**Funkcionalna verifikacija**

**projekat**

**TEMA PROJEKTA:**

|  |
| --- |
| Funkcionalna verifikacija deskju(*eng. Deskew*) modula za preprocesiranje slika i SVM modula za klasifikaciju brojeva |

**TEKST ZADATKA:**

|  |
| --- |
| - napraviti verifikacioni plan  - napraviti verifikaciono okruženje  - ispitati pokrivenost |

Mentor Studenti

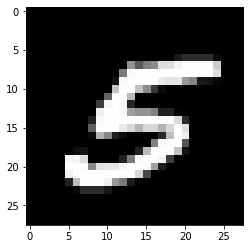
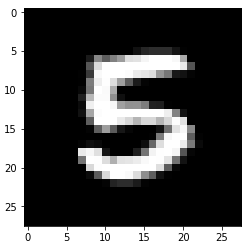
Andrea Erdeljan Đorđe Mišeljić ee4/2014

Nikola Kovačević ee24-2014

U Novom Sadu, 27. avgusta 2018. godine

**1.Uvod**

Tema projekta je funkcionalna verifikacija dva modula: Deskju(*eng. deskew*) I SVM (*eng. support vector macnije*). Funkcionalnost deskju (*eng. deskew*) modula je preprocesiranje slika, odnosno da sliku ukoliko je ona nakošena ispravi. Primer toga je prikazan na naredne dve slike:

*1.Pre preprocesiranja 2.Nakon preprocesiranja*

Na slici broj 1 se može videte slika koja nije ispravljena, a na slici 2 je prikazan rezultat obrade slike 1 od strane deskju (*eng. deskew*)modula. Drugi modul je SVM (*support vector machine*), njegova funkcionalnost je da zaključi koji se broj nalazi na slici koja mu je prosleđena (u ovom slučaju to bi bio broj 5). Dva modula funkcionišu zajedno, odnosno pre nego što se slika klasifikuje, ona biva preprocesirana od strane deskju(*eng. deskew*) modula, čime se postiže veća uspešnost klasifikacije.

Metodologija koja se koristi kako bi se verifikovala funkcionalnost ova dva modula je UVM (*eng. universal verification methodology).* Jedna od glavnih karakteristika ove metodologije je korišćeje UVC (engl. Universal Verification Component), odnosno univerzalnih verifikacionih komponenti koje imaju istu strukturu (sadrže monitore, drajvere, sekvencere, ...), što omogućava njihovo lako korišćenje bilo kao nezavisnih komponenti ili unutar nekog većeg Sistema. Simulacije rada ova dva modula u UVM okruženju su pokretane u “Questasim advanced” simulatoru, čiji je proizvođač “mentor graphics”.

**2. Verifikaciono okruženje**

Verifikaciono okruženje kreirano za potrebe verifikacije deskju (eng. deskew) i SVM modula je prikazano na sledećoj slici:



*Slika 3.*

Verifikacioni okruženje prikazano na prethodnoj slici kao i svako okruženje bazirano na UVM metodologiji, sadrži sledeće kompononte: “test”, “environment”, “agent”, “configuration”, “scoreboard”, “monitor”, “driver”, “sequencer”, “sequence”, “sequence item”. Svaka od ovih komponenti predstavlja klasu opisanu pomoću sistem verilog (*eng. system verilog*) programskog jezika. Sve prethodno pomenute klase se već nalaze u UVM biblioteci, I nasleđivanjem ovih klasa je moguće prilagoditi okruženje tako da je moguće verifikovati bilo koji modul.

Komponente verifikacionog okruženja prikazanog na slici 3 su:

1. “bram\_sequence”
2. “axil\_sequence”
3. “axis\_sequence”
4. “bram agent”

* “bram driver”
* “bram monitor”
* “bram sequencer”

1. “axil agent”

* “axil driver”
* “axil monitor”
* “axil sequencer”

1. “axis agent”

* “axis driver”
* “axis monitor”
* “axis sequencer”

1. “interrupt agent”

* “interrupt monitor”

1. “scoreboard”
2. “configuration”
3. “environment”
4. “test”

**2.1** **“BRAM sequence”, “BRAM driver” i “BRAM sequencer”**

Da bi se testirala funkcionalnost deskju (*eng. deskew*) modula neophodno je da jedan deo okruženja implementira BRAM memoriju, odnosno da kada modul ima zahtev za čitanjem iz BRAM memorije neki deo okruženja odgovori na taj zahtev i prosledi odgovarajući podatak. Komponente zadužene za taj posao su “BRAM sequence”, “BRAM driver” i “BRAM sequencer**”.**

