Univerzitet u Beogradu

Elektrotehnički fakultet

Magistarski rad

Jedan pristup kvantitativnoj analizi dužina veza u slučajevima 2D i 3D VLSI realizacija 2D sistoličkih polja

Mentor:
Prof. dr Veljko Milutinović

Kandidat:
Aleksandar Milenković

Sadržaj

Tabele

Slike

1. Uvod

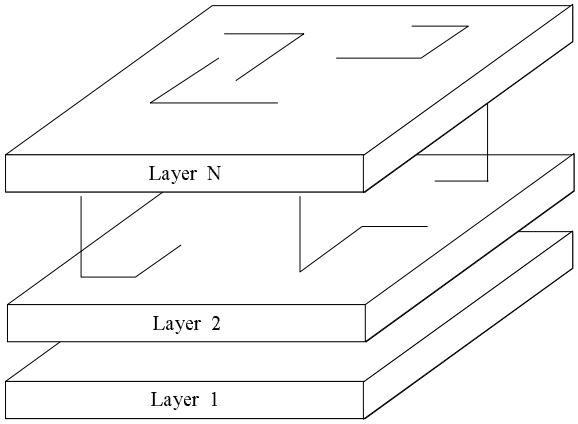
Razvoj VLSI tehnologije praćen je stalnim rastom zahteva za većom procesnom snagom i bržom obradom. Povećanje procesne snage, brzine čipa, smanjenje potrošnje i dimenzija su glavni izazovi koji stoje pred VLSI tehnologijom. Koncept trodimenzionalne (3D) integracije je predložen sa ciljem da se prevaziđu ograničenja postojeće dvodimenzionalne (2D) VLSI tehnologije u pogledu gustine pakovanja i performanse (brzine). Takođe, novi koncept je otvorio mogućnost formiranja novih višefunkcionalnih uređaja sa paralelnim procesiranjem. Jedna vrsta visoko paralelnih struktura su sistolička polja. Napredak 2D VLSI tehnologije izražen, pre svega, povećavanjem raspoložive površine na čipu, omogućio je VLSI implementaciju sistoličkih polja.

U prvom delu poglavlja date su osnove koncepta 3D integracije sa osvrtom na prednosti koje donosi taj koncept. U drugom delu dat je opis koncepta sistoličkih polja sa osvrtom na oblasti primene.

1.1. O tehnologiji

Razvoj VLSI tehnologije ogleda se, pre svega, u stalnom smanjivanju dimenzija tranzistora, sa jedne strane, i u povećavanju površine čipa, sa druge strane; takav trend omogućio je povećanje gustine pakovanja, tako da su danas raspoloživi procesorski čipovi sa 10 miliona tranzistora. Međutim, fundamentalni faktor skaliranja za MOS tehnologiju od 0.2μm nameće potrebu da se traže alternativni putevi koji će omogućiti dalji rast gustine pakovanja komponenti i povećavanje performanse. Jedan od mogućih budućih pravaca daljeg razvoja VLSI tehnologije je i koncept trodimenzionalne integracije (3D VLSI) i koncept 3D pakovanja dvodimenzionalnih VLSI komponenti.

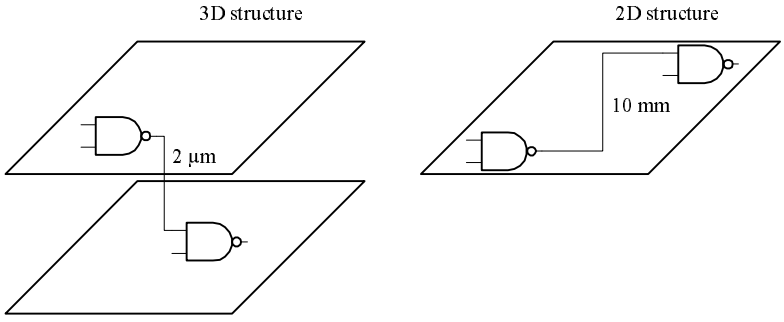
3D VLSI čip se sastoji od više aktivnih nivoa koji su postavljeni jedan iznad drugog. Struktura jednog 3D VLSI čipa kao i kratak opis načina fabrikovanja takvog čipa dati su na slici 1.



Slika 1. Struktura 3D VLSI čipa.

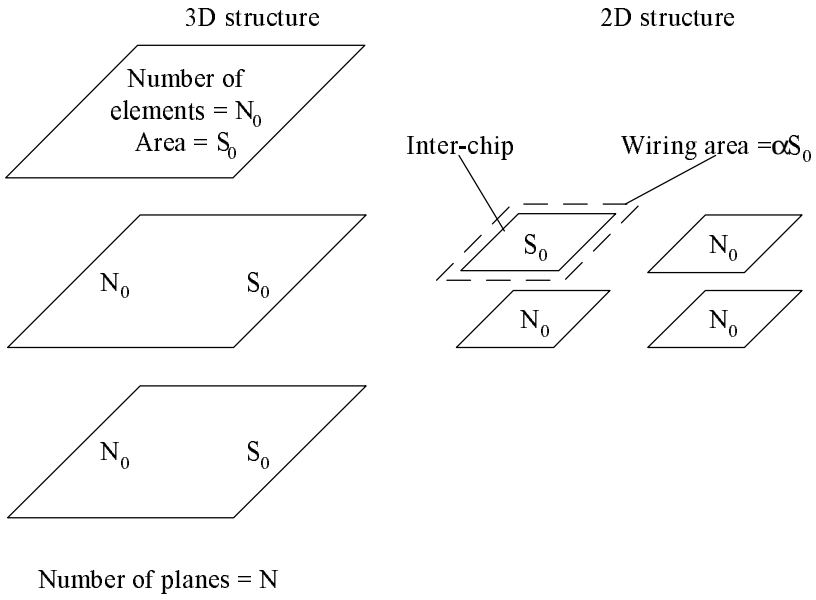
Opis: 3D VLSI čip se sastoji od više aktivnih nivoa koji su postavljeni jedan iznad drugog. U fabrikaciji aktivnih nivoa može se koristiti SOI (*Silicon on Insulator*) tehnologija. U ovoj tehnologiji prvo se vrši fabrikacija elemenata u jednom aktivnom nivou korišćenjem 2D procesa. Zatim se nanosi neaktivni sloj - izolator (*silicon dioxide*) koji se planarizuje "ecovanjem" pre nego što se nanese sledeći polisilicijumski sloj. Novi sloj silicijuma koji je rekristalizovan laserskim snopom koristi se za fabrikovanje aktivnih elemenata korišćenjem 2D procesa. Ponavljanjem navedene procedure može se dobiti čip sa više aktivnih nivoa.

Tehnologija fabrikovanja čipova u tri dimenzije i pakovanja 2D čipova u tri dimenzije ima brojne prednosti u odnosu na klasičnu tehnologiju fabrikovanja čipova u dve dimenzije. Na slici 2 prikazana je dužina veza u 3D i 2D strukturama [Koku86]. U 3D strukturi komponente mogu biti bliže jedna drugoj, što rezultuje kraćim vezama između komponenti. Skraćivanjem dužina veza skraćuje se kašnjenje na vezi; skraćivanjem kašnjenja na vezama koje pripadaju kritičnoj stazi za podatke povećava se brzina takta, pa prema tome i performansa. Na slici 3 ilustrovan je odnos gustina pakovanja 3D i 2D struktura. Povećavanjem gustine pakovanja više komponenti se može smestiti na isti čip, čime se smanjuje *off-chip* komunikacija. Smanjenje *off-chip* komunikacije direktno utiče na povećanje propusne moći. U 3D strukturama mogu se koristiti vertikalne veze za povezivanje većeg broja komponenti, što je ilustrovano na slici 4.



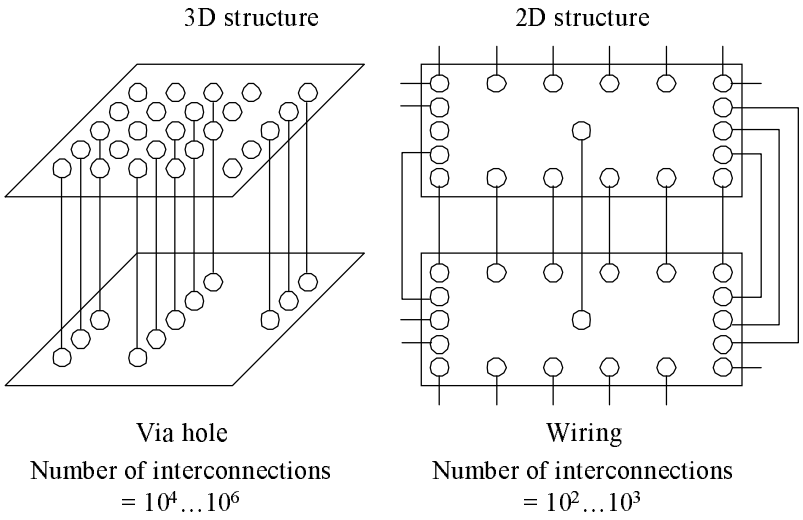
Slika 2. Poređenje dužina veza u 2D i 3D strukturi.

Opis: U 3D strukturi međusobno povezane komponente mogu biti bliže jedna drugoj, pa su time i dužine veza kraće. Na slici je pokazan primer povezivanja dva NI kola u 3D i 2D strukturi, redom. Razmeštanjem komponenti koje se povezuju u različite aktivne ravni tako da budu jedna ispod druge postiže se vrlo značajno skraćivanje veza.



Slika 3. Poređenje gustina pakovanja 2D i 3D strukture.

Opis: Posmatra se 3D struktura sa N ravni. Broj komponenti razmeštenih u jednoj aktivnoj ravni je N_0, a površina svake od ravni je S_0. Na osnovu slike, gustina pakovanja u 3D strukturi je NN_0/S_0. Gustina pakovanja u 2D strukturi je $N_0/\alpha S_0$, pri čemu α predstavlja odnos ukupne potrebne površine za povezivanje između različitih komponenti i površine S_0. Prema tome, odnos gustina pakovanja 3D i 2D struktura je αN. Parametar α ima vrednost, tipično, između 2 i 5; na osnovu toga, gustina pakovanja kod 3D struktura je od 20 do 50 puta veća od gustine pakovanja kod 2D struktura.



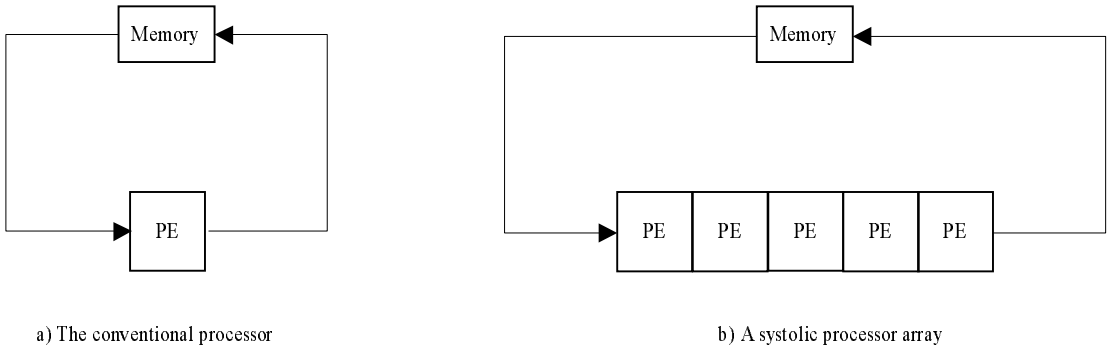
Slika 4. Poređenje broja veza u 2D i 3D strukturi.

Opis: 3D struktura dozvoljava povezivanje većeg broja komponenti koje se mogu razmestiti u različite ravni i povezati vertikalnim vezama. Tipičan broj vertikalnih veza u 3D strukturi je između 10^4 i 10^6, dok je tipičan broj veza u slučaju povezivanja 2D struktura između 10^2 i 10^3.

1.2. O sistoličkim poljima

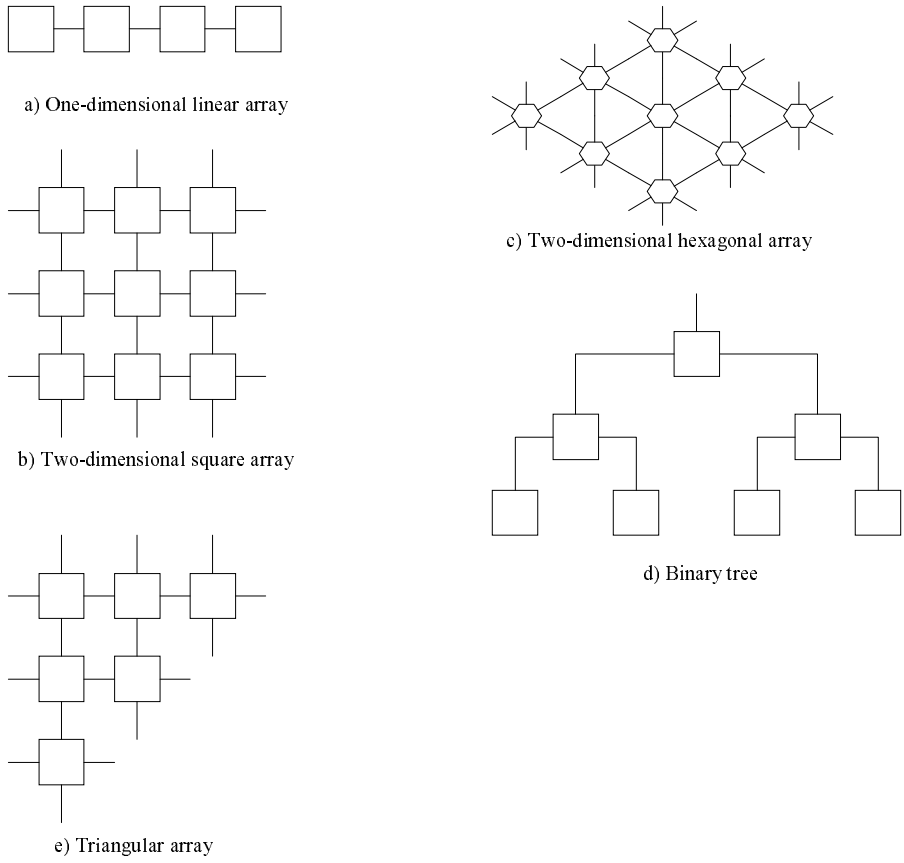
Pojava sistoličkih polja bila je rezultat napretka u VLSI tehnologiji i pojave aplikacija koje su zahtevale veliku procesnu snagu. Pojam sistolička polja prvi put je uveden u radu H. T. Kunga i C. E. Leisersona 1978. godine i od tada je veliki broj istraživanja posvećen arhitekturi i VLSI implementaciji sistoličkih polja.

Sistolička polja se sastoje od identičnih, međusobno povezanih procesnih elemenata (ćelija) koji mogu da izvršavaju jednostavne aritmetičke operacije. Podaci u sistoličkom polju se prenose između ćelija u duhu protočne obrade, a komunikacija sa spoljnim svetom se odvija preko ivičnih ćelija (*boundary cells*), koje imaju funkciju ulazno/izlaznih portova za ceo sistem. Osnovni princip funkcionisanja sistoličkih polja ilustrovan je na slici 5. Funkcija memorije je analogna funkciji srca; memorija "pumpa" podatke kroz niz procesnih elemenata. Jednom dohvaćeni podatak iz memorije efektivno se koristi u svakom od procesnih elemenata. Pored toga, ovakav pristup odlikuje se: (a) mogućnošću proširivanja, (b) jednostavnim tokom podataka i kontrolnih signala i (c) korišćenjem prostih i uniformnih ćelija. U zavisnosti od vrste problema razvijene su brojne konfiguracije sistoličkih polja koje su pokazane na slici 6.



Slika 5. Koncept sistoličkih polja.

Opis: Zamenom jednog procesnog elementa (PE) sa nizom procesnih elemenata postiže se veća procesna snaga bez povećavanja broja pristupa memoriji. Suština ovakvog pristupa je da jednom dohvaćeni podatak iz memorije bude efektivno korišćen u svim procesnim elementima kroz koje prolazi. Ovakav pristup je primenljiv kod široke klase problema koji su poznati kao *compute-bound* problemi, kod kojih se nad jednim podatkom vrši više izračunavanja, odnosno gde je broj izračunavanja veći od broja ulazno/izlaznih operacija. Primer algoritma iz ove klase problema je množenje matrica; broj izračunavanja tipa $c + a*b$ je $O(n^3)$, dok je broj ulazno/izlaznih operacija $O(n^2)$, pri čemu je n broj vrsta, odnosno kolona matrice. Sa druge strane su *I/O bound* problemi kod kojih je broj ulazno/izlaznih operacija veći od broja izračunavanja. Primer algoritma iz ove klase problema je sabiranje matrica; broj izračunavanja je n^2, a broj ulazno/izlaznih operacija je $3n^2$.

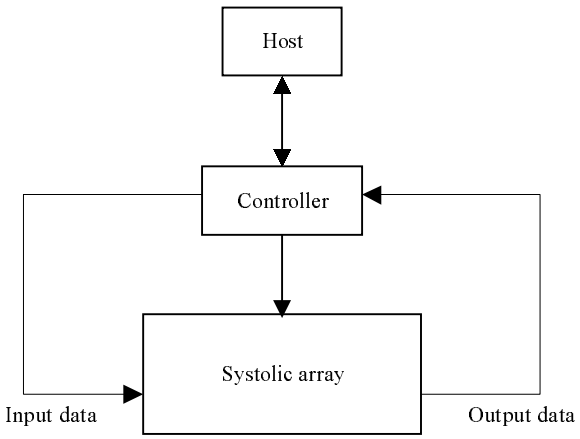


Slika 6. Tipične konfiguracije sistoličkih polja.

Opis: U zavisnosti od aplikacija koriste se različite strukture sistoličkih polja. Tako se, na primer, jednodimenzionalni niz koristi za izračunavanje konvolucije i diskretne Furijeove transformacije; dvodimenzionalni kvadratni niz se koristi u oblasti dinamičkog programiranja, rešavanja problema iz teorije grafova, itd; dvodimenzionalni heksagonalni niz se koristi u matričnim algoritmima, za izračunavanje tranzitivnog zatvaranja, u problemima prepoznavanja oblika; sistolička polja u obliku stabla se koriste, pre svega, u problemima pretraživanja.

Algoritmi koji su podesni za implementaciju pomoću sistoličkih polja mogu se naći u brojnim aplikacijama iz oblasti digitalne obrada signala, slike i prepoznavanja oblika kao što su jednodimenzionalna konvolucija, dvodimenzionalana konvolucija i korelacija, diskretna Furijeova transformacija, interpolacija, uparivanje oblika, procesiranje radarskih signala, itd; takođe, sistolička polja se koriste u brojnim matričnim i nenumeričkim algoritmima koji uključuju linearno i dinamičko programiranje, algoritme za rad sa grafovima, itd. Zapravo, većina algoritama iz gore pomenutih aplikacija spada u grupu *compute-bound* algoritama i zahteva implementaciju pomoću sistoličkih polja kada se koriste za rad u realnom vremenu.

Tipična struktura sistema sa sistoličkim poljem prikazana je na slici 7. Značaj sistoličkih polja u oblasti procesiranja digitalnih signala u realnom vremenu potvrđen je i činjenicom da mnogi proizvođači imaju proizvode bazirane na sistoličkim poljima (ESL-TRW, Hughes, NCR, GE, Motorola).



Slika 7. Tipična struktura sistema sa sistoličkim poljem.

Opis: Sistolička polja se uglavnom koriste kao akceleratori za ubrzavanje izvršavanja kritičnih operacija koje zahtevaju veliku procesnu snagu. Sistoličko polje je povezano preko kontrolera sa *host* računarom. Kontroler sistoličkog polja obavlja sledeće funkcije: (a) prihvata ulazne podatake sa magistrale, (b) generiše upravljačke signale kojima se kontroliše rad sistoličkog polja na osnovu programa koji je pripremljen na *host* računaru, (c) prosleđuje ulazne podatke sistoličkom polju, (d) prikuplja dobijene rezultate i (e) prosleđuje dobijene podatke *host* računaru.

2. Definicija problema

Osnovni problem koji se razmatra u ovoj tezi je kvantitativna analiza dužina veza na čipu i evaluacija skraćivanja dužina veza prilikom prelaska sa 2D na 3D VLSI tehnologiju. U prvom delu ovog poglavlja data je formalna definicija problema; u drugom delu istaknut je značaj problema koji se razmatra.

2.1. Problem i njegova suština

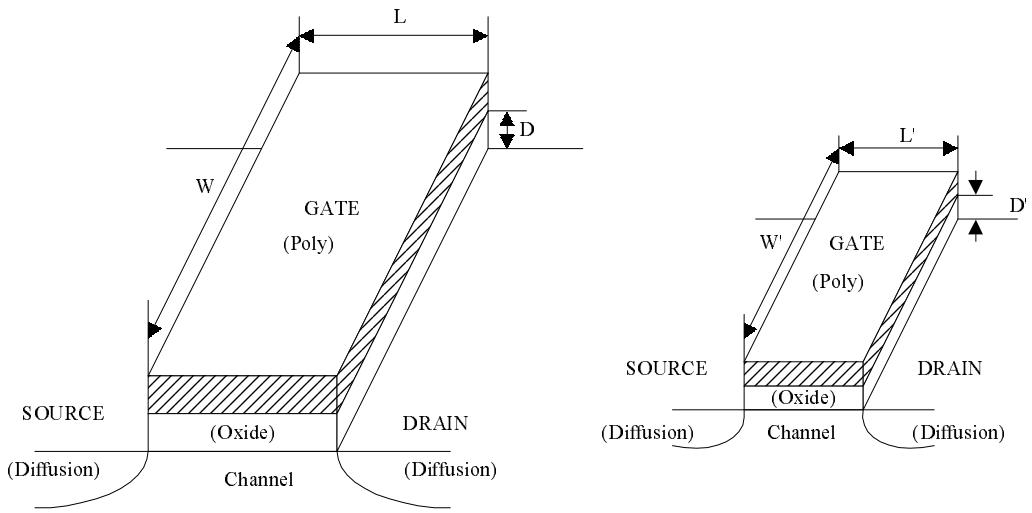
U prethodnom poglavlju razmatrana su poboljšanja koja se očekuju usled prelaska sa 2D na 3D VLSI tehnologiju. Jedan od osnovnih razloga za prelazak sa 2D na 3D VLSI je poboljšanje performanse. Poboljšanje performanse se postiže kroz (a) skraćivanje veza na čipu i (b) povećavanje gustine pakovanja.

Skraćivanjem dužina veza na čipu smanjuje se kašnjenje na vezama, pa prema tome i ukupno kašnjenje na kritičnoj stazi za podatke. Takođe, skraćivanjem dužina veza na čipu smanjuje se potrebna površina na čipu, što opet omogućava povećavanje brzine takta. Povećavanjem gustine pakovanja omogućava se smeštanje više komponenti na jedan čip, čime se potencijalno eliminiše potreba za komunikacijom sa uređajima na drugom čipu (*off-chip* komunikacija). Eliminisanje ili smanjivanje *off-chip* komunikacije doprinosi povećanju propusne moći.

I pored činjenice da se poslednjih godina pojavio veliki broj radova koji se bavi problemima 3D integracije, još nije dat odgovor na pitanje u kolikoj meri 3D pristup može da poboljša performansu. Do sada su 3D čipovi uglavnom razvijani u istraživačkim laboratorijama i ne postoji CAD (*Computer Aided Design*) alat koji bi olakšao komplikovan dizajnerski posao i omogućio analizu i poređenje različitih pristupa. Osnovni cilj ovog istraživanja je analiza dužina veza na 2D i 3D čipovima i njihovo poređenje. U tom cilju razvijeno je okruženje koje koristi postojeće CAD alate za 2D VLSI i originalno razvijen programski paket za analizu dužina veza na čipu.

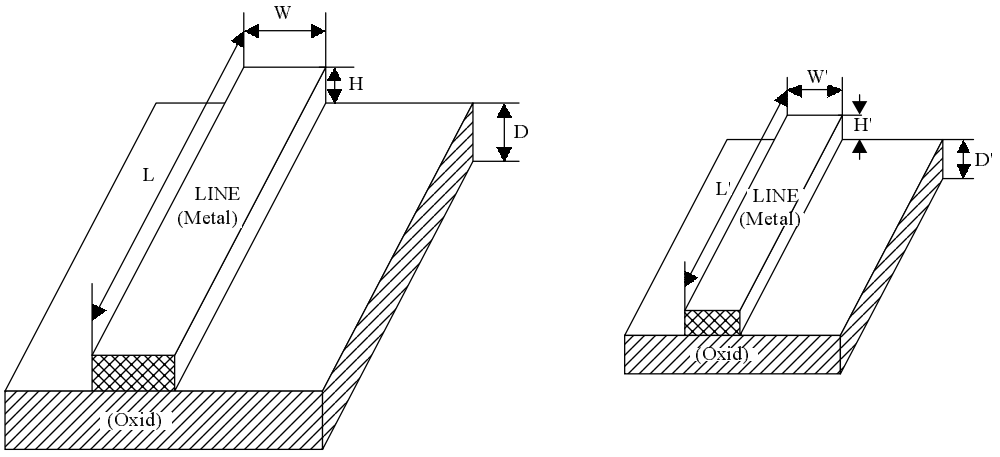
2.2. Problem i njegov značaj

Eksponencijalni rast kompleksnosti integrisanih kola u klasičnom 2D pristupu omogućen je kombinacijom: (a) smanjivanja dimenzija tranzistora i veza i (b) povećavanja maksimalane površine čipa. Efekti smanjivanja dimenzija (skaliranja) tranzistora i veza prikazani su na slikama 8 i 9, redom.



Slika 8. Skaliranje dimenzija i napona tranzistora.

Opis: Skaliranje dimenzija tranzistora podrazumeva da se sve linearne dimenzije W, L i D dele sa konstantnim parametrom α koji predstavlja faktor skaliranja. Takođe, napon V se deli sa istim faktorom α. Na osnovu toga, nakon skaliranja dimenzije tranzistora definisane su sledećim formulama: $L' = L/\alpha$, $W' = W/\alpha$, $D' = D/\alpha$; napon V' iznosi $V' = V/\alpha$. Osnovni parametri tranzistora (kašnjenje, kapacitivnost i jačina struje) i veza između parametara polaznog i skaliranog tranzistora dati su sledećim jednačinama:
Kašnjenje kroz gejt (*gate delay*): $\tau \sim L^2/V \Rightarrow \tau' = \tau/\alpha$,
Kapacitivnost gejta (*gate capacitance*): $C \sim LW/D \Rightarrow C' = C/\alpha$,
Jačina struje (*drain-to-source current*): $I \sim WV^2/LD \Rightarrow I' = I/\alpha$.
Na osnovu jednačina zaključuje se da se skaliranjem dimenzija i napona tranzistora smanjuje kašnjeje, kapacitivnost i struja kroz gejt, proporcionalno faktoru skaliranja α.



Slika 9. Skaliranje dimenzija veza.

Opis: Slično kao kod skaliranja tranzistora, skaliranjem veza sve linearne dimenzije L, W i D delimo sa faktorom skaliranja α. Kašnjenje na vezi je obrnuto proporcionalno proizvodu podužne kapacitivnosti i induktivnosti. Odnos između kašnjenja polazne i skalirane veze dat je sledećom jedanačinom:
Kašnjenje ne vezi (*interconnection delay*): $\tau_i = 1/RC \sim DH/L^2 \Rightarrow \tau'_i = \tau_i$.
Na osnovu jednačine sledi da skaliranjem dimenzija veza kašnjenje na vezama ostaje nepromenjeno.

Na osnovu jednačina koje opisuju parametre skaliranog tranzistora sledi da se skaliranjem smanjuje kašnjenje, kapacitivnost i struja. Međutim, skaliranjem dimenzija veza kašnjenje kroz veze ostaje nepromenjeno jer se skaliranjem smanjuje podužna kapacitivnost, ali se povećava podužna otpornost. Na osnovu toga sledi da će smanjivanjem dimenzija tranzistora brzina takta, a time i performansa integrisanih kola, biti dominantno određene kašnjenjem na vezama. Zapravo, značaj kašnjenja na vezama je još i veći nego što se da zaključiti iz teorije skaliranja. Naime, kompleksnost čipova stalno raste, tj. raste površina čipa, pa je često neopravdano tvrditi da skaliranjem dolazi do skraćivanja dužina veza. Na taj način kašnjenje na vezama postaje još dominantnije [Kung88]. Imajući sve gore navedeno u vidu jasno je od kolikog je značaja optimizovati veze, odnosno učiniti ih što je moguće kraćim.

3. Postojeća rešenja

Koliko je autoru poznato, u otvorenoj literaturi nema radova koji se odnose na kvantitativnu analizu skraćenja dužina veza pri prelasku sa 2D na 3D VLSI. Do sada su rađene samo grube procene na bazi laboratorijskih prototipskih implementacija [Grin84], [Malh87], [Akas88] koje ukazuju na činjenicu da ušteda sigurno postoji i da je relativno velika, ali se nije ulazilo u to koliko je velika.

Najbliže što je do sada rađeno je kvalitativna analiza [Tong95] koja postavlja određenu osnovu za dalju nadgradnju u pravcu kvantitativne analize, jer definiše metodologiju za 3D povezivanje. U daljem tekstu biće reči prvo o dosadašnjim implementacijama, a zatim i o predloženoj metodologiji 3D rutiranja.

3.1. 3D realizacije

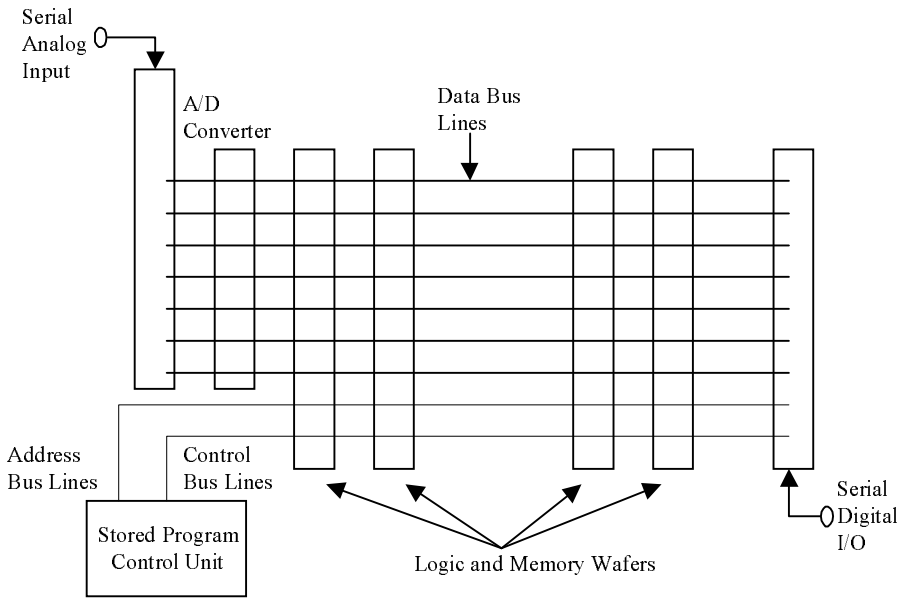
Poslednjih godina pojavile su se brojne 3D strukture poznate pod imenima *3D chips* [Rebe95], *3D Computers* [Grin84], *Orthogonal Chip Mount* [Malh87]. U ovom poglavlju dat je prikaz nekoliko najvažnijih projekata iz oblasti 3D integracije.

3.1.1. Hughes 3D Computer

3D procesor razvijen u *Hughes Research Labaratory* bio je prvi sistem koji je izgrađen trodimenzionalnom integracijom poluprovodničkih kolača (*wafer-scale integration*) [Grin84]. Procesor je deo sistema koji je poznat kao 3D *Computer*. Ceo sistem je razvijen sa ciljem da se koristi za analizu slike i procesiranje signala. Poprečni presek 3D procesora sa objašnjenjima dat je na slici 10. 3D procesor sadrži više poluprovodničkih kolača (15 u tekućoj realizaciji) povezanih međusobno preko vertikalnih veza. Svaki kolač sadrži NxN (N=128) identičnih procesnih elemenata. Kroz poluprovodnički kolač vertikalno prolazi NxN linija koje povezuju procesne elemente razmeštene u različitim poluprovodničkim kolačima. Svi procesni elementi obavljaju bit-serijske operacije i rade istovremeno. Jednostavnost procesnih elemenata omogućava masovni paralelizam kojim se nadoknađuje gubitak u brzini usled bit-serijske aritmetike. Postoji pet različitih tipova poluprovodničkih kolača (procesnih elemenata):

- Memorija. Ovaj tip poluprovodničkog kolača sadrži procesne elemente koji mogu da pamte 16-bitni podatak, da razmenjuju sadržaj sa četiri susedna elementa i da izvršavaju proste logičke operacije nad podacima. Takođe, ovaj tip kolača omogućuje I/O komunikaciju između 3D procesora i kontrolera.
- Akumulator. Ovaj tip poluprovodničkog kolača izvršava aritmetičke i logičke operacije između podatka koji je smešten u akumulatoru i podatka koji pristiže bit-serijski preko vertikalnih linija. Operacije se izvršavaju bit-serijski.
- Brojač. Ovaj tip kolača se koristi za brojanje događaja; za pamćenje se koristi 5-bitni pomerački registar.

- Replikator. Ovaj tip kolača koristi se kao sprežna mreža u sistemu.
- Komparator. Ovaj tip kolača sadrži procesne elemente koji mogu da porede lokalni podatak sa podatkom koji pristiže bit-serijski po vertikalnim linijama.

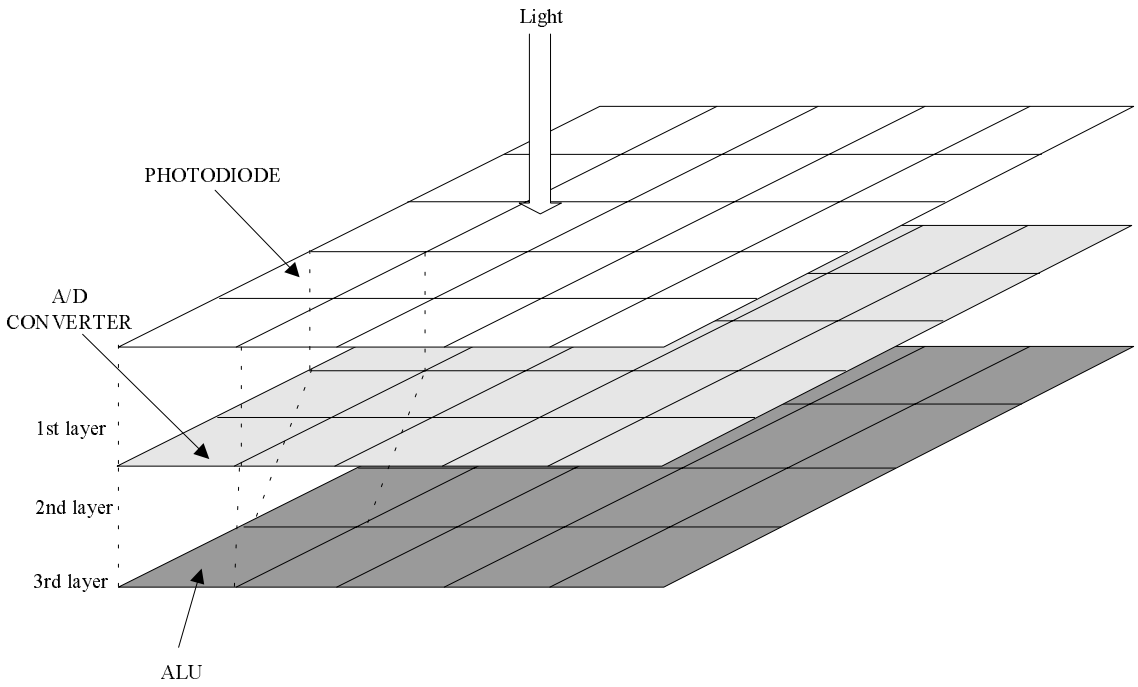


Slika 10. Poprečni presek 3D procesora.

