1 Uvod

U novije vreme uložen je veliki napor ka povećanju brzine rada digitalnih sistema, tako da danas imamo veoma moćne uređaje koji se odlikuju grafikom visoke rezolucije i zahtevnim multimedijalnim sposobnostima. Tako je veoma brzo izračunavanje postalo standard za rad sistema namenjenih prosečnim korisnicima. Pored toga, današnji korisnik ima potrebu da ostvaruje izračunavaja sa bilo koje lokacije. Zahtevi za mobilnošću korisnika postavljaju ograničenja u pogledu veličine, težine uređaja i potrošnje energije. Iako se tehnologija proizvodnje baterija stalno poboljšava, neposredno se nameće veliki problem koji se odnosi na smanjenje potrošnje energije ovih uređaja.

Na početku, prvenstveno zbog ne tako strogih zahteva za procesiranjem, mobilne aplikacije su se odlikovale malom potrošnjom i malom propusnošću. Međutim, današnje aplikacije su veoma zahtevne po pitanju brzine rada, propusnosti i naravno mikro potrošnji. Ovo najbolje ilustruje primer *notebook* računara koji zahtevaju sposobnost izračunavanja istovetnu kao i kod *desktop* mašine, uz daleko manju potrošnju. Podjednako je zahtevan i razvoj komunikacionih sistema kao što su mreže digitalne mobilne telefonije (*3G*), koje zahtevaju brzo izvršenje kompleksnih algoritama za kompresiju signala slike, govora i mnogobrojnih drugih aplikacija. Takođe su značajna i korišćenja raznih bežičnih veza preko kojih se prenose različiti tipovi informacija (*Bluetooth, Wireless i dr.).* Ovo uslovljava postojanje inteligentnih mreža koje omogućavaju komunikaciju sa servisima ili sa ljudima koji se nalaze na bilo kom mestu i u bilo koje vreme. Iz ovoga je očigledna potreba da se fiksne radne stanice zamene prenosivim koje će odlikovati manja potrošnja [1].

Čak i u neprenosivim aplikacijama, veoma je značajan problem male potrošnje. Ranije ovaj problem nije bio toliko izražen, s obzirom da su korišćena velika kućišta i

disipirana energija je odvođena različitim vrstama hladnjaka. Sa povećanjem gustine i veličine čipova nastaju poteškoće u obezbeđivanju adekvatnog hlađenja i mogu značajno povećati cenu sistema ili ograničiti njegovu funkcionalnost.

U drugom poglavlju su prikazane vrste disipacije snage, mesto i uslovi njihovog nastanka. Pored toga prikazano je i koliki je udeo svake od pomenutih disipacija u ukupnoj disipaciji.

U trećem poglavlju su analizirane najvažnije strategije za smanjenje potrošnje snage. To su tehnike za prilagođenje procesa tehnologije, tehnike redukcije dinamičke potrošnje, *Power down* tehnike i tehnike smanjenja snage usled struja curenja. Najvažnija tehnika redukcije kako statičke tako i dinamičke potrošnje je tehnika skaliranja napona napajanja. Pored ove tehnike opisan je i postupak izbora optimalnog napona napajanja. U sekciji 3.2 su date tehnike za redukciju dinamičke potrošnje. Razrađena je i metoda redukcije potrošnje korišćenjem *power down* tehnika, kao što su: ukidanje napona napajanja, dozvola/zabrana taktovanja, korišćenje flip-flop-ova sa signalom dozvole kao i particionisanje memorije. Od tehnika za smanjenje snage usled struja curenja su prikazane tehnike multipragovskog dizajniranja i promenjivog praga napona provođenja.

Četvrto poglavlje razrađuje metode projektovanja za malu potrošnju i načine formiranja *VHDL* kodova kola za *low power*. Prikazane su i osnove o modelovanju komponenti za malu potrošnju, gde je dato nekoliko primera modelovanja. Na kraju su objašnjeni tokovi analize potrošnje.

Peto poglavlje sadrži konkretno rešenje projektovanja standardne i modifikovane *ALU* jedinice. Modifikacija se odnosi na preuređenje standardnog *ALU*-a radi manje potrošnje.

Na kraju rada se nalazi dodatak, koji sadrži *VHDL* kodove predhodno pomenutih *ALU* jedinica i kompletni *Reporti* nakon obavljenih koraka sinteze, implementacije i analize potrošnje.

Ovaj rad sadrži i softverski deo dat na *CD*-u na kome se nalaze projekti standardnog (*Alu_base*) i modifikovanog (*Alu_lowpower*) *ALU*-a.

2 Izvori disipacije snage

Ukupna potrošnja *CMOS* kola, P_{total}, data je sledećom relacijom:

$$P_{total} = P_{dynamic} + P_{static} \quad (1)$$

gde je: $P_{dynamic}$, dinamička disipacija snage; P_{static}, statička disipacija snage.

Dinamička disipacija snage, $P_{dynamic}$, se izražava kao:

$$P_{dynamic} = P_{cap} + P_{scc} \quad (2)$$

Gde je P_{cap}, komutirana snaga i P_{scc}, snaga kratkog spoja.

Pri čemu važi:

$$P_{cap} = a_{o\to1} f_{CLK} \int_0^T i_{V_{DD}}(t) V_{DD} dt = a_{0\to1} f_{CLK} C_l V_{DD}^2 \quad (3)$$

Komutirana snaga, P_{cap}, srazmerna je taktnoj frekvenciji f_{CLK}, $\alpha_{0\to1}$- faktoru prelaza sa 0→1 u toku jednog taktnog perioda, C_l- komutiranoj kapacitivnosti, i kvadratu napona napajanja V_{DD}^2. Sledeća jednačina predstavlja snagu disipacije gejta kada su oba tranzistora provodna.

$$P_{scc} = a_{0\to1} f_{CLK} I_{sc} (\frac{t_r + t_f}{2}) V_{DD} \quad (4)$$

To znači da je P_{scc} posledica "kratkog spoja između napona napajanja i mase " koji se javlja kada su oba, *PMOS* i *NMOS,* tranzistora simultano aktivni (provodni). Na osnovu (4) se vidi da je P_{scc} direktno srazmerna struji kratkog spoja I_{sc}. Struja, I_{sc}, protiče od napajanja ka masi kada oba tranzistora istovremeno vode. Disipirana snaga,

P_{scc} je mala i učestvuje u ukupnoj potrošnji (u relaciji (1)) sa oko 20% [2]. Snaga disipacije usled kratkog spoja može biti kontrolisana od strane vremena tranzicije u kolu (t_r - vreme rastuće ivice i t_f - vreme opadajuće ivice signala).

Poslednja komponenta ukupne disipacije snage, definisana u (1), odgovara statičkoj disipaciji, a postoji zbog konačne vrednosti struje curenja i oblika je:

$$P_{static} = I_{leakage} V_{DD} \tag{5}$$

Kao što se vidi iz (5) statička disipacija P_{static} direktno zavisi od struje curenja tranzistora, $I_{leakage}$, i napona napajanja V_{DD}. Struja curenja $I_{leakage}$, zavisna je od tehnologije izrade kola.

Na osnovu predhodnih razmatranja totalna disipirana snaga iznosi:

$$P_{total} = a_{0\to 1} f_{CLK} C_l V_{DD}^2 + a_{0\to 1} f_{CLK} I_{cs} (\frac{t_r + t_f}{2}) V_{DD} + I_{leakage} V_{DD} \tag{6}$$

U cilju efikasnijeg sagledavanja ove problematike uvešćemo proizvod-energija-kašnjenje koji se može interpretirati kao iznos potrošene energije pri svakom prelazu. Ovaj proizvod je posebno koristan kod upoređivanja različitih stilova realizacije kola sa tačke gledišta disipacije. Pri ovome se usvaja da je kod disipacije snage važna samo prva komponenta, pa shodno tome

$$energija\_po\_prelazu = \frac{P_{total}}{f_{CLK}} = C_{efektivno} V_{DD}^2 \tag{7}$$

gde je: $C_{efektivno} = p_t C_l$

3 Strategije za smanjenje potrošnje

Na osnovu predhodnog razmatranja, zaključili smo da ukupnu disipaciju čine dinamička ($P_{dynamic}$), tzv. *switching power*, i statička komponenta snage (P_{static}). Shodno tome, napori projektanata, koji se odnose na redukciju snage, trebaju biti usmereni ka redukciji:

- dinamičke potrošnje, i
- snage usled struje curenja.

Treba istaći da za tehnologije čije su širine kanala veće od 90 nm, dominantan uticaj ima prvi član. Šta više 75% od ukupne potrošnje čini dinamička potrošnja [3]. To znači da ako želimo da smanjimo potrošnju, a da pri tome zadržimo neophodnu funkciju kola, treba da minimizujemo $\alpha_{0\to1}$, C_l, V_{DD} ili f_{CLK}. S obzirom da je dinamička potrošnja proporcionalna sa kvadratom V_{DD}, a statička proporcionalna sa V_{DD}, evidentno je da smanjenje V_{DD}-a vodi ka najefikasnijoj redukciji potrošnje. Smanjenje (skaliranje) napona napajanja, takođe prati i skaliranje *threshold voltage* (V_{th}) tranzistora, odnosno napona praga provođenja. Na žalost, skaliranjem se povećava struja curenja, pa sada statička disipacija ima dominantni uticaj kako na potrošnju tako i na performanse dizajna.

Primer

Kada se koristi 0,13 μm tehnologija sa V_{th}=0,7 V, struja curenja po tranzistoru je u osegu od 10-20 pA. Neka u cilju smanjenja potrošnje redukujemo V_{th} na 0,2-0,3 V. Struja curenja će se povećati na 10-20 nA. To znači, da ako u *VLSI* kolu imamo ugrađeno 10 M tranzistora, tada će statička potrošnja biti reda 150 mW.

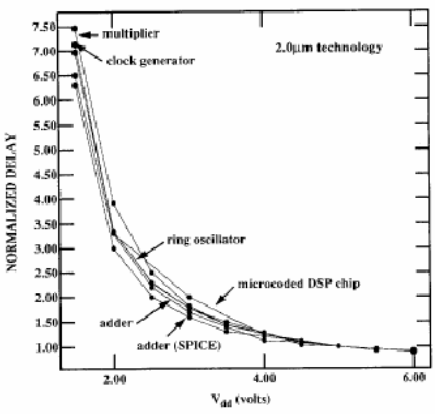
U zavisnosti od tipa digitalnih kola, sinhroni ili asinhroni, koriste se različite strategije za redukciju potrošnje. Logika koja se bazira na sinhronim kolima koristi registre (lečeve) koji se ubacuju između stepena koji vrše izračunavanje. Stepeni se realizuju kao kombinacione mreže, a pamćenje informacije u lečevima vrši se nailaskom svakog taktnog impulsa. Da bi se kod logike zasnovane na sinhronom dizajnu smanjila potrošnja, neophodno je minimizirati komutatorske aktivnosti. Redukcija potrošnje se izvodi tako što se komutatorske aktivnosti logike ne obavljaju, kada kolo ne izvršava željenu funkciju. Ovo je važan aspekt jer logički moduli mogu komutirati i trošiti energiju čak i u slučaju kada se aktivno ne koriste. To znači da dizajn sinhronih kola treba da se bazira na specijalnim kolima kao i rešenjima kojima se detektuje neaktivnost nekog logičkog modula i preuzimanja odgovarajućih akcija u cilju smanjenja potrošnje kakve su tehnike: *power-down, clock gating* itd.

Kod asinhronih digitalnih kola situacija je nešto drugačija, jer se njihov princip rada bazira na konceptu *power-down* kada ta kola nisu aktivna. Raspodela disipirane energije u vremenu kod asinhronih kola, u odnosu na sinhrona, je ravnomernija.

3.1 Prilagođenje procesa tehnologije

3.1.1 Tehnološki pristupi skaliranju napona napajanja

Ključnu stvar za smanjenje potrošnje, kako statičke tako i dinamičke, predstavlja smanjenje napona napajanja kola. Iz relacije (7) vidi se da je energija po prelazu direktno proporcionalna kvadratu napona napajanja. Iz ovoga je evidentno da je potrebno redukovati napon napajanja.



Slika 1• *Kašnjenje u funkciji napona napajanja za različita kola*

Nažalost, predloženo rešenje za *low-power* dizajn koje se zasniva na redukciji napona napajanja stvara nove probleme. Naime, smanjenjem napona napajanja povećava se propagaciono kašnjenje signala kroz gejtove. Na slici 1 prikazan je ovaj efekat. Za sva kola (videti Tabelu 1), smanjenjem napona napajanja, dobija se ista zavisnost normalizovanog kašnjenja u funkciji napona napajanja.

Tabela 1• *Tehnološki detalji komponenata sa slike 1*

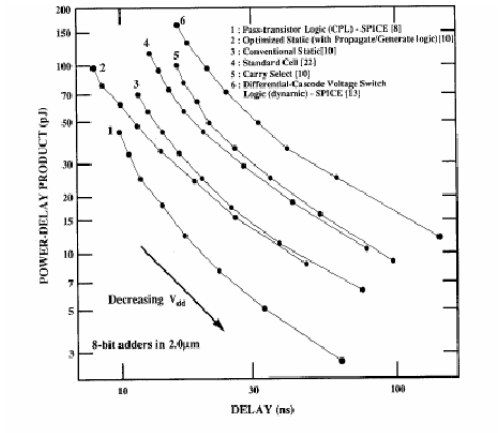
Komponente (2μm tehnologija)	Broj tranzistora	Površina (mm^2)	Komponente
DSP čip	44 802	94	20-bitna staza podataka
Množač	20 432	12.2	24 x 24 bit
Sabirač	256	0.083	Kombinaciona mreža
Ring oscilator	102	0.055	51 stepen
Takt generator	56	0.04	Unakrsno spregnuta NOR kola

Na osnovu slike 1 vidi se da se smanjenjem V_{DD}-a smanjuje brzina rada kola. Kašnjenje se drastično povećava kako se V_{DD} približava naponu praga provođenja komponenata. Tačna analiza kašnjenja je prilično složena. Ako se uzmu u obzir nelinearne karakteristike *CMOS* kola, pokazuje se da izvod prvog reda, na adekvatan način, aproksimira eksperimentalno određenu zavisnost kašnjena definisanu sa:

$$T_d = \frac{C_l V_{DD}}{I} = \frac{C_l V_{DD}}{mC_{ox}(W/L)(V_{DD} - V_{th})^2} \tag{8}$$

Gde su: T_d - kašnjenje, *W/L*- odnos širine i dužine kanala, V_{th}- napon praga provođenja, C_{ox}- kapacitivnost oksida, V_{DD}- napon napajanja, C_l- ukupna komutirana kapacitivnost i μ- pokretljivost nosilaca.

Karakteristike proizvod-snaga-kašnjenje za nekoliko različitih kola i topologija, određene eksperimentalnim merenjima i *SPICE* simulacijama, prikazane su na slici 2.



Slika 2• *Proizvod snaga-kašnjenje u funkciji brzine rada za različite tipove kola*

Uočimo da proizvod-snaga-kašnjenje opada sa povećanjem kašnjenjenja (tj. smanjenjem napona napajanja). Kod dobrog dizajna je potrebno da se nađe kompromis između kašnjenja i proizvoda-snaga-kašnjenje, čime se postiže dobra propusnost uz malu potrošnju [4].

Analizom slike 1 i slike 2 uočava se da zavisnost kašnjenja i potrošnja energije u funkciji skaliranja V_{DD}-a, za datu tehnologiju, predstavlja "prihvatljivo" rešenje jer je relativno nezavisno od logičkog stila i kompleksnosti kola. Dobijeni rezultati su od koristi za optimizaciju arhitektura male potrošnje, kod kojih je V_{DD} slobodna promenjiva. Kod ovakvih rešenja dozvoljava se promena arhitekture (uvođenjem paralelizma ili protočnosti) kako bi se zadržala ili povećala propusnost sistema.

3.1.1.1 Optimalna veličina tranzistora sa skaliranjem napona

Nezavisno od izbora logičke familije i topologije, određivanje optimalne veličine dimenzija tranzistora igra bitnu ulogu u redukovanju potrošnje. Osnovni problem se sada sastoji u određivanju optimalnog odnosa širine/dužine kanala (*W/L*) *CMOS*

tranzistora za datu tehnologiju. Više detalja o ovoj problematici dato je u [4]. Ilustracije radi za 2 μm tehnologiju izabira se napon napajanja od 3,3 V, što predstavlja uštedu energije od 60% u odnosu na slučaj kada kada je V_{DD} =5 V.

3.1.1.2 Izbor optimalnog napona napajanja

Povećanje kašnjenja usled smanjenja napona napajanja može biti kompenzovano korišćenjem paralelnih arhitektura. Na slici 1 se vidi, kako se napon napajanja približava naponima praga komponenata, kašnjenja kroz logička kola se naglo povećavaju. Kod smanjenja napona napajanja, stepen paralelizma, zbog dodatno ugrađene logike, može da se poveća do tačke u kojoj obim ugrađenog hardvera ima veći efekat od povećanja potrošnje. To ukazuje da postoji optimalan napon napajanja sa stanovišta arhitekture. Za određivanje vrednosti ovog napona, koristi se sledeći model potrošnje koji važi za fiksnu propusnost sistema u funkciji napona:

$$Power(N) = NC_{ref}V^2\frac{f_{ref}}{N} + C_{ip}V^2\frac{f_{ref}}{N} + C_{\text{int}erface}V^2 f_{ref} \qquad (13)$$

gde je N- broj paralelnih procesora, C_{ref}- ukupna komutirana kapacitivnost jednog procesora, C_{ip}- kapacitivnost zbog postojanja međuprocesorske komunikacije a posledica je paralelizma u radu sistema, i $C_{interface}$ je dodatna kapacitivnost sprege.

U opštem slučaju, C_{ip} i $C_{interface}$ su funkcije od N, pa normalizovana potrošnja u odnosu na referentni slučaj može da se iskaže kao:

$$P_{norm} = \left(1 + \frac{C_{ip}(N)}{NC_{ref}} + \frac{C_{\text{int}erface}(N)}{C_{ref}}\right)\left(\frac{V}{V_{ref}}\right)^2 \qquad (14)$$

Pri veoma niskim naponima napajanja (bliskim naponima praga), broj procesora raste brže nego što se član $(V/V_{ref})^2$ smanjuje, a to rezultuje povećanjem potrošnje sa daljim smanjenjem napona napajanja. Komponente sa manjim naponima praga imaju tendenciju da smanje optimalni napon, ipak pri naponima praga ispod 0.2 V disipacija snage usled subpragovske struje počinje da dominira i ograničava dalje poboljšanje potrošnje.

Još jedno ograničenje koje se odnosi na izbor najnižeg dozvoljenog napona napajanja predstavlja margina šuma V_{noise}. Zbog ovoga je potrebno ograničiti optimalni napon sa donje strane, pa je:

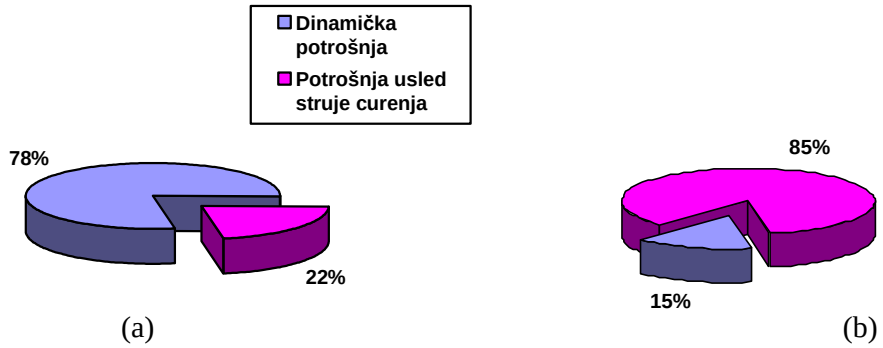
$$V_{noise} \leq V_{optimal} \leq V_{critical} \quad (15)$$

Napon napajanja će se, za određenu tehnologiju, nalaziti negde između napona koga diktira margina šuma, sa jedne, i kritičnog napona, sa druge strane. Povećanje površine čipa u direktnoj je vezi sa stepenom paralelizma u kolu. Ispitivanja su pokazala [4], da je optimalni napon napajanja relativno nezavistan od stepena paralelizma i za 0.2 μm tehnologiju nalazi se oko 1.5 V. Slična analiza koja je koristila napon praga od 0.5 V za 0.8 μm proces, rezultovala je u optimalnom naponu od oko 1 V, uz smanjenje potrošnje za faktor 10. Dalje smanjenje napona praga omogućava rad i pri nižim naponima napajanja, a samim tim i veće uštede energije.

3.2 Smanjenje dinamičke potrošnje

Dinamička potrošnja je uslovljena komutiranjem *CMOS* gejtova. Metal-oksid-poluprovodni (*MOS*) tranzistori ne komutiraju trenutno a i signali nemaju nulto vreme prelaza. Ovo uzrokuje dodatnu potrošnju, ponekad nazvanu "disipacija snage usled kratkog spoja" (P_{scc}). Ova potrošnja smanjuje se redukcijom prekidačkih aktivnosti $\alpha_{0\rightarrow 1}$ i redukcijom izlazne kapacitivnosti C_l.

Slika 3(a) prikazuje odnos dinamičke potrošnje i snage curenja u dizajnu kada u kolu nije izvršena optimizacija potrošnje. Nakon sprovedene optimizacije potrošnje odnos se menja i sada dinamička potrošnja predstavlja svega 15% od ukupne potrošnje (slika 3(b)) [5].



Slika 3• *Dinamička potrošnja: (a) Pre optimizacije, (b) nakon izvršene optimizacije*

Veći deo dinamičke snage se troši na punjenje i pražnjenje unutrašnjih i spoljašnjih kapacitivnosti. Ako je *I/O* opterećenje kola veliko, tada je dinamička struja *I/O* podsistema takođe velika i ima veliki uticaj na ukupnu potrošnju. Smanjenje disipirane snage u samom integrisanom kolu se standardno postiže: (a) smanjenjem frekvencije rada prekidačke logike; (b) redukovanjem obima hardvera prekidačke logike; (c) redukovanjem vremena trajanja prekidačkih aktivnosti u okviru kola; i (d) smanjenjem parazitnih kapacitivnosti kola.

U metode za redukciju dinamičke potrošnje spadaju: (i) Redukcija prekidačkih aktivnosti kola; (ii) minimizacija gličeva; (iii) redukcija izlaznih kapacitivnosti; (iv) korektna sinteza taktnog stabla. Opis odgovarajućih metoda i načini realizacije dati su u sledećim poglavljima.

3.2.1 Redukcija kapacitivnosti

Komutirana izlazna kapacitivnost se sastoji od tri člana:

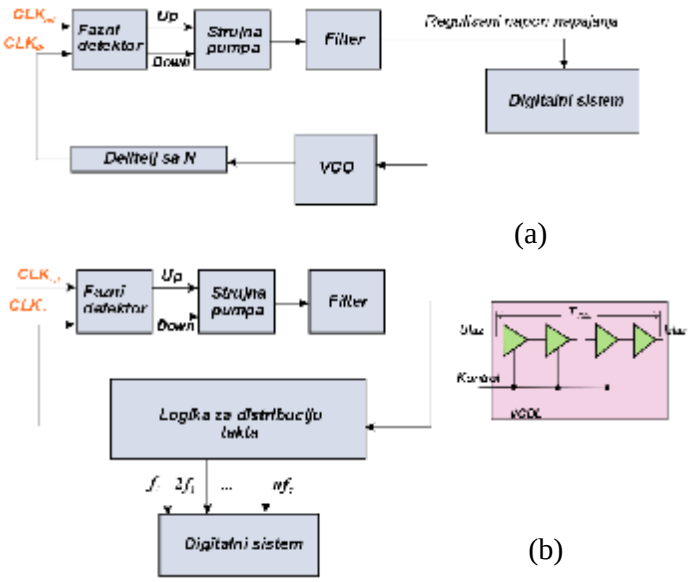
$$C_l = C_{fo} + C_w + C_p \qquad (16)$$

Gde je*:* C_{fo}- ulazna kapacitivnost gejtova koji opterećuju (*fanout*) kolo; C_w- kapacitivnost veza; i C_p- predstavlja izlaznu parazitnu kapacitivnost kola. U submikronskoj tehnologiji C_w ima najveći uticaj, ali nažalost njenu vrednost je najteže proceniti. Pri ovome se mora se uzeti u obzir uticaj preslušavanja (*cross talk*) [2]. Ovu vrednost projektanti sistema nemogu direktno da odrede, jer se njena tačna vrednost

zna nakon što je *layout* kola gotov. Drugim rečima ova kapacitivnost zavisi od fizičkog rasporeda gejtova i dužine veza u samom kolu.

3.2.2 Redukcija taktne frekvencije

Iz jednačina (3) i (4) vidimo da je disipacija snage direktno srazmerna taktnoj frekvenciji f_{CLK}. Ovo ukazuje na važnost redukcije taktne frekvencije. Najpoznatije tehnike redukcije taktne frekvencije su tehnike višestruke frekvencije zasnovane na: (a) *PLL*-u, i (b) na liniji za kašnjenje (*DLL*). Ove tehnike se standardno koriste u *VLSI* projektovanju integrisanih kola.



Slika 4• *Tehnike višestruke frekvencije zasnovane na: (a) PLL-u, (b) DLL-u*

Kod tehnike bazirane na *PLL*-u indirektno se kontroliše napon napajanja sistema. Tehnika bazirana na *DLL*-u sadrži liniju za kašnjenje koju kontroliše napon na izlazu iz filtera. Linija za kašnjenje na svom izlazu generiše više signala različitih frekvencija, koji se koriste u sistemu.

3.3 *Power down* metode za smanjenje potrošnje

U sinhronim kolima, u toku svakog taktnog impulsa, kombinaciona logika koja je locirana između dva registra obavlja specificiranu funkciju nad ulaznim vrednostima. U najvećem broju slučajeva ovako projektovan sistem ne radi maksimalno efikasno, jer su delovi čipa neaktivni (ne obavljaju korisnu funkciju), ali i dalje troše snagu. Drugim rečima dodatna potrošnja postoji zbog nepotrebne promene stanja signala na ulazima nekorišćenih delova kola kao i opterećenje koje ova kola unose u sistemu za distribuciju taktnog signala.

Da bi se smanjila potrošnja, neophodno je minimizirati komutacione aktivnosti onih delova logike koji u datom trenutku ne obavljaju korisnu funkciju, tj. isključivanje (*power down*) izvršnih jedinica koje ne obavljaju korisne operacije. Poznate tehnike za *power down* su:

- Ukidanje napona napajanja,
- Gejtovanje takta,
- Korišćenje flip-flop-ova sa dozvolom i
- Particionisanje memorije.

U tekstu koji sledi biće dat kratak opis svake od ovih tehnika.

3.3.1 Ukidanje napona napajanja

Ukidanje napona napajanja redukuje potrošnju snage na nulu. Ovo je najefikasniji način štednje snage u neaktivnim modulima. Da bi se usvojila ova metodologija moraju biti ispunjeni sledeći uslovi:

- Prekidač napajanja mora biti dobro projektovan. Vrednost otpornosti u stanju *on/off,* i kašnjenja su ključne za rad ovog prekidača. Dobro rešenje za prekidač je tranzistor sa malom otpornošću pri vođenju. Zbog toga njegova širina mora biti velika što rezultuje velikom kapacitivnošću. Da bi se ostvarilo brzo

komutiranje prekidača iz stanja *on* u *off* potrebno je ugraditi kolo za baferovanje, što dodatno utiče na potrošnju.

- Nakon uključenja napajanja potrebno je da prođe neko vreme ΔT, pre nego što se regulisani napon napajanja stabilizuje. Imajući ovo u vidu, ova metoda postaje primenjiva samo kod onih sistema koji imaju vreme uključenja duže od ΔT.

- Deo projekta kome se isključuje napajanje ne sme da ima ugrađene elemente kakvi su registri ili memorije zbog toga što se njihov sadržaj gubi isključenjem napajanja. Da bi se ovaj problem rešio treba ugraditi dodatnu logiku koja pamti a kasnije obnavlja podatke. Ugrađena logika unosi dodatnu potrošnju snage. Korektno rešenje podrazumeva da se učini pravi kompromis između potrošnje i ugrađenog hardvera.

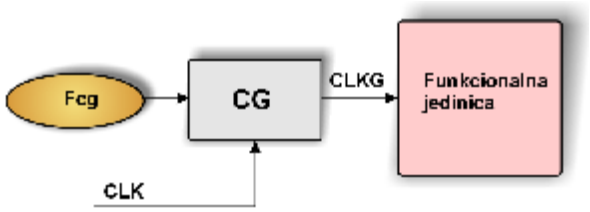
- Uključenje/ukidanje napona napajanja dovodi do pojave tranzijentnih smetnji na linijama za dovod napajanja. Ovi nepoželjni efekti moraju biti adekvatno uklonjeni tehnikom ortogonalnih razvođenja veza.

3.3.2 Dozvola/zabrana taktovanja

Dozvola taktovanja (***Clock Gating*- *CG***) je jedna od najpoznatijih tehnika za smanjenje potrošnje i veoma je efikasna za realizaciju u digitalnim kolima. Cilj ove tehnike je da ne dozvoli distribuciju taktnog signala do određenih delova kola (flip-flopova, mreže za razvođenje taktonih impulsa i druge logike) pod određenim okolnostima određenim od strane *CG* kola. Ušteda energije se uglavnom ostvaruje smanjenjem: kapacitivnosti u mreži za distibuciju takta, i redukcijom komutatorskih aktivnosti u logici za razvođenje taktnog signala [6, 7].

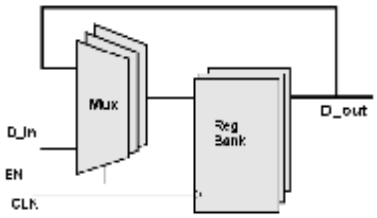
Princip implementacije *CG*-a je ilustrovan na slici 5. *CG* je blok koji, kada su ispunjeni određeni uslovi, dozvoljava/nedozvoljava taktovanje određene funkcionalne jedinice. Dozvola taktovanja je određena funkcijom *Fcg*. *CLK* je sistemski takt, a *CLKG* je "gejtovani" takt funkcionalne jedinice. *CG* tehnike su uspešno

implementirane u mnogim mikroprocesorima kao dozvola/zabrana takta na nivou: (a) modula; (b) registara; i (c) ćelija.



Slika 5• *Princip dozvole/zabrane takta*

Dozvola/zabrana (gejtovanje) takta na nivou modula podrazumeva uključenje /isključenje taktne pobude čitavom bloku ili modulu u dizajnu. Obično, odluku o tome koji modul treba biti gejtovan, donosi se na nivou sistemskog upravljanja, ili od strane projektanta. Metodom gejtovanja takta štedi se velika količina energije, naročito kada ceo blok nije operativan. Idealan slučaj je kada se blok koriti samo u toku izvršenja specificirane operacije (Na primer: delovi prijemnika i predajnika u primopredajniku ne moraju biti istovremeno aktivni. Tako na primer predajnik može biti isključen u toku prijema, a prijemnik u toku predaje). Mogućnosti ovakvog gejtovanja takta su ograničene i moraju biti identifikovane od strane projektanta.



Slika 6• *Sinhrono učitavajući registri*

Metodom gejtovanja takta na nivou registara, dozvoljava/zabranjuje se taktovanje pojedinačnih registara ili skupova registra. Registri u kojima se vrši sinhroni upis se obično realizuju od *D* flip-flop-ova i multipleksera (slika 6). U *D* flip-flop-ove se vrši upis podataka u toku svakog taktnog intervala. Kod *CG* tehnike, registar se ne taktuje ako na *D* ulazima flip-flop-ova nema novih podataka. Time se postiže znatna ušteda u disipaciji. Za potrebe gejtovanja registra neophodno je ugraditi dodatnu

logiku. Za rad ove logike neophodna je dodatna energija. Ugradnja ove logike je isplativa ako se na nivou celokupnog sistema postiže ušteda.

