ELEKTROTEHNIČKI FAKULTET UNIVERZITETA U BEOGRADU

IMPLEMENTACIJA PROVERE ISPRAVNOSTI I AŽURIRANJA IPV4 ZAGLAVLJA UPOTREBOM *MICROBLAZE* PROCESORA

– Master rad –

Kandidat:

Srđan Durković 2013/3105

Mentor:

doc. dr Zoran Čiča

Beograd, Decembar 2014.

SADRŽAJ

1. UVOD

Internet je jedan od simbola modernog doba. Krajem 20-og i u 21-om veku, Internet tehnologije su doživele vrtoglav razvoj. Iako je prvobitno razvijan za potrebe američke vojske, razvojem tehnologije i povećanjem dostupnosti personalnih računara, Internet je postao dostupan većem broju ljudi. To je uslovilo i razvoj novih servisa, koji su privlačili nove korisnike, što je dalje uslovljavalo razvoj Internet infrastrukture. Ključna stvar po kojoj se Internet razlikovao od dotadašnjih komunikacionih tehnologija (npr. telefonije ili telegrafije) je to što Internet koristi komutaciju paketa. To je zahtevalo i razvoj novih mrežnih uređaja (rutera) koji će upravljati tim paketima. Poruke se pre slanja dele na pakete, i zatim nezavisno prenose kroz mrežu. Zadatak rutera je da na osnovu informacija iz zaglavlja paketa, prosledi paket na odgovarajući izlaz. Kako je saobraćaj tokom godina eksponencijalno rastao, to su i kapaciteti mrežnih uređaja postali kritični. FPGA čipovi predstavljaju vrlo zgodan izbor za razvoj novih i unapređenje postojećih tipova rutera, pošto omogućavaju laku modifikaciju i unapređenje dizajna, kao i dodavanje novih funkcionalnosti.

Tema ovog rada je upravo implementacija jedne funkcije paketskog procesiranja (provera i ažuriranje IPv4 zaglavlja) upotrebom *MicroBlaze soft* procesora. Za realizaciju sistema je korišćena *ML507* razvojna ploča proizvođača *Xilinx*. Odabran je *soft* tip procesora, jer pruža veliku fleksibilnost u pogledu izbora periferija i performansi samog procesora. Za razvoj ovog dizajna korišćen je softverski alat EDK (*Embedded Development Kit*) proizvođača *Xilinx*. EDK se sastoji od dve komponente: XPS (*Xilinx Platform Studio*) i SDK (*Software Development Kit*). XPS služi za razvoj hardverskog dela projekta, i znatno olakšava dizajniranje sistema. Za razvoj softverskog dela projekta, korišćen je SDK, koji je kompatibilan sa EDK alatom. U radu je opisan tok razvoja dizajna, sa objašnjenjima pojedinih opcija koje se nude korisnicima. EDK pruža veliku fleksibilnost u pogledu menjanja dizajna i dodavanja novih funkcionalnosti. SDK je razvojno okruženje u kome se vrši razvoj aplikacije u C/C++ programskim jezicima, koje procesor izvršava. U ovom radu, EDK je korišćen za kreiranje hardverske osnove projekta i razvoj nove periferije, dok je u SDK alatu razvijena aplikacija koja vrši proveru i ažuriranje IPv4 zaglavlja.

Rad je organizovan na sledeći način. U drugom poglavlju je objašenjeno šta se sve podrazumeva pod paketskim procesiranjem, i koje se to funkcije izvršavaju u ruterima. Jedna od njih je tema ovog rada. U trećem poglavlju je dat pregled arhitekture *MicroBlaze* procesora i izložene su njegove najznačajnije karakteristike. U četvrtom poglavlju su objašnjene mogućnosti koje EDK alat pruža korisnicima, kao i njegova uloga u razvoju *embedded* sistema. Predstavljen je proces razvoja dizajna, kao i opcije koje su na raspologanju korisnicima. U petom poglavlju se nastavlja opis razvoja dizajna, implementacijom periferije koja služi za proveru i ažuriranje IPv4 zaglavlja. Hardverski deo periferije se odnosi na ulazno/izlazne jedinice i registre za skladištenje podataka, dok se u softverskom delu razvija aplikacija koja upravlja tom periferijom. U šestom poglavlju je izvršena verifikacija rada periferije, a dat je i tabelarni pregled zahtevanih resursa koje zahteva ovaj dizajn. Na kraju je dat kratak zaključak u kome su izloženi rezultati rada i mogućnosti za dalje razvijanje ovog projekta.

2. PROCESIRANJE PAKETA

Internet je danas dominantna telekomunikaciona mreža. Pored prvobitnih servisa poput pretraživanja veba, razmene tekstualnih poruka ili fajlova, Internet nam danas nudi i druge mogućnosti poput prenosa govora, TV signala i slično. Dakle, Internet infrastruktura pruža mogućnosti za implementaciju više različitih servisa. Stoga, može se reći da je danas pristup Internetu osnovna potreba čoveka bilo da sa radi o zadovoljenju poslovnih, školskih ili privatnih obaveza. Kako je Internet globalna telekomunikaciona mreža nastala povezivanjem više različitih mreža, njegov uspeh bi bio nemoguć bez uvođenja jedinstvenog standarda. Iako je IETF (*Internet Engineering Task Force*) definisao OSI referentni model, Internet se danas de fakto zasniva na TCP/IP (*Transport Control Protocol/Internet Protocol*) protokolu, prema kome razlikujemo 5 različitih slojeva: fizički, MAC, mrežni, transportni i sloj aplikacije.

IP tehnologija se zasniva na komutaciji paketa. To znači da se na predajnoj strani podaci segmentiraju na pakete koji se dalje kroz mrežu prenose nezavisno. Na prijemu se vrši desegmentacija odnosno rekontrukcija primljenih paketa. Svaki paket se sastoji od dva dela:

- *Payload*, odnosno korisna informacija koja se prenosi
- Zaglavlje, odnosno informacije koje služe za pravilno prosleđivanje i upravljanje paketima

Zaglavlje se sastoji od više delova jer svaki sloj ubacuje svoje informacije. Osnovu Interneta čine ruteri čiji je zadatak da provere ispravnost paketa, odrede rutu kojom može da stigne do svog odredišta i proslede ga na odgovarajući izlazni port. Dakle, u ruterima se vrši obrada paketa i to na više nivoa. Procesor fizičkog sloja prima signal, izdvaja takt iz njega, uobličava signal, usaglašava fazu signala, vrši detekciju i dekodovanje bita. Na nivou linka vrše se sledeće funkcije: razgraničavanje okvira, izdvajanje adrese odredišta i prijem okvira ako se adresa odredišta poklapa sa lokalnom adresom, provera ispravnosti okvira, određivanje mrežnog protokola [1].

Proces rutiranja ili prosleđivanja paketa predstavlja najsloženiju mrežnu funkciju, naročito ako se ima u vidu da su brzine prenosa podataka eksponencijalno rasle poslednjih godina. Najprostije, zadatak rutera je sledeći:

- Provera da li je primljeni paket validan
- Određivanje na koji izlazni port treba proslediti paket (lukap funkcija)
- Ažuriranje zaglavlja paketa
- Prosleđivanje paketa na odgovarajući izlazni port

Za izvršavanje ovih funkcija ruter koristi informacije iz IP zaglavlja. Ukoliko se utvrdi greška u zaglavlju, paket se odbacuje. Kako bi se performanse rutera optimizovale, a i zbog mnoštva funkcionalnosti koje poseduju, ruteri se dele na više ravni. To su:

- Ravan podataka
- Kontrolna ravan

- Upravljačka ravan

Ravan podataka je podsistem mrežnog čvora koji prima i šalje pakete od i ka interfejsima, obrađuje ih u skladu sa implementiranim protokolom i dalje ih prosleđuje ili odbacuje. Ove funkcije se implementiraju u hardveru.

Kontrolna ravan obuhvata širok skup kontrolnih protokola koji obezbeđuju ispravno i kvalitetno obavljanje osnovne funkcije rutera. To se odnosi na mehanizme i funkcije kontrole rada ravni podataka poput protokola rutiranja, mehanizama kvaliteta servisa, rezervacije resursa, multikast podrške i sl. Ove funkcije zahtevaju razmenu kontrolnih poruka koje zauzimaju značajno manje resursa nego podaci koji se prenose. Implementiranje ovih protokola u ravni podataka bi dovelo do nefleksibilnosti u konfigurisanju rada rutera kao i prevelikih hardverskih zahteva i one se implementiraju softverski.

Upravljačka ravan obuhvata funkcije kao što su: konfigurisanje i nadgledanje rada rutera, alarmiranje u slučaju neispravnosti, omogućavanje udaljenog pristupa ruteru i sl. Funkcije upravljačke i kontrolne ravni se izvršavaju u okviru istog operativnog sistema, tako da se često ove dve ravni posmatraju kao jedna.

U tabeli 2.1 je dat pregled funkcija koje izvršavaju u odgovarajućim ravnima.

Ravan podataka	Kontrolna ravan	Upravljačka ravan
Provera ispravnosti paketa, ispitivanje i modifikacija IP zaglavlja, filtriranje i klasifikacija paketa, lukap funkcija, šifrovanje/dešifrovanje, mehanizmi odbacivanja paketa u slučaju zagušenja, segmentacija paketa na ćelije fiksne dužine, baferisanje ćelija, raspoređivanje i prosleđivanje ćelija sa ulaznih na izlazne porove itd.	Protokoli rutiranja i određivanje lukap tabele (OSPF, RIP, BGP, MPLS, LDP), multikast podrška (IGMP, PIM), rezervacija resursa (RSVP), autentifikacija i autorizacija, menadžment ključeva za šifrovanje, kontrola pristupa itd.	Nadgledanje i dijagnostika (SNMP, ICMP), udaljeni pristup, interfejs za konfigurisanje (CLI)

Tabela 2.1. Funkcije koje vrše kontrolna, upravljačka i ravan podataka[2]

3. *MICROBLAZE* PROCESOR

Embedded sistemi su računarski sistemi posebne namene. Za razliku od računara opšte namene ovi sistemi izvršavaju specifične, prethodno definisane zadatke i zato su vrlo podesni za obavljanje nekih funkcija u mrežnim uređajima. *Xilinx embedded* rešenja koriste dva tipa procesora: *MicroBlaze* i *PowerPC*. *MicroBlaze* procesor spada u kategoriju tzv. *soft core* procesora, što znači da koristi opšte resurse FPGA čipa pa je moguće kreirati raznovrsne konfiguracije, a samim tim je i pogodan za obavljanje različitih funkcija. Može da se koristi na bilo kojoj *Xilinx* FPGA platformi koja ima dovoljno opštih hardverskih resursa.

3.1. Arhitektura procesora

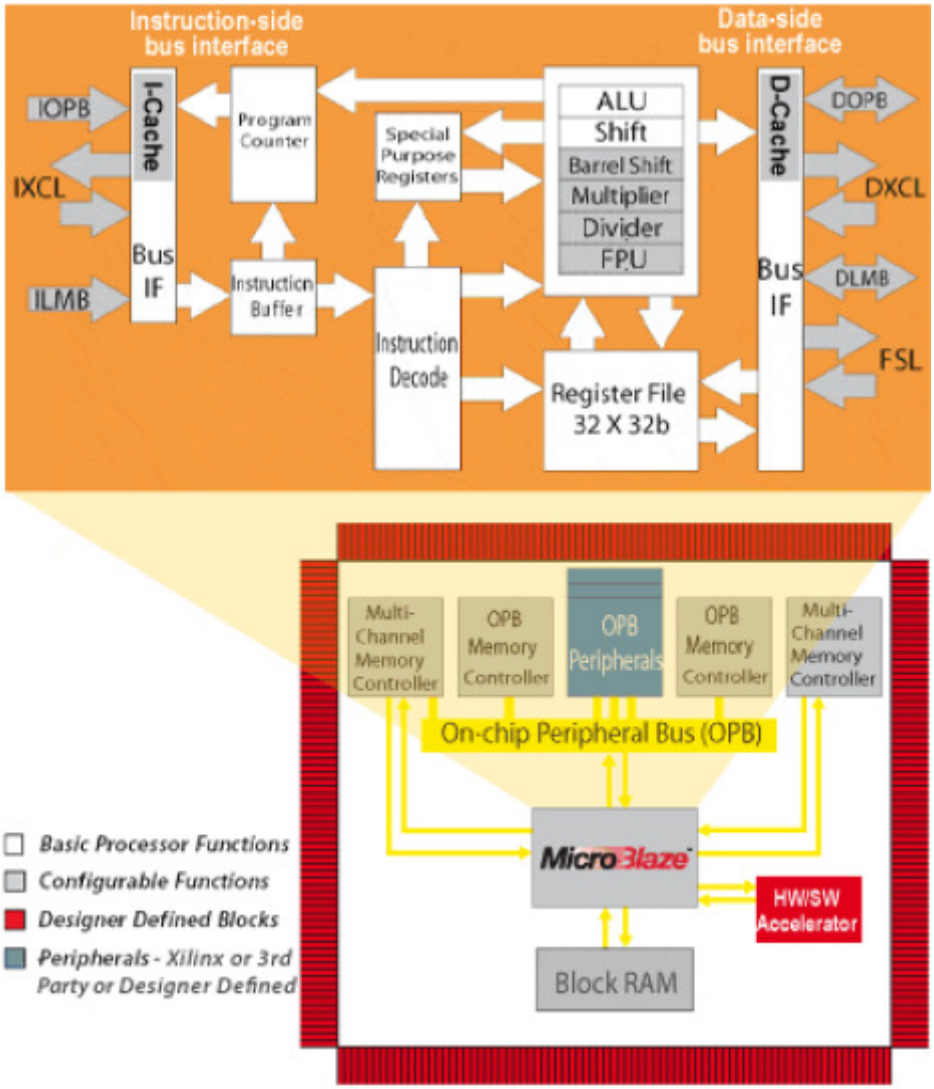
MicroBlaze procesor je 32-bitni RISC (*Reduced Instruction Set Computer*) tip procesora koji može dostići brznu i do 3GHz na najnovijim uređajima. Osnovne karakteristike ovog procesora su[3]:

- Trideset dva 32-bitna registra
- 32-bitne instrukcije
- Odvojene 32-bitne magistrale za instrukcije i podatke koje odgovaraju OPB (*On-chip Peripheral Bus*) specifikaciji
- Odvojene 32-bitne magistrale za instrukcije i podatke sa direktnom konekcijom na *on-chip block* RAM preko LMB (*Local Memory Bus*)
- 32-bitna adresna magistrala

Ovom osnovnom dizajnu mogu biti dodate neke naprednije karakteristike da bi se zadovoljile potrebe konkretne *embedded* aplikacije kao što su[3]:

- Keš memorija za instrukcije i podatke
- Hardverska logika za debagovanje
- FSL (*Fast Simplex Link*) podrška
- Hardverski množač
- *Floating Point Unit*

Ova fleksibilnost dozvoljava korisniku da balansira između zahtevanih performansi konkretne aplikacije i zahteva za hardverskm resursima *soft* procesora. Slika 2.1 prikazuje izgled *MicroBlaze* sistema. Delovi u belom su kičma *MicroBlaze* arhitekture dok su osenčeni delovi opcione mogućnosti dostupne zavisno od potreba konkretne *embedded* aplikacije. Kako je *MicroBlaze soft-core* mikroprocesor, opcione mogućnosti koje nisu implementirane neće zauzimati FPGA resurse [4].



Slika 2.1. *MicroBlaze* procesor [4]

MicroBlaze pipeline je paralelni *pipeline*, podeljen u tri staze: *fetch*, *decode* i *execute*. Generalno, svaka staza zazima jedan takt da se kompletira. Konsekventno, potrebna su tri takta (ne računajući kašnjenja) da bi se instrukcija izvršila. Svaka faza je aktivna jedan takt, tako da se tri instrukcije mogu izvršavati simultano, svaka na jednoj od tri *pipeline* staze. *MicroBlaze* implementira *Insruction Prefetch Buffer* koji smanjuje uticaj kašnjenja *multi-cycle* instrukcija. Dok je *pipeline* zauzet *multi-cycle* instrukcijama, *Instruction Prefetch Buffer* nastavlja da učitava sekvencijalne instrukcije. Jednom kada *pipeline* nastavi izvršavanje, *fetch* staza može učitati novu instrukciju direktno iz *Instruction Prefetch Buffer* umesto da čeka pristup memoriji instrukcija. *IPB* je deo kičme *MicroBlaze* arhitekture i nije isto što i keš memorija za instrukcije [4].

3.2. Organizacija i princip rada procesora

MicroBlaze procesor koristi *bigendian* format za zapisivanje podataka. Hardverski podržani tipovi podataka za *MicroBlaze* su reč, pola reči i bajt. Organizacija bita i bajtova za svaki tip je data u tabelama 2.2-2.4 [3].

Big-Endian Byte Adress	n	n+1	n+2	n+3
Big-Endian Byte Significance	*MSByte*			*LSByte*
Big-Endian Byte Order	n	n+1	n+2	n+3
Big-Endian Byte-Reversed Order	n+3	n+2	n+1	n
Little-Endian Byte Adress	n+3	n+2	n+1	n
Little-Endian Byte Significance	*MSByte*			*LSByte*
Little-Endian Byte Order	n+3	n+2	n+1	n
Little-EndianByte-Reversed Order	n	n+1	n+2	n+3
Bit Label	0			31
Bit Significance	*MSBit*			*LSBit*

Tabela 2.2. Organizacija reči [3]

Big-Endian Byte Adress	N	n+1
Big-Endian Byte Significance	*MSByte*	*LSByte*
Big-Endian Byte Order	N	n+1
Big-Endian Byte-Reversed Order	n+1	n
Little-Endian Byte Adress	n+1	n
Little-Endian Byte Significance	*MSByte*	*LSByte*
Little-Endian Byte Order	n+1	N
Little-EndianByte-Reversed Order	N	n+1
Bit Label	0	15
Bit Significance	*MSBit*	*LSBit*

Tabela 2.3. Organizacija pola reči [3]

Byte Address	N	
Bit Label	0	7
Bit Significance	*MSBit*	*LSBit*

Tabela 2.4. Organizacija bajta [3]

Kao što je već rečeno, sve *MicroBlaze* instrukcije su 32-bitne i razlikujemo dva tipa: tip A i tip B. Instrukcije tipa A imaju do dva izvorišna registra za operande i jedan odredišni registar za rezultat. Instrukcije tipa B imaju jedan izvorišni registar i jedan 16-bitni neposredni operand. Ove instrukcije imaju jedan odredišni registar. Prema funkciji koju obavljaju razlikujemo sledeće vrste instrukcija: aritmetičke, logičke, grananje, učitavanje/iščitavanje i posebne instrukcije. Sve instrukcije zahtevaju jedan takt, osim sledećih [3]:

- Učitavanje i iščitavanje podataka (dva takta)
- Množenje (dva takta)

- Grananje (tri takta)

MicroBlaze je *load/store* tip procesora što znači da može da učitava/iščitava podatke u/iz memorije. Ne može da obavlja nikakve operacije sa podacima u memoriji direktno; umesto toga, podatak iz memorije mora se učitati u *MicroBlaze* procesor i smestiti u registere opšte namene da bi se obavila bilo kakva operacija. Pristup podacima mora biti "poravnat" (tj. pristup rečima mora biti na granicama reči, pristup pola reči mora biti na pola reči), osim ako nije konfigurisan da podržava neporavnate izuzetke. Sav pristup instrukcijama mora biti poravnat na granicama reči. *Stack* konvencija korišćena u *MicroBlaze* procesorima počinje sa veće memorijske lokacije i ide dalje ka nižim memorijskim lokacijama. Podaci se iz memorije iščitavaju u suprotnom smeru; podatak sa najniže memorijske lokacije ide prvi itd. [4].

MicroBlaze procesor ima tridesetdva 32-bitna registra opšte namene i do osamnaest registara posebne namene, zavisno od konfiguracije. Registri opšte namene se obeležavaju od R0 do R31. Registri posebne namene su: *Program Counter* (PC) za određivanje adrese sledeće instrukcije, *Machine Status Register* (MSR) pokazuje status procesora npr. indikacija deljenja sa nulom, *Exception Address Register* (EAR) u koji se smešta *load/store* adresa koja je izazvala izuzetak, *Exception Status Register* (ESR) koji ukazuje kakav tip izuzetka se desio, *Floating Point Status Register* (FSR) koji ukazuje na stvari kao što su nevalidna operacija, deljenje sa nulom, *overflow*, *underflow* [4].

MicroBlaze takođe podržava *reset*, *interrupt*, *user exception*, *break* i *hardware* izuzetke. Za *interrupt* signale, *MicroBlaze* podržava samo eksterne *interrupt* izvore (koji se povezuju na *Interrupt* input port). Ukoliko su neophodni višestruki *interrupt* signali mora biti korišćen *interrupt* kontroler da upravlja višestrukim *interrupt* zahtevima ka *MicroBlaze* procesoru. *Interrupt* kontroler je dostupan za upotrebu u EDK softverskom alatu. Procesor će reagovati na *interrupt* signale ako je *Interrupt Enable* (IE) bit podešen na 1 u *Machine Status Register* (MSR). Takođe, dok traje prekidna rutina procesor onemogućava buduće *interrupt* signale brišući IE bit u MSR da bi se sprečilo gneždenje prekida što može izazvati nepredvidivosti u izvršavanju koda. IE bit je automatski postavljen opet kad se izvršava RTID instrukcija [4].

Postoji više softverskih alata koji se mogu koristiti za razvoj *embedded* sistema kao i više tipova čipova na kojima se oni mogu implementirati, od onih jeftinjih do skupljih koji nude brojne funkcionalnosti. Neke od njih su [3]:

- *Spartan-3/3A/3AN/3A DSP/3E*
- *Spartan-6*
- *Virtex-4 FX/LX/SX*
- *Virtex-5 FX/LX/SX*
- *Virtex-6*

PowerPC procesor je dostupan samo na pojedinim familijama (npr. *Viretx-4FX, Virtex-5FX* familije) jer on u suštini predstavlja fizički ugrađen procesor (dotični čipovi imaju hardverski ugrađen procesor). Naravno, i u familijama čipova koje imaju hardverski ugrađen procesor, može se i dalje implementirati i koristiti MicroBlaze *soft* procesor (koji će koristiti resurse opšte namene u čipu). Prednost *soft* pristupa *MicroBlaze* procesora je što se koristi onoliko hardverskih resursa koliko je potrebno. Na ovaj način korisnik može fleksibilno da izabere odgovarajući model koji je optimalan sa stanovištva njegovih potreba i potrošnje hardverskih resursa čipa.

3.3. Performanse procesora

Osnovne performanse *MicroBlaze* procesora su [3]:

- Radna učestanost (maksimalna vrednost za sisteme sa malo zahtevnim periferijama):
  - 307 MHz na *Virtex-6*
  - 245 MHz na *Virtex-5*
  - 154 MHz na *Spartan-6*
  - 119 MHz na *Spartan-3*
- Potrošnja LUT elemenata čipa
  - 779/1134 LUT na *Virtex-6*
  - 240/330 LUT na *Virtex-5*
  - 770/1154 LUT na *Spartan-6*
  - 1258/1821 LUT na *Spartan-3*
- Vrste magistrala:
  - *Master* - imaju mogućnost da iniciraju razmenu podataka
  - *Slave* - mogu samo da odgovore na zahtev
  - Arbitraža se sastoji iz tri koraka
    - Uređaj koji želi da postane *master* šalje zahtev
    - Arbitar kontinuirano nadgleda zahteve i šalje individualne *grant* signale svakom *master* uređaju u skladu sa šemom *master* prioriteta i sa stanjem ostalih *master* zahteva u tom trenutku
    - *Master* koji je poslao zahtev čeka dok ne dobije pristupni *grant*. Kad postojeći *master* oslobodi magistralu, *master* tada koristi adresne i kontrolne magistrale da inicira razmenu podataka ka *slave* agentu.
  - Mehanizmi arbitraže:
    - Fiksni prioritet
    - *Round-robin*
    - Hibridni

I *MicroBlaze* i *PowerPC* procesori koriste PLB v46 standard za komunikaciju sa svojom periferijom. Ovaj standard je dizajniran za komunikaciju sa periferijama koje su veoma zahtevne u pogledu brzine i propusnog opsega. Bitno je napomenuti da EDK koristi IP (*Intelectual Property*) jezgra koja su prilagođena ovom standardu. Takođe, sva IP jezgra koja su optimizovana za *MicroBlaze* procesore mogu se koristiti i za *PowerPC* i obratno.

Xilinx embedded sistem može biti praktično bilo koji računarski sistem. Tipično je projektovan da obavlja jednu funkciju tako da je mali i efikasan. Glavna prednost ovog tipa

procesora upravo je njegova fleksibilnost koja nam omogućava da obavlja različite funkcije. Na taj način značajno štedimo FPGA resurse i poboljšavamo performanse (pre svega brzinu) *embedded* sistema. Ovi sistemi su dizajnirani tako da budu što jeftiniji, minimalno troše energiju, da budu relativno brzi i da minimalno koriste FPGA resurse [3]. Važno je naglasiti i da se sistemi zasnovani na *embedded* procesorima lako modifikuju i ažuriraju (ako nema potrebe za izmenom periferije) pošto je potrebno izmeniti samo softverski kod po kom radi procesor, bez ikakve izmene u hardveru.

Softver koji upravlja sistemom je smešten u *block* RAM memoriji. Problem može biti što je na FPGA pločama ograničena veličina RAM memorije. Kada su ograničeni memorijskim kapacitetima, većina programera koristi *off-chip* memoriju, tako da je *Xilinx* razvio i nekoliko tipova memorijskih kontrolera koji obezbeđuju ovu funkcionalnost. Pored već pomenutog PLB standarda, *MicroBlaze* podržava i LMB standard koji se koristi za komunikaciju sa lokalnom *block* RAM memorijom. Takode, podržava i FSL standard koji se najčešće koristi za povezivanje sa posebnim DSP blokovima na kojima se vrše kompleksna izračunavanja i složenije matematičke operacije. *MicroBlaze* procesor ima do 16 FSL linkova. Na primer, za izračunavanje zbira dva broja bile bi potrebne 3 magistrale. Dve za slanje operanada ka DSP uređaju, i jedna za slanje rezultata ka procesoru. Upotreba FSL magistrala nam omogućava da "štedimo" nekoliko ciklusa na PLB magistrali, a takođe ne opterećujemo memorijske kapacitete [3,4].

.

4. *EMBEDDED DEVELOPMENT KIT*

Xilinx Embedded Development Kit (EDK) je skup alata i *Intellectual Property* (IP) funkcionalnosti koji korisnicima omogućavaju da dizajniraju kompletan *embedded* sistem koji se može implementirati na FPGA čipu. Ovo poglavlje opisuje proces razvijanja ovakvog sistema koristeći EDK alat. U daljem tekstu su objašnjene mogućnosti koje su na raspologanju korisnicima tokom razvoja sistema. Data su precizna uputstva tako da čitaoci mogu odmah da krenu sa korišćenjem ovog alata.

