Funkcionalna verifikacija MLP IP jezgra za prepoznavanje cifara

DIPLOMSKI RAD

Autor:
Marko Nikić EE86/2015

Mentor: dr Rastislav Struharik, redovni profesor

24.9.2020

Sadržaj

Spisak slika u tekstu

0 Zadatak projekta

- napraviti verifikacioni plan
- napraviti verifikaciono okruženje
- ispitati pokrivenost

1 Uvod

Tema ovog rada je funkcionalna verifikacija IP jezgra koje implementira algoritam višeslojnog perceptrona (eng. *Multilayer Perceptron*, skraćeno MLP) za prepoznavanje cifara. Rad je deo šireg poduhvata koji obuhvata i sistemsko projektovanje, hardversku implementaciju i osposobljavanje mikroračunarskog sistema koji će podržati ovaj hardver. Rad se sastoji iz sledećih celina:

- Prvog, uvodnog poglavlja
- Drugog poglavlja, u kom su predstavljene teorijske osnove korišćene metodologije.
- Trećeg poglavlja, u kom je izložena funkcionalna specifikacija jezgra koje se verifikuje.
- Četvrtog poglavlja, u kom je dat detaljan opis verifikacionog okruženja.
- Petog poglavlja, koje čini verifikacioni plan.
- Šestog poglavlja, gde su iskomentarisani rezultati verifikacione pokrivenosti.
- Sedmog, zaključnog poglavlja sa diskusijom o postignutim ciljevima.

2 Funkcionalna verifikacija hardverskog dizajna i njena uloga

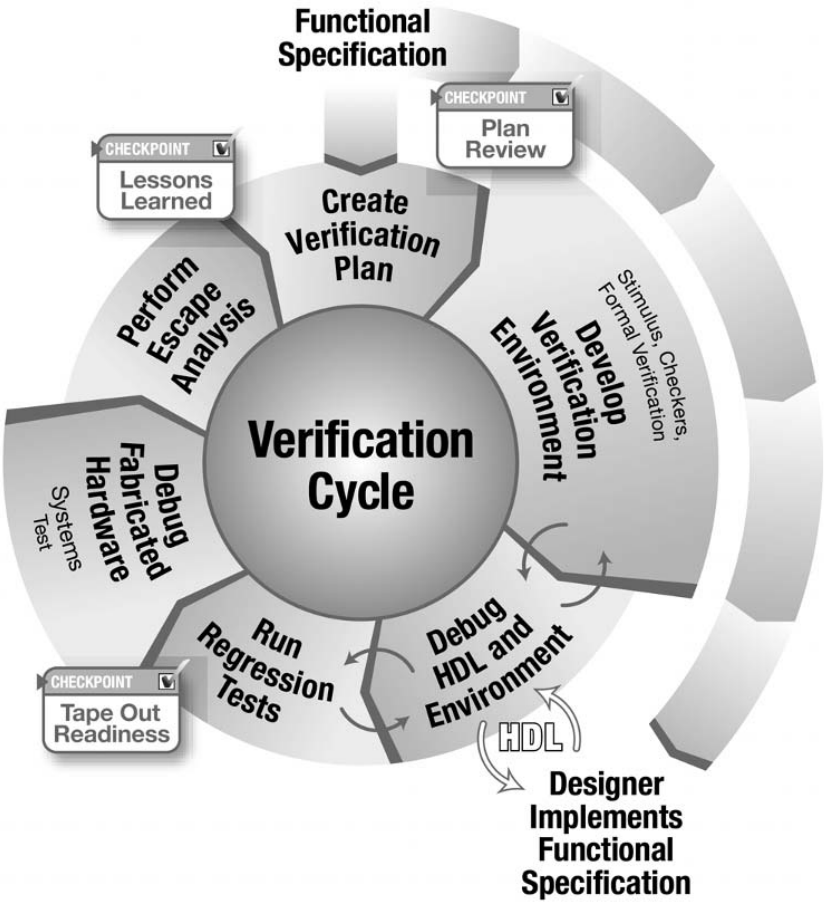
Cilj hardverskog dizajna je da napravi uređaj koji obavlja određenu funkciju na osnovu specifikacije dizajna, dok uloga funkcionalne verifikacije nije samo pronalaženje bagova već i garantovanje da dizajn obavlja svoju funkciji, odnosno funkcionalna verifikacija mora da potvrdi da je dizajn tačna reprezentacija specifikacije. Proces verifikacije je paralelan procesu pravljenja dizajna. Dizajner čita hardversku specifikaciju određenog bloka, interpretira taj opis i pravi odgovarajuću logiku na RTL nivou. Verifikacioni inženjer takođe čita hardversku specifikaciju i na osnovu

