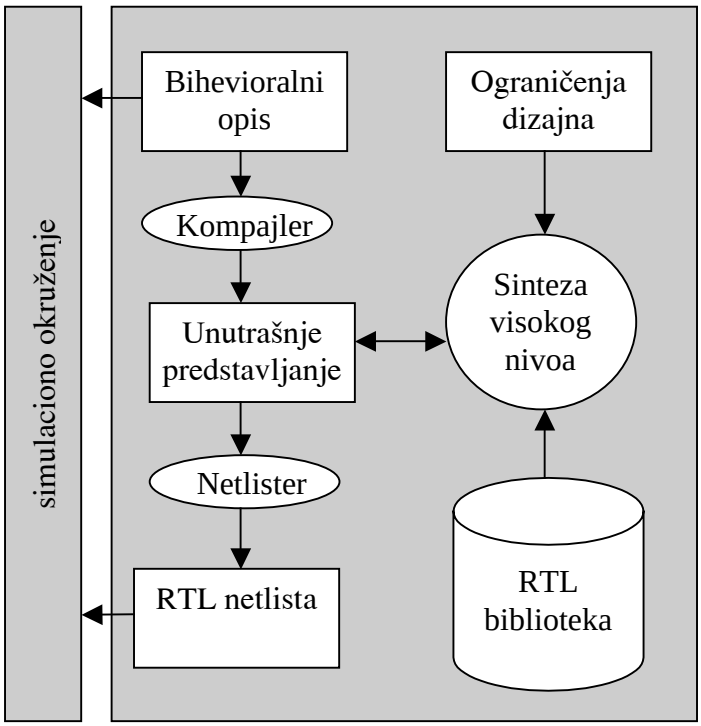
VLSI TEHNOLOGIJA: UVOD U SINTEZU NA VISOKOM NIVOU

Projektanti integrisanih kola mogu da primene metodologiju za opis-i-sintezu ASIC kola na sledećih nekoliko nivoa abstrakcije:

- **gejt nivou (*gate level*)**– klasičnim postupcima prekidačke algebre moguće je sintetizovati logiku funkcionalne i upravljačke jedinice.
- **RTL nivou (*register transfer level*)**– opisuje se ponašanje ASIC kola pomoću programa, algoritama, dijagrama toka, toka podataka, skupa instrukcija, ili generalizovanog FSM-a kod koga svako stanje obavlja neko proizvoljno složeno izračunavanje.
- **visokom nivou (*behavioral level*)_**– predstavlja sekvenca zadataka koja transformiše prezentaciju ponašanja u RTL dizajn. Dizajn čine funkcionalne jedinice kakve su ALU jedinice i multiplekseri, elementi za pamćenje kakvi su memorije i registarska polja, i sprežne jedinice kakve su multiplekseri i magistrale.

Opšti oblik strukture sistema za sintezu ASIC kola na visokom nivou (*high level synthesis*) prikazan je na Slici 1.



Slika 1. Opšti oblik sistema za sintezu ASI kola na visokom nivou .

Kompilator vrši konverziju opisa na nivou ponašanja (*behavioral level*) u internu prezentaciju. RTL biblioteka sadrži fizičke i simulacione modele komponenata koji se koriste u toku sinteze. *Netlister* generiše konačnu RTL strukturu, koju čini netlista RTL komponenata i simulacioni model za svaku komponentu. Da bi verifikovali korektnost sintetizovanog dizajna, projektant može prvo, da zada i simulira opis-na-ulazu, zatim da generiše netlistu pomoću simulacionog okruženja, i na kraju da je pridoda sistemu za sintezu visokog-nivoa.

Glavne prednosti sinteze visokog-nivoa su poboljšana produktivnost i bolja efikasnost u procesu projektovanja (projektovanje se vrši na abstraktnijem nivou, gde projektanti mogu da specificiraju, modeliraju, verifikuju, sintetizuju, i otklanjaju greške u dizajnu za kraće vreme).

Opisivanje ulaza kod sinteze visokog nivoa

Opis-ulaza (odnosi se na formu unosa dizajna) specificira namenu dizajna. Opis se obično zadaje u algoritamskoj formi i ne sadrži strukturnu intofmaciju o tome koji se tipovi komponenata koriste i na koji način se te komponente medjusobno povezuju. Ne postoji i informacija koja ukazuje na strukturu kola, kao što je to koliko se protočnih stepeni koristi i druge detalje. Projeketanti obično kreiraju ulazni-opis na nekom od specijalizovanih HDL jezika kakvi su VHDL, Verilog i dr. VHDL je svakako najšire korišćeni jezik za opis hardvera. Ovaj jezik podržava sekvencijalne i konkurentne iskaze, uslovne instrukcije, petlje, procedure i funkcije. Ipak treba naglasiti da on eksplicitno ne podržava hardversku protočnost, prekide i hijerarhijsko ponašanje.

Na slici 2 prikazan je jedan VHDL opis. Opis čini port i deklaracija promenljivih, a zatim jedan proces kojim se zaokružuje ponašanje dizajna. Proces ima pet lokalno promenljivih: *C*, *D*, *E*, *F* i *Tajmer*. Proces čeka sve dok eksterni ulaz *X* ne postane 1 a zatim izvršava niz iskaza dodele ako je vrednost *Tajmer* različita od 0.

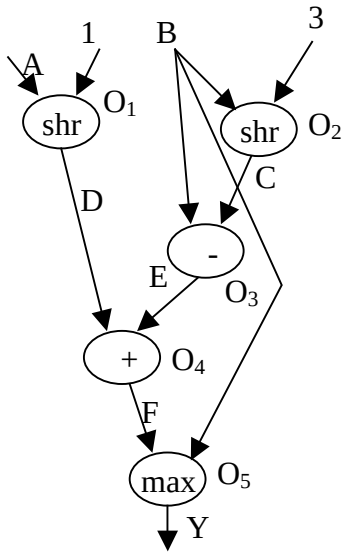
```
Entity sqrt_deo is
port (A,B: in integer;
        X: in bit;
        Y: out integer);
 End sqrt_deo;

  Architecture arch of sqrt_deo is
Begin
       P0: process
          variable C, D,E,F: integer ;
          variable Tajmer: integer ;
begin
   wait until X = 1;
          if (Tajmer !=0) then
                      D := shr (A,1);
                      C := shr (B,3);
                      E := B-C;
                      F := D+E;
                      Y := max(F,B);
                    else
                     Tajmer := Tajmer-1;
             end if;
             end process;
 end architecture arch;
```

