UNIVERZITET U BEOGRADU, ELEKTROTEHNIČKI FAKULTET, KATEDRA ZA ELEKTRONIKU

SIMULACIJA SISTEMA I TESTIRANJE NA PLOČI

DIGITALNI VLSI SISTEMI

V2.0

BEOGRAD, 2018

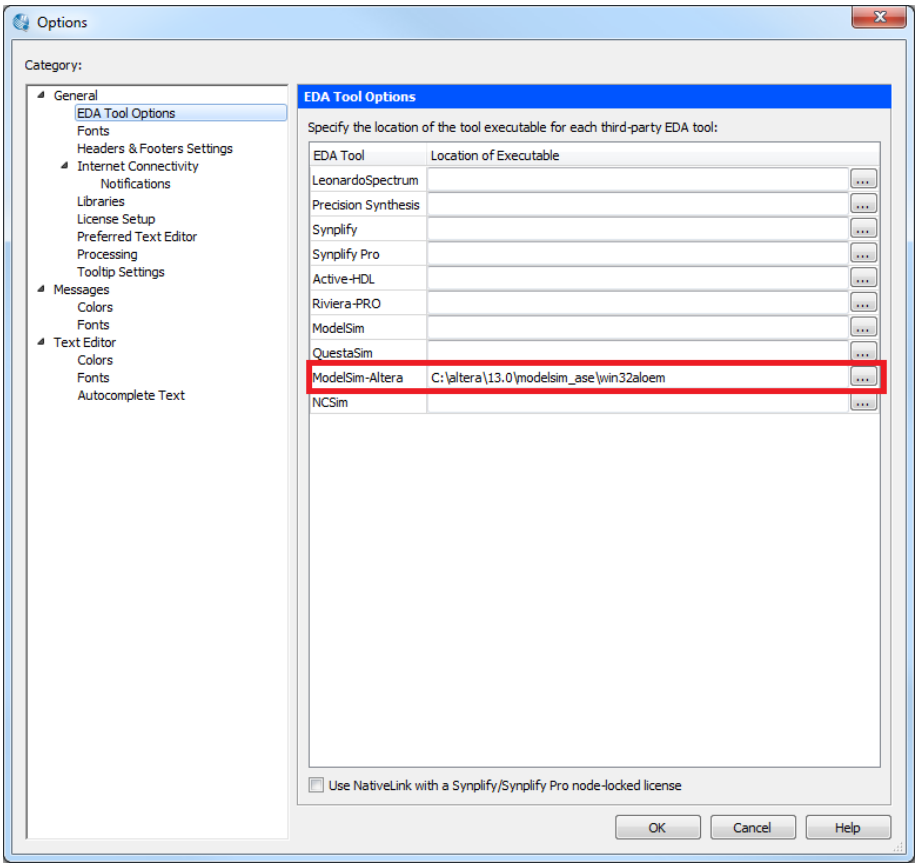
Uvod

Cilj ovog dokumenta je da se studenti upoznaju sa alatima koji im omogućavaju da simuliraju i testiraju hardverske komponente u ranim fazama razvoja kao i da ih testiraju na dostupnim hardverskim platformama.

Simulacija korišćenjem ModelSim-Altera alata

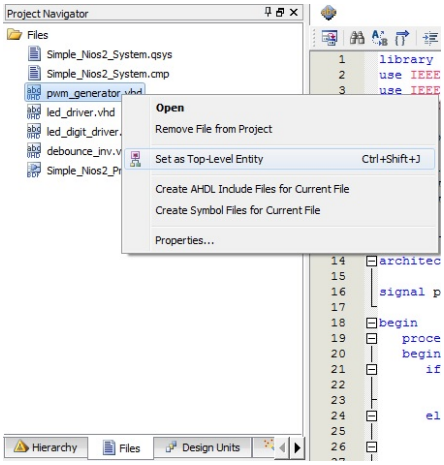
Pri razvoju hardverskih blokova najlakše je proveriti njihovu funkcionalnu ispravnost u simulatoru. ModelSim-Altera nudi 2 nivoa simulacije: funkcionalnu i *gate-level* simulaciju. Funkcionalnom simulacijom se proverava da li je dizajn logički funkcionalno ispravan bez uzimanja u obzir dostupnost različitih resursa na čipu i realnih kašnjenja po linijama veze. *Gate-level* simulacije se obavlja na modelu nakon mapiranja dizajna na odgovarajuću FPGA platformu i uzima u obzir realna kašnjenja. **Pre finalne implementacije dizajna i spuštanja na ploču potrebno je potvrditi da komponente ispravno funkcionišu u gate-level simulaciji.**

Prilikom nove instalacije najpre je potrebno podesiti putanju do izvršnog fajla simulatora u okviru Quartus II alata kao što je prikazano na Slici 1. Prikazani dijalog se dobija biranjem opcije *Tools→Options→EDA Tool Options*. Simulator je obično instaliran na sledećoj adresi C:\altera\<verzija>\modelsim_ase\win32aloem.



Slika 1. Podešavanje putanje do ModelSim-Altera simulatora

Za potrebe demonstracije korišćen je projekat Simple_Nios2_System sa predmeta Digitalni VLSI sistemi. Cilj je da se testira komponenta PWM generatora koja predstavlja sastavni deo ovog sistema. Kako se testira samo komponenta PWM generatora potrebno je ovu komponentu proglasiti za *Top-level* modul desnim klikom na fajl ***pwm_generator.vhd*** u okviru *Project Navigator*-a i biranjem odgovarajuće komande kao što je to prikazano na Slici 2.

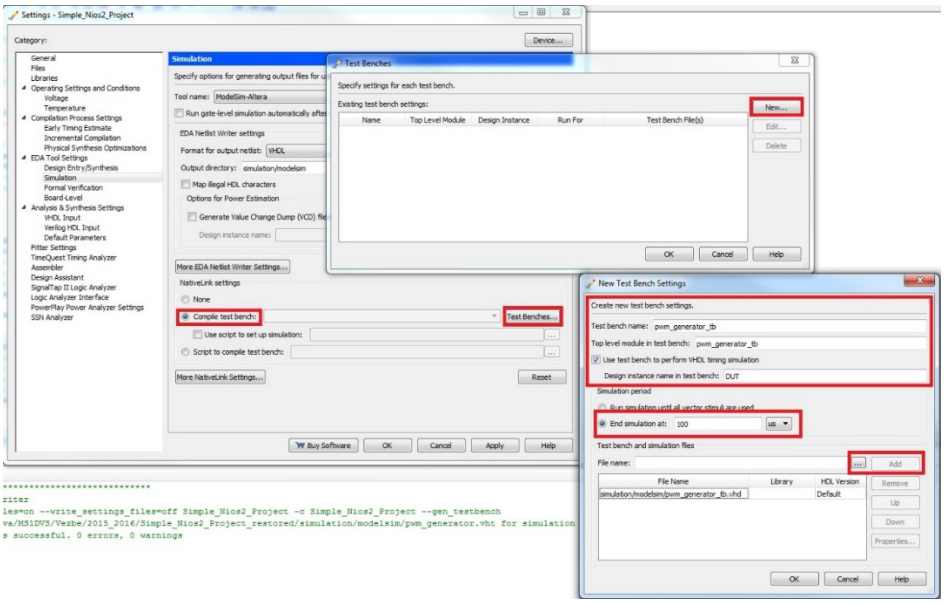


Slika 2. Podešavanje *top-level* komponente

Posle promene *top-level* komponente potrebno je uraditi ponovno kompajliranje celog dizajna.

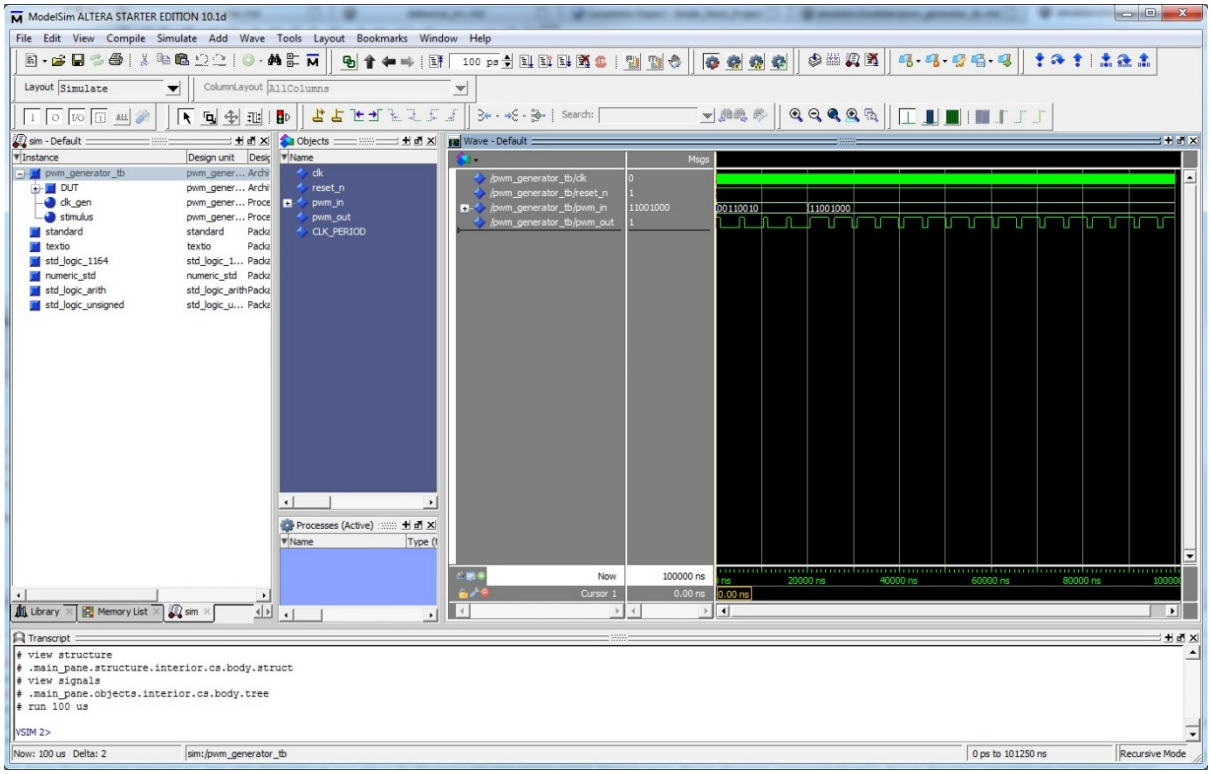
Šablon *test-bench* fajla *top-level* komponente se može automatski izgenerisati biranjem opcije *Processing→Start→Start Test Bench Template Writer*. *Test-bench* fajl sa istim imenom kao i orginalni fajl samo sa ekstenzijom **vht** je kreiran u direktorijumu **/simulation/modelsim/**. Kreirani fajl je potrebno otvoriti i sačuvati pod drugim imenom, recimo pwm_generator_tb.vhd. Ovo je važno uraditi kako bi se izbeglo slučajno prebirsavanje *test bench* fajlova ponovnim pokretanjem *template wirter*-a. Šablon automatski definiše sve potrebne signale, obavlja deklaraciju i instanciranje komponente i definiše dva procesa, jedan za inicijalizaciju sistema a drugi za definisanje pobudnih signala. U slučaju sekvencijalnih komponenti potrebno je definisati poseban proces zadužen za generisanje signala takta.

Nakon kreiranja *test bench* fajla potrebno je podesiti simulaciono okruženje u okviru sledećeg prozora *Assignments→Settings→EDA Tool Settings→Simulation*. U odelju *NativeLink settings* potrebno je odabrati opciju *Compile test bench* i klikom na taster *Test Benches→New* dodati *test bench* kreiran za komponentu PWM generatora.

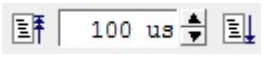


Slika 3. Podešavanje simulacionog okruženja

Nakon podešavanja simulacionog okruženja simulacija se vrlo jednostavno pokreće biranjem odgovarajuće opcije *Tools→Run Simulation Tool→RTL Simulation* (ili *Gate Level simulation*). Pokretanjem simulacije otvara se ModelSim prozor i automatski se pokreće deifnisana simulacija. Kllikom na ikonicu  *Zoom full* se kompletna simulacija prikazuje u okviru simulacionog prozora. Ova ikonica postaje aktivna tek nakon klika na *Wave* prozor simulatora.

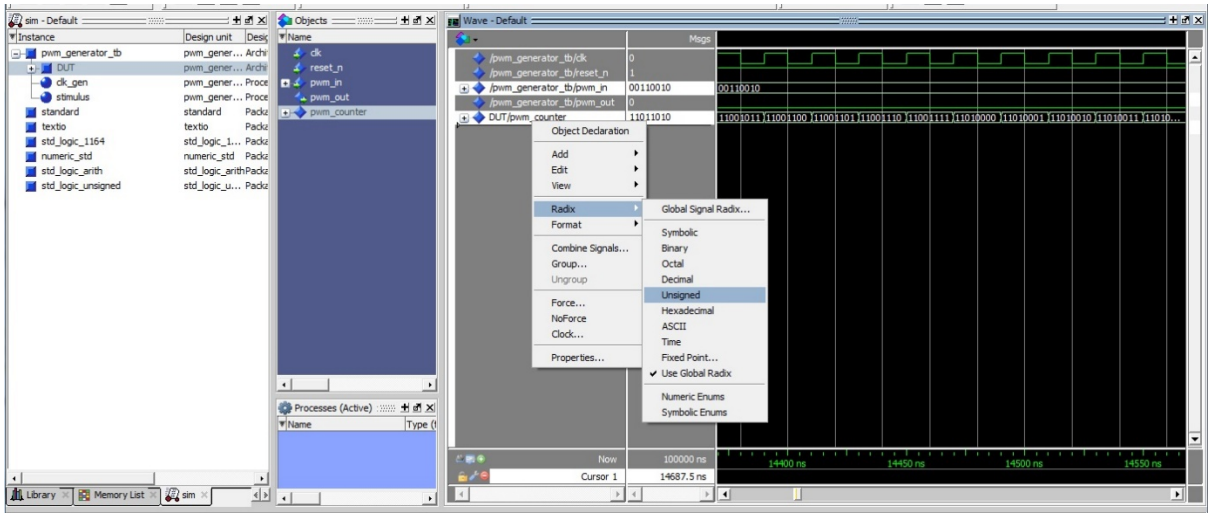


Slika 4. Izgled prozora nakon pokretanja simulatora i zumiranja rezultata

U okviru prozora *Objects* nalaze se svi signali koji su u interfejsu testirane komponente. Ako je potrebno posmatrati neke interne signale oni se mogu dobiti klikom na instancu DUT (naziv zavisi od naziva instance unutar *test bench* fajla) u okviru *pwm_generator_tb* stabla. Sada se u okviru prozora *Objects* pojavljuju svi interni signali pa je moguće posmatrati i signal *pwm_counter* koji se ne nalazi u interfejsu *pwm_generator* komponente. Ovaj signal se može dodati u prozor *Wave* jednostavnim prevlačenjem. Sada je potrebno restartovati i ponovo pokrenuti simulaciju kako bi se prikazali rezultati i za signal *pwm_counter*. Restartovanje i pokretanje simulacije se najlaše obavlja sa *Toolbar*-a pomoću sledećih ikonica 

. Vreme simulacije podesiti na željeni vremenski interval.

