УНИВЕРЗИТЕТ
У НОВОМ САДУ

ФАКУЛТЕТ
ТЕХНИЧКИХ НАУКА

Трг Доситеја Обрадовића 6, 21000 Нови Сад, Југославија
Деканат: 021 350-413; 021 450-810; Централа: 021 350-122
Рачуноводство: 021 58-220; Студентска служба: 021 350-763
Телефакс: 021 58-133; e-mail: ftndean@uns.ns.ac.yu

PROJEKAT

iz predmeta:
Formalne metode projektovanja i verifikacije hardvera

TEMA PROJEKTA:

Formalna verifikacija procesora sa RV32I setom instrukcija

TEKST ZADATKA:

Korišćenjem aplikacija sadržanih u JasperGold alatu, formalno verifikovati RISCV procesor sa I (*integer*) setom instrukcija.

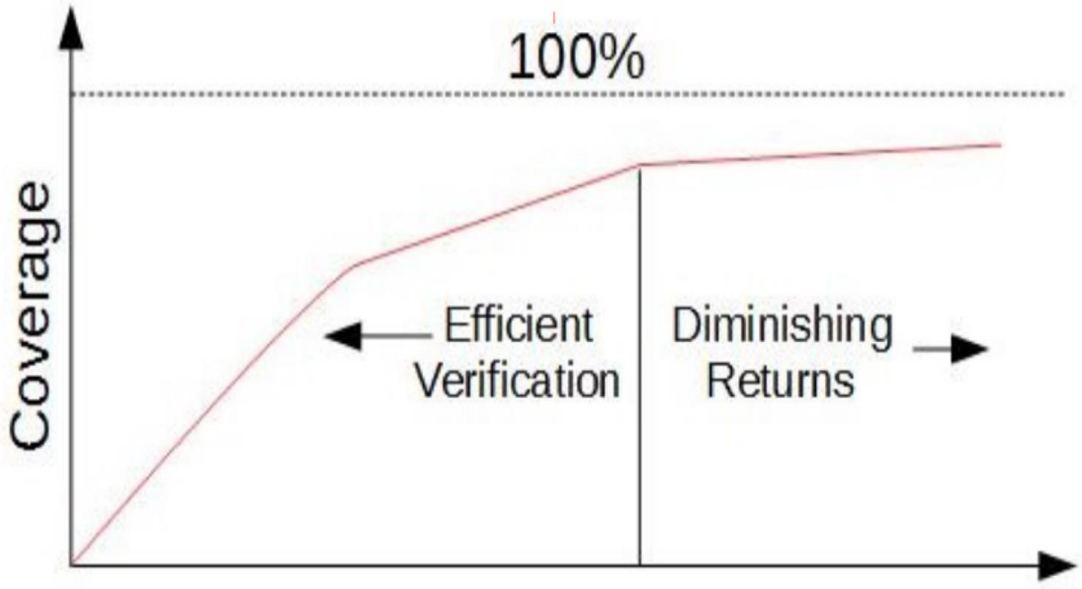
Mentor
Vuk Vranjković

Studenti:
Đorđe Mišeljić E1 5/2018
Nikola Kovačević E1 6/2018

U Novom Sadu, 17. septembra 2019 godine

1 Uvod

Dizajniranje ASIC čipova bez funkcionalnih grešaka (*eng. Bugs*) je bitan uslov za uspeh proizvoda. Kako sistemi postaju veći i kompleksniji, cilj 100% funkcionalne pokrivenosti koristeći tradicionalne metode postaje teško ostvariv. Broj usmerenih testova koji bi morao da bude napisan da bi se pokrilo celokupno ponašanje nekog SOC-a (*eng. System On Chip*) je ogroman, i zbog ovoga je praktično nemoguće simulirati moderne ASIC čipove na ovaj način. Da bi se prevazišli ovi problemi koriste se druge metode kao što je generisanje nasumičnih (*eng. radnom*) testova, kako bi se otkrili nepredviđeni problemi. Ali čak i sa drugim strategijama, praktično je nemoguće pronaći sve greške u sistemu oslanjajući se samo na simulaciju i u određenom trenutku dolazi se do tačke gde je dodatni napredak mali u odnosu na uloženo vreme (slika 1).



Slika 1. Napredak pokrivenosti u odnosu na uloženo vreme

Jedan od načina da se prevaziđe ovaj problem jeste korišćenje formalne verifikacije hardvera. Za razliku od simulacije, formalna verifikacija dokazuje ispravnost dizajna, i prilkom te verifikacije koriste se sledeći Algoritmi:

- Dokaz ispravnosti proverom ekvivalentnosti dizajna i određenje reference.
- Dokaz ispravnosti korišćenjem verifikacije zasnovane na dokazivanju svojstava (*eng. Property Driven Verification*).

Obe tehnike obećavaju veću efikasnosti, kompletniju pokrivenost i najbitnije od svega ne zahtevaju razvijanje kompleksnih testova. Iz prethodno navedenog slede sledeće prednosti:

- Nisu potrebne dugačke simulacije da bi se ostvarila dobra pokrivenost.
- Rezultati verifikacije se ne oslanjaju na kvalitet osmišljenih testova, već algoritmi dokazuju ispravnosti za sve moguće testove.

Da bi se izvršila formalna verifikacija dizajneri moraju da specificiraju svrhu dizajna uglavnom u formi svojstava (*eng. Properties*) ili tvrđenja (*eng. Assertions*). Svojstva i tvrđenja omogućavaju alatu da spozna kako dizajn treba da funkcioniše, i da na osnovu njih prijavi problem ukoliko se dizajn ne ponaša kako je opisano svojstvima i tvrđenjima.