Opis: 3D procesor sadrži više poluprovodničkih kolača koji su na slici predstavljeni vertikalnim pravougaonicima. Horizontalne linije predstavljaju linije za podatke i kontrolne linije. Linije za podatke povezuju procesne elemente koji su smešteni u različitim ravnima jedan ispod drugog. Procesni elementi obavljaju operacije bit-serijski, dok svi procesni elementi rade istovremeno, paralelno na nivou reči. Na taj način moguće je postići masivni paralelizam (sa matricom od 1024x1024 procesnih elemenata u jednom poluprovodničkom kolaču), jer su procesni elementi relativno jednostavni.

3.1.2. Mitsubishi test procesor

U laboratorijama japanske firme *Mitsubishi* razvijen je test procesor za obradu slike sa visoko paralelnom arhitekturom [Akas88]. Uređaj sadrži matricu od 5x5 fotosenzora u prvoj aktivnoj ravni, 2-bitne AD konvertore u drugoj aktivnoj ravni i 40 aritmetičko-logičkih jedinica sa pridruženim pomeračkim registrima koji su razmešteni u trećoj aktivnoj ravni. Osnovna struktura i arhitektura procesora prikazana je na slici 11.

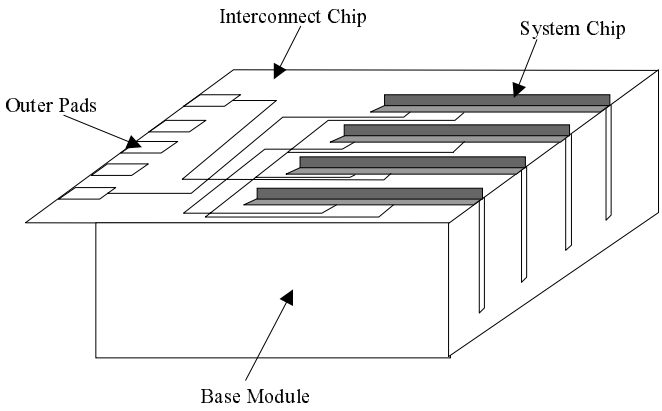


Slika 11. Osnovna struktura 3D test procesora za obradu slike firme *Mitsubishi*.

Opis: Na slici je prikazana struktura po nivoima procesora za obradu slike. Prvi nivo sadrži matricu 5x5 fotodioda, drugi nivo sadrži 2-bitne A/D konvertore, dok se u poslednjem nivou nalazi 40 ALU jedinica i pomeračkih registara. Svaki signal (piksel) se prenosi vertikalno u drugi nivo gde se vrši digitalizacija. Digitalizovani signali svaka dva susedna piksela procesiraju se u zajedničkoj ALU jedinici koja je smeštena u poslednjem nivou.

3.1.3. OCM pristup

U radu [Malh87] predloženo je jedno rešenje za hibridnu integraciju poluprovodničkih kolača (*Hibrid Wafer Scale Integration*) koje je poznato pod imenom OCM (*Orthogonal Chip Mount*). Primer koji ilustruje OCM koncept prikazan je na slici 12.



Slika 12. OCM koncept.

Opis: Mehanički držač (*Base Module*) poseduje više slotova u zavisnosti od broja čipova (*System Chip*) koji se vertikalno smeštaju u slotove. Na vrhu baznog modula nalazi se čip (*Interconnect Chip*) koji sadrži veze između sistemskih čipova, pedove prema sistemskim čipovima i pedove prema spoljašnjem svetu (*Outer Pads*).

3.2. Monolitička 3D integracija i rutiranje u 3D čipu

Od svih tehnika 3D integracije najviše se očekuje od monolitičke integracije u 3D čipu. Međutim, još nisu raspoloživi CAD alati za automatsko razmeštanje i rutiranje u 3D čipu. Osnovni elementi metodologije 3D rutiranja prezentovani su u radu čiji su autori Chao Chi Tong i Chuan-lin Wu [Tong95]. Predložena 3D metodologija rutiranja bazira se na *standard cell* i *gate array* VLSI metodologijama projektovanja.

Kako se predložena metodologija 3D rutiranja bazira na postojećim 2D metodologijama rutiranja, u daljem tekstu dat je kratak pregled postojećih metodologija 2D rutiranja. Postoje dve osnovne metodologije 2D rutiranja: globalno rutiranje (*global routing*) i rutiranje po kanalima (*channel routing*). Globalno rutiranje bazirano je na *Lee* algoritmu i radi nad celom površinom za rutiranje tražeći najkraći mogući put za svaku vezu. I pored modifikacija ovakvih algoritama koji povećavaju verovatnoću uspešnog završetka rutiranja, ovakvo rutiranje je jako sporo i ne garantuje uspešan završetak. Sa druge strane, rutiranje po kanalima primenjuje strategiju "*divide and conquer*" deleći oblast za rutiranje na male kanale. Detaljno rutiranje se dalje vrši unutar kanala. Problem rutiranja po kanalima podeljen je u četiri faze: *global routing, channel definition, channel ordering* i *detailed routing*; svaka od navedenih faza se razmatra posebno zbog velike kompleksnosti. Prednost rutiranja po kanalima je pre svega u efikasnosti i kvalitetu; takođe, razvijeni algoritmi za rutiranje po kanalima garantuju uspešan završetak rutiranja.

3.2.1. 3D raspoređivanje

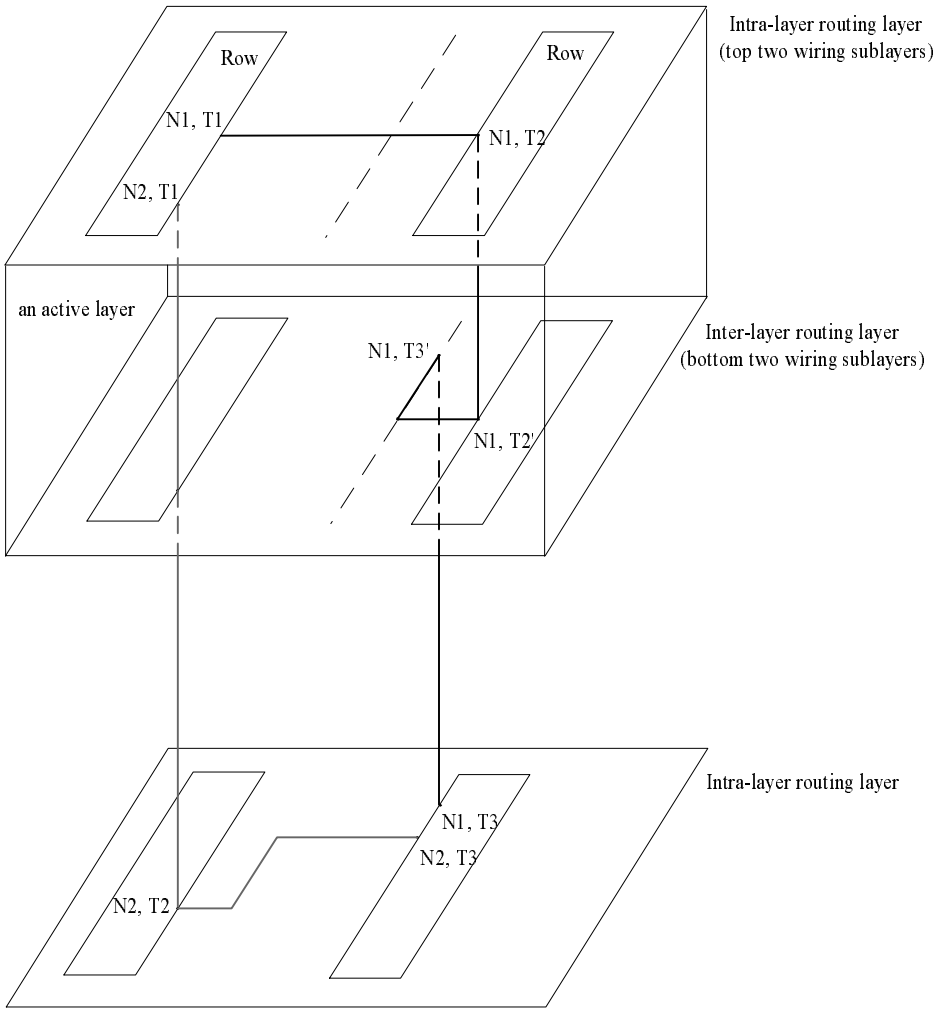
Tong se u svom istraživanju nije posebno bavio problemom 3D raspoređivanja. U svom radu krenuo je od pretpostavke da je faza raspoređivanja već urađena i dalje je razmatran samo problem 3D rutiranja. Međutim, problem raspoređivanja u 3D čipu je posebno važan jer faza raspoređivanja može značajno uticati na performansu fabrikovanog čipa.

U ovom istraživanju problem raspoređivanja u 3D čipu se ne razmatra posebno. Raspoređivanje komponenti po različitim aktivnim ravnima bazira se na jednostavnim heuristikama koje su deo predloženih rešenja. Raspoređivanje komponenti u okviru pojedinih ravni radi se automatski i bazira se na algoritmima koji su implementirani u korišćenim alatima za projektovanje 2D VLSI čipova.

3.2.2. 3D rutiranje

Dve osnovne metodologije rutiranja za 2D VLSI mogu se primeniti i na 3D VLSI. Na primer, može se koristiti 3D globalni ruter koji je baziran na traženju minimalnog razapinjućeg stabla u 3D prostoru. Međutim, verovatnoća da rutiranje bude uspešno završeno sada postaje od još većeg značaja, imajući u vidu veličinu i kompleksnost problema rutiranja u 3D prostoru, a posebno kompleksnost eventualnog ručnog rutiranja u 3D prostoru u slučaju neuspešnog završetka automatskog rutiranja. Opredeljujući se za efikasne i pouzdane algoritme predložena je 3D metodologija rutiranja po kanalima, bazirana na postojećoj 2D metodologiji. Pretpostavljeno je da se oblast za rutiranje može podeliti na 2D kanale u okviru svakog aktivnog nivoa 3D strukture i na 3D kanale između različitih aktivnih nivoa i da se detaljno rutiranje vrši u ovim kanalima.

Predložena metodologija 3D rutiranja po kanalima bazira se na dekompoziciji problema rutiranja na dva potproblema: (a) rutiranje unutar jednog aktivnog nivoa (*intra-layer routing*) i (b) rutiranje između različitih aktivnih nivoa (*inter-layer routing*). Rutiranje u okviru jednog aktivnog nivoa podrazumeva povezivanje terminala u svakom aktivnom nivou i svodi se na problem 2D rutiranja, dok drugi potproblem podrazumeva povezivanje terminala iz različitih aktivnih nivoa. U predloženoj metodologiji 3D rutiranja po kanalima problem povezivanja terminala iz različitih aktivnih nivoa svodi se na problem 2D rutiranja. Definicije osnovnih pojmova 3D rutiranja po kanalima prikazane su na slici 13.



Slika 13. Predlog metodologije rutiranja kod 3D čipova.

Opis: Na slici je prikazan način povezivanja terminala koji pripadaju različitim aktivnim nivoima. Terminali sa oznakama N1-T1, N1-T2 i N1-T3 formiraju vezu N1, a terminali N2-T1, N2-T2 i N2-T3 formiraju vezu N2. **Definicije:** U tekstu koji sledi date su definicije najvažnijih pojmova vezanih za 3D metodologiju rutiranja.

- 3D mreža: 3D mreža je skup povezanih terminala (portova standardnih ćelija) koji su raspoređeni u više aktivnih nivoa. Na slici su prikazane dve 3D mreže sa oznakama N1 i N2.
- 3D rutiranje: povezivanje terminala koji pripadaju istoj 3D mreži uz poštovanje pravila povezivanja (*Layout Rules*).
- Rutiranje unutar nivoa (*Intra-Layer Routing*): povezivanje terminala koji se nalaze u istom aktivnom nivou; ovaj problem se svodi na tradicionalno 2D rutiranje.
- Povezivanje između različitih aktivnih nivoa (*Inter-Layer Routing*): povezivanje terminala koji pripadaju različitim aktivnim nivoima.
- 3D kanal: 3D kanal uključuje preklopljeni deo dva 2D kanala iz dva susedna aktivna nivoa sa pripadajućim terminalima. Takođe, u 3D kanal je uključen i 2D kanal iz oblasti za povezivanje susednih aktivnih nivoa (*Inter-layer Routing Layer*). Terminali iz susednih aktivnih nivoa koji treba da budu povezani (u datom primeru N1-T2 i N1-T3) mapiraju se na granice 2D kanala u oblast za povezivanje susednih aktivnih nivoa. Nakon mapiranja problem se svodi na povezivanje terminala (u posmatranom primeru N1-T2' i N1-T3') u 2D kanalu. Na ovaj način, 3D rutiranje u kanalu svodi se na problem 2D rutiranja unutar jednog nivoa i na transformisanje problema rutiranja između različitih nivoa na 2D rutiranje.

Metodologija 3D rutiranja po kanalima definiše se kao metodologija koja: (a) koristi 2D rutiranje za povezivanje terminala u okviru jednog aktivnog nivoa i (b) transformiše problem rutiranja terminala iz različitih nivoa u problem 2D rutiranja. Predloženo 3D rutiranje po kanalima sastoji se od sledećih koraka:

1. 2D rutiranje po kanalima u svakom aktivnom nivou sa ciljem da se proceni širina 2D kanala za povezivanje;
2. definisanje 3D kanala za svaku od 3D mreža;
3. dodavanje pseudo terminala, ako je potrebno;
4. 2D rutiranje u kanalu u svakom aktivnom nivou sa ciljem da se odredi širina 2D kanala;
5. poravnavanje 2D kanala, ako je potrebno;
6. transformacija iz 3D u 2D koja uključuje selekciju terminala, mapiranje i razrešavanje konflikata;
7. određivanje redosleda po kojem će kanali biti rutirani;
8. detaljno 2D rutiranje u *intra-* i *inter-routing layer* nivoima;
9. poravnavanje 3D kanala ako je potrebno,
10. optimizacija.

Svaki od navedenih koraka u predloženom algoritmu 3D rutiranja po kanalima realizuje se nezavisno zbog njihove kompleksnosti. Neki od koraka u predloženom algoritmu su isti ili slični već postojećim algoritmima za 2D rutiranje. Detaljan opis tačke šest, koja predstavlja glavni korak u predloženom algoritmu za 3D rutiranje, dat je u radu [Tong95].

4. Predloženo rešenje i njegova suština

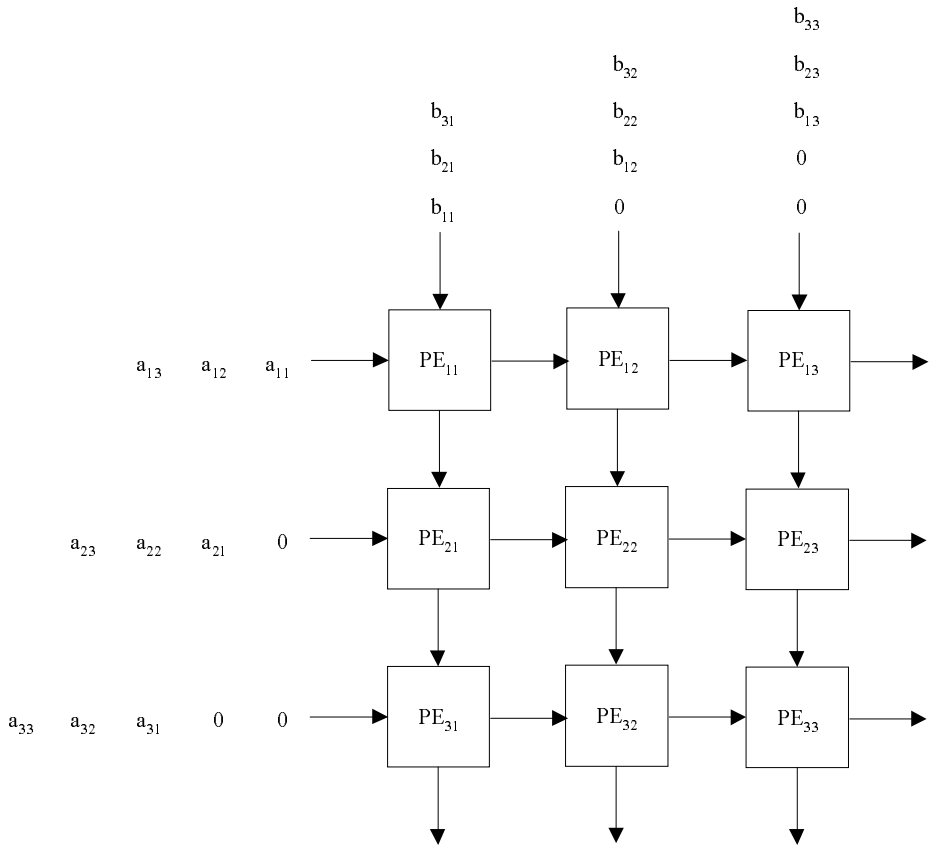
Kao što je već naglašeno, cilj ovog istraživanja je da omogući kvantitativnu analizu dužina veza i evaluaciju skraćivanja dužina veza prilikom prelaska sa 2D na 3D VLSI metodologiju projektovanja čipova. Suština predloženog rešenja je u sledećem: polazi se od Tong-ovog pristupa po kome se 3D raspoređivanje i povezivanje bazira na 2D raspoređivanju i povezivanju. Taj princip se dalje razrađuje u pravcu stvaranja konkretnog alata koji omogućava kvantitativnu analizu skraćenja veza za konkretan zadati dizajn.

Istraživanje je bazirano na sistoličkim poljima, pre svega zbog njihove regularne strukture i pogodnosti za realizaciju, kako u 2D VLSI, tako i u 3D VLSI metodologiji projektovanja čipova. Urađeno je nekoliko eksperimenata baziranih na različitim tipovima sistoličkih polja koja su opisana u radovima [Lakh96], [Lakh97].

Eksperimenti su bazirani na sistoličkom polju za matrično množenje. Pretpostavimo da su date dve kvadratne matrice A i B dimenzija NxN; treba izračunati matricu C, tako da je C=A·B. Algoritam matričnog množenja je prikazan na slici 14. Iz algoritma sledi da je osnovna operacija koju izvršava svaka ćelija sistoličkog polja oblika $c \leftarrow c + a*b$. Na slici 15 je prikazano sistoličko polje za množenje dve kvadratne matrice za slučaj da je N=3. Blok šema procesnog elementa posmatranog sistoličkog polja prikazana je na slici 16.

```
for(i=1; i≤N; i++)
   for(j=1; j≤N; j++)
      for(k=0; k≤N; k++)
         c[i,j] = c[i,j] + a[i,k]·b[k,j]
```

Slika 14. Algoritam matričnog množenja.

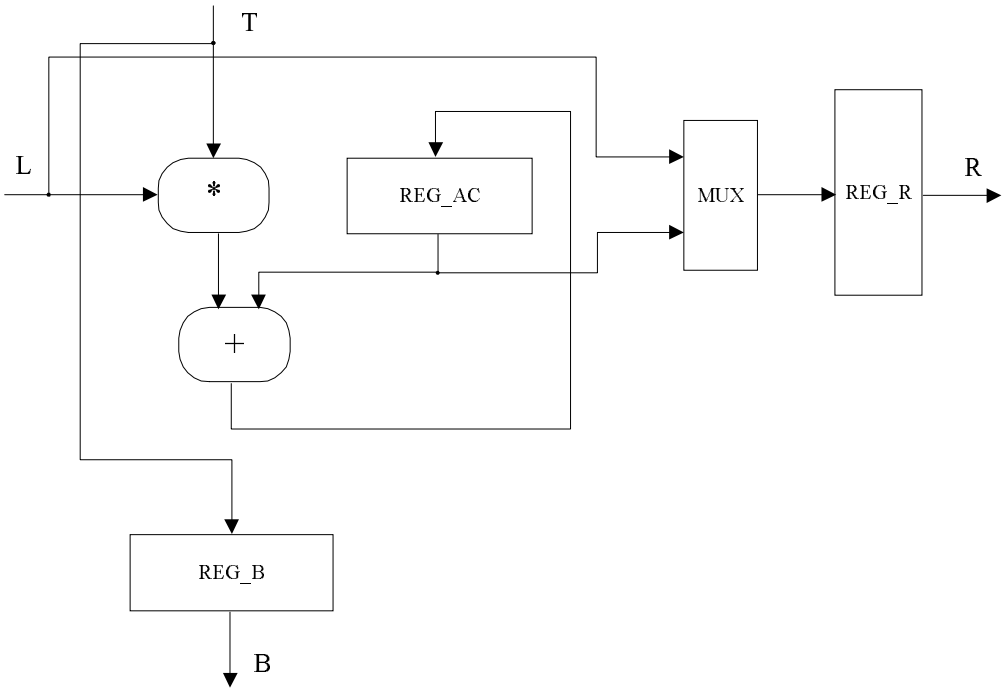


Slika 15. Sistoličko polje za množenje dve 3x3 matrice.

Opis: Za realizaciju matričnog množenja koristi se kvadratno sistoličko polje sa devet procesnih elemenata. Svakom elementu rezultujuće matrice C pridružen je po jedan procesni element. Procesni element PEij obavlja izračunavanje elementa c_{ij} rezultujuće matrice C. Sistoličko polje se može naći u jednom od dva režima rada: (a) u režimu izračunavanja i (b) u režimu iznošenja rezultata.

U režimu izračunavanja podaci se dovode graničnim procesnim elementima na način koji je prikazan na slici. Svaki procesni element prihvata podatke koji dolaze na njegove ulaze, izračunava njihov proizvod, sabira ga sa sadržajem lokalnog registra i rezultat ponovo smešta u lokalni registar. Prihvaćeni podaci koji pristižu s leva i s vrha prosleđuju se susednim procesnim elementima koje se nalaze desno i ispod uočenog procesnog elementa, redom. Za izračunavanje elementa c_{11} potrebno je N ciklusa takta, za elemente c_{12} i c_{21} potrebno je (N + 1) ciklusa takta, itd. U opšem slučaju, element c_{NN} izračunava se nakon (3*N - 2) ciklusa takta.

Nakon faze izračunavanja potrebno je izračunate elemente matrice C izneti iz sistoličkog polja. Iznošenje rezultata može se realizovati na različite načine. U ovom primeru uzeto je da nakon faze izračunavanja svaki procesni element prosleđuje rezultat svom desnom susedu. Svaki procesni element nakon što je svoj rezultat prosledio svom desnom susedu, prihvata podatke koje mu dolaze od levog suseda i prosleđuje ih desnom susedu. U opšem slučaju, za fazu iznošenja rezultata potrebno je N ciklusa takta.



Slika 16. Blok šema procesnog elementa.

Opis: Procesni element sistoličkog polja za množenje matrica uključuje množač, sabirač, akumulacioni registar REG_AC, i dva prenosna registra REG_B i REG_R. U svakom trenutku procesni element se nalazi u jednom od dva moguća režima rada: (a) u režimu izračunavanja i (b) u režimu iznošenja rezultata.
U režimu izračunavanja, u svakom ciklusu takta prihvataju se podaci koji pristižu po linijama T i L, izračunava se njihov proizvod koji se sabira sa sadržajem registra REG_AC i dobijeni zbir se smešta ponovo u registar REG_AC. Podatak koji dolazi po linijama T pamti se u registru REG_B, a podatak koji dolazi po linijama L pamti se u registru REG_R. U režimu iznošenja podataka sadržaj registra REG_AC se prebacuje u registar REG_R. Multiplekser omogućuje iznošenje rezultata iz registra REG_AC, pa se na osnovu toga iznošenje rezultata ostvaruje slanjem izračunatog podatka susednom procesnom elementu koji je smešten desno od posmatranog procesnog elementa.

U okviru svakog eksperimenta predloženo je više kandidat 3D arhitektura. Svaka od predloženih arhitektura realizovana je korišćenjem postojećih alata za 2D VLSI uključujući i klasičnu 2D realizaciju posmatranog sistoličkog polja. Realizacija 3D strukture podrazumeva formiranje opisa u CIF formatu posebno za svaki od aktivnih nivoa u 3D čipu. Kvantitativna analiza dužina veza za svako od predloženih rešenja i postojeće rešenje vrši se korišćenjem originalno razvijenog programskog paketa za analizu dužina veza MARS. Kao rezultat analize dobija se raspodela dužina veza na osnovu koje se vrši poređenje postojećih i predloženih rešenja. Definicija izvedenih eksperimenata data je u tekstu koji sledi.

Eksperiment #1

U okviru eksperimenta #1 posmatraju se različite realizacije kvadratnog sistoličkog polja za množenje matrica sa četiri četvorobitna procesna elementa. Porede se dužine veza u okviru jezgra čipa za slučajeve: (a) 2D realizacije, (b) 3D realizacije sa dva aktivna nivoa, i (c) 3D realizacije sa četiri aktivna nivoa.

Eksperiment #2

Eksperiment #2 baziran je takođe na sistoličkom polju za množenje matrica sa četiri četvorobitna procesna elementa. Porede se dužine veza u slučajevima 2D i 3D realizacije sa dva aktivna nivoa, s tim što se posmatraju realizacije koje podrazumevaju korišćenje ulazno/izlaznih pedova, tj. posmatraju se i veze koje povezuju ulazne i izlazne terminale standardnih ćelija sa ulaznim, odnosno izlaznim pedovima, redom.

Eksperiment #3

U okviru eksperimenta #3 porede se dužine veza u okviru jezgra čipa za slučajeve (a) 2D realizacije, (b) 3D realizacije u dva aktivna nivoa i (c) 3D realizacije u četiri aktivna nivoa sistoličkog polja za množenje matrica sa četiri osmobitna procesna elementa.

Eksperiment #4

U okviru eksperimenta #4 porede se dužine veza u okviru jezgra čipa za slučajeve (a) 2D realizacije i (b) 3D realizacije u tri aktivna nivoa sistoličkog polja sa devet četvorobitnih procesnih elemenata.

Eksperiment #5

U prethodnim eksperimentima razmeštanje u 3D čipu vršeno je tako da se različiti procesni elementi ili grupe procesnih elemenata nalaze u različitim aktivnim nivoima. Sledeći, kvalitativno novi korak, podrazumeva realizaciju samih procesnih elemenata u više aktivnih nivoa. Posmatra se sistoličko polje za množenje matrica sa četiri četvorobitna procesna elementa. Procesni elementi su realizovani sa dva stepena protočne obrade. Porede se dužine veza u okviru jezgra čipa za slučajeve: (a) 2D realizacije i (b) 3D realizacije u dve ravni, pri čemu je prvi stepen protočne obrade realizovan u prvom aktivnom nivou 3D strukture, dok je drugi stepen protočne obrade realizovan u drugom aktivnom nivou 3D strukture.

****

U okviru svakog eksperimenta predložena je jedna ili više 3D realizacija, koje se razlikuju po broju aktivnih ravni ili po načinu razmeštanja komponenti u 3D strukturi. Pri tome, posmataju se dva pristupa u razmeštanju: (a) u jednom aktivnom nivou razmešta se jedan ili grupa procesnih elemenata, što je slučaj u prva četiri eksperimenta i (b) sami procesni elementi su realizovani u više aktivnih nivoa, tj. svaka aktivna ravan sadrži deo svakog od procesnih elemenata koji čine sistoličko polje. U eksperimentima se porede dužine veza i površine čipova predloženih 3D i postojećih 2D realizacija.

5. Uslovi i pretpostavke

U ovom poglavlju su dati osnovni uslovi i pretpostavke pod kojima je rađeno istraživanje. Generalno, uslovi se odnose na specifikaciju realnog okružanja. Pretpostavke definišu uprošćenja koja analizu čine mogućom i/ili manje kompleksnom, a da pri tom nema negativnih posledica na tačnost rezultata. Uslovi i pretpostavke su dati kroz definiciju okruženja u kojem se radi analiza, kao i kroz specifikaciju korišćenih alata i tehnologija. Na kraju, dat je kratak pregled osnovnih definicija CIF formata.

5.1. Definicija okruženja u kome se vrši analiza

Kao što je već naglašeno, ovo istraživanje je bazirano na predloženoj metodologiji 3D rutiranja po kanalima koja je prikazana u trećem poglavlju [Tong95]. Predložena metodologija podrazumeva *standard cell* ili *gate array* metodologije projektovanja VLSI čipova. U ovom istraživanju usvojena je *standard cell* metodologija projektovanja VLSI čipova. Međutim, za sada nema raspoloživih alata za automatsko raspoređivanje sa povezivanjem u 3D čipu. Kako se predložena metodologija 3D rutiranja po kanalima bazira na postojećim 2D metodologijama, u ovom istraživanju korišćeni su postojeći alati za projektovanje integrisanih kola po *standard cell* metodologiji projektovanja 2D VLSI čipova. Proces raspoređivanja sa povezivanjem u slučaju 3D struktura realizuje se posebno za svaku aktivnu ravan u 3D strukturi.

Dostupni alati za projektovanje 2D čipova po *standard cell* pristupu ne poseduju mogućnost analize dužina veza. U tom cilju razvijen je programski paket MARS za analizu dužina veza na 2D čipu koji je opisan u CIF formatu. Programski paket MARS prihvata opis jednog 2D čipa ili jednog aktivnog nivoa 3D čipa, a kao izlaz daje raspodelu dužina veza.

Pretpostavljeno je da je minimalana dužina 3D veza (veze koje povezuju terminale iz različitih aktivnih nivoa) jednaka minimalnoj dužini 2D veza (veze koje povezuju terminale u okviru jednog aktivnog nivoa) u okviru aktivnih ravni. Takođe, pretpostavlja se da je maksimalna dužina 3D veza jednaka maksimalnoj dužini 2D veza. Generalno, dužina 3D veza je u velikoj meri određena metodologijom razmeštanja u 3D strukturi. Usvojene pretpostavke o dužini 3D veza garantuju da dobijeni rezultati eksperimenata predstavljaju gornju granicu rezultata koji se mogu očekivati u praksi.

5.2. Lista alata i tehnologija u kojima se vrši analiza

Generalno, projektovanje VLSI čipova odvija se kroz tri faze: (a) unošenje šeme, (b) testiranje logike i vremenskih odnosa i (c) raspoređivanje sa povezivanjem. U analizi su korišćeni postojeći alati za projektovanje VLSI čipova po *standard cell* metodologiji. Za unošenje šeme koristi se programski paket OrCAD. Kao izlaz iz faze unošenja šeme dobija se

net lista koja se dalje koristi u fazi testiranja logike i vremenskih odnosa i fazi raspoređivanja sa povezivanjem. Testiranje logike i vremenskih odnosa se vrši programskim alatom GateSim. Kao alat za poslednju fazu projektovanja VLSI čipova koristi se program L-Edit, odnosno deo programa koji obavlja automatsko raspoređivanje sa povezivanjem. Kao rezultat dobija se fajl koji sadrži opis svih nivoa maski čipa u CIF formatu.

Dobijeni opis čipa u CIF formatu koristi se kao ulaz u programski paket za analizu dužina veza MARS. Proces analize veza odvija se u četiri koraka: (a) izdvajanje relevantnih nivoa maski koje definišu veze, (b) ekspandovanje definicija CIF simbola, (c) izdvajanje relevantnih geometrijskih primitiva po tipovima i (d) određivanje setova povezanih geometrijskih primitiva, određivanje i analiza veza u okviru svakog seta. Kao izlaz iz svake od pomenutih faza dobija se opis koji odgovara CIF formatu, tako da se na kraju svakog koraka rezultati mogu vizuelno verifikovati korišćenjem programskog paketa L-Edit.

5.3. CIF format

Programski paket za analizu dužina veza prihvata opis čipa u CIF formatu, pa su u ovom poglavlju date osnovne definicije od interesa za razumevanje algoritama koji se koriste u programskom paketu za analizu veza.

CIF (*Caltech Intermediate Format*) je ASCII format namenjen razmeni informacija o geometrijskoj strukturi svih nivoa maski između dizajnera integrisanih kola i proizvođača. CIF format je prvi put definisan u knjizi *Introduction to VLSI* [Mead80]. Opis CIF formata u BNF notaciji dat je u Apendiksu #1. U ovom poglavlju dat je opis najvažnijih komandi.

Definicija CIF simbola započinje komandom oblika:

DS nnn a b;

pri čemu je *nnn* broj koji je pridružen posmatranom CIF simbolu, dok *a* i *b* predstavljaju opcione faktore skaliranja. Sve komande koje slede komandu DS i prethode komandi DF definišu posmatrani simbol. Osnovna jedinica mere CIF formata je 0.01 mikron, a faktori skaliranja omogućuju definisanje svih vrednosti u mikronima umesto u stotim delovima mikrona. Naime, sve vrednosti unutar definicije se množe sa faktorom *a*, a rezultat se deli sa faktorom *b*.

Jednom definisani CIF simbol može se pozvati komandom *C* (*call*). Komanda *C*, pored simbola koji se poziva, opciono omogućuje definisanje transformacije koja se primenjuje na posmatrani simbol:

C nnn transformation;

gde *nnn* predstavlja broj simbola koji se poziva, a *transformation* definiše transformaciju koja se primenjuje na pozvani simbol. Jedna transformacija može biti sačinjena od nekoliko translacija, ogledanja i rotacija. Ukoliko je definisano više transformacija primenjuju se redom s leva na desno. U tabeli 1. dato je nekoliko primera komandi poziva ranije definisanog simbola sa transformacijom uz odgovarajuća objašnjenja.

Komanda	Objašnjenje
C 55 T -100 10;	poziv simbola 55, koji se translira za 100 jedinica u negativnom smeru po x osi i za 10 jedinica po y osi
C 99 MX;	poziv simbola 99 na koji treba primeniti operaciju ogledanja u odnosu na Y osu, tj. X koordinate treba pomnožiti sa -1
C 22 MY;	poziv simbola 22 na koji treba primeniti operaciju ogledanja u odnosu na X osu, tj. Y koordinate treba pomnožiti sa -1
C 44 R 0 1;	poziv simbola 44 koji treba rotirati za 90 stepeni

Tabela 1. Primeri *Call* instrukcija sa objašnjenjima.

CIF format poseduje četiri komande za kreiranje geometrijskih primitiva: pravougaonici (*boxes*), poligoni (*polygons*), krugovi (*flashes or circles*), i žice (*wires*). U analizi koja je vršena od interesa je jedino komanda kojom se definiše geometrijska primitiva tipa *box*. U daljem tekstu dat je primer definicije geometrijske primitive tipa *box* sa pripadajućim objašnjenjima:

B 176 6 -144 98.

Definicija geometrijske primitive tipa *box* započinje komandom *B*; prva dva parametra specificiraju dimenzije primitive po x i y osi, redom, a druga dva parametra određuju koordinate centra primitive.

Svaka geometrijska primitiva mora biti pridružena jednom od nivoa maski za fabrikaciju. Nivo maske se definiše komandom:

L layername.

Broj i vrsta nivoa maski određeni su tehnologijom koja je podržana. U eksperimentima korišćena je CMOS SCN 2.0μm tehnologija.

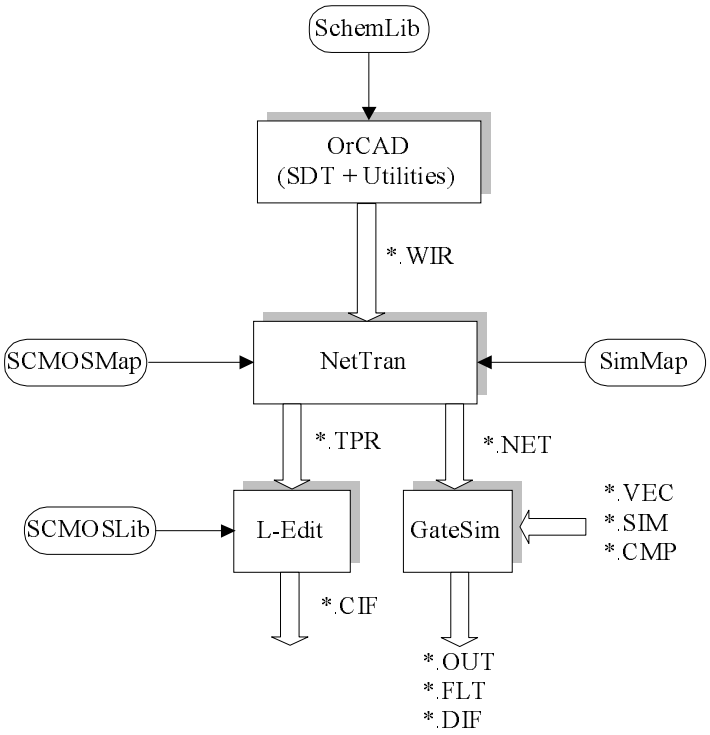
Programski paket za analizu dužina veza MARS baziran je na CIF formatu koji je podržan u programu L-Edit. Naime, CIF format podržan od programa L-Edit uvodi neka ograničenja i proširenja u odnosu na format koji je definisan standardom. Međutim, odstupanja su neznatna i ne umanjuju primenjivost programskog paketa MARS. Detalji o restrikcijama i proširenjima CIF formata koji je podržan programom L-Edit dati su u [Tann90].

