Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621.38
DOI: https://doi.org/10.24867/33BE03Stamenkovic

FORMALNA VERIFIKACIJA DVOJEZGARNOG JEDNO-CIKLUSNOG RISC-V PROCESORA I DELA MEMORIJSKOG PODSISTEMA

FORMAL VERIFICATION OF A DUAL CORE SINGLE-CYCLE RISC-V CORE WITH PART OF MEMORY SUBSYSTEM

Petar Stamenković, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu je prikazan način verifikacije dvojezgarnog jedno-ciklusnog RISC-V procesora i dela memorijskog podsistema pomoću formalnih metoda i JasperGold alata kreiranog od strane kompanije Cadence. Dat je osnovni opis dizajna koji se verifikuje, osnovni operatori SystemVerilog jezika koji se koristi kao i verifikacione tehnike koje su upotrebljene za efikasniju verifikaciju sistema.*

Ključne reči: *Formalna verifikacija, RISC-V, Memorijski podsistem, Pokrivenost dizajna, Keš memorija*

Abstract – *This paper presents the way that dual core single-cycle RISC-V processor with part of memory subsystem is verified with formal methods by using a formal tool JasperGold developed by Cadence. Basic description of design is given, alongside with basic SystemVerilog operators and verification tecniques that were used for efficent verification of the system.*

Keywords: *Formal verification, RISC-V, Memory subsystem, Design coverage, Cache memory*

1. UVOD

Formalna verifikacija podrazumeva upotrebu alata koji matematički analiziraju dostižna stanja u dizajnu umesto da proračunavaju vrednosti za konkretno zadate ulazne vektore. Danas predstavlja veoma efikasan i temeljit način verifikacije jer teoretski može da potvrdi totalno odsustvo greški, uz idealno napisane osobine. Alat pruža ogroman broj mogućnosti i veliki broj aplikacija koje olakšavaju proveru određenih aspekata sistema kao što su provera pokrivesnoti (*Coverage app*), provera validnosti napisanih osobina (*FPV*), provera ekvivalentsnoti sistema (*SEQ*) i tako dalje. Formalna verifikacija sa sobom nosi neke veoma važne prednosti u odnosu na verifikaciju baziranu na simulaciji i neke od njih su:

1. Za jednostavne *RTL* modele nema potrebe za pisanjem test okruženja.
2. Lako ispitivanje nama nepoznatog sistema.
3. Efikasnije i uglavnom kraće vreme proveravanja.
4. Omogućena je potpuna pokrivenost
5. Dostupnost kontra-primera
6. Dostupnost analize beskonačnih putanja

U radu je najviše korišćena *FPV* aplikacija koja radi sa osobinama. Osobina (*property*) je jezička konstrukcija koja formalno opisuje neki aspekt digitalnog sistema i predstavlja uopštenje već poznate *assert* naredbe. Definišu se pomoću *HDL* jezika od kojih je u ovom radu korišćen *SystemVerilog*. Na njih se primenjuju verifikacione akcije *assert, assume* ili *cover*.

2. KRATAK OPIS PROJEKTOVANOG DIZAJNA

Kao što je pomenuto, projektovani dizajn je RISC-V sistem i on se sastoji od sledećih komponenti :

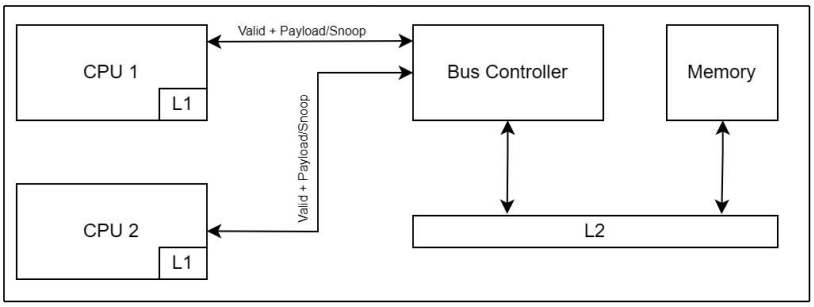
- Dva RISC-V jedno-ciklusna procesora sa sopstvenim L1 keš memorijama.
- Globalna magistrala sa kontrolerom.
- Deljena L2 keš memorija sa keš kontrolerom.
- Globalna memorija za podatke.

