ELEKTROTEHNIČKI FAKULTET UNIVERZITETA U BEOGRADU

HARDVERSKA PARALELIZACIJA ARITMETIČKIH OPERACIJA

–Diplomski rad –

Kandidat:

Željko Marinković 2009/0440

Mentor:

doc. dr Zoran Čiča

Beograd, Septembar 2017.

SADRŽAJ

1. UVOD

Imajući u vidu prednost hardverskih rešenja u odnosu na softverska po pitanju brzine rada kao i činjenicu da dobar deo poslova koje procesorska jedinica obavlja u toku izvršavanja nekog programa čine aritmetičke operacije, logično se otvara mogućnost da se te potrebe realizuju na čipovima u vidu logičkih kola. Time bi se, barem teoretski, mogao ubrzati rad različitih sistema i rasteretiti njihova procesorska jedinica.

U ovom radu su prikazane dve realizacije aritmetičkih operacija. Jedna je prosto sabiranje operanada, a druga je proizvoljno odabrana kombinacija množenja i sabiranja. Takođe prikazan je rad komponenata koje obavljaju ove operacije u kombinaciji sa baferima imajući u vidu da operandi ne moraju, u opštem slučaju, dolaziti istom brzinom kojom se mogu obraditi. Za dužinu operanada koji se obrađuju je uzeta vrednost 32 bita i predviđeno je da ih može biti 2, 3 ili 4.

Za realizaciju ovog rada korišćen je VHDL programski jezik za opisivanje hardvera i ISE razvojno okruženje kompanije XILINX® za simulaciju i sintezu.

U drugom poglavlju će biti dat kratak opis samog pojma paralelne obrade i kratak pregled nekih rešenja iz oblasti hardverske paralelizacije. U poglavlju tri će biti prikazane komponente koje se koriste u rešenju, kroz opis osobina i načina realizacije. Takođe će biti dat prikaz njihovog rada u simuliranim uslovima. U četvrtom poglavlju će biti dat prikaz simulacije rada celog projekta u obe varijante kao i procenat zauzeća resursa programabilnog čipa nakon obavljene sinteze. Na kraju, u petom poglavlju, će biti predstavljen zaključak.

2. PARALELIZACIJA

2.1. Pojam paralelne obrade

Paralelizacija u kompjuterskoj obradi se razvila kao posledica potrebe za njenim ubrzanjem imajući u vidu da se komplikovani problemi često, do neke granice, mogu podeliti na jednostavnije potprobleme i zatim zasebno rešiti. U početku se ubrzanje realizovalo kroz povećanje frekvencije rada i/ili dupliranjem dužine procesorske reči (4, 8, 16, 32 i 64 bita). Međutim, vremenom se javlja problem u frekvencijskom skaliranju koje, zbog potrošnje, više ne prati razvoj procesorskih jedinica, a koji i dalje prate Moore-ov zakon o gustini pakovanja. Teoretski bi se bzina obrade trebala povećati (tj. vreme obrade smanjiti) onoliko puta koliko se doda novih procesora ili jezgara. To naravno nije moguće jer uvek postoji neki deo problema koji nije moguće nezavisno rešiti.

Obično se paralelna obrada realizuje povećanjem standardnih resursa mašine koja tu obradu vrši uz adekvatan softver koji te resurse treba da iskoristi. To može biti povećanje broja jezgara samog procesora, izvršnih jedinica u jezgru, brojem samih procesora. Definisanjem načina na koji procesori koriste memoriju kao i gde se ona fizički nalazi, definišemo i tipove paralelne obrade. Ona može biti višejezgarna (*multi-core*), simetrično multiprocesiranje, distribuirana obrada itd.

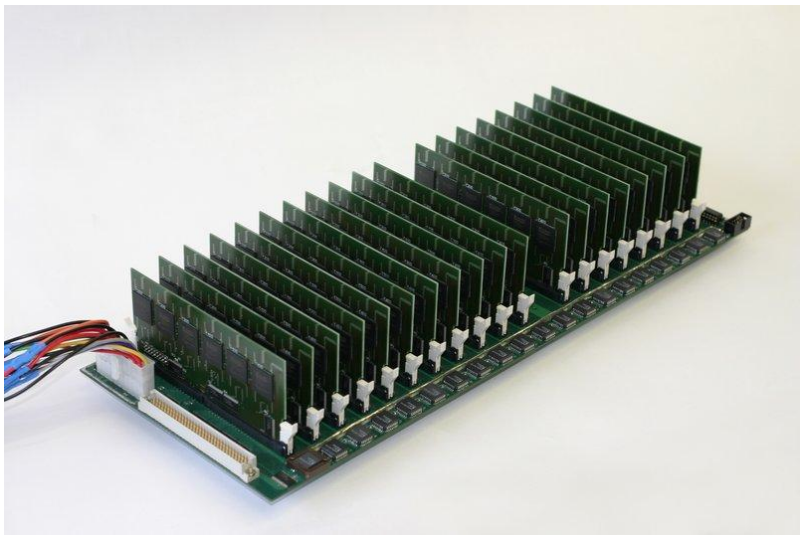
2.2. Paralelna obrada pomoću FPGA

Ovo je jedan od egzotičnijih načina obrade, i koristi se najčešće za aplikacije koje treba realizovati u realnom vremenu nad velikom količinom podataka kao što su digitalno procesiranje signala radio teleskopa, napredna video kompresija, transkodovanje i hiperspektralna fotografija u realnom vremenu [1]. Jedan od primera realizacije je HERC [2] (*High-End Reconfigurable Computing System*) Berklijevog BEE2 projekta.

Predlog da se čipovi koje je moguće rekonfigurisati koriste kao koprocesorske jedinice na mašinama opšte namene dao je Gerald Estrin. Ukoliko se u kombinaciji sa klasičnim procesorima, neki algoritam više puta reprodukuje na istom čipu ili se primeni na više čipova moguće je ostvariti visok nivo paralelne obrade.

Neke od realizacija obrade pomoću FPGA čipova su:

- Copacobana (*Cost-Optimized Parallel COde Breaker*, slika 2.2.1.) je mašina bazirana isključivo na FPGA (poslednja verzija iz 2008, Virtex-4) , a namenjena za razbijanje šifri generisanih pomoću DES algoritma [3].



Slika 2.2.1. Copacobana

- PACT XPP (*eXtreme Processing Platform*) je arhitektura koju je moguće menjati u toku izvršenja, a koja je pogodna za aplikacije u multimediji, telekomunikacijama, simulaciji itd [4][5].
- Montium [6] je rešenje namenjeno bežičnim komunikacionim sistemima, obradi slike, procesiranju biomedicinskih signala, primeni u radarskoj tehnologiji itd.
- Morphosys [7] je rekonfurabilni procesor namenjen za paralelnu obradu podataka i visokozahtevno procesiranje kao što su video kompresija i automatsko prepoznavanje (*Automatic Target Recognition*).

3. ARHITEKTURA REŠENJA

U ovom poglavlju će biti dat pregled komponenti koje se u ovom radu realizovane i koje se u njemu koriste, kao i njihov način rada. Komponente koje će biti predstavljene su: *FIFOBafer* koji se koristi za čuvanje operanada, njihovog broja i rezultata obrade, zatim *aritm_komp_jedan* i *aritm_komp_dva* koje vrše konkretnu aritmetičku obradu i konrolišu upis u izlazni bafer i na kraju *KomParOb* koja koristi prethodno navedene komponente da bi realizovala kompletnu funkcionalnost.