nje pravi testove kojima utvrđuje da li dizajn obavlja željenu funkcionalnost. Razlog za ovakvu praksu je 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. Proces verifikacije se može predstaviti verifikacionim ciklusom koji se sastoji iz sledećih koraka, predstavljenih i na slici 1 [3]:

- Funkcionalna specifikacija
- Pravljenje verifikacionog plana
- Razvijanje verifikacionog okruženja
- Debug HDL-a i okruženja
- Pokretanje regresije
- Debagovanje fabrikovanog hardvera
- *Escape* analiza

Verifikacioni ciklus se odvija u smeru kazaljke na satu, počevši od funkcionalne specifikacije, preko verifikacionog plana, sve do *escape* analize. Funkcionalna specifikacija opisuje željeni proizvod i nju zadaje sistemski arhitekta. Ona obuhvata specifikaciju interfejsa preko koga dizajn komunicira, funkcije koje mora da izvršava i uslove koji utiču na dizajn. Ključni deo verifikacionog ciklusa je verifikacioni plan jer on daje detaljan opis verifikacionih napora. On daje odgovor na pitanja "šta verifikujem?" i "kako ću 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 završetka verifikacionog plana započinje razvoj verifikacionog okruženja. Ono se sastoji iz softverskog koda, koji je uglavnom specifičan za dizajn, i alata, koji se koriste na više projekata. [3]

Naredni korak u verifikacionom ciklusu je debug dizajna i verifikacionog okruženja. U ovom koraku verifikacioni inženjer debaguje hardver pokretanjem testova definisanih u verifikacionom planu. Kako testovi odmiču, pronalaze se i analiziraju nove anomalije. Analiza otkriva uzrok anomalije koji može biti ili unutar okruženja ili unutar HDL dizajna. Ukoliko je anomalija u dizajnu, verifikacioni inženjer o tome obaveštava dizajn tim, čija je obaveza da grešku ispravi. Ukoliko je anomalija u verifikacionom okruženju, onda je verifikacioni inženjer zadužen za ispravljanje greške. Nakon završenog debaga sledi regresija, odnosno kontinualno puštanje testova definisanih u verifikacionom planu. 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 se



Slika 1: Verifikacioni ciklus

pojaviti neka anomalija. 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 *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 zašto bag nije uhvaćen u verifikacionom ciklusu. Ovo je jako bitno jer verifikacioni tim uči na svojim greškama kako u narednom verifikacionom ciklusu ne bi ponovio iste. [3]

Za verifikaciju MLP harverskog dizajna je odabrana univerzalna verifikaciona metodologija (eng. *Universal Verification Methodology*, UVM). UVM je standardizovana metodologija za funkcionalnu verifikaciju sa pomoćnom bibliotekom u SystemVerilog jeziku. Jedna od glavnih karakteristika ove metodologije je UVC (eng. *Universal Verification Component*), odnosno univerzalne verifikacione komponente koje imaju istu strukturu (sadrže monitore, drajvere, sekvencere,...). UVM obezbeđuje

framework za verifikaciju zasnovanu na pokrivenosti koja ne zahteva kreiranje velikog broja testova, osigurava temeljnu verifikaciju na osnovu zadatih parametara i olakšava proces pronalaženja problema. [1]

3 Funkcionalna specifikacija MLP jezgra

Višeslojni perceptron (eng. *Multilayer Perceptron*) je klasifikator u obliku višeslojne mreže neurona. Neuron je komponenta sa više ulaza i jednim izlazom, koja svaki od ulaza množi sa odgovarajućom težinom i na kraju sabira sve dobijene proizvode i pripadajući *bias*. Taj zbir se propagira kroz nelinearnu funkciju aktivacije. Izlaz funkcije aktivacije jeste izlaz neurona.

