces koji definise brojanje
	PROCESS(reset,clk)
	BEGIN
		IF(reset='1')THEN--ako je asinhroni reset aktivan
			q_int<=(OTHERS=>'0');--brojac resetujemo na 0
		ELSIF(clk'EVENT and clk='1')THEN--uzlazna ivica takta
			IF(inc='1')THEN--ako je kontrolni signal za dozvolu brojanja aktivan kreni da brojis
				IF(q_int = "1001")THEN -- ako smo dosli do 9 sledeca vrednost brojaca treba da bude 0
					q_int<=(OTHERS=>'0');
				ELSE                                    -- u suprotnom inkrementiramo brojac za 1
```

```
                                        q_int<=q_int+'1';
                                    END IF;-- end if (q_int = "1001")
                                END IF;-- end if inc='1'
                        END IF;-- end if clk
                END PROCESS;
                -- prosledjivanje internog stanja brojaca na izlaz entiteta
                --da je q definisan u modu buffer, tada q_int ne bi bio neophodan
                q<=q_int;
END shema;
```

Kod fajla **displej.vhd** je:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;--neophodno jer radimo sa STD_LOGIC i STD_LOGIC_VECTOR

--kolo koje broj sa ulaza pretvara u signale koji pale ili gase diode displeja
ENTITY displej IS
PORT
(
        broj:   IN STD_LOGIC_VECTOR(3 DOWNTO 0);--vrednost broja na ulazu
        displej:    OUT STD_LOGIC_VECTOR(6 DOWNTO 0)--vektor ciji svaki indeks utice na jednu diodu displeja
);
END displej;

ARCHITECTURE shema OF displej IS
BEGIN
        --proces koji definise prikaz displeja
        --kombinaciona logika kreirana sekvencijalnim kodom
                --                      relacije diode na displeju i indeksa vektora displej
                --                      - 0
                --                    5| |1
                --                      - 6
                --                    4| |2
                --                      - 3
        -- logicka nula pali diodu, logicka jedinica gasi diodu
        PROCESS(broj)
        BEGIN
                CASE broj IS
                        WHEN "0000" =>
                                displej<="1000000";--cifra 0
                        WHEN "0001" =>
                                displej<="1111001";--cifra 1
                        WHEN "0010" =>
                                displej<="0100100";--cifra 2
                        WHEN "0011" =>
                                displej<="0110000";--cifra 3
                        WHEN "0100" =>
                                displej<="0011001";--cifra 4
                        WHEN "0101" =>
                                displej<="0010010";--cifra 5
                        WHEN "0110" =>
                                displej<="0000010";--cifra 6
                        WHEN "0111" =>
                                displej<="1111000";--cifra 7
                        WHEN "1000" =>
                                displej<="0000000";--cifra 8
                        WHEN "1001" =>
                                displej<="0010000";--cifra 9
                        WHEN OTHERS =>--ovo pokriva brojeve koji ne bi smeli da se pojave na ulazu
```

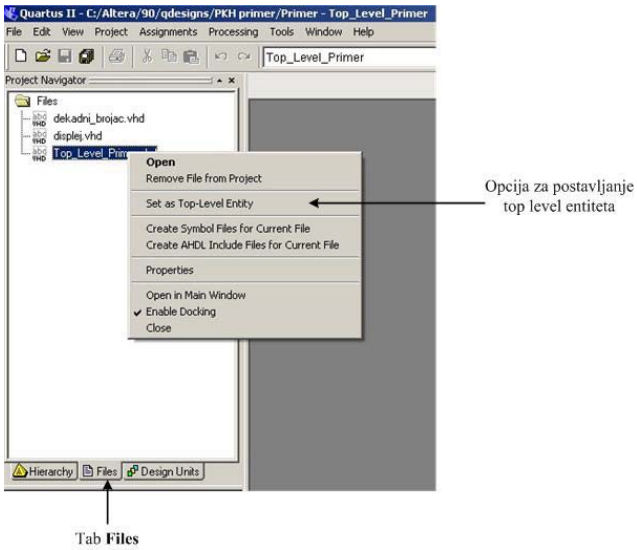
```
                                        displej<="0001001";--cifra H kojom signaliziramo da pojavu broja koji nije u opsegu 0-9
                END CASE;
        END PROCESS;
END shema;
```

Kod fajla **Top_Level_Primer.vhd** je:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;--neophodno jer radimo sa STD_LOGIC i STD_LOGIC_VECTOR

--top level entitet koji uvozi u vidu komponenti dekadni brojac i displej logiku
ENTITY top_level_primer IS
PORT
(
        clk:            IN STD_LOGIC;--signal takta
        reset:          IN STD_LOGIC;--asinhroni signal reseta
        inc:            IN STD_LOGIC;--kontrolni signal za dozvolu brojanja
        displej_out:    OUT STD_LOGIC_VECTOR(6 DOWNTO 0)--izlaz koji kontrolise displej
);
END top_level_primer;

ARCHITECTURE shema OF top_level_primer IS
        --deklaracija komponente dekadnog brojaca
        COMPONENT dekadni_brojac IS
        PORT
        (
                clk:    IN STD_LOGIC;--signal takta
                reset:  IN STD_LOGIC;--asinhroni signal reseta
                inc:    IN STD_LOGIC;--kontrolni signal za dozvolu brojanja
                q:      OUT STD_LOGIC_VECTOR(3 DOWNTO 0)--vrednost brojaca
        );
        END COMPONENT;
        --deklaracije logike za kontrolu displeja
        COMPONENT displej IS
        PORT
        (
                broj:   IN STD_LOGIC_VECTOR(3 DOWNTO 0);--vrednost broja na ulazu
                displej:OUT STD_LOGIC_VECTOR(6 DOWNTO 0)--vektor ciji svaki indeks utice na jednu diodu displeja
        );
        END COMPONENT;
        --deklaracija internog signala neophodnog za povezivanje izlaza dekadnog brojaca
        -- sa ulazom u displej logiku
        SIGNAL broj_int:STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
        --instanciranje komponente brojaca
        brojac_inst: dekadni_brojac
        PORT MAP
        (
                clk => clk,
                reset => reset,
                inc => inc,
                q => broj_int
        );
        --instanciranje komponente za kontrolu displeja
        displej_inst: displej
        PORT MAP
        (
                broj => broj_int,
                displej => displej_out
```

