UNIVERZITET U BEOGRADU, ELEKTROTEHNIČKI FAKULTET, KATEDRA ZA ELEKTRONIKU

PROJEKTOVANJE JEDNOSTAVNOG PROCESORSKOG SISTEMA NA FPGA

DIGITALNI VLSI SISTEMI

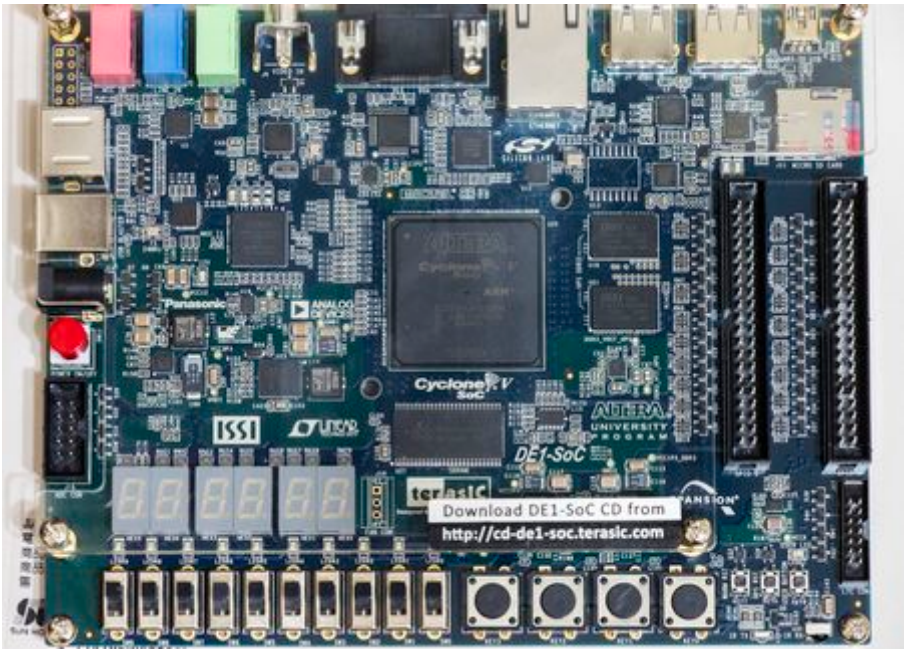
V2.0

BEOGRAD, 2018

Uvod

Cilj vežbi na predmetu Digitalni VLSI sistemi je da se studenti upoznaju sa metodologijom projektovanja sistema na čipu, programiranjem tih sistema i njihovim debagovanjem. Tipičan sistem na čipu se sastoji od procesora, memorije i periferija povezanih na zajedničku magistralu i eksternih komunikacionih interfejsa. U našem slučaju sistem na čipu biće realizovan u FPGA sa soft procesorskim jezgrom. Soft procesor predstavlja procesor dat u vidu HDL opisa koji se realizuje programiranjem FPGA čipa. Za razliku od toga postoje hard procesorska jezgra koja predstavljaju parče čipa namenjeno samo procesoru koje pruža maksimalne performanse ali je fleksibilnost ograničena.

Tokom izvođenja vežbi koristi se razvojni sistem DE1-SoC prikazan na Slici 1. koji proizvodi kompanija Terasic. Sve informacije o razvojnom sistemu mogu se pronaći na sledećem linku.



Slika 1. DE1-SoC razvojni sistem

Razvojni sistem na sebi sadrži FPGA čip *Cyclone V* kompanije *Altera* tako da će postupak projektovanja biti zasnovan na alatima ove kompanije. Potrebno je napomenuti da je metodologija projektovanja univerzalna i da druge kompanije poput *Xilinx*-a nude veoma slične alate tako da studenti ne bi trebalo da imaju problema da koriste bilo koju od dostupnih platformi.

Svaki proizvođač FPGA čipova nudi svoju varijantu soft procesorskog jezgra koja je optimizovana za arhitekturu njihovih FPGA čipova mada se često može implementirati i u nekom drugom FPGA čipu uz odgovarajući gubitak performansi. Takođe postoje IP soft procesorskih jezgara koji su ili *open source* ili ih nude kompanije koje se bave isključivom razvojem IP blokova. Soft procesorsko jezgro kompanije Altera je Nios2 procesor i sistem koji će biti projektovan tokom vežbi je baziran upravo na ovom procesoru.

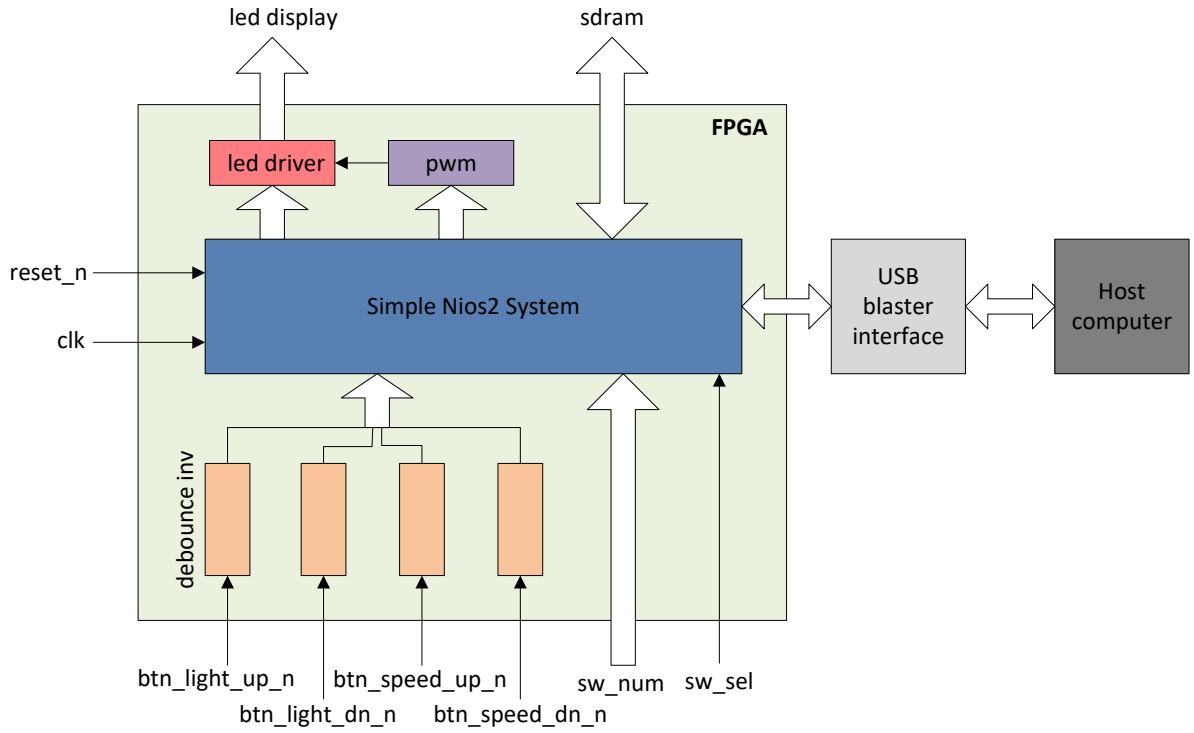
U nastavku će biti dat opis sistema kao i detaljno uputstvo korak po korak realizacije tog sistema.

Opis sistema

Na Slici 2. je prikazana blok šema jednostavnog sistema na čipu koji će biti projektovan tokom vežbe. Sistem se sastoji od procesorskog sistema koji uključuje Nios2 soft procesorsko jezgro i niz

periferija povezanih na zajedničku Avalon magistralu. Procesorski sistem je prikazan na Slici 3. ali više reči o njemu će biti kasnije.

Osnovna funkcionalnost sistema je da prihvata dva 8-bitna broja sabira ih i prikazuje rezultat na LED displeju koji menja osvetljaj od najvećeg do najmanjeg sa periodom koji se može kontrolisati pomoću tastera na ploči. Procesor očitava brojeve sa prekidača sabira ih, konvertuje zbir u BCD i šalje ka LED drajveru. Kako na ploči nije dostupno 16 prekidača, za unos oba broja se koristi isti set od 8 prekidača dok se informacija o tome koji broj se trenutno unosi zadaje pomoću dodatnog prekidača. Periferija LED drajvera obezbeđuje konverziju iz BCD prikaza u odgovarajuće signale za aktiviranje dioda na LED displeju. LED drajver takođe prima enable signal iz PWM modula čime se obezbeđuje promena jačine osvetljaja LED displeja. Perioda PWM modula se kontroliše sa procesora. Kako se tasterima menja period promene osvetljaja displeja obezbeđeno je da se tasteri očitavaju u prekidnoj rutini koja se aktivira na uzlaznu ivicu signala sa tastera. Kako bi se sprečilo lažno višestruko aktiviranje prekidne rutine koje se može javiti kao posledica odskakivanja tastera u sistem su za svaki taster uvedeni moduli za deabunsiranje. Osim debaunsiranja ovi moduli i komplementiraju aktivnu vrednost signala. Procesor smešta instrukcije i podatke u odgovarajuću RAM memoriju. U ovu svrhu se može koristiti memorija sa FPGA čipa ali ona ima dosta ograničen kapacitet. Druga opcija je korišćenje eksternog SDRAM modula. Na razvojnoj ploči DE1-SoC postoji SDRAM modul kapaciteta 64MB i on je korišćen za smeštanje podataka i instrukcija procesora. Komunikacija sa računarom se obavlja serijskim putem pomoću JTAG UART-a koji pored komunikacije služi i za programiranje čipa. USB Blaster drajver omogućava konverziju UART u USB interfejs tako da se koristi jedinstven USB priključak za programiranje i komunikaciju sa Host računarom.



Slika 2. Blok šema jednostavnog sistema na čipu

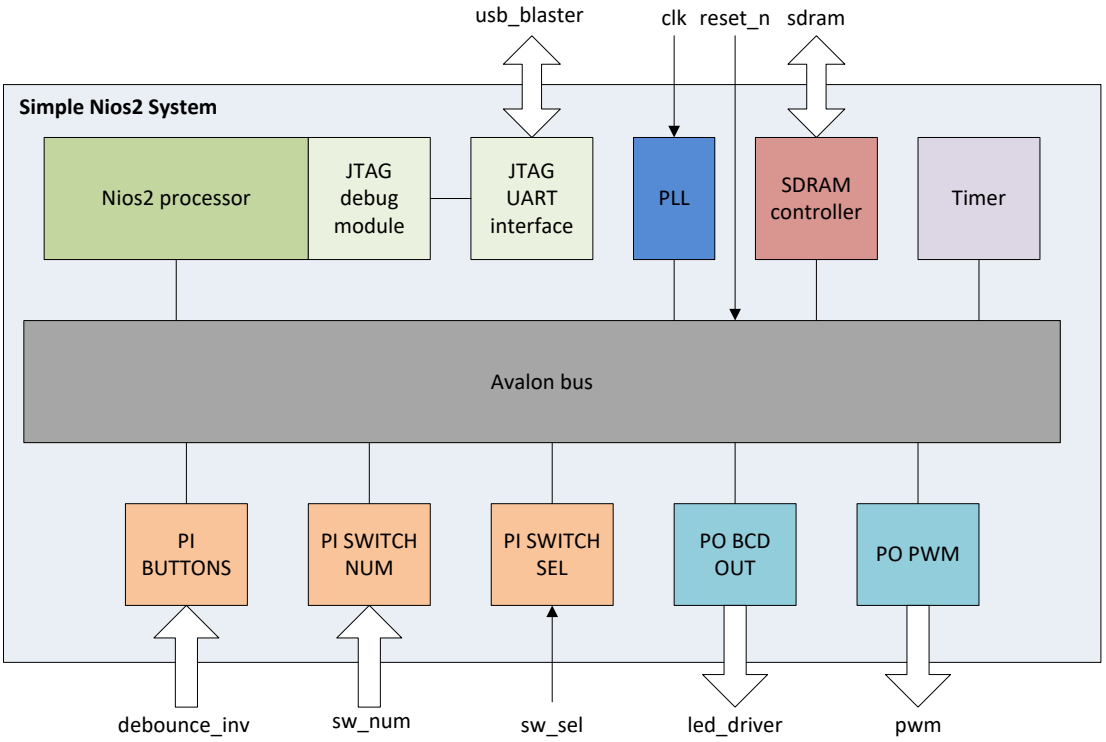
Realizacija sistema

Pre početka same realizacije je potrebno razložiti problem i odrediti koje delove je najpogodnije rešavati u hardveru a koje u softveru. Ovakva dekompozicija problema se naziva

hardware-software co-design. Moguće je i kasnije menjati sistem tako što se neke vremenski zahtevne stvari prebacuju iz softvera u hardver radi optimizacije ali je najbolje na početku uraditi što bolju dekompoziciju problema kako bi se broj iteracija sveo na minimum. Kada se odredi dekompozicija potrebno je na jasan način definisati interfejse i protokole između komponenti u sistemu. Jasno definisanje interfejsa i protokola je jako važno jer omogućava podelu posla na više ljudi i posle jednostavno uklapanje delova sistema u celinu.

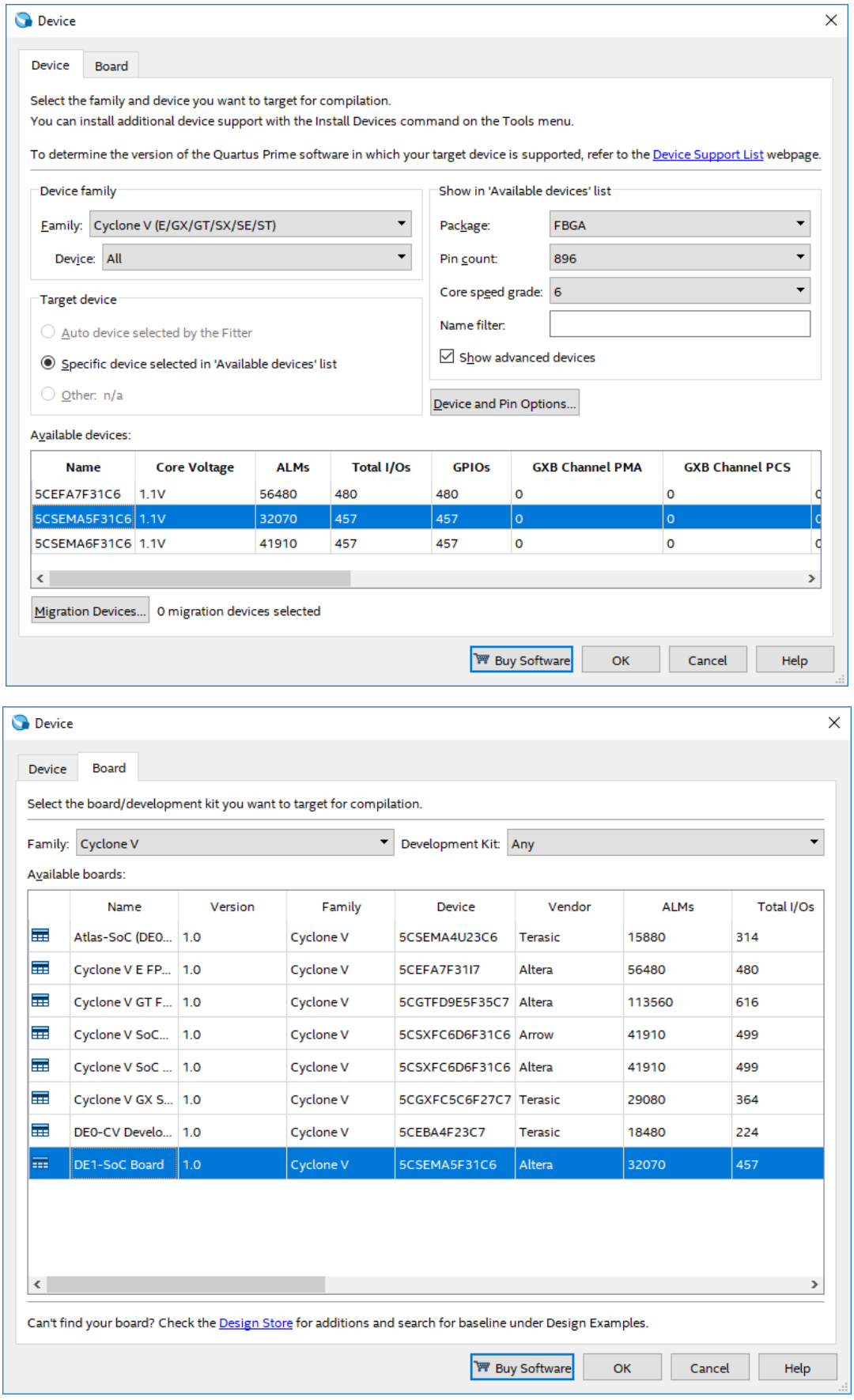
Komponente ***led driver***, ***pwm*** i ***debounce inv*** su realizovane kao zasebni VHDL moduli. Kako je akcenat ovog dokumenta na realizaciji sistema, realizacija ovih posebnih komponenti neće biti razmatrana već će se one koristiti kao gotovi moduli.

Za projektovanje procesorskog sistema prikazanog na Slici 3. se koristi Platform Designer alat koje je deo Quartus Prime programskog paketa. Ovaj alat omogućava realizaciju procesorskog sistema jednostavnim izborom funkcionalnih blokova, definisanjem njihovih parametara i povezivanjem u grafičkom interfejsu. Generisanje logike za povezivanje na Avalon magistralu, adresnih dekodera i multipleksera je automatizovano u Platform Designer alatu i projektant sistema ne mora da razmišlja o tome.



Slika 3. Blok šema jednostavnog procesorskog sistema

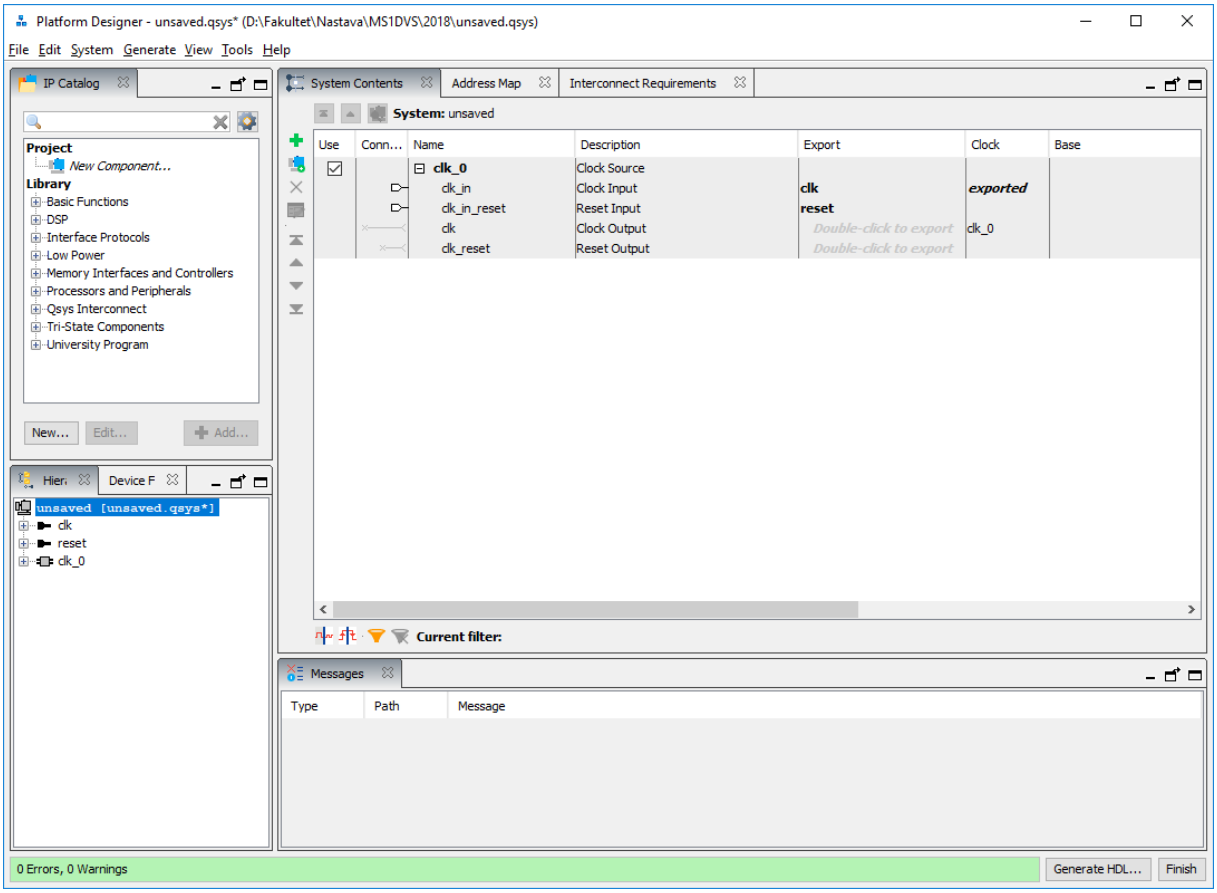
Nakon pokretanja Quartus Prime alata potrebno je kreirati novi projekat pod nazivom ***Simple_Nios2_Project*** i podesiti putanju na kojoj će biti sačuvan. Obavezno kreirati novi direktorijum MS1DVS na D disku i ovaj direktorijum podesiti za projekat. Sledeći dijalog nudi ubacivanje postojećih fajlova i može se preskočiti pošto se postojeći fajlovi mogu i naknadno dodati. Vrlo je važno podesiti familiju čipova na koji će dizajn biti implementiran. U ovom primeru se koristi ***Cyclone V 5CSEMA5F31C6***. Druga opcija je da se u okviru kartice Board odabere familija čipova ***Cyclone V*** i zatim selektuje **DE1-SoC Board**. Oba načina su prikazana na Slici 4.