6. Alati

U ovom radu korišćen je programski paket *Tanner*, za projektrovanje VLSI sistema po *standard cell* metodologiji, i originalno razvijeni programski paket *MARS*, za analizu dužina veza na čipu koji je dat u CIF formatu. Ovo poglavlje sastoji se od tri dela. U prvom delu data je osnovna struktura programskog paketa *Tanner*. U drugom delu prikazan je način formiranja CIF opisa čipa upotrebom programa za razmeštanje sa povezivanjem. U trećem delu prikazana je struktura programskog paketa *MARS* za analizu veza na čipu.

6.1. Programski paket Tanner

Programski paket Tanner uključuje programske alate NetTran, GateSim i L-Edit, i biblioteke SchemLib, SimMap, SCMOSMap i SCMOSLib. Kao programski alat za unošenje i verifikaciju šeme koristi se OrCAD. Projektovanje VLSI čipova po *standard cell* metodologiji korišćenjem programskih paketa OrCAD i Tanner, kao i opis strukture programskog paketa Tanner prikazani su na slici 17.



Slika 17. Struktura programskog paketa Tanner i pristup projektovanju VLSI čipova.

Opis: Programski paket Tanner sadrži programske alate OrCAD, GateSim i L-Edit. Programski alat OrCAD uključuje OrCAD SDT (*Shematic Design Tool*) program za unošenje šeme čipa koji se projektuje, i OrCAD *Utilities* programe koji služe za verifikaciju ispravnosti unete šeme i generisanje izlaznih net-listi. Program NetTran translira ulaznu netlistu (*.WIR) koristeći biblioteke SimMap i SCMOSMap u formate koji su potrebni programskim paketima GateSim (*.NET) i L-Edit (*.TPR), redom. Programski alat GateSim omogućuje logičku simulaciju na bazi opisa koji je dat u fajlu *.NET, ulaznih test vektora koji se nalaze u fajlu *.VEC (ili *.CMP) i komandi koje kontrolišu simulaciju zadatih u fajlu *.SIM. Rezultati logičke simulacije dobijaju se u fajlu *.OUT (ili *.FLT, ili *.CMP). L-Edit je *layout editor* koji služi za razmeštanje i povezivanje logičke šeme koja je data ulaznim fajlom (*.TPR), a kao izlaz generiše fajl u CIF formatu koji se može slati na fabrikaciju (*.CIF).

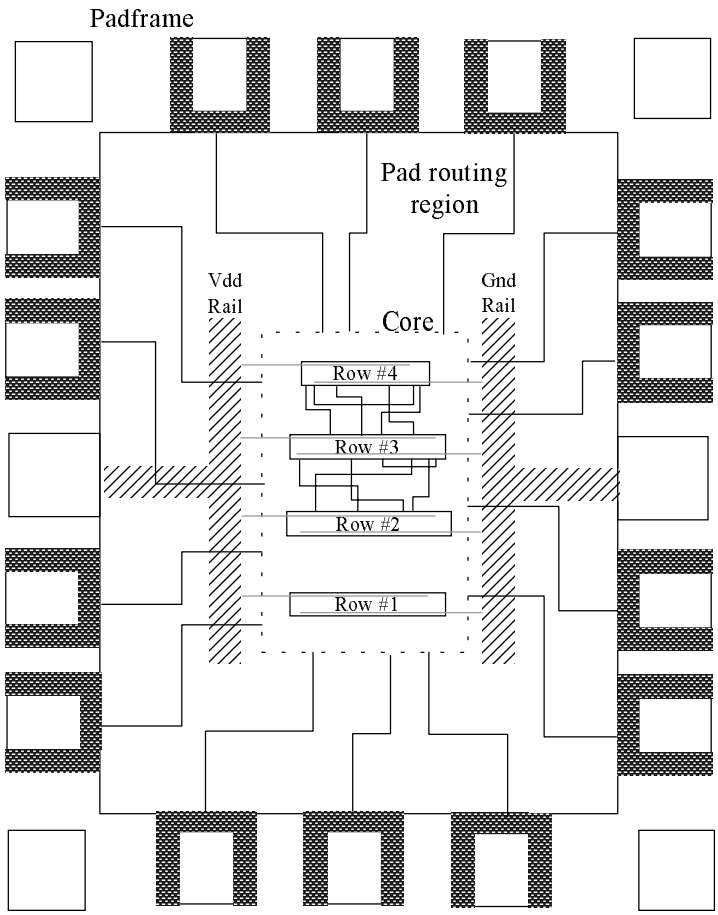
Programski paket Tanner uključuje sledeće biblioteke: SchemLib, SimMap, SCMOSMap i SCMOSLib. SchemLib je biblioteka standardnih ćelija koju koristi OrCAD SDT za unošenje logičke šeme čipa; ova biblioteka je nezavisna od formata pojedinih proizvođača. SimMap je biblioteka makro-definicija standardnih ćelija koju koristi program NetTran da bi pripremio fajl (*.NET) koji se koristi kao ulaz u paket GateSim. SCMOSMap je biblioteka makro-definicija standardnih ćelija koju koristi program NetTran da bi pripremio fajl (*.TPR) koji se koristi kao ulaz u paket GateSim. SCMOSLib je *layout standard cell* biblioteka koju koristi program L-Edit.

U nastavku dat je opis rada sa paketom Tanner po koracima:

(a) unošenje šeme pomoću OrCAD SDT programskog alat i biblioteke SchemLib;
(b) proveravanje unete šeme korišćenjem uslužnih alata (Annotate, CleanUp, Erc);
(c) generisanje netlist fajla (*.WIR) u *wirelist* formatu korišćenjem OrCAD Netlist uslužnog programa;
(d) translacija fajla *.WIR u fajl *.NET za logičku simulaciju pomoću NetTran programa i SimMap biblioteke;
(e) priprema potrebnih fajlova *.SIM, *.VEC, *.CMP za logičku simulaciju;
(f) logička simulacija fajla *.NET pomoću GateSim paketa;
(g) translacija fajla *.WIR u fajl *.TPR, koji se koristi kao ulaz u program L-Edit, pomoću NetTran paketa i biblioteke SCMOSMap.
(h) kreiranje fajla *.CIF pomoću programa L-Edit i SCMOSLib biblioteke.

6.2. L-Edit: razmeštanje i povezivanje

U ovom poglavlju razmatra se tipičan dizajn jednog čipa. Poseban alat SPR (*Standard Cell Place and Route*) u okviru programa L-Edit omogućuje automatsko generisanje svih nivoa maski čipa (*chip layout*) iz opisa koji je zadat net listom. SPR alat se sastoji od sledećih modula: *Block Generator*, *Padframe Generator* i *Pad Router*. Tipično, čip se sastoji iz tri dela: jezgro (*core*), okvir čipa (*padframe*) i oblast koja povezuje jezgro i okvir čipa (*padframe routing region*). Jezgro čipa sadrži "srce" dizajna, odnosno samu funkcionalnu logiku i smešteno je unutar okvira čipa koji sadrži pedove (*pads*). Pedovi omogućuju povezivanje i zaštitu logike unutar čipa i pinova na kućištu. Struktura čipa projektovanog po *standard cell* metodologiji prikazana je na slici 18.



Slika 18. Struktura čipa projektovanog po *standard cell* metodologiji.

Opis: Tipično, VLSI čip se sastoji od jezgra čipa (*core*), okvira čipa (*padframe*) i oblasti koja povezuje jezgro i okvir čipa (*padframe routing region*).
Jezgro čipa se sastoji od kanala u kojima se razmeštaju standardne ćelije (*rows*) i kanala koji sadrže interkonekcije (veze) kojima su povezane standardne ćelije (*channels*). Interkonekcije u kanalima za povezivanje standardnih ćelija su realizovane, tipično, u dva nivoa maske Metal1 i Metal2. Svi horizontalni segmenti veza pripadaju jednom nivou maske, na primer nivou Metal1, a svi vertikalni segmenti veza pripadaju drugom nivou maske, u ovom slučaju nivou Metal2. Na slici je pokazano i razvođenje signala Vdd i Gnd do svakog kanala koji sadrži standardne ćelije.
Oblast za povezivanje jezgra i okvira čipa sadrži interkonekcije koje povezuju ulazne odnosno izlazne terminale standardnih ćelija sa pedovima na okviru čipa. Tipično, segmenti veza koji su razmešteni u oblasti za povezivanje jezgra sa okvirom čipa izvedeni su u nivou Metal2.
Okvir čipa sadrži pedove preko kojih se odvija komunikacija čipa sa spoljašnjim svetom. Postoje tri vrste pedova: ulazni, izlazni i bidirekcioni, kao i pedovi koji su povezani sa signalima Vdd i Gnd.

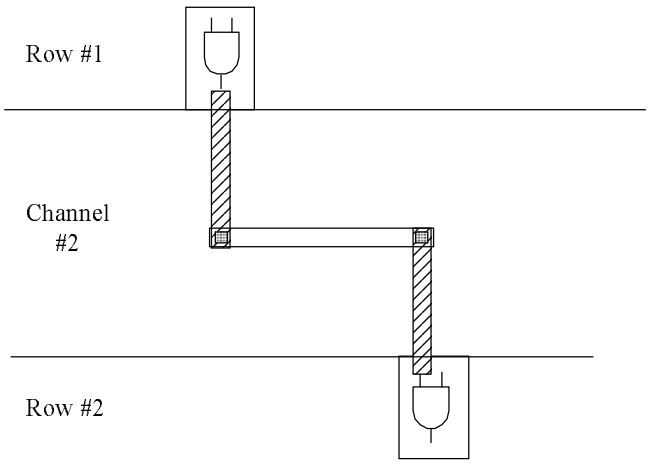
U daljem tekstu prikazani su relevantni delovi CIF opisa jednog čipa. Na slici 19 prikazano je zaglavlje jednog tipičnog dizajna. Na slici 20 prikazani su relevantni detalji CIF opisa jednog kanala za povezivanje standardnih ćelija. Slika 21 prikazuje relevantne elemente definicije jedne standardne ćelije na primeru dvoulaznog Xor kola, dok je na slici 22 dat primer definicije jednog kanala u kome su razmeštene standardne ćelije. Slike 23 i 24 prikazuju relevantne detalje jezgra i okvira čipa, redom, dok slika 25 sadrži definiciju kompletnog čipa i deo kojim se instancira ceo čip.

```
(CIF written by the Tanner Research layout editor, L-Edit);
(Version: 3.10);
(Serial No. 01631);
(TECHNOLOGY: SCN);
(DATE: 19 Jan 1996);
(FABCELL: Chip  1024 x 1424 Microns);
(L-Edit Layer Poly = CIF Layer CPG);
(L-Edit Layer Active = CIF Layer CAA);
(L-Edit Layer Metal1 = CIF Layer CMF);
(L-Edit Layer Metal2 = CIF Layer CMS);
(L-Edit Layer N Select = CIF Layer CSN);
(L-Edit Layer N Well = CIF Layer CWN);
(L-Edit Layer Poly Contact = CIF Layer CCP);
(L-Edit Layer ActiveContact = CIF Layer CCA);
(L-Edit Layer Via = CIF Layer CVA);
(L-Edit Layer Overglass = CIF Layer COG);
(L-Edit Layer Pad Comment = CIF Layer XP);
(L-Edit Layer Icon/Outline = CIF Layer CX);
```

Slika 19. Zaglavlje CIF opisa 4-bitnog CLA sabirača koji je dizajniran po SCN tehnologiji.

Opis: Pored opštih informacija kao što su verzija programa, datum i serijski broj, zaglavlje specificira pre svega tehnologiju fabrikovanja, površinu koju zauzima projektovani čip i nazive svih relevantnih nivoa maski. SCN označava *standard cell CMOS N-Well* tehnologiju.

```
(SCALING: 1 CIF Unit = 1/2 Lambda, 1 Lambda = 1/1 Microns);
DS2 100 2;
9 Channel~1;
LCMF;
B 176 6 -14 -219;
...
B 8 8 130 -236;
LCMS;
B 8 8 286 -236;
...
B 8 8 130 -236;
LCVA;
B 4 4 130 -236;
...
B 4 4 286 -236;
DF;
```



Slika 20. Elementi definicije jednog kanala za povezivanje.

Opis: Sve interkonekcije unutar jednog kanala realizovane su u dva nivoa (LCMF - Metal1, LCMS - Metal2). Veze između elemenata u LCMF i LCMS nivou ostvarene su preko VIA elemenata koji su navedeni iza LCVA komande. Na slici je pokazan primer realizacije jedne interkonekcije koja povezuje dve standardne ćelije. Veza je sastavljena od dva vertikalna segmenta koji su realizovani u nivou Metal2, jednog horizontalnog segmenta koji je realizovan u nivou Metal1 i dva *via* elementa preko kojih se ostvaruje veza između dva nivoa.

```
(SCALING: 1 CIF Unit = 1/2 Lambda, 1 Lambda = 1/1 Microns);
DS9 100 2;
9 XOr2;
94 Abut 46 66 CX;
94 A 10 74 CMS;
94 Out 42 98 CMS;
94 B 82 76 CMS;
94 Gnd 0 8 CMF;
94 Gnd 92 8 CMF;
94 Vdd 0 124 CMF;
94 Vdd 92 124 CMF;
94 Cross 26 74 CMS;
94 Cross 62 58 CMS;
94 V3.02 92 132 CX;
LCPG;
B 4 132 18 70;
...
LCAA;
B 10 12 87 126;
...
LCMF;
B 92 16 46 124;
...
LCMS;
B 8 64 42 70;
...
LCSN;
B 92 46 46 37;
...
LCWN;
B 112 76 46 104;
LCCP;
B 4 4 10 62;
...
LCCA;
B 4 4 66 18;
...
LCVA;
B 4 4 42 98;
...
DF;
```

Slika 21. Elementi definicije dvoulaznog Xor logičkog kola.

Opis: Definicija neke standardne ćelije uključuje sledeće relevantne nivoe maski: LCPG, LCAA, LCMF, LCMS, LCSN, LCWN, LCCA i LCVA. Proširenje CIF formata koje unosi L-Edit specificira, između ostalog, i sve relevantne tačke (portove) preko kojih se posmatrana ćelija povezuje sa drugim ćelijama (komande koje počinju sa 94). To proširenje se koristi prilikom izdvajanja portova iz CIF opisa.

```
(SCALING: 1 CIF Unit = 1/2 Lambda, 1 Lambda = 1/1 Microns);
DS11 100 2;
9 Row~1;
C9 T 138 0;
C9 T 46 0;
C9 T -46 0;
C9 T -138 0;
C9 T -230 0;
LCMS;
B 8 6 72 75;
...
DF;
```

Slika 22. Elementi definicije kanala za razmeštanje standardnih ćelija.

Opis: Definicija vrste za smeštanje standardnih ćelija uključuje pozive definicija standardnih ćelija i definisanje geometrijskih primitiva u LCMS nivou za povezivanje terminala (portova) standardnih ćelija sa vezama koje pripadaju kanalima za povezivanje.

```
(SCALING: 1 CIF Unit = 1/2 Lambda, 1 Lambda = 1/1 Microns);
DS12 100 2;
9 Core;
C5 T 0 0;
C4 T 0 0;
C3 T 0 0;
C2 T 0 0;
C8 T 0 520;
C10 T 0 140;
C11 T 0 -208;
94 Abut 14 228 CX;
94 Gnd 320 228 CMF;
94 Vdd -292 228 CMF;
94 U10_P2 184 718 CMS;
94 U9_P2 66 718 CMS;
...
LCMF;
B 54 16 269 528;
...
LCMS;
B 54 6 317 -235;
...
DF;
```

Slika 23. Elementi definicije jezgra čipa.

Opis: Definicija jezgra čipa uključuje pozive ranije definisanih kanala u kojima se vrši razmeštanje standardnih ćelija, i kanala u kojima su definisane veze između standardnih ćelija. U linijama koje počinju sa 94 navedena su mesta na obodu jezgra čipa preko kojih se vrši povezivanje jezgra sa pedovima na okviru čipa.

```
(SCALING: 1 CIF Unit = 1/2 Lambda, 1 Lambda = 1/1 Microns);
DS18 100 2;
9 Frame;
C13 R 0 1 T -600 -1422;
...
C14 T -600 1000;
C14 MX T 200 1000;
C16 MX R 0 1 T -600 -600;
C16 R 0 1 T -600 -600;
C17 R 0 1 T -600 -200;
DF;
```

Slika 24. Elementi definicije okvira čipa.

Opis: Definicija okvira čipa uključuje pozive definicija ulaznih, izlaznih i bidirekcionih pedova, kao i pedova preko kojih se ostvaruje napajanje i uzemljenje.

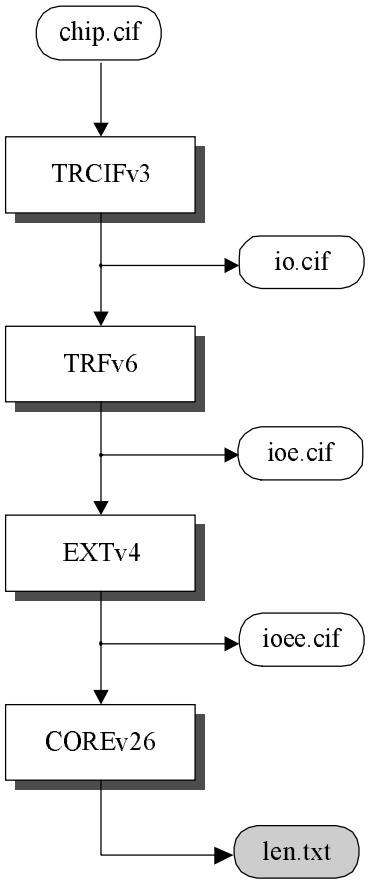
```
(SCALING: 1 CIF Unit = 1/2 Lambda, 1 Lambda = 1/1 Microns);
DS19 100 2;
9 Chip;
C12 T -14 -222;
C18 T 0 0;
LCMF;
B 270 48 465 -78;
B 270 48 -465 -78;
LCMS;
B 6 490 -113 741;
... {definicija elemenata koji povezuju jezgro čipa i okvir čipa}
DF;
C19; (instanciranje dizajna)
E
```

Slika 25. Elementi definicije celog čipa i instanciranje dizajna.

Opis: Definicija kompletnog čipa uključuje poziv opisa jezgra čipa i okvira čipa, kao i definicije geometrijskih primitiva preko kojih je ostvareno povezivanje jezgra sa pedovima na okviru čipa. Kraj definicije praćen je komandom C19 koja instancira definisani simbol 19, a to je simbol koji predstavlja ceo čip.

6.3. MARS - programski paket za analizu dužina veza

U cilju evaluacije skraćivanja dužina veza prilikom prelaska sa 2D na 3D VLSI tehnologiju razvijen je originalni softverski paket za analizu dužina veza na 2D čipu čiji je opis dat u CIF formatu. Programski paket za analizu dužina veza MARS može se koristiti i za druge namene, na primer za poređenje različitih alata za raspoređivanje sa povezivanjem. Struktura programskog paketa za analizu veza data je na slici 26.



Slika 26. Struktura programskog paketa za analizu veza na čipu.

Opis: Program TRCIFv3 prihvata CIF opis čipa koji se analizira i izdvaja sve relevantne informacije koje su potrebne za određivanje veza. Izlaz programa TRCIFv3 prosleđuje se programu TRFv6 koji vrši instanciranje relevantnih delova standardnih ćelija i formira fajl koji sadrži opis svih geometrijskih primitiva koje su relevantne za određivanje veza u nivoima LCMF, LCMS, LCVA. Program EXTv4 vrši izdvajanje geometrijskih primitiva po tipu, u LCVA nivou, koje mogu biti: (a) *via* elementi, (b) ulazni terminali i (c) izlazni terminali. Program COREv26 određuje skupove povezanih geometrijskih primitiva, pronalazi interkonekcije u okviru skupa i izračunava dužinu veza i drugu potrebnu statistiku.

U daljem tekstu dat je detaljan opis programa koji čine programski paket MARS za analizu dužina veza.

TRCIFv3

Program TRCIFv3 prihvata opis čipa u CIF formatu i izdvaja samo relevantne informacije neophodne za određivanje veza. U tabeli 2 navedeni su nivoi maski koje sadrže opise geometrijskih primitiva značajnih za određivanje veza u zavisnosti od vrste simbola.

Simboli	Nivoi maske (*Fabrication Layers*)
Kanali za povezivanje (*Channels*)	LCMF, LCMS, LCVA
Kanali za smeštanje standardnih ćelija (*Rows*)	LCMS
Jezgro čipa (*Core*)	LCMS
Čip (*Chip*)	LCMS
Standardne ćelije (*Cells: gates, flip-flops,...*)	LCVA (ulazni terminali + izlazni terminal)

Tabela 2. Nivoi maske koji su relevantni za određivanje veza u zavisnosti od vrste simbola.

Pored toga, program TRCIFv3 vrši i dodatnu analizu koja ima za cilj da identifikuje terminale (portove) standardnih modula i njihov tip (ulazni ili izlazni). Dobijeni fajl (io.cif) je, takođe u CIF formatu, tako da se može koristiti kao ulaz u program L-Edit čime je omogućena i vizuelna verifikacija urađene ekstrakcije relevantnih geometrijskih primitiva.

TRFv6

Program TRFv6 obrađuje fajl koji je dobijen programom TRCIFv3 (io.cif). Program TRFv6 vrši instanciranje svih standardnih ćelija, kanala za povezivanje i kanala za smeštanje standardnih ćelija, što podrazumeva primenu zadatih transformacija nad definicijama simbola zadatih u ulaznom fajlu (io.cif). Kao rezultat dobija se novi fajl (ioe.cif) koji sadrži sve geometrijske primitive za svaki od tri relevantna nivoa maski LCMF, LCMS i LCVA. U programu TRFv6 implementiran je sledeći algoritam: najpre se formira tabela simbola koja sadrži ime simbola, opis parametara transformacije primenjene nad simbolom i pokazivač na listu sa spiskom svih simbola koji se pozivaju u definiciji posmatranog simbola sa parametrima transformacije koja se primenjuje nad datim simbolom. Prilikom instanciranja simbola kreće se od kraja tabele simbola. Za svaki ulaz u tabeli simbola proverava se da li definicija simbola sadrži poziv drugih simbola; ako ovaj uslov nije ispunjen (lista je prazna) prelazi se na sledeći simbol u tabeli simbola. Ako definicija simbola uključuje poziv drugih simbola vrši se ekspanzija dela definicije posmatranog simbola (deo bez poziva drugih simbola), što podrazumeva primenu transformacije nad datim simbolom i upis opisa svih geometrijskih simbola u izlazni fajl; zatim se prolazi kroz listu pozvanih elemenata, uzimaju se parametri transformacije i formira rezultujuća transformacija za ulaz u tabeli simbola koji odgovara pozvanom simbolu. Ako definicija pozvanog simbola ne sadrži pozive drugih simbola vrši se ekspanzija; u protivnom, prolazi se kroz njegovu listu pozvanih simbola i ažuriraju parametri transformacija.

EXTv4

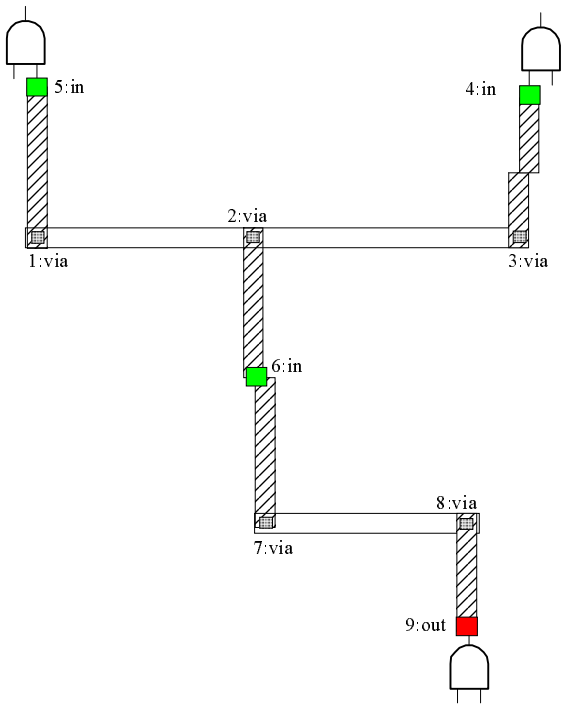
Izlazni fajl iz programa TRFv6 sadrži opise geometrijskih struktura u relevantnim nivoima LCMF, LCMS i LCVA. Opis geometrijskih primitiva u LCVA nivou dodatno je proširen informacijama koje omogućuju određivanje da li je posmatrani opis *via* element, ulazni ili izlazni terminal. Međutim, izlazni fajl ioe.cif sadrži opise geometrijskih primitiva koje nisu potrebne za proces određivanja veza i njihovih dužina. Zadatak programa EXTv4 je da obezbedi uklanjanje opisa redundantnih geometrijskih primitiva i da formira novi fajl (ioee.cif) koji sadrži opise posebno (a) *via* elementa, (b) ulaznih terminala, (c) izlaznih terminala, (d) *cmf* i (e) *cms* geometrijskih primitiva.

COREv26

Program COREv26 je glavni program MARS paketa koji određuje skupove povezanih geometrijskih primitiva, utvrđuje broj veza, određuje interkonekcije u okviru skupa povezanih geometrijskih primitiva i izračunava potrebnu statistiku.

Prvi korak predstavlja formiranje tabela sa informacijama o svakoj od geometrijskih primitiva, posebno za *cmf* i *cms* elemente, *via* elemente, ulazne i izlazne terminale. Nakon kreiranja tabela prelazi se na sledeći korak koji podrazumeva pronalaženje skupa povezanih geometrijskih primitiva. Ako se usvoji definicija da su dve geometrijske primitive u relaciji ako se seku ili dodiruju, pronalaženje skupa povezanih geometrijskih primitiva zapravo je određivanje refleksivno-tranzitivnog zatvaranja nad skupom geometrijskih primitiva. Zbog veličine problema za implementaciju algoritma koristi se lista. U listu se umeće jedna geometrijska primitiva; zatim se pronalaze sve geometrijske primitive koje se u direktnoj vezi sa tekućom primitivom; ukoliko se pronađene primitive već ne nalaze u listi, vrši se njihovo umetanje na kraj liste. Postupak se ponavlja sve dok se ne dođe do kraja liste.

Nakon formiranja skupa povezanih geometrijskih primitiva analiza se vrši u okviru svakog skupa povezanih geometrijskih primitiva posebno. Na osnovu informacija iz skupa povezanih geometrijskih primitiva formiraju se dve tabele: prva, tabela terminala koja sadrži informacije o terminalnim elementima (*via*, ulazni, izlazni) i druga, tabela *cms* i *cmf* geometrijskih primitiva koja sadrži informacije o neterminalnim elementima skupa. Za svaki teminalni element određuje se lista terminala sa kojima je taj terminal direktno u vezi. Dva terminala su u direktnoj vezi ako postoji jedna ili više neterminalnih primitiva koje ih povezuju, i ne postoji nijedan terminalni element između njih. Algoritam određivanja veza u jednom skupu povezanih geometrijskih primitiva prikazan je na slici 27.



Slika 27. Prikaz algoritma određivanja veza u skupu povezanih geometrijskih primitiva.

Opis: Na slici je prikazan jedan mogući skup povezanih geometrijskih primitiva sa označenim terminalnim elementima. Na osnovu definicije veze između terminala formira se tabela povezanosti terminala, koja za svaki terminal sadrži listu terminala povezanih sa posmatranim terminalom. Jedan skup može sadržati jednu ili više veza; broj različitih veza određen je brojem IN (ulaznih) terminala. U posmatranom primeru mogu se uočiti tri interkonekcije: (a) 9-8-7-6, (b) 9-8-7-6-2-3-4 i (c) 9-8-7-6-2-1-5.

Prvi korak u određivanju veza je da se odrede početni i završni terminal. Početni terminal je uvek tipa IN, a završni terminal je tipa OUT (izlazni) ili VIA u zavisnosti od toga da li skup povezanih terminala uključuje terminalni element tipa OUT ili ne, redom. U algoritmu se koristi struktura tipa steka. Početni element se stavlja na stek; zatim se iz tabele traže elementi sa kojima je dati terminal povezan i jedan od njih se stavlja na vrh steka, a ostali kandidati se stavljaju na rezervni stek. U svakom koraku algoritma proveravamo da li je na vrhu steka završni element; ukoliko jeste, na steku se nalazi veza opisana nizom terminala; u suprotnom, stavljamo novi element na stek pod uslovom da se taj element ne nalazi na steku. Ukoliko se desi da se na stek ne može staviti ni jedan novi element, prelazi se u režim skidanja sa steka, sve dok se ne dođe do elementa koji se nalazi na vrhu rezervnog steka. Bira se alternativni put i postupak se ponavlja sve dok se ne dođe do završnog elementa na vrhu steka.

Algoritam je demonstriran na konkretnom primeru. Određuje se veza koja povezuje ulazni terminal 4 (početni) i izlazni terminal 9 (završni). Na stek se stavlja broj terminala 4; terminal 4 je u vezi sa terminalom 3, pa se ovaj stavlja na vrh steka. Terminal 3 je u vezi sa terminalima 2 i 4, pa se na vrh steka stavlja terminal 2; terminal 4 je već na steku, pa u ovom koraku nije potrebno pamtiti alternativni put na rezervnom steku. Terminal 2 je povezan sa terminalima 1, 3 i 6; terminal 1 se stavlja na stek, a na rezervnom steku se pamti informacija da je od terminala 2 postojao i jedan alternativni put ka terminalu 6; terminal 3 je već na steku pa se ne stavlja na rezervni stek. Terminal 1 je povezan sa terminalima 2 i 5, pa se na vrh steka stavlja terminal 5 (jer je 2 već na steku). Terminal 5 je poslednji jer se nijedan terminal ne može staviti na stek. To znači da je izabrana pogrešna grana, pa se prelazi u režim skidanja sa steka. Redom se skidaju elementi sve dok se ne dođe do elementa koji se nalazi na vrhu rezervnog steka. U ovom slučaju skidaju se elementi 5 i 1. Element 2 je na vrhu rezervnog steka i sadrži alternativni put ka terminalu 6. Tako se na vrh steka stavlja element 6, zatim 7, 8 i 9. Terminal 9 je završni, tako da se na steku nalazi veza zadata terminalnim čvorovima 4-3-2-6-7-8-9. Na osnovu koordinata terminalnih elemenata može se izračunati dužina interkonekcije.

Program COREv26 omogućuje analizu veza koje povezuju terminalne elemente standardnih ćelija u jezgru čipa. Takođe, mogu se analizirati i veze koje povezuju ulazne (izlazne) terminale standardnih ćelija sa ulaznim (izlaznim) pedovima na okviru čipa. Za interkonekcije ovog tipa može se odrediti kolika je dužina dela intekonekcije koji pripada jezgru čipa, odnosno, kolika je dužina dela interkonekcije koji pripada oblasti za povezivanje jezgra sa okvirom čipa.

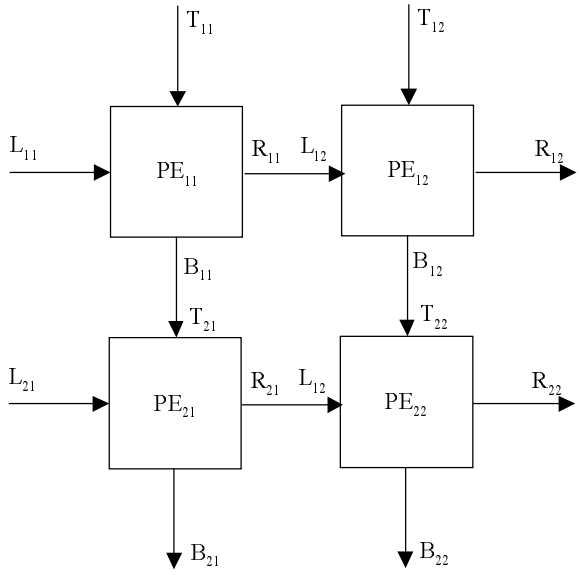
Programski paket MARS je razvijen na programskom jeziku C++ korišćenjem programskog paketa Borland C++ 4.5. Programski paket MARS radi pod operativnim sistemom Windows95. Detalji programa koji čine programski paket MARS dati su u Apendiksu #2.

7. Detalji predloženog rešenja

U ovom poglavlju prikazani su detalji postojećih 2D i predloženih 3D arhitektura posmatranih sistoličkih polja za svaki od ranije definisanih eksperimenata.

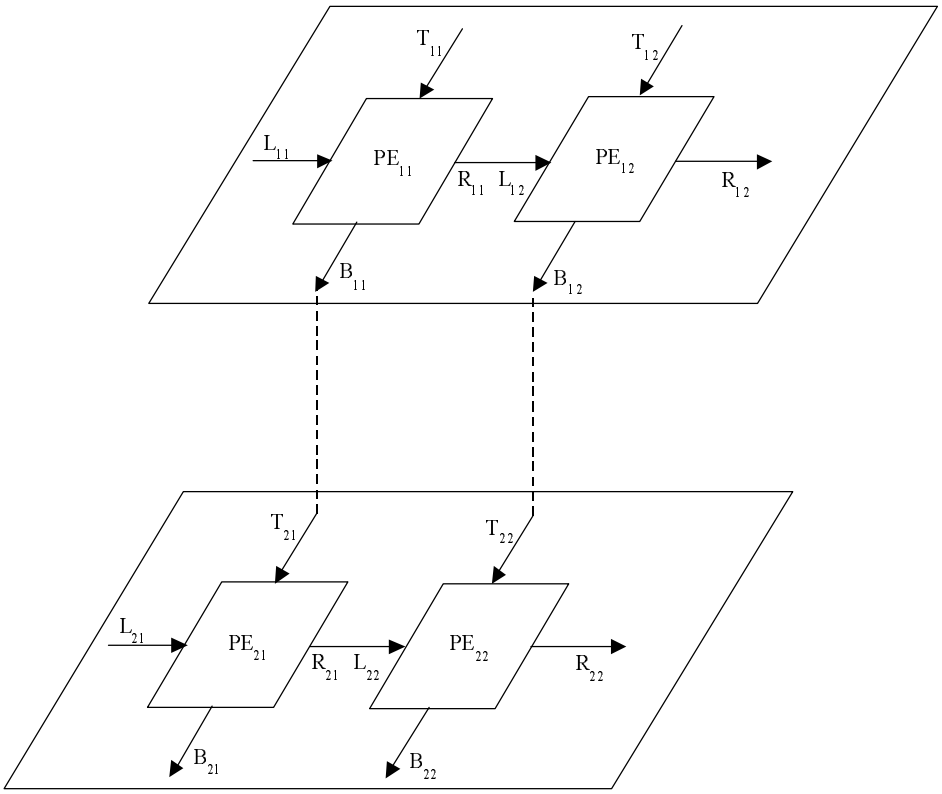
Eksperiment #1

Posmatra se sistoličko polje za množenje matrica sa četiri procesna elementa. Struktura posmatranog sistoličkog polja data je na slici 28, a blok šema jednog procesnog elementa data je na slici 16. Komunikacija između procesnih elemenata odvija se po linijama širine četiri bita; jedan procesni element sadrži množač 2x2, četvorobitni sabirač, tri četvorobitna registra i jedan četvorobitni multiplekser 2x1. Eksperiment #1 podrazumeva poređenje dužina veza za tri različite realizacije posmatranog sistoličkog polja: (a) 2D realizacija čija je blok šema prikazana na slici 28, (b) 3D realizacija u dva aktivna nivoa čija je blok šema prikazana na slici 29 i (c) 3D realizacija u četiri aktivna nivoa čija je blok šema prikazana na slici 30. Za sva tri eksperimenta realizacija ne uključuje pedove, tj. posmatra se samo jezgro čipa.