“BRAM driver” komponenta ima direktan pristup bram interfejsu modula. Ona čeka na zahtev za podatkom od strane deskju modula tako što proverava da li je modul postavio na logičku jedinicu “en” port čime daje znak da mu je potreban podatak iz memorije. Kada je uočen “en” signal, drajver prosleđuje sekvenci adresu lokacije sa koje bi deskju (*eng. deskew*) u stvarnom okruženju hteo da pročita iz bram memorije, i ona treba da vrati podatak koji je ekvivalentan podatku koji bi se nalazio u bram memoriji na datoj adresi. Kada drajver dobije taj podatak on ga postavlja na “out\_port” bram interfejsa, čime ga moduo dobija na korišćenje (odnosno podatak je isčitan iz memorije). Ova komunikacija između drajvera i sekvence se obavlja preko “bram sequencer” komponente koja je tu kako bi tu komunikaciju sinhronizovala. Način na koji se realizuje prethodno opisani postupak je:

1.virtual task body();

2. read\_deskew\_images();

3. req = bram\_frame::type\_id::create("req");

4. assert(std::randomize(num\_of\_images) with {num\_of\_images > 0; num\_of\_images <=5;};

5. forever begin

6. start\_item(req);

7. finish\_item(req);

8. if(image < num\_of\_images)begin

9. if(req.interrupt)begin

10. req.interrupt = 0;

11. image ++;

12. end

13. else begin

14. req.in\_data=images\_array[image \* 784 + req.address];

15. end

16. end

17. else

18. break;

19. start\_item(req);

20. finish\_item(req);

21. end // forever begin

22.endtask : body

1.task run\_phase(uvm\_phase phase);

2. forever begin

3. @(posedge vif.clk)begin

4. if(interrupt\_done == 1)begin

5. interrupt\_done = 0;

6. req.interrupt = 1;

7. continue;

8. end

9. address = vif.axi\_address;

10. if(vif.axi\_en)begin

11. if(vif.axi\_address < 784)begin

12. seq\_item\_port.

get\_next\_item(req);

13. req.address = address;

14. seq\_item\_port.

get\_next\_item(req);

15. vif.axi\_in\_data=req.in\_data;

16. seq\_item\_port.item\_done();