Slika 2 Jednostavan VHDL opis

Interna prezentacija

Sistemi za sintezu integrisanih kola na visokom nivou kompajliraju (prevode) opis ponašanja u internu prezentaciju. Svi zadaci koji se u daljem toku obrade (vidi Sliku 1) odnose na sintezu za polaznu osnovu uzimaju ovu prezentaciju. Postoji više tipova internih prezentacija. Najbolja je ona kojom se najskladnije ipisuje problem. Tako na primer za probleme digitalnog procesiranja signala najbolja prezentacija je DFG (*Data Flow Graph*). DFG čini skup čvorova, pri čemu svaki čvor odgovara jednoj operaciji u početnom opisu (u konkretnom slučaju VHDL opisu). Dva čvora *Oi* i *Oj* se povezuju pomoću

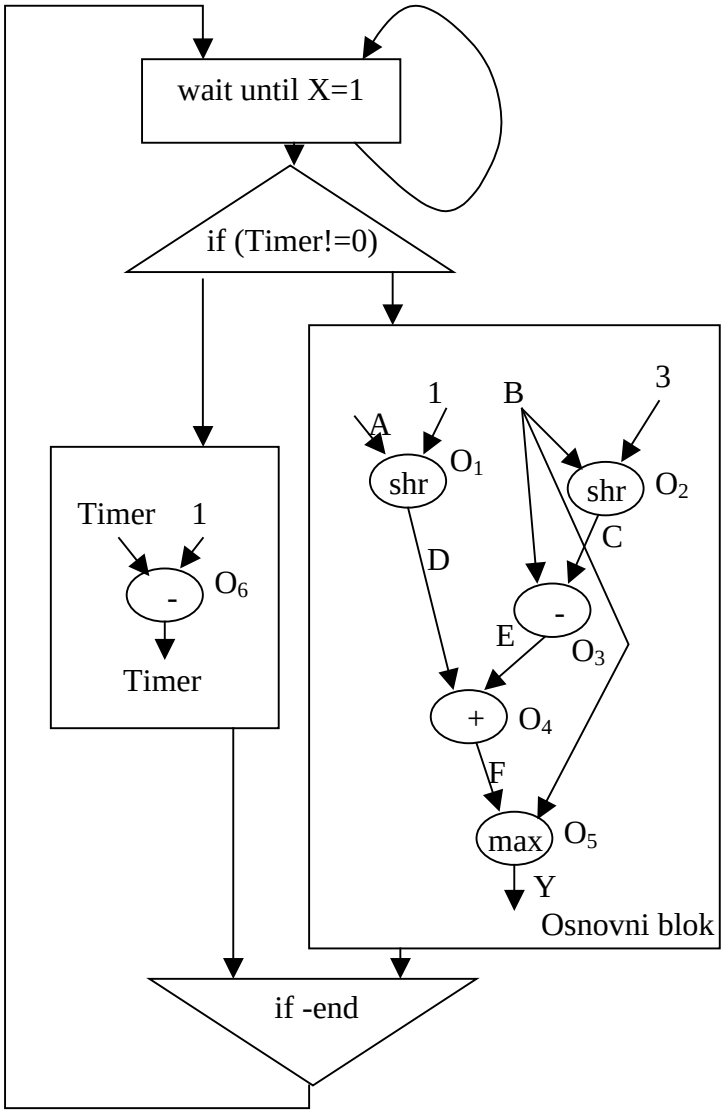
potega ako izmedju njih poostoji zavisnost po podacima. (To znači da je rezultat operacije *Oi* ulaz za operaciju *Oj*). Drugim rečima, poteg zavisnosti koji povezuje *Oi* i *Oj* ukazuje da se operacija *Oj* ne može izvršiti pre operacije *Oi.* S obzirom da se DFG prezentacija zasniva isključivo na zavisnosti po podacima, njen opis najbolje odgovara paralelnoj prezentaciji. Na slici 3 prikazan je DFG za izraz Y=max((Ashr1)+(B-(Bshr3)),B).



Slika 3 DFG prezentacija

DFG nije dobra forma za prezentaciju reaktivnog ponašanja sistema sa ugranjenim mikroračunarem (*embedded systems*), kod kojih se upravljačka sekvenca bazira na spoljnim uslovima. Kod ovakvih slučajeva pored zavisnosti po podacima mora se predstaviti i upravljačka zavisnost. Ovo se izvodi na taj način što se DFG proširuje sa upravljačkim čvorovima. Tipičan primer koji se odnosi na proširenje DFG-a je CDFG (*control data flow graph*). CDFG omogućava prezentaciju upravljačkih struktura kakve su grananja i petlje, a pored toga sadrži specijalne čvorove za prezentaciju *if* uslova, *case* i *loop* konstrukcija kao i sekvenci za izračunavanje. Na slici 4 prikazana je CDFG prezentacija za VHDL opis sa slike 2. U konkretnom slučaju izrazom Y=max((Ashr1)+(B-(Bshr3)),B) izračunava se vrednost *Y* samo ako je spoljni uslov *X* potvrdjen (*X*=1) i ako je *Tajmer* razlličit od 0. Čvor *if-begin* proverava da li je *Tajmer* različit od 0, a *if-end* čvor se odnosi na kraj *branch* iskaza. Izračunavanje koje prati iskaz ugradjuje se u osnovni (*dataflow*) blok.

CDFG prezentacijom zadržava (očuvava) se upravljačka struktura specificirana od strane projektanta. Kod CDFG-a svaki blok iskaza tipa dodela (iskazi koji pripadaju početno zadatom opisu na nivou ponašanja) se predstavlja kao poseban osnovni blok. Informacija koja se odnosi na zavisnost po podacima predstavlja se samo u okviru osnovnog bloka, dok zavisnosti po podacima koje prelaze granice bloka se ne predstavljaju eksplicitno. Sistem za sintezu koji direktno radi (vrši sintezu) na osnovu DFG prezentacije mora da sadrži strukturu osnovnih blokova. Tako na primier, dve operacije u dva sekvencijalna osnovna bloka se nikad ne izvršavaju zajedno i pored toga što izmedju njih ne postoje bilo kakve realne zavisnosti. Ovo je jedan od glavnih nedostataka kada se CDFG prezentaciji direktno koristi za sintezu.



Slika 4 CDFG prezentacija

Model sinteze na visokom nivou

Logička sinteza se zasniva na formalizmu *Boole*-ove algebre, dok se sekvenicjalna sinteza bazira na FSM modelu. Kod sinteze na visokom nivou FSM model se proširuje na taj način što se uvodi dodela promenljivih. FSM model čine skup stanja, skup prelaza iz jednog stanja u drugo, i skup akcija koje prate ova stanja ili prelaze. Formalnije kazano FSM je petorka oblika

$$< S, I, O, f: S \times I, h: S \times I \rightarrow O >$$

gde je: *S* -skup stanja; *I*-skup ulaznih vrednosti; *O*-skup izlaznih vrednosti; *f* i *h*-su funkcije naredno-stanje i izlaz koje preslikavaju unakrsni proizvod od *S* i *I* u *S* i *O*, respektivno.