```
    );
END shema;
```

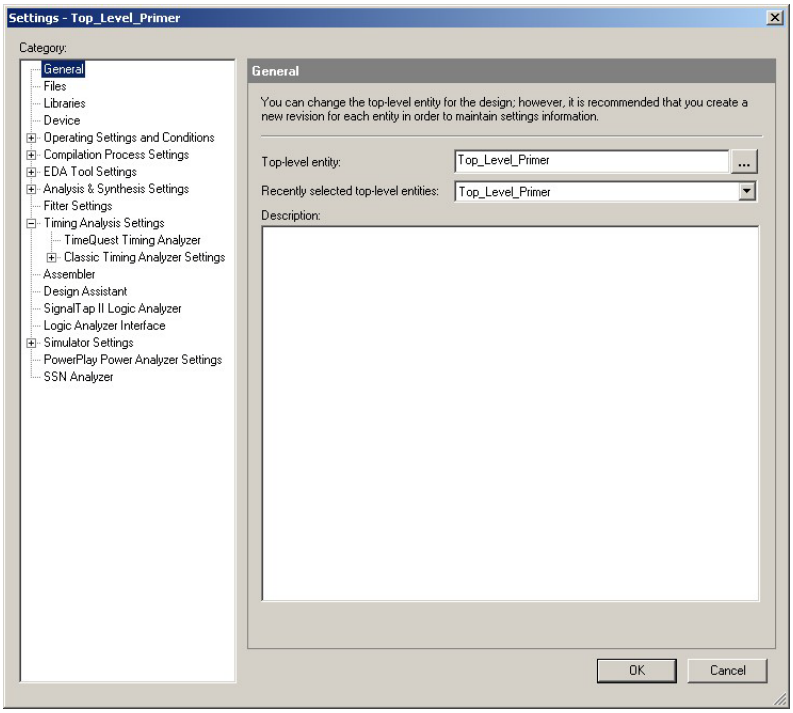
Prvi vhdl fajl opisuje jedan dekadni brojač koji broji kružno u opsegu 0 do 9. Drugi vhdl fajl kreira logiku koji dekadni broj sa ulaza pretvara u izlazni signal koji pali i gasi diode na displeju radi prikaza broja sa ulaza. Treći vhdl fajl je top level entitet koji uvozi entitete iz prva dva fajla u vidu komponenti i povezuje ih kako međusobno, tako i sa pinovima FPGA čipa. Konačan dizajn sadrži dekadni brojač koji broji unapred i čije se trenutno stanje pomoću displej logike vodi na jedan led displej radi vizuelnog prikaza. Svaki od kreiranih fajlova će po snimanju biti po difoltu uključen u projekat, sem u slučaju kada se isključi opcija **Add file to current project** u dnu prozora dijaloga za snimanje fajla. Ako su navedeni fajlovi već pripremljeni, mogu se dodati u projekat pomoću aktivacije opcije za dodavanje novih fajlova u projekat koja se nalazi u **Project ->Add/Remove Files in Project** čime će se otvoriti prozor veoma sličan onome prikazanom na slici 3.2.2.3. Nakon što smo dodali fajlove u projekat, treba definisati top level entitet. To se radi tako što se u delu koji prikazuje strukturu projekta otvori tab **Files**. Desnim klikom miša na vhdl fajl koji sadrži top level entitet se otvara meni u kome treba da izaberemo opciju **Set as Top-Level Entity**, čime postavljamo dotični fajl, odnosno entitet, koji je u njemu definisan, kao top level entitet projekta (slika 3.2.3.5). Po difoltu top level entitet projekta je onaj fajl koji sadrži entitet identičnog naziva onome koji je postavljen tokom kreiranja projekta. Top level entitet se uvek može promeniti primenom opcije **Set as Top-Level Entity**. Treba primetiti da je u meniju, koji se otvara na desni klik miša, na raspolaganju i opcija za uklanjanje fajla iz projekta.



Slika 3.2.3.5. – Izbor top level entiteta

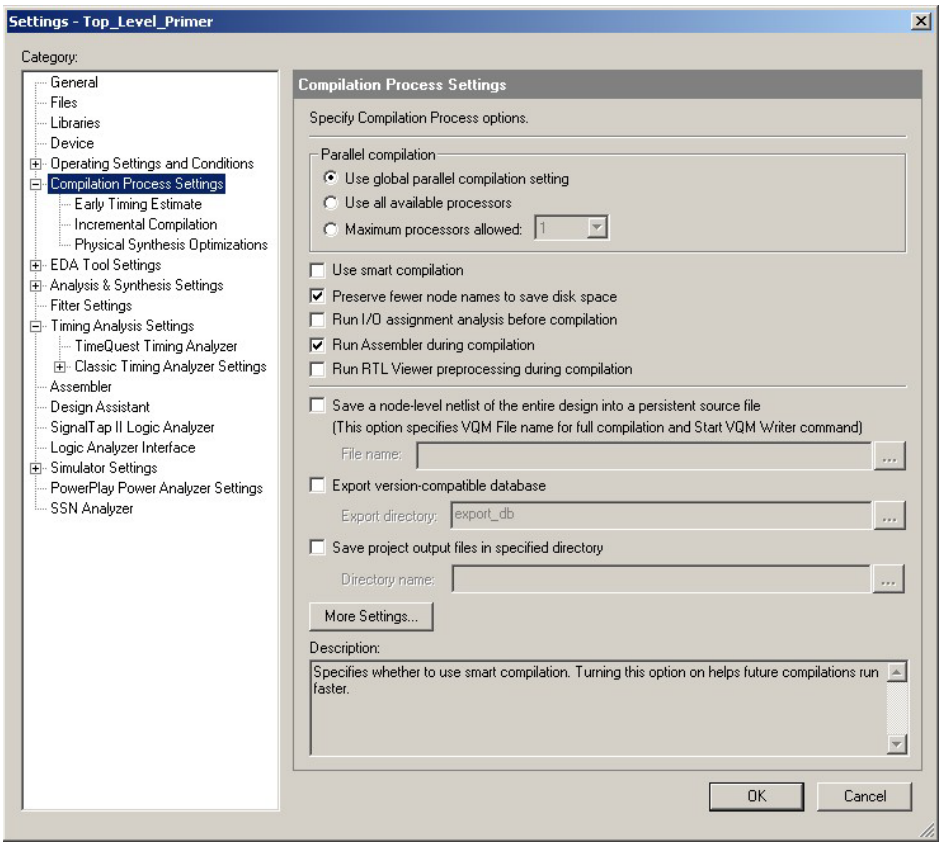
3.2.4. Podešavanje opcija projekta i kompajlera

Nakon što smo uneli dizajn i definisali top level entitet, neophodno je pokrenuti kompajliranje. Međutim, u prvom momentu treba pokrenuti samo prvi korak kompajliranja, a to je analiza i sinteza. Klikom na opciju **Assignments->Settings** otvara se prozor koji omogućava podešavanje projekta i parametara kompajliranja (slika 3.2.4.1).



Slika 3.2.4.1. – Prozor za podešavanje projekta

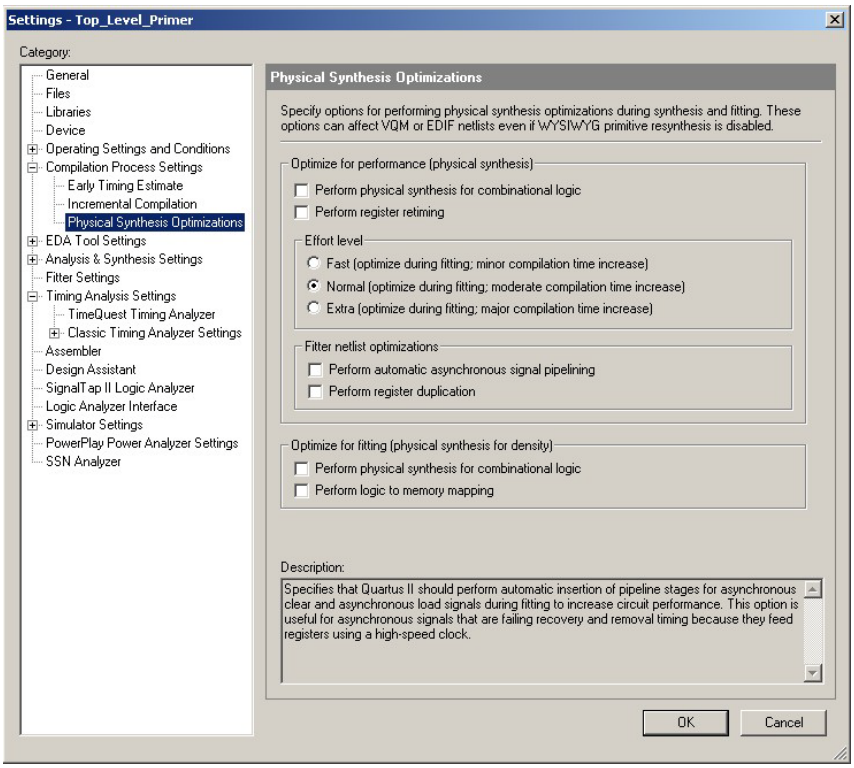
General meni daje podatak o top level entitetu, kao i mogućnost promene top level entiteta. **Files** meni otvara identičan prozor onome koji se dobija izborom opcije **Project-> Add/Remove Files in Project** iz glavnog menija. Očigledno ova opcija služi za dodavanje i uklanjanje fajlova iz projekta. **Libraries** meni omogućava uključivanje dodatnih biblioteka u projekat. **Device** meni služi za promenu FPGA čipa (prozor koji se otvara je gotovo identičan onome sa slike 3.2.2.4 iz dijaloga prilikom kreiranja projekta). **Operating Settings and Conditions** meni omogućava izbor radnih uslova (ovu opciju treba ostaviti na difoltu sem ako nisu unapred poznati radni uslovi). **Compilation Process Settings** meni daje mogućnost podešavanja parametara kompajliranja (slika 3.2.4.2). U okviru ovih podešavanja treba uključiti opciju **Smart Compilation** (nije po difoltu uključena) koja može da ubrza proces kompajliranja. Ova opcija uključuje pametno kompajliranje koje pokušava da detektuje šta je promenjeno u dizajnu od poslednjeg kompajliranja, pa da se u procesu kompajliranja što više koriste već gotovi rezultati prethodnog kompajliranja, i da se procesiraju samo izmenjeni delovi dizajna. Takođe treba primetiti da u dnu prozora postoji **Description** polje za ispis opisa selektovane opcije čime je olakšano razumevanje svake od ponuđenih opcija. Unutar **Compilation Process Settings** postoje još tri podmenija na levoj strani prozora **Early Timing Estimate**, **Incremental Compilation** i **Physical Synthesis Optimizations**. Kod sva tri ova podmenija takođe postoji u dnu prozora **Description** polje sa korisnim opisom selektovane opcije u okviru ovih podmenija. U okviru **Incremental Compilation** je korisna opcija (istog naziva kao i podmeni) **Incremental Compilation** koja aktivira inkrementalno kompajliranje kojim se ubrzava proces kompajliranja. Ova opcija je uključena po difoltu.



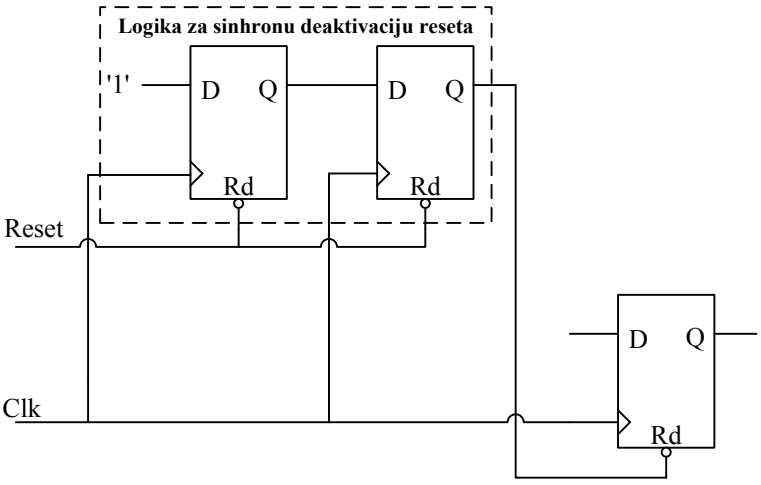
Slika 3.2.4.2. – Prozor za podešavanje parametara kompajliranja

Podmeni **Physical Synthesis Optimizations** sadrži korisna podešavanja koja se koriste prilikom koraka sinteze i postavljanja tokom kompajliranja (slika 3.2.4.3). Tako na primer **Effort Level** ima tri izbora **Fast**, **Normal** (difolt izbor) i **Extra**. Ova opcija definiše ponašanje optimizacionog procesa, **Extra** nivo značajno produžuje vreme kompajliranja, ali potencijalno nalazi najoptimalnije rešenje, a **Fast** nivo postiže najkraće vreme kompajliranja, ali nalazi najmanje kvalitetno rešenje. **Normal** nivo je sredina između ova dva ekstrema. Ovde treba imati na umu da u slučaju postavljanja složenog dizajna na čip dobijamo veoma složen problem kako rasporediti dizajn po internim resursima čipa i rešenja koja se nalaze tipično nisu optimalna, već suboptimalna. Izbor **Effort Level** nivoa definiše koliko će algoritam optimizacije dugo tražiti što kvalitetnije rešenje, odnosno kriterijum kada će odustati od dalje pretrage. Opcija **Perform physical synthesis for combinational logic** aktivira proces optimizacije kombinacione logike u cilju dobijanja boljih performansi. Opcija **Register retiming** aktivira proces optimizacije registara tako što se menjaju njihovi vremeski parametri poput vremena postavljanja i držanja, koja će biti objašnjena nešto kasnije, a u cilju ostvarivanja bolje performanse implementacije. Opcija **Perform automatic asynchronous signal pipelining** aktivira proces dodavanja dodatne logike za asinhroni reset i učitavanje (*load*) radi postizanja boljih performansi implementacije. Ova opcija je korisna za implementacije koje rade na veoma visokim frekvencijama (100MHz i više), jer u slučaju asinhronih signala može da dođe do nezgodne situacije prilikom deaktivacije asinhronog signala. Na primer, ako se asinhroni reset aktivira i drži dovoljno dugo svi flip-flopovi na koje je doveden asinhroni signal reseta će biti resetovani. Međutim, problem može da nastane ako se reset deaktivira neposredno pre dolaska aktivne ivice signala takta. Tada će neki flip-flopovi da reaguju na tu ivicu takta, a neki će da je preskoče i reagovaće tek na sledeće ivice

takta. Razlog za ovu situaciju su različita fizička kašnjenja asinhronog signala, ali i signala takta do flip-flopova. Ovaj problem može da se prevaziđe logikom koja je prikazana na slici 3.2.4.4.



Slika 3.2.4.3. – Prozor za podešavanje parametara kompajliranja u fazi sinteze i postavljanja



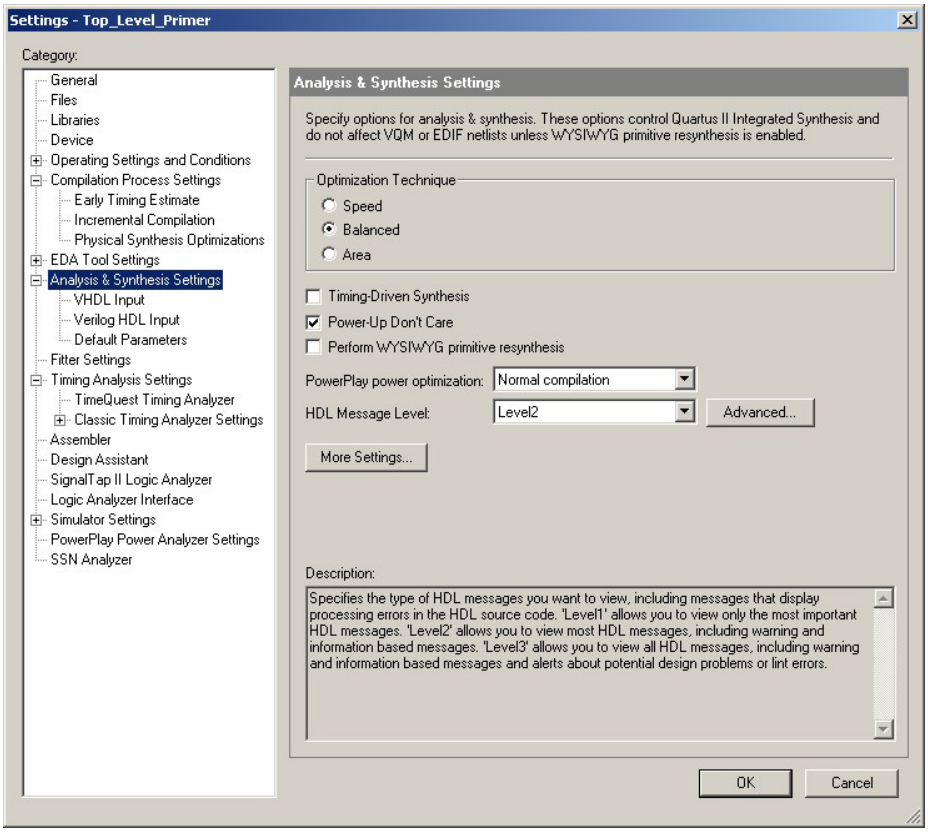
Slika 3.2.4.4. – Logika za sinhronu deaktivaciju asinhronog reseta

Kolo sa slike 3.2.4.4 omogućava asinhroni reset, ali sinhroni izlazak iz reseta, čime se prevazilazi navedeni problem asinhronog reseta. Kolo radi tako što se, po dolasku aktivne vrednosti reseta (u primeru sa slike to je '0'), resetuju asinhrono dva gornja flip-flopa sa slike. Čim se pojavi vrednost '0' na izlazu gornjeg desnog flip-flopa, izvršiće se asinhroni reset donjeg flip-flopa. Kada se reset deaktivira, gornji levi flip-flop će na prvu aktivnu ivicu takta koju detektuje izvršiti promenu svog izlaza na vrednost '1', a potom će na sledeću ivicu takta i gornji desni flip-flop da promeni vrednost svog izlaza na vrednost '1' čime će se deaktivirati asinhroni

reset donjeg flip-flopa. Na ovaj način, u stvari, svi flip-flopovi, koje bi kontrolisali dva flip-flopa iz logike za sinhronu deeaktivaciju reseta, imali bi neposredno posle aktivne ivice takta deaktiviran asinhroni reset, i time bi svi počeli da reaguju na prvu sledeću ivicu takta, odnosno na istu ivicu takta što je i bio cilj.

Još dve korisne opcije postoje u podmeniju **Physical Synthesis Optimizations**. Opcija **Perform register duplication** omogućava dupliranje registara u cilju dobijanja boljih performansi, najčešće u vidu smanjenog maksimalnog kašnjenja u dizajnu. Opcija **Perform logic to memory mapping** omogućava da se deo logike izmesti u neiskorišćenu internu memoriju u cilju dobijanja boljih performansi ili u slučaju da nema dovoljnog broja CLB blokova u čipu za smeštanje dizajna.

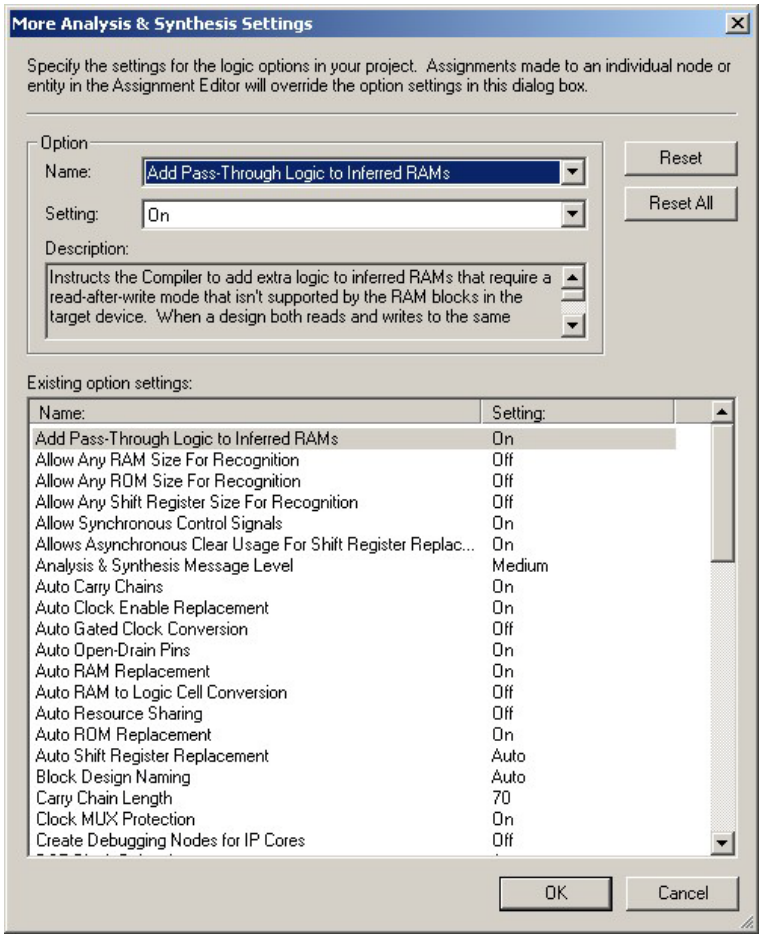
EDA Tool Settings meni omogućava podešavanje alata drugih kompanija ukoliko su oni na raspolaganju.



Slika 3.2.4.5. – Prozor za podešavanje parametara procesa analize i sinteze

Analysis and Synthesis Settings meni omogućava podešavanje procesa analize i sinteze u kompajlera (slika 3.2.4.5). I ovde u svim prozorima postoji **Description** polje za opis selektovane opcije. **Optimization Technique** daje mogućnost podešavanja kriterijuma optimizacije dizajna u vidu **Speed**, **Balanced** (difolt vrednost) i **Area** izbora. Optimizacija dizajna će pokušati da postigne maksimalnu frekvenciju dizajna u slučaju izbora **Speed** opcije, a u slučaju izbora **Area** opcije će pokušati da postigne minimalno zauzeće resursa. Opcija **Balanced** predstavlja sredinu između navedena dva ekstrema. **Power-Up Don't Care** opcija definiše da se svim registrima, kojima nisu definisane inicijalne vrednosti, postave one inicijalne vrednosti koje omogućavaju najbolje performanse dizajna tj. najmanje zauzimanje internih

resursa FPGA čipa. Klikom na **More Settings** dugme, otvara se nov prozor u kome se nalaze dodatna podešavanja parametara procesa analize i sinteze (slika 3.2.4.6). Za svaku od opcija piše njen opis, odnosno njena uloga. U zavisnosti od tipa i željenih performansi dizajna treba aktivirati one opcije za koje se na osnovu njihovog opisa utvrdi da mogu dovesti do boljih rezultata kompajliranja. Treba, međutim, napomenuti da je u slučaju veoma složenog dizajna, koji troši velik broj internih resursa čipa, optimizacija takvog dizajna veoma složena. Ponekad se dešava da iako izaberemo opciju (ili više opcija) koja bi trebala da doprinese boljim performansama implementacije (ili bar da ostanu iste kao i pre uključivanja opcije), u stvari, dobijemo lošije performanse, što ukazuje na složenost kriterijuma optimizacije i uopšte optimizacionog postupka. Očigledno, u ovakvim situacijama algoritam optimizacije završi u nekom lošem lokalnom optimalnom rešenju.

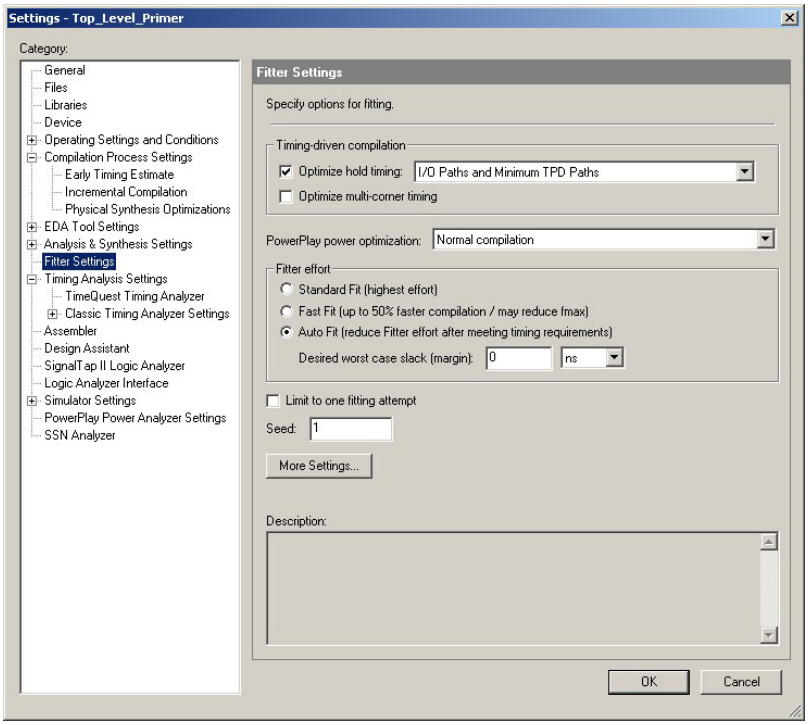


Slika 3.2.4.6. – Prozor za podešavanje dodatnih parametara procesa analize i sinteze

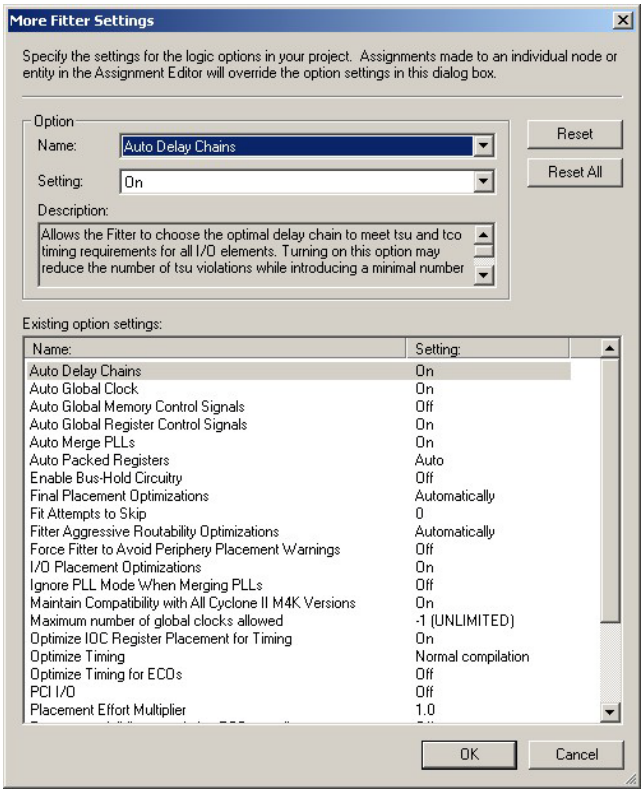
U okviru podešavanja parametara procesa analize i sinteze, postoje i podmeniji u kojima se podešavaju i parametri VHDL i Verilog jezika. Na primer, za VHDL jezik se podešava standard koji će se koristiti prilikom analize VHDL dizajna - VHDL-89 ili VHDL-93 (difolt izbor) standard.

Fitter meni definiše podešavanje parametara postavljanja i rutiranja dizajna na izabrani FPGA čip (slika 3.2.4.7). I ovde postoji **Description** polje za opis selektovane opcije. Interesantna opcija je **Fitter effort** koja definiše koliko će se algoritam optimizacije truditi tokom procesa postavljanja. **Auto Fit** izbor definiše da će nakon ispunjavanja željenih

vremenskih ograničenja, proces postavljanja skratiti tj. smanjiti broj pokušaja da se nađe još optimalnije rešenje, dok će **Standard Fit** izbor tražiti kvalitetnije rešenje sve dok ne bude siguran da više ne može da nađe bolje rešenje (ovaj izbor povećava vreme kompajliranja).

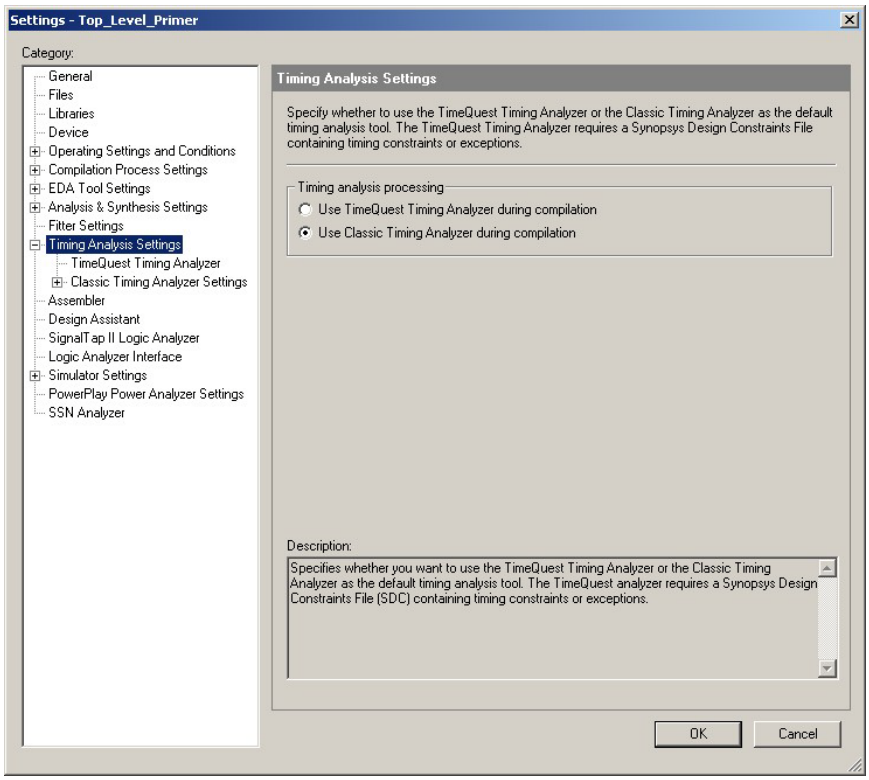


Slika 3.2.4.7. – Prozor za podešavanje parametara procesa postavljanja i rutiranja



Slika 3.2.4.8. – Prozor za podešavanje dodatnih parametara procesa postavljanja i rutiranja

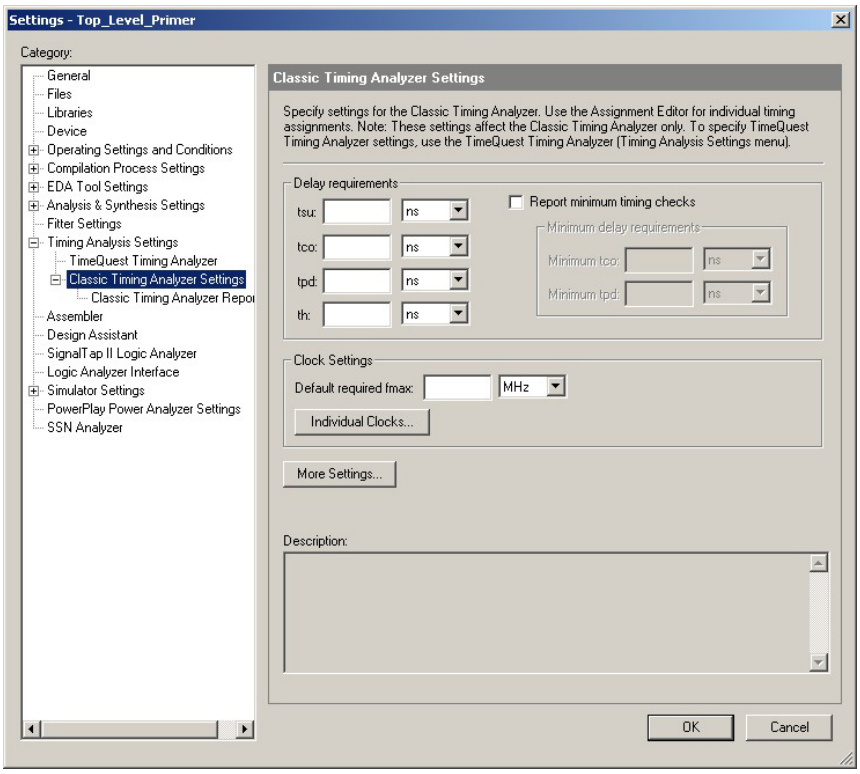
Klikom na **More Settings** dugme se otvara nov prozor u kome se nalaze dodatna podešavanja parametara procesa postavljanja i rutiranja (slika 3.2.4.8). Za svaku od opcija piše njen opis, odnosno njena uloga. U zavisnosti od tipa i željenih performansi dizajna treba aktivirati one opcije za koje se na osnovu njihovog opisa utvrdi da mogu dovesti do boljih rezultata kompajliranja.



Slika 3.2.4.9. – Prozor za podešavanje parametara procesa vremenske analize

Timing Analysis Settings meni prikazan na slici 3.2.4.9 definiše podešavanje parametara vremenske analize postavljenog dizajna (dizajna kojeg je proces postavljanja i rutiranja prostorno rasporedio na čip). I ovde u svim prozorima postoji **Description** polje za opis selektovane opcije. Postoje dve opcije vremenske analize **TimeQuest** i **Classic**. Svakoj od ove dve navedene opcije je dodeljen jedan podmeni kao što se može videti sa slike 3.2.4.9. **TimeQuest** opcija zahteva definisanje *Synopsis Design Constraints* fajla sa ekstenzijom .sdc i ova opcija neće biti obrađena u okviru ovih skripti. Podmeni **Classic** opcije je prikazan na slici 3.2.4.10. U okviru ovog podmenija se postavljaju (ukoliko to želimo) vremenska ograničenja za registre tj. flip-flopove koja važe u čitavom projektu. Vreme postavljanja t_{su} (*setup time*) predstavlja vreme za koje signal na ulazu flip-flopa mora biti postavljen pre dolaska aktivne ivice signala takta da bi sa sigurnošću bio prihvaćen na dotičnu ivicu takta. Vreme kašnjenja sa ulaza na izlaz t_{co} (*clock to output time*) predstavlja vreme za koje će signal sa ulaza flip-flopa da se pojavi na izlaznom pinu (direktno ili posle kombinacione logike) nakon aktivne ivice takta. Vreme kašnjenja kombinacione logike t_{pd} (*pin to pin delay*) predstavlja vreme za koje će signal sa ulaza kombinacione logike da se pojavi na izlazu kombinacione logike (podrazumeva se situacija kada promena jednog ulaza kombinacione logike utiče na promenu bar jednog od izlaza kombinacione logike). Vreme držanja t_h (*hold time*) predstavlja vreme za koje signal na ulazu flip-flopa mora imati stabilnu vrednost nakon pojave aktivne vrednosti ivice takta da bi flip-flop ispravno obavio svoju funkciju. Ukoliko želimo da postavimo posebna vremenska ograničenja

samo za pojedine signale tj. registre, tada se mora koristiti **Assignments Editor** dostupan preko glavnog menija kao **Assignments->Assignments Editor** opcija.

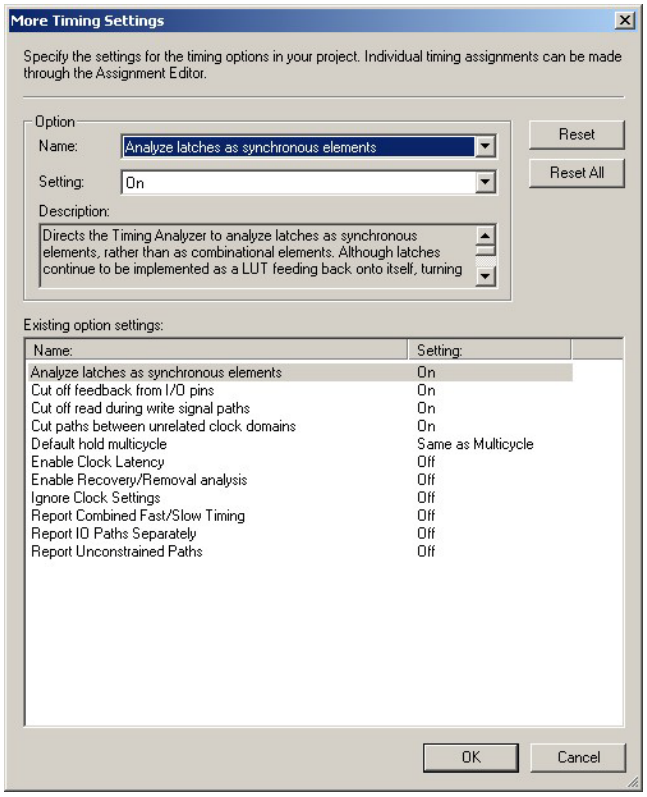


Slika 3.2.4.10. – Prozor podmenija Classic opcije

Podešavanja parametara takta su veoma značajna. U polju **Default required fmax** se podešava željena maksimalna frekvencija na nivou čitavog dizajna, a klikom na dugme **Individual Clocks** se otvara prozor u kome se definišu taktovi u dizajnu i njihova željena frekvencija. Podešavanje individualnih taktova je veoma značajno jer se njime definiše da se signali, koje izaberemo u ovom podešavanju, tretiraju kao globalni taktovi čime će u FPGA čipu biti rutirani brzim takt linijama (čiji je broj ograničen). Individualni takt se definiše nakon završetka procesa analize i sinteze ukoliko želimo da koristimo asistenciju razvojnog okruženja prilikom unosa naziva signala ili može da se definiše i pre ukoliko želimo sami da unosimo naziv signala koji koristimo kao takt u dizajnu. Primer definisanja individualnih taktova će biti prikazan nešto kasnije u skriptama.

Klikom na **More Settings** dugme se otvara nov prozor u kome se nalaze dodatna podešavanja parametara procesa klasične vremenske analize (slika 3.2.4.11). Za svaku od opcija piše njen opis, odnosno njena uloga. U zavisnosti od tipa i željenih performansi dizajna treba aktivirati one opcije za koje se na osnovu njihovog opisa utvrdi da mogu dovesti do boljih rezultata kompajliranja.

Dodatni podmeni **Classic Timing Analyzer Reporting** definiše nivo detaljnosti izveštaja vremenske analize kada se koristi **Classic** opcija. Ovde se bira broj najsporijih, odnosno putanja sa najvećim kašnjenjem koji će biti prikazan u izveštaju, a takođe omogućava definisanje kriterijuma pomoću kojih se pojedine putanje u dizajnu isključuju iz izveštaja.



Slika 3.2.4.11. – Prozor za podešavanje dodatnih parametara procesa klasične vremenske analize

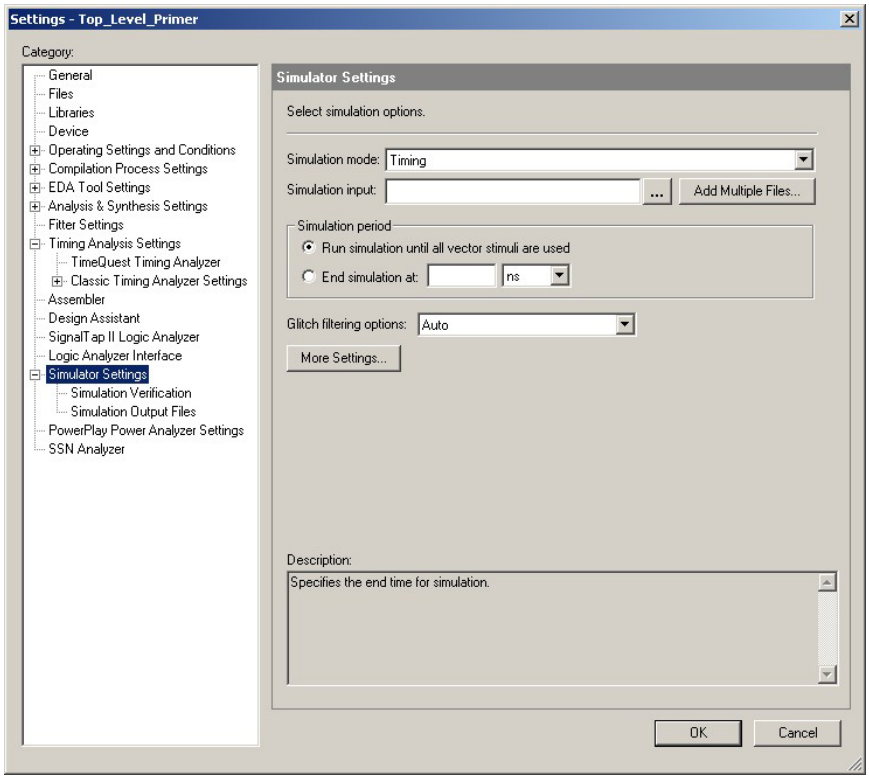
Assembler meni daje mogućnost podešavanja opcija asemblera. Asembler u okviru procesa kompajliranja predstavlja korak kompajliranja u kome se generiše fajl koji sadrži konfiguraciju FPGA čipa koja odgovara kompajliranom dizajnu.

Design Assistant meni daje pregled podešavanja opcija koje omogućavaju razvojnom okruženju da ukažu dizajneru na potencijalne greške u dizajniranju. Aktiviranjem opcije **Run Design Assistant during compilation** se omogućava da razvojno okruženje signalizira tokom procesa kompajliranja na potencijalne greške u dizajnu (u pitanju nisu sintaksne greške, već potencijalno loše projektovanje dizajna) iz skupa opcija koje su selektovane u **Design Assistant** meniju.

SignalTap II Logic Analyzer meni daje mogućnost formiranja i aktivacije internog logičkog analizatora u FPGA čipu uz implementirani dizajn za potrebe testiranja i verifikacije dizajna na samom čipu. **Logic Analyzer Interface** meni daje mogućnost aktivacije interfejsa ka eksternom logičkom analizatoru tako što se grupa internih signala dizajna povezuje na eksterni logički analizator preko manjeg broja pinova FPGA čipa. Oba menija zahtevaju da se uključi odgovarajući fajl koji definiše koji interni signali će se posmatrati u dizajnu. Primer upotrebe logičkih analizatora će biti dat kasnije.

Simulator Settings meni podešava opcije ugrađenog simulatora (slika 3.2.4.12). **Simulation mode** padajući meni daje izbor tipa simulacije: funkcionalna simulacija, vremenska simulacija, i vremenska simulacija koja koristi **Fast Time Modeling** radi ubrzavanja procesa simulacije. **Simulation input** omogućava uključivanje fajla koji sadrži stimuluse (simulirane vrednosti ulaza dizajna). Takođe je omogućen i izbor trajanja simulacije. Klikom na dugme **More Settings** se otvara prozor sa dodatnim podešavanjima ugrađenog simulatora. **Simulation**

Verification podmeni daje dodatna podešavanja koja između ostalog omogućuju detekciju gličeva i narušavanja vremenskih ograničenja u vremenskoj analizi, kao i podešavanje poređenja očekivanih i ostvarenih vrednosti posmatranih signala u simulaciji. **Simulation Output Files** podmeni daje podešavanje prikaza rezultujućeg fajla simulacije, a takođe omogućava generisanje fajlova koji se koriste u analizi potrošnje snage implementiranog dizajna. Sam proces simulacije će biti prikazan kasnije u skriptama.



Slika 3.2.4.12. – Prozor za podešavanje opcija ugrađenog simulatora

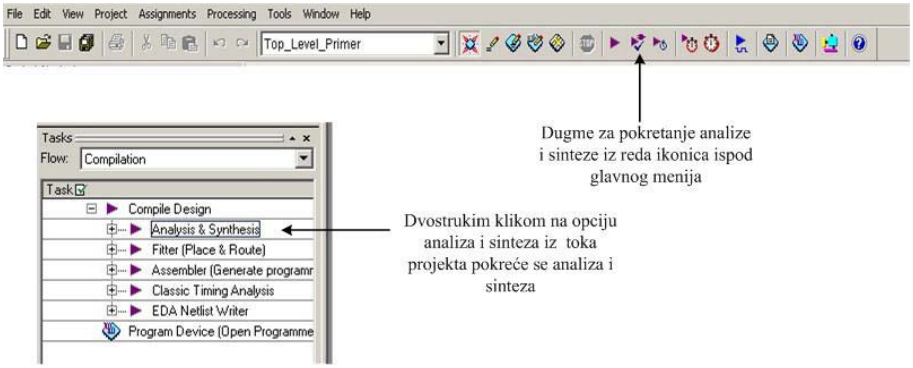
PowerPlay Power Analyzer Settings meni daje mogućnost podešavanja analizatora potrošnje snage. U analizi se može koristiti konkretan ulazni fajl sa simuliranim ponašanjem dizajna na osnovu kojeg se procenjuje snaga ili se može koristiti proces analize snage gde se ulazni signali definišu samo sa verovatnoćom tranzicija. Treba imati na umu da se potrošnja snage sastoji od statičke komponente i dinamičke komponente. Statička komponenta zavisi samo od veličine dizajna i samog izabranog FPGA čipa, dok dinamička komponenta zavisi od tranzicija signala u dizajnu. Što je veći broj tranzicija, veća je potrošnja snage, naročito u CMOS tehnologiji. Naime, u statičkom stanju postoje samo veoma male struje curenja, ali kada dođe do tranzicije, tranzistori tada zbog promena stanja provode 'veće' struje čime se povećava dinamička potrošnja. Otuda je važno u analizi uvek proceniti najgori slučaj koliko je to naravno moguće, da bi se dobila kvalitetna procena maksimalne potrošnje snage jer je ona bitna za dimenzionisanje izvora napajanja, koje mora biti sposobno da podnese i vršne vrednosti potrošnje snage, jer u suprotnom bi dolazilo do nestabilnosti u radu napajanih komponenti u koje spada i FPGA čip. Očigledno, najgori slučaj bi podrazumevao što veći broj tranzicija u dizajnu.

SSN Analyzer meni omogućava podešavanje opcija SSN analizatora koji analizira uticaj naponskih smetnji između pinova uzrokovanih promenama vrednosti signala na pinovima. Ovaj analizator omogućava da se uradi planiranje rasporeda pinova dizajna, tj. na koje pinove FPGA

čipa treba vezati portove top level entiteta, a da se izbegne potecijalni pogrešan rad dizajna usled međusobnih naponskih smetnji pinova FPGA čipa.

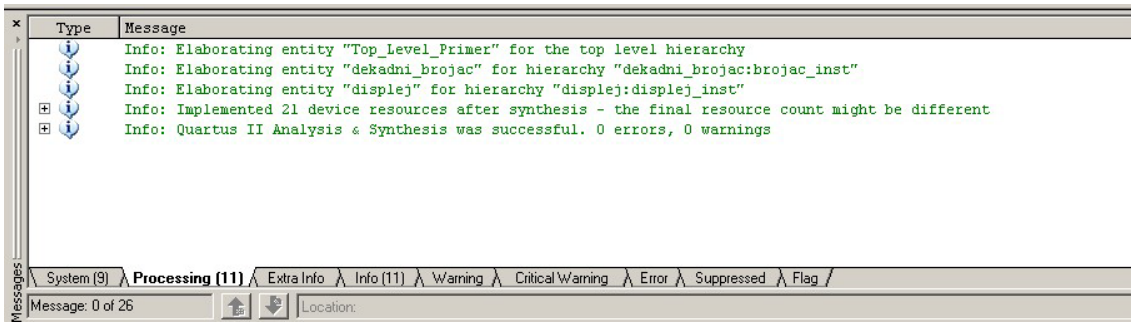
3.2.5. Analiza i sinteza

U ovoj sekciji ćemo opisati kako izvršiti analizu i sintezu dizajna, kao prvog koraka procesa kompajliranja dizajna. Kada smo uneli dizajn i definisali top level entitet, proces analize i sinteze se može pokrenuti na nekoliko načina. Prvi način je upotrebom dugmeta iz reda ikonica u glavnom prozoru (slika 3.2.5.1). Drugi način je biranjem opcije **Processing->Start->Start Analysis & Synthesis** iz glavnog menija. Treći način je kombinacijom CTRL+K tastera. Četvrti način je dvostrukim klikom na **Analysis & Synthesis** opciju u toku projekta (slika 3.2.5.1). Važno je napomenuti da se dvostrukim klikom na bilo koju opciju iz toka projekta, koja kod sebe ima strelicu, pokreće dotična opcija, odnosno taj korak kompajliranja. Pri tome, ukoliko neki od prethodnih koraka procesa kompajliranja nisu izvršeni, prvo će se oni izvršiti, pa tek onda selektovana opcija. Ostali koraci procesa kompajliranja iza selektovane opcije neće biti izvršeni.



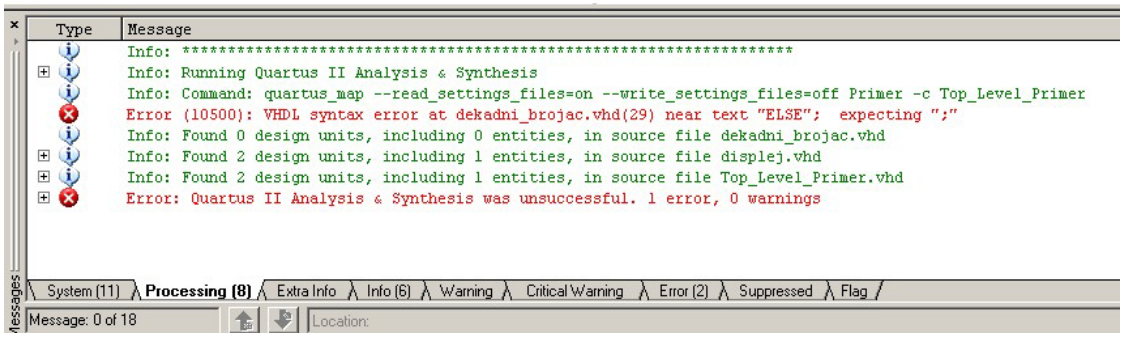
Slika 3.2.5.1. – Pokretanje analize i sinteze

U delu za ispis obaveštenja će biti ispisane odgovarajuće poruke koje prate tok kompajliranja (u ovom slučaju samo analize i sinteze) - slika 3.2.5.2. Tabovima se može filtrirati prikaz poruka tako da se prikažu samo poruke od interesa, na primer, samo poruke o greškama ili samo poruke upozorenja. Poruke o greškama treba obavezno pročitati da bi se utvrdile greške koje zatim treba korigovati, a poruke upozorenja treba pročitati i na osnovu njih treba utvrditi da li možda ima logičkih grešaka u dizajnu ili ne. Na primer, u porukama upozorenja može da piše da je usled procesa optimizacije uklonjen deo dizajna jer ne utiče ni na jedan izlaz. Ako je taj deo dizajna ipak trebao da utiče na bar jedan od izlaza, onda smo svesni da je u pitanju neka logička greška u dizajnu, u suprotnom ako smo svesni da taj deo dizajna zaista ne utiče ni na jedan izlaz, takvo upozorenje možemo ignorisati.

