UNIVERZITET U BEOGRADU
ELEKTROTEHNIČKI FAKULTET

HARDVERSKA IMPLEMENTACIJA FRAGMENTACIJE IP PAKETA

–Master rad–

Mentor:
doc. dr Zoran Čiča, docent

Kandidat:
Marko Marić 3046/2014

Beograd, Septembar 2016.

SADRŽAJ

1. UVOD

Današnji svet je teško zamisliti bez Interneta. Uveliko je prisutan u svim oblastima delovanja ljudske zajednice: poslovna (elektronska pošta, konferencijske veze,…), razonoda (facebook, viber, igrice,…), kućna (pametna kuća), i dr.

Servisi, koje nudi Internet, su sve raznovrsniji i time privlače sve više korisnika. Povećanjem broja korisnika povećava se i opterećenje mreže i borba za održanje kvaliteta servisa koje korisnik traži. Na putu do svog odredišta IP (*Internet Protocol*) paket prolazi kroz mrežu koja može imati različite vrste linkova. Svaki link zbog svojih fizičkih svojstava određuje i dozvoljenu veličinu paketa koja se može poslati kroz taj link. Kao posledica toga pojavljuje se proces fragmentacije poslatog paketa, odnosno podela na dva ili više delova u odnosu na prvobitno poslatog.

Jedan od važnijih servisa sa poslovnog aspekta su, svakako, VPN mreže. VPN mreže se tipično zasnivaju na upotrebi tunela kroz javni domen (tipično Internet), pri čemu se korisnički paketi enkapsuliraju u paket koji prolazi kroz tunel, a sve u cilju ostvarivanja sigurnosti komunikacije u okviru VPN mreže. Usled dodavanja dodatnog zaglavlja može se desiti da paket koji prolazi kroz tunel bude duži od dozvoljene vrednosti (tzv. MTU vrednost - *Maximum Transsmision Unit*) pa je potrebno izvršiti fragmentaciju paketa na dva dela u ovakvom slučaju.

Ovaj rad predlaže jedno rešenje za realizaciju podele paketa na dva dela i njihovu rekonstrukciju na kraju tunela. Programski jezik korišćen u ovom radu je VHDL (*VHSIC Hardware Description Language*). Korišćeno je razvojno okruženje ISE proizvođača Xilinx. Glavni projekat, kao i testbenč za verifikaciju korektnog rada, priložen je u elektronskom formatu.

U poglavlju 2. biće objašnjena osnovna ideja postupka fragmentacije, njene prednosti, mane, standard. Potom, u narednom poglavlju biće dat kratak prikaz rezultata teze Aleksandra Dželatovića, jer ova teza vrši proširenje navedene teze, dodavajući podršku za fragmentaciju. Biće navedene i promene koje su urađene na tom projektu kako bi se adaptirali ovoj temi. U četvrtom poglavlju, biće predstavljena simulacija koja prikazuje validnost koda realizovanog u okviru ove teze. Peto poglavlje rezimira rad i navodi konačan zaključak.

2. FRAGMENTACIJA

2.1. Maximum Transmission Unit (MTU)

Fragmentacija IP paketa predstavlja proces podele originalno poslatog paketa na više paketa manje dužine. Razlog zašto se radi fragmentacija je taj što svaki medijum preko kojeg se šalje paket, zbog svojih fizičkih svojstava, ima svoj specifičan broj koji ukazuje na maksimalno dozvoljenu veličinu paketa koja se može poslati. Taj broj se naziva *Maximum Transsmision Unit* (MTU). Vrednost ovog parametra varira u zavisnosti koji protokol se koristi na sloju linka. U tabeli 2.1 su navedene vrednosti za protokole koji se tipično koriste danas:

Tabela 2.1. Tabela vrednosti MTU za različite protokole

PROTOKOL	MTU (U BAJTOVIMA)
Ethernet v2	1500
Ethernet with LLC and SNAP, PPPoE	1492
Ethernet Jumbo Frames	1501 - 9198
PPPoE over Ethernet v2	1492
PPPoE over Ethernet Jumbo Frames	1493 - 9190
WLAN (802.11)	7981
Token Ring (802.5)	4464
FDDI	4352

Vrednost koja treba da se postavi za MTU može se odrediti na dva načina:

1) Određivanjem MTU na svakom linku ponaosob;

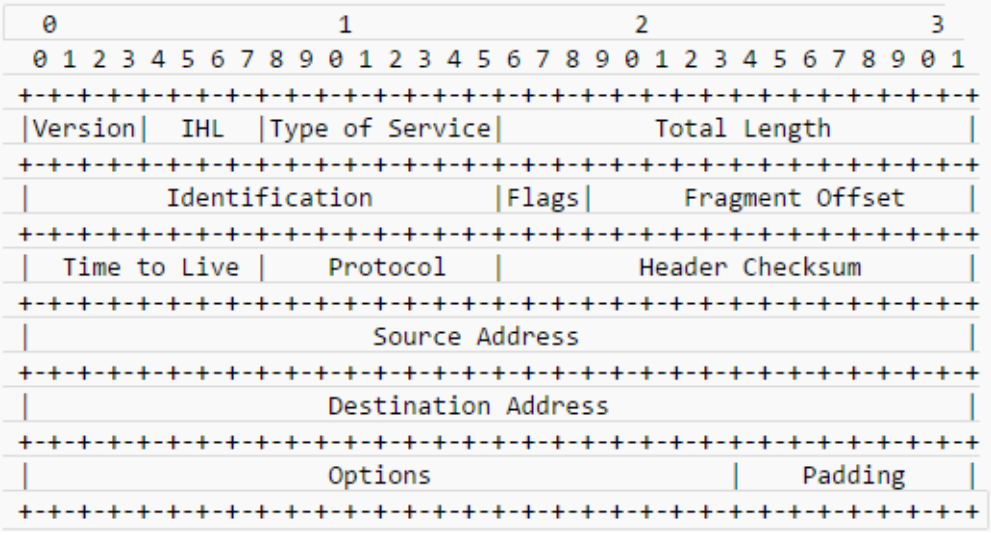
2) Path MTU Discovery.

Prvi način je veoma jednostavan i on na svakom fizičkom linku između dva uređaja određuje vrednost MTU za taj link. Ovaj način je bolji, jer se može brzo odrediti MTU za link. Mada, ima i dosta mana za ovu metodu. Kako se na svakom linku određuje vrednost MTU, može se desiti da se vrednost MTU smanji tokom daljeg slanja kroz mrežu. To povlači dalju fragmentaciju, već fragmentiranog paketa, odnosno povećava se upotreba zaglavlja koji smanjuje količinu korisnog saobraćaja.

Drugi način predstavlja unapred određivanje vrednosti najmanjeg MTU koji se može poslati kroz mrežu. Princip se zasniva na slanju paketa i odziva ICMP porukom. Predajni deo šalje paket određene dužine koji se ne može fragmentirati (pogledati sledeće potpoglavlje) i posmatra da li će se pojaviti ICMP poruka koja obaveštava da je paket isuviše velik. Ukoliko se pojavi takva poruka, smanjuje se testni paket i tako sve dok ne prođe fragment bez problema. Ovaj način omogućava mnogo lakše određivanje veličine fragmenata i time smanje potrebu za procesiranje novih fragmenata, kao i ekonomičnije korišćenje zaglavlja. Međutim, činjenica da je potrebno neko vreme da se odredi najmanji MTU sa kojim fragment može proći kroz mrežu, dosta usporava slanje paketa, pogotovo ako putanja ima mnogo hopova. Takođe, praktično je teško odrediti ovom metodom najmanji MTU zato što većina uređaja ima u sebi ugrađeno dropovanje ICMP saobraćaja, uglavnom zbog sprečavanja DoS (*Denial of Service*) napada.

2.2. IP zaglavlje

Jedna od razlika između IPv4 i IPv6 protokola je u tome što kod IPv6 protokola ne postoji mogućnost fragmentacije. Kod IPv6 protokola se paket odbacuje ukoliko je veći od MTU. Zato kad govorimo o fragmentaciji, podrazumeva se da je u pitanju IPv4 protokol. Na slici 2.2.1 je prikazan izgled IPv4 zaglavlja.



Slika 2.2.1 IPv4 Zaglavlje

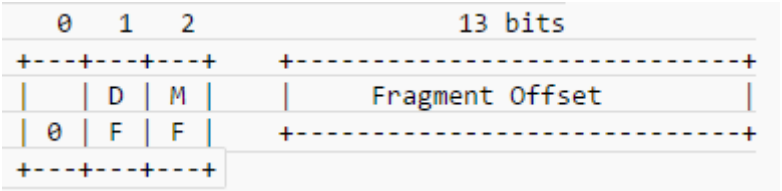
Polja od značaja za proces fragmentacije su Identification, Flags i Fragment Offset.

Polje Flags pokazuje da li je pristigli paket fragment nekog većeg paketa. Ovo polje sadrži 3 bita i njihova značenja su sledeća:

Bit 0: Rezervisan, mora biti 0;

Bit 1: 0= dozvoljeno fragmentiranje, 1= zabranjeno fragmentiranje;

Bit 2: 0= poslednji fragment, 1= ima još fragmenata.



Slika 2.2.2. Polja za fragmentaciju

Fragment Offset predstavlja poziciju posmatranog fragmenta unutar originalno poslatog paketa, mereno u jedinicama od po 8 bajtova. Teorijski najveći fragment može biti 8192 (2^{13}) osmobajtnih jedinica, odnosno 8192*8= 65536 bajtova. Međutim, kako polje Total Length meri veličinu paketa u bajtovima i predstavlja zbir zaglavlja i korisnog dela paketa, maksimalni mogući fragment je nešto manji. Maksimalni mogući broj bajtova unutar Total Length polja je 2^{16}=65536 bajtova. Kako zaglavlje može minimalno biti 20 bajtova, to znači da maksimalna veličina korisnog saobraćaja može biti 65516 bajtova, odnosno 8189 osmobajtnih jedinica.

Kako je IP nekonekcioni (*connectionless*) protokol, to znači da dva fragmenta ne moraju da idu jedan za drugim, odnosno mogu biti izmešani sa fragmentima od drugog paketa. Krajnji uređaj može da ih razlikuje na osnovu polja Identification.

Predajni uređaj setuje polje Identification na jedinstvenu vrednost za svaki paket kreiranu na osnovu izvorišne i odredišne adrese i protokola koji se koristi za slanje paketa. Na ovaj način, krajnji uređaj može lako identifikovati i lakše baferovati odgovarajuće fragmente dok ne stigne i poslednji fragment, kada nastupa sastavljenje svih fragmenata u prvobitno poslati paket.

2.3. Prednosti i mane fragmentacije

U daljem tekstu su nabrojane neke od najbitnijih prednosti i mana uvođenja fragmentacije.

Prednosti:

1) Smanjuje se zagušenja u saobraćaju – definisanjem veličine paketa se pokušava smanjiti zagušenje linka;
2) Sprečavanje monopolisanja linka – ukoliko bi se slao ceo paket može se desiti da mu je potrebno više vremena i time onemogući ostatku saobraćaja da normalno funkcioniše.

Mane:

1) Osetljivost na greške u prenosu – ukoliko se desi greška u prenosu na bilo kom od fragmenata, potrebno je poslati sve fragmente ispočetka;
2) Dodatno opterećenje rutera koji rekonstruiše paket - ruter ima dodatni posao prilikom obrade fragmenata u odnosu na obradu nefragmentiranih paketa (čuvanje fragmenata, detektovanje kompletiranja paketa i potom rekonstrukcija originalnog paketa)
3) Više vremena neophodno za slanje – kako je neophodno imati sve delove paketa da bi ga sastavili na kraju transporta kroz mrežu, što je paket podeljen na više delova to mu više vremena treba da se prenese.

2.4. Fragmentacija i opis ideje rada

Ideja ovog rada je fragmentacija paketa koji se prenosi kroz tunel, kao i uspešno spajanje fragmenata u originalni paket na prijemnom kraju tunela. Rad predstavlja unapređenje projekta iz master rada Aleksandra Dželatovića u kome su paketi šifrovani AES enkripcijom i poslati kroz tunel kroz mrežu. U ovom radu nisu korišćena polja unutar zaglavlja iz razloga što je moguće odbacivanje fragmenata unutar mreže kako preventiva od DoS napada. Umesto toga, koristiće se neiskorišćeni deo zaglavlja. Zbog kompleksnosti opšteg slučaja, uvedene su sledeće pretpostavke:

1) Medijum kroz koji se prenose fragmenti koristi Ethernet kao protokol i time prihvata se vrednost MTU=1500b, odnosno njena najbliža vrednost (detaljnije u sledećem poglavlju);
2) Paket će biti podeljen maksimalno na dva dela;
3) Korisni deo paketa je celobrojni umnožak od 128 bita;
4) Koristi se samo jedan tunel.