Embedded sistemi su vrlo kompleksni. Sam razvoj hardverskog ili softverskog dela projekta je vrlo zahtevan poduhvat. Situacija se dodatno usložnjava kada treba ove dve komponente uskladiti da funkcionišu kao jedan sistem i konačno sve to implementirati na FPGA čipu. Kako bi pojednostavio ovaj proces, *Xilinx* nudi nekoliko setova alata, koji se jednim imenom zovu ISE *Design Suite*. Razlikujemo dve komponente EDK softvera: *Xilinx Platform Studio* (XPS) i *Software Development Kit* (SDK). XPS je razvojno okruženje koje se koristi za definisanje hardverskog dela sistema. XPS alatom se može upravljati koristeći *bash shell* komandnu liniju ili GUI, što će biti demonstrirano u ovom radu. SDK je integrisano razvojno okruženje komplemetarno sa XPS alatom koje služi za kreiranje i verifikaciju C/C++ aplikacija koje procesor izvršava [5].

Tipično, ISE razvojno okruženje se koristi za kreiranje *Embedded Processor source* fajla, koji je zatim dodat ISE projektu.

- U XPS alatu se vrši specifikacija mikroprocesora, periferija i interkonekcija između ovih komponenti.
- SDK se koristi za razvoj softvera.

4.1. *Base System Builder*

BSB (*Base System Builder*) je vizard u XPS alatu koji brzo i efikasno formira radni dizajn. Nakon toga se ovaj dizajn može menjati i prilagođavati. *Xilinx* preporučuje korišćenje BSB vizarda za kreiranje osnove svakog novog *embedded* sistema. Takođe, u slučaju da se ukaže potreba za nekim promenama BSB značajno štedi vreme automatizovanjem standardnih zahteva za konfiguracijom hardverske i softverske platforme. U vizardu se formira radna verzija projekta koja sadrži sve osnovne elemente koji su neophodni za kompleksnije sisteme. BSB omogućava kreiranje *project* fajla; izbor ploče; izbor i konfiguraciju procesora i ulazno/izlaznih interfejsa; dodavanje periferija; kreiranje softvera; i generisanje izveštaja o projektu [5].

BSB prepoznaje komponente sistema i konfiguraciju izabrane ploče, i omogućava adekvatne opcije u skladu sa potrebama korisnika. Kao što je već rečeno, BSB pruža više mogućnosti za izbor odgovarajuće ploče. Može se izabrati neka od opcija koje su na rasploganju, od *Xilinx* proizvoda ili od njegovih partnera. Nakon što se izaberu i odgovarajuće periferije, BSB kreira UCF (*User Constraints File*) fajl koji dodeljuje pinove izabranim periferijama. Takođe, BSB kreira i kompletnu platformu i test aplikacije koje su spremne da budu spuštene i pokrenute na ploči. Za svaku odabranu opciju postoje podrazumevane vrednosti, koje se mogu menjati u XPS alatu. Ukoliko se dizajn vrši za određenu ploču, BSB omogućava izbor vrste procesora (*MicroBlaze* ili *PowerPC*,

zavisno od izabranog FPGA čipa pri čemu je *PowerPC* ponuđen samo za one čipove koji imaju hardverski ugrađen procesor), sa različitim kompatibilnim i često korišćenim periferijama. Tako dobijamo početnu hardversku platformu. Naknadno se može dodati još procesora ili periferija, ako je potrebno. Mogu se podešavati sledeće opcije kod procesora [5]:

- Frekvencija referentnog takta
- Frekvencija takta na procesorskoj magistrali
- Konfiguracija procesora za debagovanje
- Podešavanje keša
- *Floating Point Unit* (FPU) podešavanja

BSB razume koji tipovi eksterne memorije i ulazno/izlaznih uređaja su dostupni na izabranoj ploči i omogućava korisnicima da izabere sledeće:

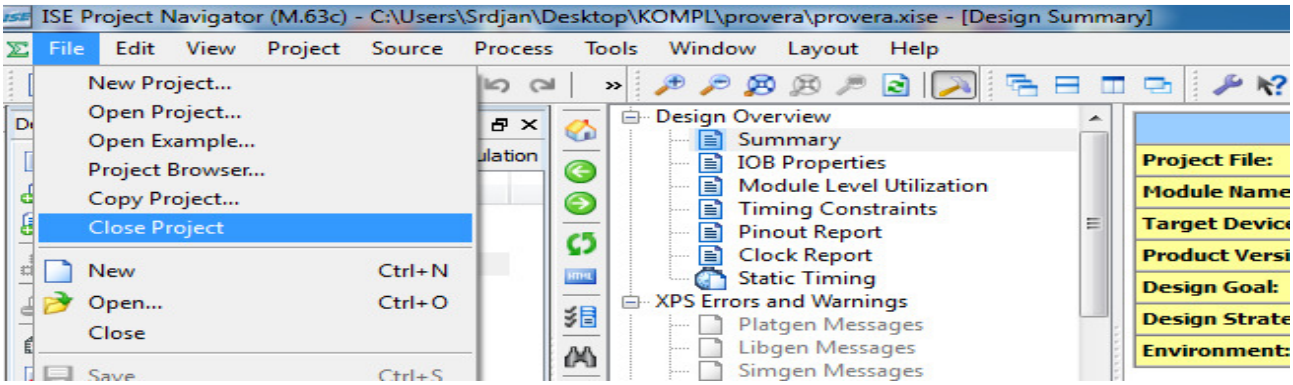
- Tip uređaja
- Tip perfiferije
- Brzina u baudima
- *Parity*
- Korišćenje signala prekida (*interrupts*)

Za eksternu memoriju i ulazno/izlazne uređaje je dostupna i detaljna dokumentacija. BSB vizard pruža mogućnost da se ubace dodatne periferije. Periferije moraju biti podržane od strane izabrane FPGA ploče. Nakon što se odaberu ulazni i izlazni uređaji u BSB vizardu, generišu su i jednostavne C aplikacije. Za razvoj softvera je preporučen SDK alat, o čemu će više reči biti kasnije. Nakon što se izaberu odgovarajuća podešavanja, BSB prikazuje sumarni izveštaj o sistemu. Pre nego što se završi sa generisanjem projekta korisnik se može vratiti na neku od prethodnih stranica i promeniti neka podešavanja. U ovom radu biće korišćen *Virtex-ML507* ploča sa *MicroBlaze* procesorom. Neke ploče nude i mogućnost izbora *PowerPC* procesora. U gotovo svim slučajevima način korišćenja alata je isti kao i za *MicroBlaze*. Napomenimo da se konfiguracija procesora može vršiti i za ploču koja nije ponuđena (*proprietary* ploča), ali tada se pojedina podešavanja moraju dodatno uraditi pošto alat nema podatke o takvoj ploči u svojoj bazi (npr. moraju se podesiti lokacije pinova, definisati periferije koje su na raspolaganju i dr.).

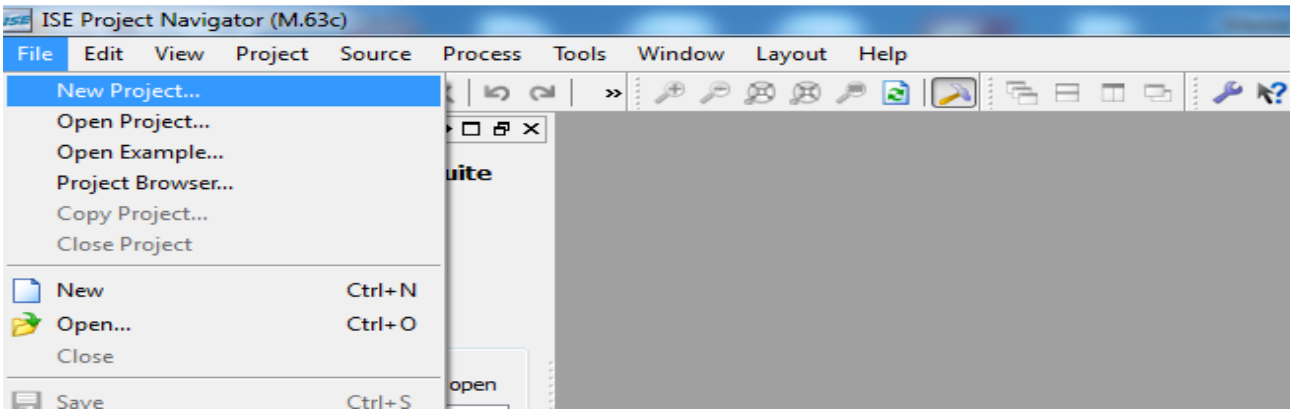
4.2. Kreiranje novog *embedded* projekta

U ovom potpoglavlju, sledi detaljan opis razvoja *embedded* sistema, korak po korak. Kreiraće se novi projekat, u koji je uključen *embedded* procesor. ISE automatski startuje XPS alat i otvara BSB kako bi kompletirali dizajn. XMP fajl je *top-level* fajl koji sadrži opis *embedded* sistema. Sve informacije o projektu su sačuvane u ovom fajlu. XMP je kreiran i njime se upravlja u ISE alatu, kao i sa drugim *source* fajlovima, kao što su HDL kod ili UCF fajlovi [5].

Dakle, prvi korak je pokretanje ISE *Project Navigator* alata. Ukoliko je neki projekat otvoren, trebalo bi ga zatvoriti, klikom na opciju *File* > *Close project,* slika 4.2.1. Zatim se formira novi projekat sa *embedded* processor sistemom kao *top level* entitetom, izborom opcije *File* > *New Project*, slika 4.2.2. Prilikom kreiranja projekta, biraju se opcije nabrojane u tabeli 4.2.3 u skladu sa činjenicom da je projekat namenjen za rad sa pločom *ML507*.



Slika 4.2.1. Zatvaranje projekta u ISE alatu



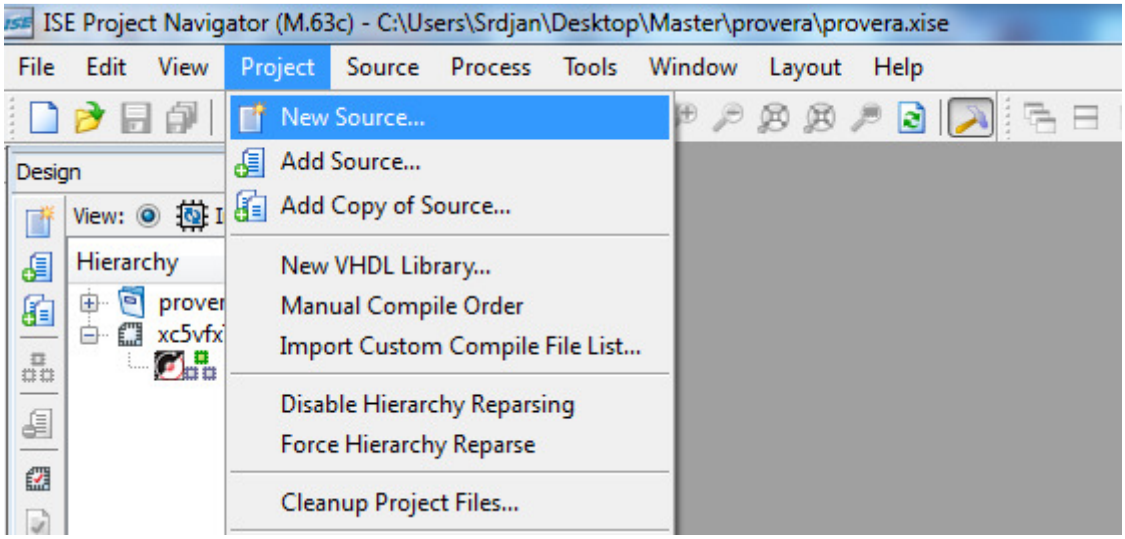
Slika 4.2.2. Otvaranje novog projekta

Wizard Screen	***System Property***	***Setting or Command to Use***
Create New Project	*Name*	Odabrati ime projekta (bez razmaka). U ovom primeru, ime je *provera*.
	Loacation and Working Directory	Odabrati lokaciju i radni direktorijum (bez razmaka). U ovom primeru putanja je *Desktop/Master*.
	Description	Može se dodati i opis projekta (opciono).
	Top-level source type	*Select HDL* (podrazumevano)
Project Settings	*Product Category*	*All*
	Family	*Virtex5*
	Device	*XC5VFX70T*
	Package	*FF1136*
	Speed	*-1*
	Synthesis Tool	*XST (VHDL/Verilog)*

	Simulator	*User-specific*
	Preferred Language	*VHDL*
	Ostalo prihvatiti podrazumevan	podešavanja
Project Summary	Pokazuje pregled izabranih opcija u New Project vizardu.	Bez promena.

Tabela 4.2.3. Podešavanja koja treba odabrati prilikom kreiranja novog projekta

Kad se završi sa definisanjem projekta, klikne se na dugme *Finish*, *New Project* vizard se zatvara i ISE *Project Navigator* otvara projekat koji smo upravo kreirali. Sledeći korak je dodavanje *source* fajla, korišćenjem *New Source* vizarda. U opcijama se klikne na *Project* > *New Source* dugme kao na slici 4.2.4, i biraju se opcije navedene u tabeli 4.2.5.



Slika 4.2.4. Dodavanje novog *Source* fajla u projekat

Wizard Screen	***System Property***	***Setting or Command to Use***
Select Source Type	*Source Type*	*Embedded Processor*
	File name	system
	Location	Prihvatiti podrazumevanu lokaciju.
	Add to project	Ostaviti čekirano
Project Summary	Pokazuje pregled izabranih opcija u New Source vizardu	Bez promena

Tabela 4.2.5. Podešavanja koja treba odabrati prilikom dodavanja novog *Source* fajla

Nakon završetka, ISE prepoznaje da smo izabrali *embedded* sistem i pokreće XPS. Trebalo bi da se pojavi dijalog prozor, sa pitanjem da li se želi kreiranje *Base system* projekta upotrebom BSB vizarda. Nakon prihvatanja, kreira se projekat sa podešavanjima datim u tabeli 4.2.6. (ukoliko su neka podešavanja preskočena u tabeli, za ta podešavanja su korišćene podrazumevane (difolt) vrednosti.

Wizard Screen	*System Property*	*Setting or Command to Use*
Welocome to the Base System Builder	*Project type options*	*I would like to create a new design*
Board selection	*Board Vendor*	*Xilinx*
	Board Name	*Virtex 5 ML507 Evaluation Platform*
	Board Revision	*A*
System Configuration	*Type of system*	*Single-Processor System*
Processor Configuration	*Processor Type*	*MicroBlaze*
	System Clock Frequency	*100 MHz*
	Local Memory	*64 KB*
	Enable Floating Point Unit	Otčekirati ovu opciju
Peripheral Configuration	*Processor 1 (MicroBlaze)* *Peripheral list*	Ukloniti sledeće periferije sa liste podrazumevanih vrednosti *"Processor 1 (MicroBlaze)":* • *IIC_EEPROM* • *SRAM* • *DDR2_SRAM* • *Ethernet_MAC* • *Hard_Ethernet_MAC* • *LEDs_Positions* • *SysACE_CompactFlash* • *FLASH* • *RS232_Uart_1* • *RS232_Uart_2* • *PCIe Bridge*
Cache Configuration		Kliknuti *Next*
Application Configuration	*Example Application Options*	Ostaviti podrazumevane vrednosti
Summary	*System Summary Page*	

Tabela 4.2.6. Podešavanja koja treba odabrati u BSB alatu

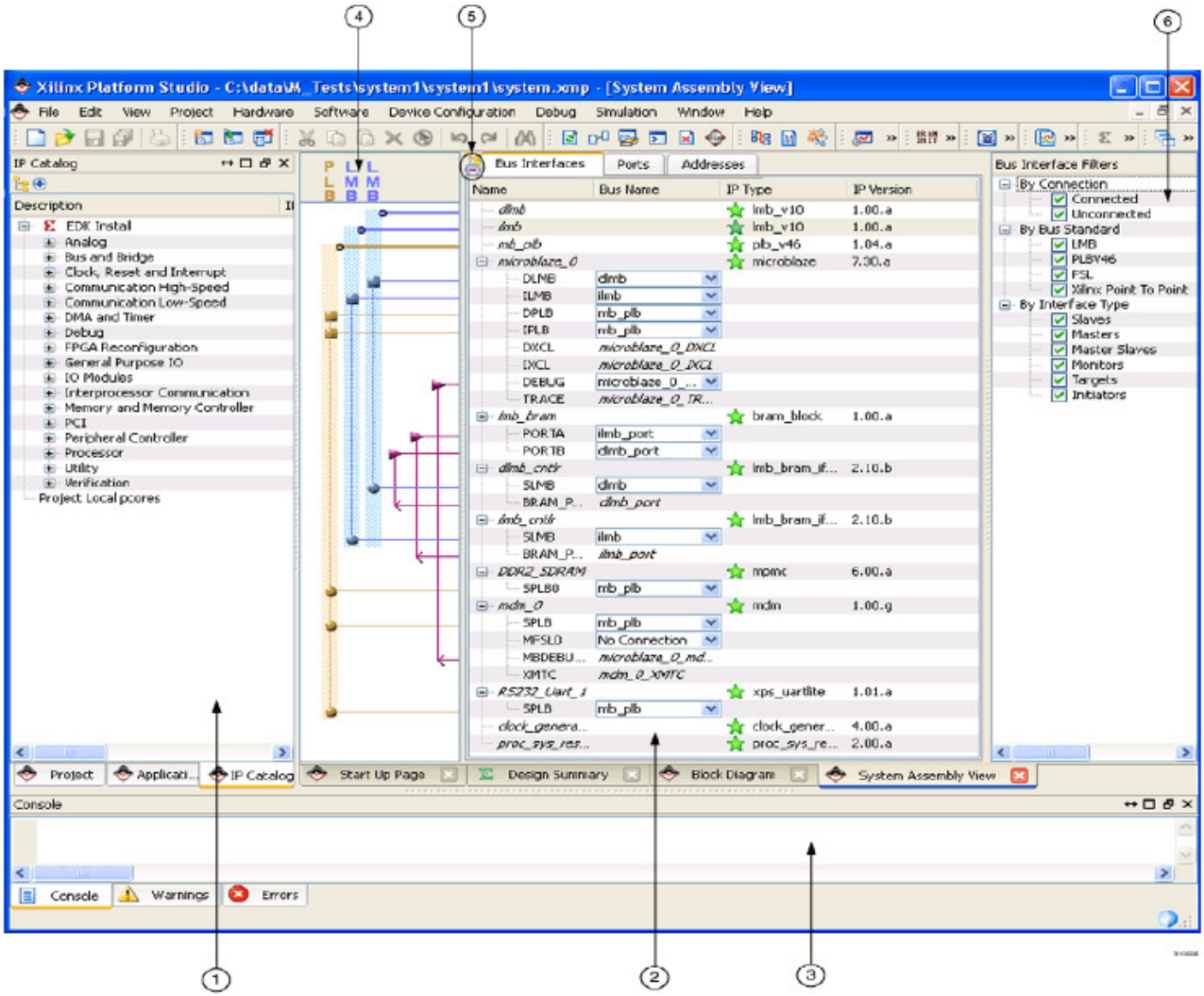
Napravimo pregled odabranih opcija. Prvo je ponuđeno da li da se kreira novi dizajn ili da se učita postojeći. Pošto ništa još nije kreirano, bira se kreiranje novog dizajna. Dalje se nudi izbor FPGA ploče. U ovom primeru koristiće se *Xilinx Virtex 5 ML507* ploča. BSB nudi mogućnost razvijanja dizajna i sa dva procesora, ali će se ovde koristiti sistem sa jednim procesorom (*single processor*). Izabran je *MicroBlaze* procesor, sa radnom učestanošću od 100 MHz i lokalnom

memorijom od 64 KB. Ovde je izabrana maksimalna vrednost RAM memorije, mada je i ona dosta skromna. Ovo može prestavljati problem, jer aplikacija koje se izvršava na procesoru mora biti optimizovana da stane u dodeljenu memoriju. Na to treba obratiti pažnju prilikom pisanja C koda. Od periferija, u ovom primeru će biti korišćene samo LED diode, DIP svičevi i *Push* dugmad. Ostale ponuđene periferije treba ukloniti, selektovanjem date periferije i klikom na *Remove*. Ostalo treba prihvatiti podrazumevane (difolt) vrednosti. Kad se završi sa svim selekcijama, klikne se na *Finish*, a onda u dijalog prozoru *OK*.

4.3. Xilinx Platform Studio

Nakon što je kreirana osnova projekta uz pomoć BSB vizarda, XPS nudi mogućnost ažuriranja i nadogradnje projekta. Na slici 4.3.1. je dat izgled glavnog prozora u XPS. Ovaj prozor je podeljen na tri dela:

- *Project Information Area* (1)
- *System Assembly View* (2)
- *Console Window* (3)



Slika 4.3.1. Izgled XPS prozora[5]

U glavnom prozoru postoje i tri labele koje identifikuju sledeće obalsti:

- *Connectivity Panel* (4)
- *View Buttons* (5)
- *Filters* (6)

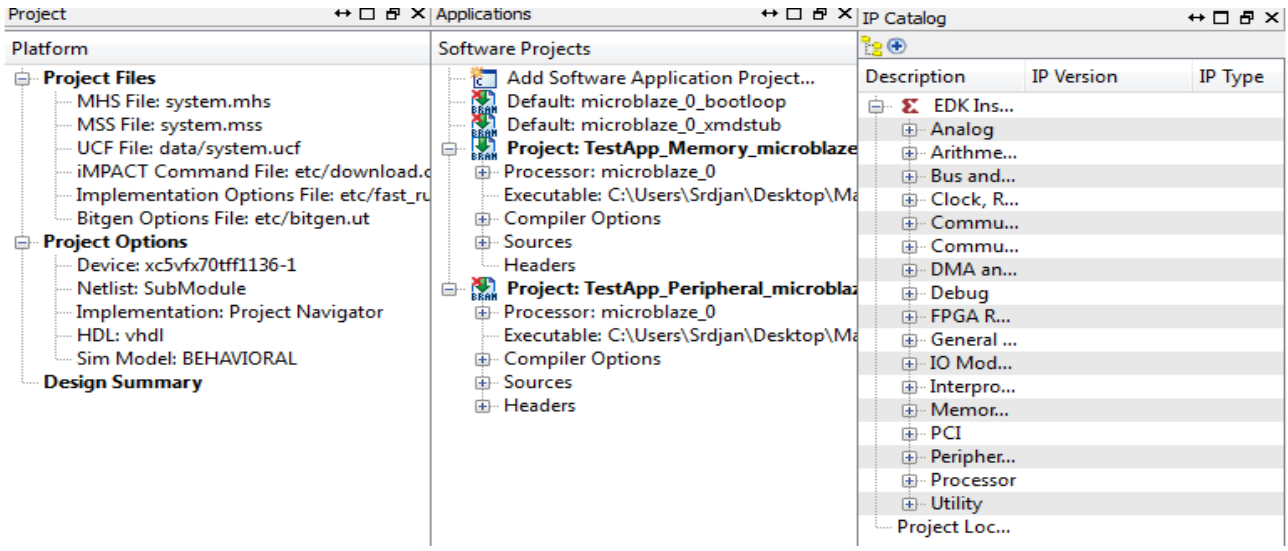
4.3.1. Project Information Area

Project Information Area nudi informacije o projektu i omogućava njegovu kontrolu. Ovaj deo prozora sadrži: *Project, Applications* i *IP Catalog* tabove.

i) Project Tab

Project tab, prikazan na slici 4.3.1.1 a), daje listu referenci na fajlove vezane za projekat. Postoje tri kategorije informacija[5]:

- *Project Files* sadrži: *Microprocessor Hardware Specification* (MHS) fajl, *Microprocessor Software Specification* (MSS) fajl, *User Constarints File* (UCF), *iMPACT Command* fajl, *Implentation Option* fajl i *Bitgen Option* fajl
- *Project Options* pruža informacije kao što su: *Device, Netlist, Implementation, Hardware Description Language (HDL) i Sim Model*
- *Design Summary* sadrži grafički izgled stanja *embedded* dizajna i daje lak pristup sistemskim fajlovima



a) **b)** **c)**

Slika 4.3.1.1. a) *Project Tab*, b) *Applications Tab*, c) *IP Catalog Tab*

ii) Applications Tab

Applications tab, prikazan na slici 4.3.1.1 b), daje listu podešavanja opcija softvera, *header* fajlova, i *source* fajlova koji su povezani sa aplikacijom. U ovom tabu može se [5]:

- Kreirati i dodati softver, izgraditi projekat, i smestiti u RAM memoriju
- Podesiti opcije kompajlera

- Dodati *source* i *header* fajlovi u projekat

Bitno je napomenuti da, iako se softver može kreirati i njime upravljati u XPS, ipak se za te svrhe preporučuje korišćenje SDK alata.

iii) IP Catalog Tab

IP Catalog tab, prikazan na slici 4.3.1.1 c), pruža informacije o IP jezgrima, uključujući[5]:

- Ime jezgra i status licence (nelicenciran, zaključan, ili otključan)
- Verzija i status
- Podržani procesori
- Klasifikacija

Dodatne informacije o IP jezgrima, uključujući istoriju izmena, specifikaciju i *Microprocessor Peripheral Description* (MPD) fajl, su dostupne kad se klikne desno dugme na mišu na IP jezgru u *IP Catalog* tabu. IP jezgra su grupisana hijerarhijski po funkciji.

