Viša Tehnička Škola - Subotica

Burány Nándor
Divéki Szabolcs

Digitalna elektronika

-skripta-

Subotica, 2007.

Predgovor

Skripta je napisana za studente druge godine Više Tehničke Škole u Subotici i sadrži gradivo predavanja iz predmeta *Digitalna elektronika*. Ovaj predmet predstavlja neposredni nastavak predmeta *Digitalna tehnika* i *Analogna elektronika*. U *Digitalnoj tehnici* studenti su se upoznali sa logičkim osnovama digitalnog projektovanja. Glava o logičkim kolima u *Analognoj elektronici* pružala je uvid u hardverska rešenja koja čine osnove digitalnih uređaja.

Skripta obrađuje aktuelna rešenja za realizaciju digitalnih kola prema programu za predmet *Digitalna elektronika*. Polazi se od (I. deo) opštih pitanja vezanih za fizičku realizaciju. U nastavku (II. deo) se razmatra projektovanje zasnovano na kolima malog i srednjeg stepena integracije (*SSI, MSI*) koje je svoj vrhunac dostiglo sedamdesetih i osamdesetih godina XX veka.

Sedamdesetih godina XX veka su se pojavila kola koja se programiraju softverski (mikroprocesori, mikrokontroleri) a ubrzo zatim i kola koja se programiraju hardverski (*programmable logic device – PLD*). U okviru drugih predmeta će se studenti upoznati sa kolima koja se softverski programiraju. Digitalno projektovanje zasnovano na *PLD*-ovima se detaljno obrađuje u okviru ovog predmeta. U cilju toga III. deo skripte prikazuje savremeni alat za hardversko programiranje, jezik za opis hardvera *Verilog*. U ovom predmetu se stiže do opisa jezika i do računarskih simulacija. U okviru predmeta *Projektovanje elektronskih uređaja* će se studenti upoznati sa softverima za automatsku sintezu digitalnih kola i sa samim postupkom programiranja.

Karakteristično je za savremeni način projektovanja da veći deo zadatka u digitalnim sistemima obavljaju mikroprocesori i mikrokontroleri koji se softverski programiraju kao i *PLD*-ovi koji se hardverski programiraju dok je uloga *SSI* i *MSI* kola da obavljaju pomoćne zadatke (pobuđivanje, prilagođavanje nivoa itd.).

Uređaji koji se softverski programiraju imaju širu oblast primene i veoma su pogodni za obavljanje složenih zadataka kao što su upravljanje, merenje, prikazivanje rezultata itd. Međutim, postoje i takvi zadaci obrade signala i upravljanja koji se ne mogu rešiti mikroprocesorima ili mikrokontrolerima, jer su oni previše spori za takve primene. U ovom slučaju rešenje nam pružaju kola sa hardverskim programiranjem.

U izvesnoj meri uređaji koji se softverski programiraju i uređaji koji se hardverski programiraju su konkurencija jedan drugom ali u velikom broju primena oni se međusobno dopunjuju. U mnogim uređajima pored mikrokontrolera se mogu naći i jedan ili više *PLD* kola.

Programabilna logička kola su danas dostigla takav nivo složenosti da je u njima moguće realizovati čak i kompletan mikrokontroler. Na ovaj način korisnik može da projektuje i hardverski programira integrisano kolo sa željenim hardverskim resursima koji se zatim softverski programira sa željenim skupom naredbi. Ovakvi zadaci često prevazilaze znanje jednog inženjera i zbog toga proizvođači *PLD* uređaja podržavaju ovaj pravac projektovanja nudeći gotova rešenja projektantima.

Autori

Uvod

Digitalne realizacije su karakteristične u mnogim oblastima savremene tehnike. Računarska tehnika, telekomunikacije i upravljanje industrijskim procesima su oblasti koje najviše prednjače u primeni digitalnih rešenja.

U početku su samo mehanički i elektromagnetni prekidači (releji) bili na raspolaganju za realizaciju digitalnih uređaja. Bilo je pokušaja za realizaciju digitalnih uređaja primenom elektronskih cevi koje su dugo vremena držali monopol na polju analogne obrade signala. Prvi elektronski računar je sagrađen pomoću elektronskih cevi. Velike dimenzije, nedovoljna pouzdanost u radu i značajna potrošnja su bili faktori koji su sprečili širu primenu ovih uređaja.

Nakon otkrića tranzistora, skoro istovremeno su se pojavile analogne i digitalne primene. Povezivanjem tranzistora, dioda i otpornika bilo je moguće izgraditi brza digitalna kola malih dimenzija i male potrošnje. Od kraja pedesetih godina XX veka osnovu digitalne elektronike nesporno su dala integrisana rešenja. Razvoj tehnologije za proizvodnju poluprovodničkih komponenata omogućilo je projektovanje (integrisanje) sve komplikovanijih električnih kola na površini jedne kristalne ploče.

Na početku razvoja integrisanja električnih kola samo nekoliko logičkih kapija je bilo moguće smestiti na jednu pločicu. Ova tehnologija je dobila naziv mali stepen integracije (*small scale of integration – SSI*). Posle ovoga tehnologija srednjeg stepena integracije (*medium scale of integration – MSI*) je omogućila smeštanje složenijih celina (multipleksori, brojači itd.) na jednu pločicu. U to vreme projektovanje digitalnih uređaja se sastojalo od spretne kombinacije SSI i MSI integrisanih kola.

Smatralo se da će pravac razvoja dovesti do sve složenijih električnih kola, koja će u potpunosti integrisati sva neophodna digitalna rešenja za funkcionisanje nekog uređaja. Ovakva integrisana kola su projektovana npr. za digitalne satove, merne uređaje itd. Međutim, ubrzo je postalo jasno da su primene toliko brojne da nije ekonomično posebno razvijati integrisana kola za svaku primenu (*application specific integrated circuit – ASIC*). Razvoj i proizvodnja integrisanih kola je ekonomična samo ako se proizvodnja istih vrši u velikim serijama.

Prepoznavanje ovih ograničenja je usmerilo razvoj u dva pravca. Sa jedne, strane pojavile su se mikroprocesori koji se ne mogu samostalno primeniti u konkretne svrhe, ali njihovim povezivanjem sa odgovarajućim pomoćnim električnim kolima (memorije, prilagodni stepeni itd.) i dodavanjem softvera moguće je rešiti proizvoljne probleme. Sa druge strane, digitalna kola koja se hardverski programiraju (*programmable logic device – PLD*) sadrže veliki broj unapred proizvedenih logičkih blokova, i hardverskim programiranjem se mogu povezati na način da obavljaju željenu funkciju. Ova električna kola pripadaju kolima visokog stepena integracije (*large scale integration – LSI*) odnosno kolima vrlo visokog stepena integracije (*very large scale integration – VLSI*).

Skriptu predlažemo svakome ko želi da se upozna sa osnovama digitalne elektronike.

Autori

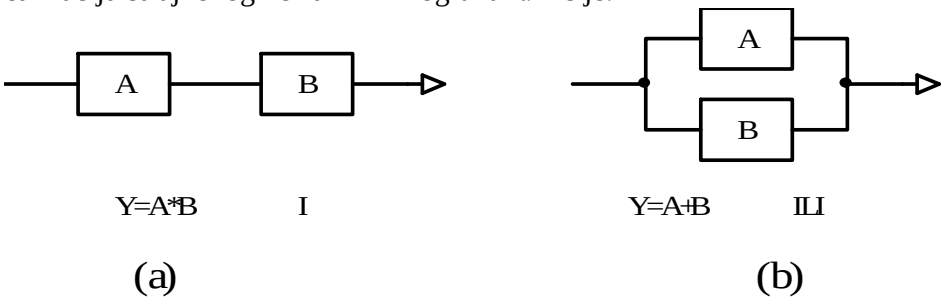
I. Pitanja vezana za realizaciju digitalnih elektronskih kola

1. Fizičke osobine digitalnih električnih kola

U ovoj glavi se razmatraju fizičke osobine digitalnih kola. Karakteristike koje se ovde razmatraju podjednako važe kako za *SSI* i *MSI* jednostavna električna kola tako i za komplikovanija (*LSI*, *VLSI*) električna kola koja se hardverski i softverski programiraju. Razlog za to je velika sličnost u polaznim materijalima i proizvodnim procesima kao i u topološkim rešenjima kod svih digitalnih električnih kola.

1.1. *Strujna logika-naponska logika*

U početku digitalni uređaji su se gradili pomoću prekidača i releja. Logičke funkcije kod tih uređaja su se realizovale pomoću takozvane strujne logike: ako je struja ulazila u objekat upravljanja zahvaljujući odgovarujućim stanjima prekidača, to se smatralo stanjem logičke jedinice. Ako nije bilo struje zbog otvorenosti nekog prekidača, to se smatralo stanjem logičke nule. Slika 1-1 prikazuje realizaciju strujne logike za *I* i *ILI* logičke funkcije.

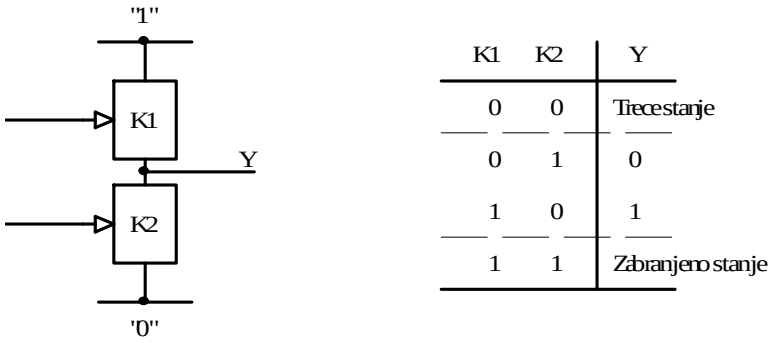


Slika 1-1: Realizacija (a) I i (b) ILI logičke funkcije pomoću strujne logike.

Električna kola sa strujnom logikom su skoro izašla iz upotrebe, ali proteklih godina kod pojedinih rešenja za integrisana kola su ponovo ušla u centar interesovanja. Kod većine današnjih digitalnih kola je naponska logika dominantna. Potrebna logička stanja se i kod naponske logike postižu zatvaranjem ili otvaranjem prekidača. U ovom slučaju cilj nije protok struje od ulazne tačke (tačaka) do izlazne tačke (tačaka) nego dobijanje određenih standardnih naponskih nivoa za reprezentaciju logičke nule i jedinice.

Slika 1-2a prikazuje principijelnu šemu jednog digitalnog (logičkog) elementa sa naponskom logikom. Dva prekidača (*K1*, *K2*) su redno vezana (mogu se posmatrati kao naponski delitelji). Slobodni izvodi su vezani na izvore koji obezbeđuju naponske nivoe koji odgovaraju logičkoj jedinici ("1") i nuli ("0"). Obično je reč o jednom izvoru, čiji se jedan kraj veže na potencijal zemlje koji se smatra logičkom nulom, dok potencijal prisutan na drugom kraju izvora se smatra logičkom jedinicom. Teoretski gledano, prekidači se mogu uključiti na četiri različita načina koji su prikazani u tabeli na slici 1-2b.

Funkcionisanje savremenih digitalnih uređaja se obično zasniva na dva naponska nivoa, odnosno na dvema logičkim vrednostima. Vrednost izlaza u prvom redu tabele je neodređena što ukazuje na odstupanje od definicije dvovalentne logike. Pošto su oba prekidača isključena, izlaz Y prelazi u stanje visoke impedanse koje se još naziva i treće stanje (engl. *three-state* ili *3-state*). U tom slučaju drugi elementi priključeni na istu tačku određuju naponski nivo izlaza datog logičkog elementa. Ova mogućnost se koristi kod onih digitalnih uređaja kod kojih se komunikacija između više logičkih elemenata ostvaruje kroz jedinstveni zajednički provodnik. U različitim vremenskim intervalima se aktiviraju različiti logički elementi. Aktivni element u datom trenutku upravlja provodnikom, dok su drugi elementi u stanju visoke impedanse i njihovi uticaji na vrednost logičkog nivoa se mogu zanemariti.



Slika 1-2: (a) Principijelna šema logičkog elementa sa naponskom logikom i (b) varijacije stanja prekidača.

U drugom redu tabele *K2* provodi, dok *K1* ne provodi. Znači, izlaz Y je u kratkom spoju sa zemljom (tačka sa nultim potencijalom) i na taj način se ostvaruje logička nula. Treći red odgovara suprotnom slučaju, *K1* provodi dok *K2* ne provodi. Znači, na izlazu se pojavljuje potencijal koji odgovara stanju logičke jedinice. Realni prekidači u uključenom stanju ne predstavljaju idealni kratak spoj i kao rezultat toga dolazi do promene napona na izlazu logičkog elementa pri opterećenju. Prema tome, u realnim kolima, vrednosti logičkih nivoa odstupaju od idealnih. Slično ovome, vremena uključivanja i isključivanja su konačna i zato se uticaji upravljačkih signala prenose sa izvesnim kašnjenjima što dovodi do kašnjenja u funkcionisanju logičkih elemenata. Ovi problemi se detaljno obrađuju u narednim poglavljima.

Kombinacija prikazana u poslednjem redu tabele predstavlja nedozvoljeno stanje. Prekidači *K1* i *K2* ne smeju biti istovremeno u stanju provođenja, jer bi to kratko spojio krajeve naponskog izvora. Velika struja koja bi nastala kao posledica kratkog spoja, dovela bi do uništenja prekidača.

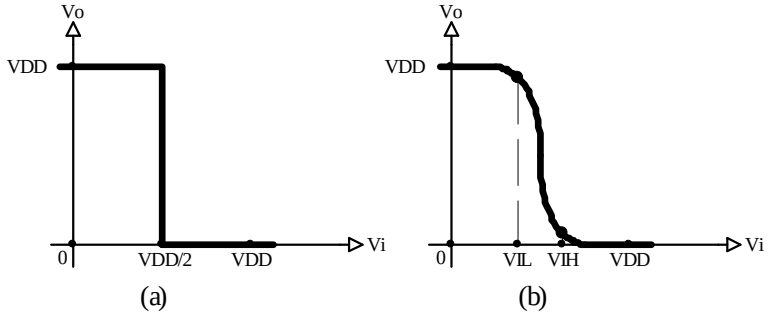
1.2. *Fizičke karakteristike*

Kataloški podaci digitalnih kola pored opisa logičkih funkcija daju i fizičke karakteristike. U fizičke osobine spadaju: prenosna karakteristika, nominalni logički nivoi, margine smetnji (margina šuma), razna kašnjenja, opteretljivost izlaza, potrošnja, temperaturni opsezi, kućište itd. U narednim tačkama su date definicije ovih karakteristika i analiza njihovih uticaja na funkcionisanje digitalnih kola.

1.2.1. Prenosna karakteristika

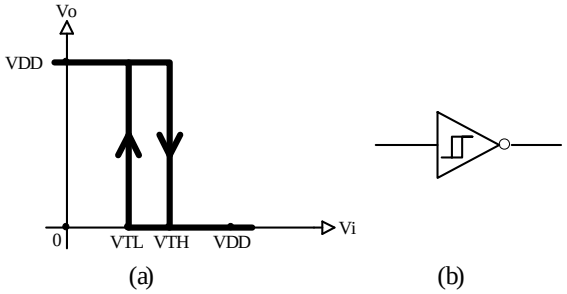
Ako se upravljanje prekidačima logičkog elementa prikazanog na slici 1-2a reši na taj način da zajednički upravljački signal otvara i zatvara te prekidače u protivfazi, dobija se idealizovana prenosna karakteristika prikazana na slici 1-3a. Dok je vrednost ulaznog upravljačkog napona ispod polovine napona napajanja ($V_{CC}/2$), vodi gornji prekidač i izlaz je na visokom logičkom nivou. U suprotnom slučaju, kada je vrednost upravljačkog napona iznad polovine napona napajanja, provodi donji prekidač i izlaz je na niskom logičkom nivou. Znači, kada je na ulazu nizak logički nivo, izlaz je na visokom i suprotno. Dobijana karakteristika predstavlja *NE* logičku funkciju (logički invertor).

U realnim slučajevima otpornost prekidača se menja kontinualno pod uticajem upravljačkih signala a ne skokovito. Rezultat ovoga je prenosna karakteristika realnih invertora koja je prikazana na slici 1-3b. Na dijagramu su označene tačke kod kojih je vrednost nagiba -1. Između ovih vrednosti ulaznog napona (V_{IL}, V_{IH}) nagib krive je velik i zbog toga mala promena ulaznog napona je u stanju da promeni nivo izlaznog napona sa logičke nule na logičku jedinicu i obrnuto. Iz ovog razloga nepoželjno je funkcionisanje digitalnih kola u ovom intervalu (zabranjena zona).



Slika 1-3: Prenosne karakteristike logičkog invertora: (a) idealan slučaj, (b) realan slučaj.

Pored jednoznačnih krivih (slika 1-3) primenjuju se i dvoznačne, histerezisne (Šmitove) krive (slika 1-4a). Zahvaljujući odgovarajućoj unutrašnjoj pozitivnoj povratnoj sprezi, prelaz sa jednog logičkog nivoa na drugi je skokovit umesto da bude postepen. Na ovaj način moguće je digitalne signale sa dugačkim vremenima uzlaza i silaza uobličiti u pravougaone signale što poboljšava otpornost na smetnje (tačka 1.2.3). Šematske oznake digitalnih kola sa Šmitovom prenosnom karakteristikom obično sadrže i simbol Šmitove prenosne karakteristike: na slici 1-4b je prikazana šematska oznaka invertora sa Šmitovom prenosnom karakteristikom. Histerezisnu karakteristiku je moguće ugraditi u ulazni stepen kod bilo kog digitalnog kola.



Slika 1-4: (a) Prenosna karekteristika invertora sa histerezisom i (b) njegova šematska oznaka.

1.2.2. Logički nivoi

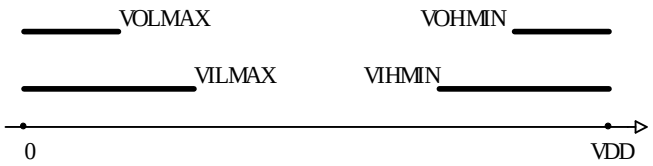
Logički nivoi izlaza (V_{OL}, V_{OH}) digitalnih kola su približno stabilni za intervale koje se nalaze ispod i iznad zabranjene zone dok je napon napajanja konstantan i ne zavisi značajno od ulaznog signala ili promene struje opterećenja. Naponska razlika između dva nivoa se naziva logičkom amplitudom.

Kod pažljivo projektovanih digitalnih kola nizak izlazni logički nivo je u svakom slučaju značajno niži od maksimalne vrednosti niskog ulaznog logičkog nivoa ($V_{OL}<V_{IL}$). Slično, visok izlazni logički nivo značajno prevazilazi donju graničnu vrednost visokog ulaznog logičkog nivoa ($V_{OH}>V_{IH}$).

Ova dva uslova je potrebno ispuniti da bi se elementi digitalnih kola (koji mogu biti kompletna integrisana kola ili njihovi unutrašnji podsklopovi) mogli vezati u kaskadu. Obrada signala kod digitalnih sistema se obično vrši u više stepena i zato je kaskadna veza digitalnih kola neizbežna.

1.2.3. Margine smetnji

Kako za ulazne tako i za izlazne logičke nivoe kataloški podaci ne sadrže konkretne vrednosti već su dati intervali koji su funkcije eventualnih promena u naponu napajanja, promene opterećenja, odstupanja u tehnološkom procesu i drugih uticajnih veličina. Prikazivanjem ovih intervala jedan iznad drugog (slika 1-5), moguće je odrediti maksimalne amplitude smetnji (šuma) na koje su otporni ulazi digitalnih kola. Sve dok su vrednosti amplitude smetnji manje od margine smetnji, logički nivoi generisani na izlazu digitalnog kola će biti ispravni.



Slika 1-5: Prikaz margine smetnji kod digitalnih kola.

Prema slici, margina smetnji za nizak ulazni logički nivo je data sledećom formulom:

$$NM_0 = V_{IL\max} - V_{OL\max}$$

dok u slučaju visokog logičkog nivoa margina smetnji je:

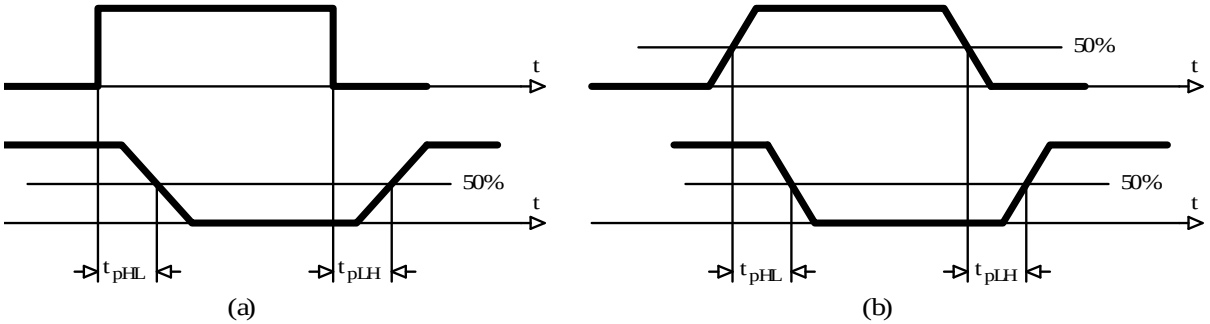
$$NM_1 = V_{OH\min} - V_{IH\min}\,.$$

Kao što ćemo kasnije videti, velika margina smetnji predstavlja prednost ali zahteva veliki napon napajanja koji dovodi do velike potrošnje. Korišćenje integrisanih kola sa velikom marginom smetnji se preporučuje u industrijskim primenama gde je nivo smetnji značajan.

1.2.4. Kašnjenja

Uključivanje i isključivanje tranzistorskih prekidača u digitalnim kolima se vrši u konačnom vremenu. Kašnjenje električnog kola je vreme koje protiče između pojave ulaznog (upravljačkog) signala i formiranja izlaznog logičkog nivoa. Pored konačnog vremena uključivanja i isključivanja na kašnjenja utiču i parazitne kapacitivnosti. Struje koje pune i prazne parazitne kapacitivnosti su konačne i zato se promene logičkih nivoa dešavaju u konačnom vremenu.

Slika 1-6a prikazuje kašnjenja jednog logičkog invertora pri pobudi sa idealizovanim ulaznim signalom (pravougaoni signal). Nakon pojave skoka u ulaznom signalu do promene logičkog nivoa izlaza dolazi tek posle proteklog vremena kašnjenja t_{pHL} odnosno t_{pLH}. U realnosti (slika 1-6b) nije moguće obezbediti idealni pravougaoni signal na ulazu digitalnih kola. Umesto toga, signali koji se dovode na ulaze se dobijaju iz prethodnih stepena koji su veoma slični posmatranom invertoru. U tom slučaju, kašnjenje se računa kao proteklo vreme između preseka signala sa linijom koja predstavlja polovinu logičke amplitude ili napona napajanja (na slici 1-6b obeležen sa 50%). Obično se razlikuju kašnjenja pri silaznim (t_{pHL}) i uzlaznim ivicama (t_{pLH}).

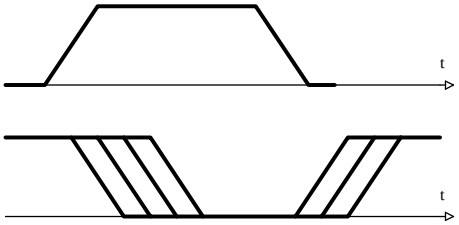


Slika 1-6: Kašnjenja kod digitalnih kola: (a) pri idealizovanom ulaznom signalu, (b) u slučaju realnog ulaznog signala.

Pored unutrašnjih kašnjenja kod integrisanih kola postoje još i kašnjenja na vodovima. Ovo je posebno izraženo kod vodova štampanih električnih kola ali se ne smeju zanemariti ni kod *LSI*, *VLSI* integrisanih kola jer se na unutrašnjim vezama između pojedinih blokova javljaju značajna kašnjenja.

Kašnjenja se ne smeju smatrati konstantnim veličinama. Zbog odstupanja između pojedinih primeraka u serijskoj proizvodnji, zbog odstupanja u naponu napajanja, temperaturi i u

drugim veličinama, proizvođači u katalоškim podacima navode intervale kašnjenja umesto konkretnih vrednosti. Na vremenskim dijagramima se intervali crtaju na način koji je prikazan na slici 1-7. Kod analize uticaja kašnjenja treba uzeti u obzir najnepovoljniji mogući slučaj.



Slika 1-7: Grafički prikaz intervala kašnjenja u katalоškim podacima za digitalna kola.

1.2.5. Opteretljivost izlaza

Otpornost prekidača u uključenom stanju kod digitalnih kola je uvek veća od nule dok u isključenom stanju manja od beskonačnog. Upravljanje ovim prekidačima zahteva konačne vrednosti struja. Pri projektovanju digitalnih sistema često se javlja potreba da izlaz jednog logičkog elementa obezbeđuje ulazni signal za jedan ili više njemu sličnih elemenata. Retko se javlja situacija kada je na izlaz potrebno vezati druge potošače (*LED*, prenosne linije itd.).

Zbog konačne vrednosti ulaznih i izlaznih otpornosti, broj logičkih elemenata koji se mogu vezati na jedan izlaz je konačan. U slučaju preopterećenja izlazni logički nivoi se pomeraju i reakcija narednog stepena na promenjene logičke nivoe može da bude pogrešna. Tokom projektovanja neophodno je uzeti u obzir da sa povećanjem opterećenja dolazi i do porasta vremena kašnjenja.

Opteretljivost izlaza se zadaje brojem ulaza logičkih kola koji se mogu vezati na izlaz a ne veličinom struje koju je u stanju da daje izlaz. Veličine ulaznih struja variraju i unutar jedne familije integrisanih kola i zbog toga pri određivanju opteretljivosti izlaza proizvođači računaju sa tzv. standardnim ulaznim strujama.

Obično pri različitim logičkim nivoima (logička nula ili jedinica) ulazi nejednako opterećuju izlaze. Pri određivanju opteretljivosti izlaza uvek treba uzeti u obzir najnepovoljniju situaciju.

1.2.6. Potrošnja

U toku rada, digitalna kola troše struju iz izvora napajanja. U katalоškim podacima se daje prosečna vrednost struje jer je ona zavisna od logičkih nivoa na izlazu. Pored toga, potrošnja digitalnih kola zavisi još i od učestalosti promene logičkih nivoa (frekvencija rada). Razlog za to je da gornji i donji prekidači na slici 1-2a pri promeni logičkih nivoa na kratko vreme provode istovremeno da bi se smanjila vremena kašnjenja. Otpornost prekidača u provodnom stanju je konačna i zbog toga u fazi preklapanja (kada oba prekidača vode) neće se javiti beskonačna struja ali će se potrošnja značajno povećati u odnosu na statičko stanje. Slično, i punjenje i pražnjenje parazitnih kapacitivnosti u digitalnim kolima dovodi do povećanja potrošnje. Zbog kratkotrajnih strujnih impulsa potrebno je filtrirati napon napajnja digitalnih kola kondezatorima dobrog kvaliteta smeštenih u blizini samog kola.

Maksimalna frekvencija rada digitalnog kola je u tesnoj vezi sa potrošnjom i kao osnov za upoređivanje različitih familija integrisanih kola uzima se proizvod potrošnje i kašnjenja. Ovaj proizvod se označava sa *PDP (power-delay product)* i jedinica mere je ista kao i za energiju (*J* odnosno *pJ*).

$$PDP = P_D t_p .$$

Razlog zašto se potrošnja navodi u *pJ* je to što su kašnjenja red veličine *ns* a potrošnje su reda veličine *mW*. Smatra se da je neka familija integrisanih kola bolja od druge ako je *PDP* njenih logičkih elemenata realizovanih u datoj tehnologiji manja od druge.

1.2.7. Temperaturni opsezi

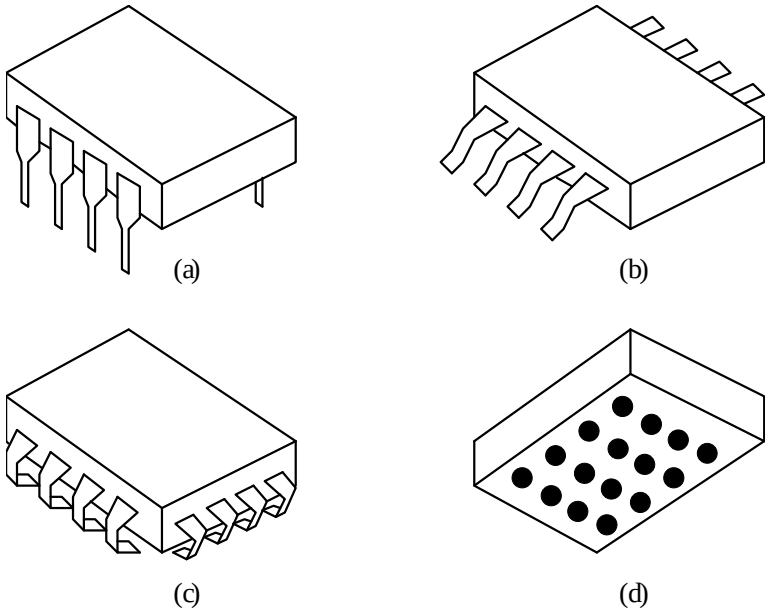
U katalоškim podacima proizvođači definišu temperaturni opseg rada i skladištenja integrisanih kola. Postoje tri temperaturna opsega: za komercijalne, industrijske i vojne primene. Obično, svaki tip integrisanog kola je proizveden u sve tri verzije, za sva tri temperaturna opsega. Opseg temperature koju podrazumeva komercijalna upotreba je od 0^oC do $+70^oC$, opseg temperature za industrijsku primenu je od -25^oC do $+85^oC$, dok za vojnu primenu je od -55^oC do $+125^oC$. Temperatura skladištenja je jedinstvena za sve tri primene i pripada opsegu od -65^oC do $+150^oC$.

Unutrašnja temperatura integrisanih kola je funkcija temperature okoline i gubitaka (potrošnje). Prema katalоškim podacima, najveća dozvoljena unutrašnja temperatura je obično 150^oC. Manja temperatura okoline omogućava rad sa većim gubicima u kolu a da se pri tome ne pređe granica od 150^oC. Ove zavisnosti se daju tabelarno ili u obliku dijagrama. Neznatno prekoračenje opsega radne temperature ne dovodi do trenutnog kvara, ali se karakteristike integrisanog kola značajno menjaju.

Ugradnja integrisanih kola se obično vrši lemljenjem. Ovo predstavlja značajno temperaturno opterećenje za kolo. Proizvođači dozvoljavaju lemljenje pri temperaturama od $250^oC – 300^oC$ u vremenskom trajanju od *10s - 60s*. Pri mašinskom lemljenju integrisanih kola definiše se temperaturni profil koji dato kolo može da izdrži.

1.2.8. Kućišta

Poluprovodničke pločice na kojima su realizovana integrisana kola se smeštaju u odgovarajuća kućišta radi adekvatne mehaničke čvrstoće i dobijanja masivnih priključaka. Integrisana kola su dugi niz godina bila proizvedena u takozvanom *DIL* (*dual in line* – dvorednom) kućištu (slika 1-8a). Ugradnja se sastojala od umetanja nožica integrisanog kola u otvore na štampanoj pločici i od samog lemljenja. Rastojanje između nožica koje se nalaze u jednom redu je standardno i iznosi *0,1 inča* (*2,54 mm*). Za izradu kućišta se koriste materijali koji imaju dobre izolacione karakteristike kakvi su npr. plastika i keramika.



Slika 1-8: Kućišta integrisanih kola: (a) DIL kućište, (b) SMD kućište tipa SO, (c) SMD kućište tipa PLCC, (d) SMD kućište tipa BGA.

Povećanje stepena integracije neminovno je dovelo do povećanja broja izvoda na integrisanim kolima. Tehnologija ugradnje integrisanih kola umetanjem nožica u rupe na štampanoj pločici nije omogućavala značajno smanjivanje rastojanja među nožicama. Ovo je dovelo do pojave

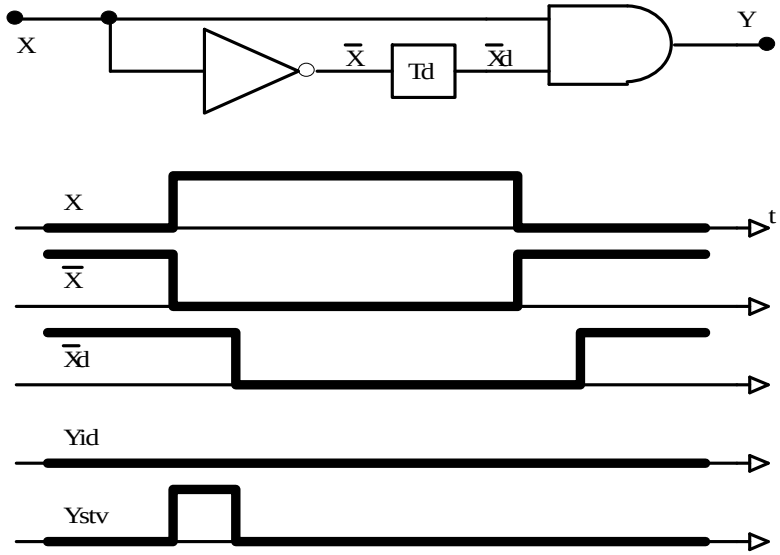
novih kućišta koja se površinski montiraju (*surface mounted device – SMD*). Kućište *SO* (slika 1-8b) je slično *DIL* kućištu, sa razlikom da su izvodi savijeni u obliku krila galeba kako bi obezbedili dobro naleganje na površinu štampane pločice. U početku su rastojanja između nožica *SO* kućišta bila *0,05 inča* (*1,27 mm*) a kasnije su ove mere još smanjene. Kod *PLCC* kućišta (slika 1-8c) postoje izvodi na sve četiri strane, a za dodatno smanjenje dimenzija, nožice su savijane ispod kućišta. Kod *BGA* kućišta (slika 1-8d) ne postoje izvodi u klasičnom smislu. Na donjoj strani kućišta su formirana metalna ostrva u ravni sa izolatorom. Pri lemljenju metalna ostrva se pozicioniraju na lemne kugle smeštene na štampanoj pločici i zagrevanjem celog integrisanog kola se formiraju električni kontakti.

Kućišta za površinsku montažu su bila skuplja u početku i proces montaže je bio komplikovan. Međutim, razlika u cenama je već oko 1990 godine bila na strani *SMD* tehnologije s tim da ugradnju vrše pretežno roboti (osim kod maloserijske proizvodnje i popravki). Primena *SMD* komponenti omogućava montiranje kola sa obe strane štampane pločice što dalje smanjuje dimenzije uređaja.

1.3. *Posledice kašnjenja: hazardi*

Posledice kašnjenja (tačka 1.2.4), osim kašnjenja u formiranju izlaznih signala mogu biti i privremene ili trajne greške u funkcionisanju uređaja. Greške koje nastaju zbog kašnjenja se nazivaju hazardima jer su nepredvidive i imaju štetne posledice.

Ako se posmatra jednostavno kolo koje je dato na slici 1-9, može se ustanoviti da u idealnom slučaju vrednost izlaza bi trebala uvek da bude logička nula jer (zbog prisustva invertora na jednom ulazu *I* kola) ne mogu se pojaviti istovremeno dve jedinice na ulazima što je uslov za formiranje logičke jedinice na izlazu. U realnim okolnostima neophodno je uključiti u razmatranje i kašnjenje invertora koje je predstavljeno blokom T_d. Zbog kašnjenja, nakon uzlazne ivice promenljive *X*, na oba ulaza *I* kola stižu logičke jedinice u kratkom intervalu zbog čega vrednost izlaza privremeno postaje logička jedinica.



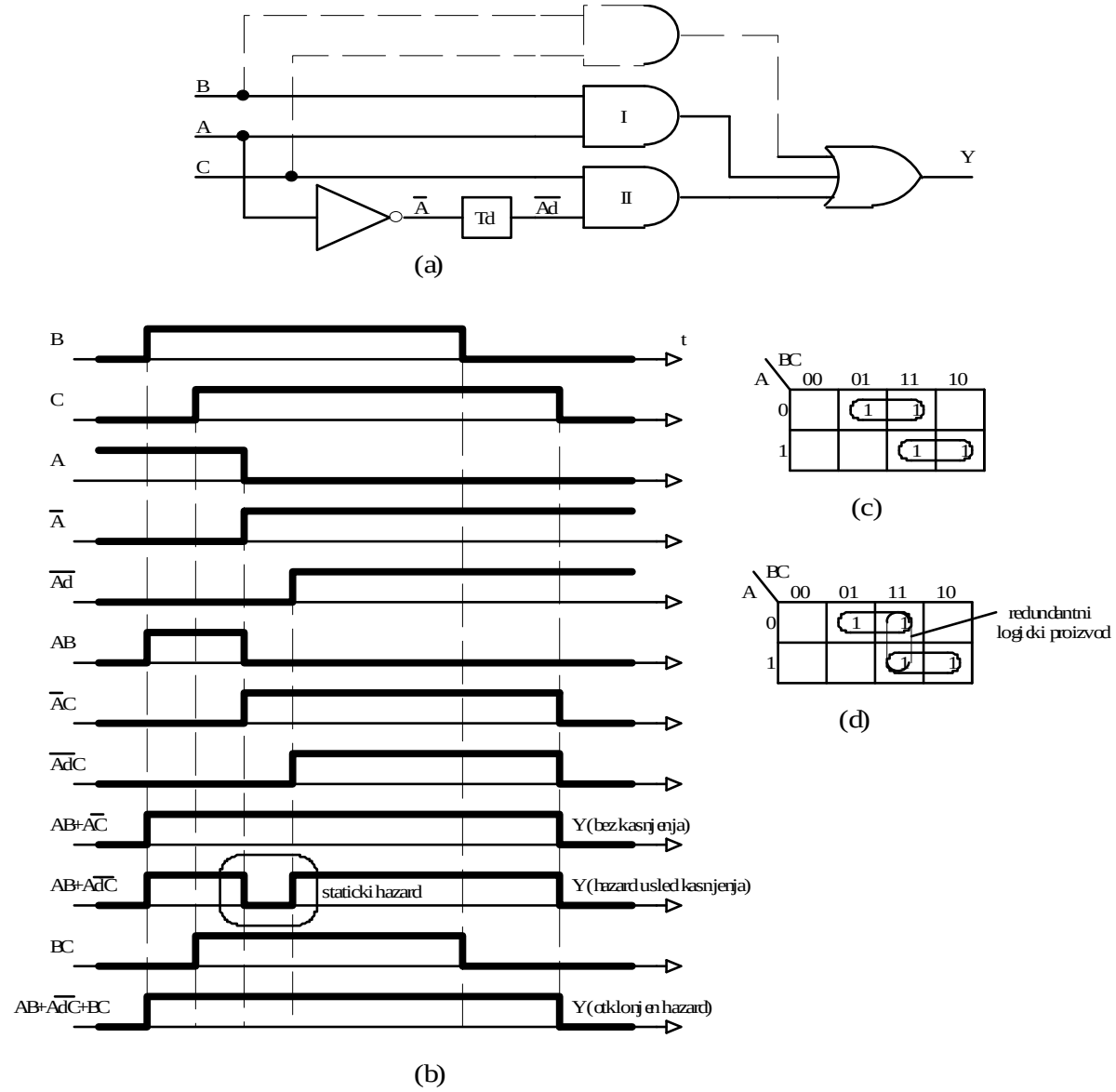
Slika 1-9: Pojava hazarda uzrokovana kašnjenjem u jednostavnom digitalnom kolu.

Prikazane pojave u nekim slučajevima mogu da budu i korisne: na ovaj način moguće je generisati kratkotrajne impulse u okolini uzlazne ivice datog signala bez korišćenja pasivnog diferencijatora ili nekog monostabilnog kola. U nastavku će se vršiti analiza različitih hazardnih pojava i tražiće se rešenje za otklanjanje hazarda.

1.3.1. Statički hazardi

Statičkim hazardom nazivamo kratkotrajne pojave impulsa na izlazu nekog logičkog kola pod uticajem promene u vrednosti ulaznih signala u intervalu kada vrednost logičke funkcije izlaza treba da bude sve vreme konstantna. Primer na slici 1-9 predstavlja slučaj statičkog hazarda.

Slika 1-10a prikazuje jedno složenije digitalno kolo sa hazardom koji treba otkloniti. Logička funkcija prikazanog digitalnog kola se sastoji od zbira dva logička proizvoda i realizuje se dvostepenom *I-ILI* logičkom mrežom. Jedino se kašnjenje invertora uzima u obzir radi jednostavnije analize.



Slika 1-10: Statički hazard i njegovo otklanjanje: (a) minimizirana mreža, (b) vremenski dijagrami, (c) Karnaugh-ova tabela sa označenim logičkim proizvodima, (d) Karnaugh-ova tabela za dopunjenu mrežu bez hazarda.

Ako se zanemare sva kašnjenja, na ulaz *ILI* kola dolazi logička jedinica ili iz jednog ili iz drugog *I* kola i logički nivo izlaza se održava konstantno na logičkoj jedinici (slika 1-10b). Ako se uključi u razmatranje kašnjenje invertora, vrednost logičkog proizvoda *AB* padne na nulu pre nego što proizvod $(\Delta\overline{A})C$ postigne vrednost logičke jedinice i kao rezultat, dobija se privremena logička

nula na izlazu. Ovo se smatra hazardom jer može da dovodi do ozbiljnih smetnji u radu narednih stepena.

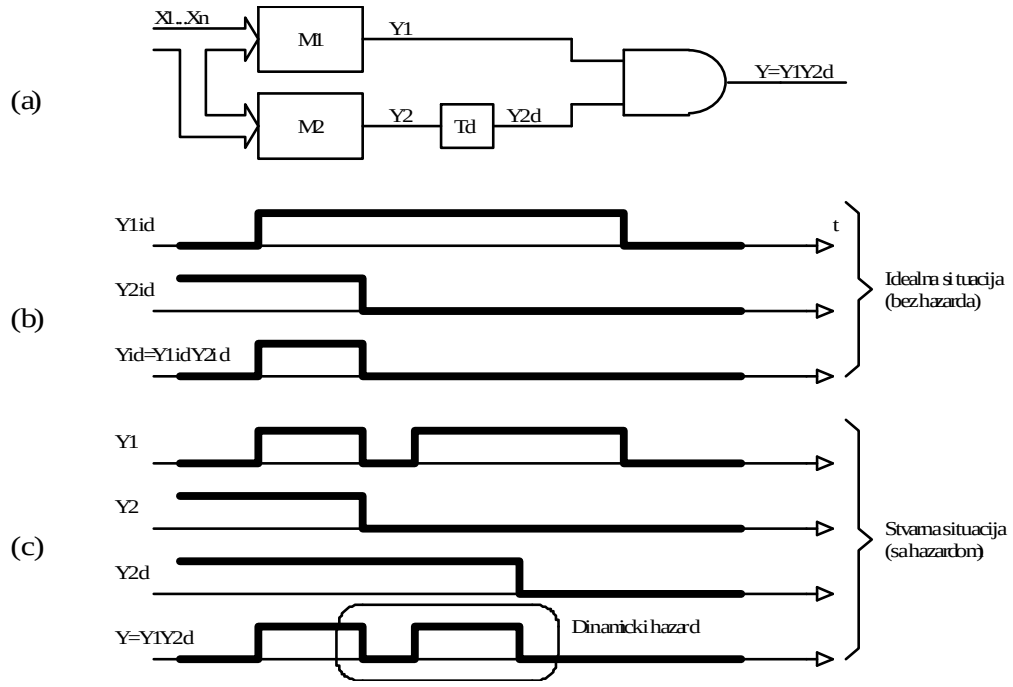
Za otklanjanje hazarda posmatrajmo *Karnaugh*-ovu tabelu prikazanu na slici 1-10c. Statički hazard nastupa zato što se prim implikanti koje sačinjavaju logičku funkciju, dodiruju umesto da se preklapaju. Ako je pri promeni ulaznih promenljivih potrebno preći iz jednog prim implikanta u drugi, može se javiti prelaz ne preko susedne ivice već preko polja tabele na kojoj je vrednost logičke funkcije nula. Za vreme takvog prelaza javlja se hazard.

Jedan način za otklanjanje hazarda je izmena logike datog digitalnog kola. Uvođenjem redundantnog logičkog proizvoda (prikazan isprekidanom linijom na slici 1-10d) koji pokriva mesto prelaza, otklanja se mogućnost pojave hazarda. Redundantni logički proizvod je realizovan pomoću *I* kola koje je prikazano isprekidanom linijom na slici 1-10a. Analizom se može pokazati da *I* kolo koje je dodato funkciji predstavlja takav logički proizvod, koji je u toku minimizacije izostavljen jer (ako se zanemari hazard) je bio suvišan pri realizaciji date logičke funkcije.

Postoje i druge metode za otklanjanje statičkih hazarda. Jedan pristup je da se kašnjenja nastala u jednoj grani kola, kompenzuju namerno postavljenim kašnjenjima u drugim granama kola. Drugi, veoma često primenjen pristup je korišćenje takt signala za sinhronizaciju. Sinhronizacija omogućava kontrolu nad vremenskim trenucima kada je dozvoljeno korišćenje izlaznih vrednosti digitalnog kola. Zabrana treba da traje dovoljno dugo da eventualni hazard iščezne pre pojave novog signala dozvole.

1.3.2. Dinamički hazardi

Dinamički hazardi se javljaju prilikom promene logičkog nivoa izlaza digitalne mreže. Pojava višestrukih promena u logičkom nivou izlaza pre ulaska u novo stanje ukazuje na postojanje dinamičkog hazarda u datom kolu.



Slika 1-11: Nastanak dinamičkog hazarda.

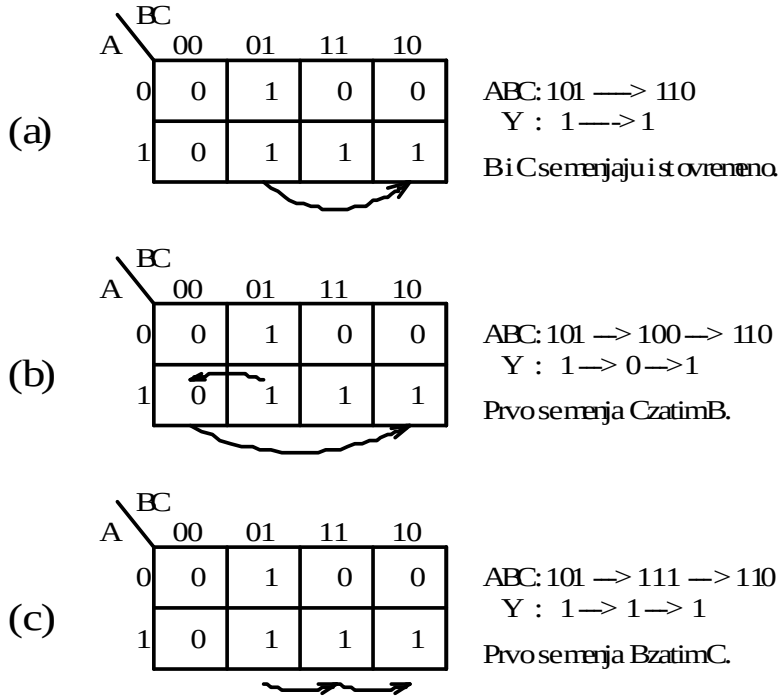
Kao primer posmatrajmo kolo koje je prikazano na slici 1-11a. Kolo sadrži dve podmreže (*M1* i *M2*). Pretpostavimo, da funkcija logičke mreže *M1* stadrži statički hazard koji na izlazu prouzrokuje privremenu logičku jedinicu. Ako pri tome promena na izlazu mreže *M2* kasni za T_{d2}, onda pomoću dijagrama na slici 1-11c se može pokazati, da vrednost izlaza *Y* neće trajno preći u

stanje logičke nule kada vrednost izlaza *Y2* opadne na nulu (što bi bilo u idealnom slučaju), nego će se javiti jedan niz prelaza *1-0-1-0* na izlazu.

Dinamički hazardi mogu da prouzrokuju ozbiljne smetnje u radu narednih stepena kod digitalnih uređaja. Pojava dinamičkog hazarda je u značajnoj meri posledica statičkog hazarda u nekoj podmreži. Ako se otklone statički hazardi i dinamički hazard će nestati.

1.3.3. Funkcionalni hazardi

Funkcionalni hazard je jedna druga nepoželjna pojava u digitalnim mrežama. Javlja se kada se vrednosti dve ili više ulaznih promenljivih menjaju približno istovremeno. Neka se vrednosti promenljivih *A, B* i *C* logičke funkcije kojoj odgovara *Karnaugh*-ova tabela na slici 1-12a menjaju sa *101* na *110*. U principu možemo pretpostaviti da se vrednosti promenljivih *B* i *C* menjaju istovremeno i zato će se na izlazu konstatno javljati logička jedinica. Međutim, u realnosti ne mogu se očekivati tačno istovremene promene.



Slika 1-12: Nastanak funkcionalnog hazarda.

Na slici 1-12b se analizira slučaj kada se prvo vrednost promenljive *C* opadne na logičku nulu, zatim posle kratkog kašnjenja vrednost promenljive *B* postaje logička jedinica. Ovaj prelaz, pri kojoj se pojavljuje kratkotrajna lažna nula, označena je strelicama u *Karnaugh*-ovoj tabeli.

Kada se vrednosti ulaznih promenljivih menjaju u suprotnom redosledu (prvo se menja *B*, a zatim *C*) tada se, na osnovu slike 1-12c, može zaključiti da neće doći do pojave hazarda.

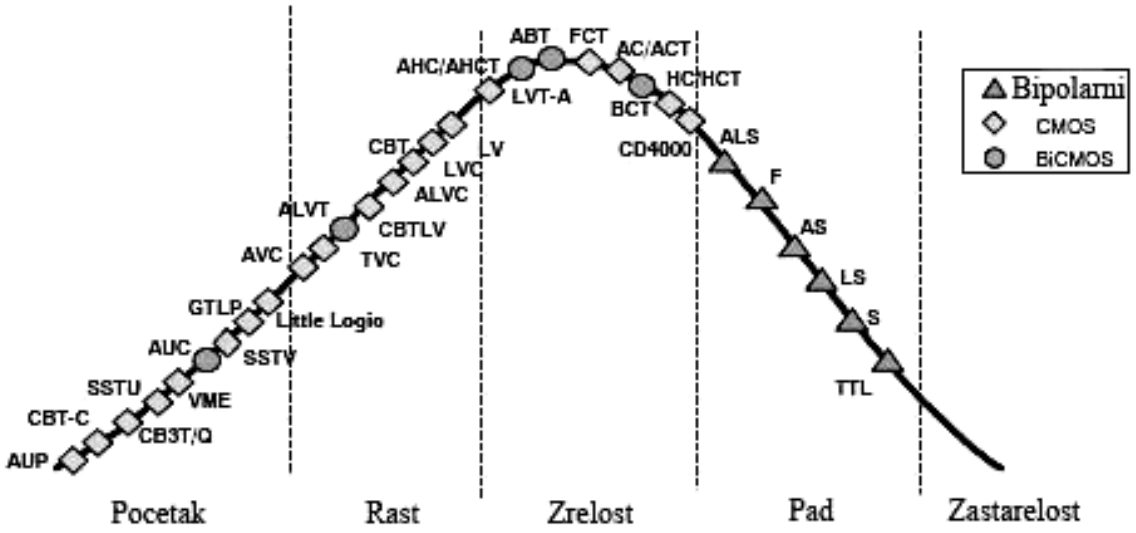
Pristup u otklanjanju funkcionalnih hazarda se može sastojati od uvođenja planiranih kašnjenja ali za sistematsko rešenje ovog problema je neophodna sihronizacija ulaza sa taktom.

1.4. *Tehnologije izrade integrisanih kola*

Digitalna kola se mogu realizovati u različitim tehnologijama. Pri tome optimizacija pojedinih parametara se vrši na manji ili veći uštrb drugih karakteristika. Npr. smanjenem kašnjenja moguće je povećati maksimalnu frekvenciju rada uređaja ali to povlači za sobom i veće gubitke. Ili, smanjenjem napona napajanja smanjuju se gubici ali se pogoršava margina smetnji itd.