Brojni sistem za prikazivanje rezultata simulacije se može jednostavno promeniti selektovanjem odgovarajućih signala u okviru *Wave* prozora i biranjem opcije na primer *Radix→Unsigned* kao što je to prikazano na Slici 5.



Slika 5. Promena brojnog sistema za prikazivanje rezultata

Prilikom ponovnog pokretanja simulacije iz *Quartus*-a svi naknadno dodati signali i podešeni brojni sistemi se gube i potrebno ih je ponovo dodati što može biti prilično frustrirajuće u slučaju simulacije nekog složenijeg dizajna. Na svu sreću trenutno podešavanje simulatora (skup signala koji se testira, brojni sistemi za prikaz) se može sačuvati u okviru posebnog fajla sa ekstenzijom *.do biranjem opcije *File→Save Format*. Prethodno sačuvana podešavanja se mogu učitati biranjem opcije *File→Load→Macro File*.

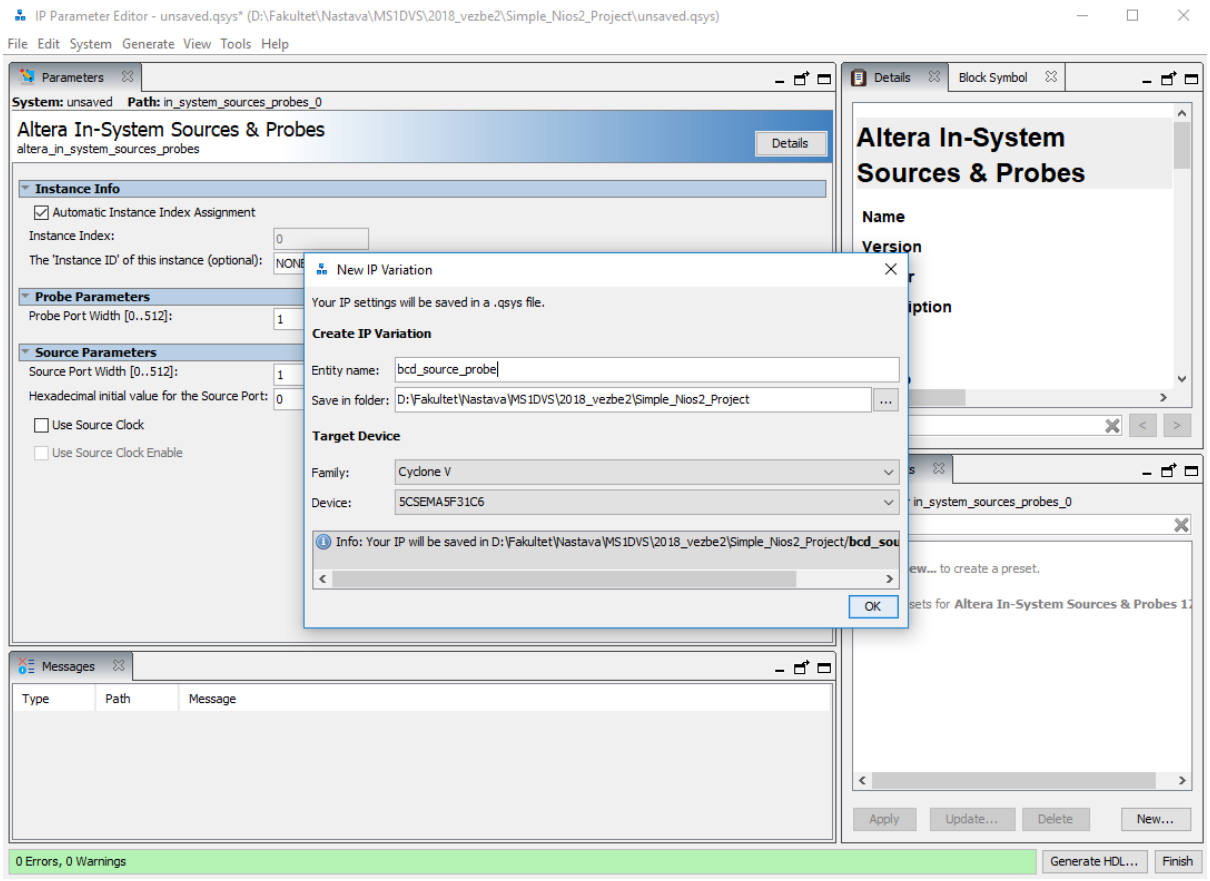
Testiranje na realnom hardveru

Iako *gate-level* simulacija može dati dosta dobru sliku kako će se dizajn ponašati nakon implementacije dosta često se javlja potreba za debagovanjem i posmatranjem internih signala dizajna za vreme rada na realnom hardveru. Na taj način se mogu otkloniti greške koje nisu uočene tokom simulacije i mogu se uočiti reakcije sistema na realne signale koji dolaze sa odgovarajućeg eksternog modula. U nastavku će biti prikazan odgovarajući set alata dostupan u okviru *Quartus*-a koji omogućava posmatranje i postavljanje internih signala u dizajnu za vreme rada na realnom hardveru.

In-System Sources and Probes

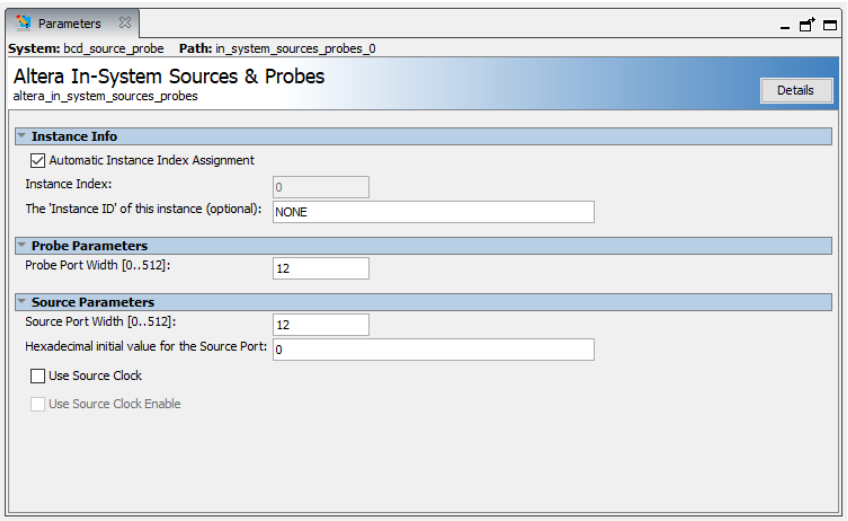
Postavljanje vrednosti ulaznih signala internih komponenti kao i posmatranje njihovih izlaznih signala sa host računara putem JTAG UART interfejsa, se može postići jednostavno korišćenjem alata *In-System Sources/Probes*. Na raspolaganju su gotove komponente koje obezbeđuju vezu sa host računarom i potrebno ih je samo dodati u dizajn i povezati sa odgovarajućim signalima. Kao primer biće korišćena periferija *led_driver*-a iz projekta korišćenog kao primer u predmetu Digitalni VLSI sistemi sa master studija odseka za elektroniku. Ovaj modul prihvata trocifreni broj zadat u BCD kodu (12 bita širina ulaznog signala) i ispisuje ga na LED displeju koji se nalazi na ploči. Ideja je da se pored mogućnosti postavljanja ovog ulaza sa procesora obezbedi mogućnost postavljanja njegove vrednosti „ručno“ sa host računara. Pored toga potrebno je obezbediti mogućnost očitavanja izlaza *bcd_out* izlaza procesorkog sistema na host računaru. U tu svrhu potrebno je dodati komponentu tipa Altera In-System Sources & Probes kao i jedan multiplekser LPM_MUX. Gotove komponente se dodaju u sistem korišćenjem opcije *Tools → IP Catalog*. Nakon biranja ove komande sa desne strane se otvara se prozor sa svim dostupnim IP komponentama. Komponenta multipleksera se nalazi pod *Basic Functions → Miscellaneous → LPM_MUX* dok se komponenta *In-System Sources/Probes* nalazi pod *Simulation; Debug and Verification → Debug and Performance → Altera In-System Sources & Probes.*

Dvostrukim klikom na komponentu otvara se prozor *IP Parameter Editor*-a prikazan na Slici 6. Najpre je potrebno je navesti ime komponente koja će biti kreirana. U našem slučaju ime komponente je ***bcd_source_probe***.



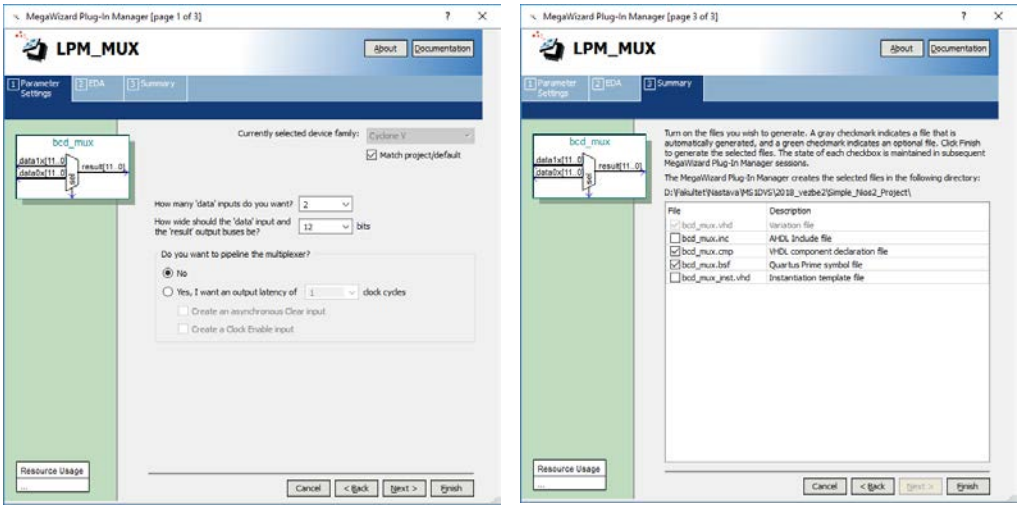
Slika 6. Dodavanje *In-System Sources and Probes* komponente

Nakon toga se otvara prozor za podešavanje parametara komponente. Kako je ideja da navedena komponenta postavlja vrednosti ulaza *bcd_in* komponente *led_driver*, ali i da u isto vreme omogući posmatranje *bcd_out* izlaza procesorskog dela sistema to je potrebno podesiti 12 bita širinu i za *source* i za *probe port* kao što je to prikazano na Slici 7.



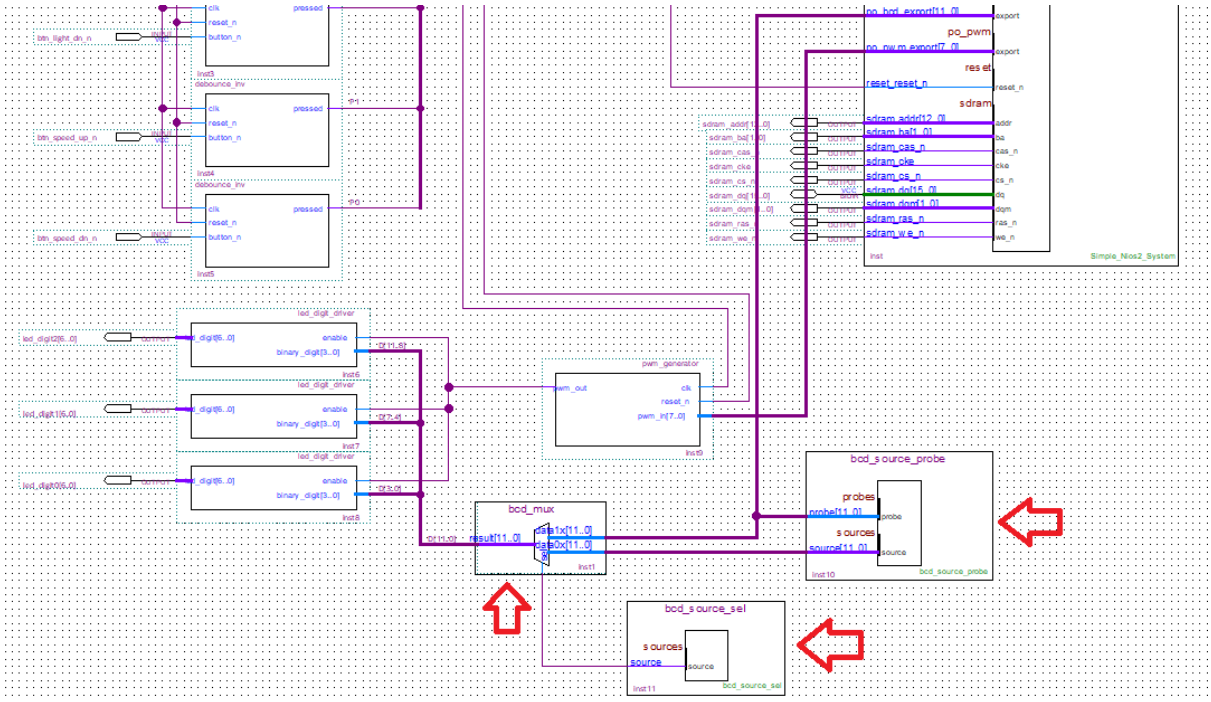
Slika 7. Podešavanje parametara *In-System Sources and Probes* komponenti

Nakon podešavanja parametara generisati fajlove nove komponente klikom na *Generate HDL* u donjem levom uglu prozora. Kako će ove komponente biti dodavane u okviru *Schemtic*-a obavezno štiklirati opciju za generisanje *.bsf* fajla simbola. Prateći sličan postupak potrebno je kreirati komponentu multipleksera 12bit 2 u 1 sa nazivom ***bcd_mux*.** Pošto predstavlja jednostavnu komponentu kreiranje multipleksera ne koristi *IP Parameter Editor* već stariji *MegaWizard Plug-In Manager*. Kod ovog alata se podešavanje generisanja .bsf fajla obavlja u Summary kartici kao što je prikazano na Slici 8.