4.3.2. System Assembly View

System Assembly View pruža mogućnost da se vide i konfigurišu elementi sistema. Ako prethodno nije maksimizovan, otvoriti *System Assembly View* tab na dnu panela.

i) *Bus Interface, Ports, and Addresses* Tab

System Assembly View sadrži tri panela, koji se mogu otvoriti klikom na odgovarajuću opciju:

- *Bus Interface* tab prikazuje magistrale korišćene u dizajnu. Koristi se za izmenu informacija i konekcija za svaku magistralu
- *Ports* tab prikazuje portove korišćene u u dizajnu. Sadrži detalje i opcije za promenu portova
- *Addresses* tab prikazuje opseg adresa za svaku IP instancu u projektu. Klikom na opciju *Generate Addresses* automatski se generiše adresna mapa sistema.

ii) Connectivity Panel

Kada se izabere *Bus Interface* tab, vidi se *Connectivity Panel*, koji predstavlja grafičku reprezentaciju konekcija hardverske platforme. Prelaskom mišem preko *Connectivity Panel* vide se dostupne konekcije na magistralama. Vertikalna linija predstavlja magistralu, a horizontalna linija predstavlja interfejs magistrale ka IP jezgru. Ako je moguće napraviti kompatibilnu konekciju, konektor je prikazan na preseku magistrale i interfejsa ka IP jezgru. Linije su u boji, kako bi pokazale kompatibilnost magistrala. Različiti simboli pokazuju da li je IP blok povezan na magistralu kao *master* (kvadrat) ili *slave* (krug). Prazni konektor pokazuje konekciju koja se može uspostaviti. Popunjeni konektor predstavlja postojeću konekciju. Da bi se konekcija uspostavila ili raskinula klikne se na simbol konektora.

iii) Filters

XPS omogućava filetere koji se mogu korisititi da se promeni izgled *Bus Interface* i *Ports* taba u *System Assembly View*. Filteri su izlistani u *Filters*, kada su *Bus Interface* ili *Ports* tabovi

izabrani. Korišćenje ovih filtara može uprostiti izgled *connectivity panel* kad se kreira dizajn sa velikim brojem različitih magistrala.

iv) View Buutons

Kako bi se mogle lakše sortirati informacije i revidirati dizajn, *System Assembly View* omogućava dva dugmeta koji menjaju način na koji su podaci prikazani:

- *Change to Hierarchial/Flat View* dugme
  - Podrazumevani izgled se zove hijerahijski izgled. Informacije koje su prikazane za dati dizajn su bazirane na instancama IP jezgara na hardverskoj platformi i organizovane u obliku proširivog stabla.
  - U *flat view*, informacije su sortirane alfanumerički po svakoj koloni
- *Expand/Collapse All Tree Nodes* dugme

  Ikonica +/- proširuje ili skuplja sve veze između magistrale i IP jezgara i omogućava brzo i lako povezivanje periferija sa magistralom.

v) Console Window

Console window omogućava povratne informacije tokom izvršavanja nekih opcija. Postoje tri taba: *Console, Warnings* i *Errors*.

vi) Start Up strana

Start Up strana sadrži relevantne informacije vezane za verziju XPS koja se koristi, uključujući set linkova sa različitim informacijama. Takođe, tu je i tab koji olakšava lociranje EDK dokumentacije. Ukoliko *Start Up* strana nije već otvorena, može se otvoriti klikom na *Help* > *View Start Up Page.*

vii) Directory View

BSB automatski kreira direktorijum projekta sa imenom *embedded* sistema koji je kreiran. U ovom direktorijum su sadržani poddirektorijumi vezani za projekat [5]:

- *_xps*
  Sadrži fajlove generisane od strane XPS i drugih alata za interni menadžment projekta. Ovaj direktorijum korisnici ne koriste.
- *blockdiagram*
  Sadrži fajlove vezane za blok dijagram
- *data*
  Sadrži UCF fajl.
- *etc*
  Sadrži fajlove koji sadrže opcije koje služe za korišćenje raznih alata. Ovaj direktorijum je prazan ako se ne primenjuju nikakve akcije van BSB.
- *pcores*
  Koristi se za dodavanje periferija. Detaljnije će biti objašnjen u nastavku.

Druga dva direktorijuma sadrže fajlove generisane od strane BSB:

- *TestApp_Memory_microblaze_0*

- *TestApp_Peripheral_microblaze_1*

Ovi direktorijumi sadrže C kod test aplikacije i *header* fajlove. Iako su dostupni za upotrebu, u ovom radu će se koristiti aplikacija koja će biti razvijena u SDK alaltu, o čemu će biti reči kasnije.

U direktorijumu glavnog projekta prikazanom na slici 4.2.3.1, postoji takođe nekoliko fajlova. Neki od njih su [5]:

- system.xmp

  Ovo je *top-level* fajl. XPS čita ovaj fajl i grafički prikazuje njegov sadržaj u XPS korisničkom interfejsu.

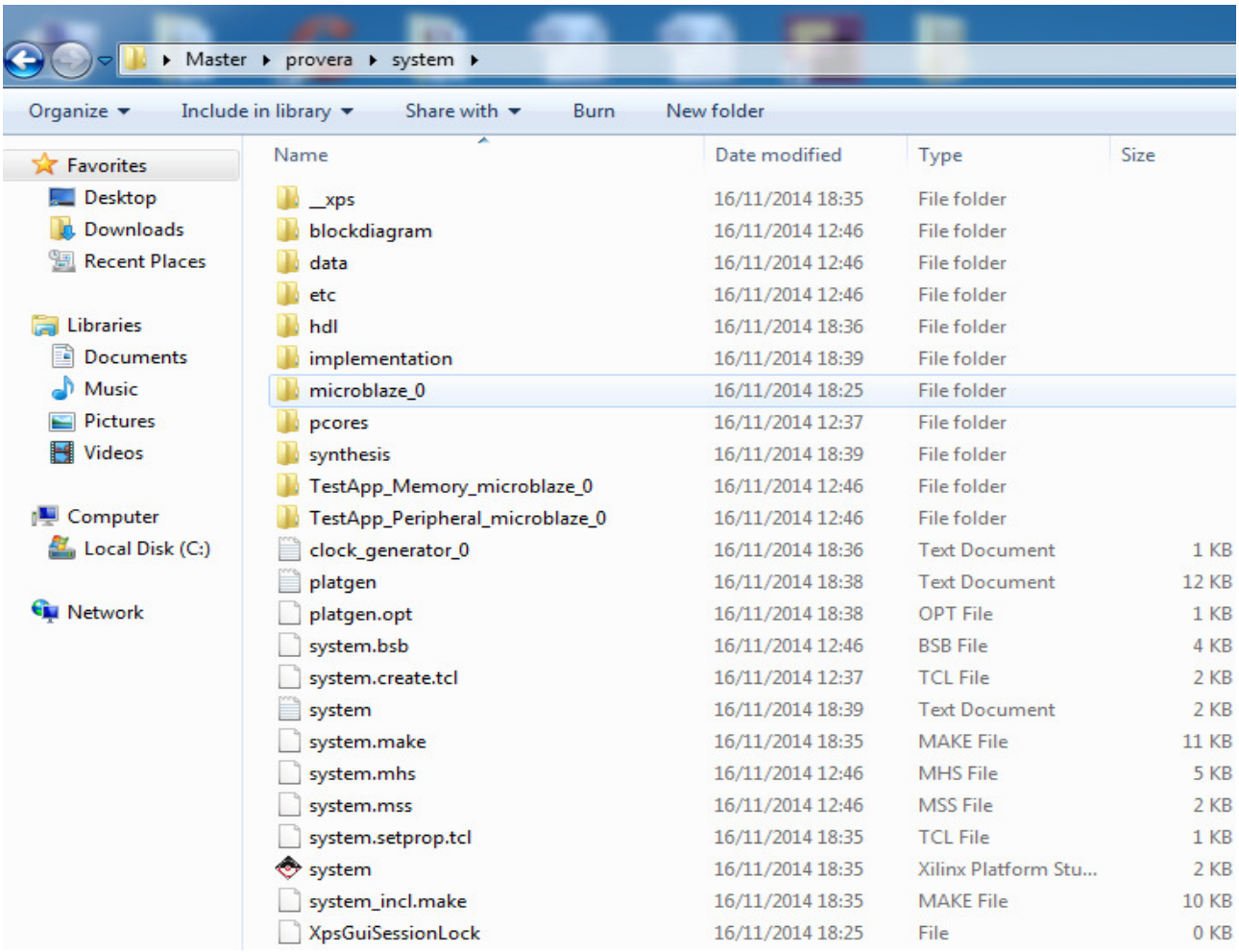
- system.mhs

  MHS fajl sadrži elemente sistema, njihove parametre, i konekcije u tekstualnom formatu. MHS fajl je hardverska osnova projekta.

- system.mss

  MSS fajl sadrži softverski deo dizajna, opisuje elemente sistema i različite softverske parametre vezane za periferije u tekstualnom formatu. MSS fajl je softverska osnova projekta.

MHS i MSS fajlovi su glavni proizvod razvijanog XPS dizajna. Kompletan hardver i softver sistema je sadržan u ovim fajlovima.



Slika 4.3.2.1. Pregled direktorijuma projekta

4.4. Pregled hardverske platforme

Hardverska platforma *embedded* sistema uključuje jedan ili više procesora, zajedno sa različitim brojem periferija i memorijskih blokova. Ovi IP blokovi koriste *interconnect* mrežu za komunikaciju. Portovi se povezuju na "spoljni svet", što može biti ostatak FPGA platforme, ili van FPGA platforme uopšte. Ponašanje svakog procesora ili periferije može biti podešeno. Od parametara implementacije zavise opcione karakteristike i specifikacija kako je hardverska platforma implementirana na FPGA ploči.

4.4.1. Razvoj hardverske platforme u Xilinx Platform Studio alatu

XPS obezbeđuje interaktivno razvojno okruženje koje omogućava da se specificiraju svi aspekti hardverske platforme. Kao što je već rečeno, opis hardvera je sadržan u MHS fajlu. MHS fajl, koji je tekstualni fajl koji se može menjati, je *source* fajl koji predstavlja hardversku komponentu *embedded* sistema. Tu su sadržane sve instance periferija zajedno sa njihovim parametrima, kao i konfiguracija *embedded* sistema uključujući arhitekturu magistrala, periferija, procesora, konekcija i adresnog prostora. Sadržaju MHS fajla se može prstupiti na sledeći način:

- Izabere se *Project* tab u *Project Information Area* u XPS
- U *Project Files* se dva puta klikne na MHS File: system.mhs
- Primera radi, izborom opcije *Edit* > *Find*, pretražiti *xps_uart16550*. U fajlu se nalazi više ovih instanci. Ukoliko se nalazi u okviru *Begin/End* para, onda se radi o portu koji je deo IP. Ukoliko se *PORT* nalazi pri vrhu MHS fajla, onda povezuje *embedded* sistem sa okruženjem.

4.4.2. Hardverska platforma u System Assembly View

System Assembly View u XPS alatu prikazuje IP instance hardverske platforme u obliku proširivog stabla ili u obliku tabele. IP elementi, njihovi portovi, podešavanja i parametri se mogu konfigurisati u *System Assembly View* i sve promene se zapisuju direktno u MHS fajl. Izmena imena porta ili nekog parametra se izvršava nakon pritiska dugmeta *Enter* ili klikom na OK. XPS sve modifikacije sistema automatski zapisuje u MHS fajl. Direktno menjanje MHS fajla nije preporučljivo, naročito ne za početnike.

i) Eksportovanje hardverske platforme

Kako je već rečeno, u EDK alatu hardverski deo sistema se razvija u XPS alatu, dok se softver razvija i debaguje u SDK alatu. SDK zahteva informacije u vezi sa hardverskom platformom, pa je neophodno eksportovati system.xml fajl. U ovom fajlu su informacije koje SDK zahteva kako bi bilo moguće razvijati softver na hardverskoj platformi koja je dizajnirana. To se vrši na sledeći način:

- U XPS alatu se izabere *Project* > *Export Hardware Design to SDK*
- Lokacija direktorijum je već izabrana i ne može biti promenjena kad se dizajn eksportuje iz XPS. Podrazumevana putanja je system/SDK/SDK_Export/hw u direktorijumu projekta.
- Izabere se opcija *Export Only*.

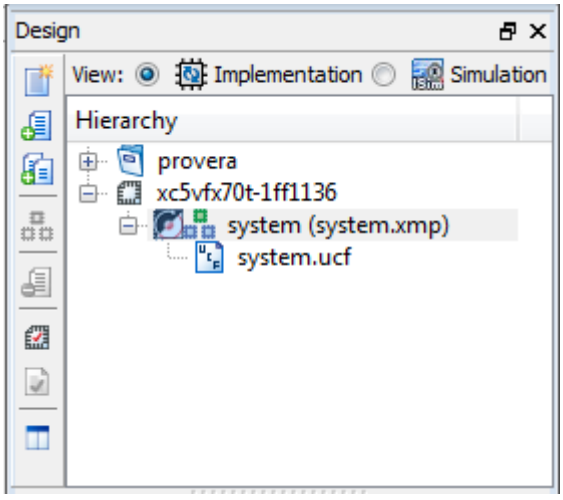
Vrlo je važno razumeti šta se dešava prilikom operacije eksportovanja. Kad se izabere opcija *Export Only*, program kreira nekoliko fajlova koji su korišćeni od strane SDK alata. Pored XML fajla, uključena je i dokumentacija softverskih drajvera i IP hardvera, tako da se njima može

pristupiti i iz SDK. Druga opcija, *Export&Launch SDK*, eksportuje dizajn i automatski otvara SDK gde se dalje razvija softverski deo projekta.

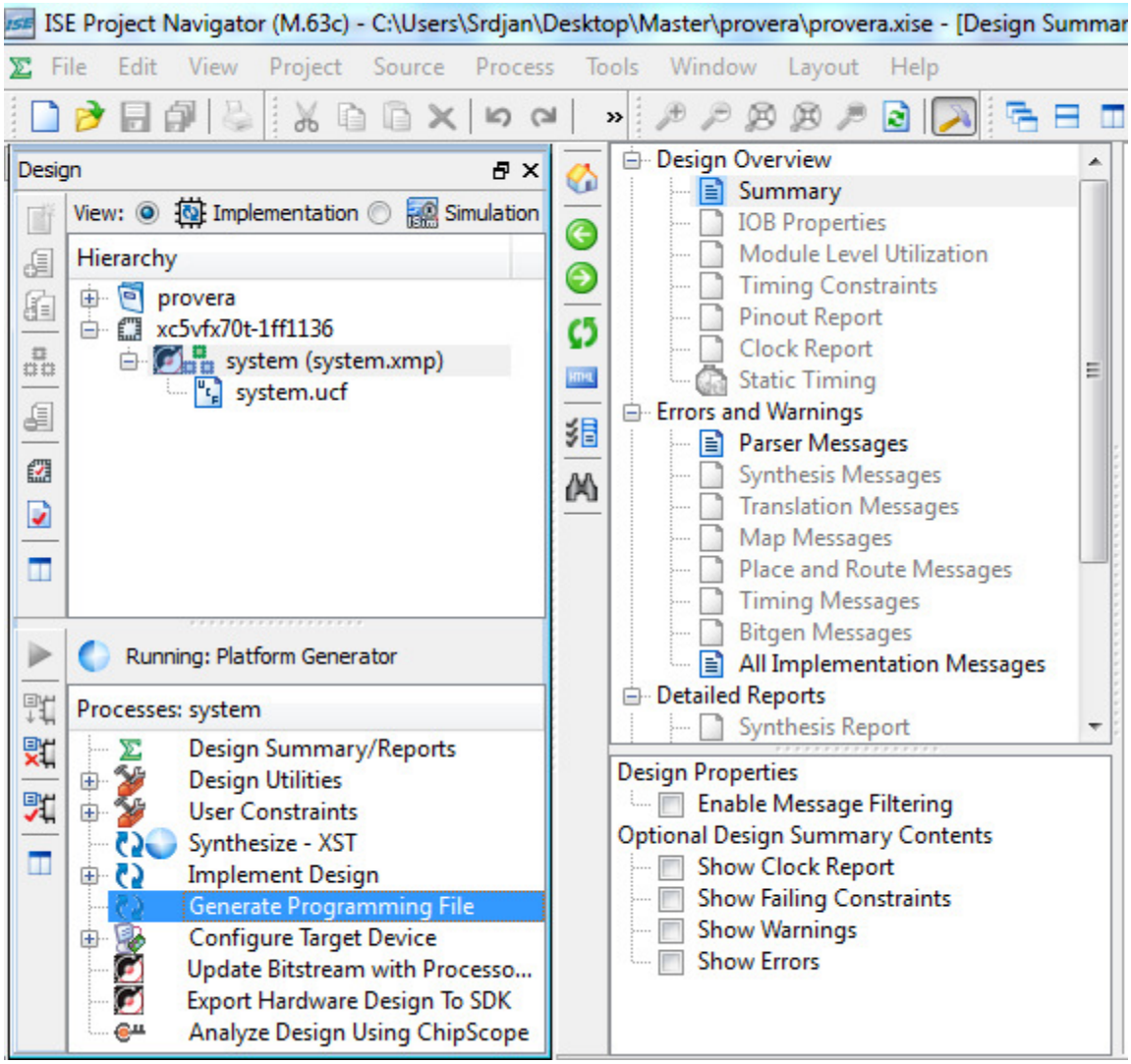
ii) Generisanje bitstream fajla

Nakon opisa hardvera *embedded* sistema u XPS alatu, koristi se ISE *Project Navigator* za implementaciju dizajna i generisanje *bitstream* fajla. C kod nije deo ovog bitstream fajla i on će biti dodat kasnije u SDK. Postupak je sledeći.

- Otvori se ISE *Project Navigator*
- U Project Navigator softveru se izabere *Project* > *Add Source* i izabere se *system/data/system.ucf* fajl. Ovaj korak je neophodan, jer *ISE Place and Route* alat zahteva informacije kao, na primer, na koje pinove se vezuju odgovarajuće periferije i slično.
- U dijalog prozoru se klikne OK
- Klikne se na *system.xmp* u *Design* prozoru
- U *Process: system* se klikne dva puta na *Generate Programming File*, kako bi se kreirao *bitstream* fajl. Za ovaj proces je potrebno nekoliko minuta, dok se ne pojavi poruka "*Process "Generate Programming File" completed successfully*".
- Generisani *bitstream* fajl se zove system.bit. Pored njega, tu je još i fajl edkBmmFile_bd.bmm, koji služi da SDK učita sadržaj instrukcijske (ROM) memorije na željenu ploču. Ovi fajlovi će biti korišćeni prilikom razvoja aplikacije.



Slika 4.4.2.1. System.ucf fajl dodat u projekat



Slika 4.4.2.2. Generisanje *bitstream* fajla

SDK omogućava razvoj aplikacije koja se izvršava u *embedded* sistemu. SDK je komplementarna sa XPS. Dizajn počinje kreiranjem softverskog projekta. SDK okruženje može upravljati sa više softverskih projekata. Kada se softverski projekat kreira, SDK zahteva da se kreira hardverska platforma i *board support package*. Hardverska platforma je dizajn *embedded* hardvera koji je kreiran u XPS. Ona uključuje XML, *bitstream* i BMM fajl. Kad se u SDK importuje XML fajl, tada ustvari importujemo hardversku platformu. U ISE *Design Suite 12* alatu, moguće je da više hardverskih platformi postoji u jednom radnom okruženju. O radu u SDK softveru, više reči će biti u narednom poglavlju, gde će biti objašnjeno kako se implementira željena softverska funkcionalnost u *embedded* sistemima.

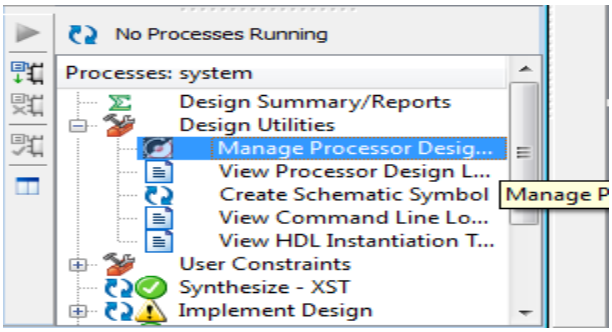
5. IMPLEMENTACIJA PROVERE I AŽURIRANJA IPV4 ZAGLAVLJA

Embedded sistemi se najčešće koriste za obavljanje nekih specifičnih zadataka. Prednost ovakve realzacije je što zauzimaju relativno malo resursa, imaju pristojne performanse i pružaju veliku fleksibilnost u smislu dodavanja novih i menjanja postojećih funkcionalnosti. U ovom radu je realizovana provera i ažuriranje IPv4 zaglavlja. Kao što je već pomenuto, ovo je funkcija koja se izvršava u ruterima i na osnovu koje utvrđujemo da li je došlo do greške prilikom prenosa paketa. Ova funkcionalnost je implementirana u posebnoj periferiji, koja je naknadno razvijena i dodata u postojeći projekat. Od ulazno/izlaznih periferija sadrži DIP svičeve i LED diode. Osim toga, ima i 12 registara koje služe za skladištenje odgovarajućih informacija. Ovde je dat pojednostavljen primer, čiji je cilj da se objasne mogućnosti koje EDK alat pruža, i način korišćenja istog. Izborom odgovarajuće kombinacije na DIP sviču, korisnik bira određeno zaglavlje. C aplikacija koje je razvijena u SDK, obrađuje to zaglavlje i pali odgovarajuću diodu zavisno da li je validno ili nije. Takođe, ukoliko je zaglavlje validno ono se ažurira i smešta u registre u ovoj periferiji. Na ploči ima 8 LED dioda, tako da 4 služe za brojanje korektnih zaglavlja, a 4 za brojanje pogrešnih.

5.1. *Create and Import Peripheral* vizard

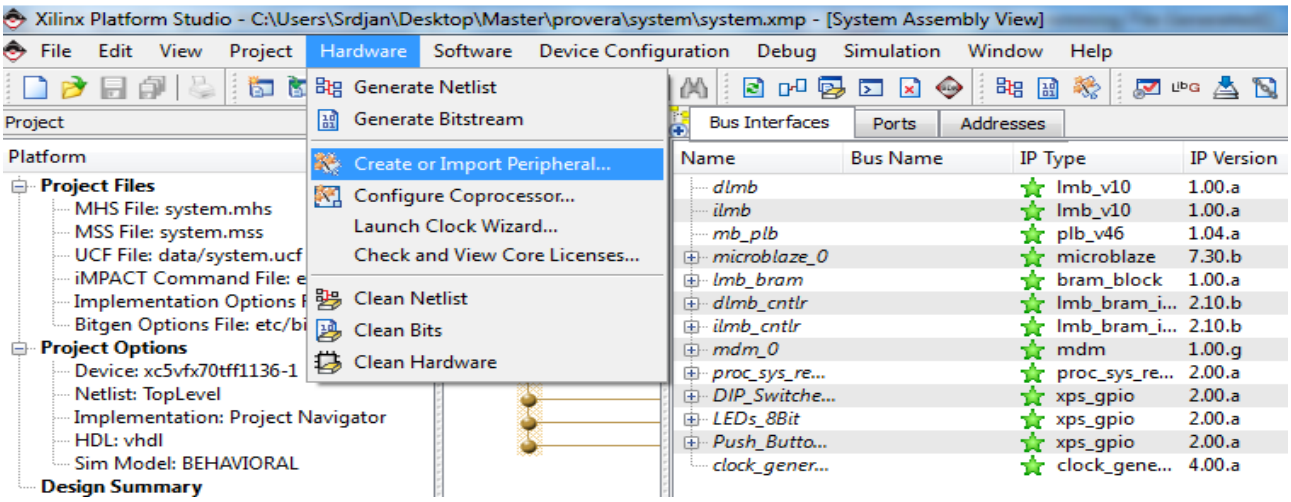
XPS značajno olakšava dizajn *embedded* sistema jer automatizuje dosta operacija, i svodi ih na seriju odgovarajućih selekcija. Prilagođavanje dizajna može biti jednostavno poput podešavanja par parametara na nekom od IP jezgara ili kompleksnije poput razvoja potpuno novog IP jezgra i integrisanja u postojeći dizajn. Za to se koristi CIP vizard koji pruža slične mogućnosti kao i BSB. On obezbeđuje radni okvir za dizajn, uključujući logiku za interfejse magistrale kao i HDL templejt fajl tako da je znatno olakšano integrisanje logike koju korisnik razvija. Svi fajlovi koji su neophodni za uključivanje korisničkog IP jezgra (*pcore*) u *embedded* sistem su obezbeđeni od strane CIP vizarda. U ovom poglavlju je objašnjeno šta se dešava tokom razvoja korisnički definisanog IP jezgra. Postupak je sledeći:

- Pokrene se ISE *Project Navigator*, otvori se prethodni projekat, klikne na system.xmp, i dvostruki klik na *Manage Processor Design* (locirano u okviru *Design Utilities*) slika 5.1.1, kako bi se pristupilo XPS alatu.



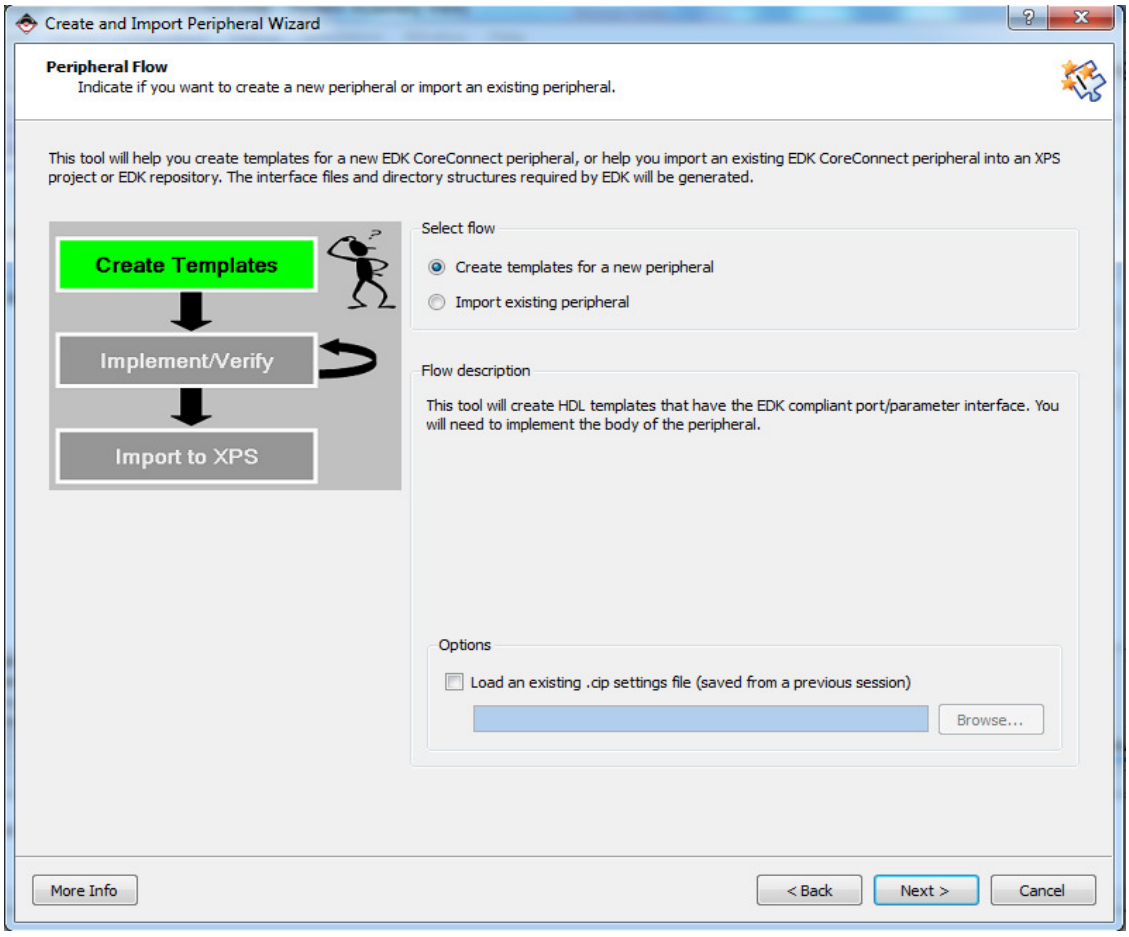
Slika 5.1.1. Pokretanje XPS alata iz ISE softvera

- U XPS alatu se otvori *Hardware* > *Create or Import Peripheral*, slika 5.1.2.