2 Korišćene tehnologije

Formalna verifikacija sistema izvršena je pomoću JasperGold platforme za formalnu verifikaciju, odnosno aplikacija sadržanih u platformi. U nastavku je dat spisak svih aplikacija i naznaka da li je za specifičnu aplikaciju postojala licenca prilikom izrade projekta ili ne:

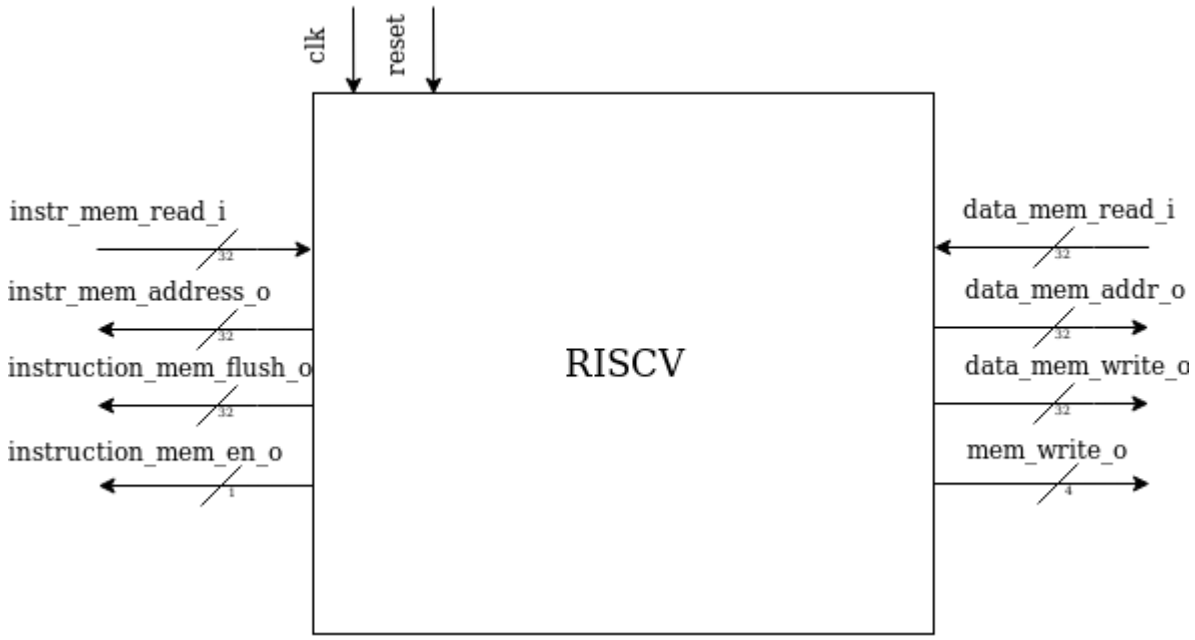
- Formal Property Verification App. Licenca: DA
- X-Propagation Verification App. Licenca: NE
- Connectivity Verification App. Licenca: NE
- Architectural Modeling App. Licenca: DA
- CSR Verification App. Licenca: NE
- Behavioral Property Synthesys App. Licenca: DA
- Superlint App. Licenca: DA
- Low Power Verification App. Licenca: NE
- Sequental Equivalence Checking App. Licenca: DA
- Security Path Verification App. Licenca: DA
- Coverage App. Licenca: DA
- Coverage Unreachability App. Licenca: NE
- Clock Domain Crossing Verification App. Licenca: DA
- Functional Safety Verification App. Licenca: NE
- RTL Development App. Licenca: DA
- Deadlock detection App. Licenca: NE
- Executable specification App. Licenca: DA

Kako bi se izvršila formalna verifikacija zasnovana na dokazivanju svojstava (*eng. Property driven Verification*) korišćena je *Formal Property Verification* aplikacija navedena na prethodnom spisku.

Svojstva (*eng. properties*) korišćena da se opiše sistem predstavljena su pomoću tvrđenja u System Verilog jeziku za opis hardvera (*eng. System Verilog Assertions*) i *Formal Property Verification* aplikacija je korišćena da bi se navedena svojstva validirala.

3 Formalna verifikacija RISCV procesora

Sistem koji je potrebno verifikovati je procesor sa RV32I setom instrukcija. Interfejs procesora je prikazan na sledećoj slici:

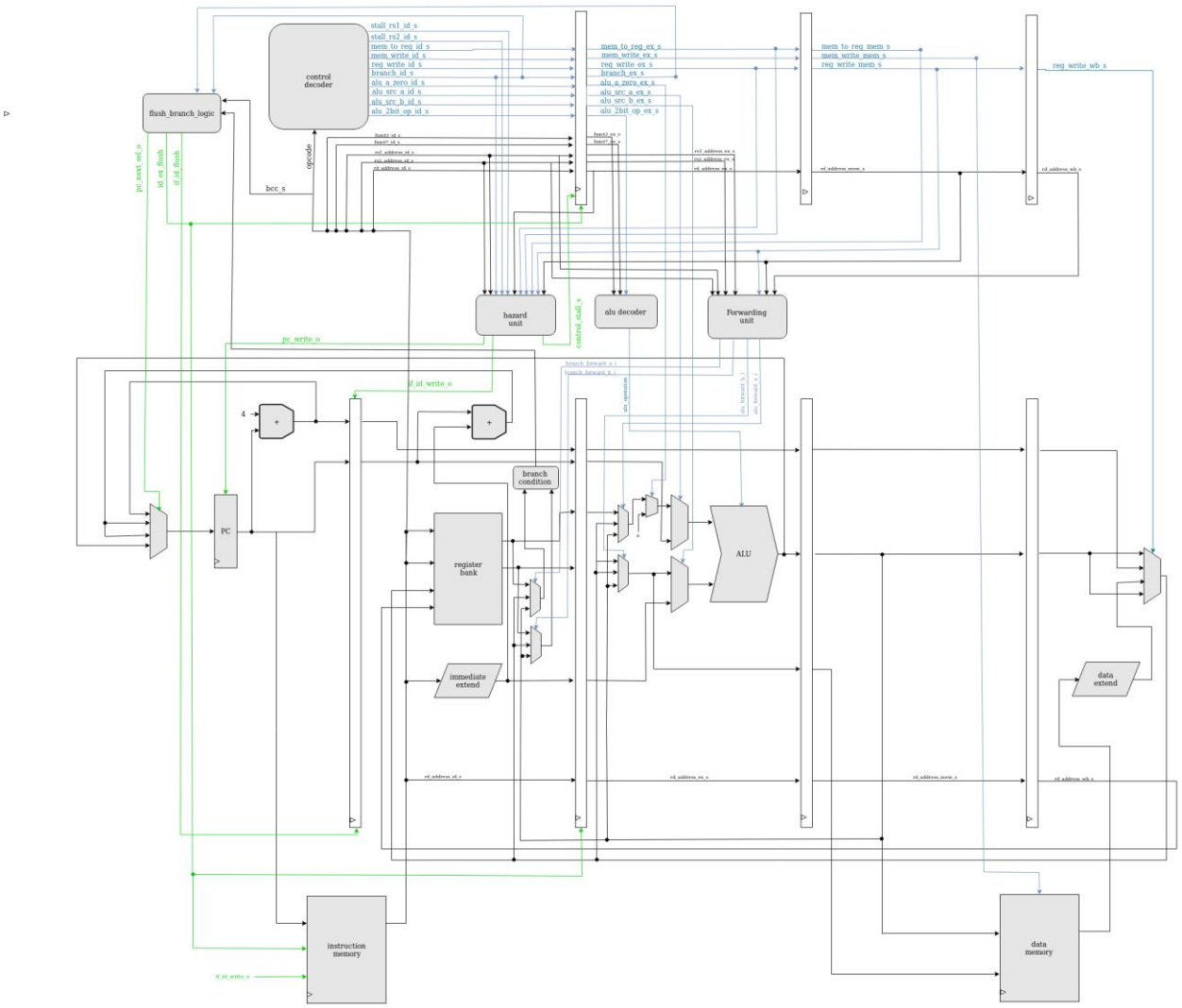


slika 2. Interfejs RISCV procesora

Kao što se može videti interfejs se sastoji od linija za komunikaciju sa memorijom za podatke (linije na desnoj strani) i memorijom za instrukcije (linije na levoj strani). Dve memorije su sinhrone BRAM memorije. Pre nego što se počne sa pisanjem svojstava koja opisuju procesor, neophodno je uvesti ograničenja formalnom alatu, kako bi on generisao vrednosti na interfesju procesora skladno tome kako je zamišljeno da funkcionišu BRAM memorije. Ograničenja su sledeća i mogu se naći u *assumptions.sv* izvornom fajlu:

- Na liniju *instr_mem_read_i* mogu da pristižu samo validne instrukcije.
- Ukoliko se *instruction_mem_en_o* izlaz spusti na nisku vrednost (logička nula), u narednom taktu se mora ponoviti prethodna instrukcija na instr_mem_read_i ulazu (desio se *stall*).
- Ukoliko se *instruction_mem_flush_o* izlaz podigne na visok nivo (logička jedinica), u narednom taktu se moraju pojaviti sve *nule na instr_mem_read_i* ulazu.

RISCV procesor je verifikovan kao *white box* odnosno nije posmatran samo njegov interfejs, već su posmatrane i promene na njegovim unutrašnjim signalima i blokovima. Na sledećoj slici je prikazan blok dijagram RISCV procesora:



slika 3. RISCV procesor blok dijagram

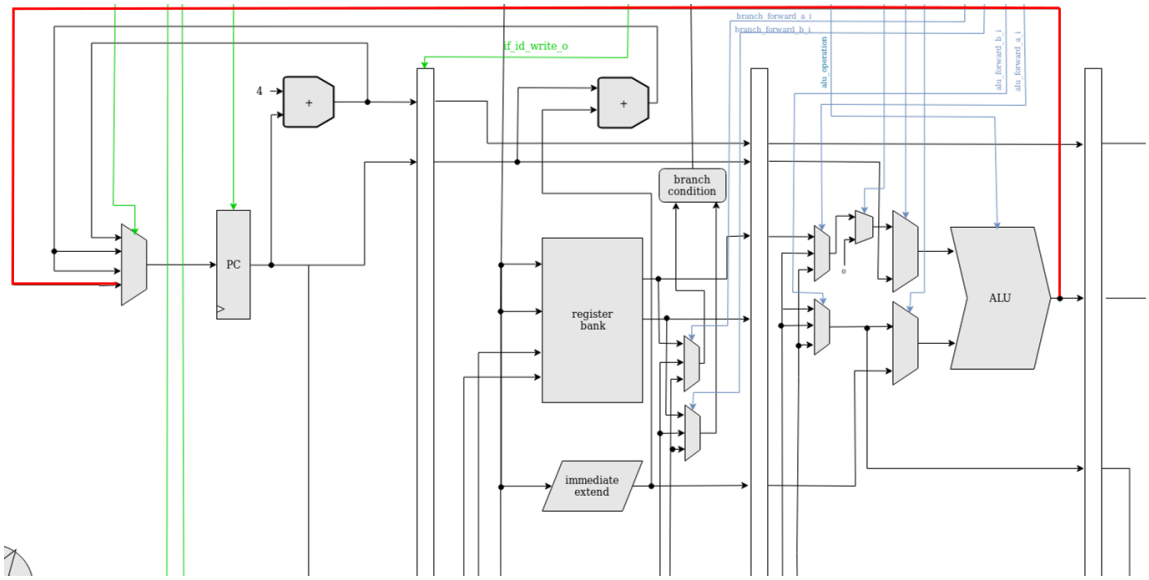
Sa slike se može videti da je u pitanju procesor sa protočnom obradom, te je prilikom formalne verifikacije pažnja bila usmerena na ispitivanje ispravnosti rada delova procesora koji su podložni grešci zbog postojanja protočne obrade. Svojstva (*eng. Properties*) koja su ispitana, podeljena su u grupe:

- Provera ispravnosti skokova, uslovnih i bezuslovnih. C*hecker* je instanciran u *data_path-u.*
- Provera ispravnosti prosleđivanja (*eng. Forwarding)* podataka. *Checker* je instanciran u *control_path-u.*
- Provera ispravnosti rada programskog brojača. C*hecker* je instanciran *u TOP_RISCV.*
- Provera ispravnosti generisanja *stall* signala. *Checker* je instanciran u *control_path-u*.