Projektanti često koriste *CG* tehniku i na nivou ćelija. Na primer, banka registara može biti projektovana tako da registri u toj banci mogu biti taktovani, samo u trenutku kada se učitavaju novi podaci. Slično, memorijski blok može biti taktovan samo u toku ciklusa pristupa radi čitanja/upisa. Sa jedne strane, metod *CG*-a na nivou ćelija uzrokuje povećanje površine na silicijumu, ali sa druge strane, ograničava količinu energije koja može da se uštedi. Imajući ovo u vidu svi registri u dizajnu moraju biti preprojektovani kako bi obezbedili implementaciju *CG* tehnike.

3.3.3 Korišćenje flip-flop-ova sa signalom dozvole

CG predstavlja "mekšu" varijantu u odnosu na tehniku ukidanja napona napajanja. Korišćenje flip-flop-ova sa signalom dozvole je naredna manje agresivna i manje efektivna strategija za uštedu energije. U ovom slučaju registri se realizuju pomoću flip-flopova koji imaju signal dozvole rada. Kada je signal dozvole rada neaktivan, izlaz flip-flop-a je neaktivan i samim tim redukuju se prekidačke aktivnosti u samom kolu. Međutim, takt i dalje ostaje aktivan a to ima za posledicu veliku disipaciju energije.

Treba istaći da kontrola snage zasnovana na flip-flop-ovima sa signalom dozvole može biti od koristi, ali implementacija zasnovana na *CG*-u je u suštini superiornija.

3.3.4 Particionisanje memorije

Particionisanje memorije (segmentacija), predstavlja rešenje koje smanjuje disipaciju snage detektovanjem neaktivnosti u toku pristupa memoriji. Svrha memorije je da čuva upisane podatke i da omogući njihovo čitanje kada je to potrebno. ***Farrahi*** [8] preporučuje da se na memoriju ne gleda kao na celinu, nego kao na skup nezavisnih memorijskih segmenata. Svaki memorijski segment ima svoj takt i signale za osvežavanje (kada je u pitanju *DRAM*). Kada je memorijski segment u stanju mirovanja, tada se on može postaviti u ***sleep mode*** u toku koga ne postoji taktovanje

kao i osvežavanje. Ako u memoriji nema zapamćene korisne informacije tada je ona neaktivna. Treba naglasiti da memorija nije u stanju mirovanja kada se njoj pristupa. Ona može čuvati informaciju od značaja koja se gubi isključivanjem memorije. Takođe memorija može čuvati nevažeće odnosno nebitne informacije. Svakoj memorijskoj lokaciji može biti dodeljeno vreme života. Ono određuje aktivnost promenjive, odnosno vremenski period od upisa do poslednjeg čitanja memorijske lokacije. Za segment se kaže da je u stanju neaktivan ako ne sadrži aktivnu promenjivu. U okviru istog segmenta, tehnika particionisanja memorije teži da sačuva promenjive čija se vremena života preklapaju.

Ovakvim načinom pristupa produžava se period kada je memorijski segment neaktivan, a samim tim se redukuje disipacija.

3.4 Smanjenje potrošnje usled struje curenja

Porastom gustine pakovanja kod *CMOS* kola, disipacija snage usled struje curenja postaje sve dominantnija, jer ona postoji i kada je kolo neaktivno. Ova disipacija može da iznosi i više od 50% u odnosu na ukupnu snagu disipacije. Jedna od metoda za redukciju ove disipacije je korišćenje višestrukih naponskih pragova. U toku procesa sinteze na logičkom i fizičkom nivou, ovom tehnikom zahteva se lociranje niskopragovskih ćelija na kritičnim putanjama signala, a visokopragovskih čelija na nekritičnim putanjama. Ostale poznate tehnike imaju za cilj redukciju snage usled struje curenja podrazumevaju da se moraju uvesti značajne novine u metodi projektovanja. Mnogi projektanti, u cilju redukcije disipirane snage, koriste tehniku smanjenjnja napona substrata.

Od trenutka uveđenja 0.5 μm tehnologije, naponski nivoi kod *IP*-ova (***I**ntellectual **P**roperty*) su skalirali do 1V/0.1 μm. Skaliranjem tehnologije na ispod 100 nm, dolazi do redukovanja napona napajanja a to ima za posledicu i skaliranje napona praga provođenja ispod 0.25 V. Ovaj efekat ima dominantni uticaj na povećanje struje curenja tranzistora. Aproksimativno svako smanjenje napona praga provođenja od 65-

85 mV (u zavisnosti od temperature) dovodi do povećanja subpragovske struje curenja za jedan red veličine.

$$I_{sub} = I_o\left[e^{-\frac{V_{th}}{S}}\left(1-e^{-q\frac{V_{ds}}{kT}}\right)\right] \quad , \text{ kada je } V_{gs}=0 \qquad (17)$$

Kao što se vidi iz relacije (17), subpragovska struja curenja, I_{sub}, raste eksponencijalno kako se napon praga provođenja, V_{th}, smanjuje.

3.4.1 Multipragovski dizajn

Fabrike za proizvodnju poluprovodnika, za isti proces proizvodnje, nude komponente koje imaju više pragova. Osnovni moto ovakvog dizajna je da se ostvari kontrola i smanji struja curenja, čime se sa aspekta potrošnje poboljšavaju performanse sistema [9]. Uz tranzistore koji imaju standardni V_{th}, na istom čipu se izrađuju kako niskopragovski tako i visokopragovski tranzistori. Obično niskopragovska kola imaju za red veličine veće struje curenja od kola koja imaju standardne V_{th}, a visokopragovska kola imaju struje curenja koje su red veličine ispod standardnih.

Kod specijalnih aplikacija, mogu postojati specijalna kola koja se karakterišu malim strujama curenja, a koja imaju za cilj da redukuju struje curenja za još jedan red veličine. Ovo smanjenje struje curenja nije neograničeno jer ima za posledicu smanjenje brzine rada kola. Ilustracije radi, kašnjenje kroz kolo može da varira od 20-200% kada se upoređuju kola sa standardnim i visokim pragom provođenja.

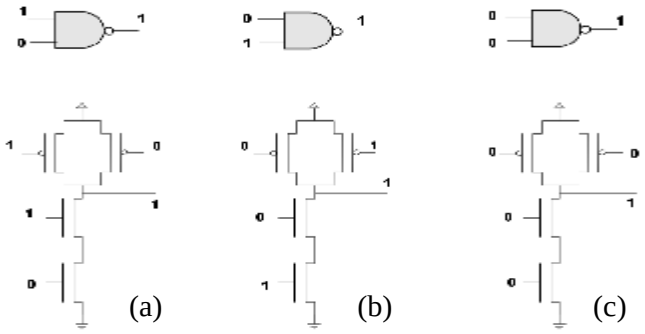
Izazov za *EDA* alate je korišćenje karakteristika ćelija iz dizajn biblioteka, čime bi se kreirala implementacija koja će zadovoljiti vremenska ograničenja, smanjujući struje curenja što je moguće više.

Najjednostavnije je, kao meru za ocenu kvaliteta dizajna, koristiti određeni procenat visokopragovskih i niskopragovskih ćelija. Vrlo je važno imati na umu da je cilj smanjenje ukupne struje curenja. Dizajn sa velikim procentom visokopragovskih ćelija, na prvi pogled može izgledati bolje rešenje, ali takođe može biti i inferioran u

odnosu na dizajn sa manjim procentom ovih ćelija. Iz ovog razloga ovaj kompromis mora se uzeti u obzir još na početku procesa sinteze.

Bolji pristup, koristeći višestruko pragovske ćelije, je da se napravi post-logička rutina za sintezu koja zamenjuje niskopragovske ćelije sa visokopragovskim ćelijama, sve dok se ne naruše ograničenja u pogledu kritičnog vremena izvršenja kao i ostala pravila u dizajnu. Ćelije mogu biti kreirane tako da imaju isti *footprint* kako bi se olakšao postupak optimizacije koja prati fazu razmeštanja ćelija na površini silicijuma.

Sofisticiraniji pristup, koji se koristi tokom sinteze, kao komponentu funkcije optimizacije, uzima u obzir snagu disipacije usled struje curenja. Zato je važno imati biblioteke koje su korektno kreirane za izvršenje ovog vida optimizacije. Informacija o curenju, zavisno od stanja, može se iskorstiti radi smanjenja struje curenja u dizajnu. Podrazumeva se da je struja koju ćelija troši zavisna od stanja na njenim ulazima. Ilustracije radi na slici 7 prikazani su različiti načini pobude jednog *NAND* gejta pri čemu je za sve ulazne kombinacije napon na izlazu na vrednost logičke jedinice. Struja curenja je najveća za kolo sa slike 7(*a*), a najmanja za kolo sa slike 7(*c*). U konkretnom slučaju u cilju smanjenja dinamičke snage, može se koristiti tehnika zamene pinova čime se smanjuje ukupna struja curenja komponente ili curenje za dato stanje. Da bi izvršili ovu optimizaciju, neophodno je da biblioteke sadrže informacije o curenju (zavisnog od stanja) ćelija. Još jedna neophodna komponenta je poznavanje verovatnoća stanja na ulazima. Za više detalja o ovoj problematici videti Referencu [10].



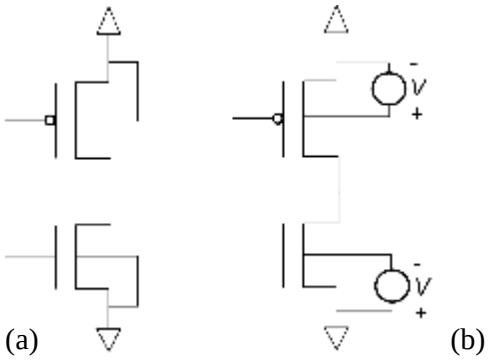
Slika 7• *Curenje u zavisnosti od ulaznih signala*

Još jedna relevantna oblast koja se odnosi na višepragovski dizajn je uticaj osetljivosti na šum. Višepragovski uređaji će po prirodi imati manju osetljivost na smetnje. Alati za optimizaciju koji simultano uzimaju u obzir uticaj integriteta signala mogu koristiti ove ćelije radi poboljšanja osobina integriteta signala kao i struja curenja.(Napomena: Pojam integritet signala se odnosi na uticaje *EMC*-a, preslušavanja po linijama za razvođenje signala, smetnji od napona napajanja itd.)

3.4.2 Promenljivi prag napona provođenja

Prag napona provođenja kao i struja curenja kod *CMOS* tranzistora se može regulisati promenom napona polarizacije (***B**ack **B**iasing **V**oltage- BBV*). Promena napona praga provođenja je približno proporcionalan kvadratnom korenu *BBV*-a. Kada napon praga provođenja padne ispod 0.25 V, svrsishodnije je koristiti promenljivi *BBV*.

Važna prednost ovog pristupa je ta što tokom perioda kada je potrebno intenzivno procesiranje, napon praga se može smanjiti, čime se poboljšava brzina rada ćelije. Kada ćelije rade sa manjom brzinom ili su u pasivnom stanju (*idle*), napon praga se može povećati, čime se smanjuje struja curenja.



Slika 8• *Korišćenje promenljivog BBV-a*

Značajan uticaj korišćenja promenljivog *BBV*-a je taj što se uvode dva nova izvoda za svaku ćeliju. Standardna praksa u *ASIC* dizajnu je da se kreiraju ćelije koje povezuju *N-Well* na V_{DD} i *P-Well* oblasti na masu. Kod fizičke implementacije ovo su unapred definisani kontakti dizajnirani unutar ćelije, povezani na napajanje i masu respektivno. Na slici 8 prikazan je koncept osnovne strukture promenjive polarizacije.

Na slici 8(a) je prikazana tradicionalna implementacija, kod koje se povezivanje vrši direktno na V_{DD} i V_{SS}, a na slici 8(b) prikazano je kako se vrši povezivanje korišćenjem polarizacije.

4 Metode projektovanja za malu potrošnju

Projektovanje sistema, pored optimizacija za velike brzine i male površine, mora da sadrži i sredstva za optimizaciju za malu potrošnju. Metode projektovanja za malu potrošnju koriste se tokom svih faza projektovanja od sistemskog do tehnološkog nivoa. U isto vreme, projektovanje radi efikasnih ušteda energije ima za efekat uštedu u površini čipa i uštedu projektantskog vremena. Na slici 9 su prikazani nivoi projektovanja sa odgovarajućim tehnikama, koje se koriste za *low power* projektovanje.

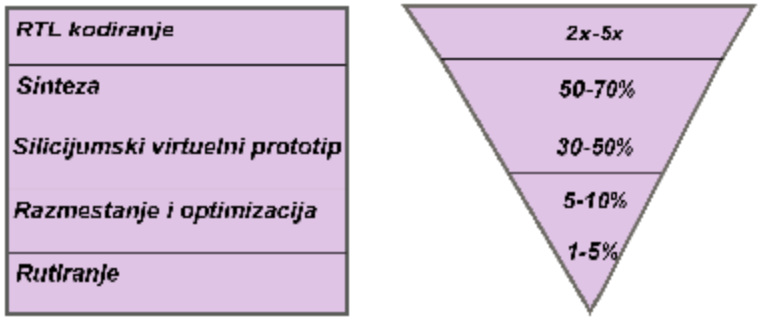
Slika 9• *Nivoi projektovanja low power dizajna*

Na sistemskom nivou se koriste tehnike koje su najefikasnije po pitanju potrošnje, kao što su: *Power down* tehnike i tehnike particionisanja dizajna. Algoritamski nivo omogućava korišćenje tehnika kao što su: kompleksnost, paralelno izvršavanje više operacija, lokalnost, regularnost i prezentacija podataka. Na nivou arhitekture se koriste tehnike: skaliranje napona napajanja, korišćenje paralelizama, izbora skupa instrukcija, korelisanost signala, *CG* tehnike i dr. Logički nivo, takođe vrlo važan za malu potrošnju, obuhvata sledeće tehnike: određivanje širine kanala tranzistora, logička optimizacija, upravljanjem logike za *power down*, *low-swing* logika, *adiabatic* kola i dr. Najniži nivo je fizički i obuhvata metode redukcije potrošnje kao što su: redukcija napona praga provođenja i multipragovski dizajn.

Iako je ušteda energije najveća na sistemskom nivou, o njoj se mora voditi računa tokom svih koraka toka projektovanja (*design flow)* [3].

4.1 *VHDL* metode za redukciju potrošnje

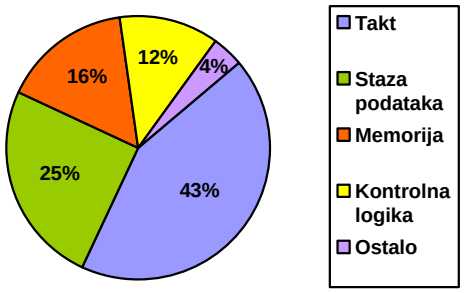
Za dobar dizajn, važna su sledeća dva aspekta uštede energije: (a) nivo apstrakcije, i (b) kvalitet čipa na površini silicijuma. Što je nivo projektovanja viši, veće su i mogućnosti uštede energije, iz razloga što je veći stepen slobode prilikom projektovanja. Na slici 10 prikazano je kako se na različitim nivoima, prilikom projektovanja, mogu ostvariti uštede energije.



Slika 10• *Ušteda energije tokom faza projektovanja*

Disipacija na *RTL* nivou se može redukovati od *2x* do *5x*. Ovo ukazuje na važnost pravilnog projektovanja na *RTL* nivou.

Analizom rada jednog procesora koji se odnosi na izvršenje skupa *benchmark* programa, nakon statističkog sređivanja dobijenih rezultata po pitanju potrošnje, dobijeni su rezultati koji su prikazani na slici 11. Rezultati se odnose na potrošnju pojedinih blokova u okviru *CPU*-a. Kao što se vidi sa slike 11 logika takta troši najviše energije, 43% od ukupne potrošnje. Zbog toga se na projektovanje logike takta i njegove distribucije mora obratiti posebna pažnja. Staza podataka troši oko 25% od ukupne potrošnje. Pažljivim projektovanjem registara, magistrala, *ALU*-a i drugih blokova koji čine stazu podataka, može se značajno redukovati potrošnja. Oko 16% od ukupne potrošnje odlazi na memorijski blok. Najmanji, ali ne i zanemarljiv, potrošač je *I/O* kontrolna logika. Korektnom realizacijom *I/O* drajvera može se postići znatna ušteda.



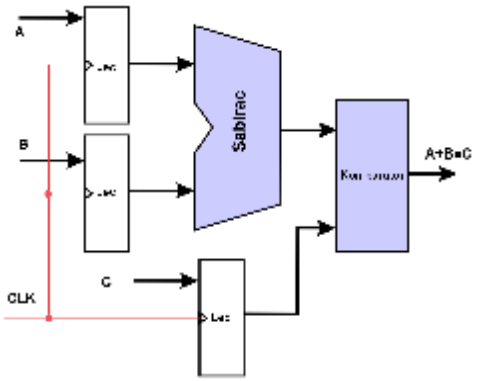
Slika 11• *Potrošnja određenih blokova jednog procesora*

U daljem tekstu ćemo ukazati na neke od tehnika za redukciju potrošnje na *RTL* nivou.

4.1.1 Skaliranje napona napajanja na arhitekturnom nivou

Ranije spomenuti tehnološki pristupi redukcije napona napajanja bili su bazirani na smanjenju napona napajanja uz očuvanje brzine komponente. Uz primenu odgovarajućih tehnika na *RTL* nivou pomoću *CMOS* gejtova mogu se postići i niže

vrednosti proizvoda energija-kašnjenje (*energy-delay product*). Tako na primer, kada kolo radi sa malim naponom napajanja, promenom arhitekture kola može se kompenzovati smanjenje brzine rada kola. Radi ilustracije, analiziraćemo arhitekturu koju čini 8-bitna staza podataka komponovana od sabirača i komparatora kao na slici 12.



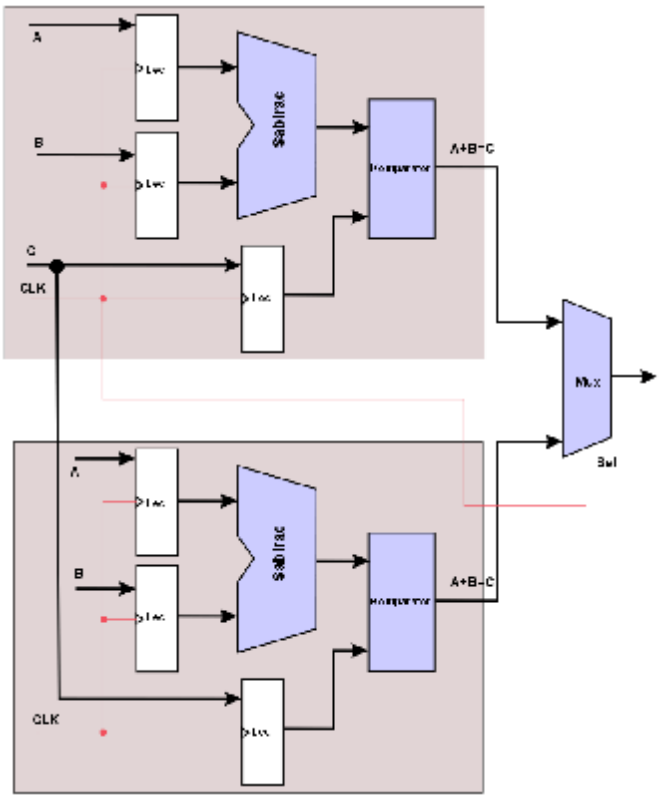
Slika 12• *Staza podataka*

Za konkretno rešenje, u najgorem slučaju kašnjenje kroz leč & sabirač & komparator, za napon napajanja 5 V, iznosi približno 25 ns. To znači da sistem, u najboljem slučaju, može biti taktovan signalom periode T=25 ns. Pri ovome treba da bude jasno, da se pri maksimalnom propusnošću radni napon ne može više redukovati, jer to uzrokuje dodatno kašnjenje koje se ne može tolerisati.

Nezavisno od svega, stazu podataka sa slike 12, koristićemo kao referentnu u daljoj analizi. Disipirana snaga referentne staze podataka data je sa:

$$P_{ref} = a_{0\to 1} C_{ref} V_{ref}^2 f_{ref} \qquad (18)$$

Jedan od načina da se ne naruši vrednost propusnosti ogleda se u korišćenju paralelne arhitekture. Konkretno rešenje prikazano je na slici 13, gde se koriste dve identične staze podataka. Napomenimo da svaka staza podataka radi na polovini početno zadate brzine (taktuje se duplo nižom frekvencijom) čime početna vrednost propusnosti ostaje nepromenjena.



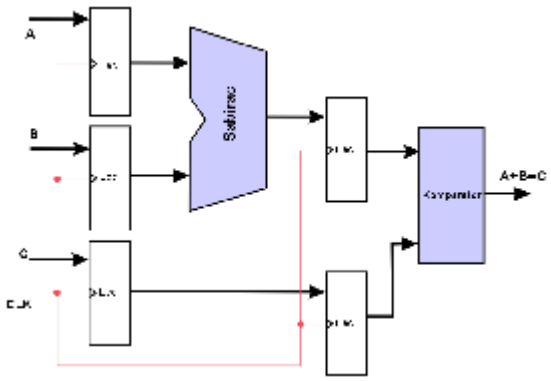
Slika 13• *Paralelna implementacija staze podataka*

Pošto su se sada zahtevi za vremenom procesiranja sabirača i komparatora povećali sa 25 ns na 50 ns, to znači da se napon napajanja može redukovati sa 5 V na 2.9 V (kašnjenje pri 2.9 V je dvostruko veće od kašnjenja pri 5 V- vidi sliku 1 na strani 7). Sa druge strane kapacitivnost staze podataka se povećava za faktor 2.15 (određuje se nakon izrade *layout*-a kola [4]), a radna frekvencija se smanjuje za faktor 2 (zbog uvođenja paralelizma). Snaga disipacije sada iznosi:

$$P_{par} = a_{0\to 1} C_{par} V_{par}^2 f_{par} = (2.15 C_{ref})(0.58 V_{ref})^2 \left(\frac{f_{ref}}{2}\right) \approx 0.36 P_{ref} \qquad (19)$$

Upoređivanjem (18) i (19) uočava se da je ušteda snage značajna. No kao posledicu ovaj metod ima negativni efekat koji se ogleda u povećanju površine čipa, što ga čini nepogodnim kada je površina na silicijumu ograničena. Treba imati na umu da paralelizam iziskuje i dodatno rutiranje, što zahteva pažljivu optimizaciju.

Drugi mogući pristup rešavanja ovog problema odnosi se na primenu protočne obrade, kao što je prikazano na slici 14.



Slika 14• *Protočna implementacija dela staze podataka*

Umetanjem dodatnih protočnih lečeva, kritični put postaje max (T_{adder}, $T_{comparator}$), što omogućava sabiraču i komparatoru da rade sa manjom brzinom. U ovom primeru, kašnjenja kroz protočne stepene treba balansirati. Ova tehnika obezbeđuje uslov za smanjenje napona napajanja sa 5 V na 2.9 V bez narušavanja sistemske propusnosti. Tehnika protočnosti iziskuje manje prostora u odnosu na paralelizam jer se zahteva ugradnja samo protočnih registra. Potrošnja protočne staza podataka iznosi:

$$P_{pipe} = a_{0\to 1} C_{pipe} V_{pipe}^2 f_{pipe} = (1.15 C_{ref})(0.58 V_{ref})^2 f_{ref} \approx 0.39 P_{ref} \qquad (20)$$

Upoređivanjem (19) i (20) uočava se da je ušteda snage u oba slučaja skoro identična, ali je dodatno ugrađeni hardver manji kod protočno organizovane staze podataka značajno manji. Zbog toga je ovo rešenje superiornije.

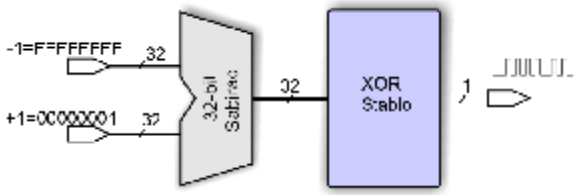
Naredno poboljšanje se dobija kombinacijom protočnosti i paralelizma. S obzirom da ova arhitektura redukuje kritični put, a samim tim i zahteve za brzinom rada za faktor 4, napon napajanja može biti redukovan na vrednost pri kojoj se povećava kašnjenje za faktor 4. Potrošnja je u ovom slučaju iznosi:

$$P_{parpipe} = a_{0\to 1} C_{parpipe} V_{parpipe}^2 f_{parpipe} = (2.5 C_{ref})(0.4 V_{ref})^2 \left(\frac{f_{ref}}{2}\right) \approx 0.2 P_{ref} \qquad (21)$$

Upoređivanjem (18) i (21) uočavamo da paralelno protočna implementacija rezultuje petostrukom smanjenju potrošnje u odnosu na referentnu arhitekturu [4].

4.1.2 Redukcija gličeva

Gličevi su kratkotrajni nepoželjni signali koji se javljaju kao rezultat propagacije signala različitim putevima. Kao ilustraciju razmotrimo primer dat na slici 15. Na izlazu 32-bitnog sabirača je povezana kombinaciona mreža koju čini *XOR* stablo a koristi se da broji broj jedinica u sumi na izlazu sabirača, a na svom izlazu generiše bit parne parnosti. U slučaju kada se vrši sabiranje vrednosti +1 i –1, tada će bit parnosti na izlazu *XOR* stabla oscilovati veći broj puta pre nego što se stabilizuje na vrednost 0. S obzirom da su krajnji i inicijalni rezultati isti, sve aktivnosti koje se odnose na izlazni čvor ne nose novu informaciju nego doprinose povećanoj disipaciji usled egzistencije $P_{dynamic}$ i P_{scc}. Oscilacije se takođe propagiraju i do ostalih kombinacionih blokova i mogu generisati lažne aktiivnosti.



Slika 15• *Primer generisanja gliča*

Ova propagacija se može zaustaviti ugradnjom sekvencijalnih elemenata ili eliminisanjem ovih tranzijntnih impulsa za slučaj kada je njihovo vreme propagacije kroz logiku duže od vremena trajanja ovih impulsa. Gličevi nisu aktuelni samo sa aspekta potrošnje nego i korektnog rada. Zbog parazitnih kapacitivnih sprega, gličevi takođe utiču na integritet signala.

4.1.2.1 Upravljanje na nivou gejta

Osnovna ideja za sprečavanje propagacije gličeva je korišćenje dizajna baziranog na protočnoj obradi. Nažalost ovaj veoma uspešan metod zahteva ugradnju dodatnh registara, povećava latenciju, usložnjava kontrolnu logiku i komplikuje izvođenje

distribucije takta. Logika stabla za razvođenje takta kao i logika dodatno ugrađenih registra su solidni potrošači kako statičke tako i dinamičke energije, pa zbog toga projektant mora da učini kompromis u dizajnu. Šta više, protočnost nije uvek moguća jer ona kasni generisanje izlaznih podataka za jedan ili više taktnih intervala. U nekim slučajevima, zahtevi za promenom u arhitekturi se mogu odnositi na kompajler kao i na operativni sistem za rad u realnom vremenu, što je često nemoguće. Kompromisno rešenje je da se podeli taktni interval u dve ili više faza. Višefazno taktovanje koristi se za maskiranje (za dozvolu ili zabranu propagacije) signala staze podataka uz pomoć jednostavnih *AND* kola ili uz pomoć lečeva. Standardna implementacija je dobro poznata dvofazna *master-slave* leč logika. Na žalost cena koja mora da se plati sa aspekta generisanja takta i distribucije, statičke vremenske analize, i kompleksnosti dizajna mora biti pažljivo procenjena pre korišćenja ovako složenih šema koje se sinhrono taktuju.

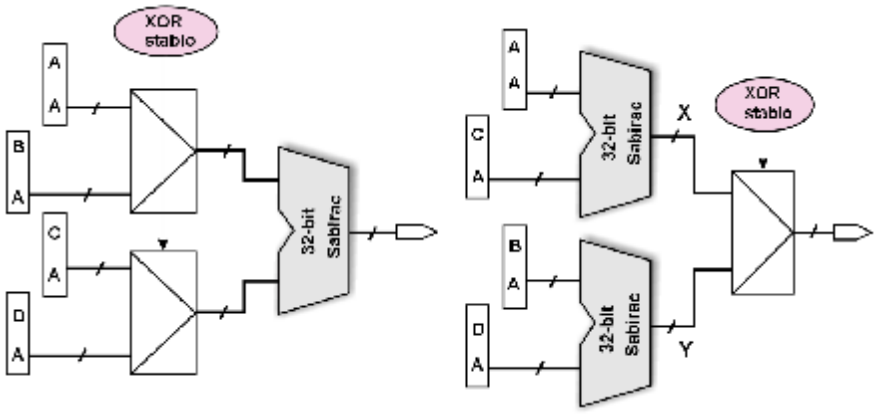
Drugi pristup sastoji se u nalaženju dobrog balansa kašnjenja koji postoji između različitih puteva u kombinacionoj mreži. Ćelije za kašnjenje se direktno instanciraju u *RTL* nivo a fino podešavanje kašnjenja se izvodi u fazi razmeštaja i rutiranja. S obzirom da su kašnjenja zavisna kako od varijacija u procesu proizvodnje tako i temperature, ova metoda je veoma teška za implementaciju. Ove tehnike se ne preporučuju kod tehnologije gde se koristi skaliranje *CMOS* kola. Ovaj pristup je jedino isplativ kod *full-custom* dizajna i kod projektovanja specifično visoko performansnih blokova.

Aktivnost gličeva se takođe može redukovati korišćenjem *Boole*-ovih jednačina tipa suma proizvoda. Uz pomoć ovog pristupa mogu se smanjiti gličevi, ali po ceni povećanja površine čipa i dinamičke potrošnje snage.

Kada se koriste veoma brzi logički blokovi poželjnije je implementirati domino logiku. Kod ovog tipa logike ne javljaju se gličevi ali projektant treba da raspolaže namensko projektovanim bibliotečkim ćelijama kao i specifičnom logikom za razvođenje signala.

4.1.2.2 Upravljanje na nivou bloka

Ova tehnika se sprovodi preuređivanjem logičke strukture. Da bi ilustrovali ovo rešenje analizirajmo sliku 16. Koristimo *XOR* stablo da bi izabrali *A* ili *B* kao prvi operand 32-bit-nog sabirača i *C* ili *D* kao drugi operand.



Slika 16• *Redukcija gličeva blokovskim preuređivanjem*

S obzirom da su *A,B,C* i *D* izlazi registra, oni su stabilni podatci, ali ako selektorski signal multipleksera osciluje, tada će operandi sabirača biti nestabilni i generisaće gličeve koji troše snagu. Ako koristimo dva sabirača da bi prvo izračunali *X* i *Y* sume a zatim ih multipleksiramo, tada će sabirači prihvatati stabilne ulaze i imati dosta manju potrošnju snage zbog izostanka gličevima. Ova redukcija se sprovodi po ceni ugradnje jednog dodatng 32-bit-nog sabiračkog bloka.