Funkcije *f* i *h* se mogu specificirati *Boole*-ovim jednačinama, tabelama stanja, ili dijagramima stanja. FSM model je efikasan kada je broj stanja do reda nekoliko stotina. Nakon toga model postaje nerazumljiv za projektante. Šta više i komponente manje složenosti kakvi su U/I interfejsi ili kontroleri magistrale, kada se broje svi memorijski elementi (vrednosti zapamćenih u memoriji), mogu imati po nekoliko stotinu stanja. Sa ciljem da se FSM model prilagodi kompleksnim rešenjima, uvodi se skup

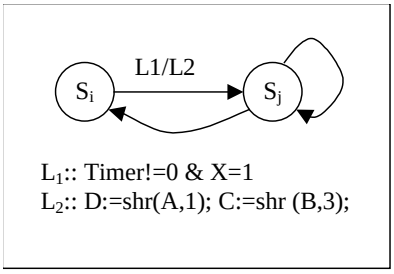
celobrojnih i FP (*floating point*) promenljivih koje se čuvaju (memorišu) u registrima, registarskim poljima, i memorijama. Tako na primer promenljivom tipa 16-bitna celobrojna vrednost (*integer*) može se predstaviti 2^{16} ili 65 536 različitih stanja, što znači da se uvodjenjem 16-bitne promenljive kod FSM modela broj stanja redukuje za 65 536. Korišćenje promenljivih ukazuje na to da se FSM model proširuje na FSMD, tj. FSM sa stazom podataka (*FSM with the datapath*).

Formalno FSMD se definiše na sledeći način: Skup memorijsko promenljivih *VAR*, skup izraza EXP = {f(x,y,z,...) | x,y,z,..., ∈EVAR}, i skup upisa (dodela) u memoriju A = {x ⇐ e | X ∈EVAR, e ∈ EXP} . Definiše se još i skup statusnih signala kao logički odnos izmedju dva izraza iz skupa EXP, kao STAT = { Rel (a,b) | a,b ∈ EXP} . Na osnovu ovih defnicija FSMD se definiše kao petorka

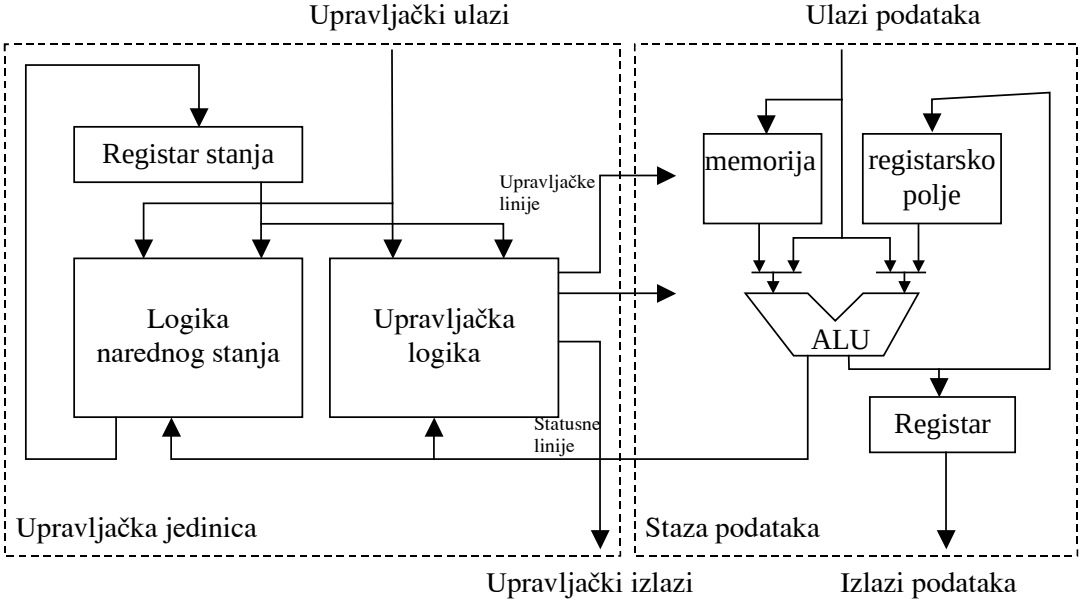
$$< S, I \times STAT, O \times A, f, h >$$

U ovom slučaju skup stanja *S* je proširen na skup ulaznih vrednosti koji sadrži izraze na osnovu kojih se odredjuje status, kao i izlazni skup koji uključuje dodele vrednosti memorijskim elementima. Funkcije *f* i *h* se definišu kao funkcije preslikavanja na sledeći način, $S \times (I \times STAT) \rightarrow S$ i $S \times (I \times STAT) \rightarrow (O \times A)$, respektivno.Na ovaj način, naredno stanje i izlaz FSMD-a ne zavisi samo od tekućeg stanja i spoljnih signala nego takodje i od internih statusnih signala koji ukazuju da li je relacija izmedju veličine staze podataka istinita (*true*) ili lažna (*false*).

FSMD izračunava (odredjuje) nove vrednosti promenljivima koje se čuvaju u memorijskim elementima staze podataka, ali takodje FSMD dodeljuje vrednosti spoljnim signalima. Ovo je prikazano na delu (odsečku) dijagrama stanja sa Slike 5, gde u izrazu *L1* učestvuju promenljive tipa spoljna i status, a *L2* sadrži promenljive tipa dodela.



(a)



(b)

Slika 5 FSMD model (a), i implementacija (b)

FSMD je univerzalni model kojim se mogu predstaviti sva hardverska rešenja (dizajna). Ovaj model može podjednako uspešno da predstavi kako ona rešenja koja su upravljačko dominantna (*control-dominated*) tako i rešenja koja su dominantno zavisna po podacima (*data dominated*). Upravljačko dominatni dizajn se standardno izvodi kao rešenje koje se karakteriše velikom upravljačkom jedinicom i malom stazom podataka. Tipičan primer je serijsko-paralelni konvertor čiju stazu podataka čini jedan pomerački registar. Rešenja koja su dominantno zavisna popodacima, kakvi su FIR filtri, imaju minimalnu upravljačku jedinicu ali veliku stazu podataka koja obavlja operaciju filtriranje.