3.1 Provera ispravnosti skokova, uslovnih i bezuslovnih

Iz ukupnog seta instrukcija RV32I procesora postoje 3 tipa instrukcija koje izazivaju skokove: bezuslovni skok (JAL), relativni bezuslovni skok (JALR) i uslovni skok. Da bi se ispitalo da li procesor izvršava skok kada se pojavi jedna od instrukcija napisano je 5 svojstava. Izvorni fajl u kome su svojstva napisana zove se ***branch_checker.sv.***

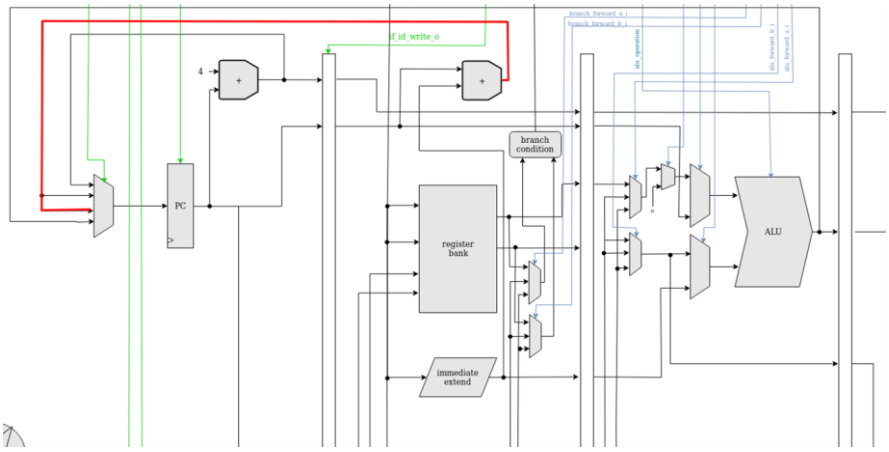
Prvo svojstvo testira da li JALR instrukcija izaziva skok na pravu lokaciju, odnosno da li programski brojač dobija pravu vrednost nakon što se skok izvrši. Na sledećoj slici nalazi se uvećan deo slike 3. na kome se vidi šta treba da se desi ukoliko se JALR instrukcija nađe u EX fazi protočne obrade:



Slika 4. JALR instrukcija u EX fazi

Crvenom linijom na prethodnoj slici naznačeno je da naredna vrednost programskog brojača treba da bude rezultat aritmetičko logičke jedinice. Ako se pogleda izvorni fajl tvrdnja koja pokriva ovo svojstvo napisana je u 34. liniji, i ona tvrdi da ukoliko se u EX (*eng. execute*) fazi protočne obrade nađe JALR instrukcija, i ukoliko se nisu desili *flush* id_ex registra i *stall* , naredna vrednost programskog brojača biće jednaka izlazu ALU jedinice.

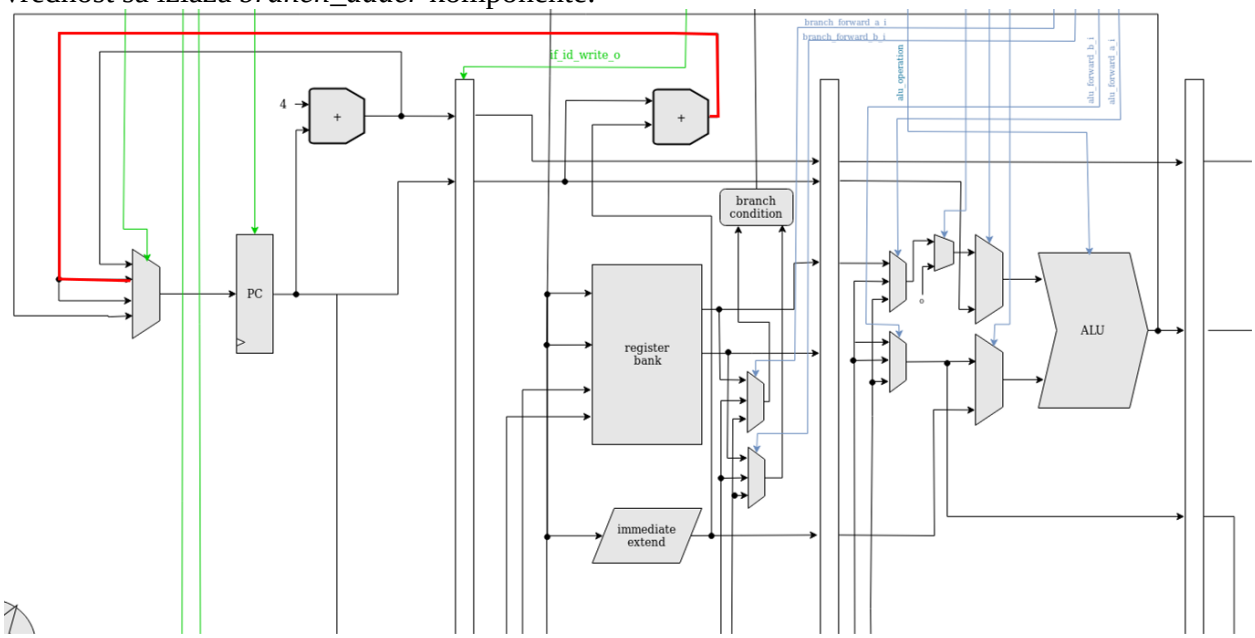
Naredno svojstvo tvrdi da ukoliko se JAL instrukcija pojavi u ID fazi protočne obrade, programski brojač uzima pravu vrednost kao što je prikazano na sledećoj slici, koja takođe predstavlja uvećanu sliku 3 :