Sve navedene pretpostavke su i realne. U praksi, hostovi uglavnom šalju pakete poštujući MTU ograničenje eternet standarda, a kao posledica toga u našem slučaju to može izazvati minimalno prelivanje novoga paketa (koji se šalje kroz tunel) preko MTU granice usled dodavanja njegovog zaglavlja na korisnički paket u procesu enkapsulacije, što znači da maksimalno može biti ukupno dva fragmenta. Realizovano rešenje se koristi samo za jedan tunel, tj. za svaki tunel se mora kreirati poseban modul i ovaj nedostatak može biti predmet budućih istraživanja.

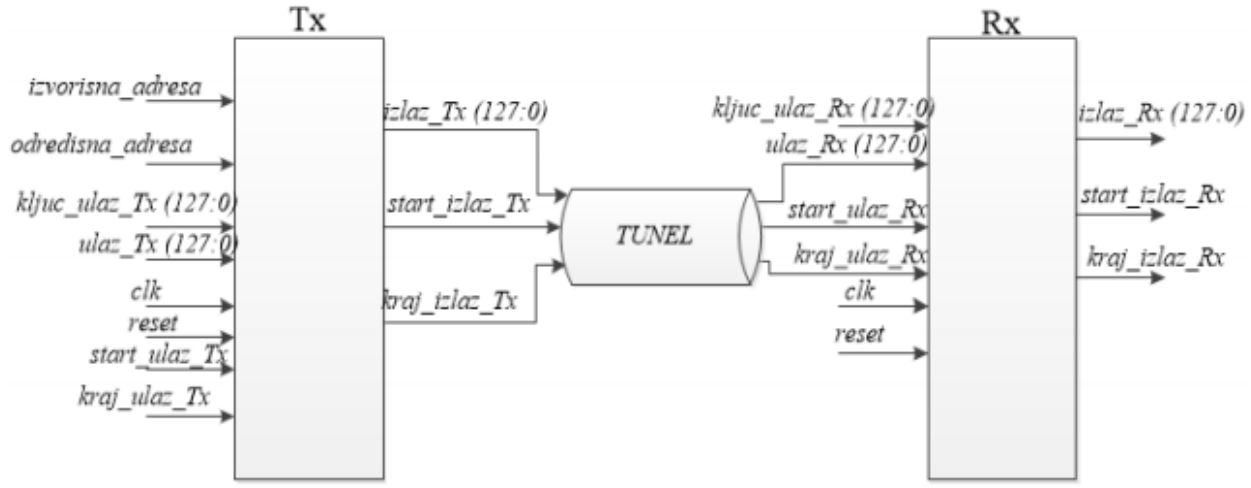
3. REALIZACIJA FRAGMENTACIJE

U ovom poglavlju biće objašnjeno rešenje koje je primenjeno za hardversku realizaciju fragmentacije IP paketa korišćenjem VHDL programskog jezika. Kao osnova ovog rada korišćen je master rad „Implementacija sigurnog IP tunela zasnovanog na AES enkripciji" Aleksandra Dželatovića. Njegov rad je modifikovan samo u delu koji se odnosi na dodavanje i skidanje IP zaglavlja. Deo koji se odnosi na AES enkripciju nije modifikovan. Takođe, ulazni i izlazni parametri unutar „crne kutije" predajnog i prijemnog dela nisu promenjeni. Ceo rad je zasnovan na dodavanju dodatnih procesa unutar arhitekture. Izlazni parametri iz predajnog dela, odnosnu ulazni parametri za prijemni deo nisu promenjeni. U narednom potpoglavlju biće objašnjen njegov rad i rešenje. Drugo potpoglavlje pokriva rešenje ovog rada i modifikacije koje su napravljene.

3.1. Kratak rezime rada „Implementacija sigurnog IP tunela zasnovanog na AES enkripciji"

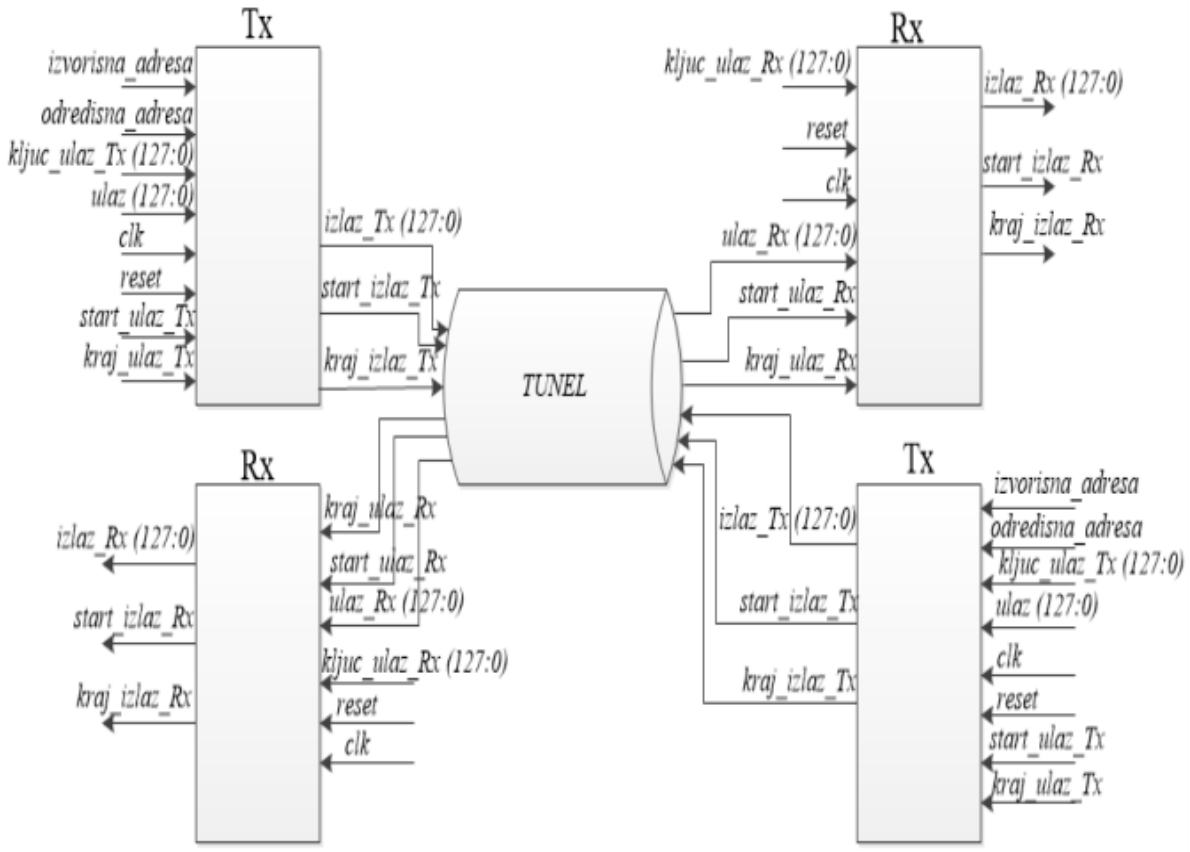
Ideja rada je kreiranje sigurnog IP tunela za slanje paketa. Paketi se, pre nego što su poslati, šifruju AES enkripcijom i time postižu sigurnost prilikom slanja kroz mrežu. Pre slanja, paket je enkapsuliran dodavanjem odgovarajućeg IP zaglavlja na paket. Na prijemnom delu, paketi se izvlače iz tuneovanog paketa, potom dešifruju i na kraju šalju na izlaz iz modula.

Kao ulazni portovi koriste se: *izvorišna_adresa*, *odredišna_adresa*, *kljuc_ulaz_Tx*, *kljuc_ulaz_Rx*, *ulaz_Tx*, *start_ulaz_Tx*, *kraj_ulaz_Tx*, *clk* i *reset*. Ulazi *izvorišna_adresa* i *odredišna_adresa*, kao što im imena kažu, predstavljaju IPv4 adrese izvorišta i odredišta za slanje paketa (adrese krajnjih tačaka tunela). Ove parametre upisujemo unutar zaglavlja paketa. Ulazi *kljuc_ulaz_Tx* i *kljuc_ulaz_Rx* koriste se za AES enkripciju u predajnom i prijemnom delu, respektivno. Ulaz *ulaz_Tx* predstavlja paket koji treba poslati kroz IP tunel i njegov početak i kraj su obeleženi adekvatnom signalizacijom na ulazima *start_ulaz_Tx* i *kraj_ulaz_Tx*. Iz predajnog dela šalju se tri signala ka modulu koji predstavlja drugi sloj OSI modela. U simulacionom modelu sa slike 3.1.1 za potrebe testiranja je preskočen drugi sloj OSI modela i direktno su spojene tačke tunela. Tri para signala nazvani su: *izlaz_Tx* i *ulaz_Rx*, *start_izlaz_Tx* i *start_ulaz_Rx* i *kraj_izlaz_Tx* i *kraj_ulaz_Rx*, gde prvi signal je izlaz iz predajnog dela, a drugi signal ulaz u prijemni deo. Na slici 3.1.1. je prikazana šema opisane arhitekture.



Slika 3.1.1. Rešenje sigurnog IP tunela [1]

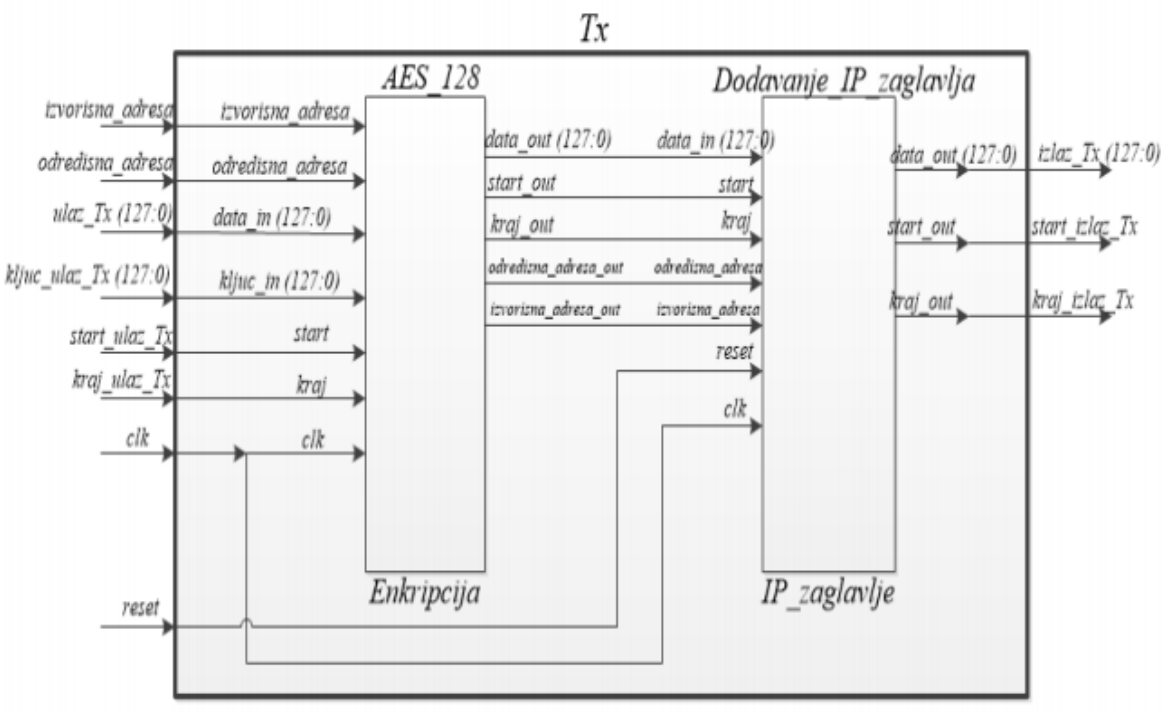
Na slici 3.1.1 je predstavljen samo jedan smer komunikacije. Krajnji rezultat rada je neometan rad u oba smera, odnosno dvosmerna komunikacija, kao što je predstavljeno na slici 3.1.2.