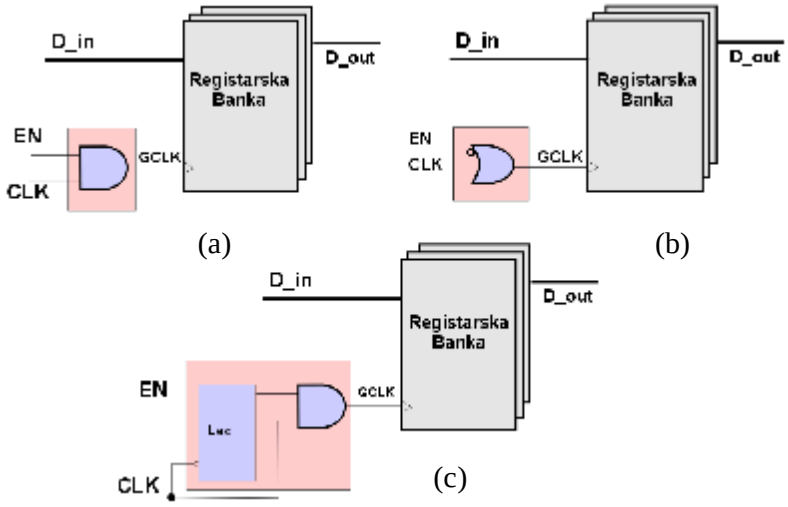
4.1.3 Dozvola/zabrana takta na nivou registra

U toku tradicionalnog automatskog *ASIC* dizajna predpostavlja se da svi registri sinhrono učitavaju podatke. Ova paradigma je narušena kod *CG* ***(Clock Gating)*** tehnike na *RTL* nivou, što stvara dodatne projektantske izazove u *ASIC* dizajnu. U daljem tekstu ukazaćemo na izazove sa kojima se srećemo kod masovng *CG*-ovanja u konkretnim rešenjima i ukazaćemo na neke od mogućih rešenja. Radi jednostavnijeg objašnjenja usvajamo da registri *CG*-uju rastućom ivicom takta, ali svi primeri na koje ukazujemo koristiće invertovane taktove tj. registre koji se okidaju sa negativnom ivicom.

4.1.3.1 Primeri implementacije kola za gejtovanje

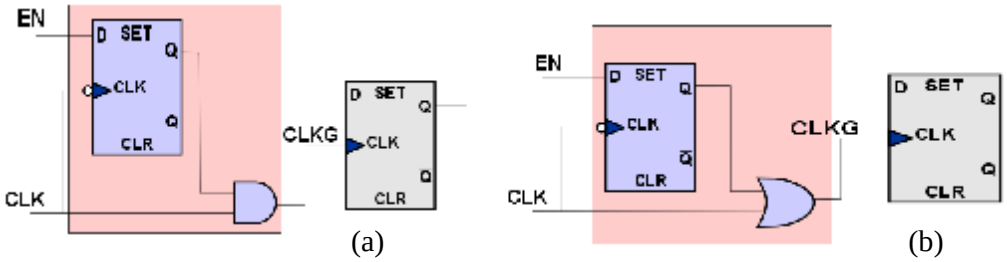
Kod jednostavne *CG* konfiguracije, slika 17(a), gličevi koji se javljaju u signalu *enable* (dozvola) kada je taktni signal na visoko propagiraju se do pina *Clock* kao ulaznog pina registra. Propagacija najvećeg broja gličeva može se uspešno zaustaviti korektnim generisanjem *enable* signala, *EN*, koji se dovodi na ulaz *AND* gejta. U ovom slučaju korektnost se pre svega odnosi na ograničenja u pogledu *setup* i *hold* vremena registara u kojima se vrši lečovanje informacija. Bilo kakve nepoželjne promene u toku rada (zbog parazitnih sprega sa drugim signalima), mogu da uzrokuju lečovanje pogrešnih vrednosti u *CG*-ovane registre. Nešto sigurniji metod sa aspekta *CG*-ovanja sastoji se u korišćenju *OR* kola kao što je prikazano na slici 17(b). Ovo kolo zadržava izlaznu vrednost na logičkoj jedinici kada je signal dozvole *EN* na visokom nivou. Nepoželjni gličevi u toku rada kod ove konfiguracije mogu da dovedu do punjenja pogrešnih vrednosti u flip-flop, ali će njihova konačno zapamćena vrednost biti ona koja se javlja na rastućoj ivici takta i što je najvažnije biti korektna.

Bolji način da se izbegne ovaj problem je da se ugradi leč koji će na *EN* putu biti osetljiv na nizak nivo signala, vidi sliku 17(c). Ovim se zamrzava izlaz leča na rastućoj ivici taktnog signala i omogućava da novi *EN* signal na ulazu *AND* kola bude stabilan kada je takt na visokom nivou. Pored toga, signal *EN* može pozajmiti vreme od leča, tako da u suštini ima ceo taktni period za propagiranje ka leču.



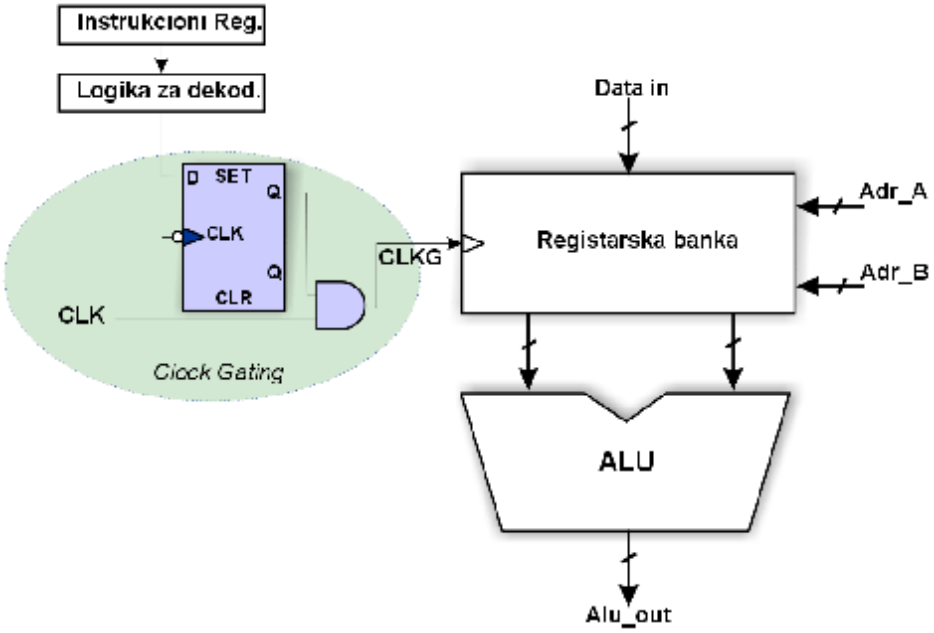
Slika 17• *Primeri implementacije kola za gejtovanje takta*

Umesto leča može biti korišćen flip-flop koji se okida opadajućom ivicom (slika 18). Ova varijanta obezbeđuje generisanje "čistog" signala na ulaz *AND* kola, a kao rešenje zahteva da *EN* signal bude stabilan pre opadajuće ivice takta, što nažalost rezultuje strogim vremenskim ograničenjima na putu signala *EN*.



Slika 18• *Flip-flop CG implementacija:(a)pomoću AND kola, (b)pomoću OR kola*

Kolo dato na slici 19 prikazuje *CG*-nje registarske banke jedne proizvoljne staze podataka.



Slika 19• *Primer takt gejtovanja*

Kolo za *CG* i skup registara moraju biti fizički bliski kako bi se smanjio uticaj košenja takta (*clock skew)* i sprečile neželjene optimizacije puta signala u fazi sinteze. Logika koju čine *CG* i registri, se može modelirati pomoću dva odvojena procesa u istom hijerarhijskom boku. Prvo se obavlja njihova sintetiza a zatim se ova kola ubacuju u

dizajn koristeći hijerarhijski princip sa *dont touch* atributom. *VHDL* kôd prikazan na slici 20 opisuje skup registara i pridruženo *CG* kolo:

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

-- entity CG_RF_e sadrži pored signala za taktovanje (clk) i
-- 5bit-ne adresne linije adrA, adrB; Ulaznu magistralu
-- za podatke, datain, koja je 8bit-na; ctrl,kontrolni ulaz; ulaz wr;
-- i 8bit-ne izlaze A i B;
entity CG_RF_e is
      port (
      clk   :in std_logic;
      adrA  :in std_logic_vector (4 downto 0);
      adrB  :in std_logic_vector (4 downto 0);
      datain:in std_logic_vector (7 downto 0);
      wr    :in std_logic;
      ctrl  :in std_logic;
      A,B   :out std_logic_vector (7 downto 0));
end CG_RF_e;

architecture CG_RF_a of CG_RF_e is
signal clkg: std_logic; -- predstavlja sig. Dobijen od CG ćelije;
type ram is array (0 to 31) of std_logic_vector (7 downto 0);
signal RF: ram;

-- Arhitektura CG_RF_a je opisana pomoću dva procesa i to: CG i Clkg
-- Proces CG opisuje CG ćeliju i osetljiv je na signal takta i
-- kontrolni signal ctrl;
begin
      CG:Process (clk,ctrl)
      variable qint:std_logic;
      begin
            if clk='0' then
            qint :=ctrl;
            end if;
            clkg<=(not qint) and clk;
      end process;

-- Proces Clkg osetljiv je na signal gejtovanog takta clkg i
-- opisuje stazu podataka;
      Clkg:process (clkg)
      begin
            if clkg='1' then
                  if wr='1' then
                        RF(conv_integer(adrA))<=datain;
                  else
                        A<=RF(conv_integer(adrA));
                        B<=RF(conv_integer(adrB));
                  end if;
            end if;
      end process;
end;
```

Slika 20• *VHDL kôd skupa registara i pridruženog CG kola*

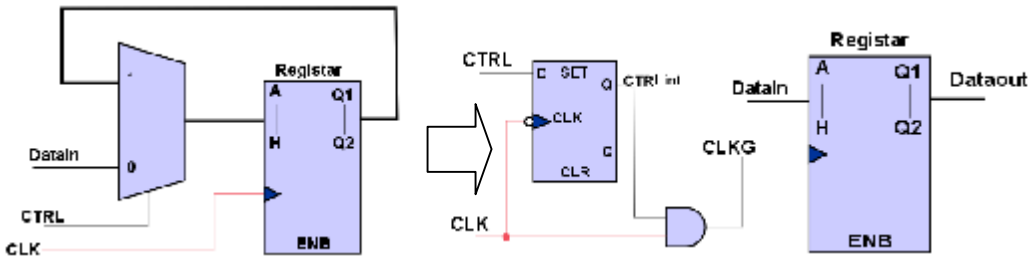
VHDL kôd prikazan na slici 21 predstavlja primer koji ilustruje tehniku sa odvojenim izvršnim procesima [19]. Inicijalni proces (*P0*) u arhitekturi *listing2_1_a* je transformisan u tri procesa (*P1, P2, P3*), kao što je prikazano u arhitekturi *listing2_2_a*. *Load* signal bez gličeva *c_load* je generisan i kombinovan sa taktom, *clk*, da generiše gejtovan takt, *clkg*, da bi bio upotrebljen od strane procesa *P2*.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
-- Entitet listing2_e sadrži: 8bit-ne ulaze A, B, C i E;
-- Izlaze X, D i Z koji su takođe 8bit-ni; signal takta clk;
-- i signal load;
entity listing2_e is
      port(
      clk : in std_logic;
      load : in std_logic;
      A,B,C,E: in std_logic_vector(7 downto 0);
      X,D,Z: out std_logic_vector(7 downto 0));
end listing2_e;

-- Za predhodno opisani entitet u nastavku
-- imamo dve arhitekture i to listing2_1_a i
-- listing2_2_a
-- Listing2_1_a opisuje određenu stazu podataka;
architecture listing2_1_a of listing2_e is
begin
      P0 : process (clk)
      begin
            if (clk'event and clk='1')then
                  X<= A + B;
                  D<= E;
                  if (load='1') then
                        Z <= C;
                  end if;
            end if;
      end process;
end architecture;

-- Arhitektura koja deli process P0 na tri prosesa
-- i to P1, P2 i P3 prikazana je u nastavku;
architecture listing2_2_a of listing2_e is
signal Gclk : std_logic;
-- Proces P1 je osetljiv na clk;
begin
      P1 : process (clk)
      begin
            if clk'event and clk='1' then
                  X<= A + B;
                  D<= E;
            end if;
      end process;
-- Proces P2 je osetljiv na unutrašnji signal Gclk;
      P2 : process (Gclk)
      begin
```

```
        if Gclk'event and Gclk='1' then
              if (load='1') then
                    Z <= C;
              end if;
        end if;
  end process;

  P3: process (clk, load)
  Variable c_load: std_logic;
  begin
        if clk = '0' then
              c_load := load;
        end if;
        Gclk <= clk and c_load;
  end process;
  end;
```

Slika 21• *VHDL kôd skupa registara i pridruženog CG kola pomoću više procesa*

U nekim hardverskim rešejnima, korite se strukture sa flip-flop-ovima kod kojih se signal dozvole generiše kao što je prikazano na slici 22(a). U principu ugrađeni flip-flopovi povećavaju površinu i potrošnju, ali prednost ovog rešenja u odnosu na ono zasnovano na *CG*-nju je ta što se dizajn može lako testirati. Pored toga i problem košenja takta je lakše rešiti. Ova struktura može biti lako transformisana u *CG* strukturu kao što je to prikazano na slici 22(b).



(a) (b)

Slika 22• *Transformacija logike za dozvolu rada u CG logiku*

Pri ovome treba naznačiti da se pomenutom transformacijom postiže značajna ušteda u potrošnji i površini čipa. *VHDL* kôd prikazan na slici 23, opisuje flip-flop sa dozvolom rada i odgovarajuću *CG* verziju.

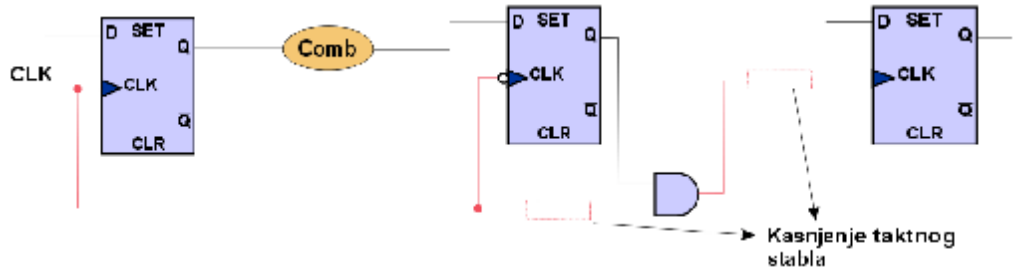
```
-- kôd koji opisuje stazu podataka realizovanu
-- pomoću flip-flopova sa signalom dozvole;
library ieee;
use ieee.std_logic_1164.all;
```

```
entity EDFF_e is
      port(
      clk : in std_logic;
      ctrl : in std_logic;                -- signal dozvole;
      datain: in std_logic_vector(7 downto 0);
      dataout : out std_logic_vector(7 downto 0));
end EDFF_e;

-- Arhitektura EDFF_a opisuje DFF sa signalom dozvole
-- pomoću procesa P0 koji je osetljiv na takt, clk;
architecture EDFF_a of EDFF_e is
begin
      P0:process (clk)
      begin
            if clk'event and clk = '1' then
                  if (ctrl='1') then
                        dataout <= datain;
                  end if;
            end if;
      end process;
end;

--Kôd koji opisuje resenje sa CG;
library ieee;
use ieee.std_logic_1164.all;
entity CG_DFF_e is
      port(
      clk : in std_logic;
      ctrl : in std_logic;
      datain: in std_logic;
      dataout : out std_logic);
end CG_DFF_e;
-- Arhitektura CG_DFF_a je opisana pomoću sledećih procesa:
-- Proces P1, osetljiv na unutrašnji signal clkg, koji
-- koji opisuje registarsku banku;
-- Proces P2, osetljiv na spoljašnji takt i kontrolni signal,
-- koji opisuje CG kolo;
architecture CG_DFF_a of CG_DFF_e is
signal clkg : std_logic;
begin
      P1:process (clkg)
      begin
            if clkg'event and clkg = '1' then
                  dataout <= datain;
            end if;
      end process;

      P2:process (clk, ctrl)
      variable ctrl_int: std_logic;
      begin
            if clk = '0' then
                  ctrl_int := ctrl;
            end if;
            clkg <= clk and ctrl_int;
      end process;
end;
```

Slika 23• *Kôd koji opisuje flip-flop sa dozvolom rada i njegovu odgovarajuću CG-nu verziju*

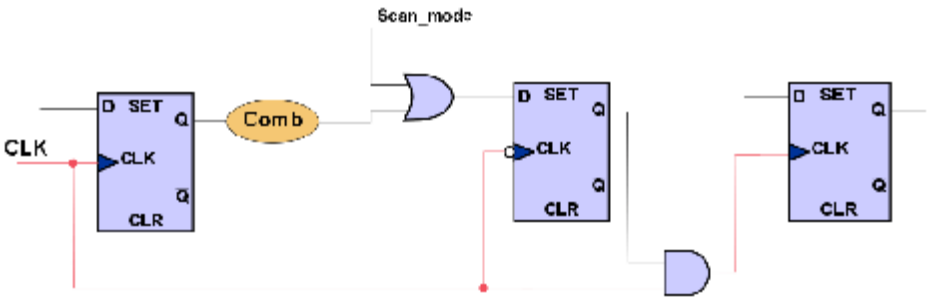
4.1.3.2 Problemi kod CG-ovane logike zasnovane na D flip-flop-ovima

Problemi sa tajmingom. Kola za taktovanje (*AND* ili *OR tipa*) ne smeju promeniti oblik taktnog signala osim da dozvolu za generisanje takta postave u stanje *on* ili *off*. Nažalost, uvođenje logike za *CG*-e rezultuje u narušavanju vremena *setup* i *hold*. Šta više, u mnogim rešenjima *CG* se sprovodi pre sinteze taktnog stabla [20,21]. Kod postojećih sredstava za sintezu, postavljanje određenih promenljivih obezbeđuje projektantu pristup da specificira kritična vremena pre procesa sinteze. Biranjem ovih vremena projektant mora da proceni uticaj kašnjenja taktnog stabla od *CG* kola do gejtovanog registra kao što je prikazano na slici 24.



Slika 24• *Problemi kašnjenja kod CG tehnike*

Problemi u vezi testabilnosti. *CG*-om se uvodi veći broj taktnih domena u dizajnu, a to utiče na testabilnost kola. Jedan od načina da se poboljša testabilnost dizajna sastoji se u ubacivanju upravljačke tačke, u konkretnom rešenju *OR* gejt (kao što je prikazano na slici 25), koja je kontrolisana dodatnim signalom *scan_mode*. Njegov zadatak je da eliminiše funkciju *CG*-a u toku faze testiranja i time povrati upravljivost signala takta.



Slika 25• *Problemi testabilnosti*

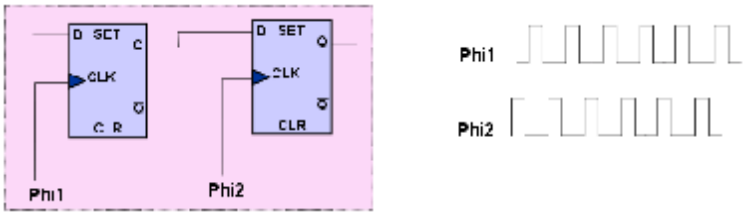
Problemi u CAD-u. Određivanje koje bi flip-flop-ove trebalo grupisati za *CG*, predstavlja izazov za projektanta. Za rešavanje ovog problema koriste se sledeće tehnike:

- Detekcija uslova *hold* [22]. Flip-flop-ovi kojima je isti *hold* uslov se izdvajaju i grupišu u jednu celinu radi pobude od strane istog *CG* kola. Ova metoda se ne mogu primeniti na flip-flop-ovima koji se dodatno upravljaju sa signalom dozvole.

- Detekcija redundantnog taktovanja [23]. Ova metoda je zasnovana na simulaciji. Flip-flop-ovi se grupišu u odnosu na trasiranje u simulaciji na takav način da *CG* kola budu zajednička. Ova metoda ne može biti automatizovana.

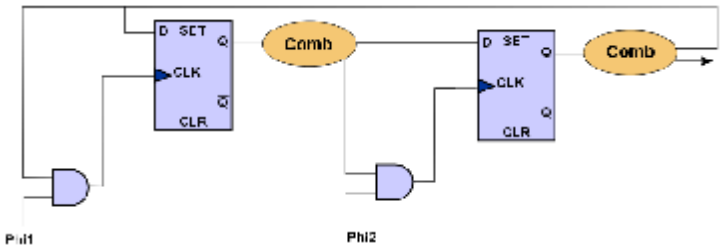
4.1.3.3 Dizajn zasnovan na lečevima

U nekim aplikacijama, dizajni zasnovani na lečevima se preferišu u odnosu na dizajne zasnovane na *D* flip-flop-ovima (*DFF*). Osnovni koncept je taj da se *DFF* može podeliti u dva leča a svaki od njih se može taktovati nezavisnim takt signalom. Dva taktna signala koja se međusobno ne preklapaju prikazana su na slici 26.



Slika 26• *Master-slave leč sa nepreklapajućim taktom*

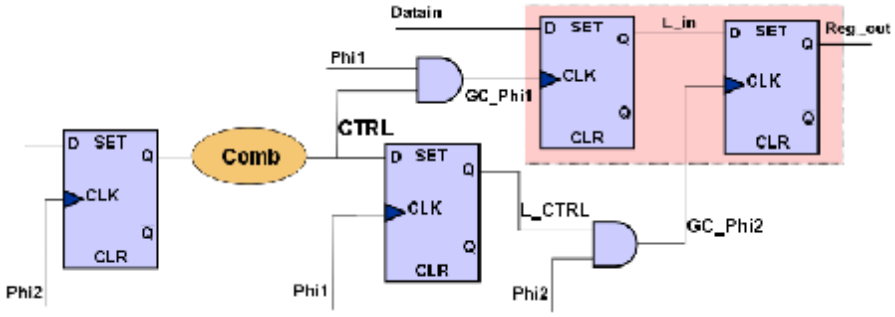
Između dva leča, sa ciljem da se kreira protočna staza podataka, ubacuje se kombinaciona mreža (slika 27). Glavna prednost ovog pristupa je ta što ova šema podržava veće košenje takta u odnosu na rešenje zasnovano na *DFF*-ma. Druga prednost se sastoji u skraćenju vremena za taktovanje što je prirodna posledica protočno organizovane staze podataka.



Slika 27• *Takt gejtovanje bazirano na lečevima*

CG-nje bazirano na lečevima se lako implementira. Na slici 27 prikazan je jednostavan i robustan način da se to uradi [24]. Standardno *AND* kolo se koristi za generisanje dozvole takta. Kod ove konfiguracije ne javljaju se gličevi, jer upravljački signal, generisan kada je *Phi1* na visoko, je stabilan i ostaje stabilan kada *Phi2* pređe u stanje visoko.

U slučaju registara, imajući u vidu činjenicu da upravljački signal dolazi bilo od leča koji se taktuje od strane *Phi2* ili od kombinacione funkcije čiji se ulazi taktuju od *Phi2*, neophodno je da se doda leč koji se taktuje od strane *Phi1* sa ciljem da se zakasni upravljački signal kao što je prikazano na slici 28 [25]. Naglasimo da *AND* kolo mora biti veoma oprezno projektovano i odabrano.



Slika 28• *Takt gejtovanje staze podataka bazirane na lečevima*

Da bi se zaštitili od optimizacije koju vrše sredstva za sintezu, *AND* kolo mora biti locirano u posebnom hijerarhiskom nivou, kome je dodeljen atribut *don't touch.* Kod koji je prikazan na slici 29 predstavlja *VHDL* opis leč-bazirane 32-bit-ne registarske banke kod koje se koristi *CG*-nje. Blok *CG* sadrži dva logička kola (leč i dvoulazno *AND* kolo) izdvojeni u okviru posebne hijerarhije. Ovom bloku se može dodeliti

atribut *don't touch*. Blok *RB* predstavlja registarsku banku, a *RGB* sadrži strukturni opis *CG*-ovane registarske banke.

```
-- U sledećem delu su dati VHDL kodovi: za registarsku banku,
-- kasnije kod za CG ćeliju i na kraju kompletno kolo sa slike 28;
-- Opis 32 bit-ne registarske banke sačinjene od leča
-- osetljivih na visok nivo;
library ieee;
use ieee. std_logic_1164.all;

entity RB is
      Port (
      GC_Phi1, GC_Phi2 : in std_logic;
      datain: in std_logic_vector(31 downto 0);
      Reg_out: out std_logic_vector(31 downto 0));
end RB;

-- Arhitektura Register_Bank je opisana pomoću dva
-- posebna procesa. Proces P0 je osetljiv na GC_Phi1 signal
-- koji dolazi od strane CG kola, a proces P1 je osetljiv
-- na signal GC_Phi2 signal koji takođe dolazi od strane CG kola;
architecture Register_Bank of RB is
signal L_In: std_logic_vector(31 downto 0);
begin
      P0: process (GC_Phi1,datain)
      begin
            if (GC_Phi1 = '1') then
                  L_In <= datain;
            end if;
      end process;

      P1: process (GC_Phi2, L_In)
      begin
            if (GC_Phi2 = '1') then
                  Reg_out <= L_In;
            end if;
      end process;
end Register_Bank;

-- Opis CG ćelije;
library ieee;
use ieee.std_logic_1164.all;

entity GC is
      port(
      CTRL, Phi1, Phi2: in std_logic;
      GC_Phi1, GC_Phi2: out std_logic);
end GC;

architecture Gated_Clock of GC is
signal L_CTRL: std_logic;
begin
      P0:process (Phi1, CTRL)
      begin
            If (Phi1 = '1') then
                  L_CTRL <= CTRL;
            end if;                  -- Opis Leča;
```

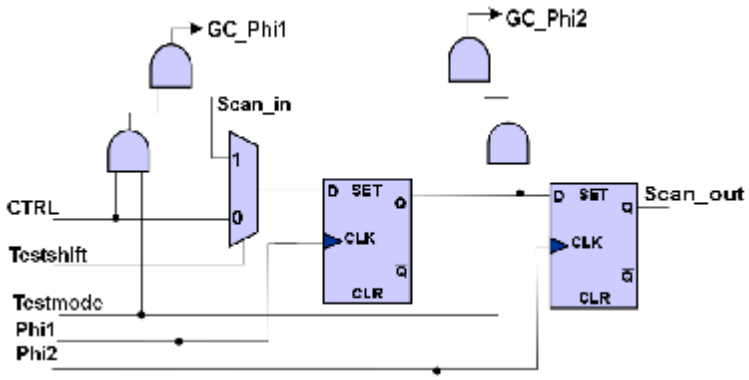
```
      end process;
      GC_Phi1 <= CTRL And Phi1;
      GC_Phi2 <= L_CTRL And Phi2;    -- Opis And kola;
end Gated_Clock;

-- Strukturni opis kompletnog kola;
library ieee;
use ieee.std_logic_1164.all;

-- Entitet GRB prihvata na ulazu dvofazne taktne signale Phi1 i Phi2,
-- kontrolni signal, ctrl, generisan od strane kombinacione logike,
-- 32 bit-nu ulaznu magistralu podataka datain, a generiše
--32 bit-ni izlaz Reg_out;
entity GRB is
      Port (
      Phi1, Phi2 : in std_logic;
      ctrl : in std_logic;
      datain: in std_logic_vector(31 downto 0);
      Reg_out: out std_logic_vector(31 downto 0));
end GRB;

-- U okviru ove arhitekture se prvo deklarišu komponente
-- koje se kasnije instanciraju;
architecture Gated_Clock_Register_Bank of GRB is
component RB
      port (
      Phi1, Phi2: in std_logic;
      datain: in std_logic_vector(31 downto 0);
      Reg_out: out std_logic_vector(31 downto 0));
end component;
component GC
      port (
      CTRL, Phi1, Phi2: in std_logic;
      CG_Phi1, CG_Phi2: out std_logic);
end component;
signal GC_Phi1, GC_Phi2: std_logic;

-- poomoćni signali dobiveni od strane CG kola, koji
-- taktuju lečeve;
begin
      RB_instance: RB port map(
                              GC_Phi1, GC_Phi2, datain, Reg_out);
      GC_instance: GC port map(
                              CTRL, Phi1, Phi2, GC_Phi1, GC_Phi2);
end Gated_Clock_Register_Bank;
```

Slika 29• *VHDL kôd 32 bit-ne registarske banke sa CG-om*

4.1.3.4 Zahtevi dizajna zasnovanog na lečevima

Jedan od dizajna koji se odnosi na leč zasnovanom *CG*-ovanju implementiran je kod procesora *ARM*. U konkretnom slučaju, sredstvo za sintezu pronalazi tajming petlje analiziranjem upravljačkih puteva. Sa slike 27 vidimo da signal za dozvolu rada takta,

kod svakog leča, zavisi od vrednosti izračunate od strane logike a zatim dovodi na ulaz drugih lečeva. Kao što smo pomenuli kod procesora *ARM* najmanje jedan od ovih puteva može biti prekinut koristeći atribute za zabranu dozvole rada (kao na primer: *set_disable_timing* za *Synopsys* dizajn kompajler) [24].



Slika 30• *Blok za za takt gejtovanje sa predviđenim testiranjem*

Drugi zahtevi u dizajnu se odnose na testabilnost. Slika 30 [25] prikazuje modifikacije u *CG* bloku u cilju poboljšanja testabilnosti dizajna zasnovanog na lečevima.

4.1.3.5 Kašnjenje takta

U cilju uštede snage, za pobudu više flip-flop-ova koriti se jedinstveno kolo, pod uslovom da je signal dozvole zajednički za sve registre. U slučaju kada kolo za taktovanje nema dovoljni *fanout* tada je njegov izlaz potrebno realizovati kao taktno stablo. Kada se između kola za taktovanje i registra koje on taktuje umetne stablo za taktovanje, taktni signal kod logike za taktovanje mora da stigne znatno pre taktnog signala na ulaze u registre a takođe i signal dozvole mora biti stabilan pre nego što takt stigne na ulaz *CG*-a. Ovo uvodi značajna vremenska ograničenja koja se odnose na signal dozvole, a koja moraju biti sagledana u toku procesa sinteze. U suprotnom, ovo rešenje rezultiraće vremenskim razlikama nakon sinteze taktnog stabla.

Jedan način za rešavanje problema se sastoji u specifikaciji kašnjenja takta kod *CG*-a na takav način da, u procesu sinteze, kašnjenje takta na izlazu *CG*-a bude manje u odnosu na kašnjenje takta na ulazu u registre. Razlika u kašnjenjima predstavlja razliku u kašnjenjima između *CG* kola i taktnog stabla i taktnog stabla tj. između *CG*-a i registra. Ovo iziskuje da sredstvo za sintezu obezbedi uslov da se signal dozvole

generiše na vreme. Problem sa ovim pristupom je taj što bi projektant trebalo da proceni razliku kašnjenja znatno pre koraka sinteze taktnog stabla. U principu projektant može koristiti konzervativni pristup (rešenje u najgorem sučaju) procene kašnjenja u toku sinteze taktnog stabla ili da svesno specificira kašnjenje ograničavajući *fanout* svake *CG* ćelije.

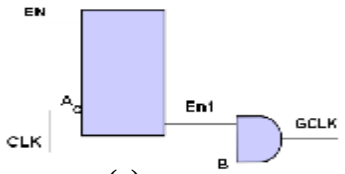
4.1.3.6 Efekti košenja takta

Još jedan problem kod leč zasnovanih arhitektura javlja se zbog košenja takta (*clock skew*) između leča i ulaza *AND* kola (signal *En1* na slici 31(a)). Košenje može dovesti do pojave gličeva na izlazu *CG* kola (signal *GCLK)*. Da bi kolo funkcionisalo ispravno košenje takta između leča i *AND* kola treba da bude manje od kašnjenja kroz leč. Slika 31(b) ilustruje slučaj kada takt dolazi ranije na leč. U ovom slučaju košenje između *AND* kola i leča treba da je manje od vremena postavljanja leča i ulazno-izlaznog kašnjenja leča, kako bi kolo funkcionisalo ispravno. Prema tome, vremenska razlika između leča i *AND* kola, C_s, treba biti pažljivo odabrana i treba da zadovolji sledeći uslov:

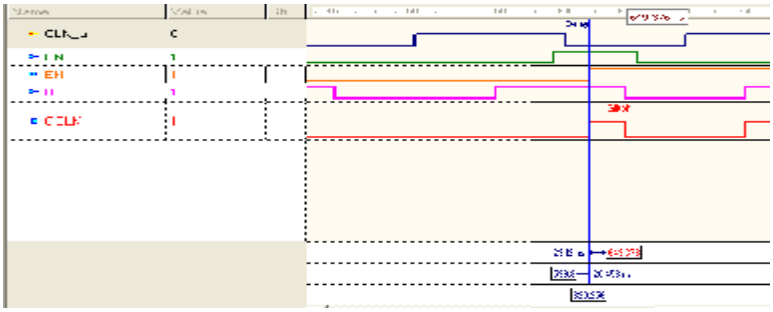
$$-(s + d_{in}) < C_s < d_{CLK} \qquad (22)$$

gde je, *s*- vreme postavljanja leča, d_{in}- ulazno/izlazno kašnjenje leča, C_s- razlika u vremenu pristizanja takta između leča i *AND* kola (vreme pristizanja takta na *AND* kolu – vreme pristizanja taktnog signala na leču), i d_{CLK} - kašnjenje koje unosi leč od pojave takta do generisanja izlaza.