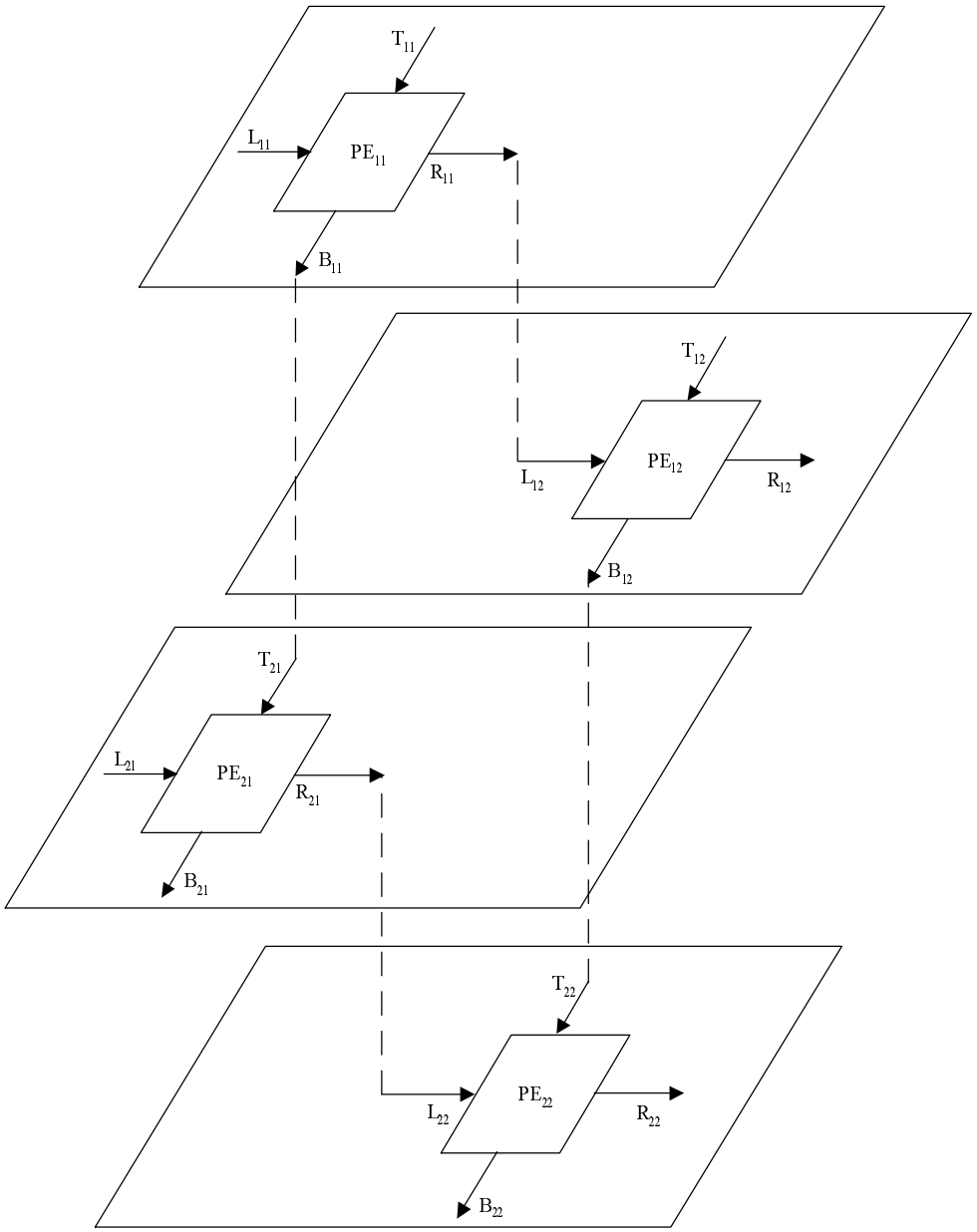
Slika 28. Struktura sistoličkog polja za množenje matrica sa četiri procesna elementa.

Opis: Na slici je prikazana blok šema sistoličkog polja sa četiri procesna elementa koji su realizovani u jednoj ravni. Sve ulazne, izlazne i linije koje povezuju susedne procesne elemente su širine četiri bita.



Slika 29. Predlog realizacije sistoličkog polja u dva aktivna nivoa.

Opis: Na slici je prikazana blok šema 3D realizacije sa dva aktivna nivoa posmatranog sistoličkog polja. Procesni elementi PE_{11} i PE_{12} nalaze se u prvoj aktivnoj ravni, a procesni elementi PE_{21} i PE_{22} nalaze se u drugoj aktivnoj ravni. Veze B_{11}-T_{21} i B_{12}-T_{22} povezuju elemente iz različitih ravni, pa se te veze sele u treću dimenziju, odnosno predstavljaju 3D veze. Raspoređivanje u okviru pojedinih ravni definisano je algoritmom za raspoređivanje koji koristi SPR (*Standard Cell Place and Route*) modul za automatsko raspoređivanje sa povezivanjem programa L-Edit.



Slika 30. Predlog realizacije sistoličkog polja u četiri aktivna ravni.

Opis: Na slici je prikazana blok šema 3D realizacije u četiri ravni posmatranog sistoličkog polja. Svaki od četiri procesna elementa nalazi se u posebnoj aktivnoj ravni. Sve veze između procesnih elemenata (R_{11}-L_{12}, B_{11}-T_{21}, B_{12}-T_{22}, R_{21}-L_{22}) sele se u treću dimenziju.

Eksperiment #2

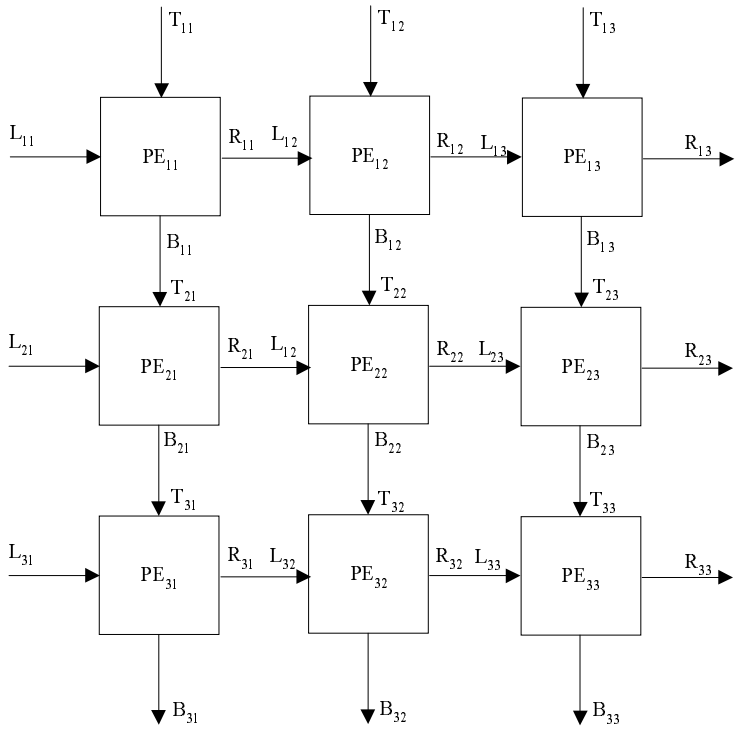
U eksperimentu #2 posmatra se identično sistoličko polje za množenje matrica sa četiri procesna elementa kao u eksperimentu #1. Posmatraju se dve realizacije: (a) 2D realizacija i (b) 3D realizacija sa dva aktivna nivoa; pri tome, za razliku od eksperimenta #1, realizacije uključuju korišćenje ulazno/izlaznih pedova. Ulazni pedovi su pridruženi ulaznim linijama L_{11}, L_{21}, T_{11}, T_{12}, a izlazni pedovi su pridruženi izlaznim linijama R_{12}, R_{22}, B_{21}, B_{22}.

Eksperiment #3

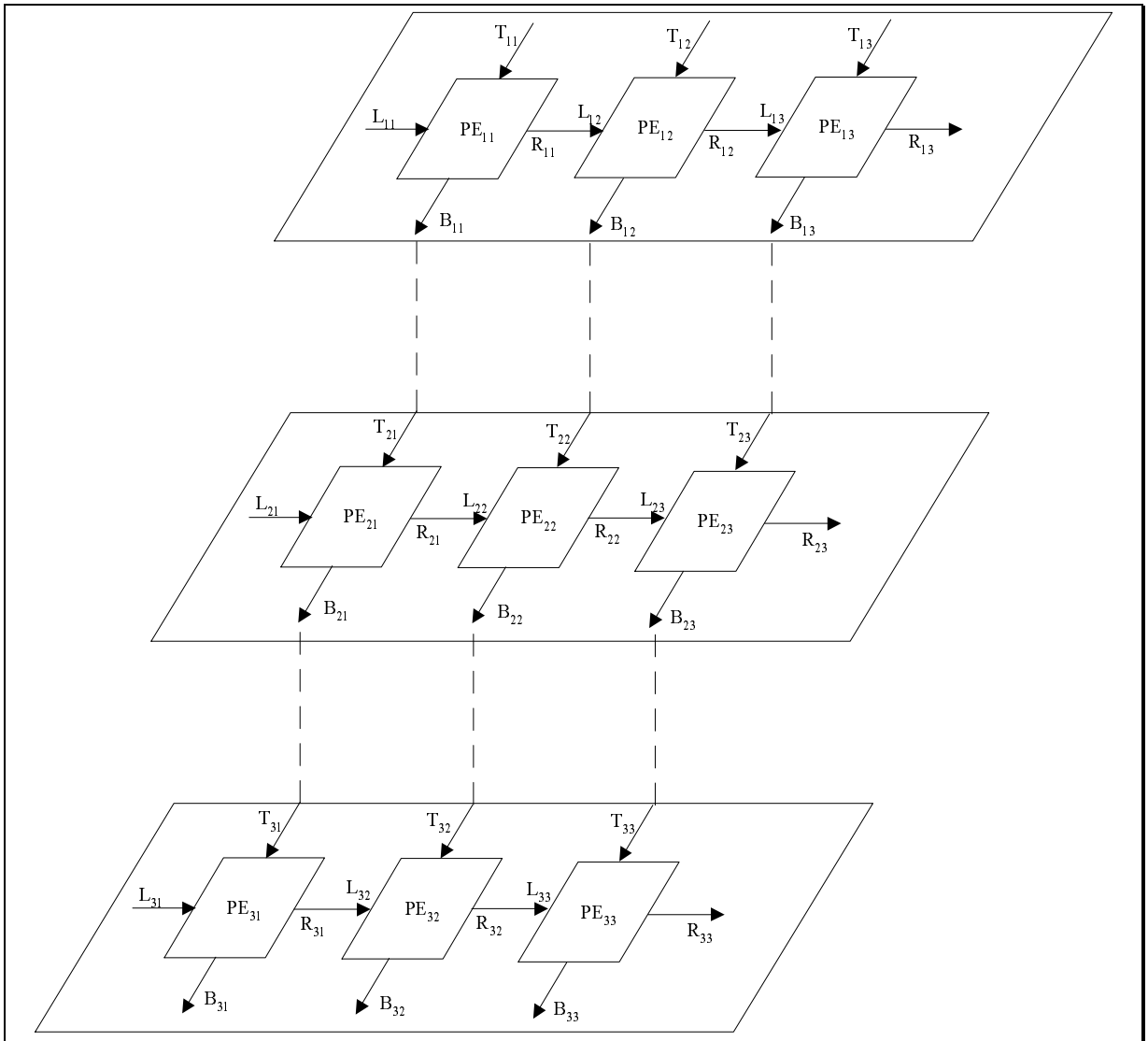
Posmatra se sistoličko polje za množenje matrica sa četiri procesna elementa sa istom strukturom kao u eksperimentu #1, s tim što se komunikacija između procesnih elemenata odvija preko 8-bitnih linija. Procesni element sadrži jedan 4x4 množač, 8-bitni sabirač, tri 8-bitna registra i jedan 8-bitni multiplekser 2x1. Kao i u eksperimentu #1 posmatraju se tri realizacije: (a) 2D realizacija, (b) 3D realizacija u dve aktivne ravni i (c) 3D realizacija u četiri aktivne ravni.

Eksperiment #4

Posmatra se sistoličko polje za množenje matrica sa devet procesnih elemenata. Struktura posmatranog sistoličkog polja data je na slici 31, a blok šema jednog procesnog elementa data je na slici 16. Eksperiment #4 podrazumeva poređenje dužina veza za dve različite realizacije posmatranog sistoličkog polja: (a) 2D realizacija čija je blok šema prikazana na slici 31, (b) 3D realizacija u tri aktivne ravni čija je blok šema prikazana na slici 32.



Slika 31. Struktura sistoličkog polja sa devet procesnih elemenata.

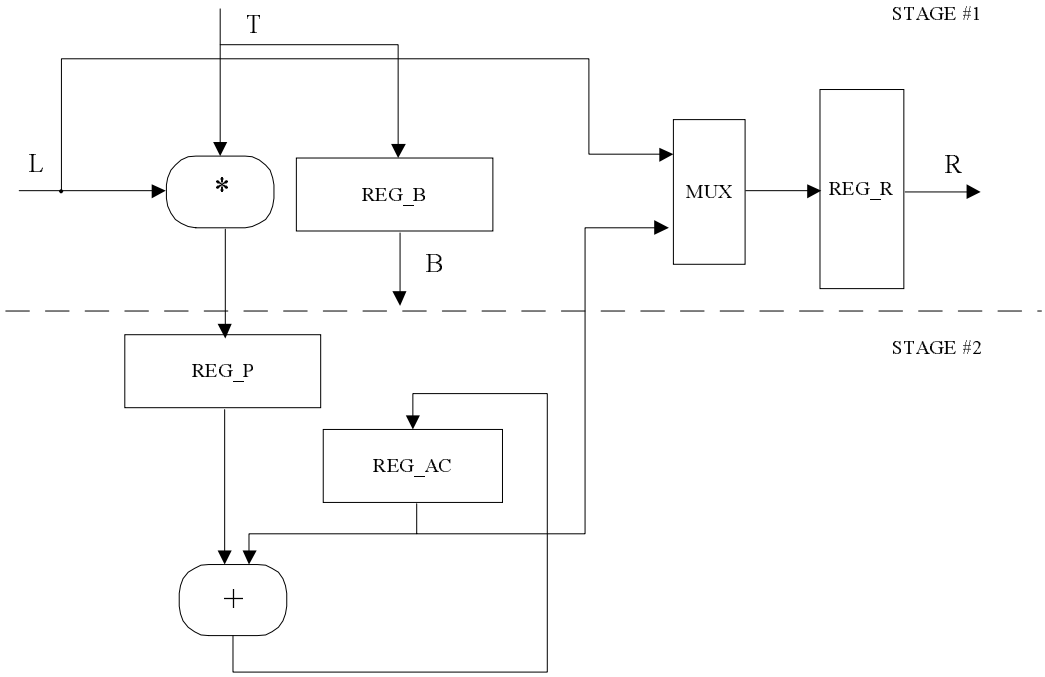


Slika 32. Predlog realizacije sistoličkog polja sa devet elemenata u tri aktivne ravni.

Opis: Na slici je pokazana blok šema 3D realizacije u tri ravni sistoličkog polja sa devet procesnih elemenata. Procesni elementi PE_{11}, PE_{12} i PE13 nalaze su u prvoj ravni, procesni elementi PE_{21}, PE_{220} i PE_{23} nalaze se u drugoj ravni, a procesni elementi PE_{31}, PE_{32}, PE_{33} nalaze se u trećoj ravni.

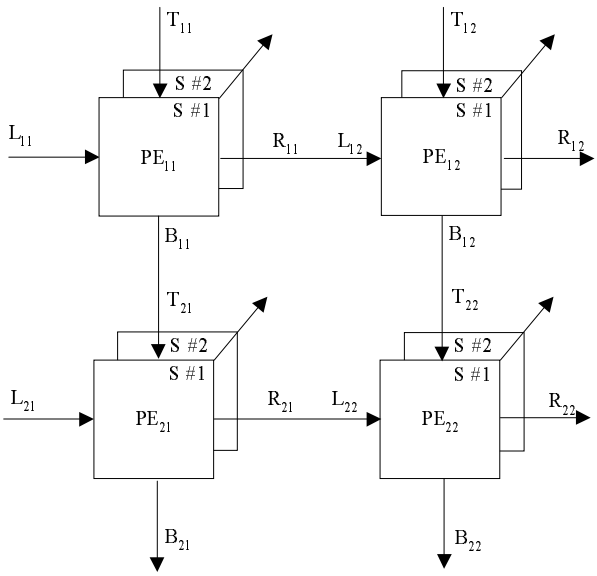
Eksperiment #5

U prethodnim eksperimentima razmatrane su klasične 2D realizacije i nekoliko 3D realizacija. Razmeštanje po ravnima u slučaju 3D realizacije vršeno je tako što su različiti procesni elementi ili grupe procesnih elemenata razmeštane po različitim ravnima. Sledeći kvalitativno novi korak bi bio da se sami procesni elementi realizuju u više aktivnih nivoa. U eksperimentu #5 posmatra se 3D realizacija sistoličkog polja za množenje matrica sa četiri procesna elementa, pri čemu se sami procesni elementi realizuju u dva aktivna nivoa. Posmatra se procesni element koji svoju funkciju obavlja u dva stepena protočne obrade. Blok šema procesnog elementa prikazana je na slici 33. Eksperiment #5 podrazumeva poređenje dve realizacije: (a) 2D realizacije i (b) 3D realizacije u dve aktivne ravni ravni čija je blok šema prikazana na slici 34.



Slika 33. Blok šema procesnog elementa koji koristi protočnu obradu.

Opis: Prikazana je struktura jednog procesnog elementa koji svoju funkciju obavlja u dva stepena protočne obrade. U prvom stepenu obavlja se množenje podataka koji dolaze po linijama L i T. U drugom stepenu dobijeni rezultat množenja iz prethodnog ciklusa sabira se sa sadržajem registra REG_A. Ciklus takta u ovom slučaju je određen sa Tclk=max(Tmul, Tadd); Tmul je kašnjenje kroz množač, a Tadd je kašnjenje kroz sabirač.



Slika 34. Realizacija sistoličkog kod koga su procesni elementi realizovani u dve ravni.

Opis: Procesni elementi ostvaruju svoju funkciju u dva stepena protočne obrade. Prvi stepen protočne obrade realizovan je u prvom aktivnom nivou, a drugi stepen protočne obrade u drugom aktivnom nivou.

8. Napomena o analitičkom modelovanju

Analitičko modelovanje podrazumeva formiranje matematičkog modela problema koji se analizira. U opštem slučaju, matematički model se može koristiti za različite vrste analiza. Na primer, matematički model se može koristiti sa ciljem da se da formalan dokaz ispravnosti glavne ideje istraživanja. Takođe, pomoću matematičkog modela može se dati gruba procena performanse i kompleksnosti. Najzad, primenom matematičkog modela mogu se dobiti rezultati kao dopuna simulacionoj analizi u uslovima kada zbog kompleksnosti i dimenzije problema simulaciona analiza nije moguća i/ili je vrlo zahtevna u pogledu procesorske snage i potrebnog vremena.

U ovom istraživanju zadatak matematičkog modela bi bio da da grubu procenu skraćenja dužina veza prilikom prelaska sa 2D na 3D VLSI. Do sada su razvijani matematički modeli bazirani na teoriji grafova koji su imali za cilj da daju procenu kompleksnosti i maksimalnu dužinu veza u 3D strukturi. Jedan primer takvog matematičkog modela pokazan je u radu [Aboe87].

Gore pomenuti matematički modeli imaju opštiji karakter i rezultat dobijen takvom analizom ima uglavnom samo teorijski značaj. Stoga, imajući u vidu uslove i pretpostavke pod kojima se vrši ovo istraživanje, trebalo bi razviti novi matematički model baziran na definisanim uslovima i pretpostavkama ovog istraživanja. Takav model bi bio baziran na kvalitativnoj analizi koja je pokazana u radu [Tong95] i uključivao bi elemente predložene metodologije 3D rutiranja, što je vrlo kompleksan zadatak.

Sa druge strane, korišćenjem postojećih alata za 2D VLSI projektovanje čipova mogu se dobiti tačni rezultati u pogledu dužine veza koju koristimo kao meru performanse, odnosno površine čipa koju koristimo kao meru kompleksnosti.

U uslovima kada su kompleksnost i potrebno vreme za razvoj matematičkog modela vrlo veliki i kada očekivani rezultati na bazi matematičkog modela nisu od velike važnosti u sprovedenom istraživanju, razvoj matematičkog modela nije od interesa, pa ovo istraživanje ne uključuje analitičko modelovanje.

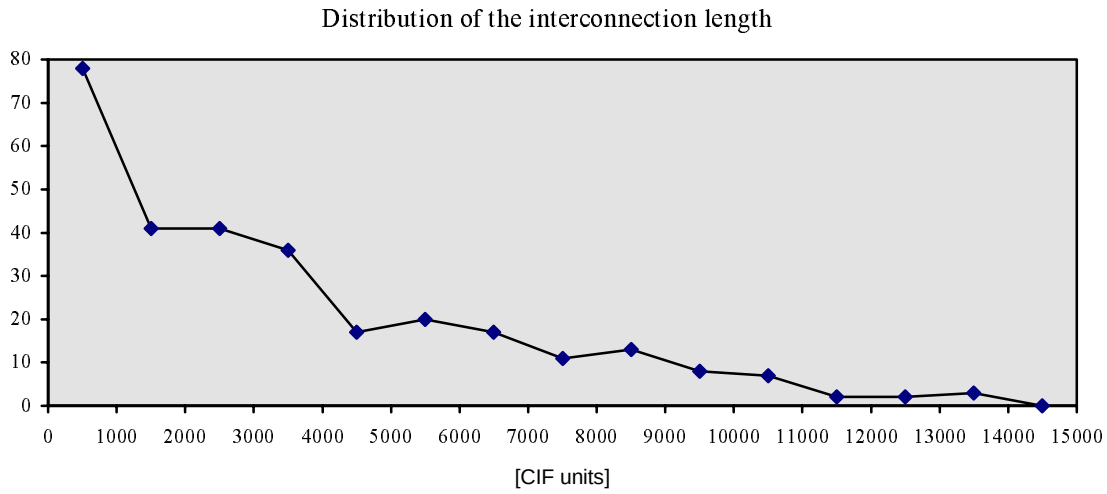
9. Simulaciona analiza

U ovom poglavlju prikazani su rezultati eksperimenata. Za svaku od realizacija u okviru eksperimenata prikazana je raspodela dužina veza, srednja dužina veza i površina jezgra čipa ili celog čipa, u zavisnosti od toga da li se posmatra realizacija bez I/O pedova ili sa I/O pedovima, redom. Takođe, za svaki eksperiment dat je grafik sa uporednim prikazom raspodela dužina veza, srednje dužine veza i površine čipa. Dužine veza su date u CIF jedinicama (CIF *units*); vrednost CIF jedinice zavisi od tehnologije. U izvedenim eksperimentima korišćena je SCN 2.0μm tehnologija, pa je jedna CIF jedinica jednaka 0.5μm. Površine čipova date su u milimetrima kvadratnim (mm^2).

Raspodela dužina veza prikazana je na sledeći način: posmatra se broj veza čije su dužine u opsegu koji je dat na apscisi. Dužine veza su date u CIF jedinicama. U posmatranim eksperimentima usvojeno je da je širina opsega 1000 CIF jedinica.

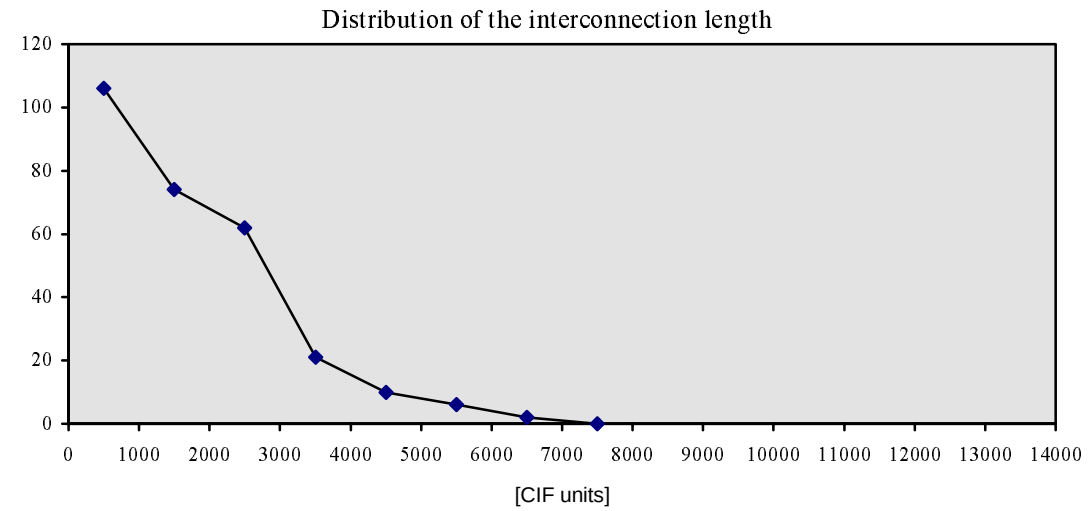
Eksperiment #1

U ovom eksperimentu posmatrane su tri realizacije sistoličkog polja za množenje matrica sa četiri četvorobitna procesna elementa: (a) 2D realizacija čija je blok šema prikazana na slici 28, (b) 3D realizacija u N=2 ravni čija je blok šema prikazana na slici 29 i (c) 3D realizacija u N=4 ravni čija je blok šema prikazana na slici 30. Posmatraju se samo veze unutar jezgra čipa (realizacije ne uključuju pedove), tj. analiziraju se samo veze između izlaznih i ulaznih terminala standardnih ćelija. Raspodele dužina veza za svaku od realizacija prikazane su na slikama 35, 36, i 37, redom. Uporedni prikaz za sve tri realizacije prikazan je na slici 38.



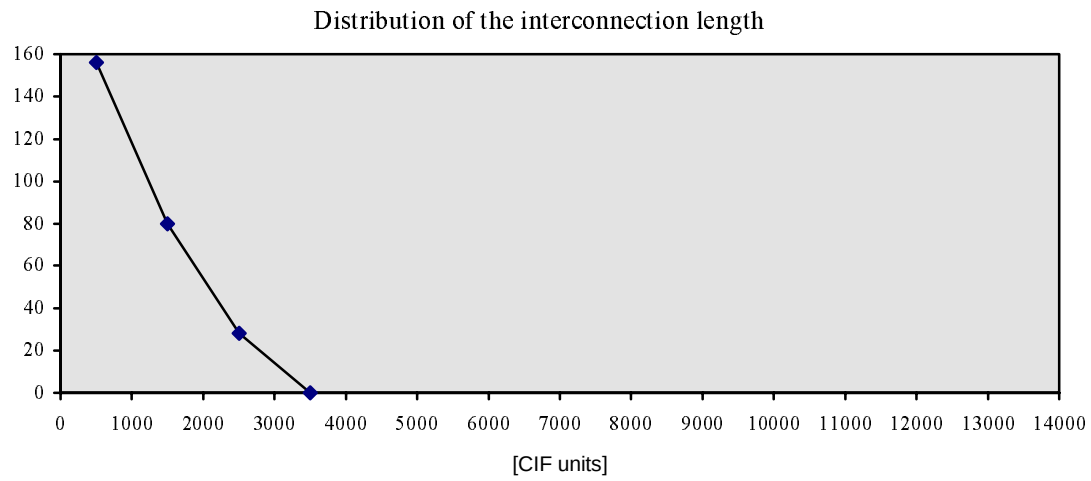
Slika 35. Eksperiment #1: raspodela dužina veza u slučaju 2D realizacije.

Opis: U jezgru čipa uočeno je ukupno 296 veza; prosečna dužina je 3490 CIF jedinica (1745μm). Dimenzije jezgra čipa su 1816μm x 2668μm, odnosno, površina jezgra čipa je 4.845mm^2.



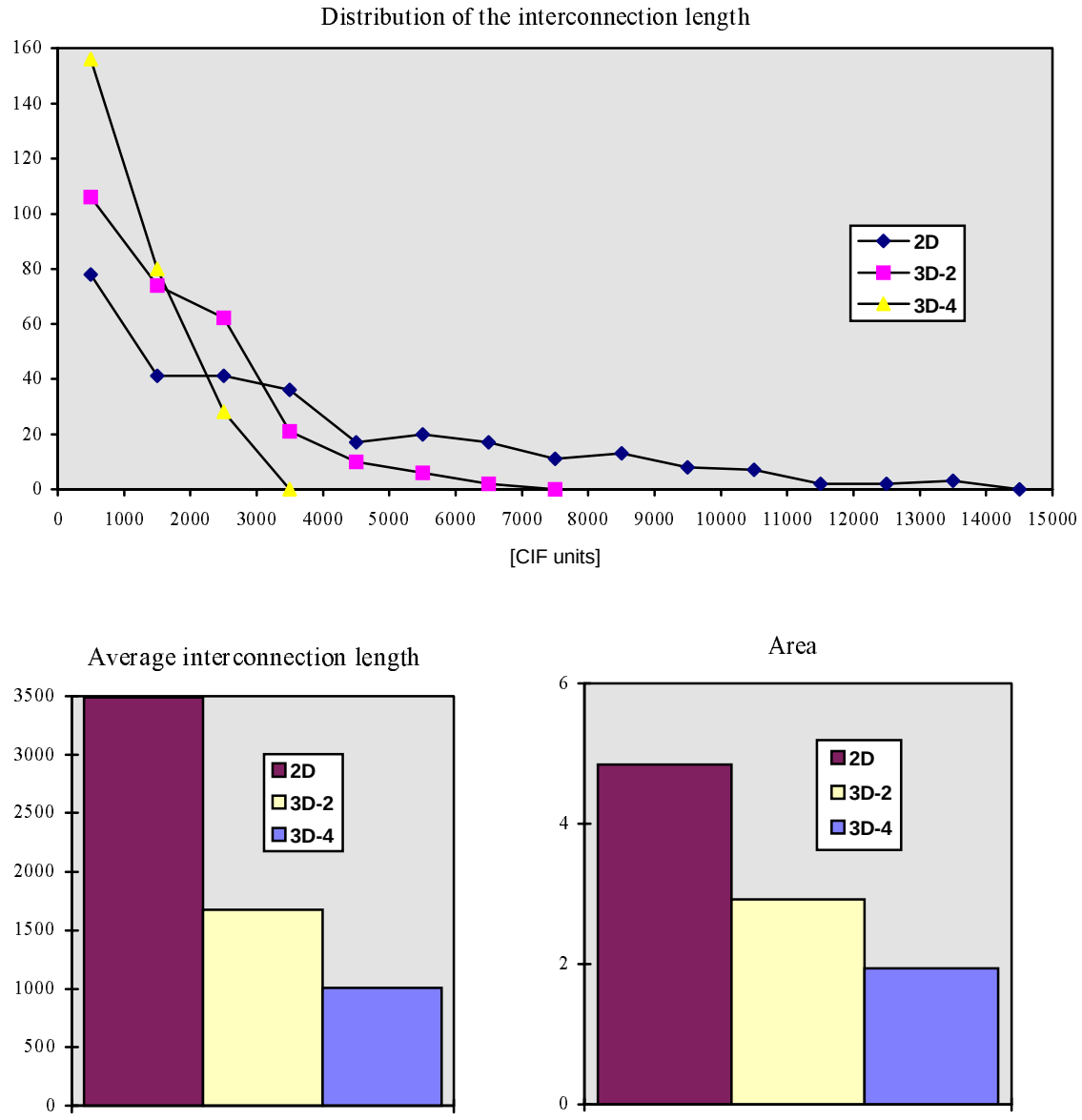
Slika 36. Eksperiment #1: raspodela dužina veza za slučaj 3D realizacije sa dve aktivne ravni.

Opis: U svakom od dva aktivna nivoa uočeno je po 140 veza; na osnovu toga sledi da se 16 veza seli u treću dimenziju. Prosečna dužina 2D veza (veze koje su u jednom aktivnom nivou) iznosi 1676 CIF jedinica. Pod pretpostavkom da je prosečna dužina 3D veza (veze koje se prostiru u više aktivnih nivoa) jednaka prosečnoj dužini 2D veza, dobija se ukupna prosečna dužina veza od 1676 CIF jedinica (837.5μm). Ukoliko bi se dužina 3D veza zanemarila, ukupna prosečna dužina veza bi bila 1585 CIF jedinica. Dimenzije svake od dve aktivne ravni je 1002μm x 1448μm, pa je površina svake od aktivnih ravni 1.46mm^2, odnosno ukupna površina je 2.92mm^2.



Slika 37. Eksperiment #1: raspodela dužina veza u slučaju 3D realizacije sa četiri ravni.

Opis: U svakom od četiri aktivna nivoa uočeno je po 66 2D veza; na osnovu toga sledi da u ovoj realizaciji postoji 32 3D veze. Pod pretpostavkom da je prosečna dužina 3D veza jednaka prosečnoj dužini 2D veza dobija se da je ukupna prosečna dužina veza 1007 CIF jedinica (503.4μm). Zanemarivanjem dužina 3D veza dobija se da je ukupna prosečna dužina veza 898 CIF jedinica. Dimenzije svake od četiri aktivne ravni su 677μm x 716μm; prema tome, površina svake od aktivnih ravni je 0.485mm^2, odnosno ukupna površina je 1.94mm^2.

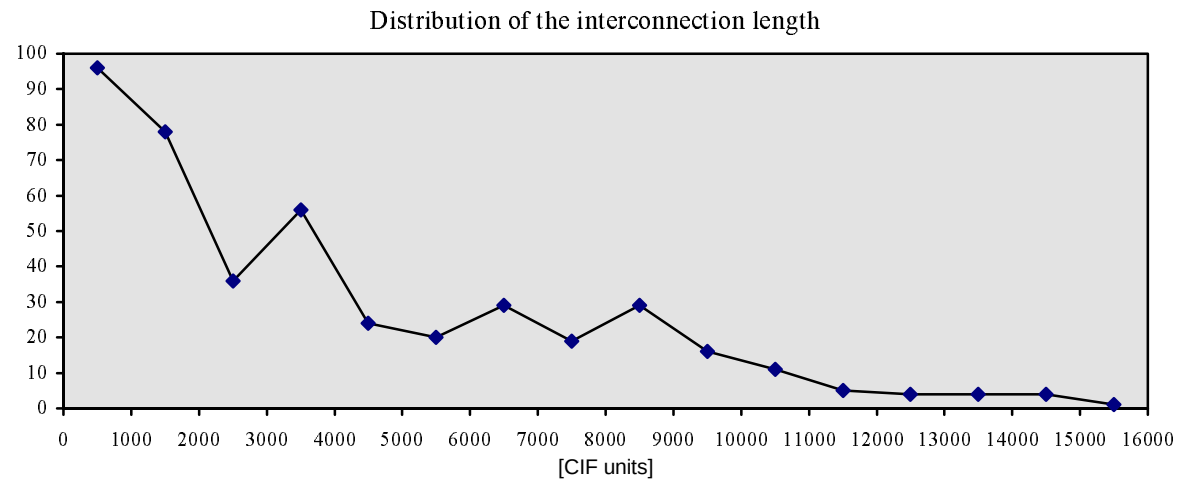


Slika 38. Eksperiment #1: Uporedni prikaz raspodele dužina veza, srednje dužine veza i površine jezgra čipova.

Opis: Srednja dužina veza u 3D realizaciji sa dve ravni je 48% srednje dužine veza u 2D realizaciji; srednja dužina veza u 3D realizaciji sa četiri aktivne ravni je 29% srednje dužine veza u 2D realizaciji. Ukupna površina u slučaju 3D realizacije u dva aktivna nivoa (zbir površina aktivnih ravni) je 60% površine u slučaju 2D realizacije; ukupna površina u slučaju 3D realizacije u četiri aktivna nivoa je 40% površine u okviru 2D realizacije.