Slika 8. Podešavanje parametara *LPM_MUX* komponente

Kako se selekcioni port multipleksera kontroliše sa host računara potrebno je dodati još jednu komponentu *In-System Sources and Probes* ***bcd_source_sel*** sa *source* portom širine 1 bit i bez *probe* porta koja služi za selekciju odgovarajućeg ulaza multipleksera. Nakon toga je potrebno novokreirane komponente dodati u sistem kao što je prikazano na Slici 9.



Slika 9. Povezivanje dodatih komponenti u već postojeći sistem

Da bi kompajliranje bilo uspešno potrebno je dodati novokreirane komonente *bcd_source_probe.qsys* i *bcd_source_sel.qsys* u projekat preko menija *Project → Add/Remove Files to Project*. Nakon toga je potrebno je iskompajlirati ceo dizajn, isprogramirati FPGA čip i pokrenuti softverski deo sistema. Kontrola *Source* i *Probe* instanci se obavlja pomoću *In-System Sources and Probes* editora koji se pokreće iz *Tools* menija glavnog prozora *Quartus*-a.

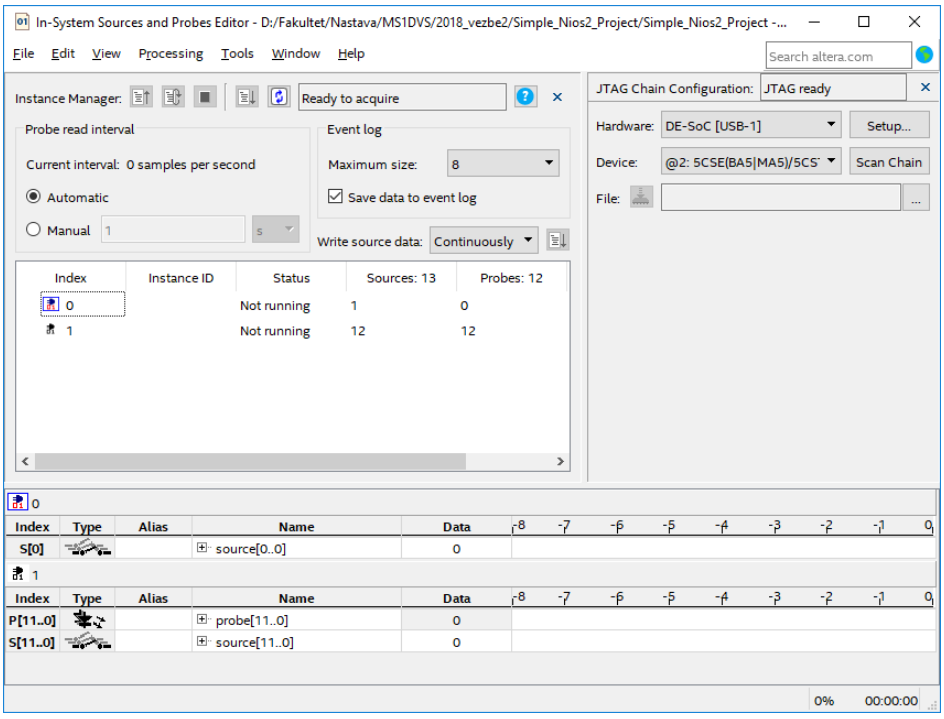
Po pokretanju dobija se prozor prikazan na Slici 10. U donjem delu prozora su prikazani *source* i *probe* signali iz dodatih komponenti. U ovom delu, mogu se podešavati željene vrednosti *source* signala. U slučaju da je *Write source data* podešeno na kontinualno upisivanje ove promene će se odmah preneti u sistem.

Očitavanja *probe* komponenti se postiže klikom na ikonicu  . Kontinualno očitavanje se postiže klikom na ikonicu  . Učestanost očitavanja nije velika i podrazumevana vrednost je 9 odbiraka u sekundi. Učestanost odabiranja se može promeniti ali nije moguće postići očitavanja u realnom vremenu.

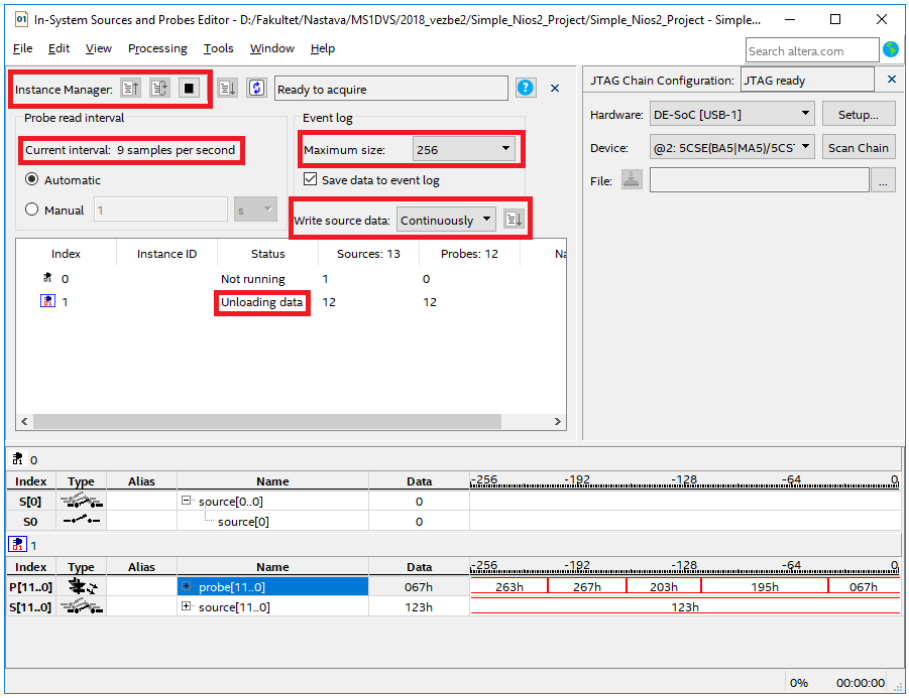
Veličina bafera se takođe može promeniti iz *editor*-a. Podrazumevana vrednost je 8 odbiraka dok je na Slici 11. veličina bafera promenjena na 256 odbiraka.

Format prikaza podataka na magistralama se može promeniti pomoću opcije *Edit → Bus Display Format*.

Sumirano *In-System Sources and Probes* alat omogućava postavljanje signala unutar sistema pomoću korisničkog interfejsa sa host računara što omogućava veću kontrolu prilikom debagovanja sistema. Omogućeno je i posmatranje signala unutar sistema ali uz izvesna ograničenja po pitanju pristupa (nije moguće posmatrati interne signale) i učestanosti odabiranja.



Slika 10. *In-System Sources and Probes* editor

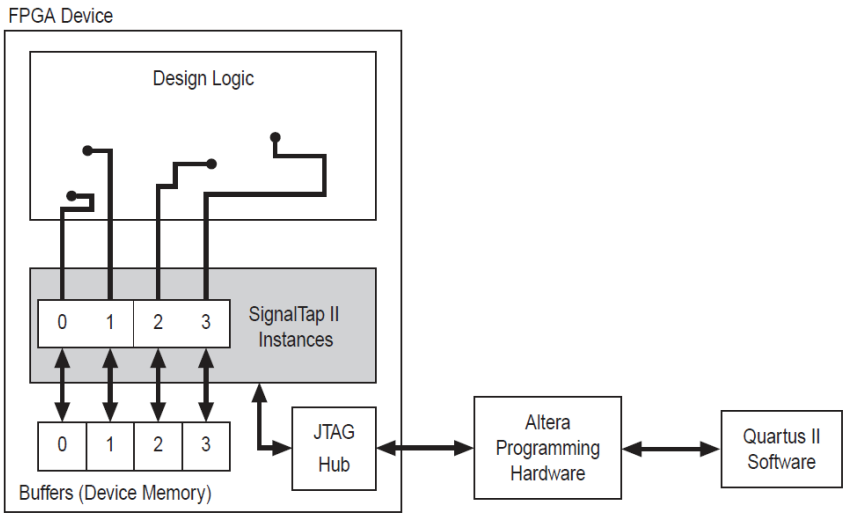


Slika 11. *In-System Sources and Probes* editor tokom rada

SignalTap II Logic Analyzer

Iako omogućava promenu i posmatranje sadržaja određenih internih signala za vreme rada na realnoj hardverskoj platformi *In-System Sources and Probes* alat nije pogodan za ozbiljnije testiranje s obzirom da ne omogućava snimanje signala u realnom vremenu i ne omogućava posmatranje internih signala komponenti u dizajnu.

SignalTap II logički analizator omogućava posmatranje bilo kog internog signala u dizajnu i prikazivanje sačuvanih odbiraka nakon pojave odgovarajućeg triger signala. Osnovna arhitektura logičkog analizatora prikazana je na Slici 12.

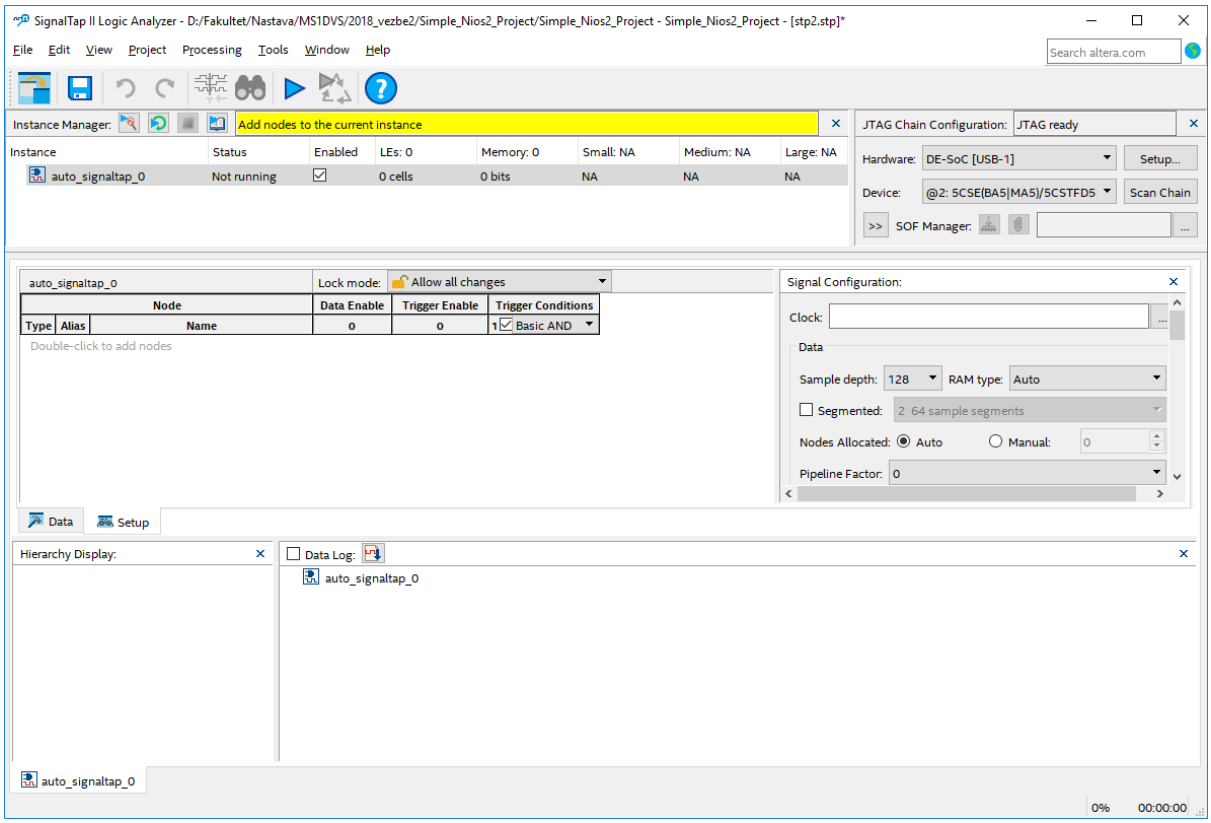


Slika 12. Arhitektura *SignalTap II* logičkog analizatora

Ideja je da se za svaki posmatrani signal rezerviše bafer u internoj memoriji FPGA čipa u kojem se čuvaju odbirci posmatranih signala. Bafer je cirkularnog tipa i za vreme rada nove vrednosti snimljenih odbiraka se prepisuju preko starih vrednosti. Nakon pojave određenog logičkog uslova koji

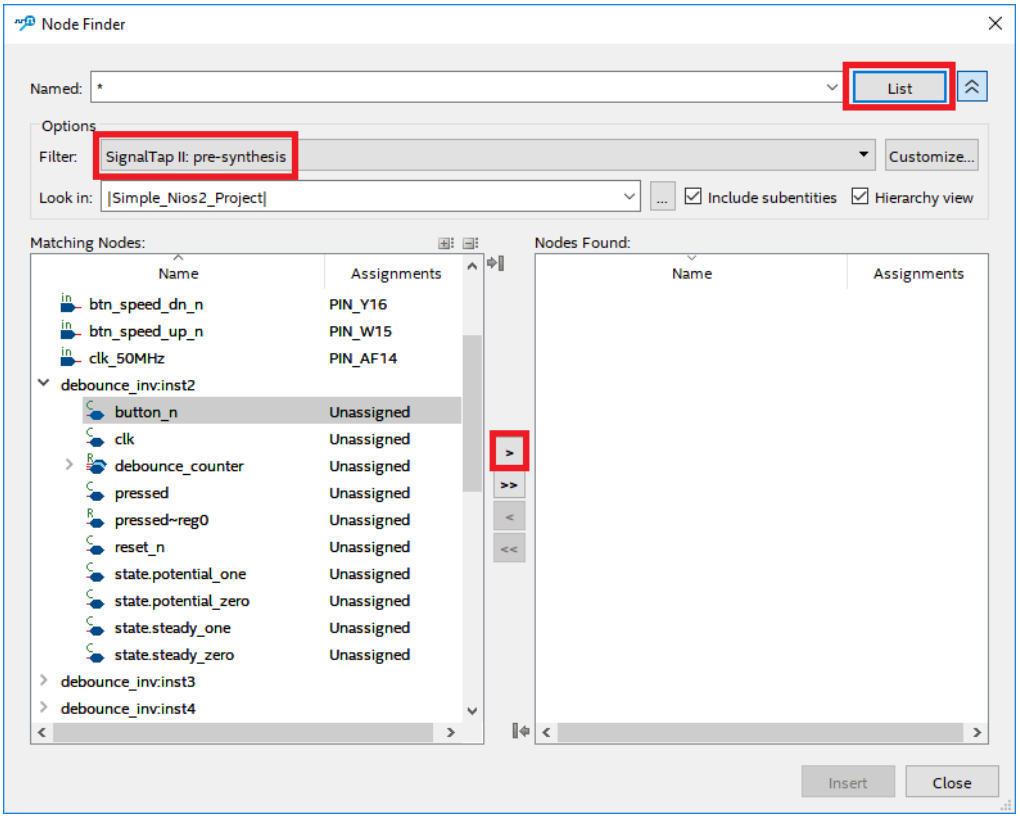
se naziva triger, sadržaji lokalnih bafera se šalju serijskim putem na host računar gde se mogu dalje analizirati. Interesantno je da za korišćenje ovog načina testiranja nije potrebno menjati sistem, izvlačiti interne signale u interfejs komponente ili koristiti neke dodatne eksterne pinove.