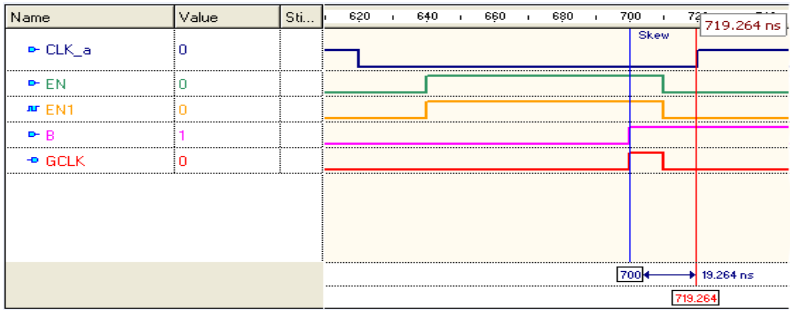
U zavisnosti od relativnog položaja leča i *AND* kola, pomenuti zahtevi mogu postaviti veoma stroga ograničenja koja se odnose na sintezu taktnog stabla.



(a)



(b)



(c)

Slika 31• *Košenje takta CG ćelije: (a)CG ćelija, (b)pozitivno košenje takta, (c) negativno košenje takta između leča i AND kola*

Najbolji način da se kontroliše relativni tajming između dva taktna signala je da se kompletna struktura smesti u jednoj ćeliji, nazvana integrisana *CG* ćelija (*ICG*). Ćelija, za potrebe *CG*-nja, treba da bude specijalno projektovana uz zadovoljenje svih ograničenja koja su ranije razmatrana. Ova ćelija se u principu ne može modelirati ni kao kombinaciona ni kao sekvencijalna, pa se zbog toga u *Liberty* bibliotečnom formatu koristi kao *state-table* model [18].

Još jedan način otklanjanja ovog problema je da se osigura da su leč i *AND* kolo blizu jedno drugom u toku faze razmeštanja kola, što postavlja stroga ograničenja na izvođenja kratkih veza između ovih kola. Ovo rešenje pojednostavljuje sintezu taktnog stabla, jer se smanjuje košenje takta između ovih signala u toku faze rutiranja signala.

4.1.3.7 Sinteza stabla takta

Kod automatizovanog *ASIC* dizajn okruženja taktni signal ostaje nepromenjen u toku sinteze, a sinteza taktnog stabla se izvodi kao jedan od poslednjih koraka u toku projektovanja, nakon faza razmeštanja i rutiranja. S obzirom da se manuelno izvedenim (na nivou modula) *CG*-om ugrađuje samo nekoliko kola za razvođenje

takta u dizajnu, sredstva za sintezu taktnog stabla mogu da budu efikasna samo u slučaju kad se sprovede nekoliko manuelnih intervencija. Kod masivnih *CG*-ova, sredstva za sintezu taktnog stabla moraju automatski da registruju prisustvo *CG*-ova na linijama za razvođenje takta. Zahtevi koji se odnose na sintezu stabla takta su:

- Optimizacija taktnog stabla u prisustvu logike.
- Podrška radu integrisanoj *CG* ćeliji kod taktnog stabla. Ovo je sekvencijalna ćelija ali nije krajnja tačka u taktnom stablu pa se zbog toga s njom mora upravljati na specijalan način.
- Podrška zahtevima koji se odnose na različite relativne latencije u različitim tačkama taktnog stabla.
- Stroga kontrola košenja takta između leča i *AND* kola za slučaj da se ne koristi integrisana *CG* ćelija.

4.1.3.8 Fizičko CG-ovanje

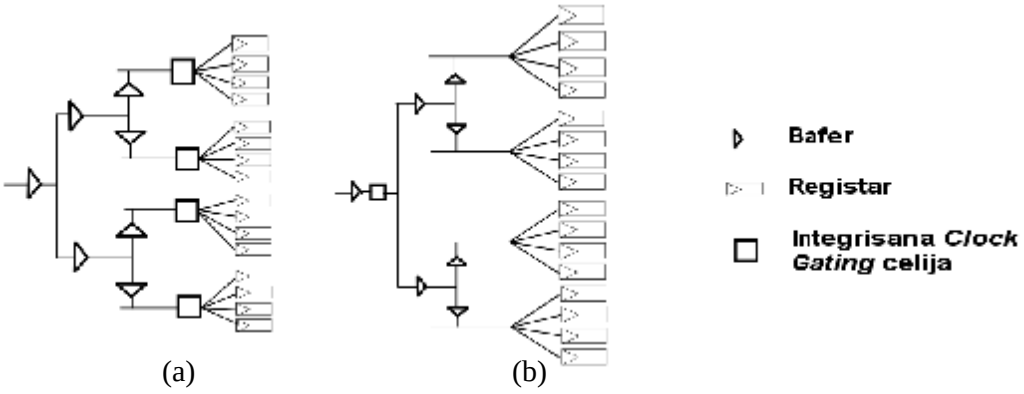
Fizičko *CG*-ovanje simultano uzima u obzir ranije pomenute faktore, latenciju i probleme koji se odnose na sintezu taktnog stabla.

Postoji čitav spektar pristupa koji uzimaju u obzir razmeštaj *CG* ćelija u taktnom stablu. Projektanti se često odlučuju da smeste takt gejtovane ćelije što je moguće bliže konačnom razmeštaju odgovarajućih registara kako je to prikazano na slici 32(a). Razmeštaj se može sprovesti u toku fizičke sinteze specificiranjem granice bliskosti *CG* ćelija i registara.

Jedna od prednosti ovog pristupa je ta da se lakše izvršava procena latencije *CG*-ova. Uticaj taktnog stabla je minimalan, zato što su *CG* ćelije smeštene što bliže registrima pa zbog toga ne postoji potreba za ubacivanje baferskih post *CG* ćelija.

Nedostatak ovog pristupa je taj što on dozvoljava taktovanje većem delu komutacija u kodnom stablu čak i u slučaju kada su razvođenja takata usmerena ka registrima koji treba da budu blokirani od strane *CG*-a. Da bi se uštedelo što je moguće više energije,

potrebno je da se *CG*-uje što više bafera u taktnom stablu. Ovo predstavlja težak problem projektantu, iz razloga što ne poznaje fizička ograničenja tajminga.



Slika 32• *Postavljanje takt-gejtovanih ćelija: (a) ćelije u blizini registara, (b) ćelije sa post-gate baferovanjem*

Kod *CG* sistema gde se vodi računa o fizičkom rasporedu, sinteza taktnog stabla treba da se sprovede zajedno sa razmeštajem i *CG*-njem sa ciljem da se optimizira razmeštaj i broj *CG* ćelija u taktnom stablu. Ovo se koristi da izbalansira kašnjenje *enable* signala, u odnosu na iznos uštađene potencijalne snage, smeštanjem *CG* ćelija što je moguće bliže korenu taktnog stabla (vidi sliku 32(b)).

Projektovanjem sistema koji ima pristup informaciji o fizičkom tajmingu koja se sprovodi umetanjem *CG* ćelija i sintezom taktnog stabla, moguće je poboljšati košenje takta i tajming, i smanjiti vršnu snagu. U opštem slučaju stroža ograničenja koja se odnose na košenje takta iziskuju da se veći broj aktivnosti obavi u užem vremenskom prozoru, a to sa druge strane ima za uticaj povećanje vršne snage. Ako su ovi događaji distribuirani za veći vremenski period, vršna snaga se može izbalansirati, čime se značajno smanjuje uticaj vršne snage na pad napona na vezama.

4.1.4 Konačni automati (*FSM*)

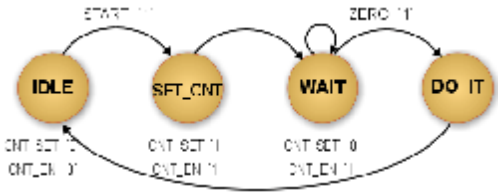
Konačni automati ***(Finite-State Machines- FSM-s)*** su standardni gradivni blokovi digitalnih sistema. Oni se masovno koriste za generisanje sekvenci signala, za provere ulaznih sekvenci i kao upravljački delovi staze podataka. Strukturu *FSM*-a čine registar stanja i dva logička bloka. Ulazni logički blok (*next-state*) određuje naredno

stanje u funkciji tekućeg stanja i novog stanja na ulazu. Izlazni logički blok generiše izlaze kao funkciju tekućeg stanja (za *Moor*-ov *FSM*). Potrošači snage mogu biti logički blokovi ili deo za distribuciju takta flip-flop-ovima koji pripadaju registru stanja. Ukazaćemo na različite tehnike za minimizaciju potrošnje snage, koristeći eksplicitno kodiranje stanja i tehniku *CG*-ovanja.

4.1.4.1 CG- ovani FSM

Osnovna ideja kod *CG*-ovanog *FSM*-a se sastoji u tome da nije korisno imati prekidačke aktivnosti u *next-state* logici ili u delu za distribuciju takta ako registar stanja uzorkuje isti vektor [26]. Analizirajmo jedan jednostavan, veoma čest, primer.

Na slici 33 prikazan je *FSM* koji interaguje sa *timer-counter* logikom u cilju implementacije veoma velikog kašnjenja, reda nekoliko hiljada taktnih ciklusa, pre nego što izvrši kompleksnu ali veoma kratku operaciju u *DO_IT* stanju. Moguće je koristiti *CG* tehnike radi zamrzavanja takta i ulaznih signala sve dok se marker *ZERO*, koji je izlaz *timer-countera* ne postavi u stanje "1".



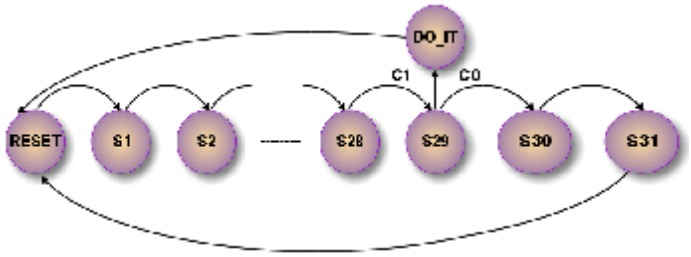
Slika 33• *Primer jednostavnog FSM-a*

Ova ideja je efikasna zato što *FSM* troši najviše vremena u stanju *WAIT*. Zamrzavanje takta može biti još efikasnije ako pretpostavimo da se *FSM* koristi za upravljanje veoma složenim stazama podataka čiji se izlazi ne koriste u stanju *WAIT*. Ako smo u stanju da *CG*-ujemo takt ili da maskiramo ulaze staze podataka tada se može ostvariti znatna ušteda dinamičke potrošnje tokom faze odbrojavanja *timer-countera*.

Za programere softvera važi pravilo 90/10. To znači da od ukupnog vremena izvršenja program 90% vremena troši na kôd petlji a 10% na ostali sekvencijalni kôd. U suštini, ovo pravilo važi i za *FSM*. Zadatak *RTL* dizajnera je da pokuša da izdvoji ove aktivnosti *FSM*-a, izoluje ih, a zatim zamrzne ostatak logike čiji je hardver obiman a koji za najveći deo vremena ne obavlja korisna izračunavanja.

4.1.4.2 Kodiranje stanja

Ovaj tip tehnike koristi kodiranje stanja sa ciljem da smanji aktivnost logike u ulaznim i izlaznim kombinacionim blokovima. Ideja je da se koristi kodiranje koje minimizuje promene iz jednog stanja u drugo, pod uslovom da je verovatnoća tih prelaza velika. Drugim rečima, mi bi trebalo, da sa velikom verovatnoćom, minimizujemo *Hamming* rastojanje prelaza. Ovo zahteva da sredstva za projektovanje treba da izračunaju verovatnoće prelaza na izlazu, na osnovu *FSM*-ovih ulaza, i odrede iznos verovatnoće za svaki prelaz. Napomenimo da je ova tehnika slična tehnici predikcije granjanja koja se koristi kod superskalarnih procesora. Treba naglasiti da je predikcija verovatnoće težak proces i ne može sa sigurnošću da garantuje potpuno pogađanje.

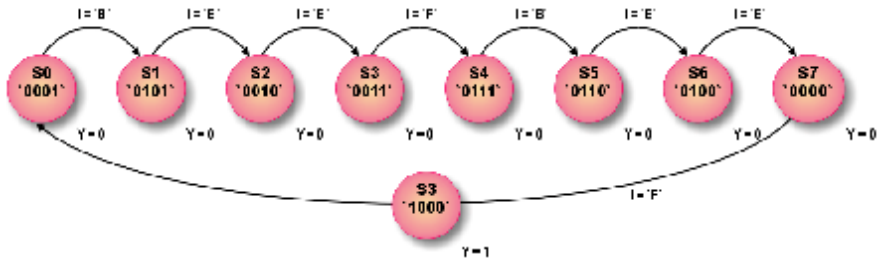


Slika 34• *Primer FSM-a sa Gray-ovim kodiranjem stanja*

U primeru prikazanom na slici 34 stanja od *RESET* do *S29* su sekvencijalno ulančana sa verovatnoćom prelaza 100%. Zbog toga *Gray*-ovo kodiranje predstavlja najbolje rešenje. Ako usvojimo da uslov *C0* ima znatno manju verovatnoću od *C1*, *Gray*-ovo kodiranje se neće koristiti za inkrementiranje iz stanja S29 u S30 i S29 u S31.

Ipak, ono što predstavlja dobitak u složenosti logike narednog stanja može da predstavlja gubitak u aktivnosti izlazne logike. Zbog toga projektant mora da nađe kompromis. Ako sada posmatramo redukciju snage u izlaznoj logici takođe možemo da izaberemo razumno kodiranje stanja. Standardni pristup se zasniva na *one hot* kodiranju sa ciljem da se optimizuje brzina, površina i potrošnja izlazne logike [27]. Ovaj pristup važi samo za male *FSM* (manje od 8-10 stanja) zbog velikog obima registra stanja i povećane kompleksnosti *next-state* logike. Za veće *FSM*-ove neophodno je obaviti analizu koja se razlikuje od slučaja do slučaja. Dobro rešenje je

da se grupišu stanja koja generišu iste izlaze i da se njima dodele kodovi sa minimalnim *Hamming*-ovim rastojanjem.



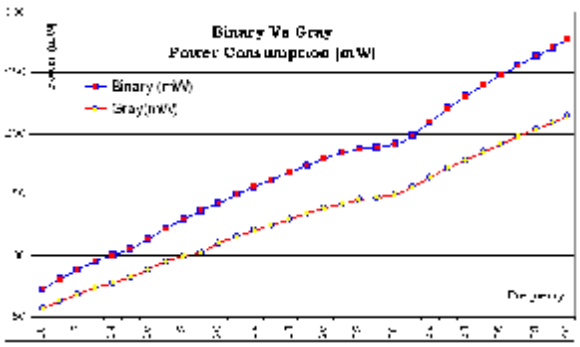
Slika 35• *Primer FSM-a sa zero-output logikom*

Jednostavan primer je dat na slici 35, gde je *FSM* korišćen da prepozna niz "*BEEFBEEF*" koji se dovodi na ulaz *I*, i da postavi marker *Y=1* ako je prepoznata kompletna sekvenca. Predloženim kodiranjem postiže se kako minimalna aktivnost logike *next-state*, zbog aktivnosti usled *Gray*-ovog kodiranja, tako i izostanka potrošnje u izlaznoj logici jer ortogonalno kodiranje definiše da *MSB* registra stanja bude istovremeno i marker *Y* [28].

Ilustracije radi, u Tabeli 2 je dato poređenje nekoliko vrsta kodiranja stanja. *Gray*-ovo kodiranje je najbolji izbor kako sa aspekta broja maksimalnih tranzicija tako i u pogledu ukupne potrošnje (slika 36).

Tabela 2• *Poređenje stanja za različite načine kodiranja*

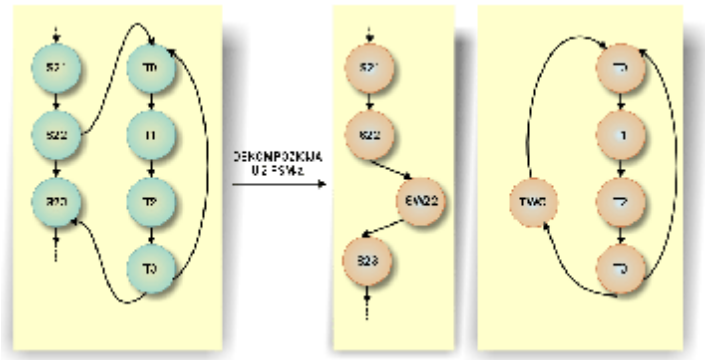
Stanje	One hot	Gray	Binary	LFSR
S0	00000001	000	000	111
S1	00000010	001	001	110
S2	00000100	011	010	100
S3	00001000	010	011	000
S4	00010000	110	100	001
S5	00100000	111	101	010
S6	01000000	101	110	101
S7	10000000	100	111	011
Ukupan broj tranzicija	18	8	14	15
Broj tranzicija po taktu	2	1	3	3
Učitavanje takta	8	3	3	3



Slika 36• *Razlika između Binarnog i Gray-ovog kodiranja po pitanju potrošnje*

4.1.4.3 Particija FSM-a

Obično *FSM* se može podeliti na manje delove. Ideja se sastoji u dekompoziciji velikog *FSM*-a na nekoliko manjih *FSM*-ova koji se karakterišu manjim registrima stanja i manjim kombinacionim blokovima. Samo aktivni *FSM* prihvata takt i komutirane ulaze. Ostali su statički i nemaju dinamičku potrošnju. Ukažimo na ovu tehniku pomoću veoma jednostavnog primera na slici 37 [29].



Slika 37• *Rasčlanjivanje FSM-a*

Neka je dat veliki *FSM* koji sadrži mali podprogram, koji se veoma često koristi kod realnih aplikacija. Moguće je podeliti obimni *FSM* na dva dela i izolovati potprogramsku petlju. Postupak ćemo sprovesti umetanjem stanja čekanja, *SW22* i *TW0,* izmedju ulaznih i izlaznih tačaka potprograma u oba *FSM*-a. Novi *FSM*-ovi rade međusobno isključivo; kada je jedan operativni drugi se nalazi u stanju *wait.* U ovom stanju, takt i ulazi mogu biti gejtovani sa ciljem da smanje dinamičku potrošnju (šta više, napon napajanja se može isključiti radi uštede gubitaka usled struje curenja).

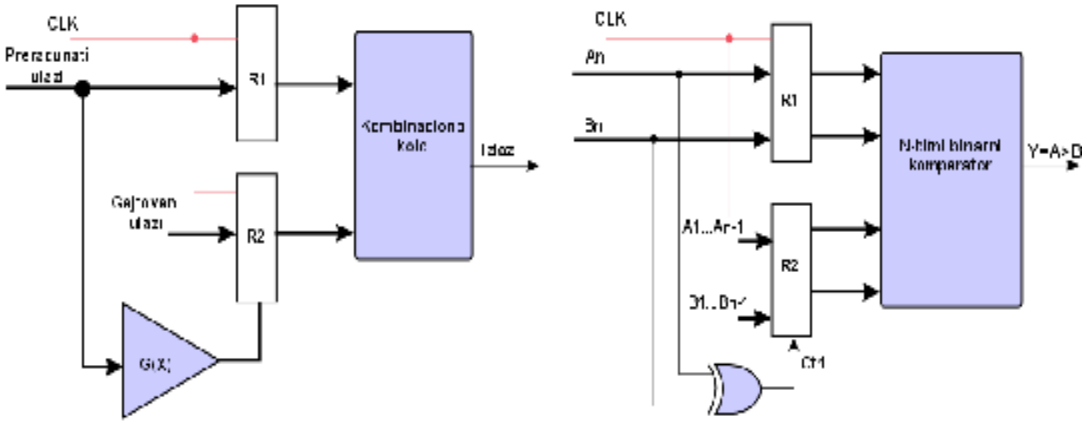
Ušteda u snazi je veća kada se izoluju veoma mali podskupovi stanja u kojima inicijalni *FSM* se zadržava najveći deo vremena.

4.1.5 Staze podataka

Značajni deo energije se može utrošiti u stazi podatka, usled komutatorskih aktivnosti koje ne doprinose fukcionalnosti kola. Da bi se smanjila odnosno redukovala potrošnja energije predložene su različite tehnike. Najznačajnije su: (a) logika za izračunavanje-unapred [30], (b) *guarded* evaluacija [31] i (c) *CG*-ovanje upravljačkog signala [32]. Ove tehnike se standardno koriste od strane projektanata kada se želi postići ušteda energije, a implementiraju se u ranijoj fazi projektovanja (na *RTL* nivou).

4.1.5.1 Tehnike izračunavanja-unapred

Princip izračunavanja-unapred se sastoji u identifikaciji logičkog uslova koji važi za neke od ulaza kombinacionih kola za kojie se izlaz neće menjati. Na slici 38(a) prikazan je opšti primer ovakvog kola.



(a) (b)

Slika 38• *Logika za preračunavanje: (a) blok šema, (b) aplikacija sa komparatorom*

Ulazi kombinacione logike *f(x)* se dele na ulaze koji se unapred izračunavaju i ulaze koji se gejtuju. Ako je izlaz *Y* zavistan od gejtovanih ulaza, tada funkcija *G(X)* generiše upravljački signal za registar *R2*, koji ne menja svoje izlaze. *Yeap* [33]

opisuje sistematsku metodu za određivanje funkcije *G(X)*, ali nažalost, za zadate ulaze rešenje koje se bazira na particiji nije univerzalno, što znači da projektant mora da odredi one ulaze za koje se dobija najbolji kompromis između potrošnje energije-performansi-površine. Veliki broj implementacionih alternativa logike za izračunavanje-unapred su date u *Practical Low-Power Digital VLSI Design* [33]. Na slici 38(b) prikazan je jednostavan i realan primer logike za izračunavanje-unapred. To je binarni komparator koji određuje da li je *A>B*. (Videti *VHDL* kod na slici 39 nakon ovog paragrafa). U ovom slučaju, uslov za izračunavanje-unapred je veoma lako odrediti. A_n i B_n su ulazi koji se izračunavaju-unapred, dok su *MS* bitovi i preostali bitovi gejtovani ulazi, pa se zbog toga izračunavanje-unapred izvodi jednostavnim *XOR* kolom. Očigledno je da projektant mora da zna neke detalje koji se odnose na statistiku ulaznih signala kako bi efikasno primenio tehnike za unapred-izračunavanje. U praksi izbor *R1*, *R2* i funkcija izračunavanje-unapred u velikoj meri zavise od iskustva projektanta.

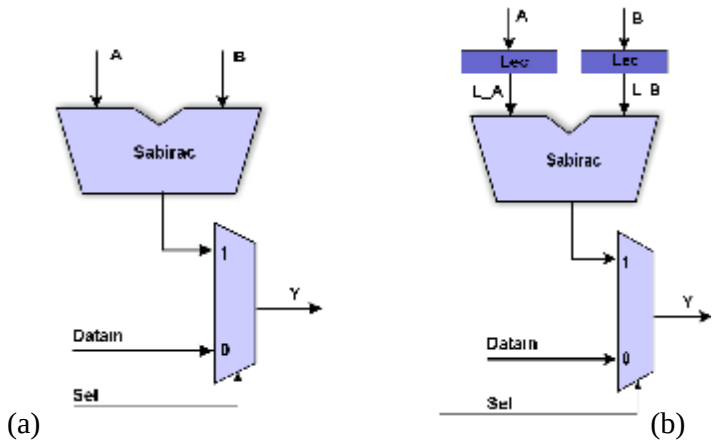
```
-- Kôd koji sledi opisuje binarni 32 bit-ni komparator
-- u koji je ugrađena logika za izračunavanje-unapred;
library ieee;
use ieee.std_logic_1164.all;

-- Deklaraciju entitety PC_Comp čine dva 32 bit-na ulaza A i B,
-- taktni ulaz i jednobitni izlaz Y;
entity PC_Comp is
      port (
      A,B: in Std_Logic_Vector(31 downto 0);
      Clk : in Std_Logic;
      Y : out Std_Logic);
end PC_Comp;

-- Arhitekturu b32Comp čine dva procesa
-- u okviru prvog procesa izdvajaju se bitovi najveće težine
-- radi preračunavanja, dok u okviru procesa P2
-- dozvoljavamo/zabranjujemo upis preostalih bitova podataka
-- A i B u registar R2;
architecture b32Comp of PC_comp is
signal Ctrl : std_logic;
signal A_R1, B_R1 : Std_Logic;
signal PC_A_R2, PC_B_R2 : Std_Logic_Vector(30 downto 0);
begin
-- izračunavanje-unapred bitova najveće težine;
      Ctrl <= A(31) Xor B(31);

-- Komparacija se obavlja nezavisno od procesa
-- i ako je A veće od B postavlja izlaz na visoko stanje;
Y <= '1' when ((A_R1 & PC_A_R2) > (B_R1 & PC_B_R2)) else '0';

      P1 : process (Clk)
```

```
      begin
            if (Clk'event and Clk ='1') then
                  A_R1 <= A(31);
                  B_R1 <= B(31);
            end if;
      end process;

      P2 : process (Clk)
      begin
            if (Clk'Event and Clk ='1') then
                  if (Ctrl = '0') then
                        PC_A_R2 <= A(30 downto 0);
                        PC_B_R2 <= B(30 downto 0);
                  end if;
            end if;
      end process;
end b32Comp;
```

Slika 39• *VHDL kôd komparatora sa logikom za izračunavanje-unapred*

4.1.5.2 Guarded evaluaciona tehnika

Ova tehnika je primenljiva kod ugrađenih kombinacionih blokova kod kojih se izlazi nalaze u *idle* stanju. Na svim ulazima ugrađenog bloka se ubacuju transparentni lečevi. Sa ciljem da se odredi *idle* uslov dodaje se upravljačka logika, koja se koristi da zabrani rad lečevima. Na slici 40 je prikazan jednostavan primer ove tehnike. Izlaz *ALU*-a se može, ali nemora, biti korišćen u zavisnosti od selektorskog ulaza multipleksera. Ako se ne koristi, lečevi čuvaju predhodne izlazne vrednosti *ALU*-a. Očigledno je da će, za magistrale velikog obima, ušteda u disipaciji biti značajna. Na slici 41 dat je *VHDL* opis kola sa slike 40 (b).



(a) (b)

Slika 40• *Primer staze podataka: (a) originalno kolo, (b) modifikovano kolo*

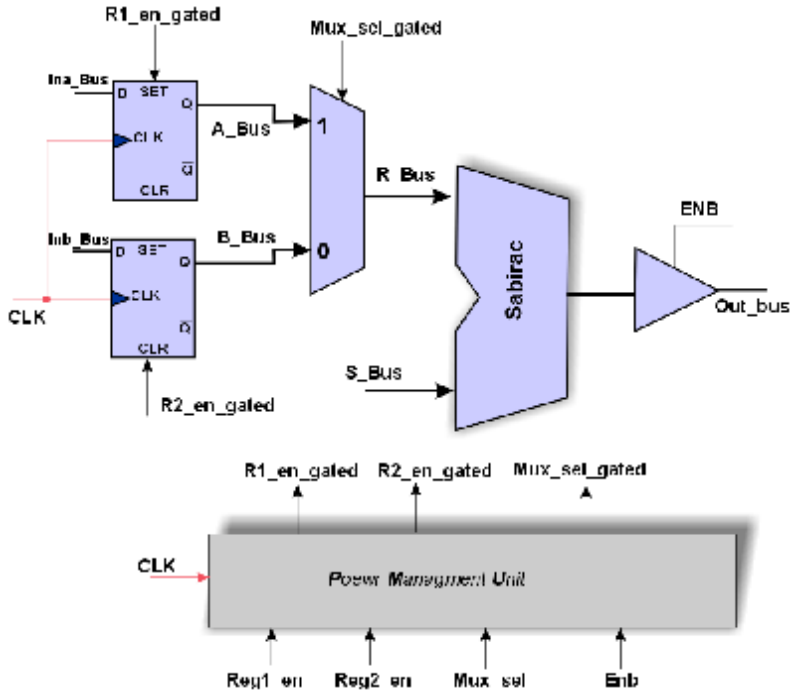
```
library ieee;
use ieee.std_logic_1164.all;

-- Stazu podataka čine: sabirač sa ulazima L_a i L_B,
-- dva 32 bit-na leča sa ulazima A i B i izlazima L_A i L_B,
-- multiplekser izlazom Y;
entity GE_Alu is
      port (
      A,B: in Std_Logic_Vector(31 downto 0);
      Datain: in Std_Logic_Vector(31 downto 0);
      Sel : in Std_Logic;
      Y : out Std_Logic_vector (31 downto 0));
end GE_Alu;

-- Arhitekturu Garded_evaluation_Alu čini jedan proces osetljiv na
signale Sel, A i B
-- Signal Sel istovremeno selektuje multiplekser i loaduje
-- A i B u leče ve;
architecture Garded_evaluation_Alu of GE_Alu is
signal L_A, L_B : Std_logic_vector(31 downto 0);
begin
      process (Sel, A, B)
      begin
            if Sel='1' then
                  L_A<=A;
                  L_B<=B;
            end if;
      end process;
Y<=(L_A + L_B) when (Sel='1') else Datain;
end Garded_evaluation_Alu;
```

Slika 41• *VHDL kôd staze podataka sa slike 40(b)*

4.1.5.3 Tehnike CG-ovanja upravljačkog signala

Sve predhodno predstavljene tehnike imaju za cilj da redukuju komutatorske aktivnosti u delu staze podataka. Tehnike gejtovanja upravljačkog signala polaze od analiza, koje nude finu granularnost, sa ciljem da se smanje aktivnosti na magistralama staze podataka. Metoda je zasnovana na posmatranju koncepta *don't care* (***O**bservability **D**on't care **C**oncept- ODC*) [34], kako bi se detektovao uslov kada se magistrala ne koristi a time i zaustavila propagacija komutatorskih aktivnosti kroz modul(e) koji pobuđuju magistralu. *Kapadia* [32] je dao dobru formulaciju ovog koncepta koji se tiče gejtovanja upravljačkih signala u stazi podataka. Na slici 42 prikazan je primer staze podataka. Kada *Enb* nije aktivan, *Mux_sel*, *Reg1_en* i *Reg2_en* mogu biti gejtovani, što rezultuje smanjenjem komutatorskih aktivnosti do

100% na magistralama *R_Bus*, *A_Bus* i *B_Bus*. Kada je *Mux_sel* aktivan, tada u zavisnosti od vrednosti *Mux_sel* mogu biti gejtovani *Reg1_en* ili *Reg2_en*.



Slika 42• *Control-signal gating tehnika*

Uslovi gejtovanja staze podataka sa slike 42 dati su sledećim jednačinama:

$$R1\_en\_gated = reg1\_en \text{ AND } (\text{NOT}(mux\_sel \text{ OR } (\text{NOT } enb)))_{@(T+1)} \quad (23)$$

$$R2\_en\_gated = reg2\_en \text{ AND } (\text{NOT}(\text{NOT } mux\_sel \text{ OR NOT } enb)))_{@(T+1)} \quad (24)$$

$$(mux\_sel\_gated)_{@T} = (mux\_sel\_gated)_{@(T+1)} \text{ if } ((\text{NOT } enb)_{@(T+1)} == True) \quad (25)$$

Sufiks *@T* označava vrednost promenljive ili funkcije u tekućem taktnom intervalu, *@T-1* je vrednost jedan takt ranije, a *@T+1* jedan takt kasnije.