Slika 3.1.2. Logički izgled dvosmerne komunikacije [1]

3.1.1. Predajni deo

Na slici 3.1.1.1 je prikazan izgled unutar „crne kutije" predajnika:



Slika 3.1.1.1. Izgled unutar predajnika - Tx [1]

Predajni deo ima dva procesa koja mora obaviti: AES enkripciju i dodavanje IPv4 zaglavlja.

AES enkripcija predstavlja standard za šifrovan prenos podataka. Podaci prolaze kroz 11 rundi šifrovanja, koristeći pri tome 4 funkcije: SubByte, ShiftRow, MixColumn i AddRoundKey.

SubByte funkcija menja bajt unutar podatka sa bajtom iz posebne tabele nazvane S-box. Predstavlja nelinearnu funkciju pošto izlaz nema nikakve direktne veze sa ulazom. ShiftRow pomera bajtove unutar kolone matričnog prikaza podatka. Podatak od 16 bita se prikaže kao matrica 4x4 i svaki red se ciklično pomera ulevo za jedno mesto više nego prethodni red. MixColumn množi svaku kolonu matrice posebnim polinomom. AddRoundKey obavlja XOR operaciju podatka i prethodno generisanog kljuca. Zbog toga što je ovaj deo koda irelevantan za rezultat ovog rada, neće biti dalje opisan. Detaljnije objašnjenje se može naći u [5].

Celina dodavanja zaglavlja ima zadatak da kreira i „zalepi'' zaglavlje na osnovu koga će prijemni deo odlučiti o daljim koracima u prosleđivanju paketa. Vrednosti koje su unete za određena polja su prikazana u sledećem delu koda:

```
zaglavlje(159 downto 156):=x"4";--polje version, mi radimo sa IPv4
zaglavlje(155 downto 152):=x"5";--polje IHL
zaglavlje(151 downto 146):="000000";--polje DSPC
zaglavlje(145 downto 144):="00";--polje ECN
zaglavlje(143 downto 128):=conv_std_logic_vector(nov_tot_len,16);--polje total length
zaglavlje(127 downto 112):=x"0000";--!!!!!!!!!!!!! polje Identification
```

zaglavlje(111 downto 109):="000";--polje flags

zaglavlje(108 downto 96):="0000000000000";--polje Fragment Offset

zaglavlje(95 downto 88):=x"40";--polje Time To Live

zaglavlje(87 downto 80):=x"11";-- polje Protocol, izabrao sam da protokol bude UDP

zaglavlje(63 downto 0):=izlaz_adr;

--odredjivanja polja Header Checksum

zbir:=("0000" & zaglavlje(15 downto 0))+zaglavlje(31 downto 16)+zaglavlje(47 downto 32)+zaglavlje(63 downto 48)+zaglavlje(95 downto 80)+zaglavlje(111 downto 96)+zaglavlje(127 downto 112)+zaglavlje(143 downto 128)+zaglavlje(159 downto 144);

novi_zbir:=not (zbir(15 downto 0)+zbir(19 downto 16)); --sabiramo prva cetiri bita sa preostala 16 bita i vrsimo negaciju

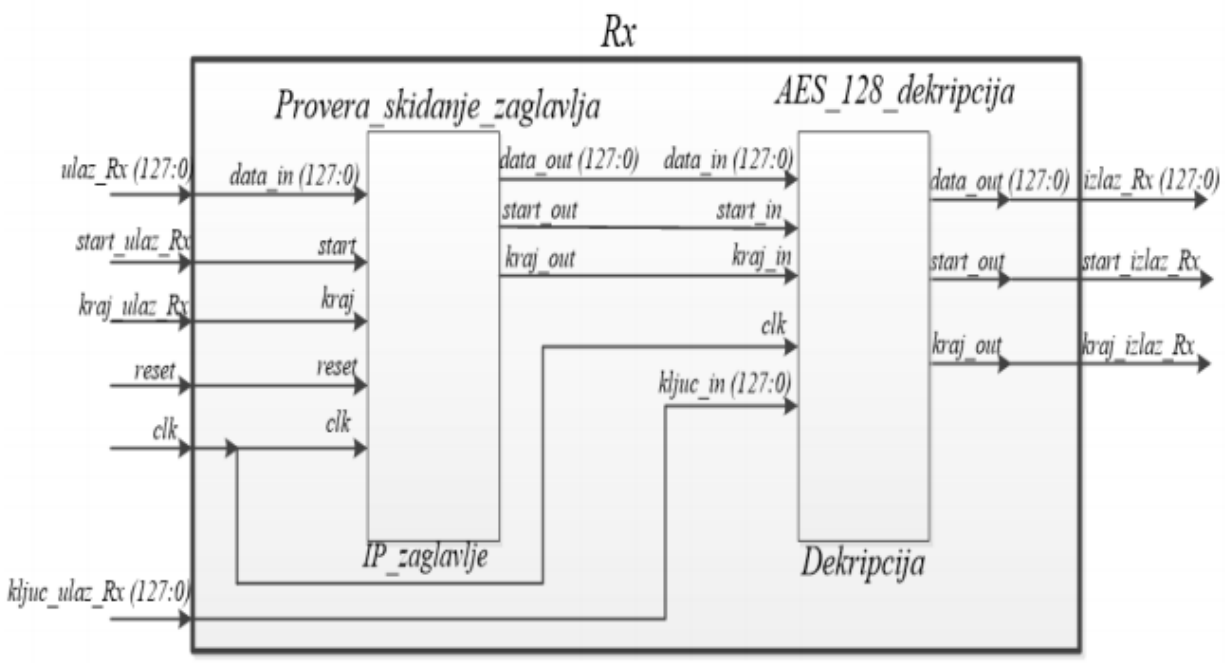
zaglavlje(79 downto 64):=novi_zbir;--polje Header Checksum

Kao što se može primetiti, većina polja ima fiksnu vrednost i ona neće biti promenjena ni u ovom radu. Jedina polja na koja možemo uticati su Total Length, Header Checksum, Source Address i Destination Address. Polja Identification, Flags i Fragmentation Offset, koji se koriste za fragmentaciju prema standardu, takođe, ostaće nepromenjena, iz već objašnjenih razloga.

Ukupna dužina zaglavlja iznosi 160b, dok su izlazi od 128b. Dakle, u prvom taktu za ispisivanje izlaza, proći će prvih 128 bita, a onda u narednom ostalih 32 bita uz dopunjavanje nulama ostalih neiskorišćenih bita.

3.1.2. Prijemni deo

Na slici 3.1.2.1 je prikazan izgled prijemnog dela.



Slika 3.1.2.1. Izgled unutar prijemnika - Rx [1]

Prijemni deo se zasniva na obrnutom procesu analize i sinteze paketa u odnosu na predajni deo. Takođe, sastoji se iz dva dela - skidanje zaglavlja i AES dekripcija.

Deo za skidanje zaglavlja prihvata paket na ulazu u prijemnik. Zatim se analizira zaglavlje koje je nalepljeno na originalni paket paketa. Zaglavlje se smatra odbačenim ukoliko nije zadovoljen barem jedan od navedenih uslova:

1) Header Checksum se ne poklapa sa rezultatom koji je prijemnik dobio računanjem prema zadatom standardu;
2) TTL polje je dostiglo vrednost nula;
3) Paket nije prema IPv4 protokolu.

Ukoliko su svi gore navedeni uslovi zadovoljeni, prelazi se na AES dekripciju.

AES dekripcija predstavlja proces obrnut enkripciji. Takođe se sastoji iz četiri funkcije: AddRoundKey, inverzni MixColumn, inverzni ShiftRows i inverzni SubByte (S-box). AddRoundKey obavlja istu funkciju, kao i istoimena funkcija u predajnom delu, odnosno operaciju XOR sa zadanim ključem. Zatim se radi funkcija slična MixColumn, sa tom razlikom što se koristi drugačija matrica za množenje. Inverzna funkcija ShiftRows radi proces obrnut nego u predajnom delu, odnosno pomera bite za po jedno mesto u desno. Na kraju se vrši zamena bajtova na osnovu tabele S-box sa drugačijim sadržajem nego na početku enkripcije. Postupak se ponavlja 11 puta, kao i u predajnom delu. Zbog toga što bi ovaj deo opteretio rad, neće biti dalje reči o AES dekripciji. Više informacija o AES dekripciji može se naći u [5].

3.2. Analiza rešenja implementacije fragmentacije IP paketa

Promena koja je urađena u ovom radu odnosi se na delove Dodavanje_IP_zaglavlja i Provera_skidanje_zaglavlja. Ulazi i izlazi su ostali nepromenjeni, ali sama funkcija koja se obavlja unutar oba segmenta se razlikuje.

3.2.1. Predajni deo

Unutar predajnog dela iskorišćen je ceo koncept koji je urađen u prethodnom radu. Slanje prvog dela zaglavlja od 128 bita je ostao nepromenjen. Prilikom slanja drugog dela zaglavlja dodaju se nove promenljive, koje će biti od koristi za prijemni deo. Od 96 neiskorišćenih (rezervnih) bita koji se prenose u drugom delu zaglavlja, iskoristićemo 32 bita za parametre koji će se koristiti u prijemnom delu.

```
pom:=f & "1111111111" & id_pak; -- pomocne promenljive f, id tunela (adresa memorije) i id paketa

ch_sum:=not (pom(23 downto 16)+pom(15 downto 8)+pom(7 downto 0)); -- check sum-a nastala od novih promenljivi
```

Nove promenljive su:

1) *id_tunela* – odnosi se na identifikaciju tunela, odnosno para izvorišne i odredišne adrese; kako je za ovu promenljivu odabrano 10 bita, to znači da je moguće kreirati 2^{10}=1024 mogućih tunela, odlučeno je da ova promenljiva predstavlja istovremeno i adresu u RAM memoriji unutar prijemnog dela; ideja je da predajni deo dobije odgovor od nekog drugog modula (ili kontrolne ravni) odgovor u vidu broja koji predstavlja jedinstveni broj kombinacije izvorišne i odredišne adrese ili da ga sam interno kreira; ovakva realizacija prevazilazi temu ovog rada i radi jednostavnosti postavljeno je u testu uvek ista kombinacija adresa i isti id:tunela="1111111111";

2) *id_pak* – odnosi se na identifikaciju paketa; svaki paket će imati „jedinstven" broj kojim će se identifikovati da li dva fragmenta pripadaju istom paketu; izabrano je 12 bita za ovu promenljivu što će biti sasvim dovoljno da se, ukoliko neki paket ostane dosta dugo u prijemnom delu nespojen sa svojim drugim delom, ukloni iz RAM memorije;

```
if (id_pak="111111111111") then
        id_pak:="000000000000";
    else
        id_pak:=id_pak+1;
end if;
```

3) *f* – odnosi se na numeraciju fragmenata; ova promenljiva ima 2 bita informacije, samim tim četiri moguće kombinacije: ''00''- nije fragment, ''01''- prvi deo, ''10''- drugi deo i ''11''- nije u upotrebi.

```
if (f="00" OR f="10") then f:="01"; -- numeracija fragmenta
    else f:="11";
end if;
if (f="00" OR f="10") then f:="00"; -- numeracija fragmenta
    else f:="10";
end if;
```

4) *ch_sum*- ekvivalent polju Check Sum unutar IP zaglavlja; obavlja jednostavnu operaciju negacije sabiranja 3 grupe od 8 bita gore navedenih promenljivi.

```
pom:=f & "1111111111" & id_pak;
ch_sum:=not (pom(23 downto 16)+pom(15 downto 8)+pom(7 downto 0));
```

U polju Total Length, umesto konkretnog broja bita, upisuje se broj 128-bitnih reči korisnog paketa. Kako smo uzeli pretpostavku da je u pitanju Ethernet protokol to znači da je MTU=1500b. Dakle, pošto je 11*128=1408<1500<12*128=1536, izabrano je za ovaj rad MTU=11 128-bitnih reči. Predajni deo će proveravati da li je Total_Length veći od vrednosti MTU. Ukoliko je manje, proći će kao ceo paket i u prijemnom delu će se procesirati kao paket. Ukoliko je veći, predajni deo će ga podeliti na dva dela i prilepiti nove promenljive u drugom delu zaglavlja.

3.2.2. Prijemni deo

Prijemni deo se sastoji iz tri procesa koji se odvijaju u paraleli. Prvi proces vrši prihvatanje pristiglog signala i preusmeravanje na jedan od ostala dva procesa u zavisnosti od podataka koje dobije u zaglavlju. Drugi proces je predviđen samo za fragmente i odnosi se na čuvanje i proveru sa pristiglim delom. Treći proces je namenjen i za pakete i za fragmente i ima funkciju ravnomernog slanja sklopljenih paketa na izlaz iz prijemnog dela.