1.4.1. Popularnost i životni ciklus familija integrisanih kola

U početku bipolarni tranzistori su korišćeni kao prekidački elementi u digitalnim kolima. Kasnije su se pojavili mosfet prekidači, kojima su postignute bolje prenosne karakteristike pri čemu je i potrošnja smanjena. Digitalna kola izrađena u mosfet tehnologiji su danas više zastupljena ali nisu nestalala ni bipolarna rešenja. Na slici 1-13 se može videti dijagram koji prikazuje popularnost onih integrisanih kola koje danas proizvodi *Texas Instruments*. Trouglovi označavaju one familije koje su izrađene u bipolarnoj tehnologiji, pravougaonici ukazuju na *CMOS* familije dok krugovi označavaju familije sa kombinovanim tehnologijama.



Slika 1-13: Dijagram popularnosti Texas Instruments-ovih digitalnih kola.

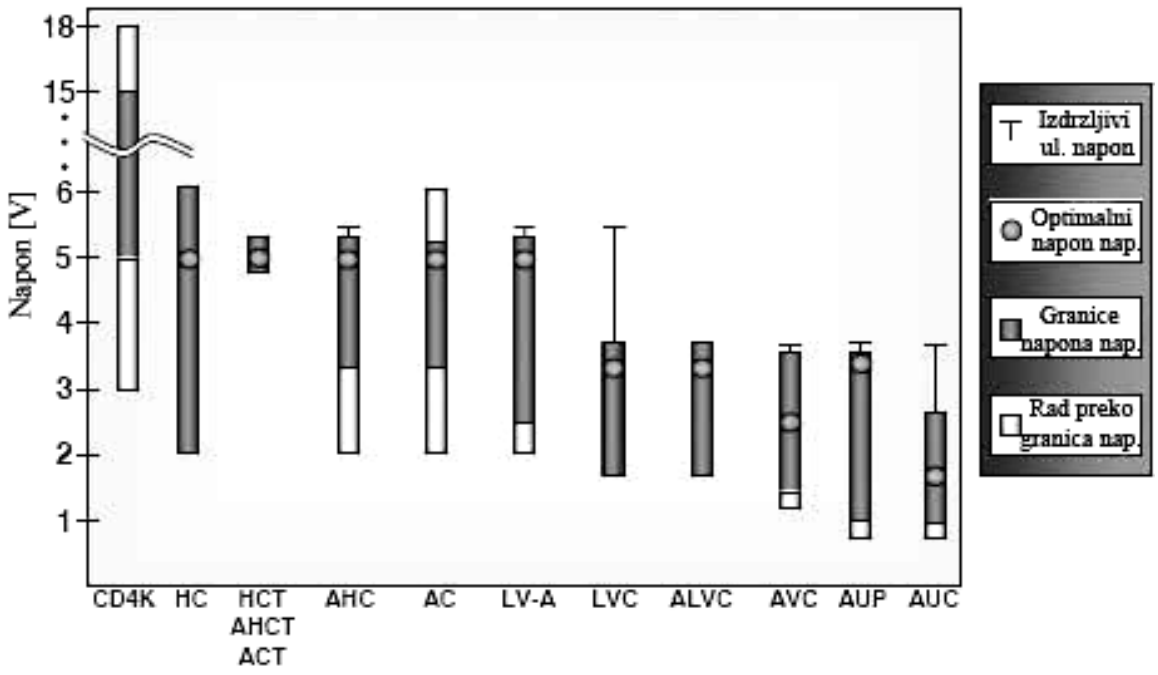
Na desnoj strani dijagrama su prikazane one familije integrisanih kola (*TTL, S, LS,* ...) koje su se počele proizvoditi šezdesetih i sedamdesetih godina *XX*. veka i njihova popolarnost je već opala do danas. Na vrhu dijagrama se nalaze ona kola koja su razvijena osamdesetih godina i danas su najzastupljeniji proizvodi u industriji.

Na levoj strani su prikazane familije integrisanih kola u povoju koje su izrađene novim i obećavajućim tehnologijama. Većina prikazanih familija integrisanih kola se izrađuju u *CMOS* tehnologiji zbog manje potrošnje i bolje prenosne karakteristike. Danas se bipolarne tehnologije najviše primenjuju u onim oblastima gde su potrebne velike struje za pobudu.

1.4.2. Podela prema naponu napajanja

Za novo razvijena kola je karakteristično da se smanjuju vrednosti napona napajanja i umesto jedne konstantne vrednosti mogu da se primenjuju u širem intervalu napona. Podela familija integrisanih kola prema naponu napajanja je prikazana na slici 1-14.

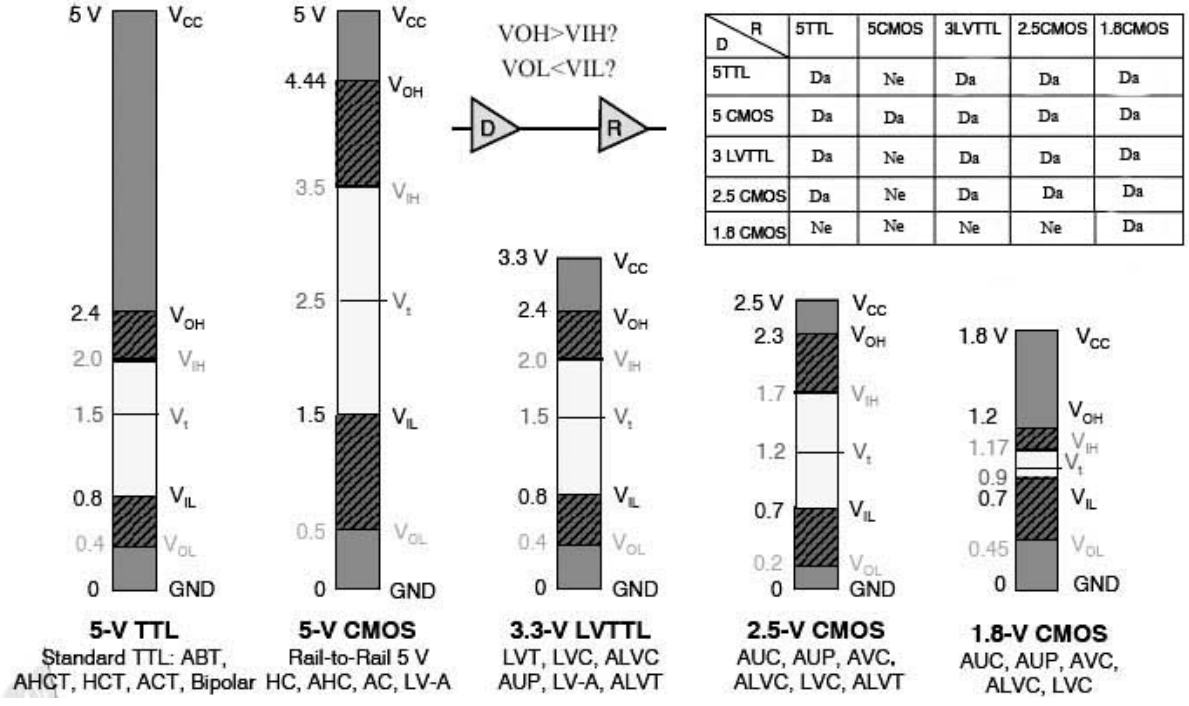
Krugovi na dijagramu označavaju one vrednosti napona napajanja koje obezbeđuju optimalno funkcionisanje integrisanih kola iz date familje. Sivi segment predstavlja onaj opseg napona napajanja za koje su parametri integrisanog kola unutar deklarisanih vrednosti. Unutar svetlog segmenta integrisano kolo je još uvek funkcionalno ali su moguća odstupanja od deklarisanih vrednosti u nekim parametrima. Crna linija označava granice do koje integrisano kolo podnosi sve ulazne i izlazne prenapone bez oštećenja.



Slika 1-14: Podela familija logičkih kola prema naponu napajanja.

1.4.3. Kompatibilnost ligičkih nivoa

Slika 1-15 prikazuje karakteristične logičke nivoe za razne familije logičkih kola. Kod ranije razvijenih logičkih kola u bipolarnoj tehnologiji i sa naponom napajanja od *5V* logički nivoi su asimetrični u odnosu na *0V* i *5V*. Za nove generacije integrisanih kola, posebno kod *CMOS* kola, je karakteristična simetričnost logičkih nivoa.



D \ R	5TTL	5CMOS	3LVTTL	2.5CMOS	1.8CMOS
5TTL	Da	Ne	Da	Da	Da
5 CMOS	Da	Da	Da	Da	Da
3 LVTTL	Da	Ne	Da	Da	Da
2.5 CMOS	Da	Ne	Da	Da	Da
1.8 CMOS	Ne	Ne	Ne	Ne	Da

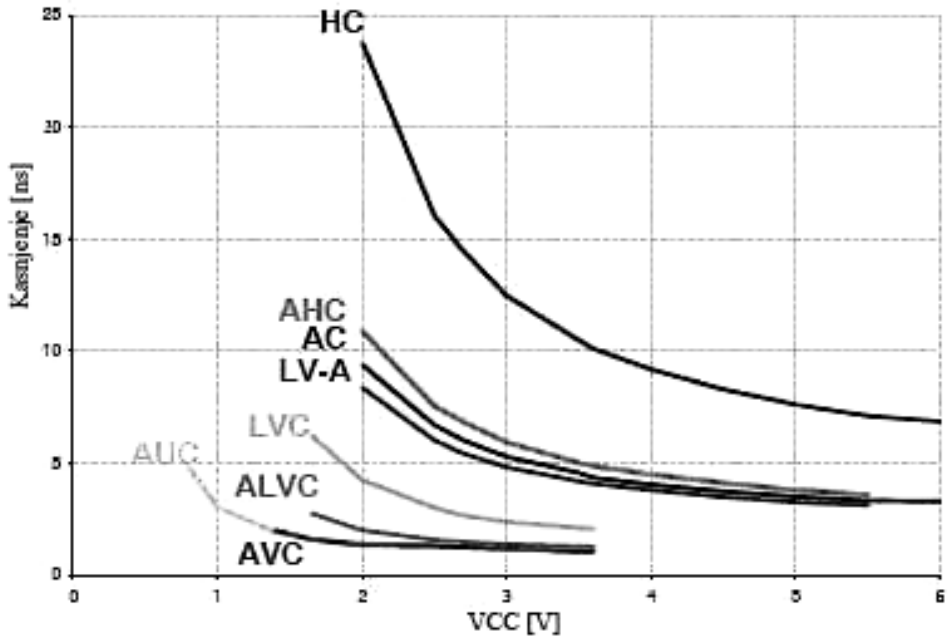
Slika 1-15: Logički nivoi pojedinih familija integrisanih kola i njihova međusobna kompatibilnost.

Tabela u gornjem desnom uglu slike prikazuje kompatibilnost između pojedinih familija logičkih kola. Logički nivoi integrisanih kola unutar jedne familije su uvek kompatibilni. Povezivanje integrisanih kola iz različitih familija je moguće samo ako su ispunjeni uslovi $V_{OL}<V_{IL}$ i $V_{OH}>V_{IH}$. Ponekad se desi da korišćena familija integrisanih kola ne sadrži potrebnu logičku funkciju. U takvoj situaciji se potrebno logičko kolo bira iz druge familije koja je kompatibilna sa prvom. Za postizanje kompatibilnosti mogu se koristiti specijalna kola za sprezanje.

U tabeli su i takve familije označene kao kompatibilne kod kojih je izlazni visoki logički nivo (V_{OH}) pobudnog kola (D) višlji od napona napajanja (V_{CC}) prijemnog kola (R). Takva veza je dozvoljena samo u slučaju ako proizvođač za datu familiju logičkih kola dozvoljava napon na ulazu veći od napona napajanja.

1.4.4. Zavisnost kašnjenja od napona napajanja

Pravac razvoja je decenijama bio sniženje napona napajanja kako bi se smanjili gubici i kašnjenja. Međutim, potrebno je naglasiti da kola koja su projektovana za širi opseg napona napajanja imaju manja kašnjenja kada su priključena na veće napone napajanja (slika 1-16). Na dijagramu su nacrtane krive kašnjenja pojedinih familija za deklarisane opsege napona napajanja prema kataloškim podacima.

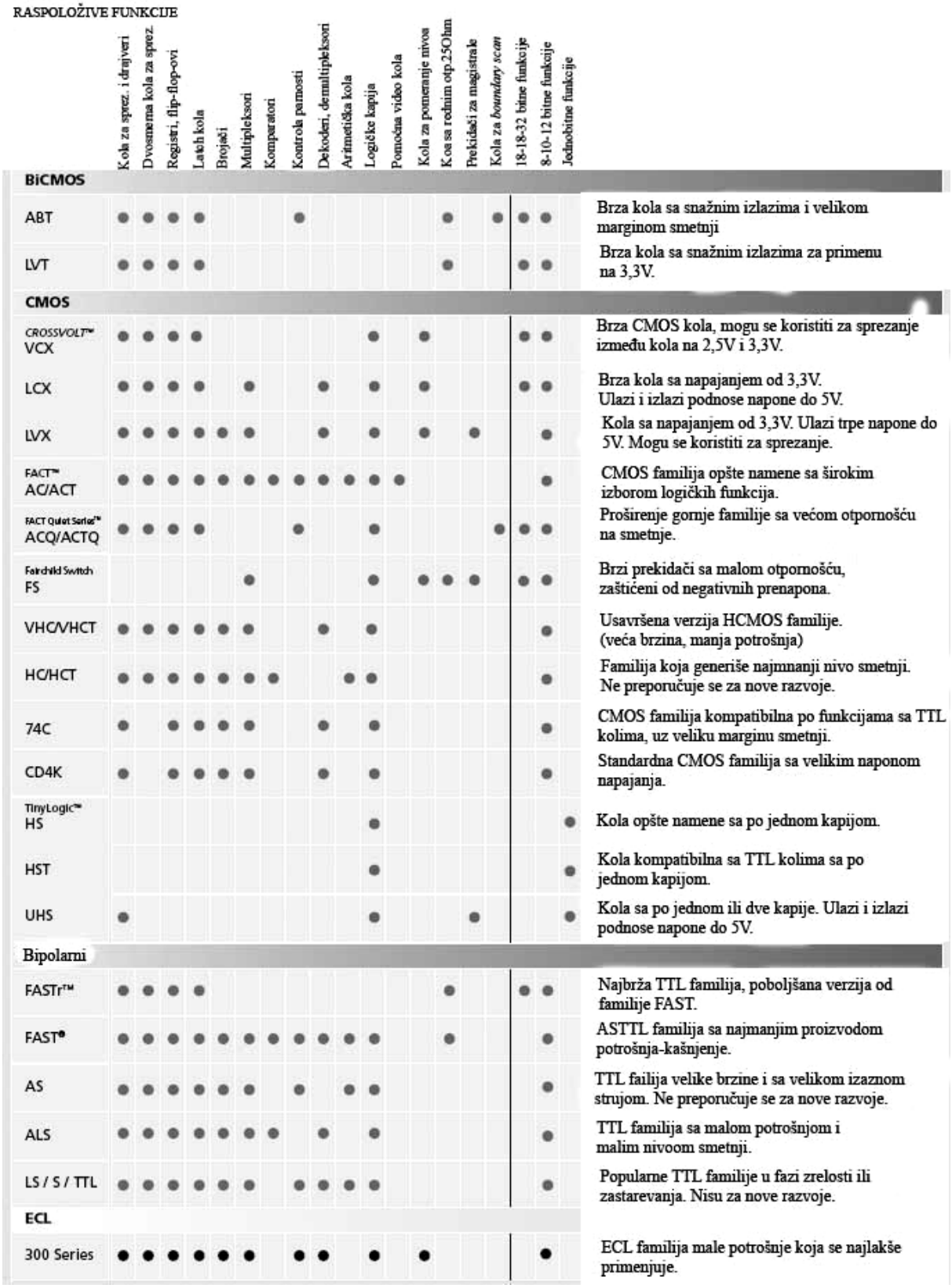


Slika 1-16: Zavisnost kašnjenja od primenjenog napona napajanja za razne familije logičkih kola.

1.4.5. Izbor logičkih funkcija po raznim familijama logičkih kola

Slika 1-17 prikazuje primer tabele koja daje izbor logičkih funkcija iz pojedinih familija logičkih kola (za proizvođač *Fairchild*). Detaljniji podaci o konkretnim funkcijama se mogu dobiti iz liste proizvoda konkretnog proizvođača. Na desnoj strani tabele su navedene oblasti u kojima se preporučuje primena date familije integrisanih kola. Najveći izbor logičkih funkcija iz bipolarne tehnologije imaju familije sa oznakama *TTL*, *S*, *LS* i *F*. Od *CMOS* familija, serije AC i ACT su najzrelije i sadrže najviše ponuđenih rešenja za razne logičke funkcije.

U prikazanom izboru proizvoda su spomenuti samo SSI i MSI kola. Navedene tehnologije se primenjuju i kod LSI i VLSI integrisanih kola koja se softverski ili hardverski programiraju.



Slika 1-17: Izbor logičkih funkcija realizovanih u raznim familijama logičkih kola.

II. Digitalno projektovanje primenom SSI i MSI funkcionalnih jedinica

2. Kombinacione mreže

Kombinacione mreže su digitalna kola kod kojih se vrednost izlaza određuje na osnovu trenutne vrednosti ulaza. U realnosti je potrebno neko određeno vreme (zbog kašnjenja, poglavlje 1.3) za formiranje važećeg izlaznog logičkog nivoa posle svake promene vrednosti ulaza ali nakon toga vrednost izlaza je nezavisna od prethodnih vrednosti ulaza. Ovo je naglašeno zbog toga jer ponašanje sekvencijalnih mreža (3. glava) odstupa od navedenog.

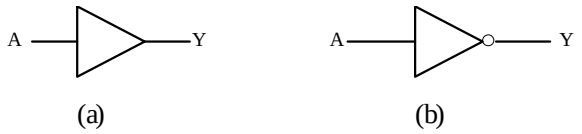
Vrste kombinacionih mreža koje se obrađuju u ovoj glavi se proizvode od šezdesetih godina u obliku SSI i MSI funkcionalnih jedinica. Njihova primena se ne ograničava samo na izgradnju digitalnih uređaja upotrebom SSI i MSI integrisanih kola. Ova rešenja se takođe nalaze i u strukturi mikrokontrolera, mikroprocesora i *PLD*-ova. Navedene funkcije se pojavljuju i u jezicima za opis hardvera koji se koriste pri programiranju *PLD*-ova (III deo).

2.1. *Kola za sprezanje*

Povezivanje različitih funkcionalnih jedinica se često vrši preko kola za sprezanje (prilagođenje). Ova kola mogu da menjaju logičke nivoe, prilagođavaju impedanse i propuštaju signale u direktnom ili u invertovanom obliku. Kod kola za sprezanja akcenat nije na logičkim funkcijama, nego na kolima za pojačavanje koja se koriste u prekidačkom režimu. Obično se veći broj kola za sprezanje (npr. osam komada) ugrađuje u jedno *DIL* ili *SO* kućište i oni pripadaju kategoriji *SSI* kola.

2.1.1. Neinvertujuća i invertujuća kola za sprezanje

Neinvertujuća kola za sprezanje (bafer, engl. *buffer*) i invertujuća kola za sprezanje (invertor, engl. *inverter*) se prave sa velikom ulaznom i malom izlaznom otpornošću. Ovaj izbor impedansi omogućuje vezivanje velikog broja ulaza narednih stepena na izlaz nekog prethodnog stepena. Odgovarajuće šematske oznake su prikazane na slici 2-1.

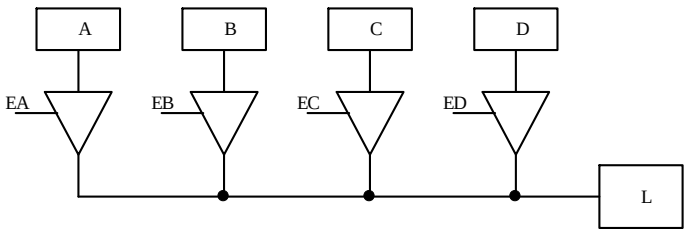


Slika 2-1: Šematske oznake kola za sprezanje : a) bafer, b) invertor.

U jednostavnijim slučajevima kola za sprezanje zahtevaju samo jedan napon napajanja. Ako je potrebno povezati dve logičke jedinice sa različitim naponima napajanja, ulazni i izlazni deo kola za sprezanje se povezuje na posebne izvore napajanja.

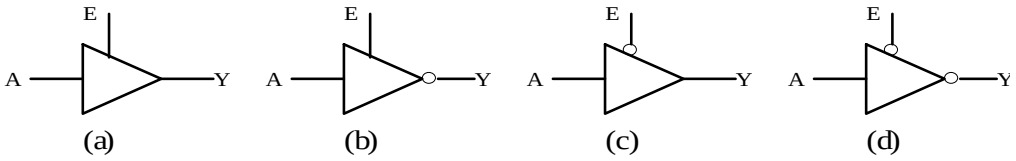
2.1.2. Kola za sprezanje sa tri stanja

Kada se iz više izvora (*A, B, C, D*) prenosi signal prema jedinstvenom korisniku (L) preko jednog zajedničkog provodnika na način kako je prikazan na slici 2-2, neophodno je koristiti kola za sprezanje sa tri stanja. Samo jedno kolo za sprezanje sa tri stanja je aktivno u nekom trenutku, dok se ostali drže u trećem stanju (stanje visoke impedanse). Aktivacija se vrši pomoću signala dozvole (engl. *enable*) E_A, E_B, E_C, E_D.



Slika 2-2: Prenos signala preko zajedničkog provodnika korišćenjem kola za sprezanje sa tri stanja.

Četiri idejna rešenja kola za sprezanje sa tri stanja su u ponudi proizvođača (slika 2-3). Kola za sprezanje mogu da propuste signal bez invertovanja (slike a i c) ili sa invertovanjem (slike b i d) dok signal dozvole može da bude aktivan na logičkoj jedinici (slike a i b) ili na logičkoj nuli (slike c i d). Obično se i kod kola za sprezanje sa tri stanja veći broj komada smešta u zajedničko kućište.

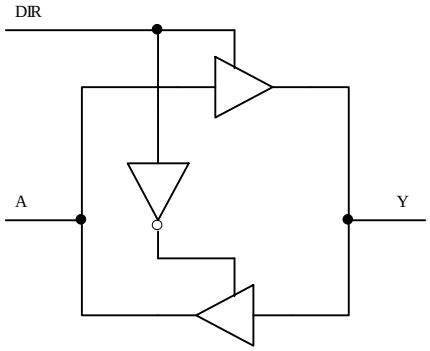


Slika 2-3: Vrste kola za sprezanje sa tri stanja.

2.1.3. Dvosmerna kola za sprezanje

Kada je potreban dvosmeran prenos signala između različitih funkcionalnih jedinica, upotrebljavaju se dvosmerna kola za sprezanje. Unutrašnja struktura dvosmernog kola za sprezanje koji je sposoban za dvosmerni prenos signala je prikazan na slici 2-4. Signal *DIR* određuje smer prenosa signala i zahvaljujući internoj negaciji uvek je samo jedno kolo za sprezanje aktivno. Kolo za sprezanje koje je isključeno, ne utiče na logički nivo izlaza jer je u stanju visoke impedanse.

Za dvosmerna kola za sprezanje takođe važi pravilo da u jedno kućište se smešta više primeraka. Proizvode se i takvi tipovi čija je namena povezivanje mreža sa različitim naponima napajanja i različitim logičkim nivoima.



Slika 2-4: Unutrašnja struktura dvosmernog kola za sprezanje.

2.2. *Dekoder*

Dekoderi su kombinacione mreže sa više ulaza i izlaza, gde svaka dozvoljena kombinacija ulaznih promenljivih aktivira poseban izlaz.

2.2.1. Potpuni dekoder

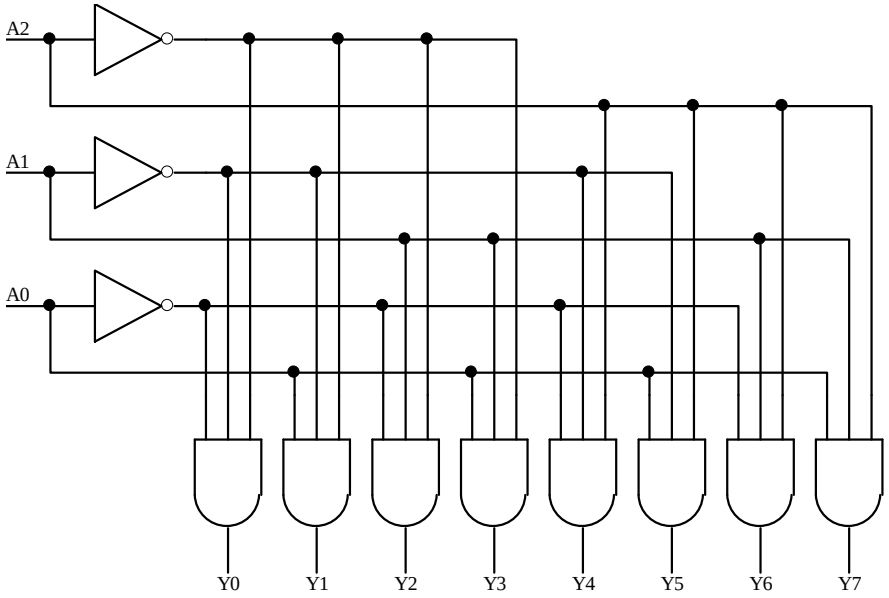
Ulazi potpunog dekodera su binarno kodirani brojevi. Ako je broj ulaznih linija potpunog dekodera jednak *n*, onda je broj mogućih varijacija logičkih nivoa 2^n i zato je broj izlaznih linija jednak 2^n. Za svaku ulaznu kombinaciju uvek je samo jedna izlazna linija aktivna. Kombinaciona tabela potpunog dekodera sa tri ulaza i osam izlaza (3/8) je prikazana u tabeli 2-1.

A2	A1	A0	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

$$Y0 = \overline{A2} \cdot \overline{A1} \cdot \overline{A0}$$
$$Y1 = \overline{A2} \cdot \overline{A1} \cdot A0$$
$$Y2 = \overline{A2} \cdot A1 \cdot \overline{A0}$$
$$Y3 = \overline{A2} \cdot A1 \cdot A0$$
$$Y4 = A2 \cdot \overline{A1} \cdot \overline{A0}$$
$$Y5 = A2 \cdot \overline{A1} \cdot A0$$
$$Y6 = A2 \cdot A1 \cdot \overline{A0}$$
$$Y7 = A2 \cdot A1 \cdot A0$$

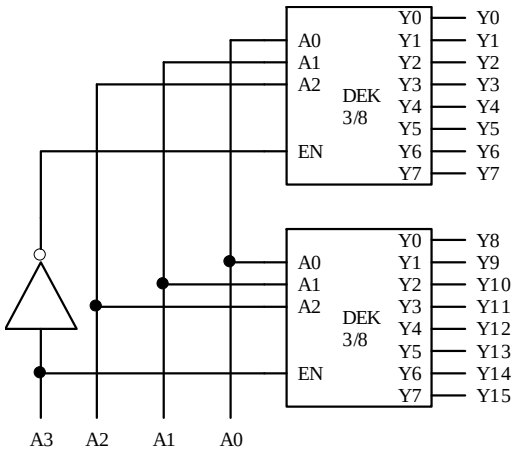
Tabela 2-1: Kombinaciona tabela i logičke funkcije potpunog dekodera 3/8.

Za realizaciju dekodera se koriste logička *I* kola sa potrebnim brojem ulaza jer se svaka izlazna logička funkcija *Y0...Y7* sastoji od jednog logičkog proizvoda (slika 2-5).



Slika 2-5: Realizacija potpunog dekodera pomoću I kola.

Integrisana kola namenjena za funkciju dekodiranja obično sadrže i ulaz za dozvolu (*EN – enable*). Ovo omogućava povećanje kapaciteta dekodera: povezivanjem dva dekodera *3/8* moguće je dobiti dekoder *4/16* (slika 2-6), itd.



Slika 2-6: Realizacija dekodera 4/16 pomoću dva dekodera 3/8.

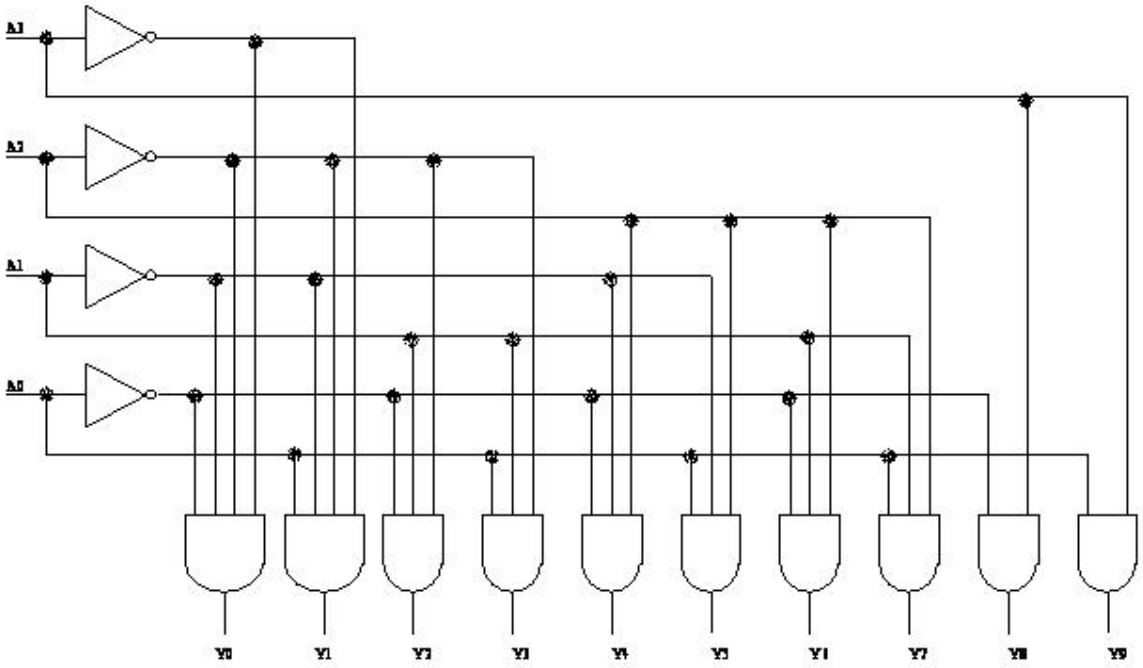
2.2.2. Nepotpuni dekoder

Potpuni dekoder se može uprostiti ako se pouzdano zna da se nikad neće pojaviti neke ulazne kombinacije. Način uprošćavanja se sastoji od izostavljanja onih *I* kola koja realizuju takve logičke proizvode čije vrednosti nikada ne mogu postati jedinice. Nepotpuni dekoder dobijen na ovaj način će ispravno funkcionisati ali moguće je i dalje uprošćavanje.

Tipičan primer nepotpunog dekodera je *BCD* dekoder, koji služi za dekodiranje binarno kodiranih decimalnih cifara. Kombinaciona tabela *BCD* dekodera se dobija iz kombinacione tabele dekodera *4/16* izostavljanjem zadnjih šest linija jer kombinacije brojeva od *10* do *15* se ne mogu pojaviti (tabela 2-2). Minimizacijom logičkih funkcija *Y0...Y9* definisanih na ovaj način dolazi se do logičke mreže predstavljenoj na slici 2-7.

A3	A2	A1	A0	Y9	Y8	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	0	0	0	0

Tabela 2-2: Kombinaciona tabela BCD dekodera.



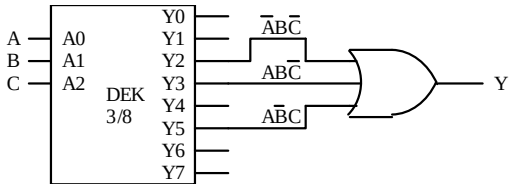
Slika 2-7: Logička mreža koja realizuje minimizirani BCD dekoder.

2.2.3. Realizacija logičkih funkcija pomoću dekodera

Dekoderi realizuju logičke proizvode svojih ulaznih promenljivih i zato je moguće dodavanjem *ILI* kola na izlaz dekodera realizovati bilo koju logičku funkciju koja je data u obliku zbira logičkih proizvoda. Npr. funkcija:

$$Y = \overline{A}B\overline{C} + AB\overline{C} + A\overline{B}C$$

je realizovana na način prikazan na slici 2-8.



Slika 2-8: Realizacija logičke funkcije korišćenjem dekodera.

2.3. Koder

Digitalna obrada se vrši nad kodiranim signalima. Npr. kada se pritisne jedan taster na tastaturi računara, nastaje niz logičkih nula i jedinica (kod). Suština kodiranja je dodeliti različit kod svakoj ulaznoj vrednosti.

2.3.1. Potpuni koder

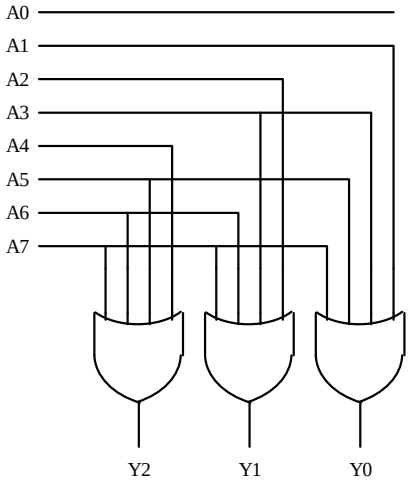
Izlazni kod dužine *n* bita potpunog kodera se formira na osnovu 2^n ulaza. U tabeli 2-3 je data kombinaciona tabela dekodera sa osam ulaza i tri izlaza (8/3). U tabeli su prikazani samo oni redovi od svih mogućih $2^8=256$ u kojima samo jedna ulazna promenljiva ima vrednost logičke jedinice. Na slici su takođe dati izrazi za logičke funkcije dobijeni logičkom minimizacijom. Funkcije dekodera se u ovom slučaju dobijaju u obliku čistog zbira umesto sume proizvoda.

A7	A6	A5	A4	A3	A2	A1	A0	Y2	Y1	Y0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Y0=A1+A3+A5+A7
Y1=A2+A3+A6+A7
Y2=A4+A5+A6+A7

Tabela 2-3: Kombinaciona tabela kodera 8/3 i odgovarajuće izlazne logičke funkcije.

Logička šema koja se dobija na osnovu prethodnih jednačina za koder 8/3 je prikazana na slici 2-9. Logička vrednost izlaza je netačna ako na ulazima logičke mreže ima više od jedne jedinice. Zbog ovih kombinacija, koder se može proširiti sa jednim logičkim kolom koje upozorava na nedozvoljene kombinacije na ulazima.



Slika 2-9: Logička mreža koja realizuje koder 8/3.

2.3.2. Nepotpuni koder

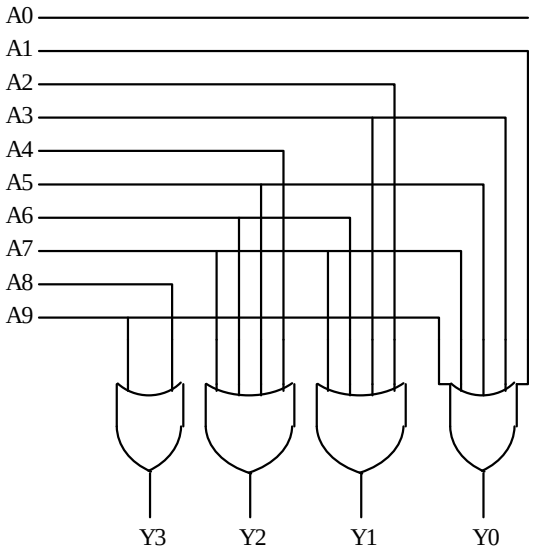
Koder je nepotpun kada je broj mogućih stanja na ulaznim linijama koja se kodiraju pomoću *n* izlaznih linija manji od 2^n. Tipičan primer za to je kodiranje cifara decimalnog brojnog sistema pomoću četiri izlazne linije (*bit*-a). Naziv ovog kodera je *DC-BCD* koder. Odgovarajuće kombinacione tabele su prikazane u tabeli 2-4, gde su date i minimizovane logičke funkcije.

Ai	Y3	Y2	Y1	Y0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Y0=A1+A3+A5+A7+A9
Y1=A2+A3+A6+A7
Y2=A4+A5+A6+A7
Y3=A8+A9

Tabela 2-4: Kombinaciona tabela DC/BCD kodera i odgovarajuće logičke funkcije.

Logička mreža koja realizuje *DC-BCD* koder je prikazana na slici 2-10. Neophodno je naglasiti da realizacija funkcije zaštite od istovremene pojave više logičkih jedinica na ulazu kodera je zadatak korisnika.



Slika 2-10: Logička mreža koja realizuje DC-BCD koder.

2.3.3. Prioritetni koder

Rečeno je da oni koderi koji su prikazani u prethodnim tačkama generišu pogrešan kod odnosno neupotrebljiv kod na izlazima ako su više od jedne logičke jedinice prisutne na ulaznim linijama. Tipičan primer za ovo je obrada prekida kod mikrokontrolera. Više signala mogu istovremeno da imaju vrednost logičke jedinice jer je priroda logičkih signala asinhrona (potiču iz spoljašnjih kola). U takvim situacijama mikrokontroler odlučuje koji zahtev za prekid će se prvi opslužiti. Ovakav zadatak se unutar mikrokontrolera rešava pomoću prioritetnog kodera.

Ulazi prioritetnog kodera su rangirani po važnostima (prioritetima). Ako su prisutne više jedinice na ulazima kodera onda se formira kod onog ulaza čiji je prioritet najveći.

Na slici 2-15 je data kombinaciona tabela prioritetnog kodera 8/3. Sa *x* smo obeležili one vrednosti ulaza koje ne utiču na funkcionisanje prioritetnog kodera. Potrebno je napomenuti da u tabeli nije prikazan slučaj kada su svi ulazi na logičkoj nuli. Ako je neophodno uzeti i taj slučaj u obzir, dodaje se kolo koje je posebno projektovano za tu namenu.

Ulaz	Izlaz (kod)
1xxxxxxx	000
01xxxxxx	001
001xxxxx	010
0001xxxx	011
00001xxx	100
000001xx	101
0000001x	110
00000001	111

Tabela 2-5: Kombinaciona tabela prioritetnog kodera 8/3.

2.4. *Pretvarači koda*

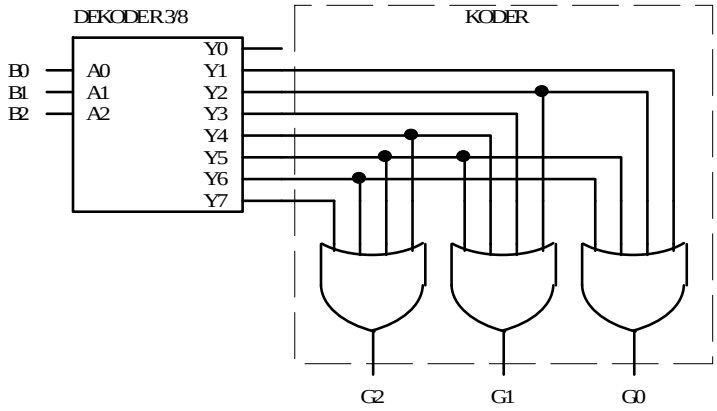
Kombinacione mreže, koje podatke iz jednog kodnog sistema pretvaraju u drugi, nazivamo pretvaračima koda. Teoretski se svaki pretvarač koda može konstruisati kaskadnom vezom jednog dekodera i jednog kodera. Međutim, često postoji veliki prostor u minimizaciji jednog takvog pretvarača koda.

2.4.1. Pretvarač prirodnog binarnog koda u *Gray*-ov kod

Pretvaranje trobitnog prirodnog binarnog broja u *Gray*-ov kod se vrši prema tabeli 2-6. Rešenje koje se dobija kaskadnom vezom dekodera i kodera prikazano je na slici 2-11.

Prirodni binarni kod	*Gray*-ov kod
000	000
001	001
010	011
011	010
100	110
101	111
110	101
111	100

Tabela 2-6: Uporedna tabela trobitnog prirodnog binarnog koda i trobitnog Gray-ovog kola.



Slika 2-11: Pretvarač koda realizovan kaskadnom vezom dekodera i kodera.

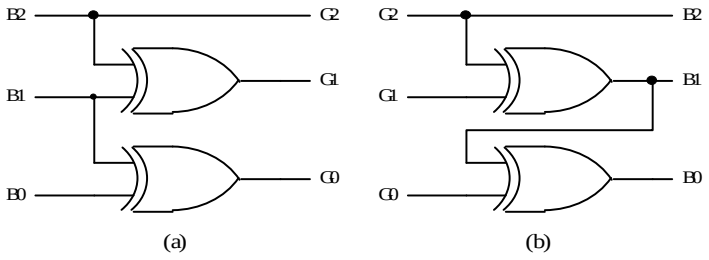
Na osnovu tabele mogu se napisati sledeće jednačine za pojedinačne bitove *Gray*-ovog koda:

$$G_2 = B_2\overline{B_1}\,\overline{B_0} + B_2\overline{B_1}B_0 + B_2B_1\overline{B_0} + B_2B_1B_0$$
$$G_1 = \overline{B_2}B_1\overline{B_0} + \overline{B_2}B_1B_0 + B_2\overline{B_1}\,\overline{B_0} + B_2\overline{B_1}B_0$$
$$G_0 = \overline{B_2}\,\overline{B_1}B_0 + \overline{B_2}B_1\overline{B_0} + B_2\overline{B_1}B_0 + B_2B_1\overline{B_0}$$

Minimizacijom prethodnih logičkih funkcija se dobijaju sledeći, jednostavniji izrazi:

$$G_2 = B_2$$
$$G_1 = B_2 \oplus B_1$$
$$G_0 = B_1 \oplus B_0$$

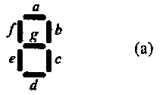
Pojednostavljena šema pretvarača koda je prikazana na slici 2-12a dok minimizovana logička mreža prikazana na slici 2-12b obavlja inverznu operaciju u odnosu na prethodnu mrežu.



Slika 2-17: Minimizovani pretvarači koda: (a) iz binarnog u Gray-ov kod i (b) iz Gray-ovog koda u binarni.

2.4.2. Pretvarač BCD koda u 7-segmentni kod

Najčešće korišćen pretvarač koda je digitalno kolo koje na osnovu binarno kodiranih decimalnih cifara pokreće sedmosegmenti displej. Kao naziv za ovaj pretvarač često se u kataloškim podacima (pogrešno) navodi izraz dekoder. Segmenti LCD ili LED displeja koji služe za prikazivanje jedne decimalne cifre su postavljene u obliku osmice. Oznake segmenata su prikazane na slici 2-13a. Slika 2-13b prikazuje kombinacije segmenata koji treba da se aktiviraju za prikazivanje pojedinih cifara decimalnog brojnog sistema.



(a)


(b)

Slika 2-13: (a) Oznake segmenata sedmosegmentog displeja, (b) kombinacije segmenata za prikazivanje cifara decimalnog sistema.

Tabela 2-7 prikazuje kombinacionu tabelu koja odgovara navedenom pretvaraču koda.

Cifra	D C B A	a b c d e f g
0	0 0 0 0	1 1 1 1 1 1 0
1	0 0 0 1	0 1 1 0 0 0 0
2	0 0 1 0	1 1 0 1 1 0 1
3	0 0 1 1	1 1 1 1 0 0 1
4	0 1 0 0	0 1 1 0 0 1 1
5	0 1 0 1	1 0 1 1 0 1 1
6	0 1 1 0	1 0 1 1 1 1 1
7	0 1 1 1	1 1 1 0 0 0 0
8	1 0 0 0	1 1 1 1 1 1 1
9	1 0 0 1	1 1 1 1 0 1 1

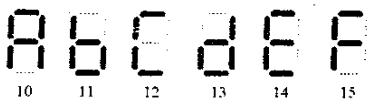
Tabela 2-7: Kombinaciona tabela pretvarača koda iz BCD koda u 7-segmentni kod.

Za svaki segment je potrebno napisati odgovarajuće jednačine i nakon minimizacije se dolazi do sledećih rezultata:

$$a = B + D + AC + \overline{A}\,\overline{C}$$
$$b = \overline{C} + AB + \overline{A}\,\overline{B}$$
$$c = A + \overline{B} + C$$
$$d = D + \overline{A}B + A\overline{B}C + \overline{A}\,\overline{C} + B\overline{C}$$
$$e = \overline{A}B + \overline{A}\,\overline{C}$$
$$f = D + \overline{A}\,\overline{B} + \overline{A}C + \overline{B}C$$
$$g = D + \overline{A}B + B\overline{C} + \overline{B}C$$

Dobijene logičke funkcije se sintetizuju dvostepenim *I-ILI* logičkim mrežama koja se zatim integrišu u jedno *MSI* kolo. Ova kola obično sadrže još i jedan upravljački ulaz, pomoću kojeg se mogu svi segmenti istovremeno isključiti. Korist od upravljačkog ulaza je velika ako je potrebno istovremeno prikazati više cifara.

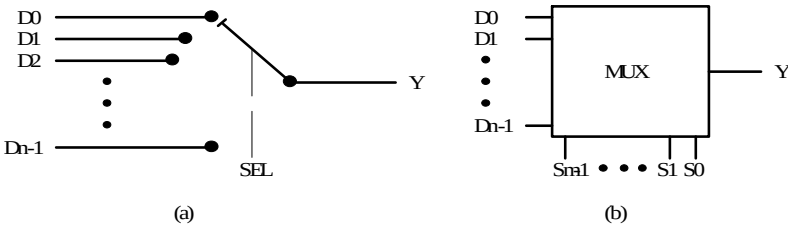
Ako se na ulaz rešenja koje je prikazano gore dovodi nepostojeći kod, prikaz na displeju će biti besmislen. Malim promenama je moguće rešiti da displej prikazuje simbole sa slike 2-14 za kodne reči heksadecimalnog brojnog sistema od 10 do 15.



Slika 2-14: Način prikazivanja cifara heksadecimalnog brojnog sistema veće od devet pomoću sedmosegmentnog displeja.

2.5. *Multipleksor*

Multipleksor se može prikazati kao višepoložajni prekidač (slika 2-15). Prosleđuje signale ($D_0 ... D_{n-1}$) sa svojih ulaza na izlaz. Selekcioni ulazi multipleksora ($S_0 ... S_{m-1}$) određuju ulazni signal koji će se u datom momentu proslediti na izlaz.



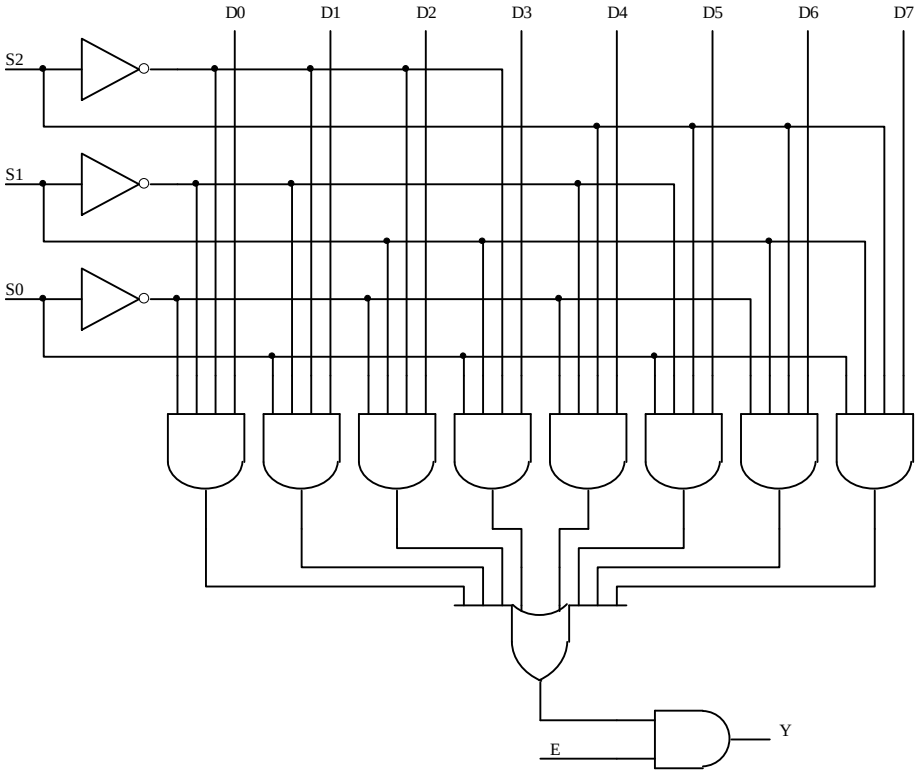
Slika 2-15: (a) Unutrašnja struktura multipleksora i (b) njegova šematska oznaka.

2.5.1. Konstruisanje digitalnih multipleksora

Proizvođači obično u jedno *MSI* kolo integrišu multipleksor sa 2, 4, 8 ili 16 ulaza, pri čemu je broj selekcionih ulaza redom 1, 2, 3 i 4. Izlazna funkcija multipleksora sa osam ulaza za podatke (*D0...D7*) i tri selekciona ulaza (*S0...S2*) je:

$$Y = D_0 \overline{S}_2 \overline{S}_1 \overline{S}_0 + D_1 \overline{S}_2 \overline{S}_1 S_0 + ... + D_7 S_2 S_1 S_0 .$$

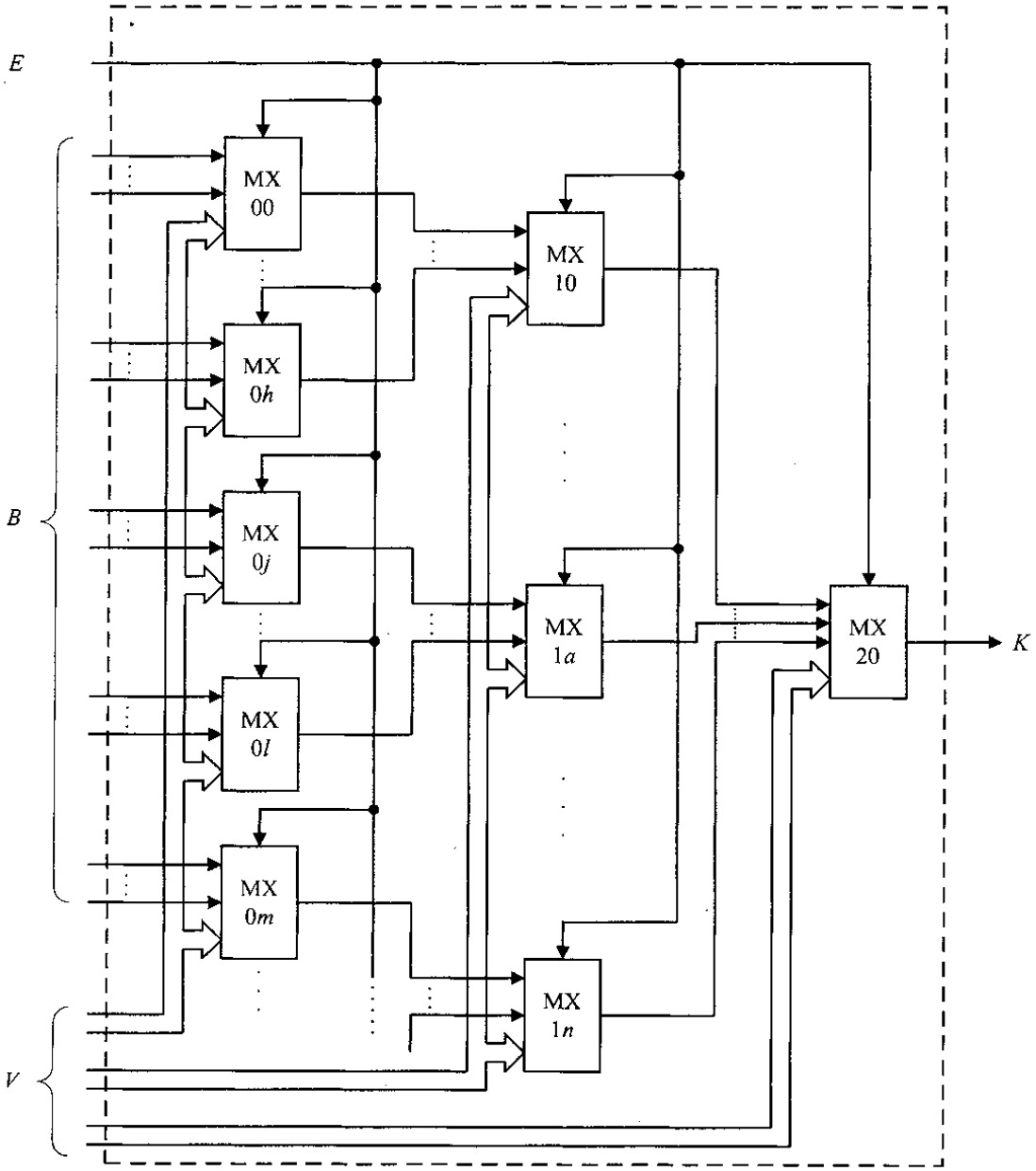
Realizacija kola se vrši korišćenjem osam komada troulaznih *I* kola i jednog osmoulaznog *ILI* kola. Logička mreža prikazana na slici 2-16 realizuje dati multipleksor, sa time da je izveden i ulaz za dozvolu rada (E) pomoću kojeg je moguće stanje izlaza prevesti u logičku nulu nezavisno od ulaza za podatke i selekcionih ulaza. Ulaz za dozvolu se može koristiti i za proširivanje multipleksora. Ovo je značajno kada je potreban multipleksor sa više ulaza za podatke od onoga što je na raspolaganju u integrisanoj formi (obično se proizvode do 16 ulaza).