Kako bi započeli korišćenje *SignalTap II* alata potrebno je najpre kompajlirati postojeći dizajn (izbaciti prethodno dodate komponente sources, probes i mux) a zatim pokrenuti *SignaTap II Logic Analyzer* iz menija *Tools* u okviru glavnog prozora *Quartus* alata, nakon čega se otvara prozor prikazan na Slici 13.



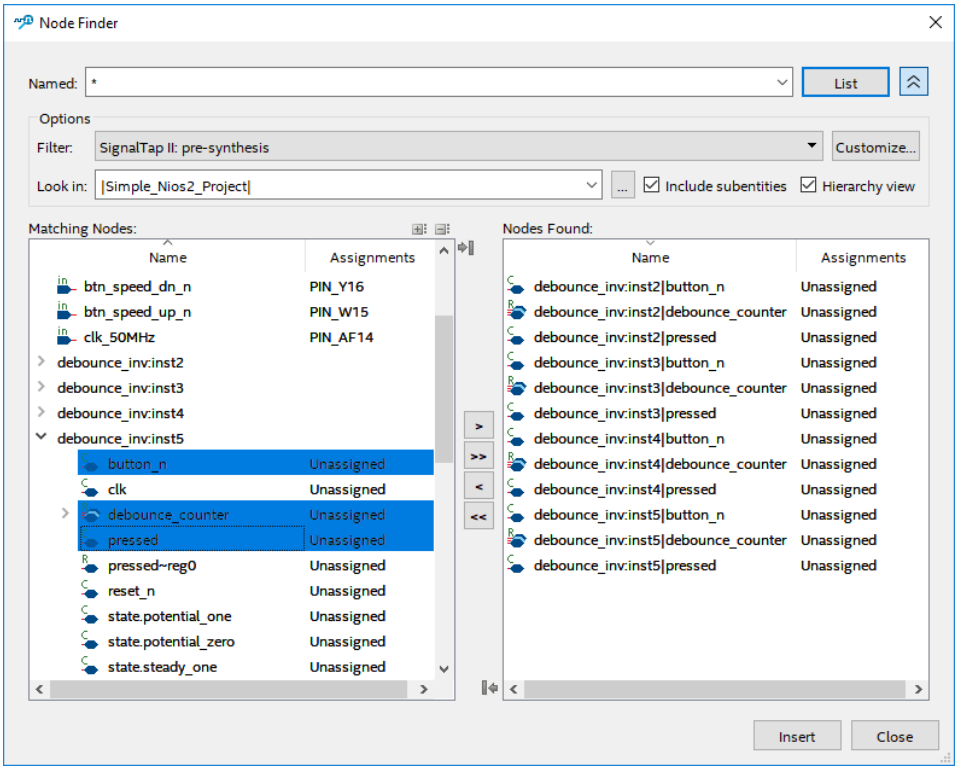
Slika 13. Početni prozor *SignalTap II Logici Analyzer*-a

U centralnom delu ovog prozora u okviru odeljka *Setup* mogu se dodati signali koji će se posmatrati tokom testiranja. Signali se dodaju dvostrukim klikom ma prazan prostor polja *Setup* nakon čega se otvara novi prozor *Node Finder*. U okviru ovog prozora mogu se dodati signali koji će biti posmatrani tokom testiranja. Obezbeđeni su različiti filtri kako bi se prikazivali samo signali od interesa a takođe je moguće pretraživati odgovarajuće signale po imenu. Prilikom sinteze alat obavlja optimizaciju i menja imena određenih signala tako da je mnogo jednostavnije baratati sa imenima signala pre sinteze. Potrebno je podesiti filtar na *SignalTap II: pre-synthesis* kako bi se prikazala imena signala pre njihove izmene u procesu sinteze i optimizacija a koja odgovaraju originalnim imenima u dizajnu. Klikom na taster *List* izlistavaju se svi signali u dizajnu koji odgovaraju zadatom filtru. Za potrebe ove demonstracije biće testirana komponenta *debounce_inv*. Dodavanje signala koji će biti posmatrani se obavlja selekcijom odgovarajućeg signala u levom delu prozora *Node Finder* i klikom na odgovarajuću strelicu koja se nalazi na sredini prozora kao što je to prikazano na Slici 14.



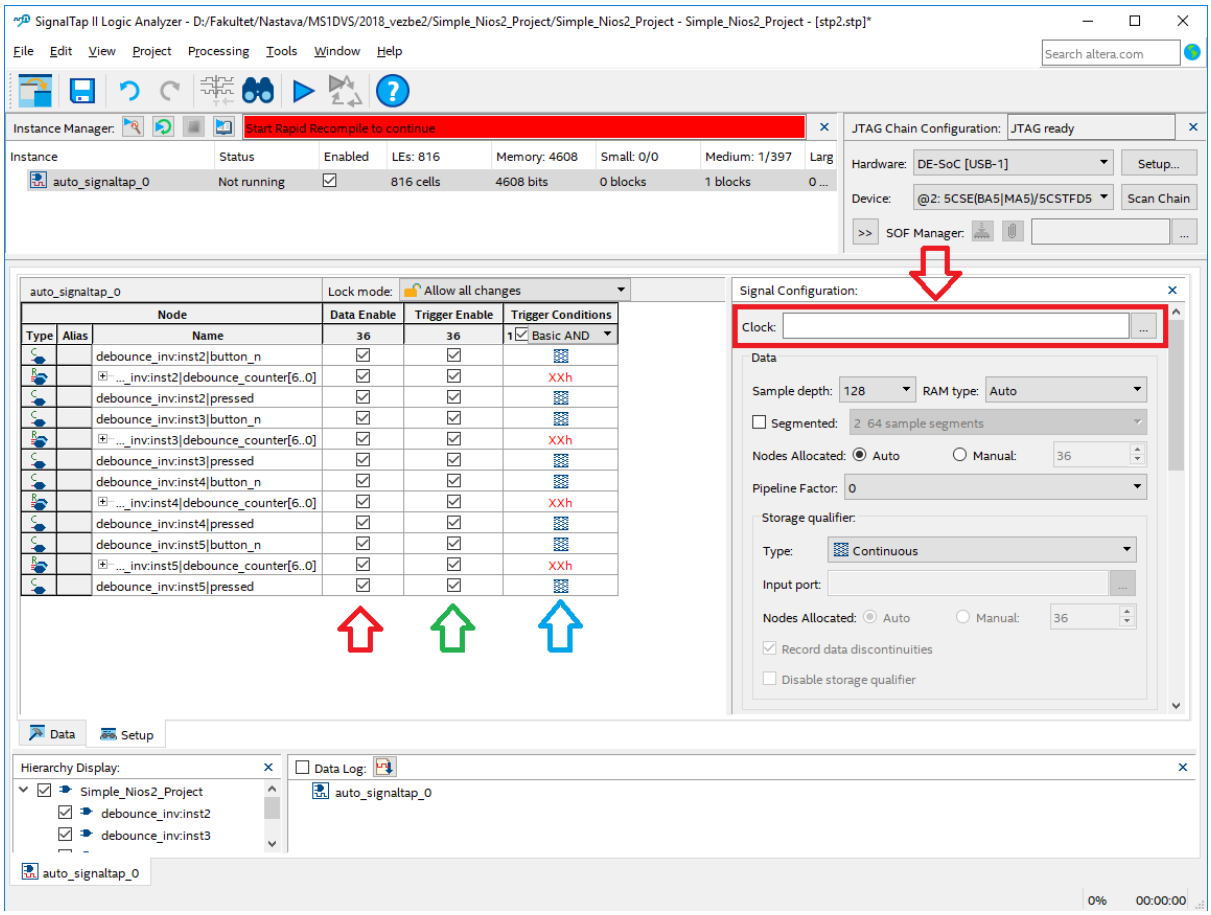
Slika 14. Dodavanje signala koji će biti posmatrani tokom testiranja

Potrebno je dodati signale ***button_n***, ***debounce_counter*** i ***pressed*** iz svake od 4 prisutne komponente *debounce_inv*. Nakon toga prozor *Node Finder*-a bi trebalo da izgleda kao na Slici 15. i klikom na taster Insert selektovani signali se dodaju u trenutnu instancu *SignalTap II*.



Slika 15. Selektovanje svih signala koji će biti posmatrani u okviru *SignalTap II Logic Analyzer*-a

Nakon dodavanje selektovnaih signala osnovni prozor *SignalTap II Logic Analyzer*-a trebalo bi da ima izgled kao na Slici 16.



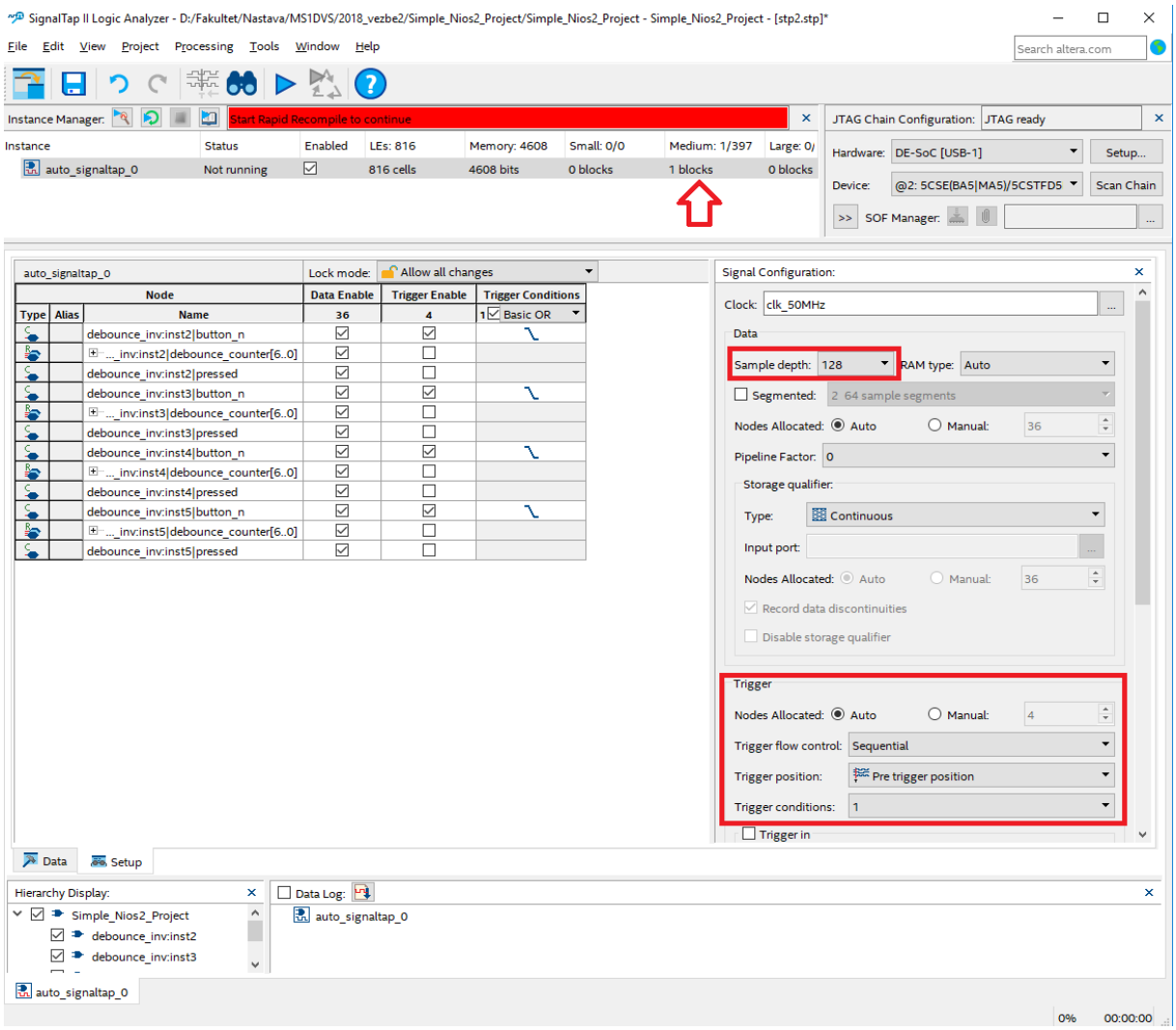
Slika 16. Izgled *SignalTap II Logic Analyzer*-a nakon dodavanja signala

Odbirci posmatranih signala se snimaju na svaku uzlaznu ivicu signala takta koji je takođe potrebno definisati. Kod sinhronog dizajna postoji jedan signal takta i njega je potrebno koristiti i za odabiranje posmatranih signala. Kako sve komponenta *debounce_inv* koristi eksterni takt učestanosti od 50 MHz to je potrebno ovaj takt podesiti u polju *Clock* klikom na tri tačke i biranjem odgovarajućeg signala ***clk_50MHz*** u *Node Finder*-u.

Pored svakog signala u okviru centralnog prozora na Slici 16. mogu se uočiti tri polja: *Data Enable* (crvena strelica), *Trigger Enable* (zelena strelica) i *Trigger Conditions* (plava strelica). Polje *Data Enable* definiše da li odbirke odgovarajućeg signala treba čuvati za kasnije prikazivanje. Polje *Trigger Enable* označava da li vrednost odgovarajućeg signala utiče na pojavu triger događaja nakon kog se sačuvani odbirci šalju ka računaru. I polje *Trigger Conditions* omogućava definisanje uslova za odgovarajući signal koji utiču na pojavu triger signala.

U okviru ovog primera testiranje će biti podešeno tako da se svaki put kada se pritisne neki od 4 prisutna tastera podaci pošalju na računar. Ovaj uslov određuje stanje signala ***button_n*** tako da je na svim ostalim signalima potrebno deaktivirati *Trigger Enable*. Silazna ivica na bilo kom od tastera označava da je taster pritisnut i može se koristiti kao detekcija pritiska tastera. Desnim klikom na polje *Trigger Condition* potrebno je odabrati odgovarajuću vrednost uslova za sva 4 tastera. Finalni triger predstavlja logičku funkciju svih pojedinačno definisanih triger uslova. Logička funkcija može biti jednostavna I i ILI funkcija ili neki složeniji uslov (*Advanced*). Kako je podrazumevana vrednost I logička

funkcija potrebno ju je promeniti u logičko ILI. Nakon ovih podešavanja dobija se prozor kao na Slici 17.