Slika 4. Podešavanje familije čipa za implementaciju dizajna

Po završetku kreiranja projekta otvara se glavni prozor Quartus Prime alata. Sada je potrebno kreirati novi Qsys fajl u kom će biti opisan naš jednostavni procesorski sistem. Platform Designer se pokreće komandom ***File → New → Qsys System File***.

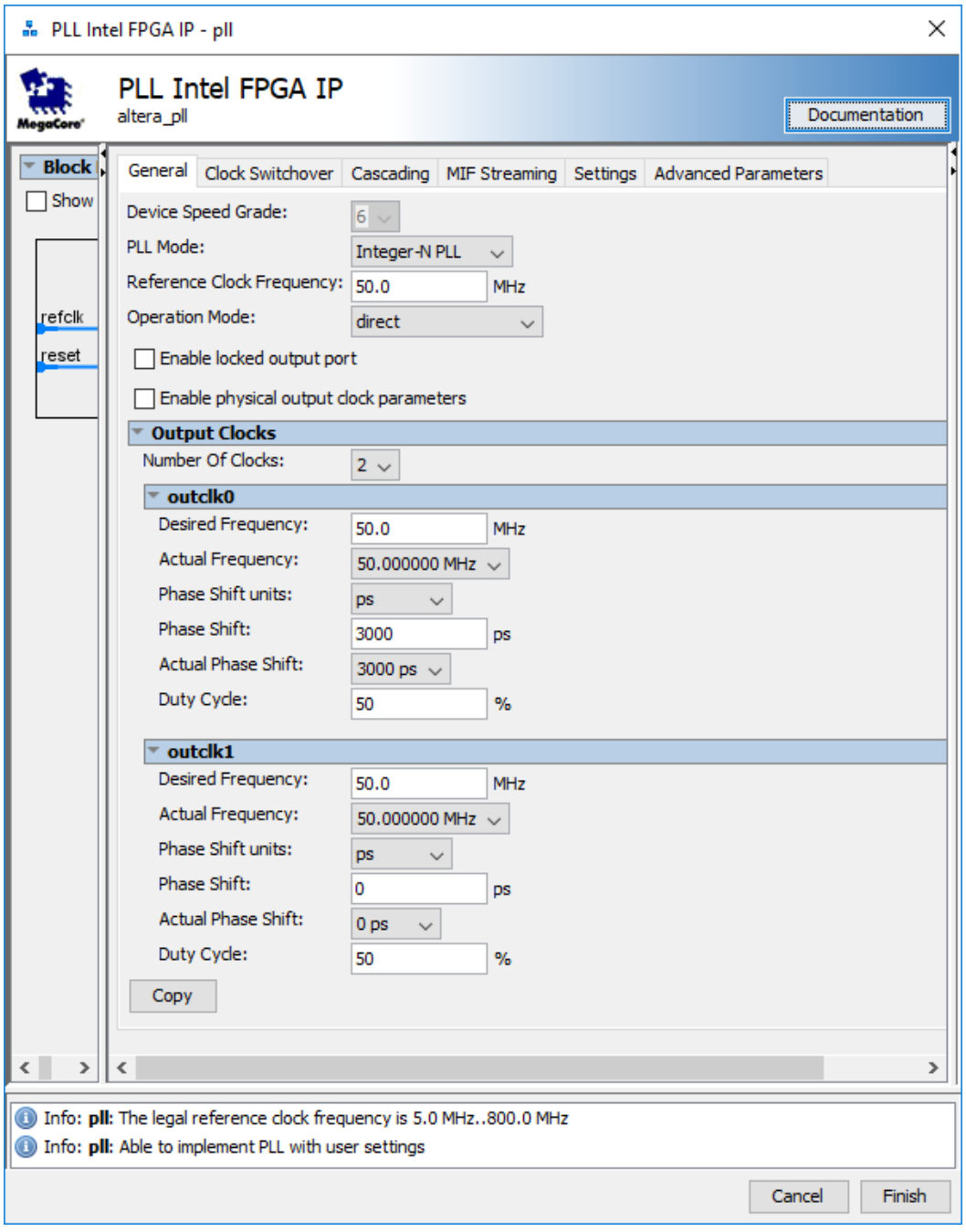
Po pokretanju Platform Designer-a otvara se prozor prikazan na Slici 5. U centralnom delu se nalazi opis sistema koji se kreira s tim što na početku postoji samo jedna komponenta koja predstavlja eksterni signal takta. Duplim klikom na ovu komponentu sa desne strane se otvaraju podešavanja i može se promeniti učestanost eksternog signala takta. Kako na razvojnoj platformi DE1-SoC postoji oscilator koji radi na učestanosti 50MHz potrebno je podesiti vrednost učestanosti signala takta na ovu vrednost. Desnim klikom na komponentu otvara se čitav niz podešavanja. Promeniti naziv komponente u clk_50MHz.



Slika 5. Početni prozor pri pokretanju Platform Designer alata

SDRAM memorija koja se nalazi na čipu zahteva takt od 50MHz, takođe bilo bi dobro i da ostatak sistema radi na istoj učestanosti kako bi se izbegli problemi usled sinhronizacije. Kako bi SDRAM memorija i procesorski deo sistema bili sinhronizovani potrebno je obezbediti da takt koji se vodi na procesorski deo sistema prednjači 3 ns u odnosu na takt koji se vodi na SDRAM modul. Ovo se obezbeđuje korišćenjem PLL modula. U okviru biblioteke komponenti koja se nalazi sa leve strane Platform Designer prozora postoje gotove PLL komponente koje se mogu uključiti u sistem. Za potrebe ovog primera korišćena je **PLL Intel FPGA IP** komponenta koja se nalazi u ***Basic Functions→ Clocks; PLLs and Reset→ PLL→ PLL Intel FPGA IP***. Komponenta se dodaje duplim klikom. Po dodavanju komponente ona se pojavljuje u odeljku za opis sistema i otvara se dijalog za podešavanje parametara. Kako je ulazna učestanost 50MHz to je potrebno podesiti za ***Reference Clock Frequency*** parametar. Izlaz ***locked*** neće biti korišćen tako da se ova opcija može isključiti. Kako su potrebna 2 signala takta (jedan interni za komponente procesorskog sistema i jedan koji će se voditi ka SDRAM modulu)

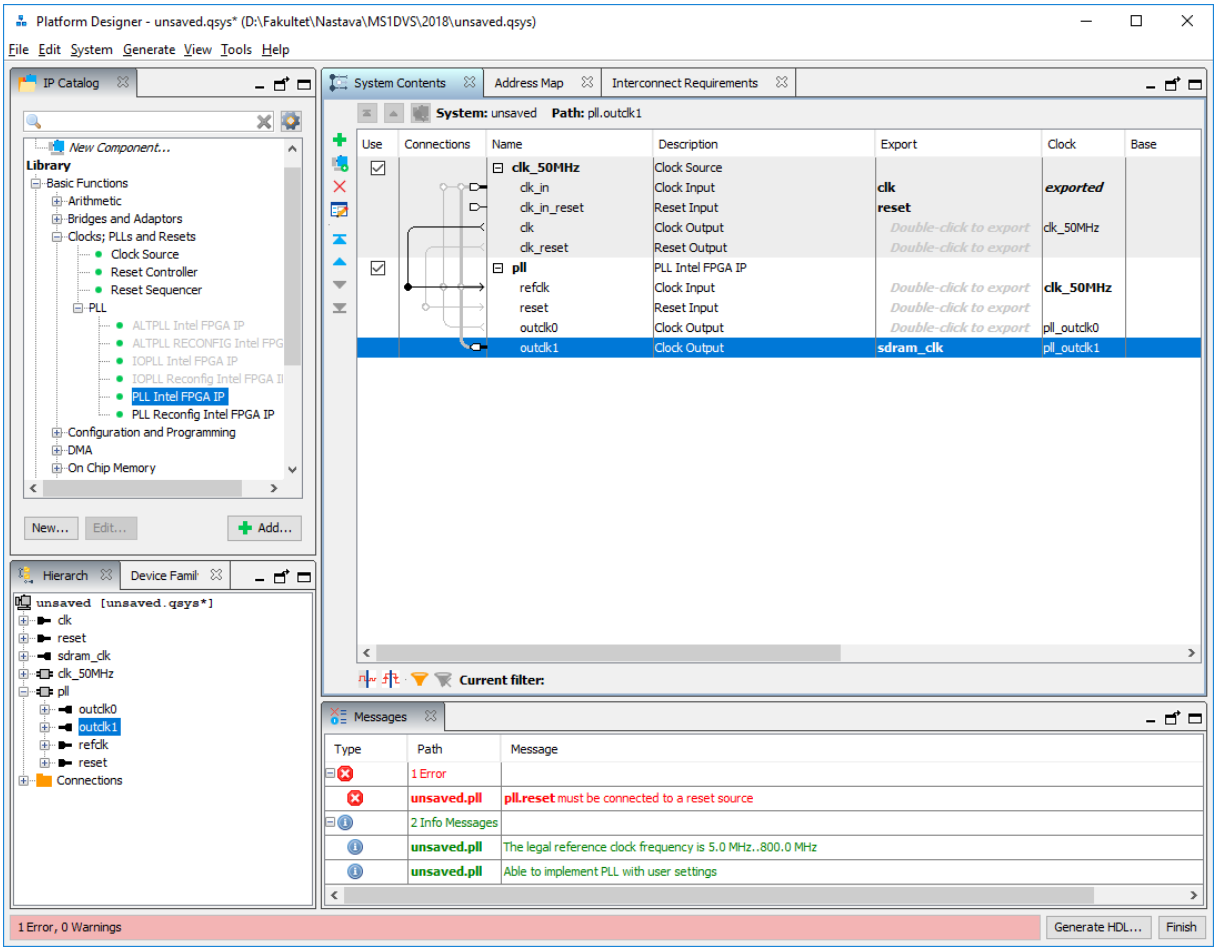
potrebno je aktivirati 2 izlaza postavljanjem parametra ***Number of Clocks*** na 2. Za svaki izlazni signal se zadaje željena izlaznu učestanost (u našem slučaju 50MHz) i alat automatski računa odgovarajuće faktora deljenja i množenja kao i najpribližniju učestanost željenoj koja se može realizovati. Kako je potrebno da takt koji se vodi na procesorski sistem prednjači 3ns u odnosu na takt koji se vodi na SDRAM module to je potrebno za ovaj takt (recimo outclk0) podesiti fazno kašnjenje **3000ps**. Podešavanja PLL modula su prikazana na Slici 6. Klikom na ***Finish*** se završava konfigurisanje PLL modula.



Slika 6. Podešavanje PLL modula u Platform Designer-u

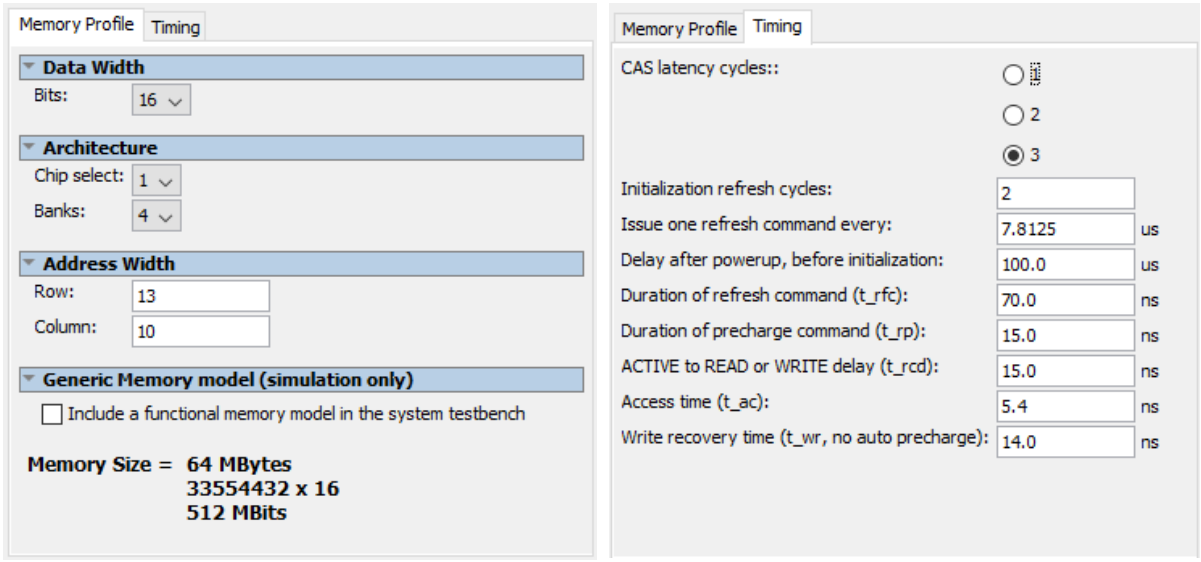
Trenutno u sistemu postoje modula za takt i PLL međutim oni nisu međusobno povezani. Moguće konekcije između kompatibilnih signala u Platform Designer-u su prikazane svetlo sivom bojom. Klikom na odgovarajući beli kružić ostvaruje se konekcija između dve linije i one postaju crne. Veza se može raskinuti ponovnim klikom na crni kružić. Ulazni signal takta u PLL modul ***refclk*** se dovodi spolja i predstavlja izlazni signal ***clk*** komponente izvora takta ***clk_50MHz***. Izlaz PLL-a **outclk0** će biti

korišćen interno unutar procesorskog sistema za sve komponente koje zahtevaju signal takta dok je izlaz **outclk1** potrebno izvesti van sistema jer će biti korišćen za SDRAM modul. Izbacivanje odgovarajućeg signala u interfejs sistema je omogućeno duplim klikom u kolonu ***Export*** i upisom odgovarajućeg naziva signala. Podesiti naziv izvezenog signala na ***sdram_clk***. Prozor Platform Designer-a bi sada trebalo da izgleda kao na Slici 7.



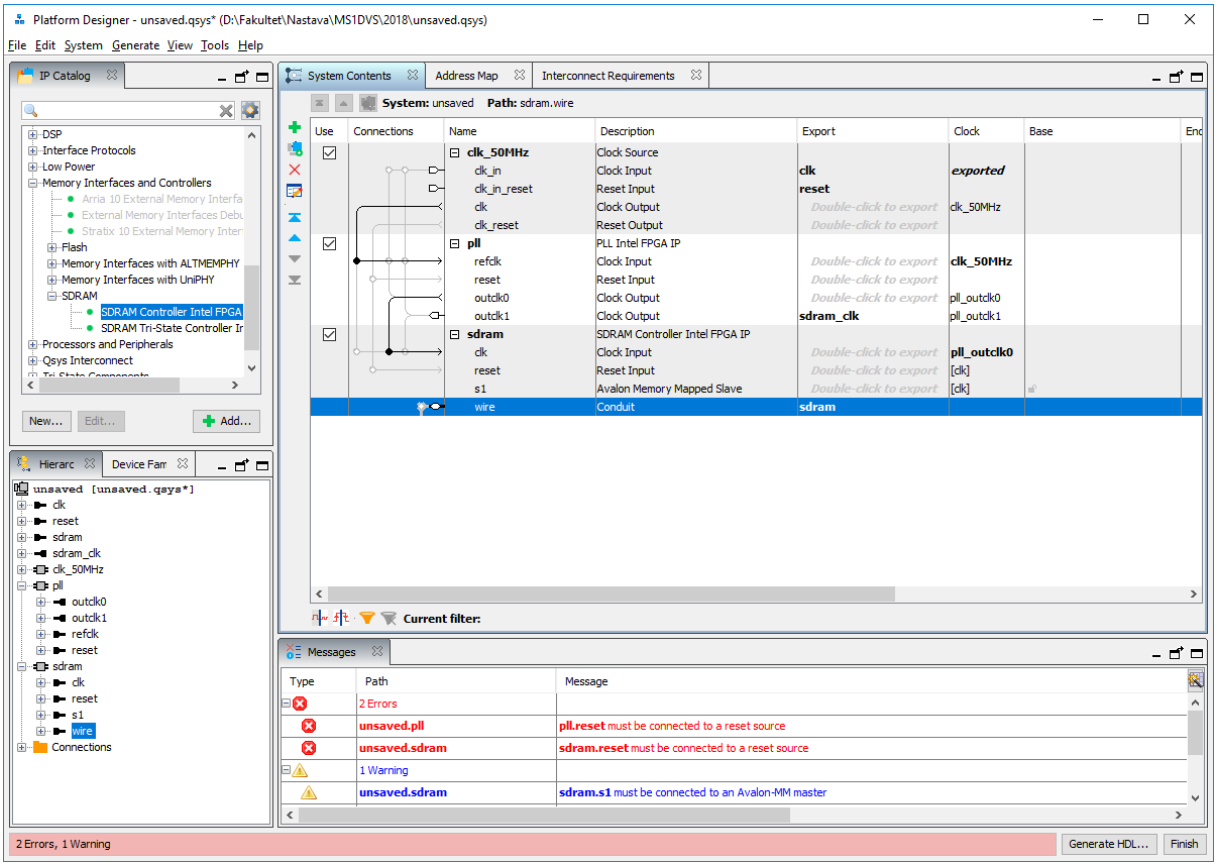
Slika 7. Komponenta PLL povezana u sistem

Kako će se podaci i instrukcije procesora čuvati u eksternoj SDRAM memoriji potrebno je u sistem dodati odgovarajući memorijski kontroler koji će obezbediti vezu kao ovoj memoriji. Isto kao za PLL i za memorijski kontroler postoji gotova komponenta koja se može uključiti u sistem. Kako koristimo SDRAM modul onda se taj kontroler može naći u biblioteci komponenti pod ***Memory Interfaces and Controllers → SDRAM → SDRAM Controller Intel FPGA IP***. Po pokretanju se otvara dijalog za podešavanje parametara koji je potrebno postaviti kao na Slici 8.



Slika 8. Podešavanje SDRAM kontrolera

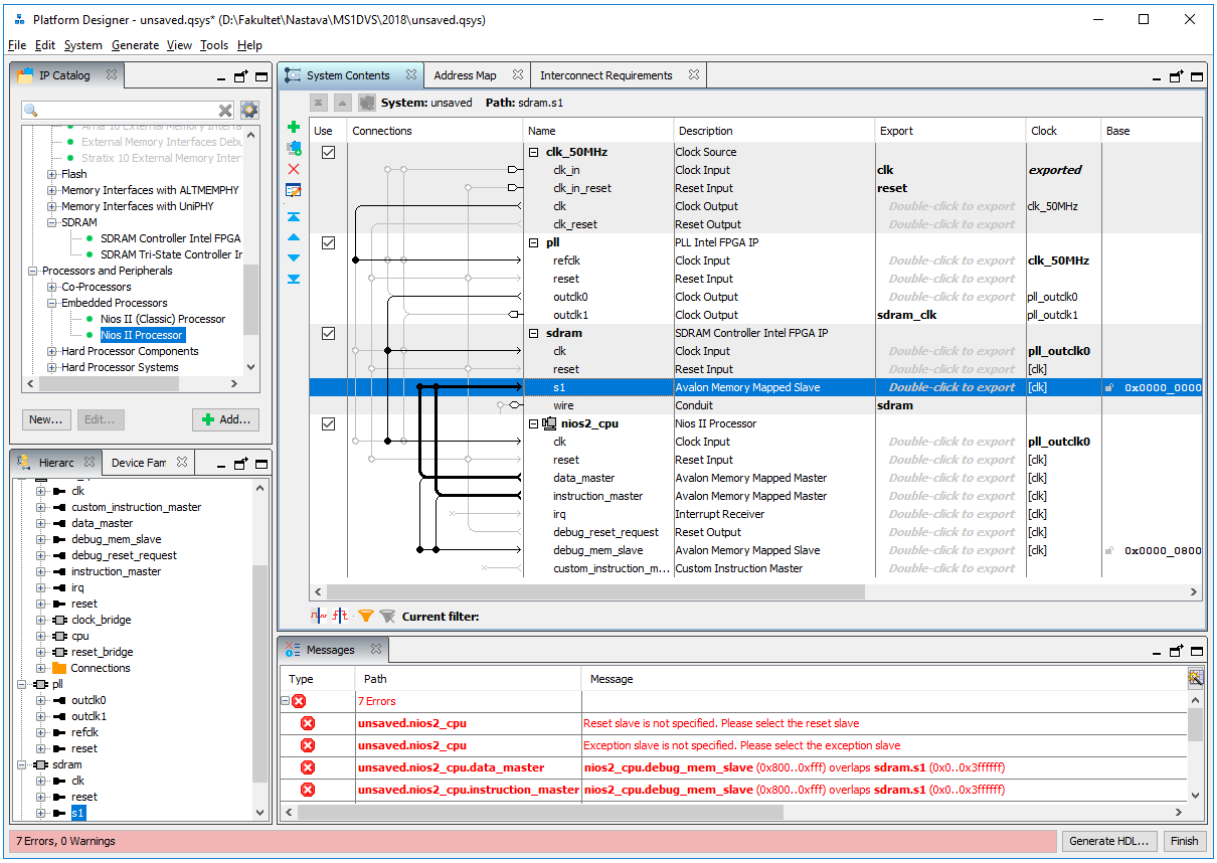
Za ulazni takt SDRAM kontrolera se koristi izlazni signal outclk0 PLL komponente. Ovaj signal se koristi kao takt i za ostale komponente sistema. Signalne linije koje se vode direktno ka SDRAM modulu označene su kao ***wire*** i potrebno ih je izvesti u interfejs procesorskog sistema kao što je prikazano na Slici 9.