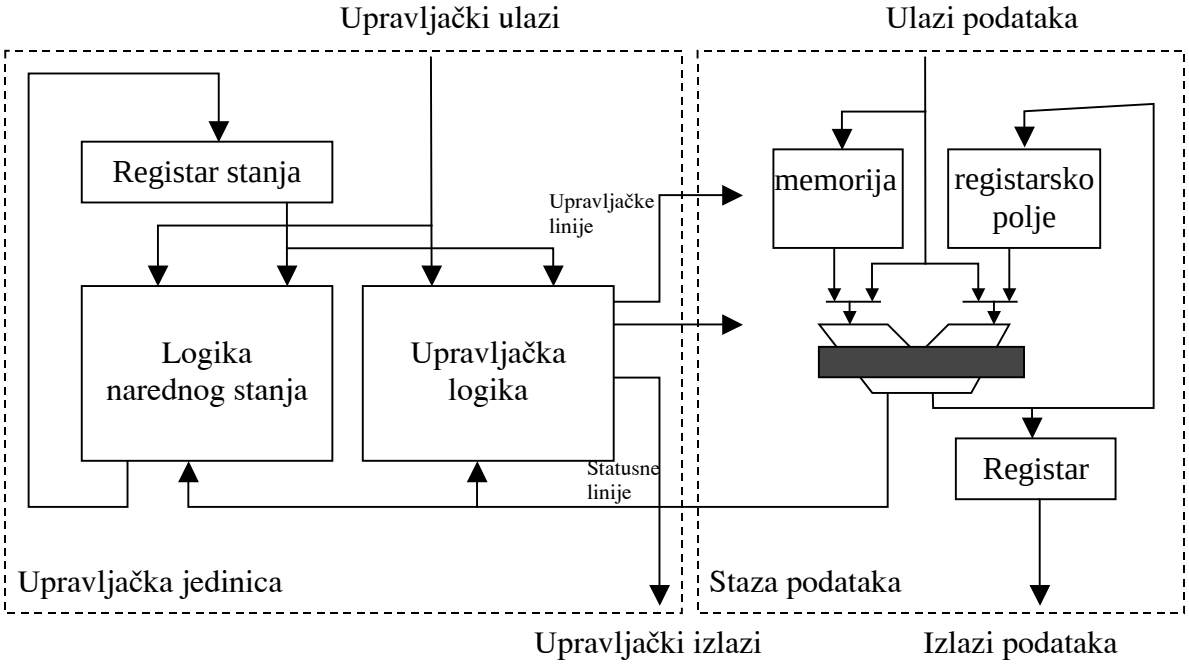
FSMD model uobičajeno se implementira pomoću upravljačke jedinice i staze podataka, pri čemu svako stanje u modelu odgovara taktnom signalu u implementaciji. Upravljačka jedinica je ta koja implementira FSM model, koristeći registar stanja i dve kombinacione logičke mreže koje odredjuju vrednost registra stanja za naredni taktni interval (funkcija narednog stanja *f*, i vrednosti izlaza i upravljačkih signala (izlazna funkcija *h*). Na slici 5b) ove dve kombinacione mreže su prikazani kao dva bloka, logika naredno-stanje i upravljačka-logika, respektivno. Implementaicju staze podataka čini skup memorijskih elemenata (registri, brojači, registarska polja, memorije), skup funkcionalnih jedinica (ALU-ovi, multiplekseri, pomerači, komparatori), i skup sprežnih elemenata (linije, magistrale, multiplekseri).

Protočnom implementacijom FSMD-a povećavaju se performase. Na slici 6 prikazana su tri protočna stila.

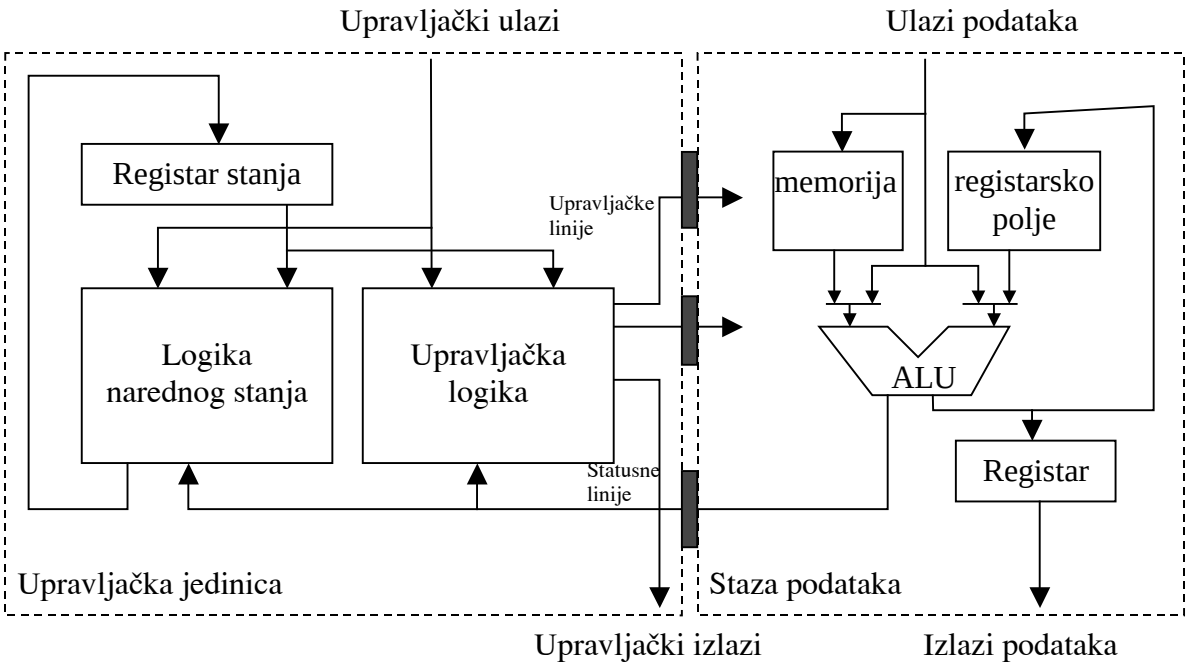
(a) **Protočnost na nivou komponenata** – sa ciljem da se u okviru staze podataka poveća iskorišćenost funkconalnih jedinica, projektanti ove jedinice izvode kao protočne. Kao što se vidi sa Slike 6a) ALU čine dva protočna stepena.

(b) **Protočnost na nivou upravljanja** – u toku svakog stanja FSMD model obavlja tri zadatka. Odredjuje upravljačke signale, izračunava novu vrednost za jednu ili veći broj promenljivih koje se čuvaju u elementima za pamćenje (*storage units*) i odredjuje naredno stanje. S obzirom da se ova tri zadatka ponavljaju u svakom stanju, ona se mogu uprotočiti u tri stepena. Da bi se postigao protočni efekat, stanje na upravljačkim i statusnim linijama je neophodno lečovati kako je to prikazano na Slici 6b). FSM mora da se restruktuira kako bi se u toku operacija grananja (*branch*) usaglasilo protočno kašnjenje.