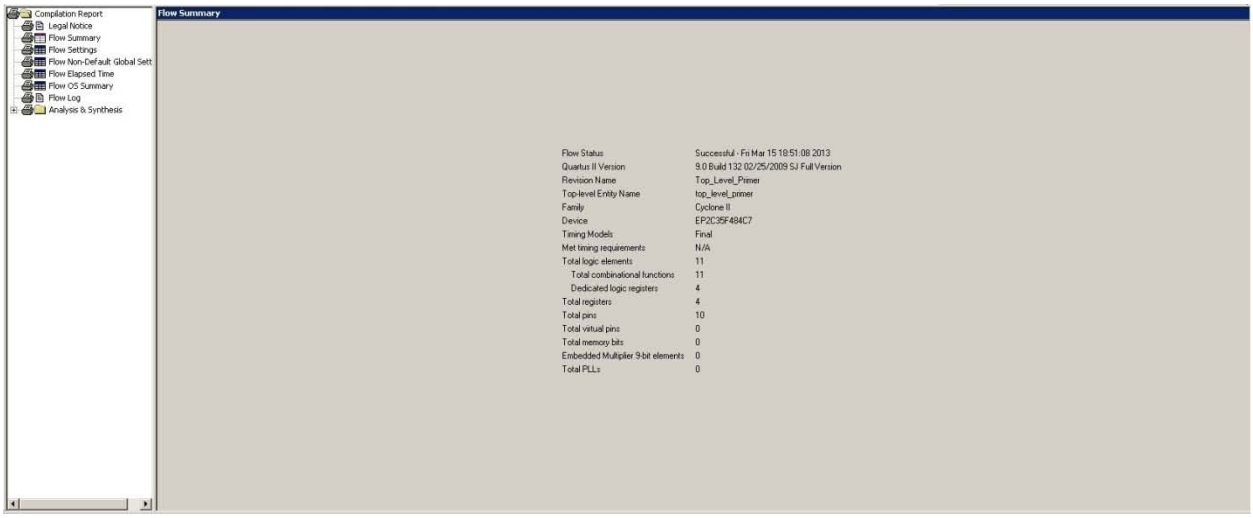


Slika 3.2.5.2. – Prikaz poruka kreiranih u toku izvršavanja procesa analize i sinteze

Na primer, ako na jednom mestu u kodu izostavimo tačku zarez na kraju jedne linije ispisaće se obaveštenje o grešci i lokaciji gde je greška uočena. U VHDL fajlu dekadni_brojac.vhd na kraju linije 28 izostavimo tačku zarez. Na slici 3.2.5.3 je prikazan ispis poruka među kojima je i poruka o uočenoj greški. U pitanju je sintaksna greška uočena kod linije 29 u VHDL fajlu dekadnog brojača. Ako nam nije jasna poruka o grešci, desnim klikom miša na poruku greške dobija se meni u okviru koga treba izabrati opciju **Help** koja otvara pomoćni prozor sa detaljnijim opisom tipa greške. Ista napomena za **Help** opciju važi i za ostale poruke.



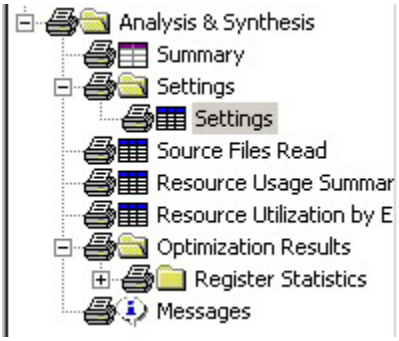
Slika 3.2.5.3. – Ispis poruke o uočenoj greški



Slika 3.2.5.4. – Ispis izveštaja obavljenog procesa analize i sinteze

Nakon uspešno izvršene analize i sinteze u centralnom delu prozora Quartus aplikacije se otvara izveštaj procesa kompajliranja prikazan na slici 3.2.5.4. U većem delu ekrana se nalazi sažeti izveštaj procesa analize i sinteze. Važne informacije su broj zauzetih internih resursa čipa. Međutim, pošto proces analize i sinteze još nije striktno vezan za sam konkretan FPGA čip, ovde je u pitanju veoma dobra procena, koja može da se razlikuje od konačne implementacije i raspoređivanja dizajna na sam čip u okviru procesa postavljanja i rutiranja. U koloni sa leve strane se mogu otvarati izveštaji procesa kompajliranja, a na slici 3.2.5.4 se nalazi samo deo vezan za proces analize i sinteze jer je jedino on izvršen. Kako se budu završavali i ostali delovi procesa kompajliranja, tako će i oni biti uključeni u ovu levu kolonu. Slika 3.2.5.5 pokazuje koji izveštaji su dostupni za proces analize i sinteze. **Summary** izveštaj je sažeti izveštaj koji je po difoltu otvoren i koji je prikazan na slici 3.2.5.4. **Settings** izveštaj pokazuje podešavanje svih parametara vezanih za proces analize i sinteze, za koje smo u prethodnoj sekciji videli gde se podešavaju. **Source Files Read** izveštaj ukazuje koji fajlovi projekta su korišćeni u procesu

analize i sinteze (u našem slučaju su korišćena sva tri VHDL fajla). **Resource Usage Summary** prikazuje detaljnije iskorišćenje internih resursa. **Resource Utilization by Entity** daje prikaz iskorišćenih resursa po entitetu, što je veoma dobra opcija za procenu koji entiteti su najkritičniji sa aspekta potrošnje internih resursa čipa (slika 3.2.5.6). Na primer, možemo uočiti da displej ne troši registre što je i bilo za očekivati jer je u pitanju kombinaciona logika, za razliku od dekadnog brojača koji koristi četiri flip-flopa (tj. registarska bita). **Optimization Results** daje informacije o eventualno izvršenim optimizacijama dizajna i takođe daje statistiku registara (preciznije flip-flopova) korišćenih u dizajnu (na primer, koliko njih koristi asinhroni reset, ili koliko njih koristi signal dozvole za takt i sl.). **Messages** daje ispis poruka koje su generisane u toku izvršavanja procesa analize i sinteze. Ovaj izveštaj je zgodan za filtriranje poruka tako da se prikažu samo one poruke koje su vezane samo za konkretan korak kompajliranja (u ovom slučaju procesa analize i sinteze) u slučaju da je više koraka kompajliranja završeno pa deo za ispis poruka prikazuje poruke iz svih izvršenih koraka kompajliranja.



Slika 3.2.5.5. – Izveštaji vezani za proces analize i sinteze

Analysis & Synthesis Resource Utilization by Entity

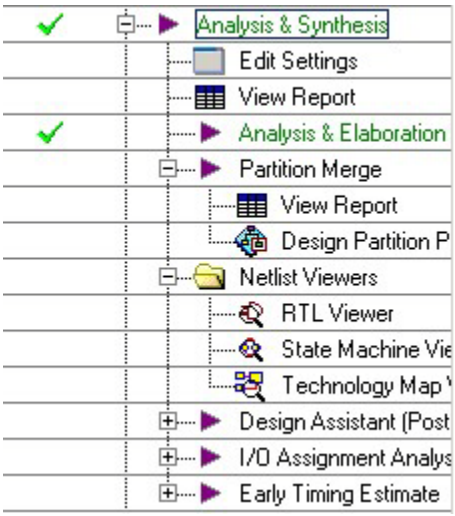
	Compilation Hierarchy Node	LC Combinationals	LC Registers	Memory Bits	DSP Elements	DSP 9x9	DSP 18x18	Pins	Virtual Pins	Full Hierarchy Name	Library Name
1	⊟ |top_level_primer	11 (0)	4 (0)	0	0	0	0	10	0	|top_level_primer	work
2	|dekadni_brojac:brojac_inst|	4 (4)	4 (4)	0	0	0	0	0	0	|top_level_primer|dekadni_brojac:brojac_inst	work
3	|displej:displej_inst|	7 (7)	0 (0)	0	0	0	0	0	0	|top_level_primer|displej:displej_inst	work

Slika 3.2.5.6. – Iskorišćenost resursa po entitetima

Podopcije koje su dostupne pod **Analysis & Synthesis** opcijom toka projekta su prikazane na slici 3.2.5.7. Treba primetiti da pokretanje procesa analize i sinteze izvršava samo osnovnu podopciju **Analysis & Elaboration** koja u stvari i predstavlja proces analize i sinteze. Ostale podopcije su opcione i izvršavaju se samo ako ih samostalno aktiviramo ili ako ih u podešavanju projekta aktiviramo (poput na primer **Design Assistant** podopcije čiju smo mogućnost podešavanja videli u prethodnoj sekciji). Sve **Edit Settings** podopcije unutar toka projekta otvaraju odgovarajući prozor kojim se podešava odgovarajuća opcija (neki od menija navedenih u prethodnoj sekciji). Tako, na primer, **Edit Settings** podopcija neposredno ispod **Analysis & Synthesis** opcije otvara prozor za podešavanje parametara procesa analize i sinteze prikazan na slici 3.2.4.5. Sve **View Report** podopcije unutar toka projekta prikazuju odgovarajući izveštaj (tipično **Summary** izveštaj) po principu prikazanom na slici 3.2.5.4, tj. u centralnom većem delu ekrana se prikazuje sam izveštaj, a u koloni sa leve strane se prikazuje meni koji omogućuje izbor ostalih dostupnih izveštaja.

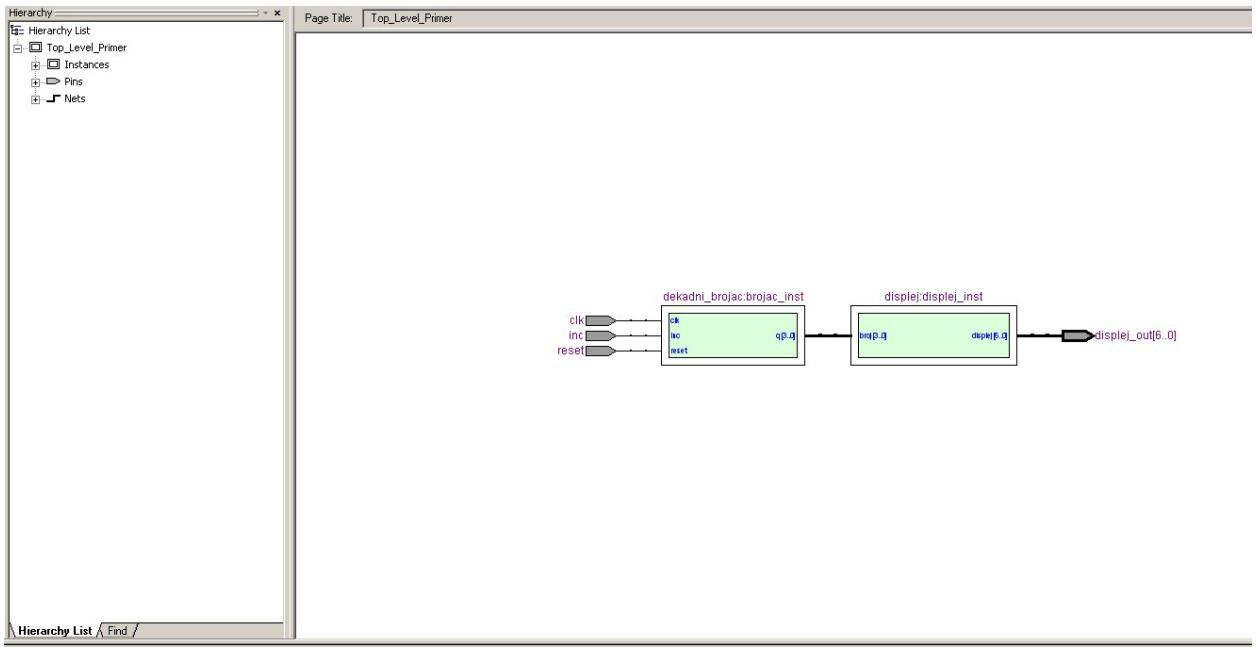
Partition Merge podopcija omogućava spajanje particija ukoliko je u dizajnu kreirano više particija. Nešto više reči o particijama će biti kasnije u skriptama. Takođe, **Design Partition**

Planner otvara prozor u kome se može videti pregled postojećih particija sa osnovnim podacima o njima.



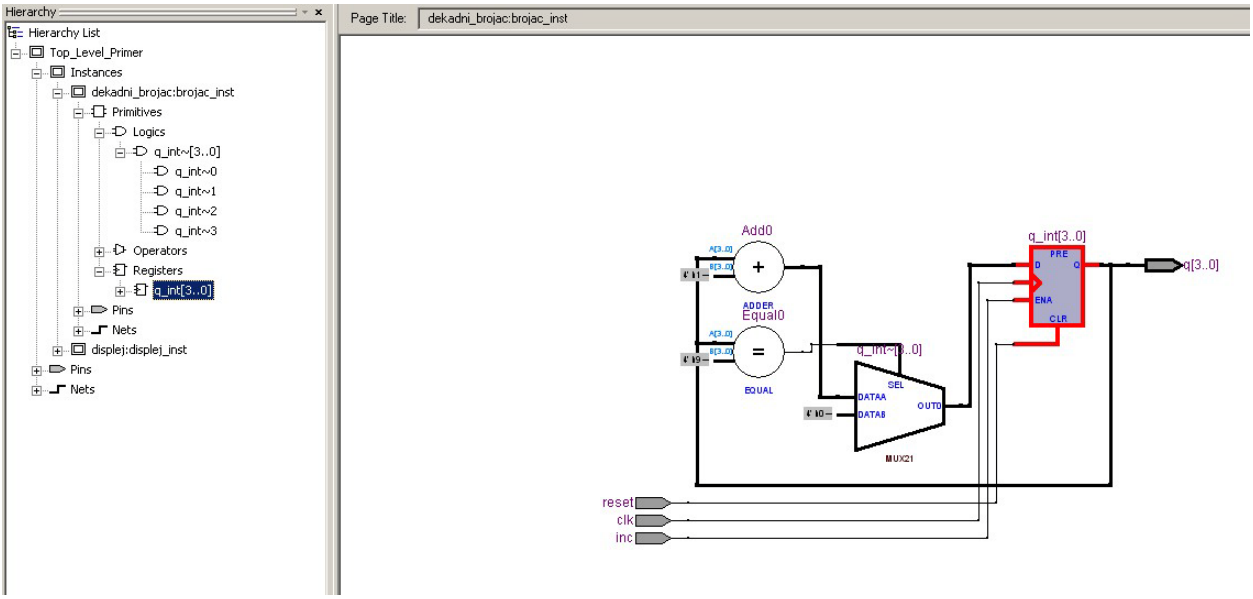
Slika 3.2.5.7. – Podopcije dostupne pod Analysis & Synthesis

Netlist Viewers podopcije omogućuju grafički prikaz dizajna nakon izvršenja procesa sinteze. Tako, **RTL Viewer** daje prikaz sheme dizajna koristeći hijerarhijski princip prikaza, pri čemu najniži nivo prikaza koristi primitivne (osnovne) elemente. Ova podopcija se aktivira duplim klikom na ovu podopciju u toku projekta ili izborom opcije **Tools->Netlist Viewers ->RTL Viewer** iz glavnog menija. Prikaz za dizajn korišćen u primeru projekta je dat na slici 3.2.5.8. Sa leve strane kolone se nalazi navigacija kojom možemo selektovati prikaz određenog dela dizajna, instance neke komponente, pina ili linije. Na slici 3.2.5.8 je prikazan najviši hijerarhijski nivo. Dvostrukim klikom na neku od prikazanih instanci prikazuje se njena unutrašnja struktura.



Slika 3.2.5.8. – Prikaz dizajna upotrebom podopcije RTL Viewer

Unutrašnja struktura brojača je data na slici 3.2.5.9. Pošto je u meniju sa leve strane selektovan registar u kom se čuva trenutno stanje brojača, ovaj registar je obeležen (crvenom bojom) u prikazu strukture dekadnog brojača. Na sličan način se obeležavaju i ostali delovi dizajna kada su selektovani u meniju sa leve strane. Meni sa leve strane omogućava tako šetanje na jednostavan način po čitavom dizajnu radi uvida u strukturu pojedinih delova dizajna. Na slici 3.2.5.8 možemo videti da u donjem delu menija sa leve strane postoji i tab **Find** koji se koristi za nalaženje određenog dela dizajna pomoću odgovarajućeg naziva dotičnog dela dizajna što je korisna opcija u slučaju veoma složenog dizajna za brzo nalaženje dela koji nas interesuje.

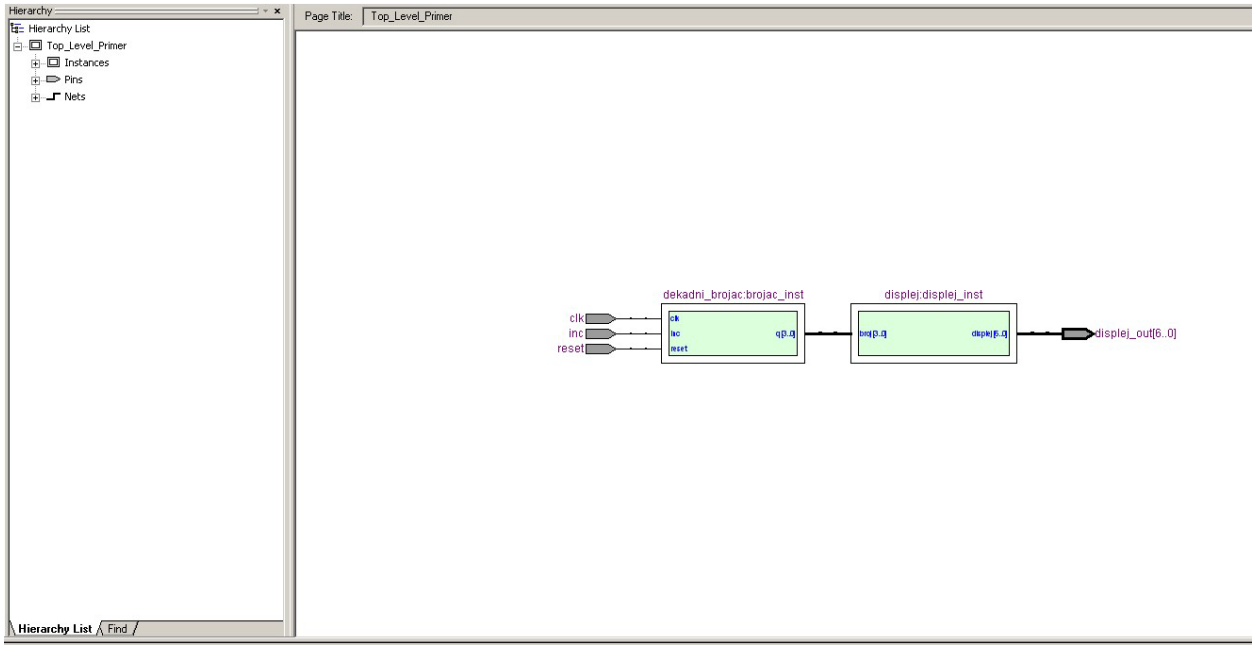


Slika 3.2.5.9. – Prikaz strukture dekadnog brojača

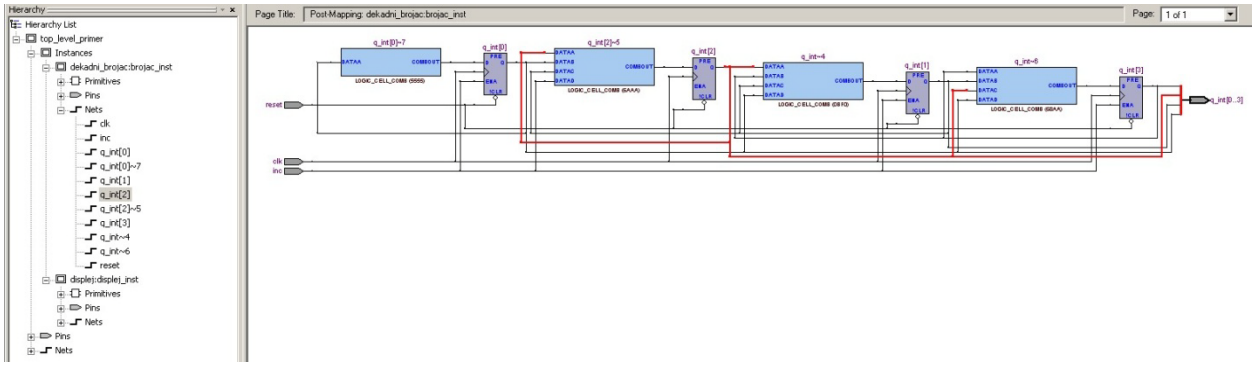
State Machine Viewer podopcija daje prikaz svih konačnih automata koji postoje u dizajnu. Prikaz ove podopcije će biti dat kasnije u skriptama jer korišćeni primer projekta ne sadrži konačni automat. Ova podopcija se aktivira ili dvostrukim klikom na ovu podopciju u toku projekta ili izborom opcije **Tools->Netlist Viewers->State Machine Viewer** iz glavnog menija.

Technology Map Viewer daje prikaz po sličnom principu kao **RTL Viewer**, ali prikaz uzima u obzir tehnologiju FPGA čipa, odnosno strukturu CLB blokova koji se nalaze u FPGA čipu. To, na primer, znači da se za formiranje kombinacionih funkcija uzima u obzir njihova realizacija pomoću LUT tabela. Prikaz dizajna korišćenog u primeru projekta sa stanovišta **Technology Map Viewer** je dat na slici 3.2.5.10. Dvostrukim klikom na neku od instanci se dobija prikaz njene unutrašnje strukture. Na slici 3.2.5.11 je prikazana struktura dekadnog brojača sa stanovišta **Technology Map Viewer**. Crvenom bojom je obeležena izabrana linija u meniju sa leve strane. Meni sa leve strane ima identičnu ulogu kao i u slučaju **RTL Viewer** prikaza. I u ovom slučaju se na dnu menija sa leve strane nalazi **Find** tab za lakše nalaženje pojedinih delova dizajna na osnovu naziva dotičnih delova koje tražimo. Treba primetiti da je prikaz u ovom slučaju vezan za tehnologiju čipa i obeležava flip-flopove i kombinacione delove sa stanovišta njihove strukture u samom izabranom FPGA čipu. Dvostrukim klikom na blok kombinacione funkcije koji u stvari predstavlja jednu LUT tabelu dobija se shematski prikaz kombinacione funkcije koju predstavlja dotična LUT tabela, odnosno blok kombinacione funkcije. Primer prikaza strukture dve kombinacione funkcije unutar dekadnog brojača je dat na slici 3.2.5.12. Podopcija **Technology Map Viewer** se aktivira ili dvostrukim klikom na ovu

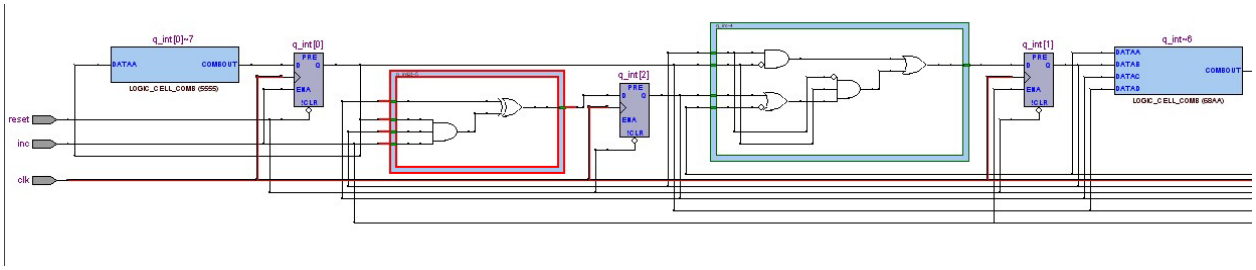
podopciju u toku projekta ili izborom opcije **Tools->Netlist Viewers->Technology Map Viewer** iz glavnog menija.



Slika 3.2.5.10. – Prikaz dizajna upotrebom podopcije Technology Map Viewer



Slika 3.2.5.11. – Prikaz dekadnog brojača sa stanovišta podopcije Technology Map Viewer



Slika 3.2.5.12. – Prikaz unutrašnje strukture kombinacione funkcije

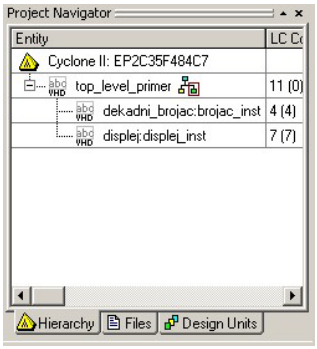
Pokretanjem **Design Assistant** podopcije (dvostrukim klikom) dobićemo upozorenje za asinhroni reset signal iz razloga objašnjenog u sekciji 3.2.4 i čije jedno potencijalno rešenje je prikazano na slici 3.2.4.4. U suštini, preporuka je da se koristi ili sinhroni reset u dizajnu ili bar da se koristi sinhrona deaktivacija reseta na način prikazan na slici 3.2.4.4. U slučaju ovog jednostavnog dizajna koji će raditi na spori signal takta (takt niske frekvencije) nema potrebe

menjati dizajn. Ako bi izmenili dizajn dekadnog brojača tako da umesto asinhronog reseta koristi sinhroni reset, tada bi dobili analizom i sintezom da se koristi 16 blokova kombinacione funkcije umesto 11 blokova koji su korišćeni u slučaju asinhronog reseta, što je i logično jer upotreba sinhronog reseta u principu povećava broj logičkih kola.

Podopcija **I/O Assignments Analysis** pruža analizu dizajna za izabrane pinove FPGA čipa na koje ćemo vezati portove top level entiteta. Da bi ova podopcija imala svrhu neophodno je izabrati pinove FPGA čipa na koje ćemo vezati portove top level entiteta što se radi u **Pin Planner** prozoru koji se otvara dvostrukim klikom na njega u toku projekta ili izborom opcije **Assignments->Pins** iz glavnog menija ili izborom opcije **Assignments->Pin Planner** iz glavnog menija. Proces izbora pinova na koje ćemo vezati portove top level entiteta će biti prikazan kasnije.

Podopcija **Early Timing Analysis** (aktivira se dvostrukim klikom) daje procenu vremenske analize dizajna. Naime, ona radi samo delimično proces postavljanja i rutiranja jer se optimizacija postavljanja dizajna na čip ne radi do kraja, i potom se vrši procena vremenske analize. Ideja ove podopcije je da bez aktiviranja dugotrajnog procesa optimizacije prilikom postavljanja i rutiranja dobijemo uvid u performanse dizajna sa stanovišta vremenske analize. Naravno, tek kad u potpunosti izvršimo proces postavljanja i rutiranja će vremenska analiza moći precizno da se uradi.

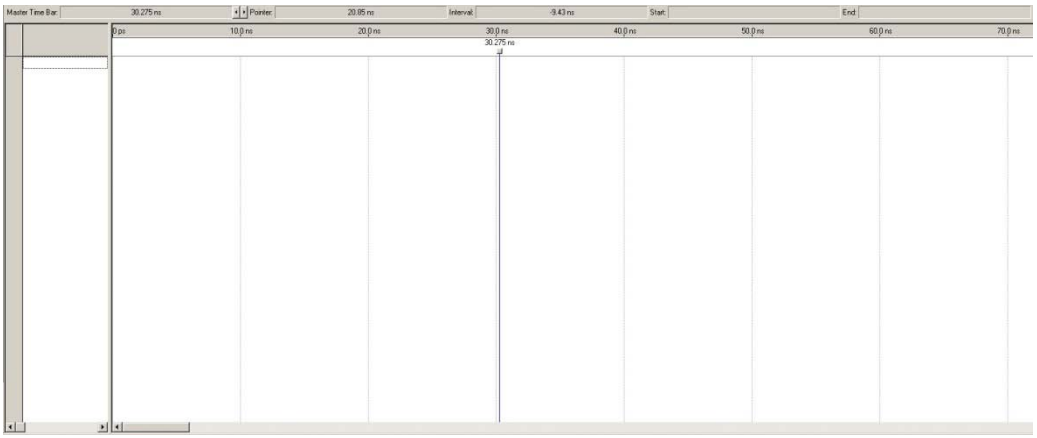
Napomenimo da će po okončanju procesa analize i sinteze, izborom taba **Hierarchy** u strukturi projekta, biti prikazana hijerahija entiteta pri čemu se na vrhu hijerarhije nalazi top level entitet - u ovom slučaju entitet top_level_primer (slika 3.2.5.13).



Slika 3.2.5.13. – Prikaz hijerarhije projekta

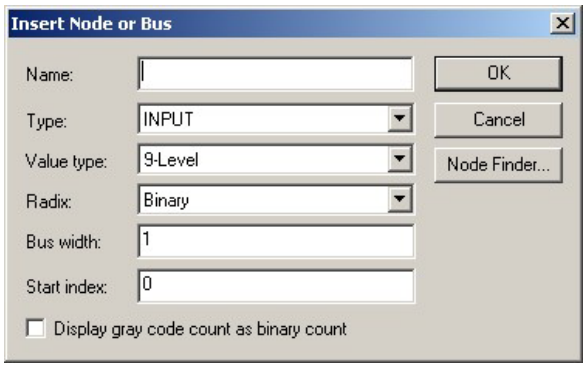
3.2.6. Funkcionalna simulacija

Nakon analize i sinteze poželjno je uraditi funkcionalnu simulaciju dizajna da bi se uverili da dizajn radi korektno sa logičkog stanovišta u idealnim uslovima. Na žalost, Quartus se uglavnom oslanja na alate drugih proizvođača i ugrađeni simulator nije u potpunosti udoban za složeni dizajn, ali uz pažljivo dizajniranje testiranja se čak i dizajn složene strukture može na adekvatan način testirati. Takođe, napomenimo da u slučaju složenog dizajna je poželjno prvo testirati pojedinačne delove, pa tek onda kompletan dizajn jer se tako greške znatno brže uoče i samim tim i isprave. U slučaju projektnog primera koji koristimo dizajn je veoma jednostavan i njegova simulacije nije problematična.



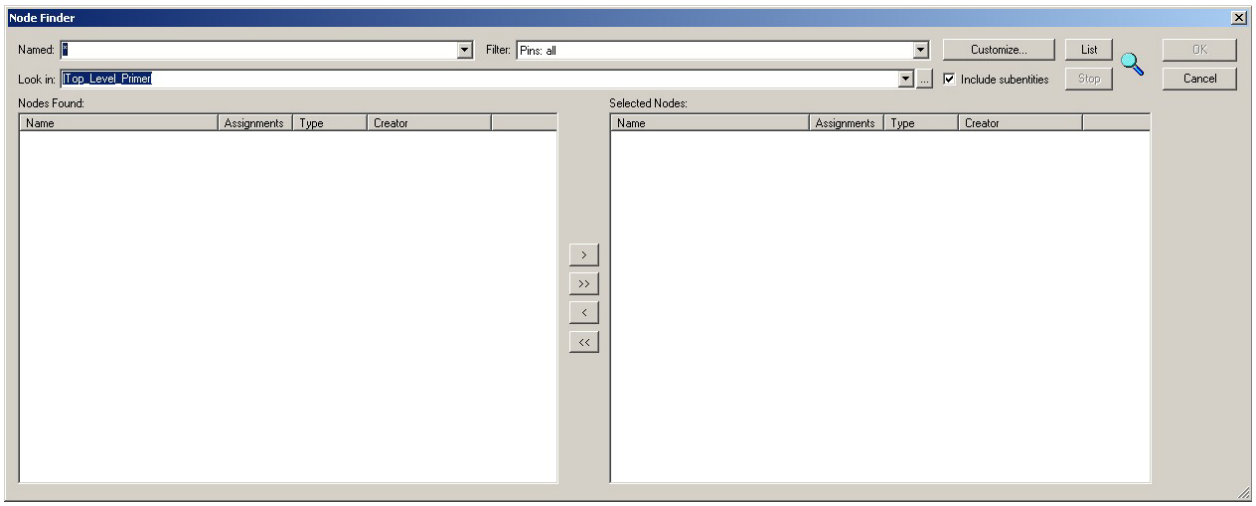
Slika 3.2.6.1. – Prozor za kreiranje .vwf fajla

Da bi se uopšte mogla koristiti funkcionalna simulacija neophodno je okončati proces analize i sinteze dizajna. Nakon toga je potrebno kreirati jedan simulacioni fajl (tzv. *Vector Waveform* fajl ekstenzije .vwf) koji će sadržati ulazne stimuluse, kao i listu signala čije će se vrednosti posmatrati tokom simulacije. Da bismo kreirali ovakav fajl, biramo opciju **Files->New** iz glavnog menija, pri čemu u prozoru koji se otvori selektujemo **Vector Waveform File**. Otvara se prozor u centralnom delu Quartus aplikacije prikazan na slici 3.2.6.1. U glavnom meniju pod opcijom **Edit->End Time** se može definisati trajanje stimulusa u kreiranom .vwf fajlu. Dužina trajanja se uvek može modifikovati, s tim što treba imati na umu da ako se vreme produžava svi stimulusi se kopiraju po kružnom principu. Na primer, ako trajanje produžavamo sa 200ns na 400ns, tada se vrednost svih do tada unetih stimulusa kopira iz prvih 200ns u dodatnih 200ns. Opcijom **Edit->Insert->Insert Node or Bus** iz glavnog menija ubacujemo stimuluse i signale koje ćemo posmatrati. Ovu opciju možemo izabrati iz menija koji se pojavi na desni klik miša u levu kolonu sa slike 3.2.6.1. Aktiviranjem ove opcije dobijamo prozor kao na slici 3.2.6.2.



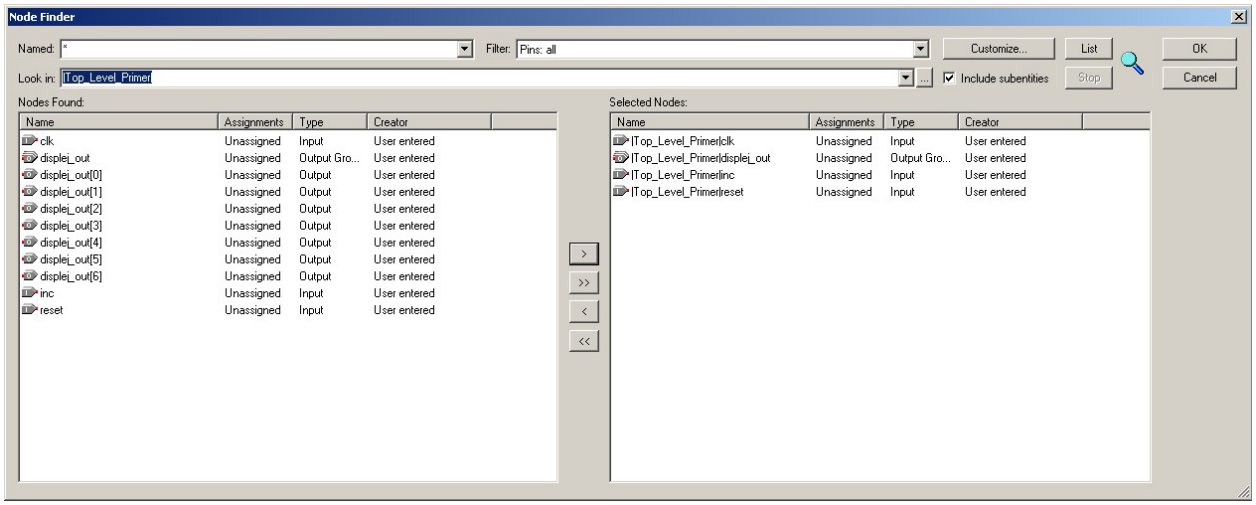
Slika 3.2.6.2. – Prozor za dodavanje signala u simulacioni .vwf fajl

Ako znamo ime signala koji želimo da posmatramo možemo ga ukucati u **Name** polje. U suprotnom koristimo dugme **Node Finder** za aktivaciju pretrage za željenim signalima. Ostala polja definišu osobine izabranog signala poput načina prikaza (binarni, heksadecimalni...). **Node Finder** prozor je prikazan na slici 3.2.6.3 i on je pogodan za brzo nalaženje i selekciju većeg broja čvorova koje hoćemo da posmatramo.

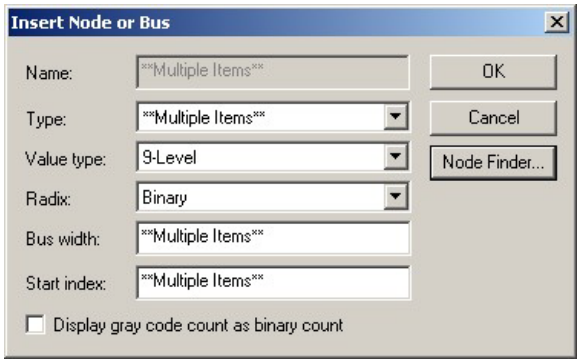


Slika 3.2.6.3. – Node Finder prozor

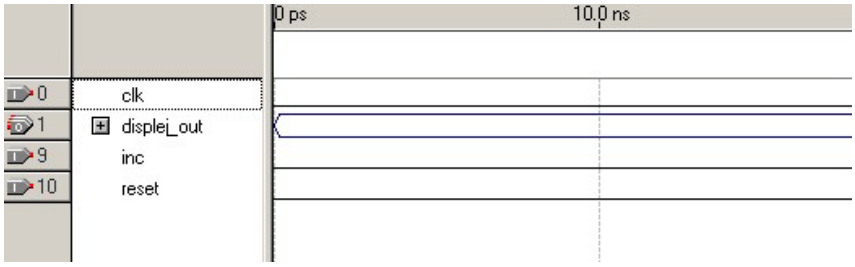
Named polje omogućava da zadamo naziv signala koji želimo da nađemo, gde * označava džoker znak. **Filter** padajući meni daje izbor tipa signala koji tražimo (svi pinovi, samo ulazni pinovi, svi signali...). **Look in** polje zadaje gde treba obaviti pretragu (u kom delu dizajna), pri čemu opcija **Include subentities** omogućuje da se pretraga vrši i u potomcima navedenog entiteta. **Nodes Found** polje predstavlja listu nađenih signala, a **Selected Nodes** listu selektovanih signala koje želimo da uključimo u kreirani .vwf fajl. Strelice između ovih polja omogućavaju prebacivanje signala iz jednog polja u drugo, pri čemu dvostruke strelice premeštaju kompletno polje. Izaberimo opciju **Pins: all** u **Filter** meniju i kliknimo **List** da bi aktivirali pretragu. Spisak nađenih pinova je prikazan na slici 3.2.6.4. Signali magistrale se prikazuju dvojako i kao magistrala i kao razdvojene linije. Selektujmo sve pinove, pri čemu izlaz displej_out selektujemo samo u vidu magistrale (slika 3.2.6.4). Selekciju više signala možemo raditi sa pritisnutim tasterom CTRL (isto kao i u tekstualnim dokumentima ili explorer aplikacijama). Klikom na OK potvrđujemo selekciju i tada se vraćamo na prozor za dodavanje signala (slika 3.2.6.5). Nakon klika na OK u ovom prozoru videćemo prikaz selektovanih signala u kreiranom .vwf fajlu - slika 3.2.6.6.



Slika 3.2.6.4. – Node Finder prozor

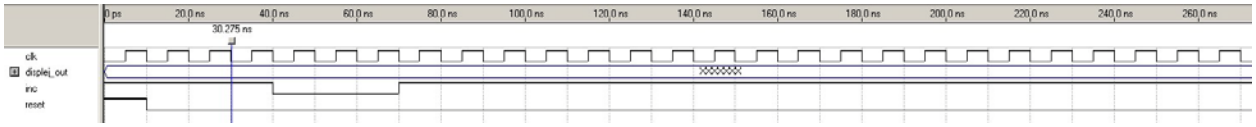


Slika 3.2.6.5. – Popunjen prozor za dodavanje signala



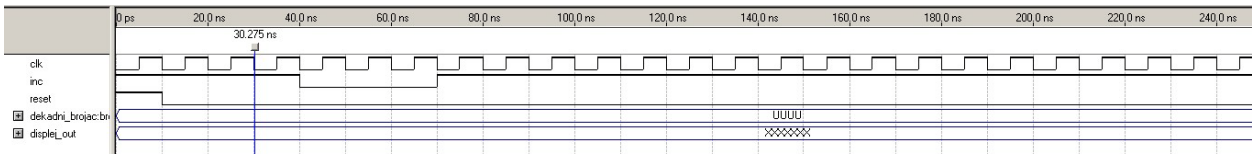
Slika 3.2.6.6. – Dodati signali u .vwf fajl

Desnim klikom na bilo koji od dodatih signala otvara se meni gde se selekcijom **Properties** opcije otvara meni za izbor prikaza selektovanog signala. Takođe, selekcijom **Value** opcije se zadaje vrednost signala (npr. logička jedinica). Omogućuje se u grafičkom delu koji prikazuje izgled signala (desna strana slike 3.2.6.6) selekcija dela signala gde se desnim klikom i izborom opcije **Value** podešava vrednost signala za selektovani opseg. Za signal *clk* izaberimo u **Value** opciji vrednost **Clock**. Pošto ćemo raditi funkcionalnu simulaciju, možemo ostaviti difolt vrednosti za takt jer se u funkcionalnoj analizi posmatraju idealni uslovi sem ako se u samoj simulaciji ne podese određeni vremenski kriterijumi, odnosno ograničenja. Za *reset* selektujmo početni deo i njega postavimo na vrednost logičke jedinice. Signal *inc* podesimo da ima vrednost logičke jedinice sem jednog kratkog opsega, nešto nakon deaktivacije signala *reset*. Izlazni signal *displej_out* (isto važi i za bilo koji interni signal) ne možemo definisati jer će on da dobije konkretnu vrednost tek tokom simulacije. Konačne vrednosti ulaznih signala su prikazani na slici 3.2.6.7. Napomenimo da u **View** grupi glavnog menija se nalaze opcije za zumiranje slike.



Slika 3.2.6.7. – Postavljene vrednosti ulaznih signala

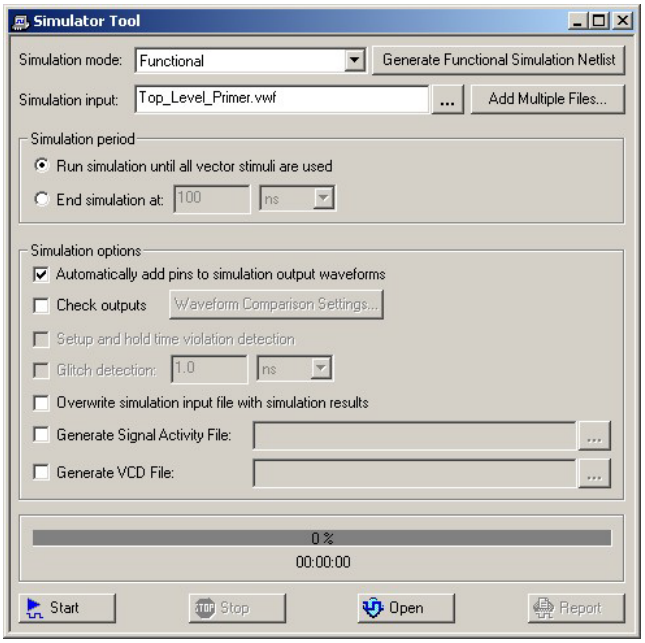
Dodajmo na kraju koristeći **Node Finder** opciju i interni signal *q_int* koji predstavlja sadržaj dekadnog brojača. Ovaj signal ćemo lako naći ako u **Filter** padajućem meniju izaberemo **Design Entry** opciju. Signali ubačeni u .vwf fajl mogu da se mišem vuku do željenih pozicija (*Drag & Drop*) i tako napravimo raspored signala kakav želimo. Na slici 3.2.6.8 je prikazan konačan izgled kreiranog .vwf fajla koji treba snimiti.



Slika 3.2.6.8. – Konačan izgled .vwf fajla

Napomenimo da prilikom selektovanja signala u **Node Finder** prozoru ne treba birati signale kombinacione logike (oznaka C na pinu sa leve strane od imena signala), već samo registre (oznaka R na pinu sa leve strane od imena signala) ili pinove (oznaka odgovara smeru pina). Razlog je što interni signali kombinacione logike ne mogu da se prikažu u toku simulacije. Nije greška ni ako se izabere neki signal koji ne može da se prikaže u simulaciji. Tada će po prikazu rezultata simulacije samo biti data informacija da dotični signali ne mogu biti prikazani.

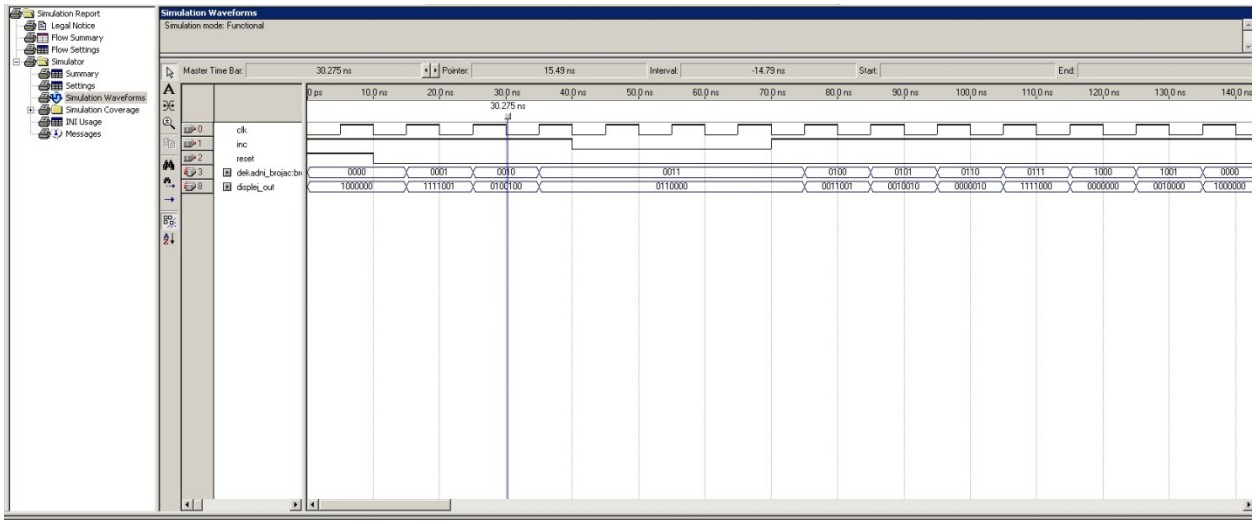
Nakon što smo kreirali fajl koji sadrži vrednosti stimulusa treba pokrenuti simulaciju. Ugrađeni alat za simulaciju se pokreće tako što se aktivira opcija **Processing->Simulator Tool** iz glavnog menija. Otvara se prozor za kontrolu simulacije prikazan na slici 3.2.6.9. U padajućem meniju **Simulation mode** biramo funkcionalnu simulaciju, a u **Simulation input** polje unosimo naziv .vwf fajla koji koristimo za generisanje ulaznih stimulusa (pored ovog polja postoji i dugme za brauzovanje do željenog fajla). **Simulation period** meni omogućava da biramo da simulacija traje dok se ne dođe do kraja simuliranih ulaza u izabranom .vwf fajlu ili specificiramo trajanje simulacije. U simulacionim opcijama biramo dodatne opcije za simulaciju poput mogućnosti da se selektovani .vwf fajl prepiše rezultatima simulacije ako želimo da sačuvamo rezultate simulacije. Takođe, interesantne su i poslednje dve opcije koje generišu dodatne fajlove na osnovu simulacije koji se potom mogu koristiti za procenu snage dizajna što ćemo videti nešto kasnije.



Slika 3.2.6.9. – Prozor ugrađenog simulatora

Pre pokretanja funkcionalne simulacije neophodno je kliknuti na dugme **Generate Functional Simulation Netlist** u gornjem desnom uglu čime se generiše netlista koja će se koristiti u procesu simulacije. Svaki put kad se izvrši neka promena u dizajnu, neophodno je

koristiti ovu funkciju pre započinjanja funkcionalne simulacije. Funkcionalna simulacija se startuje klikom na dugme **Start** u donjem levom ugla. Dugme **Stop** služi za nasilno okončavanje simulacije. Dugme **Open** otvara selektovani .vwf fajl u grafičkom editoru ukoliko želimo da ga editujemo. Dugme **Report** služi za prikaz rezultata simulacije. Pokrenimo simulaciju i nakon njenog završetka kliknimo na dugme **Report**. Prikazuje se prozor prikazan na slici 3.2.6.10.



Slika 3.2.6.10. – Prikaz rezultata simulacije

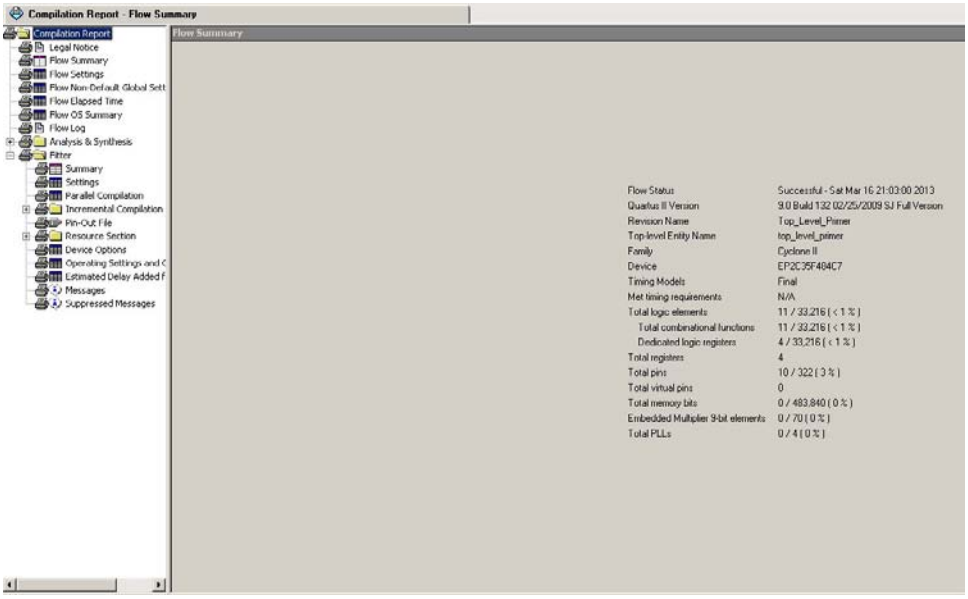
U koloni sa leve strane se nalazi meni sa opcijama prikaza izveštaja simulacije. **Summary** opcija daje sažetak izveštaja u vidu trajanja simulacije, broja tranzicija sa stanovišta svih signala ostvarenih tokom simulacije i sl. **Settings** opcija daje pregled podešavanja simulatora. **Simulation waveform** opcija daje grafički prikaz rezultata simulacije i ova opcija je prikazana na slici 3.2.6.10. **Simulation Coverage** opcija daje pregled pokrivenosti simulacije u vidu liste signala koji su tokom simulacije imali obe vrste tranzicije (sa '0' na '1' i obrnuto), liste signala koji nisu imali tranziciju sa '0' na '1' i liste signala koji nisu imali tranziciju sa '1' na '0'. Ova opcija može da bude zgodna ako želimo da utvrdimo na brz način da li smo pokrili sve moguće tranzicije (odnosno događaje) u dizajnu. **Messages** opcija daje spisak poruka koje su javile u toku simulacije, a te poruke su se inače pojavile i u delu za ispis poruka u donjem delu glavnog prozora. Ako pogledamo grafički prikaz rezultata simulacije, uverićemo se da brojač broji na uzlaznu ivicu takta samo kada je reset neaktivan, a signal inkrementa aktivan. Kada su i signal reseta i signal inkrementa neaktivni brojač ostaje u trenutnom stanju.

Napomenimo samo da je .vwf fajl tekstualnog oblika i da se nakon generisanja može menjati u bilo kom tekstualnom editoru ako to želimo, a ne samo putem grafičkog editora prikazanog u ovoj sekciji. Primera radi u kreiranom fajlu opis ponašanja ulaza *inc* je:

```
TRANSITION_LIST("inc")
{
        NODE
        {
                REPEAT = 1;
                LEVEL 1 FOR 40.0;
                LEVEL 0 FOR 30.0;
                LEVEL 1 FOR 930.0;
        }
}
```

Kao što vidimo izdvojeni segment definiše ponašanje ulaznog signala *inc*. Vremenska jedinica je deklarisana u zaglavlju (HEADER) .vwf fajla i u ovom slučaju je ona setovana na ns, što znači da su sve vremenske jedinice u prikazanom delu opisa ponašanja *inc* signala u ns. HEADER deo deklariše osnovne osobine .vwf fajla poput već pomenute vremenske jedinice, ali i dužine trajanja simulacije, početnog trenutka simulacije, itd. Iz datog primera, vidimo da *inc* signal ima vrednost '1' prvih 40ns, zatim vrednost '0' sledećih 30ns i onda narednih 930ns vrednost '1'. Sadržaj .vwf fajla se lako može razumeti i samim tim na jednostavan način modifikovati. Kao što smo videli, postoji TRANSITION_LIST svojstvo kojim opisujemo ponašanje signala. Takođe, za opis signala se koriste svojstvo SIGNAL koji opisuje osobine signala koje podešavamo preko prozora sa slike 3.2.6.2, i svojstvo DISPLAY_LINE koje podešava način (**Radix** opcija iz prozora sa slike 3.2.6.2) i poziciju prikaza signala. U principu sva podešavanja .vwf fajla treba obavljati u grafičkom editoru opisanim u ovoj sekciji, ali u slučaju kada želimo da postavimo velik broj različitih vrednosti nekih ulaznih signala to može biti prilično naporno upotrebom samo grafičkog editora i u tom slučaju za takve signale možemo u tekstualnom editoru uneti u njihova TRANSITION_LIST svojstva željena ponašanja, odnosno vrednosti takvih signala. Pri tome, treba ipak bekapovati .vwf fajl pre unosa izmena putem tekstualnog editora kao meru predostrožnosti u slučaju da nešto pogrešimo prilikom unosa izmena jer tada bi .vwf fajl bio neupotrebljiv s obzirom da ga grafički editor više ne bi mogao uspešno otvoriti.

3.2.7. Proces postavljanja i rutiranja



Slika 3.2.7.1. – Izveštaj procesa postavljanja i rutiranja

Nakon što smo funkcionalnom simulacijom potvrdili ispravnost rada kreiranog dizajna, treba pokrenuti proces postavljanja i rutiranja. Proces postavljanja i rutiranja se aktivira dvostrukim klikom na opciju **Fitter** u toku projekta ili biranjem opcije **Processing->Start ->Start Fitter** iz glavnog menija. Deo za ispis poruka će prikazivati poruke koje se generišu tokom izvršenja procesa postavljanja i rutiranja, pri čemu važe iste napomene koje su navedene u sekciji 3.2.5 (pasus ispod slike 3.2.5.1) za ispis poruka tokom procesa analize i sinteze. Nakon obavljanja procesa postavljanja i rutiranja, u centralnom delu glavnog prozora Quartus aplikacije se pojavljuje sažeti izveštaj (**Summary**) procesa postavljanja i rutiranja (slika 3.2.7.1). Ovaj

izveštaj je veoma sličan sažetom izveštaju procesa analize i sinteze, sa tom razlikom da se sada u obzir uzima i izabrani FPGA čip pa se u izveštaju navode i procenti iskorišćenja svih resursa koji su na raspolaganju u čipu. Leva kolona koja predstavlja meni za izbor izveštaja sada ima pridodate izveštaje generisane u procesu postavljanja i rutiranja.

Slika 3.2.7.1 pokazuje koji izveštaji su dostupni za proces postavljanja i rutiranja. **Summary** izveštaj je sažeti izveštaj koji je po difoltu otvoren po okončanju procesa postavljanja i rutiranja i koji je prikazan na slici 3.2.7.1. **Settings** izveštaj pokazuje podešavanje svih parametara vezanih za proces postavljanja i rutiranja, za koje smo već u sekciji 3.2.4 videli gde se podešavaju. **Parallel Compilation** izveštaj daje informaciju o tome koliko procesora je detektovano na računaru na kome se odvija proces kompajliranja (preciznije koraka postavljanja i rutiranja) i kako su oni iskorišćeni tokom procesa kompajliranja. **Incremental Compilation** sadrži izveštaje koji se odnose na proces inkrementalnog kompajliranja (podrazumeva se da je u pitanju korak postavljanja i rutiranja). Inkrementalno kompajliranje podrazumeva da se rekompajlira samo onaj deo dizajna koji je promenjen, dok se za nepromenjeni deo dizajna uzimaju rezultati prethodnog kompajliranja. Ovi izveštaji daju informaciju o tome koliko je efikasno bilo inkrementalno kompajliranje. Na primer, dobija se informacija koliko elemenata je trebalo da bude izuzeto iz rekomapjliranja, i koliko je zaista i bilo izuzeto iz rekompajliranja dizajna. **Pin-Out File** izveštaj daje pregled kako su povezani pinovi FPGA čipa. Svi portovi top level entiteta bivaju raspoređeni na pinove i to se vidi iz ovog izveštaja. Za sve neiskorišćene pinove je navedeno u ovom izveštaju kako se trebaju povezati sa okolinom na štampanoj ploči (većina tih pinova se tipično povezuje preko otpornika na masu štampane ploče, ali detalje treba ipak pročitati u samom izveštaju). Videćemo kasnije kako se mogu rasporediti portovi top level entiteta na željene pinove FPGA čipa. **Resource Section** sadrži više izveštaja koji se odnose na iskorišćenje resursa FPGA čipa. **Resource Usage Summary** izveštaj daje uvid u iskorišćenost svih resursa u čipu - broj iskorišćenih logičkih elemenata, koliko logičkih elemenata ima iskorišćeno samo kombinacioni deo, samo registarski deo (flip-flop) ili oboje, koliko je internih memorijskih blokova iskorišćeno, koliko je globalnih takt linija iskorišćeno, koliki je maksimalni *fan out* itd. *Fan out* predstavlja broj linija koji izlaze iz jednog izvorišta, tj. broj odredišta na koje jedno izvorište povezano. Tipično su u pitanju registri. **Input Pins** i **Output Pins** izveštaji daju detaljne podatke o pinovima na koje su vezani ulazni, odnosno izlazni portovi top level entiteta. U slučaju da top level entitet sadrži i bidirekcione portove postojao bi i ekvivalentan izveštaj **Bidir Pins** jer bi na takve pinove bili vezani bidirekcioni portovi top level entiteta. Među podacima se nalazi naziv pina, lokacija pina, ko je dodelio pin portu top level entiteta (korisnik ili kompajler) itd. **I/O Banks Usage** izveštaj daje pregled iskorišćenosti banaka FPGA čipa u vidu odnosa broja iskorišćenih pinova i ukupnog broja pinova svake banke. **All Package Pins** izveštaj daje pregled svih pinova, tj. predstavlja izveštaj ekvivalentan već pomenutom **Pin-Out File** izveštaju. **Output Pin Default Load For Reported TCO** izveštaj daje pregled podržanih standarda za I/O pinove i specificira difolt vrednost izlazne kapacitivnosti za svaki od standarda koja se koristi u procesu kompajliranja sem ako korisnik ne specificira drugačije (vrednosti koje je korisnik specificirao bi se pojavile u izveštajima za izlazne i bidirekcione pinove). **Resource Utilization by Entity** izveštaj daje pregled iskorišćenih resursa po entitetima dizajna (slika 3.2.7.2). Ovaj izveštaj je veoma sličan onome prikazanom na slici 3.2.5.6 iz procesa analize i sinteze, sa razlikom da se sada u kolonama precizno pojavljuju realni resursi FPGA čipa na koje su raspoređivani entiteti dizajna u procesu postavljanja i rutiranja.

Fitter Resource Utilization by Entity

	Compilation Hierarchy Node	Logic Cells	Dedicated Logic Registers	I/O Registers	Memory Bits	M4Ks	DSP Elements	DSP 9x9	DSP 18x18	Pins	Virtual Pins	LUT-Only LCs	Register-Only LCs	LUT/Register LCs	Full Hierarchy Name	Library Name
1	⊟ |top_level_primer	11 (0)	4 (0)	0 (0)	0	0	0	0	0	10	0	7 (0)	0 (0)	4 (0)	|top_level_primer	work
2	⊟ |dek:adni_brojac:brojac_inst|	4 (4)	4 (4)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	4 (4)	|top_level_primer|dek:adni_brojac:brojac_inst	work
3	⊟ |displej:displej_inst|	7 (7)	0 (0)	0 (0)	0	0	0	0	0	0	0	7 (7)	0 (0)	0 (0)	|top_level_primer|displej:displej_inst	work

Slika 3.2.7.2. – Iskorišćenost resursa po entitetima nakon okončanog procesa postavljanja i rutiranja

Control Signals

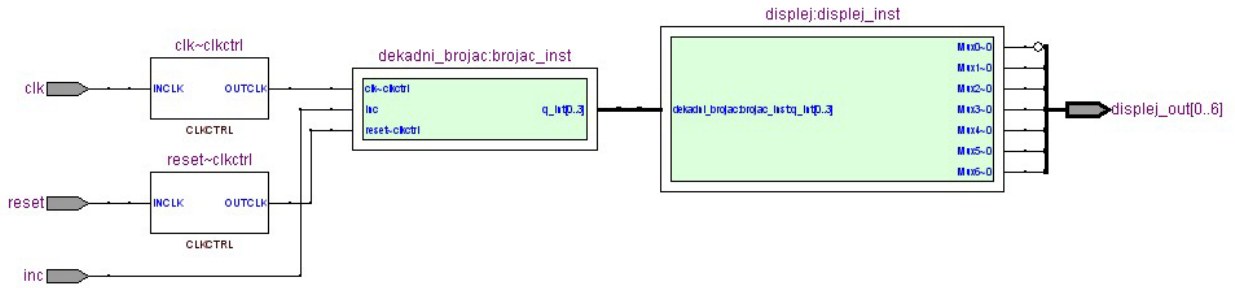
	Name	Location	Fan-Out	Usage	Global	Global Resource Used	Global Line Name	Enable Signal Source Name
1	clk	PIN_M1	4	Clock	yes	Global Clock	GCLK3	--
2	inc	PIN_A9	4	Clock enable	no	--	--	--
3	reset	PIN_M2	4	Async. clear	yes	Global Clock	GCLK1	--

Slika 3.2.7.3. – Pregled kontrolnih ulaznih signala

Delay Chain Summary daje sažeti pregled korišćenih elemenata kašnjenja kod I/O pinova. Naime, kod svakog I/O pina se nalazi dodatna logika koja se može konfigurisati (nešto slično unutrašnjim logičkim elementima, odnosno CLB blokovima). Ova dodatna logika sadrži i elemente za kašnjenje koji mogu da zakasne ulazni signal na ulaznom pinu, odnosno izlazni signal na izlaznom pinu. Integer brojevi, koji se javljaju u ovom izveštaju, definišu kako su konfigurisani odgovarajući elementi kašnjenja, odnosno definišu veličinu kašnjenja. Koliko kašnjenje odgovara navedenom integer broju se mora pogledati u dokumentaciji za izabrani FPGA čip. **Pad to Core Delay Chain Fanout** izveštaj daje detalje konekcije za korišćene elemente kašnjenja, odnosno daje informaciju na koje interne signale su oni vezani. **Control Signals** izveštaj daje pregled kontrolnih ulaznih signala (portova) dizajna, na koje pinove su vezani, tip kontrolnog signala i da li su rutirani globalnim linijama FPGA čipa (slika 3.2.7.3). Možemo videti da je kompajler definisao signal takta kao takt, signal reseta kao asinhroni reset, a signal inkrementa kao signal dozvole za takt. Signali takta i reseta su rutirani brzim globalnim linijama takta. **Global & Other Fast Signals** izveštaj daje pregled signala koji su rutirani brzim globalnim takt linijama. **Non-Global High Fan-Out Signals** izveštaj daje pregled signala koji se ne rutiraju brzim takt linijama i njihove *fan out* vrednosti. **Logic and Routing Section** sadrži izveštaje koji daju statistiku iskorišćenih linija interkonekcije u FPGA čipu kao i logičkih celina (LAB-ova u Alterinim čipovima) u čipu. **Interconnection Usage Summary** daje pregled iskorišćenosti po tipovima linija interkonekcije koje postoje u FPGA čipu. Preostali izveštaji u ovoj grupi se odnose na statistiku iskorišćenja LAB-ova koji predstavljaju osnovnu jedinicu u Cyclone II familiji koja je korišćena kao FPGA čip za dizajn koji koristimo kao primer. Preostali izveštaji daju pregled podešavanja korišćenih za čip poput pretpostavljenih radnih uslova. Ova podešavanja su korišćena tokom procesa postavljanja i rutiranja. **Messages** izveštaj daje pregled svih poruka generisanih u toku procesa postavljanja i rutiranja, i koje su takođe pojavile u delu za ispis poruka u dnu glavnog prozora Quartus aplikacije. **Suppressed Messages** daje pregled poruka koje su takođe generisane, ali koje se nisu pojavile u delu za ispis poruka tokom procesa postavljanja i rutiranja (otuda u nazivu ovog izveštaja reč potisnute).

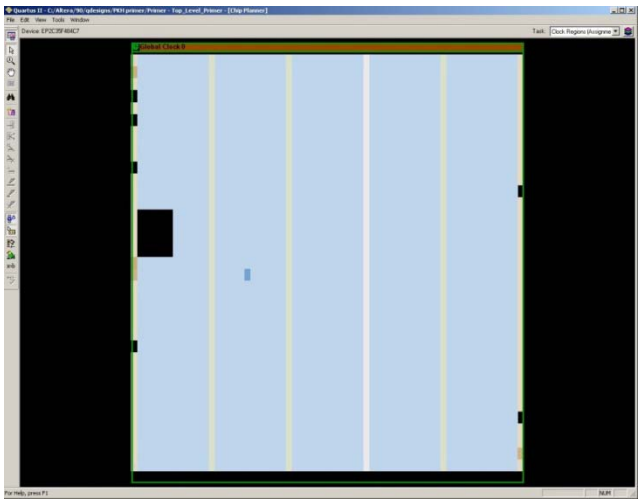
U okviru **Fitter** menija u toku projekta postoje i dodatne podopcije. Ponovo nailazimo na podopciju **Design Assistant** koji ima istu ulogu kao i kod procesa analize i sinteze. **Technology Map Viewer** daje šematski prikaz dizajna na čipu po sličnom principu kao u procesu analize i sinteze, s tim što se sada uzima u obzir konkretno zauzeće internih resursa čipa koje je određeno nakon izvršenja procesa postavljanja i rutiranja. Slika 3.2.7.4 daje šematski prikaz dizajna upotrebom ove opcije koja se aktivira ili dvostrukim klikom na ovu podopciju u toku projekta ili izborom opcije **Tools->Netlist Viewers->Technology Map Viewer (Post-Mapping)** u glavnom meniju. I ovde nakon otvaranja prozora sa šematskim prikazom sa leve strane stoji meni za brauzovanje kroz šematski prikaz i opcijama selekcije delova dizajna, kao i **Find** tab za

nalaženje delova dizajna. I dalje se dvostrukim klikom na neku instancu otvara u prozoru njena interna struktura.



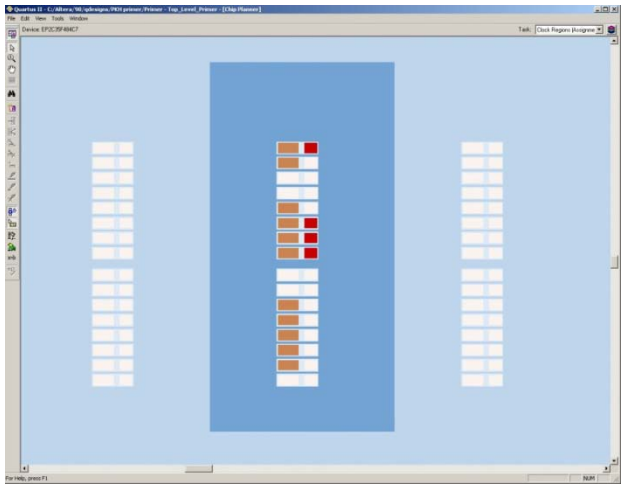
Slika 3.2.7.4. – Šematski prikaz dizajna nakon procesa postavljanja i rutiranja

Chip Planner podopcija otvara u potpuno novom prozoru prikaz dizajna na prostornoj shemi FPGA čipa. U stvari, ovom podopcijom vizuelno vidimo kako izgleda interna struktura izabranog FPGA čipa i raspored kompajliranog dizajna na tom čipu (slika 3.2.7.5).



Slika 3.2.7.5. – Prostorni raspored dizajna na FPGA čipu

Možemo, naravno, zumirati sliku radi preciznijeg pregleda dizajna u pojedinim delovima čipa (slika 3.2.7.6). Na slici 3.2.7.6 je prikazan LAB u kome je raspoređen naš dizajn.

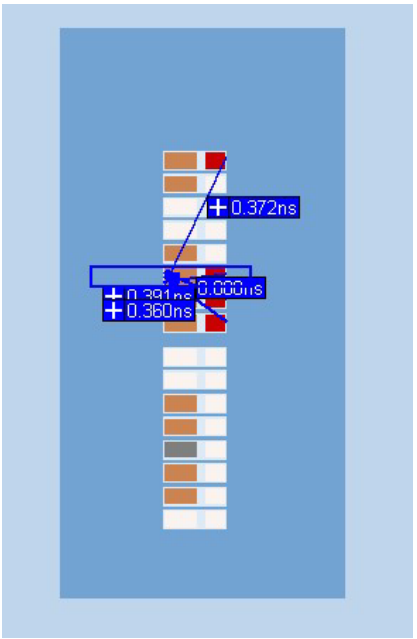


Slika 3.2.7.6. – Zumirani prikaz LAB-a u kom je raspoređen dizajn

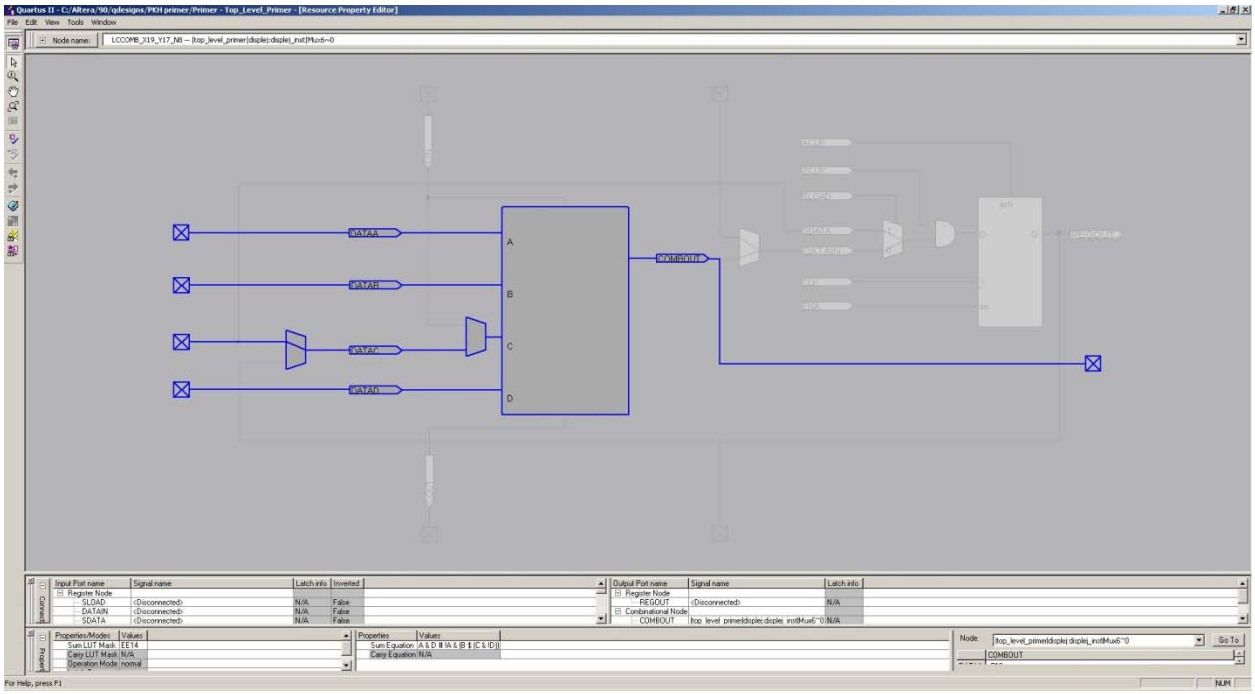
Takođe je moguće izborom opcije **View->Equations** otvoriti u donjem delu prozora ispis podataka o selektovanom elementu, poput koji signali ulaze u selektovani logički element, izraz (jednačina) konfigurisan u elementu i gde sve završavaju izlazi iz selektovanog logičkog elementa (slika 3.2.7.7). Takođe, postoje i opcije **Go To** za ulaze i izlaze selektovanog logičkog elementa koji nam olakšavaju praćenje signala između logičkih elemenata. Pored logičkih elemenata mogu se selektovati i pinovi, kao i specijalni resursi FPGA čipa poput internih memorija. U meniju sa ikonicama koji se nalazi sa leve strane se mogu selektovati i prikazi konekcija selektovanog elementa sa drugim elementima - sa kim su povezani izlazi selektovanog elementa ili ulazi selektovanog elementa. Isto može da se postigne **Generate Connections** opcijama u okviru **View** grupe. Na slici 3.2.7.8. su prikazane ulazne linije selektovanog logičkog elementa.

Slika 3.2.7.7. – Prikaz detalja logičkog elementa u donjem delu prozora

Dvostrukim klikom na jedan logički element u LAB-u dobijamo detaljan prikaz konfiguracije logičkog elementa kao na slici 3.2.7.9. U donjem delu otvorenog prozora se nalaze detalji konfiguracije i šta je raspoređeno u dotični logički element.



Slika 3.2.7.8. – Prikaz povezanosti ulaznih linija selektovanog logičkog elementa



Slika 3.2.7.9. – Prikaz konfiguracije logičkog elementa

Chip Planner podopcija može da posluži za vizuelnu inspekciju dizajna naročito ako postoji problem sa postignutim performansama u pogledu maksimalne frekvencije dizajna. Tada se može pokušati utvrditi da li može nekako da se optimizuje kritična putanja između dva registra sa najvećim kašnjenjem. Naravno, u praksi retko se može vizuelnim putem u slučaju složenog dizajna naći bolje rešenje od onoga što je našao kompajler, ali ipak imamo i tu opciju na raspolaganju.

3.2.8. Proces generisanja konfiguracionog fajla FPGA čipa

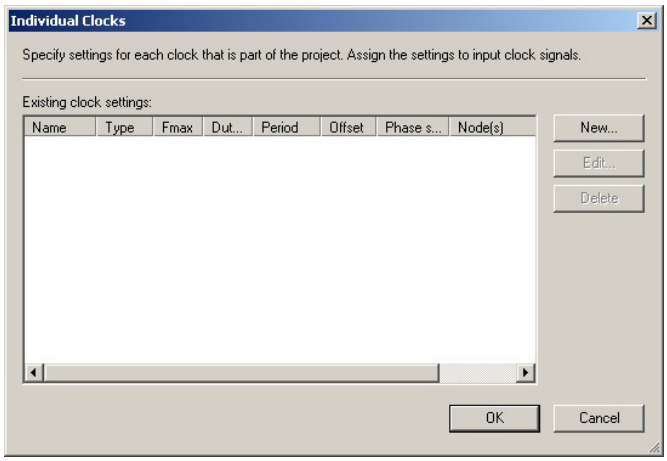
Nakon procesa postavljanja i rutiranja, treba pokrenuti proces generisanja konfiguracionog fajla FPGA čipa. Ovaj proces se pokreće dvostrukim klikom na opciju **Assembler** u toku projekta ili biranjem opcije **Processing->Start->Start Assembler** iz glavnog menija. Kao rezultat procesa se dobijaju fajlovi sa ekstenzijom .pof i .sof koji se mogu koristiti za konfigurisanje FPGA čipa tako da obavlja funkciju projektovanog dizajna.

3.2.9. Klasična vremenska analiza

U principu bi klasična vremenska analiza trebala da se radi pre generisanja konfiguracionog fajla, ali u difolt redosledu toka projekta u Quartus aplikaciji, ona ide nakon generisanja konfiguracionog fajla. Ova analiza treba da utvrdi da li projektovani dizajn nakon procesa postavljanja i rutiranja zadovoljava vremenska ograničenja dizajna. Na primer, ako kreiramo sekvencijalnu logiku koja treba da radi na taktu vrednosti 50MHz, vremenska analiza treba da utvrdi da li je podržana ova frekvencija za dizajn ili je, ipak, maksimalna podržana frekvencija dizajna nešto niža što bi značilo da moramo da modifikujemo dizajn ili da biramo drugi (napredniji) FPGA čip.

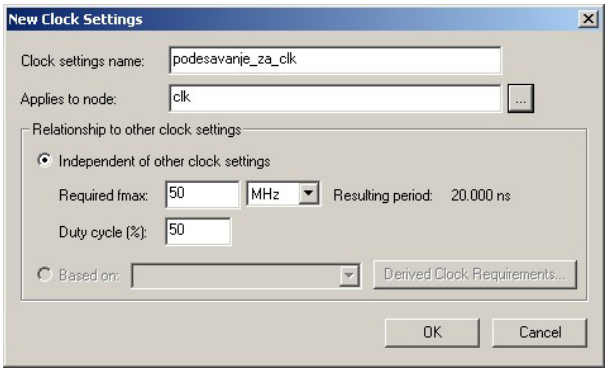
Pre pokretanja vremenske analize treba da konfigurišemo parametre klasične vremenske analize. Otvorimo **Assignments->Settings** opciju iz glavnog menija i u njoj izaberimo **Timing**

Analysis Settings meni u okviru koga treba da izaberemo **Classic Timing Analyzer Setting**s podmeni koji smo već prikazali na slici 3.2.4.10. U polje **Default required fmax** unesimo željenu maksimalnu frekvenciju dizajna, na primer, 50MHz. Kliknimo na dugme **Individual Clocks** kojim se otvara prozor prikazan na slici 3.2.9.1 koji sadrži listu taktova dizajna.



Slika 3.2.9.1. – Prozor za prikaz liste taktova dizajna

Kliknimo na dugme **New** kojim ćemo otvoriti prozor za selekciju takta kojeg ćemo ubaciti u listu (slika 3.2.9.2).

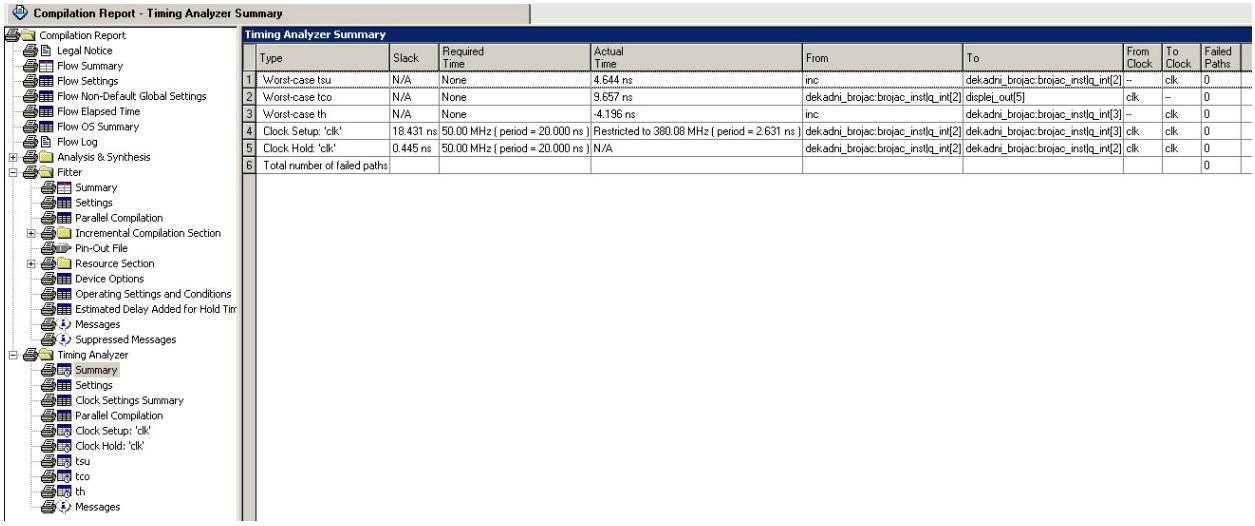


Slika 3.2.9.2. – Prozor za selekciju i podešavanje takta

U polje **Clock settings name** treba ubaciti naziv pod kojim će biti zavedeno podešavanje takta kojeg ćemo ubaciti u listu taktova dizajna. U polje **Applies to node** treba staviti naziv signala takta, a dugme pored tog polja otvara meni za nalaženje željenog signala ako ne znamo naziv signala. Meni za nalaženje signala takta je identičan onome sa slike 3.2.6.3 koji smo koristili za ubacivanje signala u .vwf simulacioni fajl. U polje **Required fmax** treba ubaciti maksimalnu frekvenciju takta koju želimo da postignemo. **Duty cycle** polje definiše odnos trajanja logičke nule i jedinice u periodičnom signalu takta. Na primer vrednost 50% označava da će one isto trajati. Klikom na OK, ovo podešavanje, odnosno signal takta će biti ubačen na listu. **Važno je napomenuti da je opisana podešavanja signala takta čak poželjno uraditi nakon izvršenog procesa analize i sinteze, a pre procesa postavljanja i rutiranja.** Razlog je što se ova podešavanja uključuju u proces postavljanja i rutiranja u cilju nalaženja rešenja koje će zadovoljiti postavljena podešavanja za signale takta u dizajnu. Nakon što smo obavili definisanje i podešavanje taktova dizajna, možemo pokrenuti klasičnu vremensku analizu. Ona se pokreće dvostrukim klikom na opciju **Classic Timing Analysis** u toku projekta ili izborom

opcije iz glavnog menija **Processing->Start->Start Classic Timing Analyzer** ili kombinacijom tastera CTRL+SHIFT+L.

Među izveštajima se sada nalaze pridodati i izveštaji vezani za vremensku analizu (slika 3.2.9.3). **Summary** izveštaj prikazan na slici 3.2.9.3 sadrži sažetak procesa vremenske analize u kom se nalaze najgori slučajevi za vremena t_{su}, t_{co} i t_h (čije su definicije date u sekciji 3.2.4). U slučaju da postoji samo kombinaciona logika od bar jednog ulaznog pina dizajna do bar jednog izlaznog pina dizajna biće dat i najgori slučaj za t_{pd} vreme. S druge strane ako dizajn sadrži samo kombinacionu logiku tada neće biti prikazana vremena u t_{su}, t_{co} i t_h izveštaju. Takođe se u ovom delu nalaze i podaci vezani za signale takta dizajna, njihovi zahtevi i postignute maksimalne frekvencije dizajna. U slučaju da je vremenska analiza utvrdila narušavanje zahteva nekog takta ili nekog drugog vremenskog ograničenja koje smo postavili (nije nađen prostorni raspored dizajna na čipu koji bi postigao željenu frekvenciju takta) to će biti signalizirano crvenom bojom teksta kod narušenog zahteva/ograničenja.



	Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock	Failed Paths
1	Worst-case tsu	N/A	None	4.644 ns	inc	dek:adni_brojac:brojac_inst|q_int[2]	--	clk	0
2	Worst-case tco	N/A	None	9.657 ns	dek:adni_brojac:brojac_inst|q_int[2]	displej_out[5]	clk	--	0
3	Worst-case th	N/A	None	-4.196 ns	inc	dek:adni_brojac:brojac_inst|q_int[3]	--	clk	0
4	Clock Setup: 'clk'	18.431 ns	50.00 MHz (period = 20.000 ns)	Restricted to 380.08 MHz (period = 2.631 ns)	dek:adni_brojac:brojac_inst|q_int[2]	dek:adni_brojac:brojac_inst|q_int[3]	clk	clk	0
5	Clock Hold: 'clk'	0.445 ns	50.00 MHz (period = 20.000 ns)	N/A	dek:adni_brojac:brojac_inst|q_int[2]	dek:adni_brojac:brojac_inst|q_int[2]	clk	clk	0
6	Total number of failed paths								0

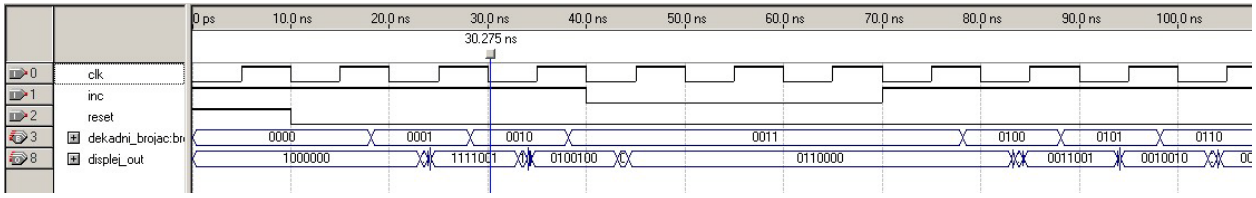
Slika 3.2.9.3. – Sažeti izveštaj vremenske analize

Settings izveštaj daje pregled podešavanja koja smo postavili za proces klasične vremenske analize. **Clock Settings Summary** daje pregled podešavanja za sve signale takta u dizajnu, pri čemu smo način podešavanja signala takta opisali na početku ove sekcije. **Parallel Compilation** izveštaj ima istu ulogu kao i u slučaju procesa postavljanja i rutiranja. **Clock Setup** i **Clock Hold** izveštaji daju listu internih signala (tačnije linija) poređanih od najgoreg slučaja do najboljeg slučaja sa stanovišta ostvarenih margina za *setup* i *hold* vremena. Kada je dizajn veoma složen i sadrži velik broj linija ovi izveštaji ograničavaju dužinu liste koja će biti prikazana. Napomenimo da crvena boja u ovim izveštajima označava linije koje narušavaju zadata vremenska ograničenja. Takođe, napomenimo da termin *slack* koji se javlja u ovim izveštajima označava marginu za koju je navedena linija ispunila ili nije ispunila zadato vremensko ograničenje. Pozitivna margina označava ispunjenje ograničenja (crna boja), a negativna margina označava da nije ispunjeno ograničenje (crvena boja). Izveštaji **tsu**, **tco** i **th** se generišu za pinove dizajna (portove top level entiteta) pri čemu su i oni poređani od najgoreg ka najboljem slučaju. Naziv izveštaja odgovara odgovarajućem vremenskom parametru t_{su}, t_{co} ili t_h koji su definisani u sekciji 3.2.4. Izveštaji **Clock Setup**, **Clock Hold**, **tsu**, **tco** i **th** se neće pojaviti ako se dizajn sastoji samo od kombinacione logike. Takođe, potencijalno postoji i **tpd** izveštaj koji odgovara vremenskom parametru t_{pd} ako postoji samo kombinaciona logika od bar jednog ulaznog do bar

jednog izlaznog pina dizajna. **Messages** izveštaj sadrži poruke koje su se javile tokom procesa vremenske analize. I ovde važe sve napomene u vezi poruka koje su navedene i kod **Messages** izveštaja procesa analize i sinteze.

3.2.10. Vremenska simulacija

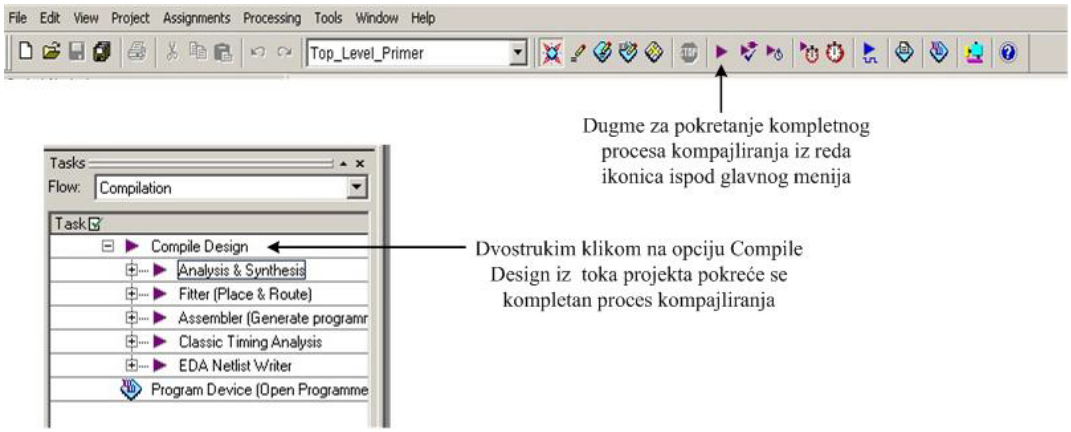
Vremenska simulacija je veoma slična funkcionalnoj simulaciji sa jednom značajnom razlikom - kašnjenja linija i elemenata nisu idealna, već realna i odgovaraju onima koja su dobijena vremenskom analizom. Da bi se mogla izvršiti vremenska simulacija mora biti izvršen proces postavljanja i rutiranja, jer se tada dobija stvaran raspored dizajna na FPGA čipu i samim tim se mogu izračunati realna kašnjenja linija i elemenata dizajna. U suštini, može se izvršiti i rana procena kašnjenja izborom podopcije **Early Timing Estimate** iz procesa analize i sinteze toka projekta, ali je bolje vremensku simulaciju raditi sa realnim vrednostima koje se mogu odrediti nakon izvršenja procesa postavljanja i rutiranja. Vremenska simulacija se aktivira i podešava identično kao i funkcionalna simulacija sa razlikom da je izabrana opcija **Timing** u padajućem meniju **Simulation Mode**. U slučaju vremenske simulacije dugme **Generate Functional Simulation Netlist** nije aktivno jer ono nije potrebno za vremensku simulaciju. Na slici 3.2.10.1 je prikazan izgled rezultata vremenske simulacije. Za razliku od funkcionalne simulacije sada uočavamo gličeve koji se javljaju usled uzimanja u obzir realnih kašnjenja dizajna. Međutim, napomenimo da vremenska simulacija nije od velikog značaja jer izveštaj vremenske analize ukazuje da li postoje narušena vremenska ograničenja u dizajnu ili ne. Tako da su najvažnije funkcionalna simulacija radi testiranja logičke ispravnosti dizajna i hardversko testiranje dizajna u FPGA čipu na samoj štampanoj ploči radi verifikacije konačne hardverske implementacije jer tu se uzimaju u obzir i konekcije sa drugim čipovima što razvojno okruženje ne može da pokrije u potpunosti.



Slika 3.2.10.1. – Rezultat vremenske simulacije

3.2.11. Pokretanje kompletnog procesa kompajliranja

U prethodnim sekcijama smo prolazili kroz korake procesa kompajliranja i navodili kako se svaki korak ponaosob aktivira. Međutim, često je zgodno odmah pokrenuti kompletan proces kompajliranja. Kompletan proces kompajliranja se pokreće klikom na odgovarajuće dugme iz reda ikonica ispod glavnog menija (slika 3.2.11.1), dvostrukim klikom na **Compile Design** opciju u toku projekta (slika 3.2.11.1), izborom opcije iz glavnog menija **Processing->Start Compilation** ili kombinacijom tastera CTRL+L. Pokretanjem kompletnog procesa kompajliranja izvršiće se proces analize i sinteze, proces postavljanja i rutiranja, proces generisanja konfiguracionog fajla FPGA čipa i proces vremenske analize.



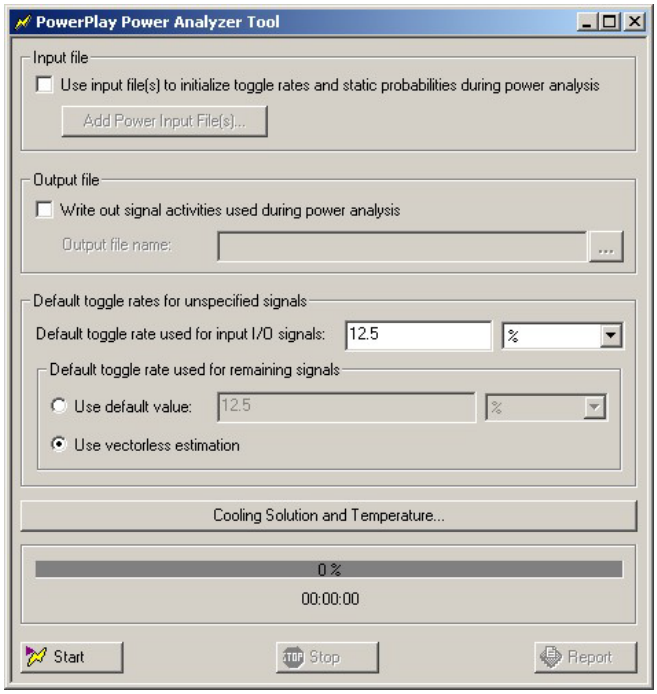
Slika 3.2.11.1. – Pokretanje kompletnog procesa kompajliranja

3.2.12. Analiza potrošnje snage dizajna

Uvek je bitno da sistem troši što manje energije. Za autonomne sisteme koji rade na baterije se time ostvaruje veća autonomija rada. Za sisteme koji imaju stalan izvor energije time se smanjuje kako ukupna potrošnja snage, tako se dobija i fleksibilniji dizajn jer se u sistem ugrađuju manji izvori napajanja koji tipično zauzimaju manje prostora na štampanoj ploči što je često od velikog značaja. Takođe, prilikom projektovanja je važno dobro proceniti potrošnju snage projektovanog hardvera da bi se ugradili dovoljni izvori snage, jer u suprotnom može doći do ispada iz rada sistema prilikom vršne potrošnje snage projektovanog hardvera. Projektovani dizajn će biti spušten na FPGA čip, a sam FPGA čip će biti deo neke štampane ploče tj. hardvera pa je važno proceniti i potrošnju snage projektovanog dizajna kada se njime programira izabrani FPGA čip, pošto ta potrošnja snage dizajna tj. FPGA čipa učestvuje u ukupnoj potrošnji hardvera. Na osnovu analize možemo odlučiti da li je potrebno da pokušamo optimizovati dizajn tako da on troši manje snage. Dizajn na FPGA čipu troši uvek neku statičku snagu, ali je veoma bitna dinamička komponenta. Što je veći broj tranzicija u dizajnu veća je i dinamička potrošnja snage. Otuda je često zgodno u slučaju sekvencijalnih kola imati mogućnost isključivanja i uključivanja takta kad se sekvencijalno kolo ne koristi, odnosno koristi. Na taj način kada se sekvencijalno kolo ne koristi, gašenjem takta sigurno ukidamo sve tranzicije signala (sam signal takta predstavlja signal sa najviše tranzicija) i time smanjujemo dinamičku potrošnju snage.

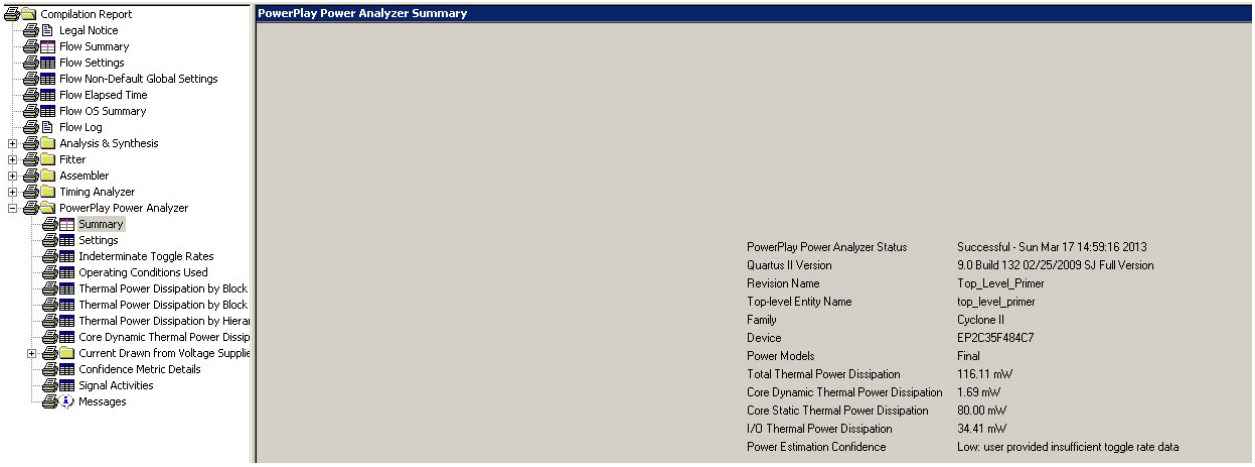
Prozor za analizu potrošnje snage dizajna, prikazan na slici 3.2.12.1, otvaramo izborom opcije **Processing->PowerPlay Power Analyzer Tool** iz glavnog menija. U okviru ovog prozora možemo selektovati ulazni fajl koji sadrži tranzicije signala i koji smo generisali prilikom funkcionalne ili vremenske simulacije dizajna. Ova selekcija je zgodna za generisanje realnih situacija koje se dešavaju u dizajnu. Naravno, ovde je poželjno koristiti rezultat simulacije (funkcionalne ili vremenske) koji predstavlja najgori slučaj sa stanovišta potrošnje, odnosno koji sadrži najveći broj tranzicija koji se javlja tokom rada dizajna jer je ova informacija neophodna za pravilan izbor dovoljno jakog izvora napajanja koji može da podrži vršne vrednosti potrošnje. S druge strane, ako koristimo rezultat simulacije koji predstavlja najčešće situacije koje se javljaju u dizajnu, ta procena potrošnje snage nam govori kolika bi trebala da bude prosečna potrošnja dizajna što je isto važna informacija. Ukoliko nemamo rezultate simulacije ili ona ne pokriva sve signale u dizajnu, meni **Default toogle rates for unspecified signals** daje mogućnost podešavanja učestanosti tranzicija za signale koji nisu pokriveni korišćenim ulaznim fajlom ili za sve signale ukoliko se ne koristi ulazni fajl sa rezultatima

simulacije. Za ulazne signale top level entiteta se u odgovarajuća polja unosi procenjena učestanost tranzicija tih signala. A za interne signale se takođe unosi učestanost ili se ostavlja samom simulatoru da izvrši tu procenu (opcija **Use vectorless estimation**). Opcija **Use vectorless estimation** se preporučuje samo za kombinacionu logiku, dok za sekvencijalnu logiku ne daje dobru procenu. Klikom na dugme **Cooling Solution and Temperature** se otvara prozor za podešavanje radnih uslova čipa čime se postiže preciznija procena potrošnje snage.



Slika 3.2.12.1. – Prozor za analizu potrošnje snage

Sa dugmetom **Start** se pokreće proces procene potrošnje snage dizajna. Sa dugmetom **Stop** se nasilno okončava proces procene. Klikom na dugme **Report** se otvara izveštaj procesa procene potrošnje snage dizajna. Pokrenimo proces procene bez korišćenja ulaznog fajla, pri čemu ostavimo intezitet tranzicija na difolt vrednosti od 12.5%, pri čemu aktivirajmo opciju **Use default value** umesto opcije **Use vectorless estimation**. Generisani izveštaji se pridodaju ostalim izveštajima koji su generisani tokom kompajliranja (slika 3.2.12.2).



Slika 3.2.12.2. – Prikaz izveštaja procesa procene potrošnje snage dizajna

Po difoltu se otvara **Summary** izveštaj koji predstavlja sažetak rezultata procesa procene potrošnje snage. Ovaj izveštaj sadrži procenu ukupne potrošnje, dinamičke i statičke komponente potrošnje, kao i potrošnje pinova (tj. I/O blokova) čipa. Takođe daje nivo kvaliteta procene. **Settings** izveštaj daje pregled podešavanja simulatora za procenu potrošnje čija smo podešavanja opisali u sekciji 3.2.4. **Indeterminate Toggle Rate** izveštaj daje podatke o signalima za koje nije mogla da se proceni učestanost tranzicija. **Operating Conditions Used** izveštaj daje pregled podešavanja radnih uslova koja su korišćena u procesu procene, pošto radni uslovi čipa nezanemarljivo utiču na potrošnju snage. **Thermal Power Dissipation by Block** izveštaj daje pregled potrošnje po delovima dizajna, pri čemu je potrošnja data kako ukupna, tako i po sastavnim komponentama potrošnje. Da bi se ovaj izveštaj generisao mora biti uključena opcija **Write power dissipation by block to report file** u podešavanjima simulatora za procenu snage (ova opcija nije po difoltu uključena). **Thermal Power Dissipation by Block Type** izveštaj daje pregled potrošnje po tipovima blokova FPGA čipa (I/O blokovi, registri, kombinacione ćelije itd.). I ovde je potrošnja prikazana kako ukupno, tako i u vidu sastavnih komponenti. **Thermal Power Dissipation by Hierarchy** izveštaj daje pregled potrošnje hijerarhijski po entitetima dizajna. I ovde je potrošnja prikazana kako ukupno, tako i u vidu sastavnih komponenti. **Core Dynamic Thermal Power Dissipation by Clock Domain** izveštaj daje pregled ukupne dinamičke potrošnje snage internih resursa FPGA čipa po svim domenima takta koji postoje u dizajnu**. Current Drawn from Voltage Supplies** sadrži više izveštaja koji se odnose na jačine struja koje se vuku iz izvora napajanja na koja je priključen FPGA čip. **Summary** daje pregled jačine struje koja se vuče iz izvora napajanja po izvorima napajanja, pri čemu se pored ukupne struje vide i statička i dinamička komponenta jačine struje. Gleda se jačina struje, a ne napon iz prostog razloga što izvori napajanja daju konstantnu vrednost napona, a u zavisnosti od potrošnje potrošača koji je priključen na izvor napajanja se razlikuje i jačina struje (u ovom slučaju potrošač je FPGA čip). **VCCIO Supply Current Drawn by I/O Elements** izveštaj sadrži pregled jačine struje po I/O bankama koja se vuče iz VCCIO izvora napajanja koji je zadužen za napajanje I/O blokova (I/O blok se često označava terminom I/O banka) FPGA čipa. **VCCIO Supply Current Drawn by Voltage** izveštaj sadrži pregled jačina struja po vrednostima napona izvora napajanja jer u nekim situacijama mogu da se koriste VCCIO izvori različitih vrednosti napona ako se koriste različiti standardi za I/O pinove. **Confidence Metric Details** daje pregled statistike tranzicija signala u izvršenoj proceni. **Signal Activities** izveštaj daje pregled inteziteta tranzicija po signalima koji su korišćeni u proceni potrošnje snage. Da bi se ovaj izveštaj generisao mora biti uključena opcija **Write signal activities to report file** u podešavanjima simulatora za procenu snage (ova opcija nije po difoltu uključena).

PowerPlay Power Analyzer Status	Successful - Sun Mar 17 15:31:12 2013
Quartus II Version	9.0 Build 132 02/25/2009 SJ Full Version
Revision Name	Top_Level_Primer
Top-level Entity Name	top_level_primer
Family	Cyclone II
Device	EP2C35F484C7
Power Models	Final
Total Thermal Power Dissipation	135.00 mW
Core Dynamic Thermal Power Dissipation	2.28 mW
Core Static Thermal Power Dissipation	80.07 mW
I/O Thermal Power Dissipation	52.65 mW
Power Estimation Confidence	Low: user provided insufficient toggle rate data

Slika 3.2.12.3. – Prikaz izveštaja procesa procene potrošnje snage dizajna za povećani intezitet tranzicija

Povećajmo sada u prozoru sa slike 3.2.12.1 intezitete tranzicija na 90%. **Summary** izveštaj za ovaj slučaj je prikazan na slici 3.2.12.3. Sada je procenjena ukupna potrošnja snage povećana i iznosi 135mW u odnosu na 116.1mW iz izveštaja prikazanog na slici 3.2.12.2 što je posledica povećanja inteziteta tranzicija signala. Takođe, možemo primetiti da je u oba slučaja statička komponenta ista (80mW). Povećala se dinamička komponenta sa 1.69mW na 2.28mW. Takođe se usled povećane dinamike (inteziteta tranzicija) ulaznih signala na ulaznim pinovima, kao i izlaznih signala na izlaznim pinovima povećala potrošnja I/O blokova sa 34.41mW na 52.65mW. Treba znati da I/O blokovi, pre svega izlazni (i bidirekcioni) pinovi u njima, troše dosta snage jer se oni povezuju sa (udaljenim) eksternim čipovima pa je neophodno generisati veću snagu izlaznog signala da bi on pouzdano stigao do svog odredišta van FPGA čipa, dok za internu komunikaciju delova u FPGA čipu nije potrebno generisati suviše velike snage signala jer su rastojanja mala, znatno manja nego između FPGA čipa i drugih čipova na istoj štampanoj ploči.

Aktivirajmo opciju **Generate Signal Acitivity File** u prozoru za pokretanje funkcionalne/ vremenske simulacije koji je prikazan na slici 3.2.6.9. Ovom opcijom će u fajlu ekstenzije .saf biti zapisani inteziteti tranzicija signala ostvareni tokom funkcionalne ili vremenske simulacije. Pokrenimo vremensku simulaciju da bi generisali .saf fajl. U prozoru za pokretanje procesa procene potrošnje snage prikazanom na slici 3.2.12.1 dodajmo generisani fajl putem **Input File** menija i startujmo procenu. **Summary** izveštaj procene je prikazan na slici 3.2.12.4. Kao što vidimo, procena je dosta slična onoj sa slike 3.2.12.3, ali nivo kvaliteta procene je sada veći.

PowerPlay Power Analyzer Status	Successful - Sun Mar 17 16:21:22 2013
Quartus II Version	9.0 Build 132 02/25/2009 SJ Full Version
Revision Name	Top_Level_Primer
Top-level Entity Name	top_level_primer
Family	Cyclone II
Device	EP2C35F484C7
Power Models	Final
Total Thermal Power Dissipation	135.83 mW
Core Dynamic Thermal Power Dissipation	3.81 mW
Core Static Thermal Power Dissipation	79.79 mW
I/O Thermal Power Dissipation	52.23 mW
Power Estimation Confidence	Medium: user provided moderately complete toggle rate data

Slika 3.2.12.4. – Prikaz Summary izveštaja procesa procene potrošnje snage sa uključenim .saf fajlom

Na sličan način, aktivacijom opcije **Generate VCD File** u prozoru za pokretanje funkcionalne/ vremenske simulacije sa slike 3.2.6.9 možemo generisati i fajl sa ekstenzijom .vcd koji sadrži rekonstrukciju vrednosti signala tokom simulacije. I ovaj fajl možemo uključiti u proces procene potrošnje snage putem već pomenutog **Input File** menija. Prilikom dodavanja fajlova (.saf ili .vcd) u proces procene potrošnje snage, možemo dodati samo .saf fajl, samo .vcd fajl ili oba fajla. Takođe, ako imamo više generisanih .saf ili .vcd fajlova (naravno, vezanih za isti dizajn, odnosno projekat) možemo ih sve uključiti u proces procene potrošnje snage. Napomenimo samo još na kraju da je .saf i .vcd fajlove bolje generisati u procesu vremenske simulacije, nego u procesu funkcionalne simulacije jer se tada dobijaju pouzdaniji rezultati procene potrošnje snage dizajna.

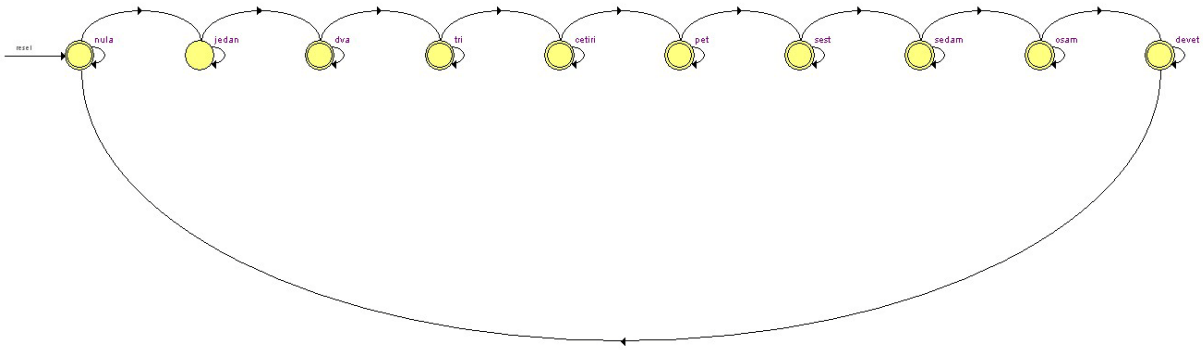
3.2.13. Primer konačnog automata

Izmenimo kod dekadnog brojača tako da obavlja svoju funkciju pomoću konačnog automata. VHDL kod modifikovanog dekadnog brojača je:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;--neophodno jer radimo sa STD_LOGIC i STD_LOGIC_VECTOR
```

```
--dekadni brojac koji kruzno broji unapred 0 do 9
ENTITY dekadni_brojac IS
PORT
(
        clk:    IN STD_LOGIC;--signal takta
        reset:  IN STD_LOGIC;--asinhroni signal reseta
        inc:    IN STD_LOGIC;--kontrolni signal za dozvolu brojanja
        q:      OUT STD_LOGIC_VECTOR(3 DOWNTO 0)--vrednost brojaca
);
END dekadni_brojac;

ARCHITECTURE shema OF dekadni_brojac IS
        TYPE stanja_brojaca IS (nula, jedan, dva, tri, cetiri, pet, sest, sedam, osam, devet);
        SIGNAL stanje:stanja_brojaca;
        SIGNAL q_int: STD_LOGIC_VECTOR(3 DOWNTO 0);--interno stanje brojaca
BEGIN

        PROCESS(reset,clk)
        BEGIN
                IF(reset='1')THEN--ako je asinhroni reset aktivan
                        stanje<=nula;--inicijalno stanje
                ELSIF(clk'EVENT and clk='1')THEN--uzlazna ivica takta
                        CASE stanje IS--definicija konacnog automata tj. njegovih tranzicija
                                WHEN nula=>
                                        IF(inc='1')THEN
                                                stanje<=jedan;
                                        END IF;
                                WHEN jedan=>
                                        IF(inc='1')THEN
                                                stanje<=dva;
                                        END IF;
                                WHEN dva=>
                                        IF(inc='1')THEN
                                                stanje<=tri;
                                        END IF;
                                WHEN tri=>
                                        IF(inc='1')THEN
                                                stanje<=cetiri;
                                        END IF;
                                WHEN cetiri=>
                                        IF(inc='1')THEN
                                                stanje<=pet;
                                        END IF;
                                WHEN pet=>
                                        IF(inc='1')THEN
                                                stanje<=sest;
                                        END IF;
                                WHEN sest=>
                                        IF(inc='1')THEN
                                                stanje<=sedam;
                                        END IF;
                                WHEN sedam=>
                                        IF(inc='1')THEN
                                                stanje<=osam;
                                        END IF;
                                WHEN osam=>
                                        IF(inc='1')THEN
                                                stanje<=devet;
                                        END IF;
```

```
                    WHEN devet=>
                        IF(inc='1')THEN
                            stanje<=nula;
                        END IF;
                END CASE;
            END IF;-- end if clk
    END PROCESS;

    PROCESS(stanje)
    BEGIN
                CASE stanje IS--vrednost brojaca u zavisnosti od stanja automata
                    WHEN nula=>
                        q_int<="0000";
                    WHEN jedan=>
                        q_int<="0001";
                    WHEN dva=>
                        q_int<="0010";
                    WHEN tri=>
                        q_int<="0011";
                    WHEN cetiri=>
                        q_int<="0100";
                    WHEN pet=>
                        q_int<="0101";
                    WHEN sest=>
                        q_int<="0110";
                    WHEN sedam=>
                        q_int<="0111";
                    WHEN osam=>
                        q_int<="1000";
                    WHEN devet=>
                        q_int<="1001";
                END CASE;
    END PROCESS;

    -- prosledjivanje internog stanja brojaca na izlaz entiteta
    q<=q_int;
END shema;
```