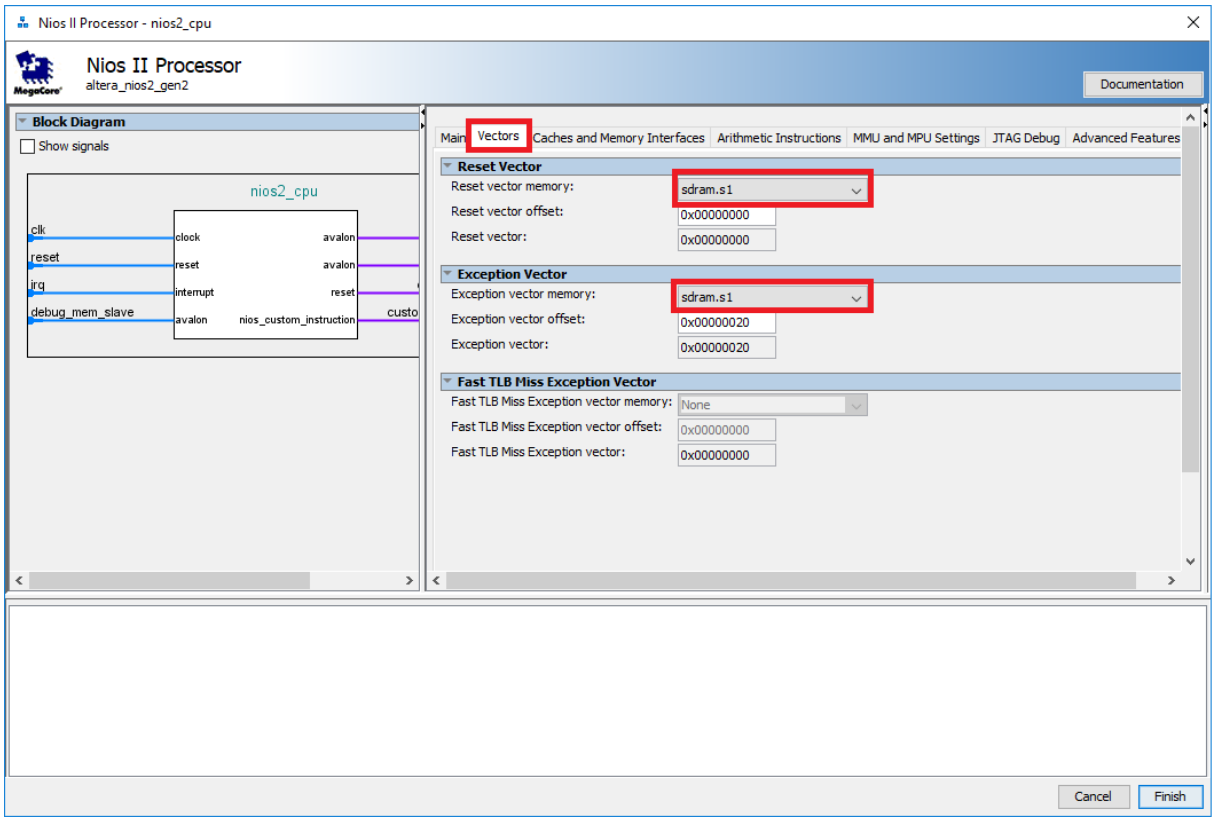
Slika 9. Povezivanje SDRAM kontrolera u sistem

Nios2 soft procesorsko jezgro takođe postoji u biblioteci komponenti pod ***Processors and Peripherals → Embedded Processors → Nios II Processor.*** Pojavljuje se prozor za konfigurisanje

procesora. Na raspolaganju je više verzija soft procesora rastuće složenosti. U *Lite* verziji softvera na raspolaganju je najjednostavnija verzija procesora. Složenije verzije se mogu koristiti s tim što se u tom slučaju generiše vremenski ograničen programski fajl koji posle određenog vremena prestaje sa radom. Odabrati Nios II/e verziju i zatvoriti dijalog za konfiguraciju. Potrebno je povezati signal takta na ulaz procesora. Kako će se i instrukcije i podaci nalaziti u eksternoj SDRAM memoriji potrebno je povezati ***data_master*** i ***instruction_master*** izlaze sa slave interfejsom ***s1*** memorijskog kontrolera kao što je prikazano na Slici 10. Kada je procesor povezan sa memorijom mogu se podesiti adrese **reset** i ***exepction*** vektora odabirom ***sdram.s1*** iz padajuće liste za odgovarajuću memoriju što je prikazano na Slici 11. Tačne adrese će biti određene na samom kraju projektovanja sistema.



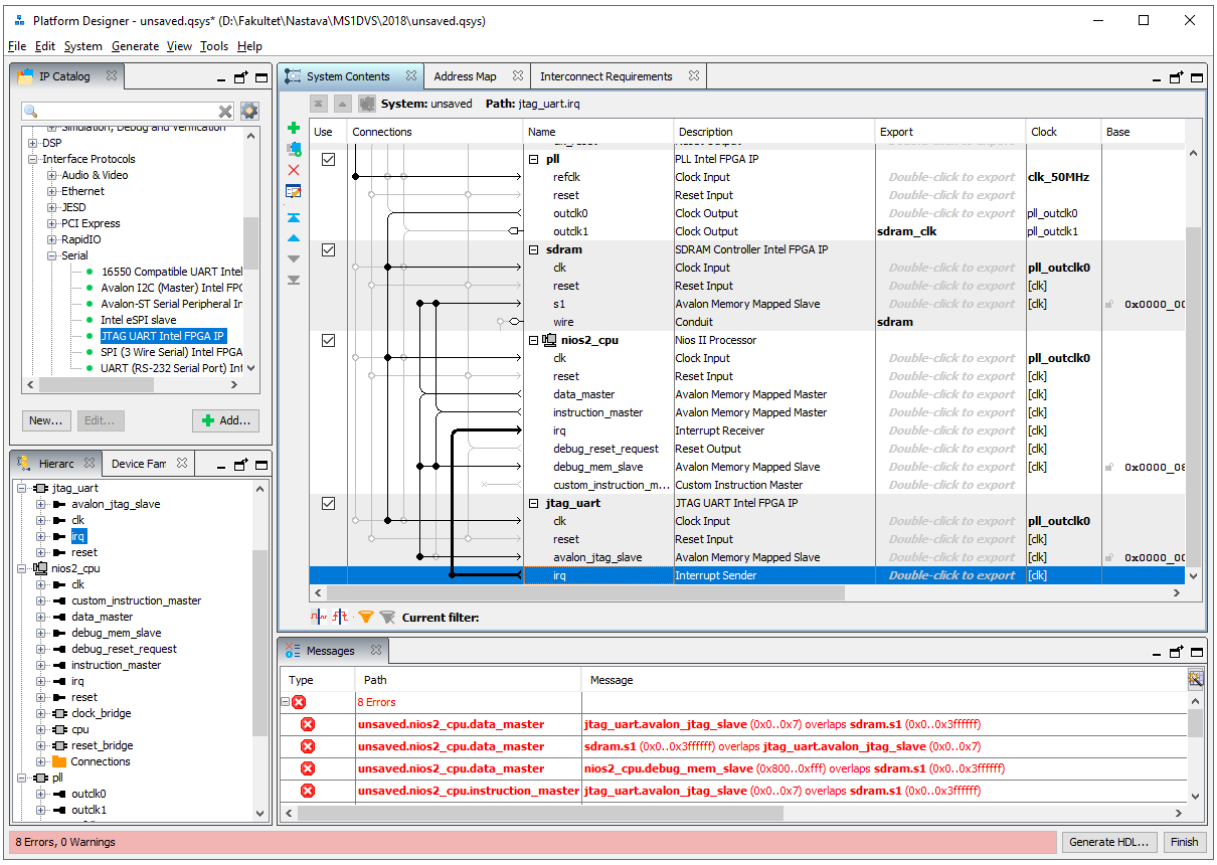
Slika 10. Povezivanje Nios 2 procesora u sistem



Slika 11. Podešavanje memorije za instrukcije i podatke

Kako bi se obezbedila komunikacija sa host računarom i programiranje procesora potrebno je u sistem dodati periferiju JTAG UART-a koja se nalazi pod ***Interface Protocols → Serial →JTAG UART Intel FPGA IP***.

Ostaviti podrazumevana podešavanja za UART. Potrebno je dovesti takt i povezati slave ulaz ***avalon_jtag_slave*** na ***data_master*** magistralu procesora. Kako periferija UART-a generiše prekide potrebno je povezati liniju prekida UART-a ***irq*** sa odgovarajućim ulazon na procesoru. Povezivanje UART-a u sistem je prikazano na Slici 12.



Slika 12. Povezivanje JTAG UART-a u sistem

U sistemu prikazanom na Slici 2. postoji nekoliko eksternih komponenti koje nisu povezane direktno na Avalon magistralu procesora. Veza sa ovim eksternim periferijama se najlakše ostvaruje preko paralelnih portova koji mogu biti ulazni, izlazni ili bidirekcioni. Komponenta paralelnog porta se nalazi pod ***Processors and Peripherals → Peripherals → PIO (Parallel I/O) Intel FPGA IP***. Kao što je prikazano na Slici 3. u ovom primeru će biti potrebno 5 ovakvih komponenti (3 ulazne i 2 izlazne).

Konfiguracija svake komponente data je u listi ispod:

1. ***pi_buttons***
   - width: 4
   - direction: input
   - Synchronously capture
   - Edge type: Risig
   - Enable bit-clearing for edge capture register
   - Generate IRQ
   - IRQ type: edge
   - Naziv izvezenog signala: pi_buttons
2. ***pi_switch_num***
   - width: 8
   - direction: input
   - Naziv izvezenog signala: pi_switch_num
3. ***pi_switch_sel***
   - width: 1
   - direction: input

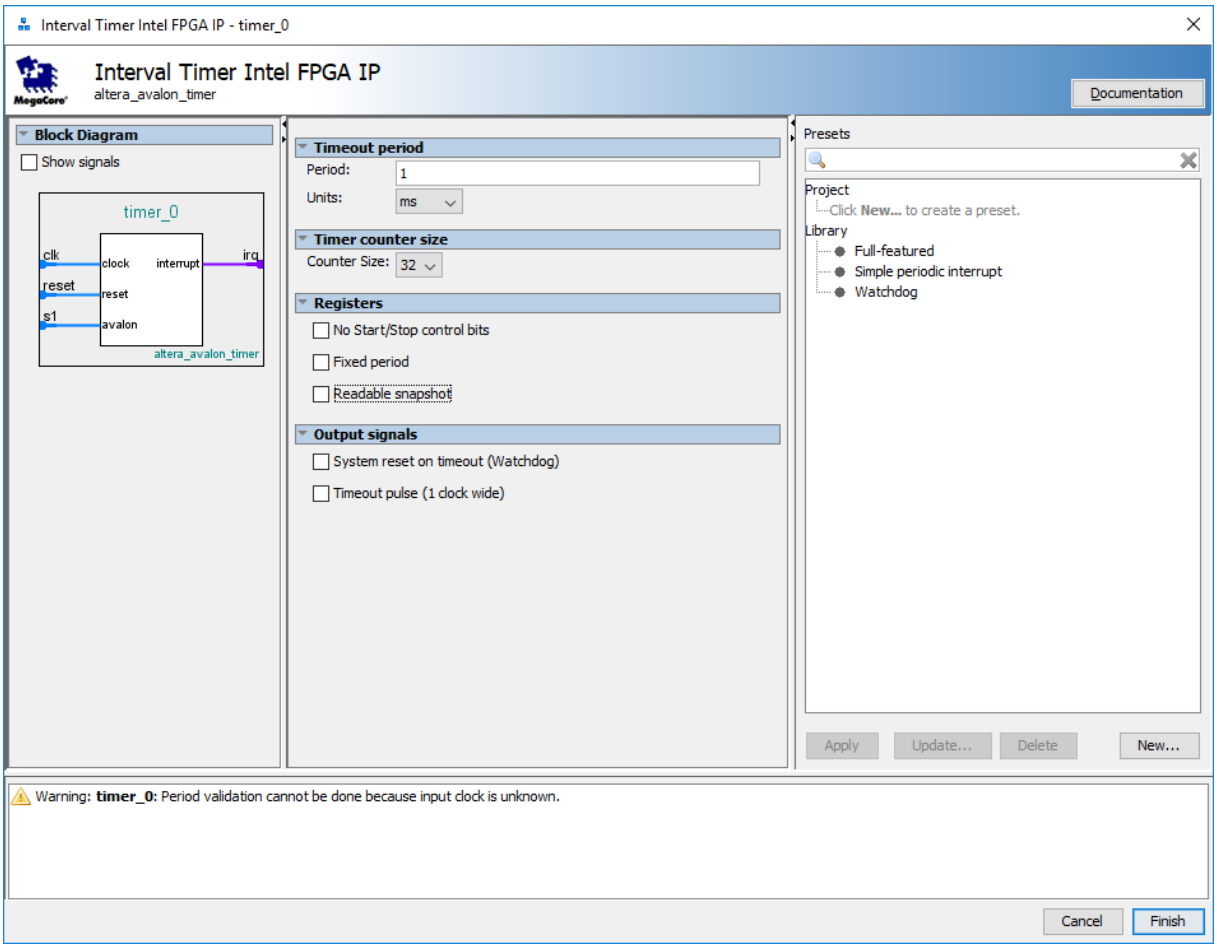
- Naziv izvezenog signala: pi_switch_sel

4. ***po_bcd***
   - width: 12
   - direction: output
   - Naziv izvezenog signala: po_bcd
5. ***po_pwm***
   - width: 8
   - direction: output
   - output port reset value: 0xff
   - Naziv izvezenog signala: po_pwm

Obavezno izvesti vam sistema (*export*) odgovarajuće ***conduit*** signale paralelnih portova.

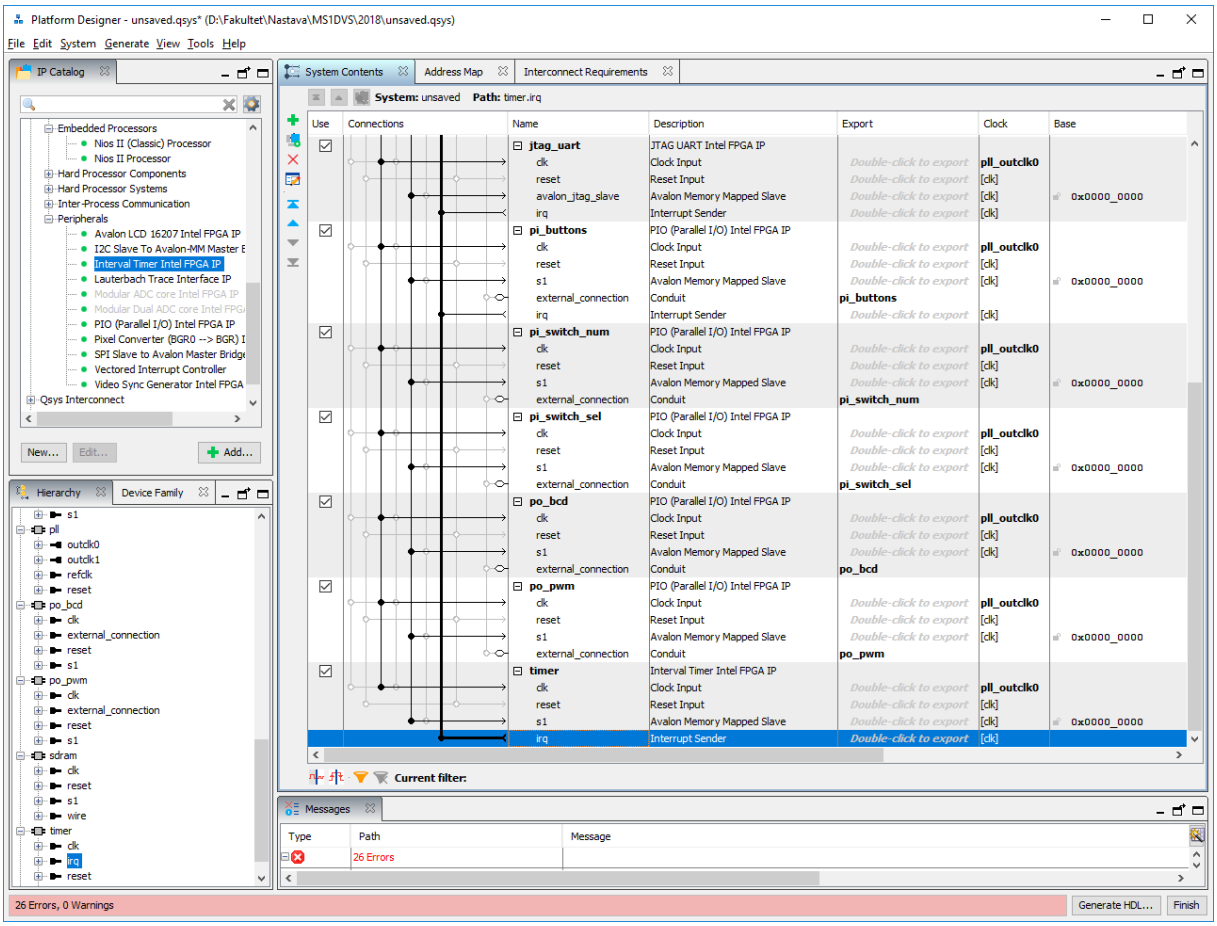
Potrebno je dovesti odgovarajući signal takta na svaki od paralelnih portova i povezati slave interfejs ***s1*** na ***data_master*** magistralu procesora. Kako paralelni port ***pi_buttons*** ima mogućnost generisanja prekida potrebno je povezati njegovu prekidnu liniju ***irq*** sa procesorom.

Pri opisu sistema je navedeno da se osvetljaj LED displeja menja periodično od najslabijeg do najjačeg i obrnuto. Osvetljaj se kontroliše pomoću modula PWM-a a promena osvetljaja menjanjem faktora ispunjenosti pwm-a. Potrebno je obezbediti u sistemu periodični događaj pri kome će se menjati faktor ispunjenosti pwm-a. Učestanost ovog događaja određuje brzinu promene osvetljaja displeja i treba obezbediti da se može menjati putem tastera. Periferija koja obezbeđuje ovu funkcionalnost je tajmer koji se nalazi pod ***Processors and Peripherals → Peripherals → Interval Timer Intel FPGA IP***. Podesiti tajmer kao na Slici 13.



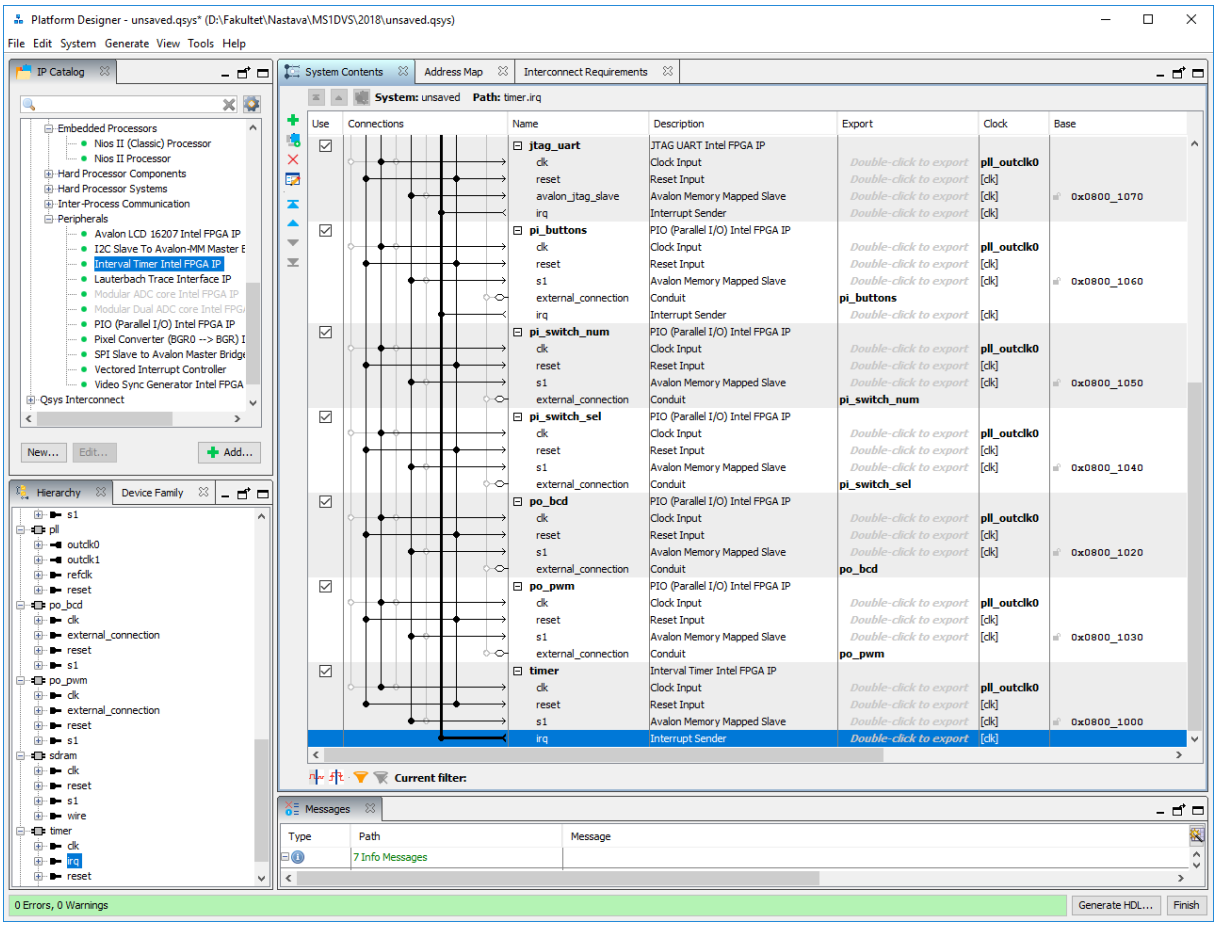
Slika 13. Podešavanje periferije tajmera

Potrebno je povezati signal takta, slave interfejs tajmera sa data_master magistralom i prekidnu liniju tajmera sa procesorom kao što je prikazano na Slici 14.