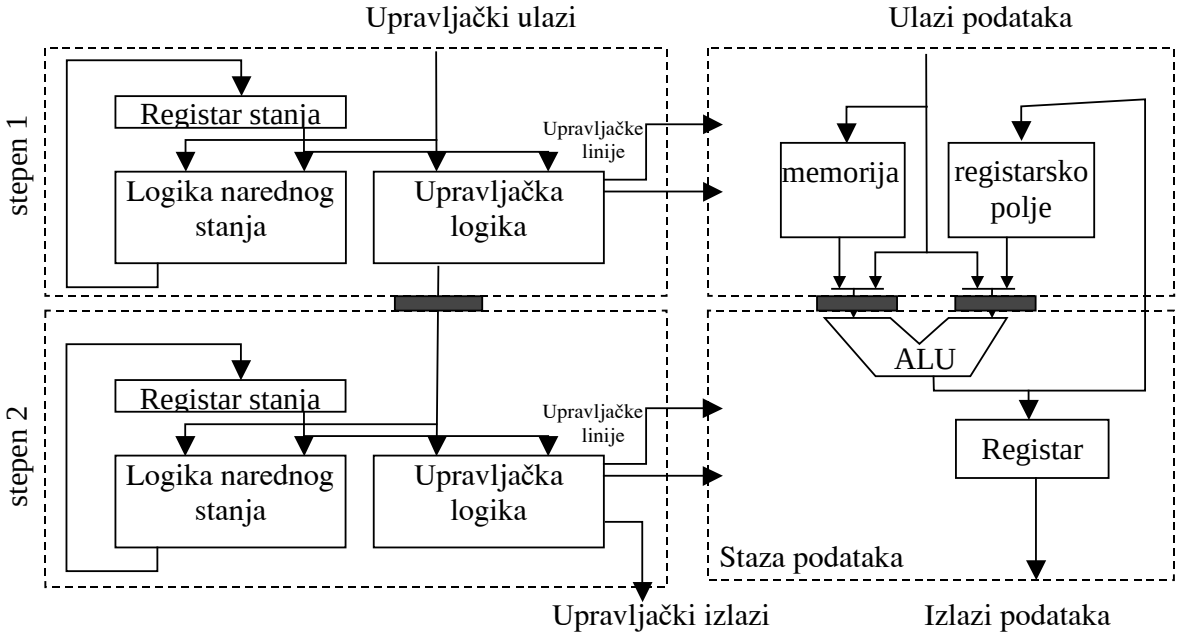
(c) **Protočnost na nivou staze podataka** – kod aplikacija tipa signal-procesiranje FSMD model obavlja istu sekvencu operacija nad svakim elementom ulazne povorke podataka. Sobzirom da se u okviru staze podatka ove operacije izvode repetitivno, moguće je iste učiniti protočnim kako je to prikazano na Slici 6c). Kod ovog stila realizacije protočnosti upravljačka jedinica svakog stepena je obično veoma jednostavna ili skoro da je nema



(a)



(b)



(c)

Slika 6. Protočni FSMD model; (a) protočnost na nivou komponenata; (b) protočnost na nivou upravljanja; i (c) protočnost na nivou staze podataka

Zadaci kod sinteze

Sinteza integrisanih kola na visokom nivou preslikava opis ponašanja u FSMD model pri čemu staza-podataka obavlja aktivnosti koje su u vezi sa dodelom promenljivih dok upravljlačka jedinica implementira upravljačke konstrukcije. S obzirom da FSMD model odredjuje iznos izračunavanja u svakom stanju, neophodno je prvo odrediti broj i tip resursa (memorijske jedinice, funkcionalne jedinice, i jedinice za medjusobno povezivanje) koje će se koristiti u okviru staze podataka. Alokacija (dodela) je zadatak kojim se definišu potrebni resursi za data ograničenja dizajna.

Naredni zadatak kod preslikavanja opisa ponašanja u FSMD model se odnosi na particiju (podelu) opisa ponašanja u stanja (ili upravljačkih koraka) tako da alocirani resursi mogu da izračunaju (izvrše) sve dodele promenljivih u svakom stanju. Proces particije ponašanja u vremenske intervale se naziva planiranje-izvršenja (*scheduling*). I pored toga što se planiranjem-izvršenja svaka operacija dodeljuje pojedinom stanju, ona se ne dodeljuje odredjenoj komponenti. Da bi se ostvarila korektna implementacija neophodno je svakoj promenljivoj dodeliti memorijsku jedinicu, svakoj operaciji funkcionalnu jedinicu, a svakom prenosu sa U/I portova ka jedinicama i izmedju jedinica odgovarajuću jedinicu za medjusobnu spregu (*interconnection unit*). Ovaj zadatak se naziva povezivanje (*binding*), ili deoba resursa (*resource sharing*). Povezivanjem se definiše struktura staze podataka alil ne i struktura upravljačke jedinice. Konačni zadatak se odnosni na sintezu upravljanja, a on se sastoji u redukciji broja kodiranih stanja i projektovanju logika mreže (bloka) kojim se definiše naredno-stanje kao i logike bloka upravljačka-logika koja definiše sve upravljačke signale u okviru upravljačke jedinice (vidi Sliku 5). Postupak sinteze upravljanja zasniva se na poznatim tehnikama o logičkoj sintezi (optimizaciji, minimizaciji) i FSM sintezi (teoriji automata).

Alokacija

Kao zadatak alokacija odredjuje koji će se tip (vrsta) i iznos (broj) resursa koristiti u okviru arhitekture čipa. Alokacijom se takodje odredjuje šema taktovanja, memorijska hijerarhija, i stil protočne obrade.

Cilj alokacije je da učini odgovarajući kompromis izmedju cene dizajna i performansi. Ako početno zadati opis sadrži ugradjeni paralelizam, alokacijom većeg broja hardverskih resursa povećava se površina čipa i cena, ali se takodje stvaraju veće mogućnosti za paralelni rad ili pristup memoriji, tako da se dizajn karakteriše boljim performansama. Sa druge strane, alokacijom manjeg broja hardverskih resursa smanjuje se površina čipa i cena, pri čemu se forsira sekvencijalno izvršenje operacija, tako da za posledicu imamo lošije performanse.

Da bi se ostvario dobar kompromis, postupkom alokacije mora se odrediti tačna površina čipa kao i numerička vrednost performansi. Grubom aproksimacijom odnosa cena-performansi odredjuje se broj funkcionalnih jedinica i broj upravljačkih koraka, respektivno. Da bi imali pouzdaniju procenu odnosa cena-performansi moramo koristiti fizičke modele koji se čuvaju u RTL biblioteci. U Tabeli 1 prikazane su tri komponente: dve ALU implementacije (ALU-F i ALU-S), koje obavljaju operacije tipa *Shift*, *Add* i *Subtract*, i jedna jedinica MAX koja obavlja operaciju *Maximal*. Jedinica ALU-F je brza i obavlja sve operacije za 20ns, dok je jedinici ALU-S potrebno 70 ns da obavi operaciju ali je zbog toga ova jedinica manja i jevtinija.

Tabela 1 RTL biblioteka

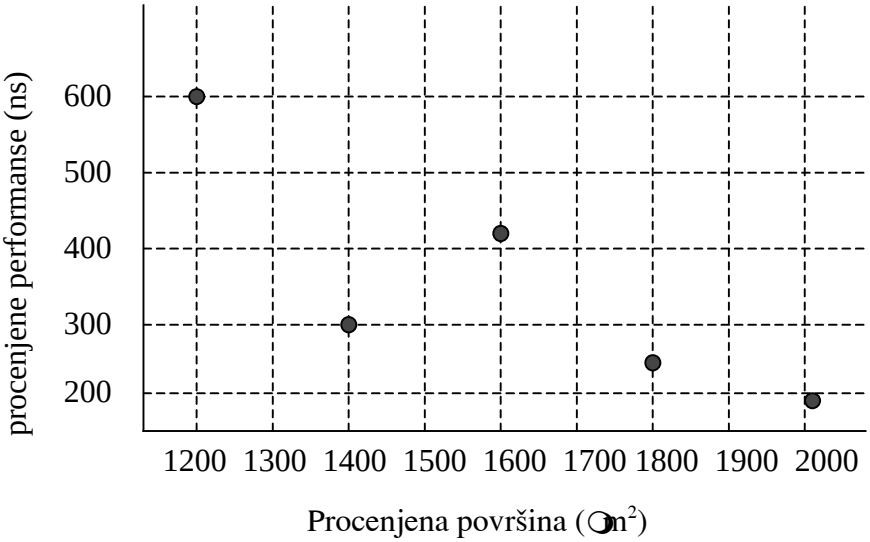
Komponente (operacije)	**kašnjenje (ns)**	**površina (μm^2)**
ALU-F (+/-/shr)	20	600
ALU-F (+/-/shr)	70	400
MAX (mac)	80	800

Koristeći ovu biblioteku, za konkretno analizirani primer, možemo naći kompromis izmedju površine i performansi. U Tabeli 2, za primer sa Slike 3, prikazanoje pet alokacija

Tabela 2 Moguće alokacije sa DFG sa Slike 3

Alokacija	broj ALU-F	broj ALU-S	broj MAX	površina (μm^2)
A	0	1	1	1.200
B	1	0	1	1.400
C	0	2	1	1.600
D	1	1	1	1.800
F	2	0	1	2.000

Jednostavna procena površine za svaku alokaciju čini zbir površina individualnih komponenti biblioteke. Moguće je takodje, kao što je prikazano na Slici 7, proceniti performanse za svaku alokaciju. Kao što se može videti sa Slike 7, za alokaciju A potrebna je najmanja površina ali su performanse najgore. Kod alokacije E performanse su najbolje ali je dizajn najskuplji. Naš izbor za dalju sinatezu jedne od pet alokacija zavisiće od kritičnosti aplikacije (vreme potrebno da se izvrši aplikacija).



Slika 7. Kompromis površina-performansi za moguće alokacije iz Tabele 2

Moguće je nacrtati slične krive na osnovu kojih se odredjuje kompromis izmedju performansi i broja memorijskih jedinica, broja portova kod svake memorijske jedinice, i broja jedinica za sprezanje (povezivanje) elemenata. Odabiranjem odgovarajućih tačaka kod ovih krivih, projektant odredjuje optimalan broj resursa za svaki tip. Današnja savremena CAD sredstva za alokaciju resursa u značajnoj meri pomažu projektantu da napravi pravi izbor dizajna. Ova sredstva raspolažu metrikama koje veoma pouzdano odredjuju površinu, performanse, taktnu frekvenciju, i iskorišćenost resursa.

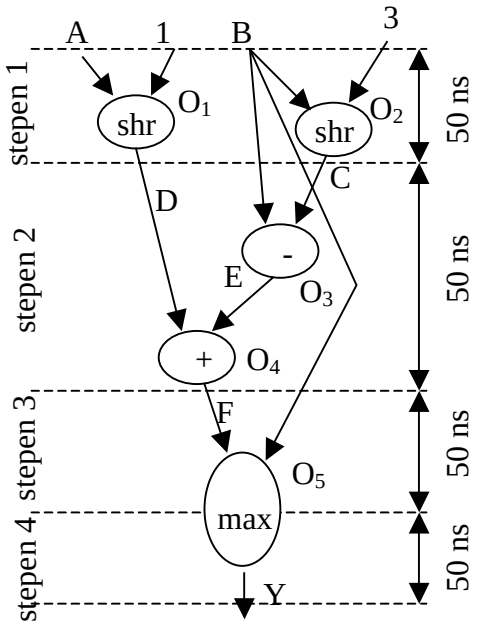
Planiranje izvršenja

Naredni korak u postupku sinteze se odnosi na planiranje-izvršenja operacije (aritmetičkih/logičkih) i memorijskih pristupa (čitanje/upis) u odgovarajućim taktnim intervalima. Postoje dva tipa algoritama za planiranje-izvršenja, prvi imaju za cilj optimizaciju, a drugi polaze od specificiranih ograničenja. Ako je u toku postupka alokacije korisnik u potpunosti specificirao sve dostupne resurse, kao i trajanje taktnih impulsa, tada cilj algoritma za planiranje-izvršenja je da generiše dizajn sa najboljim performansama, ili dizajn koji će datu aplikaciju izvršavati za najkraći vremenski period. Drugim rečima, planiranje-izvršenja mora da maksimizira korišćenje alociranih resursa. Ako pre planiranja-izvršenja lista resursa nije dostupna, ali su specificirane željene performanse, cilj algoritma za planiranje-izvršenja je da

generiše dizajn sa najnižom mogućom cenom, ili sa najmanjim brojem funkcionalnih jedinica. Ovaj pristup se naziva planiranje-izvršenja sa vremenskim ograničavanjem (*time constrained scheduling*)

Kod planiranja-izvršenja koje se bazira na ograničavanje broja resursa (*resource-constrained scheduling*) uobičajena je praksa da se kreira (konstruiše) plan-izvršenja koga karakteriše jedno stanje po taktnom intervalu (u datom trenutku). Planiranje-izvršenja operacija se tako izvodi da se ne premaši ograničenje po resursima kao i naruši princip zavisnosti po podacima. U toku planiranja-izvršenja osigurano je sledeće: Ako je izvršenje operacije *Oi* planirano u toku upravljačkog koraka *Sj*, tada mora biti dostupan resurs koji će izvršiti *Oi* a takodje i svi prethodnici čvora *Oi* moraju biti već isplanirani.

U prethodno opisanom primeru, alokacionim zadatkom E (vidi Sliku 7) odredili smo da su neophodna dva brza ALU-a (ALU-F) i jedna MAX jedinica da bi se ostvarile procenjene performanse od 200ns. Pokušajmo sada da izvršimo planiranje-izvršenje ovog primera koristeći alokaciju E i taktni interval od 50ns. Algoritam može da isplanira-izvršenje u prvom upravljačkom koraku dve *Shift* operacije (O_1 *i* O_2), jer ove operacije ne zavise od prethodnih i zbog toga što su oba ALU-a dostupna. U drugom upravljačkom koraku algoritam za planiranje-izvršenja odredjuje da se obe operacije, sabiranje O_3 i oduzimanje O_4, mogu isplanirati radi izivršenja, jer su oba resursa (dve brze ALU-F jedinice) dostupna. Čvor MAX se ne može planirati-za-izvršenje sve dok prethodna operacija O_4 ne izračuna svoj rezultat, pa se zbog toga ona isplanira radi izvršenja u narednom stanju. Konačni plan izvršenja prikazan je na Slici 8.



