Univerzitet u Sarajevu
Elektrotehnički fakultet Sarajevo
Odsjek za telekomunikacije

DPPSN – Realizacija VHDL testnih okruženja za testiranje referentnog 10Gbps Ethernet preklopnika

Projektni zadatak iz predmeta Arhitekture paketskih čvorišta

Ćutahija Zerina, 1685/17085

Hasanbegović Selma, 1574/17753

Mahovac Nerman, 1575/ 17919

Repeša Almin, 1684/17550

Velić Nejra, 1634/17313

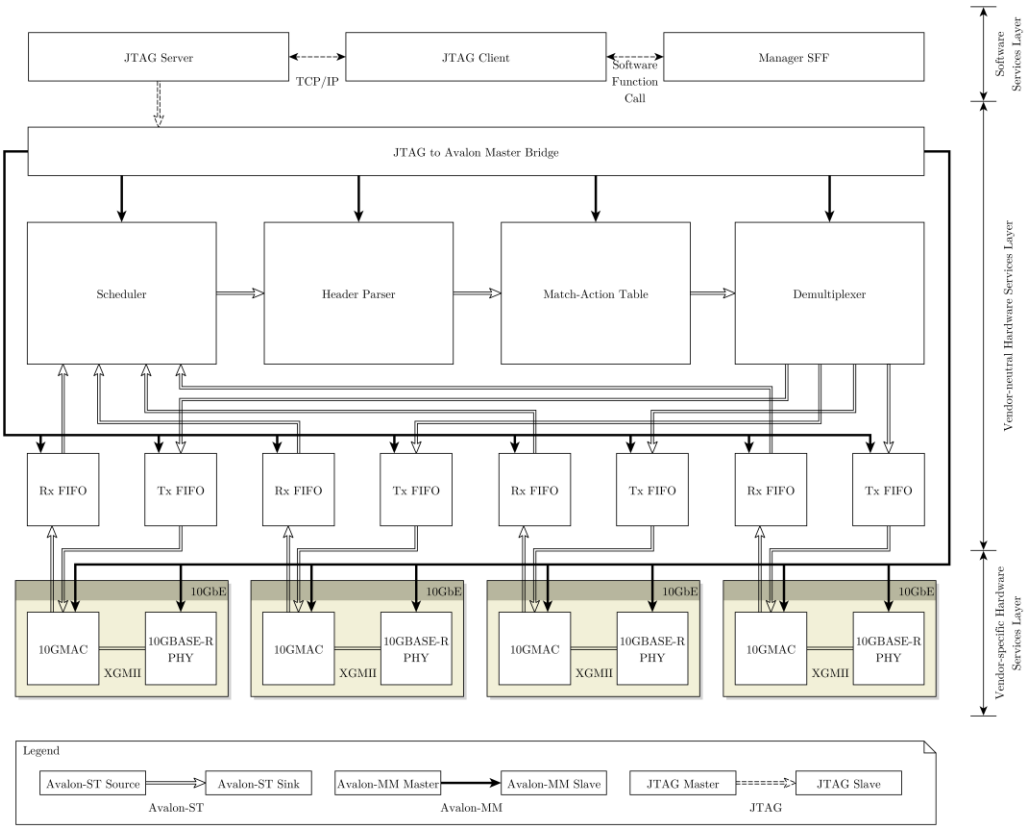
Sarajevo, januar 2020.

Postavka zadatka

1. Analizirati i teorijski obraditi komponente referentnog 10Gbps Ethernet preklopnika [1] (poslužitelj ulaznih redova čekanja, parser zaglavlja paketa, podudarna tabela, izlazni demultiplekser) datog na [2].
2. Napisati VHDL testna okruženja za testiranje rada sljedećih sklopova: dppsn_scheduler, dppsn_header_parser, dppsn_match_action_table i dppsn_demux.
3. Izvršiti testiranje komponenti preklopnika upotrebom ModelSim simulatora. Za formiranje sadržaja testnih signala na Avalon-ST i Avalon-MM sučeljima, preporučuje se upotreba [3] i [4]. Prilikom testiranja pretpostaviti da između dva sukcesivna paketa (na ulaznim Avalon-ST sučeljima) postoji najmanje jedan takt interval pauze. Frekvencija takt signala iznosi 156,25 MHz.
4. Ukoliko se prilikom testiranja utvrdi da postoje greške u dizajnu u nekom od navedenih sklopova, potrebno je popraviti uočene greške i dokazati upotrebom simulatora da korigovani sklopovi ispravno funkcionišu.

DPPSN – Duboko programabilno paketsko čvorište

Ogledni preklopnik, prikazan na slici 1, omogućava programabilnu komutaciju 10G Ethernet saobraćaja. Za realizaciju oglednog preklopnika izabrana je hibridna platforma bazirana na kombinaciji softverske realizacije koja se izvršava na procesorima opce namjene i hardverske implementaciji na FPGA čipu. Budući da je realizacija hardverskih servisnih slojeva predložene arhitekture bazirana na razvojnoj platformi sa Intel-ovim FPGA čipom, za interkonekciju komponenti unutar arhitekture odabrana je familija Avalon sučelja [1].

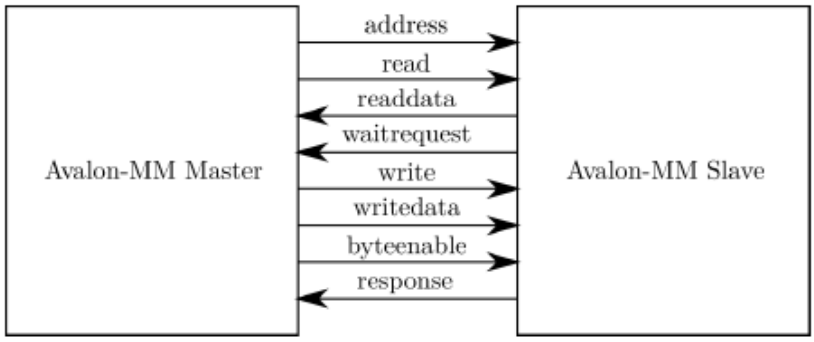


Slika 1. Arhitektura oglednog preklopnik [1]

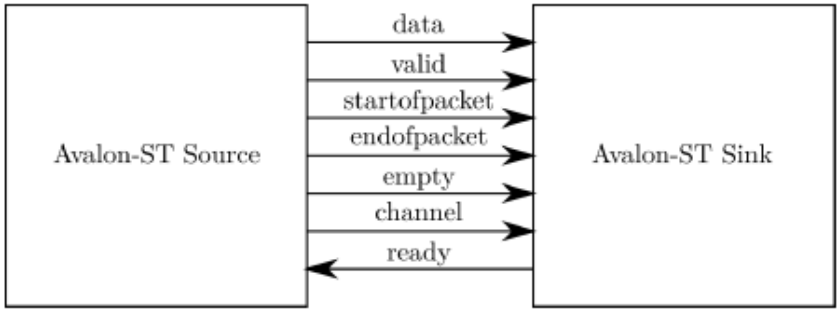
U interkonekciji komponenti mogu se uočiti dva načina prenosa informacija:

1. sporadični prenos informacija (npr. prenos kontrolnih i statusnih informacija) – koji je realiziran upotrebom Avalon-MM (*Avalon Memory-Mapped*) sučelja i
2. kontinualni prenos informacija (npr. prenos paketa) – koji je realiziran upotrebom Avalon-ST (*Avalon Streaming*) sučelja.

Za sporadično čitanje ili upis podataka (u memorijske registre) između nalogodavca (*Master*) i izvršioca (*Slave*) putem Avalon-MM sučelja, a signali koji se koriste za prenos su navedeni na slici 2, dok se za prenos paketa između izvora (*Source*) i odredišta (*Sink*) koristi Avalon-ST sučelje, a signali koji se koriste su dati na slici 3 [1].



Slika 2. Tipični signali za prenos podataka putem Avalon-MM sučelja [1]



Slika 3. Tipični signali za prenos podataka putem Avalon-ST sučelja [1]

Osnovne funkcionalnosti oglednog preklopnika su:

- posluživanje dolaznih redova čekanja,
- parsiranje zaglavlja paketa,
- održavanje i pretraga tabele prosljeđivanja,
- raspoređivanje paketa u odlazne redove čekanja,
- upravljanje i nadzor nad procesima prosljeđivanja paketa.

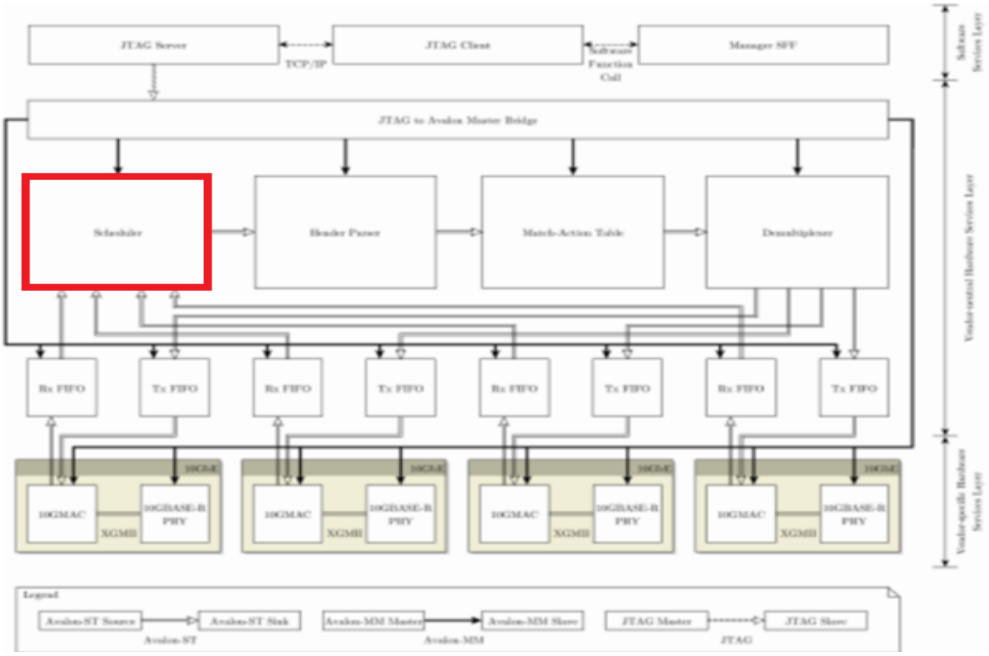
Dolazni i odlazni redovi čekanja su implementirani upotrebom Avalon-ST Single Clock FIFO jezgre. Punjenje i pražnjenje redova čekanja se obavlja putem Avalon-ST sučelja, dok se putem Avalon-MM sučelja može vršiti nadzor nad dužinom reda čekanja.

U nastavku će biti obrađene i testirane sljedeće komponente oglednog preklopnika pomoću VHDL testnog okruženja: Scheduler, Header Parser, Match-Action Tabel i Demultiplexer. Prilikom testiranja je pretpostavljeno da između dva sukcesivna paketa postoji najmanje jedan takt interval pauze. Frekvencija takt signala je $156.25\,MHz$, što je iskorišteno prilikom

testiranja komponenti gdje se, prilikom kreiranja zasebnog procesa zaduženog za simulaciju takt signala, uzela vrijednost perioda takta $clk\_period = 6.4\,ns\ (1/156.25\,MHz)$.