Slika 14. Povezivanje periferije tajmera u procesorski sistem

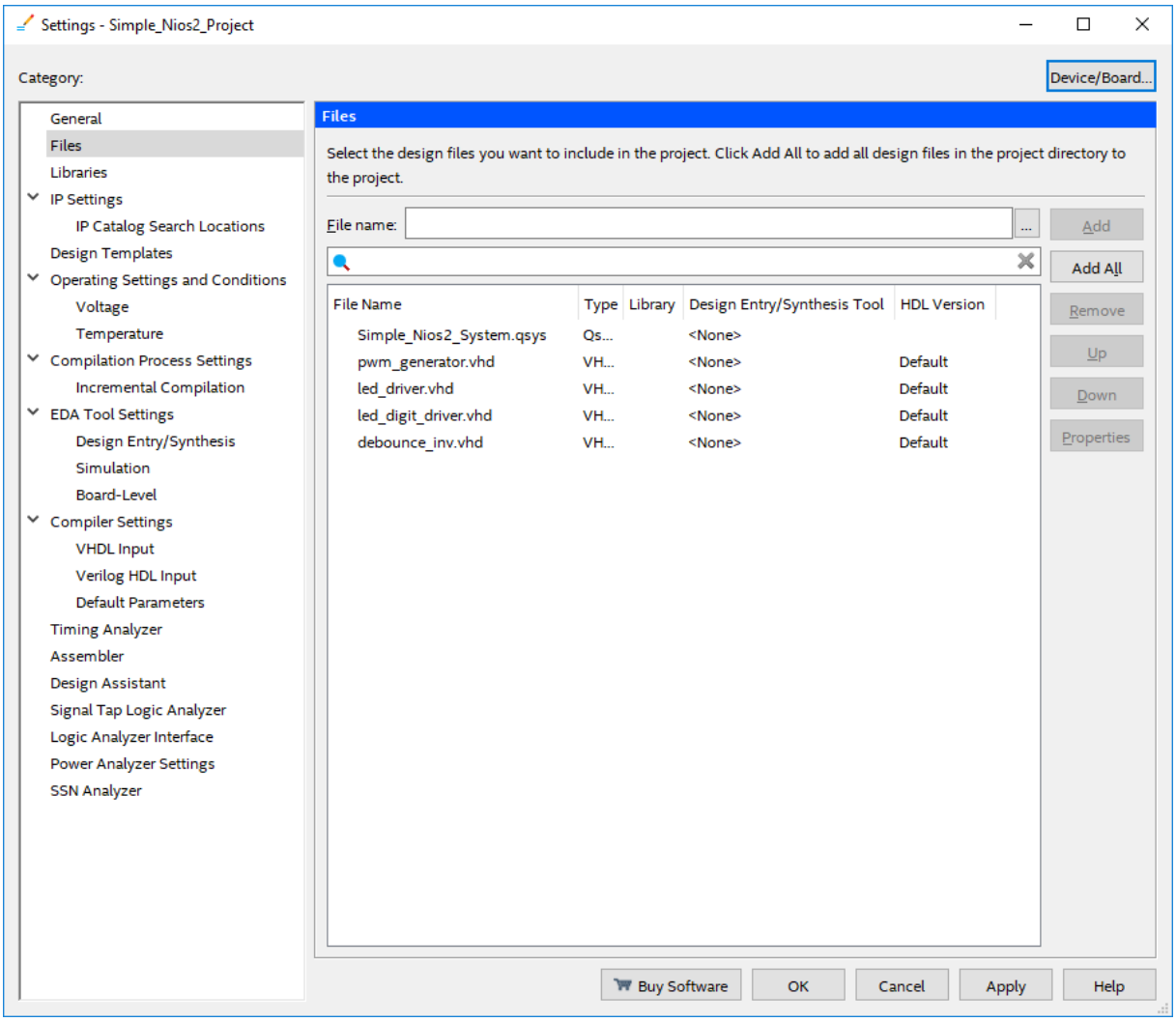
Trenutna situacija bi trebalo da bude kao na Slici 14. sa ukupno 26 grešaka. Ako postoji više nastavite dalje sa čitanjem kako biste razrešili ovih 26 a posle je potrebno rešiti one što preostanu. Kada se pogledaju greške većina njih se odnosi na preklapanje adresnog opsega. Ovo je potpuno logično pošto su bazne adrese svih komponenti u sistemu postavljene na 0. Kako bi se obezbedilo raspoređivanje periferija u adresni prostor procesora bez preklapanja potrebno je dodeliti svim periferijama validne bazne adrese. Bazne adrese se mogu automatski generisati pomoću opcije ***System → Assign Base Addresses***. Kada se ovo uradi može se primetiti da bazne adrese više nisu 0 i da je broj grešaka spao na 10. Ako se dalje analiziraju greške može se primetiti da se one uglavnom odnose na nepovezivanje signala reseta. Signali za reset svih komponenti se mogu automatski povezati kreiranjem jedinstvene reset mreže pomoću opcije ***System → Create Global Reset Network***. Po izvršenju ove komande trebalo bi da nestanu sve greške i upozorenja kao što je prikazano na Slici 15.



Slika 15. Kompletiran procesorski sistem bez grešaka i upozorenja

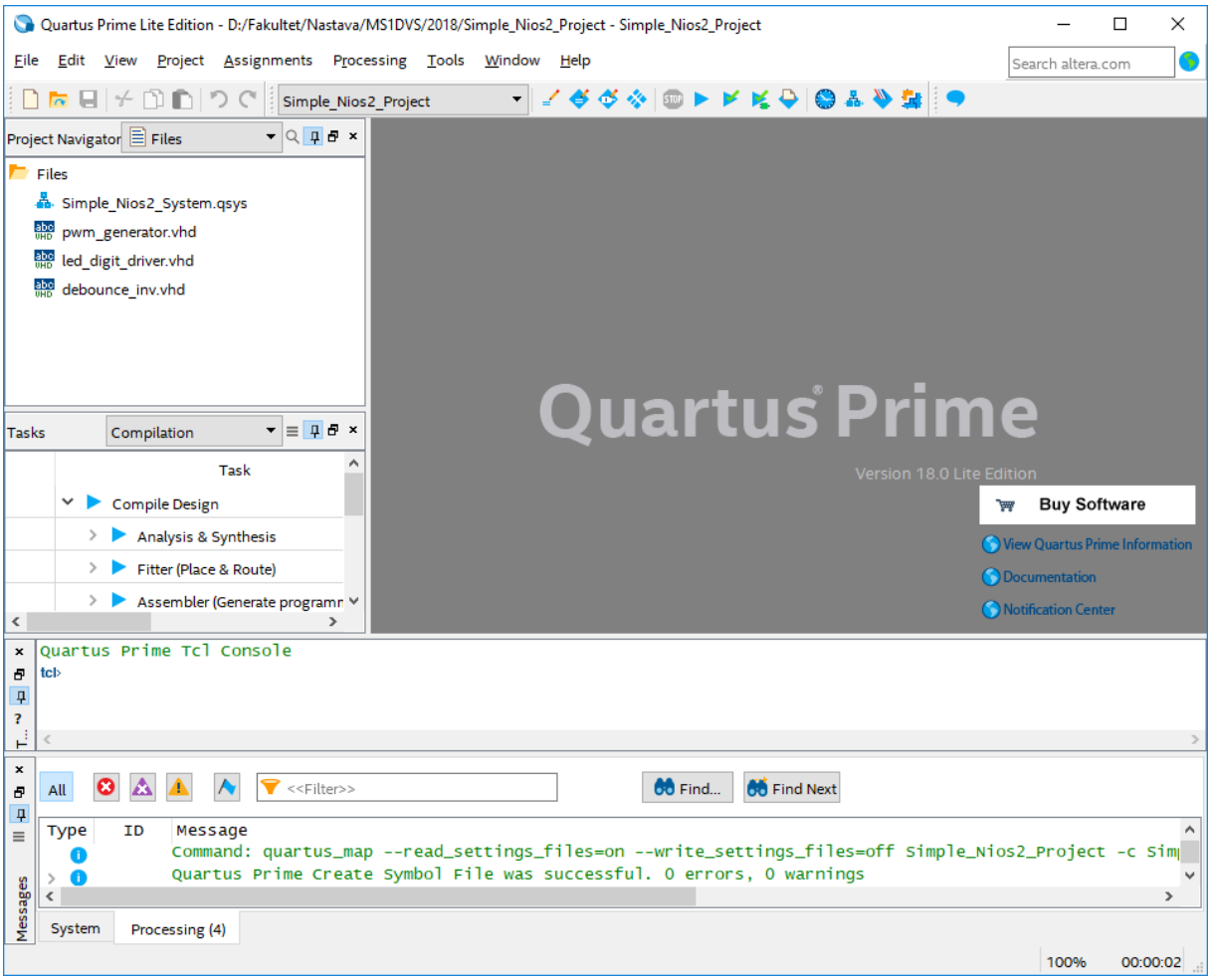
Po kompletiranju procesorskog sistema potrebno ga je sačuvati pod imenom ***Simple_Nios2_System*** i pokrenuti generisanje odgovarajućih fajlova ***.sopcinfo*** i ***.bsf*** izborom opcije ***Generate HDL...*** u okviru menija ***Generate***, podesiti jezik na VHDL. Na kraju isključiti Platform Designer i vratiti se u okruženje Quartus Prime-a.

Na stranici predmeta (http://tnt.etf.rs/~ms1dvs/Vezbe/MS1DVS_prvi_cas.zip) se nalaze kodovi eksternih komponenti sistema ***led_digit_driver.vhd, pwm_generator.vhd, debounce_inv.vhd i pin_assignment_DVS_DE1_SoC.tcl***. Potrebno je iskopirati ove fajlove u direktorijum gde se nalazi projekat. Dodatni fajlovi se uključuju u projekat pomoću opcije iz glavnog prozora Quartus Prime-a ***Project → Add/Remove Files in Project…*** čime se otvara dijalog za dodavanje novih fajlova. Klikom na dugme ***Add All*** dodaju se svi relevantni fajlovi iz direktorijuma u kom se nalazi projekat. Po dodavanju fajlova dijalog bi trebalo da izgleda kao na Slici 16.



Slika 16. Dodavanje novih fajlova u projekat

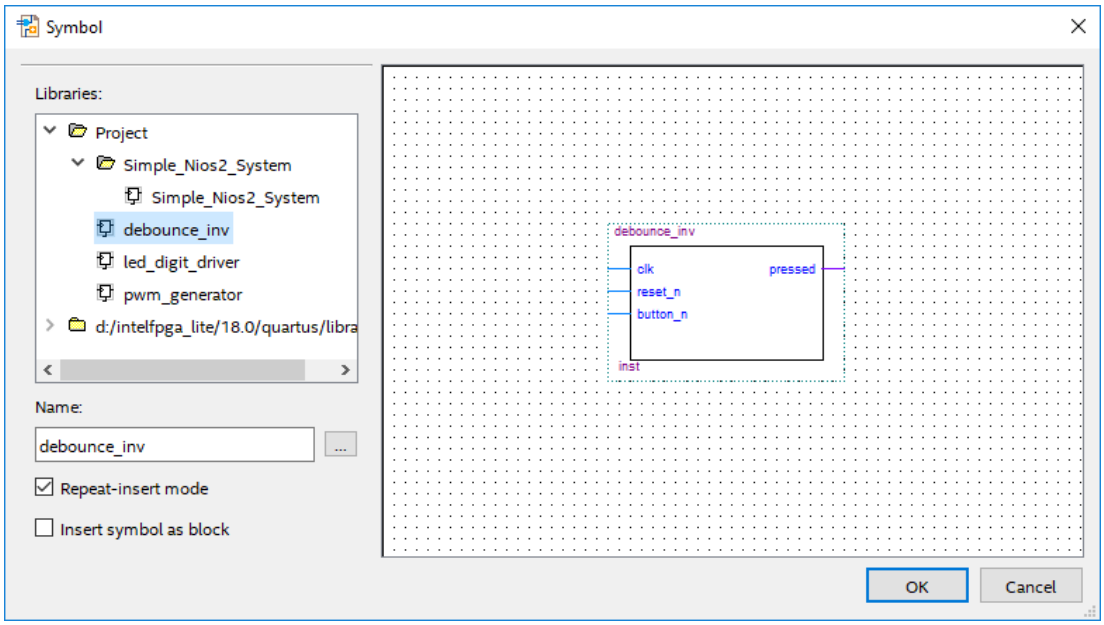
Dodati fajlovi su sada izlistani u okviru stranice ***Files*** Project Navigatora kao što je prikazano na Slici 17.



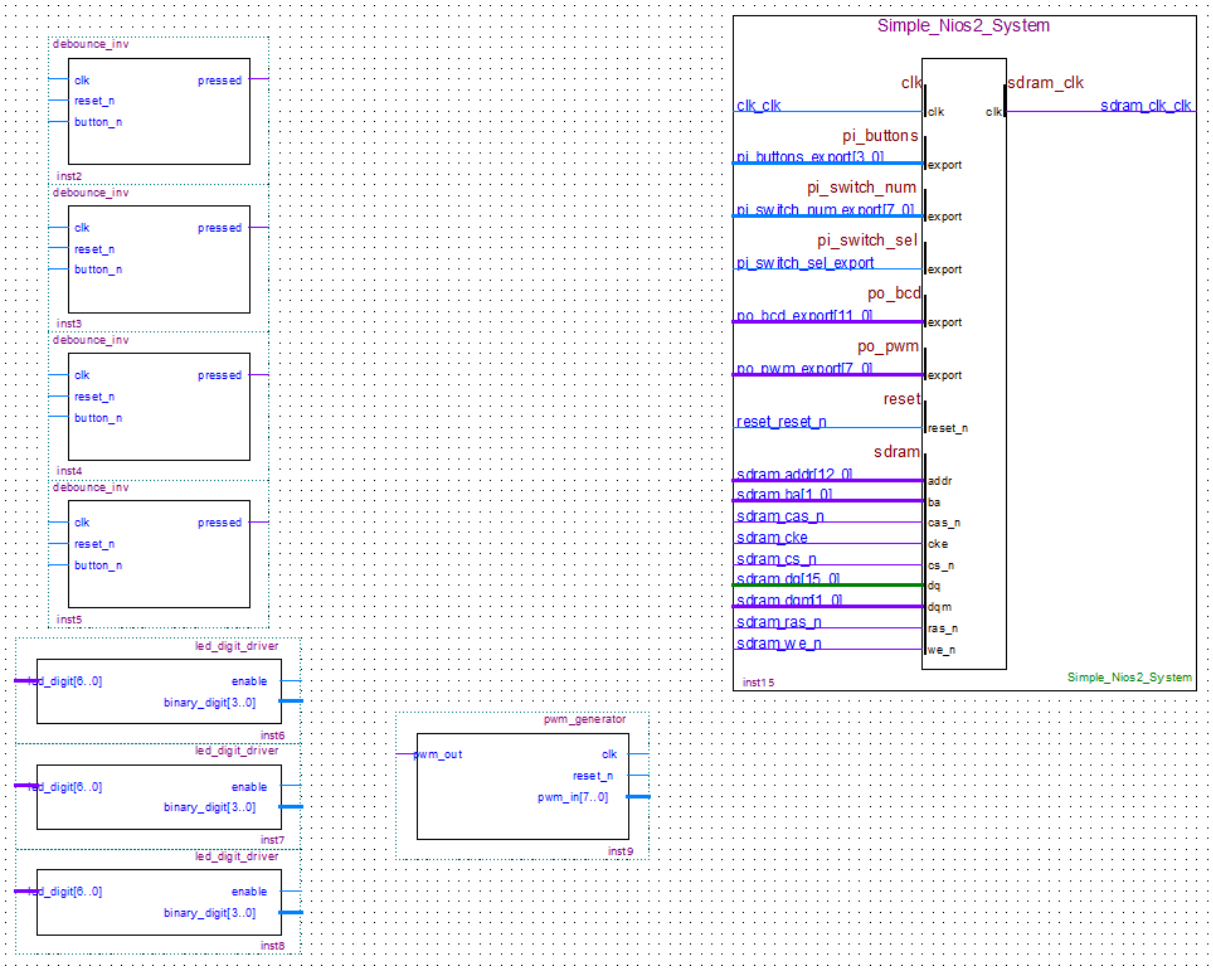
Slika 17. Dodati fajlovi

Sledeći korak je povezivanje svih komponenti sistema u jednu celinu koja je prikazana na Slici 2. Prvi način je kreiranje ***top level*** vhd modula u kome će biti uključene sve komponente. Interfejs komponente procesorskog sistema se nalazi na putanji ***Simple_Nios2_System/synthesis/Simple_Nios2_System.vhd***. Drugi način je kreiranje ***top level schematic***-a gde su sve komponente instancirane kao blokovi u grafičkom interfejsu i povezane linijama. Kako je prvi način prilično poznat iz dosadašnjih projekata u ovom projektu će povezivanje sistema biti obavljeno na drugi način.

Potrebno je kreirati novi fajl sa ***File → New → Block Diagram/Schematic File*** i sačuvati ga pod nazivom ***Simple_Nios2_Project***. Pre spajanja sistema potrebno je kreirati fajlove sa simbolima eksternih komponenti koje se nalaze u zasebnim vhdl fajlovima. Fajlovi sa simbolima se mogu kreirati tako što se dok je selektovan odgovarajući modul u Project Navigator-u klikne desni klik i odabere opcija ***Create Symbol Files for Current File***. Ovo je potrebno uraditi za sve eksterne fajlove koji će biti korišćeni kao blokovi u schematic-u. Novi element se dodaje pomoću ***Symbol Tool***-a koji ima oznaku I kola i nalazi se na traci sa alatima u gornjem delu prozora. Klikom na ovaj alat otvara se dijalog za dodavanje novog elementa. Biblioteka sa našim komponentama se naziva ***Projects*** i njen sadržaj je prikazan na Slici 18. Klikom na OK započinje se dodavanje komponente klikom negde na prazan prostor. Alat za dodavanje simbola se isključuje pritiskom na taster ***Esc***. Potrebno je dodati sve komponente u sistem kao na Slici 19.