Slika 17. *SignalTap II* nakon podešavanja triger-a

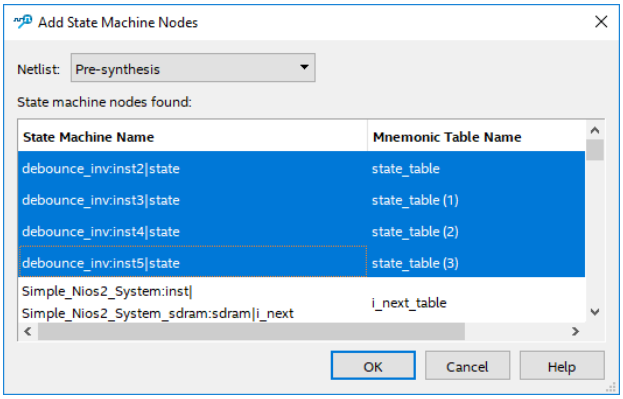
Broj odbiraka koji je će biti prikazan zavisi od veličine bafera koji se koristi i može se podesiti u okviru polja *Sample Depth* označenog na Slici 17. Ovde je potrebno voditi računa da se instanca *SignalTap II Logic Analyzer*-a kao i baferi realizuju unutar resursa FPGA čipa i od dostupnosti ovih resursa zavisi koju veličinu bafera je moguće realizovati. *SignalTap II* omogućava ranu procenu zauzeća resursa (označeno crvenom strelicom na Slici 17), i u slučaju pokušaja alokacije prevelikih bafera ispisuje se poruka *Can't fit*... Za potrebe ovog primera potrebno je podesiti dubinu bafera na 1K.

Moguće je takođe podesiti vremenski položaj triger signala na jedan od tri predefeinisanih:

1) *Pre trigger position* - po pojavi trigera ka računaru se šalju odbirci signala koji su došli nakon trigera
2) *Center trigger position* - po pojavi trigera ka računaru se šalju odbirci signala od kojih se polovina desila pre a polovina posle trigera
3) *Post trigger position* - po pojavi trigera ka računaru se šalju odbirci signala koji su se desili pre trigera

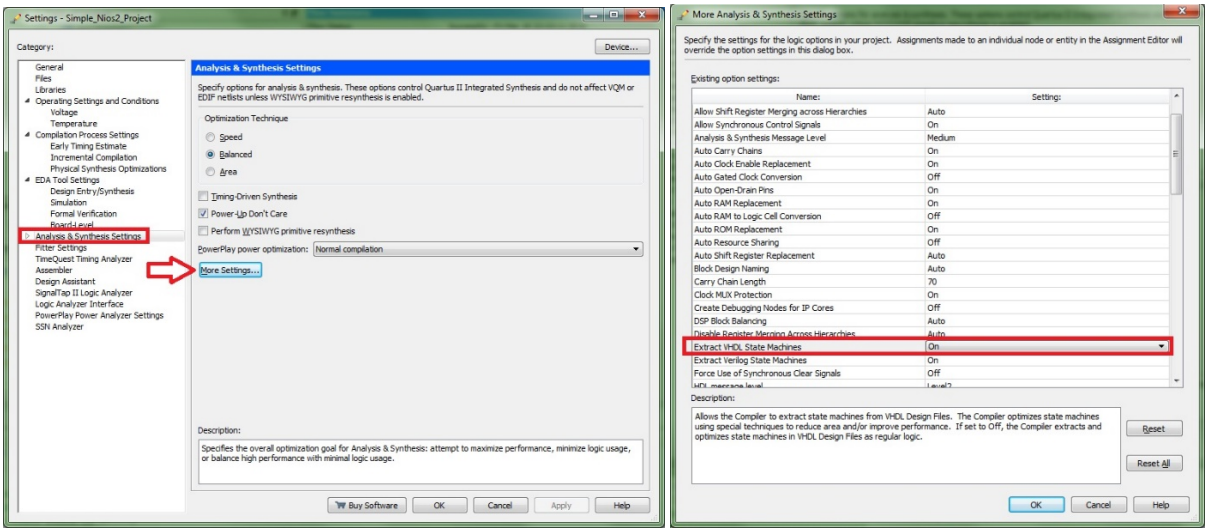
Za potrebe ovog primera ostaviti *Pre trigger position*.

Pored standardnih signala moguće je posmatrati i mašine stanja sa simboličkim imenima koja su im dodeljena prilikom pisanja koda. Mašine stanja se dodaju desnim klikom na prazan prostor *Setup* prozora i biranjem opcije *Add State Machine Nodes…* kao što je prikazano na Slici 18.



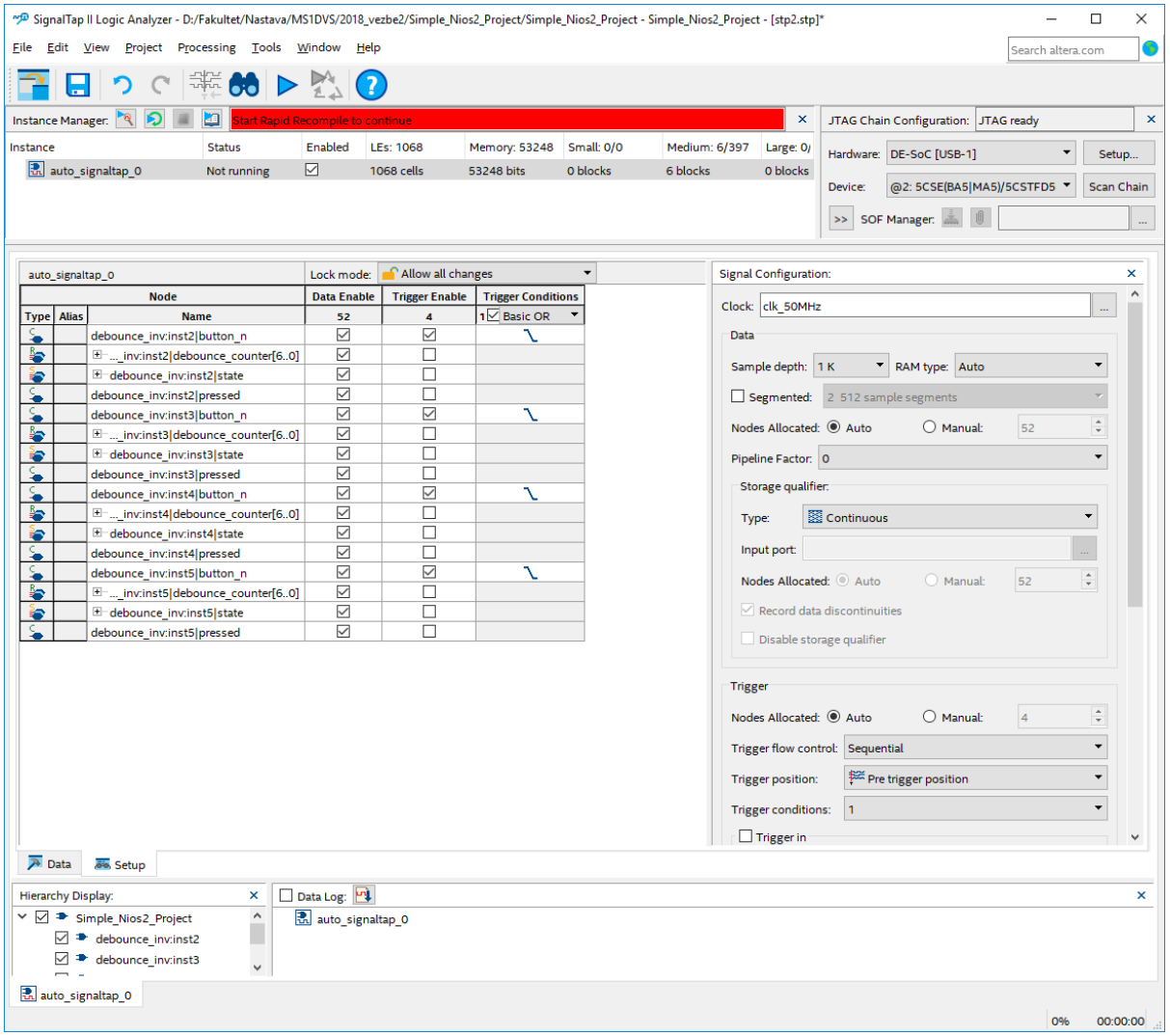
Slika 18. Dodavanje mašina stanja

Quartus automatski prilikom sinteze obavlja ekstrakciju mašina stanja. Ako interne mašine stanja nisu prepoznate to znači ili da su pisane na nestandaran način pa ih alat nije prepoznao ili je isključena opcija automatske ekstrakcije mašina stanja. U slučaju drugog problema, automatska ekstrakcija se može ponovo uključiti iz *Quartus*-a sledećim postupkom. Potrebno je iz menija *Assignments* odabrati opciju *Settings…* Nakon toga se otvara prozor kao na Slici 19. U okviru odeljka *Analysis & Synthesis Settings* potrebno je kliknuti na taster *More Settings…* Time se otvara novi prozor prikazan na Slici 19. u okviru kojeg je potrebno uključiti opciju ***Extract VHDL State Machines***. Nakon uključivanja ove opcije potrebno je ponovo kompajlirati ceo dizajn kako bi se obavila ekstrakcija mašina stanja.



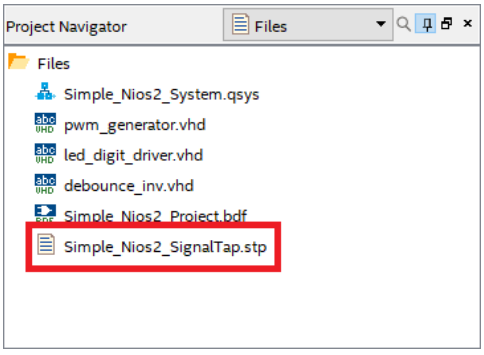
Slika 19. Uključivanje automatske ekstrakcije mašina stanja

Nakon svih podešavanja prozor *SignalTap II* bi trebalo da izgleda kao na Slici 20.



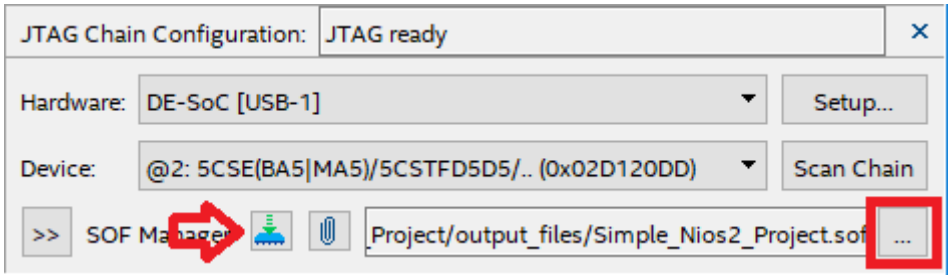
Slika 20. *SignalTap II* nakon dodavanja mašina stanja

Sačuvati trenutnu komponentu *SignalTap II* korišćenjem prečice **Ctrl+S**. Proveriti u okviru *Quartus*-a da li je dodata prethodno sačuvana komponenta kao na Slici 21, i ako nije, dodati je ručno i zatim pokrenuti kompajliranje celog dizajna.



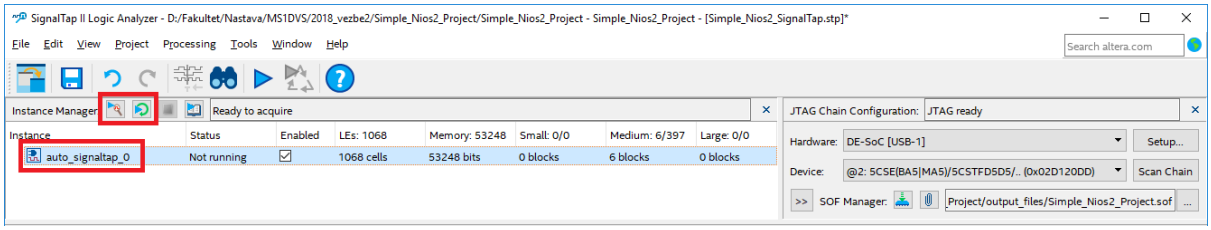
Slika 21. Komponenta *SignalTap II* uključena u *Quartus* projekat

Nakon završetka procesa kompajliranja potrebno je programirati FPGA iz *SignalTap II* alata postavljanjem odgovarajućeg ***SOF*** fajla i klikom na ikonicu za programiranje kao što je prikazano na Slici 22.



Slika 22. Programiranje FPGA čipa iz *SignalTap II* alata

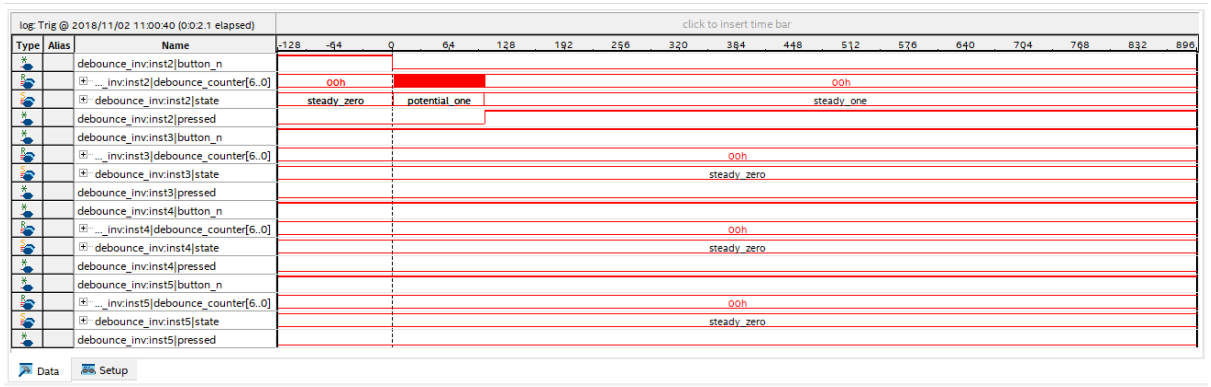
Nakon programiranja potrebno je pokrenuti analizu selektovanjem odgovarajuće instance u okviru *SignaTap II* alata i klikom na ikonicu **Run Analysis** ili **Autorun Analysis** . *Autorun* omogućava da se svaki put po ispunjenju triger uslova novi podaci prikazuju u okviru *SignalTap II* alata dok obična analiza podrazumeva da će biti ispisani podaci nakon prvog trigera po pokretanju analize, nakon čega se analiza mora ponovo pokrenuti.