Ove jednačine se mogu implementirati u ***P**ower **M**anagment **U**nit*-u (*PMU*) kako je to prikazano na slici 42. *PMU* generiše sve upravljačke signale za stazu podataka. *VHDL* opis *PMU*-a je oblika:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

-- PMU čine ulazi: Clk- takt, Enb- dozvola rada, Mux_sel selektorski
```

```
-- ulaz multipleksera, Reg1_en i Reg2_en kao signale dozvole
-- rada registara; Na izlazu PMU generiše kontrolne signale za
-- stazu podataka;
entity PMU is
      port (
      Reg1_en, Reg2_en, Mux_sel, Enb,Clk : in Std_Logic;
      R1_en_gated, R2_en_gated, Mux_sel_gated : out Std_Logic);
end PMU;

-- Procesi R1EG i R2EG u okviru arhitekture Power_Management_Unit
-- generišu signale dozvole rada registara R1 i R2 respektivno
-- Proces MSG upravlja radom multipleksera;
architecture Power_Management_Unit of PMU is
signal Enb_int, R1_en_tmp, R2_en_tmp : Std_logic;
begin
      R1EG : Process (Clk)
      begin
            if (Clk'Event and Clk='1') then
                  R1_en_tmp<=Not(Mux_sel Or (Not Enb));
                  -- formula po kojoj se upravlja radom R1;
            end if;
            R1_en_gated<=R1_en_tmp And reg1_en;
      end process;

      R2EG : Process (Clk)
      begin
            if (Clk'Event and Clk='1') then
                  R2_en_tmp<=Not(Not(Mux_sel)Or(Not Enb));
                  -- formula po kojoj se upravlja radom R2;
            end if;
            R2_en_gated<= R2_en_tmp And reg2_en;
      end process;

      MSG : Process (Clk)
      begin
            if (Clk'Event and Clk='1') then
                  Enb_int<=Not Enb;
                  if (Enb_int ='0') then
                        Mux_sel_gated<=Mux_sel;
                  end if;
            end if;
      end process;
end  Power_Management_Unit;
```

Slika 43• *VHDL kôd PMU-a*

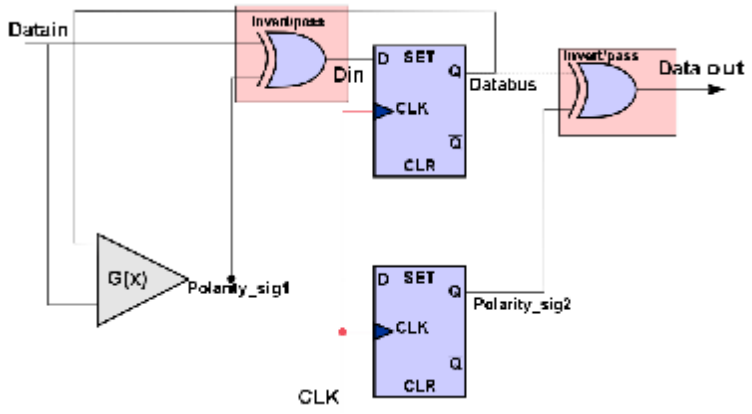
4.1.6 Kodiranje magistrale

Složeni sistemi obično sadrže obimne i dugačke magistrale, koje troše veliki iznos energije uglavnom zbog velikih parazitnih kapacitivnosti i značajnih komutatorskih aktivnosti. Za rešavanje ovog problema predložene su brojne tehnike koje se odnose

na različite nivoe dizajna: *low-swing-bus* [35], *charge-recycling-bus* [36], protočnost u radu magistrale [37], multipleksiranje magistrale, i tehnike kodiranja podataka koji se prenose po magistrali. Najprikladnije za *VHDL* kodiranje sa aspekta male potrošnje su tehnike za kodiranje podataka koji se prenose magistralom. Prema tome, skocentrisaćemo se na jednu od njih a u nastavku ćemo kratko objasniti ostale tehnike za kodiranje podtaka na magistrali.

4.1.6.1 Kodiranje sa inverzijom

Kodiranje sa inverzijom (*bus invert encoding*) je pogodno za aplikaciju kada se prenose paralelni i sinhroni signali kao što je slučaj kod internih magistrala kod savremenih *SoC* arhitektura [38]. Ideja je jednostavna: pre slanja podatka, predajnik poredi tekuću vrednost sa predhodnom i odlučuje da li da je preda ili da pošalje invertovanu vrednost zajedno sa polaritetom signala.



Slika 44• *Šema invertovanog kodiranja magistrale*

Banka *XOR* gejtova na predajnoj i prijemnoj strani invertuje podatke na magistrali, ako je potrebno. Na slici 44 prikazana je arhitektura bloka za kodiranje sa inverzijom, a u nastavku je dat odgovarajući *VHDL* kod.

```
-- Kôd odgovara slici 44 i predstavlja jedan način
-- kodiranja magistrale;
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

-- Ulazi su Clk i 32 bit-ni Datain, a izlaz
```

```
-- je kodirani 32 bit-ni Dataout;
entity BIE is
      port (
      Datain : in Std_Logic_vector(31 downto 0);
      Dataout : out Std_Logic_vector(31 downto 0);
      Clk : in  Std_logic);
end BIE;

architecture Bus_Invert_Encoding of BIE is
signal Din, Databus : Std_logic_Vector(31 downto 0);
signal  Polarity_sig1, Polarity_sig2 : Boolean;
begin
      Polarity_sig1<= Polarity_Gen(Datain, Databus);
   --Polarity_Gen je procedura koja je istinita(TRUE)
   --kada je vrednost na magistrali takođe istinita;
      Process (Clk)
      begin
            if Clk'Event and Clk='1' then
                  if Polarity_sig1(Datain, Databus) then
                        Databus<=Not Datain;
                        -- invertovanje podataka sa ulaza kada je
                        -- ispunjen predhodni uslov;
                  else
                        Databus<=Datain;
                  end if;
                  Polarity_sig2<=Polarity_sig1;
            end if;
      end process;
            Dataout<=Polarity_sig2 Xor Databus;
end Bus_Invert_Encoding;
```

Slika 45• *VHDL kôd kola za invertovano kodiranje magistrale*

4.1.6.2 Ostale tehnike kodiranja podataka na magistrali

Kod tehnike kodiranja sa inverzijom određuje se *Hamming*-ova distanca između dva uzastopna koda pa se zatim tom kodu u toku predaje pridružuje signala polariteta, koji utiče na strukturu interfejsa između predajnika i prijemnika. Jedan drugi pristup se bazira na predpostavci da ako je *MSB =1*, tada se predaje invertovani kod. *MSB=1* se koristi kao informacija o polaritetu. Ova tehnika je efiksna kada vektor koga treba preneti koristi prezentaciju elemenata u dvojičnom komplementu pri čemu *MSB* predstavlja bit znaka.

Druga standardna metoda se bazira na sledećem principu: vrednost koja se predaje po magistrali je inkrement u odnosu na predhodnu vrednost, ovo posebno važi za adresnu magistralu. Zbog toga, linije mogu ostati ne promenjene (nema potrošnje snage) sve

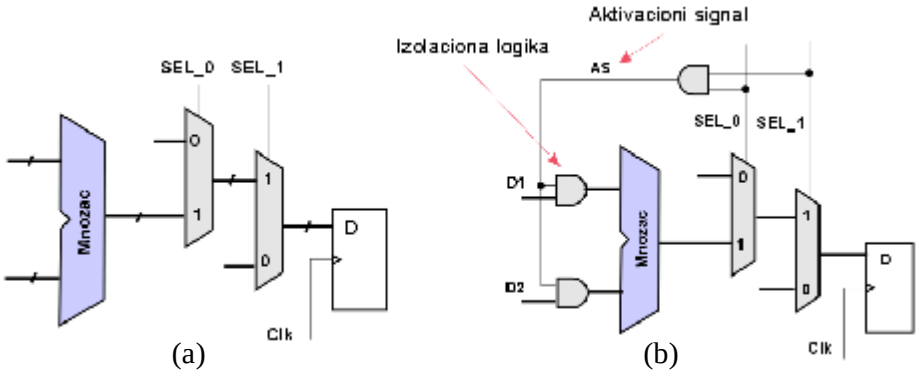
dok su kodovi uzastopni, pri čemu se na prijemnoj strani mora ugraditi posebna logika za vraćanje signala u prvobitno stanje.

Konačno, moguće je koristiti znanje o skupu simbola ili verovatnoći pojavljivanja simbola pa se to znanje može iskoristiti za kodiranje kako bi se smanjile komutatorske aktivnosti po magistrali. Na primer, ako se sekvenca 0101=>1010 pojavljuje 90% od ukupnog vremena, moguće je uštedeti snagu ako se kod 0101=>0000 i 1010=>0001.

4.1.7 Izolacija operanda

Iako se *CG*-ovanjem štedi energija kod taktovanih ili sekvencijalnih delova dizajna, uštede energije u kombinacionim delovima nisu moguće ovom metodom. Na *RTL* nivou najpopularnija tehnika za redukciju energije u kombinacionom delu, predstavlja izolacija operanda [39]. Slično kao i *CG*, osnovni koncept u ovom slučaju je da se logički blokovi isključe kada ne obavljaju korisno izračunavanje.

Isključivanja kombinacionog bloka obuhvataju aktivnosti u bloku ali ne dozvoljavaju da se ulazi prihvataju u taktnim ciklusima u toku kojih se izlazi bloka ne koriste. Osnovni koncept je prikazan na slici 46(a). Naglasimo da se izlaz množača koristi samo kada su kontrolni signali multipleksera, *SEL_0* i *SEL_1* na visokom nivou. U ciklusima kada je bilo koji od kontrolnih signala na niskom nivou, ako se ulazi množača menjaju, množač će obaviti izračunavanje, ali se njegov rezultat neće koristiti. Potrošena snaga može biti značajna ako ovi se ovi pasivni ciklusi javljaju u dugim periodima.



Slika 46• *Izolacija Operanda: (a)originalno kolo, (b)nakon izolacije operanda*

Na slici 46(b) ilustrovana je tehnika izolacije operanda koja se primenjuje na kolo množača. Prvo, aktivacioni signal, *AS*, se generiše sa ciljem da detektuje cikluse kada je množač neaktivan. Aktivacioni signal je na visokom nivou u aktivnim taktnim ciklusima pa se tada izlaz iz množača koristi, a kada se ne koristi izlaz množača *AS* je na niskom nivou. Ovaj signal se koriti da izoluje množač putem zamrzavanja njegovih ulaza u toku pasivnih ciklusa koristeći pri tome skup kola koja se nazivaju izolaciona logika. Slika 46(b) ilustruje korišćenje *AND* kola kao izolacione logike, ali takođe, za realizaciju izolacione logike, mogu se koristiti *OR* kola ili lečevi. Korišćenjem *AND/OR* kola izbegava se uvođenje novih sekvencijalnih elemenata i smanjuje uticaj na ostatak odvijanja aktivnosti. Pored toga, naglasimo da se pokazalo da je implementacija *AND/OR* kola jeftinija i da ista daju bolje rezultate sa aspekta uštede energije.

Izolacijom operanda štedi se na potrošnji putem redukcije komutacija usled ulaznih izolovanih operanada, ali se pri tome usložnjava tajming, povećavaju površina i potrošnja zbog ugradnje dodatnih kola koja se koriste za generisanje aktivacionog signala i sinteze izolacione logike. Ovaj *overhead* mora biti pažljivo procenjen sa aspekta dobitka uštede u potrošnji u odnosu na povećnja površine i propagacionog kašnjenja.

4.1.8 Optimizacija na logičkom nivou

Optimizacije kola na logičkom nivou redukuju potrošnju kola putem njihovih transformisanja u različite ali funkcionalno ekvivalentne implementacije. Ove transformacije uključuju korišćenje tehnika na *RTL* nivou, kao i tehnika na nivou gejtova. Zbog velikog broja sredstava za logičku sintezu, koja se danas nude na svetskom tržištu ove tehnike predstavljaju dobre kandidate za automatsku optimizaciju.

Cilj ovih tehnika je da smanje bilo dinamičku potrošnju bilo disipaciju usled kratkog spoja. S obzirom da se obe komponente javljaju u toku komutacije, ove tehnike se oslanjaju na dobijene statistike o promeni stanja na ulaznim pinovima za sve ćelije u kolu. Da bi se dobila ova informacija i iskoristila od strane sredstava za optimizaciju,

korisnik može da: (a) simulira rad kola kako bi odredio komutatorske aktivnosti na ulaznim pinovima svih ćelija; (b) proceni statističke aktivnosti na svim primarnim ulazima. U drugom slučaju sredstva za optimizaciju moraju da propagiraju promene aktivnosti od primarnih ulaza ka internim ulazima ćelija koristeći: (*i*) *BDD* tehniku (***B**inary **D**ecision **D**iagram*) koja se bazira na tehnikama verovatnoće prostiranja signala [40], (*ii*) neke interne simulacije.

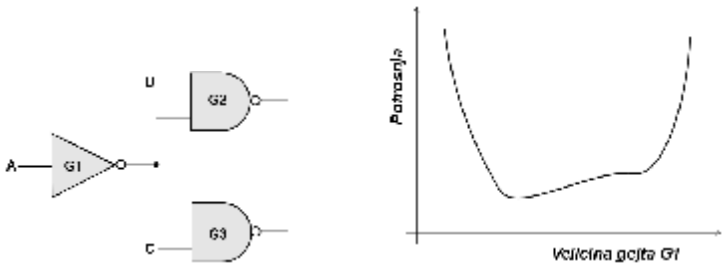
Ove tehnike, takođe, zahtevaju procenu vrednosti parazitnih kapacitivnosti *net*-ova i *I/O* pinova ćelija integrisanog kola, ove vrednosti se dobijaju iz bibliotečkih modela ćelija, ožičavanja, itd.

4.1.8.1 Širina kanala i baferovanje

Širina kanala kola utiče na potrošnju na sledeće načine:

- Velike ćelije doprinose većim kapacitivnostima čime se povećava ukupna dinamička disipacija.
- Za dato prelazno-ulazno-vreme, gejt čija je širina kanala veća ima veću struju kratkog spoja.
- Izlaz kola koje karakteriše veća opteretljivost ima oštriji nagib u odnosu na kolo koje ima manju opteretljivost, čime se direktno smanjuju dinamičke struje kratkog spoja u *fanout* gejtovima.

Baferovanje ima sličan efekat kao i širina kanala kola, jer ugradnja bafera doprinosi povećanju kako kapacitivnosti tako i struja kratkog spoja (to je negativan efekat), ali sa druge strane poboljšava nagib izlaznih signala (pozitivan efekat).



Slika 47• *Širina kanala kola i potrošnja snage*

Ovi koncepti se mogu jasnije ilustrovati koristeći primere. I pored toga što se demonstraciona ideja zasniva na korišćenju promene širine kanala, poenta je da se ona može primeniti i na baferovanje. Razmotrimo kolo sa slike 47, kod koje gejt G1 pobuđuje dva gejta *G2* i *G3*. Usvojimo da su ulazna prelazna vremena za *G1, G2* i *G3* data sa t_i, t_o i t_o. Širine kanala gejtova su W_1,W_2 i W_3, a odgovarajuće frekvencije rada su f_1,f_2 i f_3 respektivno. Ulazna frekvencija za *G1* je f_i, a ulazne i izlazne kapacitivnosti su C_i i C_o, respektivno. Koristeći poznate jednačine koje se odnose na potrošnju imamo:

$$P = C_o V_{DD}^2 f_1 + C_i V_{DD}^2 f_i + k\left(W_1 t_i f_1 + W_2 t_o f_2 + W_3 t_0 f_3\right) \qquad (26)$$

Prvi član se odnosi na potrošnju usled komutacije na kapacitivnosti C_o. Ako usvojimo da je *source/drain* kapacitivnost kola C_o mala, ovaj član biće takoreći konstantan i sporo će rasti sa povećanjem širine kanala gejta *G1*. Drugi član se odnosi na komutiranu snagu na ulazu gejta *G1* i linearno se povećava sa povećanjem širine kanala *G1*. Zadnja tri člana se odnose na disipaciju usled struja kratkog spoja i odnose se na sva tri gejta. Kako se širina kanala *G1* povećava, povećava se i struja kratkog spoja *G1*, jer se povećava W_1, dok se struja kratkog spoja kola *G2* i *G3* smanjuje zbog redukcije prelaznih vremena ulaznih signala. Zbog toga, sa povećanjem širine kanala gejta *G1*, disipacija se prvo smanjuje a zatim počinje da raste, pokazujući jasan minimum kao i mogućnost za optimizaciju (slika 47) [41].

Promena širine kanala i baferovanje se takođe koriste za balansiranje puteva u stazi podataka. Različita vremena pristizanja signala na različitim ulazima gejtova prouzrokuju nepoželjne komutacije ili pojavu gličeva na izlazu, što uzrokuje dodatnu disipaciju. Ove tehnike se mogu koristiti za izjednačavanje vremena pristizanja signala na ulazima datih gejtova, čime se redukuju gličevi.

4.1.8.2 Tehnološko mapiranje

Tehnološko mapiranje podrazumeva optimalno preslikavanje logičkog bloka koristeći ćelije iz zadate biblioteke. Tehnološko mapiranje koje se odnosi na *low power* se zasniva na sledećim činjenicama:

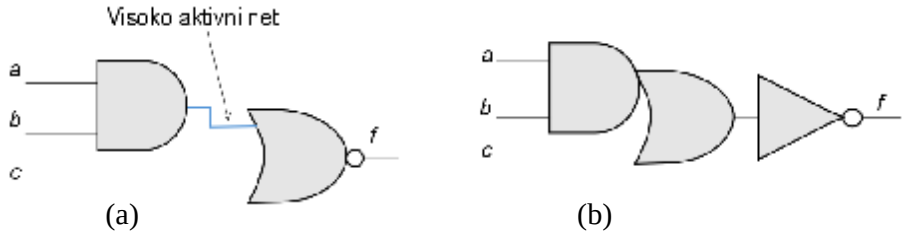
- Internii čvorovi bibliotečkih ćelija obično imaju manje kapacitivnosti od spoljašnih čvorova.

- Širina kanala gejta ima veliki uticaj na potrošnju.

- Preslikavanje koje rezultira najmanjoj potrošnji ne podudara se sa mapiranjem koje se odnosi na minimalno kašnjenje ili mapiranje u cilju ostvarivanja minimalne površine [42,43].

Prvo zapažanje ukazuje da je bolje da se čvorovi sa većom komutatorskom aktivnošću u dizajnu preslikaju u interne čvorove. Na ovaj način se smanjuje potrošnja jer se redukuje kapacitivnost najaktivnijih čvorova. Drugo zapažanje ukazuje da proces određivanja širine kanala ćelije mora da napravi kompromis između kapacitivnosti drajvovanja i potrošnje ćelija. Treća opservacija ukazuje da potrošnja eksplicitno predstavlja deo cene dizajna kojom se teži da se ostvari tehnološko rešenje preslikavanja osetljivo sa aspekta potrošnje.

Dve implementacije jednostavnog *AOI* (***A**nd-**O**r-**I**nvert*) kola prikazane su na slici 48. Naglasimo da je *net* $X=a\cdot b$ veoma aktivan. Kolo na slici 48(b) implementira aktivni *net X* kao interni *net* i zbog toga je disipacija manja.



Slika 48• *Tehnološko mapiranje za low power: (a)originalno kolo, (b) mapirano kolo*

4.1.8.3 Faza dodeljivanja

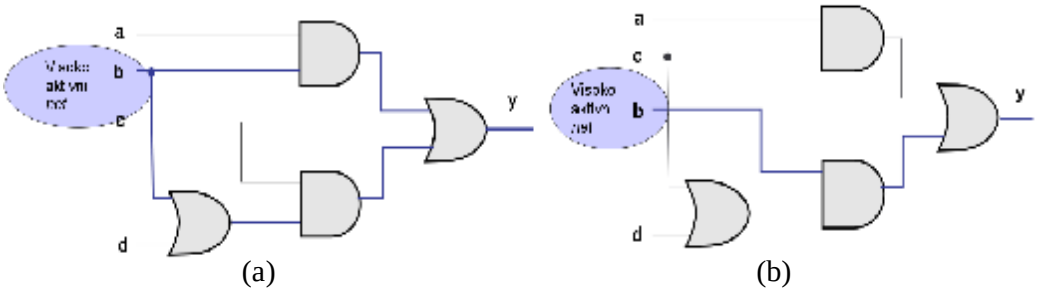
Faza dodeljivanja invertuje ulaze operacije, i istovremeno, invertuje izlaz. Ovom transformacijom štedi se na potrošnji na sledeće načine: Prvo, pošto se ovom transformacijom dodaju invertori u mreži u kojoj predhodno nije bilo invertora, ona pruža mogućnost za nekoliko drugih transformacija. Dva invertora koji su jedan do drugog mogu biti spojeni i uklonjeni, a invertor na izlazu gejta se može pridodati gejtu korišćenjem složenih kola iz biblioteke. Kao drugo, ovom tehnikom se mogu ukloniti

invertori iz visokoaktivnih čvorova i premestiti u čvorove mreže gde postoje niže aktivnosti.

4.1.8.4 Algebarske transformacije

Algebarske transformacije koriste algebarske osobine kako bi se dobile ekvivalentne implementacije datog kola u cilju smanjenja potrošnje. Osobine koje se najviše koriste za redukciju potrošnje podrazumevaju korišćenje zakona o komutativnosti, asocijativnosti i distributivnosti.

Komutativnost se koristi kod transformacije nazvane zamena mesta pinova. Na gejt nivou, veliki broj *Boole*-ovih operacija, *AND, OR, NAND, NOR* i *XOR* su komutativne, jer njihovi ulazi mogu promeniti mesta bez uticaja na funkcionalnost. U zavisnosti od ulazne kapacitivnosti i brzine komutiranja, ulazni pinovi kola mogu uzajamno promeniti mesta, povezujući pin sa nižom ulaznom kapacitivnošću sa većom brzinom komutiranja čime se u značajnoj meri smanjuje potrošnja. Iste tehnike se mogu koristiti na *RTL* nivou pri čemu se manipuliše zamenom mesta blokova kao što su sabirači i množači. Uprkos direktnom uticaju smanjenja potrošnje, ova tehnika generiše veći broj mogućnosti za primenu drugih tehnika i primenu raznih tipova algoritama.



(a) (b)

Slika 49• *Dva načina realizacije funkcije ab+bc+cd*

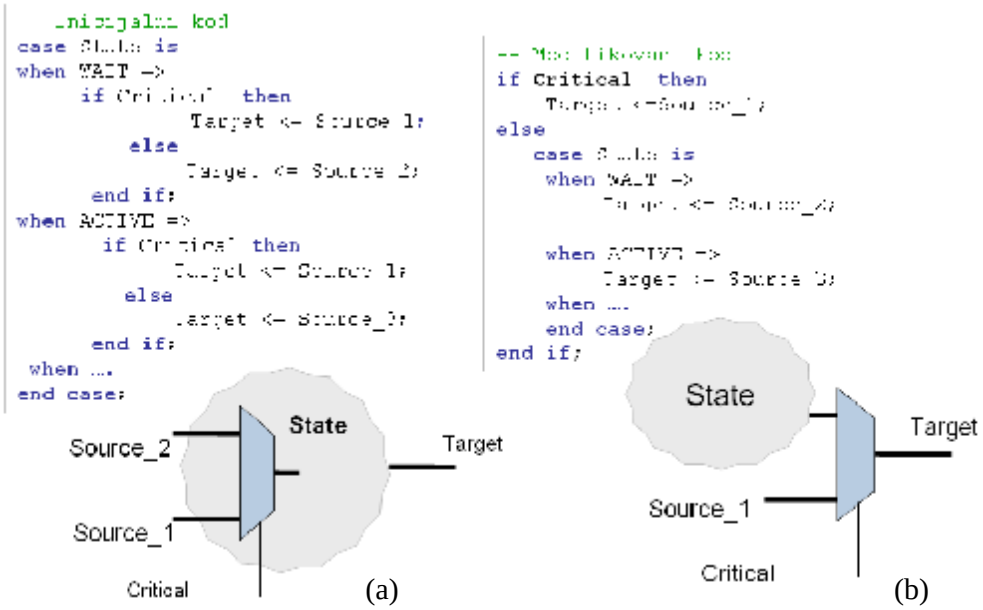
Asocijativne i distributivne osobine operacija na nivou gejta kakve su *AND*, *OR* i *XOR*, kao i *RTL* operatori, kakvi su sabirači i množači, koriste se u transformaciji za postizanje male potrošnje koja se naziva faktorisanje. Faktorisanje se bazira na ideji da se disipirana snaga od strane operacije i aktivnost na njenim izlazima zavisi od aktivnosti na njenim ulazima. Zbog toga, disipacija snage određena *net*-om zavisi od broja operacija na koji se ta aktivnost odnosi. Cilj faktorisanja je da se redukuje broj

logičkih nivoa koji se odnose na visokoaktivne *net*-ove. Ovo je ilustrovano na slici 49 koja prezentuje dve implementacije funkcije *ab+bc+cd*. Naglasimo da ulaz *b* ima najvišu aktivnost. Podebljane linije na slici identifikuju visokoaktivne *net*-ove. U ovom slučaju implementacija na slici 49(b) troši manje energije jer se signal *b* prostire samo kroz jedan gejt.

4.1.9 Dodatne napomene u vezi *VHDL* projektovanja

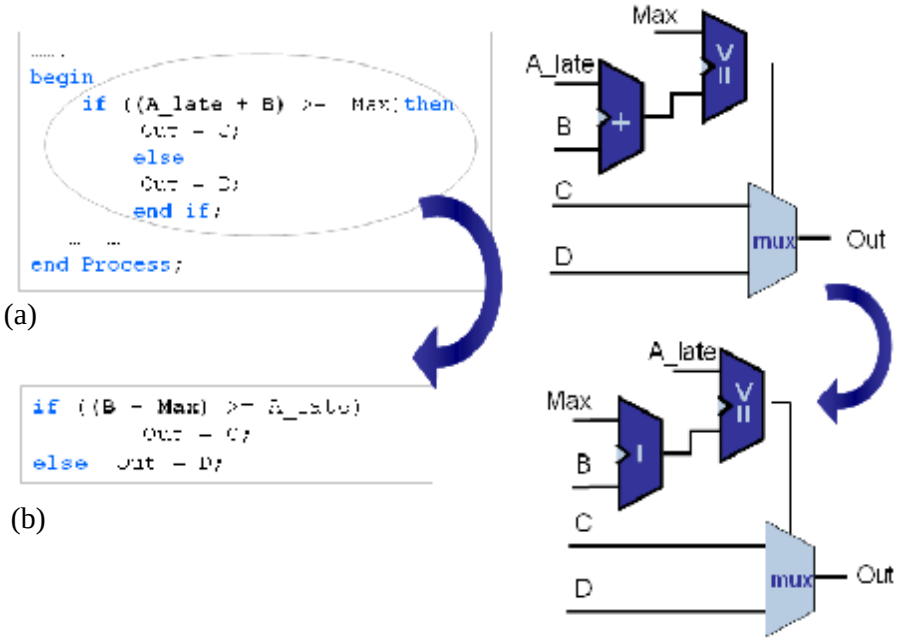
Iz dosad navedenog se može zaključiti da je od izuzetne važnosti, za *low power* dizajn, način formiranja *VHDL* koda. U tom cilju sagledajmo sledeće preporuke [44].

Primer1. **Definicija stanja automata.** U fazi projektovanja mora se voditi računa o prioritetu i redosledu pristizanja pobudnih signala. Sa tačke gledišta složenosti logike *FSM*-a, na slici 50 prikazan je jedan primer koji se odnosi na dobru i lošu realizaciju stanja *FSM*-a. Kao što se vidi sa slike 50(a) kod loše realizacije u okviru procesa selekcije se vrši izbor stanja, a kod dobre realizacije, tj. rešenja na slici 50(b), stanje se definiše pre procesa selekcije.



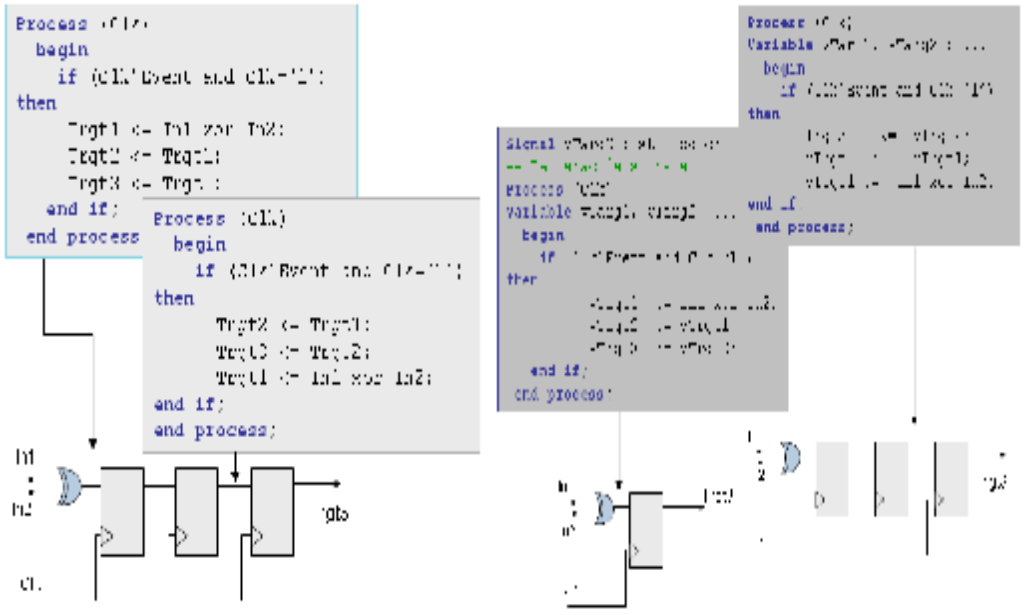
Slika 50• *Primeri korišćenja case petlje*

***Primer2*. Redosled pristizanja signala.** Na slici 51 signal *Max* se upoređuje sa signalom *A+B*. Izlaz komparatora se nakon toga koristi kao selektorski ulaz multipleksera. U slučaju da signal *A* zakasni, tada sa aspekta tajminga dolazi do generisanja nekorektnog izlaza. Superiornije rešenje je prikazano na slici 51(b). U ovom slučaju od vrednosti *Max*, koja je ranije uspostavljena, prvo se oduzma signal *B*, takođe ranije uspostavljen. Nakon toga se izlaz kola za oduzimanje upoređuje sa *A_late* koji se kasnije uspostavlja. Kašnjenje signala *A_late* biće anulirano propagacijom signala kroz kolo za oduzimanje.



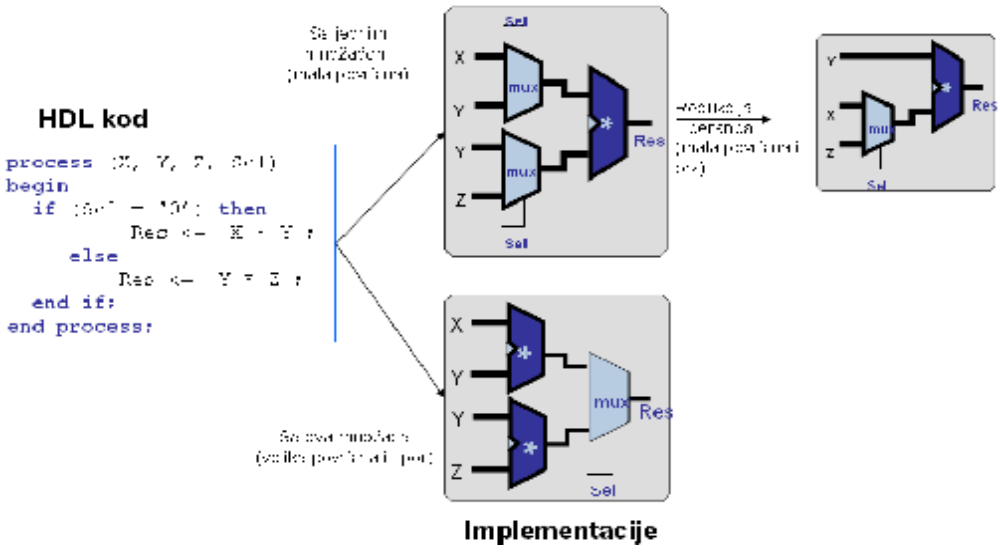
Slika 51• *Realizacija staze podataka sa zakasnelim signalom*

***Primer3*. Korišćenje signala ili promenjivih**. Na slici 52 prikazan je način sinteze hardvera kada se u *VHDL* kodu koriste signali i promenjive. Važno je uočiti da kod korišćenja signala nije važan redosled navođenja istih, dok je kod korišćenja promenjivih (*variable*) bitan redosled navođenja. Sa aspekta potrošnje superiornije je rešenje sa slike 52(b), pod uslovom da se promenljive *vTrgt1* i *vTrgt2* ne koriste u daljem kodu.



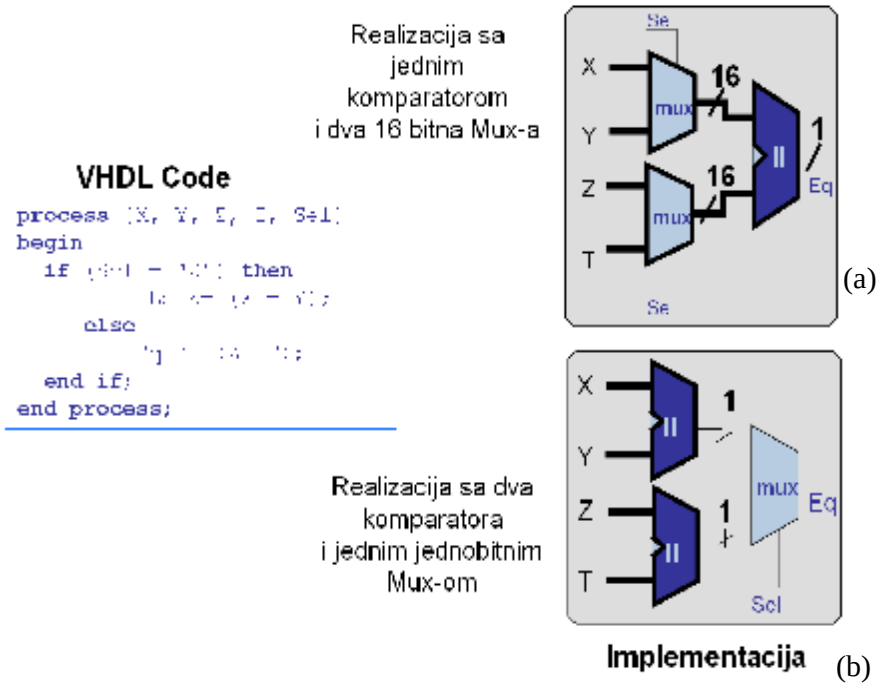
Slika 52• *Korišćenje signala ili promenjivih*

***Primer4*. Planiranje i alokacija hardvera.** Raspodela resursa i uređenje operanda je veoma važan zahtev na koji se mora strogo voditi računa u toku projektovanja. Ako je zadat *HDL* kod prikazan na slici 53, tada se može uočiti da se operand *Y* koristi u oba dela *If* naredbe. Shodno tome možemo preurediti kod sa ciljem da dobijemo rešenje koje je zauzima manje površine, a odlikuje se većom brzinom i manjom potrošnjom.



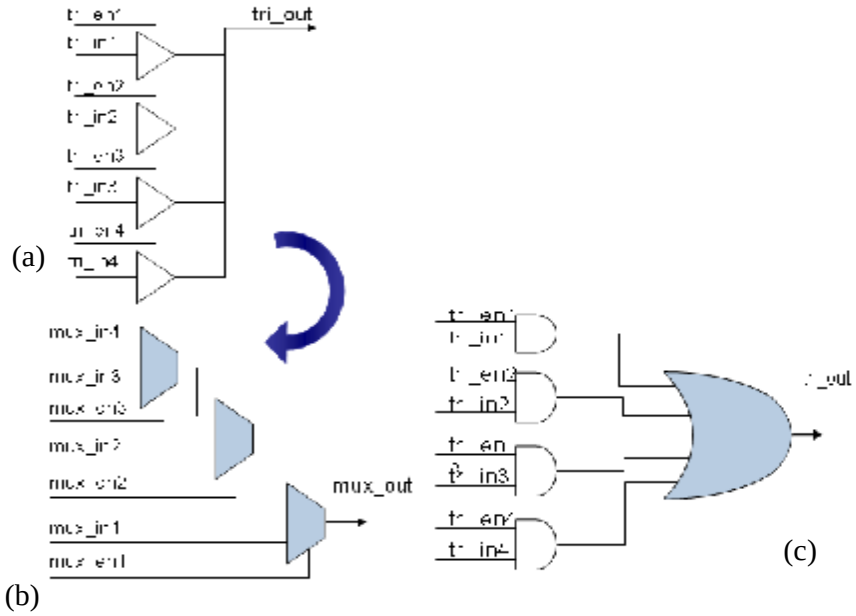
Slika 53• *Uređenje operanda*

***Primer5*. Uticaj magistrala na potrošnju.** Magistrale su veliki potrošači snage. Sledeći primer ukazuje na mogućnost izbegavanja korišćenja obimnih magistrala. Realizacija 16 bit-nih multipleksera ima za efekat veliku potrošnju. Daleko bolje rešenje je korišćenje jednobitnog multipleksera kao što je prikazano na slici 54(b). Ovom realizacijom smo izbegli razvođenje 16 bit-nih magistrala po čipu.



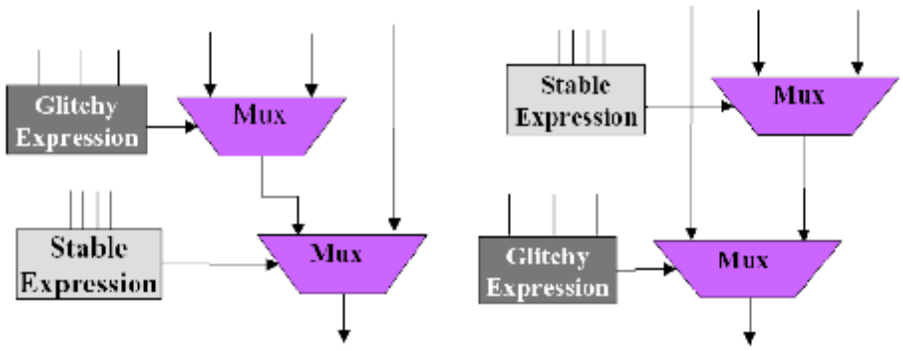
Slika 54• *Izbegavanje korišćenja obimnih magistrala*

***Primer6*. Izbegavanje korišćenja trostatičkih bafera.** Zbog velike parazitne kapacitivnosti na izlaznim linijama (*tri_out* na slici 55(a)) trostatički baferi predstavljaju velike potrošače. Preporuka je da se baferi zamene multiplekserima. Shodno tome i *VHDL* kod treba biti modifikovan. Dva moguća rešenja prikazana su na slikama 55(b) i 55(c). Rešenje sa slike 55(c) je superiornije sa aspekta brzine, površine i potrošnje.



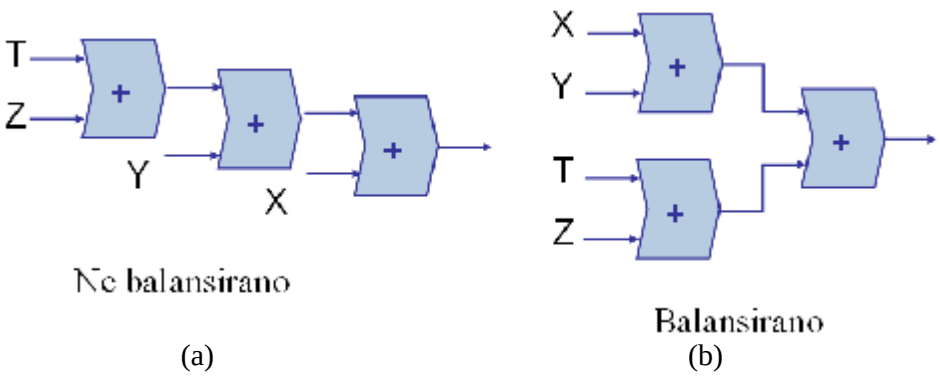
Slika 55• *Trostatički baferi*

***Primer7.* Korišćenje *If...Then...Else* iskaza.** Radi prevencije propagacije prekidačkih aktivnosti kôd *If...Then...Else* iskaza se najčešće preuređuje. Preporuka je da projektant brzo promenljive signale dovede na ulaz zadnjim stepenima dizajna, čime se postiže minimizacija propagacije signala kroz sistem a indirektno se minimizira i pojava gličeva.



Slika 56• *Redukcija propagacije prekidačkih aktivnosti*

***Primer8.* Izjednačavanje kašnjenja**.Veoma često treba voditi računa o balansiranom kašnjenju signala kroz dizajn. Videti sliku 57.



(a) (b)

Slika 57• *Balansiranje signala*

Ako svi primarni ulazi (*T, Z, Y* i *X*) imaju isto vreme pristizanja i istu verovatnoću komutiranja, tada se balansiranjem stabla kao na slici 57(b) izjednačavaju putevi.

4.2 Modeliranje

U predhodnim sekcijama je predstavljen skup rešenja za automatsku optimizaciju potrošnje. Da bi bila efikasna, rešenja za optimizaciju, trebala bi biti podržana od strane pouzdanih i tačnih tehnika za modeliranje potrošnje i analizu rada kola. Trenutni *EDA* alati pružaju obimnu podršku za modeliranje potrošnje energije na nivou logičkih blokova kao i za analizu potrošnje snage na nivou celog sistema. Ukazaćemo u daljem tekstu na neka tipična rešenja [45].

4.2.1 Disipacija usled komutiranja

Potrošnja snage usled komutiranja izračunava se pomoću formule CV^2f. Da bi tačno proračunali potrošenu snagu usled komutacije, u biblioteci se moraju naći podatci o naponu i kapacitivnosti. Kapacitivnost se određuje na osnovu kapacitivnosti pina, koji se specificira u biblioteci ćelija, a kapacitet veze se može odrediti na osnovu podataka koji se dobijaju od sredstava za projektovanje fizičkog *layout*-a, ili da se izračunaju na osnovu raspoloživih modela o vezama.

4.2.2 Interna disipacija

Interna disipacija ćelije uključuje disipaciju prouzrokovanu komutatorskim aktivnostima u unutrašnjim čvorovima ćelije kao i snagu koja se troši usled struje kratkog spoja. Interna disipacija ćelije zavisi od vremena trajanja usponskih i opadajućih ivica ulaznih impulsa kao i od izlaznih kapacitivnosti. Ova disipacija se modelira u biblioteci kao *lookup* tabela, takozvani nelinearni model disipacije (*NLPM*), koji je indeksiran sa dve promenjive: vreme rasta/pada ivice ulaznih impulsa, i izlazna kapacitivnost.

Lookup tabela se specificira šablonom koji se određuje vrednosću indeksa u tabeli za različite ulaze. Kod specificiranja vrednosti potrošnje, koristi se specifični šablon tabela, pri čemu se specificiraju samo vrednosti disipacije u različitim tačkama u tabeli. Primer jednodimenzionalnog šablona, *power_1D*, i dvodimenzijalnog *power_2D* predstavljen je na slici 58.