Prvi proces prihvata paket i tretira ga prema procesu opisanom u prethodnom poglavlju.

```
if (zaglavlje(191 downto 188)="0100") then
        if( zaglavlje(127 downto 120)/="00000000") then
        duz_pak:=zaglavlje(175 downto 160)-2;
        total_length:=duz_pak;
```

```
zbir:=("0000" & zaglavlje(191 downto 176))+zaglavlje(175 downto 160)+zaglavlje(159 downto 144)+zaglavlje(143 downto 128)+zaglavlje(127 downto 112)+zaglavlje(111 downto 96)+zaglavlje(95 downto 80)+zaglavlje(79 downto 64)+zaglavlje(63 downto 48)+zaglavlje(47 downto 32);

novi_zbir:=zbir(15 downto 0)+zbir(19 downto 16);
```

Dodatno proverava i novu Check_Sum kreiranu na osnovu novih promenljivih.

```
pom_zbir:=zaglavlje(31 downto 24)+zaglavlje(23 downto 16)+zaglavlje(15 downto 8);

novi_pom_zbir:=pom_zbir(7 downto 0)+zaglavlje(7 downto 0);
```

Zatim proverava vrednost promenljive *f*.

```
if (f="00") then

        i<=2;

else

ka_fifo_cekanje<=conv_std_logic_vector(0,78) & adr_mem & id_paketa & merac_takta & f & duz_pak ;

upis_fifo_cek<='1';

i<=3;

end if;
```

Ukoliko je *f*="00" tretira se kao paket. U tom slučaju, nadolazeće 128-bitne reči se prosleđuju u FIFO modul nazvan *fifo_paket*. Ispred svakog 128-bitnog signala lepe se još dva bita koji daju naznaku da li je početak, kraj ili sredina paketa. Prvi bit označava "start", a drugi bit "kraj". Dakle, numeracija je sledeća: "10"- start, "01"- kraj i "00"- sredina paketa.

```
if(i=2) then

        if(duz_pak=total_length) then

                ka_fifo_paket<="10" & data_in; -- pocetak paketa

                duz_pak:=duz_pak-1;

                upis_paket<='1';

        else

                ka_fifo_paket<="00" & data_in; -- sredina paketa

                duz_pak:=duz_pak-1;

        end if;

        if(kraj='1') then

                ka_fifo_paket<="01" & data_in; -- kraj paketa

                i<=0;

        end if;

end if;
```

Ukoliko je *f* različito od vrednosti "00", tretira se kao fragment. U tom slučaju, slanje ka izlazu mora da se zakasni do onog trenutka kada naiđe drugi deo paketa. Za to su nam potrebni i nove promenljive, kao i ceo pristigli fragment. Kako fragment, ne mora biti jednak MTU, moguća je potreba za dodatnom obradom podataka. Takođe, potrebna je provera da li se u RAM memoriji već nalazi drugi deo paketa, za šta je potrebno još dodatno vreme. Iz navedenih razloga, umesto obrade fragmenta u trenutku pristizanja, šalje se u FIFO nazvan *fifo_fragment_cekanje*, koji služi kao bafer. Kao prvi podatak u FIFO, šalju se nove promenljive uz dodatak još jedne promenljive nazvane *merac_takta*. Promenljiva *merac_takta* služi za proveru "starosti" fragmenta. Ukoliko se

fragment nalazi isuviše dugo unutar memorije, možemo smatrati da je drugi deo izgubljen i samim tim odbaciti taj fragment unutar memorije i zameniti novim fragmentom.

```
if(i=3) then
        ka_fifo_cekanje<=data_in;
        if(kraj='1') then
                i<=0;
        end if;
end if;
```

Drugi proces se odnosi na analizu fragmenata. Iz *fifo_fragment_cekanje* čita se zapis sa promenljivama. Pronalazi se adresa memorije (*id_tunela*) i postavlja se kao adresa za upis i ispis iz memorije.

```
adresa:=conv_integer(iz_fifo_cek(49 downto 40));
```

Proverava se promenljiva *pom* koja nosi podatke da li je upisano nešto u toj lokaciji memorije.

```
if (pom(adresa)='0') then
        dalje_fifo_cek<='1';
        j<=5;
else
        j<=2;
        adr_mem_ispis_1<=iz_fifo_cek(49 downto 40);
        adr_mem_ispis_2<=iz_fifo_cek(49 downto 40);
end if;
```

Promenljiva *pom* se sastoji od 1024 bita koji nulama i jedinicama pokazuju da li je nešto upisano na toj adresi, pri čemu pozicija bita odgovara lokaciji memorije. Ukoliko je na odgovarajućoj poziciji nula (nema ništa upisano) pristupa se proceduri bezuslovnog upisa u memoriju. Ukoliko je na toj poziciji jedinica (postoji podatak na toj adresi memorije) pristupa se proceduri čitanja iz memorije i zatim upoređivanjem sa novim fragmentom.

Memorija se sastoji od dve paralelne RAM memorije. Prva memorija sadrži deo fragmenta sa promenljivama, dok druga sadrži preostali deo od celog fragmenta. Maksimalna moguća širina RAM memorije koja se može kreirati u programu je 1152=9*128, što tačno odgovara maksimalnoj veličini fragmenta ukoliko je MTU=11 (MTU se sastoji od zbira fragmenta i zaglavlja, a kako je zaglavlje veličine dve 128-bitne reči, ostalih 9 pripada fragmentu). Prva memorija ima širinu od 40 bita. Ukoliko je fragment manji od 1152 bita, na kraju mu se dodaje niz nula.

U zavisnosti od ishoda upoređivanja fragmenta u *fifo_fragment_cekanje* i RAM memorije moguća su 4 slučaja:

1) Fragment u *fifo_fragment_cekanje* je prvi deo, u RAM memoriji drugi deo;
2) Fragment u RAM memoriji je prvi deo, fragment u *fifo_fragment_cekanje* je drugi deo;
3) Fragment u *fifo_fragment_cekanje* treba upisati umesto trenutnog u RAM memoriji;
4) Fragment u RAM memoriji treba vratiti u memoriju, a iz *fifo_fragment_cekanje* odbaciti.

U prvom slučaju čita se sadržaj iz *fifo_fragment_cekanje* onoliko puta koliko je navedeno u podatku o dužini fragmenta koji se nalazi kao promenljiva za analizu fragmenata. Dotle, podatak iz memorije se čuva u pomoćnoj promenljivoj. Pročitani red iz *fifo_fragment_cekanje* se prosleđuje u novi FIFO, nazvan *fifo_fragment* i koji ima istu funkciju kao *fifo_paket*.

```
if (duz_fr1>0) then
        if (duz_fr1=duz_fr1_max) then
                ka_fifo_fragment<="10" & iz_fifo_cek;
                upis_fragment<='1';
                dalje_fifo_cek<='1';
                duz_fr1:=duz_fr1-1;
        else
                ka_fifo_fragment<="00" & iz_fifo_cek;
                upis_fragment<='1';
                dalje_fifo_cek<='1';
                duz_fr1:=duz_fr1-1;
        end if;
```

Kada se pročita i poslednji niz od 128 bita koji se nalazi u *fifo_fragment_cekanje*, prelazi se na deo paketa iz RAM memorije. Iz pomoćne promenljive čita se najviših 128 bita. Zatim se niža 1024 bita preslikavaju u 1024 bita najviša bita. Na taj način se pomeraju podaci iz memorije i čita se pravilan redosled reči kako su stizali na ulaz. Proces se ponavlja dok dužina fragmenta u memoriji (čiji podatak smo pročitali iz prve memorije) ne pokaže 0. Zatim se u promenljivoj *pom* postavlja 0 na mestu gde je bila adresa podatka iz RAM memorije. Na ulaz u *fifo_fragment* se šalje 128-bitna reč sa istim principom lepljenja dva bita za start i kraj paketa.

```
if (duz_fr2>1) then
        ka_fifo_fragment<="00" & iz_memorije(1151 downto 1024); -- sredina paketa
        upis_fragment<='1';
        pomeraj:=iz_memorije(1023 downto 0);
        iz_memorije(1151 downto 128):=pomeraj;
        duz_fr2:=duz_fr2-1;
        dalje_fifo_cek<='0';
else
        ka_fifo_fragment<="01" & iz_memorije(1151 downto 1024);
        upis_fragment<='1';
        dalje_fifo_cek<='0';
        pom(adresa):='0';
        nastavi:='0';
        j<=0;
end if;
```

Drugi slučaj predstavlja obrnut slučaj u odnosu na prvi. Procedura je potpuno ista, sa tom razlikom što prvo se čita pomoćna promenljiva za podatak iz RAM memorije, a zatim fragment iz *fifo_fragment_cekanje*.

```
if (duz_fr2<=2) then dalje_fifo_cek<='1'; end if;
if (duz_fr2>=1) then
        if (duz_fr2=duz_fr2_max) then
                ka_fifo_fragment<="10" & iz_druge_memorije(1151 downto 1024);
                upis_fragment<='1';
                pomeraj:=iz_druge_memorije(1023 downto 0);
                iz_memorije(1151 downto 128):=pomeraj;
                duz_fr2:=duz_fr2-1;
        else
                ka_fifo_fragment<="00" & iz_memorije(1151 downto 1024);
                upis_fragment<='1';
                pomeraj:=iz_memorije(1023 downto 0); --shiftovanje fragmenta
                iz_memorije(1151 downto 128):=pomeraj;
                pom(adresa):='0';
                duz_fr2:=duz_fr2-1;
        end if;
else
        if (duz_fr1>1) then
                ka_fifo_fragment<="00" & iz_fifo_cek; -- sredina paketa
                upis_fragment<='1';
                duz_fr1:=duz_fr1-1;
                dalje_fifo_cek<='1';
                if (duz_fr1<=1) then dalje_fifo_cek<='0'; end if;
        else
                ka_fifo_fragment<="01" & iz_fifo_cek;
                upis_fragment<='1';
                dalje_fifo_cek<='0';
                j<=0;
        end if;
end if;
```

Treći deo se odnosi na slučaj kada je fragment u memoriji dovoljno dugo da se može smatrati zastarelim i samim tim ga odbacujemo i zamenjujemo sa fragmentom iz *fifo_fragment_cekanje*. Razlika između trenutka pristizanja novog i starog fragmenta mora biti veća od 50 da bi mogao da se ispuni uslov za ovaj slučaj. Vrednost 50 je izabrana proizvoljno (može biti promenjena u zavisnosti od učestalosti slanja paketa sa ovim *id_tunela*). Upis se vrši uz pomoć pomoćne promenljive za upis u memoriju. Koristi se sličan princip kao za ispis, sa tom razlikom što se pomoćna promenljiva stalno resetuje na nulu pre početka rada sa njom. Kako veličina paketa

može varirati, tako i početak fragmenta unutar pomoćne promenljive može biti na različitim pozicijama. Na osnovu vrednosti dužine fragmenta, možemo odrediti početak fragmenta i njega izdvojiti, a potom nalepiti ostatak od 128 bita sa nulama, kako bismo dobili situaciju kao u prvom slučaju. Ovakav razlog upisa u memoriju je neophodan iz razloga što razvojno okruženje ne može manipulisati delom niza ukoliko mu bar jedna pozicija u nizu (početna ili krajna) nije konstantna. Kako varira dužina paketa, varira i početak i kraj, ako bismo ostavili da nule budu na početku. Postoje dva moguća rešenja:

1) Imati fiksni početak – prvo se upisuje fragment redosledom kako nailazi i na kraju se nalepe nule;
2) Imati fiksni kraj – prvo se nanižu nule, a zatim fragment u obrnutom redosledu u odnosu na to kako nailazi.