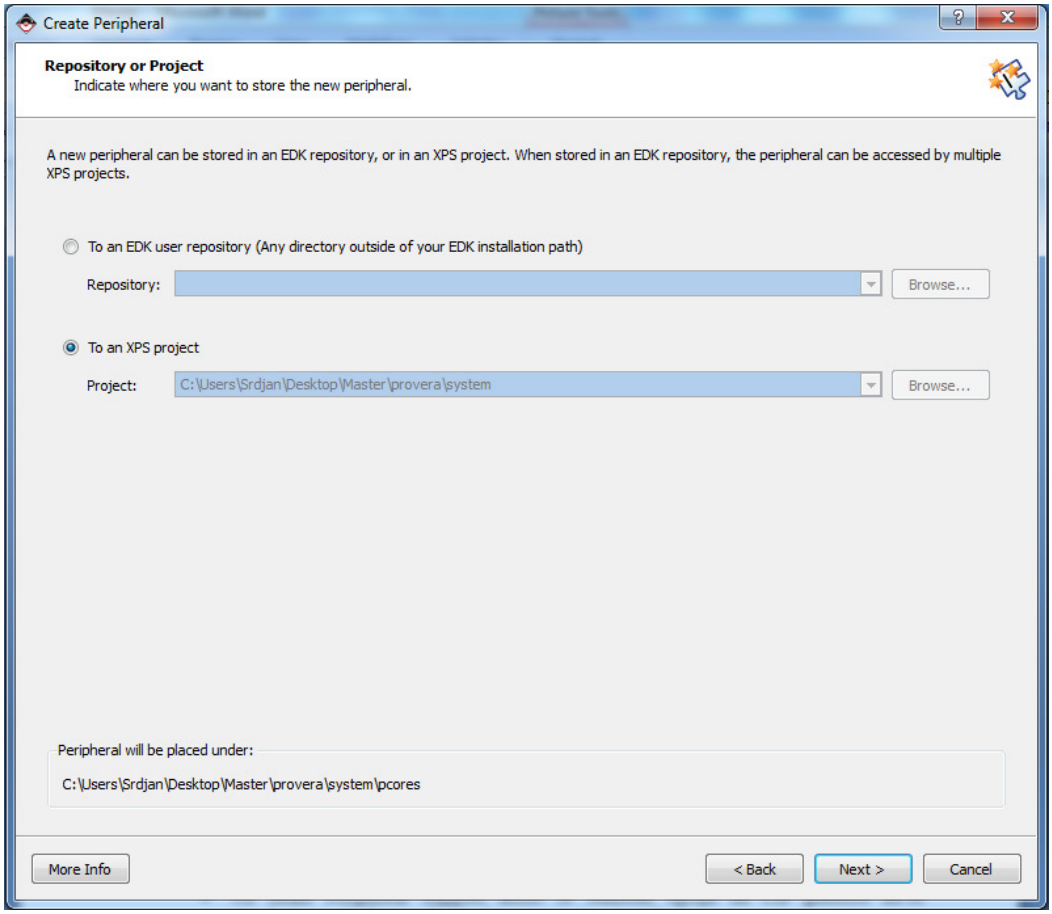
Slika 5.1.2. Opcija za kreiranje ili importovanje periferije

- Otvoriće se *Peripheral Flow* stranica, slika 5.1.3. Ovde se može izabrati da li će se kreirati nova ili importovati postojeća periferija. Odabere se opcija *Create templates for a new peripheral*.



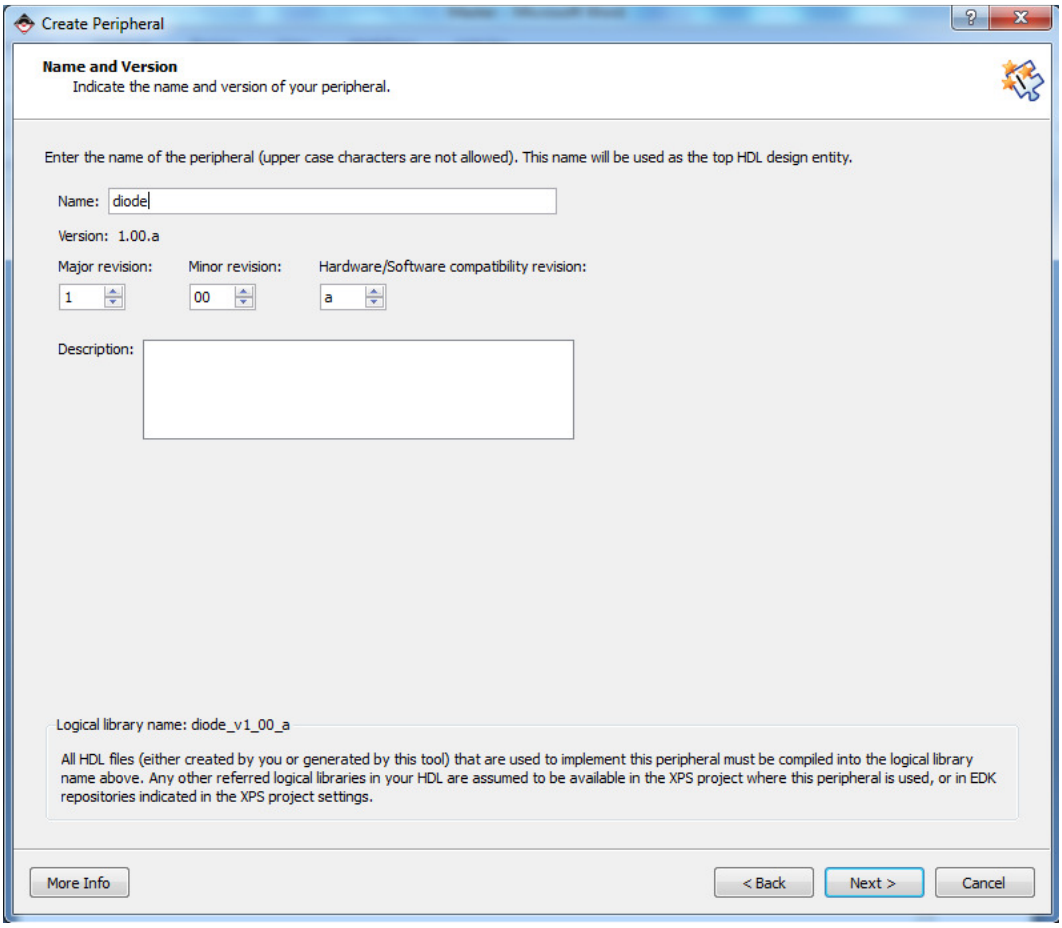
Slika 5.1.3. *Peripheral Flow* strana

- Na *Repository or Project* stranici, slika 5.1.4, treba odabrati gde će se sačuvati fajlovi periferije. Ukoliko se periferija koristi u više projekata treba odabrati opciju *To an EDK user repository*. Na ovaj način više različitih sistema može da importuje datu periferiju. Kako se u ovom primeru radi sa *single embedded* projektom treba odabrati opciju *To an XPS project*. Pošto smo CIP vizard pokrenuli iz XPS, lokacija direktorijuma je popunjena automatski. Kliknuti *Next.*



Slika 5.1.4. *Repository or Project* strana

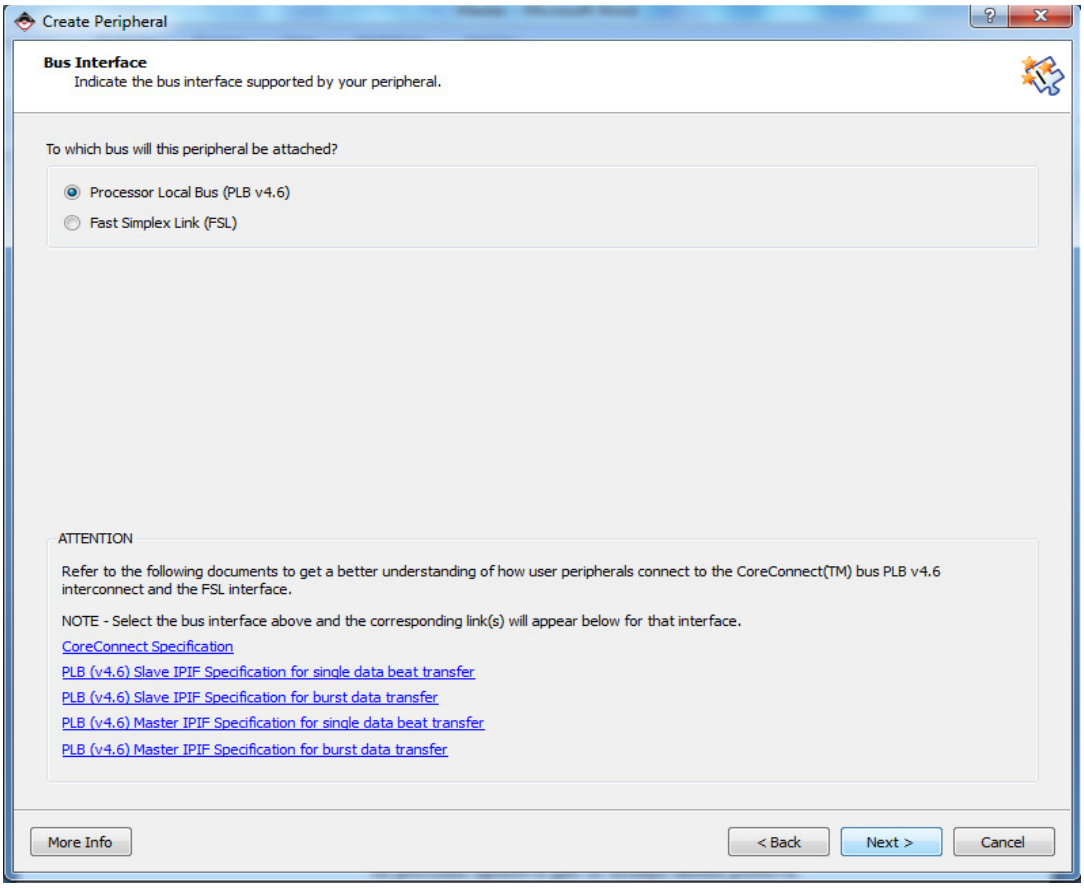
- Na *Name and Version* strani, slika 5.1.5 treba upisati ime i verziju periferije. U ovom primeru koristiće se ime *diode*. Podešavanja za verziju su ostavljena sa podrazumevanim vrednostima. Može se dodati i opis periferije u *Description* polju.



Slika 5.1.5. ***Name and Version*** **strana**

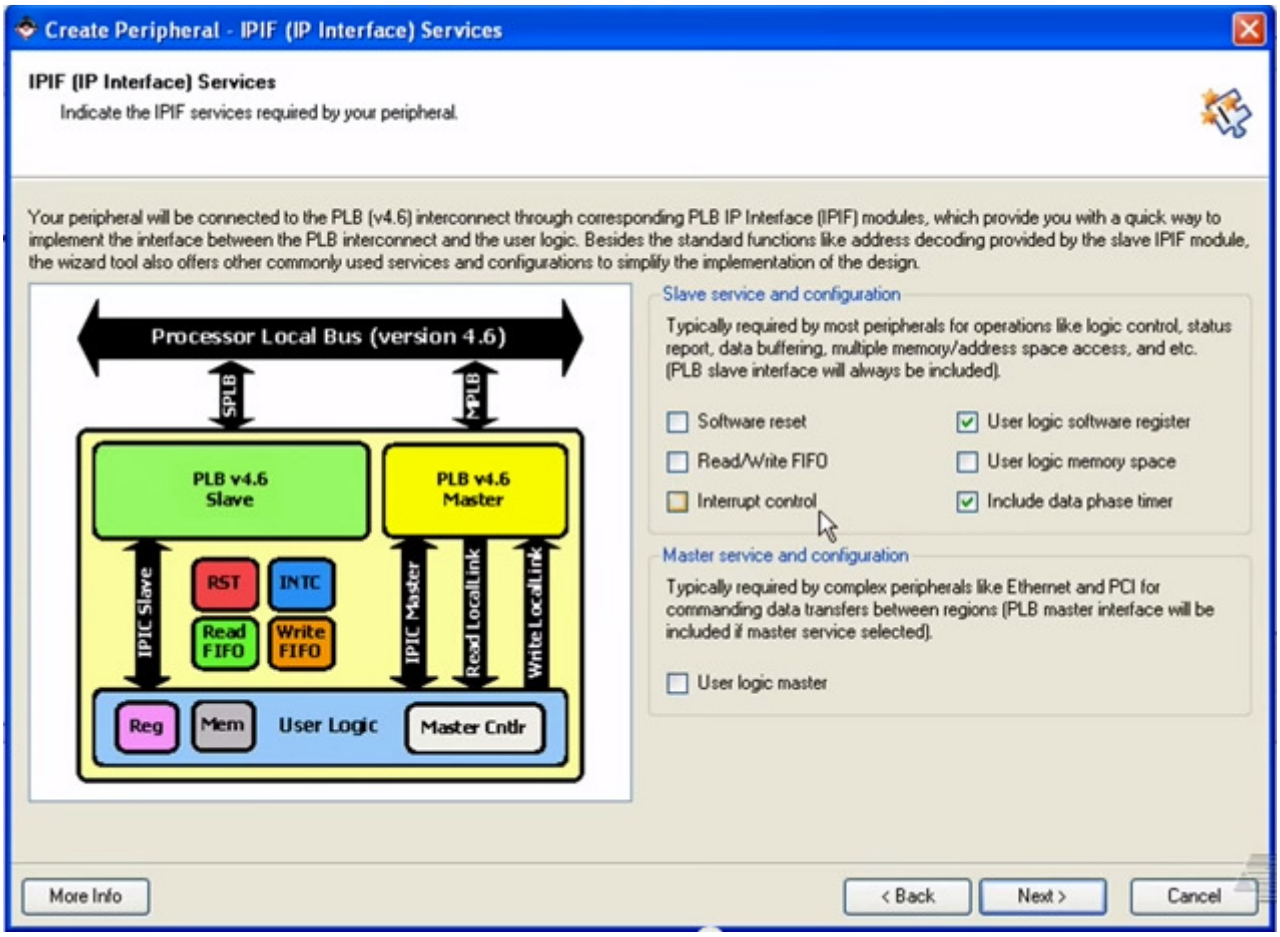
- Na *Bus Interface* strani, slika 5.1.6 bira se procesorska magistrala na koju se povezuje periferija koju razvijamo. Na raspolaganju su:
  - *Processor Local Bus* (PLBv46) omogućava *high-speed* interfejs između procesora i periferije. Koristi se i kod *MicroBlaze* i kod *PowerPC* procesora.
  - *Fast Simplex Link* (FSL) je *point-to-point* FIFO interfejs. Može se koristiti sa *MicroBlaze* procesorom, dok se najčešće ne koristi sa *PowerPC* procesorima.

U ovom primeru se koristi PLB v46. Kliknuti *Next*.



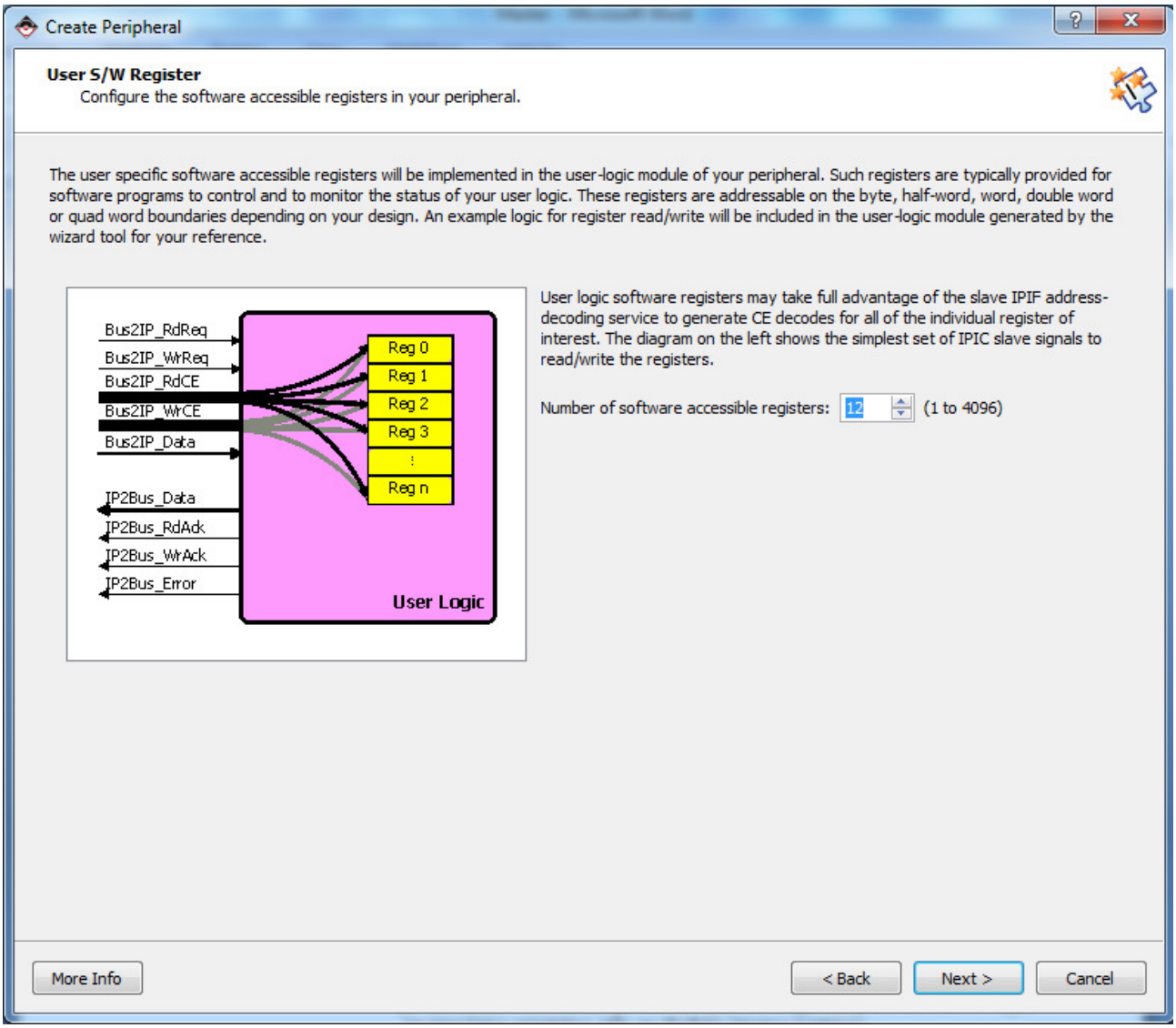
Slika 5.1.6. ***Bus Interface*** **strana**

- Na IPIF strani, slika 5.1.7 su opcije koje su na raspolaganju korisniku, zavisno da li je nova periferija definisana kao *master* ili *slave*. Dodatno objašnjenje za svaku od ovih opcija može se dobiti klikom na *More Info.* U ovom primeru će se koristiti *User logic software register* i *Include data phase timer.User logic software regiser* su registri koji su dostupni koristeći *memory map* adrese u softverskom kodu. Registri su nam neophodni za smeštanje zaglavlja, ali i za prosleđivanje informacija koje diode treba upaliti. *Include data phase timer* je *time-out* funkcionalnost koju koriste sve periferije u dizajnu. *Software reset* blok daje mogućnost resetovanja kompletne periferije softverski. *Read/write FIFO* daje mogućnost implementacije *Read/Write FIFO* funkcionalnosti ka i od magistrale. *Interrupt control* opcija omogućava periferiji da generiše *interrupt* signale koji bi bili obrađeni u procesoru. U ovom primeru nam neće biti neophodne ove opcije.



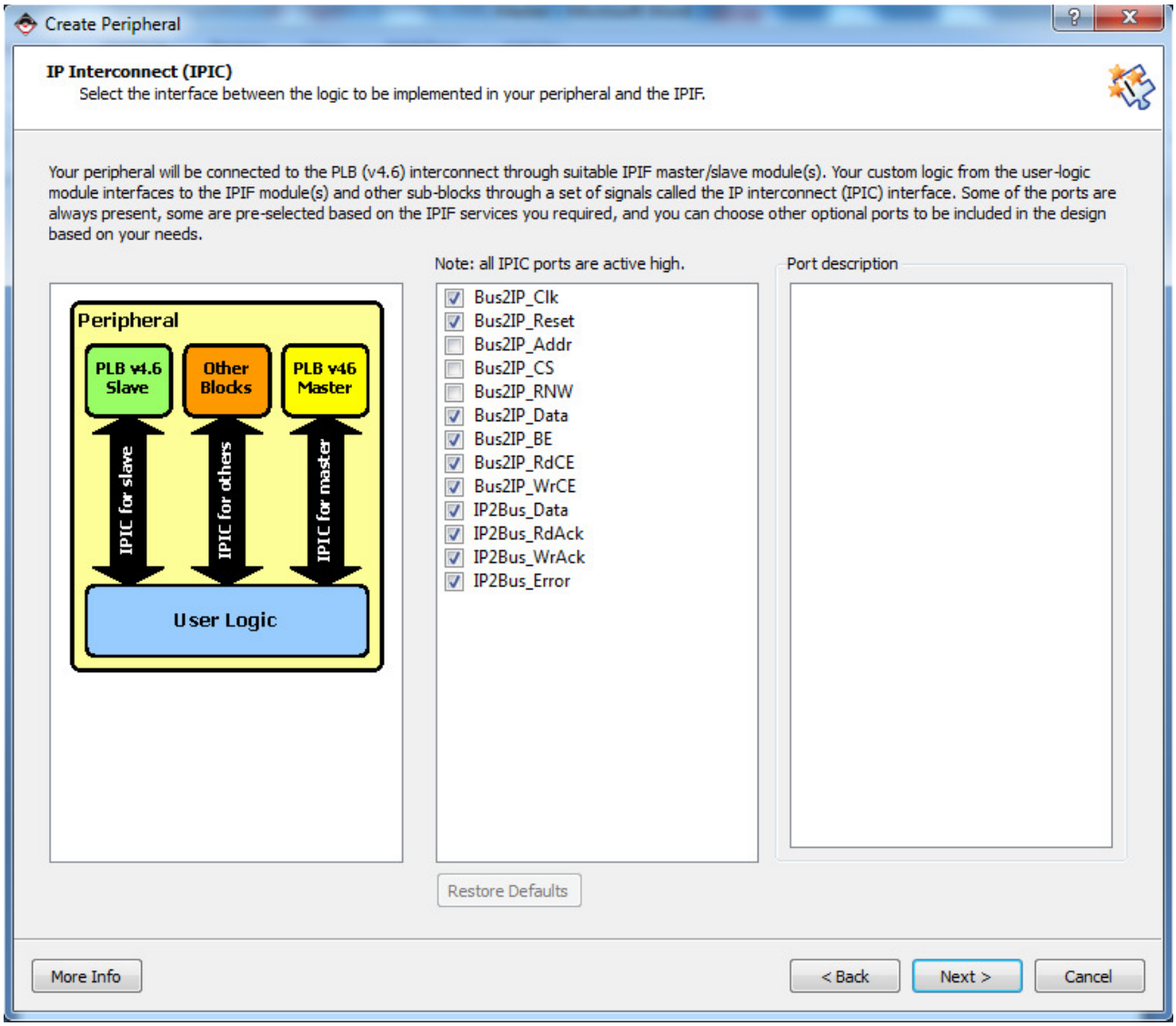
Slika 5.1.7. ***IPIF Services*** **strana**

- Na strani *User S/W registers*, slika 5.1.8 treba odabrati koliko registara se koristi. U ovom primeru se koristi 12 (maksimum je 4096).



Slika 5.1.8. ***User S/W Register*** **strana**

- Na *IP Interconnect* strani, slika 5.1.9 je spisak IPIC signala koje CIP vizard nudi. Objašnjenja za ove signale se mogu pogledati ako se neki od njih odabere. Ostaviti podrazumevane vrednosti i kliknuti *Next*.



Slika 5.1.9. *IP Interconnect* strana

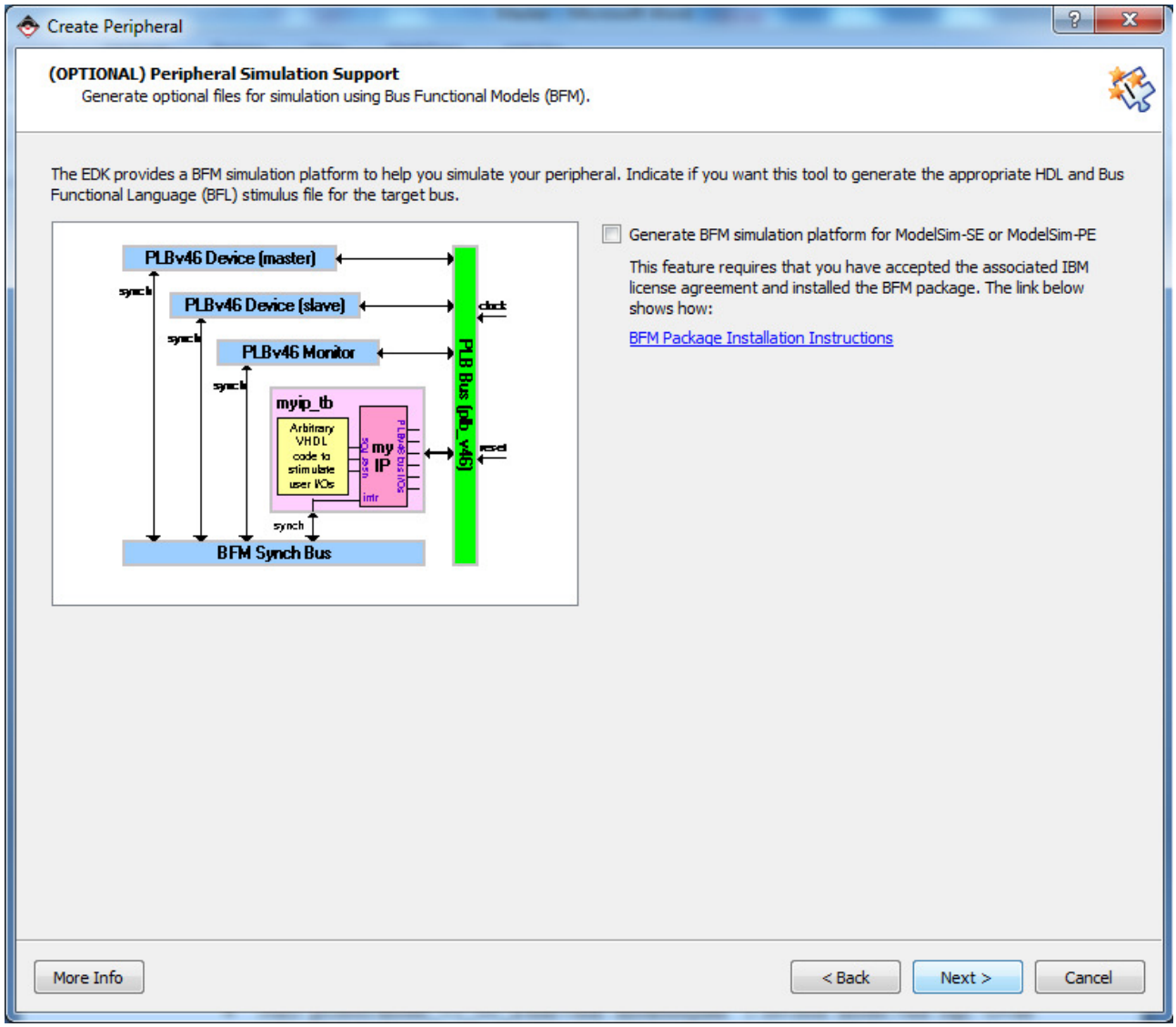
- Na strani *Peripheral Simulation Support,* slika 5.1.10 može se odabrati opcija da CIP generiše BFM simulacionu platformu. Prethodno je potrebno instalirati:
  - BFM simulacioni paket za EDK
  - *ModelSim-SE* ili *Model Sim-PE*

U ovom primeru se neće koristiti ova opcija, tako da samo treba kliknuti *Next*.