1. SCHEDULER

Prva u nizu komponenti koje su testirane u ovom projektnom zadatku je Scheduler. Njegov položaj u DPPSN arhitekturi je prikazan na slici 4.



Slika 4. Položaj Scheduler-a u DPPSN arhitekturi

Glavni zadatak Scheduler-a je *posluživanje dolaznih redova čekanja* jednim od tri implementirana načina posluživanja:

- Prioritetno posluživanje
- Round Robin posluživanje
- Deficitarni Round Robin način posluživanja.

Odabir načina posluživanja se vrši putem selektorskog signala koji je sastavni dio arhitekture komponente, dok se sami paketi od dolaznih redova čekanja prema Header Parser-u prenose putem Avalon-ST sučelja i da će paketi biti posluženi samo onda kada je Header Parser spreman da ih primi što se prema Avalon-ST sučelju može kontrolisati korištenjem signala **out_ready**. Bitno je napomenuti da u ovakvoj arhitekturi DPPSN-a Scheduler poslužuje četiri dolazna reda čekanja i da u jednom trenutku može biti poslužen samo jedan paket iz jednog od redova čekanja.

Najlakši način da se prepozna iz kojeg reda čekanja dolazi paket koji se poslužuje je posmatranje zadnja četiri bita signala **in_channel** na sljedeći način:

- **Dolazni red čekanja in0** – "1000"
- **Dolazni red čekanja in1** – "0100"
- **Dolazni red čekanja in2** – "0010"
- **Dolazni red čekanja in3** – "0001"

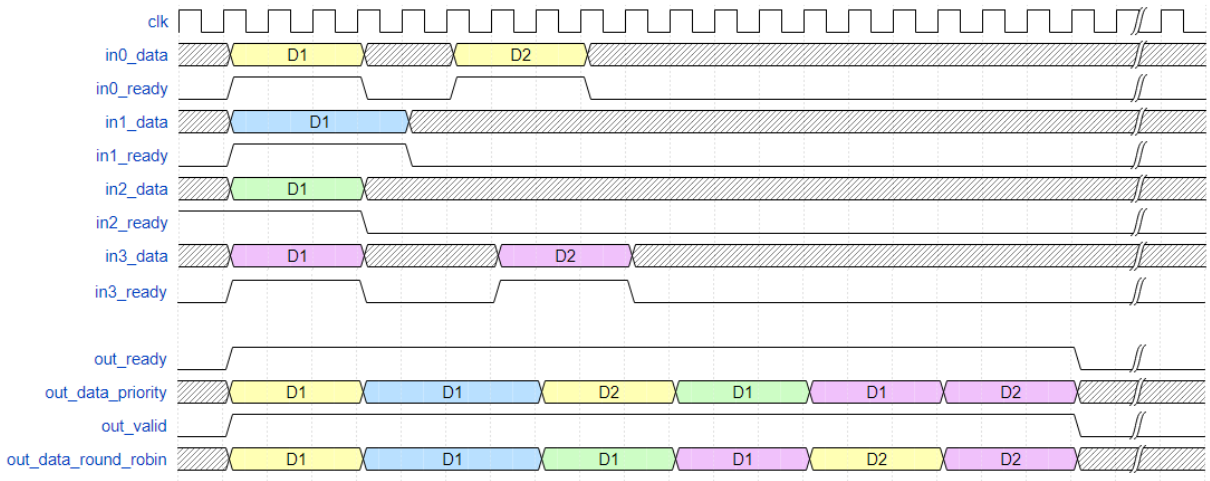
Da bi uopšte mogli diskutovati o ispravnosti rada ove komponente ukratko ćemo objasniti način rada svakog od mogućih načina posluživanja.

Prioritetno posluživanje kao što mu samo ime govori stavlja različite prioritete za različite redove čekanja i uvijek prvo poslužuje one pakete koji dolaze iz redova čekanja sa većim prioritetom. Detaljnom analizom VHDL koda ove komponente smo zaključili da najveći prioritet imaju paketi u prvom dolaznom redu čekanja (**in0**), dok najmanji prioritet imaju paketi iz četvrtog reda posluživanja (**in3**), osim što je zaključeno da porastom indeksa reda čekanja dolazi do smanjenja prioriteta odgovarajućeg reda može se zaključiti i da ne dolazi do prekidanja već započetog posluživanja paketa ukoliko tokom njegovog posluživanja dođe paket u red čekanja sa većim prioritetom.

Round Robin način posluživanja je zapravo način cikličkog posluživanja redova čekanja. Naime kod ovog načina posluživanja se nakon posluživanja paketa iz prvog dolaznog reda čekanja (**in0**) poslužuje paket iz prvog sljedećeg nepraznog reda čekanja sa većim indeksom, bez obzira da li u nultom redu čekanja postoji još paketa koji se trebaju poslužiti. Nakon što se provjeri da li ima paketa u četvrtom redu čekanja (**in3**) i eventualnog posluživanja jednog paketa ponovo se provjerava da li ima paketa u prvom redu čekanja i ciklus se ponavlja sve dok svi paketi ne budu posluženi.

Zbog toga što su neki redovi čekanja koristili pakete puno veće dužine nego ostali uveden je deficitarni Round Robin koji ima isti princip rada kao i Round Robin s tim što su uvedeni kvantumi. Paket iz određenog reda čekanja će biti poslužen tek onda kada njegova dužina bude manja ili jednaka od vrijednosti kvantuma. U VHDL kodu ove komponente je drugačije tretirana vrijednost kvantuma koji se također definišu putem signala **csr_writedata** i njihova vrijednost se smanjuje sa svakim novim posluživanjem paketa iz dolaznih redova čekanja dok ne dođe do vrijednosti nula kada opet kvantumi dobivaju vrijednost definisanu već pomenutim signalom.

Prije samog pisanja testnog okruženja za Scheduler, napravljen je dijagram, prikazan na slici 5, koji demonstira princip rada Scheduler-a i daje uvid u očekivane rezultate prilikom testiranja ove komponente.



Slika 5. Primjer načina rada Scheduler-a

Sa slike 5 se jasno može zaključiti da signal **out_ready** zapravo određuje da li će ova komponenta uopšte raditi, a na osnovu odabranog načina posluživanja i podatka da li uopšte

ima paketa i validnih podataka u njima za svaki od redova ponaosob, ove informacije se kontrolišu putem signala **in_startofpacket** i **in_valid**, se onda pojavljuje signal **inx_ready** gdje x predstavlja broj reda čekanja na osnovu kojeg se zna koji će paket i iz kog reda čekanja biti poslužen u posmatranom trenutku.

Scenario testiranja:

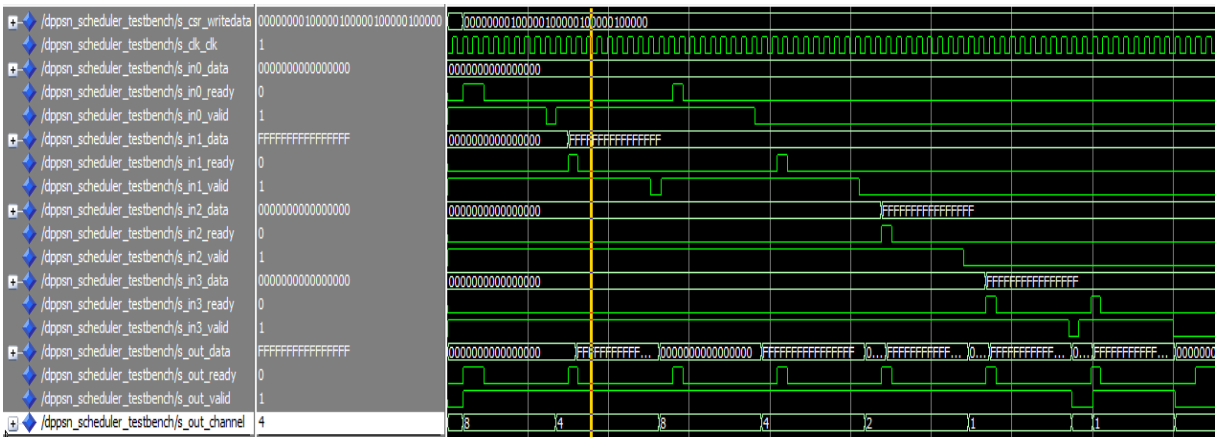
Nakon proučavanja same komponente zaključeno je da je potrebno ispitati eventualnu pogrešku selektorskog signala, ponašanje komponente ukoliko u istom takt intervalu dođu paketi u više različitih redova čekanja, pojavu paketa u redu čekanja većeg prioriteta prilikom posluživanja paketa iz rada čekanja koji je manjeg prioriteta kod prioritetnog načina posluživanja. Također, potrebno je ispitati i način preskakanja praznih redova čekanja kod Round Robin i Deficitarni Round Robin načina posluživanja, ukoliko ima više paketa u redu čekanja da li se poslužuje samo jedan kod Round Robin načina testiranja. Potrebno je i mijenjati vrijednost kvantuma i veličine paketa kod Deficitarnog Round Robin načina posluživanja.

1.1 PRIORITETNO POSLUŽIVANJE

Obzirom da za prioritetno posluživanje nije bitna veličina paketa u ovom scenariju su odabrani paketi najmanje moguće veličine: 8 bolokova od 64 bita. Da bi se vidjelo da li je pravilno implementiran ovaj način posluživanja potrebno je u početnom trenutku dovesti validan paket u svaki od redova čekanja, nakon dolaska cijelog paketa potrebno je čekati jedan takt interval i u tri od četiri reda čekanja (nulti, prvi i treći) ponovo dovesti paket. Ovim scenariom će se vidjeti da li se poslužuju paketi po prioritetu redova čekanja, također se može vidjeti da li se prekida posluživanje već započetog paketa ukoliko se pojavi paket u redu čekanja većeg prioriteta i da li se nakon posluživanja paketa iz reda čekanja manjeg prioriteta poslužuje paket iz reda čekanja većeg prioriteta ukoliko se pojavio u međuvremenu.

U prvih $10\ ns$ se također dovodi neispravan selektorsli signal kako bi se moglo vidjeti šta se dešava u sa komponentom u takvoj situaciji. Očekivanje je da komponenta ne radi ništa sve dok selektorski signal nije validan, a nakon toga u razmatranom slučaju redom trebaju biti posluženi paketi iz redova čekanja: paket iz prvog reda čekanja (**in0**), paket iz drugog reda čekanja (**in1**), paket iz prvog reda čekanja (**in0**), paket iz drugog reda čekanja (**in1)**, paket iz trećeg reda čekanja (**in2)**, paket iz četvrtog reda čekanja (**in3**) i paket iz četvrtog reda čekanja (**in3**), što je u skladu sa prethodno objašnjenim.

Prikaz signala nakon pisanja VHDL testne skripte je pirkazan na slici 6, gdje su radi preglednosti prikazani samo oni signali koji su neophodni da se vidi ispravan rad komponente.