Slika 2-16: Logička mreža koja realizuje multipleksor sa osam ulaza za podatke.

2.5.2. Proširivanje multipleksora

Način proširivanja multipleksora je prikazan na slici 2-17. Podaci se vode na ulaze odgovarajućeg manjeg multipleksora. Na selekcione ulaze ovih multipleksora se vode isti signali. Izlazni podaci multipleksora iz prvog stepena se vode na sledeći multipleksor koji odlučuje o tome koji podaci se dalje prosleđuju sa prethodnog stepena. Proširivanje se može izvesti u više stepena.



Slika 2-17: Principijelna šema za proširivanje multipleksora.

2.5.3. Realizacija logičkih funkcija pomoću multipleksora

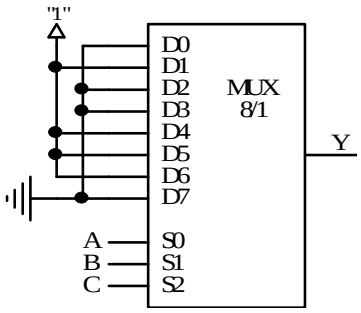
Pored osnovne funkcije (biranje podataka), multipleksori se mogu efikasno primeniti i u realizaciji logičkih funkcija. Uslov za realizaciju je da funkcija bude izražena u obliku sume proizvoda i da proizvodi sadrže sve ulazne promenljive. Za realizaciju sledeće funkcije:

$$Y = \overline{C}\,\overline{B}A + C\overline{B} + CB\overline{A}$$

neophodno je ovaj izraz prevesti u normalnu formu:

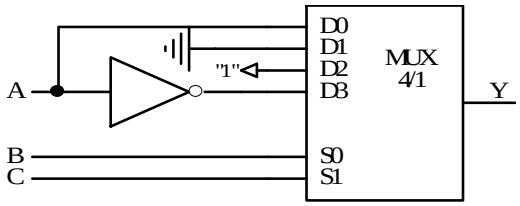
$$Y = \overline{C}\,\overline{B}A + C\overline{B}\,\overline{A} + C\overline{B}A + CB\overline{A}$$

Realizacija podrazumeva povezivanje promenljivih *A*, *B* i *C* na selekcione ulaze i dovođenje odgovarajućih naponskih nivoa na ulaze za podatke. Naponski nivoi se određuju na osnovu normalizovane funkcije. Svakom proizvodu date funkcije odgovara tačno jedan ulaz za podatke koji se povezuje na logičku jedinicu a oni ulazi koji nisu pokriveni logičkim proizvodima u datoj funkciji se vežu na logičku nulu (slika 2-18).



Slika 2-18: Realizacija logičke funkcije od tri promenljive multipleksorom sa osam ulaza za podatke.

Postoji i efikasniji način korišćenja multipleksora u realizaciji logičkih funkcija od onog koji je dat u prethodnom primeru. Tako npr. pomoću multipleksora sa osam ulaza za podatke moguće je realizovati logičke funkcije od četiri promenljive odnosno korišćenjem multipleksora sa četiri ulaza za podatke moguće je realizovati logičke funkcije od tri promenljive (slika 2-19).

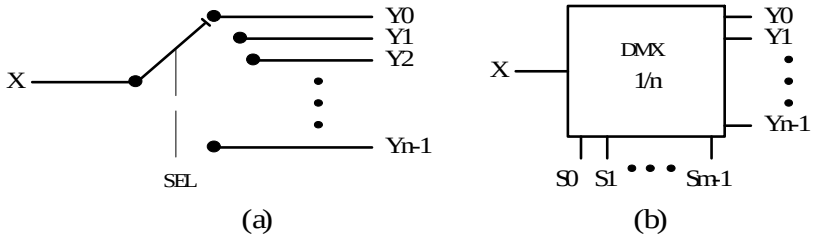


Slika 2-19: Realizacija logičke funkcije od tri promenljive multipleksorom 4/1.

Promenljive *B* i *C* su na uobičajeni način vezani na selekcione ulaze. Zatim na pojedinačne ulaze za podatke se vezuju logičke nule ili jedinice ako je vrednost funkcije nezavisna od promenljive *A* a za date vrednosti promenljivih *B* i *C* ima vrednost nulu odnosno jedinicu. Ako vrednost funkcije zavisi od promenljive *A* onda se na odgovarajući ulaz podatka direktno dovodi promenljiva *A* ili njegova invertovana vrednost, tako da se na izlazu multipleksora dobija ispravna vrednost funkcije.

2.6. *Demultipleksor*

Uloga demultipleksora je suprotna od multipleksora: informacija se sa jedne linije podatka prosleđuje na jedan od više izlaznih linija podataka. Princip funkcionisanja se može prikazati pomoću jednog višepoložajnog prekidača (slika 2-20a). Biranje izlazne linije podatka se vrši odgovarajućim kodom na selekcionim ulazima (slika 2-20b, ulazi $S_0...S_{m-1}$).

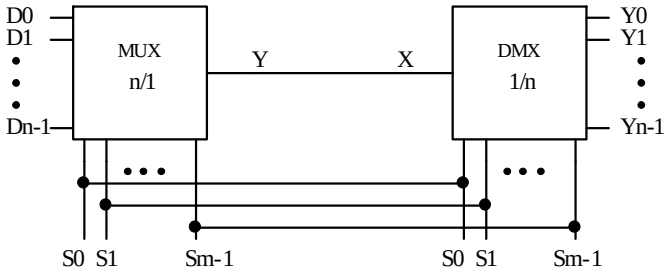


Slika 2-20: (a) Funkcija demultipleksora prikazana višepoložajnim prekidačem i (b) njegova šematska oznaka.

Demultipleksor se može zameniti dekoderima ako se na ulaz za dozvolu kod dekodera dovode ulazni podaci. Ulazi dekodera se koriste kao selekcioni ulazi. Vrednost izlaza koji se adresira selekcionim ulazima je logička jedinica ako je rad kola omogućen signalom dozvole. Ako je vrednost ulaznog signala nula, onda je i vrednost izlaza nula jer je rad kola onemogućen. Na ovaj način, u oba slučaja se dobija ispravna vrednost na izlaznim kanalima. Proizvođači obično nude iste komponente pod nazivima dekoder i demultipleksor.

2.6.1. Prenos više podataka preko zajedničkog kanala

Povezivanjem multipleksora i demultipleksora je moguće preneti digitalne podatke preko smanjenog broja provodnika. Ako je broj ulaznih linija $n=2^m$, moguće je realizovati prenos podataka preko $m+1$ linije. Odgovarajuća logička šema je prikazana na slici 2-21. Selekcioni ulazi su na isti način povezani kod oba kola i zato je redosled ulaznih i izlaznih linija identičan.



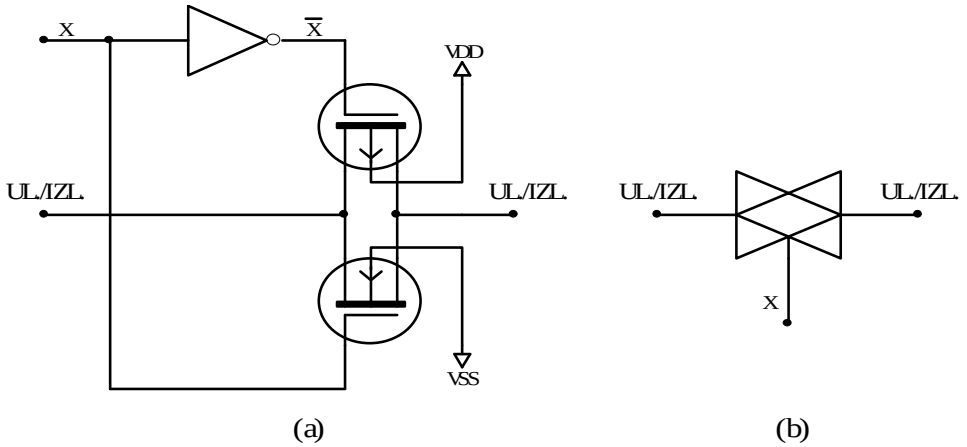
Slika 2-21: Prenos podataka preko smanjenog broja linija korišćenjem multipleksora i demultipleksora.

Nedostatak ove metode je da prenos podataka sa pojedinih ulaznih kanala na pojedine izlazne kanale se ne dešava istovremeno nego jedno za drugim. Ovaj princip se naziva vremenskim multipleksom. Kapacitet zajedničkog kanala se deli među pojedinim podacima i zbog toga dolazi do značajnih smanjenja u brzini prenosa.

2.6.2. Analogni multipleksor/demultipleksor

Digitalni multipleksori i demultipleksori su pogodni samo za prenos logičkih nivoa za razliku od višepoložajnih prekidača prikazanih na slikama 2-15 i 2-20 koji su u stanju da prenose digitalne i analogne signale u oba smera. Multipleksiranje/demultipleksiranje analognih signala je potrebno npr. kod višekanalnih mernih uređaja.

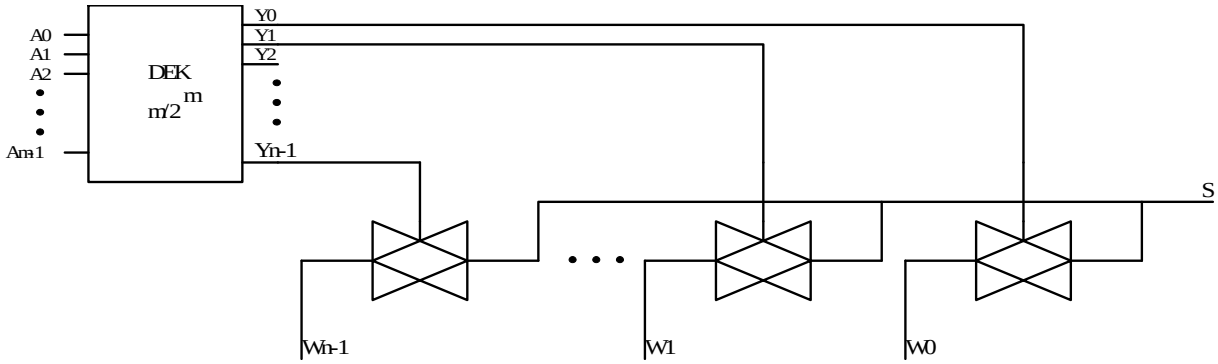
U integrisanoj tehnici je takođe moguće realizovati multipleksore/demultipleksore za analogne signale. Analogni prekidač čini osnovu ovih kola, koji pod uticajem upravljačkog signala otvara ili zatvara poluprovodnički kanal. Analogni prekidač se formira paralelnom vezom N i P kanalnog mosfeta (slika 2-22), ali postoje i druga rešenja. Suština je ta da otpornost kanala u uključenom stanju treba da bude mala a u isključenom stanju velika.



Slika 2-22: (a) Konstrukcija analognog prekidača pomoću N kanalnog i P kanalnog mosfeta i (b) šematska oznaka analognog prekidača.

Analogni multipleksor/demultipleksor (slika 2-23) se dobija primenom potrebnog broja analognih prekidača kojima se upravlja pomoću dekodera. Na jednu stranu prekidača se dovode odgovarajući analogni signali dok se drugi kraj svih prekidača veže u jednu tačku koja predstavlja izlaz. Uloga dekodera je da obezbedi takvo upravljanje analognim prekidačima da na bilo koju kombinaciju upravljačkih signala uvek samo jedan prekidač bude u provodnom stanju. Isto kolo se

može koristiti i kao demultipleksor, jer analogni prekidači provode signale u oba smera kada su uključeni i neprovodni su u oba smera kada su isključeni.



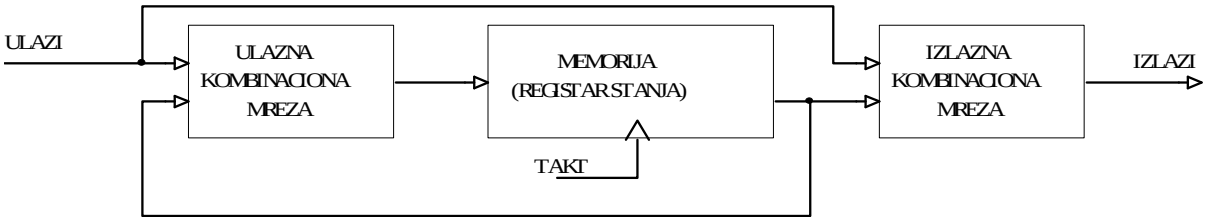
Slika 2-23: Realizacija analognog multipleksora/demultipleksora primenom analognih prekidača i dekodera.

3. Sekvencijalne mreže

Sekvencijalne mreže su digitalna kola koja poseduju osobinu pamćenja (memorisanja). Digitalno kolo pamti određene informacije o stanju samog kola u prošlosti i o upravljačkim signalima dovedenim u proslošti. Izlazni signali sekvencijalne mreže se formiraju na osnovu zapamćene informacije i novih vrednosti logičkih nivoa na ulazima. Sekvencijalne mreže se redovno nazivaju i logičkim automatima jer se često primenjuju u oblasti automatskog upravljanja.

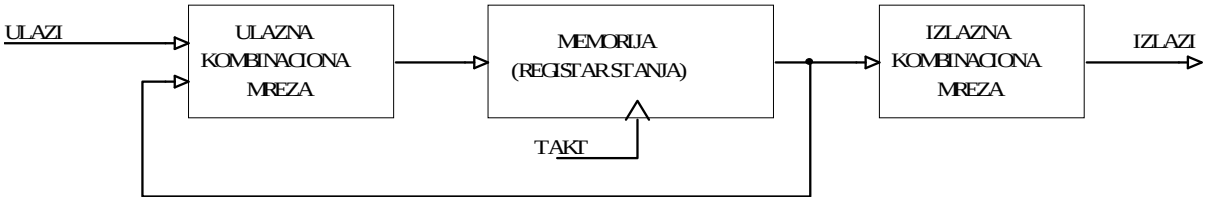
Kapacitet sekvencijalnih mreža za pamćenje je najčešće mali i obično je reč o svega nekoliko bita. Kod *n*-bitnih sekvencijalnih mreža postoje 2^n različitih stanja. Zbog konačnog broja stanja, sekvencijalne mreže se još nazivaju i automatima sa konačnim brojem stanja (engl. *finite-state machine*) .

Pored memorije, sekvencijalne mreže sadrže i kombinacione elemente (mreže). Na osnovu unutrašnje strukture razlikujemo dve vrste logičkih automata. Na slici 3-1 je prikazana blok šema *Mealy*-jevog automata. Kolo, pored memorije, sadrži i dve kombinacione mreže: jednu na ulazu, drugu na izlazu. Ulazna kombinaciona mreža određuje novo stanje automata na osnovu trenutne vrednosti logičkih nivoa na ulazima i na osnovu trenutnog stanja automata. Istovremeno, izlazna kombinaciona mreža definiše izlazne digitalne signale na osnovu trenutne vrednosti signala na ulazima i trenutnog stanja automata.



Slika 3-1: Blok šema sekvencijalne mreže Mealy-jevog tipa .

Na slici 3-2 je prikazan *Moore*-ov automat koji je jednostavnije konstrukcije. Kod ovog tipa se vrednosti izlaza formiraju samo na osnovu trenutnog stanja automata. Pojednostavljeni slučaj *Moore*-ovog automata se dobija kada se izostavlja izlazna kombinaciona mreža a izlazi memorije (svi ili neki) su ujedno i izlazi automata.



Slika 3-2: Blok šema sekvencijalne mreže Moore-ovog tipa.

Kod projektovanja sekvencijalnih mreža ni *Mealy*-jev ni *Moore*-ov tip automata ne uživa apsolutnu prednost. Prvi tip je obično brži a drugi je jednostavniji.

Kod većine sekvencijalnih mreža trenutak promene stanja se sinhronizuje takt signalom (engl.: *clock*). Takt signal se obično sastoji od periodične povorke pravougaonih impulsa, mada periodičnost nije neophodan uslov. Takt signal se koristi samo za sinhronizaciju rada memorije, dok kombinacione mreže funkcionišu bez sinhronizacionih signala. Moguće je konstruisati i asinhrone sekvencijalne mreže. Preglednost funkcionisanja takvih mreža je značajno smanjena u odnosu na sinhrone mreže i zato njihovo projektovanje zahteva mnogo više pažnje.

U ovoj glavi prvo će se razmotriti memorijski elementi nakon čega slede načini opisivanja i konstruisanja sekvencijalnih mreža. Na kraju ove glave se daje pregled često korišćenih sekvencijalnih mreža dostupnih u obliku integrisanih kola *SSI* odnosno *MSI* klase.

Pojmovi i metode koje se ovde uvode (osim klasičnog načina projektovanja primenom *SSI* i *MSI* integrisanih kola) se primenjuju i kod projektovanja primenom programabilnih logičkih kola.

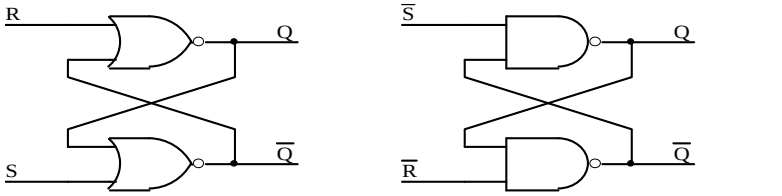
3.1. *Elementarne memorije*

Memoriju sekvencijalnih mreža sačinjavaju *latch*-evi i *flip-flop*-ovi. Ova kola su u stanju da pamte informaciju veličine jednog bita i sami po sebi predstavljaju sekvencijalnu mrežu. Informacija se pamti korišćenjem povratne sprege. Upisana informacija se pamti sve dok je napon napajanja prisutan. Kod pojedinih memorijskih elemenata se primenjuje okidanje na nivo dok kod drugih ivično okidanje.

3.1.1. *Latch* kola

Jednobitna memorijska kola koja se okidaju na nivo se nazivaju *latch* kolima. Dva osnovna kola koja mogu da obavljaju ovu funkciju se dobijaju ukrštenim povezivanjem logičkih kola i nazivamo ih *SR latch* kolima (slika 3-3).

Na slici 3-3a je prikazana šema *SR latch* kola konstruisana *NILI* kolima. Kolo poseduje dva stabilna stanja zahvljajujući ukrštenom povezivanju: prvi slučaj odgovara setovanom stanju kada su vrednosti izlaza $Q=1$, $\overline{Q}=0$, dok za obrnute vrednosti izlaza se kaže da je kolo u resetovanom stanju. Slična je situacija kod *SR latch* kola izgrađenih od *NI* kola (slika 3-3b). Ponašanje kola u zavisnosti od vrednosti S (set) i R (reset) ulaza je prikazano u tabelama ispod odgovarajućih šema.



(a)

S	R	Q_{n+1}	$\overline{Q_{n+1}}$
0	0	Q_n	$\overline{Q_n}$
0	1	0	1
1	0	1	0
1	1	0	0

ZABRANJENO

(b)

S	R	Q_{n+1}	$\overline{Q_{n+1}}$
0	0	1	1
0	1	0	1
1	0	1	0
1	1	Q_n	$\overline{Q_n}$

ZABRANJENO

Slika 3-3: SR latch kolo sa ukrštenim povezivanjem logičkih kola: (a) NILI kolima, (b) NI kolima.

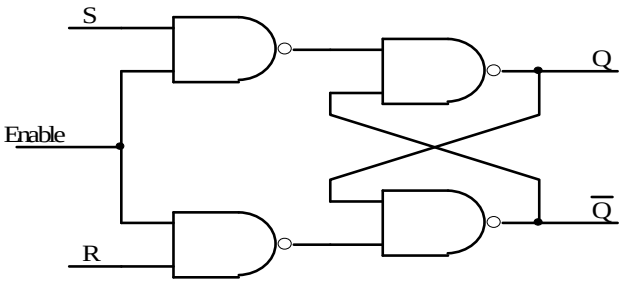
SR *latch* kolo konstruisano *NILI* kolima se setuje logičkom jedinicom dovedenom na *S* ulaz i resetuje se logičkom jedinicom na *R* ulazu. Za vreme setovanja ili resetovanja drugi ulaz koji se ne koristi mora da bude na logičkoj nuli. Kada se ne koriste ove operacije, potrebno je oba ulaza držati na logičkoj nuli i tada se vrednosti izlaza ne menjaju.

Slučaj kada se na oba ulaza dovode logičke jedinice, naziva se nedozvoljenom kombinacijom. Tada privremeno oba izlaza postaju logičke nule ali je neizvesno koji će izlaz ostati na nuli posle povratka ulaznih vrednosti na nule. Ukoliko se vrednosti ulaza u različitim trenucima vraćaju na nule tada će onaj ulaz odrediti stanje izlaza koji se zadnji vratio na nulu. Sa druge strane, ako se vrednosti ulaza menjaju istovremeno, onda kašnjenja logičkih kapija odlučuju o stanju izlaza.

Upravljanje *SR latch* kolima konstruisanih od *NI* kola se vrši pomoću niskih logičkih nivoa. U ovom slučaju će nule koje se istovremeno pojavljuju na ulazima, prouzrokovati neizvesnost u funkcionisanju i zato je neophodno ove situacije izbegavati kod *NI* kola.

Promene stanja *SR latch* kola se po potrebi mogu sinhronizovati pomoću jednog upravljačkog signala *(Enable)* na način kako je prikazano na slici 3-4. Sve dok je upravljački signal (signal dozvole) na logičkoj nuli, izlazi ulaznih *NI* kola su na logičkim jedinicama i vrednosti koji su upisane u *latch* kolo se ne mogu menjati. U trenutku promene signala *Enable* na visoki logički

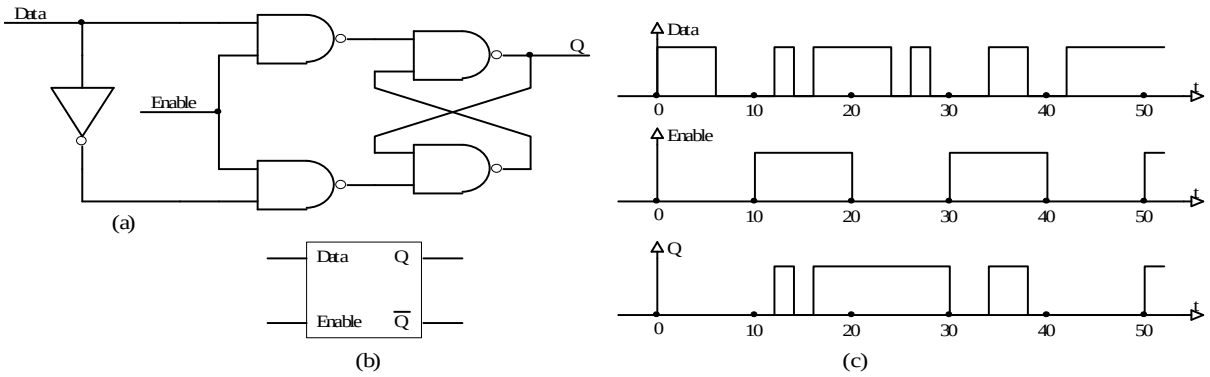
nivo, ulazna *NI* kola postaju aktivna i propuštaju ulazne *S* i *R* signale koji prouzrokuju odgovarajuću promenu stanja.



Slika 3-4: SR latch kolo sinhronizovan signalom dozvole (Enable).

Dok je signal *Enable* na visokom logičkom nivou svaka promena vrednosti ulaza *S* i *R* utiče na stanje *latch* kola. Ova vrsta sinhronizacije se naziva okidanjem na naponski nivo. Kolo koje se dobija na ovaj način se naziva transparentno *latch* kolo jer je "providno" za set i reset signale sve dok je upravljački signal aktivan.

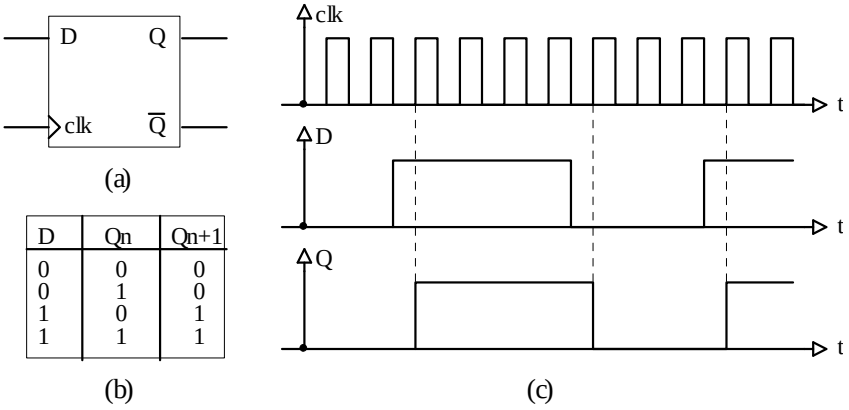
Nedozvoljene kombinacije ulaznih signala je moguće izbeći dodavanjem invertora prethodnom kolu (slika 3-5a). Uloga invertora je da zaštiti kolo od zabranjene kombinacije (kada su istovremeno prisutni set i reset signali na ulazima). Ovo kolo se naziva *D latch* kolo jer poseduje samo jedan ulaz za podatak (*Data*). Uobičajena šematska oznaka *D latch*-a je data na slici 3-5b, a princip rada se može shvatiti na osnovu dijagrama na slici 3-5c. Osobina transparencije je prisutna i kod ovog kola. Ulaz *Data* nema uticaja na izlaz kada je signal dozvole odsutan, dok u suprotnom slučaju signal sa ulaza (sa malim kašnjenjem) prelazi na izlaz. Signal koji je prisutan na izlazu u trenutku silazne ivice signala dozvole ostaje važeći sve dok signal dozvole ne postane ponovo aktivan.



Slika 3-5: D latch kolo sa sinhronizacijom (a), šematska oznaka (b) i dijagrami za objašnjenje principa rada (c).

3.1.2. *Flip-flop*-ovi

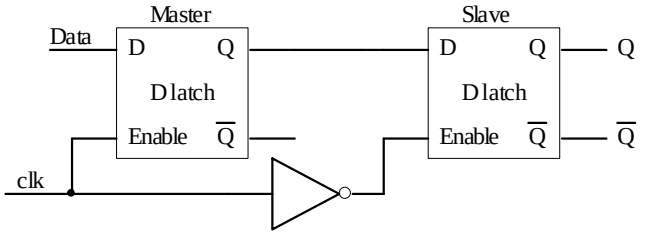
Flip-flop-ovi su elementarne memorije koje se okidaju na ivicu upravljačkog signala. Upis podatka se vrši na uzlaznu ili na silaznu ivicu upravljačkog signala, zavisno od realizacije kola. Podatak koji se upisuje je zadat ulaznim signalom ili ulaznim signalima. U odnosu na izvedbu ulaza, postoje nekoliko tipova *flip-flop*-ova. Kao prvi primer, navešćemo *D flip-flop* koji je u izvesnom srodstvu sa *D latch* kolom. Šematska oznaka *D flip-flop*-a, tabela koja prikazuje njegov princip rada i odgovarajući vremenski dijagrami su prikazani na slici 3-6. Kod svake uzlazne ivice takt signala (na dijagramu je to označeno isprekidanom linijom) se upisuje vrednost ulaza *D* u izlaz *Q*. Strelica na šematskoj oznaci uvek označava ulaz za takt signal (clk).



D	Qn	Qn+1
0	0	0
0	1	0
1	0	1
1	1	1

Slika 3-6: D flip-flop: (a) šematska oznaka, (b) kombinaciona tabela, (c) vremenski dijagrami.

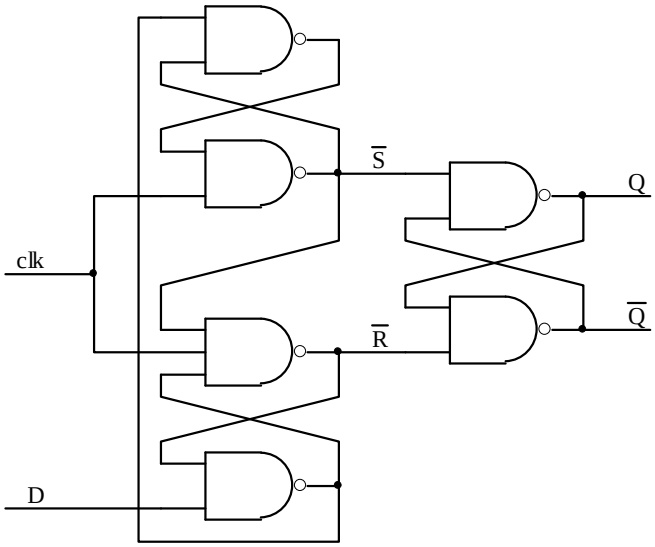
D flip-flop se može realizovati pomoću kaskadne veze dva *D latch* kola kao što je prikazano na slici 3-7 (*master-slave* veza). Sve dok je takt signal na visokom logičkom nivou, izlaz *Q* prvog *latch* kola (*master*) neprekidno preuzima vrednost ulaza *D* ali je istovremeno drugo *latch* kolo u zatvorenom stanju jer je signal dozvole doveden preko jednog invertora.



Slika 3-7: Realizacija D flip-flop-a kaskadnom vezom dva D latch kola (master-slave veza).

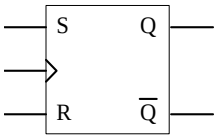
Posle silazne ivice takt signala prvo *latch* kolo se zatvara (zadržava stanje koje je bilo aktivno pri silaznoj ivici takt signala), dok drugo (*slave*) *latch* kolo postaje transparentno. Posmatrajući spolja može se reći da upis u *flip-flop* se vrši na silaznoj ivici takt signala.

Za realizaciju *D flip-flop*-a postoje i druga rešenja koja su bolja jer sadrže manji broj logičkih kapija i vremena kašnjenja su im manja. Slika 3-8 prikazuje jedno takvo kolo koje se okida uzlaznom ivicom. Sve dok je takt signal na niskom logičkom nivou i ulazi $\overline{S}$ i $\overline{R}$ su na visokom logičkom nivou i nije moguće izvršiti upis u izlazno *latch* kolo. Po dve logičke kapije su prisutne na ulazima koje se takođe mogu smatrati *latch* kolima čija je funkcija priprema signala za upis u izlazno *latch* kolo. U trenutku promene takt signala na visoki logički nivo vrši se upis pripremljene vrednosti ulaznog signala u izlazno *latch* kolo.



Slika 3-8: Efikasnija realizacija D flip-flop-a sa manjim brojem logičkih kapija i manjim vremenima kašnjenja.

Postoje i druge vrste *flip-flop*-ova, koji se od *D flip-flop*-a razlikuju po broju ulaza i po upravljačkom algoritmu. To su *RS*, *T* i *JK flip-flop*-ovi. *RS flip-flop* se može smatrati kao verzija *RS latch* kola koja se okida ivicom takt signala. Njegova šematska oznaka i kombinaciona tabela su prikazana na slici 3-9.

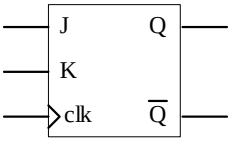


S	R	Qn	Qn+1
0	0	Q	Q
0	1	Q	0
1	0	Q	1
1	1	Q	nije def.

(a) (b)

Slika 3-9: (a) Šematska oznaka RS flip-flop-a i (b) njegova kombinaciona tabela.

Stanje izlaza *RS flip-flop*-a, slično odgovarajućem *latch* kolu, je nepredvidljivo u onim slučajevima kada su istovremeno prisutne logičke jedinice na oba ulaza. Kao rešenje ovog problema nastao je *JK flip-flop* čija su šematska oznaka i kombinaciona tabela prikazana na slici 3-10. *JK flip-flop*-ovi invertuju vrednosti svojih izlaza pri svakoj silaznoj ivici takt signala u slučaju istovremeno prisutnih logičkih jedinica na oba ulaza.

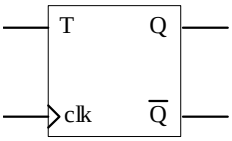


J	K	Qn	Qn+1
0	0	Q	Q
0	1	Q	0
1	0	Q	1
1	1	Q	$\overline{Q}$

(a) (b)

Slika 3-10: (a) Šematska oznaka JK flip-flop-a i (b) njegova kombinaciona tabela.

T flip-flop poseduje samo jedan ulaz za podatke (slika 3-11a) i zato se njegova kombinaciona tabela sastoji samo od dva reda (slika 3-11b). Ako je na ulazu *T* logička jedinica onda se vrednost izlaza invertuje pri svakoj uzlaznoj ivici takt signala, dok za logičku nulu vrednost izlaza ostaje nepromenjena.

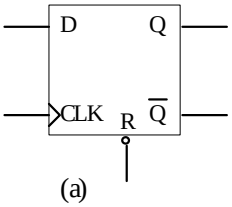


T	Qn	Qn+1
0	Q	Q
1	Q	$\overline{Q}$

(a) (b)

Slika 3-11: (a) Šematska oznaka T flip-flop-a i (b) njegova kombinaciona tabela.

Obično se dva ili četiri *flip-flop*-a smeštaju u zajedničko kućište kod *MSI* tehnologije. Pored sinhronizacionih ulaza redovno ovi *flip-flop*-ovi sadrže i asinhrone set i/ili reset ulaze. Na slici 3-12 su prikazana šematska oznaka i kombinaciona tabela *D flip-flop*-a sa asinhronim reset ulazom. Reset ulaz je nezavisan od takt signala. U prikazanom primeru mali krug na Reset ulazu na šematskoj oznaci ukazuje na osobinu da se proces resetovanja aktivira niskim logičkim nivoom.

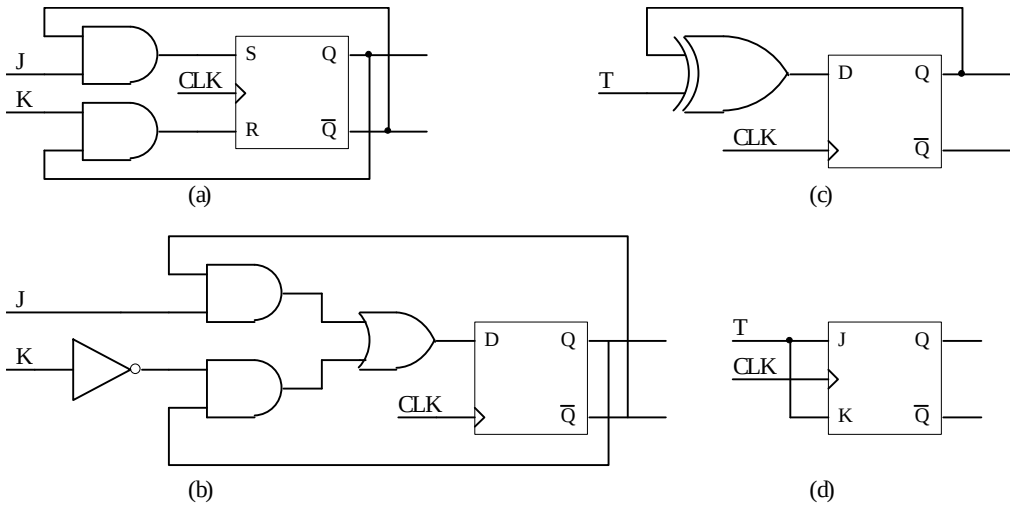


R	CLK	D	Qn	Qn+1
0	X	X	0	1
1	↑	0	0	1
1	↑	1	1	0

(a) (b)

Slika 3-12: D flip-flop sa asinhronim reset ulazom: (a) šematska oznaka, (b) kombinaciona tabela.

Za konstruisanje sekvencijalnih mreža je moguće primeniti bilo koji tip *flip-flop*-a. Tip primenjenog *flip-flop*-a utiče na realizaciju kombinacionog dela sekvencijalne mreže. Složenost kombinacionih mreža se značajno smanjuje pri optimalnom izboru tipa *flip-flop*-a. Nažalost, ne postoji jednoznačno pravilo za biranje odgovarajućeg tipa *flip-flop*-a i jedino ponavljanjem sinteze sekvencijalne mreže se može odrediti optimalan izbor. Po potrebi *flip-flop* se može pretvoriti u drugi *flip-flop* dodavanjem potrebne kombinacione mreže. Slika 3-13 prikazuje primere takvih transformacija.



Slika 3-13: Transformacija flip-flop-ova: (a) Transformacija RS flip-flop-a u JK flip-flop, (b) transformacija D flip-flop-a u J-K flip flop, (c) transformacija D flip-flop-a u T flip-flop, (d) transformacija JK flip-flop-a u T flip-flop.

U današnjoj VLSI tehnologiji više nije presudan broj korišćenih logičkih kapija za realizaciju kombinacionog dela sekvencijalnih meža. Čak ni minimizacija broja korišćenih *flip-flop*-ova više nije bitna pri realizaciji. Mnogo je važnije da se način realizacije prilagodi unutrašnjoj strukturi programabilnog logičkog kola *(PLD)* i da se dobije što preglednije i pouzdanije rešenje.

3.2. *Opis i konstruisanje logičkih automata*

Sekvencijalne mreže koje su konstruisane za rešavanje problema u automatici ili u obradi signala nazivamo logičkim automatima. Obično pri projektovanju logičkih automata se polazi od jezičkog opisa problema. Na osnovu toga se nacrta dijagram stanja ili njemu ekvivalentna tabela stanja koja tačno odslikava ponašanje logičkog automata. Na osnovu kombinacione tabela se određuju jednačine kombinacione mreže logičkog automata. Povezivanjem dobijene kombinacione mreže sa odgovarajućim *flip-flop*-ovima nastaje kompletna mreža.

Na isti način projektuju proizvođači registre opšte namene i brojače (*MSI* kola koja se serijski proizvode) o kojima će biti reči u poglavljima 3.3 i 3.4. Softveri koji služe za sintezu digitalnih kola unutar programabilnih logičkih kola takođe na sličan način konstruišu logičke automate.

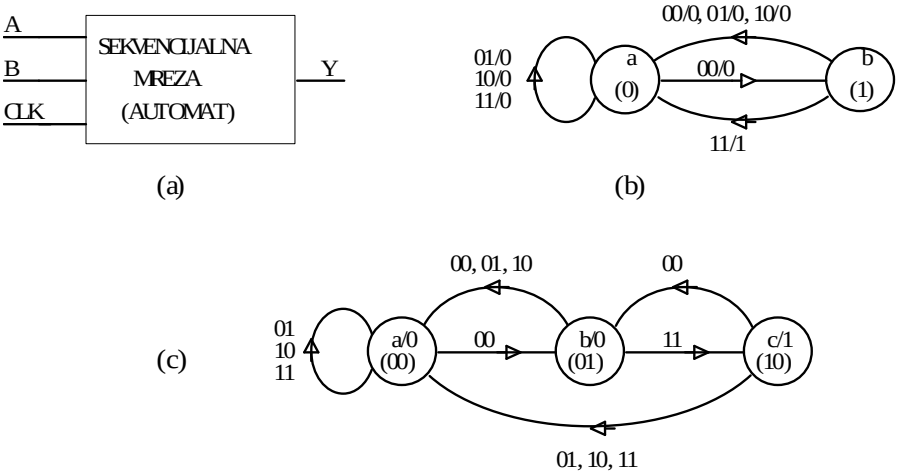
3.2.1. Dijagram stanja

Dijagram stanja predstavlja precizan i pregledan način opisivanja logičkih automata. Dijagrami stanja na slici 3-14a prikazuju jedan automat, koji daje logičku jedinicu na *Y* izlazu kada su se u dve prethodne periode takt signala redom pojavile kombinacije *00* a zatim *11* na ulazima *A* i *B*. U svim ostalim slučajevima izlaz je na logičkoj nuli.

Čvorovi grafa obeleženi krugovima odgovaraju pojedinim stanjima automata. Stanja se kodiraju odgovarajućim brojem *flip-flop*-ova. U krugove se upisuju kodovi stanja ili neki jasni

nazivi stanja. Linije sa strelicama koje povezuju čvorove predstavljaju prelaze između stanja. Na linije se pišu ulazne kombinacije pri kojima se dešavaju date promene stanja.

Dijagram stanja se menja u zavisnosti od tipa sekvencijalne mreže (*Mealy*-jev ili *Moore*-ov model). Kod *Mealy*-jevog modela na granama grafa se prikazuju vrednosti izlaza koji se javljaju pri određenom prelazu (slika 3-14b), dok kod *Moore*-ovog modela vrednosti izlaza se dobijaju dekodiranjem trenutnog stanja i zato se vrednosti izlaza upisuju u krugove koji označavaju stanja (slika 3-14c). I broj stanja kod navedenih modela može biti različit: broj stanja *Mealy*-jevog modela za dati automat je dva, a kod *Moore*-ovog modela tri.



Slika 3-14: (a) Jedna sekvencijalna mreža, (b) njen dijagram stanja u slučaju Mealy-jevog modela, (c) dijagram stanja u slučaju Moore-ovog modela.

3.2.2. Tabela stanja

Sledeća faza sinteze automata je formiranje tabele stanja. Tabela sadrži iste podatke koje sadrži i dijagram stanja ali u pogodnijoj formi za sintezu mreže. Tabele stanja koje odgovaraju prethodnom dijagramu stanja (slika 3-14) su prikazane u tabelama 3-1 i 3-2.

Tabela 3-1: Tabela stanja datog automata u slučaju Mealy-jevog modela.

Trenutno stanje	Sledeće stanje BA= 00 01 10 11	Izlaz Y BA= 00 01 10 11
a (0)	b a a a	0 0 0 0
b (1)	a a a a	0 0 0 1

Tabela 3-2: Tabela stanja datog automata u slučaju Moore-ovog modela.

Trenutno stanje	Sledeće stanje BA= 00 01 10 11	Izlaz Y
a (00)	b a a a	0
b (01)	a a a c	0
c (10)	b a a a	1

U tabelu je potrebno upisati sva moguća stanja sekvencijalne mreže i sve kombinacije ulaznih signala. Zatim za sve te kombinacije se određuju naredna stanja i vrednosti izlaznih promenljivih. U tabelama 3-1 i 3-2 nazivi trenutnih stanja su upisana u prvu kolonu (*a*, *b* odnosno *a*, *b*, *c*) a odgovarajući kodovi stanja su dati u zagradama. U središnjem delu tabele se nalaze naredna stanja, koja kod oba modela zavise i od vrednosti ulaznih promenljivih. Zbog toga su iznad narednih stanja navedene pripadajuće kombinacije ulaznih signala.

Na desnoj strani tabele su definisane vrednosti izlazne promenljive (*Y*). Kod *Mealy*-jevog modela (tabela 3-1) vrednosti izlaza zavise i od trenutnih vrednosti ulaza. Zato su iznad vrednosti izlaza date sve moguće kombinacije ulaza. Kod *Moore*-ovog automata vrednosti izlaza su

jednoznačno određene trenutnim stanjima automata i zbog toga su svi izlazi navedeni u jednoj koloni.

3.2.3. Kodiranje stanja

Pomoću *n flip-flop*-ova teoretski je moguće kodirati najviše 2^n različitih stanja i zato je i realizacija složenih automata moguća sa relativno malim brojem elementarnih memorija. Kod klasičnog načina projektovanja sekvencijalnih mreža primenom *MSI* kola težnja je bila da se koristi minimalan broja *flip-flop*-ova. Kao rezultat, ulazne i izlazne kombinacione mreže su postale komplikovane. Posebnu pažnju treba posvetiti izlaznoj kombinacionoj mreži jer dekodiranje stanja zahteva puno komponenata.

Kod realizacije sa minimalnim brojem *flip-flop*-ova nije svejedno kako se kodiraju pojedina stanja. Dobar izbor koda može značajno smanjiti složenost realizacije sekvencijalne mreže pri čemu se i pouzdanost rada povećava. Za optimizaciju postoje određene sistematizovane metode ali one ne daju uvek optimalna rešenja. Danas se pod optimizacijom podrazumeva računarska sinteza i simulacija različitih izvedbi. O konačnom rešenju se odlučuje posle razmatranja pojedinih rešenja.

Tipovi korišćenih elementarnih memorija (*D, T, JK, RS*) takođe utiču na kompleksnost mreže koja se realizuje. Za određivanje optimalnog tipa *flip-flop*-a takođe ne postoji sistematizovani postupak. Postupak optimizacije je još kompleksniji ako se u okviru istog projekta koriste različiti tipovi *flip-flop*-ova. U ovom slučaju se posebno preporučuje računarska sinteza. Softveri za projektovanje omogućuju brzo analiziranje različitih rešenja što skraćuje vreme pronalaženja najboljeg rešenja.

Kod savremenih načina projektovanja primenom *PLD* kola nije neophodno ili nije svrsishodno kodirati stanja sa minimalnim brojem memorijskih elemenata. Često se primenjuje rešenje: jedno stanje - jedan memorijski element. Kao rezultat, dobije se veliki broj *flip-flop*-ova ali je izbegnut proces dekodiranja stanja.

3.2.4. Jednačine upravljanja za *flip-flop*-ove

Kod oba modela automata naredno stanje se određuje na osnovu trenutnog stanja i trenutne vrednosti ulaznih promenljivih. Zadatak projektanta je izrada odgovarajućih kombinacionih mreža (eksitacione mreže) koje na svojim izlazima daju potrebne ulazne signale za *flip-flop*-ove. Pod uticajem tih signala upisuje se odgovarajuća vrednost u *flip-flop* pri narednoj uzlaznoj ili silaznoj ivici takt signala.

Najjednostavnija jednačina za kombinacione mreže *flip-flop*-ova se dobija za *D flip-flop:* na *D* ulaz treba dovesti novo (sledeće) stanje *flip-flop*-a. Kod drugih *flip-flop*-ova potrebno je uzeti u obzir koja kombinacija ulaznog signala može da dovede do željenog stanja na izlazu.

Za konstruisanje ulazne i izlazne kombinacione mreže potrebno je tabelu 3-1 (*Mealy*-jev model) svesti na takav oblik da postupak izvođenja jednačina date mreže bude što jednostavniji (tabela 3-3).

Qn	BA	Qn+1=D	Y
0	00	1	0
	01	0	0
	10	0	0
	11	0	0
1	00	0	0
	01	0	0
	10	0	0
	11	0	1

Tabela 3-1: Eksitaciona i izlazna kombinaciona tabela Mealy-jevog automata datog na slici 3-14b.

Tražena eksitaciona jednačina na osnovu date tabele je:

$$D = \overline{Q}\,\overline{B}\,\overline{A}$$

Da bi realizacija kola u složenijim slučajevima bila jednostavnija, potrebno je izvršiti minimizaciju odgovarajućih logičkih funkcija.

Eksitacione i izlazne tabele za *Moore*-ov model (slika 3-14c) su date u tabeli 3-4. Na osnovu tabele, u slučaju realizacije željenog kola sa *D flip-flop*-ovima tražene jednačine za eksitaciju su:

$$D1 = \overline{Q1}\,Q0\,BA$$
$$D0 = \overline{Q1}\,\overline{Q0}\,\overline{B}\,\overline{A} + Q1\,\overline{Q0}\,\overline{B}\,\overline{A} = \overline{Q0}\,\overline{B}\,\overline{A}$$

$Q1_nQ0_n$	BA	$Q1_{n+1}$=D1, $Q0_{n+1}$=D0	Y
00	00	01	0
	01	00	0
	10	00	0
	11	00	0
01	00	00	0
	01	00	0
	10	00	0
	11	10	0
10	00	01	1
	01	00	1
	10	00	1
	11	00	1
11	00	00	0
	01	00	0
	10	00	0
	11	00	0

Tabela 3-4: Eksitaciona i izlazna kombinaciona tabela Moore-ovog automata datog na slici 3-14c.

3.2.5. Formiranje izlaza

Kod *Mealy*-jevog automata broj promenljivih stanja je manji ali je potrebno dovesti i ulazne promenljive na izlaznu kombinacionu mrežu. Na osnovu tabele 3-1 oblik izlazne jednačine je:

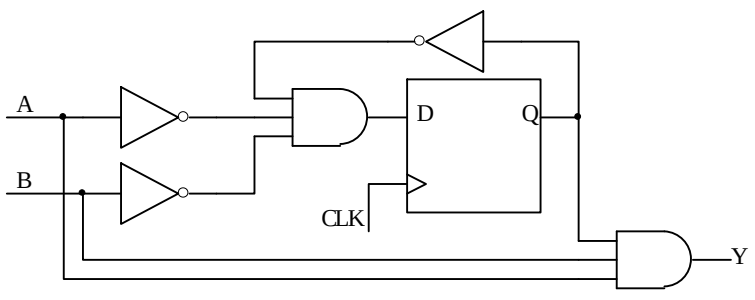
$$Y = QBA\,.$$

Vrednosti izlaznih promenljivih kod *Moore*-ovog modela zavise samo od trenutnog stanja mreže. Najjednostavniji slučaj je kada su izlazi *flip-flop*-ova ujedno i izlazi samog automata. Ako to nije slučaj, izlazni signali se generišu pomoću odgovarajuće kombinacione mreže. Ulazi kombinacione mreže su stanja *flip-flop*-ova. Na osnovu tabele 3-4 izlazna jednačina *Moore*-ovog automata (slika 3-14c) posle minimizacije je:

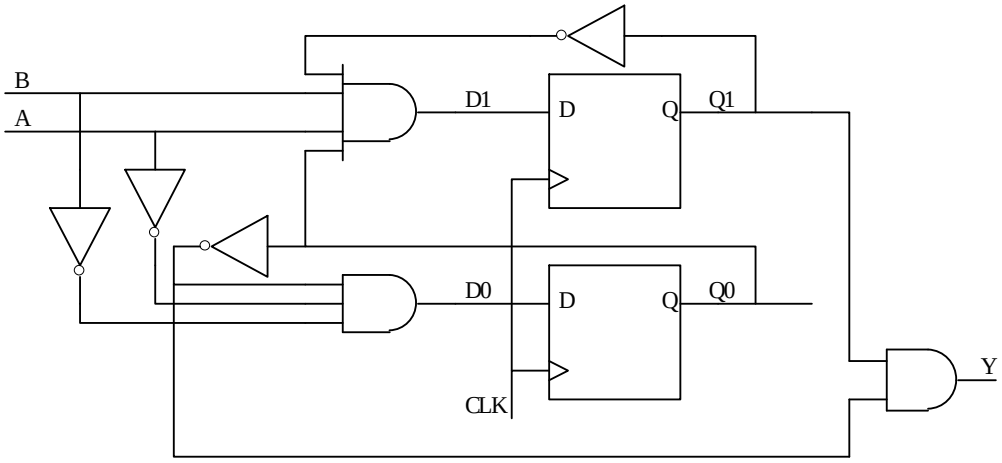
$$Y = Q1\,\overline{Q0}\,.$$

3.2.6. Formiranje kompletne sekvencijalne mreže

Crtanje kompletne sekvencijalne mreže se vrši sumiranjem dosadašnjih rezultata. Memorijski elementi koji se postavljaju prema odabranom kodu, predstavljaju centralni deo mreže. Na osnovu upravljačkih jednačina za *flip-flop*-ove se crta ulazna kombinaciona mreža. Na kraju, prema izlaznim jednačina-ma se formira izlazna kombinaciona mreža. Logička mreža *Mealy*-jevog automata (slika 3-14b) je prikazana na slici 3-15. Slično, odgovarajuća logička šema *Moore*-ovog automata (slika 3-14c) je prikazana na slici 3-16.