17. end

18 end

19. end

20. end

21.endtask: run phase

Kod sa leve strane realizuje rad „bram sequence“ komponente, dok kod sa desne strane realizuje rad „bram driver“ komponente. Unutar bram sekvence se nalaze sve slike koje bi deskju modul trebalo da obradi, i te slike se učitavaju iz fajla u drugoj liniji koda sa leve strane pozivom „read\_deskew\_images()“ funkcije i smeštaju niz pod nazivom „ images array“ (svaka slika je predstavljena pmoću 784 piksela u formatu 28x28). Broj slika koji je izvučen iz tog falja iznozi 100, i te slike se koriste kako bi se testirala ispravnost modula. U narednoj liniji koda se pravi objekat unutar koga će se nalaziti određeni podatak koji se šalje drajveru. Taj objekat je takođe klasa napisana u sistem verilog jeziku, i u toj klasi su definisana 3 polja : „data“, „address“ i „interrupt“. Prilikom komunikacije između drajvera i sekvence, ovaj objekat se prosleđuje, i na osnovu polja unutar njega sekvenca odnosno drajver zna sta treba da uradi. Nakon toga se vrši randomizacija promenljive „num\_of\_images“, i na osnovu vrednosti koja se tu dobije sekvenca zna koliko slika treba da se obradi. U 5. liniji koda kreće beskonačna for petlja i započinje se proces komunikacije između sekvence i drajvera. Proces započinje pozivanjem blokirajuće metode „start\_item()“ u 6. liniji koda koja zaustavlja izvršavanje sekvence dok ne dobije odgovor od drajvera. Drajver će odgovoriti sekvenci u onom trenutku kada deskju modulu bude neophodan podatak iz bram memorije, što se može videti u kodu na desnoj strani. Tu se vidi da je drajver takođe realizovan kao beskonačna petlja koja svaki takt proverava da li je deskju modul postavio logičku jedinicu na „en“ port. Kada se to desi, drajver poziva blokirajući metodu „get\_next\_item(req)“ (linija 12 ) koja čeka da se pozove „start\_item(req)“ metoda (ona je već pozvana u sekvenci), koja kada se pozove daje drajveru na raspolaganje req objekat napravljen u sekvenci. Kada drajver to vidi on nastavlja sa izvršavanjem, i postavlja „address“ polje unutar „req“ objekta na vrednost adrese koju vidi na bram interfejsu modula, i poziva metodu „item\_done(req)“, čime vraća promenjeni objekat sekvenci. Nakon toga se kontrola vraća sekvenci koja poziva metodu „finish\_item()“ u 7. liniji koda i nstavlja sa izvršavanjem. Nakon toga u 13. liniji koda sekvenca postavlja „data“ polje u „req“ objektu na određenu vrednost u zavisnoti od adrese koja je postavio drajver na „address“ polju „req“ objekta.Nakon toga se ponovo poziva u 19. liniji koda „start\_item(req)“ metoda koja prosleđuje drajveru „req“ objekat u kome se nalazi podatak koji drajver treba da postavi na „in\_data“ port bram interfejsa. Dok se ne pozove „start\_item(req)“ metoda drajver čeka u 14. liniji koda odnosno kod „get\_next\_item(req)“ blokirajuće metode, i kada se ona odblokira (sekvenca pozvala „start\_item()“ metodu), drajver postvalja na „in\_data“ port bram interfejsa podatak koji je modulu potreban, ponovo poziva „item\_done“ metodu, u 16. liniji koda čijim pozivanjem pozivanjem se odblokira „finish item()“ metoda u sekvenci, čime je čitanje iz bram memorije završeno i jedan piksel je poslat modulu. Prethodno opisani proces se ponavlja pri svakom narednom isčitavanju pojedinačnog piksela. Takođe kada modul završi sa obradom slike i spreman je da za obradu naredne slike , on na svom „interrupt“ interfejsu postavlja logičku jedinicu, i u skladu sa tim sekvenca prelazi na narednu sliku u nizu „ image\_array“. Informaciju da se interapt desio prvi vidi „interrupt monitor“ (slika 3) i on tu informaciju prosleđuje drajveru preko TLM konekcije(slika 3), odnosno kada se desi interapt poziva se metoda unutar drajver komponente koja menja vrednost „interrupt\_done“ promenljive čija vrednost se proverava u 4. liniji drajver koda. Kada se desi interapt i ta promenljiva se postavi na logičku jedinicu, ispunjava se uslov u 4. liniji koda i u „req“ objektu se „interrupt“ polje postavlja na logičku jedinicu, a „interrupt\_done“ promenljiva se vraća na logičku nulu. Promenu „interrupt“ polja u „req“ objektu sekvenca vidi prvi put kad joj drajver pošalje „req“ objekat nakon pojave interapta. Taj interapt sekvenca obrađuje u u 8. liniji koda, i ako se on desi uvećava se promenljiva image, preko koje sekvenca zna da treba da treba da šalje narednu sliku pri svakom narednom zahtevu za čitanjem deskju modula.

Kada se sve slike obrade sekvenca se završava, a samim ti se završava i simulacija.

**2.2 “BRAM monitor”**

Bram monitor je zadužen za nadgledanje portova na bram interfejsu, i da na osnovu podataka koje vidi na bram interfejsu “scoreboard” komponenti prosledi sliku koja treba da se deskjuuje, kao i deskjuovanu sliku kako bi referentni model unutar “scoreboard” komponente mogao da napravi svoje rezultate deskjuovanja, i njih uporedi sa rezultatima koje daje deskju modul. Način na koji je to realizovano oslanja se na sam način rada deskju modula. Deskju (eng. deskew) započinje sa obradom slike tako što izvlači piksel po piksel iz bram memorije I radi određene proračune sa njima, i taj proces ponavlja dok ne uradi proračune sa svim pikselima slike. Za vreme izvlačenja tih pisksela “bram monitor” to nadgleda, i smešta piksele slike u red koji se nalazi u “image\_transaction” objektu, i kada pokupi svih 784 piksela on taj objekat putem TLM komunikacije prosleđuje “scoreboard” komponenti, i time je “scoreboard” komponenti poslata slika koja nije deskjuovana (ispravljena). Kada je potrebno poslati “scoreboard” komponenti sliku koju je obradio moduo opet se koristi način rada deskju (*eng. deskew*) modula. Odnosno monitor posmatra trenutke kada modul postavlja “we” port na vrednost 0011, čime moduo govori da želi da upisuje u bram memoriju obrađenu sliku, a kako on upisuje u memoriju piksel po piksel, monitor ih redom kupi i smešta u red koji se nalazi u “image\_transaction” objektu sve dok ne pokupi sve piksele koji su obrađeni od strane modula. Kada se to desi monitor preko TLM interfejsa prosleđuje “scoreboard” komponenti “image\_transaction” objekat unutar koga se nalazi slika koju je obradio modul, nakon čega referentni model unutar “scoreboard” komponente vrši poređenje svojih rezultata i rezultata koje je dobio od modula.