Slika 6. Prikaz rada prioritetnog posluživanja

Radi jednostavnijeg praćenja pojedinih signala na dijagramu, signali **in0_data, in1_data, in2_data, in3_data, out_data** su prikazani u hexadecimalnom obliku, dok je signal **out_channel** prikazan u *unsigned* obliku pa su tako paketi koji dolaze iz reda čekanja nula prikazani brojem osam, paketi iz reda čekanja jedan su prikazani brojem četiri, paketi iz reda čekanja dva su prikazani brojem dva i paketi iz reda čekanja tri su prikazani brojem jedan. Detaljnim posmatranjem dijagrama došli smo do zaključka da prioritetno posluživanje radi ispravno što je i bio cilj datog razmatranja.

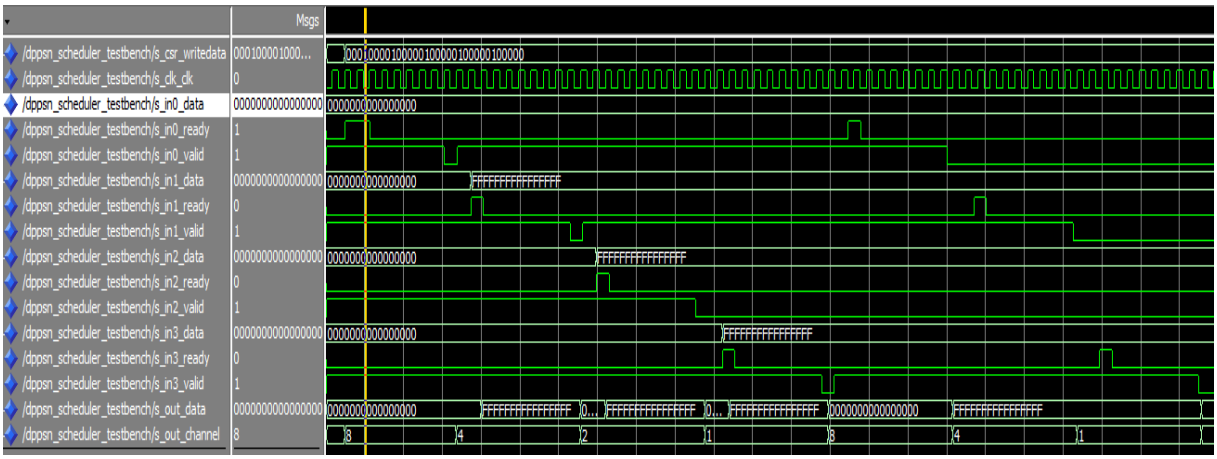
1.2 ROUND ROBIN POSLUŽIVANJE

Obzirom da za Round Robin posluživanje nije bitna veličina paketa i u ovom scenariju su odabrani paketi najmanje moguće veličine. Da bi se vidjelo da li je pravilno implementiran ovaj način posluživanja potrebno je u početnom trenutku dovesti validan paket u svaki od redova čekanja, nakon dolaska cijelog paketa potrebno je čekati jedan takt interval i u tri od četiri reda čekanja (prvi – **in0**, drugi – **in1** i četvrti – in3) ponovo dovesti paket. Ovim scenariom će se vidjeti da li se paketi poslužuju ciklički, da li se preskaču prazni redovi čekanja te da li se nakon posluživanja paketa iz posljednjeg reda čekanja poslužuje paket iz prvog reda čekanja.

U prvih 10 ns se također dovodi neispravan selektorsli signal kako bi se moglo vidjeti šta se dešava u sa komponentom u takvoj situaciji. Očekivanje je da komponenta ne radi ništa sve dok selektorski signal nije validan. Nakon toga u razmatranom slučaju redom trebaju biti posluženi paketi iz redova čekanja: paket iz prvog reda čekanja (**in0**), paket iz drugog reda čekanja (**in1**), paket iz trećeg reda čekanja (**in2**), paket iz četvrtog reda čekanja (**in3)**, paket iz prvog reda čekanja (**in0)**, paket iz drugog reda čekanja (**in1**) i paket iz četvrtog reda čekanja (**in3**), što je u skladu sa prethodno objašnjenim.

Prikaz signala nakon pisanja VHDL testne skripte je pirkazan na slici 7, gdje su radi preglednosti prikazani samo oni signali koji su neophodni da se vidi ispravan rad komponente.

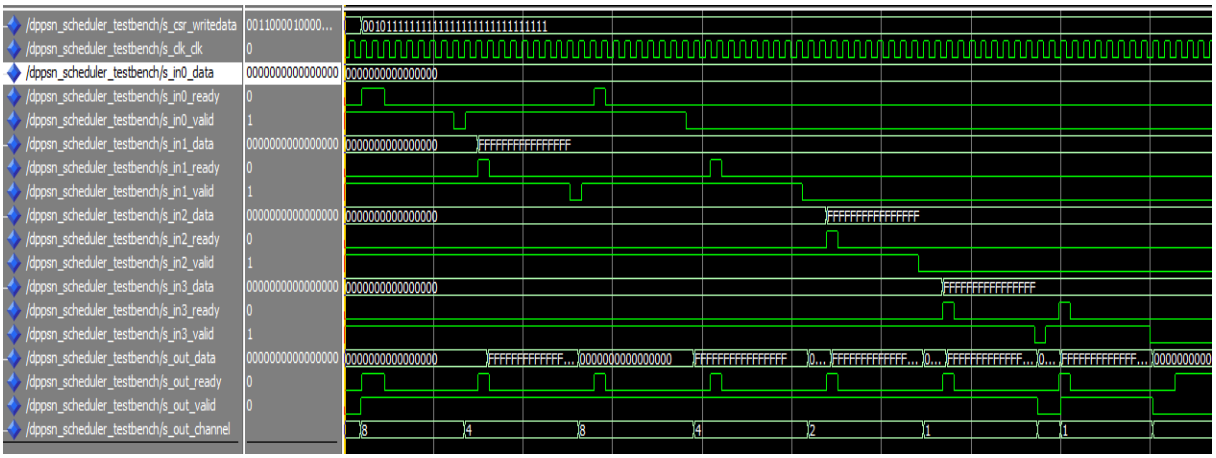
Obzirom da su identični signali kao u prethodnom slučaju, i ovde se prikazuju identični oblici za prikazivanje različitih signala. Također, kako se rezultati podudaraju sa očekivanim vrijednostima zaključujemo da i ovaj način posluživanja ispravno radi.



Slika 7. Prikaz rada Round Robin posluživanja

1.3. DEFICITARNI ROUND ROBIN NAČIN POSLUŽIVANJA

Kako je potrebno izvršiti više testiranja za ovaj način posluživanja, u prvom slučaju ćemo dovoditi pakete jednake dužine na isti način kao u prethodnim slučajevima, i u zavisnosti od veličine kvantuma koja kada je jednaka svi paketi bi trebali biti posluženi kao kod Round Robin načina posluživanja, a kada se promijeni vrijednost kvantuma neki paketi bi trebali čekati sa svojim posluživanjem. Ukoliko je vrijednost kvantuma jednaka za sve redove čekanja dobivamo testni scenarij kao sa slike 8.



Slika 8. Primjer rada Deficitarnog Round Robina sa jednakim kvantumima

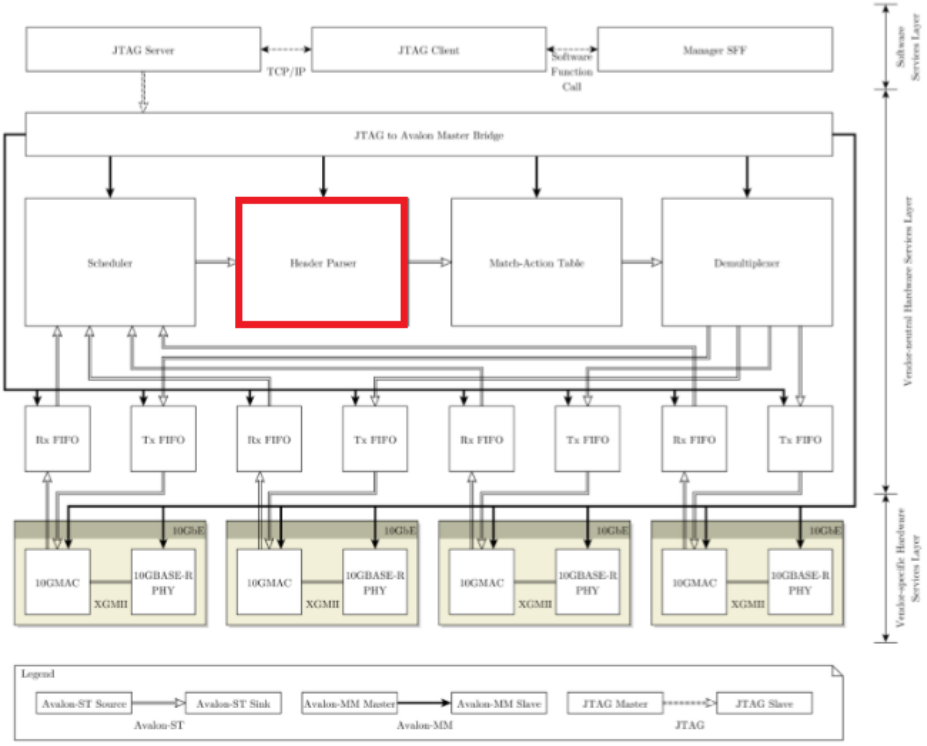
Vidimo da u ovom slučaju nismo dobili signale u skladu sa očekivanim vrijednostima i da pri jednakom odabiru kvantuma za sve redove čekanja (test daje jednake rezultate i za minimalnu i maksimalnu vrijednost kvantuma) ovaj način posluživanja radi kao i prioritetno posluživanje.

U nastavku je testiran rad ovog posluživanja za različitu vrijednost kvantuma za sve redove čekanja. Prilikom bilo koje kombinacije kvantuma dobije se grafik koji je prikazan na slici 8 koji odgovara prioritetnom načinu posluživanja, pa zaključujemo da ovaj način posluživanja nije ispravno implementiran u izvornom kodu koji smo trebali testirati, a zbog nedostatka vremena nismo bili u mogućnosti da pokušamo otkloniti pomenute probleme.

Prilikom testiranja ovog slučaja glavni akcenat se stavlja na varijaciju dužine paketa kao i veličine kvantuma. Osim tih osobina potrebno je da i ovaj način posluživanja posjeduje sve karakteristike kao i Round Robin način posluživanja.