Slika 23. Pokretanje analize

U slučaju da se testira komponenta čiji rad zavisi od rada procesora potrebno je pre testiranja isprogramirati odgovarajući *Nios II* procesor. Kako komponente za debaunsiranje rade nezavisno od procesora, u ovom slučaju nije potreban ovaj dodatni korak.

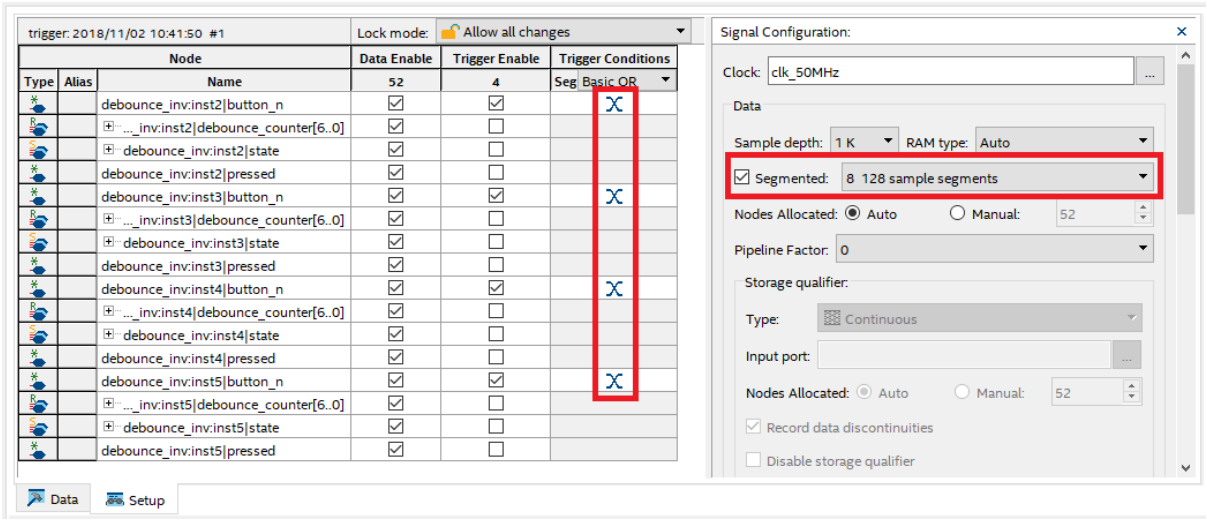
Nakon pritiska na bilo koji taster aktivira se triger i sačuvani podaci se prikazuju u okviru odeljka Data kao što je prikazano na Slici 24.



Slika 24. Vremeski oblici posmatranih signala u okviru *SignalTap II* alata

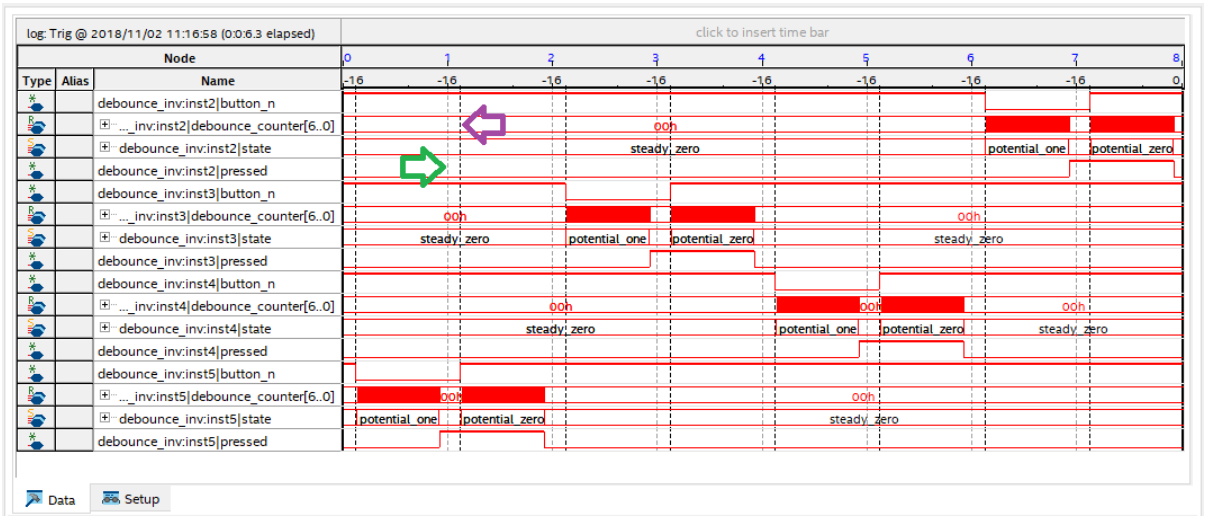
Na Sllici 24. vidi se jedan prelaz ulaznog signala i rad internog brojača dok su ostatak vremena signali na konstatnim vrednostima, tako da veliki deo bafera ne nosi nikakvu novu informaciju. Kako je komponenta za debaunsiranje projektovana tako da je ulazni signal potrebno da bude stabilan 100 perioda signala takta da bi se promenio izlaz, to je dovoljno nešto više od 100 odbiraka kako bi se snimio ceo prelazni proces. Kako bi se što bolje iskoristio dostupni bafer moguće ga je izdeliti na više jednakih celina tako da se svaka celina popunjava sa novom pojavom triger uslova. U posmatranom primeru bafer veličine 1K odbiraka se može podeliti na 8 bafera veličine 128 odbiraka biranjem

odgovarajuće opcije u delu prozora *Data* kao što je to prikazano na Slici 25. Na ovoj slici je podešeno da se triger generiše na bilo koju ivicu ulaznog signala a ne samo na silaznu kako je bilo podešeno ranije.



Slika 25. Redefinisanje triger uslova i podešavanje segmentiranog bafera

Nakon kompajliranja dizajna, programiranjem FPGA i pokretanjem procesa analize započinje se prikupljanje podataka. Svaki pritisak na taster će popuniti minimum dva segmenta bafera (prilikom pritiska i otpuštanja tastera). Dakle potrebno je 4 puta pritisnuti tastere kako bi se popunili svi dostupni baferi i rezultat prikazao u okviru *Data* odeljka. Dobijeni rezultat je prikazan na Slici 26.



Slika 26. Vremenski oblici posmatranih signala u okviru *SignalTap II* alata pri korišćenju segmentiranog bafera

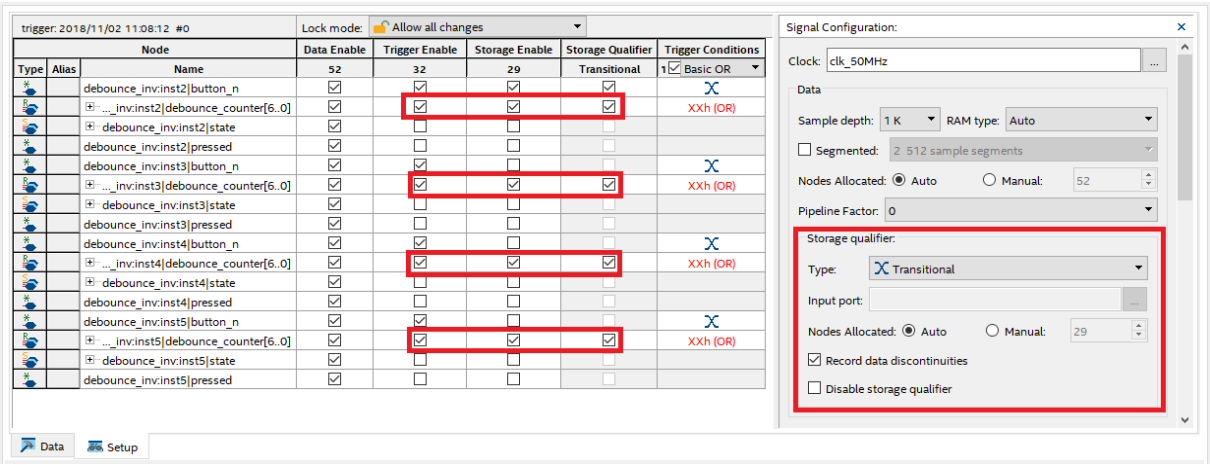
Na Slici 26. zelena strelica pokazuje na isprekidanu liniju koja predstavlja granicu između trenutnog i narednog segmenta bafera, dok ljubičasta strelica pokazuje na isprekidanu liniju koja označava trenutak pojave triger uslova.

Drugi način za uštedu prostora u baferu je selektivno skladištenje odbiraka. Za razliku od podrazumevanog kontinualnog čuvanja odbiraka posmatranih signala moguće je definisati uslove pod kojima će odbirci biti čuvani. Dakle odbrici posmatranih signala će biti sačuvani u baferu jedino ako je

ispunjen definisani uslov (*Storage Qualifier*) dok će u suprotnom biti ignorisani. Opcije dostupne za uslovno skladištenje su:

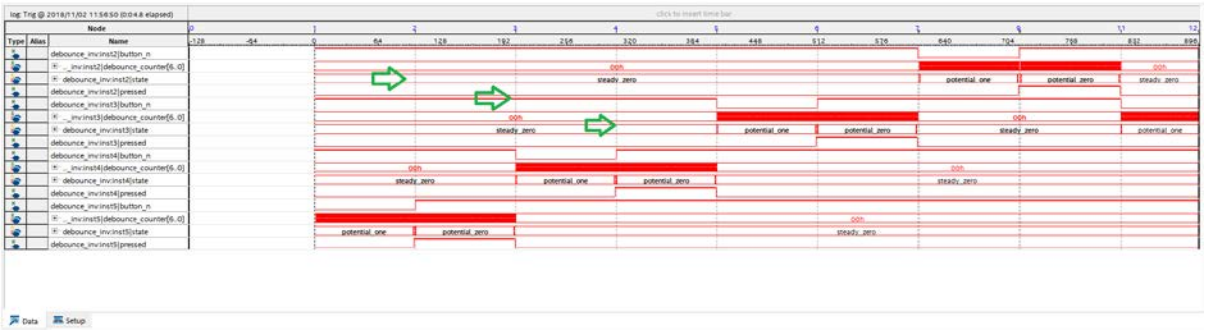
1) *Continuous* – kontiualno odabiranja, odbirci se čuvaju na svaki takt.
2) *Input port* – odbirci posmatranih signala se čuvaju samo u onim trenucima kada signal definisan kao Input port ima visoku vrednost.
3) *Transitional* – odbirci posmatranih signala se čuvaju samo u onim trenucima kada specificirani signali menjaju vrednost.
4) *Conditional* – odbrici ulaznih signala se čuvaju kada je ispunjen odgovarajuću uslov koji se specificira na identičan način kao uslov za triger signal.
5) *Start-stop* – definišu se uslovi za početak i kraj skladištenja odbiraka posmatranih signala

U posmatranom primeru cilj je da se čuvaju obrici signala samo u prelaznom procesu odnosno u trenucima dok brojač menja svoju vrednost. Iz tog razloga je odabran uslov *Transitional* i testiranje je podešeno kao na Slici 27. **Obratiti pažnju da je neophodno uključiti i *Storage Enable* i *Trigger Enable* za signale koji se koriste za uslovno skladištenje!**



Slika 27. Podešavanje uslovnog čuvanja odbiraka

Nakon kompajliranja dizajna, programiranjem FPGA i pokretanjem procesa analize započinje se prikupljanje podataka. Svaki pritisak na taster će izazvati odgovarajuću promenu stanja i pokrenuti rad brojača. Nakon nekoliko pritisaka na tastere bafer se popunjava i rezultat se prikazuje u odeljku Data. Dobijeni rezultat je prikazan na Slici 28.



Slika 28. Vremenski oblici posmatranih signala u okviru *SignalTap II* alata pri korišćenju uslovnog skladištenja

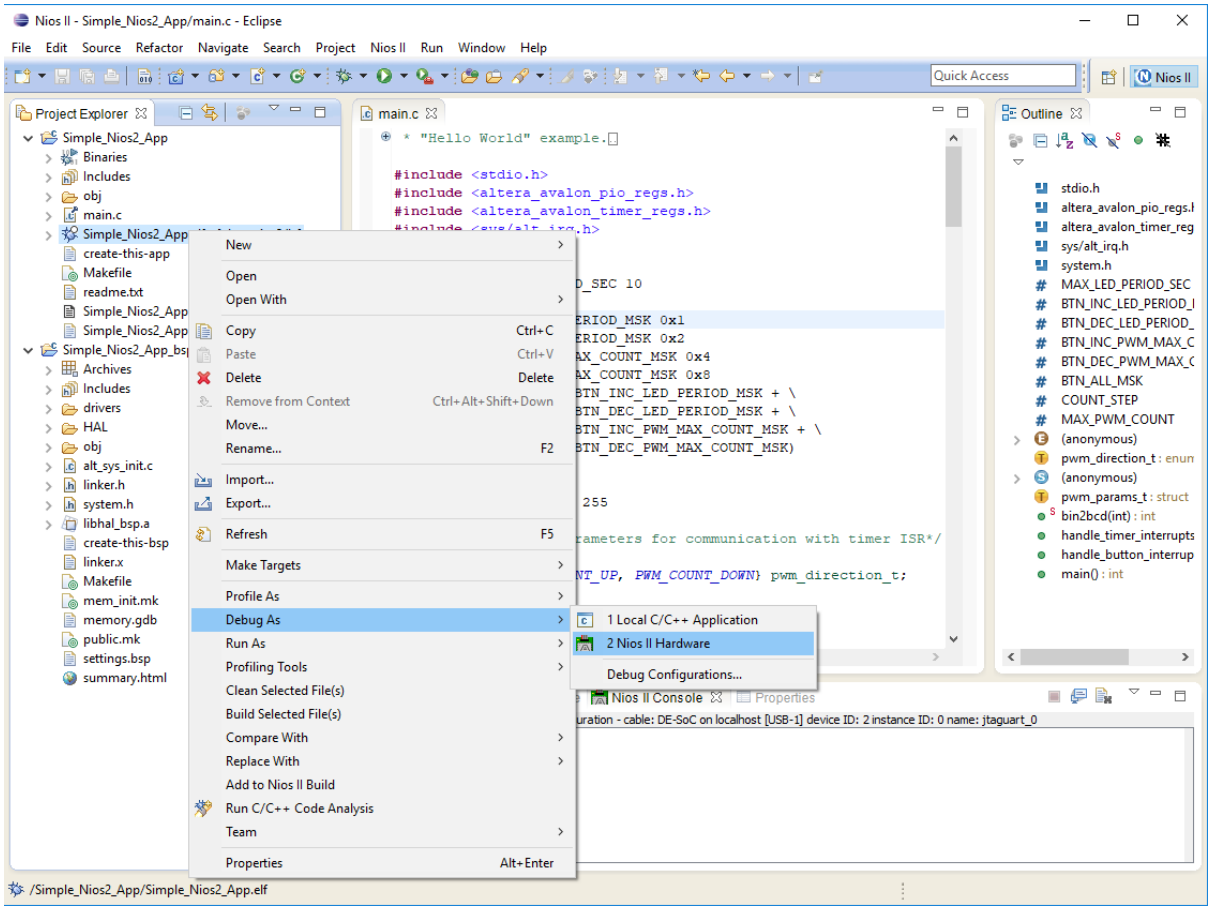
Zelenim strelicama na Slici 28. su označene linije koje predstavljaju diskontinuitete, odnosno odbirci sa jedne i sa druge strane tih linija dogodili su se u razmaku većem od 1 taktnog intervala. Na

ovaj način omogućeno je opet kreiranje segmentiranog bafera s tim što sada segmenti ne moraju biti jednakih dimenzija.