Neuroni u MLP-u su potpuno povezani (eng. *fully connected*), što znači da je izlaz svakog neurona iz jednog sloja povezan na ulaz svakog neurona iz narednog sloja. Dakle, na ulaz višeslojnog perceptrona se dovodi vektor koji predstavlja objekat koji se klasifikuje. Ovaj vektor se dalje množi i sabira sa unapred određenim koeficijentima i na izlazu mreže se dobija rezultat klasifikacije. Svaki od izlaznih neurona predstavlja jednu od kategorija klasifikacije: ako k-ti neuron u poslednjem sloju ima najveću izlaznu vrednost, ulazni vektor pripada k-toj kategoriji.

U našem slučaju MLP ima 3 sloja: ulazni, skriveni i izlazni. Funkcija aktivacije je *Leaky ReLu* i data je sledećom jednačinom

$$f(x) = \begin{cases} 0,001 \cdot x, & x < 0 \\ x, & x \geq 0. \end{cases} \tag{1}$$

Ulazni vektori su slike iz MNIST baze veličine 28x28 piksela, zbog čega ima 784 ulaznih neurona (odnosno postoje 784 komponente ulaznog vektora), a ulazi u neurone prvog sloja su vrednosti piksela od 0 do 1. U skrivenom sloju ima 30 neurona, a kako je u pitanju klasifikacija cifara od 0 do 9, u poslednjem, izlaznom sloju ima 10 neurona.

MLP IP core komunicira sa svojim okruženjem preko dva standardna AXI4 interfejsa: AXI *Lite* i AXI *Stream*. AXI *Lite* se koristi za čitanje statusnih i upisivanje u komandne registre, dok se AXI *Stream* koristi za slanje podataka potrebnih za izračunavanje (slika, težine i biasi). Za detalje o specifikaciji ovih interfejsa čitaoca upućujemo na [2].

Spremnost MLP jezgra za novu klasifikaciju dobijamo čitanjem stanja unutrašnjeg *ready* registra. Ukoliko je njegova vrednost 0, jezgro je zauzeto klasifikovanjem, a ako je njegova vrednost 1, jezgro je slobodno i može se započeti nova klasifikacija. Klasifikacija se započinje postavljanjem unutrašnjeg *start* registra na 1. Nakon toga je neophodno preko AXI *Stream* interfejsa poslati prvo 28x28 piksela slike koja se klasifikuje, a potom težine i *bias*-e za svaki neuron u MLP-u.

Slanje ovih podataka je potrebno izvršiti u tačno definisanom redosledu, kako bi jezgro znalo da ih protumači na pravilan način: prvo se šalje svih 784 piksela slike, a potom težine i *bias* za svaki neuron pojedinačno. Nakon što se klasifikacija završi, *ready* registar ima vrednost 1 i iz *cl_ num* registra se može pročitati klasifikovani broj.

Unutrašnji registri i njihove adrese su prikazani u tabeli 1.

Tabela 1: Registri MLP IP jezgra

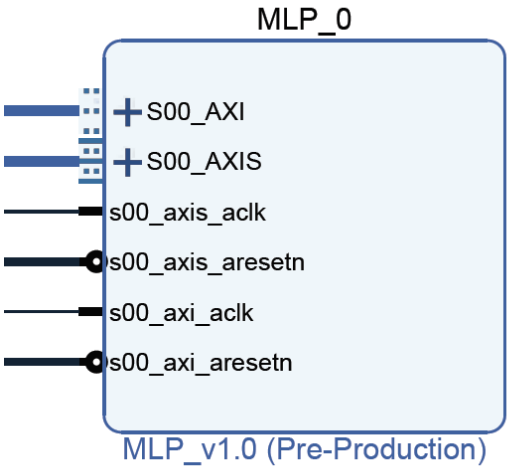
naziv registra	adresa pristupa AXIL interfejsom
start	0
ready	4
toggle	8
cl_num	12

Interfejs MLP jezgra je prikazan na slici 2. Sa slike uočavamo:

- S00_AXI - predstavlja standardni AXI *Lite* interfejs
- S00_AXIS - predstavlja standardni AXI *Stream* interfejs
- s00_axi_aclk - ulazni AXI *Lite* taktni signal
- s00_axis_aclk - ulazni AXI *Stream* taktni signal
- s00_axi_aresetn - ulazni AXI *Lite* reset signal
- s00_axis_aresetn - ulazni AXI *Stream* reset signal

4 Verifikaciono okruženje

Verifikaciono okruženje je bazirano na univerzalnoj verifikacionoj metodologiji (UVM) i sadrži sledeće komponente: test, environment, scoreboard, configuration i agente za AXI *Stream* i AXI *Lite* interfejse, njima pripadajuće monitore, drajvere i sekvencere,. Šematski prikaz verifikacionog okruženja je dat na slici 3. Svaka od komponenti predstavlja klasu koja nasleđuje odgovarajuću UVM klasu (nalaze se u UVM bilioteci) i opisane su pomoću *SystemVerilog* jezika.



Slika 2: MLP IP jezgro sa naznačenim interfejsima

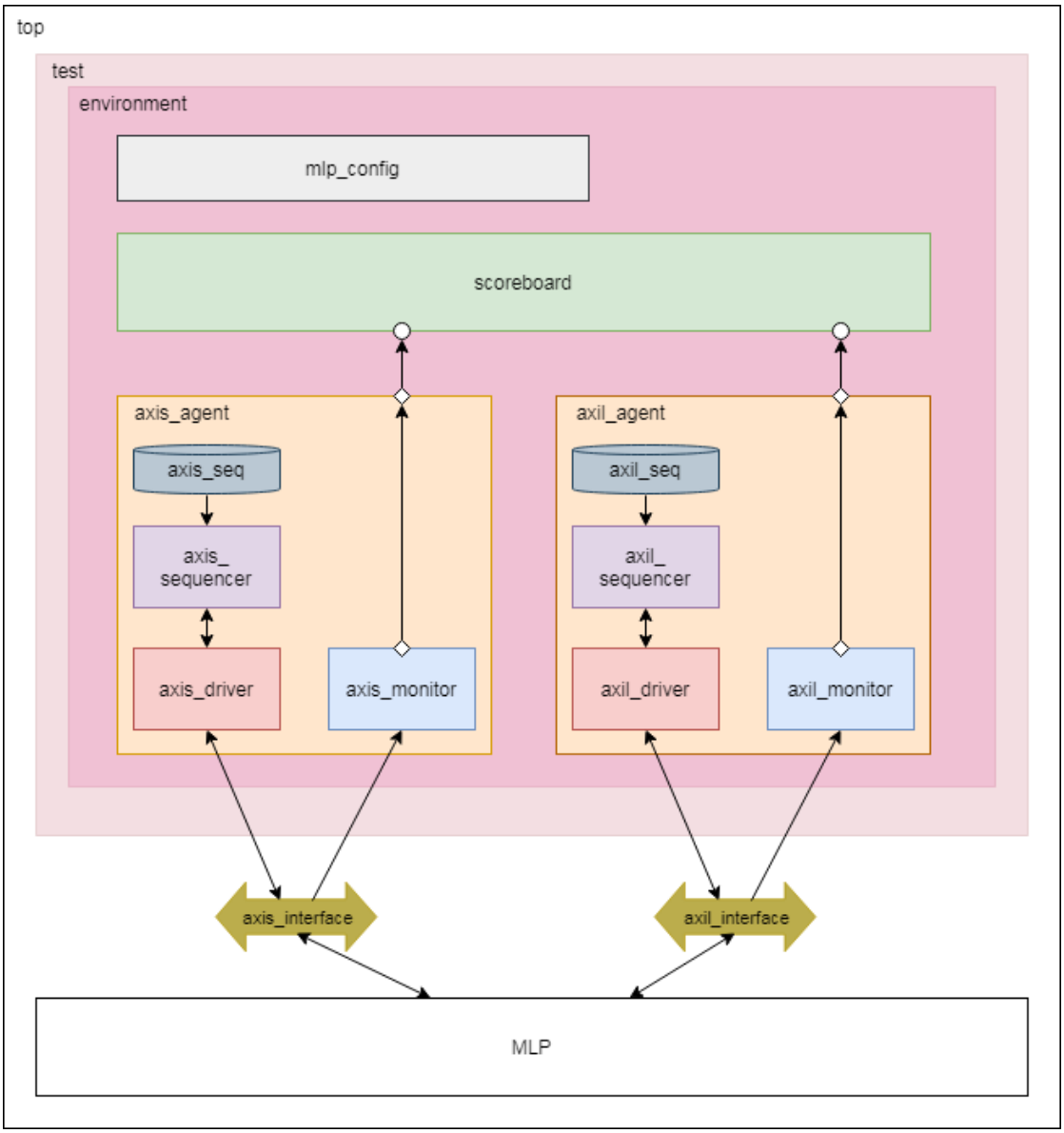
4.1 axil_driver, axil_sequence, axil_sequencer, axil_monitor, axil_agent