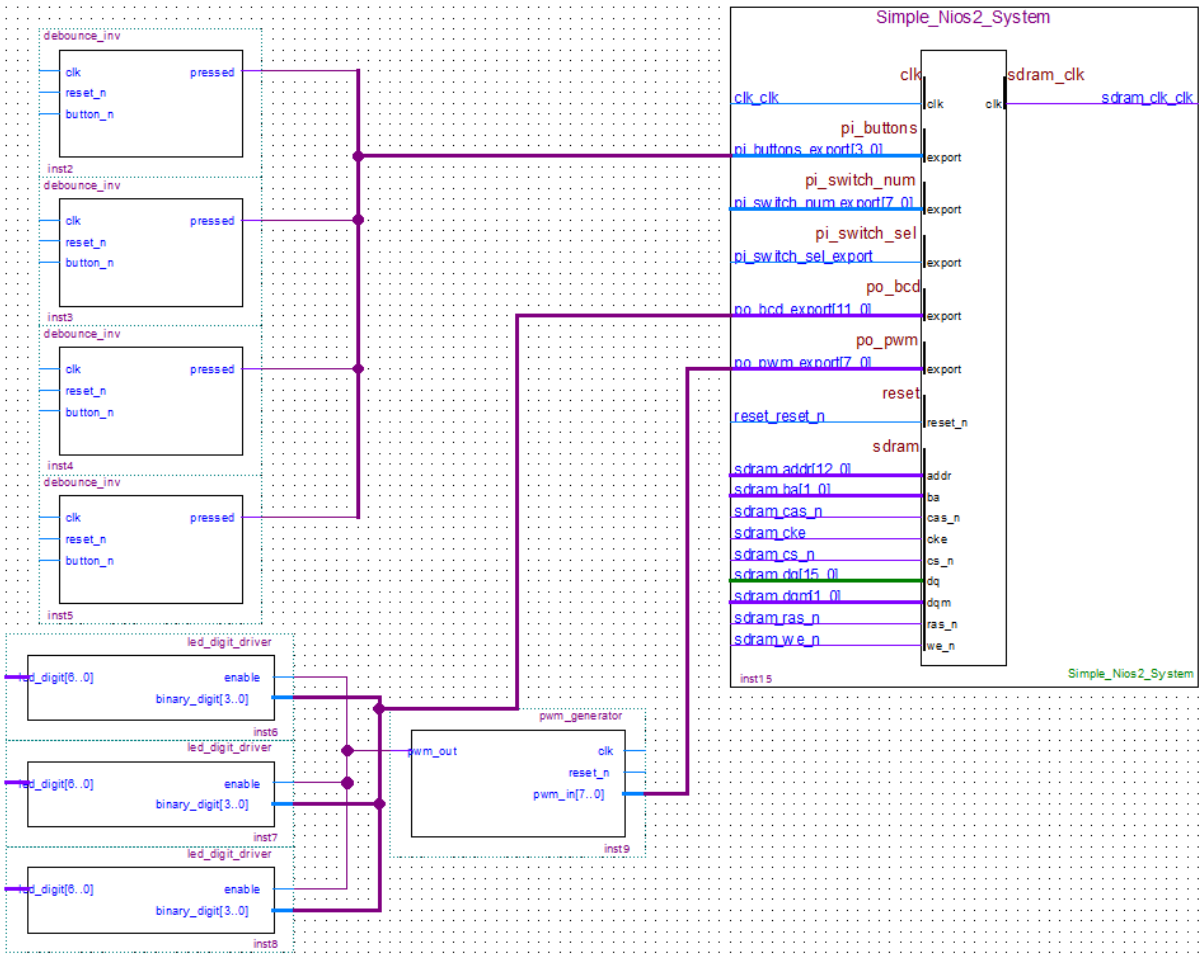


Slika 18. Sadržaj biblioteke simbola Project



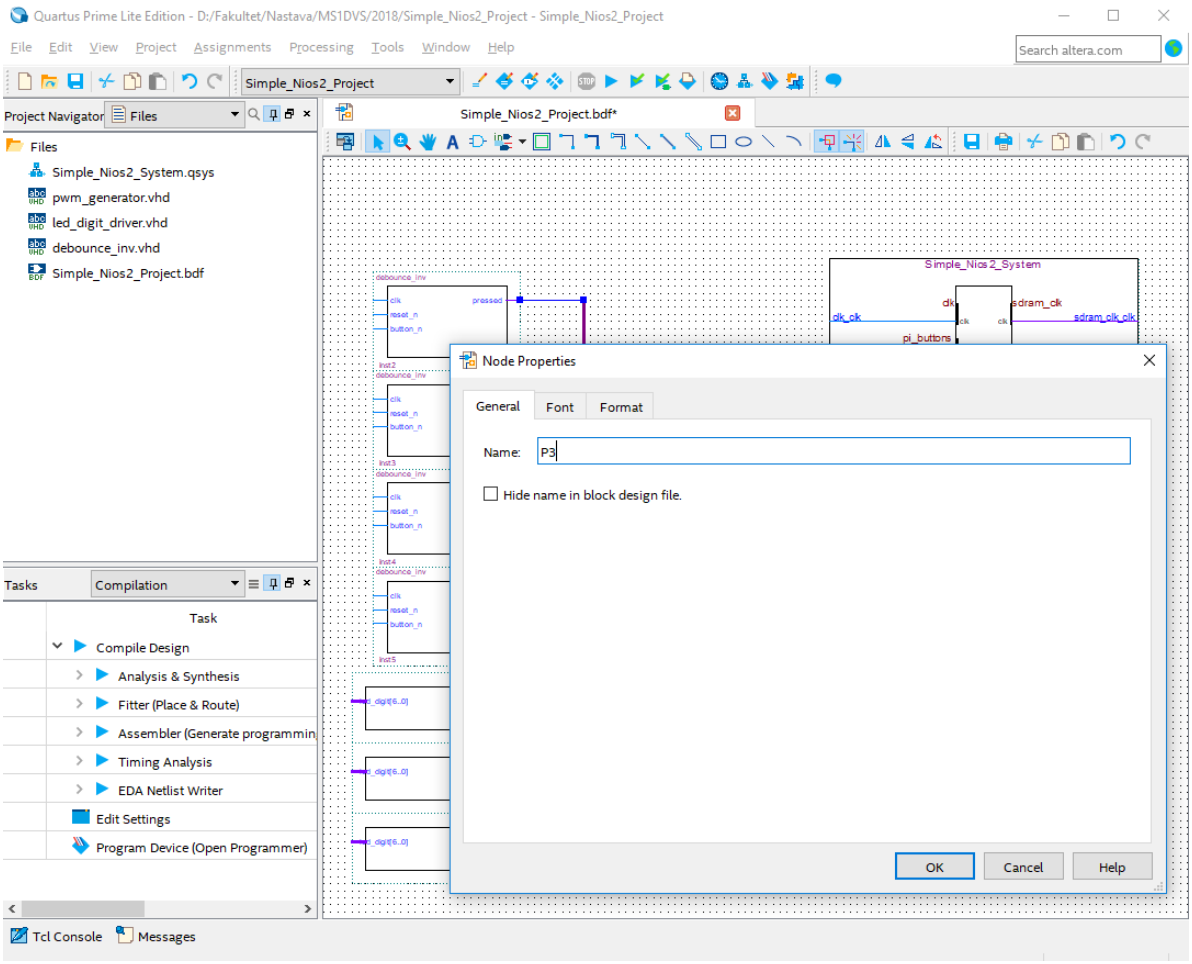
Slika 19. Sve komponente dodate u schematic

Povezivanje komponenti sistema je prikazano na Slici 20.

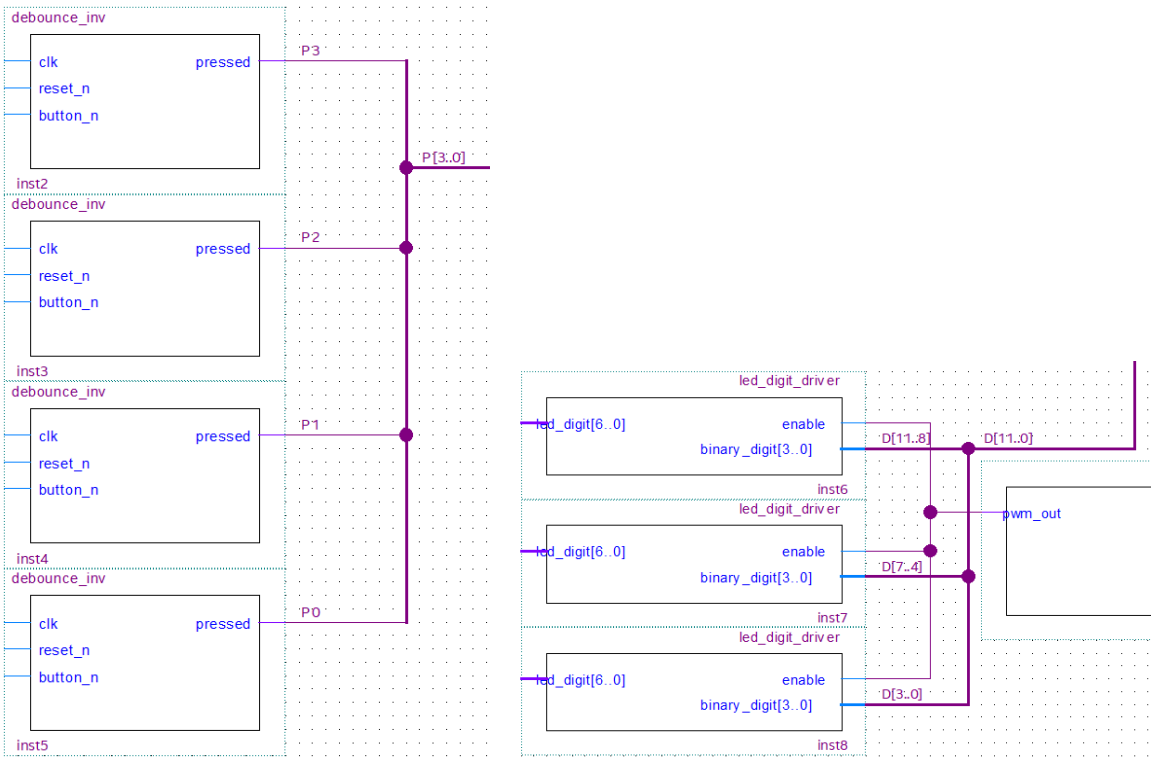


Slika 20. Povezivanje komponenti

Kako se 4 jednobitna signala (tanja linija) izlaza debounce_inv kola ***pressed*** spaja u jedan 4-bitni ulaz procesorski sistem potrebno je to obezbediti pri povezivanju. Signali se povezuju kao što je prikazano na Slici 20. pomoću linije ***Orthogonal Bus*** (deblja linija) način spajanje se definiše u navođenjem imena mreže. Potom je potrebno selektovati liniju koja izlazi iz odgovarajućeg kola i iz menija koji se pojavljuje posle desnog klika odabrati opciju ***Properties***. Otvara se dijalog u koji se može upisati ime mreže. Na Slici 21. je prikazano dodeljivanje imena ***P3*** (četvrti bit signala P) izlazu najvišeg ***debounce_inv*** kola. Potrebno je ponoviti isti postupak i za ostala 3 izlaza. Liniji koja ide direktno u procesorski sistem i koja predstavlja 4-bitni signal potrebno je dodeliti ime ***P[3..0]*** čime se definiše da se ona sastoji iz 4 jednobitna signala P3, P2, P1, P0 kao što je prikazano na Slici 22. Istim postupkom je potrebno obezbediti grananje 12-bitnog izlaza ***po_bcd*** na 3 ulaza *led_digit_driver* komponenti.



Slika 21. Dodeljivanje imena liniji veze



Slika 22. Spajanje jednobitnih signala u višebitni i grananje višebitnih magistrala

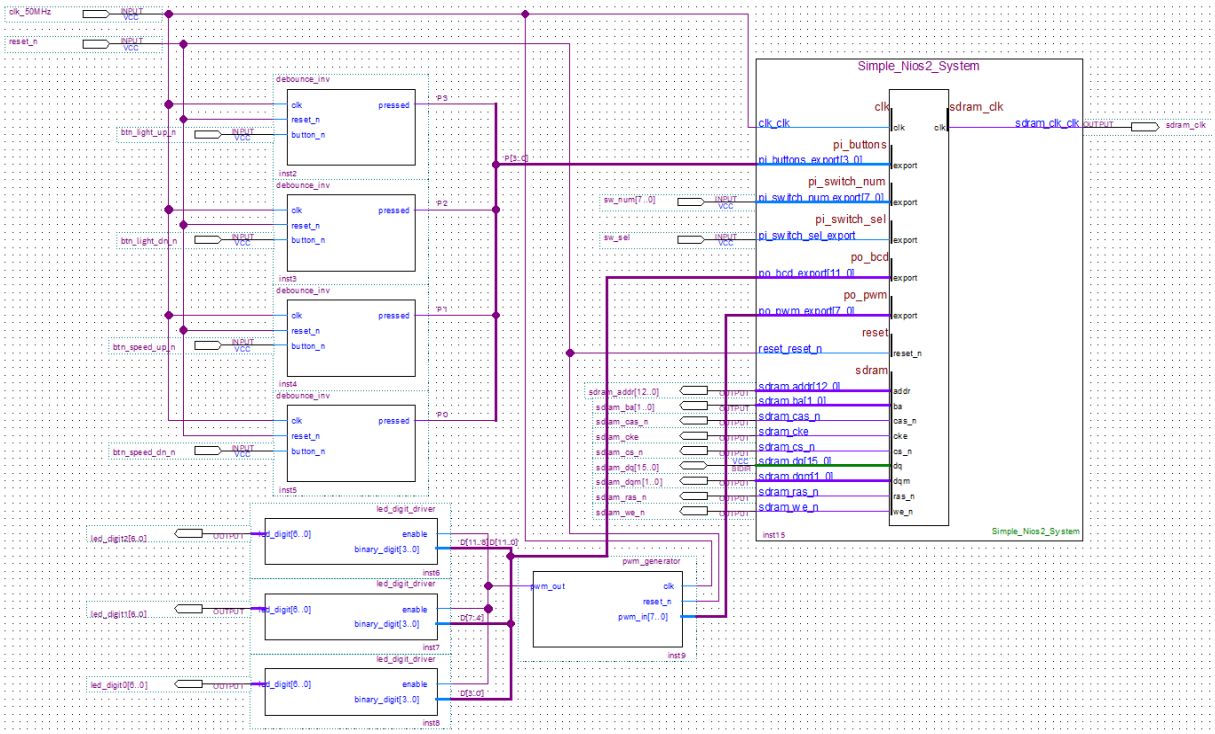
Desnim klikom na komponentu ***Simple_Nios2_System*** i odabirom opcije ***Generate Pins for Symbol Ports*** generišu se odgovarajući pinovi za sve portove koji do sada nisu povezani. Povezati ***clk*** i ***reset*** na ostale komponente koje imaju odgovarajuće ulaze i iskoristiti opciju za generisanje pinova na njima kada su svi ostali signali povezani. Kako se izlazni pinovi pll-a ne koriste potrebno ih je obrisati.

U ovom primeru su promenjena imena nekoliko signala prikazanih u tabeli 1. pošto su automatski generisana imena bila glomazna.

Tabela 1. Promena automatski generisanih imena

automatski generisano ime	novo ime
clk_clk	clk_50MHz
reset_reset_n	reset_n
button_n	btn_light_up_n
button_n12	btn_light_dn_n
button_n13	btn_speed_up_n
button_n14	btn_speed_dn_n
led_digit11	led_digit2
led_digit10	led_digit1
led_digit	led_digit0
pi_switch_num_export	sw_num
pi_switch_sel_export	sw_sel
sdram_clk_clk	sdram_clk

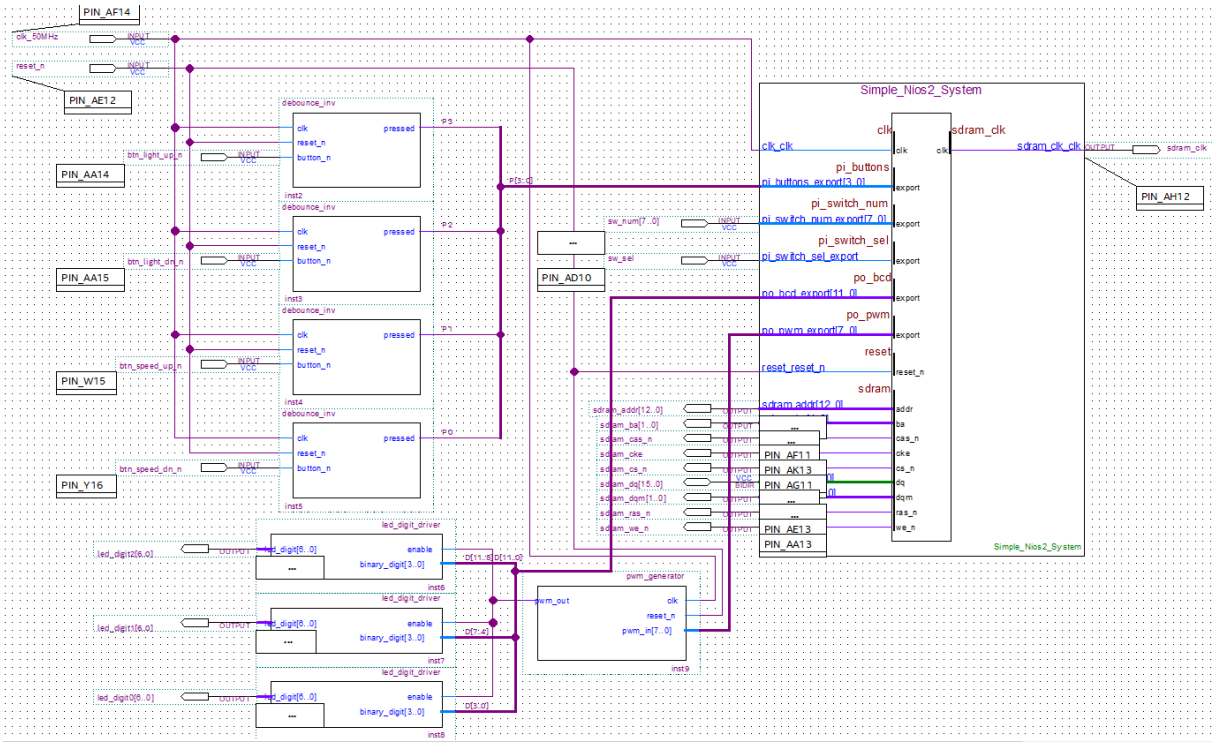
Finalni izgled sistema je prikazan na Slici 23.



Slika 23. Finalni izgled sistema

Na kraju je potrebno povezati pinove koji se nalaze u interfejsu našeg finalnog sistema sa pinovima na FPGA čipu. Ovo se najlakše može uraditi pomoću TCL skripte. U okviru fajlova koje ste

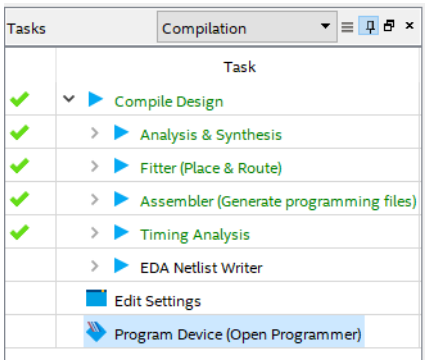
iskopirali nalazi se pin_assignment_DVS_DE1_SoC.tcl skripta koja je namenjena za ovaj primer. Možete je otvoriti pomoću bilo kog tekstualnog editora. Potrebno je proveriti da li se imena signala u skripti slažu sa imenima signala u finalnom sistemu ako to nije slučaj potrebno ih je uskladiti. Kada su nazivi signala u skripti i interfejsu sistema usklađeni može se pristupiti povezivanju pinova. Skripta se pokreće tako što u glavnom Quartus II prozoru pokrenete komandu ***Tools → Tcl Scripts…*** odaberete **pin_assign.tcl** kao na Slici 24. i pokrenete je komandom ***Run***. Pored svakog porta pojaviće se spisak pinova na koje je povezan kao na Slici 25.



Slika 25. Finalni sistem sa dodeljenim pinovima

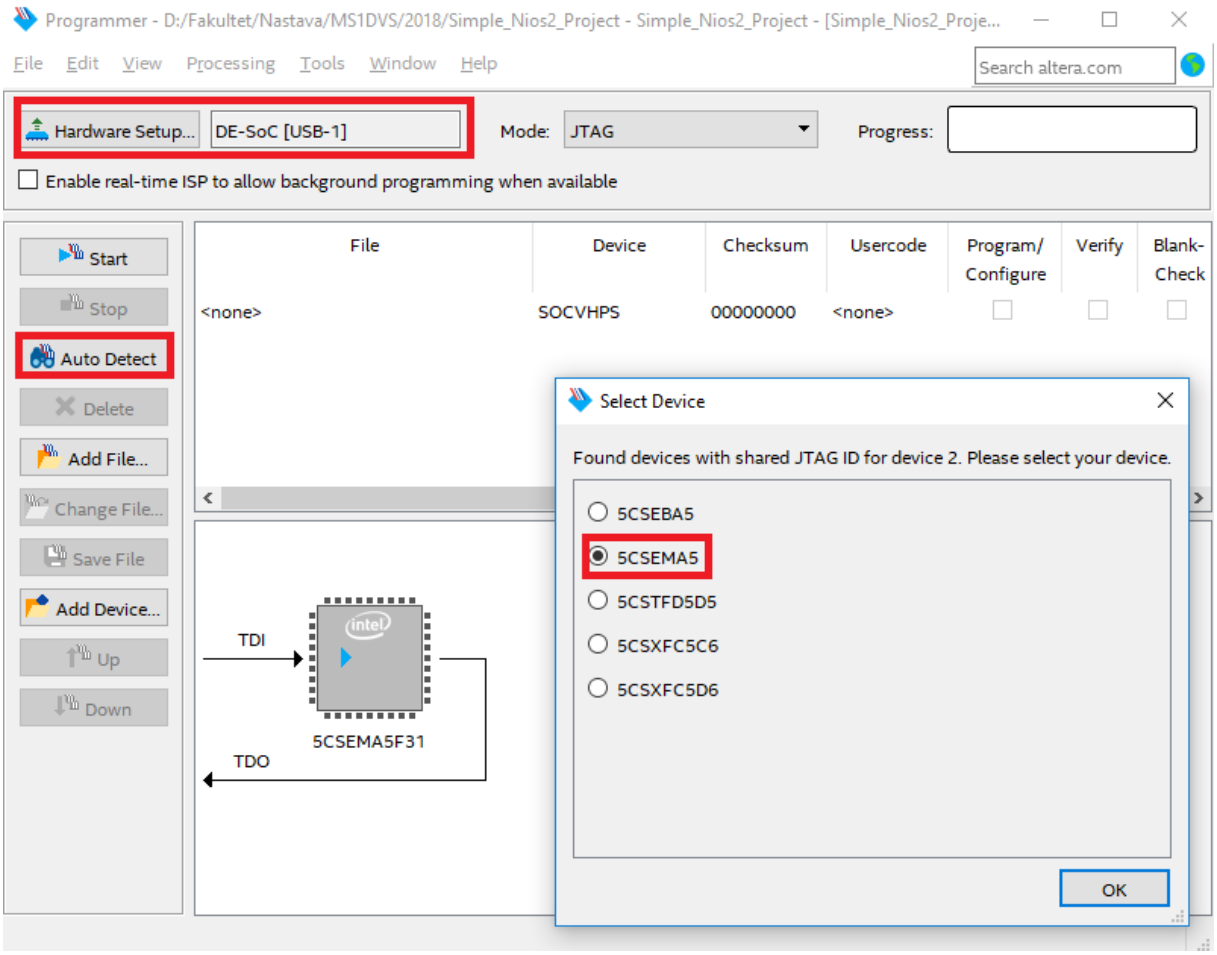
Kompajliranje sistema se pokreće komandom ***Processing → Start Compilation*** i posle nekog vremena i uz malo sreće se završava bez grešaka. Osnovni proizvod kompajliranja je ***.sof*** fajl koji služi za programiranje FPGA čipa i nalazi se u direktorijumu ***output_files***.