Blok šema sistema je data na slici 1 na sledećoj stranici. Projektovanje sistema je rađeno u 3 faze, gde je prva faza podrazumevala procesor koji sadrži i keš memoriju i *data memory* komponentu za podatke. U nastavku je ustanovljeno da to nije optimalno, pa je u fazi 2 *data memory* komponenta izbačena van procesora. Konačno faza 3 sadrži ceo sistem i njegovo povezivanje. Svaku fazu je pratila jednostavna verifikacija i pisanje kratkih test scenarija za simulaciju u okviru alata *Vivado*.

2.1. RISC-V JEDNO-CIKLUSNI PROCESOR

Srce ovog sistema predstavlja upravo pomenuti RISC-V procesor. Inicijalno, procesor je sadržao osnovne komponente RISC-V arhitekture a zatim je po fazama dodato ostalo. Procesor podržava instrukcije tipova R, I ,L, S, U, B i J. Sadrži sopstvenu L1 memoju koja je direktno mapirana i ima 256 lokacija i uz nju je i keš kontroler za istu.

NAPOMENA: Ovaj rad proistekao je iz master rada čiji mentor je bio dr Vuk Vranjković, vanr. prof.



Slika 1 **:** *Blok šema projektovanog sistema*

Protkol koherenntnosti koji je odabran za ovaj rad je MESI i on je takođe implementiran u okviru L1 keš memorije. Suštinski, ako se desi pogodak (*cache hit*) u L1 keš memoriji procesora koji traži podatak, on se jednostavno upisuje u registar. Ako se desi promašaj (*cache miss*), prvo proveravamo dostupnost podatka u drugom procesoru, zatim u L2 keš memoriji i na samom kraju u globalnoj memoriji za podatke koja predstavlja poslednji nivo memorijske hijerarhije. Procesor se sastoji od sledećih komponenti: aritmetičko-logička jedinica (*ALU*), instrukciona memorija, sabirač (*sa 4 i sa poljem imm*), modul za grananje (*branch condition*), kontroler samog procesora (*kontorlni signali*), modul za generisanje konstante (*immediate generator*), programski brojač(*PC*), multiplekseri, registarski fajl, modul za prosleđivanje podatka koji se upisuje u registarski fajl (*write back*) i L1 keš memorija sa svojim kontrolerom.

Bitno je napomenuti da se magistrali pristupa samo u slučaju L ili S tipa instrukcija i da u slučaju istovremenog pristupa oba procesora postoji implementriana arbitraža koja odlučuje koji će imati prednost u tom momentu. Inicijalno, prednost ima prvi procesor. Upis u sve memorije kao i registarski fajl se izvršava na opadajuću ivicu taktnog signala dok je čitanje iz svih asinhrono.

2.2. Ostatak sistema

Procesor je instanciran dva puta i oba su povezana na zajedničku magistalu koja za cilj ima efikasno rutiranje i prosleđivanje podataka, bilo to od strane drugog procesora ili L2 memorije. Magistrala dobija informacije kao što su adresa, operacija, tag kao i sam podatak i pomoću istih odlučuje gde se šta šalje. Takođe, poseduje indikatore o tome gde se podatak može naći, što delom predstavlja *snoop* protokol. Ukoliko jezgro 1 nema podatak, magistrala će proveriti da li ga ima drugo jezgro i tek u slučaju da ga nema će se osvrnuti na ostatak sistema.
Deljena L2 keš memorija je set-asocijativna sa dva kanala i ima 1024 lokacije koje su podeljene u 512 setova (*4 puta veći kapacitet od L1*). Algoritam zamene podataka unutar kanala koji se koristi je LRU i implementiran je pomoću LRU bita koji je sastavni deo svake linije u L2 memoriji. Od trenutka kada su oba podatka validna unutar jednog seta u memoriji, LRU biti tih kanala moraju uvek biti suprotni.
Globalna memorija za podatke predstavlja poslednji nivo memorijske hijerarhije i nakon odluke u fazi 2, izbačena je van procesora. Njoj pristupamo u slučaju da podatak nije dostupan niti u jednom od procesora niti u L2 keš memoriji. Tokom rada, njen kapacitet je bio 1024 kako bi se alatu olakšala provera osobina, međutim nakon iste, moguće je povećati njen kapacitet.