**2.3 “axil\_sequence” , “axil driver” i “axil sequencer”**

Axi lajt (*eng. axi lite*) interfejs deskju(*eng.deskew*) I SVM modula omogućava da se preko njega pošalje komanda modulu da započne sa obradom slike, da se pročita da li je moduo spreman za obradu naredne slike ili u slučaju SVM modula da se pročita klasifikovani broj (. Kako bi ovo moglo da se uradi, mora da se ispoštuje standardni AXI lajt (*eng. AXI lite*) protokol, i za to je zadužena „axil driver“ komponenta verifikacionog okruženja. Preko “axil sequence” komponente se šalju zahtevi za čitanjem preko AXI lajt (*eng. AXI lite*) interfejsa ili zahtevi za upisom, i pri svakom tom zahtevu “axil driver” kompontenta vodi računa da protokol bude ispoštovan, i ako se radilo o čitanju “axil driver” vraća pročitanu vrednost sekvenci preko “req” objekta. Komunikacija između “axil driver” komponente i “axil\_sequence” komponente se odvija preko sekvencera, koji je zadužen da ta komunikacija ispoštuje određeni UVM protokol.

**2.4 “axil\_monitor”**

“axil monitor” komponenta je zadužena za nadgledanje AXI lajt (*eng. axi lite*) interfejsa, i da svaki put kada uoči čitanje unutrašnjih registara modula preko ovog interfejsa, pošalje pročitani podatak “scoreboard” komponenti. Slanje podatka se obavlja preko TLM interfejsa. Takođe unutar “axil monitor” komponente se vrši merenje pokrivenosti(*eng. coverage*), i time se potvrđuje da li su se sve moguće kombinacije upisa i čitanja preko AXI lajt (*eng. AXI lite*) interfejsa desile. Kombinacije koje je neophodno pokriti su:

1. Da li su se desila čitanja iz svih unutrašnjih registara modula
2. Da li se desilo upisivanje u sve unutrašnje register modula
3. Da li se desilo čitanje posle upisa
4. Da li se desio upis posle čitanja
5. Da li se desio upis posle upisa
6. Da li se desilo čitanje posle čitanja

**2.5 “axis\_sequence” , “axis driver” i “axis sequencer”**

Iz razloga što SVM modul za komunikaciju u realnom okruženju koristi AXIS interfejs, preko koga dobija neophodne podatke koji su mu neophodni za komunikaciju, neohpodno je da okruženje implementira komponente koje će biti zadužene da se ispoštuje AXIS standardni protokol, kao i da se obezbedi dobijanje slike koja se klasifikuje kao i neophodnih podataka koje SVM modul mora da dobija. Slike za klasifikaciju kao i podaci koji su neophodni SVM modulu nalaze se u “axis sequence” komponenti, i ona ih šalje tačno onim redosledom kojim modul mora da ih dobija kako bi pravilno izvršio proračune. Podatak koji je potreban modulu se šalje svaki put kada modul na svom “interrupt” interfejsu postavi logičku jedinicu, u tom trenutku “axis drajver” komponenta preko “get\_next\_item()” metode traži podatak, sekvenca mu ga daje, i on ga postavlja na AXIS interfejs pošujući AXIS standardne protokole. Način na koji je realizovano da “axis driver” komponenta sazna da se dogodio interapt je isti kao I kod “bram driver” komponente, odnosno svaki put kada se desi interapt, “interrupt monitor” komponenta koja nadgleda “interrupt” interfejs, preko TLM interfejsa kojim je povezan sa “axis driver” komponentom obaveštava drajver. Tada se poziva metoda unutar “axis driver” komponente, određena promenljiva se menja, na osnovu čega drajver zna da treba da postavi na AXIS interfejs podatak. Komunikacija između drajvera i sekvence odvija se preko sekvencera, koji je zadužen da se prilikom te komunikacije ispoštuju protokoli UVM metodologije.

**2.6 “axis\_monitor”**

Uloga “axis” monitor komponente je da nadglead AXIS interfejs, I da svaki put kada se postavi podatak na interfejst, pri čemu je ispoštovan AXIS protokol, taj podatak prosledi “scoreboard” komponenti. Način na koji se podatak prosleđuje iz “axis monitor” komponente ka “scoreboard” komponenti je preko TLM interfejsa ( pogledati sliku 3).