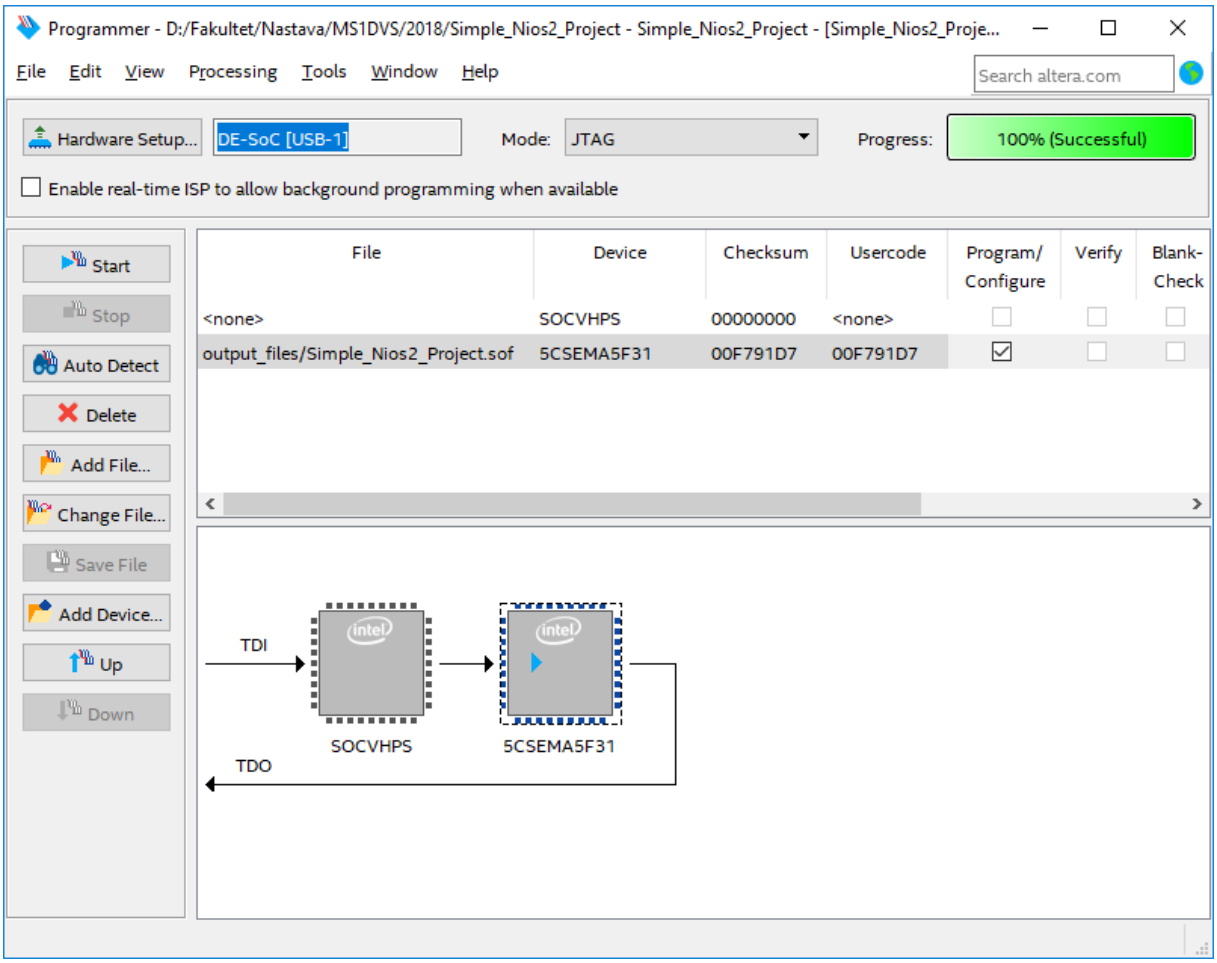
Na kraju je potrebno pokrenuti alat za programiranje FPGA čipa klikom na prečicu ***Program Device (Open Programmer)*** u odeljku za kompajliranje prikazanom na Slici 26.



Slika 26. Otvaranje alata za programiranje FPGA čipa

Otvara se prozor programatora prikazan na Slici 27. Potrebno je podesiti port na koji je povezan uređaj koji se programira klikom na dugme ***Hardware Setup***. U slučaju da u listi za izbor uređaja nije dostupna nijedna opcija a ploča je povezana sa računarom proverite da li su drajveri za USB blaster dobro instalirani. Drajveri se nalaze na sledećoj lokaciji: ***C:\intelFPGA_lite\18.0\quartus\drivers\usb-blaster***. Nakon toga je potrebno podesiti JTAG lanac klikom na ***Auto Detect*** taster i odabirom čipa 5SCEMA5 kao na Slici 27. Nakon toga se pojavljuju dve komponente u JTAG lancu jedna odgovara HPS (hard processor system) delu sistema koji se ne koristi u ovoj vežbi i drugi deo 5CSEMA5 koji odgovara FPGA delu sistema. Desnim klikom na ovu drugu komponentu i odabirom opcije ***Change file*** može se postaviti odgovarajući ***.sof*** fajl koji je upravo izgenerisan. Ako su port i trenutno aktivan fajl podešeni može se započeti programiranje čipa. Važno je da opcija ***Program/Configure*** bude izabrana. Programiranje se započinje klikom na taster ***Start***. U gornjem desnom uglu se prikazuje progres procesa programiranja i kada dođe do 100% FPGA čip je u potpunosti isprogramiran. Na LED displeju DE1-SoC ploče pojaviće se tri nule.





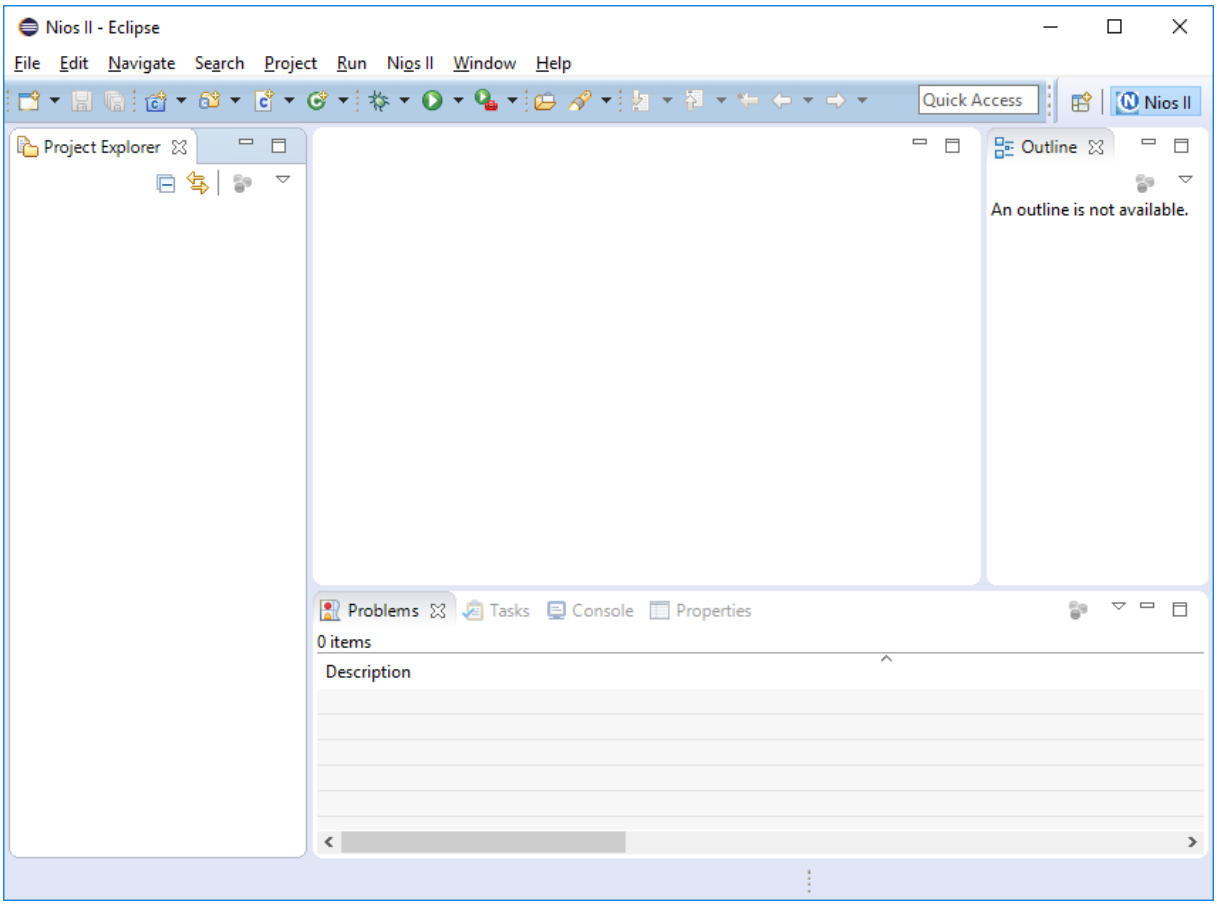
Slika 27. Izgled prozora programatora

Programiranje sistema

Za sada je projektovan samo hardverski deo sistema i funkcionalnosti koje su planirane za softver još uvek ne postoje. Tako da pritisak na tastere ili menjanje položaja prekidača ne utiče na prikaz LED displeja. Kako bi se ostvarile sve zamišljene funkcionalnosti potrebno je isprogramirati Nios II procesor. Nios II predstavlja namenski procesor slično kao bilo koji drugi mikrokontrolerski modul koji ste upoznali tokom osnovnih studija. Može se programirati u asembleru ili nekom višem programskom jeziku C/C++. Alat koji se koristi za projektovanje softverskog dela sistema naziva se Nios II SBT (*software build tools*) i može se koristiti iz grafičkog okruženja (*Eclipse*) ili iz komandne linije. Više detalja o Nios II procesoru kao i o alatima za programiranje može se naći na https://www.intel.com/content/www/us/en/products/programmable/processor/nios-ii.html. U ovom uputstvu biće prikazan postupak projektovanja softvera na primeru jednostavnog sistema koji je realizovan u prethodnom odeljku.

Na početku je potrebno pokre nuti *Nios II Software Build Tools for Eclipse*. Otvara se prozor za podešavanje trenutno aktivnog radnog direktorijuma. Podrazumevane putanje će biti kreirane na osnovu trenutno aktivnog radnog direktorijuma tako da je najbolje postaviti direktorijum projekta za koji se projektuje softver da bude trenutno aktivan. Nakon toga se otvara prozor prikazan na Slici 28. Potrebno je obratiti pažnju na gornji desni ugao i ikonicu *Nios II* koja označava da su unutar *Eclipse* okruženja aktivirani dodaci koji omogućavaju razvoj softvera za *Nios II* procesorski sistem. Sa leve strane se nalazi odeljak *Project Explorer* u kom se prikazuju trenutno aktivni projekti sa celom direktorijumskom strukturom. Kako još uvek nije kreiran nijedan projekat taj odeljak je prazan. U

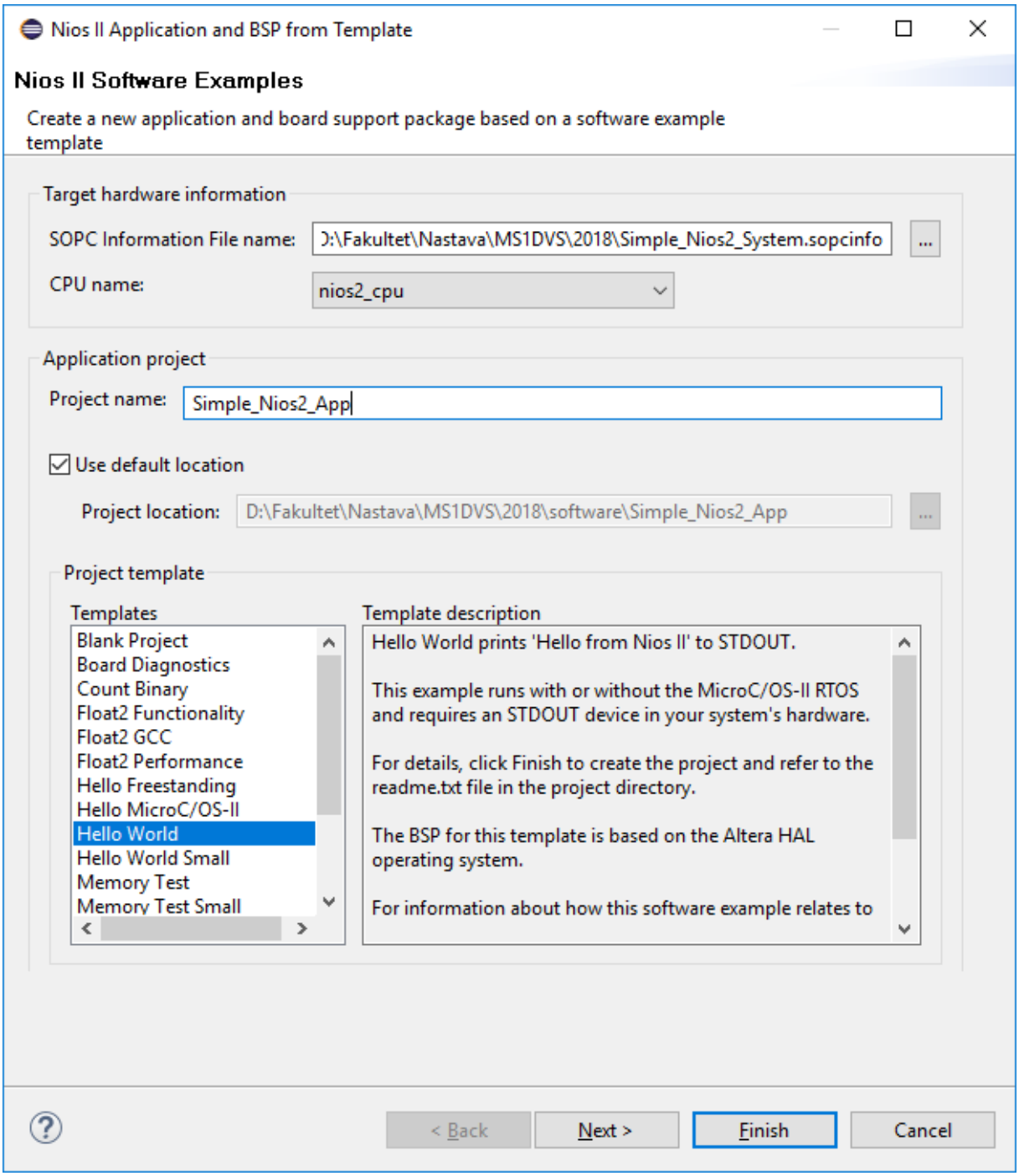
donjem delu se nalazi više odeljaka u kojima se izlistavaju poruke o greškama, upozorenjima, izveštaji o toku procesa kompajliranja i sl.



Slika 28. Nios II SBT radno okruženje

Najlakše je započeti projektovanje softvera kreiranjem odgovarajućeg templejta koji se posle modifikuje dodavanjem korisničkog koda. Izborom opcije ***File → New → Nios II Application and BSP from Template*** otvara se prozor za kreiranje softverskog projekta. Svaki softverski projekat se sastoji iz najmanje dve celine: korisnička aplikacija i BSP (*board supprot package*). BSP predstavlja skup funkcija specifičnih za hardversku platformu na kojoj će se aplikacija izvršavati. Korisnička aplikacija poziva funkcije iz BSP-a kako bi iskoristila funkcionalnosti hardverske platforme. Idealna dekompozicija softvera bi podrazumevala da sve funkcije koje pristupaju hardveru na najnižem nivou (registrima) budu smeštene u BSP kako bi se omogućila maksimalna portabilnost (hardverska nezavisnost korisničke aplikacije). Ovo nažalost često nije moguće ali i ne predstavlja osnovni prioritet pri projektovanju jako optimizovanih namenskih sistema. Za kreiranje BSP-a potrebne su informacije o hardverskom sistemu za koji se projektuje softver. Te informacije uključuju broj procesora u sistemu, spisak periferija u sistemu dostupne registre, memorijsku mapu i sl. Pri generisanju hardverskog dela sistema u okviru *Qsys* alata jedan od proizvoda je ***.sopcinfo*** fajl. Ovaj fajl sadrži sve potrebne informacije o procesorskom sistemu za koji se projektuje softver. Za potrebe ovog primera je potrebno odabrati ***Simple_Nios2_System.sopcinfo*** koji je kreiran ranije. Kako svaki procesor zahteva poseban softverski projekat u slučaju da se radi o multiprocesorskom sistemu potrebno je naznačiti za koji procesor se kreira softverski projekat. Potom je potrebno odrediti ime softverskog projekta, za potrebe ovog primera staviti naziv ***Simple_Nios2_App***. Podrazumevana putanja za aplikaciju je ***software*** direktorijum u okviru trenutno aktivnog direktorijuma. To se naravno može promeniti eksplicitnom specifikacijom željene putanje. Na raspolaganju je više različitih primera projekata. Nama je

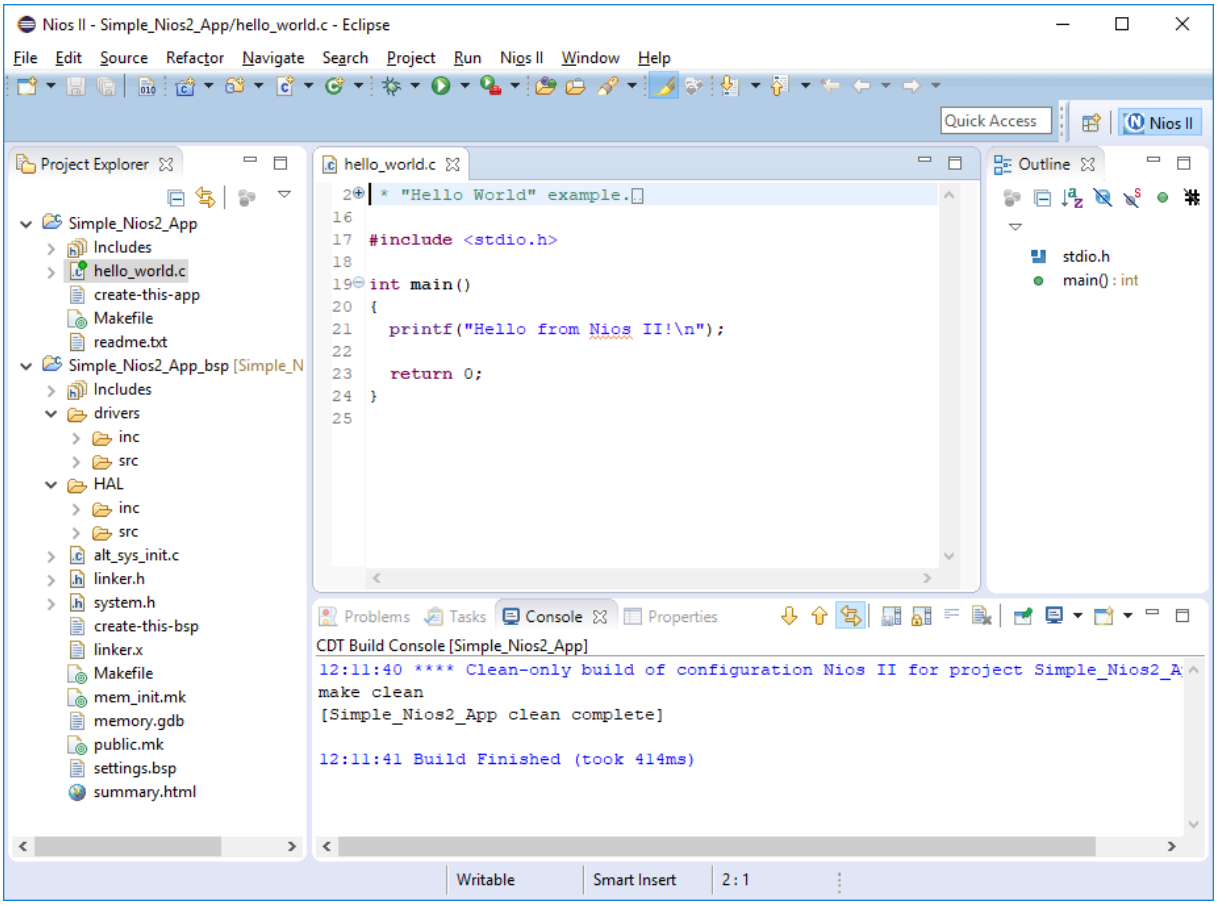
najinteresantniji ***Hello World*** koji kreira osnovni BSP i omogućava komunikaciju između procesora i host računara serijskim putem. Postoji i varijanta ovog primera koja implementira ***MicroC/OS-II RTOS*** ali je u ovom primeru nećemo koristiti. Po završenom podešavanju prozor za kreiranje primera bi trebalo da izgleda kao na Slici 29.



Slika 29. Kreiranje primera softverskog projekta

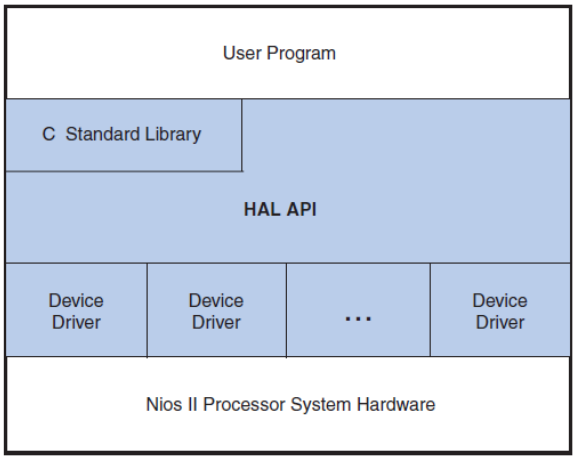
Klikom na taster ***Next*** može se podesiti naziv i putanja do BSP biblioteke. Ovde nije potrebno ništa menjati. Po kreiranju projekta u odeljku ***Project Explorer***-a pojavljuju se dva direktorijuma. Direktorijum ***Simple_Nios2_App*** sadrži fajlove korisničke aplikacije koji su u idealnom slučaju nezavisni od platforme dok direktorijum ***Simple_Nios2_App_bsp*** sadrži biblioteke koje su specifične za hardverski deo sistema (što je i naznačeno u zagradi uz ime direktorijuma). BSP sadrži dosta automatski

kreiranih fajlova što se može videti na Slici 30. Najvažniji fajl je ***system.h*** koji sadrži opise ključnih komponenti sistema, bazne adrese svih perfierija i njihove osnovne karakteristike. U okviru BSP direktorijuma mogu se primetiti dva direktorijuma ***HAL*** i ***drivers***. ***HAL*** (hardware abstraction layer) predstavlja skup opštih funkcija za pristup hardverskim komponentama i upravljanjem delovima sistema. U okviru direktorijuma ***drivers*** nalaze se funkcije koje apstrahuju ponašanje hardverskih komponenti sistema sakrivajući direktan pristup registirma od korisničke aplikacije na taj način omogućavajući da aplikacija bude što više nezavisna od hardvera koji se nalazi u sistemu. Slojevita struktura softverskog projekta prikazana je na Slici 31.



