|  |  |  |
| --- | --- | --- |
|  | UNIVERZITET U NOVOM SADU FAKULTET TEHNIČKIH NAUKA  KATEDRA ZA MIKRORAČUNARSKU ELEKTRONIKU |  |

Funkcionalna verifikacija hardverske implementacije sistema za klasifikaciju brojeva

Kandidat mentor Kovačević Nikola Vuk Vranjković

Jul 2018

Sadržaj

[1.Uvod 3](#_Toc523515835)

[2. Funkcionalna specifikacija 6](#_Toc523515836)

[2.1 Opis interfejsa 6](#_Toc523515837)

[2.2 Teorijska analiza modula za ispravljanje slike (*eng. deskew module*) 11](#_Toc523515838)

[2.3 Specifikacija modula za ispravljanje slike (*eng. deskew module*) nakon hardverske implementacije 14](#_Toc523515839)

[2.3 Teorijska analiza SVM (Support Vector Machine) modula 16](#_Toc523515840)

[2.4 Specifikacija SVM (*eng. Support Vector Machine*) modula nakon hardverske implementacije 17](#_Toc523515841)

[3. Verifikacioni plan 17](#_Toc523515842)

[4. Verifikaciono okruženje 17](#_Toc523515843)

[4.1 Opis UVM metodologije 17](#_Toc523515844)

[4.2 Opis verifikacionog okruženja 17](#_Toc523515845)

[4.2.1 “BRAM sequence”, “BRAM driver” i “BRAM sequencer” 19](#_Toc523515846)

[4.2.2 “BRAM monitor” 21](#_Toc523515847)

[4.2.3 “axil\_sequence” , “axil driver” i “axil sequencer” 22](#_Toc523515848)

[4.2.4“axil\_monitor” 22](#_Toc523515849)

[4.2.5 “axis\_sequence” , “axis driver” i “axis sequencer” 22](#_Toc523515850)

[4.2.6 “axis\_monitor” 23](#_Toc523515851)

[4.2.7“interrupt monitor” 23](#_Toc523515852)

[4.2.8“bram agent”, “axil agent”, “axis agent”, “interrupt agent” 23](#_Toc523515853)

[4.2.9 “scoreboard” 23](#_Toc523515854)

[4.2.10“environment” i “configuration” 24](#_Toc523515855)

[4.2.11“Test” 24](#_Toc523515856)

[4.3 Konfigurisanje verifikacionog okruženja 24](#_Toc523515857)

[5. Otklanjanje problema (*eng. debug*) verifikacionog okruženja i hdl dizajna 24](#_Toc523515858)

[6. Regresija, fabrikacija i analiza propuštenih bagova (*eng. escape analysis*) 24](#_Toc523515859)

[7. Zaključak 25](#_Toc523515860)

[8. Literatura 25](#_Toc523515861)

# 1.Uvod

Cilj hardverskog dizajna je da napravi uređaj koji obavlja određenu funkciju na osnovu specifikacije dizajna, dok uloga funkcionalne verifikacije nije samo pronalaženje bagova već i garantovanje da dizajn obavlja svoju funkciju, odnosno mora da potvrdi da je dizajn tačna reprezentacija specifikacije.

Proces verifikacije je paralelan procesu pravljenja dizajna. Dizajner čita hardversku specifikaciju određenog bloka, interpretira deskripciju opisanu ljudskim rečima, i pravi odgovarajuću logiku uglavnom na RTL nivou. Verifikacioni inženjer takođe čita hardversku specifikaciju i na osnovu nje pravi testove kojima utvđuje da li dizajn obavlja željenu funkcionalnost. Razlog zašto je ovo dobra praksa jeste što svaka specifikacija može da bude dvosmislena zbog nedostatka detalja ili konfliktnih opisa, te ako više od jedne osobe analizira istu specifikaciju povećava se šansa da specifikacija dizajna bude u potpunosti implementirana. Karakteristike koje čine uspešnog verifikacionog inženjera su:

* Sposobnost da razume dizajn
* Sposobnost da radi blisko i srdačno sa dizajn inženjerima
* Sposobnost da razume prednosti i mane različitih verifikacionih alata
* Sposobnost da koristi te alate kako bi pronašao bagove u dizajnu.

Pri postavljanju zadataka i ciljeva verifikacije moraju se uzeti u obzir sledeća ograničenja:

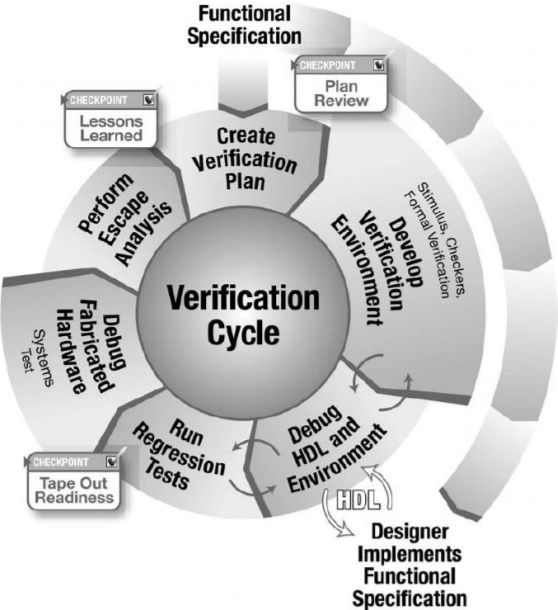
* Rok za završetak proizvoda - Uspešnost kompjuterskih proizvoda umnogome zavisi od pogađanja tržišta u pravo vreme.
* Cene - Troškovi procesa proizvodnje moraju biti svedeni na minum kako bi se uvećao profit
* Kvaliteta – Kupci očekuju uređaj koji će da zadovolji određene standard

Pronalaženje balansa uzimajući u obzir ograničenja je jako bitno i umnogome zavisi od proizvoda. Ovaj problem je jako težak iz razloga što se optimizovanjem dva ograničenja škodi trećem. Primer toga je skok cene proizvodnje usled skraćivanja roka za završetak proizvoda i maksimizovanja kvaliteta.

