ELEKTROTEHNIČKI FAKULTET UNIVERZITETA U BEOGRADU

HARDVERSKA IMPLEMENTACIJA MODULA ZA SORTIRANJE PAKETA

–Master rad–

Kandidat:
Budimir Miletić 2013/3076

Mentor:
doc. dr Zoran Čiča

Beograd, Septembar 2015.

SADRŽAJ

1. UVOD

Sa ulaskom u 21. vek, Internet je nastavio sa ogromnim porastom. Bez ikakvog merenja, porast je primetan na svim frontovima: broj hostova, broj korisnika, količina saobraćaja, broj linkova, kapacitet pojedinačnih linkova, ili porast mreža nudilaca Internet servisa. Taj kontinualni brzi porast, povezan sa raznolikim servisima za koje se očekuje da će Internet da obezbedi, stvorio je dva glavna izazova u projektovanju i implementaciji komutacije paketa (IP ruteri, ATM komutatori, Gigabit Ethernet komutatori i FR komutatori), upravljanje redovima čekanja i prosleđivanje. [3]

Krosbar komutatori imaju dobre osobine poput jednostavnosti i neblokiranja, ali su problematični za realizaciju velikih komutacionih polja zbog potrebe za velikim brojem prekidača. Banyan komutatori predstavljaju jednostavna rešenja sa optimalnijim strukturama koje omogućavaju efikasniju konstrukciju velikih komutatora (sa velikim brojem ulaza i izlaza). Banyan strukture imaju osobinu samoprosleđivanja 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, što je cilj ove teze. [2]

U drugom poglavlju ovog rada će ukratko biti reči o vrstama komutacija i komutatora. Takođe će biti predstavljen Banyan komutator. U trećem poglavlju su opisane struktura za sortiranje, bitonička i par-nepar, i njihov princip rada. Nakon teorijskog dela, u četvrtom poglavlju, biće opisana implementacija bitoničke i par-nepar strukture za sortiranje kao i opis principa rada dizajna, ulaznih i izlaznih signala. U poglavlju 5 su prikazani primeri za različite dimenzije komutatora, kao i za različite dužine magistrala. Na ovaj način biće analizirane performanse struktura kao i njihovo poređenje. Na kraju je dat zaključak u kome su izložena završna razmatranja teze.

2. KOMUTATORI

Tehnika komutacije paketa se zasniva na činjenici da se u komunikaciji između korisnika razmenjuju paketi, tj. da su podaci koje korisnici razmenjuju smešteni u pakete. Svaki paket se prenosi kroz mrežu prolazeći kroz mrežne čvorove. Po ulasku u mrežni čvor, određuje se na koji izlaz mrežnog čvora treba poslati dotični paket. Paket zatim čeka na svoj redosled za slanje zajedno sa paketima drugih veza koji se šalju preko istog izlaza mrežnog čvora. Očigledno, preko izlaza, tj. linka, koji je povezan na dotični izlaz, se uvek šalju paketi ako ih ima za slanje, tako da pauza u jednoj vezi ne utiče na iskorišćenost linka koji će tada da prenosi pakete drugih veza koje su takođe u toku. Komutacija paketa postiže visoku iskorišćenost resursa u mreži, ali sada nedostaje garancija kvaliteta servisa, tj. ne može se znati kakvu uslugu će dobiti korisnik. 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 Brz razvoj mreža za prenos podataka doveo je do potrebe komutacionih čvorova za komutaciju podataka tako da je tehnika komutacije paketa je danas dominantna tehnologija.

Paketski komutatori mogu da se podele na:

1. vremenske komutatore i
2. prostorne komutatore.

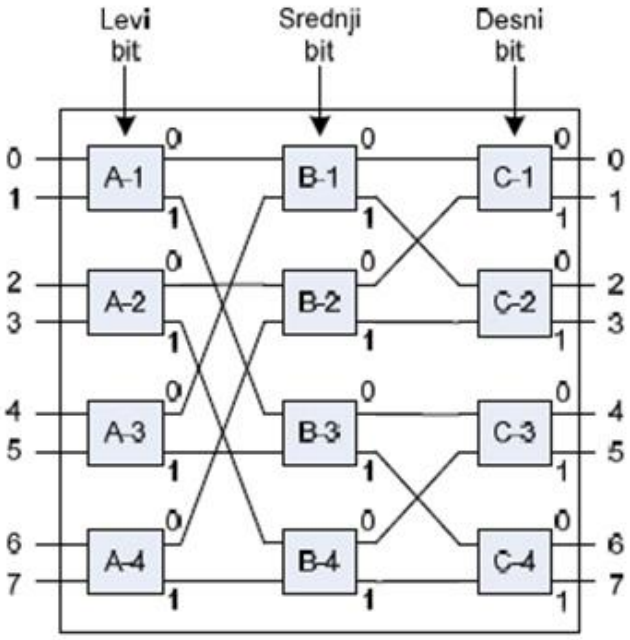
Vremenski komutatori izvršavaju komutaciju paketa u vremenskom domenu, a prostorni u prostornom domenu. Prostorni komutatori se dele na:

1. komutatore sa jednostrukim putanjama
2. komutatore sa višestrukim putanjama

Komutatori sa jednostrukim putanjama predstavljaju komutatore kod kojih postoji samo jedan put između ulaza i i izlaza. Ako postoji više putanja, tada je u pitanju komutator sa višestrukim putanjama. Komutatori sa jednostrukim putanjama se dalje dele na:

1. krosbar komutatore,
2. potpuno povezani komutatore i
3. Banyan komutatore. [2]

Banyan komutator je višestepeni komutator sa prekidačima u svakom stepenu koji usmeravaju pakete shodno reperezentaciji rednog broja izlaznog porta u binarnom obliku. Za n ulaza i n izlaza, mreža sadrži $\log_2$ (n) stepena sa po n/2 prekidača. Prvi stepen usmerava pakete shodno bitu najveće težine rednog broja izlaznog porta, drugi stepen odluku o usmeravanju donosi na osnovu prvog sledećeg bita i td. [4]



Slika 2.1. Banyan komutator sa 8 ulaza i 8 izlaza. [5]

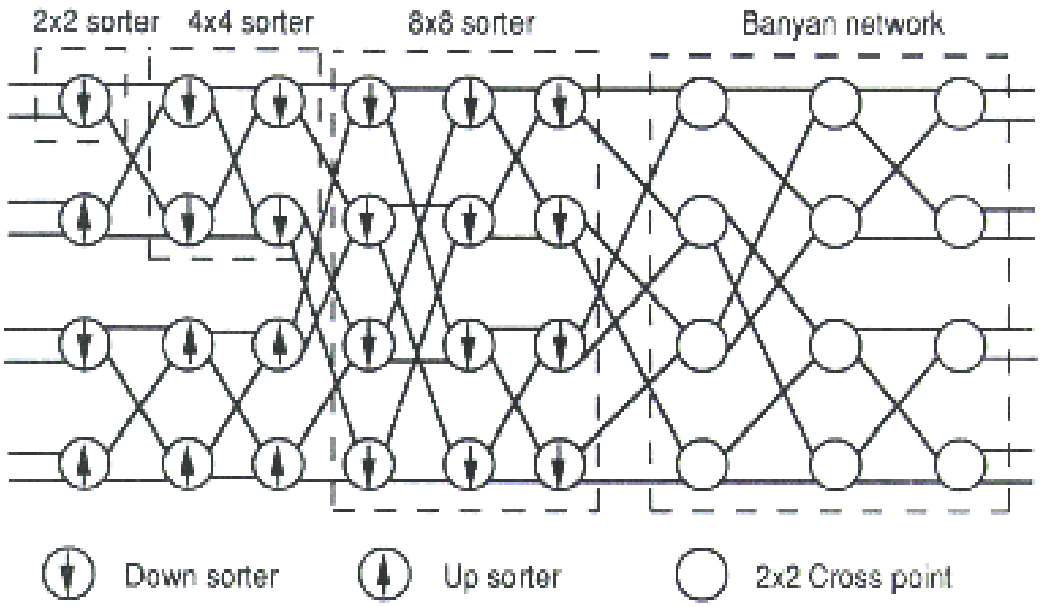
Banyan strukture poseduju osobinu samorutiranja. Usled samorutiranja može doći do kolizije. Banyan komutatori su interno blokirajući komutatori, što znači da za pojedine parove ulaz/izlaz nije moguće istovremeno proslediti pakete usled unutrašnje blokade. U slučaju kolizije, samo jedan paket se propušta što znači da će drugi paket biti izgubljen, što bi trebalo izbeći. Zbog toga se uz Banyan strukture tipično koriste i mreže (strukture) za sortiranje koje dovode pakete na ulaze strukture u sortiranom redosledu (bez neaktivnih ulaza između aktivnih ulaza). U slučaju kad su paketi sortirani, ne može doći do interne blokade, pod uslovom da svi paketi imaju različite destinacije tj.izlazne portove. [2]

3. STRUKTURE ZA SORTIRANJE

Postoji više struktura za sortiranje. Jedna od poznatijih struktura je Batcher struktura koja se često koristi u kombinaciji sa Banyan strukturama, pa se često koristi termin Batcher-Banyan struktura (komutator) čime se označava Banyan struktura (omega ili n-cube) ispred koje se nalazi Batcher struktura za sortiranje.

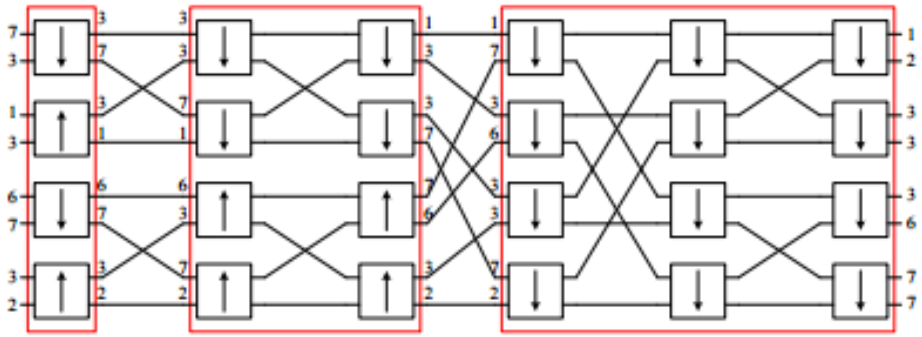
Batcher struktura se sastoji iz delova (tzv. merge kaskada), pri čemu su svi delovi sastavljeni od 2x2 svičeva. Ukupan broj delova je jednak $\log_2 N$, gde je N broj ulaznih, odnosno izlaznih portova. Prvi deo ima samo jednu kaskadu, drugi deo ima dve kaskade, treći deo tri kaskade i tako redom do poslednjeg dela koji ima $\log_2 N$ kaskada. Svaka kaskada sadrži N/2 svičeva. Na slici 3.1 je prikazana Batcher (2x2, 4x4, 8x8) - Banyan struktura. Struktura 2x2 sortira 2 ulazna porta, 4x4x sortira 4, a 8x8 sortira 8 ulaznih portova, tačnije pakete koje oni šalju. Kao što vidimo Batcher 2x2 struktura se sastoji od 1 dela, 4x4 se sastoji iz 2 dela, odnosno 8x8 struktura se sastoji iz tri dela, čiji se broj kaskada uvećava za jedan sa rednim brojem dela (prvi deo - jedna kaskada, drugi deo - dve kaskade i treći deo - tri kaskade).

