Funkcionalna verifikacija hardvera
projekat

TEMA PROJEKTA:

Funkcionalna verifikacija Video Motion Detection sistema – ARPS algoritam

TEKST ZADATKA:

- napraviti verifikaciono okruženje
- ispitati pokrivenost

Mentor
Nikola Kovačević

Studenti
Stefan Stanković EE17/2015
Grigorije Vešić EE10/2015
Đorđe Savić EE122/2015
Nemanja Milošević EE220/2015

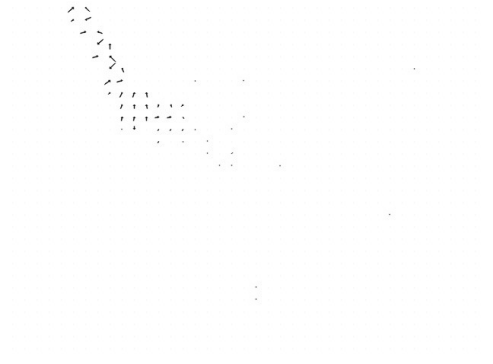
U Novom Sadu, 14. avgusta 2020. godine

1.Uvod

Tema projekta je funkcionalna verifikacija modula za video detekciju pokreta (*eng. Video Motion Detection system*). Funkcija ovog modula je da izračuna vektore pomeraja svakog bloka jednog frejma(trenutnog) u odnosu na njegov prethodni (referentni) susedni frejm pomoću prilagodljivog obrasca pretraživanja (*eng. Adaptive Rood Search Pattern*), kraće ARPS. Rezolucija frejmova za koje je modul predvdjen da radi je 256x256 piksela, dok je veličina blokova za koje se računa vektor pomeraja 16x16 piksela. Ilustracija rada modula može se videti na sledećem primeru. Na slikama 1.1 i 1.2 mogu se videti referentni i trenutni frejm izdvojeni iz video snimka "*golfer"*. Na slici 1.3 može se videti rezultat izvršavanja ARPS algoritma implementiranog u VMD modulu, što se ogleda u nalaženju vektora, dok se na slici 1.4 može videti rekonstruisan frejm na osnovu vektora pomeraja i referentnog frejma.