**2.7 “interrupt monitor”**

“interrupt monitor” komponenta je zadužena za nadgledanje interapt interfejsa modula. I da kada se on desi o tome preko TLM interfejsa obavesti “axis driver” komponentu, kao I “bram driver” komponentu.

**2.8 “bram agent”, “axil agent”, “axis agent”, “interrupt agent”**

Uloga agenta je da unutar sebe okupi drajver, sekvencer monitor komponentu, i na taj način obezbedi mogućnost ponovnog korišćenja (*eng. reusability*).

**2.9 “scoreboard”**

Unutar “scoreboard” komponente su implementirani referenti modeli, koji na osnovu podataka koje “scoreboard” dobija od monitora treba da urade svoje proračune koje porede sa rezultatima koje su dali moduli. Referentni modeli koji su implementirani su referenti model za deskju (*eng. deskew*) komponentu i SVM komponentu. Referentni model deskju modula je funkcija koje se poziv unutar “write\_bram ” funkcije a ona se poziva kada “bram monitor” komponenta pošalje neobrađenu sliku “scoreboard” komponenti. Tada se funkciji koja je referentni model prosleđuje neobrađena slika na osnovu koje ona vrši sopstvenu obradu slike, i obrađenu sliku smešta u niz. Kada “bram monitor” komponenta pošalje sliku obrađenu od strane deskju (*eng.deskew*) modula, opet se poziva “write\_bram” funkcija stim što se sada poredi slika obrađena od strane referentnog modela I slika obrađena od strane modula, I ako postoji razlika prijavljuje se greška. Funkcija koja realizuje referentni model SVM modula se poziva tek kada “scoreboard” komponenta dobije sve neophodne podatke od “axis monitor” komponente. “scoreboard” prima podatke od“axis monitor” komponente tako što svaki puta kada monitor pošalje podatak pozove se “write\_axis(axis\_frame tr)” funkcija koja te podatke čuva u odgovarajućim nizovima. Kada se dobije poslednji podatak, poziva se funkcija koja je referenti model, i ona radi sopstvenu klasifikaciju broja koji se nalazi na slici koja je prosleđena od strane “bram monitora”. Poređenje broja koji je klasifikova referentni model i svm modul radi se u “write\_axil(axil\_frame tr)” funkciji kada se preko AXI lajt (*eng. AXI lite*) interfejsa pročita broj koji je klasifikovao SVM modul.

**2.10 “environment” i “configuration”**

Komponenta “enviorment” instancira i enkapsulira prethodno opisane komponente, ali samo one koje su neophodne kako bi verifikovao deskju ili SVM modul. “Environment” komponenta zna koje komponente treba da instancira na osnovu “configuration” komponente. Polja unutur “configuration” komponente se koriste kako bi se odredilo koje komponente okruženja će se koristiti pri verifikaciji modula a koje neće. Polja koja to određuju su “is\_axis” i “is\_bram”. Ako se verifikuje deskju(*eng.deskew*) modul tada se “is\_bram” polje postavlja na vrednost “WITH\_BRAM” a polje “is\_axis” na vrednost “WITHOUT\_AXIS”, dok kada se verifikuje SVM komponenta “is\_bram” se postavlja na vrednost “WITHOUT\_BRAM”, a polje “is\_axis” na vrednost “WITH\_AXIS”. U slučaju da se verifikuje deskju (*eng. deskew*) modul neće se instancirati “axis agent”, dok ako se verifikuje SVM modul neće biti instanciran “bram agent”, dok će sve ostalo ostati isto.

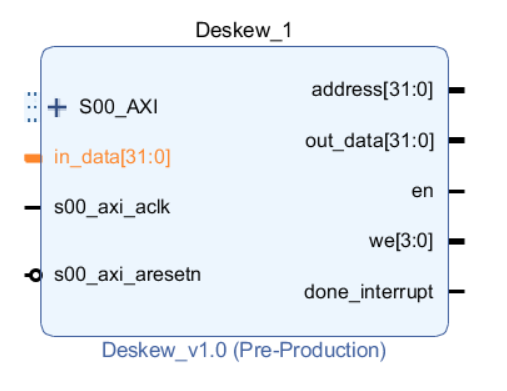
**2.11 “Test”**

“Test” objekat instancira I enkapsulira “environment” komponentu i pokreće sekvence na određenom sekvenceru. Koju sekvencu pušta zavisi od toga koji moduo se verifikuje (deskju ili SVM), a to zaključuje na osnovu polja u “configuration” komponenti. U slučaju da se verifikuje deskju (*eng. deskew*) modul, onda se na “bram sequencer” komponenti pušta “bram\_seq” sekvenca, a na “axil sequencer” komponenti se pušta “dskw\_axil\_seq” sekvenca, dok ako se verifiuje SVM modul, na “axis sequencer” se pušta “axis\_seq” sekvenca, a na “axil\_sequencer” se pušta “axis\_seq” sekvenca.