Slika 3-15: Logička mreža koja realizuje Mealy-jev automat dat na slici 3-14b.



Slika 3-16: Logička mreža koja realizuje Moore-ov automat dat na slici 3-14c.

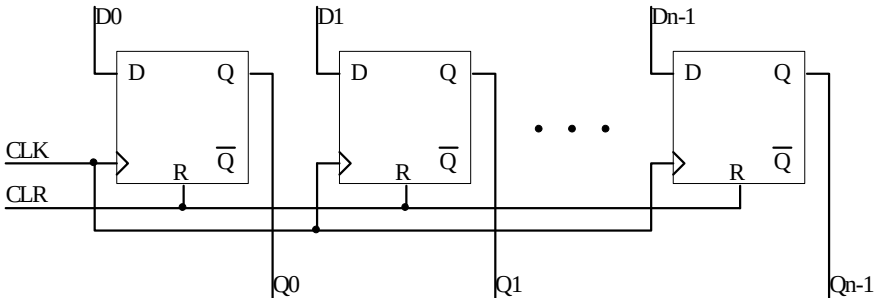
Ulogu prikazanih sekvencijalnih mreža (praćenje ulaznih sekvenci) može da obavlja i jedan mikrokontroler koji je programiran na odgovarajući način. Međutim, ne treba zaboraviti da su hardverska rešenja u ovakvim slučajevima neuporedivo jednostavnija i brzina njihovog rada je znatno veća.

3.3. *Registri*

Registri su sekvencijalne mreže koje služe za privremeno pamćenje male količine podataka. Pamćenje podataka obično vrši niz *D flip-flop*-ova ili neki drugi tipovi *flip-flop*-ova ili *latch* kola. Upis i čitanje podataka se vrši serijski (bit po bit) ili paralelno (svaki bit istovremeno). Postoje stacionarni i *shift* registri. Pored pamćenja, *shift* registri su u stanju i da pomeraju podatke bit po bit.

3.3.1. Obični (stacionarni) registri

Na slici 3-17 je prikazana logička šema stacionarnog registra kapaciteta od *n* bita. Za pamćenje se koriste *D flip-flop*-ovi kojima se upravlja zajedničkim takt (*CLK*) signalom. Upis se vrši paralelno pri silaznoj ivici takt signala. Sadržaj *flip-flop*-ova se može anulirati pomoću asinhronih reset ulaza koji su povezani na *CLR* signal. Čitanje podataka sa izlaza je uvek moguće, osim jednog kratkog vremenskog intervala posle upisa, dok izlazi *flip-flop*-ova ne stignu u novo stabilno stanje. Ako je potrebno, korišćenjem takvih *flip-flop*-ova koji poseduju izlaze sa tri stanja, moguće je više registara povezati na jednu zajedničku liniju podataka.



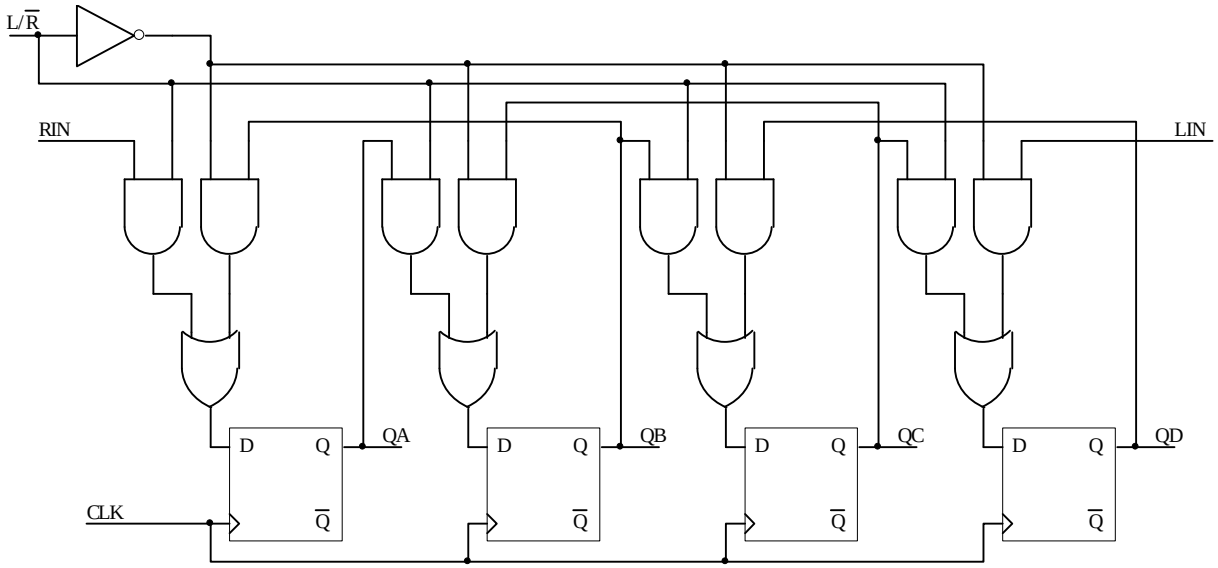
Slika 3-17: Logička šema stacionarnog registra.

3.3.2. Pomerački *(shift)* registri

Kod *shift* registara moguće je preneti sadržaj elementarnih memorija u susednu memorijsku lokaciju pri aktivnoj ivici takt signala. Pomeranje se može vršiti ili u levom ili u desnom smeru dok kod univerzalnih pomeračkih registara smer pomeranja se može zadati odgovarajućim upravljačkim signalom. Pri pomeranju sadržaj prethodnog memorijskog elementa se

upisuje u naredni, dok sadržaj zadnjeg memorijskog elementa se gubi. Kod složenih pomeračkih registara, pored serijskog, postoji i mogućnost paralelnog upisa i čitanja.

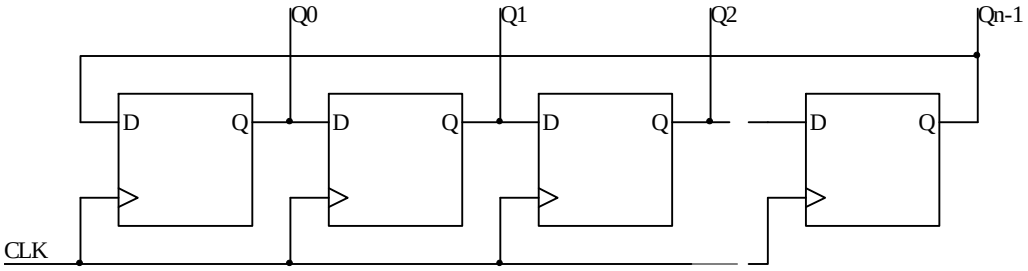
Na slici 3-18 je prikazana logička šema univerzalnog četvorobitnog pomeračkog registra. Smer pomeranja se bira pomoću $L/\overline{R}$ signala. *RIN* predstavlja serijski ulaz pri pomeranju udesno, dok pri pomeranju ulevo funkciju serijskog ulaza obavlja signal *LIN*. Serijski izlaz je *QA* ili *QD* u zavisnosti od smera pomeranja. Čitanje je moguće obaviti i u paralelnom režimu. Kod ovog rešenja ne postoji mogućnost paralelnog upisa. Takt (*CLK*) signal, čija je funkcija sinhronizacija pomeranja podataka, je zajednički za sve *flip-flop*-ove.



Slika 3-21: Logička šema dvosmernog pomeračkog registra.

3.3.3. Kružni registri (kružni brojači)

Spajanjem izlaza pomeračkog registra na sopstveni serijski ulaz se dobija kružni registar. Podatak koji je jednom upisan u registar će kružiti sve dok je takt signal prisutan na njegovom upravljačkom ulazu. Kako ovaj registar u normalnom režimu rada ne poseduje ulaz za podatke i po svojoj prirodi generiše ponavljajuće sekvence, ovo kolo se još naziva i kružni brojač (slika 3-19).

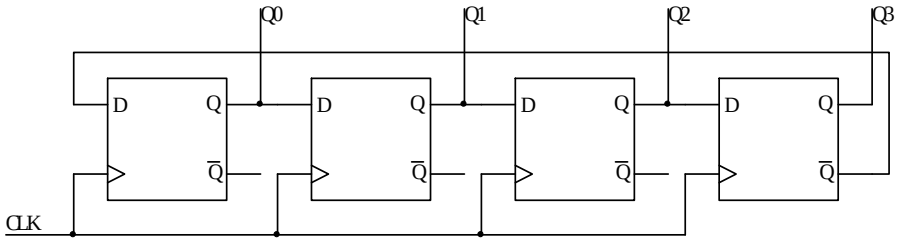


Slika 3-19: Logička šema kružnog registra (brojača).

U praktičnim primenama za kružni tok podataka potrebno je izvršiti upis odgovarajućeg sadržaja u brojač. Ovo se obično vrši paralelno. Najčešće se samo jedan *flip-flop* setuje i u ostale se upisuje logička nula ili obrnuto. Postoje i takva rešenja kod kojih su stanja *flip-flop*-ova nakon uključenja proizvoljna, ali posle nekoliko periode takt signala samo jedan *flip-flop* ostaje u setovanom stanju dok su svi ostali resetovani. Nakon prelaznog režima rad takvog kružnog brojača je pravilan.

Ako se izlaz zadnjeg *flip-flop*-a vraća na serijski ulaz kružnog brojača u invertovanom obliku (slika 3-20a) dobija se Džonsonov kružni registar (brojač). Nule i jedinice u normalnom

režimu rada kruže prema redosledu koji je dat u tabeli koja je prikazana na slici 3-20b. U slučaju *n* *flip-flop*-ova, sekvenca istih brojeva se ponavlja posle *2n* periode takt signala. Brojač će ući u pravilan režim rada i ako se svi *flip-flop*-ovi resetuju u trenutku uključenja. Ako se to propusti, moguće je da kolo na izlazu generiše pogrešne vrednosti. Raznim povratnim spregama je moguće obezbediti pravilno funkcionisanje kola.



(a)

STANJE	Q0	Q1	Q2	Q3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1
0	0	0	0	0

(b)

Slika 3-20: Jedno moguće rešenje (a) za logičku šemu Džonsonovog kružnog registra (brojača) i (b) odgovarajuća tabela stanja.

3.4. *Brojači*

Registri koji pod uticajem upravljačkih impulsa prolaze kroz unapred određena stanja se nazivaju brojačima. Upravljački signali mogu da potiču iz izvora takt signala ili iz bilo kog drugog kola koje je u stanju da generiše digitalne signale. Impulsi su obično periodični, ali brojači mogu da rade i pod uticajem impulsa koji se pojavljuju u slučajnim vremenskim trenucima. Pored takt ulaza pojedini brojači imaju i dodatne ulaze kojima se određuje način brojanja.

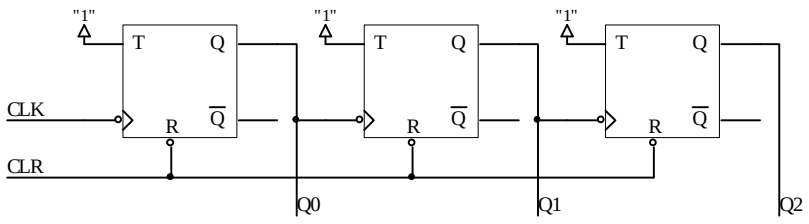
Redosled stanja brojača može da bude jednak redosledu binarnih brojeva (binarni brojač), ali po potrebi je moguće formirati i proizvoljan redosled. Broj različitih stanja kod brojača se naziva moduo brojača. Pored memorijskih elemenata brojači obično sadrže i ulaznu kombinacionu mrežu koja obezbeđuje odgovarajuće promene stanja u registru koji je centralni deo brojača. Obično, izlazna kombinaciona mreža koja je svojstvena sekvencijalnim mrežama ne postoji kod brojača. Izlazi memorijskih elemenata su ujedno i izlazi brojača. Znači, brojači su mreže tipa *Moore*.

Brojače je moguće svrstati u dve kategorije: asinhroni (serijski) i sinhroni (paralelni) brojači.

3.4.1. Asinhroni (serijski) brojači

Pri realizaciji asinhronih brojača *flip-flop*-ovi nemaju zajednički upravljački signal, nego se izlaz prethodnog *flip-flop*-a veže na takt ulaz narednog *flip-flop*-a. Na ovaj način se kompleksnost ulazne kombinacione mreže značajno smanjuje ili čak postaje suvišan.

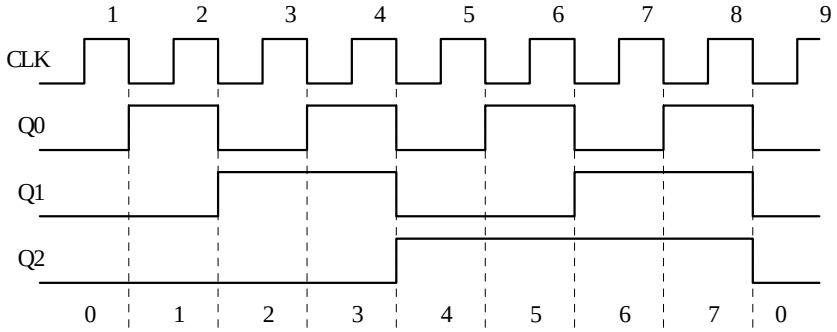
Najjednostavnija struktura asinhronog brojača se dobija kaskadnom vezom *flip-flop*-ova (slika 3-21). Upravljački signal se dovodi na takt ulaz prvog memorijskog elementa, a izlazi pojedinih *flip-flop*-ova se povezuju na takt ulaze narednih *flip-flop*-ova. Ako je potrebno, početno stanje brojača je moguće podesiti reset (*CLR*) signalom koji se istovremeno dovodi na asihnrone reset ulaze svih *flip-flop*-ova.



Slika 3-21: Asinhroni brojač realizovan T flip-flop-ovima.

Ukoliko korišćeni *flip-flop*-ovi reaguju na silaznu ivicu takt signala, brojač će brojati unapred u prirodnom binarnom kodu. Za slučaj asinhronog brojača prikazanog na slici 3-21

vremenski dijagrami su dati na slici 3-22. Posmatrajući redosled stanja kroz koja brojač prolazi (tabela 3-5) pri sukcesivnom delovanju takt signala, može se zaključiti da brojač broji po redosledu binarnih brojeva. Moduo brojanja je osam za slučaj tri memorijska elementa. U opštem slučaju, pomoću *n* memorijskih elemenata moguće je realizovati brojač modula 2^n. Ako je potreban brojač unazad, koriste se *flip-flop*-ovi sa okidanjem na uzlaznu ivicu i na takt ulaz narednog *flip-flop*-a se dovodi $\overline{Q}$ umesto izlaza *Q*.



Slika 3-22: Vremenski dijagrami takt signala i pojedinih izlaza asinhronog brojača.

Q2	*Q1*	*Q0*
0	*0*	*0*
0	*0*	*1*
0	*1*	*0*
0	*1*	*1*
1	*0*	*0*
1	*0*	*1*
1	*1*	*0*
1	*1*	*1*

Tabela 3-5: Redosled stanja trobitnog asinhronog brojača.

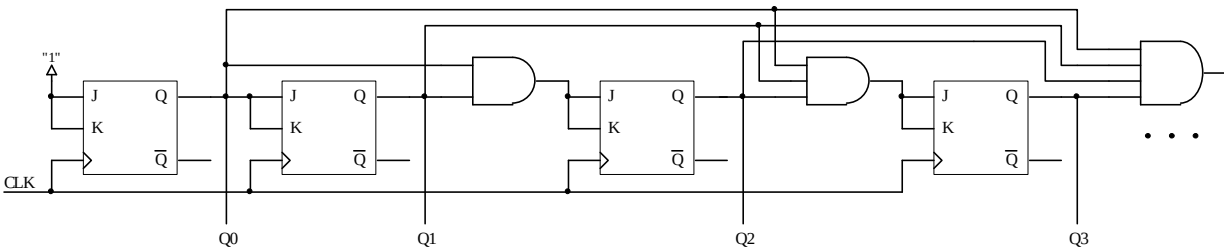
Na vremenskim dijagramima koji su prikazani na slici 3-22 su zanemarena kašnjenja *flip-flop*-ova. Zbog kaskadne veze, pojedina stanja se pojavljuju sa određenim vremenom kašnjenja u odnosu na upravljački signal. Ako se snimaju stanja pojedinih *flip-flop*-ova za vreme prelaznog procesa, dobijeni rezultat će najverovatnije biti pogrešan.

Sa jedne strane, rešenje ovog problema se sastoji u ograničavanju frekvencije takt signala (to obezbeđuje dovoljno vremena za iščezavanje hazardnih pojava), dok sa druge strane, iščitavanje vrednosti je potrebno vršiti neposredno pre pojave narednog upravljačkog signala. Kada se asinhroni brojač koristi kao delitelj frekvencije (npr. na slici 3-21 samo izlaz *Q2* se vodi dalje), frekvencija takt signala je ograničena samo sa vremenom kašnjenja prvog *flip-flop*-a.

3.4.2. Sinhroni (paralelni) brojači

Kod sinhronih brojača takt ulazi pojedinih *flip-flop*-ova su povezani na zajednički upravljački signal i tako se postiže da vrednosti izlaza približno u istom trenutku postanu važeći. Za promenu stanja *flip-flop*-ova u odgovarajućim vremenskim trenucima, neophodna je adekvatna kombinaciona mreža. Kod binarnih brojača (tabela 3-5) vrednost izlaza onog *flip-flop*-a koji je na mestu najmanje značajnog bita, menja se u svakoj periodi takt signala. Ovo se najlakše realizuje *T flip-flop*-om.

Ako se ceo brojač gradi pomoću *T flip-flop*-ova i ako je u datom momentu potrebno menjati stanje nekog *flip-flop*-a, tada se na njegov ulaz dovodi logička jedinica. Posmatrajući binarne brojeve na slici 3-26 može se zaključiti da bit veće težine se setuje samo kada su vrednosti svih ostalih bitova sa manjim težinama na logičkoj jedinici (npr. posle stanja *0111* sledi stanje *1000*). Dekodiranje ovih kombinacija se vrši pomoću *I* kapija. Na taj način je moguće formirati sinhroni binarni brojač koji je prikazan na slici 3-23. Do istog rezultata se dolazi i opštom metodom sinteze sekvencijalnih kola (objašnjeno kod logičkih automata).



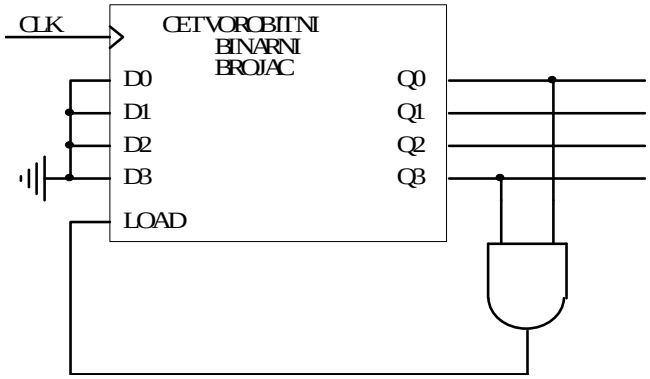
Slika 3-23: Logička šema sinhronog binarnog brojača.

Ova šema je samo dvostepena: posle aktivne ivice takt signala potrebno je da prođe samo jedno kašnjenje *I* kola i *flip-flop*-a kako bi vrednosti izlaza *flip-flop*-ova postale važeći.

Odgovarajućom promenom ulazne kombinacione mreže kod sinhronih brojača moguće je postići brojanje unazad. Kod brojača proizvedenih u *MSI* tehnologiji postoje i dvosmerna rešenja: kod ovih kola postoje dve posebne ulazne kombinacione mreže za oba slučaja (brojanje napred i nazad) i pomoću jednog kontrolnog signala se određuje koja će od te dve mreže biti aktivna.

Često se javlja potreba za povećanjem modula brojanja. *MSI* kola za brojanje obično sadrže i odgovarajuće izlaze (prenos, engl.: *carry*) i ulaze za kaskadno vezivanje pojedinih brojačkih modula. Prethodni modul, pomoću bita prenosa obaveštava naredni modul o tome da je stigao do najvećeg broja i da naredni stepen mora da inkrementira vrednost sopstvenog registra.

Moduo brojanja brojača sastavljenog od *n* memorijskih elemenata je moguće smanjiti sa 2^n. Za to postoje dva načina. Prvi način je primenom metode koja je prikazana kod projektovanja logičkih automata. Taj postupak se sastoji od formiranja dijagrama stanja i tabele stanja. Kod druge metode polazi se od gotovog brojača modula 2^n, koji na uobičajeni način funkcioniše do predviđenog zadnjeg stanja. Na brojač se dodaje jedan dekoder koji je posebno projektovan za detektovanje zadnjeg stanja (slika 3-24). Kada se detektuje zadnje stanje, dekoder sinhrono ili asinhrono resetuje sve *flip-flop*-ove. Posle toga se proces brojanja nastavlja sa nule. U prikazanom slučaju realizovani brojač je modula deset jer se posle devetog stanja (1001) vrši paralelni upis nultih vrednosti preko linija (D0...D3).



Slika 3-24: Logička šema brojača modula deset dobijena skraćivanjem modula brojanja.

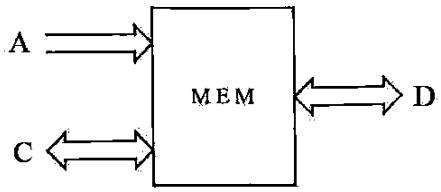
4. Složene mreže

Digitalna kola koja se razmatraju u nastavku se nazivaju složenim mrežama jer sadrže kako kombinacione tako i sekvencijalne elemente. Memorije su po prirodi sekvencijalne mreže kod kojih se upravljanje realizuje kombinacionim mrežama. Aritmetička kola se prvenstveno realizuju kombinacionim mrežama pri čemu se za njihovo upravljanje koriste logički automati. *D/A* i *A/D* pretvarači pored obe vrste (kombinacionih i sekvencijalnih) digitalnih elemenata sadrže i analogna kola.

4.1. *Memorije*

Memorije su digitalna kola koja služe za trajno ili privremeno pamćenje velike količine podataka. Uloga registara (poglavlje 3.3) je takođe pamćenje podataka, ali pamćenje velikih količina podataka zahteva efikasniju organizaciju. Danas su uređaji čija je osnovna namena pamćenje podataka veoma raznovrsni (tu spadaju i magnetni- i optički mediji). Ovde se razmatraju samo poluprovodničke memorije.

Za sve memorije važi blok dijagram koji je prikazan na slici 4-1. Pomoću adresnih linija (na šemi označene sa C) se određuje elementarna ćelija memorije sa kojom se želi komunicirati. Preko upravljačkih ulaza *V* se određuje svrha komunikacije: upis, čitanje ili dobijanje povratnih informacija o stanju memorije (npr. zauzetost). Pomoću linija A se prosleđuju podaci od drugog kola ka memoriji ili obrnuto.



Slika 4-1: Blok šema memorijskog kola.

4.1.1. Podela i osobine memorijskih kola

Podelu memorija koje sadrže mnoštvo različitih hardverskih rešenja je moguće izvršiti po različitim principima:

- Pristup memorijama onih uređaja koji se softverski programiraju treba da bude ostvareno u što kraćem vremenu i zato se tada primenjuju brze memorije, sa mogućnošću promene podataka u njima. Kod ovih uređaja istovremeno su potrebne i memorije u kojima upisani podatak ostaje permanentno i služe za pamćenje osnovnih programa.
- Način pamćenja podataka u memorijskim ćelijama može biti statičan ili dinamičan. Statičke memorijske ćelije su zapravo *flip-flop*-ovi (kao i kod registara), koji se na odgovarajući način organizuju kako bi pristup pojedinim ćelijama bio jednostavniji. Statičke ćelije pamte podatke sve dok se ne izvrši novi upis u njih i dok postoji napajanje. Postoje i takva rešenja gde memorije poseduju ugrađene *Li-ion*ske ćelije zahvaljujući kojima su sposobne da pamte svoj sadržaj više godina bez spoljašnjeg napajanja.
- Kod dinamičkih ćelija se koriste ulazne kapacitivnosti pojedinih mosfetova za pamćenje podataka. Prilikom upisa parazitna kapacitivnost se puni ili prazni u zavisnosti od toga da li je potrebno pamtiti logičku nulu ili jedinicu. Čitanje se vrši proverom stanja kanala mosfeta: provodno stanje kanala označava upisanu logičku jedinicu dok kanal u zakočenom stanju logičku nulu. Naelektrisanja skladištena u kapacitivnosti polako cure i zbog toga upisani podaci bi nestali posle određenog vremena. Rešenje ovog problema je periodično osvežavanje podataka. Projektovana su posebna kola za upravljanje procesom osvežavanja koji je sličan procesu čitanja.
- Identifikacija ćelije sa kojom se želi komunicirati se obično vrši pomoću adresiranja. Adresiranje bilo koje memorijske lokacije se naziva još i neposrednim pristupom (engl. :

random access). Pomoću adresiranja moguće je pristupiti bilo kojoj memorijskoj lokaciji za približno isto vreme ali je cena toga prilično složena mreža za adresiranje. Značajno se smanjuje kompleksnost memorije ako se podaci serijski upisuju odnosno iščitavaju. Ove memorije poseduju serijski pristup čiji je nedostatak da se vreme pristupa povećava sa povećanjem adrese tražene memorijske lokacije.

- Većina današnjih memorijskih kola se proizvode u *CMOS* tehnologiji ali postoje i kola izrađena u bipolarnoj tehnologiji. Veličina podatka čemu se može pristupiti u jednom koraku (jedno adresiranje ili inkrementiranje adrese) je jedan bit ili više bita (npr. osam ili šesnaest).
- Najvažniji parametri memorijskih kola, pored vrednosti napona napajanja i logičkih nivoa, su kapacitet (broj memorijskih ćelija) i brzina (upis, čitanje i brisanje).

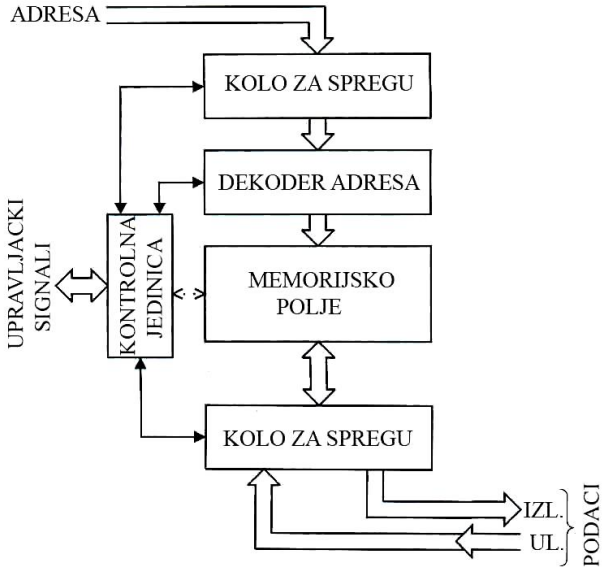
Postoje razni tipovi memorijskih kola :

- *RAM (engl.:random access memory)* memorije su memorije sa neposrednim pristupom i koriste se kao operativne memorije uređaja jer poseduju brz pristup memorijskim lokacijama (obično nekoliko desetina *ns*). Proizvode se u statičkoj i u dinamičkoj izvedbi. Dinamičke *RAM* ćelije su jeftinije od statičkih jer je njihova gustina pakovanja na silicijumskoj pločici znatno veća. Sa druge strane, dinamičke *RAM* ćelije zahtevaju spoljašnja kola za osvežavanje sadržaja memorijskih ćelija što komplikuje proces projektovanja.
- *ROM* (engl.: *read-only memory*) memorije su takve memorije čiji se sadržaj ne može menjati ali ga je moguće neograničen broj puta čitati. Analizom se može utvrditi da su *ROM* memorije zapravo kombinacione mreže sastavljene od kaskadne veze dekodera i kodera. Dekoder vrši dekodiranje vrednosti ulaznih adresnih linija a koder formira odgovarajuće izlazne vrednosti. Osnovni programi uređaja se obično čuvaju u ovom tipu memorije. Upis sadržaja u memoriju mogu da vrše proizvođači (engl.: *mask programmable ROM*) ili korisnici (*OTP ROM* – engl.: *one-time programmable ROM*). *ROM*-ovi se primenjuju samo u serijskoj proizvodnji jer je mogućnost jednostrukog programiranja u fazi razvoja nedovoljna.
- *EPROM* (engl.: *electrically programmable ROM*) je memorija sa mogućnošću višestrukog programiranja pomoću odgovarajućih upravljačkih signala. Pre promene sadržaja *EPROM*-a neophodno je izbrisati prethodno upisane podatke pomoću ultraljubičastih zraka. Ova kola su prepoznatljiva po prozoru koji se nalazi na gornjoj strani kućišta gde ulaze ultraljubičasti zraci tokom brisanja. Upis se vrši na sličan način kao i kod dinamičkih memorija sa tom razlikom da naelektrisanja koja menjaju stanje kanala mosfeta su smeštena tako da ostaju prisutna i nakon više godina. *EPROM* se može mnogo puta reprogramirati, ali se mora uzeti u obzir da proces brisanja traje više minuta što često usporava razvoj odnosno proizvodnju što je dovelo do pada popularnosti *EPROM*-a.
- Brisanje i upis kod *EEPROM*-a *(engl.: electrically erasable PROM)* se vrši pomoću električnih signala i zato one ne sadrže prozor na svojim kućištima za propuštanje ultraljubičastih zraka. Podrazumeva se da je brzina brisanja i upisa značajno manja u odnosu na brzinu brisanja i upisa kod *RAM*-a. *EEPROM*-i se često proizvode sa serijskim komunikacionim interfejsom. Ako nije potrebna velika brzina prenosa podataka, razumno je koristiti serijsku komunikaciju jer značajno smanjuje kućište pošto se koristi samo jedna linija za podatke.
- Pod nazivom *flash EEPROM*-i ili jednostavnije *flash* memorije se podrazumevaju takva kola koja se realativno brzo mogu brisati električnim signalima (red veličine je sekunda). Posle operacije brisanja moguće je izvšiti novi upis ali za razliku od *EPROM*-a i *EEPROM*-a, one ne zahtevaju posebno napajanje u toku programiranja odnosno upravljačke signale povećanih amplituda. Operacije brisanja i upisa mogu da vrše i mikrokontroleri koji upravljaju radom datog sistema. Obično postoje određena ograničenja pri brisanju odnosno dozvoljava se samo brisanje po blokovima. *Flash* memorije se mogu koristiti kao

programske memorije jer je njihova brzina čitanja velika ali ipak ne mogu zameniti *RAM* memorije jer je njihovo vreme upisa veliko. Međutim, trajno pamćenje podataka nakon isključenja napajanja predstavlja veliku prednost.

4.1.2. Unutrašnja struktura memorijskih kola

Kao što je spomenuto, pamćenje velike količine podataka zahteva određenu organizaciju unutar memorije. Struktura onih memorija koje rade na principu adresiranja je prikazana na slici 4-2.



Slika 4-2: Unutrašnja struktura memorijskog kola.

Centralni deo memorije je polje memorijskih ćelija, koje se sastoji od potrebnog broja elementarnih memorija. Adresa ćelije kojoj se želi pristupiti ulazi u adresni dekoder preko adresnog kola za spregu (bafer). Uloga dekodera je selektovanje (identifikacija) ćelije sa kojom se želi ostvariti komunikacija. Složenost dekodera veoma brzo raste (kvadratno) sa povećanjem broja memorijskih ćelija. Zato se elemetarne memorije postavljaju u matrični oblik umesto rednog (linearnog) rasporeda. U tom slučaju, adresiranje se vrši pomoću dva manja dekodera (dekoderi reda i kolone). Ukupan broj korišćenih logičkih kola za realizaciju ta dva dekodera je značajno manji nego u slučaju jednog dekodera. Ovaj pristup podrazumeva postojanje dva adresna ulaza za svaku memorijsku ćeliju. Zbog težnje za što manjim kućištima, adresne linije dinamičkih memorija velikih kapaciteta se multipleksiraju: posebno se učitavaju i pamte adrese kolona i redova.

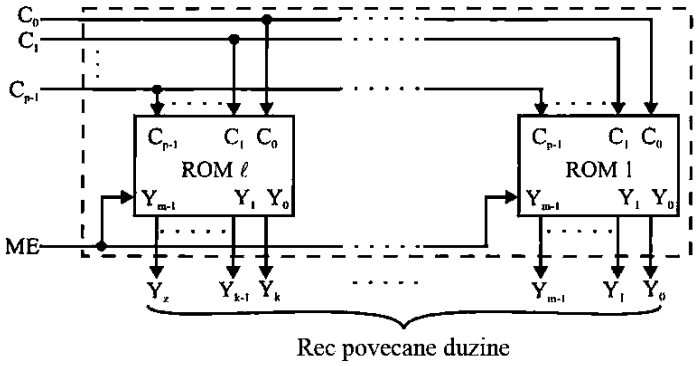
Unos podataka odnosno njihovo iščitavanje se vrši preko kola za spregu za podatke (bafer). Obično se iste linije podataka koriste za komunikaciju u oba smera. Razlog za to je uprošćavanje kućišta. Prilikom upisa, podaci moraju biti stabilni već neko vreme neposredno pre upisa. Kod iščitavanja neophodno je uzeti u obzir kašnjenje koje postoji između adresiranja i pojave podataka na izlazu. Kataloški podaci sadrže konkretne vrednosti ovih kašnjenja.

Uloga upravljačke jedinice je generisanje odgovarajućih upravljačkih signala prilikom upisa, čitanja ili brisanja za selektovanu memorijsku ćeliju. Inicijalizacija upisa se vrši pomoću *WE* (engl.: *write enable*) signala koji se šalje upravljačkoj jedinici memorije. Za čitanje je potrebno aktivirati signal *OE* (engl.: *output enable*). Upravljačka jedinica poseduje još jedan ulaz sa nazivom *CS* (engl.: *chip select*) pomoću kojeg je moguće zaustaviti ili pokrenuti rad celog kola. Uloga ovog ulaza je da omogući proširivanje kapaciteta memorije.

4.1.3. Proširivanje kapaciteta

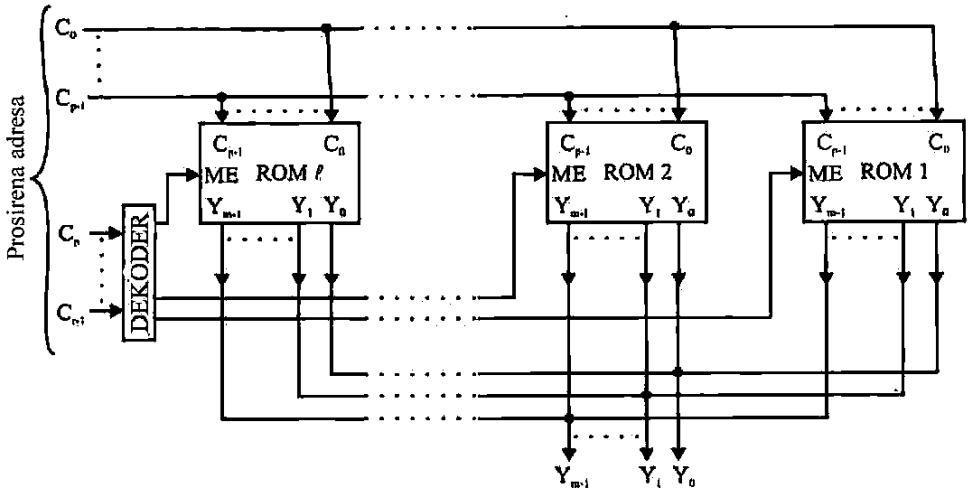
Kapacitet memorije dostupan unutar jednog kućišta je često je nedovoljan. Ili je potrebno povećati broj bitova (veličinu reči) koji se istovremeno iščitavaju ili broj reči u memoriji koji se može adresirati. Proširivanje se postiže korišćenjem više memorijskih kola koja se povezuju na odgovarajući način na zajedničkoj štampanoj ploči.

Proširivanje dužine reči je jednostavniji zadatak: memorijska kola se poređaju jedan pored drugog i adresne i upravljačke linije sa identičnim oznakama se povezuju (slika 4-3). U konkretnom slučaju se radi o *ROM* memorijama i funkcija jedinog upravljačkog signala je otvaranje njihovih izlaza. Izlazne linije podataka se mogu posmatrati kao deo jedne zajedničke magistrale podataka.



Slika 4-3: Način proširivanje dužine reči pri povezivanju ROM-ova.

Povećanje broja adresabilnih memorijskih reči se postiže povezivanjem više memorijskih kola pri čemu je potrebno dodatno spoljašnje kolo (dekoder) koje naizmenično aktivira pojedina memorijska kola (slika 4-4). Linije podataka pojedinih memorijskih kola se spajaju na zajedničku magistralu podataka. Deo adresnih linija je takođe zajednički, dok se ostatak vodi na dekoder koji odlučuje o tome koje kolo će se aktivirati u datom trenutku. U prikazanom slučaju (povezivanje *ROM*-ova), uloga dekodera je generisanje signala dozvole pojedinim memorijskim kolima. Kod *RAM*-ova su signali *WE* i *OE* zajednički za sva kola, dok su signali dozvole (*CS*) jedinstveni za svaku memoriju i generiše ih spoljašnji dekoder.



Slika 4-4: Povećanje broja adresabilnih memorijskih reči odgovarajućom vezom memorijskih kola.

4.2. *Aritmetičke jedinice*

Prvi složeniji digitalni uređaji su bili računari koji su razvijeni za izvršavanje masovnih i komplikovanih proračuna u kratkom vremenskom intervalu. Glavni zadaci današnjih računara su uređivanje i pamćenje podataka ali se i dalje značajni resursi posvećuju proračunima.

Aritmetičke jedinice sačinjavaju osnovu proračuna u računarima. Prvenstveno je reč o kombinacionim mrežama, koje vrednosti na ulazima posmatraju kao binarne brojeve i pomoću odgovarajućih mreža formiraju rezultat željene aritmetičke operacije. U nastavku se razmatraju jednostavna kola za sabiranje, množenje i aritmetički komparatori.

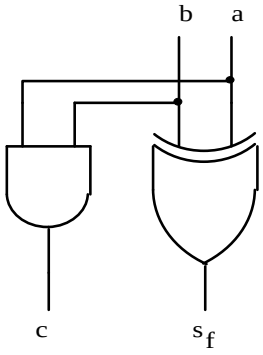
4.2.1. Sabirači

Osnovna kola u binarnoj aritmetici su sabirači. Operacija oduzimanja se realizuje sabiranjem u drugom komplementu. Operacija množenja se često izvodi korišćenjem sabiračkih kola.

Osnovnu jedinicu složenijih sabirača čine polusabirači, koji sabiraju dva jednobitna binarna broja prema tabeli koja je prikazana na slici 4-5a. Rezultat sabiranja u decimalnom brojnom sistemu može biti 0, 1 ili 2. Najmanje značajan bit u rezultatu predstavlja zbir sabiranja (s_f) dok je najznačajniji bit (c – engl.: *carry*) predstavlja prenos i formira ulaz za sledeći stepen.

a	b	c	s_f
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(a)



(b)

Slika 4-5: (a) Kombinaciona tabela polusabirača i (b) njegova logička šema.

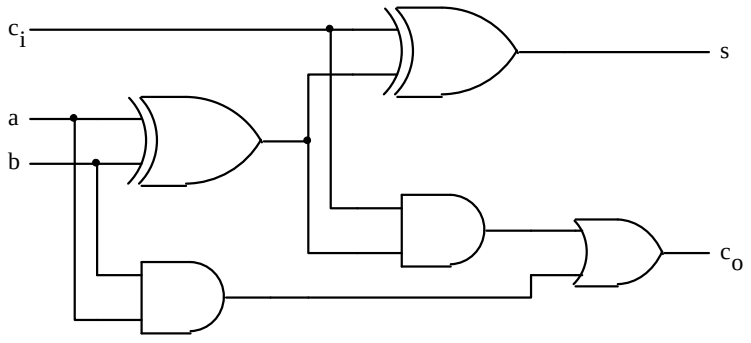
Jednačine koje se dobiju na osnovu prethodne tabele su:

$$s_f = \bar{a}b + a\bar{b} = a \oplus b$$
$$c = ab\,.$$

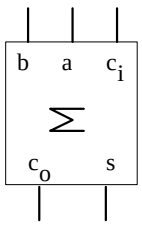
Na osnovu ovih jednačina je nacrtana šema polusabirača koja je prikazana na slici 4-5b. Nedostatak polusabirača je da se ne mogu povezati u kaskadu kako bi sabirali višebitne brojeve. Za ovo je potrebno rešiti sabiranje bita prenosa iz prethodnog polusabirača sa bitovima sledećeg polusabirača. Na ovaj način se dobija potpuni sabirač za koji važe sledeće jednačine:

$$s = a \oplus b \oplus c_i$$
$$c_o = (a \oplus b)c_i + ab$$

Logička šema potpunog sabirača dobijena na osnovu prethodnih jednačina je prikazana na slici 4-6a, a njegova šematska oznaka koja će se u nastavku koristiti je data na slici 4-6b.



(a)

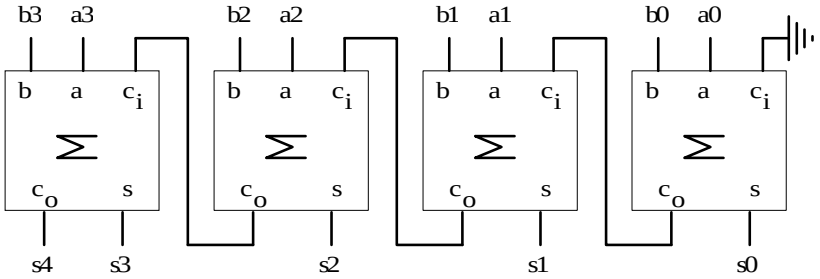


(b)

Slika 4-6: (a) Logička šema potpunog sabirača i (b)njegova šematska oznaka.

Kaskadnom vezom potpunih sabirača na način koji je prikazan na slici 4-7 moguće je sabiranje višebitnih brojeva. Formiranje zbira kod date mreže se vrši paralelno, s tim da se bitovi

prenosa prostiru serijski kroz pojedine module sabirača što značajno usporava rad kola. Postoje kola koja pomoću dvostepene logičke mreže određuju da li se može očekivati bit prenosa kod nekog bita. Takvo rešenje značajno ubrzava rad sabirača.



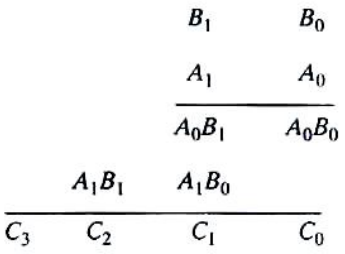
Slika 4-7: Sabiranje dva četvorobitna broja kaskadnom vezom potpunih sabirača.

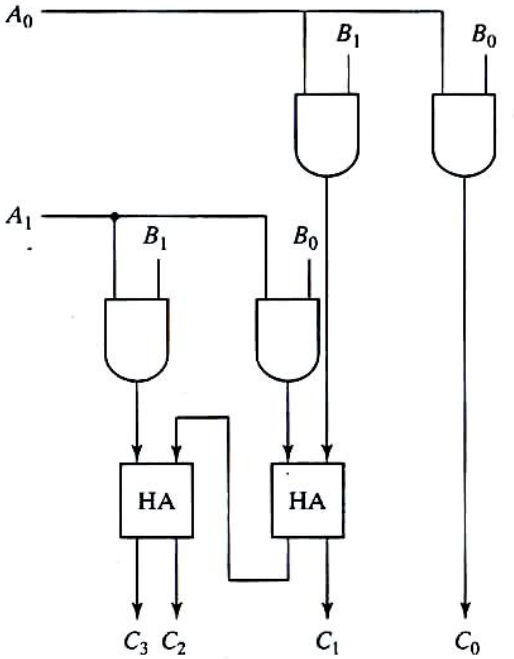
4.2.2. Množači

Digitalni množači se mogu realizovati na više načina. Množenje se može svesti na uzastopno sabiranje: prvi množilac se množi sa pojedinačnim bitovima drugog množioca i dobijeni međurezultati se sabiraju. Ovakav množač se rešava sekvencijalnom mrežom koja pomoću jednog logičkog automata usmerava tok podataka između registara.

Moguće je konstruisati množač i primenom kombinacione mreže. U ovom slučaju neophodno je formirati takvu kombinacionu tabelu u kojoj su sve moguće varijacije množioca uzete u obzir. Strukture množača koji su formirani pomoću sekvencijalnih mreža su jednostavnije (sastoje se od manjeg broja logičkih kapija), ali su znatno sporiji od množača realizovanih pomoću kombinacionih mreža jer se rezultat množenja dobija tek nakon više koraka.

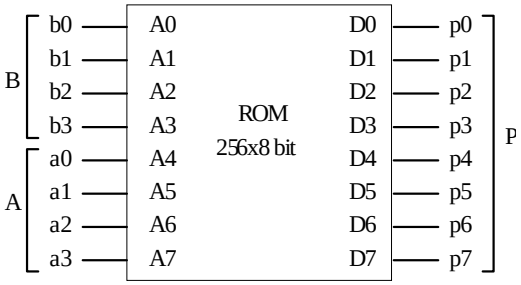
Moguće je konstruisati dvobitni množač na bazi kombinacionih mreža i bez određivanja njihove kombinacione tabele (slika 4-8). Vrednost najmanje značajnog bita A_0 se množi sa ciframa (bitovima) broja B pomoću dva I kola. Najmanje značajan bit (C_0) međurezultata je ujedno i najmanje značajan bit množenja. Sledeći značajniji bit množenja (C_1) se dobija pomoću jednog polusabirača u koji ulazi bit A_1 i najmanje značajan bit broja B. Najznačajnije bitove množenja (C_2 i C_3) formira drugi polusabirač koji sabira eventualni bit prenosa iz prvog sabiranja i logički proizvod A_1B_1. Ispravnost navedenog postupka dokazuje pismeno množenje koje je navedeno pored logičke šeme.





Slika 4-8: Kombinaciona mreža koja množi dva dvobitna broja.

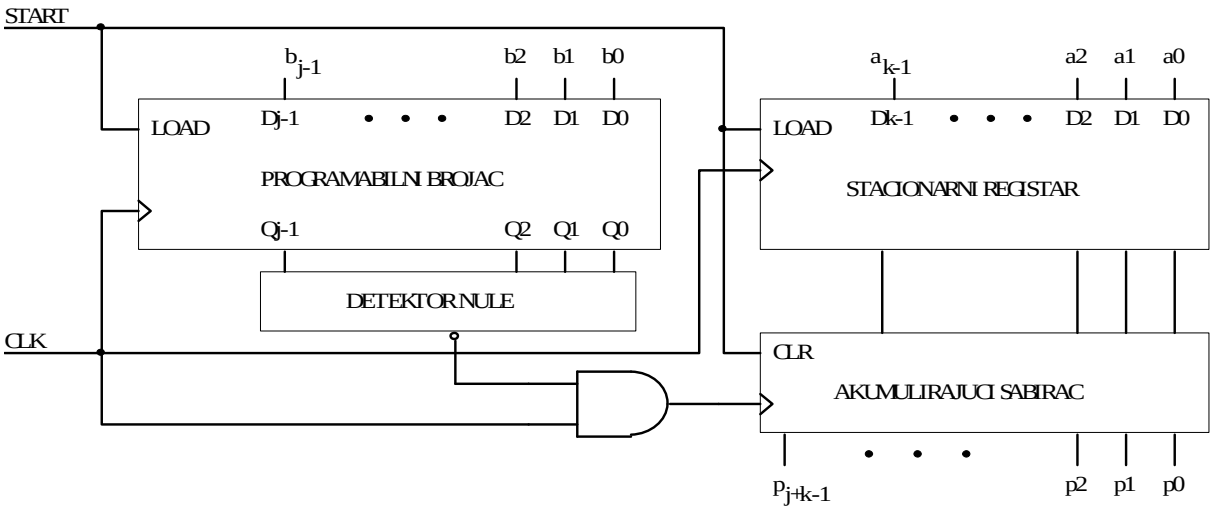
Realizacija digitalnih kola za množenje brojeva od više bita zahteva sve veći broj logičkih kapija. Rešenje množača za veće binarne brojeve je u primeni *ROM*-a. *ROM* kapaciteta 256x8 bita (slika 4-9) koji je na odgovarajući način programiran, može da obavlja funkciju množača za dva četvorobitna broja. Kada se množioci (*A* i *B*) spoje na odgovarajuće adresne linije, iščita se sadržaj *ROM*-a što se može smatrati kao jedna tabela koja sadrži sve moguće vrednosti proizvoda (*P*).



Slika 4-9: Realizacija množača sa ROM-om.

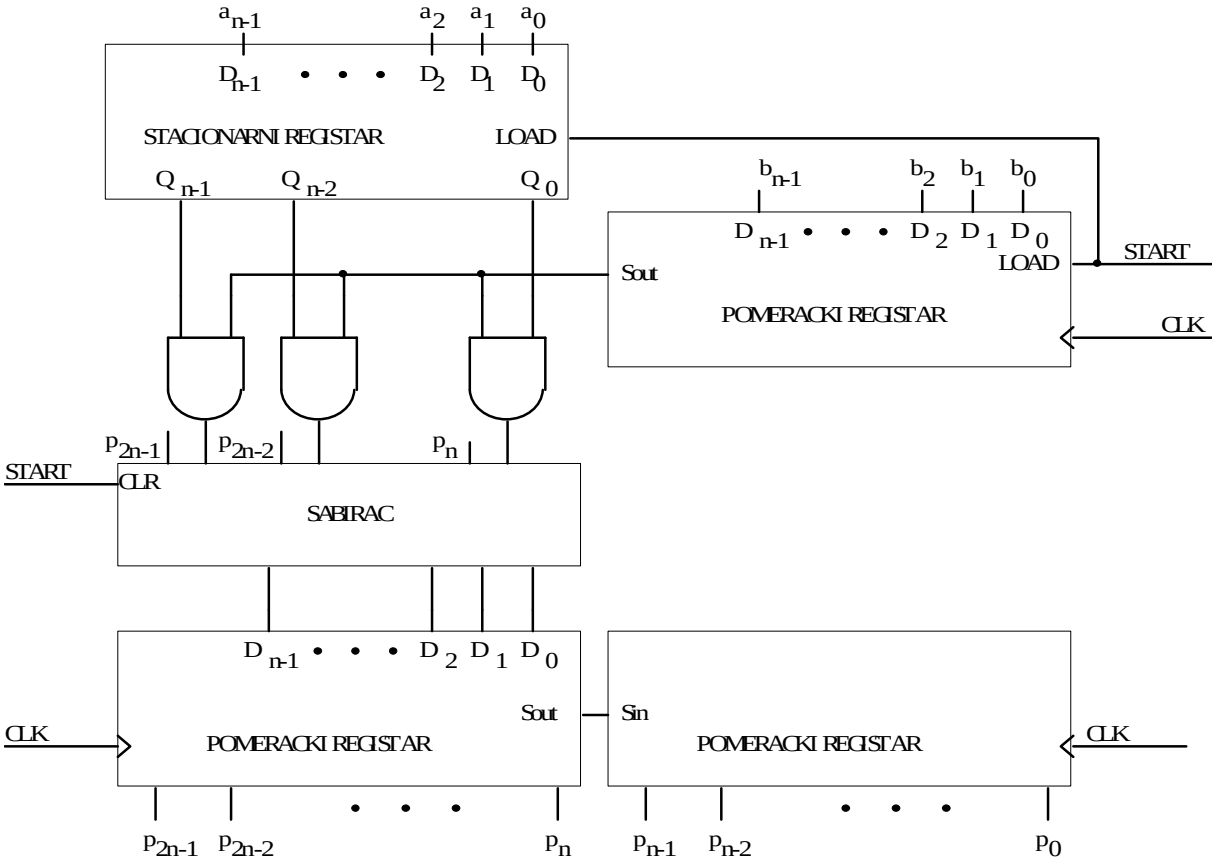
U *VLSI* tehnologiji se realizuju 16x16 bitni ili još veći množači u formi kombinacionih mreža kao sastavni delovi mikroprocesora.