Proces verifikacije može da se predstavi verifikacionim ciklusom koji se sastoji iz sledećih koraka:

* Funkcionalna specifikacija
* Pravljenje verifikacionog plana
* Razvijanje verifikacionog okruženja
* Debug HDL-a i okruženja
* Pokretanja regresije
* Debagovanja fabrikovanog hardvera
* Eskejp (*eng.escape*) analize

Vizuelni prikaz prethodnih koraka prikazan je na slici 1:



**Slika 1. Verifikacioni ciklus**

Verifikacioni ciklus odvija se u pravcu kazaljke na satu, počevši od funkcionalne specifikacije, preko verifikacionog plana, sve do eskejp analize.

Funkcionalna specifikacija opisuje željeni proizvod i nju zadaje sistemski arhitekta. Ona obuhvata specifikaciju interfejsa preko koga komunicira, funkcije koju mora da izvršava i uslove koji utiču na dizajn.

Krucijalni deo verifikacionog ciklusa je verifikacioni plan jer on daje detaljan opis verifikacionih napora. On daje odgovor na pitanje “šta verifikujem” “ kako cu to da verifikujem”. Verifikacioni plan se sastoji od specifičnih testova i metoda, potrebnih alata, kriterijuma završetka verifikacije, resursa, funkcija koje treba da se verifikuju, funkcija koje ne moraju da se verifikuju na datom nivou hijerarhije.

Nakon što je verifikacioni plan završen započinje razvijanje verifikacionog okruženja. Ono se sastoji iz softverskog koda koji je uglavnom specifičan za dizajn i alata koji omogućuju koji su generični i mogu da se koriste na različitim projektima. Najbitnije komponente verifikacionog okruženja su stimulusi i čekeri, u zavisnosti od njih postoje sledeći tipovi verifikacionog okruženja:

* Deterministički
* Nasumični (*eng. Randomized*)
* Formalno nastrojeni
* Generatori testova

Naredni korak u verifikacionom ciklusu je debug dizajna i verifikacionog okruženja. U ovom koraku verifikacioni inženjer debaguje hardver pokrećući testove definisane u verifikacionom planu, i kako testovi odmiču pronalaze se i analiziraju anomalije. Analiza oktriva uzrok anomalije koji može da bude unutar okruženja ili unutar HDL dizajna. Ukoliko je anomalija u dizajnu verifikacioni inženjer o tome obaveštava dizajn tim na kome je da to ispravi, a ukoliko je anomalija u okruženju, onda je verifikacioni inženjer zadužen da ispravi tu grešku.

Nakon završenog debaga sledi process regresije, odnosno kontinualnog puštanja testova definisanih u verifikacionom planu. Ovaj korak je bitan iz 2 razloga:

* Prvi razlog je da verifikaciona okruženja često imaju elemente podložne randomizaciji, i na ovaj način se pušta što više različitih scenarija.
* Drugi razlog je da verifikacioni tim mora da ponovi sve testove nakon ispravljanja anomalija pronađenih u dizajnu ili okruženju kako bi proverili da sve funkcioniše pravilno.

Fabrikacija hardvera je korak koji sledi kada su zadovoljeni svi kriterijumi fabrikacije. Nakon fabrikacije sledi proces debaga, odnosno proizvod se postavlja u planirani sistem čime se testira da li će neka anomalija da se pojavi. U ovom delu se vidi najveći značaj verifikacije, a to je da se izbegnu problemi na fabrikovanom hardveru, jer oni mogu da budu veoma skupi.

Ako se prilikom testiranja fabrikovanog hardvera pronađu problemi, verifikacioni tim mora da izvrši eskejp (*eng. escape*) analizu kojom se utvrđuje zašto je verifikaciono okruženje previdelo taj problem.Verifikacioni tim reprodukuje problem u verifikacionom okruženju, kako bi ako je to moguće zaključili razlog zašto bag nije uhvaćen u verifikacionom ciklusu. Ovo je jako bitno jer verifikacioni tim uči na svojim greškama tako da u narednom verifikacionom ciklusu neće ponoviti iste.

Tema ovog rada se zasniva na praćenju prethodno navedenih smernica kako bi se verifikovala funkcionalnost dva modula: SVM (*eng. support vector machine*) i deskju (*eng. Deskew*). SVM modul je zasnovan na principima mašinskog učenja u uloga mu je da pravilno klasifikuje brojeve na slikama koje su mu prosleđene. Klasifikacija nije savršena, odnosno može se desiti da klasifikator pogreši, a kako bi se povećala stopa uspešnosti klasifikacije koristi se deskju modul. Njegova uloga je preprocesiranje slika kako bi se stopa uspešnosti povećala, i ono je realizovano tako da ako je neki broj na slici krivo napisan on bude ispravljen od strane deskju modula, i tek nakon toga prosleđen SVM modulu na klasifikaciju.

# 2. Funkcionalna specifikacija

## 2.1 Opis interfejsa

Interfejse koje SVM i deskju moduli poseduju su sledeći:

* BRAM
* AXI4-Lite
* AXI4-Stream

**BRAM** interfejs predstavlja specifičan interfejs komunikacije sa BRAM(*eng. Block RAM*) memorijom. Signali preko kojih modul komunicira sa BRAM memorijom su prikazani u narednoj tabeli:

Tabela 1. Signali BRAM interfejsa

|  |  |
| --- | --- |
| Naziv signala | Opis signala |
| in\_data | Ulazni port podataka iz BRAM memorije |
| out\_data | Izlazni port podataka ka BRAM memorije |
| Address | Ulazni port kojim se određuje sa koje lokacije se čita iz memorije ili upisuje. |
| en | Ulazni Kontrolni port čitanja podatak iz bram memorije. Kada je on visok može se čitati iz memorija |
| we | Ulazni Kontrolni port upisa podataka u bram memoriju. Kada je on visok može se upisivati u memoriju. |

Moduli koji poseduju BRAM interfejs uglavnom komuniciraju sa BRAM memoriju jer se u njoj nalaze neophodni podaci koje treba obraditi, unutar nje smeštaju rezultat obrade ili se unutar BRAM nalazi *lookup* tabela na koju se modul referencira.