```
Power_lut_template (power_1D) {
Variable_1 : input_transition_time;
Index_1 ("1000, 1004, 1005, 1006");
}
power_lut_template (power_2D) {
variable_1 : input_transition_time;
variable_2 : total_output_net_capacitance;
index_1 ("1000, 1001, 1002");
index_2 ("1000, 1001, 1002, 1003, 1004, 1005, 1006");
}
```

Slika 58• *Jednodimenzionalni i dvodimenzionalni šablon*

Napredniji metod modeliranja koji obezbeđuje korišćenje više vrednosti napona i više promenjivih naziva se skalabilni polinomni model snage (***S**calable **P**olynomial **P**ower **M**odel- SPPM*). Ovaj format je opisan kasnije.

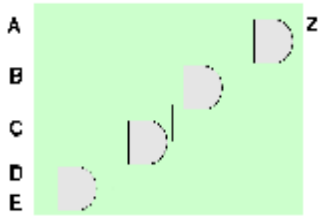
Interna disipacija može biti zavisna od stanja, putanje ili od oboje. Na zavisnost od stanja ukazuje činjenica da interna disipacija ćelija može biti različita što zavisi od stanja ulaza i izlaza ćelije koje se ne komutiraju. Tako na primer, potrošena snaga kada *clock pin* bloka *RAM* memorije komutira, zavisi od toga da li je *RAM* u stanju čitanja, pisanja ili u stanju *idle*. Ovaj efekat se može sagledati pomoću *State-*

dependency construct "*When*". *State-dependent* model potrošnje za *RAM* ćeliju prikazan je u nastavku teksta na slici 59, i ukazuje da se disipacija posebno specificira za sledeća tri stanja: *read*, *write* i *idle.*

```
Cell (RAM)
....
pin (clk)
internal power {
when (!RD & WR & CS) {/*write state*/
rise_power (power_1D)
      values ("1.5 2.5 5.6 7.7")}
fall_power (power_1D)
      values ("1.7 2.4 5.3 7.2")}
when (RD & !WR & CS) {/*read state*/
rise_power (power_1D)
      values ("2.5 3.5 6.6 8.7")}
fall_power (power_1D)
      values ("2.7 3.4 6.3 8.2")}
when (!CS) {  /*idle state*/
power (power_1D)
values ("0.6 2.0 1.6 4.7")}
}
```

Slika 59• *Model potrošnje za RAM*

Path-dependency ukazuje na činjenicu da je interna disipacija ćelije različita u zavisnosti od toga koja promena ulaza je izazvala komutaciju. U primeru prikazanom na slici 60, potrošnja snage je veća ako je ulazna promena izazvana promenom na ulazu *D*, umesto promene na ulazu *A*. Ovaj efekat se može opisati pomoću *Path-dependency construct related_pin*, koja se odnosi na internu disipaciju i data je kodom na slici 61.



Slika 60• *Path-dependency*

```
Cell (AND5)
Pin (Z)
Internal power {
```

```
Related_pin A {
Power (power_2D)
Values ("0.5 1.5 3.6," "0.6 1.7 4.0,"Ö.)}
ÖÖÖÖ
Related_pin D {
Power (power_2D)
Values ("1.5 2.5 5.6," "1.6 2.7 5.7,"Ö.)}
Ö...Ö
}
```

Slika 61• *Path-dependency construct related_pin kola sa slike 60*

4.2.3 Modeliranje statičke potrošnje i potrošnje usled struje curenja

Kao što smo rekli u predhodnim poglavljima, curenje u kolima može nastati iz više različitih izvora: ***subthreshold leakage***, snižavanje granica indukovanih drejnom, curenje drejna indukovane gejtom itd. [46]. Nezavisno od fizičkih razloga koji se odnose na potrošnju usled struja curenja, u bibliotekama su često opisane ćelije kod kojih je definisana ukupna disipacija usled struja curenja. Model curenja je ***state-dependent*** i može se specificirati za različite vrednosti struje curenja za različite vrednosti napona na ulazima ćelija. Jedan primer modela struje curenja ćelija je oblika kao na slici 62.

```
cell my_cell() {
leakage_power () {
when: "A & B"
value: 5.5
}
cell_leakage_power: 4.0
}
```

Slika 62• *Model struje curenja*

Na osnovu ovog primera jasno je da ćelija *my_cell* troši 5,5 jedinica curenja kada su oba ulaza ***A*** i ***B*** na visoko, a troše 4,0 jedinica u ostalim slučajima.

4.2.4 Skalabilni polinomski modeli snage (*SPPM*)

Opšte je poznato da je moguće kreirati tabelu za interpolaciju funkcije na osnovu nekoliko karakterističnih tačaka, uz korišćenje faktora za skaliranje, nazvanih *k*-faktori, radi ekstrapolacije nepoznatih međutačaka.

Kada se koristi veći broj napona za napajanje, veći broj napona pragova provođenja, i napona inverzne polarizacije, primenom ove metodologije ne ostvaruje se željena tačnost. Zbog toga su potrebne nešto sofisticiranije tehnike koje će se koristiti kod energetsko efikasnijih rešenja. Šta više, kako veličina čipa raste a veličina tranzistora se smanjuje, promene u temperaturi unutar čipa postaju značajne i moraju biti odgovarajuće modelovane.

Skalabilni polinomski modeli predstavljaju efikasn i brži alternativni pristup u odnosu na korišćenje *lookup* tabela u kojima se memorišu nelinearne vrednosti. Ovi modeli uzimaju u obzir bibliotečne informacije i koriste jednačine za tačno izračunavanje disipacije. Svaki od napona korišćenih za napajanje kao i inverznu polarizaciju predstavlja promenjivu u jednačini, i memoriše se u biblioteci sa informacijama o ćeliji. Format zasnovan na korišćenju jednačina omogućava sredstvima za projektovanje da odrede tačne podatke o tajmingu i disipaciji ćelija za širok operativni opseg uslova rada. Ćelije, sa tačke gledišta tajminga, se opisuju skalabilnim polinomskim modelima kašnjenja (*SPDM*-ovima), kad je u pitanju disipacija koriste se *SPPM*-ovi a kada su u pitanju struje curenja koriste se skalabilni polinomski modeli struja curenja *SPLM*-ovi. Zbog njihove kompaktne prezentacije oni mogu biti upotrebljeni za modelovanje gde postoje mnogo veći stepeni slobode. Naprimer, oni nam dozvoljavaju da modelujemo uticaj varijacija napona napajanja i promena temperature bez povećavanja obima kola.

SPPM sintaksa omogućava projektantima da koriste do 7 promenjivih. Za veoma obimne podatke gde postoje nagle promene vrednosti, samo jedan polinom ne može dobro da obuhvati ceo radni opseg od interesa. U ovom slučaju, treba da se koristi segmentna ili adaptivna domenska polinomska sintaksa.

4.2.5 Modeliranje aktivnosti

U predhodnom delu diskutovano je o modelima fizičkih komponenata koje se koriste kod modeliranja potrošnje, komutirane snage, interne snage i gubitaka. Ostala komponenta, koja treba da se modelira na odgovarajući način, je aktivnost komutiranja.

Na osnovu korišćene analize toka, komutirajuća aktivnost se može detaljnije modelirati na različitim nivoima. Ako se zahteva kompletan vremensko bazirani profil disipacije čipa, da bi se u detalje sagledale komutatorske aktivnosti neophodno je koristiti *value change dump VCD* ili *VCD+* formate .

Ako nas interesuje samo prosečna disipacija snage, koristi se nešto kompaktnija prezentacija komutatorske aktivnosti koja se naziva *switching activity interchange format* (*SAIF*) [10]. *SAIF* je *ASCII* format i sagledava statističke osobine komutacija svakog čvora u dizajnu u zavisnosti od statičkih i dinamičkih atributa koji mogu biti zavisni kako od stanja tako i od putanje. Obuhvaćeni atributi su predstavljeni sledećim:

4.2.4.1 Statički atributi

- *T0*: vreme provedeno u stanju 0
- *T1*: vreme provedeno u stanju 1
- *TX*: vreme provedeno u nepoznatom stanju
- *TZ*: vreme provedeno u stanju visoke impedanse
- *TB*: vreme provedeno u stanju sudara na magistrali (dva ili više drajvera simultano upravljaju istim objektom)

4.2.4.2 Dinamički atributi

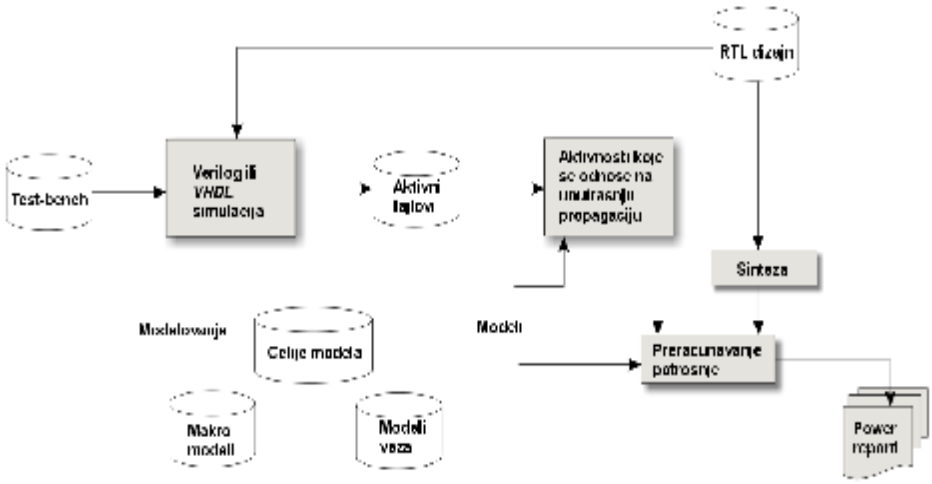
- *TC*: broj prelaska 0→1 ili 1→0. Ovo može biti podeljeno u prelaze usponskih ili opadajućih ivica.
- *TG*: broj transportnih gličeva. Ovo su gličevi na izlazu gde je širina impulsa na izlazu veća nego kašnjenje gejta. Ovo troši istu snagu kao i pun prelaz.
- *IG*: broj inercijalnih gličeva. Ovo su gličevi kod kojih je širina impulsa na izlazu manja nego kašnjenje gejta. Ovi gličevi ne troše istu energiju kao i kod pune tranzicije, i koristi se *derating* faktor za izračunavanje disipacije.

Uprkos osnovnim atributima, *SAIF* jezik poseduje konstrukcije stanja i *path* zavisnosti, sa ciljem da uzme u obzir specifične informacije o statistici komutiranja pojedinog čvora ili ćelija. Oba atributa, statički i dinamički su zavisni od stanja, pa se sagledavanja statistike komutiranja za različita stanja ćelija mogu izvesti posebno. *State-dependent* statički atributi su korisni za izračunavanje disipacije usled curenja i za izračunavanje dinamičke snage.

Dinamički atributi takođe mogu biti *path-dependent*, jer uzimaju u obzir posebne komutatorske aktivnosti zasnovane na tome koji je ulazni put uzrokovao prelaz na nekom pinu.

4.3 Tokovi analize

Optimizacija potrošnje koja se sprovodi od strane *power compiler*-a [47] daje podatke o prosečnoj potrošnji i koristi *SAIF* kompaktnu prezentaciju koja se odnosi na komutirajuću aktivnost. Moguće je, za potrebe analize disipacije, koristiti *SAIF* počev od simulacija na *RTL* ili na gejt nivou. I pored toga što simulacija na gejt nivou nije tako tačna, simulacija zasnovana na *RTL* nivou takođe nije toliko tačna što je ujedno i njena prednost. Na slici 63 prikazan je tok analize disipacije kod *RTL* bazirane simulacije.



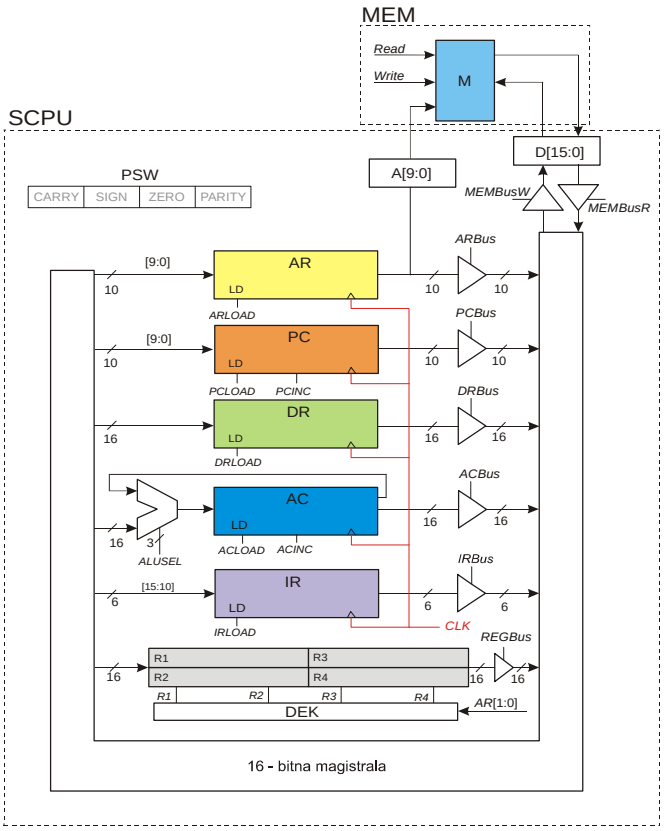
Slika 63• *Analiza disipacije RTL bazirane simulacije*

Tok *gate-level* simulacije je vrlo sličan, sa izuzetkom da se ne zahteva aktivnost koja se odnosi na internu propagaciju, jer se sagledava na ulaznim pinovima ćelija u netlisti, na gejt nivou, uz pomoć korišćenja simulacije na nivou gejta. S obzirom da je ova aktivnost sagledana u svim detaljima, moguće je korišćenje *state and path-dependent* informacija koja se dobija iz bibliotečkih modela, pa zatim da se uz pomoć *SAIF*-a obavi tačnija analiza disipacije. Kod post-razmeštajne ili post-rutirajuće netliste, kapacitivnosti veze se mogu ponovo odrediti i u ponovnom pokušaju izračunavanja disipacija tačnije odrediti.

Poznatiji program koji se koristi za procenu disipacije je *PrimePower* [48].

5 Optimizacija potrošnje *ALU*-a

Struktura jednog jednostavnog procesora nazvanog *SCPU* (***S**imple **C**entral **P**rocessing **U**nit*) prikazana je na slici 64. Blok *MEM* predstavlja memoriju, a blok *SCPU* je jednoadresni 16 bit-ni procesor. *SCPU* je interno organizovan oko 16 bit-ne magistrale. Gradivnim blokovima *A*[9:0] i *D*[15:0] *SCPU* je spregnut sa *MEM* preko adresne magistrale i magistrale podataka, respektivno.

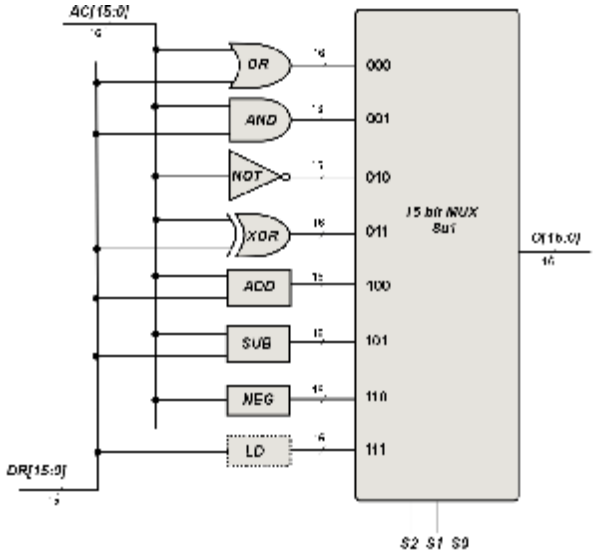


Slika 64• *Struktura jednostavnog procesora*

Od interesa ovog rada, sa tačke gledišta potrošnje, predstavlja projektovanje *Low power ALU*-a. U nastavku prvo će biti dat opis standardne strukture *ALU*-a, a nakon toga njene modifikovane verzije koja se odlikuje manjom potrošnjom.

5.1 Standardna aritmetičko-logička jedinica

Aritmetičko-logička jedinica (***A**rithmetic **L**ogic **U**nit- ALU*) (videti sliku 65) je izvršna jedinica i predstavlja deo procesora koji vrši neposrednu obradu podataka. Od logičkih operacija *ALU* obavlja *AND, OR, NOT* i *XOR*, a od aritmetičkih *ADD, SUB* i *NEG*. U toku izvršenja operacije *LD, ALU* direktno prosleđuje izvorni memorijski operand akumulatoru *AC*. Izvorni 16-bitni operandi se dovode na ulaz *ALU*-a preko linija *AC*[15:0] i *DR*[15:0], a rezultat se dobija na izlazu *O*[15:0].



Slika 65• *Aritmetičko-logička jedinica*

Logičke operacije obavljaju sledeći gradivni blokovi:

- *OR* obavlja pobitnu logičku *OR* operaciju nad 16-bitnim operandima
- *AND* obavlja pobitnu logičku *AND* operaciju nad 16-bitnim operandima
- *NOT* obavlja logičku *NOT* operaciju nad 16-bitnim operandom

- *XOR* obavlja pobitnu logičku *XOR* operaciju nad 16-bitnim operandima

Aritmetičke operacije obavljaju sledeći gradivni blokovi:

- *ADD*[*] obavlja aritmetičku operaciju *ADD* nad 16 bit-nim operandima
- *SUB*[*] obavlja aritmetičku operaciju *SUB* nad 16 bit-nim operandima
- *NEG*[*] obavlja aritmetičku operaciju *NEG* nad 16-bitnim operandom, tj. određuje dvojični komplement
- *LD* prosleđuje 16 bit-ni izvorišni operand sa linije *DR*[15:0] na ulaz *MUX*

Na osnovu tipa operacije multiplekser *MUX*, tipa 8-u-1, selektuje jedan od 16 bit-nih ulaznih podataka. Selektorski ulazi *S0, S1* i *S2* (naznačeni kao *ALUSEL* na slici 64) definišu koji će se od ulaza proslediti na izlaz.

Signal *ALUSEL* predstavljaju tri bita najmanje težine, polja *opkôda* instrukcije (vidi sliku 66).

opkôd				M	
X	S2	S1	S0	Y	Y

Slika 66• *Sadržaj IR registra*

Na primer, ako se izvršava operacija *ADD* tada imamo da je *S2*={1}, *S1*={0} i *S0*={0}, dok ćemo kod operacije *XOR* imati da je *S2*={0}, *S1*={1} i *S0*={1}. Operaciji premeštanja odgovara sledeća kombinacija selektorskih signala *S2*={1}, *S1*={1} i *S0*={1}.

Za projektovanje *ALU*-a korišćen je softver ***Xilinx® ISE™ Project Navigator Release Version: 7.1 i*** [49].

Kompletan *VHDL* kôd *ALU*-a dat je u dodatku na kraju rada kao entitet *Alu_base*. Arhitekturu *Alu_base_a* čine sledeće instancirane komponente: *Mux_Nbit, Or_Nbit, And_Nbit, Not_Nbit, Xor_Nbit, Add_Nbit, Sub_Nbit i Neg_Nbit.* Ove komponente predstavljaju gradivne blokove *ALU*-a, i prikazane su na slici 65. *VHDL* opis ovih kola takođe je dat u dodatku.

---

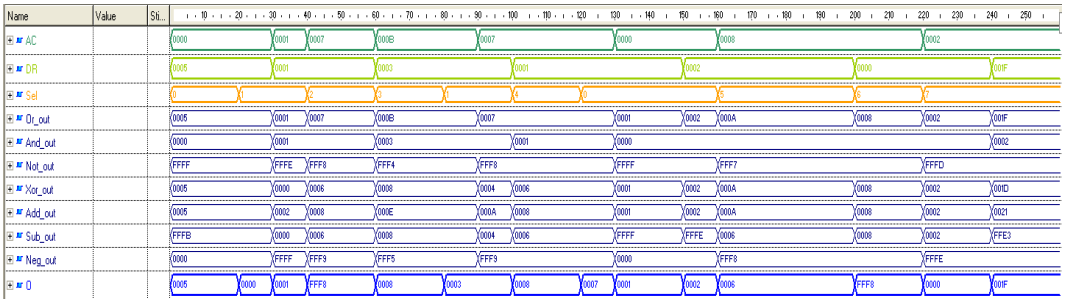
[*] *u cilju pojednostavljenja realizacije aritmetičkih operacija ADD, SUB i NEG, usvojeno je da kola za sabiranje, oduzimanje i negaciju ne generišu izlaz C_{out} i ne prihvataju ulaz C_{in}*

Kôd *ALU*-a napisan na *VHDL*-u je verifikovan pomoću *Test Bench*-a *Alu_base_tb* koji je dat na slici 67.