Slika 3.2.13.1. – Prikaz konačnog automata

Nakon završetka procesa analize i sinteze možemo videti, pomoću podopcije **State Machine Viewer** koju smo opisali u sekciji 3.2.5, izgled realizovanog konačnog automata. Prikaz konačnog automata je dat na slici 3.2.13.1. Prikaz daje shemu tranzicija između stanja konačnog automata. U donjem delu prozora koji prikazuje izgled konačnog automata postoje dva taba **Transitions** i **Encoding**. Tab **Transitions** prikazan na slici 3.2.13.2 daje pregled uslova tranzicija između stanja. Pod tranzicijom se podrazumeva i ostanak u istom stanju. Kao što

vidimo, uslov za ostanak u istom stanju je neaktivna vrednost signala inkrementa, a za prelazak u drugo stanje je uslov aktivna vrednost signala inkrementa. Signal reseta se ovde ne uzima u obzir jer je asinhron, iako njegova aktivna vrednost vraća automat u stanje *nula*. Da je signal reseta sinhron, tada bi signal reseta ušao u spisak uslova tranzicije između stanja. Tab **Encoding** daje prikaz binarnog kodiranja stanja konačnog automata. Izbor načina kodiranja stanja konačnog automata se bira tako što se otvori opcija dodatnih podešavanja (**More Settings**) u meniju **Analysis & Synthesis Settings** podešavanja parametara kompajlera koji je prikazan na slici 3.2.4.5. U prozoru koji se otvori klikom na **More Settings** dugme treba izabrati opciju **State Machine Processing** i nju podesiti na željenu vrednost. Difolt vrednost ove opcije je **Auto** što znači da kompajler sam bira način kodiranja stanja konačnog automata. Slika 3.2.13.3 prikazuje kodiranje sa minimalnim brojem bita.