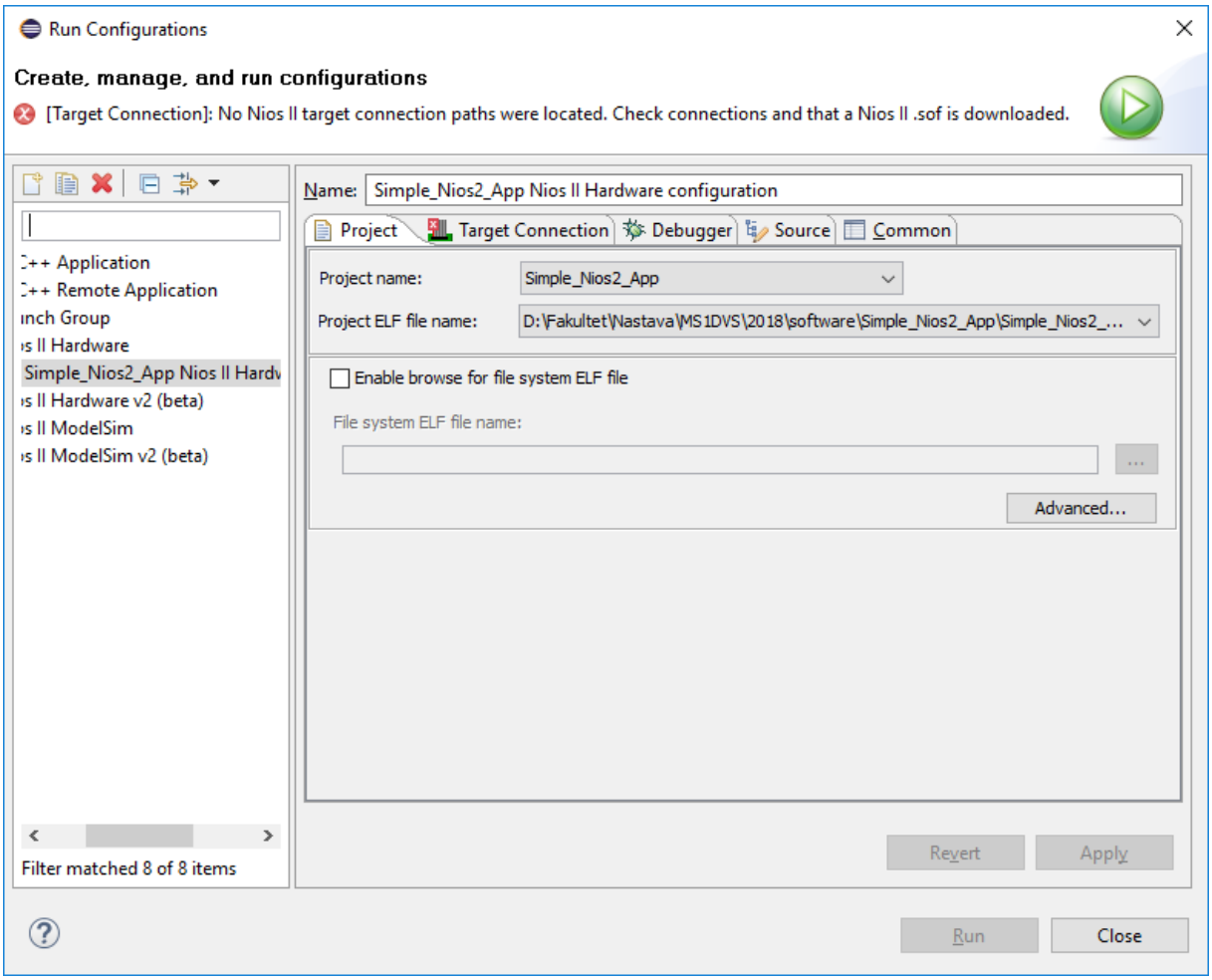
Slika 30. Direktorijumska struktura softverskog projekta

U okviru korisničke aplikacije se nalazi samo jedan ***.c*** fajl koji implementira funkcionalnost primera. Kao što se vidi sa Slike 30. kod je veoma jednostavan i koristi funkcije iz standardne C biblioteke za slanje karaktera serijskim putem host računaru. Hardver koji omogućava ovu funkcionalnost je JTAG UART koji predstavlja deo procesorskog sistema realizovanog ranije.



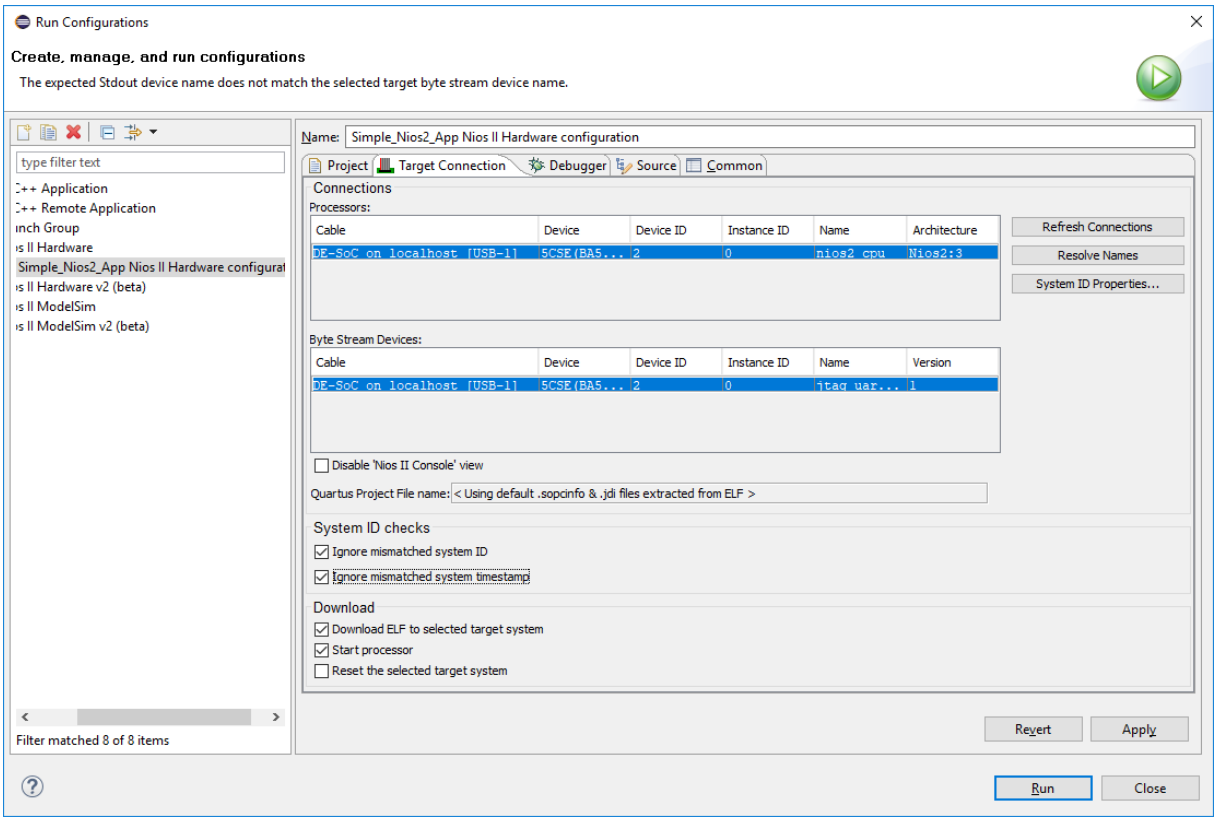
Slika 31. Slojevita struktura softverskog projekta

Kreirani primer je kompletan i spreman za kompajliranje. Kreiranje izvršnog fajla se započinje komandom ***Project → Build All***. Proces kreiranja izvršnog fajla obuhvata kompajliranje i povezivanje BSP-a i fajlova korisničke aplikacije u jedinstveni fajl sa ekstenzijom ***.elf*** (*executable and linking format file*). Pre nego što se ovaj fajl spusti u programsku memoriju potrebno je proveriti da li je FPGA čip isprogramiran tako da implementira odgovarajući procesorski sistem. Ako jeste može se započeti postupak spuštanja procesorskog programa. Desnim klikom na odgovarajući ***elf*** fajl otvara se padajući meni u okviru koga je potrebno odabrati opciju ***Run As → Nios II Hardware***. U nekim slučajevima može doći do ne prepoznavanja konekcije sa hardverom i pri tom se pojavljuje prozor za konfigurisanje prikazan na Slici 32.



Slika 32. Problem pri spuštanju izvršnog fajla na hardversku platformu

Aktiviranjem odeljka ***Target Connection*** i osvežavanjem konekcije klikom na taster ***Refresh Connections*** pojavljuju se aktivni procesor u delu ***Processors*** i ***Byte Stream Devices***. Nekada problem predstavljaju i neusklađeni ID i timestamp procesora tako da bi trebalo isključiti te provere kao što je prikazano na Slici 33. Po otklanjanju ovih problema proces spuštanja izvršnog fajla se nastavlja klikom na taster ***Run***. Izvršni fajl se smešta u memoriju određenu za instrukcije i započinje se izvršavanje instrukcije na adresi definisanoj ***reset*** vektorom pri kreiranju procesorskog sistema. U donjem delu prozora otvara se odeljak ***Nios II Console*** u kojem je ispisana poruka ***Hello from Nios II!***. Na ovaj način je uspostavljena osnovna komunikacija između procesorskog sistema i host računara. Sama Nios II konzola se može koristiti kao terminal za komunikaciju sa procesorom slanjem i prijemom teksutalnih poruka. Procesor može prihvatati ulaze i slati podatke korišćenjem funkcija iz standardne C biblioteke kao što su ***printf***, ***scanf***, ***putchar***, ***getchar*** i sl.



Slika 33. Osvežena konekcija sa procesorom i otklonjena greška

Korisničku aplikaciju je najlakše projektovati modifikacijom primera koji je upravo kreiran. Ako se pogleda struktura ***hello_world.c*** programa vidi se da on uključuje standardnu biblioteku za upis i ispis ***stdio.h*** i jednu funkciju ***main*** sa povratnim argumentom tipa ***int***. Vrlo je važno da svaka korisnička aplikacija sadrži funkciju ***main*** u okviru koje se izvršava. Kako bi se ostvarila veza sa hardverskim sistemom potrebno je poznavati strukturu sistema, načine pristupa i programiranja hardverskih modula, adrese registara i sl. Detaljan opis svake od periferija koje su date u okviru ***Quartus II*** paketa dat je u dokumentu ***Embedded Peripherals IP – User Guide***

https://www.intel.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_embedded_ip.pdf

Za početak cilj je da menjanjem položaja prekidača menjamo broj koji se prikazuje na LED displeju. Prekidači su povezani sa procesorskim sistemom pomoću komponente ***PIO Core*** i njen opis se može pronaći u pomenutom dokumentu. Set registara kojima se može pristupiti sa procesora prikazan je na Slici 34. Specificirani su ofseti adrese registara u odnosu na baznu adresu periferije kao i osnovna funkcija svakog od registara. Svim registrima se može pristupiti direktno računanjem apsolutne adrese registra i korišćenjem HAL funkcija za direktan pristup. Međutim ovaj način rada treba izbegavati iz razloga što stvara kod koji se jako teško održava i nije portabilan. Drugi način je korišćenje funkcija dostupnih u vidu drajvera za odgovarajuću periferiju. U okviru direktorijuma ***Simple_Nios2_App_bsp*** nalazi se direktorijum ***drivers*** u kome se nalaze svi dostupni drajveri periferija u sistemu. Interfejs funkcija i definicija makroa za upravljanje PIO periferijom dat je u fajlu ***altera_avalon_pio_regs.h*** iz ***inc*** direktorijuma. Kao ulazni argumenti ovih funkcija su bazna adresa konkretne periferije dok je ofset sakriven unutar implementacije. Korisnička aplikacija uvek koristi identične funkcije i u slučaju promene ofseta registara nije potrebno menjati korisničku aplikaciju što znatno olakšava razvoj. Definicije baznih adresa svih periferija i nekih osnovnih opcija date su u fajlu ***system.h***. Jednostavni program koji očitava vrednost sa prekidača i prenosi je na LED displej prikazan je na Slici 35.

Offset	Register Name		R/W	Fields				
				(n-1)	...	2	1	0
0	data	read access	R	Data value currently on PIO inputs.				
		write access	W	New value to drive on PIO outputs.				
1	direction *(1)*		R/W	Individual direction control for each I/O port. A value of 0 sets the direction to input; 1 sets the direction to output.				
2	interruptmask *(1)*		R/W	IRQ enable/disable for each input port. Setting a bit to 1 enables interrupts for the corresponding port.				
3	edgecapture *(1)*, *(2)*		R/W	Edge detection for each input port.				
4	outset		W	Specifies which bit of the output port to set.				
5	outclear		W	Specifies which output bit to clear.				

Slika 34. Registarska mapa PIO periferije

```
#include <stdio.h>
#include <altera_avalon_pio_regs.h>
#include <system.h>

int main()
{
  int a = 0, b = 0, sel = 0;

  printf("Hello from Nios II!\n");

  sel = IORD_ALTERA_AVALON_PIO_DATA(PI_SWITCH_SEL_BASE);

  if (sel)
  {
     a = IORD_ALTERA_AVALON_PIO_DATA(PI_SWITCH_NUM_BASE);
  }
  else
  {
     b = IORD_ALTERA_AVALON_PIO_DATA(PI_SWITCH_NUM_BASE);
  }

  IOWR_ALTERA_AVALON_PIO_DATA(PO_BCD_BASE, a);
  return 0;
}
```

Slika 35. Očitavanje prekidača i ispis na displej

Kompajliranjem projekta posle ovih izmena i spuštanjem koda na ploču može se ustanoviti da LED displej i dalje ne reaguje na promenu položaja prekidača. Razlog za to je što će se kod prikazan na Slici 35. jednom izvršiti i neće moći da reaguje na promene stanja prekidača. Ako promenimo stanje prekidača i resetujemo procesor pritiskom na odgovarajući taster na LED displeju će se prikazati vrednost koja odgovara trenutnom stanju prekidača međutim svaka naredna promena ne proizvodi nikakve reakcije. Zbog toga je pri pisanju programa za namenske sisteme uobičajeno da sav kod osim delova vezanih za inicijalizacije bude smešten unutar ***while (1)*** petlje što omogućava njegovo konstantno izvršavanje. Druga stvar koja se može primetiti je da LED displej zahteva podatak u BCD formatu tako da je za ispravan prikaz potrebno obaviti konverziju iz binarnog broja u BCD. Konverzija se može obaviti projektovanjem posebnog hardverskog modula ili jednostavnim korišćenjem resursa procesora kao što je to učinjeno u ovom primeru. Iako korišćeni procesor nema ugrađene delitelje i množače on ima mogućnost da obavi konverziju emuliranjem odgovarajućih instrukcija što naravno košta dodatne procesorske cikluse. Na Slici 36. je prikazan kod koji se izvršava u beskonačnoj petlji i koji implementira konverziju iz binarnog u bcd format. Dodato je i očitavanje drugog prekidača pri čemu se na LED displeju prikazuje zbir ova dva broja.

```c
#include <stdio.h>
#include <altera_avalon_pio_regs.h>
#include <system.h>

static int bin2bcd(int a)
{
  a = ((a/100)<<8) +
      (((a - (a/100)*100)/10)<<4) +
      (a - (a/10)*10);
  return a;
}

int main()
{
  int a = 0, b = 0, sel = 0, c = 0;

  printf("Hello from Nios II!\n");

  while (1)
  {
      sel = IORD_ALTERA_AVALON_PIO_DATA(PI_SWITCH_SEL_BASE);

      if (sel)
      {
         a = IORD_ALTERA_AVALON_PIO_DATA(PI_SWITCH_NUM_BASE);
      }
      else
      {
         b = IORD_ALTERA_AVALON_PIO_DATA(PI_SWITCH_NUM_BASE);
      }

      c = bin2bcd(a + b);

      IOWR_ALTERA_AVALON_PIO_DATA(PO_BCD_BASE, c);
  }

  return 0;
}
```

Slika 36. Očitavanje prekidača i ispis na displej u beskonačnoj petlji

Trenutno se na LED displeju ispisuje zbir 2 osmobitna broja zadatih prekidačima. Osvetljaj displeja je konstantan. U sistemu postoji PWM modul povezan na drajver displeja koji omogućava kontrolu osvetljenosti. Periodično povećanje i smanjenje osvetljenosti se može postići menjanjem faktora ispunjenosti PWM modula. Periferija tajmera se može iskoristiti da daje periodične događaje u okviru kojih će se menjati faktor ispunjenosti PWM modula a samim tim i osvetljaj displeja. Periferija tajmera se može konfigurisati da radi kao izvor periodičnih prekida fiksne periode, *watchdog* tajmer, sistemski takt ili *timestamp* tajmer. U konkretnom primeru potrebno je da tajmer generiše periodične prekide međutim želimo i mogućnost da se period tih prekida menja tokom izvršavanja programa. Za standardne primene postoje funkcije drajvera koje implementiraju osnovne funkcionalnosti. Međutim kako se tajmer u našem primeru ne koristi ni jedan od standardnih načina onda je potrebno pristupati direktno registrima tajmera prikazanim na Slici 37.

Offset	Name	R/W	Description of Bits						
			15	...	4	3	2	1	0
0	status	RW	*(1)*					RUN	TO
1	control	RW	*(1)*			STOP	START	CONT	ITO
2	periodl	RW	Timeout Period – 1 (bits [15:0])						
3	periodh	RW	Timeout Period – 1 (bits [31:16])						
4	snapl	RW	Counter Snapshot (bits [15:0])						
5	snaph	RW	Counter Snapshot (bits [31:16])						

Slika 37. Registri periferije tajmera

Kako bi periferija tajmera bila kompatibilna i sa 16-bitnim procesorima svi registri su veličine 16-bita. Statusni registar označava trenutno stanje tajmer modula. Bit **RUN** je 1 dok god je brojanje u toku u suprotnom je 0. Bit **TO** (*time out*) označava da je brojač završio jedan period brojanja i ostaje 1 sve dok se eksplicitno ne obriše sa master modula (procesora). Upisom odgovarajućih bita u kontrolni registar se upravlja radom brojača. Postavljanjem bita **ITO** na 1 omogućava se generisanje prekida kada procesor izbroji jedan period brojanja. Postavljenjem bita **CONT** na 1 aktivira se kontinualan mod brojanja u kome brojač kada dođe do 0 učitava vrednost periode i započinje novi ciklus brojanja, u suprotnom brojač se zaustavlja čim dođe do 0. Upisivanjem 1 na poziciju **START** bita brojač započinje sa brojanjem dok upisivanjem 1 na poziciju **STOP** bita brojač zaustavlja brojanje. Upisivanje 0 na pozicije **START** i **STOP** bita nema nikakvog efekta. U okviru fajla ***altera_avalon_timer_regs.h*** nalaze se rutine za pristup registrima kao i pozicije i maske za pristup određenim bitima unutar registara. Preporučuje se uvek korišćenje ovih simboličkih imena kako bi se kod učinio čitljivijim i portabilnijim. Primer koda kojim se inicijalizuje i startuje tajmer prikazan je na Slici 38. Period tajmera je podešen tako da se osvetljaj LED displeja promeni od najtamnijeg do najsvetlijeg (***pwm_max_count***) pa opet do najtamnijeg za vreme od 10 sekundi (inicijalna vrednost ***led_period_sec*** promenljive). Kako bi se odredila perioda tajmera koja ovo ispunjava potrebno je poznavati učestanost signala takta na kom radi modul tajmera. Ova informacija se nalazu u ***system.h*** fajlu pod imenom ***TIMER_FREQ***. Kako bi se izbrojao period od ***led_period_sec*** sekundi potrebno je da prođe ***led_period_sec* TIMER_FREQ*** perioda tajmera. Međutim za ovo vreme LED displej treba da promeni osvetljaj od minimalnog do maksimalnog i oped nazad do minimalnog. To znači da je potrebno ***(2*pwm_max_count + 1)*** promeniti faktor ispunjenosti PWM generatora pa je prekida potrebno generisati na svakih ***led_period_sec* TIMER_FREQ/(2*pwm_max_count + 1)*** taktova kao što je urađeno na Slici 38. Tajmer je podešen tako da generiše prekide i radi u kontinualnom modu.

```
17 #include <stdio.h>
18 #include <altera_avalon_pio_regs.h>
19 #include <altera_avalon_timer_regs.h>
20 #include <sys/alt_irq.h>
21 #include <system.h>
22
23 #define MAX_LED_PERIOD_SEC 10
24
25 #define MAX_PWM_COUNT 255
26
27 static int bin2bcd(int a)
28 {
29   a = ((a/100)<<8) +
30       (((a - (a/100)*100)/10)<<4) +
31       (a - (a/10)*10);
32   return a;
33 }
34
35 int main()
36 {
37   int a = 0, b = 0, sel = 0, c = 0;
38
39   unsigned int pwm_max_count = MAX_PWM_COUNT;
40   unsigned int led_period_sec = MAX_LED_PERIOD_SEC;
41   unsigned int timer_period = (led_period_sec*TIMER_FREQ)/(2*pwm_max_count + 1);
42
43   printf("Hello from Nios II!\n");
44
45   /* Set initial timer period */
46   IOWR_ALTERA_AVALON_TIMER_PERIODL(TIMER_BASE, timer_period);
47   IOWR_ALTERA_AVALON_TIMER_PERIODH(TIMER_BASE, timer_period >> 16);
48
49   /* Configure and start the timer */
50   IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_BASE, ALTERA_AVALON_TIMER_CONTROL_ITO_MSK +
51                                                ALTERA_AVALON_TIMER_CONTROL_CONT_MSK +
52                                                ALTERA_AVALON_TIMER_CONTROL_START_MSK);
53
```

```
53
54
55    while (1)
56    {
57        sel = IORD_ALTERA_AVALON_PIO_DATA(PI_SWITCH_SEL_BASE);
58
59        if (sel)
60        {
61            a = IORD_ALTERA_AVALON_PIO_DATA(PI_SWITCH_NUM_BASE);
62        }
63        else
64        {
65            b = IORD_ALTERA_AVALON_PIO_DATA(PI_SWITCH_NUM_BASE);
66        }
67
68        c = bin2bcd(a + b);
69
70        IOWR_ALTERA_AVALON_PIO_DATA(PO_BCD_BASE, c);
71    }
72
73    return 0;
74 }
```

Slika 38. Inicijalizacija tajmera

Kako modul tajmera generiše prekid potrebno je napisati prekidnu rutinu tajmer modula. U okviru Nios II sistema može postojati više prekidnih kontrolera. Najjednostavniji je interni prekidni kontroler IIC (*internal interrupt controller*) koji je ugrađen u Nios II procesor. Ovaj kontroler ne podržava vektorske prekide već po prijemu prekida kontrolu preusmerava na jedinstvenu adresu za obradu izuzetaka (*exception address*). Hardver ima informaciju koji se prekid desio i omogućava softveru da maskira odgovarajući prekid. HAL na osnovu informacije koji se prekid desio preusmerava kontrolu ka odgovarajućoj prekidnoj rutini. Kako bi se poboljšale performanse korišćenjem vektorskih prekida postoji mogućnost korišćenja eksternog kontrolera prekida koji se može dodati kao posebna komponenta unutar procesorskog sistema. Sam mehanizam prekida poznat je od ranije i isti je kao na većini namenskih sistema.