```
-- TestBench Template
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Alu_base_tb  is
      generic (N: integer:=15);
end entity;
architecture TB_a of Alu_base_tb is
component Alu_base is
      generic ( N: integer:=15);
      port(
--Vektori ulaznih signala
      AC, DR: in std_logic_vector (N downto 0);
      Sel: in std_logic_vector (2 downto 0);--Selektorski ulaz
      O: out std_logic_vector (N downto 0));--Izlazni vektor
end component;
Signal AC, DR, O: std_logic_vector (N downto 0);
Signal Sel: std_logic_vector (2 downto 0);
begin
      UUT:Alu_base            -- Blok koji testiramo je osnovna
      generic map (N=>N)      -- verzija ALU-a
      port map(AC,DR,Sel,O);

      Sel<="000",                          --sekvenca instrukcija
   "001" after 20ns,"010" after 40ns,"011" after 60ns,
      "001" after 80ns,"100" after 100ns,"000" after 120ns,
   "101"after 160ns,"101" after 180ns,"110" after 200ns,
   "111" after 220ns;
      -- Promena podataka na AC magistrali
      AC<="0000000000000000","0000000000000001" after30ns,
"0000000000000111" after 40ns,"0000000000001011" after60ns,
"0000000000000111" after 90ns,"0000000000000000" after 130ns,
"0000000000001000" after 160ns,"0000000000000010" after 220ns;
      -- Promena podataka na DR magistrali
      DR<="0000000000000101","0000000000000001" after30ns,
"0000000000000011" after 60ns,"0000000000000001" after100ns,
"0000000000000010" after150ns,"0000000000000000" after200ns,
"0000000000011111" after 240ns;
end architecture;
```

Slika 67• *VHDL kôd test bench-a*

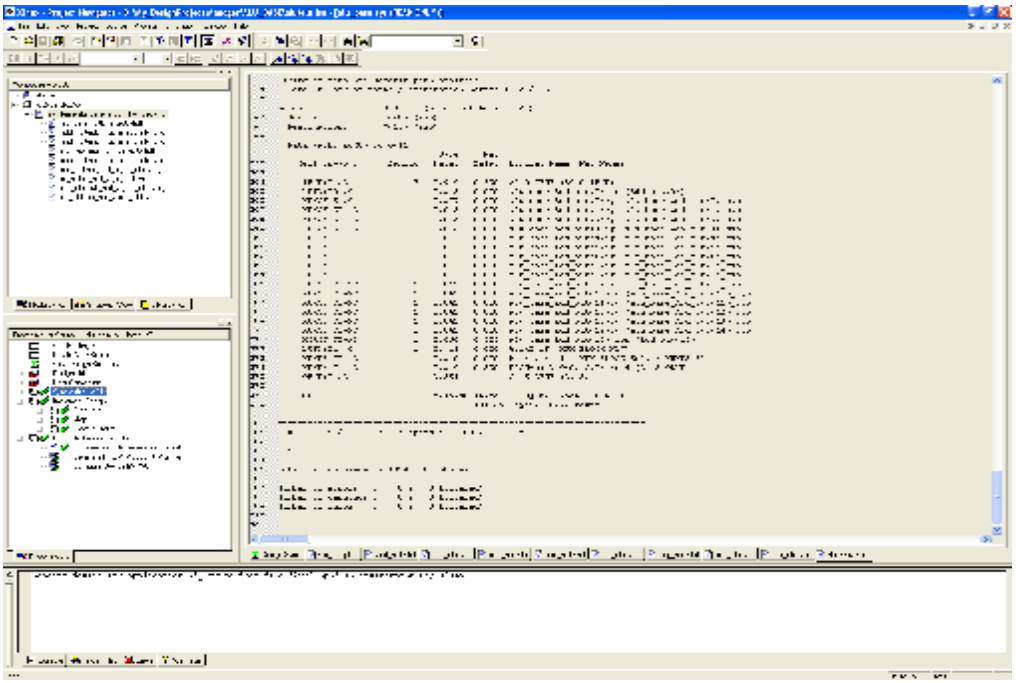
Za testiranje su odabrane slučajne vrednosti na magistralama *AC* i *DR*. Izvršeno je testiranje svih instrukcija, koje su podržane od strane *ALU*-a. Redosled testiranja instrukcija odabran je na osnovu analize izvršavanja instrukcija realne programske sekvence. Nakon testiranja dobijen je dijagram kao na slici 68.



Slika 68• *Vremenski dijagram dobijen nakon testiranja*

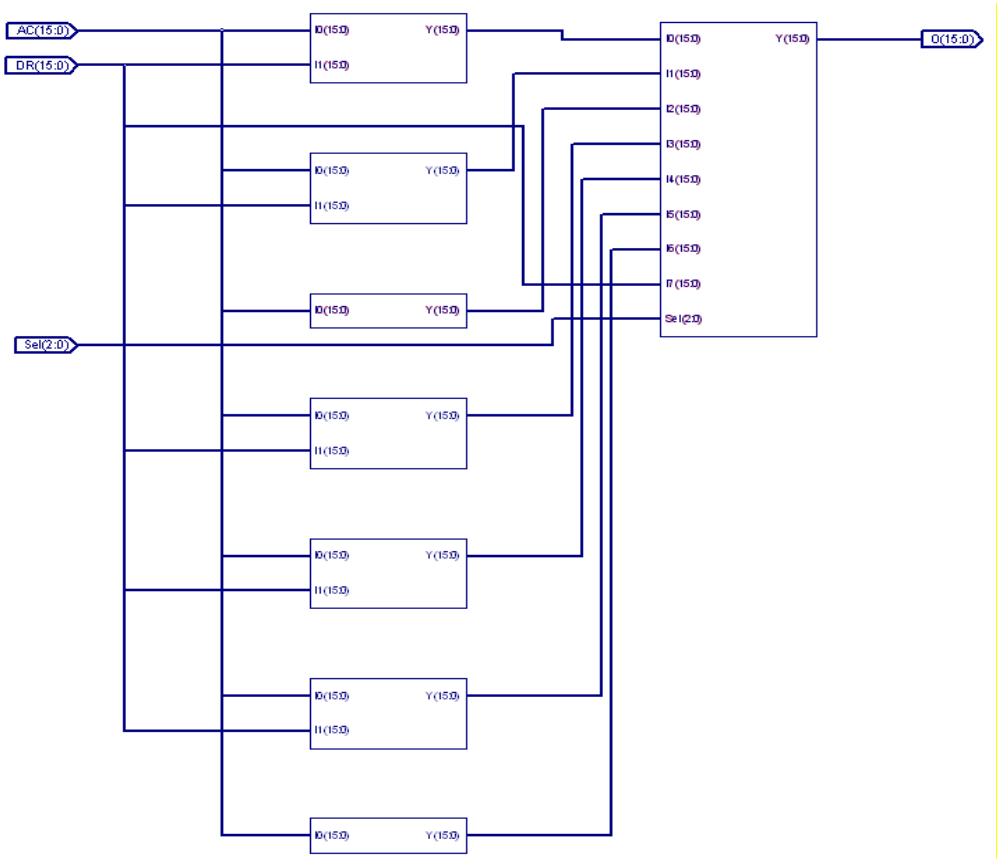
Može se primetiti da 16 bit-ni signali na izlazima blokova (*Or_out, And_out, Not_out, Xor_out, Add_out, Sub_out i Neg_out*) menjaju vrednost sa svakom promenom 16 bit-nih signala na magistralama *AC* ili *DR*. Ovo znači da gradivni blokovi (*Or, And, Not, Xor, Add, Sub* i *Neg*) troše energiju pri svakoj promeni signala na *AC* ili *DR*, a da se pri tome koristi samo jedan izlaz *ALU*-a.

Sinteza *ALU* jedinice je izvršena uz pomoć sredstva za sintezu ***Xilinx XST Vhdl***, koji je implementiran u ***Project Navigator***-u. Slika 69 ilustruje radno okruženje i deo *Synthesis Report*-a. Kompletan *Report* je dat u dodatku na kraju rada kao *Alu_base.prj*.



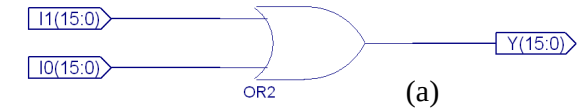
Slika 69• *Report dobijen nakon sinteze*

Nakon izvršene sinteze dobijena je šema *ALU*-a na *RTL* nivou koja je prikazana na slici 70.

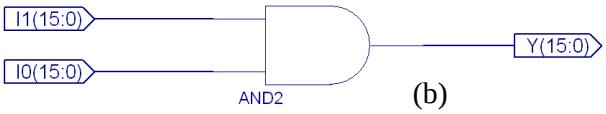


Slika 70• *ALU na RTL nivou*

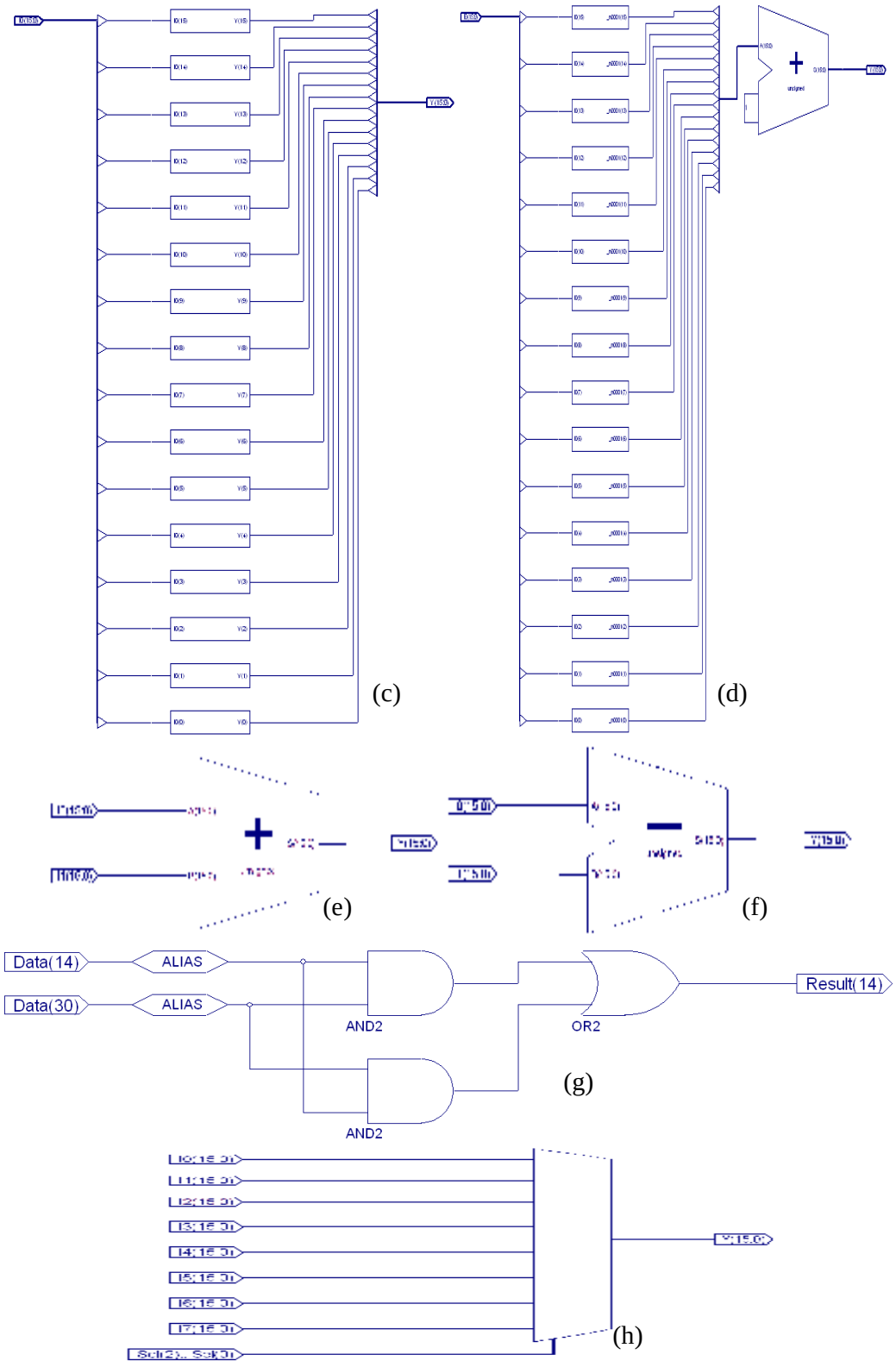
Blokovi prikazani na slici 70 odozgo na dole su: *Or, And, Not, Xor, Add, Sub* i *Neg*. Na slici 71 su dati izgledi svih gradivnih blokova na *RTL* nivou i to 16 bit-ni: (a) *Or*, (b) *And*, (c) *Not*, (d) *Neg*, (e) *Add*, (f) *Sub*, (g) *Xor* i (h) *Mux*.



(a)

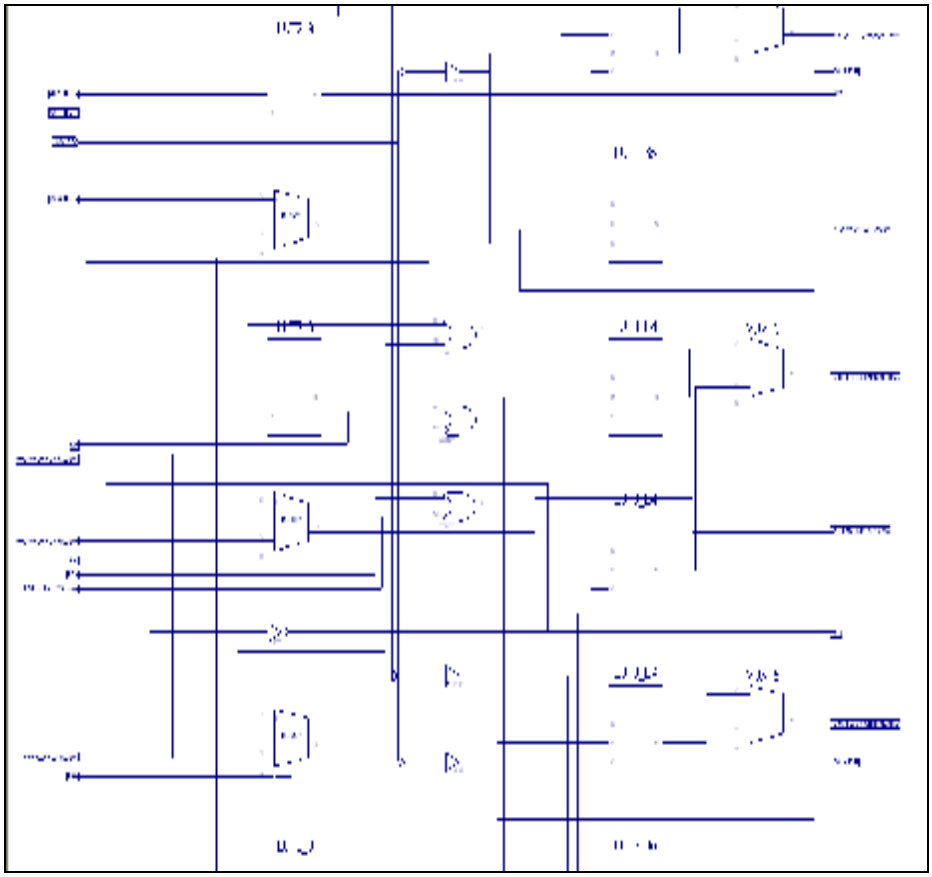


(b)



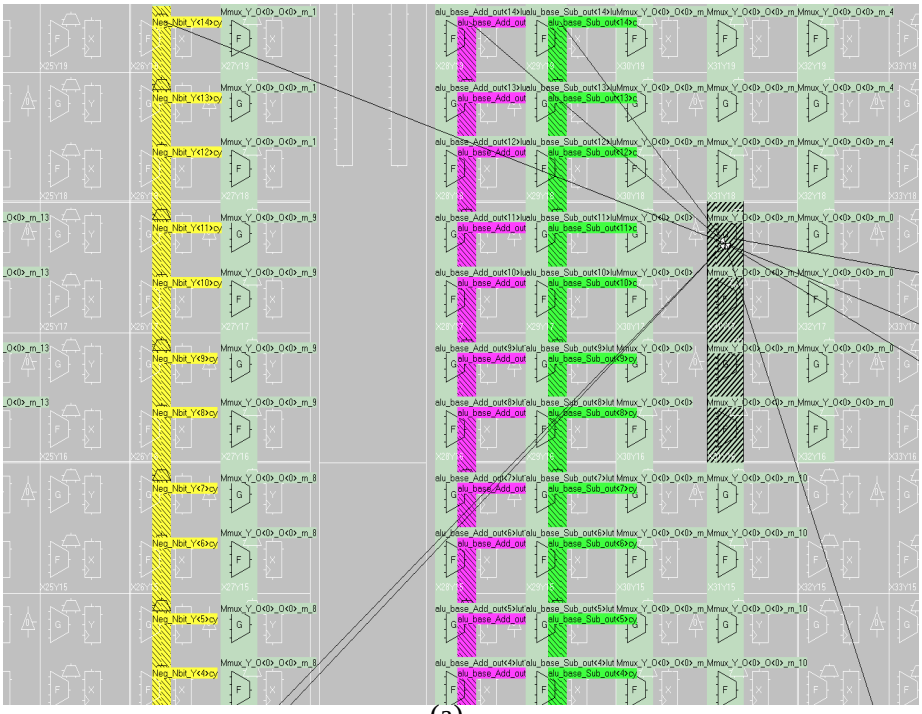
Slika 71• *Gradivni blokovi ALU-a*

Kako je tehnološki izgled rešenja *ALU*-a na *RTL* nivou dosta obiman (10 stranica), i ne može se prikazati, na slici 72 je dat samo mali deo *ALU*-a. Kompletan izgled na tehnološkom nivou nalazi se na *CD*-u koji je priložen uz rad.

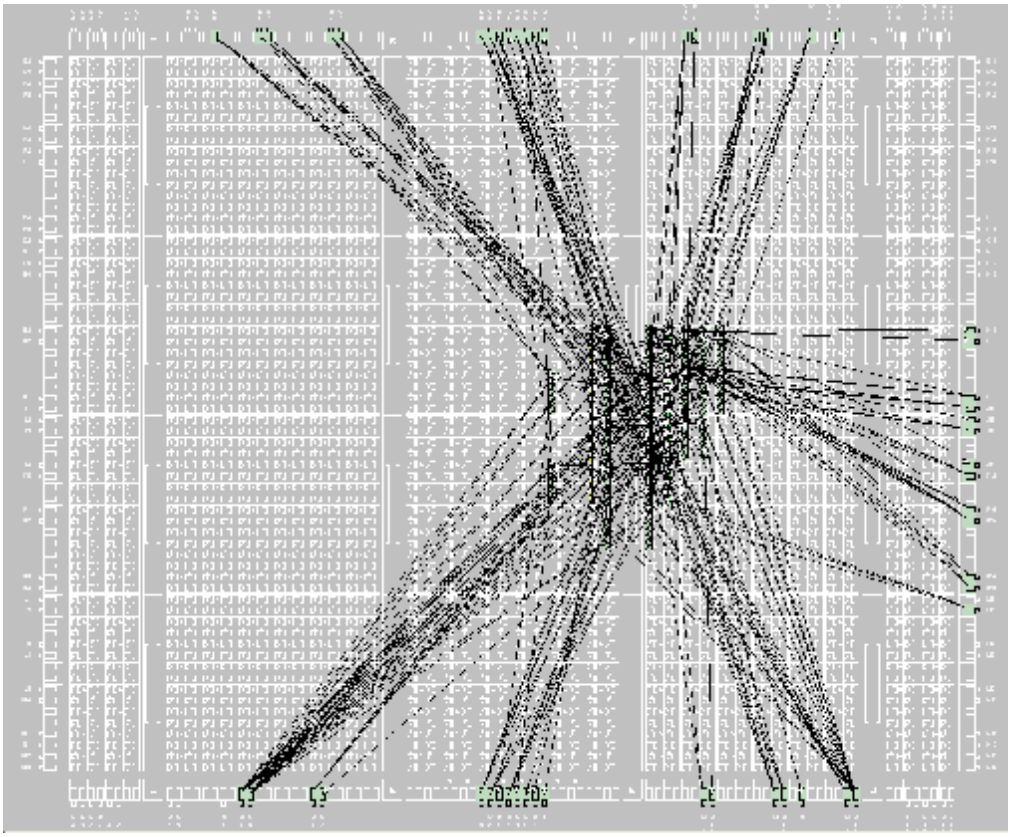


Slika 72• *Izgled ALU-a na tehnološkom nivou*

U nastavku je obavljena implementacija na čip iz *Xilinx*-ove familije ***Virtex™ 2p*** sa oznakom **xc2vp2fg256-6** (najmanji u seriji sa 3168 logičkih ćelija, 12 multiplekserskih blokova 18x18 bit-a i veličine 17x17 mm [50]). Alat izvršava implementaciju u tri koraka i to: (1) translacija, (2) mapiranje, (3) razmeštanje i rutiranje. Nakon izvršene implementacije dobijen je izgled čipa kao na slici 73(a). Na slici 73(b) je dat izgled kola sa rutiranim vezama. *Report*-i nakon izvršnih faza implementacije su dati na *CD*-u.



(a)



(b)

Slika 73• *Izgled čipa nakon obavljene faze implementacije*

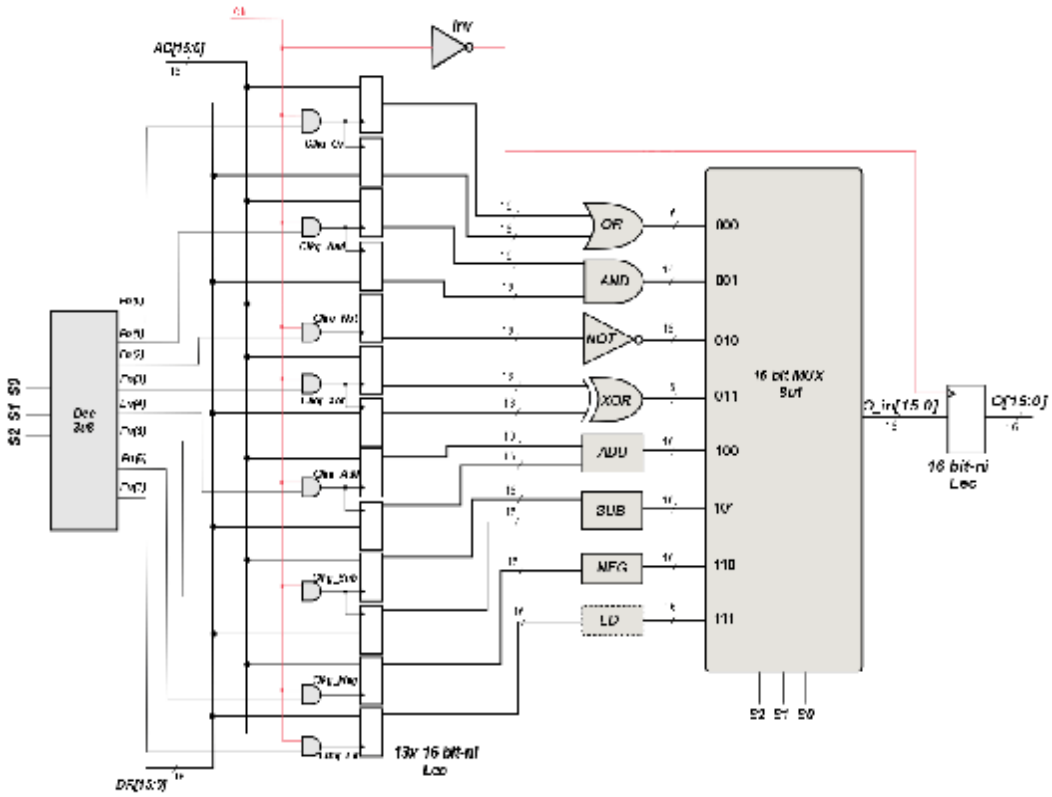
Poslednji korak koji je izvršen je analiza snage uz pomoć ***XPower*** alata. Kompletan *report* koji je dobijen nakon ove analize je prikazan u dodatku (*Alu_base.ncd*), a deo rezultata koji se odnosi na upoređivanje performansi sa modifikovanim *ALU*-om opisan je u podsekciji 5.2.

5.2 Realizacija modifikovanog *ALU*-a

Ideja o redukciji potrošnje koja će biti izložena u ovoj podsekciji se bazira na sledećem konceptu. *ALU* sa slike 65 se sastoji od osam funkcionalnih blokova koji rade paralelno. U jednom trenutku *ALU* obavlja samo jednu operaciju čiji se rezultat bira multiplekserom. To znači da sedam funkcionalnih blokova od ukupno osam komutira svoje izlaze nepotrebno. Smanjenje potrošnje baziraće se upravo na ideji da samo jedna funkcionalna jedinica bude aktivna u toku izvršenja operacije, a ostale će biti inhibirane. Da bi se ovaj koncept sproveo neophodno je pre početka *ALU* operacije dekodirati opkod *ALU*-a i napuniti samo one lečeve koji su na putu generisanja izlaza. Dodatno, izlaz *ALU*-a se takođe lečuje. Nedostatak ovog pristupa ogleda se u ugradnji dodatnih lečeva na ulazima svake funkcionalne jedinice, leča na izlazu *ALU*-a, ugradnji dekodera opkoda *ALU* operacije, i redukciji brzine rada. Struktura modifikovanog *ALU*-a je prikazana na slici 74.

Lečevi prihvataju signale sa magistrala *AC* i *DR* u trenutku kada je aktivan signal *En(i)* koji predstavlja jedan od izlaza dekodera *ALU*-ovog opkoda. Dekoder je tipa 3 u 8. Na primer, ako je potrebno obaviti operaciju *Or*, podaci sa magistrala *AC* i *DR* će biće prosleđeni na ulaze *Or* kola ako je aktivan signal *En(0)*, u trenutku nailaska prednje ivice takta *Clk*. Shodno slici 74 taktni signal *Clkg_Or* dobija se na izlazu *And* kola koje predstavlja *CG* ćeliju. U konkretnom slučaju za *Or* operaciju *En(0)* biće aktivan samo u slučaju kada je *Sel*='000'. Sličan zaključak važi i za ostale instrukcije. Leč koji se nalazi na izlazu iz multipleksera dodatno smanjuje potrošnju, tako što ignoriše nepotrebne promene na izlazu nastale usled promene *Sel* ulaza multipleksera.

Ovaj leč mora biti taktovan sa zakašnjenjem, u odnosu na lečeve na ulazu, za vreme koje je potrebno da rezultat bude dostupan na izlazu iz multipleksera.



Slika 74• *Modifikovani ALU*

Treba naglasiti da takt *Clk* ne predstavlja globalni takt procesora. *Clk* treba da bude zakašnjen u odnosu na globalni takt. Zakašnjen je za vreme koje je potrebno da validan podatak bude stabilan na magistrali, kako bi *ALU* na ulazu imala prave podatke.

VHDL kôd kompletne *ALU_lowpower* je dat u dodatku. Gradivni blokovi su ostali nepromenjeni u odnosu na predhodnu realizaciju *ALU*-a. Dodati su novi blokovi i to: (a) *Dec*, dekoder; (b) *Lach*, leč; (c) *And_1bit* i (d) *inv1*, invertor. *VHDL* kodovi navedenih blokova su dati na slici 75.