	Source State	Destination State	Condition
1	nula	nula	(!inc)
2	nula	jedan	(inc)
3	jedan	jedan	(!inc)
4	jedan	dva	(inc)
5	dva	dva	(!inc)
6	dva	tri	(inc)
7	tri	tri	(!inc)
8	tri	cetiri	(inc)
9	cetiri	cetiri	(!inc)
10	cetiri	pet	(inc)
11	pet	pet	(!inc)
12	pet	sest	(inc)
13	sest	sest	(!inc)
14	sest	sedam	(inc)
15	sedam	sedam	(!inc)
16	sedam	osam	(inc)
17	osam	osam	(!inc)
18	osam	devet	(inc)
19	devet	nula	(inc)
20	devet	devet	(!inc)

Slika 3.2.13.2. – Prikaz konačnog automata

	Name	state_bit_3	state_bit_2	state_bit_1	state_bit_0
1	nula	0	0	0	0
2	jedan	0	0	0	1
3	dva	0	0	1	0
4	tri	0	0	1	1
5	cetiri	0	1	0	0
6	pet	0	1	0	1
7	sest	0	1	1	0
8	sedam	0	1	1	1
9	osam	1	0	0	0
10	devet	1	0	0	1

Slika 3.2.13.3. – Kodiranje stanja konačnog automata sa minimalnim brojem bita

Takođe treba primetiti da sada dizajn koji sadrži dekadni brojač realizovan pomoću konačnog automata troši više resursa od prvobitne verzije dizajna. Povećanje broja potrošenih resursa je posledica uvođenja logike konačnog automata. Konačni automati su zgodni za kreiranje dizajna čije se ponašanje može opisati upotrebom konačnog automata jer je tada i lakše razviti dizajn, a lakše ga je i testirati, odnosno verifikovati. Kao što smo videli iz datog primera konačni automat se tipično kreira upotrebom CASE konstrukcije. Pri tome prikazana varijanta, razdvaja konačni automat na dva dela. Prva (gornja) CASE konstrukcija definiše tranzicije između stanja konačnog automata, a druga (donja) CASE konstrukcija definiše akcije koje se preduzimaju u okviru trenutnog stanja. Druga varijanta konačnog automata je da se i tranzicije između stanja i akcije koje se izvršavaju u stanjima stave u zajedničku CASE konstrukciju. VHDL kod ove druge varijante bi tada bio:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;--neophodno jer radimo sa STD_LOGIC i STD_LOGIC_VECTOR

--dekadni brojac koji kruzno broji unapred 0 do 9
ENTITY dekadni_brojac IS
PORT
(
	clk:	IN STD_LOGIC;--signal takta
	reset:	IN STD_LOGIC;--asinhroni signal reseta
	inc:	IN STD_LOGIC;--kontrolni signal za dozvolu brojanja
	q:	OUT STD_LOGIC_VECTOR(3 DOWNTO 0)--vrednost brojaca
);
END dekadni_brojac;

ARCHITECTURE shema OF dekadni_brojac IS
	TYPE stanja_brojaca IS (nula, jedan, dva, tri, cetiri, pet, sest, sedam, osam, devet);
	SIGNAL stanje:stanja_brojaca;
	SIGNAL q_int: STD_LOGIC_VECTOR(3 DOWNTO 0);--interno stanje brojaca
BEGIN

	PROCESS(reset,clk)
	BEGIN
		IF(reset='1')THEN--ako je asinhroni reset aktivan
			stanje<=nula;--inicijalno stanje
			q_int<="0000";
		ELSIF(clk'EVENT and clk='1')THEN--uzlazna ivica takta
			CASE stanje IS--definicija konacnog automata tj. njegovih tranzicija
			--kao i definisanje izvrsavanja akcija
				WHEN nula=>
					q_int<="0000";--moze se izostaviti
					IF(inc='1')THEN
						stanje<=jedan;
						q_int<="0001";
					END IF;
				WHEN jedan=>
					q_int<="0001";--moze se izostaviti
					IF(inc='1')THEN
						stanje<=dva;
						q_int<="0010";
					END IF;
				WHEN dva=>
					q_int<="0010";--moze se izostaviti
					IF(inc='1')THEN
						stanje<=tri;
						q_int<="0011";
					END IF;
				WHEN tri=>
					q_int<="0011";--moze se izostaviti
					IF(inc='1')THEN
						stanje<=cetiri;
						q_int<="0100";
					END IF;
				WHEN cetiri=>
					q_int<="0100";--moze se izostaviti
					IF(inc='1')THEN
						stanje<=pet;
						q_int<="0101";
					END IF;
				WHEN pet=>
					q_int<="0101";--moze se izostaviti
```

```
                    IF(inc='1')THEN
                        stanje<=sest;
                        q_int<="0110";
                    END IF;
                WHEN sest=>
                    q_int<="0110";--moze se izostaviti
                    IF(inc='1')THEN
                        stanje<=sedam;
                        q_int<="0111";
                    END IF;
                WHEN sedam=>
                    q_int<="0111";--moze se izostaviti
                    IF(inc='1')THEN
                        stanje<=osam;
                        q_int<="1000";
                    END IF;
                WHEN osam=>
                    q_int<="1000";--moze se izostaviti
                    IF(inc='1')THEN
                        stanje<=devet;
                        q_int<="1001";
                    END IF;
                WHEN devet=>
                    q_int<="1001";--moze se izostaviti
                    IF(inc='1')THEN
                        stanje<=nula;
                        q_int<="0000";
                    END IF;
            END CASE;
        END IF;-- end if clk
END PROCESS;

-- prosledjivanje internog stanja brojaca na izlaz entiteta
q<=q_int;
END shema;
```

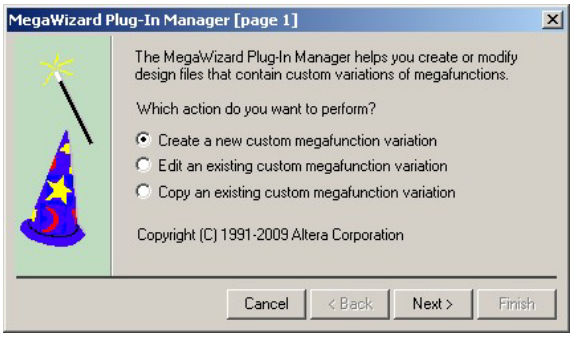
Ova druga varijanta konačnog automata definiše identično ponašanje dekadnog brojača kao i prva varijanta, ali troši više resursa od prve varijante u ovom konkretnom slučaju. U prvoj varijanti su se flip-flopovi, korišćeni za čuvanje informacije o stanju automata, koristili i za definisanje vrednosti brojača putem kombinacione logike koja je definisana u donjem procesu prve varijante. U drugoj varijanti imamo posebne flip-flopove za brojač, a posebne za čuvanje stanja automata. Takođe, za deo linija koda druge varijante je navedeno u komentarima da se mogu izostaviti. One su dodate samo za povećanje pouzdanosti sistema. Naime, ako sistem krene iz reseta onda su te linije nepotrebne, ali ako sistem po dobijanju napajanja ne krene iz reseta, onda su one neophodne da bi se izbegla neodređena vrednost na izlazu brojača. Naravno, ove dodatne linije dodatno povećavaju količinu upotrebljenih resursa. Napomenimo još da je u obe varijante umesto internog signala *q_int* mogao da se direktno koristi izlazni port *q*.

3.2.14. Upotreba megafunkcija

Quartus softverski paket ima na raspolaganju tzv. megafunkcije koje omogućavaju upotrebu specijalnih resursa FPGA čipa poput internih memorijskih blokova na jednostavan način. Isto tako, u megafunkcijama koje su na raspolaganju, se nalaze i često korišćene funkcionalnosti poput eternet kontrolera, PCI kontrolera i sl. U suštini megafunkcije predstavljaju takozvane IP (*Intelectual Property*) funkcionalnosti koje su razvili drugi dizajneri i

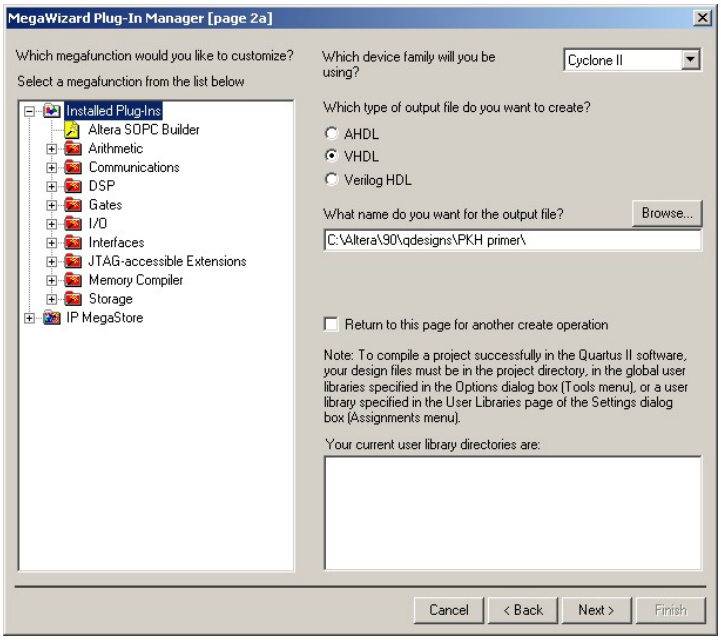
stavili (besplatno ili uz naknadu) na raspolaganje drugim korisnicima. **Važno je napomenuti da upotreba megafunkcija značajno olakšava razvoj složenog dizajna** jer nije potrebno samostalno razviti sve delove dizajna, već upotrebom megafunkcija možemo koristiti već gotove funkcionalnosti i fokusirati se samo na pojedine delove dizajna čime se znatno brže dolazi do konačnog rešenja.

Za otvaranje menija za izbor megafunkcije treba aktivirati opciju **Tools->MegaWizard Plug-In Manager** iz glavnog menija. Po aktiviranju ove opcije otvara se meni prikazan na slici 3.2.14.1. Ponuđene su tri mogućnosti. Prva mogućnost je kreiranje novog primerka megafunkcije, druga mogućnost je modifikacija ranije kreiranog primerka megafunkcije i treća mogućnost je kopiranje ranije kreiranog primerka megafunkcije. U primeru koji ćemo slediti izabraćemo prvu opciju - kreiranje novog primerka megafunkcije.



Slika 3.2.14.1. – Meni za izbor akcije u radu sa megafunkcijama

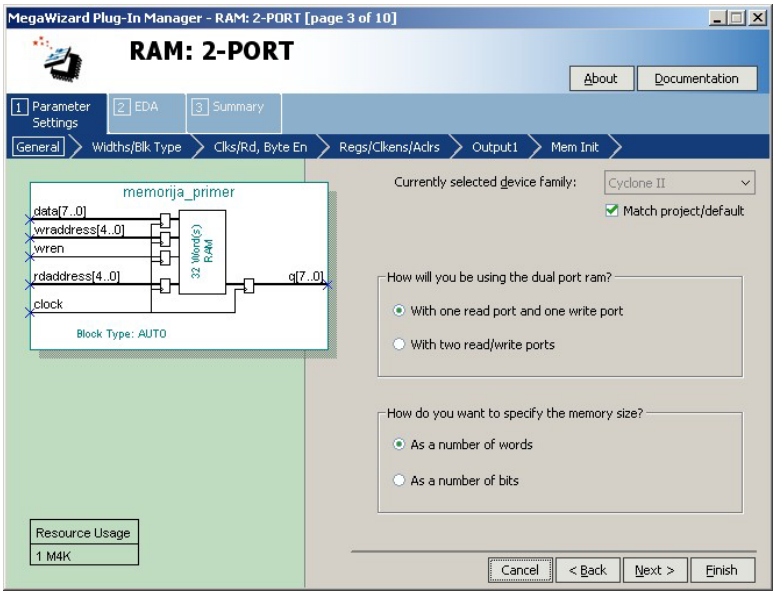
Nakon izbora opcije za kreiranje novog primerka megafunkcije i klika na dugme **Next** otvara se prozor sa izborom megafunkcija koje su na raspolaganju (slika 3.2.14.2). Megafunkcije koje su na raspolaganju su grupisane u logičke celine.



Slika 3.2.14.2. – Početni meni za izbor megafunkcije koju želimo da koristimo

Kao što se vidi sa slike 3.2.14.2, na raspolaganju su aritmetičke funkcije, DSP funkcije, interfejs funkcije itd. Na primer, u grupi interfejs funkcija se nalaze megafunkcije koje

omogućavaju kreiranje interfejsa po PCI, PCI-E, RapidIO i drugim standardima. U padajućem meniju, koji se nalazi u gornjem desnom uglu prozora, je moguć izbor familije čipa za koju se kreira primerak megafunkcije. Zatim, ispod padajućeg menija se nalazi meni sa opcijama programskog jezika u kom će biti kreiran primerak megafunkcije, pri čemu se nudi i VHDL jezik. U polju za definisanje lokacije i naziva primerka megafunkcije je potrebno uneti naziv megafunkcije pre nastavka procesa kreiranja novog primerka megafunkcije. Unesimo naziv memorija_primer, a u meniju sa desne strane izaberimo opciju **Memory Compiler->RAM: 2-PORT** koja omogućava kreiranje dvoportne RAM memorije. Kliknimo na dugme **Next** da bi nastavili sa kreiranjem primerka megafunkcije. Otvara se početni prozor u kreiranju primerka izabrane megafunkcije (slika 3.2.14.3). Kod svake megafunkcije se na početnom prozoru nalazi link ka dokumentaciji koja opisuje ulogu megafunkcije i načine podešavanja primerka megafunkcije, pošto tipično svaka megafunkcija ima više parametara koji mogu da se podese prilikom kreiranja primerka selektovane megafunkcije. U jednom projektu se može kreirati i koristiti više primeraka jedne iste megafunkcije, pri čemu svaki primerak može biti parametrizovan različito. Sam izgled prozora kroz koje se prolazi prilikom kreiranja primerka megafunkcije se razlikuje od megafunkcije do megafunkcije ne samo po sadržaju, već i po dizajnu samih prozora, što je posledica različitih grupa dizajnera koji su kreirali megafunkcije.

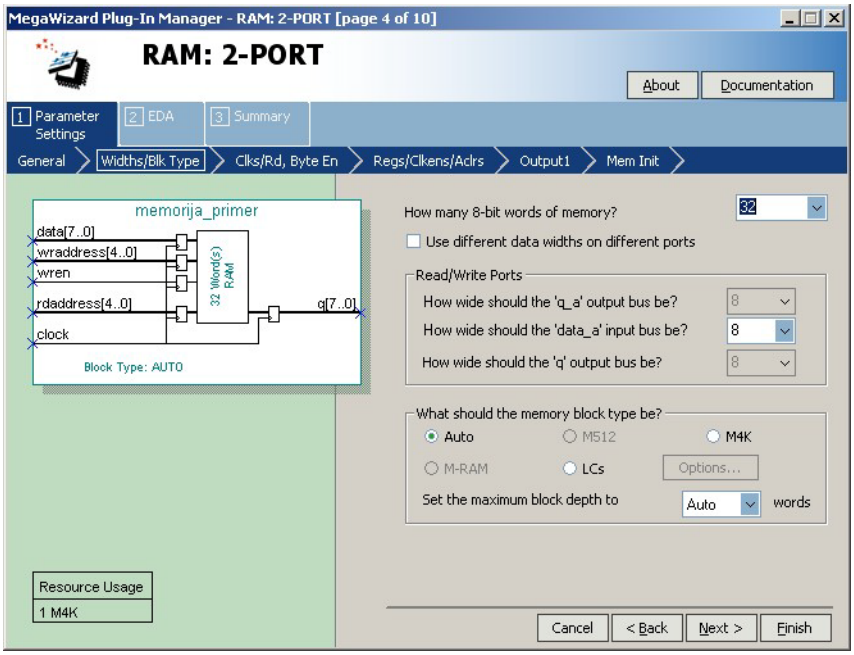


Slika 3.2.14.3. – Početni prozor za podešavanje dvoportne RAM memorije

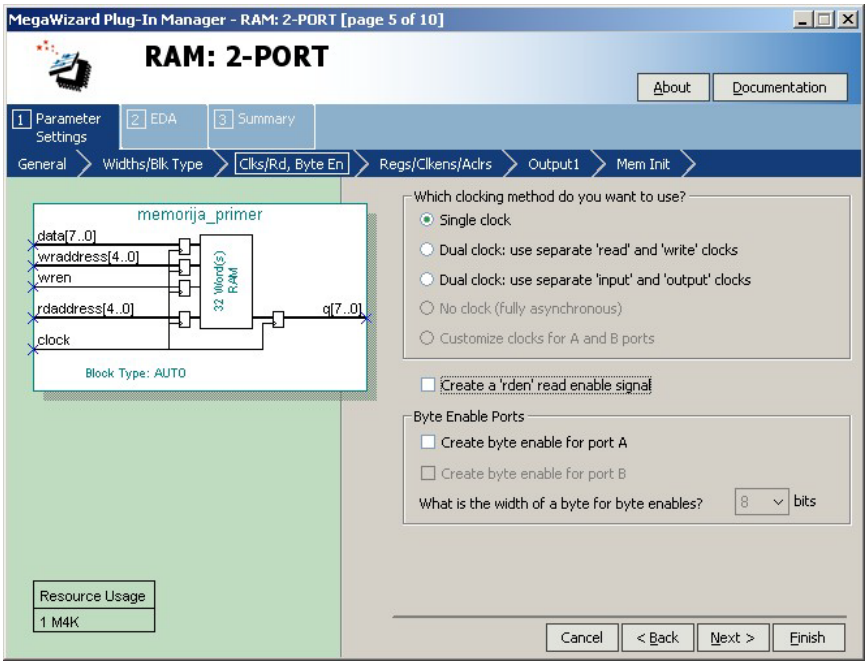
Prvi prozor u podešavanju daje prva podešavanja. Prvo podešavanje se odnosi na familiju čipa i u primeru je ostavljeno difolt podešavanje koje definiše da se uzme familija čipa koja je izabrana u projektu. Drugo podešavanje definiše tip dvoportne RAM memorije i selektovaćemo opciju gde postoji jedan port za čitanje i jedan port za upis podataka. Poslednje podešavanje definiše posmatranje veličine memorije - u vidu broja reči ili broja bita. Selektovaćemo opciju da se veličina memorije posmatra u vidu broja reči.

Sledeći prozor u podešavanju je prikazan na slici 3.2.14.4. Ovaj prozor omogućava podešavanja širine magistrala podataka, kao i broja reči memorije. Takođe omogućava opciju izbora tipa memorijskog bloka koji će se koristiti za smeštanje instanci kreiranog primerka dvoportne memorije. **Auto** opcija podrazumeva da će kompajler sam odlučiti u koji tip memorijskog bloka će smestiti instancu kreiranog primerka dvoportne memorije. U prikazu

simbola memorije sa leve strane možemo videti parametre memorije poput širine magistrale podataka ili adresne magistrale. Promenom parametara koji utiču na ove širine, promeniće se vrednosti i na simbolu memorije. U donjem levom uglu je uvek prikazana procena broja memorijskih blokova FPGA čipa koje bi instanca kreiranog primerka dvoportne memorije zauzela. Ostavimo sva podešavanja na difolt vrednostima i krenimo dalje u podešavanjima.



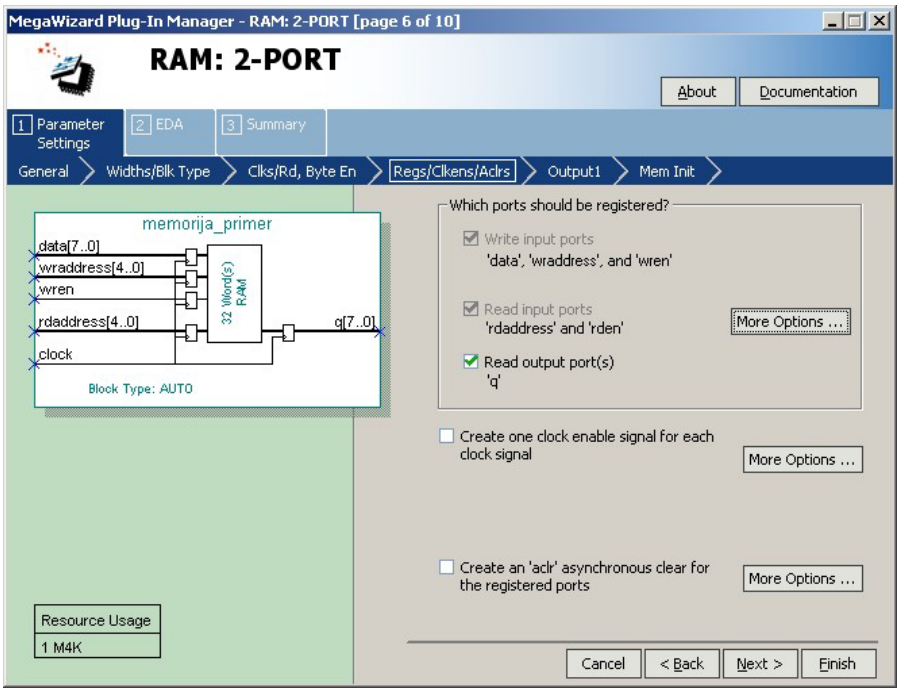
Slika 3.2.14.4. – Drugi po redu prozor za podešavanje dvoportne RAM memorije



Slika 3.2.14.5. – Treći po redu prozor za podešavanje dvoportne RAM memorije

Na slici 3.2.14.5 je prikazan treći po redu prozor za podešavanje dvoportne memorije. Ovaj prozor omogućava selekciju takt signala dvoportne memorije. Difolt podešavanje je i da port za upis i port za čitanje rade na istom taktu, ali moguće je definisati i da oni rade na

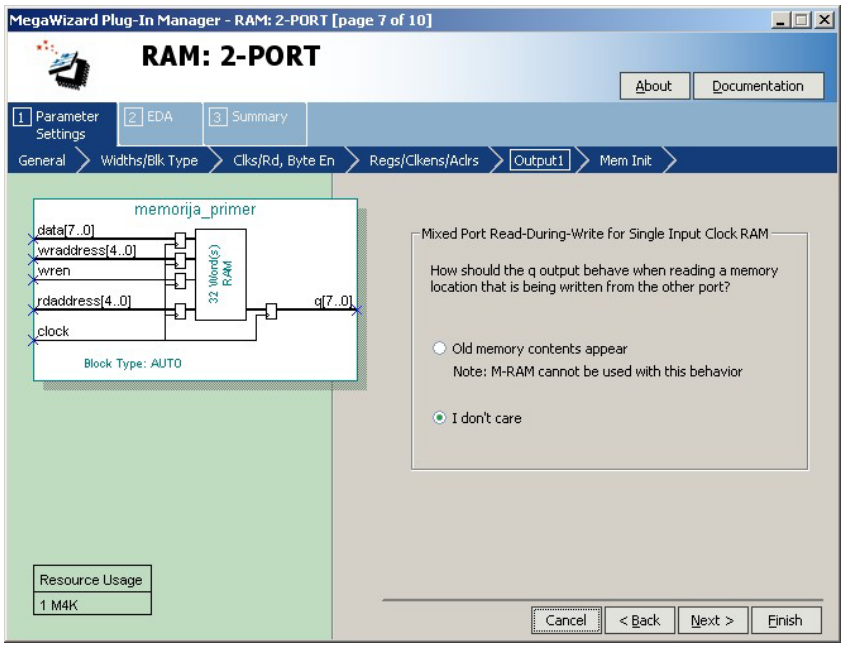
različitim taktovima. Moguće je dodati signal dozvole za čitanje. Kada nema ovog signala na izlazu se uvek nalazi podatak sa lokacije na koju ukazuje vrednost na adresnoj magistrali za čitanje. Kada se signal dozvole za čitanje koristi onda će na izlazu biti podaci na koje ukazuje adresna magistrala za čitanje samo kad je signal dozvole za čitanje aktivan. Takođe je moguće kreirati tzv. signale dozvole na nivou bajta koji predstavljaju maske za upis podataka u memoriju, upisaće se samo oni bajtovi magistrale podataka kod kojih je signal dozvole aktivan. Ostavimo i u ovom prozoru difolt podešavanja i krenimo na sledeći prozor u podešavanjima.



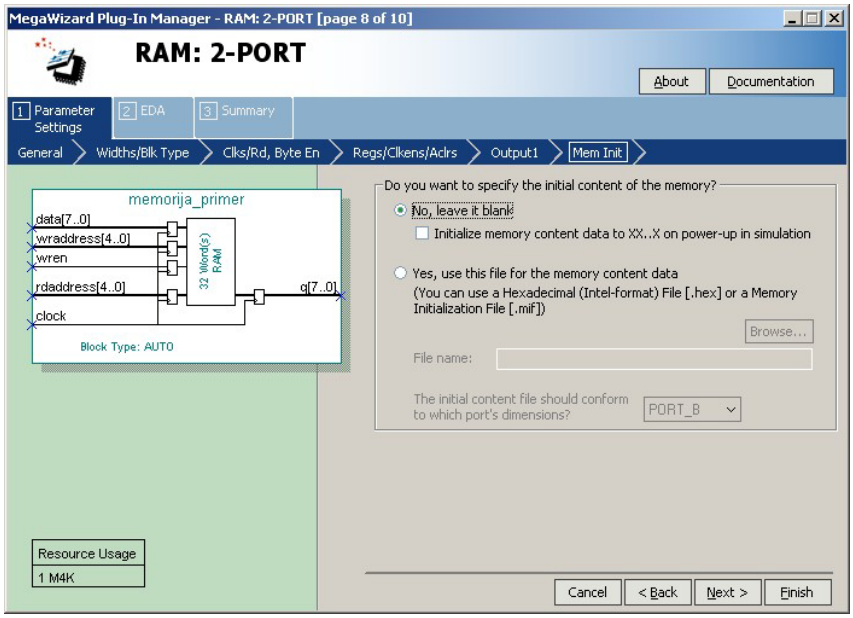
Slika 3.2.14.6. – Četvrti po redu prozor za podešavanje dvoportne RAM memorije

Na slici 3.2.14.6 su prikazana podešavanja za upotrebu registara u kreiranom primerku memorije, između ostalih i registri na izlazu iz memorije. Ukoliko se koriste registri na izlazu, pročitani podatak će biti zakašnjen za jedan period takta, ali ukupna maksimalna frekvencija dizajna će potencijalno biti veća. Takođe je moguće podesiti asinhroni signal reseta za sve registre koji se koriste u memoriji, a takođe je moguće dodati i signale dozvole za sve taktove koji se dovode do memorije. Ostavimo i na ovom prozoru difolt podešavanja i krenimo dalje u podešavanjima.

Prozor prikazan na slici 3.2.14.7 definiše ponašanje memorije kada se istovremeno pristupa nekoj lokaciji i za čitanje i za upis. Možemo izabrati opciju da je svejedno šta će se pročitati ili opciju da se pročita stari sadržaj lokacije. Ostavimo i na ovom prozoru difolt podešavanja i krenimo dalje u podešavanjima.

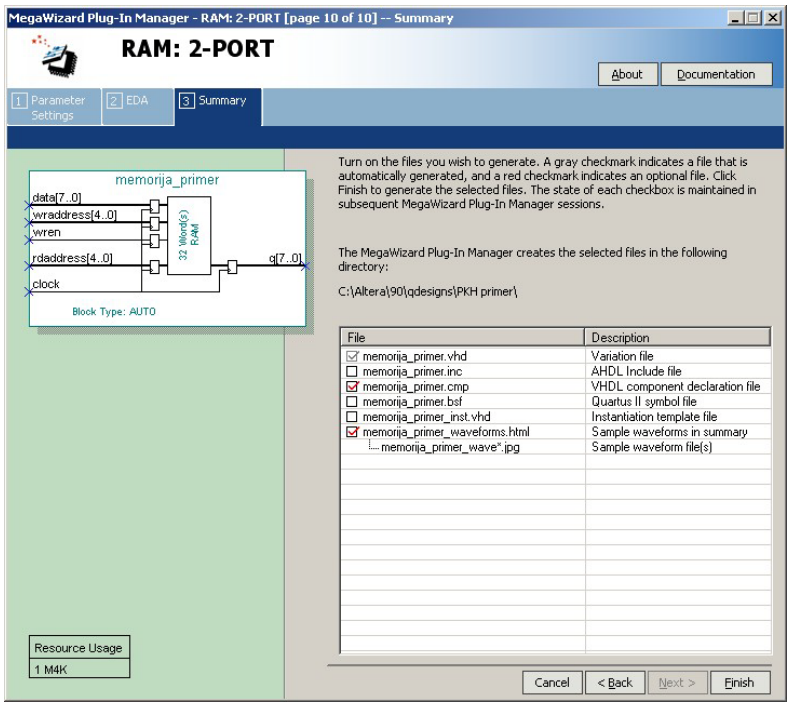


Slika 3.2.14.7. – Peti po redu prozor za podešavanje dvoportne RAM memorije



Slika 3.2.14.8. – Šesti po redu prozor za podešavanje dvoportne RAM memorije

Prozor prikazan na slici 3.2.14.8 definiše inicijalni sadržaj memorije. Možemo staviti prazan sadržaj ili učitati inicijalni sadržaj iz nekog fajla. Kreiranje inicijalnog sadržaja memorije će biti prikazano u sledećoj sekciji. Sledeći prozor u podešavanjima samo navodi koja je biblioteka neophodna u simuliranju dizajna koji koristi kreirani primerak dvoportne RAM memorije. Poslednji prozor, prikazan na slici 3.2.14.9, daje listu izbora fajlova koji će biti kreirani prilikom formiranja primerka dvoportne RAM memorije.



Slika 3.2.14.9. – Poslednji prozor za podešavanje dvoportne RAM memorije

Nakon klika na dugme **Finish** biće kreiran primerak izabrane megafunkcije, u našem primeru to je bila dvoportna RAM memorija. Po difoltu se kreirani primerak megafunkcije pridodaje projektu. Sadržaj kreiranog VHDL fajla koji sadrži opis kreiranog primerka dvoportne memorije je sledeći:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY altera_mf;
USE altera_mf.all;

ENTITY memorija_primer IS
	PORT
	(
		clock		: IN STD_LOGIC ;
		data		: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
		rdaddress	: IN STD_LOGIC_VECTOR (4 DOWNTO 0);
		wraddress		: IN STD_LOGIC_VECTOR (4 DOWNTO 0);
		wren		: IN STD_LOGIC  := '1';
		q		: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
	);
END memorija_primer;


ARCHITECTURE SYN OF memorija_primer IS

	SIGNAL sub_wire0: STD_LOGIC_VECTOR (7 DOWNTO 0);



	COMPONENT altsyncram
	GENERIC (
		address_reg_b		: STRING;
```

```
        clock_enable_input_a                    : STRING;
        clock_enable_input_b                    : STRING;
        clock_enable_output_a                   : STRING;
        clock_enable_output_b                   : STRING;
        intended_device_family                  : STRING;
        lpm_type            : STRING;
        numwords_a                  : NATURAL;
        numwords_b                  : NATURAL;
        operation_mode              : STRING;
        outdata_aclr_b              : STRING;
        outdata_reg_b               : STRING;
        power_up_uninitialized              : STRING;
        read_during_write_mode_mixed_ports          : STRING;
        widthad_a                   : NATURAL;
        widthad_b                   : NATURAL;
        width_a             : NATURAL;
        width_b             : NATURAL;
        width_byteena_a             : NATURAL
    );
    PORT (
            wren_a    : IN STD_LOGIC ;
            clock0    : IN STD_LOGIC ;
            address_a: IN STD_LOGIC_VECTOR (4 DOWNTO 0);
            address_b: IN STD_LOGIC_VECTOR (4 DOWNTO 0);
            q_b       : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
            data_a    : IN STD_LOGIC_VECTOR (7 DOWNTO 0)
    );
    END COMPONENT;

BEGIN
    q   <= sub_wire0(7 DOWNTO 0);

    altsyncram_component : altsyncram
    GENERIC MAP (
        address_reg_b => "CLOCK0",
        clock_enable_input_a => "BYPASS",
        clock_enable_input_b => "BYPASS",
        clock_enable_output_a => "BYPASS",
        clock_enable_output_b => "BYPASS",
        intended_device_family => "Cyclone II",
        lpm_type => "altsyncram",
        numwords_a => 32,
        numwords_b => 32,
        operation_mode => "DUAL_PORT",
        outdata_aclr_b => "NONE",
        outdata_reg_b => "CLOCK0",
        power_up_uninitialized => "FALSE",
        read_during_write_mode_mixed_ports => "DONT_CARE",
        widthad_a => 5,
        widthad_b => 5,
        width_a => 8,
        width_b => 8,
        width_byteena_a => 1
    )
    PORT MAP (
        wren_a => wren,
        clock0 => clock,
        address_a => wraddress,
        address_b => rdaddress,
        data_a => data,
```

```
        q_b => sub_wire0
);
END SYN;
```

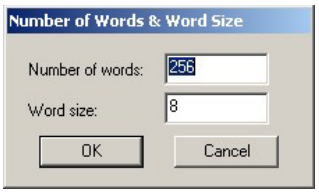
Ovu dvoportnu memoriju u vidu komponente možemo da koristimo u ostalim VHDL fajlovima našeg projekta. Da ne bude zabune, možemo kreirati i više instanci ove komponente koja predstavlja dvoportnu RAM memoriju, odnosno ovu komponentu tretiramo kao i sve druge komponente koje kreiramo u našem dizajnu. Na veoma sličan način se formiraju i primerci drugih megafunkcija. Takođe možemo formirati i više različitih primeraka jedne te iste megafunkcije. Ako želimo da promenimo parametre kreirane funkcije, koristili bi opciju modifikacije postojećeg primerka megafunkcije u prozoru sa slike 3.2.14.1.