CIP vizard kreira dva HDL fajla koji implementiraju radni okvir za *pcore*.

- diode.vhd fajl, koji sadrži PLBv46 *bus* interfejs logiku. Kako će u ovom projektu portovi da se konektuju na "spoljašnji svet", potrebno je modifikovati imena portova i dati im odgovarajuća imena (primer dat u poglavlju 5.2 jasno ilustruje navedeno). U fajlu su precizna uputstva gde se dodaju imena portova.

– user_logic.vhd fajl, koji je templejt fajl gde se dodaje logika koju periferija obavlja. Iako je uvek moguće kreirati dodatne *source* fajlove, u ovom primeru je dovoljan samo user_logic.vhd fajl.

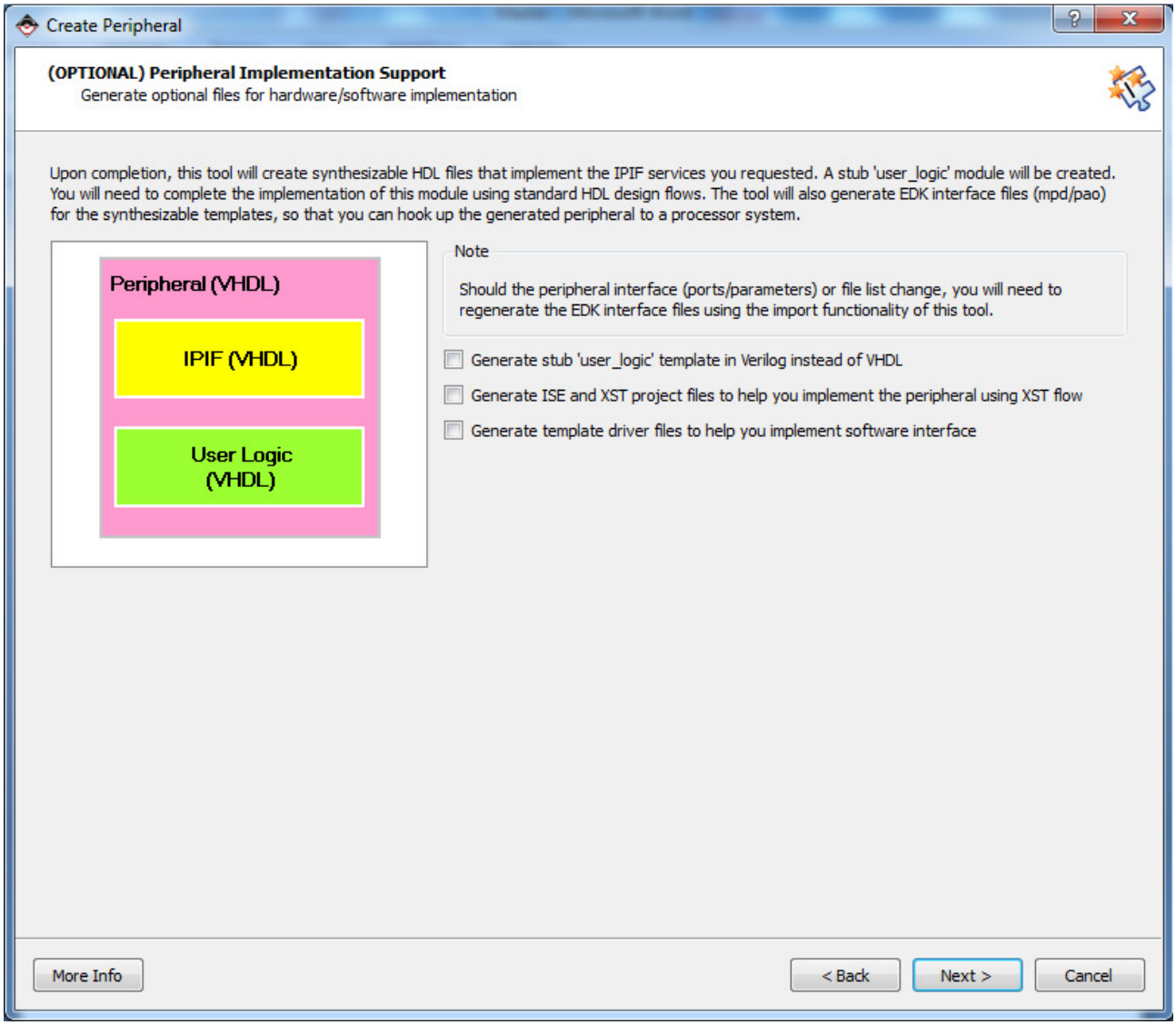


Slika 5.1.10. ***Peripheral Simulation Support*** **strana**

- *Peripheral Implementation Support* strana, slika 5.1.11 nudi tri opcije za kreiranje opcionih fajlova za hardversku i softversku implementaciju.
  - Prva opcija je da se user_logic kreira u Verilogu umesto u VHDL.
  - Ukoliko je potrebno da se *pcore* dizajn implementira u potpunosti (za vremenske analize i simulacije), kliknuti na opciju *Generate ISE and XST files to help you implement the peripheral using XST flow*. CIP vizard kreira potrebne ISE *project* fajlove. Ukoliko je periferija *low-speed*, ovo nije neophodno.

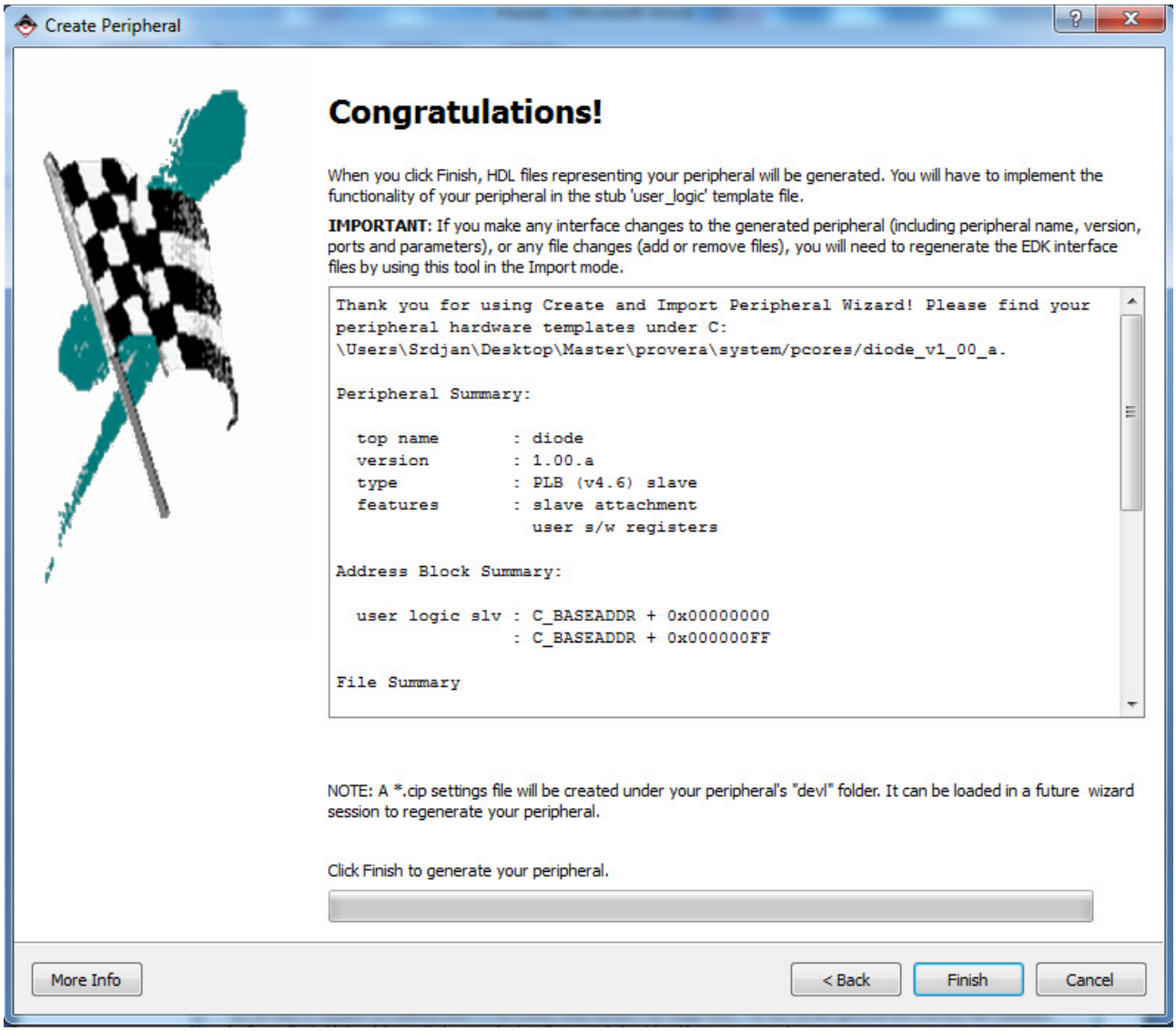
– Ukoliko periferija zahteva kompleksnije softverske drajvere, klinkuti na *Generate template driver files to help you impement software interface.*

U ovom primeru nećemo odabrati nijednu od ovih opcija. Kliknuti *Next*.



Slika 5.1.11. ***Peripheral Implementation Support*** **strana**

Kad završimo sa podešavanjima, dobićemo *Summary report* stranu, slika 5.1.12. Kliknuti *Finish* da bi se završio proces generisanja periferije.



Slika 5.1.12. ***Summary report*** **strana**

5.2. Modifikacija CIP templejt fajla

Sledeći korak jeste modifikacija diode.vhd i user_logic.vhd fajlova, kako bi se implementirala odgovarajuća funkcionalnost.

- U XPS alatu, kliknuti *File* > *Open*
- Naći *pcores/diode_v1_00_a/hdl/vhdl* direktorijum i otvoriti diode.vhd fajl. Ovde treba dodati ulazne i izlazne portove na dva mesta:
  - Deklaracija portova top level entiteta, slika 5.2.1
  - Mapa portova za instancu user_logic, slika 5.2.2

Skrolovati do linije 165 otprilike. U delu predviđenom za to treba ubaciti uokvireni kod sa slike.

```
162   port
163   (
164     -- ADD USER PORTS BELOW THIS LINE ------------------
165     DIP_switch                          : in std_logic_vector(7 downto 0);
166     LEDs                                : out std_logic_vector(7 downto 0);
167     --USER ports added here
168     -- ADD USER PORTS ABOVE THIS LINE ------------------
```

Slika 5.2.1. Dodavanje portova u diode.vhd fajlu

Takođe, skrolovati kursor do linije 390 otprilike. U odgovarajućem delu, ubaciti uokvireni deo koda.

```
390     port map
391     (
392       -- MAP USER PORTS BELOW THIS LINE ------------------
393       LEDs(7 downto 0)                  => LEDs(7 downto 0),
394       DIP_switch(7 downto 0)            => DIP_switch(7 downto 0),
395       --USER ports mapped here
396       -- MAP USER PORTS ABOVE THIS LINE ------------------
```

Slika 5.2.2. Mapiranje portova u diode.vhd fajlu

Sačuvati promene u fajlu. Dodavanje korisničkih portova i njihovo mapiranje su najčešće izmene koje se vrše u *<ip core name>.vhd* fajlu.

Na isti način otvoriti i user_logic.vhd fajl. Skrolovati do linije 99 otprilike, i ubaciti uokvireni kod sa slike 5.2.3 (ubacivanje deklaracije portova u deo gde se deklarišu portovi entiteta).

```
97    port
98    (
99      -- ADD USER PORTS BELOW THIS LINE ------------------
100     DIP_switch                          : in std_logic_vector(7 downto 0);
101     LEDs                                : out std_logic_vector(7 downto 0);
102     --USER ports added here
103     -- ADD USER PORTS ABOVE THIS LINE ------------------
```

Slika 5.2.3. Dodavanje portova u user_logic.vhd fajlu

Dalje, u delu za deklaraciju signala, oko linije 132, ubaciti sledeće signale:

```
signal led_out : std_logic_vector(7 downto 0);

signal hdr : std_logic_vector(159 downto 0);

signal hdr1: std_logic_vector(159 downto
0):="01000101000000000000000000011000001000100001000100100000000000001000000000
00110000000000000000010001111011111110001100110101111101011100010010000011111000
01011";

signal      hdr2                                                    :
std_logic_vector(159 downto
0):="01100101000000000000000000000100000000000110000010001000100010010000000000
00110000000000000000010001111011111110001100110101111101011100010010000011111000
01011";

signal hdr3                                                         :
std_logic_vector(159 downto
0):="01000101000000000000000000111001100000000000000000100000000000000010000000000
10001101110000110000111000000101010000000000000000011100000010101000000000000110
00111";

signal hdr4                                                         :
std_logic_vector(159 downto
0):="01100101000000000000000000111001100000000000000000100000000000000010000000000
10001101110000110000111000000101010000000000000000011100000010101000000000000110
00111";

signal hdr5                                                         :
std_logic_vector(159 downto
0):="01000101000000000000000000011110000011100010001100100000000000000010000000000
00110101100011110011010101111000100000001010011000111010111100010000000010100000
01100
";
```

Signal *led_out* predstavlja vrednost na osnovu koje se pale odgovarajuće diode. Vrednost signala *header* zavisi od kombinacije na DIP sviču, i omogućava procesoru iščitavanje željene vrednosti. Ostali signali (*hdr1* do *hdr5*) su vrednosti zaglavlja koje ćemo koristiti u testiranju (u finalnoj implementaciji bi sadržaj zaglavlja bio upisan u registar zaglavlja iz ravni podataka). Na osnovu vrednosti prekidača na DIP sviču, u registar zaglavlja se upisuje sadržaj jednog od ovih pet signala. U ovom delu, korisnik može ubaciti i druga zaglavlja koja želi da testira. U okviru user_logic.vhd fajla, definišu se i dva procesa. Prvi proces služi za iščitavanje podataka iz registara u zavisnosti od kombinacije na DIP sviču. Kako bi se kod uprostio, vrednost zaglavlja se dodeljuje signalu *hdr*. Na ovaj način će kod u procesu koji reguliše iščitavanje zaglavlja biti znatno jednostavniji nego da se u njemu vrši provera stanja na DIP sviču.

```
decode1: process(BUS2IP_Clk, Bus2IP_Reset, DIP_switch)
      begin
      if (BUS2IP_Clk 'event and BUS2IP_Clk = '1' and Bus2IP_Reset = '0') then
            if(DIP_switch="00000001")then
                  hdr(31 downto 0)<=hdr2(31 downto 0);
                  hdr(63 downto 32)<=hdr2(63 downto 32);
                  hdr(95 downto 64)<=hdr2(95 downto 64);
                  hdr(127 downto 96)<=hdr2(127 downto 96);
                  hdr(159 downto 128)<=hdr2(159 downto 128);
            elsif (DIP_switch="00000010")then
                  hdr(31 downto 0)<=hdr1(31 downto 0);
```

```
                hdr(63 downto 32)<=hdr1(63 downto 32);
                hdr(95 downto 64)<=hdr1(95 downto 64);
                hdr(127 downto 96)<=hdr1(127 downto 96);
                hdr(159 downto 128)<=hdr1(159 downto 128);
          elsif (DIP_switch="00000100")then
                hdr(31 downto 0)<=hdr3(31 downto 0);
                hdr(63 downto 32)<=hdr3(63 downto 32);
                hdr(95 downto 64)<=hdr3(95 downto 64);
                hdr(127 downto 96)<=hdr3(127 downto 96);
                hdr(159 downto 128)<=hdr3(159 downto 128);
          elsif (DIP_switch="00001000")then
                hdr(31 downto 0)<=hdr4(31 downto 0);
                hdr(63 downto 32)<=hdr4(63 downto 32);
                hdr(95 downto 64)<=hdr4(95 downto 64);
                hdr(127 downto 96)<=hdr4(127 downto 96);
                hdr(159 downto 128)<=hdr4(159 downto 128);
          elsif (DIP_switch="00010000")then
                hdr(31 downto 0)<=hdr5(31 downto 0);
                hdr(63 downto 32)<=hdr5(63 downto 32);
                hdr(95 downto 64)<=hdr5(95 downto 64);
                hdr(127 downto 96)<=hdr5(127 downto 96);
                hdr(159 downto 128)<=hdr5(159 downto 128);
          else
                hdr(159 downto 0)<=(others=>'1');
          end if;
      end if;
      end process decode1;
```

Drugi proces reguliše paljenje dioda. Četiri diode služe za prikaz broja validnih zaglavlja, a druge četiri za prikaz broja pogrešnih zaglavlja. Signal *leds_out* upravlja diodama, pa njegova četiri bita se iščitavaju iz *slv_reg0* (u kome su smešteni podaci o broju validnih zaglavlja), a druga četiri iz *slv_reg1* (u kome su smešteni podaci o broju pogrešnih zaglavlja). Prethodno procesor upisuje odgovarajuće vrednosti u ove registre. Vrednost signala *leds_out* se zatim dodeljuje izlaznom portu *LEDs* koji je vezan na pinove LED dioda (ovo će biti objašnjeno u odeljku 5.3.1).

```
decode2: process(BUS2IP_Clk, Bus2IP_Reset)
  begin
    if (BUS2IP_Clk 'event and BUS2IP_Clk = '1' and Bus2IP_Reset = '0') then
      led_out(3 downto 0) <= slv_reg0(28 to 31);
      led_out(7 downto 4) <= slv_reg1(28 to 31);
      LEDs <=led_out;
    end if;
  end process decode2;
```

Promene treba napraviti i u delu kojim se reguliše isčitavanje registara. Prikazani kod na slici 5.2.4 prikazuje registre vidljive procesoru. Promene postoje samo u delu vezanom za registre *slv_reg2* do *slv_reg6* jer procesor iz njih iščitavaju zaglavlja. U procesu *decode1* je definisano da se na osnovu stanja na DIP sviču podaci upisuju u signal *hdr*. Ovde se ti podaci koje sadrži signal *hdr* prosleđuju procesoru na obradu. Pronaći *SLAVE_REG_READ_PROC*, i napraviti izmene prikazane na slici 5.2.4.

```
338     case slv_reg_read_sel is
339       when "100000000000" => slv_ip2bus_data <= slv_reg0;
340       when "010000000000" => slv_ip2bus_data <= slv_reg1;
341       when "001000000000" => slv_ip2bus_data <= hdr(31 downto 0);
342       when "000100000000" => slv_ip2bus_data <= hdr(63 downto 32);
343       when "000010000000" => slv_ip2bus_data <= hdr(95 downto 64);
344       when "000001000000" => slv_ip2bus_data <= hdr(127 downto 96);
345       when "000000100000" => slv_ip2bus_data <= hdr(159 downto 128);
346       when "000000010000" => slv_ip2bus_data <= slv_reg7;
347       when "000000001000" => slv_ip2bus_data <= slv_reg8;
348       when "000000000100" => slv_ip2bus_data <= slv_reg9;
349       when "000000000010" => slv_ip2bus_data <= slv_reg10;
350       when "000000000001" => slv_ip2bus_data <= slv_reg11;
351       when others => slv_ip2bus_data <= (others => '0');
352     end case;
353
354   end process SLAVE_REG_READ_PROC;
```

Slika 5.2.4. Izmene u kodu u user_logic.vhd fajlu

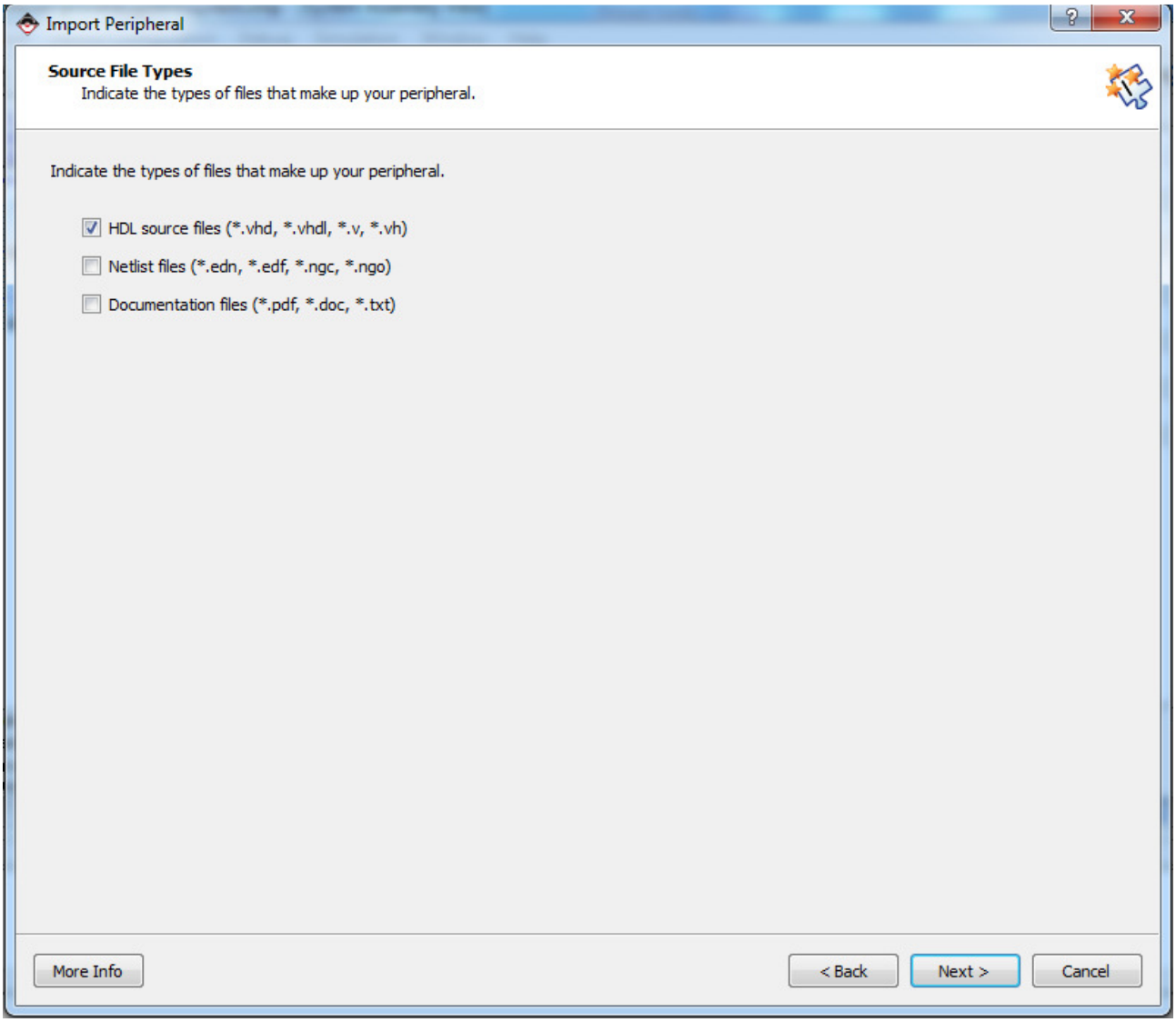
5.3. Dodavanje kreirane periferije u *Processor System*

Modifikovani su diode.vhd i user_logic.vhd fajlovi dodavanjem novih portova. Svaki put kada se promene ovi fajlovi u smislu da se menjaju portovi ili parametri u MPD fajlu, CIP vizard mora da se reimportuje. Tako se regenerišu PSF, MPD i PAO fajlovi, koji su interfejs fajlovi ka EDK. Kada se importovanje završi, nova periferija može biti dodata u *embedded* dizajn. Pre nego što se izvrši reimportovanje periferije, mali podsetnik šta je do sada urađeno:

- Prvi put kad je pokrenut CIP vizard, kreirana je periferija *diode*, podešeni *bus* interfejsi i generisani zahtevani templejt fajlovi.
- Sada treba dodati *diode* u postojeći projekat, opet koristeći CIP vizard. Periferija *Diode* je importovana u XPS-odgovarajući direktorijum, a CIP vizard kreira MPD i PAO fajlove.