Slika 1.1 Referentni frejm

Slika 1.2 Trenutni frejm



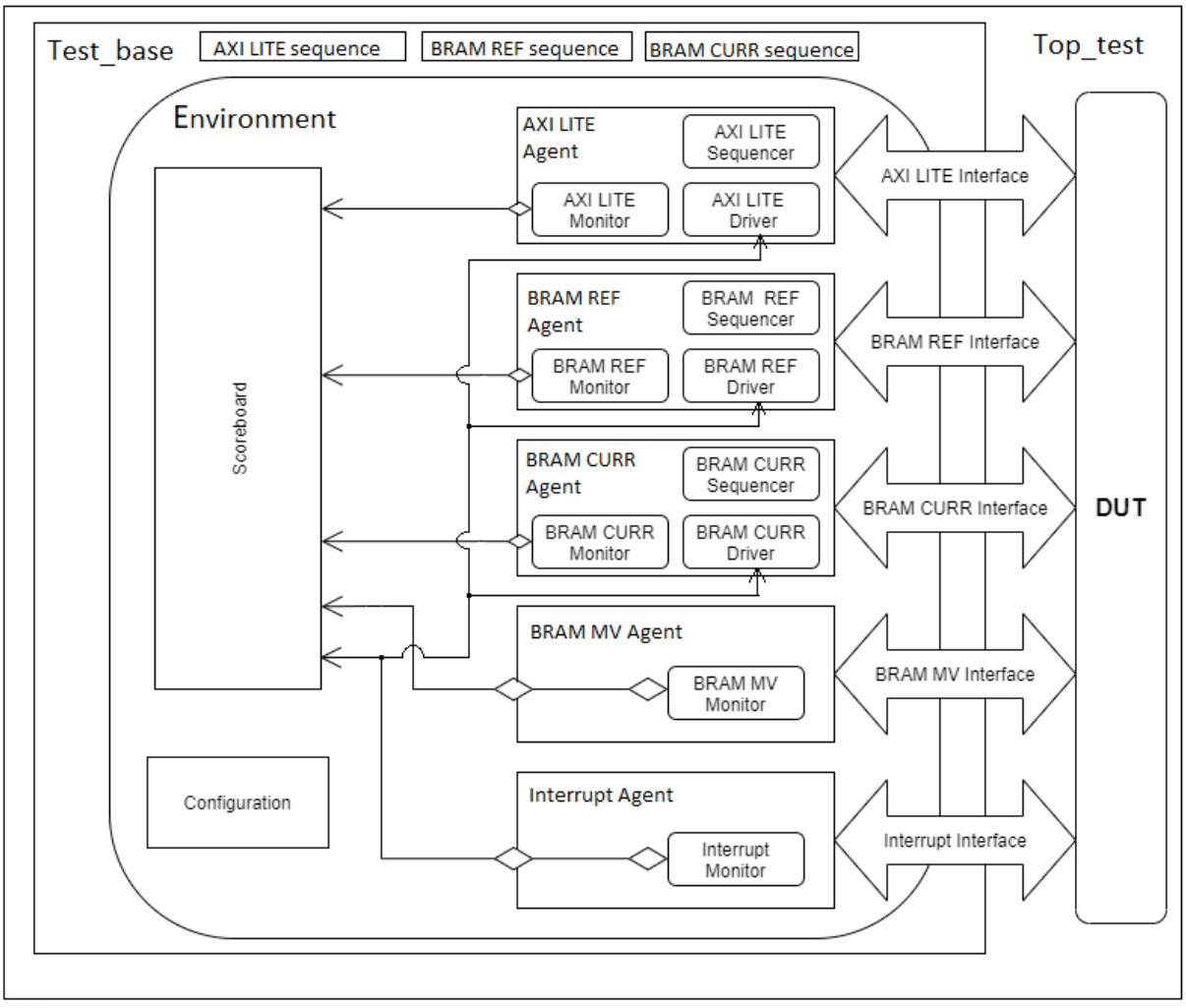
Slika 1.3 Vektori pomeraja

Slika 1.4 Rekonstruisan frejm

Metodologija koja se koristi kako bi se verifikovala funkcionalnost ovog modula je UVM (*eng. Universal Verification Methodology).* Jedna od glavnih karakteristika ove metodologije je korišćenje UVC (*eng. 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. Simulacija rada ovog modula u UVM okruženju je pokretana u "Questa Sim" simulatoru, čiji je proizvođač "Mentor graphics".

2. Verifikaciono okruženje

Verifikaciono okruženje kreirano za potrebe verifikacije „VMD“ modula je prikazano na sledećoj slici:



Slika 1.5 Verifikaciono okruženje

Verifikacioni okruženje prikazano na prethodnoj slici kao i svako okruženje bazirano na UVM metodologiji, sadrži sledeće komponente: “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 1.5 su:

1) “AXI_LITE_sequence”
2) “BRAM_REF_sequence”
3) “BRAM_CURR_sequence”
4) “AXI LITE Agent”
    - “ AXI LITE driver”
    - “ AXI LITE monitor”
    - “ AXI LITE sequencer”
5) “BRAM REF Agent”
    - “ BRAM REF driver”
    - “ BRAM REF monitor”
    - “ BRAM REF sequencer”
6) “ BRAM CURR Agent”
    - “ BRAM CURR driver”
    - “ BRAM CURR monitor”
    - “ BRAM CURR sequencer”
7) “ BRAM MV Agent”
    - “ BRAM MV monitor”
8) “Interrupt Agent”
    - “Interrupt monitor”
9) “Scoreboard”
10) “Configuration”
11) “Environment”
12) “Tests”

2.1 “AXI LITE sequence”, “ AXI LITE driver” , “ AXI LITE sequencer” i “ AXI LITE monitor”

Axi lajt (*eng. AXI LITE*) interfejs VMD modula omogućava da se preko njega pošalje komanda modulu da započne sa računanjem vektora pomeraja, tako što se pročita da li je modul spreman za obradu naredna dva frejma. 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 čitanje preko Axi lajt (*eng. AXI LITE*) interfejsa gde se čita vrednost registra na adresi **4'b0100** i ukoliko je

vrednost „1“ modul je spreman. Zatim se šalju i zahtevi za upis kada se u registar na adresi **4'b0000** upisuje prvo vrednost „1“, a zatim i vrednost „0“ kako bi se osigurao ispravan rad modula. Pri svakom tom zahtevu “axil driver” kompontenta vodi računa da protokol bude ispoštovan. 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.

“AXI LITE 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 registre 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

2.2 “BRAM REF sequence” ,“BRAM REF driver” i “BRAM REF sequencer”

Kako bi testirali funkcionalnost VMD modula neophodno je da jedan deo okruženja implementira BRAM memoriju, odnosno da kada modul ima zahtev za čitanje iz BRAM memorije neki deo okruženja odgovori na taj zahtev i prosledi odgovarajući podatak. Komponente zadužene za taj posao su “BRAM REF sequence”, “BRAM REF driver” i “BRAM REF sequencer**”**.