Obratićemo pažnju na 8x8 strukturu. Ideja Batcher strukture je jednostavna. Prvi deo ima za zadatak da formira N/2 sortiranih listi od 2 člana (na osnovu N/2 parova sortiranih listi od jednog člana), drugi deo formira N/4 listi od 4 člana (na osnovu N/4 parova sortiranih listi od dva člana) i tako redom do poslednjeg dela koji formira jednu sortiranu listu od N članova (na osnovu 1 para sortiranih listi od N/2 članova). Ako pogledamo sliku 3.1, videćemo da upravo tako funkcioniše Batcher struktura. Smer strelice u sviču ukazuje na koji izlaz sviča se usmerava paket sa većom vrednošću odredišne adrese. Ukoliko je na ulazu prisutan samo jedan paket, tada se on usmerava na izlaz koji odgovara manjoj vrednosti odredišne adrese. Prvi deo formira 4 sortirane liste od po dva člana, pri čemu smer strelice u dotičnom redu dela određuje tip sortiranja - opadajući (strelica nagore) ili rastući (strelica nadole) redosled. Drugi deo formira 2 sortirane liste od po četiri člana, pri čemu smer strelice u dotičnom redu dela određuje tip sortiranja na identičan način kao kod prvog dela. Poslednji (treći) deo formira finalnu sortiranu listu paketa po rastućem redosledu koja se prosleđuje na ulaze Banyan strukture. Struktura samih delova je jednaka odgovarajućoj n-cube strukturi. Tako se treći deo sastoji od jedne 3-cube strukture, drugi deo od dve 2-cube strukture, a prvi deo od četiri 1-cube strukture (koje odgovaraju samim svičevima). Pri tome neparne n-cube strukture u delovima imaju svičeve kod kojih su strelice orijentisane nadole, a parne n-cube strukture u delovima imaju svičeve kod kojih su strelice orijentisane nagore (pretpostavka je da se n-cube strukture u delu broje od 1, pri čemu se broje u smeru nadole). Engleski termin merge network se koristi za označavanje mreža koje vrše spajanje dveju sortiranih listi sa ulaza u merge mrežu u jednu sortiranu listu. Kao što smo naveli Batcher struktura se sastoji od merge delova koji sadrže odgovarajući broj merge mreža - vidimo da se broj listi dvostruko smanjuje na izlazu iz svakog od delova, i takođe broj merge mreža koje rade u paraleli u jednom delu zavisi od broja parova listi koje se spajaju.

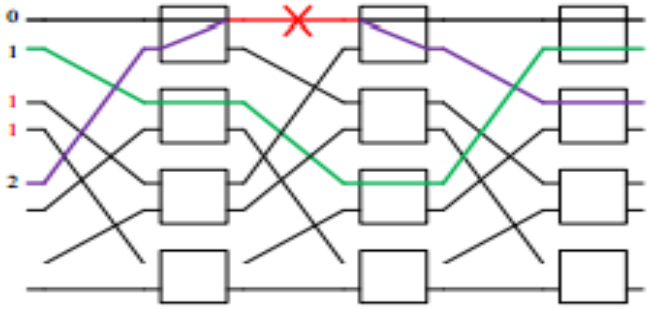


Slika 3.1 Batcher-Banyan struktura. [6]

U slučaju da nema spoljašnje kontrole (raspoređivača) koja bi nadgledala da ulazni portovi ne šalju istovremeno pakete na isti izlazni port, može se desiti situacija da na ulazu u Batcher mrežu uđu paketi namenjeni istom izlaznom portu. Kada dva paketa namenjena istom izlaznom portu dođu na ulaze istog sviča u Batcher strukturi, svejedno je kako će ta dva paketa biti prosleđena na izlaze sviča. Nakon sortiranja, paketi namenjeni istom izlaznom portu će biti jedni do drugih. Primer ove situacije je prikazan na slici 3.2 za 8x8 Batcher strukturu. Kao što se vidi, paketi namenjeni izlaznom portu 3 su sortirani jedni do drugih, a isto važi i za pakete namenjene izlaznom portu 7. Lista je i dalje sortirana u neopadajućem redosledu (nije rastuća jer postoje isti brojevi u nizu). Ovakva situacija može da izazove interno blokiranje jer će da se naruši uslov da ne sme da postoji neaktivan ulaz između aktivnih ulaza. Naime, samo jedan od paketa namenjenih istom izlaznom portu sme da se prosledi što znači da će nastati eventualna rupa između paketa koji se prosleđuju (rupa neće nastati samo ako nema većih elemenata u listi tj. paketa se većom vrednošću odredišne adrese jer će tada neprosleđeni elementi biti na samom kraju sortirane liste), a rupu će da formiraju neprosleđeni paketi. Primer blokiranja koji se javlja u ovakvoj situaciji je dat na slici 3.3 (paketi čije je odredište označeno crvenom bojom se ne prosleđuju jer su namenjeni istom odredištu kao paket sa drugog ulaza). Stoga je veoma važno izbeći ovakvu situaciju i obezbediti da se ne prosleđuje više paketa namenjenih istom portu kada se koristi struktura za sortiranje jer njena primena ne bi imala željenog efekta.



Slika 3.2 Primer Batcher strukture kada ima više paketa namenjenih istom izlaznom portu. [2]



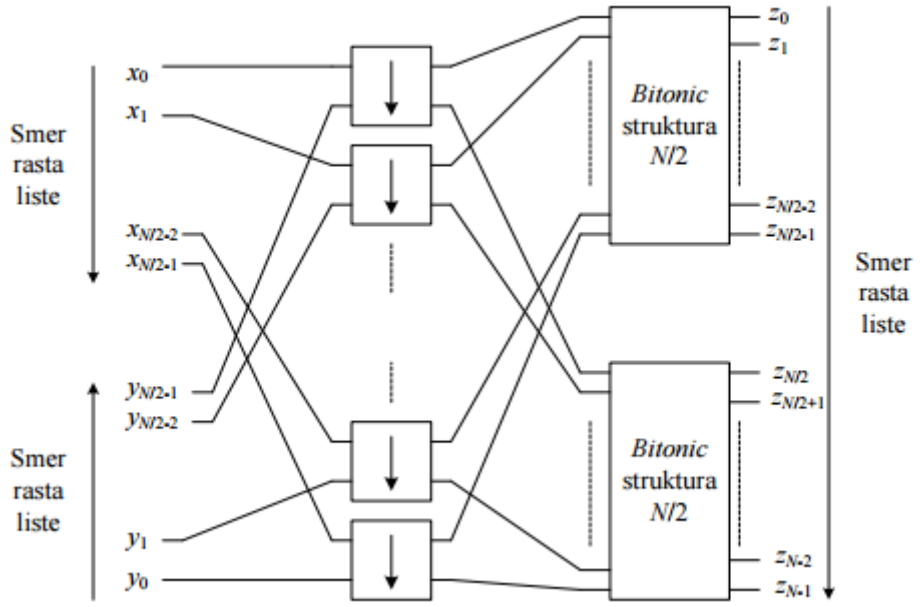
Slika 3.3 Primer internog blokiranja usled pojave rupe između sortiranih paketa zbog neprosleđivanja paketa namenjenih istom portu. [2]

Postoji više rešenja koja se mogu primeniti kako bi se sprečila situacija da se više paketa namenjenih istom portu pošalje istovremeno. Jedan način je upotreba algoritama raspoređivanja ako se Batcher-Banyan komutator koristi u strukturi sa baferima na ulazu. Druga varijanta se sastoji iz tri faze. U prvoj fazi se šalju samo odredišne adrese. Na izlazu Batcher strukture se vrši provera za svaki član sortiranog niza da li je njegov prethodnik namenjen istom izlaznom portu ili ne. Ako se prethodnik razlikuje, dotičnom ulaznom portu se šalje pozitivna potvrda (druga faza). Samo ulazni portovi koji su primili pozitivnu potvrdu će poslati svoje pakete kroz Batcher-Banyan komutator (treća faza), dok će drugi baferisati dotične pakete i pokušati u narednim slotovima. Pošto se u trećoj fazi ne šalju paketi namenjeni istom portu, Batcher struktura za sortiranje će kreirati rastuću listu i samim tim će Banyan struktura moći bez internog blokiranja da komutira sve pakete sa svojih ulaza. Ova varijanta je jednostavna za implementaciju jer koristi resurse koji se ionako koriste za komutaciju paketa, ali je problem što nije efikasna kao algoritmi raspoređivanja, ukoliko se na ulazu koriste VOQ redovi za čekanje (ako su u pitanju FIFO baferi tada je ova varijanta optimalna tj. ne mogu se postići bolji rezultati od nje, ali tada može doći do HOL blokade). Postoje i druge varijante za sprečavanje da više paketa bude poslato na isti izlazni port. Napomenimo da je Batcher struktura za sortiranje zasnovana na algoritmu za spajanje dve sortirane liste u jednu zajedničku sortiranu listu pod nazivom bitoničko spajanje (bitonic merge). Pored ovog algoritma postoji i algoritam par-nepar spajanje (odd-even merge). [2]

3.1. Bitonička struktura

Ideja bitoničkog spajanja je da poredi članove liste u obrnutom redosledu (najmanji član jedne liste sa najvećim članom druge liste, drugi po redu najmanji član liste sa drugim po redu najvećim članom liste,...) i da one koji su manji šalje u jednu bitoničku strukturu dimenzije N/2, a one koji su veći u drugu bitoničku strukturu dimenzije N/2. Na ovaj način će biti prosleđeni u gornju bitoničku strukturu dimenzije N/2 elementi koji će pripadati nižoj polovini konačnog sortiranog niza, a u donju bitoničku strukturu dimenzije N/2 elementi koji će pripadati višoj polovini konačnog sortiranog niza. Naime prvo poređenje gde x_i bude veći od odgovarajućeg člana niza y s kojim se poredi, x_i i svi članovi niza x iza njega (sa većim indeksom) će biti prosleđeni u donju bitoničku strukturu jer će oni sigurno biti veći u preostalim poređenjima (y članovi će biti sve nižih vrednosti, a x članovi sve viših vrednosti). Pre te situacije su se prosleđivali članovi y niza na donju bitoničku strukturu. To znači da će sigurno na ulaz donje bitoničke strukture da bude dovedeno N/2 najviših vrednosti gledajući oba niza zbirno (član x niza najmanje vrednosti a da je prosleđen na donju bitoničku strukturu ima sigurno veću vrednost od svih članova x niza prosleđenih na gornju strukturu, a isto važi i za y članove koji su prosleđeni na gornju strukturu jer se sa njihovim najvećim predstavnikom upravo poredio član x niza najmanje vrednosti koji je prosleđen na donju bitoničku strukturu; analogan zaključak se može izvesti i za y člana najmanje vrednosti a da je prosleđen na donju bitoničku strukturu). Pošto se bitoničke strukture dimenzije N/2 formiraju na analogan način, svaka od bitoničkih struktura dimenzije N/2 će sortirati u rastućem redosledu članove sa svojih ulaza i time će se spajanjem ta dva niza sa izlaza bitoničkih struktura dimenzije N/2 dobiti kompletno sortiran niz dimenzije N (sortiran u rastućem redosledu). Inače, termin bitonički niz označava niz kod koga postoji indeks k takav da je niz do tog člana sa indeksom k rastući, a posle njega niz je opadajući (na primer, niz 0,1,2,3,4,2,1 je bitonički). Bitonički niz je i onaj niz kod kog je niz prvo opadajući (do člana sa indeksom k) pa rastući. Ulaz u bitoničku strukturu dimenzije N je bitonički (ulaz podrazumeva gledanje svih ulaza kao jednog niza $x_0x_1...x_{N/2-1}y_{N/2-1},..,y_1,y_0$) jer x niz raste i potom će rasti za još jedan član ako je najveći član y niza veći od najvećeg člana x niza i nakon toga će doći do opadanja vrednosti. Ako je najveći član y niza manji od najvećeg člana x niza tada će opadanje krenuti odmah sa prelazom na članove y niza. Da bi bitonička struktura radila korektno neophodno je da njeni ulazi zbirno gledano čine kružni bitonički niz. Kružni bitonički niz je niz dobijen od bitoničkog niza rotiranjem za k mesta ($0 \leq k \leq N-1$). Očigledno, bitonički niz je ujedno i kružni bitonički niz (rotiranje za 0 mesta bitoničkog niza). Rastući i opadajući nizovi su takođe kružni bitonički nizovi. Dokazano je da ulazi u bitoničku strukturu dimenzije N/2 takođe čine kružni bitonički niz (ako je ulaz u bitoničku strukturu dimenzije N bitonički), čime je omogućeno rekurzivno formiranje bitoničke strukture po principu prikazanom sa slike 3.1.1.