Kod množača koji se realizuju sekvencijalnim mrežama potrebno je skrenuti pažnju na dva osnovna principa. Logička mreža koja je prikazana na slici 4-10 pod uticajem START signala briše sadržaj akumulatora, u stacionarni registar upisuje prvog množioca, zatim upisuje drugi množilac u programabilni brojač. Pod uticajem takt signala brojač broji unazad dok akumulacioni sabirač sabira svaki put aktuelni sadržaj akumulatora sa množiocem. Proces traje sve dok brojač ne stiže do nule. U trenutku kada brojač dostiže vrednost nula, zahvaljujući sukcesivnom sabiranju, u akumulatoru se nalazi traženi proizvod.



Slika 4-10: Množenje realizovano sukcesivnim sabiranjem.

Drugi osnovni princip se zasniva na proceduri ručnog množenja bit po bit (slika 4-11). Na početku procesa prvi množilac se smešta u stacionarni registar a drugi množilac u pomerački registar. Niz *I* kola vrši množenje sa pojedinim bitovima drugog množioca. Sabirač, posle pomeranja, sabira sopstveni sadržaj sa međurezultatom. Množenje je završeno kada je svaki bit izašao iz pomeračkog registra.



Slika 4-11: Sekvencijalna mreža koja realizuje množenje bit po bit.

4.2.3. Aritmetički komparatori

Upoređivanje dva binarna broja je zadatak koji se često javlja u digitalnim sistemima. Naziv kola koje vrši upoređivanje je (aritmetički) komparator. Rezultat komparacije je jednakost dva broja, ali operaciju komparacije je moguće proširiti i na određivanje relacije koji je broj veći ili manji u slučaju nejednakosti. U nastavku se razmatra samo ovaj, složeniji komparator koji se naziva univerzalni komparator.

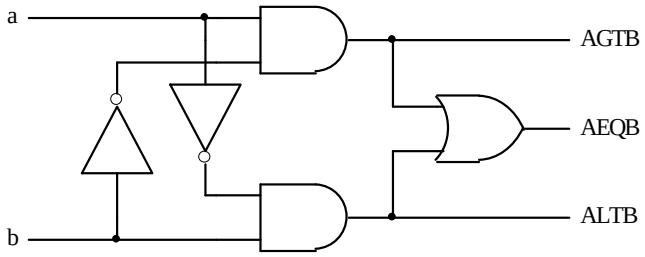
Komparacija dva jednobitna broja se vrši prema kombinacionoj tabeli koja je prikazana na slici 4-12a. Izlazne promenljive se formiraju prema sledećim jednačinama:

$$AGTB = a\bar{b}$$
$$AEQB = ab + \bar{a}\bar{b} = \overline{a \oplus b}$$
$$ALTB = \bar{a}b.$$

Vrednost premenljive *AGTB* (engl.: *greater then*) je jednak jedinici samo ako je $a > b$, *AEQB* (engl.: *equal*) je jedinica ako je $a = b$ i vrednost *ALTB* (engl.: *less then*) je jedinica ako je $a < b$. Logička šema univerzalnog komparatora koja je realizovana na osnovu prethodnih jednačina je data na slici 4-12b.

A	B	AGTB	AEQB	ALTB
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

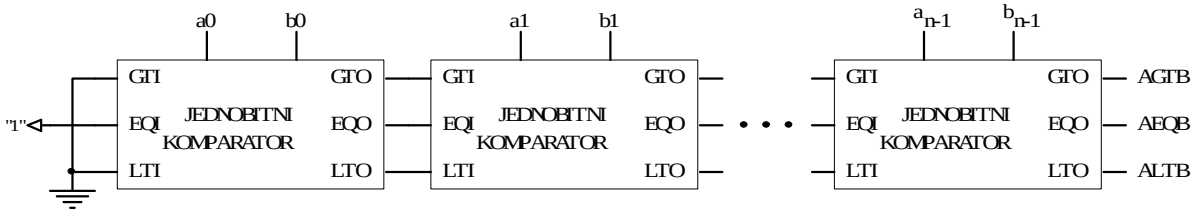
(a)



(b)

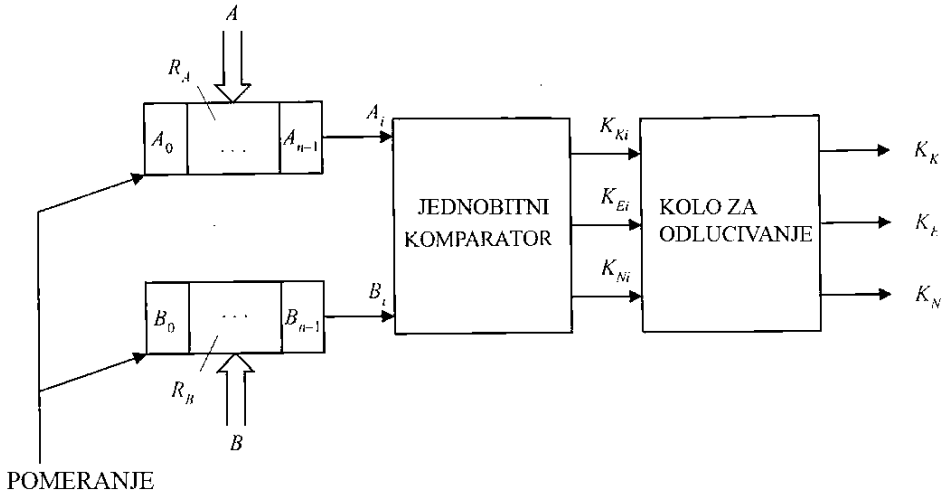
Slika 4-12: (a) Kombinaciona tabela jednobitnog univerzalnog komparatora i (b) njegova logička šema.

Krajnji cilj u vezi komparatora je projektovanje takvih kola koja vrše komparaciju višebitnih brojeva. Teoretski je moguće napisati kombinacionu tabelu za višebitne komparatore na način koji je prikazan na slici 4-12a i na osnovu toga izvršiti sintezu odgovarajuće mreže. Upoređivanje bit po bit je međutim mnogo svrsishodnije i zbog toga je neophodno izmeniti mrežu, koja je prikazana na slici 4-12b, tako da je moguće izvršiti kaskadno povezanje pojedinih stepena. Način kaskadnog povezanja jednobitnih komparatora je prikazan na slici 4-13. Smanjena je brzina komparacije ali i složenost kola.



Slika 4-13: Kaskadno povezivanje aritmetičkih komparatora.

Prema slici 4-14, upoređivanje se može vršiti serijskim postupkom primenom odgovarajuće sekvencijalne mreže. Vrednosti koje su upisane u registre se pomeraju korak po korak i dovode na ulaz jednobitnog komparatora. Upoređivanje je svrsishodno početi od bita najveće binarne težine. Čim se uoči razlika kod nekog bita, izlazi $A>B$ odnosno $A<B$ se mogu trenutno aktivirati. Sa druge strane, jednakost se može ustanoviti samo kada su već svi bitovi provereni.



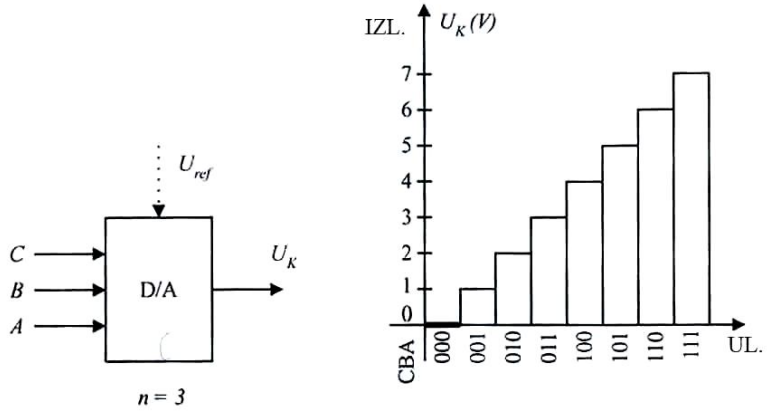
Slika 4-14: Logička šema serijskog aritmetičkog komparatora.

4.3. *D/A konvertori*

Kod rešavanja različitih tehničkih problema često je potrebno pretvoriti digitalne signale u analogne. Pri digitalno-analognoj (*D/A*) konverziji rezultat je napon koji je srazmeran datom broju.

4.3.1. Način funkcionisanja

D/A konvertor svakom ulaznom broju dodeljuje jedan napon diskretne vrednosti. Obično su moguće vrednosti napona linearno raspoređene po nekoj skali. Broj različitih vrednosti a time i rezolucija skale zavisi od broja kodnih reči u datom kodnom sistemu. Obično se koristi prirodni binarni kod i tako pomoću koda veličine *n* bita moguće je definisati 2^n različitih izlaznih vrednosti za napon. Šematska oznaka i prenosna karakteristika trobitnog *D/A* konvertora su prikazane na slici 4-15. Broj bitova u praksi je često mnogo veći npr. 8, 10, 12 ili još veći.

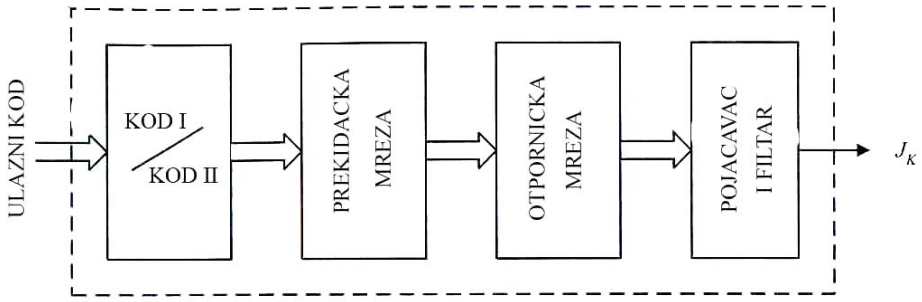


Slika 4-15: (a) Šematska oznaka D/A konvertora i (b) njegova prenosna karakteristika.

D/A konvertor generiše izlazni napon deljenjem iz odgovarajućeg osnovnog signala (referentnog napona). Uslov preciznog pretvaranja je tačan i stabilan osnovni signal a takođe i deljenje mora biti bez greške.

4.3.2. Struktura

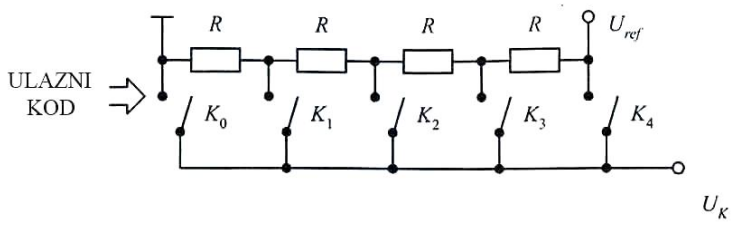
Centralni deo *D/A* konvertora je jedna otpornička mreža (slika 4-16) čija je uloga deljenje osnovnog signala. Odnos deljenja se podešava pomoću analognih prekidača. Upravljanje prekidačima mogu direktno da vrše bitovi ulaznog broja ali se može desiti da za dobijanje tačnog rezultata je neophodno izvršiti konverziju tog broja u neki drugi kodni sistem. Izlazni signal otporničke mreže je moguće direktno koristiti ali se obično pre toga propušta kroz stepene za pojačavanje i filtraciju.



Slika 4-16: Blok šema digitalno/analognog konvertora.

Slike od 4-17 do 4-20 prikazuju karakteristične rasporede otporničkih mreža i prekidača. Otpornička mreža kod takozvanog direktnog pretvarača (slika 4-17) se sastoji od redne veze

otpornika istih vrednosti. Svaka moguća vrednost diskretnog napona je na raspolaganju u tačkama spajanja dva otpornika. Od svih prekidača, istovremeno je samo jedan uključen, koji se određuje na osnovu ulazne kodne reči i tako se izabrana vrednost napona propušta na izlaz (U_K). Neophodo birati takav kodni sistem kod kojeg se u svakoj kodnoj reči nalazi samo jedna jedinica ili se ovaj problem rešava uvođenjem dekodera ili kodnog pretvarača. Struktura mreže je jednostavna ali se pri velikoj rezoluciji sastoji od velikog broja prekidača i otpornika velike tačnosti čija je realizacija skupa.

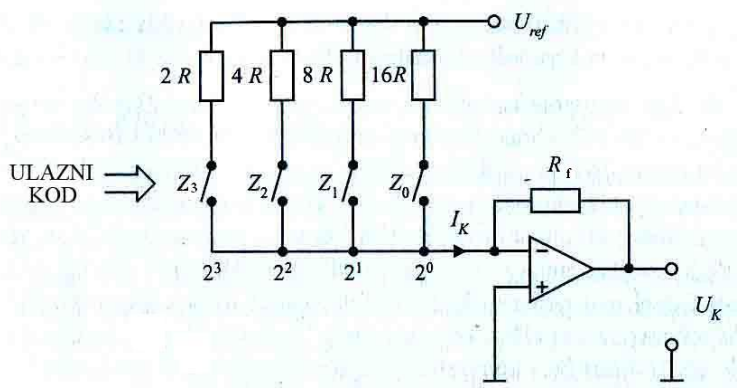


Slika 4-17: Principska šema direktnog D/A kovertora.

Šema četvorobitnog *D/A* konvertora sa težinskom otporničkom mrežom je prikazana na slici 4-18. Struje koje prolaze kroz pojedine otpornike se sabiraju pomoću operacionog pojačavača i na taj način se formira diskretni izlazni napon koji odgovara ulazoj kodnoj reči:

$$V_O = -R_f \cdot V_{REF} \cdot \frac{1}{2^n R}(2^0 Q_0 + 2^1 Q_1 + 2^2 Q_2 + ... + 2^{n-1} Q_{n-1})$$

Promenljive Q_i su vrednosti pojedinih bitova od ulazne kodne reči. Najveću teškoću kod ovog rešenja predstavlja realizacija težinskih otpornika sa tačnim vrednostima. U integrisanoj tehnici je posebno teško realizovati otpornike koji imaju veoma male ili veoma velike vrednosti otpornosti sa velikom tačnošću. Problem je i to da vrednosti struja koje prolaze kroz pojedine prekidače su veoma različite i zbog toga su i padovi napona na prekidačima različiti što doprinosi grešci u konverziji.

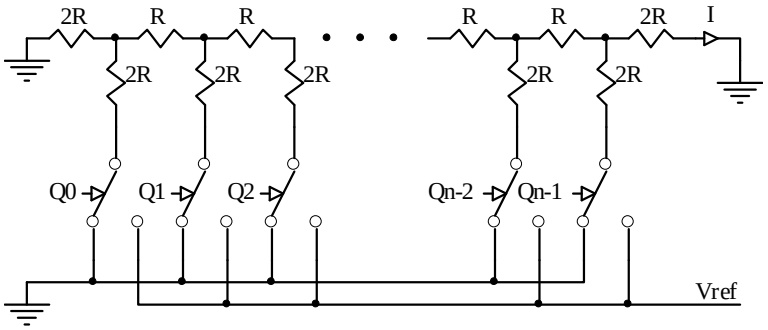


Slika 4-18: D/A konvertor realizovan težinskom otporničkom mrežom.

U integrisanoj tehnici najveći uspeh su doživeli oni *D/A* konvertori čija se struktura zasniva na lestvičastoj otporničkoj mreži jer je potrebno precizno reprodukovati samo dve vrednosti otpornosti *R* i 2*R* (slika 4-19). Povezivanjem određenih prekidača na referentni potencijal se formiraju pojedinačne komponente izlazne struje koja je na desnoj strani šeme označena sa *I*. Najveća komponenta struje se dobija uključenjem desnog prekidača i ta vrednost se prepolovljava kako se kreće ulevo. Konačna formula za struju *I* je:

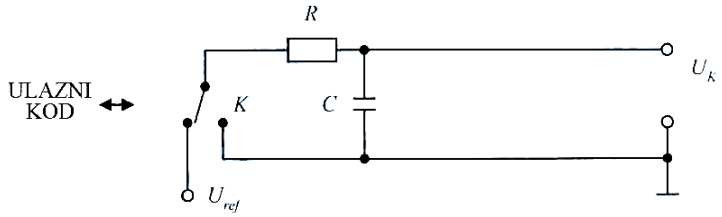
$$I = \frac{V_{REF}}{6R} \cdot \frac{1}{2^{n-1}} \cdot (2^{n-1} Q_{n-1} + 2^{n-2} Q_{n-2} + ... + 2^2 Q_2 + 2^1 Q_1 + 2^0 Q_0)$$

gde su Q_i bitovi ulazne kodne reči. Prema tome, veličina izlazne struje je i u ovom slučaju srazmerna ulaznoj kodnoj reči. Ovu struju je moguće direktno koristiti kao izlazni signal ili ako je potrebno, moguće ju je pretvoriti u napon pomoću jednog operacionog pojačavača.



Slika 4-19: D/A konvertor realizovan lestvičastom otporničkom mrežom R-2R.

Na slici 4-20 je prikazano rešenje za realizaciju *D/A* konvertora koje sadrži najmanji broj komponenata. Prekidač *K* se periodično prebacuje u levi i desni položaj. Upravljanje prekidačem se vrši tako da je odnos vremena za koje je prekidač u levom položaju i periode uključivanja identičan sa odnosom trenutne vrednosti ulaznog kodiranog broja i njegove maksimalne vrednosti (ovo se naziva impulsno-širinskom modulacijom). Posle odgovarajućeg filtriranja (koje vrši *RC* član) vrednost izlaznog napona (srednja vrednost impulsa) će biti proporcionalna vrednosti ulaznog broja.



Slika 4-20: D/A konvertor koji radi na principu impulsno-širinske modulacije.

4.3.3. Karakteristike

Glavne karakteristike *D/A* konvertora su: rezolucija i vreme uspostavljanja. Rezolucija se zadaje brojem bita pod pretpostavkom da se podaci na ulaz dovode u binarnom kodu. Podatak o rezoluciji ukazuje ujedno i na tačnost pretvaranja. Pretvarače je neophodno tako konstruisati da njihova prenosna funkcija bude monotona. Na osnovu prethodnih pretpostavki se zaključuje da je greška konverzije uvek manja od napona koji odgovara bitu najmanje težine.

Za uspostavljanje stabilnog izlaznog napona je potrebno određeno ovreme. Kod većine pretvarača ovo vreme je reda *μs* ili *ns*, dok kod impulsno-širinskih pretvarača je značajno veće jer je vremenska konstanta *RC* člana nekoliko puta veća od periode prekidanja koja je već sama po sebi prilično dugačka.

4.4. *A/D konvertori*

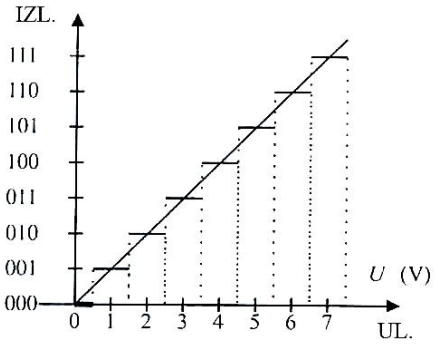
Kada je potrebno analogne signale (vrednost napona ili neke druge kontinualne veličine) u digitalnoj formi zapamtiti, obraditi, poslati, prikazati itd., neophodno je prethodno izvršiti analogno-digitalnu konverziju. Uzorku analognog signala se prilikom koverzije dodeljuje odgovarajući broj iz korišćenog kodnog sistema.

4.4.1. Princip rada

Kod *A/D* konverzije potrebno je rešiti nekoliko zadataka. Prvo, neophodno je u pravilnim vremenskim razmacima uzeti uzorke iz analognog signala. Ovaj postupak se naziva diskretizacijom po vremenu. Proces pretvaranja nije trenutan i zbog toga je neophodno tako odabrati periodu uzorkovanja da se konverzija prethodnog uzorka završi pre uzimanja narednog.

Drugi zadatak je upoređivanje uzorka sa vrednostima na jednoj diskretnoj skali. Kao rezultat upoređivanja, vrednost uzorka se zamenjuje jednim brojem sa ove skale koji je najbliži po vrednosti uzorku. Ovaj proces se naziva diskretizacija po amplitudi.

Zadnja faza konverzije je formiranje izlaznog koda koji odgovara diskretizovanoj vrednosti. Ovaj proces se obavlja prema unapred definisanoj prenosnoj karakteristici *A/D* konvertora (slika 4-21).

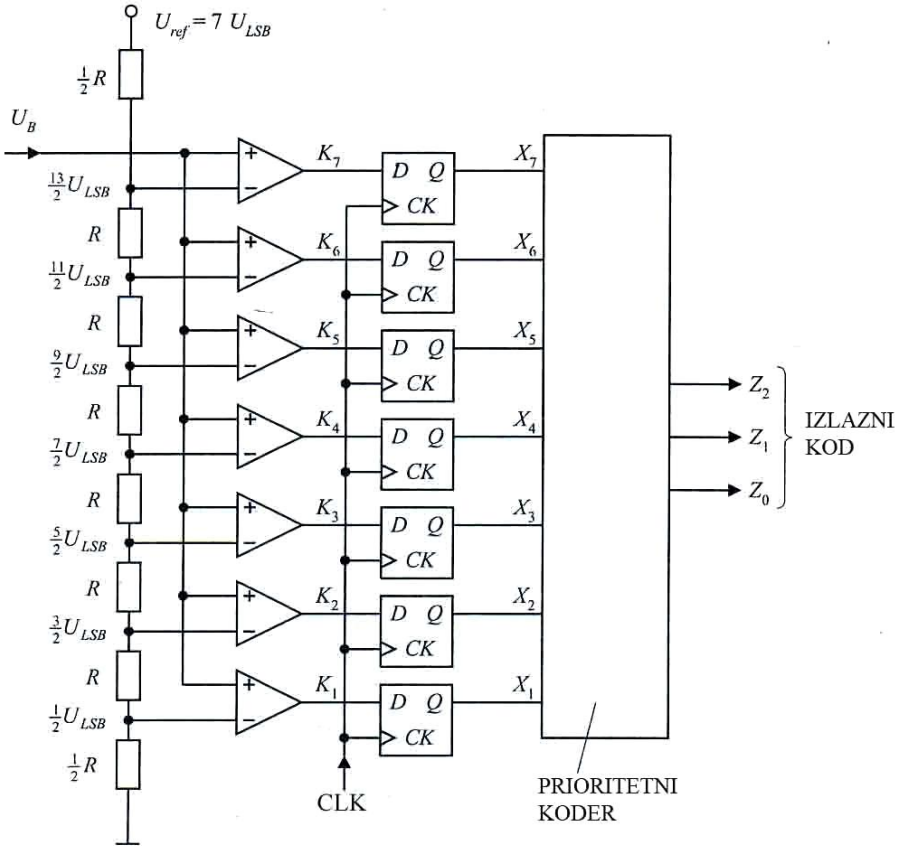


Slika 4-21: Prenosna karakteristika trobitnog A/D konvertora.

4.4.2. Struktura

Postoje tri različita rešenja za današnje *A/D* konvertore: direktni (engl.: *flash*) tip, rešenje sa postepenim približavanjem (sukcesivnom aproksimacijom) i sa postupkom brojanja.

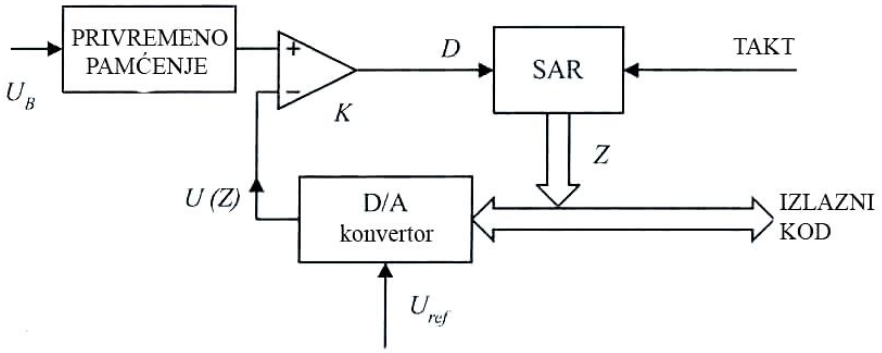
Direktni *D/A* konvertor u slučaju n-bitnih kodova sadrži 2^n-*1* naponskih komparatora (slika 4-22). Na jedan ulaz komparatora se dovode odgovarajući naponski nivoi prema datoj prenosnoj karakteristici. Ovi naponi se formiraju otporničkim razdelnikom povezanim na izvor referentnog napona. Na taj način se dobija željena skala diskretnih vrednosti. Na druge ulaze komparatora se dovodi analogni napon koji treba da se konvertuje.



Slika 4-22: Direktni A/D konvertor sa trobitnim izlazom.

U zavisnosti od vrednosti napona, izlazi nekih komparatora će biti na niskom nivou dok su drugi na visokom. Diskretizaciju po vremenu vrši takt signal: izlazne vrednosti komparatora periodično upisuje u *flip-flop*-ove. Izlazni stepen se sastoji od jednog prioritetnog kodera: izlazni kod se formira na osnovu jedinice na izlazu komparatora na najvišem položaju.

Kod *A/D* konvertora sa sukcesivnom aproksimacijom izlazni kod veličine *n* bita se formira u *n* koraka. Princip rada je prikazan na slici 4-23. Ulazni signal se uzorkuje na početku periode kovertovanja i ta vrednost se održava konstantnom na ulazu komparatora *K* do kraja ciklusa pretvaranja. Konvertor sadrži jedan *D/A* kovertor čija je rezolucija ista kao i samog *A/D* konvertora. Izlaz *D/A* konvertora je vezan na drugi ulaz komparatora. Izlaz komparatora vrši upravljanje registrom za sukcesivnu aproksimaciju (engl.: *successive approximation register – SAR*) koji je centralni deo šeme.



Slika 4-23: Blok šema A/D kovertora sa sukcesivnom aproksimacijom.

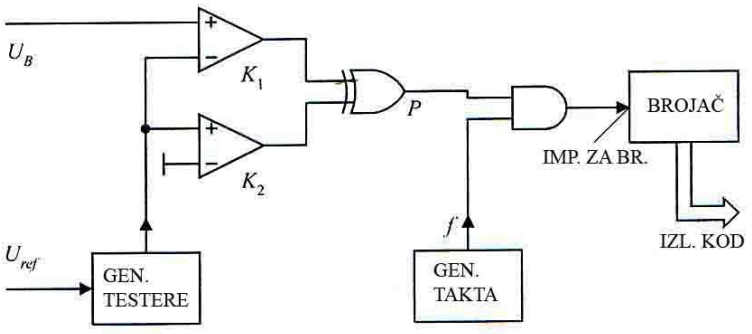
Na početku konverzije setuje se bit najveće težine (engl.: *most significant bit - MSB*) u registru. Prema toj vrednosti *D/A* konvertor generiše odgovarajući analogni signal a na osnovu toga komparator javlja registru da li je ulazni signal veći ili manji od vrednosti koja se nalazi na sredini skale.

Ako je ulazni signal veći, pri odgovarajućoj ivici takt signala, vrednost *MSB*-a ostaje nepromenjena dok se u suprotnom slučaju vraća na nulu. Istovremeno, setuje se vrednost susednog bita u registru. U sledećoj periodi takt signala se ponovo upoređuje vrednost ulaznog signala sa izlazom *D/A* konvertora i komparator informiše registar o opravdanosti setovanja datog bita. Ako je setovanje bilo opravdano, vrednost bita ostaje jedinica, ako ne, onda se resetuje. Na kraju periode takt signala opet se u registru setuje sledeći bit.

Ovaj proces se nastavlja sve dok nisu određene vrednosti svih bitova. Tada se u registru nalazi digitalni kod koji odgovara ulaznom analognom signalu. U prodaji se mogu naći integrisana kola koja u sebi sadrže kompletan *A/D* konvertor sa sukcesivnom aproksimacijom.

U treći skup rešenja A/D konverzije spadaju brojačka rešenja. Razvijeno je više takvih postupaka. Osnovni princip kod svih je generisanje pravougaonog impulsa sa vremenom trajanja srazmernim sa ulaznim analognim signalom. Pri tome uloga brojača je određivanje broja perioda takt signala generisanih za vreme trajanja pravougaonog impulsa. Broj dobijen na kraju brojanja će biti srazmeran analognom naponu.

Princip rada je prikazan na slici 4-24. Pravougani signal se generiše upoređivanjem analognog signala i linearno rastućeg testerastog signala. Naponski komparatori (pomoću *EX-ILI* kola) dozvoljavaju rad brojača sve dok je vrednost testerastog signala veća od nule ali još nije premašila ulazni napon.



Slika 4-24: Blok šema A/D konvertora koji koristi testerasti signal i brojač.

Nedostatak prikazanog rešenja je da osim strmine testerastog signala i tačnost frekvencije takt signala utiče na tačnost konverzije. Strminu testerastog signala je moguće stabilizovati pomoću odgovarajućeg referentnog napona, ali dodatni problemi mogu nastati zbog pomeranja *RC* vrednosti koji definišu testerasti signal. Postoje i druga brojačka rešenja koja primenjuju testerasti signal sačinjen od rastućeg i padajućeg segmenta. Pomoću ove tehnike je moguće izbeći osetljivost kako na promene *RC* člana tako i na stabilnost frekvencije takt signala. Jedino je važno kod ovog *A/D* konvertora da se vrednost referentnog napona održi na konstantnom nivou.

4.4.3. Karakteristike

Proces pretvaranja kod direktnih tipova *A/D* konvertora je veoma kratak (obično je vreme konverzije ispod 1μs), ali zahtevaju veliki broj komponenata i zbog toga im je cena visoka. Ovo rešenje se primenjuje do šest ili osam bita rezolucije i samo za konverziju brzo promenljivih signala (digitalni osciloskopi, telekomunikacije).

Prosečno vreme konverzije *A/D* konvertora sa sukcesivnom aproksimacijom je nekoliko μs. Rezolucija konverzije je obično osmobitna ili veća. Oblasti primene ovih kola su najčešće kod upravljanja brzim procesima.

Primenom *A/D* konvertora sa testerastim signalom i brojačem moguće je dostići veliku rezoluciju po niskoj ceni jer je broj korišćenih kola relativno mali. Nedostatak ovog tipa je dugačko vreme konverzije koje je često reda veličine sekunda. Kod digitalnih prenosnih instrumenata se uvek ugrađuju ovakvi tipovi konvertora jer korisnik ne bi ni bio u stanju da očita izmerenu vrednost neke veličine kada bi se cifre brzo menjale. Ovaj tip *A/D* konvertora je pogodan i za digitalizaciju sporopromenljivih fizičkih veličina (npr. temperatura, nivo tečnosti itd.).

III. Projektovanje primenom programabilnih logičkih kola (PLD)

Jezik za opis hardvera *Verilog*

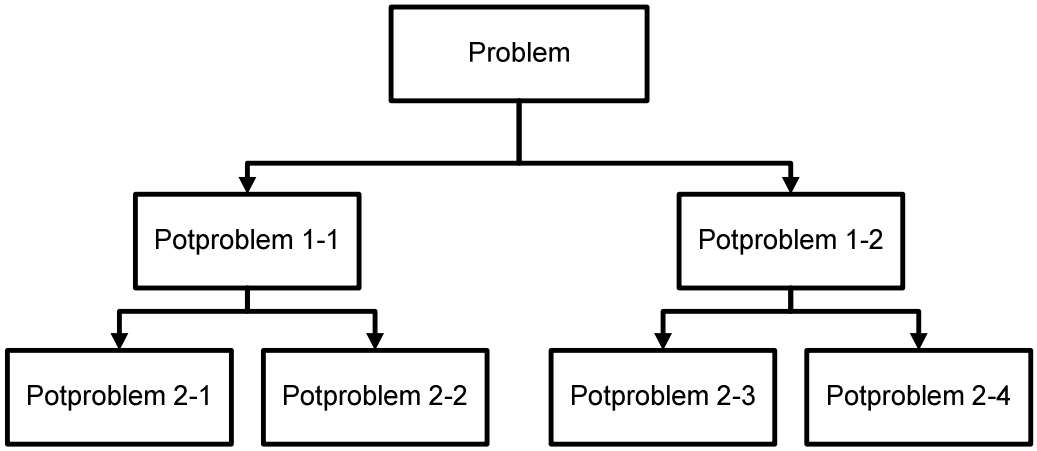
5. Pristupi projektovanju

Ovaj deo skripte je posvećen digitalnom projektovanju zasnovanom na jeziku za opis hardvera (*engl.: hadware description language – HDL*) koji nosi naziv *Verilog*. Pre upoznavanja sa jezičkim strukturama, prezentovaće se uobičajeni pristupi koji se primenjuju pri projektovanja digitalnih kola.

Postoje dva osnovna pristupa projektovanju:

- ***Top-down* pristup:**

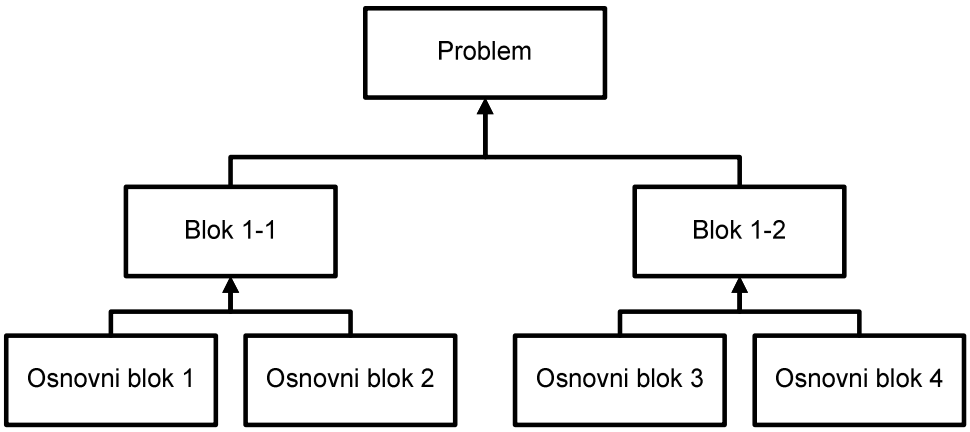
Definisani složeni problem se rešava deljenjem na više jednostavnijih podproblema koji se dalje dele na još jednostavnije podproblema itd. Proces se nastavlja sve dok podproblemi ne postanu dovoljno mali i jednostavni za rešavanje. Slika 5-1 pokazuje proces projektovanja *top-down* metodom.



Slika 5-1: Top-down pristup projektovanju.

- ***Bottom-up* pristup:**

Ova metoda projektovanja polazi od osnovnih blokova koji su na raspolaganju za rešavanje zadatog problema. Njihovim korišćenjem se grade blokovi veće kompleksnosti. Ovi blokovi se dalje koriste za dobijanje blokova još veće kompleksnosti itd. Proces se nastavlja sve dok se ne reši problem. Slika 5-2 pokazuje proces projektovanja *bottom-up* metodom.



Slika 5-2: Bottom-up pristup projektovanju.

Pri projktovanju kola u digitalnoj elektronici najčešće se koristi kombinacija ove dve metode. Projektanti električnih kola definisani problem dele na manje celine, koje posebno rešavaju. Te celine se ponovo dele na podceline sve dok se ne stiže do nivoa kada je moguće

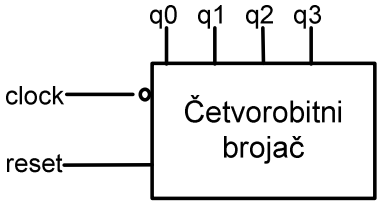
koristiti gotove blokove koji su unapred definisani i predstavljaju osnovne komponente u razvojnim okruženjima.

Sa druge strane, inženjeri koji projektuju komponente, polaze od nivoa prekidača, koji su osnovni elementi u datoj tehnologiji integrisanih kola. Na nivou prekidača prvo izgrađuju proste funkcionalne jedinice (npr. logička kola) zatim kombinacijom tih jedinica se formiraju složeniji blokovi koji će biti gradivni elementi razvojnog okruženja.

Metode koje su prezentovane podjednako se primenjuju kod projektovanja digitalnih kola sa *SSI*, *MSI* komponentama ili *PLD* kolima. Za ilustraciju navedenih pristupa projektovanju poslužićemo se primerom četvorobitnog brojača sa asinhronim reset signalom. Brojač treba projektovati tako da ima mogućnost asinhronog resetovanja i da se okida na silaznu ivicu takt impulsa.

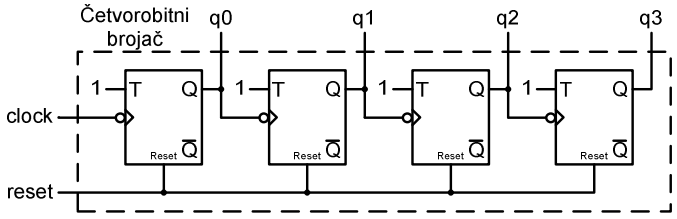
5.1. *Četvorobitni brojač*

Brojač će imati četiri izlaza (*q0, q1, q2 i q3*) i dva ulaza (*clock* i *reset*) kao što je prikazano na slici 5-3.



Slika 5-3: Šematski simbol četvorobitnog brojača.

Pretpostavimo da razvojno okruženje koje se koristi za realizaciju datog brojača kao osnovne gradivne elemente poseduje *D flip-flop* i sva potrebna logička kola. Projektovanje četvorobitnog brojača pomoću *D flip-flop*-ova i logičkih kola u jednom koraku bi predstavljalo težak zadatak. Međutim korišćenjem *T flip-flop*-ova se veoma lako projektuje zahtevani brojač. Slika 5-4 prikazuje šemu četvorobitnog brojača realizovog *T flip-flop*-ovima.



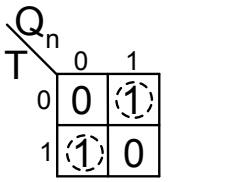
Slika 5-4: Ostvarivanje četvorobitnog brojača T flip-flop-ovima

Sledeći korak u rešavanju zadatka je sinteza *T flip-flop*-a pomoću *D flip-flop*-a i potrebnih logičkih kola. Kombinaciona tabela za sintezu *T flip-flop*-a pomoću *D flip-flop*-a je prikazana u tabeli 5-1. Slika 5-5 prikazuje minimizaciju logičke funkcije ulaza *D flip-flop*-a pomoću Karnaugh-ove tabele. Slika 5-6 predstavlja šematski prikaz realizacije *T flip-flop*-a pomoću jednog *D flip-flop*-a i potrebnih logičkih kola.

Tabela 5-1:Kombinaciona tabela za sintezu T flip-flop-a pomoću D flip-flop-a.

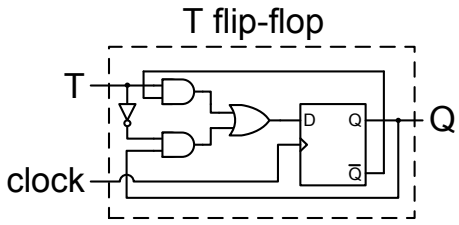
T	Q_n	Q_{n+1}	D
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Slika 5-5: Karnaugh-ova tabela za minimizaciju ulazne logičke funkcije D flip-flop-a.

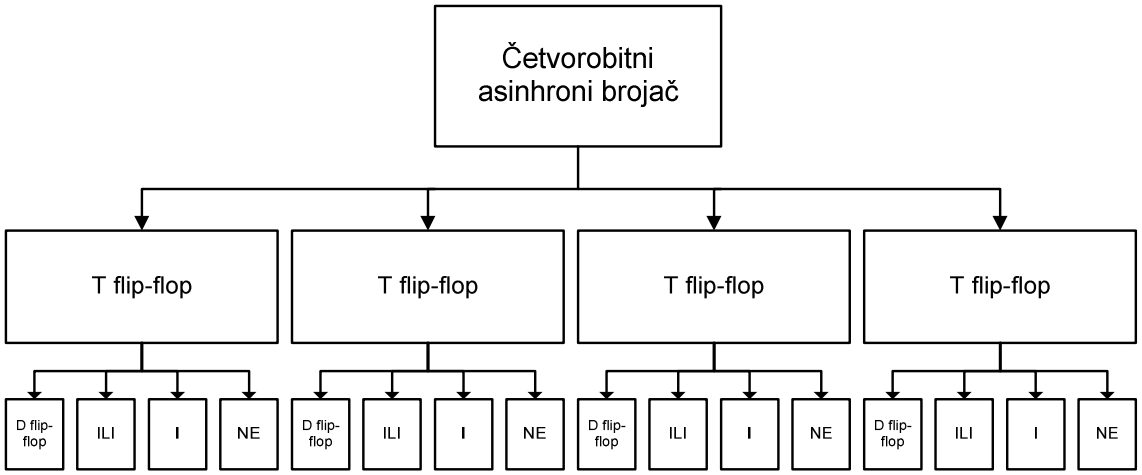


$$D = T \cdot \overline{Q} + \overline{T} \cdot Q$$

Slika 5-6: Logička šema ostvarivanja T flip-flop-a pomoću D flip-flop-a i potrebnih logičkih kola.



Za rešavanje ovog zadatka koristio se *top-down* pristup projektovanju. Dobijena hijerarhijska struktura je prikazana blok dijagramom na slici 5-7.



Slika 5-7: Hijerarhijska struktura dobijena pri projektovanju četvorobitnog brojača top-down metodom.

U procesu projektovanja četvorobitnog brojača *top-down* metodom, prvi korak je bio definisanje bloka najvišeg nivoa koji opisuje funkciju brojanja. Na sledećem nivou brojač je implementiran *T flip-flop*-ovima. Na trećem nivou *T flip-flop*-ovi su realizovani pomoću *D flip-flop*-ova i potrebnih *ILI*, *I* i *NE* logičkih kola. Na ovom nivou se dodiruju *top-down* i *bottom-up* metode jer su korišćeni *D flip-flop*-ovi i logička kola koja su osnovni gradivni elementi razvojnog okruženja nastali *bottom-up* pristupom projektovanju.

Umesto korišćenja *D flip-flop*-a kao osnovnog gradivnog elementa razvojnog okruženja, *D flip-flop* bi se mogao projektovati pomoću logičkih kola. Rezultat bi bio još jedan nivo u hijerarhiji *top-down* metode projektovanja. U tom slučaju dodirna tačka *top-down* i *bottom-up* metode projektovanja bi bila čisto na nivou logičkih kapija.

5.2. *Moduli u Verilog HDL-u*

U *Verilog HDL*-u je hijerarhijsko projektovanje omogućeno uvođenjem koncepta modula (engl.: *module*). Moduli su gradivni elementi u *Verilog HDL*-u i predstavljaju skup osnovnih gradivnih elemenata (engl.: *primitive*) i/ili gradivnih elemenata nižih nivoa.

Koncept modula omogućava da se skup elemenata koji sačinjavaju modul koristi na više mesta u projektu. Moduli se povezuju sa okolinom preko svojih *port*-ova (ulazi i izlazi). Korišćenjem *port*-ova unutrašnjost modula je sakrivena od okoline. Kada se jedan modul posmatra iz perspektive njegove okoline sve što se vidi to su njegove ulazne i izlazne linije (*port*-ovi). Ovo omogućuje promene unutar modula a da se pri tome ne mora menjati ostatak projekta. Druga prednost koju pruža korišćenje *port*-ova je mogućnost timskog rada na istom projektu.

Za timski rad je potrebno hijerarhijski organizovati projekat, opisati funkcije modula i definisati *port*-ove. Nakon toga projektovanje određenih modula ili grupe modula se dodeljuje pojedinim članovima tima.

Primeri modula sa slike 5-7 su četvorobitni brojač i *T flip-flop*.

Definicija modula u *Verilog HDL*-u počinje pomoću ključne reči *module* i završava se sa *endmodule*. Svaki modul mora posedovati jedinstveno ime kako bi se mogao razlikovati od ostalih modula. Takođe, svaka definicija modula mora sadržati listu *port*-ova koja opisuje ulazne i izlazne linije modula.

Oblik definicije modula u *Verilog HDL*-u je prikazan u primeru 5-1:

Primer 5-1: Najvažniji elementi definicije svakog modula.

```
module <ime_modula> (<lista_port-ova>);

< telo_modula>

endmodule
```

Centralni deo modula koji nosi naziv telo modula može da sadrži definicije sa četiri različita nivoa apstrakcije. Pojedini nivoi se primenjuju prema potrebama projekta. Ista funkcionalnost se može opisati korišćenjem sva četiri nivoa apstrakcije, pojedini nivoi se mogu i mešati.

Nivoi apstrakcije su sledeći:

- **Nivo ponašanja ili nivo algoritma (engl.: *behavioral* or *algorithmic level*)**:
  Predstavlja najviši nivo apstrakcije u *Verilog HDL*-u. Na ovom nivou je težište na određivanju algoritma koji rešava problem a ne na samoj hardverskoj realizaciji (koji će se elementi koristiti i kako će se oni povezati). Projektovanje je veoma slično programiranju u *C* jeziku.
- **Nivo toka podataka (engl.: *dataflow level*)**:
  Na ovom nivou se projektovanje vrši definisanjem toka podataka između registara i način njihove obrade u registrima.
- **Nivo kapija (engl.: *gate level*)**:
  Moduli se implementiraju korišćenjem logičkih kola i električnih veza kojima se kola povezuju.
- **Nivo prekidača (engl.: *switch level*)**:
  Predstavlja najniži nivo apstrakcije koji *Verilog HDL* podržava. Moduli se grade pomoću prekidača (tranzistorskih), memorijskih elemenata i veza kojima se ovi elementi međusobno povezuju.

Verilog HDL dozvoljava korišćenje svih nivoa apstrakcije unutar jednog projekta. Uobičajeno, izraz *RTL* (engl.: *register transfer level*) se odnosi na izvorni kod koji sadrži definicije i na nivou ponašanja i na nivou toka podataka.

Kada raste nivo apstrakcije povećavaju se fleksibilnost i tehnološka nezavisnost projekta, a kada se nivo apstrakcije približava nivou prekidača isto se smanjuje. U takvom slučaju mala promena u opisu zadatka može izazvati značajne promene u projektu. Može se povući paralela sa

programiranjem u *C* jeziku i u asembleru. Lakše je programirati u višim programskim jezicima kao što je jezik *C* nego u asembleru. Program napisan u *C* jeziku se lako prenosi na druge računare dok program napisan u asembleru je specifičan računaru na kojem je napisan i ne može se lako preneti na druge.

5.3. *Instance*

Modul je uzor na osnovu čega se kreira konkretan objekat. Kada se modul poziva, *Verilog HDL* kreira jedinstveni objekat na osnovu uzora. Svaki objekat poseduje jedinstveno ime, ulazno/izlazni interfejs, parametre i nosioce podataka. Proces kreiranja jedinstvenog objekta na osnovu uzora (modula) se naziva instanciranje (engl.: *instantiation*) a novonastali objekat je instanca (engl.: *instance*).

Četvorobitni brojač prikazan na slici 5-4 se sastoji od 4 instance *T flip-flop*-a. Svaki *T flip-flop* instancira jedan *D flip-flop*, dva *I* kola i po jedno *NE* i *ILI* kolo. Primer 5-2 sadrži definiciju modula 4-bitnog brojača. Svaka instanca mora posedovati jedinstveno ime. Dvostruka kosa crta (//) označava jednolinijski komentar. Komentari sadrže informacije za projektanta i nemaju uticaja na digitalni sistem koji se realizuje.

Primer 5-2: Instanciranje modula.

```
// Pretpostavimo da je modul T flip-flopa već definisan sa imenom T_FF.
module brojac(q, t, clock, reset);
output [3:0] q;                      // četvorobitni izlazni signal
input t, clock, reset;               // jednobitni ulazni signali
T_FF tff0(q[0], 1, clock, reset);    // Instanciranje modula T_FF sa imenom tff0.
T_FF tff1(q[1], 1, q[0], reset);     // Instanciranje modula T_FF sa imenom tff1.
T_FF tff2(q[2], 1, q[1], reset);     // Instanciranje modula T_FF sa imenom tff2.
T_FF tff3(q[3], 1, q[2], reset);     // Instanciranje modula T_FF sa imenom tff3.
endmodule
```

Verilog HDL ne dozvoljava definisanje modula unutar definicije drugog modula. Umesto toga se vrši instanciranje potrebnog modula.

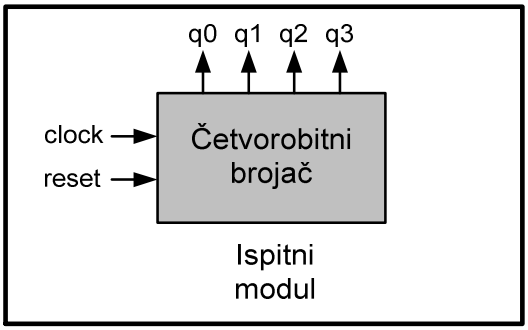
Pojmovi definicija modula i instanca modula se ne smeju mešati. U poređenju ovih pojmova sa procesom izgradnje zgrade, plan zgrade odgovara definiciji modula, proces izgradnje zgrade instanciranju a gotova zgrada instanci.

5.4. *Simulacija*

Pre nego što se primenom odgovarajućeg hardvera (*PLD*) realizuje projektovno kolo i pristupi testiranju kola realizovanog na bazi *HDL* opisa, prvo se vrši simulacija na računaru. Funkcionalnost *HDL* opisa se proverava primenom ulaznih signala i posmatranjem izlaznih signala. Treba imati na umu da je takvo ispitivanje dobro u onolikoj meri u kojoj meri pametno odaberemo ulazne (pobudne) signale. Matematičari rade na razvoju sistematskih postupaka za ispitivanje ispravnosti *HDL* opisa ali je zasad to u preliminarnoj fazi.

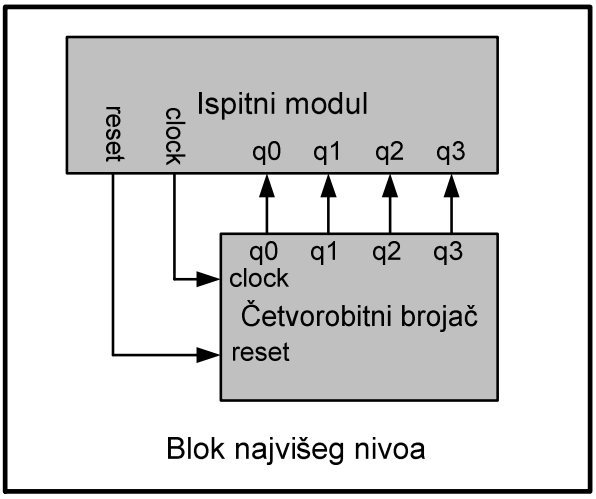
Blok koji obezbeđuje ulazne signale i prima izlazne signale se obično naziva ispitni blok (engl.: *stimulus block* ili *test bench*). Ispitni blok je takođe jedan modul napisan u jeziku za opis hardvera (*HDL*). Osnovno pravilo je da se ne sme pomešati ispitni blok sa projekom koji se realizuje. Postoje dva načina primene ispitnog bloka na objekat koji se testira:

- Ispitni blok instancira modul koji se testira i obezbeđuje ulazne i prima izlazne signale. Ispitni blok postaje *Verilog* modul najvišeg nivoa u hijerarhiji projektovanja. Primer koji ilustruje ovaj princip je prikazan na slici 5-8. Objekat testiranja je četvorobitni brojač prikazan na slici 5-3.



Slika 5-8: Ispitni blok (modul) instancira modul kojim se realizuje postavljeni zadatak.