2. HEADER PARSER

Drugi element, u koji signal dolazi iz Scheduler-a je Header Parser, prikazan na slici ispod:



Slika 9. Položaj Header Parser-a u DPPSN arhitekturi

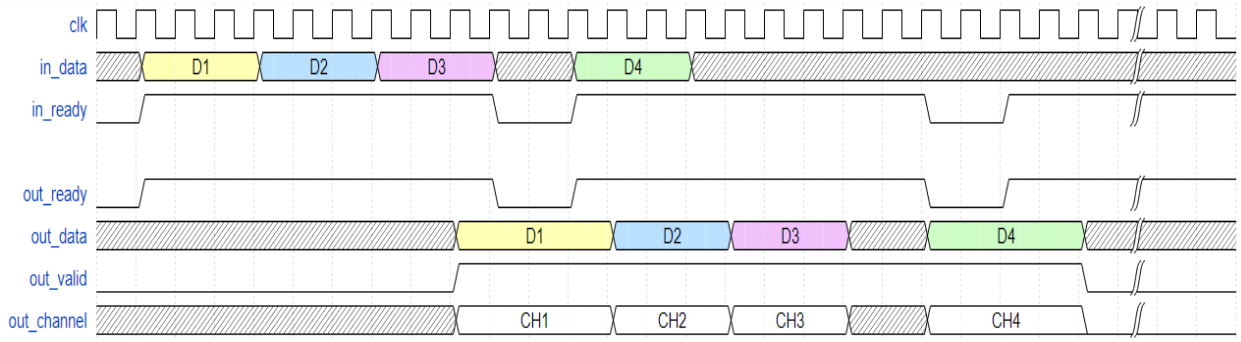
Osnovna uloga Header Parser-a je *parsiranje (raščlanjivanje) zaglavlja paketa.* Paketi u Header Parser dolaze iz Scheduler-a, gdje sklop za parsiranje zaglavlja paketa, u skladu sa konfiguracijom zadanom putem Avalon-MM sučelja, izdvaja odabrane oktete iz zaglavlja paketa i pridružuje ih signalizacijskom kanalu *channel* na izlaznom Avalon-ST sučelju.

Unutar VHDL koda za header parser definisani su sljedeći signali: **data, valid, startofpacket, endofpacket, empty.** To su ulazni signali u header parser, koji dolaze iz schedulera kanalom **in_channel**. Sa druge strane u skladu sa radom Avalon-MM sučelja komponenta match action table treba da šalje signal **ready**, u onom trenutku kada bude spremna da primi signal koji obradi header parser.

U arhitekturi header parsera definisan je proces osjetljiv na signale **reset_reset_n** i **clk_clk.** Ukoliko se desi da se sklop resetuje, odnosno da se signal **reset_reset_n** spusti na 0, svi signali treba da budu na 0, te da nema prenosa podataka. U suprotnom se na osnovu stanja u kojem se sistem nalazi, na uzlaznu ivicu **clk_clk** signala, te ukoliko je sljedeći sklop spreman za prijem (što se inicira signalom **out_ready**), vrši izdvajanje odabranog okteta paketa i pridruživanje izlaznom kanalu.

U sljedećem procesu se na uzlaznu ivicu **clk_clk** signala, te ukoliko je sljedeći sklop spreman za prijem (što se inicira signalom **out_ready**), uvodi kašnjenje od 8 takt intervala, te se ulazni signali **data, valid, startofpacket, endofpacket** i **empty**, dodjeljuju zakašnjelim signalima, koji se na kraju prosljeđuju na izlaz.

Na osnovu svega navedenog, prije samog testiranja izvršeno je crtanje dijagrama, prikazanog na slici 10, koji demonstrira rad sklopa:



Slika 10. Primjer načina rada header parsera

Na slici 10 je prikazan očekivani rad sklopa nakon testiranja. Izvršeno je slanje paketa D1, D2 i D3, te nakon pauze od 2 takt intervala i paketa D4.

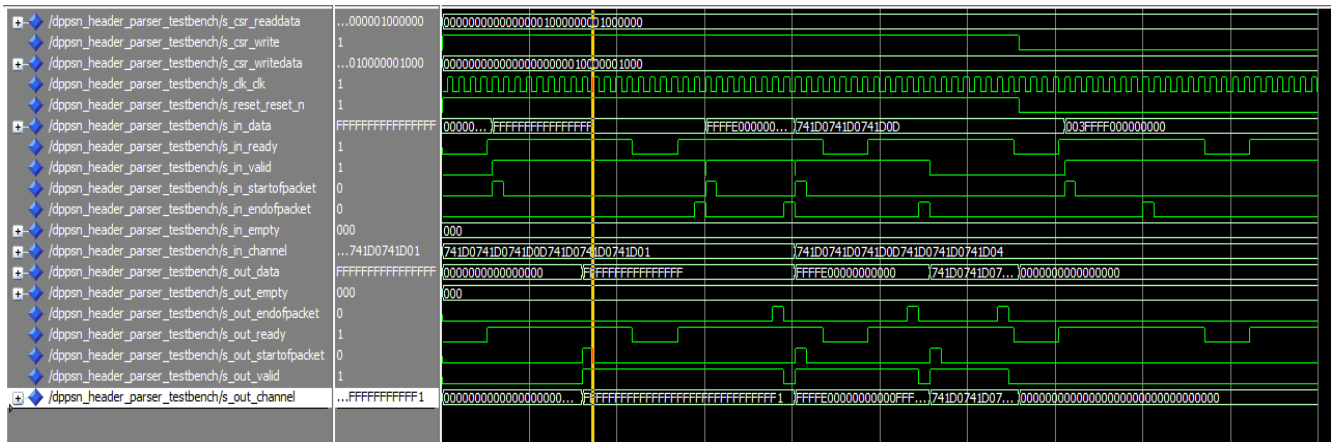
Kako je već rečeno, header parser uvodi kašnjenje od 8 takt intervala, pa zbog toga vidimo da signal **out_data** kasni za 8 takt intervala, plus dodatni takt interval na početku jer slanje nije počelo odmah. Što znači da će nakon 9 takt intervala početi prosljeđivanje paketa D1 na izlaz. Tokom njegovog prosljeđivanja, potrebno je da signal **out_ready** bude konstantno postavljen na $'1'$, što znači da je sljedeći sklop spreman za prijem paketa. Na slici 10 je prikazan slučaj kada se tokom slanja paketa D1, signal **out_valid** spusti na $'0'$ u trajanju od 2 takt intervala, te se u tom periodu obustavlja slanje. Kada se **out_valid** vrati na $'1'$, prenos se nastavlja nesmetano.

Paketi D2 i D3 su prosljeđeni odmah nakon paketa D1, dok je za prenos paketa D4 potrebno čekati 2 takt intervala koliko nije bilo prenosa i dodatna 2 takt intervala tokom kojih je **out_valid** spušten na $'0'$, što je ukupno kašnjenje od 4 takt intervala. Nakon njegovog dizanja na $'1'$, paket D4 se šalje.

Na osnovu svega navedenog, u nastavku su dati testni scenariji u VHDL testbench-u:

1. <u>Prvi testni scenarij</u>

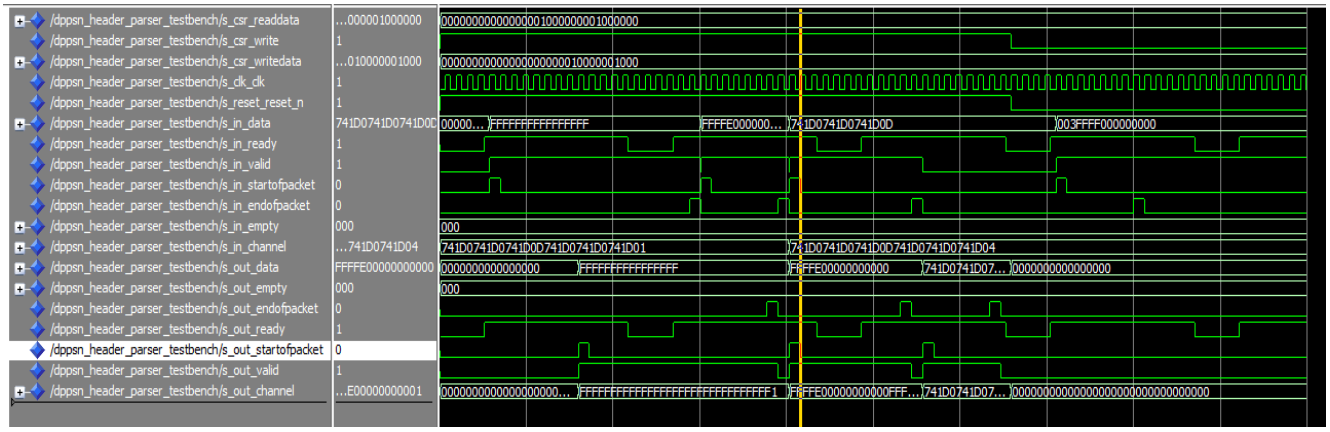
U prvom testnom scenariju šalje se veliki paket koji se sastoji od $15x64$ $8x64$ bita. Očekuje se prenos 13 dijelova paketa od po 8 bajta, dok je signal **out_ready** postavljen na $'1'$. Naredna 4 takt intervala je signal **out_ready** spušten na $'0'$, zbog toga je prenos zaustavljen, te se u naredna 2 takt intervala kada se **out_ready** vrati na $'1'$ prenesu preostalih 16 bajta od poslanog paketa.



Slika 11. Prikaz prvog scenarija u VHDL-u

2. Drugi testni scenarij

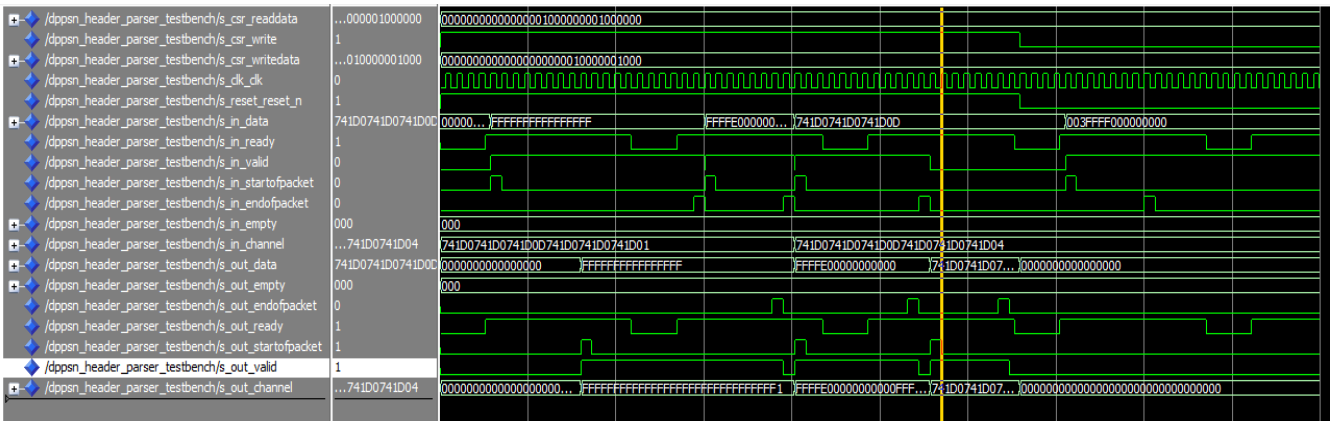
U ovom testnom scenariju šalje se najmanji paket, koji se sastoji od $8x64$ bita. Očekuje se prenos svih $8x64$ bita u 8 takt intervala, jer je signal **out_ready** uvijek postavljen na $'1'$.