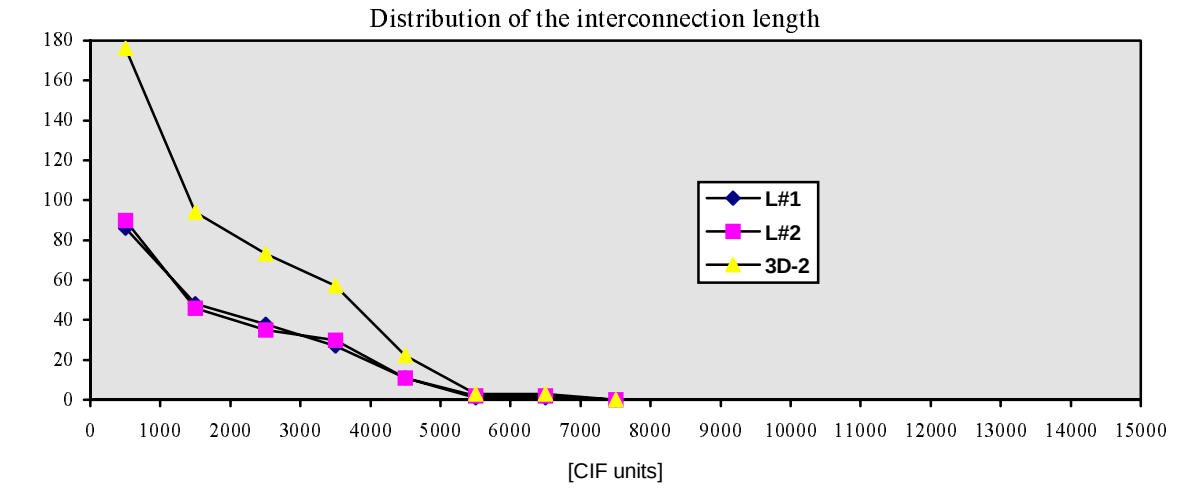
Eksperiment #2

Eksperiment #2 se bazira na eksperimentu #1, s tim što realizacija uključuje korišćenje pedova. Posmatrane su dve realizacije: (a) 2D realizacija i (b) 3D realizacija sa dve aktivne ravni. Raspodele dužina veza za dve posmatrane realizacije prikazane su na slikama 39 i 40, redom. Uporedni prikaz dat je na slici 41.



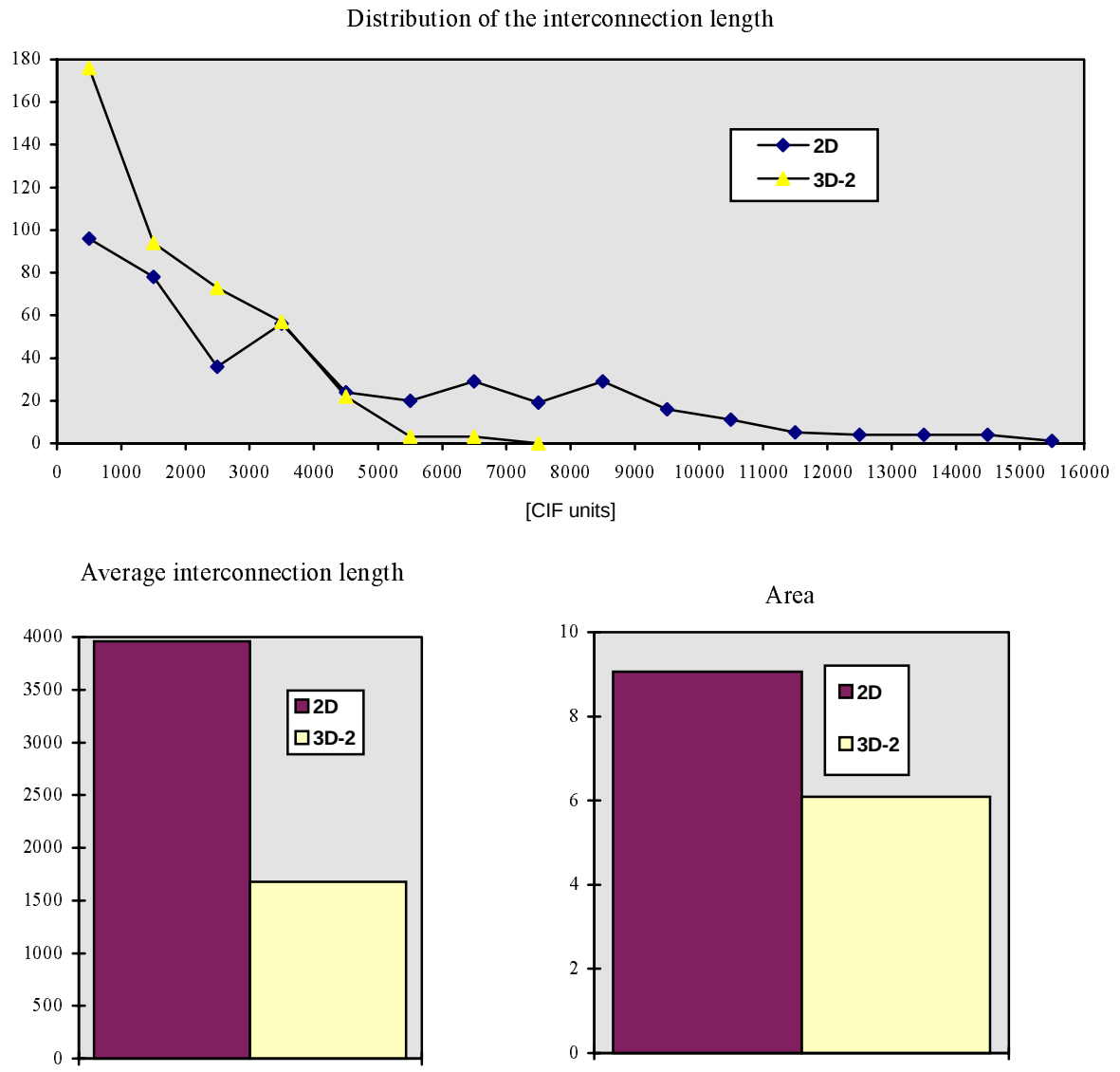
Slika 39. Eksperiment #2: raspodela dužina veza u slučaju 2D realizacije.

Opis: U slučaju realizacije sa ulazno/izlaznim pedovima mogu se uočiti tri vrste veza: (a) prvu grupu čine veze koje povezuju ulazni ped sa ulaznim terminalima standardnih ćelija; (b) drugu grupu čine veze koje povezuju izlazne terminale sa ulaznim terminalima i (c) treću grupu čine veze koje povezuju izlazne terminale standardnih ćelija sa izlaznim pedovima. Srednja dužina veza je 3957 CIF jedinica. Dimenzije čipa su 2443µm x 3711µm, odnosno površina čipa je 9.066mm^2.



Slika 40. Eksperiment #2: raspodela dužina veza u slučaju 3D realizacije sa dve aktivne ravni.

Opis: Ravan #1 sadrži elemente PE_{11} i PE_{12} sa pripadajućim ulaznim pedovima (tri grupe po četiri peda), izlaznim pedovima (jedna grupa od četiri peda) i ulaznim pedom za kontrolni signal multipleksera. Ravan #2 sadrži elemente PE_{21} i PE_{22} sa pripadajućim ulaznim pedovima (jedna grupa od četiri peda), izlaznim pedovima (tri grupe po četiri peda) i ulaznim pedom za signal takta. Srednja dužina veza u ravni #1 je 1670 CIF jedinica; dimenzije ravni #1 su 1517µm x 2039µm, odnosno površina ravni #1 je 3.093mm^2. Srednja dužina veza u ravni #2 je 1683 jedinica; dimenzije ravni #2 su 1496µm x 2003µm, odnosno površina ravni #2 je 2.996mm^2. Ukupna površina čipa u slučaju 3D realizacije izračunava se po formuli $N \cdot \max_i(A_i)$; A_i predstavlja površinu i-te aktivne ravni. Na osnovu toga ukupna površina posmatranog 3D čipa je 6.186 mm^2.

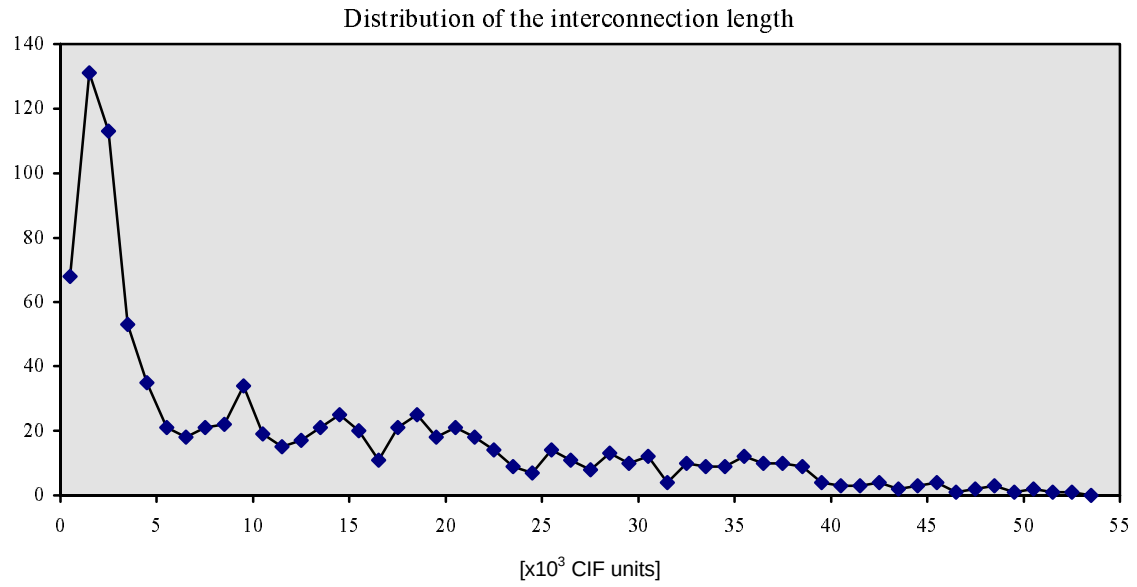


Slika 41. Eksperiment #2: uporedni prikaz raspodele dužina veza, srednje dužine veza i ukupne površine čipa.

Opis: Srednja dužina veza u slučaju 3D realizacije sa dve aktivne ravni je 42% srednje dužine veza u slučaju 2D realizacije. Površina čipa u slučaju 3D realizacije je 68% površine u slučaju 2D realizacije.

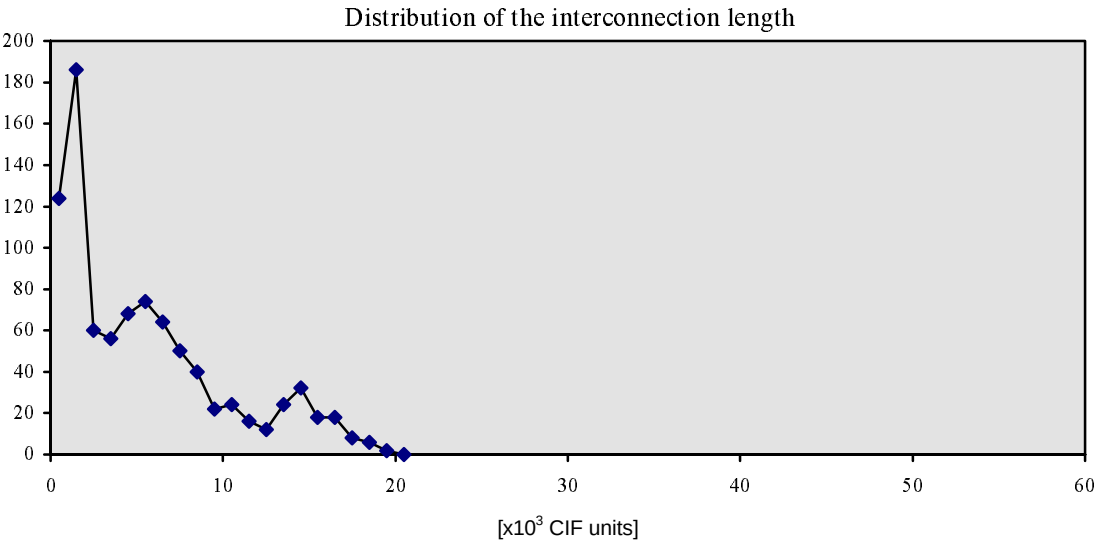
Eksperiment #3

Eksperiment #3 se bazira na eksperimentu #1, s tim što se posmatra sistoličko polje kod kojih je dužina reči osam bita. Posmatrane su tri realizacije: (a) 2D realizacija čija je blok šema pokazana na slici 28, (b) 3D realizacija sa dve aktivne ravni čija je šema prikazana na slici 29 i (c) 3D realizacija sa četiri aktivne ravni čija je šema prikazana na slici 30. Raspodele dužina veza za posmatrane realizacije prikazane su na slikama 42, 43 i 44, redom. Uporedni prikaz dat je na slici 45.



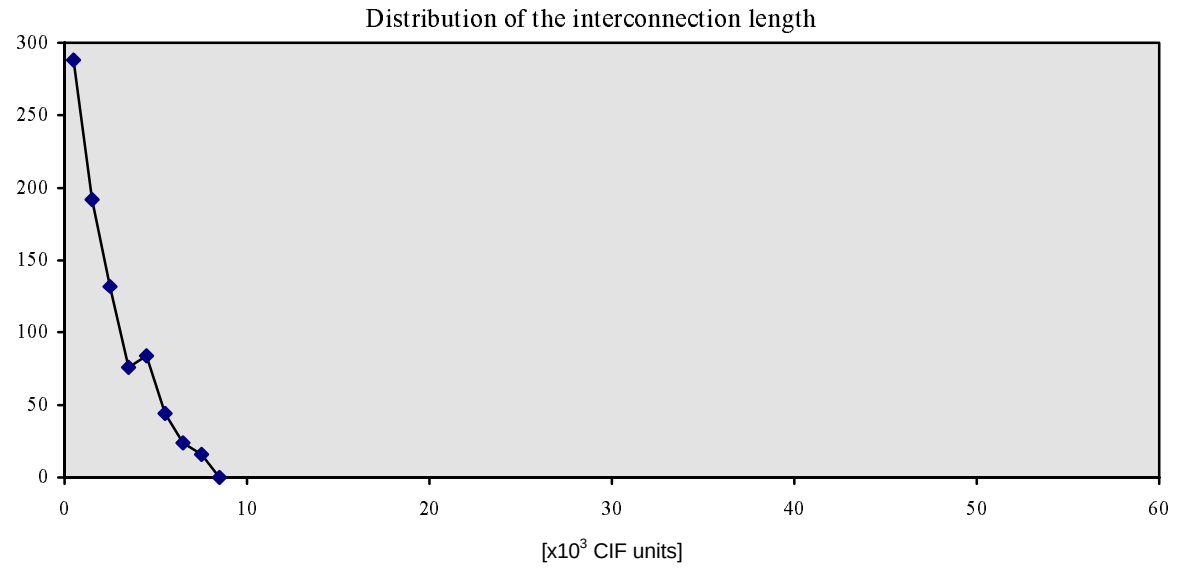
Slika 42. Eksperiment #3: raspodela dužina veza u slučaju 2D realizacije.

Opis: Uočeno je 952 veze (599 skupova povezanih geometrijskih primitiva); srednja dužina veza je 12537 CIF jedinica. Dimenzije jezgra čipa su 4775µm x 11128µm, odnosno površina jezgra čipa je 53.136mm^2.



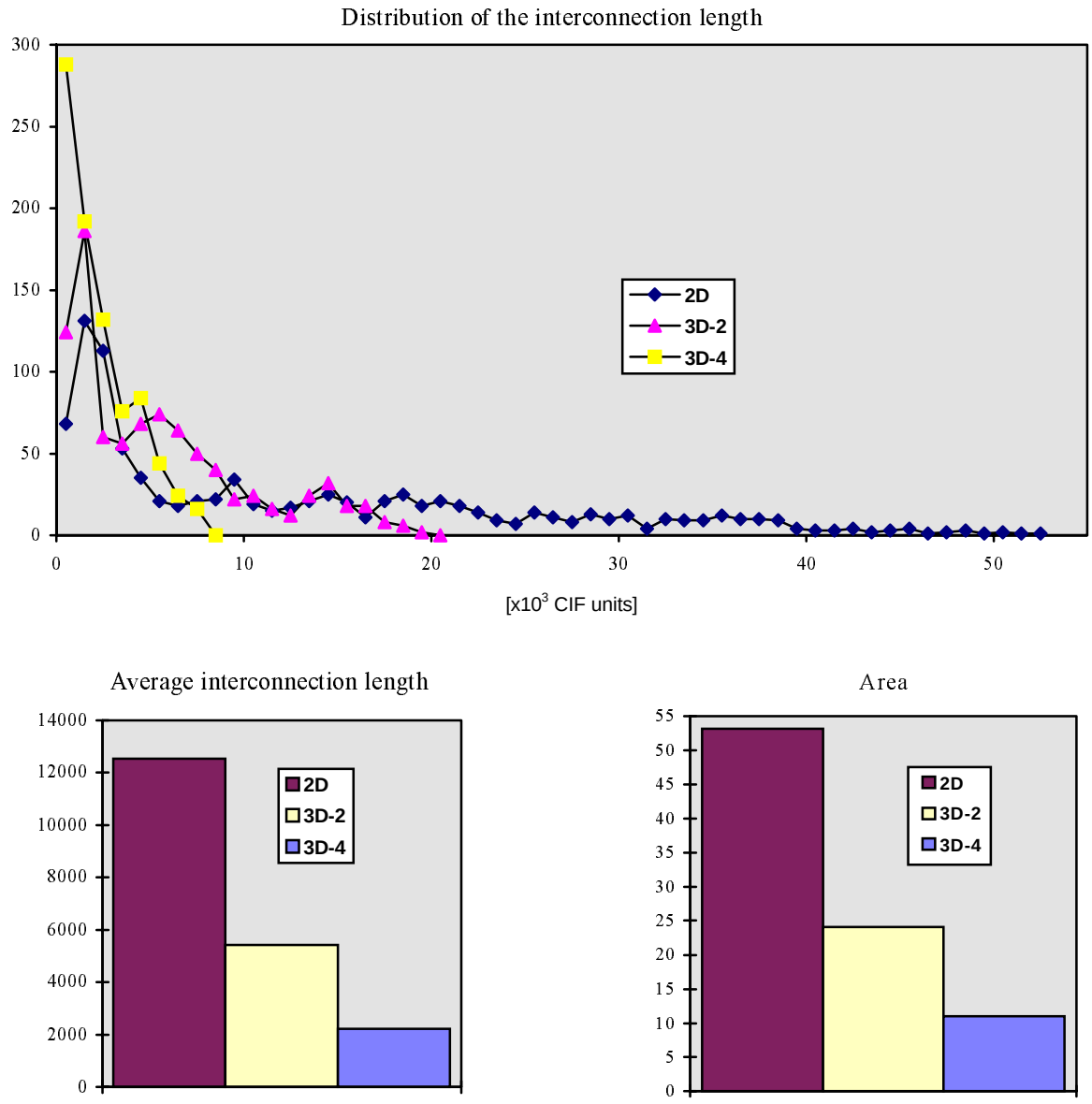
Slika 43. Eksperiment #3: raspodela dužina veza u slučaju 3D realizacije sa dve aktivne ravni.

Opis: Uočeno je 908 veza (604 skupa povezanih geometrijskih primitiva), a srednja dužina veza je 5429 jedinica. Dimenzije jezgra čipa su 2619µm x 4600µm, odnosno ukupna površina jezgra čipa je 24.09 mm^2.



Slika 44. Raspodela dužina veza u slučaju 3D realizacije sa četiri aktivne ravni.

Opis: Uočeno je 856 veza (600 skupova povezanih geometrijskih primitiva), a prosečna dužina jedne interkonekcije je 2229 CIF jedinica. Dimenzije jedne aktivne ravni su 1433μm x 1918μm, pa je ukupna površina 3D čipa sa četiri aktivne ravni 10.99 mm^2.

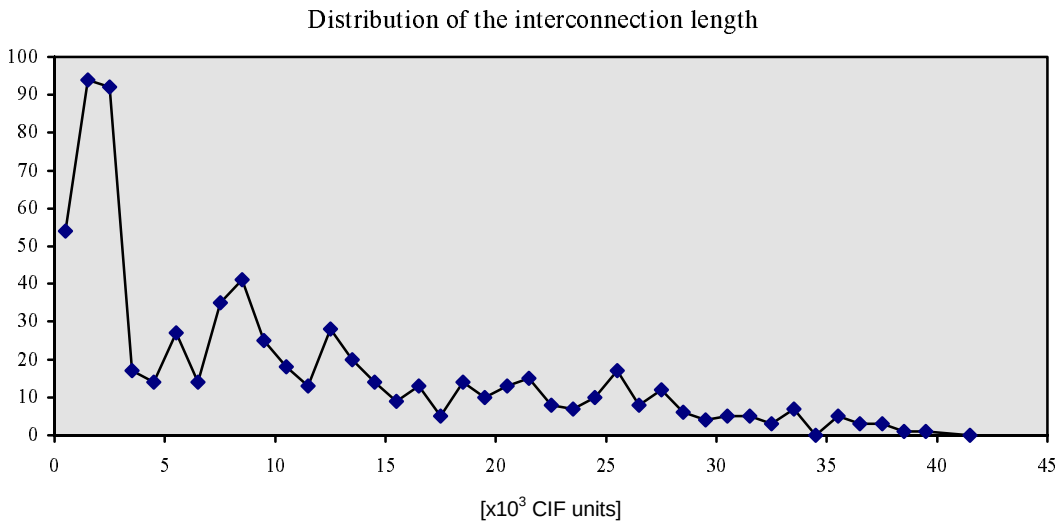


Slika 45. Eksperiment #3: uporedni prikaz raspodela dužina veza, srednje dužine veza i površine čipova.

Opis: Srednja dužina veza u slučaju 3D realizacije u dve aktivne ravni je 43% srednje dužine u slučaju 2D realizacije; srednja dužina veza u slučaju 3D realizacije sa četiri aktivne ravni je 18% srednje dužine veza u slučaju 2D realizacije. Ukupna površina jezgra čipa u slučaju 3D realizacije u dve ravni je 45% površine jezgra čipa u slučaju 2D realizacije; površina jezgra čipa u slučaju 3D realizacije u četiri ravni je 20% površine jezgra čipa u slučaju 2D realizacije.

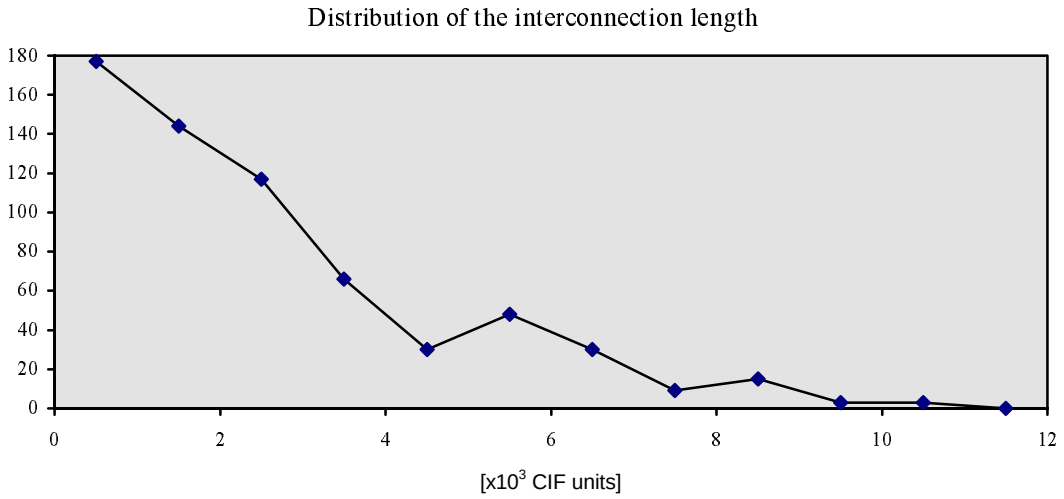
Eksperiment #4

U eksperimentu #4 posmatrane su dve realizacije sistoličkog polja sa devet procesnih elemenata: (a) 2D realizacija čija je šema prikazana na slici 31 i (b) 3D realizacija sa tri aktivne ravni čija je šema prikazana na slici 32. Dužina reči je četiri bita. Posmatraju se samo veze unutar jezgra čipa. Raspodele dužina veza za dve posmatrane realizacije prikazane su na slikama 46 i 47, redom. Uporedni prikaz dat je na slici 48.



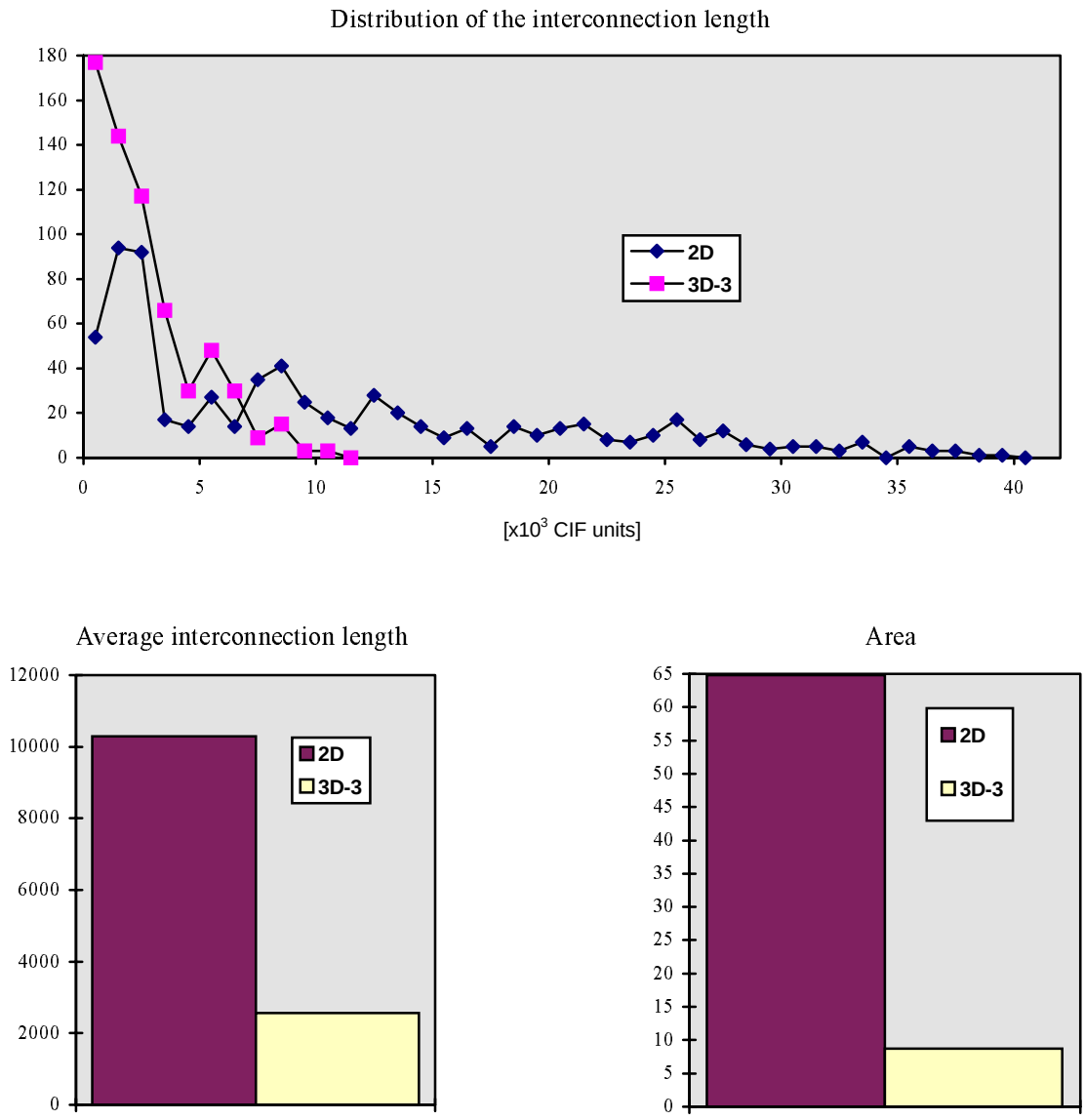
Slika 46. Eksperiment #4: raspodela dužina veza u slučaju 2D realizacije.

Opis: U ovoj realizaciji uočeno je 690 veza; prosečna dužina veza je 10298 CIF jedinica. Dimenzije jezgra čipa su 7135μm x 9078μm, odnosno površina jezgra čipa je 64.77 mm^2.



Slika 47. Eksperiment #4: raspodela dužina veza u slučaju 3D realizacije sa tri aktivne ravni.

Opis: U svakoj od ravni postoji po 214 veza; ukupno 48 veza se seli u treću dimenziju; ukoliko se pretpostavi da je srednja dužina 3D veza jednaka srednjoj dužini veza u 2D ravnima, ukupna srednja dužina veza je 2563 CIF jedinice. Dimenzije jezgra čipa u svakoj od ravni su 1376μm x 2110μm, odnosno ukupna površina 3D čipa sa tri aktivne ravni je 8.71 mm^2.

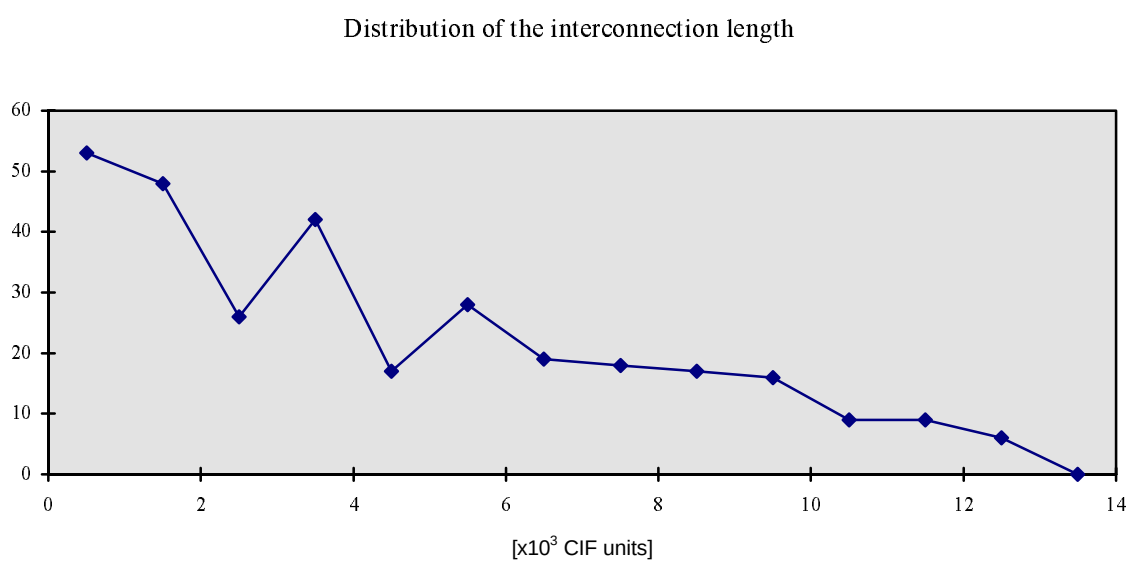


Slika 48. Eksperiment #4: uporedni prikaz raspodele dužina veza, srednje dužine veza i površine čipova.

Opis: Srednja dužina veza u slučaju 3D realizacije u tri aktivne ravni je 25% srednje dužine u slučaju 2D realizacije. Ukupna površina jezgra čipa u slučaju 3D realizacije u tri ravni je 14% površine jezgra čipa u slučaju 2D realizacije.

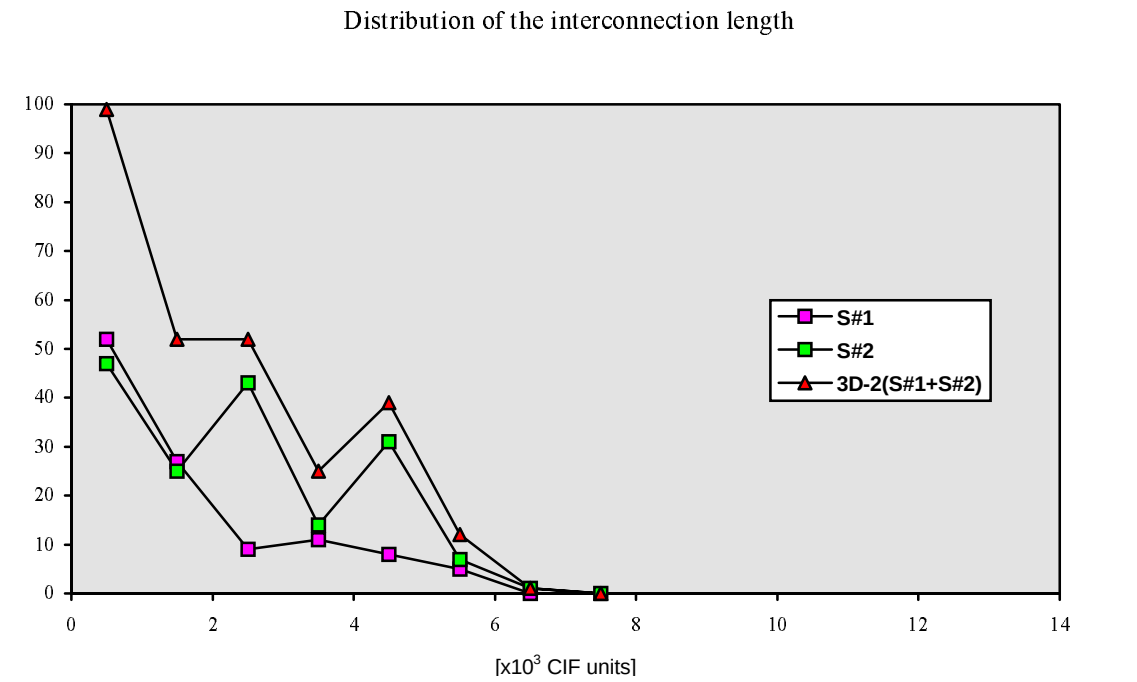
Eksperiment #5

U okviru eksperimenta #5 razmatra se rešenje u kome se procesni elementi realizuju u više aktivnih ravni. Posmatrane su dve realizacije: (a) 2D realizacija, (b) 3D realizacija u dve aktivne ravni čija je šema prikazana na slici 34. Raspodele dužina veza za posmatrane realizacije prikazane su na slikama 49 i 50, redom. Uporedni prikaz dat je na slici 51.