- Drugi način primene ispitnog bloka je da se predmet testiranja i blok koji vrši testiranje instanciraju unutar jednog praznog modula najvišeg nivoa u hijerarhiji projektovanja (engl.: *dummy block*). Ispitni blok komunicira sa modulom testiranja kroz *port*-ove. Ovaj način primene ispitnog bloka je prikazan na slici 5-9. Jedina funkcija bloka najvišeg nivoa je instanciranje ispitnog modula i modula koji se testira.



Slika 5-9: Jedan prazan modul instancira ispitni modul i projektovani modul (četvorobitni brojač).

6. Osnovni koncepti *Verilog HDL*-a

Ova glava se bavi osnovnim pojmovima i konvencijama *Verilog HDL*-a. Izložena materija je prilično suvoparna ali je neophodna za razumevanje i praćenje narednih glava.

6.1. *Jezičke konvencije*

Osnovne jezičke konvencije *Verilog HDL*-a su veoma slične pravilima programskog jezika *C*. Verilog jezičke konstrukcije se sastoje od niza zapisa (engl.: *token*). Ti zapisi mogu biti sledećeg tipa: primedbe (komentari), znakovi za razdvajanje, brojevi, nizovi znakova (engl: *string*), identifikatori (engl.: *identifier*) i ključne reči (engl.: *keyword*). *Verilog HDL* je *case-sensitive* jezik, što znači da razlikuje mala i velika slova. Na primer, identifikatori *brojac* i *Brojac* se ne odnose na isti podatak jer prvi identifikator počinje sa malim slovom *b* a drugi sa velikim *B*. Sve ključne reči su definisane malim slovima u *Verilog HDL*-u.

6.1.1. Prazna mesta

Prazan karakter, tabulator i nova linija zajedničkim imenom se u *Verilog HDL*-u nazivaju praznim mestima. Prazni karakteri se u procesu prevođenja *Verilog HDL* opisa ignorišu osim kada se koriste za međusobno razdvajanje komentara, brojeva, identifikatora, stringova i ključnih reči. Kod *string*-ova uzimaju se u obzir i prazna mesta.

6.1.2. Komentari

Komentari su neophodni u softverskim jezicima radi dokumentovanja i lakšeg razumevanja programskog koda pri kasnijem analiziranju. Isto važi i za jezike za opis hardvera. Osnovno pravilo je da se stavi komentar svugde gde postoji sumnja da će jedna naredba ili grupa naredbi predstaviti problem za razumevanje posle nedelju ili mesec dana. Oni koji se ipak ustežu od pisanja komentara treba da imaju na umu da pisanje komentara ne umanjuje inteligenciju programera.

Postoje dva načina pisanja komentara (primer 6-1):

- **Jednolinijski komentari**. Znaci „//“ nalažu prevodiocu opisa na hardverskom jeziku da sve što sledi od te tačke pa do kraja reda preskače u procesu prevođenja.
- **Višelinijski komentari**. Višelinijski komentar počinje znacima „**/***“ i završava se sa znacima „***/**“.

Primer 6-1: Pisanje jednolinijskih i višelinijskih komentara.

```
a = b + c;              // Ovo je jednolinijski komentar

/*Ovo je višelinijski
 komentar*/
```

6.1.3. Operatori

Operatore delimo u tri kategorije: imamo unarne, binarne i ternarne operatore (primer 6-2). Unarni operatori se odnose na jedan operand i stavljaju se ispred operanada, binarni operatori se stavljaju između dva operanda. Ternarni operatori se sastoje od dva posebna znaka koji razdvajaju tri operanada.

Primer 6-2: Unarni, binarni i ternarni operatori.

```
a = ~b;                 // ~ je unarni operator. b je operand.
a = b && c;             // && je binarni operator. b i c su operandi.
a = b ? c : d;          // ?: je ternarni operator. b, c i d su operandi.
```

6.1.4. Predstavljanje brojeva u *Verilog HDL*-u

U *Verilog HDL*-u brojevi se mogu navoditi na dva načina: sa veličinom broja ili bez veličine.

- **Brojevi sa veličinom (engl.: s*ized numbers*)**:

  Brojevi sa veličinom su sledećeg oblika:

  <veličina>'<osnova_brojnog_sistema><broj>

<veličina> je decimalni broj koji predstavlja broj bitova pomoću kojih se zapisuje dati broj.

Brojni sistemi koje *Verilog HDL* podržava su: decimalni sistem (oznaka je 'd ili 'D), binarni sistem (oznaka je 'b ili 'B), oktalni sistem (oznaka je 'o ili 'O) i heksadecimalni sistem (oznaka je 'h ili 'H).

<**Broj**> se zadaje ciframa *0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f* ili podskupom ovih cifara koji odgovara brojnom sistemu koji se koristi. Cifre se mogu pisati i malim i velikim slovima (*a, b, c, d, e, f* ili *A, B, C, D, E, F*)(primer 6-3).

Primer 6-3: Pisanje brojeva sa veličinom.

```
3'b101          // trobitni binaran broj
16'hfa3e        // 16-bitni heksadecimalan broj
16'HFA3e        // Isti broj kao prethodni samo su neka slova velika
```

- **Brojevi bez veličine (engl.: *unsized numbers*):**

Veličinu brojeva u bitovima koji su zadati bez parametra <**veličina**> određuje simulator ili arhitektura računara (ali dužina ne može biti manja od 32 bita).

Broj koji nema parametar <**osnova_brojnog_sistema**> je po dogovoru uvek decimalan broj (primer 6-4).

Primer 6-4: Definisanje brojeva bez veličine.

```
'o7651          // 32-bitni oktalni broj
'hAB3           // 32-bitni heksadecimalan broj
4556            // 32-bitni decimalan broj
```

- ***x* i *z* vrednosti među ciframa**

  Verilog HDL poseduje dva specifična simbola: jedno za nepoznato stanje (*x*) a drugo za stanje visoke impedanse (*z*) (primer 6-5). Ovi simboli su neophodni pri modeliranju rada realnih kola.

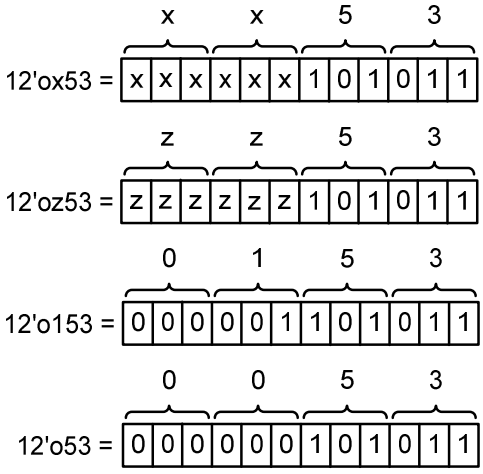
Primer 6-5: Korišćenje x i z vrednosti u ciframa broja.

```
12'h13x         // 12-bitni heksadecimalni broj. Vrednosti četiri
                // najmanje značajna bita su nepoznate.
6'hx            // 6-bitni heksadecimalni broj nepoznate vrednosti.
32'bz           // 32-bitni binarnan broj. Sve binarne pozicije su u
                // stanju visoke impedanse.
```

Vrednosti x i z u slučaju heksadecimalnog brojnog sistema zamenjuju 4 bita, u slučaju oktalnog 3 bita a u slučaju binarnog brojnog sistema jedan bit.

Važno je objasniti šta se dešava u slučaju kada je broj zadatih cifara manji od veličine broja. Ako je cifra najveće težine zadatog broja *0, x* ili *z*, neodređeni bitovi veće težine se takođe dopunjuju sa vrednostima *0, x* ili *z*. Ovo omogućuje jednostavan način dodeljivanja vrednosti *0, x* ili *z* celom vektoru. Ako je cifra najveće težine 1, neodređeni bitovi se dopunjuju sa 0.

Slika 6-1 prikazuje slučajeve kada je broj definisanih cifara manji od veličine broja. Brojevi su u oktalnom brojnom sistemu.



Slika 6-1: Primeri brojeva kod kojih je broj zadatih cifara manji od definisane veličine broja.

- **Pisanje negativnih brojeva**

Negativni brojevi se zadaju stavljanjem znaka minus pre parametra za veličinu broja. Nije dozvoljeno staviti znak minus između parametra za brojni sistem i samog broja (primer 6-6).

Pri obradi *Verilog HDL* izvornog koda (simulacija, sinteza) za predstavljanje negativnih brojeva softveri koriste drugi komplement.

Primer 6-6: Zadavanje negativnih brojeva.

```
-8'hA1          // 8-bitni negativan broj
8'h-A1          // Pogrešno zadavanje negativnog broja
```

- **Znak donja crta**

Znak donja crta „_“ je dozvoljen pri zadavanju brojeva na bilo kojem mestu osim ispred prvog karaktera. Jedini razlog zašto je dozvoljena donja crta jeste da poboljša preglednost *HDL* opisa (primer 6-7). Znak donja crta u brojevima se ignoriše u procesu tumačenja *HDL* opisa.

Primer 6-7: Korišćenje znaka donja crta.

```
12'b1100_0101_1111    // Znak donja crta se koristi kako bi se povećala preglednost
                      // HDL opisa
```

- **Znak pitanja**

Kada se radi o brojevima, znak pitanja „?“ (primer 6-8) predstavlja alternativu za stanje visoke impedanse (*z*). Znak pitanja se još koristi u izrazima **casex** i **casez** kako bi se povećala preglednost *HDL* opisa. Ove dve naredbe ćemo kasnije detaljno objasniti.

Primer 6-8: Označavanje stanja visoke impedanse znakom pitanja.

```
4'b01??         // Identičan izrazu: 4'b01zz
```

6.1.5. Nizovi znakova

Nizovi znakova (engl.: *string*) su skupovi karaktera koji se nalazi između dva navodnika. Jedino ograničenje pri zadavanju *string*-ova je da se mora nalaziti u jednoj liniji, ne može se prostirati u više linija (primer 6-9).

Primer 6-9: Definisanje stringa.

```
"Ovo je jedan string."
```

6.1.6. Identifikatori

Identifikatori (engl.: *identifier*) su imena objekata u *Verilog HDL*-u koja omogućavaju jednoznačno pozivanje na njih. Identifikatori se grade pomoću slova, brojeva, znaka donje crte i znaka dolara (*$*). *Verilog HDL* razlikuje mala i velika slova u identifikatorima. Identifikatori se ne mogu početi brojevima i znakom dolara. (Znak dolara kao prvi karakter u identifikatoru je rezervisan za sistemske funkcije.)

6.1.7. Ključne reči

Ključne reči (engl.: *keyword*) su specijalni identifikatori rezervisani za definisanje jezičkih konstrukcija. Sve ključne reči se obavezno pišu malim slovima (primer 6-10).

Primer 6-10: Primena ključnih reči.

```
reg broj;          // reg je ključna reč; broj je identifikator
input Input;       // input je ključna reč; Input je identifikator
                   // input i Input nisu isti jer Verilog HDL razlikuje velika i
                   // mala slova.
```

6.2. *Tipovi podataka i tipovi nosioca podataka*

Ovo poglavlje je posvećeno tipovima podataka koji se koriste u jeziku *Verilog*. Uvode se ključne reči koje se koriste za njihovu definiciju i pravila vezana za pojedine tipove nosioca podataka.

6.2.1. Logičke vrednosti u *Verilog HDL*-u

Kao što je navedeno kod objašnjenja načina pisanja brojnih vrednosti (tačka 6.4.1) *Verilog HDL* podržava četiri logičke vrednosti koje su prikazane u tabeli 6-1. To su moguće vrednosti za sve vrste podataka.

Tabela 6-1 : Logičke vrednosti koje Verilog HDL podržava.

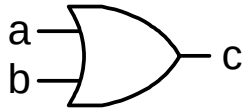
Vrednosti	*Stanja*
0	*Logička nula*
1	*Logička jedinica*
x	*Nepoznato stanje*
z	*Stanje visoke impedanse*

6.2.2. Čvorovi, veze

Osnovni tip podatka u *Verilog HDL*-u je logičko stanje veze odnosno čvora (engl.: *net*) koje služi za povezivanje hardverskih elemenata. Kao i u realnim kolima, logička vrednost *net*-a (veze) je određena izlazom hardverskog elementa koji je povezan na taj *net*.

Na slici 6-2 izlaz *ILI* logičkog kola je povezan na nosilac podatka *c*, tipa *net*. Logički nivo podatka *c* na izlazu kola u svakom trenutku je određena vrednošću izlaza *ILI* kola.

Slika 6-2: Primer za nosioce podatka tipa net.



Nosioci podataka tipa *net* se najčešće deklarišu pomoću ključne reči ***wire*** (primer 6-11). Podaci su redovno veličine jednog bita osim kada se deklarišu kao vektor. U literaturi pojam ***wire*** se često koristi umesto *net*. Po dogovoru, vrednost nekog podatka tipa ***wire*** koji nije povezan na izlaz nekog elementa je jednaka stanju visoke impedanse. Ako je dati nosilac podatka povezana sa izlazom nekog elementa, njegova vrednost postaje logička vrednost izlaza datog elementa.

Primer 6-11: Deklaracija podataka tipa net pomoću ključne reči wire.

```
wire c;          // Deklariše se podatak c, tipa veza koja je
                 // povezana sa izlazom ILI kola. (Slika 6-2)
wire a, b;       // Deklarišu se podaci a i b, tipa veza koje su
                 // povezane na ulaz ILI kola. (Slika 6-2)
wire d = 1'b0;   // Podatak d je tipa veza i ima konstantnu
                 // vrednost nula.
```

Kada je vrednost nekog podatka konstantna kao u slučaju podatka *d*, nije ga preporučljivo povezati sa izlazom nekog digitalnog kola. Kada vrednost izlaza datog elementa postaje jedan, dolazi do kolizije logičkih vrednosti jer vrednost podatka *d* tipa veza treba da je nula.

Potrebno je napomenuti da veza (*net*) nije ključna reč i predstavlja jednu celu klasu tipa podataka kao što su ***wire***, ***wand***, ***wor***, ***tri***, ***triand***, ***trior***, ***trireg*** itd. Najčešće se koristi tip ***wire***. Ostali tipovi podataka se ređe koriste.

6.2.3. Registri

Registri su elementi koji služe za pamćenje logičkih vrednosti. Registri pamte svoje sadržaje sve dok se ne zamene novim vrednostima.

Ne treba mešati pojam registra u *Verilog HDL*-u sa hardverskim registrima sastavljenim od *flip-flop*-ova koji dobijaju nove vrednosti na rastućoj ili na opadajućoj ivici takt signala. Pojam registra u *Verilog HDL*-u označava nosilac podatka koji pamti vrednost koja je njemu dodeljena u nekom ranijem momentu. Za razliku od veze (*net*), registru nije potreban drajver (izlaz nekog logičkog kola) da bi imao vrednost različitu od stanja visoke impedanse. Nosiocima podataka tipa registar u *Verilog HDL*-u nisu potrebni takt signali koji sinhronišu promenu stanja. Sadržaj registra u *Verilog HDL*-u se može promeniti u bilo kom trenutku putem odgovarajuće dodele.

Nosioc podatka tipa registar se deklarišu pomoću ključne reči ***reg***. Podrazumevana vrednost podatka tipa ***reg*** je neodređeno stanje (tj. *x*), ta vrednost važi sve dok se registru ne dodeli neka konkretna vednost. Primer 6-12 prikazuje način deklarisanja nosioca podatka tipa ***reg***:

Primer 6-12: Deklaracija nosioca podataka tipa registar pomoću ključne reči reg.

```
reg reset;              // Nosilac podatka reset pamti dodeljene vrednosti.
initial                 // Jezička konstrukcija koja će biti objašnjena kasnije.
begin
   reset = 1'b1;        // Pomoću reset signala se resetuje neko digitalno kolo.
   #100 reset = 1'b0;   // Posle 100 vremenskih jedinica reset signal se deaktivira.
end
```

6.2.4. Vektori

Nosioci podataka tipa *net* i ***reg*** se mogu deklarisati kao vektorske veličine (širina je veća od jednog bita). Ako nije navedena širina podatka u bitovima, podrazumevana vrednost je jedan bit. U tom slučaju podatka je skalarna veličina (primer 6-13).

Primer 6-13: Deklaracija vektora.

```
wire izlaz;                 // izlaz je skalarni nosilac podatka tipa veza (net).
wire [7..0] led_indikator;  // 8-bitni vektorski nosilac podatka
wire [15..0] ledA, ledB;    // dva 16-bitna vektorska nosioca podataka
reg takt_signal;            // skalarni nosilac podatka
reg [0..255] mem;           // 256-bitni vektorski nosilac podatka
```

Vektori se deklarišu kao *[veći_broj : manji_broj]* ili *[manji_broj : veći_broj]*. Levi broj u uglastoj zagradi uvek predstavlja bit najveće težine u vektoru. U gornjem primeru za nosilac podatka *ledB* najznačajniji bit je bit *15*, dok za podatak *mem* najznačajniji biti je bit *0*.

Moguće je adresirati jedan bit ili grupu bitova unutar vektorskog podatka. Ova mogućnost je prikazana u primeru 6-14 nad vektorskim poacima definisanim u prethodnom primeru.

Primer 6-14: Adresiranje pojedinačnih ili grupe bitova iz nekog vektora.

```
ledB [7];          // Označava sedmi bit podatka ledB
ledA[1:0];         // Dva najmanje značajna bita podatka ledA
                   // Sintaksa ledA[0:1] je nelegalna jer najznačajniji bit
                   // se uvek mora nalaziti na levoj strani opsega.
mem [0:1];         // Dva najviše značajna bita podatka mem.
```

6.2.5. Celi brojevi

Kao nosioc podatka za ceo broj (engl.: *integer*) se koristi nosilac opšte namene registarskog tipa. Za njihovu deklaraciju se koristi ključna reč ***integer*** (primer 6-15). Za cele brojeve moguće je koristiti i deklaraciju ***reg*** ali je uobičajeno da se podatak tipa ***integer*** koristi za potrebe brojanja. Podrazumevana širina podatka ***integer*** zavisi od širine reči računara na kome se vrši simulacija, minimalna širina je *32* bita. Nosilac podatka tipa ***reg*** pamti podatke kao pozitivne cele brojeve, dok nosilac podatka tipa ***integer*** može da pamti i negativne brojeve.

Primer 6-15: Deklaracija podataka tipa integer.

```
integer brojac;          // Nosilac podatka brojac je tipa integer.
initial
    brojac = -1;         // Početna vrednost podatka brojac je -1.
```

6.2.6. Realni brojevi

Realni brojevi su takođe podaci registarskog tipa u *Verilog HDL*-u. Deklaracija se vrši pomoću ključne reči ***real***. Vrednosti podatka ***real*** se mogu zadati u decimalnom obliku (tj. *3.14*) ili u eksponencijalnom obliku (tj. *3e2 = 300*) (primer 6-16). Treba obratiti pažnju da se za odvajanje razlomljenog dela broja u *Verilog HDL*-u se koristi tačka umesto zareza.

Veličinu nosioca podatka tipa ***real*** zavisi od računara na kome se vrši simulacija. Podrazumevana vrednost podatka ***real*** pri deklarisanju je *0*. U slučaju dodele vrednosti nosiocu podatka ***real*** nosiocu ***integer*** vrši se zaokruživanje na najbliži celi broj.

Primer 6-16: Deklaracija i korišćenje nosica podataka tipa real.

```
real alfa;              // Podatak alfa je tipa real.
initial
begin
    alfa = 4e10;        // Nosiocu podatka alfa se dodeljuje broj u eksponencijalnom
                        // obliku.
    alfa = 5.31;
end
integer i;              // Podatak i je definisan kao integer.
    i = alfa;           // Nosioc podatka i poprima vrednost 5 ( 5.31 je zaokružen
                        // na najbliži ceo broj).
```

6.2.7. Nizovi

Verilog HDL podržava građenje jednodimenzionalnih nizova (engl.: *array*) od tipova podataka ***reg***, ***integer*** i vektora registara. Nizovi se ne mogu praviti od podataka tipa ***real*** i ne mogu se praviti višedimenzionalni nizovi. Deklaracija nizova se vrši u sledećoj formi: *<ime_niza>[veći_broj : manji_broj]* ili *<ime_niza>[manji_broj : veći_broj]* (primer 6-17).

Primer 6-17: Deklaracija nizova.

```
integer brojevi[0:7];           // Niz od 8 podataka tipa integer.
reg mem[31:0];                  // Niz od 32 1-bitnih podataka tipa reg.
reg [3:0] port [0:7];           // Niz od 8 port-ova. Svaki port je 4-bitni.
integer matrix [4:0] [4:0];     // Ilegalna deklaracija. Verilog HDL ne podržava
                                // višedimenzionalne nizove.
brojevi[4];                     // Označava četvrti element niza brojevi.
port[5];                        // Peti element niza port.
```

Potrebno je napomenuti da se ne smeju mešati pojmovi vektor i niz. Vektor je jedan podatak širok *n*-bita, dok je niz skup više elemenata širokih *1*-bit ili *n*-bita.

6.2.8. Memorije

U savremenom digitalnom projektovanju pomoću *PLD*-ova često je potrebno ostvariti registre i memorijske blokove, (RAM, ROM). Memorije se u *Verilog HDL*-u delarišu kao niz registara (primer 6-18). Elementi memorije se nazivaju rečima (engl.: *words*). Reči mogu biti široki 1-bit ili n-bita. Potrebno je razlikovati n 1-bitnih registara od jednog n-bitnog registra.

Primer 6-18: Deklaracija memorije kao niza registara.

```
reg mem1bit[0:1023];            // Memorija mem1bit je skup 1024 1-bitnih reči
reg [7:0] membyte [0:1023];     // Memorija membyte je skup 1024
                                // 8-bitnih reči (bajta).
membyte[521];                   // Učitava se reč veličine jednog bajta
                                // čija je adresa 521 u nizu membyte.
```

6.2.9. Parametri

Verilog HDL omogućava definisanje konstanti unutar modula pomoću ključne reči ***parameter*** (primer 6-19). Parametri se ne mogu koristiti kao podaci.

Primer 6-19: Deklarisanje parametara.

```
parameter broj_dioda = 5;      // broj_dioda je deklarisan kao konstanta
                               // sa vrednošću 5.
```

Poželjno je koristiti parametre u definiciji *Verilog HDL* modula. Potrebno je izbegavati brojčane vrednosti jer njihovo korišćenje povećava verovatnoću greške u *HDL* opisu. Korišćenje parametara povećava preglednost i fleksibilinost *HDL* opisa i olakšava njegovu izmenu.

6.2.10. Sistemske funkcije *$stop* i *$finish*

Sistemska funkcija ***$stop*** privremeno zaustavlja izvršavanje simulaciju koja je definisana u *HDL* opisu kako bi se omogućilo pregledanje vrednosti onih signala koji su od interesa. Sintaksa ***$stop*** funkcije je prikazana u primeru 6-20.

Primer 6-20: Sistemska funkcija $stop.

```
$stop;
```

Sistemska funkcija ***$finish*** prekida izvršavanje simulacije (primer 6-21).

Primer 6-21: Sistemska funkcija $finish.

```
$finish;
```

6.2.11. Naredba *'define*

Direktiva ***'define*** se koristi za definisanje tekstualnih *makro*-a koje će program prevodilac tumačiti na odgovarajući način. Način definisanja tekstualnih makroa je prikazan u primeru 6-22.

Primer 6-22: Definisanje tekstualnih makroa.

```
'<ime_makroa> <tekst_makroa>
```

Pri definisanju i korišćenju makroa uvek je prisutan znak '. Svrha tekstualnih *makro*-a je da ubrza pisanje *HDL* opisa (dugački tekstovi se mogu zameniti kratkim imenima *makro*-a) i da poveća njegovu preglednost. Ovo se postiže tako što prevodilac zamenjuje ime_makroa sa tekst_makroa u celom *HDL* opisu (primer 6-23).

Primer 6-23: Korišćenje makroa.

```
'define S $stop;          // Simboli 'S se zamenjuju svugde u programu sa $stop.

'define WORD_REG reg [31:0]

                          // Sada se može definisati 32-bitni registar u programu kao:
'WORD_REG reg32;          // Nosilac podatka reg32 je 32-bitni registar.
```

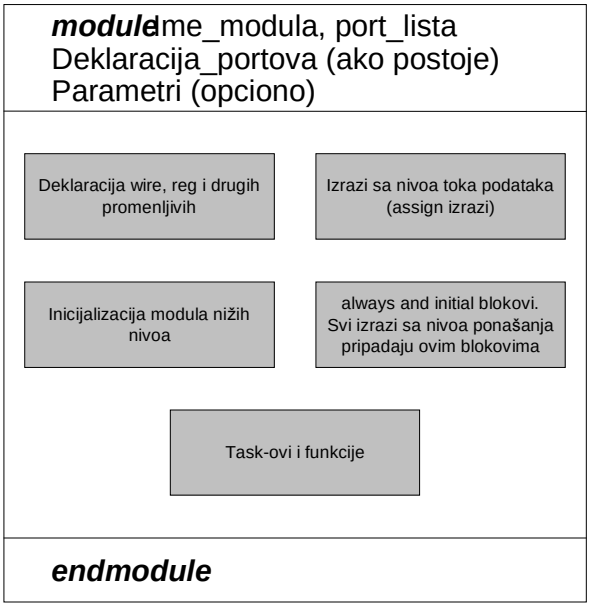
7. Moduli i *port*-ovi

U prethodnim glavama su predstavljeni koncepti hijerarhijskog projektovanja, osnovne jezičke konvencije hardverskog jezika *Verilog* i raspoloživi tipovi podataka odnosno nosioci podataka. Ova glava detaljno obrađuje ranije uvedeni pojam modula i daje opis i pravila u vezi *port*-ova.

7.1. *Moduli*

U glavi 5 je rečeno da je modul gradivni element koji čini osnovu hijerarhijskog projektovanja u jeziku *Verilog*. Akcenat je bio na definisanju i instanciranju modula a ne na unutrašnjoj strukturi modula. Ova glava detaljno obrađuje unutrašnjost modula.

Modul je tekst fajl formiran od *ASCII* karaktera. Sastoji od nekoliko različitih delova kao što je prikazano na slici 7-1.



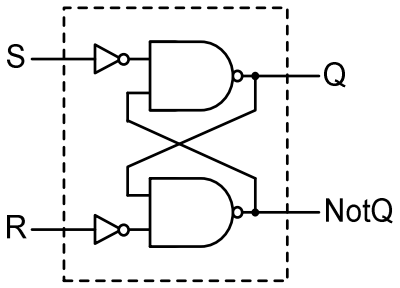
Slika 7-1: Sastavni delovi modula.

Definisanje modula uvek počinje pomoću ključne reči ***module***. Prvo se zadaju ime modula i lista *port*-ova, zatim se deklarišu vrste *port*-ova i uvode parametri. Lista portova i deklaracija portova su prisutni samo ako postoji interakcija modula i okruženja.

Telo modula sačinjavaju pet komponenti koje su: deklaracija tipa podataka, izrazi sa nivoa toka podataka, instanciranje modula nižih nivoa, izrazi na nivoau ponašanja i *Verilog* funkcije. Navedene komponente se mogu nalaziti u bilo kom redosledu unutar tela modula.

Definicija modula se uvek završava pomoću ključne reči ***endmodule***. Samo su ključna reč ***module***, ime modula i ključna reč ***endmodule*** obavezni sve ostalo se koristi prema potrebi. *Verilog HDL* dozvoljava definisanje više modula unutar jednog *ASCII* fajla i to u bilo kom redosledu. Redosled nema veze sa hijerarhijom.

Razni delovi modula će se ilustrovati pomoću primera modula koji definiše *RS latch*. Slika 7-2 prikazuje logičku šemu *RS latch*-a koji ima dva ulaza *S* i *R* i dva izlaza *Q* i *NotQ*. U primeru 7-1 data je definicija modula za *RS latch* i definicija jednog ispitnog modula (pod imenom *Stimulus*) koji može poslužiti za testiranje *latch*-a.



Slika 7-2:Logička šema RS latch-a.

Primer 7-1: Definisanje modula za RS latch-a i ispitnog modula.

```
// Ime modula i lista port-ova
module RS_latch(Q, NotQ, R, S);

// Deklaracija vrste port-ova
output Q, NotQ;
input R, S;
// Instanciranje modula nižih nivoa. U ovom slučaju se instanciraju NI kapije koji su
// osnovni gradivni elementi (primitive) koji su definsani malim slovima
// u Verilog HDL-u.
nand n1(Q, ~S, NotQ);
nand n2(NotQ, ~R, Q);

// Kraj modula, koji je obavezan
endmodule
```

```
// Ime modula i lista port-ova
module Stimulus;
// Nema liste port-ova jer je modul Stimulus modul najvišeg nivoa.

// Deklaracija tipova podataka.
wire q, notq;
reg set, reset;

// Instanciranje modula nižeg nivoa čija definicija je data gore.
RS_latch f1(q, notq, set, reset);

// Blok koji pripada nivou ponašanja (engl. behavioral level), služi za
// zadavanje pobudnih signala.
initial
begin
        set = 0; reset = 0;
        #5 reset = 1;
        #5 reset = 0;
        #5 set = 1;
        #5 $finish;
end

// Obavezna ključna reč na kraju modula.
endmodule
```

Kod prethodnog primera možemo zapaziti nekoliko karakterističnih detalja:

- Nisu prisutne sve komponente koje su prikazane na slici 7-1 u definiciji modula *RS latch*-a. Ne postoje ni deklaracije tipova podataka ni izrazi na nivoa toka podataka (***assign***) i nivoa ponašanja (***always*** ili ***initial***), ipak je definicija modula korektna.
- Modul sa imenom *Stimulus* za *RS latch* poseduje deklaraciju tipova podataka i izraze na nivou ponašanja ali nema listu *port*-ova, deklaraciju *port*-ova, a nema ni izraza na nivou toka podataka.

Još jednom da se naglasi: sve komponente modula osim ključne reči ***module***, ime modula i ključne reči ***endmodule*** su opcionog karaktera i koriste se po potrebi u bilo kom redosledu.

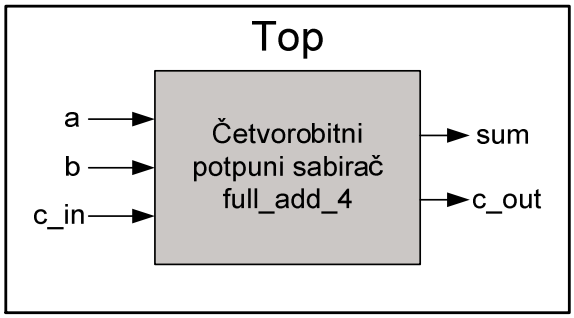
7.2. *Port-ovi*

Port-ovi su sprežne tačke modula preko kojih se omogućava komunikaciju sa drugim modulima. U *Verilog HDL* opisu uloga *port*-ova je analogna nožicama kod integrisanih kola. Jedini način da okruženje komunicira sa modulom je preko njegovih *port*-ova. Unutrašnja struktura modula je nevidljiva iz okruženja. Ovo daje veliku fleksibilnost u projektovanju jer se mogu načiniti promene u konstrukciji modula bez uticaja na način komuniciranja sa drugim modulima. *Port*-ovi se još nazivaju i terminali (engl.: ***terminal***).

7.2.1. Lista *port*-ova

U definiciji modula lista *port*-ova je opcionog karaktera. Ukoliko ne postoji komunikacija između okruženja (drugi moduli) i datog modula, *port*-ovi nisu potrebni.

Na slici 7-3 je prikazana struktura sa modulom *Top* koji nema listu *port*-ova i modula *full_add_4* koji se instancira u modulu *Top* i zato ima listu *port*-ova. Prvi redovi u definiciji navedenih modula su prikazani u primeru 7-2.



Slika 7-3: I/O portovi Top i full_add_4 modula.

Primer 7-2: Lista port-ova.

```
module full_add_4(sum, c_out, a, b, c_in);     // Modul sa listom port-ova.
module Top;                                    // Modul bez liste port-ova.
```

Top je modul najviše hijerarhije i zbog toga nema ulazne signale i ne gerneriše izlazne signale za eventualne više module u hijerarhiji. Zato nema ni listu *port*-ova. Modul koji je najviši u hijerarhiji komunicira samo sa softverom koji obrađuje *HDL* opis, što se ne manifestuje u definiciji modula.

7.2.2. Deklaracija *port*-ova

Svaki *port* koji se nalazi u listi *port*-ova mora biti deklarisan unutar modula. Pri deklarisanju zadaje se smer prenosa podataka preko *port*-a: postoje ulazni, izlazni i dvosmerni *port*-ovi (tabela 7-1).

Tabela 7-1: Vrste portova

Tip *port*-a	Opis
input	Ulazni *port*
output	Izlazni *port*
inout	Bidirekcioni *port*

Primer 7-3 prikazuje deklaraciju *port*-ova na primeru 4-bitnog potpunog sabirača prikazanog na slici 7-3.

Primer 7-3: Deklaracija port-ova.

```
module full_add_4(sum, c_out, a, b, c_in);

// Početak deklaracije port-ova
output [3:0] sum;
output c_out;
```

```
input [3:0] a, b;
input c_in;
// Kraj deklaracije port-ova

//-------------------------------------
// Telo modula
//-------------------------------------
endmodule
```

Ako se neki *port* deklariše kao ulazni, izlazni ili bidirekcioni, tom *port*-u se automatski dodeljuje nosilac podatka tipa ***wire***. Prema tome, ako je potreban *port* tipa ***wire***, dovoljno ga je deklarisati ključnim rečima ***input***, ***output*** ili ***inout,*** deklaracija tipa podaatka se izostavlja. Funkcija izlaznih *port*-ova je često pamćenje određenih vrednosti i u tim situacijama je potrebno da se za njih deklariše tip podatka ***reg***.

U gornjem primeru svim *port*-ovima pripadaju nosioci podataka tipa ***wire***. Primer 7-4 prikazuje deklaraciju *port*-ova *D flip-flop*-a, u kome *port*-u *q* pripada podatak tipa ***reg***.

Primer 7-4: Deklaracija port-ova D flip-flopa.

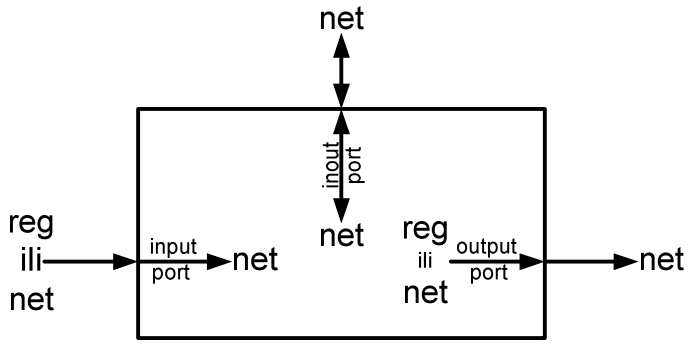
```
modulu DFF(q, d, clock, reset);

output q;          // Pošto je port q deklarisan kao izlaz,  posle deklaracije
                   // mu automatski pripada nosilac podatka tipa wire.
reg q;             // Pošto izlazna vrednost treba da se pamti,
                   // podatka koji mu pripada
                   // se posebno deklariše kao podatak tipa reg.
input d, clock, reset;
//-------------------------------------
// Telo modula
//-------------------------------------
endmodule
```

Ulazni i bidirekcioni (***input***, ***inout***) *port*-ovi se ne mogu deklarisati kao ***reg*** jer nosioci podataka tipa ***reg*** pamte vrednosti a uloga ulaznih *port*-ova je da prenose promene eksternih signala modulu.

7.2.3. Pravila za povezivanje *port*-ova

Port možemo shvatiti kao objekat koji se sastoji iz dva dela koja su međusobno povezana. Prvi deo pripada unutrašnjosti modula, dok drugi okruženju. Ova analogija će olakšati razumevanje pravila koja postoje pri instanciranju modula. Prevodilac *HDL* opisa javlja grešku ako su neka od ovih pravila prekršena. Pravila povezivanja su (slika 7-4):

- **Ulazni *port*-ovi**
  Ulaznim *port*-ovima modula uvek moraju pripadati podaci tipa *net*. Mogu se povezati sa nosiocima podataka tipa ***reg*** ili *net* u okruženju.
- **Izlazni *port*-ovi**
  Podaci koji pripadaju **i**zlaznom *port*-u modula mogu biti tipa ***reg*** ili *net*. Mogu se povezati u okruženju samo sa nosiocima podataka tipa *net*.
- **Bidirekcioni *port*-ovi**
  Bidirekcioni *port*-ovi modula moraju uvek biti tipa *net* i u okruženju se mogu povezati samo sa nosiocima podataka tipa *net*.



Slika 7-4: Pravila za povezivanje portova.

- **Poklapanje broja bita podataka**
  Verilog HDL omogućava povezivanje nosioca podataka različitih veličina pri povezivanju modul. Program prevodilac će generisati poruku sa upozorenjem o nepoklapanju veličina povezanih nosioca podataka.
- **Nepovezani *port*-ovi**
  Verilog HDL dozvoljava postojanje nepovezanih *port*-ova (nisu povezani ni sa kojim drugim *port*-om. Npr. uloga nekih *port*-ova može biti pružanje informacije u procesu debagovanja i u tom slučaju ostaju nepovezani. Pri kasnijoj hardverskoj realizaciji takve portove treba izostaviti.

Primer 7-5 ilustruje pravila povezivanja *port*-ova. Pretpostavimo da se *modul full_add_4* instancira u modulu *Top* prema slici 7-3. Način povezivanja *port*-ova je u većini slučajeva pravilan, međutim nosilac podatka *SUM* ne može da se deklariše na tip ***reg*** unutar modula *Top* pošto je povezan sa izlaznim *port*-om instanciranog modula i mora da prati vrednosti koje dolaze odatle.

Primer 7-5: Nedozvoljeno povezivanje port-ova.

```
module Top;

// Deklaracija nosioca podataka koji povezuju modul Top sa modulom full_add_4.
reg [3:0] A, B;
reg C_IN;
reg [3:0] SUM;              // Ispravno: wire [3:0] SUM;
wire C_OUT;

        // Instanciranje modula full_add_4 sa jedinstvenim imenom fa0.

        full_add_4 fa0(SUM, C_OUT, A, B, C_IN);

        // Instanciranje modula full_add_4 sadrži jedno ilegalno povezivanje
        // port-ova. Nosioc podatka SUM tipa reg u modulu full_add_4 je povezan
        // sa nosiocem podatka SUM u Top modulu koji je isto tipa reg.
        //-------------------------------------
        // Tu dolazi HDL opis koji predstavlja ispitni deo modula Top.
        //-------------------------------------
endmodule
```

7.2.4. Povezivanje *port*-ova sa signalima iz okruženja

Postoje dva načina povezivanja *port*-ova navedenih u listi *port*-ova sa spoljnim okruženjem (port-ovima drugih modula). Ove dve metode se ne mogu mešati.

- **Povezivanje preko uređene liste (engl.: *ordered list*)**

Korišćenje uređenih lista u povezivanju signala je najintuitivnija metoda za početne projektante. Signali koji se povezuju ovom metodom se navode u istom redosledu i u instanciranju modula i u definiciji modula. Ponovo ćemo se poslužiti primerom 4-bitnog potpunog sabirača sa slike 7-3.

HDL opis koji prikazuje povezivanje *port*-ova modula *Top* i *full_add_4* korišćenjem uređene liste je prikazan u primeru 7-6. Podaci iz modula Top: *SUM, C_OUT, A, B* i *C_IN* se pojavljuju u istom redosledu pri instanciranju modula *full_add_4* kao *port*-ovi *sum, c_out, a, b* i *c_in* u listi port-ova pri definisanju tog modula.

Važno je naglasiti da imena nosioca podataka i *port*-ova koji se povezuju na bazi uređene liste ne moraju biti identična. U primeru 7-6 *sum* i *SUM* su dva različita identifikatora jer *Verilog HDL* razlikuje mala i velika slova.

Primer 7-6: Povezivanje korišćenjem uređene liste.

```
module full_add_4(sum, c_out, a, b, c_in);

output [3:0] sum;
output c_out;
input [3:0] a, b;
input c_in;

        //-------------------------------------
        // Telo modula
        //-------------------------------------

endmodule
```

```
module Top;

reg [3:0] A, B;
reg C_IN;
wire [3:0] SUM;
wire C_OUT;

        // Instanciranje modula full_add_4 sa jedinstvenim imenom fa0.
        // Podaci se povezuju na osnovu pozicije koju zauzimaju u listi port-ova.

        full_add_4 fa0(SUM, C_OUT, A, B, C_IN);

        //-------------------------------------
        // Tu dolazi HDL opis koji predstavlja ispitni deo modula Top.
        //-------------------------------------

endmodule
```

- **Povezivanje *port*-ova na osnovu njihovih imena**

Veliki projekti mogu sadržati module sa 50 i više *port*-ova. Pamćenje redosleda *port*-ova pri definisanju modula u takvom slučaju je nepraktično i velika je verovatnoća greške u programiranju. Kao rešenje ovog problema *Verilog HDL* omogućava povezivanje *port*-ova na osnovu njihovih imena. U tom slučaju redosled port-ova je nebitan. Primer 7-7 prikazuje povezivanje podataka modula *Top* sa *port*-ovima četvorobitnog potpunog sabirača na osnovu njihovih imena.

Primer 7-7: Povezivanje port-ova na osnovu imena za primer 7-6.

```
full_add_4 fa0(.c_out(C_OUT), .sum(SUM), .b(B), .c_in(C_IN), .a(A));
```

Važno je napomenuti da se ne moraju navesti svi *port*-ovi kada se koristi povezivanje *port*-ova na osnovu njihovih imena. *Port*-ovi koji su nepotrebni u datoj primeni se mogu izostaviti pri povezivanju. Kada je u gornjem primeru *port c_out* nepotreban, način instanciranja modula full_add_4 je prikazan u primeru 7-8.

Primer 7-8: Nepotpuno povezivanje port-ova na osnovu imena za primer 7-6 ako je port c_out nepotreban.

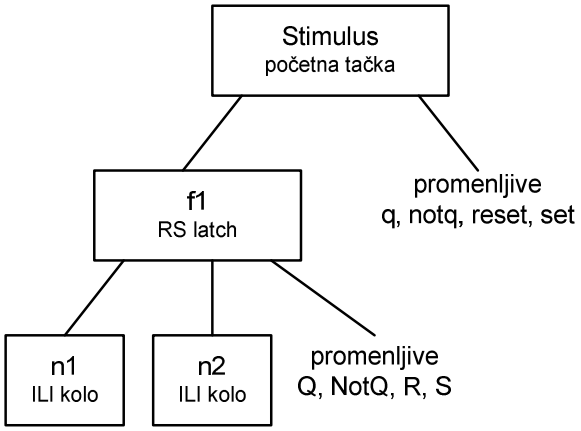
```
full_add_4 fa0(.sum(SUM), .b(B), .c_in(C_IN), .a(A));
```

7.3. *Hijerarhijska imena*

U glavi 5 je objašnjen koncept hijerarhijskog projektovanja kojeg podržava *Verilog HDL*. U toku projektovanja svakoj instanci modula i nosiocu podatka treba dodeliti identifikator (ime). Svaki identifikator poseduje jedinstveno mesto u hijerarhiji projekta. Dobijena struktura omogućava dodeljivanje jedinstvenog (hijerarhijskog) imena svakom indentifikatoru. Hijerarhijsko ime je niz identifikatora razdvojenih tačkom (“.”). Ime se formira tako da se obezbedi mogućnost adresiranja identifikatora sa bilo kog mesta u projektu navođenjem hijerarhijskog imena.

Pri kreiranja hijerarhijskog imena početna tačka je modul najvišeg nivoa. Hijerarhijsko ime identifikatora opisuje tačnu putanju (engl.: *path*) između početne tačke i datog identifikatora u hijerarhiji projekta. Proces kreiranja hijerarhijskih imena će se prikazati za slučaj *RS latch*-a iz primera 7-1. Hijerarhijska struktura projekta je prikazana na slici 7-5.

Za ovu simulaciju modul na vrhu hijerarhije je modul Stimulus. Identifikatori koji su definisani u ovom modulu su *q, notq, reset i set*. Stimulus modul instancira *f1* koji je modul tipa *RS_latch*. Instanca *f1* instancira *n1* i *n2 ILI* kola. *Q, NotQ, R* i *S* su nosioci podataka u instanci *f1*.



Slika 7-5:Hijerarhijska struktura formirana za simulaciju RS latch-a.

Za dobijanje hijerarhijskih imena zapisuju se imena početne tačke i svih instanci koje se nalaze na putanji do željenog identifikatora.

U primeru 7-9 su prikazana hijerarhijska imena koja se mogu napisati za sliku 7-5. Obratiti pažnju na tačke kojom se razdvajaju imena nivoa u hijerarhiji.

Primer 7-5: Hijerarhijska imena.

```
Stimulus.q          Stimulus.f1.Q
Stimulus.notq       Stimulus.f1.NotQ        Stimulus.f1
Stimulus.reset      Stimulus.f1.R           Stimulus.f1.n1
Stimulus.set        Stimulus.f1.S           Stimulus.f1.n2
```

8. *HDL* opis na nivou logičkih kapija

U glavi 5, 6 i 7 su postavljeni temelji projektovanja u *Verilog HDL*-u razmatranjem pristupa projektovanju, jezičkih konvencija, tipova podataka i strukture modula. Spomenuto je da se HDL pisi mogu formirati na četiri mguća nivoa. U ovoj glavi se detaljno razmatra nivo logičkih kapija (engl.: *gate level*).

Nivo logičkih kapija je drugi nivo apstrakcije i većina projekata se izrađuje na ovom ili na višim nivoima apstrakcije. Najniži nivo apstrakcije odnosno prvi nivo je nivo prekidača (engl.: *switch level*) koji je pogodan alat samo za projektante digitalnih komponenti: kašnjenja u kolu se mogu precizno pratiti i može se dobro oceniti efikasnost korišćenja osnovnih elemenata.

Projektovanje na nivou logičkih kapija je važno pre svega zbog toga što se na tom nivou mogu primeniti mnogi dosadašnji rezultati projektovanja: složenija kola projektovana primenom tradicionalnih *SSI* i *MSI* kola se tako najlakše mogu preneti u svet *PLD*-ova. Sa druge strane projektovanje na nivou logičkih kapija je najbliže projektantima sa osnovnim znanjem iz tradicionalne digitalne elektronike jer postoji korespondencija jedan na prema jedan između logičke šeme i *Verilog HDL* opisa digitalnog kola. Ovo je bio razlog za opredeljenje da se u ovom tekstu počinje sa opisom na nivou logičkih kapija.

8.1. *Vrste kapija*

Verilog HDL podržava projektovanje koje polazi od prostih logičkih kola. Logička kola su prisutna u hardverskom jeziku kao predefinisani osnovni elementi (engl.: ***primitive***. Njihova primena se vrši instanciranjem isto tako kao bilo kog drugog modula. Jedina razlika je da nije potrebno definisati module osnovnih elemenata jer su njihove definicije već sadržane u *Verilog HDL*-u. Postoje dve klase osnovnih elemenata (logičkih kola):

- *I* i *ILI* kola,
- kola za sprezanje.

8.1.1. *I* i *ILI* kola

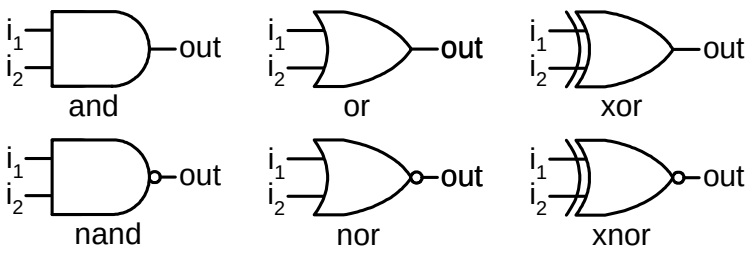
I i *ILI* kola imaju jedan jednobitni (skalarni) izlaz i više jednobitnih (skalarnih) ulaza. Prvi *port* u listi *port*-ova je izlazni a svi ostali su ulazni *port*-ovi. Princip rada osnovnih logičkih kola u *Verilog HDL*-u je identičan onim logičkim kolima koja su realizovana kao posebna integrisana kola: posle svake promene vrednosti nekog ulaznog signala se vrši izračunavanje novog izlaznog logičkog nivoa i njeno postavljanje na izlaz logičkog kola.

Imena definisanih modula za *I* i *ILI* logička kola u *Verilog HDL*-u su prikazana u tabeli 8-1. Ta imena se uvek navode na prikazani način kao što je slučaj i sa ključnim rečima.

Tabela 8-1: Imena modula za I i ILI kola definisana u Verilog HDL-u

Ime modula	*Funkcija*
and	*I* kolo
nand	*NI* kolo
or	*ILI* kolo
nor	*NILI* kolo
xor	Isključivo *ILI* kolo
xnor	Isključivo *NILI* kolo

Na slici 8-1 su prikazani šematski simboli razmatranih logičkih kola sa dva ulaza. Ulazi su imenovani sa **i1** i **i2** dok je izlaz naznačen sa **out**.



Slika 8-1: Šematski simboli I / ILI logičkih kola definisanih u Verilog HDL-u i imena njihovih modula.

Primeri instanciranja logičkih kola koja su definisana u *Verilog HDL*-u su dati u primeru 8-1. Kod svake instance, izlazni *port* logičke kapije (*out*) je vezan na nosilac podatka tipa *net* sa imenom *OUT* i nosioci podataka istog tipa *IN1* i *IN2* su priključeni na ulaze logičkih kola *i1* i *i2*. Međusobno povezivanje izlaza logičkih kola koje je prisutno u datom opisu nije uobičajeno.

U pimeru 8-1 prvih šest instanci odgovaraju dvoulaznim logičkim kapijama, sedma instanca je troulazno *NI* kolo. U *Verilog HDL*-u *I* i *ILI* logičke kapije mogu imati proizvoljan broj ulaza.

Osma instanca u primeru 8-1 nema ime. U slučaju logičkih kapija takvo instanciranje se takođe smatra pravilnim. Mogu se instancirati logička kola na stotine bez ijednog imena instanci.

Primer 8-1: Instaciranje I / ILI logičkih kola.

```
wire OUT, IN1, IN2, IN3;

// Instanciranje osnovnih logičkih kapija sa imenima
and a1(OUT, IN1, IN2);
nand na1(OUT, IN1, IN2);
or or1(OUT, IN1, IN2);
nor nor1(OUT, IN1, IN2);
xor x1(OUT, IN1, IN2);
xnor nx1(OUT, IN1, IN2);

// Instanciranje NI kola sa tri ulaza
nand na1_3inp(OUT, IN1, IN2, IN3);

// Instanciranje I kola bez imena
and (OUT, IN1, IN2);      // Dozvoljen način instanciranja logičkih kola
```

Izlazne vrednosti datih logičkih kola na sve moguće varijacije ulaznih vrednosti se određuju na osnovu kombinacionih tabela. Kombinacione tabele za data logička kola su definisane za dva ulaza i prikazane su na slici 8-2. Izlazne vrednosti logičkih kola sa više od dva ulaza se izračunavaju iterativnim postupkom na osnovu kombinacione tabele za dva ulaza.

and		i1			
		0	1	x	z
i2	0	0	0	0	0
	1	0	1	x	x
	x	0	x	x	x
	z	0	x	x	x

nand		i1			
		0	1	x	z
i2	0	1	1	1	1
	1	1	0	x	x
	x	1	x	x	x
	z	1	x	x	x

or		i1			
		0	1	x	z
i2	0	0	1	x	x
	1	1	1	1	1
	x	x	1	x	x
	z	x	1	x	x

nor		i1			
		0	1	x	z
i2	0	1	0	x	x
	1	0	0	0	0
	x	x	0	x	x
	z	x	0	x	x

xor		i1			
		0	1	x	z
i2	0	0	1	x	x
	1	1	0	x	x
	x	x	x	x	x
	z	x	x	x	x

xnor		i1			
		0	1	x	z
i2	0	1	0	x	x
	1	0	1	x	x
	x	x	x	x	x
	z	x	x	x	x

Slika 8-2: Kombinacione tabele dvoulaznih I / ILI logička kola.