Slika 12. Prikaz drugog scenarija u VHDL-u

3. Treći testni scenarij

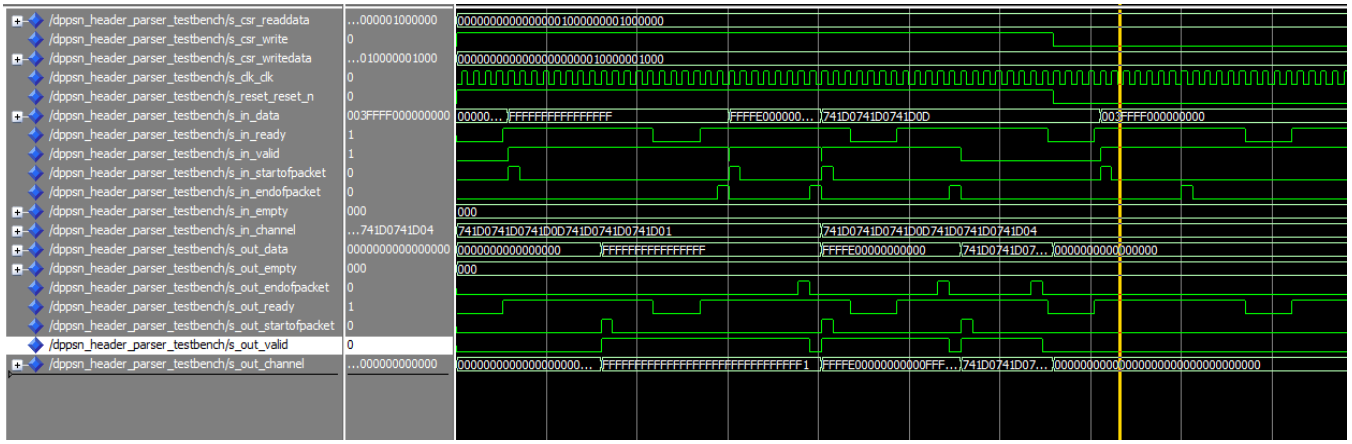
U trećem testnom scenariju šalje se opet najmanji paket, od $8x64$ bita. Prvo se vrši prenos $2x64$ bita, dok je **out_ready** postavljen na $'1'$. Naredna 4 takt intervala je postavljen na $'0'$, u tom operiodu nema slanja, nakon toga se **out_ready** opet diže na $'1'$, te se nastavlja slanje preostalih $6x64$ bita. Tako da se prenos ovog paketa vrši za vrijeme od 12 takt intervala.



Slika 13. Prikaz trećeg scenarija u VHDL-u

4. Četvrti testni scenarij

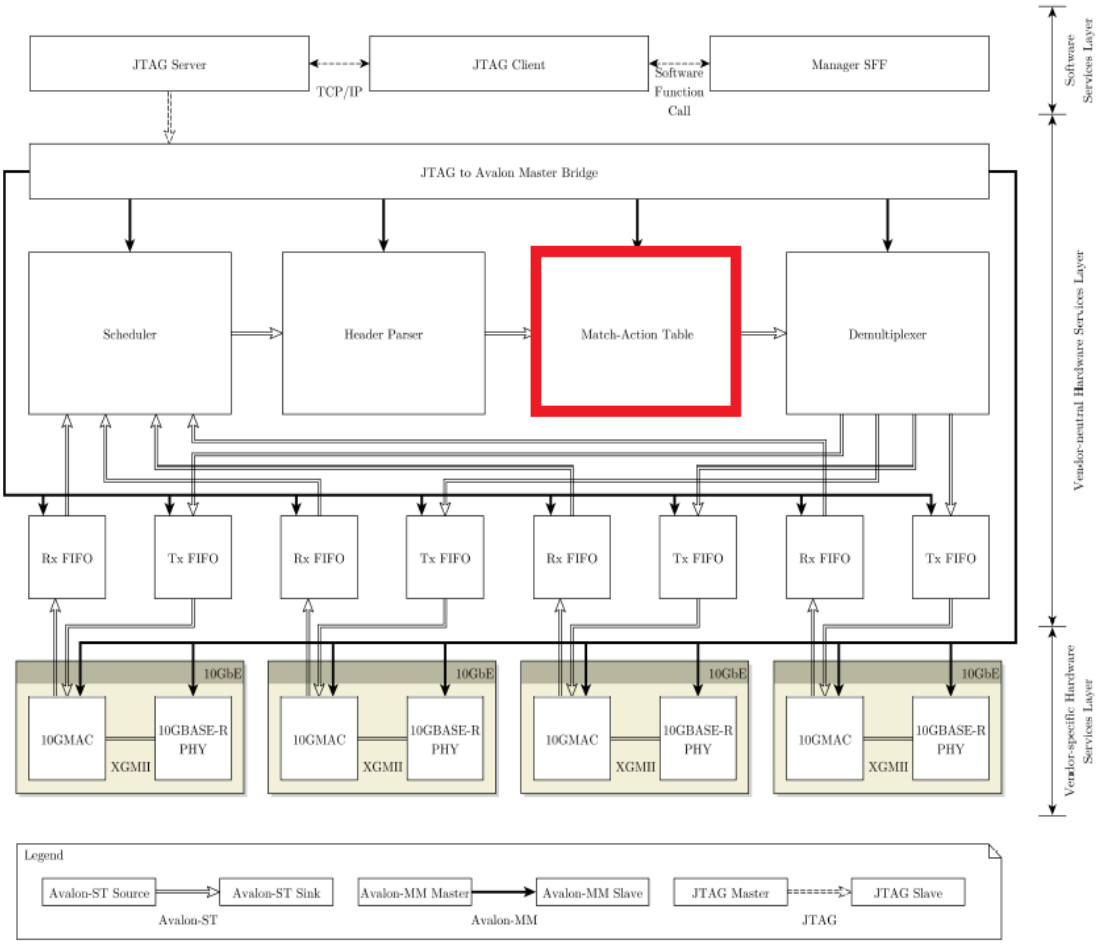
Četvrti testni scenarij je testiranje sistema u odnosu na vrijednost **reset_reset_n** signala. Ukoliko je ovaj signal jednak ′1′, sistem treba da radi u skladu sa prva tri testna scenarija navedena iznad. U ovom slučaju je **reset_reset_n** postavljen na ′0′ i prema tome bi trebalo da svi signali na izlazu budu na ′0′, što je i potvrđeno slikom 14.



Slika 14. Prikaz četvrtog scenarija u VHDL-u

3. MATCH ACTION TABLE

Treća komponenta koju je potrebno testirati u DPPSN arhitekturi je Match-Action Table, a njen položaj u DPPSN arhitekturi je prikazan na slici 15.

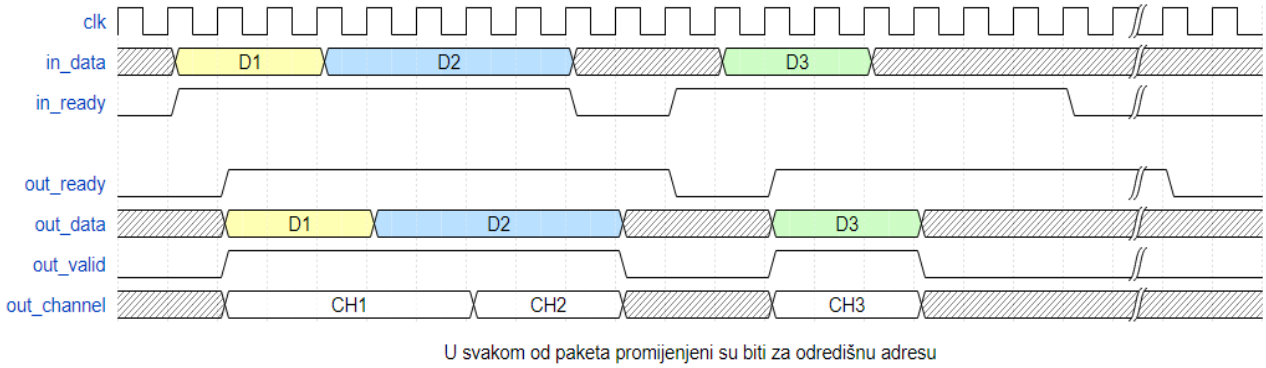


Slika 15. Položaj Match-Action tabele u DPPSN arhitekturi

Glavni zadatak ove komponente je *pretraživanje tabela proslijeđivanja i određivanje na koji izlazni red čekanja će biti proslijeđen paket*. Potrebno je napraviti dvije tabele i to tabelu podataka i tabelu maski čijim se pretraživanjem određuje izlaz iz ove komponente. Prije svega je potrebno popuniti ove dvije tabele, a to se radi pomoću signala **csr_adress** i **csr_writedata**. Da bi se pronašlo podudaranje potrebno je da maskiranje podataka bude jednako maskiranju signala **in_channel** koji ovoj komponenti prosljeđuje Header Parser.

Bitno je napomenuti da se paketi od Header Parsera prema demultiplekseru prenose putem Avalon-ST sučelja i da će biti proslijeđeni tek onda kada je izlaz spreman da primi podatke što se kontroliše putem signala **out_ready.** Analizom koda može se zaključiti da se u ovoj komponenti svi ulazni signali, osim signala **in_channel**, samo prenose na izlaz ove komponente nepromijenjeni, dok se signal **in_channel**, kao što je ranije opisano, koristi za pretraživanje tabele proslijeđivanja i imat će neku vrijednost različitu od svih nula jedino u slučaju ako se pronađe podudaranje.

Na osnovu do sad navedenog, prije samog testiranja kreiran je dijagram koji predstavlja očekivano ponašanje ove komponente koji je prikazan na slici 16. Prikazani au samo oni signali koji su najvažniji za posmatranje ove komponente kao i pretpostavka izlaznih signala na osnovu vrijednosti koje se nalaze na ulazu ove komponente.

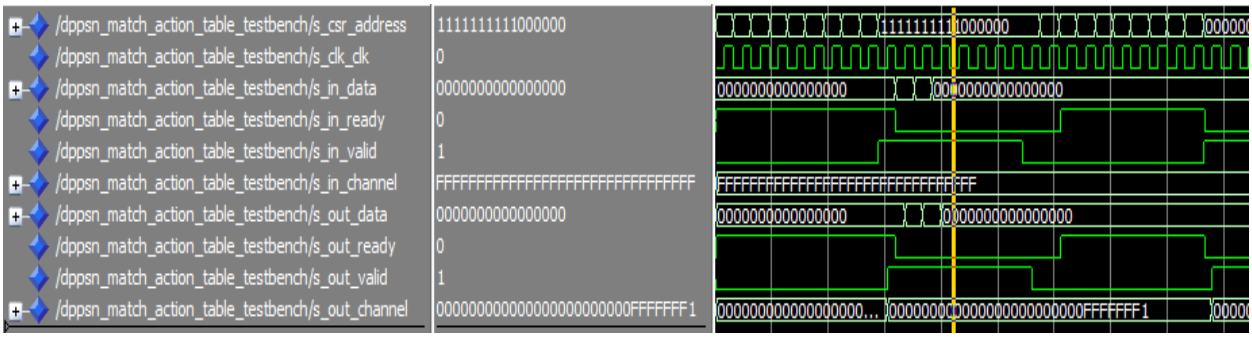


Slika 16. Prikaz očekivanih vrijednosti za Match-Action Table

Također je potrebno napomenuti da na dijagramu nisu prikazani svi signali koji se koriste, a koji su potrebni da bi se realizovala komunikacija putem Avalon-ST sučelja, a jedini razlog tome jeste preglednost dijagram. Također, pije samog crtanja ovih grafika pretpostavljeno da su tabela podataka i tabela maskiranja popunjene i da vrijednost podudaranja zavisi isključivo od **in_channel** signala. U nastavku će biti prikazani testni scenariji za ovu komponentu.