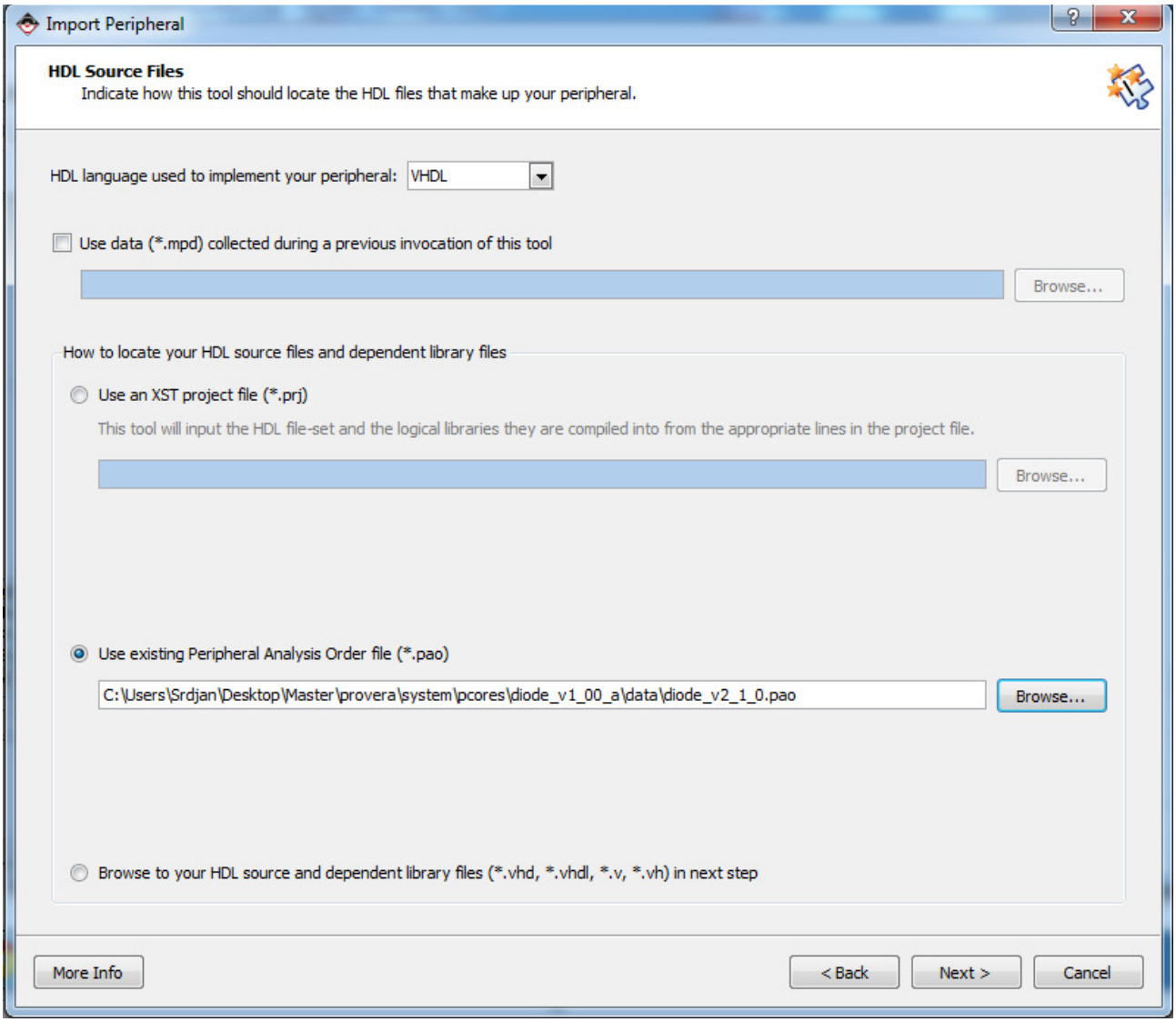
Postupak je sledeći:

- Otvoriti CIP vizard i čekirati opciju *import an existing peripheral to an XPS project*.
- Na *Repository or Project* strani, odabrati postojeći projekat.
- Na strani *Name and Version*, izabrati diode, sa *Name drop-down* liste. Verzija se ne zahteva, ali je u ovom primeru već odabrana.
- Ako CIP vizard pita da li da izmeni postojeću periferiju sa ovim imenom, kliknuti *Yes*.
- Odabrati HDL *source* fajl, slika 5.3.1. Kliknuti *Next*.



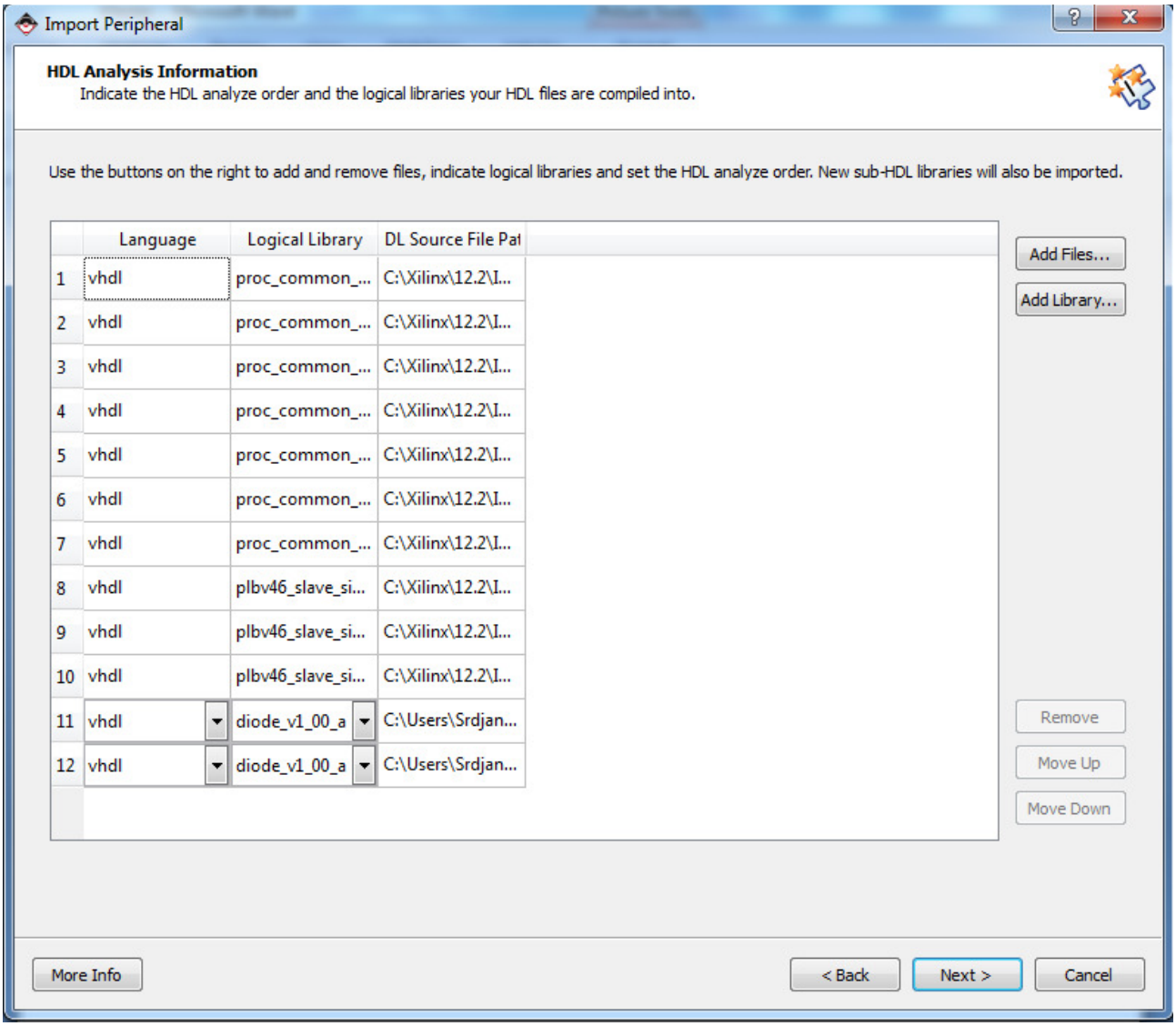
Slika 5.3.1. ***Source File Types*** **strana**

- CIP vizard importuje *pcores* koji su bili kreirani na različite načine. Ukoliko je CIP vizard korišćen da se kreira periferija, najbolji način da se pravilno lociraju i identifikuju izvorišni fajlovi je da se koristi *Peripheral Analysis Order* (PAO) fajl. Odabrati opciju *Use existing Peripheral Analysis Order* fajl (*.pao), slika 5.3.2.
- Pronaći odgovarajući PAO fajl. On je lociran u *diode/data* subdirektorijumu. Kada se koristi PAO fajl da se lociraju potrebni *source* fajlovi, nije neophodno dodavati nikakve dodatne fajlove ili biblioteke. Ukoliko se importuje kompleksnija periferija koja sadrži više fajlova, treba pogledati listing putanja biblioteka i HDL *source* fajlova, da bi se utvrdilo da li su svi potrebni fajlovi i biblioteke uključene.



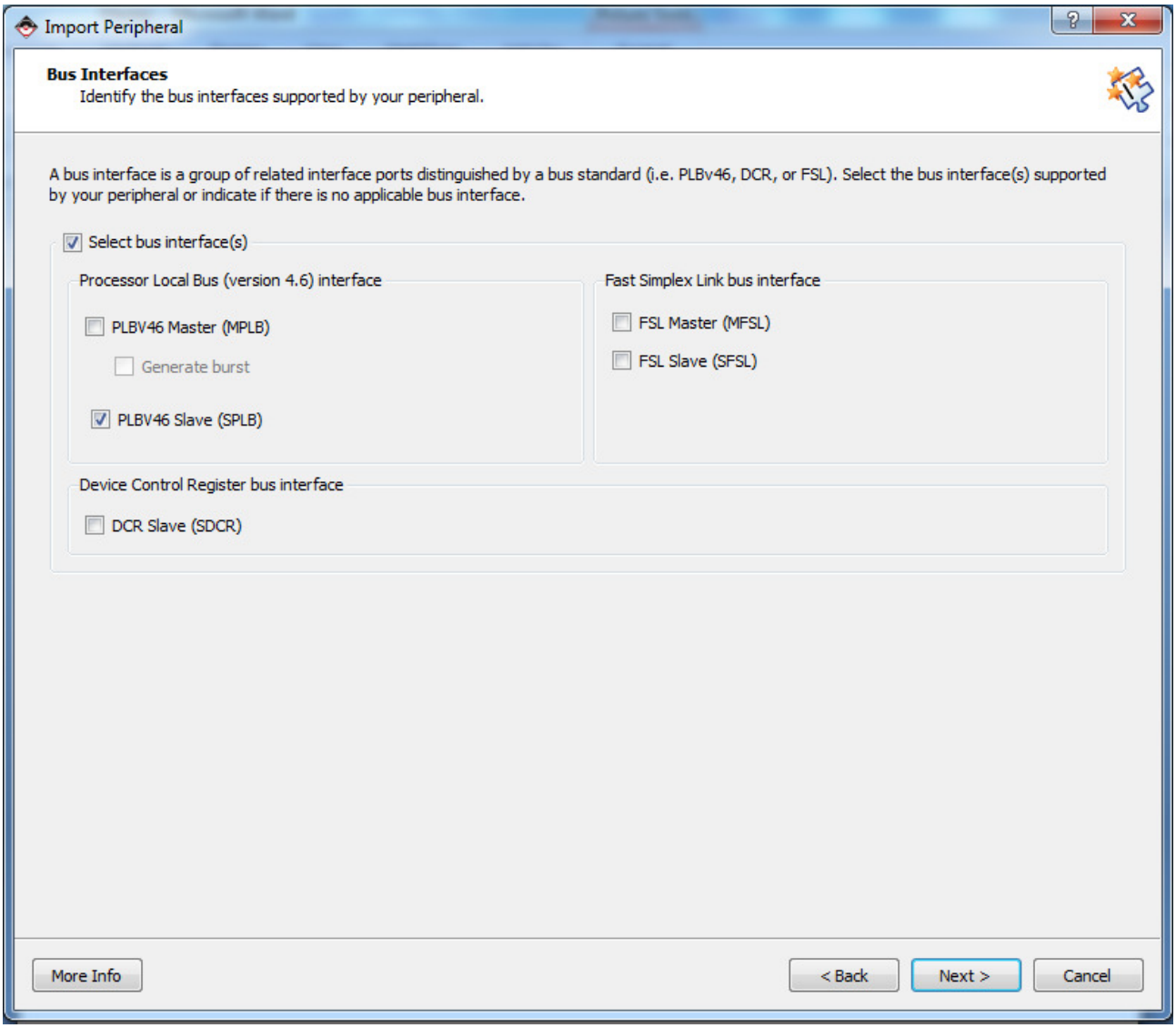
Slika 5.3.2. *HDL Source File* strana

- Na *HDL Analysis Information* stranici, slika 5.3.3, treba proveriti da li su user_logic.vhd i diode.vhd fajlovi na dnu liste. Ukoliko dva VHDL fajla koji su dodati nisu još uvek kompajlirani, kliknuti *Next* da se kompajliraju i otvoriti *Bus Interface* stranicu.



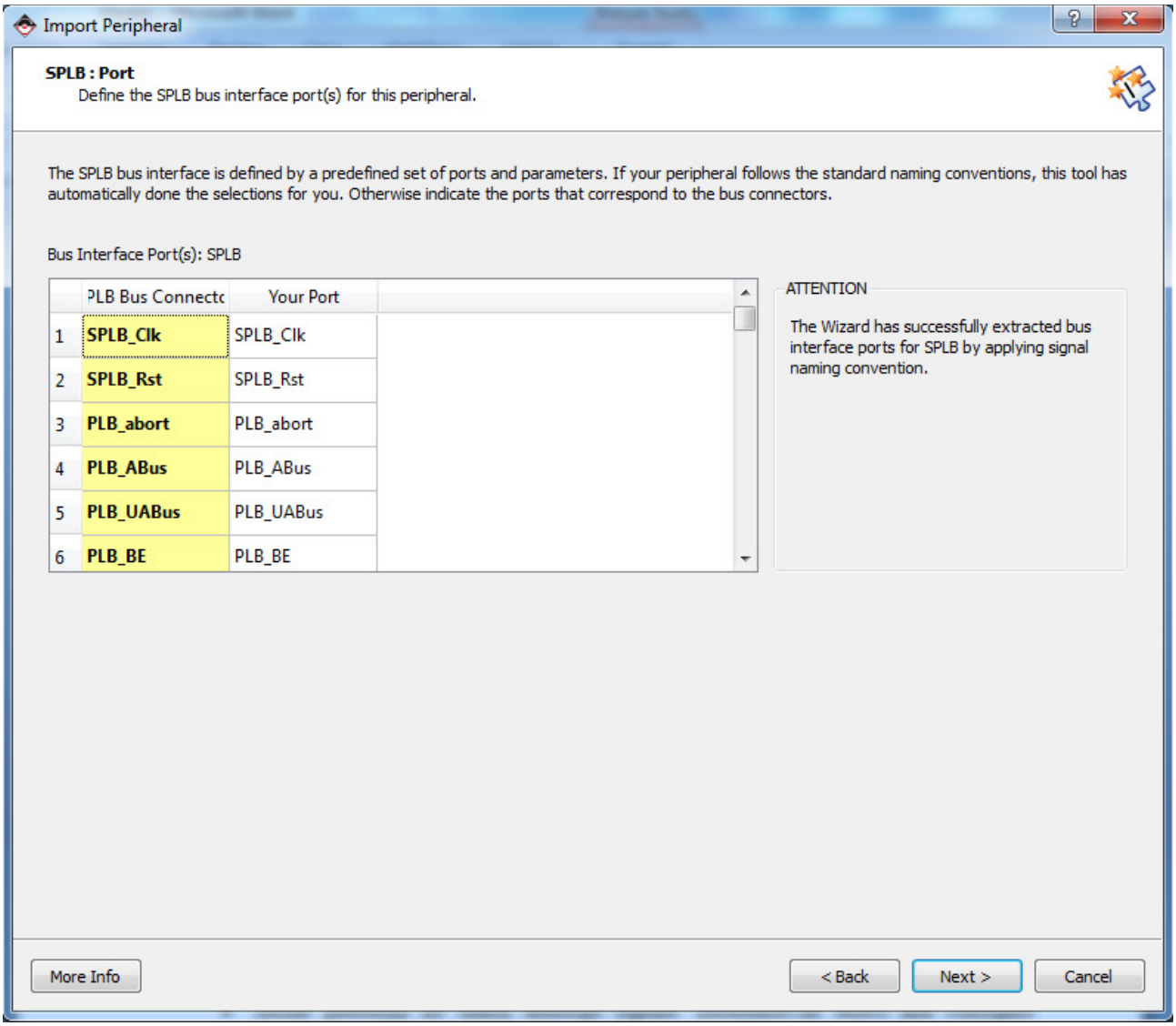
Slika 5.3.3. ***HDL Analysis Information*** **strana**

- Odabrati odgovarajući *bus* interfejs, slika 5.3.4. Periferija *diode* koristi *PLBv46 Slave* interfejs.



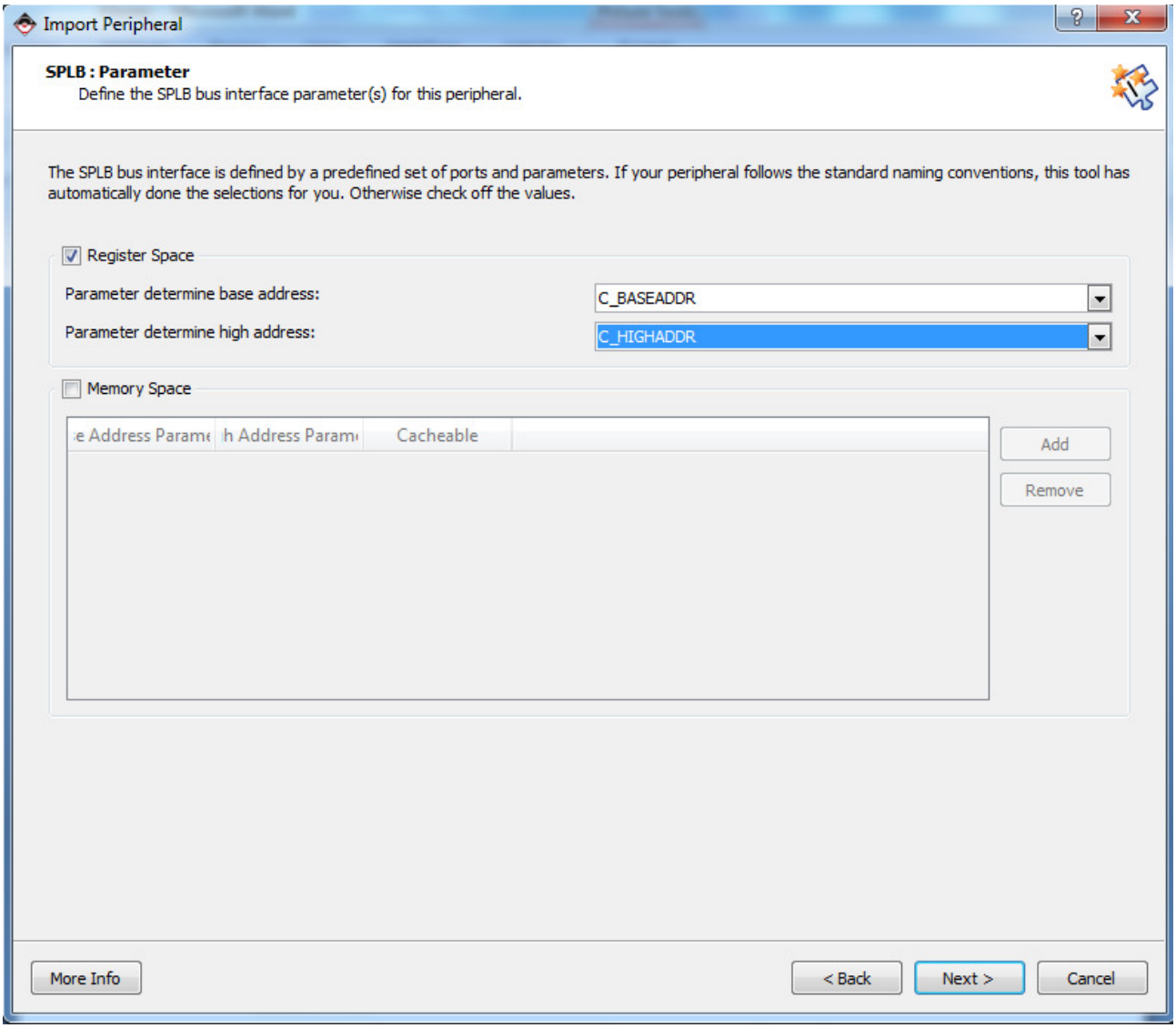
Slika 5.3.4. *Bus Interfaces* strana

- Na *SPLB:Port* stranici, slika 5.3.5 je prikazan kompletan listing svih PLBv46 *bus* signala koji su korišćeni u dizajnu. Dat je spisak signala i pridruženih *bus* protokola koje je CIP vizard automatski konfigurisao. Ukoliko je periferija kreirana pomoću CIP vizarda, svi neophodni signali su uključeni. Ukoliko je periferija kreirana pomoću nekog drugog alata ili sadrži kompleksne *bus* interfejse, ova strana je korisna za analiziranje *bus* signala.



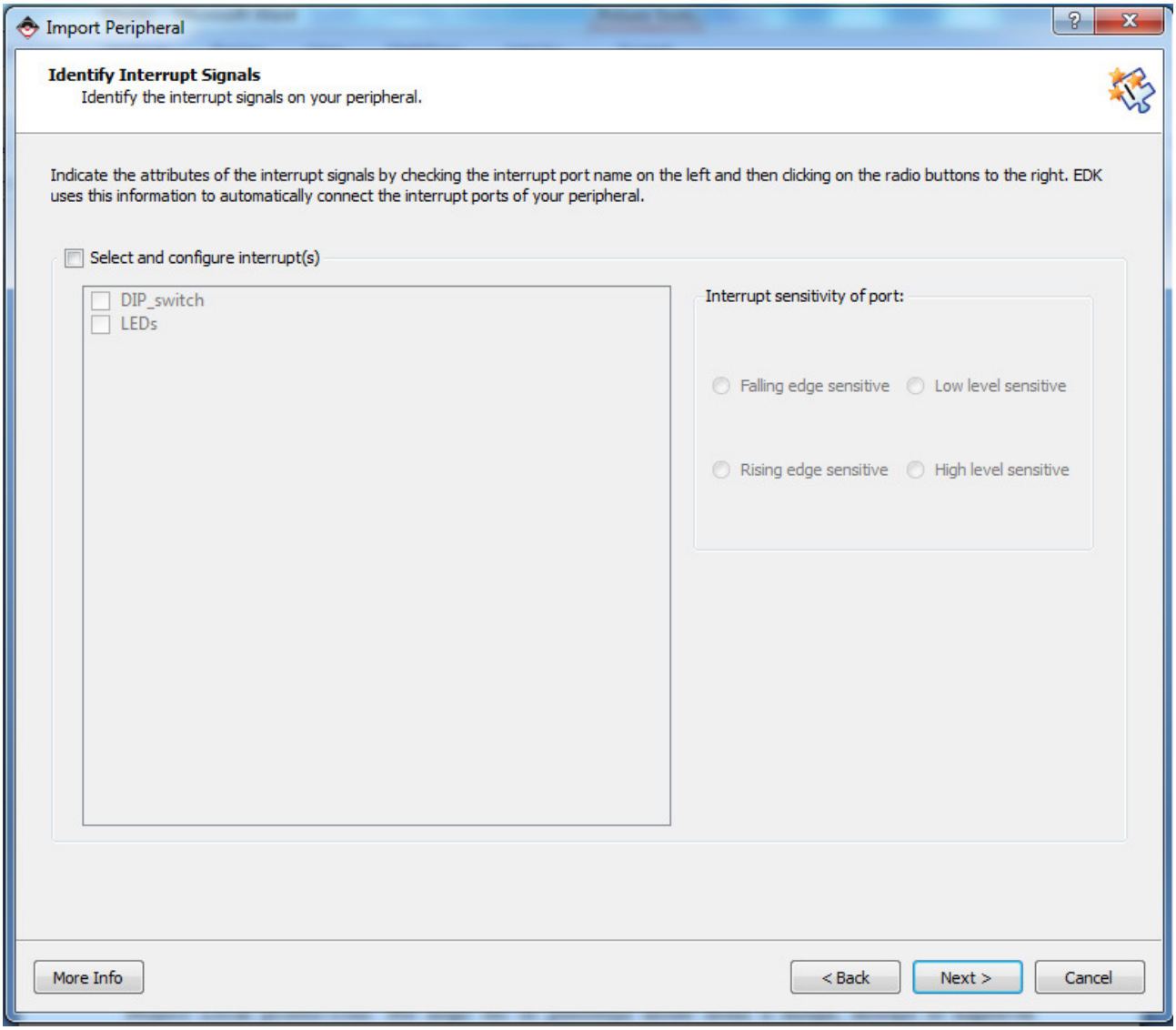
Slika 5.3.4. *SPLB : Port* strana

- Na *SPLB:Parameter* stranici, slika 5.3.5 periferija *diode* je mapirana u jedan adresni opseg, koji XPS bira kad je *pcore* uključen u dizajn. Kompleksnije periferije mogu također sadržati memorijske blokove koji moraju biti dostupni. Kao *high adress* odabrati *C_HIGHADRR*. Kliknuti *Next*.