**3. Fukcionalna verifikacija deskju(*eng. deskew*) modula**

**3.1 Opis deskju (*eng. deskew*) modula**

Kao što je prethodno navedeno uloga deskju (*eng. deskew*) modula je preprocesiranje slika radi povećanja uspešnost klasifikacije brojeva koju vrši SVM modul. Slika koju Deskew treba da obradi predstavljena je u formatu 28x28 piksela (ukupno 784 piksela) i nalazi se u dvopristupnoj BRAM memoriji kod koje je na jedan pristup povezan Deskew modul koji ima mogućnost čitanja i pisanja iz memorije, a preko drugog pristupa processor upisuje sliku koja treba da se obradi, odsnosno isčitava sliku kada je deskju(*eng.deskew*) modul obradi. Veličina BRAM memorije je 2048 lokacija, svaka od lokacija predstavlja vrednost jednog piksela, gde se na prvih 784 lokacije upisuje slika koja treba da se obradi, a na lokacije od 784 do 1577 deskju(*eng. deskew*) modul upisuje obrađenu sliku (slike koje se obrađuju su u format 28x28 piksela). Način na koji deskju (*eng deskew*) modul komunicira sa svojom okolinom je preko AXI lajt (*eng. AXI lite*) interfejsa ,BRAM interfejsa, i interapt interfejsa kao što se može videte na narednoj slici:



*Slika 4*

-S00\_axi – predstavlja standardni AXI lite interfejs. Preko njega se može poslati komanda da modul započne sa obradom slike, kao i da se proveri da li je modul spreman za obradu slike ili još uvek nije završio sa obradom prethodne.

-BRAM – interfejs deskju(*eng. deskew*) modula predstavlja način komunikacije modula sa BRAM memorijom. Ovaj interfejs se sastoji od sledećih ulaznih i izlaznih portova:

* Address –izlazni port preko koga deskew šalje adresu lokacije sa koje hoće da pročita p-odatak iz BRAM memorije, ili lokacije na koju hoće da upiše podatak.
* In\_data – ulazni port preko koga modul čita podatke iz BRAM memorije.
* Out\_data – izlazni port preko koja modul upisuje podatke u BRAM memoriju.
* en – izlazni kontrolni port preko koga se omogućava čitanje ili upis u memoriju.
* we – izlazni kontrolni port preko koga se dozvoljava upis podataka u memoriju.

-Done\_interrupt – port obaveštava kada je deskew modul završio sa obradom slike .

-s00\_axi\_aclk – ulazni port preko koga se dovodi klok signal.

-s00\_axi\_areset – ulazni port preko koga se dovodi reset signal.

**3.2 Verifikacioni okruženje deskju (*eng. deskew*) modula**

Pri verifikaciji deskju (*eng.deskew*) modula polja u “configuration” komponenti verifikacionog okruženja se postavljaju u skladu sa tim koja se komponenta verifikuje, i iz tog razloga verifikaciono okruženje izgleda kao na sledećoj slici:



Slika 5

Na slici se može uočiti da okruženje nema “axis agent” komponentu za razliku od okruženja na slici 3, razlog tome je što deskju(*eng. deskew*) modul nema AXIS interfejs. Takođe zbog podešavanja polja u “configuration” komponenti, “test” komponenta pokreće samo dve sekvence: “dskw\_axil\_seq” sekvencu koja je povezana sa “axil\_sequencer” komponentom, I “bram\_seq” koja je povezana sa “bram sequencer” komponentom. “Bram seq” je već opisana, dok je “dskw\_axil\_seq” sekvenca koja šalje specifične komande namenjene AXI lajt (*eng. AXI lite*) interfejsu deskju(*eng. deskew*) modula. Ona je napisana tako da šalje modulu naredbu da krene sa obradom slike, nakon čega ona proverava da li je modul završio sa obradom tako što šalje naredbu za čitanje na AXI lajt (*eng. AXI lite*) interfejs, i kada vidi da je modul spreman ponovo šalje komandu da moduo započne sa obradom naredne slike. Koliko puta će “dskw\_axil\_seq” da pošalje naredbu za početak obrade slike zavisi od toga koliko slika želi da se obradi, a to zavisi od ceolobrojne promenljive “num\_of\_images” koja se randomizuje na početku simulacije, čija vrednost je ograničena na opsegu između 1 i 100 (broj slika koji če se obraditi je između 1 i 100)