Na kraju ponovimo još jednom najvažnije napomene. Megafunkcije su veoma korisne za brzo kreiranje delova dizajna koji su pokriveni megafunkcijama. Megafunkcije omogućavaju podešavanje svojih parametara tako da se lako mogu prilagoditi našim potrebama. Pri prvoj upotrebi neke megafunkcije je poželjno pročitati njenu dokumentaciju radi lakšeg razumevanja kako same megafunkcije, tako i njenih podešavanja i upotrebe.

3.2.15. Kreiranje inicijalnog sadržaja internih memorije

Nekada je zgodno kreirati sadržaj internih memorija koje se koriste u dizajnu, pri čemu je to obavezno ako su u pitanju ROM memorije. Quartus podržava dva formata sa definisanje sadržaja memorije mif (*Memory Initialization File*) format i hex (*Hexadecimal*) format.

Da bismo kreirali novi mif fajl iz glavnog menija pokrećemo opciju **File->New**, i u okviru prozora za izbor tipa fajla, koji želimo da kreiramo, biramo opciju **Memory Files-> Memory Initialization File**. Nakon definisanja našeg izbora, da želimo da kreiramo novi mif fajl otvara se prozor za definisanje veličine memorije (slika 3.2.15.1). Veličina memorije se definiše u vidu dva parametra: broja reči u memoriji, tj. broja lokacija u memoriji i u vidu širine jedne reči, odnosno lokacije.

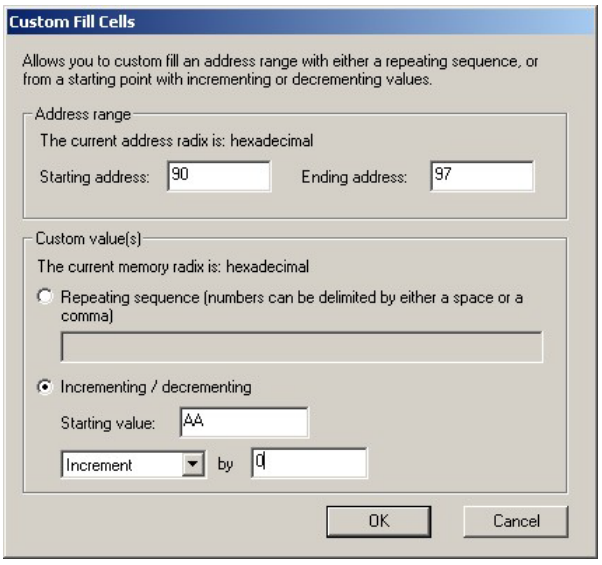


Slika 3.2.15.1. – Definisanje veličine memorije za koju kreiramo mif fajl

Nakon definisanja veličine memorije otvara se editor sadržaja memorije prikazan na slici 3.2.15.2. Editor prikazuje sve lokacije i njihove trenutno definisane sadržaje. Da bi se promenio sadržaj lokacije dovoljno je selektovati lokaciju i uneti novi sadržaj. Postoje i opcije za brz unos sadržaja većeg broja lokacija. Može se selektovati red ćelije, i izborom opcije **Edit->Fill Cells with 0's** ili **Edit->Fill Cells with 1's** iz glavnog menija ćelije tog reda će biti popunjene sa nulama, odnosno jedinicama. Ukoliko se odabere opcija **Edit->Custom Fill Cells** iz glavnog menija, otvara se prozor prikazan na slici 3.2.15.3 koji omogućava unos opsega lokacija u koje želimo da unesemo nov sadržaj i sam sadržaj tih lokacija. Sadržaj lokacija može da se unese u vidu ponavljajuće sekvence sadržaja ili u vidu inkrementirajućeg/dekrementirajućeg sadržaja. U slučaju da izaberemo opciju sa inkrementirajućim/dekrementirajućim sadržajem, pri čemu korak inkrementa/dekrementa postavimo na 0, tada ćemo upisati isti sadržaj u sve lokacije iz zadatog opsega.

Addr	+0	+1	+2	+3	+4	+5	+6	+7
00	00	00	00	00	00	00	00	00
08	00	00	00	00	00	00	00	00
10	00	00	00	00	00	00	00	00
18	00	00	00	00	00	00	00	00
20	00	00	00	00	00	00	00	00
28	00	00	00	00	00	00	00	00
30	00	00	00	00	00	00	00	00
38	00	00	00	00	00	00	00	00
40	00	00	00	00	00	00	00	00
48	00	00	00	00	00	00	00	00
50	00	00	00	00	00	00	00	00
58	00	00	00	00	00	00	00	00
60	00	00	00	00	00	00	00	00
68	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00
78	FF	FF	FF	FF	FF	FF	FF	FF
80	00	00	00	00	00	00	00	00
88	00	00	00	00	00	00	00	00
90	00	00	00	00	00	00	00	00
98	00	00	00	00	00	00	00	00
a0	AA	AA	AA	AA	AA	AA	AA	AA
a8	00	00	00	00	00	00	00	00
b0	00	00	00	00	00	00	00	00
b8	00	00	00	00	00	00	00	00
c0	00	BB	00	00	00	00	00	00
c8	00	00	00	00	00	00	00	00
d0	00	00	00	00	00	00	00	00
d8	00	00	00	00	00	00	00	00
e0	00	01	02	03	04	05	06	07
e8	00	00	00	00	00	00	00	00
f0	00	00	00	00	00	00	00	00
f8	00	00	00	00	00	00	00	00

Slika 3.2.15.2. – Editor sadržaja memorije



Slika 3.2.15.3. – Prozor za definisanje popunjavanja opsega lokacija

Ako želimo da promenimo izgled tabele sa sadržajem lokacija, treba da koristimo opcije iz **View** grupe u glavnom meniju. **Cells per Row** opcija daje mogućnost definisanja broja lokacija po jednom redu tabele. **Address Radix** opcija daje mogućnost odabira formata prikaza adresa lokacija, pri čemu je heksadecimalni format difolt vrednost. **Memory Radix** opcija daje mogućnost odabira formata prikaza sadržaja lokacija, pri čemu je heksadecimalni format difolt vrednost. Ukoliko želimo da promenimo veličinu memorije, možemo pokrenuti **Edit->Memory Size Wizard** opciju iz glavnog menija.

Formiranje novog hex fajla se odvija na potpuno identičan način kao i formiranje novog mif fajla. Jedina razlika je u formatu u kom će biti zapisan sadržaj memorije u sam fajl. Oba fajla se mogu otvoriti u bilo kom običnom tekst editoru i modifikovati ukoliko je tako lakše podesiti željeni sadržaj memorije.

Tako, na primer, sadržaj mif fajla za sadržaj memorije prikazan na slici 3.2.15.2 izgleda:

```
WIDTH=8;
DEPTH=256;

ADDRESS_RADIX=HEX;
DATA_RADIX=HEX;

CONTENT BEGIN
        [000..077]  :  00;
        [078..07F]  :  FF;
        [080..09F]  :  00;
        [0A0..0A7]  :  AA;
        [0A8..0C0]  :  00;
        0C1  :  BB;
        [0C2..0E0]  :  00;
        0E1  :  01;
        0E2  :  02;
        0E3  :  03;
        0E4  :  04;
        0E5  :  05;
        0E6  :  06;
        0E7  :  07;
        [0E8..0FF]  :  00;
END;
```

Kao što možemo videti postoje definicije za veličinu memorije i za format prikaza adresa lokacija i sadržaja lokacija. Sadržaj lokacija se prikazuje ili u vidu opsega sukcesivnih lokacija koje imaju isti sadržaj ili u vidu pojedinačne lokacije sa njenim sadržajem.

Sadržaj hex fajla je nešto duži jer se svaka lokacija ponaosob navodi i deo tog fajla koji definiše sadržaj lokacija u opsegu e0 do ef je:

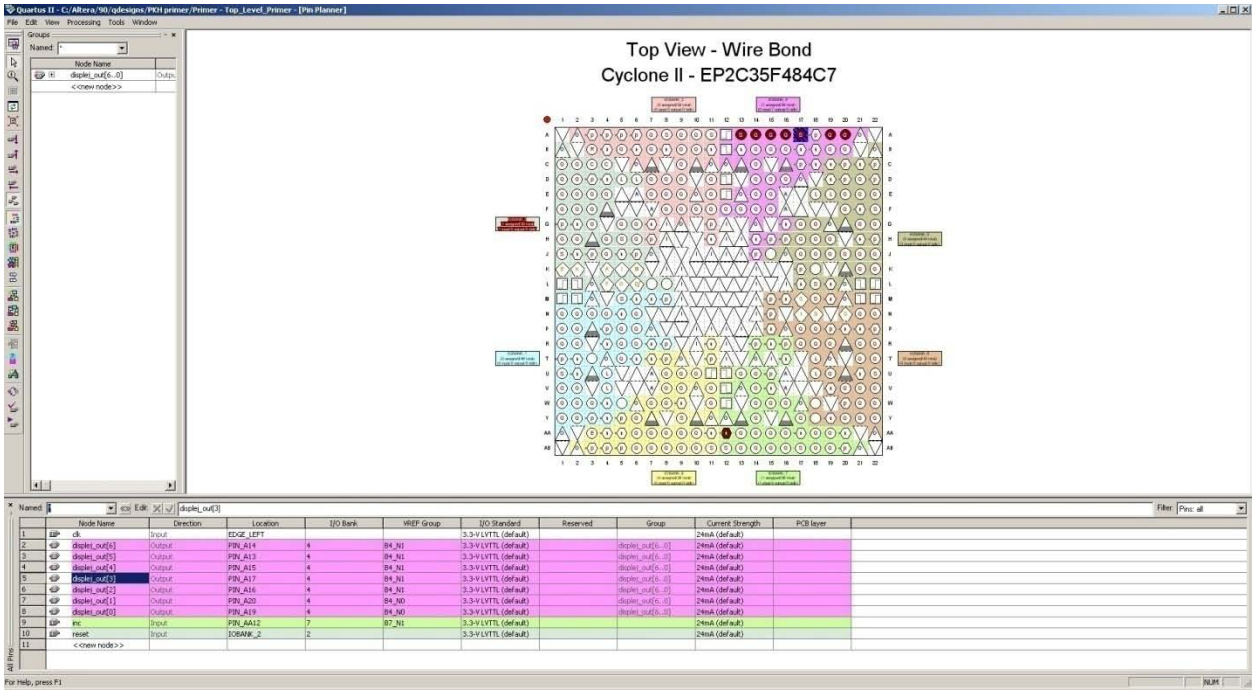
```
:0100E000001F
:0100E100011D
:0100E200021B
:0100E3000319
:0100E4000417
:0100E5000515
:0100E6000613
:0100E7000711
:0100E8000017
:0100E9000016
:0100EA000015
:0100EB000014
:0100EC000013
:0100ED000012
:0100EE000011
:0100EF000010
```

U hex fajlu se navode samo lokacije i njihovi sadržaji, tj. nema definicija veličine memorije kao u slučaju mif fajla. Otuda se pri otvaranju postojećeg hex fajla u editoru, pre samog prikaza fajla u editoru, zadaje pitanje širine lokacije, odnosno reči. Takođe, hex fajl se uvek otvara sa heksadecimalnim formatom prikaza lokacija i sadržaja lokacija. Napomenimo još da je ovaj fajl definisan u skladu sa Intelovim hex formatom. Kod njega je problem što se koriste i sume za proveru parnosti (*checksum*) za sadržaj svakog zapisa, pa otuda nije u potpunosti jednostavno ručno menjati sadržaj ovih lokacija u tekst editoru kao u slučaju mif fajla.

3.2.16. Dodela pinova

Kada se kreira dizajn treba povezati portove top level entiteta na pinove FPGA čipa, odnosno treba dodeliti pinove portovima top level entiteta. Ako se ne uradi dodela pinova, kompajler će sam da rasporedi portove top level entiteta po pinovima FPGA čipa u cilju

postizanja optimalnih performansi. Ovo nekad zna da bude zgodna opcija ukoliko pravimo raspored čipova na štampanoj ploči gde će FPGA čip biti dominantan i određivati raspored ostalih čipova. Otuda u zavisnosti kako kompajler rasporedi pinove na portove top-level entiteta, formiraće se raspored čipova na štampanoj ploči koji će omogućavati najbolje performanse hardvera. S druge strane, možemo svim portovima top-level entiteta unapred dodeliti pinove FPGA čipa i time uneti dodatno ograničenje u procesu kompajliranja jer portovi top-level entiteta imaju fiksirane pozicije na odgovarajućim pinovima, što potencijalno može dovesti do nešto nižih performansi dizajna u FPGA čipu, ali nam omogućava da unapred planiramo raspored čipova na štampanoj ploči. Postoji i mogućnost koja predstavlja sredinu između ove dve krajnje opcije, a to je da delu portova top level entiteta dodelimo pinove, a delu ne, već pustimo da to uradi kompajler.



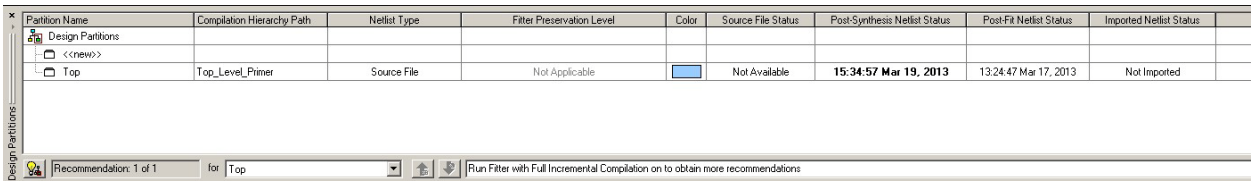
Slika 3.2.16.1. – Prozor za dodelu pinova

Prozor za dodelu pinova (slika 3.2.16.1) se otvara tako što se selektuje opcija **Assignments->Pins** iz glavnog menija. Kao što vidimo sa slike 3.2.16.1, u centralnom delu prozora je grafički prikazan raspored pinova na FPGA čipu. U donjem delu prozora su izlistani portovi top level entiteta i u okviru njega ćemo dodeljivati pinove. U delu za dodelu pinova se nalazi kolona **Location** u okviru koje biramo lokaciju pina koji dodeljujemo odgovarajućem portu top level entiteta. Naravno, u slučaju vektora (magistrala), svakom indeksu vektora će biti dodeljen jedan pin. U okviru ove kolone možemo izabrati konkretan pin, ili možemo samo definisati grupu pinova koji dolaze u obzir za dotični port, a da kompajler sam onda izabere koji će od pinova iz selektovane grupe da izabere. Grupa pinova može biti definisana u vidu banke FPGA čipa ili u vidu odgovarajuće ivice FPGA čipa (donja, gornja, leva i desna ivica). Selektovanjem nekog porta top level entiteta u listi na dnu prozora, biće označena pozicija dodeljenog pina na prikazu FPGA čipa u centralnom delu prozora, naravno pod uslovom da je dotičnom portu već izvršena dodela nekog pina ili grupe pinova. U koloni **I/O Standard** se može izabrati standard signala koji će biti razmenjivan preko dotičnog pina. U koloni **Current Strength** se može izabrati jačina struje koja će ići preko dotičnog pina. Naravno, ove dve kolone

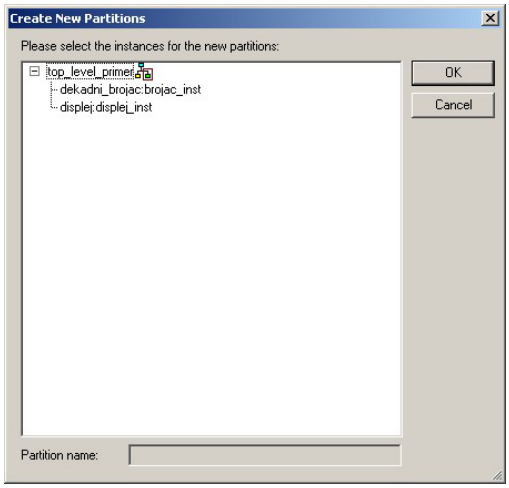
treba podešavati u skladu sa tim na šta su dotični pinovi povezani i koji standardi su podržani na dotičnim pinovima. Važno je napomenuti da prilikom projektovanja štampane ploče treba pročitati specifikacije korišćenog FPGA čipa da bi se utvrdilo koji pinovi se za šta koriste. Većina pinova se može konfigurisati u željenom smeru (ulazni, izlazni ili bidirekcioni), ali neki imaju mogućnost samo jednog smera, na primer samo ulaznog smera. Takođe, za neke funkcionalnosti se mogu koristiti samo neki pinovi. Na primer, za brze multigigabitske primopredajnike su rezervisani samo neki pinovi, ili za signale takta je takođe preporučljivo koristiti samo određene pinove FPGA čipa. U slučaju razvojnih ploča koje sadrže hardversko razvojno okruženje za FPGA implementacije, treba pročitati specifikacije razvojne ploče koje govore sa čim je povezan svaki od pinova FPGA čipa. Razvojne ploče su idealne za ranu fazu projektovanja dizajna jer nude hardversko test okruženje gde je FPGA čip povezan sa mnogim komponentama (eternet portom, DRAM memorijom, SRAM memorijom, USB portom...) i na jednostavan način možemo razviti željene funkcionalnosti koje uključuju i komunikaciju sa drugim čipovima. Nakon što razvijemo dizajn koristeći razvojnu ploču, možemo pristupiti projektovanju konačne štampane ploče koja će sadržati FPGA čip sa projektovanim dizajnom i sve ostale neophodne čipove i koja će imati željenu funkcionalnost.

3.2.17. Kreiranje particija

Particije mogu biti pogodne kada želimo da izdelimo dizajn na ’nezavisne’ celine. Svaka od celina se nezavisno posmatra i nikakva optimizacija između nezavisnih celina neće biti urađena (čak i ako bi potencijalno bila moguća). Ovo je pogodna opcija kada često menjamo jedan deo složenog dizajna, a želimo da skratimo vreme kompajliranja procesa analize i sinteze. Upotrebom inkrementalnog kompajliranja u procesu analize i sinteze će u što većoj meri biti korišćeni rezultati prethodnog kompajliranja, i cilj je da se rekompajlira samo izmenjeni deo dizajna, odnosno izmenjena particija čime bi se skratilo vreme kompajliranja.

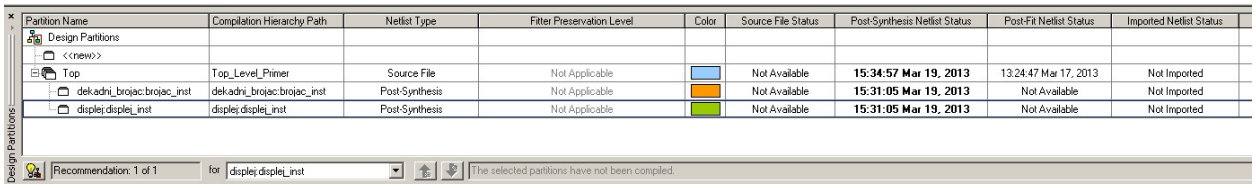


Slika 3.2.17.1. – Prozor za kreiranje particija



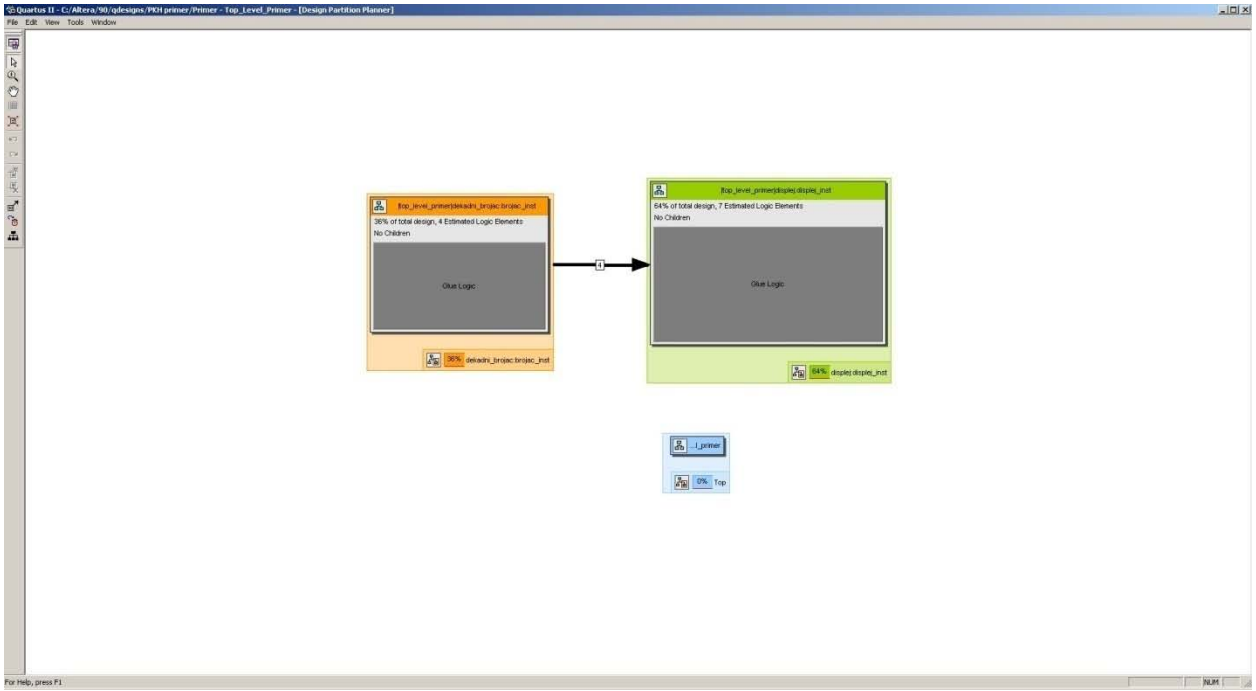
Slika 3.2.17.2. – Prozor za dodavanje entiteta kao particija

Particije se kreiraju selekcijom opcije **Assignments->Design Partitions Window** čime se otvara prozor u dnu glavnog prozora aplikacije (slika 3.2.17.1). Da bi smo kreirali novu particiju treba uraditi dvostruki klik na polje **<<new>>**, čime se otvara prozor za dodavanje nove particije (slika 3.2.17.2). Prozor prikazuje hijerarhijsku strukturu projekta i treba izabrati odgovarajući entitet ili entitete. Sa pritisnutim CTRL tasterom se može izvršiti selekcija više entiteta odjednom. Klikom na dugme **OK** se potvrđuje selekcija entiteta koje želimo da definišemo kao particije. Na slici 3.2.17.3 je prikazan izgled prozora za kreiranje particija nakon što smo kao particije definisali dekadni brojač i displej logiku.



Slika 3.2.17.3. – Prozor za kreiranje particija sa dodatim particijama

Sada kada se pokrene proces analize i sinteze u **Summary** izveštaju procesa analize i sinteze neće biti prikazana procena potrošnje internih resursa čipa, jer su particije nezavisno kompajlirane. Tek sa pokretanjem podopcije **Partition Merge** u okviru opcije **Analysis & Synthesis** toka projekta će ove dve particije biti spojene u jednu celinu i prikazaće se procena potrošnje internih resursa. Treba primetiti da, ako na primer izmenimo samo VHDL kod displej logike, u procesu analize i sinteze neće biti rekompajliran dekadni brojač. Pokretanjem podopcije **Design Partition Planner** u okviru opcije **Analysis & Synthesis** toka projekta otvara se prozor sa pregledom postojećih particija i njihovih međusobnih relacija - slika 3.2.17.4.



Slika 3.2.17.4. – Prozor za grafički pregled postojećih particija

3.2.18. Programiranje FPGA čipa

Nakon što smo kreirali dizajn, dodelili pinove, kompajlirali dizajn i dobili konfiguracioni fajl za programiranje FPGA čipa, neophodno je i zaista programirati FPGA čip da bi on izvršavao dizajnirane funkcionalnosti. Kod FPGA čipova se najčešće koristi tehnika programiranja bazirana na SRAM ćelijama, što znači da se konfiguracija FPGA čipa čuva van samog FPGA čipa. FPGA čip se stoga može programirati sa samog računara, a takođe se na štampanoj ploči koja sadrži FPGA čip može postaviti PROM ili fleš memorija iz koje će po dobijanju napajanja ili po završetku reseta, čip preuzeti svoju konfiguraciju. Prva varijanta sa spuštanjem dizajna na FPGA čip sa računara se koristi tokom faze projektovanja dizajna jer tada ima mnogo izmena i modifikacija samog dizajna i tada je mnogo efikasnije programirati FPGA čip sa računara na kome se i radi kompajliranje dizajna. Nakon faze projektovanja, kada se formira konačna verzija štampane ploče sa FPGA čipom koristi se eksterna memorija poput PROM memorije iz koje će se programirati FPGA čip čime štampana ploča postaje sposobna za samostalan rad. U okviru ovih skripti će ukratko biti prikazan metod programiranja FPGA čipa sa host računara. Kao primer će biti korištena Alterina DE2 razvojna ploča koja sadrži FPGA čip Cyclone II EP2C35F672C6.