Kod uobičajenih kombinacionih tabela se redovno navode samo varijacije definisanih ulaznih logičkih nivoa (nula ili jedinica). U gornjim tabelama je dato ponašanje kola i pri neodređenom logičkom nivou i pri stanju visoke impedanse. To je rađeno iz razloga što se i pri fizičkoj realizaciji javljaju navedene situacije.

Logičko kolo na svom izlazu formira logičku nulu ili logičku jedinicu (međuvrednosti nisu karakteristične), bez obzira na neodređeno stanje ili stanje visoke impedanse na ulazu. Jedini problem je da se u mnogim situacijama ne zna, koji od logičkih nivoa će se uspostaviti. Tabele koje važe u hardverskim jezicima, u maksimalnoj meri prate ono što se dešava pri fizičkoj realizaciji: ako se izlazni logički nivo može odrediti na bazi ulaza, u tabelu je upisana *0* ili *1*, ako se ne može odrediti, navodi se *x* (nepoznata, neodređena vrednost).

8.1.2. Kola za sprezanje

Ova logička kola imaju jedan skalarni ulaz i jedan ili više skalarnih izlaza. Zadnji *port* naveden u listi *port*-ova je priključen na ulaz a svi ostali na izlaz. Naredna razmatranja u ovoj tački će se odnostiti samo na logička kola sa jednim izlazom. Primena više izlaza je pogodna kada iz istog kola treba pobuditi više linija.

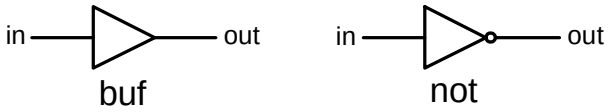
Verilog HDL podržava dva tipa kola za sprezanje: bafere (neinvertujući) i invertore (invertujući). Imena definisanih bafera u *Verilog HDL*-u su prikazana u tabeli 8-2. Ta imena se uvek pišu na isti način kao što je slučaj i kod ključnih reči.

Tabela 8-2: Imena definisanih modula za kola za sprezanje u Verilog HDL-u.

Ime modula	*Funkcija*
buf	bafer
not	invertor

Slika 8-3 prikazuje šematske oznake kola za sprezanje za slučaj jednog izlaza. Ime ulaznog *port*-a je *in*, a ime izlaznog *port*-a je *out*.

Slika 8-3: Šematske oznake i imena modula za neinverujuće i invertujuće kolo za sprezanje..



Primer 8-2 prikazuje način instanciranja kola za sprezanje u *Verilog HDL*-u. Ponovo se naglašava da ova logička kola mogu imati više izlaza ali samo jedan ulaz koji je uvek zadnji u listi *port*-ova.

Primer 8-2: Instanciranje bafera i invertora.

```
// Instanciranje bafera i invertora sa jednim ulazom i izlazom
buf b1(OUT, IN);
not n1(OUT, IN);

// Instanciranje sa više od jednog izlaza
buf b1_3out(OUT1, OUT2, OUT3, IN);

// Instanciranje invertora bez imena instance
not(OUT1, OUT2, IN);
```

Kombinacione tabele navedenih kola za sprezanje za slučaj sa jednim izlazom su prikazane na slici 8-4.

Slika 8-4: Kombinacione tabele kola za sprezanje.

buf	
in	out
0	0
1	1
x	x
z	x

not	
in	out
0	1
1	0
x	x
z	x

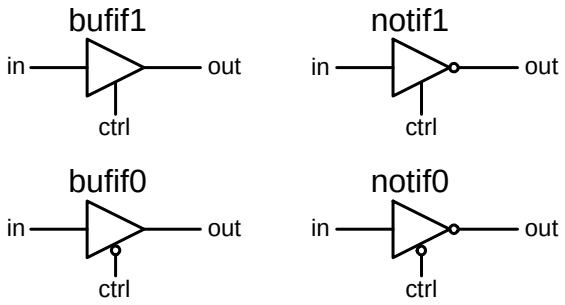
8.1.3. Kola za sprezanje sa tri stanja

U *Verilog HDL*-u postoje i kola za sprezanje sa kontrolnim ulazom isto kao kod hardverskih rešenje (tačka 2.1.2). Postoje četiri varijante tih kola: njihova imena modula i funkcije su sumirane u tabeli 8-3. I ova imena modula se koriste samo u istovetnoj formi.

Ime modula	*Funkcija*
bufif1	neinvertujući bafer, signal dozvole je aktivan na logičku jedinicu
bufif0	neinvertujući bafer, signal dozvole je aktivan na logičku nulu
notif1	invertujući bafer, signal dozvole je aktivan na logičku jedinicu
notif0	invertujući bafer, signal dozvole je aktivan na logičku nulu

Tabela 8-3: Imena modula i funkcije kola za sprezanje sa tri stanja definisana u Verilog HDL-u.

Ova kola za sprezanje se ponašaju kao obična kola za sprezanje (tačka 8.1.2) dok su kontrolni ulazi aktivni. U suprotnom slučaju njihovi izlazi su u stanju visoke impedanse. Šematski simboli kola su prikazani na slici 8-5.



Slika 8-5: Šematski simboli kola za sprezanje sa tri stanja.

Primer 8-3 prikazuje način instanciranja kola za sprezanje sa tri stanja. Prvi *port* u listi *port*-ova je izlaz, drugi je ulaz signala a treći je kontrolni ulaz. Radi uprošćenja i ovde se može izostaviti ime pri instanciranju.

Primer 8-3: Instanciranje logičkih kola bufif i notif.

```
// Instanciranje logičkih kola bufif
bufif1 b1(out, in, control);      // sa sopstvenim imenom
bufif0 (out, in, control);        // bez sopstvenog imena

// Instanciranje logičkih kola notif
notif1 n1(out, in, control);      // sa sopstvenim imenom
notif0 (out, in, control);        // bez sopstvenog imena
```

Na slici 8-6 su prikazane kombinacione tabele koje definišu ponašanje kola za sprezanje sa tri stanja. U tabelama su uzete u obzir sve moguće situacije na ulazima (pored logičke *0* i *1* i neodređeno stanje i stanje visoke impedanse).

bufif1		ctrl			
		0	1	x	z
in	0	z	0	x	x
	1	z	1	x	x
	x	z	x	x	x
	z	z	x	x	x

notif1		ctrl			
		0	1	x	z
in	0	z	1	x	x
	1	z	0	x	x
	x	z	x	x	x
	z	z	x	x	x

bufif0		ctrl			
		0	1	x	z
in	0	0	z	x	x
	1	1	z	x	x
	x	x	z	x	x
	z	x	z	x	x

notif0		ctrl			
		0	1	x	z
in	0	1	z	x	x
	1	0	z	x	x
	x	x	z	x	x
	z	x	z	x	x

Slika 8-6: Kombinacione tabele kola za sprezanjesa tri stanja definisanih u Verilog HDL-u.

Kola za sprezanje sa tri stanja u *Verilog HDL*-u imaju analognu ulogu kao odgovarajuća hardverska kola. Obično se koriste u situacijama kada više kola treba da koriste zajedničku liniju za prenos u različitim vremenskim intervalima (vremenski multpleks). U takvim situacijama signali se priključuju na zajedničku liniju preko kola za sprezanje sa tri stanja od kojih istovremeno uvek samo jedan dobije dozvolu. Na taj način se izbegava kolizija signala na zajedničkoj liniji.

Pod kolizijom se podrazumeva situacija kada više izlaza logičkih kola vezanih na zajedničku liniju pokušavaju da ostvare različite logičke nivoe. U takvoj situaciji neki izlazi daju a neki primaju preveliku struju, nastaju veliki gubici (zagrevanje) i u nekim situacijama može da dođe i do pregorevanje komponenti. Ne treba zanemariti ni činjenicu da se pri koliziji formira naponski nivo koji nije pravilani logički nivo što može da dovodi do neočekivanog ponašanja u kolu.

8.2. *Primer projektovanja na nivou logičkih kapija: četvorobitni potpuni sabirač*

Ranije (tačka 4.2.1) smo se upoznali sa pojmom jednobitnog polusabirača i potpunog sabirača i videli smo kako se može izgraditi višebitni sabirač kaskadnom vezom potpunih sabirača. U ovoj tački će se formirati opis četvorobitnog potpunog sabirača na hardverskom jeziku.

Lista *port*-ova je već bila definisana u primeru 7-2. Za projektovanje se koriste isključivo logička kola. Na kraju projektovanja će se formirati ispitni modul kojim će se proveriti funkcionalnost četvorobitnog potpunog sabirača. Primenićemo *top-down* pristup projektovanju: četvorobitni sabirač će se raščlaniti na četiri jednobitna potpuna sabirača.

Kombinaciona tabela jednobitnog potpunog sabirača je prikazana na slici 8-7. Korišćene su sledeće oznake:

- *c_in (carry input):* ulazni bit prenosa.
- *a, b:* ulazni jednobitni binarni brojevi.
- *sum:* zbir jednobitnih binarnih brojeva *a* i *b*.
- *c_out (carry out):* izlazni bit prenosa.

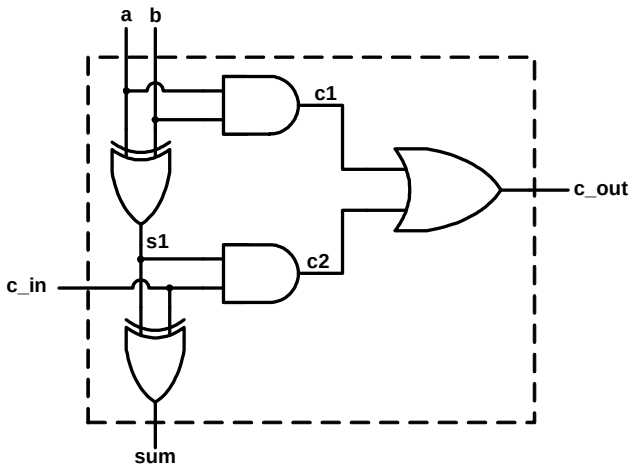
c_in	a	b	sum	c_out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Slika 8-7: Kombinaciona tabela jednobitnog potpunog sabirača.

Logičke jednačine izlaznih funkcija *sum* i *c_out* su:

$$\text{sum} = (\text{a} \oplus \text{b} \oplus \text{c\_in})$$
$$\text{c\_out} = \text{a} \cdot \text{b} + \text{c\_in} \cdot (\text{a} \oplus \text{b})$$

Polazeći od jednačina dobija se logička šema jednobitnog potpunog sabirača koja je prikazana na slici 8-8.



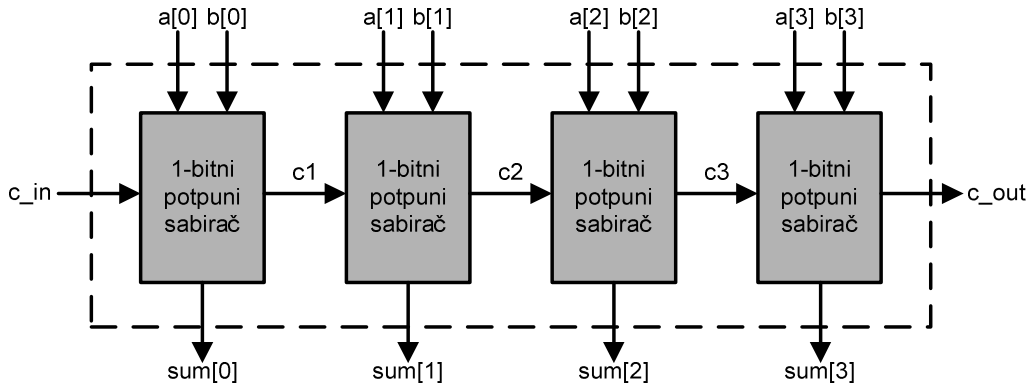
Slika 8-8: Logička šema jednobitnog potpunog sabirača

Na osnovu gornje šeme, u primeru 8-4 je napisan *HDL* opis jednobitnog potpunog sabirača na nivou logičkih kapija. Pri instanciranju logičkih kapija instance nemaju imena. Nazivi unutrašnjih veza (čvorova) su istovetni kao na logičkoj šemi na slici 8-8.

Primer 8-4: Verilog HDL opis jednobitnog potpunog sabirača.

```
// Definicija modula
module FullAdder1bit(sum, c_out, a, b, c_in);

// Deklaracije port-ova
output sum, c_out;
input a, b, c_in;

// deklaracije internih veza
wire s1, c1, c2;

// Instanciranje logičkih kapija (bez imena instanci)
xor (s1, a, b);
xor (sum, s1, c_in);

and (c1, a, b);
and (c2, s1, c_in);

or (c_out, c1, c2);

endmodule
```

Kaskadnom vezom četiri jednobitna potpuna sabirača nastaje četvorobitni sabirač koji je prikazan na slici 8-9.



Slika 8-9: Logička šema četvorobitnog sabirača.

Na osnovu logičke šeme je formiran *Verilog HDL* opis četvorobitnog sabirača dat u primeru 8-5. Pored potrebnih deklaracija ovaj modul vrši četiri instanciranja modula jednobitnog potpunog sabirača definisnog u primeru 8-4.

Primer 8-5: HDL opis četvorobitnog sabirača.

```
// Definicija četvorobitnog sabirača
module FullAdder4_bit(sum, c_out, a, b, c_in);

// Deklaracija port-ova
output [3:0] sum;
output c_out;

input [3:0] a, b;
input c_in;

// Deklaracija internih veza (čvorova)
wire c1, c2, c3;

// Instanciranja četiri jednobitna potpuna sabirača
```

```
FullAdder1bit fa0(sum[0], c1, a[0], b[0], c_in);
FullAdder1bit fa1(sum[1], c2, a[1], b[1], c1);
FullAdder1bit fa2(sum[2], c3, a[2], b[2], c2);
FullAdder1bit fa3(sum[3], c_out, a[3], b[3], c3);

endmodule
```

Potrebno je naglasiti da se koristi više istih imena u listi *port*-ova za modul jednobitnog potpunog sabirača i četvorobitnog sabirača ali se ta imena odnose na različite podatke. *Port sum* jednobitnog potpunog sabirača je skalarna veličina, dok *port sum* četvorobitnog sabirača je vektorska veličina široka četiri bita. Imena definisana unutar nekog modula su dostupna samo za taj modul zato ne dolazi do zbrke. Imena definisana unutar nekog modula su nevidljiva van modula osim ako se ne navodi celo hijerarhijsko ime datog podatka.

Svakoj instanci modula jednobitnog potpunog sabirača se moralo pridružiti jedinstveno ime u procesu instanciranja. Isto nije bilo potrebno pri instanciranju osnovnih elementa (logičkih kapija) pri definisanju modula jednobitnog potpunog sabirača (primer 8-4).

Kada je projekat završen, generisanjem ulaznih i posmatranjem izlaznih signala se mora proveriti funkcionalnost *HDL* opisa. Potrebno je definisati jedan ispitni modul koji generiše ulazne signale za projekat i prima izlazne signale. Analizom izlaznih signala će projektant ustanoviti da li dati modul funkcioniše pravilno.

U primeru 8-6 je data definicija ispitnog modula pod imenom *Stimulus*.

Primer 8-6: Modul Stimulus za ispitivanje četvorobitnog sabirača.

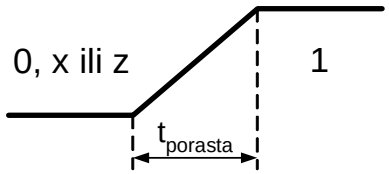
```
module Stimulus;

// Deklaracija tipova podataka za modul Stimulus
reg [3:0] A, B;
reg C_IN;
wire [3:0] SUM;
wire C_OUT;

// Instanciranje četvorobitnog sabirača
FullAdder4_bit FA1(SUM, C_OUT, A, B, C_IN);

// Definisanje ulaznih signala za četvorobitni sabirač
initial
begin
        A = 4'd0;     B = 4'd0;     C_IN = 1'b0;

        #5      A = 4'd3;     B = 4'd4;
        #5      A = 4'd2;     B = 4'd5;
        #5      A = 4'd9;     B = 4'd9;
        #5      A = 4'd10;    B = 4'd15;
        #5      B = 4'd5;     C_IN = 1'b1;
end

endmodule
```

8.3. *Kašnjenja*

Logička kola koja su se koristila u prethodnim primerima su bila bez kašnjenja, odnosno sa nultim kašnjenjem. Kod realnih logičkih kola uvek se javlja kašnjenje promene na izlazu u odnosu na promenu na ulazu. To se mora uzeti u obzir i u hardverskim opisima da bi pri simulaciji i pri kasnijoj realizaciji projekta dobili realne rezultate. Za simulaciju kašnjenja u realnim digitalnim sistemima, *Verilog HDL* omogućava definisanje tri vrste kašnjenja za osnovna logička kola.

- **Kašnjenje porasta**

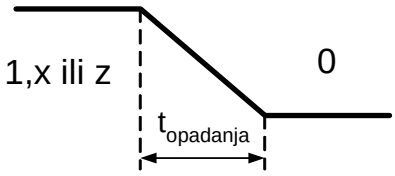
Kašnjenje porasta (engl.: *rise delay*) je vreme potrebno izlazu da se njegova logička vrednost sa *0*, *x* ili *z* vrednosti poraste na *1* (slika 8-10).



Slika 8-10: Prikaz vremena kašnjenja kod rastuće ivice signala.

- **Kašnjenje opadanja**

Kašnjenje opadanja (engl.: *fall delay*) je vreme potrebno izlazu da se njegova logička vrednost sa *1*, *x* ili *z* opadne na *0* (slika 8-11).



Slika 8-11: Prikaz vremena kašnjenja kod opadajuće ivice signala.

- **Kašnjenje isključivanja**

Kašnjenje isključivanja (engl.: *turn-off delay*) je vreme potrebno izlazu da se njegova logička vrednost sa *0, 1* ili *x* pređe u stanje visoke impedanse (*z*).

U *Verilog HDL*-u ne zadaje se posebna vrednost kašnjenja izlaza kada se izlaz menja sa poznate vrednosti (*0* ili *1*) u nepoznato stanje (*x*) Softver za prevođenje odnosno simulaciju automatski uzima minimalnu vrednost od prethodna tri kašnjenja.

Verilog HDL podržava tri načina zadavanja gore definisanih kašnjenja u logičkim kolima:

- Jedno kašnjenje je zadato i ta vrednost se koristi za sve prelaze.
- Dva kašnjenja su zadata i odnose se na kašnjenje porasta i opadanja. Kašnjenje isključivanja postaje manja vrednost od prethodna dva kašnjenja.
- Sve tri vrste kašnjenja su zadata i odnose se redom na kašnjenje porasta, opadanja i isključivanja.

Ukoliko ni jedna vrsta kašnjenja nije zadata, podrazumevana vrednost je *0*, odnosno nema kašnjenja. Jezičke konstrukcije za zadavanja kašnjenja su prikazane u primeru 8-7.

Primer 8-7: Jezičke konstrukcije za zadavanja kašnjenja.

```
// Svi prelazi imaju jednako kašnjenje koje je određeno vrednošću parametra delay_time.
and #(delay_time) a1(out1, in1, in2);

// Za kašnjenje porasta i opadanja su zadate različite vrednosti.
or #(rise_value, fall_value) o1(out2, in1, in2);

// Za kašnjenje porasta, opadanja i isključivanja su zadate tri različite vrednosti.
and #(rise_val, fall_value, turn_off_value) b1(out3, in, control);
```

U primeru 8-8 su prikazani konkretni slučajevi za zadavanje kašnjenja. Potrebno je naglasiti da su zadate brojne vrednosti relativne vrednosti, ne pretpostavlja se nikakva konkretna merna jedinica.

Primer 8-8: Primeri kašnjenja.

```
and #(5) a1(out1, in1, in2);          // Svi prelazi imaju kašnjenje od 5
                                      // vremenskih jedinica.
and #(4,6) a2(out2, in1, in2);        // Vreme porasta = 4, vreme opadanja = 6.
bufif0 #(3,4,5) b1(out3, in1, in2);   // Vreme porasta = 3, vreme opadanja = 4,
                                      // vreme isključivanja = 5
```

8.3.1. Mininimalne, tipične i maksimalne vrednosti kašnjenja

Na današnjem nivou tehnološkog razvoja nije moguće proizvesti dva integrisana kola sa istim karaktreristikama. Posledica ovog tehnološkog ograničenja je da kašnjenja integrisanih kola istih tipova variraju unutar određenih granica. Uvođenjem minimalnog, tipičnog i maksimalnog vremena kašnjenja moguće je proveriti ispravnost rada digitalnih kola bez obzira na to koji konkretan primerak integrisanog kola će se koristiti.

Smisao pojedinih vrednosti je sledeći:

- Minimalno vreme kašnjenja je najkraće potrebno vreme da se pojavi odziv na izlazu nakon promene ulaza.
- Tipično vreme kašnjenja je očekivano vreme kašnjenja.
- Maksimalno vreme kašnjenja je najduže potrebno vreme izlaza da odgovori na promenu na ulazu.

Verilog HDL omogućava definisanje minimalnog, tipičnog i maksilnog kašnjenja umesto jedne jedine konkretne vrednosti npr. za vreme porasta. Pomoću odgovarajućeg izbora u simulacionom softveru se odlučuje koja će se vrednost kašnjenja koristiti pri simulaciji. Izbor se odnosi na sva logička kola u projektu tako da će se recimo simulacija sprovesti sa minimalnim vremenima za sve instance.

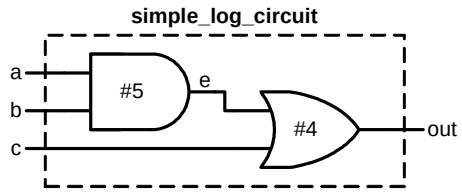
Ako se za određeno kašnjenja zadaje samo jedna vrednost, podrazumeva se da je to tipična vrednost kašnjenja. Primer 8-9 prikazuje način zadavanja minimalnih, tipičnih i maksimalnih vrednosti kašnjenja.

Primer 8-9: Minimalne, tipične i maksimalne vrednosti kašnjenja.

```
and #(4:5:6) a1(out1, i1, i2);
// Vremena kašnjenja porasta, opadanja i isključivanja su jednaka:
// Min  = 4, Tip = 5, Max = 6.

and #(3:4:5, 5:6:7) a2(out2, in1, in2);
// Vremena porasta:                Min = 3, Tip = 4, Max = 5
// Vremena opadanja:               Min = 6, Tip = 7, Max = 8
// Vremena isključivanja:          Min = 3, Tip = 4, Max = 5
// U slučaju kada su zadata samo vremena porasta i opadanja, vreme isključivanja se računa
// kao minimalna vrednost od vremena porasta i opadanja

and #(2:3:4, 3:4:5, 4:5:6) a3(out3, in1, in2);
// Vremena porasta:                Min = 2, Tip = 3, Max = 4
// Vremena opadanja:               Min = 3, Tip = 4, Max = 5
// Vremena isključivanja:          Min = 4, Tip = 5, Max = 6
```

8.3.2. Primer za analizu kašnjenja

Neka je dat modul *simple_log_circuit* u kojoj je implementirana logička funkcija: $\text{out} = a \cdot b + c$. Implementacija ovog modula na nivou kapija šematski je prikazana na slici 8-12. *Verilog HDL* opis modula *simple_log_circuit* je dat u primeru 8-10.



Slika 8-12: Logička šema za simple_log_circuit.

Primer 8-10: Definicija modula simple_log_circuit u hardverskom jeziku.

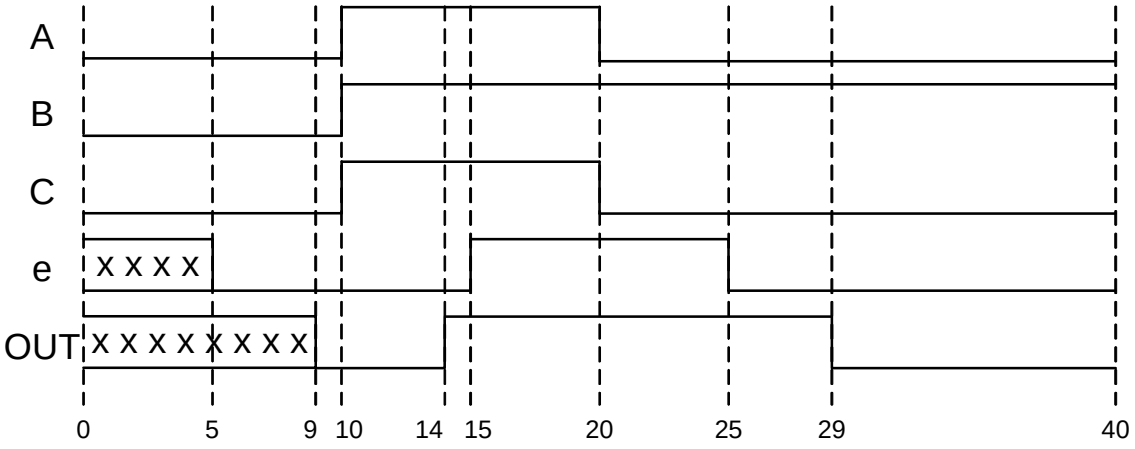
```
module simple_log_circuit(out, a, b, c);

// Deklaracija port-ova
output out;
input a, b, c;

// Interne veze
wire e;

// Instanciranje potrebnih primitiva za realizaciju date kombinacione mreže
and #(5) a1(e, a, b);       // Kašnjenje je 5 vremenskih jedinica
or #(4) o1(out, e, c);      // Kašnjenje je 4 vremenskih jedinica

endmodule
```

Ovaj modul se testira ispitnim modulom sa imenom *stimulus* koji je prikazan u primeru 8-11.

Primer 8-11: Ispitni modul stimulus za testiranje simple_log_circuit-a.

```
module stimulus;

// Deklaracija internih nosioca podataka
reg A, B, C;
wire OUT;

// Instanciranje modula simple_log_circuit
simple_log_circuit slc1(OUT, A, B, C);

// Definisanje ulaznih signala za modul simple_log_circuit.
// Primenjene jezičke konstrukcije će se objasniti kasnije.
// Simulacija se završava posle 40 vremenskih jedinica
initial
begin
            A = 1'b0;    B = 1'b0;    C = 1'b0;
      #10   A = 1'b1;    B = 1'b1;    C = 1'b1;
      #10   A = 1'b1;    B = 1'b0;    C = 1'b0;
      #20   $finish;
end

endmodule
```

Na slici 8-13 su prikazani vremenski dijagrami dobijeni simulacijom. Logičke vrednosti signala *e* i *OUT* se ne mogu znati pre isteka odgovarajućih kašnjenja zato je simulator na početku dijagrama stavio znakove *x*.



Slika 8-13: Vremeski dijagram dobijen simulacijom modula simple_log_circuit

9. *HDL* opis na nivou toka podataka

Projektovanje na nivou logičkih kapija je svrsishodno kada je reč o jednostavnim digitalnim kolima čija logička šema je poznata od ranije ili se lako razvije. U takvoj prilici za projektanta je najjedostavnije da instancira i poveže potrebna logička kola u okviru jednog Verilog HDL opisa na nivou logičkih kapija.

Kod kompleksnijih projekata u kojima je broj logičkih kola velik, proces projektovanja na nivou logičkih kapija postaje neefikasan. Rešenje je da se proces projektovanja podigne na jedan viši nivo apstrakcije da bi se projektanti mogli koncentrisati samo na traženu obradu podataka i da se ne opterećuju sa fizičkom implementaciju pomoću logičkih kola.

U *Verilog HDL*-u nivo toka podataka obezbeđuje jedan viši nivo apstrakcije gde je akcenat na toku i načinu obrade podataka umesto da se bavimo instanciranjem i povezivanjem logičkih kapija.

Brzo povećavanje broja logičkih kapija koji se mogu integrisati u jedno programabilno kolo je dovelo do sve veće popularnosti projektovanja na nivou toka podataka. Više nijedna kompanija ne može sebi priuštiti luksuz da resurse svojih inženjera troši na projektovanje na nivou logičkih kapija.

I danas se za realizaciju digitalnih kola koriste logičke kapije ali formiranje konkretnih kola vrše softveri za projektovanje na osnovu *HDL* opisa. Ovaj proces se naziva (automatizovanom) logičkom sintezom. Logička sinteza je postala popularna zato što su na raspolaganju moćni softveri za sintezu i od strane proizvođača *PLD*-ova i od strane nezavisnih softverskih kuća. Ovaj pristup omogućava projektantima da se koncentrišu na optimizaciju toka podataka u digitalnim kolima umjesto da gube vreme oko hardverskih detalja.

Za maksimalnu fleksibilnost u procesu projektovanja, projektanti obično koriste *HDL* opise koji kombinuju nivo toka podataka i nivo ponašanja. Izraz *RTL* (engl.: *Register Transfer Level*) koji se sreće u literaturi se odnosi na *HDL* opise koji modeliraju i na nivou toka podataka i na nivou ponašanja.

9.1. *Kontinualna dodela*

Kontinualna dodela je osnovni način dodele neke vrednosti nosiocu podatka tipa *net* na nivou toka podataka. U ovom smislu kontinualna dodela zamenjuje logičke kapije u *HDL* opisima ali se pri tome povećava nivo apstrakcije, omogućava opis kombinacionog kola bilo koje složenosti. Izraz za kontinualnu dodelu se počinje ključnom reči ***assign***. Odgovarajuća jezička konstrukcija je prikazana u primeru 9-1.

Primer 9-1: Jezička konstrukcija za kontinualnu dodelu vrednosti.

```
assign <kašnjenje> <izraz>;
```

Vrednost <***kašnjenje***> u izrazima kontinualne dodele je opcionalnog karaktera i služi za zadavanje vremenskog intervala između promene vrednosti desne strane ***assign*** izraza i pojave te promene na levoj strani izraza.

Kontinualne dodele imaju sledeće karakteristike:

- Izraz za definisanje dodele sadrži znak jednakosti. Na levoj strani ključne reči znaka jednakosti mora biti skalarni ili vektorski podatak tipa *net* ili pridruženi skalarni ili vektorski podatak tipa *net*. Operacija pridruživanja je objašnjena u tački 9.4.8. Na levoj strani znaka jednakosti ne može da stoji podatak tipa ***reg***.
- Kontinualne dodele su uvek aktivne. Pri svakoj promeni vrednosti operanada na desnoj strani znaka jednakosti odmah se izračunava nova vrednost izraza koja se odmah ili nakon propisanog kašnjenja dodeljuje nosiocu podatka na levoj strani izraza.

- Operandi na desnoj strani izraza mogu biti podaci tipa ***reg***, *net* ili *Verilog HDL* funkcijski pozivi (ovo poslednje nije obrađeno u ovom opisu). Nosioci podataka tipa ***reg*** i *net* mogu biti skalarnog ili vektorsk0g karaktera.

Primeri kontinualnih dodela su prikazani u primeru 9-2. Operatori kao što su *&, ^, |, { , } i +* su objašnjeni u poglavlju 9.4. U ovom momentu je važno samo obratiti pažnju na oblik ***assign*** izraza.

Primer 9-2: Primeri kontinualne dodele.

```
// Kontinualna dodela. Podaci  out, in1 i in2 su tipa net.
assign out = in1 & in2;

// Kontinualna dodela vektorskim podacima tipa net
// addr1_bits i addr2_bits su 16-bitni vektorski podaci tipa reg.
assign addr[15:0] = addr1_bits[15:0] ^ addr2_bits[15:0];

// U izrazu kontinualne dodele je prisutna operacija pridruživanja skalarne
// i vektorske promenlive tipa net
assign {c_out, sum[3:0]} = a[3:0] + b[3:0] + c_in;
```

U sledećoj tački je predstavljen kraći način zapisa kontinualne dodele nosiocima podataka tipa *net*.

9.1.1. Implicitna kontinualna dodela

Umesto posebne deklaracije nosioca podatka tipa *net* i pisanja izraza za kontinualnu dodelu, *Verilog HDL* omogućava jedan kraći način opisa gde se te dve operacije vrše u jednom koraku. U primeru 9-3 je prikazan regularan i implicitni način pisanja kontinualnih dodela.

Primer 9-3: Regularan i implicitni način kontinualne dodele.

```
//Regularan način pisanja kontinualne dodele,
// prvo se deklariše nosioc podatka.
wire out;
assign out = in1 + in2;

// Isti efekat se postiže implicitnom kontinualnom dodelom.
wire out = in1 + in2;
```

9.2. *Kašnjenja u dodelama*

Vrednostima kašnjenja se određuje potrebno vreme da se promena u vrednosti operanada na desnoj strani jednačine odrazi na vrednost operanda na levoj strani jednačine. Kašnjenja u izrazima kontinualne dodele se mogu deklarisati na tri načina:

- kašnjenja u regularnim dodelama (definisani ključnom reči ***assign***),
- kašnjenja kod implicitnih dodela,
- kašnjenja definisana pri deklaraciji nosioca podataka tipa *net*.

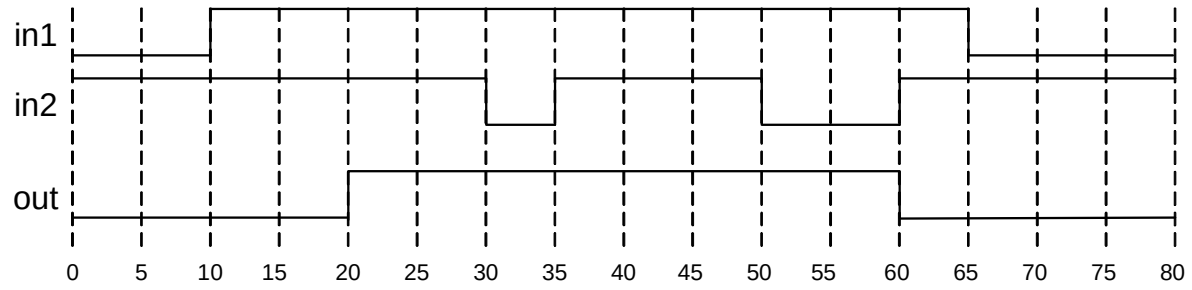
9.2.1. Kašnjenja u regularnim dodelama

Prvi način zadavanje kašnjenja se koristi u regularnim izrazima za kontinualnu dodelu. Vrednost kašnjenja se stavlja posle ključne reči ***assign.*** Primer 9-4 (realizuje *I* logićku funkciju) prikazuje način zadavanja kašnjenja u regularnoj dodeli.

Primer 9-4: Zadavanje kašnjenja u regularnoj dodeli.

```
assign #10 out = in1 & in2;
```

U primeru 9-4 rezultat promene u vrednostima signala *in1* i *in2* se pojavljuje na izlazu logičkog *I* kola sa kašnjenjem od *10* vremenskih jedinica. Na slici 9-1 su prikazani vremenski dijagrami signala na nosiocima podataka koji su korišćeni u primeru 9-4.



Slika 9-1: Vremenski dijagram signala na nosiocima podataka iz primera 9-4

Vremenski dijagrami prikazuju sledeće:

- U vremenskom trenutku *10* oba ulaza su na jedinici, dok se jedinica na izlazu pojavljuje na izlazu tek u trenutku *20*, sa kašnjenjem od *10* vremenskih jedinica, u skladu sa kašnjenjem definisanim u primeru 9-4.
- *in2* ima vrednost nula u intervalu od *30* do *35*, koja bi trebala vrednost izlaza promeniti u nulu u vremenskom intervalu između *40* i *45*, ali se to ne dešava jer je trajanje logičke nule suviše kratko (kraće od kašnjenja koje je definisano kao *10* vremenskih jedinica).
- logičke vrednosti na oba nosioca podatka na desnoj strani znaka jednakosti u vremenskom intervalu od *60* do *65* vremenskih jedinica su jedinice. Kao posledica, izlazni signal *out* bi trebao da ima vrednost jedan u intervalu vremena od *70* do *75*, ali se ni to ne dešava jer istovremeno trajanje jedinica je kraće od kašnjenja.
- Na početnom delu dijagrama *out* (pre isteka kašnjenja) naznačene su nepoznate vrednosti (*x*) jer u tom intervalu se ne može odrediti leva strana izraza (smatra se da logičko stanje zavisi od događanja pre posmatranog intervala).

Ponašanje da logička stanja kratkog trajanja u kontinualnim dodelama ne uzrokuju promene na levoj strani ***assign*** izraza zove se inercijalno kašnjenje. Slična pojava je bila prisutna i pri opisu na nivou logičkih kapija, što je objašnjeno u poglavlju 8.3.

9.2.2. Kašnjenja u implicitnim kontinualnim dodelama

Spomenuto je (tačka 9.1.1) da *Verilog HDL* omogućava definisanje implicitne kontinualne dodele pri deklaraciji tipa podatka. Istovremeno se mogu zadati i kašnjenja. Zadavanje kašnjenja u implicitnoj dodeli je prikazano u primeru 9-5:

Primer 9-5: Primer zadavanja kašnjenja u implicitnoj dodeli.

```
wire #10 out = in1 & in2;

// Gornji izraz ima isti efekat kao posebna deklaracija tipa podatka i
// pisanje izraza za kontinualnu dodelu sa kašnjenjem.
wire out;
assign #10 out = in1 & in2;
```

9.2.3. Kašnjenja definisana pri deklaraciji nosioca podataka tipa *net*

Kašnjenje je moguće definisati za podatak tipa *net*, bez upotrebe izraza za kontinualnu dodelu. Ako je kašnjenje definisano za podatak tipa *net*, onda se svaka promena prenosi na nosilac podatka tipa *net* sa datom vrednošću kašnjenja. U primeru 9-6 je prikazana odgovarajuća jezička konstrukcija za kašnjenja koja su definisana pri deklaraciji tipa nosioca podatka. Ovaj način definisanja kašnjenja se može koristiti i pri opisu na nivou logičkih kapija.

Primer 9-6: Kašnjenje definisano pri deklaraciji podatka tipa net.

```
wire #10 out;
assign out = in1 & in2;

// Gornji izrazi imaju isti efekat kao i sledeći:
wire out;
assign #10 out = in1 & in2;
```

Posle kratkog upoznavanja sa osnovnim pravilima za kontinualnu dodelu i kašnjenja, slede pravila u vezi izraza, operatora i operanada koji se koriste u izrazima kontinualne dodele.

9.3. *Izrazi, operandi i operatori*

Nivo toka podataka opisuje digitalno kolo korišćenjem matematičkih izraza umesto instanciranja logičkih kola. Ti matematički izrazi i operatori i operandi u njima čine osnovu modeliranja na nivou toka podataka.

9.3.1. Izrazi

Izrazi su kostrukcije koje kombinuju operatore i operande radi dobijanja željene obrade podataka (primer 9-7).

Primer 9-7: Tri izraza koji kombinuju razne operatore i operande.

```
a ^ b
addr1[7:4] + addr2[7:4]
in1 | in2
```

9.3.2. Operandi

Uopšteno, operandi mogu biti podaci bilo kog tipa definisanih u poglavlju 6.2. Neke konstrukcije prihvataju samo određene tipove operanada. Operand može biti konstanta, ceo broj (***integer***), realan broj (***real***), *net*, ***reg***, vektor, deo vektorske veličine (koji može sadržati jedan ili više bitova) ili funkcijski poziv (u ovom opisu, kao što je već napomenuto, funkcijski pozivi ne predstavljaju predmet razmatranja). U primeru 9-8 se prikazuju načini korišćenja operanada.

Primer 9-8: Korišćenje raznih tipova operanada.

```
integer count, final_count;
final_count = count + 1;              // count je operand tipa integer

real a, b, c;
c = a – b;                            // a i b su operandi tipa real

reg [7:0] reg1, reg2;
reg [3:0] reg_out;
reg_out = reg1 [3:0] ^ reg2[3:0];     // reg1[3:0] i reg2[3:0] su operandi koji
                                      // predstavljaju deo vektorskih
                                      // podataka reg1 i reg2
```

9.3.3. Operatori

Nad operandima se primenjuju razne operacije s ciljem dobijanja željenih rezultata. Opreacije se definišu pomoću operatora. U *Verilog HDL-u* je na raspolaganju veliki broj operatora (primer 9-9). Tipovi operatora su predstavljeni u poglavlju 9.4.

Primer 9-9: Primeri korišćenja operatora.

```
d1 && d2    // && je binarni operator nad operandima d1 i d2.
!a[0]       // ! je unarni operator nad operandom a[0].
C >> 2      // >> je binarni operator nad operandima C i 2.
```

9.4. *Tipovi operatora*

Verilog HDL pruža mogućnost za korišćenje velikog broja različitih tipova operatora. Operatori mogu biti aritmetički, logički, relacioni, redukcioni, pomerački, pridruživački ili uslovni. Neki operatori su slični operatorima korišćenih u *C* programskom jeziku. Svaki operator ima jedinstveni simbol. Tabela 9-1 prikazuje sve operatore grupisane po kategorijama.

Tip operatora	Simbol operatora	Operacija	Broj operanada
Aritmetički	*	množenje	2
	/	deljenje	2
	+	sabiranje	2
	-	oduzimanje	2
	%	deljenje po modulu	2
Logički	!	logička negacija	1
	&&	logičko I	2
	||	logičko *ILI*	2
Relacioni	>	veće od	2
	<	manje od	2
	> =	veće ili jednako	2
	< =	manje ili jednako	2
Jednakosti	= =	jednakost	2
	!=	nejednakost	2
	= = =	case jednakost	2
	!= =	case nejednakost	2
Bit	~	bit negacija	1
	&	bit *I*	2
	|	bit *ILI*	2
	^	bit *EX-ILI*	2
	^~ ili ~^	bit *EX-NILI*	2
Redukcioni	&	redukcioni *I*	1
	~&	redukcioni *NI*	1
	|	redukcioni *ILI*	1
	~|	redukcioni *NILI*	1
	^	redukcioni *EX-ILI*	1
	^~ ili ~^	redukcioni *EX-NILI*	1
Pomerački	>>	pomeranje udesno	2
	<<	pomeranje ulevo	2
Pridruživanje	{ }	pridruživanje	bilo koji broj
Umnožavanje	{ { } }	umnožavanje	bilo koji broj
Uslovni	? :	uslovni	3

Tabela 9-1: Lista svih operatora grupisanih po kategorijama

9.4.1. Aritmetički operatori

Aritmetičke operacije se mogu odnositi na jedan operand (unarne operacije) ili na dva operanda (binarne operacije).

- **Binarni operatori**

Binarni aritmetički operatori su: množenje (*), deljenje (/), sabiranje (+), oduzimanje (-) i deljenje po modulu (%). Binarni operatori se primenjuju nad dva operanda. U primeru 9-10 je prikazan način primene binarnih operatora.

Primer 9-10: Primeri korišćenja binarnih aritmetičkih operatora.

```
A = 4'b0011; B = 4'b0100;      // Podatak A i B su vektorske veličine tipa reg
D = 6; E = 4;                  // Podaci D i E su tipa integer

A * B              // Množenje A i B. Rezultat je 4'b1100.
D / E              // Deljenje D sa E. Rezultat je 1. Ostatak deljenja se odbacuje.
A + B              // Sabiranje A i B. Rezultat je 4'b0111.
B – A              // Oduzimanje A od B. Rezultat je 4'b0001.

// Ako je vrednost nekog bita operanda nepoznata (x), onda je i rezultat celog izraza nepoznat (x).
in1 = 4'b101x;
in2 = 4'b1011;

sum = in1 + in2;   // Rezultat sabiranja će biti: 4'bx

// Operator deljenja po modulu daje ostatak pri deljenju dva broja.
13 % 3         // Rezultat je 1.
16 % 4         // Rezultat je 0.
-7 % 2         // Rezultat je -1 jer se uzima predznak prvog operanda.
7 % -2         // Rezultat je 1 iz istog razloga.
```

- **Unarni operatori**

Operatori + i – mogu biti i unarni operatori (primer 9-11). U tom slučaju se koriste za zadavanje pozitivnog ili negativnog predznaka operanda. Unarni operatori + i – imaju veći prioritet od binarnih operatora + i –.

Primer 9-11: Primeri unarnih aritmetičkih operatora.

```
-4      // Minus 4
+5      // Plus 5
```

Korišćenje negativnih brojeva se preporučuje samo u tom slučaju kada je tip nosioca podatka ***integer*** ili ***real***. Kod brojeva koji su dati u obliku **<veličina>'<osnova_brojnog_sistema><broj>** bi trebalo izbegavati korišćenje negativnog predznaka jer takve brojeve *Verilog HDL* prevodi u oblik drugog komplementa i rezultat može biti neočekivan (primer 9-12).

Primer 9-12: Primer neočekivanog rezultata pri deljenju negativnog broja datog u obliku <veličina>'<osnova_brojnog_sistema><broj>.

```
// Za predstavljanje negativnih brojeva se preporučuje korišćenje
// nosioca podatka tipa integer ili real.
// Ako se koristi nosioc podatka integer ili real, dobija se očekivani rezultat.
-10 / 5        // Rezultat je -2.

// Izbegavati oblik <veličina>'<osnova_brojnog_sistema><broj> za predstavljanje neg. brojeva
-'d10 / 5      // Rezultat je (drugi komplement od 10) / 5 = (2^32 – 10) / 5
```

```
// gde 32 je podrazumevana širina reči datog računara.
// Rezultat je neočekivan i pogrešan.
```

9.4.2. Logički operatori

Logički operatori koji se mogu koristiti u Verilog HDL-u su: logičko *I* (znak *&&*), *ILI* (znak ||) i *NE* (znak *!*) operatori (primer 9-13). Operatori *&&* i || povezuju dva operanda (binarni operatori). Operator *!* se odnosi na jedan operand (unarni operator). Rad logičkih operatora određuju sledeća pravila:

- Rezultat logičkih operacija je uvek veličine jedan bit i može biti *0* (netačan), *1* (tačan) ili *x* (neodređen).
- Ako je vrednost operanda različita od nule, njegov logički ekvivalent je *1* (tačan). Ako je vrednost operanda jednaka nuli, njegov logički ekvivalent je *0* (netačan). Ako je vrednost jednog ili više bita kod nekog operanda *x* ili *z*, njegov logički ekvivalent je *x* (neodređeno).
- U logičkim operacijama operandi mogu biti nosioci podataka i izrazi.

Korišćenje zagrada za grupisanje logičkih izraza je preporučljivo kako bi se povećala preglednost *HDL* opisa. Druga prednost korišćenja zagrada je da se ne mora paziti na prioritete korišćenih operatora.

Primer 9-13: Primeri korišćenja logičkih operatora.

```
// Prosti slučajevi.
A = 3; B = 0;
A && B                  // Rezultat je: ( 1 && 0 ) = 0.
A || B                  // Rezultat je: (1 || 0 ) = 1.
!A                      // Rezultat je: NE( 1 ) = 0.
!B                      // Rezultat je: NE( 0 ) = 1.

// Neodređeni bitovi u operandima.
A = 2'b0x; B = 2'b10;
A && B                  // Rezulat je: ( x && 1 ) = x.

// Primena izraza kao operanda.
(a == 1 ) && ( b == 3 )  // Rezultat je 1 ako su oba uslova tačna.
                         // Rezultat je 0 ako je barem jedan od ta dva uslova netačan.
```

9.4.3. Relacioni operatori

Relacione operacije su: veće od (znak >), manje od (znak <), veće ili jednako (znak >=), manje ili jednako (znak <=). Rezultat relacione operacije je *1* ako je iskaz tačan, odnosno *0* ako je iskaz netačan. Ukoliko jedan od operanada u iskazu sadrži *x* ili *z* vrednost, rezultat relacione operacije je *x*. Ovi operatori funkcionišu na identičan način kao njihovi odgovarajući operatori u programskom jeziku *C*. U primeru 9-14 su prikazani načini primene relacionih operatora.

Primer 9-14: Primeri korišćenja relacionih operacija.

```
A = 4; B = 2;
X = 4'b1010; Y = 4'b1011; Z = 4'b1xxx;

A <= B;     // Rezultat je 0.
A > B;      // Rezultat je 1.
Y <= X;     // Rezultat je 0.
Y < Z;      // Rezultat je x.
```

9.4.4. Operatori jednakosti

U ovu grupu spadaju operacije logičke jednakosti (znak = =), logičke nejednakosti (znak *!*=), *case* jednakosti (znak = = =) i *case* nejednakosti (znak *!*= =). Opreacije jednakosti kao rezultat daju *1* ako je iskaz tačan, odnosno *0* ako je netačan (tabela 9-2). Ovi operatori upoređuju operande bit po bit. Ako su operandi različitih veličina, dopuni se operand manje veličine sa nulama sa strane veće težine pre upoređivanja.

Izrazi	Opisi	Moguće logičke vrednosti
a = = b	a jednako b, rezultat je neodređen ako je x ili z prisutan među bitovima a ili b	0, 1, x
a != b	a nije jednako b, rezultat je neodređen ako je x ili z prisutan među bitovima a ili b	0, 1, x
a = = = b	a jednako b, pri ispitivanj jednakosti se razmatraju i vrednosti x ili z u bitovima a i b	0, 1
a != = b	a nije jednako b, pri ispitivanj jednakosti se razmatraju i vrednosti x ili z u bitovima a i b	0, 1

Tabela 9-2: Opisi operacija jednakosti.

Važno je naglasiti razliku između operatora logičke jednakosti (= =, *!*=) i case jednakosti (= = =, *!*= =) (primer 9-15). Operatori logičke jednakosti (= =, *!*=) kao rezultat proveravanja jednakosti dva operanda daju *x*, ako barem jedan od dva operanda sadrži *x* ili *z*. Za razliku od toga, operatori case jednakosti (= = =, *!*= =) proveravaju oba operanda bit po bit, uključujući i vrednosti *x* i *z*. Operatori case jednakosti nikada ne daju rezultat *x*.

Primer 9-15: Primeri korišćenja operacija jednakosti.

```
A = 4; B = 3;
X = 4'B1010; Y = 4'b1101;
Z = 4'b1xxz; M = 4'b1xxz; N = 4'b1xxx;

A == B      // Rezultat je 0.
X != Y      // Rezultat je 1.
X == Z      // Rezultat je x.
Z === M     // Rezultat je 1 jer svi bitovi se poklapaju uključujući x i z.
Z === N     // Rezultat je 0 jer bitovi sa najmanjom binarnom težinom se razlikuju.
M !== N     // Rezultat je 1.
```

9.4.5. Operatori bit po bit

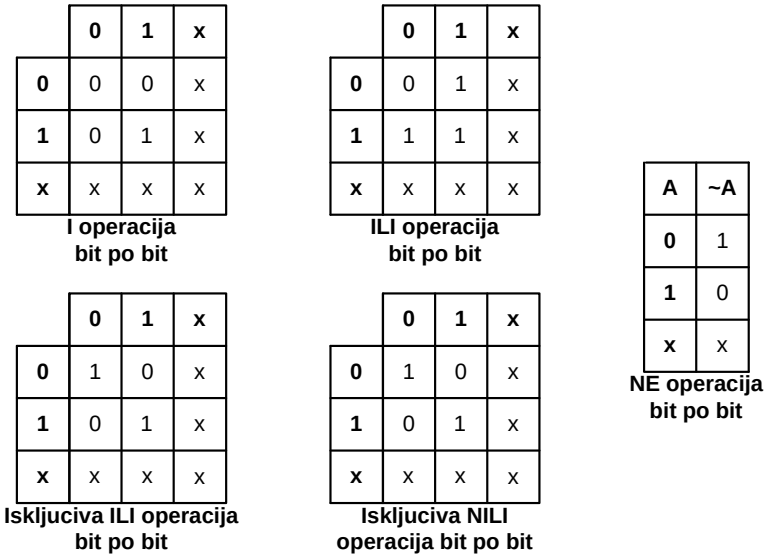
Operacije nad bitovima su: negacija (znak ~), *I* (znak *&*), *ILI* (znak |), *EX-ILI* (znak ^), *EX-NILI* (znak ^~ ili ~^). Operatori bit po bit izvršavaju bit po bit operacije nad jednim ili dva operanda. Pri negaciji, pojedinačno se negiraju bitovi operanda. U preostalim slučajevima aktuelna operacija se izvršava redom nad pojedinačnim bitovima prvog operanda i njemu odgovarajućim bitom drugog operanda. Ako su dva operanda različitih veličina, manji će se proširiti nulama na veličinu većeg operanda. U primeru 9-16 su prikazane primene operatora tipa bit po bit.

Primer 9-16: Primeri upotrebe operacija bit po bit.

```
X = 4'b1010;        Y = 4'b1101 ;       Z = 4'b10x1;

~X          // Negacija bit po bit. Rezultat je 4'b0101.
X & Y       // I operacija bit po bit. Rezultat je 4'b1000.
```

```
X | Y      // ILI operacija bit po bit. Rezultat je 4'b1111.
X ^ Y      // EX-ILI operacija bit po bit. Rezultat je 4'b0111.
X ^~ Y     // EX-NILI operacija bit po bit. Rezultat je 4'b1000.
X & Z      // I operacija bit po bit. Rezultat je 4'b10x0.
```

Kombinacione tabele za operacije bit po bit su prikazane na slici 9-2. Vrednost *z* u operacijama bit po bit se tretira kao vrednost *x*. U suštini, na isti način smo postupili i pri projektovanju na nivou logičkih kapija (8. fejezet).



Slika 9-2: Kombinacione tabele za operacije bit po bit.

Važno je naglasiti razliku između operatora nad bitovima: ~, &, | i logičkih operatora: *!*, *&&*, ||. U logičkim operacijama operandi se smatraju za jednu logičku promenljivu bez obzira na broj bita dok se kod opearcija bit po bit operacija izvršava nad pojedinim bitovima operanda (primer 9-17).

Primer 9-17: Primer koji prikazuje razliku između operacija bit po bit i logičkih operacija.

```
X = 4'b1010;        Y = 4'b0000;

X | Y;      // ILI operacija bit po bit. Rezultat je 4'b1010;
X || Y;     // Logička ILI operacija. Izraz je ekvivalentan sa 1 || 0 = 1.
```

9.4.6. Redukcioni operatori

Redukcione operacije su: *I* (znak *&*), *NI* (znak *~&*), *ILI* (znak |), *NILI* (znak ~|), *EX-ILI* (znak ^) i *EX-NILI* (znak ~^ ili ^~). Redukcioni operatori se odnose samo na jedan operand (unarne operacije). Izvršavaju se operacije nad bitovima operanda i kao rezultat uvek se dobija jedan bit.

Kombinacione tabele redukcionih operatora se poklapaju sa kombinacionim tabelama operacija bit po bit prikazanih u tački 9.4.5 (slika 9-2). Razlika između njih je da se redukcione operacije izvršavaju nad bitovima jednog operanda krećući se od bita najveće težine prema bitu najmanje težine, dok se operacije bit po bit izvršavaju nad bitom (bitovima) određene težine jednog ili dva operanda. Ovo je prikazano u primeru 9-18.

Primer 9-18: Primeri sa redukcionim operacijama.

```
X = 4'b1010;

&X          // Izračunava se pomoću jednačine: 1 & 0 & 1 & 0 = 1'b0.
|X          // Izračunava se pomoću jednačine: 1 | 0 | 1 | 0 = 1'b1.
^X          // Izračunava se pomoću jednačine: 1 ^ 0 ^ 1 ^ 0 = 1'b0.

// Redukcione operacije EX-ILI ili EX-NILI se mogu iskoristiti za generisanje
// bita parnosti ili neparnosti nekog vektora.
```

9.4.7. Operatori pomeranja

Operacije pomeranja su: pomeranje udesno (znak >>) i pomeranje ulevo (znak <<). Pri ovim operacijama pomera se sadržaj vektorskog operanda udesno ili ulevo za dati broj binarnih mesta. Prvi operand je vektor čiji se sadržaj pomera a drugi operand je vrednost koja određuje broj binarnih mesta za koja se vrši pomeranje. Prazna binarna mesta se tokom pomeranja ne popunjavaju bitovima koji su ispali već nulama. Načini primene operatora pomeranja su prikazani u primeru 9-19.

Primer 9-19: Primeri sa operacijama pomeranja.

```
X = 4'b1100;

Y = X >> 1;  // Y = 4'b0110. Sadržaj vektora X se pomera za jedan bit udesno i
             // mesto najviše značajnog bita se popunjava sa 0.
Y = X << 1;  // Y = 4'b1000. Sadržaj vektora X se pomera za jedan bit ulevo i
             // mesto najmanje značajnog bita se popunjava sa 0.
Y = X << 2;  // Y = 4'b0000. Sadržaj vektora X se pomera za dva bita ulevo i
             // mesta dva najmanje značajnih bitova se popunjavaju sa 0.
```

Operatori pomeranja su korisni jer omogućuju implementaciju algoritama za množenje (koji se sastoje od pomeranja i sabiranja), algoritama za kodovanje u telekomunikacijama itd.

9.4.8. Operator pridruživanja

Operacija pridruživanja (znak *{, }*) omogućava povezivanje više operanada u jedan operand. Operandi moraju imati specificiranu veličinu. Operandi bez zadate veličine su nedozvoljeni jer se ne bi mogla izračunati veličina rezultata.

Operacija pridruživanja se sastoji od niza operanada razdvojenih zarezom unutar vitičastih zagrada. Operandi mogu biti skalarni podaci tipa *net* ili ***reg***, vektorski podaci tipa *net* ili ***reg***, deo vektorskog podatka ili konstante sa veličinom (primer 9-20).

Primer 9-20: Primeri sa operacijama pridruživanja.

```
A = 1'b1;    B = 2'b00;   C = 2'b10;   D = 3'b110;

Y = { B, C };                  // Y = 4'b0010
Y = { A, B, C, D, 3'b001 };    // Y = 11'b10010110001
Y = { A, B[0], C[1] };         // Y = 3'b101
```

9.4.9. Operator umnožavanja

Operacija umnožavanja (znak *{ { } }*) ima dva operanda. Prvi operand je konstanta koja određuje koliko puta je potrebno uzastopno napisati drugi operand (primer 9-21). Operatore umnožavanja i pridruživanja unutar jednog izraza je moguće kombinovati na sledeći način:

{ $konst_1${$broj_1$}, $konst_2${$broj_2$}, ..., $konst_n${$broj_n$} }

Primer 9-21: Primeri operacija umnožavanja u kombinaciji sa pridruživanjem.

```
reg A;
reg [1:0] B, C;
A = 1'b1;     B = 2'b00;   C = 2'b10;
Y = { 4{A} };              // Y = 4'b1111
Y = { 4{A}, 2{B} };        // Y = 8'b11110000
Y = { 4{A}, 2{B}, C };     // Y = 10'b1111000010
```

9.4.10. Uslovni operator

Uslovni operator (znak ***?:***) ima tri operanda. Način upotrebe je sledeći:

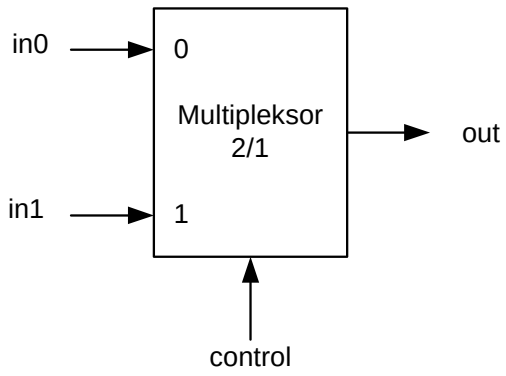
uslov ? *izraz_za_slučaj_tačnog_uslova* : *izraz_za_slučaj_netačnog_uslova*;

Pri izvršavanju uslovne operacije, prvo se izračunava tačnost (logička vrednost) iskaza *uslov*. Ako je *uslov* tačan, koristi se *izraz_za_slučaj_tačnog_uslova*. Ako je *uslov* netačan, koristi se *izraz_za_slučaj_netačnog_uslova*.

Ako je tačnost iskaza *uslov* neodređena (*x*) onda se izračunavaju vrednosti oba izraza (*izraz_za_slučaj_tačnog_uslova* : *izraz_za_slučaj_netačnog_uslova*) i rezultat se formira upoređivanjem izraza. Rezultat se određuje bit po bit: na onim binarnim mestima gde oba izraza daju istu logičku vrednost, stavlja se ta vrednost dok u slučaju nepoklapanja stavlja se vrednost *x*.

Uslovni operatori se često koriste na nivou toka podataka za modeliranje uslovnih dodeljivanja: tačnost uslova određuje, vrednost kojeg izraza će se dodeliti nosiocu podatka na levoj strani ***assign*** izraza.

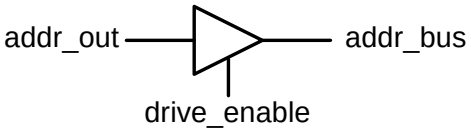
Uslovna dodela radi kao digitalni multipleksor 2/1 (slika 9-3, primer 9-22) tako da se uslovna operacija može koristiti za opisivanje multipleksora.



Slika 9-3: Demonstracija funkcionisanja uslovne operacije pomoću multipleksora 2/1.

Pored opisivanja multipleksora na nivou toka podataka, uslovni operatori se primenjuju i za opisivanje bafera sa tri stanja (slika 9-4, primer 9-22).

Slika 9-4: Kolo za sprezanje sa tri stanja.



Primer 9-22: Opisivanje multipleksora i kola za sprezanje sa tri stanja pomoću uslovne operacije.

```
// Opisivanje multipleksora 2/1 (slika 9-3) pomoću uslovne operacije.
assign out = control ? in1 ? in2;

// Modeliranje kola  za sprezanje sa tri stanja (slika 9-4)
// korišćenjem uslovne opercije.
assign addr_bus = drive_enable ? addr_out : 36'bz;
```

Uslovne operacije je moguće koristiti i na iterativan način. Izrazi unutar uslovnih operacija mogu da sadrže druge uslovne operacije. Ova mogućnost je prikazana preko jednog primera.

Zadatak je da se uporede dva dvobitna broja (*A* i *B*). Vrednost izlaza treba da bude logička jedinica ako su oba broja parna ili neparna, u suprotnom slučaju vrednost izlaza postaje logička nula. Dovoljno je ispitati parnost bita najmanje težine na način koji je prikazan u primeru 9-23. Izlaz se određuje u dva koraka (na iterativan način) primenom uslovnih operacija.

Primer 9-23: Uslovne opercije korišćene na iterativan način.

```
reg [1:0] A, B;              // Podaci A, B su reg tipa, veličine 2 bita.

assign out = A[0] ? ( B[0] ? 1'b1 : 1'b0 ) : ( B[0] ? 1'b0 : 1'b1);
```

9.4.11. Hijerarhija operacija

Prioritete operacija u izrazima bez zagrada *Verilog HDL* određuje prema tabeli 9-2. Preporučuje se korišćenje zagrada za razdvajanje izraza kako bi se izbegli problemi oko određivanja prioriteta operacija. Izuzetak čine unarne operacije kod kojih ne postoji neizvesnost oko prioriteta.

Operacije	Simboli operatora	Prioriteti
unarni množenje, deljenje, deljenje po modulu	+ - ! ~ * / %	Najveći prioritet
sabiranje, oduzimanje pomeranje	+ - << >>	
relacione jednakosti	< <= > >= = = != = = = != =	
redukcioni logički	& ~& ^ ~^ | ~| && ||	
uslovni	? :	Najmanji prioritet

Tabela 9-2: Hijerarhija operacija

10. *HDL* opis na nivou ponašanja

Najviši nivo za opis digitalnih kola pomoću hardverskog jezika je nivo ponašanja. Visoki nivo apstrakcije omogućava da se projektant ne opterećuje detaljima realizacije, već ponašanjem kola koje se projektuje. Ovi opisi jako liče na programe napisane u programskom jeziku *C* ali uvek treba imati na umu da hardverski opis služi za formiranje određenog digitalnog kola. Na osnovu *HDL* opisa softver za sintezu razvije kolo koje se na kraju ostvaruje programiranjem odgovarajućeg *PLD*-a.

Verilog HDL moduli za opis simulacija obično sadrže opise na nivou ponašanja koji nisu predviđeni za hardversku realizaciju, njihova namena je samo ispitivanje drugih modula.

10.1. *Struktuirane procedure*

Opisi na nivou ponašanja se mogu pisati u okviru jedne od dve moguće struktuirane procedure (engl.: *structured procedure*). Na osnovu ključnih reči kojima se uvode te procedure oni se zovu ***initial*** i ***always*** procedure. Instrukcije na nivou ponašanja se mogu pojaviti samo unutar tih procedura.

Jedan *Verilog HDL* modul može da sadrži i više struktuiranih procedura. Definicija nove procedure se ne može početi dok nije završena prethodno započeta. Sve te procedure u toku simulacije projektovanog modula počinju u trenutku *t=0* i izvršavaju se paralelno (konkurentno), za razliku od softverskih algoritama gde se izvršavanje programa dešava sekvencijalno, korak po korak. Ta osobina je povezana sa time da se u digitalnom kolu koje se realizuje na bazi *HDL* opisa, razni blokovi takođe rade istovremeno, paralelno obavljaju funkcije.

10.1.1. Procedura tipa *initial*

Procedura tipa ***initial*** počinje sa ključnom reči ***initial***. Posle ključne reči sledi ***initial*** blok koji može da sadrži jednu ili više dodela vrednosti. Izvršavanje operacija u ***initial*** bloku počinje u trenutku *t=0* i izvršavanje se dešava tačno jedan puta. Ako jedan modul sadrži više ***initial*** procedura, izvršavanje svakog od njih počinje u trenutku *t=0*, izvršavaju se i završavaju se međusobno nezavisno.

Pri projektovanju digitalnih kola ***initial*** procedure se obično koriste za jednokratna podešavanja. U većini slučajeva ***initial*** procedure se ne odnose na opis nekog kola, već se koriste u simulacionim modulima za zadavanje pobudnih signala.

Ako ***initial*** procedura sadrži samo jednu dodelu vrednosti, ona se piše neposredno iza ključne reči ***initial***. Ako se iza ključne reči navodi ***initial*** blok koji sadrži više dodela, pre bloka treba napisati ključnu reč ***begin***, a na kraj bloka treba staviti ključnu reč ***end***. U primeru 10-1 se mogu videti razne ***initial*** procedure.

Primer 10-1: Simulacioni modul koji sadrži više initial procedura.

```
module Stimulus;
reg a, b, m;
initial
        m=1'b0;          // U slučaju jedne dodele nisu potrebne ključne reči begin i end.
initial
begin
        #5 a=1'b1;       // U slučaju dve dodele već je potrebna primena ključnih reči
        #25 b=1'b0;      // begin i end. Kašnjenja u procedurama će se objasniti kasnije.
end
initial
        #50 $finish;     // Procedura tipa initial koja sadrži jednu instrukciju koja definiše
                         // kraj simulacije.
endmodule
```

U toku simulacije gore navedenog modula sve tri ***initial*** procedure kreću u trenutku *t=0*. Pojedina dodeljivanja će se izvršavati sa kašnjenjem, ako je tako definisano u opisu. Stvarna vremena izvršavanja dodeljivanja i instrukcija su data u tabeli 10-1.

vreme	dodela
0	m=1'b0
5	a=1'b1
30	b=1'b0
50	$finish

Tabela 10-1: Stvarna vremena izvršavanja dodela i instrukcija iz primera 10-1.

10.1.2. Procedura tipa *always*

Definisanje ***always*** procedure počinje sa ključnom reči ***always***, nakon čega sledi ***always*** blok sa jednom ili više dodela. I u ovom slučaju je potrebno koristiti ključne reči ***begin*** i ***end*** ako se radi o bloku koji sadrži više dodela. Dodele iz ***always*** bloka se ciklički ponavljaju u toku celog vremena simulacije (ako je reč o simulacionom modulu) ili za sve vreme rada projektovanog kola. U primeru 10-2 je dat modul za generisanje takt signala (koji sadrži ***always*** proceduru), potreban za simulaciju nekog kola.

Primer 10-2: Modul za generisanje takt signala sa always procedurom.

```
module clock_gen;
reg clock;
initial
        clock=1'b0;             // Takt signal kreće sa niskog logičkog nivoa.
always
        #10 clock=~clock;       // Posle svakih deset vremenskih jedinica treba invertovati
                                // logičku vrednost takt signala.
initial
        #1000 $finish;          // Procedura tipa initial za definisanje kraja simulacije.
endmodule
```

10.2. *Dodela vrednosti u procedurama*

Dodele definisane u procedurama tipa ***initial*** i ***always*** dodeljuju novu vrednost nosiocima podataka tipa ***reg***, ***integer*** i ***real***. Tako dobijene vrednosti se ne menjaju dok se ne vrši nova dodela. Ovo je značajna razlika u odnosu na dodele na nivou toka podataka: tamo je dodela nove vrednosti nosiocima podataka tipa net izvršavana kontinualno. Čim se desi neka promena u ulaznim podacima, odmah je izračunata i dodeljena nova vrednost izlaznom nosiocu podataka.

Dodele o kojima je ovde reč se definišu znakom za dodelu, desno od znaka stoji neki izraz, levo od znaka se navodi ime nekog nosioca podataka, neki njegov bit ili grupa bita ili pridruženi nosioci podataka.

Razlikujemo dve vrste dodela u procedurama: postoje blokirajuće i neblokirajuće dodele. Razlike među tim dodelama će se objasniti u narednoj tački.

10.2.1. Blokirajuće dodele

Znak dodele u blokirajućim dodelama je znak jednakosti (=). Ove dodele se izvršavaju u onom redosledu, kako su navedeni u ***initial*** ili ***always*** bloku. Dodela koja je trenutno stigla u fazu izvršavanja blokira sledeće dodele u tom bloku. Dodele navedene u dvema različitim ***initial*** ili ***always*** procedurama ne blokiraju jedan drugog, izvršavaju se nezavisno. U primeru 10-3 se vidi deo

jednog modula sa blokirajućim dodelama, a u tabeli 10-2 je sumirano po kom redosledu pojedine dodele dolaze na izvršavanje i u kom momentu se izvršavaju.

Primer 10-3: Deo modula sa blokirajućim dodelama.

```
reg x, y, z;
reg [15:0] reg_a, reg_b;
integer count;
initial
begin
        x=1'b0; y=1'b; x=1'b1;
        count=0;
        reg_a=16'b0; reg_b=reg_a;
        #15 reg_a[2]=1'b1;
        #10 reg_b[15:13]={x,y,z};
        count=count+1;
end
```

Tabela 10-2: Redosled dolaska na izvršavanje i vreme izvršavanja dodela iz primera 10-3.

Redosled dolaska na izvršavanje	Vreme izvršavanja	Dodela
1	0	x=1'b0
2	0	y=1'b1
3	0	z=1'b1
4	0	count=0
5	0	reg_a=1'b0
6	0	reg_b=reg_a
7	15	reg_a[2]=1'b1
8	25	reg_b[15:13]={x,y,z}
9	25	count=count+1

U dodelama se može desiti da nosilac podatka sa leve strane znaka dodele nema isti broj bita kao izračunata vrednost izraza sa desne strane znaka dodele. Ako nosilac podatka ima manji broj bita, bitovi manje težine iz vrednosti izraza se dodeljuju nosiocu podatka, a bitovi veće težine otpadaju. U suprotnom slučaju nedefinisani bitovi se popunjavaju nulama.

10.2.2. Neblokirajuće dodele

U neblokirajućim dodelama se koristi znak <=. Taj znak se ujedno koristi i u operacijama upoređivanja, ali softver koji tumači *HDL* opis ume da rastumači da li se radi o upoređenju ili o dodeli. Kod ovih dodela dodela koja je na redu ne blokira izvršavanje narednih dodela. Ponovićemo primer 10-3, ali tri dodele ćemo promeniti da budu neblokirajuće i pratićemo posledice (primer 10-4). U tabeli 10-4 je sumirano po kom redosledu pojedine dodele iz primera 10-4 dolaze na izvršavanje i u kom momentu se izvršavaju.

Primer 10-4: Deo modula sa blokirajućim i neblokirajućim dodelama.

```
reg x, y, z;
reg [15:0] reg_a, reg_b;
integer count;
initial
begin
        x=1'b0; y=1'b; x=1'b1;
        count=0;
        reg_a=16'b0; reg_b=reg_a;
        reg_a[2]<= #15 1'b1;
        reg_b[15:13]<= #10 {x,y,z};
```

```
        count<=count+1;
end
```

Redosled dolaska na izvršavanje	Vreme izvršavanja	Dodela
1	0	x=1'b0
2	0	y=1'b1
3	0	z=1'b1
4	0	count=0
5	0	reg_a=1'b0
6	0	reg_b=reg_a
7	15	reg_a[2]=1'b1
7	10	reg_b[15:13]={x,y,z}
7	0	count=count+1

Tabela 10-3: Redosled dolaska dodela na izvršavanje iz primera 10-4 i stvarna vremena njihovog izvršavanja.

Može se reći da se prvih šest dodela izvršavaju u trenutku *t=0*, i to u redosledu pisanja (osobina blokirajućih dodela). Posle toga, ali još uvek u trenutku *t=0*, preostale tri dodele dolaze na izvršavanje istovremeno. Ako nije naznačeno nikakvo kašnjenje, naznačene dodele se dešavaju odmah (u trenutku *t=0*). Ako ima kašnjenja, nosilac podatka sa leve strane znaka dodele će nakon isteka kašnjenja primiti vrednost koja je izračunata na osnovu vrednosti podataka u izrazu sa desne strane znaka dodele u trenutku *t=0*.

Sa neblokirajućim dodelama se obično opisuju digitalna kola u kojima se dešava konkurentni (istovremeni) prenos podataka pod delovanjem zajedničkog kontrolnog signala. U primeru 10-5 se dešavaju tri dodele na uzlaznoj ivici takt signala. Vremenska kontrola dodela u procedurama (pomoću takt signala ili drugačije) će se obraditi detaljno u poglavlju 10.3.

Primer 10-5: Konkurentni prenos podataka primenom neblokirajućih dodela.

```
always @ (posedge clock)
begin
        reg1<=#1 in1;
        reg2<=@(negedge clock) in2^in3;
        reg3<=#1 reg1;
end
```

U gornjem primeru na uzlaznoj ivici takt signala dešavaju se sledeće stvari:

- Odrede se vrednosti podataka u izrazima sa desne strane znakova dodele (*in1*, *in2*, *in3*, *reg1*), dotični izrazi se odmah izračunaju i dobijene vrednosti zapisuju u privremenoj memoriji.
- Stvarne dodele se dešavaju kada im dođe vreme. Prva i treća dodela se odigravaju nakon kašnjenja od jedne vremenske jedinice (zbog naznačenog kašnjenja), dok se druga dodela izvršava na najbližoj silaznoj ivici takt signala.
- Nije bitno po kom redosledu će se vršiti upis u odgovarajuće registre (izvršavanje dodele) pošto sadržaj privremene memorije ne zavisi od redosleda. Npr. u trećoj dodeli *reg3* dobija "staru" vrednost *reg1*, bez obzira na činjenicu da je u prvoj dodeli reg1 dobio novu vrednost.

Da bismo još bolje utvrdili pravila prenosa podataka kod neblokirajućih dodela, prvo posmatrajmo primer 10-6.

Primer 10-6: HDL opis sa blokirajućim dodelama koji dovodi do efekta protrčavanja.

```
always @ (posedge clock)
a=b;
always @ (posedge clock)
b=a;
```

Navedene dve ***always*** procedure kreću istovremeno (pri svakoj uzlaznoj ivici takt signala). Nijedan od napisanih blokirajućih dodela u navedenim procedurama nema prioritet (pošto su napisane u dvema procedurama) zato će redosled izvršavanja biti slučajan. Kada se jedna dodela izvrši, oba registra će sadržati istu vrednost, sledeća dodela već nema efekta.

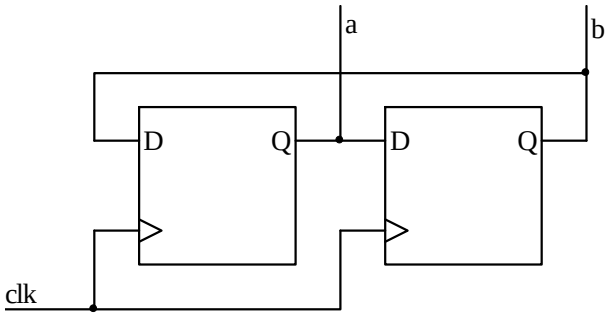
Efekat protrčavanja se može sprečiti primenom neblokirajućih dodela (primer 10-7). U ovom slučaju se prvo zapišu početne vrednosti podataka sa desne strane znaka za dodelu. Nakon toga se izvrše dodele korišćenjem zapisanih vrednosti i rezultat neće zavisiti od redosleda izvršavanja.

Primer 10-7: Sprečavanje efekta protrčavanja primenom neblokirajućih dodela.

```
always @ (posedge clock)
a<=b;
always @ (posedge clock)
b<=a;
```

Na osnovu gornjeg *HDL* opisa sintetizuje se digitalno kolo (kružni brojač) prikazano na slici 10-1. U kolu će se nakon svake uzlazne ivice takt impulsa (*clk*) zameniti sadržaji *flip-flop*-ova.

Osnovna osobina ivično okidanih kola je da uzorkuju ulazne signale na odgovarajućoj ivici takt signala i nakon toga izvrše potreban upis u skladu sa uzorkovanim vrednostima. Ako se ulazni signali promene nakon ivice takt signala, to neće imati nikakvog efekta na upis.



Slika 10-1: Digitalno kolo sintetizovano na bazi HDL opisa iz primera 10-7.

Efekat protrčavanja se može izbeći i primenom blokirajućih dodela i to na način prikazan u primeru 10-8. Uvođenjem (privremenih) nosilaca podataka temp_a i temp_b, početne vrednosti se zapisuju i tražena smena podataka se pravilno odigrava.

Primer 10-8: Sprečavanje efekta protrčavanja privremenim memorisanjem podataka.

```
always @ (posedge clock)
begin
        temp_a=a;
        temp_b=b;
        a=temp_b;
        b=temp_a;
end
```

10.3. *Vremenska kontrola dodela u opisima na nivou ponašanja*

Ako u proceduralnim dodelama prilikom opisa na nivou ponašanja ne navodimo nikakvu vremensku kontrolu, u toku simulacije opisa neće biti napredovanja na vremenskoj skali. Pri projektovanju realnih kola preciznije modelovanje zahteva primenu vremenske kontrole. U struktuiranim procedurama (***initial*** i ***always***) vremenska kontrola se može vršiti na tri načina:

- zadavanjem kašnjenja;
- ivičnim okidanjem;
- okidanjem na nivo signala.

10.3.1. Vremenska kontrola zadavanjem kašnjenja

Zadavanjem kašnjenja definišemo vreme koje treba da prođe između dolaska dodele na izvršavanje i njenog stvarnog izvršavanja. Kašnjenja uvodimo sa znakom #, posle znaka se piše vrednost kašnjenja u vidu brojčane vrednosti, identifikatora ili *min/typ/max* izraza.

Pri pravilnom načinu pisanja, kašnjenje se navodi pre dodele. Ovaj slučaj je prikazan u primeru 10-9.

Primer 10-9: Pravilan način pisanja kašnjenja kod dodela.

```
initial
begin
        #10 y=1'b1;
        #latency z=1'b0;
        #(4,5,6) q=1'b0;
end
```

Prva dodela se izvršava nakon kašnjenja od deset vremenskih jedinica računato u odnosu na *t=0*. Ako se na desnoj strani znaka dodele pojavljuje neki izraz (umesto konstante u datoj dodeli), izračunavanje izraza kasni isto toliko. S obzirom na osobine blokirajućih dodela, druga dodela dolazi na izvršavanje tek nakon izvršavanja prve dodele. Pošto smo kod druge dodele vrednost kašnjenja zadali u vidu identifikatora (konstante), potrebno je sačekati vreme određeno identifikatorom da bi se dodela izvršila. Nakon izvršavanja druge dodele treća dodela dolazi na izvršavanje kod koje smo kašnjenje zadali sa *min/typ/max* izrazom.

Drugi način pisanja kašnjenja je navođenje kašnjenja unutar same dodele. Odgovarajuće vreme kašnjenja se navodi na desnoj strani znaka za dodelu (= ili <=). U ovom slučaju izračunavanje vrednosti izraza koji figuriše na desnoj strani dodele se vrši u momentu dolaska dodele na izvršavanje (ne posle kašnjenja, kao što je slučaj kod pravilnog zadavanja kašnjenja), a sama dodela kasni onoliko koliko je propisano. U primeru 10-10 u prvoj ***initial*** proceduri kašnjenje je zadato unutar dodele, zato se sabiranje dešava u trenutku *t=0*, dok dodela kasni pet vremenskih jedinica. Druga ***initial*** procedura pokazuje kako se može postići isti rezultat uvođenjem privremenog nosioca podatka (*temp_xz*) i pravilnim načinom pisanja kašnjenja.

Primer 10-10: Pisanje kašnjenja unutar izraza za dodelu.

```
initial
y= #5 x+z;

initial
begin
        temp_xz=x+z
        #5 y=temp_xz
end
```

Kao što je već napomenuto ranije, izvršavanje svih ***initial*** i ***always*** procedura počinje u trenutku *t=0*. Ne može se međutim utvrditi redosled izvršavanja dodela u pojedinim procedurama

koja su naznačena za trenutak *t=0*, zato može nastupiti efekat protrčavanja. Zadavanjem nultog kašnjenja može se utvrditi redosled.

U primeru 10-11 sve dodele se izvršavaju u *t=0*, ali dodele unutar prve procedure se izvršavaju pre dodela u drugoj proceduri, pa će krajnji rezultat biti *x=y=1*. U principu, rizično je nosiocu podataka istovremeno dodeliti vrednost u dvema procedurama, ali uvođenjem nultog kašnjenja, redosled izvršavanja dodela se može dodatno precizirati.

Primer 10-11: Pisanje nultog vremena kašnjenja radi preciziranja redosleda izvršavanja.

```
initial
begin
        x=0;
        y=0;
end

initial
begin
        #0 x=1;
        #0 y=1;
end
```

10.3.2. Vremenska kontrola primenom ivičnog okidanja i okidanja na nivo

Većina savremenih digitalnih sistema je sinhrona mreža, a sinhronizacija se u većini slučajeva izvodi ivičnim okidanjem. Ivično okidanje i okidanje na nivo se može naznačiti na sledeća četiri načina u hardverskim opisima na nivou ponašanja:

- pravilno ivično okidanje,
- čekanje na imenovani događaj,
- čekanje na ivicu jednog od grupe signala,
- čekanje na odgovarajući logički nivo.

Primenom pravilne metode opisuje se ivično okidanje, koriste se ključne reči **@** i ***posedge*** ili ***negedge***. U primeru 10-12, unutar prve ***always*** procedure, dodelu treba izvršiti i na uzlaznoj i na silaznoj ivici takt signala. U drugoj proceduri dodela se vrši na uzlaznoj ivici takta, u trećoj na silaznoj ivici takta. U četvrtoj proceduri se memoriše vrednost podatka *d* u momentu kada dodela dolazi na izvršavanje, a dodela te vrednosti nosiocu podatka *q* se izvršava na uzlaznoj ivici takta.

Primer 10-12: Pravilna metoda vremenske kontrole (definisanje ivičnog okidanja).

```
always @ (clock)
        q=d;
always @ (posedge clock)
        q=d;
always @ (negedge clock)
        q=d;
always
        q=@(posedge clock) d;
```

Potrebno je naglasiti da u hardverskim opisima ne postoji nikakav gotov izvor takt signala, niti neki poseban, rezervisani naziv za takt signal (u gornjem primeru *clock* nije ključna reč). Takt signal koji je potreban za simulaciju *HDL* opisa mora da formira projektant prema svojim zahtevima u jednom pomoćnom modulu. Isti pristup važi i pri hardverskoj realizaciji *HDL* opisa: *PLD* kolo ne sadrži unutrašnji izvor takta, takt signal se mora obezbediti pomoću posebnog kola.

Ako vremensku kontrolu planiramo da ostvarimo čekanjem na imenovani događaj, prvo

treba deklarisati događaj. Naznačena dodela će se izvršiti kada nastupi taj događaj. Deklaracija događaja se vrši sa ključnom reči ***event,*** aktiviranje događaja se vrši sa znakom **->**, a događaj se uzima u obzir (izvrši se dodela) kod znaka **@**.

U primeru 10-13 deklaracija događaja je urađena u prvom redu *HDL* opisa. Sledi procedura tipa ***always*** koja definiše kada se javlja imenovani događaj (šta je ustvari događaj koji se čeka). Način korišćenja ključne reči ***if*** koja se koristi u ovoj jezičkoj konstrukciji će se objasniti u sledećem poglavlju (10.4). U sledećoj ***always*** proceduri je navedena dodela koju treba izvršiti kada se desi imenovani događaj.

Primer 10-13: Vremenska kontrola čekanjem na imenovani događaj.

```
event received_data;

always @(posedge clock)
begin
        if (last_data_packet)
                ->received_data;
end

always @ (received_data)
data_buf={data_pkt[0], data_pkt[1], data_pkt[2], data_pkt[3]};
```

Primena pravilne metode vremenske kontrole (ivično okidanje) se može uopštiti tako da se izvršavanje dodele vezuje na promenu jednog signala iz grupe signala. Sličan je način definisanja ovakve vremenske kontrole, samo se pojedina imena signala povezuju sa ključnom reči ***or*** (primer 10-14).

Primer 10-14: Vremenska kontrola čekanjem na ivicu jednog od grupe signala.

```
always @(reset or clock or d)
begin
        if (reset)
                q=1'b0;
        else   if(clock)
                q=d;
end
```

Može da dovodi do nesporazuma to što je u gornjem primeru primenjena sinhronizacija dodele na ivicu signala, ipak će se na osnovu ovog opisa sintetizovati *D latch* kolo (tačka 3.1.1) sa okidanjem na nivo. Procedura ***always*** kreće na uzlaznoj ili silaznoj ivici jednog od signala *reset*, *clock* i *d* jer se kod kola sa okidanjem na nivo može očekivati promena u ponašanju kada se pomeni nivo ulaznih signala (a to se dešava kod ivice signala). Znači, ako se desi promena u jednom od tri ulazna signala, u zavisnosti od novih logičkih nivoa, ili će se izvršiti asinhroni reset *latch* kola (*q=1'b0*), ili će *latch* postati transparentan (*q=d*) ili se, u slučaju niskog nivoa *clock* signala, zadržava zatečeno stanje.

Pored ivičnog okidanja u *HDL* opisima se može koristiti i okidanje na nivo. To se ne radi sa znakom **@** već sa ključnom reči ***wait***. U primeru 10-15 inkrementiranje podatka *count* se dešava nakon svakih dvadeset vremenskih jedinica pod uslovom da je signal *count_enable* na visokom logičkom nivou.

Primer 10-15: Vremenska kontrola čekanjem na logički nivo (okidanje na nivo).

```
always
        wait (count_enable) #20 count=count+1;
```

10.4. *Uslovne dodele*

U struktuiranim procedurama primenom ključnih reči ***if*** i ***else*** se može definisati pod kojim uslovima treba izvršiti naznačenu dodelu i pod kojim uslovima treba izostaviti tu dodelu iz obrade. Postoje tri slučaja u opisivanju uslovnih dodela.

U prvom slučaju (primer 10-16) koristimo samo ključnu reč ***if***. Ako je u primeru *clock* signal na niskom logičkom nivou, izvršiće se dodela, inače ne.

Primer 10-16: Izvršavanje dodele vezano za tačan uslov.

```
if (!clock)    buffer=data;
```

Ako neke dodele treba da se izvršavaju pri tačnom uslovu a neke pri netačnom, koriste se ključne reči ***if*** i ***else*** (primer 10-17). Ako je uslov tačan, potrebno je izvršiti dve dodele navedene između ključnih reči ***begin*** i ***end***, u suprotnom slučaju izvršava se instrukcija ***$display***. Ova instrukcija ne služi za opisivanje nekog hardvera već prouzrokuje određenu akciju kod logičkog simulatora.

Primer 10-17: Vezivanje dodela za tačan i netačan uslov.

```
if (number_queued<MAX_Q_DEPTH)
begin
        data_queued=data;
        number_queued= number_queued+1;
end
else
        $display ("Queue full, Try again");
```

U trećem slučaju uslov može da ima više od dve vrednosti i pri svakoj vrednosti treba izvrštiti neku drugu dodelu. U primeru 10-18, pri vrednostima parametra *alu_control* od *0, 1* ili *2*, izvršavaju se određene propisane dodele, međutim ako nijedan uslov nije ispunjen, aktivira se instrukcija ***$display***. U prikazanoj jezičkoj konstrukciji koristi se lanac ključnih reči ***if – else if – else***.

Primer 10-18: Dodele vezane za više od dva uslova.

```
if (alu_control==0)
        y=x+z;
else if (alu_control==1)
        y=x-z;
else if (alu_control==2)
        y=x*z;
else
        $diplay ("Invalid alu control signal");
```

10.5. *Višestruka grananja*

Pisanje ***if-else*** struktura iz prethodne tačke u slučaju većeg broja uslova je komplikovano i dobijeni opis je nepregledan. U takvim situacijama prikladnije je korišćenje strukture sa ključnom reči ***case***.

10.5.1. Struktura *case*

Za formiranje ove strukture koriste se ključne reči ***case***, ***endcase*** i ***default***. Primena ključne reči ***default*** nije obavezna. Ako se koristi, može da se pojavi samo jednaput. U primeru 10-19 prethodnu ***if-else*** strukturu smo pretvorili u ***case*** strukturu da bi dobili jednostavniji opis.

Primer 10-19: HDL opis iz primera 10-18 prerađen na case strukturu.

```
reg [1:0] alu_control;

case (alu_control)
        2'd0: y=x+z;
        2'd0: y=x-z;
        2'd0: y=x*z;
        default: $diplay ("Invalid alu control signal");
endcase
```

Opis multipleksora (poglavlje 2.5) je najsvrsishodnije primenom ***case*** strukture. Primer 10-20 daje opis multipleksora 4/1 (prikazan je ceo modul).

Primer 10-20: HDL opis multipleksora 4/1 primenom case strukture.

```
module mux4_to_1(out, i0, i1, i2, i3, s1, s0);
output out;
input i0, i1, i2, i3;
input s1, s0;
reg out;
always @(s1 or s0 or i0 or i1 or i2 or i3)
case ({s1, s0})
        2'd0: out=i0;
        2'd1: out=i1;
        2'd2: out=i2;
        2'd3: out=i3;
        default: $diplay ("Invalid control signals");
endcase
endmodule
```

U gornjem primeru procedura ***always*** se svaki put aktivira kada se promeni logički nivo bilo kog od ulaznih nosioca podataka. Slično primeru 10-14 i ovde se radi o okidanju na nivo, ali procedura kreće kada se promeni logički nivo (znači: kod ivice). Kodovi pojedinih slučajeva su dobijeni povezivanjem jednobitnih selekcionih ulaza (s1, s0). Red sa ključnom reči default se može izostaviti, pošto nema uticaja na sintezu fizičkog kola. Izlazni nosilac podatka (out) je deklarisan da bude tipa registar, ali se ipak na osnovu datog *HDL* opisa sintetizuje kombinaciono kolo (multipleksor).

U ***case*** strukturi u pojedinim slučajevima umesto jedne dodele može da se javi više njih. Tada dodele treba grupisati sa ključnim rečima ***begin*** i ***end***. Pojedini bitovi u kodovima slučajeva mogu uzimati vrednosti *x* ili *z*, na taj način se neka kola mogu preciznije opisati.

10.5.2. Primena ključnih reči *casex casez*

Primenom ključnih reči ***casex*** i ***casez*** mogu se skratiti razni opisi sa grananjima. Pri korišćenju ključne reči ***casez*** u kodu slučaja na pojedinim pozicijama mogu da stoje z vrednosti i one neće imati uticaja na izvršavanje uslovne dodele. Kod ključne reči ***casex*** u kodu mogu biti i *x* i *z* vrednosti i te pozicije se neće razmatrati pri izvršavanju dodele. U primeru 10-21 dat je skraćeni opis jednog specijalnog kodera koji izlazni kod formira na osnovu toga da li na odgovarajućoj poziciji postoji logička jedinica, vrednosti ostalih bitova nemaju uticaja na kod.

Primer 10-21: Primena ključne reči casex za definisanje specijalnog kodera.

```
reg[3:0] encoding;
integer state;
casex (encoding)
        4'b1xxx: state=3;
        4'bx1xx: state=2;
        4'bxx1x: state=1;
```

```
        4'bxxx1: state=0;
        default: state=0;
endcase
```

10.6. *Petlje u opisima na nivou ponašanja*

Pri opisivanju na nivou ponašanja ***always*** procedure prouzrokuju stalno ponavljanje dodela za celo vreme simulacije ili (kod ostvarenog hardvera) za celo vreme rada kola. Primenom petlji u *Verilog HDL*-u ponavljanje dodela se može vezati za neki uslov. Na taj način se može podešavati broj ponavljanja ili vreme izvršavanja.

Postoje četiri ključne reči za definisanje petlji, to su: ***while***, ***for***, ***repeat*** i ***forever***. Petlje se mogu definisati samo u opisima na nivou ponašanja, prema tome mogu se pojaviti samo u ***initial*** i ***always*** procedurama.

10.6.1. Petlja tipa *while*

Ova petlja se definiše pomoću ključne reči ***while*** i izraza u zagradi pored te ključne reči. Dodele koje pripadaju petlji se ponavljaju dok izraz u zagradi daje tačnu vrednost. Dodele se neće izvršiti nijednom ako je izraz netačan u startu. Ako ima više od jedne dodele u petlji, oni treba da se smeste između ključnih reči ***begin*** i ***end***. U primeru 10-22 prikazan je deo jednog modula koji opisuje sedmobitni brojač pomoću petlje tipa ***while***.

Primer 10-22: Korišćenje petlje tipa while za opisivanje brojača.

```
integer count;
initial
begin
        count=0;
        while (count < 127)
        begin
                $display ("count=%d", count);
                count=count+1;
        end
end
```

Izraz u zagradi pored instrukcija ***$display*** definiše šta treba ispisati na monitor računara na kojoj se vrši simulacija. Pri svakom prolasku kroz petlju na monitoru se pojavi tekst *count*= i trenutna vrednost podatka *count* u decimalnom obliku.

10.6.2. Petlja tipa *for*

Ova petlja se definiše pomoću ključne reči ***for*** i izraza u zagradi pored te ključne reči. Izraz sadrži tri elementa:

- jedan početni uslov,
- proveru uslova za izlazak iz petlje,
- jednu dodelu koja modifikuje vrednost podatka kojom je određen momenat izlaska iz petlje.

Brojač dat u prethodnom primeru se jednostavije može opisati primenom ***for*** petlje (primer 10-23). Treba međutim reći da je petlja tipa ***while*** opštijeg karaktera i ima širu primenu.

Primer 10-23: Primena petlje tipa for za opis brojača.

```
integer count;
initial
        for (count=0; count<128; coun=count+1)
                $display ("count=%d", count);
```

10.6.3. Petlja tipa *repeat*

Ova petlja se definiše pomoću ključne reči ***repeat*** i brojčane vrednosti u zagradi pored te ključne reči. Umesto brojčane vrednosti može da se pojavi i neki identifikator ili neki izraz, ali se njegova vrednost sračuna pre ulaska u petlju i unutar petlje se smatra konstantom. Prema tome, petlja tipa ***repeat*** se koristi u situacijama kada dodele navedene u petlji treba izvršiti određeni broj puta. U primeru 10-24 dat je opis brojača iz prethodnih primera primenom petlje repeat.

Primer 10-24: Primena petlje tipa repeat za formiranje brojača.

```
integer count;
initial
begin
        count=0;
        repeat (128)
        begin
                $display ("count=%d", count);
                count=count+1;
        end
end
```

10.6.4. Petlja tipa *forever*

Ova petlja se definiše pomoću ključne reči ***forever***. Definicija ne sadrži uslov za izlazak iz petlje, zato se petlja u principu neograničeno izvršava (pri simulacijama trajaće do kraja simulacije, pri hardverskoj realizaciji trajaće sve dok kolo dobije napajanje). U primeru 10-25 formira se takt signal primenom petlje tipa ***forever***.

Primer 10-25: Primena petlje tipa forever za formiranje takt signala.

```
initial
begin
        clock=1'b0;
        forever #10 clock=~clock;
end
```

10.7. *Redni i paralelni blokovi*

Dodele koje čine celinu se grupišu u blokove. Dosad su mnogo puta korišćene ključne reči ***begin*** i ***end*** kojima se formiraju redni blokovi. U rednim blokovima dodele dolaze na izvršavanje u redosledu pisanja. Pored rednih blokova postoje i paralelni. Dodele u paralelnim blokovima dolaze na izvršavanje istovremeno (vreme izvršavanja može da se razlikuje samo ako su definisana kašnjenja).

10.7.1. Redni blokovi

Redni blokovi se počinju sa ključnom reči ***begin*** i završavaju se sa ključnom reči ***end***. Dodele navedene između tih ključnih reči se izvršavaju u onom redosledu kako su napisane. Novoj dodeli se ne pristupa sve dok prethodna nije izvršena. Zadate vrednosti kašnjenja su relativne vrednosti: kašnjenje navedeno za trenutno aktivnu dodelu se računa od završetka prethodne dodele.

U dosadašnjim primerima smo već na puno mesta koristili redne blokove. U primeru 10-26 navodimo jedan redni blok bez kašnjenja. Navedene četiri dodele se izvršavaju u redosledu pisanja ali se sve to dešava u $t=0$. Na taj način bitovi podataka y i w se formiraju od prethodno dodeljenih vrednosti podataka x i y.

Primer 10-26: Redni blok bez kašnjenja.

```
reg x,y;
reg [1:0] z, w;
```

```
initial
begin
        x=1'b0;
        y=1'b1;
        z={x,y};
        w={y,x};
end
```

U primeru 10-27 dodele u rednom bloku sadrže kašnjenje (izuzev prvog). Pošto su kašnjenja nadovezuju, prva dodela se izvršava u *t=0*, druga u *t=5*, treća u *t=15* a četvrta u *t=35*.

Primer 10-26: Redni blok sa kašnjenjima.

```
reg x,y;
reg [1:0] z, w;
initial
begin
        x=1'b0;
        #5 y=1'b1;
        #10 z={x,y};
        #20 w={y,x};
end
```

10.7.2. Paralelni blokovi

Dodelama u paralelnom bloku prethodi ključna reč ***fork*** a na kraju stoji ključna reč ***join***. Dodele iz paralelnog bloka dolaze na izvršenje istovremeno, slično kao kod neblokirajućih dodela (tačka 10.2.2) ali se u ovom slučaju ne vrši memorisanje podataka pri ulasku u blok.

Vremena izvršavanja možemo podešavati zadavanjem kašnjenja, čekanjem na događaj, ivičnim okidanjem i okidanjem na nivo (tačka 10.3.2). Kašnjenja se računaju u odnosu na momenat ulaska u blok. Redosled navođenja dodela nema uticaja na redosled izvršavanja.

U primeru 10-27 je ponovljen prethodni primer ali umesto rednog bloka koristi se paralelni. Na kraju nosioci podataka dobijaju iste vrednosti, ali menjaju se vremena izvršavanja. Navedena kašnjenja se tumače kao apsolutne vrednosti.

Primer 10-27: Paralelni blok sa kašnjenjima.

```
reg x,y;
reg [1:0] z, w;
initial
fork
        x=1'b0;
        #5 y=1'b1;
        #10 z={x,y};
        #20 w={y,x};
join
```

Pri korišćenju paralelnog bloka može da nastupi efekat protrčavanja ako sa dva ili više dodela pokušavamo istovremeno dodeliti vrednost nekom nosiocu podatka. Takođe može biti problematično ako u istom bloku definišemo vrednost nekog podatka i koristimo isto u nekoj drugoj dodeli. U primeru 10-28 vrednosti *z* i *w* su nesigurne u momentu napuštanja bloka jer se ne zna da li su vrednosti *x* i *y* definisane na vreme.

Primer 10-28: Paralelni blok bez kašnjenja (javlja se efekat protrčavanja).

```
reg x,y;
reg [1:0] z, w;
```

```
initial
fork
        x=1'b0;
        y=1'b1;
        z={x,y};
        w={y,x};
join
```

10.7.3. Kombinovani blokovi

Redni i paralelni blokovi se mogu i kombinovati (primer 10-29). U prikazanom slučaju glavni blok je rednog tipa, dok je unutrašnji blok paralelnog tipa. Razume se da će se prvo izvršiti prva dodela rednog bloka, zatim dolaze dve dodele unutar paralelnog bloka i na kraju sledi zadnja dodela iz rednog bloka. Dodele u paralelnom bloku dolaze na red u istom momentu ali zbog navedenih kašnjenja izvršavaju se tek u momentima *t=5* i *t=10*.

Primer 10-29: Kombinacija rednog i paralelnog bloka.

```
initial
begin
        x=1'b0;
        fork
                #5 y=1'b1;
                #10 z={x,y};
        join
        #20 w={y,x};
end
```

10.7.4. Imenovani blokovi

Blokovi se mogu imenovati. Kod tako dobijenih imenovanih blokova:

- možemo deklarisati lokalne nosioce podataka,
- pojedinim nosiocima podataka možemo pristupiti preko hijerarhijskih imena,
- možemo prekinuti izvršavanje.

U primeru 10-30 nosilac podatka *i* se pojavljuje unutar dva bloka, ali pošto su ti blokovi imenovani, ne dolazi do zabune. Čak ni to nije problem što nosioci podataka nisu istog tipa.

Primer 10-30: Primena imenovanih blokova.

```
module top;

initial
begin: block1
integer i;
.........
end

initial
fork: block2
reg i;
.........
join
endmodule
```

10.7.5. Prekidanje izvršavanja imenovanog bloka

Korišćenjem instrukcije ***disable*** može se prekinuti imenovani blok. Tako se mogu izostaviti izvesne dodele ili se može izaći iz petlje. U primeru 10-31 prikazan je način izlaska iz

petlje tipa ***while***. Ako je naredni bit vektora *flag* jedinica, prekida se izvršenje bloka pod imenom *block1*. Primenom instrukcije ***disable*** može se prekinuti bilo koji blok, ne samo onaj u kome je navedena instrukcija ***disable***. Ovo je značajno odstupanje od pravila softverskih jezika, kod kojih se može prekinuti samo onaj program/potprogram koji se tog momenta izvršava.

Primer 10-31: Prekidanje imenovanog bloka.

```
initial
begin
        flag=16'b0010_0000_0000_0000;
        i=0;
        begin: block1
        while (i<16)
                begin
                        if (flag[i])
                        begin
                                $display ("True bit at element number %d", i);
                                disable block1;
                        end
                        i=i+1;
                end
        end
end
```

11. Literatura

[1] Szittya Ottó: Digitális és analóg technika informatikusoknak, I i II tom, LSI Oktatóközpont, Budapest, 1999.

[2] Dejan Živković, Miodrag Popović: Impulsna i digitalna elektronika, Nauka, Beograd, 1993.

[3] Samir Palnitkar: Verilog HDL, A Guide to Digital Design and Synthesis, Sunsoft Press, 1996.

[4] Michael Ciletti: Advanced Digital Design with the Verilog HDL, Prentice Hall, 2003.

[5] Morris Mano: Digital Design, III izdanje, Prentice Hall, 2002.

[6] Matijevics István: Digitális Technika, Szabadkai Műszaki Főiskola, Szabadka, 2003.

[7] Janovics Sándor, Tóth Mihály: A logikai tervezés módszerei, Műszaki Könyvkiadó Budapest, 1973.

[8] Arató Péter: A logikai rendszerek tervezése, III izdanje, Tankönyvkiadó, Budapest, 1990.

[9] Peter Ammon: Kapumátrix áramkörök, Berendezésorientált áramkörök, Műszaki Könyvkiadó, Budapest, 1989.

[10] Radomir Stanković, Milena Stanković: Logičko projektovanje, Nauka, Beograd, 1991.

[11] Tihomir Aleksić: Logička sinteza digitalnih mreža, Naučna knjiga, Beograd, 1975.

[12] Ripka Gábor: Felületre szerelhető alkatrészek, Műszaki Könyvkiadó, Budapest, 1992.

[13] Texas Instruments: Logic Selection Guide, Dallas, Texas, 2003.

[14] Fairchild: Logic Selection Guide, 2003.

[15] www.ti.com

[16] www.fairchildsemi.com

[17] www.prenhall.com

[18] www.xilinx.com

[19] www.altera.com

[20] www.atmel.com

[21] www.latticesemiconductor.com

[22] www.simucad.com