slika 5. JAL instrukcija u ID fazi

Ako se pogleda izvorni fajl, svojstvo koje opisuje prethodno ponašanje napisano je u 27. liniji. Ono tvrdi da ukoliko procesor primi JAL instrukciju i ukoliko prethodna instrukcija nije JALR, naredno stanje koje programski brojač treba da uzme jeste vrednost *branch_adder* komponente kao na slici 5. JALR instrukcija ne sme da bude u EX fazi zato što bi to značilo da je skok već izvršen od strane JALR instrukcije, odnosno JAL instrukcija bi bila odbačena (*eng. Flushed*).

Kao i kod JAL instrukcije slično važi i za instrukcije uslovnog skoka. Odnosno ukoliko se instrukcija uslovnog skoka nađe u ID fazi, ostvaren je uslov za skok i JALR instrukcija se ne nalazi u EX fazi ostvariće se skok, odnosno programski brojač će kao i kod JAL instrukcije uzeti vrednost sa izlaza *branch_adder* komponente:



Slika 6. Intrukcija uslovnog skoka u ID fazi

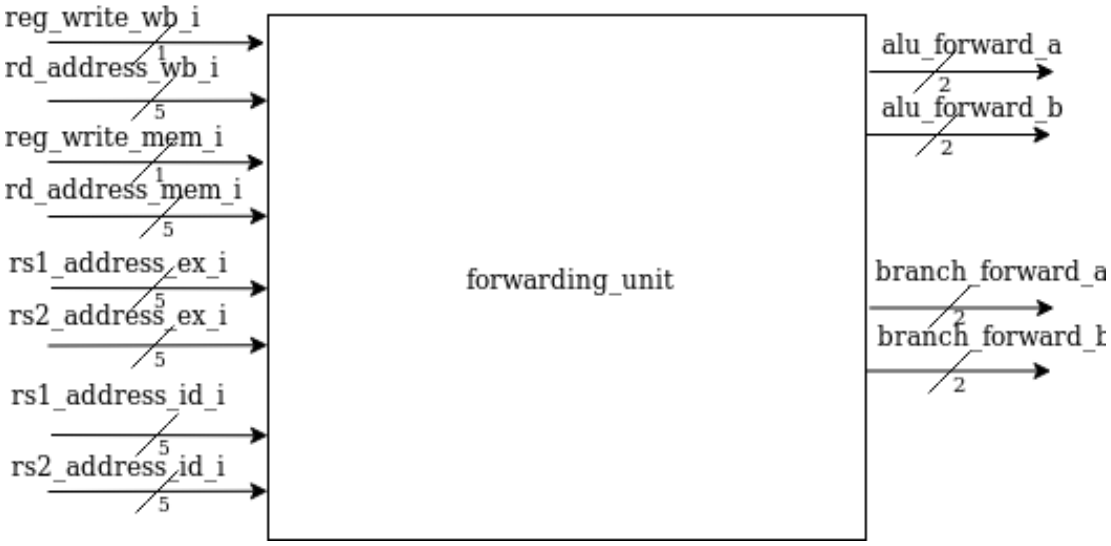
Svojstvo koje opisuje prethodno ponašanje napisano je u 30. i 31. liniji u izvornom kodu. U 30. liniji nalazi se dodatni uslov koji mora da bude ostvaren, odnosno ukoliko je on logička jedinica, to znači da uslovni skok treba da se desi, u suprotnom ne treba. 31. linija je Svojstvo, koje tvrdi da ukoliko procesor primi instrukciju uslovnog skoka, instrukcija prethodno primljena nije JALR i ako je ispunjen uslov skoka, onda će sledeće stanje programskog brojača biti izlaz *branch_adder* komponente kao na slici 6.

Poslednja dva svojstva su slična, i napisana su u 22. i 25. liniji izvornog koda. Ona tvrde da ukoliko se desi sekvenca da procesor primi prvo JALR instrukciju, a nakon toga JAL instrukciju ili instrukciju uslovnog skoka, tada će programski brojač uzeti izlaz sa ALU jedinice kao na slici 4.

3.2 Provera ispravnosti prosleđivanja (*eng. Forwarding)* podataka.

Da bi se izbegli hazardi podataka koja se javljaju usled postojanja protočne obrade neophodna je jedinica za prosleđivanje podataka (*eng. Forwarding unit*). Ona u pravovremenim trenutcima treba da prosleđuje podatke iz WB (*Write Back*) ili MEM (*memory*) faze protočne obrade na ulaze ALU jedinice u EX fazi ili na ulaze logike uslovnog skoka u ID fazi. Svojstva koja vrše provere nalaze se u ***forwarding_unit_checker.sv*** *i* ***forwarding_checker.sv*** izvornim fajlovima. U prvom izvornom fajlu nalaze se svojstva koja opisuju ispravnost *forwarding_unit* bloka posmatrajući ga kao crnu kutiju, dok se u drugom izvornom fajlu nalaze svojstva koja opisuju kako treba da se vrši prosleđivanje posmatrajući ceo procesor.