Testiranje softverskog dela sistema

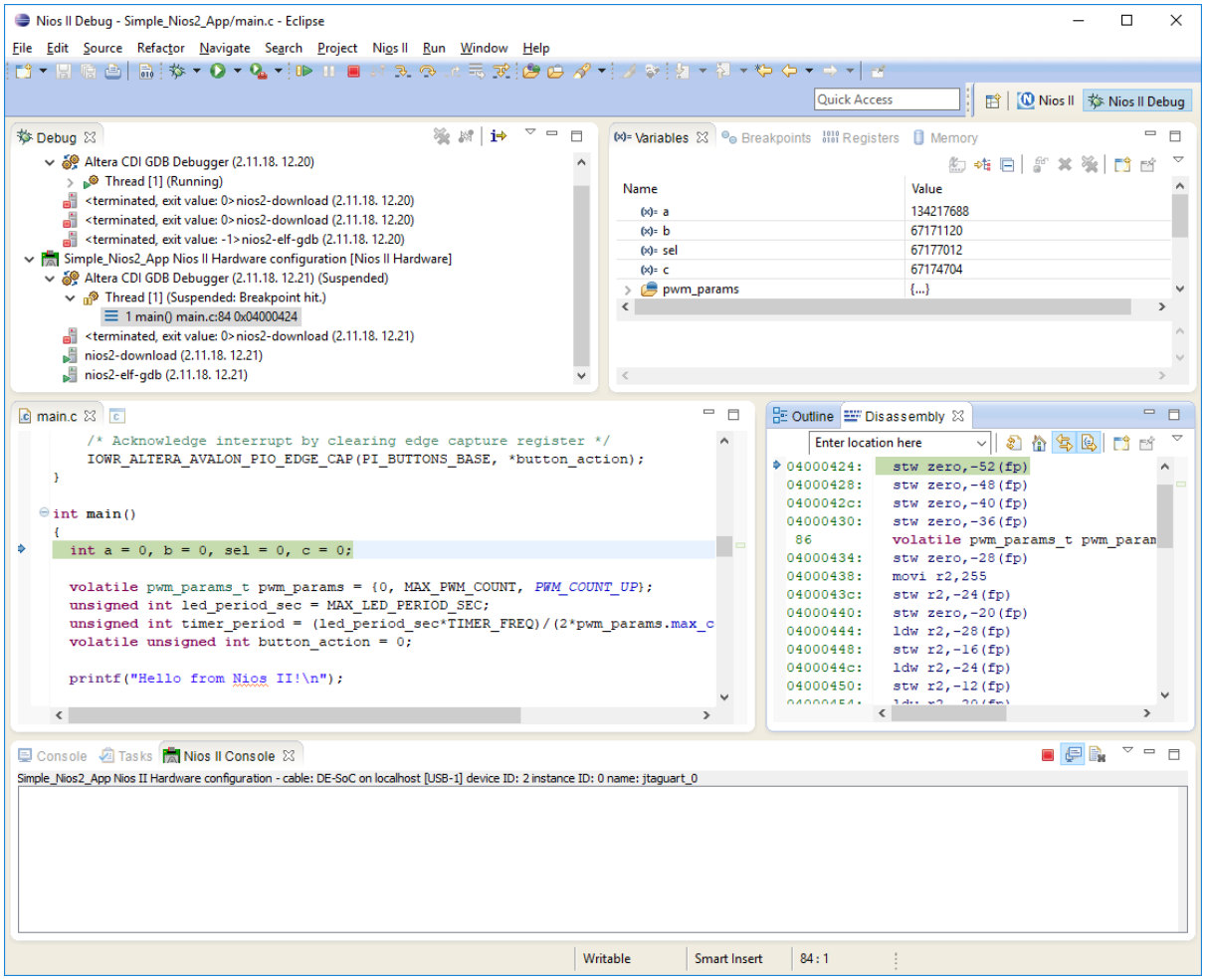
U slučaju da se sistem ne sastoji samo od hardverskog dela već ima delove procesiranja koji su realizovani u softveru potrebno je obezbediti efikasno testiranje i softverskog dela sistema. *Nios II Software Build Tools* pruža standardne mogućnosti testiranja sistema kao što su postavljanje *breakpoint*-a, posmatranje vrednosti određenih promenljivih i registara kao i posmatranje sadržaja memorije.

Da bi se započelo debagovanje softvera potrebno je najpre iskompajlirati ceo dizajn i isprogramirati FPGA spuštanjem SOF fajla. Potom je potrebno iskompajlirati softverski deo sistema. Nakon toga postupak je sličan kao kod standardnog programiranja procesora, selektuje se ELF fajl i klikne se desni taster miša s tim što se umesto komande *Run As→Nios II Hardware* koristi komanda *Debug As→Nios II Hardware* kao što je prikazano na Slici 29.



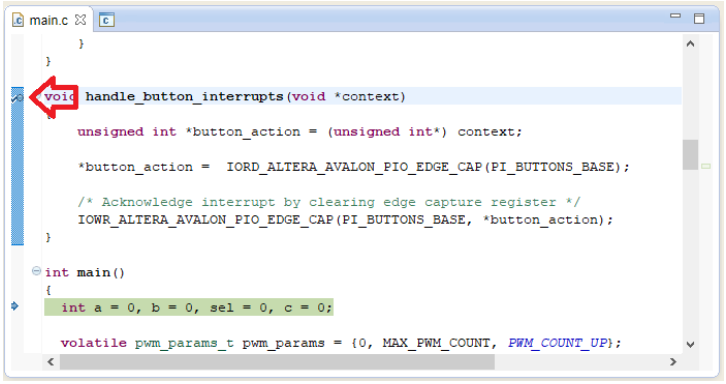
Slika 29. Pokretanje procesa debagovanja softverskog dela sistema

Pojavljuje se dijalog sa obaveštenjem da će alata preći u pogled za debagovanje na koji treba odgovoriti sa *Yes*. *Eclipse* omogućava prilagođavanje radnog okruženja različitim potrebama korišćenjem drugačijeg rasporeda i vrste prozora (pogleda). Nakon toga se startuje program pri čemu se izvršavanje pauzira na prvoj instrukciji *main* funkcije. Kroz program se kreće standardnim instrukcijama za debagovanje ***Step Into*** (F5), ***Step Over*** (F6), ***Resume*** (F8). Okruženje je moguće potpuno prilagoditi svojim potrebama uvođenjem odgovarajućih prozora. Dodavanje prozora se obavlja jednostavno biranjem komande *Window→Show View→...* Korisni prozori su: ***Variables***, ***Memory***, ***Registers***, ***Breakpoints***, ***Dissasembly*** i njih bi trebalo uključiti u okruženje za debagovanje. Primer okruženja za debagovanje prikazan je na Slici 30.



Slika 30. Okruženje za debagovanje u okviru *Nios II Software Built Tools*-a

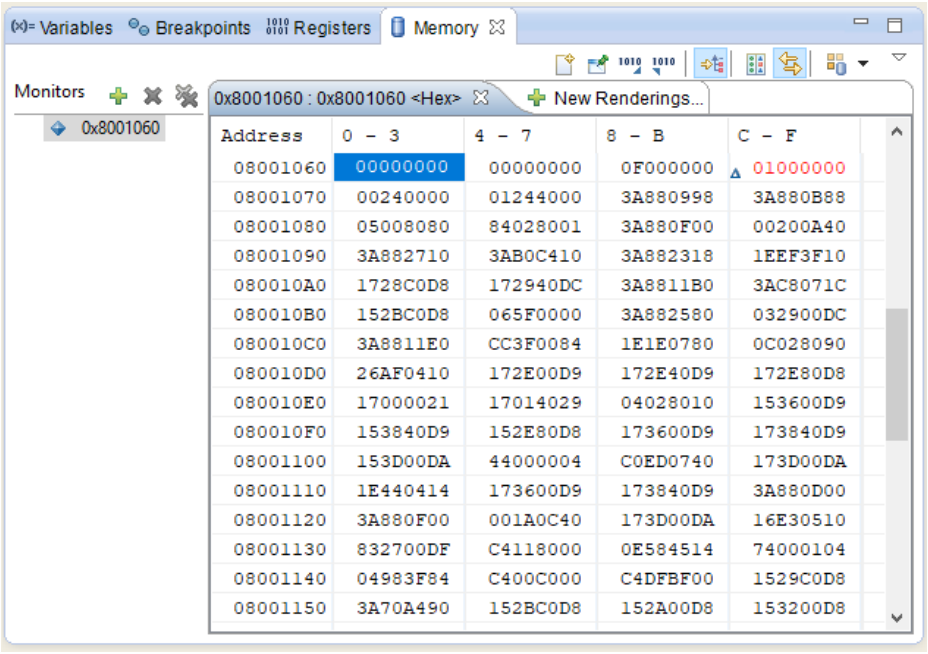
Breakpoint se može jednostavno postaviti dvostrukim klikom na marginu pored odgovarajuće instrukcije (na Slici 30 označeno plavom bojom). Za potrebe testiranja potrebno je postaviti *breakpoint* prekidnu rutinu ***handle_button_interrupts*** kao što je prikazano na Slici 31.



Slika 31. Postavljanje *breakpoint*-a na prekidnu rutinu

Nakon postavljanja *breakpoint*-a izvršavanje se može nastaviti pomoću komande *Resume* (F8). Na ovaj način se program izvršava kontinualno sve do pojave *breakpoint*-a. Kako je *breakpoint* postavljen na prekidnu rutinu aktiviranu pritiskom na neki od tastera regularan rad procesora neće biti prekinut sve dok se ne pritisne neki od 4 korišćena tastera. Po pritisku nekog od tastera procesor ulazi u prekidnu rutinu i pauzira izvršavanje na selektovanoj instrukciji.

Kako su sve periferije memorijski mapirane, vrednosti njihovih registara (koji su dostupni procesoru preko Avalon magistrale) se mogu očitati sa odgovarajuće memorijske adrese. Adresa na kojoj je mapirana *pi_buttons* periferija se može naći u okviru *system.h* fajla pod imenom ***PI_BUTTONS_BASE*** i u ovom dizajnu ima vrednost 0x8001060. Kako bi se posmatrao sadržaj registara ***pi_buttons*** periferije potrebno je selektovati prozor *Memory* i klinuti na zeleni znak plus. Zatim je potrebno uneti odgovarajuću memorijsku adresu, u našem slučaju 0x8001060 i kliknuti OK. Nakon toga se otvara prozor kao na Slici 32.



Slika 32. Posmatranje sadržaja registara *pi_buttons* periferije

Kako bi se rastumačio sadržaj memorije potrebno je poznavati memorijsku mapu posmatrane periferije. Registarska mapa PIO periferije prikazana je na Slici 33.

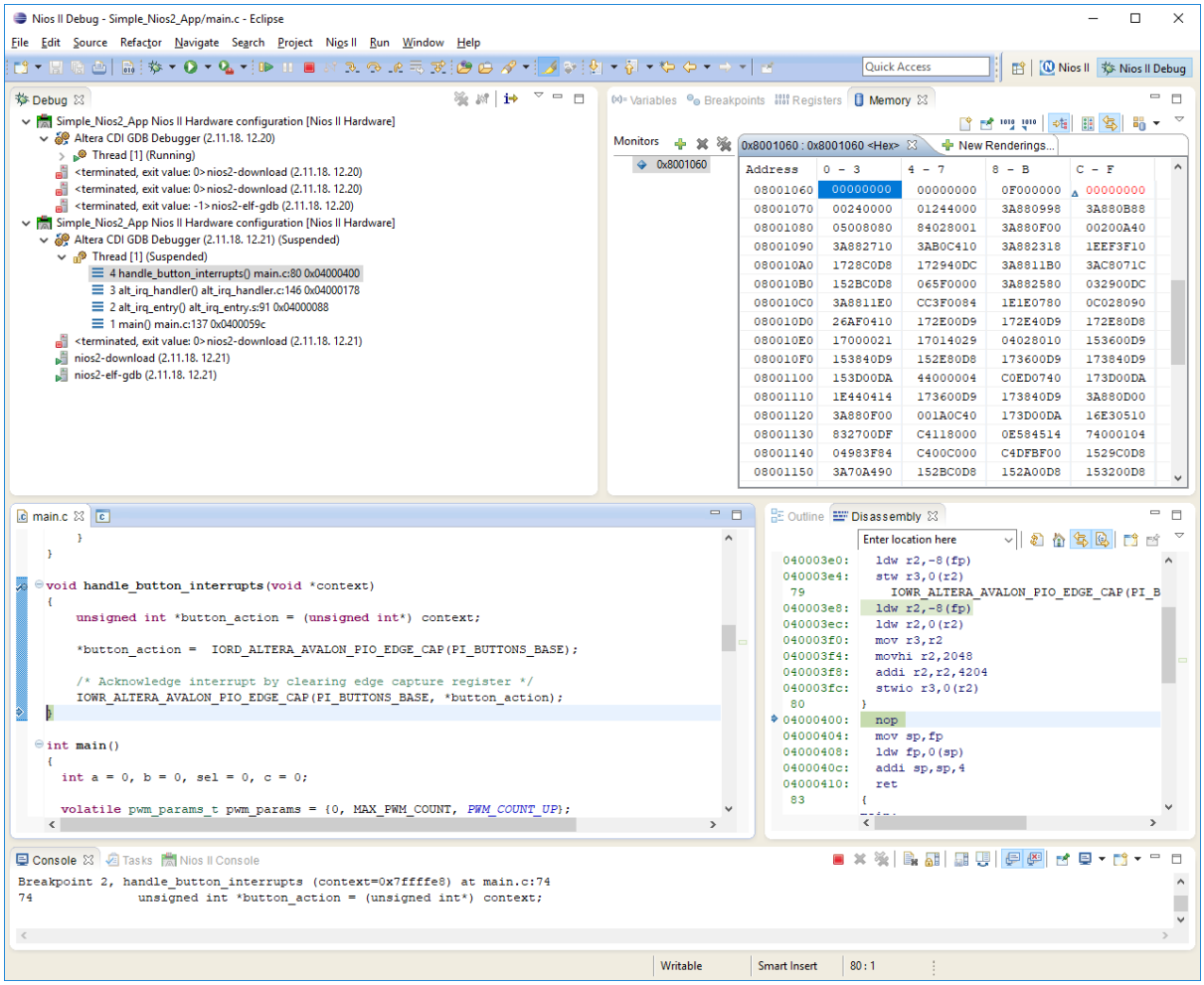
Table 11-2: Register Map for the PIO Core

Offset	Register Name		R/W	(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. Outset value is not stored into a physical register in the IP core. Hence it's value is not reserve for future use.				
5	outclear		W	Specifies which output bit to clear. Outclear value is not stored into a physical register in the IP core. Hence it's value is not reserve for future use.				