“BRAM REF sequence” se sastoji od više komponenti, prva nazvana “ARPS_IP_bram_ref_base_seq” služi za učitavanje „ .txt “ fajlova u koji su smešteni pikseli u heksadecimalnom zapisu. Pikseli su širine 8 bita i za potrebe modula grupisani su u 32-bitne vrednosti te se kao takve učitavaju u red (*eng. Queue*). Jedna slika koja je u „grejskejl“ (*eng. Greyscale*) formatu dimenzija 256*256 piksela, ukupno 65536, smešta se u red veličine 16384 lokacije širine 32 bita iz gore navedenog razloga gde se u jednom podatku nalaze 4 piksela. Drugi fajl se naziva “ARPS_IP_bram_ref_base_seq_2” i njegova razlika u odnosu na prethodni je ta što se u red upisuju nasumične vrednosti piksela umesto konkretnih vrednosti piksela koji čine sliku. Iz ovih fajlova izvedeni su fajlovi “ARPS_IP_bram_ref_simple_seq” i “ARPS_IP_bram_ref_simple_seq_2” čija će funkcionalnost biti opisana zajedno sa drajverom.

“BRAM REF driver” komponenta ima direktan pristup bram interfejsu modula. Nakon što je VMD modul startovan pomoću „AXI LITE agenta“, drajver prosleđuje sekvenci adresu lokacije sa koje bi VMD 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 bram_ref interfejs, čime ga modul dobija na korišćenje (odnosno podatak je iščitan iz memorije). Ova komunikacija između drajvera i sekvence se obavlja preko "BRAM REF sequencer" komponente koja je tu kako bi tu komunikaciju sinhronizovala. Način na koji se realizuje prethodno opisani postupak je:

```
1.virtual task body();
2.    ARPS_IP_def  def = new();
3.    num_of_seq = def.get_num_of_seq();
4.    start_frame_r = def.get_ref_frame_num();
5.    read_ref_img(start_frame_r);
6.req=ARPS_IP_bram_ref_transaction::type_id:
:create("req");
7.    forever begin
8.      if(cnt_seq < num_of_seq) begin
9.          `uvm_do(req)
10.         address_write_r = req.address_ref;
11.         if(req.interrupt)begin
12.           req.interrupt = 0;
13.           if(start_frame_r<4) begin //only seq_2
14.              start_frame_r++;
15.           end                              //only seq_2
16.           read_ref_img(start_frame_r);
17.           cnt_seq++;
18.       end
19.       else begin
20.         `uvm_do_with(req, {req.data_ref_frame
== ref_queue[address_write_r/4]; } )
21.       end
22.    end
23.    else begin
24.        break;
25.    end
26.  end // forever begin
27.endtask : body
```

```
1.task run_phase(uvm_phase phase);
2.    forever begin
3.        @(posedge vif.clk)begin
4.            if(interrupt_o == 1)begin
5.                interrupt_o = 0;
6.                seq_item_port.get_next_item(req);
7.                req.interrupt = 1;
8.                seq_item_port.item_done();
9.                continue;
10.           end
11.           address = vif.axi_address;
12.           seq_item_port.get_next_item(req);
13.           req.address_ref = address;
14.           seq_item_port.item_done();
15.           seq_item_port.get_next_item(req);
16.           vif.data_ref=req.data_ref_frame;
17.           seq_item_port.item_done();
18.       end
19.    end
20.endtask: run phase
```

Kod sa leve strane realizuje rad „BRAM REF sequnce" komponente, naime i "ARPS_IP_bram_ref_simple_seq" i "ARPS_IP_bram_ref_simple_seq_2" imaju istu funkcionalnost sa malom razlikom u linijama 13 i 15 koja će biti opisana u nastavku, dok kod sa desne strane realizuje rad „BRAM REF driver" komponente. Unutar bram sekvence se nalazi referentni frejm koji bi modul trebalo da poredi sa trenutnim frejmom smeštenim u "BRAM CURR agent" da bi na taj način računao vektor pomeraja. U drugoj liniji koda kreira se objekat klase „ARPS_IP_def" koji sadrži metode pomoću kojih se parametrima sekvence, „num_of_seq" - prosleđuje koliko poređenja frejmova treba da se izvrši i „start_frame_r" - redni broj frejma koji