Slika 8 Primer planiranja izvršenja bazirano na alokaciji E

Kao što se vidi sa Slike 8 operacije O_3 i O_4 se planiraju radi uizvršenja u istom stanju i pored toga što izmedju ta dva čvora postoji zavisnost. Ovakvo planiranje je izvodljivo samo ako je dostupan dovoljan broj komponenata a propagaciono kašnjenje signala kroz komponente je kraće od periode taktnog signala. U konkretnom slučaju alocirani ALU-ovi imaju kašnjenje od 20ns, dostupne su dve ALU jedinice, pa je moguće ulančati operacije *Add* u *Subtract* u okviru taktnog perioda od 50ns.

Suprotan efekat se javlja kada je propagaciono kašnjenje signala kroz funkcionalnu jedinicu duže od periode taktnog signala. U takvim situacijama planer mora da obezbedi nekoliko stanja kako bi operacija završila. U konretnom slučaju komponenta MAX ima propagaciono kašnjenje od 80ns dok je trajanje taktnog intervala 50ns. To znači da je za izvršenje operacije O_5 potreban period od dva taktna intervala. Planiranje izvršenja operacija čije izvršenje traje više od jedan taktni interval naziva se više-taktno izvršenje (*multicycling*).

Kada kod algoritama za planiranje-izvršenja postoji ograničenja sa aspekta vremena (*time-constrained schedulling*) ide se na soluciju da se fiksira maksimalan broj upravljačkih koraka za date operacije. Na osnovu ovog performansnog ograničenja (vreme izvršenja) kao i ograničenja usled zavisnosti izmedju čvorova (prvo mora da se izvrše prethodnici) moguće je odrediti vremensko najraniji upravljački korak e_i

kao i vremensko najkasniji upravljački korak l_i tako da u okviru tog intervala je moguće planirati izvršenje čvora *Oi*. Koristeći e_i i l_i za sve čvorove moguće je proceniti (odrediti) maksimalan broj funkconalnih jedinica ili cenu dizajna. Algoritmi za planiranje-izvršenja kod kojih postoje ograničenja sa aspekta vremena izvršenja selektuju čvor *Oi* i u svakom upravljačkom koraku izmedju trenutaka e_i i l_i procenjuju njegovu cenu, a zatim selektuju stanje *Si* za koje je cena najniža. Važan cilj je da se u svakom koraku minimizira broj funkcionalnih jedinica.

Povezivanje

Povezivanje (*binding*), kao zadatak, ima za cilj da u okviru svakog taktnog intervala dodeli operacije i memorijske pristupe dostupnim hardverskim jedinicama. Resursi kakvi su funkconalna jedinica, memorijska jedinica ili jedinica za povezivanje (sprežna) mogu biti deljive od strane različitih operacija radi izvršenja, (deljive radi) pristupa podacima, ili prenosa podataka za slučaj da su sve ove aktivnosti uzajamno isključive. Na primer, dve operacije dodeljene dvaju različitim upravljačkim koracima su uzajamno isključive, jer se one nikad ne izvršavaju simultano, pa se zbog toga mogu povezati na istu hardversku jedinicu.

U zavisnosti od tipa jedinice proces povezivanja čine sledeća tri tipa podzadataka:

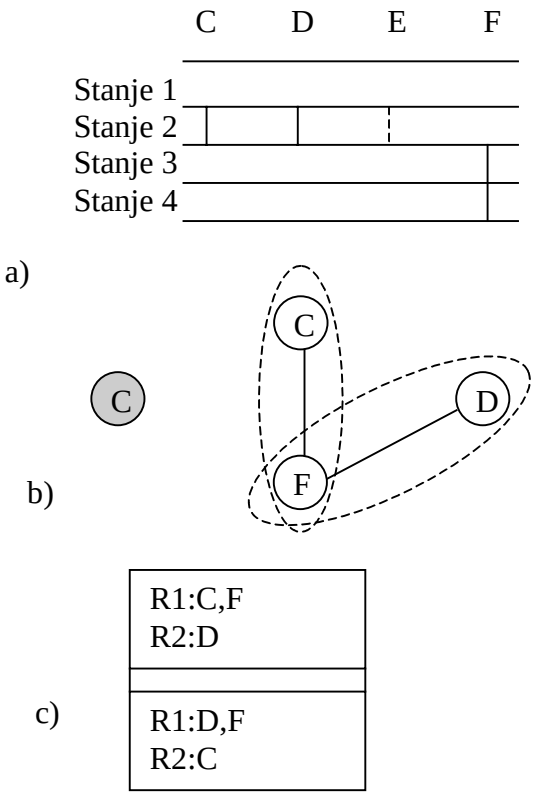
- **povezivanje kod memorisanja** – ima zadatak da varijablama dodeli memorijski prostor. Memorijski prostor može biti različitog tipa, a tipično su to registar, registarska polja ili memorijske lokacije. Dve promenljive kojima se u toku datog stanja ne pristupa simultano (istovremeno) mogu biti dodeljene istom portu registarskog polja ili memorije.
- **povezivanje funkconalnih jedinica** – u okviru upravljačkog koraka svaka operacija se dodeljuje odgovarajućoj funkcionalnoj jedinici. Funkcionalna jedinica ili protočni stepen mogu da izvršavaju samo po jednu operaciju po taktnom intervalu.
- **povezivanje bloka za sprezanje** – dodeljuje jedinicu za spregu, kakav je multiplekser ili magistrala, u toku svakog prenosa podataka izmedju portova, funkcionalnih jedinica, ili memorijskih jedinica.

I pored toga što su navedena izdvojeno, sva tri podzadatka su uzajamno isprepletena (zavisna) tako da ako želimo da dobijemo optimalne rezultate, podzadaci mora da se izvršavaju konkurentno.

Za primer planiranja-izvršenja sa Slilke 8 ilustriraćemo proces povezivanja. Da bi (po)vezali promenljive za registre moramo sve promenljive iz opisa da podelimo na kompatibilne skupove. Skup promenljivih je kompatibilan ako sve promenljive skupa nisu istovremeno u životu. Da bi odredili kompatibilne skupove, moramo odrediti vremena života promenljivima, kako je to prikazano na Slici 9a). Na osnovu vremena života kreira se kompatibilni graf kod koga svaki čvor predstavlja promenljivu, a svaki poteg povezuje dve promenljive koje imaju uzajamno isključiva vremena života. U konretnom primeru, promenljive *C* i *D* se ažuriraju (vrši se upis) u stanju 1 a čitaju u stanju 2, dok se promenljive *E* ažurira i čita u stanju 2, a promenljiva *F* ažurira u stanju 2 a čita u stanju 3. Na slici 9b) prikazan je graf kompatibilnosti za ove promenljive.