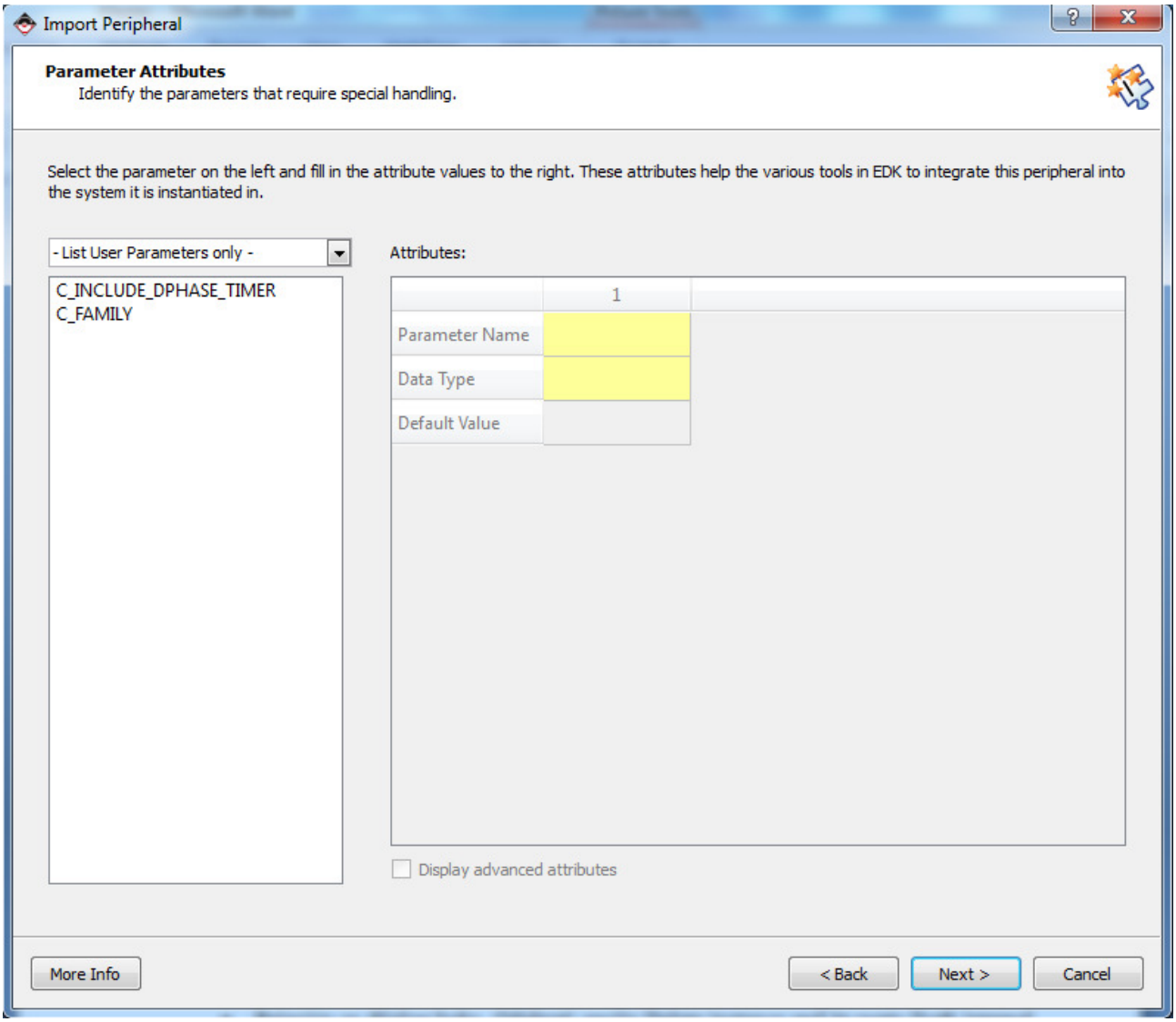
Slika 5.3.5. ***SPLB : Parameter*** **strana**

- Na sledećoj stranici ostaviti podrazumevana podešavanja i kliknuti *Next*.
- Periferija *diode* ne sadrži *interrupt* signale. Deselektovati opciju *Select and configure interrupts*, slika 5.3.6 i kliknuti *Next* da bi se otvorila *Parameter Attribute* stranica.



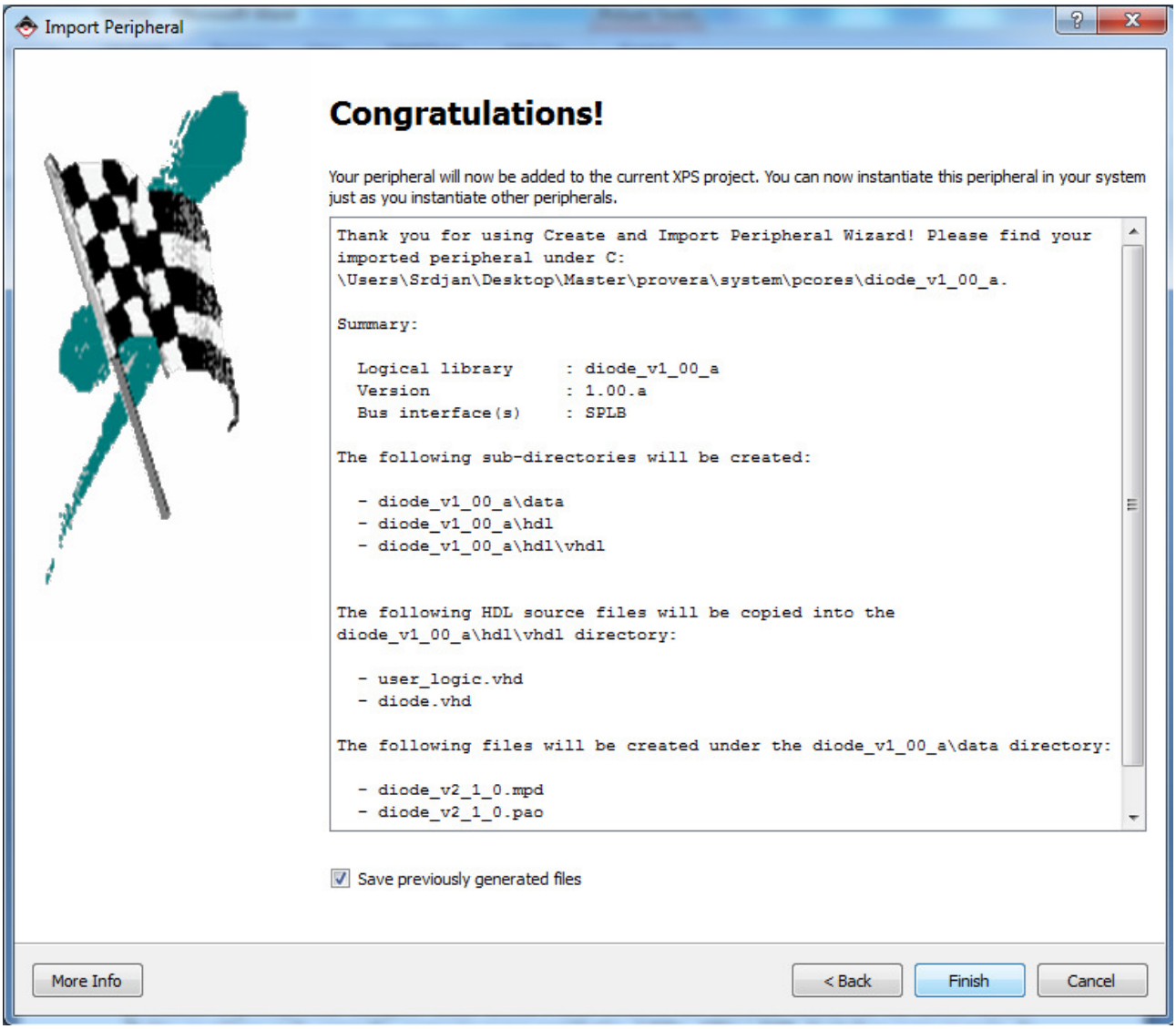
Slika 5.3.6. ***Identify Interrupt Signals*** **strana**

- Na stranici *Port Atributes* su dostupne detaljnije informacije o dodatim portovima. Klinuti *Next*.



Slika 5.3.6. ***Parameter Attributes*** **strana**

- Konačno, prikazuje se stranica sa detaljima projekta, slika 5.3.7. Klinuti *Finish* kako bi se proces importovanja kompletirao.

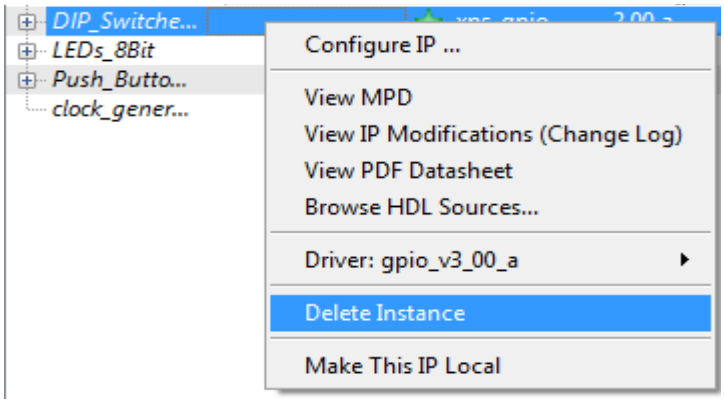


Slika 5.3.7. *Summary* strana

5.3.1. Dodavanje periferije diode u projekat

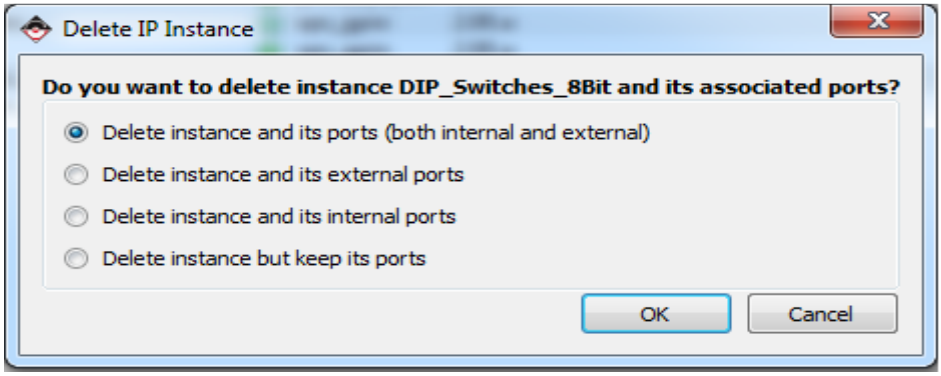
U XPS alatu, u *Project Information* delu, u *IP Catalog* tabu može se videti periferija *diode* u delu *Project Local pcores/User*. Pre nego što se periferija *diode* doda u dizajn, moraju se napraviti promene u postojećem dizajnu. LED diode i DIP svičevi su trenutno nakačeni na GPIO izlaze. Pošto se njima sada upravlja pomoću nove periferije, *LEDs_8Bit i DIP_Switches pcore* moraju da budu uklonjeni iz dizajna.

- U *System Assembly View* desni klik na *LEDs_8Bit* i *DIP_Switches*, i odabrati opciju *Delete Instance*, slika 5.3.1.1.



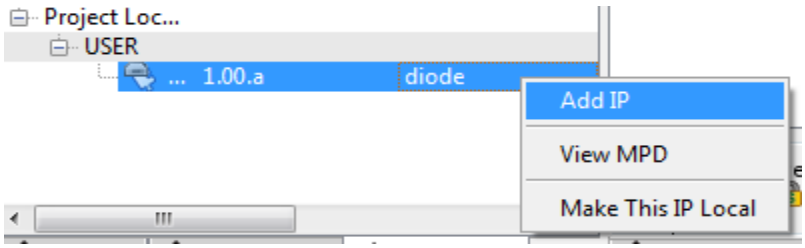
Slika 5.3.1.1. Brisanje instance u XPS alatu

- Pojaviće se dijalog prozor. Odabrati opciju *Delete instance and its ports (both internal and external)*, slika 5.3.1.2.



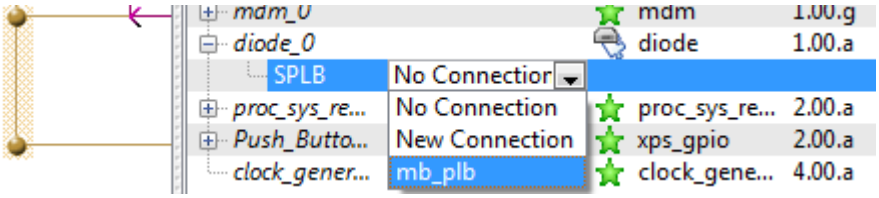
Slika 5.3.1.2. Brisanje instance i portova

- Locirati *diode* u *IP Catalog* tabu, desni klik, i odabrati opciju *Add IP*, slika 5.3.1.3. U dijalog prozoru kliknuti OK. XPS dodaje IP u *System Assembly View*, i može se videti u *Bus Interface* tabu.



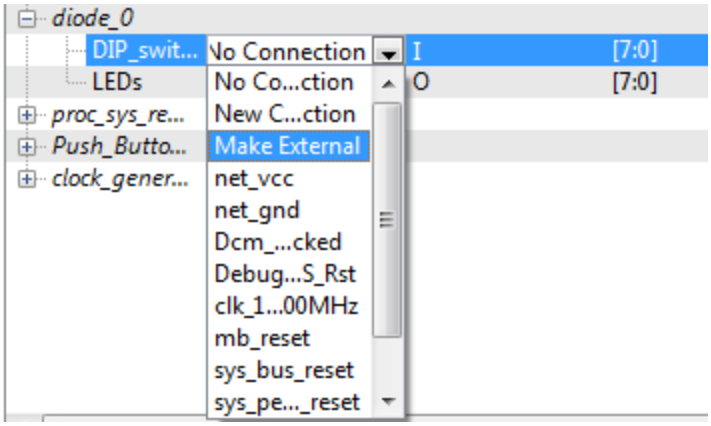
Slika 5.3.1.3. Dodavanje periferije u dizajn

- Kliknuti na + pored imena periferije, otvoriti *drop-down* meni i odabrati opciju *mb_plb* kako bi se povezali na PLB *bus*, slika 5.3.1.4.



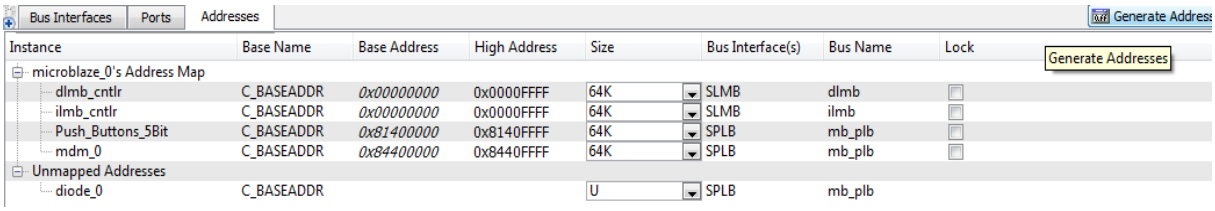
Slika 5.3.1.4. Povezivanje periferije na PLB magistralu

- Otvoriti *Ports* tab, proširiti diode_0, odabrati *Make External* iz *drop-down* menija u *Net* koloni, slika 5.3.1.5. Podrazumevana imena su *diode_0_LEDs_pin* i *diode_0_DIP_switch*. Ime se može promeniti klikom na opciju *Name* u *Net* koloni, respektivno.



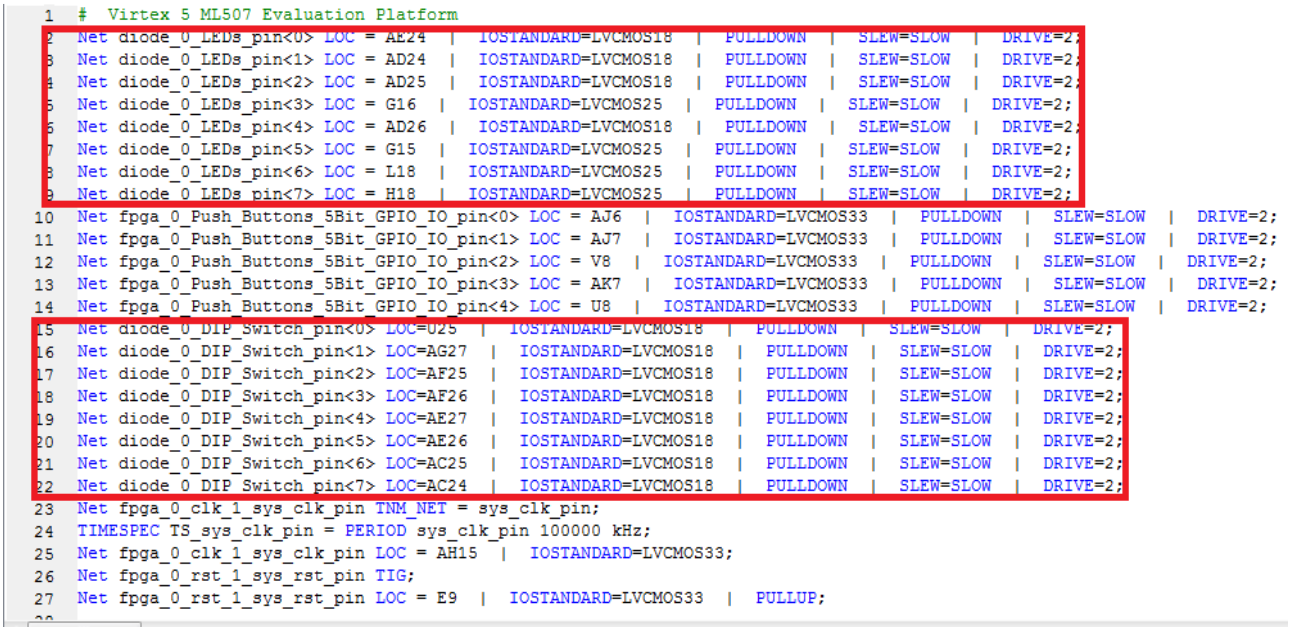
Slika 5.3.1.5. Povezivanje portova na izlaz

- Otvoriti *Adresses* tab. *Diode_0* bi trebalo da se nalaze u delu *Unmapped Addresses*, slika 5.3.1.6. Ukoliko nije tu, kliknuti *Project* > *Rescan User Repositories*.
- Kliknuti na *Generate Addresses* kako bi se periferiji dodelio odgovarajući adresni opseg.



Slika 5.3.1.6. Generisanje adresnog prostora za periferiju

- Konačno treba napraviti izmene i u system.ucf fajlu. Otvoriti projekat u ISE alatu, kliknuti na opciju *File* > *Open*, i odabrati *provera/data/system/system.ucf* fajl. Pronaći *fpga_0_LEDs_8Bit_GPIO_IO_0* i *fpga_0_DIP_Switches_GPIO_IO_0*. Dodele pinova su ostale u ucf fajlu, iako su periferije prethodno izbrisane. Bitno je zapamtiti da uklanjanje periferije ne znači automatski i ažuriranje UCF fajla.
- Zameniti *fpga_0_LEDs_8Bit_GPIO_IO_0_pin* sa *diode_0_LEDs_pin* na svih osam lokacija. Takođe, *fpga_0_DIP_Switches_GPIO_IO_0* zameniti sa *diode_0_DIP_Switch_pin* i sačuvati UCF fajl, slika 5.3.1.7.



```
#  Virtex 5 ML507 Evaluation Platform
Net diode_0_LEDs_pin<0> LOC = AE24  |  IOSTANDARD=LVCMOS18  |  PULLDOWN  |  SLEW=SLOW  |  DRIVE=2;
Net diode_0_LEDs_pin<1> LOC = AD24  |  IOSTANDARD=LVCMOS18  |  PULLDOWN  |  SLEW=SLOW  |  DRIVE=2;
Net diode_0_LEDs_pin<2> LOC = AD25  |  IOSTANDARD=LVCMOS18  |  PULLDOWN  |  SLEW=SLOW  |  DRIVE=2;
Net diode_0_LEDs_pin<3> LOC = G16  |  IOSTANDARD=LVCMOS25  |  PULLDOWN  |  SLEW=SLOW  |  DRIVE=2;
Net diode_0_LEDs_pin<4> LOC = AD26  |  IOSTANDARD=LVCMOS18  |  PULLDOWN  |  SLEW=SLOW  |  DRIVE=2;
Net diode_0_LEDs_pin<5> LOC = G15  |  IOSTANDARD=LVCMOS25  |  PULLDOWN  |  SLEW=SLOW  |  DRIVE=2;
Net diode_0_LEDs_pin<6> LOC = L18  |  IOSTANDARD=LVCMOS25  |  PULLDOWN  |  SLEW=SLOW  |  DRIVE=2;
Net diode_0_LEDs_pin<7> LOC = H18  |  IOSTANDARD=LVCMOS25  |  PULLDOWN  |  SLEW=SLOW  |  DRIVE=2;
Net fpga_0_Push_Buttons_5Bit_GPIO_IO_pin<0> LOC = AJ6  |  IOSTANDARD=LVCMOS33  |  PULLDOWN  |  SLEW=SLOW  |  DRIVE=2;
Net fpga_0_Push_Buttons_5Bit_GPIO_IO_pin<1> LOC = AJ7  |  IOSTANDARD=LVCMOS33  |  PULLDOWN  |  SLEW=SLOW  |  DRIVE=2;
Net fpga_0_Push_Buttons_5Bit_GPIO_IO_pin<2> LOC = V8  |  IOSTANDARD=LVCMOS33  |  PULLDOWN  |  SLEW=SLOW  |  DRIVE=2;
Net fpga_0_Push_Buttons_5Bit_GPIO_IO_pin<3> LOC = AK7  |  IOSTANDARD=LVCMOS33  |  PULLDOWN  |  SLEW=SLOW  |  DRIVE=2;
Net fpga_0_Push_Buttons_5Bit_GPIO_IO_pin<4> LOC = U8  |  IOSTANDARD=LVCMOS33  |  PULLDOWN  |  SLEW=SLOW  |  DRIVE=2;
Net diode_0_DIP_Switch_pin<0> LOC=U25  |  IOSTANDARD=LVCMOS18  |  PULLDOWN  |  SLEW=SLOW  |  DRIVE=2;
Net diode_0_DIP_Switch_pin<1> LOC=AG27  |  IOSTANDARD=LVCMOS18  |  PULLDOWN  |  SLEW=SLOW  |  DRIVE=2;
Net diode_0_DIP_Switch_pin<2> LOC=AF25  |  IOSTANDARD=LVCMOS18  |  PULLDOWN  |  SLEW=SLOW  |  DRIVE=2;
Net diode_0_DIP_Switch_pin<3> LOC=AF26  |  IOSTANDARD=LVCMOS18  |  PULLDOWN  |  SLEW=SLOW  |  DRIVE=2;
Net diode_0_DIP_Switch_pin<4> LOC=AE27  |  IOSTANDARD=LVCMOS18  |  PULLDOWN  |  SLEW=SLOW  |  DRIVE=2;
Net diode_0_DIP_Switch_pin<5> LOC=AE26  |  IOSTANDARD=LVCMOS18  |  PULLDOWN  |  SLEW=SLOW  |  DRIVE=2;
Net diode_0_DIP_Switch_pin<6> LOC=AC25  |  IOSTANDARD=LVCMOS18  |  PULLDOWN  |  SLEW=SLOW  |  DRIVE=2;
Net diode_0_DIP_Switch_pin<7> LOC=AC24  |  IOSTANDARD=LVCMOS18  |  PULLDOWN  |  SLEW=SLOW  |  DRIVE=2;
Net fpga_0_clk_1_sys_clk_pin TNM_NET = sys_clk_pin;
TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 100000 kHz;
Net fpga_0_clk_1_sys_clk_pin LOC = AH15  |  IOSTANDARD=LVCMOS33;
Net fpga_0_rst_1_sys_rst_pin TIG;
Net fpga_0_rst_1_sys_rst_pin LOC = E9  |  IOSTANDARD=LVCMOS33  |  PULLUP;
```

Slika 5.3.1.7. Izmene koje je potrebno napraviti u system.ucf fajlu

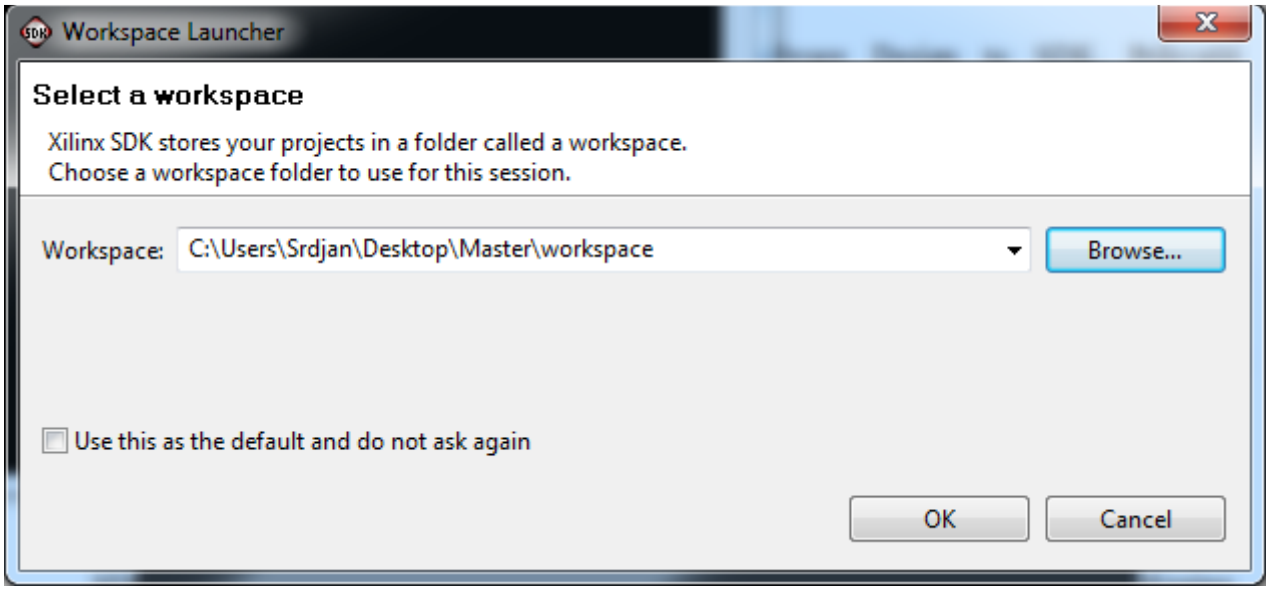
Ukoliko se u diode.vhd ili user_logic.vhd fajlovima vrše neke izmene u kodu (kao npr. izmena funkcionalnosti i sl.) nakon importovanja periferije, potrebno je ponovo generisati *bitstream* fajl. To se radi na sledeći način.