Tabela 3.2.18.1. – Raspored pinova po portovima dizajna

Port	Pin
clk	N2
reset	N25
inc	N26
displej_out(6)	V13
displej_out(5)	V14
displej_out(4)	AE11
displej_out(3)	AD11
displej_out(2)	AC12
displej_out(1)	AB12
displej_out(0)	AF10

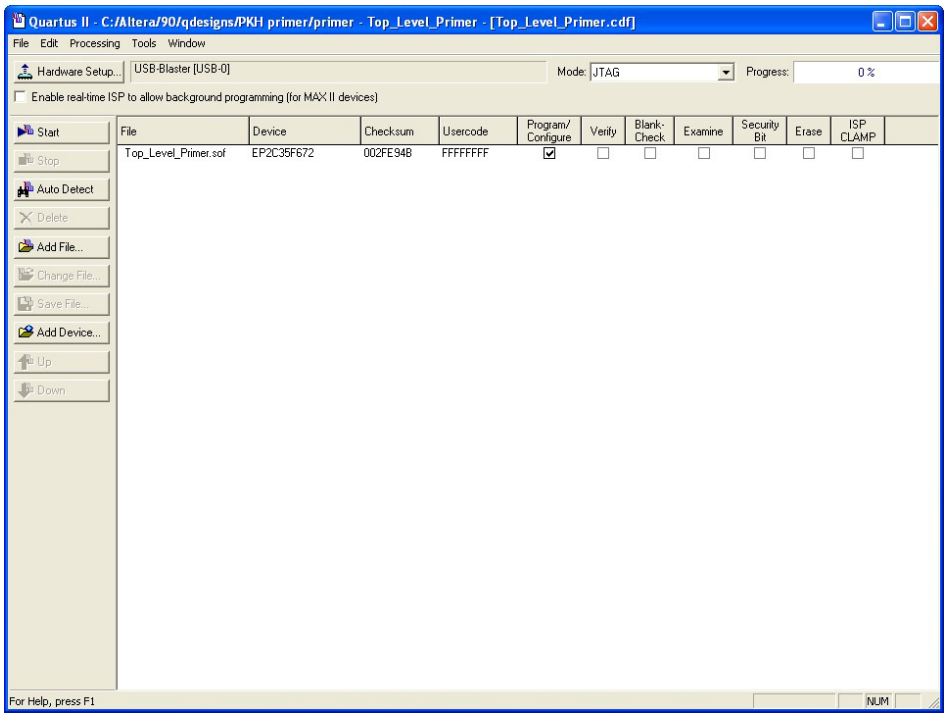
U okviru podešavanja projekta podesimo čip na Cyclone II EP2C35F672C6, i takođe izvršimo dodelu pinova u skladu sa tabelom 3.2.18.1. Signal **clk** smo povezali sa 50MHz izvorom takta, signale **reset** i **inc** smo povezali sa prekidačima, dok smo **displej_out** povezali na led diode koje čine jedan sedmosegmentni displej. Modifikujmo VHDL kod fajla top_level_primer.vhd:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;--neophodno jer radimo sa STD_LOGIC i STD_LOGIC_VECTOR
USE ieee.std_logic_unsigned.all;--neophodno za operaciju sabiranja
--top level entitet koji uvozi u vidu komponenti dekadni brojac i displej logiku
ENTITY top_level_primer IS
PORT
(
	clk:		IN STD_LOGIC;--signal takta
	reset:		IN STD_LOGIC;--asinhroni signal reseta
	inc:		IN STD_LOGIC;--kontrolni signal za dozvolu brojanja
	displej_out:	OUT STD_LOGIC_VECTOR(6 DOWNTO 0)--izlaz koji kontrolise displej
);
END top_level_primer;

ARCHITECTURE shema OF top_level_primer IS
	--deklaracija komponente dekadnog brojaca
```

```
	COMPONENT dekadni_brojac IS
	PORT
	(
		clk:	IN STD_LOGIC;--signal takta
		reset:	IN STD_LOGIC;--asinhroni signal reseta
		inc:	IN STD_LOGIC;--kontrolni signal za dozvolu brojanja
		q:	OUT STD_LOGIC_VECTOR(3 DOWNTO 0)--vrednost brojaca
	);
	END COMPONENT;
	--deklaracije logike za kontrolu displeja
	COMPONENT displej IS
	PORT
	(
		broj:	IN STD_LOGIC_VECTOR(3 DOWNTO 0);--vrednost broja na ulazu
		displej:	OUT STD_LOGIC_VECTOR(6 DOWNTO 0)--vektor ciji svaki indeks utice na jednu diodu displeja
	);
	END COMPONENT;
	--deklaracija internog signala neophodnog za povezivanje izlaza dekadnog brojaca
	-- sa ulazom u displej logiku
	SIGNAL broj_int:STD_LOGIC_VECTOR(3 DOWNTO 0);
	SIGNAL inc_in:STD_LOGIC;--interni signal inkrementa
	SIGNAL brojac:STD_LOGIC_VECTOR(27 DOWNTO 0);--brojac koji koristimo da odredimo period jedne sekunde
	CONSTANT perioda:STD_LOGIC_VECTOR(27 DOWNTO 0):=X"2FAF080";--perioda jedne sekunde
	--X"2FAF080" pomnozen sa 20ns daje 1s
BEGIN
  --proces za generisanje signala impulsa inkrement signala u periodi od 1s
	PROCESS(reset,clk)
	BEGIN
		IF(reset='1')THEN
			inc_in<='0';
			brojac<=(OTHERS=>'0');
		ELSIF(clk'EVENT AND clk='1')THEN
			IF(inc='0')THEN--resetuj brojac
				inc_in<='0';
				brojac<=(OTHERS=>'0');
			ELSE
				IF(brojac=perioda)THEN
					brojac<=(OTHERS=>'0');--reset brojaca
					inc_in<='1';--generisanje jednog impulsa inkrementa za dekadni brojac
				ELSE
					brojac<=brojac+'1';--inkrement brojaca
					inc_in<='0';
				END IF;
			END IF;
		END IF;
	END PROCESS;
	--instanciranje komponente brojaca
	brojac_inst: dekadni_brojac
	PORT MAP
	(
		clk => clk,
		reset => reset,
		inc => inc_in,--vezujemo za interni signal
		q => broj_int
	);
	--instanciranje komponente za kontrolu displeja
	displej_inst: displej
	PORT MAP
	(
		broj => broj_int,
```

```
            displej => displej_out
    );
END shema;
```

Modifikacija služi da se promena brojača odvija na nivou periode od 1s uz upotrebu takta od 50MHz. Modifikacija je neophodna da bi vizuelno mogao da se isprati rad brojača prikazom na sedmosegmentnom displeju. Ova modifikacija VHDL koda je mogla da se uradi i na drugačije načine što se ostavlja čitaocu da pokuša da nađe alternativnu modifikaciju koda koja bi radila istu funkcionalnost. Aktivirajmo kompletno kompajliranje dizajna. Kao rezultat procesa generisanja konfiguracionog fajla FPGA čipa dobijen je .sof fajl koji ćemo koristiti u procesu programiranja FPGA čipa sa host računara. Da bismo programirali FPGA čip na razvojnoj ploči neophodno je povezati ploču sa host računarom. U ovom slučaju se koristi običan USB kabl, ali u zavisnosti od ploča mogu se koristiti i drugi tipovi kablova. Takođe, neophodno je povezati ploču na izvor napajanja i uključiti ploču. Tačan redosled operacija za povezivanje razvojne ploče sa host računarom (povezivanje ploče sa izvorom napajanja i sa host računarom, aktivacija ploče) treba izvršiti na osnovu dokumentacije same razvojne ploče. Nakon ovih operacija pokreće se programer iz Quartus aplikacije. Programer se pokreće izborom opcije **Tools-> Programmer** iz glavnog menija čime se otvara prozor prikazan na slici 3.2.18.1. Difolt podešavanja omogućavaju da se izvrši programiranje FPGA čipa sa host računara. U slučaju da kabl i razvojna ploča nisu automatski detektovani, dugmetom **Hardware Setup** se pokreće meni za ručno podešavanje metode programiranja sa host računara. Klikom na **Start** dugme se pokreće proces programiranja FPGA čipa koji će da rekonfiguriše čip da radi u skladu sa novim dizajnom koji je sadržan u selektovanom .sof fajlu. Sa dugmetom **Add File** se mogu dodati drugi .sof fajlovi, jer se po difoltu otvara .sof fajl čiji naziv je jednak nazivu top level entiteta. Nakon što se spusti dizajn na FPGA čip možemo videti da se na sedmosegmentnom displeju po aktivaciji inkrement signala (uz neaktivan reset signal) prikazuju dekadni brojevi u kružnom redosledu od 0 do 9. Pri tome se prikaz displeja inkrementira na svaku sekundu.



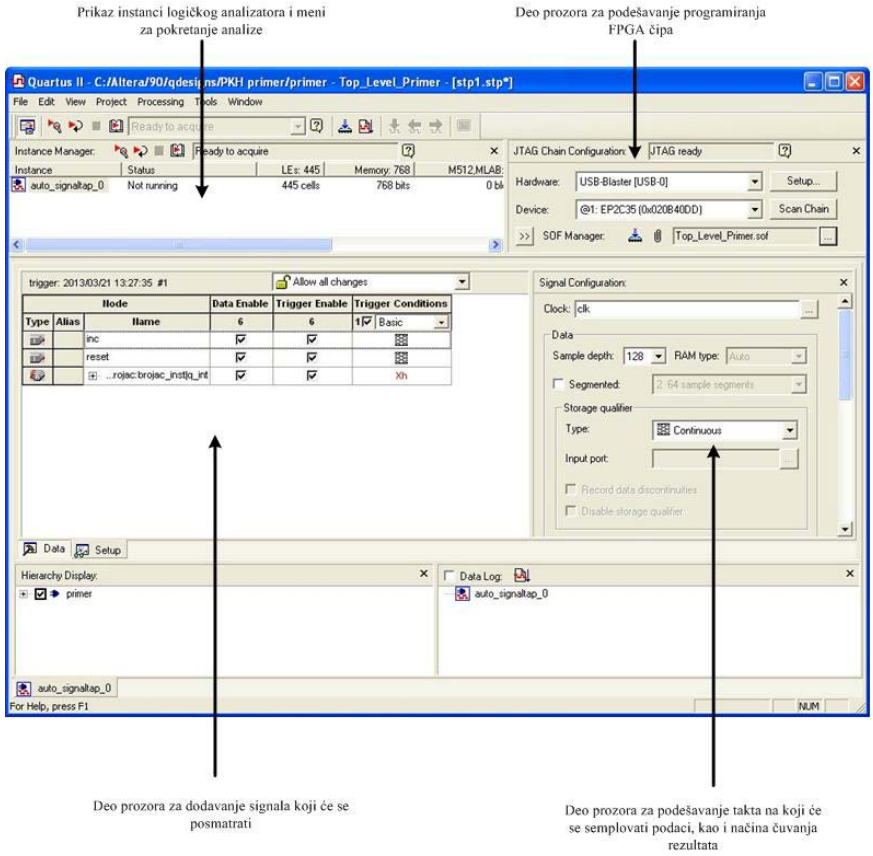
Slika 3.2.18.1. – Prozor za programiranje FPGA čipa sa host računara

3.2.19. Interni logički analizator

Zahvaljujući JTAG interfejsu, koji se koristi i za programiranje FPGA čipova, moguće je koristiti interne logičke analizatore koji omogućavaju posmatranje internih signala u FPGA čipu za vreme rada samog FPGA čipa na štampanoj ploči. Time je omogućeno praćenje realnih dešavanja u samom čipu čime je omogućena efikasna verifikacija i testiranje dizajna u realnim situacijama. Ovo je naročito korisno za testiranje delova dizajna koji su povezani sa eksternim čipovima. Treba samo imati na umu da interni logički analizator zauzima određene resurse na FPGA čipu, tako da po okončanju hardverskog testiranja dizajna uz pomoć internog logičkog analizatora, treba iz finalne verzije dizajna ukloniti interni logički analizator.

Tabela 3.2.19.1. – Raspored pinova po portovima dizajna gde je displej vezan za *expansion header*

Port	Pin
clk	N2
reset	N25
inc	N26
displej_out(6)	J21
displej_out(5)	F23
displej_out(4)	F24
displej_out(3)	E25
displej_out(2)	E26
displej_out(1)	J22
displej_out(0)	D25

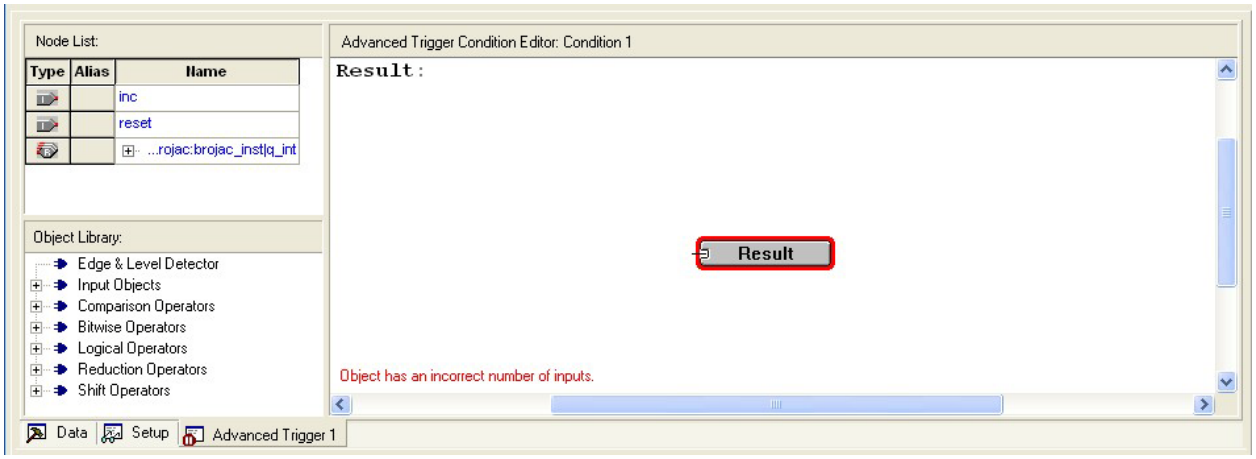


Slika 3.2.19.1. – Prozor za konfigurisanje internog logičkog analizatora

Interni logički analizator se podešava aktiviranjem opcije **Tools->SignalTap II Logic Analyzer** iz glavnog menija. Prozor prikazan na slici 3.2.19.1 se koristi za podešavanje internog logičkog analizatora. Kao primer ćemo koristiti originalni dizajn koji sačinjavaju tri VHDL fajla čija struktura je data u sekciji 3.2.3. Raspored dodeljenih pinova je dat u tabeli 3.2.19.1, pri čemu se opet kao u prethodnoj sekciji koristi Alterina DE2 razvojna ploča. Sada se **displej_out** linije vode na zaglavlje za proširenje (*expansion header*) na koje se mogu priključiti drugi hardverski moduli ili kablovi, a takođe se mogu priključiti i sonde osciloskopa ili eksternog logičkog analizatora.

Po otvaranju prozora je automatski kreirana jedna instanca logičkog analizatora koja je prikazana u delu prozora koji prikazuje sve prisutne instance logičkog analizatora. Za kreiranje nove instance logičkog analizatora treba kliknuti desnim klikom na polje prikaza instanci logičkog analizatora i izabrati opciju **Create Instance**. Isto može da se uradi izborom opcije **Edit->Create Instance** u glavnom meniju.

Kada se selektuje jedna instanca logičkog analizatora, mogu se dodati interni signali koji će se posmatrati. To se postiže dvostrukim klikom u delu prozora za dodavanje signala koji će se posmatrati ili **Edit-> Add Nodes** opcijom iz glavnog menija. Takođe, u glavnom meniju postoji i opcija (**Edit-> Add State Machine Nodes**) za dodavanje konačnih automata dizajna koji se žele posmatrati. **Add Nodes** opcija otvara prozor za nalaženje čvorova (**Node Finder** prozor) identičan onome korišćenom za nalaženje čvorova koje želimo koristiti u simulaciji prikazanom na slici 3.2.6.4. Dodajmo signale **reset**, **inc** i **q_int** kao signale koje želimo da posmatramo. Dodati čvorovi su vidljivi na slici 3.2.19.1. Padajući meni u delu prozora koji prikazuje signale koji se posmatraju nudi dve opcije. Difolt opcija **Allow all changes** omogućava da se svi parametri koji se odnose na podešavanje posmatranja izabranih signala internog logičkog analizatora mogu menjati, dok druga opcija **Allow trigger condition changes only** omogućava samo modifikaciju triger uslova analizatora. Triger u stvari predstavlja okidač na koji će analizator da pokrene akviziciju podataka. Kod svakog signala koji smo dodali postoji mogućnost podešavanja njegovih parametara posmatranja. **Data Enable** opcija definiše da li će signal da se posmatra u logičkom analizatoru ili ne, tj. da li će da se radi akvizicija vrednosti ovog signala ili ne. **Trigger Enable** opcija definiše da li će dotični signal da se koristi kao triger za startovanje akvizicije podataka logičkog analizatora ili ne. Ukoliko je izabrana **Trigger Enable** opcija, onda **Trigger Conditions** opcija omogućava definisanje trigera. Po difoltu su ponuđena osnovna (**Basic**) definisanja trigera, poput uzlazne ivice signala, silazne ivice signala, bilo koje ivice signala, nivo signala jednak nuli, nivo signala jednak jedinici i *don't care* vrednost. Navedene mogućnosti osnovnog definisanja trigera se nalaze u meniju koji se otvara na desni klik miša na polje gde je prikazano trenutno podešavanje trigera dotičnog signala. Izborom opcije **Advanced** dobija se novi potprozor sa sopstvenim menijem (slika 3.2.19.2) gde je omogućeno složenije podešavanje trigera i gde je dozvoljena upotreba na primer logičkih funkcija (I, ILI...), ali i drugih funkcija kao što se vidi iz menija sa leve strane potprozora. Ovom potprozoru odgovara tab **Advanced Trigger *i***, gde *i* definiše redni broj naprednog trigera koji smo definisali. Inače u dnu prozora za dodavanje signala koje ćemo posmatrati se uvek nalaze tabovi **Data** i **Setup**. **Setup** tab odgovara podešavanjima parametara posmatranja odabranih signala i njega smo u stvari prethodno opisivali, a **Data** tab se koristi za prikaz izgleda signala nakon izvršene akvizicije.



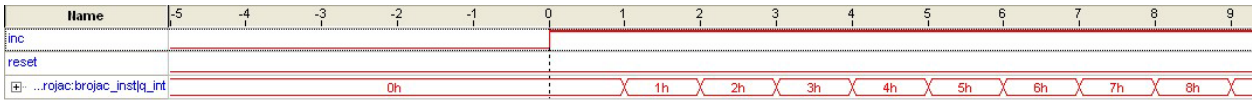
Slika 3.2.19.2. – Prozor za napredno konfigurisanje trigera

Takođe je važno definisati i opšte parametre posmatranja, kao na primer koliko semplova svakog signala želimo da uzmemo po jednoj akviziciji. Logično, što veći broj semplova želimo, interni logički analizator će zauzeti veću količinu internih resursa FPGA čipa. Takođe je bitno definisati i signal takta na kojim će da se uzimaju semplovi signala koje posmatramo. Ova podešavanja se rade u delu prozora za definisanje takta na koji će se uzimati semplovi posmatranih internih signala koji se nalazi desno od dela za dodavanje signala koje želimo da posmatramo. U polje **Clock** unosimo signal takta na koji će se vršiti semplovanje. I ovde se može umesto direktnog kucanja naziva signala koristiti meni za nalaženje signala (**Node Finder** prozor) koji se otvara klikom na dugme sa desne strane polja **Clock**. **Sample Depth** padajući meni omogućava podešavanje broja semplova koji će biti uzet u toku akvizicije podataka internog logičkog analizatora. **Storage Qualifier** meni definiše kriterijume za čuvanje podataka. Difolt kriterijum **Continuous** definiše da se čuvaju podaci prikupljeni na svaku ivicu takta, dok drugi kriterijumi omogućavaju podešavanje dodatnih uslova za čuvanje podataka. U slučaju da nije izabran difolt kriterijum pojaviće se **Storage Enable** i **Storage Qualifier** opcije kod dela koji sadrži listu odabranih signala za posmatranje koja omogućava da se za svaki signal izabere izabrani **Storage Qualifier** kriterijum ili difolt **Storage Qualifier** kriterijum (tada opcija **Storage Qualifier** nije selektovana kod signala) i da li će signal uopšte da se čuva za posmatranje (to definiše **Storage Enable** opcija).

Sva podešavanja koja su navedena se odnose na posmatranje rada dizajna u čipu i definisanje triger signala za određivanje situacija koje želimo da posmatramo, odnosno žargonski rečeno da uhvatimo. Međutim, nekada je veoma značajno posmatrati ponašanje dizajna neposredno nakon završetka programiranja FPGA čipa, odnosno odmah na samom početku rada čipa. Da bi se to radilo neophodno je definisati tzv. *Power-Up Trigger*, koji omogućava da se trigeri aktiviraju odmah nakon početka rada FPGA čipa. Da bi se dodao *Power-Up Trigger*, neophodno je kliknuti na instancu logičkog analizatora desnim klikom i u meniju koji se otvori treba izabrati opciju **Enable Power-Up Trigger**. Ovo je moguće i iz glavnog menija izborom opcije **Edit->Enable Power-Up Trigger**, pri čemu je naravno morala biti selektovana instanca internog logičkog analizatora. Ispod instance će se pojaviti **Power-Up Trigger** podinstanca i kada nju selektujemo, mogu se definisati trigeri kao što je već ranije opisano uz naglasak da su mogućnosti podešavanja smanjene i moguća samo u poljima obeležena plavom bojom.

Nakon što smo definisali sva podešavanja potrebno je rekompajlirati projekat da bi se u obzir uzeo i interni logički analizator. U gornjem levom uglu se nalazi deo u kome se definiše

povezivanje host računara sa pločom koja sadrži FPGA čip koji se programira, a takođe se nalazi i meni za izbor .sof fajla koji želimo da spustimo na FPGA čip. Naravno, tu se nalazi i dugme za pokretanje procesa programiranja FPGA čipa. Nakon što spustimo dizajn, neophodno je da pokrenemo rad logičkog analizatora upotrebom opcije **Processing->Run Analysis** iz glavnog menija ili opcije **Processing->Autorun Analysis** takođe iz glavnog menija. Prva opcija pokreće rad analizatora gde on čeka na triger signal da bi izvršio akviziciju i nakon toga se prikazuju podaci i zaustavlja se dalja akvizicija dok je opet sami ne pokrenemo. Druga opcija pokreće rad analizatora gde on čeka na triger signal da bi izvršio akviziciju. Međutim, sada nakon obavljene akvizicije ne dolazi do zaustavljanja rada analizatora, već se on automatski ponovo pokreće i čeka na novi triger signal. Ove dve opcije imaju i svoje ikonice na glavnom prozoru koje omogućuju brz pristup i aktivaciju ovih opcija. Prikaz signala nakon obavljene akvizicije je dat na slici 3.2.19.3. Kao što vidimo sa slike, deo semplova ima negativne vrednosti vremenskih trenutaka što označava da su oni prikupljeni pre aktivacije trigera koja definiše vremenski trenutak 0. U delu za podešavanje odnosa prikazanih semplova i pozicije trigera koji se otvara neposredno ispod dela za podešavanje **Storage Qualifier** opcije se može selektovati da pozicija trigera bude malo posle početka semplovanja (pogodno kada analiziramo događaje nakon trigera), da pozicija trigera bude na sredini (pola semplova pre trigera i pola semplova posle trigera) i da pozicija trigera bude malo pre završetka semplovanja (pogodno kada analiziramo događaje pre trigera). Prva opcija koja je ujedno i prikazana na slici 3.2.19.3 je difolt opcija. Takođe, u tom delu je moguće definisati i broj trigera, a potom je moguće u delu prozora gde se podešavaju trigeri (deo gde se dodaju i signali koji se posmatraju) svaki triger ponaosob podesiti. Napomenimo, da je za kreiranje primera sa slike 3.2.19.3 definisan triger koji je zahtevao da signal reseta bude na neaktivnoj vrednosti (logička nula) i uzlaznu ivicu signala inkrementa.



Slika 3.2.19.3. – Prikaz vrednosti signala nakon izvršene akvizicije podataka

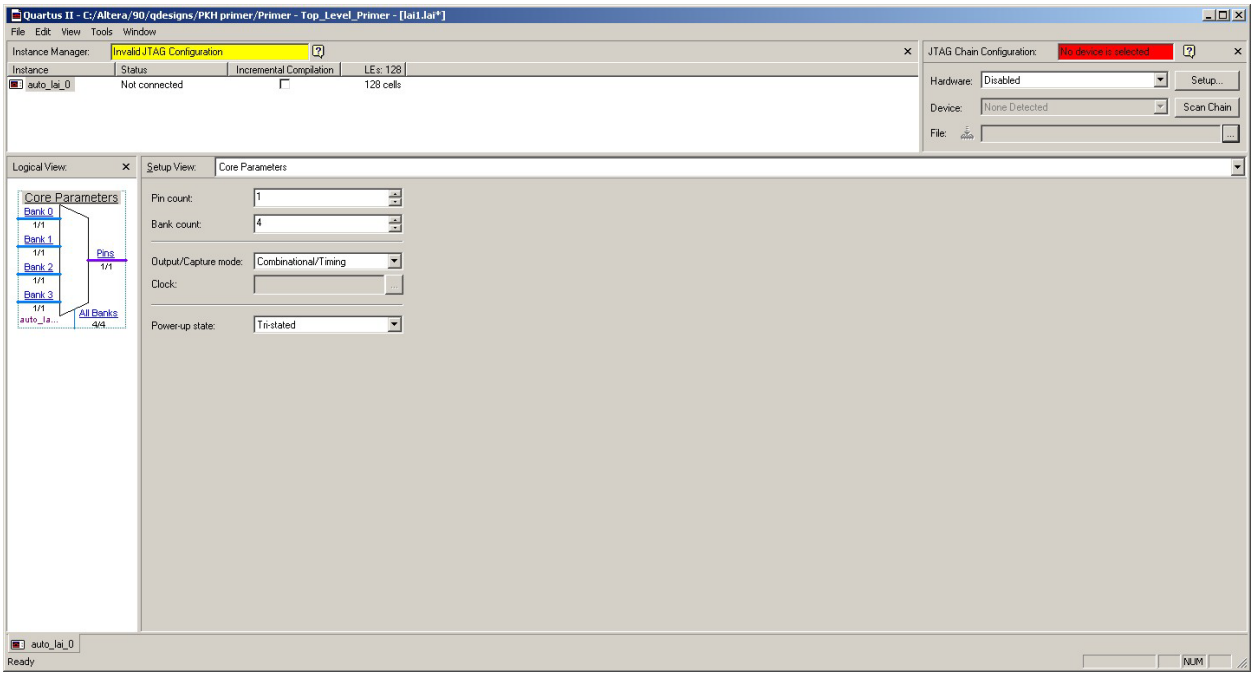
Napomenimo da postoji još jedan način za aktiviranje i upotrebu internog logičkog analizatora, a to je uz pomoć megafunkcije **SignalTap II Logic Analyzer** koja se nalazi u grupi **JTAG-accessible Extensions**. Upotrebom navedene megafunkcije se može generisati VHDL komponenta internog logičkog analizatora koja se potom može instancirati u okviru razvijanog dizajna. Nakon generisanja konfiguracionog fajla za programiranje FPGA čipa i potom programiranja FPGA čipa, može se pristupiti internom logičkom analizatoru aktiviranjem opcije **Tools->SignalTap II Logic Analyzer** iz glavnog menija koju smo već prethodno opisali.

Kao što se može primetiti iz svega izloženog u okviru ove sekcije, interni logički analizator koji nudi Quartus razvojno okruženje je veoma moćan i fleksibilan alat koji nam omogućava da lako podesimo sve parametre koji će nam omogućiti da efikasno posmatramo događaje od interesa i veoma brzo i kvalitetno hardverski testiramo i verifikujemo svoj dizajn.

3.2.20. Eksterni logički analizator

Eksterni logički analizator predstavlja drugo rešenje za posmatranje signala na dizajniranom hardveru. Eksterni logički analizator je poseban merni uređaj koji omogućava posmatranje i analizu digitalnih signala, kao i njihovo memorisanje radi kasnije obrade i analize. Ovi uređaji su skupi, pri čemu cena u značajnoj meri zavisi od njihovog propusnog opsega. Propusni opseg određuje koja je maksimalna brzina signala koju možemo da posmatramo. Što je

veći propusni opseg, možemo da posmatramo sve brže signale, ali je i cena uređaja znatno veća. Pored logičkog analizatora mogu se koristiti i osciloskopi za posmatranje signala, pri čemu se osciloskopi koriste više za posmatranje oblika signala, dok se logički analizatori koriste za posmatranje digitalne vrednosti signala (’0’ ili ’1’). Da bi se moglo pristupiti posmatranim signalima, neophodno je da se oni ’izvuku’ na mesto gde se mogu nakačiti sonde eksternog logičkog analizatora (ili osciloskopa). Tipično, to su izvučene ’iglice’ na koje se mogu nakačiti sonde analizatora, pri čemu iglice mogu biti pojedinačne ili u manjim grupama, ali i u vidu zaglavlja za proširenje (*expansion header*). Uvek je važno da bude izvučena i masa, jer je ona neophodna za određivanje nivoa signala - napon je jednak razlici potencijala, pa je uvek bitno da jedna sonda bude nakačena na masu hardvera i ta sonda je u stvari tzv. GND (*Ground*) sonda uređaja. Pored iglica, mogu biti korišćeni i konektori koji su prilagođeni strukturi sondi koje se koriste.



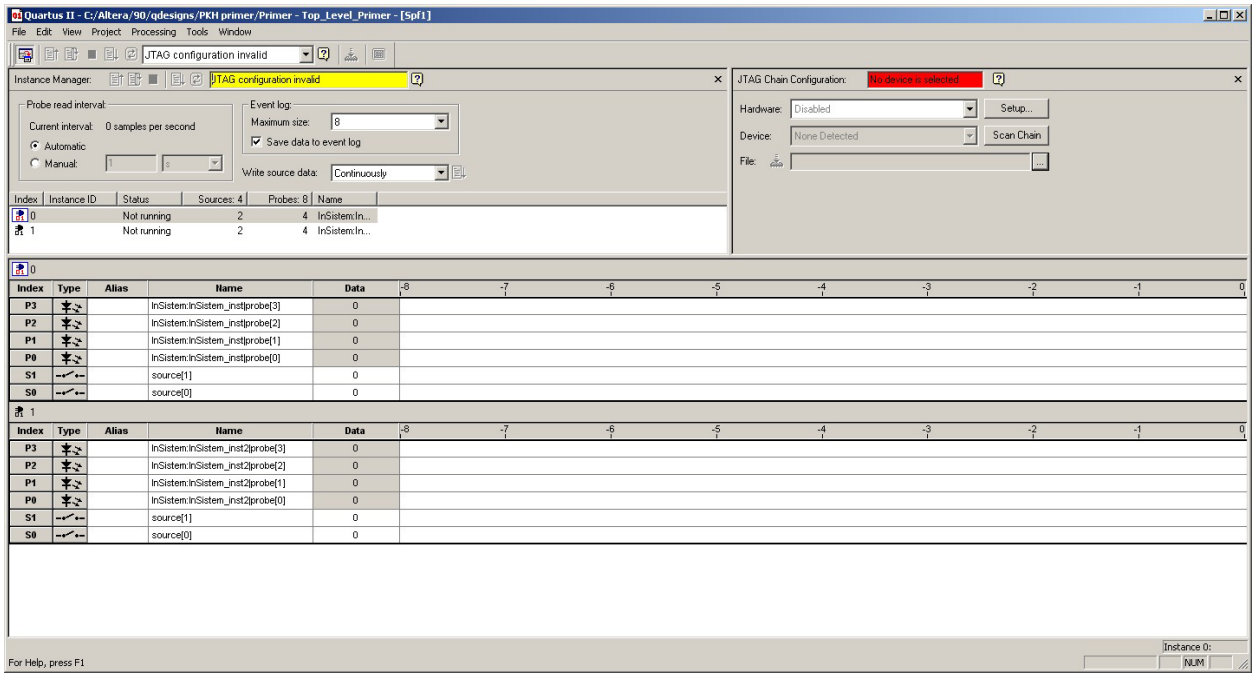
Slika 3.2.20.1. – Prozor za podešavanje interfejsa ka eksternom logičkom analizatoru

Ako želimo da posmatramo interne signale u FPGA čipu, važno je da deo pinova FPGA čipa povežemo sa navedenim ’iglicama’, odnosno tačkama posmatranja. Interne signale koje želimo da posmatramo dodatno povezujemo za dotičnim pinovima (pored regularnih konekcija koje ti signali ostvaruju u dizajnu), a samim tim i sa ’iglicama’ na koje kačimo sonde eksternog logičkog analizatora. Naravno, što više internih signala želimo da posmatramo treba nam veći broj pinova, a broj pinova je ograničen. Stoga Quartus softverski paket nudi mogućnost konfigurisanja interfejsa ka eksternom logičkom analizatoru tako da se više internih signala multipleksira na manji broj pinova FPGA čipa. Ovu mogućnost predstavlja opcija **Tools->Logic Analyzer Interface Editor** iz glavnog menija. Po aktiviranju ove opcije otvara se prozor gde definišemo koje signale želimo da posmatramo i pinove koje koristimo za povezivanje sa eksternim logičkim analizatorom (slika 3.2.20.1). **Logical View** daje logički prikaz trenutnog podešavanja interfejsa, dok **Setup View** daje menije za podešavanje interfejsa. U okviru **Core Parameters** menija se podešavaju osnovni parametri, broj pinova koji se koriste za povezivanje sa logičkim analizatorom (**Pin Count** polje), broj internih signala koji se posmatra (**Bank Count**

polje), izlazni mod posmatranih signala koji može biti kombinacioni ili sekvencijalni preko registra (**Output/Capture mode** polje), takt za slučaj da izlaz ide preko registra (**Clock** polje) i vrednost signala nakon programiranja FPGA čipa (**Power-Up state** polje). U okviru **Pins** menija se selektuju pinovi FPGA čipa koje će da koristi interfejs ka logičkom analizatoru. U okviru **Bank** menija se definišu interni signali koji se žele posmatrati. Dodavanje signala se radi klikom na polje signala i izborom opcije **Add Nodes** iz menija koji se otvori tom prilikom, ili pomoću opcije **Edit->Add Nodes** iz glavnog menija.

3.2.21. Kontrola signala u dizajnu sa host računara

Quartus nudi još dve interesantne opcije koje mogu da olakšaju proces hardverske verifikacije dizajna. Opcija **Tools->In-System Memory Content Editor** iz glavnog menija omogućava da se u toku rada dizajna menja i čita sadržaj internih memorija FPGA čipa sa host računara, što može biti zgodno za upotrebu memorije koja bi predstavljala izvor stimulusa za dizajn u toku procesa verifikacije. Interne memorije kod kojih se želi koristiti ova opcija, moraju imati uključenu mogućnost **Allow In-System Content Editor to capture and update content...** Navedena mogućnost se podešava prilikom kreiranja megafunkcije interne memorije.



Slika 3.2.21.1. – Prozor za kontrolu virtuelnih ulaza i nadgledanje sondi

Opcija **Tools->In-System Sources and Probes Editor** iz glavnog menija omogućava da se u dizajn ubace virtuelni ulazi i sonde koji bi se koristili za stimulisanje dizajna u toku rada sa host računara (slika 3.2.21.1). Pri tome, važno je napomenuti da je ova opcija uglavnom zgodna za mali broj signala koji imaju kontrolne uloge i čijom aktivacijom/deaktivacijom sa host računara možemo stimulisati različite događaje u dizajnu. Na primer, signali reseta i inkrementa u brojaču. S druge strane, sonde bi ovde takođe bile zgodne za mali broj signala i koje bi se koristile kao svojevrsna signalizacija pojedinih događaja - neka vrsta ekvivalenta LED diodama koje se obično koriste za vizuelnu signalizaciju pojedinih događaja u dizajnu. Da bi se opcija **Tools->In-System Sources and Probes Editor** mogla koristiti neophodno je u dizajnu kreirati i instancirati kao komponentu megafunkciju **In-System Sources and Probes** koja je smeštena u

JTAG-accessible Extensions grupi megafunkcija. Na dotičnu komponentu treba povezati željene interne signale, na virtuelne ulaze vezujemo signale koje želimo da kontrolišemo, a na sonde vezujemo signale koje želimo da posmatramo. Prozor za kontrolu virtuelnih ulaza i nadgledanje sondi (slika 3.2.21.1) nudi mogućnosti podešavanja kontrole i nadgledanja. **Probe read interval** podešavanje definiše periodu uzimanja podataka sa sonde. Sonde se mogu očitavati automatski ili se može ručno podesiti perioda očitavanja. Upis kontrolnih signala (**Write source data**) se može vršiti kontinualno (automatski) ili ručno. U **Event log** podešavanju se definiše broj semplova koji će biti zapamćen. Pošto može da se kreira i više instanci komponente za kontrolu virtuelnih ulaza i nadgledanje sondi, postoji i potprozor prikaza postojećih instanci koji se nalazi neposredno ispod prethodno navedenih podešavanja. U donjoj polovini prozora su prikazani virtuelni ulazi i sonde postojećih instanci. Klikom na **Data** polje virtuelnog ulaza se može postaviti nova vrednost virtuelnog ulaza. Isto može da se uradi i selekcijom opcije **Edit->Set Value of Source** iz glavnog menija kada se selektuje željeni virtuelni ulaz.