AXI (*Advanced Extensible Interface*) magistrale predstavljaju poslednju evoluciju ARM AMBA (*Advanced Microcontroller Bus Architecture*) magistrale. Trenutno je aktuelna četvrta verzija AXI protokola, AXI4, standardizovana 2010. godine. Kao što je več rečeno, AXI protokol pripada grupi ARM AMBA familije SoC magistrala. AMBA protokol je specifikacija otvorenog tipa koja omogućava povezivanje i kontrolu velikog broja kontrolera i periferijskih jedinica unutar multi-master sistema. Kao i većina ostalih članova AMBA familije, AXI je namenjena za korišćenje unutar SoC sistema visokih performansi, koji rade na visokim učestanostima. AXI komunikacioni protokol ima sledeće karakteristike:

* Adresne i kontrolne faze odvojene su od faza prenosa podataka
* Kontrolni signali validnosti pojedinačnih bajtova prilikom prenosa podataka (*Byte Strobes*) omogućavaju neporavnate prenose podataka
* Podržan je prenos podataka u blokovima (*Burst-based Transaction*), pri čemu je neophodno specificirati samo početnu adresu bloka
* Kanali za upis i čitanje podataka su razdvojeni, omogućavajući jednostavnu implementaciju DMA prenosa
* Inicirane transakcije mogu se kompletirati u redosledu koji je različit od redosleda u kom su zadate (*Out-of-order Completion*)
* Moguće je jednostavno uvođenje protočne obrade, u cilju zadovoljavanja zahtevanih vemenskih karakteristika Sistema

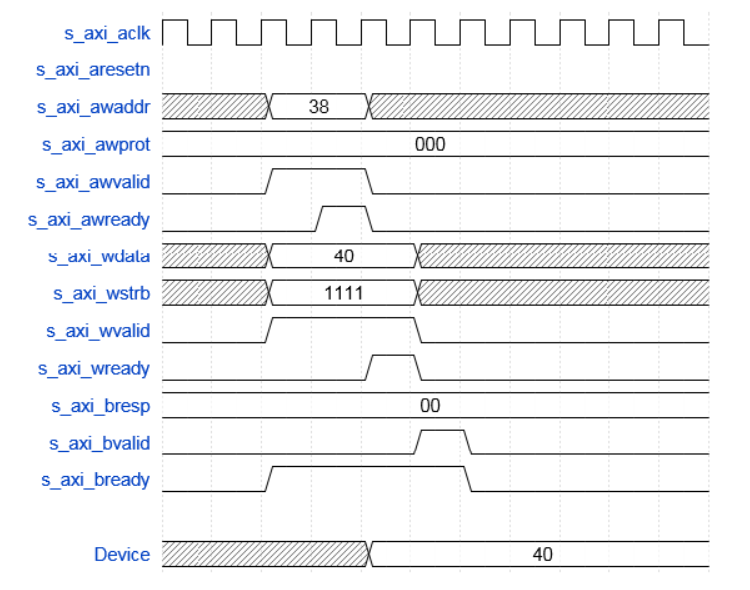
Sve AXI kompatibilne periferije dele se u dve grupe: AXI mastere i AXI slejvove. sve AXI kompatibilne periferije dele se u dve grupe: AXI mastere i AXI slejvove. AXI masteri iniciraju transakciju, dok AXI slejvovi mogu samo da odgovaraju na započetu transakciju. U svakoj AXI transakciji mora da učestvuje tačno jedan AXI master (*AXI Master*) i jedan AXI slejv (*AXI Slave*).

**AXI4*-Lite (AXIL)*** –Koristi za komunikaciju sa memorijski mapiranim modulima, ali kod kojih ne postoji mogućnost prenosa podataka u blokovima, već je moguće prenositi samo jedan podatak po transakciji. Usled toga su hardverski resursi neophodni za implementaciju AXI4-*Lite* jako mali, kao i njegov interfejs. Tabela signala AXIL **slave** interfesja je:

*Tabela 2. Signali AXI-Lite interfejsa u slučaju AXI slejv komponente*

|  |  |
| --- | --- |
| Naziv signala | Opis signala |
| s\_axi\_clk | Sinhronizacioni signal AXI interfejsa |
| s\_axi\_aresetn | Reset signal AXI interfejsa |
|  | Adresni kanal za upis |
| s\_axi\_awaddr | Početna adresa bloka od koje će biti upisivani podaci |
| s\_axi\_awprot | Vrsta pristupa. Retko se koristi |
| s\_axi\_awvalid | Indikacija od validnosti adrese. Kada je visok, trenutna adresa je validna |
| s\_axi\_awready | Indikacija da je AXI **slave** spreman da prihvati adresu i ostale kontrolne signale iz kanala |
|  | Kanal upisanih podataka |
| s\_axi\_wdata | Podatak koji treba upisati |
| s\_axi\_wstrb | Signal validnosti pojedinačnih bajtova unutar podatka koji se upisuje. Svaki bit je koji je postavljen znači da je njemu pridužen bajt u podatku validan |
| s\_axi\_wvalid | Indikacija validnosti podatka koji se upisuje. Kada je visok trenutni podatak je validan |
| s\_axi\_wready | Indikacija da AXI **slave** spreman da prihvati podatak |
|  | Statusni kanal upisa |
| s\_axi\_bresp | Status kompletirane transakcije upisa |
| s\_axi\_bvalid | Indikacija da je podatak u statusu transakcije validan |
| s\_axi\_bready | Indikacija da je AXI **master** spreman da prihvati podatak o statusu |
|  | Adresni kanal za čitanje |
| s\_axi\_araddr | Početna adresa bloka od koje će biti čitani podaci |
| s\_axi\_arprot | Vrsta pristupa. Retko se koristi |
| s\_axi\_arvalid | Indikacija validnosti adrese |
| s\_axi\_arready | Indikacija da je AXI **slave** spreman da prihvati adresui ostale kontrolne signale iz kanala |
|  | Kanal pročitanih podataka |
| s\_axi\_rdata | Podatak koji je pročitan |
| s\_axi\_rresp | Status kompletirane transakcije čitanja |
| s\_axi\_rvalid | Indikacija validnosti podatka koji se čita. Kada je visok trenutni podatak je validan |
| s\_axi\_rready | Indikacija da je AXI **master** spreman za prihvati podatak |