Najpre je potrebno registrovati prekidnu rutinu pozivom funkcije alt_ic_isr_register čiji je prototip prikazan na Slici 39. Argument ***ic_id*** označava prekidni kontroler i u slučaju korišćenja internog kontrolera prekida IIC nije bitan. Argument ***irq*** označava prekidnu liniju na koji je povezan uređaj i najbolje je koristiti simboličko ime iz ***system.h*** fajla. Argument ***isr*** je pokazivač na funkciju koja implementira odgovarajuću prekidnu rutinu. Prekidna rutina može biti bilo koja funkcija čiji je prototip ***void (void *isr_context)***. Argument ***isr_context*** predstavlja pokazivač na strukturu podataka koja se prosleđuje prekidnoj rutini.

```
int alt_ic_isr_register(alt_u32 ic_id,
                        alt_u32 irq,
                        alt_isr_func isr,
                        void *isr_context,
                        void* flags)
```

Slika 39. Prototip funkcije za registrovanje prekidne rutine

Prototip funkcije za registrovanje prekidne rutine nalazi se u fajlu ***sys/alt_irq.h*** pa ga je potrebno uključiti glavni program. Kako bi se prosleđivali parametri prekidnoj rutini definisana je strktura ***pwm_params_t*** koja sadrži polja sa trenutnom i maksimalnom ispunjenošću impulsa PWM modula kao i o smeru promene faktora ispunjenosti. Kako bi se dobio pregledniji kod za smer promene faktora ispunjenosti definisan je novi nabrojivi tip ***pwm_direction_t*** koji sadrži dve moguće vrednosti ***PWM_COUNT_UP*** i ***PWM_COUNT_DOWN***. U okviru glavnog programa definisana je promenljiva tipa ***pwm_params_t***. Obratiti pažnju da je za ovu promenljivu postavljeno svojstvo ***volatile*** koje označava da se mogu desiti promene promenljive van glavnog toka programa, u našem slučaju prekidna rutina

može menjati sadržaj ove promenljive. U okviru glavnog programa registruje se prekidna rutina tajmera korišćenjem funkcije sa Slike 39. na način prikazana na Slici 40. Takođe je obezbeđena promena osvetljaja displeja slanjem ***count*** parametra ***pwm*** modulu unutar beskonačne petlje.

Na Slici 40. je prikazana i prekidna rutina tajmera ***handle_timer_interrupts***. Kako je ulazni parametar pokazivač tipa ***void*** najpre je potrebno izvršiti konverziju u odgovarajući tip podataka kako bi se moglo pristupiti poljima strukture. Potvrda prekida se ostvaruje brisanjem ***TO*** bita u statusnom registru tajmera što je učinjeno kao prva funkcija prekidne rutine. Potom se u zavisnosti od smera menja trenutna vrednost polja ***count*** i eventualno menja smer u slučaju da je ***count*** dobio neku od svojih graničnih vrednosti. Pri kreiranju prekidnih rutina potrebno je voditi računa da se što manja količina posla radi u okviru same prekidne rutine kako bi se sprečilo ispuštanje drugih prekida u sistemu.

```
17 #include <stdio.h>
18 #include <altera_avalon_pio_regs.h>
19 #include <altera_avalon_timer_regs.h>
20 #include <sys/alt_irq.h>
21 #include <system.h>
22
23 #define MAX_LED_PERIOD_SEC 10
24
25 #define MAX_PWM_COUNT 255
26
27 /*Structure of PWM parameters for communication with timer ISR*/
28
29 typedef enum {PWM_COUNT_UP, PWM_COUNT_DOWN} pwm_direction_t;
30
31 typedef struct {
32   unsigned int count;
33   unsigned int max_count;
34   pwm_direction_t direction;
35 } pwm_params_t;
36
37 static int bin2bcd(int a)
38 {
39   a = ((a/100)<<8) +
40       (((a - (a/100)*100)/10)<<4) +
41       (a - (a/10)*10);
42   return a;
43 }
44
45 void handle_timer_interrupts(void *context)
46 {
47     pwm_params_t *pwm_params = (pwm_params_t*) context;
48
49     /* Acknowledge interrupt by clearing status register */
50     IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_BASE, 0x0);
51
52     pwm_params->count = (pwm_params->direction == PWM_COUNT_UP)? pwm_params->count+1 : pwm_params->count-1;
53     if ((pwm_params->count == 0) || (pwm_params->count >= pwm_params->max_count))
54     {
55         if (pwm_params->count > pwm_params->max_count) pwm_params->count = pwm_params->max_count;
56         if (pwm_params->count < 0) pwm_params->count = 0;
57
58         pwm_params->direction = (pwm_params->direction == PWM_COUNT_UP) ? PWM_COUNT_DOWN : PWM_COUNT_UP;
59     }
60 }
```

```c
int main()
{
  int a = 0, b = 0, sel = 0, c = 0;

  volatile pwm_params_t pwm_params = {0, MAX_PWM_COUNT, PWM_COUNT_UP};
  unsigned int led_period_sec = MAX_LED_PERIOD_SEC;
  unsigned int timer_period = (led_period_sec*TIMER_FREQ)/(2*pwm_params.max_count + 1);

  printf("Hello from Nios II!\n");

  /* Set initial timer period */
  IOWR_ALTERA_AVALON_TIMER_PERIODL(TIMER_BASE, timer_period);
  IOWR_ALTERA_AVALON_TIMER_PERIODH(TIMER_BASE, timer_period >> 16);

  /* Configure and start the timer */
  IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_BASE, ALTERA_AVALON_TIMER_CONTROL_ITO_MSK +
                                               ALTERA_AVALON_TIMER_CONTROL_CONT_MSK +
                                               ALTERA_AVALON_TIMER_CONTROL_START_MSK);

  /* Register timer ISR */
  alt_ic_isr_register(TIMER_IRQ_INTERRUPT_CONTROLLER_ID,
                      TIMER_IRQ,
                      handle_timer_interrupts,
                      (void*)&pwm_params,
                      0x0);

  while (1)
  {
      IOWR_ALTERA_AVALON_PIO_DATA(PO_PWM_BASE, pwm_params.count);

      sel = IORD_ALTERA_AVALON_PIO_DATA(PI_SWITCH_SEL_BASE);

      if (sel)
      {
         a = IORD_ALTERA_AVALON_PIO_DATA(PI_SWITCH_NUM_BASE);
      }
      else
      {
         b = IORD_ALTERA_AVALON_PIO_DATA(PI_SWITCH_NUM_BASE);
      }

      c = bin2bcd(a + b);

      IOWR_ALTERA_AVALON_PIO_DATA(PO_BCD_BASE, c);
  }

  return 0;
}
```

Slika 40. Implementacija i registracija prekidne rutine tajmera

Po implementaciji prekidne rutine tajmera obezbeđena je periodična promena osvetljaja LED displeja od 0 do 255 sa periodom od 10 sekundi. U ovom trenutku međutim nije moguće promeniti niti maksimalni osvetljaj displeja niti periodu promene osvetljaja. Predviđeno je da se ovi parametri menjaju pomoću tastera dostupnih na razvojnoj platformi. Signali sa tastera su dovedeni u procesorski sistem pomoću PIO periferije Nios2 sistema. Ovo je identična periferija kao ona koja je korišćena kod prekidača. Razlika je jedino u tom što se signal sa prekidača očitavao sa aktivnim nivoom dok je kod tastera važno uhvatiti uzlaznu ivicu signala sa tastera. Omogućeno je i generisanje prekida procesoru kako bi sistem pravovremeno odreagovao na akciju pritiska tastera.

Posmatranjem registarske mape PIO periferije sa Slike 34. može se uočiti da postoje 2 registra koje ćemo koristiti u ovoj konfiguraciji PIO periferije a to su ***interrupt mask*** i ***edge capture*** registri. Kako nas ne interesuje trenutni nivo ulaznog signala već samo informacija da li je uhvaćena odgovarajuća ivica to se ulazni signal ne očitava iz ***data*** registra koji sadrži trenutni nivo već iz ***edge capture*** registra koji sadrži trenutno uhvaćene ivice. PIO periferija za tastere je podešena tako da generiše prekid pri svakom setovanju nekog bita u ***edge capture*** registru pod uslovom da sam prekid nije maskiran upisom nule na odgovarajuće mesto ***interrupt mask*** registra. Kao što se može videti iz dokumentacije biti ***edge capture*** registra se setuju prilikom detekcije odgovarajuće ivice ulaznog signala. Biti unutar ***edge capture*** registra ostaju setovani dok god se eksplicitno ne obrišu. Ukoliko je prilikom kreiranja periferije odabrana opcija ***Enable bit-clearing for edge capture register*** upisom

jedinice na odgovarajući bit ovog registra vrši se brisanje tog bita. Ukoliko ova opcija nije odabrana bilo koji upis briše kompletan ***edge capture*** registar.

PIO periferija sadrži jednu prekidnu liniju koja se može povezati sa bilo kojim master modulom u sistemu koji prihvata prekide od drugih modula. Prekid se može konfigurisati da bude aktivan na nivo ili ivicu signala i generiše se ako je ispunjen uslov za prekid na bilo kom bitu PIO periferije. Koji tačno bit je generisao prekid može se saznati očitavanjem odgovarajućeg registra ***data*** (ako je aktivan na nivo) ili ***edge capture*** (ako je aktivan na ivicu) unutar prekidne rutine. Prekid ostaje aktiviran dok god se ne potvrdi privremenim maskiranjem (upisom nule u ***interrupt mask*** registar) u slučaju prekida aktivnog na nivo ili brisanjem bita iz ***edge capture*** registra u slučaju prekida aktivnog na ivicu.

Dalje je potrebno osmisliti arhitekturu koda kojim se dodaje funkcionalnost tastera. Svaki put kada se detektuje uzlazna ivica sa nekog od tastera generisaće se prekid. Unutar prekidne rutine je obavezno očitati ***edge capture*** registar i resetovati odgovarajuće bite unutar njega kako bi se izvršila potvrda prekida. U ovom primeru je odabrano da se u prekidnoj rutini tastera obavlja samo ovaj minimalni skup operacija dok se sve ostale akcije obavljaju unutar glavnog programa na osnovu vrednosti očitane iz prekidne rutine. Ovo je u skladu sa praksom da se unutar prekidne rutine obavlja što manji deo posla kako bi se sprečilo zadržavanje procesora u stanju prekida i mogućnost propuštanja novih prekida u sistemu. Definisana je nova promenljiva ***button_action*** koja će se prosleđivati prekidnoj rutini i zbog toga joj je dodeljen deskriptor ***volatile***.

Unutar glavnog programa unutar beskonačne petlje ispituje se vrednost ***button_action*** promenljive i u slučaju da je ona različita od nule izvršavaju se odgovarajuće operacije. Najpre se ispituje svaki bit ove promenljive kako bi se odredilo koju je akciju potrebno izvršiti i na osnovu toga se preračunavaju vrednosti periode promene osvetljaja LED displeja ***led_period_sec*** i maksimalnog faktora ispunjenosti pwm modula (maskimalna jačina osvetljaja displeja) ***pwm_params.max_count***. Na osnovu novih vrednosti za perioda i maksimalnu jačinu osvetljaja preračunava se nova perioda tajmera za generisanje prekida i ova nova perioda se upisuje u registre tajmera za definisanje perioda rada. U slučaju da je omogućeno programsko pokretanje i zaustavljanje tajmera prilikom upisa u registar perioda tajmer se automatski zaustavlja. Kako bi se tajmer ponovo pokrenuo potrebno je to eksplicitno obaviti upisom jedinice na ***START*** bit u kontrolnom registru. Voditi računa da i ostali kontrolni biti budu ispravno podešeni! Na kraju je potrebno resetovati vrednost ***button_action*** registra kako bi se omogućilo prihvatanje novih komandi sa tastera. Program koji implementira kompletnu funkcionalnost prikazan je na Slici 41.

```c
#include <stdio.h>
#include <altera_avalon_pio_regs.h>
#include <altera_avalon_timer_regs.h>
#include <sys/alt_irq.h>
#include <system.h>

#define MAX_LED_PERIOD_SEC 10

#define BTN_INC_LED_PERIOD_MSK 0x1
#define BTN_DEC_LED_PERIOD_MSK 0x2
#define BTN_INC_PWM_MAX_COUNT_MSK 0x4
#define BTN_DEC_PWM_MAX_COUNT_MSK 0x8
#define BTN_ALL_MSK (BTN_INC_LED_PERIOD_MSK + \
                     BTN_DEC_LED_PERIOD_MSK + \
                     BTN_INC_PWM_MAX_COUNT_MSK + \
                     BTN_DEC_PWM_MAX_COUNT_MSK)

#define COUNT_STEP 5
#define MAX_PWM_COUNT 255

/*Structure of PWM parameters for communication with timer ISR*/

typedef enum {PWM_COUNT_UP, PWM_COUNT_DOWN} pwm_direction_t;

typedef struct {
  unsigned int count;
  unsigned int max_count;
  pwm_direction_t direction;
} pwm_params_t;

static int bin2bcd(int a)
{
  a = ((a/100)<<8) +
      (((a - (a/100)*100)/10)<<4) +
      (a - (a/10)*10);
  return a;
}

void handle_timer_interrupts(void *context)
{
    pwm_params_t *pwm_params = (pwm_params_t*) context;

    /* Acknowledge interrupt by clearing status register */
    IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_BASE, 0x0);

    pwm_params->count = (pwm_params->direction == PWM_COUNT_UP)? pwm_params->count+1 : pwm_params->count-1;
    if ((pwm_params->count == 0) || (pwm_params->count >= pwm_params->max_count))
    {
        if (pwm_params->count > pwm_params->max_count) pwm_params->count = pwm_params->max_count;
        if (pwm_params->count < 0) pwm_params->count = 0;

        pwm_params->direction = (pwm_params->direction == PWM_COUNT_UP) ? PWM_COUNT_DOWN : PWM_COUNT_UP;
    }
}

void handle_button_interrupts(void *context)
{
    unsigned int *button_action = (unsigned int*) context;

    *button_action =  IORD_ALTERA_AVALON_PIO_EDGE_CAP(PI_BUTTONS_BASE);

    /* Acknowledge interrupt by clearing edge capture register */
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(PI_BUTTONS_BASE, *button_action);
}

int main()
{
  int a = 0, b = 0, sel = 0, c = 0;

  volatile pwm_params_t pwm_params = {0, MAX_PWM_COUNT, PWM_COUNT_UP};
  unsigned int led_period_sec = MAX_LED_PERIOD_SEC;
  unsigned int timer_period = (led_period_sec*TIMER_FREQ)/(2*pwm_params.max_count + 1);
  volatile unsigned int button_action = 0;

  printf("Hello from Nios II!\n");

  /* Set initial timer period */
  IOWR_ALTERA_AVALON_TIMER_PERIODL(TIMER_BASE, timer_period);
  IOWR_ALTERA_AVALON_TIMER_PERIODH(TIMER_BASE, timer_period >> 16);

  /* Configure and start the timer */
  IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_BASE, ALTERA_AVALON_TIMER_CONTROL_ITO_MSK +
                                               ALTERA_AVALON_TIMER_CONTROL_CONT_MSK +
                                               ALTERA_AVALON_TIMER_CONTROL_START_MSK);

  /* Register timer ISR */
  alt_ic_isr_register(TIMER_IRQ_INTERRUPT_CONTROLLER_ID,
                      TIMER_IRQ,
                      handle_timer_interrupts,
                      (void*)&pwm_params,
                      0x0);

  /* Reset button edge capture register for all buttons */
  IOWR_ALTERA_AVALON_PIO_EDGE_CAP(PI_BUTTONS_BASE, BTN_ALL_MSK);

  /* Enable interrupts for all buttons */
  IOWR_ALTERA_AVALON_PIO_IRQ_MASK(PI_BUTTONS_BASE, BTN_ALL_MSK);

  /* Register buttons ISR */
  alt_ic_isr_register(PI_BUTTONS_IRQ_INTERRUPT_CONTROLLER_ID,
                      PI_BUTTONS_IRQ,
                      handle_button_interrupts,
                      (void*)&button_action,
                      0x0);
```

```
121
122     while (1)
123     {
124         IOWR_ALTERA_AVALON_PIO_DATA(PO_PWM_BASE, pwm_params.count);
125
126         sel = IORD_ALTERA_AVALON_PIO_DATA(PI_SWITCH_SEL_BASE);
127
128         if (sel == 1)
129         {
130             a = IORD_ALTERA_AVALON_PIO_DATA(PI_SWITCH_NUM_BASE);
131         }
132         else
133         {
134             b = IORD_ALTERA_AVALON_PIO_DATA(PI_SWITCH_NUM_BASE);
135         }
136
137         c = bin2bcd(a + b);
138
139         IOWR_ALTERA_AVALON_PIO_DATA(PO_BCD_BASE, c);
140
141         if (button_action)
142         {
143             if (button_action & BTN_INC_LED_PERIOD_MSK)
144             {
145                 led_period_sec = (led_period_sec == MAX_LED_PERIOD_SEC) ? MAX_LED_PERIOD_SEC : led_period_sec + 1;
146                 printf("LED period changed to: %d sec\n", led_period_sec);
147             }
148
149             if (button_action & BTN_DEC_LED_PERIOD_MSK)
150             {
151                 led_period_sec = (led_period_sec == 0) ? 0 : led_period_sec - 1;
152                 printf("LED period changed to: %d sec\n", led_period_sec);
153             }
154
155             if (button_action & BTN_INC_PWM_MAX_COUNT_MSK)
156             {
157                 pwm_params.max_count = (pwm_params.max_count > (MAX_PWM_COUNT - COUNT_STEP)) ? MAX_PWM_COUNT : pwm_params.max_count + COUNT_STEP;
158                 printf("PWM max count changed to: %d\n", pwm_params.max_count);
159             }
160
161             if (button_action & BTN_DEC_PWM_MAX_COUNT_MSK)
162             {
163                 pwm_params.max_count = (pwm_params.max_count < COUNT_STEP) ? 0 : pwm_params.max_count - COUNT_STEP;
164                 printf("PWM max count changed to: %d\n", pwm_params.max_count);
165             }
166
167             /* Re-calculate timer period */
168             timer_period = (led_period_sec*TIMER_FREQ)/(2*pwm_params.max_count + 1);
169
170             /* Update timer period */
171             IOWR_ALTERA_AVALON_TIMER_PERIODL(TIMER_BASE, timer_period);
172             IOWR_ALTERA_AVALON_TIMER_PERIODH(TIMER_BASE, timer_period >> 16);
173
174             /* Restart timer */
175             IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_BASE, ALTERA_AVALON_TIMER_CONTROL_ITO_MSK +
176                                                          ALTERA_AVALON_TIMER_CONTROL_CONT_MSK +
177                                                          ALTERA_AVALON_TIMER_CONTROL_START_MSK);
178
179             button_action = 0;
180         }
181     }
182
183     return 0;
184 }
```

Slika 41. Kompletan program za projekat iz primera