se poredi. Frejm se učitava iz fajla u petoj (5.) liniji koda sa leve strane pozivom „read_ref_img()“ funkcije i smešta u niz pod nazivom „ ref_queue“. U narednoj liniji koda se pravi objekat koji se prosleđuje drajveru. Taj objekat je takođe klasa napisana u sistem verilog jeziku, i u toj klasi su definisana 3 polja : „address_ref“, „data_ref_frame“ i „interrupt“. Prilikom komunikacije između drajvera i sekvence, ovaj objekat se prosleđuje, i na osnovu polja unutar njega sekvenca i drajver vrše određene operacije. U 7. liniji koda kreće beskonačna for petlja i započinje se proces komunikacije između sekvence i drajvera. U 9. liniji koda se poziva blokirajući makro „uvm_do()“ koji zaustavlja izvršavanje sekvence dok ne dobije odgovor od drajvera. Drajver će odgovoriti sekvenci u onom trenutku kada VMD 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 poziva blokirajuću metodu „get_next_item(req)“ (linija 12) koja čeka da se pozove „uvm_do(req)“ makro (on je već pozvan 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_ref“ polje unutar „req“ objekta na vrednost adrese koju vidi na bram interfejsu modula (linija 13), i poziva metodu „item_done(req)“ (linija 14), čime vraća promenjeni objekat sekvenci. Nakon toga se kontrola vraća sekvenci koja nastavlja sa izvršavanjem. Zatim u 10. liniji koda sekvenca postavlja „address_write_r“ polje na određenu vrednost u zavisnoti od adrese koju je postavio drajver na „address_ref“ polju „req“ objekta. Nakon toga se poziva u 20. liniji koda „uvm_do_with()“ makro koj prosleđuje drajveru „req“ objekat u kome se nalazi podatak koji drajver treba da postavi na „data_ref“ port bram interfejsa. Makro u sebi sadrži dodelu vrednosti „data_ref_frame“ polju objekta „req“ iz reda „ref_queue“ sa pozcije koja je uslovljena adresom dobijenom prethodno opisanim koracima, takođe adresa se deli brojem 4 zbog načina rada VMD modula koji ima 65536 adresa dok je zbog grupisanja 4 8-bitna piksela u 32-bitnu vrednost potrebno da se koristi svaka četvrta adresa. Dok se ne pozove linija 20. u sekvenci drajver čeka u 15. liniji koda odnosno kod „get_next_item(req)“ blokirajuće metode, i kada se ona odblokira (sekvenca pozvala „uvm_do_with()“ metodu), drajver postvalja na „data_ref“ port bram interfejsa podatak koji je modulu potreban (linija 16), ponovo poziva „item_done“ metodu, u 17. liniji koda čijim pozivanjem se kontrola vraća sekvenci, čime je čitanje iz bram memorije završeno i jedna 32-bitna vrednost koju čine 4 piksela je poslata modulu. Prethodno opisani proces se ponavlja pri svakom narednom zahtevu adrese od strane modula koji dobija nova 4 piksela. Takođe kada modul završi sa obradom frejma i spreman je za obradu narednih frejmova , on na svom „interrupt“ interfejsu postavlja logičku jedinicu i u skladu sa tim sekvenca prelazi na naredni frejm koji bazna sekvenca ponovo učitava u „ref_queue“. Informaciju da se interapt desio prvi vidi „interrupt monitor“ (slika 1) i on tu informaciju prosleđuje drajveru preko TLM konekcije (slika 1), odnosno kada se desi interapt poziva se metoda unutar drajver komponente koja menja vrednost „interrupt_o“ 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 (linija 7), a „interrupt_o“ promenljiva se vraća na logičku nulu (linija 5). Linije drajvera 6 i 8 sadrže „get_next_item(req)“ i „item_done(req)“ metode, respektivno, koje prosleđuju „req“ objekat sekvenci sa vrednosti interapta koji tek tada sekvenca po prvi put dobije. Taj interapt sekvenca obrađuje u u 11. liniji koda, i ako se on desi uvećava se promenljiva „start_frame_r“ u liniji 14 koja se zatim u 16. redu prosledi kao parametar funkciji „read_ref_img()“ kako bi bazna sekvenca učitala naredni frejm. Sekvenca “ARPS_IP_bram_ref_simple_seq_2” za razliku od “ARPS_IP_bram_ref_simple_seq” ima dve dodatne linije (13. i 15) koje se sastoje od IF uslova

gde se proverava vrednost promenljive „start_frame_r“ od koje zavisi test scenario. Naime ako je vrednost „start_frame_r“ jednaka nuli onda se u test uključuju granični slučajevi, dok se za vrednost četiri preskaču granični slučajevi i u red „ref_queue“ se učitavaju samo potpuno randomizovane vrednosti piksela. Zatim se povećava promenljiva „cnt_seq“ (linija 17), preko koje sekvenca u IF uslovu proverava da li je drajveru poslat broj frejmova unapred određen promenljivom „num_of_seq“ u (linija 8).
Kada se obradi zadati broj frejmova sekvenca se završava, a samim tim se završava i simulacija.

2.3 “BRAM REF monitor”

“BRAM REF monitor” je zadužen za nadgledanje portova na bram interfejsu, i da na osnovu podataka koje vidi na bram_ref interfejsu “scoreboard” komponenti prosledi vrednosti piksela koji čine referentni frejm kako bi ih “scoreboard” dalje obradio. Način na koji je to realizovano oslanja se na sam način rada VMD modula koji započinje sa obradom slike tako što izvlači piksel po piksel iz bram memorije i radi određene proračune sa njima, a taj proces ponavlja dok ne uradi proračune sa svim pikselima frejma. Za vreme izvlačenja tih pisksela bram monitor to nadgleda i prosleđuje piksele putem TLM komunikacije “scoreboard” komponenti, dok “scoreboard” komponenta smešta piksele u red. Sama implementacija “scoreboard” komponente u kojoj se vrši provera dobijenih podataka od strane bram monitora i referentnog modela će biti opisana kasnije u tekstu.