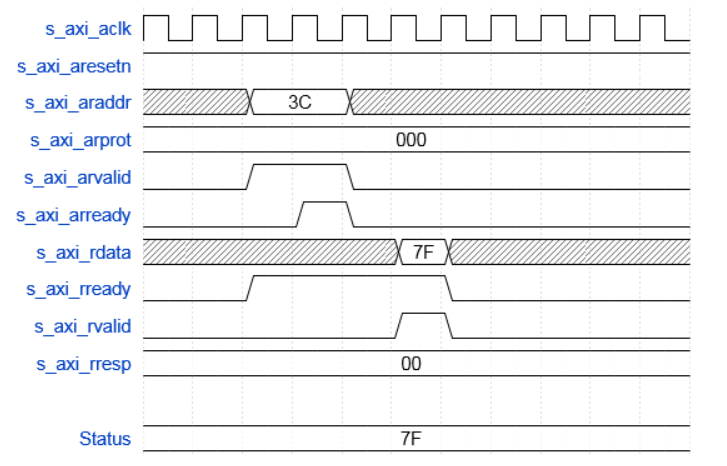
U nastavku će biti prikazani i komentarisani tipični talasni oblici signala AXI-*Lite* interfejsa u slučaju transakcija upisa i čitanja podataka.



Slika 2. Transakcija upisa podataka korišćenjem AXI-Lite interfejsa

Na slici 2 prikazani su talasni oblici relevantnih signala slejv AXI-*Lite* protokola hipotetičkog IP jezgra, u slučaju transakcije upisa podatka. Transakcija upisa podatka započinje tako što AXI master postavlja *s\_axi\_awaddr* magistralu na vrednost adrese lokacije unutar IP jezgra na koje želi upisati podatak. U primeru sa slike 2, adresa lokacije na koju se želi upisati podatak je 0x38. Istovremeno AXI master postavlja i signal *s\_axi\_awvalid* na jedinicu, signalizirajući IP jezgru da je nova validna adresa postavljena na *s\_axi\_awaddr* magistrali. Kada je IP jezgro spremno da prihvati adresu, postavlja *s\_axi\_awready* signal na jedinicu. U sledećem taktu AXI master postavlja *s\_axi\_awvalid* signal na nulu i uklanja adresu sa *s\_axi\_awaddr* magistrale. Istovremeno, ili i ranije kao što je slučaj na slici 2, AXI master postavlja podatak koji želi da upiše na adresiranu lokaciju unutar IP jezgra na *s\_axi\_wdata* magistralu (na slici 2 reč je o vrednosti 0x40). Takođe postavlja i indikaciju koji bajtovi unutar podatka su validni i trebaju da se upišu u adresiranu lokaciju, postavljajući odgovarajuće bitove *s\_axi\_wstrb* magistrale na jedinicu. AXI master postavlja i *s\_axi\_wvalid* signal na jedinicu, kako bi signalizirao IP jezgru da je na *s\_axi\_wdata* magistrali validni podatak. Kada je IP jezgro spremno da upiše podatak u adresiranu lokaciju, postalja *s\_axi\_wready* signal na jedinicu, i istovremeno upisuje podatak (na slici 6.7 podatak 0x40 se upisuje u hipotetički *Device* registar, jer se on nalazi na adresi 0x38). Nakon detekcije visokog

nivoa na *s\_axi\_wready* signalu, AXI master uklanja podatke sa *s\_axi\_wdata* i *s\_axi\_wstrb* magistrala, i postavlja *s\_axi\_wvalid* signal na nulu. Istovremeno i IP jezgro postavlja signal *s\_axi\_wready* na nulu. Poslednji deo transakcije upisa podatka predstavlja indikacija o finalnom statusu transakcije. Ovu informaciju generiše IP jezgro, preko statusnog kanala upisa. IP jezgro postavlja potrebni kod statusa na *s\_axi\_bresp* magistralu, istovemeno postavljajući i *s\_axi\_bvalid* signal na jedinicu i zatim čeka da *s\_axi\_bready* signal postane visok. AXI master signalizira da je spreman za prihvatanje finalnog statusa transakcije upisa, postavljajući *s\_axi\_bready* signal na jedinicu. Kada, nakon postavljanja validnog finalnog statusa transakcije upisa, IP jezgro detektuje da je *s\_axi\_bready* signal visok, u narednom taktu uklanja ove informacije sa statusnog kanala upisa, na taj način kompletirajući AXI-*Lite* transakciju upisa podatka.