1. <u>Prvi testni scenario</u>

U ovom testnom scenariu je odlučeno da se prvo popune tabela podataka i tabela maskiranja pomoću kontrolnih signala na takav način da će njihovim modulo dva množenjem kao i modulo dva množenjem podataka u tabeli maskiranja sa signalom **in_channel**. Nakon toga potrebno je proslijediti cijeli paket koji se dovodi na ulaz ove komponente i na izlaz se proslijeđuju svi signali sa ulaza, osim **in_channel**, koji zbog prigodno podešenih kontrolnih signala i njega samog daje podudaranje da se paket proslijedi na posljednji red čekanja. Rezultati testiranja su prikazani na slici 17.

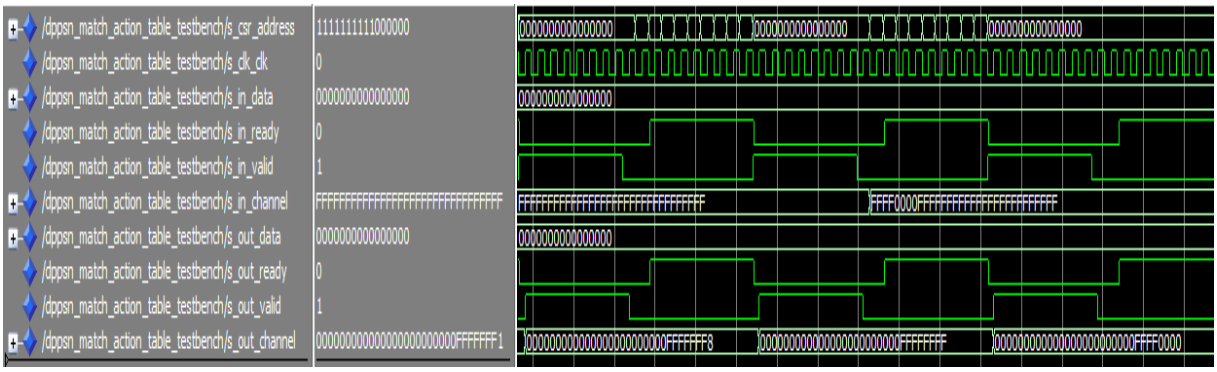


Slika 17. Prvi testni scenario za Match-Action Table

Vidimo da će paket zaista biti proslijeđen na posljednji red čekanja. U prikazu signala **out_channel** koji je prikazan u hexadecimalnom obliku radi lakše preglednosti zadnji bit jednak jedinici što znači da je u pitanju posljednji red čekanja, pa zaključujemo da je ovaj testni scenario dao rezultate u skladu sa očekivanjima.

2. Drugi testni scenario

Drugi testni scenario je identičan prvom samo što smo prigodnim odabirom kontrolnih signala, **csr_adress** i **csr_writedata**, i ulaznog signala **in_channel** predvidjeli da će prvi sljedeći paket biti proslijeđen na prvi red čekanja, paket iza njega na sve redove čekanja i posljednji paket je odabran tako da bude odbačen u demultiplekseru, komponenti koja slijedi nakon Match-Action tabele. Na slici 18 je prikazan rezultat testiranja ovog scenarija.



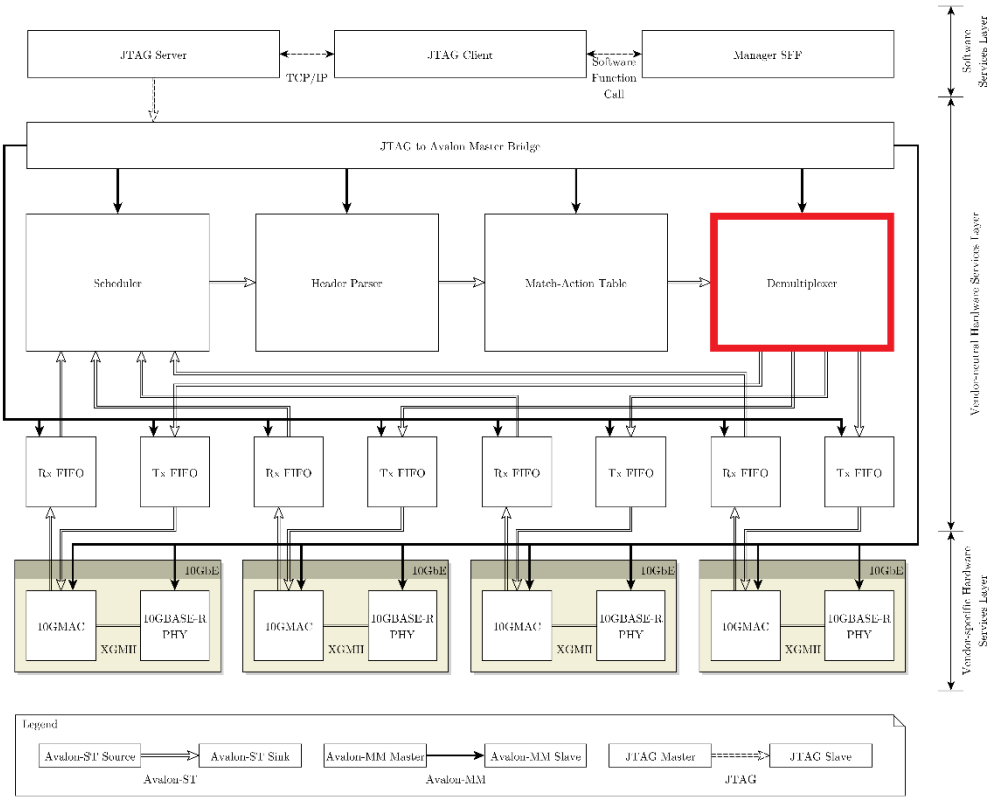
Slika 18. Drugi testni scenario za Match-Action Table

Kao što možemo vidjeti sa grafika dobivenog nakon pisanja testne skripte posmatrajući signal **out_channel**, naročito zadnju cifru njegovog hexadecimalnog zapisa koji je prikazan možemo doći do zaključka da se i ovaj put rezultati testiranja podudaraju sa predpostavljenim vrijednostima. Na osnovu ova dva pozitivno provedena scenarija sa više mogućih vrijednosti podudaranja zaključujemo da ova komponenta radi ispravno i da se paketi proslijeđuju na ispravnu adresu određenu prije svega kontrolnim signalima **csr_adress** i **csr_writedata**.

4. DEMULTIPLEKSER

Posljednja komponenta koju je potrebno obraditi jeste izlazni demultiplekser čiji je položaj unutar preklopnika prikazan na slici 19.

Osnovna uloga izlaznog demultipleksera je *raspoređivanje paketa u odlazne redove čekanja* i to u skladu sa uputama iz signalizacijskog kanala na ulaznom Avalon-ST sučelju. Odlazni redovi čekanja su implementirani pomoću Avalon-ST Single Clock FIFO jezgre. Putem Avalon-ST sučelja se prenose paketi od izvora (*source*) do odredišta (*sink*). Demultiplekser jedan ulaz i ima četiri izlaza za četiri reda čekanja (izlazna – *Tx FIFO* na slici 19). Raspoređivanje je moguće obaviti istovremeno u više odlaznih redova čekanja (*multicast*) ili zadržati paket u privremenoj memoriji.



Slika 19. Položaj demultipleksera u DPPSN arhitekturi

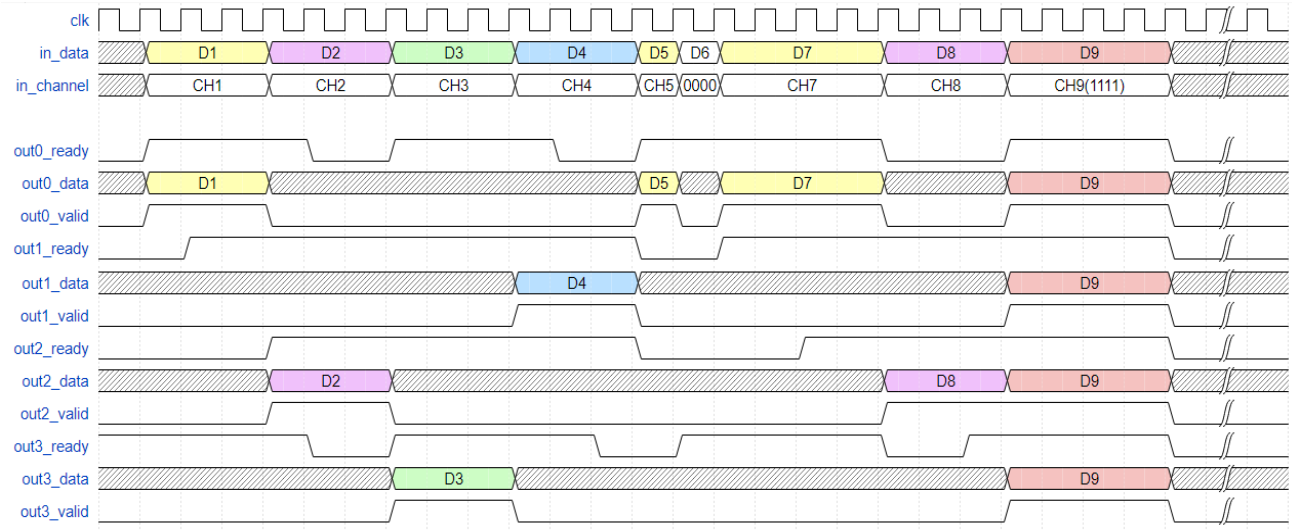
Unutar arhitekture demultipleksera definisan je proces u svrhu odabira odgovarajućeg reda čekanja, dok u opisu entiteta su oni predstavljeni oznakama **out0, out1, out2** i **out3.** Kao što je već navedeno, korišteno je Avalon-ST sučelje u svrhu implementacije redova čekanja, a kako se isto sučelje koristi prilikom 'komunikacije' izvora i odredišta, za svaki od izlaznih redova čekanja je bilo potrebno definisati sljedeće signale: **data, valid, startofpacket, endofpacket, empty** i **ready**, što u konačnici daje 24 signala za izlazne redove čekanja.