- U XPS alatu se naprave izmene u kodu i klikne na *Save*. Nakon toga se klikne na *Project* > *Clean all generated files*
- Kad se taj proces završi klikne se na *Hardware* > *Generate Netlist*
- Nakon toga se u ISE alatu klikne na *Generate Programming File*

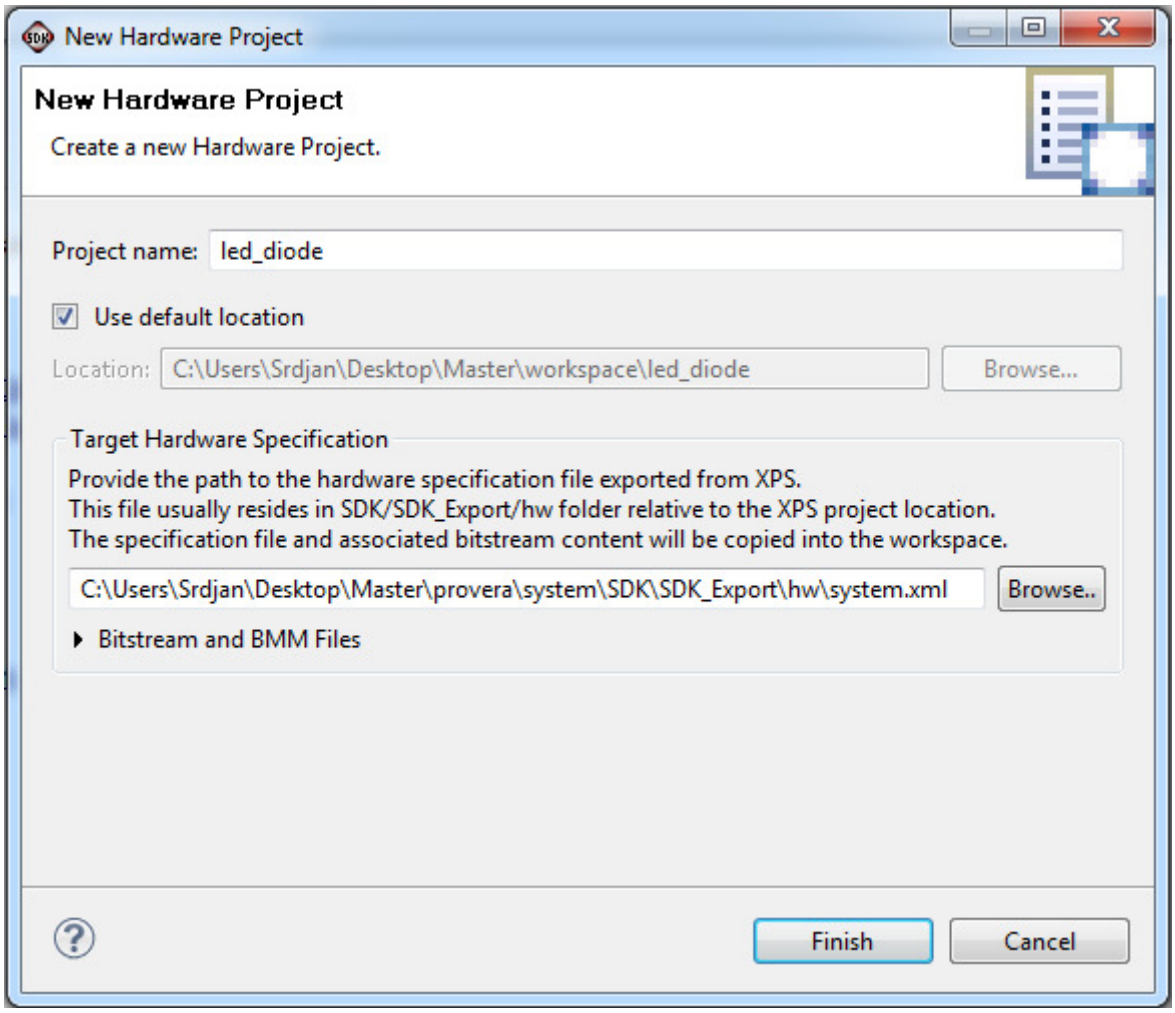
5.3.2. Eksportovanje dizajna i generisanje novog bitstream fajla

Sledeći korak je da se hardverski dizajn eksportuje i da se generiše novi *bitstream* fajl. Nakon toga, treba napisati aplikaciju u C, i dizajn je spreman za testiranje.

- U XPS alatu, odabrati *Project* > *Export Hardware Design to SDK*. Prihvatiti podrazumevani direktorijum i kliknuti *Export Only*.
- Kad se proces eksportovanja završi, zatvoriti XPS, vratiti se u ISE, i dvostruki klik na *Generate Programming File*.
- Kad se završi ovaj proces (obično traje par minuta), otvoriti SDK i kreirati novi radni prostor (*workspace*), slika 5.3.2.1. Kad se SDK otvori, potrebno je označiti eksportovani system.xml fajl.
- Odabrati *File* > *New* > *Xilinx Hardware Platform Specification* za kreiranje novog hardverskog projekta.
- Dati proizvoljno ime projektu (ovde je korišćeno *led_diode*) i označiti system.xml fajl, slika 5.3.2.2. Dotični xml fajl bi trebalo da je smešten u *system/SDK/SDK_Export/HW* folderu projekta.
- Kliknuti *Finish*.



Slika 5.3.2.1. Definisanje *workspace* foldera u SDK alatu

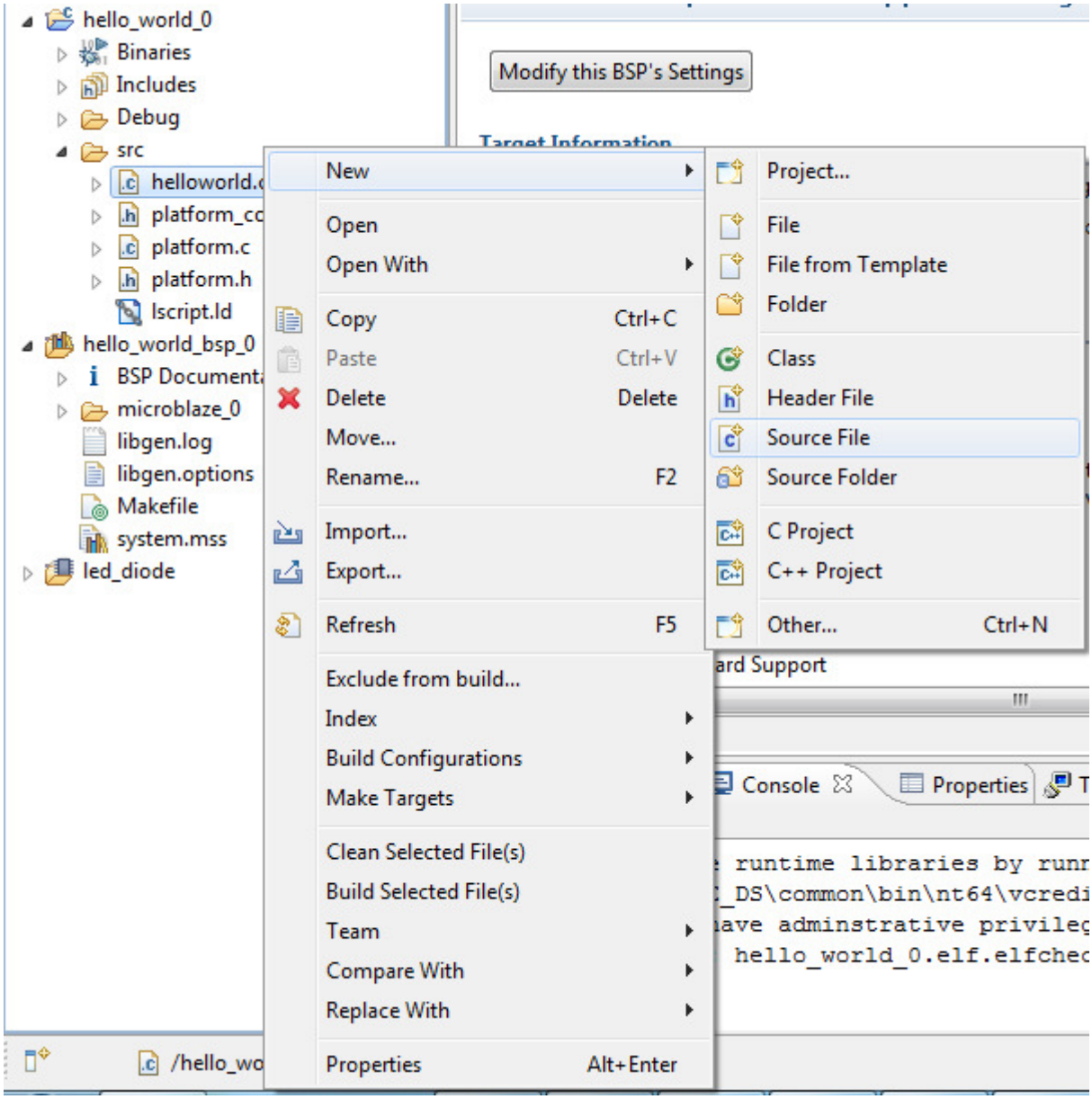


Slika 5.3.2.2. Kreiranje novog projekta

i) Razvoj softvera za kreiranu periferiju

SDK otvara *C/C++ Perspective* sa tabelom u kojoj su prikazane sve periferije u dizajnu. Proveriti da li je *diode_0* na listi. Postupak razvoja aplikacije je sledeći:

- Kreirati novi *Xilinx C* projekat, klikom na *File* > *New* > *Xilinx C Project*. Koristiti *Hello World* templejt, i ostaviti podrazumevano ime koje je predložio SDK. Korišćenje templejta je korisno, naročito za početnike, jer je na taj način već urađena većina podešavanja drajvera i slično. Napredniji korisnici, mogu kasnije ova podešavanja vršiti i ručno. Kliknuti *Next*.
- Podrazumevano podešavanje na ovaj strani je da SDK kreira novi BSP paket. Ostaviti tako. Kliknuti *Finish* da bi se generisao projekat.
- U *Project Explorer* kliknuti desni klik na novi projekat i odabrati *New* > *Source File*, slika 5.3.2.3.



Slika 5.3.2.3. Dodavanje novog *source* fajla u projekat

- Staviti sledeća podešavanja:
  - *Source Folder: hello_world_0/src*
  - *Source File: leds.c*
  - *Template: Default C source template*

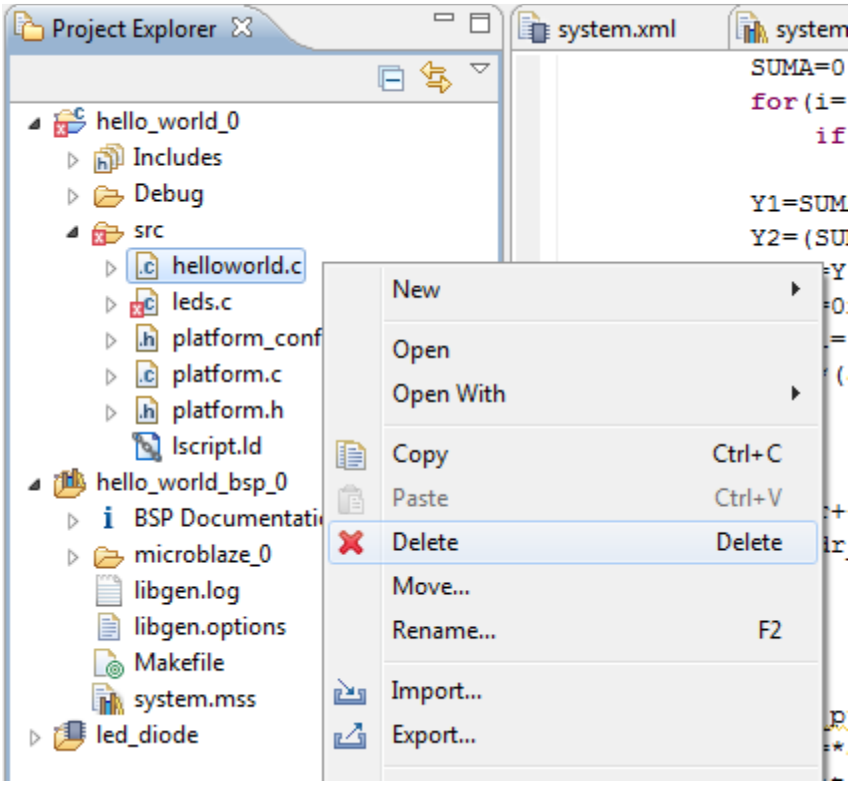
  Kliknuti *Finish*.

leds.c fajl se otvara u SDK prozoru, i tu treba da se unese C kod, koji procesor izvršava. U prvom delu koda je *while/do* petlja koja omogućava izvršavanje koda kad se pritisne neko *push* dugme. Na DIP sviču je prethodno izabrana kombinacija na osnovu koje se učitava odgovarajuće zaglavlje (ovaj deo je definisan u user_logic.vhd fajlu). Dalje se vrši provera zaglavlja. Zaglavlje se deli na blokove od po 16 bita. Zatim se ti blokovi sabiraju i ukoliko je zbir jednak 0xFFFF čeksuma je u redu. Osim toga treba utvrditi ispravnost polja verzije i ttl. Ukoliko je zaglavlje validno, *ttl* polje se umanjuje za 1 i vrši se ažuriranje zaglavlja koje se potom upisuje u odgovarajuće registre u periferiji. Takođe, i brojač *corr* (broj validnih zaglavlja) kojim se reguliše paljenje dioda se uvećava za jedan. Ukoliko je zaglavlje pogrešno uvećava se brojač *uncorr* koji takođe reguliše paljenje dioda. Četiri diode na ploči služe za brojanje validnih zaglavlja, a četiri za brojanje pogrešnih. Ove informacije (o broju tačnih i pogrešnih zaglavlja) se zatim smeštaju u *slv_reg0* i *slv_reg1*. Adrese ovih registara su *XPAR_DIODE_0_BASEADDR* i *XPAR_DIODE_0_BASEADDR+1*. Na kraju koda je još jedna *while* petlja. Ona se izvršava dok se *push* dugme ne vrati u nepritisnuto stanje kako bi se izbegla višestruka provera i ažuriranje istog zaglavlja. Kao što se vidi kod je veoma jednostavan tako da je znatno lakše napisati softverski kod za proveru i ažuriranje zaglavlja nego napisati kod koji opisuje hardver koji obavlja istu funkciju.

```
#include "xparameters.h"
#include "mb_interface.h"
#include "stdio.h"
#include "xgpio.h"
#include "xutil.h"
#include "stdlib.h"

int main() {
	int i, x1, x2, x[5], data, *addr_ptr, *addr_ptr1, y[10], Y1=0, Y2=0,
	SUMA=0, check=0, cor=0, uncor=0;

	do
	{
		do
		{
			addr_ptr = XPAR_PUSH_BUTTONS_5BIT_BASEADDR;
			data=*addr_ptr;// ucitavanje podataka sa push button-a
		}while(data==0);// kod se izvrsava dalje kad se preko push button-a
unese bilo koja vrijednost razlicita od , tj kada se pritisne neki od push
button-a

		addr_ptr1=XPAR_DIODE_0_BASEADDR;//ucitavanje podataka iz periferije
		SUMA=0x00000000;//postavljanje sume na 0
		check=0;
		for(i=0; i<5; i++)
		{
			y[2*i]=((*(addr_ptr1+4-i+2))>>16)&0x0000FFFF;//ucitavanje
```

```
                y[2*i+1]=(*(addr_ptr1+4-i+2))&0x0000FFFF;//blokova od po 16
bita
                SUMA=SUMA+y[2*i]+y[2*i+1];// sabiranje dobijenih blokova
podataka
            }
            if((y[0]>>12)!=4 || y[4]>>8==0)// provera polja verzije i TTL
                check=1;
            Y1=SUMA&0x0000FFFF;//racunanje check sume
            Y2=(SUMA>>16)&0x0000FFFF;
            SUMA=Y1+Y2;

            if(SUMA!=0x0000FFFF)// provera check sume
                check=1;//greska

            if(check==0)//ukoliko je zaglavlje validno, uvecava se brojac,
zaglavlje se ažurira i upisuje u odgovarajuce registre
            {
                cor++;
                *(addr_ptr1) = cor;
                x1=(y[4]>>8)&0x000000FF;
                x1=x1-1;
                x2=(y[4])&0x000000FF;
                y[4]=((x1<<8)|x2);
                SUMA=0;
                for(i=0; i<10; i++)
                    if(i!=5)
                        SUMA=SUMA+y[i];
                Y1=SUMA&0x0000FFFF;
                Y2=(SUMA>>16)&0x0000FFFF;
                SUMA=Y1+Y2;
                y[5]=0xFFFFFFFF-SUMA;
                for(i=0; i<5; i++)
                    *(addr_ptr1+7+i) = ((y[2*i]<<16)|(y[2*i+1]));
            }
            else//ako je zaglavlje pogrešno, uvecava se odgovarajuci brojac
            {
                uncor++;
                *(addr_ptr1+1) = uncor;
            }

            do//petlja za kašnjenje
            {
                addr_ptr = XPAR_PUSH_BUTTONS_5BIT_BASEADDR;
                data=*addr_ptr;// ucitavanje podataka sa push button-a
            }while(data!=0);// kod se izvrsava dalje kad se preko push button-a
unese bilo koja vrijednost razlicita od 0

        }while(1);
}
```

- Kliknuti *Save*, i SDK će automatski kompajlirati projekat. SDK će prijaviti grešku zbog toga što u projektu postoji više *main()* funkcija. Treba izbrisati *hello_world.c* i *platform.c*. Desni klik na ove fajlove, i izabrati *Delete*, slika 5.3.2.4. Sada bi projekat trebalo da se kompajlira uspešno.

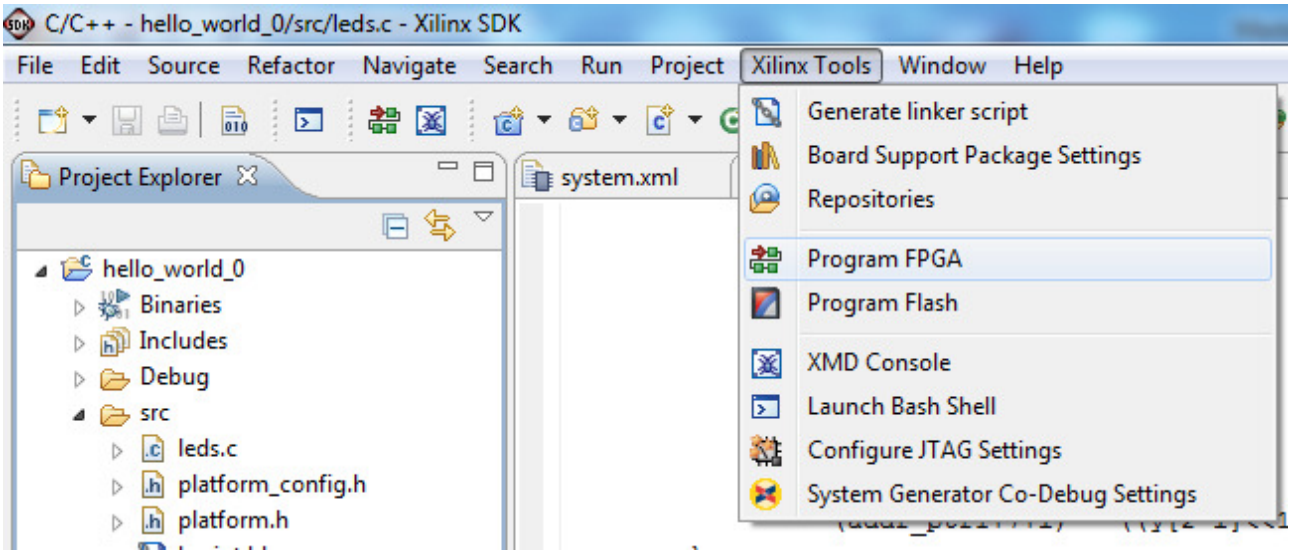


Slika 5.3.2.4. Brisanje C fajla

6. VERIFIKACIJA DIZAJNA

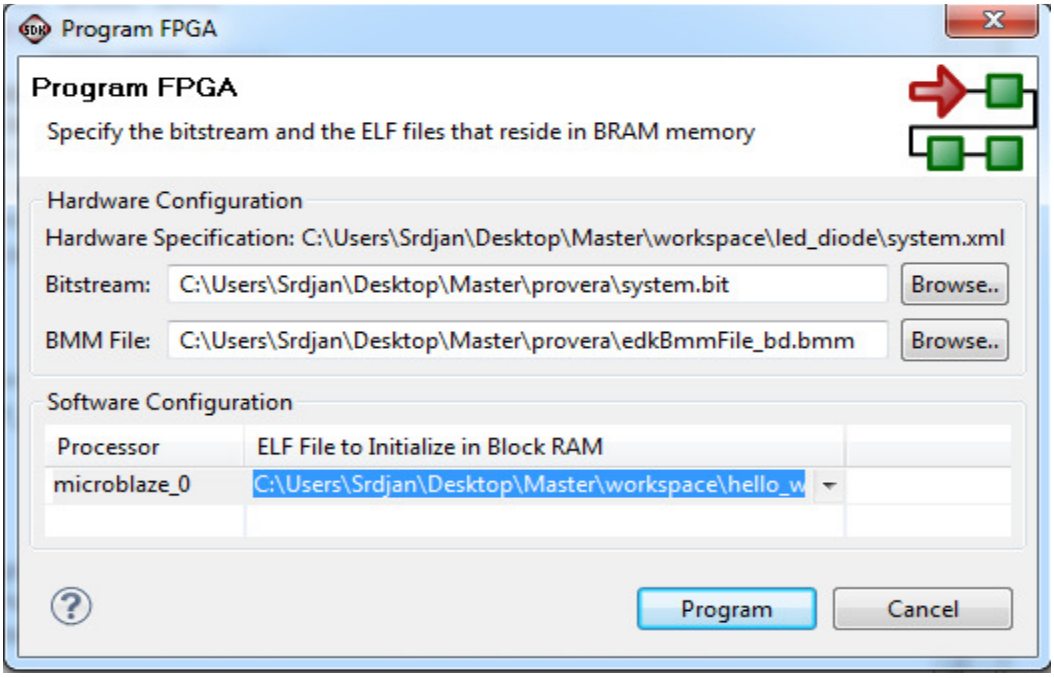
Dakle, sada je završen razvoj hardverskog i softverskog dela projekta. U ISE alatu je prvo kreiran projekat. Dodali smo *source* fajl u vidu *embedded* procesora i u BSB konfigurisali hardverske parametre: tip procesora, veličinu RAM memorije, periferije itd. Kreirana je nova periferija koja vrši proveru i ažuriranje IPv4 zaglavlja, i nakon toga importovana u dizajn. Konačno, u SDK alatu je napisana odgovarajuća C aplikacija koju procesor izvršava i koja vrši zadatu funkciju. Na kraju preostaje verifikacija dizajna. U ovom primeru je korišćena *Virtex ML507* ploča. Na DIP sviču, korisnik bira odgovarajuću kombinaciju. U diode.vhd fajlu se dekodovanjem stanja DIP sviča bira odgovarajuće zaglavlje za ispitivanje. Pritiskom na *Push* dugme, pokreće se izvršavanje C koda. Procesor na osnovu kombinacije sa DIP sviča, učitava odgovarajuće zaglavlje iz registara u periferiji, i vrši proveru. Ukoliko je tačno, zaglavlje se ažurira i smešta u registre periferije koju smo importovali. Takođe, pale se odgovarajuće diode. Na osnovu upaljenih LED dioda može se prebrojati koliko je tačnih odnosno pogrešnih zaglavlja obrađeno. Spuštanje dizajna na ploču se vrši na sledeći način:

- U SDK otvoriti *Xilinx Tools* > *Program FPGA*, slika 6.1.



Slika 6.1. Spuštanje dizajna na ploču

- Treba ubaciti putanju do *bitstream* (*.bit) i *block memory map* (*.bmm) fajlova koji se nalaze u *provera* folderu, slika 6.2.
- U polju "*ELF File to Initialize in Block RAM*" odabrati elf fajl, slika 6.2.
- Kliknuti na *Program*



Slika 6.2. Podešavanje BIT, BMM i ELF fajlova

Postoji nekoliko zaglavlja, pri čemu je samo jedno zaglavlje korektno. Ono se iščitava ukoliko je na DIP sviču aktivna kombinacija 00000100. Druge kombinacije iščitavaju pogrešna zaglavlja, što bi trebalo da se vidi paljenjem odgovarajućih dioda. Testiranje dizajna na *ML507* razvojnoj ploči je potvrdilo ispravnost rada dizajna. U tabeli 6.3 je prikazana zauzetost resursa FPGA čipa. Iz tabele se vidi da se koristi mala količina resursa kao i da je zadovoljeno vremensko ograničenje za takt procesora navedeno u tabeli 4.2.6.

Slice Logic Utilization	***Used***	***Available***	***Utilization***
Number of Slice Registers	1875	44800	4%
Number of Slice LUTs	2019	44800	4%
Number of BlockRAM/FIFO	16	148	10%
Total Memory used (KB)	576	5328	10%
Number of BUFG/BUFGCTRLs	2	32	6%
Number of BSCANs	1	4	25%
Number of DSP48Es	3	128	2%
Number of PLL_ADVs	1	6	16%
Average Fanout of Non-Clock Nets	4.20		

Tabela 6.3 Zauzetost FPGA resursa

7. ZAKLJUČAK

U ovom master radu je predstavljen proces dizajniranja *embedded* sistema koji vrši funkciju provere i ažuriranja IPv4 zaglavlja. Ovo je jedna od funkcija koja se izvršava u ravni podataka rutera. Ova implementacija se razlikuje po tome što nije čisto hardverska, već predstavlja kombinaciju hardverskog i softverskog rešenja. Generalno, razvoj ovakvih sistema je vrlo komplikovan jer zahteva dizajn i hardvera i softvera kao i njihovo međusobno integrisanje. Međutim, u ovom radu je za razvoj hardverskog odnosno softverskog dela projekta korišćen EDK alat. Ovaj alat značajno olakšava razvoj *embedded* sistema, jer je veliki deo posla automatizovan. Proces dizajniranja sistema se u dobroj meri svodi na serije odgovarajućih selekcija među ponuđenim opcijama, što je detaljno objašnjeno u radu. Takođe, EDK prilikom razvoja projekta generiše templejt fajlove sa preciznim uputstvima, tako da je i dodavanje novih funkcionalnosti značajno pojednostavljeno. U ovom radu je korišćena *Virtex ML507* ploča sa *MicroBlaze soft* procesorom. Ovaj tip procesora ima veliki broj opcionih karakteristika koji korisnicima pruža veliku fleksibilnost u radu i mogućnost balansiranja između zahteva *embedded* sistema i zauzetih resursa FPGA čipa. Prednost ovakvog rešenja je jednostavnost, mogućnost lakog modifikovanja postojećih i dodavanja novih funkcionalnosti ili periferija. Takođe, u ovakvim sistemima nema potrebe za izmenama u hardveru (sem ako se sama struktura periferije menja), već se sve modifikacije vrše u softveru. Prednost se ogleda i u tome što se za različita procesiranja paketa može koristiti ista hardverska arhitektura bez zauzimanja dodatnih resursa, već se samo modifikuje i unapređuje softver. Mana ovakvog rešenja jeste manja brzina u odnosu na čisto hardversku implementaciju. Cilj ovog rada je da se predstave mogućnosti koje EDK alat nudi u razvoju *embedded* sistema. U tezi su data detaljna uputstva sa odgovarajućim slikama, tako da novi korisnici mogu odmah da počnu da koriste ove alate za svoje potrebe. U ovom radu je obrađena samo jedna od funkcija paketskog procesiranja koja se obavlja u ruterima, i ona predstavlja osnovu koja se lako može proširiti i drugim funkcijama paketskog procesiranja.

LITERATURA

[1] Zoran Čiča, Materijali sa predavanja iz predmeta Komutacioni sistemi

[2] Z. Čiča, „Analysis and Implementation of Packet Processing Functions in Internet Routers," *Proc. of TELFOR 2012*, Belgrade, Serbia, November 2012.

[3] MicroBlaze procesor reference guide, http://www.xilinx.com/support/documentation/sw_manuals/mb_ref_guide.pdf

[4] Rod Jesman, Fernando Martinez Vallina, Jafar Saniie, MicroBlaze Tutorial, http://ecasp.ece.iit.edu/tutorials/microblaze_tutorial.pdf

[5] EDK Concepts, Tools and Techniques, http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/edk_ctt.pdf