Slika 3. Transakcija čitana podataka korišćenjem AXI-Lite interfejsa

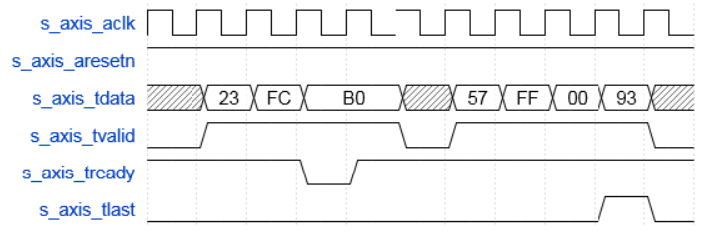
Na slici 3 prikazani su talasni oblici relevantnih signala slejv AXI-*Lite* protokola hipotetičkog IP jezgra, u slučaju transakcije čitanja podatka. Transakcija čitanja podatka započinje tako što AXI master postavlja *s\_axi\_araddr* magistralu na vrednost adrese lokacije unutar IP jezgra sa koje želi pročitati podatak. U primeru sa slike 3, adresa lokacije sa koje se želi pročitati podatak je 0x3C. Istovremeno AXI master postavlja i signal *s\_axi\_arvalid* na jedinicu, signalizirajući IP jezgru da je nova validna adresa postavljena na *s\_axi\_araddr* magistrali. Kada je IP jezgro spremno da prihvati adresu, postavlja *s\_axi\_arready* signal na jedinicu. U sledećem taktu AXI master postavlja *s\_axi\_arvalid* signal na nulu i uklanja adresu sa *s\_axi\_araddr* magistrale. Istovremeno, ili i ranije kao što je slučaj na slici 3, AXI master postavlja i *s\_axi\_rready* signal na jedinicu, kako bi signalizirao IP jezgru da je spreman za prihvat podatka. Kada je IP jezgro spremno da prosledi pročitani podatak sa adresirane lokacije AXI masteru, postavlja ga na *s\_axi\_rdata* magistralu (na slici 3 pročitani podatak predstavlja sadržaj hipotetičkog *Status* registra, i ima vrednost 0x7F). Istovremeno IP jezgro postavlja i *s\_axi\_rvalid* signal na jedinicu, signalizirajući AXI masteru da se na *s\_axi\_rdata* magistrali nalazi validan podatak. Nakon detekcije visokog nivoa na *s\_axi\_rready* signalu, IP jezgro uklanja podatke sa *s\_axi\_rdata* magistrale i postavlja *s\_axi\_rvalid* signal na nulu. Ovim je kompletirana AXI-*Lite* transakcija čitanja podatka.

**AXI4*-Stream*** – ova varijanta AXI4 protokola ne sadrži adresnu fazu prilikom transfera podataka. Koristi se u aplikacijama gde koncept adrese nije prisutan ili potreban (kao što su *dataflow* DSP aplikacije: kompresija zvuka, slike, videa, filtriranje, enkripcija, itd.). Kako nema adresne faze, ovaj interfejs je vrlo jednostavan i sastoji se iz minimalnog broja signala, ali se ne može koristiti za komunikaciju sa memorijski mapiranim modulima. Sa druge strane ne postoji nikakvo ograničenje na broj podataka koji se mogu preneti pomoću jedne transakcije. Ova vrsta interfejsa je unidirekciona, što znači da se podaci mogu prenositi samo u jednom smeru, od mastera ka slejvu. Ukoliko je potrebno ostvariti bidirekcioni prenos podataka, moraju se koristiti dva AXI4-*Stream* interfejsa i svaki od učesnika u komunikaciji mora implementirati i logiku AXI4-*Stream* mastera kao i logiku AXI4*-Stream* slejva. Tabela signala AXI4-stream masterinterfesja je:

*Tabela 3. Signali AXI-Stream interfejsa u slučaju AXI master komponente*

|  |  |
| --- | --- |
| Naziv signala | Opis signala |
| m\_axis\_aclk | Sinhronizacioni signal AXI interfejsa |
| m\_axis\_aresetn | Reset signal axi interfejsa |
| m\_axis\_tdata | Magistrala podataka |
| m\_axis\_tvalid | Indikacija da je tekući podatak validan |
| m\_axis\_tready | Indikacija da je AXI slejv spreman da primi sledeći podatak |
| m\_axis\_tlast | Indikacija da je tekući podatak ujedno i poslednji koji je potrebno preneti |

Primer Prenosa bloka podataka korišćenjem AXI4-stream interfejsa je prikazan u nastavku:

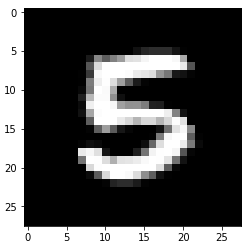
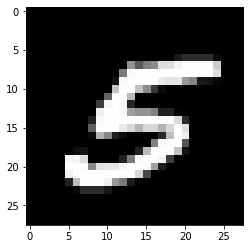


Slika 4. Prenos bloka podatka korišćenjem AXI-Stream interfejsa

Proces prenosa započinje kada AXI-*Stream* master postavi prvi podatak na *m\_axis\_tdata* magistralu i podigne *m\_axis\_tvalid* signal na jedinicu. Postavljeni podatak ostaje na *m\_axis\_tdata* magistrali sve dok AXI-*Stream* slejv modul ne signalizira AXI-*Stream* masteru da je spreman za prihvat sledećeg podatka, postavljajući *m\_axis\_tready* signal na jedinicu, što se lepo može videti na slici 6.9. Kada AXI-*Stream* master uoči da je AXI-*Stream* slejv spreman za prijem sledećeg podatka, on u sledećem taktu na *m\_axis\_tdata* magistralu postavlja sledeći podatak. Ukoliko iz nekog razloga AXI-*Stream* master nije u stanju da postavi sledeći podatak na *m\_axis\_tdata* magistralu, on ovu činjenicu signalizira AXI-*Stream* slejvu spuštanjem *m\_axis\_tvalid* podatka. Opisani postupak prenosa podataka nastavlja se dok se ne dodje do poslednjeg podatka koji je potrebno preneti. Kada AXI-*Stream* master postavi poslednji podatak na *m\_axis\_tdata* magistralu i označi ga kao validan, postavljajući *m\_axis\_tvalid* signal na jedinicu, istovremeno postavlja i *m\_axis\_tlast* signal na jedinicu, signalizirajući da je reč o poslednjem podatku iz tekućeg paketa. Nakon što AXI-*Stream* slejv preuzme i ovaj podatak, postavljajući *m\_axis\_tready* signal na jedinicu, tekuća transakcija prenosa bloka podataka je završena.