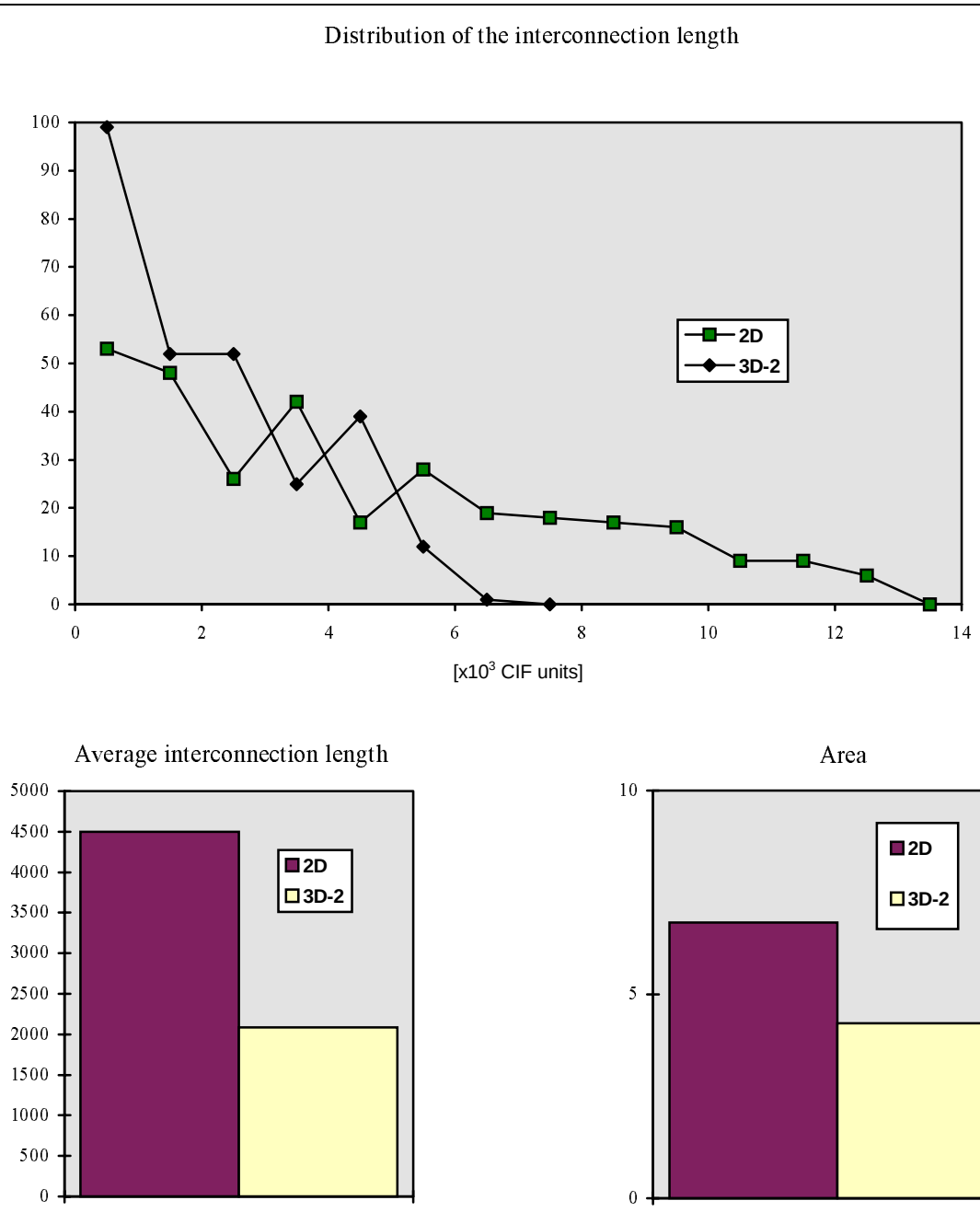
Slika 49. Eksperiment #5: raspodela dužina veza u slučaju 2D realizacije.

Opis: Uočeno je 312 veza (224 skupa povezanih geometrijskih primitiva); srednja dužina veza je 4501 CIF jedinica. Dimenzije jezgra čipa su 1944µm x 3394µm, odnosno površina jezgra čipa je 6.59 mm^2.



Slika 50. Eksperiment #5: raspodela dužina veza u slučaju realizacije u dve aktivne ravni.

Opis: Stepeni protočne obrade S#1 i S#2 realizovani su u različitim ravnima. U ravni #1 u kojoj je realizovan prvi stepen protočne obrade uočeno je 112 veza; srednja dužina veza u ravni #1 je 1660 CIF jedinica; dimenzije ravni #1 su 1104µm x 1328µm, odnosno površina je 1.466 mm^2. U ravni #2 u kojoj je realizovan drugi stepen protočne obrade uočeno je 168 veza; srednja dužina veza u ravni #2 je 2371 CIF jedinica; dimenzije ravni #2 su 1309µm x 1640µm; površina potrebna za realizaciju drugog stepena protočne obrade je 2.146 mm^2. Ukupna srednja dužina veza 3D čipa je 2086 CIF jedinica, a ukupna površina 3D čipa je 4.29 mm^2.



Slika 51. Eksperiment #5: uporedni prikaz raspodela dužina veza, srednjih dužina veza i površine jezgra čipova.

Opis: Srednja dužina veza u slučaju 3D realizacije je 45% srednje dužine veza u slučaju 2D realizacije. Ukupna površina jezgra čipa u slučaju 3D realizacije sa dva aktivna nivoa je 60% površine jezgra čipa u slučaju 2D realizacije.

******

U tabeli 3 prikazani su sumarni rezultati izvršenih eksperimenata. Vrednosti srednje dužine veza i ukupne površine čipa su skalirane prema vrednostima ovih veličina za slučaj 2D realizacija u okviru posmatranih eksperimenata.

	Srednja dužina veza				Ukupna površina čipa			
	2D	3D-2	3D-3	3D-4	2D	3D-2	3D-3	3D-4
Exp #1	100%	48%	-	29%	100%	60%	-	40%
Exp #2	100%	42%	-	-	100%	68%	-	-
Exp #3	100%	43%	-	18%	100%	45%	-	21%
Exp #4	100%	-	25%	-	100%		14%	-
Exp #5	100%	46%	-	-	100%	65%	-	-

Tabela 3. Rezultati eksperimenata.

Na osnovu dobijenih rezultata zaključuje se da je srednja dužina veza u slučaju 3D realizacija sa dva aktivna nivoa (3D-2) od 2.08 do 2.38 puta kraća od srednje dužine veza u slučaju 2D realizacije, zavisno od posmatranog eksperimenta. Skraćenje u slučaju 3D realizacija sa četiri aktivne ravni je između 3.4 i 5.5 puta. Ukupna površina čipa u slučaju 3D-2 realizacije je između 1.4 i 2.2 puta manja od površine u slučaju 2D realizacije. Smanjenje ukupne površine u slučaju 3D realizacije sa četiri aktivna nivoa je između 2.5 i 4.7 puta, u zavisnosti od posmatranog eksperimenta. U eksperimentu #4 srednja dužina veza za 3D-3 realizaciju je 4 puta manja od srednje dužine veza za 2D realizaciju sistoličkog polja sa devet procesnih elemenata.

Na osnovu prikazanih rezultata može se zaključiti da je poboljšanje usled prelaska sa 2D VLSI na 3D VLSI tehnologiju značajno. Skraćenje srednje dužine veza i smanjivanje ukupne površine čipa je naročito izraženo u eksperimentima u kojima je broj veza relativno veliki, što je slučaj u eksperimentima #3 i #4.

10. Implementaciona analiza

Generalno, poglavlje o implementacionoj analizi podrazumeva diskusiju o kompleksnosti predloženog rešenja. U konkretnom slučaju, data je diskusija o kompleksnosti predloženog pristupa analizi veza na čipu. Takođe, dat je i osvrt na eventualni dalji razvoj programskog paketa MARS u pravcu profesionalne verzije programa.

Predloženi pristup poređenju dužina veza na 2D i 3D čipovima baziran je na postojećim alatima za projektovanje 2D čipova po *standard cell* metodologiji. U slučaju 3D realizacija posmatra se svaka aktivna ravan posebno. Za veze koje povezuju standardne ćelije iz različitih aktivnih ravni usvojeno da je njihova prosečna dužina jednaka prosečnoj dužini veza u svakoj od aktivnih ravni. Na ovaj način, razvojem alata za analizu veza na 2D čipu omogućeno je poređenje 2D i 3D pristupa.

Programski paket za analizu veza na čipu koji je zadat u CIF formatu obuhvata nekoliko programa koji su detaljno opisani u poglavlju 6. Programski paket je razvijen pod operativnim sistemom Windows'95, odnosno DOS7.0. Najkompleksniji program programskog paketa MARS je program COREv26 koji vrši određivanje skupova povezanih geometrijskih primitiva, određivanje veza i izračunavanje statistike. Vreme izvršavanja ovog programa (reda nekoliko minuta) je neuporedivo kraće od vremena koje je potrebno programu za raspoređivanje sa povezivanjem, koji svoj posao obavi za nekoliko sati.

Programski paket MARS pored određivanja dužina veza omogućava sakupljanje raznih statističkih podataka. Za veze koje povezuju izlazne, odnosno ulazne terminale standardnih ćelija sa izlaznim, odnosno ulaznim pedovima omogućeno je određivanje dužine dela veze koji pripada jezgru čipa, odnosno dužine dela veze koji pripada oblasti za povezivanje jezgra čipa sa okvirom čipa. Pored osnovne namene, programski paket MARS se može koristiti i za poređenje efikasnosti različitih programskih paketa za razmeštanje sa povezivanjem.

Za sada, programski paket MARS je akademski alat. Ukoliko bi postojao interes, programski paket MARS bi mogao evoluirati u profesionalnu verziju, bilo kao deo nekog programskog paketa za raspoređivanje sa povezivanjem, bilo kao samostalni program. U tom cilju, bilo bi potrebno uraditi sledeće: (a) otkloniti zavisnost programa iz paketa MARS od komandi kojima L-Edit proširuje osnovnu sintaksu CIF formata, (b) realizovati *user-friendly* korisnički interfejs koji bi omogućio lako pokretanje programa i (c) realizovati interfejs za grafički prikaz dobijenih rezultata i razne vrste statistika. Sa druge strane, rezultati dobijeni programskim paketom MARS mogu se koristiti za *post-layout* simulaciju vremenskih odnosa, pa je od interesa povezivanje ovog programa sa alatima za *post-layout* simulaciju.

11. Zaključak

U opštem slučaju, zaključak treba da sadrži: (a) rekapitulaciju onoga što je urađeno i šta je glavni doprinos, (b) diskusiju na temu ko će da ima koristi od sprovedenog istraživanja i (c) diskusiju o novotvorenim problemima i mogućim putevima za njihovo rešavanje. Sledeći ovu strukturu u ovom poglavlju prvo je data rekapitulacija onoga što je urađeno, zatim diskusija o korisnosti sprovedenog istraživanja i na kraju izneti su novootvoreni problemi.

U sprovedenom istraživanju prikazan je jedan pristup kvantitativnoj analizi dužina veza kod 2D i 3D realizacija 2D sistoličkih polja za množenje matrica. Važnost problema ogleda se u činjenici da do sada nije data kvantitativna mera poboljšanja koja se dobija prelaskom sa 2D na 3D VLSI. Poboljšanje je izraženo kroz skraćivanje dužina veza čime se skraćuje kašnjenje na kritičnoj stazi za podatke (performansa) i smanjuje zahtevana površina čipa (performansa + kompleksnost). Analiza je vršena na primeru sistoličkih polja jer poseduju visoko regularnu strukturu, što olakšava postupak razmeštanja u 3D strukturama, u uslovima kada ne postoji alat za razmeštanje u 3D čipu. Takođe, očekuje se da razvojem VLSI tehnologije sistolička polja postanu jedna od glavnih aplikacija u domenu VLSI projektovanja čipova.

Istraživanje je zasnovano na metodologiji 3D rutiranja koja se bazira na postojećoj metodologiji 2D rutiranja za *standard cell* metodologiju projektovanja VLSI čipova. U analizi su korišćeni postojeći alati za projektovanje 2D VLSI čipova i originalno razvijeni programski paket MARS za analizu dužina veza na 2D čipu koji je zadat u CIF formatu. Programski paket za analizu dužina veza na čipu radi sledeće: (a) izdvaja sve relevantne geometrijske primitive koje formiraju veze između terminalnih elemenata standardnih ćelija, (b) formira skupove povezanih geometrijskih primitiva, (c) pronalazi veze u okviru skupova geometrijskih primitiva i (d) izračunava dužine veza i druge statistike od interesa.

Izvedeno je nekoliko eksperimenata u kojima se analiziraju različite realizacije sistoličkih polja za množenje matrica. Svaki eksperiment uključuje klasičnu 2D realizaciju i jednu ili više predloženih 3D realizacija. Realizacije se porede u pogledu raspodele dužine veza, srednje dužine veza i ukupne potrebne površine čipa.

Dobijeni rezultati pokazuju da se prelaskom na 3D VLSI značajno povećava performansa, naročito u uslovima kada kašnjenje na vezama dominantno određuje performansu, što je slučaj, naročito u savremenoj submikronskoj VLSI tehnologiji. Srednja dužina veza se smanjuje od 2 do 2.5 puta prelaskom sa 2D realizacije na 3D realizaciju sa dva aktivna nivoa, u zavisnosti od eksperimenta. U slučaju 3D realizacije sa četiri aktivna nivoa srednja dužina veza je od 3 do 5 puta kraća u odnosu na srednju dužinu veza u slučaju 2D realizacije. Ukupna površina čipa u slučaju 3D realizacije (koja je jednaka maksimalnoj površini u jednoj ravni pomnoženoj sa brojem ravni) smanjuje se od 1.6 do 2 puta u odnosu na površinu u slučaju 2D realizacije, u zavisnosti od eksperimenta. Ukupna površina čipa u slučaju 3D realizacije sa četiri aktivne ravni je od 2.5 do 5 puta manja u odnosu na 2D realizaciju, u zavisnosti od eksperimenta.

Rezultati ovog istraživanja pokazuju potencijalno veliki dobitak u performansi i kompleksnosti prilikom prelaska sa 2D na 3D VLSI. I pored činjenice da su eksperimenti rađeni za posebnu klasu sistoličkih polja, rezultati su od interesa ne samo za buduće dizajnere sistoličkih polja, već i za dizajnere svih VLSI uređaja.

Očekuje se da će novi pristup naći prvo primenu u aplikacijama gde su gustina pakovanja, performansa i mala kompleksnost od posebnog značaja. Ovo istraživanje je delom inicirano iz JPL-a (*Jet Propulsion Labaratory*) koji je deo NASA (*National Aeronautics and Space Administration*). Naime, NASA je usvojila novi koncept svemirskih istraživanja koji podrazumeva da se prihvatanje i obrada podataka radi u svemiru, a da se gotovi rezultati šalju na Zemlju, za razliku od dosadašnjeg pristupa gde su se podaci prikupljali u svemiru i prosleđivali na Zemlju gde je vršena obrada. Takođe, zahteva se da novi sistemi budu malih dimenzija da bi se minimizovali troškovi slanja u svemir, male potrošnje i velike procesne snage. Imajući u vidu prednosti 3D VLSI pristupa u pogledu gustine pakovanja (manje dimenzije), performanse (kraći ciklus takta) i kompleksnosti (manja površina) jasno je da je 3D pristup glavni kandidat za ovakve sisteme.

Ovo istraživanje je relativno novo i postoji dosta novih otvorenih problema. Pre svih, to je problem razvoja CAD alata za projektovanje 3D VLSI čipova, posebno programa za raspoređivanje stanadardnih ćelija u 3D čipu i programa za 3D rutiranje. Takođe, od interesa je razvoj alata koji bi omogućili egzaktnu analizu uticaja dužine veza na performanse kola.

Jedan od interesantnih novootvorenih problema je i analiza 3D implementacija različitih tipova standardnih procesorskih resursa kao što su sabirači, množači, pomerači i registarski fajl i njihovo poređenje u pogledu performanse i kompleksnosti sa standardnim 2D implementacijama.

12. Literatura

[Tong95] C.C. Tong, C. Wu, "Routing in a Three-Dimensional Chip," *IEEE Transactions on Computers*, vol. 44, no. 1, 1995, pp. 106-117.

[Kung88] S. Y. Kung, *VLSI Array Microprocessors*, Prientic-Hall, Englewood Clifs, New Jersey, USA, 1988.

[Lakh96] S. Lakhani, Y. Wang, A. Milenković, V. Milutinović, "2D Matrix Multiplication on a 3D Systolic Array," *Microelectronics Journal*, vol. 27, no. 1, 1996.

[Lakh97] S. Lakhani, Y. Wang, A. Milenković, V. Milutinović, "3D Convolution on a 3D Systolic Array," *International Journal of Mini and Microprocessors*, vol. 19, no. 1, 1997.

[Mead80] C. Mead, L. Conway, *Introduction to VLSI Systems*, Addison-Wesley Publishing Company, Reading, Massachusetts, USA, 1980.

[Ullm84] J. Ullman, *Computational Aspects of VLSI*, Computer Science Press, Rockville, Marylend, USA, 1984.

[Milu94] V. Milutinović, *Projektovanje i arhitektura RISC procesora za VLSI*, Nauka, Beograd, Srbija, Jugoslavija, 1994.

[Milu94a] V. Milutinović, D. Božanić, D. Polomčić, M. Aleksić, *Uvod u projektovanje računarskih VLSI sistema*, Nauka, Beograd, Srbija, Jugoslavija, 1994.

[West85] N. Weste, Kamran Eshraghian, *Princples CMOS VLSI Design*, Addison-Wessley Publishing Company, Reading, Massachusetts, USA, 1985.

[Rebe95] M. Reber, R. Tielert, "Global Optimization of Physical Design for Vertically Stacked Integrated Circuit," 3D Packaging, Advanced Technology Workshop, Los Angeles, Octobar, 1995.

[Tann87a] *SchemLib - A Schematic Library Condensed Manual*, 1987, Tanner Research, Pasadena, California, USA.

[Tann87b] *GayeSim - A Gate Level Simulator Condensed Manual*, 1987, Tanner Research, Pasadena, California, USA.

[Tann90] *L-Edit User's Manual*, Tanner Research, 1990, Pasadena, California, USA.

[Dehk93] P. Dehkordi, D. Pouldin, “Design for Packageability,” *IEEE Computer*, vol. 26, no. 4, 1993, pp. 76-81.

[Akas88] Y. Akasaka, T. Nishimura, H. Nakata, “Trends in Three-Dimensional Integration,” *Solid State Technology*, vol. 28, 1988., pp. 81-89.

[Koku86] A. Kokubu, “Three Dimensional Device Project,” Tutorial of the ISCA-86, Tokyo, Japan, June1986.

[Grin84] J. Grinberg, G. Nudd, D. Etchells, “A Cellular VLSI Architecture,” *IEEE Computer*, vol. 17, no. 1, 1986., pp. 69-80.

[Fort87] J. Fortes, B. Wah, “Systolic Arrays - From Concept to Implementation,” *IEEE Computer*, vol. 20, no. 5, 1987, pp. 12-17.

[Aboe87] M. Aboelaze, B. Wah, “Complexities of Layouts in Three-Dimensional VLSI Circuits,” IEEE International Symposium on Circuits and Systems, May, 1987, Philadelphia, PA, USA.

[Malh87] S. D. S. Malhi, H.E. Davis, R. J. Stierman, K. E. Bean, C. C. Driscoll, P. K. Chatterjee, “Orthogonal Chip Mount - A 3D Hybrid Wafer Scale Integration Technology,” *Technical Digest, International Electron Devices Meeting*, 1987, pp. 104-106.

[Phil88] R. Philhower, J.F. McDonald, “Interconnection Complexity Study for Piggy Back WSHP GaAs Systolic Processor,” IEEE International Conference on Systolic Arrays, May 1988, San Diego, California, USA.

Apendiks #1:
BNF notacija CIF formata

U ovom apendiksu prikazana je osnovna BNF notacija CIF formata.

```
cifFile                 =    {{blank} [command] semi} endCommand {blank}.

command                 =    primCommand | defDeleteCommand |
                             defStartCommand semi {{blank}
                             [primCommand]semi}defFinishCommand.

primCommand             =    polygonCommand | boxCommand |
                              roundFlashCommand | wireCommand |
                             layerCommand | callCommand |
                             userExtensionCommand | commentCommand.

polygonCommand          =    "P" path.

boxCommand              =    "B" integer sep integer sep point
                             [sep point].

roundFlashCommand       =    "R" integer sep point.

wireCommand             =    "W" integer sep path.

layerCommand            =    "L" {blank} shortname.

defStartCommand         =    "D" {blank} "S" integer
                             [sep integer sep integer].

defFinishCommand        =    "D" {blank} "F".

defDeleteCommand        =    "D" {blank} "F".

callCommand             =    "C" integer transformation.

userExtensionCommand    =    digit UserText.

commentCommand          =    "(" commentText ")".

endCommand              =    "E".

transformation          =    {{blank} ("T" point | "M" {blank} "X" |
                             "M" {blank} "Y" | "R" point)}.

path                    =    point {sep point}.

point                   =    sInteger sep sInteger.

sInteger                =    {sep}["-"]integerD.

integer                 =    {sep} integerD.

integerD                =    digit {digit}.
```

```
shortname               =     c[c][c][c].
c                       =     digit | upperChar.

userText                =     {userChar}.

commentText             =     {commentChar} | commentText
                              “(“commentText“)” commentText.

semi                    =     {blank} “;” {blank}.

sep                     =     upperChar | blank.

digit                   =     “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7”
                              | “8” | “9”.

upperChar               =     “A” | “B” | ... | “Z”.

blank                   =     any ASCII character except digit, upperChar,
                              “-“, “(“, “)”, or “;”.

userChar                =     any ASCII character except “;”.

commentChar             =     any ASCII character except “(“ or “)”.
```

Apendiks #2: MARS - programski alat za analizu dužina veza

Ovaj apendiks sadrži izvorni kod programa koji sačinjavaju programski paket MARS.

TRCIFv3

```
# include <fstream.h>
# include <string.h>
# include <stdlib.h>
# include <iomanip.h>

# define IN 0  // input terminal type
# define OUT 1  // output terminal type

void extrcoor(char *buf, int *px, int *py)
{
 char *p = new(20);
 p=strtok(buf, " ");
 p=strtok(NULL, " ");
 p=strtok(NULL, " ");
 *px = atoi(p);
 p=strtok(NULL, " ");
 *py = atoi(p);
}

void extrccva(char *buf, int *px, int *py)
{
 char *p=strtok(buf, " ");
 p=strtok(NULL, " ");
 p=strtok(NULL, " ");
 p=strtok(NULL, " ");
 *px = atoi(p);
 p=strtok(NULL, ";");
 *py = atoi(p);
}

typedef struct TERM_ENTRY
{
 int cx;
 int cy;
 int type;
 };

void main(int argc, char **argv)
{

 // Check weather there are enough arguments
 if (argc < 3)
        {
         cerr << "Usage: trcifv3 infile outfile\n";
         exit(0);
        }
 // Open the input file and connect it to stream "ins"
 ifstream ins(argv[1]);
 if (!ins)
        {
         cerr << "Cannot open: " << argv[1];
         exit(1);
        }
 // Open the output file and connect it to stream "outs"
 ofstream outs(argv[2]);
```

```
if (!outs)
        {
         cerr << "Cannot open: " << argv[2];
         exit(1);
        }

const bufsize = 80;
char buf[bufsize], buf1[bufsize];
char c;
int iwflag = 0, idefs=0, ichann=0, irow=0, icore=0, ichip=0,
         irlcms=0, iclcms=0, ichlcms=0, igate=0, iframe=0,
         igatecva=0;
int outx, outy, cvax, cvay;
TERM_ENTRY termt[10];  // terminal table of a cell
int itermt=0;          // index for terminal table

while (ins.get(buf, bufsize) && outs)
        {
              if (buf[0] == '(') iwflag =1;
              else if (strstr(buf, "DS")) {iwflag =1; idefs=1;}
              else if (strstr(buf, "DF;")) {iwflag=1; idefs=0; ichann=0;
                                    icore=0; iframe=0; irow=0; ichip=0;
                                    irlcms=0;iclcms=0; ichlcms=0;
                                    igate=0; itermt=0;}
              else if (buf[0] == '9'&& buf[1]==' ')
                    {
                     iwflag =1;
                     if (strstr(buf, "Channel~") && idefs) ichann=1;
                     else if (strstr(buf, "Core") && idefs) icore =1;
                     else if (strstr(buf, "Row~") && idefs) irow =1;
                     else if (strstr(buf, "Chip") && idefs) ichip =1;
                     else if (strstr(buf, "Frame") && idefs) iframe =1;
                     else if (!((strstr(buf, "Pad") || strstr(buf,"Tie") ||
                              strstr(buf, "Crosser"))) && idefs) igate =1;
                    }
              else if (ichann && (strstr(buf, "LCMF") || strstr(buf, "LCMS")
                       || strstr(buf, "LCVA"))) iwflag =1;
              // LCMS in Row
              else if (irow && strstr(buf, "LCMS")) {iwflag =1; irlcms =1;}
              else if (irow && buf[0]!='B' && irlcms) irlcms=0;
              else if (irow && irlcms) iwflag=1;
              // LCMS in Core
              else if (icore && strstr(buf, "LCMS")) {iwflag =1; iclcms=1;}
              else if (icore && buf[0]!='B' && iclcms) iclcms=0;
              else if (icore && iclcms) iwflag =1;
              // LCMS in Chip
              else if (ichip && strstr(buf, "LCMS")) {iwflag =1; ichlcms=1;}
              else if (ichip && buf[0]!='B' && ichlcms) ichlcms=0;
              else if (ichip && ichlcms) iwflag=1;
              // for gate extract only CVA layer
              //output terminal
              else if (igate && buf[0] =='9' && buf[1]=='4' &&
              (strstr(buf, "Out") || strstr(buf, "Q")))
                     {
                      iwflag=0; cout << buf << " OUT\n";
                      extrcoor(buf, &outx, &outy);
                      termt[itermt].cx = outx;
                      termt[itermt].cy = outy;
                      termt[itermt].type = OUT;
                      itermt++;
                     }
              // input terminal
              else if (igate && buf[0]=='9' && buf[1]=='4' && strstr(buf, "CMS")
                       && !strstr(buf, "Cross"))
                       {
                        iwflag=0; cout << buf << " IN\n";
                        extrcoor(buf, &outx, &outy);
                        termt[itermt].cx = outx;
                        termt[itermt].cy = outy;
                        termt[itermt].type = IN;
                        itermt++;
                       }
              else if (igate && strstr(buf, "LCVA")) {iwflag=1; igatecva=1;}
              else if (buf[0]!='B' && igatecva) {igatecva=0;}
              else if (igatecva)
                    {
                     for(int i=0; i<strlen(buf); i++) buf1[i]=buf[i];
                     extrccva(buf1, &cvax, &cvay);
                     iwflag =0;
                     for(i=0; i<itermt; i++)
```

```
                        if ((termt[i].cx<=cvax+2)&&(termt[i].cx>=cvax-2)&&
                                (termt[i].cy<=cvay+2)&& (termt[i].cy>=cvay-2))
                                {
                                      iwflag =1;
                                      if (termt[i].type==IN)
                                              strcat(buf, " <in>");
                                      else
                                              strcat(buf, " <out>");
                                      break;
                                }
                         }
               else if (buf[0]== 'B' && idefs && ichann) iwflag =1;
               else if (buf[0]== 'C' && (icore || irow || ichip)) iwflag=1;
               else if ((buf[0]=='C'&& !iframe) || buf[0]=='E') iwflag=1;
               else iwflag = 0;
               if (iwflag)
                    {
                      outs.write(buf, strlen(buf));
                      ins.get(c); outs.put(c);
                    }
               else
                 ins.get(c);
         } // end while
 ins.close();
 outs.close();
} // end main
```

TRFv6

```
# include <fstream.h>
# include <string.h>
# include <stdlib.h>

typedef struct TRAN
{
 char stype[2];
 int xt;
 int yt;
};

typedef struct LIST_ITEM
{
 LIST_ITEM *pnitem;
 int isym;
 TRAN rtran;
};

typedef struct TABLE_ENTRY
{
 char sname[15];
 TRAN rtran;
 LIST_ITEM *plist;
};

int extrint(char *input)
{

 char *pb = strpbrk(input, "0123456789");
 return atoi(pb);
}

/*
procedure extran extracts the transformation described
with (type, x, y). It is considered that exists only one
transformation in the sequence of the transformations;
the type of that transformation is translation.
*/
void extrtran(char *input, TRAN *tran)
{
 char *p = strtok(input, " ");
 if (p)
        {
         p = strtok(NULL, " ");
         if (p[0]=='T')
                 {
        tran->stype[0] = 'T';
        p = strtok(NULL, " ");
        tran->xt = atoi(p);
```

```
        p = strtok(NULL, " ");
        tran->yt = atoi(p);
                }
        }
}

/*
procedure extrpoly applies transformation on the box
given with B xl yl cx cy
*/
void extrpoly(char *input, TRAN *tran, char *p1)
{
 char *p = strtok(input, " ");
 if (p[0] == 'B')
        {
          p1[0] = p[0]; p1[1] = '\0';
          p1 = strcat(p1, " ");
          p = strtok(NULL, " ");
          p1 = strcat(p1, p); p1 = strcat(p1, " ");
          p = strtok(NULL, " ");
          p1 = strcat(p1, p); p1 = strcat(p1, " ");
          p = strtok(NULL, " ");
          int x = atoi(p) + tran->xt;
          char pom[10];
          itoa(x, pom, 10);
          p1 = strcat(p1, pom); p1 = strcat(p1, " ");
          p = strtok(NULL, " ");
          int y = atoi(p) + tran->yt;
          itoa(y, pom, 10);
          p1 = strcat(p1, pom); p1 = strcat(p1, ";");
          p= strtok(NULL, "\0");
          if (p) p1=strcat(p1, p);         // for in and out comments
        }
}


void main(int argc, char **argv)
{
 // Check weather there are enough arguments
 if (argc < 3)
         {
          cerr << "Usage: trf3 infile outfile\n";
          exit(0);
         }
 // Open the input file and connect it to stream "ins"
 ifstream ins(argv[1]);
 if (!ins)
         {
          cerr << "Cannot open: " << argv[1];
          exit(1);
         }
 // Open the output file and connect it to stream "outs"
 ofstream outs(argv[2]);
 if (!outs)
         {
          cerr << "Cannot open: " << argv[2];
          exit(1);
         }

 const bufsize = 80;
 char buf[bufsize];
 char c;
 const tablesize=200;
 TABLE_ENTRY table[tablesize];

 int isymbol;  // identification number of the symbol
 int ivalid = 0; // flag for "inside symbol definition"

 // make symbols table
 while (ins.get(buf, bufsize))
         {
               if (buf[0]=='D' && buf[1]=='S')
                {
                 ivalid = 1;
                 isymbol = extrint(buf);
                 // control print
                 cout << isymbol << "\n";
                }
               else if (buf[0] == 'D' && buf[1] == 'F')
                 ivalid = 0;
```

```
              else if (buf[0]=='9' && buf[1]==' ')
               {
                char *pom = strtok(buf, " ");
                pom = strtok(NULL, " ");
                for(int i=0; i<strlen(pom); i++)
                      {
                       table[isymbol].sname[i] = pom[i];
                       // control print
                       cout << table[isymbol].sname[i];
                      }
                cout << "\n";
                table[isymbol].rtran.stype[1] = 'T';
                table[isymbol].rtran.xt = 0;
                table[isymbol].rtran.yt = 0;
                table[isymbol].plist = NULL;
               }
              else if (buf[0]=='C'&& ivalid)
               {
                LIST_ITEM  *litem;
                litem = new LIST_ITEM;
                // init litem
                litem->pnitem = NULL;
                litem->isym = extrint(buf);
                //extract transformation
                extrtran(buf, &litem->rtran);
                // put new item at the end of the list
                LIST_ITEM *ptr = table[isymbol].plist;
                if (!ptr) table[isymbol].plist = litem;
                else
                      {
                       LIST_ITEM *sptr = ptr;
                       while (ptr) { sptr=ptr; ptr = ptr->pnitem;}
                       sptr->pnitem = litem;}
                      }
              ins.get(c);
         }
 // end of make table
ins.close();
ins.open(argv[1]);

ofstream olcmf("lcmf.ci");
ofstream olcms("lcms.ci");
ofstream olcva("lcva.ci");
int ilcmf=0, ilcms=0, ilcva=0;
char obuf[80];

while (isymbol > 0)
{
       LIST_ITEM *p2 = table[isymbol].plist;
       if (p2)
              { // expand isymbol
                    // durinf expansion apply transformation define in the
                    // corresponding table entry
                    cout << "expansion " << isymbol << "\n";
                    ins.seekg(0);
                    while(ins.get(buf, bufsize))
                      {
                            ins.get(c);
                            if (strstr(buf, "DS") && extrint(buf)==isymbol) break;
                      };
                    while(ins.get(buf, bufsize))
                      {
                            if (strstr(buf, "DF;")) {ilcmf=0; ilcms=0; ilcva=0; break;}
                            else if (strstr(buf, "LCMF")) {ilcmf=1; ilcms=0; ilcva=0;}
                            else if (strstr(buf, "LCMS")) {ilcmf=0; ilcms=1; ilcva=0;}
                            else if (strstr(buf, "LCVA")) {ilcmf=0; ilcms=0; ilcva=1;}
                            else if (buf[0]=='B' && ilcmf)
                              {
                                    extrpoly(buf, &table[isymbol].rtran, obuf);
                                    olcmf.write(obuf, strlen(obuf));
                                    olcmf.put('\n');
                                    olcmf.flush();
                              }
                            else if (buf[0]=='B' && ilcms)
                              {
                                    extrpoly(buf, &table[isymbol].rtran, obuf);
                                    if (strstr(table[isymbol].sname, "Chip"))
                                            strcat(obuf, "<oc>");
                                    olcms.write(obuf, strlen(obuf));
                                    olcms.put('\n');
```

```
                              olcms.flush();
                            }
                          else if (buf[0]=='B' && ilcva)
                            {
                                  extrpoly(buf, &table[isymbol].rtran, obuf);
                                  olcva.write(obuf, strlen(obuf));
                                  olcva.put('\n');
                                  olcva.flush();
                            };
                          ins.get(c);
                     }
            };
      while (p2)
            {
             if (table[p2->isym].plist)
                   {// there are elements in the list
                    // apply transformation on the symbol isym
                    cout << "transform table " << p2->isym << "\n";
                    if (p2->rtran.stype[0] == 'T')
                          {
                           table[p2->isym].rtran.xt += p2->rtran.xt;
                           table[p2->isym].rtran.yt += p2->rtran.yt;
                          };
                    // apply transformation on the all symbols
                    // which are in list isym
                    LIST_ITEM *p3 = table[p2->isym].plist;
                    while (p3)
                          {
                           // apply transformation
                           cout << "tansform list " << p3->isym << "\n";
                           if ((p3->rtran.stype[0] == 'T') &&
                                       (p2->rtran.stype[0] == 'T'))
                                 {
                                  p3->rtran.xt += p2->rtran.xt;
                                  p3->rtran.yt += p2->rtran.yt;
                                 };
                           p3 = p3->pnitem;
                          }
                   }
            else
              {// expand isym with transformation
                    //
                    cout << "expansion " << p2->isym << "\n";
                    // expand isym with transformation p2->tran
                    ins.seekg(0);
                    while(ins.get(buf, bufsize))
                     {
                      ins.get(c);
                      if (strstr(buf, "DS") && extrint(buf)==p2->isym) break;
                      };
                    while(ins.get(buf, bufsize))
                      {
                            if (strstr(buf, "DF;")) { ilcmf=0; ilcms=0; ilcva=0; break;}
                            else if (strstr(buf, "LCMF")) {ilcmf=1; ilcms=0; ilcva=0;}
                            else if (strstr(buf, "LCMS")) {ilcmf=0; ilcms=1; ilcva=0;}
                            else if (strstr(buf, "LCVA")) {ilcmf=0; ilcms=0; ilcva=1;}
                            else if (buf[0]=='B' && ilcmf)
                              {
                                    extrpoly(buf, &p2->rtran, obuf);
                                    olcmf.write(obuf, strlen(obuf));
                                    olcmf.put('\n');
                                    olcmf.flush();
                              }
                      else if (buf[0]=='B' && ilcms)
                              {
                               extrpoly(buf, &p2->rtran, obuf);
                               olcms.write(obuf, strlen(obuf));
                               olcms.put('\n');
                               olcms.flush();
                              }
                      else if (buf[0]=='B' && ilcva)
                            {
                             extrpoly(buf, &p2->rtran, obuf);
                             olcva.write(obuf, strlen(obuf));
                             olcva.put('\n');
                             olcva.flush();
                            };
             ins.get(c);
            }
      };
```

```
         p2 = p2->pnitem;
        }
        isymbol = isymbol - 1;
  }
  olcmf.close();
  olcms.close();
  olcva.close();

  ifstream iflcmf("lcmf.ci");
  ifstream iflcms("lcms.ci");
  ifstream iflcva("lcva.ci");

  // make outs in CIF format
  outs.write("DS1 100 2;\n", 11);
  outs.write("LCMF;\n", 6);
  while(iflcmf.get(c) && outs) outs.put(c);
  outs.write("LCMS;\n", 6);
  while(iflcms.get(c) && outs) outs.put(c);
  outs.write("LCVA;\n", 6);
  while(iflcva.get(c) && outs) outs.put(c);
  outs.write("DF;\nC1;\nE\n", 10);
  outs.close();
}
```