Dakle, zadatak izlaznog demultipleksera je da rasporedi pakete u odgovarajuće redove čekanja. Navedeno raspoređivanje se vrši na osnovu posljednja četiri bita signala **in_channel.** Naime, jedan od signala koji se dovodi na demultiplekser kao ulazni signal jeste signalizacijski kanal, **in_channel**, tj. signal koji sadrži korisničku signalizaciju koja prati podatak u svakom ciklusu transfera paketa. Posljednja četiri bita ovog signala predstavljaju određene izlazne redove čekanja na koje je potrebno prosljeđivati pakete, pri čemu vrijedi sljedeća notacija:

- **Odlazni red čekanja out0** – "1000"
- **Odlazni red čekanja out1** – "0100"
- **Odlazni red čekanja out2** – "0010"
- **Odlazni red čekanja out3** – "0001"

U skladu sa prethodno navedenim, unutar arhitekture je definisan proces odabira odgovarajućeg reda čekanja na osnovu provjere vrijednosti jednog od posljednja četiri bita signala **in_channel.** Na primjer, za prvi red čekanja (**out0**) provjerava se da li četvrti bit signala **in_channel** ima vrijednost ′1′, (**in_channel(3)**). Ukoliko ima, svim portovima koji odgovaraju navedenom kanalu, a definisani su u entitetu, se dodjeljuju vrijednosti odgovarajućih signala. Ukoliko ne, svakom izlaznom portu se dodjeljuju nule. Navedeno je urađeno za svaki izlazni red čekanja, tj. vršena je provjera vrijednosti bita na odgovarajućem mjestu unutar signala **in_channel**.

Prije samog pisanja testnog okruženja za demultiplekser, napravljen je dijagram, prikazan na slici 20, koji demonstira princip rada demultipleksera i daje uvid u očekivane rezultate prilikom testiranja ove komponente.



Slika 20. Primjer načina rada demultipleksera

Paketi smješteni u **in_data** signal dolaze iz komponente *match-action table*. Kao što je prethodno rečeno, svaki paket prati korisnička signalizacija sadržana u signalu **in_channel** na osnovu koje se određuje odlazni red čekanja na koji treba proslijediti odgovarajući paket. Dakle, očekivano je da će podaci, zajedno sa pratećim signalima (**in_startofpacket**, **in_endofpacket**, **in_valid**, **in_empty**) biti proslijeđeni na izlaze koji su određeni sa posljednja četiri bita signala **in_channel**, u skladu sa prethodno navedenom notacijom.

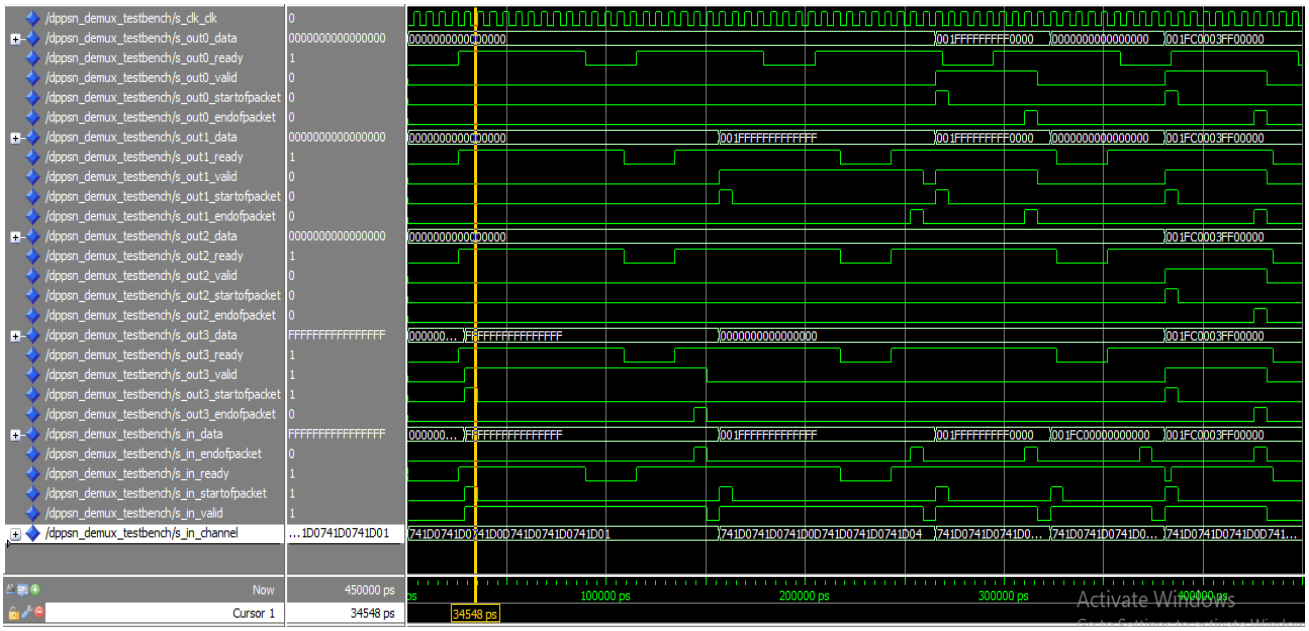
Na dijagramu su prikazani karakteristični scenariji i očekivane akcije kada postoje paketi koji trebaju biti proslijeđeni samo na jedan red čekanja/izlaz (npr. paket D1), kada se paket ne treba proslijediti niti na jedan izlaz (paket D6) kao i na svaki izlaz (paket D9). Na dijagramu nisu prikazani specifični slučajevi kada potreban izlaz nije spreman da primi paket (***out_ready*** = *'0'*) ili kada paket koji se prenosi nije validan (***in_valid*** = *'0'*), što će biti razmatrano u nastavku.

Prilikom testiranja komponente, u VHDL *testbench*-u su kreirani odvojeni stimulacijski procesi za izlazne redove čekanja koji simuliraju dostupnost pojedinih redova čekanja. Ovim procesima je simulirana tehnika potiskivanja unazad (*backpressure*) pomoću koje odredište vrši kontrolu toka. Također, postoji i odvojen stimulacijski proces za ulazni signal koji, između ostalog, daje i **in_ready** signal. U skladu sa navedenim, pojavljuju se mnoge pauze prilikom slanja paketa zbog nedostupnosti izlaza i/ili ulaza.
U nastavku su navedeni karakteristični scenariji za koje je testirana komponenta kao i vremenski prikaz signala za pojedinačne scenarije.

1. Prvi testni scenarij

U okviru prvog scenarija provjera se da li se paket ispravno prosljeđuje na potrebni izlazni red čekanja. Odabrani paket od 15 blokova po 64 bita ($15x64$ bita) treba biti proslijeđen na četvrti odlazni red čekanja – **out3** (posljednja četiri bita **in_channel** signala su "0001").

Zbog činjenice da za prenos svakog paketa potreban jedan takt interval, za prenos prvog seta paketa je potrebno 15 takt intervala. Pored toga, potrebno je sačekati sa prenosom 4 takt intervala za koje je signal **out3_ready** jednak $'0'$. Uzima se u obzir i 1 takt interval za koje je signal **in_valid** jednak $'1'$. Dakle, vrijeme potrebno za prenos prvog seta paketa jeste $19 + 1$ takt intervala.



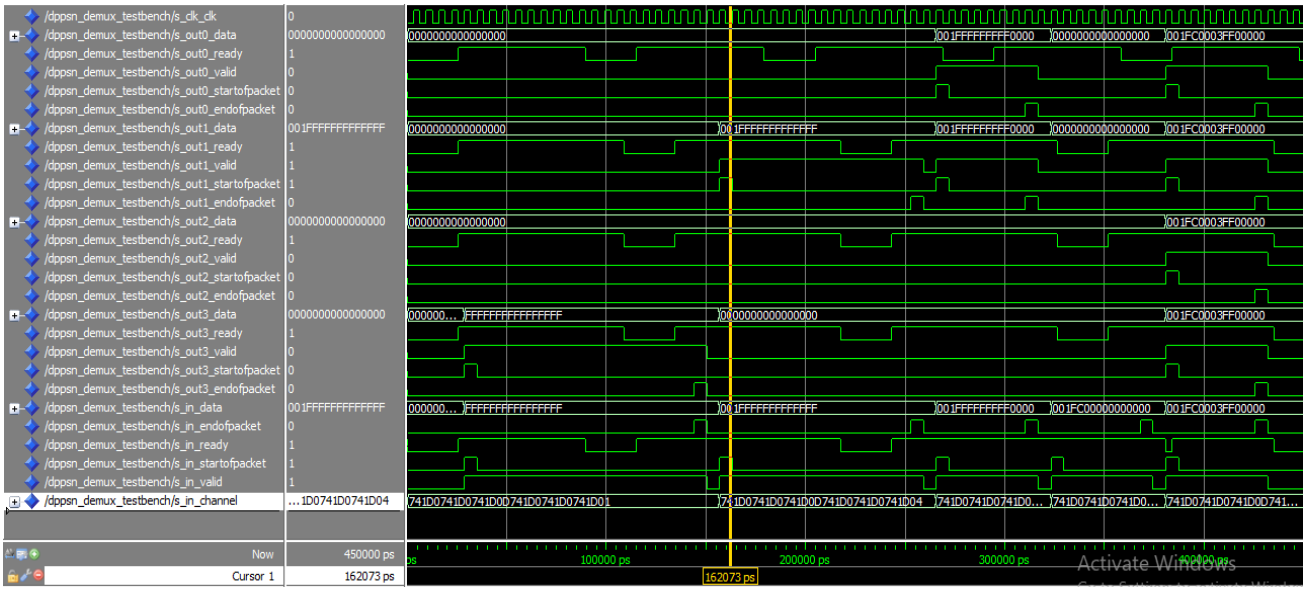
Slika 21. Paket se prosljeđuje na četvrti red čekanja

Na slici 21 se da primjetiti da je posljednja vrijednost signala **in_channel** jednaka $'1'$, što u binarnom zapisu daje vrijednost "0001" te vidimo da je paket zaista proslijeđen na četvrti odlazni red čekanja (**out3**), dok su ostali izlazi prazni.

2. <u>Drugi testni scenarij</u>

Drugom testnom scenariju odgovara provjera da li se paket od 12 dijelova od po 64 bita ($12x64$ bita) prosljeđuje na drugi odlazni red čekanja – **out1** (posljednja četiri bita **in_channel** signala su "0100").

Za prenos ovog paketa je potrebno 12 takt intervala, kao i dodatna 4 takt inetrvala zbog signala **out1_ready** koji je jednak $'0'$ (drugi odlazni red čekanja nije spreman za prijem paketa) i trajanje 1 takt intervala zbog nevalidnog ulaznog signala – $\boldsymbol{in\_valid} = '0'$. Dakle, vrijeme potrebno za prenos drugog paketa jeste $16 + 1$ takt intervala.