## 2.2 Teorijska analiza modula za ispravljanje slike (*eng. deskew module*)

Kao što je prethodno pomenuto, uloga deskju modula je preprocesiranje slika radi poboljšanja rezultata klasifikacije. Slike koje deskju modul obrađuje su crno bele slike na kojima se nalaze brojevi, i obrada se zasniva na tome da ako je broj na slici krivo napisan bude ispravljen.Primer rada deskju modula ilustrovan je sledećim primerom:



*Slika 5.* Pre preprocesiranja *6*. Nakon preprocesiranja

Na slici 5 prikazan je broj pre procesiranja slika od strane deskju modula, dok je na slici 6 prikazan broj nakon procesiranja.

Obrada koju modul vrši odvija se u 3 faze. U prvoj fazi se vrši proračun prostornih momenata slike prateći sledeću generalnu matematičku formulu za pronalaženje prostornih momenata proizvoljnog reda:

(1)

Formula vrši sumiranje prateći x i y koordinate slike. I(x,y) predstavlja vrednost piksela u datoj kordinati. U zavisnosti od vrednosti parametara „i“ i „j“ mogu da se računaju momenti prvog, drugor, trećeg itd. reda. Na primer ako se uzme da je i=0, i da je j=1, tada se računa moment prvog reda, odnosno posmatra se zakrivljenost slike u pravcu *y* ose. Za potrebe obrade slike (ispravljanje brojeva) koja je neophodna kako bi se pospešila klasifikacija neophnodni su prostorni momenti nultog i prvog reda, te se prethodna formula uopštuje:

Moment nultog reda:

(2)

Momenti prvog reda:

Moment nultog reda računa težinu slike, odnosno sumira vrednosti svih piksela slike i neophodan je u narednoj fazi. Momenti prvog reda korisni su u slučaju da je broj na slici centriran, i u toj situaciji bi oni bili sasvim dovoljni da se izvrši ispravljanje, ali pošto to često nije slučaj neophodno je izračunati centralne momente, i oni se računaju u drugoj fazi.

Druga faza predstavlja proračun centralnih momenata. Generalna formula za njihov proračun je:

Formula za računanje centralnih momenata direktno zavisi od proračuna u fazi jedan, odnosno od prostornih momenata nultot i prvog reda, odnosno oni su neophodni kako bi se izračunali i . Parametri i predstavljaju sumu svih piksela po x odnosno po y (m10, m01) osi podeljenu sa ukupnom težinom slike m00. Centralni momenti koji su dali najbolje rezultati prilikom ispravljanja slike (eng. deskewing) slike su mu11 i mu02 te su oni računati u algoritmu.

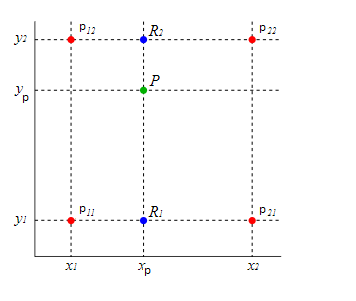
Uopštenje prethodne formule za proračun centralnih momenata mu11 i mu02 je:

Na osnovu proračunatih centralnih momenata računa se transformaciona matrica dimenzije 2x3:

U trećoj i konačnoj fazi vrši se ispravljanje slike na osnovu proračuna iz prethodnih faza. Na svaki piksel slike primenjuje se transformaciona matrica, i to na sledeći način:

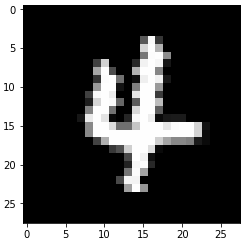
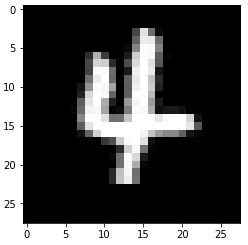
,

Ovom transformacijom se vrši promena vrednosti piksela i to sa vrednošću piksela na kordinatama (xp, yp). Ali pre nego što se izvrši promena vrednosti, neophodno je izvršiti interpolaciju piksela na kordinati (xp, yp). Pod interpolacijom se podrazumeva promena vrednosti piksela u zavisnosti od vrednosti okolnih piksela, i realizuje se na sledeći način:



Slika 7. Interpolacija

Sledeći primer ilustruje efekat interpolacija:

Slika 8. Bez interpolacije slika 9. Sa interpolacijom

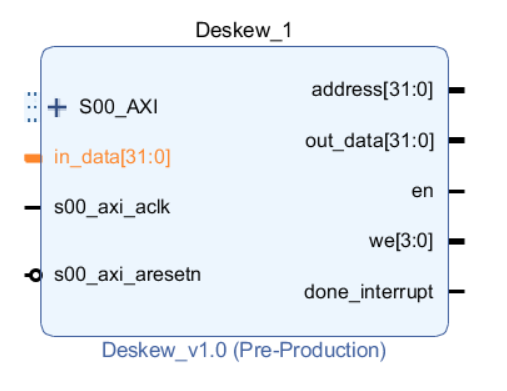
Efekat interpolacije je očigledan i na osnovu rezultata koji se vidi na prethodnom primeru može se reći da je interpolacija u stvari razmrljavanje slike. U slučaju da nije urađena interpolacija postojala bi šansa da slika bude loše klasifikovana kada se prosledi SVM klasifikatoru.

## 2.3 Specifikacija modula za ispravljanje slike (*eng. deskew module*) nakon hardverske implementacije

Slike koje modul obrađuje predstavljene su preko 784 piksela u formatu 28x28. Vrednost svakog piksela se nalazi u opsegu 0 – 1, gde 0 predstavlja najtamniju vrednost piksela a 1 najsvetliju. Kako bi mogla da se izvrši hardverska implementacija izvršena je kvantizacija vrednosti piksela, i njihove vrednosti su predstavljene preko 16 bita, gde je donjih 14 bita predstavlja razlomljeni deo broja, a gornja 2 celobrojni.