**3.3 Verifikacioni plan deskju (*eng. deskew*) modula**

Verifikacioni plan se sastoji iz sledećih koraka:

1. Provera funkcionalnosti reseta sistema. Ona je proverena na početku simulacije kada se ceo sistem resetuje.
2. Provera funkcionalnosti AXI lajt (*eng. Axi lite*) interfejsa. Ona se ogleda u proveri da li moduo poštuje AXI lajt (*eng. Axi lite*) protokol ili ne. Na osnovu signala koji su dobijeni nakon simulacije vizuelno je zakljućeno da moduo poštuje protokol.
3. Provera da li modul obavlja željenu funkcionalnost. Ona se vrši u “scoreboard” komponenti i detaljno objašnjenje je dato u 2.9.

**3.4 Verifikaciona pokrivenost deskju (*eng. deskew*) modula**

Provera pokrivenosti vrši se nadgledanjem AXI lajt (*eng. AXI lite*) interfejsa. Situacije koje su od posebnog značaja I koje je potrebno pokriti su:

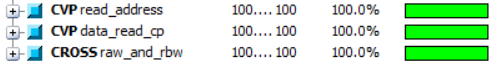
1. Da li se desilo upisivanje u sve unutrašnje register modula
2. Da li su se desila čitanja iz svih unutrašnjih registara modula
3. Da li se desilo čitanje posle upisa
4. Da li se desio upis posle čitanja
5. Da li se desio upis posle upisa
6. Da li se desilo čitanje posle čitanja

Nakon puštanja simulacije, rezultati merenja pokrivenosti su sledeći:



Slika 6

Prethodna slika potvrđuje stavku 1) sa prethodnog spiska, čime se potvrđuje da je izvršeno upisivanje u sve register.



Slika 7

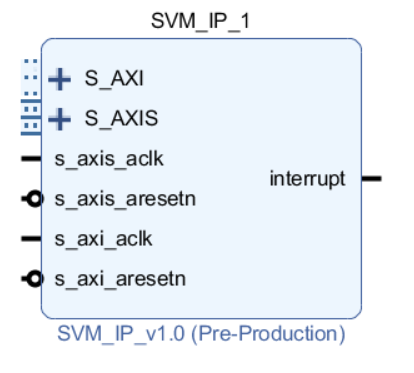
Na prethodnoj slici prvi CVP (*eng. Cover point*) govori o tome da li je tokom simulacije čitano iz svih unutrašnjih registara (druga stavka sa gornjeg spiska), drugi CVP govori o tome da li se menjala vrednost unutrašnjih registara, odnosno da li su se pojavile sve moguće vrednosti kod unutrašnjih registara, i treća pokrivenost predstavlja presek (*eng. cross*) između prve dve čime se dobija uvid u to da li se pojavila 3), 4), 5) i 6) stavka. Na osnovu rezultata vidimo da se svaka od stavki pojavila, odnosno pokrivenost je savšena.

**4. Fukcionalna verifikacija SVM (*eng. Support Vector Machine*) modula**

**4.1 Opis SVM (*eng. Support Vector Machine*) modula**

U mašinskom učenju SVM (*Support Vector Machine*) je binarni klasifikator koji na osnovu skupa podataka na kojima je istreniran (pomoćnih vektora) odlučuje da li ulazni podatak pripada nekoj klasi ili ne. Kako bi doneo tu doluku za nepoznati ulazni podatak, on čuva neke od podataka na kojima je treniran i njima pridružene težine (*lambde*) i znak (*target*).