Nakon ovoga graf kompatibilnosti mora da se podeli na klike (*cliques*). Klika je potpuno povezani podgraf, drugim rečima, podgraf koga čini nekoliko čvorova, pri čemu je svaki čvor povezan sa svim svojim susednim čvorovima. Klika ukazuje na skup uzajamno isključivih čvorova koji se uzajamno mogu povezati preko istog resursa.

Kod analiziranog primera, particija grafa kompatibilnosti na klike rezultira tome da postoje dva moguća rešenja (vidi Sliku 9c)). Kod oba rešenja za čuvanje promenljivih potrebna su po dva registra. Prvo rešenje koristi registar R1 za čuvanje promenljivih *C* i *F* a registar R2 za čuvanje promenljive *D*. Promenljivu *E* ne treba memorisati jer njen život ne prelazi granice izmedju stanja i može se implementirati pomoću linija (žica).



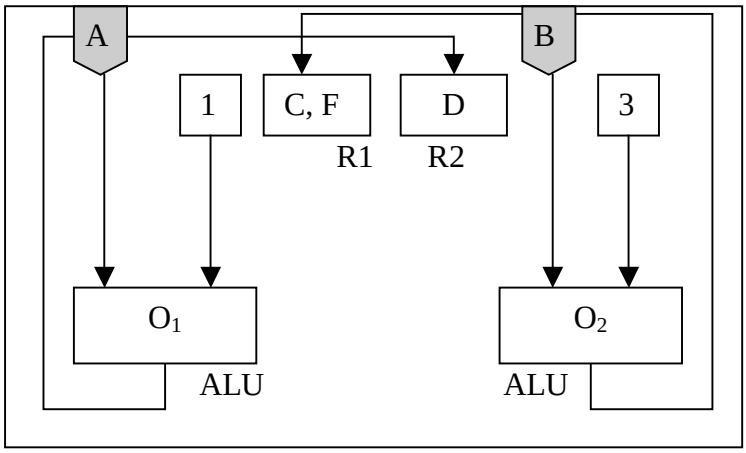
Slika 9. Povezivanje kod memorisanja; (a) vreme života promenljivih; (b) graf kompatibilnosti; (c) moguća rešenja

Naredni zadatak kod procesa povezivanja se odnosi na povezivanje funkcionalnih jedinica. Ovom aktivnošću u toku svakog taktnog intervala vrši se dodela operacija alociranim funkcionalnim jedinicama. Moguće je naći rešenje za problem povezivanja funkcionalnih jedinica ako se u datom trenutku izvršava po jedno stanje. S obzirom da se obe operacije O_1 i O_2 mogu povezati na bilo koji od dostupnih ALU-ova prvo stanje nudi dva izbora za povezivanje. Usvojićemo da se O_1 povezuje sa prvim ALU-om a O_2 sa drugim. Na Slici 10a) prikazan je parcijalni dizajn nakon povezivanja operacija u prvom stanju.

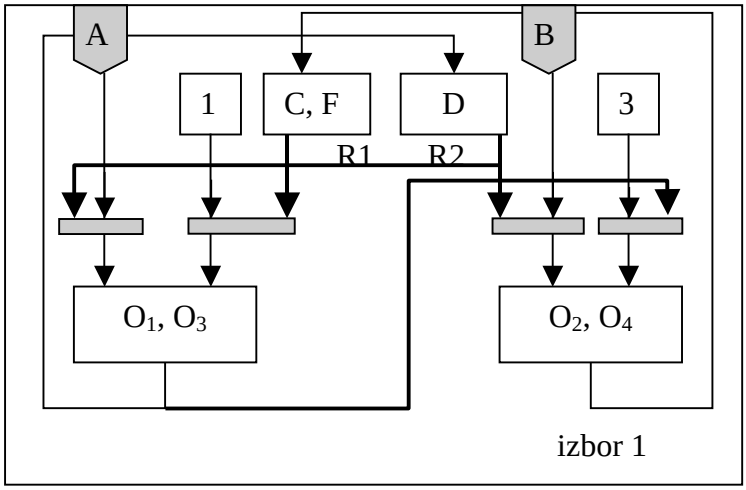
Proces povezivanja produžava za operacije u drugom stanju. S obzirom da stanje 2 ima dve operacije O_3 i O_4, a oba ALU-a su sposobna da izvrše obe operacije, imamo na raspolaganju dva različita izbora: Povezivanje O_3 na ALU1 i O_4 na ALU2, i obrnuto. Na Slici 10b) prikazan je rezultujući parcijalni dizajn za prvi izbor. Nove veze su prikazane udebljanim (*bold*) linijama. Da bi se kompletiralo parcijalno povezivanje stanja 1 i 2 dizajn zahteva četiri dodatna dvo-ulazna multipleksera ili osam novih tro-statičkih drajvera. Na Slici 10c) prikazan je rezultat drugog izbora: Tri dodatna multipleksera ili šest tro-statičkih drajvera je dovoljno pošto je veza od *B* ka drugom ALU-u već prethodno ostvarena. To znači da drugi izbor rezultira optimalnijem dizajnu.

Ipak, opisani algoritam povezivanja koristi »*greedy*» pristup i može voditi ka suboptimalnim rešenjima.

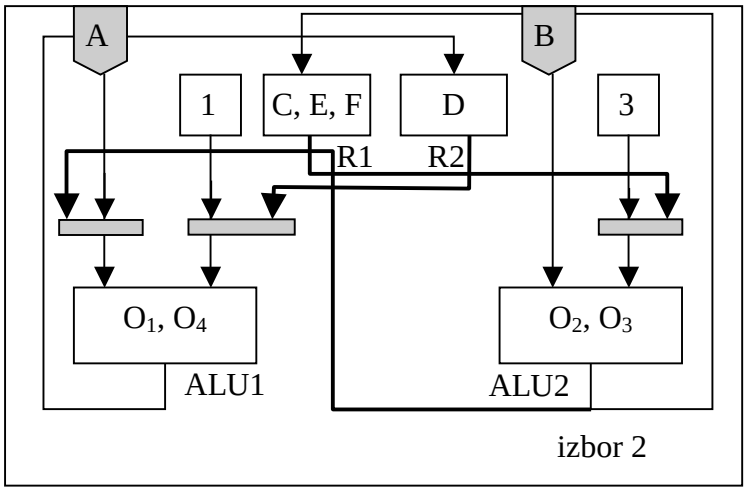
a)



b)



c)



Slika 10 Povezivanje funkcionalne jedinice: (a) parcijalni dizajn nakon povezivanja operacija u stanju 1; (b) stanje 2, izbor 1; i (c) stanje 2, izbor 2