Kako bi *deskew* modul mogao da obradi sliku, ona mu mora biti data na raspolaganje, odosno slika mora biti smeštena u BRAM memoriju kojoj deskju modul ima pristup. Slika se smešta na prve 784 lokacije (jedna lokacija jedan piksel) BRAM memorije veličine 2048 lokacija širine 32 bita, dok na lokacije od 784 do 1567 modul smešta obrađenu sliku. Kada je slika smeštena u memoriju deskju modulu se šalje signal, i on započinje obradu. Proces čitanja slike iz memorije se odvija u 3 faze kako bi se izvršili proračuni navedeni u 2.2. U prvoj fazi deskju modul računa prostorne momente slike, a kako bi ih izračunao neophodno mu je da isčitava piksel po piksel prateći vertikalnu osu slike idevši odozgo na dole i sa leva na desno. Odnosno redosled kojim se izvlače pikseli iz BRAM memorije kako bi se to postiglo je: 0, 28, 56, ....755; 1, 29, 57, 756; 2, 30, 58, .....757; ...........; 27, 55, 83, .... 783;. U drugoj fazi računaju se centralni momenti, i opet je neophodno isčitavati piksele iz BRAM memorije istim redosledom kao u prvoj fazi. U trećoj fazi vrši se ispravljanje slike (*eng. deskewing*) na osnovu proračuna izvršenih u prve dve faze. Tada se pikseli poslednji put isčitavaju iz memorije istim redosledom kao u prve dve faze, stim što sada piksel koji se izvuče obradi, dobije novu vrednost i smesti na istu lokaciju u memoriji samo pomerenu za 784 mesta (neobrađeni piksel koji se nalazi na nultoj lokaciji u memoriji, nakon obrade se smešta na 784. lokaciju).

Interfejs preko koga deskju modul komunicira sa svojom okolinom (BRAM memorijom i procesorom) prikazan je na sledećoj slici:



*Slika 4*

**S00\_axi** predstavlja standardni AXI **lite** interfejs. Preko njega se može pristupiti untrašnjim registrima deskju modula. Upisivanjem logičke jedinice u *start* registar koji se nalazi na adresi 0 šalje se signal modulu da započne sa obradom slike, dok se isčitavanjem *ready* registra koji se nalazi na adresi 4 dobija uvid u to da li je modul završio sa obradom prehodne slike.

**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.

**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 poodatak 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.

## 2.3 Teorijska analiza SVM 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 (pomoćni vektori) na kojima je treniran i njima pridružene težine (*lambde*) i znak (*target*). Realizacija SVM klasifikatora prati sledeću formulu:

U prethodnoj formuli Y predstavlja podatak koji se klasifikuje, predstavlja matricu pomoćnih vektora koji su dobije prilikom treniranja mreže(svaki red u matrici predstavlja jedan pomoćni vektor), odnosno to su podataci isti kao i Y stim što su oni korišćeni tokom treniranja mreže i neophodni su kako bi se uspešno klasifikovao Y. Tokom klasifikacije je neophodno znati da li sami pomoćni vektori pripadaju datoj klasi ili ne, i tu informaciju u prethodnoj formuli daje *ti.* Ako je ti jednako 1 to znači da pomoćni vektor i pripada datoj klasi, dok ako je ti jednako -1, to znaći da pomoćni vektor i ne pripada klasi. Takođe prilikom treniranja mreže dobijaju se lagranžovi multiplikatori (λ) pri čemu svaki lagranžov multiplikator ima sebi pridružen pomoćni vektor (λi  ima sebi pridruzen i). Parametar bj predstavlja ofset (bias) koji se isto dobija prilikom treniranja mreže.

Kao što se iz date formule vidi neophodno je izvršiti onoliko sumiranja koliko ima pomoćnih vektora. U svakoj iteraciji sumiranja mora se izvršiti matrično množenje podatka koji se klasifikuje (Y) i pomoćnog vektora (Xi), dizanje na kub dobijene vrednosti i na kraju množenje svega to sa lagranžovim multiplikatorim (λi) i *targeta* (ti). Rezultat (C) prethodne sume je realan broj koji je ili pozitivan ili negativan. Ako je broj pozitivan onda dati podatak vrlo verovatno pripada datoj klasi, a ako je negativan onda vrlo verovatno ne pripada klasi. Što je broj slika na kojima se mreža trenira veći to je veći i broj pomoćnih vektora. Klasifikator je tada precizniji, ali količina proračuna koji mora da se izvrši se povećava.

Mana prethodnog klasifikatora je ta što on može da odredi samo da li dati podatak pripada ili ne pripada jednoj klasi, odnosno ako postoji više klasa kojima podatak može da pripada, tada ovaj klasifikator podbacuje. Primer toga je tema ovog rada, odnosno kada je neophodno klasifikovati koji je broj na slici, tada broj može da pripada jednoj od 10 klasa. Način da se reši ovaj problem je da se napravi onoliko klasifikatora koliko ima klasa kojima podatak može da pripada. Svaki klasifikator u tom slučaju mora da ima svoj set pomoćnih vektora, lagranžovih multiplikatora ,targeta kao i svoj specifični ofset *(eng.bias),* odnosno neophodno je izvršiti onoliko različitih treniranja koliko ima klasifikatora. Za primer klasifikacije brojeva neophodno je imati 10 različitih klasifikatora, 10 ofseta(*eng.bias*),10 setova pomoćnih vektora, lagranžovih multiplikaotora i *targeta.* Tada se prethodna formula pretvara u :

Sada je neophodno izvršiti 10 sumiranja odnosno klasifikacija (j = 0,1,...,9 predstavlja klasifikatore za brojeve 0,1,...9). Nakon što se sve klasifikacije izvrše i dobije se 10 različitih rezultata za C, klasa kojoj podatak Y pripada jeste ona kod koje je rezultat C najpozitivniji. Odnosno ako je C5 najpozitivnije, tada je broj na slici koja se klasifikuje najverovatnije broj 5.

## 2.4 Specifikacija SVM modula nakon hardverske implementacije

Slike koje modul klasifikuje predstavljene su preko 784 piksela u formatu 28x28. Vrednost svakog piksela se nalazi u opsegu 0 – 1, gde 0 predstavlja najtamniju vrednost piksela a 1 najsvetliju. Kako bi mogla da se izvrši hardverska implementacija izvršena je kvantizacija vrednosti piksela slika koje se klasifikuju, i njihove vrednosti su predstavljene preko 16 bita, gde je donjih 14 bita predstavlja razlomljeni deo broja, a gornja 2 celobrojni. Ista vrsta kvantizacije je izvršena je i za pomoćne vektore, lagranžove multiplikatore, *targete i* ofsete. Takođe radi lakše implementacije *target-i* i lagranžovi multiplikatori su spojeni u jedan parametar nazvan *lambda.*

Hardverska implementacije je izvršena tako da nakon što modul dobije sliku koju treba da klasifikuje, on izvršava 10 proračuna, pri čemu svaki proračun predstavlja jednu klasifikaciju koja daje određeni rezultat (Cj u 2.3). Nakon svakog izvršenog proračuna, taj proračun se poredi sa prethodnim, i ako je pozitivniji zamenjuje ga i smešta se u registar. Kada se svi proračuni izvrše, redni broj najpozitvnijeg rezultata je najverovatnije broj koji se nalazi na slici koja se klasifikuje. Slika koje se klasifikuje dobija se preko *fifo* kanala i smešta se u unutrašnju BRAM memoriju, i nakon toga započinje se sa prvim od deset proračuna. Prilikom izvršavanja ovih proračuna modulu se preko istog fifo kanala preko kog je dobio sliku dovode pomoćni vektori i

# 3. Verifikacioni plan

# 4. Verifikaciono okruženje

## 4.1 Opis UVM metodologije

## 4.2 Opis verifikacionog okruženja

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”

### 4.2.1 “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 “in\_data” port modula, čime ga modul dobija na korišćenje (odnosno podatak je isčitan iz memorije). Ova komunikacija između drajvera i sekvence se obavlja preko “bram sequencer” komponente, čija je uloga da sonhronizuje komunikaciju. 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“ komponentu, 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 u niz pod nazivom „ images array“ (svaka slika je predstavljena pmoću 784 piksela u formatu 28x28). Broj slika koji je izvučen iz tog falja iznosi 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 šta 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 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 natavlja 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 postavlja 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 tim se završava i simulacija.

### 4.2.2 “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 modul opet se koristi način rada deskju (*eng. deskew*) modula. Odnosno monitor posmatra trenutke kada modul postavlja “we” port na vrednost 0011, čime modul 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.

### 4.2.3 “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 modul 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.

### 4.2.4“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

### 4.2.5 “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.

### 4.2.6 “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).

### 4.2.7“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.

### 4.2.8“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*).

### 4.2.9 “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.

### 4.2.10“environment” i “configuration”

Komponenta “enviorment” instancira i enkapsulira prethodno opisane komponente, ali samo one koje su neophodne kako bi se 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.

### 4.2.11“Test”

“Test” objekat instancira I enkapsulira “environment” komponentu i pokreće sekvence na određenom sekvenceru. Koju sekvencu pušta zavisi od toga koji modul 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.

## 4.3 Konfigurisanje verifikacionog okruženja

# 5. Otklanjanje problema (*eng. debug*) verifikacionog okruženja i hdl dizajna

# 6. Regresija, fabrikacija i analiza propuštenih bagova (*eng. escape analysis*)

# 7. Zaključak

# 8. Literatura

**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 modul 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 modul poštuje AXI lajt (*eng. Axi lite*) protokol ili ne. Na osnovu signala koji su dobijeni nakon simulacije vizuelno je zakljućeno da modul 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:

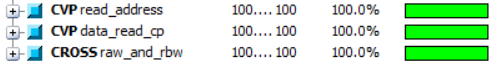
1. Da li se desilo upisivanje u sve unutrašnje register modula
2. Da li su se desila čitanja iz svih unutrašnjih registara modula
3. Da li se desilo čitanje posle upisa
4. Da li se desio upis posle čitanja
5. Da li se desio upis posle upisa
6. Da li se desilo čitanje posle čitanja

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

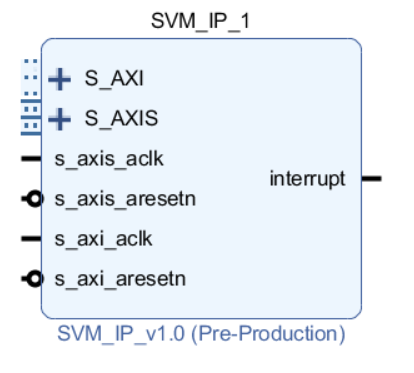
Na prethodnoj slici prvi CVP (*eng. Cover point*) govori o tome da li je tokom simulacije čitano iz svih unutrašnjih registara (druga stavka sa gornjeg spiska), drugi CVP govori o tome da li se menjala vrednost unutrašnjih registara, odnosno da li su se pojavile sve moguće vrednosti kod unutrašnjih registara, 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*).

Modul 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. Modul 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*). Modul 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 modul spreman za narednu klasifikaciju ili se čita rezultat klasifikacije, preko se drugog primaju podaci (slika koja se klasifikuje, pomoćni vektori itd.), dok preko trećeg modul 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 modul 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:

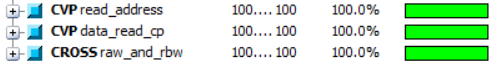
1. Provera funkcionalnosti reseta sistema. Ona je proverena na početku simulacije kada se ceo sistem resetuje.
2. Provera funkcionalnosti AXI lajt (*eng. Axi lite*) interfejsa. Ona se ogleda u proveri da li modul poštuje AXI lajt (*eng. Axi lite*) protokol ili ne. Na osnovu signala koji su dobijeni nakon simulacije vizuelno je zakljućeno da modul 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