3. PROCES VERIFIKACIJE PROJEKTOVANOG SISTEMA

Slično procesu projektovanja, i proces verifikacije je rađen u 3 faze. Verifikacija procesora koji je imao i keš memoriju i *data memory* komponentu predstavalja verifikaciju faze 1 dok je verifikacija finalne verzije procesora bez *data memory* komponente predstavljena sa verifikacijom faze 2. Verifikacija celokupno povezanog sistema je verifikacija faze 3. Kao što je pomenuto, pre same formalne verifikacije pisani su jednostavni testovi za test okruženje u okviru alata *Vivado* gde je proverena osnovna funkcionalnost sistema za par smišljenih ulaznih kombinacija instrukcija.

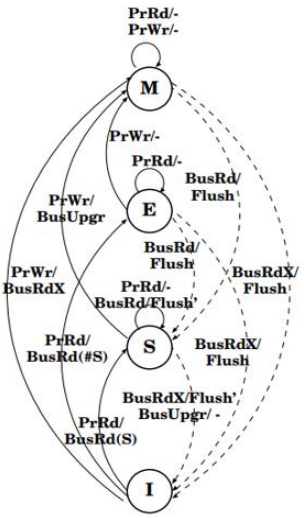
3.1. Verifikacioni plan

Verifikacioni plan predstavlja prvi korak u svakoj verifikaciji i predstavlja bitan dokument jer se upravo na njega ceo verifikacioni tim oslanja. Sastavlja se na osnovu funkcionalne specifikacije sistema i preporuka je pisati ga „hladne glave“ kako bi rasterećeni obuhvatili sve bitne tačke za proveru sistema.
Faza 1 i faza 2 su pokrivene istim verifikacionim planom obzirom da obe obuhvataju proveru procesora uz modifikaciju pomenute *data memory* komponente. Radi se provera svih komponenti unutar procesora kao i proveri tranzicija mašina stanja i MESI protkola koherentnosti.
Faza 3 je pokrivena drugim verifikacionim planom i on podrazumeva proveru interakcija između svih komponenti sistema kao što su dešavanje nakon *flush* operacije, provera oba kanala u jednom setu L2 memorije, LRU biti i slično. Takođe se proverava i upis i čitanje kako iz L2 keš memorije, tako i iz globalne memorije za podatke.

3.2. Verifikaciono okruženje

Za razvijanje verifikacionog okruženja je korišćena metoda referentog modela, to jest, napisana je *checker* komponenta. Za povezivanje ove komponente i samog dizajna se koristila *bind* komanda i napisana skripta. Slično kao i za verifikacioni plan, i u ovom slučaju su napisana dva referentna modela, jedan za verifikaciju procesora i jedan za verifikaciju celokupnog sistema. U ovim fajlovima su

pisane osobine koje za ulogu imaju proveru funkcionalnosti sistema. Oba referentna modela imaju sličnu strukturu i ona se sastoji od sledećih delova :



Slika 2 **:** *MESI protkol*[2]

- Deklaracija portova dizajna procesora odnosno sistema
- Sekcija za definisanje pomoćnih signala, parametara i struktura.
- Sekcija za definisanje restrikcija i ograničenja (*assumes*)
- Sekcija za definisanje tzv. *grey box* signala koji idu kroz hijerarhiju i uzimaju se direktno iz dizajna.
- Pomoćni kod (*Auxillary code*)
- Sekcija za definisanje tvrdnji i pomoćnih tačaka pokrivenosti (*asserts, covers*)