Principska rekurzivna šeme za spajanje dve sortirane liste od N/2 članova u jednu sortiranu listu od N članova su prikazana je na slici 3.1.1 za bitoničko spajanje. [2]



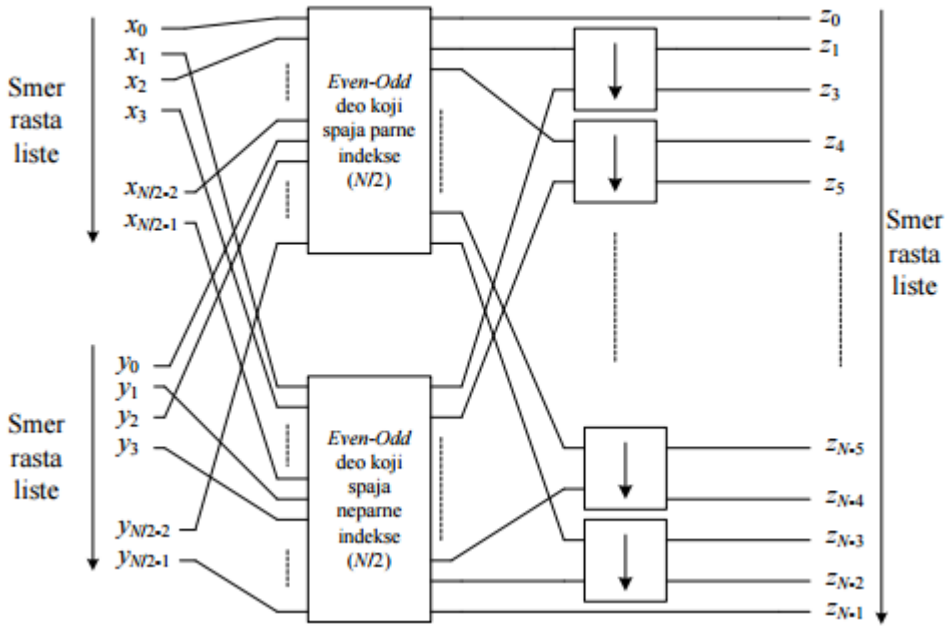
Slika 3.1.1. Principska rekurzivna šema za bitoničko spajanje [2]

.

3.2. Par – nepar struktura

Par-nepar spajanje listi tj. nizova je nešto intuitivnije. Naime, vrši se učešljavanje članova niza tako što se parni indeksi šalju u gornju strukturu dimenzije N/2, a neparni u donju strukturu dimenzije N/2. Svaka od struktura izvrši sortiranje članova i potom se vrši poređenje članova na izlazu. Najniži član sortirane liste gornje strukture se ne poredi jer je on sigurno najmanji (najmanji članovi obe liste su završili u gornjoj strukturi), a isto važi i za najveći clan sortirane liste donje strukture (najveći članovi obe liste su završili u donjoj strukturi). Preostali članovi se redom porede i dodatno preuređuju za jedno mesto na svičevima za poređenje koji se nalaze na izlazima. Na taj način se dobija kompletno sortirana lista u rastućem redosledu na izlazu.

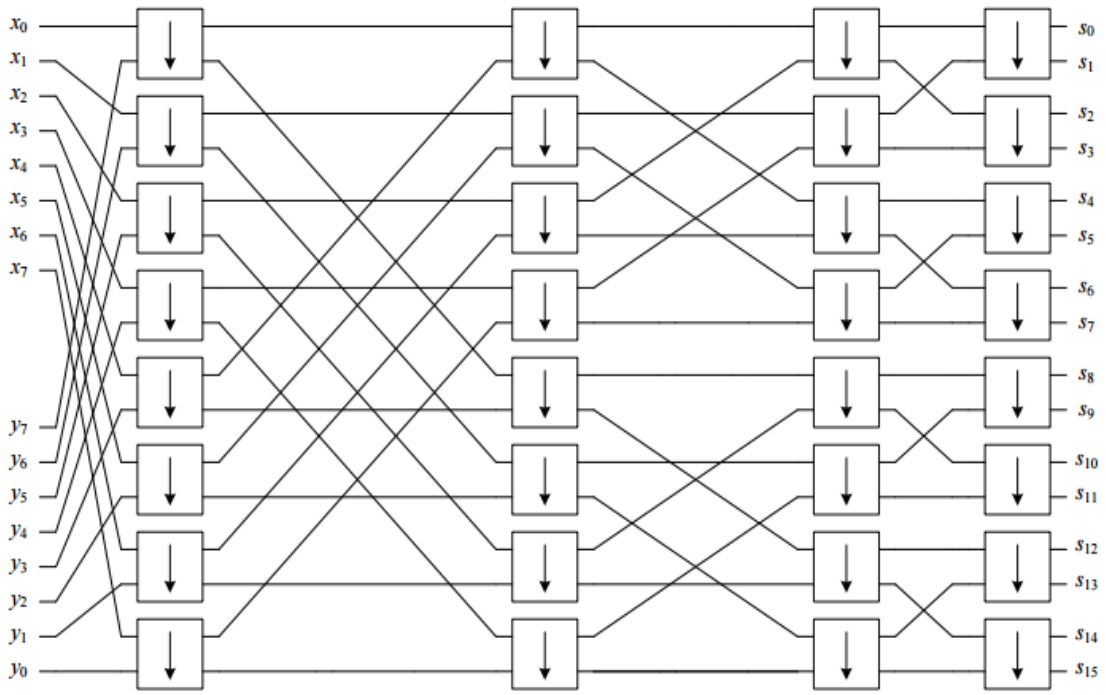
Principska rekurzivna šeme za spajanje dve sortirane liste od N/2 članova u jednu sortiranu listu od N članova su prikazana je na slici 3.1.2 za par-nepar spajanje.



Slika 3.2.1 Principska rekurzivna šema za par-nepar spajanje. [2]

Napomenimo da struktura za bitoničko povezivanje ima šemu povezivanja identičnu kao n-cube struktura. Na slici 3.2.2 je prikazane struktura za spajanje dve sortirane liste dimenzije 8 u jednu sortiranu listu dimenzije 16 struktura koje su dobijene rekurzivnim principom i može se videti da je to ustvari n-cube struktura. Upotrebom svič elemenata sa suprotnim prosleđivanjem (svič element sa strelicom usmerenom nagore) dobija se opadajuća (sortirana) spojena lista na izlazu. Pri tome, u slučaju par-nepar spajanja se moraju obrnuti smerovi lista na ulazu (što je logično jer sada najveći elementi obeju listi moraju završiti u gornjoj par-nepar strukturi koja će sada da prima neparne indekse, a najmanji elementi obeju listi moraju završiti u donjoj par-nepar strukturi koja će sada da prima parne indekse). Upravo u tu svrhu se koriste takvi svičevi (sa strelicom usmerenom nagore) u pojedinim bitoničkim strukturama u Batcher sorteru da bi se formirale opadajuće liste u skladu sa zahtevom bitoničke strukture - da jedna lista bude rastuća, a druga opadajuća na njenom ulazu.

Par-nepar struktura ima manje svič elemenata, dok bitonička struktura ima jednak broj svič elemenata u svim kaskadama (kolonama). Strukture za sortiranje mogu da poprave performanse Banyan komutatora (pod uslovom da se koristi omega ili n-cube struktura), ali unose dodatne kaskade, tj. troše dodatne hardverske resurse. Broj dodatnih kaskada koje unosi struktura za sortiranje je jednaka $1+2+...+\log_2 N=[\log_2 N\cdot(1+\log_2 N)]/2$, što je za veliko N ipak relativno velik broj kaskada. Otuda se koriste i druge tehnike za unapređivanje performansi Banyan komutatora i one se tipično zasnivaju na umnožavanju Banyan struktura tako što se kreira više putanja između proizvoljnog ulaznog i proizvoljnog izlaznog porta čime se smanjuje verovatnoća interne blokade (ili se u potpunosti ukida problem interne blokade). Umnoženi Banyan komutatori spadaju u grupu komutatora sa višestrukim putanjama, za razliku od Banyan komutatora koji spadaju u grupu komutatora sa jednostrukim putanjama. [2]



Slika 3.2.2 Struktura za bitoničko spajanje odgovara n-cube strukturi. [2]

4. IMPLEMENTACIJA STRUKTURA ZA SORTIRANJE

U ovom poglavlju je objašnjena implementacija algoritama za bitoničko i par-nepar sortiranje. Urađeno je nekoliko verzija za svaki algoritam zavisno od broja ulaza odnosno izlaza. Takođe, svaki od ovih tipova je implementiran pomoću kombinacione logike i pomoću registara, radi postizanja veće brzine rada. Dakle, implementirane i testirane su sledeće varijante:

1. bitonički 8x8 pomoću kombinacione logike
2. bitonički 8x8 sa registrima
3. bitonički 16x16 pomoću kombinacione logike
4. bitonički 16x16 sa registrima
5. bitonički 32x32 pomoću kombinacione logike
6. bitonički 32x32 sa registrima
7. par-nepar 8x8 pomoću kombinacione logike
8. par-nepar 8x8 sa registrima
9. par-nepar 16x16 pomoću kombinacione logike
10. par-nepar 16x16 sa registrima
11. par-nepar 32x32 pomoću kombinacione logike
12. par-nepar 32x32 sa registrima

Lako se mogu kreirati i implementacije za duže nizove na osnovu varijanti predstavljenih u ovoj tezi.

Od ulaznih i izlaznih signala u svakoj varijanti ima 8/16/32 ulaznih i isti broj izlaznih interfejsa, kao i clk signal takta. Kod implementacija sa registrima, postoje još ulazni start signal, koji signalizira dolazak podataka za obradu, kao i izlazni flag signal koji signalizira dolazak paketa na izlaz. Treba napomenuti da su algoritmi testirani za različite dužine ulaznih magistrala 1b ,8b, 16b, 32b, 64b, 128b, 256b.

Osnovna komponenta svih algoritama koji su realizovani pomoću kombinacione logike je entitet **cmpr** koji vrši funkciju komparatora. Ima dva signala na ulazu koje međusobno poredi i na prvi izlaz šalje manji, a na drugi izlaz veći signal. Ovo je osnovni entitet čijim kombinovanjem se realizuju varijante sa 4, 8, 16, 32 ulaza i izlaza. Ovakvo rešenje podrazumeva da se odredišna aderesa stavlja na najviše pozicije vektora, pri čemu pošto se porede vrednosti kompletnog vektora, nije bitno kolika je dužina te informacije tj. odredišne adrese, pa je rešenje validno kao osnovna gradivna jedinica za proizvoljnu dimenziju sortera.

```
entity cmpr is

generic
(
	n:integer:=16
);

port
(
	x1,x2:in std_logic_vector(n-1 downto 0);
	y1,y2:out std_logic_vector(n-1 downto 0)
);
end cmpr;

architecture Behavioral of cmpr is
```

```
begin
            process(x1,x2)
begin
                  if(x1<x2)then
                        y1<=x1;
                        y2<=x2;
                  else
                        y1<=x2;
                        y2<=x1;
                  end if;
            end process;
end Behavioral;
```

4.1. Bitonička struktura

Entitet sa 4 ulaza i izlaza koji izvršava bitonički algoritam se realizuje korišćenjem 4 komparatora. Stoga je u ovoj realizaciji **cmpr** korišćen kao komponenta, i to dva komparatora u prvoj kaskadi i dva u drugoj, koji su međusobno povezani preko odgovarajućih signala. Podrazumijeva se da na prva dva ulazna interfejsa dolaze signali sortirani u rastućem rasporedu, a na druga dva poređani u opadajućem redosledu. Na ulaz prvog komparatora u prvoj kaskadi idu signali sa prvog i trećeg interfejsa, a na drugi komparator u istoj kaskadi signali sa drugog i četvrtog. Na prvi komprator u drugoj kaskadi dolaze signali sa prvih izlaza komparatora u prvoj kaskadi, a druga dva signala na drugi komparator u drugoj kaskadi.

```
entity bitonic_4x4 is
generic
(
      n:integer:=16
);
port
(
      x1,x2,x3,x4:in std_logic_vector(n-1 downto 0);
      y1,y2,y3,y4:out std_logic_vector(n-1 downto 0)
);
end bitonic_4x4;

architecture Behavioral of bitonic_4x4 is
      component cmpr is
      generic
      (
            n:integer:=16
      );
      port
      (
            x1,x2:in std_logic_vector(n-1 downto 0);
            y1,y2:out std_logic_vector(n-1 downto 0)
      );
      end component;
      signal y11,y12,y13,y14:std_logic_vector(n-1 downto 0);


begin
      ulaz1:cmpr
      generic map
      (
            n=>n
```

```
	)
	port map
	(
		x1=>x1,
		x2=>x3,
		y1=>y11,
		y2=>y13
	);

	ulaz2:cmpr
	generic map
	(
		n=>n
	)
	port map
	(
		x1=>x2,
		x2=>x4,
		y1=>y12,
		y2=>y14
	);

	izlaz1:cmpr
	generic map
	(
		n=>n
	)
	port map
	(
		x1=>y11,
		x2=>y12,
		y1=>y1,
		y2=>y2
	);

	izlaz2:cmpr
	generic map
	(
		n=>n
	)
	port map
	(
		x1=>y13,
		x2=>y14,
		y1=>y3,
		y2=>y4
	);
end Behavioral;
```

Na sličan način je realizovan i entitet sa 8 ulaza/izlaza. Ovakav entitet koji funkcioniše sa delimično sortiranim listama na ulazu je neophodan za realizaciju entiteta koji radi sa nesortiranim podacima. Pored komparatora, on se sastoji i od bloka za bitoničko sortiranje sa 4 ulaza/izlaza. Princip je sličan, samo što je sada u prvoj kaskadi n/2, tj. 4 komparatora, a u drugoj dva 4x4 bloka.

```
entity bitonic_8x8 is

generic
(
	n:integer:=16
```

```
);
port
(
	clk:in std_logic;
	x1,x2,x3,x4,x5,x6,x7,x8:in std_logic_vector(n-1 downto 0);
	y1,y2,y3,y4,y5,y6,y7,y8:out std_logic_vector(n-1 downto 0)
);
end bitonic_8x8;

architecture Behavioral of bitonic_8x8 is
	component cmpr is
	generic
	(
		n:integer:=16
	);
	port
	(
		x1,x2:in std_logic_vector(n-1 downto 0);
		y1,y2:out std_logic_vector(n-1 downto 0)
	);
	end component;

	component bitonic_4x4 is
	generic
	(
		n:integer:=16
	);
	port
	(
		x1,x2,x3,x4:in std_logic_vector(n-1 downto 0);
		y1,y2,y3,y4:out std_logic_vector(n-1 downto 0)
	);
	end component;

	signal xx1,xx2,xx3,xx4,xx5,xx6,xx7,xx8:std_logic_vector(n-1 downto 0);
	signal z1,z2,z3,z4,z5,z6,z7,z8:std_logic_vector(n-1 downto 0);
	signal y11,y12,y13,y14,y15,y16,y17,y18:std_logic_vector(n-1 downto 0);
begin
	process(clk)
	begin
	if(clk'EVENT and clk='1')then
		xx1<=x1;
		xx2<=x2;
		xx3<=x3;
		xx4<=x4;
		xx5<=x5;
		xx6<=x6;
		xx7<=x7;
		xx8<=x8;
		y1<=z1;
		y2<=z2;
		y3<=z3;
		y4<=z4;
		y5<=z5;
		y6<=z6;
		y7<=z7;
		y8<=z8;
	end if;
	end process;
```

```
	ulaz1:cmpr

generic map
	(
		n=>n
	)
	port map
	(
		x1=>xx1,
		x2=>xx5,
		y1=>y11,
		y2=>y15
	);

	ulaz2:cmpr
	generic map
	(
		n=>n
	)
	port map
	(
		x1=>xx2,
		x2=>xx6,
		y1=>y12,
		y2=>y16
	);

	ulaz3:cmpr
	generic map
	(
		n=>n
	)
	port map
	(
		x1=>xx3,
		x2=>xx7,
		y1=>y13,
		y2=>y17
	);

	ulaz4:cmpr
	generic map
	(
		n=>n
	)
	port map
	(
		x1=>xx4,
		x2=>xx8,
		y1=>y14,
		y2=>y18
	);

	izlaz1:bitonic_4x4
	generic map
	(
		n=>n
	)
	port map
	(
```

```
            x1=>y11,
            x2=>y12,
            x3=>y13,
            x4=>y14,
            y1=>z1,
            y2=>z2,
            y3=>z3,
            y4=>z4
      );

      izlaz2:bitonic_4x4
      generic map
      (
            n=>n
      )
      port map
      (
            x1=>y15,
            x2=>y16,
            x3=>y17,
            x4=>y18,
            y1=>z5,
            y2=>z6,
            y3=>z7,
            y4=>z8
      );

end Behavioral;
```

Realizacija entiteta koji sortira potpuno neuređene liste je nešto komplikovanija. Od komponenti sadrži **cmpr, bitonic4x4 i bitonic8x8** entitete. U prvoj kaskadi je takođe 4 komparatora. Oni sortiraju 2 po 2 ulazna signala. Zatim, u drugoj kaskadi je blok 4x4. Kao što je već rečeno, na prva dva ulaza mu se šalju signali sortirani po rastućem redosledu, a na druga dva po opadajućem. Dakle, na prva dva ulaza se šalju signali sa izlaza prvog komparatora, a na druga dva signali sa drugog komparatora, ali u obrnutom redosledu (na treći ulaz ide signal sa drugog izlaza, a na četvrti sa prvog). Slično se i drugi 4x4 blok veže sa trećim i četvrtim komparatorom. U poslednjoj kaskadi je 8x8 blok, čiji su ulazni interfejsi vezani sa izlazima blokova 4x4 u središnjoj kaskadi. Opisana realizacija je kombinaciona i process je kreiran unutar arhitekture entiteta koji sortira potpuno neuređene liste, vrši sinhronizaciju ulaza i izlaza tog kombinacionog sortera na signal takta, tako da je entitet **bitonic_8x8_nesort** sinhronizovan na takt.

```
entity bitonic_8x8_nesort is
generic
(
      n:integer:=16
);
port
(
      clk:in std_logic;
      x1,x2,x3,x4,x5,x6,x7,x8:in std_logic_vector(n-1 downto 0);
      y1,y2,y3,y4,y5,y6,y7,y8:out std_logic_vector(n-1 downto 0)
);
end bitonic_8x8_nesort;

architecture Behavioral of bitonic_8x8_nesort is
      component cmpr is
      generic
      (
```

```
            n:integer:=16
      );
port
      (
            x1,x2:in std_logic_vector(n-1 downto 0);
            y1,y2:out std_logic_vector(n-1 downto 0)
      );
      end component;

      component bitonic_4x4 is
      generic
      (
            n:integer:=16
      );
      port
      (
            x1,x2,x3,x4:in std_logic_vector(n-1 downto 0);
            y1,y2,y3,y4:out std_logic_vector(n-1 downto 0)
      );
      end component;

      component bitonic_8x8 is
      generic
      (
            n:integer:=16
      );
      port
      (
            clk:in std_logic;
            x1,x2,x3,x4,x5,x6,x7,x8:in std_logic_vector(n-1 downto 0);
            y1,y2,y3,y4,y5,y6,y7,y8:out std_logic_vector(n-1 downto 0)
      );
      end component;

      signal xx1,xx2,xx3,xx4,xx5,xx6,xx7,xx8:std_logic_vector(n-1 downto 0);
      signal z1,z2,z3,z4,z5,z6,z7,z8:std_logic_vector(n-1 downto 0);
      signal
y11,y12,y13,y14,y15,y16,y17,y18,y21,y22,y23,y24,y25,y26,y27,y28:std_logic_vector
(n-1 downto 0);

begin
      process(clk)
      begin
      if(clk'EVENT and clk='1')then
            xx1<=x1;
            xx2<=x2;
            xx3<=x3;
            xx4<=x4;
            xx5<=x5;
            xx6<=x6;
            xx7<=x7;
            xx8<=x8;
            y1<=z1;
            y2<=z2;
            y3<=z3;
            y4<=z4;
            y5<=z5;
            y6<=z6;
            y7<=z7;
            y8<=z8;
```

```
	end if;
	end process;
ulaz1:cmpr
	generic map
	(
		n=>n
	)
	port map
	(
		x1=>xx1,
		x2=>xx2,
		y1=>y11,
		y2=>y12
	);

	ulaz2:cmpr
	generic map
	(
		n=>n
	)
	port map
	(
		x1=>xx3,
		x2=>xx4,
		y1=>y13,
		y2=>y14
	);

	ulaz3:cmpr
	generic map
	(
		n=>n
	)
	port map
	(
		x1=>xx5,
		x2=>xx6,
		y1=>y15,
		y2=>y16
	);

	ulaz4:cmpr
	generic map
	(
		n=>n
	)
	port map
	(
		x1=>xx7,
		x2=>xx8,
		y1=>y17,
		y2=>y18
	);

	sredina1:bitonic_4x4
	generic map
	(
		n=>n
	)
	port map
```

```
	(
		x1=>y11,
		x2=>y12,
		x3=>y14,
		x4=>y13,
		y1=>y21,
		y2=>y22,
		y3=>y23,
		y4=>y24
	);

	sredina2:bitonic_4x4
	generic map
	(
		n=>n
	)
	port map
	(
		x1=>y15,
		x2=>y16,
		x3=>y18,
		x4=>y17,
		y1=>y25,
		y2=>y26,
		y3=>y27,
		y4=>y28
	);

	izlaz1:bitonic_8x8
	generic map
	(
		n=>n
	)
	port map
	(
		clk=>clk,
		x1=>y21,
		x2=>y22,
		x3=>y23,
		x4=>y24,
		x5=>y28,
		x6=>y27,
		x7=>y26,
		x8=>y25,
		y1=>z1,
		y2=>z2,
		y3=>z3,
		y4=>z4,
		y5=>z5,
		y6=>z6,
		y7=>z7,
		y8=>z8
	);

end Behavioral;
```

Na sličan način su realizovane i strukture sa 16 odnosno 32 ulaza/izlaza. Takođe su prvo napravljeni entiteti koji rade sa delimično uređenim listama i zatim ugrađeni kao komponenta u entitet

sa neuređenim listama na ulazu. Realizacija je nešto jednostavnija jer se koristi prethodno urađena struktura sa 8 ulaza i izlaza, pa postoje samo dve kaskade.

```
entity bitonic_16x16_nesort is
generic
(
	n:integer:=16
);
port
(
	clk:in std_logic;
	x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16:in
std_logic_vector(n-1 downto 0);
	y1,y2,y3,y4,y5,y6,y7,y8,y9,y10,y11,y12,y13,y14,y15,y16:out
std_logic_vector(n-1 downto 0)
);
end bitonic_16x16_nesort;

architecture Behavioral of bitonic_16x16_nesort is

	component bitonic_8x8_nesort is
	generic
	(
		n:integer:=16
	);
	port
	(
		clk:in std_logic;
		x1,x2,x3,x4,x5,x6,x7,x8:in std_logic_vector(n-1 downto 0);
		y1,y2,y3,y4,y5,y6,y7,y8:out std_logic_vector(n-1 downto 0)
	);
	end component;

	component bitonic_16x16 is
	generic
	(
		n:integer:=32
	);
	port
	(
		clk:in std_logic;
		x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16:in
std_logic_vector(n-1 downto 0);
		y1,y2,y3,y4,y5,y6,y7,y8,y9,y10,y11,y12,y13,y14,y15,y16:out
std_logic_vector(n-1 downto 0)
	);
	end component;

	signal
xx1,xx2,xx3,xx4,xx5,xx6,xx7,xx8,xx9,xx10,xx11,xx12,xx13,xx14,xx15,xx16:std_logic
_vector(n-1 downto 0);
	signal
z1,z2,z3,z4,z5,z6,z7,z8,z9,z10,z11,z12,z13,z14,z15,z16:std_logic_vector(n-1
downto 0);
	signal
yy1,yy2,yy3,yy4,yy5,yy6,yy7,yy8,yy9,yy10,yy11,yy12,yy13,yy14,yy15,yy16:std_logic
_vector(n-1 downto 0);

begin
	process(clk)
```

```
begin
if(clk'EVENT and clk='1')then
	xx1<=x1;
	xx2<=x2;
	xx3<=x3;
	xx4<=x4;
	xx5<=x5;
	xx6<=x6;
	xx7<=x7;
	xx8<=x8;
	xx9<=x9;
	xx10<=x10;
	xx11<=x11;
	xx12<=x12;
	xx13<=x13;
	xx14<=x14;
	xx15<=x15;
	xx16<=x16;
	y1<=z1;
	y2<=z2;
	y3<=z3;
	y4<=z4;
	y5<=z5;
	y6<=z6;
	y7<=z7;
	y8<=z8;
	y9<=z9;
	y10<=z10;
	y11<=z11;
	y12<=z12;
	y13<=z13;
	y14<=z14;
	y15<=z15;
	y16<=z16;
end if;
end process;

ulaz1:bitonic_8x8_nesort
generic map
(
	n=>n
)
port map
(
	clk=>clk,
	x1=>xx1,
	x2=>xx2,
	x3=>xx3,
	x4=>xx4,
	x5=>xx5,
	x6=>xx6,
	x7=>xx7,
	x8=>xx8,
	y1=>yy1,
	y2=>yy2,
	y3=>yy3,
	y4=>yy4,
	y5=>yy5,
	y6=>yy6,
	y7=>yy7,
```

```
      y8=>yy8
);

ulaz2:bitonic_8x8_nesort
generic map
(
      n=>n
)
port map
(
      clk=>clk,
      x1=>xx9,
      x2=>xx10,
      x3=>xx11,
      x4=>xx12,
      x5=>xx13,
      x6=>xx14,
      x7=>xx15,
      x8=>xx16,
      y1=>yy9,
      y2=>yy10,
      y3=>yy11,
      y4=>yy12,
      y5=>yy13,
      y6=>yy14,
      y7=>yy15,
      y8=>yy16
);

izlaz1:bitonic_16x16
generic map
(
      n=>n
)
port map
(
      clk=>clk,
      x1=>yy1,
      x2=>yy2,
      x3=>yy3,
      x4=>yy4,
      x5=>yy5,
      x6=>yy6,
      x7=>yy7,
      x8=>yy8,
      x9=>yy16,
      x10=>yy15,
      x11=>yy14,
      x12=>yy13,
      x13=>yy12,
      x14=>yy11,
      x15=>yy10,
      x16=>yy9,
      y1=>z1,
      y2=>z2,
      y3=>z3,
      y4=>z4,
      y5=>z5,
      y6=>z6,
      y7=>z7,
```

```
            y8=>z8,
            y9=>z9,
            y10=>z10,
            y11=>z11,
            y12=>z12,
            y13=>z13,
            y14=>z14,
            y15=>z15,
            y16=>z16
      );

end Behavioral;
entity bitonic_32x32_nesort is
generic
(
      n:integer:=32
);
port
(
      clk:in std_logic;
      x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17,x18,x19,x20,x21
,x22,x23,x24,x25,x26,x27,x28,x29,x30,x31,x32:in std_logic_vector(n-1 downto 0);
      y1,y2,y3,y4,y5,y6,y7,y8,y9,y10,y11,y12,y13,y14,y15,y16,y17,y18,y19,y20,y21
,y22,y23,y24,y25,y26,y27,y28,y29,y30,y31,y32:out std_logic_vector(n-1 downto 0)
);
end bitonic_32x32_nesort;

architecture Behavioral of bitonic_32x32_nesort is

      component bitonic_16x16_nesort is
      generic
      (
            n:integer:=32
      );
      port
      (
            clk:in std_logic;
            x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16:in
std_logic_vector(n-1 downto 0);
            y1,y2,y3,y4,y5,y6,y7,y8,y9,y10,y11,y12,y13,y14,y15,y16:out
std_logic_vector(n-1 downto 0)
      );
      end component;

      component bitonic_32x32 is
      generic
      (
            n:integer:=32
      );
      port
      (
            clk:in std_logic;

      x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17,x18,x19,x20,x21
,x22,x23,x24,x25,x26,x27,x28,x29,x30,x31,x32:in std_logic_vector(n-1 downto 0);

      y1,y2,y3,y4,y5,y6,y7,y8,y9,y10,y11,y12,y13,y14,y15,y16,y17,y18,y19,y20,y21
,y22,y23,y24,y25,y26,y27,y28,y29,y30,y31,y32:out std_logic_vector(n-1 downto 0)
      );
      end component;
```

```
signal
xx1,xx2,xx3,xx4,xx5,xx6,xx7,xx8,xx9,xx10,xx11,xx12,xx13,xx14,xx15,xx16,xx17,xx18
,xx19,xx20,xx21,xx22,xx23,xx24,xx25,xx26,xx27,xx28,xx29,xx30,xx31,xx32:std_logic
_vector(n-1 downto 0);
      signal
z1,z2,z3,z4,z5,z6,z7,z8,z9,z10,z11,z12,z13,z14,z15,z16,z17,z18,z19,z20,z21,z22,z
23,z24,z25,z26,z27,z28,z29,z30,z31,z32:std_logic_vector(n-1 downto 0);
      signal
yy1,yy2,yy3,yy4,yy5,yy6,yy7,yy8,yy9,yy10,yy11,yy12,yy13,yy14,yy15,yy16,yy17,yy18
,yy19,yy20,yy21,yy22,yy23,yy24,yy25,yy26,yy27,yy28,yy29,yy30,yy31,yy32:std_logic
_vector(n-1 downto 0);

begin
      process(clk)
      begin
      if(clk'EVENT and clk='1')then
            xx1<=x1;
            xx2<=x2;
            xx3<=x3;
            xx4<=x4;
            xx5<=x5;
            xx6<=x6;
            xx7<=x7;
            xx8<=x8;
            xx9<=x9;
            xx10<=x10;
            xx11<=x11;
            xx12<=x12;
            xx13<=x13;
            xx14<=x14;
            xx15<=x15;
            xx16<=x16;
            xx17<=x17;
            xx18<=x18;
            xx19<=x19;
            xx20<=x20;
            xx21<=x21;
            xx22<=x22;
            xx23<=x23;
            xx24<=x24;
            xx25<=x25;
            xx26<=x26;
            xx27<=x27;
            xx28<=x28;
            xx29<=x29;
            xx30<=x30;
            xx31<=x31;
            xx32<=x32;
            y1<=z1;
            y2<=z2;
            y3<=z3;
            y4<=z4;
            y5<=z5;
            y6<=z6;
            y7<=z7;
            y8<=z8;
            y9<=z9;
            y10<=z10;
            y11<=z11;
```

```
        y12<=z12;
        y13<=z13;
        y14<=z14;
        y15<=z15;
        y16<=z16;
        y17<=z17;
        y18<=z18;
        y19<=z19;
        y20<=z20;
        y21<=z21;
        y22<=z22;
        y23<=z23;
        y24<=z24;
        y25<=z25;
        y26<=z26;
        y27<=z27;
        y28<=z28;
        y29<=z29;
        y30<=z30;
        y31<=z31;
        y32<=z32;
end if;
end process;

ulaz1:bitonic_16x16_nesort
generic map
(
        n=>n
)
port map
(
        clk=>clk,
        x1=>xx1,
        x2=>xx2,
        x3=>xx3,
        x4=>xx4,
        x5=>xx5,
        x6=>xx6,
        x7=>xx7,
        x8=>xx8,
        x9=>xx9,
        x10=>xx10,
        x11=>xx11,
        x12=>xx12,
        x13=>xx13,
        x14=>xx14,
        x15=>xx15,
        x16=>xx16,
        y1=>yy1,
        y2=>yy2,
        y3=>yy3,
        y4=>yy4,
        y5=>yy5,
        y6=>yy6,
        y7=>yy7,
        y8=>yy8,
        y9=>yy9,
        y10=>yy10,
        y11=>yy11,
        y12=>yy12,
```

```
        y13=>yy13,
        y14=>yy14,
        y15=>yy15,
        y16=>yy16
);

ulaz2:bitonic_16x16_nesort
generic map
(
        n=>n
)
port map
(
        clk=>clk,
        x1=>xx17,
        x2=>xx18,
        x3=>xx19,
        x4=>xx20,
        x5=>xx21,
        x6=>xx22,
        x7=>xx23,
        x8=>xx24,
        x9=>xx25,
        x10=>xx26,
        x11=>xx27,
        x12=>xx28,
        x13=>xx29,
        x14=>xx30,
        x15=>xx31,
        x16=>xx32,
        y1=>yy17,
        y2=>yy18,
        y3=>yy19,
        y4=>yy20,
        y5=>yy21,
        y6=>yy22,
        y7=>yy23,
        y8=>yy24,
        y9=>yy25,
        y10=>yy26,
        y11=>yy27,
        y12=>yy28,
        y13=>yy29,
        y14=>yy30,
        y15=>yy31,
        y16=>yy32
);

izlaz1:bitonic_32x32
generic map
(
        n=>n
)
port map
(
        clk=>clk,
        x1=>yy1,
        x2=>yy2,
        x3=>yy3,
        x4=>yy4,
```

```
x5=>yy5,
x6=>yy6,
x7=>yy7,
x8=>yy8,
x9=>yy9,
x10=>yy10,
x11=>yy11,
x12=>yy12,
x13=>yy13,
x14=>yy14,
x15=>yy15,
x16=>yy16,
x17=>yy32,
x18=>yy31,
x19=>yy30,
x20=>yy29,
x21=>yy28,
x22=>yy27,
x23=>yy26,
x24=>yy25,
x25=>yy24,
x26=>yy23,
x27=>yy22,
x28=>yy21,
x29=>yy20,
x30=>yy19,
x31=>yy18,
x32=>yy17,
y1=>z1,
y2=>z2,
y3=>z3,
y4=>z4,
y5=>z5,
y6=>z6,
y7=>z7,
y8=>z8,
y9=>z9,
y10=>z10,
y11=>z11,
y12=>z12,
y13=>z13,
y14=>z14,
y15=>z15,
y16=>z16,
y17=>z17,
y18=>z18,
y19=>z19,
y20=>z20,
y21=>z21,
y22=>z22,
y23=>z23,
y24=>z24,
y25=>z25,
y26=>z26,
y27=>z27,
y28=>z28,
y29=>z29,
y30=>z30,
y31=>z31,
y32=>z32
```

```
	);
end Behavioral;
```

4.2. Par – nepar struktura

Implementacija algoriatama za par-nepar sortiranje je slična. I tu je gradivni element komparator, koji se dalje koristi za složenije strukture sa više ulaza i izlaza. Kod entiteta sa 4 ulaza/izlaza, u prvoj kaskadi su dva komparatora. Na ulaz ovog bloka dolaze signali sortirani u dve rastuće liste. Na ulaz prvog komparatora idu prvi i treći signal, a na ulaz drugog, drugi i četvrti. Prvi izlaz prvog komparatora se šalje na prvi izlaz, dok se drugi izlaz drugog komparatora povezuje na poslednji izlaz. Preostala dva signala se dovode na ulaz komparatora u drugoj kaskadi, i dalje na presotala dva izlazna interfejsa.

```
entity even_odd_4x4 is
generic
(
	n:integer:=8
);
port
(
	x1,x2,x3,x4:in std_logic_vector(n-1 downto 0);
	y1,y2,y3,y4:out std_logic_vector(n-1 downto 0)
);
end even_odd_4x4;

architecture Behavioral of even_odd_4x4 is
	component cmpr is
	generic
	(
		n:integer:=16
	);

	port
	(
		x1,x2:in std_logic_vector(n-1 downto 0);
		y1,y2:out std_logic_vector(n-1 downto 0)
	);
	end component;
	signal y12,y13:std_logic_vector(n-1 downto 0);
begin
	ulaz1:cmpr
	generic map
	(
		n=>n
	)
	port map
	(
		x1=>x1,
		x2=>x3,
		y1=>y1,
		y2=>y12
	);

	ulaz2:cmpr
	generic map
	(
```

```
n=>n
	)
	port map
	(
		x1=>x2,
		x2=>x4,
		y1=>y13,
		y2=>y4
	);

	izlaz1:cmpr
	generic map
	(
		n=>n
	)
	port map
	(
		x1=>y12,
		x2=>y13,
		y1=>y2,
		y2=>y3
	);
end Behavioral;
```

Entitet 8x8 koji radi sa delimično uređenim listama koristi kao komponente entitet 4x4 i komparator. Ovaj entitet se kasnije koristi kao komponenta u realizaciji strukture koja obrađuje potpune neuređene liste. U prvoj kaskadi su dva 4x4 bloka. Na ulaze prvog se dovode signali sa parnim indeksima, a na ulaze drugog sa neparnim. Opet se signal sa prvog interfejsa prvog 4x4 bloka šalje na prvi izlazni interfejs, kao i poslednji signal drugog 4x4 bloka na poslednji izlazni interfejs. U drugoj kaskadi su tri komparatora, koji vrše sortiranje preostalih signala.

```
entity even_odd_8x8 is
generic
(
	n:integer:=8
);
port
(
	clk:in std_logic;
	x1,x2,x3,x4,x5,x6,x7,x8:in std_logic_vector(n-1 downto 0);
	y1,y2,y3,y4,y5,y6,y7,y8:out std_logic_vector(n-1 downto 0)
);
end even_odd_8x8;

architecture Behavioral of even_odd_8x8 is
	component cmpr is
	generic
	(
		n:integer:=16
	);

	port
	(
		x1,x2:in std_logic_vector(n-1 downto 0);
		y1,y2:out std_logic_vector(n-1 downto 0)
	);
	end component;

	component even_odd_4x4 is
generic
```

```
	(
		n:integer:=8
	);
	port
	(
		x1,x2,x3,x4:in std_logic_vector(n-1 downto 0);
		y1,y2,y3,y4:out std_logic_vector(n-1 downto 0)
	);
	end component;
	signal xx1,xx2,xx3,xx4,xx5,xx6,xx7,xx8:std_logic_vector(n-1 downto 0);
	signal z1,z2,z3,z4,z5,z6,z7,z8:std_logic_vector(n-1 downto 0);
	signal y12,y13,y14,y15,y16,y17:std_logic_vector(n-1 downto 0);

begin
	process(clk)
	begin
	if(clk'EVENT and clk='1')then
		xx1<=x1;
		xx2<=x2;
		xx3<=x3;
		xx4<=x4;
		xx5<=x5;
		xx6<=x6;
		xx7<=x7;
		xx8<=x8;
		y1<=z1;
		y2<=z2;
		y3<=z3;
		y4<=z4;
		y5<=z5;
		y6<=z6;
		y7<=z7;
		y8<=z8;
	end if;
	end process;

	ulaz1:even_odd_4x4
	generic map
	(
		n=>n
	)
	port map
	(
		x1=>xx1,
		x2=>xx3,
		x3=>xx5,
		x4=>xx7,
		y1=>z1,
		y2=>y12,
		y3=>y14,
		y4=>y16
	);

	ulaz2:even_odd_4x4
	generic map
	(
		n=>n
	)
	port map
	(
```

```
            x1=>xx2,
            x2=>xx4,
            x3=>xx6,
            x4=>xx8,
            y1=>y13,
            y2=>y15,
            y3=>y17,
            y4=>z8
      );

      izlaz1:cmpr
      generic map
      (
            n=>n
      )
      port map
      (
            x1=>y12,
            x2=>y13,
            y1=>z2,
            y2=>z3
      );

      izlaz2:cmpr
      generic map
      (
            n=>n
      )
      port map
      (
            x1=>y14,
            x2=>y15,
            y1=>z4,
            y2=>z5
      );

      izlaz3:cmpr
      generic map
      (
            n=>n
      )
      port map
      (
            x1=>y16,
            x2=>y17,
            y1=>z6,
            y2=>z7
      );
end Behavioral;
```

Kod realizacije par-nepar algoritma sa potpuno neuređenim listama kao komponente se koriste **cmpr, even_odd_4x4 i even_odd_8x8** entiteti. U prvoj kaskadi je 4 komparatora koji sortiraju dva po dva ulazna signala. Zatim se signali sa njihovih izlaznih interfejsa šalju na ulazne interfejse **even_odd_4x4** blokova u drugoj kaskadi. Dalje se signali sa izlaza ovih blokova šalju na ulaz **even_odd_8x8** bloka u trećoj kaskadi.

```
entity even_odd_8x8_nesort is
generic
(
      n:integer:=16
```

```
);
port
(
	clk:in std_logic;
	x1,x2,x3,x4,x5,x6,x7,x8:in std_logic_vector(n-1 downto 0);
	y1,y2,y3,y4,y5,y6,y7,y8:out std_logic_vector(n-1 downto 0)
);
end even_odd_8x8_nesort;

architecture Behavioral of even_odd_8x8_nesort is
	component cmpr is
	generic
	(
		n:integer:=16
	);

	port
	(
		x1,x2:in std_logic_vector(n-1 downto 0);
		y1,y2:out std_logic_vector(n-1 downto 0)
	);
	end component;

	component even_odd_4x4 is
	generic
	(
		n:integer:=8
	);
	port
	(
		x1,x2,x3,x4:in std_logic_vector(n-1 downto 0);
		y1,y2,y3,y4:out std_logic_vector(n-1 downto 0)
	);
	end component;

	component even_odd_8x8 is
	generic
	(
		n:integer:=8
	);
	port
	(
		clk:in std_logic;
		x1,x2,x3,x4,x5,x6,x7,x8:in std_logic_vector(n-1 downto 0);
		y1,y2,y3,y4,y5,y6,y7,y8:out std_logic_vector(n-1 downto 0)
	);
	end component;

	signal xx1,xx2,xx3,xx4,xx5,xx6,xx7,xx8:std_logic_vector(n-1 downto 0);
	signal z1,z2,z3,z4,z5,z6,z7,z8:std_logic_vector(n-1 downto 0);
	signal
y11,y12,y13,y14,y15,y16,y17,y18,y21,y22,y23,y24,y25,y26,y27,y28:std_logic_vector
(n-1 downto 0);

begin
	process(clk)
	begin
	if(clk'EVENT and clk='1')then
		xx1<=x1;
		xx2<=x2;
```

```
        xx3<=x3;
        xx4<=x4;
        xx5<=x5;
        xx6<=x6;
        xx7<=x7;
        xx8<=x8;
        y1<=z1;
        y2<=z2;
        y3<=z3;
        y4<=z4;
        y5<=z5;
        y6<=z6;
        y7<=z7;
        y8<=z8;
end if;
end process;

ulaz1:cmpr
generic map
(
        n=>n
)
port map
(
        x1=>xx1,
        x2=>xx2,
        y1=>y11,
        y2=>y12
);

ulaz2:cmpr
generic map
(
        n=>n
)
port map
(
        x1=>xx3,
        x2=>xx4,
        y1=>y13,
        y2=>y14
);

ulaz3:cmpr
generic map
(
        n=>n
)
port map
(
        x1=>xx5,
        x2=>xx6,
        y1=>y15,
        y2=>y16
);

ulaz4:cmpr
generic map
(
        n=>n
```

```
)
port map
(
      x1=>xx7,
      x2=>xx8,
      y1=>y17,
      y2=>y18
);

sredina1:even_odd_4x4
generic map
(
      n=>n
)
port map
(
      x1=>y11,
      x2=>y12,
      x3=>y13,
      x4=>y14,
      y1=>y21,
      y2=>y22,
      y3=>y23,
      y4=>y24
);

sredina2:even_odd_4x4
generic map
(
      n=>n
)
port map
(
      x1=>y15,
      x2=>y16,
      x3=>y17,
      x4=>y18,
      y1=>y25,
      y2=>y26,
      y3=>y27,
      y4=>y28
);

izlaz1:even_odd_8x8
generic map
(
      n=>n
)
port map
(
      clk=>clk,
      x1=>y21,
      x2=>y22,
      x3=>y23,
      x4=>y24,
      x5=>y25,
      x6=>y26,
      x7=>y27,
      x8=>y28,
      y1=>z1,
```

```
            y2=>z2,
            y3=>z3,
            y4=>z4,
            y5=>z5,
            y6=>z6,
            y7=>z7,
            y8=>z8
      );

end Behavioral
```

Na sličan način su implementirane varijante sa 16 odnosno 32 ulaza/izlaza.

Kod ovakve realizacije algoritama za sortiranje, novi podaci ne mogu biti obrađivani dok se prethodni ne proslede na izlaz entiteta. Kako se sve strukture sastoje od više kaskada, to znači da će u svakom taktu samo jedna kaskada biti aktivna, dok ostale neće raditi. Kako bi se ubrzao rad, realizovane su varijante sa registrima. Prednost ovakve realizacije je što je u svakom taktu svaka komponenta aktivna. Dakle, kad prva kaskada obradi podatke i prosledi ih drugoj, ona je spremna da prima nove podatke. Time se postiže znatno veća efikasnost. I kod ovih relizacija osnovni gradivni element je komparator, s tim što se od prethodno opisanog razlikuje jer ima dodatne ulazne i izlazne signale. To su *clk* signal takta, *start* signal koji signalizira dolazak paketa na ulaz i *flag* signal koji signalizira isporuku podataka na izlaznim interfejsima.

```
entity cmpr is

generic
(
      n:integer:=16
);

port
(
      clk: in std_logic;
      start: in std_logic;
      x1,x2:in std_logic_vector(n-1 downto 0);
      y1,y2:out std_logic_vector(n-1 downto 0);
      flag:out std_logic
);
end cmpr;

architecture Behavioral of cmpr is
begin
            process(x1,x2,clk)
            begin
                  if(clk'event and clk='1')then
                        flag<=start;
                        if(x1<x2)then
                              y1<=x1;
                              y2<=x2;
                        else
                              y1<=x2;
                              y2<=x1;
                        end if;
                  end if;
            end process;
end Behavioral;
```

I entitet sa 4 ulaza/izlaza je definisan slično kao onaj sa kombinacionom logikom, osim što ima dodatne *clk*, *start* i *flag* interfejse. *Flag* signali komporatora u prvoj kaskadi su mapirani na *start*

signale komparatora u drugoj kasadi. Dakle čim se u prvoj kaskadi obrade podaci prosleđuju se drugoj, koja ih dalje obrađuje, a prva kaskada je spremna za nove ulazne podatke. Treba opet napomenuti da je ovo entitet koji radi sa polusortiranim ulaznim podacima, koji služi kao komponenta u realizaciji entiteta koji sortira potpuno neuređene liste.

```
entity bitonic_8x8_reg is
generic
(
	n:integer:=16
);
port
(
	clk,start:in std_logic;
	x1,x2,x3,x4,x5,x6,x7,x8:in std_logic_vector(n-1 downto 0);
	y1,y2,y3,y4,y5,y6,y7,y8:out std_logic_vector(n-1 downto 0);
	flag:out std_logic
);
end bitonic_8x8_reg;

architecture Behavioral of bitonic_8x8_reg is
	component cmpr is
	generic
	(
		n:integer:=16
	);
	port
	(
		clk,start:in std_logic;
		x1,x2:in std_logic_vector(n-1 downto 0);
		y1,y2:out std_logic_vector(n-1 downto 0);
		flag:out std_logic
	);
	end component;

	component bitonic_4x4 is
	generic
	(
		n:integer:=16
	);
	port
	(
		clk,start:in std_logic;
		x1,x2,x3,x4:in std_logic_vector(n-1 downto 0);
		y1,y2,y3,y4:out std_logic_vector(n-1 downto 0);
		flag:out std_logic
	);
	end component;

	signal y11,y12,y13,y14,y15,y16,y17,y18:std_logic_vector(n-1 downto 0);
	signal f1,f2,f3,f4,f5,f6:std_logic;

begin
	ulaz1:cmpr
	generic map
	(
		n=>n
	)
	port map
	(
		clk=>clk,
```

```
        start=>start,
        x1=>x1,
        x2=>x5,
        y1=>y11,
        y2=>y15,
        flag=>f1
);

ulaz2:cmpr
generic map
(
        n=>n
)
port map
(
        clk=>clk,
        start=>start,
        x1=>x2,
        x2=>x6,
        y1=>y12,
        y2=>y16,
        flag=>f2
);

ulaz3:cmpr
generic map
(
        n=>n
)
port map
(
        clk=>clk,
        start=>start,
        x1=>x3,
        x2=>x7,
        y1=>y13,
        y2=>y17,
        flag=>f3
);

ulaz4:cmpr
generic map
(
        n=>n
)
port map
(
        clk=>clk,
        start=>start,
        x1=>x4,
        x2=>x8,
        y1=>y14,
        y2=>y18,
        flag=>f4
);

izlaz1:bitonic_4x4
generic map
(
        n=>n
```

```
	)
port map
	(
		clk=>clk,
		start=>f1,
		x1=>y11,
		x2=>y12,
		x3=>y14,
		x4=>y13,
		y1=>y1,
		y2=>y2,
		y3=>y3,
		y4=>y4,
		flag=>f5
	);

	izlaz2:bitonic_4x4
	generic map
	(
		n=>n
	)
	port map
	(
		clk=>clk,
		start=>f3,
		x1=>y15,
		x2=>y16,
		x3=>y18,
		x4=>y17,
		y1=>y5,
		y2=>y6,
		y3=>y7,
		y4=>y8,
		flag=>f6
	);

	flag<=f5 and f6;
end Behavioral;
```

Entitet za sortiranje neuređenih podataka je realizovan slično kao što je već opisano.

```
entity bitonic_8x8_nesort_reg is
generic
(
	n:integer:=16
);
port
(
	clk,start:in std_logic;
	x1,x2,x3,x4,x5,x6,x7,x8:in std_logic_vector(n-1 downto 0);
	y1,y2,y3,y4,y5,y6,y7,y8:out std_logic_vector(n-1 downto 0);
	flag:out std_logic
);
end bitonic_8x8_nesort_reg;

architecture Behavioral of bitonic_8x8_nesort_reg is
	component cmpr is
	generic
	(
		n:integer:=16
```

```
	);
	port
	(
		clk,start:in std_logic;
		x1,x2:in std_logic_vector(n-1 downto 0);
		y1,y2:out std_logic_vector(n-1 downto 0);
		flag:out std_logic
	);
	end component;

	component bitonic_4x4 is
	generic
	(
		n:integer:=16
	);
	port
	(
		clk,start:in std_logic;
		x1,x2,x3,x4:in std_logic_vector(n-1 downto 0);
		y1,y2,y3,y4:out std_logic_vector(n-1 downto 0);
		flag:out std_logic
	);
	end component;

	component bitonic_8x8_reg is
	generic
	(
		n:integer:=16
	);
	port
	(
		clk,start:in std_logic;
		x1,x2,x3,x4,x5,x6,x7,x8:in std_logic_vector(n-1 downto 0);
		y1,y2,y3,y4,y5,y6,y7,y8:out std_logic_vector(n-1 downto 0);
		flag:out std_logic
	);
	end component;

	signal
y11,y12,y13,y14,y15,y16,y17,y18,y21,y22,y23,y24,y25,y26,y27,y28:std_logic_vector
(n-1 downto 0);
	signal f1,f2,f3,f4,f5,f6:std_logic;

begin
	ulaz1:cmpr
	generic map
	(
		n=>n
	)
	port map
	(
		clk=>clk,
		start=>start,
		x1=>x1,
		x2=>x5,
		y1=>y11,
		y2=>y15,
		flag=>f1
	);
```

```
ulaz2:cmpr
generic map
(
      n=>n
)
port map
(
      clk=>clk,
      start=>start,
      x1=>x2,
      x2=>x6,
      y1=>y12,
      y2=>y16,
      flag=>f2
);

ulaz3:cmpr
generic map
(
      n=>n
)
port map
(
      clk=>clk,
      start=>start,
      x1=>x3,
      x2=>x7,
      y1=>y13,
      y2=>y17,
      flag=>f3
);

ulaz4:cmpr
generic map
(
      n=>n
)
port map
(
      clk=>clk,
      start=>start,
      x1=>x4,
      x2=>x8,
      y1=>y14,
      y2=>y18,
      flag=>f4
);

sredina1:bitonic_4x4
generic map
(
      n=>n
)
port map
(
      clk=>clk,
      start=>f1,
      x1=>y11,
      x2=>y12,
      x3=>y14,
```

```
            x4=>y13,
            y1=>y21,
            y2=>y22,
            y3=>y23,
            y4=>y24,
            flag=>f5
      );

      sredina2:bitonic_4x4
      generic map
      (
            n=>n
      )
      port map
      (
            clk=>clk,
            start=>f3,
            x1=>y15,
            x2=>y16,
            x3=>y18,
            x4=>y17,
            y1=>y25,
            y2=>y26,
            y3=>y27,
            y4=>y28,
            flag=>f6
      );

      izlaz1:bitonic_8x8_reg
      generic map
      (
            n=>n
      )
      port map
      (
            clk=>clk,
            start=>f5,
            x1=>y21,
            x2=>y22,
            x3=>y23,
            x4=>y24,
            x5=>y28,
            x6=>y27,
            x7=>y26,
            x8=>y25,
            y1=>y1,
            y2=>y2,
            y3=>y3,
            y4=>y4,
            y5=>y5,
            y6=>y6,
            y7=>y7,
            y8=>y8,
            flag=>flag
      );
end Behavioral;
```

Vrlo slično kao što je već opisano se realizuju i strukture sa 16 odnosno 32 ulaza/izlaza.

5. ANALIZA PERFORMANSI STRUKTURA ZA SORTIRANJE

Performanse implementiranih algoritama su dobijene izvršavanjem procesa analize i sinteze. Analizarino je više slučajeva za dužine ulaznih/izlaznih magistrala od 8, 16, 32, 64 i 128 bita. Korišćeno je ISE razvojno okruženje za FPGA čipove proizvođača Xilinx. Odabran je uređaj XC5VFX70T-2FF1136 familije Virtex5. Za varijante sa 32 ulaza/izlaza nije bilo moguće izvršiti kompajliranje zbog nedostatka pinova, pa su date približne vrijednosti korišćenih resursa.

Tabela 5.1. Procenjene performanse za različite dužine ulaznih i izlaznih magistrala za bitonic 8x8 algoritam

	Vrednost				
Naziv	8 bita	16 bita	32 bita	64 bita	128 bita
Broj slajs registara	256/44880 (1%)	512/44800 (1%)	1024/44800 (2%)	2048/44800 (4%)	4096/44800 (9%)
Broj slajs LUT-ova	921/44880 (2%)	975/44800 (2%)	2328/44800 (5%)	4632/44800 (10%)	9240/44800 (20%)
Broj potpuno iskorišćenih parova LUT-FF	149/1028 (14%)	263/1224 (21%)	512/2840 (18%)	1024/5656 (18%)	2048/11288 (18%)
Broj globalnih taktova (BUFG/BUFGCTRL)	1/32 (3%)	1/32 (3%)	1/32 (3%)	1/32 (3%)	1/32 (3%)
Broj pinova	129/640 (20%)	257/640 (40%)	513/640 (80%)	1025/640 (160%)	2049/640 (320%)
Maksimalna frekvencija	147 MHz	135 MHz	119 MHz	104 MHz	84 MHz

Tabela 5.2. Procenjene performanse za različite dužine ulaznih i izlaznih magistrala za bitonic 8x8 algoritam sa registrima

	Vrednost				
Naziv	8 bita	16 bita	32 bita	64 bita	128 bita
Broj slajs registara	385/44880 (1%)	769/44880 (1%)	1537/44800 (3%)	3073/4480 (6%)	6145/44800 (13%)
Broj slajs LUT-ova	537/44880 (1%)	961/44880 (2%)	2325/44800 (5%)	4629/4480 (10%)	9237/44800 (20%)
Broj potpuno iskorišćenih parova LUT-FF	363/559 (64%)	769/961 (80%)	1532/2329 (65%)	3059/4643 (65%)	6131/9251 (66%)
Broj globalnih taktova (BUFG/BUFGCTRL)	1/32 (3%)	1/32 (3%)	1/32 (3%)	1/32 (3%)	1/32 (3%)
Broj pinova	131/640 (20%)	259/640 (40%)	515/640 (80%)	1027/649 (160%)	2051/640 (320%)
Maksimalna frekvencija	413 MHz	412 MHz	412 MHz	290 MHz	235 MHz

Tabela 5.3. Procenjene performanse za različite dužine ulaznih i izlaznih magistrala za bitonic 16x16 algoritam

	Vrednost				
Naziv	8 bita	16 bita	32 bita	64 bita	128 bita
Broj slajs registara	896/44880 (2%)	2561/44800 (5%)	3584/44800 (8%)	7168/44800 (16%)	14336/44800 (32%)
Broj slajs LUT-ova	3196/44880 (7%)	3201/44800 (7%)	9288/44800 (20%)	18504/44800 (41%)	36936/44800 (82%)
Broj potpuno iskorišćenih parova LUT-FF	896/3196 (28%)	2561/3201 (80%)	3528/9344 (37%)	7112/18560 (38%)	14280/36992 (38%)
Broj globalnih taktova (BUFG/BUFGCTRL)	1/32 (3%)	1/32 (3%)	1/32 (3%)	1/32 (3%)	1/32 (3%)
Broj pinova	257/640 (40%)	515/640 (80%)	1025/640 (160%)	2049/640 (320%)	4097/640 (640%)
Maksimalna frekvencija	147 MHz	135 MHz	119 MHz	104 MHz	84 MHz

Tabela 5.4. Procenjene performanse za različite dužine ulaznih i izlaznih magistrala za bitonic 16x16 algoritam sa registrima

	Vrednost				
Naziv	8 bita	16 bita	32 bita	64 bita	128 bita
Broj slajs registara	1281/44880 (2%)	2561/44800 (5%)	5121/44800 (11%)	10241/44800 (22%)	20481/44800 (45%)
Broj slajs LUT-ova	1809/44880 (4%)	3841/44800 (8%)	7753/44800 (17%)	15433/44800 (34%)	30793/44800 (68%)
Broj potpuno iskorišćenih parova LUT-FF	1206/1884 (64%)	2533/3869 (65%)	5093/7781 (65%)	10213/15461 (66%)	20453/30821 (66%)
Broj globalnih taktova (BUFG/BUFGCTRL)	1/32 (3%)	1/32 (3%)	1/32 (3%)	1/32 (3%)	1/32 (3%)
Broj pinova	259/640 (40%)	515/640 (40%)	1027/640 (160%)	2051/640 (320%)	4099/640 (640%)
Maksimalna frekvencija	412 MHz	412 MHz	326 MHz	291 MHz	235 MHz

Tabela 5.5. Procenjene performanse za različite dužine ulaznih i izlaznih magistrala za bitonic 32x32 algoritam

	Vrednost				
Naziv	8 bita	16 bita	32 bita	64 bita	128 bita
Broj slajs registara	2816/44800 (6%)	5632/44800 (12%)	11264/44800 (25%)	22528/44800 (50%)	45056/44800 (100%)
Broj slajs LUT-ova	9320/44800 (20%)	13696/44800 (30%)	27328/44800 (61%)	54464/44800 (121%)	108736/44800 (242%)
Broj potpuno iskorišćenih parova LUT-FF	2720/9416 (28%)	5464/13864 (39%)	11096/27496 (40%)	22360/54632 (40%)	44888/108904 (41%)
Broj globalnih taktova (BUFG/BUFGCTRL)	1/32 (3%)	1/32 (3%)	1/32 (3%)	1/32 (3%)	1/32 (3%)
Broj pinova	513/640 (80%)	1025/640 (160%)	2049/640 (320%)	4097/640 (640%)	8193/640 (1280%)
Maksimalna frekvencija	144 MHz	135 MHz	119 MHz	104 MHz	84 MHz

Tabela 5.6. Procenjene performanse za različite dužine ulaznih i izlaznih magistrala za bitonic 32x32 algoritam sa registrima

	Vrednost				
Naziv	8 bita	16 bita	32 bita	64 bita	128 bita
Broj slajs registara	3841/122880 (3%)	7681/122880 (6%)	15361/122880 (12%)	30721/122880 (25%)	61441/122880 (50%)
Broj slajs LUT-ova	5457/122880 (4%)	11521/122880 (9%)	23265/122880 (18%)	46305/122880 (37%)	92161/122880 (75%)
Broj potpuno iskorišćenih parova LUT-FF	3589/5709 (62%)	7625/11577 (65%)	15305/23321 (65%)	30665/46361 (66%)	61385/92217 (66%)
Broj globalnih taktova (BUFG/BUFGCTRL)	1/32 (3%)	1/32 (3%)	1/32 (3%)	1/32 (3%)	1/32 (3%)
Broj pinova	515/960 (53 %)	1027/960 (106%)	2051/960 (213%)	4099/960 (427%)	8195/960 (853%)
Maksimalna frekvencija	412 MHz	412 MHz	326 MHz	291MHz	268 MHz

Tabela 5.7. Procenjene performanse za različite dužine ulaznih i izlaznih magistrala za even_odd 8x8 algoritam

	Vrednost				
Naziv	8 bita	16 bita	32 bita	64 bita	128 bita
Broj slajs registara	256/44800 (1%)	512/44800 (1%)	1024/44800 (2%)	2048/44800 (4%)	4096/44800 (9%)
Broj slajs LUT-ova	674/44800 (1%)	920/44800 (2%)	1843/44800 (4%)	3667/44800 (8%)	7317/44800 (16%)
Broj potpuno iskorišćenih parova LUT-FF	125/761 (15%)	256/1176 (21%)	512/2355 (21%)	1024/4691 (21%)	1932/9481 (20%)
Broj globalnih taktova (BUFG/BUFGCTRL)	1/32 (3%)	1/32 (3%)	1/32 (3%)	1/32 (3%)	1/32 (3%)
Broj pinova	129/640 (20%)	257/640 (40%)	513/640 (80%)	1025/640 (160%)	2049/640 (320%)
Maksimalna frekvencija	145 MHz	136 MHz	119 MHz	105 MHz	85MHz

Tabela 5.8. Procenjene performanse za različite dužine ulaznih i izlaznih magistrala za even_odd 8x8 algoritam sa registrima

	Vrednost				
Naziv	8 bita	16 bita	32 bita	64 bita	128 bita
Broj slajs registara	305/44800 (1%)	609/448800 (1%)	1217/44800 (2%)	2433/44800 (5%)	4865/44800 (10%)
Broj slajs LUT-ova	422/44800 (1%)	913/44880 (2%)	1840/44800 (4%)	3664/44800 (8%)	7312/44800 (16%)
Broj potpuno iskorišćenih parova LUT-FF	286/441 (64%)	595/927 (64%)	1203/1854 (64%)	2419/3678 (65%)	4851/7326 (66%)
Broj globalnih taktova (BUFG/BUFGCTRL)	1/32 (3%)	1/32 (3%)	1/32 (3%)	1/32 (3%)	1/32 (3%)
Broj pinova	131/640 (20%)	259/640 (40%)	515/640 (80%)	1027/640 (160%)	2051/640 (320%)
Maksimalna frekvencija	412 MHz	412 MHz	326 MHz	291 MHz	235 MHz

Tabela 5.9. Procenjene performanse za različite dužine ulaznih i izlaznih magistrala za even_odd 16x16 algoritam

	Vrednost				
Naziv	8 bita	16 bita	32 bita	64 bita	128 bita
Broj slajs registara	880/122880 (1%)	1760/44800 (3%)	3520/44800 (7%)	7040/44800 (15%)	14080/44800 (31%)
Broj slajs LUT-ova	2198/44800 (4%)	3592/44800 (8%)	7192/44800 (16%)	14328/44800 (31%)	29608/44800 (63%)
Broj potpuno iskorišćenih parova LUT-FF	767/2311 (33%)	1480/3872 (38%)	3016/7696 (39%)	6088/15280 (39%)	12000/30688 (39%)
Broj globalnih taktova (BUFG/BUFGCTRL)	1/32 (3%)	1/32 (3%)	1/32 (3%)	1/32 (3%)	1/32 (3%)
Broj pinova	257/640 (40%)	513/640 (80%)	1025/640 (160%)	2049/640 (320%)	4097/640 (640%)
Maksimalna frekvencija	143 MHz	136 MHz	119 MHz	105 MHz	85 Hz

Tabela 5.10. Procenjene performanse za različite dužine ulaznih i izlaznih magistrala za even_odd 16x16 algoritam sa registrima

	Vrednost				
Naziv	8 bita	16 bita	32 bita	64 bita	128 bita
Broj slajs registara	1009/122880 (2%)	2017/44800 (4%)	4033/44800 (9%)	8065/44800 (18%)	16129/44800 (36%)
Broj slajs LUT-ova	1418/44800 (3%)	2521/44800 (5%)	6104/44800 (13%)	12152/44800 (27%)	24248/44800 (54%)
Broj potpuno iskorišćenih parova LUT-FF	949/1478 (64%)	2017/2521 (80%)	4005/6132 (65%)	8037/12180 (65%)	16101/24276 (66%)
Broj globalnih taktova (BUFG/BUFGCTRL)	1/32 (3%)	1/32 (3%)	1/32 (3%)	1/32 (3%)	1/32 (3%)
Broj pinova	259/640 (40%)	515/640 (80%)	1027/640 (160%)	2051/640 (320%)	4099/640 (640%)
Maksimalna frekvencija	412 MHz	412 MHz	326 MHz	291 MHz	235 MHz

Tabela 5.11. Procenjene performanse za različite dužine ulaznih i izlaznih magistrala za even_odd 32x32 algoritam

	Vrednost				
Naziv	8 bita	16 bita	32 bita	64 bita	128 bita
Broj slajs registara	2752/44800 (6%)	5504/44800 (12%)	11008/44800 (24%)	22016/44800 (49%)	44032/44800 (98%)
Broj slajs LUT-ova	6716/44800 (14%)	10800/44800 (24%)	21620/44800 (48%)	43092/44800 (96%)	86060/44800 (192%)
Broj potpuno iskorišćenih parova LUT-FF	2411/7057 (34%)	4824/11480 (42%)	9816/22812 (43%)	19800/45308 (43%)	39304/90788 (43%)
Broj globalnih taktova (BUFG/BUFGCTRL)	1/32 (3%)	1/32 (3%)	1/32 (3%)	1/32 (3%)	1/32 (3%)
Broj pinova	513/640 (80%)	1025/640 (160%)	2049/640 (320%)	4097/640 (640%)	8193/640 (1280%)
Maksimalna frekvencija	150 MHz	136 MHz	119 MHz	105 MHz	85 MHz

Tabela 5.12. Procenjene performanse za različite dužine ulaznih i izlaznih magistrala za even_odd 32x32 algoritam sa registrima

	Vrednost				
Naziv	8 bita	16 bita	32 bita	64 bita	128 bita
Broj slajs registara	3057/122880 (2%)	6113/122880 (4%)	12225/122880 (9%)	24449/122880 (19%)	48897/122880 (39%)
Broj slajs LUT-ova	4330/122880 (3%)	9169/122880 (7%)	18512/44880 (15%)	36848/122880 (29%)	73520/122880 (59%)
Broj potpuno iskorišćenih parova LUT-FF	2854/4533 (62%)	6057/9225 (65%)	12169/18568 (65%)	24393/36904 (66%)	48841/73576 (66%)
Broj globalnih taktova (BUFG/BUFGCTRL)	1/32 (3%)	1/32 (3%)	1/32 (3%)	1/32 (3%)	1/32 (3%)
Broj pinova	515/960 (53%)	1027/960 (106%)	2051/960 (213%)	4099/960 (426%)	8195/960 (853%)
Maksimalna frekvencija	412 MHz	412 MHz	326 MHz	291 MHz	235 MHz

Prema podacima iznad, može se zaključiti da se broj zahtevanih pinova povećava linearno sa porastom veličine ulaza/izlaza u sorter jer svaki bit podrazumeva jedan pin. Povećanjem dimenzije sortera (veći broj izlaza/ulaza) broj zahtevanih pinova takođe linearno raste. Međutim, u kompletnoj realizaciji to ne bi predstavljalo problem jer bi paketi pristizali serijski preko gigabitskih linkova i onda se unutar samog čipa konvertovali u paralelni prenos koji je korišćen u realizaciji sortera opisanoj u ovoj tezi.

Takođe, potrošnja resursa prvenstveno zavisi od dimenzije komutatora, a još više od veličine ulaza/izlaza, pa treba odabrati veličinu ulaza/izlaza prema željenom kapacitetu koji struktura za sortiranje treba da podrži (veća veličina ulaza/izlaza daje veći protok, ali po cenu zauzimanja resursa

čipa). Očekivano maksimalna radna frekvencija raste ako se koriste registri između kaskada, ali uz cenu veće količine zauzetih registara u čipu.

6. ZAKLJUČAK

U ovom radu je prikazana implementacija algoritma za sortiranje koji se koriste kod Banyan komutatora. Algoritmi su realizovani za različit broj ulaza i izlaza. Da bi se postigla veća brzina rada, u odnosu na strukture implementirane pomoću kombinovane logike, implementirane su i strukture sa registrima.

Banyan mreža ima veliki broj poželjnih osobina. Ima regularnu strukturu. Poseduje jedinstvenost putanje, tj. postoji tačno jedna putanja koja povezuje bilo koji ulaz sa bilo kojim izlazom Banyan mreže. Uspostavljanje takve putanje ostvaruje se na distribuirani način, korišćenjem jednostavne procedure samoprosleđivanja, gde se informacija o prosleđivanju na svakom stepenu dobija direktno iz adrese odredišta sadržane u zaglavlju paketa. Jedinstvenost putanje osigurava da, kada se Banyan mreža koristi za brzu komutaciju ćelija, sve ćelije određene veze slede istu putanju. Saglasno tome, sprečeno je isporučivanje ćelija van redosleda, a varijacija kašnjenja između ćelija je mala, što vodi ka prostijim protokolima sinhronizacije za servise u realnom vremenu kao što su servisi za prenos govornog i/ili audio signala. Jedna nepogodnost ovih mreža, prouzrokovana internim konfliktima (različite putanje ulaz/izlaz mogu da imaju jedan ili više zajedničkih linkova za međupovezivanje), upravo je dovela do pojave vrlo velikog broja različitih realizacija komutatora koje se baziraju na Banyan mreži. Namera svih tih realizacija je da umanji, ili eliminiše pojavu blokiranja, uz umereno povećanje složenosti.

Par-nepar struktura ima manje svič elemenata, dok bitonička struktura ima jednak broj svič elemenata u svim kaskadama (kolonama). Strukture za sortiranje mogu da poprave performanse Banyan komutatora (pod uslovom da se koristi omega ili n-cube struktura), ali unose dodatne kaskade, tj. troše dodatne hardverske resurse.

Realizovane implementacije su parametrizovane sa stanovišta veličine ulaza i izlaza pa se lako mogu prilagoditi željenom kapacitetu koji se želi podržati. Takođe, realizovane implementacije su portabilne, tj. mogu se bez ikakvih izmena implementirati i na FPGA čipovima drugih proizvođača poput Altere.

LITERATURA

[1] Doc. dr Zoran Čiča, Materijali i beleške sa predavanja sa predmeta: Programiranje komunikacionog hardvera

[2] Doc. dr Zoran Čiča, Materijali i beleške sa predavanja sa predmeta: Komutacioni sistemi

[3] http://telekomunikacije.etf.bg.ac.rs/predmeti/te5itm/ATMkomplet.pdf 05.09.2015.

[4] http://es.elfak.ni.ac.rs/rmif/Materijal/TCP-IP.pdf 01.09.2015.

[5] https://brakussale.wordpress.com/2013/02/28/rutiranje/ 20.09.2015.

[6] http://www.laynetworks.com/Banyan%20Switch.htm 20.09.2015.