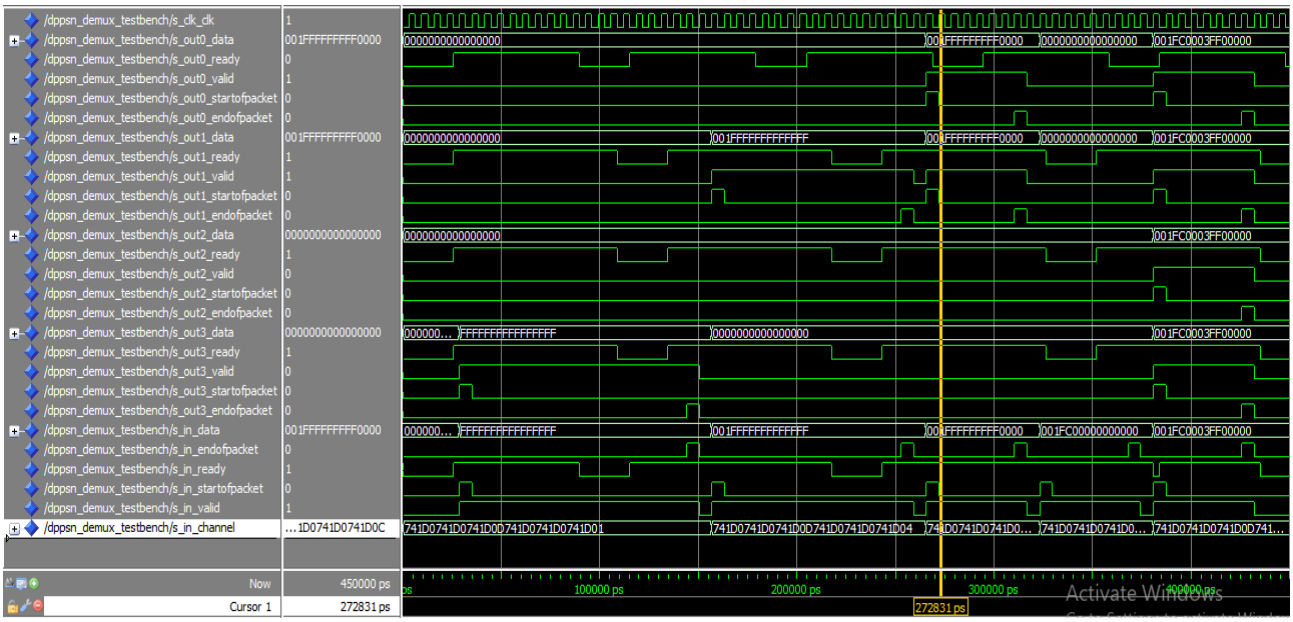
Slika 22. Paket se prosljeđuje na drugi red čekanja

Na slici 22 se da primjetiti da je posljednja vrijednost signala **in_channel** jednaka $'4'$, što u binarnom zapisu daje vrijednost "0100" te vidimo da je paket zaista proslijeđen na drugi odlazni red čekanja (**out1**), dok su ostali izlazi prazni.

3. Treći testni scenarij

Treći scenario provjerava da li se paket od 8 blokova od po 64 bita($8x64$ bita) ispravno prosljeđuje na više od jednog izlaza, u ovom slučaju na dva i to na prvi – **out0** i drugi – **out1** izlaz (posljednja četiri bita **in_channel** signala u ovom slučaju su "1100").

Za prenos ovog paketa na dva različita izlaza je potrebno 8 takt intervala kao i 1 takt interval za koje je uzeto da ulazni signal nije validan; pauza između različitih paketa ($\boldsymbol{in\_valid} = '0'$). U ovom slučaju, zbog činjenice da tokom trajanja prenosa paketa potrebni odlazni redovi čekanja su spremni za prelazak u naredni ciklus njegovog transfera, paket se isključivo prenosi za vrijeme trajanja $8 + 1$ takt intervala.



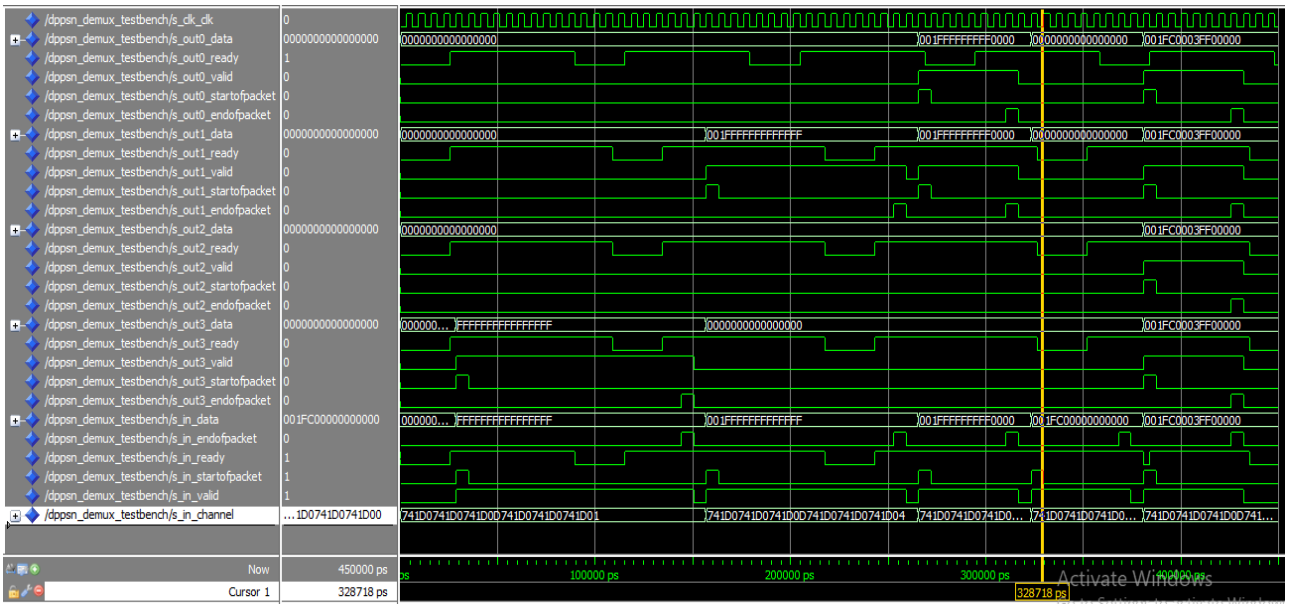
Slika 23. Paket se prosljeđuje na prvi i drugi red čekanja

Na slici 23 se da primjetiti da je posljednja vrijednost signala **in_channel** jednaka vrijednosti $'C'$, što u binarnom zapisu daje vrijednost "1100". Vidimo da je paket zaista proslijeđen na dva različita odlazna reda čekanja i to na prvi i drugi izlaz (**out0** i **out1**), dok su preostala dva izlaza prazna.

4. Četvrti testni scenarij

U narednom, četvrtom scenariju se provjerava šta će se desiti sa paketom ukoliko se on ne treba proslijediti niti na jedan odlazni red čekanja (posljednja četiri bita **in_channel** signala u ovom slučaju su "0000").

U tom slučaju se na svaki izlazni red čekanja prosljeđuju nule, a za prenos narednog paketa treba sačekati onoliko takt intervala koliko bi trajao prenos inicijalnog paketa (od 8 blokova ($8x64$ bita)) tj. za prenos u ovom slučaju je potrebno $8 + 1$ ($\boldsymbol{in\_valid} = '0'$) takt intervala, gdje 1 takt interval predstavlja pauzu između slanja dva paketa. U ovom slučaju nema pauze prilikom slanja zbog **in_ready** signala.



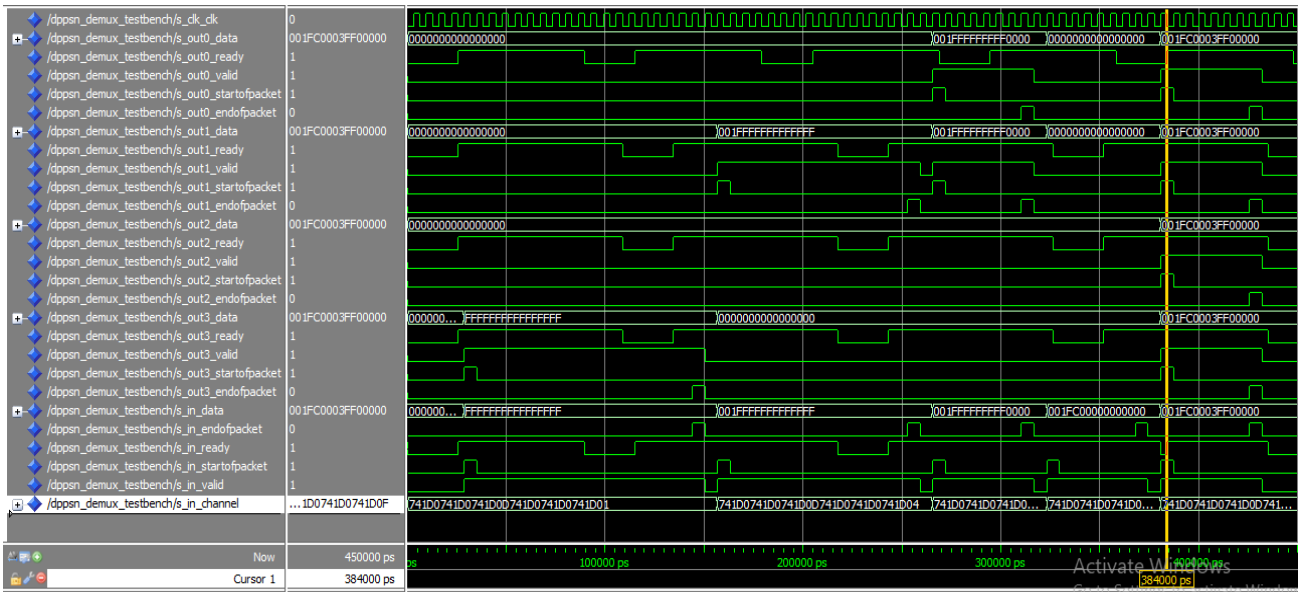
Slika 24. Paket se ne prosljeđuje ni na jedan izlaz

Na slici 24 se da primjetiti da je posljednja vrijednost signala **in_channel** jednaka ′0′, što u binarnom zapisu daje vrijednost "0000". Uočava se da paket zaista nije proslijeđen ni na jedan odlazni red čekanja; na sve izlaze su proslijeđene nule, što je u skladu sa očekivanim.

5. Peti testni scenarij

U posljednjem, petom scenariju provjerava se slučaj kada se paket treba proslijediti na sve redove čekanja. U tom slučaju posljednja četiri bita signala **in_channel** imaju vrijednost "1111" (na ovaj način su označeni svi izlazi).

Paket sačinjen od 8 blokova od po 8 bajta ($8x64$ bita) se zaista prosljeđuje na svaki od izlaznih redova čekanja, a vrijeme potrebno za njegov prenos zavisi od spremnosti svakog od izlaza da primi paket (**out0_ready**, **out1_ready**, **out2_ready**, **out3_ready**). U ovom slučaju, za prenos paketa je potrebno 8 takt intervala, nakon čega se signal **in_valid** obara na vrijednost ′0′ jer, nakon ovog paketa, ne postoji više paketa za slanje. Vrijeme potrebno za njegov prenosi iznosi 8 takt intervala.



Slika 25. Paket se prosljeđuje ni svaki odlazni red čekanja

Na slici 25 se da vidi da je posljednja vrijednost signala **in_channel** jednaka vrijednosti $'F'$, što u binarnom zapisu daje vrijednost "1111" te se uočava da je paket zaista proslijeđen na svaki izlaz, a trajanje njegovog prenosa je u skladu sa pretpostavljenim.

Literatura:

1. Duboko programabilno paketsko čvorište (DPPSN)

2. https://github.com/eniokaljic/DPPSN

3. Avalon Interface Specification

4. 10-Gbps Ethernet MAC MegaCore Function, User Guide

5. Predavanje iz predmeta Arhitekture paketskih čvorišta, 2019/2020 (Predavanje 5)