Preko AXI Lajt (*eng. AXI Lite*) interfejsa jezgro dobija komandu za početak klasifikacije i daje informaciju o tome da li je spremno za novu klasifikaciju, kao i rezultat klasifikacije.

Sekvencer ima ulogu stimulus generatora. U njemu se kontrolišu podaci koje će drajver slati dizajnu koji se verifikuje. Ova kontrola podataka se vrši u definisanim sekvencama koje određuju broj, sadržaj i redosled transakcija. [1] *Driver* komponenta prevodi podatke sa nivoa transakcije (iz sekvence) na nivo interfejsa. Prilikom postavljanja signala na interfejs drajver vodi računa da AXI *Lite* protokol bude ispoštovan. U slučaju da je u pitanju transakcija čitanja, drajver pročitan podatak sa interfejsa zapisuje nazad u transakciju (eng. *transaction item*).

Drajver u beskonačnoj petlji u *run* fazi koristi blokirajuću metodu *get_ next_ item* kako bi od sekvencera iz sekvence dobio transakciju na osnovu koje će postaviti signale na interfejs. Nakon što završi sa postavljanjem signala na interfejs, on javlja sekvenceru da je sa trenutnom transakcijom završio koristeći neblokirajuću metodu *item_ done*.

AXI *Lite* sekvenca je napisana tako da obezbedi pravilan redosled upisa i čitanja u unutrašnje registre MLP jezgra. Kôd sekvence se sastoji od *'uvm_ do_ with* makroa kojima je obezbeđeno da se drajveru prosleđuju transakcije u tačno definisanom redosledu: pročitaj da li je jezgro spremno za rad - pročitaj klasifikovan broj - započni novu klasifikaciju. Kôd se nalazi u petlji sa uslovom koji garan-



Slika 3: Verifikaciono okruženje MLP jezgra

tuje da se sa započinjanjem nove klasifikacije sačeka sve dok jezgro zaista ne bude spremno. *'uvm_ do_ with* makro implementira metode *start_ item*, randomizaciju sa ograničenjima i *finish_ item*. Sve ovo omogućava sinhronizaciju sekvence i rada drajvera. U sekvenci se pozove blokirajuća metoda *start_ item*, koja čeka da drajver zatraži sledeću transakciju. Nakon toga sekvenca randomizuje transakciju sa zadatim ograničenjima i poziva blokirajuću *finish_ item* metodu, obaveštavajući drajver da slobodno procesira randomizovanu transakciju.

Monitor nadgleda interfejs i uočene promene pakuje u transakciju koju šalje *scoreboard*-u. Monitor čeka da na interfejsu postanu aktivni odgovarajući AXI *ready* i *valid* signali. Kada oni dobiju vrednost logičke jedinice, monitor adresu i pročitan ili upisan podatak upisuje u transakciju, koju prosleđuje *scoreboard*-u. U monitoru je implementirano i prikupljanje pokrivenosti. Scenariji koje proveravamo za pokrivenost su:

- Čitanje iz svih internih registara
- Upis u sve interne registre
- Čitanje posle upisa
- Upis posle čitanja
- Upis posle upisa
- Čitanje posle čitanja

U ovu svrhu su kreirane odgovarajuće grupe pokrivenosti (eng. *covergroups*) i tačke pokrivenosti (eng. *coverpoints*).

Agent enkapsulira drajver, sekvencer i monitor u jednu komponentu koja komunicira sa ostatkom verifikacionog okruženja.

4.2 axis_driver, axis_sequence, axis_sequencer, axis_monitor, axis_agent

axis_ sequence sadrži više transakcija. Ove transakcije predstavljaju sliku za klasifikaciju i težine i bias za svaki neuron. Sekvenca obezbeđuje da se oni pošalju u tačno predefinisanom redosledu. Svaka transakcija ima polje koje predstavlja red podataka. U ovaj *dataQ* red prebacujemo potrebne podatke iz datih redova (*yQ, weightQ, biasQ*). Primetimo da ovde, iako se sekvenca kreira i pozivaju se metode *start_ item* i *finish_ item*, ipak izostaje randomizacija. Razlog tome je što nije bilo polja u transakciji koja bi bilo potrebno randomizovati. Jedini način da se dobije smisleni, usmeren test je da se jezgru šalju podaci smislenog sadržaja i

broja. Kako su *dataQ i len* jedina polja AXI *Stream* transakcije, nije preostalo ništa vredno randomizovanja.

Komponenta *axis_driver* postavlja signale na interfejs za svaki podatak iz transakcije, a kada stigne do poslednjeg podatka, aktivira i poseban signal *s_axis_tlast*. Svoj rad sa sekvencom sinhroniše na ranije opisan način.

Monitor nadgleda AXI *Stream* interfejs, sakuplja podatak jedan po jedan i skladišti ih u red. Kada uoči signal za poslednji podatak u transakciji, sakupljenu transakciju šalje komponenti *scoreboard*.

Agent gore navedene komponente okuplja u jednu.

4.3 scoreboard

Unutar ove komponente implementiran je referentni model MLP jezgra. Na osnovu podataka koje dobija od monitora, *scoreboard* vrši proračune i upoređuje svoj rezultat klasifikacije sa onim koje je dalo jezgro. Monitori nadgledaju interfejse i svaki put kada uoče smislenu transakciju, oni je prosleđuju scoreboardu preko TLM portova. Kad god transakcija pristigne na port koji je povezan sa monitorom, aktiviraće se njemu pripadajuća *write* funkcija.

Kada AXI *Lite* monitor pošalje transakciju, aktivira se *write_axil* funkcija. U njoj se proverava sadržaj transakcije na osnovu kog se saznaje stanje jezgra i rezultat klasifikacije. Ako sadržaj pristigle transakcije govori da je pročitana vrednost 1 iz ready registra, postavlja se promenljiva *res_ready* na 1, kako se prilikom sledećeg aktiviranja funkcije *write_axil* ne bi izgubila informacija da je jezgro spremno za rad. U suprotnom, proverava se da li je pročitan *cl_num* registar i da li je *res_ready* na 1 (što bi značilo da je pre ovoga već jednom pročitano da je *res_ready* na 1). Ako to jeste sličaj, klasifikovani broj se upoređuje sa rezultatom do kog je došao referentni model i prijavljuje se greška ukoliko je došlo do neslaganja.

write_axis funkcija sakuplja sve pristigle transakcije i tumači ih kao ulazne piksele, težine ili *bias*-e. I referentnom modelu je, kao i jezgru, bitno da se podaci šalju u predefinisanom redosledu (što je obezbeđeno u sekvenci) radi njihovog ispravnog tumačenja. Jednu AXI *Stream* transakciju čine ili pikseli slike ili težine za jedan neuron ili bias za jedan neuron. Monitor će svaki od nabrojanih slučaja tretirati kao zasebnu transakciju koju će poslati komponenti *scoreboard*, što će svaki put dovesti do aktiviranja *write_axis* funkcije. Zato je neophodno korišćenjem pomoćnih promenljivih i proverom brojnih uslova tačno odrediti koji podaci su već stigli, a koji trenutno stižu. Kada je i poslednji podatak primljen, *write_axis* poziva funkciju *calc_res* u kojoj je implementiran referentni model jezgra. Ova funkcija koristi isti algoritam kao i MLP IP *core*, a za računanje koristi prethodno pristigle podatke skladištene u odgovarajuće redove.

4.4 environment i test

Environment instancionira i enkapsulira sve prethodno opisane komponente. Test instancionira environment i pokreće sekvence na sekvencerima.

5 Verifikacioni plan

Verifikacija se vrši u sledećim etapama:

1. Provera funkcionalnosti reseta sistema. Proverena je resetovanjem celog sistema na početku simulacije. Kako je klasifikacija tekla po planu i nakon ovog reseta, zaključeno je da je ova funkcionalnost ispoštovana.

2. Provera funkcionalnosti AXI Lite interfejsa. Vizuelnim posmatranjem signala dobijenih tokom simulacije je zaključeno da je protokol ispoštovan.

3. Provera funkcionalnosti MLP jezgra. Scoreboard tokom simulacije nije prijavio nijednu grešku, što znači da nikada nije došlo do neslaganja rezultata referentnog modela i jezgra, te da jezgro obavlja željenu funkcionalnost.

6 Verifikaciona pokrivenost

Sakupljanje pokrivenosti implementirali smo u axil_monitor komponenti i proverili smo sledeće slučajeve:

- Čitanje iz svih internih registara. *Coverpoint* read_address proverava da li se desio upis na sve adrese od značaja.

- Upis u sve interne registre. *Coverpoint* write_address proverava da li se desilo čitanje sa svih adresa od značaja.

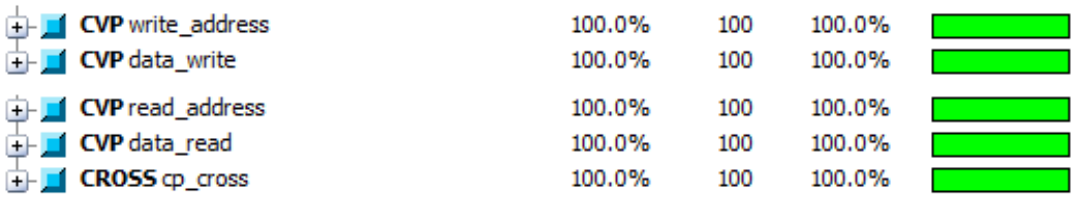
- Čitanje posle upisa.

- Upis posle čitanja.

- Upis posle upisa.

- Čitanje posle čitanja.

Coverpoint cp_cross proverava poslednja četiri slučaja. Rezultati analize pokrivenosti su prikazani na slici 4. Vidimo da je pokrivenost svih tačaka 100%, što znači da je test scenario dobro napisan i da je uspeo da izverifikuje funkcionalnost jezgra u željenoj meri.



Slika 4: Rezultati prikupljanja pokrivenosti

7 Zaključak

U ovom radu je pomoću UVM metodologije uspešno verifikovano IP jezgro koje implementira MLP algoritam za prepoznavanje cifara. Na osnovu simulacionih rezultata i rezultata prikupljanja pokrivenosti zaključeno je da projektovani hardver ispunjava zahteve funkcionalne specifikacije i da su napisani testovi u dovoljnoj meri izverifikovali postavljene zahteve. U verifikacionom okruženju nisu uočeni propusti. Komponente okruženja pisane su na sistematičan način, pottpuno nezavisne jedna od druge i od dizajna koji se verifikuje.

Potencijalna poboljšanja mogla bi biti u pisanju novih, raznovrsnijih testova.

Literatura

[1] Andrea Erdeljan. Materijal za vežbe iz predmeta Funkcionalna verifikacija hardvera - vežba 5, 2019.

[2] Rastislav Struharik. *Projektovanje složenih digitalnih sistema : računarske vežbe*. Fakultet tehničkih nauka, Novi Sad, 2017.

[3] Bruce Wile, John Goss C., and Wolfgang Roesner. *COMPREHENSIVE FUNCTIONAL VERIFICATION - THE COMPLETE INDUSTRY CYCLE*. Morgan Kaufmann, San Francisco, 2005.