Da bi se razumela svojstva napisana u ***forwarding_unit_checker.sv*** izvornom fajlu, mora da se razume interfejs *forwading_unit* komponente:



slika 7. Interfejs *forwarding_unit* komponente

Specifikacija **forwarding_unit** komponente je sledeća:

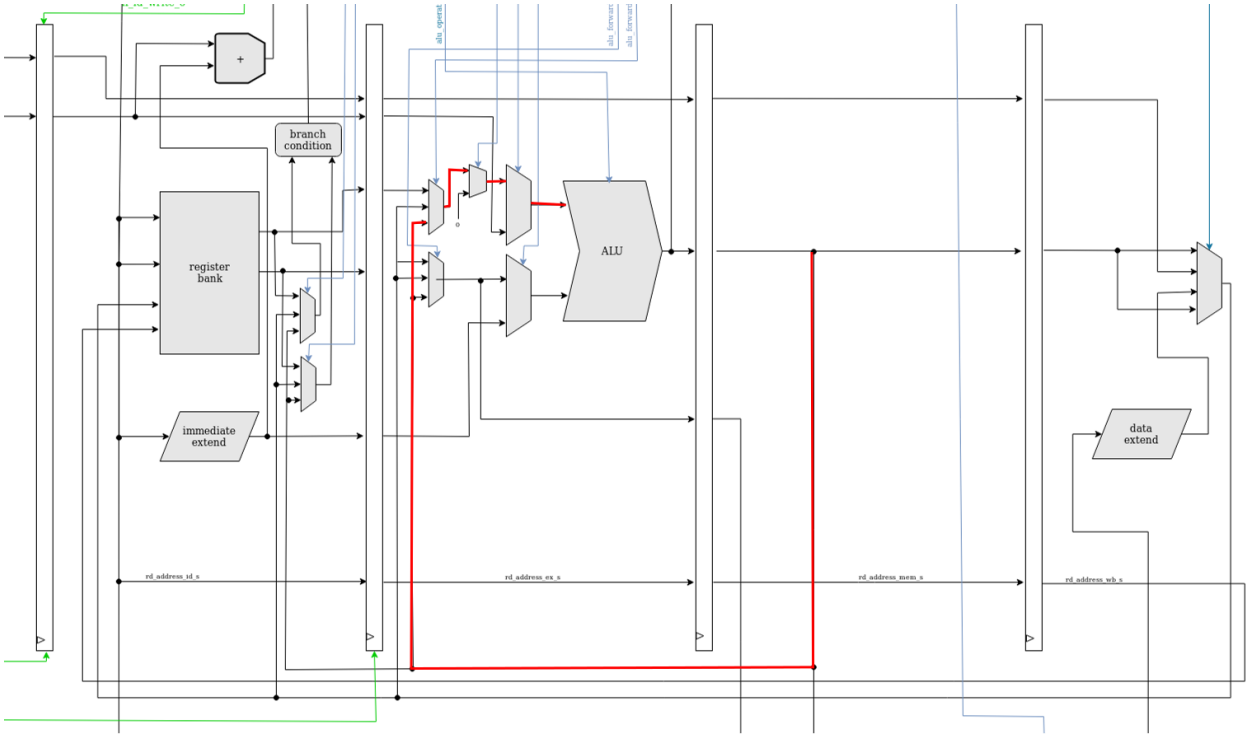
- a*lu_forward_a* izlaz će dobiti vrednost „00“ ukoliko nije potrebno da se vrši prosleđivanje na ulaz „a“ alu jedinice. Vrednost „01“ će se pojaviti ukoliko je potrebno da se prosledi podatak iz WB faze pri čemu je rs1_address_ex_i = rd_address_wb_i i reg_write_wb_i = 1. „10“ se pojavljuje ukoliko je potrebno prosleđivanje iz MEM faze i tada je rs1_address_ex_i = rd_address_mem_i i reg_write_mem_i = „1“. Ukoliko se desi da je neophodno istovremeno prosleđivanje iz MEM faze i WB faze, prednost ima MEM faza.
- Vrednosti za *alu_forward_b* izlaz se generišu na isti način kao i za *alu_forward_a*, stim što se umesto *rs1_address_ex_i* proverava *rs2_address_ex_i.*
- Vrednosti za *branch_forward_a* izlaz se generišu na isti način kao i za *alu_forward_a*, stim što se umesto *rs1_address_ex_i* proverava *rs1_address_id_i.*
- Vrednosti za *branch_forward_b* izlaz se generišu na isti način kao i za *alu_forward_a*, stim što se umesto *rs1_address_ex_i* proverava *rs2_address_id_i.*

Svojstvo koje je bilo od interesa proveriti jeste da li prosleđivanje iz MEM faze ima prednost u odnosu na prosleđivanje iz WB faze. Sitacija kada se to može desiti i kada *alu_forward_a dobija vrednost „10“ je kada je* rs1_address_mem_i = rd_address_mem_i, reg_write_mem = „1“ i ako je rs1_address_ex_i = rd_address_wb_i i reg_write_wb = „1“. Ovo je provereno pomoću dve tvrdnje u 43. liniji i 49. liniji izvornog koda, za prosleđivanje na ulaze

ALU jedinice i na ulaze logike uslovnog skoka respektivno. Pored ovog svojstva provereno je još da li je moguće istovremeno proslediti vrednosti iz MEM faze ili WB faze na oba ulaza alu jedinice ili oba ulaza logike uslovnog skoka. Ovo je provereno pomoću dve tvrdnje u 46. i 52. liniji izvornog koda, za ulaze ALU jedinice i ulaze logike uslovnog skoka respektivno.

Za proveru da li prosleđivanje funkcioniše na nivou celog procesora zadužena su svojstva u ***forwarding_checker.sv*** izvornom fajlu. Osam tvrđenja je napisano, 4 opisuju ispravno prosleđivanje na ulaze alu jedinice, dok ostala 4 opisuju uslove ispravnog prosleđivanja na ulaze logike uslovnog skoka.