EXTv4

```
/* Description:

 input files: lcmf.ci, includes cif description of the cif layer
              lcms.ci,
              lcva.ci,
 izlazni fajl:  user defined name

 Date: August, 1996
*/

# define VIA 0
# define IN 1
# define OUT 2
# define OTHER 2

#define INCORE 0
#define OUTCORE 1

# include <fstream.h>
# include <string.h>
# include <stdlib.h>

typedef struct PPOLY
{
 int lx;
 int ly;
 int cx;
 int cy;
};

typedef struct ENCVA
{
 PPOLY poly;
 char type;
};

typedef struct ENCMF
{
 PPOLY poly;
 char type;
};

typedef struct ENCMS
{
 PPOLY poly;
 char type;
 char place;
};

typedef struct POINT
{
 public:
```

```
  int x;
  int y;
};


/*
procedure extppoly extracts parameters
lx - duzina po x osi   ly - duzina po y osi
cx - x koordinata centra      cy - y koordinata centra
*/
void extppoly(char *input, PPOLY* poly)
{
 char *p = strtok(input, " ");
 if (p[0] == 'B')
        {
         p = strtok(NULL, " ");
         poly->lx = atoi(p);
         p = strtok(NULL, " ");
         poly->ly = atoi(p);
         p = strtok(NULL, " ");
         poly->cx = atoi(p);
         p = strtok(NULL, " ");
         poly->cy = atoi(p);
        }
}


// begin of cross procedure
int cross(PPOLY p1, PPOLY p2)
{
 POINT er, // end point right
               el,
               et,
               eb;
 // compute endpoints
 er.x = p1.cx - p1.lx/2;
 el.x = p1.cx + p1.lx/2;
 er.y = p1.cy;
 el.y = p1.cy;

 eb.y = p1.cy - p1.ly/2;
 et.y = p1.cy + p1.ly/2;
 eb.x = p1.cx;
 et.x = p1.cx;

 if (((p2.cx >= er.x-p2.lx/2) && (p2.cx <= el.x+p2.lx/2)) &&
        ((p2.cy >= eb.y-p2.ly/2) && (p2.cy <= et.y+p2.ly/2)))
        return 1;
 else
        return 0;
}
// end of cross procedure



/*
 mline: make line
*/
void mline(int lx, int ly, int cx, int cy, char* b)
{
 b[0]='B'; b[1]=' '; b[2]='\0';
 char p[10];
 itoa(lx, p, 10);
 strcat(b,p); strcat(b, " ");
 itoa(ly, p, 10);
 strcat(b,p); strcat(b, " ");
 itoa(cx, p, 10);
 strcat(b,p); strcat(b, " ");
 itoa(cy, p, 10);
 strcat(b,p); strcat(b, ";");
}


void main(int argc, char **argv)
{

 if (argc < 2)
        {
         cerr << "Usage: extv4 outfile\n";
         exit(0);
```

```
        }

// Open the output file and connect it to stream ofcif
ofstream ofcif(argv[1]);

if (!ofcif)
        {
         cerr << "Cannot open: " << argv[1];
         exit(1);
        }

const bufsize = 80;
char buf[bufsize], buf1[bufsize];
char c;
PPOLY poly;      // saves current box parameter

ifstream iflcmf("lcmf.ci");
ifstream iflcms("lcms.ci");
ifstream iflcva("lcva.ci"); // input file with cva description

ofstream ofvia("via.ci");
ofstream ofin("in.ci");
ofstream ofout("out.ci");
ofstream ofcmf("cmf.ci");
ofstream ofcms("cms.ci");

const maxcva = 10000;
const maxcmf = 10000;
const maxcms = 20000;

ENCVA tcva[maxcva];
ENCMF tcmf[maxcmf];
ENCMS tcms[maxcms];
int ncva=0,
         ncmf=0,
         ncms=0;
char place;

// create tcva table
while(iflcva.get(buf, bufsize))
 {
       for(int i=0; i<strlen(buf); i++) buf1[i]=buf[i];
       buf1[i]='\0';
       extppoly(buf, &poly);
       if (ncva >= maxcva)
              {
               cerr << "Not enough entries in tcva\n";
               exit(1);
              };
       tcva[ncva].poly = poly;
       if (strstr(buf1, "<out>"))
              tcva[ncva].type = OUT;
       else if (strstr(buf1, "<in>"))
              tcva[ncva].type = IN;
       else
              tcva[ncva].type = VIA;
       iflcva.get(c);
       ncva++;
 };
cout << "tcva table is created with " << ncva << " entries\n";
// create tcmf table
while(iflcmf.get(buf, bufsize))
 {
       extppoly(buf, &poly);
       if (ncmf >= maxcmf)
              {
               cerr << "Not enough entries int tcmf\n";
               exit(1);
              };
       tcmf[ncmf].poly = poly;
       tcmf[ncmf].type = OTHER;
       iflcmf.get(c);
       ncmf++;
 };
cout << "tcmf table is created with " << ncmf << " entries\n";

// create tcms table
while(iflcms.get(buf, bufsize))
 {
       if (strstr(buf, "<oc>")) place=OUTCORE;
```

```
      else place = INCORE;
      if (ncms >= maxcms)
             {
              cerr << "Not enough entries int tcms\n";
              exit(1);
             };
      extppoly(buf, &poly);
      tcms[ncms].poly = poly;
      tcms[ncms].type = OTHER;
      tcms[ncms].place = place;
      iflcms.get(c);
      ncms++;
 };
 cout << "tcms table is created with " << ncms << " entries\n";

for(int icva=0; icva<ncva; icva++)
 {
      if(tcva[icva].type==OUT)
        {
             cout << "out terminal\n";
             //make out buf
             mline(8,8,tcva[icva].poly.cx,tcva[icva].poly.cy, buf);
             // write
             ofout.write(buf, strlen(buf));
             ofout.put('\n');
        }
      else if (tcva[icva].type==IN)
        {
             cout << "in terminal\n";
             //make out buf
             mline(8,8,tcva[icva].poly.cx,tcva[icva].poly.cy, buf);
             // write
             ofin.write(buf, strlen(buf));
             ofin.put('\n');
        }
      else
        {
             int icms, icmf;
             int fcmf=0, fcms=0, ffcmf=0, ffcms=0;
             cout << "check cmf tabele\n";
             for(icmf=0; icmf<ncmf; icmf++)
                    if (tcmf[icmf].poly.lx==8 && tcmf[icmf].poly.ly==8 &&
                           tcmf[icmf].poly.cx==tcva[icva].poly.cx &&
                           tcmf[icmf].poly.cy==tcva[icva].poly.cy)
                                  {fcmf=1; break;}
             for(icms=0; icms<ncms; icms++)
                    if (tcms[icms].poly.lx==8 && tcms[icms].poly.ly==8 &&
                             tcms[icms].poly.cx==tcva[icva].poly.cx &&
                             tcms[icms].poly.cy==tcva[icva].poly.cy)
                                  {fcms=1; break;}

             if (fcmf && fcms)
                    {
                           // make out buf
                           mline(8,8,tcva[icva].poly.cx,tcva[icva].poly.cy, buf);
                           // write
                           cout << "via element\n";
                           ofvia.write(buf, strlen(buf));
                           ofvia.put('\n');
                           // change type in CMF and CMS table
                           tcmf[icmf].type = VIA;
                           tcms[icms].type = VIA;
                    }
              else
                    {
                      for(icmf=0; icmf<ncmf; icmf++)
                             if (cross(tcva[icva].poly, tcmf[icmf].poly))
                                    {ffcmf=1; break;}
                      for(icms=0; icms<ncms; icms++)
                             if (cross(tcva[icva].poly, tcms[icms].poly))
                                     {ffcms=1; break;}
                      if (ffcmf||ffcms)
                                  {
                                   mline(8,8,tcva[icva].poly.cx,tcva[icva].poly.cy, buf);
                                   // write
                                   cout << "via element\n";
                                   ofvia.write(buf, strlen(buf));
                                   ofvia.put('\n');
                                  }
                    }
```

```
              }; // end of else section
  }; // end of for section
 ofvia.close();
 ofin.close();
 ofout.close();

 for(int icmf=0; icmf<ncmf; icmf++)
  if (tcmf[icmf].type==OTHER)
        {
         // make buf
         mline(tcmf[icmf].poly.lx,tcmf[icmf].poly.ly,
                              tcmf[icmf].poly.cx,tcmf[icmf].poly.cy, buf);
         // write
         ofcmf.write(buf, strlen(buf));
         ofcmf.put('\n');
        };

 for(int icms=0; icms<ncms; icms++)
  if (tcms[icms].type==OTHER)
        {
         // make buf
         mline(tcms[icms].poly.lx,tcms[icms].poly.ly,
                              tcms[icms].poly.cx,tcms[icms].poly.cy, buf);
         // write
         if (tcms[icms].place==OUTCORE)
       strcat(buf, "<oc>");
         ofcms.write(buf, strlen(buf));
         ofcms.put('\n');
        };
  ofcmf.close();
  ofcms.close();

  ifstream ifcmf("cmf.ci");
  ifstream ifcms("cms.ci");
  ifstream ifvia("via.ci");
  ifstream ifin("in.ci");
  ifstream ifout("out.ci");
  // make outs in CIF format
  ofcif.write("DS1 100 2;\n", 11);
  ofcif.write("LCMF;\n", 6);
  while(ifcmf.get(c) && ofcif) ofcif.put(c);
  ofcif.write("LCMS;\n", 6);
  while(ifcms.get(c) && ofcif) ofcif.put(c);
  ofcif.write("LCPG;\n", 6);
  while(ifvia.get(c) && ofcif) ofcif.put(c);
  ofcif.write("LCAA;\n", 6);
  while(ifin.get(c) && ofcif) ofcif.put(c);
  ofcif.write("LCCP;\n", 6);
  while(ifout.get(c) && ofcif) ofcif.put(c);
  ofcif.write("DF;\nC1;\nE\n", 10);
  ofcif.close();
}
```

COREv26

```
/*
 DESCRIPTION:
 AUTHOR:
 Aleksandar Milenkovic
 Date:
 sep-okt 1996
*/
#include <fstream.h>
#include <string.h>
#include <stdlib.h>

#define USED 1
#define UNUSED 0

#define INSIDE 0
#define OUTSIDE 1

#define CMF 0
#define CMS 1
#define VIA 2
#define IN 3
#define OUT 4
```

```
typedef struct POLY
{
 int cx;
 int cy;
 int lx;
 int ly;
};

typedef struct LITEM
{
 public:
  int en; // entry number
  LITEM *next;
  int type;
  int place;
};

typedef struct TENTRY
{
 public:
  POLY p;
  int flag;
  // reserved for extension
};

typedef struct TCMSENTRY
{
 public:
  POLY p;
  int flag;
  // reserved for extension
  int position;
};


void extrparam(char *buf, POLY *p)
{
 char *t=strtok(buf, " ");
 t=strtok(NULL, " ");
 p->lx=atoi(t);
 t=strtok(NULL, " ");
 p->ly=atoi(t);
 t=strtok(NULL, " ");
 p->cx=atoi(t);
 t=strtok(NULL, ";");
 p->cy=atoi(t);
}

  const tablesize = 15000;

  // cmf table
  int ncmf = 0; // number of elements in the cmf table
  TENTRY cmf[tablesize];
  // cms table
  int ncms = 0;
  TCMSENTRY cms[tablesize];
  // via table
  int nvia = 0;
  TENTRY via[tablesize];
  // in terminal table
  int nin = 0;
  TENTRY in[tablesize];
  // out terminal table
  int nout;
  TENTRY out[tablesize];

  // lset points to set of connected lines
  const NSET = 2000;
  LITEM *lset[NSET];
  int nls=0;
  int nused=0; // number of used cmf polygons

// point
typedef struct POINT
{
 public:
  int x;
  int y;
};
```

```
int cross(POLY p1, POLY p2)
{
 POINT er, // end point right
                el,
                et,
                eb;
 // compute endpoints
 er.x = p1.cx - p1.lx/2;
 el.x = p1.cx + p1.lx/2;
 er.y = p1.cy;
 el.y = p1.cy;

 eb.y = p1.cy - p1.ly/2;
 et.y = p1.cy + p1.ly/2;
 eb.x = p1.cx;
 et.x = p1.cx;

 if (((p2.cx >= er.x-p2.lx/2) && (p2.cx <= el.x+p2.lx/2)) &&
        ((p2.cy >= eb.y-p2.ly/2) && (p2.cy <= et.y+p2.ly/2)))
        return 1;
 else
        return 0;
}


// put: begins creating of a lset with a cmf polygon
void put(int en,int type, int nls)
{
 LITEM *n=new(LITEM);
 n->type=type;
 n->en=en;
 n->next=NULL;
 n->place=INSIDE;
 if (lset[nls])
       {
        LITEM *p=lset[nls], *t;
        while(p)
              {
       t=p;
       p=p->next;
              }
        t->next = n;
       }
 else
       lset[nls]=n;
 if (type==CMF) {cmf[en].flag=USED;nused++;}
 else if (type==CMS) cms[en].flag=USED;
 else if (type==VIA) via[en].flag=USED;
 else if (type==IN) in[en].flag=USED;
 else if (type==OUT) out[en].flag=USED;
}
// end of put procedure

// exlist procedure
void exlist(int en, int type, int nls)
{
 switch (type)
  {
       int j;
       case CMF:
              //via
              for(j=0;j<nvia;j++)
               if (!via[j].flag && cross(cmf[en].p,via[j].p)) //poly_via(en,j))
                      put(j, VIA, nls); // put via element in list
              break;

       case CMS:
              //cms
              for(j=0;j<ncms;j++)
               if (!cms[j].flag && cross(cms[en].p, cms[j].p))
                              put(j, CMS, nls);
              //via
              for(j=0;j<nvia;j++)
               if (!via[j].flag && cross(cms[en].p, via[j].p))
                              put(j, VIA, nls); // put via element in list
              //in
              for(j=0;j<nin;j++)
               if (!in[j].flag && cross(cms[en].p, in[j].p))
                              put(j, IN, nls); // put via element in list
```

```
                //out
                for(j=0;j<nout;j++)
                 if (!out[j].flag && cross(cms[en].p, out[j].p))
                                put(j, OUT, nls); // put via element in list
                break;

        case VIA:
                //cms
                for(j=0;j<ncms;j++)
                 if (!cms[j].flag && cross(cms[j].p, via[en].p))
                  // problem caused with via extension
                  // insert control mechanism
                        if (!((cms[j].p.lx==6) && ((cms[j].p.cx+cms[j].p.lx/2) ==
                                        (via[en].p.cx-via[en].p.lx/2))))
                                put(j, CMS, nls);
                //cmf
                for(j=0;j<ncmf;j++)
                 if (!cmf[j].flag && cross(cmf[j].p, via[en].p))
                                put(j, CMF, nls); // put via element in list
                break;

        case IN:
                break;

        case OUT:
                break;
  }
}
// end of exlist procedure

//part for terminal table
        typedef struct LCON {
          int en;
          LCON *next;
        };
        typedef struct TCON {
          int type;
          int en;
          LCON *ptrlt;  // pointer for the list of connected terminals
          POLY p;       // poly
          int nct; // number of connected terminals
          int stat; // for in elements (unused=0, used=1)
          LCON *ptrlb; // pointer for the list of connected boxes
        };

        const NTERM=1000;
        TCON termt[NTERM];
        int ntt=0;

// part for box table
        typedef struct BCON {
          int type;
          int en;
          LCON *ptrlb;
          POLY p;
          LCON *ptrclos;
        };
        const NBOX=1000;
        BCON boxt[NBOX];
        int nbt=0;

// lcput procedure
void lcput(int j, int k)
{
 LCON *p,*s, *n=new(LCON);
 n->en=k;
 n->next=NULL;

 p=termt[j].ptrlt;
 if (!p)
          termt[j].ptrlt=n;
 else
          {
                while(p)
                        {s=p; p=p->next;}
                s->next=n;
          };
 if (termt[k].type==VIA) (termt[j].nct)++;

 n=new(LCON);
```

```
 n->en=j;
 n->next=NULL;
 p=termt[k].ptrlt;
 if (!p)
         termt[k].ptrlt=n;
 else
         {
               while(p)
                       {s=p; p=p->next;}
               s->next=n;
         }
 if (termt[j].type==VIA) termt[k].nct++;
}
//end of lcput procedure

const maxcd=6; // maximal critical distance in ctwot procedure

// ctwot
int ctwot(int j, int k)
{
       int cut=0;
       LCON *pj=termt[j].ptrlb;
       while(pj)
         {
               LCON *pk=termt[k].ptrlb;
               while(pk)
                 {
                       if(pj->en==pk->en)
                         {cut=1; break;}
                       pk=pk->next;
                 }
               if(cut) break;
               pj=pj->next;
         };
       if(!cut)
         {
               pj=termt[j].ptrlb;
               LCON *pk=termt[k].ptrlb;
               LCON *c1=NULL, *c2=NULL, *tp, *ts, *n;
               while(pj)
                {
                 LCON *ttmp=boxt[pj->en].ptrclos;
                 while(ttmp)
                       {
                        if(!c1)
                               {
                                n=new(LCON);
                                n->en=ttmp->en; n->next=ttmp->next;
                                c1=n;
                               }
                        else
                               {
                                int alexist=0;
                                tp=c1;
                                while(tp)
                                       {
                                        if (tp->en==ttmp->en)
                                               {alexist=1; break;}
                                        ts=tp;
                                        tp=tp->next;
                                       }
                                if (!alexist)
                                       {
                                        n=new(LCON);
                                        n->en=ttmp->en;
                                        n->next=NULL;
                                        ts->next=n;
                                       }
                               };
                        ttmp=ttmp->next;
                       }
                 pj=pj->next;
                };
               while(pk)
                {
                 LCON *ttmp=boxt[pk->en].ptrclos;
                 while(ttmp)
                       {
                        if(!c2)
                               {
```

```
                        n=new(LCON);
                        n->en=ttmp->en; n->next=ttmp->next;
                        c2=n;
                       }
                  else
                       {
                        int alexist=0;
                        tp=c2;
                        while(tp)
                              {
                               if (tp->en==ttmp->en)
                                     {alexist=1; break;}
                               ts=tp; tp=tp->next;
                              }
                        if (!alexist)
                              {
                               n=new(LCON);
                               n->en=ttmp->en;
                               n->next=NULL;
                               ts->next=n;
                              }
                       };
                  ttmp=ttmp->next;
                 }
                pk=pk->next;
          };
         pj=c1;
         while(pj)
           {
                pk=c2;
                while(pk)
                  {
                       if (pj->en==pk->en)
                               {cut=1; break;}
                       pk=pk->next;
                  };
                if (cut) break;
                pj=pj->next;
           }
         delete c1;
         delete c2;
    } // end if (!cut)

  if (cut)
    {
         if ((termt[j].p.cx<=termt[k].p.cx+maxcd) &&
                 (termt[j].p.cx>=termt[k].p.cx-maxcd))
                { // at the same x line
                  // + insert something between them
                  int f=0;
                  for (int l=0;l<ntt;l++)
                              if ((l!=j) && (l!=k) &&
                                 (termt[l].p.cx<=termt[k].p.cx+maxcd)
                                 && (termt[l].p.cx>=termt[k].p.cx-maxcd) &&
                                        (((termt[l].p.cy < termt[k].p.cy) &&
                                        (termt[l].p.cy > termt[j].p.cy))||
                                        ((termt[l].p.cy < termt[j].p.cy) &&
                                        (termt[l].p.cy > termt[k].p.cy))))

                                       f=1;
                  if (f) return 0;
                  else return 1;
                }
         else if ((termt[j].p.cy<=termt[k].p.cy+maxcd) &&
                              (termt[j].p.cy>=termt[k].p.cy-maxcd))
                {
                  int f=0;
                  for (int l=0;l<ntt;l++)
                              if ((l!=j) && (l!=k)
                                 && (termt[l].p.cy<=termt[k].p.cy+maxcd)
                                      && (termt[l].p.cy>=termt[k].p.cy-maxcd) &&
                                      (((termt[l].p.cx <= termt[k].p.cx) &&
                                      (termt[l].p.cx >=termt[j].p.cx))||
                                      ((termt[l].p.cx <= termt[j].p.cx) &&
                                      (termt[l].p.cx >=termt[k].p.cx))))
                                             f=1;
                  if (f) return 0;
                  else return 1;
                 }
    else
```

```
                return 1;
         }
        else
          return 0;
}
// end of ctwot


const maxnrcstack=100;
const maxnstack=150;

int stack[maxnstack];
int nstack;

typedef struct TRSTACK
{
 int node;
 int rc1;
 int rc2;
};
TRSTACK rcstack[maxnrcstack];
int nrcstack;

int putonstack(void)
{
 int node =stack[nstack-1];
 LCON *ptr=termt[node].ptrlt;
 int alexist=0, cnt=0;
 while(ptr)
        {
         alexist=0;
         for(int j=0; j<nstack;j++)
                if(stack[j]==ptr->en)
         {alexist=1; break;}
         if (!alexist)
               {
                cnt++;
                if (cnt==1)
                        {
                                stack[nstack]=ptr->en;
                                nstack++;
                                cout << "putonstack: " << stack[nstack-1] << "\n";
                        }
                 else if ((cnt==2) && !alexist)
                        {
                                rcstack[nrcstack].node =  node;
                                rcstack[nrcstack].rc1 = ptr->en;
                                cout << "put on reserve stack: node= "
                                        << node << "rc1=" << ptr->en << "\n";
                                nrcstack++;
                        }
               else if ((cnt==3) && !alexist)
                        {
                         rcstack[nrcstack-1].rc2=ptr->en;
                         cout << "put second connection on reserve stack: node= "
                                << rcstack[nrcstack-1].node << "rc2= " << ptr->en << "\n";
                         }
               else
                      {
                                cerr << "e3 err: putonstack\n";
                                exit(1);
                      };
               };
         ptr=ptr->next;
 }//end while
 if (cnt) return 1;
 else return 0;
} // end procedute putonstack;

//============================
void main(int argc, char** argv)
{
  if (argc<2)
        {
         cerr << "Usage: core26 infile\n";
         exit(0);
        }
  ifstream ins(argv[1]);
  if (!ins)
        {
```

```
         cerr << "Cannot open: " << argv[1] << "\n";
         exit(1);
        }
 const bufsize = 80;
 char buf[bufsize];
 char c;
 POLY param;
 int icmf=0, icms=0, ivia=0, iin=0, iout=0;

 int i;
 int beg, end; // begin and end elements
 // fill tables with polygon parameters
 while (ins.get(buf, bufsize))
       {
        // save buf
        char buf1[80];
        for(int pcnt=0; pcnt<80; pcnt++) buf1[pcnt]=buf[pcnt];
        if(buf[0]=='B')
               {
                extrparam(buf, &param);
                if (icmf)
                      {
                       cmf[ncmf].p=param;
                       cmf[ncmf].flag=UNUSED;
                       ncmf++;
                      }
               else if (icms)
                      {cms[ncms].p=param; cms[ncms].flag=UNUSED;
                       if(strstr(buf1, "<oc>")) // cms poly is out of the core
                              cms[ncms].position=OUTSIDE;
                       else
                              cms[ncms].position=INSIDE;
                        ncms++;}
               else if (ivia)
                      {via[nvia].p=param; via[nvia].flag=UNUSED; nvia++;}
               else if (iin)
                      {in[nin].p=param; in[nin].flag=UNUSED; nin++;}
               else if (iout)
                      {out[nout].p=param; out[nout].flag=UNUSED; nout++;}
               }
        else if (buf[0]=='L' && buf[1]=='C'&&
                              buf[2]=='M' && buf[3]=='F')
               {
                icmf=1; icms=0; ivia=0; iin=0; iout=0;
               }
        else if (buf[0]=='L' && buf[1]=='C' &&
                               buf[2]=='M' && buf[3]=='S')
               {
                      icmf=0; icms=1; ivia=0; iin=0; iout=0;
               }
        else if (buf[0]=='L' && buf[1]=='C' &&
                 buf[2]=='P' && buf[3]=='G')
               {
                icmf=0; icms=0; ivia=1; iin=0; iout=0;
               }
        else if (buf[0]=='L' && buf[1]=='C' &&
                 buf[2]=='A' && buf[3]=='A')
               {
                      icmf=0; icms=0; ivia=0; iin=1; iout=0;
               }
        else if (buf[0]=='L' && buf[1]=='C' &&
                 buf[2]=='C' && buf[3]=='P')
               {
                      icmf=0; icms=0; ivia=0; iin=0; iout=1;
               };
        ins.get(c);
       };
// end of fill table

// make sets of connected polygons
for(i=0; i<NSET;i++) lset[i]=NULL;
while (nused < ncmf)
 {
        for(i=0;i<ncmf;i++)
              if (!cmf[i].flag)
        {put(i,CMF,nls); break;};

        LITEM *p=lset[nls];
        while(p)
              {
```

```
               exlist(p->en,p->type,nls);
               p=p->next;
              };
       nls++;
 }

//show results
//variables which count in and out terminals in a set

// find net frame
int outclen, termlen, ovlen, newoutclen, rememb;  // lengths

int nsin=0, nsout=0, nlset=0;
POINT top, bottom, left, right;

// variables needed for statistics
int nlt=0; // line counter
ofstream ofout("len.txt");
ofout.write("set\tline\tinclen\toutclen\ttotlen\n", 31);
double avtlen=0.0;
char pom[80], pom1[10];
// end of statistics
//distribution vector
const ndvect=100;
int dvect[ndvect];
for(i=0; i<ndvect; i++) dvect[i]=0;
for(i=0; i<nls; i++)
 {

      LITEM *p=lset[i];

      cout << "\n>>>>> Set " << i << "\n";
      while(p)
       {
        // show elements
        if (p->type==CMF)
              cout << "cmf[" << p->en << "] " <<
             cmf[p->en].p.lx/2  << " " <<
             cmf[p->en].p.ly/2 << " " <<
             cmf[p->en].p.cx/2. << " " <<
             cmf[p->en].p.cy/2. << "\n";
        else if (p->type==CMS)
              cout << "cms[" << p->en << "] " <<
             cms[p->en].p.lx/2  << " " <<
             cms[p->en].p.ly/2 << " " <<
             cms[p->en].p.cx/2. << " " <<
             cms[p->en].p.cy/2. << "\n";
        else if (p->type==VIA)
              cout << "via[" << p->en << "] " <<
             via[p->en].p.lx/2  << " " <<
             via[p->en].p.ly/2 << " " <<
             via[p->en].p.cx/2. << " " <<
             via[p->en].p.cy/2. << "\n";
        else if (p->type==IN)
              cout << "in[" << p->en << "] " <<
             in[p->en].p.lx/2  << " " <<
             in[p->en].p.ly/2 << " " <<
             in[p->en].p.cx/2. << " " <<
             in[p->en].p.cy/2. << "\n";
        else if (p->type==OUT)
              cout << "out[" << p->en << "] " <<
             out[p->en].p.lx/2  << " " <<
             out[p->en].p.ly/2 << " " <<
             out[p->en].p.cx/2. << " " <<
             out[p->en].p.cy/2. << "\n";
        p=p->next;
       }

      p=lset[i];
      nsin=0; nsout=0; nlset=0;
      top.x=0; top.y=-10000;
      bottom.x=0; bottom.y=10000;
      left.x=10000; left.y=0;
      right.x=-10000; right.y=0;
      while(p)
       {
        if (p->type==VIA)
              {
                    if (via[p->en].p.cx > right.x)
                      {
```

```
                        right.x=via[p->en].p.cx;
                        right.y=via[p->en].p.cy;
                  };
                if (via[p->en].p.cx < left.x)
                  {
                        left.x=via[p->en].p.cx;
                        left.y=via[p->en].p.cy;
                  };
                if (via[p->en].p.cy > top.y)
                  {
                        top.x=via[p->en].p.cx;
                        top.y=via[p->en].p.cy;
                  };
                if (via[p->en].p.cy < bottom.y)
                  {
                        bottom.x=via[p->en].p.cx;
                         bottom.y=via[p->en].p.cy;
                        }
                }
      else if (p->type==IN)
            {
                if (in[p->en].p.cx > right.x)
                  {
                        right.x=in[p->en].p.cx;
                        right.y=in[p->en].p.cy;
                        };
                if (in[p->en].p.cx < left.x)
                  {
                        left.x=in[p->en].p.cx;
                        left.y=in[p->en].p.cy;
                  };
                if (in[p->en].p.cy > top.y)
                  {
                        top.x=in[p->en].p.cx;
                        top.y=in[p->en].p.cy;
                  };
                if (in[p->en].p.cy < bottom.y)
                  {
                        bottom.x=in[p->en].p.cx;
                        bottom.y=in[p->en].p.cy;
                  };
                nsin++;
            }
      else if (p->type==OUT)
            {
                if (out[p->en].p.cx > right.x)
                  {
                        right.x=out[p->en].p.cx;
                        right.y=out[p->en].p.cy;
                  };
                if (out[p->en].p.cx < left.x)
                  {
                        left.x=out[p->en].p.cx;
                        left.y=out[p->en].p.cy;
                        };
                if (out[p->en].p.cy > top.y)
                  {
                        top.x=out[p->en].p.cx;
                        top.y=out[p->en].p.cy;
                  };
                if (out[p->en].p.cy < bottom.y)
                  {
                        bottom.x=out[p->en].p.cx;
                        bottom.y=out[p->en].p.cy;
                        };
           nsout++;
           }
      p=p->next;
     }

    p=lset[i];
    while(p)
      {
          if (p->type==CMS)
                if (!((cms[p->en].p.cx <= right.x+4) &&
                      (cms[p->en].p.cx >= left.x-4) &&
                      (cms[p->en].p.cy <= top.y+4)  &&
                      (cms[p->en].p.cy >= bottom.y-4)))
                        {
                              p->place=OUTSIDE;
```

```
                        }
        if (p->type==CMS && p->place)
                cout << "|cms[" << p->en << "] doesn't belong nt\n";
        if (p->type==CMS && cms[p->en].position)
               cout << "new cms[" << p->en << "]doesen't belong nt\n";

        p=p->next;
  };
// number of lines in a set
if(nsin)
  nlset=nsin;
else
  nlset=nsout;

outclen=0; newoutclen=0;  rememb=0;
if (!(nsin && nsout))
  {
        LITEM *pom=lset[i];
        while(pom)
          {
               if ((pom->place==OUTSIDE) && (pom->type==CMS))
                      if (cms[pom->en].p.lx==6)
                             outclen += cms[pom->en].p.ly;
                      else
                             outclen += cms[pom->en].p.lx;

               if ((pom->type==CMS) && (cms[pom->en].position==OUTSIDE))
                             if (cms[pom->en].p.lx==6)
                                    newoutclen += cms[pom->en].p.ly;
                             else
                                    newoutclen += cms[pom->en].p.lx;
               pom=pom->next;
           };
  };
rememb=outclen-newoutclen;// part added on incore length
cout << "fIN= " << nsin << "fOUT= " << nsout << "\n";
p=lset[i]; ntt=0; nbt=0;
while(p)
 {
  if ((p->type==VIA) ||(p->type==IN) || (p->type==OUT))
               {
                termt[ntt].type=p->type;
                termt[ntt].en=p->en;
                termt[ntt].ptrlt=NULL;
                termt[ntt].nct=0;
                termt[ntt].stat=UNUSED;
                termt[ntt].ptrlb=NULL;
                if (p->type==VIA)
                       {termt[ntt].p.cx=via[p->en].p.cx;
                        termt[ntt].p.cy=via[p->en].p.cy;
                        termt[ntt].p.lx=via[p->en].p.lx;
                        termt[ntt].p.ly=via[p->en].p.ly;}
                else if (p->type==IN)
                       {termt[ntt].p.cx=in[p->en].p.cx;
                        termt[ntt].p.cy=in[p->en].p.cy;
                        termt[ntt].p.lx=in[p->en].p.lx;
                        termt[ntt].p.ly=in[p->en].p.ly;}
                else if (p->type==OUT)
                       {termt[ntt].p.cx=out[p->en].p.cx;
                        termt[ntt].p.cy=out[p->en].p.cy;
                        termt[ntt].p.lx=out[p->en].p.lx;
                        termt[ntt].p.ly=out[p->en].p.ly;};
                ntt++;
               }
  else if ((p->type==CMS) || (p->type==CMF))
               {
                boxt[nbt].type=p->type;
                boxt[nbt].en=p->en;
                boxt[nbt].ptrlb=NULL;
                boxt[nbt].ptrclos=NULL;
                if (p->type==CMF)
                       {boxt[nbt].p.cx= cmf[p->en].p.cx;
                        boxt[nbt].p.cy= cmf[p->en].p.cy;
                        boxt[nbt].p.lx= cmf[p->en].p.lx;
                        boxt[nbt].p.ly= cmf[p->en].p.ly;}
                else if (p->type==CMS)
                       {boxt[nbt].p.cx= cms[p->en].p.cx;
                        boxt[nbt].p.cy= cms[p->en].p.cy;
                        boxt[nbt].p.lx= cms[p->en].p.lx;
                        boxt[nbt].p.ly= cms[p->en].p.ly;}
```

```
                nbt++;
               };
  p=p->next;
 };
// endo of while (create table)
// fill termt table with term-box connections
for (int j=0; j<ntt; j++)
       for (int k=0; k<nbt; k++)
        if (cross(termt[j].p, boxt[k].p))
               { // ubacivanje novog elementa u listu
                LCON *n =new(LCON), *tp, *ts;
                n->en=k;
                n->next=NULL;
                tp=termt[j].ptrlb;
                if (tp)
                      {
                       while(tp) {ts=tp; tp=tp->next;};
                       ts->next=n;
                      }
                else
                      termt[j].ptrlb=n;
               }
// end of filling table with term-box connections
// fill boxt table with box-box connections
for(j=0; j<nbt; j++)
  for(int k=j+1; k<nbt; k++)
        if ((j!=k) && cross(boxt[j].p, boxt[k].p))
               {
                int flag=0;
                for(int l=0; l<ntt; l++)
                      if (cross(boxt[j].p, termt[l].p) &&
                                  cross(boxt[k].p, termt[l].p))
                                  { flag=1; break;}
                if (!flag) // box-box cut only
                      {
                       LCON *tp, *ts, *n=new(LCON);
                       n->en=k;
                       n->next=NULL;
                       tp=boxt[j].ptrlb;
                       if (!tp)
                              boxt[j].ptrlb=n;
                       else
                              {
                                    while(tp) {ts=tp; tp=tp->next;}
                                    ts->next=n;
                              }
                       n=new(LCON);
                       n->en=j;
                       n->next=NULL;
                       tp=boxt[k].ptrlb;
                       if (!tp)
                              boxt[k].ptrlb=n;
                       else
                              {
                                    while(tp) {ts=tp; tp=tp->next;}
                                    ts->next=n;
                              };
                       } // end if (flag)
               };
// end of filling boxt table with box-box connections
// closure computation
for(j=0; j<nbt; j++)
       {
        LCON *n=new(LCON), *tp, *closure;
        n->en=j;
        n->next=NULL;
        closure=n;
        tp=closure;
        while(tp)
              {
               LCON *tp1=boxt[tp->en].ptrlb;
               while (tp1)
                     {
                      LCON *ttp, *tts;
                      int alexist=0;
                      ttp=closure;
                      while(ttp)
                            {
                             if (ttp->en==tp1->en)
                                    {alexist=1; break;}
```