2.4 “BRAM CURR sequence”,“BRAM CURR driver” i “BRAM CURR sequencer”

Komponente “BRAM CURR sequence”, “BRAM CURR driver” i „BRAM CURR sequencer“ vrše identičnu funkciju kao i “BRAM REF sequence”, “BRAM REF driver” i „BRAM REF sequencer“ sa malom razlikom u frejmovima. Naime dok „BRAM CURR“ kompnente simuliraju bram memoriju koja VDM modulu na zahtev prosleđuje trenutni frejm koji čini video, „BRAM REF“ komponente prosleđuju frejm koji je prethodio trenutnom frejmu, odnosno referentni frejm. Zbog sličnosti u radu ovih komponenti dalji opis istih je suvišan.

2.5 “BRAM CURR monitor”

“BRAM CURR monitor“ takođe ima istu fukciju kao i “BRAM REF monitor“ s tim da ova komponenta prilikom nadgledanja bram_curr interfejsa „scoreboard“ komponenti prosleđuje piksele koji čine trenutni frejm za dalju obradu.

2.6 “BRAM MV monitor”

“BRAM MV monitor” ima sličnu funkciju kao i prethodni monitori, a to je nadgledanje bram_mv interfejsa. Naime VMD modul rezultate poređenja dva frejma u vidu vekotra pomeraja smešta u posebnu bram memoriju. Prilikom proračuna svaki vektor se preko TLM komunikacije prosleđuje „scoreboard“ komponenti koja te vektore smesti u red i poredi ih sa vektorima pomeraja dobijenim od strane referentnog modela koji je implementiran u „scoreboard“ komponenti.

2.7 “Interrupt monitor”

Ova komponenta je zadužena za nadgledanje interapt interfejsa VMD modula. Kada se interapt desi o tome preko TLM interfejsa obavesti “BRAM REF driver” komponentu,“BRAM CURR driver” komponentu kao I “AXI LITE driver” komponentu.

2.8 “AXI LITE agent”, “BRAM REF agent”, “BRAM CURR agent”, “BRAM MV agent” i “Interrupt agent”

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

2.9 “Scoreboard”

Unutar “scoreboard” komponente je implementiran referentni model, koji na osnovu podataka koje dobija od monitora treba da uradi svoj proračun i na kraju uporedi sa rezultatima koje je dao modul. Referentni model, tj. prediktor, koji je implementiran je sama realizacija ARPS algoritma u programskom jeziku sistem verilog (*eng. systemverilog*). Prilikom rada VMD modula prvo se u „scoreboard“ komponentu šalju pikseli prikupljeni na bram_ref i bram_curr interfejsima da bi se smestili u za to predviđene redove unutar write_bram_ref i write_bram-curr funkcija, „ref_queue“ i „curr_queue“ respektivno. Prilikom upisa u ove redove koji su dvodimenzionalni upisuje se i vrednost flag promenljivih kako bi se tačno znalo koje lokacije su popunjene pikselima na osnovu odgovarajućih adresa. Nakon toga ta dva reda se prosleđuju kao parametri funkcije „motionARPS“ prilikom njenog poziva. Kada VMD modul izvršava proračun BRAM MV monitor” komponenta šalje dobijene vektore pomeraja na bram_mv interfejsu “scoreboard” komponenti koja zatim unutar write_bram_mv funkcije smešta vektore u red

„mv_bram_q“. Nakon završetka rada VMD modula takođe unutar write_bram_mv funkcije poziva se funkcija „motionARPS“ koja na osnovu dobijenih frejmova vrši proračun i smešta vektore pomeraja u novi red nazvan „mv_ref“ (primetiti da se ovaj put *ref* ne odnosi na referentni frejm iz „BRAM REF agenta“ već na referentni model „scoreboard“ komponente) i tako vrši funkciju referentnog modela. Pomću flag-ova je regulisano da se dva reda , „mv_ref“ i „mv_bram_q“, tek porede kada su i VMD modul i referentni model završili svoje proračune i ukoliko se desi da prilikom poređenja postoji neslaganje prijavljuje se greška. U suprotnom ispisuje se poruka da su vektori pomeraja jednaki čime je potvrđeno da VMD modul ispravno radi.

2.10 “Environment” i “Configuration”

Komponenta “environment” instancira i enkapsulira prethodno opisane komponente, ali samo one koje su neophodne kako bi se verifikovao VMD modul. “Environment” komponenta zna koje komponente treba da instancira na osnovu “configuration” komponente. Polja unutar “configuration” komponente se koriste kako bi se konfigurisale komponente okruženja koje će se koristiti pri verifikaciji modula. Tako se kreira pet agenata od kojih su 3 aktivna („AXI LITE agent“, „BRAM REF agent“ i „BRAM CURR agent“), odnosno sadrže sekvencer drajver i monitor, dok su dva agenta pasivna („BRAM MV agent“ i „Interrupt agent“) koji se sastoje samo od monitora zaduženih za nadgledanje signala.

2.11 “Tests”

“Test” objekat instancira i enkapsulira “environment” komponentu i pokreće sekvence na određenom sekvenceru. Koju sekvencu pušta zavisi od toga koji test se koristi. Postoje dva testa nazvana „ARPS_IP_test_simple“ i „ARPS_IP_test_simple_2“. U slučaju da se koristi test „ARPS_IP_test_simple“, onda se na „bram ref sequencer“ komponenti pušta “bram_ref_seq” sekvenca iz klase „ARPS_IP_bram_ref_simple_seq“, a na “bram curr sequencer” komponenti se pušta “bram_curr_seq” sekvenca iz klase „ARPS_IP_bram_curr_simple_seq“, dok ako se koristi „ARPS_IP_test_simple_2“ na istim sekvencerima se puštaju sekvence istog naziva sa razlikom u tome što su ove sekvence instance klasa „ARPS_IP_bram_ref_simple_seq_2“ i „ARPS_IP_bram_curr_simple_seq_2“. Na “axil sequencer” se pušta “axil_seq” sekvenca koja je nezavisna od toga koji test se pušta i uvek vrši istu funkciju. Razlika između ova dva testa ogleda se u tome što je „ARPS_IP_test_simple“ zadužen za testiranje VMD modula kada se u sekvence jedan po jedan učitavaju frejmovi koji čine video čime pikseli imaju unapred određenu, fiksnu, vrednost, dok „ARPS_IP_test_simple_2“ prilikom testiranja u sekvence učitava randomizovane vrednosti piksela sa ograničenjima koja su osmišljena kako bi pokrila sve moguće scenarije. Naime scenariji su definisani tako da ispitaju granične slučajeve tako što za dva frejma (referentni i trenutni) prvo učitaju sve piksele vrednosti nula (belo-belo), zatim vrednosti nula za

referentni frejm i vrednosti 255 za trenutni frejm (belo-crno), nakon toga ide kombinacija „crno-belo“ i posle nje dolazi „crno-crno“, da bi na kraju pikseli za oba reda bili potpuno randomizovani vrednostima u opsegu od 0 do 255 (širina piksela je 8 bita) . Sa ova dva testa su pokrivene sve mogućnosti rada VMD modula kako se bi se proverila njegova ispravnost fukcionisanja. Da bi se pokrenuo određeni test potrebno je napraviti izmenu u „run.do“ fajlu koji se nalazi u folderu „sim“ tako što se ispred komande pokretanja simulacije „ARPS_IP_test_simple“ testa stavi simbol za komentar „ # “, a za „ARPS_IP_test_simple_2“ test se isti taj simbol izbriše. U slučaju ponovnog pokretanja prvog testa potrebno je ponovo zameniti mesta simbola za komentarisanje. Takođe u folderu „sim“ se nalazi i skripta „wave.do“ koja služi za dodavanje svih signala od interesa u „wave“ prozor „Questa sim“ simulatora, kao i njihovo grupisanje po interfejsima.

3. Fukcionalna verifikacija VMD (*eng. Video Motion Detection*) modula

3.1 Opis VMD modula

Modul koji se verifikuje ima ulogu određivanja vektora pomeraja (*eng. Motion Vectors)* svakog makro bloka trenutnog frejma u odnosu na referentni(prethodni) frejm. Pomeraji su definisani parametrom P_SIZE, dok veličinu bloka definiše parametar MB_SIZE. Maksimalni pomeraji po x i y osi su -P_SIZE i P_SIZE. U našem slučaju pomeraj je 7, a veličina makro bloka je 16x16. Pomeraj se posmatra u odnosu na početnu tačku makro bloka (gornja leva).

Algoritam koji se koristi u VMD modulu je ARPS (*eng. Adaptive Rood Pattern Search)*, postoje i drugi algoritmi ali on je izabran zbog njegovih karakteristika, kao što su nizak PSNR (*eng. Peak Signal-to-Noise Ratio)* kao i promenljiv broj koraka u određivanju vektora, koji je u globalu manji od ostalih algoritama. Njegova mana je što ne može da detektuje pomeraje koji se odvijaju po dijagonali, ova mana može da se vidi na slici 1.4. Kao i u svim algoritmima ovog tipa (*Block Matching*), koji određuju vektore pomeraja, tako što računaju sumu apsolutne razlike izmedju dva bloka, potrebno je imati na raspolaganju 2 susedna frejma koja će se smeštati u određenu memoriju.

Rezolucija frejma koji se koristi je 256x256 pix, „*grayscale“*, što znači da je za svaki piksel potrebno 8 bit-a. Veličina podataka koja je potrebna za jedan frejm je 64KB. Za smeštanje tih podataka (piksela) korišćene su dvopristupne BRAM memorije širine 32 bit-a i dubine 16384. U okviru jedne memorijske lokacije smeštaju se 4 uzastopna piksela. Raspored smeštanja piksela u BRAM memoriju sa označenim indeksima može se videti na slici 3.2 , dok njihovo indeksiranje u okviru frejma može se videti na slici 3.1

256 pix

256 pix						
	0	1	...	...	254	255
	255	256	...	...	510	511
	...	...	...	...	...	...
	...	...	...	...	...	...
	65024	65025	...	...	65278	65279
	65280	65281	...	...	65534	65535

Slika.3.1 Indeksiranje u okviru frejma

32 bit				
8 bit	8 bit	8 bit	8 bit	address
0	1	2	3	0x0
4	5	6	7	0x4
8	9	10	11	0x8
...	...	...	...	...
65532	65533	65534	65535	0xfffe

Slika.3.2 Raspored smeštanja piksela u BRAM memoriju

Za smeštanje rezultata obrade modula tj. vektora pomeraja, koristi se BRAM memorija širine 32 bit-a i dubine od 512. Dubina od 512 lokacija se dobija na osnovu vektora pomeraja koji ima 2 koordinate, a broj vektora se odredjuje na osnovu formula (ROW*COL)/(MB_SIZE*MB_SIZE) gde ROW i COL predstavljaju ukupan broj redova i kolona, a MB_SIZE veličinu makro bloka. Koordinate se smeštaju na uzastopnim memorijskim lokacijama za jedan vektor. Podatak koji se smešta u memorijsku lokaciju je tipa *signed,* što znači da mogu da se nađu i negativne vrednosti.

VMD modul koji se posmatra zajedno sa njegovim interfejsima i portovima može se videti na slici 3.3 pod nazivom „AXI_ARPS_IP_0". On se sastoji se od 4 standardna interfejsa i 3 jednobitna ulazna ili izlazna porta.

Interfejs pod nazivom „s00_axi" je standardni *AXI LITE* interfejs. Preko njega šaljemo komandu za pokretanje modula, kao i čitanje trenutnog stanja modula, da li je spreman za pokretanje ili je zauzet.

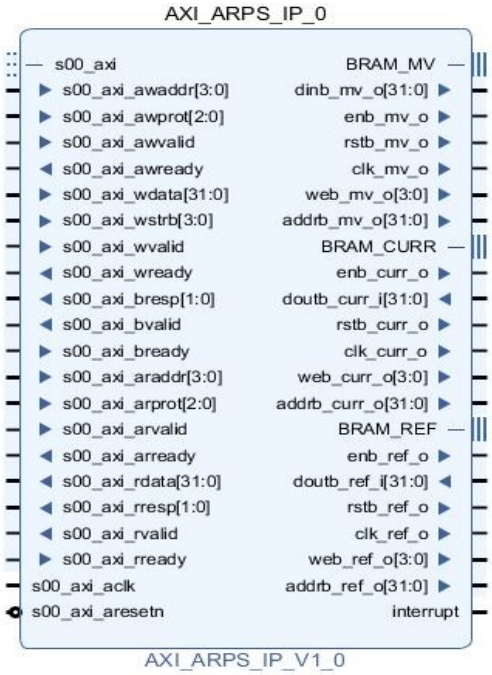
Interfejs pod nazivom „BRAM_MV" je standardni BRAM interfejs za komunikaciju sa BRAM memorijom koga čine 7 portova od kojih se sledećih 6 koristi:

1) dinb_mv_o – izlazni 32-bitni port koji šalje podatke BRAM memoriji
2) enb_mv_o - izlazni jednobitni port za aktivaciju BRAM memorijskog interfejsa
3) rstb_mv_o – izlazni jednobitni port za restart BRAM memorije
4) clk_mv_o – izlazni jednobitni port za taktovanje BRAM memorije
5) web_mv_o – izlazni 3-bitni port za omogućavanje upisa u BRAM memoriju
6) addrb_mv_o – izlazni 32-bitni port za adresiranje BRAM memorije

Interfejsi pod nazivima „BRAM_CURR" i "BRAM_REF" imaju iste portove kao i interfejs „BRAM_MV" s tim da je jedina razlika u imenima i što se kod pomenuta 2 interfejsa koriste ulazni 32 bit-ni portovi za podatke koji se dobijaju od BRAM memorija, a ne izlazni 32 bit-ni port.

Od ostalih portova imamo:

1) s00_axi_aclk – ulazni port na koji dovodimo takt signal
2) s00_axi_aresetn – ulazni port na koji dovodimo reset signal
3) interrupt – izlazni port na koji nam modul signalizira kada je završio sa radom



Slika.3.3 Interfejsi i portovi modula

3.2 Verifikacioni plan VMD modula

Verifikacioni plan se sastoji iz sledećih koraka:

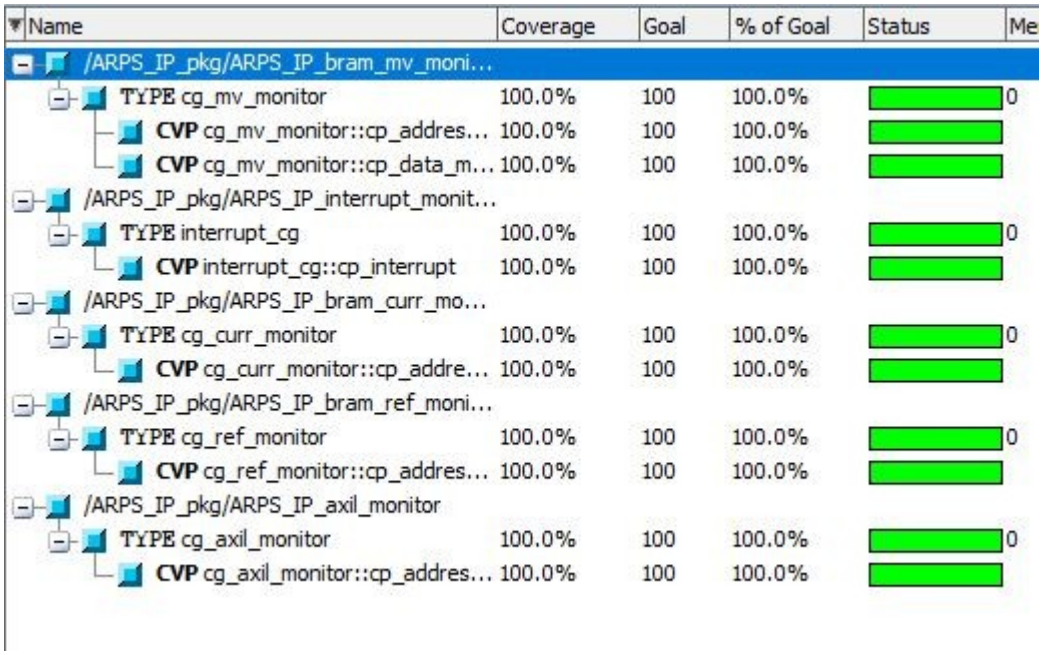
1) Provera funkcionalnosti reseta sistema. Ona je proverena na početku simulacije kada se ceo sistem resetuje.
2) Provera funkcionalnosti Axi lajt (*eng. AXI LITE*) interfejsa. Ona se ogleda u proveri da li modul poštuje Axi lajt (*eng. AXI LITE*) protokol ili ne. Na osnovu signala koji su dobijeni nakon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

3.3 Verifikaciona pokrivenost VMD modula

Provera pokrivenosti vrši se nadgledanjem svih interfejsa. Situacije koje su od posebnog značaja i koje je potrebno pokriti su:

1) Da li se desilo upisivanje u sve unutrašnje registre modula
2) Da li su se desila čitanja iz svih unutrašnjih registara modula
3) Da li je modul zatražio podatke za sve adrese iz memorije
4) Da li su upisani vektori pomeraja na svaku adresu
5) Da li su se desile sve vrednosti vektora pomeraja iz opsega
6) Da li se desio „interrupt“

Nakon puštanja simulacije, rezultati merenja pokrivenosti su sledeći:



Slika 3.4 Coverage

Na slici 3.4 u okviru „cg_mv_monitor“ prvi CVP (*eng. Cover point*) govori o tome da li su se tokom simulacije na svaku adresu upisali određeni vektori pomeraja, dok drugi CVP govori da li su se desile sve moguće vrednosti vektora pomeraja čime su pokrivene situacije 4) i 5). Nakon toga se nalazi „interrupt_cg“ čiji CVP proverava da li su se desile obe vrednosti interapta kako bi bila ispunjena situacija broj 6). Zatim naredna dva CVP nazvani „cg_curr_monitor“ i „cg_ref_monitor“ pokazuju da je VMD model zahtevao podatke od obe BRAM memorije sa

svake adrese iz opsega čime se dobija na uvid u ispunjenost stavke 3). Na kraju su ispunjene i stavke 1) i 2) tako što su pomoću CVP u „cg_axil_monitor“ pokrivene situacije i upisa i čitanja registara. Priložena slika pokazuje da su sve stavke ispunjene, odnosno da je pokrivenost odlična.