Izabran je prvi način zbog preglednosti i jednostavnosti verifikacije upisa. Nakon toga, fragment sa dodatim nulama se upisuje u drugu memoriju, a u prvu se upisuju podaci sa promenljivama koje su potrebne za kasniju obradu.

```
if (duz_fr1>1) then
        za_upis_pomeraj:=u_memoriju_pom(1023 downto 0);
        u_memoriju_pom:=za_upis_pomeraj & iz_fifo_cek;
        duz_fr1:=duz_fr1-1;
        dalje_fifo_cek<='1';
        if (duz_fr1=1) then
                dalje_fifo_cek<='0';
        end if;
 else
        za_upis_pomeraj:=u_memoriju_pom(1023 downto 0);
        u_memoriju_pom:=za_upis_pomeraj & iz_fifo_cek;
        if (duz_fr1_max<9) then
                pocetak:=conv_integer(duz_fr1_max)*128-1;
                kraj:=1151-pocetak;
                u_memoriju_temp(1151 downto kraj):=u_memoriju_pom(pocetak downto 0);
                u_memoriju_pom:=u_memoriju_temp;
        end if;
        adr_mem_upis_1<=zaglavlje_cek(49 downto 40);
        adr_mem_upis_2<=zaglavlje_cek(49 downto 40);
        upisi_u_prvu<="1";
        upisi_u_drugu<="1";
        u_prvu_memoriju<=zaglavlje_cek(39 downto 0);
        u_drugu_memoriju<=u_memoriju_pom;
        pom(adresa):='1';
        j<=0;
        nastavi:='0';
```

```
end if;
```

Poslednji slučaj je da prethodni nije zadovoljen. U tom slučaju redovi od 128 bita se šalju u promenljivu odbačen, koja služi kao ''kanta za otpatke'' i u nju će se upisivati i prebrisavati svi podaci vezani za taj fragment.

```
if (j=6) then
        upisi_u_prvu<="0";
        upisi_u_drugu<="0";
        if (duz_fr1>0) then
                odbacivanje:=iz_fifo_cek;
                duz_fr1:= duz_fr1-1;
                if (duz_fr1=2) then
                        dalje_fifo_cek<='0';
                end if;
        else
                j<=0;
        end if;
end if;
```

Kada se dva dela fragmenta spoje (bilo u prvom ili drugom slučaju) šalju se ka *fifo_fragment* koji zajedno sa *fifo_paket* predstavlja treći proces. Ovaj proces ima zadatak da ispisuje sadržaje iz ova dva FIFO modula prema round-robin principu. Na osnovu promenljive *robin* se određuje iz kog FIFO modula će se čitati podaci. Ukoliko je *robin*=''01'' čita se iz fifo_paket, a ukoliko je *robin*=''10'' čita se iz fifo_fragment. Vrednost robin=''00'' je uzeta kao međustanje za prelazak na čitanje iz *fifo_fragment* zbog same pozicije koda za ovaj slučaj. Takođe, vrednost robin=''11'' je uzeta kao međufaza prilikom prelaska na čitanje iz *fifo_paket*. Pozicije ova dva stanja su postavljena na osnovu pozicije u kodu za dva osnovna stanja. Kada se završi čitanje iz jednog od FIFO modula, odmah se prelazi na drugi. Ukoliko je neki od njih prazan, prebacuje se na drugi.

```
if(clk' event and clk='1') then
        start_out<='0';
        kraj_out<='0';
        citaj_paket<='0';
        citaj_fragment<='0';
        data_out<=(others=>'0');
        if(robin="00") then --medjufaza za robin="10"
                if (dalje_pom='1') then
                        robin:="10";
                        dalje_pom:='0';
                else
                        dalje_pom:='1';
                        citaj_fragment<='1';
                end if;
```

```
end if;


if (robin="01") then -- citanje iz fifo_paket
        if (prazan_fifo_paket='1' and jos_paket='0') then --ukoliko je fifo_paket prazan prelazi se na fifo_fragment
                robin:="10";
                citaj_fragment<='1';
        else
                if (dalje_paket='1') then
                        jos_paket:='1'; --kad se isprazni fifo_paket naznaka da ima jos jedan deo paketa da se iscita
                        citaj_paket<='1';
                        procitano:=procitano_paket;
                        start_kraj:=procitano(129 downto 128);
                        data_out<=procitano(127 downto 0);
                        if (start_kraj="10") then -- znak za start paketa
                                start_out<='1';
                                kraj_out<='0';
                        end if;
                        if (start_kraj="01") then --znak za kraj paketa i prelazak na fifo_fragment
                                kraj_out<='1';
                                start_out<='0';
                                citaj_paket<='0';
                                robin:="10";
                                citaj_fragment<='1';
                                jos_paket:='0';
                                dalje_paket:='0';
                        end if;
                        if (start_kraj="00") then --sredina paketa
                                kraj_out<='0';
                                start_out<='0';
                        end if;
                else --zakasnjenje za citanje iz fifo_paketa
                        dalje_paket:='1';
                        robin:="11"; --prelazak u medjufazu
                end if;
                end if;
        end if;
```

```
if (robin="10") then --citanje iz fifo_fragment
        if(prazan_fifo_fragment='1' and jos_fragment='0') then -- ukoliko je fifo_fragment prazan prelazi se na fifo_paket
                robin:="01";
                citaj_fragment<='0';
        else
                if (dalje_fragment='1') then
                        jos_fragment:='1'; --kad se isprazni fifo_fragment naznaka da ima jos jedan deo paketa da se iscita
                        procitano:=procitano_fragment;
                        start_kraj:=procitano(129 downto 128);
                        data_out<=procitano(127 downto 0);
                        citaj_fragment<='1';
                if (start_kraj="10") then --znak za start paketa
                        start_out<='1';
                        kraj_out<='0';
                end if;
                if (start_kraj="01") then --znak za kraj paketa i prelazak na fifo_paket
                        kraj_out<='1';
                        start_out<='0';
                        jos_fragment:='0';
                        citaj_fragment<='0';
                        robin:="01";
                        dalje_fragment:='0';
                end if;
                if (start_kraj="00") then --sredina paketa
                        kraj_out<='0';
                        start_out<='0';
                end if;
        else
                dalje_fragment:='1';
                robin:="00"; --prelazak u medjufazu
        end if;
        end if;
end if;
if(robin="11") then --medjufaza za robin="01"
        if (dalje_pom='1') then
                robin:="01";
                citaj_paket<='1'; neophodan korak za pravilno citanje
```

```
                    dalje_pom:='0';
                else
                    dalje_pom:='1';
                    citaj_paket<='1';
                end if;
            end if;
        end if;
```

Na ovaj način je ostvaren ravnomerni ispis celih paketa i onih nastalih spajanjem dva fragmenta, bez monopolisanja jednog od ova dva tipa. Kada se ne čita ni iz jednog FIFO modula, šalju se sve nule na izlaz.

4. VERIFIKACIJA DIZAJNA I ANALIZA PERFORMANSI

4.1. Verifikacija dizajna

U okviru ovog potpoglavlja biće prikazana verifikacija rada celog dizajna. Verifikacija će se raditi iz tri dela. Razlog je taj što ukoliko ih povežemo na red nećemo moći potvrditi validan rad prijemnog dela jer ćemo uvek imati isti slučaj testiran. Sve signale prikazaćemo u heksadecimalnom obliku.

Prvi test treba da nam prikaže regularan rad predajnog dela. Na ulaz u predajnik dovešćemo signal koji je veći od MTU. Kao rezultat, trebalo bi dobiti dva podeljena fragmenata. Prvi deo biće veličine MTU, dok drugi će prenositi ostatak paketa. Ulaz će biti manji od 2*MTU zato što dizajn je predviđen za rad sa maksimalno 2 fragmenta. Na ulaz ćemo dovesti sledeći signal:

izvorisna_adresa<=x"9c2a700b";

odredisna_adresa<=x"a0f3c1cc";

data_in<=x"01000000000000000000000000000000";

data_in<=x"11111111111111111111111111111111";

data_in<=x"21222222222222222222222222222222";

data_in<=x"31333333333333333333333333333333";

data_in<=x"41444444444444444444444444444444";

data_in<=x"51555555555555555555555555555555";

data_in<=x"61666666666666666666666666666666";

data_in<=x"71777777777777777777777777777777";

data_in<=x"81888888888888888888888888888888";

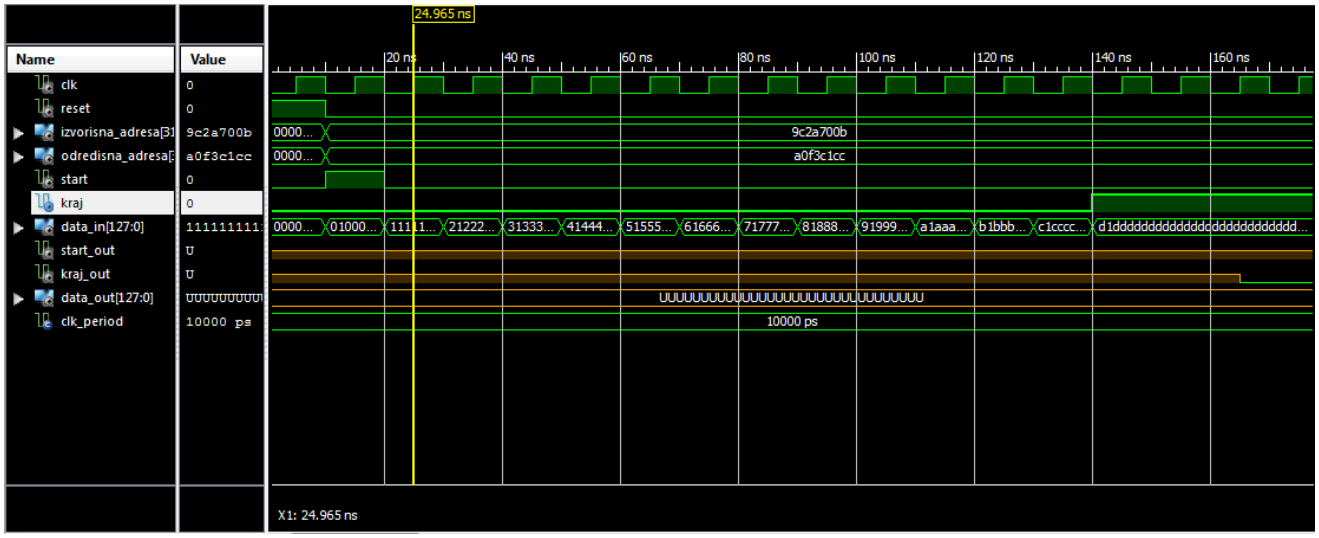
data_in<=x"91999999999999999999999999999999";

data_in<=x"a1aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";

data_in<=x"b1bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb";

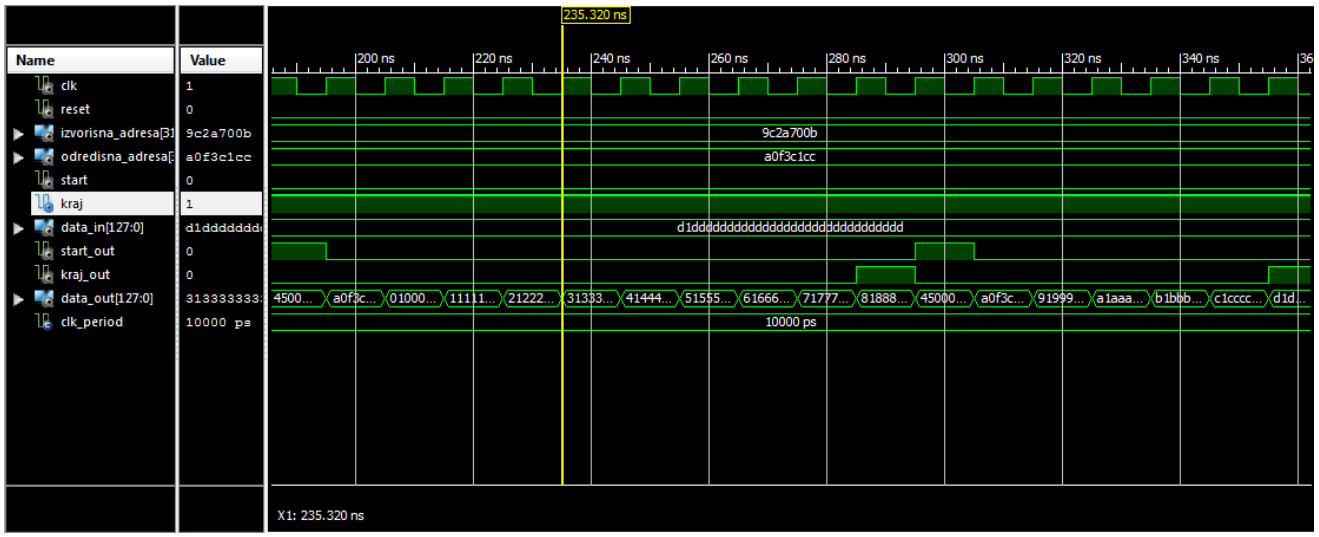
data_in<=x"c1ccccccccccccccccccccccccccccc";

data_in<=x"d1ddddddddddddddddddddddddddddd";



Slika 4.1.1. Prikaz signala na ulazu u predajnik

Na slici 4.1.1 se vidi naš ulazni signal (namerno odabran zbog lakšeg praćenja na izlazu). Prema dizajnu, ulazni podatak treba da se podeli na deo od 9*128 bita i na deo od 5*128 bita.