Restrikcije su veoma bitan deo referentnog modela jer ograničavaju vrednosti ulaza koje alat može da kreira. Osobina koju alat dokaže, uz restrikcije koje nisu dobro napisane, ne može se smatrati validnom jer smo time možda maskirali grešku. Na primer, RISC-V arhitektura ne dozvoljava da u registru x0 bude bilo koja vrednost sem 0. Ukoliko ne napišemo restrikciju koja osigurava da alat ne kreira takvu instrukciju koja će da radi upis u taj registar, naša osobina za upis u registarski fajl neće biti korektna. Još jedan primer su vrednosti maske za S ili L tip instrukcije. Maska u ovim slučajevima određuje koji deo reči od 32 bita će biti upisan ili učitan, byte (*8 bita*), half word (*16 bita*) ili word (*32 bita*). Po RISC-V standardu, polje za masku je široko 3 bita pa je alatu dostupno 8 vrednosti za istu, iako za S postoje samo 3 vrednosti koje se koriste i za L samo 5 vrednosti. Bez ograničenja, alat bi odmah dao kontra-primer sa jednom od vrednosti koje se ne koriste, što ne bi bilo korektno. Bitno je napomenuti da, obzirom da sistem radi i na rastuću i na opadajuću ivicu takta, svaka restrikcija mora imati svoju kopiju koja se evaluira i na opadajuću ivicu.

3.3. Tehnike formalne verifikacije

Kako bi se alatu olakšao rad i dokaz napisanih osobina, korišćene su razne tehnike. One su pomogle i pri tome da se kod organizuje i time inžinjeru olakša da razume povratnu informaciju alata.

Metoda crne kutije (*black box*) – Unutar instrukcione memorije su učitane instrukcije koje sistem treba da izvrši. Kako bi verifikacija bila preciznija, na ovu komponentu je primenjena *black box* tehnika. Alat sada ne posmatra unutrašnjost ove komponente i njenu funkcionalnost, već ima slobodu da na njene ulaze dovodi sve moguće vrednosti koje poštuju zadate restrikcije. Obzirom da je ova komponenta jednostavna i ne izvršava neki napredan kod, nije postojalo nikakvih mera opreza za primenu ove tehnike. Nakon primene iste, trebalo je samo napisati propratne restrikcije koje će da osiguraju korektan rad sistema. Preciznije, donjih 7 bita izlazne instrukcije je moralo da ima jednu od 7 vrednosti za svaki od podržanih tipova instrukcije.

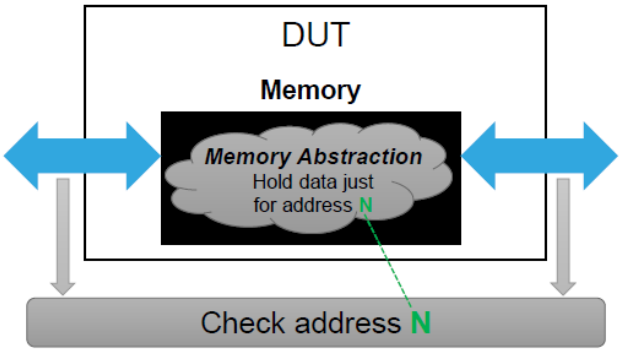
Pisanje pomoćnog AUX koda – Kod kompleksnijih dizajnova kao što je ovaj, osobine mogu biti takođe veoma kompleksne i dugačke, što nikako ne odgovara alatu. Pomoćni kod omogućava da se kreiraju neki pomoćni signali koje kasnije možemo da upotrebimo za poređenje u okviru nekog dokaza. Na primer, za proveru programskog brojača bi na prvi pogled imali 7 provera za svaki tip instrukcije. Međutim ukoliko samo napišemo jednostavni pomoćni kod i kreiramo signal *pc_ref* koji prati promenu vrednosti brojača u referentnom modelu, kao što je urađeno u ovom projektu, dovoljna je samo jedna osobina koja dokazuje validan rad programskog brojača. U nekom drugom slučaju je bolje jednu osobinu rastaviti na više manjih osobina. Recimo da želimo da proverimo skladištenje podataka u L1 keš memoriju. Postoje 3 vrednosti maske za S tip instrukcije tj. možemo skladištiti 1, 2 ili 4 bajta. Teoretski je moguće napisati jednu osobinu koja će proveriti sve maske odjednom, ali obzirom da je skladištenje podataka kompleksna operacija za alat (*jer radi sa velikim brojem stanja*), mnogo je pametnije i efikasnije to uraditi sa 3 odvojene osobine.