Prvo tvrđenje, linija 69 izvornog koda, kaže da će rezultat na ulazu „a“ aritmetičko logičke jedinice biti jednak rezultatu aritmetičko logičke jedinice u MEM fazi ukoliko je *mem_alu_forward_a_check = „1“, ex_alu_opcode_check = „1“,* i ukoliko je *reg_write_mem_s = 1, odnosno* pravilno će se izvršiti prosleđivanje iz MEM faze u EX fazu. Sledeća slika daje vizuelni prikaz prethodno opisanog:

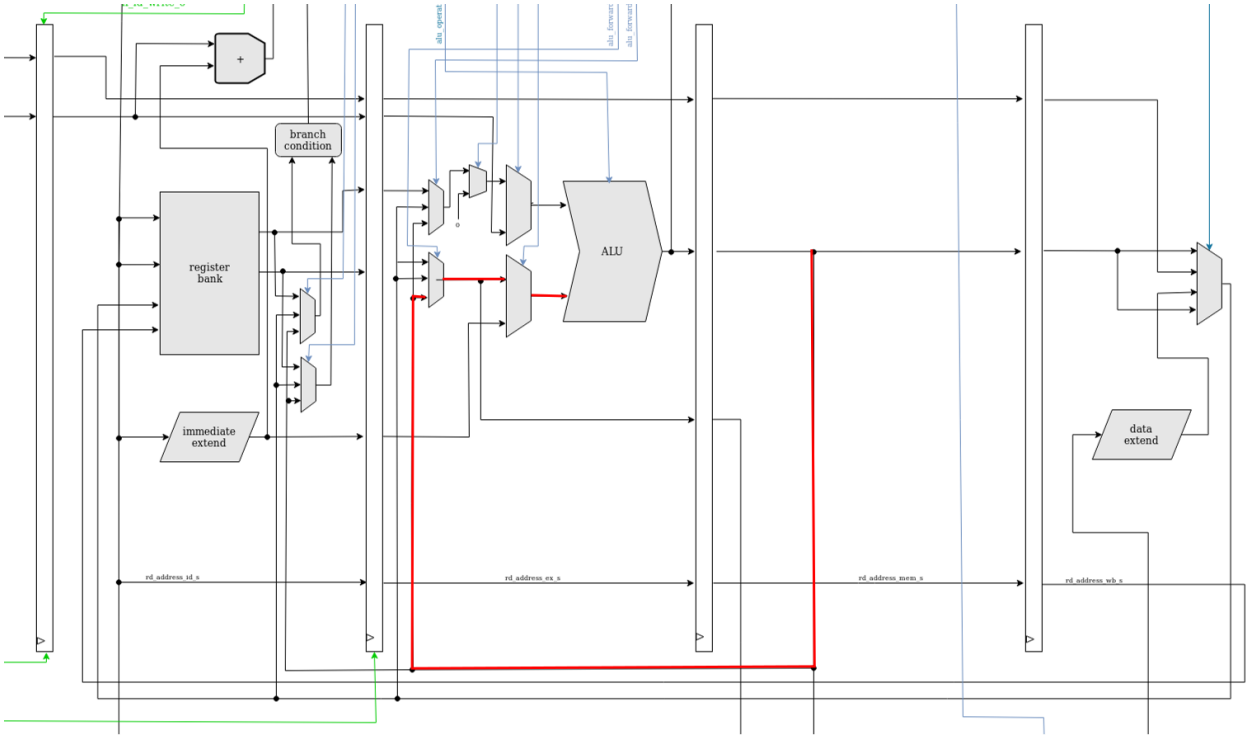


slika 8. Prosleđivanje iz MEM faze na “a” ulaz alu jedinice

Da bi se izvršilo prosleđivanje, *Mem_alu_forward_check* mora da bude na visokom logičkom nivou, odnosno rs1_address_ex_r = rd_address_mem_i rd_address_mem_i != 0. Ex_alu_opcode_check proverava da li se u EX fazi ne nalaze LUI instrukcija i AUIPC, jer kada se one pojave, mux-evi na prethodnoj slici neće propustiti prosleđenu (*eng.forwarded*) vrednost. Takođe ako se pogleda tvrđenje, uslov da je reg_write = 1 neophodan je tek u sledećem taktu nakon pojave *ex_alu_opcode_check = „1“* i *mem_alu_forward_a_check = „1“.* Razlog za to leži u hijerarhiji, odnosno *checker* u kome se nalaze tvrđenja instanciran je u *data_path-u,* a signal *reg_write_mem* nije vidljiv na tom nivou već je vidljiv samo *reg_write_wb* koji se prosleđuje iz *control_path-a.* Da bi se prevazišao ovaj problem, testira se da li je *reg_write_wb* jednak jedici takt kasnije, jer ako je on na visokom logičkom nivou, to znači da je u prethodnom taktu *reg_write_mem* bio na visokom nivou, stoga je uslov za prosleđivanje iz MEM faze u EX fazu ispunjen.

Drugo tvrđenje, linija 72. izvornog koda ima istu ulogu kao i prethodno, stim što se sada tvrdi da će se ispravna vrednost proslediti na „b“ ulaz aritmetičko logičke jedinice. Takođe

ex_alu_b_opcode_check dobija visoku vrednosti ukoliko se u u EX fazi ne nalazi niti JAL instrukcija niti „I“ tip instrukcija, jer u tim slučajevima prosleđena vrednost neće proći kroz mux-eve do „b“ ulaza alu jedinice. Sledeća slika ilustruje prethodno opisano:



slika 9. Prosleđivanje iz MEM faze na “b” ulaz alu jedinice

U 75. liniji izvornog koda napisano je tvrđenje koje opisuje kada se vrši prosleđivanje iz WB faze protočne obrade do „a“ ulaza alu jedinice u EX fazi. Vrši se ista provera kao i kod prvog tvrđenja, stim što se sada ne posmatra rd_address_mem, već rd_address_wb. Takođe sada se ne gleda samo vrednost reg_write u prethodnom ciklusu, kako je bilo u prethodnim slučajevima, već je bitno da se pojavi sekvenca reg_write = 1, reg_write = 0, što znači da u prethodno taktu nije bilo potrebno proslediti podatak iz MEM faze, ali u trenutnom taktu treba proslediti iz WB faze.