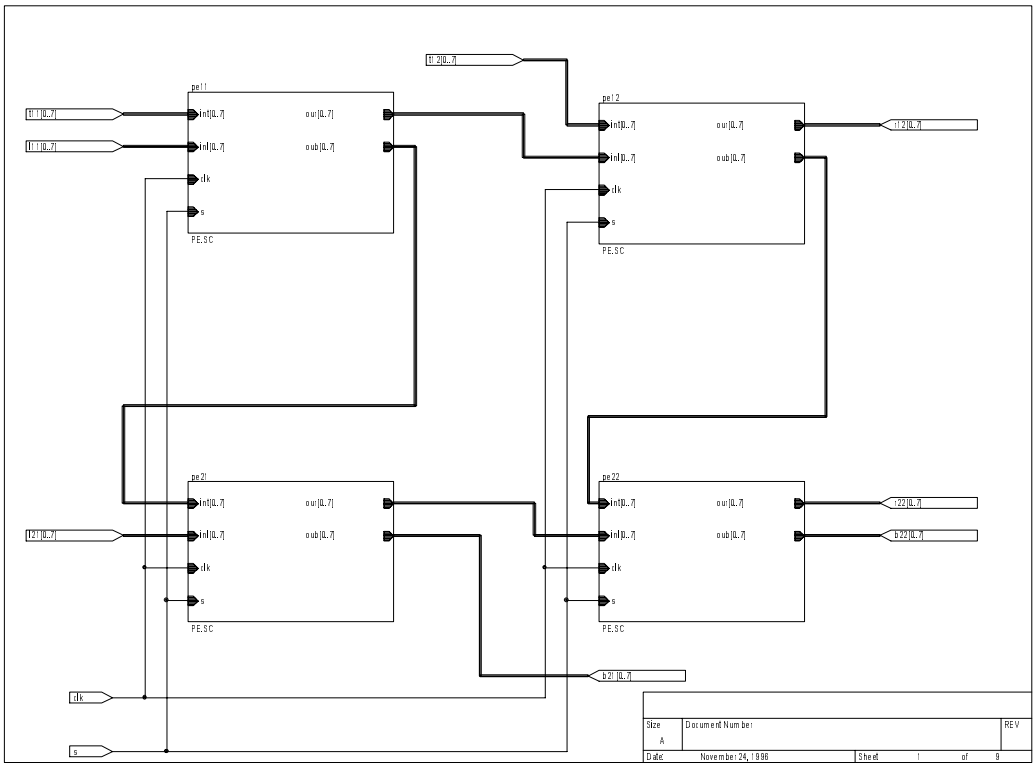
```
                                tts=ttp; ttp=ttp->next;
                               }
                         if (!alexist)
                               {
                                n=new(LCON);
                                n->en=tp1->en;
                                n->next=NULL;
                                tts->next=n;
                               }
                         tp1=tp1->next;
                        }
                  tp=tp->next;
                 }
          boxt[j].ptrclos=closure;
         }
// end of closure computation
// show box table
for(j=0; j<nbt; j++)
        {
         cout << "box[" << j << "] type" << boxt[j].type << "["
         << boxt[j].en << "]\n";
         LCON *lcp=boxt[j].ptrlb;
         while(lcp) {cout << " :b" << lcp->en; lcp=lcp->next;}
         cout << "\n";
         lcp=boxt[j].ptrclos;
         while(lcp) {cout << " :c" << lcp->en; lcp=lcp->next;}
         cout << "\n";
        }
//fill term table with connections
for(j=0; j<ntt; j++)
  {
        cout << "term[" << j << "]: type" << termt[j].type << "["
        << termt[j].en << "]" << "\n";
        for(int k=j+1; k<ntt; k++)
         {int u=ctwot(j,k);
          if (u)
                 lcput(j,k);
         }
        LCON *lcp=termt[j].ptrlt;
        cout << " listp: ";
        while(lcp) {cout << " :v" << lcp->en; lcp=lcp->next;}
        cout << "\n";
        lcp=termt[j].ptrlb;
        cout << "listb: ";
        while(lcp) {cout << " :b" << lcp->en; lcp=lcp->next;}
        cout << "\n";
  }
// end of fill table with connections
cout << "nIN= " << nsin << " nOUT= " << nsout << "\n";

// find end elements; it's depend from nsout and nsin
if ((!nsout) && (nsin))
        {//there are IN elements, only
         cout << "This set is out of scope!!\n";
         continue;
        }
else if (nsout)
        {
         for(j=0;j<ntt;j++)
                if (termt[j].type==OUT)
                break;
         if (j<ntt)
                end=j;
        };
// end of finding end element
cout << "end el: " << end << ":" << termt[end].type << " "
        << termt[end].en << "\n";

for(int k=0;k<nlset;k++)
        {
         // find begin element
         beg=-1;
         for(j=0; j<ntt; j++)
                if (nsin && (termt[j].type==IN) && !termt[j].stat)
                  {beg=j;
                        termt[j].stat=USED;
                        break;
                  }
                else if (!nsin && (termt[j].nct==1) && (termt[j].type==VIA))
                  {beg=j; break;}
```

```
if (beg==-1)
        {cout << "ne mogu da nadjem pocetni element\n"; break;}

nstack=0;
nrcstack=0;
for (int l=0; l<maxnrcstack; l++)
      {
       rcstack[l].rc1=-1;
       rcstack[l].rc2=-1;
      }
// begin with line creation
stack[nstack]=beg;
cout << "begin element= " << beg << "\n";
nstack++;
int succ=0, push=1, pop=0;
while(!succ)
      {
       if (stack[nstack-1]==end)
             {succ=1; cout << "gp:SUCCESS\n";}
       else if (push)
             {
              int f=putonstack();
              if (!f) {pop=1;push=0;}
             }
       else if (pop)
             {
              if (stack[nstack-1]==rcstack[nrcstack-1].node)
                    {// put
                     if (rcstack[nrcstack-1].rc1 != -1)
                           {
                            stack[nstack]=rcstack[nrcstack-1].rc1;
                            nstack++;
                            cout << "putonstack from reserve: " <<
                            stack[nstack-1] <<  "set=" << i << "\n";
                            rcstack[nrcstack-1].rc1=-1;
                            if (rcstack[nrcstack-1].rc2==-1)
                                  nrcstack--;
                           }
                     else if(rcstack[nrcstack-1].rc2 != -1)
                           {
                            stack[nstack]=rcstack[nrcstack-1].rc2;
                            nstack++;
                            cout << "putonstack from reserve: " <<
                            stack[nstack-1] << "\n";
                            rcstack[nrcstack-1].rc2=-1;
                            if (rcstack[nrcstack-1].rc1 == -1)
                                        nrcstack--;
                            else { cerr << "e1 err\n"; exit(1);}
                           }
                     else
                           {
                            cerr << "e2 err: \n"; exit(1);
                           };
                     push=1;
                     pop=0;
                    }
              else
                    {
                     cout << "pop form stack\n";
                     nstack--;
                     if (nstack < 0)
                            {cerr << "e4: err pop\n";
                             cout << "e4: err pop from empty stack\n" ;
                             exit(1);}
                    }
                  } //end pop

       }// end while
// show
 cout << "***line " << k<< "\n";
 for (l=0; l<nstack; l++)
       cout << " :v" << stack[l];
 cout << "\n";

 for(termlen=0, j=0; j<nstack-1; j++)
       termlen += abs(termt[stack[j+1]].p.cx - termt[stack[j]].p.cx)
                       + abs(termt[stack[j+1]].p.cy - termt[stack[j]].p.cy);
 termlen += rememb;

 cout << "Interterminal conn. length= " << termlen
```

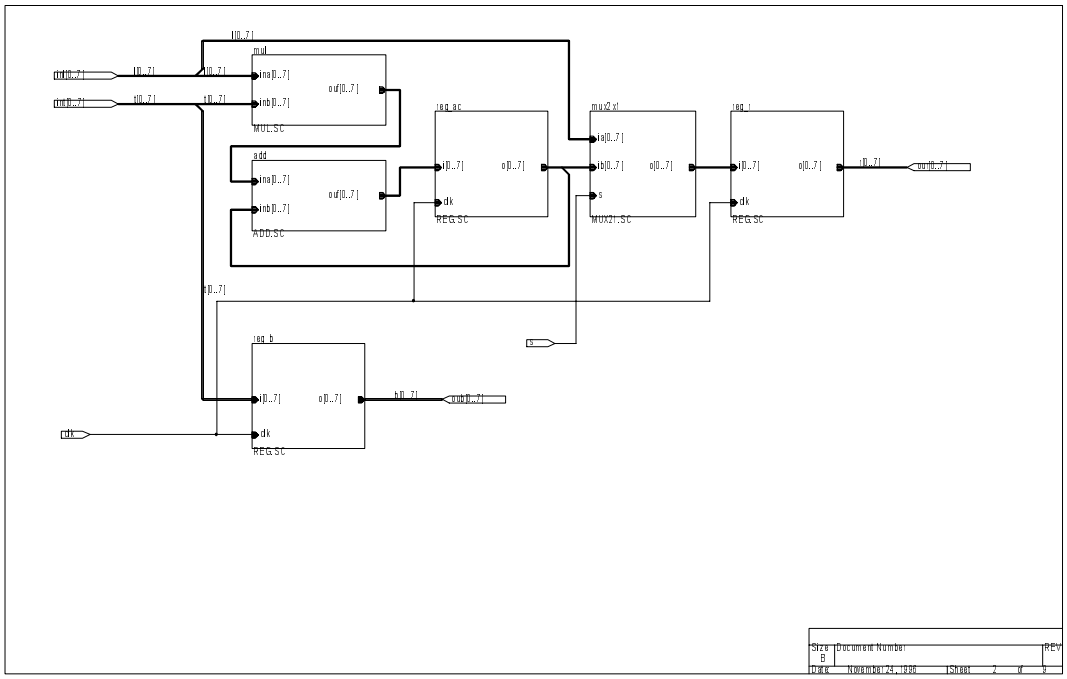
```
                    << "\tOut core conn. length= " << newoutclen << "\n";
                cout << "New outcore len= " << newoutclen << "\n";
                cout << "New incore length= " << (termlen + outclen - newoutclen)
                             << "\n";
                cout << "Overall conn length for this line= " <<
                              (termlen + outclen) << "\n";

                pom[0]='\0';
                strcat(pom, ltoa(i,pom1,10));
                strcat(pom, "\t");
                strcat(pom,ltoa(nlt, pom1,10));
                strcat(pom, "\t");
                strcat(pom, ltoa((termlen + outclen - newoutclen) , pom1,10));
                strcat(pom, "\t");
                strcat(pom, ltoa(newoutclen, pom1, 10));
                strcat(pom, "\t");
                strcat(pom, ltoa((termlen+outclen), pom1, 10));
                // distribution vector
                int index=(termlen+newoutclen)/1000;
                dvect[index]++;
                ofout.write(pom, strlen(pom));
                ofout.put('\n');
                nlt++;
                avtlen= avtlen*(double(nlt-1)/nlt) + (termlen + newoutclen)/nlt;
               }; // end for
  for(j=0;j<ntt;j++)
         {
               delete termt[j].ptrlt;
               delete termt[j].ptrlb;
         };
  for(j=0;j<nbt;j++)
         {
               delete boxt[j].ptrlb;
               delete boxt[j].ptrclos;
         }
 } // end for (i)
 pom[0]='\0';
 strcat(pom, "==================\n");
 ofout.write(pom, strlen(pom));
 pom[0]='\0';
 strcat(pom, "average total length: ");
 strcat(pom, ltoa(long(avtlen), pom1, 10));
 ofout.write(pom, strlen(pom));
 ofout.put('\n');
 pom[0]='\0';
 strcat(pom, "interconnection distribution");
 ofout.write(pom, strlen(pom));
 ofout.put('\n');
 // show distribution vector
 for (i=0; i<ndvect; i++)
        {
         if (!dvect[i]) break;
         cout << "dvect[" << i << "]=" << dvect[i]
                      << "\t%=" << double(dvect[i])/(double)nlt << "\n";
         pom[0]='\0';
         strcat(pom, ltoa(long(1000*(i+1)), pom1, 10));
         strcat(pom, "\t");
         strcat(pom, ltoa(dvect[i], pom1, 10));
         ofout.write(pom, strlen(pom)); ofout.put('\n');
        }
 ofout.close();
} // end of main
```

Apendiks #3:
Šeme iz OrCAD paketa

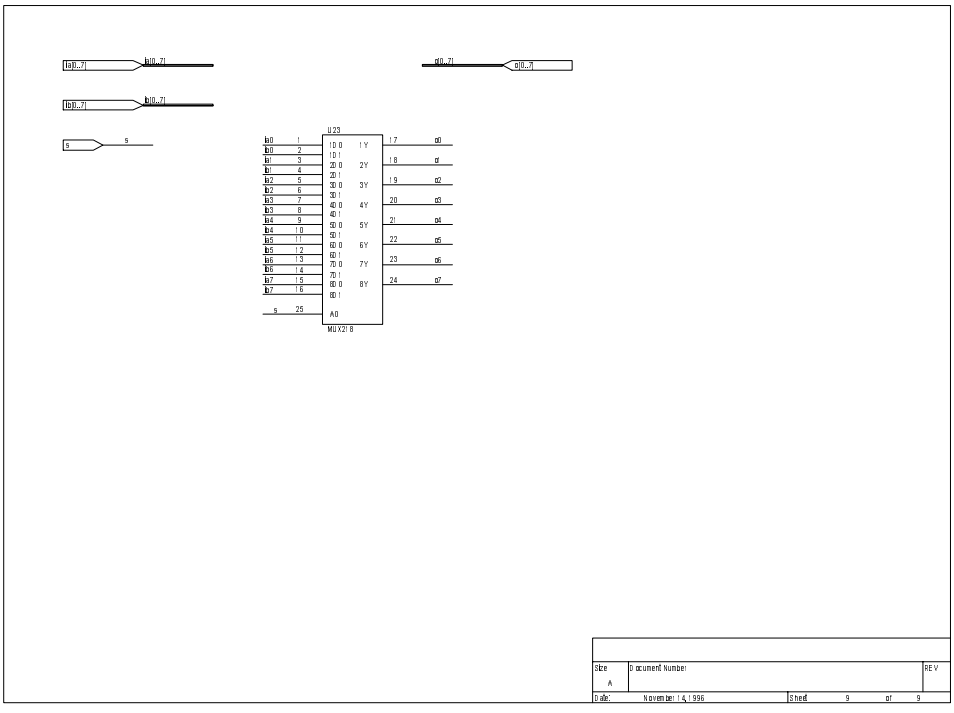
Ovaj apendiks uključuje skup hijerarhijskih šema formiranih programskim paketom OrCAD za 2D realizaciju sistoličkog polja koje se posmatra u eksperimentu 3. Glavna šema sistoličkog polja sa 4 osmobitna procesna elementa prikazana je na slici 52. Šema jednog procesnog elementa prikazana je na slici 53. Procesni element sadrži osmobitni multiplekser 2x1 koji je prikazan na slici 54, tri identična osmobitna registra čija je šema data na slici 55, osmobitni sabirač čija je šema data na slici 56, i množač 4x4 čija je šema data na slici 57. Realizovan je iterativni množač koji se sastoji od dela za generisanje parcijalnih proizvoda koji je prikazan na slici 58 i dela za sabiranje parcijalnih proizvoda koji je prikazan na slici 59. Šema potpunog sabirača prikazana je na slici 60.



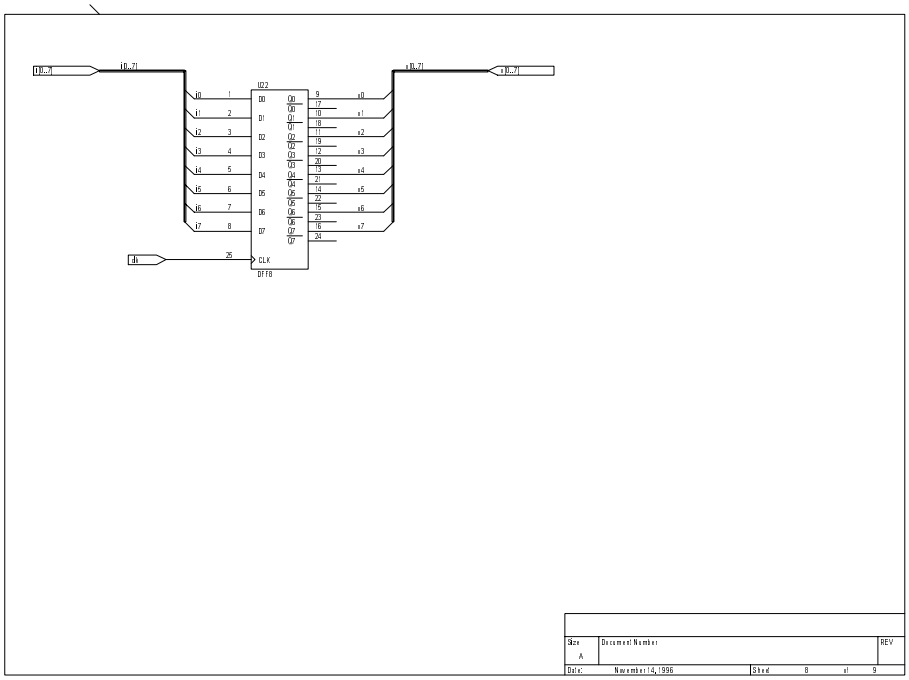
Slika 52. Šema sistoličkog polja sa četiri osmobitna procesna elementa.



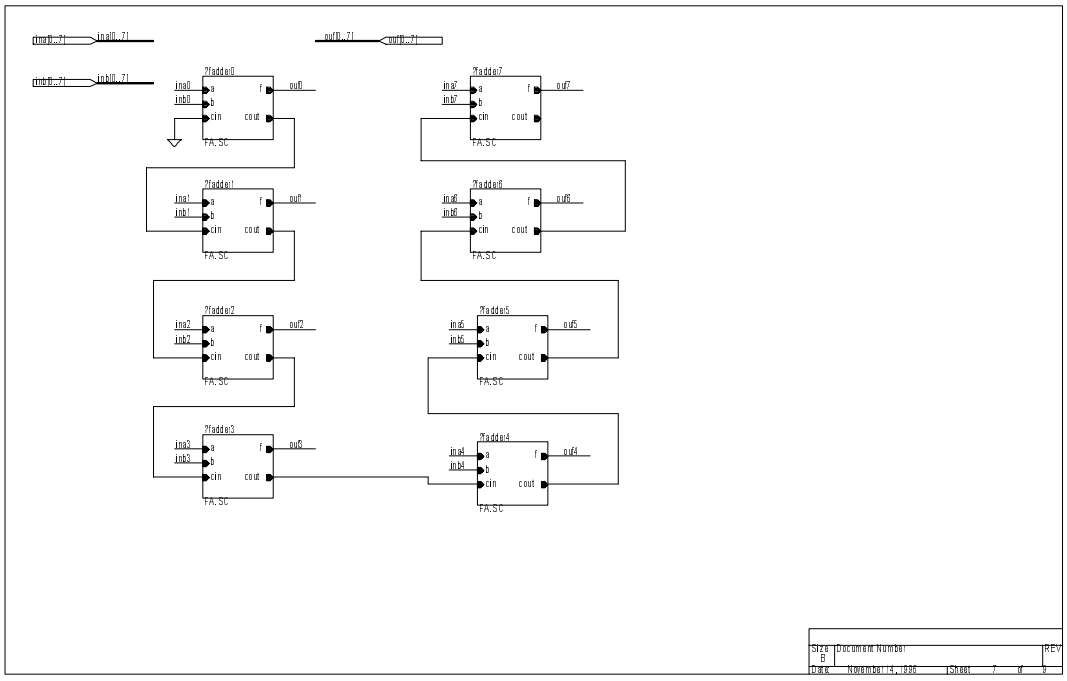
Slika 53. Šema procesnog elementa.



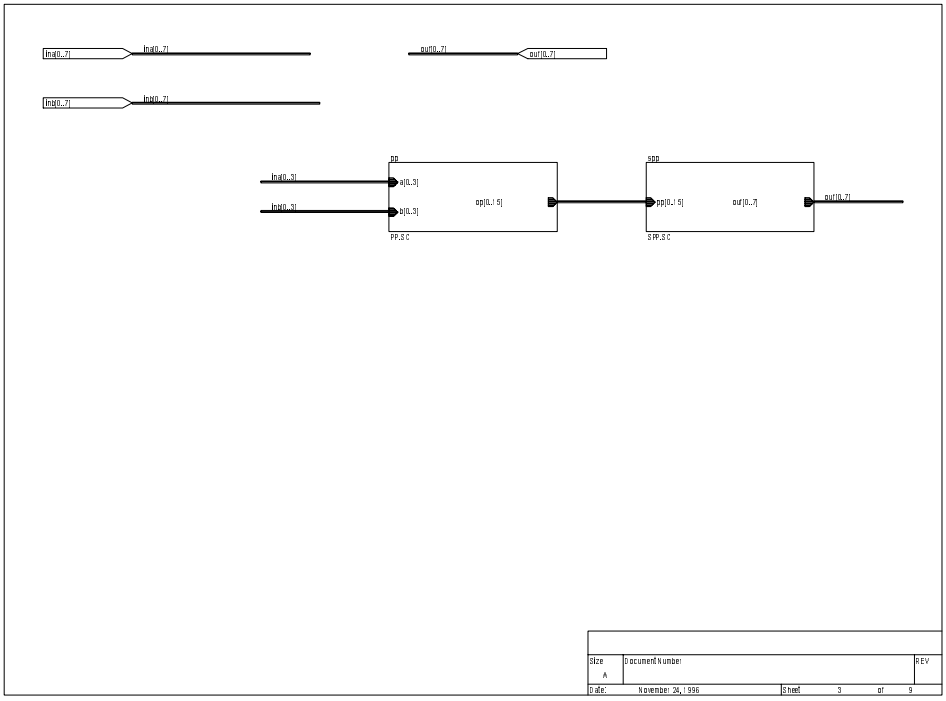
Slika 54. Osmobitni multiplekser 2x1.



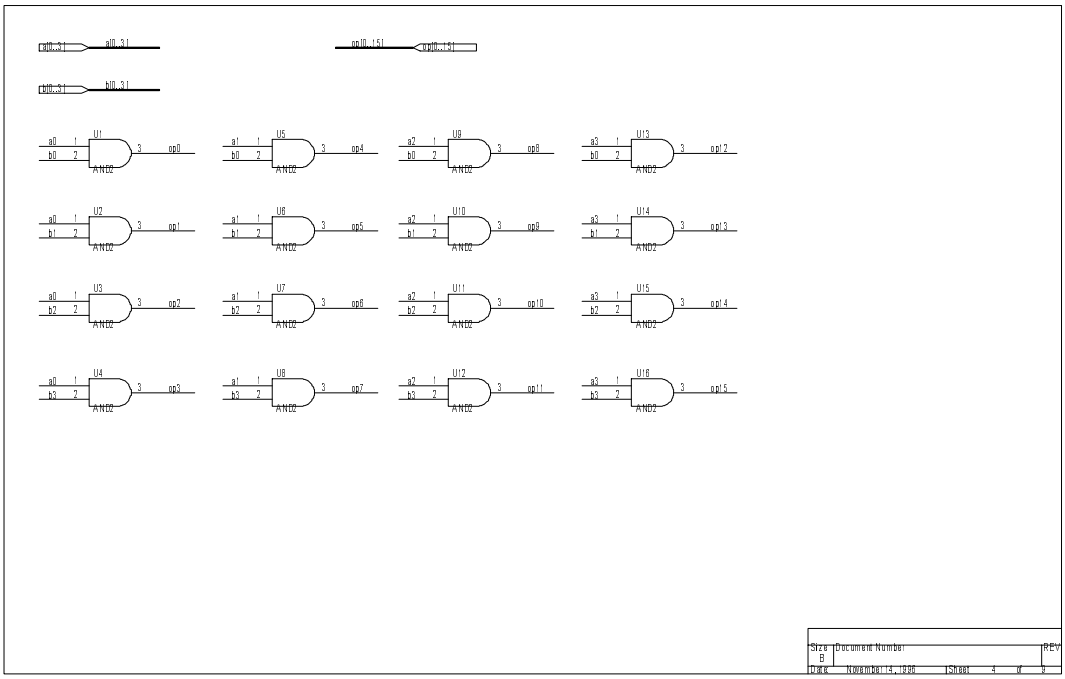
Slika 55. Osmobitni registar.



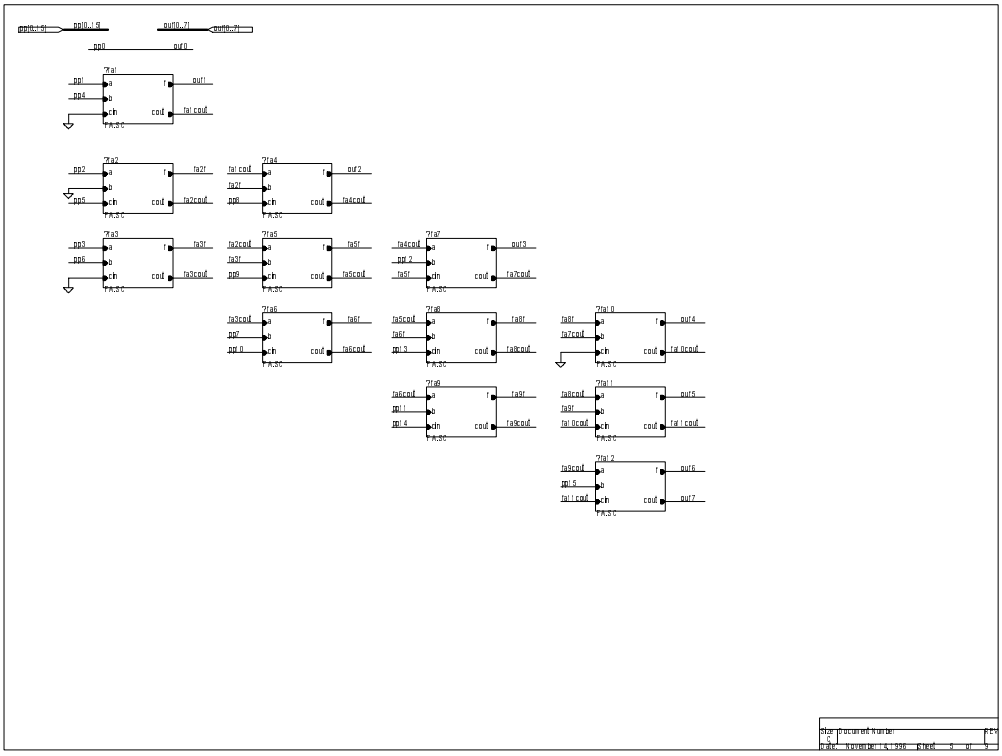
Slika 56. Šema osmobitnog *ripple-carry* sabirača.



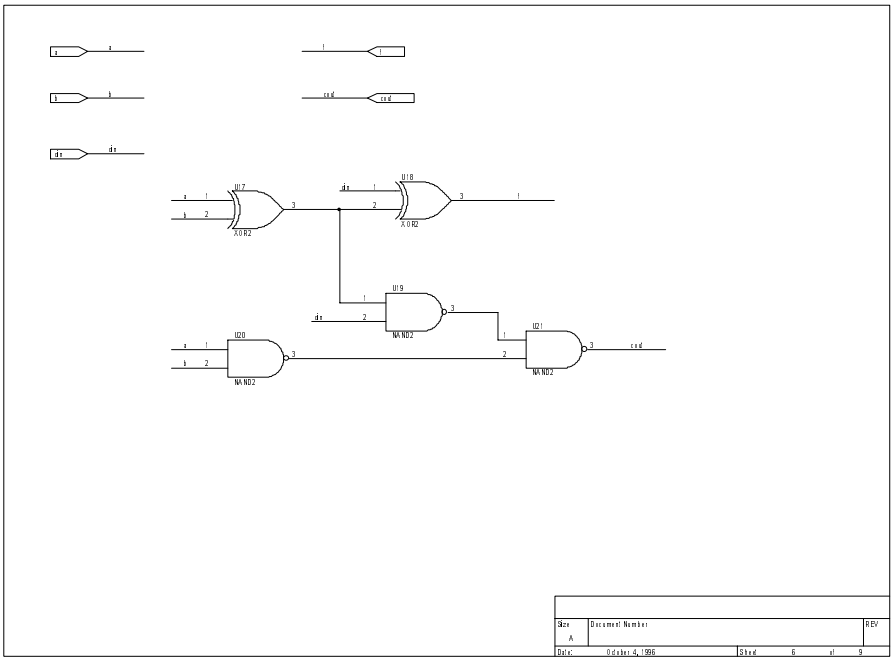
Slika 57. Blok šema iterativnog množača.



Slika 58. Generator parcijalnih proizvoda.



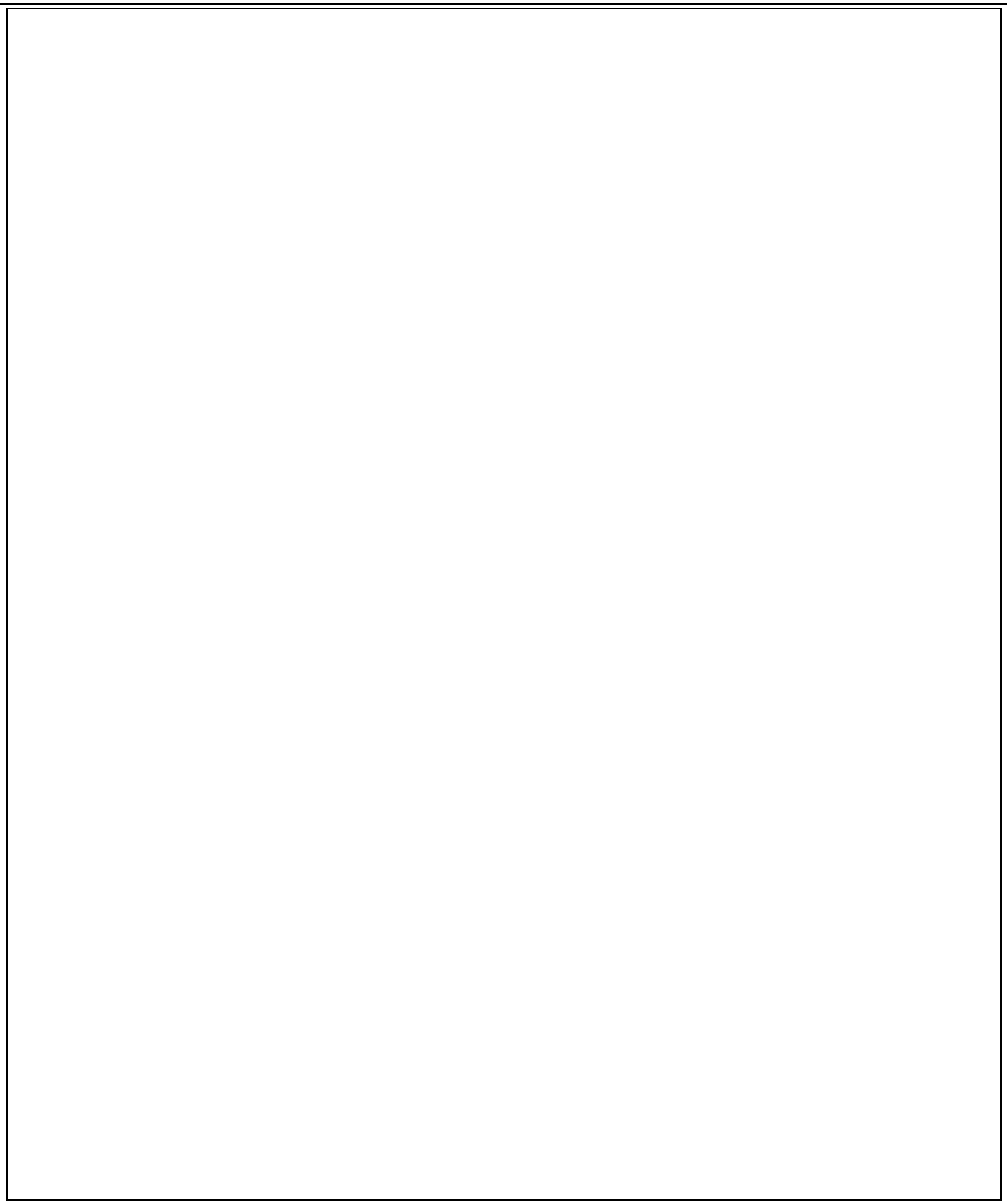
Slika 59. Šema sabirača parcijalnih proizvoda.



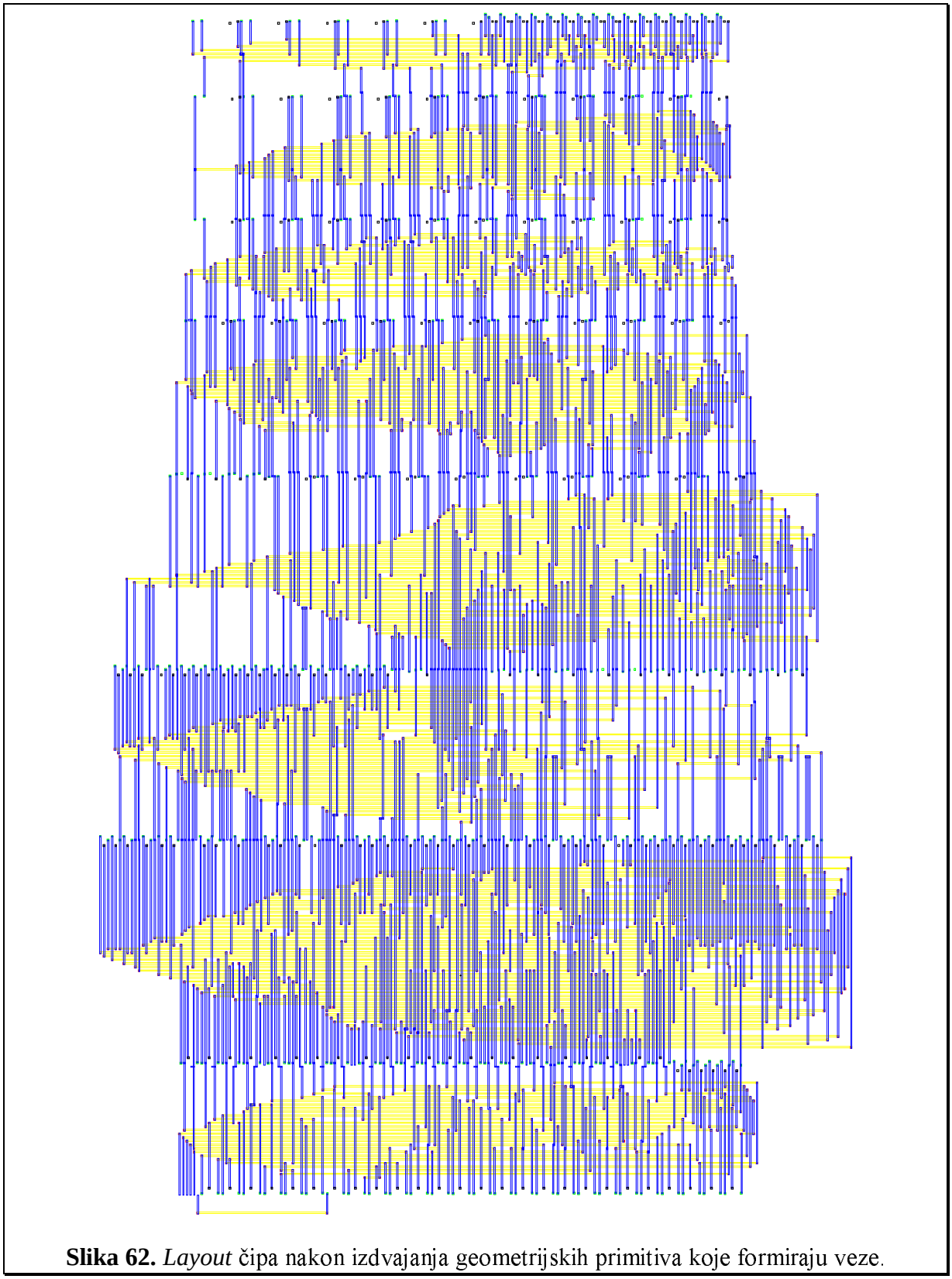
Slika 60. Potpun sabirač.

Apendiks #4:
Šeme iz L-Edit paketa

U ovom apendiksu na slici 61 prikazan je *layout* jezgra čipa sistoličkog polja za množenje matrica sa četiri četvorobitna procesna elementa. Na slici 62 prikazan je *layout* istog čipa nakon izvršene obrade sa MARS paketom koja ima za cilj da izdvoji samo one geometrijske primitive koje formiraju veze između terminala standardnih ćelija.



Slika 61. *Layout* 2D realizacije sistoličkog polja sa četiri 4-bitna procesna elementa.



Slika 62. *Layout* čipa nakon izdvajanja geometrijskih primitiva koje formiraju veze.