Upotreba nedeterminističke konstante (*location based coupling*) – Nedeterministička konstanta ili još poznato kao i slobodna varijabla je varijabla koju alat zadaje tokom provere. Ova tehnika podrazumeva proveru jedne nedeterminističke lokacije i često se koristi upravo za adrese i tagove kod keš memorija. Na ovaj način se apstrahuju svi elementi osim lokacije koja se proverava, čime se drastično smanjuje prostor koji alat mora da ispita. Ovo se često koristi u slučajevima:

- Kada sistem ima dosta simetričnih elemenata kao što su tabele, memorije ili nizovi.
- Kada sistem ima veliki adresni prostor, kao što je DMA kontroler ili memorija.

Uz ovu tehniku je potrebno napisati restrikciju koja drži

ovu varijablu stabilnom, jer ona ne sme da se menja tokom jedne iteracije rada alata. Takođe, ako je potrebno, moguće je ograničiti i vrednost ove varijable. Tehnika se uglavnom primenjuje u osobinama tako što se evaluacija iste radi pri podudaranju adrese koju je doveo alat i pomenute varijable. Grafički prikaz ove tehnike je dat na slici 3.



Slika 3 *: Location based coupling*[3]

Za većinu osobina je korišćeno pisanje pomoćnog AUX koda i za sve kompleksnije osobine (*rad sa memorijama*) je korišćena nedetermisnitčka konstanta. Međutim, postoje i osobine za koje ovo nije bilo potrebno. Ovo su uglavnom jednostavne osobine za koje je dovoljno koristiti dostupne portove i *grey-box* signale kako bi se dokaz napisao. Primer jedne takve osobine je dat ispod ovog pasusa.

```
property checking_transition_from_IDLE_to_DMEM_WRITE;
        top.cache_L2.state == 2'b00 &&
        top.cache_L2.cache_hit_out == 2'b01 |=>
        top.cache_L2.state == 2'b01;
endproperty
```

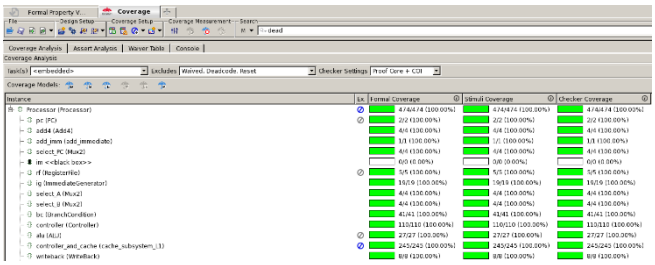
Napisana osobina jednostavno proverava da li se prelazi u stanje DMEM_WRITE u slučaju promašaja u memoriji.
Sve osobine koje su napisane su dokazane i sve pronađene greške u svim fazama su ispravljene i dokumentovane u samom radu.

4. ANALIZA REZULTATA

Nakon što su sve osobine napisane i dokazane, poslednji korak je upotreba *Coverage* aplikacije u okviru alata. Njena uloga je da nam pruži metriku i povratnu informaciju o tome koliko smo dizajna pokrili sa našim osobinama, da li su sva granjanja proverena i slično.
Formalni alat nudi 3 tipa pokrivenosti :

1. *Stimuli coverage* – Koji kod ili funkcionalnost je dostižna pomoćnu formalnog alata? Koje osobine alat može da pokrije?
2. *Checker coverage* – Ova pokrivenost nam daje informaciju o tome da li je verifikacija završena i koliko dizajna je pokriveno pomoću napisanih tvrdnji.
3. *Formal coverage* – Kombinacija rezultata prethodne dve metrike. Tačka u okviru ove pokrivenosti se smatra „pokrivenom“ ako je ista „pokrivena“ i u *Stimuli* i u *Checker* metrici. Ova metrika nam daje informaciju o ukupnoj pokrivenosti sistema.

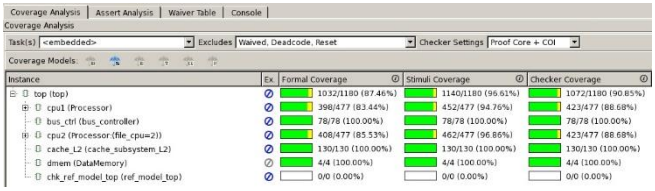
Aplikacija je pokrenuta nakon pisanja oba referetna modela, na kraju faze 2 i na kraju faze 3.
Nakon verifikacije procesora, aplikacija je pokazala maksimalnu pokrivenost (*slika 4*) nakon određenog vremena. Ovime smo dobili zeleno svetlo da nastavimo sa verifikacijom ostatka sistema.



Slika 4 : *Pokrivenost nakon verifikacije procesora*

Nakon uspešne verifikacije celokupnog sistema, aplikacija je pokrenuta i nakon faze 3.
Usled manjka memorijskih resursa na virtuelnoj mašini na kojoj je bilo rađeno, pri kraju rada aplikacije alat je izdao upozorenje zbog kojeg je provera morala biti obustavljena pred kraj. Ovime pokrivenost nije teoretski maksimalna, međutim na slici 5 možemo primetiti sledeće stvari:

- Novo dodate komponente su maksimalno pokrivene, dok je alat ostavio samo već proverene procesore za kraj.
- Većina funkcionalnosti kod procesora je dokazana, zbog čega se pretpostavlja da bi i ostatak bio dokazan sa više dostupnih resursa.
- Nema oblasti u koje alat ne može uopšte da uđe (*crvene oblasti*), već su preostale samo žute oblasti koje su „još uvek nedokazane“ (*undetermined*).



Slika 5 : *Pokrivenost nakon verifikacije sistema*

Uz sve ove tačke kao i osobine koje su dokazane, određeno je da se na ovom mestu verufikacija završi, a time i ovaj projekat.

5. ZAKLJUČAK

Nakon analize rezultata je ustanovljeno da sistem sadrži sve osobine koje su predviđene po specifikaciji. Procesori mogu da izvrše sve instrukcije koje podržavaju sa korektnim rezultatima. Za verifikaciju su uspešno iskorišćene formalne metode i tehnike za smanjenje kompleknosti. Sve pronađene greške su dokumentovane i ispravljene i analiza rezultata pomoću *Coverage* aplikacije je dala dobar rezultat čime zaključujemo da je verifikacija uspešno odrađena.

6. LITERATURA

[1] E. Seligman, T. Schubert, M. V. A. K. Kumar, *An essential toolkit for modern VLSI Design*
[2] T. Suh, *Integration and evaluation of cache coherence protocols for multiprocessor socs,* 2006
[3]"Jasper expert course", Cadence [Na mreži]. Available: https://www.cadence.com/en_US/home/training/all-courses.html

Kratka biografija

Petar Stamenković rođen je u Novom Sadu 2000. godine. Diplomirao je na Fakultetu tehničkih nauka, na smeru Energetika, elektronika i telekomunikacije 2023. godine. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Energetika, elektronika i telekomunikacije odbranio je 2025. godine. Kontakt: petarstamenkovic35@gmail.com