Slika 4.1.2. Prikaz signala na izlazu iz predajnika – podeljen paket

Kao što možemo primetiti predajnik je uspešno podelio paket od 14*128 bitnih reči, na dva dela, pri čemu je prvi deo veličine MTU, a drugi ima ostatak prvobitnog paketa (slika 4.1.2).

Drugi test ima ideju da prikaže rad prijemnog dela pri čemu na ulaz ne dolaze vezano dva dela istog paketa. Na ulaz ćemo dovesti 3 različita paketa, pri čemu su dva podeljena na dva fragmenta, dok je treći paket doveden u celosti na ulaz. Redosled pristizanja fragmenata je takođe izmešan kako bi prikazali da ulaz i izlaz zavise od trenutka pristizanja svih delova, tako da neki paket koji je ušao kasnije može izaći među prvima i obrnuto. Na ulaz dovodimo fragmente sledecim redosledom:

data_in<=x"92999999999999999999999999999999";

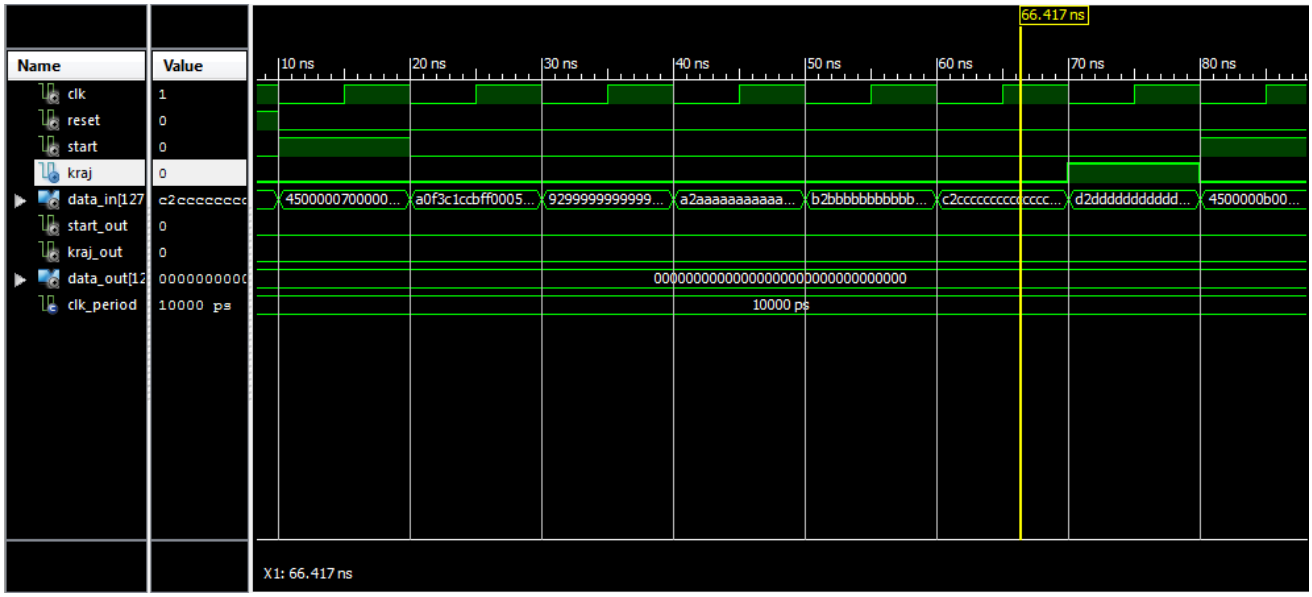
data_in<=x"a2aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";

data_in<=x"b2bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb";

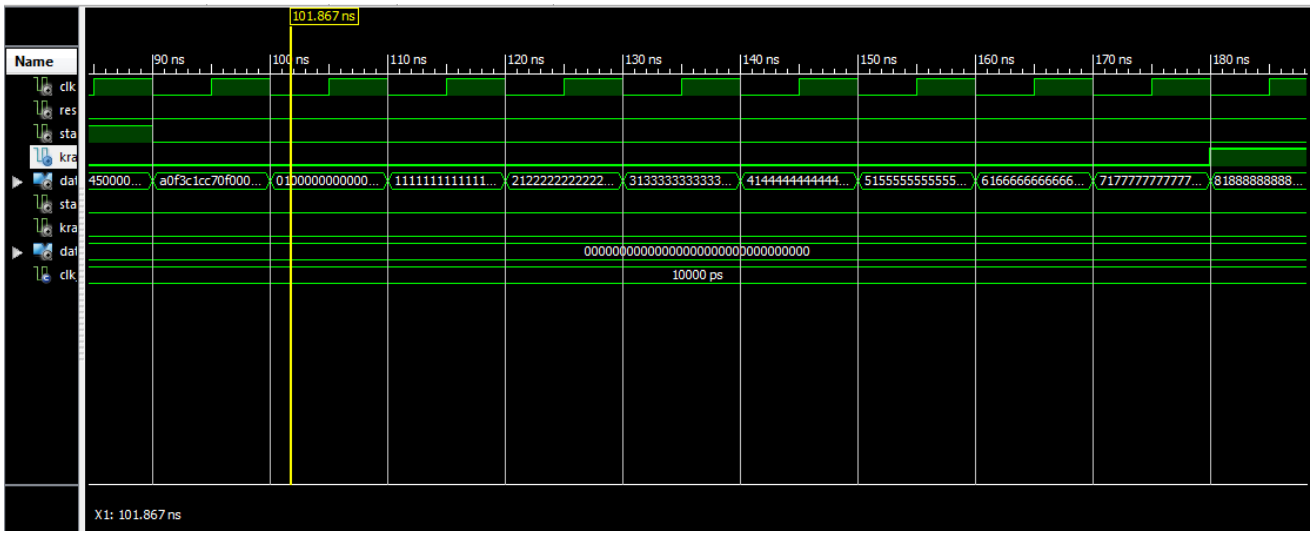
```
data_in<=x"c2ccccccccccccccccccccccccccccc";
data_in<=x"d2ddddddddddddddddddddddddddddd";

data_in<=x"01000000000000000000000000000000";
data_in<=x"11111111111111111111111111111111";
data_in<=x"21222222222222222222222222222222";
data_in<=x"31333333333333333333333333333333";
data_in<=x"41444444444444444444444444444444";
data_in<=x"51555555555555555555555555555555";
data_in<=x"61666666666666666666666666666666";
data_in<=x"71777777777777777777777777777777";
data_in<=x"81888888888888888888888888888888";

data_in<=x"03000000000000000000000000000000";
data_in<=x"13111111111111111111111111111111";
data_in<=x"23222222222222222222222222222222";
data_in<=x"33333333333333333333333333333333";
data_in<=x"43444444444444444444444444444444";
data_in<=x"53555555555555555555555555555555";
data_in<=x"63666666666666666666666666666666";
data_in<=x"73777777777777777777777777777777";
data_in<=x"83888888888888888888888888888888";

data_in<=x"91999999999999999999999999999999";
data_in<=x"a1aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
data_in<=x"b1bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb";
data_in<=x"c1ccccccccccccccccccccccccccccc";
data_in<=x"d1ddddddddddddddddddddddddddddd";

data_in<=x"02000000000000000000000000000000";
data_in<=x"12111111111111111111111111111111";
data_in<=x"22222222222222222222222222222222";
data_in<=x"32333333333333333333333333333333";
data_in<=x"42444444444444444444444444444444";
data_in<=x"52555555555555555555555555555555";
data_in<=x"62666666666666666666666666666666";
```

data_in<=x"72777777777777777777777777777777";

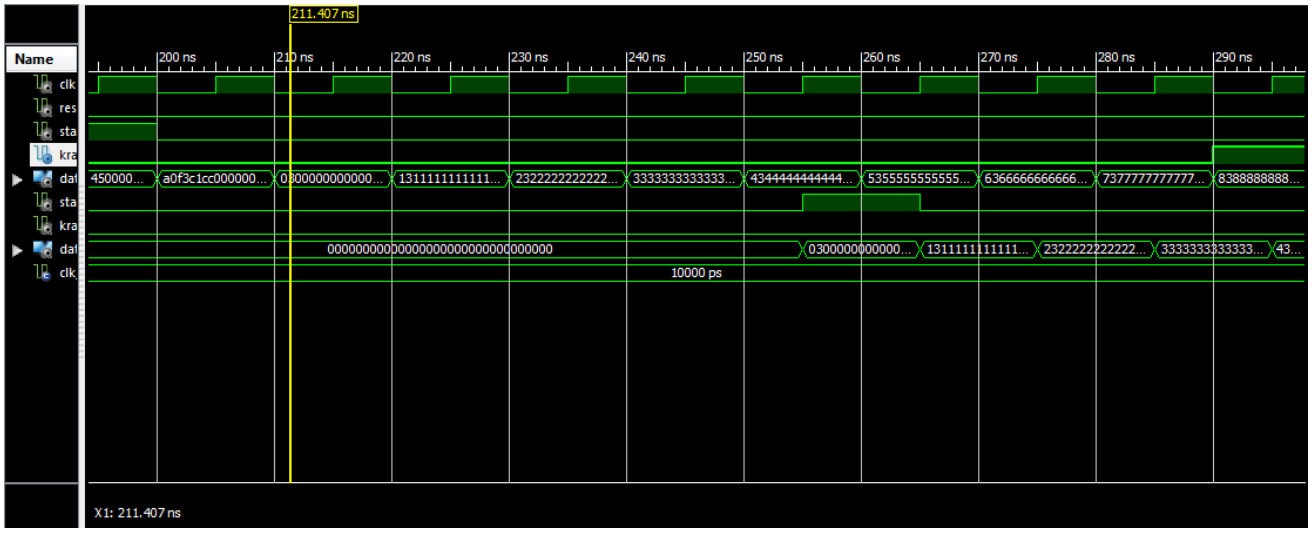
data_in<=x"82888888888888888888888888888888";



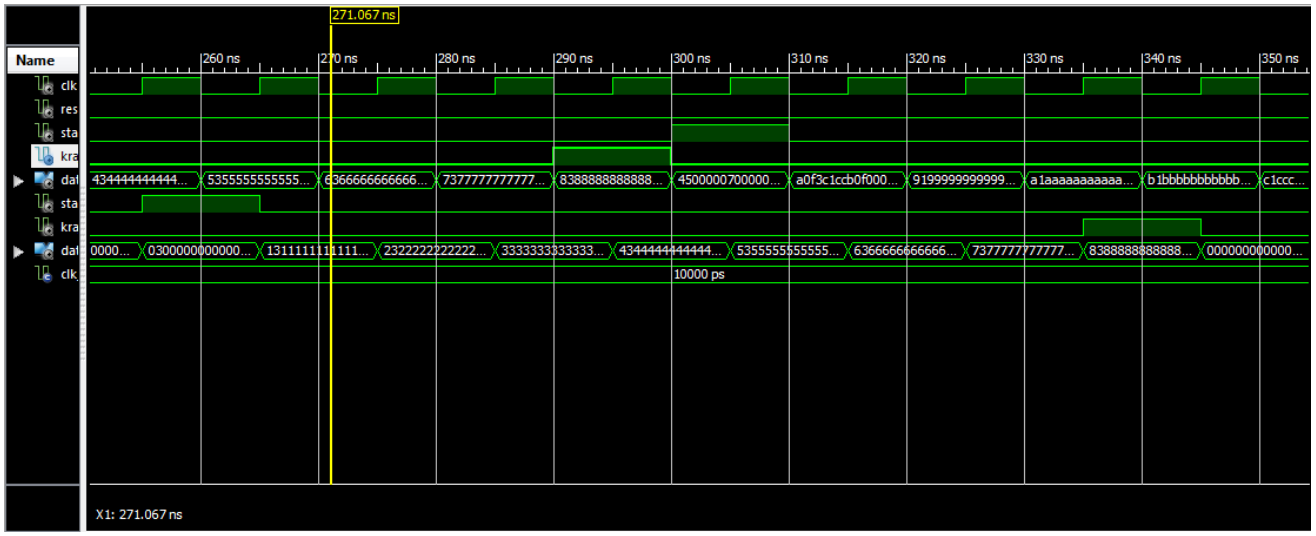
Slika 4.1.3. Drugi deo drugog paketa na ulazu u prijemnik