```
-- Dekoder
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
```

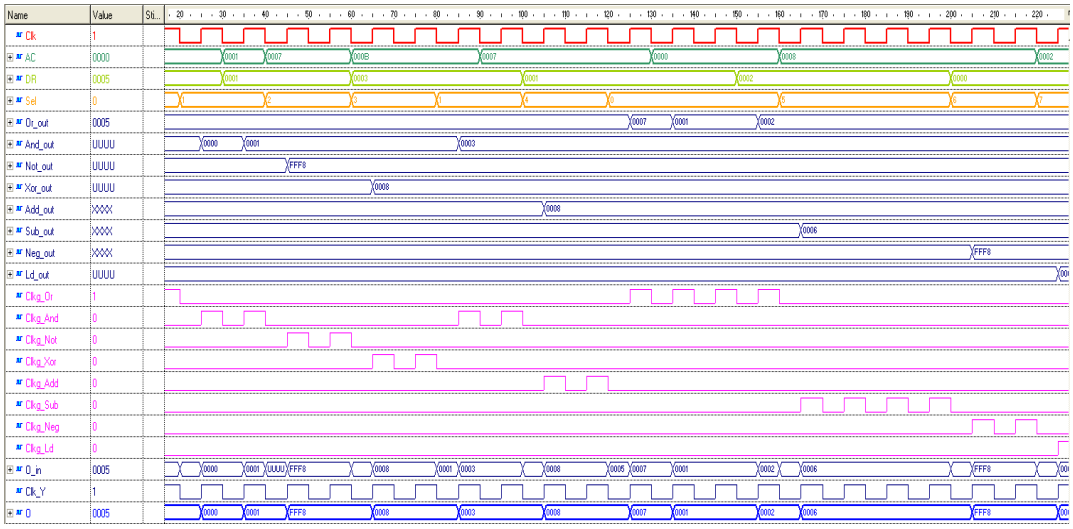
```
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Dec is
port (
            Sel:in std_logic_vector(2 downto 0);
            Y: out std_logic_vector (7 downto 0));
end Dec;

architecture Behavioral of Dec is
begin
            with Sel select
      Y<="00000001" when "000",
      "00000010" when "001",
      "00000100" when "010",
      "00001000" when "011",
      "00010000" when "100",
      "00100000" when "101",
      "01000000" when "110",
      "10000000" when "111",
      "00000000" when others;
end Behavioral;

--VHDL kod Lach-a
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity Latc_Nbit is
generic (N: in integer:=15);
      port (
      I: in std_logic_vector (N downto 0);
      Clk: in std_logic;
      Y: out std_logic_vector ( N downto 0));
end Latc_Nbit;

--Proces P0 u okviru arhitekture Latc_a
--je osetljiv na takt Clk
architecture Latc_a of Latc_Nbit is
begin
      P0: process (Clk)
      begin
       if Clk'event and Clk='1' then
            Y<=I;
            end if;
      end process;
end architecture;

--VHDL kod And_1bit kola
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

-- Entitet And_1bit predstavlja dvoulazno And kolo
entity And_1bit is
port(
```

```
            I1,I2: in std_logic;
            Y :out std_logic);
end And_1bit;

architecture Behavioral of And_1bit is
begin
        Y<=I1 and I2;

end Behavioral;

--VHDL kod invertora inv1
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
entity Inv1 is
      port(
      I: in std_logic;
      Y: out std_logic);
end entity;

architecture Inv1_a of Inv1 is
      signal Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8:std_logic;
begin
      Y1<= not I;
      Y2<=not Y1;Y3<=not Y2;Y4<=not Y3;
      Y5<=not Y4; Y6<=not Y5;Y7<= not Y6;
      Y8<=not Y7;
      Y<= Y8;
end architecture;
```

Slika 75• *VHDL kodovi sledećih blokova:Dec, Lach, And_1bit i inv1*

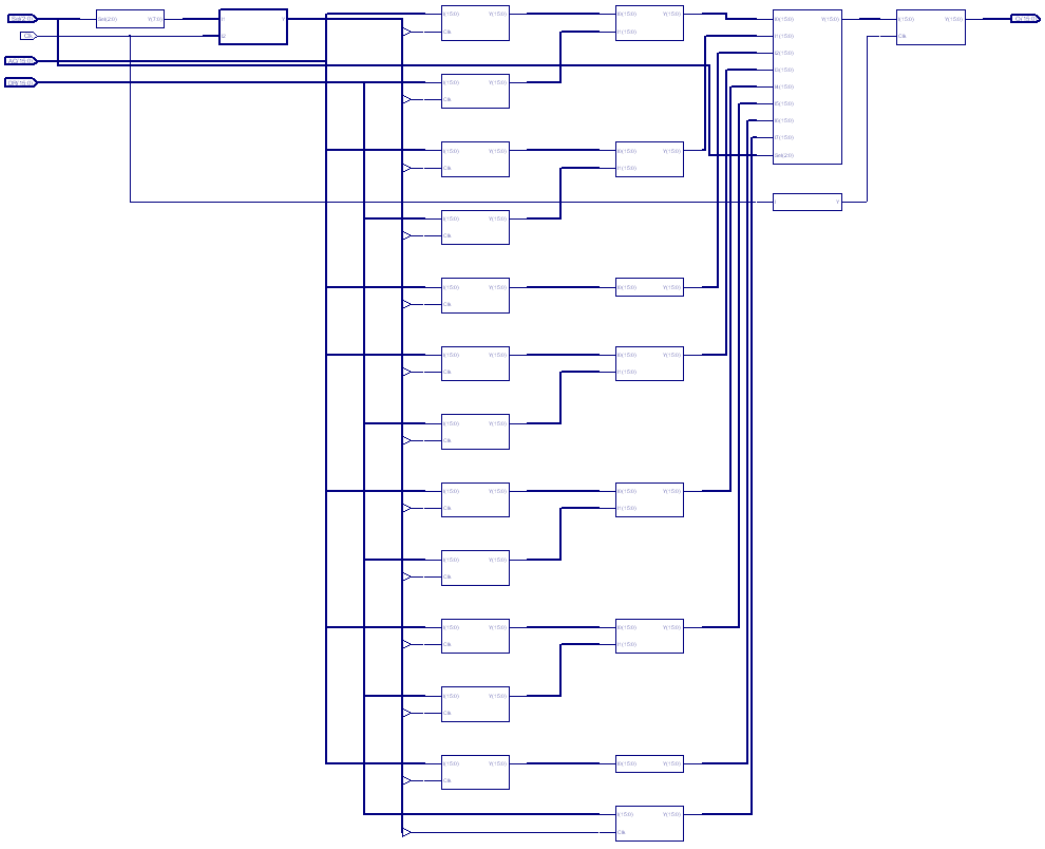
Testiranje modifikovanog *ALU*-a je izvedeno pomoću *TestBench*-a identičnog kao i kod predhodne verzije *ALU*-a, prikazanog na slici 67. Nakon izvršenog testiranja dobijen je dijagram kao na slici 76.

Sa dijagrama se vidi da se izlazi blokova (*Or_out, And_out, Not_out, Xor_out, Add_out, Sub_out, Neg_out* i *Ld_out*) ne menjaju sa svakom promenom signala na ulazu, što je svakako bolje po pitanju potrošnje. Isto tako, izlaz iz *ALU*-a *O* ne prihvata nepotrebne promene na izlazu multipleksera *O_in.*



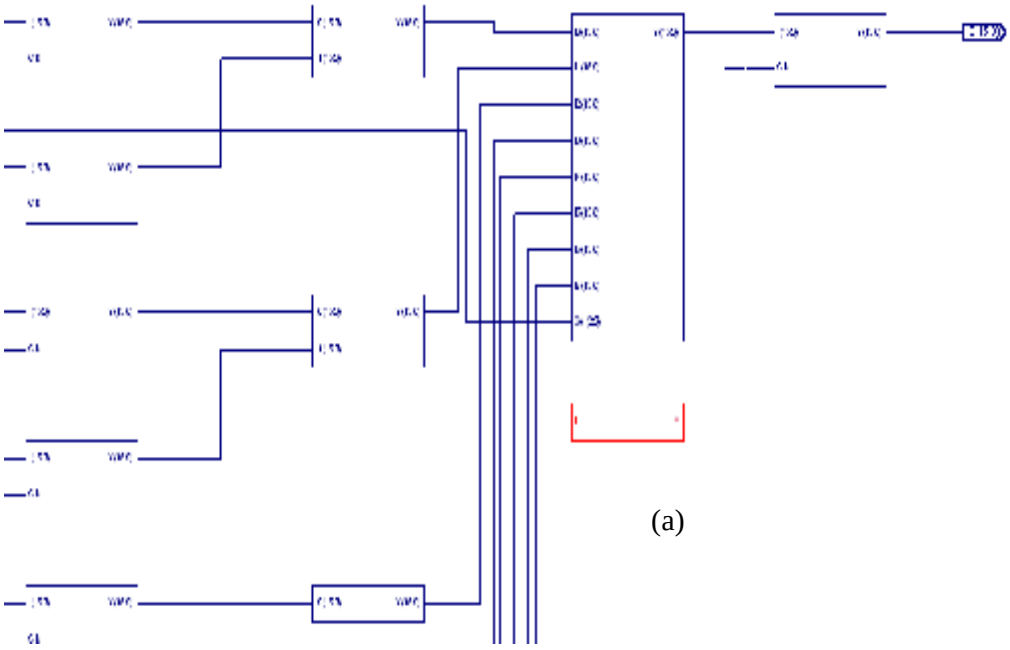
Slika 76• *Vremenski dijagram dobijen nakon izvršenja TestBencha*

Nakon izvršene sinteze istim alatima kao i u predhodnom slučaju dobijena je šema na *RTL* nivou prikazana na slici 77.

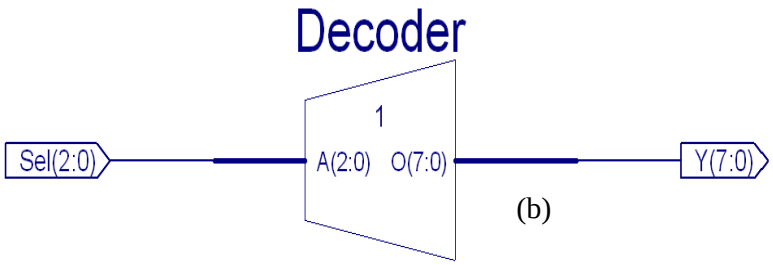


Slika 77• *Izgled na RTL nivou nakon obavljene sinteze*

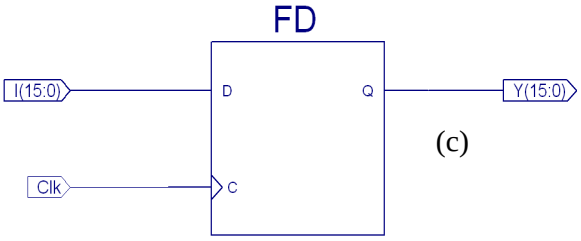
Blok za kašnjenje signala takta je dat na slici 78(a) i obeležen je crvenom bojom. On se sastoji od parnog broja redno vezanih invertora. Na slici 78(b) je dat blok *Dec*, koji predstavlja dekoder tipa 3-u-8 i . Leč je prikazan na slici 78(c).



(a)



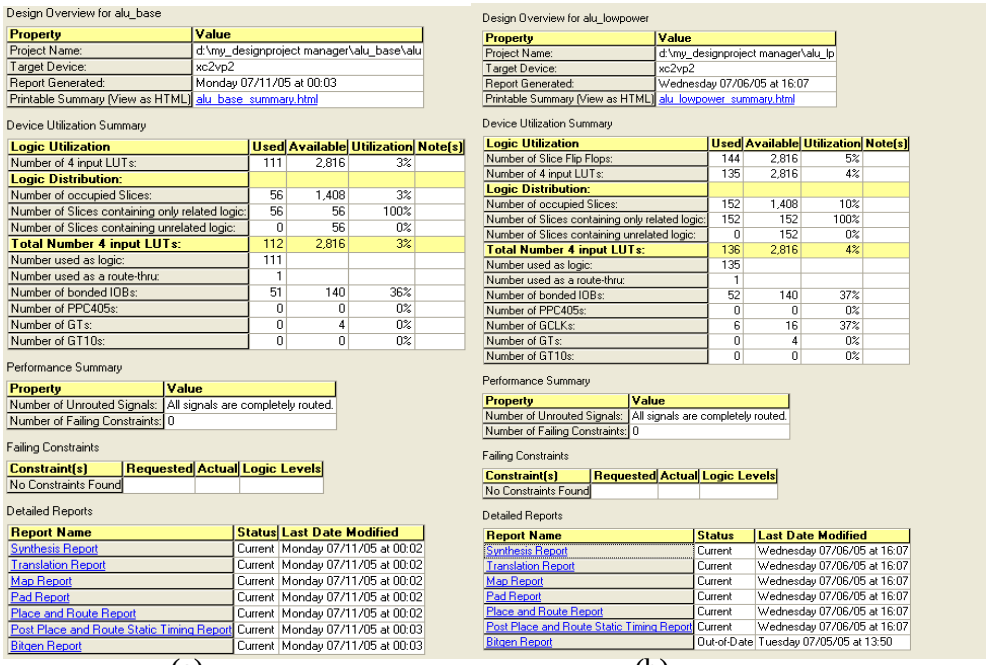
(b)



(c)

Slika 78• *Blokovi koji su dodati radi modifikacije ALU-a*

Report koji je dobijen nakon izvršene sinteze je predstavljen u dodatku (*Alu_lowpower.prj*) na kraju rada. *Design Overview ALU*-a bez modifikacija je dat na slici 79(a), a na slici 79(b) je dat *Design Overview* modifikovanog *ALU*-a.



Design Overview for alu_base

Property	Value
Project Name:	d:\my_designproject manager\alu_base\alu
Target Device:	xc2vp2
Report Generated:	Monday 07/11/05 at 00:03
Printable Summary (View as HTML)	alu_base_summary.html

Device Utilization Summary

Logic Utilization	Used	Available	Utilization	Note(s)
Number of 4 input LUTs:	111	2,816	3%	
Logic Distribution:				
Number of occupied Slices:	56	1,408	3%	
Number of Slices containing only related logic:	56	56	100%	
Number of Slices containing unrelated logic:	0	56	0%	
Total Number 4 input LUTs:	112	2,816	3%	
Number used as logic:	111			
Number used as a route-thru:	1			
Number of bonded IOBs:	51	140	36%	
Number of PPC405s:	0	0	0%	
Number of GTs:	0	4	0%	
Number of GT10s:	0	0	0%	

Performance Summary

Property	Value
Number of Unrouted Signals:	All signals are completely routed.
Number of Failing Constraints:	0

Failing Constraints

Constraint(s)	Requested	Actual	Logic Levels
No Constraints Found			

Detailed Reports

Report Name	Status	Last Date Modified
Synthesis Report	Current	Monday 07/11/05 at 00:02
Translation Report	Current	Monday 07/11/05 at 00:02
Map Report	Current	Monday 07/11/05 at 00:02
Pad Report	Current	Monday 07/11/05 at 00:02
Place and Route Report	Current	Monday 07/11/05 at 00:02
Post Place and Route Static Timing Report	Current	Monday 07/11/05 at 00:03
Bitgen Report	Current	Monday 07/11/05 at 00:03

Design Overview for alu_lowpower

Property	Value
Project Name:	d:\my_designproject manager\alu_lp
Target Device:	xc2vp2
Report Generated:	Wednesday 07/06/05 at 16:07
Printable Summary (View as HTML)	alu_lowpower_summary.html

Device Utilization Summary

Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops:	144	2,816	5%	
Number of 4 input LUTs:	135	2,816	4%	
Logic Distribution:				
Number of occupied Slices:	152	1,408	10%	
Number of Slices containing only related logic:	152	152	100%	
Number of Slices containing unrelated logic:	0	152	0%	
Total Number 4 input LUTs:	136	2,816	4%	
Number used as logic:	135			
Number used as a route-thru:	1			
Number of bonded IOBs:	52	140	37%	
Number of PPC405s:	0	0	0%	
Number of GCLKs:	6	16	37%	
Number of GTs:	0	4	0%	
Number of GT10s:	0	0	0%	

Performance Summary

Property	Value
Number of Unrouted Signals:	All signals are completely routed.
Number of Failing Constraints:	0

Failing Constraints

Constraint(s)	Requested	Actual	Logic Levels
No Constraints Found			

Detailed Reports

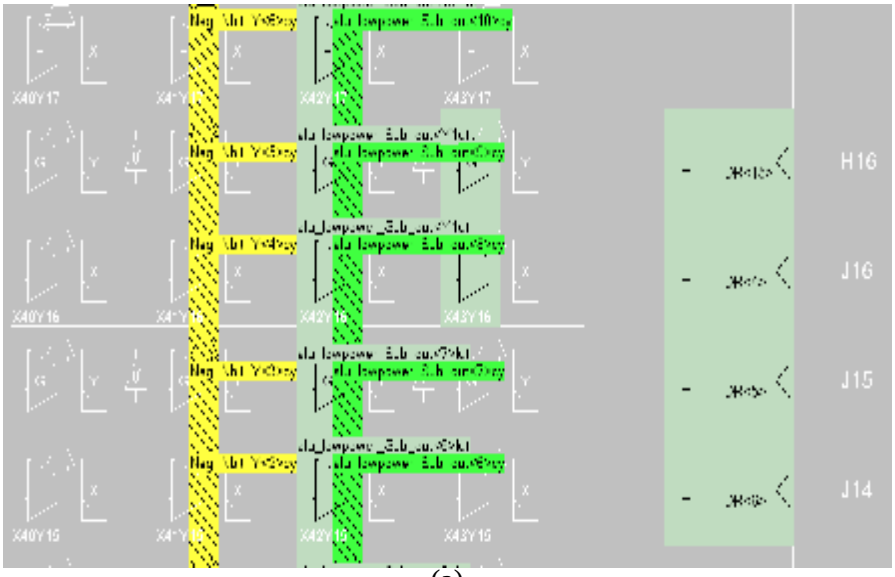
Report Name	Status	Last Date Modified
Synthesis Report	Current	Wednesday 07/06/05 at 16:07
Translation Report	Current	Wednesday 07/06/05 at 16:07
Map Report	Current	Wednesday 07/06/05 at 16:07
Pad Report	Current	Wednesday 07/06/05 at 16:07
Place and Route Report	Current	Wednesday 07/06/05 at 16:07
Post Place and Route Static Timing Report	Current	Wednesday 07/06/05 at 16:07
Bitgen Report	Out-of-Date	Tuesday 07/05/05 at 13:50

(a) (b)

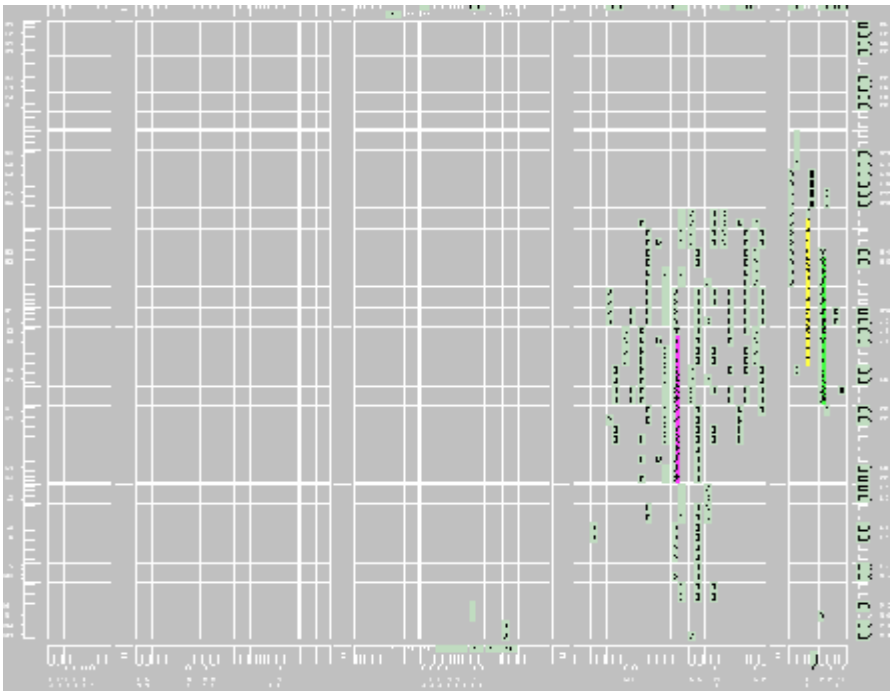
Slika 79• *Overview-ovi: (a)osnovnog ALU-a, (b) modifikovanog ALU-a*

Upoređivanjem *Device Utilization* dela oba *ALU*-a, vidi se da je veći broj logičkih kola upotrebljen kod modifikovanog *ALU*-a. Broj iskorišćenih ćelija kod standardnog *ALU*-a je 56 što čini 3% od ukupnog broja ćelija integrisanog kola, dok je broj ćelija kod modifikovanog *ALU*-a 152 što čini 10% od ukupnog broja ćelija. Samim tim je uvećana i površina na čipu koju zauzima logika za modifikaciju.

Implementacija je izvršena na identičan čip kao i kod standardnog *ALU*-a. Slika 80(a) ilustruje implementaciju modifikovanog *ALU*-a, dok je kompletna šema na tehnološkom nivou data na *CD*-u. Slika 80(b) prikazuje izgled čipa koji je dobijen pomoću ***Floorplaner***-a nakon faze mapiranja [51].



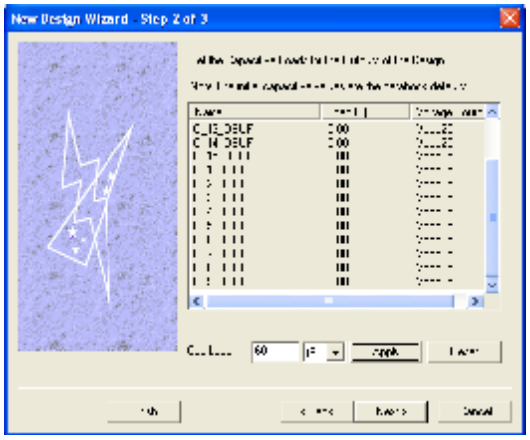
(a)



(b)

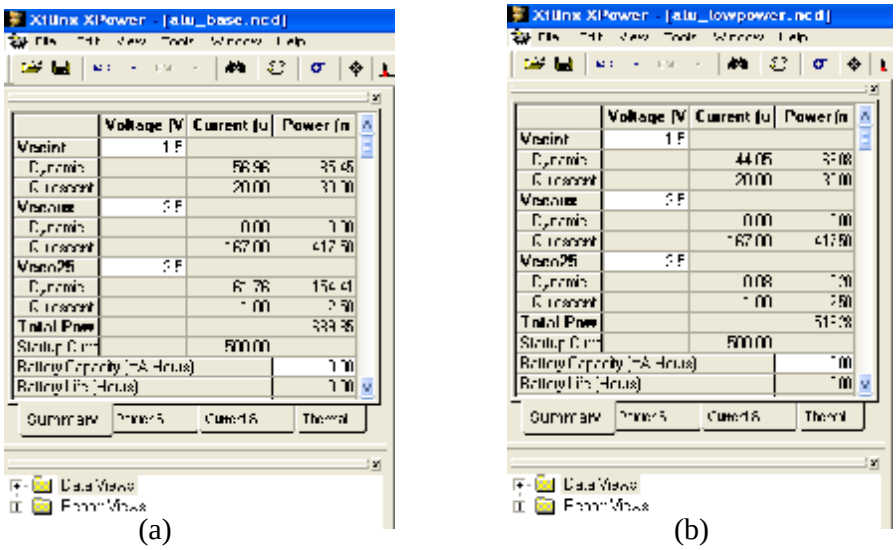
Slika 80• *Izgled modifikovanog ALU-a na čipu*

Poslednja faza pre generisanja fajla za programiranje čipa je analiza snage pomoću ***XPower*** alata. Podešavanje parametara je identično kao i kod analize snage standardnog *ALU*-a (vidi detalj sa slike 81).



Slika 81• *Detalj podešavanja parametara za analizu potrošnje*

Kompletan *report* nakon analize potrošnje je dat u dodatku kao *Alu_lowpower.ncd.* Na slici 82(a) i 82(b) su prikazani, poređenja radi, izveštaji analize potrošnje za osnovni i modifikovani *ALU* respektivno.



(a) (b)

Slika 82• *Report nakon izvršene analize: (a)osnovni ALU, (b) modifikovani ALU*

Na osnovu izvedenih analiza možemo primetiti da je statička potrošnja ista u oba slučaja, dok se dinamička drastično smanjila (sa 154.41 mW na 0.2 mW). Ovo smanjenje dinamičke potrošnje utiče i na smanjenje ukupne potrošnje sa 690 mW na 516 mW što predstavlja uštedu snage od 25%. Pri proračunu potrošnje, u oba slučaja, je usvojeno da se menja 50% ulaza. Ovakva procena daje približno realnu sliku, međutim, u nekim slučajevima ušteda može biti veća.

6 Zaključak

Opšte tendencije *VLSI* kola su da ona postaju sve složenija i da broj tranzistora po čipu raste, a shodno tome raste i disipacija. Povećanjem disipacije integrisanih kola smanjuje se pouzdanost kola, pa se zato teži smanjenju snage.

Ovaj rad upravo razmatra načine za rešavanje ovog problema. Konkretno je razmatrana potrošnja jedne *ALU* jedinice, koja predstavlja ključni gradivni blok u stazi podataka procesora.

Praktičan deo rada sastojao se u pisanju *VHDL* koda 16 bit-nog *ALU*-a, koji će imati manju potrošnju u odnosu na standardnu *ALU* jedinicu. Oba *ALU*-a realizovana su na identičnim čipovima iz *Xilinx*-ove familije *Virtex™ 2p* sa oznakom *xc2vp2fg256-6*. Pri realizaciji modifikovanog *ALU*-a korišćene su metode redukcije potrošnje kao što je izolacija operanada i preračunavanje ulaza.

Za procenu potrošnje korišćen je alat *XPower*. Nakon izvršene analize potrošnje, koja je sprovedena pod identičnim uslovima za oba *ALU*-a, došlo se do značajnog smanjenja potrošnje, od 25%, kod modifikovanog *ALU*-a.

Nedostatak analiziranog standardnog *ALU*-a je taj što su sve njegove funkcionalne jedinice aktivne, samim tim imaju određenu potrošnju, bez obzira na to koja se operacija izvršava. Modifikacija se bazira na ideji da samo jedna funkcionalna jedinica bude aktivna u toku izvršenja operacije dok će ostale, koje nisu aktivne, biti sprečene. Sprečavanjem rada ovih funkcionalnih jedinica postiže se značajna ušteda u potrošnji.

Sredstva za projektovanje pored optimizacija za veće brzine i manje površine obavezno treba da sadrže i sredstva za optimizaciju potrošnje.

Da bi se napravilo *IC* kolo neophodno je proći kroz sve nivoe projektovanja od sistemskog do tehnološkog. Uštede energije se mogu postići na svim nivoima projektovanja. Potrošnju je najbolje sagledati u veoma ranim fazama projektovanja, kada je mogućnost za uštedu maksimalna. Pravilnim *VHDL* kodiranjem još u ranim fazama projektovanja, na ***RTL*** nivou, postižu se značajne uštede u potrošnji snage.

Štednja energije se zasniva na smanjenju dinamičke i statičke portošnje kola. U ukupnoj potrošnji *IC* kola dinamička potrošnja zauzima veći deo. Korišćenjem metoda kao što su: skaliranje napona napajanja, redukcija gličeva, dozvola/zabrana taktovanja, pravilna realizacija automata, staza podataka i ostalih metoda navedenih u radu postižu se značajne uštede energije. Metode minimizacije dinamičke potošnje daju veoma dobre rezultate, tako da je sada dominantna statička potrošnja.

Za poboljšanje performansi sistema, po pitanju potrošnje, mogu se koristiti razne softverske metode kao i specijalno dizajnirana *Power-Friendly* logička kola kao što su *CPL- **C**omplementary **P**ass-gate **L**ogic-a* ili korišćenje *Triple-Oxide* tehnologije.

.

7 Dodatak

8 Reference

[1] Chandrakasan A., Sheng S., Brodersen R., *Low Power CMOS Digital Design*, IEEE Journal of Solid State Circuits, Vol. 27, No.4, April 1992, pp475-483

[2] Poppen F., *Low power Design Guide*, Oldenburger for Schungs und nutwiklungsinstitut fur Informatik Werkzeuge, http://www.lowpower.de version 30.06.00 eMail: Poppen@OFFIS.De

[3] CADENCE Technical Paper, *EMPOWERING DESIGN FOR QUALITY OF SILICON* – Low Power Design Flow.

[4] Chandrakasan A., Sheng S., Brodersen R., *Low power CMOS digital design,* IEEE Journal of Solid State Circuits Vol. 27. No. 4. April. 1992.

[5] Mahmoud Ben Naser and Csaba Andras Moritz, *A Step-by-Step Design and Analysis ofLow Power Caches for Embedded Processors*, Department of Electrical and Computer Engineering University of Massachusetts, Amherst Jan 21, 2005.

[6] G. Gerosa et al., *A2.2-w 80MHz superscalar RISC microprocesor,* IEEE J. Solid-State Circuits Vol. 29, No.12 Dec.1994.

[7] C. Piguet et al., *Low power design of 8-bit embedded CoolRISC microcontroler cores*, IEEE J. Solid-State Circuits Vol. 32, No 7.July 1997.

[8] M. Farrahi, G. E. Tellez and M. Sarrafzadeh, "*Memory segmentation to exploit sleep mode operation*", Proceedings of the Design Automation Conference, pp. 36-41, June1995

[9] S. Svilan, J. B. Burr, and G.L. Tyler, *Effects of elevated temperature on tuntable near-zero threshold CMOS,* Proc. Symp. On Low-power Electron and Design, 2001.

[10] Synopsys, Switching Activity Interchange Format (SAIF), http://www.synopsys.com

[11] K. Flautner, D. Flynn and M. Rives, *Acombined hardware-software approach for low-power SoCs,* Proc. Design Conf., 2003.

[12] S. Martin, K. Flautner, T. Mudge and D. Blaauw, Combined dynamic voltage scaling and adaptive body biasing fof lower power microprocessors under dynamic workloads, Proc. Int. Conf. on Computer-Aided Design, 2002.

[13] D. Tamura, B. Pangrle and R. Maheshwary, *Techniques for energy-efficient SoC design,* http://www.eedesign.com

[14] D. E. Lackey, S. Gould, T. R. Bednar, *Managing power and performance for SoC designs using voltage islands,* Proc. Int. Conf. Computer-Aided Design, 2002.

[15] K. Usami, M. Igarashi, F. Minami, T. Ishikawa, M. Kawakawa, *Automated low-power technique exploiting multiple supply voltages applied to media processor,* IEEE J. Solid-State Circuits Vol.33, No.3, 1998.

[16] L. Wei, K. Roy, and V. De, Low power, low *voltage CMOS design techniques for deep submicron Ics,* Proc. Int. Conf. on VLSI Design, 2000.

[17] F. Ishihara, F. Sheikh and B. Nikolić, *Level conversion for dual supply sistems,* Proc.Int. Symp. On Low-power Electron and Dessign, 2003.

[18] Synopsys, Library Compiler User Guide, Modeling Timing and Power Technology Libraries.

[19] N. Raghavan, V. Akella and S. Bakshi, *Automatic insertion of gated clocks at register transfer level,* Proc. 12th Int. Conf. on VLSI Design, January 1999.

[20] Synopsys, Power Compiler Design Manual

[21] Sequence Design Ltd., *Power Theater*

[22] F. Theeuwen and E. Seelen, *Power reduction through clock gating by symbolicmanipulation,* Proc. Symp. Logic architecture Design, Dec.1996.

[23] M. Ohnishi, A. Yamada, H. Noda and T. Kambe, A method of redundant locking detection and power reduction at RT level design, Proc.1997 Int. ymp. Low power Electronics and Design, Monterey, CA, Aug. 1997.

[24] C. Arm, J.M. Masgonty and C. Piguet, *Double-latch clocking sheme for low power IP cores,* PATMOS 2000, Goettingen, Germany, Sept.,2000.

[25] T. Scheider, VHDL Methodologie de Design et Techniques Avanceses, Dunod, Paris, France, 2001.

[26] L. Benini and G. DeMicheli, Transformation and synthesis of FSM for low power and gated-clock implementation, ACM/IEEE Int. Conf. on CAD, Nov. 1995.

[27] C. Tsui and M. Pedram, Low power state assigment targeting two and multi-evellogic implrmentation, ACM/IEEE Int. Conf. on CAD, Nov. 1994.

[28] R. Shelar and M.P. Desai, Ortogonal partitioning and gated-clock architecture for low power realization of FSMs, IEEE ASIC/SOC 2000, Sept. 2000.

[29] L. Benini, G. DeMicheli and F. Vermulen, *Finite-state machine partitioningfor low power,* IEEE ISCAS 98, May 1998.

[30] M. Alidina, J. Monteiro, S. Devadas, A. Gosh and M. Papaefthymiou, *Precomputation-based seqential logic optimization for low power,* Proc. 1994 Int. Comput. Aided Design, San Jose, CA, Nov. 1994.

[31] V. Tiwari, S. Malik and P. Ashar, *Guarded evaluation:pushing power management to logic synthesis/design,* Proc. Low power Design Symp., dana Point, CA, Apr. 1995.

[32] H. Kapadia, L. Benini and G. De Micheli, *Reducing switching activity on datapath buses with control-signal gating,* IEEE J. Solid-State Circuits, Vol.34, Mar. 1999.

[33] G. K. Yeap, *Practical Low-Power Digital VLSI Design,* Kluwer Academic Publishers, Dordrecht, 1998.

[34] G. De Micheli, *Synthesis and Optimization of Digital Circuits,* McGraw-Hill, New York, 1994.

[35] M. Hikari, H. Kojima, et al., *Data-dependent logic swing internal bus architecture for ultralow power LSIs,* IEEE J. Solid-State Circuits, Vol.30, No4, Apr.1995.

[36] H. Yamauchi, H. Akamatsu and T. Fujita, An asimtomatically zero-power change-reducing bus architecture for battery-operated ultra-high data rate ULSIs, IEEE J. Solid-State Circuits, Vol. 30, No. 4, Apr.1995.

[37] L. Benini, *Designing advanced NoCs architectures,* Int. Seminar on Application-Specific Multiprocessor SoC, Chamonix, France, July.,2003.

[38] M. Stan and W. Bureleson, *Bus-invert coding for low-power IO,* IEEE Trans. on VLSI Syst., Vol.3, No.1, Mar.1995.

[39] M. Muench, B. Wurth, R. Mehra and J. Sproch, *Automating RT-level operand isolation to minimize power consumption in datapaths,* Proc. Design Automation and test in Europe, 2000.

[40] F. najm, Transition density, a stochastic measure of activity in digital circuits, Proc. Design Automation Conf.,1991.

[41] M. Borah, R. Owens and M. Irwin, *Transistor sizing for low-power CMOS circuits,* Trans. on Computer-Aided Design, June, 1996.

[42] O. Coudert and R. Haddad, *Integrated resynthesis for low power,* Proc. Int. Symp. On Low-Power Electron and Design, 1996.

[43] V. Tiwari, P.Ashar and S. malik, *Technology mapping for low power,* Proc.Design Automation Conf.,1993.

[44] Actel Corporation, J. Alexander, *VHDL Design Tips and Low Power Design Techniques*. MAPLD 2004.

[45] Library Compiler User Guide: Modeling Timing and Power Technology Libraries, Synopsys.

[46] J. Rabaey, A. Chandrakasan, and B. Nikolić, *Digital Integrated Circuits, 2nd ed.,* 2003, Prentice Hall/Pearson.

[47] Power Compiler Reference Manual, Synopsys.

[48] PrimePower Reference Manual, Synopsys.

[49] http://www.xilinx.com

[50] Virtex-II Pro and Virtex-II pro X Platform FPGAs: *Complite Data Sheet,* Product Specification DS083 (v4.3), June, 2005

[51] Floorplaner Release Version: 7.1 i Copyright © 1995-2005 Xilinx, Inc.