ELEKTROTEHNIČKI FAKULTET UNIVERZITETA U BEOGRADU

HARDVERSKA IMPLEMENTACIJA BASELINE KOMUTATORA

– Diplomski rad–

Kandidat:

Ivana Eraković 2009/0319

Mentor:

doc. dr Zoran Čiča

Beograd, Septembar 2015.

SADRŽAJ

1. UVOD

Razvoj integrisanih kola omogućio je razvoj hardverskih komponenti visokih performansi poput ASIC i FPGA čipova, koji omogućavaju efikasno procesiranje paketa na hardverskom nivou. Komutacija paketa ima prednost znatno boljeg iskorišćenja mrežnih resursa što je jedan od najznačajnijih parametara za operatera. *Baseline* komutator, koji pripada *Banyan* tipu komutatora, se koristi u mrežama baziranim ka komutaciji paketa pa su nam samim tim zanimljivi za implementaciju.

U ovom radu biće realizovan baseline komutator u okviru Xilinx-ovog ISE softverskog paketa, koristeći programski jezik VHDL. Realizacija će podržavati komutator sa dve, tri i četiri kaskade, kao i 16-bitnu, 32-bitnu i 64-bitnu magistralu podataka. Dizajn je parametrizovan pa se veoma jednostavno može podesiti željena dužina magistrale. Koristeći činjenicu da se *baseline* komutatori većih dimenzija kreiruju putem rekurzije na veoma jednostavan način smo realizovali komutatore koristeći već kreirane komutatore manjih dimenzija polazeći od najjednostavnijeg 2x2 komutatora.

U poglavlju 2 ovog rada biće objašnjena arhitektura, kao i način rutiranja paketa kod baseline komutatora. U poglavlju 3 detaljno je opisana sama implementacija algoritma u VDHL-u. U poglavlju 4 su prikazani primeri za različite dimenzije komutatora, kao i za različite dužine magistrala. Na kraju je dat zaključak u kome su izložena završna razmatranja teze.

2. OPIS BASELINE KOMUTATORA

2.1. Komutacija paketa

Tehnika komutacije paketa se zasniva na komunikaciji između korisnika koji razmenjuju podatke smeštene u pakete. Svaki paket se prenosi kroz mrežu prolazeći kroz mrežne čvorove. Kada uđe u mrežni čvor, određuje se na koji izlaz mrežnog čvora treba poslati paket. Paket zatim čeka na svoj redosled za slanje zajedno sa paketima drugih veza koji se šalju preko istog izlaza mrežnog čvora. Ako postoje paketi za slanje oni se šalju na izlaz tako da ako dođe do pauze u slanju paketa, prenosiće se paketi drugih veza tako da iskoršćenost linka ostaje ista kao da nije bilo pauze. Komutacija paketa postiže visoku iskorišćenost resursa u mreži, ali postoji problem sa garancijom kvaliteta. Zbog toga se u mrežama uvode brojne tehnike koje pokušavaju da reše problem kvaliteta servisa (npr. IntServ i DiffServ tehnika). U slučaju komutacije paketa proces uspostave veze nije neophodan, ali može da se izvrši [1].

Komutatori sa jednostrukim putanjama predstavljaju komutatore kod kojih postoji samo jedan put između ulaza *i* i izlaza *j*. Ako postoji više putanja, tada je u pitanju komutator sa višestrukim putanjama.

2.2. Banyan komutatori

Banyan komutatori su dobili naziv po nazivu indijskog drveta koje ima sličnu strukturu grananja. Koriste se kao električni ili optički prekidači za promenu putanje podataka u mreži.

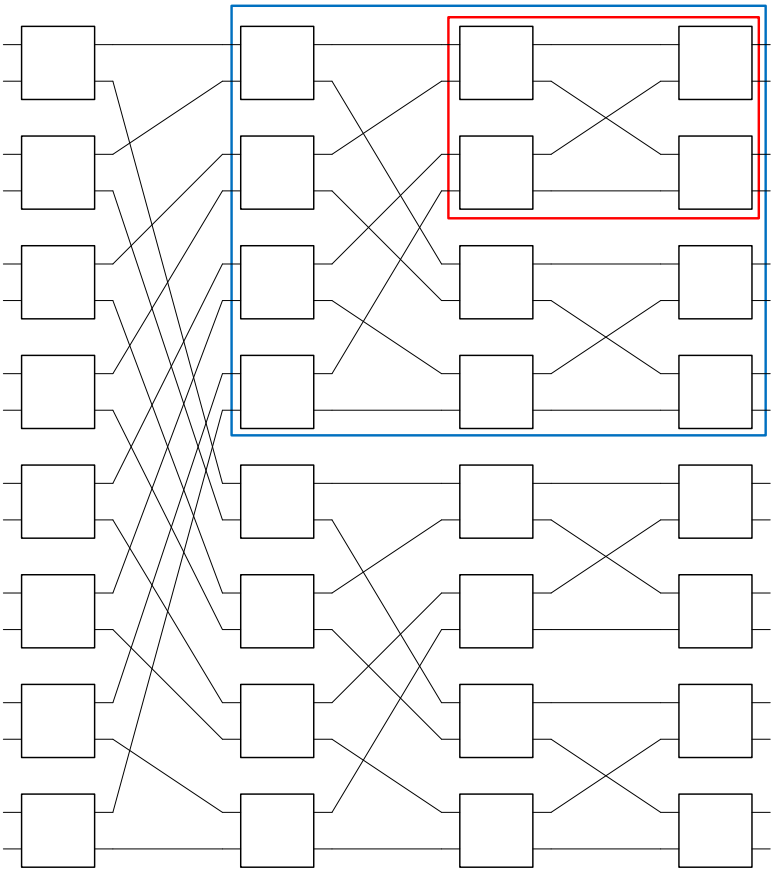
Usled problema realizacije velikih komutacionih polja, zbog potrebe za velikim brojem prekidača, pomoću krosbar komutatora, iako su bili jednostavni i imali osobinu neblokiranja, došlo je do potrebe za optimalnijom strukturom. Rešenje se javilo korišćenjem višekaskadnih struktura koje su se javile kod analognih komutacionih polja poput Banyan i Klosovih struktura.

Sve Banyan strukture su *N*x*N* dimenzija, sačinjene su od manjih elemenata *n*x*n* dimenzija i imaju sledeće osobine:

- Postoji tačno jedan put od bilo kog ulaza do bilo kog izlaza.
- Banyan strukture imaju osobinu samo-prosleđivanja i to je veoma pogodno za prosleđivanje paketa. Funkcije prosleđivanja su minimalne pa se mogu ostvariti velike brzine implementiranjem ovih funkcija u hardver komutacionog elementa.
- Jedna od bitnih osobina je skalabilnost, što znači da je mreža lako proširiva korišćenjem manjih konstrukcija kako bi se dobile veće.
- Banyan strukture su takođe i blokirajuće, pošto može da dođe do unutrašnje blokade usled činjenice da dva različuta paketa pokušaju da prođu kroz isti interni link u istom trenutku. Time se prouzrokuje degradacija propusnosti i ona raste s brojem portova u mreži.

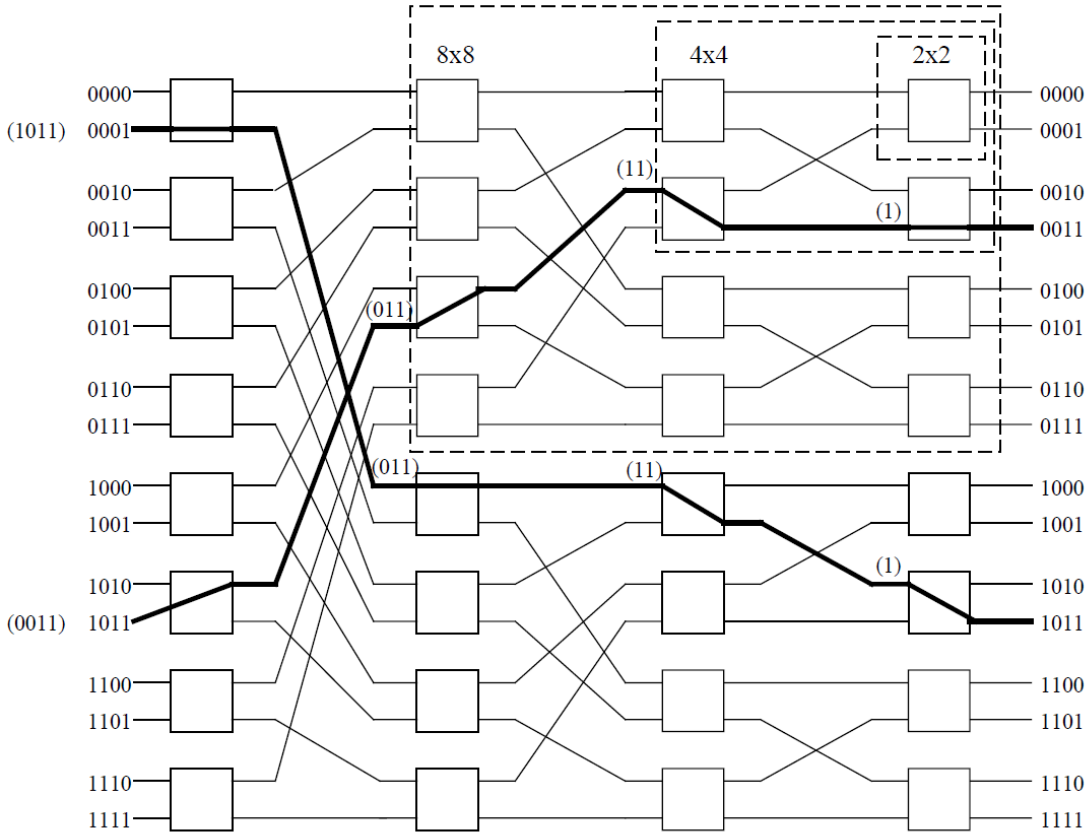
2.3. Baseline komutatori

Baseline struktura je prikazana na slici 2.3.1. Ova struktura se kreira primenom rekurzije. Na slici 2.3.1 su uokvirene *baseline* strukture za 4x4 (crvenom bojom) i 8x8 (plavom bojom) dimenzije. Struktura *N*x*N* se kreira upotrebom *n*x*n* (*n*=*N*/2) i osnovne 2x2 strukture na sledeći način. Postavlja se *n*x*n* struktura jedna iznad druge. Prva kaskada je sačinjena od 2x2 strukrura kojih ima *n* i povezuju se sa *n*x*n* strukturama na sledeći način. Vezuju se prvi izlazi svih 2x2 struktura redom na ulaze prve *n*x*n* strukrture, dok druge izlaze svih 2x2 struktura vezujemo redom na ulaze druge *n*x*n* strukture. Kao što je prikazano na slici 2.3.1, *baseline* 16x16 komutator je kreiran principom rekurzije i sačinjen je od osam 2x2 komutatora i dva 8x8 komutatora.



Slika 2.3.1. *Baseline* 16x16 komutator [1]

Redosled upotrebe bita odredišne adrese prilikom prolaska kroz svičeve u procesu samorutiranja kod *baseline* komutatora je redom kako su i navedeni u odredišnoj adresi (od najvišeg do najnižeg bita). Primer samorutiranja je prikazan na slici 2.3.2



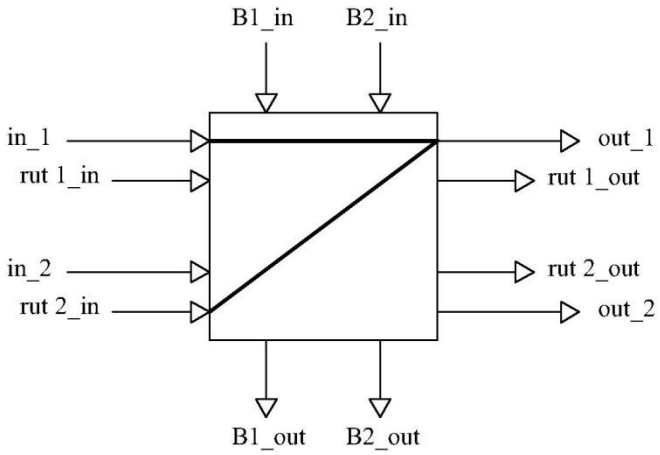
Slika 2.3.2. Proces samorutiranja kod *baseline* 16x16 komutatora [3]

Usled samorutiranja može doći do kolizije paketa. Tada se željenom putanjom prosleđuje paket višeg prioriteta, a drugi se šalje pogrešnom putanjom. Razlog za to je nemogućnost osnovne jedinice 2x2 da prosledi oba paketa istovremeno na željeni izlazni link gde se desila kolizija.

3. OPIS IMPLEMENTACIJE

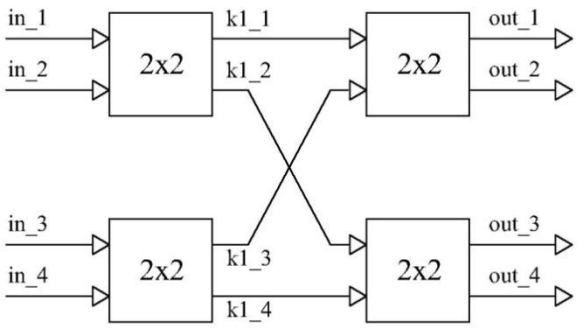
Za implementaciju je korišćen programski jezik VHDL i Xilinx-ov ISim simulator. Cela ideja hardverske implementacije je zamišljena da se polazi od osnovnog 2x2 komutatora.

Prvi entitet je samim tim *2x2.vhd*. Ona predstavlja jedinični komutator sa dva ulaza i dva izlaza. Pored paketa koji se prenose (*in1*, *in2*, *out1*, *out2*) i signala za rutiranje (*rut1_in*, *rut2_in*, *rut1_out*, *rut2_out*) koristimo i kontrolne bite (*B1_in*, *B2_in*, *B1_out*, *B2_out*). Kontrolni biti se koriste za obeležavanje paketa koji su doživeli koliziju i našli se na pogrešnoj ruti kroz komutator. U slučaju da se ti paketi ponovo nađu u koliziji automatski će gubiti prednost i biće ponovo prosleđeni na pogrešnu rutu.



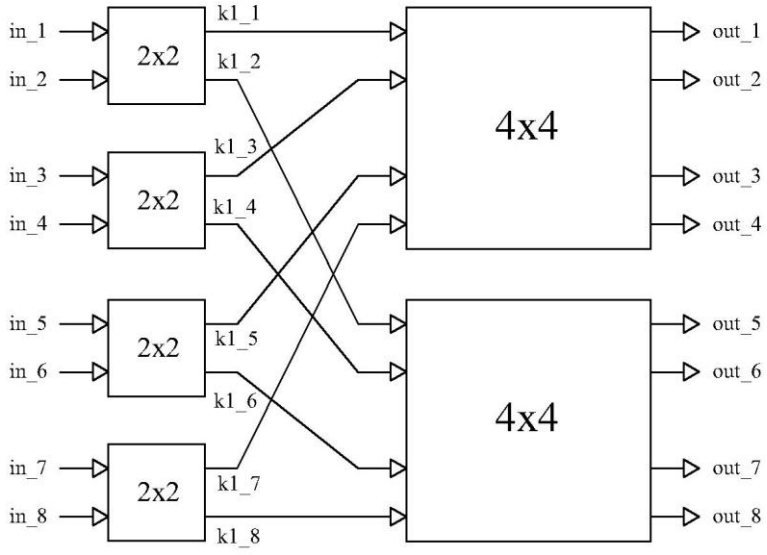
Slika 3.1. Kolizija u *baseline* 2x2 komutatoru

Da bismo mogli da koristimo ovaj entitet za realizaciju drugih komutatora uvode se parametar dužine magistrale (*N*), ukupan broj kaskada u komutatoru (*m*) i kaskada koju trenutno koristimo (*kaskada*). Logika 2x2 entiteta se zasniva na tome da se uporede ulazni kontrolni biti *B1_in* i *B2_in* da bi se videlo koji paket ima prednost. Samo u slučaju kada su *B1_in*=1, a *B2 _in*=0 paket na ulazu 2 ima prednost, u svim ostalim slučajevima prednost ima paket na ulazu1. Nakon toga se vrši analiza rute i to pomoću bita *rut1(m-kaskada)* ulaznog signala *rut1_in*. U slučaju da je taj bit 0 paket ulaza 1 se šalje na prvi izlaz, u slučaju da je 1 paket ulaza 1 se šalje na drugi izlaz. Pored toga proverava se da slučajno *rut1(m-kaskada*) nije jednako sa *rut2(m-kaskada)*. Ako se desi da su jednaki to znači da je došlo do kolizije (slika 3.1), tj. oba paketa žele da budu prosleđena na isti izlaz. Kao što je već ranije objašnjeno zna se koji paket ima prednost, a izlaznom kontrolnom bitu paketa koji nema prednost dodeljuje se vrednost 1.



Slika 3.2. *Baseline* 4x4 komutator sačinjen od četiri *baseline* 2x2 komutatora

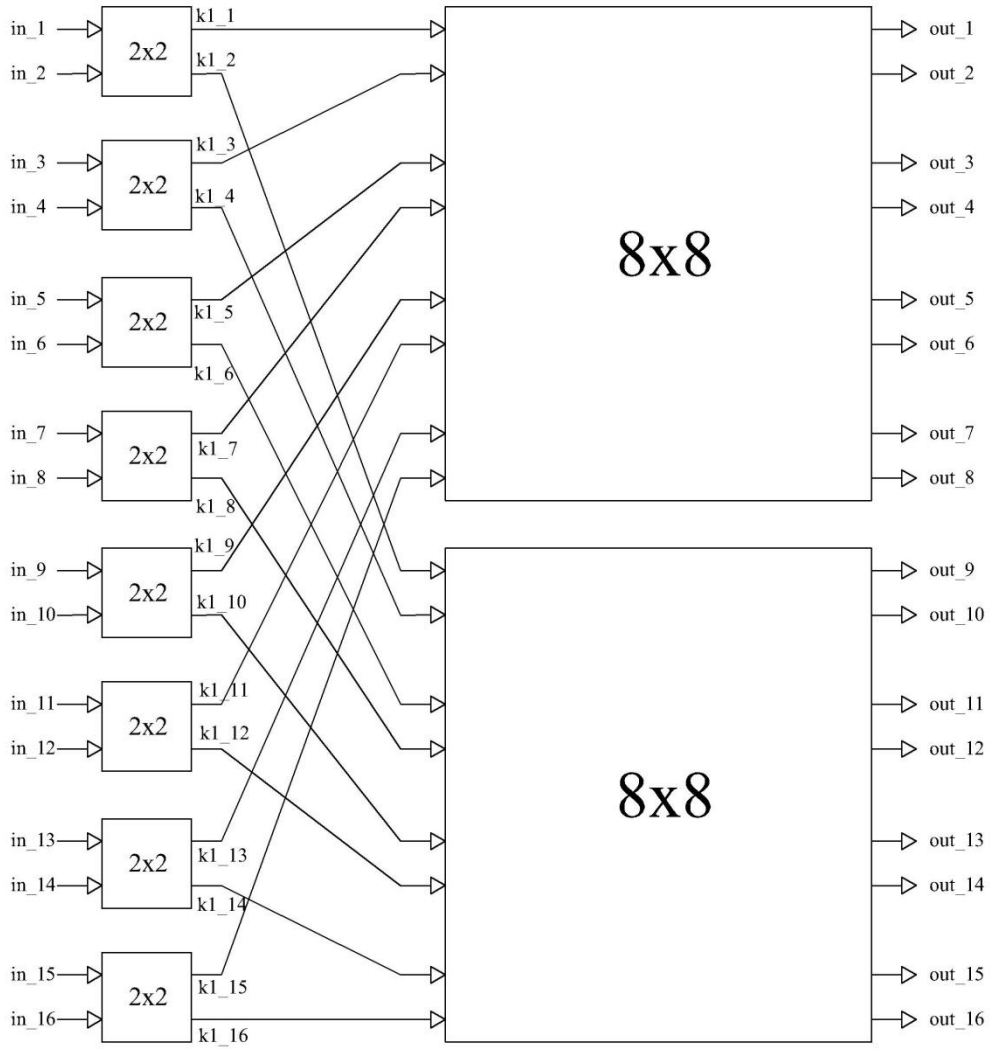
Sledeći entitet je *4x4.vhd* i predstavlja *baseline* komutator sa četiri ulaza, četiri izlaza i dve kaskade. Sačinjena je od četiri instance *2x2.vhd* komponente koje su povezane na način prikazan na slici 3.2. Prve dve instance koristimo za prvu kaskadu. Mapirali smo ulazne signale, ulazne rute i ulazne kontrolne bite prve instance na *in_1*, *in_2*, *ru1_in*, *rut2_in*, *B1_in* i *B1_2* koji pripadaju 4x4 entitetu. Za mapiranje izlaznih signala instance koristimo lokalne signale entiteta 4x4 tj. *out1* i *out2* instance mapiramo na lokalne signale *k1_1* i *1_2*, zatim *B1_out* i *B2_ot* na *B1_loc* i *B2_loc* i *rut1_out* i *rut2_out* na *rut1_loc* i *rut2_loc*. Portove druge instance mapiramo na isti način kao i prvu samo što *in_1* instance mapiramo na *in_3* funkcije 4x4, a *in_2* na *in_4*, i tako za svaki ulazni i izlazni signal instance. Sada za drugu kaskadu koristimo treću i četvrtu instancu, samo što ovog puta ulazne portove povezujemo na lokalne signale dobijene na izlazima iz prve kaskade i to po pravilu povezivanja koje važi za *baseline* komutatore (treća instanca: *in_1=>k1_1*, *in_2=>k1_3*, četvrta instanca: *in_1=>k1_2*, *in_1=>k1_4*, isto pravilo važi i za kontrolne i rutirajuće signale). I na kraju izlazni signali iz treće i četvrte instance su izlazni signali celog 4x4 komutatora.



Slika 3.3. *Baseline* 8x8 komutator sačinjen od četiri *baseline* 2x2 komutatora i dva 4x4 komutatora

Treći entitet je *8x8.vhd* i predstavlja *baseline* komutator sa osam ulaza, osam izlaza i tri kaskade. Pošto smo prethodno formirali i 4x4 komutator sada možemo da ga koristimo za formiranje 8x8 komutatora pošto su *baseline* komutatori skalabilni. Tako ćemo u ovom slučaju imati u prvoj kaskadi četiri instance *2x2.vhd* komponente i dve instance *4x4.vhd* komponente, kao što je prikazano na slici 3.3.

Kao što možemo videti na slici 3.3 ulazne signale prve četiri instance *2x2.vhd* komponente mapiramo redom na osam ulaznih signala *8x8.vhd* entiteta. Ponovo dobijamo lokalne signale kao izlaze iz prve kaskade koje povezujemo sa ulazima 4x4 instanci. Izlazni signali iz obe 4x4 instance predstavljaju izlazne signale iz celog *baseline* 8x8 komutatora.



Slika 3.4. *Baseline* 16x16 komutator sačinjen od osam *baseline* 2x2 komutatora i dva 8x8 komutatora

Isti princip realizacije važi i za baseline 16x16 komutator samo što će on biti sačinjen od osam komutatora 2x2 tipa u prvoj kaskadi i 2 komutatora 8x8 tipa (slika 3.4).

Pored gore navedenih *baseline* komutatora, korišćenjem rekurzije na veoma jednostavan način možemo kreirati i *baseline* komutatore većih dimenzija. Svaki *n*x*n* komutator biće sačinjen od *n*/2 komutatora dimenzije 2x2 i od dva komutatora dimenzije (*n*/2)x(*n*/2).

Svaki od navedenih entiteta pored svih navedenih ulaznih i izlaznih signala koristi i signale *clk* i *reset*. Signal *clk* je signal takta i prolazak paketa kroz jednu kaskadu traje jedan takt. Signal *reset* služi da se ponovno pokrenje slanja paketa, a dok ne krene slanje paketa svi izlazni signali su fiksirani na 0. *Reset* je aktivan ako ima vrednost "1".

4. SIMULACIJA I PERFORMANSE

4.1. Opis performansi

Izvršavanjem procesa analize i sinteze svih pet varijanti dizajna *baseline* komutatora, dobijene su informacije o performansama svake od implementacija. Proces analize i sinteze izvršen je u ISE razvojnom okruženju za FPGA čipove proizvođača Xilinx. U pitanju je dobra procena vrednosti i one su date u tabeli 4.1.1. U zavisnosti od dizajna komutatora bio je potreban različiti broj pinova, pa je za 4x4, 64-bitni komutator i 16x16, 16-bitni komutator izabran uređaj XC5VLX220 familije Virtex5, dok je za 4x4, 16-bitni i 32-bitni komutator, kao i za 8x8x, 16-bitni komutator izabran uređaj XC5VLX85 isto familije Virtex5.

NAZIV	4X4 16-BITNA	4X4 32-BITNA	4X4 64-BITNA	8X8 16-BITNA	16X16 16-BITNA
Broj slajs registara	152/51840 (0%)	280/51840 (0%)	536/138240 (0%)	480/51840 (0%)	1344/138240 (0%)
Broj slajs LUT-ova	161/51840 (0%)	289/51840 (0%)	545/138240 (0%)	505/51840 (0%)	1473/138240 (1%)
Broj potpuno iskorišćenih parova LUT-FF	140/173 (80%)	268/301 (89%)	521/557 (94%)	456/529 (86%)	1296/1521 (85%)
Broj globalnih taktova (BUFG/BUFGCTRL)	1/32 (3%)	1/32 (3%)	1/32 (3%)	1/32 (3%)	1/32 (3%)
Broj pinova	158/440 (35%)	286/440 (65%)	542/800 (67%)	328/440 (74%)	686/800 (85%)
Maksimalna frekvencija	602.520MHz	600.696MHz	597.081MHz	564.012MHz	463.408MHz

Tabela 4.1.1 Procenjene performanse za različite komutatore

Prema podacima iz Tabele 4.1.1 može se primetiti da se broj pinova menja za komutator sa istim brojem ulaza kome menjamo parametar dužine magistrale. Na primer, za 4x4 komutator vidimo da je broj pinova za 16-bitnu magistralu 158, za 32-bitnu je 286, dok je za 64-bitnu 542, što je posledica veće širine magistrala podataka na ulazima i izlazima komutatora. Takođe možemo primetiti da se broj pinova značajno menja sa kompleksnijim dizanom pa je broj pinova za 16x16 komutator sa 16-bitnom magistralom čak 686.

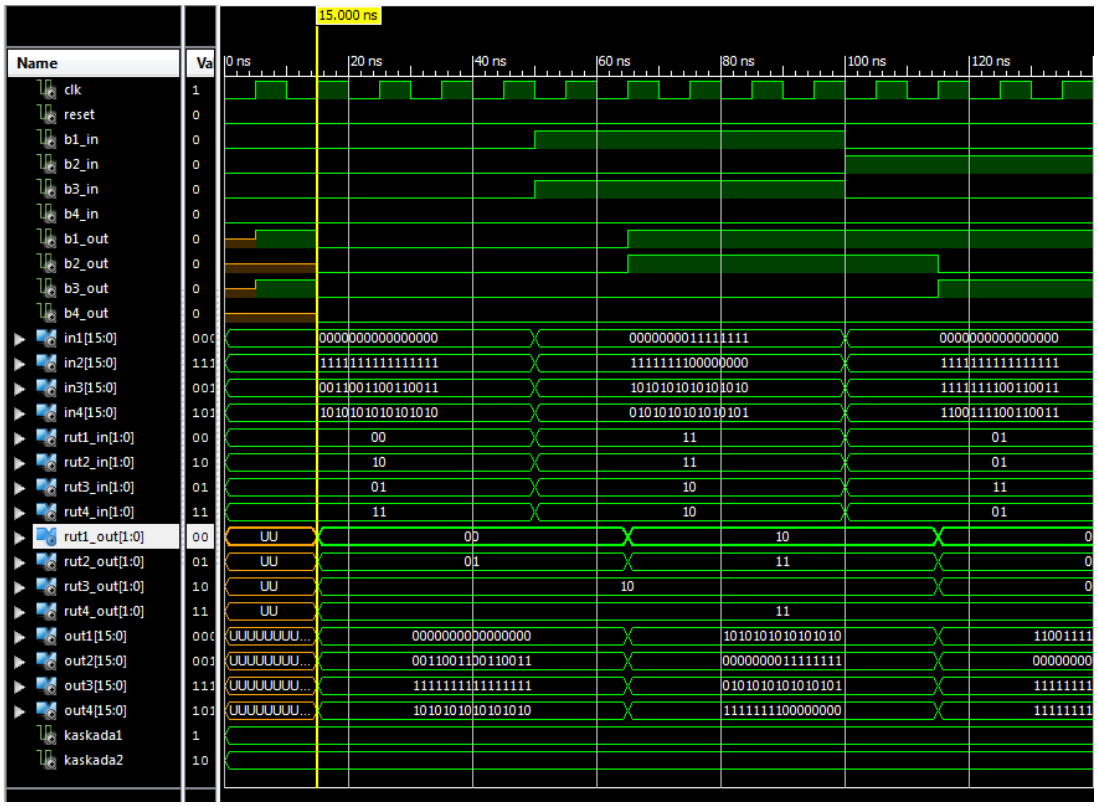
Maksimalna frekvencija se smanjuje sa povećanjem dužine magistrale, kao i sa povećanjem kompleksnosti dizajna, ali ostvarene maksimalne frekvencije potvrđuju da je implementirani dizajn

veoma efikasan. Takođe, iz Tabele 4.1.1 se može primetiti da su zauzeti minimalni hardverski resursi čipa što dodatno svedoči o efikasnosti realizovane implementacije.

4.2. Primeri simulacije

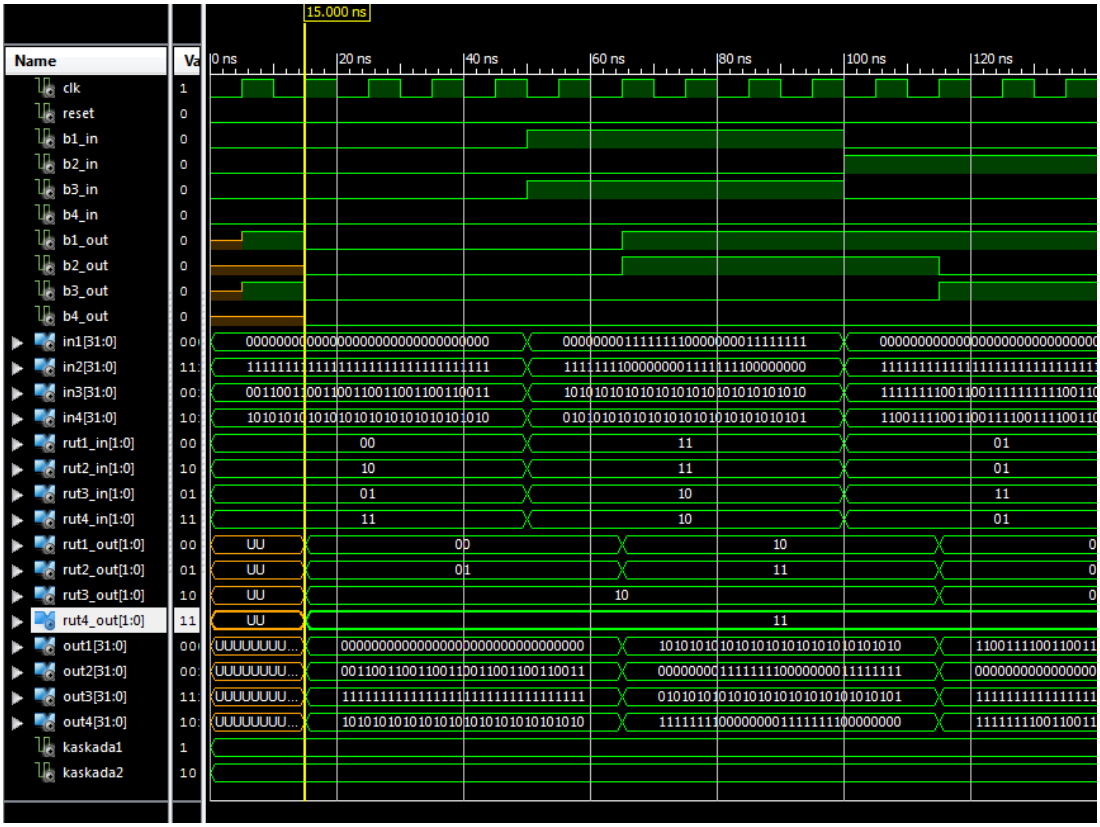
Kao primere koristićemo 4x4, 8x8 i 16x16 *baseline* komutatore i menjaćemo dužine magistrale (16-bitna, 32-bitna i 64-bitna).

Prvi primer: Prikaz rada *baseline* 4x4 komutatora sa 16-bitnom, 32-bitnom i 64-bitnom magistralom.

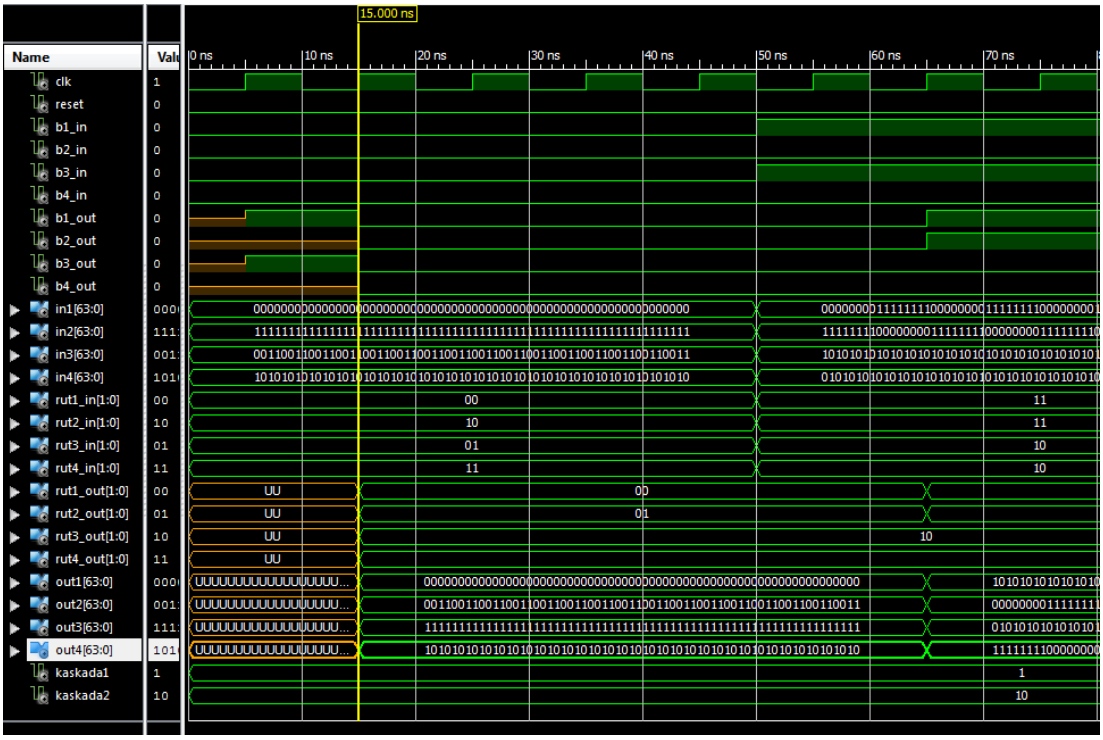


Slika 4.2.1. Simulacija *baseline* 4x4 komutator sa 16-bitnom magistralom

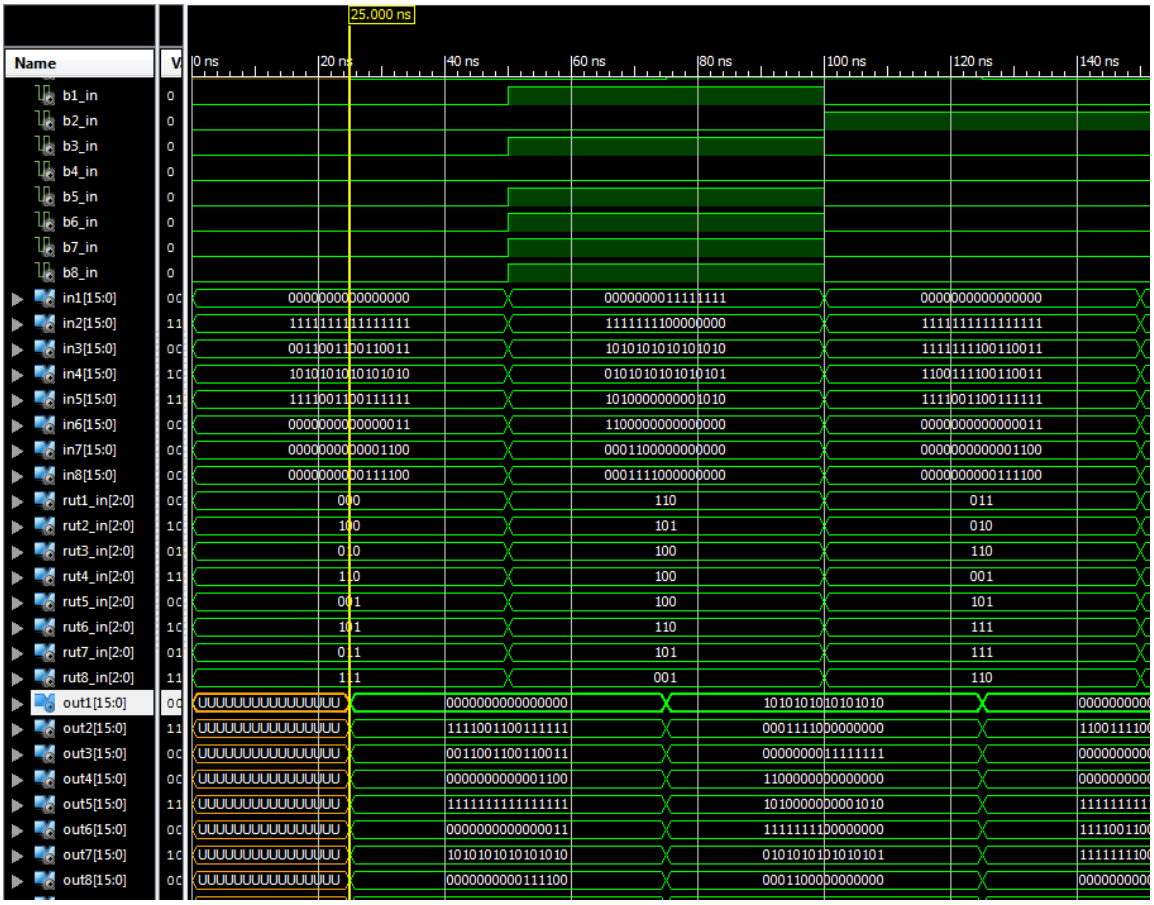
Sa slike 4.2.1 vidimo da na izlazu iz druge kaskade imamo kašnjenje od 10ns. Takođe vidimo da pri slanju paketa poslatih u 0ns ne dolazi do kolizije, tj. svaki od paketa je prosleđen na željenu rutu. Što se tiče paketa poslatih u 50ns vidimo da su u startu kontrolni biti *B1_in* i *B3_in* postavljeni na vrednost "1" pa automatski ulazni signali *in1* i *in3* nemaju prednost. Kako signali *rut1_in* i *rut2_in* imaju istu vrednost, a *in1* nema prednost signal *in2* se šalje na željeni izlaz. Isto važi i za signale *in3* i *in4*, gde *in4* ima prednost. Kada signali dođu do druge kaskade do kolizije dolazi između signala *in2* i *in4*, koji imaju isti kontrolni bit "0", tako da prednost ima signal na nižem ulazu, tj. *in2*. Sa slike 4.2.1 možemo videti da jedino *in2* ostaje sa kontrolnim bitom "0" na izlazu i rutiran je na željeni izlaz *out4*.



Slika 4.2.2. Simulacija *baseline* 4x4 komutator sa 32-bitnom magistralom



Slika 4.2.3. Simulacija *baseline* 4x4 komutator sa 64-bitnom magistralom



Slika 4.2.4. Simulacija *baseline* 8x8 komutator sa 16-bitnom magistralom

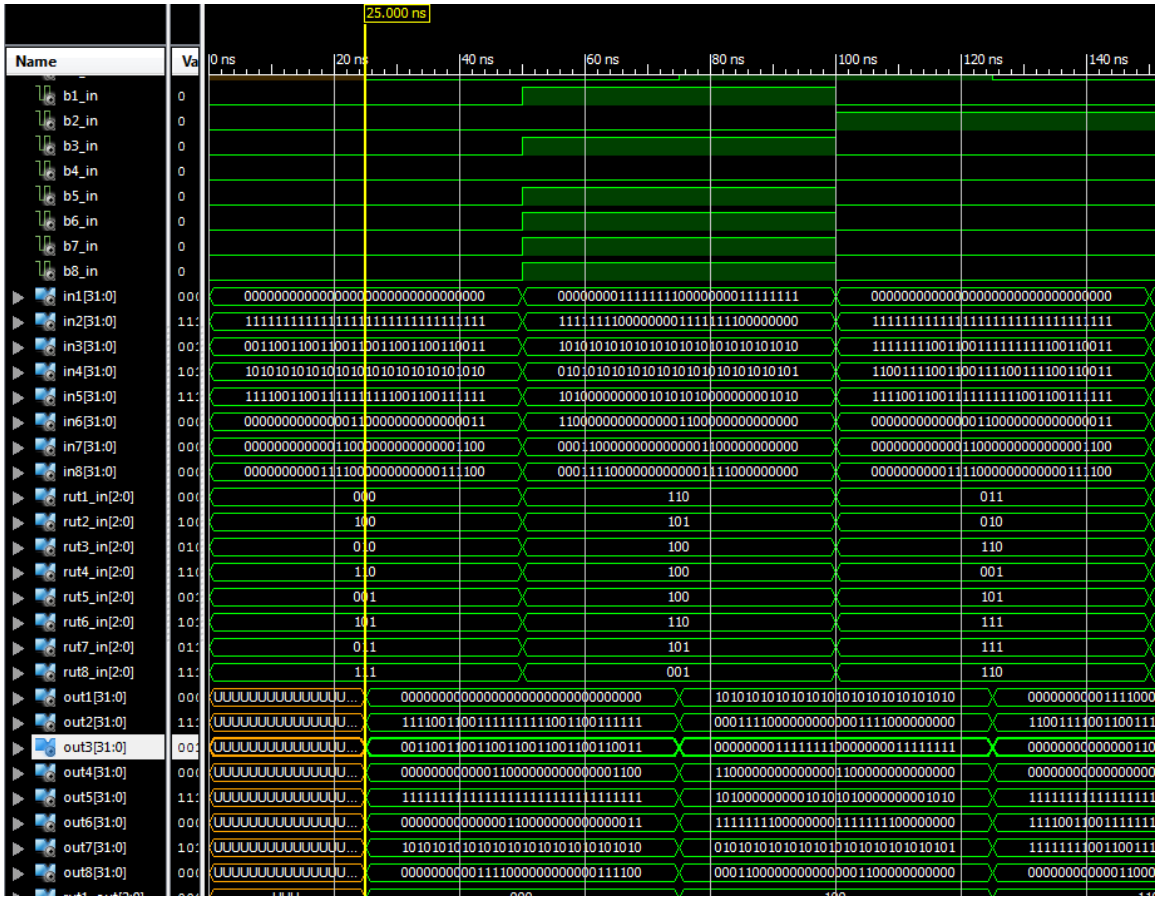
Ako posmatramo slike 4.2.2 i 4.2.3 gde smo povećali dužine magistrala vidimo da nema promena u slanju paketa, tj. da dužina magistrale ne utiče na rutiranje podataka, niti izaziva dodatno kašnjenje.

Drugi primer: Prikaz rada *baseline* 8x8 komutatora sa 16-bitnom i 32-bitnom magistralom.

Kada posmatramo sliku 4.2.4 tj. simulaciju *baseline* 8x8 komutatora vidimo da je kašnjenje na izlazu 20ns, što je više nego kod 4x4 komutatora. Razlog tome je prolazak kroz jednu kaskadu više, tako da možemo da pretpostavimo da će kašnjenje na izlazu ostati isto u slučaju da promenimo dužinu magistrale, što možemo videti na slici 4.2.5.

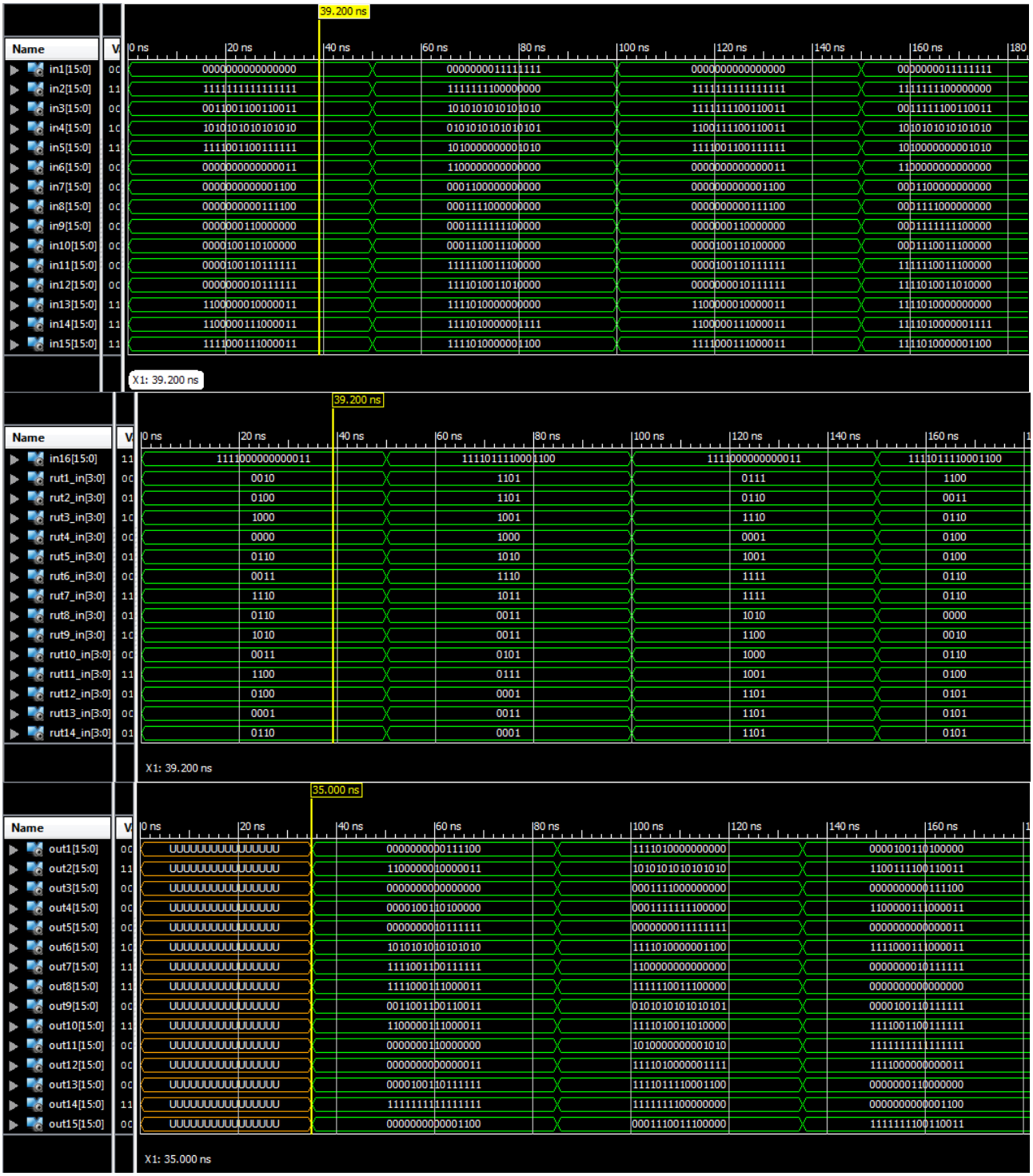
Treći primer: Prikaz rada *baseline* 16x16 komutatora sa 16-bitnom magistralom.

Sada posmatramo sliku 4.2.6 koja prikazuje simulaciju *baseline* 16x16 komutatora. Opet možemo primetiti da je jedina razlika u odnosu na komutatore sa manje ulaza zapravo kašnjenje na izlazu koje ovde iznosi 30ns. Možemo primetiti da svaka dodatna kaskada unosi dodatno kašnjenje od 10ns, što je zapravo jedan takt kašnjenja po kaskadi. Logika prosleđivanja je ostala ista. Vidimo da je signal *in1* prosleđen na *out3*, kako mu je rutirajući signal *rut1_in*="0010" vidimo da je logika ista kao i kod komutatora sa manje ulaza.



Slika 4.2.5. Simulacija *baseline* 8x8 komutator sa 32-bitnom magistralom

Kroz prethodna tri primera smo proverili da li različite dimenzije dizajna rade za različite vrednost parametara dužine magistrale. Pored toga ovim primerima su obuhvaćene situacije kada ne dolazi do kolizije kao i one kada se kolizija javlja. S obzirom da dizajn radi ispravno u svim ovim situacijama možemo smatrati da je dizajn uspešno verifikovan.



Slika 4.2.6. Simulacija *baseline* 16x16 komutator sa 16-bitnom magistralom

5. ZAKLJUČAK

U ovom radu je prikazano kako funkcioniše *baseline* komutator i način na koji prosleđuje pakete od ulaza do izlaza. Za implementaciju je korišćen Xilinx-ov ISE softverski paket, a kod je pisan u programskom jeziku VHDL.

Da bi se ostvario modularan dizajn svaka veće jedinica je konstruisana od više manjih, kao npr. 4x4 komutator sačinjen od četiri 2x2 komutatora. Na taj način smo postigli da svaki veći komutator konstruišemo od već napravljenih manjih. Time je postignuto da se na jednostavan način mogu kreirati komutatori većih dimenzija. Logika odlučivanja za prosleđivanje paketa je samo u jediničnom 2x2 komutatoru, tako da nam i to značajno olakšava konstruisanje većih komutatora. Ovde su prikazani 4x4, 8x8 i 16x16 komutatori, ali isto tako jako jednostavno od njih možemo napraviti 32x32 komutator.

Moramo naglasiti i da je logika prosleđivanja paketa kreirana u okviru ove teze veoma jednostavna što doprinosi visokim performansama realizovanog dizajna. Prosleđivanje je regulisano kontrolnim bitom i prednost su imali paketi sa nižim ulaznim indeksom. Sama logika je podložna promenama, tako da možemo reći da je ovo samo početna varijanta hardverske implementacije *baseline* komutatora. Dizajn je portabilan, odnosno može se bez ikakvih izmena kompajlirati i za druge FPGA čipove Xilinx proizvođača, ali i druge FPGA čipove drugih proizvođača.

LITERATURA

[1] Čiča Zoran, Materijali sa predmeta Komutacioni sistemi

[2] Čiča Zoran, Materijali sa predmeta Programiranje komunikacionog hardvera

[3] Zoran Petrović, Materijali sa predmeta Integrisane telekomunikacione mreže

A. KOD REALIZOVANIH ENTITETA U VHDL-U

A.1. Entitet *baseline* 2x2 komutatora

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
ENTITY Baseline2x2 IS
GENERIC
(
N: INTEGER:=16;
m: INTEGER:=1
);
PORT
(
clk,reset: IN std_logic;
B1_in, B2_in: IN std_logic;
B1_out, B2_out: OUT std_logic;
rut1_in, rut2_in: IN std_logic_vector(m-1 DOWNTO 0);
rut1_out, rut2_out: OUT std_logic_vector(m-1 DOWNTO 0);
in1, in2: IN std_logic_vector(N-1 DOWNTO 0);
out1, out2: OUT std_logic_vector(N-1 DOWNTO 0);
kaskada: IN integer:=1
);
END Baseline2x2;
ARCHITECTURE sema OF Baseline2x2 IS
BEGIN
PROCESS (clk)
BEGIN
IF(reset='1')THEN
 out1<=(others=>'0');
 out2<=(others=>'0');
 B1_out<='0';
 B2_out<='0';
ELSIF(clk'EVENT AND clk='1')THEN
 IF (B1_in='1' and B2_in='0')THEN
       IF (rut1_in(m-kaskada)='0' and rut2_in(m-kaskada)='0') THEN
             out1<=in2;
             out2<=in1;
```

```
                B1_out<=B2_in;
                B2_out<=B1_in;
                rut1_out<=rut2_in;
                rut2_out<=rut1_in;
            ELSIF (rut1_in(m-kaskada)='0' and rut2_in(m-kaskada)='1') THEN
                out1<=in1;
                out2<=in2;
                B1_out<=B1_in;
                B2_out<=B2_in;
                rut1_out<=rut1_in;
                rut2_out<=rut2_in;
            ELSIF (rut1_in(m-kaskada)='1' and rut2_in(m-kaskada)='0') THEN
                out1<=in2;
                out2<=in1;
                B1_out<=B2_in;
                B2_out<=B1_in;
                rut1_out<=rut2_in;
                rut2_out<=rut1_in;
            ELSE
                out1<=in1;
                out2<=in2;
                B1_out<=B1_in;
                B2_out<=B2_in;
                rut1_out<=rut1_in;
                rut2_out<=rut2_in;
            END IF;
ELSE
IF (rut1_in(m-kaskada)='0' and rut2_in(m-kaskada)='0') THEN
        out1<=in1;
        out2<=in2;
        B1_out<=B1_in;
        B2_out<='1';
        rut1_out<=rut1_in;
        rut2_out<=rut2_in;
            ELSIF (rut1_in(m-kaskada)='0' and rut2_in(m-kaskada)='1') THEN
                out1<=in1;
                out2<=in2;
                B1_out<=B1_in;
                B2_out<=B2_in;
                rut1_out<=rut1_in;
                rut2_out<=rut2_in;
            ELSIF (rut1_in(m-kaskada)='1' and rut2_in(m-kaskada)='0') THEN
                out1<=in2;
                out2<=in1;
                B1_out<=B2_in;
```

```
                B2_out<=B1_in;
                rut1_out<=rut2_in;
                rut2_out<=rut1_in;
        ELSE
                out1<=in2;
                out2<=in1;
                B1_out<='1';
                B2_out<=B1_in;
                rut1_out<=rut2_in;
                rut2_out<=rut1_in;
        END IF;
END IF;
END IF;
END PROCESS;
END sema;
```

A.2. Entitet *baseline* 4x4 komutatora

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
Library XilinxCoreLib;
ENTITY Baseline4x4 IS
GENERIC
(
N: INTEGER:=16;
m: INTEGER:=2
);
PORT
(
clk,reset: IN std_logic;
in1, in2, in3, in4: IN std_logic_vector(N-1 DOWNTO 0);
B1_in, B2_in, B3_in, B4_in: IN std_logic;
B1_out, B2_out, B3_out, B4_out: OUT std_logic;
rut1_in, rut2_in, rut3_in, rut4_in: IN std_logic_vector(m-1 DOWNTO 0);
rut1_out, rut2_out, rut3_out, rut4_out: OUT std_logic_vector(m-1 DOWNTO 0);
out1, out2, out3, out4: OUT std_logic_vector(N-1 DOWNTO 0);
kaskada1:IN INTEGER:=1;
kaskada2:IN INTEGER:=2
);
END Baseline4x4;
```

```
ARCHITECTURE sema OF Baseline4x4 IS
COMPONENT Baseline2x2 IS
	GENERIC
	(
	N: INTEGER:=16;
	m: INTEGER:=1
	);
	PORT(
	clk,reset: IN std_logic;
	B1_in, B2_in: IN std_logic;
	B1_out, B2_out: OUT std_logic;
	rut1_in, rut2_in: IN std_logic_vector(m-1 DOWNTO 0);
	rut1_out, rut2_out: OUT std_logic_vector(m-1 DOWNTO 0);
	in1, in2: IN std_logic_vector(N-1 DOWNTO 0);
	out1, out2: OUT std_logic_vector(N-1 DOWNTO 0);
	kaskada: IN integer:=1
	);
END COMPONENT;

SIGNAL k1_1:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL k1_2:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL k1_3:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL k1_4:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL rut1_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL rut2_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL rut3_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL rut4_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL B1_loc:STD_LOGIC;
SIGNAL B2_loc:STD_LOGIC;
SIGNAL B3_loc:STD_LOGIC;
SIGNAL B4_loc:STD_LOGIC;

BEGIN

Baseline2x2_inst1: Baseline2x2
	GENERIC MAP
	(
		N=>N,
```

```
		m=>m
	)
	PORT MAP
	(
	clk => clk,
	reset => reset,
	kaskada=>kaskada1,
	B1_in=>B1_in,
	B2_in=>B2_in,
	B1_out=>B1_loc,
	B2_out=>B2_loc,
	in1=>in1,
	in2=>in2,
	rut1_in=>rut1_in,
	rut2_in=>rut2_in,
	rut1_out=>rut1_loc,
	rut2_out=>rut2_loc,
	out1=>k1_1,
	out2=>k1_2
	);

Baseline2x2_inst2: Baseline2x2
	GENERIC MAP
	(
		N=>N,
		m=>m
	)
	PORT MAP
	(
	clk => clk,
	reset => reset,
	kaskada=>kaskada1,
	B1_in=>B3_in,
	B2_in=>B4_in,
	B1_out=>B3_loc,
	B2_out=>B4_loc,
	in1=>in3,
	in2=>in4,
	rut1_in=>rut3_in,
```

```
        rut2_in=>rut4_in,
        rut1_out=>rut3_loc,
        rut2_out=>rut4_loc,
        out1=>k1_3,
        out2=>k1_4
        );

Baseline2x2_inst3: Baseline2x2
        GENERIC MAP
        (
                N=>N,
                m=>m
        )
        PORT MAP
        (
        clk => clk,
        reset => reset,
        kaskada=>kaskada2,
        B1_in=>B1_loc,
        B2_in=>B3_loc,
        B1_out=>B1_out,
        B2_out=>B2_out,
        in1=>k1_1,
        in2=>k1_3,
        rut1_in=>rut1_loc,
        rut2_in=>rut3_loc,
        rut1_out=>rut1_out,
        rut2_out=>rut2_out,
        out1=>out1,
        out2=>out2
        );

Baseline2x2_inst4: Baseline2x2
        GENERIC MAP
        (
                N=>N,
                m=>m
        )
        PORT MAP
```

```
        (
        clk => clk,
        reset => reset,
        kaskada=>kaskada2,
        B1_in=>B2_loc,
        B2_in=>B4_loc,
        B1_out=>B3_out,
        B2_out=>B4_out,
        in1=>k1_2,
        in2=>k1_4,
        rut1_in=>rut2_loc,
        rut2_in=>rut4_loc,
        rut1_out=>rut3_out,
        rut2_out=>rut4_out,
        out1=>out3,
        out2=>out4
        );
END sema;
```

A.3. Entitet *baseline* 8x8 komutatora

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
Library XilinxCoreLib;
ENTITY Baseline8x8 IS
GENERIC
(
N: INTEGER:=16;
m: INTEGER:=3
);
PORT
(
clk,reset: IN std_logic;
in1, in2, in3, in4, in5, in6, in7, in8: IN std_logic_vector(N-1 DOWNTO 0);
B1_in, B2_in, B3_in, B4_in, B5_in, B6_in, B7_in, B8_in: IN std_logic;
B1_out, B2_out, B3_out, B4_out, B5_out, B6_out, B7_out, B8_out: OUT std_logic;
rut1_in, rut2_in, rut3_in, rut4_in, rut5_in, rut6_in, rut7_in, rut8_in: IN
std_logic_vector(m-1 DOWNTO 0);
rut1_out, rut2_out, rut3_out, rut4_out, rut5_out, rut6_out, rut7_out, rut8_out:
OUT std_logic_vector(m-1 DOWNTO 0);
out1, out2, out3, out4, out5, out6, out7, out8: OUT std_logic_vector(N-1 DOWNTO
0);
```

```
kaskada1:IN INTEGER:=1;
kaskada2:IN INTEGER:=2;
kaskada3:IN INTEGER:=3
);
END Baseline8x8;
ARCHITECTURE sema OF Baseline8x8 IS
	COMPONENT Baseline2x2 IS
	GENERIC
	(
	N: INTEGER:=16;
	m: INTEGER:=1
	);
	PORT(
	clk,reset: IN std_logic;
	B1_in, B2_in: IN std_logic;
	B1_out, B2_out: OUT std_logic;
	rut1_in, rut2_in: IN std_logic_vector(m-1 DOWNTO 0);
	rut1_out, rut2_out: OUT std_logic_vector(m-1 DOWNTO 0);
	in1, in2: IN std_logic_vector(N-1 DOWNTO 0);
	out1, out2: OUT std_logic_vector(N-1 DOWNTO 0);
	kaskada: IN integer:=1
	);
	END COMPONENT;

	COMPONENT Baseline4x4 IS
	GENERIC
	(
	N: INTEGER:=16;
	m: INTEGER:=2
	);
	PORT(
	clk,reset: IN std_logic;
	in1, in2, in3, in4: IN std_logic_vector(N-1 DOWNTO 0);
	B1_in, B2_in, B3_in, B4_in: IN std_logic;
	B1_out, B2_out, B3_out, B4_out: OUT std_logic;
	rut1_in, rut2_in, rut3_in, rut4_in: IN std_logic_vector(m-1 DOWNTO 0);
	rut1_out, rut2_out, rut3_out, rut4_out: OUT std_logic_vector(m-1 DOWNTO 0);
	out1, out2, out3, out4: OUT std_logic_vector(N-1 DOWNTO 0);
	kaskada1:IN INTEGER:=1;
	kaskada2:IN INTEGER:=2
	);
	END COMPONENT;
SIGNAL k1_1:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL k1_2:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL k1_3:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
```

```
SIGNAL k1_4:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL k1_5:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL k1_6:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL k1_7:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL k1_8:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL rut1_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL rut2_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL rut3_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL rut4_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL rut5_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL rut6_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL rut7_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL rut8_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL B1_loc:STD_LOGIC;
SIGNAL B2_loc:STD_LOGIC;
SIGNAL B3_loc:STD_LOGIC;
SIGNAL B4_loc:STD_LOGIC;
SIGNAL B5_loc:STD_LOGIC;
SIGNAL B6_loc:STD_LOGIC;
SIGNAL B7_loc:STD_LOGIC;
SIGNAL B8_loc:STD_LOGIC;

BEGIN

Baseline2x2_inst1: Baseline2x2
        GENERIC MAP
        (
                N=>N,
                m=>m
        )
        PORT MAP
        (
        clk => clk,
        reset => reset,
        kaskada=>kaskada1,
        B1_in=>B1_in,
        B2_in=>B2_in,
        B1_out=>B1_loc,
        B2_out=>B2_loc,
        in1=>in1,
        in2=>in2,
        rut1_in=>rut1_in,
        rut2_in=>rut2_in,
        rut1_out=>rut1_loc,
        rut2_out=>rut2_loc,
```

```
        out1=>k1_1,
        out2=>k1_2
        );

Baseline2x2_inst2: Baseline2x2
        GENERIC MAP
        (
                N=>N,
                m=>m
        )
        PORT MAP
        (
        clk => clk,
        reset => reset,
        kaskada=>kaskada1,
        B1_in=>B3_in,
        B2_in=>B4_in,
        B1_out=>B3_loc,
        B2_out=>B4_loc,
        in1=>in3,
        in2=>in4,
        rut1_in=>rut3_in,
        rut2_in=>rut4_in,
        rut1_out=>rut3_loc,
        rut2_out=>rut4_loc,
        out1=>k1_3,
        out2=>k1_4
        );

Baseline2x2_inst3: Baseline2x2
        GENERIC MAP
        (
                N=>N,
                m=>m
        )
        PORT MAP
        (
        clk => clk,
        reset => reset,
        kaskada=>kaskada1,
        B1_in=>B5_in,
        B2_in=>B6_in,
        B1_out=>B5_loc,
        B2_out=>B6_loc,
        in1=>in5,
```

```
        in2=>in6,
        rut1_in=>rut5_in,
        rut2_in=>rut6_in,
        rut1_out=>rut5_loc,
        rut2_out=>rut6_loc,
        out1=>k1_5,
        out2=>k1_6
        );

Baseline2x2_inst4: Baseline2x2
        GENERIC MAP
        (
                N=>N,
                m=>m
        )
        PORT MAP
        (
        clk => clk,
        reset => reset,
        kaskada=>kaskada1,
        B1_in=>B7_in,
        B2_in=>B8_in,
        B1_out=>B7_loc,
        B2_out=>B8_loc,
        in1=>in7,
        in2=>in8,
        rut1_in=>rut7_in,
        rut2_in=>rut8_in,
        rut1_out=>rut7_loc,
        rut2_out=>rut8_loc,
        out1=>k1_7,
        out2=>k1_8
        );

Baseline4x4_inst1: Baseline4x4
        GENERIC MAP
        (
                N=>N,
                m=>m
        )
        PORT MAP
        (
        clk => clk,
        reset => reset,
        kaskada1=>kaskada2,
```

```
        kaskada2=>kaskada3,
        B1_in=>B1_loc,
        B2_in=>B3_loc,
        B3_in=>B5_loc,
        B4_in=>B7_loc,
        B1_out=>B1_out,
        B2_out=>B2_out,
        B3_out=>B3_out,
        B4_out=>B4_out,
        in1=>k1_1,
        in2=>k1_3,
        in3=>k1_5,
        in4=>k1_7,
        rut1_in=>rut1_loc,
        rut2_in=>rut3_loc,
        rut3_in=>rut5_loc,
        rut4_in=>rut7_loc,
        rut1_out=>rut1_out,
        rut2_out=>rut2_out,
        rut3_out=>rut3_out,
        rut4_out=>rut4_out,
        out1=>out1,
        out2=>out2,
        out3=>out3,
        out4=>out4
        );

Baseline4x4_inst2: Baseline4x4
        GENERIC MAP
        (
                N=>N,
                m=>m
        )
        PORT MAP
        (
        clk => clk,
        reset => reset,
        kaskada1=>kaskada2,
        kaskada2=>kaskada3,
        B1_in=>B2_loc,
        B2_in=>B4_loc,
        B3_in=>B6_loc,
        B4_in=>B8_loc,
        B1_out=>B5_out,
        B2_out=>B6_out,
```

```
        B3_out=>B7_out,
        B4_out=>B8_out,
        in1=>k1_2,
        in2=>k1_4,
        in3=>k1_6,
        in4=>k1_8,
        rut1_in=>rut2_loc,
        rut2_in=>rut4_loc,
        rut3_in=>rut6_loc,
        rut4_in=>rut8_loc,
        rut1_out=>rut5_out,
        rut2_out=>rut6_out,
        rut3_out=>rut7_out,
        rut4_out=>rut8_out,
        out1=>out5,
        out2=>out6,
        out3=>out7,
        out4=>out8
        );
END sema;
```

A.4. Entitet *baseline* 16x16 komutatora

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
Library XilinxCoreLib;
ENTITY Baseline16x16 IS
GENERIC
(
N: INTEGER:=16;
m: INTEGER:=4
);
PORT
(
clk,reset: IN std_logic;
in1, in2, in3, in4, in5, in6, in7, in8, in9, in10, in11, in12, in13, in14, in15,
in16: IN std_logic_vector(N-1 DOWNTO 0);
B1_in, B2_in, B3_in, B4_in, B5_in, B6_in, B7_in, B8_in, B9_in, B10_in, B11_in,
B12_in, B13_in, B14_in, B15_in, B16_in: IN std_logic;
B1_out, B2_out, B3_out, B4_out, B5_out, B6_out, B7_out, B8_out, B9_out, B10_out,
B11_out, B12_out, B13_out, B14_out, B15_out, B16_out: OUT std_logic;
rut1_in, rut2_in, rut3_in, rut4_in, rut5_in, rut6_in, rut7_in, rut8_in, rut9_in,
rut10_in, rut11_in, rut12_in, rut13_in, rut14_in, rut15_in, rut16_in: IN
```

```
std_logic_vector(m-1 DOWNTO 0);
rut1_out, rut2_out, rut3_out, rut4_out, rut5_out, rut6_out, rut7_out, rut8_out,
rut9_out, rut10_out, rut11_out, rut12_out, rut13_out, rut14_out, rut15_out,
rut16_out: OUT std_logic_vector(m-1 DOWNTO 0);
out1, out2, out3, out4, out5, out6, out7, out8, out9, out10, out11, out12, out13,
out14, out15, out16: OUT std_logic_vector(N-1 DOWNTO 0);
kaskada1:IN INTEGER:=1;
kaskada2:IN INTEGER:=2;
kaskada3:IN INTEGER:=3;
kaskada4:IN INTEGER:=4
);
 END Baseline16x16;
ARCHITECTURE sema OF Baseline16x16 IS
        COMPONENT Baseline2x2 IS
        GENERIC
        (
        N: INTEGER:=16;
        m: INTEGER:=1
        );
        PORT(
        clk,reset: IN std_logic;
        B1_in, B2_in: IN std_logic;
        B1_out, B2_out: OUT std_logic;
        rut1_in, rut2_in: IN std_logic_vector(m-1 DOWNTO 0);
        rut1_out, rut2_out: OUT std_logic_vector(m-1 DOWNTO 0);
        in1, in2: IN std_logic_vector(N-1 DOWNTO 0);
        out1, out2: OUT std_logic_vector(N-1 DOWNTO 0);
        kaskada: IN integer:=1
        );
        END COMPONENT;

        COMPONENT Baseline4x4 IS
        GENERIC
        (
        N: INTEGER:=16;
        m: INTEGER:=2
        );
        PORT(
        clk,reset: IN std_logic;
        in1, in2, in3, in4: IN std_logic_vector(N-1 DOWNTO 0);
        B1_in, B2_in, B3_in, B4_in: IN std_logic;
        B1_out, B2_out, B3_out, B4_out: OUT std_logic;
        rut1_in, rut2_in, rut3_in, rut4_in: IN std_logic_vector(m-1 DOWNTO 0);
        rut1_out, rut2_out, rut3_out, rut4_out: OUT std_logic_vector(m-1 DOWNTO 0);
        out1, out2, out3, out4: OUT std_logic_vector(N-1 DOWNTO 0);
```

```
        kaskada1:IN INTEGER:=1;
        kaskada2:IN INTEGER:=2
        );
          END COMPONENT;

        COMPONENT Baseline8x8 IS
        GENERIC
        (
        N: INTEGER:=16;
        m: INTEGER:=3
        );
        PORT
        (
        clk,reset: IN std_logic;
        in1, in2, in3, in4, in5, in6, in7, in8: IN std_logic_vector(N-1 DOWNTO 0);
        B1_in, B2_in, B3_in, B4_in, B5_in, B6_in, B7_in, B8_in: IN std_logic;
        B1_out, B2_out, B3_out, B4_out, B5_out, B6_out, B7_out, B8_out: OUT
        std_logic;
        rut1_in, rut2_in, rut3_in, rut4_in, rut5_in, rut6_in, rut7_in, rut8_in: IN
        std_logic_vector(m-1 DOWNTO 0);
        rut1_out, rut2_out, rut3_out, rut4_out, rut5_out, rut6_out, rut7_out,
        rut8_out: OUT std_logic_vector(m-1 DOWNTO 0);
        out1, out2, out3, out4, out5, out6, out7, out8: OUT std_logic_vector(N-1
        DOWNTO 0);
        kaskada1:IN INTEGER:=1;
        kaskada2:IN INTEGER:=2;
        kaskada3:IN INTEGER:=3
        );
        END COMPONENT;

SIGNAL k1_1:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL k1_2:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL k1_3:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL k1_4:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL k1_5:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL k1_6:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL k1_7:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL k1_8:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL k1_9:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL k1_10:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL k1_11:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL k1_12:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL k1_13:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL k1_14:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL k1_15:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
```

```
SIGNAL k1_16:STD_LOGIC_VECTOR(N-1 DOWNTO 0);
SIGNAL rut1_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL rut2_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL rut3_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL rut4_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL rut5_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL rut6_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL rut7_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL rut8_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL rut9_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL rut10_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL rut11_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL rut12_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL rut13_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL rut14_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL rut15_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL rut16_loc:STD_LOGIC_VECTOR(m-1 DOWNTO 0);
SIGNAL B1_loc:STD_LOGIC;
SIGNAL B2_loc:STD_LOGIC;
SIGNAL B3_loc:STD_LOGIC;
SIGNAL B4_loc:STD_LOGIC;
SIGNAL B5_loc:STD_LOGIC;
SIGNAL B6_loc:STD_LOGIC;
SIGNAL B7_loc:STD_LOGIC;
SIGNAL B8_loc:STD_LOGIC;
SIGNAL B9_loc:STD_LOGIC;
SIGNAL B10_loc:STD_LOGIC;
SIGNAL B11_loc:STD_LOGIC;
SIGNAL B12_loc:STD_LOGIC;
SIGNAL B13_loc:STD_LOGIC;
SIGNAL B14_loc:STD_LOGIC;
SIGNAL B15_loc:STD_LOGIC;
SIGNAL B16_loc:STD_LOGIC;

BEGIN

Baseline2x2_inst1: Baseline2x2
        GENERIC MAP
        (
                N=>N,
                m=>m
        )
        PORT MAP
        (
        clk => clk,
```

```
        reset => reset,
        kaskada=>kaskada1,
        B1_in=>B1_in,
        B2_in=>B2_in,
        B1_out=>B1_loc,
        B2_out=>B2_loc,
        in1=>in1,
        in2=>in2,
        rut1_in=>rut1_in,
        rut2_in=>rut2_in,
        rut1_out=>rut1_loc,
        rut2_out=>rut2_loc,
        out1=>k1_1,
        out2=>k1_2
        );

Baseline2x2_inst2: Baseline2x2
        GENERIC MAP
        (
                N=>N,
                m=>m
        )
        PORT MAP
        (
        clk => clk,
        reset => reset,
        kaskada=>kaskada1,
        B1_in=>B3_in,
        B2_in=>B4_in,
        B1_out=>B3_loc,
        B2_out=>B4_loc,
        in1=>in3,
        in2=>in4,
        rut1_in=>rut3_in,
        rut2_in=>rut4_in,
        rut1_out=>rut3_loc,
        rut2_out=>rut4_loc,
        out1=>k1_3,
        out2=>k1_4
        );

Baseline2x2_inst3: Baseline2x2
        GENERIC MAP
        (
                N=>N,
```

```
		m=>m
	)
	PORT MAP
	(
	clk => clk,
	reset => reset,
	kaskada=>kaskada1,
	B1_in=>B5_in,
	B2_in=>B6_in,
	B1_out=>B5_loc,
	B2_out=>B6_loc,
	in1=>in5,
	in2=>in6,
	rut1_in=>rut5_in,
	rut2_in=>rut6_in,
	rut1_out=>rut5_loc,
	rut2_out=>rut6_loc,
	out1=>k1_5,
	out2=>k1_6
	);


Baseline2x2_inst4: Baseline2x2
	GENERIC MAP
	(
		N=>N,
		m=>m
	)
	PORT MAP
	(
	clk => clk,
	reset => reset,
	kaskada=>kaskada1,
	B1_in=>B7_in,
	B2_in=>B8_in,
	B1_out=>B7_loc,
	B2_out=>B8_loc,
	in1=>in7,
	in2=>in8,
	rut1_in=>rut7_in,
	rut2_in=>rut8_in,
	rut1_out=>rut7_loc,
	rut2_out=>rut8_loc,
	out1=>k1_7,
	out2=>k1_8
```

```
	);

Baseline2x2_inst5: Baseline2x2
	GENERIC MAP
	(
		N=>N,
		m=>m
	)
	PORT MAP
	(
	clk => clk,
	reset => reset,
	kaskada=>kaskada1,
	B1_in=>B9_in,
	B2_in=>B10_in,
	B1_out=>B9_loc,
	B2_out=>B10_loc,
	in1=>in9,
	in2=>in10,
	rut1_in=>rut9_in,
	rut2_in=>rut10_in,
	rut1_out=>rut9_loc,
	rut2_out=>rut10_loc,
	out1=>k1_9,
	out2=>k1_10
	);

Baseline2x2_inst6: Baseline2x2
	GENERIC MAP
	(
		N=>N,
		m=>m
	)
	PORT MAP
	(
	clk => clk,
	reset => reset,
	kaskada=>kaskada1,
	B1_in=>B11_in,
	B2_in=>B12_in,
	B1_out=>B11_loc,
	B2_out=>B12_loc,
	in1=>in11,
	in2=>in12,
	rut1_in=>rut11_in,
```

```
        rut2_in=>rut12_in,
        rut1_out=>rut11_loc,
        rut2_out=>rut12_loc,
        out1=>k1_11,
        out2=>k1_12
        );

Baseline2x2_inst7: Baseline2x2
        GENERIC MAP
        (
                N=>N,
                m=>m
        )
        PORT MAP
        (
        clk => clk,
        reset => reset,
        kaskada=>kaskada1,
        B1_in=>B13_in,
        B2_in=>B14_in,
        B1_out=>B13_loc,
        B2_out=>B14_loc,
        in1=>in13,
        in2=>in14,
        rut1_in=>rut13_in,
        rut2_in=>rut14_in,
        rut1_out=>rut13_loc,
        rut2_out=>rut14_loc,
        out1=>k1_13,
        out2=>k1_14
        );

Baseline2x2_inst8: Baseline2x2
        GENERIC MAP
        (
                N=>N,
                m=>m
        )
        PORT MAP
        (
        clk => clk,
        reset => reset,
        kaskada=>kaskada1,
        B1_in=>B15_in,
        B2_in=>B16_in,
```

```
        B1_out=>B15_loc,
        B2_out=>B16_loc,
        in1=>in15,
        in2=>in16,
        rut1_in=>rut15_in,
        rut2_in=>rut16_in,
        rut1_out=>rut15_loc,
        rut2_out=>rut16_loc,
        out1=>k1_15,
        out2=>k1_16
        );

Baseline8x8_inst1: Baseline8x8
        GENERIC MAP
        (
               N=>N,
               m=>m
        )
        PORT MAP
        (
        clk => clk,
        reset => reset,
        kaskada1=>kaskada2,
        kaskada2=>kaskada3,
        kaskada3=>kaskada4,
        B1_in=>B1_loc,
        B2_in=>B3_loc,
        B3_in=>B5_loc,
        B4_in=>B7_loc,
        B5_in=>B9_loc,
        B6_in=>B11_loc,
        B7_in=>B13_loc,
        B8_in=>B15_loc,
        B1_out=>B1_out,
        B2_out=>B2_out,
        B3_out=>B3_out,
        B4_out=>B4_out,
        B5_out=>B5_out,
        B6_out=>B6_out,
        B7_out=>B7_out,
        B8_out=>B8_out,
        in1=>k1_1,
        in2=>k1_3,
        in3=>k1_5,
        in4=>k1_7,
```

```
        in5=>k1_9,
        in6=>k1_11,
        in7=>k1_13,
        in8=>k1_15,
        rut1_in=>rut1_loc,
        rut2_in=>rut3_loc,
        rut3_in=>rut5_loc,
        rut4_in=>rut7_loc,
        rut5_in=>rut9_loc,
        rut6_in=>rut11_loc,
        rut7_in=>rut13_loc,
        rut8_in=>rut15_loc,
        rut1_out=>rut1_out,
        rut2_out=>rut2_out,
        rut3_out=>rut3_out,
        rut4_out=>rut4_out,
        rut5_out=>rut5_out,
        rut6_out=>rut6_out,
        rut7_out=>rut7_out,
        rut8_out=>rut8_out,
        out1=>out1,
        out2=>out2,
        out3=>out3,
        out4=>out4,
        out5=>out5,
        out6=>out6,
        out7=>out7,
        out8=>out8
        );

Baseline8x8_inst2: Baseline8x8
        GENERIC MAP
        (
               N=>N,
               m=>m
        )
        PORT MAP
        (
        clk => clk,
        reset => reset,
        kaskada1=>kaskada2,
        kaskada2=>kaskada3,
        kaskada3=>kaskada4,
        B1_in=>B2_loc,
        B2_in=>B4_loc,
```

```
B3_in=>B6_loc,
B4_in=>B8_loc,
B5_in=>B10_loc,
B6_in=>B12_loc,
B7_in=>B14_loc,
B8_in=>B16_loc,
B1_out=>B9_out,
B2_out=>B10_out,
B3_out=>B11_out,
B4_out=>B12_out,
B5_out=>B13_out,
B6_out=>B14_out,
B7_out=>B15_out,
B8_out=>B16_out,
in1=>k1_2,
in2=>k1_4,
in3=>k1_6,
in4=>k1_8,
in5=>k1_10,
in6=>k1_12,
in7=>k1_14,
in8=>k1_16,
rut1_in=>rut2_loc,
rut2_in=>rut4_loc,
rut3_in=>rut6_loc,
rut4_in=>rut8_loc,
rut5_in=>rut10_loc,
rut6_in=>rut12_loc,
rut7_in=>rut14_loc,
rut8_in=>rut16_loc,
rut1_out=>rut9_out,
rut2_out=>rut10_out,
rut3_out=>rut11_out,
rut4_out=>rut12_out,
rut5_out=>rut13_out,
rut6_out=>rut14_out,
rut7_out=>rut15_out,
rut8_out=>rut16_out,
out1=>out9,
out2=>out10,
out3=>out11,
out4=>out12,
out5=>out13,
out6=>out14,
out7=>out15,
```

```
        out8=>out16
        );
END sema;
```