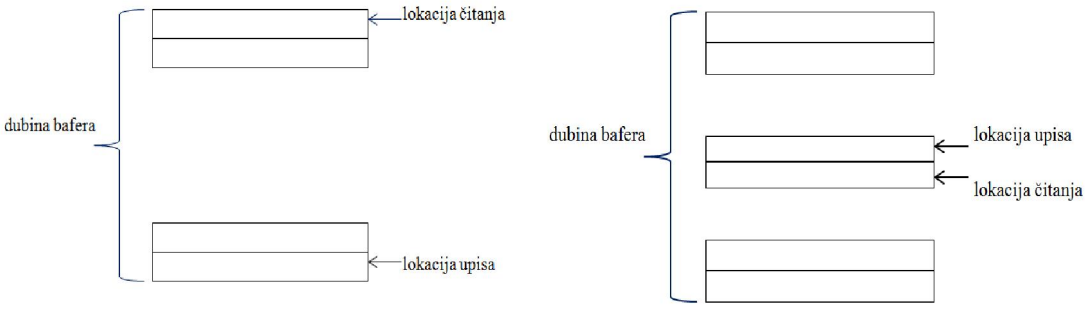
3.1. FifoBafer

Napravljen je da bude promenljive dubine i dužine reči i da upis i čitanje vrši na različitim frekvencijama. *FIFOBafer* ima dva definisana parametra, dužina reči (*duizina_reci*) i dubina bafera (*dubina_bafera*), na osnovu kojih se definiše memorija bafera i dužina ulaznih i izlaznih vektora.

Ulazni signali su taktovi za upis i čitanje (*wr_clk, rd_clk*), vektor podataka (*din*), dozvola za upis (*wr_en*), dozvola za čitanje (*rd_en*) i signal reseta (*rst*). Izlazni signali su vektor podataka (*dout*), signal kojim se označava da je bafer pun (*full*) i signal kojim se označava da je bafer prazan (*empty*).

Upis i čitanje su realizovani kao dva odvojena procesa koji se izvršavaju svaki na svom taktu. Vrši se interna kontrola stanja bafera da bi se izbegla situacija da se prebriše nešto što još nije pročitano ili pročita nešto što je već čitano. Ova kontrola se vrši upoređivanjem vrednosti internih signala *lokacija_upisa* i *lokacija_citanja*, odnosno njihovih međusobnih razlika. Na osnovu njih (*razlika1* i *razlika2*) se utvrđuje da li je bafer pun, prazan ili nijedno od ta dva pa samim tim i da li se iz njega može čitati i/ili u njega nešto upisati uzimajući u obzir vrednosti ulaznih signala *wr_en* i *rd_en*.

Signali *lokacija_upisa* i *lokacija_citanja* su celobrojnog tipa i inkrementiraju se pri upisu, odnosno čitanju po modulu *dubina_bafera*. Vrednosti koje mogu da imaju su u opsegu od 0 do *dubina_bafera*-1. Slika 3.1.1. pokazuje njihov uzajamni položaj kada je bafer pun.



Slika 3.1.1. Moguće situacije kada je bafer pun

Signal *razlika1* ima vrednost razlike između lokacije upisa i lokacije čitanja. Koristi se u slučaju kada je razlika jednaka *dubina_bafera*-1 (slika 3.1.1. levo) ili kada je nula. Kada ima vrednost *dubina_bafera*–1 aktivira se signal *pun1*. Signal *razlika2* ima vrednost razlike između

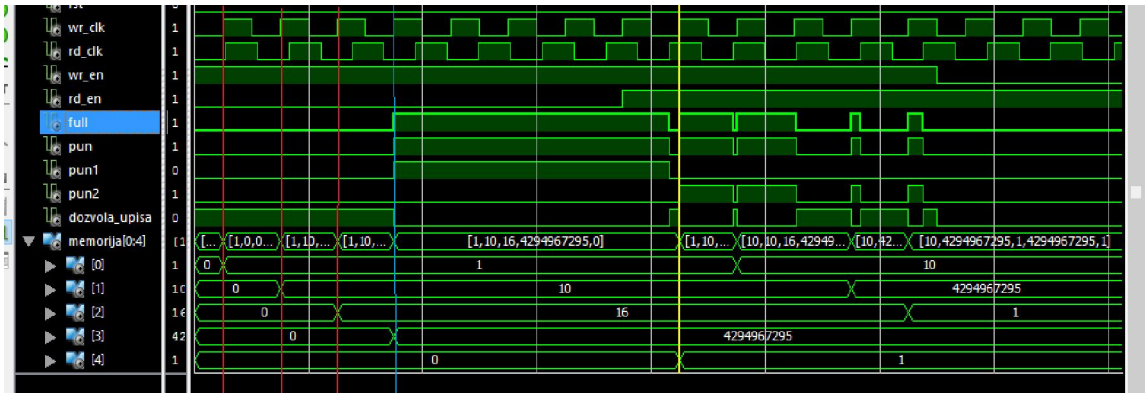
lokacije čitanja i lokacije upisa (negativna vrednost signala *razlika1*). Koristi se u slučaju kada se lokacija upisa nalazi tik iza lokacije čitanja (slika 3.1.1 desno). Tada se aktivira signal *pun2*.

Ako je bafer pun aktiviraće se jedan od dva signala, *pun1* ili *pun2*, pa samim tim i signal koji predstavlja kombinaciju ova dva, a to je signal *pun*.

Signal *pun* se prosleđuje na izlaz *full* i koristi za upravljanje upisom u bafer, da bi se sprečio upis u bafer u situaciji kada je on pun, jer kontroliše moguće vrednosti signala *dozvola_upisa*. Kada bafer nije pun (*pun*=0) signalu *dozvola_upisa* se prosleđuje vrednost ulaznog signala *wr_en* . U suprotnom *dozvola_upisa* ima vrednost 0. Podaci koji dolaze na ulaz bafera kada je on pun se odbacuju sve dok se ne oslobodi neko mesto u njemu.

Kada signal *dozvola_upisa* ima vrednost 0 na uzlaznu ivicu takta *wr_clk* upis nije moguć jer se preskače elsif grana u tom procesu. Kada *dozvola_upisa* ima vrednost 1 na uzlaznu ivicu takta upisa (*wr_clk*) vrši se upis ulaznog podatka na trenutnu lokaciju upisa i ta lokacija se zatim inkrementira po modulu *dubina_bafera*.

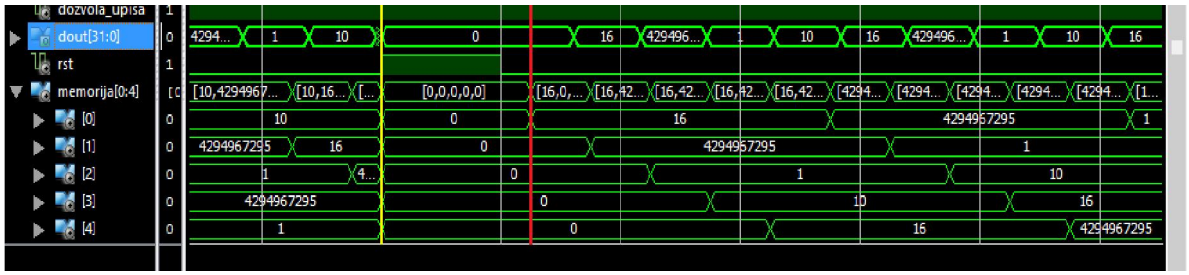
Na slici 3.1.2. se može videti kako se reguliše upis u bafer dubine 5 u skladu sa vrednostima pomoćnih signala *pun1* i *pun2*. Crvenim linijama su obeleženi trenuci kada se vrši upis na prve tri lokacije (0 do 2) u promena njihovih vrednosti. Plavom linijom je obeležen trenutak upisa na četvrtu lokaciju (3). Kako se po završetku upisa inkrementira lokacija upisa, ona dobija vrednost 4 što znači da se nalazi tik iza lokacije čitanja, jer je ovo kružni bafer, pa se aktivira signal *pun1*, a samim tim i signali *pun* i *full* što je takođe obeleženo plavom linijom. Primetiti da *dozvlola_upisa* dobija vrednost 0. Žutom linijom je obeležena situacija kada se aktivira *pun2* jer su izvršeni čitanje i upis u bafer.



Slika 3.1.2. Prikaz popunjavanja bafera dubine 5

Signalom *razlika1* se reguliše i situacija kada je bafer prazan i tada se aktivira signal *prazan*. Signal *prazan* se prosleđuje na izlaz *empty* i koristi za upravljanje čitanjem iz bafera, da bi se sprečilo čitanje iz praznog bafera, jer kontroliše moguće vrednosti signala *dozvola_citanja*. Kada bafer nije prazan (*prazan*=0) *dozvola_citanja* dobija vrednost *rd_en*. U suprotnom *dozvola_citanja* dobija vrednost nula i zabranjuje se čitanje, a izlazni vektor *dout* takođe dobija vrednost nula. Izlazni vektor *dout* će dobiti vrednost nula i u slučaju da bafer nije prazan, ali je vrednost signala *rd_en* nula.

Kada se aktivira *rst*, signali *lokacija_upisa*, *lokacija_citanja* dobijaju vrednost nula, a samim tim i izlazni signal *empty* dobija vrednost jedan. Sve lokacije u samoj memoriji bafera dobijaju vrednost nula, kao i izlazni vektor *dout*.



Slika 3.1.3. Prikaz stanja bafera pri aktiviranju reseta

Na slici 3.1.3 je žutom linijom obeležen trenutak kada se aktivira reset. Primetiti da izlaz *dout* i sve lokacije u baferu dobijaju vrednost nula. Crvenom linijom je obeležen početak ponovnog upisa u bafer.

Iz poslednje dve stavke se može zaključiti da će se u slučaju da je bafer prazan , resetovan ili da je signal *rd_en* neaktivan (ima vrednost 0) na izlazu bafera naći sve nule.

3.2. Komponenta za paralelnu obradu (KomParOb)

Komponente koje KomParOb koristi su fifo baferi i jedna od aritmetičkih komponenata (*aritm_komp_jedan* ili *aritm_komp_dva)*. Komponente se u projekat uvoze preko paketa u kome su deklarisane.

Funkcije za koje se koristi *FIFOBafer* su čuvanje operanada, čuvanje broja operanada i čuvanje rezultata obrade. Pri instanciranju se navode vrednosti potrebnih parametara *duzina_reci* i *dubina_bafera*. Parametar *dubina_bafera* je isti za sve funkcije dok se parametar *duzina_reci* razlikuje za svaku funkciju.

Ulazni signali su taktovi za upis i čitanje (*wr_clk, rd_clk*), niz ulaznih 32-bitnih vektora koji predstavljaju operande i koje treba sačuvati u ulaznim baferima (*din_ul_bafer*), vektor od 4 bita koji predstavlja dozvole za upis (*wr_en*) i signal reseta (*rst*).

Postoji samo jedan izlazni signal i to je vektor podataka iz izlaznog bafera (*dout_izl_bafer*) čija dužina zavisi od aritmetičke operacije koja se realizuje. Za *aritm_komp_jedan* je ovaj izlaz dužine 34 bita, a za *aritm_komp_dva* 65 bita.

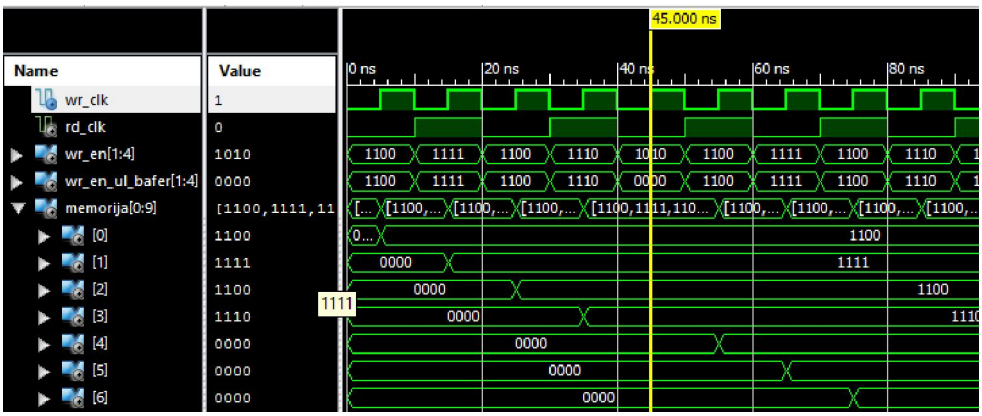
Projektom je definisano da broj operanada može biti 2, 3 ili 4. U svrhu njihovog čuvanja instancirana su četiri ulazna bafera u paraleli – po jedan za svaki operand.

Operandi koje treba obraditi dolaze na ulaz paralelno sa dozvolama za upis. Ove dozvole se ujedno tumače i kao broj operanada i treba ih sačuvati da bi se znalo iz kojih bafera treba da se čita. Dozvole se stoga čuvaju u baferu broja operanada. Kako se za dozvole upisa u ulazne bafere koristi četiri bita, jer postoje četiri ulazna bafera, *dužina_reči* bafera broja operanada pri instanciranju ima vrednost 4. Dozvole za upis se ne prosleđuju direktno baferima. KomParOb vrši proveru dozvola za upis pre upotrebe.

Upis u ulazne bafere kontroliše se signalom *wr_en_ul_bafer*. Njegova vrednost zavisi od popunjenosti ulaznih bafera (*full_ul_bafer*) i vrednosti signala *wr_en*. Ukoliko signal *full_ul_bafer* ima jednu od sledećih vrednosti: 1100, 1110 ili 1111, smatra se da su baferi puni i dalje se zabranjuje upis u njih tj. signal *wr_en_ul_bafer* dobija vrednost sve nule. Isto se dešava u slučaju da na ulaz *wr*_en dođe neregularna kombinacija bita. U tabeli 3.2.1. su date regularne vrednosti signala *wr_en*. U slučaju da na ulaz dođe neregularna kombinacija bita operandi se odbacuju.

Tabela 3.2.1. Regularne vrednosti ulaznog signala *wr_en*

Broj operanada	*wr_en*
0	0000
2	1100
3	1110
4	1111

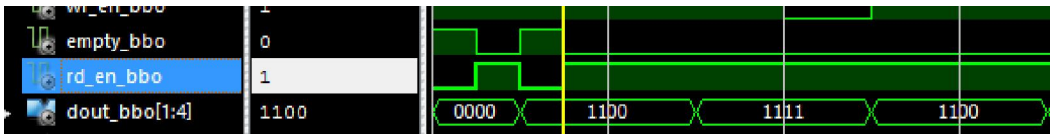


Slika 3.2.1. Upis ulaznih dozvola u bafer broja operanada

Na slici 3.2.1. se može videti da se vrednost signala *wr_en* 1010 (u trenutku 45ns) odbacuje i da signal *wr_en_ul_bafer* dobija vrednost 0000 čime se zabranjuje upis u ulaze bafere.

Upis u bafer broja operanada se takođe kontroliše dvostrukom proverom. Prvo na osnovu popunjenosti ulaznih bafera, a ne na osnovu sopstvene popunjenosti jer se čitanje iz ulaznih bafera realizuje sa jednom periodom kašnjenja u odnosu na čitanje iz bafera broja operanada. To znači da će se u ulaznim baferima, ukoliko nisu prazni, uvek nalaziti jedan podatak više u odnosu na bafer broja operanada i da će se pre popuniti. Drugom proverom se proverava ispravnost signala *wr_en*, pa se u slučaju neispravne vrednosti ona odbacuje tj. ne čuva se u baferu broja operanada.

Čitanje iz bafera broja operanada se kontroliše na osnovu vrednosti signala *empty_bbo* tj. na osnovu toga da li je ovaj bafer prazan što se može videti na slici 3.2.2.. Kada se upiše prvi podatak u ovaj bafer, signal *empty_bbo* dobija vrednost 0 pa dozvola čitanja ovog bafera (*rd_en_bbo*) dobija vrednost 1. Tako čitanje iz bafera broja operanada počinje odmah posle upisa i završava se kada se on isprazni ili eventualno resetuje.

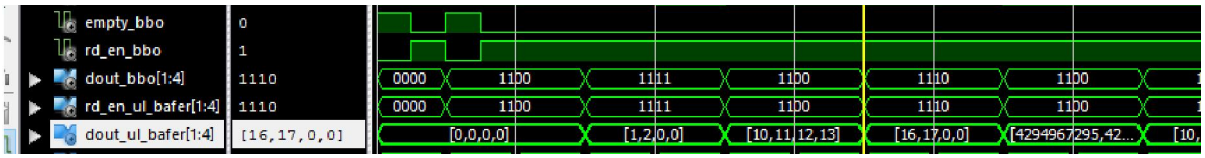


Slika 3.2.2. Prikaz kontrole čitanja bafera broja operanada

Podaci iz bafera broja operanada se prosleđuju signalu *rd_en_ul_bafer* čime se kontroliše čitanje iz ulaznih bafera. Onim baferima kojim nije aktivirana dozvola za čitanje, na izlazu za podatke se, po rešenju FIFOBafer-a, nalaze sve nule.

Operandi se zatim prosleđuju komponenti koja vrši njihovu obradu. U zavisnosti od željene funkcije, to je ili *aritm_komp_jedan* ili *aritm_komp_dva* čija su rešenja data u nastavku.

Paralelno sa operandima se komponenti prosleđuje i njihov broj pomoću signala *broj_operanada*. Ovaj signal je ustvari izlazni podatak iz bafera broja operanada *dout_bbo* koji je zakašnjen za jedan takt *rd_clk* da bi se uskladio sa operandima na koje se on odnosi. Na slici 3.2.3. se može videti da podaci iz ulaznih bafera kasne za jedan takt *rd_clk* u odnosu na dozvole čitanja.



Slika 3.2.3. Prikaz čitanja iz ulaznih bafera

Rezultat obrade koju vrši aritmetička komponenta, *dout_aritm_komp* se povezuje direktno na *din* ulaz izlaznog bafera.

Takođe i izlaz *wr_en_izl_bafera* komponente, kojim se kontroliše upis u izlazni bafer, se povezuje na *wr_en* ulaz bafera.

Čitanje iz izlaznog bafera je realizovano na isti način kao i čitanje iz bafera broja operanada. Kada bafer nije prazan tj. kada *empty_izl_bafer* ima vrednost 0 aktivira se dozvola za čitanje (*rd_en_izl_bafer*). Taktovi koji se koriste i za upis i za čitanje su po ovom rešenju isti (*rd_clk*) mada samo rešenje FIFOBafer-a koje se koristi dozvoljava različite taktove.

Dužina reči izlaznog bafera koja se definiše pri instanciranju komponente u Generic map delu treba da odgovara dužini izlaznog vektora iz aritmetičke komponente. Prema dizajniranom rešenju postoje dve komponente: *aritm_komp_jedan* i *aritm_komp_dva*, čiji su izlazni vektori dužine 34 bita odnosno 65 bita respektivno, tako da će u prvom slučaju *duzina_reci* izlaznog bafera biti 34, a u drugom 65. Ove dužine su podešene za 32-bitne dužine operanada i pokrivaju dužinu svih mogućih rezultata aritmetičkih operacija koje se izvršavaju u *aritm_komp_jedan* i *aritm_komp_dva* komponentama.

Radi preglednije simulacije, dubine svih bafera su postavljene na 10, ali bi se u praktičnoj primeni dubina određivala prema načinu na koji operandi dolaze na obradu. To podrazumeva brzinu kojom dolaze (u odnosu na brzine obrade) kao i da li dolaze kontinualno ili u naletima (*burst*-ovima).

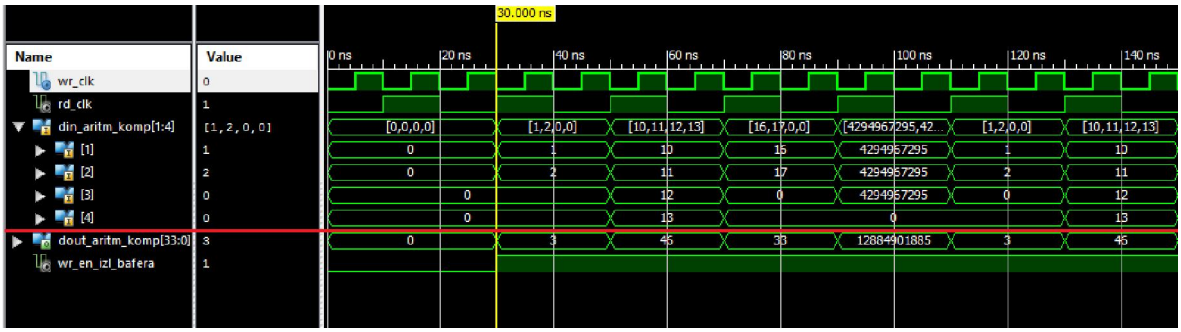
3.3. Aritmetička komponenta jedan (*aritm_komp_jedan*)

Operacija koju ova komponenta treba da izvršava je prosto sabiranje operanada. Njih može biti 2, 3 ili 4, ukoliko postoje, ili 0. Stoga je jedan od ulaznih signala i *broj_operanada*. Drugi ulazni signal je *din_aritm_komp* koji predstavlja operande koje treba obraditi. Tipa je niza od četiri 32-bitna vektora (*matrica1*) koji je već definisan u paketu jer je predviđeno da radi u kombinaciji sa ulaznim baferima koji koriste isti tip. Stoga se promena dužine operanada realizuje unutar gorepomenutog paketa. Prilikom instanciranja komponente potrebno je dodatno deklarisati dužinu operanada (*duzina*_operanada) jer se na taj način menja dužina izlaznog vektora koji predstavlja rezultat kao i dužina pomoćnog signala koji se koristi unutar komponente.

Izlazni signali su dozvola upisa u izlazni bafer (*wr_en_izl_bafer*) i naravno sam rezultat operacije (*dout_aritm_komp*).

Signal *broj_operanada* se koristi za upravljanje mogućim vrednostima izlaznog signala *wr_en_izl_bafer* koji se koristi u daljem delu dizajna. Kada *broj_operanada* ima vrednost 0000 znači da ustvari nema operanada i signal *wr_en_izl_bafer* tada ima vrednost 0. U svim ostalim slučajevima će imati vrednost 1.

Kako je maksimalan neoznačen ceo broj koji se može dobiti na ulazu $2^{duzina\_operanada}$ -1 , u slučaju da postoje četiri operanda (sabirka) maksimalan neoznačen ceo broj koji se može dobiti kao rezultat sabiranja je $(2^{duzina\_operanada} - 1)*4 = 2^{duzina\_operanada\ +2} - 4$. Za kodovanje ovog broja su potrebna dva dodatna bita pa se zato pre samog sabiranja vrši konkatanacija vektora "00" i ulaznih podataka. Ove vrednosti se smeštaju u *pomoćni_ulaz* koji je tipa *matrica_podataka* koji je definisan unutar same komponente na osnovu vrednosti *duzina_operanada*. Rezultat operacije se prosleđuje direktno na izlaz *dout_aritm_komp*.



Slika 3.3.1. Rad aritmetičke komponente jedan

Na slici 3.3.1. je prikazan simuliran rad aritmetičke komponente jedan pri čemu se može videti da se odmah dobija rezultat obrade i to jedan za drugim čime se stvara pajplajn efekat čime bi trebalo značajno da se ubrza obrada.

3.4. Aritmetička komponenta dva (*aritm_komp_dva*)

Operacija koja treba da se izvrši zavisi od broja operanada ali se može uopšteno predstaviti kao *a*b+c*d* i njen pregled je dat u tabeli 3.4.1.

Tabela 3.4.1. Prikaz aritmetike operacije u zavisnosti od broja operanada

Broj operanada	Aritmetička operacija
4	a*b+c*d
3	a*b+c
2	a*b

Ulazni i izlazni signali su potpuno isti kao i kod *aritm_komp_jedan*. Jedina razlika u odnosu na nju je dužina izlaza kao i pomoćni signali. Dužina izlazne reči je 65 bita da bi se pokrio najveći mogući broj koji se može dobiti kao rezultat izvršene aritmetičke operacije. Dužina operanada se menja u paketu kao i kod *aritm_komp_jedan*, i potrebno je pri instanciranju komponente u generic map delu deklarisati dužinu operanada.

Svi operandi se direktno prosleđuju signalu *pomoćni_ulaz* u svim slučajevima osim kada se obrađuju tri operanda. Tada *pomoćni_ulaz*(4) dobija vrednost 1, a operacija dobija izgled *a*b+c*1* . Selekcija se vrši na osnovu signala *broj_operanada*. Signal *broj_operanada* se takođe, kao i kod *aritm_komp_jedan*, koristi za upravljanje mogućim vrednostima izlaznog signala *wr_en_izl_bafer* koji se koristi u daljem delu dizajna.

Signali *proizvod1* i *proizvod3* se proširuju za po još jedan bit jer njihov zbir može da prevaziđe opseg vrednosti pokriven sopstvenom dužinom.

Kada se vrši aritmetička operacija sa dva operanda tada operacija dobija izgled *a*b+0*0* tj. *a*b* jer se operandi prosleđuju iz ulaznih bafera koji u slučaju da im dozvola za čitanje nije aktivirana na izlazu daju vrednost nula.

3.5. Paket

Sadrži komponente koje se koriste u projektu. Da bi se neka komponenta korisitila potrebno je uvesti paket u upotrebu.

Paket takođe sadrži i proceduru za generisanje takta [8], kao i korisnički definisan tip podataka *matrica1* da bi se olakšalo uvođenje i korišćenje promenljivih.

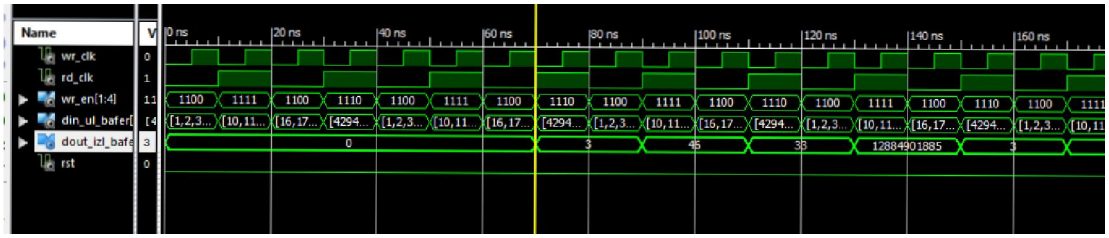
Tip *matrica1* predstavlja niz četiri 32-bitna vektora i koristi se za predstavljanje operanada koji dolaze na obradu i to na ulazu u ulazne bafere, izlazu iz njih i na ulazu u aritmetičku komponentu. Takođe se koristi za opis ulaza u KomParOb.

4. SIMULACIJA I SINTEZA

Simulacija je izvršena korišćenjem Isim alata kompanije kompanije Xilinx za Virtex7 familiju programabilnih čipova i to konkretno za xc7vx330t-1ffg1157 čip. Simulirani su: normalan način rada, ponašanje pri resetovanju, ponašanje pri zagušenju.

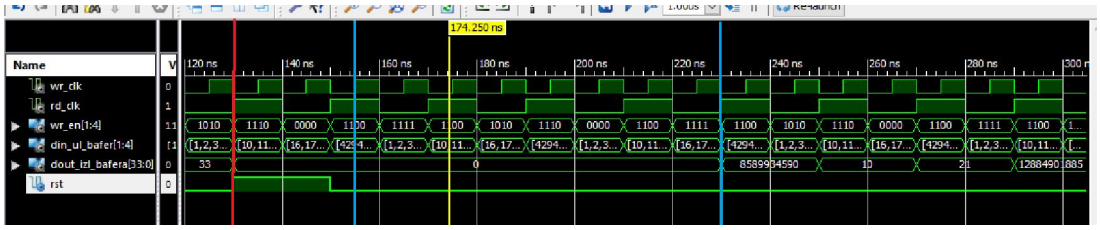
4.1. Komponenta za paralelnu obradu sa aritmetičkom komponentom jedan

Frekvencija upisa je 100 MHz a čitanja 50 MHz. Dubina svih bafera je 10. Dužina ulaznih operanada je 32 bita. Simuliran je rad bez zagušenja, resetovanje i rad sa zagušenjem.



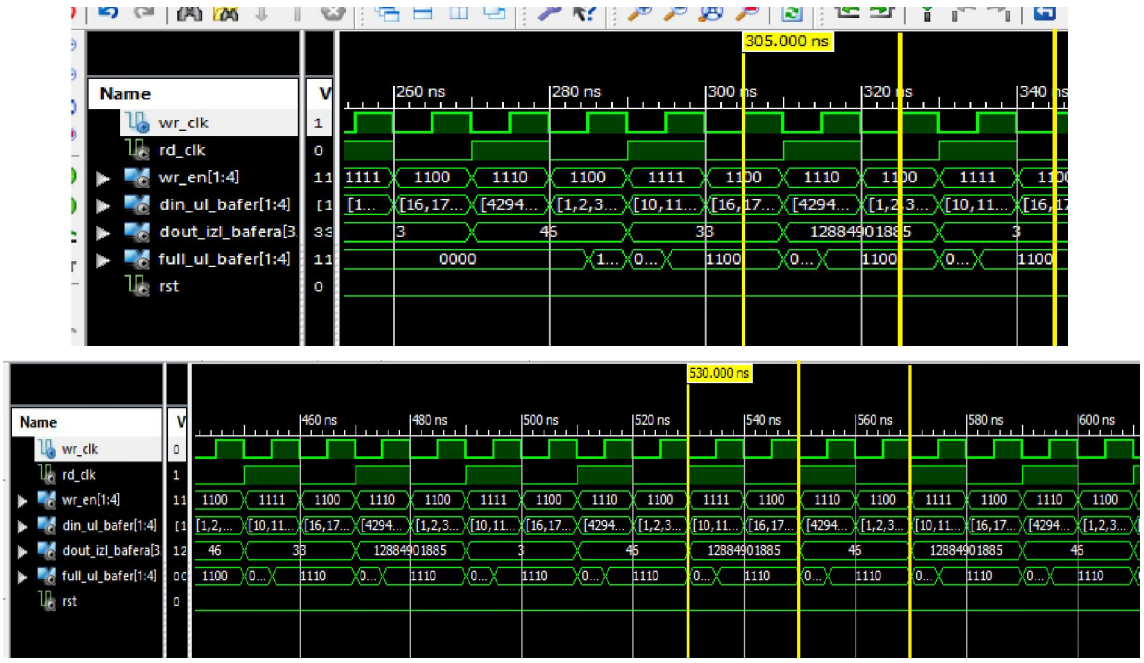
Slika 4.1.1. Simulacija rada

Na slici 4.1.1. se može videti simulacija rada. U prvom taktu upisa *wr_*clk na ulaz se nalaze operandi 1,2,3,4 i dozvola upisa *wr_en* 1100. To znači da će se u ulazne bafere upisati samo 1 i 2 čiji je rezultat sabiranja 3 što se može videti na izlazu. I ostali rezultati odgovaraju očekivanim. Žutom linijom je obeležen početak ispisa iz izlaznog bafera. Potrebno je 4 takta *rd_clk* da bi se rezultat pojavio na izlazu.



Slika 4.1.2. Simulacija rada sa resetom

Na slici 4.1.2. je prikazano ponašanje kada se aktivira reset što je obeleženo crvenom linijom. Očekivano ponašanje je da se po njegovom isključivanju u prvom sledećem taktu upisa, obeleženim plavom linijom (levo), nastavlja predviđeni posao što se može videti po rezultatu koji se dobija na izlazu, a što je obeleženo plavom linijom (desno).



Slika 4.1.3. Simulacija rada sa zagušenjem

Na slici 4.1.3. se može videti da će operandi koji dolaze na ulaz (obeleženi žutim linijama na slici 4.1.3. gore) u trenutku kada su baferi napunjeni biti odbačeni jer im nedostaju rezultati (obeleženo žutim linijama na slici 4.1.3 donji deo).

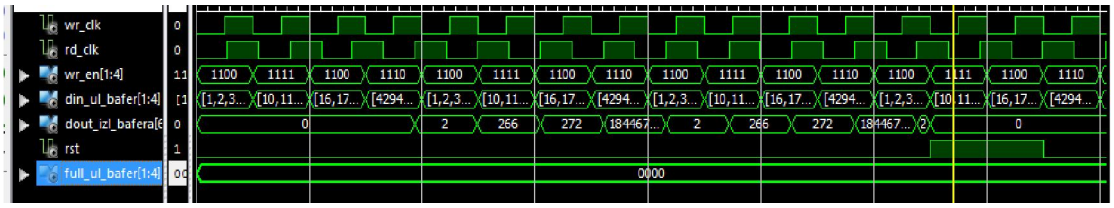
Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	1901	408000	0%
Number of Slice LUTs	1524	204000	0%
Number of fully used LUT-FF pairs	1039	2386	43%
Number of bonded IOBs	169	600	28%
Number of BUFG/BUFGCTRLs	2	32	6%

Slika 4.1.4. Zauzeće resursa FPGA čipa

Resursi koje rešenje zauzima su gotovo zanemarljivi što se može videti na slici 4.1.4. To znači da je moguće realizovati više ovakvih aritmetičkih funkcija na samo jednom čipu. Povećanjem dubine bafera se povećava i količina iskorišćenih resursa.

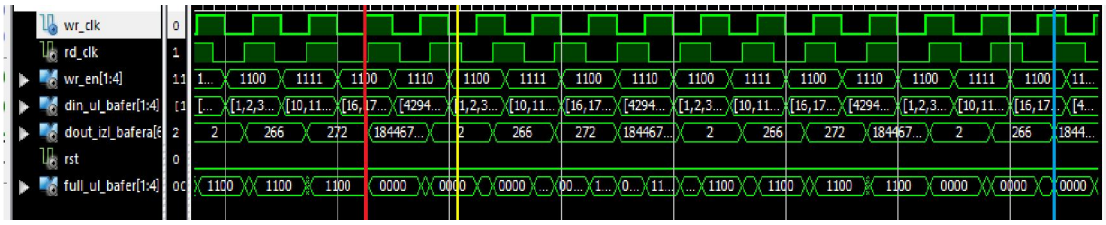
4.2. Komponenta za paralelnu obradu sa aritmetičkom komponentom dva

Frekvencija upisa je 100 MHz, a čitanja 90 MHz. Frekvencija čitanja je promenjena u odnosu na aritmetičku komponentu jedan da bi se prikazao način rada i kada su frekvecije bliske jedna drugoj. Dubina svih bafera je i dalje 10. Simuliran je rad bez zagušenja, resetovanje i rad sa zagušenjem.



Slika 4.2.1. Primer rada sa resetovanjem

Na slici 4.2.1. se može videti da se operacija regularno izvršava sve dok se sistem ne resetuje.



Slika 4.2.2. Primer rada sa zagušenjem

Na slici 4.2.2. se može primetiti da će se na uzlaznu ivicu takta upisa, obeleženog crvenom linijom, ulazni podaci odbaciti. Primetiti da rezltat koji bi trebao da se dobije (272) nedostaje (obeleženo plavom linijom).

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	2207	408000	0%
Number of Slice LUTs	1018	204000	0%
Number of fully used LUT-FF pairs	524	2701	19%
Number of bonded IOBs	200	600	33%
Number of BUFG/BUFGCTRLs	2	32	6%
Number of DSP48E1s	8	1120	0%

Slika 4.2.3. Zauzeće resursa FPGA čipa

Na slici 4.2.3. je prikazan procenat zauzeća resursa čipa. FPGA čipovi se sastoje od CLB-ova koji su opet podeljeni na slajsove. Konkretno, ovaj čip (xc7vx330t) sadrži 51000 slajsova [9], od kojih svaki sadrži po 4 LUT-a (*LookUp* Table) i po 8 flip-flopova kao i dodatnu logiku. Resursi koje ova komponenta zauzima su manji od 1%.

4.2.1. Resursi za slučaj povećane dubine bafera

Ovde je dat samo prikaz zauzeća resursa kada se dubina bafera promeni na 1024. Ova vrednost je odabrana kao difolt vrednost koju ISE Core Generator nudi za svoje FIFO bafere.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	201026	408000	49%
Number of Slice LUTs	73614	204000	36%
Number of fully used LUT-FF pairs	155	274485	0%
Number of bonded IOBs	200	600	33%
Number of BUFG/BUFGCTRLs	3	32	9%
Number of DSP48E1s	8	1120	0%

Slika 4.2.1.1. Zauzeće resursa FPGA čipa

Na slici 4.2.1.1 se može videti značajno uvećanje zauzetosti resursa čipa samo kao posledica uvećanja veličine bafera. Gotovo 50% čipa je iskorišćeno.

5. ZAKLJUČAK

Simulacija i sinteza pokazuju da je hardverska paralelizacija aritmetičkih operacija moguća. Sledeći korak bi bio modifikovanje rešenja tako da obrađuje operande koji pripadaju skupu realnih brojeva. Dalja moguća modifikacija bi bilo realizovanje složenijih algoritama obrade ili kroz njihovu kvantifikaciju tj. rastavljanje na prostije komponente i upotrebu već postojećih rešenja kroz umrežavanje ili, ukoliko je to moguće, njihovom direktnom implementacijom na jednom čipu. Takođe bi se trebalo pozabaviti načinom povezivanja ovakvog rešenja sa računarskim sistemima opšte namene.

Pri dizajniranju bi trebalo voditi računa u koju namenu se rešenje koristi. To može biti obrada na računaru opšte namene gde se javlja razlika u brzini rada magistrale i kartice na kojoj bi bilo implementirano rešenje. Druga potencijalna primena bi bila udaljena obrada preko lokalne ili globalne mreže ukoliko bi se za tako nečim ukazala potreba. Tada bi naravno bilo potrebno prilagoditi dizajn sa radom mrežnih interfejsa. U suštini se može koristiti za rešavanje problema gde postoji razlika u frekvenciji kojom treba obraditi podatke i frekvenciji kojom oni dolaze na obradu.

Naravno treba voditi računa i o samim fizičkim kapacitetima čipova koji se koriste za realizaciju.

Krajnji cilj je bio kreiranje modularnog rešenja koje bi bilo relativno lako izmeniti i prilagoditi potrebi. Tako su primera radi fifo baferi napravljeni da se lako skaliraju, a aritmetičke komponente su realizovane kao zasebni entiteti da bi se mogli menjati po potrebi.

LITERATURA

[1] BEE2: A High-End Reconfigurable Computing System; Chen Chang, John Wawrzynek, and Robert W. Brodersen University of California, Berkeley https://pdfs.semanticscholar.org/b521/f23e119a38fcd20016cc0379fe705757d478.pdf

[2] BEE2: A High-End Reconfigurable Computing System; Chen Chang, John Wawrzynek, and Robert W. Brodersen University of California, Berkeley ; strana br. 2 (115) ; druga kolona ; pretposlednji pasus https://pdfs.semanticscholar.org/b521/f23e119a38fcd20016cc0379fe705757d478.pdf

[3] http://www.copacobana.org/

[4] http://www.pactxpp.com/

[5] https://link.springer.com/article/10.1023/A:1024499601571

[6] http://www.recoresystems.com/fileadmin/downloads/Product_briefs/RS-MTP01-PB_R5.pdf

[7] http://gram.eng.uci.edu/morphosys/docs/JVSP.pdf

[8] Stack overflow internet strana http://stackoverflow.com/questions/17904514/vhdl-how-should-i-create-a-clock-in-a-testbench

[9] Virtex-7 resursi https://www.xilinx.com/support/documentation/selection-guides/virtex7-product-table.pdf

[10] Zoran Čiča, Programiranje komunikacionog hardvera, poglavlja 1 i 2 http://telekomunikacije.etf.rs/predmeti/te4ks/docs/PKH/PKH_01.pdf http://telekomunikacije.etf.rs/predmeti/te4ks/docs/PKH/PKH_02.pdf

A. KOD KOMPONENTI REALIZOVAN U VHDL-U

A.1. FIFOBafer

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use work.paket.all;

entity FIFOBafer is
generic(
	duzina_reci:integer;
	dubina_bafera:integer
);
port(
	wr_clk:in std_logic;
	rd_clk:in std_logic;
	din:in std_logic_vector(duzina_reci-1 downto 0):=(others=>'0');
	dout:out std_logic_vector(duzina_reci-1 downto 0):=(others=>'0');
	wr_en:in std_logic:='0';
	rd_en:in std_logic:='0';
	full:out std_logic:='0';
	empty:out std_logic:='1';
	rst:in std_logic:='0'
);
end FIFOBafer;

architecture shema of FIFOBafer is
	type matrica_podataka is array(0 to dubina_bafera-1) of std_logic_vector
(duzina_reci-1 downto 0);
	signal lokacija_upisa:integer:=0;
	signal lokacija_citanja:integer:=0;
	signal razlika1,razlika2:integer:=0;
	signal prazan,dozvola_upisa:std_logic:='1';
	signal pun,pun1,pun2,dozvola_citanja:std_logic:='0';
	signal memorija:matrica_podataka:=((others=>(others=>'0')));
begin



process(wr_clk,rst,dozvola_upisa)--proces upisa
begin
	if(rst='1')then
		lokacija_upisa<=0;
		memorija<=((others=>(others=>'0')));
	elsif(wr_clk' event and wr_clk='1' and dozvola_upisa='1')then
			memorija(lokacija_upisa)<=din;
			lokacija_upisa<=(lokacija_upisa+1)mod dubina_bafera;
	end if;
end process;
```

```
process(rd_clk,rst,dozvola_citanja)--proces citanja
begin
	if(rst='1')then
		lokacija_citanja<=0;
		dout<=(others=>'0');
	elsif(rd_clk' event and rd_clk='1' and dozvola_citanja='1')then
			dout<=memorija(lokacija_citanja);
			lokacija_citanja<=(lokacija_citanja+1)mod dubina_bafera;
	elsif(rd_clk' event and rd_clk='1')then  dout<=(others=>'0');
	end if;
end process;


razlika1<=lokacija_upisa-lokacija_citanja;
razlika2<=lokacija_citanja-lokacija_upisa;

with razlika1 select
	pun1<='1' when (dubina_bafera-1),
	'0' when others;
with razlika2 select
	pun2<='1' when 1,
	'0' when others;

pun<=pun1 or pun2;
full<=pun;

with pun select
	dozvola_upisa<=wr_en when '0',
	'0' when others;

with razlika1 select
	prazan<='1' when 0,
	'0' when others;

empty<=prazan;

with prazan select
	dozvola_citanja<=rd_en when '0',
	'0' when others;


end shema;
```

A.2. KomParOb

```
library ieee;
use ieee.std_logic_1164.all;
use work.paket.all;

entity KomParOb is
port(
	wr_clk,rd_clk:in std_logic;
	din_ul_bafer:in matrica1;
	dout_izl_bafera:out std_logic_vector(33 downto 0);
	wr_en:in std_logic_vector(1 to 4):="0000";
	rst:in std_logic:='0'
);
end KomParOb;
```

```
architecture shema of KomParOb is
	signal dout_ul_bafer:matrica1:=((others=>(others=>'0')));
	signal
wr_en_ul_bafer,rd_en_ul_bafer,full_ul_bafer,empty_ul_bafer:std_logic_vector(1 to
4):="0000";
	signal provera_upisa:std_logic_vector(1 to 2):="00";

	signal dout_bbo:std_logic_vector(1 to 4):="0000";
	signal
wr_en_bbo,wr_en_bbo1,wr_en_bbo2,rd_en_bbo,full_bbo,empty_bbo:std_logic:='0';

	signal wr_en_izl_bafera:std_logic:='0';
	signal dout_aritm_komp:std_logic_vector(33 downto 0):=(others=>'0');

	signal rd_en_izl_bafera,full_izl_bafer:std_logic:='0';
	signal empty_izl_bafer:std_logic:='1';

	signal broj_operanada:std_logic_vector(1 to 4):="0000";
begin


primerci_ulaznih_bafera:for i in 1 to 4 generate
primerak_ulaznog_bafera:FIFOBafer
generic map(
	duzina_reci=>32,
	dubina_bafera=>10
)
port map(
	wr_clk=>wr_clk,
	rd_clk=>rd_clk,
	din=>din_ul_bafer(i),
	dout=>dout_ul_bafer(i),
	wr_en=>wr_en_ul_bafer(i),
	rd_en=>rd_en_ul_bafer(i),
	full=>full_ul_bafer(i),
	empty=>empty_ul_bafer(i),
	rst=>rst
);
end generate;

primerak_bafera_broja_operanada:FIFOBafer
generic map(
	duzina_reci=>4,
	dubina_bafera=>10
)
port map(
	wr_clk=>wr_clk,
	rd_clk=>rd_clk,
	din=>wr_en,
	dout=>dout_bbo,
	wr_en=>wr_en_bbo,
	rd_en=>rd_en_bbo,
	full=>full_bbo,
	empty=>empty_bbo,
	rst=>rst
);

primerak_izlaznog_bafera:FIFOBafer
generic map(
```

```
	duzina_reci=>34,
	dubina_bafera=>10
)
port map(
	wr_clk=>rd_clk,
	rd_clk=>rd_clk,
	din=>dout_aritm_komp,
	dout=>dout_izl_bafera,
	wr_en=>wr_en_izl_bafera,
	rd_en=>rd_en_izl_bafera,
	full=>full_izl_bafer,
	empty=>empty_izl_bafer,
	rst=>rst
);

primerak_aritmeticke_komponente:aritm_komp_jedan
generic map(
	duzina_operanada=>32
)
port map(
	din_aritm_komp=>dout_ul_bafer,
	dout_aritm_komp=>dout_aritm_komp,
	broj_operanada=>broj_operanada,
	wr_en_izl_bafera=>wr_en_izl_bafera
);

process(rd_clk)--uskladjivanje izlaza bafera broja operanada sa izlazom ulaznih
bafera
begin
	if (rd_clk' EVENT and rd_clk='1')then
		case (dout_bbo) is
			when others=>broj_operanada<=dout_bbo;
		end case;
	end if;
end process;

-------------------------------------
--upravljanje ulaznim baferima
-------------------------------------
with full_ul_bafer select-- provera popunjenosti
	provera_upisa(1)<='0' when "1100"|"1110"|"1111" ,
	'1' when others;
with wr_en select-- provera ispravnosti dozvola za upis
	provera_upisa(2)<='1' when "1111"|"1110"|"1100",
	'0' when others;

with provera_upisa select--dozvola upisa
	wr_en_ul_bafer<=wr_en when "11",
	"0000" when others;

rd_en_ul_bafer<=dout_bbo;-- dozvola citanja se uzima iz bafera broja operanada


----------------------------------------
--upravljanje baferom broja opranada
----------------------------------------
with full_ul_bafer select--provera popunjenosti
	wr_en_bbo1<='0' when "1111"|"1110"|"1100",
	'1' when others;
```

```
with wr_en select-- provera ispravnosti dozvola za upis (broja operanada)
      wr_en_bbo2<='1' when "1111"|"1110"|"1100",
      '0' when others;

wr_en_bbo<=wr_en_bbo1 and wr_en_bbo2;--dozvola upisa

with empty_bbo select-- dozvola citanja
      rd_en_bbo<='1' when '0',
      '0' when others;



-------------------------------
-- citanje iz izlaznog bafera
-------------------------------
with empty_izl_bafer select
      rd_en_izl_bafera<='1' when '0',
      '0' when others;



end shema;
```

A.3. Aritm_komp_jedan

```
library ieee;
use ieee.std_logic_1164.all;
use work.paket.all;
use ieee.std_logic_unsigned.all;


entity aritm_komp_jedan is
generic(
      duzina_operanada:integer
);
port(
      din_aritm_komp:in matrica1;
      dout_aritm_komp:out std_logic_vector(duzina_operanada+1 downto 0);
      broj_operanada:in std_logic_vector(1 to 4):="0000";
      wr_en_izl_bafera:out std_logic:='0'
);
end aritm_komp_jedan;

architecture shema of aritm_komp_jedan is
      type matrica_podataka is array(1 to 4) of std_logic_vector
(duzina_operanada+1 downto 0);
      signal pomocni_ulaz:matrica_podataka:= ((others=> (others=>'0')));
begin


with broj_operanada select-- kontrola upisa u izlazni bafer
      wr_en_izl_bafera<='0' when "0000",
      '1' when others;

pomocni_ulaz(1)<="00"&din_aritm_komp(1);
pomocni_ulaz(2)<="00"&din_aritm_komp(2);
pomocni_ulaz(3)<="00"&din_aritm_komp(3);
```

```
pomocni_ulaz(4)<="00"&din_aritm_komp(4);

dout_aritm_komp<=pomocni_ulaz(1)+pomocni_ulaz(2)+pomocni_ulaz(3)+pomocni_ulaz(4)
;


end shema;
```

A.4. Aritm_komp_dva

```
library ieee;
use ieee.std_logic_1164.all;
use work.paket.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

----------------------------------------------------
-- operacija koju aritm_komp_dva obavlja je din(1)*din(2)+din(3)*din(4)
----------------------------------------------------
entity aritm_komp_dva is
generic(
      duzina_operanada:integer
);
port(
      din_aritm_komp:in matrica1;
      dout_aritm_komp:out std_logic_vector(duzina_operanada*2 downto 0);
      broj_operanada:in std_logic_vector(1 to 4):="0000";
      wr_en_izl_bafera:out std_logic:='0'
);
end aritm_komp_dva;

architecture shema of aritm_komp_dva is
      signal pomocni_ulaz:matrica1:=((others=> (others=>'0')));
      signal proizvod1,proizvod2:std_logic_vector((duzina_operanada*2)-1 downto
0):=(others=>'0');
      signal proizvod3,proizvod4:std_logic_vector(duzina_operanada*2 downto
0):=(others=>'0');

begin


with broj_operanada select
      wr_en_izl_bafera<='0' when "0000",
      '1' when others;

pomocni_ulaz(1)<=din_aritm_komp(1);
pomocni_ulaz(2)<=din_aritm_komp(2);
pomocni_ulaz(3)<=din_aritm_komp(3);

with broj_operanada select
      pomocni_ulaz(4)<=conv_std_logic_vector(1, duzina_operanada) when "1110",
      din_aritm_komp(4) when others;


proizvod1<=pomocni_ulaz(1)*pomocni_ulaz(2);
proizvod3<='0' & proizvod1;
```

```
proizvod2<=pomocni_ulaz(3)*pomocni_ulaz(4);
proizvod4<='0' & proizvod2;

dout_aritm_komp<=proizvod3+proizvod4;

end shema;
```

A.5. Paket

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;



package paket is
	type matrica1 is array(1 to 4)of std_logic_vector(31 downto 0);

component FIFOBafer is
generic(
	duzina_reci:integer;
	dubina_bafera:integer
);
port(
	wr_clk:in std_logic;
	rd_clk:in std_logic;
	din:in std_logic_vector(duzina_reci-1 downto 0):=(others=>'0');
	dout:out std_logic_vector(duzina_reci-1 downto 0):=(others=>'0');
	wr_en:in std_logic:='0';
	rd_en:in std_logic:='0';
	full:out std_logic:='0';
	empty:out std_logic:='1';
	rst:in std_logic:='0'
);
end component;

component aritm_komp_jedan is
generic(
	duzina_operanada:integer
);
port(
	din_aritm_komp:in matrica1;
	dout_aritm_komp:out std_logic_vector(duzina_operanada+1 downto 0);
	broj_operanada:in std_logic_vector(1 to 4):="0000";
	wr_en_izl_bafera:out std_logic:='0'
);
end component;

component aritm_komp_dva is
generic(
	duzina_operanada:integer
);
port(
	din_aritm_komp:in matrica1;
	dout_aritm_komp:out std_logic_vector(duzina_operanada*2 downto 0);
	broj_operanada:in std_logic_vector(1 to 4):="0000";
	wr_en_izl_bafera:out std_logic:='0'
);
end component;
```

```
component KomParOb is
port(
      wr_clk,rd_clk:in std_logic;
      din_ul_bafer:in matrica1;
      dout_izl_bafera:out std_logic_vector(33 downto 0);
      wr_en:in std_logic_vector(1 to 4):="0000";
      rst:in std_logic:='0'
);
end component;

procedure clk_gen(signal clk : out std_logic; constant FREQ : real);

end paket;

package body paket is

procedure clk_gen(signal clk : out std_logic; constant FREQ : real) is
      constant PERIOD    : time := 1 sec / FREQ;        -- Full period
      constant HIGH_TIME : time := PERIOD / 2;          -- High time
      constant LOW_TIME  : time := PERIOD - HIGH_TIME;  -- Low time; always >=
HIGH_TIME
begin
      loop
            clk <= '0';
            wait for HIGH_TIME;
            clk <= '1';
            wait for LOW_TIME;
      end loop;
end procedure;

end paket;
```