Moduo za klasifikaciju cifre je isprojektovan tako da započne rad postavljanjem unutrašnjeg registra *start* na logičku jedinicu. Rezultat se čita iz unutrašnjeg *cl\_num* registra na sličan način nakon što se klasifikacija završi. Moduo prima preko fifo kanala prvo sliku koju je potrebno klasifikovati a zatim sve podatke koji su potrebni za to. Klasifikator šalje prekidni (*interrupt*) signal svaki put kada mu je potreban neki podatak, dok procesor odgovara na te zahteve saljući odgovarajuće podatke iz memorije preko modula za pristup  RAM-u (*DMA, Data memory access*). Moduo komunicira sa svojom okolinom (processor, RAM) preko tri interfejsa: AXI lajt (*eng. AXI lite*), AXIS (*eng. AXI Stream*) i interapt interfejs. Preko prvog se šalju komande modulu da započne komunikaciju, proverava da li je moduo spreman za narednu klasifikaciju ili se čita rezultat klasifikacije, preko se drugog primaju podaci (slika koja se klasifikuje, pomoćni vektori itd.), dok preko trećeg moduo obaveštava da mu treba naredni podatak. Interfejs SVM modula prikazan je na sledećoj slici:



Slika 8

Sa slike se vide sledeći interfejsi:

* -S\_AXI – predstavlja standardni AXI lajt (*eng. AXI lite*) interfejs
* -S\_AXIS- predstavlja standardni AXI protočni (*eng. AXI stream*) interfejs.
* -interrupt – predstavlja izlazni port koji daje znak da je modulu potreban naredni podatak
* -s\_axis\_aclk – ulazni AXIS klok signal
* -s\_axi\_aclk – ulazni AXIL klok signal
* -s\_axis\_aresetn – ulazni AXIS reset signal
* -s\_axi\_aresetn – ulazni AXIL reset signal

**4.2 Verifikacioni okruženje SVM modula**

Pri verifikaciji SVM modula polja u “configuration” komponenti verifikacionog okruženja se postavljaju u skladu sa tim koja se komponenta verifikuje, i iz tog razloga verifikaciono okruženje izgleda kao na sledećoj slici:



Slika 9

Na slici se može uočiti da okruženje nema “bram agent” komponentu za razliku od okruženja na slici 3, razlog tome je SVM modul nema bram interfejs. Takođe zbog podešavanja polja u “configuration” komponenti, “test” komponenta pokreće samo dve sekvence: “svm\_axil\_seq” sekvencu koja je povezana sa “axil\_sequencer” komponentom, I “axis\_seq” koja je povezana sa “axis sequencer” komponentom. “axis seq” je zadužena za slanje slike koja se klasifikuje, kao I neophodnih podataka koji potrebni za klasifikaciju (objašnjenje u 2.5), dok je “svm\_axil\_seq” sekvenca koja šalje specifične komande namenjene AXI lajt (*eng. AXI lite*) interfejsu SVM modula. Ona je napisana tako da šalje modulu naredbu da krene sa obradom slike, nakon toga ona proverava da li je modul završio sa obradom tako što šalje naredbu za čitanje na AXI lajt (*eng. AXI lite*) interfejs, i kada vidi da je modul spreman tada zna da je modul završio sa klasifikacijom nakon čeka šalje komandu za čitanje registra u kome se nalazi rezultat klasifikacije. Nakon toga sekvence ponovo šalje komandu da moduo započne sa obradom naredne slike. Koliko puta će sekvenca “svm\_axil\_seq” da pošalje naredbu za početak klasifikacije slike zavisi od toga koliko slika želi da se obradi, a to zavisi od ceolobrojne promenljive “num\_of\_images” koja se randomizuje na početku simulacije, čija vrednost je ograničena na opseg između 1 i 10 (broj slika koji če se klasifikovati je između 1 i 10). Razlog za tako mali opseg slika koje se klasifikuju je to što je simulatoru potrebno 15 minuta da klasifikuje samo jednu sliku.

**4.3 Verifikacioni plan SVMmodula**

Verifikacioni plan se sastoji iz sledećih koraka:

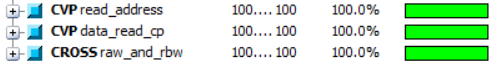
1. Provera funkcionalnosti reseta sistema. Ona je proverena na početku simulacije kada se ceo sistem resetuje.
2. Provera funkcionalnosti AXI lajt (*eng. Axi lite*) interfejsa. Ona se ogleda u proveri da li moduo poštuje AXI lajt (*eng. Axi lite*) protokol ili ne. Na osnovu signala koji su dobijeni nakon simulacije vizuelno je zakljućeno da moduo poštuje protokol.
3. Provera da li modul obavlja željenu funkcionalnost. Ona se vrši u “scoreboard” komponenti i detaljno objašnjenje je dato u 2.9.

**4.4 Verifikaciona pokrivenost SVM modula**

Analiza pokrivenosti SVM modula je ista kao i kod deskju (*eng. deskew*) modula, te će ovde biti prikazani samo rezultati:



slika 10



Slika 11