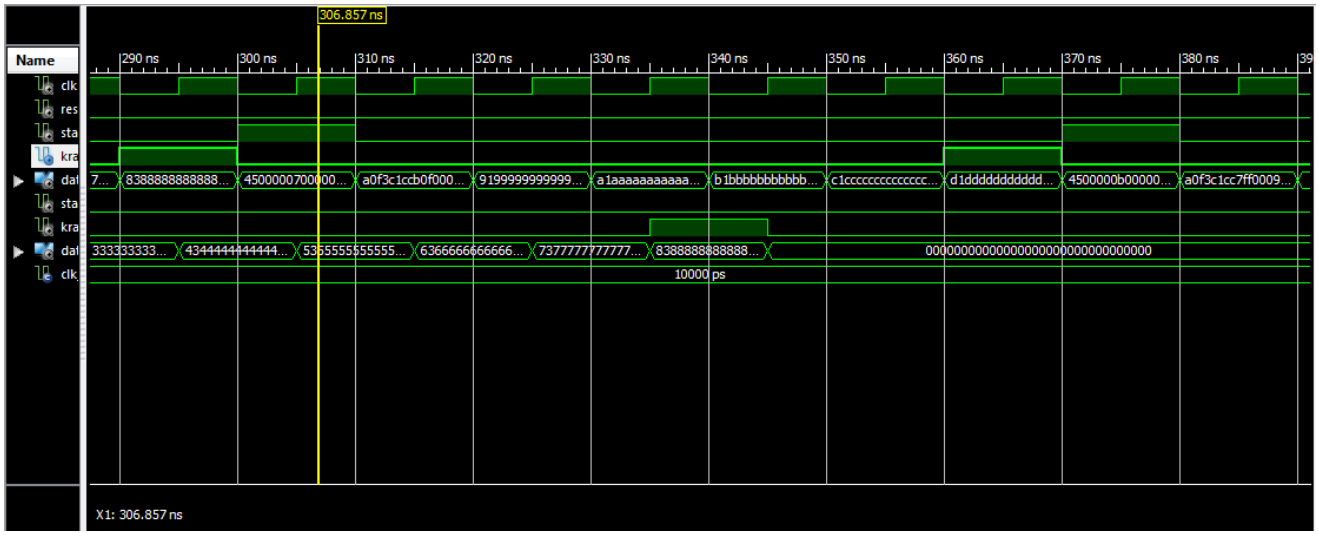
Slika 4.1.4. Prvi deo prvog paketa na ulazu u prijemnik



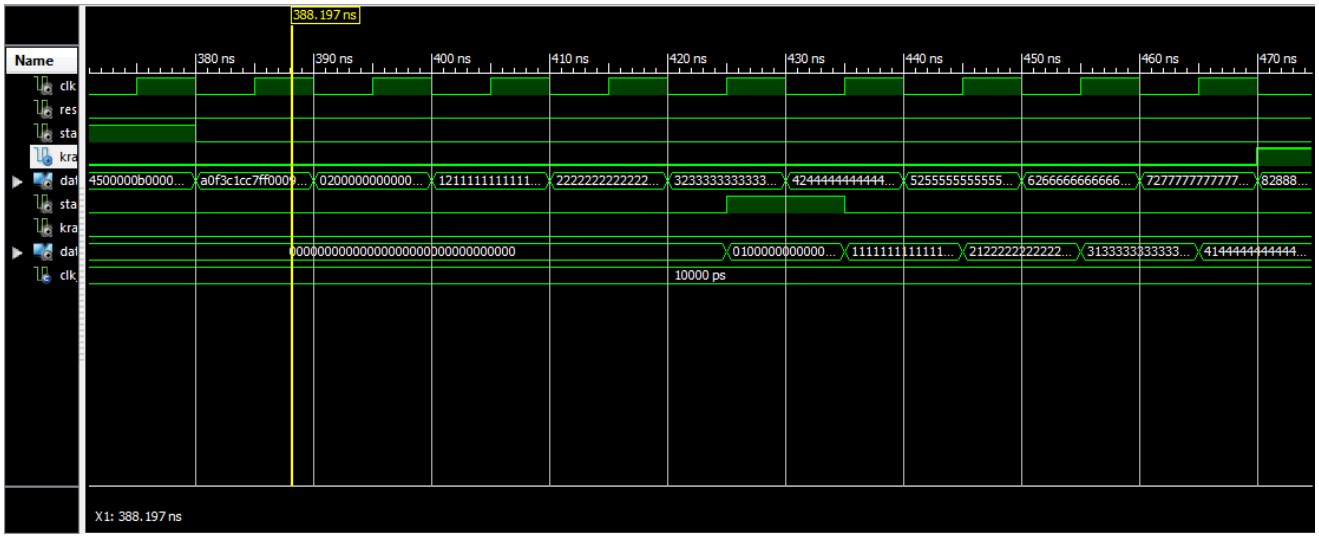
Slika 4.1.5. Treći paket na ulazu u prijemnik i početak ispisa iz njega



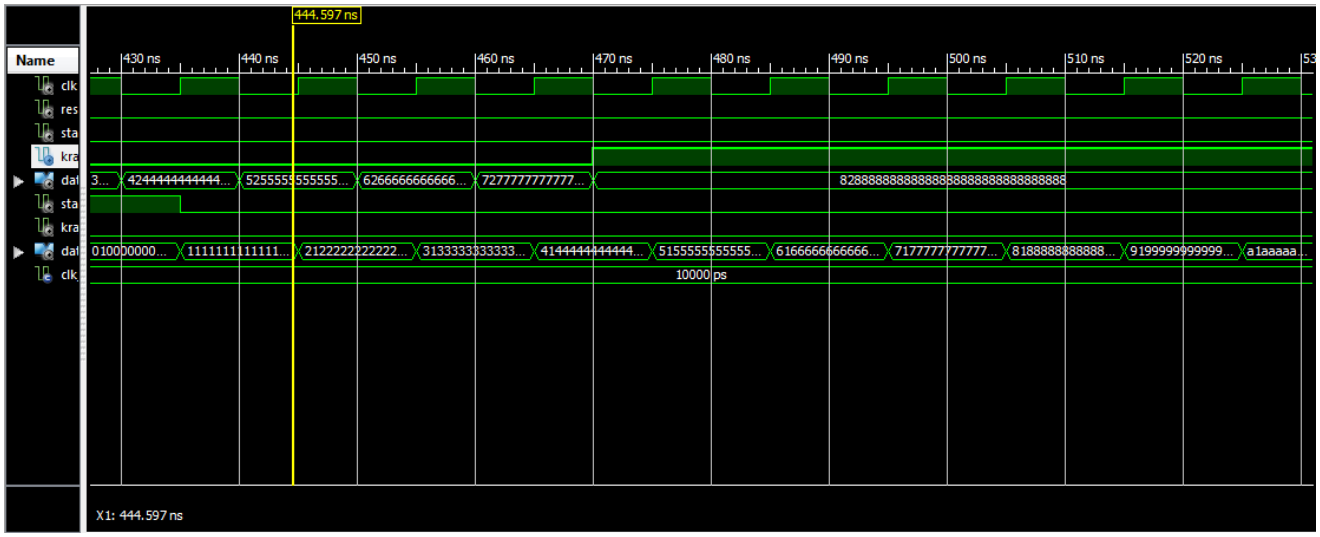
Slika 4.1.6. Ispis trećeg paketa i pristizanje drugog dela prvog paketa na ulaz prijemnika



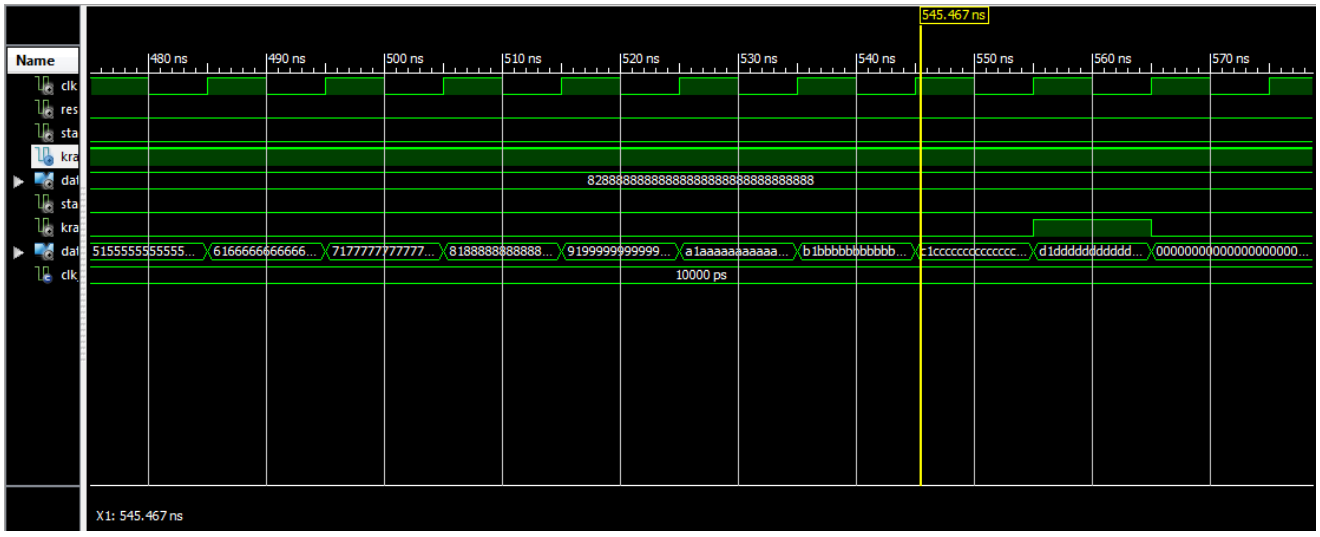
Slika 4.1.7. Pristizanje drugog dela prvog paketa na ulaz u prijemnik



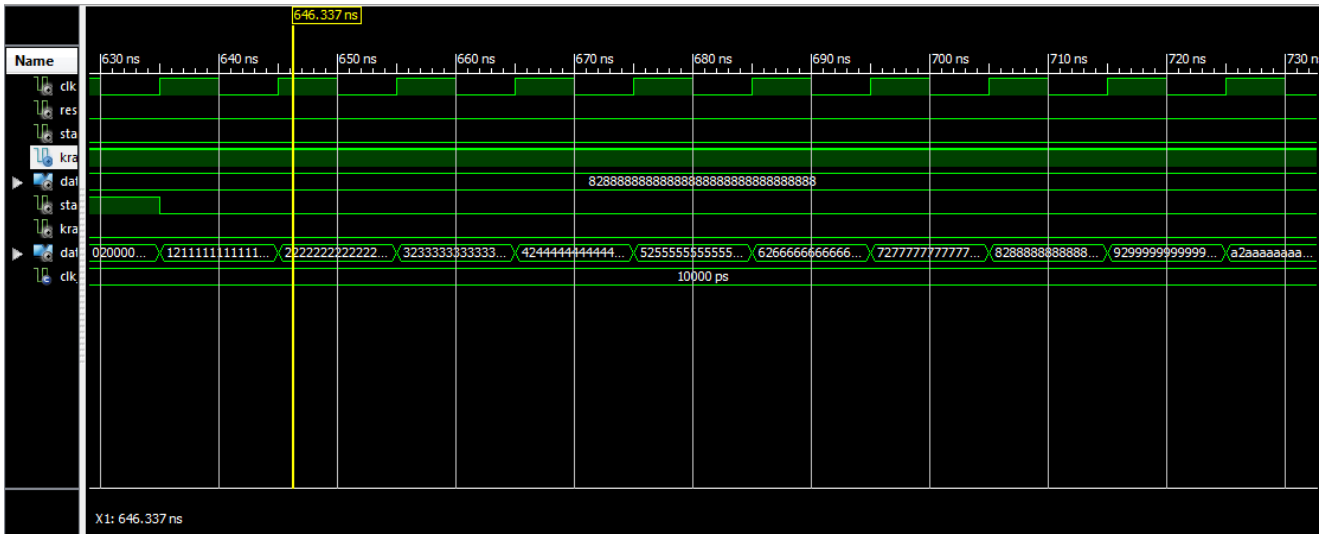
Slika 4.1.8. Pristizanje prvog dela drugog paketa na ulaz prijemnika i početak ispisa prvog paketa