Poslednje tvrđenje vezano za prosleđivanje vrednosti na aritmetičko logičku jedinicu je isto kao i prethodno, jedina razlika je u tome što se sada posmatra ulaz „b“.

Ostala 4. tvrđenja u 93., 96., 99. i 102. liniji koda vezana su za prosleđivanje u ID fazu protočne obrade. Tvrđenja su ista kao i prva četiri, samo se sada vrednosti vraćaju na ulaze logike uslovnog skoka.

3.3 Provera ispravnosti rada programskog brojača.

U izvornom kodu ***pc_checker.sv*** nalaze se svojstva vezana za ispravnost rada programskog brojača. Svojstva koja su bila od interesa su: programski brojač uvek uvećava svoju vrednost za 4 ukoliko se ne dešavaju skokovi i nikada se neće desiti da programski brojač ne menja svoju vrednosti duže od 3 takta.

Prvo svojstvo je u 41. liniji koda, i ono tvrdi da ukoliko se u ID, EX, MEM fazama ne nalaze instrukcije skoka i ukoliko nema *stall-a,* programski brojač ili je jednak nuli (ovo je tačno samo na samo početku brojanja) ili se uvećava za 4 svaki takt.

Drugo tvrđenje u 44. liniji koda kaže da se nikada neće desiti da programski brojač ne menja svoju vrednost duže od 3 takta. 3 takta je maksimalna vrednost jer *stall* može da traje maksimum 2 takta, a nakon *stall-a* može da se pojavi instrukcija skoka, pri čemu se skače na istu vrednost na kojoj se nalazi brojač i odatle još jedan takt u kome programski brojač zadržava istu vrednost kao u prva dva takta.

3.4 Provera ispravnosti generisanja *stall* signala.

Izvorni fajl je ***stall_checker.sv***. Šest svojstava napisanih u ovom fajlu opisuju ispravno generisanje *stall* signala.

- Prvo svojstvo u 23. liniji koda tvrdi da će se stall u trajanju od 2 takta desiti jedino ako se pojavi sekvenca LOAD, uslovni skok, odnosno LOAD instrukcija u EX fazi, a uslovni skok u ID fazi protočne obrade.
- Drugo svojstvo u 25. liniji koda tvrdi da *stall* može da traje maksimalno 2 takta.
- Treće svojstvo u 27. liniji koda koda tvrdi da JAL instrukcija ne može da generiše stall signal.
- Četvrto svojsto u 30. liniji koda tvrdi da uslovni skok može da generiše *stall* signal u trajanju od 0, 1 ili 2 takta.
- Peto svojstvo u 34. liniji koda tvrdi da ukoliko se LOAD instrukcija nađe i EX fazi i ukoliko je ispunjen uslov za stall (rd_ex_s = rs1_id) generisaće se stall signal za sve instrukcije u ID fazi osim za JAL, LUI, AUIPC i NOP . Za te instrukcije nije potrebno izvršavati *stall* jer one ne pristupaju registru i registarskoj banci na adresi rs1_id.
- Šesto svojstvo u 38. liniji koda je isto kao i prethodno, stim što se posmatra da li je rd_ex_s = rs1_id, u toj situaciji generisace se *stall* za sve instrukcije u ID fazi osim za LUI, AUIPC, JAL, NOP i sve instrukcije I tipa.

4. Analiza pronađenih grešaka (*eng. Escape analysis*) i zaključak

Greške koje su uhvaćene nakon što je formalni alat validirao sva svojstva su sledeće:

- Tvrdnje *u Forwarding_unit_checker.sv* fajlu su pronašle grešku u *forwarding_unit* komponenti, i to je situacija kada je neophodno istovremeno prosleditи vrednosti na oba ulaza alu jedinice ili logike uslovnog skoka. Primer takve situacije je pojava sledeće sekvence instrukcija:

```
addi a0, a0, 10
add a0, a0, a0
```

- Uz pomoć tvrdnji u *stall_checker.sv fajlu pronađeno je nekoliko problema. Prvi* je da LUI, AUIPC i JAL izazivaju nepotreban stall. To je pokazala 5. tvrdnja. Drugi je da se ne dešava stall kada se pojavi LOAD instrukcija u EX fazi i JALR u ID fazi. To je pokazala isto 5. tvrdnja. Treći problem jeste da I tip instrukcija ne treba da izazove stall ukoliko rs2_id = rd_ex i LOAD instrukcija je u EX fazi. To je pokazala 6. tvrdnja.

Cilj ovog rada je bio da se procesor pored simulacionog testiranja, podvrgne dodatnim testovima pomoću formalnog alata i na taj način utvrdi postojanje ili nepostojanje određenih grešaka. Rezultat validiranja svojstava od strane formalnog alata bio je pronalaženje nekoliko bitnih grešaka. Svojstva koja je procesor validirao nisu zahtevala puno procesorskog vremena, te je u ovoj situaciji formalni alat bio jako koristan

Radili:
Nikola Kovačević E1 6/2018
Đorđe Mišeljić E1 5/2018

Literatura:

„SVA: The Power of Assertions in SystemVerilog, second edition“ - Eduard Cerny, Surrendra Dudani, John Havlicek, Dmitry Korchemny.

Potpun kod se može naći na sledećem repozitorijumu i grani *fv*:
https://github.com/DjordjeMiseljic/RISC_VHDL.git
Putanja do foldera sa *checker-ima* je sledeća:
RV32IMA/formal_verification/checkers