Slika 33. Registarska mapa PIO modula

Na osnovu registarske mape može se zaključiti da su 3. i 4. registar PIO periferije ***interruptmask*** i ***edgecapture***. Sa slike 32. možemo očitati njihov sadržaj koji iznosi 0x0F000000 i 0x01000000. Bitno je napomenuti da je *Nios II* procesor *little-endian* odnosno da se niži bajtovi smeštaju na niže memorijske adrese. Imajuću to u vidu sadržaj očitanih registara je sledeći

***interruptmask* = 0x0000000F** i ***edgecapture* = 0x00000001**. Odatle se zaključuje da su prekidi odobreni na sva 4 tastera i da je uhvaćena ivica na tasteru koji je vezan za liniju 0 *pi_buttons* periferije. Izvršavanjem instrukcije za potvrdu prijema prekida briše se sačuvana vrednost iz *edgecapture* registra. Svaka promena u okviru posmatranih memorijskih lokacija koja je nastala kao posledica poslednje izvršene instrukcije označava se crvenom bojom kao što je to prikazano na Slici 34.



Slika 34. Brisanje sadržaja *edgecapture* registra

Prikazani alati za testiranje pružaju jako velike mogućnosti za posmatranje rada dizajna iz svih uglova međutim najveća snaga je u mogućnosti njihovog međusobnog kombinovanja. Na primer *In-System Sources and Probes* se može koristiti za forsiranje određenog triger uslova dok se *SignalTap II* koristi za posmatranje vrednosti signala. Takođe se u okviru *Nios II Software Build Tools*-a mogu pročitati adrese na koje su skladištene odgovarajuće instrukcije od značaja i potom to iskoristiti kao triger uslov u okviru *SignalTap II* alata.

U svrhu demonstracije kombinovanja ovih alata postavimo sebi jedno pitanje: **„Koliko je taktova potrebno procesoru da započne izvršavanje prekidne rutine *handle_button_interrupts*?“**. Da bismo odgovorili na ovo pitanje moramo da ispratimo tok informacija od pritiska na taster do početka izvršavanja prekidne rutine. Dakle pritisak na taster se najpre registruje u *debounce_inv* perfieriji koja zatim tu informaciju prosleđuje *pi_buttons* modulu koji je povezan na magistralu procesora. Nakon detekcije ivice ulaznog signala *pi_buttons* generiše zahtev za prekid procesoru i ulazi se u proceduru pripreme prekidne rutine i na kraju izvršavanje prve instrukcije prekidne rutine. Na osnovu ovoga zaključujemo da je potrebno izmeriti vreme od aktiviranja signala *pressed* u okviru

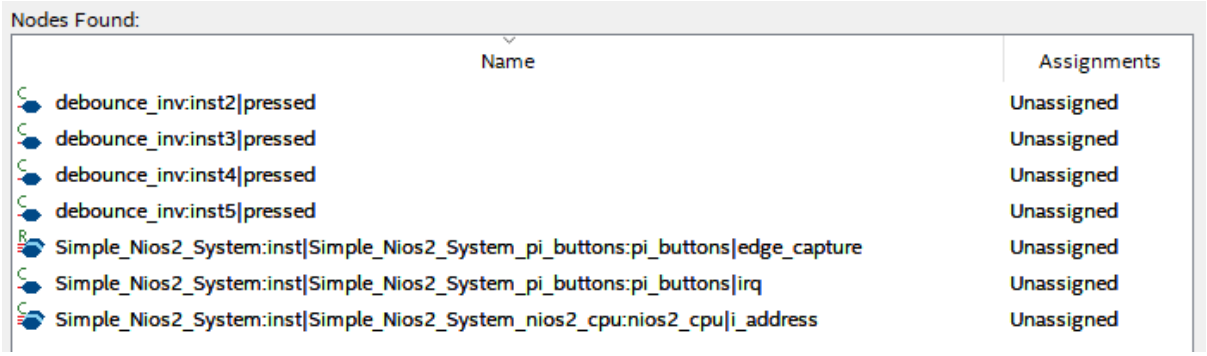
debounce_inv periferije do početka izvršavanja prve instrukcije prekidne rutine. Najpogodniji alat za ovo je *SignalTap II*.

Dakle potrebno je otvoriti novu instancu *SignalTap II* (staru instancu ako je ostala izbaciti iz dizajna). Ranije opisanim postupkom dodati *pressed* signale za sva 4 tastera.

Registri periferije *pi_buttons* se nalaze u okviru procesorskog sistema odnosno u okviru instance ***Simple_Nios2_System:inst→Simple_Nios2_System_pi_buttons:pi_buttons***. Potrebno je posmatrati registar ***edgecapture*** i liniju prekida ***irq***.

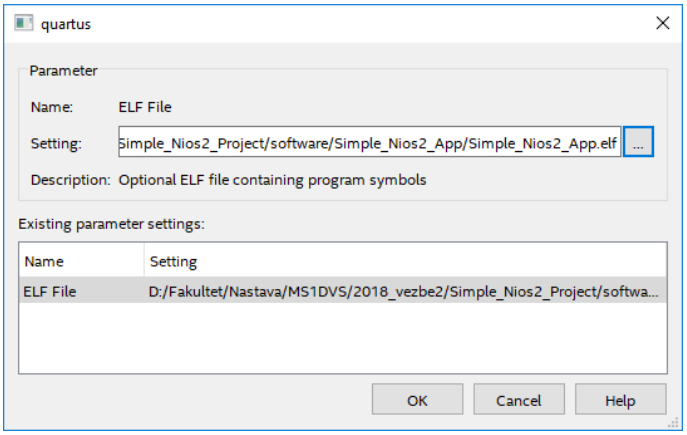
Početak ulaska u prekidnu rutinu se može označiti kao trenutak kada se na linijama instrukcijske adrese procesora nađe adresa prekidne rutine. Zbog toga je u posmatranje potrebno uključiti i adresnu magistralu za instrukcije procesora. Jezgro procesora se nalazi u okviru komponente ***Simple_Nios2_System:inst→ Simple_Nios2_System_nios2_cpu:nios2_cpu***. Potrebno je posmatrati registar ***i_address***.

Nakon dodavanja svih pomenutih signala prozor *Node Finder*-a bi trebao da izgleda kao na Slici 35.



Slika 35. Izgled *Node Finder* prozora nakon dodavanja signala od interesa

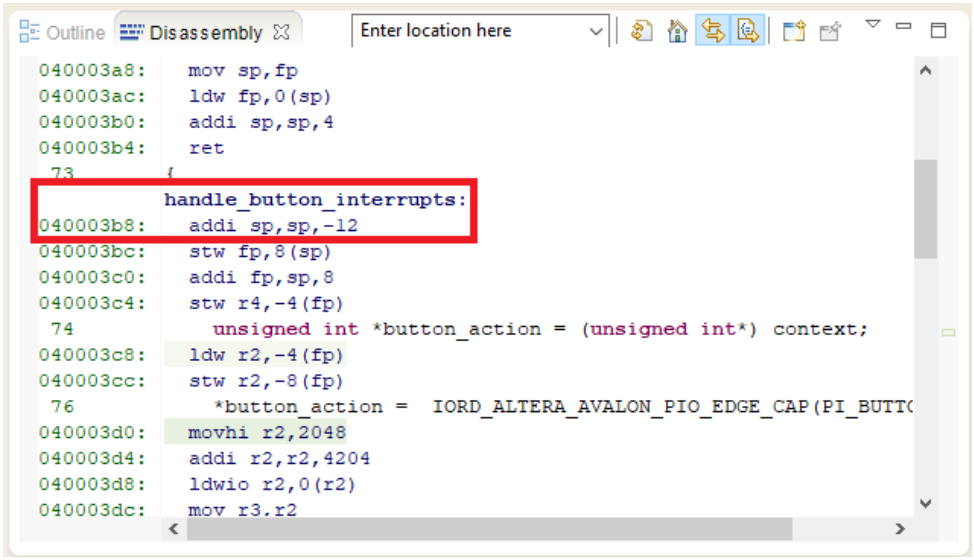
U okviru *SignalTap II* alata postoji mogućnost dodavanja posebne grupe signala pomoću *Nios II plug-in* u okviru koje se prikazuju simbolička imena instrukcija procesora koje se trenutno izvršavaju. Dodavanje ove grupe signala se obavlja desnim klikom na prazan prostor u okviru *Setup* prozora i biranjem opcije *Add Nodes with Plug-In→Nios II*. Za ELF fajl podesiti kompajlirani fajl trenutnog softverskog dela sistema koji se nalazi u direktorijumu aplikacije kao što je prikazano na Slici 36.



Slika 36. Podešavanje putanje do trenutno aktivnog ELF fajla

Takt sa kojim se radi odabiranje treba da bude jednak taktu na kom radi procesorski deo sistema. U ovom primeru to je takt od 50 MHz koji se generiše na izlazu PLL-a. Zbog toga je za takt unutar *SignalTap II* alata potrebno odabrati izlaz **out_clk0** pll modula koji se nalazi u okviru procesorkog sistema.

Potrebno je podesiti da se triger generiše onog trenutka kada se na adresnoj magistrali za instrukcije procesora pojavi adresa prekidne rutine *handle_button_interrupts*. Adresa ove prekidne rutine se može pročitati u okviru *Disassembly* prozora *Nios II Software Build Tools*-a kao što je prikazano na Slici 37.

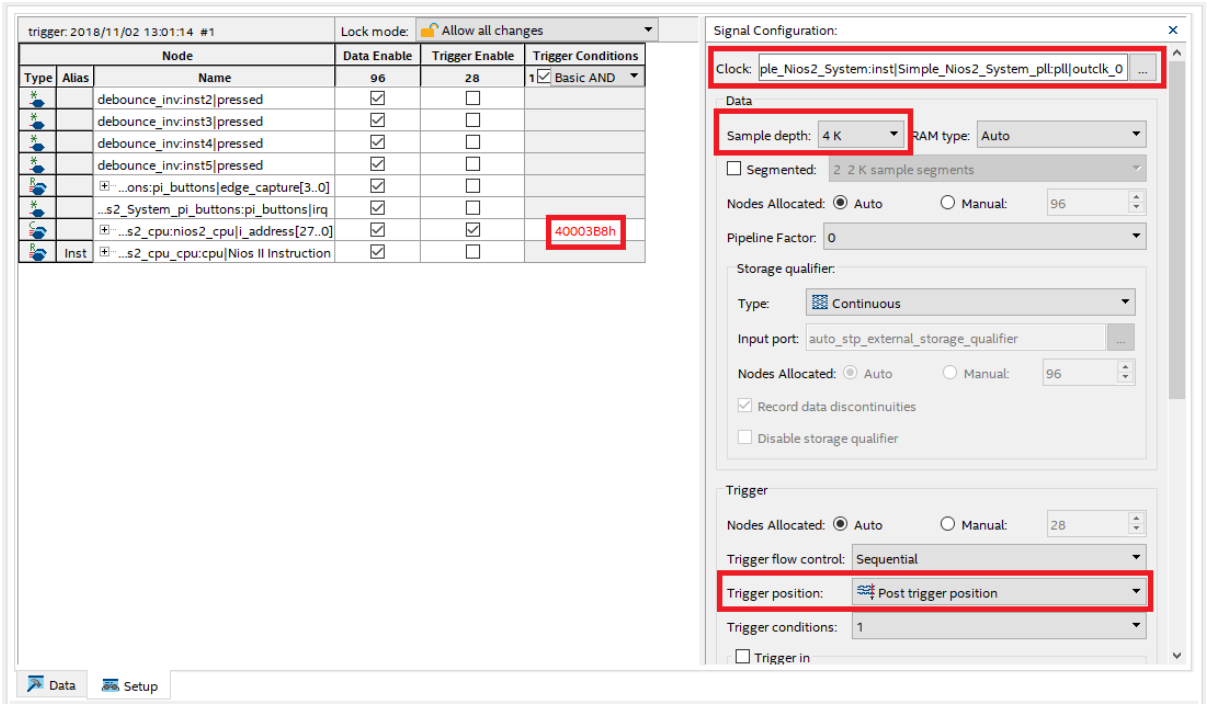


Slika 37. Određivanje adrese prve instrukcije prekidne rutine *handle_button_interrupts*

Dakle triger će se aktivirati kada se na adresnoj magistrali za instrukcije pojavi podatak 0x40003b8. Pošto nas interesuje interval od pojave pritiska na taster do ulaska u prekidnu rutinu, to je potrebno prikazati odbirke **PRE** pojave triger signala odnosno potrebno je postaviti *Post trigger position*.

Veličinu bafera podesiti na 4K odbirka, pošto veći bafer neće biti moguće sintetisati.

Nakon svih podešavanja izgled *SignalTap II* prozora bi trebao da izgleda kao na Slici 38.



Slika 38. Izgled podešenog *SignalTap II* prozora

Dalje je ceo dizajn potrebno kompajlirati i spustiti na ploču i pokrenuti analizu korišćenjem *SignalTap II* alata. Nakon toga je potrebno kompajlirati softverski deo projekta i isprogramirati *Nios II* procesor. Po pritisku na neki od taster aktivira se triger i dobijaju se dijagrami prikazani na Slici 39.

Slika 39. Vremenski dijagrami posmatranih signala

Sa Slike 39. se mogu uočiti 2 bitne stvari:

1) Triger je dobro podešen na adresu prekidne rutine.
2) Potrebno je 1860 taktova da procesor uđe u prekidnu rutinu.

Za vežbu odgovoriti na isto pitanje u slučaju da se koristi Nios II/f procesorsko jezgro.