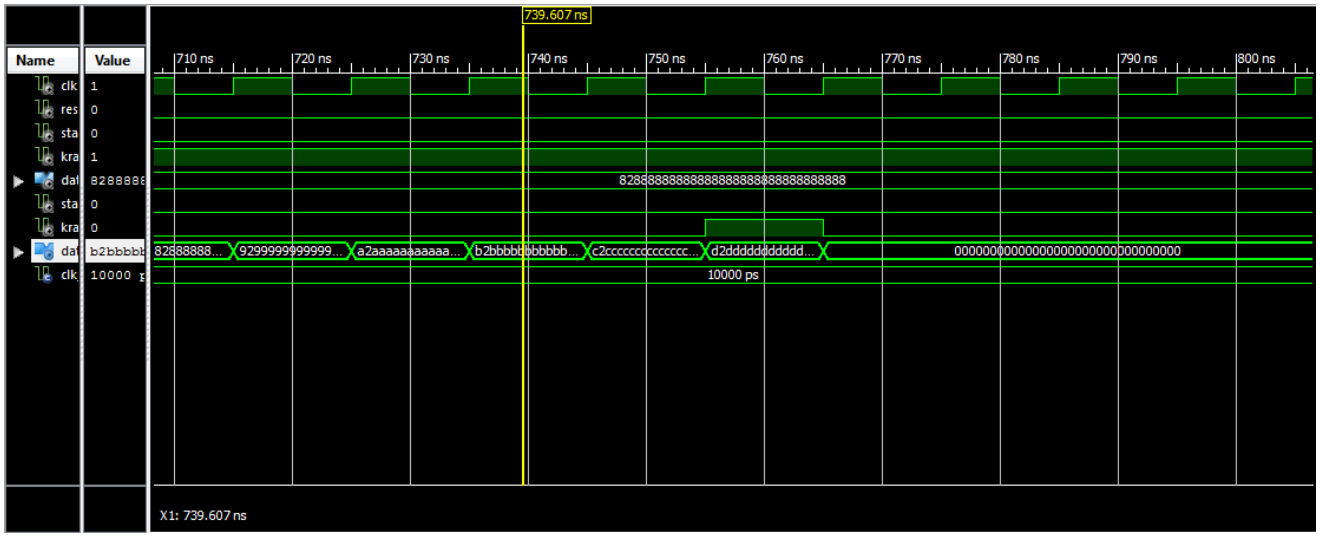
Slika 4.1.9. Ispis prvog paketa



Slika 4.1.10. Ispis prvog paketa - nastavak



Slika 4.1.11. Ispis drugog paketa



Slika 4.1.12. Ispis drugog paketa - nastavak

Slike 4.1.3-4.1.12 nam pokazuju rezultate ovog drugog testa. Kao što možemo primetiti, prvo je pristigao drugi deo drugog paketa, ali pošto je njegov prvi deo poslednji stigao na ulaz, dovelo je do toga da je drugi paket poslednji napustio prijemnik. Logično, prvi koji je napustio prijemnik je treći paket koji se u celosti pojavio na ulaz prijemnika.

Poslednji test se odnosi na test sa AES enkripcijom. Treba pokazati da izmena koja je urađena u ovom dizajnu ne menja konačni rezultat testa bez ovih izmena. Ovaj test bi bio ekvivalent testu povezivanja predajnika i prijemnika ovog dizajna, sa razlikom u šifrovanju podataka. Na ulaz su dovedeni sledeći podaci (navedeni su samo korisni podaci, bez ključa šifrovanja, jer je irelevantan ovom radu):

Prvi paket

Izvorišna adresa: 9c2a700b

Odredišna adresa: a0f3c1cc

cdb9c7ef0fd5757ebdc8052949d1de43

c046cccfb22e5985238cbfecc9e5f5e7

9439bd8da20d050f8beccbf35bf8bc71

Drugi paket

Izvorišna adresa: a0f3c1cc

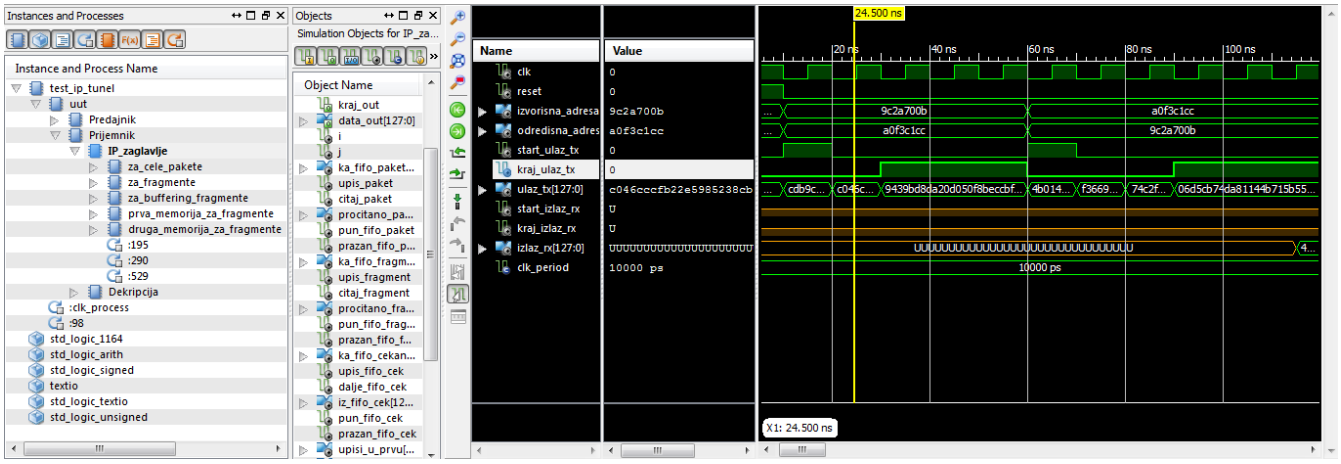
Odredišna adresa: 9c2a700b

4b01461c400643e3ef4124e787971bfb

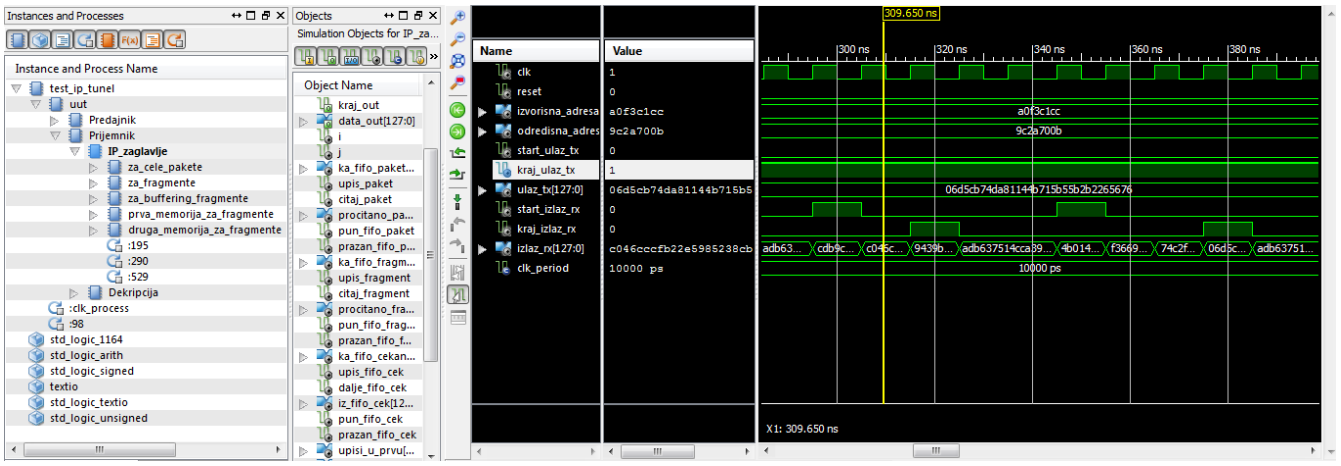
f36695e05659286177d58333816edc6a

74c2fa94ed75c18776778411f02f6aa0

06d5cb74da81144b715b55b2b2265676



Slika 4.1.13. Ulazni signali



Slika 4.1.14. Izlazni signali

Kao što može da se primeti iz gornjih slika, ulaz u predajnik je identičan izlazu iz prijemnika. Ovim potvrđujemo da je dizajn ispravan i ne remeti rad dizajna sa AES enkripcijom.

4.2. Analiza performansi

U ovom radu je korišćen FPGA čip XC5VLX300T familije Virtex5, identičan onome koji je korišćen u master radu Aleksandra Dželatovića. U tabelama 4.2.1 i 4.2.2 prikazane su performanse pre i nakon promene celokupnog dizajna.

Tabela 4.2.1. Performanse dizajna bez fragmentacije

	Korišćeno	Dostupno	Procenat iskorišćenosti
Broj slajs registara	8118	207360	3%
Broj slajs LUT-ova	35578	207360	17%
Broj potpuno iskorišćenih parova LUT-FF	7313	60732	20%
Broj pinova	842	960	87%
Broj blokova RAM/FIFO	6	324	1%
Broj globalnih taktova (BUFG/BUFGCTRL)	1	32	3%

Tabela 4.2.2. Performanse sa fragmentacijom

	Korišćeno	Dostupno	Procenat iskorišćenosti
Broj slajs registara	15474	207360	7%
Broj slajs LUT-ova	58656	207360	28%
Broj potpuno iskorišćenih parova LUT-FF	13398	60732	22%
Broj pinova	842	960	87%
Broj blokova RAM/FIFO	46	324	14%
Broj globalnih taktova (BUFG/BUFGCTRL)	2	32	6%

Kao što se može primetiti, dizajn ima prihvatljive hardverske zahteve, naročito ako se u obzir uzme kompleksnost funkcije fragmentacije na prijemnom delu. Najveća promena je svakako broj blokova za RAM i broj logičkih blokova, što je logična posledica zbog kompleksnijeg prijemnika usled potrebe za čuvanjem fragmenata. Broj RAM blokova bi mogao da se smanji ukoliko bi memorija bila van ove arhitekture tj. ukoliko bi se koristila eksterna memorija. Broj iskorišćenih pinova jeste velik, ali zato što u obzir nisu uzeti niži slojevi OSI modela. U suštini, realizovani dizajn se spaja sa drugim modulima u FPGA čipu (npr. sa modulom koji radi primopredaju na drugom sloju OSI modela) i samim tim njegovi interfejsi ni ne treba da izlaze na pinove FPGA čipa.

5. ZAKLJUČAK

U ovom radu je realizovana dodatna funkcija fragmentacije IP paketa, čime je postignuta realnija situacija sigurnog IP tunela, gde se na sloju linka koristi Ethernet kao protokol. Sam dizajn koristi pajplajn tehniku za obradu fragmenata, čime se postigla mogućnost rada sa linkovima većeg protoka. Takođe, koristi se posebna tehnika fragmentiranja kojom se izbegava problem odbacivanja kod nekih rutera zbog DoS napada.

I dalje postoji odbrana od mogućih presretača na putu do odredišta šifrovanjem paketa. Rešenje je pogodno za manje mreže zbog kapaciteta memorije. Kako je dizajn skromnog hardverskog zahteva, moguće je povećati kapacitet memorije, ali treba pažljivo pratiti i povećanje logike i ako je potrebno uzeti čip većih mogućnosti. Bolje rešenje je izmestiti memoriju van dizajna i time dobiti veći kapacitet i rasterećenje čipa.

U budućim implementacijama, moguće je uključiti i dinamičko dodeljivanje id_tunela od neke baze podataka van ovog dizajna, za različite kombinacije izvorišne i odredišne adrese. Na ovaj način, prijemnik može utvrditi da li je došlo do greške u prenosu, čime bi imali dodatnu proveru zaglavlja. Ovo rešenje je moguće koristiti i za protokole koji nisu Ethernet uz minimalne modifikacije dizajna.

LITERATURA

[1] Aleksandar Dželatović, ,,Implementacija sigurnog IP tunela baziranog na AES enkripciji", master rad, Elektrotehnički fakultet, 2014.

[2] IP fragmentation: https://en.wikipedia.org/wiki/IP_fragmentation

[3] IP fragmentation attack: https://en.wikipedia.org/wiki/IP_fragmentation_attack

[4] IP Datagram Reassembly Algorithms: https://tools.ietf.org/html/rfc815

[5] AES standard: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[6] Specifikacije za čipove familije Virtex5: http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf