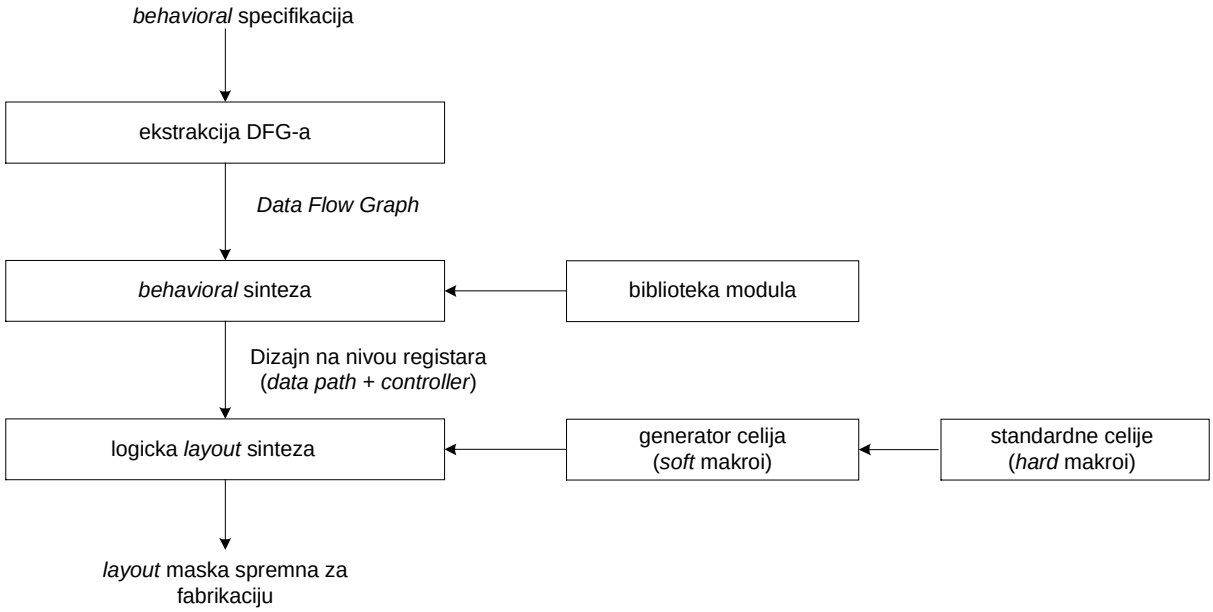
PLANIRANJE

1. 1. Behavioral sinteza

Behavioral sinteza predstavlja proces generisanja dizajna na registarskom nivou polazeći od *behavioral* specifikacije na algoritamskom nivou. Dizajn koji je izlaz (rezultat) *behavioral* sinteze čine dve interaktivne komponente: staza podataka (*data path*) i kontroler.
Staza podataka predstavlja kompozicija modula koji su selektovani iz modul-biblioteke. Biblioteka sadrži sabirače, multipleksere, registre, i druge module.
Kontroler se generiše kao FSM opis koji prihvata markere statusa (uslova) iz staze podataka i generiše upravljačke signale koji upravljaju radom različitih resursa staze podataka. Prelaz iz jednog stanja u drugo kod samog kontrolera dovodi do sekvenciranja registarskog prenosa u stazi podataka. Nakon *behavioral* sinteza standardno slede koraci logičke i *layout* sinteze radi generisanja fabričke maske *layout*-a (vidi sliku 1). U toku procesa logičke sinteze svaki modul staze podataka se preslikava u modul tipa složeno-logičko-kolo koje se najčešće realizuje pomoću stanardnih ćelija.
Sredstva projektovanja za logičku i *layout* sintezu procesiraju kontroler kako bi generisali implementaciju FSM-a. Pri ovome se pretpostavlja da ova implementacija uzima u obzir ožičenu formu PLA strukture i da je implementacija u potpunosti u CMOS tehnologiji.
Ograničenja tipa površina i trajanje taktne periode uobičajeno se uzimaju u obzir prilikom *behavioral* procesa sinteze.



Slika 1 Vertikalno integrisano okruženje za sintezu

1.2. Identifikacija osnovnih zadataka kod postupka sinteze na visokom nivou

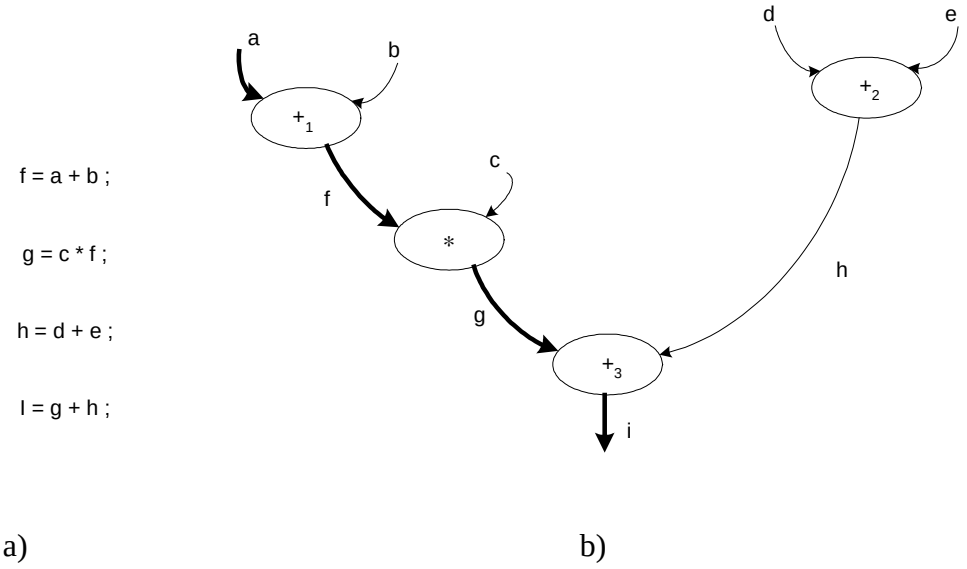
Kod sinteze na visokom-nivou dva osnovna zadatka koja treba obaviti su:

1. **planiranje izvršenja** (*scheduling*)
2. **dodela** (*allocation*)

Planiranjem se odredjuje redosled izvršenja zadataka, dok se dodelom vrši pridruživanje hardvera koji će obavljati te operacije. U suštini ova dva zadatka su veoma složena i pripadaju klasi NP-kompleta. Ovo je glavni razlog zbog čega se za rešavanje ovih problema koriste heuristički pristupi.

1.3. Modeliranje ponašanja

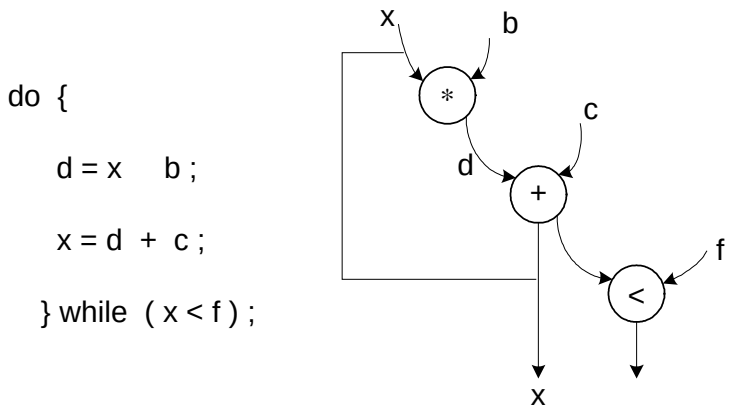
Ponašanje (*behevior*) kola na visokom-nivou se može specificirati nekim od HDL-ova. Nakon toga, na osnovu definisane specifikacije, ponašanje se može predstaviti u formi grafa na osnovu toka podataka (*data flows*) i toka-upravljanja (*control flows*)
Behavior spcifikacije, *osnovni-blok* se definiše kao neprekidna-sekvenca iskaza koja sadrži iskaz grananja samo na svom kraju. Shodno tome, graf toka podataka (*data flow graph* – DFG) je moguće izvesti iz osnovnog-bloka pri čemu se svakom čvoru pridružuje operacija, a svakom potegu se dodeljuje promenljiva. Zavisnost po podacima je odredjena smerom potega. Na slici 2a) prikazan je jedan primer osnovnog-bloka koga čine četiri iskaza, a na slici 2b) odgovarajući DFG.



a) b)

Slika 2 a) Behavior specifikacija; b) odgovarajući DFG

Promenljive kod DFG-a mogu pripadati nekom od sledeća tri skupa: primarni ulazi (V_I), primarni izlazi (V_O), medju-promenljive (V_M). Tok-podataka kod DFG-a se označava kao $v_1 \xrightarrow{O_1} v_2 \xrightarrow{O_2} ..... \xrightarrow{O_{n=1}} v_n$, gde v_i, $1 \le i \le n$ je promenljiva u $V_I \cup V_O \cup V_M$, dok O_j koji se pridružuje svakom "$\rightarrow$", $1 \le j \le n$ predstavlja operaciju (kakve su sabiranje, množenje, i dr), sa v_j kao ulaznim operandom, a v_{j+1} kao rezultatom. Kada su v_1 i v_n iste, tok-podataka se naziva ciklični-tok-podataka, a v_1 i v_n nazivamo granične promenljive. Skup svih graničnih promenljivih kod DFG-a se označava sa V_B. Za DFG kažemo da je aciklični kada ne poseduje ciklični-tok-podataka, inače je DFG ciklični. Na primer, deblje izvučeni potezi na slici 2 b) odnose na aciklični tok podataka $a \xrightarrow{+_1} f \xrightarrow{*} g \xrightarrow{+_3} i$. Ciklični graf toka-podataka obično odgovara konstrukcijama tipa petlja (*loop*), kakvi su u *behavior* specifikaciji, iskazi tipa *while* ili *for*. Na slici 2a) prkazan je osnovni-blok koji na svom kraju ima *while* iskaz. Odgovarjući DFG je dat na slici 3b), koji ima graničnu promenljivu *x* pridruženu cikličnom grafu toka-podataka odredjenu od strane iskaza *while.* Naime, tekuća vrednost promenljive u cikličnom grafu toka-podataka zavisna je od prethodne vrednosti u zadnjoj iteraciji petlje. Uočimo da graf toka-podataka sa slike 3b) ne opisuje tok-upravljanja (*control flow*), nego samo tok-podataka u okviru iskaza *while*.

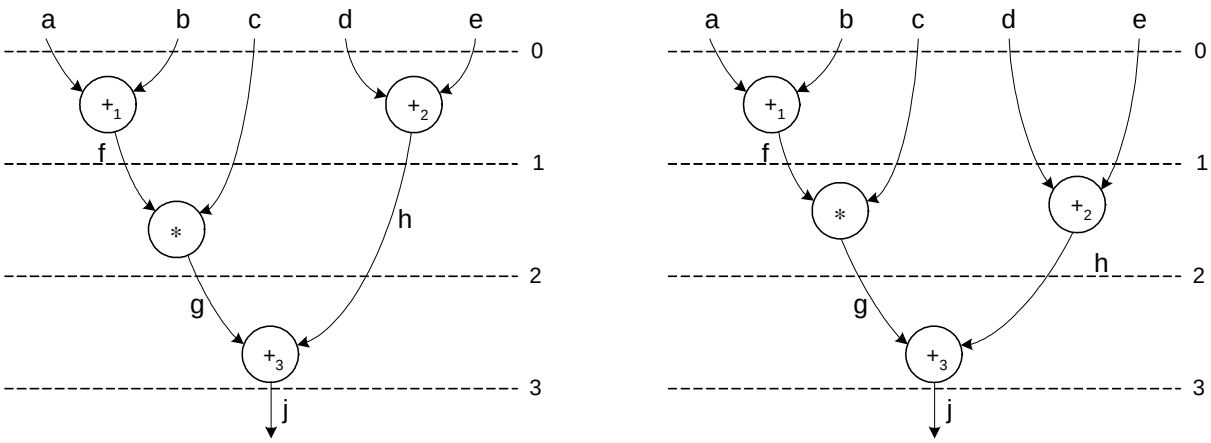


Slika 3 a) osnovni-blok koji je tipa petlja; b) odgovarajući DFG sa graničnom promenljivom *x*

Osnovni koncept planiranja izvršenja

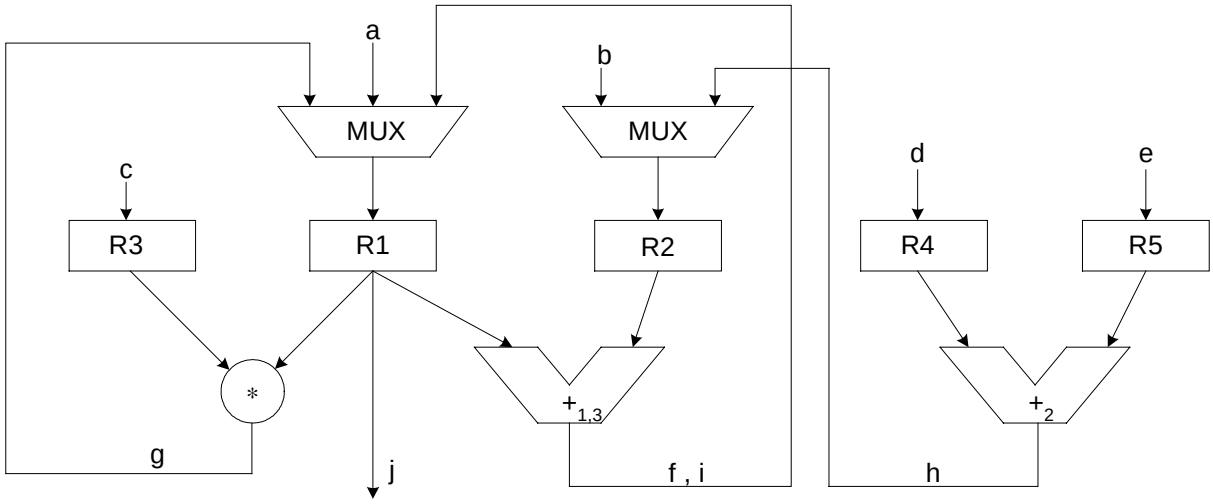
Za svaku operaciju o koja je pridružena čvoru DFG-a, $o_0\ ran$ označava najraniji vremenski ciklus (period) za koji se o može izvršiti, a $o_0\ kas$ označava najkasniji mogući vremenski ciklus za koji se o može izvršiti, a da se pri tome ne naruše zavisnosti-po-podaciima definisani od strane DFG-a. Pri ovome, labavost operacije o se definiše kao interval $[o_0\ ran - o_0\ kas]$, a mobilnost operacija o se definiše kao $o_0\ ran - o_0\ kas$. Sekvenca operacija čija je mobilnost 0 naziva se kritični-put (*critical path*). Tako na primer, operaciju $+_2$ na slici 2b) karakteriše labavost $[1, 2]$, a njena mobilnost iznosi 1. Operacije $+_1, *, i +_3$ imaju mobilnost 0 pa su zbog toga na kritičnom putu. Labavost i mobilnost se mogu koristiti da ukažu na slobodu planiranja izvršenja operacija.

Svakoj operaciji *behavior* specifikacije, u toku koraka upravljanja, planiranjem-izvršenja se dodeljuje vreme izvršenja. U daljem tekstu razmatraćemo jednostavno planiranje-izvršenja, a to je slučaj kada ne postoji operacija koja se izvršava za veći broj ciklusa. Za DFG sa slilke 2b), na slici 4 prikazana su dva različita plana izvršenja *Sa* i *Sb*, pri čemu isprekidana linija ukazuje na upravljački korak. Nakon odredjivanja plana izvršenja DFG graf nazivamo SDFG (*scheduled DFG*). U konkretnom slučaju operacija $+_2$ se može planirati radi izvršenja u ciklusu 1 (kod plana *Sa*) ili u ciklusu 2 (kod plana *Sb*), a da se pri tome ne produži ukupno vreme izvršenja sekvence. Ipak plan-izvršenja *Sa* zahteva najmanje dva sabirača jer se obe operacije $+_1$ i $+_2$ izvršavaju u istom ciklusu 1 tako da ne mogu deliti isti sabirač. Sa druge strane za plan-izvršenja *Sb* potreban je samo jedan sabirač jer se po jedna operacija sabiranja izvršava u svakom ciklusu. Na slici 5 prikazan je krajnji efekati mplementacije.

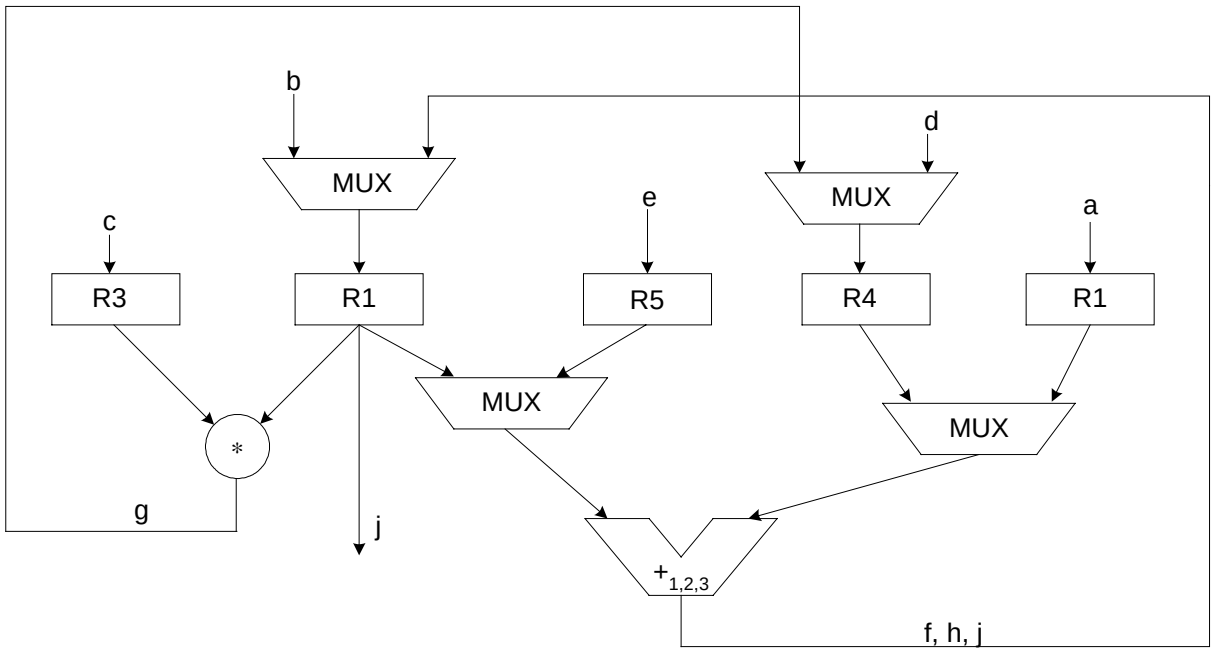


a) b)

Slika 4 Dva plana izvršenja istog DFG-a; a) *Sa*, i b) *Sb*



a)



b)

Slika 5 Dve arhitekture izvredene na osnovu dva plana izvršenja a) za plan *Sa*; i b) za plan *Sb*

Arhitektura sa slike 5a) je izvedena na osnovu plana *Sa*, a čine je pet registara, dva sabirača, jedan množač, i dva multipleksera. Arhitektura sa slike 5b) izvedena je na osnovu plana *Sb*, a čine je pet registara, jedan sabirač, jedan množač, i četiri multipleksera. To su dve različite arhitekture koje karakteriše isto ponašanje (*behavior*).

Algoritmi o planiranju izvršenja

Algoritmi za planiranje izvršenja se mogu podeliti na sledeće dve grupe:
a) transformacioni i iterativno/konstruktivni,

b) InLP (*integer linear programming*) planiranje izivršenja koje se zasnivaju na matematičkoj formulaciji

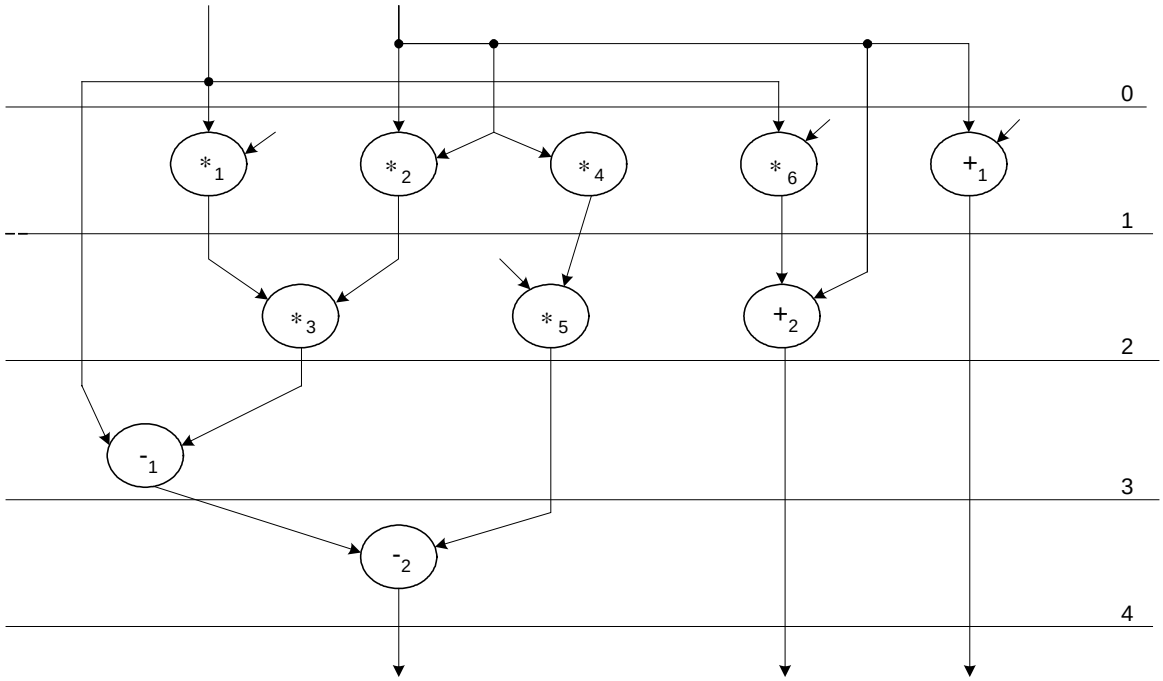
Kod transformacionih algoritama za planiranje izvršenja startuje se od inicijalnog plana-izvršenja i pokušava se da se obave transformacije koje zadržavaju *behavior*, ali teže da poboljšaju plan izvršenja. Tako na primer, polazeći od osnovnog plana-izvršenja teži se da se maksimalno paralelizuje sekvencijalni plan, a da se pri tome zadovolje kriterijumi koji se odnose na ograničenje resursa. Kod iterativno/konstruktivnog pristupa obavlja se planiranje jedne operacije po ciklusu sve dok se ne isplaniraju sve operacije. Važni algoritmi iz ove klase su:

- što-je-moguće-pre (*as-soon-as-possible: ASAP schedlling*)
- što-je-moguće-kasnije (*as-late-as-possible: ALAP scheduling*)
- *list scheduling* (LS)
- *critical path scheduling* (CPS)
- *list scheduling* (LS)
- *force directed schedlling* (FDS)
- *path based scheduling* (PBS)

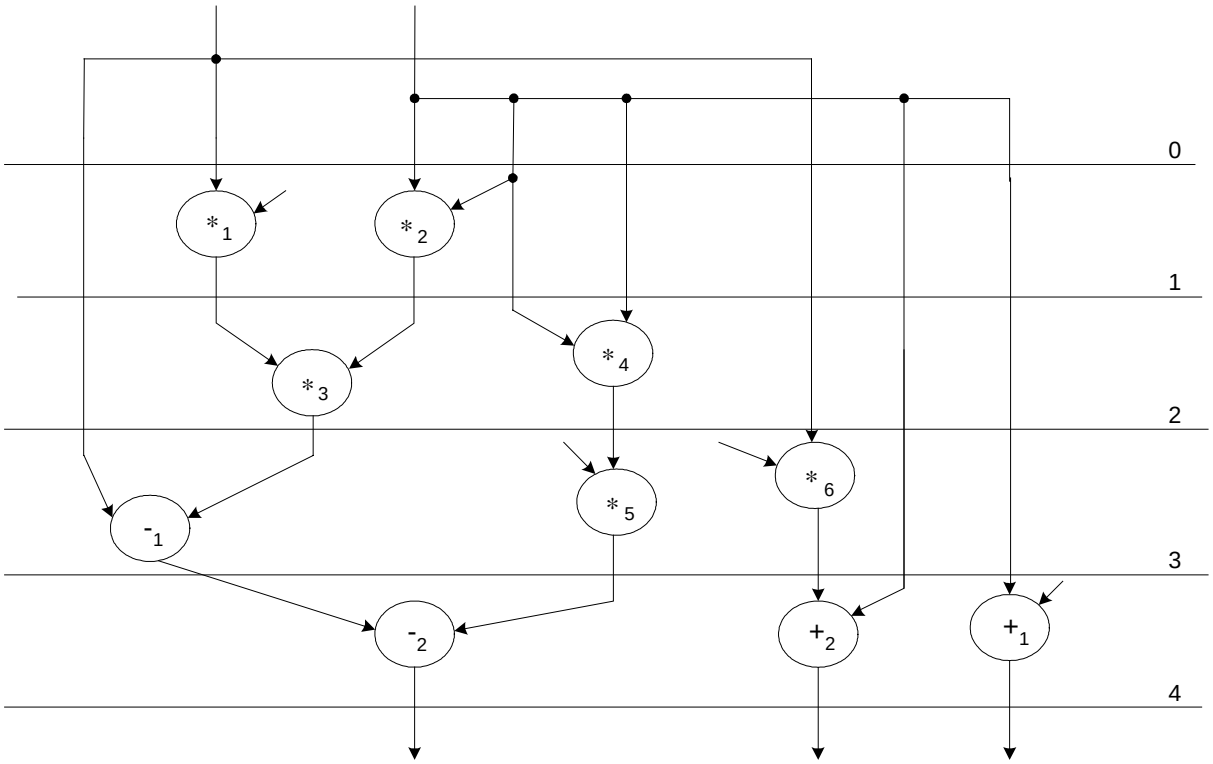
ASAP i ALAP planiranje izvršenja

ASAP i ALAP su najjednostavniji konstruktivni pristupi za planiranje izvršenja.

Svaku operaciju o iz DFG-a ASAP algoritam planira radi izvršenja u ciklusu $o.ran$, dok je ALAP algoritam planira radi izvešenja u ciklusu $o.kas$. Na slici 6 prikazana su dva DFG-a planirana za izvršenje od strane ASAP i ALAP algoritama, respektivno.



a) ASAP



b) ALAP

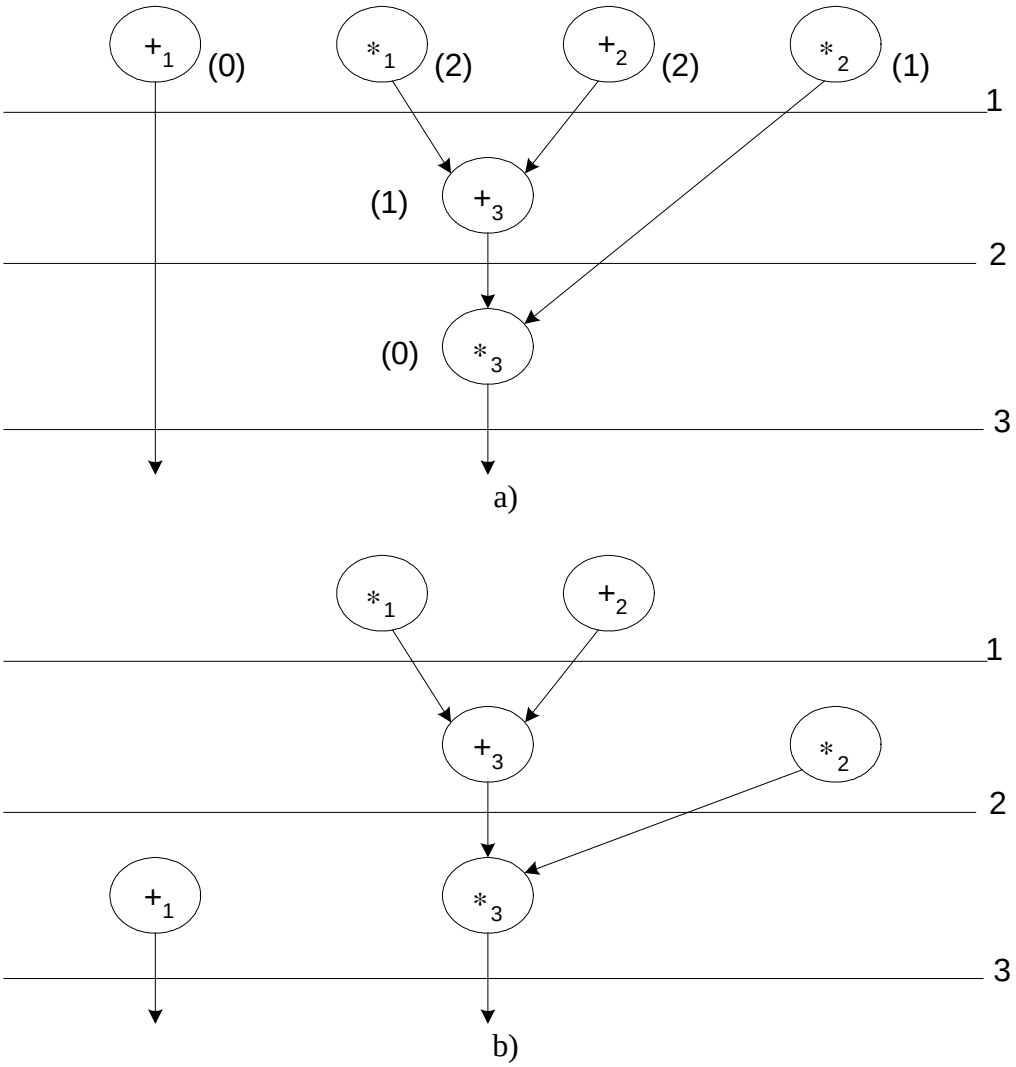
Slika 6 Planiranje-izvršenje DFG-ova od strane ASAP i ALAP algoritama, respektivno

I pored toga što ASAP i ALAP algoritmi mogu uvek da dodele najbrži algoritam planiranje-izvršenja, često je za njihovu implementaciju potreban nešto veći iznos hardvera od minimalnog. Razlog ovome je što ovi algoritmi ne vode računa o kritičnom putu i ne mogu da odlože izvršenje operacije na nekritičnom putu kako bi se uštedilo na resursu. Na primer, za planiranje-izvršenja ASAP algoritma sa slike 6 potrebna su četiri množača (množenje se planira za izvršenje u ciklusu 1). Ipak, ako se $*_4$ i $*_6$ koje se nalaze na nekritičnim putevima planiraju radi izvršenja u ciklusu 2, dok $*_5$ i $+_2$ se planiraju radi izvršenja u ciklusu 3, tada su potrebna samo dva množača, a da se pri tome ne poveća ukupan broj ciklusa izvršenja.

List schedulign **(LS)**

LS algoritam premošćava problem ASAP algoritma na taj način što kreira listu operacija uredjenih po nekoj funkciji prioriteta. Umesto da bira operacije za izvršenje što je moguće ranije, LS algoritam planira izvršenje operacija bazirano na prioritetu. Na osnovu ovog kriterijuma vreme izvršenja operacije sa najnižim prioritetom se može odložiti ako postoje ograničena sa aspekta korišćenja resursa. Funkcije prioriteta mogu biti mobilnost, urgentnost (minimalan broj ciklusa od te operacije do izlaza), i druge.

Na slici 7a) prikazan je DFG kod koga je u zagradama naznačen dodeljeni prioritet svakoj operaciji. U konkretnom slučaju prioritet operacije o se definiše kao broj operacija izmedju o i kraja bloka u kome je locirana o. Usvaja se da su ograničenja sa aspekta resursa jedan sabirač i jedan množač.



Slika 7 (a) DFG primer je svakoj operaciji kao prioritetna funkcija pridružen broj operacija od teoperacije pa do kraja bloka (broj u zagradi): (b) list *scheduling* rezultat

U tabeli 1 za svaki ciklus prikazana je lista spremnih resursa, lista prioriteta, i operacije koje se planiraju za izvršenje. Lista spremnosti se generiše i ažurira od strane LS algoritma, a čine je operacije čiji su neposredni prethodnici planirani radi izvršenja. Spremne operacije su kandidati da budu planirane za izvršenje u tekućem ciklusu samo kada su zadovoljena organičenja koja se odnose na broj resursa i prioritet. Tako na primer, u prvom ciklusu u Tabeli 1, operacije $+_1, *_1, +_2$ *i* $*_2$ su spremne za planiranje. Ali zbog ograničenja broja resursa, selektuju se samo $*_1$ *i* $+_2$ čiji je prioritet najviši i planira, u prvom ciklusu, njihovo izvršenje. Na sličan način, u drugom ciklusu, listu spremnosti čine $+_1$, $*_2$ *i* $+_3$, pri čemu se planiraju za izvršenje operacije sa najvišim prioritetom $*_2$ *i* $+_3$. Konačno, $*_3$ *i* $+_1$ se planiraju za izvršenje u trećem ciklusu. Na slici 7b) prikazan je plan-izvršenja DFG-a.

Tabela 1: LS rezultat u svakom ciklusu za DFG sa slike 7a, pod uslovom da postoje ograničenja po resursima od jedan sabirač i jedan množač

ciklus	lista spremnosti	lista prioriteta	lista plana-izvršenja
1	$(+_1, *_1, +_2, *_2)$	$(*_1, +_2, *_2, +_1)$	$*_1, +_2$
2	$(+_1, *_2, +_3)$	$(+_1, +_3, *_2)$	$*_2, +_3$
3	$(+_1, *_3)$	$(+_1, *_3)$	$*_3, +_1$

Planiranje-izvršenja po kritičnom putu (CPS)

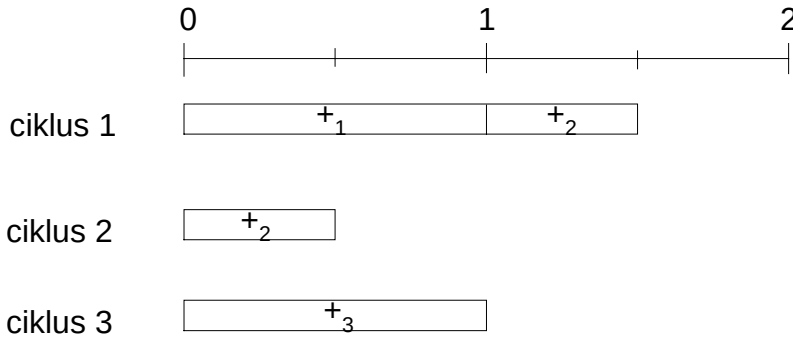
Algoritam CPS planira prvo da izvrši operacije koje se nalaze na kritičnom putu. Za ostale operacije, dopušta se sloboda, slično konceptu kako se odredjuje mobilnost. Sloboda operacije se odredjuje kao razlika izmedju vremena kada su ulazne vrednosti potrebne toj operaciji i vremena kada je rezultat te operacije potreban, umanjeno za propagaciono kašnjenje koje unosi hardver za potrebe procesiranja. Operacija čiji je iznos slobode najmanji selekture se kao prva od planera radi izvršenja.
Jedan tipičan CPS algoritam čine sledeći koraci:

- korak-1: za zadati DFG, ocenjuju se vremenska ograničenja i svakoj operaciji se dodeljuje vrednost kašnjenja
- korak-2: odredjuje se kritični put kao i broj taktnih ciklusa potreban za izvršenje uzimajući u obzir vremenska ogbraničenja. Za slučaj da se vremenska ograničenja ne mogu zadovoljiti vrati se na korak-1 i postavi nova ograničenja
- korak-3: operacijama na kritičnom putu dodeli hardverske module
- korak-4: ako ne postoji više operacija koje treba planirati radi izvršenja, završiti algoritam
- korak-5: izračunati/ažurirati slobode za sve operacije koje nisu na kritičnom putu
- korak 6: planirati-izvršenje operacije sa najmanjom slobodom
    - ako je moguće, modul koji je spreman za dodelu je deljiv od strane ove operacije. Preći na korak-4.
    - Inače, dodaj drugi hardverski modul i preći na korak-4 ili dodaj još jedan ciklus i preći na korak-2.

Planiranje tipa usmeravanje-na-silu

Planiranje tipa usmeravanje-na-silu (*force directed scheduling*) je globalni algoritam balansiranog opterećenja koji pokušava da slučajno distribuira operacije u toku ciklusa, tako da resursi budu jako iskorišćeni. Kod ove šeme za svaku operaciju se izvodi distribucioni graf (*distribution graph*) koji ukazuje na konkurentnost operacija. Zatim se izračunava "sila" (*force*), mera koja pokazuje na distribuiranost konkurentnosti, za svaku od operacija pa se ta "sila" koristi za dodeljivanje operacije onom ciklusu kod koga je iskorišćenost resursa mala.
Distribucioni graf se koristi za izračunavanje "sila" operacija istog tipa koje mogu deliti (koristiti) funkcionalni modul. Distribucioni graf za svaki ciklus prikazuje sa kolikim brojem istih operacija je taj ciklus opterećen, pri čemu za sve operacije postoji jednaka verovatnoća da budu planirane za izvršenje u toku tog mrtvog-perioda. Ako je mrtvi-period operacije *k* ciklusa, tada se u svakom ciklusu mrtvog-perioda distribucionog grafa toj operaciji dodaje *1/k*. Na slici 8 prikazan je distgribucioni graf operacije sabiranja za DFG sa slike 2b). Usvojićemo da vremenska ograničenja iznose 3 ciklusa. S obzirom da se operacije $+_1$ i $+_3$ moraju da isplaniraju u ciklusima 1 i 3, respektivno, tada se 1 dodeljuje ciklusima 1 i

3 distribucionog grafa, respektivno. Za operaciju $+_2$, s obzirom da se može planirati bilo za ciklus 1 ili 2, tada se svakom ciklusu dodaje po $1/2$.



Slika 8. Graf distribucije za operacije sabiranja kod DFG-a sa slike 2

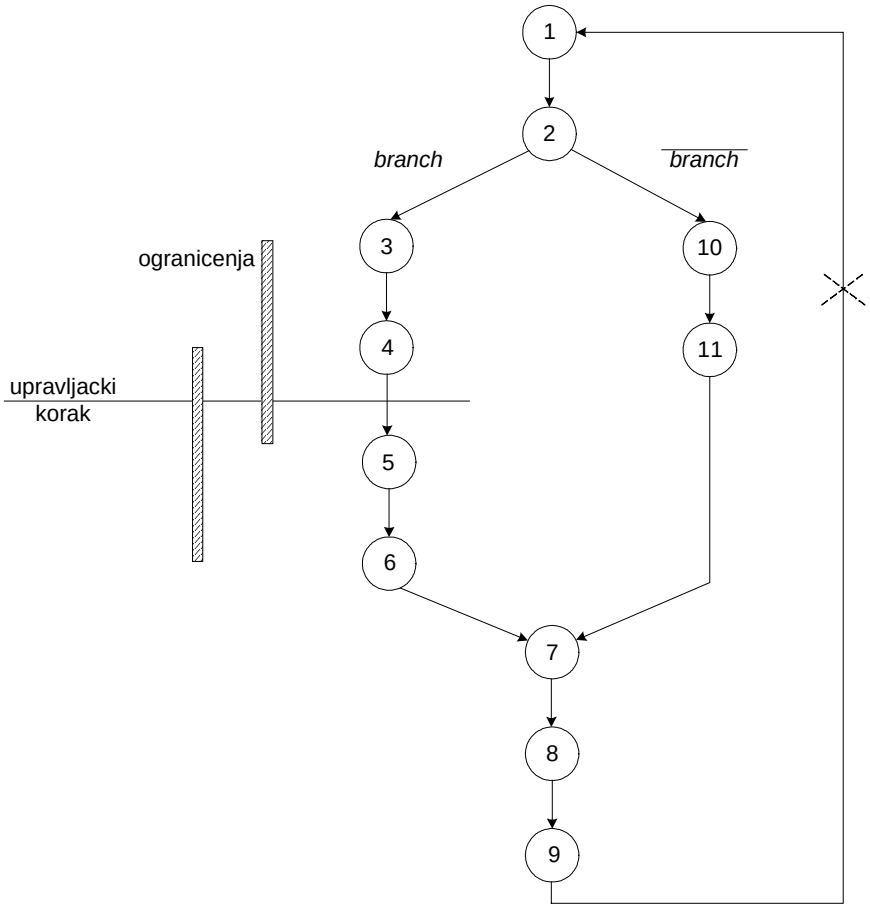
Kod zadatog grafa distribucije, za svako moguće planiranje izvršenja operacije *o* u ciklusu *i*, "sila" se izračunava na sledeći način

$$F_i = \sum_{j \in (O_0 \text{ najranije}\,,\, O_0 \text{ najkasnije})} DG_j \times \Delta(j\,,i)$$

gde je:
j opseg *o*-ih mrtvih perioda, DG_j vrednost grafa distribucije u ciklusu *j*, a $\Delta(j\,,i)$ je promena u vrednosti grafa distribucije ako je planiranje *o*-a fiksirano za ciklus *i*. Na primer, za sliku 8, uvedena "sila" koja se odnosi na dodelu operacije $+_2$ ciklusu 1 iznosi $F_1 = 1{,}5 * 0{,}5 + 0{,}5 * (-0{,}5) = 0{,}5$. Prvi član se odnosi na promenu koja se unosi kod sabiranja u ciklusu 1, imajući u vidu da se njena verovatnoća povećava sa 0,5 na 1, dok se drugi član odnosi na odstranjivanje operacije $+_2$ u ciklusu 2, što dovodi do smanjenje njene verovatnoće sa 0,5 na 0. Pozitivna sila F1 ukazuje da operaciju ne treba planirati radi izvršenja u ciklusu 1, jer ciklus 1 je jako opterećen sa operacijom sabiranja. Na sličan način se izračunava "sila" za planiranje $+_2$ u ciklusu 2 kao $F_2 = 1{,}5 * (-0{,}5) + 0{,}5 * 0{,}5 = -0{,}5$. Nakon što su odredjene sve "sila" , bira se plan sa najmanjom "sila" , u konkretnom slučaju je to F2. Zbog toga se operacija planira za izvršenje u ciklusu 2, a distribucioni graf se ažurira kako bi predstavljao odraz ove odluke.

Planiranje zasnovano na izboru puta

Planiranje zasnovano na izboru puta (*path based scheduling*), putem korišćenja uslovnih grananja za sve puteve izvršenja u opisu dizajna, minimizira broj taktnih ciklusa. Umesto da koristi DFG, da bi odredio plan izvršenja, planiranje bazirano na izboru puta koristi graf toka upravljanja (*control flow graph*-CFG) koji se dobija od opisa dizajna. Kod DFG-a, čvorovi predstavljaju operacije koje treba isplanirati, dok potezi ukazuju na relacije prethodjenja izmedju čvorova. To znači poteg od čvora *i* na *j* ukazuje da se operacija *j* može izvršiti tek nakon što se prethodno izvrši operacija *i*. Ako neki čvor ima više od jednog naslednika za takav čvor kažemo da predstavlja uslovno-grananje (*conditional branch*). Na slici 9 dat je primer DFG-a kod koga čvor 2 predstavlja uslovno grananje sa signalom *branch* kao predikat (uslov) koji se testira. Čvorovi 3, 4, 5 i 6 se izvršavaju ako je uslov *True*, inače, kada je uslov *False*, izvršavaju se čvorovi 10 i 11. Povratni poteg od čvora 9 ka čvoru 1, koji ukazuje na iterativno izvršenje, se raskida kako bi se CFG, za potrebe planiranja zasnovano na izboru puta, učinio acikličnim. Uočimo da aktuelne operacije koje su predstavljene od strane čvorova nisu od interesa za sprovedeno objašnjenje, one eksplicitno i nisu prikazane na slici 9.



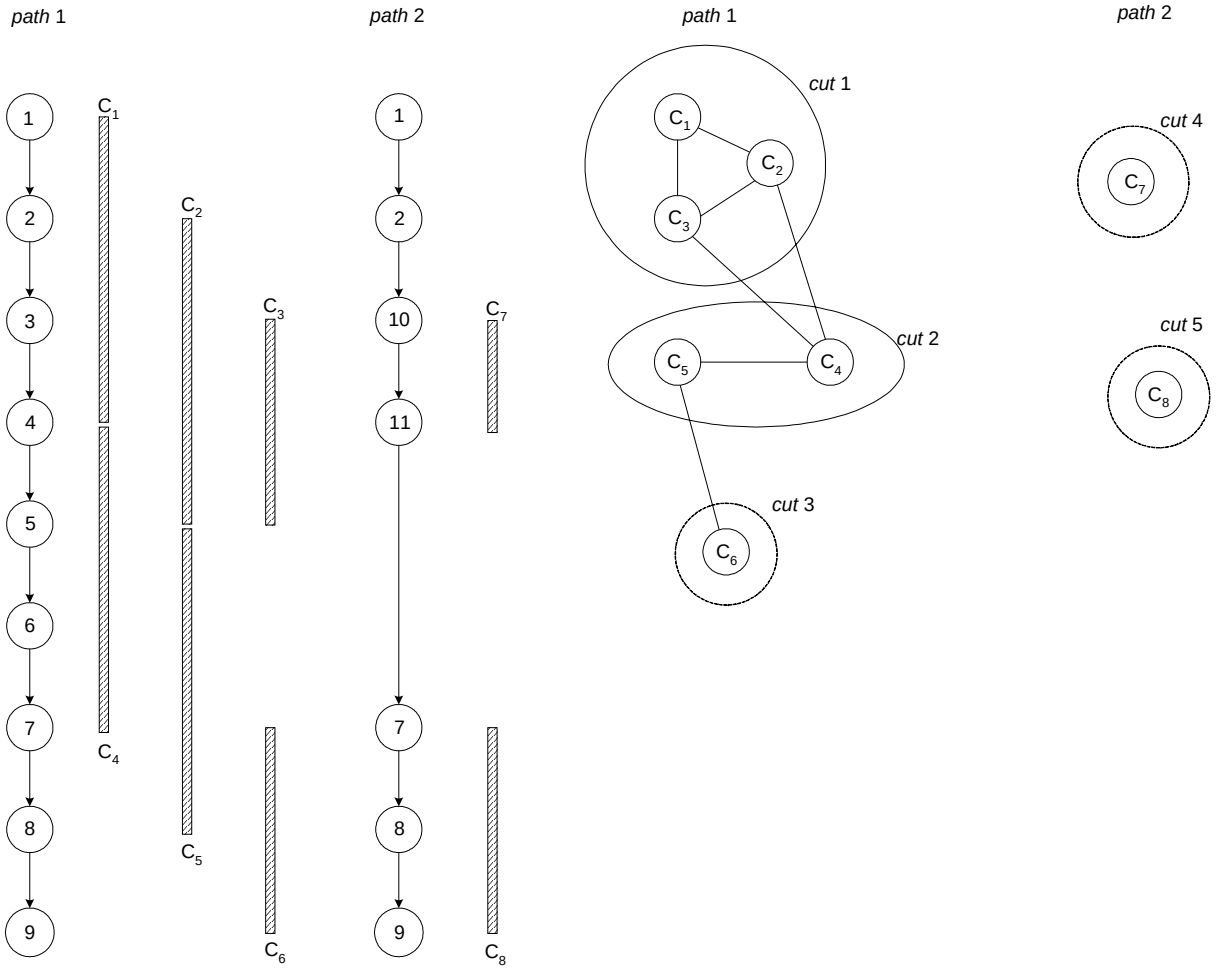
Slika 9 Primer CFG-a kod koga su ograničenja predstavljena intervalima. (Uočimo da je povratni poteg prekinut kako bi se za potrebe algoritma planiranja zasnovana na izboru puta CFG učinio acikličnim).

Ograničenja se odredjuju izmedju ona dva čvora koji mora da se isplaniraju za izvršenje u dva različita ciklusa. Tako na primer, kod CFG-a sa slike 9, ako obe operacije čvorova 3 i 5 dodeljuju vrednost istom signalu, tada se one ne mogu isplanirati za izvršenje istovremeno. Ovo ograničenje je predstavljeno intervalom od čvora 3 do čvora 5, i označeno je prugom na slici 9. I ostala ograničenja se mogu takodje izvesti. Tako na primer U/I port signal ne može istovremeno da obavlja operacije *Read* i *Write*, a takodje i operacije koje koriste isti modul ne mogu se obavljati u istom ciklusu. Kada dva ograničenja imaju intervale koji se preklapaju, oba ograničenja moraju biti zadovoljena na taj način što se u preklapajućem intervalu uvodi upravljački korak. Tako na primer, kod slike 9, upravljački korak se uvodi izmedju čvorova 4 i 5, koji zadovoljava oba ograničenja naznačena od strane i jednog i drugog intervala.

Za efikasan plan izvršenja veoma je važno minimizirati broj taktnih ciklusa, a pri tome zadovoljiti ograničenja. Algoritam koji je zasnovan na izboru puta prvo planira sve puteve izvršenja nezavisno koristeći "*clique partitioning*" tehniku kako bi se odredio minimalan broj ciklusa za svaki put. (O ovom problemu biće govora kasnije). Nakon ovoga za puteve koji se preklapaju nalaze se planovi izvršenja pri čemu je težnja da se minimizira broj ciklusa za sve puteve. Za primer sa slike 9, algoritam planiranja zasnovan na izboru puta čine sledeći koraci:

- korak 1: zadati CFG učiniti acikličnim i odrediti sve puteve izvršenja. Na slici 10 a) za CFG sa slike 9 prikazana su oba puta izvršenja, *path 1* i *path 2*.
- korak 2: Preslikati sva ograničenja u intervale koja se odnose za svaki put izvršenja. Na slilci 10a) ograničenja od C1 do C6 se pridružuju *path 1*, dok se ograničenja C7 i C8 pridružuju putu *path 2*.
- korak 3: Isplanirati izvršenje svakog puta nezavisno, koristeći *clique partitioning* tehniku putem uvodjenja minimalnog broja ciklusa.

Ovaj postupak se izvodi na sledeći način. Na slici 10 b) za svaki *path* (put) sa slike 9a) odredjuje se interval graf, kod koga čvor odgovara intervalu, a poteg ukazuje da oba intervala koja odgovaraju i jednom i drugom čvoru se povezuju potezima koji se preklapaju. "*clique*", koji je u suštini kompletan potgraf, sadrži sve intervale koje se medjusobno preklapaju.



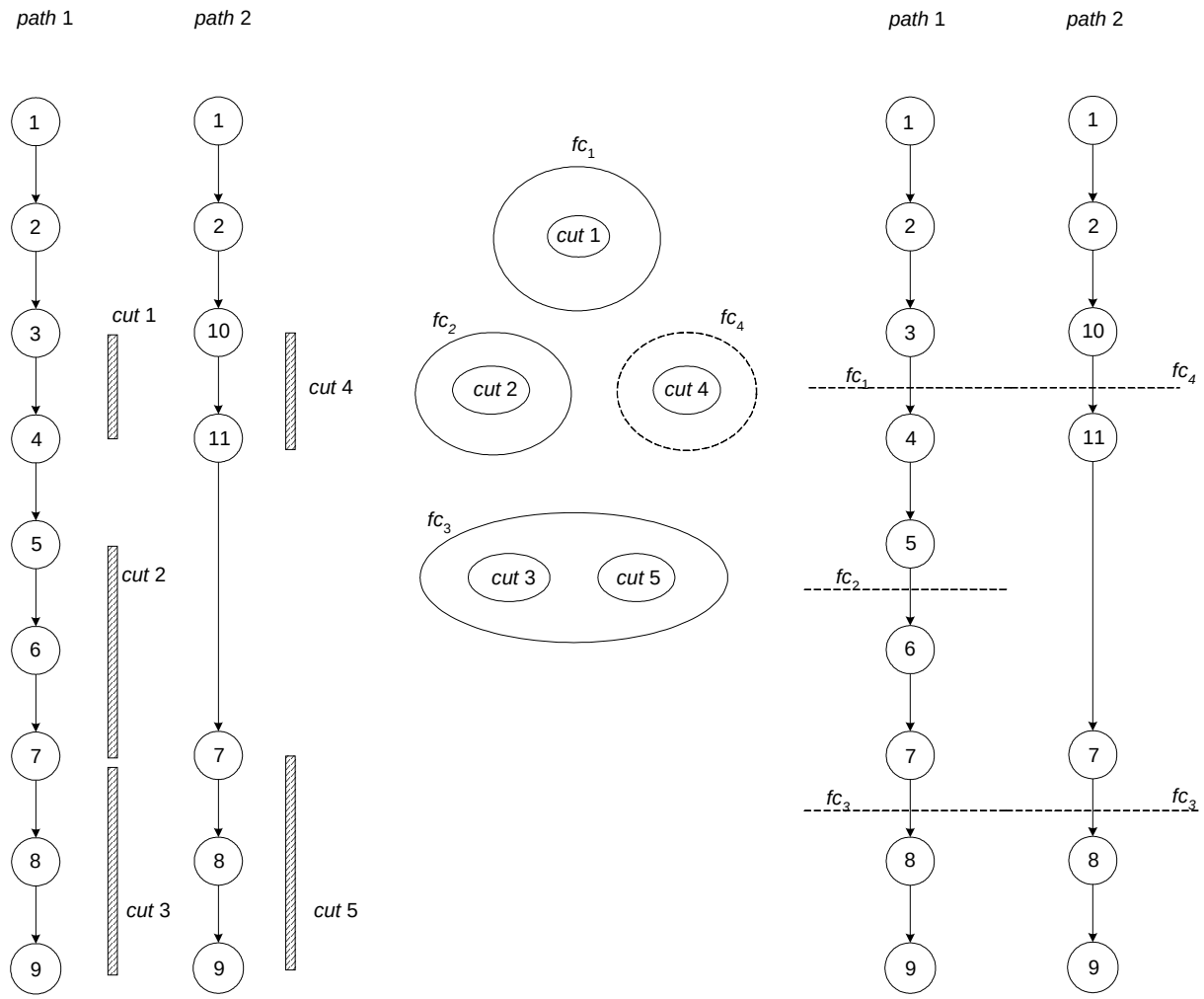
Slika 10 a) dva puta izvršenja kod CFG-a sa slike 9 i odgovarajuća ograničenja; *clique partitioning* interval grafa za svaki put izvršenja

Sada, u okviru preseka intervala u "*clique*"-u, nazvan preklapajući pod-interval, postavlja se upravljački korak koji mora da zadovolji odgovarajuća ograničenja. Pojam "*cut*" se koristi da označi jedan takav pod-interval koji se pridružuje svakom "*clique*"-u. Tako na primer presek "*cut 2*" pridružen "*clique*"-u koji sadrži C4 i C5 predstavlja podinterval od čvorova 5 ka 7 za CFG.

Korak 4: Obavlja se preklapanje svih puteva sa ciljem da se minimizira ukupan broj stanja koristeći tehniku "*clique partitioning*"

Na slici 11 b), kreiran je jedinstveni interval graf za sve "*cut*"-ove sa slike 11 a). Sada iznova, za interval graf, čvor koji odgovara preseku i poteg koji ukazuje na odgovarajuće preseke se preklapaju. Nakon toga, minimalni "*clique partitioning*" interval grafa daje minimalni skup preseka, koji se na sllici 11 označava sa *fcs*, a ispunjava plan izivršenja za sve puteve, što znači da postoji minimalni ukupni broj upravljačkih stanja. Na sllici 11 c) prikazan je konačni plan izvršenja za CFG sa slike 9 koji ima minimalan broj upravljačkih stanja.

Korak 5: Koristeći konvencionalna sredstva za logičku sintezu generiše se upravljačka logika za rezultujući plan izvršenja.

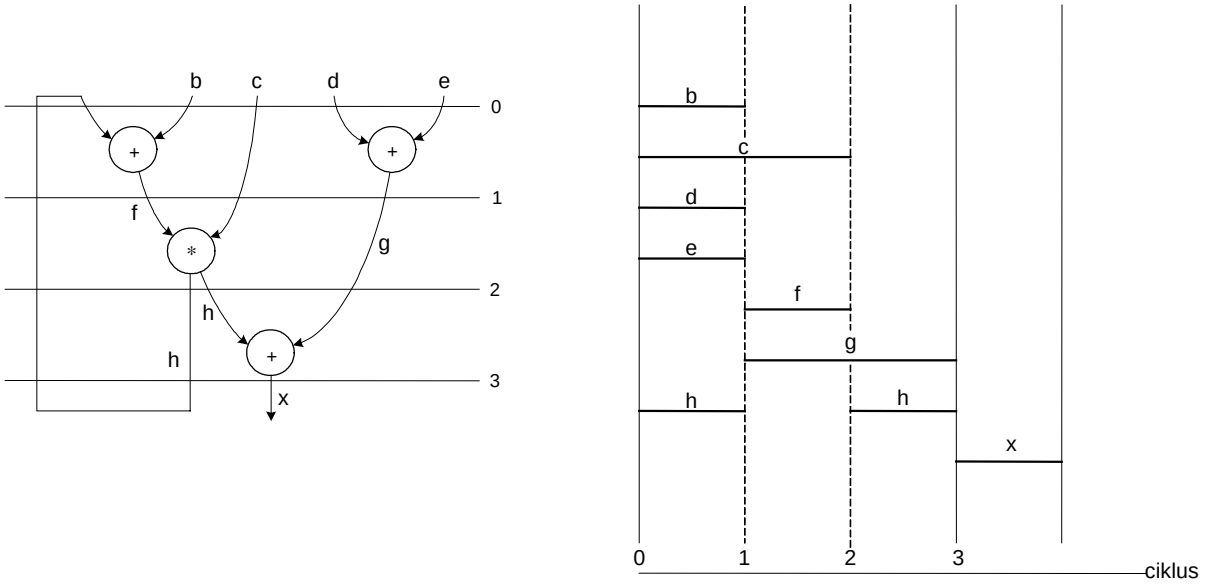


Slika 11 a) svakom preseku odredjen sa slike 9 b) pridružuje se odgovarajući put izvršenja; b) "*clique partitioning*" interval grafa na preseke (*cuts*) kako bi se odredio minimalan broj upravljačkih stanja; c) konačni plan izvršenja sa minimalnim brojem upravljačkih stanja.

Vreme života promenljive

Nakon planiranja, DFG postaje planirani DFG, ili SDFG. Kod acikličnih SDFG-ova, vreme-kreiranja promenljive v, označava se kao $v.birth$, a predstavlja trenutak u toku koga se promenljiva po prvi put definiše. Sa druge strane, vreme ukidanja primenljive v se označava sa $v.death$ i predstavlja trenutak u toku koga se ona zadnji put koristi. Shodno prethodnom, vreme života promenljive v se definiše kao interval $[v.birth, v.death]$.

Ipak kod cikličnog grafa, u toku svake iteracije petlje promenljiva može da ima veći broj kopija identičnog vremena života. Sa ciljem da se pojednostavi prezentacija svi povratni tokovi podataka, na granici petlje, se raskidaju čime se SDFG čini acikličnim, tako da se samo za prvu iterativnu petlju smatra da odredjuje vreme života. Tako na primer, vreme života promenljive f kod cikličnog SDFG-a sa slike 12 a) se definiše kao interval $[1, 2]$. Šta više, za slučaj kada je povratna veza toka podataka prekinuta, vreme života one promenljive koja se nalazi na granicama prekida se deli na dva dela. Ilustracije radi vreme-života promenljive h prikazane na slici 12 a) se definiše kao $[1, 2] \cup [2, 3]$. Tabela koja ukazuje na vreme-života promenljivih moguće je u principu kreirati na osnovu SDFG-a kada se obavi analiza vremena-života promenljivih kod SDFG-a. Na slici 12 b) prikazana je tabela koja odgovara vremenu-života promenljivih za SDFG sa slike 12 a). Kod tabele koja se odnosi na vreme-života vremenski-ciklusi su prikazani po horizontalnoj osi. Vreme-života svake promenljive je označeno linijskim segmentom čija leva i desna ivica odgovaraju vremenu kreiranja i vremenu ukidanja promenljive, respektivno.



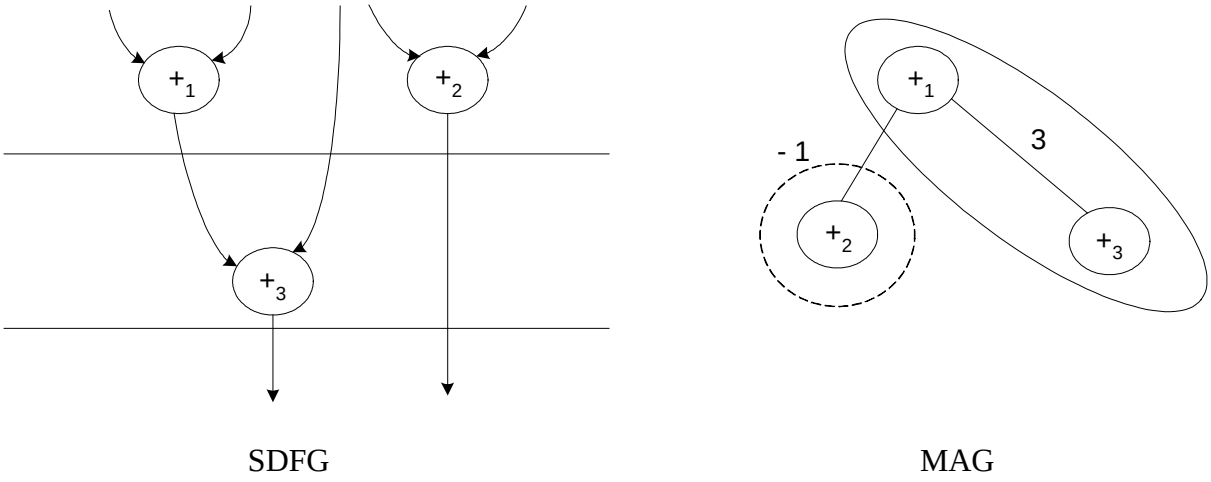
a) b)

Slika 12 a) Primer SDFG-a; b) odgovarajuća tabela-života

Dodela resursa

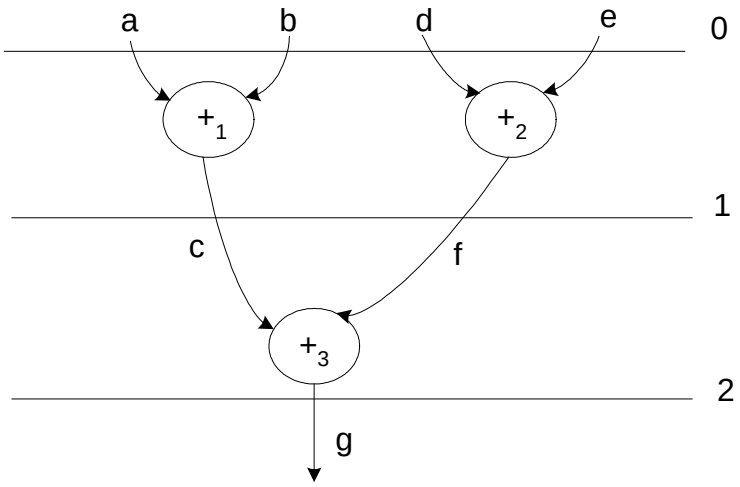
A. Osnovni koncept

Da bi generisali RTA (*register-transfer architecture*) polazeći pri tome od zadatog SDFG-a neophodno je obaviti dodelu resursa, tj raspodeliti hardverske elemente, na sledeći način: a) registre treba koristiti za čuvanje promenljivih; b) module (ili operatore) treba upotrebljavati za izivršenje operacija; i c) veze treba koristiti za medjusobno povezivanje modula i registra kao i prenos signala.
Dodela (alokacija) resursa, se označava sa R, i može se smatrati kao particija $\{R_1, R_2, ..., R_r\}$ od $V_I \cup V_O \cup V_M$, tako da za bilo koje dve promenljive v_i i v_j u R_k, $1 \le k \le r$, odgovarajuća vremena-života se ne preklapaju, pri čemu $V_I, V_O \, i \, V_M$ predstavljaju skupove primarnih ulaza, primarnih izlaza, privremeno promenljivih, respektivno. Moguće je takodje definisati i vreme-kreiranja registra R_i, koje se označava kao $R_i.birth$, kao i vreme-ukidanja, koje se označava kao $R_i.death$, kao najraniji i najkasniji trenutak promenljive koji je dodeljen registru R_i, respektivno. Pored toga, register se naziva ulazni (izlazni) registar ako se registru dodeljuje primarni ulaz (izlaz). Kada je registar istovremeno i ulazni i izlazni, isti se naziva IO regista.
Na sličan način, alokacija modula (ili funkcionalne jedinice), se označava sa M, i može se smatrati kao particija $\{M_1, M_2, ..., M_m\}$ od o, što odgovara skupu svih operacija u *behavioral* specifikaciji, tako da za bilo koje dve operacije o_i i o_j iz M_k, $1 \le k \le m$, izmedju odgovarajućih vremena izvršenja ne dolazi do konflikta. Radi pojednostavljenja, takodje se usvaja, da se samo jedan regularni bibliotečni modul koristi za alokaciju modula. To znači da svaki modul iz biblioteke može da obavi u suštini samo jednu operaciju.
Na osnovu SDFG-a, za potrebe alokacije modula, izvodi se neusmereni graf za alokaciju modula (MAG-*module allocation graph*), kod koga svaki čvor odgovara operaciji SDFG-a, a svaki poteg deljivoj (zajedničkoj) kompatibilnosti tog istog modula izmedju dva čvora koje taj poteg povezuje. Stepen (iznos) deobe kompatibilnosti izmedju operacija istog modula se može zatim rasčlanjivati ako se svakom potegu dodele različite vrednosti koje odgovaraju preferenci-deobe-modula (*module shering preference*-MSP), na osnovu ciljeva koje želimo ostvariti, a to su minimizacija površine čipa, poboljšanje performansi, testabilnost, itd. Na osnovu prethodnog, MAG kod koga postoji dodela potega na principu MSP-a postaje ponderisani graf za dodelu modula. Shodno tome dodela modula se dobija odredjivanjem skupova čvorova u MAG-u pri čemu su svi članovi medjusobno povezani, a suma ponderisanih potega je najveća.
Jedan tipičan primer ponderisanog MAG-a izveden na osnovu SDFG-a je prikazan na slici 13. Predpostavimo da nam stoje na raspolaganju samo dva sabirača, a na osnovu razmatranja koja se tiču uštede čipa MSP izmedju $+_1$ i $+_2$ iznosi –1, a izmedju $+_1$ i $+_3$ iznosi 3. S obzirom da je preferencirana deoba izmedju $+_1$ i $+_3$ veća nego izmedju $+_1$ i $+_2$, jedan će sabirač biti deljiv izmedju $+_1$ i $+_3$, a drugi biće namenjen samo za potrebe $+_2$.

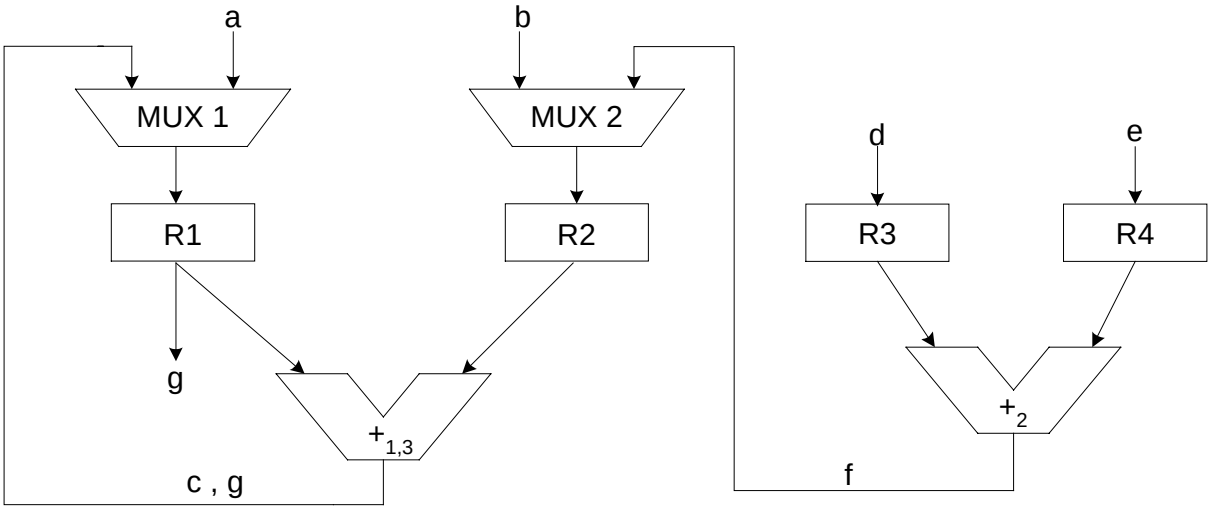


Slika 13 Primer ponderisanog MAG-a izveden na osnovu SDFG-a

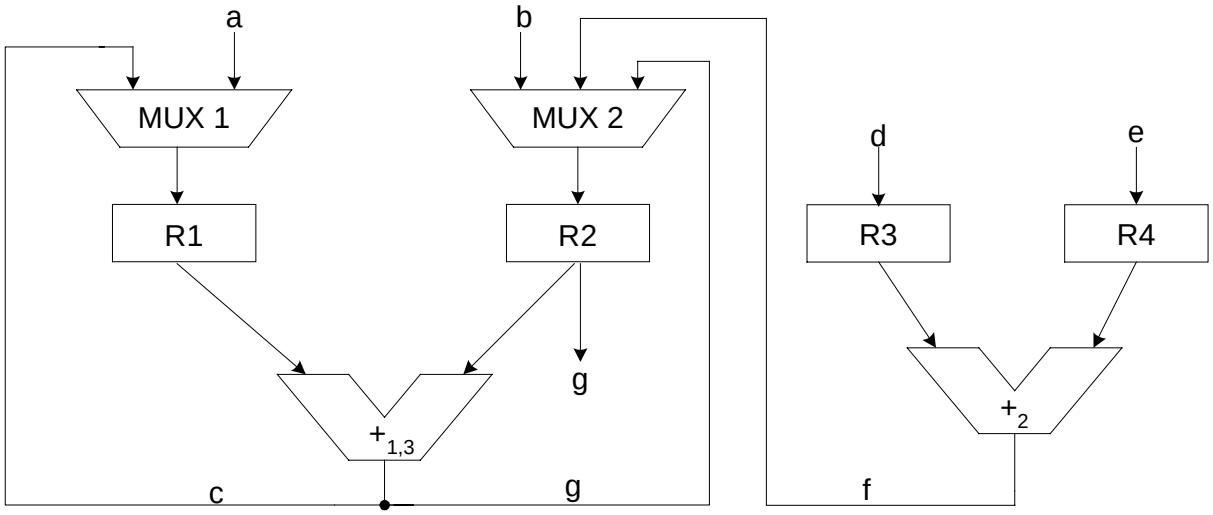
Dodela žica se vrši alokacijom veza koje su namenjene za prenos signala promenljivim koje se čuvaju u registrima ili koriste od strane modula. Na sličan način, za dva signala se može dodelliti ista žica u slučaju kada se odgovarajuća vremena-života ne preklapaju. Na slici 14 prikazan je efekat alokacije kod konačne implementacije. Kao što se vidi alocirane su dve različite arhitekture za realizaciju istog ponašanja koje se specificira od strane SDFG-a. Obe arhitekture imaju po četiri registra, ali se promenljive dodeljuju na različiti način. Tako na primer, na slici 14 b) promenljive *a* , *c* , *g* se dodeljuju registru R1 , *b* i *f* registru R2, *d* registru R3, a *e* registru R4. Sa druge strane kod slike 14 c) promenljive *a, f* i *g* su dodeljene registru R1, promenljive *b* i *c* registru R2, promenljiva *d* registru R3, a promenljiva *e* registru R4. Kod arhitekture sa sllike 14 c) potrebna je jedna veza više, a takodje i MUX2 mora da ima još jedan ulaz više u odnosu na MUX2 sa slike 14 b) da bi prihvatio promenljivu *g.*



a)



b)



c)

Slika 14 a) SDFG; b) jedna moguća arhitektura; c) druga moguća arhitektura