Univerzitet u Sarajevu
Prirodno–matematički fakultet
Odsjek za matematiku

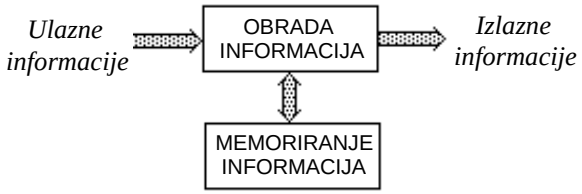
Dr. Željko Jurić

Logičke osnove digitalnih i računarskih sistema

Interna skripta – radna verzija

1. Osnovni pojmovi o računarskim sistemima

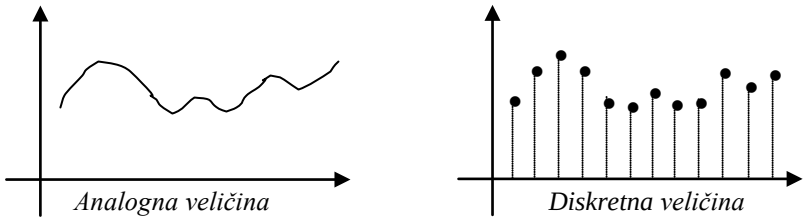
Da bismo uopće išta rekli o računarskim sistemima, moramo prvo definirati šta se uopće podrazumijeva pod pojmom računar. Generalno, računar u najširem smislu predstavlja uređaj za *automatsku obradu podataka i informacija*. Informacije koje obrađujemo nazivamo *ulazne informacije*, dok informacije koje dobijamo kao rezultat obrade nazivamo *izlazne informacije*. Zajedno sa postupkom obrade informacija, obično se uporedo vrši i postupak *uskladištavanja* (*memoriranja*) *informacija*. Stoga, cjelokupan proces obrade informacija možemo shematski prikazati kao na sljedećoj slici:



Sasvim je jasno da je obradu informacija moguće vršiti ručno, eventualno uz pomoć jednostavnijih mehaničkih ili elektronskih pomagala, poput mašina za pisanje, džepnih kalkulatora, itd. Međutim, ručna obrada informacija ne može da obradi veliku količinu informacija, spora je i podložna greškama, a pored toga često se oslanja na krajnje monotone i dosadne postupke. Stoga se u savremenoj obradi informacija javlja potreba za *računarom*, kao sredstvom za automatsku obradu informacija. U skladu sa tim, možemo dati opću definiciju šta je uopće računar:

Računar je svaki uređaj koji je sposoban da prima podatke od korisnika, da ih memorira i obrađuje prema zadanim uputama (programu) i da saopći korisniku rezultate obrade.

Danas kada god upotrijebimo riječ "računar", gotovo isključivo mislimo na tzv. ***digitalni računar***. Da bismo mogli objasniti šta se podrazumijeva pod digitalnim računarom, moramo prvo objasniti razliku između analognih i digitalnih veličina. Naime, u savremenim tehničkim sistemima, informacije se obično prenose u vidu signala koji predstavljaju promjene neke fizikalne veličine, najčešće električne prirode. Ukoliko je stanje posmatrane veličine poznato *u svakom trenutku vremena* unutar posmatranog intervala, takve veličine nazivamo ***analogne veličine***, zbog činjenice da se one lako mogu simulirati nekim drugim promjenljivim veličinama koje ispunjavaju slične (analogne) zakonitosti. S druge strane, ukoliko je stanje posmatrane veličine poznato samo *u tačno određenim vremenskim trenucima*, za takvu veličinu kažemo da je ***diskretna veličina***. One diskretne veličine koje pored navedenog mogu uzimati samo vrijednosti iz *unaprijed određenog i konačnog skupa vrijednosti*, nazivaju se ***digitalne veličine***, zbog toga što se njihove vrijednosti najčešće predstavljaju u brojčanom obliku pomoću odgovarajućih cifara (riječ "digit" zapravo znači "cifra").



Prilikom *analogne obrade podataka i informacija* koriste se odgovarajući matematski metodi, kao i eksperimentalna provjeravanja. U tom cilju se često konstruiraju modeli pomoću komponenti koje po obliku, strukturi i osobinama obično nisu slične, ali su po matematičkim karakteristikama iste kao i odgovarajući elementi stvarnog objekta nad kojim vršimo obradu. Ovakav postupak obrade nazivamo *analogna simulacija*, a uređaj za izvođenje opisanog postupka naziva se *analogni računar*. Analogni računari su prilično neprecizni, jer njihova tačnost izuzetno ovisi o tačnosti upotrijebljenih komponenti kojima se simulira objekat nad kojim vršimo obradu, i njima se dalje u ovom tekstu nećemo baviti. S druge strane, pri *digitalnoj obradi podataka*, svi podaci koji se obrađuju pretvaraju se u brojeve, tako da

se suština obrade svodi na raznovrsna izračunavanja. Tako, na primjer, *digitalni računar* prima podatke u vidu specijalno uređenih brojeva, obavlja sa njima određeni niz izračunavanja, i na kraju formira nove brojeve kao rezultat izvršene obrade. Da bi građa digitalnog računara bila što jednostavnija, sva izračunavanja koja se obavljaju svode se isključivo na najelementarnije operacije, o čemu ćemo kasnije detaljnije govoriti. Pri tome treba napomenuti da gotovo svaki digitalni računar posjeduje i dijelove koji su po svom principu rada zapravo analogni (na primjer, većina ulaznih i izlaznih uređaja računara su pretežno analogni uređaji). Ipak, glavnina digitalnih računara je digitalne prirode. U nastavku ćemo se baviti isključivo digitalnim računarima.

Mada je digitalni računar uređaj za obradu isključivo digitalnih veličina i informacija, većina informacija koje nas okružuju kao i podaci sa kojima neprestano baratamo nisu digitalne već analogne prirode. Da bismo mogli upotrebljavati digitalni računar za obradu analognih informacija, računar mora prvo da pretvori ulazne analogne informacije u digitalne informacije. Ovaj postupak se naziva ***analogno/digitalna*** (***A/D***) ***konverzija***. Tako pretvorene digitalne informacije obrađuju se u digitalnom računaru, nakon čega se dobivene digitalne informacije ponovo pretvaraju u analogne informacije pomoću postupka koji se naziva ***digitalno/analogna*** (***D/A***) ***konverzija***. A/D i D/A konverzija obavljaju se unutar dijelova računara koji spadaju u grupu *ulaznih* odnosno *izlaznih* jedinica računara, o čemu ćemo kasnije detaljnije govoriti. Proces obrade analognih informacija uz pomoć digitalnog računara shematski se može prikazati sljedećom slikom:

Postupak pretvaranja analognih veličina u digitalne (A/D konverzija) obavlja se u dvije etape. U prvoj etapi uzimaju se uzorci vrijednosti posmatrane analogne veličine u tačno određenim (najčešće ravnomjerno raspoređenim) vremenskim trenucima, dok se vrijednosti posmatrane analogne veličine u ostalim vremenskim trenucima prosto *ignoriraju*. Ovaj postupak naziva se ***diskretizacija*** (susreću se i nazivi ***uzorkovanje*** i ***semplovanje***). U drugoj etapi se vrijednosti uzetih uzoraka zaokružuju na najbližu vrijednost uzetu iz unaprijed zadanog konačnog skupa vrijednosti (koji se nazivaju ***kvanti***), a zatim se tako zaokružene vrijednosti zapisuju u vidu brojeva. Ovaj postupak naziva se ***digitalizacija*** (ili ***kvantizacija***).

Kako se prilikom diskretizacije ignorira veliki broj (praktički, beskonačno mnogo) vrijednosti posmatrane analogne veličine, prirodno je postaviti pitanje da li prilikom diskretizacije dolazi do *gubitka informacija*. Do posve neočekivanog odgovora došli su prije više od 50 godina *Claude Shannon* i *Henry Nyquist*, koji su dokazali da u slučaju da je, pojednostavljeno rečeno, brzina uzimanja uzoraka barem dva puta veća od maksimalne brzine promjene posmatrane analogne veličine, ne dolazi ni do kakvog gubitka informacija u postupku diskretizacije (ovo je čuveni ***Shannon–Nyquistov teorem o uzorkovanju***), odnosno, sve originalne informacije je moguće jednoznačno rekonstruirati iz diskretizirane veličine. S druge strane, kvantizacija *uvijek* dovodi do gubitka informacija, ali se gubitak može učiniti *po volji malim* ukoliko se kvanti izaberu tako da su dovoljno blizu jedan drugom. Postupak pretvaranja digitalnih (ili općenitije diskretnih) veličina u analogne naziva se i ***interpolacija***, i on nikada ne dovodi ni do kakvog gubitka informacija.

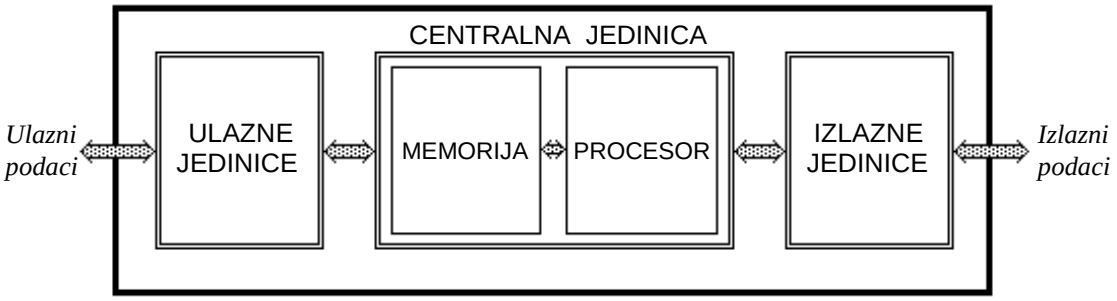
Gotovo svi današnji digitalni uređaji zasnovani su na postupcima diskretizacije i digitalizacije. Posmatrajmo na primjer *muzički signal*, koji je tipičan primjer *analogne veličine*. Danas se muzika najčešće zapisuje na *kompakt diskove* (CD-ove), koji su tipični *digitalni uređaji*. Prilikom zapisa muzike na kompakt diskove, vrši se *diskretizacija* uzimanjem uzoraka muzičkog signala 44000 puta u sekundi (kako ljudsko uho ne čuje promjene zvuka brže od 20000 titraja u sekundi, po teoremu o uzorkovanju do gubitka informacije ne dolazi ukoliko uzorke zvuka uzimamo 40000 puta u sekundi ili brže). Zatim se uzeti uzorci *kvantiziraju* i zapisuju na disk kao skupina brojeva (o samom načinu zapisa tih brojeva govorićemo kasnije). Prilikom kasnije reprodukcije kompakt diska, uređaj za reprodukciju vrši *interpolaciju* zapisanih vrijednosti čime rekonstruira originalni muzički signal.

Način na koji je građen računar i na koji su međusobno povezani njegovi dijelovi nazivamo ***arhitektura računara***. Mada postoje različite arhitekture računara, većina računara koji su danas u upotrebi zasnivaju se na arhitekturi koju je još 1946. godine predložio *John von Neumann* (ili na neznatnim modifikacijama ove arhitekture), stoga ovu arhitehturu nazivamo ***Von Neumannova arhitektura***. U nastavku ćemo govoriti isključivo o računarima zasnovanim na Von Neumannovom modelu.

Da bi računar mogao da obavlja predviđenu funkciju automatske obrade podataka, on očito mora posjedovati barem tri grupe uređaja:

- ***Ulazne jedinice***, preko kojih računar prima podatke od korisnika;
- ***Izlazne jedinice***, preko kojih računar saopćava korisniku rezultate obrade;
- ***Centralnu jedinicu***, u kojoj se obrađuju podaci, i u koju se smještaju podaci koji se obrađuju, zajedno sa instrukcijama (programom) po kojima se vrši obrada.

Centralnu jedinicu dalje možemo podijeliti na ***memoriju***, koja služi za smještanje podataka i programa, i ***centralnu procesnu jedinicu*** (koja se skraćeno naziva i samo procesna jedinica, ili, još češće, ***procesor***), koja vrši obradu podataka. Centralna procesna jedinica skraćeno se označava sa ***CPU*** (Central Processing Unit). Von Neumannov model računara shematski je prikazan na sljedećoj slici:



Ulazne jedinice, centralnu jedinicu i izlazne jedinice zajedničkim imenom nazivamo ***hardver*** računara. S druge strane, programi za računar kao i podaci koje ti programi obrađuju čine ***softver*** računara. Hardver i softver zajedno sačinjavaju jedinstvenu cjelinu koju nazivamo ***računarski sistem***. Tako, možemo reći da hardver zapravo predstavlja materijalni, opipljivi dio računarskog sistema, dok softver predstavlja, uvjetno rečeno, njegov intelektualni, neopipljivi dio.

U tipične ulazne jedinice spadaju:

- ***Prekidači***, koji predstavljaju veoma primitivne ulazne uređaje;
- ***Tastatura***, koja predstavlja najrasprostranjeniji ulazni uređaj;
- ***Miš***, koji je tipičan primjer pokaznog ulaznog uređaja;
- ***Skener,*** koji predstavlja uređaj za prenos slike sa papirnog medija u računar;
- ***Mikrofon***, koji se može koristiti za komunikaciju sa računarom putem govora.

U tipične izlazne jedinice spadaju:

- ***Svjetlosni indikatori***, koji predstavljaju veoma primitivne izlazne uređaje;
- ***Monitor*** (***ekran***), koji predstavlja najrasprostranjeniji izlazni uređaj;
- ***Štampač*** (***printer***), koji prenosi sliku formiranu u računaru na papirni medij;
- ***Crtač*** (***ploter***), koji iscrtava sliku na papiru pomoću posebnih pisaljki;
- ***Zvučnik***, za prezentaciju podataka u zvučnoj formi.

Uz pretpostavku da su čitatelju odnosno čitateljki od ranije poznate osnovne karakteristike ulaznih i izlaznih jedinica, najveći dio izlaganja koja slijede biće posvećen građi i funkcioniranju centralne jedinice, koja zapravo i predstavlja najkompleksniji dio svakog računara.

(?) Pitanja i zadaci

1.1 Definirajte računar kao sredstvo za automatsku obradu informacija.

1.2 Objasnite razlike između analognih, diskretnih i digitalnih veličina. Koje od ovih veličina dominiraju u realnom svijetu u kojem živimo?

1.3 Objasnite osnovne razlike između analognih i digitalnih računara.

1.4 Opišite kako se vrši pretvaranje iz analognih u digitalne veličine i obrnuto.

1.5 Objasnite šta predstavljaju pojmovi diskretizacija, kvantizacija i interpolacija i čemu služe.

1.6 Objasnite u osnovnim crtama šta tvrdi Shannon-Nyquistov teorem o uzorkovanju.

1.7 Smatra se da prilikom prenosa glasa kroz telefonsku mrežu ne dolazi do promjena bržih od 3400 titraja u sekundi. Koliko je često potrebno uzimati uzorke glasa da bi se ostvario digitalni prenos glasa bez gubitka informacija?

1.8 Opišite ukratko Von Neumannov model arhitekture digitalnog računara.

1.9 Objasnite šta predstavlja hardver, a šta softver računarskog sistema.

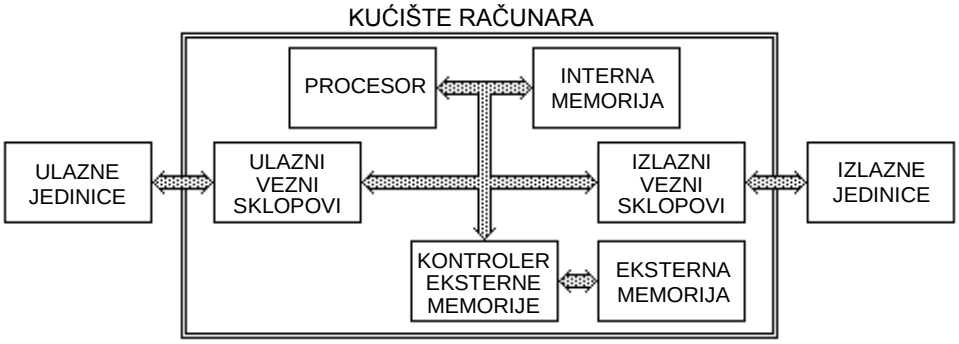
1.10 Navedite koja je uloga memorije, a koja procesora u računarskim sistemima.

1.11 Navedite koja je uloga ulaznih, a koja izlaznih jedinica u računarskim sistemima.

1.12 Navedite barem tri ulazne i barem tri izlazne jedinice.

2. Detaljnija struktura centralne jedinice računara

Većina savremenih računara građena je po već opisanom Von Neumannovom modelu. Međutim, za potpunije razumijevanje hardverske strukture savremenih računara, Von Neumannov model nije dovoljno precizan. Prvo, iz ovog modela moglo bi se zaključiti da se ulazne i izlazne jedinice *vezuju direktno na centralnu jedinicu računara*. Stvarne ulazne i izlazne jedinice, poput tastature i monitora, nikada se ne priključuju direktno na centralnu jedinicu, nego se njihovo priključenje vrši pomoću uređaja koji se nazivaju ***ulazni*** odnosno ***izlazni vezni sklopovi*** (susreće se i naziv ***kontroleri ulaznih*** odnosno ***izlaznih uređaja***). Logički gledano, ulazni i izlazni vezni sklopovi čine sastavni dio ulaznih i izlaznih uređaja *u širem smisliu*, ali kako se oni nalaze unutar kućišta računara zajedno sa procesorom i memorijom (za razliku od npr. same tastature i monitora koji su izvan kućišta, i koje možemo nazvati ulazne odnosno izlazne jedinice *u užem smislu*), fizički gledano izgleda kao da oni čine dio centralne jedinice. Drugo, Von Neumannov model predviđa samo jednu vrstu memorije, koja *neposredno* komunicira sa procesorom (komunikacija između memorije i procesora biće detaljno objašnjena kasnije). U stvarnosti, imamo dvije vrste memorija: ***internu memoriju*** koja zaista neposredno komunicira sa procesorom, i ***eksternu memoriju***, koja sa procesorom komunicira *posredno*, posredstvom uređaja koji se nazivaju ***kontroleri eksterne memorije***. Stoga, razrađenu hardversku strukturu savremenih računara možemo prikazati sljedećom slikom:



Interne memorije se još nazivaju i ***unutrašnje***, ***centralne*** ili ***radne memorije***, dok se *eksterne memorije* još nazivaju i ***vanjske***, ***periferne*** ili ***masovne memorije***. Osnovni kriterij po kojem se memorije dijele na interne i eksterne je način na koji ih vidi procesor. Interne memorije mogu neposredno komunicirati sa procesorom, dok eksterne memorije, iako logički gledano spadaju u memorije, *procesor uvijek vidi kao ulazne odnosno izlazne uređaje*, i komunicira sa njima na isti način kao npr. sa tastaturom i ekranom koristeći odgovarajuće *vezne sklopove* (kontrolere eksterne memorije). Zbog toga se pojedini uređaji koji spadaju u eksterne memorije (poput raznih vrsta diskova), ponekad svrstavaju u ulazne ili izlazne uređaje, što nije tačno sa logičkog aspekta (mada sa njima imaju mnogo sličnosti).

Interne memorije se danas grade isključivo na elektronskom principu rada, u vidu integriranih kola (čipova), dok su se ranije susretale i interne memorije koje se zasnivaju na magnetskom principu rada. Ove memorije su praktično neprenosive iz računara u računar, jer su čvrsto povezane sa procesorom i ostalim dijelovima računara. Brzina pristupa podacima koji su smješteni u internim memorijama neuporedivo je veća nego brzina pristupa podacima na eksternim memorijama. Glavno svojstvo ovih memorija je činjenica da se programi koji su u njima pohranjeni mogu neposredno izvršavati, s obzirom na činjenicu da im procesor može direktno pristupati.

Interne memorije se obično dijele na dvije skupine, nazvane ***RAM*** i ***ROM*** memorije. Kod *ROM memorija* (Read Only Memory), podaci su u memoriju upisani još u fazi proizvodnje nekim postupkom (poput progorijevanja, bušenja, itd.) koji ostavlja trajne fizičke promjene u mediju koji služi za memoriranje i stoga su podaci zapisani u njima trajni. Što je još bitnije, jasno je da se iz ovakve memorije podaci mogu samo čitati, ali ne i upisivati, odakle i potiče njeno ime. Mada se naziv *ROM* prvenstveno odnosi na vrstu interne memorije, neke eksterne memorije su također ROM tipa, u smislu da se podaci sa njih mogu samo čitati (npr. *CD ROM*). Kod *RAM memorija* (Random Access Memory)

podaci se mogu nasumice upisivati i čitati iz bilo kojeg dijela memorije i po potrebi brisati, odakle i potiče njihovo ime. Pri tome brzina pristupa pojedinim podacima pohranjenim u *RAM* memorije ne ovisi od mjesta gdje je podatak zapisan (po čemu se *RAM* memorije bitno razlikuju od recimo diskova). Nekada se nazivaju i ***R/W memorije*** (Read/Write), što je donekle neprecizno, jer za većinu eksternih memorija također možemo reći da su *R/W* tipa. Obično se tvrdi da *sve RAM memorije brišu kompletan svoj sadržaj isključivanjem napajanja*. Strogo uzevši, ovo vrijedi samo za *RAM* memorije koje su zasnovane na elektronskom principu rada, dok to ne mora biti slučaj kod *RAM* memorija koje su zasnovane na nekom drugom principu rada (npr. magnetnom). Ipak, većina današnjih *RAM* memorija koristi isključivo elektronski princip rada.

Umjesto klasičnih *ROM* memorija danas se uglavnom koriste njihovi bliski srodnici ***PROM***, ***EPROM*** i ***EEPROM***. *PROM memorija* (Programable Read Only Memory) je u trenutku proizvodnje prazna. Korisnik u nju može upisati sadržaj pomoću uređaja nazvanog ***PROM programator***, međutim taj sadržaj se može upisati *samo jedanput* (i ne može se izbrisati) zbog toga što postupak upisivanja ima destruktivno dejstvo na materijal od kojeg je pravljena memorija. Kod *EPROM memorija* (Erasable Programable Read Only Memory) postupak upisa je sličan kao kod PROM memorija, ali način na koji je izvršen upis nije potpuno destruktivan tako da postoji način da se memorija vrati u prvobitno stanje i time izbriše. Kod elektronskih *EPROM* memorija to brisanje se najčešće izvodi izlaganjem čipa jakoj svjetlosti izvjesno vrijeme. *EEPROM memorije* (Electrical Erasable Programable Read Only Memory) su prilično slične *EPROM*-ima, ali kod njih postoji mogućnost brisanja i pomoću električne struje (tipično mnogo jače nego pri normalnom radu). Logički, *EEPROM* liči na *RAM* memoriju, jer se u njega ipak mogu upisivati podaci proizvoljan broj puta čisto električkim putem (dakle, bez potrebe za vađenjem čipa iz računara). Stoga se *EEPROM* memorije ponekad nazivaju i ***Flash RAM***. Međutim, njihov sadržaj je postojan kao kod ROM memorija, tj. ne briše se tek tako sam od sebe isključenjem napajanja. Još bitniji razlog zbog čega se *EEPROM* radije svrstava u *ROM* memorije je činjenica da se upis podataka u *EEPROM* vrši neuporedivo sporije (i do milion puta) od upisa u klasične *RAM* memorije. Pored toga, upisivanje podataka u *EEPROM* kao i brisanje podataka iz *EEPROM*-a ne može se vršiti u proizvoljnom redoslijedu, nego isključivo *po blokovima* (tj. uvijek se upisuje odnosno briše čitav blok podataka odjednom). Zbog svega rečenog, *EEPROM* treba koristiti u situacijama kada postoji principijelna potreba za mijenjanjem upisanih podataka, ali koja će se vršiti vrlo rijetko.

U današnjim savremenim računarima umjesto klasičnog *ROM*-a najčešće se susreće upravo *EEPROM*. Nažalost, to otvara izvjesne mogućnosti zloupotrebe, jer je moguće napraviti takve zlonamjerne programe (tipičan primjer je program-virus poznat pod nazivom *CIH* ili *Chernobyl*) koji će prebrisati cjelokupan sadržaj upisan u *EEPROM*. Na taj način računar može postati potpuno neupotrebljiv (sve do zamjene izbrisanog *EEPROM* čipa sa čipom sa ispravnim sadržajem), s obzirom da se u *EEPROM*-u tipično nalaze podaci i programi bez kojih računar uopće ne može da započne rad.

Eksterne memorije najčešće nisu građene isključivo na elektronskom principu rada, nego na *elektromehaničkom* principu rada (poput *magnetnih diskova*) ili *optičkom* (preciznije, *optomehaničkom*) principu rada (poput *optičkih kompakt diskova*). Obično posjeduju pokretne dijelove, koji rotiraju za vrijeme rada, zbog čega je brzina pristupa podacima na njima neuporedivo manja nego kod internih memorija. S druge strane, eksterne memorije su uvijek u manjem ili većem stepenu *prenosive*, i kapacitet im je najčešće znatno veći od kapaciteta internih memorija. Kako procesor nema direktan pristup sadržaju eksterne memorije, programi pohranjeni na vanjskoj memoriji ne mogu se neposredno izvršavati, nego se moraju *prethodno prebaciti u internu memoriju*. Pored toga, kod eksternih memorija često razlikujemo sam ***memorijski medij*** na kojem su pohranjene informacije (npr. disketa) od odgovarajućeg ***pogonskog uređaja*** (***drajva***) koji vrši čitanje informacija sa medija, odnosno upis informacija na medij. U najvažnije eksterne memorije spadaju:

- ***Disketa*** (***flopi disk***), koja predstavlja jednostavan medij eksterne memorije zasnovane na magnetskom principu rada. Diskete su po građi okrugle ploče presvučene tankim slojem magnetskog materijala, upakovane u četvrtasto plastično kućište. Zovu se i ***savitljivi*** (***gipki***) ***diskovi***, što je posljedica građe prvih disketa, koje su bile upakovane u veoma savitljivo kućište (košuljicu). Danas diskete polako izlaze iz upotrebe, zbog svog prilično malog kapaciteta (po današnjim mjerilima).

- ***ZIP disk***, koji predstavlja modifikaciju diskete sa znatno povećanim kapacitetom.
- ***Hard disk*** (***tvrdi disk***, ***fiksni disk***) koji je po građi sličan disketi, samo što nije savitljiv i izrađen je od krutih materijala. Za razliku od disketa, okreće se velikom brzinom (stoga je pristup podacima na njemu mnogo brži), i upakovan je u hermetički zatvoreno kućište (zajedno sa pogonskim mehanizmom). Ovo kućište je čvrsto pričvršćeno na kućište samog računara, tako da su ovi diskovi praktično neprenosivi (mogu se prenijeti iz računara u računar samo zajedno sa kućištem diska i cjelokupnim pogonskim mehanizmom).
- ***Kompakt disk*** (***CD***), koji predstavlja medij eksterne memorije zasnovane na optičkom principu rada (podaci se čitaju i upisuju uz pomoć *lasera*). Postoje tri vrste kompakt diskova: ***CD ROM*** (na njih su podaci upisani u fazi proizvodnje, i ne mogu se mijenjati niti brisati), ***CD–R*** (Recordable) na koje se podaci mogu upisati *jedanput* (poput *PROM* memorija), i ***CD–R/W*** (Read/Write) diskovi na koje se podaci mogu upisivati i brisati *neograničen broj puta*. Pristup podacima na kompakt diskovima je mnogo sporiji nego pristup podacima na hard disku, dok se upis podataka na upisive diskove vrši izrazito sporo.
- ***DVD*** (Digital Versatile Disc) ***disk***, koji predstavlja modifikaciju kompakt diskova sa znatno uvećanim kapacitetom.
- ***Magnetne trake***, koje predstavljaju traku presvučenu magnetnim materijalom koja je praktično identična kao traka koja se koristi za snimanje muzike u kasetofonima. Najveća mana magnetnih traka je izuzetna sporost u pristupu podacima, zbog potrebe da se traka prethodno premota na odgovarajuće mjesto. Mada spominjanje magnetnih traka danas može djelovati posve zastarjelo, treba napomenuti da se one u pojedinim izvedbama (npr. ***strimer trake***) zbog svoje male cijene i ogromnog kapaciteta i danas masovno koriste za arhiviranje velikih količina podataka kojima se neće često pristupati (npr. za čuvanje rezervnih kopija važnih podataka).

U eksterne memorije možemo svrstati i ***Flash kartice*** (engl. Flash Card), koje se često u današnjem žargonu nazivaju engleskim nazivom ***USB memory stick***, s obzirom da se na računar priključuju preko univerzalne utičnice namijenjene za priključenje svih uređaja koji sa računarom komuniciraju koristeći tzv. ***serijski prenos podataka***, a koja je poznata pod nazivom ***USB*** (engl. Universal Serial Bus). Mada bi se Flash kartice po građi mogle svrstati u interne memorije (one nisu ništa drugo nego EEPROM čipovi sa odgovarajućim kontrolerom), svrstavamo ih u eksterne memorije s obzirom da se priključuju na računar spolja, preko odgovarajućeg priključka, da im procesor ne pristupa direktno nego preko kontrolera, da se programi na njima ne mogu neposredno izvršavati nego se moraju prebacivati u internu memoriju (zbog nepostojanja neposredne komunikacije sa procesorom), da su namijenjene za masovno smještanje podataka kao i za transfer podataka između računara, i da ih računar suštinski posmatra kao i svaki drugi uređaj eksterne memorije (npr. hard disk). Odavde vidimo da za određivanje da li je memorija interna ili eksterna nije toliko bitna sama njena građa ili princip rada, već način njene komunikacije sa procesorom, kao što smo već naglasili. Ipak, građa većine eksternih memorija drastično se razlikuje od građe većine internih memorija.

(?) Pitanja i zadaci

2.1 Objasnite zbog čega Von Neumannov model arhitekture računara nije dovoljno adekvatan za potrebe opisivanja svih aspekata građe današnjih digitalnih računara.

2.2 Objasnite šta su ulazni odnosno izlazni vezni sklopovi.

2.3 Objasnite osnovne sličnosti i razlike između eksternih i internih memorija.

2.4 Kako se još nazivaju eksterne odnosno interne memorije?

2.5 Objasnite šta predstavljaju kontroleri eksternih memorija.

2.6 Objasnite razlike između RAM i ROM memorija.

2.7 Objasnite kakva je razlika između ROM-a, PROM-a, EPROM-a i EEPROM-a.

2.8 Objasnite u čemu je suštinska razlika između RAM-a i EEPROM-a.

2.9 Kakve su prednosti i mane upotrebe EEPROM-a u odnosu na ROM, PROM ili EPROM?

2.10 Navedite nekoliko najvažnijih vrsta eksternih memorija.

2.11 Šta nazivamo pogonskim uređajem (drajvom) kod eksternih memorija?

2.12 Da li sve eksterne memorije uvijek posjeduju razdvojen memorijski medij i pogonski uređaj? Argumentirajte odgovor.

2.13 Šta su Flash kartice?

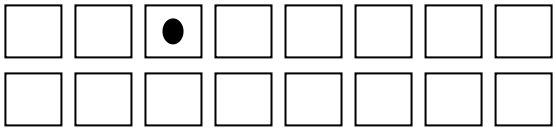
3. Jedinice za mjerenje količine informacije i kapaciteta memorije

Do značajnog napretka u razvoju digitalne tehnike dovela je spoznaja da se količina informacije može precizno mjeriti (za koju je najviše zaslužan *Claude Shannon*, osnivač matematičke teorije informacija). Od davnina je poznato da se svaka informacija može prenijeti samo putem odgovora tipa DA ili NE. Tako su, jednom prilikom, stari Rimljani zarobili neprijateljskog vojnika koji je bio nijem, tako da nije mogao odgovarati na postavljena pitanja. Međutim, Rimljani su uspjeli saznati sve informacije koje su ih zanimale, postavljajući samo pitanja DA/NE tipa, na koje je zarobljeni vojnik odgovarao klimanjem glave. Stoga je usvojeno da se kao mjera za količinu informacije definira *najmanji broj pitanja DA/NE tipa koje je potrebno postaviti da bi se prijemom odgovora na ta pitanja postigao isti efekat kao i prijemom posmatrane informacije*. Jedinica za mjerenje količine informacije naziva se ***bit*** (vidjećemo da se ista jedinica koristi i za mjerenje količine memorije). Jedan bit je *ona količina informacije koja se može dobiti odgovorom na jedno pitanje DA/NE tipa*.

Količina informacije o nekom događaju u direktnoj je vezi sa vjerovatnoćom samog događaja. Zapravo, informacija o nekom događaju (uz pretpostavku da je informacija valjana, potpuna i blagovremena) je tim vrednija što je događaj o kojem nas ona informira nevjerovatniji (ili, kako bi novinari rekli, senzacionalniji). Stoga, u novinarstvu vrijedi pravilo da "nije vijest kada pas ugrize čovjeka, nego je vijest kada čovjek ugrize psa". Preciznije, osnovna formula teorije informacija tvrdi da su količina informacije (označimo je sa i) o nekom događaju i vjerovatnoća v istog događaja povezani formulom

$$i = -\log_2 v$$

Ilustrirajmo ovu relaciju na jednom primjeru. Zamislimo da je osoba A sakrila kuglicu u jednu od 16 neprovidnih kutija, i da osoba B treba da pogodi u kojoj kutiji se nalazi kuglica. Osoba B ima pravo da postavlja pitanja osobi A na koje osoba A može odgovoriti sa da ili ne. U ovom slučaju vjerovatnoća da osoba B nasumice pogodi pravu kutiju iznosi $v = 1/16$. Odavde slijedi da informacija o lokaciji kuglice vrijedi $i = -\log_2 (1/16) = -\log_2 2^{-4} = 4$ bita, i da osoba B može pomoću 4 pitanja da/ne tipa tačno saznati gdje se kuglica nalazi. Pokažimo da je to zaista tačno. Pretpostavimo da su kutije raspoređene kao na sljedećoj slici (mada je sličnu strategiju moguće razviti za ma kakav raspored kutija):



Kutija u kojoj se nalazi kuglica posebno je označena na slici (naravno, osoba B to ne vidi). Kako osoba B može naći kuglicu pomoću 4 DA/NE pitanja? Strategija se zasniva na izboru takvih pitanja da nakon odgovora na svako od njih možemo odbaciti polovicu mogućnosti. Na primjer, osoba B može prvo pitati osobu A da li se kuglica nalazi u donjem redu kutija. Nakon što dobije odgovor (NE u našem primjeru), osoba B odmah može eliminirati razmatranje donjeg reda kutija, čime je prepolovljen broj mogućnosti i sveden sa 16 na 8. Sljedeće pitanje bi moglo biti da li se kuglica nalazi u lijevoj polovini gornjeg reda. Nakon dobijanja odgovora (DA u našem primjeru), osoba B može eliminirati razmatranje kutija u desnoj polovini gornjeg reda, čime je broj mogućnosti sveden na 4. Sada nije teško zaključiti koja bi trebala biti preostala pitanja, pomoću kojih bismo mogli tačno locirati traženu kuglicu.

Otkriće da se svaka informacija može modelirati isključivo odgovorima DA/NE tipa dovelo je do značajnog pojednostavljenja građe računara. Naime, još je Von Neumann predložio da se memorija računara projektuje tako da se sastoji od elementarnih memorijskih ćelija od kojih svaka može da zapamti jedno od dva različita stanja, koja redom predstavljaju DA odnosno NE, što je tehnološki prilično jednostavno za izvedbu (ta dva stanja, zavisno od građe memorije mogu biti ima/nema struje, ima/nema namagnetisanja, crno/bijelo, itd.). Stoga se najmanja količina memorije, koja može da zapamti jedno od dva različita stanja također naziva bit.

Bit je veoma mala količina memorije, pa se uvode i veće jedinice. Prva veća jedinica od bita je ***bajt***. Bajt je prvobitno definiran kao ona količina memorije koja je dovoljna da se zapamti jedno slovo, cifra, ili interpunkcijski znak (pri čemu se podrazumijeva standardni engleski alfabet). Danas se pod bajtom smatra povezana cjelina koja sadrži ***osam bita*** (mada u samim počecima informacione tehnologije nije uvijek bilo tako). Razlog za ovu konvenciju je slijedeći. Sa n bita može se zapamtiti 2^n različitih kombinacija. Za pamćenje svih malih i velikih slova, cifara, i klasičnih interpunkcijskih znakova dovoljno je oko osamdesetak kombinacija, što se može zapamtiti u 7 bita ($2^7 = 128$). Po konvenciji je usvojeno da se doda još jedan bit rezerve tako da po današnjim konvencijama jedan bajt ima 8 bita.

Veće jedinice od bajta su ***kilobajt*** (KB), ***megabajt*** (MB), ***gigabajt*** (GB) i ***terabajt*** (TB). Ponekad, mada znatno rjeđe, susreću se i *kilobit*, *megabit*, itd. Odnos između ovih jedinica je slijedeći:

1 KB = 1024 B
1 MB = 1024 KB = 1048576 B
1 GB = 1024 MB = 1048576 KB = 1073741824 B
1 TB = 1024 GB = 1048576 MB = 1073741824 KB = 1099511627776 B

Prisustvo neobičnog broja 1024 umjesto 1000 posljedica je činjenice da stvarne fizičke realizacije memorija imaju kapacitete koji kada se izraze u bajtima gotovo uvijek predstavljaju broj koji je stepen dvojke. Kako je broj 1024 stepen dvojke ($2^{10} = 1024$), a prilično je blizak broju 1000, uzeto je da odnosi između jedinica budu sa faktorom 1024 umjesto 1000, čime je postignuto da se kapaciteti memorija izraženi u KB, MB itd. izražavaju cijelim brojevima.

Radi ilustracije koliko praktično iznose jedan kilobajt, megabajt, itd. navešćemo nekoliko ilustrativnih primjera. Jedna stranica gusto kucanog teksta formata A4 orijentaciono sadrži oko 2000 znakova, tako da je za pamćenje jedne stranice teksta potrebno *oko 2 KB*. Pri tome treba voditi računa da se misli isključivo na suhi, čisti, neoblikovani (neformatirani) tekst. Čim počnemo da ukrašavamo tekst, npr. da mijenjamo oblik i veličinu slova ili da dodajemo slike, tabele, formule itd. zahtjevi za memorijom počinju intenzivno da rastu. Koliko jedna slika zauzima memorije, ovisi o vrsti i kvalitetu slike, njenoj veličini, broju boja i načinu zapisa slike u memoriji, ali općenito se može uzeti da jedna slika zauzima od *nekoliko kilobajta* do *nekoliko megabajta*. Može se primijetiti da slike troše neuporedivo više memorije nego tekst. To je i logično, s obzirom da slika sadrži neuporedivo veću količinu informacija od teksta (probajte, na primjer, da preko telefona nekome vjerno opišete sliku koju gledate, a koju sagovornik treba da nacrta). Ne kaže se džabe u narodu da "jedna slika vrijedi više od hiljadu riječi". Što se tiče zapisa zvučnih informacija, osnovne ideje o načinu zapisa zvučnih podataka date su u uvodnom poglavlju. Konkretna situacija jako varira od vrste i kvaliteta zvučnog zapisa, ali se generalno može uzeti da je jednu minutu muzičkog zapisa vrhunske kvalitete reprodukcije nemoguće zapisati u manje od jednog megabajta.

Računarska memorija ne služi samo za pamćenje podataka, nego i za pamćenje *programa*, tj. *uputa po kojima se vrši obrada memoriranih podataka*. Kasnije ćemo vidjeti da sa aspekta načina pamćenja podataka i programa u računarskoj memoriji, između podataka i programa praktički ne postoji nikakva razlika, barem ne kod računara zasnovanih na Von Neumannovom modelu (što je, zapravo, jedna od osnovnih osobina Von Neumannove arhitekture računara) – svi podaci i programi se pamte isključivo kao *niz brojeva*, i to u *istoj memoriji*. Po ovome, Von Neumannova arhitekutira računara spada u porodicu tzv. ***Princeton arhitektura***, koje karakterizira korištenje *iste memorije* za pamćenje programa i podataka (za razliku od tzv. ***Harvard arhitektura***, koji koriste *odvojene memorije* za pamćenje programa i podataka). U svakom slučaju, različiti brojevi u računarskoj memoriji predstavljaju (šifrovano) različite instrukcije koje procesor zna da izvrši. Ovi brojevi predstavljaju tzv. ***mašinske instrukcije***, o čemu ćemo kasnije detaljnije govoriti. Svaka mašinska instrukcija tipično zauzima od jednog do nekoliko bajtova (obično 2, 4, 6, 8 ili 10). Stoga, količina memorije zavisi od *broja mašinskih instrukcija* od kojih se program sastoji.

Koliko će zaista memorije zauzeti neki program, drastično zavisi od njegove namjene i od načina na koji je pisan. Mali sistemski programi koji upravljaju radom pojedinih dijelova računara, tzv. ***driveri***,

obično nisu duži od nekoliko kilobajta. S druge strane, aplikativni programi koji komuniciraju sa korisnikom na savremeni način, obično putem grafičkog korisničkog okruženja, a u posljednje vrijeme i putem glasa, rijetko zauzimaju manje od nekoliko megabajta memorije. Danas nije rijetkost susresti i programe koji zauzimaju nekoliko stotina megabajta memorije, pa čak i preko gigabajta, samo što se u takvim slučajevima gotovo nikada ne radi o zauzeću memorije od strane samog programa, nego od strane pratećih podataka uz program, poput slika, tekstova (npr. prateće dokumentacije), itd.

Razmotrimo sada sa aspekta izloženih objašnjenja kapacitete nekih uređaja eksterne memorije. Najmanji kapacitet imaju diskete. Njihov današnji kapacitet od 1.44 MB je veoma mali po sadašnjim mjerilima, ali treba napomenuti da sve zavisi od načina posmatranja. Na primjer, na disketu može stati preko 500 stranica čistog teksta (što ni u kom slučaju nije malo), ali nerijetko na disketu ne može da stane ni jedna jedina čitava kvalitetna slika, što je u svakom slučaju deprimirajuće. Također vidimo da na jednu disketu mogu stati samo relativno kratki programi. Što se tiče muzičkog zapisa, vidimo da bi na jednu disketu u najboljem slučaju moglo da stane nekoliko minuta muzičkog zapisa (i to ne vrhunske kvalitete), što je osnovni razlog zašto niko muziku ne snima na diskete (mada je to principijelno moguće). *ZIP* diskovi, mada su po građi veoma slični disketama, imaju znatno veći kapacitet (100 MB i 250 MB), ali su i mnogo skuplji. Optički *CD* diskovi se proizvode u dvije varijante, sa kapacitetima 650 MB i 700 MB, što je sasvim pristojan kapacitet, čak i po današnjim mjerilima.

DVD diskovi i hard diskovi su drugi ekstrem po pitanju kapaciteta memorije. Jedan jedini *DVD* disk (koji je prenosiv uređaj) može imati kapacitet i od nekoliko desetina gigabajta (u trenutku pisanja ovog teksta, kapacitet najvećih *DVD* diskova iznosio je 17 GB). Ovaj kapacitet je sasvim uporediv sa kapacitetom hard diskova (koji su neprenosivi uređaji) koji imaju kapacitete više desetina ili stotina gigabajta. Međutim, *DVD* diskovi nemaju ni izbliza fleksibilnost koju imaju hard diskovi u pogledu mogućnosti upisivanja i brisanja podataka, brzine pristupa podacima, itd. *DVD* diskovi se najčešće koriste za pohranjivanje digitalno zapisanih filmova, jer kako animirana slika nije ništa drugo nego slijed statičnih slika koje se brzo izmjenjuju (tipično 24 puta u sekundi), jasno je da već i jedna sekunda pokretne (animirane) slike može zauzeti jako mnogo memorije.

Interne RAM memorije danas kod računara opće namjene imaju kapacitete koji obično iznose više stotina megabajta (napomenimo da su prvi računari imali RAM memorije kapaciteta nekoliko kilobajta). Za razliku od RAM memorija, ROM memorije najčešće imaju mnogo manji kapacitet, često ne veći od 64 KB. ROM memorija zapravo sadrži samo najosnovnije informacije i programe koji omogućavaju računaru da započne rad, što to sasvim lijepo može stati u 64 KB. Te informacije i programi su smješteni u ROM a ne u RAM, jer su oni računaru neophodni odmah po uključenju računara, a RAM je tada prazan, s obzirom da se njegov sadržaj briše sa isključenjem napajanja računara. Flash kartice, koje kao što znamo predstavljaju EEPROM-e izvedene u formi eksterne memorije, obično imaju kapacitete u desetinama ili stotinama megabajta (npr. 64 MB, 128 MB, 256 MB, itd.).

(?) Pitanja i zadaci

3.1 Objasnite šta se podrazumijeva pod pojmom bit.

3.2 Objasnite kako se mjeri količina informacije.

3.3 Objasnite kakva je veza između količine informacije o nekom događaju sa vjerovatnoćom samog događaja.

3.4 Koliko bita vrijedi informacija gdje se nalazi bijeli kralj na šahovskoj tabli?

3.5 Koliko bita vrijedi informacija o cijeni automobila za koju se zna da iznosi između 10000 i 100000 KM, uz pretpostavku da je cijena cjelobrojna i da je svaka cijena u navedenom intervalu cijena jednako moguća? Da li kupac automobila može otkriti cijenu automobila u manje od 20 pitanja ukoliko mu trgovac nakon svakog pitanja govori samo da li je cijena automobila veća od cijene pretpostavljene u pitanju ili nije?

3.6 Objasnite na koji način je modeliranje informacija kao skupina odgovora na pitanja DA/NE tipa utjecalo na samu građu računara.

3.7 Koje su jedinice za mjerenje kapaciteta memorije i kakvi odnosi vladaju između njih? Objasnite zašto.

3.8 Koliko bita ima u 640 kilobajta?

3.9 Kakva je razlika između predstavljanja podataka i programa u računarskoj memoriji pod uvjetom da je računar građen po Von Neumannovom modelu?

3.10 Kakva je razlika između Princeton i Harvard arhitektura računara?

3.11 Šta su mašinske instrukcije i koliko memorije obično zauzima jedna mašinska instrukcija?

3.12 Pretpostavimo li da jedna stranica čistog teksta zauzima oko 2 kilobajta, koliko stranica teksta može stati na jednu disketu, a koliko na jedan kompakt disk?

3.13 Ukoliko neki digitalni fotoaparat uspijeva da snimi 24 fotografije na jednu disketu, koliko memorije ovaj fotoaparat koristi za zapis jedne slike?

3.14 Dva međusobno povezana računara razmjenjuju podatke brzinom od 56 kilobita u sekundi (pri čemu je 1 kilobit = 1024 bita).

a) Pri ovoj brzini prenosa, koliko minuta je potrebno da se izvrši prenos sadržaja diskete sa jednog računara na drugi, uz pretpostavku da je disketa puna?

b) Koliko megabajta podataka se može prenijeti za sat vremena pri ovoj brzini prenosa?

3.15 Znajući da muzički kompakt diskovi imaju kapacitet od 650 MB, a mogu zapamtiti 74 minute muzike, izračunajte:

a) Koliko kilobajta troši jedna sekunda zvučnog zapisa na kompakt disku?

b) Koliko bi se maksimalno sekundi takve vrste zvučnog zapisa moglo pohraniti na jednu disketu?

c) Koliko je vremena potrebno za prenos 10 minuta tako zapisanog tonskog zapisa sa jednog računara na drugi uz pretpostavku da su računari međusobno povezani modemskom vezom u kojoj je brzina prenosa podataka 56 kilobita u sekundi?

3.16 Koliko puta je kapacitet jedne Flash kartice od 128 MB veći od kapaciteta jedne diskete?

4. Organizacija podataka u računarskoj memoriji

Memorija, ma kakva da je, uvijek je podijeljena na elementarne memorijske ćelije koje mogu da zapamte samo 1 bit, o čemu ćemo kasnije detaljnije govoriti. Jedan bit može razlikovati samo jedno od dva moguća stanja koja ćemo po konvenciji, radi matematičke udobnosti, zvati 0 i 1. Šta zaista predstavljaju stanja 0 i 1, zavisi od principa na kojem je memorija napravljena. Kod elektronskih memorija možemo imati značenje 0 = nema struje, 1 = ima struje (ili napona), kod magnetnih diskova možmo imati značenje 0 = nema namagnetisanja, 1 = ima namagnetisanja, a kod optičkih diskova (pa i kod papirnih bušenih kartica kakve su se nekada koristile) može biti 0 = nema rupica na mediju, 1 = ima rupica na mediju, itd.

U Von Neumannovom modelu računara, po prvi put su, za razliku od prethodnih modela računara, kako podaci, tako i programi predstavljeni *na jedinstven način*, i na *istim medijima*. Po ovom modelu, svi programi i podaci pohranjeni su u memoriji isključivo kao *kombinacije nula i jedinica*. Stoga, važno je da zaključimo da apsolutno sve što računar čuva u memoriji, bio to broj, slovo, tekst, slika, muzički zapis ili program, može biti zapisano isključivo kao kombinacija "nula" i "jedinica", i nikako drugačije.

Izlaganje o organizaciji podataka u računarskoj memoriji započećemo izlaganjem organizacije *cjelobrojnih podataka*. Već smo rekli da se svaki podatak u računarskoj memoriji čuva isključivo kao skupina nula i jedinica. Stoga, da bismo proizvoljan broj zapisali u računarskoj memoriji, očigledno nam je potreban način da brojeve predstavljamo uz pomoć samo dvije cifre, 0 i 1. Za tu svrhu moramo uvesti pojam ***brojnog sistema***. Postavimo zbog toga pitanje: šta predstavlja broj 342, i po čemu se on razlikuje od brojeva 423 i 234 koji su sastavljeni od istih cifara? Znamo da svaka cifra ima svoju težinu (koja je stepen broja 10), i da broj 342 predstavlja skraćeno pisanje za izraz

$$3 \cdot 10^2 + 4 \cdot 10^1 + 2 \cdot 10^0$$

U suštini, svaki cijeli broj, zapisan kao $a_N...a_2a_1a_0$ predstavlja skraćeni zapis za izraz oblika

$$\sum_{i=0}^{N} a_i 10^i$$

dakle, za polinom čija je baza (osnova) 10, a čiji su koeficijenti cifre a_i (u opsegu od 0 do 9). Ako razmislimo zašto je baza baš 10, shvatićemo da za ovakav izbor nema suštinski nikakvog valjanog razloga. Jedini razlog je zapravo istorijski, zbog činjenice da čovjek ima 10 prstiju, i da je shvatio da za brojeve veće od 10 ne može više računati na prste, nego da mora izmisliti neki način zapisa većih brojeva. Baza je, u suštini, mogla da bude proizvoljan cijeli broj $B > 1$, jer se lako može dokazati da se svaki pozitivan broj može na jednoznačan način zapisati u obliku

$$\sum_{i=0}^{N} a_i B^i, \quad 0 \le a_i < B$$

pri čemu su sve cifre a_i, $i = 1..N$ u opsegu od 0 do B–1.

Principijelno, brojeve možemo zapisati u bilo kojoj bazi $B > 1$. Kad god baza broja nije 10, moramo to posebno naglasiti stavljanjem broja u zagradu i pisanjem baze kao indeksa iza zagrade, osim ako je baza jasna iz konteksta. Npr. broj $(3402)_5$ je broj zapisan u bazi 5. Sasvim je lako pretvoriti broj iz proizvoljne baze u bazu 10. Na primjer,

$$(3402)_5 = 3 \cdot 5^3 + 4 \cdot 5^2 + 0 \cdot 5^1 + 2 \cdot 5^0 = 477.$$

Slijedi da je $(3402)_5 = (477)_{10}$.

Ako ovo sve nekome djeluje "nastrano", treba se sjetiti da se i u praksi (npr. u mjerenju vremena i uglova) koriste baze različite od 10. Na primjer, posmatrajmo broj $23^\circ 12' 49''$. Pretvoreno u ugaone sekunde, ovaj broj iznosi:

$$23 \cdot 60^2 + 12 \cdot 60^1 + 49 \cdot 60^0 = 83569$$

Ovdje se zapravo radi o jednom "3-cifrenom" broju, zapisanom u bazi 60, sa "ciframa" "23", "12" i "49". Ovdje, naravno, "cifre" mogu biti u rasponu od 0 do 59. Standardni brojni sistem koji koristi bazu 10 nazivamo ***dekadni*** (***decimalni***) sistem.

Može se postaviti pitanje, kako se rješava obrnuti problem, tj. kako bismo npr. broj 477 pretvorili iz baze 10 (dekadni zapis) u zapis u bazi 5? Za tu svrhu posmatrajmo slijedeću tablicu stepena broja 5:

$5^0 = 1$
$5^1 = 5$
$5^2 = 25$
$5^3 = 125$
$5^4 = 625 > 477$

Na osnovu ovoga možemo napraviti slijedeće razlaganje:

$$477 = 3 \cdot 125 + \underline{102} = 3 \cdot 125 + 4 \cdot 25 + \underline{2} = 3 \cdot 125 + 4 \cdot 25 + 2 \cdot 1$$

odakle slijedi:

$$477 = 3 \cdot 5^3 + 4 \cdot 5^2 + 0 \cdot 5^1 + 2 \cdot 5^0 = (3402)_5$$

Gore opisani metod za pretvaranje brojeva iz dekadnog sistema u sistem proizvoljne baze nije pogodan za pretvaranje većih brojeva. Standardni metod za pretvaranje, zasniva se na uzastopnom cjelobrojnom dijeljenju broja sa željenom bazom, pri čemu se ispod broja piše količnik, a sa desne strane ostatak pri dijeljenju. Postupak se ponavlja sve dok se ne dođe do količnika 0 (ovaj postupak je zapravo *Hornerova shema*, poznata iz teorije polinoma). Pokažimo, na primjer, kako bi izgledalo pretvaranje broja 477 u bazu 5 prema ovom postupku:

```
   :5
477 | 2
 95 | 0
 19 | 4 ↑
  3 | 3 |
  0 |
```

Ostaci dijeljenja čitani od posljednjeg ka prvom daju upravo tražene cifre broja u željenoj bazi.

Iako je dekadni brojni sistem najviše rasprostranjen u svakodnevnom životu, njegova primjena u digitalnim računarima bila bi uzrok mnogih poteškoća u tehničkoj realizaciji uređaja. Razlog leži u činjenici da bi elektronski sklopovi za predstavljanje deset različitih cifara morali imati deset jasno definiranih različitih stanja. To bi dovelo do drastičnog usložnjavanja samog računara. Stoga je sa stanovišta lakše i pouzdanije realizacije digitalnih računara daleko povoljnije da se koristi brojni sistem koji će se moći predstaviti sa samo dva stanja kola, odnosno sa samo dvije cifre. Stoga je za primjenu u računarstvu od velikog značaja baza 2 koja daje ***binarni brojni sistem***. U binarnom brojnom sistemu javljaju se samo dvije cifre, 0 i 1, a to je upravo ono što nam je potrebno. Kao ilustraciju, pretvorimo broj 142 iz dekadnog u binarni brojni sistem:

```
142 | 0
 71 | 1
 35 | 1
 17 | 1
  8 | 0
  4 | 0
  2 | 0
  1 | 1
  0 |
```

Dakle, 142 se u binarnom brojnom sistemu piše kao 10001110. Pogodnost da se koriste samo dvije cifre "plaćena" je time da u binarnom sistemu čak i relativno mali brojevi imaju veliki broj cifara (otprilike 3.3 puta više nego u dekadnom brojnom sistemu). Računaru to, međutim, ne smeta mnogo. Bitno je napomenuti da binarne brojeve uvijek izgovaramo cifru po cifru. tj. binarni broj 10 čitamo isključivo kao "jedan nula" a nipošto kao "deset" jer to nije broj deset (to je zapravo broj 2).

Ilustrirajmo još jednom i obrnutu pretvorbu na primjeru pretvaranja broja 101010011 iz binarnog u dekadni brojni sistem:

$$(101010011)_2 = 2^8 + 2^6 + 2^4 + 2^1 + 2^0 = 256 + 64 + 16 + 2 + 1 = 339$$

Glomaznost izražavanja u binarnom sistemu u računarstvu se rješava uvođenjem ***heksadekadnog*** (ili ***heksadecimalnog***) ***brojnog sistema***, koji koristi bazu 16, i stoga se u njemu javlja 16 cifara. Pošto su ljudi izmislili znakove samo za deset cifara, nedostajućih šest cifara obično se zapisuju kao slova A, B, C, D, E i F (u matematici se također susreće i zapisivanje pomoću grčkih slova α, β, γ, δ, ε i φ). Razlog za upotrebu heksadekadnog brojnog sistema je u tome što se brojevi u ovom brojnom sistemu zapisuju veoma kompaktno, a pretvaranje iz binarnog u heksadekadni brojni sistem i obrnuto obavlja se veoma jednostavno. Da bismo pretvorili heksadekadni broj u binarni, dovoljno je svaku cifru heksadecimalnog broja pretvoriti zasebno u 4-cifreni binarni broj (po potrebi se dodaju nule sa lijeve strane ukoliko pretvoreni binarni broj ima manje od četiri cifre). Na primjer, $(C5)_{16} = (11000101)_2$, jer je $(C)_{16} = (12)_{10} = (1100)_2$ i $(5)_{16} = (5)_{10} = (101)_2$. Za obrnuto pretvaranje potrebno je binarni broj podijeliti u grupe od po četiri cifre zdesna nalijevo, i svaku grupu zasebno pretvoriti u heksadecimalnu cifru. Ovi postupci se uz malo prakse mogu uvježbati da se brzo vrše čak i napamet, bez ikakvih pomagala (što nije moguće za pretvaranje između binarnog i dekadnog sistema), što nam omogućava da se sažeto izražavamo u heksadekadnom sistemu, a da zapravo mislimo na binarne brojeve. Upravo je ovo osnovni razlog zbog čega je heksadekadni brojni sistem uopće uveden.

Sada možemo reći kako računar pamti cijele brojeve. Svaki cijeli broj pretvara se u binarni oblik, a nakon toga se svaka binarna cifra broja (koja se također naziva ***bit*** od engl. Binary digit, kao i istoimena jedinica za mjerenje kapaciteta memorije) čuva u elementarnoj memorijskoj ćeliji računara, koja može zapamtiti samo jedan bit.

Najmanji broj koji može stati u n bita je, jasno:

$$\underbrace{(0\ldots000)_2}_{n \text{ nula}} = 0$$

Interesantno je izračunati najveći broj koji može stati u n bita. To je, naravno, broj koji se u binarnom sistemu piše sa n jedinica:

$$\underbrace{(1\ldots111)_2}_{n \text{ jedinica}} = 1 \cdot 2^{n-1} + \ldots + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = \sum_{i=0}^{n-1} 2^i = 2^n - 1$$

Odavde slijede sljedeći zaključci:

- Najveći broj koji može stati u 1 bajt (8 bita) je: $2^8 - 1 = 255$
- Najveći broj koji može stati u 2 bajta (16 bita) je: $2^{16} - 1 = 65535$
- Najveći broj koji može stati u 4 bajta (32 bita) je: $2^{32} - 1 = 4294967295$

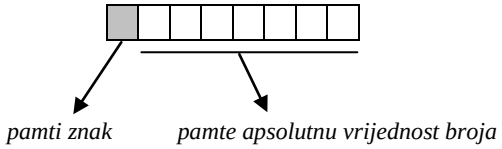
Primijetimo da su 4 bajta uglavnom dovoljna da pokriju opseg cijelih brojeva koji se najčešće javlja u praksi. Stoga se danas najčešće rezerviraju upravo 4 bajta za pamćenje cijelih brojeva.

Dosadašnje razmatranje odnosilo se samo na *nenegativne cijele brojeve*. Razmotrimo sada kako bi se mogli zapisati *cijeli brojevi sa predznakom*. Jasno je da se i pamćenje predznaka mora svesti na pamćenje "nule" ili "jedinice". Za memoriranje znaka broja postoje različiti metodi, ali se svi uglavnom svode na to da se jedan od bita "žrtvuje" za memoriranje znaka broja (npr. 0 = pozitivan, 1 = negativan), čime se efektivno smanjuje opseg brojeva po modulu koji se mogu prikazati sa *n* bita, tako da imamo:

- Sa 1 bajtom opseg je: –128 do +127
- Sa 2 bajta opseg je: –32768 do +32767
- Sa 4 bajta opseg je: –2147483648 do +2147483647

"Asimetrija" između opsega u negativnom i pozitivnom smjeru posljedica je činjenice da ne postoji "negativna nula", što omogućava da opseg u negativnom smjeru bude veći za 1.

Jedan od mogućih metoda za pamćenje negativnih brojeva je metod "***znak i vrijednost***". Kod ovog metoda unaprijed se rezervira određena količina bita za broj (npr. 8 bita). Najviši bit se proglasi za znak, a ostali biti čuvaju apsolutnu vrijednost broja:



Npr. $13 = (1101)_2$, pa imamo:

+13 se čuva kao 00001101
–13 se čuva kao 10001101

Mana ovog metoda je što nula ima 2 različita zapisa (00000000 i 10000000), tj. postoji negativna nula, kao i činjenica da se računske operacije sabiranja i oduzimanja sa ovako opisanim brojevima vrše različitim postupkom ovisno da li su brojevi istog znaka ili nisu. Ovaj nedostatak je, doduše prisutan i u "normalnom" brojnom sistemu sa bazom 10, jer mi ne sabiramo na isti način brojeve +15 i +11, i brojeve +15 i –11. U drugom primjeru mi sabiranje zapravo svodimo na oduzimanje. Vidjećemo kasnije da se ovaj nedostatak može i izbjeći.

Razmotrimo sada kako se *realni brojevi* zapisuju u računarskoj memoriji. Postoje neki matematički zakoni (koji slijede iz tzv. neprebrojivosti skupa realnih brojeva) zbog kojih se svi realni brojevi *ne mogu* i *nikada neće moći* u potpunosti tačno predstaviti u računarskoj memoriji. Ono što je ipak moguće postići je da proizvoljan realan broj možemo zapisati sa proizvoljnom tacnošću (ali ne potpuno tačno) tj. uvijek možemo postići da greška bude manja od ma kako malog unaprijed zadatog broja $\varepsilon>0$. Primijetimo prvo da se i brojevi koji nisu cijeli mogu pisati u binarnom brojnom sistemu koristeći decimalni zarez (koji bi se tada tačnije trebao zvati *binarni zarez*, a cifre iza zareza *binale* umjesto *decimale*). Na primjer, po analogiji sa dekadnim brojnim sistemom, možemo zaključiti da je:

$$(101{,}011)_2 = 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 + 0\cdot 2^{-1} + 1\cdot 2^{-2} + 1\cdot 2^{-3} = 4 + 1 + 0{,}25 + 0{,}125 = 5{,}375$$

Postoji nekoliko načina kako se realni brojevi mogu čuvati u računarskoj memoriji. Pri tome, svaki od načina traži izvjesne aproksimacije.

a) Način "***razlomak***" (engl. *fraction*)

Kod ovog načina realni broj se aproksimira sa unaprijed zadanom tačnošću sa razlomkom oblika p/q gdje su p i q cijeli brojevi bez zajedničkih faktora, a zatim se p i q čuvaju u memoriji na uobičajeni način. Naime, za ma koji realni broj x i za ma kakvu zadanu tačnost $\varepsilon > 0$ uvijek je moguće naći takve cijele brojeve p i q da vrijedi $|x - p/q| < \varepsilon$. Prednost ovog načina je što se potpuno tačno mogu pamtiti i mnogi brojevi sa beskonačno mnogo decimala (npr. 1/7 i većina racionalnih brojeva, osim ako im brojnik i nazivnik nisu "preveliki"). Ipak, nepraktičnost algoritama za računanje sa ovakvim zapisom brojeva za (pogotovo za operacije koje nisu četiri standardne operacije: sabiranje, oduzimanje, množenje i dijeljenje) čine ovaj način zapisa neuobičajenim u većini programskih jezika. Neki matematički orijentirani programski jezici (*Mathematica*, *Maple*, *MuLisp* itd.) ipak poznaju ovakav zapis realnih brojeva, koji omogućava savršenu tačnost pamćenja svih racionalnih brojeva.

b) Način "***fiksni zarez***" (engl. *fixed point*)

Kod ovog načina zapisa, unaprijed se *fiksira preciznost*, odnosno *maksimalni broj decimala koje će se pamtiti*. Brojevi koji imaju više decimala zaokruže se, a brojevi koji imaju manji broj decimala dopune se nulama do zadanog broja decimala. Nakon toga se decimalni zarez odbaci i dobijeni cijeli broj se pamti u memoriji na uobičajeni način. Na primjer, ako je fiksirani broj decimala 5, tada:

Broj 32,76544	pamti se kao	3276544	
Broj 2,17	pamti se kao	217000	(3 nule su dodane)
Broj 14,3447792	pamti se kao	1434478	(broj je prethodno zaokružen)

Ovakav način zapisa naročito je pogodan u *bankarstvu*, jer sve cijene i novčani iznosi imaju fiksiran broj decimalnih mjesta (obično 2). To je razlog zašto izvjesni programski jezici namijenjeni za razvoj finansijskih aplikacija (npr. DBASE, Clipper itd.) koriste ovakav način zapisa realnih brojeva (pri čemu se fiksni broj decimala može podešavati).

Osnovna mana ovakvog zapisa je što se isti broj decimala definira kako za velike tako i za male brojeve. To ima dvije štetne posljedice. Prvo, sasvim mali brojevi (kao što je npr. naboj elektrona koji iznosi $1{,}6 \cdot 10^{-19}$) uopće se ne mogu memorirati (osim ako ne rezerviramo jako veliki broj decimalnih mjesta, pri čemu onda bespotrebno pamtimo gomilu besmislenih decimala za brojeve "normalne" veličine). Drugo, tačnost svih mjerenja izvodivih u praksi obično je srazmjerna samoj veličini koju mjerimo. Na primjer, nema ništa čudno ako kažemo da je dužina neke sobe 3,57 m, jer nije nikakav problem izmjeriti dužinu sobe sa tačnošću od jednog centimetra. Međutim, ako kažemo da je razdaljina između Sarajeva i Mostara 132776,34 m (a jeste *otprilike* tolika), rekli smo besmislicu. Prvo, nismo definirali "odakle" u Sarajevu i "dokle" u Mostaru, jer se radi o gradovima čiji je prečnik nekoliko kilometara. Čak i ako tačno preciziramo početnu i krajnju tačku (npr. "od vječne vatre u Sarajevu do Starog mosta u Mostaru"), tu razdaljinu nije moguće izmjeriti sa tačnošću od jednog centimetra niti metra. Slijedi, da ako nam je jedinica mjere metar, pamćenje dvije decimale djeluje smisleno kada mjerimo dužinu sobe, ali besmisleno kada mjerimo udaljenost između dva grada (da i ne govorimo o mjerenju udaljenosti između dva nebeska tijela). Inženjeri izlaz iz ove situacije nalaze u korištenju eksponencijalne notacije. Tako ako udaljenost između Sarajeva i Mostara (oko 132 km) treba izraziti u metrima, inženjer će radije pisati $1{,}32 \cdot 10^5$ m nego 132000 m, jer je tačnost u drugom zapisu prilično upitna. Isto tako, astronom će rađe pisati da je udaljenost Zemlje i Sunca $1{,}5 \cdot 10^8$ km nego 150000000 km, jer je ovaj drugi broj "opterećen" nulama "sumnjive" tačnosti. Fizičar će reći da je poluprečnik prve orbite u atomu vodika $5{,}29 \cdot 10^{-11}$ m a ne 0,0000000000529 m (ovdje nule na početku nisu sumnjive tačnosti, nego je ovakav zapis prosto nezgrapan). Ova razmatranja dovode nas do slijedeće ideje za zapis realnih brojeva.

c) Način "***pokretni zarez***" (engl. *floating point*)

Svaki realni broj, različit od nule, može se na jedinstven način napisati u obliku $\pm m \cdot 10^e$, gdje je e cijeli broj (nazvan eksponent), a m realan broj (nazvan mantisa) u opsegu $0{,}1 \le m < 1$. Na primjer:

$34270 = 0{,}3427 \cdot 10^5$
$47{,}19 = 0{,}4719 \cdot 10^2$
$5 = 0{,}5 \cdot 10^1$
$0{,}72 = 0{,}72 \cdot 10^0$
$0{,}0012 = 0{,}12 \cdot 10^{-2}$

Primijetimo da eksponent *e* zapravo "određuje" gdje se zaista nalazio decimalni zarez prije transformacije, odnosno on vrši njegovo "pomijeranje" odakle i potiče naziv "pokretni zarez".

Ideja zapisa realnih brojeva u pokretnom zarezu sastoji se u tome da broj pamtimo kao par (e, m). Kako je eksponent cijeli broj, on se zapisuje na uobičajeni način. Mantisu zaokružimo na fiksan, unaprijed zadan broj decimala (koji zapravo određuje željeni broj tačnih cifara broja), ili dopunimo nulama ako je broj decimala manji, a zatim odbacimo decimalni zarez i tako dobijeni cijeli broj pamtimo u memoriji (drugim riječima, mantisu pamtimo metodom fiksnog zareza).

Možemo primijetiti da i metod fiksnog i metod pokretnog zareza mogu dovesti do gubitka tačnosti usljed eventualnog zaokruživanja i odsjecanja decimala. Međutim, dok se kod metoda fiksnog zareza garantira da greška neće preći neki unaprijed dozvoljeni iznos, kod metoda pokretnog zareza se garantira da greška neće preći određeni iznos *relativno* u odnosu na veličinu samog broja koji se pamti (npr. može se garantirati da greška u zapisivanju broja *x* neće preći recimo 0.0001% od iznosa samog broja *x*). Drugim riječima, metod fiksnog zareza garantira maksimalni mogući iznos *apsolutne greške*, dok metod pokretnog zareza garantira maksimalni mogući iznos *relativne greške*.

Metod pokretnog zareza je, uz izvjesne modifikacije, prihvatila većina današnjih programskih jezika. Modifikacije se zapravo svode na to da se posmatra "pokretni zarez u bazi 2". Može se lako dokazati da se svi realni brojevi osim nule mogu također jednoznačno prikazati i u obliku $\pm m \cdot \mathrm{B}^e$, gdje je B proizvoljan cijeli broj (baza) veći od 1, cijeli broj *e* je eksponent u bazi B, a realni broj *m* je mantisa u bazi B, koja se može uvijek svesti na opseg $1/\mathrm{B} \le m < 1$. Računari koriste B = 2, na primjer:

$23 = 0{,}71875 \cdot 2^5$
$0{,}019 = 0{,}608 \cdot 2^{-5}$

Eksponent *e* se pamti kao i svaki drugi cijeli broj, dok se mantisa *m* prvo pretvara u binarni brojni sistem, zatim se tako dobijeni binarni broj pamti metodom fiksnog zareza.

Računari obično rezerviraju fiksan broj bita za pamćenje eksponenta i mantise. Tako je po jednom od važećih standarda (IEEE 754 Double) dogovoreno da se za pamćenje eksponenta rezervira 11 bita a za pamćenje mantise 52 bita. Zajedno sa jednim bitom za znak to je 64 bita ili tačno 8 bajta. Ograničenje broja bita za *m* i *e* povlači da je broj decimala mantise koji se može zapamtiti ograničen i da je opseg brojeva koji se mogu zapamtiti ograničen. Međutim, ova ograničenja ispunjavaju sve zahtjeve prakse jer se po IEEE 754 Double standardu sasvim lijepo može zapamtiti 15–16 decimala mantise, a i dozvoljeni opseg brojeva je daleko veći nego što će nam ikad zatrebati (po modulu od $5{,}0 \cdot 10^{-324}$ do $1{,}7 \cdot 10^{308}$).

Svi opisani metodi u suštini su pamtili samo *racionalne brojeve*. Problem efikasnog i što tačnijeg pamćenja što većeg podskupa skupa realnih brojeva ostaje i danas otvoren. Primijetimo da je danas najraširenij metod "floating point" u stanju da tačno zapamti samo brojeve sa konačno mnogo decimala, dok je "fraction" metod mogao da tačno zapamti i mnoge brojeve sa beskonačno mnogo decimala (npr. 1/3). Metod "floating point" ne može da potpuno tačno zapamti čak ni tako jednostavan broj kao što je 0,2. Ovaj broj zaista ima konačno mnogo decimala u dekadnom brojnom sistemu, ali beskonačno mnogo decimala (koje bi, preciznije rečeno, trebalo zvati *binale* a ne *decimale*) u binarnom brojnom sistemu, koji se koristi za memoriranje. Naime, nije teško dokazati da je:

$0{,}2 = (0{,}001100110011...)_2$

Što se tiče iracionalnih brojeva, neki programski paketi kao sto su *Mathematica*, *Maple* itd. uspijevaju da u potpunosti zapamte i neke iracionalne brojeve. Na prvom mjestu to su algebarski iracionalni brojevi koji mogu biti nula nekog polinoma sa cjelobrojnim koeficijentima. Na primjer, $\sqrt{2}$ je jedna nula polinoma x^2-2, $\sqrt{2}+\sqrt{3}$ je jedna nula (koje su ostale?) polinoma $(x^2-5)^2-24$ odnosno x^4-10x^2+1, dok je $\sin(\pi/7)$ jedna nula polinoma $64x^6-112x^4+56x^2-7$ (provjeriti ovo; odavde vidimo da je i $\sin(\pi/7)$ algebarski broj iako se ne može izraziti pomoću korijena, bar ne u realnom domenu). Da bi se zapamtio algebarski iracionalan broj u memoriji, principijelno je dovoljno samo zapamtiti koeficijente polinoma čija je on nula, što nije problem, jer su oni cijeli brojevi.

Neki matematički orijentirani programski jezici (npr. poput već pomenutih *Mathematica*-e i *Maple*-a) idu čak i dalje, i uspjevaju tačno da zapamte i neke transcedentne iracionalne brojeve, kao što su *e* i π, pamteći u memoriji *postupak kojim se može izračunati proizvoljna njihova decimala*. Principijelno je moguće tačno zapamtiti u memoriji bilo koji broj za koji postoji neka logika u principu kako se generiraju njihove decimale (tzv. *izračunljivi brojevi*). Brojevi za koje ovo nije slučaj, ne mogu se u potpunosti tačno zapamtiti u računarskoj memoriji, *niti će ikada moći*.

Već smo rekli da se u memoriji računara pamte isključivo *brojevi* (i to binarni). Stoga je neophodno naći način da se i *znakovni podaci* čuvaju u memoriji kao nizovi bita. Uobičajena je konvencija da se za svaki znak nekog teksta rezervira tačno jedan bajt (zapravo, bajt je i definiran upravo sa takvom svrhom), što je dovoljno da se zapamti $2^8 = 256$ različitih znakova po jednom bajtu. Da bi se ustanovilo kojom ce se kombinacijom nula i jedinica predstavljati koji znak, uspostavljene su odgovarajuće međunarnodno dogovorene šifre koje propisuju kombinacije nula i jedinica kojim se predstavljaju određeni znakovi, i tih dogovora bi se svi trebali pridržavati. Mada postoji više ovakvih standardnih šifri, danas ubjedljivo najviše korištena šifra (kôd) je tzv. ***ASCII*** kôd (ASCII – American Standard Codes for Information Interchange). Navedimo nekoliko primjera šifri prema ASCII standardu:

Znak:	Predstavlja se kao:
A	01000001
a	01100001
2	00110010
!	00100001
razmak	00100000

Primijetimo da se i razmak (prazan znak) tretira u memoriji kao i svaki drugi znak, i da je i za njega predviđena odgovarajuća šifra. Kompletan spisak šifri po ASCII standardu prikazan je u slijedećoj tablici (šifre su navedene u zagradi iza znaka, u dekadnom brojnom sistemu, radi uštede prostora):

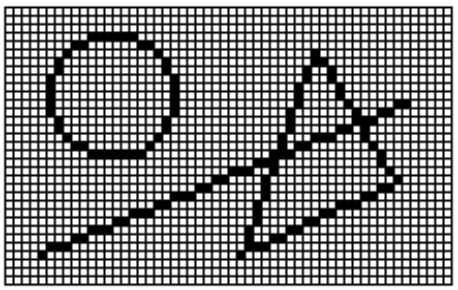
razmak (32)	**!** (33)	**"** (34)	# (35)	**$** (36)	**%** (37)
& (38)	**’** (39)	**(** (40)	**)** (41)	***** (42)	**+** (43)
, (44)	**-** (45)	**.** (46)	**/** (47)	**0** (48)	**1** (49)
2 (50)	**3** (51)	**4** (52)	**5** (53)	**6** (54)	**7** (55)
8 (56)	**9** (57)	**:** (58)	**;** (59)	< (60)	**=** (61)
> (62)	**?** (63)	@ (64)	**A** (65)	**B** (66)	**C** (67)
D (68)	**E** (69)	**F** (70)	**G** (71)	**H** (72)	**I** (73)
J (74)	**K** (75)	**L** (76)	**M** (77)	**N** (78)	**O** (79)
P (80)	**Q** (81)	**R** (82)	**S** (83)	**T** (84)	**U** (85)
V (86)	**W** (87)	**X** (88)	**Y** (89)	**Z** (90)	**[** (91)
**** (92)	**]** (93)	**^** (94)	**_** (95)	**`** (96)	**a** (97)
b (98)	**c** (99)	**d** (100)	**e** (101)	**f** (102)	**g** (103)
h (104)	**i** (105)	**j** (106)	**k** (107)	**l** (108)	**m** (109)
n (110)	**o** (111)	**p** (112)	**q** (113)	**r** (114)	**s** (115)
t (116)	**u** (117)	**v** (118)	**w** (119)	**x** (120)	**y** (121)
z (122)	**{** (123)	**|** (124)	**}** (125)	**~** (126)	

Osnovni problem ASCII standarda je u tome što on nije predvidio šifre za slova *izvan engleskog alfabeta*, poput naših slova Č, Š itd. (što ne treba mnogo da čudi, s obzirom da je ASCII američki standard namijenjen engleskom govornom području). Nažalost, za rješavanje ovog problema, pojavio se veliki broj međusobno nekonzistentnih standarda koji su predviđali šifre i za naša slova. Tako je u bivšoj Jugoslaviji nastao standard (bolje reći "zakrpa" ASCII standarda) pod nazivom YUSCII, a pojavili su se i razni međunarodno dogovoreni standardi (poput standarda LATIN 2) koji su podržavali naša slova. Osnovni problem je što ovi standardi nisu saglasni međusobno, tj. za isto slovo (npr. "Š") jedan "standard" predviđa jednu šifru, a drugi drugu. Stoga, mogu nastati veliki problemi ukoliko se dokument koji je rađen pod jednim standardom (npr. YUSCII) prenese na sistem koji radi pod drugim standardom (npr. LATIN 2). Stoga nisu rijetke pojave da se u nekom dokumentu koji sadrži naša slova ona izgube (ili pretvore u besmislene znakove) prilikom prenosa na neki drugi računar. Uzrok problema je u tome što izvorni i odredišni sistem ne koriste iste šifre za memoriranje naših slova.

U posljednje vrijeme radi se na univerzalnom rješavanju ovog problema uz pomoć standarda nazvanog ***UNICODE*** koji predviđa ne samo naša slova, nego i sva slova grčkog, arapskog, hebrejskog, indijskog, kineskog i svih drugih alfabeta koji se koriste na svijetu. Ukratko, UNICODE standard predviđa preko 30000 različitih znakova. Ovaj standard po svakom znaku rezervira 16 a ne 8 bita (odnosno dva bajta), što je dovoljno da se zapamti čak $2^{16} = 65536$ različitih znakova. Tablica UNICODE standarda je, razumije se, ogromna. Međutim, vrijedi znati da je ASCII podskup UNICODE-a u smislu da znak koji ima svoju ASCII šifru ima istu šifru i u UNICODE-u, samo sa dodatnih 8 nula ispred (da se šifra dopuni do 16 bita). Npr. slovo "A" čija je ASCII šifra 01000001, ima UNICODE šifru 0000000001000001. Napomenimo da je u *Windows* seriji operativnih sistema do prvog ozbiljnijeg prelaza sa ASCII na UNICODE standard došlo je pojavom programa za obradu teksta *Microsoft Word 97* za *Windows 95* operativni sistem. Ipak, vrijedi napomenuti da ni *Windows 95* (pa čak i noviji *Windows* sistemi) ne koriste UNICODE standard posve konzistentno, te izvjesni problemi sa nacionalnim (neengleskim) slovima i dalje ostaju. Na primjer, nije mudro u imenima datoteka i foldera koristiti slova izvan engleskog alfabeta. Primijetimo da zbog prelaska na UNICODE dokumenti pisani u programu *Microsoft Word 97* (ili nekom još novijem) zauzimaju dvostruko više memorije nego dokumenti pisani pomoću ranijih verzija programa *Microsoft Word*.

Na kraju ovog poglavlja, recimo nešto i o zapisu ostalih nenumeričkih podataka poput *slika* i *zvuka* u računarskoj memoriji. Jasno je da i ovakve podatke prvo moramo na neki način predstaviti pomoću *brojeva*. Postoji mnogo načina da se slike opišu kao skupina brojeva, koje generalno možemo podijeliti na ***rasterske*** (***bit-mapirane***) ***načine*** i ***vektorske načine***, koje ćemo u nastavku detaljnije razmotriti.

Kod *rasterskog načina* zapisa, slika se predstavlja kao veoma gusta pravougaona mreža (raster) sastavljena od kvadratića nazvanih ***pikseli*** (od engl. Picture Element). Pri tome se smatra da svaki od kvadratića sadrži unutar sebe samo jednu boju (što nije nerealno, jer su kvadratići veoma sitni). Računar tada prosto memorira informaciju o boji svakog piksela pojedinačno, koja se može opisati brojem. Na primjer, svaku od boja koja se pojavljuje na slici moguće je predstaviti jednim brojem. Broj različitih boja koje slika može imati zavisi od broja bita koje smo predvidjeli za pamćenje informacija o svakom od piksela. Obično se koristi 1, 4, 8, 16 ili 24 bita po pikselu, što redom omogućava 2, 16, 256, 65536 odnosno 16777216 boja. Slike koje koriste 1 bit po pikselu mogu predstaviti samo dvije boje (npr. crnu i bijelu) i nazivaju se ***monohromatske slike***. Na sljedećoj slici karikirano je prikazano kako izgleda raster za jednu monohromatsku sliku, i kako je zapravo predstavljena jedna geometrijska slika. Naravno, u stvarnosti je raster mnogo sitniji.



Slike koje koriste 24 bita po pikselu mogu predstaviti praktično sve boje i nijanse vidljive ljudskom oku (tzv. ***True Color*** zapis). Ostaje još jedino pitanje koji broj (tj. koja kombinacija bita) odgovara kojoj boji. U slučaju slika sa manjim brojem boja (do 256), obično se unaprijed dogovori koji će broj predstavljati koju boju, a tada se dogovorena informacija čuva zajedno sa slikom u formi tablice koja se obično naziva ***paleta***. S druge strane, u slučaju slika sa velikim brojem boja, obično pojedinačne grupe bita u informaciji o boji sadrže informaciju o relativnom udjelu svake od nekoliko osnovnih boja (obično 3) u posmatranoj boji, tako da se sama boja formira *miješanjem* osnovnih boja u različitim omjerima.

Kod rasterskog zapisa nije potrebno pamtiti *koordinate* svakog od piksela. Ovo je izbjegnuto na taj način što se podaci o pikselima pohranjuju u *unaprijed utvrđenom poretku*, obično red po red, slijeva nadesno i odozgo nadolje. Pri tome je neophodno poznavati *broj piksela u svakom redu*, kao i *broj redova piksela*. Ovi podaci, napisani u formi *produkta* (npr. 800×600) tvore ***apsolutnu rezoluciju*** ili ***dimenziju*** (***format***) slike.

Jedan očigledan nedostatak rasterskog zapisa slike je što se uvijek čuva informacija o boji svakog piksela, bez obzira na količinu detalja na slici. Tako, na primjer, rasterska slika formata 800×600 sa True Color zapisom boja zauzima $800 \cdot 600 \cdot 24 = 11520000$ bita, odnosno oko 1.4 megabajta, bez obzira da li se radi o visokokvalitetnoj fotografiji, ili o potpuno praznom bijelom listu papira! Da bi se izbjegao ovaj nedostatak, razvijeni su razni postupci za ***kompresovano zapisivanje*** rasterskih slika. Postoje razni metodi za kompresovano zapisivanje (***RLE***, ***GIF***, ***JPEG***, itd.) o kojima ovdje nećemo govoriti. Jedna jednostavna ideja za kompresiju sastoji se u tome da se umjesto boje *svakog* piksela čuva informacija o *grupama piksela* koji su obojeni istom bojom. Pored toga, postupci za kompresovano zapisivanje rasterskih slika, mogu se podijeliti na ***postupke bez gubitaka***, kod kojih prilikom kompresije ne dolazi ni do kakve degradacije kvaliteta slike, kao i ***postupke sa gubicima***, kod kojih kompresija dovodi do izvjesne degradacije kvaliteta slike. Tipičan primjer postupka sa gubicima je *JPEG* metod kompresije koji, između ostalog, kompresiju postiže tako što *pretpostavlja* boju izvjesnih piksela na osnovu boje njemu susjednih piksela, čime se postiže veoma veliki faktor kompresije (ponekad i više stotina puta). Ovaj metod nije pogodan za pamćenje slika kod kojih postoje nagli prelazi sa svijetlih na tamne nijanse, ali je zato idealan za pamćenje fotografija, kod kojih se takvi prelazi ne javljaju. Kod većine postupaka kompresije sa gubicima moguće je birati odnos između željenog faktora kompresije (memorijske uštede) i stepena degradacije slike.

Kod *vektorskog načina* zapisivanja slika, slika se u memoriji pamti kao *niz objekata*, pri čemu se za svaki objekat pamte njegove osnovne karakteristike, koje se lako mogu izraziti brojevima. Na primjer, prava linija (duž) je u načelu u potpunosti određena *koordinatama početne i krajnje tačke*, krug je određen *koordinatom centra i dužinom poluprečnika* (*radijusa*), dok je poligon određen *koordinatama svojih tjemena* (*vrhova*). Sva navedena svojstva su očito opisiva brojevima. Pored ovih *osnovnih svojstava* moguće je pamtiti i dopunska svojstva objekata. Tako je, na primjer, za liniju moguće pamtiti njenu *debljinu* i *boju*, za krug je moguće pamtiti *boju i debljinu njegovog ruba*, kao i *boju ispune njegove unutrašnjosti*, itd. Dugo vremena, osnovni problem pri vektorskom zapisivanju slika bio je kako predstaviti objekte *nepravilnog izgleda*, npr. raznorazne krivulje, kao i likove nepravilnog oblika. Ovaj problem riješili su matematičari uvodeći razne matematske modele krivih linija i likova, među kojima su najpoznatiji ***kubni splajnovi*** (engl. *cubic splines*) i ***Bézierove krive*** (engl. *Bézier curves*). Oba modela su, matematski gledano, relativno jednostavni, a njihova osnovna ideja je da se pamte koordinate *karakterističnih tačaka* u liniji ili liku, i eventualno, neke dopunske *informacije o stepenu zakrivljenosti* linije ili ruba lika između karakterističnih tačaka. Sljedeća slika prikazuje nekoliko Bezierovih krivih, od kojih je svaka, bez obzira na raznovrsnost oblika, opisana sa svega četiri karakteristične tačke.

Bez obzira na sve, vektorski način zapisa slike dolazi u obzir samo kod slika koje posjeduju određenu dozu pravilnosti, kao što su *crtane slike* (sheme, planovi, dijagrami, nacrti, itd.). Kod potpuno nepravilnih slika bez jasne strukture kao što su *fotografije*, u obzir dolazi samo rasterski način zapisa.

Zvučni podaci dobijaju sve veći značaj u današnje vrijeme, zbog sve raširenijeg pristupa multimedijalnom prezentiranju informacija. Za pamćenje zvučnih podataka postoje dva suštinski različita načina zapisivanja koje možemo nazvati ***uzorkovani*** (***semplovani***) ***zapisi*** ili ***sintetizirani zapisi***. *Uzorkovani zvučni zapisi* zasnivaju se na pamćenju *uzoraka vrijednosti intenziteta zvučnog talasa*, koji je osnovni nosilac zvučnih informacija, uzetih u određenim vremenskim trenucima. Drugim riječima, uzorkovani zvučni zapisi predstavljaju digitalizirani zapis odgovarajućeg analognog nosioca informacije, odnosno zvučnog talasa. S druge strane, *sintetizirani zvučni zapisi* sadrže informacije o visini, trajanju i boji pojedinih *tonova* koji čine zvučnu informaciju. U tom smislu, ovi zvučni zapisi slični su *notnom zapisu* koji se koristi u muzici za zapisivanje muzičkih kompozicija. Sintetizirani zvučni zapisi mogu se primijeniti samo za zapisivanje muzičkih informacija, ali ne i govornih informacija.

Uzorkovani zvučni zapisi, od kojih je najjednostavniji tzv. ***WAV*** zapis, čuvaju se u računarskoj memoriji tako što se prvo izvrši *diskretizacija* uzimanjem uzoraka muzičkog signala, obično uzimanjem 44000 uzorka u sekundi (može i manje, ali se time gubi na kvalitetu zvučnog zapisa – 8000 uzoraka u sekundi daje reprodukciju zvuka kao na lošijim radio prijemnicima), nakon čega se dobijeni uzorci *kvantiziraju*, i zapisuju kao niz binarnih brojeva, obično na 16 bita po uzorku. Na taj način jedna jedina sekunda zvučnog zapisa može zauzeti oko 86 KB, tako da su i za zapise zvuka razvijeni razni metodi kompresovanih zapisa, koje nećemo detaljnije opisivati. Recimo samo to da je najpoznatiji kompresovani uzorkovani zvučni zapis tzv. ***MP3*** (punim imenom ***MPEG–1Layer 3***) zapis, koji je dosta sličan *JPEG* metodu kompresije slika. *MP3* zapis, slično *JPEG* zapisu u slučaju slika, dovodi do izvjesne degradacije kvaliteta zvučnog zapisa. Međutim, praksa je pokazala da se pomoću *MP3* zapisa zvučni zapisi mogu sabiti preko 10 puta a da ljudsko uho ne primijeti nikakvu razliku u kvalitetu u odnosu na *WAV* zapis (ljudsko čulo sluha je mnogo manje osjetljivo na degradacije u kvalitetu nego čulo vida). Pomoću *MP3* zapisa mogu se postići i još veći faktori sabijanja (čak i preko 100 puta), ali u tom slučaju degradacija kvaliteta zvuka postaje sasvim primjetna. Zbog svojih izuzetno povoljnih osobina, *MP3* je danas ubjedljivo najrasprostranjeniji format uzorkovanog zvučnog zapisa

Uzorkovani zvučni zapisi mogu da zapamte bilo kakve zvukove, poput laveža pasa, buke automobila u prometnoj ulici, ili zvuka koji nastaje struganjem stiropora od zid. S druge strane, *sintetizirani zvučni zapisi* su u stanju da zapamte samo zvučne informacije koje imaju *veoma pravilnu strukturu* koja se može opisati nizom tonova (a ne raznih neartikuliranih zvukova), kao što su npr. muzičke kompozicije. Mada su ovakvi zvučni zapisi mnogo ograničeniji od uzorkovanih zvučnih zapisa, oni posjeduju izvjesne bitne prednosti u odnosu na uzorkovane zapise. Pored toga što zauzimaju mnogo manje memorije od uzorkovanih zvučnih zapisa (često je cijelu kompoziciju koja traje nekoliko minuta moguće zapisati u manje od stotinjak kilobajta), sintetizirani zvučni zapisi mogu se reproducirati putem elektronskih muzičkih instrumenata, i računar na osnovu njih može generirati notni zapis kompozicije, što je nemoguće postići pomoću uzorkovanih zvučnih zapisa. Najrašireniji format sintetiziranog zvučnog zapisa je ***MIDI format*** koji je prvenstveno bio razvijen za potrebe upravljanja elektronskim muzičkim instrumentima. Ovaj format zapisa muziku posmatra kao *niz događaja* (engl. *events*) koji su je generirali. Događaji mogu biti npr. pritisak ili otpuštanje neke tipke na klavijaturi sintesajzera, pritisak na dugme sintesajzera kojim se mijenja boja tona, pomjeranje palice kojom se postiže vibrirajući ton, udarac palicom u bubanj ili činelu, itd. MIDI format prosto pamti informaciju o svakom događaju (npr. koja je tipka na klavijaturi pritisnuta i kojom jačinom) kao i tačno vrijeme u kojem se događaj dogodio. Prilikom reprodukcije MIDI zapisa, računar uz pomoć odgovarajućeg hardvera oponaša (simulira) zapisani slijed događaja koji su generirali muziku. Očigledno je da je ovaj vid zapisa idealan za upravljanje elektroničkim muzičkim instrumentima. Također, iz njega je sasvim lako rekonstruirati notni zapis kompozicije.

(?) Pitanja i zadaci

4.1 Kako se po Von Neumannovom modelu vrši pamćenje podataka i programa u računarskoj memoriji?

4.2 Objasnite šta su brojni sistemi.

4.3 Pretvorite broj 13442 iz baze 10 u bazu 7. Provjerite odgovor pretvaranjem dobijenog broja u bazi 7 nazad u bazu 10.

4.4 Pretvorite broj $(413203)_5$ iz baze 5 u bazu 3. (Uputa: prvo izvršite pretvorbu u bazu 10, a zatim iz baze 10 u bazu 3).

4.5 Šta je binarni brojni sistem i zbog čega je on uveden?

4.6 Prema onome što smo do sada saznali, pojam *bit* može označavati čak tri različita, ali ipak vrlo srodna pojma. Navedite sve tri mogućnosti šta ovaj pojam može predstavljati.

4.7 Koji je razlog za uvođenje heksadekadnog brojnog sistema?

4.8 Pretvorite brojeve 25, 53, 176, 500, 1023, 2412, 4000, 33852 i 179978 iz dekadnog u binarni i heksadekadni brojni sistem.

4.9 Pretvorite brojeve, $(1000)_2$, $(110101)_2$, $(111001)_2$ $(10101101101)_2$ i $(1111100110001100110)_2$ u dekadni i heksadekadni brojni sistem.

4.10 Pretvorite brojeve $(372)_{16}$, $(F2)_{16}$, $(ABC)_{16}$, $(7D4)_{16}$, $(8FC)_{16}$, $(FFA)_{16}$, $(1000)_{16}$, $(4077)_{16}$, $(AFE0)_{16}$ i $(FFFF)_{16}$ u dekadni i binarni brojni sistem.

4.11 Prije uvođenja heksadekadnog sistema, u računarskim naukama je mnogo u upotrebi bio *oktalni brojni sistem*, koji je koristio bazu 8. Prednost ovog sistema u odnosu na heksadekadni je u tome što se koriste samo standardne cifre $0-7$. Pretvorite dekadne brojeve 127, 798, 1000 i 10000 u oktalni brojni sistem, a brojeve $(101)_8$, $(757)_8$, $(1000)_8$, $(4077)_8$ i $(77777)_8$ u dekadni brojni sistem.

4.12 Razlog za upotrebu oktalnog sistema je što on, slično heksadekadnom brojnom sistemu, također omogućava vrlo jednostavna pravila pretvorbe između oktalnog i binarnog brojnog sistema, koja se mogu obaviti napamet (ipak, heksadekadni sistem se pokazao praktičnijim za upotrebu u računarskim naukama). Otkrijte ova pravila, i pretvorite brojeve $(11001)_2$, $(1110011)_2$, $(1011011)_2$ i $(1001000111000101)_2$ u oktalni brojni sistem, a zatim pretvorite brojeve $(101)_8$, $(252)_8$ i $(4077)_8$ u binarni brojni sistem.

4.13 * Pretvorite brojeve $(101)_8$, $(252)_8$ i $(4077)_8$ u bazu 16, kao i brojeve $(8FC)_{16}$, $(4077)_{16}$ i $(AFE0)_{16}$ u bazu 8, *bez njihovog prethodnog pretvaranja u neku drugu pomoćnu bazu* (ovo traži malo više razmišljanja i koncentracije).

4.14 * Vanzemaljci sa planete Gongo koriste *kvaternarni brojni sistem*, koji koristi bazu 4 (valjda zbog toga što imaju samo jednu ruku, sa četiri prsta). Naravno, cifre kvaternarnog brojnog sistema su 0, 1, 2 i 3, ali se oni na jeziku stanovnika planete Gongo bilježe redom kao ᘐ, ᘑ, ᘓ i ᘏ. Izvršite sve navedene pretvorbe: Dekadni broj 597 u Gongo sistem (kvaternarni sistem, zapisan ciframa jezika Gongo); Broj ᘏᘓᘐᘓᘑ u dekadni sistem; Broj ᘓᘓᘏ u binarni sistem; Broj ᘏᘓᘓᘑᘐ u oktalni sistem; Broj ᘑᘓᘏᘐᘏᘓᘓ u heksadekadni sistem; Heksadekadni broj $(AB0)_{16}$ u Gongo sistem; Binarni broj $(110010011)_2$ u Gongo sistem; Oktalni broj $(771)_8$ u Gongo sistem. Sve pretvorbe izvršite neposredno, bez upotrebe ikakve pomoćne baze.

4.15 Koliki je najveći broj koji može stati u 24 bita ukoliko se ograničimo samo na nenegativne cijele brojeve (tj. ukoliko se ne vrši pamćenje znaka)?

4.16 Koliki je opseg cijelih brojeva koji može stati u 24 bita ukoliko se pamti i znak?

4.17 Pretvorite realni broj $(1101{,}01101)_2$ iz binarnog brojnog sistema u dekadni brojni sistem.

4.18 * Pretvorite realni broj 17,8125 u binarni brojni sistem. (Uputa: pretvorite prvo cijeli dio, a zatim razmislite kako bi se mogla izvršiti pretvorba decimalnog dijela).

4.19 * Pretvorite realni broj 15,3 u binarni brojni sistem. (Napomena: nemojte se iznenaditi ukoliko dobijete beskonačan slijed cifara iza zareza, koje se periodično ponavljaju).

4.20 Objasnite metod "razlomak" za pamćenje realnih brojeva u računarskoj memoriji, i ukazati na njegove dobre i loše strane.

4.21 Objasnite metod "fiksni zarez" za pamćenje realnih brojeva u računarskoj memoriji, i ukazati na njegove dobre i loše strane. Ilustrirajte ovaj metod navođenjem barem tri primjera zapisa pomoću ovog metoda uz pretpostavku da je broj decimala fiksiran na 3.

4.22 Objasnite metod "pokretni zarez" za pamćenje realnih brojeva u računarskoj memoriji, i navesti koje su prednosti i mane ovog zapisa u odnosu na ostale metode. Ilustrirajte ovaj način zapisa na nekoliko primjera.

4.23 Vidjeli smo da se pri zapisu realnih brojeva u računarskoj memoriji pomoću metoda pokretnog zareza, svaki realan broj $x \neq 0$ jednoznačno se zapisuje u obliku $x = \pm m \cdot 2^e$, gdje je e cijeli broj (tzv. *eksponent*), a m realan broj (tzv. *mantisa*) u opsegu $1/2 \leq m < 1$. Odredite eksponent i mantisu za brojeve $x = 18$ i $x = 0.1$.

4.24 Opišite šta se može učitniti po pitanju pamćenja iracionalnih brojeva u računarskoj memoriji.

4.25 Objasnite šta je ASCII, a šta UNICODE standard.

4.26 Izračunajte koliko se znakova može zapisati na jednu disketu ukoliko se koristi ASCII, a koliko ukoliko se koristi UNICODE standard.

4.27 Program "Microsoft Word 2000" za pamćenje znakova u memoriji koristi UNICODE standard. Ukoliko je poznato da "Microsoft Word 2000" za pamćenje raznih sistemskih informacija (poput širine margina, veličine stranice itd.) koristi fiksan prostor od 19.5 kilobajta memorije (neovisno od količine ukucanog teksta), i ukoliko pretpostavimo da jedna stranica gusto kucanog teksta ima u prosjeku 2500 znakova, izračunajte koliko (u prosjeku) stranica čistog teksta pisanog u programu "Microsoft Word 2000" može stati na jednu disketu.

4.28 Opišite suštinu rasterskog načina zapisa slika u računarskoj memoriji.

4.29 Objasnite šta je paleta i kada se koristi.

4.30 Objasnite zbog čega su uvedeni postupci za kompresovano zapisivanje rasterskih slika, i navesti neke od metoda za kompresovano zapisivanje.

4.31 Odredite koliko se slika formata 1024×768 piksela sa 256 boja zapisanih u rasterskom zapisu može pohraniti na jedan kompakt disk kapaciteta 650 MB, uz pretpostavku da se ne koristi nikakav metod kompresije.

4.32 Na nekom kompakt disku pohranjen je izvjesan broj digitaliziranih slika od kojih je svaka formata 1024×768 piksela, pri čemu se za pamćenje informacije o boji svake tačke koristi 16 bita po pikselu.

a) Koliko maksimalno boja mogu sadržavati ovakve slike?

b) Koliko ovakvih slika može stati na kompakt disk kapaciteta 650 MB ukoliko slike nisu kompresovane ni na kakav način?

c) Koliki bi trebao biti faktor kompresije (tj. odnos nekompresovane i kompresovane veličine) svake slike da bi na isti kompakt disk stalo 1500 slika?

4.33 Koliko rasterskih slika formata 1024×768 piksela sa True Color zapisom boja može stati na jednu disketu ukoliko se koristi neki od metoda kompresije sa faktorom kompresije 10?

4.34 U radnu memoriju nekog računara pohranjen je određeni broj slika formata 512×256 piksela, pri čemu svaki od piksela može imati jednu od 16 mogućih boja.

a) Koliko je ovakvih slika moguće snimiti na jednu disketu?

b) Koliko se ovakvih slika može prenijeti za sat vremena sa jednog računara na drugi uz pretpostavku da su računari povezani modemskom vezom u kojoj je brzina prenosa podataka 56 kilobita u sekundi?

4.35 Objasnite suštinu vektorskog zapisa slika u računarskoj memoriji.

4.36 Objasnite kakve su prednosti odnosno mane vektorskog u odnosu na rasterski zapis slika u računarskoj memoriji.

4.37 Objasnite čemu služe kubni splajnovi i Bézierove krive.

4.38 Objasnite suštinu zapisivanja zvučnih podataka u računarskoj memoriji.

4.39 Objasnite razliku između uzorkovanih i sintetiziranih zvučnih zapisa.

5. Binarna aritmetika i komplement kôdovi

Već smo rekli da digitalni računari sve podatke pamte isključivo kao brojeve u binarnom brojnom sistemu. Da bismo stekli elementarnu predstavu o tome kako računar uopće može išta računati, moramo se prvo spustiti na malo niži nivo i vidjeti kako se izvode osnovne računske operacije u ovom brojnom sistemu.

Za *sabiranje* binarnih brojava vrijedi ista logika računanja kao za dekadne brojeve osim što sabiranje brojeva 1 i 1 ne daje 2 nego 10 (u binarnom sistemu), tako da nulu pišemo a jedinicu prenosimo. Demonstrirajmo ovu operaciju na jednom primjeru:

```
   1101010
+101011001
 111000011
```

Dakle, $(1101010)_2 + (101011001)_2 = (111000011)_2$. Pretvorimo li ove brojeve u dekadni sistem, dobijamo jednakost 106 + 345 = 451, što potvrđuje tačnost računa.

Kod *oduzimanja* se također koristi sličan postupak, pri čemu kada oduzimamo manju cifru od veće, moramo izvršiti "pozajmicu" koju ćemo vratiti kasnije. Ilustrirajmo postupak na slijedećem primjeru. Kako je $(1101010)_2 + (101011001)_2 = (111000011)_2$, i kako je oduzimanje inverzna operacija sabiranju, sasvim je jasno da mora da vrijedi $(111000011)_2 - (1101010)_2 = (101011001)_2$. Provjerimo ovo neposrednim izračunavanjem:

```
 111000011
-  1101010
 101011001
```

Interesantno je da su postupci *množenja* i *dijeljenja* binarnih brojeva posebno jednostavni, čemu se može zahvaliti "popularnost" binarnih brojeva u računskoj tehnici. Na primjer, množenje u binarnom brojnom sistemu je posebno jednostavno u odnosu na dekadni sistem, jer se čitava tablica množenja koju treba zapamtiti svodi samo na trivijalnu tablicu

```
0 · 0 = 0
0 · 1 = 0
1 · 0 = 0
1 · 1 = 1
```

odnosno, u formi tablice:

·	0	1
0	0	0
1	0	1

Demonstrirajmo kompletan postupak množenja dva binarna broja na jednom primjeru:

```
  1101010 · 11010
   0000000
  1101010
 0000000
1101010
+1101010
1010110000100
```

Isti postupak smo mogli zapisati i kraće, ispuštajući parcijalne produkte sa nulama:

```
  1101010 · 11010
  1101010
1101010
+1101010
1010110000100
```

Radi potpunosti, demonstriraćemo i operaciju *dijeljenja*, koja se u binarnom brojnom sistemu izvodi naročito efikasno. Podsjetimo se prvo dijeljenja dva dekadna broja, na jednom primjeru:

```
 473701219131 : 33271 = 14237661
-33271
 140991
-133084
  79072
 -66542
  125301
 - 99813
   254889
  -232897
    219921
   -199626
     202953
    -199626
       33271
      -33271
            0
```

Najveći problem pri dijeljenju dekadnih brojeva je određivanje koliko puta djelilac "ide" u neki od faktora. U binarnom sistemu vrijedi ista logika računanja, ali kako svaka cifra količnika može biti samo 0 ili 1, nije potrebno oađredivati koliko puta djelilac "ide" u faktor (ili "ide", kada je cifra 1, ili "ne ide" kada je cifra 0). Ilustrirajmo ovo na jednom primjeru, koji je inverzan maloprije navedenom primjeru za množenje binarnih brojeva:

```
 101011000100 : 11010 = 1101010
- 11010
  100010
- 11010
   100000
  - 11010
      11010
     -11010
          00
         - 0
           0
```

Naročito je interesantno primijetiti da je u binarnom brojnom sistemu veoma lako realizirati množenje odnosno dijeljenje sa stepenima broja 2. Naime, kako se broj 2^n u binarnom brojnom sistemu piše kao jedinica iza koje slijedi n nula, slijedi da se binarni broj množi sa 2^n prosto tako što se na njegov kraj zdesna dopiše n nula (ovo je analogno sa množenjem sa 10^n u dekadnom brojnom sistemu). Slično, dijeljenje sa 2^n realizira se uklanjanjem posljednjih n cifara zdesna u binarnom zapisu broja, pri čemu je broj tačno djeljiv sa 2^n jedino ukoliko su svih posljednjih n cifara nule (u suprotnom, nakon uklanjanja cifara dobićemo cijeli dio količnika). U slučaju da binarne brojeve koje množimo odnosno dijelimo sa 2^n posmatramo kao brojeve zapisane sa *fiksnim brojem bita*, množenje odnosno dijeljenje sa 2^n svodi se na *pomjeranje* svih bita za n mjesta ulijevo odnosno udesno, pri čemu se na mjestu nedostajućih bita upisuju *nule*. Na primjer, broj 26 se u binarnom zapisu na 8 bita može predstaviti kao 00011010. Njegovo množenje sa 2^2 ($n = 2$) u zapisu sa 8 bita daje binarni broj 001101000, dok dijeljenje sa 2^2 daje binarni broj 00000110 (naravno, ovo je cijeli dio količnika, jer broj 26 nije tačno djeljiv sa 4).

Iz elementarne matematike znamo da postupak sabiranja dva cijela broja zavisi od toga jesu li oni istog znaka ili nisu. Jasno je da ista logika mora vrijediti i za binarni brojni sistem. Međutim, pronađen je interesantan način zapisa negativnih brojeva u binarnom brojnom sistemu koji omogućava da potpuno istim postupkom sabiramo i brojeve sa predznakom kao i brojeve bez predznaka neovisno od toga kakav im je predznak. To znatno pojednostavljuje konstrukciju računarskih sklopova koji obavljaju računske operacije. Taj specijalan način zapisa negativnih brojeva u binarnom brojnom sistemu zovemo "***drugi***

komplement kôd“ ili, skraćeno “***2-komplement kôd***” (***2KK***), “***kôd drugog komplementa***”, “***kôd komplementa dvojke***”, ili ponekad samo “***komplement dvojke***”.

Osnovna ideja za *2KK* je da se negativni binarni brojevi ne čuvaju na ranije opisani način u računarskoj memoriji (po sistemu “znak i vrijednost”) nego da se ukoliko je broj negativan, svaka njegova cifra izvrne (komplementira) tj. odbije od jedinice (time se dobija tzv. “***prvi komplement kôd***” ili “***komplement jedinice***”, skraćeno ***1KK***) a nakon toga se na tako izvrnut broj dodaje jedinica i dobiveni zapis čuva u memoriji.

Ilustrirajmo *2KK* zapis na jednom primjeru. Pogledajmo kako bi se u 8 bita zapisali brojevi 19, 10, -19 i -10 koristeći metod *2KK* i demonstrirajmo da se računska operacija sabiranja odvija na isti način bez obzira na znak brojeva. Kako je $19 = (10011)_2$ i $10 = (1010)_2$, što je lako provjeriti, brojevi 19 i 10 zapisani su u memoriji na sljedeći način:

19:	0	0	0	1	0	0	1	1
10:	0	0	0	0	1	0	1	0

Odredimo sada zapis brojeva –19 i –10 po metodu *2KK*:

```
–19:  11101100          –10:  11110101
    +       1               +       1
      11101101                11110110
```

Odavde neposredno slijedi zapis brojeva –19 i –10 po metodu 2KK:

–19:	1	1	1	0	1	1	0	1
–10:	1	1	1	1	0	1	1	0

Kao što je već rečeno, ovaj na prvi pogled čudan metod se najviše koristi, jer ima tu prednost da računske operacije sabiranja i oduzimanja izvodimo istim postupkom neovisno od znaka operanada, ukoliko se dogovorimo da eventualni “bit viška” (9-ti bit u našem primjeru) koji se eventualno pojavi pri računanju ignoriramo. Provjerimo npr. sabiranje 19 + (–10) = 9:

```
  00010011
+ 11110110
 100001001
 ↓
```
ignoriramo

Vidimo da nakon ignoriranja suvišne jedinice dobijamo ispravan rezultat. Provjerimo sada sabiranje 10 + (–19) = –9:

```
  00001010
+ 11101101
  11110111
  ↓
```
negativan broj

Vidimo da je rezultat zaista negativan broj (jer je najviši bit jednak jedinici), ali nije lako neposredno pročitati koji je ovo broj. Da bismo vidjeli koliki je zapravo rezultat, moramo rasplesti *2KK* zapis da dobijemo čovjeku razumljiviji zapis. Dakle, odbijmo jedinicu od onoga što smo dobili, a nakon toga izvrnemo sve cifre:

```
 11110111
-       1
 11110110  ⟶  00001001 ⟶ 9
```

Ovim smo potvrdili ispravnost rezultata. Čitateljima i čitateljkama se ostavlja da sami provjere da sabiranje $(-19) + (-10)$ daje također korektan rezultat –29.

Lako je provjeriti da *2KK* zapis čini i operaciju oduzimanja neovisnim od toga da li brojevi imaju znak ili ne (pri tome treba ignorirati eventualnu "posudbu" koju bi trebalo izvršiti ukoliko kod oduzimanja cifara najviše težine treba oduzeti veći broj od manjeg). Zbog ovog poželjnog svojstva, kao i zbog činjenice da u ovom načinu zapisa ne postoji "negativna nula" (lako je provjeriti da je 10000000 zapis broja –128), ovaj način zapisa negativnih brojeva se gotovo isključivo danas koristi za zapis negativnih brojeva u računarskoj memoriji. Međutim, vrijedi napomenuti da se sa *2KK* zapisom ne mogu ispravno realizirati operacije množenja i dijeljenja. Stoga, prilikom realizacije množenja i dijeljenja, uvijek treba izvoditi operacije samo sa *apsolutnim vrijednostima brojeva*, a nakon toga znak rezultata odrediti na osnovu znaka operanada.

Opisano "magično" svojstvo neovisnosti operacija sabiranja i oduzimanja od znaka brojeva moguće je postići i u dekadnom brojnom sistemu uz drugačiji sistem zapisa negativnih brojeva od onog koji smo navikli (cifre se tada ne odbijaju od jedinice nego od devetke, ili još općenitije, od najveće cifre razmatranog brojnog sistema). Na primjer, jasno je da vrijedi $37728 + (–2356) = 37728 – 2356 = 35372$. Pokažimo kako bi se ovo sabiranje moglo izvesti kada bismo negativni broj –2356 predstavili u odgovarajućem komplement kôdu. Kako broj 2356 ima jednu cifru manje od broja 37728, dopunićemo ga nulom na početku, tako da ćemo dobiti 02356. Odbijemo li svaku cifru od devetke, dobijamo broj 97643. Dodamo li jedinicu na ovaj broj, dobijamo 97644. Slijedi da se negativni broj –2356 u odgovarajućem komplement kôdu ("***komplement desetke***", ili ***10KK***) zapisuje kao 97644. Provjerimo sada da sabiranje $37728 + (–2356)$ daje ispravan rezultat korištenjem komplementa desetke. Zaista imamo $37728 + 97644 = 135374$, što se poklapa sa rezultatom sabiranja $37728 + (–2356)$, ukoliko ignoriramo "suvišnu" šestu cifru.

Iz izloženog primjera se vidi da komplement kôdovi zaista nisu nikakvo "magično" svojstvo binarnog brojnog sistema,. nego da vrijede za proizvoljan brojni sistem. Zaista, sasvim je lako provjeriti da ukoliko je baza brojnog sistema B, tada odgovarajući komplement kôd broja X iznosi $X_{KK} = B^n - X$, gdje je *n* broj cifara broja. Sada, ukoliko umjesto sabiranja $X + (–Y)$ izvršimo sabiranje $X + Y_{KK}$, dobijamo $X + Y_{KK} = X + (B^n - Y) = X + (–Y) + B^n$, što se poklapa sa tačnim rezultatom ukoliko ignoriramo član B^n koji se pojavljuje upravo kao "cifra viška". Na sličan način se može opravdati i izvođenje operacije oduzimanja.

Na kraju vrijedi napomenuti da se bez obzira na veliki značaj komplement kôdova u računarskoj tehnici, komplement kôdovi praktično koriste isključivo u binarnom brojnom sistemu. Naime, prednost komplement kôdova je samo u tome što ukidaju razliku između tretmana pozitivnih i negativnih brojeva, što može osjetno pojednostavniti građu računara, koji svakako rade u binarnom brojnom sistemu. S druge strane, čovjeku koji je navikao raditi sa dekadnim brojnim sistemom, komplement kôdovi donijeli bi mnogo više problema nego koristi, jer bi čovjeku zapis negativnih brojeva u *10KK* komplement kôdu, na koji nije navikao, bio potpuno "stran" i "neprirodan". Ipak, treba napomenuti da su se ranije, radi lakšeg ručnog računanja prilikom korištenja logaritamskih tablica, mantise logaritama brojeva manjih od jedinice izražavale na način koji u suštini nije ništa drugo nego *10KK* komplement kôd!

(?) Pitanja i zadaci

5.1 Izračunajte sljedeće zbirove, bez prethodnog pretvaranja operanada u dekadni brojni sistem:

a) $(100110)_2 + (111)_2$ b) $(110111)_2 + (101)_2$ c) $(111110)_2 + (10111)_2$
d) $(111001)_2 + (10001)_2$ e) $(11011100110)_2 + (10011001)_2$ f) $(1111111)_2 + (10101010)_2$

Sve rezultate provjerite pretvaranjem operanada i rezultata u dekadni brojni sistem.

5.2 Izračunajte sljedeće razlike, bez prethodnog pretvaranja operanada u dekadni brojni sistem:

a) $(100110)_2 - (111)_2$ b) $(110111)_2 - (101)_2$ c) $(111110)_2 - (10111)_2$
d) $(111001)_2 - (10001)_2$ e) $(11011100110)_2 - (10011001)_2$ f) $(1111111)_2 - (10101010)_2$

Sve rezultate provjerite pretvaranjem operanada i rezultata u dekadni brojni sistem.

5.3 Izračunajte sljedeće proizvode, bez prethodnog pretvaranja operanada u dekadni brojni sistem:

a) $(100110)_2 \cdot (111)_2$ b) $(110111)_2 \cdot (101)_2$ c) $(111110)_2 \cdot (10111)_2$
d) $(111001)_2 \cdot (10001)_2$ e) $(11011100110)_2 \cdot (10011001)_2$ f) $(1111111)_2 \cdot (10101010)_2$

Sve rezultate provjerite pretvaranjem operanada i rezultata u dekadni brojni sistem.

5.4 Izračunajte sljedeće količnike, bez prethodnog pretvaranja operanada u dekadni brojni sistem:

a) $(10000101)_2 : (111)_2$ b) $(100110101011)_2 : (110111)_2$ c) $(11110011011)_2 : (100001)_2$
d) $(10101011)_2 : (110000)_2$ e) $(100101)_2 : (1010)_2$ f) $(101)_2 : (11000)_2$

Sve rezultate provjerite pretvaranjem operanada i rezultata u dekadni brojni sistem (Napomena: u posljednja tri slučaja količnici *nisu cijeli brojevi*).

5.5 Računske operacije u drugim brojnim sistemima mogu se realizirati na analogan način kao u dekadnom i binarnom brojnom sistemu. Obavite sljedeće računske operacije, bez prethodnog pretvaranja operanada u dekadni brojni sistem:

a) $(21022)_3 + (10221)_3$ b) $(13203)_5 + (4413)_5$ c) $(37127)_8 + (14736)_8$ d) $(FC1B)_{16} + (4AD)_{16}$
e) $(21022)_3 - (10221)_3$ f) $(13203)_5 - (4413)_5$ g) $(37127)_8 - (14736)_8$ h) $(FC1B)_{16} - (4AD)_{16}$

Sve rezultate provjerite pretvaranjem operanada i rezultata u dekadni brojni sistem.

5.6 * Za obavljanje operacija množenja i dijeljenja u dekadnom brojnom sistemu potrebno je napamet poznavati tablicu množenja (za bazu 10). Analogna tablica množenja za bazu 2 je trivijalna. Sastavite tablice množenja za baze 3, 5, 8 i 16.

5.7 Uz pomoć tablica množenja sastavljenih u zadatku 5.6, izračunajte sljedeće proizvode, bez prethodnog pretvaranja operanada u dekadni brojni sistem:

a) $(21022)_3 \cdot (10221)_3$ b) $(13203)_5 \cdot (4413)_5$ c) $(37127)_8 \cdot (14736)_8$ d) $(FC1B)_{16} \cdot (4AD)_{16}$

Sve rezultate provjerite pretvaranjem operanada i rezultata u dekadni brojni sistem.

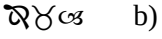
5.8 * Uz pomoć tablica množenja sastavljenih u zadatku 5.6, izračunajte sljedeće količnike, bez prethodnog pretvaranja operanada u dekadni brojni sistem:

a) $(2202121)_3 : (201)_3$ b) $(1434030)_5 : (32)_5$ c) $(51354)_8 : (127)_8$ d) $(36F902)_{16} : (B9)_{16}$

Sve rezultate provjerite pretvaranjem operanada i rezultata u dekadni brojni sistem.

5.9 * Sastavite tablicu množenja za brojni sistem koji koriste vanzemaljci sa planete Gongo, opisan u zadatku 4.14.

5.10 * Obavite sljedeća izračunavanja u brojnom sistemu vanzemaljaca sa planete Gongo, bez prethodnog pretvaranja operanada u (zemaljski) dekadni brojni sistem, koristeći se eventualno tablicom množenja sastavljenom u zadatku 5.9:

a)  b) c)  d)

Sve rezultate provjerite pretvaranjem operanada i rezultata u dekadni brojni sistem.

5.11 Objasnite kako se u binarnom brojnom sistemu vrši množenje i dijeljenje sa 2^n. Ilustrirajte opisani postupak na nekoliko primjera.

5.12 Objasnite kako se specijalizirani postupak množenja odnosno dijeljenja binarnih brojeva sa 2^n može generalizirati za slučaj brojeva zapisanih u proizvoljnoj bazi $B > 1$.

5.13 Objasnite smisao uvođenja komplement kôdova za zapis negativnih brojeva, i način kako se formiraju komplement kôdovi u binarnom brojnom sistemu.

5.14 Pretvoriti brojeve 16 i –39 u zapis po 2-komplement kôdu (*2KK*) na 8 bita. Sabrati dobijene binarne brojeve i provjeriti da li se dobija tačan rezultat –23.

5.15 Ponoviti račun iz zadatka 5.14, ali posmatrati zapis po 2-komplement kôdu na 16 bita.

5.16 Na primjeru zapisa brojeva 16 i –39 u *2KK* zapisu na 8 bita, provjeriti da izračunavanje izraza 16 – (–39) i –39 – 16 sa tako zapisanim brojevima daje ispravne rezultate 55 odnosno –55.

5.17 Na primjeru zapisa brojeva 16 i –39 u *2KK* zapisu na 8 bita, pokazati da komplement kodovi *nisu podesni* za izvođenje operacije množenja.

5.18 * Izračunati zbirove –372 + 23441, 372 + (–23441) i –372 + (–23441) koristeći kôd komplementa desetke (*10KK*).

5.19 Naći odgovarajuće komplement kôdove za sljedeće brojeve, u skladu sa bazom u kojoj su zapisani:

a) $-(212020)_3$ b) $-(34103)_5$ c) $-(4271360)_7$ d) $-(C824BF)_{16}$

6. Osnove logičke algebre

Za neposrednu hardversku realizaciju u računaru, čak i tako jednostavna operacija poput sabiranja u tako primitivnom brojnom sistemu kao sto je binarni brojni sistem nije nimalo jednostavna. Stoga je potrebno uvesti još elementarnije operacije koje se izvode samo nad ciframa 0 i 1, a koje će biti dovoljno univerzalne da se sve ostale neophodne operacije mogu svesti na njih. Sve računske operacije koje se izvode samo nad ciframa 0 i 1, a koje kao rezultat daju ponovo cifru 0 ili 1 nazivamo ***logičke operacije***, a granu matematike koja proučava takve operacije nazivamo ***logička algebra*** (susreće se i naziv ***prekidačka algebra***). Napomenimo da logička algebra ima dosta sličnosti sa tzv. ***algebrom iskaza*** u matematičkoj logici, mada nisu posve identične.

Među najvažnije operacije logičke algebre ubrajamo ***negaciju***, ***konjunkciju*** (izgovara se sa "n" i "j" odvojeno) i ***disjunkciju***. Ove operacije se definiraju na sličan način kao u algebri iskaza u matematičkoj logici, s tim da se umjesto simbola "tačno" i "netačno" (odnosno "⊤ " i "⊥") koriste cifre "1" i "0", pri čemu cifri "1" odgovara logička vrijednost "tačno" (odnosno "⊤ "), a cifri "0" logička vrijednost "netačno" (odnosno "⊥"). U nastavku su prikazane tablice ovih elementarnih operacija:

Negacija:

A	$\overline{A}$
0	1
1	0

U matematičkoj logici negacija promjenljive A obično se obilježava sa $\neg A$, dok je u računarskoj tehnici iz praktičnih razloga mnogo uobičajenije nadvlačenje promjenljive linijom, kao što je prikazano u tablici s lijeve strane.

Konjunkcija:

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

U matematičkoj logici, konjunkcija promjenljivih A i B označava se sa $A \wedge B$. Kako se rezultat operacije konjunkcije poklapa sa sa običnim produktom u slučaju kada operande označavamo sa 0 i 1, u računarskoj tehnici se obično radi kratkoće umjesto $A \wedge B$ piše samo AB (ili $A \cdot B$), kao da se radi o običnom produktu.

Disjunkcija:

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

U matematičkoj logici, disjunkcija promjenljivih A i B označava se sa $A \vee B$. U računarskoj tehnici se umjesto znaka $\vee$ često koristi znak +, kao da se radi o običnom sabiranju, ali to može unijeti zabunu (s obzirom da je $1 \vee 1 \neq 1 + 1$), tako da mi nećemo koristiti tu konvenciju.

Mada ćemo kasnije pokazati da se sve operacije koje se izvode samo nad ciframa 0 i 1 mogu svesti na tri opisane operacije (negaciju, konjunkciju i disjunkciju), za računarsku tehniku je pogodno uvesti još neke logičke operacije. Među njima je naročito značajna ***ekskluzivna disjunkcija***:

Ekskluzivna disjunkcija:

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

U matematičkoj logici ova operacija obično se obilježava znakom $\veebar$. Međutim, u računarstvu se mnogo češće upotrebljava znak $\oplus$, stoga što, formalno gledano, ova operacija ima mnoga srodna svojstva sa klasičnim sabiranjem (strogo rečeno, ona zapravo predstavlja sabiranje po modulu 2).

Četiri opisane operacije (negacija, konjunkcija, disjunkcija i ekskluzivna disjunkcija) veoma se često označavaju engleskim nazivima ***NOT***, ***AND***, ***OR*** i ***EXOR*** respektivno.

Operacije ***implikacija*** i ***ekvivalencija***, koje imaju veliku primjenu u matematičkoj logici, nemaju kao takve neku osobitu primjenu u računarskoj tehnici. S druge strane, dvije kombinirane operacije ***negacija konjunkcije*** i ***negacija disjunkcije***, koje su predstavljene sljedećim tablicama, izuzetno su značajne za digitalnu računarsku tehniku:

A	B	$\overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

A	B	$\overline{A \vee B}$
0	0	1
0	1	0
1	0	0
1	1	0

Ove dvije operacije imaju i svoja posebna imena. Tako se negacija konjunkcije naziva ***Shefferova operacija***, dok se negacija disjunkcije naziva ***Pierceova operacija*** (ili ***operacija Lukasiewitza***). Obilježavaju se znakovima $\uparrow$ i $\downarrow$:

$$A \uparrow B = \overline{AB}, \quad A \downarrow B = \overline{A \vee B}$$

Najčešće se ipak ove dvije operacije imenuju njihovim engleskim nazivima: ***NAND*** i ***NOR***.

Grana matematike koja proučava operacije konjunkcije, disjunkcije i negacije, kao i pravila rada sa njima, naziva se ***Booleova algebra*** (po *Georgu Booleu*, osnivaču matematičke logike). U tom smislu se Booleova algebra može smatrati kao podskup logičke algebre. S druge strane, Booleova algebra je, formalno gledano, ipak zasebna grana matematike, jer se operacije Booleove algebre (konjunkcija, disjunkcija i negacija) mogu definirati i na širem skupu vrijednosti koje uključuju i vrijednosti različite od 0 i 1, dok se standardna logička algebra definira isključivo na skupu $\{0, 1\}$ (primjeri takvih proširenja ilustrirani su u zadacima na kraju ovog poglavlja). Međutim takva proširenja nisu od interesa za računarsku tehniku.

Svaka funkcija koja se može izraziti samo pomoću logičkih operacija naziva se ***logička funkcija***. Na primjer, funkcija

$$Y = \left[A\overline{B} \vee \overline{C(\overline{A} \vee B)}\right] \vee C\overline{A \vee \overline{B}C}$$

predstavlja logičku funkciju. Primijetimo da linija koja nadvlači dio izraza, a ne samo individualnu promjenljivu, predstavlja negaciju čitavog dijela izraza iznad kojeg se nalazi linija. Isti izraz bi se korištenjem standardne simbolike matematičke logike, tj. korištenjem simbola "$\wedge$", "$\vee$" i "$\neg$" za operacije konjunkcije, disjunkcije i negacije respektivno, mogao napisati ovako:

$$Y = \{(A \wedge \neg B) \vee \neg[C \wedge (\neg A \vee B)]\} \vee \{C \wedge \neg[A \vee (\neg B \wedge C)]\}$$

U prethodnom izrazu moguće je ispustiti neke zagrade ukoliko usvojimo da operacija konjunkcije ima viši prioritet u odnosu na operaciju disjunkcije:

$$Y = \{A \wedge \neg B \vee \neg[C \wedge (\neg A \vee B)]\} \vee C \wedge \neg(A \vee \neg B \wedge C)$$

Bez obzira na to, može se primijetiti da je zapis koji smo usvojili mnogo kompaktniji i pregledniji (posebno način na koji se zapisuje operacija negacije). Mogućnost kompaktnijeg zapisivanja logičkih izraza ujedno predstavlja i osnovni razlog zbog kojeg se u digitalnoj i računarskoj tehnici koristi način obilježavanja logičkih operacija koji smo usvojili, a ne klasični način obilježavanja poznat iz matematičke logike.

Logičke funkcije se najčešće predstavljaju tabelom svojih vrijednosti za sve moguće kombinacije vrijednosti promjenljivih koje se javljaju u funkciji. Ove tabele nazivamo i ***tabelom istine*** logičke funkcije (s obzirom da se tako nazivaju odgovarajuće tablice u matematičkoj logici).

Primjer 6.1:

- Formirati tabelu istine za prethodno napisanu funkciju.

Tabele istine za složenije funkcije kreiraju se postupno, tako što se prvo odrede vrijednosti manjih članova, pa se nakon toga prelazi na formiranje sve većih i većih članova:

A	B	C	$\overline{B}$	$A\overline{B}$	$\overline{A}$	$\overline{A}\vee B$	$C(\overline{A}\vee B)$	$\overline{C(\overline{A}\vee B)}$	$A\overline{B}\vee\overline{C(\overline{A}\vee B)}$	$\overline{B}C$	$A\vee\overline{B}C$	$\overline{A\vee\overline{B}C}$	$CA\vee\overline{\overline{B}C}$	Y
0	0	0	1	0	1	1	0	1	1	0	0	1	0	1
0	0	1	1	0	1	1	1	0	0	1	1	0	0	0
0	1	0	0	0	1	1	0	1	1	0	0	1	0	1
0	1	1	0	0	1	1	1	0	0	0	0	1	1	1
1	0	0	1	1	0	0	0	1	1	0	1	0	0	1
1	0	1	1	1	0	0	0	1	1	1	1	0	0	1
1	1	0	0	0	0	1	0	1	1	0	1	0	0	1
1	1	1	0	0	0	1	1	0	0	0	1	0	0	0

U tablicama istine, promjenljive koje ulaze u funkciju (u našem primjeru A, B i C) i njihove vrijednosti uvijek se pišu takvim redoslijedom tako da posljednja (krajnja desna) promjenljiva mijenja svoju vrijednost u svakom redu tablice, sljedeća promjenljiva (gledano zdesna na lijevo) mijenja svoju vrijednost svaka dva reda, sljedeća svaka četiri reda, i tako dalje. Na taj način, vrijednosti svih promjenljivih posmatrane po redovima kao binarni brojevi (000, 001, 010, 011, 100, itd. sve do 111), nakon pretvaranja u dekadni brojni sistem daju slijed brojeva 0, 1, 2, 3 itd. do $2^n - 1$ gdje je n ukupan broj promjenljivih.

Primjer 6.2:

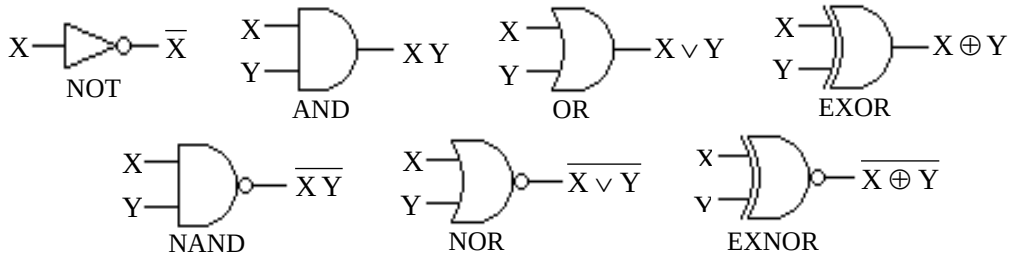
- Formirati tablicu istine za funkciju $Y = \overline{\overline{ABC}\,\overline{C\vee D}\vee D}$ (bez prethodnog pojednostavljivanja).

Ovu tablicu istine formiramo na sličan način, pri čemu ovdje imamo $2^4 = 16$ redova tablice:

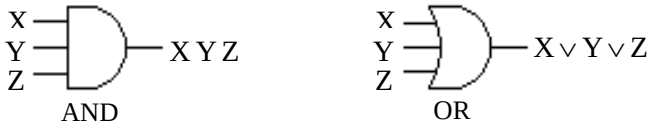
A	B	C	D	$\overline{ABC}$	$\overline{C\vee D}$	$\overline{ABC}\,\overline{C\vee D}$	$\overline{\overline{ABC}\,\overline{C\vee D}\vee D}$	Y
0	0	0	0	1	1	1	1	0
0	0	0	1	1	0	0	1	0
0	0	1	0	1	0	0	0	1
0	0	1	1	1	0	0	1	0
0	1	0	0	1	1	1	1	0
0	1	0	1	1	0	0	1	0
0	1	1	0	1	0	0	0	1
0	1	1	1	1	0	0	1	0
1	0	0	0	1	1	1	1	0
1	0	0	1	1	0	0	1	0
1	0	1	0	1	0	0	0	1
1	0	1	1	1	0	0	1	0
1	1	0	0	1	1	1	1	0
1	1	0	1	1	0	0	1	0
1	1	1	0	0	0	0	0	1
1	1	1	1	0	0	0	1	0

Jasno je da tablica istine za funkciju koja ima *n* promjenljivih mora imati 2^n redova. Odatle slijedi da su tablice istine prihvatljiv način opisivanja samo logičkih funkcija koje zavise od manjeg broja promjenljivih.

Uređaji koji služe za izvođenje osnovnih logičkih operacija nazivaju se ***logička kola*** ili ***gejtovi***. Na primjer, logičko kolo za izvođenje operacije konjunkcije ima dva ulaza, koja možemo obilježiti recimo sa X i Y, i jedan izlaz. Na ulaze X i Y dovode se vrijednosti 0 ili 1 (kakva je tačna fizička priroda ovih vrijednosti, tj. šta one fizički predstavljaju, zavisi od konkretne fizičke realizacije logičkog kola), dok se na izlazu takvog logičkog kola javlja vrijednost koja je jednaka vrijednosti konjunkcije XY. Logička kola se obično prikazuju pomoću simbola sa sljedeće slike (prikazani su simboli koji se najviše koriste, mada se u drugoj literaturi mogu susresti i neznatno drugačiji simboli):

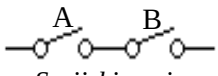


Logička kola koja izvode operacije AND, OR i NOT, nazivaju se ***osnovna logička kola***. Kolo koje izvodi NOT operaciju često se naziva i ***invertor***. Sva logička kola osim invertora mogu imati i više od dva ulaza, na primjer kao na sljedećoj slici, na kojoj su prikazana troulazna AND i OR kola:

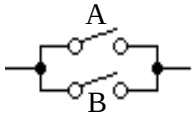


Osnovna logička kola su uglavnom uređaji prilično jednostavne građe. Teoretski, oni se mogu realizirati na raznim principima: mehaničkim, hidrauličkim, pneumatskim, elektromehaničkim, elektronskim, pa čak i biološkim. U prvim digitalnim računarima susretala su se svakakva rješenja, isprva čisto mehanička, a zatim elektromehanička (*Charles Babbage* je još prije 150 godina zamislio projekat čisto mehaničkog digitalnog računara na parni pogon, ali projekat nikada nije bio realiziran zbog nedostatka sredstava). Međutim, jedino elektronski princip rada omogućava dovoljnu brzinu rada i dovoljnu minijaturizaciju da bi se mogao napraviti računar velike brzine rada i malih dimenzija (dimenzije nikada realiziranog Babbageovog mehaničkog računara procijenjene su na veličinu fudbalskog igrališta). Naravno, u elektronskim logičkim kolima, logičke vrijednosti 0 i 1 su električne prirode, i obično predstavljaju dva različita naponska nivoa (npr. logička nula i jedinica mogu odgovarati naponskim signalima iznosa 0 V i +5 V respektivno).

Ulaženje u detaljne principe rada elektronskih logičkih kola zahtijevalo bi poznavanje elektronike, u šta ovdje ne možemo ulaziti. Stoga ćemo opisati samo osnovnu ideju. Posmatrajmo sljedeća dva spoja prekidača, serijski i paralelni spoj:



Serijski spoj



Paralelni spoj

U serijskom spoju struja može teći samo ukoliko su zatvorena oba prekidača A i B, dok u paralelnom spoju struja može teći ako je zatvoren makar jedan od prekidača A ili B. Ako zamislimo da nam zatvoren prekidač i proticanje struje simboliziraju cifru 1, a otvoren prekidač i neproticanje struje cifru 0, tada *serijski spoj* ostvaruje operaciju *konjunkcije*, a *paralelni spoj* operaciju *disjunkcije*. U elektromehaničkim izvedbama AND i OR logičkih kola zaista su se koristili prekidači, dok se u elektronskim izvedbama oni simuliraju raznim elektronskim napravama (diodama i tranzistorima).

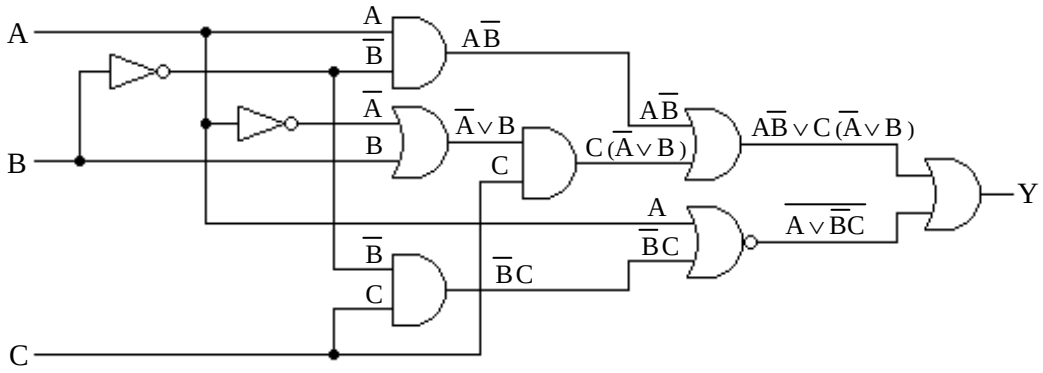
Kako ćemo kasnije vidjeti, svaka logička funkcija, bez obzira na složenost, može se izraziti formulom koja sadrži samo do sada uvedene funkcije. Odatle slijedi da se pomoću prikazanih logičkih kola može izgraditi sklop za realizaciju ma kakve logičke funkcije. Zapravo, kako se na ulaz ma kakvog digitalnog uređaja koji koristi binarni brojni sistem dovode ulazi koji isključivo mogu imati vrijednost 0 i 1 (uz pretpostavku da različite bite ulaznih podataka dovodimo na različite ulaze), kako njegovi izlazi imaju samo vrijednosti 0 i 1, i kako se svaka funkcija koja uzima vrijednosti samo 0 i 1, i u kojoj se javljaju promjenljive koje uzimaju samo vrijednosti 0 i 1 može izraziti preko logičkih operacija, slijedi da se rad ***svakog*** digitalnog uređaja može opisati isključivo korištenjem operacija i pravila logičke algebre. To je razlog zbog kojeg logička algebra ima ogroman značaj u računarskoj tehnici. Slijedi da ukoliko znamo kako da napravimo uređaje koji mogu računati operacije negacije, konjunkcije i disjunkcije, što je praktički elementarno, logička algebra nam, barem u načelu, može pomoći da napravimo bilo kakav digitalni uređaj (uključujući i digitalni računar)! Upravo to će biti i urađeno kroz tekst koji slijedi u narednim poglavljima.

Iz dosada izloženog slijedi da je povezivanjem osnovnih logičkih kola moguće realizirati svaku logičku funkciju, a samim tim i svaki digitalni uređaj, jer se rad ma kakvog digitalnog uređaja može opisati logičkim funkcijama. U suštini, svaki digitalni računar, posmatrano na najelementarnijem nivou, sastoji se samo od osnovnih logičkih kola, samo što se danas njihov broj mjeri u milionima, pa i u milijardama.

Primjer 6.3:

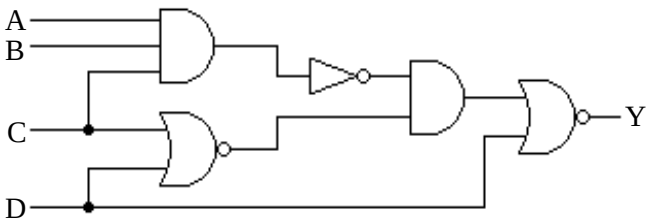
- Nacrtati sklop koji realizira logičku funkciju funkciju $Y = [A\overline{B} \vee C(\overline{A} \vee B)] \vee \overline{A \vee \overline{B}C}$.

Slično kao kod formiranja tablica istine, crtanje je najbolje izvršiti obradom prvo manjih članova, a zatim postupnim formiranjem sve većih i većih članova:

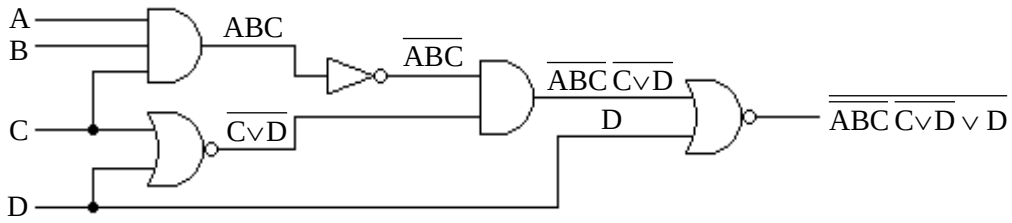


Primjer 6.4:

- Odrediti koju funkciju realizira shema na sljedećoj slici:



Logička funkcija se lako određuje ukoliko provedemo postupnu analizu od ulaza ka izlazu, uz bilježenje funkcija koje se dobijaju na izlazima iz pojedinih gejtova:



Odavde lako očitavamo da je $Y = \overline{\overline{ABC}\ \overline{C \vee D} \vee D}$, odnosno sklop realizira funkciju čiju smo tablicu istine formirali u Primjeru 6.2.

Kao i svaka oblast matematike, i logička algebra ima svoje zakone odnosno pravila. U nastavku ćemo navesti najvažnija pravila koja se odnose na tri osnovne logičke operacije (konjunkciju, disjunkciju i negaciju):

[1]	$X \cdot 0 = 0$	*Pravilo o konstanti* 0 *za konjunkciju (apsorpcija)*
[2]	$X \cdot 1 = X$	*Pravilo o konstanti* 1 *za konjunkciju (neutralni element)*
[3]	$X \vee 0 = X$	*Pravilo o konstanti* 0 *za disjunkciju (neutralni element)*
[4]	$X \vee 1 = 1$	*Pravilo o konstanti* 1 *za disjunkciju (apsorpcija)*
[5]	$X\,\overline{X} = 0$	*Zakon konzistentnosti (neprotivurječnosti)*
[6]	$X \vee \overline{X} = 1$	*Zakon isključenja trećeg*
[7]	$X\,Y = Y\,X$	*Zakon komutativnosti za konjunkciju*
[8]	$X \vee Y = Y \vee X$	*Zakon komutativnosti za disjunkciju*
[9]	$X\,(Y\,Z) = (X\,Y)\,Z$	*Zakon asocijativnosti za konjunkciju*
[10]	$X \vee (Y \vee Z) = (X \vee Y) \vee Z$	*Zakon asocijativnosti za disjukciju*
[11]	$X\,(Y \vee Z) = X\,Y \vee X\,Z$	*Zakon distributivnosti disjunkcije u odnosu na konjunkciju*
[12]	$X \vee Y\,Z = (X \vee Y)\,(X \vee Z)$	*Zakon distributivnosti konjunkcije u odnosu na disjukciju*
[13]	$X \cdot X \cdot ... \cdot X = X$	*Zakon idempotentnosti za konjunkciju*
[14]	$X \vee X \vee ... \vee X = X$	*Zakon idempotentnosti za disjunkciju*
[15]	$\overline{\overline{X}} = X$	*Pravilo dvostruke negacije*
[16]	$X\,Y \vee X\,\overline{Y} = X$	*Pravilo potpunog sažimanja*
[17]	$X\,(X \vee Y) = X$	*Zakon apsorpcije za konjunkciju*
[18]	$X \vee X\,Y = X$	*Zakon apsorpcije za disjukciju*
[19]	$\overline{X\,Y} = \overline{X} \vee \overline{Y}$	*Prva De Morganova teorema*
[20]	$\overline{X \vee Y} = \overline{X}\ \overline{Y}$	*Druga De Morganova teorema*
[21]	$X \vee \overline{X}\,Y = X \vee Y$	*Pravilo neutraliziranja negacije*

Valjanost svih navedenih pravila moguće je lako dokazati uz pomoć tablica istine, što čitatelj odnosno čitateljica mogu uraditi kao vježbu. Međutim, mnoga od navedenih pravila mogu se dokazati čisto algebarski, svođenjem na jednostavnija pravila. Tako je, na primjer, veoma važno pravilo potpunog sažimanja [16] moguće jednostavno dokazati pozivajući se na jednostavnija pravila [2], [6] i [11] (radi lakšeg snalaženja, u zagradama iznad znaka jednakosti prikazani su brojevi primijenjenih pravila u odgovarajućem koraku):

$$XY \vee X\overline{Y} \overset{(11)}{=} X(Y \vee \overline{Y}) \overset{(6)}{=} X \cdot 1 \overset{(2)}{=} X$$

Slično, pravilo apsorpcije [18] može se dokazati pozivajući se na pravila [2], [4], [8] i [11]:

$$X \vee X\,Y \overset{(2)}{=} X \cdot 1 \vee X\,Y \overset{(11)}{=} X\,(1 \vee Y) \overset{(8)}{=} X\,(Y \vee 1) \overset{(4)}{=} X \cdot 1 \overset{(2)}{=} X$$

Interesantno je primijetiti da se i same tablice istine osnovnih logičkih funkcija mogu dokazati pozivajući se na navedena pravila. Pokažimo, na primjer, da tablica istine za disjunkciju mora biti onakva kakvu smo je prikazali na početku. Po pravilu [3] vrijedi $X \vee 0 = X$, na osnovu čega je $0 \vee 0 = 0$ i $1 \vee 0 = 1$, dok po pravilu [4] vrijedi $X \vee 1 = 1$, na osnovu čega je $0 \vee 1 = 1$ i $1 \vee 1 = 1$. Na ovaj način smo "dokazali" valjanost tablice istine za disjunkciju, koristeći samo pravila logičke algebre. Slično je moguće dokazati valjanost tablica istine za konjunkciju i negaciju.

Mogućnost da se valjanost raznih pravila logičke algebre pa čak i valjanost samih tablica istine može dokazati pozivajući se na neka druga pravila, daje povoda da se određeni broj pravila proglasi elementarnim činjenicama logičke algebre, koje se prihvataju bez dokaza, i na kojima se dalje zasnivaju sva druga pravila i čitava logička algebra. Ova pravila nazivaju se ***aksiomi logičke algebre***. Aksiomi logičke algebre treba da budu što je god moguće jednostavniji i da ne zavise jedan od drugih (tj. niti jedan aksiom ne bi smio da bude dokaziv preko ostalih). Izbor pravila koja će biti uzeta za aksiome nije jednoznačan, tako da razni autori uzimaju različitu skupinu pravila za aksiome logičke algebre. Ipak, najčešći izbor je da se za aksiome logičke (preciznije, Booleove) algebre uzmu pravila [1] – [12]. Zaista, ova pravila su posve elementarna i međusobno neovisna, i omogućavaju dokazivanje svih ostalih pravila pomoću njih. Na primjer, već smo dokazali pravila [16] i [18] pozivajući se samo na navedene aksiome. Pravila [13], [14], [17] i [21] također je sasvim lako dokazati polazeći od aksioma logičke algebre. Nešto je teže dokazati pravilo dvostruke negacije i De Morganove teoreme (neki autori i ova pravila svrstavaju u aksiome, mada su oni dokazivi preko aksioma [1] – [12]). Na primjer, pravilo dvostruke negacije [15] moguće je dokazati preko aksioma [2], [5], [6], [7] i [11] recimo na sljedeći način:

$$\overline{\overline{X}} \overset{(2)}{=} \overline{\overline{X}} \cdot 1 \overset{(6)}{=} \overline{\overline{X}}(X \vee \overline{X}) \overset{(11)}{=} \overline{\overline{X}} X \vee \overline{\overline{X}}\,\overline{X} \overset{(7)}{=} \overline{\overline{X}} X \vee \overline{X}\,\overline{\overline{X}} \overset{(5)}{=} \overline{\overline{X}} X \vee 0 \overset{(5)}{=} \overline{\overline{X}} X \vee X\overline{X} \overset{(7)}{=}$$
$$= X\overline{\overline{X}} \vee X\overline{X} \overset{(11)}{=} X(\overline{\overline{X}} \vee \overline{X}) \overset{(7)}{=} X(\overline{X} \vee \overline{\overline{X}}) \overset{(6)}{=} X \cdot 1 \overset{(2)}{=} X$$

Dokaz De Morganovih teorema je prilično glomazan, ukoliko se oslonimo samo na neposrednu primjenu aksioma Booleove algebre. Ovdje ćemo kao ilustraciju navesti dokaz prve De Morganove teoreme (pravila [19]), koji je interesantan jer se u njemu višestruko koristi većina aksioma Booleove algebre:

$$\overline{XY} \overset{(2)}{=} \overline{XY} \cdot 1 \overset{(4)}{=} \overline{XY}(\overline{X} \vee 1) \overset{(6)}{=} \overline{XY}(\overline{X} \vee Y \vee \overline{Y}) \overset{(8)}{=} \overline{XY}(Y \vee \overline{X} \vee \overline{Y}) \overset{(2)}{=}$$
$$= \overline{XY}(Y \cdot 1 \vee \overline{X} \vee \overline{Y}) \overset{(6)}{=} \overline{XY}[Y(X \vee \overline{X}) \vee \overline{X} \vee \overline{Y}] \overset{(11)}{=} \overline{XY}(YX \vee Y\overline{X} \vee \overline{X} \vee \overline{Y}) \overset{(7)}{=}$$
$$= \overline{XY}(XY \vee \overline{X}Y \vee \overline{X} \vee \overline{Y}) \overset{(2)}{=} \overline{XY}(XY \vee \overline{X}Y \vee \overline{X} \cdot 1 \vee \overline{Y}) \overset{(11)}{=}$$
$$= \overline{XY}[XY \vee \overline{X}(Y \vee 1) \vee \overline{Y}] \overset{(4)}{=} \overline{XY}(XY \vee \overline{X} \cdot 1 \vee \overline{Y}) \overset{(2)}{=} \overline{XY}(XY \vee \overline{X} \vee \overline{Y}) \overset{(11)}{=}$$
$$= \overline{XY}\,XY \vee \overline{XY}(\overline{X} \vee \overline{Y}) \overset{(7)}{=} XY\,\overline{XY} \vee (\overline{X} \vee \overline{Y})\,\overline{XY} \overset{(5)}{=} 0 \vee (\overline{X} \vee \overline{Y})\,\overline{XY} \overset{(1)}{=}$$
$$= X \cdot 0 \vee (\overline{X} \vee \overline{Y})\,\overline{XY} \overset{(5)}{=} XY\overline{Y} \vee (\overline{X} \vee \overline{Y})\,\overline{XY} \overset{(3)}{=} XY\overline{Y} \vee (\overline{X} \vee \overline{Y})\,\overline{XY} \vee 0 \overset{(8)}{=}$$
$$= 0 \vee XY\overline{Y} \vee (\overline{X} \vee \overline{Y})\,\overline{XY} \overset{(1)}{=} Y \cdot 0 \vee XY\overline{Y} \vee (\overline{X} \vee \overline{Y})\,\overline{XY} \overset{(5)}{=}$$
$$= YX\overline{X} \vee XY\overline{Y} \vee (\overline{X} \vee \overline{Y})\,\overline{XY} \overset{(7)}{=} XY\overline{X} \vee XY\overline{Y} \vee (\overline{X} \vee \overline{Y})\,\overline{XY} \overset{(11)}{=}$$
$$= XY(\overline{X} \vee \overline{Y}) \vee (\overline{X} \vee \overline{Y})\,\overline{XY} \overset{(7)}{=} (\overline{X} \vee \overline{Y})\,XY \vee (\overline{X} \vee \overline{Y})\,\overline{XY} \overset{(11)}{=}$$
$$= (\overline{X} \vee \overline{Y})(XY \vee \overline{XY}) \overset{(6)}{=} (\overline{X} \vee \overline{Y}) \cdot 1 \overset{(2)}{=} \overline{X} \vee \overline{Y}$$

Dokaz De Morganovih teorema je moguće izvesti mnogo kraće, ukoliko se prethodno dokažu neki pomoćni rezultati (za primjer, pogledati zadatke na kraju ovog poglavlja). Međutim, ovaj dokaz svakako ne treba pamtiti – on je prikazan čisto kao ilustracija primjene pravila logičke algebre na transformacije logičkih izraza iz jednog oblika u drugi.

Za dva logička izraza kažemo da su ***ekvivalentni*** ukoliko uzimaju iste vrijednosti za ma kako izabrane vrijednosti promjenljivih koje se u njima javljaju. Dva ekvivalentna logička izraza uvijek se mogu svesti jedan na drugi primjenom konačno mnogo transformacija izraženih pravilima logičke algebre. Naročito su značajna pravila 19. i 20. (***De Morganove teoreme***), ne samo u logičkoj algebri, jer omogućavaju efikasnu negaciju složenih izraza koji sadrže konjunkcije i disjunkcije. Bitno je napomenuti da se logički izrazi često mogu svesti na znatno jednostavniji oblik u odnosu na njihov polazni oblik, kao što se može vidjeti iz primjera koji slijede.

Primjer 6.5:

- Pojednostaviti funkciju $Y = \left[A\overline{B} \vee \overline{C(\overline{A} \vee B)}\right] \vee C\overline{A \vee \overline{B}C}$ primjenom zakona logičke algebre.

U prikazanom postupku, koristi se veliki broj pravila logičke algebre. U ovom primjeru su ponovo, radi lakšeg snalaženja, u zagradama iznad znaka jednakosti prikazani brojevi pravila koja su primijenjena u svakom od koraka transformacije:

$$
\begin{aligned}
Y &= \left[A\overline{B} \vee \overline{C(\overline{A} \vee B)}\right] \vee C\overline{A \vee \overline{B}C} \overset{(10)}{=} A\overline{B} \vee \overline{C(\overline{A} \vee B)} \vee C\overline{A \vee \overline{B}C} \overset{(19)}{=} \\
&= A\overline{B} \vee \overline{C} \vee \overline{\overline{A} \vee B} \vee C\overline{A \vee \overline{B}C} \overset{(20)}{=} A\overline{B} \vee \overline{C} \vee \overline{\overline{A}}\,\overline{B} \vee C\overline{A}\,\overline{\overline{B}C} \overset{(20)}{=} A\overline{B} \vee \overline{C} \vee \overline{\overline{A}}\,\overline{B} \vee C\overline{A}\,\overline{\overline{B}C} \overset{(15)}{=} \\
&= A\overline{B} \vee \overline{C} \vee A\overline{B} \vee C\overline{A}\,\overline{\overline{B}C} \overset{(8)}{=} A\overline{B} \vee A\overline{B} \vee \overline{C} \vee C\overline{A}\,\overline{\overline{B}C} \overset{(14)}{=} A\overline{B} \vee \overline{C} \vee C\overline{A}\,\overline{\overline{B}C} \overset{(19)}{=} \\
&= A\overline{B} \vee \overline{C} \vee C\overline{A}\,(\overline{\overline{B}} \vee \overline{C}) \overset{(15)}{=} A\overline{B} \vee \overline{C} \vee C\overline{A}\,(B \vee \overline{C}) \overset{(11)}{=} A\overline{B} \vee \overline{C} \vee C\overline{A}B \vee C\overline{A}\overline{C} \overset{(21)}{=} \\
&= A\overline{B} \vee \overline{C} \vee \overline{A}B \vee C\overline{A}\overline{C} \overset{(7)}{=} A\overline{B} \vee \overline{C} \vee \overline{A}B \vee \overline{A}C\overline{C} \overset{(5)}{=} A\overline{B} \vee \overline{C} \vee \overline{A}B \vee \overline{A} \cdot 0 \overset{(1)}{=} \\
&= A\overline{B} \vee \overline{C} \vee \overline{A}B \vee 0 \overset{(3)}{=} A\overline{B} \vee \overline{C} \vee \overline{A}B \overset{(8)}{=} A\overline{B} \vee \overline{A}B \vee \overline{C}
\end{aligned}
$$

Vidimo da je dobijeni izraz znatno jednostavniji od polaznog izraza. Kasnije ćemo vidjeti da se, ukoliko dozvolimo i upotrebu operacije ekskluzivne disjunkcije, dobijena funkcija može još više pojednostaviti i prikazati u obliku $Y = A \oplus B \vee \overline{C}$.

Pronalaženje najjednostavnije logičke funkcije koja je ekvivalentna polaznoj logičkoj funkciji naziva se ***minimizacija logičke funkcije***. Značaj minimizacije logičkih funkcija sastoji se u tome da će jednostavnijoj funkciji odgovarati jednostavniji uređaj.

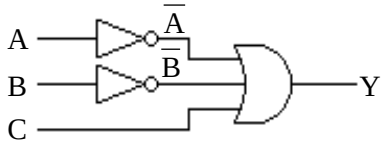
Primjer 6.6:

- Naći što jednostavniji sklop koji obavlja istu funkcionalnost kao sklop iz Primjera 6.3.

Da bismo izvršili traženo pojednostavljenje, transformiraćemo funkciju koja opisuje rad sklopa primjenom pravila logičke algebre, kao što slijedi (brojeve primijenjenih pravila nećemo više navoditi):

$$
\begin{aligned}
Y &= [A\overline{B} \vee C(\overline{A} \vee B)] \vee \overline{A \vee \overline{B}C} = (A\overline{B} \vee C\overline{A} \vee CB) \vee \overline{A}\,\overline{\overline{B}C} = (A\overline{B} \vee C\overline{A} \vee CB) \vee \overline{A}\,(B \vee \overline{C}) = \\
&= A\overline{B} \vee \overline{A}C \vee BC \vee \overline{A}B \vee \overline{A}\,\overline{C} = A\overline{B} \vee (\overline{A}C \vee \overline{A}\,\overline{C} \vee \overline{A}B) \vee BC = \\
&= A\overline{B} \vee \overline{A}(C \vee \overline{C} \vee B) \vee BC = A\overline{B} \vee \overline{A}(1 \vee B) \vee BC = A\overline{B} \vee \overline{A} \cdot 1 \vee BC = \\
&= A\overline{B} \vee \overline{A} \vee BC = \overline{A} \vee A\overline{B} \vee BC = \overline{A} \vee \overline{B} \vee BC = \overline{A} \vee \overline{B} \vee C
\end{aligned}
$$

Na osnovu dobijene funkcije (koja se očito ne može dalje skratiti), možemo nacrtati sljedeći sklop:



Kao što vidimo, dobijeni sklop je mnogo jednostavniji od sklopa koji je dobijen iz izvornog oblika funkcije. Stoga je za potrebe pojednostavljenja realizacije digitalnih uređaja uvijek potrebno izvršiti transformaciju funkcije u što je god moguće jednostavniji oblik.

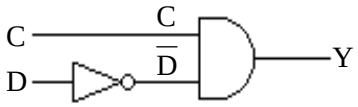
Primjer 6.7:

- Pojednostaviti i realizirati funkciju $Y = \overline{\overline{ABC}\,\overline{C \vee D} \vee D}$.

Primjenom zakona logičke algebre dobijamo (u pojedinim koracima je primijenjeno više pravila odjednom):

$$Y = \overline{\overline{ABC}\,\overline{C \vee D} \vee D} = \overline{\overline{ABC}\,\overline{C \vee D}}\,\overline{D} = (\overline{\overline{ABC}} \vee \overline{\overline{C \vee D}})\,\overline{D} = (ABC \vee C \vee D)\,\overline{D} =$$
$$= [C(AB \vee 1) \vee D]\,\overline{D} = (C \vee D)\,\overline{D} = C\overline{D}$$

Odavde slijedi traženi sklop (u kojem se, uzgred rečeno, uopće ne pojavljuju promjenljive A i B koje su bile prisutne u polaznoj funkciji):



Sljedeća dva primjera ilustriraju da primjena pravila koja na prvi pogled djeluju najočiglednija nije uvijek pravi put ka minimizaciji, nego da su često potrebne dosjetke koje nisu nimalo očigledne.

Primjer 6.8:

- Pojednostaviti logičku funkciju $Y = \overline{A}B \vee \overline{B}C \vee \overline{A}C$.

Rješenja koja se prirodno nameću su "izvlačenje" zajedničkog faktora $\overline{A}$ iz prvog i trećeg člana ili zajedničkog faktora C iz drugog i trećeg člana, čime možemo dobiti izraze $\overline{A}(B \vee C) \vee \overline{B}C$ odnosno $\overline{A}B \vee (\overline{A} \vee \overline{B})C$, koji nisu gotovo ništa jednostavniji od polaznog izraza, a ne mogu se dalje transformirati ni na kakav očigledan način. Međutim, pravi put ka minimizaciji ove funkcije mnogo je manje očigledan, kako slijedi iz prikazanog niza transformacija:

$$Y = \overline{A}B \vee \overline{B}C \vee \overline{A}C = \overline{A}B \vee \overline{B}C \vee \overline{A}C(B \vee \overline{B}) = \overline{A}B \vee \overline{B}C \vee \overline{A}CB \vee \overline{A}C\overline{B} =$$
$$= \overline{A}(B \vee BC) \vee \overline{B}(C \vee C\overline{A}) = \overline{A}B \vee \overline{B}C$$

Primjer 6.9:

- Pojednostaviti logičku funkciju $Y = ABC \vee (\overline{A} \vee B \vee \overline{C})D$.

Ovaj primjer je još interesantniji, s obzirom da je potreban priličan broj ne baš očiglednih transformacija prije nego što se pokaže da je promjenljiva B unutar zagrade zapravo suvišna:

$$Y = ABC \vee (\overline{A} \vee B \vee \overline{C})D = ABC \vee \overline{A}D \vee BD \vee \overline{C}D = ABC \vee \overline{A}D \vee BD(C \vee \overline{C}) \vee \overline{C}D =$$
$$= ABC \vee \overline{A}D \vee BCD \vee (B\overline{C}D \vee \overline{C}D) = ABC \vee \overline{A}D \vee BCD \vee \overline{C}D =$$
$$= ABC \vee \overline{A}D \vee BCD(A \vee \overline{A}) \vee \overline{C}D = ABC \vee \overline{A}D \vee ABCD \vee \overline{A}BCD \vee \overline{C}D =$$
$$= (ABC \vee ABCD) \vee (\overline{A}D \vee \overline{A}BCD) \vee \overline{C}D = ABC \vee \overline{A}D \vee \overline{C}D = ABC \vee (\overline{A} \vee \overline{C})D$$

Izloženi primjeri su već sasvim dovoljni da se zaključi da minimizacija logičkih funkcija korištenjem samo zakona logičke algebre često nije nimalo jednostavan postupak, s obzirom da ne postoje nikakve jasne preporuke koje bi ukazivale kada treba upotrijebiti koje pravilo. Uskoro ćemo vidjeti da za istu svrhu postoje mnogo praktičniji i sistematičniji postupci.

(?) Pitanja i zadaci

6.1 Objasnite šta je logička odnosno prekidačka algebra, a šta Booleova algebra.

6.2 Navedite osnovne operacije logičke algebre, navedite tablice koje opisuju dejstvo ovih operacija i objasnite simboliku koja se koristi za prikazivanje ovih operacija.

6.3 Prikažite logičku funkciju $Y = (A \vee \overline{C})\,\overline{C \vee BE} \vee (CD \vee \overline{A}E)\,\overline{B \vee C}$ korištenjem standardnih simbola matematičke logike "$\wedge$", "$\vee$" i "$\neg$".

6.4 Sastavite tabelu istine za logičku funkciju $Y = A\overline{B}\,\overline{C} \vee \overline{A \vee B \vee C} \vee \overline{\overline{A}\,\overline{B}C}$.

6.5 Objasnite šta su logička kola (gejtovi) i prikažite simbole osnovnih logičkih kola.

6.6 Objasnite osnovne principe na kojima se zasniva rad AND i OR logičkih kola.

6.7 Nacrtajte sklop koji realizira logičku funkciju $Y = A\overline{C}\,\overline{D} \vee \overline{A \vee B \vee D} \vee \overline{\overline{A}\,\overline{B}CD}$.

6.8 Provjerite De Morganove teoreme formiranjem tabela istine.

6.9 Dopustimo da se pored logičkih simbola 0 i 1 koriste i još dva "logička simbola" koja ćemo označiti sa α i β, i definirajmo logičke operacije negacije, konjunkcije i disjunkcije pomoću sljedećih tablica:

A	$\overline{A}$
0	1
α	β
β	α
1	0

A \ B	0	α	β	1
0	0	0	0	0
α	0	α	0	α
β	0	0	β	β
1	0	α	β	1

A \ B	0	α	β	1
0	0	α	β	1
α	α	α	1	1
β	β	1	β	1
1	1	1	1	1

Provjerite da ovako definirane operacije "negacije", "konjunkcije" i "disjunkcije" na skupu vrijednosti $\{0, \alpha, \beta, 1\}$ zadovoljavaju sve aksiome Booleove algebre (ovo je primjer Booleove algebre nad skupom od četiri vrijednosti). Napomena: postoje razne mogućnosti kako se može interpretirati ovakva Booleova algebra i značenje njenih vrijednosti označenih sa α i β.

6.10* Neka je dat proizvoljan skup S. Posmatrajmo partitivni skup $\mathcal{P}(S)$, odnosno skup svih podskupova skupa S. Provjerite da elementi skupa $\mathcal{P}(S)$, nad kojima su negacija, konjunkcija i disjunkcija definirani respektivno kao komplement u odnosu na skup S, presjek i unija, zadovoljavaju sve aksiome Booleove algebre, ukoliko prazan skup $\varnothing$ posmatramo kao "nulu", a skup S kao "jedinicu" (ovo pokazuje da je algebra skupova zapravo također Booleova algebra).

6.11 Dokažite pravila [13], [14], [17] i [21] logičke algebre korištenjem samo aksioma logičke algebre, odnosno pravila [1] – [12].

6.12* Dokažite drugu De Morganovu teoremu (pravilo [20]) koristeći samo aksiome logičke algebre.

6.13* Koristeći isključivo aksiome logičke algebre, dokažite sljedeću tvrdnju: Ukoliko vrijedi $P\,Q = 0$ i $P \vee Q = 1$, tada mora vrijediti $Q = \overline{P}$.

6.14* Dokažite pravilo dvostruke negacije [15] koristeći pomoćnu tvrdnju dokazanu u zadatku 6.13.

6.15* Dokažite obje De Morganove teoreme koristeći pomoćnu tvrdnju dokazanu u zadatku 6.13.

6.16 Koliko ima međusobno neekvivalentnih logičkih funkcija od *n* promjenljivih? Obrazložite odgovor. Posebno utvrdite koliko ima međusobno neekvivalentnih logičkih funkcija od 2, 3, 4, 5 i 6 promjenljivih.

6.17 Primjenom pravila logičke algebre, pojednostavite logičku funkciju $Y = AB \vee AC \vee \overline{B}C$ koliko god je to moguće. Provjerite dobijeni rezultat formiranjem tabela istine za polaznu funkciju i dobijenu funkciju.

6.18 Pojednostavite koliko je god moguće sljedeće logičke funkcije:

a) $Y = (A \vee B)(\overline{A} \vee B \vee C)(\overline{A} \vee B \vee \overline{C})$ b) $Y = A \vee ABC \vee \overline{A}BC \vee \overline{A}B \vee AD \vee \overline{A}\,\overline{D}$

6.19* Minimizirajte logičku funkciju $Y = \overline{A}(\overline{B} \vee C) \vee BC \vee A\overline{C}$ korištenjem pravila logičke algebre.

6.20 Minimizirajte logičku funkciju $Y = (A \vee \overline{C})\,\overline{C \vee BE} \vee (CD \vee \overline{A}E)\,\overline{B \vee C}$.

6.21 Minimizirajte logičku funkciju $Y = ABC \vee \overline{A}BC \vee A\overline{B}C \vee AB\overline{C}$.

6.22 Minimizirajte sljedeće logičke funkcije:

a) $Y = \overline{\overline{ABC \vee D} \vee \overline{ACD} \vee B}$ b) $Y = \overline{\overline{A\overline{B}C \vee \overline{D}} \vee \overline{\overline{ACD} \vee B}}$

c) $Y = \overline{\overline{A\overline{B}\,\overline{C} \vee D} \vee \overline{\overline{ACD} \vee \overline{B}}}$ d) $Y = \overline{\overline{A\overline{B}} \vee \overline{CD} \vee \overline{A\overline{C}} \vee \overline{B}D}$

6.23 Primjenom De Morganovih teorema, nađite negacije sljedećih logičkih funkcija, u što je god moguće jednostavnijoj formi. Šta možete zaključiti posmatrajući dobijene rezultate?

a) $Y = \overline{A} \vee \overline{B}$ b) $Y = (A \vee B)C$ c) $Y = A \vee B\overline{C}$

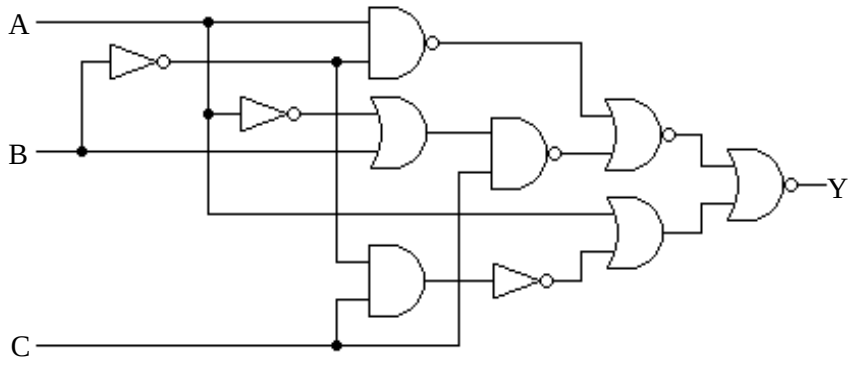
d) $Y = A \vee \overline{B}\,\overline{C}$ e) $Y = A(B \vee CD)$ f) $Y = ABC \vee B(\overline{C} \vee \overline{D})$

6.24* Nađite negacije sljedećih logičkih funkcija, u što je god moguće jednostavnijoj formi:

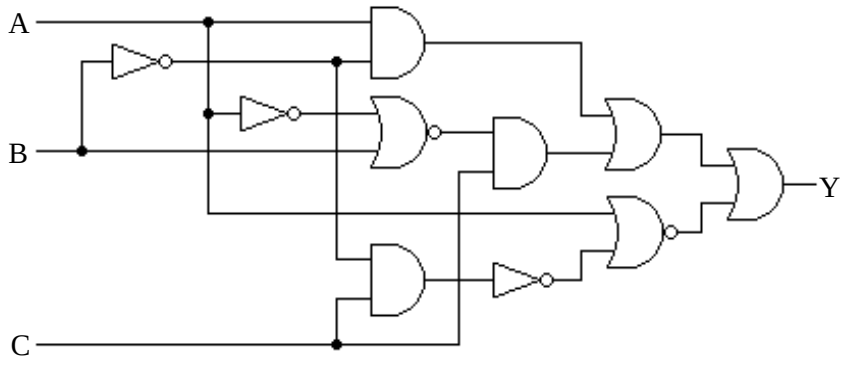
a) $Y = (A \vee \overline{B}\,\overline{C}\,\overline{D})[\overline{A}\,\overline{D} \vee B(\overline{C} \vee A)]$ b) $Y = \overline{A}BC \vee (\overline{A} \vee B \vee D)(AB\overline{D} \vee \overline{B})$

6.25 Nađite negaciju logičke funkcije iz zadatka 6.20 (u izvornom obliku) i svedite je na što je god moguću formu. Nakon toga, nađite negaciju minimizirane verzije iste funkcije (iskoristite rješenje zadatka 6.20), i pokažite da se dobija isti rezultat.

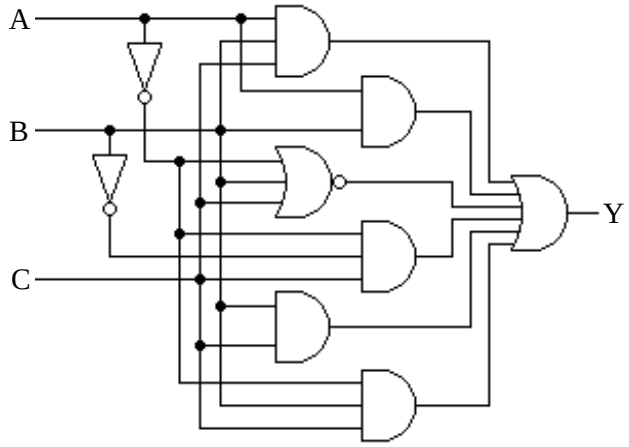
6.26 Odredite koju logičku funkciju obavlja sklop na sljedećoj slici. Zatim minimizirajte dobijenu logičku funkciju i nacrtajte odgovarajući sklop (koji realizira istu funkcionalnost kao i polazni).



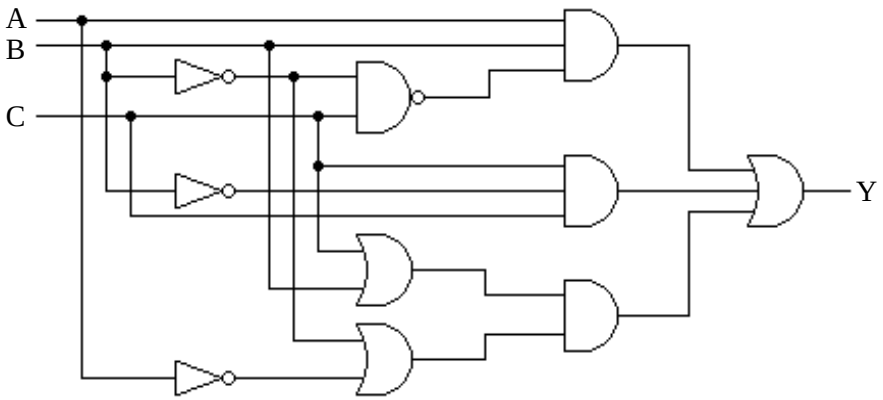
6.27 Odredite koju logičku funkciju obavlja sklop na sljedećoj slici, a zatim nađite što je god moguće jednostavniji sklop koji obavlja istu funkcionalnost.



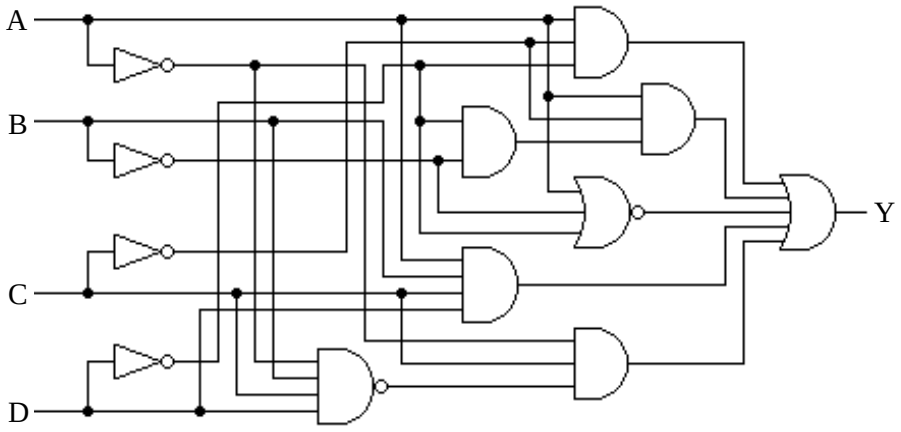
6.28* Odredite koju logičku funkciju obavlja sklop na sljedećoj slici, a zatim nađite što je god moguće jednostavniji sklop koji obavlja istu funkcionalnost.



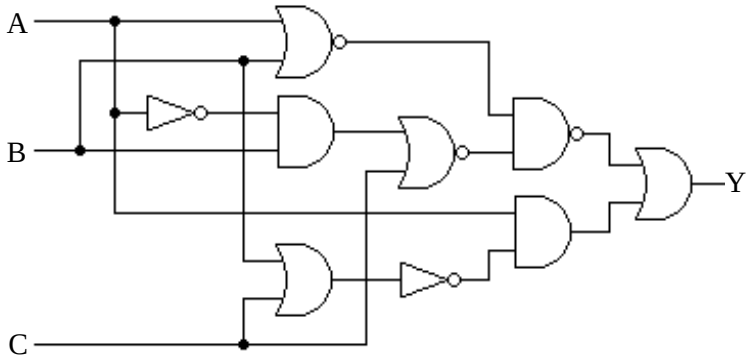
6.29 Odredite koju logičku funkciju obavlja sklop na sljedećoj slici, a zatim nađite što je god moguće jednostavniji sklop koji obavlja istu funkcionalnost.



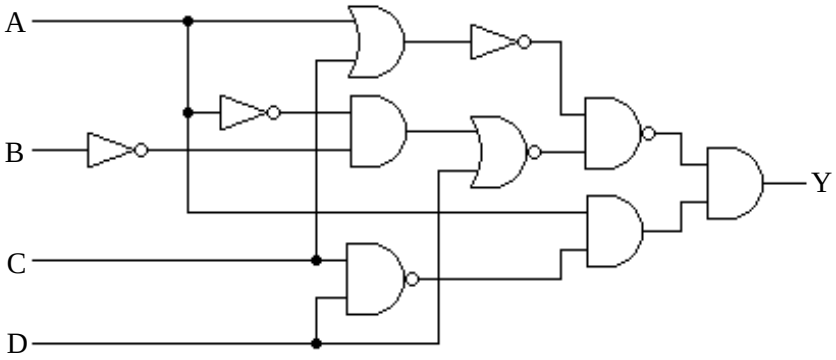
6.30** Odredite koju logičku funkciju obavlja sklop na sljedećoj slici, a zatim nađite što je god moguće jednostavniji sklop koji obavlja istu funkcionalnost.



6.31 Odredite koju logičku funkciju obavlja sklop na sljedećoj slici, a zatim nađite što je god moguće jednostavniji sklop koji obavlja istu funkcionalnost.



6.32 Odredite koju logičku funkciju obavlja sklop na sljedećoj slici, a zatim nađite što je god moguće jednostavniji sklop koji obavlja istu funkcionalnost.



7. Standardni oblici logičkih funkcija

Za potrebe prakse veoma je značajan problem da se na osnovu poznate tablice istine neke logičke funkcije odredi kako glasi njen analitički izraz (po mogućnosti u što jednostavnijoj formi). U nastavku ćemo pokazati da se svaka logička funkcija, bez obzira na njenu složenost i broj promjenljivih, uvijek može izraziti pomoću operacija *konjunkcije*, *disjunkcije* i *negacije*, tj. pomoću operacija *Booleove algebre*. Za tu svrhu, neophodno je prvo definirati neke pomoćne pojmove.

Definicija 7.1:

Logička funkcija koja se sastoji samo od *konjunkcija članova*, pri čemu je svaki član ili *promjenljiva* ili *njena negacija*, i u kojem se niti jedna promjenljiva *ne javlja više od jedanput*, zove se ***elementarna konjunkcija***. Na primjer, funkcija

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

predstavlja elementarnu konjunkciju. Sama promjenljiva ili njena negacija naziva se ***literal***.

Definicija 7.2:

Logička funkcija koja se sastoji samo od *disjunkcija članova*, pri čemu je svaki član ili *promjenljiva* ili *njena negacija*, i u kojem se niti jedna promjenljiva *ne javlja više od jedanput*, zove se ***elementarna disjunkcija***. Na primjer, funkcija

$$A \vee \overline{B} \vee \overline{D}$$

predstavlja elementarnu disjunkciju.

Literal (tj. usamljena promjenljiva ili njena negacija) se po potrebi može smatrati da predstavlja bilo elementarnu konjunkciju bilo elementarnu disjunkciju.

Definicija 7.3:

Za logičku funkciju koja ima oblik *disjunkcije članova* od kojih je svaki član *elementarna konjunkcija*, kažemo da je data u ***disjunktivnoj normalnoj formi*** (***DNF***). Na primjer, funkcija

$$A\overline{B} \vee \overline{A} \vee B\overline{C}\,\overline{D}$$

data je u disjunktivnoj normalnoj formi. Za *samu elementarnu konjunkciju* također smatramo da predstavlja DNF.

Definicija 7.4:

Disjunktivna normalna forma u kojoj svaka elementarna konjunkcija sadrži *sve promjenljive koje se javljaju u funkciji*, i u kojoj se *ista elementarna konjunkcija ne ponavlja više puta*, naziva se ***savršena disjuntivna normalna forma*** (***SDNF***). Na primjer, funkcija

$$A\overline{B}\,\overline{C} \vee \overline{A}BC \vee ABC$$

je predstavljena u savršenoj disjunktivnoj normalnoj formi. Elementarne konjunkcije od kojih se sastoji SDNF nazivaju se ***minterme*** (ili ***konstituente jedinice***).

Definicija 7.5:

Za logičku funkciju koja ima oblik *konjunkcije članova* od kojih je svaki član *elementarna disjunkcija*, kažemo da je data u ***konjunktivnoj normalnoj formi*** (***KNF***). Na primjer, funkcija

$$A\,(B \vee \overline{C})\,(\overline{A} \vee \overline{B} \vee C)$$

je data u konjunktivnoj normalnoj formi. Za *samu elementarnu disjunkciju* također smatramo da predstavlja KNF.

Definicija 7.6:

Za konjunktivnu normalnu formu u kojoj svaka elementarna disjunkcija sadrži *sve promjenljive koje se javljaju u funkciji*, i u kojem se *ista elementarna disjunkcija ne ponavlja više puta*, kažemo da je ***savršena konjunktivna normalna forma*** (***SKNF***). Na primjer, funkcija

$$(A \vee \overline{B} \vee \overline{C})\,(\overline{A} \vee B \vee C)\,(A \vee B \vee C)$$

je predstavljena u savršenoj konjunktivnoj normalnoj formi. Elementarne disjunkcije od kojih se sastoji SKNF nazivaju se ***maksterme*** ili ***klauze*** (ili ***konstituente nule***).

Pored opisanih disjunktivnih i konjunktivnih normalnih formi, postoje i druge disjunktivne i konjunktivne forme koje nisu normalne forme, jer odgovarajući članovi *nisu elementarne konjunkcije odnosno elementarne disjunkcije*. Tako je na primjer, funkcija

$$A \vee \overline{B}\,\overline{C} \vee \overline{ABC}$$

data u disjunktivnoj formi koja *nije normalna forma*, jer član $\overline{ABC}$ nije elementarna konjunkcija.

Za normalne forme uvodi se pojam *dužine*, pomoću sljedeće definicije:

Definicija 7.7:

Pod ***dužinom*** neke disjunktivne ili konjunktivne normalne forme podrazumijevamo ukupan broj literala koji se javljaju u njoj, tj. ukupan broj pojavljivanja svake od promjenljivih koja se javlja u funkciji. Na primjer, dužine formi iz definicija od 7.1 do 7.6 su redom 3, 3, 6, 9, 6 i 9.

Značaj disjunktivnih i konjunktivnih normalnih formi leži u činjenici da se svaka logička funkcija, bez obzira na njenu složenost, može prikazati kao DNF ili kao KNF (kao što ćemo uskoro pokazati). Dalje, ista logička funkcija može se na različite načine prikazati kao DNF ili KNF. Zbog toga imaju smisla sljedeće definicije:

Definicija 7.8:

Za logičku funkciju kažemo da je data u ***minimalnoj disjunktivnoj normalnoj formi*** (***MDNF***) ukoliko je ona data u *disjunktivnoj normalnoj formi* i ukoliko ne postoji *kraća* disjunktivna normalna forma koja joj je ekvivalentna (tj. koja uzima iste vrijednosti za ista stanja promjenljivih). Na primjer, funkcija

$$ABC \vee AB\overline{C} \vee D$$

je data u DNF, ali nije u MDNF, jer je ona ekvivalentna funkciji $AB \vee D$ (koja je od nje kraća za 4 literala). Analogno se definira i ***minimalna konjunktivna normalna forma*** (***MKNF***).

Mada ista logička funkcija može imati više različitih disjunktivnih ili konjunktivnih normalnih formi, savršene disjunktivne i konjunktivne normalne forme su jedinstvene do na poredak pojedinih članova.

Da bismo logičku funkciju predstavljenu formulom u kojoj se javljaju samo konjunkcije, disjunkcije i negacije sveli na *disjunktivnu normalnu formu*, potrebno je *osloboditi se svih negacija složenih članova* (uključujući i višestruke negacije) *pomoću De Morganovih pravila* (redoslijed kojim se vrši ovo oslobađanje ne mora biti jednoznačan), a zatim *obaviti sve moguće konjunkcije* (množenja) koje se mogu obaviti prema *pravilu distributivnosti disjunkcije prema konjunkciji*. Na kraju se, pomoću trivijalnih pravila, (poput pravila *idempotentnosti* ili *konzistentnosti*) oslobodimo eventualnog suvišnog pojavljivanja istih promjenljivih ili članova.

Primjer 7.1:

- Svesti funkciju $Y = \overline{AB(A \vee \overline{B}C) \vee \overline{\overline{AC}\,\overline{BC}}}$ na disjunktivnu normalnu formu.

Svođenje ćemo provesti postupno, u skladu sa gore opisanim postupkom:

$$
\begin{aligned}
Y &= \overline{AB(A \vee \overline{B}C) \vee \overline{\overline{AC}\,\overline{BC}}} = \overline{AB(A \vee \overline{B}C) \vee AC \vee BC} = \overline{AB(A \vee \overline{B}C)}\;\overline{AC}\,\overline{BC} = \\
&= [\overline{AB} \vee \overline{(A \vee \overline{B}C)}]\,\overline{AC}\,\overline{BC} = (\overline{A} \vee \overline{B} \vee \overline{A}\,\overline{\overline{B}C})(\overline{A} \vee \overline{C})(\overline{B} \vee \overline{C}) = \\
&= [\overline{A} \vee \overline{B} \vee \overline{A}(B \vee \overline{C})](\overline{A} \vee \overline{C})(\overline{B} \vee \overline{C}) = (\overline{A} \vee \overline{B} \vee \overline{A}B \vee \overline{A}\,\overline{C})(\overline{A}\,\overline{B} \vee \overline{A}\,\overline{C} \vee \overline{B}\,\overline{C} \vee \overline{C}) = \\
&= \overline{A}\,\overline{A}\,\overline{B} \vee \overline{B}\,\overline{A}\,\overline{B} \vee \overline{A}B\overline{A}\,\overline{B} \vee \overline{A}\,\overline{C}\,\overline{A}\,\overline{B} \vee \overline{A}\,\overline{A}\,\overline{C} \vee \overline{B}\,\overline{A}\,\overline{C} \vee \overline{A}B\overline{A}\,\overline{C} \vee \overline{A}\,\overline{C}\,\overline{A}\,\overline{C} \vee \\
&\quad \vee \overline{A}\,\overline{B}\,\overline{C} \vee \overline{B}\,\overline{B}\,\overline{C} \vee \overline{A}B\overline{B}\,\overline{C} \vee \overline{A}\,\overline{C}\,\overline{B}\,\overline{C} \vee \overline{A}\,\overline{C} \vee \overline{B}\,\overline{C} \vee \overline{A}B\overline{C} \vee \overline{A}\,\overline{C}\,\overline{C} = \\
&= \overline{A}\,\overline{B} \vee \overline{A}\,\overline{B} \vee 0 \vee \overline{A}\,\overline{B}\,\overline{C} \vee \overline{A}\,\overline{C} \vee \overline{A}\,\overline{B}\,\overline{C} \vee \overline{A}B\overline{C} \vee \overline{A}\,\overline{C} \vee \\
&\quad \vee \overline{A}\,\overline{B}\,\overline{C} \vee \overline{B}\,\overline{C} \vee 0 \vee \overline{A}\,\overline{B}\,\overline{C} \vee \overline{A}\,\overline{C} \vee \overline{B}\,\overline{C} \vee \overline{A}B\overline{C} \vee \overline{A}\,\overline{C} = \\
&= \overline{A}\,\overline{B} \vee \overline{A}\,\overline{B}\,\overline{C} \vee \overline{A}\,\overline{C} \vee \overline{B}\,\overline{C} \vee \overline{A}B\overline{C} = \\
&= \overline{A}\,\overline{B} \vee \overline{A}\,\overline{C} \vee \overline{B}\,\overline{C}
\end{aligned}
$$

U ovom primjeru, već predposljednji red predstavlja disjunktivnu normalnu formu. Međutim, mi smo izvršili dodatno skraćenje ove forme dvostrukom primjenom *pravila apsorpcije za disjunkciju*:

$$\overline{A}\,\overline{B} \vee \overline{A}\,\overline{B}\,\overline{C} = \overline{A}\,\overline{B}, \qquad \overline{A}\,\overline{C} \vee \overline{A}B\overline{C} = \overline{A}\,\overline{C}$$

Prilikom nalaženja DNF, postupak se može osjetno skratiti ukoliko u toku postupka vršimo primjenu pravila sažimanja i apsorpcije kada se god za to ukaže mogućnost. Tako smo, na primjer, u prethodnom primjeru već u drugom redu mogli uočiti da na osnovu pravila apsorpcije slijedi da je

$$\overline{A} \vee \overline{A}\,\overline{\overline{B}C} = \overline{A}$$

i na taj način bitno skratiti postupak.

Da bismo neku logičku funkciju predstavljenu formulom u kojoj se javljaju samo konjunkcije, disjunkcije i negacije sveli na *savršenu disjunktivnu normalnu formu*, prvo je svedemo na neku od *disjunktivnih normalnih formi* (bilo koju), a zatim u svakom članu u kojem nedostaje neka od promjenljivih *uvedemo nedostajuću promjenljivu* pomoću razvoja oblika

$$X = XY \vee X\overline{Y}$$

koji nije ništa drugo nego pravilo apsorpcije za disjunkciju napisano u obrnutom smjeru. Ukoliko u nekom članu nedostaje više promjenljivih ovaj razvoj po potrebi primjenjujemo više puta sve dok ne uvedemo *sve nedostajuće promjenljive*. Postupak ponavljamo dok se svi članovi ne dovedu u oblik koji sadrži sve promjenljive. Na kraju se oslobodimo eventualnog *suvišnog pojavljivanja istih članova* koji mogu nastati kao posljedica provedene ekspanzije.

Primjer 7.2:

- Naći savršenu disjunktivnu normalnu formu funkcije iz prethodnog primjera.

Krenućemo od DNF koju smo našli u prethodnom primjeru, i primijenićemo opisani postupak:

$$\begin{aligned} Y &= \overline{A}\,\overline{B} \vee \overline{B}\,\overline{C} \vee \overline{A}\,\overline{C} = \overline{A}\,\overline{B}C \vee \overline{A}\,\overline{B}\,\overline{C} \vee \overline{B}\,\overline{C} \vee \overline{A}\,\overline{C} = \\ &= \overline{A}\,\overline{B}C \vee \overline{A}\,\overline{B}\,\overline{C} \vee A\overline{B}\,\overline{C} \vee \overline{A}\,\overline{B}\,\overline{C} \vee \overline{A}\,\overline{C} = \\ &= \overline{A}\,\overline{B}C \vee \overline{A}\,\overline{B}\,\overline{C} \vee A\overline{B}\,\overline{C} \vee \overline{A}\,\overline{B}\,\overline{C} \vee \overline{A}B\overline{C} \vee \overline{A}\,\overline{B}\,\overline{C} = \\ &= \overline{A}\,\overline{B}C \vee \overline{A}\,\overline{B}\,\overline{C} \vee A\overline{B}\,\overline{C} \vee \overline{A}B\overline{C} \end{aligned}$$

Na ovaj način smo dobili traženu SDNF.

Svođenje proizvoljne logičke funkcije u kojoj se javljaju samo konjunkcije, disjunkcije i negacije na konjunktivnu normalnu formu, principijelno je moguće obaviti sličnim postupkom kao i svođenje na DNF, jedino je umjesto pravila distributivnosti disjunkcije prema konjunkciji potrebno koristiti *pravilo distributivnosti konjunkcije prema disjunkciji*. Međutim, za razliku od prvog pravila koje glasi $X(Y \vee Z) = XY \vee XZ$, i koje se primjenjuje rutinski zbog činjenice da podsjeća na slično pravilo $X(Y + Z) = XY + XZ$ iz elementarne matematike, primjena drugog pravila $X \vee YZ = (X \vee Y)(X \vee Z)$ zahtijeva povećanu koncentraciju, jer slično pravilo ne postoji u elementarnoj matematici. Stoga je pri primjeni ovog pravila veća vjerovatnoća da ćemo nešto previdjeti ili napraviti neku drugu grešku.

Primjer 7.3:

- Svesti funkciju $Y = A \vee B(C \vee \overline{D})$ na konjunktivnu i savršenu konjunktivnu normalnu formu.

Za ovu funkciju KNF nije teško naći ni neposrednom primjenom pravila $X \vee YZ = (X \vee Y)(X \vee Z)$:

$$Y = A \vee B(C \vee \overline{D}) = (A \vee B)(A \vee C \vee \overline{D})$$

što nije ništa drugo nego tražena KNF.

Sada bismo SKNF mogli naći primjenom razvoja

$$X = (X \vee Y)(X \vee \overline{\overline{Y}})$$

koji je analogan razvoju $X = XY \vee X\overline{Y}$. Ovaj razvoj je veoma lako dokazati:

$$\begin{aligned} (X \vee Y)(X \vee \overline{\overline{Y}}) &= XX \vee X\overline{Y} \vee YX \vee Y\overline{Y} = X \vee X\overline{Y} \vee XY \vee 0 = X \vee X(\overline{Y} \vee Y) = \\ &= X \vee X \cdot 1 = X \vee X = X \end{aligned}$$

Primjenom ovog razvoja na nađenu KNF dobijamo:

$$\begin{aligned} Y &= A \vee B(C \vee \overline{D}) = (A \vee B)(A \vee C \vee \overline{D}) = (A \vee B \vee C)(A \vee B \vee \overline{C})(A \vee C \vee \overline{D}) = \\ &= (A \vee B \vee C)(A \vee B \vee \overline{C})(A \vee B \vee C \vee \overline{D})(A \vee \overline{B} \vee C \vee \overline{D}) = \\ &= (A \vee B \vee C \vee D)(A \vee B \vee C \vee \overline{D})(A \vee B \vee \overline{C})(A \vee B \vee C \vee \overline{D})(A \vee \overline{B} \vee C \vee \overline{D}) = \\ &= (A \vee B \vee C \vee D)(A \vee B \vee C \vee \overline{D})(A \vee B \vee \overline{C})(A \vee \overline{B} \vee C \vee \overline{D}) = \\ &= (A \vee B \vee C \vee D)(A \vee B \vee C \vee \overline{D})(A \vee B \vee \overline{C} \vee D)(A \vee B \vee \overline{C} \vee \overline{D})(A \vee \overline{B} \vee C \vee \overline{D}) \end{aligned}$$

Ovim smo formirali i traženu SKNF.

Kao što je već rečeno, prilikom ručnog svođenja iole složenije logičke funkcije na KNF mnogo je lakše pogriješiti nego prilikom svođenja na DNF (uglavnom zbog već opisanog psihološkog faktora). Da bi se uklonile ove poteškoće, svođenje proizvoljne logičke funkcije na KNF najčešće se obavlja *indirektnim postupkom*, odnosno postupkom *dvojne negacije*. Tako, da bi se neka funkcija svela na KNF (ili SKNF), najpraktičnije je prvo formirati *njenu negaciju*, zatim tu negaciju svesti na DNF (odnosno SDNF), i na kraju *negirati dobijeni rezultat* korištenjem De Morganovih pravila. Suština postupka oslanja se na činjenicu da De Morganova pravila prevode disjunktivne normalne forme u konjunktivne normalne forme, i obrnuto.

Primjer 7.4:

- Svesti funkciju iz Primjera 7.3 na konjunktivnu i savršenu konjunktivnu normalnu formu postupkom dvojne negacije.

  Nađimo prvo negaciju funkcije Y i svedimo je na DNF i SDNF.

$$\begin{aligned}
\overline{Y} &= \overline{A \vee B(C \vee \overline{D})} = \overline{A}\,\overline{B(C \vee \overline{D})} = \overline{A}(\overline{B} \vee \overline{C \vee \overline{D}}) = \overline{A}(\overline{B} \vee \overline{C}D) = \\
&= \overline{A}\,\overline{B} \vee \overline{A}\,\overline{C}D = && \text{/ DNF /} \\
&= \overline{A}\,\overline{B}C \vee \overline{A}\,\overline{B}\,\overline{C} \vee \overline{A}\,\overline{B}\overline{C}D \vee \overline{A}\,\overline{B}\overline{C}D = \\
&= \overline{A}\,\overline{B}CD \vee \overline{A}\,\overline{B}C\overline{D} \vee \overline{A}\,\overline{B}\,\overline{C}D \vee \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} \vee \overline{A}\,\overline{B}\overline{C}D \vee \overline{A}\,\overline{B}\overline{C}D = \\
&= \overline{A}\,\overline{B}CD \vee \overline{A}\,\overline{B}C\overline{D} \vee \overline{A}\,\overline{B}\,\overline{C}D \vee \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} \vee \overline{A}\,\overline{B}\overline{C}D && \text{/ SDNF /}
\end{aligned}$$

KNF oblik funkcije Y sada nalazimo negacijom DNF za funkciju $\overline{Y}$:

$$Y = \overline{\overline{Y}} = \overline{\overline{A}\,\overline{B} \vee \overline{A}\,\overline{C}D} = \overline{\overline{A}}\,\overline{\overline{B}}\,\overline{\overline{A}}\,\overline{\overline{C}}D = (A \vee B)(A \vee C \vee \overline{D})$$

Analogno, SKNF oblik funkcije Y nalazimo negacijom SDNF za funkciju $\overline{Y}$:

$$\begin{aligned}
Y = \overline{\overline{Y}} &= \overline{\overline{A}\,\overline{B}CD \vee \overline{A}\,\overline{B}C\overline{D} \vee \overline{A}\,\overline{B}\,\overline{C}D \vee \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} \vee \overline{A}\,\overline{B}\overline{C}D} = \\
&= \overline{\overline{A}\,\overline{B}CD}\;\overline{\overline{A}\,\overline{B}C\overline{D}}\;\overline{\overline{A}\,\overline{B}\,\overline{C}D}\;\overline{\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}}\;\overline{\overline{A}\,\overline{B}\overline{C}D} = \\
&= (A \vee B \vee \overline{C} \vee \overline{D})(A \vee B \vee \overline{C} \vee D)(A \vee B \vee C \vee \overline{D})(A \vee B \vee C \vee D)(A \vee \overline{B} \vee C \vee \overline{D})
\end{aligned}$$

Kao što vidimo, dobili smo rezultat koji se od rezultata prethodnog primjera razlikuje samo u poretku članova. Ovo je i logično, jer smo već rekli da se svaka logička funkcija na jedinstven način može predstaviti savršenom disjunktivnom ili savršenom konjunktivnom normalnom formom, u smislu da se dvije SDNF ili dvije SKNF za istu funkciju mogu eventualno razlikovati samo u poretku pojedinih članova.

Primjer 7.5:

- Svesti funkciju $Y = A \vee \overline{B(A \vee \overline{C})}$ na KNF i SKNF.

  Primjenom istog postupka kao u prethodnom primjeru, imamo:

$$\overline{Y} = \overline{A \vee \overline{B(A \vee \overline{C})}} = \overline{A}\,\overline{\overline{B(A \vee \overline{C})}} = \overline{A}B(A \vee \overline{C}) = A\overline{A}B \vee \overline{A}B\overline{C} = \overline{A}B\overline{C}$$

$$Y = \overline{\overline{Y}} = \overline{\overline{A}B\overline{C}} = A \vee \overline{B} \vee C$$

Rezultat je trivijalan u smislu da se funkcija reducira na makstermu. Primijetimo da se bilo koja maksterma u smislu navedenih definicija može smatrati kao DNF, KNF i SKNF (ali ne i kao SDNF) u isto vrijeme. Slično se svaka minterma može smatrati kao DNF, SDNF i KNF (ali ne i kao SKNF) u isto vrijeme!

Za tablično zadane logičke funkcije sasvim je lako odrediti njihov zapis u obliku savršene disjuktivne ili konjunktivne normalne forme. Ako uočimo da svaka minterma *poprima vrijednost 1 samo za jednu kombinaciju promjenljivih* (dakle, u samo jednom redu tablice istine), dok svaka maksterma *poprima vrijednost 0 samo za jednu kombinaciju promjenljivih*, veoma je lako dokazati sljedeća pravila (ona su, zapravo, gotovo očigledna):

- Da bismo formirali *savršenu disjunktivnu normalnu formu* za tablično zadanu logičku funkciju, za svaki red tablice u kojem funkcija uzima vrijednost 1 formiramo *mintermu* u kojoj *one promjenljive koje su jednake nuli u tom redu tablice ulaze sa negacijom*, a *one koje su jednake jedinici ulaze bez negacije*. Tražena SDNF je *disjunkcija svih takvih mintermi*.
- Da bismo formirali *savršenu konjunktivnu normalnu formu* za tablično zadanu logičku funkciju, za svaki red tablice u kojem funkcija uzima vrijednost 0 formiramo *makstermu* u kojoj *one promjenljive koje su jednake nuli u tom redu tablice ulaze bez negacije*, a *one koje su jednake jedinici ulaze sa negacijom*. Tražena SKNF je *konjunkcija svih takvih makstermi*.

Opisani postupak je najlakše ilustrirati kroz konkretan primjer.

Primjer 7.6:

- Predstaviti funkciju datu sljedećom tabelom kao SDNF i kao SKNF.

A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Primjenom opisanih pravila neposredno nalazimo:

$$Y = \overline{A}\,\overline{B}\,\overline{C} \vee \overline{A}\,\overline{B}C \vee \overline{A}BC \vee A\overline{B}\,\overline{C} \vee ABC \qquad / \text{ SDNF } /$$

$$Y = (A \vee \overline{B} \vee C)(\overline{A} \vee B \vee \overline{C})(\overline{A} \vee \overline{B} \vee C) \qquad / \text{ SKNF } /$$

Na osnovu činjenice da se svaka logička funkcija može predstaviti tabelom, i činjenice da se svaka tabelarno zadana logička funkcija može predstaviti kao SDNF ili SKNF, neposredno slijedi da se svaka logička funkcija može izraziti preko operacija konjunkcije, disjunkcije i negacije, čime je zapravo dokazana već ranije iskazana tvrdnja.

Minterme se često označavaju oznakom m_i ili K_i gdje je *i* indeks (redni broj) reda tabele istine u kojoj minterma uzima vrijednost 1, pri čemu se indeksiranje vrši od nule (a ne od jedinice). Tako, indeks raspisan kao binarni broj ujedno daje vrijednosti promjenljivih za koje minterma uzima vrijednost 1. Slično, maksterme se označavaju oznakom M_i, gdje je *i* indeks reda tabele istine u kojoj maksterma uzima vrijednost 0. U skladu sa ovakvim označavanjem, prethodna funkcija se može zapisati i na sljedeće načine:

$$Y = m_0(A, B, C) \vee m_1(A, B, C) \vee m_3(A, B, C) \vee m_4(A, B, C) \vee m_7(A, B, C) \quad / \text{ SDNF } /$$

$$Y = M_2(A, B, C)\, M_5(A, B, C)\, M_6(A, B, C) \quad / \text{ SKNF } /$$

Ukoliko je iz konteksta jasno od kojih promjenljivih se sastoje minterme odnosno maksterme, oznake promjenljivih se mogu izostaviti, tako da možemo skraćeno pisati samo:

$$Y = m_0 \vee m_1 \vee m_3 \vee m_4 \vee m_7 \quad / \text{ SDNF } /$$

$$Y = M_2\, M_5\, M_6 \quad / \text{ SKNF } /$$

Primjer 7.7:

- Izraziti ekskluzivnu disjunkciju i implikaciju pomoću osnovnih logičkih operacija (konjunkcije, disjunkcije i negacije).

  Ekskluzivna disjunkcija, kao što znamo, može se predstaviti sljedećom tabelom istine:

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

Sada na osnovu pravila za nalaženje SDNF i SKNF neposredno slijede relacije:

$$A \oplus B = m_1(A, B) \vee m_2(A, B) = \overline{A}B \vee A\overline{B}$$

$$A \oplus B = M_0(A, B)\, M_3(A, B) = (\overline{A} \vee \overline{B})(A \vee B)$$

Prva od ovih formula koristi se znatno češće. Međutim, negacijom druge formule lako se izvodi formula

$$\overline{A \oplus B} = AB \vee \overline{A}\,\overline{B}$$

koja se veoma često koristi.

Za implikaciju vrijedi sljedeća tablica istine, koja je poznata iz elementarne matematičke logike:

A	B	A⇒B
0	0	1
0	1	1
1	0	0
1	1	1

Odavde slijedi:

$$A \Rightarrow B = \overline{A} \vee B$$

Drugim riječima, operacija implikacije svodi se na prostu makstermu $M_2(A, B)$.

Već smo rekli da se svaka logička funkcija može izraziti preko operacija konjunkcije, disjunkcije i negacije. U suštini su dovoljne *samo negacija i konjunkcija*, jer se disjunkcija može izraziti preko ove dvije operacije korištenjem formule

$$A \vee B = \overline{\overline{A}\,\overline{B}}$$

Na sličan način zaključujemo da su dovoljne samo *negacija i disjunkcija,* jer vrijedi formula

$$AB = \overline{\overline{A} \vee \overline{B}}$$

Štaviše, svaka logička funkcija može se zapravo izraziti pomoću jedne jedine operacije: *Shefferove operacije* (negacije konjunkcije). Alternativno, svaka logička funkcija može se izraziti i samo pomoću *Pierceove operacije* (negacije disjunkcije). Da bi dokazali ovu tvrdnju, dovoljno je pokazati da se konjunkcija, disjunkcija i negacija mogu izraziti preko Shefferove ili Pierceove operacije. I zaista, vrijede sljedeća pravila (zagrade su neophodne, s obzirom da ove operacije *nisu asocijativne*):

$$\overline{X} = \overline{XX} = X \uparrow X$$
$$X \vee Y = \overline{\overline{X}\,\overline{Y}} = \overline{X} \uparrow \overline{Y} = (X \uparrow X) \uparrow (Y \uparrow Y)$$
$$XY = \overline{\overline{X\,Y}} = \overline{X \uparrow Y} = (X \uparrow Y) \uparrow (X \uparrow Y)$$

$$\overline{X} = \overline{X \vee X} = X \downarrow X$$
$$X \vee Y = \overline{\overline{X \vee Y}} = \overline{X \downarrow Y} = (X \downarrow Y) \downarrow (X \downarrow Y)$$
$$XY = \overline{\overline{X} \vee \overline{Y}} = \overline{X} \downarrow \overline{Y} = (X \downarrow X) \downarrow (Y \downarrow Y)$$

Dokazana tvrdnja omogućava da se ma koja logička funkcija može realizirati pomoću *samo jednog tipa logičkih kola*, npr. samo pomoću NAND kola, ili samo pomoću NOR kola. Ova činjenica je od velikog značaja u digitalnoj tehnici, jer je tehnološki mnogo jednostavnije na malom prostoru proizvesti npr. 1000 kola istog tipa, nego 300 kola različitih tipova. Stoga se NAND i NOR kola nazivaju ***univerzalna logička kola***. Principijelno je čak i cijelu centralnu jedinicu računara moguće izgraditi samo pomoću NAND ili NOR kola. Svođenje proizvoljne funkcije na oblik u kojem se javljaju samo Shefferova ili Pierceova operacija u načelu se može obaviti direktnom primjenom gore prikazanih formula, ali dobijeni izrazi mogu biti mnogo glomazniji nego što bi mogli biti. U Poglavlju 10. ćemo se upoznati sa prikladnijim postupcima za ostvarenje istog cilja.

Iz dosada izloženog vidljivo je da postoji više različitih elementarnih logičkih funkcija koje se mogu iskoristiti za realizaciju proizvoljne logičke funkcije. Ponekad se u primjenama susreće i tzv. ***Žegalkinova algebra***, u kojoj se koriste samo operacije konjunkcije i ekskluzivne disjunkcije. I ove dvije operacije su dovoljne za realizaciju proizvoljne logičke funkcije, ako pored njih dozvolimo i upotrebu konstante 1. Zaista, imamo

$$\overline{X} = 1 \oplus X, \qquad X \vee Y = X \oplus Y \oplus XY$$

Posljednja formula se može napisati i u sljedećem obliku (podrazumijeva se da konjunkcija, odnosno množenje, ima veći prioritet u odnosu na operaciju ekskluzivne disjunkcije):

$$X \vee Y = 1 \oplus (1 \oplus X)(1 \oplus Y)$$

Mada ova formula djeluje komplikovanija od prethodne, njena prednost je u tome što se lako generalizira na proizvoljan broj promjenljivih. Naime, indukcijom je lako pokazati da vrijedi:

$$X_1 \vee X_2 \vee \ldots \vee X_n = 1 \oplus (1 \oplus X_1)(1 \oplus X_2) \ldots (1 \oplus X_n)$$

Žegalkinova algebra je značajna u teoretskim analizama logičkih funkcija zbog činjenice da su njene zakonitosti u priličnoj mjeri srodne zakonima obične aritmetike, pri čemu operacija ekskluzivne disjunkcije preuzima ulogu sabiranja. Posebno se ističe zakon distributivnosti ekskluzivne disjunkcije prema konjunkciji, tj. pravilo $X(Y \oplus Z) = XY \oplus XZ$. Navedena pravila, zajedno sa pravilima $X \oplus X = 0$ i $X \oplus 0 = X$, kao i osobinom komutativnosti i asocijativnosti ekskluzivne disjunkcije, dovoljna su za osnovne manipulacije sa formulama Žegalkinove algebre. Posebno, primjenom ovih pravila svaka funkcija može se izraziti u ***standardnoj formi Žegalkinove algebre***, koji je izražen kao ekskluzivna disjunkcija članova koji mogu biti ili konstanta 1, ili individualne promjenljive, ili konjunkcije (produkti) individualnih promjenljivih.

Primjer 7.8:

- Predstaviti logičku funkciju $Y = \overline{A}BC \vee A\overline{B}C \vee AB\overline{C}$ u standardnoj formi Žegalkinove algebre.

Primjenom ranije navedenih pravila možemo pisati:

$$\begin{aligned} Y = \overline{A}BC \vee A\overline{B}C \vee AB\overline{C} &= 1 \oplus [1 \oplus (1 \oplus A)BC][1 \oplus A(1 \oplus B)C][1 \oplus AB(1 \oplus C)] = \\ &= 1 \oplus (1 \oplus BC \oplus ABC)(1 \oplus AC \oplus ABC)(1 \oplus AB \oplus ABC) = \\ &= 1 \oplus (1 \oplus AC \oplus ABC \oplus BC \oplus ABC \oplus ABC \oplus ABC \oplus ABC \oplus ABC)(1 \oplus AB \oplus ABC) = \\ &= 1 \oplus (1 \oplus AC \oplus BC)(1 \oplus AB \oplus ABC) = \\ &= 1 \oplus 1 \oplus AB \oplus ABC \oplus AC \oplus ABC \oplus ABC \oplus BC \oplus ABC \oplus ABC = \\ &= AB \oplus AC \oplus BC \oplus ABC \end{aligned}$$

Vidimo da postoji mnogo različitih elementarnih logičkih funkcija koje su same za sebe dovoljne za izražavanje proizvoljne logičke funkcije. Postoji i opći rezultat koji iskazuje potrebne i dovoljne uvjete da bi se pomoću nekoliko izabranih elementarnih logičkih funkcija mogla izraziti proizvoljna logička funkcija. Za tu svrhu, potrebno je uvesti nekoliko pojmova, koji su dati kroz sljedeću definiciju:

Definicija 7.9:

Za logičku funkciju $f(x_1, x_2, x_3, ...x_n)$ kažemo:

- da ***zadržava nulu*** ukoliko vrijedi $f(0,0,0,...0) = 0$;
- da ***zadržava jedinicu*** ukoliko vrijedi $f(1,1,1,...1) = 1$;
- da je ***linearna*** ukoliko se može napisati u obliku $f(x_1, x_2, x_3, ...x_n) = a_0 \oplus a_1x_1 \oplus a_2x_2 \oplus ...a_nx_n$ gdje su a_0, a_1, ... a_n neke konstante iz skupa $\{0,1\}$;
- da je ***samodualna*** ukoliko vrijedi $f(\overline{x}_1, \overline{x}_2, \overline{x}_3, ...\overline{x}_n) = \overline{f(x_1, x_2, x_3, ...x_n)}$;
- da je ***neopadajuća*** (ili ***monotona***) ukoliko iz $x_i' \le x_i''$ slijedi $f(x_1, ...x_i', ...x_n) \le f(x_1, ...x_i'', ...x_n)$ za sve indekse *i* od 1 do *n*.

Pomenuti opći rezultat iskazan je kroz sljedeću teoremu, poznatu kao ***Postova teorema***:

Teorema 7.1 (E. L. Post):

Da bi se pomoću skupa logičkih funkcija $f_1(x_1, x_2, ...x_n)$, $f_2(x_1, x_2, ...x_n)$, ... $f_k(x_1, x_2, ...x_n)$ mogla izraziti proizvoljna logička funkcija, potrebno je i dovoljno da se među funkcijama f_1, f_2, ... f_k nalazi barem jedna funkcija koja *ne zadržava nulu*, barem jedna funkcija koja *ne zadržava jedinicu*, barem jedna funkcija koja *nije linearna*, barem jedna funkcija koja *nije samodualna*, i barem jedna funkcija koja *nije neopadajuća*.

Na primjer, za slučaj klasične *Booleove algebre*, konjunkcija zadržava nulu i jedinicu i neopadajuća je, ali *nije linearna* i *nije samodualna*, dok je negacija linearna i samodualna, ali *ne zadržava niti nulu niti jedinicu* i *nije neopadajuća*. Stoga su po Teoremi 7.1. konjunkcija i negacija dovoljne za realizaciju proizvoljne logičke funkcije. U slučaju *Žegalkinove algebre* imamo konjunkciju (koja *nije linearna* niti *samodualna*, ali je neopadajuća i zadržava nulu i jedinicu), kao i ekskluzivnu disjunkciju koja zadržava nulu i linearna je, ali *ne zadržava jedinicu*, *nije samodualna* i *nije neopadajuća*. Očigledno nam za ispunjenje uvjeta Teoreme 7.1. nedostaje funkcija koja *ne zadržava nulu*. Takva je očigledno "funkcija" konstanta 1 (funkcija sa nula argumenata), koja očigledno ne zadržava nulu (mada je linearna, samodualna, neopadajuća i zadržava jedinicu). U slučaju *Shefferove* i *Pierceove operacije*, svaka od njih ne zadržava niti nulu niti jedinicu, nije linearna, nije samodualna i nije monotono rastuća, tako da je po Teoremi 7.1. svaka od njih sama za sebe dovoljna za realizaciju proizvoljne logičke funkcije.

Na kraju, treba još istaći da se svaka logička funkcija može napisati i kao *obični aritmetički izraz*, koristeći samo klasične aritmetičke operacije sabiranja, oduzimanja i množenja. Zaista, konjunkcija se na skupu vrijednosti $\{0,1\}$ svakako poklapa sa običnim množenjem, dok za negaciju i disjunkciju vrijede sljedeće formule, koje se lako provjeravaju prostim uvrštavanjem:

$$\overline{X} = 1 - X, \qquad X \vee Y = X + Y - XY = 1 - (1 - X)(1 - Y)$$

Druga od dvije ponuđene varijante za izražavanje disjunkcije preko aritmetičkih operacija lako se generalizira za slučaj više promjenljivih:

$$X_1 \vee X_2 \vee \ldots \vee X_n = 1 - (1 - X_1)(1 - X_2) \ldots (1 - X_n)$$

Primjenom navedenih formula, svaka logička funkcija može se predstaviti kao *polinom* po svim promjenljivim od kojih funkcija zavisi. Pored toga, taj polinom se uvijek može svesti na tzv. ***multilinearni polinom***, odnosno polinom koji je *prvog stepena po svakoj od svojih promjenljivih*. Zaista, pošto za $X \in \{0,1\}$ vrijedi $X^n = X$ za ma kakvo n, to se svi viši stepeni ma koje od promjenljivih mogu zamijeniti prvim stepenom. Postupak kojim se logička funkcija prevodi u ekvivalentni oblik multilinearnog polinoma naziva se ***aritmetizacija*** logičke funkcije.

Primjer 7.9:

- Aritmetizirati logičku funkciju $Y = \overline{A}BC \vee A\overline{B}C \vee AB\overline{C}$.

Primjenom ranije navedenih pravila možemo pisati:

$$\begin{aligned}
Y = \overline{A}BC \vee A\overline{B}C \vee AB\overline{C} &= 1 - [1 - (1 - A)BC][1 - A(1 - B)C][1 - AB(1 - C)] = \\
&= 1 - (1 - BC + ABC)(1 - AC + ABC)(1 - AB + ABC) = \\
&= 1 - (1 - AC + ABC - BC + ABC^2 - AB^2C^2 + ABC - A^2BC^2 + A^2B^2C^2)(1 - AB + ABC) = \\
&= 1 - (1 - AC + ABC - BC + ABC - ABC + ABC - ABC + ABC)(1 - AB + ABC) = \\
&= 1 - (1 - AC - BC + 2ABC)(1 - AB + ABC) = \\
&= 1 - (1 - AB + ABC - AC + A^2BC - A^2BC^2 - BC + AB^2C - AB^2C^2 + 2ABC - 2A^2B^2C + 2A^2B^2C^2) = \\
&= 1 - (1 - AB + ABC - AC + ABC - ABC - BC + ABC - ABC + 2ABC - 2ABC + 2ABC) = \\
&= 1 - (1 - AB - AC - BC + 3ABC) = AB + AC + BC - 3ABC
\end{aligned}$$

Primijetimo da se u aritmetiziranom obliku logičke funkcije mogu javljati i konstante različite od 0 i 1, kao i da pojedini podizrazi dobijenog izraza mogu također uzimati vrijednosti različite od 0 i 1 (na primjer, podizraz $AB + AC$ u gore prikazanom aritmetiziranom obliku funkcije ima vrijednost 2 za $A = B = C$). Značaj aritmetizacije logičkih funkcija leži prvenstveno u činjenici da ona omogućava da se logički izrazi tretiraju kao obični aritmetički izrazi, što ponekad može biti korisno.

(?) Pitanja i zadaci

7.1 Definirajte sljedeće pojmove: literal, elementarna konjunkcija, elementarna disjunkcija, disjunktivna normalna forma (DNF), savršena disjunktivna normalna forma (SDNF), konjunktivna normalna forma (KNF), savršena konjunktivna normalna forma (SKNF), minterma, maksterma.

7.2 Kako definiramo dužinu neke konjunktivne odnosno disjunktivne normalne forme?

7.3 Definirajte minimalnu disjunktivnu formu (MDNF) i minimalnu konjunktivnu normalnu formu (MKNF).

7.4 Prikažite funkcije $m_{14}(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$ i $M_{14}(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$ u analitičkom obliku (koristeći standardne logičke funkcije).

7.5 Dokažite da vrijedi $M_i(x_1, x_2, x_3, ...) = \overline{m_i(x_1, x_2, x_3, ...)}$ za svako *i*.

7.6 Svedite logičku funkciju $Y = (A \vee \overline{B} \vee C)(A \vee \overline{B} \vee \overline{C})(\overline{A} \vee B \vee C)(\overline{A} \vee \overline{B} \vee C)$ na DNF primjenom zakona distributivnosti, a zatim proširite dobijenu DNF do SDNF oblika.

7.7 Svedite logičku funkciju $Y = (\overline{A} \vee \overline{B} \vee D)(\overline{A} \vee D)(C \vee D)$ prvo na neki od DNF oblika, a zatim na SDNF oblik. SDNF oblik prikažite i pomoću m_i notacije.

7.8 Svedite logičku funkciju $Y = \overline{A}\,\overline{B}\,\overline{C} \vee \overline{A}\,\overline{B}C \vee A\overline{B}C \vee ABC$ prvo na neki od KNF oblika, a zatim na SKNF oblik. SKNF oblik prikažite i pomoću M_i notacije.

7.9 Nađite SDNF i SKNF za logičku funkciju $Y = AB\overline{C} \vee B\overline{C} \vee \overline{A}\,\overline{C} \vee \overline{B}C$. Rezultat izrazite u standardnom obliku, kao i korištenjem m_i odnosno M_i notacije.

7.10 Nađite SDNF i SKNF za logičku funkciju $Y = \overline{(A \vee \overline{\overline{B}} \vee \overline{\overline{A}C}\;\overline{A\overline{C}})\,\overline{\overline{A} \vee BC}}$.

7.11 Data je logička funkcija $Y = m_{15} \vee m_{10} \vee m_9 \vee m_8 \vee m_7 \vee m_2 \vee m_1 \vee m_0$ pri čemu sve minterme zavise od promjenljivih A, B, C i D respektivno.

a) Predstavite datu funkciju u SDNF obliku, korištenjem standardne notacije.
b) Predstavite datu funkciju u SKNF obliku. Rezultat prvo prikažite koristeći M_i notaciju, a zatim standardnu notaciju.
c) Nađite negaciju ove logičke funkcije u SDNF obliku. Rezultat prvo prikažite koristeći m_i notaciju, a zatim standardnu notaciju.
d) Nađite negaciju ove logičke funkcije u SDNF obliku. Rezultat prvo prikažite koristeći m_i notaciju, a zatim standardnu notaciju.

7.12 Data je logička funkcija $Y = \overline{\overline{AB \vee \overline{C}} \vee \overline{A \vee \overline{BC}}}$.

a) Predstavite ovu funkciju tabelom istine.
b) Na osnovu formirane tabele očitajte direktno SDNF i SKNF oblik zadane logičke funkcije. Provjerite rezultat primjenom pravila logičke algebre na polaznu funkciju.
c) Pronađite barem dvije DNF date funkcije koje su kraće od njene SDNF.
d) Pronađite barem jednu KNF date funkcije koja je kraća od njene SKNF.

7.13 Predstavite logičku funkciju $Y = A \Rightarrow B$ u SDNF obliku.

7.14 Predstavite sljedeće logičke funkcije u SDNF i SKNF obliku:
a) $Y = f(A, B, C) = A$ b) $Y = f(A, B, C) = B\overline{C}$ c) $Y = f(A, B, C) = 1$ d) $Y = f(A, B, C) = 0$

7.15 Svođenjem na SDNF oblik, pokažite da vrijedi $\overline{X} \oplus Y = X \oplus \overline{Y} = \overline{X \oplus Y}$.

7.16 Prikažite logičku funkciju $Y = B\overline{A \oplus C} \vee D(A \oplus C)$ u DNF i KNF obliku.

7.17 Pojednostavite sljedeće logičke funkcije, koliko god je to moguće:

a) $Y = \overline{\overline{B}(A \oplus C)} \vee \overline{D\,\overline{A \oplus C}}$ b) $Y = \overline{\overline{B} \vee (A \oplus C)}\;\overline{D \vee \overline{A \oplus C}}$

7.18 Izrazite Shefferovu operaciju preko Pierceove operacije, kao i Pierceovu operaciju preko Shefferove operacije.

7.19 Bez prethodnog pojednostavljenja, prikažite logičku funkciju $Y = \overline{A}B \vee \overline{A\overline{C} \vee \overline{B}C}$

a) Samo pomoću Shefferove operacije b) Samo pomoću Pierceove operacije

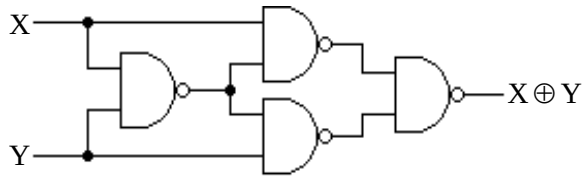
Nakon toga, ponovite prikaz polazeći od prethodno pojednostavljenog oblika iste funkcije.

7.20 Prikažite logičku funkciju $Y = A \oplus B$ preko Shefferove odnosno preko Pierceove operacije.

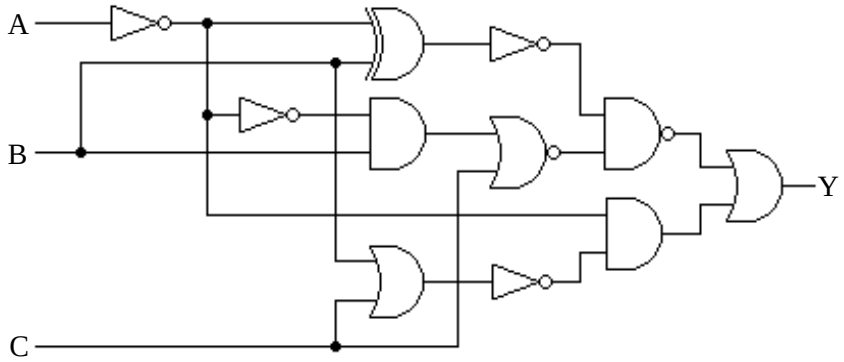
7.21 Dokažite identitet $A\overline{B} \vee A\overline{C}\,(\overline{AB\overline{C}} \vee \overline{A}\,\overline{B}) = (A \downarrow A) \downarrow B$.

7.22 Dokažite identitet $[(A \downarrow A) \downarrow (B \downarrow B)] \downarrow (A \downarrow B) = [A \uparrow (B \uparrow B)] \uparrow [B \uparrow (A \uparrow A)]$.

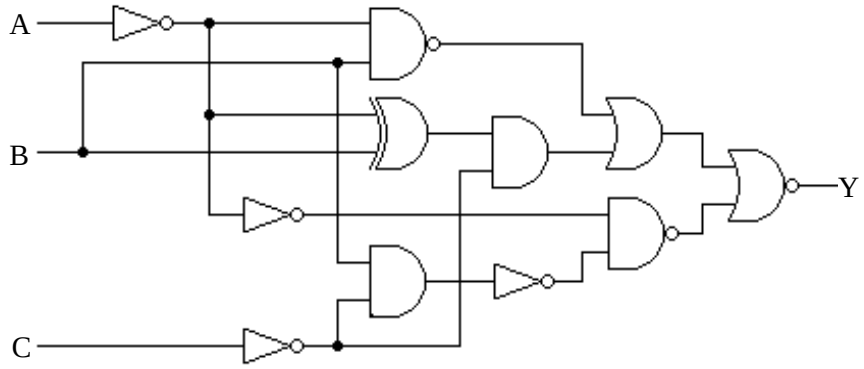
7.23 Koristeći prikaz ekskluzivne disjunkcije preko konjunkcije, disjunkcije i negacije, pokažite da sklop na slici simulira EXOR logičko kolo koristeći isključivo NAND logička kola.



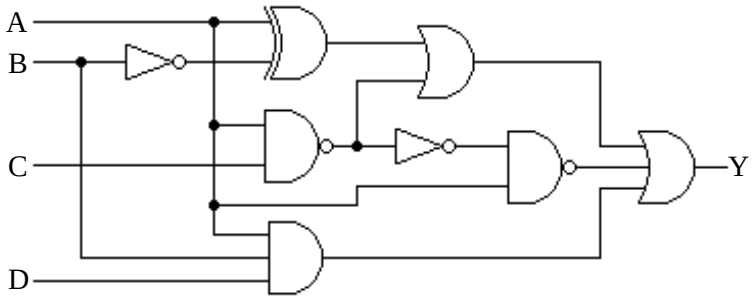
7.24 Odredite koju logičku funkciju obavlja sklop na sljedećoj slici, a zatim nađite što je god moguće jednostavniji sklop koji obavlja istu funkcionalnost. Za realizaciju je, po potrebi, pored AND, OR i NOT logičkih kola, dozvoljeno koristiti i EXOR odnosno EXNOR logička kola.



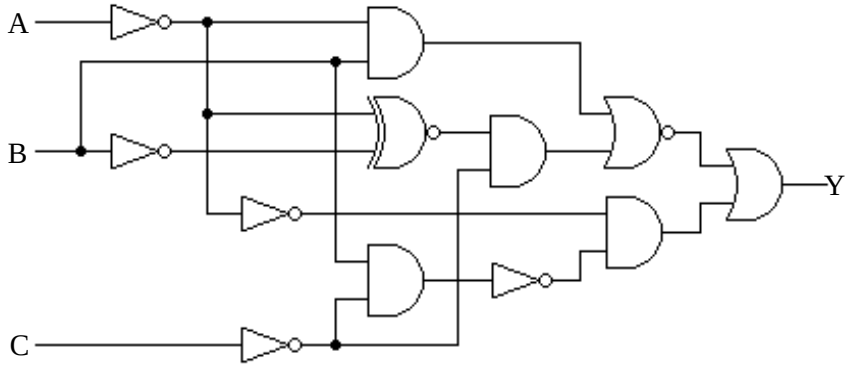
7.25 Odredite koju logičku funkciju obavlja sklop na sljedećoj slici, a zatim nađite što je god moguće jednostavniji sklop koji obavlja istu funkcionalnost.



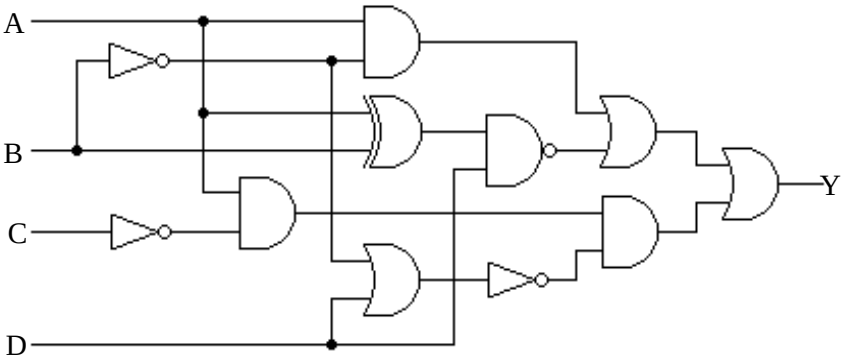
7.26 Odredite koju logičku funkciju obavlja sklop na sljedećoj slici, a zatim nađite što je god moguće jednostavniji sklop koji obavlja istu funkcionalnost.



7.27 Odredite koju logičku funkciju obavlja sklop na sljedećoj slici, a zatim nađite što je god moguće jednostavniji sklop koji obavlja istu funkcionalnost.



7.28 Odredite koju logičku funkciju obavlja sklop na sljedećoj slici, a zatim nađite što je god moguće jednostavniji sklop koji obavlja istu funkcionalnost.



7.29 Prikažite sljedeće logičke funkcije u standardnom obliku Žegalkinove algebre:

a) $Y = A \Rightarrow B$ b) $Y = \overline{A \oplus B}\ \overline{AB}$ c) $Y = (AB \vee C)(\overline{C} \vee AC)$

d) $Y = \overline{A}B \vee \overline{A\overline{C} \vee \overline{B}C}$ e) $Y = (A \vee B \vee C)(A \vee \overline{B} \vee C)$ f) $Y = \overline{AB \vee \overline{C}} \vee \overline{A \vee \overline{BC}}$

7.30 Ispitajte koje od logičkih funkcija $Y = 0$, $Y = 1$, $Y = A$, $Y = \overline{A}$, $Y = AB$, $Y = A \vee B$, $Y = A \oplus B$, $Y = A \downarrow B$, $Y = A \uparrow B$, $Y = A \Rightarrow B$ posjeduju svako od svojstava definiranih u Definiciji 7.9.

7.31* Dokažite da se svaka neopadajuća (monotona) logička funkcija različita od konstanti 0 ili 1 može izraziti samo preko operacija konjunkcije i disjunkcije.

7.32 Aritmetizirajte sljedeće logičke funkcije:

a) $Y = A \oplus B$ b) $Y = \overline{A \oplus B}$ c) $Y = \overline{A \vee B}$

d) $Y = A \Rightarrow B$ e) $Y = \overline{A \oplus B}\ \overline{AB}$ f) $Y = (AB \vee C)(\overline{C} \vee AC)$

g) $Y = \overline{A}B \vee \overline{A\overline{C} \vee \overline{B}C}$ h) $Y = (A \vee B \vee C)(A \vee \overline{B} \vee C)$ i) $Y = \overline{AB \vee \overline{C}} \vee \overline{A \vee \overline{BC}}$

8. Minimizacija logičkih funkcija (Quineov algoritam)

Kao što smo već vidjeli, minimizacija logičkih funkcija nije nimalo jednostavan zadatak ako se oslonimo samo na pravila Booleove algebre. Veliki značaj problema minimizacije doveo je do razvoja brojnih metoda za rješavanje ovog problema. Odmah na početku neophodno je naglasiti da problem minimizacije logičkih funkcija spada u klasu vrlo nezgodnih, takozvanih *NP kompletnih problema*, što kao posljedicu ima činjenicu da sve poznate metode za minimizaciju logičkih funkcija imaju vrijeme izvršavanja koje je *eksponencijalna funkcija* broja promjenljivih koje se pojavljuju u funkciji koju treba minimizirati, i da su izgledi da će ikada biti otkriven neki bitno efikasniji postupak veoma mali (bolje rečeno, vjeruje se da bitno efikasniji postupci zapravo i ne postoje). Stoga se sve poznate metode za minimizaciju mogu primjenjivati samo za slučajeve u kojima se javlja relativno mali broj promjenljivih. U praksi to znači da se ručna minimizacija (tj. minimizacija bez pomoći računara) može izvršiti za funkcije sa do desetak promjenljivih (i to uz veliki napor), dok se uz pomoć računara može izvršiti minimizacija funkcija koji sadrže maksimalno 15–20 promjenljivih (zavisno od brzine računara). Potpuna minimizacija funkcija od pedesetak promjenljivih ne može se izvršiti za vrijeme kraće od starosti svemira čak i ukoliko se angažiraju istovremeno svi računarski resursi trenutno raspoloživi na Zemlji! Na svu sreću, u praksi se najčešće javljaju slučajevi u kojima treba minimizirati funkcije koje sadrže svega nekoliko promjenljivih, što uz postojeće metode ne predstavlja neki osobit problem.

Poznate metode za minimizaciju funkcija uglavnom su dobro opisane u brojnoj literaturi, i njihov je cilj pretežno nalaženje *minimalnih disjunktivnih* odnosno *konjunktivnih normalnih formi* (MDNF i MKNF). Međutim, u većini raspoložive literature se uglavnom objašnjava samo *postupak* kako se te metode primjenjuju, bez ulaska u njihovu suštinu. Kako takav pristup nosi opasnost da pojedini detalji metoda budu pogrešno protumačeni, na ovom mjestu ćemo, prije izlaganja same ***Veitchove metode***, koja je vjerovatno najpogodnija metoda minimizacije za ručnu upotrebu, izložiti kratke osnove matematičkog aparata na kojima se zasnivaju ova, kao i gotovo sve druge metode za minimizaciju logičkih funkcija. Na prvom mjestu, potrebno je definirati neke dodatne pojmove.

Definicija 8.1:

Svaka logička funkcija φ koja poprima vrijednost nula za sve vrijednosti promjenljivih za koju i logička funkcija F također poprima vrijednost nula, naziva se ***implikanta*** funkcije F (ponekad i ***preduvjet*** funkcije F). Također se kaže i da funkcija φ ***ulazi*** u funkciju F, odnosno da se ***sadrži*** u funkciji F. Alternativno, možemo reći da je funkcija φ implikanta funkcije F ukoliko za sve vrijednosti promjenljivih za koju funkcija φ poprima vrijednost 1, funkcija F također poprima vrijednost 1.

Naziv *implikanta* odnosno *preduvjet* potiče od činjenice da je funkcija φ implikanta funkcije F ako i samo ako je implikacija $\varphi \Rightarrow F$ *bezuvjetno tačna* (tj. tačna za sve vrijednosti promjenljivih). Također, nije teško provjeriti da je funkcija φ implikanta funkcije F ako i samo ako vrijedi $F \vee \varphi = F$. Bitno je naglasiti da se pojam "sadrži" ne smije pogrešno protumačiti tako da se pomisli da se funkcija AB sadrži u funkciji ABC koja ima jedan član više. Tačno je zapravo obrnuto – funkcija ABC sadrži se u funkciji AB, jer je $AB \vee ABC = AB$ (funkcija AB apsorbuje u sebe funkciju ABC)!

Iz definicije neposredno slijedi da je svaka elementarna konjunkcija neke disjunktivne normalne forme ujedno i njena implikanta. Međutim, neka disjunktivna normalna forma često može imati i kraćih implikanti nego što su njene elementarne konjunkcije. Na primjer, posmatrajmo sljedeću funkciju:

$$Y = ABCD \vee A\overline{B}\,\overline{C}D \vee \overline{A}C\overline{D} \vee \overline{A}\,\overline{B}D \vee \overline{A}\,\overline{B}\,\overline{C}$$

Sve elementarne konjunkcije koje čine ovu funkciju su svakako i njene implikante. Međutim, lako se možemo uvjeriti pisanjem tabele istine da su funkcije $\overline{B}\,\overline{C}D$ i $\overline{A}\,\overline{B}$ također njene implikante. Pri tome implikanta $\overline{B}\,\overline{C}D$ predstavlja dio implikante $A\overline{B}\,\overline{C}D$, dok implikanta $\overline{A}\,\overline{B}$ predstavlja dijelove implikanti $\overline{A}\,\overline{B}D$ i $\overline{A}\,\overline{B}\,\overline{C}$. Zbog toga, implikanta $\overline{B}\,\overline{C}D$ apsorbuje u sebe implikantu $A\overline{B}\,\overline{C}D$, dok

implikanta $\overline{A}\,\overline{B}$ apsorbuje u sebe implikante $\overline{A}\,\overline{B}D$ i $\overline{A}\,\overline{B}\,\overline{C}$. Kako za svaku implikantu φ funkcije F vrijedi $F \vee \varphi = F$, možemo pisati:

$$\begin{aligned} Y &= ABCD \vee A\overline{B}\,\overline{C}D \vee \overline{A}C\overline{D} \vee \overline{A}\,\overline{B}D \vee \overline{A}\,\overline{B}\,\overline{C} = \\ &= ABCD \vee A\overline{B}\,\overline{C}D \vee \overline{A}C\overline{D} \vee \overline{A}\,\overline{B}D \vee \overline{A}\,\overline{B}\,\overline{C} \vee \overline{B}\,\overline{C}D \vee \overline{A}\,\overline{B} \vee \overline{A}\,\overline{B} = \\ &= ABCD \vee (A\overline{B}\,\overline{C}D \vee \overline{B}\,\overline{C}D) \vee \overline{A}C\overline{D} \vee (\overline{A}\,\overline{B}D \vee \overline{A}\,\overline{B}) \vee (\overline{A}\,\overline{B}\,\overline{C} \vee \overline{A}\,\overline{B}) = \\ &= ABCD \vee \overline{B}\,\overline{C}D \vee \overline{A}C\overline{D} \vee \overline{A}\,\overline{B} \vee \overline{A}\,\overline{B} = \\ &= ABCD \vee \overline{B}\,\overline{C}D \vee \overline{A}C\overline{D} \vee \overline{A}\,\overline{B} \end{aligned}$$

Vidimo da smo dobili prostiju funkciju nego što je polazna funkcija (slučajno je u ovom primjeru dobijena funkcija upravo njena minimalna disjunktivna normalna forma). Očigledno su za proces minimizacije veoma značajne one implikante koje mogu apsorbovati u sebe druge implikante, čime se funkcija svakako skraćuje. Zbog toga je značajna sljedeća definicija:

Definicija 8.2:

Za implikantu φ logičke funkcije F koja ima oblik elementarne konjunkcije kažemo da je ***prosta implikanta*** ukoliko niti jedan njen dio nije također implikanta funkcije F (drugim riječima, ukoliko ne postoji druga kraća implikanta koja je može apsorbovati u sebe).

Na primjer, za prethodnu funkciju, implikanta $A\overline{B}\,\overline{C}D$ nije prosta implikanta, jer je njen dio $\overline{B}\,\overline{C}D$ također implikanta ove funkcije. S druge strane, neposrednim provjeravanjem se može provjeriti da ni jedan dio implikanti $ABCD$, $\overline{B}\,\overline{C}D$, $\overline{A}C\overline{D}$ i $\overline{A}\,\overline{B}$ nije implikanta date funkcije, stoga su navedene implikante ujedno i proste implikante.

Veoma je lako pokazati da sve elementarne konjunkcije koje čine neku MDNF *moraju biti proste implikante*. Naime, kada neka elementarna konjunkcija φ neke funkcije F u MDNF zapisu ne bi bila prosta implikanta, tada bi se ona mogla apsorbovati u neku drugu kraću implikantu φ_1, pa bi ukoliko primijenimo jednakost $F = F \vee \varphi_1$, nakon apsorpcije $\varphi \vee \varphi_1 = \varphi$ dobili novu funkciju koja je ekvivalentna funkciji F, a kraća je od nje, što je u protivurječnosti sa pretpostavkom da je funkcija F u minimalnoj disjunktivnoj normalnoj formi. Također je lako pokazati da je disjunkcija *svih prostih implikanti* neke funkcije F ekvivalentna polaznoj funkciji. Naime, svaka prosta implikanta je ili minterma ili neki njen dio, tako da će razvoj disjunkcije svih prostih implikanti u savršenu disjunktivnu formu dati upravo SDNF polazne funkcije. Disjunkcija svih prostih implikanti ponekad se naziva i ***skraćena disjunktivna normalna forma*** polazne funkcije. Međutim, disjunkcija *svih* prostih implikanti neke funkcije *ne mora* nužno biti njena MDNF (teoretski se može desiti da skraćena disjunktivna normalna forma bude duža čak i od savršene disjunktivne normalne forme, jer kod funkcija koje ovise od mnogo promjenljivih prostih implikanti može biti više nego mintermi). Posmatrajmo, na primjer, sljedeću funkciju:

$$Y = AB \vee BC \vee \overline{A}C$$

Mada se lako možemo uvjeriti da su sve tri elementarne konjunkcije koje čine ovu funkciju proste implikante, ova funkcija ipak nije MDNF, što možemo pokazati na sljedeći način:

$$Y = AB \vee BC \vee \overline{A}C = AB \vee (A \vee \overline{A})BC \vee \overline{A}C = AB \vee ABC \vee \overline{A}BC \vee \overline{A}C = AB \vee \overline{A}C$$

Drugim riječima, prosta implikanta BC može se jednostavno izbaciti iz funkcije, a da se vrijednost funkcije ne promijeni, što nije očigledno na prvi pogled (usput, upravo demonstrirana činjenica koja tvrdi da je $XY \vee YZ \vee \overline{X}Z = XY \vee \overline{X}Z$, i koju smo do sada više puta koristili u primjerima i zadacima iz Poglavlja 6, poznata je u logičkoj algebri pod imenom ***teorema o konsenzusu***). Zbog toga je pogodno uvesti sljedeću definiciju:

Definicija 8.2:

Logičke funkcije iz kojih se ne može izbaciti ni jedan član (ili, općenitije, niti jedan njihov dio) a da se njihova vrijednost ne promijeni, nazivaju se ***nesvodljive logičke funkcije***.

Jasno je da svaka MDNF mora biti nesvodljiva, jer u suprotnom ne bi bila minimalna. Ipak, svaka nesvodljiva DNF sastavljena od prostih implikanti nije nužno i MDNF. Na primjer, posmatrajmo sljedeće dvije funkcije:

$$Y_1 = AC\overline{D} \vee \overline{A}\,\overline{C}D \vee A\overline{B}\,\overline{D} \vee \overline{B}\,\overline{C}D$$

$$Y_2 = AC\overline{D} \vee \overline{A}\,\overline{C}D \vee A\overline{B}\,\overline{C}$$

Obje funkcije se sastoje samo od prostih implikanti, i obje su nesvodljive, što nije teško provjeriti. Ipak, ove dvije funkcije su *ekvivalentne*, a funkcija Y_2 je kraća! Međutim, ono što je izvjesno je da *najkraća* od svih mogućih nesvodljivih disjunktivnih formi upravo minimalna disjunktivna normalna forma (MDNF ne mora biti jedinstvena ukoliko postoji više najkraćih nesvodljivih DNF). Na osnovu svega do sada rečenog, nazire se jedan sistematičan put koji vodi ka nalaženju MDNF, koji možemo nazvati ***generalni algoritam za minimizaciju logičkih funkcija***, a koji možemo iskazati kroz sljedeća dva koraka:

1. ***Pronaći sve proste implikante zadate funkcije***;
2. ***Iz nađenog skupa prostih implikanti izabrati što je god moguće manji broj što je god moguće kraćih prostih implikanti, čija disjunkcija tvori zadanu funkciju***.

Različiti konkretni metodi za minimizaciju razlikuju se po tome kako se konkretno izvode koraci 1. i 2. u gore prikazanom algoritmu (kod Veitchove metode koju ćemo nešto kasnije detaljno razmotriti, ovi koraci se, na izvjestan način, izvode uporedo).

Razmotrimo prvo kako bi se principijelno mogao izvesti korak 1. Pored operacije apsorbovanja, operacija koja najviše doprinosi eventualnom skraćenju logičkih funkcija je operacija *sažimanja*, koja se može iskazati kroz pravilo $XY \vee X\overline{Y} = X$. Međutim, treba imati u vidu da se često više različitih parova članova mogu na razne sažimati jedan sa drugim. Sljedeći primjer prikazuje dva različita sažimanja izvršena nad istom funkcijom, ali primijenjena na dva različita para članova:

$$Y = AB\overline{C} \vee ABC \vee A\overline{B}C = AB \vee A\overline{B}C$$

$$Y = AB\overline{C} \vee ABC \vee A\overline{B}C = AB\overline{C} \vee AC$$

U prvom slučaju smo izvršili sažimanje prvog sa drugim članom po promjenljivoj C, dok smo u drugom slučaju izvršili sažimanje drugog sa trećim članom po promjenljivoj B. U oba slučaja smo kao rezultat dobili funkcije u kojima dalja sažimanja nisu neposredno moguća. Međutim, da smo odmah na početku udvostručili član ABC koji se može sažimati na dva različita načina, mogli bismo izvršiti oba sažimanja, i tako dobiti još jednostavniju funkciju:

$$Y = AB\overline{C} \vee ABC \vee A\overline{B}C = AB\overline{C} \vee ABC \vee ABC \vee A\overline{B}C = AB \vee AC$$

Odavde direktno slijedi da je prilikom vršenja sažimanja moguće prvo uočiti sve parove članova koji se mogu sažimati, a zatim sve takve parove prosto zamijeniti rezultatima njihovog sažimanja. Sažimanja mogu biti naročito efikasna ako se krene od savršene disjunktivne normalne forme, jer u njoj svaki član sadrži sve promjenljive, pa broj parova članova koji se mogu sažimati može biti jako veliki. Posmatrajmo, na primjer, sljedeću funkciju zadanu u SDNF:

$$Y = AB\overline{C}\,\overline{D} \vee \overline{A}B\overline{C}D \vee A\overline{B}\,\overline{C}\,\overline{D} \vee A\overline{B}CD \vee A\overline{B}C\overline{D} \vee A\overline{B}\,\overline{C}D \vee ABCD$$

Prvi član u ovoj funkciji može se sažimati samo sa trećim članom (po promjenljivoj B), i kao rezultat tog sažimanja dobija se član (implikanta) $A\overline{C}\,\overline{D}$. Drugi član ne može se sažimati ni sa čim. Treći član može se sažimati (osim sa prvim članom, što je već obavljeno) sa petim (po promjenljivoj C) i šestim (po promjenljivoj D) članom, čime se kao rezultat sažimanja dobijaju članovi $A\overline{B}\,\overline{D}$ i $AB\overline{C}$. Četvrti član se također može sažimati sa petim i šestim članom, čime se dobijaju članovi $A\overline{B}C$ i $A\overline{B}D$, ali se može sažimati i sa sedmim članom, čime se dobija član ACD. Ovim smo iscrpili sva sažimanja koja se mogu obaviti. Slijedi da je polazna funkcija ekvivalentna sljedećoj funkciji (drugi član, koji se nije mogao sažimati ni sa čim je prosto prepisan):

$$Y = A\overline{C}\,\overline{D} \vee \overline{A}B\overline{C}D \vee A\overline{B}\,\overline{D} \vee A\overline{B}\,\overline{C} \vee A\overline{B}C \vee A\overline{B}D \vee ACD$$

U novodobivenoj funkciji je također moguće izvršiti neka sažimanja. Prvi i drugi član ne mogu se sažimati ni sa čim, ali se zato treći član može sažimati sa šestim, čime se dobija član $A\overline{B}$. Isti rezultat dobija se i nakon sažimanja četvrtog i petog člana, čime su iscrpljena sva moguća sažimanja u ovoj funkciji. Slijedi da se polazna funkcija može predstaviti i u sljedećem obliku:

$$Y = A\overline{C}\,\overline{D} \vee \overline{A}B\overline{C}D \vee A\overline{B} \vee ACD$$

Ovim su isrpljene sve mogućnosti sažimanja. Intuitivno se može naslutiti da se, nakon što se iscrpe sve mogućnosti sažimanja, dobija upravo forma u kojoj su svi članovi zapravo *proste implikante*. Ovo je precizno formulirano kroz tzv. ***Quineovu teoremu***, koju navodimo bez dokaza:

Teorema 8.1 (W. V. Quine):

Ukoliko se na funkciju zadanu u savršenoj disjunktivnoj formi primijeni operacija sažimanja na sve parove članova koji se mogu sažimati, i ukoliko se isti postupak ponavlja na novodobijene funkcije sve dok se ne iscrpe sve mogućnosti za sažimanje, kao rezultat se dobija skraćena disjunktivna normalna forma, tj. forma koja sadrži samo proste implikante, i to sve proste implikante.

Ovim je pokazano kako se može provesti korak 1. generalnog algoritma za minimizaciju logičkih funkcija. Postupak je u potpunosti šabloniziran i nedvosmislen, tako da ne stvara nikakve principijelne poteškoće (ako zanemarimo činjenicu da je pri ručnom radu veoma lako previdjeti neki par članova koji se mogu sažimati, i da sam postupak može biti dugotrajan). Svi metodi za minimizaciju logičkih funkcija zasnivaju se na gore opisanom postupku, mada je on u nekim metodama, kao što je Veitchova metoda, zamaskiran skoro do neprepoznatljivosti, i znatno prilagođeniji za ručno računanje.

Korak 2. je mnogo delikatniji, i teško ga je u potpunosti formalizirati. Zapravo, već korak 2. sam za sebe predstavlja NP kompletan problem, poznat pod nazivom ***problem optimalnog pokrivanja***. Podsjetimo se da je u ovom koraku potrebno odabrati što manji skup što manjih prostih implikanti čija disjunkcija daje polaznu funkciju. Najpregledniji način da se to uradi je formiranje tzv. ***tablice pokrivanja*** (ili ***implikantne matrice***), čiji redovi odgovaraju nađenim prostim implikantama, a kolone mintermama polazne savršene disjunktivne normalne forme funkcije. U presjeku *i*-tog reda i *j*-te kolone upisuje se znak "+" ukoliko *i*-ta implikanta predstavlja dio *j*-te minterme.Na primjer, sljedeća tablica predstavlja tablicu pokrivanja za funkciju koju smo upravo odredili proste implikante:

Proste implikante \ *Minterme*	$AB\overline{C}\,\overline{D}$	$\overline{A}B\overline{C}D$	$A\overline{B}\,\overline{C}\,\overline{D}$	$A\overline{B}CD$	$A\overline{B}C\overline{D}$	$A\overline{B}\,\overline{C}D$	$ABCD$
$A\overline{C}\,\overline{D}$	+		+				
$\overline{A}B\overline{C}D$		+					
$A\overline{B}$			+	+	+	+	
ACD				+			+

Tablica pokrivanja omogućava da relativno lako odredimo koje se implikante mogu izbaciti. Naime, mi moramo izabrati implikante čija disjunkcija daje polaznu funkciju, tj. koja daje istu funkciju kao i disjunkcija svih mintermi. Kako znak "+" zapravo označava koja je minterma sadržana u kojoj implikanti, neophodno je odabrati takav skup implikanti kod kojeg će u svakoj koloni tablice pokrivanja biti barem jedan znak "+". U slučaju kada je broj implikanti veliki, ne postoji generalna sugestija kako odabrati *najmanji* takav skup, pa u takvim slučajevima pomaže ili intuicija, ili metod isprobavanja različitih varijanti i izbor najbolje varijante. Ipak, u svakom slučaju, postupak treba započeti posmatranjem kolona u kojima se nalazi samo jedan znak "+", kao što su prva, druga, peta, šesta i sedma kolona u navedenom primjeru. Odgovarajuće implikante nazivaju se ***esencijalne implikante*** i njih svakako moramo zadržati. Tek kada razmotrimo sve esencijalne implikante, prelazimo na razmatranje ostalih (neesencijalnih) implikanti, i od njih biramo minimalan skup implikanti koji pokriva znacima "+" preostale kolone tablice pokrivanja.

U razmotrenom primjeru lako uviđamo da su u sve nađene proste implikante esencijalne. Naime, prvu implikantu moramo zadržati, jer jedino ona unosi znak "+" u prvu kolonu. Iz istog razloga moramo zadržati drugu, treću i četvrtu implikantu, jer jedino one unose znakove "+" u drugu, petu, šestu i sedmu kolonu. Kao zaključak slijedi da se niti jedna implikanta ne može izbaciti, odnosno da nađena disjunktivna forma ujedno predstavlja i traženu MDNF.

Opisani postupak predstavlja najstariji poznati sistematičan (mada i vrlo glomazan) postupak za minimizaciju logičkih funkcija, i poznat je pod nazivom ***Quineov algoritam***. Ovaj algoritam traži da se obavezno krene od funkcije predstavljene u savršenoj disjunktivnoj formi, što je pogodno za slučaj kada je funkcija zadana tabelarno (u suprotnom je potrebno prethodno funkciju ručno svesti na SDNF). Quineov algoritam ćemo detaljno ilustrirati na još jednom primjeru.

Primjer 8.1:

- Quineovim algoritmom naći minimalnu disjunktivnu normalnu formu funkcije predstavljene sljedećom tablicom istine:

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

Početni korak je formiranje SDNF oblika funkcije, što nije teško uraditi:

$$Y = \overline{A}\,\overline{B}C\overline{D} \vee \overline{A}B\overline{C}\,\overline{D} \vee \overline{A}BCD \vee A\overline{B}\,\overline{C}\,\overline{D} \vee A\overline{B}\,\overline{C}D \vee A\overline{B}CD \vee AB\overline{C}\,\overline{D} \vee ABC\overline{D} \vee ABCD$$

Sada je potrebno pronaći sve moguće parove članova koji se mogu sažimati. Ako članove u ovoj funkciji označimo rednim brojevima od 1 do 9, tada možemo uočiti da su parovi članova koji se mogu sažimati parovi (2, 7), (3, 9), (4, 5), (4, 7), (5, 6), (6, 9), (7, 8) i (8, 9). Kao rezultat ovih sažimanja dobijaju se redom implikante $B\overline{C}\,\overline{D}$, BCD, $A\overline{B}\overline{C}$, $A\overline{C}\,\overline{D}$, $A\overline{B}D$, ACD, $AB\overline{D}$ i ABC. Slijedi da se polazna funkcija može napisati u sljedećem obliku (član 1. koji se ni sa čim ne može sažimati je prepisan):

$$Y = \overline{A}\,\overline{B}CD \vee B\overline{C}\,\overline{D} \vee BCD \vee A\overline{B}\,\overline{C} \vee A\overline{C}\,\overline{D} \vee A\overline{B}D \vee ACD \vee AB\overline{D} \vee ABC$$

Ukoliko razmotrimo dobivenu funkciju, vidjećemo da se ni jedan član ne može sažimati sa nekim drugim članom, što znači da smo odredili sve proste implikante. Sada prelazimo na formiranje tablice pokrivanja, odnosno implikantne matrice:

Proste Implikante	*Minterme*								
	$\overline{A}\,\overline{B}CD$	$\overline{A}B\overline{C}\,\overline{D}$	$\overline{A}BCD$	$A\overline{B}\,\overline{C}\,\overline{D}$	$A\overline{B}\,\overline{C}D$	$A\overline{B}CD$	$AB\overline{C}\,\overline{D}$	$AB\overline{C}D$	$ABCD$
$I_1 = \overline{A}\,\overline{B}CD$	+								
$I_2 = B\overline{C}\,\overline{D}$		+					+		
$I_3 = BCD$			+						+
$I_4 = A\overline{B}\,\overline{C}$				+	+				
$I_5 = A\overline{C}\,\overline{D}$				+			+		
$I_6 = A\overline{B}D$					+	+			
$I_7 = ACD$						+			+
$I_8 = AB\overline{D}$							+	+	
$I_9 = ABC$								+	+

Ovdje smo nađene proste implikante označili oznakama od I_1 do I_9 radi lakšeg snalaženja. Odavde ovmah vidimo da su implikante I_1, I_2 i I_3 esencijalne, tako da ih svakako moramo zadržati, jer bi njihovim izbacivanjem prva, druga odnosno treća kolona tablice pokrivanja ostale nepokrivene (svaka od njih sadrži samo po jedan znak "+"). Sada je od preostalih prostih implikanti (koje nisu esencijalne) potrebno izabrati što manji skup tako da budu pokrivene i preostale kolone tabele. Redove koji odgovaraju esencijalnim implikantama, kao i kolone koje su već pokrivene esencijalim implikantama (kolone 1, 2, 3, 7 i 9) najbolje je precrtati da nam ne odvlače pažnju. Tako dobijamo sljedeću tablicu:

	$A\overline{B}\,\overline{C}\,\overline{D}$	$A\overline{B}\,\overline{C}D$	$A\overline{B}CD$	$ABC\overline{D}$
$I_4 = A\overline{B}\,\overline{C}$	+	+		
$I_5 = A\overline{C}\,\overline{D}$	+			
$I_6 = A\overline{B}D$		+	+	
$I_7 = ACD$			+	
$I_8 = AB\overline{D}$				+
$I_9 = ABC$				+

Da bi dobivena funkcija bila nesvodljiva, nikada ne smijemo uzeti implikantu koja pokriva kolone koje su već pokrivene drugim implikantama. Postoji jedan jednostavan postupak koji uvijek garantira da ćemo pronaći *nesvodljivu* formu. Naime, možemo analizirati redom jednu po jednu implikantu, i eliminirati je iz razmatranja ukoliko ustanovimo da ona prekriva one redove koji su već pokriveni ostalim implikantama. Na primjer, krenemo li od implikante I_4, vidimo da je možemo eliminirati iz razmatranja, jer ona ne pokriva niti jednu kolonu koja nije pokrivena ostalim implikantama (preciznije, implikantama I_5 i I_6). Nakon njene eliminacije, implikantu I_5 ne smijemo izbaciti, jer bi u suprotnom prva kolona ostala nepokrivena. Također, ne smijemo izbaciti ni implikantu I_6, jer u suprotnom druga kolona ostaje nepokrivena. Sada, implikantu I_7 možemo izbaciti, jer ona ne pokriva niti jednu kolonu

koja nije pokrivena preostalim implikantama I_5, I_6, I_8 i I_9. Slično, možemo izbaciti i implikantu I_8, jer implikante I_5, I_6 i I_9 prekrivaju sve kolone. Nakon ovih izbacivanja, implikantu I_9 moramo zadržati, da pokrijemo četvrtu kolonu. Tako smo dobili nesvodljivu formu koju čine upravo implikante I_5, I_6 i I_9.

Izbacivanje implikanti smo mogli izvršiti i drugim poretkom, čime bismo mogli dobiti i druge nesvodljive forme. Za funkciju iz ovog primjera moguće je pronaći čak šest nesvodljivih pokrivanja, koja su sva iste dužine (tri implikante, svaka sa po tri promjenljive). To su, redom, pokrivanja sastavljena od trojki implikanti: (I_4, I_6, I_8), (I_4, I_6, I_9), (I_4, I_7, I_8), (I_4, I_7, I_9), (I_5, I_6, I_8) i (I_5, I_6, I_9). Slijedi da su sve odgovarajuće nesvodljive forme ujedno i minimalne, tako da tražena funkcija ima čak 6 različitih ekvivalentnih minimalnih disjunktivnih normalnih formi. Na primjer, uzmemo li implikante I_5, I_6 i I_9, dobijamo sljedeću MDNF (ne smijemo zaboraviti uključiti i esencijalne implikante I_1, I_2 i I_3):

$$Y = \overline{A}\,\overline{B}C\overline{D} \vee B\overline{C}\,\overline{D} \vee BCD \vee A\overline{C}\,\overline{D} \vee A\overline{B}D \vee ABC$$

Interesantno je napomenuti da postoji i jedan čisto algebarski (mada najčešće izuzetno glomazan) postupak za nalaženje *svih nesvodljivih pokrivanja*. Za tu svrhu, posmatramo sve proste implikante *kao neovisne promjenljive*, i formiramo pomoćnu funkciju Φ u vidu konjuktivne forme, u kojoj je svaka elementarna disjunkcija formirana od prostih implikanti koje pokrivaju po jedan red implikantne matrice. Konkretno, za prethodnu tablicu, funkcija Φ glasi ovako:

$$\Phi = (I_4 \vee I_5)\,(I_4 \vee I_6)\,(I_6 \vee I_7)\,(I_8 \vee I_9)$$

Zatim, funkciju Φ treba prevesti u disjunktivnu normalnu formu, primjenom pravila distribucije i apsorbovanja. U konkretnom primjeru, nakon nešto dužeg računa, dobija se

$$\Phi = I_4 I_6 I_8 \vee I_4 I_6 I_9 \vee I_4 I_7 I_8 \vee I_4 I_7 I_9 \vee I_5 I_6 I_8 \vee I_5 I_6 I_9$$

Svaka elementarna konjukcija u dobijenoj disjunktivnoj normalnoj formi odgovara jednom nesvodljivom pokrivanju, u šta se možemo lako uvjeriti. Ovaj postupak nije teško dokazati, što se ostavlja čitatelju ili čitateljici za vježbu.

Bitno je napomenuti da ne postoji nikakva sugestija koja *apriori* ukazuje kojim redoslijedom treba eliminirati suvišne implikante da bi se dobila minimalna forma. Situacija može bitno ovisiti od poretka u kojem eliminiramo implikante, a samim tim i od načina kako su implikante sortirane. Pretpostavimo, na primjer, da smo dobili sljedeću matricu prekrivanja prilikom minimizacije neke logičke funkcije:

	$\overline{A}\,\overline{B}\,\overline{C}$	$\overline{A}B\overline{C}$	$\overline{A}BC$	$A\overline{B}\,\overline{C}$	$A\overline{B}C$	ABC
$I_1 = \overline{A}\,\overline{C}$	+	+				
$I_2 = \overline{B}\,\overline{C}$	+			+		
$I_3 = \overline{A}B$		+	+			
$I_4 = BC$			+			+
$I_5 = A\overline{B}$				+	+	
$I_6 = AC$					+	+

Eliminiramo li na početku implikantu I_1, vidimo da moramo zadržati implikante I_2 i I_3. Ukoliko sada eliminiramo implikantu I_4, vidjećemo da možemo eliminirati implikantu I_5 a zadržati implikantu I_6, čime dobijamo nesvodljivu formu sastavljenu od implikanti I_2, I_3 i I_6 (koja je, zapravo, i minimalna). Međutim, ukoliko nakon eliminacije implikante I_1 odmah eliminiramo implikantu I_6 (jer ni ona ne pokriva ništa što nije pokriveno ostalim implikantama), došli bismo do zaključka da moramo sačuvati sve preostale implikante I_2, I_3, I_4 i I_5, čime bismo dobili nesvodljivu formu koja *nije minimalna*!

Jedina garancija za dobijanje zaista minimalne forme je formiranje *svih mogućih* nesvodljivih pokrivanja, i izbor najkraćeg od njih. Sva moguća nesvodljiva pokrivanja moguće je odrediti recimo pomoću maločas opisane algebarske metode. Na primjer, za prethodni primjer imamo

$$\Phi = (I_1 \vee I_2)\,(I_1 \vee I_3)\,(I_3 \vee I_4)\,(I_2 \vee I_5)\,(I_5 \vee I_6)\,(I_4 \vee I_6) = \ldots =$$
$$= I_1 I_3 I_5 I_6 \vee I_1 I_2 I_4 I_6 \vee I_2 I_3 I_4 I_5 \vee I_1 I_4 I_5 \vee I_2 I_3 I_6$$

Odavde vidimo da postoji ukupno 5 nesvodljivih pokrivanja, od kojih nije teško odabrati najmanja pokrivanja – u ovom slučaju to su pokrivanja (I_1, I_4, I_5) ili (I_2, I_3, I_6). Međutim, kako u općem slučaju takvih pokrivanja može biti zaista veoma mnogo (pogotovo kada funkcija zavisi od mnogo promjenljivih), koriste se izvjesni intuitivni postupci. Na primjer, dobra je ideja uvijek birati implikante koje ćemo zadržati takvim redoslijedom tako da svaka nova implikanta pokriva što je god moguće više do tada nepokrivenih kolona. Recimo, u gornjem primjeru, izaberemo li prvo implikantu I_1, sljedeća implikanta koju bi trebalo uzeti je I_4, jer ona prekriva dvije nove kolone, dok implikante I_2 i I_3 pokrivaju samo jednu kolonu koja već nije prekrivena implikantom I_1. Nakon izbora implikanti I_1 i I_4, izbor implikante I_5 koja pokriva preostale dvije kolone je očigledan, čime smo dobili minimalnu nesvodljivu formu, koju čine implikante I_1, I_4 i I_5. Ipak, ni takva strategija ne mora uvijek garantirati optimalnost.

Quineov algoritam može se iskoristiti i za nalaženje *minimalne konjunktivne normalne forme* zadane logičke funkcije. Za tu svrhu, prvo je potrebno u savršenoj disjunktivnoj normalnoj formi predstaviti *negaciju zadane logičke funkcije*. To nije teško učiniti, s obzirom da se SDNF negacije neke logičke funkcije sastoji od onih i samo onih mintermi koje se ne nalaze u SDNF same funkcije. Nakon toga se na tako formiranu SDNF primjenjuje Quineov algoritam, čime dobijamo MDNF negacije logičke funkcije. Na kraju, negacijom dobijene MDNF dobijamo MKNF polazne funkcije.

Primjer 8.2:

- Quineovim algoritmom naći minimalnu konjunktivnu formu funkcije iz Primjera 8.1.

Prvo je potrebno naći SDNF negacije ove funkcije, što je lako uraditi direktno iz tabele, zamjenom uloga nula i jedinica u koloni koja predstavlja vrijednosti funkcije:

$$\overline{Y} = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} \vee \overline{A}\,\overline{B}\,\overline{C}D \vee \overline{A}\,\overline{B}CD \vee \overline{A}B\overline{C}D \vee \overline{A}BC\overline{D} \vee A\overline{B}C\overline{D} \vee AB\overline{C}D$$

Parovi mintermi koje se mogu sažimati su $(1, 2)$, $(2, 3)$, $(2, 4)$ i $(4, 7)$, dok se peta i šesta minterma ne mogu sažimati ni sa čim, čime nakon prve etape dolazimo do sljedećeg prikaza:

$$\overline{Y} = \overline{A}\,\overline{B}\,\overline{C} \vee \overline{A}\,\overline{B}D \vee \overline{A}\,\overline{C}D \vee B\overline{C}D \vee \overline{A}BC\overline{D} \vee A\overline{B}C\overline{D}$$

Lako se vidi da su u ovom prikazu iscrpljene sve dalje mogućnosti sažimanja, tako da dobijeni prikaz predstavlja skraćenu disjunktivnu normalnu formu negacije zadane logičke funkcije. Sada prelazimo na formiranje implikantne matrice:

Minterme

Proste implikante	$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$	$\overline{A}\,\overline{B}\,\overline{C}D$	$\overline{A}\,\overline{B}CD$	$\overline{A}B\overline{C}D$	$\overline{A}BC\overline{D}$	$A\overline{B}C\overline{D}$	$AB\overline{C}D$
$I_1 = \overline{A}\,\overline{B}\,\overline{C}$	+	+					
$I_2 = \overline{A}\,\overline{B}D$		+	+				
$I_3 = \overline{A}\,\overline{C}D$		+		+			
$I_4 = B\overline{C}D$				+			+
$I_5 = \overline{A}BC\overline{D}$					+		
$I_6 = A\overline{B}C\overline{D}$						+	

Na osnovu implikantne matrice vidimo da su implikante I_1, I_2, I_4, I_5 i I_6 esencijalne. Međutim, ovih 5 implikanti ujedno prekrivaju sve kolone implikantne matrice, tako da implikantu I_3 možemo izbaciti. Na taj način dolazimo do jedinstvene nesvodljive disjunktivne normalne forme negacije zadane logičke funkcije, koja je, prema tome, ujedno i njena MDNF:

$$\overline{Y} = \overline{A}\,\overline{B}\,\overline{C} \vee \overline{A}\,\overline{B}\,D \vee B\overline{C}D \vee \overline{A}BC\overline{D} \vee A\overline{B}C\overline{D}$$

Konačno, MKNF polazne funkcije dobijamo negacijom dobijenog izraza:

$$Y = (A \vee B \vee C)(A \vee B \vee \overline{D})(\overline{B} \vee C \vee \overline{D})(A \vee \overline{B} \vee \overline{C} \vee D)(\overline{A} \vee B \vee \overline{C} \vee D)$$

Ovim je završen opis Quineovog algoritma za minimizaciju logičkih funkcija. Ovaj algoritam je posve jasno formuliran, i stoga je pogodan za programiranje na računaru. E. J. McCluskey je predložio modifikaciju ovog algoritma, poznatu pod nazivom ***Quine–McCluskeyjev algoritam***, u kojem se u prvom koraku članovi koji se sažimaju na pogodan način numeriraju binarnim brojevima, čime se znatno dobija na preglednosti, i olakšava implementacija algoritma prilikom programiranja na računaru (drugi korak, tj. traženje optimalnog pokrivanja, isti je kao kod običnog Quineovog algoritma). S obzirom da ovaj algoritam u suštini ne donosi ništa novo u odnosu na izvorni Quineov algoritam, osim olakšanog predstavljanja implikanti u računarskom programu i nešto većoj preglednosti pri ručnom računanju (u smislu da se lakše uočavaju članovi koji se mogu sažimati), njegov opis nećemo navoditi. Zainteresirani čitatelji i čitateljice upućuju se na širu literaturu u kojoj je ova problematika detaljno obrađena.

Quineov odnosno Quine–McCluskyjev algoritam je, pored tzv. ***Rothovog algoritma***, jedan od najpodesnijih algoritama za minimizaciju logičkih funkcija za programiranje na računaru. Međutim, svi spomenuti algoritmi su prilično glomazni i stoga nepodesni za ručni rad, tako da je već za npr. šest promjenljivih gotovo nemoguće ručno provesti cijeli postupak, a da se pri tome nigdje ne napravi greška. Pored toga, dužina postupka koji treba provesti drastično raste (po eksponencijalnom zakonu) sa porastom broja promjenljivih. Naime, funkcija od n promjenljivih može imati do 2^n mintermi u savršenoj disjunktivnoj normalnoj formi, dok broj prostih implikanti može ići do oko $3^n/n$. Slijedi da za funkciju od 10 promjenljivih tabela pokrivanja može biti i formata recimo 5000×1000, što je prilično mnogo i za računar (o ručnom radu nema ni govora). Za slučaj funkcija sa preko 15–20 promjenljivih, čak ni upotreba računara ne dolazi u obzir.

Da bi se omogućila minimizacija funkcija sa još većim brojem promjenljivih, razvijeni su i razni heuristički algoritmi, kao što je npr. ***Espresso algoritam*** (koji ovdje nećemo opisivati), koji može uspješno raditi i sa većim brojem promjenljivih, ali *ne garantira* pronalaženje optimalnog rješenja, već samo rješenja koje je "dovoljno dobro" (tj. koje nije "previše loše"). S druge strane, kako se u praktičnim situacijama uglavnom susreće potreba za minimizacijom funkcija koje ovise od malog broja promjenljivih (recimo do 6), razvijene su i metode koje su znatno prilagođenije za ručni rad i koje znatno brže vode rješenju nego Quineov algoritam (mada se, u suštini, radi samo o prerušenim verzijama Quineovog algoritma). Jednoj od najefikasnijih takvih metoda posvećeno je sljedeće poglavlje.

(?) Pitanja i zadaci

8.1 Definirajte pojmove implikante i proste implikante.

8.2 Pokažite da je φ implikanta funkcije F ako i samo ako je implikacija $\varphi \Rightarrow F$ bezuvjetno tačna, tj. ako i samo ako vrijedi $\varphi \Rightarrow F = 1$.

8.3 Pokažite da je φ implikanta funkcije F ako i samo ako vrijedi $F \vee \varphi = F$.

8.4 Provjerite da su za logičku funkciju $Y = ABCD \vee A\overline{B}\,\overline{C}D \vee \overline{A}C\overline{D} \vee \overline{A}\,\overline{B}D \vee \overline{A}\,\overline{B}\,\overline{C}$ elementarne konjukcije $ABCD$, $\overline{B}\,\overline{C}D$, $\overline{A}C\overline{D}$ i $\overline{A}\,\overline{B}$ njene proste implikante. Provjeru izvršite formiranjem tablica istine.

8.5 Definirajte pojam skraćene disjunktivne normalne forme.

8.6 Definirajte pojam nesvodljivih logičkih funkcija.

8.7 Ispitajte da li su logičke funkcije $Y = AC\overline{D} \vee \overline{A}\,\overline{C}D \vee A\overline{B}\,\overline{D} \vee \overline{B}\,\overline{C}D$ i $Y = AC\overline{D} \vee \overline{A}\,\overline{C}D \vee A\overline{B}\,\overline{C}$ nesvodljive.

8.8 Ispitajte da li je logička funkcija $Y = ABC \vee \overline{A}D \vee BD \vee \overline{C}D$ nesvodljiva.

8.9 Opišite generanu tehniku na kojoj se zasnivaju svi sistematični postupci minimizacije logičkih funkcija.

8.10 Formulirajte precizno Quineovu teoremu.

8.11* Dokažite Quineovu teoremu.

8.12 Data je logička funkcija
$Y = \overline{A}\,\overline{B}CD \vee \overline{A}B\overline{C}\,\overline{D} \vee \overline{A}BCD \vee A\overline{B}\,\overline{C}\,\overline{D} \vee A\overline{B}\,\overline{C}D \vee A\overline{B}CD \vee AB\overline{C}\,\overline{D} \vee ABC\overline{D} \vee ABCD$
Koristeći Quineov algoritam, odredite:
a) Sve proste implikante date funkcije
b) Sve nesvodljive disjunktivne normalne forme (NDNF) date funkcije
c) Minimalnu disjunktivnu normalnu formu (MDNF) date funkcije

8.13 Nađite minimalnu konjunktivnu formu funkcije iz zadatka 8.12.

8.14 Pomoću Quineovog algoritma, nađite sve proste implikante, sve nesvodljive disjunktivne normalne forme, kao i minimalnu disjunktivnu normalnu formu za sljedeće logičke funkcije:
a) $Y = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} \vee \overline{A}\,\overline{B}CD \vee \overline{A}\,\overline{B}CD \vee \overline{A}B\overline{C}D \vee A\overline{B}\,\overline{C}\,\overline{D} \vee A\overline{B}CD \vee AB\overline{C}\,\overline{D} \vee ABC\overline{D} \vee ABCD$
b) $Y = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} \vee \overline{A}B\overline{C}\,\overline{D} \vee \overline{A}B\overline{C}D \vee \overline{A}BCD \vee A\overline{B}\,\overline{C}\,\overline{D} \vee AB\overline{C}\,\overline{D} \vee AB\overline{C}D \vee ABCD$

8.15 Pomoću Quineovog algoritma, nađite minimalnu konjunktivku formu za logičku funkciju
$Y = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} \vee \overline{A}B\overline{C}\,\overline{D} \vee \overline{A}B\overline{C}D \vee \overline{A}BCD \vee A\overline{B}\,\overline{C}\,\overline{D} \vee AB\overline{C}\,\overline{D} \vee AB\overline{C}D \vee ABCD$

8.16 Pomoću Quineovog algoritma, nađite sve proste implikante, sve nesvodljive disjunktivne normalne forme, kao i minimalnu disjunktivnu normalnu formu za sljedeće logičke funkcije:
a) $Y = A\overline{B} \vee B\overline{C} \vee \overline{A}\,\overline{B}C$ b) $Y = A\overline{C} \vee \overline{B}C \vee \overline{A}\,\overline{B}$ c) $Y = A\overline{B} \vee A\overline{C} \vee B\overline{C} \vee \overline{A}C \vee \overline{B}C$

8.17 Pomoću Quineovog algoritma, nađite sve proste implikante, sve nesvodljive disjunktivne normalne forme, kao i minimalnu disjunktivnu normalnu formu za logičku funkciju $Y = m_0 \vee m_4 \vee m_5 \vee m_7 \vee m_8 \vee m_{12} \vee m_{13} \vee m_{15}$, gdje se podrazumijeva da sve minterme zavise od promjenljivih A, B, C i D.

8.18 Pomoću Quineovog algoritma, nađite sve proste implikante, sve nesvodljive disjunktivne normalne forme, kao i minimalnu disjunktivnu normalnu formu za sljedeće logičke funkcije, uz pretpostavku da sve minterme zavise od promjenljivih A, B, C i D:

a) $Y = m_0 \vee m_2 \vee m_5 \vee m_7 \vee m_8 \vee m_{10}$

b) $Y = m_0 \vee m_1 \vee m_2 \vee m_5 \vee m_7 \vee m_{12} \vee m_{13} \vee m_{14}$

c) $Y = m_0 \vee m_2 \vee m_3 \vee m_5 \vee m_6 \vee m_7 \vee m_8 \vee m_{10} \vee m_{11} \vee m_{12} \vee m_{14} \vee m_{15}$

8.19 Pomoću Quineovog algoritma, nađite minimalnu konjunktivnu normalnu formu za logičku funkciju $Y = M_4 M_5 M_{10} M_{11} M_{12} M_{13} M_{14}$, gdje se podrazumijeva da sve maksterme zavise od promjenljivih A, B, C i D.

8.20* Pomoću Quineovog algoritma, nađite minimalnu disjunktivnu normalnu formu logičku funkciju $Y = m_0 \vee m_1 \vee m_3 \vee m_7 \vee m_8 \vee m_{11} \vee m_{12} \vee m_{16} \vee m_{20} \vee m_{23} \vee m_{24} \vee m_{27} \vee m_{28} \vee m_{31}$ uz pretpostavku da sve minterme zavise od promjenljivih A, B, C, D i E.

8.21 Provjerite rezultate Primjera 6.8 i 6.9 pomoću Quineovog algoritma.

8.22 Riješite zadatke 6.17 i 6.19 pomoću Quineovog algoritma.

8.23 Riješite zadatak 6.24 pomoću Quineovog algoritma.

8.24 Riješite zadatke 6.28 i 6.30 pomoću Quineovog algoritma.

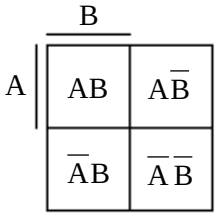
8.25* U jednoj sali sa 5 šaltera rade dva službenika, jedan pripravnik i šef. Svaki šalter je predviđen za različite poslove, a stranke zahtjev za uslugom signaliziraju pritiskom na dugme ispred odgovarajućeg šaltera. Pripravnik poznaje samo poslove koji se obavljaju na prva dva šaltera (ostale poslove ne poznaje). U slučaju da ima veći broj istovremenih zahtijeva, tako da službenici i pripravnik ne mogu da prihvate sve zahtjeve, u pomoć im priskače i šef. Šef zahtijeva da mu se obezbijedi signalizacija koja će javljati kada je neophodno da se i on uključi u rad. Za tu svrhu, svaki pritisak na dugme ispred *i*-tog šaltera uzrokuje signal $S_i = 1$ na jednom od 5 ulaza odgovarajućeg digitalnog sklopa (inače je $S_i = 0$). Signal $Y = 1$ na izlazu iz sklopa aktivira signalizaciju da je šef potreban. Nacrtajte strukturu što je god moguće jednostavnijeg sklopa koji obavlja traženu funkcionalnost.

9. Grafoanalitički pristup minimizaciji (Veitchovi dijagrami)

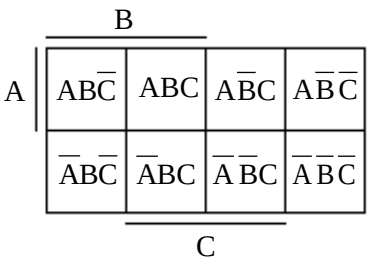
U prethodnom poglavlju smo upoznali Quineovu metodu za minimizaciju logičkih funkcija. Ova metoda, kao i praktično sve druge metode minimizacije, zasniva se na pronalaženju svih prostih implikanti, a zatim pronalaženju što manjeg skupa prostih implikanti čija disjunkcija tvori traženu funkciju. Kako je ova metoda prilično nepodesna za ručni rad, ***E. W. Veitch*** je predložio modifikaciju ove metode za slučaj manjeg broja promjenljivih, koja omogućava da pronalaženje prostih implikanti i njihov optimalni izbor budu gotovo očigledni. Veitchova metoda je *grafoanalitičkog tipa*, i zasniva se na primjeni specijalnih tablica nazvanih ***Veitchovi dijagrami***. U gotovo isto vrijeme, ***M. Karnaugh*** je predložio skoro identičnu metodu, baziranu na tablicama nazvanim ***Karnaughove mape***. Zapravo, Veitchova i Karnaughova metoda su praktično iste metode, a Veitchovi dijagrami i Karnaughove mape razlikuju se samo po rasporedu pojedinih elemenata unutar tablica, dok je princip rada sa njima potpuno isti. Tablice koje ćemo mi koristiti su Veitchovi dijagrami, iako se one u literaturi također često pogrešno imenuju kao Karnaughove mape (pošto je Karnaughov rad bio mnogo bolje publikovan, često se njegovim imenom pogrešno naziva i metoda koja je potekla od Veitcha).

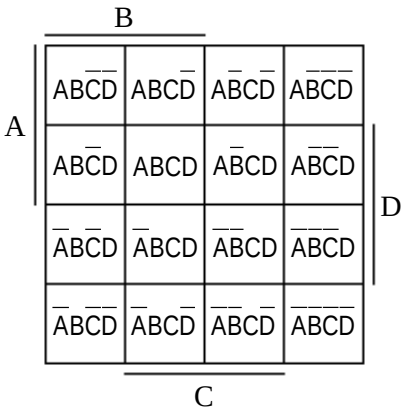
Mana Veitchove (ili Karnaughove) metode je u tome što ju je veoma teško formalizirati tako da nije pogodna za programiranje na računaru (zbog toga nije dobro koristiti naziv Veitchov algoritam), i što je praktično upotrebljiva samo za mali broj promjenljivih (recimo do šest, mada su u doba prije masovne upotrebe računara zabilježeni slučajevi projektanata koji su Veitchovu metodu primjenjivali čak i za slučajeve 10 do 12 promjenljivih, što zaista graniči sa mazohizmom). Mi ćemo se ograničiti na slučaj primjene Veitchove metode za funkcije sa dvije, tri i četiri promjenljive.

Osnovna ideja Veitchove metode za funkcije od n promjenljivih sastoji se u tome da se njihova tablica istine reorganizira na takav način da članovi koji se mogu sažimati budu uočljivi na prvi pogled. Da bi se to postiglo, formira se tablica sa 2^n polja, koja je organizirana tako da svakoj promjenljivoj odgovara tačno jedna polovina tablice, po mogućnosti sastavljena od međusobno susjednih polja, a njenoj negaciji preostala polovina. Pri tome, podjele za različite promjenljive trebaju da budu tako organizirane da svakom polju odgovara u presjeku različita minterma od n promjenljivih. Tako organizirane tablice nazivaju se Veitchovi dijagrami. Na primjer, Veitchov dijagram za dvije promjenljive organiziran je tako da jednoj promjenljivoj (recimo A) odgovara *gornja polovina tablice*, a njenoj negaciji *donja polovina*, dok drugoj promjenljivoj (recimo B) odgovara *lijeva polovina*, a njenoj negaciji *desna polovina*. Ovo je grafički prikazano na sljedećoj slici, pri čemu crta sa strane i iznad označava zonu koju "pokrivaju" promjenljive A i B. Vidimo da svaka minterma formirana od promjenljivih A i B ima tačno određenu poziciju u dijagramu.



Sličnim rezonovanjem moguće je formirati Veitchove dijagrame za tri i četiri promjenljive. Ovi dijagrami prikazani su na sljedećoj slici:





Principijelno je moguće formirati i Veitchove dijagrame sa više od 4 promjenljive, ali su oni jako nepregledni. Naime, u Veitchovim dijagramima sa 2, 3 i 4 promjenljive, zone koje pokriva svaka od promjenljivih sastoje se od međusobno susjednih polja. Pokazuje se da u dvije dimenzije to nije moguće učiniti za slučaj 5 ili više promjenljivih, tako da neka od promjenljivih (npr. E) mora pokrivati više manjih cjelina koje, mada čine polovicu tablice, nisu sastavljene od međusobno susjednih polja što, kao što ćemo kasnije vidjeti, znatno otežava primjenu metode. Alternativno se Veitchovi dijagrami sa 5 i 6 promjenljivih mogu formirati kao prostorne strukture oblika kvadra, što je također nepregledno. Zbog toga Veitchova metoda nije prikladna za veći broj promjenljivih.

Prvi korak u primjeni Veitchove metode sastoji se u tome da se funkcija čiju minimalnu formu tražimo predstavi u formi Veitchovih dijagrama. To je najlakše uraditi za slučaj kada je funkcija zadana tabelarno. Da bismo neku tabelarno zadanu funkciju predstavili Veitchovim dijagramom, dovoljno je sve vrijednosti koje funkcija uzima u pojedinim redovima tabele upisati u Veitchov dijagram umjesto odgovarajućih mintermi koje odgovaraju pojedinim redovima (pri čemu minterme formiramo kao pri predstavljanju funkcije savršenom disjunktivnom normalnom formom).

Primjer 9.1.

- Predstaviti sljedeću tabelarno zadanu funkciju Veitchovim dijagramom.

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

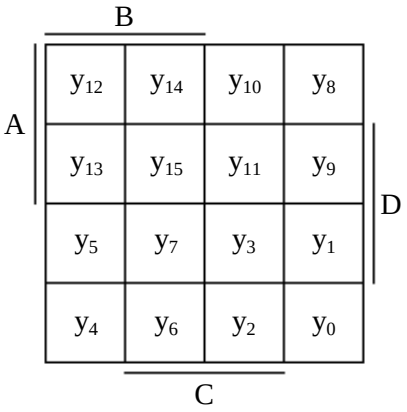
Ako numeriramo redove ove tablice redom od 0 do 7, tada vidimo da funkcija uzima vrijednost 1 u redovima 1, 2, 3, 6 i 7, a vrijednost 0 u redovima 0, 4 i 5. Redovima od 0 do 7 odgovaraju minterme $\overline{A}\,\overline{B}\,\overline{C}$, $\overline{A}\,\overline{B}C$, $\overline{A}B\overline{C}$, $\overline{A}BC$, $A\overline{B}\,\overline{C}$, $A\overline{B}C$, $AB\overline{C}$ i ABC, tako da lako možemo formirati traženi Veitchov dijagram:

A \ B				
	1	1	0	0
	1	1	1	0
		C		

Ovaj primjer možemo lako uopćiti. Neka u *k*-tom redu tablice istine funkcija Y uzima vrijednost y_k (pri čemu smo redove numerirali brojevima od 0 do 7). Tada ce prikaz funkcije Veitchovim dijagramom izgledati ovako:

A \ B				
	y_6	y_7	y_5	y_4
	y_2	y_3	y_1	y_0
		C		

Ovaj raspored vrijedi zapamtiti, jer nam omogućava da brzo formiramo Veitchov dijagram za svaku funkciju od tri promjenljive koja je zadana tablicom istine. Pimijetimo da se vrijednosti od y_0 do y_7 popunjavaju u karakterističnom poretku, od "uglova" ka "sredini" dijagrama. Analogno bismo za funkciju od četiri promjenljive dobili sljedeći raspored:



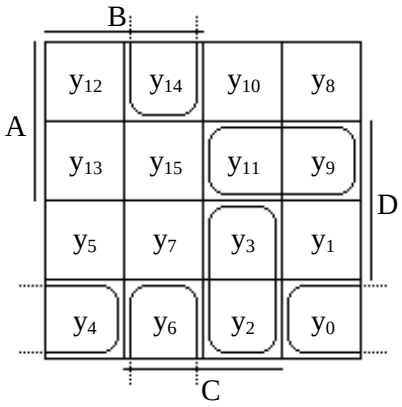
Ovaj raspored također treba upamtiti. Primijetimo da se i ovdje popunjavanje vrši od "uglova" ka "sredini" dijagrama.

Osnovna korist od predstavljanja funkcija Veitchovim dijagramima je u tome što se iz njih može, uz malo vještine, neposredno očitati MDNF ili MKNF polazne funkcije. Da bismo objasnili taj postupak, moramo prethodno uvesti neke pojmove.

Definicija 9.1:

Dva polja u Veitchovom dijagramu koja odgovaraju mintermama koje se razlikuju samo u stanju jedne promjenljive čine skupinu koja se naziva ***par***. Na primjer, na sljedećoj slici zaokružena su četiri tipična para u Veitchovom dijagramu za funkcije četiri promjenljive (analogni zaključci vrijede i za dijagrame za funkcije tri promjenljive):

.



Ako pažljivije proučimo Veitchove dijagrame za tri i četiri promjenljive, vidjećemo da par mogu sačinjavati dva polja koja su *susjedna* po horizontali ili po vertikali (ali ne i dijagonalno). Međutim, par mogu sačinjavati i dva polja koja se nalaze *na krajnje suprotnim pozicijama* bilo po horizontali (poput y_0 i y_4 na gornjoj slici), bilo po vertikali (poput y_6 i y_{14} na gornjoj slici). Ova polja također možemo smatrati susjednim ukoliko zamislimo da se Veitchov dijagram *periodički ponavlja* u sva četiri smjera, što je naznačeno crtkanim linijama na gornjoj slici. Alternativno, ova polja možemo također smatrati susjednim ukoliko smatramo da je dijagram namotan na *torus*, tako da su lijevi i desni kraj dijagrama međusobno spojeni, kao i gornji i donji kraj (zbog toga se kaže da Veitchovi dijagrami formiraju *topologiju torusa*). Ako rezimiramo ova razmatranja, možemo načiniti spisak svih mogućih parova koji se mogu javiti u Veitchovom dijagramu za funkcije četiri promjenljive:

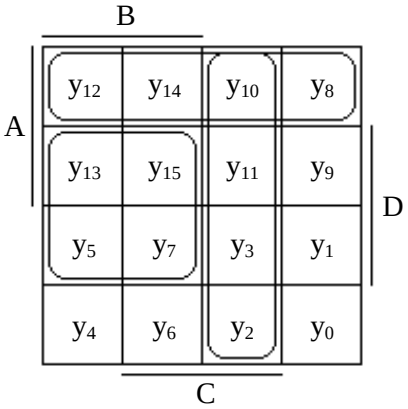
- Parovi sačinjeni od susjeda po horizontali: (y_{12}, y_{14}), (y_{13}, y_{15}), (y_5, y_7), (y_4, y_6), (y_{10}, y_{14}), (y_{11}, y_{15}), (y_3, y_7), (y_2, y_6), (y_8, y_{10}), (y_9, y_{11}), (y_1, y_3) i (y_0, y_2);
- Parovi sačinjeni od susjeda po vertikali: (y_{12}, y_{13}), (y_{14}, y_{15}), (y_{10}, y_{11}), (y_8, y_9), (y_5, y_{13}), (y_7, y_{15}), (y_3, y_{11}), (y_1, y_9), (y_4, y_5), (y_6, y_7), (y_2, y_3) i (y_0, y_1);
- Parovi sačinjeni od polja na suprotnim krajevima dijagrama po horizontali: (y_8, y_{12}), (y_9, y_{13}), (y_1, y_5) i (y_0, y_4);
- Parovi sačinjeni od polja na suprotnim krajevima dijagrama po vertikali: (y_4, y_{12}), (y_6, y_{14}), (y_2, y_{10}) i (y_0, y_8).

Nije teško pokazati da, bez obzira na broj promjenljivih, polja y_i i y_j uz $i < j$ čine par ako i samo ako je razlika $j - i$ stepen broja 2, i ukoliko pri tome binarni zapis broja j ima tačno jednu jedinicu više nego binarni zapis broja i.

Iz definicije para neposredno slijedi da par zapravo obrazuju polja koja odgovaraju mintermama *koje se mogu međusobno sažimati*. Zbog toga, možemo uzeti da svaki par predstavlja jednu *elementarnu konjunkciju* koja se dobija sažimanjem odgovarajućih mintermi. Na primjer par (y_9, y_{11}) sa prethodne slike predstavlja elementarnu konjunkciju $A\overline{B}D$, jer polje y_9 predstavlja mintermu $A\overline{B}\,\overline{C}D$, a polje y_{11} mintermu $A\overline{B}CD$, čijim se sažimanjem dobija upravo $A\overline{B}D$. Koju elementarnu konjunkciju predstavlja neki par možemo lako očitati direktno iz dijagrama. Na primjer, par (y_9, y_{11}) u potpunosti leži u zoni koju pokriva promjenljiva A, u potpunosti leži izvan zone koju pokriva promjenljiva B, i u potpunosti leži u zoni koju pokriva promjenljiva D. Što se tiče promjenljive C, ovaj par napola leži u zoni koja je pokrivena promjenljivom C, a napola ne leži, zbog toga promjenljiva C ne ulazi u elementarnu konjunkciju koju predstavlja ovaj par (po njoj se, zapravo, vrši sažimanje). Na sličan način direktno očitavamo da parovima (y_2, y_3), (y_6, y_{14}) i (y_0, y_4) odgovaraju elementarne konjunkcije $\overline{A}\,\overline{B}C$, $BC\overline{D}$ i $\overline{A}C\overline{D}$ respektivno.

Definicija 9.2:

Dva para u Veitchovom dijagramu koji predstavljaju elementarne konjunkcije koje se razlikuju samo u stanju jedne promjenljive čine skupinu koja se naziva ***četvorka***. Na primjer, na sljedećoj slici zaokružene su tri tipične četvorke u Veitchovom dijagramu za funkcije četiri promjenljive.

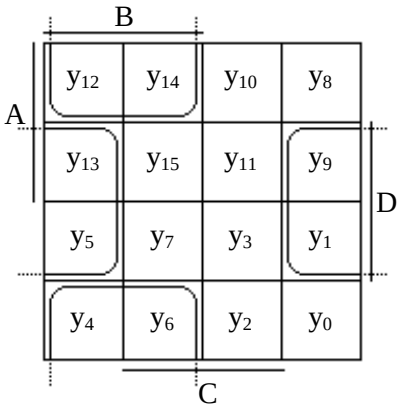


Odavde vidimo da se četvorke mogu formirati od dva para koji su susjedni bilo po vertikali bilo po horizontali, čime nastaju karakteristične figure oblika *kvadrata*, ili *horizontalnog* odnosno *vertikalnog štapića*. U Veitchovim dijagramima za funkcije četiri promjenljive na ovaj način je moguće formirati sljedeće četvorke:

- Četvorke oblika horizontalnih štapića, koje nastaju spajanjem dva horizontalna para koji su susjedni po horizontali: $(y_8, y_{10}, y_{12}, y_{14})$, $(y_9, y_{11}, y_{13}, y_{15})$, (y_1, y_3, y_5, y_7) i (y_0, y_2, y_4, y_6);
- Četvorke oblika vertikalnih štapića, koje nastaju spajanjem dva vertikalna para koji su susjedni po vertikali: $(y_4, y_5, y_{12}, y_{13})$, $(y_6, y_7, y_{14}, y_{15})$, $(y_2, y_3, y_{10}, y_{11})$ i (y_0, y_1, y_8, y_9);

- Četvorke oblika kvadrata, koje nastaju spajanjem dva horizontalna para koji su susjedna po vertikali, ili dva vertikalna para koji su susjedni po horizontali: $(y_{12}, y_{13}, y_{14}, y_{15})$, $(y_{10}, y_{11}, y_{14}, y_{15})$, $(y_8, y_9, y_{10}, y_{11})$, $(y_5, y_7, y_{13}, y_{15})$, $(y_3, y_7, y_{11}, y_{15})$, (y_1, y_3, y_9, y_{11}), (y_4, y_5, y_6, y_7), (y_2, y_3, y_6, y_7) i (y_0, y_1, y_2, y_3).

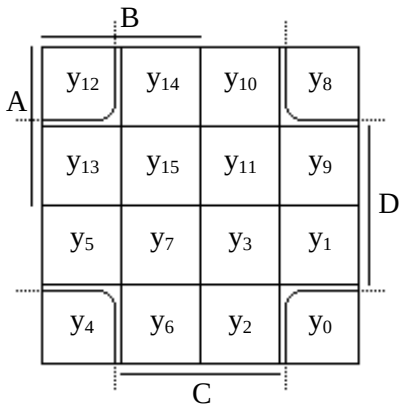
Pored navedenih četvorki koje su očigledne, moguće je formirati i manje očigledne četvorke, koje su sastavljene od dva para koja se nalaze na *krajnje suprotnim stranama dijagrama*, a koji se nalaze unutar istog sloja dijagrama, bilo po horizontali, bilo po vertikali. Na sljedećoj slici su prikazane dvije takve četvorke:



Za ove četvorke možemo također smatrati da su sastavljene od susjednih parova, ukoliko zamislimo da se dijagram *periodički produžuje* na sve strane. Ovakve četvorke se ne javljaju u dijagramima za funkcije tri promjenljive, dok je u dijagramu za funkcije četiri promjenljive moguće formirati sljedeće četvorke ovog tipa:

- Četvorke sastavljene od parova na krajnje suprotnim stranama unutar istog sloja po horizontali: $(y_8, y_9, y_{12}, y_{13})$, (y_1, y_5, y_9, y_{13}) i (y_0, y_1, y_4, y_5);
- Četvorke sastavljene od parova na krajnje suprotnim stranama unutar istog sloja po vertikali: $(y_4, y_6, y_{12}, y_{14})$, $(y_2, y_6, y_{10}, y_{14})$ i (y_0, y_2, y_8, y_{10}).

U Veitchovom dijagramu za četiri promjenljive moguće je formirati još jednu četvorku (y_0, y_4, y_8, y_{12}), koja se sastoji od *četiri rubna polja*:



Ovim smo ilustrirali sve četvorke koje se mogu javiti u dijagramima sa četiri promjenljive. Generalno, polja y_i, y_j, y_k i y_l uz $i < j < k < l$ čine četvorku ako i samo vrijedi $j - i = l - k$ i ako pri tome paovi indeksa (i, j), (k, l) i (i, k) ispunjavaju ranije navedene uvjete neophodne za formiranje para. Možemo primijetiti da ukoliko zamislimo da se dijagram *periodično produžuje* (ili da je *namotan na torus*), sve četvorke zapravo imaju *oblik štapića ili oblik kvadrata*.

Analogno kao u slučaju parova, četvorke zapravo sačinjavaju parovi koji predstavljaju elementarne konjunkcije koje se mogu dalje sažimati u prostije elementarne konjunkcije. Zbog toga za svaku

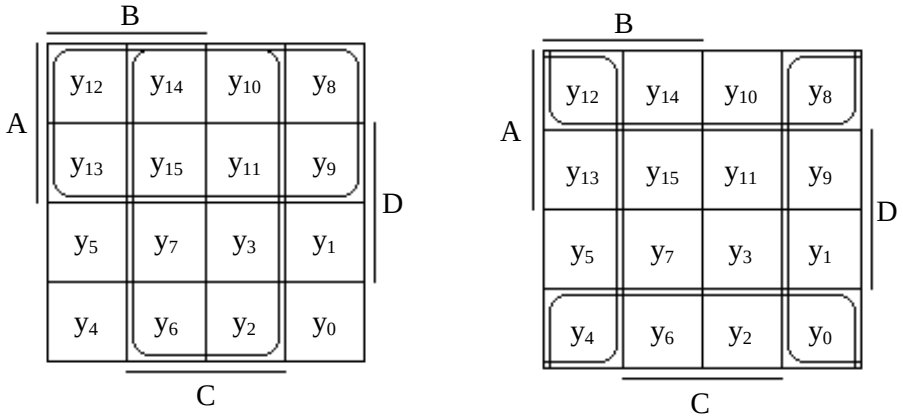
četvorku također možemo smatrati da predstavlja jednu elementarnu konjunkciju, dobijenu sažimanjem odgovarajućih elementarnih konjunkcija koje predstavljaju parove od kojih je četvorka sastavljena. Stoga se elementarna konjunkcija koju predstavlja četvorka sastoji od onih članova koji su zajednički za obe elementarne konjunkcije koje opisuju parove od kojih je četvorka sastavljena. Na primjer, četvorka $(y_4, y_6, y_{12}, y_{14})$ sa jedne od prethodnih slika predstavlja elementarnu konjunkciju $B\overline{D}$, jer se sastoji od parova (y_4, y_6) i (y_{12}, y_{14}) koji predstavljaju elementarne konjunkcije $\overline{A}B\overline{D}$ i $AB\overline{D}$ respektivno.

Elementarnu konjunkciju koju predstavlja neka četvorka također možemo očitati direktno iz dijagrama, na analogan način kao u slučaju parova. Tako, na primjer, četvorka o kojoj je riječ u potpunosti leži u zoni koju pokriva promjenljiva B, i u potpunosti leži izvan zone koju pokriva promjenljiva D, dok za promjenljive A i C imamo polovično prekrivanje, stoga ove promjenljive ne ulaze u elementarnu konjunkciju koja predstavlja ovu četvorku.

Definicija 9.3:

Dvije četvorke u Veitchovom dijagramu koji predstavljaju elementarne konjunkcije koje se razlikuju samo u stanju jedne promjenljive čine skupinu koja se naziva ***osmica***.

Za slučaj funkcija tri promjenljive, postoji samo jedna osmica, koja prekriva *čitav dijagram*. U dijagramima za funkcije četiri promjenljive, mogu se formirati osmice koje imaju jedan od četiri moguća oblika, koji su prikazani na sljedećim slikama:



Preciznije, sve osmice koje se mogu formirati u dijagramu za funkcije četiri promjenljive su:

- Tri pravougaone osmice sa dužom horizontalnom stranom: $(y_8, y_9, y_{10}, y_{11}, y_{12}, y_{13}, y_{14}, y_{15})$, $(y_1, y_3, y_5, y_7, y_9, y_{11}, y_{13}, y_{15})$ i $(y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7)$;
- Tri pravougaone osmice sa dužom vertikalnom stranom: $(y_4, y_5, y_6, y_7, y_{12}, y_{13}, y_{14}, y_{15})$, $(y_2, y_3, y_6, y_7, y_{10}, y_{11}, y_{14}, y_{15})$ i $(y_0, y_1, y_2, y_3, y_8, y_9, y_{10}, y_{11})$;
- Dvije rasječene osmice: $(y_0, y_1, y_4, y_5, y_8, y_9, y_{12}, y_{13})$ i $(y_0, y_2, y_4, y_6, y_8, y_{10}, y_{12}, y_{14})$.

Elementarne konjunkcije koje odgovaraju osmicama definiraju se analogno kao za parove i četvorke. Za slučaj funkcija četiri promjenljive sve osmice predstavljaju elementarne konjunkcije koje se svode samo na usamljenu promjenljivu ili njenu negaciju.

Analogno se može definirati i ***šesnaestica***, ali za slučaj dijagrama za funkcije četiri promjenljive postoji samo jedna jedina šesnaestica, koja obuhvata *sva polja dijagrama* (i njoj odgovara konstanta 1). Parove, četvorke, osmice itd. zajedničkim imenom nazivamo ***konture***. Za slučaj Veitchovih dijagrama za funkcije sa 5 ili više promjenljivih, moguće je definirati i *veće konture od šesnaestica*. Međutim, Veitchovi dijagrami za funkcije sa više od 4 promjenljive su veoma nepraktični, jer parove, četvorke itd. mogu tvoriti i polja koja uopće *nisu susjedna* u dijagramu, čak ni uz periodično produženje dijagrama ili namatanje na torus (što je još gore, polja koja *jesu* susjedna *ne moraju* činiti par), tako da je takve

konture *teško uočiti*, a Veitchova metoda određivanja MDNF i MKNF upravo se zasniva na *prepoznavanju karakterističnih kontura* unutar Veitchovih dijagrama.

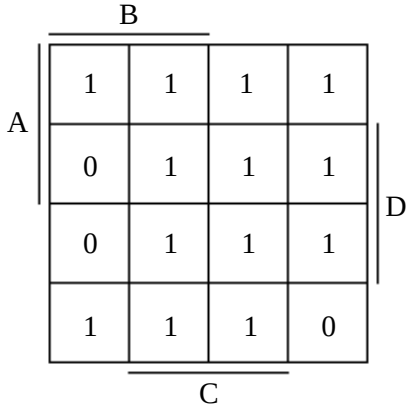
Nakon što smo definirali pojam kontura, sasvim je jednostavno objasniti suštinu Veitchove metode za određivanje MDNF. Primijetimo prvo da svaka kontura koja u Veitchovom dijagramu za zadanu funkciju prekriva samo polja u kojima se nalazi jedinica zapravo predstavlja *implikantu* zadane funkcije, dok svaka kontura koja prekriva samo jedinice a koja nije sadržana u nekoj većoj konturi koja također pokriva samo jedinice predstavlja *prostu implikantu* funkcije. Stoga je za određivanje MDNF potrebno prekriti *sva polja u kojima se nalazi jedinica* sa *što je god moguće manjim brojem što je god moguće većih kontura* (pri tome, isto polje smije biti prekriveno sa više kontura). Naime, kako konture maksimalne veličine koje prekrivaju samo jedinice odgovaraju prostim implikantama, ovaj postupak odgovara traženju *minimalnog skupa prostih implikanti* čija disjunkcija formira datu funkciju. Tražena MDNF se tada formira kao *disjunkcija implikanti koje odgovaraju svakoj od upotrebljenih kontura*. Odavde vidimo da je Veitchova metoda zaista samo prerušeni Quineov algoritam u kojem je mučno određivanje prostih implikanti sažimanjem svedeno na prepoznavanje karakterističnih kontura u dijagramu, što je postignuto veoma oštroumnim raspoređivanjem polja unutar dijagrama, tako da članovi koji se mogu sažimati odgovaraju međusobno susjednim poljima.

Primjer 9.2:

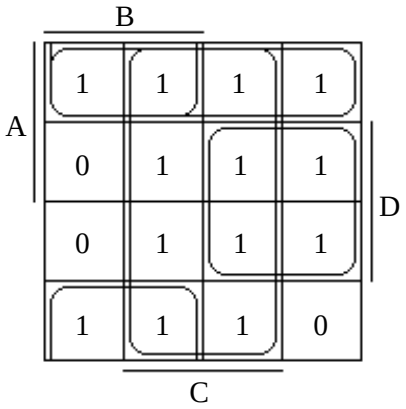
- Odrediti minimalnu disjunktivnu formu funkcije koja je predstavljena sljedećom tablicom istine:

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

Prvi korak u određivanju MDNF je predstavljanje ove funkcije Veitchovim dijagramom, što je prema već opisanom postupku urađeno na sljedećoj slici:



Sada je potrebno sve jedinice u dijagramu prekriti na što bolji način. Lako možemo uočiti da optimalno prekrivanje možemo postići korištenjem jedne osmice i tri četvorke, kao što je prikazano na sljedećoj slici:



Na kraju je potrebno utvrditi koje elementarne konjunkcije odgovaraju pojedinim konturama, što je posve lako izvesti posmatranjem zona u dijagramu koje prekrivaju pojedine promjenljive. Tražena MDNF glasi:

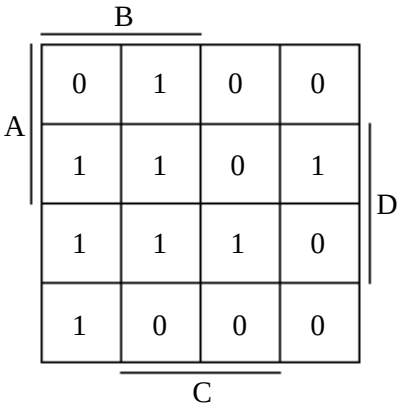
$$Y = C \vee \overline{B}D \vee A\overline{D} \vee B\overline{D}.$$

Dobra strana Veitchove metode je što ona uvijek daje ispravnu disjunktivnu normalnu formu funkcije čak i ukoliko nismo izvršili optimalno prekrivanje jedinica, pod uvjetom da smo ispravno očitali elementarne konjunkcije koje odgovaraju upotrebljenim konturama. Međutim, u tom slučaju pronađena DNF (mada ispravna) neće biti optimalna, tj. neće biti MDNF. U slučaju da postoji nekoliko jednako dobrih pokrivanja, to znači da imamo više različitih MDNF, koje su u suštini podjednako dobre.

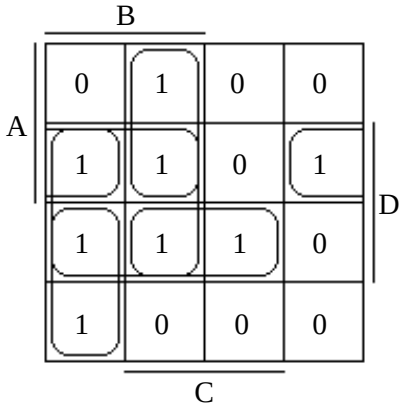
Pri traženju optimalnog prekrivanja nije pametno poći od najvećih kontura (koje su obično najuočljivije), nego od onih kontura koje prekrivaju jedinice koje se mogu grupirati na samo jedan način (tj. koje imaju samo jednu jedinicu u susjedstvu). Takve konture nazivamo ***esencijalne konture***, a njima odgovaraju *esencijalne implikante* u Quineovoj metodi. Tipičnu grešku koja se može napraviti ukoliko se ne pridržavamo ovog pravila ilustrira sljedeći primjer.

Primjer 9.3:

- Odrediti MDNF za funkciju predstavljenu sljedećim Veitchovim dijagramom:



U ovom Veitchovom dijagramu se na prvi pogled uočava jedna četvorka. Ukoliko zaokružimo ovu četvorku, preostaju nepokrivene četiri jedinice koje se mogu povezati u par sa susjednim jedinicama. Na taj način dobijamo sljedeće prekrivanje:



Na osnovu ovakvog prekrivanja, mogli bismo očitati sljedeću funkciju:

$$Y = BD \vee ABC \vee \overline{A}\overline{B}\overline{C} \vee A\overline{C}D \vee \overline{A}CD$$

Međutim, ukoliko pažljivo razmotrimo dijagram, primijetićemo da je četvorka koju smo prvu ugledali zapravo suvišna, jer su sve jedinice koje ona pokriva već prekrivene sa preostala četiri para. Stoga, očitana DNF nije nesvodljiva, jer se prva elementarna konjunkcija može odstraniti. Slijedi da tražena minimalna disjunktivna normalna forma funkcije glasi:

$$Y = ABC \vee \overline{A}\overline{B}\overline{C} \vee A\overline{C}D \vee \overline{A}CD$$

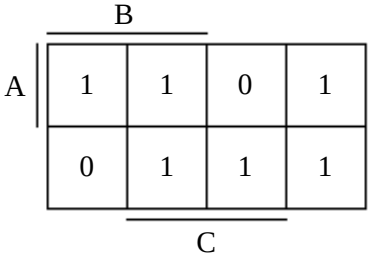
Da smo prilikom prekrivanja krenuli od jedinica koje se mogu prekriti na jedinstven način, ovakvu grešku ne bismo napravili.

Nakon što iscrpimo sve esencijalne konture, za optimalno prekrivanje preostalih jedinica pomaže samo intuicija (upravo zbog toga Veitchova metoda nema striktnu formu algoritma). Srećom, kako se Veitchova metoda koristi samo za mali broj promjenljivih, optimalno prekrivanje se veoma brzo uoči, pogotovo kada se korisnik metode malo izvježba. Dobra je strategija svakom novom konturom pokušati prekriti što je god moguće više do tada nepokrivenih jedinica, kao što je ilustrirano u sljedećem primjeru.

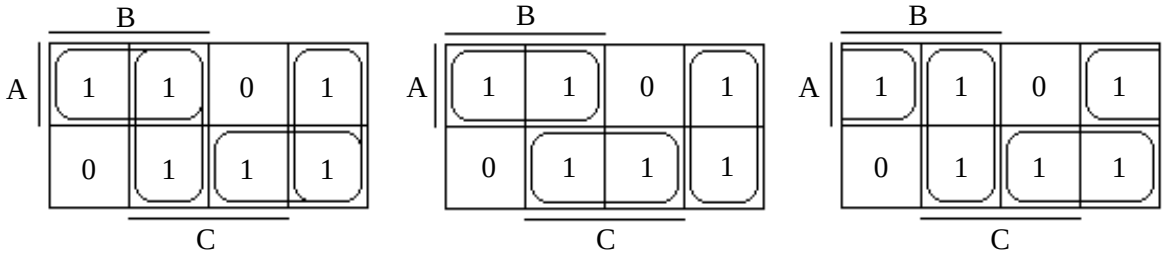
Primjer 9.4:

- Naći MDNF za funkciju $Y = \overline{A}\,\overline{B}\,\overline{C} \vee \overline{A}\,\overline{B}C \vee \overline{A}BC \vee A\overline{B}\,\overline{C} \vee AB\overline{C} \vee ABC$

Ovu funkciju možemo predstaviti Veitchovim dijagramom na sljedeći način:



U ovom primjeru ne postoji niti jedna esencijalna kontura, jer se svaka jedinica može prekriti na više različitih načina. Postoji mnogo načina kako možemo prekriti sve jedinice u ovom dijagramu. Nas zanimaju samo *nesvodljiva prekrivanja*, tj. prekrivanja u kojima niti jedna kontura nije suvišna u smislu da prekriva jedinice koje su već prekrivene drugim konturama. Sljedeća slika prikazuje tri od ukupno pet mogućih nesvodljivih prekrivanja:



Mada su sva tri prikazana prekrivanja očigledno nesvodljiva, prvo prekrivanje sadrži *jednu konturu više* nego drugo i treće prekrivanje, i stoga nije optimalno. Primijetimo da se u prvom slučaju nismo držali pravila da svakom novom konturom pokrivamo što je god moguće više do tada nepokrivenih jedinica, dok smo se u drugom i trećem slučaju tog pravila pridržavali. Također, primijetimo da smo u drugom i trećem slučaju svaku od jedinica prekrili samo jednom konturom. Ukoliko se to desi (što je rijetko moguće postići), to je siguran znak da je pronađeno rješenje optimalno. Stoga, možemo očitati dvije minimalne disjunktivne normalne forme za zadanu funkciju:

$$Y = AB \vee \overline{A}C \vee \overline{B}\,\overline{C}$$

$$Y = A\overline{C} \vee BC \vee \overline{A}\,\overline{B}$$

Lako je provjeriti da su ovo ujedno i jedine dvije MDNF za zadanu funkciju.

Mada Veitchova metoda (kao i sve druge metode minimizacije) određuje MDNF za zadanu funkciju, to ne znači da se dobijena funkcija ne može dodatno skratiti ručno pomoću pravila Booleove algebre. Međutim, ono što eventualno dobijemo daljim skraćivanjem neće više biti disjunktivna normalna forma. Na primjer, razmotrimo sljedeću funkciju od 7 promjenljivih:

$$Y = ADF \vee AEF \vee BDF \vee BEF \vee CDF \vee CEF \vee G$$

Nije teško pokazati da je ova funkcija u minimalnoj disjunktivnoj normalnoj formi. Ipak, ova funkcija se relativno lako može dodatno pojednostaviti primjenom pravila Booleove algebre:

$$\begin{aligned} Y &= ADF \vee AEF \vee BDF \vee BEF \vee CDF \vee CEF \vee G = \\ &= (AD \vee AE \vee BD \vee BE \vee CD \vee CE)\,F \vee G = \\ &= [A\,(D \vee E) \vee\ B\,(D \vee E) \vee C\,(D \vee E)]\,F \vee G = \\ &= (A \vee B \vee C)\,(D \vee E)\,F \vee G \end{aligned}$$

Dobijena funkcija je znatno jednostavnija od polazne, mada se očito ne radi o disjunktivnoj (niti o konjunktivnoj) normalnoj formi. Također, često se dodatna optimizacija može izvršiti ako dozvolimo upotrebu i drugih logičkih operacija osim konjunkcije, disjunkcije i negacije. Tako se, na primjer, ukoliko dozvolimo i upotrebu ekskluzivne disjunkcije, funkcija iz Primjera 9.3 može dodatno optimizirati na sljedeći način:

$$Y = ABC \vee \overline{A}B\overline{C} \vee A\overline{C}D \vee \overline{A}CD = B(AC \vee \overline{A}\,\overline{C}) \vee D(A\overline{C} \vee \overline{A}C) = B\overline{A \oplus C} \vee D(A \oplus C)$$

Nažalost, za eventualnu dodatnu optimizaciju funkcija koje su predstavljene u vidu MDNF ili MKNF ne postoje nikakva sistematična pravila, već samo neki intuitivni postupci, koji ne vode nužno ka optimalnom rješenju. Međutim, u sljedećim poglavljima ćemo vidjeti da se, zbog raznovrsnih razloga, dodatna optimizacija funkcija koje su već svedene na MDNF ili MKNF najčešće i ne zahtijeva.

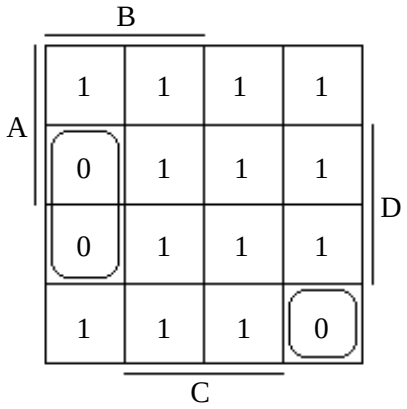
Veitchova metoda može se koristiti i za određivanje minimalne konjunktivne normalne forme. Pri određivanju MKNF, umjesto jedinica prekrivamo nule u dijagramu (na isti način kao kod određivanja

MDNF). Svakoj konturi u izvršenom prekrivanju tada odgovara po jedna *elementarna disjunkcija* u MKNF, pri čemu su promjenljive koje formiraju svaku elementarnu disjunkciju *negirane* u odnosu na slučaj kada ista kontura predstavlja elementarnu konjunkciju. Na primjer, ukoliko neka kontura predstavlja elementarnu konjunkciju $A\overline{B}D$, tada ta ista kontura predstavlja elementarnu disjunkciju $\overline{A} \vee B \vee \overline{D}$. Valjanost ovog postupka neposredno slijedi iz činjenice da je MKNF neke funkcije jednaka negaciji MDNF njene negacije.

Primjer 9.5:

- Odrediti minimalnu konjunktivnu normalnu formu za funkciju iz Primjera 9.2.

Da bismo odredili MKNF za zadanu funkciju, možemo se poslužiti istim Veitchovim dijagramom koji smo imali u Primjeru 9.2, ali ćemo ovaj put prekrivati *nule*. Kako u čitavom dijagramu imaju samo tri nule, lako je odrediti optimalno prekrivanje. Pri tome nulu u donjem desnom uglu ne možemo ni sa čim upariti, tako da ona *sama za sebe* predstavlja konturu:



Par nula sa lijeve strane u potpunosti leži unutar zona koje prekrivaju promjenljive B i D, a izvan zone koju prekriva promjenljiva C. Usamljena nula u donjem desnom uglu u potpunosti leži izvan zona koje prekriva svaka od četiri promjenljive A, B, C i D. Na osnovu ovoga, slijedi da tražena MKNF glasi:

$$Y = (\overline{B} \vee C \vee \overline{D})(A \vee B \vee C \vee D)$$

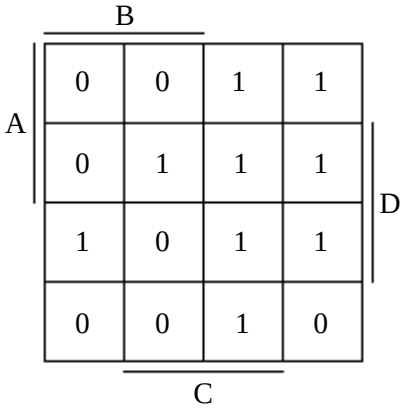
Za ovu funkciju nađena MKNF je potpuno iste dužine kao i njena MDNF, što ne mora uvijek biti slučaj (nekada je MDNF kraća od MKNF za istu funkciju, a nekada je obrnuto). Ipak, minimalna konjunktivna normalna forma za ovu funkciju je, na izvjestan način, *povoljnija* za realizaciju u odnosu na minimalnu disjunktivnu normalnu formu. Naime za realizaciju MKNF potrebno nam je jedno troulazno OR kolo, jedno četveroulazno OR kolo, jedno dvoulazno AND kolo i dva invertora (NOT kola). U digitalnoj tehnici se, zbog tehnoloških razloga, kao kriterij za procjenu povoljnosti realizacije često koristi *neophodni broj ulaza u logička kola* prilikom realizacije funkcije, što u ovom slučaju daje 11 ulaza. Međutim, MDNF za istu funkciju je glasila:

$$Y = C \vee \overline{B}D \vee A\overline{D} \vee B\overline{D}$$

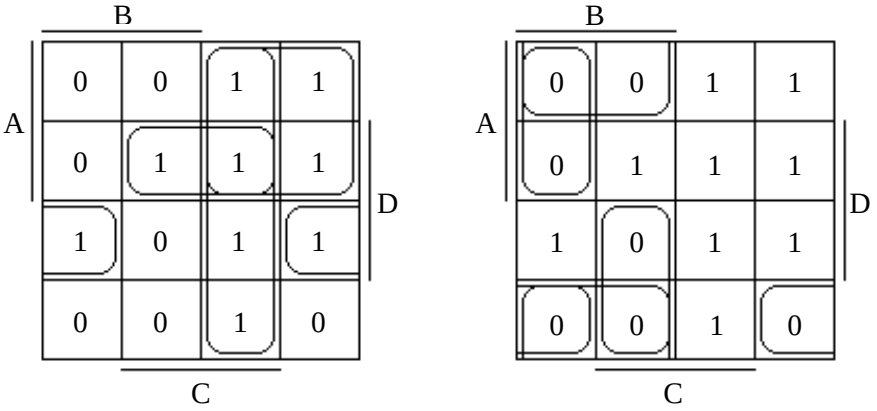
Za njenu realizaciju potrebna su tri dvoulazna AND kola, jedno četveroulazno OR kolo, i dva invertora, što ukupno daje 12 ulaza. Slijedi da je u ovom slučaju MKNF neznatno povoljnija za realizaciju.

Primjer 9.6:

- Odrediti MDNF i MKNF za funkciju predstavljenu sljedećim Veitchovim dijagramom, i ustanoviti koja je od dvije dobijene forme povoljnija za realizaciju.



Za određivanje MDNF i MKNF izvršićemo optimalno prekrivanje jedinica i nula, kao što je prikazano na sljedećoj slici:



Optimalno prekrivanje kako jedinica tako i nula može se izvršiti na jednistven način, odakle slijedi jedinstvena minimalna disjunktivna odnosno konjunktivna normalna forma:

$$Y = A\overline{B} \vee \overline{B}C \vee ACD \vee \overline{A}\,\overline{C}D \qquad /MDNF/$$

$$Y = (\overline{A} \vee \overline{B} \vee C)(A \vee \overline{B} \vee \overline{C})(A \vee C \vee D)(\overline{B} \vee D) \qquad /MKNF/$$

U ovom slučaju MDNF je povoljnija za realizaciju, jer zahtijeva 17 ulaza u logička kola dok MKNF zahtijeva 18 ulaza. Nije teško vidjeti da je broj logičkih funkcija kod kojih je MDNF povoljnija za realizaciju u odnosu na MKNF jednak broju logičkih funkcija kod kojih je MKNF povoljnija za realizaciju u odnosu na MDNF. Naime, ukoliko je za neku funkciju MDNF povoljnija za realizaciju (u odnosu na MKNF), zamjenom svih jedinica u Veitchovom dijagramu nulama i obrnuto, dobijamo funkciju kod koje je MKNF povoljnija za realizaciju!

Dobivena MDNF se može dodatno ručno optimizirati. Međutim, pri optimizaciji ne treba biti brzoplet, jer su nekada prividno jednostavnije funkcije zapravo složenije za realizaciju. Na primjer, ukoliko iz prva dva člana u dobivenoj MDNF izvučemo zajednički faktor (negaciju promjenljive B), dobijamo funkciju

$$Y = \overline{B}(A \vee C) \vee ACD \vee \overline{A}\,\overline{C}D$$

koja zahtijeva 16 ulaza. Međutim, ukoliko pored toga izvučemo zajednički faktor iz posljednja dva člana, dobijamo funkciju

$$Y = \overline{B}(A \vee C) \vee D(AC \vee \overline{A}\,\overline{C})$$

koja zahtijeva ponovo 17 ulaza (mada djeluje prostije od prethodne). Ipak, posljednju funkciju možemo zapisati u obliku

$$Y = \overline{B}(A \vee C) \vee D\,\overline{A \oplus C}$$

koja zahtijeva svega 11 ulaza, uz pretpostavku da nam je na raspolaganju kolo koje na izlazu daje negaciju ekskluzivne disjunkcije (EXNOR kolo). Odavde vidimo da krajnji oblik na koji trebamo da svedemo funkciju može bitno zavisiti od toga šta nam stoji na raspolaganju za njenu realizaciju. O nekim dodatnim aspektima realizacije logičkih funkcija govorićemo u narednom poglavlju.

Interesantno je napomenuti da u slučaju kada je za realizaciju dozvoljena upotreba EXOR ili EXNOR logičkih kola, povoljnija realizacija može slijediti iz disjunktivne forme koja nije minimalna, u odnosu na realizaciju koja slijedi iz minimalnih formi. Na primjer, razmotrimo funkciju iz Primjera 9.4. Ova funkcija ima dvije MDNF (i jednu MKNF), iz kojih se ne vidi nikakva mogućnost eventualne realizacije uz upotrebu EXOR ili EXNOR logičkih kola. Međutim, ista funkcija ima još tri nesvodljive disjunktivne normalne forme, koje su veoma pogodne za realizaciju pomoću EXNOR logičkih kola. Na primjer, iz prvog Veitchovog dijagrama direktno slijedi

$$Y = AB \vee \overline{A}\,\overline{B} \vee BC \vee \overline{B}\,\overline{C} = \overline{A \oplus B} \vee \overline{B \oplus C}$$

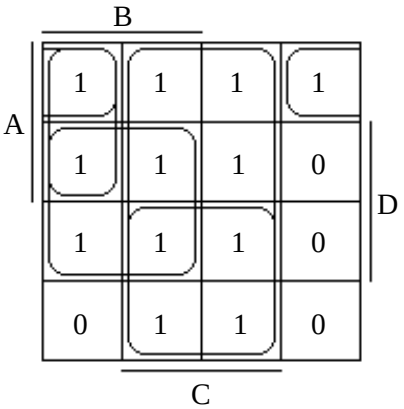
Na taj način, dolazimo do realizacije koja zahtijeva samo 6 ulaza, ali uz korištenje EXNOR logičkih kola. O mogućnostima primjene Veitchove metode za realizaciju uz pomoć EXOR ili EXNOR logičkih kola biće više govora u poglavljima koja slijede.

Veitchovi dijagrami mogu poslužiti i za minimizaciju funkcije koja je predstavljena u obliku proizvoljne disjunktivne ili konjunktivne normalne forme, bez potrebe da se prvo formira tablica istine pa da se iz nje formira Veitchov dijagram (što je još jedna velika prednost u odnosu na Quineov algoritam, koji uvijek polazi od SDNF). Za tu svrhu, prvo je potrebno u prazan Veitchov dijagram upisati konture koje odgovaraju elementarnim konjunkcijama zadane DNF ili elementarnim disjunkcijama zadane KNF. Nakon toga, popunimo upisane konture jedinicama (za slučaj DNF) ili nulama (za slučaj KNF), ostatak tablice popunimo suprotnim vrijednostima i, konačno, izvršimo prekrivanje tablice novim konturama na optimalan način.

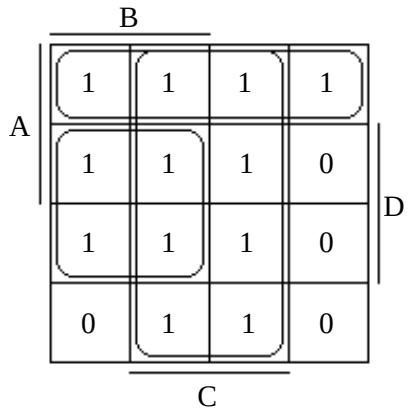
Primjer 9.7:

- Pojednostaviti funkciju $Y = AB\overline{C} \vee \overline{A}C \vee BD \vee A\overline{C}\,\overline{D} \vee C$.

Kako se radi o funkciji koja ima oblik DNF, upisaćemo u Veitchov dijagram konture koje odgovaraju svim elementarnim konjunkcijama ove funkcije, njihovu unutrašnjost ispunićemo jedinicama, a preostala polja nulama. Time dobijamo dijagram kao na sljedećoj slici:



Preostaje još samo da izvršimo optimalno prekrivanje ovih jedinica, što je izvršeno na sljedećoj slici:



Odavde vidimo da MDNF za zadanu funkciju glasi:

$$Y = A\overline{D} \vee BD \vee C$$

Ova funkcija se ne može dodatno ručno optimizirati. Također, lako je provjeriti da MKNF za ovaj primjer glasi

$$Y = (B \vee C \vee \overline{D})(A \vee C \vee D)$$

i da je, prema tome, složenija od nađene MDNF (zahtijeva jedan ulaz više).

Uz malo prakse, funkcije koje su zadane u proizvoljnoj DNF ili KNF moguće je lako upisati u Veitchov dijagram bez potrebe za crtanjem kontura koje odgovaraju pojedinim elementarnim konjunkcijama odnosno disjunkcijama. Ukoliko funkcija koju želimo da minimiziramo nije zadana niti u konjunktivnoj niti u disjunktivnoj normalnoj formi, tada ju je neophodno pomoću već opisanih postupaka prvo svesti na DNF ili na KNF (ne nužno na SDNF ili SKNF), nakon čega se može primijeniti Veitchova metoda.

Mada smo rekli da su Veitchovi dijagrami za funkcije sa 5 promjenljivih nezgrapni i nepraktični za upotrebu, minimizacija funkcija od 5 promjenljivih može se, uz malo dosjetki, lako izvršiti pomoću dva Veitchova dijagrama za funkcije od 4 promjenljive. Kako se to može uraditi, najlakše je objasniti kroz konkretan primjer.

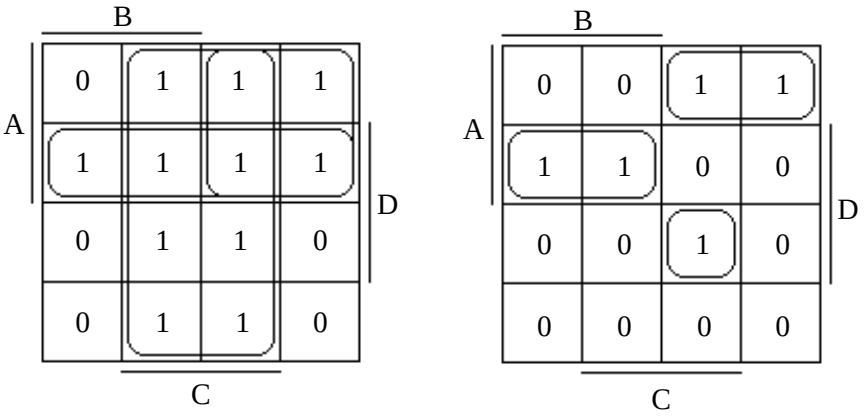
Primjer 9.8:

- Funkciju $Y = \overline{A}CE \vee ABDE \vee AB\overline{C}D\overline{E} \vee A\overline{B}\,\overline{C}E \vee A\overline{B}\,D\,\overline{E} \vee ACE \vee ABCD\overline{E} \vee \overline{A}\,\overline{B}CD\overline{E}$ svesti na minimalnu disjunktivnu normalnu formu.

Ova funkcija zavisi od 5 promjenljivih. Izabraćemo jednu od promjenljivih, recimo E, i grupirati zajedno sve članove koje sadrže nju ili njenu negaciju:

$$Y = (\overline{A}C \vee ABD \vee A\overline{B}\,\overline{C} \vee AC)E \vee (AB\overline{C}D \vee A\overline{B}\,\overline{D} \vee ABCD \vee \overline{A}\,\overline{B}CD)\overline{E}$$

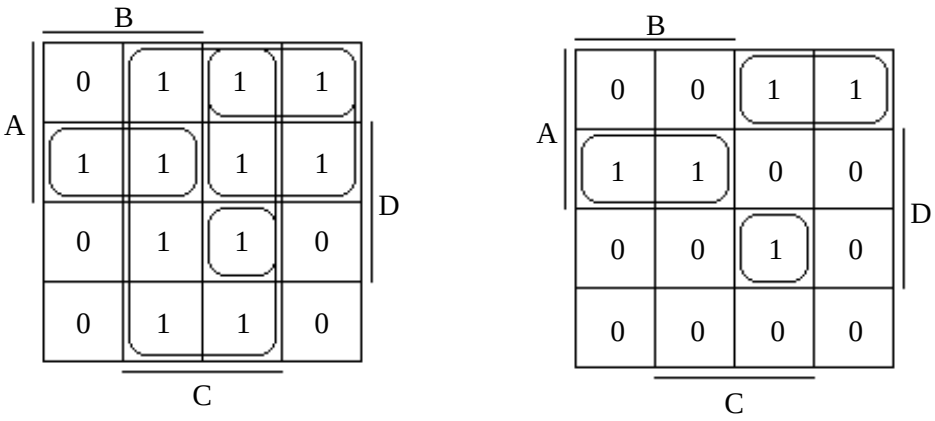
Kako obje funkcije koje se nalaze u zagradi zavise samo od po 4 promjenljive, možemo izvršiti njihovu minimizaciju pomoću Veitchovih dijagrama za funkcije od 4 promjenljive. Taj postupak je prikazan na sljedećoj slici (lijevi dijagram odgovara prvoj funkciji u zagradi, a desni dijagram drugoj funkciji):



Iz ovih dijagrama neposredno vidimo da MDNF oblici za funkcije u zagradama glase respektivno $AD \vee A\overline{B} \vee C$ i $ABD \vee A\overline{B}\,\overline{D} \vee \overline{A}\,\overline{B}\overline{C}D$. Stoga možemo pisati:

$$\begin{aligned} Y &= (AD \vee A\overline{B} \vee C)E \vee (ABD \vee A\overline{B}\,\overline{D} \vee \overline{A}\,\overline{B}CD)\overline{E} = \\ &= ADE \vee A\overline{B}E \vee CE \vee ABD\overline{E} \vee A\overline{B}\,\overline{D}\,\overline{E} \vee \overline{A}\,\overline{B}CD\overline{E} \end{aligned}$$

Dobijena funkcija zaista ima oblik DNF i znatno je kraća od polazne funkcije (za 12 literala). Međutim, ovo nije MDNF zadane funkcije! Razloge za to nije teško otkriti. Naime, kako smo obje funkcije koje se javljaju u zagradama optimizirali *neovisno jednu od druge*, potpuno smo isključili svaku mogućnost sažimanja po preostaloj promjenljivoj E. S druge strane, pretpostavimo da smo Veitchov dijagram za prvu funkciju u zagradi prekrili neoptimalno, na način prikazan na sljedećoj slici, u kojem je očigledno prisustvo "neoptimalne" konture ABD, kao i potpuno suvišnih kontura $A\overline{B}\,\overline{D}$ i $\overline{A}\,\overline{B}CD$:



Iz ovakvog prekrivanja možemo izvršiti sljedeće očitanje:

$$\begin{aligned} Y &= (ABD \vee A\overline{B} \vee C \vee A\overline{B}\,\overline{D} \vee \overline{A}\,\overline{B}CD)E \vee (ABD \vee A\overline{B}\,\overline{D} \vee \overline{A}\,\overline{B}CD)\overline{E} = \\ &= ABDE \vee A\overline{B}E \vee CE \vee A\overline{B}\,\overline{D}E \vee \overline{A}\,\overline{B}CDE \vee ABD\overline{E} \vee A\overline{B}\,\overline{D}\,\overline{E} \vee \overline{A}\,\overline{B}CD\overline{E} = \\ &= ABD(E \vee \overline{E}) \vee A\overline{B}E \vee CE \vee A\overline{B}\,\overline{D}(E \vee \overline{E}) \vee \overline{A}\,\overline{B}CD(E \vee \overline{E}) = \\ &= ABD \vee A\overline{B}E \vee CE \vee A\overline{B}\,\overline{D} \vee \overline{A}\,\overline{B}CD \end{aligned}$$

Vidimo da smo dobili disjunktivnu normalnu formu koja ima čak 6 literala manje nego prethodna forma (i ona je, zapravo i MDNF tražene funkcije). Nije teško utvrditi šta se zapravo desilo: neoptimalne i suvišne konture u lijevom dijagramu poklapaju se sa konturama u desnom dijagramu, tako da su odgovarajuće elementarne konjunkcije u obje zagrade identične, što omogućava sažimanje po preostaloj promjenljivoj E. Uz malo prakse, nije nikakav problem direktno očitati krajnji rezultat posmatrajući uporedo oba dijagrama, bez ručnog sažimanja po preostaloj promjenljivoj E.

Ukoliko pažljivije razmotrimo opisani primjer, možemo izvesti sljedeće opće zaključke o minimizaciji funkcija od 5 promjenljivih pomoću dva spregnuta Veitchova dijagrama za funkcije od 4 promjenljive:

- Isplati se zaokružiti konturu koja u jednom od spregnutih dijagrama nije optimalna, ukoliko identična kontura postoji u drugom dijagramu;
- Isplati se u jedan od spregnutih dijagrama uvesti suvišne konture, ukoliko identične konture postoje u drugom dijagramu.

Očigledno je da je potrebna nešto veća količina iskustva i domišljatosti za dobivanje minimalne forme funkcije od 5 promjenljivih korištenjem dva Veitchova dijagrama za funkcije od 4 promjenljive. Međutim, čak i ukoliko se ne dobije posve minimalna forma, uloženi trud se isplati, jer je bilo kakva optimizacija svakako bolja od nikakve. Postupak sličan izloženom postupku može se primijeniti i na minimizaciju funkcija od po 6 promjenljivih, samo se u tom slučaju koriste 4 spregnuta dijagrama od po 4 promjenljive. Ovaj postupak čitatelj ili čitateljica mogu lako izvesti sami, ukoliko su dobro razumjeli gore izloženi postupak za funkcije sa 5 promjenljivih.

Ovim smo završili izlaganje osnovnih koncepata Veitchove metode. Međutim, time nisu iscrpljene sve mogućnosti koje Veitchova metoda pruža. Kako su Veitchovi dijagrami osnovni alat za savremenu analizu i projektovanje digitalnih struktura, sa njima ćemo se intenzivno susretati i kroz poglavlja koja slijede. Tom prilikom ćemo kroz razne praktične probleme ukazati i na još neke aspekte Veitchove metode.

(?) Pitanja i zadaci

9.1 Precizno opišite razne tipove kontura koje se mogu javljati u Veitchovim dijagramima.

9.2 Pokažite primjerom da je moguće konstruisati dijagrame sa drugačijim razmještajem polja od Veitchovih dijagrama sa kojima je moguće ostvariti isti efekat.

9.3 Pokažite da dva polja u Veitchovom dijagramu koja odgovaraju mintermama sa indeksima i i j uz $i<j$ čine par ako i samo ako je razlika $j-i$ stepen broja 2, i ukoliko pri tome binarni zapis broja j ima tačno jednu jedinicu više nego binarni zapis broja i.

9.4 Pokažite da četiri polja u Veitchovom dijagranu koja odgovaraju mintermama sa indeksima i, j, k i l uz $i<j<k<l$ čine četvorku ako i samo vrijedi $j-i=l-k$ i ako pri tome parovi indeksa (i, j), (k, l) i (i, k) ispunjavaju uvjete navedene u 9.3 za formiranje para.

9.5 Izvedite koje uvjete trebaju zadovoljavati indeksi mintermi u Veitchovom dijagramu da bi odgovarajuća polja činila osmicu.

9.6 Pokažite zbog čega je već za slučaj 5 promjenljivih nemoguće formirati Veitchov dijagram na posve pregledan način.

9.7 Objasnite šta su esencijalne konture.

9.8 Objasnite suštinu Veitchove metode minimizacije i pokažite da je ona zapravo ekvivalentna sa Quineovim algoritmom.

9.9 Za funkciju iz Primjera 9.4, postoji ukupno pet nesvodljivih prekrivanja Veitchovog dijagrama, od kojih su prikazana tri. Pronađite preostala dva nesvodljiva prekrivanja.

9.10 Predstavite logičku funkciju $Y=\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}\vee\overline{A}\,\overline{B}\,\overline{C}D\vee\overline{A}B\overline{C}\,\overline{D}\vee\overline{A}B\overline{C}D\vee AB\overline{C}\overline{D}\vee AB\overline{C}D$ pomoću Veitchovog dijagrama, a zatim očitajte MDNF i MKNF oblik date logičke funkcije.

9.11 Predstavite logičku funkciju $Y=(A\vee B\vee C)(A\vee B\vee\overline{C})(\overline{A}\vee\overline{B}\vee C)(\overline{A}\vee\overline{B}\vee\overline{C})$ pomoću Veitchovog dijagrama, a zatim očitajte MDNF i MKNF oblik date logičke funkcije.

9.12 Koristeći Veitchove dijagrame, nađite MDNF i MKNF oblike za sljedeće logičke funkcije, uz pretpostavku da sve minterme zavise od promjenljivih A, B, C i D:

a) $Y=m_0\vee m_2\vee m_4\vee m_6$

b) $Y=m_1\vee m_2\vee m_3\vee m_5\vee m_8\vee m_9\vee m_{11}\vee m_{15}$

c) $Y=m_0\vee m_1\vee m_4\vee m_9\vee m_{11}\vee m_{12}\vee m_{14}\vee m_{15}$

d) $Y=m_0\vee m_1\vee m_2\vee m_3\vee m_4\vee m_6\vee m_8\vee m_9\vee m_{10}\vee m_{11}\vee m_{15}$

U sva četiri slučaja utvrdite da li je za realizaciju povoljniji MDNF ili MKNF oblik.

9.13 Nađite MDNF i MKNF oblik za logičku funkciju $Y=M_1M_3M_7M_9M_{11}M_{15}$ koristeći Veitchove dijagrame, uz pretpostavku da sve maksterme zavise od promjenljivih A, B, C i D.

9.14 Data je logička funkcija $Y=m_0\vee m_1\vee m_2\vee m_7\vee m_8\vee m_9\vee m_{10}\vee m_{15}$, pri čemu sve minterme zavise od promjenljivih A, B, C i D.

a) Koristeći Veitchove dijagrame, nađite MDNF i MKNF oblike ove logičke funkcije.

b) Koristeći Veitchove dijagrame, nađite MDNF i MKNF oblike negacije ove logičke funkcije.

9.15 Data je logička funkcija $Y=(\overline{A}\vee\overline{B}\vee C)(\overline{A}\vee C)(B\vee C)$.

a) Koristeći Veitchove dijagrame, nađite MDNF i MKNF oblike ove logičke funkcije.

b) Koristeći Veitchove dijagrame, nađite MDNF i MKNF oblike negacije ove logičke funkcije.

9.16 Nađite MDNF i MKNF oblik za logičku funkciju $Y = \overline{AB}(AC \vee \overline{B}) \vee \overline{A}B\overline{C}$ koristeći Veitchove dijagrame.

9.17 Funkciju $Y = \overline{A}C\,\overline{B \oplus D} \vee \overline{BC[A\overline{C} \vee (B \oplus D)]}$ predstavite Veicthovim dijagramom, a zatim očitajte MDNF i MKNF oblik te logičke funkcije.

9.18* Nađite MDNF i MKNF oblik za logičku funkciju $Y = m_0 \vee m_4 \vee m_{18} \vee m_{19} \vee m_{22} \vee m_{23} \vee m_{25} \vee m_{29}$ uz pretpostavku da sve minterme zavise od promjenljivih A, B, C, D i E. Za minimizaciju koristite Veitchove dijagrame. Koji je od ova dva oblika povoljniji za realizaciju?

9.19* Nađite MDNF i MKNF oblik za logičku funkciju $Y = M_0 M_4 M_{18} M_{19} M_{22} M_{23} M_{25} M_{29}$ uz pretpostavku da sve maksterme zavise od promjenljivih A, B, C, D i E. Za minimizaciju koristite Veitchove dijagrame. Koji je od ova dva oblika povoljniji za realizaciju?

9.20* Za logičku funkciju $Y = \overline{A}CE \vee ABDE \vee A\overline{B}\,\overline{C}E \vee ACE \vee ABD\overline{E} \vee A\overline{B}\,\overline{D}\,\overline{E}$, nađite MDNF i MKNF oblik koristeći Veitchove dijagrame.

9.21** Data je logička funkcija

$Y = m_0 \vee m_1 \vee m_2 \vee m_3 \vee m_4 \vee m_5 \vee m_{20} \vee m_{21} \vee m_{22} \vee m_{23} \vee m_{29} \vee m_{32} \vee m_{33} \vee m_{34} \vee m_{36} \vee m_{37} \vee m_{38}$
$\vee m_{39} \vee m_{40} \vee m_{42} \vee m_{44} \vee m_{46} \vee m_{47} \vee m_{52} \vee m_{53} \vee m_{54} \vee m_{55} \vee m_{61} \vee m_{63}$

pri čemu sve minterme zavise od promjenljivih A, B, C, D, E i F. Koristeći četiri Veitchova dijagrama za funkcije od po 4 promjenljive, nađite MDNF i MKNF oblik ove logičke funkcije.

9.22** Data je logička funkcija

$Y = m_3 \vee m_7 \vee m_{12} \vee m_{14} \vee m_{15} \vee m_{19} \vee m_{23} \vee m_{27} \vee m_{28} \vee m_{29} \vee m_{31} \vee m_{35} \vee m_{39} \vee m_{44} \vee m_{45} \vee m_{46}$
$\vee m_{48} \vee m_{49} \vee m_{50} \vee m_{52} \vee m_{53} \vee m_{55} \vee m_{56} \vee m_{57} \vee m_{59}$

pri čemu sve minterme zavise od promjenljivih A, B, C, D, E i F. Koristeći četiri Veitchova dijagrama za funkcije od po 4 promjenljive, nađite MDNF i MKNF oblik ove logičke funkcije.

9.23 Provjerite rezultate Primjera 6.8 i 6.9 pomoću Veitchovih dijagrama.

9.24 Riješite zadatke 6.17 i 6.19 pomoću Veitchovih dijagrama.

9.25 Riješite zadatak 6.24 pomoću Veitchovih dijagrama.

9.26 Riješite zadatke 6.28 i 6.30 pomoću Veitchovih dijagrama.

9.27 Riješite zadatke 8.14 i 8.15 pomoću Veitchovih dijagrama.

9.28 Riješite zadatak 8.16 pomoću Veitchovih dijagrama.

9.29 Riješite zadatak 8.17 pomoću Veitchovih dijagrama.

9.30 Riješite zadatak 8.18 pomoću Veitchovih dijagrama.

9.31 Riješite zadatak 8.19 pomoću Veitchovih dijagrama.

9.32* Riješite zadatak 8.20 pomoću Veitchovih dijagrama.

9.33* Riješite zadatak 8.25 pomoću Veitchovih dijagrama.

9.34* Na izlazu Y iz nekog digitalnog sklopa sa 5 ulaza A, B, C, D i E javlja se vrijednost 1 ako i samo ako se jedinica nalazi na barem 2 od ukupno 5 ulaza. Formirajte tablicu istine datog sklopa i odredite funkciju koja opisuje rad sklopa prvo u SDNF i SKNF, a zatim i u MDNF i MKNF obliku. Nacrtajte strukturu sklopa koja slijedi iz oblika koji je povoljniji za realizaciju.

9.35* Na izlazu Y iz nekog digitalnog sklopa sa 5 ulaza A, B, C, D i E javlja se vrijednost 1 ako i samo ako se jedinica nalazi na barem 3 od ukupno 5 ulaza. Formirajte tablicu istine datog sklopa i odredite funkciju koja opisuje rad sklopa prvo u SDNF i SKNF, a zatim i u MDNF i MKNF obliku. Nacrtajte strukturu sklopa koja slijedi iz oblika koji je povoljniji za realizaciju.

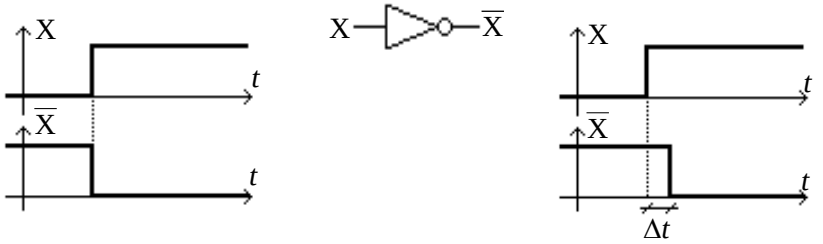
9.36* Na izlazu Y iz nekog digitalnog sklopa sa 5 ulaza A, B, C, D i E javlja se vrijednost 1 ako i samo ako se jedinica nalazi na barem 4 od ukupno 5 ulaza. Formirajte tablicu istine datog sklopa i odredite funkciju koja opisuje rad sklopa prvo u SDNF i SKNF, a zatim i u MDNF i MKNF obliku. Nacrtajte strukturu sklopa koja slijedi iz oblika koji je povoljniji za realizaciju.

9.37* Na izlazu Y iz nekog digitalnog sklopa sa 5 ulaza A, B, C, D i E javlja se vrijednost 1 ako i samo ako se jedinica nalazi na tačno 3 od ukupno 5 ulaza. Formirajte tablicu istine datog sklopa i odredite funkciju koja opisuje rad sklopa prvo u SDNF i SKNF, a zatim i u MDNF i MKNF obliku. Koliko je ulaza u logička kola potrebno za realizaciju ovog sklopa na osnovu dobijenih minimiziranih oblika?

9.38* Na izlazu Y iz nekog digitalnog sklopa sa 5 ulaza A, B, C, D i E javlja se vrijednost 1 ako i samo ako se jedinica nalazi na tačno 4 od ukupno 5 ulaza. Formirajte tablicu istine datog sklopa i odredite funkciju koja opisuje rad sklopa prvo u SDNF i SKNF, a zatim i u MDNF i MKNF obliku. Koliko je ulaza u logička kola potrebno za realizaciju ovog sklopa na osnovu dobijenih minimiziranih oblika?

10. Realizacija logičkih funkcija

Kao što smo već vidjeli, AND, OR i NOT logička kola dovoljna su za realizaciju bilo koje logičke funkcije. Pri tome je, radi uštede u količini neophodnih komponenti, potrebno prvo izvršiti minimizaciju tražene logičke funkcije. Postupci opisani u prethodnim poglavljima omogućavaju svođenje proizvoljne funkcije na MDNF ili MKNF oblik. Mada se često dobijene MDNF ili MKNF mogu dodatno optimizirati primjenom pravila Booleove algebre, takve dodatne optimizacije često nose sa sobom neke *neželjene posljedice*. Jedna od takvih posljedica je ***povećanje kašnjenja izlaza u odnosu na ulaze***. Naime, idealno bi se izlaz iz svakog logičkog kola trebao da promijeni *istog trenutka* nakon što se promijeni vrijednost na ulazu. U stvarnosti, zbog raznih prelaznih pojava unutar samih logičkih kola, promjene vrijednosti na izlazu uvijek *kasne* u odnosu na promjene signala na ulazu. Sljedeća slika prikazuje idealno i stvarno ponašanje *invertora*:



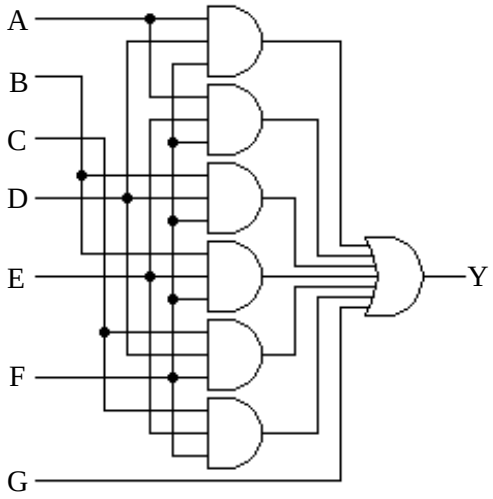
Idealno *Realno*

U navedenom primjeru, promjena izlaznog signala kasni za vremenski iznos Δt u odnosu na promjenu ulaznog signala. Kod savremenih elektronskih logičkih kola, kašnjenje Δt je iznimno malo (manje od milionitog dijela sekunde), ali je uvijek prisutno.

Nije teško uvidjeti da se od svih mreža logičkih kola koje se mogu formirati za datu logičku funkciju minimalno kašnjenje izlaza u odnosu na ulaze postiže u mrežama formiranim na osnovu DNF ili KNF (tipično MDNF ili MKNF). Naime, u takvim mrežama bilo koji signal od ulaza do izlaza prolazi kroz *najviše tri logička kola*. U slučaju DNF, bilo koji signal sigurno prolazi kroz završno OR kolo, a prije toga eventualno kroz AND kolo koje formira odgovarajuću elementarnu konjunkciju, i kroz invertor ukoliko se odgovarajuća promjenljiva pojavljuje sa znakom negacije. Slično vrijedi i za KNF, samo što je uloga AND i OR kola zamijenjena. U svim drugim oblicima logičkih funkcija, ulazni signali često moraju proći kroz više od tri logička kola prije nego što dođu do izlaza, čime se povećava ukupno kašnjenje izlaza u odnosu na ulaze. Posmatrajmo, na primjer, sljedeću logičku funkciju:

$$Y = ADF \vee AEF \vee BDF \vee BEF \vee CDF \vee CEF \vee G$$

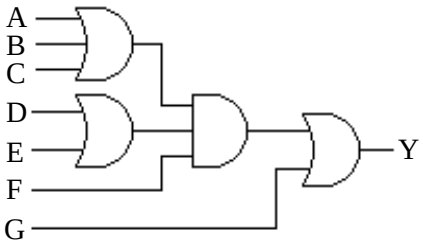
Ova funkcija ima oblik MDNF, a odgovara joj mreža logičkih kola prikazana na sljedećoj slici:



U ovoj mreži svaki ulazni signal prolazi kroz *najviše dva logička kola* prije nego što dođe do izlaza (zapravo kroz tačno dva logička kola). Ukoliko svako logičko kolo unosi isto kašnjenje Δt (što je znatno pojednostavljena pretpostavka), ukupno kašnjenje izlaza u odnosu na ulaz iznosiće $2\Delta t$. S druge strane, lako je provjeriti da se razmatrana logička funkcija može ručno optimizirati u sljedeći oblik (koji nije niti DNF niti KNF oblik):

$$Y = (A \vee B \vee C)(D \vee E)F \vee G$$

Ovoj funkciji odgovara mreža logičkih kola na sljedećoj slici:



Ova mreža je očigledno znatno jednostavnija od prethodne mreže i štedi čak 15 ulaza u logička kola (10 prema 25 u prethodnoj mreži). Međutim, ova mreža je *sporija* od prethodne. Primijetimo da signal sa ulaza G do izlaza prolazi samo kroz jedno logičko kolo, dok signal F prolazi kroz dva logička kola. Međutim, signali A, B, C, D i E prolaze kroz *tri logička kola*. Ukoliko svako logičko kolo unosi kašnjenje Δt, tada maksimalno kašnjenje izlaza u odnosu na promjenu nekog od ulaza može iznositi $3\Delta t$ (maksimalno kašnjenje jednako je *zbiru kašnjenja duž najduže, bolje rečeno* "*najsporije*" *putanje od ulaza ka izlazu*) što je nepovoljnije nego u prethodnoj mreži.

Razumije se da postojanje kašnjenja ograničava maksimalnu brzinu rada sa kojima može raditi mreža logičkih kola, jer interval između dvije dozvoljene promjene nekog od ulaza mora biti duži od trajanja maksimalnog kašnjenja u mreži. Generalno se može zaključiti da je mreža izvedena iz DNF ili KNF ***optimalna u odnosu na brzinu***, mada ne mora biti optimalna u odnosu na složenost mreže. Mreže izvedene iz DNF i KNF obično se nazivaju ***mreže u dva nivoa*** (odnosno ***dvonivoske mreže***), jer signali od ulaza ka izlazu uvijek prolaze kroz najviše dva logička kola, ako ne računamo invertore koji se obično ne broje, jer su u mnogim praktičnim izvedbama digitalnih sklopova često pored vrijednosti promjenljivih na ulazu takođe raspoložive i njihove negacije (kao neovisne promjenljive). Za razliku od ovakvih mreža, mreže kroz kojih signal od ulaza do izlaza može da prođe kroz više od dva kola ne računajući invertore, nazivaju se ***mreže u više nivoa*** (odnosno ***višenivoske mreže***).

Kao još jedan ilustrativan primjer, možemo razmotriti mrežu koja realizira logičku funkciju:

$$Y = CDE \vee ABCE \vee ABCD \vee ABDE$$

Ova funkcija je u MDNF obliku, tako da joj odgovara mreža u dva nivoa sa ukupno 19 ulaza u logička kola. Međutim, ova funkcija se može dodatno optimizirati na razne načine. Jedna mogućnost je, recimo

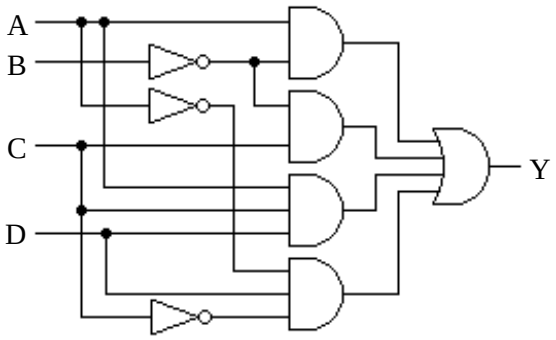
$$Y = CE(D \vee AB) \vee ABD(C \vee E)$$

Realiziramo li ovu logičku funkciju, trebaće nam 15 ulaza u logička kola, čime smo uštedili 4 ulaza. Međutim, tako dobijena mreža biće u četiri nivoa (nacrtajte)! Drugim riječima, uz pretpostavku da sva upotrijebljena logička kola imaju ista kašnjenja, takva dodatno optimizirana mreža imaće dvostruko veće ukupno kašnjenje u odnosu na mrežu dobijenu iz MDNF oblika, odnosno dvostruko je sporija od nje!

Već smo vidjeli da se mreža koja realizira neku logičku funkciju može osjetno pojednostaviti dozvolimo li upotrebu i drugih logičkih kola osim AND, OR i NOT kola, poput EXOR kola. Na primjer, posmatrajmo sljedeću logičku funkciju:

$$Y = A\overline{B} \vee \overline{B}C \vee ACD \vee \overline{A}\,\overline{C}\,D$$

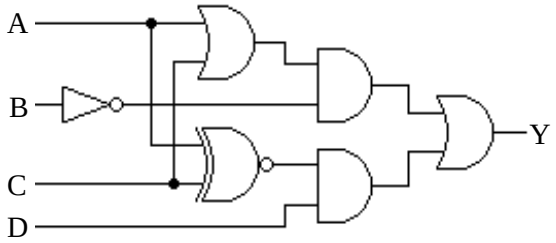
Ovoj funkciji odgovara mreža logičkih kola sa sljedeće slike:



S druge strane, istu funkciju lako možemo napisati u sljedećem obliku:

$$Y = \overline{B}(A \vee C) \vee D(AC \vee \overline{A}\,\overline{C}) = \overline{B}(A \vee C) \vee D\,\overline{A \oplus C}$$

Ovako napisanoj funkciji odgovara sljedeća mreža logičkih kola:



Ova mreža *djeluje* znatno jednostavnije od prethodne, jer koristi manje logičkih kola, i manje ulaza u logička kola. Međutim, prilikom konkretne realizacije, ova prividna jednostavnost je prilično upitna, jer je EXNOR kolo po svojoj unutrašnjoj građi *znatno složenije* nego AND, OR i NOT logička kola. Također, ova mreža je *mreža u tri nivoa* (dok je prethodna bila *u dva nivoa*), i vrlo vjerovatno je *znatno sporija* od prethodne, jer EXNOR kola obično imaju znatno veće kašnjenje izlaza u odnosu na ulaz nego AND, OR i NOT kola.

Upotreba NAND i NOR kola, zajedno sa AND, OR i NOT logičkim kolima pruža također interesantne mogućnosti za realizaciju logičkih funkcija, tim prije što su NAND i NOR logička kola tipično podjednako brza kao i standardna AND, OR i NOT logička kola. Posmatrajmo, na primjer, sljedeću logičku funkciju:

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

Ova funkcija ima MDNF oblik, i zahtijeva 13 ulaza za realizaciju. Njen neznatno modificirani ekvivalentni oblik

$$Y = AB\overline{C} \vee (\overline{A} \vee \overline{B})C$$

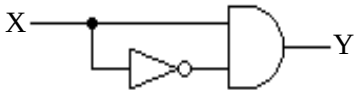
zahtijeva jedan ulaz manje za realizaciju. Međutim, primjenom De Morganovih pravila, ova funkcija se može napisati i kao

$$Y = AB\overline{C} \vee \overline{AB}C$$

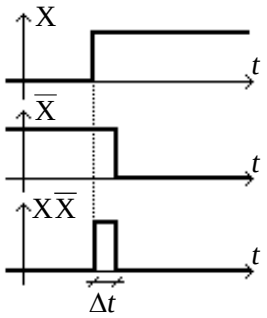
što zahtijeva ukupno 10 ulaza (provjerite), ukoliko pretpostavimo da nam je na raspolaganju NAND logičko kolo. Ipak, može se primijetiti da će dobijena mreža biti u tri nivoa. Može se još primijetiti i da se posljednji dobijeni oblik funkcije lako može napisati u obliku $Y = AB \oplus C$, što zahtijeva svega 4 ulaza, ali i znatno sporije EXOR kolo. Vidimo da realizacija logičke funkcije može bitno zavisiti od toga šta imamo na raspolaganju, i kakvi se zahtjevi postavljaju sa aspekta neophodne brzine rada. Još jednom treba napomenuti da za transformacije poput prikazanih ne postoje nikakva sistematična pravila (mada ćemo u Poglavlju 11. vidjeti da Veitchovi dijagrami ponekad mogu ukazati na mogućnost efikasne primjene EXOR i EXNOR logičkih kola).

Sa današnjim stanjem raspoložive tehnologije, brzina rada neke mreže često je *kritičniji faktor od njene složenosti*. Zbog svih navedenih razloga, kao i zbog činjenice da PLA strukture koje ćemo kasnije razmotriti omogućavaju efikasnu realizaciju funkcija predstavljenih u MDNF ili MKNF obliku, obično se u praksi dodatna optimizacija MDNF ili MKNF formi provodi samo u izuzetnim slučajevima, kada za to postoje neki specifični razlozi.

Kašnjenje koje neizbježno unose sva logička kola može da dovede do veoma neugodne pojave kratkotrajnih ***lažnih vrijednosti*** na izlazu u trenucima kada neki od ulaza mijenja vrijednost. Posmatrajmo prvo krajnje pojednostavljen primjer, prikazan sklopom na sljedećoj slici:

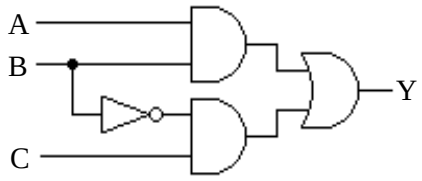


Na osnovu zakona logičke algebre slijedi da bi izlaz iz ovog sklopa trebao uvijek da bude jednak nuli, s obzirom da je $Y = X\overline{X} = 0$. Međutim, sljedeće rezonovanje će nas uvjeriti da stvarnost često "nema sluha" za matematičke zakonitosti. Pretpostavimo da je do nekog trenutka $t = t_0$ signal X bio na vrijednosti 0, nakon čega prelazi na vrijednost 1. Izlaz iz invertora će tada promijeniti vrijednost sa vrijednosti 1 na vrijednost 0, ali ne u trenutku $t = t_0$, već tek u trenutku $t = t_0 + \Delta t$, gdje je Δt kašnjenje koje unosi invertor. Slijedi da su u intervalu od $t = t_0$ do $t = t_0 + \Delta t$ oba ulaza u AND kolo jednaka jedinici, zbog čega je u tom intervalu Y = 1! Na izlazu se, dakle, javlja kratkotrajna lažna vrijednost. Ovo je ilustrativno prikazano na sljedećoj slici:



Strogo rečeno, prikazana slika nije u potpunosti tačna, nego je izlaz još dodatno pomjeren udesno, zbog činjenice da i samo AND kolo također unosi neko kašnjenje. Ipak, slika dovoljno jasno ilustrira činjenicu koju smo željeli da istaknemo, a to je da u realnim uvjetima zakon neprotivurječnosti, koji tvrdi da je $X\overline{X} = 0$, može kratkotrajno da bude narušen (isto vrijedi i za zakon isključenja trećeg, koji tvrdi da je $X \vee \overline{X} = 1$).

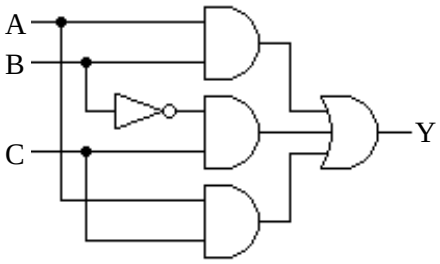
Prikazani primjer ne djeluje previše zabrinjavajuće. Naime, može izgledati da nema nikakve potrebe za kreiranjem sklopova poput prethodnog, s obzirom da bi njegov očekivani izlaz trebao uvijek da bude nula (mada se nekada kreiraju upravo sklopovi poput prethodnog, nazvani ***digitalni diferencijatori***, čiji je cilj generiranje kratkotrajnih impulsa na izlazu u trenucima kada ulaz mijenja vrijednost). Međutim, kašnjenja mogu uzrokovati neočekivane efekte i u mnogo realističnijim sklopovima. Posmatrajmo, na primjer, sljedeću mrežu:



Ovoj mreži očigledno odgovara logička funkcija $Y = AB \vee \overline{B}C$. Stoga, kada su svi ulazi na jedinici, tj. kada je $A = B = C = 1$, izlaz Y je također na jedinici, jer je $AB = 1$. Pretpostavimo da u nekom trenutku ulaz B promijeni vrijednost sa 1 na 0. Izlaz bi *trebao da zadrži* vrijednost 1 prema gore napisanoj formuli, jer mada prva elementarna konjunkcija AB tada postaje 0, druga elementarna konjunkcija u kojoj se javlja negacija promjenljive B postaje 1, tako da bi izlaz trebao da bude jednak jedinici. Međutim, u stvarnosti, izlaz Y će nakratko poprimiti vrijednost 0! Naime, kada B promijeni vrijednost sa 1 na 0, izlaz iz invertora će promijeniti vrijednost sa 0 na 1 tek nakon izvjesnog kašnjenja Δt. Za to vrijeme, na oba AND kola jedan od ulaza jednak je nuli, pa su njihovi izlazi a samim tim i izlaz Y jednaki nuli! Tek nakon što invertor promijeni svoj izlaz, Y će se vratiti na vrijednost 1.

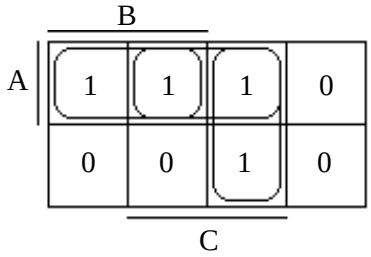
Strogo uzevši, provedeno razmatranje nije sasvim tačno, jer smo razmatrali samo kašnjenje koje unosi invertor, a zanemarili u potpunosti kašnjenja koja unose ostala logička kola. Međutim, nije teško pokazati da razmatranje ostaje u snazi i ako pretpostavimo da oba AND kola imaju približno jednaka kašnjenja. S druge strane, ukoliko gornje AND kolo unosi mnogo veće kašnjenje nego donje, njegov će se izlaz prebaciti na nulu tek nakon što izlaz iz donjeg kola već dobije ispravnu vrijednost, tako da će izlaz zadržati ispravnu vrijednost, i lažna vrijednost se neće pojaviti. Drugim riječima, *rizik pojave lažne vrijednosti u ovoj mreži postoji*, a da li će do njega zaista doći, zavisi od konkretne vrijednosti kašnjenja koja unose pojedina logička kola. Stoga se logičke mreže u kojima postoji mogućnost pojave kratkotrajnih lažnih vrijednosti na izlazu u trenucima kada signal na nekom od ulaza mijenja svoju vrijednost, nazivaju se ***rizične mreže*** ili ***mreže sa hazardom***.

Pojava lažnih vrijednosti na izlazu obično traje toliko kratko da najčešće ne pravi veće probleme. Ipak, ova pojava može presudno da djeluje na rad sekvencijalnih mreža, koje ćemo razmatrati kasnije. U sekvencijalnim mrežama problem rizika se obično rješava uvođenjem tzv. *sinhronizacionih impulsa*, o kojima ćemo kasnije detaljno govoriti. Međutim, u mrežama sa dva nivoa hazard (rizik) je lako moguće otkloniti uvođenjem dodatnih logičkih kola. Posmatrajmo, na primjer, sljedeću mrežu koja realizira logičku funkciju $Y = AB \vee AC \vee \overline{B}C$ koja je, kao što smo već ranije vidjeli, ekvivalentna prethodnoj funkciji (teorema o konsenzusu):



Lako je vidjeti da se u ovoj mreži lažne vrijednosti izlaza ne javljaju. Naime, kada nakon vrijednosti ulaza $A = B = C = 1$ ulaz B promijeni vrijednost na 0, dodatni član AC zadržava vrijednost 1, čime održava izlaz na jedinici tokom spornog perioda u kojem invertor još uvijek nije "odradio svoj posao". Dakle, prikazana mreža je, mada složenija, *mreža bez rizika*. Ovaj primjer demonstrira da se dvije mreže koje su *formalno ekvivalentne* sa aspekta ekvivalentnosti logičkih funkcija koje predstavljaju ne moraju biti ekvivalentne sa aspekta ponašanja, zbog neizbježnih kašnjenja koja unose logička kola. Ova činjenica znatno otežava kako analizu, tako i projektiranje logičkih mreža (u prisustvu rizika, neki od zakona logičke algebre poput zakona neprotivurječnosti i zakona isključenja trećeg, kao što smo već vidjeli, ne smiju se bezuvjetno koristiti).

Opisanu ideju za uklanjanje hazarda lako je generalizirati za proizvoljne mreže sa dva nivoa. Naime, potrebno je za svaku od promjenljivih koja se u funkciji javlja kako sa negacijom tako i bez negacije obezbijediti da se u funkciji nalazi *barem jedna prosta implikanta koja ne sadrži tu promjenljivu* (čak i ukoliko će ona biti višak sa aspekta optimalnosti), ukoliko takva postoji. Takve implikante veoma je lako naći uz pomoć Veitchovih dijagrama, kao što je prikazano na primjeru Veitchovog dijagrama za razmatranu funkciju (još jednom napominjemo da izabrane konture *nisu optimalne*, nego su formirane sa ciljem *uklanjanja rizika*):



Tako, na primjer, implikanta AC (koja je suvišna sa aspekta optimalnosti) ne sadrži promjenljivu B, koja se u preostale dvije proste implikante jedanput javlja sa negacijom, a drugi put bez nje. Generalno, da bi mreža bila bez rizika, za *svaki par susjednih jedinica u Veitchovom dijagramu mora postojati barem jedna kontura koja ih prekriva*.

Bitno je napomenuti da predloženi postupak za uklanjanje rizika sprečava samo pojavu eventualnih lažnih vrijednosti koje mogu nastati u trenutku kada *jedan od ulaznih signala* mijenja vrijednost. Kratkotrajne lažne vrijednosti su i dalje moguće u slučajevima kada *više od jednog ulaznog signala* istovremeno mijenja vrijednost (pogledati npr. Zadatak 10.14), i takve lažne vrijednosti je veoma teško, a ponekad čak i nemoguće ukloniti. Zbog toga, u svim digitalnim sistemima treba strogo izbjegavati istovremeno mijenjanje vrijednosti više od jedne ulazne promjenljive. Srećom, u praksi se potreba za takvim promjenama javlja relativno rijetko. Zbog toga se analiza rizika uvijek vrši samo u odnosu na promjenu *jedne od ulaznih promjenljivih*.

U mrežama sa više nivoa (poput mreže iz Primjera 6.3) mogu postojati različiti putevi kuda signal može stići od nekog od ulaza do izlaza, koji mogu prolaziti kroz različit broj različitih logičkih kola, i prema tome trpiti različit iznos kašnjenja. Takvi signali mogu na više mjesta međudjelovati sami sa svojim zakašnjelim vrijednostima (koji su putovali drugim putem), što može dovesti do prilično nezgodnih efekata. Zbog toga, mreže sa više nivoa često posjeduju vrlo visok stepen rizika, koji se može očitovati i tako da izlazni signal u kratkom vremenskom periodu i po nekoliko puta promijeni svoju vrijednost nakon promjene nekog od ulaznih signala prije nego što se ustali na pravoj vrijednosti (tzv. ***dinamički rizik***). Nažalost, za uklanjanje rizika u mrežama sa više nivoa ne postoje nikakvi jednostavni sistematični postupci kao što je opisani postupak za mreže sa dva nivoa. Ovo je još jedan razlog koji ide u prilog tome da mreže sa više nivoa treba izbjegavati. Slično vrijedi i za mreže u kojima se javlja mnogo različitih tipova logičkih kola, poput EXOR kola (pogledati npr. Zadatak 10.15).

U digitalnoj integriranoj tehnici, iz čisto tehničkih razloga najpovoljnije je neku logičku funkciju realizirati preko *samo jedne vrste logičkih kola*, npr. preko NAND kola, jer je tehnološki mnogo lakše na malom prostoru proizvesti veći broj istih kola, nego manji broj različitih kola. Rješenje koje bismo dobili zamjenom svakog od AND, OR ili NOT logičkih kola njihovim ekvivalentnim realizacijama preko NAND kola bilo bi veoma neekonomično, jer bi sadržavalo mnogo veći broj kola nego što je zaista potrebno. Zbog toga su razvijeni jednostavni sistematični postupci koji omogućavaju da se proizvoljna logička funkcija realizira samo pomoću NAND ili samo pomoću NOR logičkih kola. Ovi postupci tipično polaze od DNF ili KNF zadane funkcije (po mogućnosti MDNF ili MKNF), mada se mogu lako generalizirati i za funkcije proizvoljnog oblika, iako za tim nema prevelike potrebe.

Da bismo realizirali neku funkciju samo pomoću NAND kola, najbolje je krenuti od njene *minimalne disjunktivne normalne forme* i izvršiti *dvostruku negaciju*. Na unutrašnju negaciju tada treba primijeniti De Morganova pravila, i po potrebi iskoristiti činjenicu da je $\overline{A} = \overline{AA}$ (da bi se oslobodili potrebe za invertorima). Postupak je najbolje ilustrirati na konkretnom primjeru:

Primjer 10.1:

- Realizirati funkciju $Y = AB\overline{C} \vee \overline{A}C \vee BD \vee A\overline{C}\,\overline{D} \vee C$ samo pomoću NAND kola.

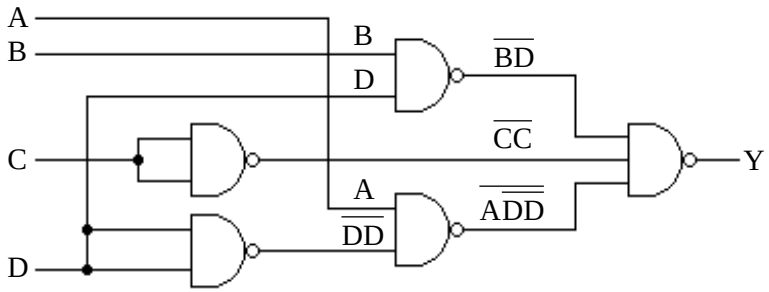
Nađimo prvo MDNF za datu funkciju. To smo već uradili u Primjeru 9.7 iz prethodnog poglavlja, gdje smo dobili slijedeći oblik:

$$Y = A\overline{D} \vee BD \vee C$$

Primjenom dvostruke negacije i elementarnih transformacija dobijamo:

$$Y = \overline{\overline{A\overline{D} \vee BD \vee C}} = \overline{\overline{A\overline{D}}\,\overline{BD}\,\overline{C}} = \overline{\overline{A\overline{DD}}\,\overline{BD}\,\overline{CC}}$$

Dobiveni oblik je neposredno pogodan za realizaciju pomoću NAND kola, kao što je prikazano na sljedećoj shemi:

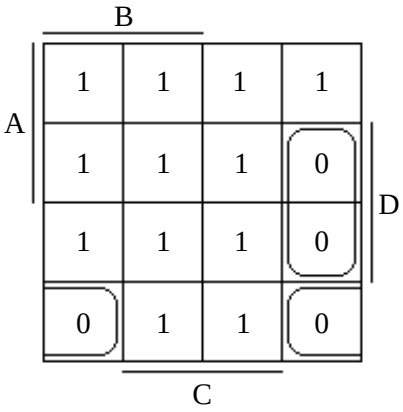


Da bismo izvršili realizaciju neke funkcije pomoću NOR kola, koristimo analogan postupak, samo polazimo od minimalne konjunktivne umjesto disjunktivne normalne forme.

Primjer 10.2:

- Realizirati funkciju $Y = A\overline{D} \vee BD \vee C$ pomoću NOR kola.

Prva stvar koju trebamo uraditi je da očitamo MKNF za ovu funkciju, što možemo uraditi pomoću sljedećeg dijagrama:



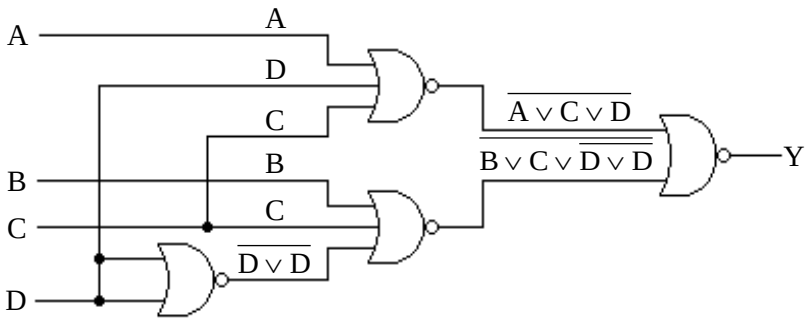
Iz prikazanog dijagrama direktno očitavamo MKNF za zadanu funkciju:

$$Y = (B \vee C \vee \overline{D})(A \vee C \vee D)$$

Primjenom dvostruke negacije i elementarnih transformacija dobijamo:

$$Y = \overline{\overline{(B \vee C \vee \overline{D})(A \vee C \vee D)}} = \overline{\overline{B \vee C \vee \overline{D}} \vee \overline{A \vee C \vee D}} =$$
$$= \overline{\overline{B \vee C \vee \overline{\overline{D \vee D}}} \vee \overline{A \vee C \vee D}}$$

Dobijeni oblik je neposredno pogodan za realizaciju pomoću NOR kola, kao što je prikazano na sljedećoj shemi:



Ponekad se dešava da je MDNF mnogo kraća od MKNF za istu funkciju, ili obratno. Zbog toga su razvijeni postupci da se iz MDNF dobije realizacija pomoću NOR kola, odnosno iz MKNF realizacija pomocu NAND kola. Ideja postupka je da prvo negiramo funkciju, zatim da realiziramo njenu negaciju, i na kraju da negiramo rezultat.

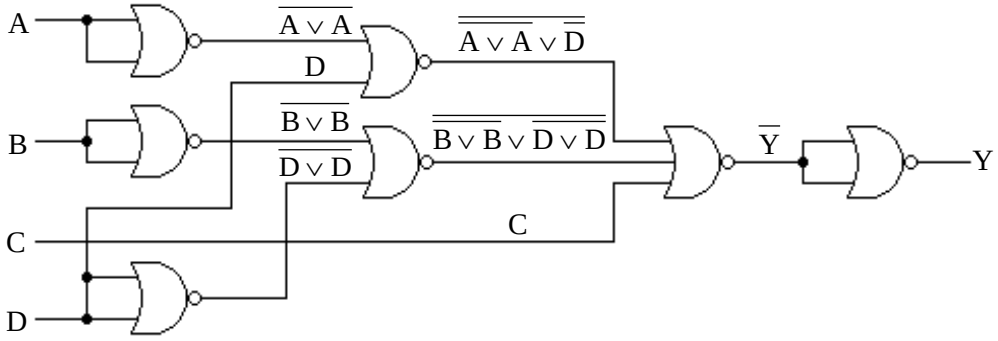
Primjer 10.3:

- Realizirati funkciju $Y = A\overline{D} \vee BD \vee C$ pomoću NOR elemenata bez njene prethodne transformacije na oblik MKNF.

Ukoliko negiramo funkciju Y, a zatim primijenimo elementarne transformacije, možemo pisati:

$$\overline{Y} = \overline{A\overline{D} \vee BD \vee C} = \overline{A\overline{D}}\,\overline{BD}\,\overline{C} = (\overline{A} \vee D)(\overline{B} \vee \overline{D})\overline{C} = \overline{\overline{(\overline{A} \vee D)(\overline{B} \vee \overline{D})\overline{C}}} =$$
$$= \overline{\overline{\overline{A} \vee D} \vee \overline{\overline{B} \vee \overline{D}} \vee C} = \overline{\overline{A \vee A \vee D} \vee \overline{B \vee B \vee D \vee D} \vee C}$$

Dobijeni oblik funkcije $\overline{Y}$ je pogodan za neposrednu realizaciju pomoću NOR elemenata. Da bismo dobili funkciju Y, dovoljno je samo negirati izlaz, kao što je izvedeno na sljedećoj slici:



Ukoliko pažljivo razmotrimo provedene postupke, lako se možemo uvjeriti da vrijedi sljedeća teorema:

Teorema 10.1:

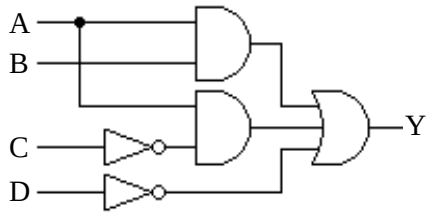
Sklop realiziran na bazi ma kakve disjunktivne normalne forme u kojoj svaka elementarna konjunkcija sadrži barem dvije promjenljive ne mijenja svoju funkciju ukoliko izlazno OR kolo i sva AND kola zamijenimo sa NAND kolima, a invertore zamijenimo sa NAND kolima sa spojenim ulazima. Također, sklop realiziran na bazi ma kakve konjunktivne normalne forme u kojoj svaka elementarna disjunkcija sadrži barem dvije promjenljive ne mijenja svoju funkciju ukoliko izlazno AND kolo i sva OR kola zamijenimo sa NOR kolima, a invertore zamijenimo sa NOR kolima sa spojenim ulazima.

Teorema ne vrijedi ukoliko se neka od elementarnih konjunkcija ili disjunkcija svode samo na jednu promjenljivu, ili njenu negaciju. Lako je vidjeti da u tom slučaju takva promjenljiva ide direktno na izlazno OR kolo (za slučaj MDNF) odnosno AND kolo (za slučaj MKNF), prolazeći eventualno samo kroz invertor, a ne prolazeći kroz AND (odnosno OR) kolo prethodnog nivoa. Da bi se Teorema 10.1 mogla primijeniti i na ovakve slučajeve, potrebno je prethodno izvršiti inverziju svake takve promjenljive, tj. negirati je ako nije bila negirana a ukloniti joj negaciju ukoliko nije bila negirana, nakon čega se primjenjuje isti postupak kao u Teoremi 10.1. Valjanost ovog postupka veoma je lako dokazati.

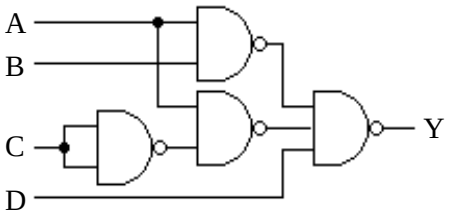
Primjer 10.4:

- Realizirati funkciju $Y = AB \vee A\overline{C} \vee \overline{D}$ pomoću NAND logičkih kola koristeći Teoremu 10.1.

Kada bismo ovu funkciju realizirali pomoću AND, OR i NOT logičkih kola dobili bismo sljedeći sklop:



Očigledno, uvjeti Teoreme 10.1 nisu ispunjeni, zbog prisustva jednočlane elementarne konjunkcije (negacije promjenljive D). Stoga ćemo primijeniti opisani modificirani postupak. Drugim riječima, invertiraćemo promjenljivu D tako što ćemo joj ukloniti negaciju, a zatim zamijeniti sva AND kola i OR kolo NAND kolima, a preostali invertor (za promjenljivu C) NAND kolom sa spojenim ulazima. Na taj način dobijamo sklop sa sljedeće slike:



(?) Pitanja i zadaci

10.1 Objasnite kako struktura neke logičke mreže utiče na kašnjenje vrijednosti na izlazu iz mreže u odnosu na trenutak promjene vrijednosti na nekom od ulaza u mrežu.

10.2 Objasnite zbog čega su logičke mreže realizirane na bazi DNF odnosno KNF optimalne sa aspekta brzine rada.

10.3 Šta su dvonivoske a šta višenivoske mreže? Koje su prednosti jednih, a koje drugih vrsta logičkih mreža?

10.4 Razmotrimo logičku funkciju $Y = CDE \vee ABCE \vee ABCD \vee ABDE$ u MDNF obliku, i njenu optimiziranu (sa aspekta složenosti) verziju $Y = CE(D \vee AB) \vee ABD(C \vee E)$. Ukoliko sva upotrijebljena logička kola unose kašnjenje od $\Delta t = 10\,\text{ns}$, koliko je maksimalno kašnjenje izlaza u odnosu na ulaz za polaznu, a koliko za optimiziranu mrežu?

10.5 Transformirajte (intuitivnim putem) sljedeće funkcije, zadane u MDNF obliku, na oblike koje omogućavaju što je god moguće ekonomičniju realizaciju pomoću AND i OR logičkih kola, uz pretpostavku da je dozvoljeno koristiti mreže u više nivoa. Za svaki od slučaja odredite broj neophodnih ulaza u logička kola i broj nivoa mreže.

a) $Y = ABCD \vee ABDE \vee F$ b) $Y = ACD \vee ABE \vee BC \vee BD$

c) $Y = ADE \vee BDE \vee AC \vee BC$ d) $Y = AD \vee AE \vee AF \vee BD \vee BE \vee CD \vee CE$

e) $Y = ACF \vee ACG \vee ADF \vee ADG \vee BCF \vee BCG \vee BDF \vee BDG \vee EF \vee EG$

10.6 Transformirajte logičku funkciju $Y = A\overline{B}\,\overline{C} \vee \overline{A}\,CD \vee \overline{B}D$ u oblik koji omogućava što je god moguće ekonomičniju realizaciju uz pomoć AND, OR i NOT logičkih kola, uz pretpostavku da je dozvoljeno koristiti mreže u više nivoa.

10.7 Nađite oblike sljedećih logičkih funkcija koji omogućavaju najekonomičniju realizaciju, uz pretpostavku da je pored osnovnih logičkih kola dozvoljena i upotreba EXOR i EXNOR logičkih kola.

a) $Y = A\overline{C}(B \oplus D) \vee \overline{AC \vee \overline{A}C\overline{B \oplus D}}$ b) $Y = \overline{A\overline{C}(B \oplus D)} \vee BC(\overline{A}C \vee \overline{B \oplus D})$

c) $Y = \overline{AC \vee A\overline{C}(B \oplus D)} \vee \overline{A}C\overline{B \oplus D}$ d) $Y = \overline{\overline{AC} \vee A\overline{C}(B \oplus D)} \vee \overline{A}C\overline{B \oplus D}$

10.8 Na izlazu Y iz nekog digitalnog sklopa sa 4 ulaza A, B, C i D javlja se vrijednost 1 ako i samo ako se jedinica nalazi na neparno mnogo ulazâ (1 ili 3). Formirajte tablicu istine datog sklopa i odredite pripadnu funkciju u MDNF i MKNF obliku. Šta se može zaključiti? Kako biste pojednostavili ovu funkciju ukoliko je dozvoljena i upotreba ekskluzivne disjunkcije?

10.9* Na izlazu Y iz nekog digitalnog sklopa sa 4 ulaza A, B, C i D javlja se jedinica ako i samo ako se jedinica nalazi na tačno dva od 4 njegova ulaza (na bilo koja dva). Formirajte tablicu istine datog sklopa, odredite funkciju koja opisuje njegov rad u MDNF i MKNF obliku, a zatim odredite najjednostavniji mogući oblik ove funkcije uz upotrebu ekskluzivne disjunkcije.

10.10 Objasnite na koji način kašnjenja koja unose logička kola mogu dovesti do pojave kratkotrajnih lažnih informacija na izlazima iz digitalnih sklopova.

10.11 Objasnite šta su logičke mreže sa rizikom, i na koji način se rizik može otkloniti.

10.12 Posmatrajmo mrežu koja realizira logičku funkciju $Y = AB \vee \overline{B}C$. Neka za $t < t_0$ vrijedi $A = B = C = 1$, i neka u trenutku $t = t_0$ ulaz B promijeni vrijednost na $B = 0$. Pokažite (crtanjem vremenskih dijagrama) da će u slučaju da sva logička kola imaju isto kašnjenje Δt doći do kratkotrajnog lažnog impulsa na izlazu, a da do toga neće doći ukoliko AND kolo koje realizira član AB ima 3 puta veće kašnjenje ($3\Delta t$). Koliko minimalno treba iznositi kašnjenje tog AND kola u odnosu na preostala kola da se ne pojavi lažni impuls?

10.13 Prikažite sljedeće logičke funkcije u obliku koji omogućava realizaciju pomoću mreže bez rizika:

a) $Y = \overline{A}\,C \vee B\overline{C}$ b) $Y = (A \vee B)(\overline{B} \vee C)$

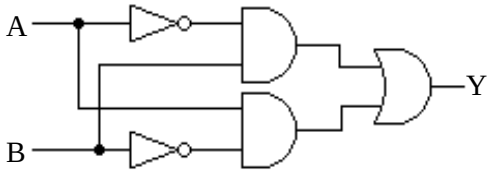
10.14 Prikažite sljedeće logičke funkcije u obliku koji omogućava realizaciju pomoću mreže bez rizika, uz pretpostavku da sve minterme odnosno maksterme ovise od promjenljivih A, B, C i D:

a) $Y = m_0 \vee m_4 \vee m_5 \vee m_6 \vee m_7 \vee m_9 \vee m_{11} \vee m_{13} \vee m_{14}$

b) $Y = M_2\,M_4\,M_6\,M_{10}\,M_{11}\,M_{12}\,M_{14}$

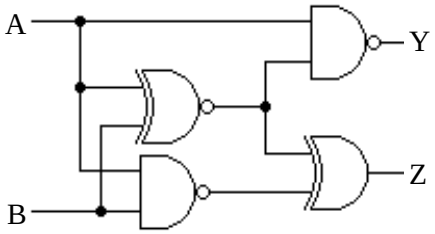
10.15* Prikažite logičku funkciju $Y = m_0 \vee m_1 \vee m_3 \vee m_4 \vee m_7 \vee m_{11} \vee m_{12} \vee m_{15} \vee m_{16} \vee m_{17} \vee m_{20} \vee m_{28}$ u obliku koji omogućava realizaciju pomoću mreže bez rizika, uz pretpostavku da sve minterme maksterme ovise od promjenljivih A, B, C, D i E.

10.16 Posmatrajmo sljedeću logičku mrežu:

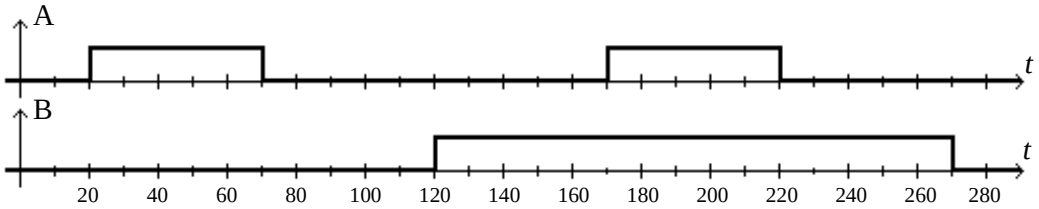


Ova mreža bi trebala da realizira logičku funkciju $Y = \overline{A}B \vee A\overline{B}$, odnosno $Y = A \oplus B$. Stoga bi trebalo biti $Y = 1$ i kad je $A = 0$ i $B = 1$, i kad je $A = 1$ i $B = 0$. Pretpostavimo da sva upotrijebljena logička kola imaju isto kašnjenje Δt. Neka do trenutka $t = t_0$ vrijedi $A = 0$ i $B = 1$, i neka u trenutku $t = t_0$ oba ulaza A i B istovremeno promijene vrijednosti na $A = 1$ i $B = 0$. Nacrtajte vremenske dijagrame koji prikazuju šta se dešava u pojedinim tačkama mreže, i uvjerite se da u jednom trenutku dolazi do pojave kratkotrajne lažne vrijednosti $Y = 0$ na izlazu.

10.17 Data je logička mreža sa dva izlaza, kao na sljedećoj slici:

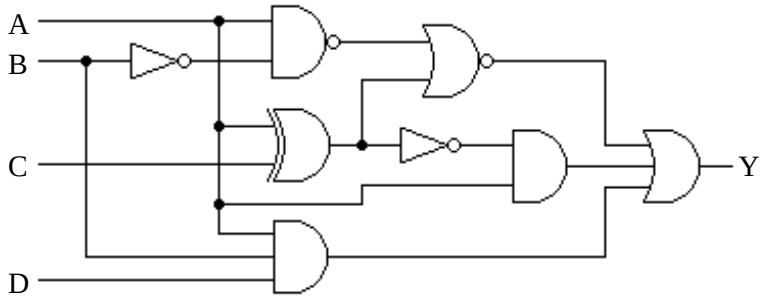


Utvrdite prvo koje bi logičke funkcije ova mreža trebala da realizira (u što je god moguće jednostavnijem obliku). Pretpostavimo sada da se na ulaze A i B dovedu signali čiji je vremenski dijagram promjene prikazan na sljedećoj slici (vrijeme na osi *t* dato je u nanosekundama):



Nacrtajte vremenske dijagrame signala u karakterističnim tačkama mreže, kao i na izlazu iz mreže, uz pretpostavku da NAND logička kola unose kašnjenje od $\Delta t = 10$ ns, a EXOR i EXNOR logičko kolo kašnjenje od $\Delta t = 20$ ns.Ustanovite da li mreža ispravno obavlja zadanu funkcionalnost. Šta se može primijetiti?

10.18 Dat je logički sklop kao na sljedećoj slici:



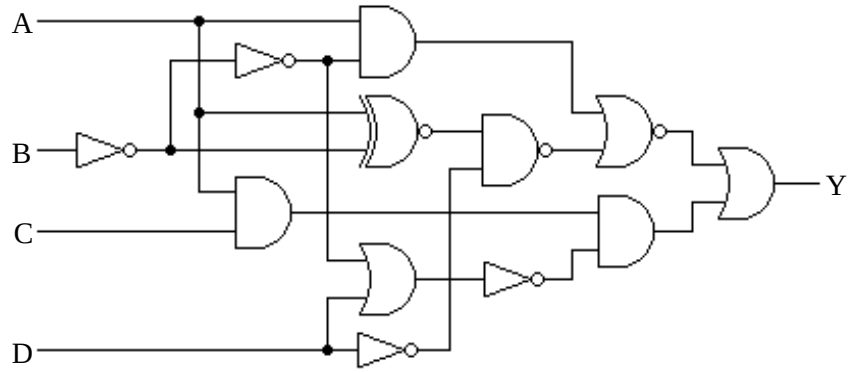
a) Odredite koju logičku funkciju obavlja dati sklop, a zatim nacrtajte što je god moguće jednostavniju mrežu u dva nivoa koja bi trebalo da logički obavlja istu funkcionalnost (po potrebi koristite Veitchove dijagrame).

b) Kako se mreža dobijena pod a) može dodatno optimizirati ukoliko se dopusti upotreba mreža sa više nivoa?

c) Ukoliko sva logička kola unose kašnjenje od $\Delta t = 10\,\text{ns}$, osim EXOR logičkog kola koje unosi kašnjenje od $\Delta t = 20\,\text{ns}$, koliko iznosi maksimalno kašnjenje vrijednosti na izlazu u odnosu na trenutak promjene nekog od ulaznih signala za polaznu mrežu?

d) Uz iste uvjete kao u prethodnom pitanju, koliko iznosi maksimalno kašnjenje vrijednosti na izlazu u odnosu na trenutak promjene nekog od ulaznih signala za mrežu nastalu nakon obavljenih pojednostavljenja izvršenih pod a) odnosno pod b)?

10.19 Dat je logički sklop kao na sljedećoj slici:



a) Odredite koju logičku funkciju obavlja dati sklop, a zatim nacrtajte što je god moguće jednostavniju mrežu u dva nivoa koja bi trebalo da logički obavlja istu funkcionalnost, pri čemu je dozvoljeno samo korištenje standardnih AND, OR i NOT logička kola (po potrebi koristite Veitchove dijagrame).

b) Kako se mreža dobijena pod a) može dodatno optimizirati ukoliko se dopusti upotreba drugih vrsta logičkih kola?

c) Ukoliko sva logička kola unose jednako kašnjenje od $\Delta t = 10\,\text{ns}$, koliko iznosi maksimalno kašnjenje vrijednosti na izlazu u odnosu na trenutak promjene nekog od ulaznih signala za polaznu mrežu?

d) Ukoliko sva logička kola unose kašnjenje od $\Delta t = 10\,\text{ns}$, osim EXNOR logičkog kola koje unosi kašnjenje od $\Delta t = 25\,\text{ns}$, koliko iznosi maksimalno kašnjenje vrijednosti na izlazu u odnosu na trenutak promjene nekog od ulaznih signala za polaznu mrežu?

e) Koliko iznosi maksimalno kašnjenje vrijednosti na izlazu u odnosu na trenutak promjene nekog od ulaznih signala za mrežu nastalu nakon obavljenih pojednostavljenja pod a) i b), pod uvjetima navedenim pod c) odnosno d)?

10.20 Data je logička funkcija $Y = \overline{A}\,\overline{B}C \vee AC \vee AD$.

a) Realizirajte datu funkciju samo uz pomoć NAND logičkih kola, koristeći prethodno svođenje funkcije na MDNF oblik.

b) Realizirajte datu funkciju samo uz pomoć NOR logičkih kola, koristeći prethodno svođenje funkcije na MKNF oblik.

c) Realizirajte datu funkciju samo uz pomoć NOR logičkih kola, negirajući njenu negaciju.

d) Realizirajte datu funkciju samo uz pomoć NAND logičkih kola, negirajući njenu negaciju.

10.21 Transformirajte sljedeće funkcije na oblike koje omogućavaju realizaciju pomoću NAND odnosno NOR kola, tako što ćete ih prethodno prvo svesti na MDNF odnosno MKNF oblik:

a) $Y = A\overline{B}\,\overline{C} \vee \overline{A}\,C \vee \overline{A}\,B$

b) $Y = (A \vee B \vee C)(A \vee \overline{B})(\overline{A} \vee \overline{C})$

c) $Y = \overline{A}B \vee A \vee \overline{C} \vee \overline{D}$

d) $Y = \overline{AB}\,\overline{AC}$

e) $Y = \overline{AB \vee \overline{A}\,\overline{C}}$

f) $Y = (A \vee B)(\overline{A} \vee C)$

g) $Y = \overline{(A \vee B)(\overline{A} \vee C)}$

h) $Y = \overline{A \vee B}\,\overline{\overline{A} \vee C}$

i) $Y = (A \vee B)(\overline{A} \vee C \vee D)(\overline{A} \vee \overline{C})$

j) $Y = (A \vee B)(\overline{B} \vee C)(\overline{A} \vee \overline{C})$

10.22 Transformirajte sljedeće funkcije na oblike koje omogućavaju realizaciju pomoću NAND odnosno NOR kola, tako što ćete ih prethodno prvo svesti na MDNF odnosno MKNF oblik:

a) $Y = A\overline{C}(B \oplus D) \vee \overline{AC} \vee \overline{A}C\overline{B \oplus D}$

b) $Y = \overline{A\overline{C}(B \oplus D)} \vee BC(\overline{A}C \vee \overline{B \oplus D})$

c) $Y = \overline{AC \vee A\overline{C}(B \oplus D)} \vee \overline{A}C\overline{B \oplus D}$

d) $Y = \overline{\overline{AC} \vee A\overline{C}(B \oplus D)} \vee \overline{\overline{A}C\overline{B \oplus D}}$

10.23 Transformirajte sljedeće funkcije na oblike koje omogućavaju realizaciju pomoću NAND odnosno NOR kola, tako što ćete ih prethodno prvo svesti na MDNF odnosno MKNF oblik.

a) $Y = \overline{\overline{ABC \vee D} \vee \overline{ACD} \vee B}$

b) $Y = \overline{\overline{A\overline{B}C \vee \overline{D}} \vee \overline{\overline{ACD} \vee B}}$

c) $Y = \overline{\overline{A\overline{B}\,\overline{C} \vee D} \vee \overline{\overline{A}CD \vee \overline{B}}}$

d) $Y = \overline{\overline{A\overline{B} \vee \overline{C}D} \vee \overline{\overline{A}C \vee \overline{B}D}}$

10.24* Pronađite oblik funkcije iz zadatka 10.23 pod c) koji omogućava optimalnu realizaciju uz pomoć NOR kola, pod uvjetom da je dozvoljeno koristiti povećani broj nivoa logičke mreže.

10.25 Transformirajte sljedeće funkcije na oblike koje omogućavaju realizaciju pomoću NAND odnosno NOR kola, tako što ćete ih prethodno prvo svesti na MDNF odnosno MKNF oblik.

a) $Y = [\overline{B} \vee (A \oplus C)][D \vee \overline{A \oplus C}]$

b) $Y = AB \vee A\overline{B}(C \oplus D) \vee \overline{A}B\overline{C \oplus D}$

10.26 Dokažite egzaktno Teoremu 10.1.

10.27 Precizno formulirajte, a zatim dokažite teoremu koja poopćava Teoremu 10.1 za slučaj kada se neka od elementarnih konjukcija odnosno disjunkcija svodi samo na literal (tj. na promjenljivu ili njenu negaciju).

11. Formalno projektiranje kombinacionih sklopova

Kombinacioni sklopovi (ili ***kombinacione mreže***) su digitalni sklopovi sa određenim brojem ulaza (recimo N) i određenim brojem izlaza (recimo M) kod kojih svi ulazi i izlazi mogu imati isključivo vrijednosti 0 i 1 i kod kojih vrijednosti izlaza u nekom trenutku zavise isključivo od vrijednosti ulaza u tom istom trenutku.

Ukoliko ulaze kombinacionog sklopa označimo sa x_i ($i = 1 .. N$) a izlaze sa y_j ($j = 1 .. M$), tada je iz definicije očigledno da je funkcioniranje ma kakvog kombinacionog sklopa u potpunosti definirano skupom od *j* jednačina

$$y_j = F_j(x_1, x_2, \ldots x_N), \quad j = 1 .. M$$

pri čemu su F_j neke logičke funkcije. Ako treba projektirati neki kombinacioni sklop, pri malom broju ulaza (recimo, do 6) najbolje je primijeniti ***formalni metod projektiranja***. Prema ovom metodu, prvo na osnovu tražene funkcionalnosti sklopa (koja je obično opisana tekstom) ispišemo kako bi trebala izgledati tablica istine koja opisuje rad tog sklopa, a zatim odredimo što jednostavnije funkcije koje realiziraju upravo takvu tablicu istine (recimo, pomoću Veitchovih dijagrama). Na osnovu poznatih funkcija sasvim je lako realizirati traženi sklop.

Kao što vidimo, suština formalne metode je posve jednostavna. U praksi je obično najveći problem upravo prvi korak, tj. prevođenje tekstualnog opisa rada sklopa u formaliziranu tablicu istine koja opisuje njegov rad, pogotovo ukoliko tekstualni opis rada sklopa nije dovoljno jasan i precizan. Vještina provođenja ovog koraka stiče se *samo iskustvom*. U ovom poglavlju će formalna metoda projektiranja kombinacionih sklopova biti ilustrirana na nekoliko karakterističnih primjera. U slučaju kombinacionih sklopova sa većim brojem ulaza, formalni metod projektiranja *nije pogodan*, te se koriste drugi metodi koje ćemo kasnije demonstrirati. Napomenimo da u svakom računaru postoji mnogo sastavnih dijelova koji su po građi upravo kombinacioni sklopovi. Na prvom mjestu, to su svi *sklopovi koji obavljaju neke računske operacije*, uključujući i kompletnu *aritmetičko-logičku jedinicu procesora* (dio procesora zadužen za računanje). Drugi karakterističan primjer kombinacionog sklopa koji se javlja u računarskim sistemima je *ROM memorija*. O građi ovih kombinacionih sklopova detaljno ćemo govoriti kasnije.

Primjer 11.1.

- Projektirati kombinacioni sklop sa tri ulaza i tri izlaza. Na ulaz se dovodi broj X zapisan u binarnom brojnom sistemu sa 3 bita, a na izlazu se kao rezultat dobija broj Y, također zapisan kao trobitni binarni broj, koji je povezan sa brojem X formulom $Y = (3 \cdot X + 5) \bmod 8$ gdje riječ "mod" označava "ostatak pri dijeljenju sa".

Projektiranje započinjemo prikazom tablice koja opisuje traženo funkcioniranje sklopa u dekadnom brojnom sistemu:

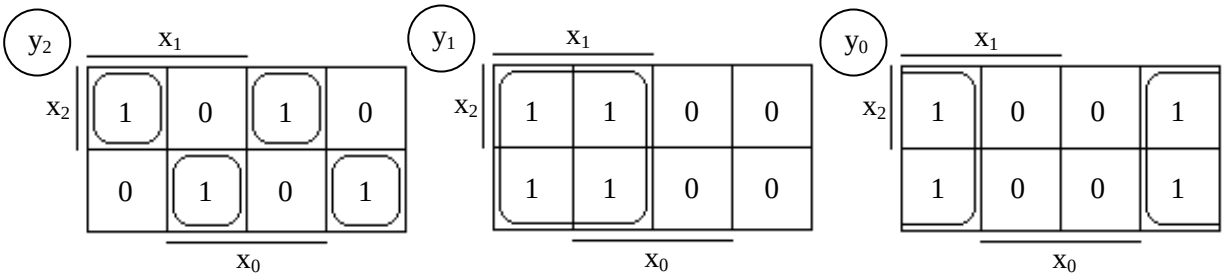
X	0	1	2	3	4	5	6	7
$3 \cdot X + 5$	5	8	11	14	17	20	23	26
$Y = (3 \cdot X + 5) \bmod 8$	5	0	3	6	1	4	7	2

Ukoliko brojeve X i Y izrazimo u binarnom sistemu kao $X = (x_2;x_1;x_0)_2$ i $Y = (y_2;y_1;y_0)_2$ gdje su x_i i y_i odgovarajuće binarne cifre brojeva X i Y (oznakom poput $(x_2;x_1;x_0)_2$ označavaćemo binarni broj koji je

sastavljen od cifara x_2, x_1 i x_0 u tom poretku, odnosno $(x_2;x_1;x_0)_2 = x_2 \cdot 2^2 + x_1 \cdot 2^1 + x_0 \cdot 2^0$; znak ";" uveli smo da izbjegnemo brkanje sa produktom), na osnovu gornje tablice može se formirati sljedeća tablica:

x_2	x_1	x_0	y_2	y_1	y_0
0	0	0	1	0	1
0	0	1	0	0	0
0	1	0	0	1	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	1	1	1
1	1	1	0	1	0

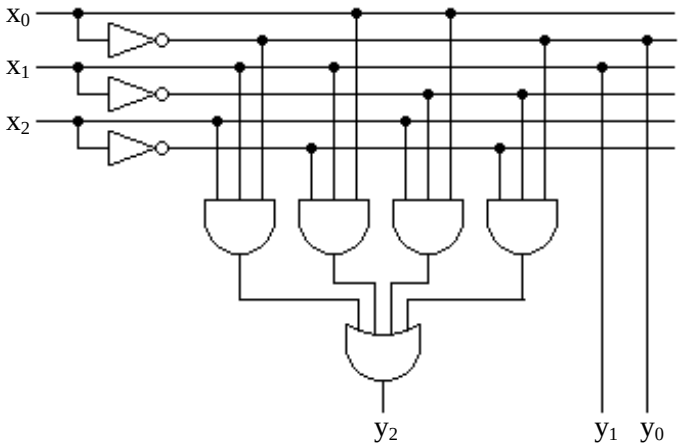
Ovo nije ništa drugo, nego tablica istine koja opisuje rad traženog sklopa. Sada je potrebno y_2, y_1 i y_0 izraziti kao funkcije od x_2, x_1 i x_0, što možemo učiniti pomoću sljedećih Veitchovih dijagrama:



Iz ovih dijagrama lako se očitavaju izrazi za y_2, y_1 i y_0:

$$y_2 = x_2 x_1 \overline{x}_0 \vee \overline{x}_2 x_1 x_0 \vee x_2 \overline{x}_1 x_0 \vee \overline{x}_2 \overline{x}_1 \overline{x}_0$$
$$y_1 = x_1$$
$$y_0 = \overline{x}_0$$

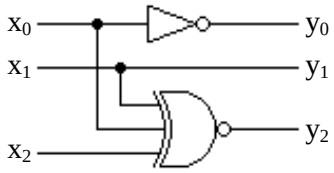
Preostalo je još samo da se nacrta šema sklopa koji realizira ove funkcije. Da bi se ostvarila veća preglednost crteža, obično se svi ulazi u sklop zajedno sa negiranim ulazima koji odgovaraju onim promjenljivim čije se negacije javljaju u traženim funkcijama vode *horizontalnim linijama*, dok se izlazi realiziraju *vertikalnim linijama* preko odgovarajućih logičkih kola. Doduše, na ovaj način krajevi horizontalnih linija na shemi ostaju neprikljućeni (tj. "vise u zraku"), ali se na taj način dobija mnogo preglednija shema. Na sljedećoj slici prikazana je šema traženog sklopa realizirana opisanim postupkom:



Može se primijetiti da je funkcija koja realizira izlaz y_2 prilično složena (ona je zapravo savršena disjunktivna normalna forma), što je posljedica činjenice da se ni jedna jedinica u dijagramu nije mogla grupirati niti sa jednom drugom jedinicom (isto vrijedi i za nule). Međutim, kada se god u nekom dijelu Veitchovog dijagrama (ili u čitavom dijagramu) pojavi karakterističan raspored nula i jedinica koji *podsjeća na šahovska polja*, to ukazuje na činjenicu da se odgovarajući članovi mogu ručno pojednostaviti *svođenjem na operaciju ekskluzivne disjunkcije*. U razmatranom primjeru imamo:

$$y_2 = x_2 x_1 \overline{x}_0 \vee \overline{x}_2 x_1 x_0 \vee x_2 \overline{x}_1 x_0 \vee \overline{x}_2 \overline{x}_1 \overline{x}_0 = x_2 (x_1 \overline{x}_0 \vee \overline{x}_1 x_0) \vee \overline{x}_2 (x_1 x_0 \vee \overline{x}_1 \overline{x}_0) =$$
$$= x_2 (x_1 \oplus x_0) \vee \overline{x}_2 \overline{x_1 \oplus x_0} = \overline{x_2 \oplus (x_1 \oplus x_0)} = x_2 \oplus x_1 \oplus x_0$$

Stoga, ukoliko dozvolimo da se u sklopu pojave i EXNOR kola, šema će biti mnogo jednostavnija, kao što je prikazano na sljedećoj slici:



Primjer 11.2:

- Projektirati kombinacioni sklop koji računa vrijednost funkcije $Y = X^2 - 25$, gdje je X trobitni broj na ulazu u sklop. U slučaju da je rezultat negativan broj, bite rezultata Y treba predstaviti u *2KK* kôdu.

Očigledno, i u ovom slučaju traženi sklop ima tri ulaza. Da bismo odredili neophodni broj izlaza, sastavimo prvo tablicu koja opisuje rad sklopa u dekadnom brojnom sistemu:

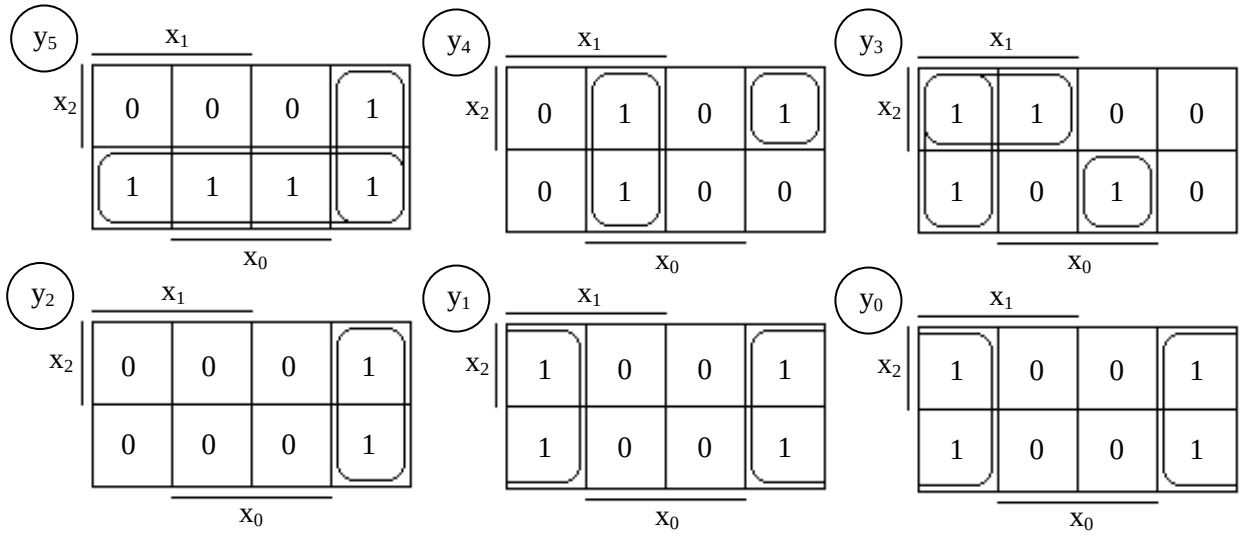
X	0	1	2	3	4	5	6	7
$X^2 - 25$	–25	–24	–21	–16	–9	0	11	24

Najveći broj koji se može pojaviti na izlazu je 24, koji se može zapisati u 5 bita (11000 binarno). Međutim, kako izlaz može biti i negativan, moramo predvidjeti i jedan bit za signalizaciju znaka, tako da nam za pozitivne rezultate treba ukupno 6 izlaza. Također, najmanji rezultat koji se može pojaviti je –25, koji se u *2KK* može također predstaviti sa 6 bita, jer je $25 = (11001)_2 = (011001)_2$, pa je zapis broja –25 u *2KK* kôdu 100110 + 1 = 100111 (nulu na početku smo dodali radi pamćenja znaka). Slijedi da je 6 bita za pamćenje broja Y potrebno i dovoljno za sve slučajeve, tako da traženi sklop ima 6 izlaza.

Pretvorimo li redom brojeve –25, –24, –21, –16, –9, 0, 11 i 24 u 6-bitne binarne brojeve, dobijamo redom brojeve 100111, 101000, 101011, 110000, 110111, 000000, 001011 i 011000 (negativni brojevi su u 6-bitnom *2KK*). Nakon uvođenja oznaka $X = (x_2;x_1;x_0)_2$ i $Y = (y_5;y_4;y_3;y_2;y_1;y_0)_2$, možemo sastaviti sljedeću tablicu istine:

x_2	x_1	x_0	y_5	y_4	y_3	y_2	y_1	y_0
0	0	0	1	0	0	1	1	1
0	0	1	1	0	1	0	0	0
0	1	0	1	0	1	0	1	1
0	1	1	1	1	0	0	0	0
1	0	0	1	1	0	1	1	1
1	0	1	0	0	0	0	0	0
1	1	0	0	0	1	0	1	1
1	1	1	0	1	1	0	0	0

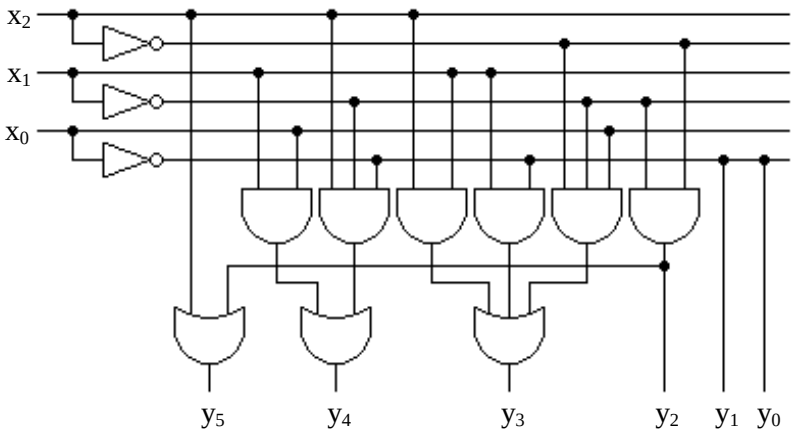
Na osnovu ove tablice, možemo pomoću Veitchovih dijagrama odrediti zavisnosti izlaza y_5, y_4, y_3, y_2, y_1 i y_0 od ulaza x_2, x_1 i x_0:



Iz ovih dijagrama lako se očitavaju tražene funkcije:

$$y_5 = \overline{x}_2 \vee \overline{x}_1\overline{x}_0 \qquad y_4 = x_1x_0 \vee x_2\overline{x}_1\overline{x}_0 \qquad y_3 = x_2x_1 \vee x_1\overline{x}_0 \vee \overline{x}_2\overline{x}_1x_0$$
$$y_2 = \overline{x}_1\overline{x}_0 \qquad y_1 = \overline{x}_0 \qquad y_0 = \overline{x}_0$$

Na osnovu ovih relacija, sasvim lako je nacrtati traženi sklop:



Prilikom crtanja sheme smo iskoristili činjenicu da funkcije y_5 i y_2 imaju jedan zajednički član (zapravo, čitava funkcija y_2 se javlja kao član u funkciji y_5), što nam je omogućilo da uštedimo jedno AND kolo (pri tome mreža ostaje u dva nivoa, tako da ništa nismo izgubili ni na brzini).

Posve je jasno da je na način sličan kao u Primjeru 11.1 i Primjeru 11.2 moguće napraviti sklop koji realizira bilo kakvu funkciju Y = F(X) gdje su X i Y binarno kodirani brojevi, pod uvjetom da ulaz X nema mnogo bita tako da je moguće koristiti formalne metode minimizacije. Primijetimo da broj bita izlaza Y ne unosi bitne komplikacije, mada nesumnjivo produžava trajanje postupka, jer treba minimizirati veći broj promjenljivih. Broj bita ulaza je mnogo kritičniji.

Primjer 11.3:

- Napraviti kombinacioni sklop koji je u stanju da sabira dva broja od po dva bita i da daje na izlazu rezultat sabiranja.

Traženi sklop očito ima četiri ulaza. Prikažimo sada tablicu sabiranja brojeva koji se mogu prikazati sa dva bita, u dekadnom i binarnom brojnom sistemu:

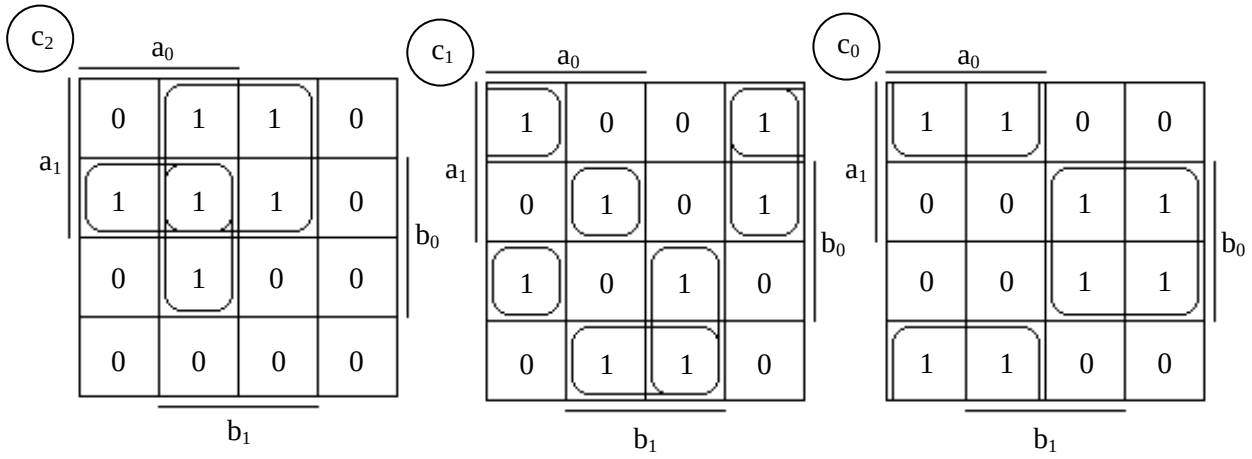
A \ B	0	1	2	3
0	0	1	2	3
1	1	2	3	4
2	2	3	4	5
3	3	4	5	6

A \ B	00	01	10	11
00	000	001	010	011
01	001	010	011	100
10	010	011	100	101
11	011	100	101	110

Odavde vidimo da su nam potrebna tri izlaza, jer rezultat sabiranja dva broja od po dva bita može imati najviše tri bita (općenito, rezultat sabiranja jednog broja od m bita i drugog od n bita može imati najviše $\max(m, n)$ bita za $m \neq n$, odnosno $m+1$ bita za $m = n$, što je lako pokazati). Ukoliko predstavimo brojeve A i B preko svojih bita kao $A = (a_1;a_0)_2$ i $B = (b_1;b_0)_2$, a rezultat sabiranja $C = A + B$ kao $C = (c_2;c_1;c_0)_2$, možemo formirati sljedeću tablicu istine:

a_1	a_0	b_1	b_0	c_2	c_1	c_0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

Izlaze c_2, c_1 i c_0 treba izraziti kao funkcije od a_1, a_0, b_1 i b_0, što ćemo izvesti preko Veitchovih dijagrama, kao na sljedećoj slici:



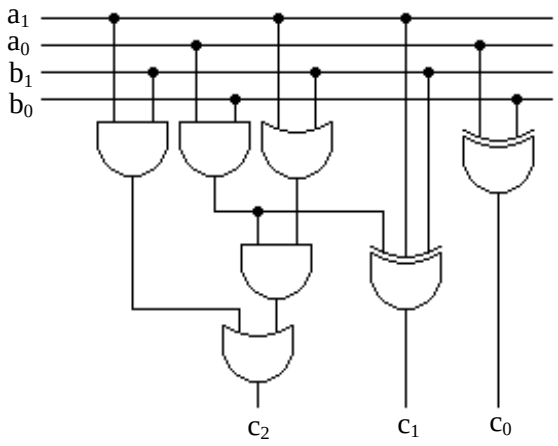
Iz ovih dijagrama očitavamo tražene funkcije:

$$c_2 = a_1 b_1 \vee a_0 a_1 b_0 \vee a_0 b_0 b_1 = a_1 b_1 \vee a_0 b_0 (a_1 \vee b_1)$$
$$c_1 = a_1 \bar{b}_0 \bar{b}_1 \vee \bar{a}_0 \bar{a}_1 b_1 \vee \bar{a}_0 a_1 \bar{b}_1 \vee \bar{a}_1 \bar{b}_0 b_1 \vee a_0 \bar{a}_1 b_0 \bar{b}_1 \vee a_0 a_1 b_0 b_1$$
$$c_0 = a_0 \bar{b}_0 \vee \bar{a}_0 b_0 = a_0 \oplus b_0$$

Funkcija kojom se realizira izlaz c_1 je prilično glomazna. Ako dopustimo upotrebu ekskluzivne disjunkcije, tada se ova funkcija može osjetno skratiti pomoću pravila logičke algebre na sljedeći način:

$$\begin{aligned} c_1 &= a_1 \bar{b}_0 \bar{b}_1 \vee \bar{a}_0 \bar{a}_1 b_1 \vee \bar{a}_0 a_1 \bar{b}_1 \vee \bar{a}_1 \bar{b}_0 b_1 \vee a_0 \bar{a}_1 b_0 \bar{b}_1 \vee a_0 a_1 b_0 b_1 = \\ &= (\bar{a}_0 \vee \bar{b}_0) a_1 \bar{b}_1 \vee (\bar{a}_0 \vee \bar{b}_0) \bar{a}_1 b_1 \vee a_0 b_0 (\bar{a}_1 \bar{b}_1 \vee a_1 b_1) = \\ &= (\bar{a}_0 \vee \bar{b}_0)(a_1 \bar{b}_1 \vee \bar{a}_1 b_1) \vee a_0 b_0 (\bar{a}_1 \bar{b}_1 \vee a_1 b_1) = (\bar{a}_0 \vee \bar{b}_0)(a_1 \oplus b_1) \vee a_0 b_0 \overline{a_1 \oplus b_1} = \\ &= \overline{a_0 b_0} (a_1 \oplus b_1) \vee a_0 b_0 \overline{a_1 \oplus b_1} = a_0 b_0 \oplus a_1 \oplus b_1 \end{aligned}$$

Na mogućnost upotrebe ekskluzivne disjunkcije za dodatnu optimizaciju funkcije c_1 ukazuju jedinice u lijevoj polovini Veitchovog dijagrama koje obrazuju šahovski raspored. Na ovaj način dobijamo sljedeću shemu traženog sklopa (uz korištenje EXOR kola):

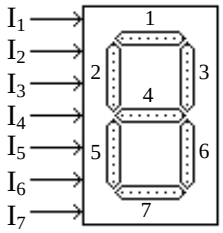


Ovdje smo iskoristili činjenicu da se član $a_0 b_0$ javlja na dva mjesta.

Na način koji je potpuno analognan Primjeru 11.3 mogu se projektirati i sklopovi koji obavljaju druge računske operacije nad operandima sa malim brojem bita, poput oduzimanja i množenja. Dalje je moguće napraviti *binarni komparator*, tj. sklop koji poredi dva binarna broja A i B dovedena na ulaze i signalizira kakav je odnos između njih (npr. preko tri izlaza $c_>$, $c_<$ i $c_=$ na kojima se javlja logička jedinica ako i samo ako je A>B, A<B odnosno A=B respektivno), itd.

Primjer 11.4:

- Za numeričku indikaciju rezultata u jednostavnijim digitalnim sistemima koriste se ***sedmosegmeni indikatori*** (***displeji***), koji su sastavljeni od sedam segmenata kojima se može individualno upravljati pomoću sedam ulaznih signala. Ovakvi indikatori su, na primjer, prisutni u digitalnim satovima i džepnim kalkulatorima. Svaki segment može biti u jednom od dva vidljivo različita stanja (npr. može svijetliti ili biti ugašen, može biti u jednoj od dvije moguće boje, itd.) u zavisnosti od logičkog stanja (0/1) na odgovarajućem ulazu. Na sljedećoj slici prikazan je jedan takav indikator, u kojem su segmenti numerirani brojevima od 1 do 7, a odgovarajući upravljački ulazi označeni sa I_1 do I_7.



Potrebno je projektirati kombinacioni sklop koji na ulazu prihvata cifru u opsegu $0 \div 9$ (binarno kodiranu), a koji na izlazu generira upravljačke signale za sedmosegmentni indikator koji su potrebni da se na indikatoru iscrta lik odgovarajuće cifre.

Za tu svrhu, na prvom mjestu trebamo razmotriti kako bi se mogle prikazati cifre iz opsega $0 \div 9$ na indikatoru. Najlogičnija varijanta prikazana je na sljedećoj slici:

Odavde možemo formirati tablicu koja prikazuje koje segmente treba aktivirati za prikaz pojedinih cifara:

CIFRA	AKTIVNI SEGMENTI
0	1, 2, 3, 5, 6, 7
1	3, 6
2	1, 3, 4, 5, 7
3	1, 3, 4, 6, 7
4	2, 3, 4, 6
5	1, 2, 4, 6, 7
6	1, 2, 4, 5, 6, 7
7	1, 3, 6
8	1, 2, 3, 4, 5, 6, 7
9	1, 2, 3, 4, 6, 7

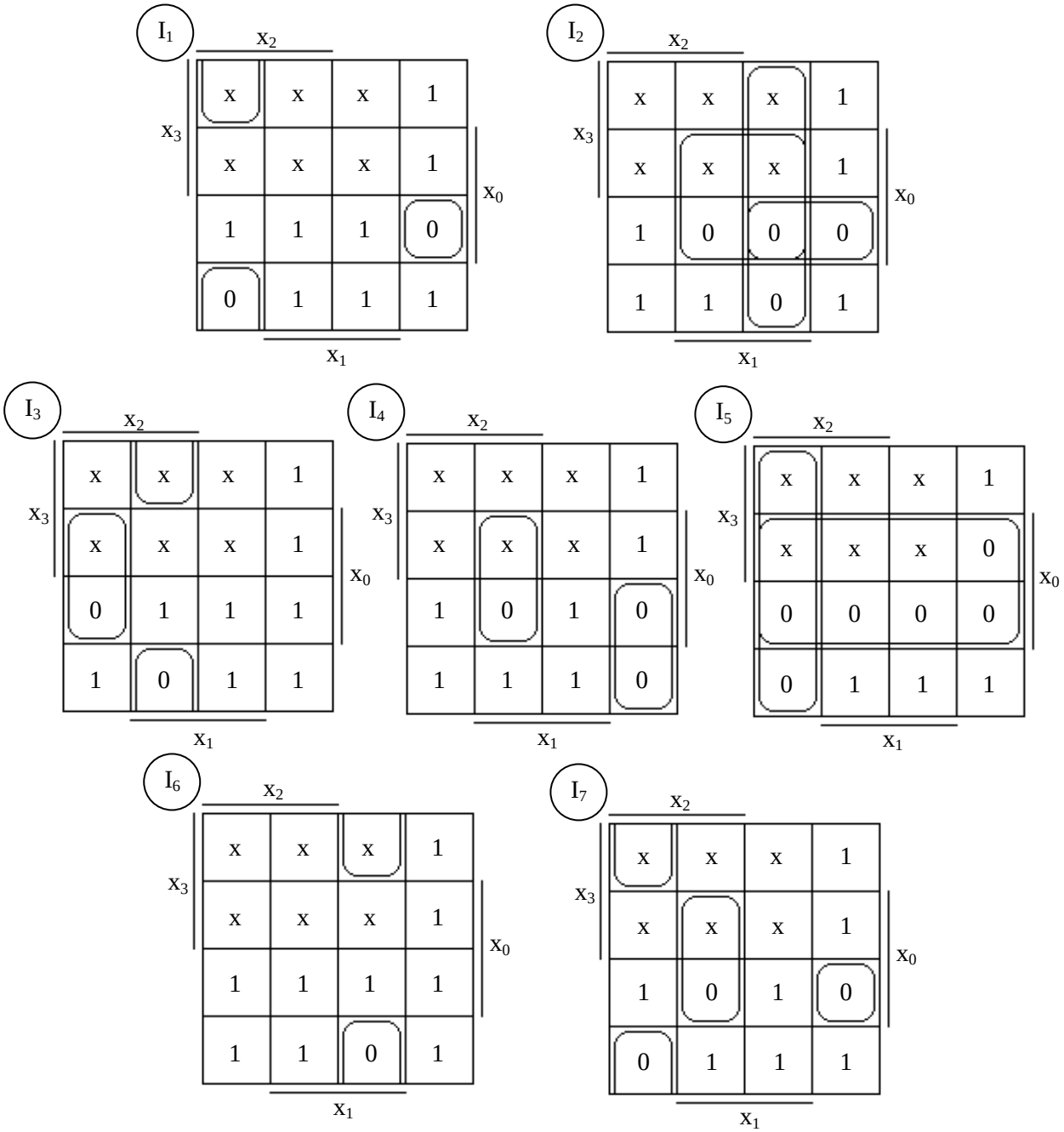
Ukoliko označimo cifru koju dovodimo na ulaz traženog kombinacionog sklopa sa X, a bite njenog binarnog zapisa redom sa x_3, x_2, x_1 i x_0 (tj. $X = (x_3;x_2;x_1;x_0)_2$), lako možemo sastaviti tablicu koja daje zavisnost pobudnih signala I_1, I_2, I_3, I_4, I_5, I_6 i I_7 od x_3, x_2, x_1 i x_0:

x_3	x_2	x_1	x_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
0	0	0	0	1	1	1	0	1	1	1
0	0	0	1	0	0	1	0	0	1	0
0	0	1	0	1	0	1	1	1	0	1
0	0	1	1	1	0	1	1	0	1	1
0	1	0	0	0	1	1	1	0	1	0
0	1	0	1	1	1	0	1	0	1	1
0	1	1	0	1	1	0	1	1	1	1
0	1	1	1	1	0	1	0	0	1	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

Sada je potrebno odrediti funkcije koje predstavljaju ovisnost ulaza od izlaza, što ćemo uraditi pomoću Veitchovih dijagrama. Uočimo da je gornja tablica nepotpuna, jer u njoj nedostaju redovi koji odgovaraju kombinacijama ulaza 1010, 1011, 1100, 1101, 1110 i 1111 koje se ne mogu pojaviti, s obzirom da se na ulaz dovodi cifra u opsegu od 0 do 9. Za logičke funkcije čije tablice istine nisu kompletne s obzirom da znamo da se neke kombinacije ulaznih promjenljivih neće (ili ne mogu) nikada pojaviti kažemo da su ***nepotpuno definirane***.

Nepotpuno definirane logičke funkcije pružaju mogućnost za bolju optimizaciju uz pomoć Veitchovih dijagrama nego potpuno definirane logičke funkcije. Prilikom popunjavanja Veitchovih dijagrama za nepotpuno definirane logičke funkcije, u polja koja odgovaraju nedostajućim redovima tablice istine upisujemo znak "x". Svako takvo polje prilikom optimizacije možemo smatrati bilo kao jedinicu, bilo kao nulu, zavisno kako nam više odgovara. Drugim riječima, takva polja *ne moraju biti prekrivena*, ali neka od njih *možemo prekriti* ukoliko ćemo na taj način dobiti bolje prekrivanje (tj. veće konture, koje će odgovarati kraćim implikantama).

Kako sve funkcije koje trebamo da odredimo imaju pretežno jedinice za vrijednosti, dok su nule znatno rjeđe, praktičnije je tražiti prekrivanja nula u Veitchovim dijagramima. Optimalna prekrivanja nula prikazana su na sljedećoj slici:



Iz ovih dijagrama lako očitavamo minimalne konjunktivne normalne formule za funkcije kojima se realiziraju pobude I_1 do I_7:

$$I_1 = (x_3 \vee x_2 \vee x_1 \vee \overline{x}_0)(\overline{x}_2 \vee x_1 \vee x_0)$$

$$I_2 = (\overline{x}_1 \vee \overline{x}_0)(x_2 \vee \overline{x}_1)(x_3 \vee x_2 \vee \overline{x}_0)$$

$$I_3 = (\overline{x}_2 \vee x_1 \vee \overline{x}_0)(\overline{x}_2 \vee \overline{x}_1 \vee x_0)$$

$$I_4 = (\overline{x}_2 \vee \overline{x}_1 \vee \overline{x}_0)(x_3 \vee x_2 \vee x_1)$$

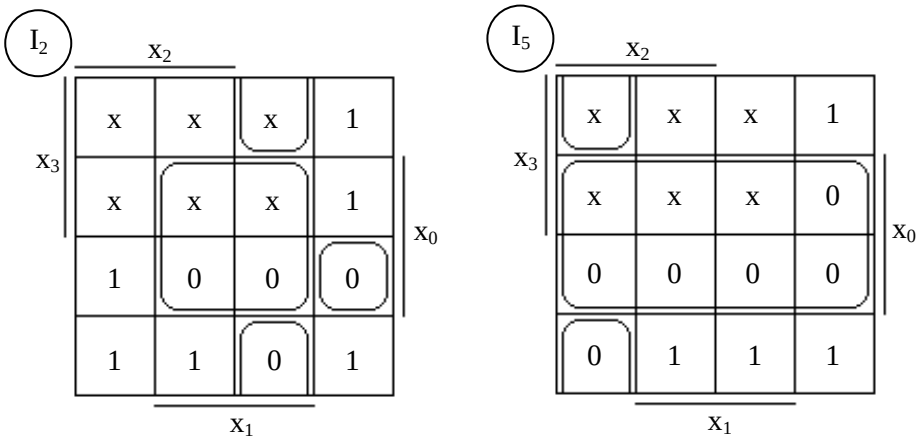
$$I_5 = \overline{x}_0(\overline{x}_2 \vee x_1)$$

$$I_6 = x_2 \vee \overline{x}_1 \vee x_0$$

$$I_7 = (\overline{x}_2 \vee x_1 \vee x_0)(x_3 \vee x_2 \vee x_1 \vee \overline{x}_0)(\overline{x}_2 \vee \overline{x}_1 \vee \overline{x}_0)$$

Ukoliko bismo se odlučili da neovisno ove funkcije realiziramo neovisno jednu od druge, tada bismo pored tri zajednička invertora (promjenljiva x_3 se nigdje ne javlja negirana) trebali upotrijebiti 6 AND kola i 14 OR kola, sa ukupno 55 ulaza. Međutim, može se primijetiti da u ovim funkcijama postoje *zajednički faktori*. Tako, na primjer, oba faktora koji čine funkciju I_1 javljaju se i u funkciji I_7 (odnosno, cijela funkcija I_1 je faktor u funkciji I_7). Preostali faktor u funkciji I_7 javlja se i u funkciji I_4. Ovi zajednički faktori mogu se uočiti i na Veitchovim dijagramima, po prisustvu *zajedničkih kontura*. Tako, sve konture koje su prisutne u dijagramu za funkciju I_1 postoje i u dijagramu za funkciju I_7, dok dijagrami za funkcije I_4 i I_7 imaju zajedničku konturu. Ukoliko uočimo ovu činjenicu, uštedjećemo 3 OR kola (dva troulazna i jedno četveroulazno), a posljednje troulazno AND kolo postaje dvoulazno. Ovim smo ukupan broj ulaza sveli sa 55 na 44.

Veitchov metod nam pruža mogućnost za dalje uštede. Naime, ukoliko prilikom realizacije sistema (skupa) funkcija ne optimiziramo svaku funkciju za sebe nego se trudimo da formiramo *konture koje se javljaju u više različitih dijagrama*, postići ćemo da se pojavi *veći broj zajedničkih članova* (odnosno *faktora*) u funkcijama, tako da se krajnji sklop može realizirati sa *manjim brojem elemenata*. Na primjer, dijagrame za funkcije I_2 i I_5 bolje je formirati kao na sljedećoj slici:



Ovi dijagrami sami za sebe ne formiraju optimalna prekrivanja nula i njima odgovaraju sljedeće funkcije, koje posmatrane same za sebe *nisu optimalne*:

$$I_2 = (\overline{x}_1 \vee \overline{x}_0)(x_2 \vee \overline{x}_1 \vee x_0)(x_3 \vee x_2 \vee x_1 \vee \overline{x}_0)$$

$$I_5 = \overline{x}_0(\overline{x}_2 \vee x_1 \vee x_0)$$

Međutim, na ovaj način dijagram za funkciju I_2 sadrži jednu konturu koja je zajednička sa dijagramom za funkciju I_6 i konturu koja je zajednička u dijagramima za funkcije I_1 i I_7, dok dijagram za funkciju I_5 takođe sadrži konturu koja se javlja u dijagramima za funkcije I_1 i I_7. Kao posljedicu ovoga, imamo pojavu novih zajedničkih faktora koji omogućavaju uštedu još 3 OR kola, i svođenje broja ulaza

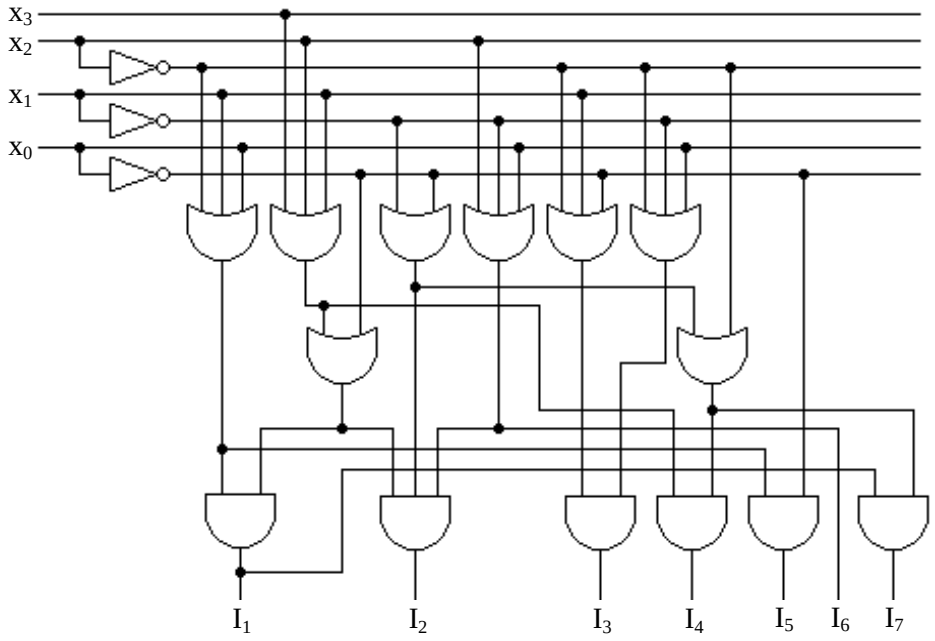
na 37. Drugim riječima, bolje je prvo formirati konture koje se mogu iskoristiti u *više različitih dijagrama* čak i ukoliko one nisu optimalne kada se dijagrami posmatraju neovisno jedan od drugog, a tek nakon toga prekriti preostale nepokrivene elemente (jedinice ili nule, ovisno da li tražimo DNF ili KNF) na optimalan način.

Ovim još nismo iscrpili sve mogućnosti za optimizaciju koje pruža Veitchov metod, ako dozvolimo da koristimo i *višenivoske mreže*. Primijetimo da je mala kontura koja pokriva samo jedno polje u dijagramima za funkcije I_1, I_2 (novom) i I_7 u potpunosti sadržana u paru koji se nalazu u dijagramu za funkciju I_4, dok je zajednički par u dijagramima za funkcije I_4 i I_7 u potpunosti sadržan u četvorki koja se javlja u dijagramu za funkciju I_2. Očigledno, kada se god u Veitchovom dijagramu za neku funkciju pojavi kontura koja je u potpunosti sadržana u nekoj konturi u Veitchovom dijagramu za neku drugu funkciju, to je znak da elementarna konjunkcija ili disjunkcija koja odgovara većoj konturi predstavlja *dio elementarne konjunkcije ili disjunkcije* koja odgovara manjoj konturi. I zaista, vidimo da faktor $x_3 \vee x_2 \vee x_1$ koji se javlja u funkciji I_4 predstavlja dio faktora $x_3 \vee x_2 \vee x_1 \vee \overline{x}_0$ koji se javlja u funkcijama I_1, I_2 i I_7, dok faktor $\overline{x}_1 \vee \overline{x}_0$ koji se javlja u funkciji I_2 predstavlja dio faktora $\overline{x}_2 \vee \overline{x}_1 \vee \overline{x}_0$ koji se javlja u funkcijama I_4 i I_7. Ova činjenica može se iskoristiti da uštedimo još tri ulaza, čime smo broj ulaza sveli na 34, što je značajna ušteda u odnosu na 55, koliko bismo imali u slučaju da smo sve funkcije realizirali neovisno jednu od druge.

Ovim smo iscrpili sve mogućnosti optimizacije ukoliko se ograničimo samo na konjuktivne normalne forme funkcija. To još uvijek ne znači da eventualno nije moguće ručno izvršiti i neke dodatne transformacije. Tako, na primjer, ukoliko dozvolimo i upotrebu EXNOR kola, funkcija I_3 može se napisati u obliku

$$I_3 = \overline{x}_2 \vee \overline{x_1 \oplus x_0}$$

čime možemo prištedjeti još četiri ulaza (mada na taj način koristimo i znatno složenije EXNOR kolo). Na sljedećoj slici je prikazana konačna šema sklopa bez izvršene posljednje transformacije, odnosno samo uz upotrebu standardnih logičkih kola (AND, OR i NOT):



Upravo projektirani kombinacioni sklop možemo shvatiti i kao sklop koji pretvara jedan zapis cifara od 0 do 9 (4-bitni binarni zapis) u drugi, 7-bitni zapis (prilagođen sedmosegmentnom indikatoru), odnosno kao sklop za konverziju zapisa podataka (kôdova) iz jedne forme u drugu. Ovakvi kombinacioni sklopovi nazivaju se ***konverzione matrice***.

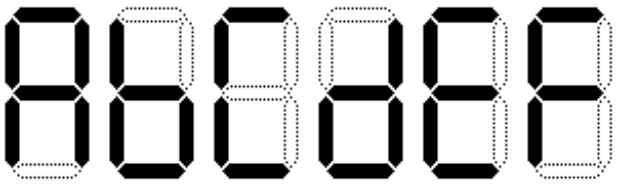
Mada smo prilikom projektiranja prethodnog sklopa pretpostavili da se zabranjene kombinacije (1010, 1011, 1100, 1101, 1110 ili 1111) neće nikada pojaviti na ulazu sklopa, jasno je da će se, ukoliko se one ipak pojave, nešto morati pojaviti na izlazu sklopa. Šta će se tačno pojaviti, možemo utvrditi tek kada završimo sa projektiranjem. Naime, nakon što smo odredili analitičke izraze za funkcije koje daju zavisnosti izlaza od ulaza, moguće je lako izračunati kakve će biti vrijednosti izlaza za zabranjene kombinacije ulaza. Za konkretan slučaj, na osnovu analitičkih izraza za funkcije I_1, I_2, I_3, I_4, I_5, I_6 i I_7 koje smo izveli nije teško odrediti njihove vrijednosti za zabranjene kombinacije ulaza. Ove vrijednosti su prikazane u sljedećoj tablici:

x_3	x_2	x_1	x_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
1	0	1	0	1	0	1	1	1	0	1
1	0	1	1	1	0	1	1	0	1	1
1	1	0	0	0	1	1	1	0	1	0
1	1	0	1	1	1	0	1	0	1	1
1	1	1	0	1	1	0	1	1	1	1
1	1	1	1	1	0	1	0	0	1	0

Ovu tablicu smo mogli formirati još brže, bez uvrštavanja u analitičke izraze, tako što u odgovarajućim Veitchovim dijagramima iz kojih smo odredili ove analitičke izraze pogledamo koji smo znak "x" tretirali kao nulu, a koji kao jedinicu. Na taj način, možemo sve znakove "x" zamijeniti konkretnim vrijednostima 0 ili 1, nakon čega se odmah mogu očitati vrijednosti koje odgovaraju redovima koji nedostaju u izvornoj tabeli.

Ako pažljivije pogledamo, vidjećemo da su izlazi koji odgovaraju zabranjenim kombinacijama ulaza 1010, 1011, 1100, 1101, 1110 ili 1111 identični kao izlazi koji odgovaraju legalnim kombinacijama ulaza 0010, 0011, 0100, 0101, 0110 ili 0111. Drugim riječima, ukoliko na ulaze dovedemo bite koji predstavljaju binarne zapise brojeva 10, 11, 12, 13, 14 ili 15, na displeju priključenom na izlaz sklopa ispisaće se 2, 3, 4, 5, 6 ili 7! Ovo nije plod puke slučajnosti. Naime, primijetimo da se šest zabranjenih kombinacija i njihovih šest dozvoljenih "dvojnika" razlikuju samo u vrijednosti bita x_3. Ukoliko znamo da se zabranjene kombinacije nikada neće pojaviti, vrijednost ovog bita za ulazne kombinacije 0010, 0011, 0100, 0101, 0110 ili 0111 možemo prosto da *ignoriramo*, što smo upravo i uradili tretirajući oznake "x" u Veitchovom dijagramu kako nam kad odgovara. Stoga, ne treba da čudi što sklop daje iste izlaze npr. za ulaze 1010 i 0010, jer on ta dva ulaza zapravo i ne razlikuje, s obzirom da se bit x_3 ignorira za ove kombinacije ulaza.

Na kraju, napomenimo da se konverziona matrica za sedmosegmentni indikator može projektirati i tako da kombinacije ulaza 1010, 1011, 1100, 1101, 1110 ili 1111 ne budu zabranjene, nego da za njih sklop na izlazu daje upravljačke signale za sedmosegmentni indikator koji će dovesti do prikaza odgovarajućih *heksadekadnih cifara* A, B, C, D, E i F, koje je također moguće prikazati na sedmosegmentnom displeju na sljedeći način:



Projektiranje ovakve konverzione matrice ostavlja se čitatelju ili čitateljki za vježbu.

(?) Pitanja i zadaci

11.1 U ovom poglavlju smo vidjeli da se logičke funkcije često mogu bolje optimizirati ukoliko se neke od kombinacija ulaznih promjenljivih nikada neće pojaviti na ulazu (tj. ukoliko su tražene funkcije nepotpuno definirane). Nađite minimalne disjunktivne forme za sljedeće nepotpuno definirane logičke funkcije, uz pretpostavku da minterme ovise od promjenljivih A, B, C i D:

a) $Y = m_3 \vee m_4 \vee m_6 \vee m_9 \vee m_{10} \vee m_{15}$, *nedefinirana za* $m_0 \vee m_8 \vee m_{11} = 1$

b) $Y = m_0 \vee m_2 \vee m_8 \vee m_9$, *nedefinirana za* $m_1 \vee m_3 \vee m_5 \vee m_7 \vee m_{15} = 1$

c) $Y = m_1 \vee m_7 \vee m_{11} \vee m_{13}$, *nedefinirana za* $m_0 \vee m_2 \vee m_4 \vee m_5 \vee m_{10} \vee m_{15} = 1$

d) $Y = m_1 \vee m_5 \vee m_7 \vee m_8 \vee m_9 \vee m_{13} \vee m_{15}$, *nedefinirana za* $m_4 \vee m_{14} = 1$

11.2 Za logičku funkciju $Y = m_1 \vee m_5 \vee m_6 \vee m_8 \vee m_9 \vee m_{12}$, nađite MDNF i MKNF oblike ukoliko je poznato da se nikada neće pojaviti takve vrijednosti ulaznih promjenljivih za koje vrijedi $m_0 \vee m_2 \vee m_{10} \vee m_{11} \vee m_{14} \vee m_{15} = 1$. Sve minterme ovise od promjenljivih A, B, C i D

11.3 Data je logička funkcija $Y = (\overline{A} \vee C \vee D)(A \vee \overline{C} \vee D)(\overline{A} \vee B)$, data u MKNF obliku.

a) Kako glasi MDNF oblik date funkcije?

b) Kako se može optimizirati MKNF oblik date funkcije, ukoliko je poznato da se nikada neće pojaviti takve kombinacije ulaznih promjenljivih za koje vrijedi $M_0 M_2 M_4 M_{15} = 0$ (pri tome se podrazumijeva da maksterme također zavise od promjenljivih A, B, C i D)?

c) Kako se može optimizirati MDNF oblik date funkcije, uz iste pretpostavke kao pod b)?

11.4* Nađite minimalne disjunktivne forme za sljedeće nepotpuno definirane logičke funkcije, uz pretpostavku da sve minterme ovise od promjenljivih A, B, C, D i E:

a) $Y = m_2 \vee m_8 \vee m_9 \vee m_{10} \vee m_{13} \vee m_{15} \vee m_{16} \vee m_{18} \vee m_{19} \vee m_{23}$,
nedefinirana za $m_3 \vee m_{11} \vee m_{17} \vee m_{22} \vee m_{31} = 1$

b) $Y = m_0 \vee m_1 \vee m_2 \vee m_9 \vee m_{13} \vee m_{16} \vee m_{18} \vee m_{24} \vee m_{25}$,
nedefinirana za $m_8 \vee m_{10} \vee m_{17} \vee m_{19} = 1$

11.5 Data je logička funkcija $Y = AB\overline{C}DE \vee \overline{A}\,\overline{B}\,\overline{D}\,\overline{E} \vee \overline{A}BE \vee \overline{A}\,\overline{D}E \vee \overline{B}C\overline{E} \vee CDE$.

a) Nađite MDNF i MKNF oblik date funkcije.

b) Nađite MDNF i MKNF oblik date funkcije uz dodatnu pretpostavku da se nikada neće pojaviti takve vrijednosti promjenljivih za koje vrijedi $A\overline{B}\,\overline{C}\,\overline{D} \vee B\overline{C}D\overline{E} \vee A\overline{D}E = 1$.

11.6 Neka je dat skup logičkih funkcija

$X = m_1 \vee m_2 \vee m_3 \vee m_5 \vee m_7 \vee m_9 \vee m_{11} \vee m_{13} \vee m_{15}$

$Y = M_2 M_3 M_4 M_5 M_6 M_7 M_8 M_{10} M_{12} M_{14}$

$Z = m_0 \vee m_1 \vee m_2 \vee m_3 \vee m_5 \vee m_7$

pri čemu sve minterme zavise od promjenljivih A, B, C, D.

a) Nađite MDNF oblike za funkcije X, Y i Z, posmatrajući ih *neovisno jednu od druge*.

b) Primjer 11.4 jasno ilustrira da se kod kombinacionih mreža sa više izlaza, često mogu postići ekonomičnije realizacije ukoliko se funkcije koje opisuju zavisnost izlaza od ulaza ne optimiziraju neovisno jedna od druge, nego kao *cjelina* (tj. kao ***sistem logičkih funkcija***). Nađite DNF oblike za funkcije X, Y i Z koji omogućavaju ekonomičnije realizacije u odnosu na MDNF oblike dobijene pod a).

Koliko je ulaza u logička kola potrebno u oba slučaja?

11.7 Dat je skup logičkih funkcija

$X = \overline{A}B\overline{C}D \vee BC \qquad Y = \overline{A}\,\overline{B}CD \vee \overline{A}BD \vee ACD \qquad Z = \overline{A}B\overline{C}D \vee ABC \vee CD$

Nađite MDNF i MKNF forme ovih funkcija posmatrajući ih zasebno, a zatim razmotrite mogu li se naći DNF odnosno KNF forme koje omogućavaju ekonomičniju realizaciju, posmatrajući ove funkcije kao sistem logičkih funkcija. Koliko je ulaza u logička kola potrebno za realizaciju u oba slučaja?

11.8 Dat je skup logičkih funkcija

$X = \overline{A}B\overline{C}D \vee A\overline{B}\,\overline{C}\,\overline{D} \vee AB\overline{C} \vee \overline{A}C\overline{D} \qquad Y = AC\overline{D} \vee B\overline{C}D \qquad Z = A\overline{D} \vee C\overline{D}$

pri čemu je poznato da se nikada neće pojaviti takve vrijednosti ulaznih promjenljivih za koje je CD = 1. Uz takvu pretpostavku, nađite MDNF i MKNF forme ovih funkcija posmatrajući ih zasebno, a zatim razmotrite mogu li se naći DNF odnosno KNF forme koje omogućavaju ekonomičniju realizaciju, posmatrajući ove funkcije kao sistem logičkih funkcija. Koliko je ulaza u logička kola potrebno za realizaciju u oba slučaja?

11.9 Dat je skup logičkih funkcija

$X = B\overline{C}(A \vee D) \vee \overline{A}\,\overline{B}D \qquad Y = \overline{A}\,\overline{B}C \vee A\overline{D}(\overline{B} \vee \overline{C}) \qquad Z = \overline{C}[AB \vee (A \oplus D)] \vee \overline{A}\,\overline{B}C$

pri čemu je poznato da se nikada neće pojaviti takve vrijednosti ulaznih promjenljivih za koje je BC = 1. Uz takvu pretpostavku, nađite MDNF i MKNF forme ovih funkcija posmatrajući ih zasebno, a zatim razmotrite mogu li se naći DNF odnosno KNF forme koje omogućavaju ekonomičniju realizaciju, posmatrajući ove funkcije kao sistem logičkih funkcija. Koliko je ulaza u logička kola potrebno za realizaciju u oba slučaja?

11.10 Projektirajte kombinacioni sklop koji realizira funkciju $Y = (X^2 + 1) \bmod 7$ gdje operacija "mod" označava "ostatak pri dijeljenju sa", pri čemu je ulazni podatak X 3-bitni broj. Realizacija treba da se zasniva na strukturno optimalnoj mreži u dva nivoa sa AND, OR i NOT logičkim kolima, dobijenoj iz disjunktivnih normalnih formi.

11.11 Projektirajte kombinacioni sklop koji realizira funkciju $Y = (3X^2 + 1) \bmod 13$, pri čemu je ulazni podatak X 3-bitni broj. Realizacija treba da se zasniva na strukturno optimalnoj mreži u dva nivoa sa AND, OR i NOT logičkim kolima, dobijenoj iz disjunktivnih normalnih formi.

11.12 Kombinacioni sklopovi koji realiziraju funkciju $Y = X + 1$ gdje su X i Y *n*-bitni binarni brojevi nazivaju se ***n-bitni binarni inkrementeri***. Projektirajte formalnim putem 4-bitni binarni inkrementer. Razmotrite moguće realizacije na osnovu struktirno optimalnih mreža u dva nivoa sa AND, OR i NOT logičkim kolima, dobijenih kako iz disjunktivnih, tako i konjunktivnih normalnih formi. Zatim ispitajte kako se realizacija može učiniti ekonomičnijom ukoliko se dozvoli i upotreba EXOR logičkih kola.

11.13 U digitalnoj tehnici se ponekad koristi takav zapis dekadnih brojeva prema kojem se svaka cifra dekadnog broja neovisno kodira kao grupa od 4 binarne cifre (npr. broj 372 zapisuje se kao 0011 0111 0010). Ovakav zapis zove se ***BCD zapis*** (od engl. Binary Coded Decimal). Za rad sa brojevima u BCD zapisu često se koristi kombinacioni sklop koji se naziva ***BCD inkrementer***. On na svojim ulazima X prima jednu BCD cifru (tj. 4-bitni binarni broj u opsegu od 0 do 9), a na izlazima Y daje vrijednost $Y = X + 1$, osim u slučaju kada je $X = 9$. U tom slučaju, imamo $Y = 0$, a na dodatnom izlazu nazvanom C (od engl. Carry – prenos) javlja se jedinica (u skladu sa činjenicom da je $9 + 1 = 10$, "0" pišemo, a "1" prenosimo). U svim ostalim slučajevima je $C = 0$. Projektirajte formalnim putem BCD inkrementer. Razmotrite moguće realizacije na osnovu struktirno optimalnih mreža u dva nivoa sa AND, OR i NOT logičkim kolima, dobijenih kako iz disjunktivnih, tako i konjunktivnih normalnih formi. Zatim ispitajte kako se realizacija može učiniti ekonomičnijom ukoliko se dozvoli i upotreba EXOR logičkih kola. U svim slučajevima možete podrazumijevati da se na ulazu nikada neće pojaviti kombinacije koje ne predstavljaju legalnu BCD cifru (tj. binarni broj izvan opsega $0 - 9$).

11.14 Kombinacioni sklop poznat pod nazivom ***2-bitni pomjerač*** (ili ***šifter***) posjeduje dva obična ulaza navzana I_1 i I_0, jedan dodatni upravljački ulaz nazvan SHIFT, i dva izlaza nazvana O_1 i O_0. Ukoliko je SHIFT = 0, ulazi I_1 i I_0 se prosto prosljeđuju na izlaze O_1 i O_0 respektivno. Ukoliko je SHIFT = 1, ulaz I_0 se prosljeđuje na izlaz O_1, dok se na izlazu O_0 javlja nula. Projektirajte ovaj kombinacioni sklop na bazi strukturno optimalne mreže u dva nivoa sa AND, OR i NOT logičkim kolima, dobijene iz disjunktivnih normalnih formi.

11.15 Kombinacioni sklop poznat pod nazivom ***2-bitni razmjenjivač*** (ili ***swapper***) posjeduje dva obična ulaza navzana I_1 i I_0, jedan dodatni upravljački ulaz nazvan SWAP, i dva izlaza nazvana O_1 i O_0. Ukoliko je SWAP = 0, ulazi I_1 i I_0 se prosto prosljeđuju na izlaze O_1 i O_0 respektivno. Ukoliko je SWAP = 1, ulaz I_0 se prosljeđuje na izlaz O_1, dok se ulaz I_1 prosljeđuje na izlaz O_0. Projektirajte ovaj kombinacioni sklop na bazi strukturno optimalne mreže u dva nivoa sa AND, OR i NOT logičkim kolima, dobijene iz disjunktivnih normalnih formi.

11.16 Radi otežavanja prisluškivanja trobitnih digitalnih podataka koji se prenose između neka dva uređaja, na ulazu i izlazu komunikacionog kanala priključeni su uređaji (blokovi) za šifriranje i dešifriranje, prema sljedećoj slici:

Blok za šifriranje obavlja funkciju $Y = (5X + 1) \bmod 8$, gdje je X trobitni broj na njegovom ulazu. Blok za dešifriranje, naravno, obavlja inverznu funkciju. Projektirati navedene blokove na bazi strukturno optimalne mreže u dva nivoa sa AND, OR i NOT logičkim kolima, dobijene iz disjunktivnih normalnih formi.

11.17 Pokažite da rezultat sabiranja jednog broja od *m* bita i drugog od *n* bita može imati najviše $\max(m, n)$ bita za $m \neq n$, odnosno $m + 1$ bita za $m = n$.

11.18 Projektirajte kombinacioni sklop koji oduzima dva dvobitna binarna broja A i B. Kako rezultat oduzimanja može biti i negativan, rezultat je predstavljen u kôdu drugog komplementa na 3 bita. Realizaciju prvo zasnujte na strukturno optimalnoj mreži u dva nivoa sa AND, OR i NOT logičkim kolima, dobijenoj iz disjunktivnih normalnih formi, a zatim razmotrite šta se može učiniti ukoliko se dozvoli upotreba mreža sa više nivoa, kao i upotreba EXOR logičkih kola.

11.19 Projektirajte ponovo kombinacioni sklop koji oduzima dva dvobitna binarna broja A i B, ali ovaj put pretpostavite da se nikada neće pojaviti takve vrijednosti na ulazima za koje je $A < B$ (tako da negativni rezultati nisu mogući). Ispitajte mogućnosti realizacije na osnovu strukturno optimalnih mreža u dva nivoa sa AND, OR i NOT logičkim kolima, dobijenih iz disjunktivnih odnosno konjunktivnih normalnih formi, a zatim razmotrite šta se može učiniti ukoliko se dozvoli upotreba EXOR logičkih kola.

11.20 Ispitajte šta će se desiti ukoliko se u sklopu projektiranom u prethodnom zadatku ipak pojave takve vrijednosti ulaza za koje vrijedi $A < B$.

11.21 Projektirajte kombinacioni sklop koji oduzima dva dvobitna binarna broja A i B pod uvjetom da je $A > B$, a u suprotnom daje nulu kao rezultat (ovaj sklop realizira operaciju C = A '–' B, gdje je A '–' B = $\max\{A - B, 0\}$, odnosno A '–' B = $A - B$ za $A \geq B$, i A '–' B = 0 za $A < B$). Ispitajte mogućnosti realizacije na osnovu strukturno optimalnih mreža u dva nivoa sa AND, OR i NOT logičkim kolima, dobijenih iz disjunktivnih odnosno konjunktivnih normalnih formi, a zatim razmotrite šta se može učiniti ukoliko se dozvoli upotreba mreža sa više nivoa.

11.22 Projektirajte kombinacioni sklop koji obavlja operaciju $C = A + B - 3$, pri čemu su ulazni podaci A i B dvobitni binarni brojevi. Izlaz C treba imati minimalan broj bita koji je dovoljan da se predstave svi rezultati. U slučaju da je rezultat negativan, treba ga predstaviti u kôdu drugog komplementa. Realizaciju prvo zasnujte na strukturno optimalnoj mreži u dva nivoa sa AND, OR i NOT logičkim kolima, dobijenoj iz disjunktivnih normalnih formi, a zatim razmotrite šta

se može učiniti ukoliko se dozvoli upotreba mreža sa više nivoa, kao i upotreba EXOR odnosno EXNOR logičkih kola.

11.23 Projektirajte kombinacioni sklop koji obavlja operaciju $C = A^2 + B^2$, pri čemu su ulazni podaci A i B dvobitni binarni brojevi. Izlaz C treba imati minimalan broj bita koji je dovoljan da se predstave svi rezultati. Realizaciju prvo zasnujte na strukturno optimalnoj mreži u dva nivoa sa AND, OR i NOT logičkim kolima, dobijenoj iz disjunktivnih normalnih formi, a zatim razmotrite šta se može učiniti ukoliko se dozvoli upotreba mreža sa više nivoa, kao i upotreba EXOR odnosno EXNOR logičkih kola.

11.24 Koliko najviše bita može imati rezultat množenja jednog broja od *m* bita i drugog od *n* bita?

11.25 Projektirajte kombinacioni sklop koji množi dva broja A i B od po dva bita. Realizaciju prvo zasnujte na strukturno optimalnoj mreži u dva nivoa sa AND, OR i NOT logičkim kolima. Nakon toga, razmotrite kako se može dobiti jednostavnija struktura ukoliko se dozvoli upotreba mreža saviše nivoa, kao i upotreba EXOR logičkih kola.

11.26 Projektirajte kombinacioni sklop koji obavlja operaciju $C = A \cdot (B + 2)$, pri čemu su ulazni podaci A i B dvobitni binarni brojevi. Izlaz C treba imati minimalan broj bita koji je dovoljan da se predstave svi rezultati. Realizaciju prvo zasnujte na strukturno optimalnoj mreži u dva nivoa sa AND, OR i NOT logičkim kolima, dobijenoj iz disjunktivnih normalnih formi, a zatim razmotrite šta se može učiniti ukoliko se dozvoli upotreba mreža sa više nivoa, kao i upotreba EXOR logičkih kola.

11.27 Projektirajte kombinacioni sklop koji dijeli dva broja A i B od po dva bita. Predviđeno je da sklop ima dvije grupe izlaza, Q i R. Na izlazima Q se pojavljuje cijeli dio količnika A/B, a na izlazima R ostatak pri dijeljenju (drugim riječima, $Q = \lfloor A/B \rfloor$ i $R = A \bmod B$). Možete podrazumijevati da se nikada na ulazu neće pojaviti B = 0. Realizaciju zasnujte na strukturno optimalnoj mreži u dva nivoa sa AND, OR i NOT logičkim kolima, dobijenoj iz disjunktivnih normalnih formi.

11.28 Ispitajte šta će se desiti ukoliko se u sklopu projektiranom u prethodnom zadatku ipak na ulazu pojavi vrijednost B = 0.

11.29* Projektirajte kombinacioni sklop sličan sklopu iz Zadatka 11.27, ali uz pretpostavku da je broj A trobitni, a ne dvobitni broj.

11.30* Ispitajte šta će se desiti ukoliko se u sklopu projektiranom u prethodnom zadatku ipak na ulazu pojavi vrijednost B = 0.

11.31 Projektirajte kombinacioni sklop nazvan ***2-bitni paralelni komparator***, koji poredi dva broja A i B od po dva bita. Sklop ima tri izlaza, koja ćemo obilježiti redom sa $Y_{[=]}$,$Y_{[<]}$ i $Y_{[>]}$. U zavisnosti od rezultata poređenja, na jednom i samo jednom od ova tri izlaza treba da se pojavi jedinica, u ovisnosti da li je A = B, A < B ili A > B respektivno. Realizaciju prvo zasnujte na strukturno optimalnoj mreži u dva nivoa sa AND, OR i NOT logičkim kolima, dobijenoj iz disjunktivnih normalnih formi, a zatim razmotrite šta se može učiniti ukoliko se dozvoli upotreba mreža sa više nivoa, kao i upotreba EXNOR logičkih kola.

11.32 Projektirajte kombinacioni sklop, poznat pod nazivom ***višefunkcionalna logička jedinica***, koji posjeduje dva obična jednobitna ulaza nazvana A i B, i tri upravljačka ulaza, nazvana c_2, c_1 i c_0. U zavisnosti od vrijednosti ovih upravljačkih ulaza, na jednobitnom izlazu Y iz sklopa pojavljuje se vrijednost u skladu sa sljedećom tabelom:

$(c_2;c_1;c_0)_2$	$(000)_2$	$(001)_2$	$(010)_2$	$(011)_2$	$(100)_2$	$(101)_2$	$(110)_2$	$(111)_2$
Y	1	$A \vee B$	$\overline{AB}$	$A \oplus B$	$\overline{A \oplus B}$	AB	$\overline{A \vee B}$	0

Sklopovi poput ovog često su sastavni dio mikroprocesorskih struktura. Projektirajte ovaj sklop, na bazi dvonivoske mreže dobijene iz minimalne disjunktivne normalne forme.

11.33* Za konverzionu matricu za upravljanje sedmosegmentnim displejom razmatranu u Primjeru 11.4, nađite disjunktivne normalne forme za funkcije $I_1 \div I_7$ koje omogućavaju najekonomičniju realizaciju na bazi dvonivoske mreže sa AND, OR i NOT logičkim kolima.Vodite računa da maksimizirate broj članova koji se javljaju kao zajednički u više funkcija.

11.34* Projektirajte konverzionu matricu za upravljanje sedmosegmentnim displejom koja radi slično poput konverzione matrice obrađene u Primjeru 11.4, ali za koju ulazne kombinacije koje odgovaraju brojevima od 10 do 15 nisu zabranjene, nego dovode do prikaza odgovarajućih heksadekadnih cifara $A \div F$ na displeju (mogući izgledi displeja za ove kombinacije prikazane su na kraju Primjera 11.4). Naravno, za ulazne kombinacije koje odgovaraju brojevima od 0 do 9, ova konverziona matrica treba da radi isto kao i konverziona matrica obrađena u Primjeru 11.4. Realizaciju ove konverzione matrice zasnujte na

a) Konjunktivnim normalnim formama

b) Disjunktivnim normalnim formama

U oba slučaja, vodite računa da maksimizirate broj članova odnosno faktora koji se javljaju kao zajednički u više funkcija.

11.35 Klasični binarni zapis nije jedini način za zapis cijelih brojeva samo pomoću cifara 0 i 1. U praksi se, u izvjesnim primjenama, koriste i drugi zapisi. Naročito je često u upotrebi tzv. ***Grayov kôd***, koji ima osobinu da se zapisi u Grayovom kôdu dva susjedna cijela broja uvijek razlikuju tačno u jednom bitu, što nije slučaj sa običnim binarnim zapisom (npr. binarni zapisi brojeva 15 i 16 razlikuju se u 4 bita, a brojeva $2^n - 1$ i 2^n u n bita). Na primjer, brojevi od 0 do 15 se u 4-bitnom Grayovom zapisu predstavljaju respektivno kao 0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100, 1100, 1101, 1111, 1110, 1010, 1011, 1001 i 1000. Ova osobina može biti poželjna u nekim primjenima. Projektirajte konverzionu matricu koja 4-bitni broj X na ulazu, zapisan u klasičnom binarnom zapisu, pretvara u 4-bitni broj Y na izlazu, zapisan u Grayovom kôdu, kao i konverzionu matricu koja obavlja inverznu funkciju (tj. koja 4-bitni broj Y na ulazu, zapisan Grayovom kôdu, pretvara u 4-bitni broj X na izlazu, zapisan u klasičnom binarnom zapisu). U oba slučaja pronađite najekonomičnije realizacije, pri čemu je dozvoljeno korištenje EXOR logičkih kola.

11.36 Projektirajte konverzionu matricu koja četverobitni broj $X = (x_3;x_2;x_1;x_0)_2$ zapisan u sistemu "znak i vrijednost" (pri čemu x_3 čuva informaciju o znaku, a $(x_2;x_1;x_0)_2$ o apsolutnoj vrijednosti broja) pretvara u četverobitni broj Y zapisan u *1KK* kôdu (za $X \geq 0$, broj ostaje neizmijenjen, tj. $Y = X$), kao i konverzionu matricu koja obavlja obrnutu funkciju (pretvara Y zapisan u *1KK* kôdu u X zapisan u sistemu "znak i vrijednost"). U oba slučaja pronađite najekonomičnije realizacije, pri čemu je dozvoljeno korištenje EXOR logičkih kola.

11.37 Projektirajte konverzionu matricu koja četverobitni broj $X = (x_3;x_2;x_1;x_0)_2$ zapisan u sistemu "znak i vrijednost" (uz iste konvencije kao u prethodnom zadatku) pretvara u četverobitni broj Y zapisan u *2KK* kôdu, kao i konverzionu matricu koja obavlja obrnutu funkciju. Pri pretvorbi iz *2KK* zapisa u sistem "znak i vrijednost", pretpostavite da se kombinacija 1000, koja odgovara broju –8 i koja se ne može predstaviti u sistemu "znak i vrijednost" sa 4 bita, neće nikada pojaviti na ulazu. U oba slučaja pronađite najekonomičnije realizacije, pri čemu je dozvoljeno korištenje EXOR logičkih kola.

11.38 Projektirajte konverzionu matricu koja četverobitni broj X zapisan u *1KK* kôdu pretvara u četverobitni broj Y zapisan u *2KK* kôdu, uz pretpostavku da je on negativan (tj. za $X \geq 0$, broj ostaje neizmijenjen, tj. $Y = X$), kao i konverzionu matricu koja obavlja obrnutu funkciju (pretvara Y zapisan u *2KK* kôdu u X zapisan u *1KK* kôdu). Pri pretvorbi iz *2KK* zapisa u *1KK* zapis, pretpostavite da se kombinacija 1000, koja odgovara broju –8 i koja se ne može predstaviti u *1KK* zapisu sa 4 bita, neće nikada pojaviti na ulazu. U oba slučaja pronađite najekonomičnije realizacije, pri čemu je dozvoljeno korištenje EXOR logičkih kola.

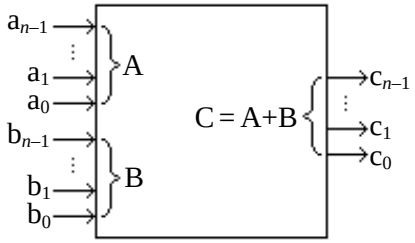
12. Sinteza složenijih kombinacionih sklopova

Kao što smo već rekli, formalnom metodom mogu se projektirati samo sklopovi sa *malim brojem ulaza*. Stoga se sklopovi sa velikim brojem ulaza projektiraju na druge načine. U jednostavnijim slučajevima, ponekad pomaže pristup da se prvo projektira sličan sklop sa manjim brojem ulaza, a zatim indukcijom nasluti kako bi mogle izgledati jednačine koje opisuju sklop sa traženim brojem ulaza (pogledajte, na primjer, zadatke 12.1 i 12.2). Pri tome je, na neki način, potrebno dokazati korektnost naslućenih jednačina, obično koristeći princip matematičke indukcije. Međutim, ovakav pristup je uspješan samo u veoma specifičnim slučajevima. Mnogo univerzalniji pristup sastoji se u tome da funkcionalnost traženog sklopa na neki način *razbijemo na manje sastavne dijelove* (nazvane ***blokovi*** ili ***ćelije***), od kojih se se svaki za sebe može projektirati formalnom metodom, a zatim da sklopimo sve sastavne dijelove u jednu cjelinu. Nažalost, nikakva pravila ne postoje kako bi se to razbijanje moglo izvršiti, već je potrebno upotrijebiti *intuiciju*. Također, sklopovi dobijeni na ovaj način su često vrlo neefikasni sa aspekta brzine rada, mada znaju biti prilično ekonomični. Ovakav pristup, koji nazivamo ***ćelijsko-intuitivni metod projektiranja***, demonstriraćemo kroz nekoliko karakterističnih primjera.

Primjer 12.1:

- Projektirati sklop koji sabira dva broja od po *n* bita (pri čemu je *n* proizvoljan, ali unaprijed fiksiran broj, koji može biti i veoma veliki).

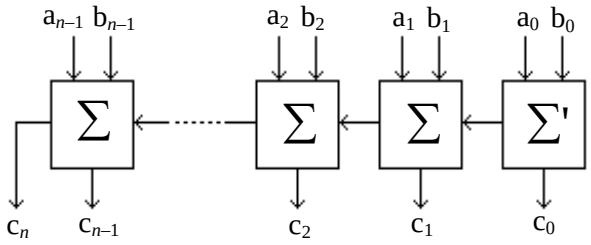
Ovdje formalna metoda očigledno ne dolazi u obzir, s obzirom na činjenicu da bi traženi sklop trebao da ima $2n$ ulaza i n+1 izlaza, jer kad sabiramo dva broja od po n bita, rezultat može imati također n bita ili eventualno n+1 bit ukoliko se pri sabiranju najstarijih bita pojavi prenos. Stoga bi, recimo za $n = 16$, odgovarajuća kombinaciona tablica trebala da ima $2^{2n} = 2^{32} = 4294967296$ redova, što je očigledno neizvodljivo. Pored toga, u postavci problema uopće nije specificirano koliki je n, tako da bismo očito morali da nađemo neki postupak koji se može primijeniti za proizvoljno n.



Da bismo projektirali ovaj sklop, analizirajmo šta mi zapravo radimo kada *ručno sabiramo dva binarna broja*. Prvo ćemo posmatrati jedan konkretan primjer, a zatim ćemo izvući općeniti zaključak:

```
 10110100
+11010110
---------
110001010
```

Očigledno, sabiranje se obavlja počev od bita najmanje težine, prema bitima veće težine, i bilježi se samo *najniža cifra zbira*, dok se viša cifra *pamti kao prenos*. U svakom koraku, sabiraju se *dva bita zajedno sa bitom prenosa* koji je preostao od prethodnog sabiranja (osim pri sabiranju dva najniža bita, kada nema prenosa od ranije). Odavde lako možemo zaključiti da se postupak sabiranja dva binarna broja od po n bita može shematski predstaviti sljedećim blok dijagramom.



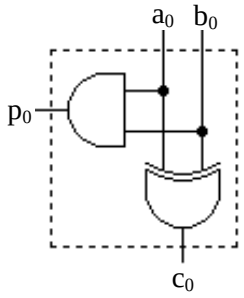
Posljednji blok sa desne strane (označen sa $\sum'$) treba da sabere dvije binarne cifre i da generira dva izlaza: cifru rezultata i informaciju da li ima ili nema prenosa. Svaki sljedeći blok (gledano s desna nalijevo), u suštini sabira tri ulaza: dvije cifre i eventualni prenos koji dolazi iz prethodnog bloka. Blok koji sabira tri ulaza i daje kao izlaze cifru rezultata i informaciju o prenosu naziva se u tehničkoj literaturi ***puni*** (ili ***potpuni***) ***sumator*** (ili ***sabirač***), dok se sličan blok koji ima dva ulaza naziva ***polusumator*** (ili ***polusabirač***). U literaturi se ti blokovi obično označavaju znacima $\sum$ i $\sum'$, pa smo ih i mi tako označili. Često se susreću i oznake FA i HA (od engleskih termina Full Adder i Half Adder). U domaćoj literaturi se umjesto oznake $\sum'$ za polusumator koristi i oznaka PS.

Ovim smo zapravo obavili *najteži dio posla* – razbili smo čitav sklop na blokove $\sum$ i $\sum'$ koji su svaki za sebe dovoljno jednostavni da se mogu projektirati formalnom metodom. Preostaje nam *lakši dio posla*: projektirati unutrašnjost ovih blokova.

Projektirajmo prvo polusumator. Ako njegove ulaze (bite koji se sabiraju) označimo sa a_0 i b_0, cifru rezultata sa c_0 a informaciju o prenosu sa p_0, tada polusumatoru odgovara sljedeća tablica istine:

a_0	b_0	c_0	p_0
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

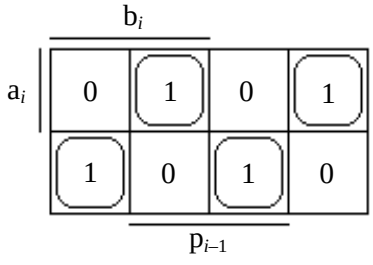
Odavde neposredno očitavamo $c_0 = a_0 \oplus b_0$ i $p_0 = a_0 b_0$. Stoga se strukutura polusumatora može prikazati sljedećom slikom:

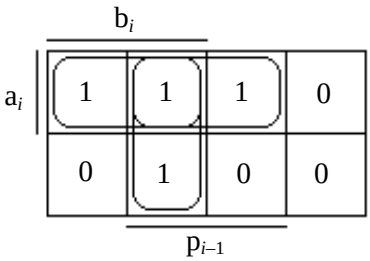


Ostaje da sastavimo puni sumator. Sličnom logikom zaključujemo da njemu odgovara sljedeća tablica istine:

a_i	b_i	p_{i-1}	c_i	p_i
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

Izraze za c_i i p_i odredićemo uz pomoć Veitchovih dijagrama:





Iz ovih dijagrama lako se određuju traženi izrazi:

$$c_i = \bar{a}_i b_i \bar{p}_{i-1} \vee \bar{a}_i \bar{b}_i p_{i-1} \vee a_i b_i p_{i-1} \vee a_i \bar{b}_i \bar{p}_{i-1} = (\bar{a}_i b_i \vee a_i \bar{b}_i)\bar{p}_{i-1} \vee (\bar{a}_i \bar{b}_i \vee a_i b_i) p_{i-1} =$$
$$= (a_i \oplus b_i)\bar{p}_{i-1} \vee \overline{a_i \oplus b_i}\, p_{i-1} = (a_i \oplus b_i) \oplus p_{i-1} = a_i \oplus b_i \oplus p_{i-1}$$

$$p_i = a_i b_i \vee a_i p_{i-1} \vee b_i p_{i-1} = a_i b_i \vee (a_i \vee b_i) p_{i-1}$$

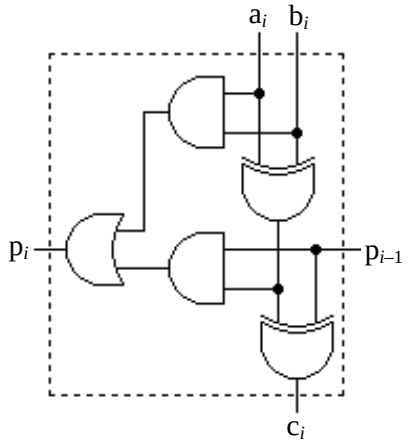
Na ovaj način nam je potrebno jedno troulazno EXOR kolo (ili dva dvoulazna), dva dvoulazna AND kola, i dva dvoulazna OR kola, što ukupno daje 11 (ili 12) ulaza. Interesantno je primijetiti da ukoliko izraz za p_i napišemo u savršenoj disjunktivnoj umjesto u minimalnoj disjunktivnoj normalnoj formi, možemo ga lako preurediti da se u njemu pojave zajednički EXOR član koji postoji u funkciji c_i:

$$p_i = \bar{a}_i b_i p_{i-1} \vee a_i b_i \bar{p}_{i-1} \vee a_i \bar{b}_i p_{i-1} \vee a_i b_i p_{i-1} = (\bar{a}_i b_i \vee a_i \bar{b}_i) p_{i-1} \vee a_i b_i =$$
$$= (a_i \oplus b_i) p_{i-1} \vee a_i b_i$$

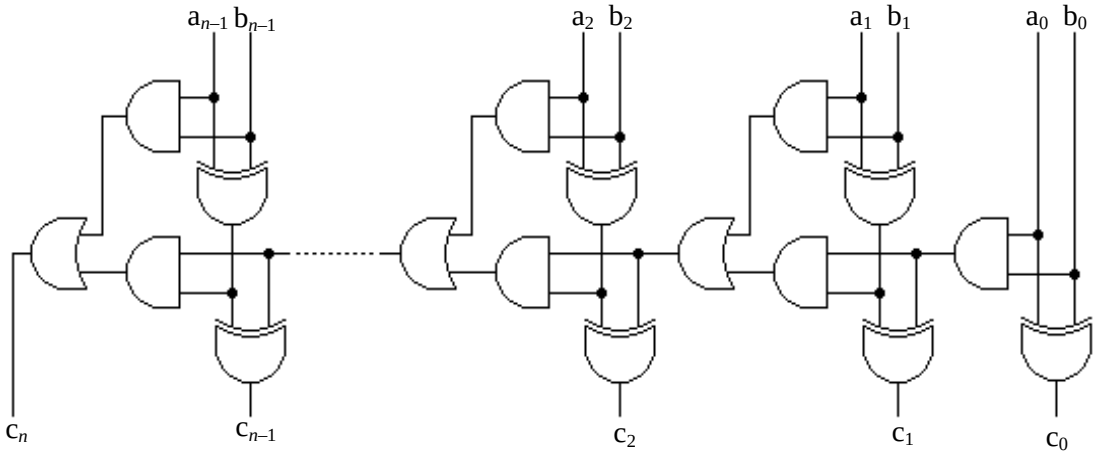
Primjenom ove optimizacije (koja je prilično lukava, i koja bi se teško mogla izvesti samo posmatranjem Veitchovih dijagrama) potrebna su nam dva dvoulazna EXOR kola, dva dvoulazna AND kola i samo jedno dvoulazno OR kolo, što daje ukupno 10 ulaza, čime smo ostvarili malu uštedu. Ista optimizacija mogla se, uz malo više dosjetljivosti, izvršiti i iz minimalnog izraza za funkciju p_i, ako uočimo da je $X \vee Y = XY \vee (X \oplus Y)$, što se lako provjerava:

$$p_i = a_i b_i \vee (a_i \vee b_i) p_{i-1} = a_i b_i \vee [a_i b_i \vee (a_i \oplus b_i)] p_{i-1} = a_i b_i \vee a_i b_i p_{i-1} \vee (a_i \oplus b_i) p_{i-1} =$$
$$= a_i b_i \vee (a_i \oplus b_i) p_{i-1}$$

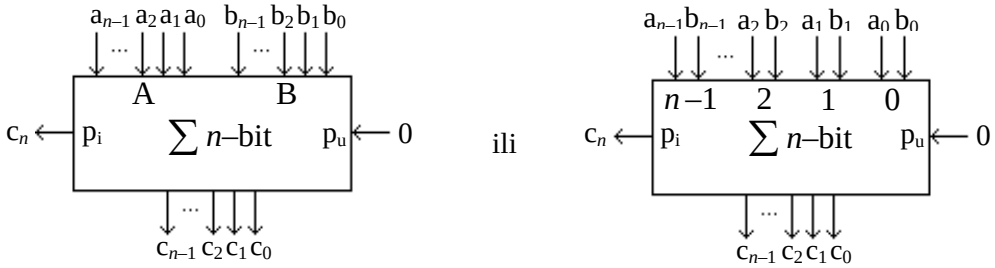
Na osnovu dobijenih funkcija možemo lako nacrtati traženu strukturu punog sumatora:



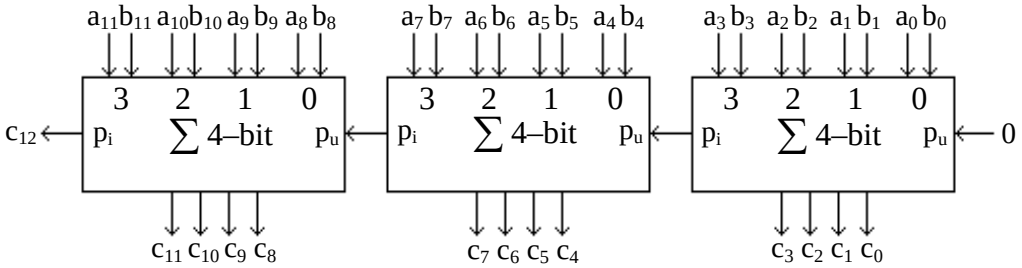
Za kompletiranje rješenja preostaje jos jedan korak – projektirane blokove je potrebno ubaciti u blok dijagram od kojeg smo krenuli. Na taj način dobijamo kompletnu shemu traženog sklopa:



U digitalnoj tehnici se razmotreni blokovi polusumator i puni sumator često prikazuju na shemama kao elementarni funkcionalni blokovi, *bez ulaženja u njihovu strukturu*. Također se, u složenijim shemama, čitav sklop koji smo upravo projektirali, i koji se naziva ***n-bitni paralelni binarni sumator***, često crta kao *jedan blok*, kao na sljedećim slikama:



Na ovim slikama javlja se i dodatni ulaz obilježen sa p_u. Naime, u praksi se obično posljednji polusumator u paralelnom sumatoru zamjenjuje punim sumatorom pri čemu mu je treći ulaz obilježen sa p_u (ili p_{-1}). Normalno se na taj ulaz uvijek dovodi nula. Međutim, postojanje ovog ulaza omogućava neke efikasne trikove. Na primjer, sljedeća slika prikazuje kako od tri 4-bitna paralelna sumatora napraviti jedan 12-bitni paralelni sumator:



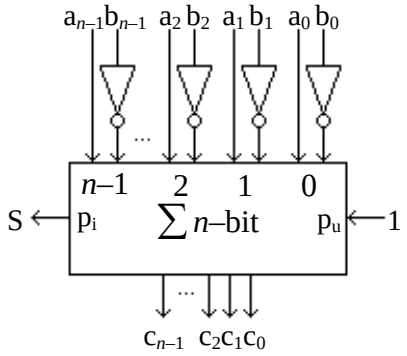
Stoga se ulaz p_u često naziva ***ulazni prenos*** (engl. *input carry*) jer omogućava prihvatanje prenosa iz prethodnog bloka prilikom kaskadnog vezivanja blokova. Analogno, izlaz p_i predstavlja ***izlazni prenos*** (engl. *output carry*).

Projektirani sklop je optimalan po pitanju broja upotrijebljenih elemenata, ali je *izrazito neoptimalan po pitanju brzine rada*. Naime, podsjetimo se da svako logičko kolo unosi neizbježno kašnjenje. Pogledajmo sada kako je formiran ovaj sklop. Izlaz c_1 i prenos p_1 ne mogu se formirati dok se ne formira prenos p_0, izlaz c_2 i prenos p_2 ne mogu se formirati dok se ne formira prenos p_1, itd. Dakle, za formiranje prenosa p_2 treba čekati na formiranje prenosa p_1, za formiranje prenosa p_3 treba čekati na formiranje prenosa p_2, itd. Izlazi c_i kasne jedan za drugim kako *i* raste, i kašnjenje je tim veće što je veće *i*. Slijedi da izlaz c_n može akumulirati prilično kašnjenje, pogotovo ukoliko je *n* veliko.

Opisani paralelni sumator naziva se i ***sumator sa serijskom propagacijom prenosa***, zbog činjenice da se informacija o prenosu prenosi redom (serijski) kroz sastavne blokove zdesna nalijevo. U brzim digitalnim sklopovima (npr. savremenim digitalnim računarima) ovakvi sumatori ne mogu zadovoljiti zbog svoje sporosti. Zbog toga su formirani i brži paralelni sumatori, nazvani ***sumatori sa paralelnim formiranjem prenosa***. Građa ovakvih sumatora je dosta složenija, i u nju nećemo ulaziti. Osnovna ideja je da se informacija o prenosu ne propagira postupno kroz čitavu mrežu, već da se formalnim putem projektira dvonivoska mreža (optimalna sa aspekta brzine) koja će odmah generirati sve cifre rezultata za grupu od po nekoliko bita (obično 4), i informaciju o izlaznom prenosu. Takvi paralelni sumatori mogu se vezati u kaskadu na isti način kao i sumatori sa serijskom propagacijom prenosa.

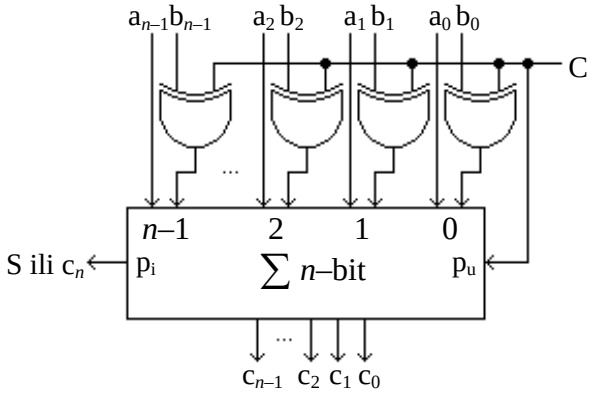
Na sličan način se može napraviti sklop koji oduzima dva broja od po *n* bita. Za tu svrhu bi trebalo raščlaniti sklop na elementarne blokove koje bi mogli nazvati ***poluoduzimač*** i ***puni oduzimač***. Poluoduzimač bi na osnovu dvije cifre a_0 i b_0 na ulazu trebao generirati cifru rezultata i informaciju da li nam je trebalo posuđivanje ili ne. Puni oduzimač bi trebao imati tri ulaza: dvije cifre i prethodnu posudbu (engl. *borrow*) a na osnovu njih bi trebao generirati novu cifru i novu posudbu. Projektiranje ovakvog sklopa ostavlja se čitatelju ili čitateljki za vježbu.

Sklop za oduzimanje dva binarna broja od po *n* bita može se napraviti i jednostavnom prepravkom *n*-bitnog paralelnog sumatora, uz pomoć relacije $A - B = A + (-B)$. Slijedi da ukoliko broj –B zapišemo u obliku kôda drugog komplementa (*2KK*), tada oduzimanje lako svodimo na sabiranje. Da bismo broj –B predstavili u *2KK* zapisu, potrebno je invertirati sve bite broja B, i na rezultat dodati jedinicu. Invertiranje svih bita lako je izvesti pomoću invertora, dok je dodavanje preostale jedinice najbolje izvesti dovođenjem jedinice na ulaz p_u, kao na sljedećoj slici:



Lako se provjerava da se na izlazu obilježenom sa S (Sign) dobija znak rezultata: S = 0 za pozitivan rezultat, i S = 1 za negativan rezultat.

Ukoliko uočimo da vrijedi $X \oplus 0 = X$ i $X \oplus 1 = \overline{X}$, možemo objediniti funkcije sabiranja i oduzimanja u jedinstven sklop prikazan na sljedećoj slici. Ovaj sklop za C = 0 vrši sabiranje, a za C = 1 oduzimanje brojeva dovedenih na ulaze:



Na ovaj način smo dobili jedan višefunkcionalni sklop, koji predstavlja specijalan (i to vrlo jednostavan) slučaj višefunkcionalnih sklopova koji spadaju u klasu kombinacionih sklopova nazvanih ***aritmetičko-logičke jedinice***, o kojima će kasnije biti više govora.

Intuitivnim putem nije teško projektirati i sklopove za množenje (mada se kao rezultat dobijaju prilično glomazni sklopovi). Primijetimo prvo da produkt dva broja koji imaju *m* i *n* bita respektivno može imati najviše *n*+*m* bita, jer je $(2^n-1)\cdot(2^m-1) = 2^{n+m} - 2^n - 2^m + 1$ što je uvijek manje od $2^{n+m}-1$. S druge strane, produkt zaista može imati tačno *m*+*n* bita, jer je ovaj produkt veći od $2^{n+m-1}-1$, pa se ne može predstaviti sa *m*+*n*–1 bita. Dakle, binarni množač koji množi dva broja od *m* i *n* bita respektivno, mora imati *m*+*n* ulaza i *m*+*n* izlaza.

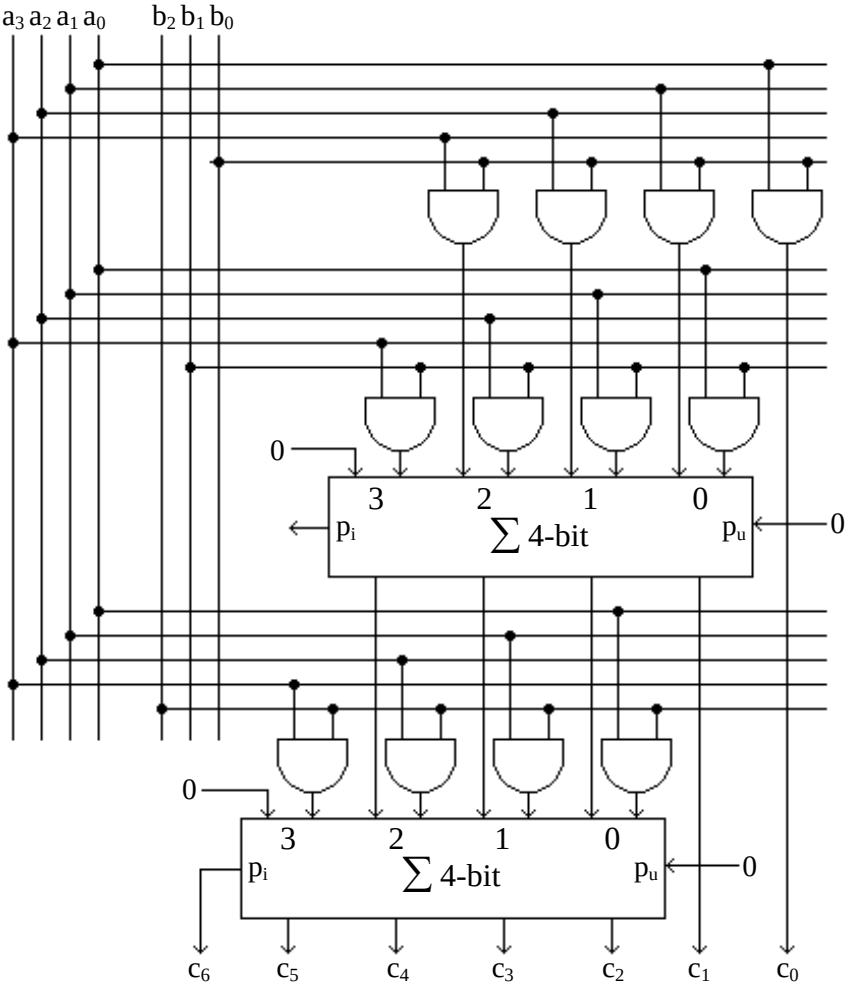
Primjer 12.2:

- Formirati sklop koji množi dva broja, od kojih jedan ima 4 bita, a drugi 3 bita.

Traženi sklop ima 7 ulaza (a_3, a_2, a_1, a_0, b_2, b_1 i b_0) i 7 izlaza (c_6, c_5, c_4, c_3, c_2, c_1 i c_0), pri čemu je jasno da je $A = (a_3;a_2;a_1;a_0)_2$, $B = (b_2;b_1;b_0)_2$, $C = (c_6;c_5;c_4;c_3;c_2;c_1;c_0)_2$ i $C = A \cdot B$. Razmotrimo prvo kako bismo ručno računali produkt $A \cdot B$:

$$
\begin{array}{rcccccccc}
 & & a_3 & a_2 & a_1 & a_0 & \cdot & b_2 & b_1 & b_0 \\
\hline
 & & a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0 & & & & \\
 & a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 & & & & & \\
+ & a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 & & & & & \\
\hline
c_6 & c_5 & c_4 & c_3 & c_2 & c_1 & c_0 & & &
\end{array}
$$

Članove oblika a_ib_j lako formiramo uz pomoć AND kola, nakon čega dobijene parcijalne produkte možemo sabrati uz pomoć paralelnih sumatora. Na osnovu provedenog razmatranja nije teško sastaviti shemu traženog sklopa:



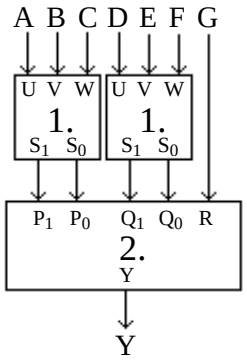
Sličnim postupkom moguće je realizirati i dijeljenje, samo što bi se umjesto paralelnih sabirača morali koristiti paralelni oduzimači (koji se također mogu realizirati pomoću paralelnih sabirača, kao što smo već vidjeli).

Primjer 12.3:

- Projektirati sklop sa sedam ulaza (A, B, C, D, E, F i G) i jednim izlazom (Y), na kojem treba da se pojavi logička jedinica ako i samo ako se na barem pet od sedam ulaza nalazi logička jedinica.

Kako traženi sklop ima sedam ulaza, njegovo projektiranje formalnom metodom je prilično nepraktično. Nikakav problem nije napisati direktno savršenu disjunktivnu normalnu formu funkcije koja opisuje rad sklopa (to bi bila disjunkcija svih mogućih mintermi u kojima se barem pet promjenljivih javlja bez znaka negacije), ali bi sklop realiziran iz takve forme zahtijevao 29 AND kolo sa po 7 ulaza, jedno OR kolo sa 29 ulaza, i 7 invertora (ukupno 239 ulaza), što je očigledno veoma neekonomično. Potražimo stoga bolje rješenje intuitivnim putem.

Na prvi pogled ne vidi se nikakav jednostavan način na koji bi se traženi sklop mogao razbiti na blokove sa manjim brojem ulaza. Međutim, nakon malo razmišljanja, može se uočiti da razbijanje prikazano na sljedećoj slici može biti od koristi:

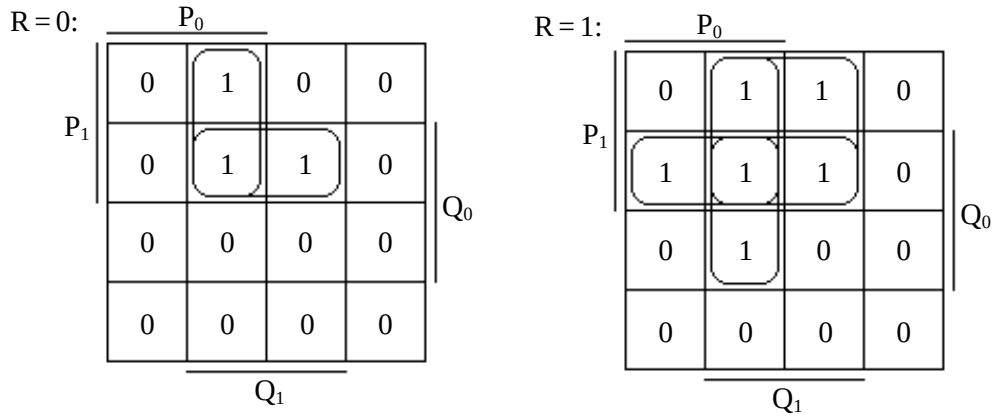


Blokovi označeni sa "**1.**" treba da prebroje na koliko se njihovih ulaza označenih sa U, V i W nalaze logičke jedinice, i da te informacije (koje mogu biti 0, 1, 2 ili 3) proslijede na svoje izlaze S_1 i S_0 kao binarni broj (00, 01, 10 ili 11). Te informacije, zajedno sa ulazom G koji nije obrađen u blokovima "**1.**" treba da posluže bloku "**2.**"da utvrdi koliko je ulaza na logičkoj jedinici, i da u zavisnosti od toga odredi izlaz Y. Lako je vidjeti da su pomenute informacije dovoljne. Na primjer, neka je na ulazima u blok "**2.**" $P_1 = 1$, $P_0 = 0$, $Q_1 = 0$, $Q_0 = 1$ i $R = 1$. Na osnovu $P_1 = 1$ i $P_0 = 0$ zaključujemo da su od tri ulaza A, B i C dva bila na logičkoj jedinici. Slično, na osnovu $Q_1 = 0$ i $Q_0 = 1$ zaključujemo da je od tri ulaza D, E i F jedan na logičkoj jedinici.Konačno, na osnovu $R = 1$ zaključujemo da je ulaz G na logičkoj jedinici. Slijedi da su ukupno četiri ulaza na logičkoj jedinici, stoga treba da bude $Y = 1$.

Ovim razbijanjem je problem pojednostavljen jer blokovi "**1.**" imaju po 3 ulaza, a blok "**2.**" pet ulaza, što je već podnošljivo. Nije nikakav problem sastaviti tablicu istine za blokove "**1.**" i projektovati njihovu strukturu. Međutim, nakon malo razmišljanja, i bez pisanja tablice istine može se zaključiti da blokovi "**1.**" nisu ništa drugo nego *puni sumatori* koje smo već koristili. Zaista, ukoliko U, V i W mogu imati samo vrijednosti 0 ili 1, tada je broj jedinica na ulazima jednak U+V+W, tako da su S_1 i S_0 zapravo izlazi prenosa i cifre sume iz punog sumatora. Ostaje dakle da projektiramo blok "**2.**" Kako se radi o bloku sa 5 ulaza, trebala bi nam tablica istine sa $2^5 = 32$ reda. Da prištedimo na prostoru, ovu tablicu ćemo prikazati na sljedeći način, u kojem su u posebnim stupcima razdvojene vrijednosti izlaza Y za $R = 0$ i za $R = 1$:

P_1	P_0	Q_1	Q_0	Y (R=0)	Y (R=1)
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	0	0
0	1	0	0	0	0
0	1	0	1	0	0
0	1	1	0	0	0
0	1	1	1	0	1
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	0	1
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	1	0	1
1	1	1	0	1	1
1	1	1	1	1	1

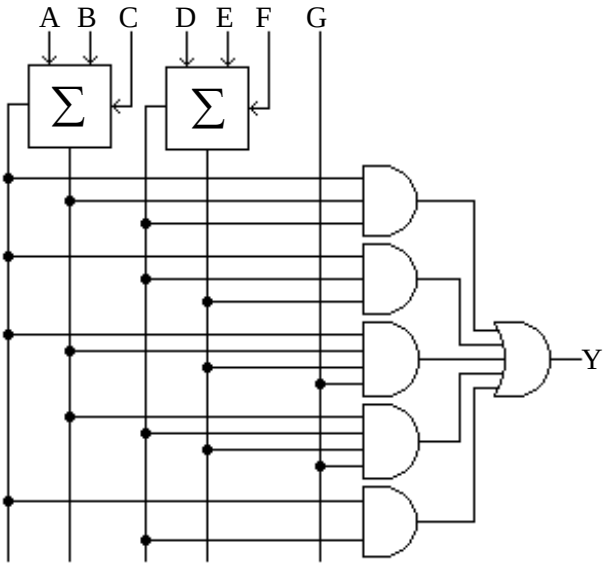
Ovakav prikaz tablice istine ne samo da štedi prostor. već i olakšava minimizaciju funkcije Y pomoću dva Veitchova dijagrama za funkcije od 4 promjenljive, jer se odmah kolone koje odgovaraju vrijednostima funkcije Y za R = 0 i R = 1 mogu prepisati u dva Veitchova dijagrama kao da se radi o dvije odvojene funkcije (pod uvjetom da upravo R uzmemo za preostalu promjenljivu). Odgovarajući Veitchovi dijagrami, sa pripadnim konturama koje su formirane kao u Primjeru 9.8 (u kome je razmatrana minimizacija funkcija od 5 promjenljivih) prikazani su na sljedećoj slici:



Na osnovu ovih dijagrama imamo:

$$Y = (P_1P_0Q_1 \vee P_1Q_1Q_0)\,\overline{R} \vee (P_1P_0Q_1 \vee P_1Q_1Q_0 \vee P_1P_0Q_0 \vee P_0Q_1Q_0 \vee P_1Q_1)\,R =$$
$$= P_1P_0Q_1 \vee P_1Q_1Q_0 \vee P_1P_0Q_0R \vee P_0Q_1Q_0R \vee P_1Q_1R$$

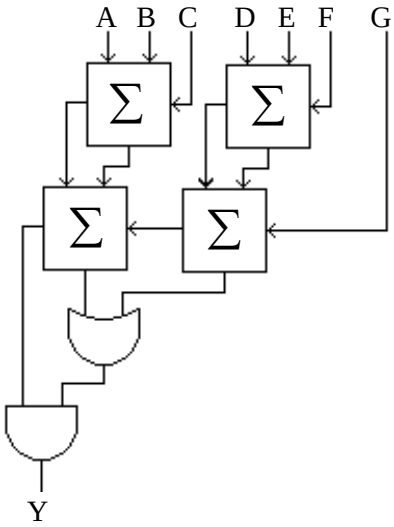
Sada nije teško nacrtati shemu traženog sklopa, čime je projektiranje završeno:



Vrijedi napomenuti sa se, ukoliko raspolažemo sa punim sumatorima kao gotovim komponentama, problem može riješiti još jednostavnije, jer se i blok "**2.**" može relativno lako svesti na sumatore. Naime, ukoliko posmatramo grupe ulaza $(P_1;P_0)_2$ i $(Q_1;Q_0)_2$ kao dva dvobitna binarna broja, tada zbir $(P_1;P_0)_2 + (Q_1;Q_0)_2 + R$ daje kao rezultat *broj ulaza* $A \div G$ koji su na logičkoj jedinici. Ovaj zbir lako možemo formirati pomoću 2-bitnog paralelnog sumatora na čiji jedan par ulaza dovodimo P_1 i P_0, na drugi par ulaza Q_1 i Q_0, dok R dovodimo na ulazni prenos. Sada na osnovu broja ulaza $A \div G$ koji su na logičkoj jedinici koji dobijamo na izlazu iz paralelnog sumatora, možemo lako zaključiti da li je taj broj veći ili jednak od 5 ili nije. Preciznije, ukoliko izlaze iz paralelnog sumatora označimo sa c_2, c_1 i c_0 (c_2 je zapravo izlazni prenos), tada za Y možemo formirati sljedeću tablicu istine:

c_2	c_1	c_0	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Odavde je sasvim lako odrediti funkciju Y u zavisnosti od c_2, c_1 i c_0 (npr. pomoći Veitchovog dijagrama, koji nećemo crtati), koja glasi $Y = c_2c_1 \vee c_2c_0 = c_2(c_1 \vee c_0)$. Stoga, na osnovu svega do sada rečenog, možemo lako nacrtati sljedeću shemu traženog sklopa (pri čemu smo 2-bitni paralelni sumator također prikazali preko punih sumatora kao njegovih sastavnih dijelova):



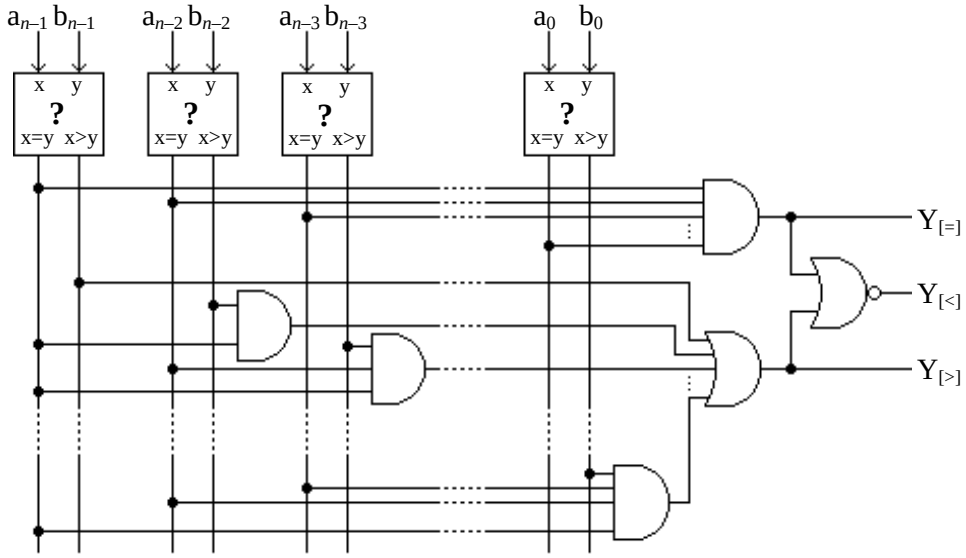
Primijetimo da je posljednji način rješavanja ovog problema logički ekvivalentan sa sljedećim pristupom: izračunati zbir A + B + C + D + E + F + G, a zatim testirati da li je ovaj zbir veći ili jednak od 5. Upravo razmotreni primjer je veoma ilustrativan i poučan, jer demonstrira činjenicu da je puni sumator veoma koristan blok za sintezu složenijih sekvencijalnih sklopova, koji se lijepo može upotrijebiti čak i na mjestima gdje to nije očigledno na prvi pogled (kao u ovom primjeru).

Primjer 12.4:

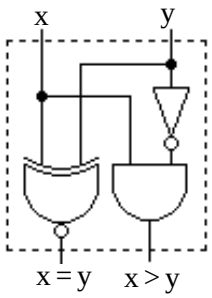
- Projektirati sklop (tzv. ***n-bitni paralelni binarni komparator***) koji poredi dva broja A i B od po *n* bita. Sklop treba da na jedan i samo jedan od svoja tri izlaza nazvana redom $Y_{[=]}$, $Y_{[<]}$ i $Y_{[>]}$ izbaci logičku jedinicu u zavisnosti od toga da li je A = B, A < B ili A > B respektivno.

Ovaj problem ćemo riješiti na tri različita načina, kao ilustraciju različitih pristupa rješavanju iste problematike. Razmotrimo prvo kako bi se uopće mogla porediti dva binarna broja. Označimo brojeve koje se porede sa $A = (a_{n-1};a_{n-2};a_{n-3};...a_1;a_0)_2$ i $B = (b_{n-1};b_{n-2};b_{n-3};...b_1;b_0)_2$. Jasno je da je A = B ako i samo ako su svi odgovarajući biti brojeva A i B jednaki, tj. ako je $a_{n-1} = b_{n-1}$, $a_{n-2} = b_{n-2}$, $a_{n-3} = b_{n-3}$, ... $a_1 = b_1$ i $a_0 = b_0$. S druge strane, imamo A > B ako je neki *i*-ti bit broja A veći od odgovarajućeg bita broja B, dok su svi biti na većim težinama od *i*-tog jednaki u brojevima A i B. Drugim riječima, vrijedi A > B ukoliko je $a_{n-1} > b_{n-1}$ ili ukoliko je $a_{n-1} = b_{n-1}$ i $a_{n-2} > b_{n-2}$ ili ukoliko je $a_{n-1} = b_{n-1}$, $a_{n-2} = b_{n-2}$ i $a_{n-3} > b_{n-3}$ itd. Sličnu logiku možemo izvesti i za A < B. Međutim, ista stvar se može izvesti i jednostavnije ukoliko primijetimo da je A < B ako i samo ako nije niti A = B niti A > B, što znači da izlaz $Y_{[<]}$ možemo formirati na osnovu izlaza $Y_{[=]}$ i $Y_{[>]}$ primjenom samo jednog NOR logičkog kola.

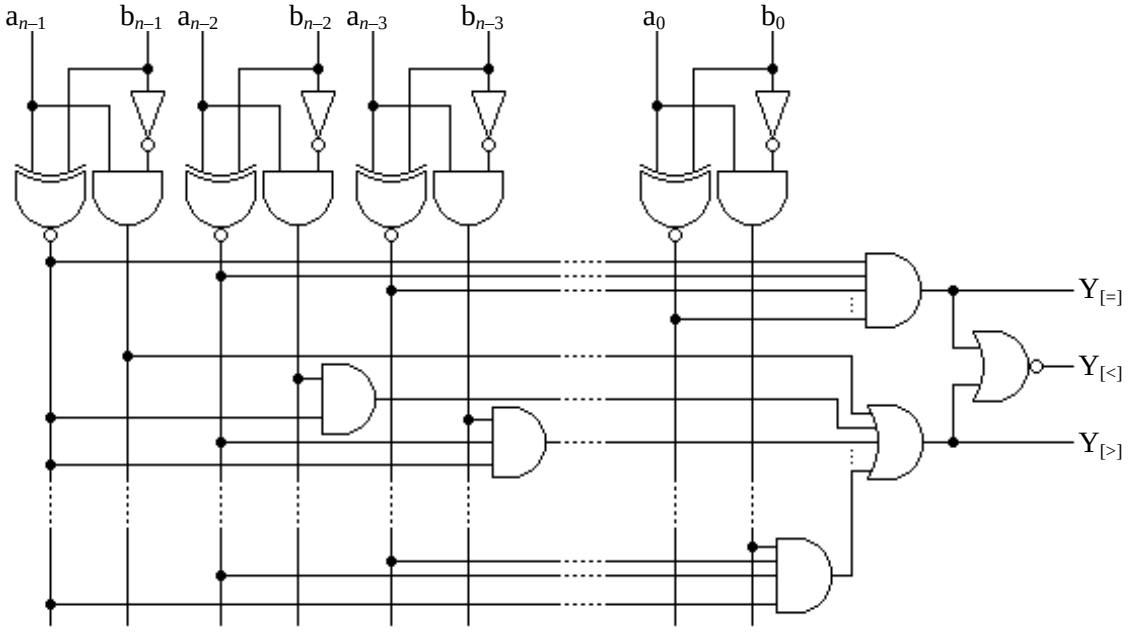
Iz provedenog razmatranja slijedi da se posmatrani problem može riješiti tako što se formiraju blokovi (označeni na sljedećoj blok strukturi znakom "**?**") koji će ispitivati kakav je međusobni odnos pojedinačnih bita a_i i b_i, nakon čega se formira dodatna logika koja će iz rezultata poređenja pojedinačnih bita izvesti konačan zaključak u skladu sa provedenim razmatranjem:



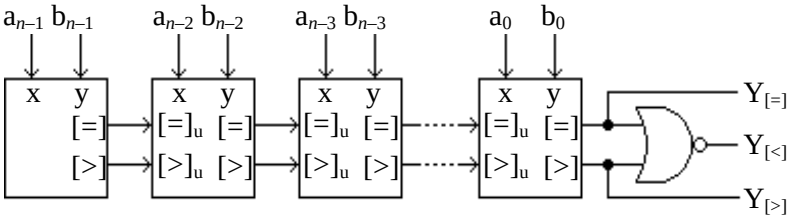
Blok označen sa "**?**" je krajnje jednostavne strukture. Lako je zaključiti da može biti $x > y$ jedino ukoliko je $x = 1$ i $y = 0$, dok $x = y$ može biti ukoliko je $x = 0$ i $y = 0$ ili $x = 1$ i $y = 1$. Stoga, izlazu $x > y$ odgovara logička funkcija $x\overline{y}$, dok izlazu $x = y$ odgovara logička funkcija $xy \vee \overline{x}\overline{y}$ odnosno $\overline{x \oplus y}$:



Sada nije nikakav problem nacrtati kompletnu shemu traženog sklopa:



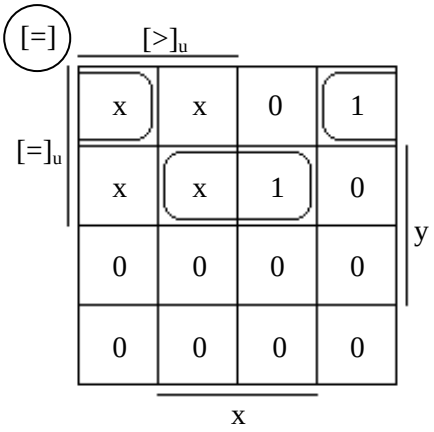
U predloženom rješenju sklop se sastojao od niza ćelija koje međusobno "ne sarađuju" i od dodatne logike (relativno složene) koja donosi konačan zaključak. Problemu se može pristupiti i na drugi način, formiranjem ćelija koje međusobno "sarađuju" tako što prenose informacije jedna drugoj, čime otpada potreba za dodatnom logikom, ali se *usložnjava struktura same ćelije*. Naime, primijetimo da kada poredimo dva binarna broja, mi vršimo poređenje slijeva nadesno, pri čemu *možemo prekinuti analizu* čim naiđemo na dvije različite cifre a_i i b_i, jer dalji rezultat poređenja više ne ovisi od cifara manje težine od *i*. Slijedi da je moguće formirati niz ćelija na sljedeći način. Krajnja lijeva ćelija poredi dva bita najviše težine, i prenosi tu informaciju narednoj ćeliji. Svaka sljedeća ćelija prvo analizira ulaznu informaciju. Ukoliko ulazna informacija kaže da je $A > B$ ili $A < B$ (na osnovu dotadašnjih poređenja), ćelija samo tu informaciju prenosi sljedećoj ćeliji, jer je rezultat poređenja *već određen*, i više *ne zavisi od bita koji se nalaze na toj i svim nižim pozicijama*. Jedino ukoliko ulazna informacija kaže da je $A = B$ (na osnovu dotadašnjih poređenja), ćelija treba da provjeri odgovarajuće bite, i da na osnovu njih odredi informaciju koja će se prenositi dalje. Ovom rezonovanju odgovara sljedeća blok struktura:

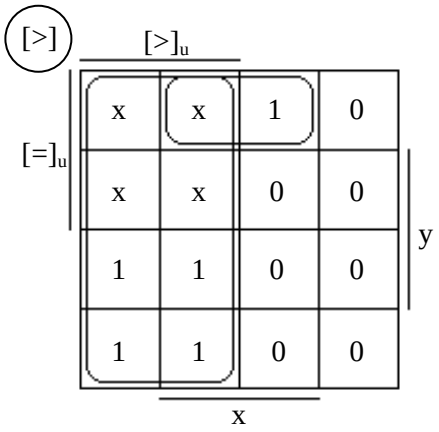


Krajnja lijeva ćelija očigledno je identična ćeliji koju smo kreirali prilikom prvog načina rješavanja problema, dok za ostale ćelije možemo formirati sljedeću tablicu istine:

$[=]_u$	$[>]_u$	x	y	[=]	[>]
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	0	0
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	0	0
1	0	1	0	0	1
1	0	1	1	1	0
1	1	0	0	x	x
1	1	0	1	x	x
1	1	1	0	x	x
1	1	1	1	x	x

Ovoj tablici odgovaraju sljedeći Veitchovi dijagrami:



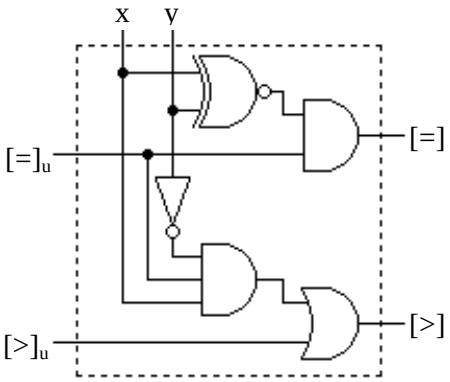


Posljednja četiri reda u tablici istine popunjena su sa "x", s obzirom da ulazna informacija ne može imati vrijednosti $[=]_u$ i $[>]_u$ istovremeno na jedinici. Na osnovu formiranih dijagrama neposredno slijedi:

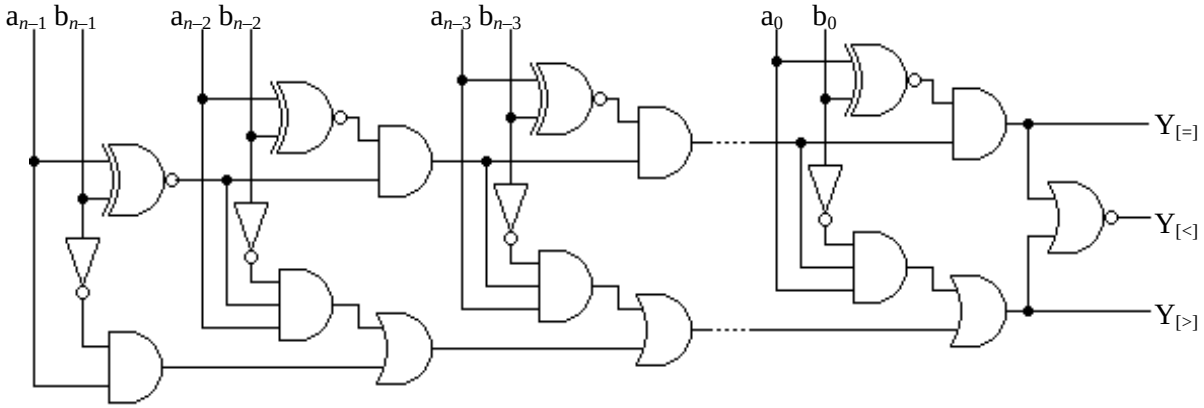
$$[=] = \overline{x}\,\overline{y}\,[=]_u \vee x\,y\,[=]_u = (\overline{x}\,\overline{y} \vee x\,y)\,[=]_u = \overline{x \oplus y}\,[=]_u$$

$$[>] = [>]_u \vee x\,\overline{y}\,[=]_u$$

Stoga, tražene ćelije imaju strukturu kao na sljedećoj slici:



Konačno, možemo nacrtati i kompletnu shemu traženog sklopa:

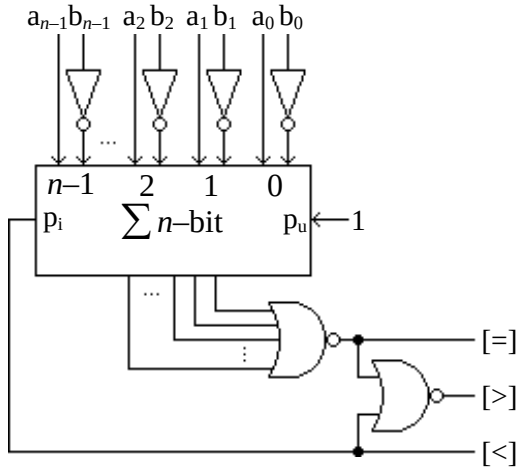


Interesantno je uporediti predložena dva rješenja sa aspekta efikasnosti. Prvo rješenje zahtijeva $n(n+1)/2+7n+1$, a drugo $10n-3$ ulaza u logička kola (provjeriti ovo), odakle slijedi da je drugo rješenje efikasnije sa aspekta broja ulaza u logička kola. Razlika nije velika za manje vrijednosti n (recimo za $n=4$), ali za $n=32$ ta razlika iznosi $753:317$ u korist drugog rješenja. S druge strane, prvo rješenje je *mnogo brže* za veliko n, jer u njemu ulazni signal prođe kroz najviše četiri logička kola do izlaza $Y_{[=]}$ i $Y_{[>]}$ (a pet do izlaza $Y_{[<]}$) neovisno od n, dok u drugom rješenju neki ulazni signali prolaze kroz više od n logičkih kola dok dođu do izlaza (npr. ulazni signal b_{n-2} prolazi kroz n+2 logička koja prije nego što dođe do izlaza $Y_{[<]}$), čime se znatno povećava kašnjenje. Za koje ćemo se rješenje odlučiti, prvenstveno zavisi od toga da li nam je važnija brzina rada ili broj upotrijebljenih komponenti. Prvo rješenje se u tehničkoj literaturi naziva ***brzi paralelni binarni komparator***, dok je drugo rješenje poznato pod imenom ***paralelni binarni komparator sa serijskom propagacijom***.

U razmotrenom primjeru paralelnog binarnog komparatora sa serijskom propagacijom, informacija se propagira kroz ćelije *slijeva nadesno*, prateći logiku koju čovjek primjenjuje prilikom poređenja dva binarna broja, za razliku od paralelnog binarnog sumatora, kod kojeg se informacija propagira zdesna nalijevo. Interesantno je da je moguće napraviti paralelni binarni komparator kod kojeg će se informacija propagirati *zdesna nalijevo*, od bita manje težine prema bitima više težine, kao kod paralelnog binarnog sumatora (s druge strane, nije moguće napraviti paralelni binarni sumator kod kojeg bi se informacija prenosila slijeva nadesno, od bita više težine ka bitima manje težine). Čitateljima i čitateljkama se

ostavlja za vježbu da razmotre kako bi ovakav komparator mogao da izgleda, i da uporede njegovu efikasnost sa do sada razmotrenim rješenjima za paralelni binarni komparator.

Za kraj, razmotrićemo još jedno rješenje za paralelni binarni komparator, koje se dosta često koristi u računarskoj tehnici, a koje je posebno interesantno zbog toga što svodi postavljeni problem na problem koji smo već ranije riješili. Naime, ako označimo $C = A - B$, tada iz $A = B$, $A < B$ ili $A > B$ respektivno slijedi $C = 0$, $C < 0$ ili $C > 0$. Odavde zaključujemo da je dovoljno oduzeti brojeve A i B, što možemo izvesti na već opisani način. Nakon toga, ukoliko je rezultat negativan, što možemo utvrditi posmatrajući izlaz S iz oduzimača, zaključujemo da je $A < B$. Ukoliko je rezultat nula, što možemo utvrditi primjenom NOR operacije na sve izlaze iz oduzimača, zaključujemo da je $A = B$. Konačno, $A > B$ ukoliko nije niti $A < B$ niti $A = B$. Slijedi da se paralelni binarni komparator može realizirati i shemom sa sljedeće slike:



Efikasnost ovog rješenja najviše zavisi od toga kako je realiziran paralelni binarni sumator koji se u njemu pojavljuje, a tipično leži negdje između efikasnosti prvog i drugog razmotrenog rješenja (treba napomenuti da se dobija veoma spor sklop ukoliko se sumator realizira kao u Primjeru 12.1).

U ovom poglavlju izloženi su neki karakteristični primjeri projektiranja složenijih kombinacionih sklopova na bazi ćelijsko-intuitivne metode. Vidimo da je projektiranje tim lakše što smo više sklopova već isprojektirali, jer se veoma često dešava da se neki od blokova koji su razvijeni za potrebe jednog sklopa mogu upotrijebiti i u nekom drugom, na prvi pogled potpuno nevezanom sklopu. Zbog toga je za efikasnu primjenu ćelijsko-intuitivne metode potrebno isprojektirati nekoliko karakterističnih sklopova, koji će se kasnije moći efikasno iskoristiti kao "gradivni elementi" mnogih drugih sklopova. Takvim karakterističnim kombinacionim sklopovima posvećeno je sljedeće poglavlje.

(?) Pitanja i zadaci

12.1 U Zadatku 11.12 smo formalnim putem projektirali 4-bitni binarni inkrementer. Posmatrajući dobijeno rješenje, intuitivnim putem naslutite kako bi izgledale jednačine koje opisuju rad *n*-bitnog binarnog inkrementera, pri čemu je *n* proizvoljan, ali unaprijed fiksiran broj.

12.2 U Zadatku 11.35 smo formalnim putem projektirali konverzionu matricu koja 4-bitni broj na ulazu zapisan u klasičnom binarnom zapisu konvertira u 4-bitni broj na izlazu zapisan u Grayevom kôdu, kao i konverzionu matricu koja obavlja obrnutu pretvorbu. Posmatrajući dobijeno rješenje, intuitivnim putem naslutite kako bi izgledale jednačine koje opisuju rad konverzione matrice koja *n*-bitni broj na ulazu zapisan u klasičnom binarnom zapisu konvertira u *n*-bitni broj na izlazu zapisan u Grayevom kôdu, kao i jednačine koje opisuju rad konverzione matrice koja obavlja obrnutu pretvorbu. Podrazumijeva se da je *n* proizvoljan, ali unaprijed fiksiran broj.

12.3 Nacrtajte strukturu punog sumatora koristeći samo NAND logička kola.

12.4 Pokažite kako se može konstruisati puni sumator pomoću dva polusumatora, i eventualno ponešto dodatne logike. U čemu se suštinski razlikuje takva realizacija punog sumatora u odnosu na neposrednu realizaciju (osim po složenosti)?

12.5 Vidjeli smo da se potpuni sumator često može iskoristiti na prilično neočekivane načine. Pretpostavimo sada da na raspolaganju imamo samo *jedan potpuni sumator* (koji ne možemo rastaviti na sastavne dijelove, npr. zato što je izveden u formi čipa) i ništa drugo. Pokažite kako je pomoću njega moguće realizirati sljedeće logičke funkcije:

a) $Y = \overline{A}$ b) $Y = AB$ c) $Y = A \vee B$ d) $Y = A \oplus B$ e) $Y = \overline{A \oplus B}$

12.6 Pokažite da je koristeći *samo potpune sumatore* (i ništa drugo) moguće realizirati ma kakvu logičku funkciju (ili skupinu logičkih funkcija).

12.7 U Primjeru 12.1 smo intuitivnim putem projektirali *n*-bitni paralelni binarni sumator. Uporedite ovakav dizajn za $n = 2$ sa aspekta funkcionalnosti, efikasnosti i brzine sa 2-bitnim binarnim sumatorom koji je formalnim putem projektiran u Primjeru 11.3. Šta se može zaključiti?

12.8 Objasnite ulogu ulaznog prenosa prilikom realizacije *n*-bitnih paralelnih binarnih sumatora kao standardnih komponenti za gradnju složenijih sklopova.

12.9 Rečeno je da se ćelijsko-intuitivnim metodom tipično dobijaju rješenja koja su često veoma neefikasna sa aspekta brzine. Pokažite kako je moguće iskoristiti ćelijsko-intuitivni metod da se dobije rješenje koje je *optimalno sa aspekta brzine*.

12.10* Koristeći ideje iz prethodnog zadatka, projektirajte 4-bitni paralelni binarni sumator, koji je optimalan sa aspekta brzine (4-bitni sumator sa paralelnim formiranjem prenosa).

12.11 Rečeno je da se sklop za oduzimanje dva *n*-bitna broja $A = (a_{n-1}...a_2;a_1;a_0)_2$ i $B = (b_{n-1}...b_2;b_1;b_0)_2$ (*n*-bitni paralelni binarni oduzimač) može se napraviti na isti način kao i paralelni binarni sabirač, samo se umjesto polusumatora i punih sumatora koriste blokovi koje možemo nazvati *poluoduzimač* i *puni oduzimač*.

a) Projektirajte unutrašnju strukturu poluoduzimača i oduzimača, uz pretpostavku da su nam na raspolaganju svi tipovi logičkih kola.

b) Nacrtajte blok strukturu *n*-bitnog paralelnog binarnog oduzimača na bazi projektiranih blokova.

12.12 Vidjeli smo da kod oduzimača realiziranog na bazi sabirača i invertora, postojanje izlaznog prenosa zapravo signalizira negativan rezultat. Da li to vrijedi i kod oduzimača realiziranog u prethodnom zadatku?

12.13 Da li se paralelni binarni oduzimači (bez obzira na način izvedbe) također mogu kaskadno vezati i tako tvoriti veće oduzimače, kao što se mogu vezivati paralelni binarni sumatori?

12.14 Objasniti kako se, uz malo dodatne logike, paralelni binarni sabirači mogu prilagoditi tako da obavljaju kako operaciju sabiranja, tako i operaciju oduzimanja.

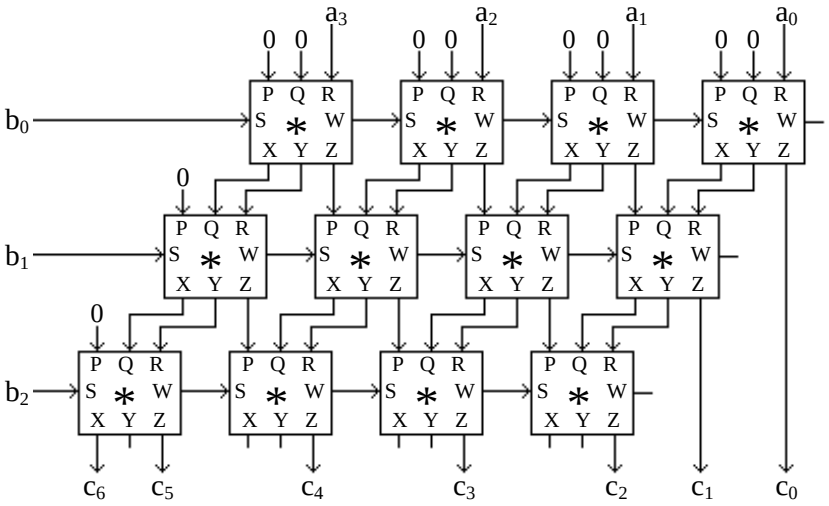
12.15 U Zadatku 11.32 razmatrano je formalno projektiranje višefunkcionalne logičke jedinice. Za slučaj kada je $c_2 = c_1 = c_0 = 0$, ova jedinica je davala izlaz $Y = 0$, bez obzira na ulaze A i B. Projektirajte intuitivnim putem modifikaciju ove višefunkcionalne jedinice koja se sastoji u sljedećem. Jedinica ima jedan dodatni ulaz nazvan p_u i jedan dodatni izlaz p_i. Za slučaj kada je $c_2 = c_1 = c_0 = 0$, izlazi Y i p_i treba da respektivno sadrže cifru rezultata i informaciju o prenosu dobijenu prilikom sabiranja $A + B + p_u$ (drugim riječima, jedinica treba da se ponaša kao potpuni sumator). Za sve ostale kombinacije c_2, c_1 i c_0, jedinica treba da radi kao i ranije, pri čemu se tada ulaz p_u ignorira, a vrijednost izlaza p_i je nedefinirana. Pokušajte dobiti što je god moguće jednostavniju realizaciju.

12.16 Prateći vrijednosti signala u svim karakterističnim tačkama sklopa, uvjerite se da množač koji smo projektirali u Primjeru 12.2 ispravno računa produkte $1101 \cdot 101$, $0110 \cdot 011$ i $1110 \cdot 110$.

12.17 Projektirajte intuitivnim putem sklop koji množi dva broja od po dva bita, pretpostavljajući da su nam na raspolaganju samo polusumatori, i AND logička kola. Uporedite ovakav dizajn sa aspekta funkcionalnosti, efikasnosti i brzine sa 2-bitnim množačem koji ste formalnim putem projektirali u Zadatku 11.25. Šta se može zaključiti?

12.18 Projektirajte intuitivnim putem sklop koji množi jedan broj od tri bita sa jednim brojem od dva bita, pretpostavljajući da su nam na raspolaganju samo polusumatori, puni sumatori i AND logička kola. Na osnovu nacrtane strukture, odredite jednačine koje opisuju rad ovog sklopa i prevedite ih oblik koji omogućava realizaciju optimalnu sa aspekta brzine.

12.19* Binarni množač projektiran u Primjeru 12.3 djeluje prilično neujednačene strukture. Međutim, binarni množač moguće je realizirati vezanjem posve identičnih ćelija, na osnovu blok strukture prikazane na sljedećoj slici:



Analizirajući ovu blok strukturu, kao i postupak množenja binarnih brojeva, otkrijte kako izgleda unutrašnjost blokova označenih sa "*", kao i značenje njihovih ulaza odnosno izlaza.

12.20* Pretpostavimo da imamo na raspolaganju samo množače koji množe dva broja od po četiri bita (koje ne možemo rastaviti na sastavne dijelove), i paralelne binarne sumatore. Kako možemo od ovih komponenti napraviti množač koji množi dva broja od po osam bita?

12.21** Djelitelje je moguće projektirati intuitivnim putem koristeći sličnu logiku koja je korištena za projektiranje množača. Projektirajte djelitelj koji dijeli dva broja, od kojih djeljenik ima 4 bita, a djelilac 3 bita (sklop treba da daje cijeli dio količnika, kao i ostatak pri dijeljenju). Pretpostavite da djelilac nikad neće biti nula.

12.22 Objasnite kako je moguće da se intuitivnim putem često dobijaju rješenja koja su efikasnija od rješenja dobijena formalnim putem, čak i u slučajevima kada je sklop dovoljno jednostavan da se može lako projektirati formalnim putem (pogledajte npr. zadatke 12.7 i 12.17).

12.23 Uporedite dva rješenja za problem iz Primjera 12.3 sa aspekta ekonomičnosti, kao i sa aspekta brzine rada.

12.24 Kombinacioni sklop poznat pod nazivom ***obračunska mreža*** (ili ***Tally mreža***) definirana je na sljedeći način. Ona ima n ulaza, koje možemo označiti sa I_1, I_2, ... I_n), i n+1 izlaz, koje možemo označiti sa C_0, C_1, ... C_n. Na izlazu C_i ($i = 0..n$) nalazi se jedinica ako i samo ako se jedinica nalazi na tačno i od ukupno n ulaza (nebitno kojih), tako da se u svakom trenutku jedinica nalazi na jednom i samo jednom od izlaza iz sklopa. Razmotrite kako biste najlakše mogli intuitivnim putem projektirati ovaj sklop (uz pretpostavku da je n zadan).

12.25 Pokažite da brzi paralelni binarni komparator i paralelni binarni komparator sa serijskom propagacijom projektirani u Primjeru 12.4 zaista zahtijevaju $n(n+1)/2 + 7n + 1$ odnosno $10n - 3$ ulaza u logička kola, pri čemu je n broj bita brojeva koji se porede.

12.26 Izračunajte koliko iznosi maksimalno kašnjenje izlaza u odnosu na ulaz za brzi paralelni binarni komparator i paralelni binarni komparator sa serijskom propagacijom ukoliko je $n = 32$, i ukoliko sva logička kola osim EXNOR logičkih kola unose kašnjenje od $\Delta t = 10\,\text{ns}$, a EXNOR logičko kolo kašnjenje od $\Delta t = 25\,\text{ns}$.

12.27 U Primjeru 12.4 projektiran je paralelni binarni komparator sa serijskom propagacijom, zasnovan na ćelijama kod kojih se informacije propagiraju slijeva nadesno, odnosno od bita najviše težine ka bitima manje težine. Projektirajte modifikaciju paralelnog binarnog komparatora sa serijskom propagacijom, koji će biti zasnovan na ćelijama kod kojih se informacije propagiraju zdesna nalijevo, odnosno od bita najmanje težine ka bitima više težine. Uporedite njegovu ekonomičnost i brzinu rada sa paralelnim binarnim komparator sa serijskom propagacijom projektiranim u Primjeru 12.4. Posebno razmotrite slučaj $n = 32$. Ima li ovakav binarni komparator ikakvu prednost u odnosu na rješenja predložena u Primjeru 12.4?

12.28 Analizirajte efikasnost i brzinu rada binarnog komparatora zasnovanog na paralelnom binarnom sabiraču, koji je projektiran na kraju Primjera 12.4, uz pretpostavku da je paralelni binarni sabirač realiziran kao u Primjeru 12.1.

12.29 Kombinacioni sklop, poznat pod nazivom ***2-bitni funkcijski generator***, posjeduje 6 ulaza, od kojih su 2 *ulaza za podatke*, nazvani A i B, i 4 tzv. *upravljačka ulaza*, nazvani f_0, f_1, f_2 i f_3. Sklop posjeduje jedan izlaz Y, koji je sa ulazima povezan jednačinom

$$Y = f_0\,\overline{A}\,\overline{B} \vee f_1\,\overline{A}B \vee f_2\,A\overline{B} \vee f_3\,AB$$

Ovaj sklop je dobio svoje ime zbog činjenice da se samo pomoću njega može realizirati svaka logička funkcija od dvije promjenljive A i B. Objasnite kako. Specijalno, pokažite kako se pomoću njega mogu realizirati sljedeće logičke funkcije:

a) $Y = \overline{A}$ b) $Y = \overline{B}$ c) $Y = AB$ d) $Y = A \vee B$

e) $Y = A \oplus B$ f) $Y = \overline{A \oplus B}$ g) $Y = A \Rightarrow B$ h) $Y = B \Rightarrow A$

12.30 Projektirajte intuitivnim putem logičku mrežu koja ima 6 ulaza, od kojih su 4 ulaza nazvana A, B, C i D namijenjena za unos podataka, a dva ulaza nazvana c_1 i c_0 predstavljaju upravljačke ulaze, kojima se određuje funkcioniranje mreže. Mreža posjeduje 2 izlaza Y i Z, na koje se prosto prosljeđuju vrijednosti sa nekih od ulaza, u zavisnosti od vrijednosti na upravljačkim ulazima, a u skladu sa sljedećom tablicom:

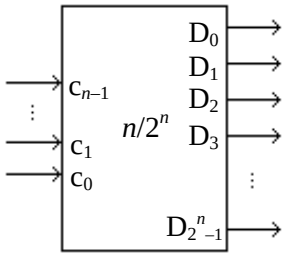
$(c_1;c_0)_2$	$(00)_2$	$(01)_2$	$(10)_2$	$(11)_2$
Y, Z	Y = A, Z = D	Y = B, Z = C	Y = C, Z = B	Y = D, Z = A

Napomena: Ovakve logičke mreže spadaju u porodicu ***multipleksera*** (odnosno ***selektora***), koji se detaljno obrađuju u Poglavlju 14.

13. Kombinacioni sklopovi bazirani na dekoderima

Iz dosadašnjih primjera vidjeli smo da projektiranje kombinacionih sklopova intuitivnom metodom nije uvijek niti jednostavno niti očigledno i da često zahtjeva dosta dosjetki. Da bi se olakšalo projektiranje složenijih kombinacionih sklopova, dizajnirani su neki *karakteristični kombinacioni sklopovi* koji *olakšavaju projektiranje drugih složenijih sklopova*, s obzirom da se mogu upotrebljavati kao kao već gotovi blokovi bez potrebe da ulazimo u njihovu strukturu. Već smo upoznali neke takve karakteristične sklopove: *polusumator*, *puni sumator*, i *n-bitni paralelni sumator*. U ovom i sljedećem poglavlju ćemo razmatrati još nekoliko karakterističnih kombinacionih sklopova koji se često javljaju kao sastavni dijelovi složenijih kombinacionih sklopova, kao i njihovu primjenu za projektiranje složenijih kombinacionih sklopova.

Jedan od najvažnijih i najviše korištenih kombinacionih sklopova je ***dekoder***. Ovaj sklop posjeduje n ulaza i 2^n izlaza. Na izlazima iz dekodera uvijek se javlja logička jedinica na samo jednom od ukupno 2^n izlaza, i to na onom izlazu čiji *redni broj* (izlazi su numerirani od 0 nadalje) *pretvoren u binarni broj odgovara binarnom broju dovedenom na ulaz* (posmatranom kao n-bitni binarni broj). Na svim ostalim izlazima javlja se logička nula. Drugim riječima, dekoder u zavisnosti od binarnog broja dovedenog na ulaz aktivira onaj i samo onaj izlaz čiji redni broj odgovara dovedenom binarnom broju (zapravo, neki dekoderi imaju i manje od 2^n izlaza, ali se takvi dekoderi mogu koristiti jedino ukoliko smo sigurni da se neki binarni brojevi na ulazu neće nikada pojaviti.). Dekoder sa n ulaza i 2^n izlaza obično se naziva dekoder "n u 2^n", ili skraćeno $n/2^n$. Dekoder se obično prikazuje blok strukturom kao na sljedećoj slici:



Da bismo razmotrili *unutrašnju strukturu dekodera*, moramo razmotriti logičke funkcije koje opisuju njegov rad. Iz opisa rada dekodera, vidimo da te funkcije imaju sljedeći oblik:

$$D_0 = \begin{cases} 1, \text{ ako je } (c_{n-1};...c_2;c_1;c_0)_2 = (0...000)_2 \\ 0, \text{ u suprotnom} \end{cases}$$

$$D_1 = \begin{cases} 1, \text{ ako je } (c_{n-1};...c_2;c_1;c_0)_2 = (0...001)_2 \\ 0, \text{ u suprotnom} \end{cases}$$

$$D_2 = \begin{cases} 1, \text{ ako je } (c_{n-1};...c_2;c_1;c_0)_2 = (0...010)_2 \\ 0, \text{ u suprotnom} \end{cases}$$

...

$$D_{2^n-1} = \begin{cases} 1, \text{ ako je } (c_{n-1};...c_2;c_1;c_0)_2 = (1...111)_2 \\ 0, \text{ u suprotnom} \end{cases}$$

Generalno, $D_i = 1$ ako i samo ako je $(c_{n-1}, ... c_2, c_1, c_0)_2 = (i)_{10}$. Veoma se lako zaključuje da funkcije D_k, $k = 0 .. 2^n-1$ nisu ništa drugo nego minterme obrazovane od promjenljivih c_i, $i = 0 .. n-1$ (odnosno $D_k = m_k(c_{n-1}, ... c_2, c_1, c_0)$), tako da možemo pisati:

$$D_0 = \bar{c}_{n-1}...\bar{c}_2\bar{c}_1\bar{c}_0 \qquad D_1 = \bar{c}_{n-1}...\bar{c}_2\bar{c}_1c_0 \qquad D_2 = \bar{c}_{n-1}...\bar{c}_2c_1\bar{c}_0$$

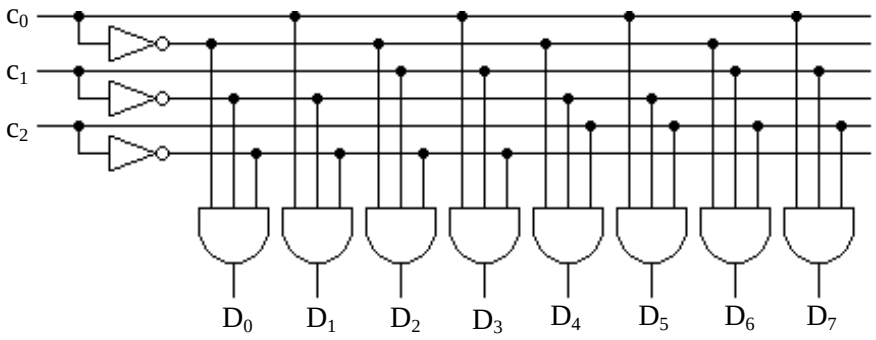
...

$$D_{2^n-1} = c_{n-1}...c_2c_1c_0$$

Na osnovu ovih logičkih funkcija sasvim je lako nacrtati unutrašnju strukturu dekodera. Posmatrajmo, na primjer, konkretan slučaj $n = 3$, odnosno dekoder 3/8. Ovaj dekoder opisan je sljedećim sistemom funkcija:

$$D_0 = \overline{c}_2\overline{c}_1\overline{c}_0 \quad D_1 = \overline{c}_2\overline{c}_1 c_0 \quad D_2 = \overline{c}_2 c_1\overline{c}_0 \quad D_3 = \overline{c}_2 c_1 c_0$$
$$D_4 = c_2\overline{c}_1\overline{c}_0 \quad D_5 = c_2\overline{c}_1 c_0 \quad D_6 = c_2 c_1\overline{c}_0 \quad D_7 = c_2 c_1 c_0$$

Odavde neposredno slijedi unutrašnja struktura dekodera 3/8:



Izlazi iz dekodera često se obilježavaju samo rednim brojevima 0, 1, 2, 3 itd. (umjesto D_0, D_1, D_2, D_3 itd.). Može se primijetiti da izlazi dekodera sa *n* ulaza zapravo predstavljaju *sve moguće minterme* koje se mogu formirati od *n* promjenljivih, tako da se dekoder ponekad naziva i ***generator mintermi***. Postoje i dekoderi čiji su svi izlazi *negirani*. Takvi dekoderi predstavljaju ***generatore makstermi***, i oni se simbolički predstavljaju uz dodatne kružiće na izlazima (kako kod NAND i NOR logičkih kola).

Na osnovu opisa rada dekodera, kao i činjenice da dekoder djeluje kao generator mintermi, slijedi da je izuzetno lako uz pomoć dekodera realizirati bilo koju logičku funkciju ili sistem logičkih funkcija. Dovoljno je predstaviti tražene funkcije u savršenoj disjunktivnoj normalnoj formi i odgovarajuće minterme na izlazu iz dekodera povezati preko "ili" kola (za slučaj dekodera sa invertiranim izlazima, umjesto SDNF potrebno je formirati SKNF, a maksterme na izlazu iz dekodera povezati pomoću "i" kola). U slučaju kada je funkcija data tabelarno, nije čak ni potrebno pisati savršenu disjunktivnu (odnosno konjunktivnu) normalnu formu, nego je odgovarajuće povezivanje moguće izvršiti neposredno iz tablice.

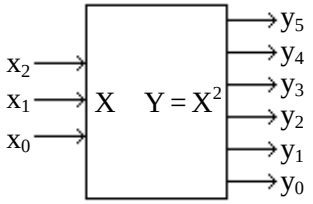
Primjer 13.1:

- Pomoću dekodera i OR logičkih kola realizirati sklop koji računa kvadrat ulaznog trobitnog broja.

Rad traženog sklopa moguće je opisati sljedećom tablicom:

X	0	1	2	3	4	5	6	7
X^2	0	1	4	9	16	25	36	49
X^2 (binarno)	0	1	100	1001	10000	11001	100100	110001

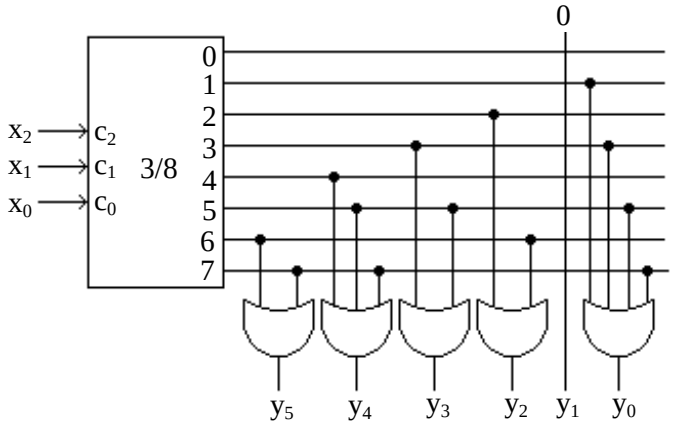
Odavde zaključujemo da traženi sklop treba da ima 6 izlaza, jer se najveći broj koji se javlja na izlazu može predstaviti pomoću 6 bita:



Na osnovu prethodne tablice lako formiramo tablicu istine traženog sklopa:

x_2	x_1	x_0	y_5	y_4	y_3	y_2	y_1	y_0
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1
0	1	0	0	0	0	1	0	0
0	1	1	0	0	1	0	0	1
1	0	0	0	1	0	0	0	0
1	0	1	0	1	1	0	0	1
1	1	0	1	0	0	1	0	0
1	1	1	1	1	0	0	0	1

Konačno, na osnovu tablice, neposredno slijedi realizacija pomoću dekodera i OR logičkih kola:



Dekoder se ponaša kao gotovo idealno sredstvo za realizaciju logičkih funkcija, koje naizgled čini nepotrebnim sve tehnike projektiranja koje smo do sada razmatrali. Međutim, veliki nedostatak dekodera je njihova glomaznost. Naime, dekoder sa n ulaza ima 2^n izlaza. Na primjer, za $n = 10$ imamo 1024 izlaza. Potpuno je neprihvatljivo imati sklop sa 1024 izlaza, i njega koristiti dalje za realizaciju funkcija koje se traže. Postoji ipak jedan važan slučaj u *integriranoj tehnici* kada ovako veliki broj izlaza ne smeta, a to je situacija u kojoj su svi ti izlazi izvedeni samo *interno*, tj. unutar čipa, bez njihove vidljivosti izvan samog čipa. Tako nešto se koristi npr. u ROM memorijama, o čemu ćemo govoriti u nastavku teksta. Prije nego što kažemo nešto o tome, ukažimo na još jednu bitnu činjenicu. Dekoder prema strukturi koju smo do sada razmotrili traži 2^n AND kola sa po n ulaza, odnosno ukupno $n \cdot 2^n$ ulaza. Npr. za $n = 16$ ovaj broj iznosi $16 \cdot 2^{16} = 1048576$. Mada ovoliki broj ulaza nije nedostižan sa današnjom integriranom tehnikom, treba ipak razmotriti postoje li bolja rješenja. Velika ušteda može se postići primjenom strategije ***"podijeli i osvoji"*** (engl. Divide and Conquer ili, u našem žargonu, "zavadi pa vladaj"). Ova strategija predstavlja jednu od najmoćnijih strategija u računarskim naukama uopće (kako u dizajnu hardvera, tako i u programiranju). Osnovna ideja sastoji se u tome da problem podijelimo na više manjih podproblema, nakon čega se rješenje ovih podproblema kombinira u rješenje početnog problema. Pomatrajmo, na primjer, dekoder 4/16. Njegove izlazne funkcije su sljedeće:

$$D_0 = \bar{c}_3\bar{c}_2\bar{c}_1\bar{c}_0 \quad D_1 = \bar{c}_3\bar{c}_2\bar{c}_1c_0 \quad D_2 = \bar{c}_3\bar{c}_2c_1\bar{c}_0 \quad D_3 = \bar{c}_3\bar{c}_2c_1c_0$$
$$D_4 = \bar{c}_3c_2\bar{c}_1\bar{c}_0 \quad D_5 = \bar{c}_3c_2\bar{c}_1c_0 \quad D_6 = \bar{c}_3c_2c_1\bar{c}_0 \quad D_7 = \bar{c}_3c_2c_1c_0$$
$$D_8 = c_3\bar{c}_2\bar{c}_1\bar{c}_0 \quad D_9 = c_3\bar{c}_2\bar{c}_1c_0 \quad D_{10} = c_3\bar{c}_2c_1\bar{c}_0 \quad D_{11} = c_3\bar{c}_2c_1c_0$$
$$D_{12} = c_3c_2\bar{c}_1\bar{c}_0 \quad D_{13} = c_3c_2\bar{c}_1c_0 \quad D_{14} = c_3c_2c_1\bar{c}_0 \quad D_{15} = c_3c_2c_1c_0$$

Ukoliko bismo ovaj dekoder realizirali na klasični, već opisani način, trebalo bi nam 16 4-ulaznih AND kola, odnosno 64 ulaza. Posmatrajmo sada umjesto izvornog dekodera dva dekodera 2/4 od kojih na jedan dovodimo samo ulaze c_0 i c_1, a na drugi samo ulaze c_2 i c_3. Ovim dekoderima odgovaraju redom sljedeće izlazne funkcije:

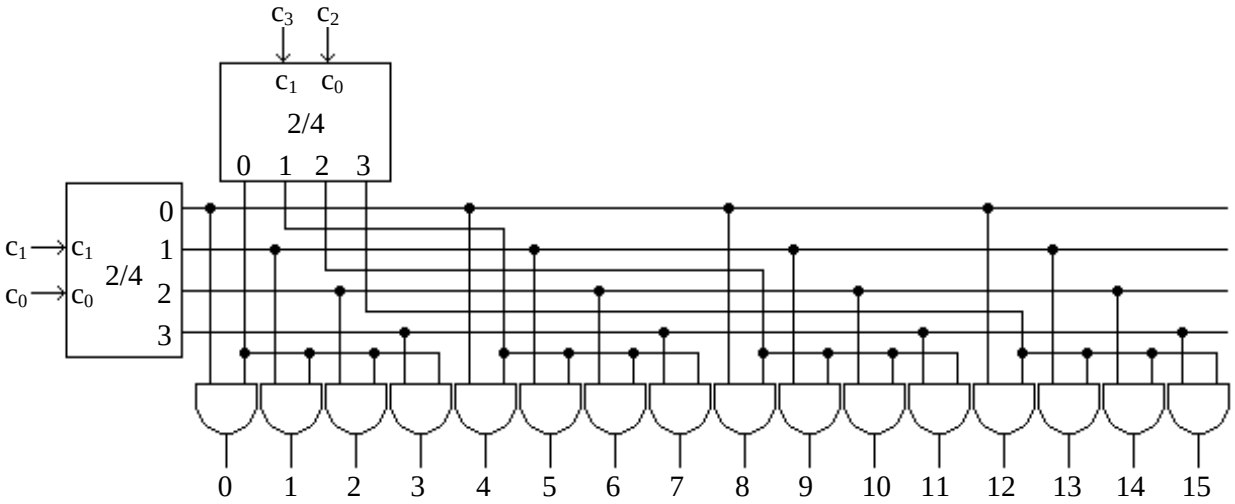
$D_0' = \bar{c}_1\bar{c}_0$ $D_1' = \bar{c}_1 c_0$ $D_2' = c_1\bar{c}_0$ $D_3' = c_1 c_0$

$D_0'' = \bar{c}_1\bar{c}_0$ $D_1'' = \bar{c}_1 c_0$ $D_2'' = c_1\bar{c}_0$ $D_3'' = c_1 c_0$

Uporedimo li izlaze ovih dekodera sa izlazima iz traženog dekodera 4/16, vidimo da njegove izlaze možemo izvesti iz izlaza iz pomenuta dva dekodera 2/4 na sljedeći način:

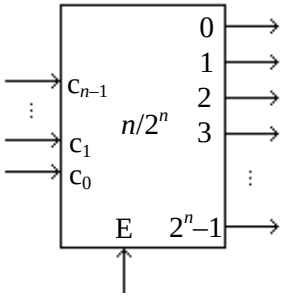
$D_0 = D_0' D_0''$ $D_1 = D_1' D_0''$ $D_2 = D_2' D_0''$ $D_3 = D_3' D_0''$
$D_4 = D_0' D_1''$ $D_5 = D_1' D_1''$ $D_6 = D_2' D_1''$ $D_7 = D_3' D_1''$
$D_8 = D_0' D_2''$ $D_9 = D_1' D_2''$ $D_{10} = D_2' D_2''$ $D_{11} = D_3' D_2''$
$D_{12} = D_0' D_3''$ $D_{13} = D_1' D_3''$ $D_{14} = D_2' D_3''$ $D_{15} = D_3' D_3''$

Slijedi da traženi dekoder 4/16 možemo formirati i iz dva dekodera 2/4 i 16 dvoulaznih AND kola prema sljedećoj slici:

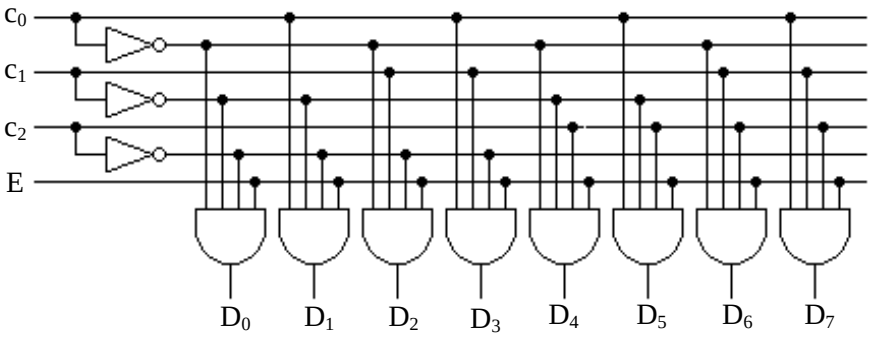


Može se postaviti pitanje šta smo ovim dobili. Dva dekodera 2/4 sadrže četiri dvoulazna AND kola, što čini ukupno 16 ulaza. Zajedno sa 32 ulaza koji potiču od 16 dvoulaznih AND kola imamo ukupno 48 ulaza, što je za 16 ulaza manje nego u slučaju da smo traženi dekoder 4/16 realizirali neposredno. Uštede su još veće za slučaj većih dekodera. Razmotrimo npr. šta nam donosi ova strategija za slučaj $n = 16$. Već smo rekli da klasična struktura ovakvog dekodera zahtijeva 1048576 ulaza. Ukoliko bismo razbili dekoder 16/65536 na dva dekodera 8/256, trebalo bi nam dva puta po 256 osmoulaznih AND kola, kao i 65536 dvoulaznih AND kola za kombiniranje izlaza iz ova dva dekodera u tražene izlaze, što je ukupno 135168 ulaza. Ovim smo uštedili čak 913408 ulaza, odnosno ostvarili smo gotovo osmostruko smanjenje broja ulaza! Možemo dobiti još veću uštedu ukoliko svaki od dekodera 8/256 realiziramo preko dva dekodera 4/16, a svaki od dekodera 4/16 preko dekodera 2/4. Nije teško izračunati da tada ukupan broj ulaza iznosi 132288, što je ušteda od još dodatnih 2880 ulaza.

Dekoderi koji se proizvode kao samostalne komponente (tj. dekoderi koji nisu sastavni dio drugih složenijih sklopova) gotovo uvijek se prave sa jednim dopunskim ulazom, koji se označava sa E, a naziva ***omogućavajući ulaz*** (engl. Enable):



Kada je E = 1, dekoder radi normalno, dok su za E = 0 svi izlazi iz dekodera na nuli, bez obzira na stanje ostalih ulaza. Ovakav dekoder je veoma lako napraviti. Naime, dovoljno je na sva izlazna AND kola standardnog dekodera dodati jedan ulaz, i na njega dovesti ulaz E. Na primjer, na sljedećoj slici je prikazana unutrašnja struktura dekodera 3/8 sa omogućavajućim ulazom:

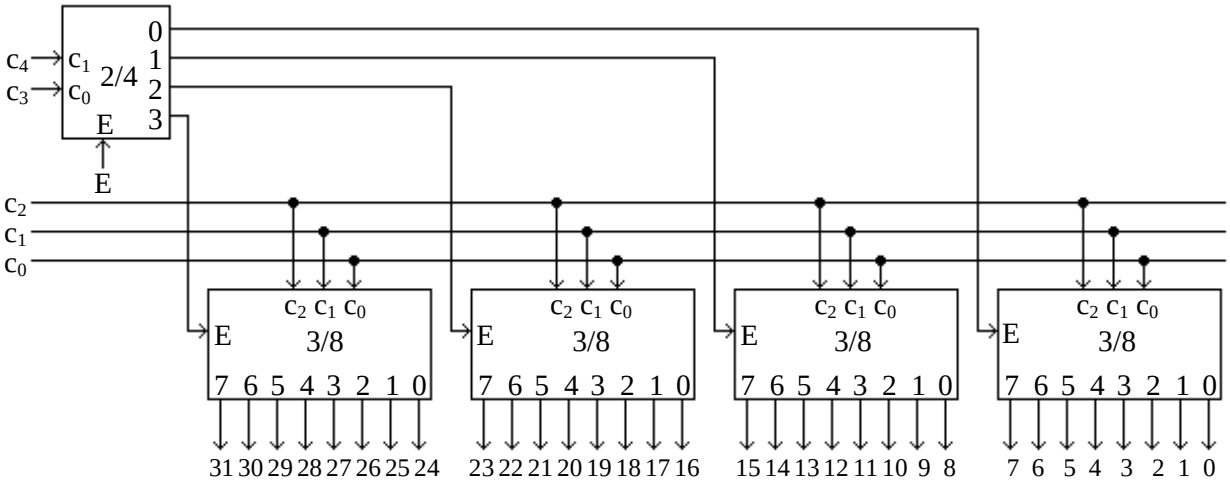


Glavni razlog za uvođenje dekodera sa omogućavajućim ulazom je mogućnost gradnje većih dekodera od više manjih, proširivanjem izlaznih mintermi putem dovođenja odgovarajućih signala na E ulaze dekodera. Tako se dekoder formata $(m+n)/2^{m+n}$ može sagraditi pomoću jednog dekodera formata $m/2^m$ i 2^m dekodera formata $n/2^n$. Ovo je posebno značajno s obzirom na činjenicu da se dekoderi većeg formata od 4/16 veoma rijetko proizvode kao samostalne komponente.

Primjer 13.2:

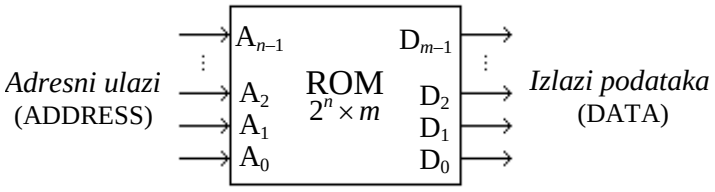
- Realizirati dekoder 5/32 od četiri dekodera 3/8 i jednog dekodera 2/4. Dekoderi posjeduju omogućavajuće ulaze.

Ako na dekodere 3/8 dovedemo ulaze c_2, c_1 i c_0, dobićemo sve minterme formirane od ovih promjenljivih. Da bismo formirali sve minterme od promjenljivih c_4, c_3, c_2, c_1 i c_0 koje nam trebaju za dekoder tipa 5/32, moramo ove minterme proširiti svim kombinacijama promjenljivih c_4, c_3 i njihovih negacija, tj. svim mintermama od promjenljivih c_4 i c_3. Ove minterme možemo generirati pomoću dekodera 2/4 na čije ulaze dovodimo promjenljive c_4 i c_3. Konačno, proširenje izvodimo preko E ulaza, jer se E ulazi praktično množe sa svim mintermama na izlazu iz dekodera. Slijedi da tražena realizacija ima oblik kao na sljedećoj slici:



Primijetimo da se ovakvom izvedbom većih dekodera također postiže ušteda u broju ulaza u odnosu na klasičnu realizaciju. Obična izvedba dekodera 5/32 sa omogućavajućim ulazom zahtijevala bi $6 \cdot 32 = 192$ ulaza u AND logičkih kola. Sa ovakvom realizacijom, imamo 4 dekodera 3/8 sa E ulazom od kojih svaki zahtijeva $4 \cdot 8 = 32$ ulaza u AND logička kola, i jedan dekoder 2/4 sa E ulazom koji zahtijeva $3 \cdot 4 = 12$ ulaza u AND logička kola, što daje ukupno 140 ulaza. Ušteda od 52 ulaza ni u kom slučaju nije zanemarljiva!

Pređimo sada na analizu jednog od najvažnijih kombinacionih sklopova u računarskim sistemima i u digitalnoj tehnici uopće, a to je ***ROM memorija***. Ovaj sklop tijesno je povezan sa dekoderom. Sa pojmom ROM memorije neformalno smo se upoznali već u uvodnim poglavljima, a sada je vrijeme da se pojam ROM memorije definira i *formalno*. U svim memorijama, pa tako i u ROM memoriji, podaci su organizirani u male jedinice, koje logički gledano podsjećaju na pregrade na nekoj polici. Te jedinice nazivaju se ***adrese***. Svaki podatak zauzima po jednu adresu. ROM memorija se definira kao digitalni sklop sa *n* ulaza nazvanih ***adresni ulazi*** (engl. Address Input) i *m* izlaza nazvanih ***izlazi podataka*** (engl. Data Output), kojem na ulazu saopštavamo *adresu nekog podatka* (naravno, zapisanu kao binarni broj) a na izlazu dobijamo *podatak smješten na toj adresi*, odnosno *sadržaj te adrese*. Numeracija adresa počinje od *nule*. U praktičnim izvedbama ROM memorija *m* je najčešće 4, 8 ili 16, dok je *n* broj koji ovisi od kapaciteta memorije, a određuje broj memorijskih adresa. Jasno je da adresa može biti ukupno 2^n, tako da su adrese numerirane od 0 do 2^n-1.



Pošto se u ROM memoriju ne moze ništa upisivati, slijedi da uvijek *za istu kombinaciju adresnih ulaza dobijamo iste vrijednosti na izlazima* (koji ovise o tome šta je upisano u memoriju u fazi proizvodnje). Međutim, u skladu sa definicijom kombinacionog sklopa, očigledno vidimo da ROM memorija nije ništa drugo nego jedan *kombinacioni sklop*. Zapravo, vrijedi i obrnuto: svaki kombinacioni sklop se, na izvjestan način, može posmatrati i kao ROM memorija (koja "pamti" njegovu tablicu istine, a čije kombinacije ulaznih promjenljivih predstavljaju "adrese").

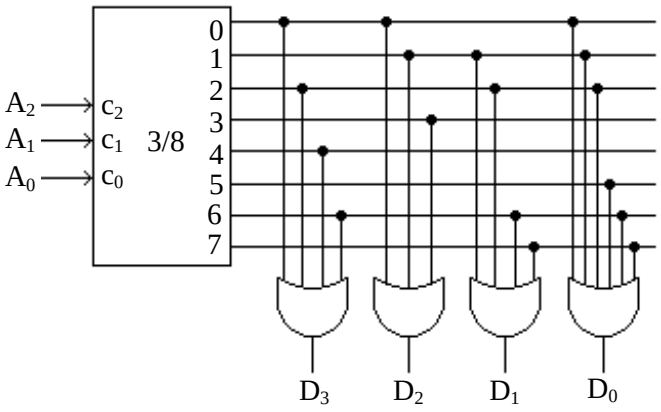
Primjer 13.3:

- Neka ROM memorija ima kapacitet od 8 adresa numeriranih redom od 0 do 7. Na te adrese upisani su redom sljedeći podaci: 13, 7, 11, 4, 0, 9, 3 i 11. Projektirati ovu ROM memoriju.

Tražena ROM memorija je kombinacioni sklop sa tri ulaza A_2, A_1, A_0 (jer se adrese u opsegu od 0 do 7 mogu predstaviti sa 3 bita) i četiri izlaza D_3, D_2, D_1 i D_0 (jer se upisani podaci mogu smjestiti u 4 bita). Rad ovog sklopa možemo prikazati sljedećom tablicom:

ADRESA	A_2	A_1	A_0	PODATAK	D_3	D_2	D_1	D_0
0	0	0	0	13	1	1	0	1
1	0	0	1	7	0	1	1	1
2	0	1	0	11	1	0	1	1
3	0	1	1	4	0	1	0	0
4	1	0	0	0	0	0	0	0
5	1	0	1	9	1	0	0	1
6	1	1	0	3	0	0	1	1
7	1	1	1	11	1	0	1	1

Ovaj kombinacioni sklop možemo projektirati na bilo koji dosada opisani način, ali zbog razloga koji će uskoro biti jasan, ROM memorije se projektiraju *isključivo pomoću dekodera* (čak se često pod ROM memorijama smatraju oni i samo oni kombinacioni sklopovi koji su projektirani samo pomoću dekodera i pratećih OR logičkih kola). Možemo odmah da nacrtamo shemu tražene memorije, jer se sve što je neophodno za projektiranje vidi iz gore napisane tablice:



Rekli smo da je kod ROM memorija najčešće $m = 8$ zbog toga što se podaci obično organiziraju tako da na jednoj adresi bude smješten 1 bajt. Posto adresa ima 2^n, a svaka adresa može sadržavati po m bita, slijedi da je kapacitet ROM memorije $m \cdot 2^m$ bita, odnosno tačno 2^n bajta ako je $m = 8$. Za druga dva tipična slučaja $m = 4$ i $m = 16$ dobijamo da su kapaciteti memorije u bajtima redom 2^{n-1} i 2^{n+1} bajta. Dakle, u svim tipičnim slučajevima, kapaciteti ROM memorija izraženi u bajtima predstavljaju stepen broja 2, što smo već nagovijestili u uvodnim poglavljima. Vidjećemo kasnije da ista stvar vrijedi i za RAM memorije.

Primjer 13.4:

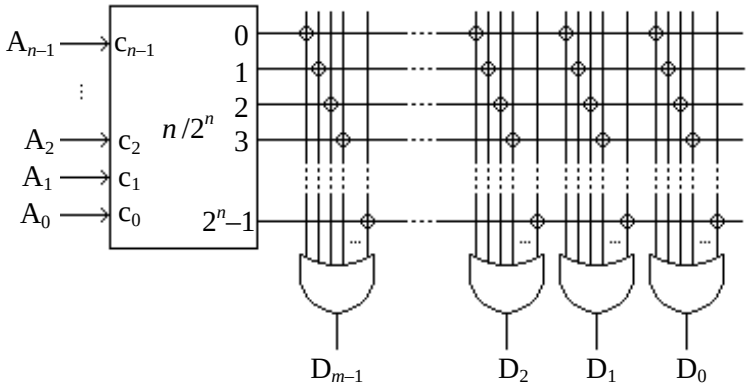
- Odrediti koliko adresnih ulaza mora imati ROM memorija kapaciteta 32 KB organizirana tako da svaka adresa sadrži 16-bitni podatak (2 bajta).

32 KB izraženo u bajtima iznosi $32 \cdot 1024$, odnosno 32768 bajta. Kako na svaku adresu staje podatak od 2 bajta, slijedi da tražena memorija posjeduje $32768 / 2 = 16384$ adrese. Stoga imamo:

$$2^n = 16384 \Rightarrow n = \log_2 16384 = 14$$

Dakle, potrebno je 14 adresnih ulaza.

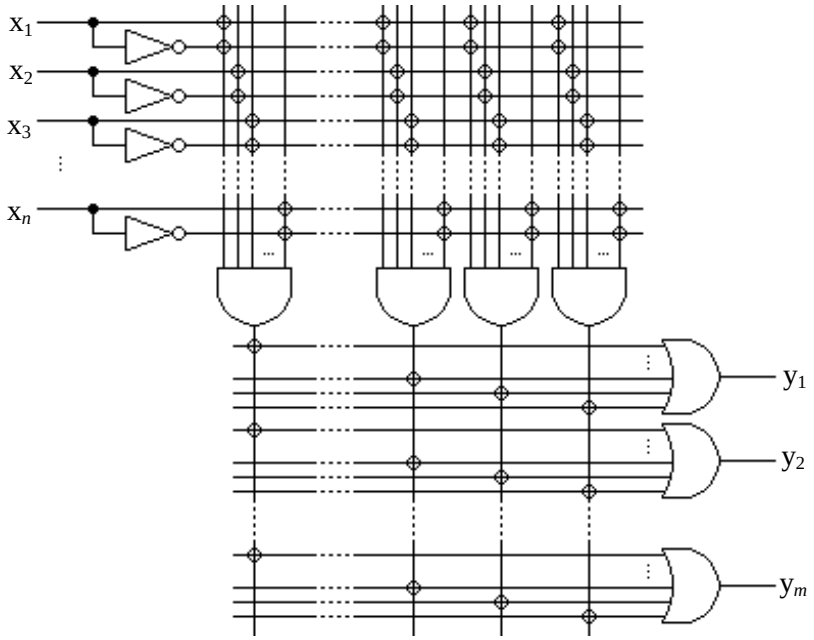
Iz svega dosada rečenog vidimo da su podaci koji su upisani u ROM memoriju zapravo određeni *rasporedom čvorova* nakon izlaza iz dekodera. Broj izlaza iz dekodera, koji je često veoma veliki (npr. za $n = 16$ broj izlaza iznosi $2^{16} = 65536$) nije nepremostiv problem, s obzirom da svi izlazi ostaju interno unutar čipa, tj. ne izlaze izvan sklopa. Prema tome, sadržaj memorije je moguće odrediti samo posmatranjem rasporeda čvorova. Upravo ova činjenica dovodi do ideje za građu ***programabilnih ROM memorija***, tj. PROM, EPROM i EEPROM memorija, koje su danas praktično u potpunosti istisnule klasične ROM memorije. Osnovna ideja je omogućiti korisniku da *sam formira raspored čvorova*. Na sljedećoj slici je prikazana principijelna struktura PROM memorije sa 2^n adresa (dakle, sa n adresnih ulaza) i m-bitnim podacima (dakle sa m izlaza podataka):



Na ovoj slici kružići predstavljaju *topljive čvorove* koji u normalnom radu neće pregoriti, ali hoće ukoliko kroz njih ako pustimo *jaču struju* (nekoliko puta veću nego pri normalnom radu). Na početku, ovaj sklop se ponaša kao ROM memorija u koju su upisane *sve jedinice*. Pretpostavimo npr. da je $m = 4$. Sada, ako na primjer želimo da na adresu 2 upišemo podatak 0010, tri čvora treba ukloniti. To možemo uraditi tako što ćemo na adresne ulaze dovesti broj 2 (binarno kodiran), a kroz izlaze D_3, D_2 i D_0 pustiti jaču struju koja će proteći upravo kroz neželjene čvorove i uništiti ih.

EPROM i EEPROM memorije se zasnivaju na istoj shemi kao i PROM memorije, a razlikuju se *samo po građi čvorova koji su predstavljeni kružićima*. Naime, kod njih kružići ne predstavljaju topljive čvorove, već spojeve građeni od specijalnih elemenata (tzv. FAMOS i NMOS tranzistora), koji su u normalnim uvjetima *nevodljivi* (tj. ne predstavljaju spoj). Propuštanjem jače struje oni postaju *vodljivi*, tj. počinju se ponašati kao obični čvorovi. Oni u takvom vodljivom stanju ostaju i *nakon prestanka djelovanja struje*, sve dok se ne izlože *jakoj svjetlosti* (za slučaj EPROM-a) ili *jakoj struji suprotnog polariteta* (za slučaj EEPROM-a). Dakle, EPROM i EEPROM se na početku ponašaju kao ROM memorija u koju su upisane *sve nule*, a prilikom upisa čvorove koji odgovaraju jedinicama treba učiniti vodljivim na sličan način kao što se vrši pregaranje topljivih čvorova kod PROM memorija.

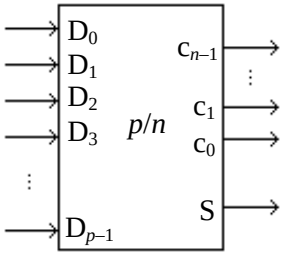
Programabilne ROM memorije (PROM, EPROM, EEPROM) djeluju gotovo idealne za realizaciju logičkih funkcija, jer se sa jednom ROM memorijom kapaciteta $2^n \times m$ bita može realizirati proizvoljan skup od m logičkih funkcija od po n promjenljivih prostim "programiranjem" rasporeda čvorova na izlazu iz dekodera, i to neposredno na osnovu tablice istine, bez potrebe za mučnim postupcima minimizacije. Međutim, na ovaj način se isplati realizirati samo zaista složene logičke funkcije, jer su realizacije na bazi ROM memorija dosta *neracionalne* (tj. skupe), ne samo zbog velike složenosti dekodera koji se nalaze u njima, nego i zbog činjenice da izlazna "ili" kola u programabilnim ROM memorijama imaju ogroman broj ulaza (2^n). Posmatrajmo, na primjer, neku logičku funkciju od deset promjenljivih, koja se nakon minimizacije može prikazati u MDNF obliku sa svega petnaestak prostih implikanti. Njena realizacija preko programabilne ROM memorije tražila bi ROM sa 10 ulaza i jednim izlazom, u kojem se dalje nalazi dekoder 10/1024 (koji uz najekonomičniju realizaciju traži 2240 ulaza u AND kola) i 1024-ulazno OR kolo, što je očigledno rasipanje resursa za funkciju koja bi se očigledno mogla realizirati relativno jednostavno i pomoću običnih AND, OR i NOT logičkih kola. Očigledno, ROM memorije potpuno ignoriraju bilo kakvu dobit koja se postiže minimizacijom, jer bi iste resurse zahtijevala i funkcija koja *nije minimizirana*. Da bi se mogle iskoristiti prednosti koje pruža minimizacija, proizvode se i integrirane komponente nazvane ***programabilne logičke matrice*** ili ***PLA*** (Programable Logic Array) ***komponente***. Jedna PLA komponenta se, kao i ROM memorije, proizvodi u jednom čipu, a njena struktura je prikazana na sljedećoj slici:



Ova slika prikazuje PLA komponentu namijenjenu za realizaciju skupa od m funkcija od po n promjenljivih. Ulazi su obilježeni sa x_i, $i = 1..n$, a izlazi sa y_j, $j = 1..m$. Sva AND kola imaju po n ulaza (u nekim izvedbama i manje), dok je broj AND kola neki unaprijed određeni broj p ovisan od tipa PLA komponente, pri čemu je tipično $p << 2^n$ (npr. za $n = 10$ može biti $p = 20 \div 100$). S druge strane, svako OR kolo ima po p ulaza (u nekim izvedbama i manje), dok je njihov broj m. Da bismo iskoristili PLA komponentu, funkciju moramo predstaviti u disjunktivnoj normalnoj formi (po mogućnosti minimalnoj), nakon čega nam AND kola služe za *formiranje pojedinih implikanti* koje ulaze u traženu funkciju, dok OR kola služe za *objedinjavanje formiranih implikanti*. Oba ova postupka provode se, kao kod programabilnih ROM-ova, "programiranjem" rasporeda čvorova, koji su na slici označeni kružićima. Kružići mogu biti topljivi čvorovi kao kod PROM-a, ili čvorovi promjenljive vodljivosti, kao kod EPROM-a i EEPROM-a, tako da imamo kako neizbrisive, tako i izbrisive PLA komponente. Primijetimo da su PLA komponente *mnogo jednostavnije građe* nego programabilne ROM memorije, jer ne sadrže glomazne dekodere sa ogromnim brojem izlaza, niti OR kola sa ogromnim brojem ulaza.

Očigledno, maksimalan broj implikanti koje se mogu formirati je p, tako da sa PLA komponentama nije moguće realizirati funkcije koje posjeduju više od p prostih implikanti. Također, ukoliko neka PLA komponenta posjeduje AND kola sa $q < n$ ulaza, tada možemo realizirati samo one funkcije kod kojih je dužina svih implikanti manja od q. Da bismo smanjili broj implikanti i skratili njihovu dužinu, očigledno treba primijeniti postupke minimizacije, tako da su PLA komponente "vratile u modu" metode za minimizaciju logičkih funkcija, koje su jedno vrijeme bile "potisnute u stranu" pojavom programabilnih ROM komponenti. Suštinska prednost PLA komponenti nad programabilnim ROM memorijama je u činjenici da se često ista funkcija može realizirati sa PLA komponentom mnogo jednostavnije građe (i samim tim manje cijene) nego što je odgovarajuća ROM memorija. Na primjer, već pomenuta funkcija od 10 promjenljivih čija MDNF posjeduje 15 prostih implikanti, može se realizirati PLA komponentom kod koje je $n = 10$, $p = 15$ i $m = 1$. Takva PLA komponenta posjeduje 15 10-ulaznih AND kola i jedno 15-ulazno OR kolo, što je ukupno 165 ulaza, odnosno skoro 20 puta ekonomičnije od odgovarajuće programabilne ROM memorije! Danas su PLA komponente toliko jeftine i fleksibilne da se iole složenije logičke funkcije danas gotovo isključivo realiziraju uz njihovu pomoć.

Sljedeći karakterističan kombinacioni sklop koji se često susreće u računarskoj tehnici je ***koder***. Ovaj sklop je, na izvjestan način, inverzan dekoderu, a posjeduje p ulaza i n izlaza, pri čemu je $2^{n-1} < p \le 2^n$ (zapravo, najčešće je $p = 2^n$, ali ne mora biti). Ukoliko se na *samo jednom od ulaza* kodera nalazi logička jedinica (ulazi su numerirani od 0 nadalje), na izlazima se pojavljuje *binarno zapisan redni broj ulaza koji je na logičkoj jedinici*. Drugim riječima, koder na izlazu daje binarno kodiran redni broj ulaza koji je na logičkoj jedinici. Ukoliko je više ulaza istovremeno na logičkoj jedinici, ili ukoliko niti jedan od ulaza nije na logičkoj jedinici, izlaz iz kodera *nije definiran*. Koderi obično posjeduju i jedan dodatni izlaz S, na kojem je logička jedinica ako i samo ako je barem jedan od ulaza na logičkoj jedinici. Koder sa p ulaza i n izlaza obično se naziva koder "p u n", ili skraćeno p/n (kod kodera je uvijek $p > n$ tako da ne postoji opasnost da se po oznaci koder pomiješa sa dekoderom), a obično se prikazuje blok strukturom kao na sljedećoj slici:



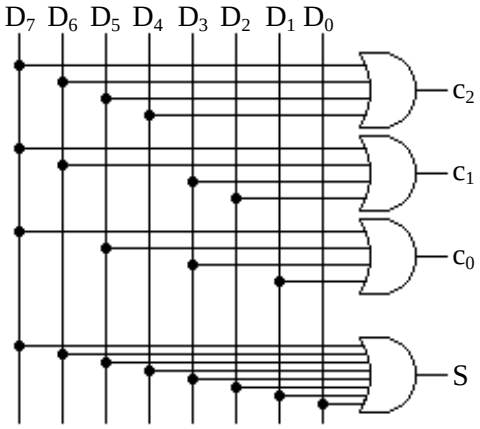
Slično kao i kod dekodera, ulazi kodera se često obilježavaju samo rednim brojevima od 0 do p–1.

Koderi su tipično sklopovi sa velikim brojem ulaza, koje stoga nije lako projektirati formalnom metodom. Ipak, zbog velikog broja *zabranjenih ulaznih kombinacija*, kodere je sasvim lako projektirati *intuitivnim postupkom*, što ćemo ilustrirati kroz konkretan primjer.

Primjer 13.5:

- Projektirati strukturu kodera 8/3.

Traženi koder ima 8 ulaza: D_0, D_1, D_2, D_3, D_4, D_5, D_6 i D_7, kojima redom odgovaraju trobitni binarni brojevi 000, 001, 010, 011, 100, 101, 110 i 111. Vidimo da se bit c_0 treba aktivirati ukoliko su aktivni ulazi D_1, D_3, D_5 ili D_7, bit c_1 se treba aktivirati ukoliko su aktivni ulazi D_2, D_3, D_6 ili D_7, dok se bit c_2 treba aktivirati ukoliko su aktivni ulazi D_4, D_5, D_6 ili D_7. Odavde direktno slijedi $c_0 = D_7 \vee D_5 \vee D_3 \vee D_1$, $c_1 = D_7 \vee D_6 \vee D_3 \vee D_2$ i $c_2 = D_7 \vee D_6 \vee D_5 \vee D_4$. Konačno, izlaz S treba da bude na jedinici ukoliko je barem jedan od ulaza na jedinici, odnosno $S = D_7 \vee D_6 \vee D_5 \vee D_4 \vee D_3 \vee D_2 \vee D_1 \vee D_0$. Odavde neposredno slijedi struktura traženog sklopa:



Kao što je već rečeno, izlazi iz kodera nisu definirani ukoliko je više od jednog ulaza na logičkoj jedinici. Posve je jasno da će svaki koder dati nekakav izlaz čak i u takvoj situaciji. Na primjer, ukoliko u koderu iz prethodnog primjera dovedemo na jedinicu ulaze D_4 i D_5 imaćemo $c_0 = 1$, $c_1 = 0$ i $c_2 = 1$, tj.izlaz će biti kao da smo doveli jedinicu samo na ulaz D_5. Međutim, ukoliko dovedemo na jedinicu ulaze D_2 i D_4, imaćemo $c_0 = 0$, $c_1 = 1$ i $c_2 = 1$, tj.izlaz će biti kao da smo doveli jedinicu na ulaz D_6.Vidimo da u situacijama kada je više od jednog ulaza na logičkoj jedinici nikakve korisne informacije o stanju ulaza ne možemo izvući posmatrajući vrijednosti na izlazima. S druge strane, postoje koderi kod kojih je vrijednost na izlazu posve određena i u slučaju kada je više ulaza istovremeno na logičkoj jedinici. Najčešće se u takvim situacijama daje prioritet ulazu sa *najvećim rednim brojem*. Takvi koderi nazivaju se ***koderi sa prioritetom*** (***prioritetni koderi***).

Primjer 13.6:

- Projektirati koder 8/3 sa prioritetom.

Jasno je, da zbog velikog broja ulaza (8) rad ovog sklopa nije pogodno opisivati klasičnom tablicom istine, jer bi takva tablica morala imati $2^8 = 256$ redova. Zbog toga ćemo rad sklopa opisati nekom vrstom *kompresovane tablice istine*, u kojoj oznaka "x" označava "bilo šta":

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	c_2	c_1	c_0	S
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	1	x	0	0	1	1
0	0	0	0	0	1	x	x	0	1	0	1
0	0	0	0	1	x	x	x	0	1	1	1
0	0	0	1	x	x	x	x	1	0	0	1
0	0	1	x	x	x	x	x	1	0	1	1
0	1	x	x	x	x	x	x	1	1	0	1
1	x	x	x	x	x	x	x	1	1	1	1

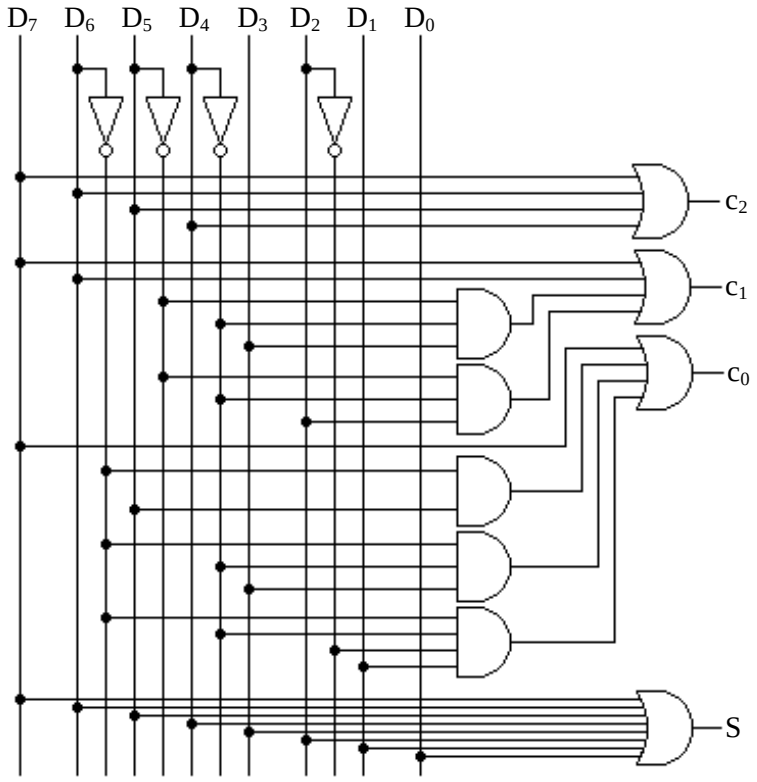
Jasno je da pri ovakvom obilježavanju dolazi u obzir samo intuitivno projektiranje. Posmatrajući tablicu, možemo primijetiti da se bit c_2 aktivira kad god su aktivni ulazi D_4, D_5, D_6 ili D_7, i samo tada. Bit c_1 se također aktivira kada su aktivni ulazi D_6 ili D_7. Međutim, c_1 se aktivira i kada su aktivni ulazi D_2 ili D_3 (bez ulaza D_6 ili D_7), ali samo pod uvjetom da pri tome nisu aktivirani ulazi D_4 i D_5. Za bit c_0 možemo reći da se aktivira ukoliko je aktivan ulaz D_7 ili ako je aktivan ulaz D_5 a nije ulaz D_6, ili ako je aktivan ulaz D_3 a nisu aktivni ulazi D_4 i D_6, ili ukoliko je aktivan ulaz D_1, a nisu aktivni ulazi D_2, D_4 niti D_6. Na osnovu ovih rezonovanja direktno slijedi:

$$c_2 = D_7 \vee D_6 \vee D_5 \vee D_4$$

$$c_1 = D_7 \vee D_6 \vee \overline{D}_5\overline{D}_4D_3 \vee \overline{D}_5\overline{D}_4D_2$$

$$c_0 = D_7 \vee \overline{D}_6D_5 \vee \overline{D}_6\overline{D}_4D_3 \vee \overline{D}_6\overline{D}_4\overline{D}_2D_1$$

Sada nije teško nacrtati shemu traženog kodera sa prioritetom:



Očigledno je koder sa prioritetom mnogo složenije strukture nego obični koder, i za njegovo projektiranje intuitivnim putem potrebno je *mnogo više domišljatosti* nego u slučaju običnog kodera. Kodere sa prioritetom sa manjim brojem ulaza (do 6) lakše je projektirati formalnim putem. Čitateljima i čitateljkama se ostavlja za vježbu da pokušaju projektirati prioritetni koder 4/2 formalnim postupkom.

Koderi *ne posjeduju univerzalnost* koju posjeduju dekoderi u smislu da oni mogu poslužiti za realizaciju složenijih logičkih funkcija. Naime, koderi se u računarskim sistemima koriste gotovo isključivo *namjenski*. Oni se najčešće koriste prilikom povezivanja vanjskih (perifernih) uređaja u računarski sistem, kada je potrebno obezbijediti da računarski sistem zna koji je uređaj zatražio intervenciju. O ovome ćemo detaljnije govoriti u kasnijim poglavljima. Zahtjevi za intervencijom dovode se na ulaze kodera, dok se na izlazu kodera dobija binarno kodiran redni broj uređaja koji je zatražio intervenciju. Ovaj broj je pogodan za dalju obradu u računarskom sistemu. Pri tome se prioritetni koderi koriste u situaciji kada je određenom uređaju neophodno dati prioritet u slučaju da više od jednog uređaja istovremeno zatraži intervenciju.

(?) Pitanja i zadaci

13.1 Objasnite šta je dekoder, i u čemu se sastoji njegova upotrebna vrijednost.

13.2 Objasnite zbog čega se dekoderi nazivaju i generatori mintermi.

13.3 Nacrtajte strukturu dekodera 2/4 realiziranog samo pomoću NAND logičkih kola, kao i samo pomoću NOR logičkih kola. Koja je od ove dvije realizacije ekonomičnija?

13.4 Nacrtajte strukturu dekodera 2/4 sa invertiranim izlazima (generatora makstermi) realiziranog samo pomoću NAND logičkih kola, kao i samo pomoću NOR logičkih kola. Koja je od ove dvije realizacije ekonomičnija?

13.5 Kako izgleda struktura dekodera 1/2?

13.6 Realizirajte logičke funkcije $Y = m_0 \vee m_2 \vee m_3 \vee m_6 \vee m_7$ i $Z = M_1 M_2 M_4 M_5 M_6$ pomoću jednog dekodera 3/8 i dva OR logička kola, uz pretpostavku da sve minterme odnosno maksterme zavise od promjenljivih A, B i C.

13.7 Realizirajte logičke funkcije $Y = m_0 \vee m_2 \vee m_3 \vee m_6 \vee m_7$ i $Z = M_1 M_2 M_4 M_5 M_6$ pomoću jednog dekodera 3/8 sa invertiranim izlazima i dva OR logička kola, uz pretpostavku da sve minterme odnosno maksterme zavise od promjenljivih A, B i C.

13.8 Projektirajte kombinacioni sklop koji realizira funkciju $Y = (3X^2 + 5) \bmod 5$ gdje operacija "mod" označava "ostatak pri dijeljenju sa", pri čemu je ulazni podatak X 3-bitni broj. Za realizaciju koristiti dekoder 3/8 i tri OR logička kola.

13.9 Realizirajte puni sabirač ukoliko su na raspolaganju samo OR logička kola i
a) dekoder 3/8 b) dekoder 3/8 sa invertiranim izlazima

13.10 Potrebno je napraviti kombinacioni sklop sa 3 ulaza koji na svom izlazu daje logičku jedinicu ako i samo ako su na tačno dva od tri ulaza jedinice, a na preostalom ulazu 0. Realizirajte ovaj sklop pomoću odgovarajućeg dekodera i jednog OR logičkog kola.

13.11 Realizirajte logičku funkciju $Y = \overline{B\overline{A \oplus C} \vee D(A \oplus C)}$ pomoću dekodera 4/16 i jednog OR logičkog kola.

13.12 Realizirajte logičku funkciju $Y = AB\overline{C}\,\overline{D} \vee \overline{A}\,\overline{B}D \vee \overline{A}C\overline{D} \vee CD$ pomoću dekodera 4/16 i jednog OR logičkog kola.

13.13 Objasnite primjenu strategije "podijeli i osvoji" za potrebe ekonomičnije realizacije dekodera sa većim brojem ulaza i izlaza.

13.14 Odredite koliko je ulaza u logička kola (ne brojeći invertore) potrebno za realizaciju dekodera 10/1024 ukoliko se dekoder realizira
a) neposredno, na najočigledniji mogući način;
b) primjenom strategije "podijeli i osvoji", tj. razbijanjem na dva dekodera 5/32 i kombiniranjem njihovih izlaza (pri tome, se pretpostavlja da se dekoderi 5/32 realiziraju neposredno).

13.15 Za realizaciju dekodera strategijom "podijeli i osvoji" nije neophodno vršiti razbijanje na dva jednaka manja dekodera (to nije ni moguće ukoliko je n neparan broj). Utvrdite kolika se ušteda u broju ulaza u AND logička kola ostvaruje ukoliko se dekoder 5/32 realizira razbijanjem na jedan dekoder 2/4 i jedan dekoder 3/8.

13.16 Utvrdite koliko je ulaza u AND logička kola potrebno za realizaciju dekodera 8/256 razbijanjem na dva dekodera, od kojih je jedan tipa $p/2^p$, a drugi tipa $q/2^q$ uz $p + q = 8$. Analizu izvršite za vrijednosti $p = 2$, $p = 3$, $p = 4$, $p = 5$ i $p = 6$. Šta se može zaključiti?

13.17* Dokažite da u slučaju kada je n ma kakav paran broj, strategija "podijeli i osvoji" daje najveće uštede u slučaju kada se dekoder $n/2^n$ razbija na dva jednaka dekodera $(n/2)/2^{n/2}$.

13.18 Posmatrajmo opći slučaj dekodera $n/2^n$. Izvedite opće formule za neophodan broj ulaza u logička kola (ne brojeći invertore) za realizaciju ovog dekodera ukoliko se on realizira

a) neposredno;

b) primjenom strategije "podijeli i osvoji", razbijanjem na dva dekodera $(n/2)/2^{n/2}$ (uz pretpostavku da je n paran) i kombiniranjem njihovih izlaza, pri čemu se dekoderi $(n/2)/2^{n/2}$ realiziraju neposredno;

c) dvostrukom primjenom strategije "podijeli i osvoji", tj. razbijanjem na dva dekodera $(n/2)/2^{n/2}$ i kombiniranjem njihovih izlaza, pri čemu se dekoderi $(n/2)/2^{n/2}$ također realiziraju primjenom strategije "podijeli i osvoji" (tj. razbijanjem na dekodere $(n/4)/2^{n/4}$, pri čemu ćemo podrazumijevati da je n djeljiv sa 4).

Izračunajte konkretno broj ulaza za sva tri slučaja za $n = 4$, $n = 8$, $n = 12$, $n = 16$, $n = 20$ i $n = 24$. Primjećujete li neku karakterističnu zavisnost između ostvarene uštede i broja n?

13.19 Objasnite ulogu omogućavajućeg ulaza E u standardnim izvedbama dekodera kao samostalnih komponenti.

13.20 Nacrtajte unutrašnju strukturu dekodera 2/4 sa omogućavajućim ulazom.

13.21 Realizirajte dekoder 4/16 koristeći pet dekodera 2/4 sa omogućavajućim ulazom. Da li tako realizirani dekoder također ima omogućavajući ulaz?

13.22 Realizirajte dekoder 3/8 koristeći dva dekodera 2/4 sa omogućavajućim ulazom i jedan invertor. Da li tako realizirani dekoder također ima omogućavajući ulaz?

13.23 Realizirajte logičku funkciju $Y = \overline{A}BC \vee AC \vee AD$ ukoliko su na raspolaganju dva dekodera 3/8 sa omogućavajućim ulazom, jedan invertor, i jedno višeulazno OR kolo.

13.24 Uporedite neophodan broj ulaza u AND logička kola za realizaciju dekodera sa omogućavajućim ulazom formata $(m+n)/2^{m+n}$ ukoliko se on realizira neposredno sa realizacijom zasnovanom na jednom dekoderu formata $m/2^m$ i 2^m dekodera formata $n/2^n$ (također sa omogućavajućim ulazima), uz pretpostavku da se dekoderi formata $m/2^m$ i $n/2^n$ realiziraju neposredno.

13.25 Odredite neophodan broj ulaza u AND logička kola za realizaciju dekodera 5/32 sa omogućavajućim ulazima ukoliko se on realizira

a) neposredno;

b) koristeći jedan dekoder 1/2 i 2 dekodera 4/16;

c) koristeći jedan dekoder 2/4 i 4 dekodera 3/8;

d) koristeći jedan dekoder 3/8 i 8 dekodera 2/4;

e) koristeći jedan dekoder 4/16 i 16 dekodera 1/2.

U svim slučajevima pretpostaviti da se pomoćni dekoderi od kojih se gradi traženi dekoder 5/32 realiziraju neposredno.

13.26 Definirajte precizno pojam ROM memorije. Posebno obratite pažnju na ulogu adresa, adresnih ulaza i izlaza podataka.

13.27 U nekoj ROM memoriji kapaciteta 4 bajta na adrese 0, 1, 2 i 3 upisani su redom brojevi 137, 34, 9 i 92 (naravno, u binarnom zapisu). Nacrtajte strukturu ove ROM memorije (realizirane na bazi dekodera).

13.28 Koliki je kapacitet (u kilobajtima) ROM memorije koja ima 14 ulaza i 4 izlaza?

13.29 Koliko adresnih ulaza mora imati ROM memorija kapaciteta 4KB ogranizirana tako da svaka adresa sadrži 4-bitni poadatak?

13.30 Kao što je već rečeno, svaki kombinacioni sklop se, uz odgovarajuću interpretaciju, može posmatrati kao ROM memorija. Opišite uz kakvu interpretaciju se kombinacioni sklop iz Primjera 11.3, koji sabira dva broja od po dva bita, može interpretirati kao ROM memorija.

13.31 Nacrtajte strukturu PROM (ili EPROM odnosno EEPROM) memorije koja posjeduje 8 adresa i koja pamti četvorobitne podatke. Na slici označite ispunjenim kružićem čvorove koji treba da budu aktivni (nespaljeni, odnosno vodljivi) ukoliko želimo da se u ovu memoriju na adrese od 0 do 7 upišu respektivno brojevi 44, 117, 39, 19, 5, 191, 0 i 14. Detaljno opišite postupak koji treba obaviti da se u ovu memoriju na adresu 3 upiše podatak 19.

13.32 Odredite koliko ulaza, koliko izlaza i koliko pregorljivih čvorova treba da ima PROM memorija koja može da zapamti

a) 128 16-bitnih podataka b) 512 8-bitnih podataka

13.33 Realizirajte puni sumator ukoliko nam je na raspolaganju PROM/EPROM/EEPROM memorija kapaciteta 8×4 bita (tj. koja može pamtiti 8 4-bitnih podataka.

13.34 Realizirajte puni sumator ukoliko nam je na raspolaganju PROM/EPROM/EEPROM memorija kapaciteta 16×8 bita.

13.35 Realizirajte kombinacioni sklop iz Primjera 11.3 koji sabira dva broja od po dva bita ukoliko nam je na raspolaganju PROM/EPROM/EEPROM memorija kapaciteta 16×4 bita.

13.36 Realizirajte konverzionu matricu za upravljanje sedmosegmentnim displejom iz Primjera 11.4 ukoliko nam je na raspolaganju PROM/EPROM/EEPROM memorija kapaciteta 16×8 bita.

13.37 Realizirajte konverzionu matricu za pretvaranje klasičnog 4-bitnog binarnog kôda u 4-bitni Grayev kôd opisanu u Zadatku 11.35 ukoliko nam je na raspolaganju PROM/EPROM/EEPROM memorija kapaciteta 16×8 bita.

13.38 Objasnite šta su PLA komponente, nacrtajte njihovu principijelnu građu, i opišite njihove prednosti i mane u odnosu na programabilne ROM memorije.

13.39 Nacrtajte strukturu PLA komponente sa 4 ulaza, 2 izlaza i 5 AND kola. Nakon toga, na slici označite koje čvorove treba zadržati a koje spaliti (uz pretpostavku da je PLA komponenta neizbrisivog tipa) da bi ta PLA komponenta realizirala skup logičkih funkcija

$Y = ABC \vee \overline{A}\,\overline{B} \vee \overline{D}$ $Z = \overline{B}C \vee A$

Može li se isti skup logičkih funkcija realizirati PLA komponentom koja sadrži manje od 5 AND logičkih kola?

13.40 Koliko pregorljivih čvorova sadrži PLA komponenta (neizbrisivog tipa) koja sadrži *n* ulaza, *m* izlaza i *p* AND logičkih kola?

13.41 Kakva praktična ograničenja na realizaciju logičkih funkcija nameću PLA komponente kod kojih AND logička kola sadrže manje od *n* ulaza (pri čemu je *n* broj ulaza u PLA komponentu), odnosno kod kojih OR logička kola sadrže manje od *p* ulaza (pri čemu je *p* broj AND logičkih kola)?

13.42 Dat je skup logičkih funkcija (od kojih su sve tri u MDNF obliku):

$X = A\overline{C}\,\overline{D} \vee \overline{A}C \vee BD$ $Y = AC \vee BD$ $Z = A\overline{D} \vee C$

a) Realizirajte ovaj skup logičkih funkcija uz pomoć PLA komponente sa 4 ulaza i 3 ulaza, bez ikakvog prethodnog transformiranja navedenih logičkih funkcija. Koliko AND kola mora sadržavati ta PLA komponenta da bi realizacija bila izvodljiva?

b) Pokažite kako se dati skup logičkih funkcija može realizirati ukoliko je na raspolaganju PLA komponenta koja sadrži svega četiri AND logička kola.

13.43 Neka je dat skup logičkih funkcija

$X = m_1 \vee m_2 \vee m_3 \vee m_5 \vee m_7 \vee m_9 \vee m_{11} \vee m_{13} \vee m_{15}$

$Y = M_2 M_3 M_4 M_5 M_6 M_7 M_8 M_{10} M_{12} M_{14}$

$Z = m_0 \vee m_1 \vee m_2 \vee m_3 \vee m_5 \vee m_7$

pri čemu sve minterme zavise od promjenljivih A, B, C, D. Realizirajte ovaj skup funkcija uz pomoć PLA komponente koja posjeduje 4 ulaza, 3 izlaza i 4 AND logička kola.

13.44 Realizirajte konverzionu matricu za pretvaranje klasičnog 4-bitnog binarnog kôda u 4-bitni Grayev kôd opisanu u Zadatku 11.35 ukoliko nam je na raspolaganju PLA komponenta sa dovoljnim brojem ulaza, izlaza i AND logičkih kola. Koliki minimalni broj ulaza, izlaza i AND logičkih kola mora posjedovati ova PLA komponenta da bi realizacija bila moguća?

13.45* Realizirajte konverzionu matricu za upravljanje sedmosegmentnim displejom iz Primjera 11.4 ukoliko nam je na raspolaganju PLA komponenta sa dovoljnim brojem ulaza i izlaza, ali sa svega 10 AND logičkih kola.

13.46 Objasnite šta je koder, i opišite njegove moguće primjene u računarskoj tehnici.

13.47 Koliko izlaza mora imati koder koji posjeduje 11 ulaza?

13.48 Koliko ulaza može imati koder koji ima 5 izlaza (računajući i izlaz S)?

13.49 Nacrtajte strukturu kodera 4/2 realiziranog samo pomoću NAND logičkih kola, kao i samo pomoću NOR logičkih kola. Koja je od ove dvije realizacije ekonomičnija?

13.50 Analizirajte kakve će biti vrijednosti na izlazu iz kodera 4/2 ukoliko se na više od jednog ulaza dovede logička jedinica.

13.51 Generalizirajući Primjer 13.5, naslutite kako izgledaju jednačine koje opisuju koder *p*/*n* u općem slučaju.

13.52 Koristeći formalne metode projektiranja kombinacionih logičkih sklopova, realizirajte prioritetni koder 4/2. Za realizaciju koristiti AND, OR i NOT logička kola. Provjerite intuitivnim putem korektnost dobijenih jednačina.

13.53 Realizirajte prioritetni koder 4/2 uz pomoć dekodera 4/16 i OR logičkih kola.

13.54 Intuitivnim putem reazizirajte koder 8/3 (bez prioriteta) uz pomoć dva kodera 4/2 i eventualno nešto dodatnih logičkih kola.

13.55* Intuitivnim putem realizirajte prioritetni koder 8/3 uz pomoć dva prioritetna kodera 4/2 i nešto dodatnih logičkih kola. Provjerite da li se u konačnici dobijaju jednačine ekvivalentne jednačinama izvedenim u Primjeru 13.6.

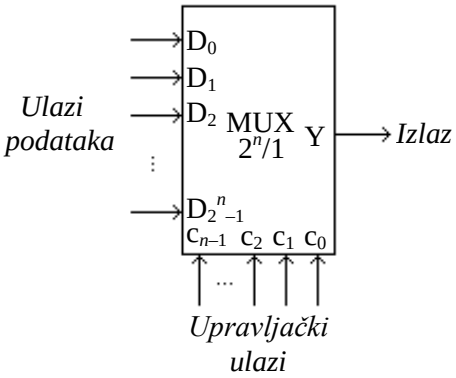
13.56 Neka zgrada ima prizemlje i tri sprata. Lift se može pozvati sa bilo kojeg sprata (uključujući i prizemlje), pri čemu se, u slučaju da je lift pozvan sa više od jednog mjesta, prioritet daje najvišem spratu. Motor lifta na ulazu zahtijeva tri logička signala S, k_1 i k_0, pri čemu $S = 1$ označava da je lift pozvan bilo odakle (u suprotnom je $S = 0$), dok 2-bitni binarni broj $(k_1;k_0)_2$ predstavlja redni broj sprata na koji lift treba da ide (podrazumijeva se da motor lifta na neki način zna na kojem se spratu lift trenutno nalazi, tako da mu je primljena informacija dovoljna da ispravno odradi nalog). Pri tome se prizemlje tretira kao sprat 0. Ove logičke signale motor lifta dobija sa izlaza iz nekog digitalnog sklopa, na čije ulaze dolaze 4 signala P_0, P_1, P_2 i P_3. Ovi signali dolaze sa tastera za poziv lifta, tako da je $P_i = 1$ ($i = 0..3$) ako i samo ako je lift pozvan sa *i*-tog sprata. Projektirajte strukturu takvog digitalnog sklopa.

13.57 Riješite isti problem kao u prethodnom zadatku, ali uz pretpostavku da se prizemlju daje apsolutni prioritet (tj. ukoliko je lift pozvan iz prizemlja, on ide u prizemlje, bez obzira na eventualne druge pozive). Također razmislite kako bi se ovaj problem mogao riješiti istim sklopom koji rješava problem iz prethodnog zadatka, uz nešto dodatnih logičkih kola.

14. Multiplekseri i njihove primjene

Još jedan izuzetno važan kombinacioni sklop u računarskoj tehnici je sklop poznat pod imenom ***multiplekser*** ili ***selektor***. To je kombinacioni sklop koji posjeduje samo jedan izlaz i $n + 2^n$ ulaza koji su razvrstani u dvije skupine: n ***upravljačkih*** (***selektorskih***, ***adresnih***) ***ulaza*** i 2^n ***ulaza podataka***. Izlaz iz multipleksera jednak je *onom od ulaza podataka čiji redni broj* (*indeks*) *pretvoren u binarni zapis odgovara kombinaciji bita dovedenoj na upravljacke ulaze*. Drugim riječima multiplekser prosljeđuje jedan i samo jedan od svojih ulaza podataka na izlaz, i to onaj ulaz čiji je redni broj definiran vrijednostima upravljačkih ulaza.

Multiplekser sa n upravljačkih ulaza obično se naziva "multiplekser 2^n u 1" ili "multiplekser $2^n/1$" (skraćeno "MUX $2^n/1$"), a obično se na shemama predstavlja sljedećim blokom (upravljački ulazi se ponekad umjesto sa c_i obilježavaju sa A_i):



Na osnovu definicije multipleksera, vidimo da se njegov rad može iskazati sljedećom jednačinom:

$$Y = \begin{cases} D_0, \text{ ako je } (c_{n-1};...c_2;c_1;c_0)_2 = (0...000)_2 \\ D_1, \text{ ako je } (c_{n-1};...c_2;c_1;c_0)_2 = (0...001)_2 \\ D_2, \text{ ako je } (c_{n-1};...c_2;c_1;c_0)_2 = (0...010)_2 \\ \quad ... \\ D_{2^n-1}, \text{ ako je } (c_{n-1};...c_2;c_1;c_0)_2 = (1...111)_2 \end{cases}$$

Na primjer, za n = 3 imamo:

$$Y = \begin{cases} D_0, \text{ ako je } (c_2;c_1;c_0)_2 = (000)_2 \\ D_1, \text{ ako je } (c_2;c_1;c_0)_2 = (001)_2 \\ D_2, \text{ ako je } (c_2;c_1;c_0)_2 = (010)_2 \\ D_3, \text{ ako je } (c_2;c_1;c_0)_2 = (011)_2 \\ D_4, \text{ ako je } (c_2;c_1;c_0)_2 = (100)_2 \\ D_5, \text{ ako je } (c_2;c_1;c_0)_2 = (101)_2 \\ D_6, \text{ ako je } (c_2;c_1;c_0)_2 = (110)_2 \\ D_7, \text{ ako je } (c_2;c_1;c_0)_2 = (111)_2 \end{cases}$$

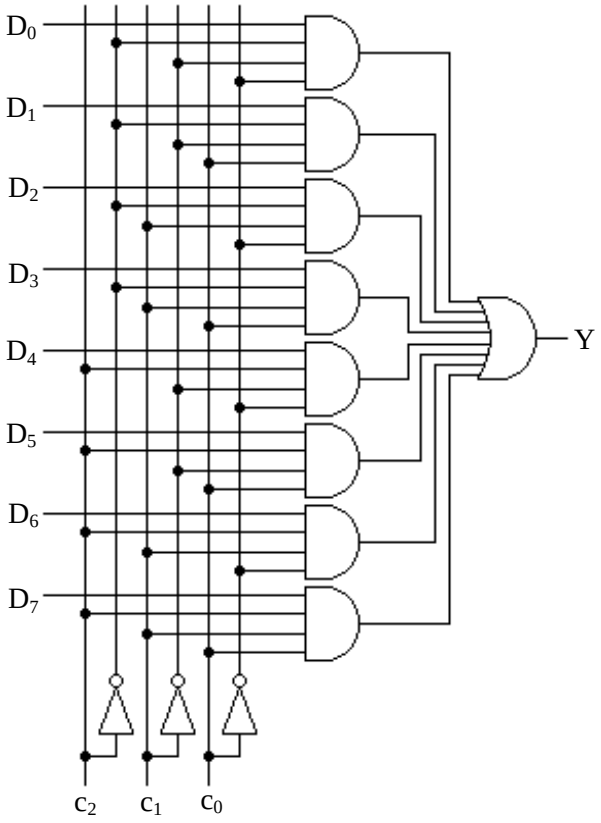
Funkciju Y nije praktično prikazivati potpunom tablicom istine jer ona ovisi od $n + 2^n$ promjenljivih. U konkretnom slučaju $n = 3$, Y je funkcija 11 promjenljivih $D_0 \div D_7$ i $c_0 \div c_2$. Za njen prikaz tablicom istine, trebala bi nam tablica od $2^{11} = 2048$ redova. Medutim, ovu funkciju mozemo prikazati u vidu neke vrste *pakovane tablice istine* na sljedeći način:

c_2	c_1	c_0	Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

Ova tablica nam omogućava da napišemo funkciju Y u vidu nečega što po formi podsjeća na savršenu disjunktivnu normalnu formu, ali u kojoj promjenljive od D_0 do D_7 igraju ulogu dopunskih množilaca:

$$Y = D_0\bar{c}_2\bar{c}_1\bar{c}_0 \vee D_1\bar{c}_2\bar{c}_1c_0 \vee D_2\bar{c}_2c_1\bar{c}_0 \vee D_3\bar{c}_2c_1c_0 \vee D_4c_2\bar{c}_1\bar{c}_0 \vee D_5c_2\bar{c}_1c_0 \vee D_6c_2c_1\bar{c}_0 \vee D_7c_2c_1c_0$$

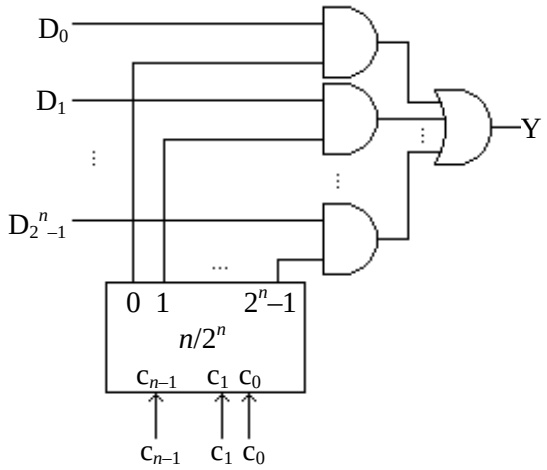
Na osnovu ove funkcije možemo lako nacrtati kako izgleda unutrašnja struktura multipleksera 8/1:



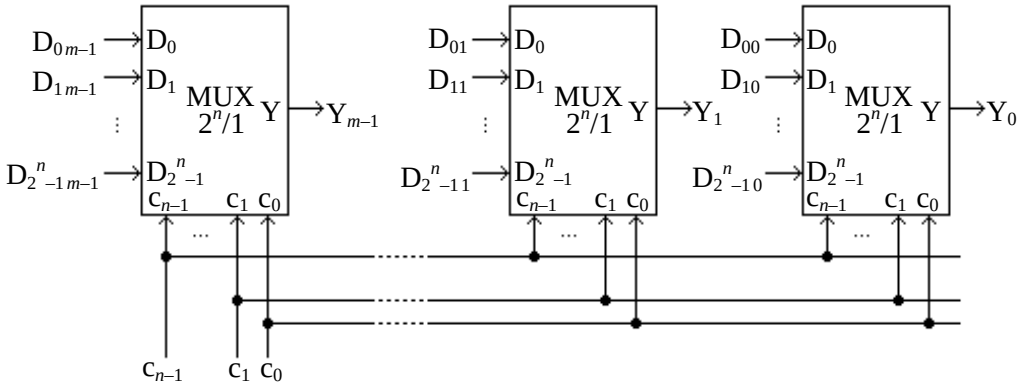
U praksi se multiplekseri zbog nekih tehnoloških razloga rijetko realiziraju prema gornjoj shemi, ali logika rada ostaje ista. Zapravo, multiplekseri su tijesno povezani sa dekoderima, i najčešće se realiziraju upravo preko dekodera i dodatne logike. Naime, primijetimo da izlaznu funkciju multipleksera možemo napisati u sljedećem obliku

$$Y = D_0 m_0 \vee D_1 m_1 \vee \ldots D_{2^n-1} m_{2^n-1}$$

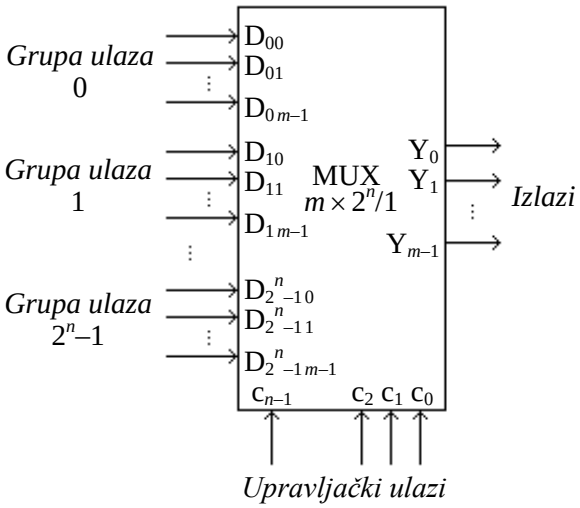
pri čemu su m_i, $i = 0..2^n-1$ minterme obrazovane od promjenljivih c_{n-1}, ... c_2, c_1 i c_0. Kako sve ove minterme možemo dobiti sa izlaza dekodera $n/2^n$, neposredno slijedi da se multiplekser $2^n/1$ može realizirati na kao na sljedećoj slici:



Na prvi pogled, ovakva realizacija djeluje manje ekonomično nego neposredna realizacija, s obzirom na složenost dekodera. Međutim, u u računarskoj tehnici se veoma često dešava da nam je potrebno nekoliko (recimo *m*) multipleksera čiji su svi adresni ulazi spojeni zajedno, kao na sljedećoj slici:

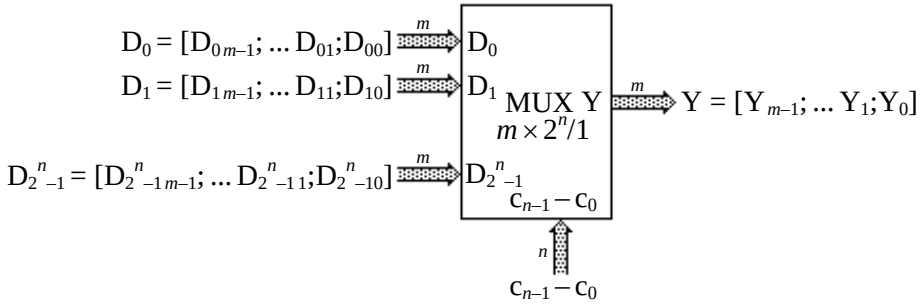


Ovakav sistem multipleksera mnogo je efikasnije realizirati preko dekodera i prateće logike, jer svih *m* multipleksera mogu koristiti jedan te isti dekoder, s obzirom da su im svi upravljački ulazi vezani zajedno. Često se ovakav sistem proizvodi kao jedna komponenta, koja se tada naziva ***višestruki multiplekser***, *m*-bitni multiplekser ili "multiplekser $m \times 2^n/1$", a crta se kao na sljedećoj slici:



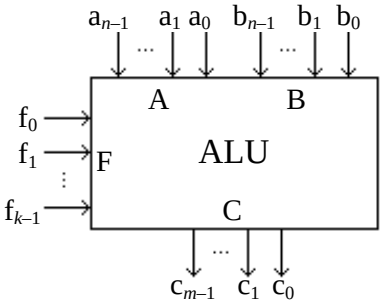
Za višestruki multiplekser može se smatrati da prosljeđuje na izlaze bite jednog i samo jednog od 2^n *m*-bitnih binarnih brojeva dovedenih na jednu od 2^n grupa ulaza podataka, i to onog binarnog broja čiji je redni broj definiran vrijednostima upravljačkih ulaza.

Simbol multipleksera $m \times 2^n/1$, ovako kako je prikazan gore, dosta je glomazan i nepregledan. Međutim, blokovi poput ovoga veoma se često susreću u sklopovima u računarskoj tehnici. Stoga se, radi pojednostavljenja shematskih dijagrama koji sadrže složenije blokove, linije koje vode signale iste vrste (npr. bite nekog binarnog broja) obično ne crtaju kao posebne linije, nego kao jedna *debela osjenčena* (*šrafirana*) *linija*, pri čemu se tačan broj linija koji tvore sabirnicu eventualno piše kraj oznake sabirnice. Tako se, na primjer, multiplekser $m \times 2^n/1$ predstavlja blokom kao na sljedećoj slici:

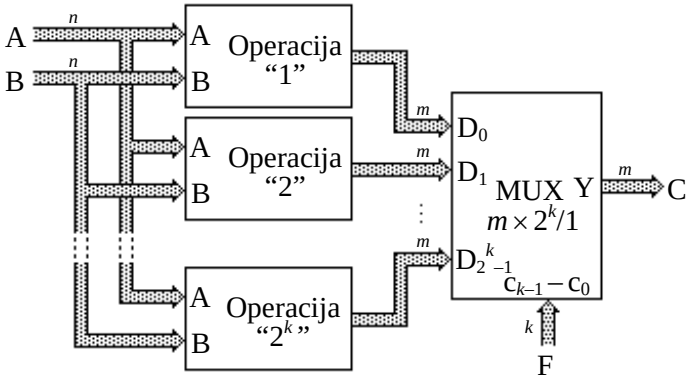


Inače, skup linija (provodnika) koji prenose međusobno srodne signale (koji najčešće predstavljaju pojedinačne bite nekog višebitnog podatka) obično se naziva ***sabirnica***. Stoga možemo reći da se na složenijim shemama pojedinačni provodnici koji čine sabirnicu ne crtaju individualno, već se čitava sabirnica crta kao jedna debela linija.

Multiplekseri imaju mnogobrojne primjene u računarskim sistemima. Na ovom mjestu ćemo prikazati primjenu multipleksera za projektiranje ***aritmetičko-logičke jedinice*** (skraćeno ***ALU*** od engl. Arithmetical-Logical Unit). Aritmetičko-logička jedinica je veoma važan dio računarskih sistema, i to onaj dio koji je zadužen za izvođenje računskih operacija. ALU je jedan od glavnih dijelova *procesora*, čiju ćemo strukturu kasnije detaljno upoznati. Formalno definirano, ALU je zapravo jedan kombinacioni sklop sa mnogo ulaza, koji se mogu podijeliti u tri grupe: dvije grupe ***ulaza operanada*** (A i B ulazi) i ***upravljački ulazi*** odnosno ***ulazi za izbor operacije*** (F ulazi), kao i jednu grupu izlaza (***izlazi za rezultat***):

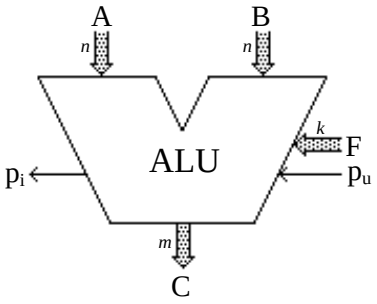


Na A ulaze dovodi se jedan binarno zapisani broj od *n* bita, a na B ulaze drugi binarno zapisani broj također od *n* bita. Na izlazima C pojavljuje se rezultat operacije C = A *op* B, pri čemu izbor operacije *op* koja će biti izvršena zavisi od stanja upravljačkih ulaza F. O kojim se konkretnim operacijama radi, zavisi od konkretne aritmetičko logičke jedinice. Uglavnom, sa *k* upravljačkih ulaza obilježenih sa f_i, $i = 0..k–1$ moguće je realizirati 2^k različitih operacija. Postoji mnogo načina kako se ALU može izvesti, međutim principijelna struktura većine ALU može se prikazati sljedećom blok shemom:



Suština je da se zapravo svih 2^k mogućih operacija koje ALU obavlja izvode istovremeno (paralelno) u blokovima označenim sa "Operacija "1"", "Operacija "2"" itd. a zatim se pomoću multipleksera vrši izbor koji će od 2^k izračunatih rezultata zaista izaći kao izlaz iz aritmetičko-logičke jedinice. Ukoliko sklopovi za izvođenje pojedinih operacija imaju zajedničkih elemenata, struktura ALU se može donekle pojednostaviti, ali generalni princip ostaje nepromijenjen.

Kada smo razmatrali *n*-bitni paralelni sumator, vidjeli smo da se može pojaviti prenos kao izlaz pri sabiranju posljednjih bita, a da je korisno imati i ulazni prenos da bi se sumatori manje dužine mogli kaskadno vezivati, tvoreći tako sumatore veće dužine. Sličan princip može se primijeniti i kod aritmetičko-logičkih jedinica, tako da praktične izvedbe ALU uvijek imaju još jedan dodatni ulaz p_u nazvan ***ulazni prenos*** i još jedan izlaz p_i zvani ***izlazni prenos*** koji omogućavaju da se više manjih aritmetičko-logičkih jedinica kaskadno vežu u jednu veću ALU. Najčešće se proizvode 4-bitne ALU, tako da se npr. 16-bitna ALU može formirati kaskadnim vezivanjem četiri 4-bitne ALU. ALU se na shemama obično obilježava na sljedeći način:



Aritmetičko-logičke jedinice obično obavljaju kako aritmetičke operacije (poput sabiranja), tako i logičke operacije (poput poređenja, konjukcije, itd.), odakle im i potiče ime, mada postoje i ALU koje obavljaju isključivo aritmetičke ili isključivo logičke operacije. Tako se, na primjer, sklop koji može obavljati kako sabiranje tako i oduzimanje, razmotren u Poglavlju 12, kao i sklopovi projektirani u zadacima 11.32 i 12.15 mogu smatrati jednostavnim primjerima aritmetičko logičkih jedinica. Sklop iz Zadatka 12.15 predstavlja jednostavnu 1-bitnu ALU koja obavlja jednu aritmetičku i 6 logičkih operacija. Ovi primjeri ukazuju da se aritmetičko-logičke jedinice mogu načelno realizirati i bez multipleksera (samo je projektiranje takvih ALU mnogo složenije).

Primjer 14.1:

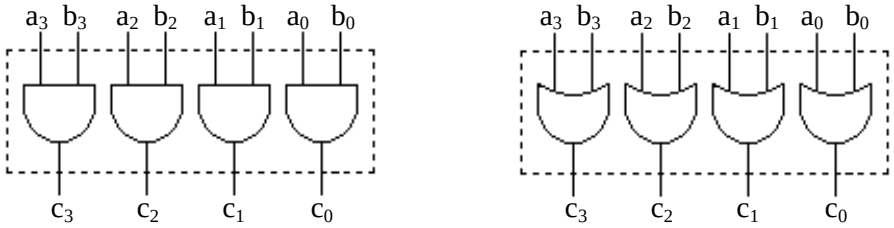
- Projektirati aritmetičko-logičku jedinicu koja obavlja četiri operacije, i to dvije *logičke, konjunkciju* i *disjunkciju,* i dvije *aritmeticke, sabiranje* i *oduzimanje*. Oba operanda imaju po četiri bita a rezultat ima također četiri bita, pri čemu eventualni peti bit koji se može pojaviti u rezultatu sabiranja treba tretirati kao izlazni prenos (za slučaj oduzimanja, bit koji signalizira negativan rezultat treba također tretirati kao izlazni prenos, preciznije kao izlaznu posudbu). Također je potrebno predvidjeti i ulaz za ulazni prenos. Logičke operacije treba obavljati sa svakim bitom posebno.

Da bismo dobili četiri operacije, potrebna su nam dva upravljačka ulaza. Operacijama je neophodno dodijeliti odgovarajuće kodove (šifre), pri čemu je izbor kodova potpuno proizvoljan. Odlučićemo se za kodiranje prema sljedećoj tablici:

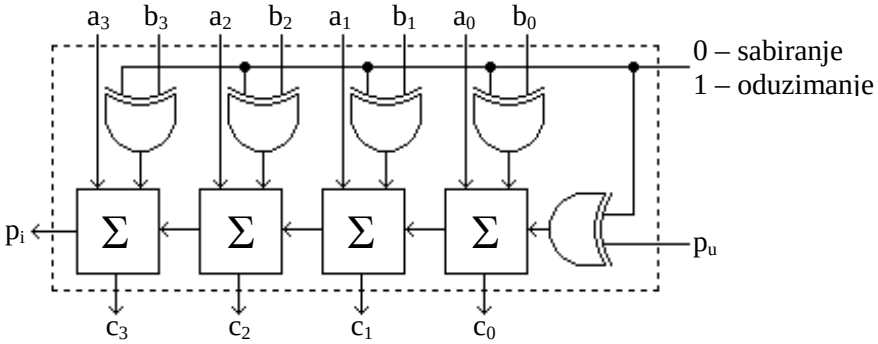
OPERACIJA	f_1	f_0
$A \wedge B$	0	0
$A \vee B$	0	1
$A + B$	1	0
$A - B$	1	1

Sada je potrebno projektirati blokove koji obavljaju svaku od neophodnih operacija. Operacije konjunkcije i disjunkcije je posve trivijalno realizirati, jer se izvode nad individualnim bitima (ulazni i

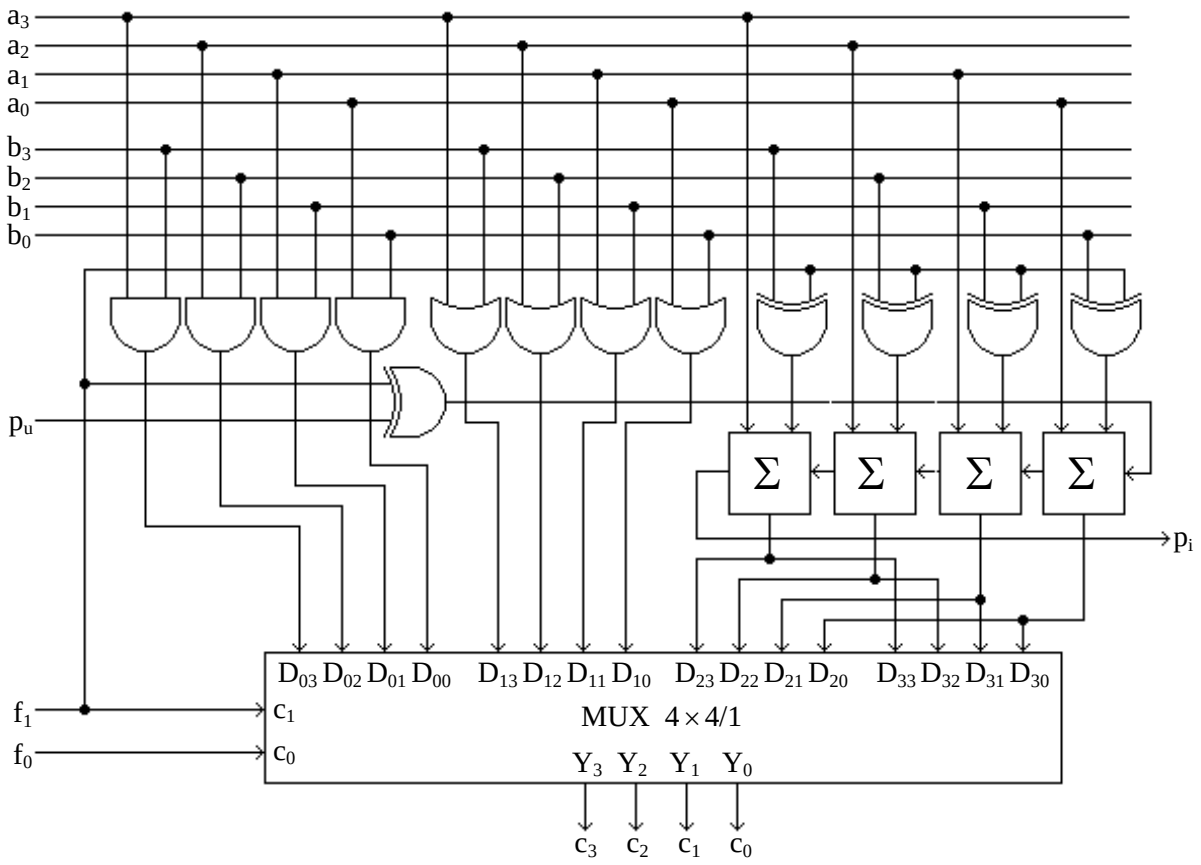
izlazni prenos očigledno ne igraju nikakvu ulogu kod ovih operacija). Ovim operacijama odgovaraju respektivno blokovi prikazani na sljedećoj slici:



Operacije sabiranja i oduzimanja već smo ranije realizirali, i vidjeli smo da se obje ove operacije izvode slično, tako da se, uz malo dosjetki, obje operacije mogu izvesti pomoću *istog sklopa*. To ćemo iskoristiti i ovdje, tj. iskoristićemo jedan sklop da obavlja obje operacije, prema sljedećoj slici:



Ovaj sklop sam za sebe već predstavlja malu aritmetičko-logičku jedinicu koja izvršava dvije operacije (sabiranje i oduzimanje). Preostaje još samo da sastavljene blokove uklopimo u generalnu shemu aritmetičko-logičke jedinice. Ulaz za izbor operacije u sabiraču/oduzimaču treba spojiti na ulaz f_0 jer se preko njega vrši izbor da li je operacija sabiranje ili oduzimanje. Konačno, dobijamo sljedeću shemu tražene aritmetičko-logičke jedinice:



Primijetimo da ovako realizirana aritmetičko-logička jedinica generalizira logičke operacije konjukcije i disjunkcije na *višebitne brojeve* (tj. brojeve različite od 0 i 1), pri čemu se ove operacije izvode nad individualnim bitima pojedinačno. Tako je, na primjer, $45 \wedge 14 = 12$ i $45 \vee 14 = 47$, jer je $45 = (101101)_2$ i $14 = (001110)_2$, pa je $101101 \wedge 001110 = 001100$ i $101101 \vee 001110 = 101111$, a dalje je $(001100)_2 = 12$ i $(101111)_2 = 47$. Ovako generalizirana "konjukcija" i "disjunkcija" imaju veliku primjenu u računarskoj tehnici, a posebno u mašinskom programiranju, s kojim ćemo se upoznati kasnije. Primijetimo da, uz ovako generaliziranu konjukciju, ne smijemo pisati $A \cdot B$ odnosno AB umjesto $A \wedge B$, jer je npr. $45 \wedge 14 = 12$, a $45 \cdot 14 = 630$. Napomenimo da se ovako generalizirane operacije konjukcije i disjunkcije obično obilježavaju znacima "&" i "|" umjesto "$\wedge$" i "$\vee$" (ovo je preuzeto iz programskog jezika C), tako da umjesto $A \wedge B$ odnosno $A \vee B$ obično pišemo $A \,\&\, B$ odnosno $A \mid B$ (npr. $45 \,\&\, 14 = 12$, i $45 \mid 14 = 47$).

Pored svoje očigledne primjene u situacijama kada treba jedan od ulaza (ili grupu ulaza) proslijediti na izlaz zavisno od situacije, multiplekseri mogu biti i jako koristan element za *realizaciju proizvoljnih logičkih funkcija*, naročito onih sa većim brojem promjenljivih. Najbolji rezultati se postižu kombiniranjem multipleksera i klasičnih logičkih kola. Ukoliko želimo da realiziramo funkciju od n promjenljivih, a na raspolaganju nam je multiplekser sa $m \le n$ adresnih ulaza, funkciju koju želimo da realiziramo prvo trebamo raspisati u disjunktivnoj normalnoj formi (ne nužno minimalnoj). Nakon toga, izaberemo po volji m promjenljivih koje dovedemo na adresne ulaze (takve promjenljive ćemo zvati ***adresne promjenljive***), a zatim izvršimo proširivanje disjunktivne normalne forme tako da se u svakom njenom članu pojave sve adresne promjenljive. Kada to izvršimo, veoma je lako upoređivanjem sa funkcijom koja opisuje rad multipleksera zaključiti šta treba dovesti na ulaze podataka da bi se realizirala tražena funkcija. To mogu biti ili konstante 0 ili 1, ili neke kombinacije od najviše $n–m$ promjenljivih koje *nisu adresne promjenljive* (takve promjenljive ćemo nazivati ***preostale promjenljive***). Ovu proceduru je najbolje ilustrirati kroz konkretne primjere.

Primjer 14.2:

- Realizirati funkciju $Y = \overline{A}B\overline{D} \vee A\overline{C}D\overline{E} \vee B\overline{D}E$ koristeći multiplekser 4/1.

Tražena funkcija ima 5 promjenljivih, a broj adresnih ulaza multipleksera 4/1 je 2. Slijedi da dvije promjenljive moraju biti adresne, dok će tri promjenljive biti preostale promjenljive. Odlučimo se da za adresne promjenljive izaberemo B i D. Možemo pisati:

$$Y = \overline{A}B\overline{D} \vee A\overline{C}D\overline{E} \vee B\overline{D}E = \overline{A}B\overline{D} \vee A\overline{C}D\overline{E}(B \vee \overline{B}) \vee B\overline{D}E =$$
$$= \overline{A}B\overline{D} \vee AB\overline{C}D\overline{E} \vee A\overline{B}\,\overline{C}D\overline{E} \vee B\overline{D}E$$

Sada je potrebno grupirati zajedno sve članove koji imaju ista stanja adresnih promjenljivih i reorganizirati članove tako da adresne promjenljive dodu na kraj svakog člana:

$$Y = (\overline{A} \vee E) \cdot B\overline{D} \vee A\overline{C}\,\overline{E} \cdot BD \vee A\overline{C}\,\overline{E} \cdot \overline{B}D$$

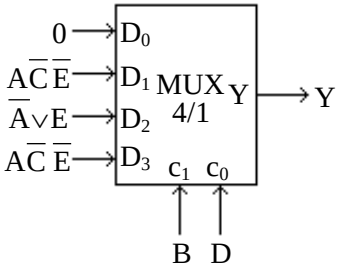
Ovako preuređenu funkciju možemo kompaktno zapisati u vidu sljedeće tablice:

B	D	Y
0	0	0
0	1	$A\overline{C}\,\overline{E}$
1	0	$\overline{A} \vee E$
1	1	$A\overline{C}\,\overline{E}$

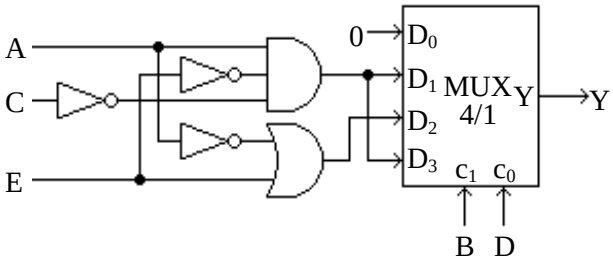
Ukoliko ovu tablicu uporedimo sa funkcijom koja opisuje rad multipleksera 4/1, vidimo da možemo postići istu funkcionalnost ako obezbijedimo da je

$D_0 = 0$, $D_1 = D_3 = A\overline{C}\,\overline{E}$ i $D_2 = \overline{A} \vee E$

Drugim riječima, zadana funkcija se može realizirati sklopom čija je principijelna struktura kao na sljedećoj slici:



Odavde je samo mali korak do potpune realizacije, koja je prikazana na sljedećoj slici:



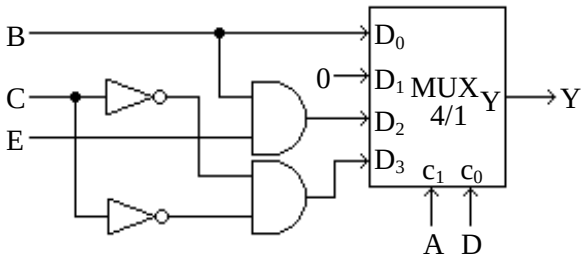
Vidimo da su nam za realizaciju potrebna tri invertora, jedno troulazno AND kolo, i jedno dvoulazno OR kolo, što ukupno čini 8 ulaza u prateća logička kola. Međutim, složenost prateće logike može drastično da varira u ovisnosti od izbora adresnih promjenljivih. Na primjer, ukoliko za adresne promjenljive uzmemo A i D, možemo pisati:

$$Y = \overline{A}B\overline{D} \vee A\overline{C}D\overline{E} \vee B\overline{D}E = \overline{A}B\overline{D} \vee A\overline{C}D\overline{E} \vee B\overline{D}E\,(A \vee \overline{A}) =$$
$$= \overline{A}B\overline{D} \vee A\overline{C}D\overline{E} \vee AB\overline{D}E \vee \overline{A}B\overline{D}E = (B \vee BE) \cdot \overline{A}\,\overline{D} \vee \overline{C}\,\overline{E} \cdot AD \vee BE \cdot A\overline{D} =$$
$$= B \cdot \overline{A}\,\overline{D} \vee \overline{C}\,\overline{E} \cdot AD \vee BE \cdot A\overline{D}$$

Traženu funkcionalnost možemo pomoću multipleksera 4/1 obezbijediti ako izaberemo da je

$D_0 = B$, $D_1 = 0$, $D_2 = BE$ i $D_3 = \overline{C}\,\overline{E}$

što dovodi do sljedeće realizacije, koja zahtijeva svega dva invertora i dva dvoulazna AND kola (ukupno 6 ulaza u prateća logička kola):



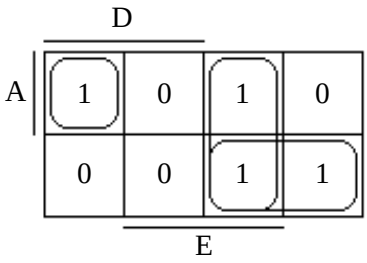
Vidimo da smo drugačijim izborom adresnih promjenljivih dobili povoljniju realizaciju. Razlike u povoljnosti realizacija za različite izbore adresnih promjenljivih mogu biti drastične. Na primjer, pokažimo šta se dobija ukoliko za adresne promjenljive izaberemo B i C:

$$Y = \overline{A}B\overline{D} \vee A\overline{C}D\overline{E} \vee B\overline{D}E = \overline{A}B\overline{D}\,(C \vee \overline{C}) \vee A\overline{C}D\overline{E}\,(B \vee \overline{B}) \vee B\overline{D}E\,(C \vee \overline{C}) =$$
$$= \overline{A}BC\overline{D} \vee \overline{A}B\overline{C}\overline{D} \vee ABC\overline{D}\overline{E} \vee A\overline{B}\,\overline{C}D\overline{E} \vee BC\overline{D}E \vee B\overline{C}\,\overline{D}E =$$
$$= (\overline{A}\,\overline{D} \vee \overline{D}E) \cdot BC \vee (\overline{A}\,\overline{D} \vee AD\overline{E} \vee \overline{D}E) \cdot B\overline{C} \vee AD\overline{E} \cdot \overline{B}\,\overline{C}$$

U ovom slučaju, traženu funkcionalnost pomoću multipleksera 4/1 možemo obezbijediti jedino ukoliko izaberemo da je

$$D_0 = AD\overline{E}\ ,\ D_1 = 0\ ,\ D_2 = \overline{A}\,\overline{D} \vee AD\overline{E} \vee \overline{D}E \quad \text{i} \quad D_3 = \overline{A}\,\overline{D} \vee \overline{D}E$$

Naravno, dobivene funkcije koje treba dovesti na ulaze podataka multipleksera uvijek treba probati optimizirati pomoću zakona Booleove algebre ili Veitchovih dijagrama, ali u ovom primjeru nije teško vidjeti da se sa ovim funkcijama ne može ništa učiniti po pitanju optimizacije. Na primjer, za funkciju D_2 imamo sljedeći Veitchov dijagram:



Vidimo da funkcija D_2 već predstavlja MDNF i ne može se dodatno optimizirati. Kada bismo realizirali funkciju Y sa ovakvim izborom adresnih promjenljivih, trebalo bi nam u najboljem slučaju 14 ulaza u prateća logička kola, i to ukoliko iskoristimo činjenicu da funkcije D_0, D_2 i D_3 imaju zajedničke članove, kao i činjenicu da cijela funkcija D_3 predstavlja dio funkcije D_2. S obzirom da bi bez primjene multipleksera realizacija funkcije Y zahtijevala 16 ulaza u logička kola, vidimo da je ušteda praktično nikakva, odnosno realizacija uz pomoć multipleksera 4/1 uz ovakav izbor adresnih promjenljivih je čak složenija nego bez njega!

Iz izloženog se može zaključiti da realizacija logičkih funkcija pomoću multipleksera može biti efikasna ili neefikasna, ovisno o izboru adresnih promjenljivih. Na žalost, ne postoji nikakvo egzaktno pravilo koje govori kako treba odabrati adresne promjenljive tako da se dobije najekonomičnija realizacija. Kao što ćemo uskoro vidjeti, Veitchovi dijagrami mogu biti od izvjesne pomoći, ali samo za slučaj manjeg broja promjenljivih. Jedini metod koji nam garantira najbolju realizaciju je isprobavanje svih mogućih kombinacija, i odabir najbolje. Međutim, broj takvih kombinacija za slučaj n promjenljivih i m adresnih ulaza iznosi $m!/[n!\,(m-n)!]$, što već za $n = 10$ i $m = 5$ iznosi 252 kombinacije. Radi ilustracije, u sljedećoj tabeli su rezimirani svi mogući slučajevi koji se mogu dobiti za posmatranu funkciju za sve moguće kombinacije od po dvije adresne promjenljive.

Adresne promjenljive	Neophodni ulazi u multiplekser	Broj ulaza u logička kola
A, B	$D_0 = 0\ ,\ D_1 = \overline{D}\ ,\ D_2 = \overline{C}D\overline{E}\ ,\ D_3 = \overline{C}D\overline{E} \vee \overline{D}E$	9
A, C	$D_0 = D_1 = B\overline{D}\ ,\ D_2 = D\overline{E} \vee B\overline{D}E\ ,\ D_3 = B\overline{D}E$	10
A, D	$D_0 = B\ ,\ D_1 = 0\ ,\ D_2 = BE\ ,\ D_3 = \overline{C}\,\overline{E}$	6
A, E	$D_0 = D_1 = D_3 = B\overline{D}\ ,\ D_2 = \overline{C}D$	6
B, C	$D_0 = ADE\ ,\ D_1 = 0\ ,\ D_2 = \overline{A}\,\overline{D} \vee AD\overline{E} \vee \overline{D}E\ ,\ D_3 = \overline{A}\,\overline{D} \vee \overline{D}E$	14
B, D	$D_0 = 0\ ,\ D_1 = D_3 = A\overline{C}\,\overline{E}\ ,\ D_2 = \overline{A} \vee E$	8
B, E	$D_0 = A\overline{C}D\ ,\ D_1 = 0\ ,\ D_2 = \overline{A}\,\overline{D} \vee A\overline{C}D\ ,\ D_3 = \overline{D}$	10
C, D	$D_0 = D_2 = \overline{A}B \vee BE\ ,\ D_1 = A\overline{E}\ ,\ D_3 = 0$	10
C, E	$D_0 = AD \vee \overline{A}B\overline{D}\ ,\ D_1 = D_3 = B\overline{D}\ ,\ D_2 = \overline{A}B\overline{D}$	10
D, E	$D_0 = \overline{A}B\ ,\ D_1 = B\ ,\ D_2 = A\overline{C}\ ,\ D_3 = 0$	6

Primjer 14.3:

- Realizirati funkciju iz prethodnog primjera koristeći multiplekser 8/1.

Kako multiplekser 8/1 ima tri adresna ulaza, tri promjenljive možemo izabrati da budu adresne promjenljive, tako da dvije promjenljive ostaju kao preostale promjenljive. Proglasimo, na primjer, C, D i E za adresne promjenljive. Tada možemo pisati:

$$\begin{aligned} Y &= \overline{A}B\overline{D} \vee A\overline{C}D\overline{E} \vee B\overline{D}E = \overline{A}B\overline{D}(C \vee \overline{C})(E \vee \overline{E}) \vee A\overline{C}D\overline{E} \vee B\overline{D}E(C \vee \overline{C}) = \\ &= \overline{A}BC\overline{D}E \vee \overline{A}BC\overline{D}\,\overline{E} \vee \overline{A}B\overline{C}\,\overline{D}E \vee \overline{A}B\overline{C}\,\overline{D}\,\overline{E} \vee A\overline{C}D\overline{E} \vee BC\overline{D}E \vee B\overline{C}\,\overline{D}E = \\ &= (\overline{A}B \vee B) \cdot C\overline{D}E \vee \overline{A}B \cdot C\overline{D}\,\overline{E} \vee (\overline{A}B \vee B) \cdot \overline{C}\,\overline{D}E \vee \overline{A}B \cdot \overline{C}\,\overline{D}\,\overline{E} \vee A \cdot \overline{C}D\overline{E} = \\ &= B \cdot C\overline{D}E \vee \overline{A}B \cdot C\,\overline{D}\,\overline{E} \vee B \cdot \overline{C}\,\overline{D}E \vee \overline{A}B \cdot \overline{C}\,\overline{D}\,\overline{E} \vee A \cdot \overline{C}D\overline{E} \end{aligned}$$

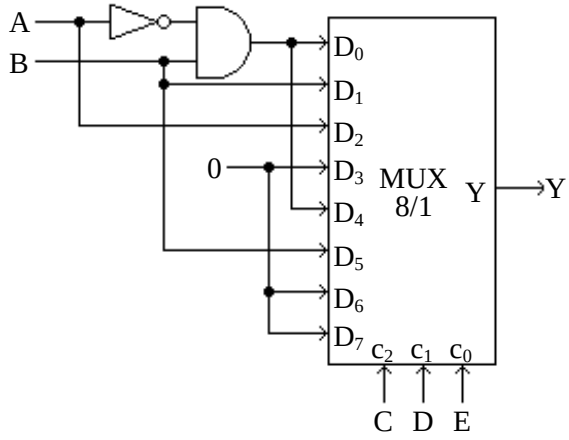
Ovako preuređenoj funkciji odgovara sljedeća tablica:

C	D	E	Y
0	0	0	$\overline{A}B$
0	0	1	B
0	1	0	A
0	1	1	0
1	0	0	$\overline{A}B$
1	0	1	B
1	1	0	0
1	1	1	0

Ukoliko ovu tablicu uporedimo sa funkcijom koja opisuje rad multipleksera 8/1, vidimo da možemo postići istu funkcionalnost ako obezbijedimo da je

$$D_0 = D_4 = \overline{A}B\ ,\quad D_1 = D_5 = B\ ,\quad D_2 = A \quad \text{i} \quad D_3 = D_6 = D_7 = 0$$

Slijedi da se traženi sklop može realizirati sljedećom shemom:



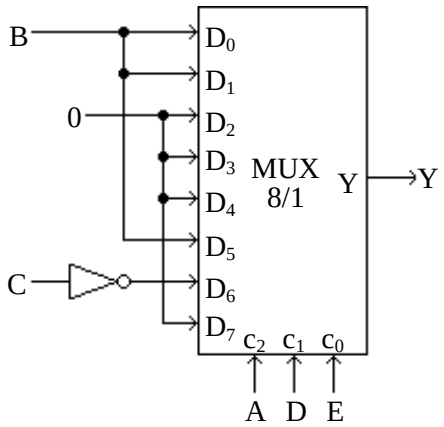
Vidimo da smo postigli veliku uštedu, jer nam je potreban samo jedan invertor i dvoulazno AND kolo. Generalno, pomoću većih multipleksera tipično se postižu veće uštede u pratećoj logici nego sa manjim multiplekserima. Međutim, i ovdje ekonomičnost realizacije bitno ovisi od izbora adresnih promjenljivih. Tako, ukoliko za adresne promjenljive izaberemo A, D i E, možemo pisati:

$$\begin{aligned} Y &= \overline{A}B\overline{D} \vee A\overline{C}D\overline{E} \vee B\overline{D}E = \overline{A}B\overline{D}(E \vee \overline{E}) \vee A\overline{C}D\overline{E} \vee B\overline{D}E(A \vee \overline{A}) = \\ &= \overline{A}B\overline{D}E \vee \overline{A}B\overline{D}\,\overline{E} \vee A\overline{C}D\overline{E} \vee AB\overline{D}E \vee \overline{A}B\overline{D}E = \\ &= B \cdot \overline{A}\,\overline{D}E \vee B \cdot \overline{A}\,\overline{D}\,\overline{E} \vee \overline{C} \cdot AD\overline{E} \vee B \cdot A\overline{D}E \end{aligned}$$

Da bismo ovu funkcionalnost postigli sa multiplekserom 8/1 uz izabrane adresne promjenljive, moramo obezbijediti da je

$$D_0 = D_1 = D_5 = B\,,\quad D_2 = D_3 = D_4 = D_7 = 0\,,\quad D_6 = \overline{C}$$

Drugim riječima, uz ovakav izbor adresnih promjenljivih, za realizaciju tražene funkcije nam osim multipleksera 8/1 treba još samo jedan invertor, kao što je prikazano na sljedećoj slici:



S druge strane, neke kombinacije adresnih promjenljivih dovode do nepovoljnih realizacija. Na primjer, ukoliko bismo za adresne promjenljive izabrali A, B i C, mogli bismo pisati sljedeći razvoj:

$$\begin{aligned}
Y &= \overline{A}B\overline{D} \vee A\overline{C}D\overline{E} \vee B\overline{D}E = \overline{A}B\overline{D}\,(C \vee \overline{C}) \vee A\overline{C}D\overline{E}\,(B \vee \overline{B}) \vee B\overline{D}E\,(A \vee \overline{A})\,(C \vee \overline{C}) = \\
&= \overline{A}BC\overline{D} \vee \overline{A}B\overline{C}\,\overline{D} \vee ABC\overline{D}\overline{E} \vee A\overline{B}\,\overline{C}D\overline{E} \vee ABC\overline{D}E \vee AB\overline{C}\,\overline{D}E \vee \overline{A}BC\overline{D}E \vee \overline{A}B\overline{C}\,\overline{D}E = \\
&= (\overline{D} \vee \overline{D}E)\cdot\overline{A}BC \vee (\overline{D} \vee \overline{D}E)\cdot\overline{A}B\overline{C} \vee (D\overline{E} \vee \overline{D}E)\cdot AB\overline{C} \vee D\overline{E}\cdot A\overline{B}\,\overline{C} \vee \overline{D}E\cdot ABC = \\
&= \overline{D}\cdot\overline{A}BC \vee \overline{D}\cdot\overline{A}B\overline{C} \vee (D\overline{E} \vee \overline{D}E)\cdot AB\overline{C} \vee D\overline{E}\cdot A\overline{B}\,\overline{C} \vee \overline{D}E\cdot ABC
\end{aligned}$$

Ovdje bismo za obezbjeđivanje tražene funkcionalnosti morali obezbijediti sljedeće jednakosti:

$$D_0 = D_1 = D_5 = 0\,,\quad D_2 = D_3 = \overline{D}\,,\quad D_4 = D\overline{E}\,,\quad D_6 = D\overline{E} \vee \overline{D}E\,,\quad D_7 = \overline{D}E$$

U ovom slučaju bi nam za realizaciju pored multipleksera 8/1 bila potrebna dva invertora, dva dvoulazna AND kola i jedno dvoulazno OR kolo, što je čak lošije nego u slučaju da smo koristili multiplekser 4/1 (uz optimalnu realizaciju). Ponovo vidimo važnost dobrog izbora adresnih promjenljivih (za čiji izbor, kao što smo već rekli, ne postoje apriorna pravila). U sljedećoj tabeli su radi poređenja rezimirani svi mogući slučajevi koji se mogu dobiti za razmatranu funkciju za sve moguće kombinacije od po tri adresne promjenljive.

Adresne promjenljive	Neophodni ulazi u multiplekser	Broj ulaza u logička kola
A, B, C	$D_0 = D_1 = D_5 = 0\,,\ D_2 = D_3 = \overline{D}\,,\ D_4 = D\overline{E}\,,\ D_6 = D\overline{E} \vee \overline{D}E\,,\ D_7 = \overline{D}E$	8
A, B, D	$D_0 = D_1 = D_3 = D_4 = 0\,,\ D_2 = 1\,,\ D_5 = D_7 = \overline{C}\,\overline{E}\,,\ D_6 = E$	4
A, B, E	$D_0 = D_1 = D_5 = 0\,,\ D_2 = D_3 = D_7 = \overline{D}\,,\ D_4 = D_6 = \overline{C}D$	4
A, C, D	$D_0 = D_2 = B\,,\ D_1 = D_3 = D_7 = 0\,,\ D_4 = D_6 = BE\,,\ D_5 = \overline{E}$	2
A, C, E	$D_0 = D_1 = D_2 = 0\,,\ D_3 = D_5 = D_6 = D_7 = B\overline{D}\,,\ D_4 = B \vee D$	5
A, D, E	$D_0 = D_1 = D_5 = B\,,\ D_2 = D_3 = D_4 = D_7 = 0\,,\ D_6 = \overline{C}$	1
B, C, D	$D_0 = D_2 = D_3 = D_7 = 0\,,\ D_1 = A\overline{E}\,,\ D_4 = E\,,\ D_5 = \overline{A} \vee \overline{E}\,,\ D_6 = \overline{A} \vee E$	8
B, C, E	$D_0 = AD\,,\ D_1 = D_2 = D_3 = 0\,,\ D_4 = \overline{A}\,\overline{D} \vee AD\,,\ D_5 = D_7 = \overline{D}\,,\ D_6 = \overline{A}\,\overline{D}$	8
B, D, E	$D_0 = D_1 = D_3 = D_7 = 0\,,\ D_2 = D_6 = A\overline{C}\,,\ D_4 = \overline{A}\,,\ D_5 = 1$	4
C, D, E	$D_0 = D_4 = \overline{A}B\,,\ D_1 = D_5 = B\,,\ D_2 = A\,,\ D_3 = D_6 = D_7 = 0$	3

Primjer 14.4:

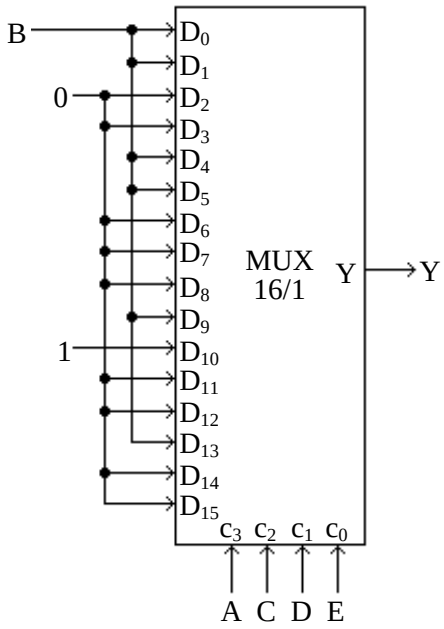
- Realizirati funkciju iz prethodnog primjera koristeći multiplekser 16/1.

Kako multiplekser 16/1 ima četiri adresna ulaza, možemo izabrati četiri promjenljive za adresne promjenljive, tako da samo jedna promjenljiva ostaje kao preostala promjenljiva. Izaberimo, na primjer, A, C, D i E kao adresne promjenljive, tako da B ostaje kao preostala promjenljiva. Tada možemo pisati:

$$\begin{aligned} Y &= \overline{A}B\overline{D} \vee A\overline{C}D\overline{E} \vee B\overline{D}E = \overline{A}B\overline{D}(C \vee \overline{C})(E \vee \overline{E}) \vee A\overline{C}D\overline{E} \vee B\overline{D}E(A \vee \overline{A})(C \vee \overline{C}) = \\ &= \overline{A}BC\overline{D}E \vee \overline{A}BCD\,\overline{E} \vee \overline{A}B\overline{C}\,\overline{D}E \vee \overline{A}B\overline{C}\,D\,\overline{E} \vee A\overline{C}D\overline{E} \vee ABC\overline{D}E \vee AB\overline{C}\,\overline{D}E \vee \overline{A}BC\overline{D}E \vee \overline{A}B\overline{C}\,\overline{D}E = \\ &= (B \vee B)\cdot\overline{A}C\overline{D}E \vee B\cdot\overline{A}CD\,\overline{E} \vee (B \vee B)\cdot\overline{A}\,C\,\overline{D}E \vee B\cdot\overline{A}\,C\,D\,\overline{E} \vee 1\cdot A\overline{C}D\overline{E} \vee B\cdot ACD\overline{E} \vee B\cdot A\overline{C}\,\overline{D}E = \\ &= B\cdot\overline{A}C\overline{D}E \vee B\cdot\overline{A}CD\,\overline{E} \vee B\cdot\overline{A}\,C\,\overline{D}E \vee B\cdot\overline{A}\,C\,D\,\overline{E} \vee 1\cdot A\overline{C}D\overline{E} \vee B\cdot ACD\overline{E} \vee B\cdot A\overline{C}\,\overline{D}E \end{aligned}$$

Jasno je da u slučaju kada je samo jedna promjenljiva preostala promjenljiva, tj. kada funkciju od *n* promjenljivih realiziramo pomoću multipleksera sa *n*–1 adresnih ulaza, tada su jedine vrijednosti koje treba eventualno dovoditi na ulaze konstante 0 i 1, preostala promjenljiva i njena negacija, tako da nam je za realizaciju funkcije pored multipleksera eventualno potreban još samo jedan invertor. Potreba za invertorom se veoma često može izbjeći dobrim odabirom adresnih promjenljivih, tako da nam je dovoljan samo multiplekser. Na sljedećoj slici prikazana je tablica koja slijedi iz gornjeg razvoja, kao i odgovarajuća realizacija, koja je vidljiva direktno iz tablice. Vidimo da nam u ovom slučaju za realizaciju funkcije uz izabrane adresne promjenljive osim multipleksera ne treba ništa drugo:

A	C	D	E	Y
0	0	0	0	B
0	0	0	1	B
0	0	1	0	0
0	0	1	1	0
0	1	0	0	B
0	1	0	1	B
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	B
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	B
1	1	1	0	0
1	1	1	1	0



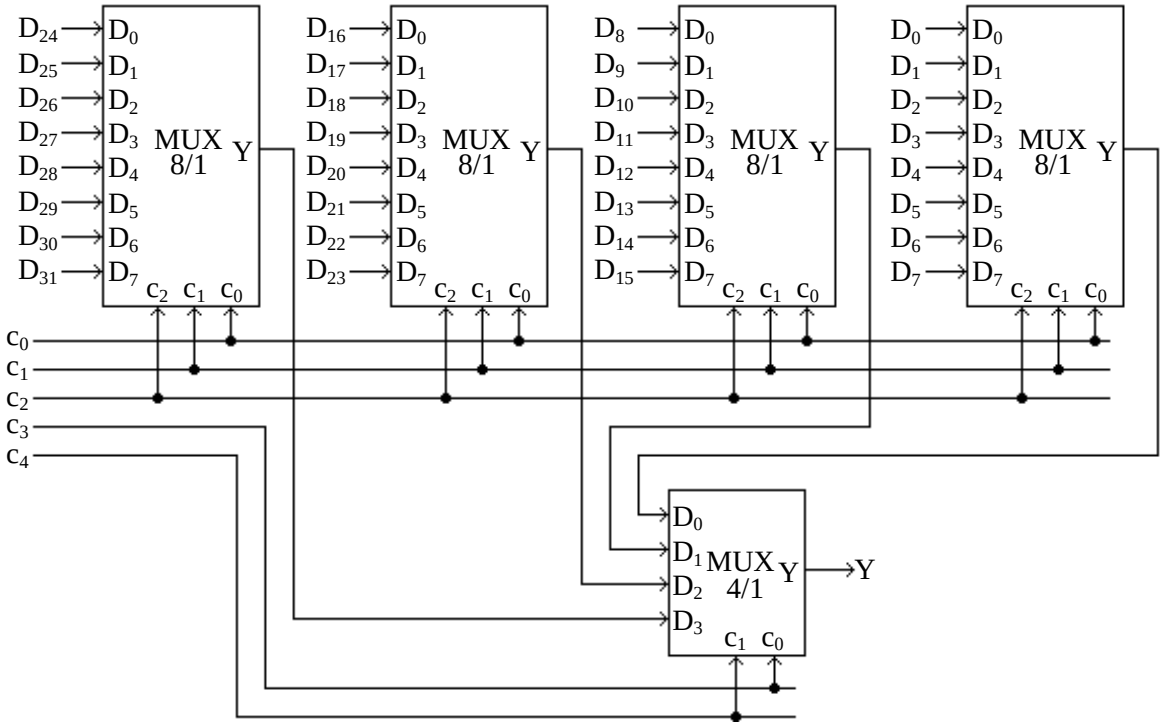
Iz izloženih primjera nije teško izvući neke opće zaključke. Za realizaciju proizvoljne funkcije od *n* promjenljivih dovoljan nam je samo multiplekser $2^n/1$, pri čemu sve promjenljive dovodimo na adresne ulaze, dok na ulaze podataka dovodimo samo nule ili jedinice, prema tablici istine tražene funkcije. Međutim, ovakva realizacija je veoma neekonomična, s obzirom da smo vidjeli sa se funkcija od *n* promjenljivih može realizirati i sa dvostruko manjim multiplekserom $2^{n-1}/1$, tako da nam pored multipleksera eventualno zatreba još samo jedan invertor, i to samo ukoliko nam zatreba negacija preostale promjenljive (što se često može izbjeći pogodnim izborom adresnih promjenljivih).

Funkcije od *n* promjenljivih mogu se realizirati i pomoću multipleksera formata $2^{n-k}/1$ gdje je $k > 1$, ali su nam tada pored multipleksera najčešće potrebna i dodatna logička kola, jer je na ulaze podataka često potrebno dovoditi razne netrivijalne logičke funkcije preostalih promjenljivih. Složenost prateće logike često se može umanjiti pogodnim izborom adresnih promjenljivih, mada do današnjeg dana nije

pronađena nikakva strategija koja bi upućivala na dobar izbor adresnih promjenljivih, osim čistog isprobavanja raznih kombinacija. Maksimalna složenost prateće logike zavisi od k, i npr. za $k = 2$ iznosi najviše petnestak ulaza u prateća logička kola, mada je u praksi najčešće znatno manja. Složenost prateće logike tipično raste sa porastom k, mada se zna desiti da su neka rješenja za veće vrijednosti k povoljnija nego neka rješenja za manje vrijednosti k. Generalno posmatrano, optimalne realizacije sa većim multiplekserima uvijek daju ekonomičniju prateću logiku nego optimalne realizacije sa manjim multiplekserima, ali ne treba zaboraviti i da složenost multipleksera osjetno raste sa porastom njegove veličine, tako da su često realizacije sa manjim multiplekserima ekonomičnije od realizacija sa većim multiplekserima (čak i ukoliko je potrebna nešto složenija prateća logika).

Iz izloženih primjera je vidljivo da multiplekseri mogu biti efikasno sredstvo za realiziranje logičkih funkcija, pogotovo onih sa većim brojem promjenljivih. Zapravo, sve do pojave PLA komponenti, multiplekseri su bili jedino rješenje koje je omogućavalo realizaciju složenijih funkcija na prihvatljivom prostoru. Pojava PLA komponenti umnogome je potisnula ogroman broj funkcionalnih rješenja u kojima su se koristili multiplekseri, mada postoje još uvijek brojne primjene u kojima su multiplekseri pogodniji od PLA komponenti.

Jedan od problema vezanih za multipleksere leži u činjenici da se broj njihovih ulaza drastično povećava sa porastom n, tako da se multiplekseri za $n > 4$ nikada ne proizvode kao samostalne komponente. Međutim, ovaj problem se može lako riješiti vezivanjem više manjih multipleksera u tzv. ***strukture tipa stabla*** (s obzirom da se multiplekseri povezuju na takav način da tvore topološku strukturu koja je u teoriji grafova poznata pod nazivom "stablo"). Naime, nije teško vidjeti da se od 2^m multipleksera tipa $2^n/1$ (ili jednog višestrukog multipleksera $2^m \times 2^n/1$, što je još pogodnije) i jednog multipleksera tipa $2^m/1$ može napraviti jedan multiplekser tipa $2^{m+n}/1$ tako što se svi adresni ulazi multipleksera $2^n/1$ spoje zajedno, a njihovi izlazi spoje na ulaze podataka multipleksera $2^m/1$. Pri tome se najnižih n adresnih bita od ukupno $m+n$ adresnih bita traženog multipleksera $2^{m+n}/1$ dovode na adresne ulaze multipleksera $2^n/1$, dok se preostalih m adresnih bita dovode na adresne ulaze multipleksera $2^m/1$. Na primjer, na sljedećoj slici prikazana je realizacija multipleksera 32/1 pomoću četiri multipleksera 8/1 i jednog multipleksera 4/1 (s obzirom da sva četiri multipleksera 8/1 imaju zajedničke adresne ulaze, realizacija preko višestrukog multipleksera 4×8/1 i multipleksera 4/1 slijedi kao trivijalna modifikacija prikazane sheme):

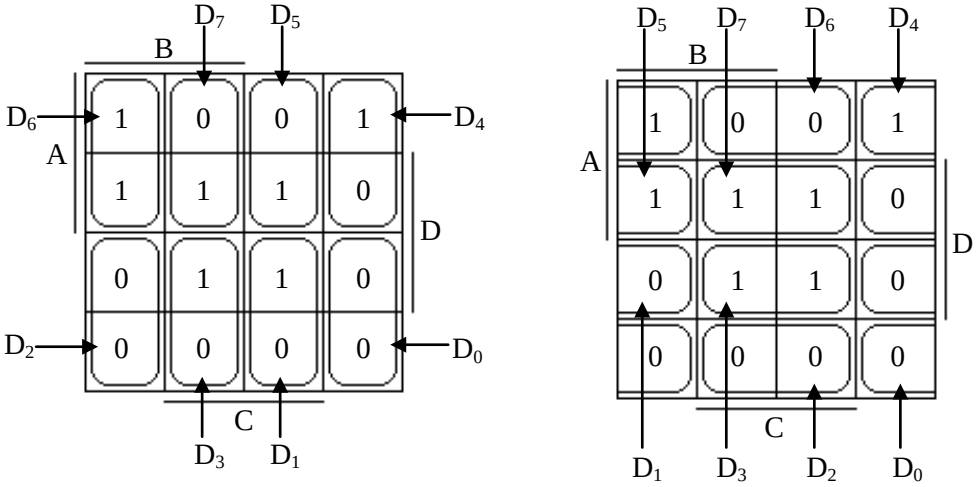


Mada smo rekli da ne postoji nikakva sistematična metoda za optimalan izbor adresnih promjenljivih, u slučaju funkcija sa manjim brojem promjenljivih ($3 \div 4$, eventualno do 6) od velike koristi mogu biti Veitchovi dijagrami. Naime, primijetimo da je izlaz iz multipleksera jednak disjunkciji produkata ulaza podataka sa svim mogućim mintermama koje se mogu formirati od promjenljivih dovedenih na adresne ulaze. Slijedi da se funkcije koje je potrebno dovesti na ulaze podataka multipleksera $2^m/1$ da bi se realizirala funkcija od $n > m$ promjenljivih mogu odrediti pomoću Veitchovih dijagrama tako što se Veitchov dijagram za zadanu funkciju podijeli na 2^m sektora od kojih svaki odgovara po jednoj mintermi formiranoj od m promjenljivih dovedenih na adresne ulaze, nakon čega se svaki od sektora posmatra kao mali Veitchov dijagram po $n–m$ preostalih promjenljivih. Funkcije koje se očitaju iz tih mini-Veitchovih dijagrama (sektora) treba dovesti na odgovarajuće ulaze podataka, u skladu sa mintermama koje opisuju posmatrane sektore. Podjela na sektore ovisi od izbora adresnih promjenljivih tako da je, uz malo vještine, moguće posmatranjem odrediti izbor adresnih promjenljivih koji će dovesti do optimalne realizacije. Iako ovo zvuči prilično komplicirano, sama procedura je dosta jednostavna, a najbolje ju je ilustrirati na konkretnom primjeru.

Primjer 14.5:

- Pronaći optimalne realizacije za logičku funkciju $Y = A\overline{C}\,\overline{D} \vee ABD \vee \overline{A}CD \vee \overline{B}CD$ pomoću multipleksera 8/1 i 4/1, korištenjem Veitchovih dijagrama.

Prvi korak je svakako predstavljanje tražene funkcije Veitchovim dijagramom, što je posve lako uraditi. Nakon toga je potrebno dobijeni dijagram izdijeliti na sektore čiji broj ovisi od izabranog multipleksera. Razmotrimo prvo realizaciju baziranu na multiplekseru 8/1. U tom slučaju, tri promjenljive moraju biti adresne a jedna preostala. Stoga je Veitchov dijagram za datu funkciju potrebno podijeliti na 8 sektora, od kojih svaki odgovara po jednoj mintermi formiranoj od tri adresne promjenljive. Na sljedećoj slici su prikazane dvije (od ukupno 4 moguće) takve podjele, zajedno sa oznakama koji sektor odgovara kojoj mintermi, odnosno kojem od osam ulaza $D_0 \div D_7$ u multiplekser odgovara koji sektor. Podjela na lijevom dijagramu izvedena je uz pretpostavku da je D preostala promjenljiva (odnosno da su A, B i C adresne promjenljive), dok je podjela na desnom dijagramu izvedena uz pretpostavku da je B preostala promjenljiva (odnosno da su A, C i D adresne promjenljive):



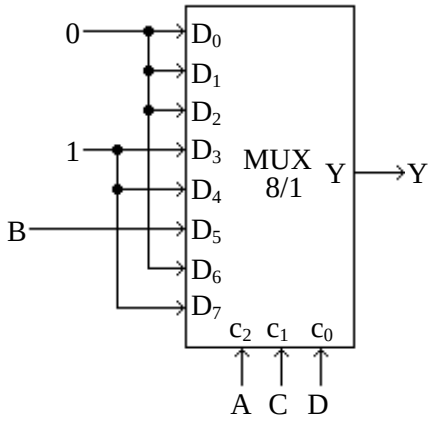
Sada je iz ovih dijagrama potrebno izvršiti odgovarajuća očitanja, posmatrajući svaki od sektora kao minijaturni Veitchov dijagram po preostaloj promjenljivoj. Tako, iz dijagrama sa lijeve strane možemo izvršiti sljedeće očitanje:

$$D_0 = 0,\ D_1 = D,\ D_2 = 0,\ D_3 = D,\ D_4 = \overline{D},\ D_5 = D,\ D_6 = 1,\ D_7 = D$$

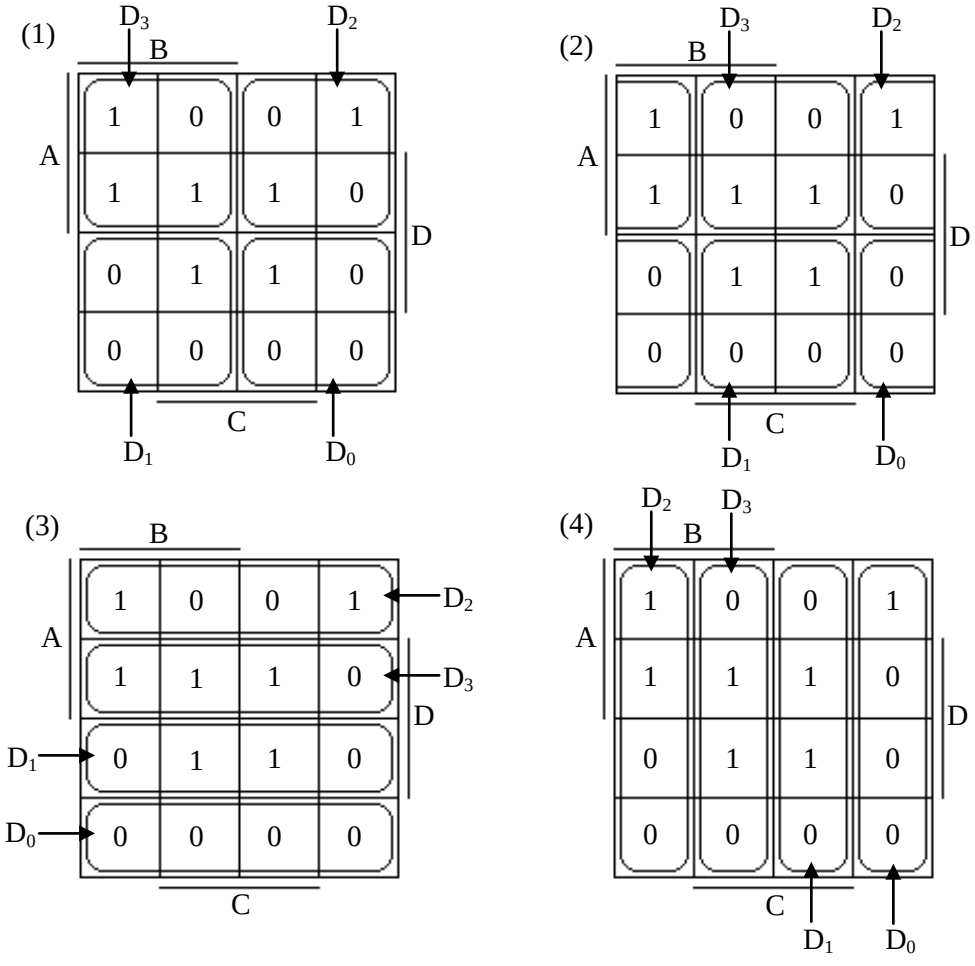
Vidimo da nam je u ovom slučaju pored multipleksera potreban i invertor. S druge strane, iz dijagrama sa desne strane možemo izvršiti sljedeće očitanje:

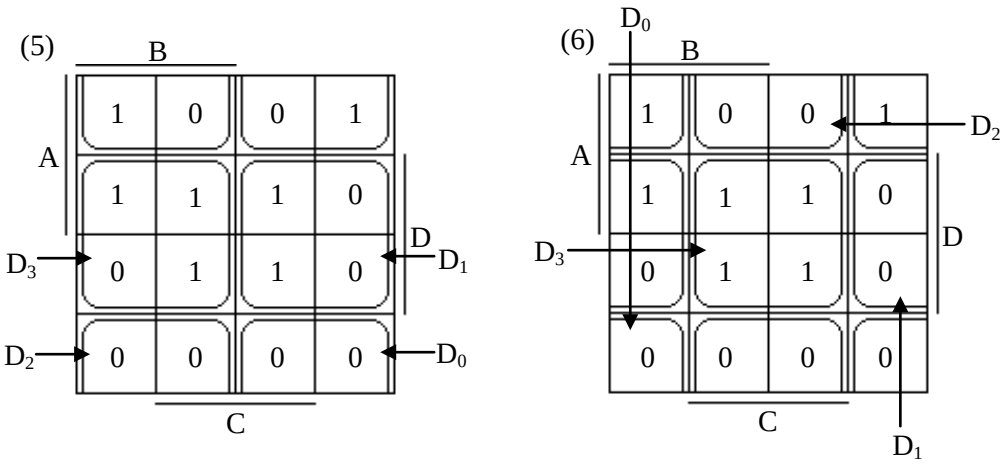
$$D_0 = 0,\ D_1 = 0,\ D_2 = 0,\ D_3 = 1,\ D_4 = 1,\ D_5 = B,\ D_6 = 0,\ D_7 = 1$$

Ovakva realizacija traži samo multiplekser i ništa više, pa je prema tome i optimalna. Slijedi da je jedna moguća optimalna realizacija preko multipleksera 8/1 predstavljena sljedećom slikom (čitatelj ili čitateljka lako mogu provjeriti da postoji još jedna optimalna realizacija koja se dobija ako izaberemo A za preostalu promjenljivu):



Razmotrimo sada moguće realizacije pomoću multipleksera 4/1. Očigledno, u ovom slučaju dvije promjenljive moraju biti adresne, a dvije ostaju kao preostale promjenljive. Stoga je Veitchov dijagram za datu funkciju potrebno podijeliti na 4 sektora, od kojih svaki odgovara po jednoj mintermi formiranoj od dvije adresne promjenljive. Zavisno od izbora adresnih promjenljivih, moguće je formirati 6 takvih podjela. Na sljedećoj slici su prikazane svih 6 mogućih podjela, zajedno sa oznakama koji sektor odgovara kojoj mintermi, odnosno kojem od četiri ulaza $D_0 \div D_3$ u multiplekser odgovara koji sektor. Dijagrami označeni na slici od (1) do (6) odgovaraju redom podjelama u kojima su za adresne promjenljive uzimani parovi promjenljivih (A, B), (A, C), (A, D), (B, C), (B, D) i (C, D):

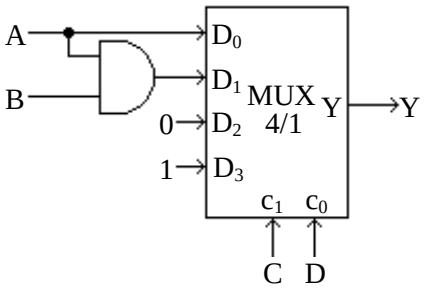




Ukoliko se svaki od sektora posmatra kao minijaturni Veitchov dijagram po preostalim promjenljivim, iz ovih dijagrama moguće je izvršiti sljedeća očitanja:

(1)	$D_0 = CD$	$D_1 = CD$	$D_2 = CD \vee \overline{C}\,\overline{D}$	$D_3 = \overline{C} \vee D$
(2)	$D_0 = 0$	$D_1 = D$	$D_2 = B \vee \overline{D}$	$D_3 = D$
(3)	$D_0 = 0$	$D_1 = C$	$D_2 = \overline{C}$	$D_3 = D$
(4)	$D_0 = A\overline{D}$	$D_1 = D$	$D_2 = D$	$D_3 = A$
(5)	$D_0 = A\overline{C}$	$D_1 = C$	$D_2 = A\overline{C}$	$D_3 = A \vee C$
(6)	$D_0 = A$	$D_1 = AB$	$D_2 = 0$	$D_3 = 1$

Odavde vidimo da je realizacija koja slijedi na osnovu podjele u dijagramu (6) najpovoljnija, jer pored multipleksera 4/1 zahtijeva još samo jedno AND kolo. Slijedi da je optimalna realizacija tražene funkcije preko multipleksera 4/1 data sljedećom shemom:

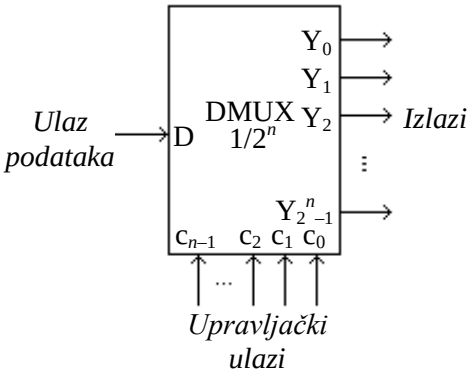


Ukoliko poredimo realizaciju tražene funkcije preko multipleksera 8/1 i 4/1, možemo zaključiti da je u konkretnom slučaju realizacija preko multipleksera 4/1 ekonomičnija bez obzira na potrebu za dodatnim AND kolom, jer je multiplekser 4/1 znatno jednostavnija komponenta od multipleksera 8/1.

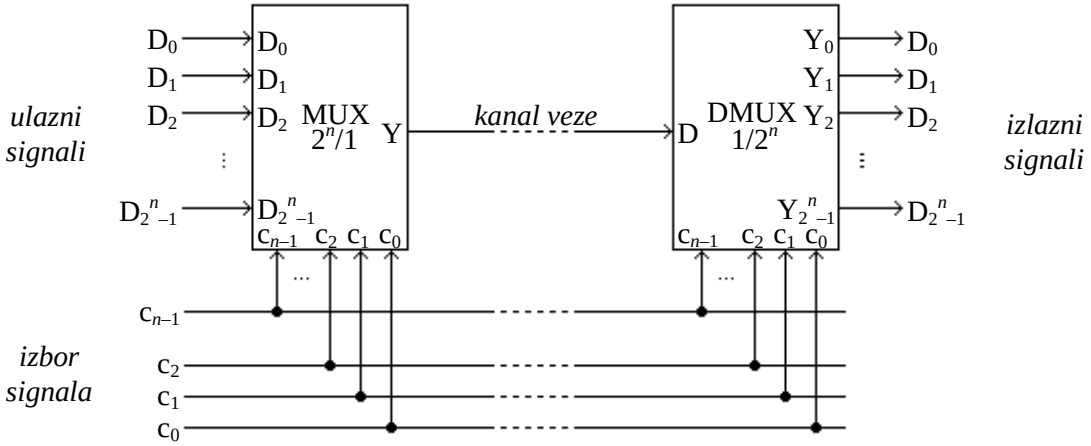
Na prvi pogled ne vidi se nikakva prednost upotrebe Veitchovih dijagrama u odnosu na analitički metod, jer je ponovo potrebno ispitati sve kombinacije adresnih promjenljivih da bi se dobila optimalna realizacija. Međutim, treba primijetiti da se sve moguće realizacije za istu funkciju očitavaju iz *istog dijagrama*, samo se razlikuje način podjele dijagrama za sektore, tako da projektant ne mora crtati podjele na sektore, već sve realizacije može prosto očitati posmatrajući nepodijeljeni dijagram (provodeći podjele "u mislima"). Pri tome, očitavanja za različite kombinacije adresnih promjenljivih ne zahtijevaju nikakvo restruktuiranje analitičkog zapisa funkcije, kao što je to slučaj kod analitičkog metoda. Vještiji projektant može čak samo pažljivo posmatrajući Veitchov dijagram prepoznati koje adresne promjenljive treba izabrati sa ciljem dobijanja optimalne realizacije. Na žalost, ovaj postupak se može provesti samo za funkcije sa malim brojem promjenljivih.

Veitchov metod omogućava i projektiranje tzv. ***kaskadnih multiplekserskih struktura***, koje često omogućavaju veoma ekonomične realizacije logičkih funkcija, pomoću više manjih multipleksera umjesto jednog većeg. Na ovakvim strukturama se nećemo zadržavati, s obzirom da se pomoću njih mogu realizirati samo izvjesne specijalne klase logičkih funkcija, kao i zbog činjenice da su ovakve tehnike projektiranja gotovo u potpunosti potisnute pojavom PLA komponenti.

Izlaganja o multiplekserima završićemo prikazom sklopa koji je inverzan multiplekseru, a poznat je pod imenom ***demultiplekser***. Ovaj sklop posjeduje 2^n izlaza i n+1 ulaz, od kojih su 2^n ulaza *adresni ulazi*, a samo jedan ulaz je *ulaz podataka*. Demultiplekser *vrijednost svog ulaza podataka prosljeđuje na jedan i samo jedan od izlaza*, i to na *onaj izlaz čiji je redni broj definiran vrijednostima upravljačkih ulaza* (tj. čiji redni broj pretvoren u binarni zapis odgovara kombinaciji bita dovedenoj na upravljačke ulaze). Pri tome se na svim ostalim izlazima pojavljuje nula, neovisno od vrijednosti ulaza podataka. Opisani demultiplekser obično se naziva "demultiplekser 1 u 2^n" ili "demultiplekser $1/2^n$" (skraćeno "DMUX $1/2^n$"), a najčešće se na shemama predstavlja sljedećim blokom:



Iz izloženog slijedi da ukoliko je ulaz podataka jednak nuli, svi izlazi iz demultipleksera su jednaki nuli, dok u slučaju da je ulaz podataka jednak jedinici, samo jedan izlaz iz demultipleksera je jednak jedinici, i to onaj čiji je redni broj definiran vrijednostima dovedenim na upravljačke ulaze. Odavde možemo zaključiti da demultiplekser zapravo nije ništa drugo nego *dekoder sa omogućavajućim ulazom*, pri čemu je omogućavajući ulaz (E) preuzeo ulogu ulaza podataka. Stoga svaki dekoder sa omogućavajućim ulazom može služiti kao demultiplekser, i obrnuto. Mada se radi o potpuno istim sklopovima, obično se dekoder sa omogućavajućim ulazom naziva demultiplekserom samo u slučajevima u kojima se on koristi u kombinaciji sa multiplekserom za *obavljanje inverzne funkcije u odnosu na multiplekser*. Tako je, na primjer, tipična primjena demultipleksera (zajedno sa multiplekserom) prikazana na sljedećoj slici:



Prikazana slika pokazuje postupak prenosa većeg broja digitalnih signala $D_0 \div D_{2^n-1}$ kroz jedinstven kanal veze (npr. na veća rastojanja). U jednom trenutku vremena prenosi se samo jedan od ukupno 2^n signala, a izbor signala koji se u datom trenutku prenosi ostvaruje se preko upravljačkih ulaza $c_0 \div c_{n-1}$. Opisana tehnika naziva se ***vremensko multipleksiranje*** (po kojoj su multiplekseri i dobili ime).

(?) Pitanja i zadaci

14.1 Objasnite šta je multiplekser i kakve su njegove primjene u računarskim sistemima.

14.2 Koliko ulaza a koliko izlaza ima multiplekser 64/1?

14.3 Nacrtajte strukturu multipleksera 4/1 realiziranog samo pomoću NAND logičkih kola, kao i samo pomoću NOR logičkih kola. Koja je od ove dvije realizacije ekonomičnija?

14.4 Objasnite kakva je veza između multipleksera i dekodera.

14.5 Može li realizacija multipleksera $2^n/1$ zasnovana na dekoderu i dodatnoj logici biti ekonomičnija od neposredne realizacije?

14.6 Objasnite kako bi se strategija "podijeli i osvoji", opisana prilikom opisa postupka projektiranja većih dekodera, mogla iskoristiti za realizaciju većih multipleksera. Posebno izvesti kakva se ušteda može postići prilikom realizacije multipleksera 64/1.

14.7 Objasnite šta je višestruki multiplekser.

14.8 Odredite koliko ukupno ulaza, a koliko izlaza ima višestruki multiplekser $4 \times 8/1$.

14.9 Nacrtajte detaljnu unutrašnju strukturu višestrukog multipleksera $2 \times 4/1$ baziranog na dekoderu, uz pretpostavku da se isti dekoder koristi u oba sastavna jednostruka multipleksera, i da je pripadni dekoder realiziran na najočigledniji mogući način.

14.10 Odredite koliki broj ulaza u logička kola (ne brojeći invertore) bi bio potreban za realizaciju višestrukog multipleksera $m \times 2^n/1$ kada bi se svih m sastavnih jednostrukih multipleksera realizirali neovisno jedni od drugih, na najočigledniji mogući način. Zatim utvrdite koliki broj ulaza u logička kola je potreban ukoliko se ovaj višestruki multiplekser realizira na bazi zajedničkog dekodera, pri čemu se dekoder realizira na najočigledniji mogući način. Uporedite ove brojeve ulaza za slučaj $n = 6$ i $m = 32$.

14.11 Ponovite analizu iz prethodnog zadatka, uz pretpostavku da se kako za realizaciju jednostrukih multipleksera, tako i za realizaciju zajedničkog dekodera koristi strategija "podijeli i osvoji". Pretpostavite pri tome da je n paran broj.

14.12 Objasnite šta su sabirnice.

14.13 Nacrtajte simbole dekodera, kodera, ROM memorije, multipleksera, višestrukog multipleksera i demultipleksera korištenjem konvencije o crtanju sabirnica.

14.14 Objasnite šta je aritmetičko logička jedinica, šta su joj ulazi i izlazi.

14.15 Nacrtajte principijelnu strukturu aritmetičko logičke jedinice koja može obavljati 4 različite opracije. Koliko ovakva aritmetičko-logička jedinica ima ulaza a koliko izlaza ukoliko su svi operandi kao i rezultat 16-bitni, a predviđen je izlazni ali ne i ulazni prenos?

14.16 Koliko ulaza a koliko izlaza ima aritmetičko-logička jedinica koja može da obavlja 32 različite operacije nad dva 8-bitna operanda A i B, ukoliko je rezultat operacije također osmobitni broj, a predviđeno je i postojanje ulaznog i izlaznog prenosa.

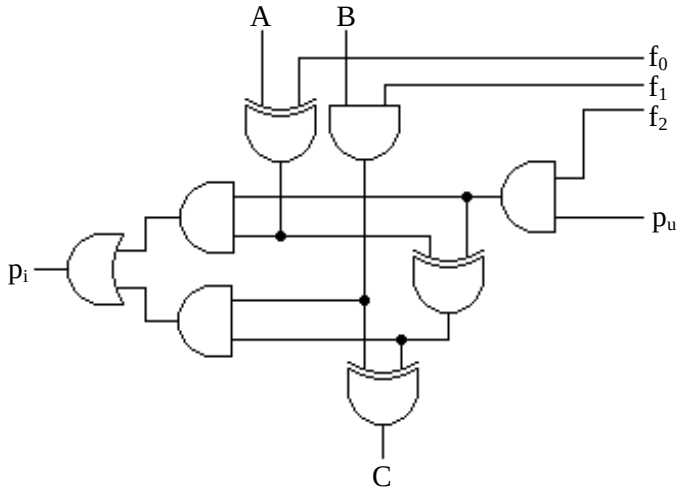
14.17 Zna se da neka aritmetičko logička jedinica (ALU) ima 22 ulaza i 9 izlaza, i da podržava ulazni i izlazni prenos. Također se zna da ova ALU koristi isti broj bita za rezultat C kao i za operande A i B. Koliko različitih operacija može izvoditi ova ALU?

14.18 Izračunajte sljedeće izraze ("&" i "|" predstavljaju generalizaciju konjukcije i disjunkcije na višebitne brojeve, koje se provode bit po bit nad individualnim bitima):
a) $33 \,\&\, 9$ b) $255 \,\&\, 113$ c) $96 \,\&\, 12$ d) $33 \,|\, 9$ e) $255 \,|\, 113$ f) $96 \,|\, 12$

14.19 Projektirajte aritmetičko-logičku jedinicu koja može obavljati 4 operacije nad dva dvobitna operanda A i B. Dvije operacije su logičke, $A \wedge B$ i $A \vee B$ (odnosno $A \& B$ i $A | B$), a dvije aritmetičke, $A + B$ i $A - B$. Za potrebe izvođenja aritmetičkih operacija, treba predvidjeti ulazni i izlazni prenos.

14.20 Projektirajte aritmetičko-logičku jedinicu koja može obavljati 3 logičke operacije $A \cdot B$, $A \vee B$ i $A \oplus B$ i jednu aritmetičku operaciju A+B. Oba operanda A i B imaju samo jedan bit, a rezultat također ima samo jedan bit. Eventualni drugi bit koji se pojavljuje kao rezultat sabiranja treba posmatrati kao izlazni prenos p_i. Također treba predvidjeti i ulaz za ulazni prenos p_u koji se koristi samo kod operacije sabiranja, i koji omogućava kaskadno vezivanje ovakvih jedinica.

14.21* Struktura aritmetičko-logičkih jedinica često može biti znatno ekonomičnija ukoliko se za njihovu realizaciju ne koriste multiplekseri (samo što to osjetno usložnjava postupak njihovog projektiranja). Ideja se zasniva na pažljivom izboru operacija koje će jedinica obavljati, i pažljivom izboru načina na koji su kodirane pojedine operacije, tako da se u što većoj mjeri iskoriste sličnosti u realizaciji pojedinih operacija. Na sljedećoj slici je prikazana jedna jednostavna 1-bitna aritmetičko-logička jedinica koja nije građena na bazi multipleksera. Analizom prikazane sheme utvrdite koje operacije podržava ova aritmetičko-logička jedinica.



14.22* Projektirajte 1-bitnu aritmetičko-logičku jedinicu, koja može da obavlja 8 operacija nad jednobitnim operandima A i B. Izbor operacije vrši se u skladu sa sljedećom tabelom:

$(f_2;f_1;f_0)_2$	$(000)_2$	$(001)_2$	$(010)_2$	$(011)_2$	$(100)_2$	$(101)_2$	$(110)_2$	$(111)_2$
C	0	$B - A$	$A - B$	$A + B$	$A \oplus B$	$A \vee B$	AB	1

Dvije trivijalne operacije ($C = 0$ i $C = 1$) uvedene su samo sa svrhom pojednostavljivanja dizajna. Realizaciju prvo izvršite na bazi multipleksera, a zatim razmotrite realizaciju u kojoj se ne koriste multiplekseri, koristeći ideje dobijene analizom prethodnog zadatka. Potrebno je predvidjeti ulazni i izlazni prenos, odnosno ulaznu i izlaznu posudbu, koji će se koristiti kod operacija sabiranja i oduzimanja.

14.23** Projektirajte 1-bitnu aritmetičko-logičku jedinicu, koja može da obavlja 16 operacija nad jednobitnim operandima A i B. Izbor operacije vrši se u skladu sa sljedećom tabelom:

$(f_3;f_2;f_1;f_0)_2$	$(0000)_2$	$(0001)_2$	$(0010)_2$	$(0011)_2$	$(0100)_2$	$(0101)_2$	$(0110)_2$	$(0111)_2$
C	0	$A \oplus B$	$\overline{A \oplus B}$	$A \vee B$	AB	$\overline{A \vee B}$	$\overline{AB}$	1
$(f_3;f_2;f_1;f_0)_2$	$(1000)_2$	$(1001)_2$	$(1010)_2$	$(1011)_2$	$(1100)_2$	$(1101)_2$	$(1110)_2$	$(1111)_2$
C	$\overline{A}$	$\overline{B}$	$A - B$	$B - A$	$A + B$	$A + 1$	B	A

I u ovom slučaju su uvedene neke trivijalne operacije, radi pojednostavljenja dizajna. Realizaciju izvršite bez upotrebe multipleksera, koristeći iskustva iz prethodna dva zadatka. Za operacije sabiranja i oduzimanja, potrebno je predvidjeti ulazni i izlazni prenos, odnosno posudbu.

14.24 Projektirajte aritmetičko-logičku jedinicu iz Zadatka 14.20 koristeći programabilnu ROM memoriju.

14.25 Potrebno je projektirati istu aritmetičko-logičku jedinicu kao u Zadatku 14.20, samo su sada operandi A i B, kao i rezultat, 4-bitni brojevi. Na raspolaganju nam stoje samo programabilne memorije. Da li se više isplati (sa aspekta utroška kapaciteta memorije) projektirati u jednoj programabilnoj memoriji traženu 4-bitnu aritmetičko-logičku jedinicu, ili vezati u kaskadu četiri 1-bitne aritmetičko-logičke jedinice projektirane u prethodnom zadatku?

14.26 Provjerite korektnost svih realizacija logičke funkcije $Y = \overline{A}B\overline{D} \vee A\overline{C}D\overline{E} \vee B\overline{D}E$ uz pomoć multipleksera 4/1, 8/1 i 16/1 koje su prikazane a nisu izvedene u primjerima 14.2, 14.3 i 14.4.

14.27 Data je logička funkcija $Y = \overline{A}BC \vee AC \vee AD$.

a) Realizirajte ovu logičku funkciju uz pomoć multipleksera 2/1 i dodatnih logičkih kola (ako zatrebaju), uzimajući A za adresnu promjenljivu.

b) Realizirajte ovu logičku funkciju uz pomoć multipleksera 4/1 i dodatnih logičkih kola (ako zatrebaju), uzimajući A i C za adresne promjenljive.

c) Realizirajte ovu logičku funkciju uz pomoć multipleksera 8/1 i invertora (ako zatreba) uzimajući A, B i D za adresne promjenljive.

14.28 Data je logička funkcija $Y = m_0 \vee m_3 \vee m_5 \vee m_7 \vee m_{11} \vee m_{12} \vee m_{13} \vee m_{15}$, pri čemu sve minterme zavise od promjenljivih A, B, C i D.

a) Realizirajte ovu logičku funkciju uz pomoć multipleksera 4/1 i dodatnih logičkih kola (ako zatrebaju), uzimajući A i B za adresne promjenljive.

b) Realizirajte ovu logičku funkciju uz pomoć multipleksera 8/1 i invertora (ako zatreba), uzimajući A, B i C za adresne promjenljive.

c) Realizirajte ovu logičku funkciju uz pomoć multipleksera 16/1.

14.29 Realizirajte logičku funkciju $Y = \overline{B}CE \vee \overline{B}D \vee B\overline{D} \vee \overline{C}D \vee A$ ukoliko je na raspolaganju multiplekser 16/1 i ništa drugo.

14.30* Data je logička funkcija (od 6 promjenljivih)

$$Y = AB\overline{C}\,\overline{D}\,\overline{E}F \vee ABCD\overline{E}F \vee A\overline{B}C\overline{D}\,\overline{E} \vee \overline{A}B\overline{C}D \vee \overline{A}BC\overline{D} \vee \overline{A}\,\overline{B}CD \vee A\overline{B}\,\overline{C}E \vee A\overline{C}D\overline{E} \vee BC\overline{D}E \vee BC\overline{D}E$$

Pokažite kako je moguće realizirati ovu logičku funkciju ukoliko nam je na raspolaganju jedan multiplekser 16/1, jedan invertor i jedno AND kolo (i ništa drugo).

14.31 Data je logička funkcija $Y = \overline{ABC \vee D} \vee \overline{\overline{ACD} \vee B}$. Realizirajte ovu funkciju preko multipleksera 8/1. Adresne promjenljive izaberite tako da se izbjegne potreba za invertorom.

14.32 Data je logička funkcija $Y = B\overline{A \oplus C} \vee D(A \oplus C)$. Koristeći Veitchove dijagrame, realizirajte ovu logičku funkciju pomoću multipleksera 4/1 uz što je god moguće manje prateće logike.

14.33 Data je logička funkcija $Y = \overline{ABC \vee D} \vee \overline{\overline{ACD} \vee B}$. Koristeći Veitchove dijagrame, realizirajte ovu logičku funkciju pomoću multipleksera 4/1 uz što je god moguće manje prateće logike.

14.34 Data je logička funkcija $Y = (\overline{A} \vee C \vee D)(A \vee \overline{C} \vee D)(\overline{A} \vee B)$. Poznato je da se na ulazu u sklop koji realizira ovu logičku funkciju nikada neće pojaviti takve vrijednosti promjenljivih za koje vrijedi $M_0 M_2 M_9 M_{10} = 0$ (sve maksterme ovise od promjenljivih A, B, C i D). Uz navedene pretpostavke, realizirajte datu logičku funkciju uz pomoć

a) multipleksera 8/1;

b) multipleksera 4/1.

c) multipleksera 2/1.

U sva tri slučaja adresne promjenljive odaberite tako da se dobije najekonomičnija realizacija.

14.35 Kao što se za realizaciju individualnih logičkih funkcija može iskoristiti multiplekser (eventualno uz nešto prateće logike), tako se za realizaciju skupine od *m* logičkih funkcija može iskoristiti *m* multipleksera, ili jedan višestruki *m*-bitni multiplekser. Objasnite u čemu je suštinska razlika ukoliko se koristi *m* običnih multipleksera u odnosu na situaciju kada se koristi jedan višestruki multiplekser. Da li se, sa obzirom na građu multipleksera, generalno posmatrano više isplati realizacija sa *m* običnih ili jednim višestrukim *m*-bitnim multiplekserom?

14.36* Dat je skup logičkih funkcija

$X = m_1 \vee m_3 \vee m_6 \vee m_7 \vee m_9 \vee m_{11} \vee m_{14} \vee m_{15}$

$Y = m_0 \vee m_1 \vee m_3 \vee m_4 \vee m_5 \vee m_8 \vee m_9 \vee m_{10} \vee m_{11} \vee m_{12} \vee m_{14}$

$Z = m_0 \vee m_1 \vee m_4 \vee m_5 \vee m_6 \vee m_7 \vee m_8 \vee m_{10} \vee m_{12} \vee m_{14} \vee m_{15}$

pri čemu sve minterme zavise od promjenljivih A, B, C i D. Realizirati ovaj skup logičkih funkcija uz pomoć

a) tri multipleksera 8/1;

b) tri multipleksera 4/1;

c) tri multipleksera 2/1.

U sva tri slučaja, adresne promjenljive individualnih multipleksera izaberite tako da se, ukupno posmatrano, dobije što je god moguće jednostavnija realizacija (tj. da eventualno neophodna prateća logika bude što jednostavnija).

14.37* Neka je dat isti skup logičkih funkcija kao u prethodnom zadatku. Realizirajte ovaj skup logičkih funkcija uz pomoć

a) višestrukog multipleksera $3 \times 8/1$;

b) višestrukog multipleksera $3 \times 4/1$;

c) višestrukog multipleksera $3 \times 2/1$.

U sva tri slučaja, adresne promjenljive izaberite tako da se dobije što je god moguće jednostavnija realizacija.

14.38 Dat je skup logičkih funkcija

$X = \overline{A}\,\overline{B}\,\overline{C} \vee \overline{A}B \vee \overline{A}C \vee \overline{A}\,\overline{D} \vee BC$

$Y = \overline{A}B\overline{C} \vee A\overline{B}D \vee BC$

$Z = ABC \vee \overline{A}\,\overline{B} \vee \overline{B}D$

Realizirajte ovaj skup logičkih funkcija ukoliko je na raspolaganju samo višestruki multiplekser $3 \times 4/1$ (i ništa drugo).

14.39 Realizirajte puni sumator uz pomoć

a) višestrukog multipleksera $2 \times 4/1$;

b) višestrukog multipleksera $2 \times 2/1$.

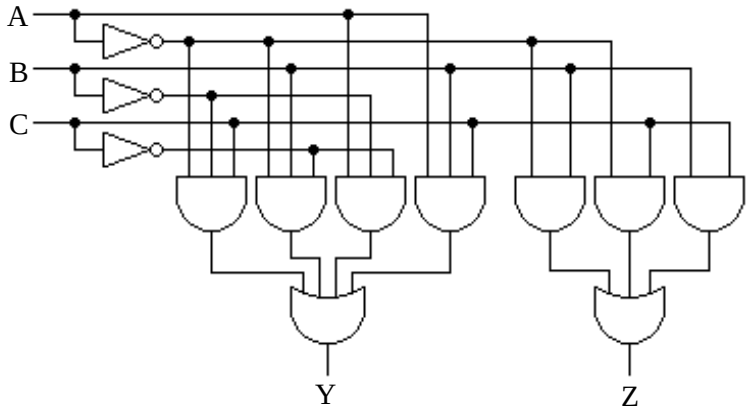
U oba slučaja, adresne promjenljive izaberite tako da se dobije što je god moguće jednostavnija realizacija.

14.40 Realizirajte sklop koji sabira dva broja A i B od po dva bita (opisan u Primjeru 11.3) uz pomoć višestrukog multipleksera $3 \times 8/1$. Adresne promjenljive izaberite tako da se izbjegne potreba za invertorom, ukoliko je to moguće.

14.41 U računarskoj tehnici mnogo se koristi sklop poznat pod nazivom ***barel-šifter***. 2^n-bitni barel-šifter ima 2^n ulaza za podatke $D_0 \div D_{2^n-1}$, n upravljačkih ulaza $c_0 \div c_{n-1}$, i 2^n izlaza $O_0 \div O_{2^n-1}$. Između ulaza i izlaza postoji zavisnost $O_i = D_{i-k}$ za $i \geq k$ i $O_i = 0$ za $i < k$, gdje je k dekadna vrijednost binarnog broja $(c_{n-1}; \ldots c_2; c_1; c_0)_2$. Drugim riječima, svi izlazi se *pomjeraju* u odnosu na ulaze za k bita, pri čemu neki biti "ispadaju", a nedostajuća mjesta se zamjenjuju nulama. Na primjer, za $n = 2$ i $k = 2$ imamo $D_0 \rightarrow O_2$, $D_1 \rightarrow O_3$, D_2 i D_3 "ispadaju", dok su O_0 i O_1 jednaki nuli. Pokažite kako se barel-šifter može realizirati pomoću višestrukog multipleksera. Posebno, realizirajte 4-bitni barel-šifter.

14.42 U nekim verzijama barel-šifterima se vrši "rotacija", odnosno biti koji "ispadaju" pojavljuju sa "suprotne strane", npr. $D_0 \to O_2$, $D_1 \to O_3$, $D_2 \to O_0$ i $D_3 \to O_1$ u slučaju 4-bitnog barel-šiftera za $k = 2$. Realizirajte 4-bitni barel-šifter "sa rotacijom".

14.43 Zaposleni ste u kompaniji koja proizvodi klonove računara poznatih proizvođača. Dobili ste zadatak da napravite funkcionalnu kopiju jedne od komponenti nekog računara, čija je shema prikazana na sljedećoj slici:



Zbog vrlo restriktivnog Zakona o zaštiti autorskih prava, Vaš šef insistira da struktura kopirane komponente ne smije biti ni nalik izvornoj komponenti (funkcionalnost, naravno, mora ostati ista). Pored toga, zet Vašeg šefa bavi se prodajom elektronskih komponenti, i spreman je da isporuči Vašoj kompaniji multipleksere po izuzetno povoljnoj cijeni. Zbog toga, Vaš šef insistira da se u kloniranom sklopu koriste samo multiplekseri 4/1, i ništa drugo. Odradite posao koji od Vas zahtijeva šef. Možete li odgonetnuti čemu zapravo služi sklop koji treba da klonirate?

14.44 Realizirajte multiplekser 8/1

a) pomoću dva multipleksera 4/1 i jednog multipleksera 2/1 (kao strukturu tipa stabla);
b) pomoću četiri multipleksera 2/1 i jednog multipleksera 4/1;
c) pomoću sedam multipleksera 2/1.

14.45 Realizirajte multiplekser 8/1

a) pomoću višestrukog multipleksera $2 \times 4/1$ i multipleksera 2/1;
b) pomoću višestrukog multipleksera $4 \times 2/1$ i multipleksera 4/1;
c) pomoću višestrukih multipleksera $4 \times 2/1$ i $2 \times 2/1$, i multipleksera 2/1.

14.46* Data je sljedeća logička funkcija, pri čemu minterme zavise od promjenljivih A, B, C, D, E i F:

$$Y = m_3 \vee m_7 \vee m_{12} \vee m_{14} \vee m_{15} \vee m_{19} \vee m_{23} \vee m_{27} \vee m_{28} \vee m_{29} \vee m_{31} \vee m_{35} \vee m_{39} \vee m_{44} \vee m_{45} \vee m_{46} \vee m_{48} \vee m_{49} \vee m_{50} \vee m_{52} \vee m_{53} \vee m_{55} \vee m_{56} \vee m_{57} \vee m_{59}$$

Realizirajte ovu logičku funkciju uz pomoć višestrukog multipleksera $4 \times 8/1$ i multipleksera 8/1.

14.47 Uporedite broj ulaza u logička kola (ne brojeći invertore) neophodan za realizaciju jednog multipleksera tipa $2^{m+n}/1$ na neposredan način sa neophodnim brojem ulaza ukoliko se isti multiplekser realizira kao struktura tipa stabla pomoću 2^m multipleksera tipa $2^n/1$ i jednog multipleksera tipa $2^m/1$, uz pretpostavku da se svi multiplekseri realiziraju na neposredan (tj. najočigledniji mogući) način.

14.48 Uporedite broj ulaza u logička kola (ne brojeći invertore) neophodan za realizaciju jednog multipleksera tipa $2^{m+n}/1$ na neposredan način sa neophodnim brojem ulaza ukoliko se isti multiplekser realizira kao struktura tipa stabla višestrukog multipleksera $2^m \times 2^n/1$ i multipleksera tipa $2^m/1$, uz pretpostavku da je višestruki multiplekser zasnovan na zajedničkom dekoderu, koji je realiziran na najočigledniji mogući način.

14.49* Pod ***kaskadnom vezom multipleksera*** smatra se takva veza multipleksera, pri kojoj izlaz iz jednog multipleksera ide na jedan od ulaza drugog multipleksera, čiji izlaz ide na jedan ud ulaza trećeg multipleksera, itd. Neke vrste logičkih funkcija (ali ne sve) se mogu veoma efikasno realizirati korištenjem kaskadno vezanih multipleksera. Neka je data logička funkcija

$Y = A\overline{B}C\overline{D} \vee A\overline{B}\,\overline{C}D \vee AB\overline{D} \vee \overline{A}\,\overline{C}$

Razmotrite kako bi se ova funkcija mogla realizirati kaskadnom vezom tri multipleksera 2/1. Pored multipleksera, za realizaciju je još eventualno dozvoljeno koristiti invertore.

14.50** Razmotrite kako bi se logička funkcija $Y = A\overline{B}\,\overline{D} \vee B\overline{D}\,\overline{E} \vee \overline{C}D \vee \overline{C}\,\overline{E}$ mogla realizirati pomoću kaskadne veze četiri multipleksera 2/1. Pored multipleksera, za realizaciju je još eventualno dozvoljeno koristiti invertore.

14.51** Razmotrite kako bi se logička funkcija $Y = \overline{A}CDE \vee ABC \vee A\overline{B}\,\overline{D} \vee \overline{A}BE \vee \overline{B}\,\overline{C}\,\overline{D} \vee \overline{B}\,\overline{D}\,\overline{E}$ mogla realizirati pomoću kaskadne veze dva multipleksera 4/1. Pored multipleksera, za realizaciju je još eventualno dozvoljeno koristiti invertore.

14.52** Data je logička funkcija $Y = A\overline{B}\,\overline{C}\,\overline{D}\,\overline{E} \vee A\overline{B}C\overline{D}E \vee AB\overline{C}E \vee ABC\overline{E} \vee \overline{A}\,\overline{B}\,\overline{D} \vee ABD$. Razmotrite kako bi se ova logička funkcija mogla realizirati kaskadnom vezom dva multipleksera 4/1. Pored multipleksera, za realizaciju je još eventualno dozvoljeno koristiti invertore.

14.53 Objasnite šta je demultiplekser, i kakve su mu primjene u računarskoj tehnici.

14.54 Nacrtajte detaljnu unutrašnju strukturu demultipleksera 1/4.

14.55 Objasnite šta je vremensko multipleksiranje.

14.56 Demultiplekser i dekoder (sa omogućavajućim ulazom) su praktično jedan te isti sklop. Da li isto vrijedi i za multiplekser i koder?

14.57 ***Višestruki demultiplekser*** (tipa $m \times 1/2^n$) je sklop koji je inverzan višestrukom multiplekseru $m \times 2^n/1$. Objasnite kako bi mogao biti građen višestruki demultiplekser. Posebno objasnite kako bi se on mogao realizirati ekonomičnije nego prostim spajanjem *m* običnih demultipleksera.

14.58 Koje logičke funkcije realizira kaskadni spoj multipleksera i demultipleksera sa zajedničkim adresnim ulazima (kao prilikom korištenja vremenskog multipleksiranja)?

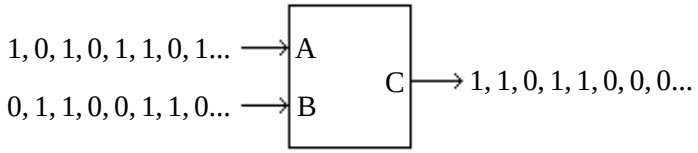
14.59* Objasnite kako se kaskadni spoj multipleksera i demultipleksera sa zajedničkim adresnim ulazima, uz nešto prateće logike može u nekim slučajevima zgodno iskoristiti za realizaciju nekih sistema logičkih funkcija. Kao primjer, realizirajte sistem funkcija iz Zadatka 14.38 uz pomoć kaskadnog spoja multipleksera 4/1 i demultipleksera 1/4.

15. Opća matematska teorija sekvencijalnih sklopova

U dosadašnjim poglavljima vidjeli smo da kombinacioni sklopovi mogu biti veoma moćni. Tako, na primjer, kombinacioni sklopovi mogu da obave bilo kakvu računsku operaciju ili matematičku funkciju nad ulaznim podacima predstavljenim u formi binarnih brojeva. Međutim, kombinacioni sklopovi su potpuno nemoćni da obave *slijed* odnosno *sekvencu* operacija jednu za drugom, pri čemu izvršavanje pojedinih operacija *zavisi od toga kako se završila jedna ili više prethodnih operacija*. Kako svaki računar posjeduje opisanu mogućnost (koja zapravo čini suštinu njegovog rada), slijedi da kombinacioni sklopovi sami za sebe *nisu dovoljni* da ostvarimo funkcionalnost koju posjeduju računari. Za tu svrhu potrebni su nam složeniji digitalni sklopovi od kombinacionih sklopova, koji se nazivaju ***sekvencijalni sklopovi*** ili ***slijedni sklopovi*** (vidjećemo kasnije da je čitav digitalni računar zapravo jedan složeni sekvencijalni sklop).

Slično kombinacionim sklopovima, sekvencijalni sklopovi također posjeduju N ulaza i M izlaza od kojih svi mogu imati vrijednosti samo 0 ili 1. Međutim, za razliku od kombinacionih sklopova, kod sekvencijalnih sklopova izlazi u nekom trenutku vremena $t = t_0$ mogu zavisiti ne samo od vrijednosti ulaza u tom istom trenutku, nego i od *ranijih vrijednosti ulaza*, tj. od vrijednosti ulaza kakve su bile u ranijim trenucima vremena $t < t_0$.

Da bismo uvidjeli potrebu za sekvencijalnim sklopovima, razmotrimo sklop poznat pod nazivom ***serijski sabirač*** (***sumator***). Ovaj sklop ima samo dva ulaza A i B, i samo jedan izlaz C. Na ulaze A i B dovode se redom (serijski) biti dva broja A i B koji se sabiraju, jedan za drugim, u unaprijed određenim vremenskim razmacima, počev od bita *najmanje težine*. Na izlazu C se pojavljuju odgovarajući biti rezultata, također jedan za drugim, u istim vremenskim razmacima. Na primjer, neka sabiramo binarne brojeve 10110101 i 01100110. Njihov zbir je 100011011, tako da bi slijed (sekvenca) ulaza i izlaza za razmotreni primjer izgledao kao na sljedećoj slici:



Opisani slijed ulaza i izlaza mogli bismo prikazati i sljedećom tablicom, koja po formi podsjeća na tablice istine:

A	B	C
1	0	1
0	1	1
1	1	0
0	0	1
1	0	1
1	1	0
0	1	0
1	0	0

Očigledno ovaj sklop *ne možemo realizirati kao kombinacioni sklop*, zbog toga što za iste kombinacije ulaza možemo dobijati različite ulaze. Na primjer, u prikazanoj tablici za A = 1 i B = 0 dva puta smo imali rezultat C = 1 (u prvom i petom redu tablice), a jedanput rezultat C = 0 (u osmom redu tablice). Iz istog razloga, ovaj sklop se *ne može opisati tablicom istine*. Vrijednosti izlaza u nekom trenutku očigledno ne zavise samo od vrijednosti ulaza u tom trenutku, nego i od ranijih vrijednosti ulaza, odnosno od čitavog slijeda (sekvence) vrijednosti koju smo dovodili na ulaze. Ovo je tipičan primjer sekvencijalnog sklopa.

Bitno je napomenuti da je matematski aparat na kojem se zasnivaju sekvencijalni sklopovi u općem slučaju veoma kompliciran, i da na mnoga pitanja do danas ne postoje zadovoljavajući odgovori. Između ostalog, sklopovi čiji izlaz zavisi ne samo od trenutnih nego i od ranijih vrijednosti ulaza ne moraju uopće biti *digitalni sklopovi* čak i ukoliko im svi ulazi i izlazi poprimaju samo vrijednosti 0 i 1! Tipičan takav slučaj ilustrira sklop sa samo jednim ulazom X (koji može poprimati samo vrijednosti 0 i 1) i samo jednim izlazom Y na kojem se u nekom trenutku vremena $t = t_0$ pojavljuje jedinica ako i samo ako je u vremenskom intervalu od nekog početnog trenutka $t = 0$ do tekućeg trenutka $t = t_0$ na ulazu X duže vremena bila jedinica nego nula. U suprotnom se na izlazu Y pojavljuje nula. Očigledno, vrijednost izlaza Y u trenutku $t = t_0$ ovisi od vrijednosti ulaza X u *svim trenucima vremena* iz intervala $(0, t_0)$, odnosno od *neprebrojivo mnogo* vrijednosti ulaza. Mada je ponašanje opisanog sklopa sasvim jasno iskazati riječima, nije sasvim lako njegov rad opisati formulom. Jedna od mogućnosti je sljedeća, ne baš jednostavna niti posve očigledna formula:

$$\mathrm{Y}(t) = \begin{cases} 1, & \text{ako je } \dfrac{1}{t}\displaystyle\int_0^t \mathrm{X}(\tau)\,\mathrm{d}\tau > \dfrac{1}{2} \\ 0, & \text{u suprotnom} \end{cases}$$

Vidimo da se čak i rad sklopa čije se funkcioniranje posve lako opisuje riječima iskazuje znatno komplikovanijom formulom u kojoj se pojavljuje *integral*, koji je sve samo ne jednostavan matematički objekat. Međutim, opisani sklop *nije digitalni sklop* (mada mu i ulaz i izlaz poprimaju samo vrijednosti 0 i 1). Naime, digitalni sklopovi obrađuju isključivo *digitalne veličine*, za koje se, kao specijalne slučajeve *diskretnih veličina* podrazumijeva da imaju dobro definirane vrijednosti samo u *diskretnim, unaprijed fiksiranim trenucima vremena*, što ovdje nije slučaj, jer vrijednost izlaza Y u trenutku $t = t_0$ zavisi od vrijednosti ulaza X u *svim trenucima vremena* iz intervala $(0, t_0)$, što znači da vrijednost ulaza X mora biti poznata u svim tačkama intervala $(0, t_0)$, što je u kontradikciji sa definicijom diskretnih veličina.

Da bismo se obezbjedili da sekvencijalni sklopovi budu digitalni sklopovi, jer su samo takvi sklopovi predmet našeg proučavanja, neznatno ćemo korigirati definiciju sekvencijalnog skopa. Stoga ćemo reći da su sekvencijalni sklopovi sklopovi sa N ulaza i M izlaza od kojih svi mogu imati vrijednosti samo 0 ili 1, kod kojih vrijednosti izlaza u nekom trenutku vremena $t = t_0$ zavise ne samo od vrijednosti ulaza u tom istom trenutku, nego i od vrijednosti ulaza kakve su bile u nekim ranijim unaprijed fiksiranim diskretnim trenucima vremena $t < t_0$.

Kako su diskretni skupovi uvijek prebrojivi, to diskretne trenutke vremena možemo numerirati *cijelim brojevima* (strogo uzevši, možemo i prirodnim brojevima, mada je numeracija cijelim brojevima pogodnija za primjene u digitalnoj tehnici), tako da možemo reći da su svi ulazi i izlazi digitalnih sklopova definirani isključivo u trenucima vremena iz skupa $\{..., t_{-2}, t_{-1}, t_0, t_1, t_2, t_3, ...\}$. Da bismo pojednostavili dalja razmatranja, umjesto konkretnog trenutka t_i možemo posmatrati samo njegov indeks i, tako da možemo smatrati da su svi ulazi i izlazi digitalnih sklopova definirani isključivo u *cjelobrojnim trenucima vremena* iz skupa $\{..., -2, -1, 0, 1, 2, 3, ...\}$. Ovakvo cjelobrojno vrijeme, koje se još naziva i ***diskretno vrijeme***, stvar je samo usvojene numeracije i ne mora nužno imati veze sa stvarnim vremenom. Stoga se ono najčešće obilježava sa n, da se ne miješa sa stvarnim (realnim) vremenom t (veza između stvarnog vremena t i diskretnog vremena n data je relacijom $t = t_n$). Ipak, u slučajevima kada su diskretni trenuci vremena ravnomjerno raspoređeni (što je u praksi najčešći slučaj), diskretno i stvarno vrijeme se podudaraju ukoliko za jedinicu mjerenja vremena odaberemo razmak T između dva susjedna diskretna trenutka, i ukoliko vremenske trenutke numeriramo u prirodnom rastućem poretku. Preciznije rečeno, tada su t i n povezani relacijom $t = n\,\mathrm{T}$. Tako, na primjer, ukoliko razmaci između diskretnih trenutaka vremena iznose $\mathrm{T} = 20\,\mathrm{ms}$, tada stvarnim trenucima vremena $t = 0\,\mathrm{ms}$, $t = 20\,\mathrm{ms}$, $t = 40\,\mathrm{ms}$, $t = 60\,\mathrm{ms}$ itd. odgovaraju diskretni trenuci vremena $n = 0$, $n = 1$, $n = 2$, $n = 3$ itd. Uočimo također da je, za razliku od stvarnog vremena, diskretno vrijeme *bezdimenzionalna veličina* (tj. nema jedinicu mjere).

Nakon ovakve konvencije možemo reći da vrijednosti izlaza sekvencijalnog sklopa u nekom diskretnom trenutku $n = n_0$, pored vrijednosti ulaza u tom istom trenutku vremena zavise i od vrijednosti ulaza u diskretnim trenucima $n = n_0 - 1$, $n = n_0 - 2$, $n = n_0 - 3$, itd. Stoga, ukoliko ulaze sekvencijalnog

sklopa označimo sa x_i ($i = 1..N$) a izlaze sa y_j ($j = 1..M$), tada je funkcioniranje ma kakvog sekvencijalnog sklopa u potpunosti definirano skupom od j jednačina

$$y_j(n) = F_j[x_1(n), x_2(n), \ldots x_N(n), x_1(n-1), x_2(n-1), \ldots x_N(n-1), x_1(n-2), x_2(n-2), \ldots x_N(n-2), ...]$$

pri čemu su F_j neke logičke funkcije, a indeks j uzima vrijednosti od 1 do M. Stvari međutim nisu ni izbliza toliko jednostavne koliko na prvi pogled izgledaju, jer vrijednosti izlaza sekvencijalnih sklopova mogu zavisiti od proizvoljno mnogo, pa čak i od beskonačno (ali prebrojivo) mnogo ranijih vrijednosti ulaza. To zapravo znači da funkcije F_j mogu zavisiti (i često zaista zavise) od *beskonačno mnogo promjenljivih*. Pred takvim funkcijama je matematski aparat koji smo do sada razvili posve nemoćan. Međutim, veoma često se beskonačnosti mogu eliminirati prevođenjem prethodne jednačine u *implicitni oblik*. Da bismo ovo pokazali, razmotrimo prvo nekoliko karakterističnih primjera iz opće teorije diskretnih sistema (nevezanih za računarsku tehniku), prije nego što pređemo na razmatranje generalnog modela za opisivanje sekvencijalnih sklopova. Posmatrajmo, na primjer, neki sef u koji se ulaže novac svaki dan i neka se u toku n-tog dana uloži količina $X(n)$. Neka je $Y(n)$ količina novca u sefu na kraju n-tog dana. Tada rad sefa možemo eksplicitno opisati formulom

$$Y(n) = X(n) + X(n-1) + X(n-2) + X(n-3) + ... = \sum_{k=0}^{\infty} X(n-k)$$

Funkcija sa desne strane, teoretski gledano zavisi od beskonačno mnogo vrijednosti $X(n)$, $X(n-1)$, $X(n-2)$, jer stanje sefa zavisi kako od današnjeg, tako i od jučerašnjeg, prekjučerašnjeg itd. uloga pa sve do u beskonačnu prošlost (barem u teoriji, jer u praksi ne možemo ići dalje od trenutka kada je sef napravljen). Međutim, ista formula može se mnogo kompaktnije prikazati u sljedećem obliku:

$$Y(n) = Y(n-1) + X(n)$$

Zaista, današnja količina novca u sefu jednaka je jučerašnjoj količini novca u sefu uvećanoj za današnji ulog. Dakle, u ovoj formuli novo stanje sefa $Y(n)$ izraženo je preko prethodnog stanja $Y(n-1)$ i trenutne vrijednosti ulaza $X(n)$. Ipak, ova formula je *implicitna*, jer se vrijednost izlaza Y nalazi kako sa lijeve, tako i sa desne strane jednakosti, tj. vrijednost izlaza se referira *na samu sebe*.

Nekada je vrijednost izlaza moguće izraziti ne preko jedne, nego preko nekoliko ali konačno mnogo prethodnih vrijednosti izlaza, i trenutne vrijednosti ulaza. Razmotrimo, kao primjer, jedan jednostavan primjer iz ekonomije. Neka je $Y(n)$ nacionalni bruto dohodak na kraju n-te godine. Ovaj dohodak možemo iskazati kao

$$Y(n) = P(n) + I(n) + X(n)$$

gdje su $P(n)$ iznos koji su potrošači utrošili na kupovinu raznih dobara, $I(n)$ investicije za kupovinu proizvodnih sredstava, i $X(n)$ ostala (neplanirana) ulaganja koja formiraju nacionalni bruto dohodak. Praksa pokazuje da su troškovi dobara za život približno proporcionalni bruto nacionalnom dohotku u toku prethodne godine, tako da možemo pisati

$$P(n) = \alpha\, Y(n-1)$$

gdje je α koeficijent proporcionalnosti. Isto tako, investicije $I(n)$ su približno proporcionalne prirastu potrošnje u proteklom periodu, tj. razlici $P(n) - P(n-1)$, tako da možemo pisati:

$$I(n) = \beta\,[P(n) - P(n-1)] = \alpha\,\beta\,[Y(n-1) - Y(n-2)]$$

Ovdje je β koeficijent proporcionalnosti. Stoga, na osnovu polazne jednačine imamo

$$Y(n) = P(n) + I(n) + X(n) = \alpha\, Y(n-1) + \alpha\,\beta\,[Y(n-1) - Y(n-2)] + X(n)$$

odnosno

$$Y(n) = \alpha\,(\beta+1)\,Y(n-1) - \alpha\,\beta\,Y(n-2) + X(n)$$

Drugim riječima, bruto nacionalni dohodak $Y(n)$ u toku godine može se približno iskazati preko bruto nacionalnih dohodaka $Y(n-1)$ i$Y(n-2)$ u toku prošle i pretprošle godine, kao i od neplaniranih ulaganja $X(n)$ u toku tekuće godine. U ovoj jednačini, oslobodili smo se zavisnosti od planiranih ulaganja $P(n)$ i $I(n)$ (koja, s obzirom da su planirana, ne predstavljaju neovisne promjenljive).

Generaliziramo li prethodna dva primjera, možemo zaključiti da se ponašanje mnogih diskretnih modela umjesto eksplicitnom jednačinom oblika

$$Y(n) = F[X(n), X(n-1), X(n-2), X(n-3), ...]$$

gdje je F neka funkcija koja može zavisiti i od beskonačno mnogo promjenljivih, može predstaviti implicitnom jednačinom oblika

$$Y(n) = \Phi[X(n), Y(n-1), Y(n-2), ... Y(n-k)]$$

pri čemu funkcija Φ zavisi samo od *konačno mnogo promjenljivih*. Jednačine ovog oblika nazivaju se ***diferentne jednačine*** (koriste se i izrazi ***rekurentne jednačine*** ili ***rekurence***), pri čemu se broj k naziva ***red diferentne jednačine***. Prethodna dva primjera (model sefa i i model nacionalnog bruto dohotka) opisivali su se diferentnim jednačinama prvog i drugog reda respektivno. Prevođenje eksplicitne jednačine u diferentnu jednačinu može biti veoma mučno i čak nije uvijek ni izvodljivo. Međutim, za gotovo sve diskretne modele koji su značajni u praksi, sasvim je lako na osnovu verbalnog opisa modela sastaviti diferentnu jednačinu koja ga opisuje, često i mnogo lakše nego eksplicitnu jednačinu.

Diferentne jednačine reda višeg od 1 uvijek je moguće svesti na skup više diferentnih jednačina prvog reda uvođenjem novih promjenljivih. Na primjer, ukoliko u prethodnu jednačinu uvedemo nove promjenljive Y_1, Y_2, ... Y_{k-1} pomoću relacija $Y_1(n) = Y(n-1)$, $Y_2(n) = Y(n-2)$, ... $Y_{k-1}(n) = Y(n-k+1)$, tada prethodnu diferentnu jednačinu k-tog reda možemo zamijeniti sljedećim sistemom diferentnih jednačina, od kojih je svaka prvog reda:

$$Y(n) = \Phi[X(n), Y(n-1), Y_1(n-1), ... Y_{k-1}(n-1)]$$
$$Y_1(n) = Y(n-1)$$
$$Y_2(n) = Y_1(n-1)$$
$$...$$
$$Y_{k-1}(n) = Y_{k-2}(n-1)$$

Na primjer, diferentnu jednačinu drugog reda koja opisuje model bruto nacionalnog dohotka, uvođenjem pomoćne promjenljive $Y_1(n) = Y(n)$ možemo zamijeniti sljedećim skupom od dvije diferentne jednačine prvog reda:

$$Y(n) = \alpha\,(\beta+1)\,Y(n-1) - \alpha\,\beta\,Y_1(n-1) + X(n)$$
$$Y_1(n) = Y(n-1)$$

U mnogim diskretnim modelima je čest slučaj da nije moguće direktno povezati ulaz i izlaz pomoću diferentne jednačine, ali je moguće uvesti konačno mnogo novih pomoćnih promjenljivih, tzv. ***promjenljivih stanja***, koje je moguće povezati sa ulazom preko diferentne jednačine, dok se izlaz može izraziti preko trenutne vrijednosti ulaza i trenutnih vrijednosti promjenljivih stanja. Mada ove promjenljive ne moraju imati nikakav konkretan smisao, često sam verbalni opis modela nameće koje bi se veličine mogle izabrati kao promjenljive stanja. Drugim riječima, moguće je model opisati sljedećim skupom jednačina:

$$S_k(n) = \Phi_k[X(n), S_1(n–1), S_2(n–1), ... S_P(n–1)], \quad k = 1 .. P$$

$$Y(n) = \Psi[X(n), S_1(n), S_2(n), ... S_P(n)]$$

Ovdje su S_1, S_2, ... S_P promjenljive stanja. Bez umanjenja opštosti pretpostavili smo da su sve diferentne jednačine prvog reda, jer smo vidjeli da to uvijek možemo postići uvođenjem pomoćnih promjenljivih (koje također možemo ubrojati u promjenljive stanja).

Svi razmotreni primjeri imali su samo jedan ulaz i jedan izlaz. Nije teško generalizirati prethodna razmatranja za diskretne modele sa više ulaza X_i, $i = 1 .. N$ i više izlaza Y_j, $j = 1 .. M$. Tako možemo reći da se većina takvih diskretnih modela može opisati sistemom jednačina oblika

$$S_k(n) = \Phi_k[X_1(n), X_2(n), ... X_N(n), S_1(n–1), S_2(n–1), ... S_P(n–1)], \quad k = 1 .. P$$

$$Y_j(n) = \Psi_j[X_1(n), X_2(n), ... X_N(n), S_1(n), S_2(n), ... S_P(n)], \quad j = 1 .. M$$

Funkcije Φ_k, $k = 1 .. P$ nazivaju se ***funkcije prelaza***, i one opisuju kako trenutne vrijednosti promjenljivih stanja zavise od trenutnih vrijednosti ulaza i prethodnih vrijednosti promjenljivih stanja. Funkcije Ψ_j, $j = 1 .. M$ nazivaju se ***funkcije izlaza***, i opisuju kako trenutne vrijednosti izlaza zavise od trenutnih vrijednosti ulaza i trenutnih vrijednosti promjenljivih stanja. Opis diskretnih modela predstavljen gornjim skupom jednačina naziva se ***opis u prostoru stanja***, dok se skup svih promjenljivih stanja naziva ***stanje sistema***. Mada teoretski gledano postoje diskretni modeli koji se ne mogu opisati u prostoru stanja, svi diskretni modeli koji imaju ikakav praktični značaj dopuštaju ovakav opis.

Mada na prvi pogled izgleda da je sam koncept stanja uveden na vještački način, on je zapravo posve prirodan. Naime, umjesto da posmatramo zavisnost izlaza od čitave historije ulaza, prirodno je pretpostaviti da se sistem u svakom trenutku vremena nalazi *u nekom stanju*, i da njegova reakcija na ulaze ne ovisi samo od ulaza, nego i od *stanja u kojem se zatekao sistem*. Na primjer, ukoliko čovjeka posmatramo kao sistem (što je u načelu barem konceptualno moguće, mada matematski opis takvog sistema prevazilazi mogućnosti današnje nauke), on neće uvijek na isti događaj (tj. na isti *ulaz*) reagirati na isti način, nego će reakcija ovisiti od toga da li je on u tom trenutku veseo, tužan, ljut, itd. (tj. od njegovog trenutnog *stanja*). Dalje, trenutno stanje sistema zavisi od njegovog *prethodnog stanja* i od *trenutnog ulaza*. Na primjer, ukoliko čovjeku koji je tužan saopćimo neku veselu vijest, njegovo raspoloženje (tj. stanje) će se sigurno promijeniti nabolje, ali vjerovatno ne u onolikoj mjeri kao u slučaju da prethodno nije bio tužan (osim ukoliko je vijest takva da u poptunosti anulira razloge za njegovu tugu). U svakom slučaju, trenutno stanje će ovisiti od prethodnog stanja i od trenutnog ulaza. U ovom primjeru smo malo pojednostavili slučaj, pretpostavljajući da je stanje zapravo raspoloženje. Raspoloženje je, u stvari, samo jedna od *promjenljivih stanja* koja opisuje čovjeka kao sistem (druge promjenljive stanja mogle bi biti npr. iznos krvnog pritiska, brzina otkucaja srca, i mnoge druge).

Vratimo se sada na digitalne sekvencijalne sklopove, kod kojih svi ulazi x_i, $i = 1 .. N$ i izlazi y_j, $j = 1 .. M$ mogu uzimati samo vrijednosti 0 i 1. Može se pokazati da je digitalne sekvencijalne sklopove *uvijek moguće* opisati u prostoru stanja, i to čak pomoću samo jedne pomoćne promjenljive S (koju ćemo zvati prosto *stanje*), pod uvjetom da dopustimo da ta promjenljiva uzima proizvoljne realne vrijednosti. Dakle, rad *svakog* digitalnog sekvencijalnog sklopa sa N ulaza i M izlaza moguće je opisati sistemom jednačina oblika

$$S(n) = \Phi[x_1(n), x_2(n), ... x_N(n), S(n–1)]$$

$$y_j(n) = \Psi_j[x_1(n), x_2(n), ... x_N(n), S(n)], \quad j = 1 .. M$$

Primijetimo da funkcije Φ i Ψ_j, $j = 1 .. M$ *nisu logičke funkcije*, jer zavise i od promjenljive S, koja može uzimati i vrijednosti različite od 0 i 1. Na ovom mjestu nećemo dokazivati da je ovakav opis uvijek moguć, jer se dokaz zasniva na posve formalnoj transformaciji jednačine oblika

$$y_j(n) = F_j[x_1(n), x_2(n), \ldots x_N(n), x_1(n–1), x_2(n –1), \ldots x_N(n–1), x_1(n–2), x_2(n–2), \ldots x_N(n–2), ...]$$

u gornji sistem jednačina, pri čemu se promjenljiva S uvodi na veoma vještački način (pomoću beskonačnih konvergentnih redova), nakon čega se dobijaju posve rogobatni izrazi za funkcije Φ i Ψ_j od kojih nema nikakve praktične koristi. Dokaz gore navedene činjenice je značajan jedino zbog toga što ukazuje da je opis digitalnih sekvencijalnih sklopova u prostoru stanja *uvijek moguć*, barem teoretski.

Za praksu su naročito bitni digitalni sekvencijalni sklopovi čiji se opis u prostoru stanja može formirati tako da stanje S može uzimati samo *konačno mnogo* (recimo *p*) *različitih vrijednosti*. Matematski modeli ovakvih sekvencijalnih sklopova nazivaju se ***konačni automati*** (ponekad se i sami takvi sekvencijalni sklopovi a ne samo njihovi matematski modeli također nazivaju konačni automati). Kako je svaki konačan skup moguće numerirati binarnim brojevima sa P bita, gdje je $2^P \geq p$, to je promjenljivu stanja S uvijek moguće zamijeniti sa P drugih promjenljivih stanja Q_1, Q_2, ... Q_P, nazvanih ***biti stanja***, od kojih svaka može uzimati samo vrijednosti 0 i 1. Nakon izvršene zamjene, slijedi da se rad svakog konačnog automata može opisati sljedećim skupom jednačina:

$$Q_k(n) = \Phi_k[x_1(n), x_2(n), \ldots x_N(n), Q_1(n-1), Q_2(n-1), \ldots Q_P(n-1)], \qquad k = 1 \,..\, P$$

$$y_j(n) = \Psi_j[x_1(n), x_2(n), \ldots x_N(n), Q_1(n), Q_2(n), \ldots Q_P(n)], \qquad j = 1 \,..\, M$$

Kako u ovom slučaju sve promjenljive od kojih zavise funkcije Φ_k i Ψ_j mogu uzmimati samo vrijednosti 0 i 1, i kako ove funkcije mogu davati samo vrijednosti 0 i 1, slijedi da su funkcije Φ_k i Ψ_j *logičke funkcije*. Dakle, za njihovo formiranje i manipulaciju sa njima moguće je koristiti pravila i zakone logičke algebre. Slijedi da se konačni automati u prostoru stanja mogu opisati jednačinama u kojima se koriste *samo logičke funkcije*. Primijetimo ipak da su jednačine u kojima se javljaju funkcije Φ_k *implicitne*, odnosno one predstavljaju *logičke diferentne jednačine*.

Neophodno je napomenuti da se u nekim drugim oblastima računarskih nauka (npr. u *teoriji jezika i automata, teoriji izračunljivosti* itd.) konačni automati definiraju na način koji naizgled nema nikakve sličnosti sa definicijom konačnog automata koju smo ovdje uveli. Ipak, bez obzira na sve različitosti, radi se o *praktično ekvivalentnim definicijama*, odnosno samo naizgled različitim a suštinski istim (odnosno *izomorfnim*) matematskim modelima za opisivanje jedne te iste stvari.

U opisanom modelu rada sekvencijalnih sklopova, *trenutne vrijednosti* promjenljivih stanja (odnosno bita stanja) izražavaju se u zavisnosti od *trenutnih vrijednosti* ulaza i *prethodnih vrijednosti* promjenljivih stanja. Međutim, praktično svi metodi za projektiranje sekvencijalnih sklopova koje ćemo kasnije razmatrati zahtijevaju opis pomoću nešto drugačijeg modela, u kojem se *buduće vrijednosti* promjenljivih stanja izražavaju preko *trenutnih vrijednosti* ulaza i promjenljivih stanja, dok se zavisnost izlaza od ulaza i promjenljivih stanja izražava na isti način kao i do sada. Drugim riječima, umjesto opisanih jednačina, koriste se jednačine oblika

$$Q_k(n+1) = \Phi_k[x_1(n), x_2(n), \ldots x_N(n), Q_1(n), Q_2(n), \ldots Q_P(n)], \qquad k = 1 \,..\, P$$

$$y_j(n) = \Psi_j[x_1(n), x_2(n), \ldots x_N(n), Q_1(n), Q_2(n), \ldots Q_P(n)], \qquad j = 1 \,..\, M$$

Konačni automati opisani ovakvim skupom jednačina nazivaju se ***konačni automati prve vrste***, za razliku od konačnih automata koji se opisuju ranije navedenim skupom jednačina, koje nazivamo ***konačni automati druge vrste***. Kako se gotovo sve metode projektiranja sekvencijalnih sklopova zasnivaju na modelima konačnih automata prve vrste, veoma je značajna sljedeća teorema:

Teorema 15.1:

Svaki konačni automat druge vrste može se modelirati kao konačni automat prve vrste.

Ovu teoremu nije teško dokazati. Njen dokaz je značajan zbog toga što ujedno ilustrira praktičan postupak za pretvaranje automata druge vrste u automate prve vrste. Naime, neka je dat opis konačnog automata druge vrste pomoću skupa jednačina

$$Q_k(n) = \Phi_k[x_1(n), x_2(n), \ldots x_N(n), Q_1(n-1), Q_2(n-1), \ldots Q_P(n-1)], \quad k = 1 \ldots P$$

$$y_j(n) = \Psi_j[x_1(n), x_2(n), \ldots x_N(n), Q_1(n), Q_2(n), \ldots Q_P(n)], \quad j = 1 \ldots M$$

Uvedemo li umjesto promjenljivih stanja $Q_k(n)$ nove promjenljive stanja $Q'_k(n)$ koje su sa starim promjenljivim stanja $Q_k(n)$ povezane relacijom $Q'_k(n) = Q_k(n-1)$, gornje jednačine postaju:

$$Q'_k(n+1) = \Phi_k[x_1(n), x_2(n), \ldots x_N(n), Q'_1(n), Q'_2(n), \ldots Q'_P(n)], \quad k = 1 \ldots P$$

$$y_j(n) = \Psi_j[x_1(n), x_2(n), \ldots x_N(n), Q'_1(n+1), Q'_2(n+1), \ldots Q'_P(n+1)], \quad j = 1 \ldots M$$

jer je $Q_k(n) = Q'_k(n+1)$. Uvrstimo li sada prvu skupinu jednačina u drugu, dobijamo:

$$Q'_k(n+1) = \Phi_k[x_1(n), x_2(n), \ldots x_N(n), Q'_1(n), Q'_2(n), \ldots Q'_P(n)], \quad k = 1 \ldots P$$

$$\begin{aligned} y_j(n) = \Psi_j[&x_1(n), x_2(n), \ldots x_N(n), \Phi_1[x_1(n), x_2(n), \ldots x_N(n), Q'_1(n), Q'_2(n), \ldots Q'_P(n)], \\ &\Phi_2[x_1(n), x_2(n), \ldots x_N(n), Q'_1(n), Q'_2(n), \ldots Q'_P(n)], \ldots \\ &\Phi_P[x_1(n), x_2(n), \ldots x_N(n), Q'_1(n), Q'_2(n), \ldots Q'_P(n)]], \quad j = 1 \ldots M \end{aligned}$$

Kako desne strane druge skupine jednačina zavise samo od promjenljivih $x_i(n)$, $i = 1 \ldots N$ i $Q'_k(n)$, $k = 1 \ldots P$, možemo ih napisati kao neke nove funkcije Ψ'_j koje zavise od tih promjenljivih, tako da možemo pisati

$$Q'_k(n+1) = \Phi_k[x_1(n), x_2(n), \ldots x_N(n), Q'_1(n), Q'_2(n), \ldots Q'_P(n)], \quad k = 1 \ldots P$$

$$y_j(n) = \Psi'_j[x_1(n), x_2(n), \ldots x_N(n), Q'_1(n), Q'_2(n), \ldots Q'_P(n)], \quad j = 1 \ldots M$$

Ovo nije ništa drugo nego matematski model konačnog automata prve vrste, što je i trebalo dokazati. Treba napomenuti da obrat ove teoreme *ne vrijedi*, odnosno postoje konačni automati prve vrste koji se ne mogu modelirati kao automati druge vrste. Drugim riječima, automati prve vrste su nešto generalniji.

Primjer 15.1:

- Dat je sekvencijalni sklop sa tri ulaza x_1, x_2 i x_3 i jednim izlazom y modeliran sljedećim skupom jednačina kao automat druge vrste:

$$Q_1(n) = x_1(n)\,Q_2(n-1) \vee \overline{x_2(n)}\,Q_1(n-1) \vee \overline{x_1(n)}\,\overline{x_3(n)}$$

$$Q_2(n) = x_3(n)\,\overline{Q_1(n-1)}\,\overline{Q_2(n-1)}$$

$$y(n) = x_1(n)\,Q_1(n)\,\overline{Q_2(n)} \vee \overline{x_3(n)}\,Q_2(n)$$

Prevesti ovaj model u model automata prve vrste.

Uvedemo li nove promjenljive stanja Q'_1 i Q'_2 preko relacija $Q'_1(n) = Q_1(n-1)$ i $Q'_2(n) = Q_2(n-1)$, gornje jednačine postaju:

$$Q'_1(n+1) = x_1(n)\,Q'_2(n) \vee \overline{x_2(n)}\,Q'_1(n) \vee \overline{x_1(n)}\,\overline{x_3(n)}$$

$$Q'_2(n+1) = x_3(n)\,\overline{Q'_1(n)}\,\overline{Q'_2(n)}$$

$$y(n) = x_1(n)\,Q'_1(n+1)\,\overline{Q'_2(n+1)} \vee \overline{x_3(n)}\,Q'_2(n+1)$$

Uvrstimo sada prve dvije jednačine u posljednju:

$$Q'_1(n+1) = x_1(n)\,Q'_2(n) \vee \overline{x_2(n)}\,Q'_1(n) \vee \overline{x_1(n)}\,\overline{x_3(n)}$$

$$Q'_2(n+1) = x_3(n)\,\overline{Q'_1(n)}\,\overline{Q'_2(n)}$$

$$y(n) = x_1(n)[x_1(n)\,Q'_2(n) \vee \overline{x_2(n)}\,Q'_1(n) \vee \overline{x_1(n)}\,\overline{x_3(n)}]\,\overline{x_3(n)\,\overline{Q'_1(n)}\,\overline{Q'_2(n)}} \vee$$
$$\vee \overline{x_3(n)}[x_3(n)\,\overline{Q'_1(n)}\,\overline{Q'_2(n)}]$$

U principu, već ovo što smo dobili predstavlja model automata prve vrste. Međutim, posljednja funkcija kojom se izražava izlaz $y(n)$ može se osjetno pojednostaviti. Izvršimo prvo njeno svođenje na disjunktivnu normalnu formu:

$$\begin{aligned}
y(n) &= x_1(n)[x_1(n)\,Q'_2(n) \vee \overline{x_2(n)}\,Q'_1(n) \vee \overline{x_1(n)}\,\overline{x_3(n)}]\,\overline{x_3(n)\,\overline{Q'_1(n)}\,\overline{Q'_2(n)}} \vee \\
&\quad \vee \overline{x_3(n)}[x_3(n)\,\overline{Q'_1(n)}\,\overline{Q'_2(n)}] = \\
&= [x_1(n)\,Q'_2(n) \vee x_1(n)\,\overline{x_2(n)}\,Q'_1(n)]\,\overline{x_3(n)\,\overline{Q'_1(n)}\,\overline{Q'_2(n)}} = \\
&= [x_1(n)\,Q'_2(n) \vee x_1(n)\,\overline{x_2(n)}\,Q'_1(n)][\overline{x_3(n)} \vee Q'_1(n) \vee Q'_2(n)] = \\
&= x_1(n)\,\overline{x_3(n)}\,Q'_2(n) \vee x_1(n)\,\overline{x_2(n)}\,\overline{x_3(n)}\,Q'_1(n) \vee x_1(n)\,Q'_1(n)\,Q'_2(n) \vee x_1(n)\,\overline{x_2(n)}\,Q'_1(n) \vee \\
&\quad \vee x_1(n)\,Q'_2(n) \vee x_1(n)\,\overline{x_2(n)}\,Q'_1(n)\,Q'_2(n)] = \\
&= x_1(n)\,\overline{x_2(n)}\,Q'_1(n) \vee x_1(n)\,Q'_2(n)
\end{aligned}$$

U posljednjem redu smo izvršili nekoliko očiglednih apsorbovanja. Lako se vidi da se dobijena DNF ne može dalje skratiti, te da je jedina eventualno moguća dodatna optimizacija izvlačenje promjenljive $x_1(n)$ ispred zagrade. Tako konačno dobijamo traženi model automata prve vrste:

$$Q'_1(n+1) = x_1(n)\,Q'_2(n) \vee \overline{x_2(n)}\,Q'_1(n) \vee \overline{x_1(n)}\,\overline{x_3(n)}$$

$$Q'_2(n+1) = x_3(n)\,\overline{Q'_1(n)}\,\overline{Q'_2(n)}$$

$$y(n) = x_1(n)[\overline{x_2(n)}\,Q'_1(n) \vee Q'_2(n)]$$

U nastavku ćemo razmatrati samo automate prve vrste, ne samo zbog toga što su generalniji, nego pretežno zbog činjenice da se većina verbalnih opisa rada nekog sekvencijalnog sklopa mnogo lakše prevode u matematski model automata prve vrste nego u matematski model automata druge vrste (kao što ćemo vidjeti u narednim poglavljima). Vrijedi još napomenuti da se jedan te isti sekvencijalni sklop može matematski modelirati na beskonačno mnogo različitih načina (interesantno je napomenuti da je problem utvrđivanja da li dva različita modela opisuju isti sekvencijalni sklop sam po sebi vrlo težak). Međutim, od tog mnoštva modela, neki su posve prirodni i direktno slijede iz verbalnog opisa rada sklopa, dok su drugi modeli posve vještački.

U praktičnim problemima veoma se često dešava da biti stanja ujedno predstavljaju i izlaze iz sekvencijalnog sklopa, tako da se funkcije izlaza svode na trivijalne funkcije $y_j(n) = Q_j(n)$, $j = 1 .. M$ (u takvim slučajevima je naravno M = P). Nešto općenitiji slučaj je slučaj kada trenutne vrijednosti izlaza zavise samo od trenutnih vrijednosti bita stanja, ali ne i od trenutnih vrijednosti ulaza (tj. izlazi su u potpunosti određeni trenutnim stanjem). U tom slučaju, jednačine kojima se opisuje sklop postaju

$$Q_k(n+1) = \Phi_k[x_1(n), x_2(n), \ldots x_N(n), Q_1(n), Q_2(n), \ldots Q_P(n)], \qquad k = 1 .. P$$

$$y_j(n) = \Psi_j[Q_1(n), Q_2(n), \ldots Q_P(n)], \qquad j = 1 .. M$$

Modele sekvencijalnih sklopova kod kojih trenutne vrijednosti izlaza zavise samo od trenutnih vrijednosti promjenljivih stanja nazivamo ***Mooreovim automatima***, a one kod kojih to nije slučaj nazivamo ***Mealyjevim automatima***. Kako svaki od ova dva modela sekvencijalnih sklopova ima svoje prednosti i mane, od velikog praktičnog značaja je sljedeća teorema:

Teorema 15.2:

Za svaki Mealyjev automat postoji Mooreov automat koji obavlja istu funkciju, i obrnuto. Pri tome, izlaz ekvivalentnog Mooreovog automata kasni jednu vremensku jedinicu za izlazom Mealyjevog automata, odnosno izlaz ekvivalentnog Mealyjevog automata prednjači za jednu vremensku jedinicu u odnosu na izlaz Mooreovog automata.

Ovu teoremu je veoma lako dokazati. Naime, neka je dat model Mealyjevog automata u obliku:

$$Q_k(n+1) = \Phi_k[x_1(n), x_2(n), \ldots x_N(n), Q_1(n), Q_2(n), \ldots Q_P(n)], \quad k = 1 \,..\, P$$
$$y_j(n) = \Psi_j[x_1(n), x_2(n), \ldots x_N(n), Q_1(n), Q_2(n), \ldots Q_P(n)], \quad j = 1 \,..\, M$$

Uvedimo sada nove izlazne promjenljive y'_j, $j = 1 \,..\, M$ koje su sa izlaznim promjenljivim y_j povezane relacijama $y'_j(n) = y_j(n–1)$, $j = 1 \,..\, M$ i uvedimo M novih promjenljivih stanja Q_k, $k = P+1 \,..\, P+M$ preko relacija $Q_k(n) = y'_{k–P}(n)$, $k = P+1 \,..\, P+M$. Tada se gornje jednačine mogu napisati u sljedećem obliku:

$$Q_k(n+1) = \begin{cases} \Phi_k[x_1(n), x_2(n), \ldots x_N(n), Q_1(n), Q_2(n), \ldots Q_P(n)], & k = 1 \,..\, P \\ \Psi_{k–P}[x_1(n), x_2(n), \ldots x_N(n), Q_1(n), Q_2(n), \ldots Q_P(n)], & k = P+1 \,..\, P+M \end{cases}$$
$$y'_j(n) = Q_{j+P}(n), \quad j = 1 \,..\, M$$

Ovo su očigledno jednačine Mooreovog tipa (sa trivijalnim izlaznim funkcijama), odnosno dobili smo ekvivalentni Mooreov automat (sa M dodatnih promjenljivih stanja), čiji izlaz kasni za jednu vremensku jedinicu u odnosu na polazni automat zbog relacija $y'_j(n) = y_j(n–1)$. Posmatrajmo sada model Mooreovog automata u obliku:

$$Q_k(n+1) = \Phi_k[x_1(n), x_2(n), \ldots x_N(n), Q_1(n), Q_2(n), \ldots Q_P(n)], \quad k = 1 \,..\, P$$
$$y_j(n) = \Psi_j[Q_1(n), Q_2(n), \ldots Q_P(n)], \quad j = 1 \,..\, M$$

Uvedimo nove izlazne promjenljive y'_j, $j = 1 \,..\, M$ koje su sa izlaznim promjenljivim y_j povezane relacijama $y'_j(n) = y_j(n+1)$, $j = 1 \,..\, M$. Kako vrijedi $y'_j(n) = y_j(n+1) = \Psi_j[Q_1(n+1), Q_2(n+1), \ldots Q_P(n+1)]$, to gornje jednačine postaju:

$$Q_k(n+1) = \Phi_k[x_1(n), x_2(n), \ldots x_N(n), Q_1(n), Q_2(n), \ldots Q_P(n)], \quad k = 1 \,..\, P$$
$$\begin{aligned} y'_j(n) = \Psi_j[&\Phi_1[x_1(n), x_2(n), \ldots x_N(n), Q_1(n), Q_2(n), \ldots Q_P(n)], \\ &\Phi_2[x_1(n), x_2(n), \ldots x_N(n), Q_1(n), Q_2(n), \ldots Q_P(n)], \ldots \\ &\Phi_P[x_1(n), x_2(n), \ldots x_N(n), Q_1(n), Q_2(n), \ldots Q_P(n)]], \quad j = 1 \,..\, M \end{aligned}$$

Ovo su očigledno jednačine Mealyjevog tipa, pri čemu izlaz odgovarajućeg Mealyjevog automata prednjači za jednu vremensku jedinicu u odnosu na polazni automat zbog relacija $y'_j(n) = y_j(n–1)$. Ovim je teorema dokazana.

Pri dokazu navedene teoreme, prelazak sa jednačina Mealyjevog tipa na jednačine Mooreovog tipa zahitjevao je uvođenje M novih bita stanja. Kako će u kasnijim poglavljima biti pokazano, ovo prevođenje se često može izvršiti uvođenjem manjeg broja bita stanja (ponekad čak i bez uvođenja novih bita stanja). Na ovom mjestu nismo ulazili u analizu takvih mogućnosti, jer nam je samo bio cilj da pokažemo da je prevođenje jednačina Mealyjevog tipa u jednačine Mooreovog tipa i obrnuto barem *principijelno moguće*.

Primijetimo da se kod opisivanja sekvencijalnih sklopova konačnim automatima ulazi x_i i izlazi y_j uvijek javljaju iskazani samo u vremenskom trenutku *n*, tj. kao $x_i(n)$ i $y_j(n)$, dok se jedino biti stanja Q_k javljaju u *različitim* vremenskim trenucima (*n* i *n*+1 kod automata prve vrste, odnosno *n* i *n*–1 kod

automata druge vrste). Stoga ćemo često radi kraćeg izražavanja izostavljati vremenski argument kod oznaka ulaza i izlaza, i pisati ga samo kod promjenljivih stanja. Drugim riječima, jednačine koje opisuju konačni automat prve vrste pisaćemo u sljedećem obliku:

$$Q_k(n+1) = \Phi_k[x_1, x_2, \ldots x_N, Q_1(n), Q_2(n), \ldots Q_P(n)], \qquad k = 1 \ldots P$$

$$y_j = \Psi_j[x_1, x_2, \ldots x_N, Q_1(n), Q_2(n), \ldots Q_P(n)], \qquad j = 1 \ldots M$$

Čak ćemo ponekad i kod promjenljivih stanja izostavljati vremenski argument u slučaju kada je on n (tj. pisaćemo samo Q_k umjesto $Q_k(n)$). U literaturi se nekada umjesto $Q_k(n+1)$ susreće i skraćena oznaka Q^+_k.

Do sada smo samo govorili o matematskim modelima za opis rada sekvencijalnih sklopova, ali ne i o tome *kako se ti matematski modeli formiraju*. Na ovom mjestu će biti ilustriran jedan jednostavniji primjer formiranja matematskog modela za sekvencijalni sklop čiji je rad opisan *verbalno*, dok će se komplikovaniji slučajevi razmatrati u poglavljima koja slijede.

Primjer 15.2:

- Formirati matematski model za *serijski sabirač* (*sumator*), čiji je verbalni opis rada dat na početku ovog poglavlja.

Već smo na samom početku ovog poglavlja uvidjeli da serijski sabirač ne može biti realiziran kao kombinacioni sklop, s obzirom na činjenicu da vrijednost izlaza C u nekom trenutku n zavisi ne samo od ulaza A i B u tom istom trenutku, nego i od ranijih vrijednosti ulaza, odnosno C(n) zavisi kako od A(n) i B(n), tako i od A(n–1), B(n–1), A(n–2), B(n–2) itd. Na prvi pogled izgleda da je jako teško opisati jednačinama rad ovog sklopa, jer na vrijednost izlaza može utjecati proizvoljno mnogo prethodnih vrijednosti ulaza. Međutim, ukoliko malo razmislimo, vidjećemo da rezultat C u nekom trenutku n pored ulaza A i B u tom istom trenutku zavisi samo od toga da li je prilikom sabiranja prethodnih cifara (tj. u trenutku n–1) *bilo prenosa* ili ga *nije bilo*. Prema tome, možemo uvesti pomoćnu promjenljivu stanja P čija vrijednost pamti da li je prilikom prethodnog sabiranja bilo ili nije bilo prenosa. Možemo npr. usvojiti da P = 1 označava da je bilo prenosa, a P = 0 da ga nije bilo (sasvim bi bilo legalno usvojiti i obrnutu konvenciju, s obzirom da je P samo pomoćna promjenljiva koja se nigdje ne javlja kao izlaz iz sklopa). Tada će izlaz C u trenutku n ovisiti od ulaza A i B i promjenljive stanja P u tom istom trenutku, dok će *buduća vrijednost* promjenljive P (tj. prenos koji će eventualno biti iskorišten za sabiranje narednih cifara) također ovisiti od istih veličina. Na osnovu ovoga nije teško sastaviti sljedeću tablicu, koja opisuje rad serijskog sabirača, uz ovako usvojenu promjenljivu stanja P:

A(n)	B(n)	P(n)	C(n)	P(n+1)
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

Ova tablica po formi je veoma slična tablici istine, ali se strogo uzevši ne radi o tablici istine, jer se u ovoj tablici ista promjenljiva P javlja u dva različita vremenska trenutka. Ovakve tablice nazivaju se ***tablice prelaza i izlaza*** sekvencijalnih sklopova (za slučaj kada su izlazi jednaki promjenljivim stanja, možemo govoriti samo o ***tablicama prelaza***). Bez obzira na tu formalnu razliku, možemo se poslužiti svim do sada razmatranim metodama logičke algebre da izrazimo C(n) i P(n+1) kao funkcije od A(n), B(n) i P(n). Zapravo, ova tablica je po formi identična kao tablica istine za *puni sumator* koji smo već razmatrali. Stoga ćemo, da ne ponavljamo isti postupak, samo napisati tražene jednačine, koje glase:

$$P(n+1) = A(n)\,B(n) \vee [A(n) \vee B(n)]\,P(n)$$

$$C(n) = A(n) \oplus B(n) \oplus P(n)$$

Alternativno, koristeći skraćenu notaciju, iste jednačine možemo zapisati i ovako:

$$P(n+1) = AB \vee (A \vee B)\,P(n)$$

$$C = A \oplus B \oplus P(n)$$

Ovo nije ništa drugo nego matematski model rada serijskog sabirača. Primijetimo da se radi o konačnom automatu prve vrste Mealyjevog tipa. Ipak, bez obzira što smo formirali matematski model, mi još ne znamo kako da *napravimo* ovaj sklop, jer ne znamo kako da razlučimo trenutke n i n+1, odnosno kako da *zadržimo* tekući prenos da bude raspoloživ u sljedećem trenutku vremena, kada budemo sabirali naredne dvije cifre. Ovaj problem će biti riješen u poglavljima koja slijede. Intuitivno, potrebna nam je neka vrsta *memorijskog elementa* (i to RAM a ne ROM tipa) koji će biti u stanju da zapamti stanje sklopa, odnosno vrijednost promjenljive P do narednog vremenskog trenutka. Stoga se sekvencijalni sklopovi često nazivaju i ***sklopovi sa pamćenjem*** (odnosno ***sa memorijom***), za razliku od kombinacionih sklopova koji se nazivaju i ***sklopovi bez pamćenja*** (ili ***bez memorije***).

Teoretski je moguće opisati i digitalne sekvencijalne sklopove koji se ne mogu opisati konačnim automatima: To su sklopovi koji se mogu opisati sistemom jednačina u prostoru stanja oblika

$$S(n) = \Phi[x_1(n), x_2(n), \ldots x_N(n), S(n-1)]$$

$$y_j(n) = \Psi_j[x_1(n), x_2(n), \ldots x_N(n), S(n)], \quad j = 1 \,..\, M$$

ali kod kojih nije moguće postići da stanje S uzima samo konačno mnogo vrijednosti, već može (i mora) uzimati *beskonačno mnogo različitih vrijednosti*, tako da ga nije moguće zamijeniti sa konačno mnogo bita stanja Q_k. Takvi sekvencijalni sklopovi nazivaju se ***beskonačni automati***, i pokazuje se da je njih nemoguće fizički realizirati ma kakvom raspoloživom tehnologijom, jer memoriranje promjenljive stanja S čija vrijednost nije ničim ograničena zahtijeva neograničene memorijske resurse, što je očigledno neizvodljivo. Začudo, uopće nije teško formirati verbalne opise sekvencijalnih sklopova koji se ne mogu fizički realizirati, mada im opis rada djeluje posve bezazleno. Tipičan primjer je sklop sa jednim ulazom X i jednim izlazom Y koji *broji* koliko puta se na ulazu (u unaprijed fiksiranim vremenskim trenucima) pojavila vrijednost 0, a koliko puta vrijednost 1, i koji na izlazu daje jedinicu ukoliko je broj pojava jedinica na ulazu veći od broja pojava nula na ulazu, dok u suprotnom na izlazu daje nulu. Nije teško dati matematski opis ovog sklopa. Na primjer, moguće je uvesti cjelobrojnu promjenljivu stanja S koja se uvećava za 1 svaki put kada je ulaz X jednak jedinici, a umanjuje svaki put kada je ulaz X jednak nuli. Tada je izlaz Y jednak jedinici ako i samo ako je $S > 0$. Drugim riječima, rad sklopa može se opisati sljedećim jednačinama:

$$S(n) = \begin{cases} S(n-1)+1, & \text{ako je } X(n)=1 \\ S(n-1)-1, & \text{ako je } X(n)=0 \end{cases} \qquad \text{ili kompaktnije,} \quad S(n) = S(n-1) + 2\,X(n) - 1$$

$$Y(n) = \begin{cases} 1, & \text{ako je } S(n) > 0 \\ 0, & \text{ako je } S(n) \le 0 \end{cases}$$

Ove jednačine predstavljaju opis traženog sklopa u prostoru stanja. Međutim, pretpostavimo da na ulaz sklopa dovedemo ogromnu sekvencu koja se sastoji samo od jedinica. Tada će se promjenljiva S neprestano povećavati. Koliko god da je memorije rezervirano za pamćenje promjenljive S, jednom će ta memorija biti iscrpljena (s obzirom da se u ma kakvoj memoriji može zapamtiti samo *konačno mnogo bita*, što ograničava maksimalnu moguću vrijednost za S), i dalje brojanje jedinica neće biti moguće, nakon čega sklop više ne može raditi ispravno. Prema tome, sklop koji obavlja traženu funkciju *nije moguće fizički realizirati*!

Prethodni zaključak djeluje posve obeshrabrujuće, tim prije što verbalni opis traženog sklopa ne djeluje komplicirano. Na sreću, u praksi stvari stoje znatno bolje. Pretpostavimo da smo se u sklopu iz prethodnog primjera odlučili da na silu *ograničimo* vrijednosti koje može uzimati promjenljiva stanja npr. na vrijednosti koje mogu stati u 64 bita. Tada ovaj sklop možemo opisati kao konačni automat sa 64 bita stanja Q_k, $k = 1 .. 64$. Može se postaviti pitanje šta smo postigli ovim ograničenjem. Očigledno, sklop prestaje raditi ispravno kada broj uzastopnih jedinica (ili nula) dovedenih na ulaze postane veći od najvećeg broja koji može stati u 63 bita (u jednom bitu se mora čuvati znak broja S), a taj broj iznosi $2^{63} - 1 = 9223372036854775807$. Čak uz pretpostavku da na ulaz sklopa dovodimo *milijardu* jedinica u jednoj sekundi, proći će oko 292 godine prije nego što sklop prestane raditi (naravno, ukoliko se prije toga ne pokvari, što je sasvim izvjesno). Dakle, mada je traženi sklop neizvodljiv *teoretski*, on postaje izvodljiv ukoliko mu se nametnu neka ograničenja koja se u praksi zapravo nikada ne dostižu.

Znatno problematičniji primjer "neostvarljivog" sekvencijalnog sklopa je ***serijski množač***. Ovaj sklop je analogan serijskom sabiraču, sa dva ulaza A i B i jednim izlazom C, na kojem se pojavljuje odgovarajuća cifra *produkta* (umjesto *zbira*) brojeva čije cifre dovodimo na ulaze. Kako produkt dva broja od *n* bita ima 2*n* cifara, serijski množač u svakom vremenskom trenutku *n* mora da ima zapamćene međurezultate koji omogućavaju formiranje narednih *n* bita produkta. Očigledno, količina međurezultata raste sa porastom broja bita koji dolaze, odnosno sa porastom vremena. Koliko god da je memorije predviđeno za pamćenje međurezultata, jednom će se ta količina istrošiti, i sklop neće moći ispravno nastaviti rad. Stoga je serijski množač moguće realizirati samo ukoliko unaprijed ograničimo dužinu sekvenci bita koje se dovode na ulaz, odnosno maksimalnu dužinu brojeva koji se množe (za razliku od serijskog sabirača, koji je može da radi *bez ikakvih ograničenja*). Međutim, u ovom primjeru memorijski zahtjevi su mnogo veći nego u primjeru brojača nula i jedinica (stoga je čak i uz ova ograničenja serijski množač, za razliku od serijskog sabirača, izuzetno kompliciran sklop). Tako, ukoliko želimo da napravimo serijski množač koji može da množi dva broja dužine 1000 bita (potreba za množenjem ovako velikih brojeva javlja se u kriptografiji), potrebno je predviditi najmanje još 1000 bita stanja za prihvaćanje narednih 1000 bita rezultata, da ne računamo i još mnoštvo promjenljivih stanja koje je potrebno uvesti za pamćenje raznih međurezultata. Serijski množač koji bi bio u stanju da pomnoži dva broja dužine milijardu bita neostvarljiv je uz današnju tehnologiju. Srećom, praksa nikada ne nameće potrebu za množenjem ovako velikih brojeva. Pokazalo se da se svi "neostvarljivi" sekvencijalni sklopovi mogu pretvoriti u "ostvarljive" uvođenjem ograničenja koja su prihvatljiva za potrebe prakse.

Izloženi dokazi da opisani brojač nula i jedinica kao i serijski množač nisu ostvarljivi bez nametanja izvjesnih ograničenja, iskazana su dosta intuitivno, i nemaju karakter strogog dokaza (na primjer, u slučaju brojača nula i jedinica, nije pokazano da se isti sklop ne može opisati nekim drugim modelom koji ne zahtijeva beskonačno mnogo različitih vrijednosti promjenljive stanja). Neostvarljivost ovih sklopova moguće je dokazati mnogo strože, tako što se pokaže da ovi sklopovi ne mogu ispuniti neka svojstva koja svi konačni automati moraju zadovoljavati (pogledajte, na primjer, zadatke koji slijede na kraju ovog poglavlja).

Treba napomenuti da, bez obzira na njihovu fizičku neostvarljivost, razni matematički modeli beskonačnih automata poput ***Turingovih mašina*** i ***stek automata*** imaju veliku primjenu u pojedinim oblastima teoretskih računarskih nauka, naročito u *teoriji jezika i automata* i *teoriji izračunljivosti*. Tom prilikom se često prikazuju i različite fizičke "realizacije" ovih automata, koje su ipak ostvarljive samo aproksimativno, jer zahtijevaju neke praktično neostvarljive koncepte. Uglavnom se radi o zahtjevima na neograničenu količinu memorijskih resursa (npr. Turingova mašina zahtijeva *traku beskonačne dužine*), koji se, u najboljem slučaju, mogu realizirati samo aproksimativno. Razlog zbog kojeg se u teoretskim analizama razmatraju i takvi neostvarljivi modeli je sljedeći. Analiza da li je neki problem uopće rješiv ili nije osjetno se pojednostavljuje ukoliko pri tome ne moramo voditi računa o količini resursa (npr. kapaciteta memorije) koji su nam na raspolaganju, tj. ukoliko kao model za rješavanje problema koristimo neki model beskonačnog automata (Turingova mašina je najčešće korišten takav model). Ukoliko se pokaže da neki problem nije rješiv čak i uz pretpostavku da su za njegovo rješavanje na raspolaganju *neograničeni resursi*, jasno je da taj problem sigurno nije rješiv niti pomoću jednog fizički ostvarljivog modela, odnosno da je prosto rečeno *nerješiv*! Nažalost, takvih problema ima i više nego što bi se moglo pretpostaviti. Djelimično sretna okolnost je što za mnoge probleme za koje se pokazuje da

su teoretski nerješivi, ta nerješivost vrijedi samo za generalno postavljenu formulaciju problem, dok su mnogi njegovi specijalni podslučajevi rješivi. Analizom da li je neki problem uopće rješiv (makar principijelno) ili ne, bavi se teorijska računarska nauka poznata pod nazivom ***teorija izračunljivosti***.

Na kraju ovog poglavlja napomenimo da bismo, ukoliko se strogo držimo definicije, mogli zaključiti da kombinacioni sklopovi zapravo *uopće ne postoje*. Zaista, posmatrajmo bilo koje elementarno logičko kolo, poput invertora. Zbog prisustva uvijek neizbježnog kašnjenja koja unose sva logička kola, rad invertora ne opisuje se tačno jednačinom oblika $Y = \overline{X}$ odnosno $Y(t) = \overline{X(t)}$, nego jednačinom oblika $Y(t) = \overline{X(t-\Delta t)}$ odnosno $Y(t+\Delta t) = \overline{X(t)}$, gdje je Δt kašnjenje koje unosi invertor. Usvojimo li upravo kašnjenje Δt za vremensku jedinicu, ovu bismo jednačinu mogli pisati kao $Y(n+1) = \overline{X(n)}$, odakle bismo mogli steći dojam da je čak i običan invertor *sekvencijalni sklop*. Međutim, u svim praktičnim razmatranjima, pod sekvencijalnim sklopovima se smatraju samo sklopovi koji su u stanju zadržati informaciju primljenu na ulazu znatno duže vrijeme nego što iznose tipična kašnjenja logičkih kola, i kod kojih projektant sklopa može utjecati na izbor vremenske jedinice koja se koristi pri opisu rada sklopa. Odnosno, pod sekvencijalnim sklopovima podrazumijevamo samo one sklopove koji bi bili takvi čak i uz pretpostavku da sva logička kola reagiraju *trenutno*. Kako se ovo postiže, vidjećemo u poglavljima koja slijede.

(?) Pitanja i zadaci

15.1 Objasnite zbog čega kombinacioni sklopovi ne mogu zadovoljiti sve potrebe digitalne tehnike.

15.2 Objasnite zbog čega računari ne mogu biti kombinacioni sklopovi.

15.3 Definirajte pojam sekvencijalnog digitalnog sklopa. Posebno naglasite zbog čega se ne smije dopustiti da vrijednosti izlaza u nekom trenutku vremena $t = t_0$ zavise od vrijednosti ulaza u *proizvoljnim* trenucima $t < t_0$.

15.4 Opišite rad serijskog sabirača kao tipičnog primjera sekvencijalnog sklopa.

15.5* Pokažite da sistem čiji je rad opisan jednačinom

$$Y(t) = \begin{cases} 1, & \text{ako je } \dfrac{1}{t}\displaystyle\int_0^t X(\tau)\,d\tau > \dfrac{1}{2} \\ 0, & \text{u suprotnom} \end{cases}$$

zaista opisuje sklop sa jednim ulazom X i samo jednim izlazom Y na kojem se u nekom trenutku vremena $t = t_0$ pojavljuje jedinica ako i samo ako je u vremenskom intervalu od nekog početnog trenutka $t = 0$ do tekućeg trenutka $t = t_0$ na ulazu X duže vremena bila jedinica nego nula.

15.6 Objasnite pojam diskretnog vremena, i vezu između diskretnog i stvarnog vremena. Posebno istaknite slučaj kada su razmaci između usvojenih diskretnih vremenskih trenutaka jednaki.

15.7 Navedite opće jednačine koja opisuje rad ma kakvog sekvencijalnog sklopa. U čemu je suštinski problem za primjenu takvih jednačina?

15.8 U apstraktnoj teoriji diskretnih sistema, često se pored kombinacionih i sekvencijalnih sklopova, teoretski razmatra i treća kategorija sklopova, nazvana ***anticipativni*** (ili ***predviđajući***) ***sklopovi***. Kod njih vrijednosti izlaza u nekom trenutku, pored sadašnjih i prošlih, mogu zavisiti i od *budućih vrijednosti ulaza*. Navedite primjer jednačine koja opisuje neki anticipativni sklop. Šta možete reći o mogućnosti fizičke realizacije takvih sklopova?

15.9 Na primjeru matematskog modeliranja sefa pokažite kako se u nekim slučajevima možemo osloboditi jednačina koje eksplicitno zavise od beskonačno mnogo vrijednosti ulaza (i koje se, prema tome, mogu kompaktno izraziti samo pomoću beskonačnih redova, produkata, itd.).

15.10 Izvedite pojednostavljeni matematski model nacionalnog bruto dohotka.

15.11 Posmatrajmo pojednostavljeni matematski model nacionalnog bruto dohotka. Usvojimo $n = 0$ za početak mjerenja vremena. Tada je, jasno, $X(n) = 0$ i $Y(n) = 0$ za $n < 0$. Neka je jedinica vremena jedna godina. Pretpostavimo da su u narednom periodu neplanirana ulaganja ista svake godine, tj. $X(n) = c$ za $n \geq 0$, gdje je c neka konstanta. Izračunajte koliki će biti nacionalni bruto dohodak za svaku godinu u periodu od narednih 5 godina. Posebno razmotrite slučaj $c = 1$, $\alpha = 0.3$ i $\beta = 0.2$.

15.12 Objasnite šta su diferentne odnosno rekurentne jednačine, i šta je red diferentne jednačine.

15.13 Da li je za model nacionalnog bruto dohotka lakše formirati eksplicitnu ili diferentnu jednačinu?

15.14 Dat je diskretni sistem sa jednim ulazom X i jednim izlazom Y opisan jednačinom

$$Y(n) = X(n) + \frac{1}{\alpha}X(n-1) + \frac{1}{\alpha^2}X(n-2) + \frac{1}{\alpha^3}X(n-3) + \ldots = \sum_{k=0}^{\infty}\frac{1}{\alpha^k}X(n-k)$$

gdje je α neka konstanta. Opišite isti sistem uz pomoć diferentne jednačine.

15.15 Objasnite zbog čega je, bez umanjenja općenitosti, uvijek moguće smatrati da su sve diferentne jednačine koje opisuju ponašanje nekog diskretnog sistema jednačine prvog reda.

15.16 Objasnite šta su promjenljive stanja i zbog čega se uvode.

15.17 Napišite opće jednačine diskrenih modela opisanih u prostoru stanja.

15.18 Opišite značenje funkcija prelaza, funkcija izlaza i stanja sistema prilikom opisa diskretnih modela u prostoru stanja.

15.19 Objasnite zbog čega je opis u prostoru stanja zapravo najprirodniji način za opisivanje rada diskretnih sistema.

15.20 Dat je diskretni sistem sa jednim ulazom X i jednim izlazom Y opisan jednačinom

$$Y(n) = \frac{1}{n}X(n) + \frac{1}{n-1}X(n-1) + \frac{1}{n-2}X(n-2) + ... + \frac{1}{2}X(2) + X(1) = \sum_{k=0}^{n-1}\frac{1}{n-k}X(n-k)$$

Pronađite opis ovog istog sistema u prostoru stanja.

15.21 Dat je diskretni sistem sa jednim ulazom X i jednim izlazom Y opisan jednačinom

$$Y(n) = \frac{1}{n}\left[X(n) + X(n-1) + X(n-2) + ... + X(2) + X(1)\right] = \frac{1}{n}\sum_{k=0}^{n-1}X(n-k)$$

Ovaj sistem naziva se ***usrednjivač***, s obzirom da mu je vrijednost izlaza u svakom trenutku jednaka aritmetičkoj sredini svih dosadašnjih vrijednosti na ulazu. Pronađite opis ovog sistema u prostoru stanja.

15.22 Da li je svaki diskretni model moguće opisati u prostoru stanja?

15.23** Dokažite da je digitalne sekvencijalne sklopove uvijek moguće opisati u prostoru stanja koristeći samo jednu promjenljivu stanja S, pod uvjetom da dopustimo da promjenljiva stanja S uzima realne vrijednosti.

15.24 Objasnite šta su konačni automati, i kako izgledaju jednačine koje ih opisuju.

15.25 Objasnite kakva je razlika između konačnih automata prve i druge vrste.

15.26 Pretvorite prikazani model konačnog automata druge vrste u model automata prve vrste:

$$Q(n) = x_1(n)\,Q(n-1) \vee x_2(n)$$

$$y_1(n) = x_1(n) \oplus Q(n)$$

$$y_2(n) = \overline{x_2(n)}\,Q(n)$$

15.27 Pretvorite prikazani model konačnog automata druge vrste u model automata prve vrste:

$$Q(n) = x_1(n)\,x_2(n) \oplus Q(n-1)$$

$$y_1(n) = x_1(n)\,Q(n) \vee x_2(n)\,\overline{Q(n)}$$

$$y_2(n) = \overline{x_1(n)}\,\overline{x_2(n)}\,Q(n)$$

$$y_3(n) = x_1(n)\,x_2(n) \vee Q(n)$$

15.28 Pretvorite prikazani model konačnog automata druge vrste u model automata prve vrste:

$$Q_1(n) = x_1(n)\,Q_2(n-1) \vee \overline{x_2(n)}\,Q_1(n-1)$$

$$Q_2(n) = Q_1(n-1) \oplus Q_2(n-1)$$

$$y_1(n) = x_1(n)\,Q_1(n) \vee x_2(n)\,Q_2(n)$$

$$y_2(n) = \overline{Q_1(n)}\,\overline{Q_2(n)}$$

15.29 Pretvorite prikazani model konačnog automata druge vrste u model automata prve vrste:

$Q_1(n) = x_1(n) \oplus Q_1(n-1)$

$Q_2(n) = x_2(n) \oplus Q_1(n-1)\, Q_2(n-1)$

$Q_3(n) = x_3(n) \oplus Q_1(n-1)\, Q_2(n-1)\, Q_3(n-1)$

$y(n) = x_1(n)\, x_2(n) \vee Q_3(n)$

15.30 Razmotrimo model konačnog automata iz Zadatka 15.27. Pretpostavimo da je sve do trenutka $n = 0$ promjenljiva stanja Q imala vrijednost 0. Počev od trenutka $n = 0$ nadalje, na ulaz x_1 dovodi se sekvenca 1, 1, 0, 1, 0, 0, 1, 0 (svaka vrijednost odgovara po jednom diskretnom trenutku), a na ulaz x_2 dovodi se sekvenca 1, 0, 0, 1, 1, 0, 1, 1. Odredite kakve će se sekvence pojaviti na izlazima y_1, y_2 i y_3.

15.31 Razmotrimo model konačnog automata iz Zadatka 15.28. Pretpostavimo da su sve do trenutka $n = 0$ promjenljive stanja Q_1 i Q_2 imale vrijednost 0. Počev od trenutka $n = 0$ nadalje, na ulaz x_1 dovodi se sekvenca 1, 0, 0, 1, 1, 1, 0, a na ulaz x_2 dovodi se sekvenca 0, 0, 0, 1, 1, 0, 1. Odredite kakve će se sekvence pojaviti na izlazima y_1 i y_2.

15.32 Razmotrimo model konačnog automata iz Zadatka 15.29. Pretpostavimo da su sve do trenutka $n = 0$ promjenljive stanja Q_1, Q_2 i Q_3 imale vrijednost 0. Počev od trenutka $n = 0$ nadalje, na ulaz x_1 dovodi se sekvenca 0, 1, 0, 1, 0, 1, 1, na ulaz x_2 dovodi se sekvenca 1, 1, 1, 0, 1, 0, 1, a na ulaz x_3 sekvenca 1, 1, 1, 0, 1, 1, 0. Odredite kakva će se sekvenca pojaviti na izlazu y.

15.33 Objasnite zbog čega se pretvaranje modela konačnih automata prve vrste u modele druge vrste ne može jednostavno obaviti na analogan način kao pretvorba iz modela druge vrste u modele prve vrste, i zbog čega takva pretvorba nije uvijek moguća.

15.34* Pretvorite prikazani model konačnog automata prve vrste u model automata druge vrste:

$Q(n+1) = x(n)\, Q(n)$

$y(n) = x(n)\, \overline{Q(n)}$

15.35** Pretvorite prikazani model konačnog automata prve vrste u model automata druge vrste:

$Q(n+1) = x_1(n)\, Q(n) \vee x_2(n)$

$y_1(n) = \overline{x_2(n)}\, x_1(n)\, Q(n) \vee x_2(n)\, \overline{x_1(n)}$

$y_2(n) = \overline{x_2(n)}\, x_1(n)\, Q(n)$

15.36* Konstruirajte primjer modela konačnog automata prve vrste koji se ne može prevesti u ekvivalentni model druge vrste.

15.37 Razmotrimo model konačnog automata iz Zadatka 15.35. Pretpostavimo da je sve do trenutka $n = 0$ promjenljiva stanja Q imala vrijednost 0. Počev od trenutka $n = 0$ nadalje, na ulaz x_1 dovodi se sekvenca 0, 1, 0, 1, 0, 1, 1, 0, a na ulaz x_2 dovodi se sekvenca 1, 1, 0, 1, 1, 0, 0, 0. Odredite kakve će se sekvence pojaviti na izlazima y_1 i y_2.

15.38 Opći diskretni modeli također mogu biti prve i druge vrste: kod modela prve vrste funkcije prelaza imaju oblik

$S_k(n+1) = \Phi_k[X_1(n), X_2(n), \ldots X_N(n), S_1(n), S_2(n), \ldots S_P(n)], \quad k = 1..P$

umjesto

$S_k(n) = \Phi_k[X_1(n), X_2(n), \ldots X_N(n), S_1(n-1), S_2(n-1), \ldots S_P(n-1)], \quad k = 1..P$

dok funkcije izlaza zadržavaju isti oblik. Opišite rad sefa iz uvodnih razmatranja pomoću modela druge vrste.

15.39 Objasnite kakva je razlika između Mealyjevih i Mooreovih konačnih automata.

15.40 Koristeći ideje iz dokaza Teoreme 15.2, pretvorite sljedeći model Mealyjevog konačnog automata u ekvivalentni model Mooreovog konačnog automata:

$Q_1(n+1) = x_1(n)\,Q_2(n-1) \vee \overline{x_2(n)}\,Q_1(n-1)$

$Q_2(n+1) = Q_1(n) \oplus Q_2(n)$

$y_1(n) = x_1(n)\,Q_1(n) \vee x_2(n)\,Q_2(n)$

$y_2(n) = \overline{Q_1(n)}\,\overline{Q_2(n)}$

15.41 Koristeći ideje iz dokaza Teoreme 15.2, pretvorite sljedeći model Mooreovog konačnog automata u ekvivalentni model Mealyjevog konačnog automata:

$Q_1(n+1) = x_1(n)\,Q_2(n-1) \vee \overline{x_2(n)}\,Q_1(n-1)$

$Q_2(n+1) = Q_1(n) \oplus x_1(n)\,x_2(n)$

$y_1(n) = Q_1(n) \oplus Q_2(n)$

$y_2(n) = \overline{Q_1(n)}\,\overline{Q_2(n)}$

$y_3(n) = Q_1(n)$

15.42 Objasnite šta su tablice prelaza i izlaza sekvencijalnih sklopova, i po čemu se one razlikuju od klasičnih tablica istine.

15.43 Izvedite matematski model serijskog sabirača, ali uz pretpostavku da vrijednost promjenljive stanja P = 1 označava da nije bilo prenosa, a vrijednost P = 0 da je bilo prenosa.

15.44 Objasnite zbog čega se sekvencijalni sklopovi nazivaju i sklopovi sa pamćenjem, odnosno sklopovi sa memorijom.

15.45 Šta su beskonačni automati, i šta možete reći o njihovoj fizičkoj ostvarljivosti?

15.46 Navedite primjere digitalnih sklopova koje nije moguće fizički realizirati, i opišite osnovnu ideju kako bi se taj sklop mogao realizirati aproksimativno, na način koji zadovoljava potrebe prakse.

15.47 Objasnite zbog čega nas činjenica da postoje relativno jednostavni modeli digitalnih sklopova koje nije moguće fizički reazlizirati ne treba previše zabrinjavati.

15.48* Pokažite da ukoliko se u ma kakvom konačnom automatu od nekog trenutka nadalje svi ulazi drže stalno na istim vrijednostima (tj. ukoliko su vrijednosti ulaza iste u svim narednim vremenskim trenucima), da tada nakon najviše P vremenskih jedinica svi izlazi iz konačnog automata moraju postati sekvence koje se periodično ponavljaju, sa periodom od najviše P vremenskih jedinica, pri čemu je P broj različitih stanja koje automat može imati.

15.49* Koristeći tvrdnju iz Zadatka 15.48, dokažite strogo da sklop sa jednim ulazom X i jednim ulazom Y koji daje jedinicu na izlazu ako i samo ako je broj dotadašnjih pojava jedinica na ulazu veći od broja pojava nula fizički neostvarljiv.

15.50* Koristeći tvrdnju iz Zadatka 15.48, dokažite strogo da serijski množač nije fizički ostvarljiv.

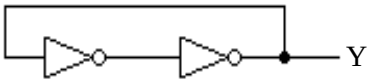
15.51 Objasniti zbog čega se matematički modeli beskonačnih automata intenzivno proučavaju u teorijskim računarskim naukama, bez obzira na njihovu fizičku neostvarljivost.

15.52 S obzirom da sva logička kola unose neizbježno kašnjenje, izlazi svih logičkih mreža u suštini nikada ne zavise od trenutnih, nego od prethodnih vrijednosti ulaza. Možemo li odatle zaključiti da su svi digitalni sklopovi zapravo sekvencijalni?

16. Elementarni automati (flip-flopovi)

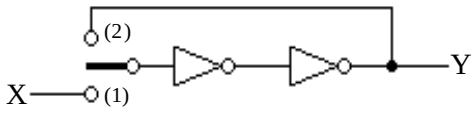
U prethodnom poglavlju smo vidjeli da se rad fizički ostvarljivih sekvencijalnih sklopova matematski opisuje pomoću *konačnih automata*. Međutim, sam matematski model sekvencijalnih sklopova još ne ukazuje na ideju kako se takvi sklopovi mogu i realizirati. Vidjeli smo da je presudno svojstvo za rad sekvencijalnih sklopova mogućnost *memoriranja informacije o stanju sklopa*. Stoga je objašnjenje rada sekvencijalnih sklopova najbolje započeti opisom rada sklopova koji posjeduju mogućnost da proizvoljno dugo vremena zapamte *jednobitnu informaciju* u jednoj (jednobitnoj) promjenljivoj stanja, i da tu informaciju proslijede na izlaz sklopa. Pri tome, mora postojati mogućnost da se preko ulaznih signala zada vrijednost informacije koja će se pamtiti. Takvi sklopovi predstavljaju najprostije sekvencijalne sklopove, koji se nazivaju ***elementarni automati*** ili ***flip-flopovi***.

Osnovna ideja na kojoj se zasniva memoriranje informacija je upotreba ***povratnih sprega***, odnosno puteva kojima se vrijednosti signala vraćaju sa izlaza na ulaz. Prisustvo povratnih sprega je *neophodan uvjet* da bi sklop bio sekvencijalan. Naime, u svim dosadašnjim sklopovima, informacije sa ulaza su stalno napredovale u jednom smjeru, ka izlazu. S druge strane, u prisustvu povratnih sprega, informacije mogu kružiti u *zatvorenom ciklusu, održavajući tako same sebe proizvoljno dugo vremena*. Na primjer, razmotrimo sljedeći spoj:



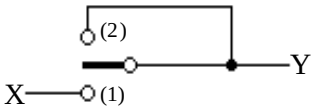
Ukoliko pokušamo utvrditi da li je na izlazu ovog spoja nula ili jedinica, lako se možemo uvjeriti da nijedna od pretpostavki $Y = 0$ ili $Y = 1$ ne dovodi ni do kakvih protivurječnosti, jer se nakon dvije inverzije ponovo dobija ista vrijednost (pri tome se na izlazu iz prvog invertora uvijek nalazi negirana vrijednost od Y). Ukoliko bismo zaista napravili nekoliko ovakvih sklopova, mogli bismo se uvjeriti da je na izlazima nekih od njih nula, a na izlazima preostalih jedinica. Izlaz iz ovog sklopa je *nepredvidljiv*, u smislu da vrijednost izlaza sa podjednakom vjerovatnoćom može biti 0 ili 1, pri čemu tačna vrijednost zavisi od posve *sekundarnih faktora*, koje je praktički nemoguće predvidjeti (npr. koji je od dva invertora prilikom uključenja sklopa prije dobio napajanje električnom energijom, koji od dva invertora brže reagira na promjene, itd.). Vidimo da je prikazani spoj u stanju da proizvoljno dugo održava u sebi jednobitnu informaciju, bez ikakvog dovođenja signala na ulaz. Prikazani spoj spada u klasu tzv. ***bistabilnih sklopova***, jer može izlaz ravnopravno održavati u jednom od dva stabilna stanja (0 ili 1) proizvoljno dugo u odsustvu bilo kakvih informacija na ulazu. Međutim, prikazani spoj je ipak *potpuno neupotrebljiv*, jer ne postoji nikakav način na koji bismo ovom sklopu mogli zadati koju informaciju da čuva (0 ili 1), odnosno ne postoji način da u njega *unesemo informaciju*.

Postoji nekoliko načina da se riješi problem unosa informacija u ovaj sklop, a svi se zasnivaju na tome da se prilikom unosa informacije dejstvo povratne sprege privremeno *prekine*. Najočigledniji način da to uradimo je da doslovce *raskinemo* povratnu spregu za potrebe unosa informacija, kao što je to principijelno pokazano na sljedećoj slici:



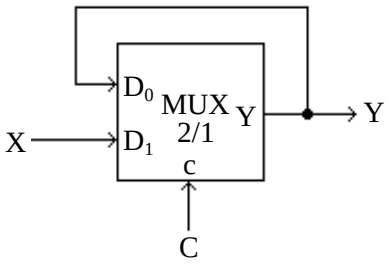
U ovom sklopu je upotrijebljen mehanički preklopnik, koji može biti u položaju označenom sa (1) ili (2). Kada je preklopnik u položaju (1), povratna sprega je raskinuta, tako da vrijedi $Y = X$, odnosno moguće je postaviti vrijednost izlaza Y na proizvoljnu vrijednost 0 ili 1 dovođenjem te vrijednosti na ulaz X. Kada se preklopnik prebaci u položaj (2), raskida se veza sklopa sa ulazom, a prethodno postavljena vrijednost za Y se samoodržava u povratnoj sprezi, i ostaje takva proizvoljno dugo neovisno od vrijednosti ulaza X, sve dok ponovo ne prebacimo preklopnik u položaj (1), kada imamo priliku da unesemo novu informaciju.

Neko bi mogao da postavi pitanje zašto su uopće potrebna dva invertora u petlji, odnosno da li bi se ista funkionalnost mogla ostvariti kao na sljedećoj slici:



Odgovor je odrečan. Naime, ukoliko bi prekidač bio u poziciji (2), tada bi izlaz bukvalno "visio", tj. ne bi bio priključen ni na šta što bi moglo da generira odgovarajuću fizičku veličinu (npr. napon ili struju) koja reprezentira logičku vrijednost "0" ili "1". U slučaju kada u petlji imamo invertore, situacija je sasvim drugačija, s obzirom da su invertori elementi koji imaju vlastito napajanje energijom koja omogućava da se na njihovom izlazu pojavi odgovarajuća vrijednost. Istu stvar možemo slikovito objasniti i ovako. Kako su na fundamentalnom nivou nosioci informacije uvijek neki vidovi materije ili energije, informacija može cirkulirati u krugu, samo uz *dovođenje energije* (analogno, voda koja se nalazi u nekom crijevu neće cirkulirati sama od sebe kroz crijevo ukoliko samo spojimo jedan kraj crijeva sa drugim, već je neophodno imati neki izvor energije poput pumpe). Invertori su *aktivni elementi* u smislu da traže napajanje energijom (tipično električnom) koja obezbjeđuje energiju neophodnu za cirkuliranje informacija u petlji (za razliku od npr. AND i OR kola koja mogu, ali ne moraju biti aktivni elementi). Zapravo, umjesto dva spojena invertora može se koristiti bilo kakav lanac elemenata od kojih je barem jedan aktivan, a koji održava logičku funkciju nepromijenjenom.

Prethodno opisana realizacija memorijskog elementa nije praktična, jer se u njoj javlja mehanički element (preklopnik). Mnogo pogodniju realizaciju dobijamo ukoliko mehanički preklopnik zamijenimo digitalnim sklopom koji simulira njegovu ulogu – multiplekserom:



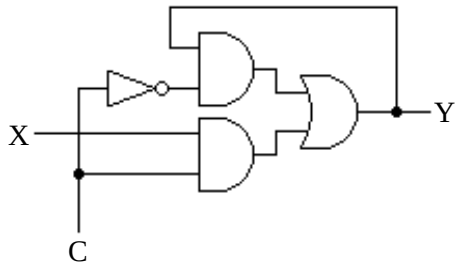
Ovdje smo uveli upravljački ulaz C. Za C = 1, multiplekser prosljeđuje svoj ulaz D_1 na izlaz Y, tako da vrijedi Y = X. Za C = 0 multiplekser prosljeđuje svoj ulaz D_0 na izlaz Y, čime zapravo održava izlaz Y na ranijoj vrijednosti neovisno od ulaza X, što možemo zapisati i kao $Y(n) = Y(n–1)$ ili $Y(n+1) = Y(n)$, gdje smo kao jedinicu vremena usvojili *neku kratku vrijednost*, koja je mnogo kraća od očekivanog vremenskog intervala između promjena vrijednosti ulaznog signala X, ali koja je svakako duža od trajanja prolaska signala kroz petlju (koje je tipično veoma kratko). Dakle, vrijednost izlaza u *narednom trenutku vremena* biće jednaka ili ulazu X (ukoliko je C = 1) ili trenutnoj vrijednosti izlaza (ukoliko je C = 0). Stoga, za ovaj sklop možemo sastaviti sljedeću tablicu, koja opisuje njegov rad:

X	C	Y(*n*)	Y(*n*+1)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Iz ove tablice, korištenjem poznatih metoda minimizacije, možemo očitati sljedeću funkcionalnu zavisnost:

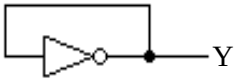
$$Y(n+1) = C\,X \vee \overline{C}\,Y(n)$$

Opisani sklop predstavlja jedan od najjednostavnijih upotrebljivih sekvencijalnih sklopova (i to Mooreovog tipa, ako usvojimo trivijalnu dodjelu $Q(n) = Y(n)$). Ovaj sklop možemo shvatiti kao ***jednobitnu RAM memoriju*** pri čemu se podatak koji želimo da upišemo dovodi na ulaz X, očitava sa izlaza Y, dok ulaz C predstavlja upravljački signal pri čemu $C = 1$ označava upis podatka, dok $C = 0$ označava memoriranje upisanog podatka (do novog upisa). Sklop sa opisanom funkcionalnošću naziva se ***asinhroni D flip-flop*** ili ***D leč*** (engl. latch – reza na vratima), i predstavlja jedan od osovnih elementarnih automata o kojem ćemo detaljnije govoriti nešto kasnije u ovom poglavlju. Za sada samo napomenimo da se asinhroni D flip-flop veoma rijetko realizira prema gore opisanoj shemi, zbog više razloga. Jedan od njih je činjenica da je u standardnoj izvedbi multiplekser 2/1 *mreža sa rizikom*. Naime, u standardnim realizacijama multipleksera 2/1 upravljački signal C na jedno AND kolo ide invertiran, a na drugo AND kolo ide bez inverzije. Stoga, u trenutku kada upravljački signal prelazi sa vrijednosti 1 na 0, zbog neizbježnog kašnjenja koje unosi invertor u toku kratkog vremenskog intervala na oba AND kola se nalazi nula na bar jednom ulazu, tako da izlaz multipleksera može kratkotrajno postati 0, bez obzira na vrijednosti ulaza. Ovo se bolje može vidjeti na sljedećoj slici, na kojoj je nacrtan isti sklop kao na prethodnoj slici, samo sa eksplicitno prikazanom strukturom multipleksera 2/1:

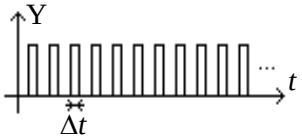


Slijedi da se u ovakvom sklopu na izlazu može javiti kratkotrajna *lažna vrijednost* 0 u trenutku prelaska upravljačkog signala C sa 1 na 0, bez obzira na vrijednost ulaza X ili na raniju vrijednost izlaza Y. Drugim riječima, čitav sklop predstavlja mrežu sa rizikom. Kako elementarni automati predstavljaju osnovu za gradnju svih složenijih sekvencijalnih sklopova, oni bi trebali biti pouzdani koliko god je to moguće, tako da rizični elementarni automati nisu prihvatljivi. Ovaj problem bi se mogao otkloniti ukoliko bismo multiplekser 2/1 realizirali kao mrežu bez rizika (po uputama iz Poglavlja 10.). Međutim, uskoro ćemo vidjeti da postoje praktičnija i jednostavnija rješenja.

Vidjeli smo da je postojanje povratne sprege uvjet za samoodržavanje informacija u sklopu bez prisustva ulaznih informacija. Međutim, prisustvo povratne sprege može biti i uzrok problema. Razmotrimo, na primjer, sljedeći spoj:

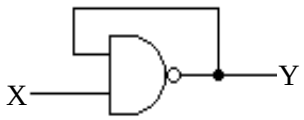


Na prvi pogled djeluje da se ovaj sklop opisuje jednačinom oblika $Y = \overline{Y}$. Međutim, ova jednačina očigledno nema rješenja. S druge strane, ukoliko uzmemo u obzir vremensko kašnjenje Δt koje unosi svaki invertor, ova jednačina se svodi na jednačinu $Y(t+\Delta t) = \overline{Y(t)}$, iz koje slijedi da će se izlaz Y *oscilirati velikom brzinom*, odnosno sam od sebe će se prebacivati sa vrijednosti 0 na vrijednost 1 i obrnuto, pri čemu je trajanje svake od te dvije vrijednosti Δt. Ovo je prikazano na sljedećoj slici:

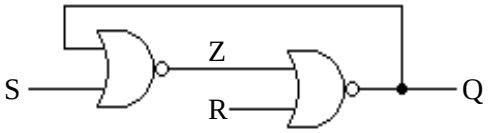


Ovakva pojava naziva se ***nestabilnost***, a sklop na slici predstavlja najjednostavniji primjer nestabilnog sklopa. Nestabilnost može dovesti do velikih problema. Najčešće su sklopovi nestabilni

samo pod određenim okolnostima. Najjednostavniji primjer je sklop na sljedećoj slici, koji za $X = 0$ ima stabilan izlaz $Y = 1$ dok za $X = 1$ oscilira, što je sasvim lako provjeriti:



Nakon neophodnih napomena o opasnostima do kojih može dovesti prisustvo povratne sprege, vraćamo se na proučavanje strukture osnovnih elementarnih automata. Pored razmotrenog načina koji se nije pokazao toliko dobar u praksi, drugi način za unošenje informacija u kaskadni spoj dva invertora zatvoren povratnom spregom ostvaruje se zamjenom invertora sa NAND ili NOR kolima, koja posjeduju dodatne ulaze u odnosu na invertor. Razmotrimo stoga sljedeći spoj:



Ovaj spoj očigledno ima dva ulaza, koji su iz praktičnih razloga, koji će uskoro postati jasni, obilježeni sa R i S. Izlaz sklopa je obilježen sa Q, jer se sklopovi poput ovoga tipično koriste kao *elementi za memoriranje promjenljivih stanja* (odnosno *bita stanja*) u složenijim sekvencijalnim sklopovima. Pomoćna promjenljiva koja predstavlja izlaz iz prvog NOR kola označena je sa Z.

Analizirajmo sada rad ovog spoja. Za $S = 0$ i $R = 1$ imamo $Z \vee R = Z \vee 1 = 1$ bez obzira na to kakva je bila vrijednost promjenljive Z, pa je $Q = 0$, nakon čega imamo $Q \vee S = 0 \vee 0 = 0$, tako da je nova vrijednost promjenljive Z jednaka jedinici. Za $S = 1$ i $R = 0$ imamo $Q \vee S = Q \vee 1 = 1$ bez obzira na to kakva je bila vrijednost izlazne promjenljive Q, pa je $Z = 0$, nakon čega imamo $Z \vee R = 0 \vee 0 = 0$, tako da nova vrijednost promjenljive Q postaje 1. Za $R = S = 0$ ovaj spoj se svodi na obični kaskadni spoj dva invertora zatvoren povratnom spregom (jer je $X \vee 0 = X$), tako da izlaz Q održava samog sebe kroz povratnu spregu, odnosno čuva informaciju koja je bila ranije prisutna (pri tome promjenljiva Z ima negiranu vrijednost u odnosu na Q). Možemo zaključiti da za $S = 0$ i $R = 1$ izlaz Q *bezuvjetno poprima vrijednost* 0 (bez obzira kakav je bio prije), za $S = 1$ i $R = 0$ izlaz Q *bezuvjetno poprima vrijednost* 1, dok za $S = 0$ i $R = 0$ izlaz Q *zadržava vrijednost kakvu je imao ranije*. Može se primijetiti da ukoliko je odmah na početku $R = S = 0$, tada je izlaz Q nepredvidljiv sve dok se jedan od ulaza S ili R ne postavi na jedinicu, nakon čega izlaz dobija jasno definiranu vrijednost koju nakon vraćanja R odnosno S na nulu zadržava do nove postavke R ili S na jedinicu. Postavljanje izlaza na jedinicu obično se naziva ***setovanje***, dok se postavljanje izlaza na 0 naizva ***resetovanje***, odakle i potiču nazivi ulaza S (Set) i R (Reset). Za $S = 1$ sklop se *setuje*, dok se za $R = 1$ sklop *resetuje*. Primijetimo još da je i u sva tri do sada opisana slučaja vrijednost promjenljive Z uvijek komplementarna (negirana) u odnosu na izlaz Q.

U dosadašnjem razmatranju potpuno smo zanemarili kombinaciju $R = S = 1$. Na prvi pogled izgleda da i ova kombinacija bezuvjetno resetuje sklop. Naime, imamo $Z \vee R = Z \vee 1 = 1$, pa je $Q = 0$. Međutim, vrijedi i $Q \vee S = Q \vee 1 = 1$, pa je vrijednost promjenljive Z također 0, za razliku od situacije $S = 0$ i $R = 1$ kada promjenljiva Z ima vrijednost 1. Upravo činjenica da ovdje promjenljive Q i Z nisu više komplementarne uvjetuje da se u slučaju da zadamo kombinaciju $R = S = 1$ sklop počinje veoma čudno ponašati nakon povratka na kombinaciju $R = S = 0$. Naime, nakon povratka na $R = S = 0$, sklop se ponaša kao kaskadni spoj dva invertora zatvoren povratnom spregom, koji mora na izlazima iz prvog i drugog invertora (tačke Z i Q) imati suprotne vrijednosti. Međutim, za vrijeme trajanja $R = S = 1$, vrijedilo je $Z = Q = 0$, tako da se sklop našao u "neprirodnoj" situaciji koje se pokušava osloboditi. Teoretski, sklop u ovoj situaciji čak može početi da *oscilira*. Zaista, ukoliko pretpostavimo da oba NOR kola imaju jednako kašnjenje Δt, tada možemo pisati sistem jednačina:

$$Z(t+\Delta t) = \overline{S(t) \vee Q(t)}\,, \qquad Q(t+\Delta t) = \overline{R(t) \vee Z(t)}$$

Pretpostavimo sada da smo do trenutka $t = 0$ (uključujući i nulu) imali kombinaciju $R = S = 1$ a za $t > 0$ kombinaciju $R = S = 0$. Odavde slijedi da je za $t \le 0$

$$Z(t+\Delta t) = \overline{1 \vee Q(t)} = 0\,, \qquad Q(t+\Delta t) = \overline{1 \vee Z(t)} = 0$$

Specijalno, za $t = 0$ imamo $Z(\Delta t) = Q(\Delta t) = 0$. Za $t > 0$ imamo:

$$Z(t+\Delta t) = \overline{0 \vee Q(t)} = \overline{Q(t)}\,, \qquad Q(t+\Delta t) = \overline{0 \vee Z(t)} = \overline{Z(t)}$$

Odavde neposredno slijedi:

$$Q(2\Delta t) = Q(\Delta t+\Delta t) = \overline{Z(\Delta t)} = \overline{0} = 1 \qquad Z(2\Delta t) = Z(\Delta t+\Delta t) = \overline{Q(\Delta t)} = \overline{0} = 1$$
$$Q(3\Delta t) = Q(2\Delta t+\Delta t) = \overline{Z(2\Delta t)} = \overline{1} = 0 \qquad Z(3\Delta t) = Z(2\Delta t+\Delta t) = \overline{Q(2\Delta t)} = \overline{1} = 0$$
$$Q(4\Delta t) = Q(3\Delta t+\Delta t) = \overline{Z(3\Delta t)} = \overline{0} = 1 \qquad Z(4\Delta t) = Z(3\Delta t+\Delta t) = \overline{Q(3\Delta t)} = \overline{0} = 1$$

Vidimo da sklop zaista oscilira. Ovaj dokaz smo izveli uz pretpostavku da oba NOR kola imaju jednaka kašnjenja. U praksi, zbog uvijek prisutnih minornih razlika u kašnjenjima pojedinih logičkih kola sklop neće trajno oscilirati nakon prelaska sa $R = S = 1$ na $R = S = 0$, nego će, nakon eventualno nekoliko oscilacija, zauzeti jedno od dva stabilna stanja $Q = 0$ ili $Q = 1$, pri čemu je praktično nemoguće prognozirati koje. Ovu pojavu nazivamo ***trkom*** (engl. *racing*). U svakom slučaju, možemo reći da će nakon prelaska sa kombinacije $R = S = 1$ na $R = S = 0$ izlaz iz sklopa biti *nepredvidljiv*. Zbog toga se kombinacija $R = S = 1$ naziva ***zabranjenom kombinacijom***, i ne koristi se.

Opisani sklop naziva se ***RS flip-flop*** (preciznije bi trebalo ***asinhroni RS flip-flop***), ***RS bistabil*** ili ***RS leč***. Njegovo ponašanje očito možemo predstaviti sljedećom funkcionalnom tablicom:

R	S	FUNKCIJA
1	0	RESETOVANJE
0	1	SETOVANJE
0	0	PAMĆENJE
1	1	NE KORISTI SE

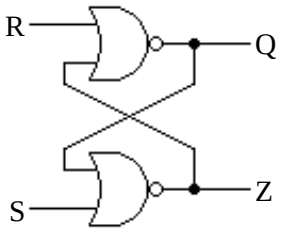
Upravo pamćenje koje se ostvaruje za $R = S = 0$ čini ovaj sklop *sekvencijalnim sklopom*, odnosno njegov rad *nije moguće posmatrati neovisno od prošlosti*. Stoga za njega možemo sastaviti ovakvu tablicu prelaza:

R	S	$Q(n)$	$Q(n+1)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	x
1	1	1	x

Iz ove tablice lako možemo očitati sljedeću funkciju prelaza:

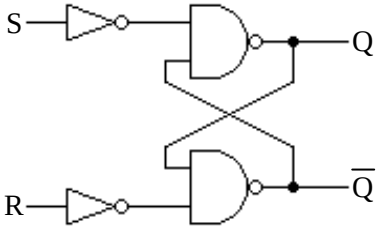
$$Q(n+1) = S \vee \overline{R}\, Q(n) \qquad \text{pod uvjetom} \qquad R\,S \neq 1$$

U literaturi se shema RS flip-flopa obično crta na nešto drugačiji način, prikazan na sljedećoj slici, koji posjeduje veći stepen simetrije, i u kojem je promjenljiva Z također izvedena kao izlaz:



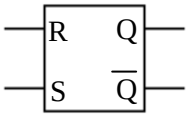
Izlaz Z se naziva ***komplementarni izlaz***, i obično se obilježava sa $\overline{Q}$, s obzirom da smo već istakli da pri normalnom radu (tj. ako zabranimo kombinaciju $R = S = 1$) promjenljiva Z uvijek ima *negiranu vrijednost* u odnosu na Q. Na ovaj način nam je na izlazu iz sklopa pored promjenljive Q prisutna i njena negacija, što često pojednostavljuje dizajn složenijih sekvencijalnih sklopova.

RS flip-flop se može izvesti i pomoću NAND kola (umjesto NOR kola), ali u tom slučaju je potrebno invertirati ulaze R i S da bi se sklop ponašao istovjetno kao i RS flip-flop baziran na NOR kolima. Pored toga, ulazi S i R trebaju zamijeniti mjesta. Drugim riječima, shema RS flip-flopa baziranog na NAND kolima ima sljedeći oblik:

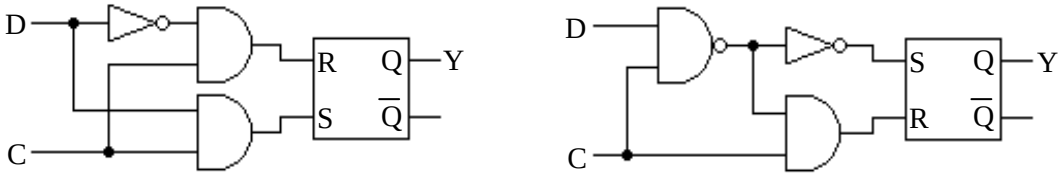


Mada ova varijanta djeluje složenije od prethodne, ona se više koristi, jer se vrlo efektno kombinira sa dodatnim elementima pri realizaciji drugih tipova flip-flopova, što će biti ilustrirano kasnije, pri opisu najpovoljnije realizacije D flip-flopa (pogledajte također i zadatke iza ovog poglavlja). Čitateljima i čitateljkama se ostavlja za vježbu da pokažu da ovaj sklop zaista obavlja funkcionalnost RS flip-flopa.

RS flip-flop se rijetko koristi kao samostalan sklop. Međutim, on je sastavni dio gotovo svih drugih elementarnih automata. Zbog toga se RS flip-flop često crta kao poseban simbol, kao na sljedećoj slici:



Na primjer, ranije opisani asinhroni D flip-flop najčešće se realizira upravo pomoću RS flip-flopa, s obzirom da je RS flip-flop *stabilan i pouzdan sklop, bez rizika*, sve dok garantiramo da mu se na oba ulaza neće istovremeno pojaviti jedinice. Sljedeća slika prikazuje dvije najčešće realizacije asinhronog D flip-flopa pomoću RS flip-flopa:



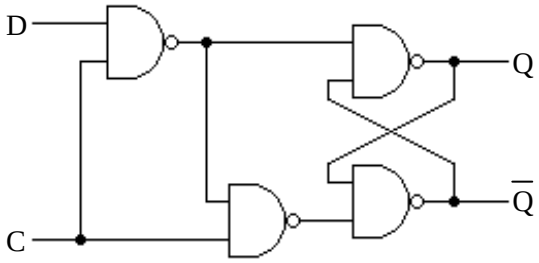
Na ovoj slici smo ulaz za podatke umjesto sa X obilježili sa D (od engl. *data*), kako se to obično radi u slučaju D flip-flopa. Mada je posve lako prostom logičkom analizom pokazati da ovi sklopovi zaista rade kao asinhroni D flip-flopovi (posmatrajući sve moguće vrijednosti za C i D), mi ćemo se u to uvjeriti analitičkim putem. Naime, ponašanje RS flip-flopa opisano je jednačinom

$$Q(n+1) = S \vee \overline{R}\, Q(n) \qquad \text{pod uvjetom} \qquad R\,S \neq 1$$

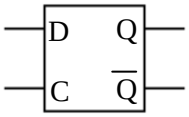
Na obje gornje šeme imamo $R = C\overline{D}$ i $S = C\,D$ (mada to za šemu sa desne strane nije očigledno na prvi pogled), tako da je uvjet $R\,S \neq 1$ očigledno ispunjen. Dalje je:

$$\begin{aligned} Y(n+1) = Q(n+1) &= S \vee \overline{R}\,Q(n) = C\,D \vee \overline{C\overline{D}}\,Q(n) = C\,D \vee (\overline{C} \vee D)\,Q(n) = \\ &= C\,D \vee \overline{C}\,Q(n) \vee D\,Q(n) = C\,D \vee \overline{C}\,Q(n) \vee (C \vee \overline{C})\,D\,Q(n) = \\ &= C\,D \vee \overline{C}\,Q(n) \vee C\,D\,Q(n) \vee \overline{C}\,D\,Q(n) = [C\,D \vee C\,D\,Q(n)] \vee [\overline{C}\,Q(n) \vee \overline{C}\,D\,Q(n)] = \\ &= C\,D \vee \overline{C}\,Q(n) = C\,D \vee \overline{C}\,Y(n) \end{aligned}$$

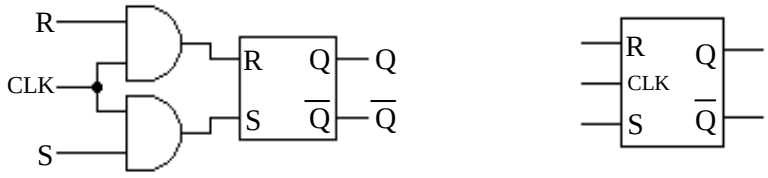
Vidimo da smo dobili jednačinu koja opisuje rad asinhronog D flip-flopa, čime smo pokazali da ovi sklopovi zaista rade kao asinhroni D flip-flopovi. Pored odsustva rizika, ovakve izvedbe asinhronih D flip-flopova imaju i tu prednost što nam je na raspolaganju i negirana vrijednost izlaza, sa komplementarnog izlaza RS flip-flopa. Mada shema sa desne strane djeluje manje prirodno u odnosu na shemu sa lijeve strane, ona se češće koristi, jer se jako lijepo kombinira sa izvedbom RS flip-flopa baziranu na NAND kolima tvoreći sljedeću shemu, u kojoj se nalaze samo četiri NAND kola, što je tehnološki jako pogodno, jer ne miješamo različite tipove logičkih kola u istom sklopu (zapravo, ovo je tehnološki najpovoljnija izvedba asinhronog D flip-flopa):



Asinhroni D flip-flop je čest sastavni dio složenijih sekvencijalnih sklopova. Stoga se za njega također uvodi poseban simbol, kao na sljedećoj slici:



Veliki problem prilikom korištenja RS flip-flopova u praksi predstavlja činjenica da će se čak i pojava smetnje u vidu kratkotrajne jedinice na R ili S ulazu flip-flopa u trenutku dok on "miruje" (tj. dok je $R = S = 0$) odraziti na promjenu stanja na izlazu, koja će ostati trajno zapamćena sve do nove promjene vrijednosti na ulazima. Dakle, čak i kratkotrajna lažna vrijednost na nekom od ulaza RS flip-flopa može da ostavi dugoročne posljedice na rad sklopa. Zbog toga je veoma opasno izlaze mreža sa rizikom priključivati na ulaze RS flip-flopa. Da bi se izbjegao ovaj problem, a ujedno i ostvarile neke druge pogodnosti, u RS flip-flopove uvodi se i treći ulaz, tzv. ***taktni ulaz***, obilježen sa CLK (od engl. *clock*). Struktura ovako modificiranog flip-flopa kao i njegov skraćeni simbol prikazani su na sljedećoj slici:



Očigledno, za $CLK = 1$ sklop radi kao obični RS flip-flop, dok za $CLK = 0$ sklop "miruje", tj. čuva zatečenu informaciju bez obzira na vrijednosti ulaza R i S. Stoga, ne bi bilo teško sastaviti tablicu prelaza ovog sklopa, u kojoj bi nova vrijednost izlaza $Q(n+1)$ zavisila od R, S, CLK i $Q(n)$. Međutim, ulaz CLK nije predviđen da se koristi kao običan ulaz. Na ovaj ulaz se dovodi povorka kratkotrajnih impulsa, tzv. ***taktnih*** ili ***klok impulsa***, kao na sljedećoj slici:

Klok impulsi se obično dovode u *pravilnim vremenskim intervalima*, a njihov broj u jedinici vremena nazivamo ***frekvencija taktnih impulsa*** i izražavamo je u *hercima* (1 Hz = 1 impuls u sekundi). Očigledno je da sklop reagira na ulaze R i S samo u trenucima nailaska klok impulsa, čime je spriječena mogućnost da kratkotrajne lažne vrijednosti na R ili S ulazima u periodu između dva klok impulsa budu "uhvaćene" u sklop. Drugim riječima, ovaj sklop je *potpuno inertan* u periodu između dva klok impulsa, što se savršeno uklapa u definiciju diskretnih sklopova, koji obrađuju informacije uzete *samo u tačno određenim trenucima vremena*. Ti trenuci vremena u ovom slučaju su određeni trenucima nailaska klok impulsa. Stoga možemo podrazumijevati da se rad taktovanog RS flip-flopa opisuje istom tablicom i istom jednačinom kao i rad običnog RS flip-flopa, ali pri čemu je jedinica mjerenja vremena precizno određena periodom nailaska klok impulsa. Na taj način se CLK uopće ne posmatra kao ulazna promjenljiva, već prosto kao sinhronizacioni ulaz na koji se dovode klok impulsi, čiji je smisao da tačno preciziraju trenutke u kojima eventualno dolazi do promjene stanja sklopa. Drugim riječima, klok impulsi daju sklopu "osjećaj za vrijeme". Treba napomenuti da klok impulsi trebaju biti dovoljno kratki da bi se izbjeglo da se vrijednosti ulaza eventualno promijene za vrijeme trajanja klok impulsa.

Vrijedi napomenuti da se i u asinhronom D flip-flopu ulaz C također može iskoristiti za dovođenje klok impulsa. Tada D flip-flop prosto pamti uzorke ulaznog signala D kakvi su bili u trenucima nailaska klok impulsa (pod uvjetom da su klok impulsi dovoljno kratki da se ulaz D ne mijenja tokom njihovog trajanja), sve do nailaska novog klok impulsa. U tom slučaju se C ne posmatra kao ulazna promjenljiva, već samo kao sinhronizacioni ulaz.

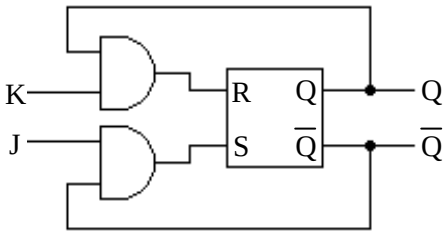
Činjenica da je kombinacija $R = S = 1$ zabranjena u RS flip-flopu značajno ograničava njegovu primjenu. Zbog toga su predložene razne modifikacije RS flip-flopa koje imaju precizno definirano ponašanje za $R = S = 1$. Tako se, na primjer, ***RS flip-flop sa dominantnim S ulazom*** za $R = S = 1$ *setuje* (kao za $R = 0$, $S = 1$), ***RS flip-flop sa dominantnim R ulazom*** se za istu kombinaciju ulaza *resetuje*, dok ***RS flip-flop sa nedominantnim ulazima*** za ovu kombinaciju ulaza čuva zatečeno stanje (kao za $R = S = 0$). Međutim, sve ove varijante nisu se pokazale previše korisne u praktičnim primjenama. Kao ubjedljivo najkorisnija modifikacija RS flip-flopa pokazala se modifikacija koju je predložio inžinjer *Jack Kilby* iz firme *Texas Instruments*, tvorac prvog integrisanog kola. Ova modifikacija, nazvana ***JK flip-flop*** (po inicijalima tvorca), ima dva ulaza J i K, koji su pandan ulazima S i R kod RS flip-flopa. JK flip-flop radi isto kao i RS flip-flop, osim što za $J = K = 1$ *invertira* (*komplementira*) *zatečeno stanje*. Drugim riječima, JK flip-flop se može opisati sljedećom tablicom prelaza:

J	K	Q(n)	Q(n+1)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Iz ove tablice možemo očitati da se rad JK flip-flopa može predstaviti sljedećom jednačinom:

$$Q(n+1) = J\,\overline{Q(n)} \vee \overline{K}\,Q(n)$$

Ovakva funkcionalnost se može postići samo uvođenjem *dodatnih povratnih sprega* u RS flip-flop. Konkretnije, JK flip-flop se *principijelno* može dobiti iz RS flip-flopa pomoću sljedeće sheme:

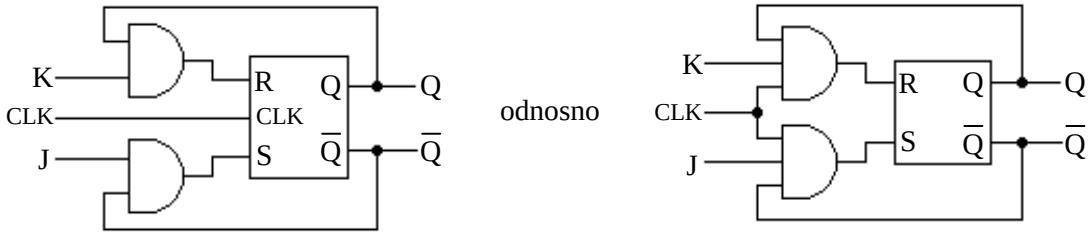


Posve je lako izvršiti logičku analizu ovog sklopa. Za $J = K = 0$ imamo $R = S = 0$, tako da sklop čuva zatečeno stanje. Za $J = 0, K = 1$ imamo $S = 0$, dok R zavisi od Q. Za $Q = 1$ imamo $R = 1$, dok za $Q = 0$ imamo $R = 0$. U oba slučaja, nova vrijednost promjenljive Q biće 0, dakle sklop se *resetuje*. Slično se pokazuje da se za $J = 1, K = 0$ sklop *setuje*. Razmotrimo još najinteresantniju kombinaciju $J = K = 1$. Ukoliko je bilo $Q = 0$, imaćemo $R = 0$, $S = 1$, pa će se sklop *setovati*. Ukoliko je bilo $Q = 1$, imaćemo $R = 1$, $S = 0$, pa će se sklop *resetovati*. U svakom slučaju, sklop će *invertirati svoje stanje*, kako je i predviđeno. U sve ovo mogli smo se uvjeriti i analitički na sljedeći način:

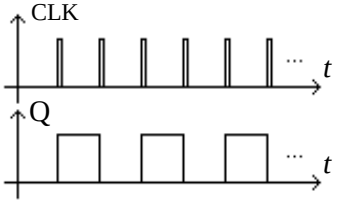
$$R = K\,Q(n), \qquad S = J\,\overline{Q(n)}, \qquad R\,S = J\,K\,Q(n)\,\overline{Q(n)} = 0 \neq 1$$

$$Q(n+1) = S \vee \overline{R}\,Q(n) = J\,\overline{Q(n)} \vee \overline{K\,Q(n)}\,Q(n) = J\,\overline{Q(n)} \vee \overline{K}\,\overline{Q(n)}\,Q(n) =$$
$$= J\,\overline{Q(n)} \vee [\overline{K} \vee \overline{Q(n)}]\,Q(n) = J\,\overline{Q(n)} \vee \overline{K}\,Q(n) \vee \overline{Q(n)}\,Q(n) = J\,\overline{Q(n)} \vee \overline{K}\,Q(n)$$

Međutim, prethodna realizacija posjeduje jedan veliki problem. Pretpostavimo da je $Q = 0$, i da na ulaz dovedemo kombinaciju $J = K = 1$. Nakon veoma kratkog vremena (koje je određeno kašnjenjima signala pri prolasku kroz logička kola od ulaza do izlaza) izlaz će se promijeniti na $Q = 1$. Ukoliko je u tom trenutku još uvijek $J = K = 1$ (a gotovo sigurno jeste), izlaz će se opet nakon kratkog vremena promijeniti na $Q = 0$. Drugim riječima, sklop će *oscilirati* velikom brzinom sve dok je $J = K = 1$, odnosno za ovu ulaznu kombinaciju sklop je *nestabilan*! Ovaj problem se može riješiti jedino uvođenjem *taktnih impulsa*, odnosno u slučaju JK flip-flopa taktni impulsi su *neophodni*. Taktne impulse možemo uvesti tako što ćemo zamijeniti obični RS flip-flopom taktovanim RS flip-flopom:



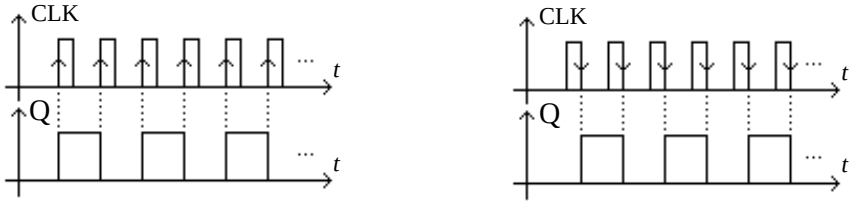
U ovom slučaju sklop može mijenjati svoje stanje samo u trenucima nailaska klok impulsa. Prema tome, ovakav sklop pri dugotrajnom prisustvu ulaznih signala $J = K = 1$ neće mijenjati svoje stanje *nekontrolirano*, velikom brzinom, nego samo u *trenucima nailaska klok impulsa*, kao što je prikazano na sljedećoj slici, koja prikazuje ponašanje izlaza kada je $J = K = 1$:



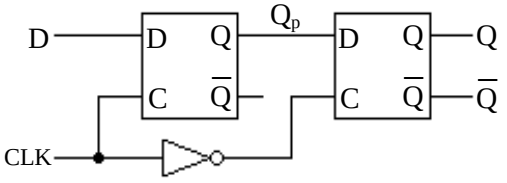
Ovdje se više ne radi o nestabilnosti, nego o *željenom ponašanju sklopa*, koje je pod potpunom kontrolom. Zapravo je i cilj bio da sklop mijenja svoje stanje za $J = K = 1$, ali ne nekontrolirano, neodređen broj puta, nego tačno *jednom* u *jednoj vremenskoj jedinici*. I zaista, ukoliko ulaz $J = K = 1$ traje kraće od jedne vremenske jedinice (tj. od razmaka između dva klok impulsa), izlaz će promijeniti vrijednost tačno jednom, kao što i treba da bude.

Taktovani (sinhroni) JK flip-flop se obilježava isto kao i taktovani RS flip-flop, uz razliku što su ulazi obilježeni sa J i K umjesto R i S. Međutim, prethodna izvedba taktovanog flip-flopa također posjeduje veoma bitan nedostatak, koji je čini praktično neupotrebljivom u praksi. Naime, ova izvedba je veoma osjetljiva na trajanje taktnih impulsa. Ukoliko taktni impulsi traju predugo, znatno duže od trajanja prebacivanja flip-flopa, izlaz će za J = K = 1 nekoliko puta promijeniti svoju vrijednost prije nego što taktni impuls prestane, što je potpuno neprihvatljivo, tim prije što je nemoguće predvidjeti kolika će nakon toga biti vrijednost izlaza. S druge strane, ukoliko taktni impulsi traju prekratko (znatno kraće od trajanja prebacivanja flip-flopa), sklop neće stići da se prebaci u novo stanje. Prema tome, taktni impulsi bi trebalo da traju otprilike koliko traje prebacivanje flip-flopa iz jednog u drugo stanje, što je tehnološki veoma teško postići.

Izlaz iz opisanog problema nađen je uvođenjem ***ivično okidanih taktovanih flip-flopova***. Kod njih je promjena stanja sklopa moguća ne tokom čitavog trajanja klok impulsa, nego samo u trenucima *prelaska* klok impulsa sa jedne na drugu vrijednost, *bez obzira na to koliko sam klok impuls traje*. Na taj način uvodimo neovisnost ponašanja sklopa od trajanja klok impulsa. Pri tome, razlikujemo flip-flopove koji se okidaju ***uzlaznom*** (***rastućom***) ***ivicom***, i flip-flopove koji se okidaju ***silaznom*** (***opadajućom***) ***ivicom*** klok impulsa. Prvi reagiraju samo u trenucima promjene klok impulsa sa 0 na 1, dok drugi reagiraju samo u trenucima promjene klok impulsa sa 1 na 0. Sljedeća slika pokazuje kako bi respektivno trebali da reagiraju JK flip-flopovi okidani uzlaznom odnosno silaznom ivicom pri dugotrajnom prisustvu ulaznog signala J = K = 1:



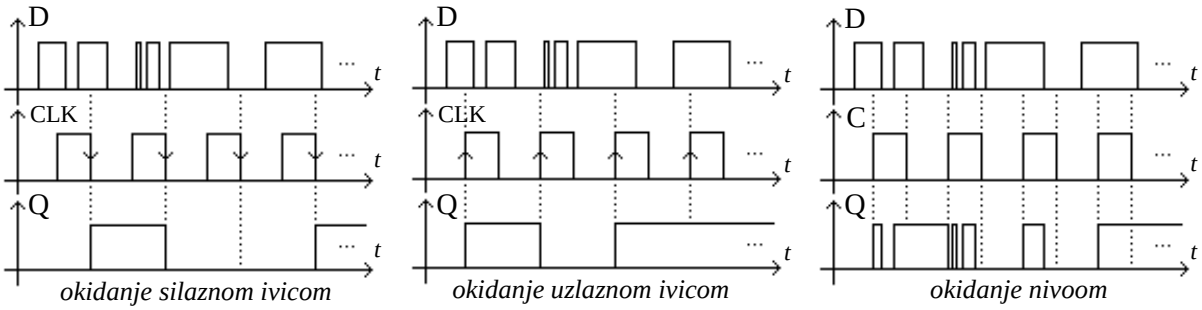
Ivično okidani taktovani flip-flopovi imaju znatno složeniju strukturu nego obični taktovani flip-flopovi (za koje se još kaže da su ***okidani nivoom***, s obzirom da reagiraju sve dok su taktni impulsi na nivou logičke jedinice). Najjednostavniju strukturu imaju *ivično okidani D flip-flopovi*, pa ćemo prvo njih razmotriti. Na sljedećoj slici je prikazana struktura *D flip-flopa okidanog silaznom ivicom* izvedenog u tzv. ***master-slejv konfiguraciji*** (engl. *master-slave* – gospodar-sluga):



Vidimo da se ovaj spoj zapravo sastoji od dva obična flip-flopa, od kojih ćemo prvi zvati *master*, a drugi *slejv*. Radi lakše analize izlaz iz *mastera* obilježili smo sa Q_p.Vidimo da za vrijeme dok je CLK = 1 imamo $Q_p = D$. Međutim, ta vrijednost ne dolazi do izlaza iz sklopa, jer za to vrijeme *slejv* ima C = 0, tako da prosto čuva zatečeno stanje. U trenutku kada taktni impuls prestane, tj. kada postane CLK = 0, *master* čuva zatečeno stanje. Međutim, u tom trenutku *slejv* dobija C = 1, tako da prosljeđuje zatečenu vrijednost promjenljive Q_p na izlaz (koja je jednaka vrijednosti ulaza D kakva je bila *neposredno prije prestanka klok impulsa*). Dalje promjene ulaza D neće se odraziti na vrijednosti izlaza bez obzira što *slejv* ima C = 1, jer tada *master* ima C = 0, tako da se vrijednost promjenljive Q_p (koja se prosljeđuje na izlaz) ne mijenja, sve do ponovnog nailaska klok impulsa. Kada naiđe novi klok impuls, *slejv* dobija C = 0 a *master* C = 1, tako da se ciklus ponavlja. Drugim riječima, za vrijeme *trajanja* klok impulsa *master* u sebe prima vrijednost sa ulaza D, ali ona se ne prosljeđuje dalje na izlaz. Tek kada klok impuls prestane, *slejv* prenosi na izlaz posljednju primljenu vrijednost u *master*. Očigledno sklop može da mijenja stanje na izlazu *samo u trenutku kada klok signal prelazi sa jedinice na nulu* (tj. samo na *silaznu ivicu klok signala*). Pri tome je izlaz jednak vrijednosti ulaza D kakva je bila u trenutku nailaska silazne ivice klok signala.

Veoma je lako prepraviti taktovani D flip-flop okidan opadajućom ivicom u D flip-flop okidan rastućom ivicom. Za tu svrhu dovoljno je samo *invertirati* CLK signal. Simboli taktovanih flip-flopova koji se okidaju rastućom odnosno opadajućom ivicom dati su na sljedećoj slici. Trokutić kraj oznake ulaza označava *ivično okidanje*, pri čemu dodatni kružić označava okidanje *negativnom ivicom*.

Može se primijetiti da je ponašanje ivično okidanih D flip-flopova potpuno drugačije od ponašanja *asinhronog D flip-flopa* koji pušta ulaz D na izlaz cijelo vrijeme dok je C ulaz na jedinici. Zbog toga se za asinhroni D flip-flop kaže da je ***prozračan*** (engl. *transparent*). Stoga se ponašanje ivično okidanih flip-flopova bolje uklapa u definiciju digitalnih sistema, koji bi trebali da reagiraju samo u *tačno određenim trenucima vremena*. Radi usporedbe, na sljedećoj slici su prikazana ponašanja ivično okidanih D flip-flopova i asinhronog D flip-flopa na proizvoljno pretpostavljeni signal na D ulazu:



Ukoliko bismo CLK ulaz shvatili kao ulaznu promjenljivu, tada bi za opisivanje rada ivično okidanih flip-flopova bila potrebna prilično velika tablica. Naime, kako ivično okidani flip-flopovi reagiraju samo u trenucima *promjene* klok impulsa, novo stanje Q(n+1) bi zavisilo kako od *trenutne*, tako i od *prethodne* vrijednosti klok impulsa, tako da bi Q(n+1) bilo izraženo kao funkcija od D(n), CLK(n), CLK(n–1) i Q(n). Tako bi, na primjer, flip-flop okidan opadajućom ivicom reagirao na ulazni signal D samo kada je CLK(n–1) = 1 i CLK(n) = 0. Čitateljima i čitateljkama se ostavlja za vježbu da sastave ovakvu tablicu za taktovani D flip-flop okidan silaznom ivicom, i da pokažu da vrijedi

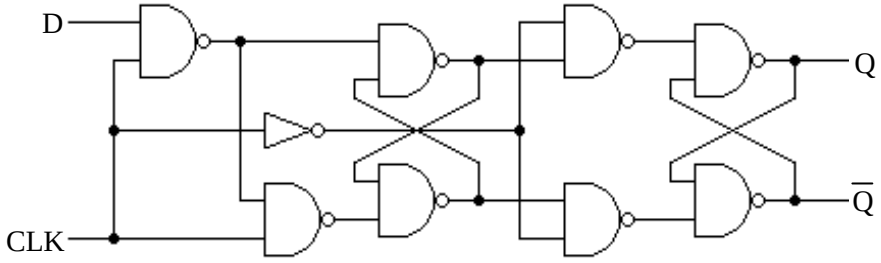
$$Q(n+1) = D(n)\,\overline{CLK(n)}\,CLK(n-1) \vee Q(n)\,[CLK(n) \vee \overline{CLK(n-1)}]$$

Međutim, već smo rekli da se u praksi CLK gotovo nikada ne posmatra kao ulazna promjenljiva, već isključivo kao signal koji određuje *sistemsko vrijeme*, odnosno trenutke u kojima se posmatraju vrijednosti ulaza i izlaza. Stoga, ukoliko usvojimo da nam CLK ulaz samo diktira vrijeme, tada je buduća vrijednost izlaza prosto jednaka vrijednosti ulaza D u trenutku nailaska ivice (rastuće odnosno opadajuće) klok signala, tako da se rad ivično okidanog D flip-flopa opisuje trivijalnom jednačinom

$$Q(n+1) = D$$

Pri tome se podrazumijeva da je jedinica mjerenja vremena jednaka *razmaku između dvije uzastopne rastuće* (*odnosno opadajuće*) *ivice klok impulsa*.

Iz izloženog se može primijetiti da je građa ivično okidanih D flip-flopova prilično složena. Na primjer, jedna od mogućih izvedbi preko NAND kola, koja neposredno slijedi iz principijelne strukture iskazane preko dva asinhrona D flip-flopa u master-slejv konfiguraciji, prikazana je na sljedećoj slici:

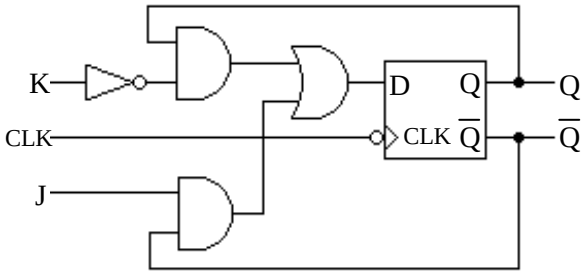


Treba napomenuti da postoje i ekonomičnija rješenja ivično okidanih D flip-flopova, koja nisu zasnovana na master-slejv konfiguraciji. Na primjer, u literaturi se često susreće shema ivično okidanog D flip-flopa sa svega 6 NAND ili NOR kola (pogledajte npr. Zadatak 16.28). Međutim, takve sheme ne slijede neposredno iz strukture iskazane preko dva asinhrona D flip-flopa, i stoga je mnogo teže objasniti (i shvatiti) kako one zapravo rade.

Ivično okidani JK flip-flop nije teško napraviti pomoću ivično okidanog D flip-flopa (bilo u master-slejv konfiguraciji, bilo u nekoj drugoj izvedbi). Naime, kako se funkcionalnost ivično okidanog D flip-flopa opisuje jednačinom $Q(n+1) = D$, to se funkcionalnost JK flip-flopa opisana jednačinom

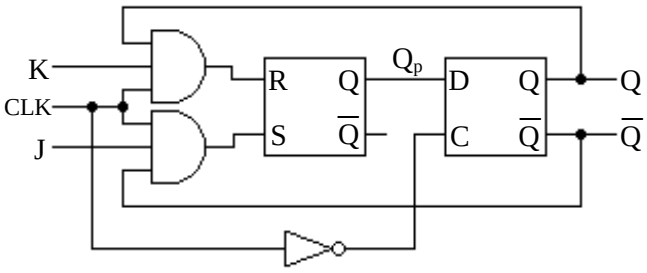
$$Q(n+1) = J\,\overline{Q(n)} \vee \overline{K}\,Q(n)$$

može jednostavno postići ukoliko obezbjedimo da je $D = J\,\overline{Q(n)} \vee \overline{K}\,Q(n)$. Pri tome će jasno i dobijeni JK flip-flop biti ivično okidan s obzirom da je i polazni D flip-flop ivično okidan. Stoga, sljedeća slika prikazuje strukturu JK flip-flopa okidanog silaznom ivicom:



Ivično okidani JK flip-flopovi također imaju svoje simbole, koji su analogni simbolima ivično okidanih D flip-flopova, a prikazani su na sljedećoj slici:

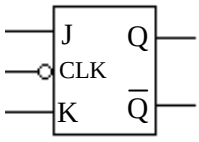
Vidimo da je ivično okidani JK flip-flop veoma složene građe, s obzirom da je već struktura ivično okidanog D flip-flopa prilično složena. Zbog toga se umjesto pravih ivično okidanih JK flip-flopova ponekad koristi i jedno polu-rješenje nazvano ***master-slejv JK flip-flop*** (skraćeno MS JK FF). Njegova je građa jednostavnija od građe taktovanih JK flip-flopova (posmatrano na nivou logičkih kola), a principijelno je prikazana na sljedećoj slici:



Svakome ko je shvatio kako radi ivično okidani D flip-flop i obični JK flip-flop neće biti teško da analizira kako radi ovaj sklop. Cijela ideja je, kao i kod ivično okidanog D flip-flopa, da se informacija o stanjima ulaza J i K prikupi u pomoćnu promjenljivu Q_p za vrijeme trajanja klok impulsa, tj. dok je CLK = 1, koja će biti prosljeđena na izlaz tek kada postane CLK = 0, tj. nakon opadajuće ivice klok impulsa. Dodatne povratne sprege služe da, kao kod običnog JK flip-flopa, obezbijede komplementiranje izlaza pri ulaznoj kombinaciji J = K = 1, u pravom trenutku (tj. u trenutku nailaska opadajuće ivice klok impulsa). Međutim, ovaj sklop se ipak ne ponaša kao pravi JK flip-flop okidan opadajućom ivicom, mada on može mijenjati svoj izlaz jedino u trenucima nailaska opadajuće ivice klok signala. Problem je u tome što je, za razliku od ivično okidanog D flip-flopa, *master* u ovom spoju RS flip-flop (a ne D flip-flop), koji ima nezgodnu osobinu da trajno pamti čak i kratkotrajne impulse dovedene na R ili S ulaz. Zbog toga će čak i kratkotrajni impulsi na J ili K ulazu za vrijeme trajanja klok impulsa biti "uhvaćeni" u *master* i proslijeđeni na izlaz nakon opadajuće ivice klok impulsa, čak i ukoliko su *prestali da djeluju prije nailaska opadajuće ivice klok impulsa*. Drugim riječima, bez obzira na to što sklop može mijenjati svoj izlaz samo u trenucima nailaska opadajuće ivice klok impulsa (čime odaje iluziju da reagira samo na opadajuću ivicu klok impulsa), on u sebe "bilježi" šta se dešavalo čitavo vrijeme tokom trajanja klok impulsa, a ne samo u trenutku nailaska opadajuće ivice, kao što to radi pravi ivično okidani JK flip-flop. Zbog toga je veoma opasno na ulaze MS JK flip-flopa priključivati izlaze iz rizičnih mreža, koje mogu generirati kratkotrajne lažne vrijednosti.

Opisana nezgodna osobina MS JK flip-flopa naziva se ***hvatanje jedinica*** (engl. *ones catching*) i na nju je naročito važno upozoriti, jer se u mnogobrojnoj literaturi ona često prešućuje, tako da se MS JK flip-flop opisuje kao *pravi* ivično okidani flip-flop, što očigledno nije u potpunosti tačno, i može dovesti do veoma opasnih zabluda. Stoga MS JK flip-flopove treba koristiti sa *kranjim oprezom*. Bitno je napomenuti da je D flip-flop u master-slave konfiguraciji pravi ivično okidani flip-flop, koji se potpuno isto ponaša kao i ivično okidani D flip-flopovi realizirani na neki drugi način, dok ista stvar ne vrijedi za JK flip-flopove. Naime, JK flip-flopovi u master-slave konfiguraciji ne ponašaju se u potpunosti onako kao što bi se trebali ponašati pravi ivično okidani flip-flopovi. Pravi ivično okidani JK flip-flop moguće je napraviti jedino indirektno iz ivično okidanog D flip-flopa (bez obzira kako je on izveden), na način koji smo već objasnili. Interesantno je napomenuti i to da se master-slave konfiguracija ne može napraviti od obična dva JK flip-flopa na način analogan kako je master-slave konfiguracija napravljena od dva D flip-flopa, niti je moguće napraviti master-slave konfiguraciju od jednog JK i jednog D flip-flopa (pogledajte, na primjer, Zadatak 16.33).

Simbol MS JK flip-flopa prikazan je na sljedećoj slici. Nedostatak trokutića kraj CLK ulaza ukazuje da se ne radi o pravom ivičnom okidanju, dok kružić (koji standardno označava negaciju) govori da sklop reagira tek nakon *prestanka* klok impulsa. Nesretna je okolnost što se u brojnoj literaturi konvencija o obilježavanju flip-flopova veoma često ne poštuje dosljedno, tako da iz simbola flip-flopa nije moguće zaključiti radi li se o običnom, ivično okidanom ili MS flip-flopu, već informaciju o tome treba tražiti u propratnom tekstu.



Do sada smo opisali razne vrste flip-flopova, od kojih su neke poslužile tek kao gradivni elementi za složenije flip-flopove (nakon što su ustanovljene njihove mane). Moguće je napraviti i ivično okidane ili master-slejv RS flip-flopove, ali su oni zbog postojanja zabranjene kombinacije R = S = 1 inferiorni u odnosu na odgovarajuće JK flip-flopove. Možemo reći da se u praksi od svih asinhronih flip-flopova u praksi gotovo isključivo susreće jedino asinhroni D flip-flop (iznimno se ponekad koristi i RS flip-flop). JK flip-flop se isključivo koristi ili u ivično okidanoj sinhronoj varijanti (okidanje silaznom ivicom se susreće češće) ili u master-slejv varijanti (koja spada negdje u "zonu sumraka" između sinhronih i asinhronih flip-flopova). Od ivično okidanih flip-flopova intenzivno se koriste se i D flip-flopovi, pri čemu se kod njih češće susreće okidanje uzlaznom ivicom. Treba još napomenuti da u literaturi vlada pravo šarenilo ne samo u načinu označavanja, nego i u *imenovanju* pojedinih vrsta flip-flopova. Tako,

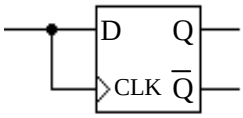
neki autori pod flip-flopovima podrazumijevaju isključivo *ivično okidane flip-flopove*, dok sve ostale flip-flopove (asinhrone i nivoom okidane) nazivaju *lečevima* (tako da govore, recimo, o D leču, ili o RS leču).

U teoretskim razmatranjima, a naročito u *teoriji brojačkih sklopova*, često se pokazuje korisnim uvesti i još jednu vrstu flip-flopa, nazvanu T flip-flop (slovo T potiče od engl. *trigger* ili *toggle*). Ovaj flip-flop ima jedan ulaz označen sa T. Za $T = 0$ flip-flop *čuva zatečeno stanje*, dok za $T = 1$ *komplementira zatečeno stanje*. Drugim riječima, tablica prelaza za T flip-flop izgleda ovako:

T	Q(n)	Q(n+1)
0	0	0
0	1	1
1	0	1
1	1	0

Rad T flip-flopa opisuje se jednačinom $Q(n+1) = T \oplus Q(n)$. Međutim, T flip-flopovi se nikada ne proizvode kao samostalni elementi, jer se neposredno vidi da T flip-flop nije ništa drugo nego JK flip-flop sa J i K ulazima *spojenim zajedno* u jedinstven ulaz nazvan T. Stoga za T flip-flopove možemo također smatrati da su uvijek *ivično okidani* (u suštini, oni i moraju biti ivično okidani, da bi se izbjegle oscilacije u slučaju da je ulaz T duže vremena na jedinici). Eventualni simbol T flip-flopa je isti kao simbol taktovanog D flip-flopa, pri čemu je ulaz D zamijenjen ulazom T.

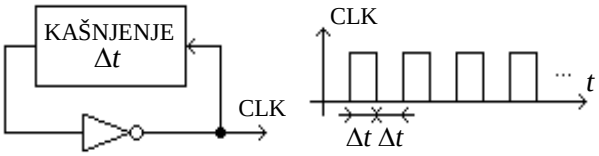
Bitno je napomenuti da čak ni ivično okidani flip-flopovi ne rješavaju sve probleme vezane za sinhronizaciju. Naime, prilikom korištenja ivično okidanih flip-flopova prećutno se pretpostavlja da će ulazne promjenljive imati fiksne (stabilne) vrijednosti u bliskoj okolini trenutka nailaska ivice klok signala. Ukoliko se neki od ulaza promijeni *upravo u trenutku nailaska ivice klok signala*, veoma je diskutabilno koju će vrijednost ulaza sklop "uhvatiti". Čak je moguće i da sklop upadne u neko nedefinirano (tzv. *metastabilno*) stanje iz kojeg mu treba izvjesno vrijeme da se "oporavi". Za to vrijeme, izlaz iz sklopa daje vrijednost koja se ne može tumačiti niti kao nula, niti kao jedinica. Doduše, vjerovatnoća da će se neki ulazni signal promijeniti baš u "nezgodnom" trenutku je prilično mala, ali uvijek postoji. U sklopu na sljedećoj slici ovaj problem je "vještački" izazvan, vezivanjem CLK i D ulaza zajedno, tako da se ulaz uvijek mijenja zajedno sa ivicom klok impulsa. Praktični eksperimenti sa ovim sklopom pokazali su da se on zaista veoma čudno i nepredvidljivo ponaša:



Opisani problem je "bolna tačka" svih digitalnih sklopova koji bi trebali da rade sa velikom pouzdanošću (poznati su slučajevi "feleričnih" kontrolera hard-diskova, koji su s vremena na vrijeme dovodili do upisa pogrešnih informacija na disk, upravo zbog upadanja u metastabilno stanje). Vrijedi napomenuti da je izbjegavanje mogućnosti upadanja u metastabilno stanje *otvoren problem*, za čijim se potpunim rješenjem još uvijek traga. Do sada postoje samo manje ili više zadovoljavajuća rješenja koja *smanjuju vjerovatnoću greške na zanemarljiv iznos*, ali je ne otklanjaju u potpunosti.

Vidjeli smo da taktni impulsi predstavljaju osnovu za uvođenje informacije o vremenu u elementarne automate. Međutim, do sada ništa nismo rekli o tome *odakle dolaze taktni impulsi*. Za njihovo generiranje zaduženi su sklopovi nazvani ***generatori taktnih impulsa*** ili ***oscilatori***. Mehanički analogon ovih sklopova je *satni mehanizam*. Stoga se generator taktnih impulsa često naziva i ***sistemski sat***, s obzirom da snabdjeva sistem informaciom o vremenu. Detaljno objašnjenje rada generatora taktnih impulsa zahtijeva dublje poznavanje fizike i elektronike, pa ćemo se sa njima upoznati samo informativno. U svakom slučaju, to su *nestabilni sklopovi*, principijelno slični invertoru čiji je izlaz vraćen na ulaz, i oni predstavljaju jednu od rijetkih klasa sklopova kod kojih je nestabilnost *korisno upotrijebljena*. Najlakše je realizirati generatore ***pravougaonih taktnih impulsa*** (odnosno generatore

četvrtki) kod kojih je trajanje impulsa jednako trajanju razmaka između dva impulsa. Stoga se takvi generatori taktnih impulsa najčešće susreću. Njih principijelno možemo realizirati kao na sljedećoj slici:



Princip rada ovog sklopa je očigledan. Ono što nije očigledno je *kako se realizira kašnjenje* u bloku u povratnoj sprezi. Ovo kašnjenje se realizira raznim inercijalnim elementima (kao što su npr. *kondenzatori*) čije funkcioniranje izlazi izvan okvira čisto digitalne tehnike, tako da u to nećemo ulaziti. Očigledno, tačnost sistemskog sata ovisi o održavanju preciznog trajanja kašnjenja Δt. Najbolji se rezultati postižu upotrebom blokova za kašnjenje u kojima se koriste ***kristali kvarca***. Oscilatori u kojima se koristi kristal kvarca nazivaju se ***kvarcni oscilatori***. U svim kvalitetnijim digitalnim sekvencijalnim sklopovima za generiranje taktnih impulsa se koriste isključivo kvarcni oscilatori, jer je preciznost taktnih impulsa često od vitalnog značaja za pouzdan rad čitavog sklopa.

(?) Pitanja i zadaci

16.1 Objasnite šta su elementarni automati (flip-flopovi).

16.2 Objasnite ulogu povratnih sprega za potrebe realizacije sekvencijalnih sklopova.

16.3 Pokažite da se serijski spoj dva invertora zatvoren povratnom spregom može nalaziti u dva stabilna stanja.

16.4 Objasnite kako se principijelno može unijeti informacija u bistabilni sklop, kao i zbog čega su za održavanje memorirane informacije u petlji neophodni aktivni elementi poput invertora.

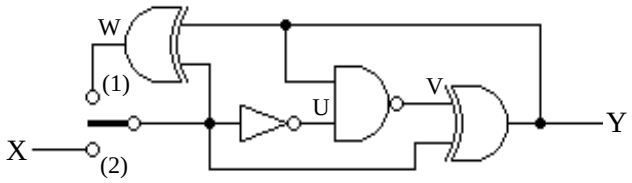
16.5 Objasnite strukturu elementarnog memorijskog elementa (asinhronog D flip-flopa) zasnovanog na multiplekseru 2/1. Sastavite njegovu tablicu prelaza i izvedite jednačinu prelaza koja opisuje njegov rad.

16.6 Vidjeli smo da realizacija asinhronog D flip-flopa zasnovana na multiplekseru 2/1 nije pouzdana ukoliko se multiplekser realizira na najjednostavniji mogući način, zbog toga što je takva realizacija multipleksera 2/1 rizična mreža. Izvedite multiplekser 2/1 kao *mrežu bez rizika* (prema uputama iz Poglavlja 10), i realizirajte (pouzdan) asinhroni D flip-flop na osnovu takve realizacije multipleksera 2/1. Uporedite sa aspekta ekonomičnosti dobijenu realizaciju sa standardnim realizacijama asinhronog D flip-flopa (zasnovanim na RS flip-flopu).

16.7 Objasnite fenomen nestabilnosti, odnosno osciliranja koji može nastati kod sekvencijalnih sklopova. Da li je nestabilnost uvijek štetna pojava?

16.8 Serijski spoj dva invertora zatvoren povratnom spregom zaista se može nalaziti u dva stabilna stanja, pri čemu su u oba stanja izlazi ta dva invertora međusobno komplementarni (negirani). Pretpostavimo, međutim, da se u početnom trenutku vremena, iz nekog razloga, na oba invertora pojavila identična vrijednost (recimo, 0), i da oba invertora imaju potpuno isto kašnjenje. Šta će se tada desiti? Da li je takva mogućnost realistična u stvarnosti?

16.9* Dat je sekvencijalni sklop, kao na sljedećoj slici:



a) Neka se prekidač nalazi u poziciji (1), tako da je ulaz X fizički odvojen od ostatka sklopa. Pokažite da postoji samo jedna stabilna vrijednost koju tada izlaz Y može imati. Koja je to vrijednost? Kakve su tada vrijednosti u tačkama U, V i W?

b) Pretpostavimo da se u početnom trenutku prekidač nalazio u poziciji (1) ali da se početna vrijednost na izlazu Y razlikovala od vrijednosti određene pod a). Šta će se tada desiti? Također pretpostavimo da makar neka od početnih vrijednosti u tačkama U, V i W nisu jednake vrijednostima određenim pod a). Šta će se tada desiti? Šta iz svega možemo zaključiti? U slučaju potrebe, pretpostavite da sva logička kola imaju isto kašnjenje Δt.

c) Neka se sada prekidač nalazi u poziciji (2). Šta se dešava na izlazu ukoliko je $X = 0$, a šta ukoliko je $X = 1$?

d) Neka je, u nekom trenutku, prekidač prebačen iz pozicije (2) nazad u poziciju (1). Šta se dešava nakon toga?

16.10 Nacrtajte strukturu RS flip-flopa zasnovanu na NOR logičkim kolima, objasnite njegov princip rada, prikažite tablicu koja opisuje njegov rad i izvedite jednačinu prelaza stanja koja ga opisuje.

16.11 Pokažite da pod određenim uvjetima RS flip-flop može postati nestabilan.

16.12 Nestabilnost se, u realnosti, kod RS flip-flopa gotovo nikada ne pojavljuje, mada je teoretski moguća. Objasnite zašto. Također, objasnite zbog čega se kombinacija ulaza $R = S = 1$ ne koristi u praksi, čak i ukoliko garantiramo da neće doći do nestabilnosti.

16.13 Nacrtajte strukturu RS flip-flopa zasnovanu na NAND logičkim kolima i pokažite da ona zaista radi kao RS flip-flop.

16.14* Polazeći od strukture običnog RS flip-flopa, razmotrite kako bi mogla izgledati realizacija

a) RS flip-flopa sa dominatnim S ulazom;
b) RS flip-flopa sa dominatnim R ulazom;
c) RS flip-flopa sa nedominatnim ulazima.

U sva tri slučaja, pronađite najekonomičnije realizacije uz pomoć NAND ili NOR logičkih kola.

16.15* Jedna od najčešćih primjena klasičnih RS flip-flopova je za utvrđivanje koji je od nekoliko različitih događaja nastupio prvi (pretpostavlja se da se nastup nekog događaja signalizira dovođenjem logičke jedinice na odgovarajući ulaz nekog digitalnog sklopa). Projektirajte (intuitivnim putem) sklop sa tri ulaza X_1, X_2 i X_3 i tri izlaza Y_1, Y_2 i Y_3, pri čemu je $Y_i = 1$ ako i samo ako se jedinica prva pojavila na ulazu X_i ($i = 1..3$). Naravno, najviše jedan od izlaza može biti na logičkoj jedinici. Generalizirajte projektirani sklop na slučaj proizvoljnog ali unaprijed fiksiranog broja ulaza (recimo, 10).

16.16 Nacrtajte moguće realizacije asinhronog D flip-flopa zasnovane na RS flip-flopu, a zatim nacrtajte najekonomičniju realizaciju asinhronog D flip-flopa zasnovanu na NAND logičkim kolima. Pokažite logičkom analizom da sve nacrtane realizacije zaista rade kao asinhroni D flip-flopovi.

16.17 Objasnite razloge za uvođenje taktnih (klok) impulsa u sekvencijalne sklopove, a zatim nacrtajte strukturu taktovanog RS flip flopa i objasnite njegov rad.

16.18 Kolika je frekvencija taktnih impulsa ukoliko taktni impulsi nailaze u razmacima od 20 ns?

16.19 Objasnite princip rada JK flip-flopa, nacrtajte njegovu strukturu zasnovanu na RS flip-flopu i dodatnoj logici, prikažite tablicu koja opisuje njegov rad i izvedite jednačinu prelaza stanja koja ga opisuje.

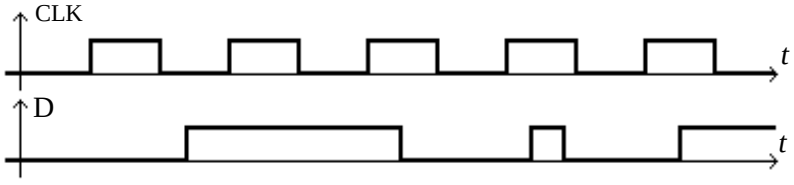
16.20 Objasnite zbog čega su taktni impulsi u JK flip-flopu neophodni.

16.21 Nacrtajte detaljnu strukturu taktovanog JK flip-flopa zasnovanu na NAND logičkim kolima.

16.22 Objasnite razloge za uvođenje sinhronih flip-flopova koji se okidaju ivicom (uzlaznom ili silaznom) taknih impulsa.

16.23 Nacrtajte principijelnu strukturu D flip-flopa okidanog silaznom ivicom izvedenog u master-slejv konfiguraciji, objasnite princip rada ovakvog flip-flopa, a zatim prikažite njegovu detaljnu strukturu zasnovanu na NAND logičkim kolima.

16.24 Na sljedećoj slici je dat vremenski dijagram signala dovedenih na CLK i D ulaze D flip-flopa:



Nacrtajte vremenski dijagram signala na izlazu Q ukoliko je u pitanju

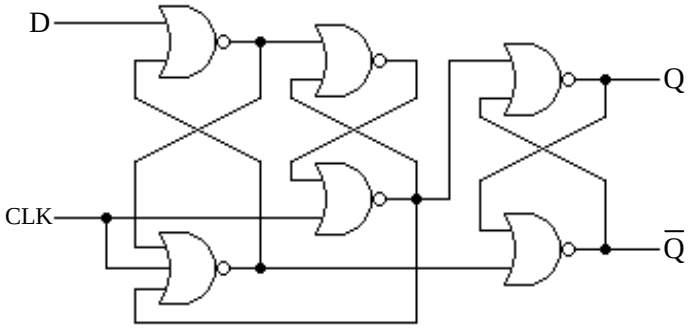
a) D flip-flop okidan opadajućom ivicom taktnih impulsa;
b) D flip-flop okidan opadajućom ivicom taktnih impulsa;
c) asinhroni D flip-flop okidan nivoom (taktni impulsi se dovode na C ulaz).

16.25 Objasnite sličnosti i razlike između asinhronih i sinhronih D flip-flopova.

16.26 Izvedite jednačinu koja opisuje rad D flip-flopa okidanog opadajućom ivicom taktnih impulsa posmatrajući CLK ulaz kao ulaznu promjenljivu ravnopravnu sa D ulazom. Prevedite ovu jednačinu u standardni model konačnog automata prve vrste.

16.27 Pod kakvim uvjetima rad sinhronog D flip-flopa možemo opisivati trivijalnom jednačinom $Q(n+1) = D$?

16.28* Rečeno je da postoje ekonomičnija rješenja ivično okidanih D flip-flopova u odnosu na rješenje zasnovano na master-slejv konfiguraciji. Na primjer, najekonomičnija realizacija D flip-flopa okidanog opadajućom ivicom taktnih impulsa prikazana je na sljedećoj shemi. Analizirajte ovu shemu i utvrdite kako ovaj sklop uopće radi.



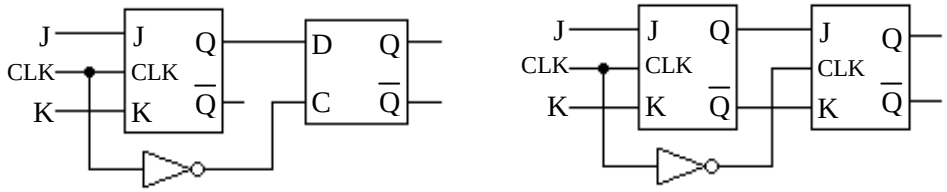
16.29 Objasnite kako se može konstruisati ivično okidani JK flip-flop.

16.30 Kako biste pomoću jednog ivično okidanog JK flip-flopa i jednog invertora napravili ivično okidani D flip-flop?

16.31 Objasnite čemu služi master-slejv JK flip-flop, a zatim nacrtajte njegovu principijelnu strukturu i detaljno objasnite princip njegovog rada.

16.32 Nacrtajte detaljnu strukturu master-slejv JK flip-flopa zasnovanu na NAND logičkim kolima.

16.33 Objasnite zbog čega se master-slejv JK flip-flop ne može realizirati pomoću struktura prikazanih na sljedećoj slici, bez obzira što je master-slejv D flip-flop realiziran na sličan način (odnosno, objasnite zbog čega *master* ni u kom slučaju ne smije biti JK flip-flop).



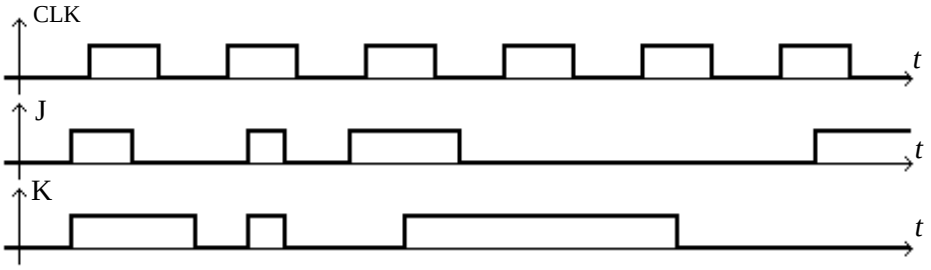
16.34 Objasnite šta se podrazumijeva pod fenomenom "hvatanja jedinica", i zbog čega master-slejv JK flip-flop nije pravi ivično okidani flip-flop (bez obzira što je master-slejv D flip-flop pravi ivično okidani flip-flop, odnosno jedna moguća realizacija ivično okidanog D flip-flopa). Može li master-slejv JK flip-flop "hvatati nule"?

16.35 Master-slejv JK flip-flop se ponaša slično JK flip-flopu okidanom opadajućom ivicom taktnih impulsa, osim što pokazuje fenomen "hvatanja jedinica". Može li se kod master-slejv JK flip-flopa invertirati CLK signal i na taj način dobiti sklop koji se ponaša slično kao JK flip-flop okidan rastućom ivicom taktnih impulsa? Obrazložite odgovor.

16.36 Ako biste u rješenju Zadatka 16.30 umjesto ivično okidanog JK flip-flopa upotrijebili master-slejv JK flip-flop, da li biste dobili pravi ivično okidani D flip-flop? Obrazložite odgovor.

16.37* Kako biste mogli master-slejv JK flip-flop pretvoriti u pravi ivično okidani JK flip-flop? Da li se takva realizacija isplati?

16.38 Na sljedećoj slici je prikazan vremenski dijagram signala dovedenih na CLK, J i K ulaze JK flip-flopa:



Nacrtajte vremenski dijagram signala na izlazu Q ukoliko je u pitanju

a) JK flip-flop okidan opadajućom ivicom taktnih impulsa;
b) JK flip-flop okidan opadajućom ivicom taktnih impulsa;
c) master-slejv JK flip-flop.

16.39 Nacrtajte kako bi se principijelno mogao realizirati ivično okidani i master-slejv RS flip-flop.

16.40 Navedite koji se tipovi flip-flopova koriste u praktičnim realizacijama sekvencijalnih sklopova.

16.41 Objasnite šta je T flip-flop i kakve su mu osobine.

16.42 Objasnite šta je pojava metastabilnosti, zbog čega i kada se može pojavljuje, i na koji način se rješavaju problemi uzrokovani ovom pojavom.

16.43 Objasnite principijelno na čemu se zasniva rad generatora taktnih impulsa.

16.44 Objasnite zbog čega se u kvalitetnimsavremenim generatorima taktnih impulsa koriste kristali kvarca.

17. Formalno projektiranje sinhronih sekvencijalnih sklopova

Na osnovu razmatranja izloženih u prethodnom poglavlju vidljivo je da analiza sekvencijalnih sklopova u općem slučaju može biti veoma složena, i često nemoguća bez uzimanja u obzir kašnjenja koja unose pojedini logički elementi. Analiza složenih sekvencijalnih sklopova proizvoljne strukture može biti praktično neizvodiva za ručni rad, i danas se uglavnom izvodi simulacijom uz pomoć računara, korištenjem specijalnih softverskih paketa. Stoga bi se moglo pomisliti da isto vrijedi i za sintezu, odnosno projektiranje sekvencijalnih sklopova. Međutim, pokazuje se da projektiranje sekvencijalnih sklopova koji obavljaju neku unaprijed zadanu funkcionalnost uopće nije mnogo složenije nego projektiranje kombinacionih sklopova. Stvar je u tome što za ostvarivanje traženih funkcionalnosti *ne moramo razmatrati sekvencijalne sklopove proizvoljne strukture*, već je dovoljno ograničiti se na *određenu podkategoriju sekvencijalnih sklopova*, koji su sa jedne strane *dovoljno općeniti* da se pomoću njih može realizirati svaka funkcionalnost koja se može ostvariti ma kakvim fizički izvodljivim sekvencijalnim sklopom, a koji su sa druge strane *dovoljno jednostavne strukture* da su njihovo projektiranje i analiza dovoljno jednostavni. Jednu takvu kategoriju predstavljaju tzv. *sinhroni sekvencijalni sklopovi*, koji su opisani sljedećom definicijom:

Definicija 17.1:

Za sekvencijalni sklop kažemo da je ***sinhroni sekvencijalni sklop***, ukoliko su istovremeno ispunjeni sljedeći uvjeti:

- U sklopu se koriste samo taktovani flip-flopovi i mreža pratećih logičkih kola koja je takva da jedini eventualni kružni tokovi signala (petlje) prolaze kroz neki od flip-flopova (drugim riječima, mreža koju dobijamo kada odstranimo sve flip-flopove predstavlja kombinacionu mrežu, bez ikakvih povratnih sprega);
- Svi flip-flopovi korišteni u sklopu koriste isti način okidanja (npr. svi su okidani silaznom ivicom taktnih impulsa);
- Na sve taktne ulaze svih flip-flopova istovremeno dolaze isti taktni impulsi, tj. impulsi koji dolaze iz istog generatora taktnih impulsa.

Ukoliko makar jedan od navedenih uvjeta nije ispunjen, sekvencijalni sklop se naziva ***asinhroni sekvencijalni sklop***. Na primjer, asinhroni sekvencijalni sklopovi mogu da uopće ne sadrže flip-flopove (mada moraju sadržavati povratne sprege, inače ne bi bili sekvencijalni), ili mogu da sadrže netaktovane flip-flopove, tako da uopće ne koriste taktne impulse, ili ih koriste samo djelimično. Dalje, asinhroni sekvencijalni sklopovi mogu da koriste taktne impulse, ali koji ne dolaze istovremeno na sve flip-flopove, ili koji dolaze iz različitih generatora taktnih impulsa (na primjer, postoje asinhroni sekvencijalni sklopovi kod kojih se izlaz iz jednog flip-flopa koristi kao taktni signal za drugi flip-flop). Također, asinhroni sekvencijalni sklopovi mogu sadržavati flip-flopove koji se okidaju na različite načine (ovo je veoma rijedak slučaj). Konačno, asinhroni sekvencijalni sklopovi mogu da imaju zatvorene tokove signala (petlje) koje ne prolaze ni kroz jedan flip-flop, čime se također narušavaju uvjeti sinhronosti sekvencijalnog sklopa. Interesantno je primijetiti da svaki sekvencijalni sklop sadrži podsklopove koji su asinhronog tipa. Tako, svaki sinhroni sekvencijalni sklop sadrži taktovane flip-flopove, koji kao svoj sastavni element posjeduju asinhroni D ili RS flip-flop, koji su asinhroni sklopovi (samim tim što ne koriste taktne impulse).

Sinhrone sekvencijalne sklopove je mnogo lakše projektirati i analizirati od asinhronih. Pored toga, uskoro ćemo vidjeti da se pomoću sinhronih sekvencijalnih sklopova može realizirati svaka funkcionalnost koja bi se mogla ostvariti ma kakvim drugim sekvencijalnim sklopom. To ipak ne znači da asinhrone sekvencijalne sklopove treba potpuno odbaciti. Naime, asinhroni sekvencijalni sklopovi mogu biti mnogo brži od sinhronih, pogotovo oni koji uopće ne koriste taktne impulse, s obzirom da su sinhroni sekvencijalni sklopovi praktično "mrtvi" u periodu između dva taktna impulsa (odnosno između dvije ivice taktnih impulsa za slučaj ivičnog okidanja). Pored toga, asinhroni sekvencijalni sklopovi mogu biti jednostavnije građe od sinhronih. Ipak, dugo vremena uopće nisu postojale nikakve

sistematične metode za projektiranje asinhronih sklopova. Takve metode su se pojavile tek u posljednje vrijeme, i mnogo su složenije od metoda za projektiranje sinhronih sekvencijalnih sklopova. Mada ovakve metode u posljednje vrijeme privlače mnogo pažnje, prvenstveno zbog potrebe razvoja komponenti sa velikom brzinom rada, na ovom mjestu ćemo se ograničiti samo na projektiranje sinhronih sekvencijalnih sklopova. Bitno je istaći da ovakvim ograničenjem ne gubimo ništa na općenitosti. Jedini mogući gubici mogu biti sa aspekta perfomansi (npr. brzine rada).

Osnovno sredstvo za projektiranje sinhronih sekvencijalnih sklopova predstavljaju tzv. ***tablice pobude*** elementarnih automata, odnosno flip-flopova. Tablice pobude su, na izvjestan način *inverzne* tablicama prelaza. Dok tablice prelaza iskazuju zavisnost novog stanja $Q(n+1)$ od tekućeg stanja $Q(n)$ i ulaza u flip-flop, tablice pobude iskazuju šta je potrebno dovesti na ulaze flip-flopa da bi se ostvario prelaz iz zadanog tekućeg stanja $Q(n)$ u zadano buduće stanje $Q(n+1)$. Na primjer, za RS flip-flop tablica pobude izgleda ovako:

$Q(n)$	$Q(n+1)$	R	S
0	0	x	0
0	1	0	1
1	0	1	0
1	1	0	x

Ovu tablicu treba tumačiti na sljedeći način. Ukoliko pretpostavimo da je tekuće stanje RS flip flopa $Q(n) = 0$, i želimo da buduće stanje ostane isto, tj. da bude $Q(n+1) = 0$, neophodno je da bude $S = 0$, dok R može imati bilo kakvu vrijednost (što je označeno sa "x" u tablici). Zaista, za $R = S = 0$ sklop ne mijenja zatečeno stanje, dok se za $R = 1$ i $S = 0$ resetuje, što opet ne mijenja zatečeno stanje, s obzirom da je sklop bio resetovan. Pretpostavimo sada da je tekuće stanje također $Q(n) = 0$, ali da želimo da buduće stanje bude $Q(n+1) = 1$, odnosno da želimo da setujemo sklop. Nije teško uvidjeti da je jedini način da to postignemo dovođenje ulazne kombinacije $R = 0$ i $S = 1$. Na sličan način se mogu rastumačiti i preostala dva reda tablice pobude.

Tablice pobude se često prikazuju i u sljedećem obliku:

PRELAZ	R	S
$0\rightarrow0$	x	0
$0\rightarrow1$	0	1
$1\rightarrow0$	1	0
$1\rightarrow1$	0	x

U literaturi se prelazi $0\rightarrow0$, $0\rightarrow1$, $1\rightarrow0$ i $1\rightarrow1$ ponekad respektivno nazivaju 0-prelaz, α-prelaz, β-prelaz i 1-prelaz.

Tablice pobude se mogu sastaviti i za druge tipove flip-flopova. Sljedeća tablica prikazuje u pakovanom obliku tablice pobude za sva četiri standardna tipa flip-flopova: RS, D, JK i T (napomenimo da pored ova četiri osnovna postoje i drugi tipovi flip-flopova). Čitateljima i čitateljkama se ostavlja za vježbu da provjere istinitost ovih tablica:

PRELAZ	R	S	D	J	K	T
$0\rightarrow0$	x	0	0	0	x	0
$0\rightarrow1$	0	1	1	1	x	1
$1\rightarrow0$	1	0	0	x	1	1
$1\rightarrow1$	0	x	1	x	0	0

Na osnovu ove tablice lako možemo analitički izraziti neophodne vrijednosti ulaza za ostvarivanje željenih prelaza iz trenutnog stanja $Q(n)$ u buduće stanje $Q(n+1)$. Tako, za slučaj D odnosno T flip-flopa

imamo relacije $D = Q(n+1)$ odnosno $T = Q(n) \oplus Q(n+1)$, dok za slučaj RS odnosno JK flip-flopa izražavanje R i S odnosno J i K preko $Q(n)$ i $Q(n+1)$ nije jednoznačno, s obzirom na prisustvo neodređenih vrijednosti označenih sa "x". U svakom slučaju, i za ove flip-flopove zavisnosti ovog tipa postoje, samo što nisu jedinstvene. Bitno je primijetiti da iz same definicije elementarnog automata slijedi da *za svaki tip elementarnog automata postoji tablica pobude*.

Na osnovu tablica pobude flip-flopova veoma je lako dokazati da se svaka funkcionalnost koja se može realizirati sekvencijalnim sklopovima može realizirati kao sinhroni sekvencijalni sklop. Pri tome je moguće koristiti bilo koji tip elementarnih automata (flip-flopova). Zaista, pretpostavimo da nam je dat opis sekvencijalnog sklopa kao konačni automat druge vrste u obliku

$$Q_k(n+1) = \Phi_k[x_1(n), x_2(n), \ldots x_N(n), Q_1(n), Q_2(n), \ldots Q_P(n)], \qquad k = 1 \,..\, P$$
$$y_j(n) = \Psi_j[x_1(n), x_2(n), \ldots x_N(n), Q_1(n), Q_2(n), \ldots Q_P(n)], \qquad j = 1 \,..\, M$$

Pretpostavimo dalje da imamo na raspolaganju P flip-flopova proizvoljnih tipova, čije ćemo izlaze označiti sa Q_k, $k = 1..P$, a čije čemo ulaze označiti sa z_{kl}, $k = 1..P$, $l = 1..\lambda(k)$ gdje je $\lambda(k)$ broj ulaza k-tog flip-flopa (tipično 1 ili 2). Na osnovu tablica pobuda slijedi da za svaki od navedenih flip-flopova postoje funkcije ϑ_{kl} koje izražavaju zavisnost oblika $z_{kl}(n) = \vartheta_{kl}(Q_k(n), Q_k(n+1))$, $k = 1..P$, $l = 1..\lambda(k)$ pri čemu funkcije ϑ_{kl} nisu nužno jedinstveno određene. Ukoliko su svi flip-flopovi taktovani istim taktnim impulsima i sa istim načinom okidanja, tada diskretni trenuci n i $n+1$ svim flip-flopovima predstavljaju *iste trenutke stvarnog vremena* tako da možemo napisati sljedeći sistem jednačina:

$$z_{kl}(n) = \vartheta_{kl}\{Q_k(n), \Phi_k[x_1(n), x_2(n), \ldots x_N(n), Q_1(n), Q_2(n), \ldots Q_P(n)]\}, \qquad k = 1 \,..\, P, \quad l = 1..\lambda(k)$$
$$y_j(n) = \Psi_j[x_1(n), x_2(n), \ldots x_N(n), Q_1(n), Q_2(n), \ldots Q_P(n)], \qquad j = 1 \,..\, M$$

Kako u svim jednačinama sve promjenljive zavise od istog formalnog argumenta n, njega možemo ukloniti, tako da možemo pisati:

$$z_{kl} = \vartheta_{kl}[Q_k, \Phi_k(x_1, x_2, \ldots x_N, Q_1, Q_2, \ldots Q_P)], \qquad k = 1 \,..\, P, \quad l = 1..\lambda(k)$$
$$y_j = \Psi_j(x_1, x_2, \ldots x_N, Q_1, Q_2, \ldots Q_P), \qquad j = 1 \,..\, M$$

Ovim smo ulaze flip-flopova z_{kl} i izlaze y_j iz sklopa izrazili sistemom običnih logičkih funkcija koje zavise od ulaza x_i u sklop i izlaza Q_k iz flip-flopova. Ovaj sistem funkcija može se na već dobro poznati način realizirati kao kombinaciona mreža. Povezivanjem takve kombinacione mreže sa ulazima i izlazima odgovarajućih flip-flopova dobijamo upravo sinhroni sekvencijalni sklop koji obavlja funkcionalnost opisanu polaznim sistemom jednačina, što je i trebalo pokazati.

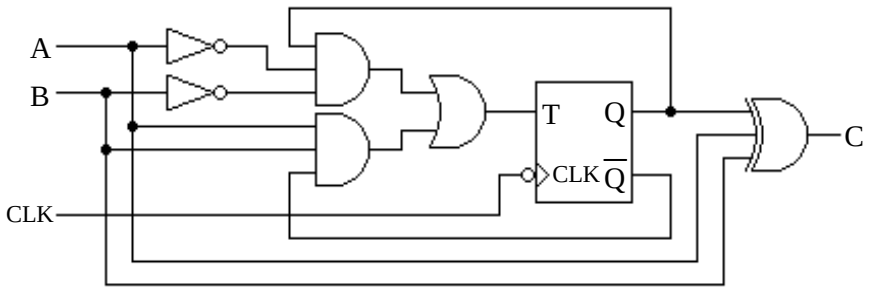
Pokažimo opisanu tehniku na jednom jednostavnom primjeru. Neka je potrebno realizirati *serijski sabirač*. U Poglavlju 15 smo vidjeli da se njegov matematski model može iskazati jednačinama

$$Q(n+1) = AB \vee (A \vee B)\, Q(n)$$
$$C = A \oplus B \oplus Q(n)$$

Ovdje smo, radi konzistencije, promjenljivu stanja koja pamti informaciju o prenosu označili sa Q umjesto sa P. Odlučimo li se da za pamćenje ovog bita stanja upotrijebimo T flip-flop, možemo iskoristiti relaciju $T = Q(n) \oplus Q(n+1)$ koja slijedi iz njegove tablice pobude. Odavde imamo:

$$\begin{aligned} T = Q(n) \oplus Q(n+1) &= Q(n) \oplus [AB \vee (A \vee B)\, Q(n)] = \\ &= Q(n)\, \overline{AB \vee (A \vee B)\, Q(n)} \vee \overline{Q(n)}\, [AB \vee (A \vee B)\, Q(n)] = \\ &= Q(n)\, \overline{AB} \left[\overline{A \vee B} \vee \overline{Q(n)}\right] \vee \overline{Q(n)}\, AB = Q(n)\, \overline{AB}\, \overline{A \vee B} \vee \overline{Q(n)}\, AB = \\ &= Q(n)\, (\overline{A} \vee \overline{B})\, \overline{A}\, \overline{B} \vee \overline{Q(n)}\, AB = Q(n)\, \overline{A}\, \overline{B} \vee \overline{Q(n)}\, AB \end{aligned}$$

Dobijena jednačina, zajedno sa jednačinom C = A ⊕ B ⊕ Q(*n*), posve je dovoljna da možemo nacrtati traženu strukturu serijskog sabirača, prema sljedećoj shemi:



Ovdje smo upotrijebili T flip-flop okidan opadajućom ivicom, što u suštini nije bitno. Bitno je jedino da klok impulsi moraju postojati, i da svi flip-flopovi (ukoliko ih ima više) budu okidani na isti način, i iz istog izvora klok impulsa.

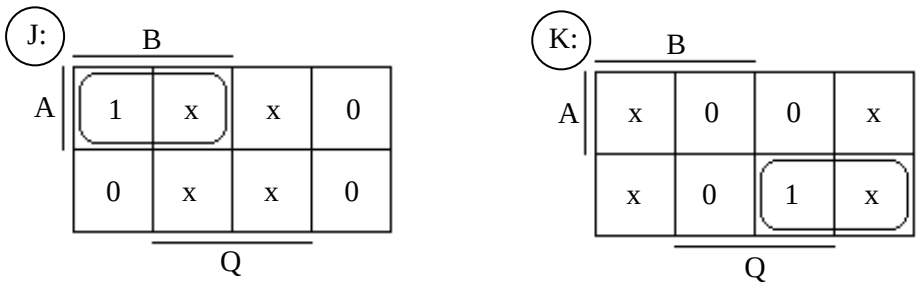
U praksi se projektiranje sinhronih sekvencijalnih sklopova obično ne vrši polazeći od matematskog modela izraženog jednačinama, već od *tablice prelaza i izlaza* koja opisuje rad traženog sekvencijalnog sklopa (što je u suštini ekvivalentno, ali mnogo praktičnije za rad). Pokažimo ovu tehniku ponovo na primjeru projektiranja *serijskog sabirača*. Ranije smo vidjeli da se serijski sabirač može opisati sljedećom tablicom prelaza i izlaza (i ovdje smo promjenljivu stanja koja pamti informaciju o prenosu označili sa Q umjesto sa P):

A	B	Q(*n*)	C	Q(*n*+1)
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

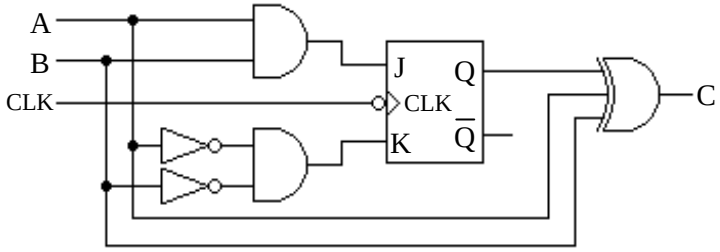
Za pamćenje samo jednog bita stanja, dovoljan nam je jedan flip-flop, koji možemo izabrati po volji. Ovaj put se odlučimo npr. za JK flip-flop. U skladu sa opisanim postupkom, potrebno je ulaze u flip-flop J i K i izlaz C izraziti u funkciji od ulaza A i B i izlaza iz flip-flopa Q. Zavisnost C od Q može se očitati direktno iz prethodne tablice. Da bismo odredili zavisnost J i K od A, B i Q nije potrebno prvo određivati zavisnost J i K od Q(*n*) i Q(*n*+1) na osnovu tablica pobude pa onda uvrštavati dobijeni izraz u zavisnost Q(*n*+1) od A, B i Q koja bi se mogla dobiti iz prethodne tablice (kao što smo maločas radili u analitičkom izvođenju). Umjesto toga, bolje je na prethodnu tablicu *dopisati nove kolone* koje odgovaraju ulazima u JK flip-flop, i popuniti te kolone vrijednostima koje se dobiju iz tablica pobude za JK flip-flop posmatrajući odgovarajuće vrijednosti za Q(*n*) i Q(*n*+1) u svakom redu. Tako dobijena tablica naziva se ***proširena tablica prelaza, izlaza i pobuda***. U razmotrenom primjeru, ona će izgledati ovako (za njeno razumijevanje obavezno analizirajte tablicu pobude za JK flip-flop):

A	B	Q(*n*)	C	Q(*n*+1)	J	K
0	0	0	0	0	0	x
0	0	1	1	0	x	1
0	1	0	1	0	0	x
0	1	1	0	1	x	0
1	0	0	1	0	0	x
1	0	1	0	1	x	0
1	1	0	0	1	1	x
1	1	1	1	1	x	0

U praktičnom radu naravno nećemo crtati novu tabelu, nego ćemo samo na običnu tablicu prelaza i izlaza dopisati nove kolone. Sada, iz ovako dobijene tablice direktno očitavamo zavisnosti J, K i C od A, B i Q. To možemo učiniti npr. pomoću Veitchovih dijagrama (Veitchov dijagram za promjenljivu C nećemo crtati, s obzirom da smo ranije već našli da je $C = A \oplus B \oplus Q$):



Iz ovih dijagrama neposredno očitavamo $J = AB$ i $K = \overline{A}\,\overline{B}$ na osnovu čega neposredno slijedi shema traženog serijskog sumatora. Može se primijetiti da smo dobili znatno jednostavniju realizaciju nego u slučaju kada smo za memoriranje promjenljive stanja koristili T flip-flop:



Na osnovu izloženog postupka za projektiranje sinhronih sekvencijalnih sklopova na primjeru serijskog sabirača, može se uočiti opća struktura postupka za formalno projektiranje sinhronih sekvencijalnih sklopova:

1. Na osnovu verbalnog opisa rada sekvencijalnog sklopa, formirati tablicu prelaza i izlaza koja opisuje rad traženog sekvencijalnog sklopa;
2. Izabrati onoliko flip-flopova koliko se bita stanja javlja u tablici prelaza i izlaza (ovaj izbor nije jednoznačan, s obzirom da se tipovi flip-flopova mogu izabrati proizvoljno);
3. Proširiti tablicu prelaza i izlaza sa onoliko novih kolona koliko ukupno ima ulaza u sve izabrane flip-flopove;
4. Popuniti novoformirane kolone na osnovu tekućih i budućih vrijednosti bita stanja u svakom redu tablice, koristeći tablice pobude za odgovarajuće flip-flopove;
5. Na osnovu dobijene proširene tablice, realizirati sve izlaze iz sklopa i sve ulaze u flip-flopove kao funkcije od ulaza sklopa i izlaza iz flip-flopova (tj. trenutnih vrijednosti bita stanja).

Svi opisani koraci potpuno su formalizirani, osim koraka 1. koji je stoga i najteži. U nekim situacijama posve je lako formirati tablicu prelaza i izlaza na osnovu verbalnog opisa sklopa, kao što je to slučaj sa serijskim sabiračem. Međutim, u većini slučajeva formiranje tablice nije jednostavno, jer nije očigledno šta izabrati za bite stanja. Stoga ćemo u nastavku posvetiti više pažnje upravo problematici formiranja ove tablice. Prije toga, uočimo da se opisani postupak veoma jednostavno može iskoristiti za pretvaranje flip-flopova jedne vrste u drugu vrstu. Naime, dovoljno je željeni tip flip-flopa posmatrati kao željeni sekvencijalni sklop (čiju tablicu prelaza znamo), a polazni flip-flop kao flip-flop koji smo izabrali u koraku 2. formalnog postupka projektiranja. Ovaj postupak ćemo ilustrirati na dva primjera.

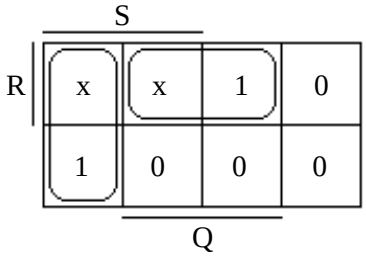
Primjer 17.1:

- Na raspolaganju nam je T flip-flop. Napraviti od njega RS flip-flop.

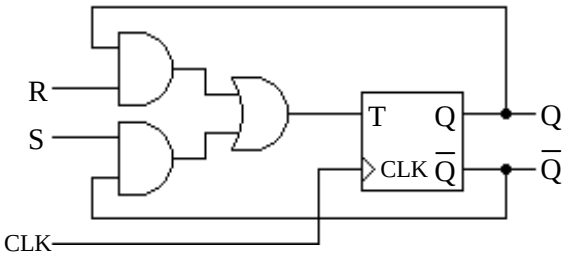
Krenućemo od tablice prelaza RS flip-flopa, na koju ćemo dodati novu kolonu T, s obzirom da nam je flip-flop koji služi za realizaciju upravo T flip-flop. Sadržaj ove kolone popunićemo na osnovu sadržaja kolona za Q(n) i Q(n+1). Pri tome možemo koristiti tablicu pobude za T flip-flop, ili poznatu činjenicu da je za T flip-flop $T = Q(n) \oplus Q(n+1)$. U svakom slučaju, dobijamo sljedeću tablicu:

R	S	Q(n)	Q(n+1)	T
0	0	0	0	0
0	0	1	1	0
0	1	0	1	1
0	1	1	1	0
1	0	0	0	0
1	0	1	0	1
1	1	0	x	x
1	1	1	x	x

Sada još jedino preostaje da realiziramo T kao funkciju od R, S i Q. Naime, kako su vrijednosti za T izabrane tako da ostvare upravo one prelaze stanja kakve ostvaruje i RS flip-flop, dobijeni sklop će tačno oponašati funkcioniranje RS flip-flopa. Funkciju koja opisuje T iz R, S i Q najlakše je odrediti pomoću Veitchovog dijagrama:



Iz ovog dijagrama očitavamo $T = RQ \vee S\overline{Q}$. Naravno, jasno je da smo do iste jednačine mogli doći i čisto analitičkim putem, ako krenemo od činjenice da je za T flip-flop $T = Q(n) \oplus Q(n+1)$, a da za RS flip-flop vrijedi $Q(n+1) = S \vee \overline{R}Q(n)$. U svakom slučaju, na osnovu dobijene jednačine neposredno slijedi shema RS flip-flopa realiziranog preko T flip-flopa:



Ovdje je bitno istaći jednu činjenicu. Kako je opisani postupak namijenjen za projektiranje *sinhronih* sekvencijalnih sklopova, krenuli smo od *taktovanog* T flip-flopa i kao krajnji rezultat dobili *taktovani* RS flip-flop. Opisani postupak ne može se iskoristiti za pretvorbu netaktovanih u taktovane flip-flopove i obrnuto. Međutim, interesantno je da isti postupak radi sasvim lijepo pođe li se od *netaktovanih* flip-flopova. Pri tome se, razumljivo, kao rezultat pretvorbe dobija ponovo netaktovani flip-flop. Ilustrirajmo ovo na jednom primjeru.

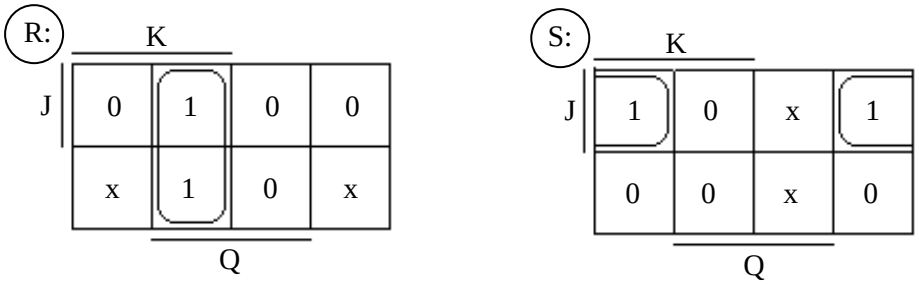
Primjer 17.2:

- Pretvoriti asinhroni RS flip-flop u (netaktovani) JK flip-flop.

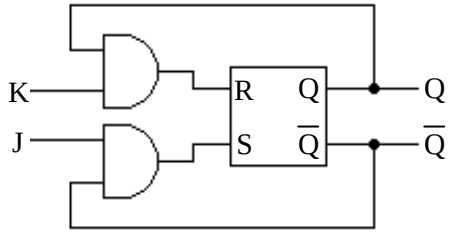
Na isti način kao u prethodnom primjeru, poći ćemo od tablice koja opisuje rad JK flip-flopa, a zatim dopisati nove kolone za R i S ulaze koje ćemo popuniti na osnovu kolona za Q(n) i Q(n+1) i tablica pobude za RS flip-flop. Na taj način dobijamo sljedeću proširenu tablicu prelaza i pobuda:

J	K	Q(n)	Q(n+1)	R	S
0	0	0	0	x	0
0	0	1	1	0	x
0	1	0	0	x	0
0	1	1	0	1	0
1	0	0	1	0	1
1	0	1	1	0	x
1	1	0	1	0	1
1	1	1	0	1	0

Vrijednosti ulaza R i S ćemo realizirati na osnovu ulaza J, K i Q uz pomoć Veitchovih dijagrama:

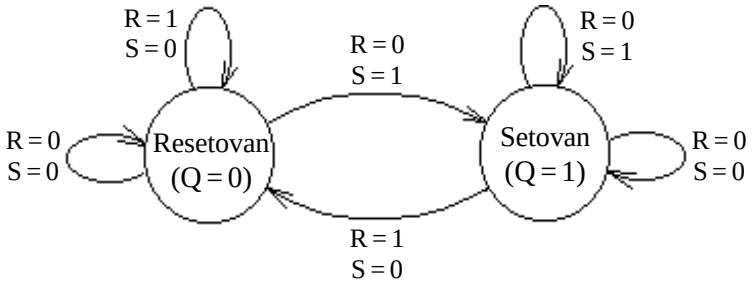


Iz ovih dijagrama očitavamo $R = KQ$ i $S = J\overline{Q}$, na osnovu čega dobijamo sljedeću sliku:



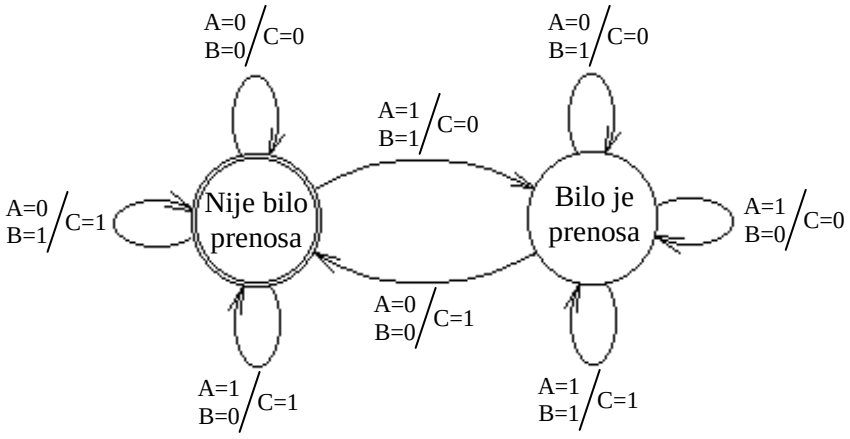
Vidimo da smo dobili već poznatu sliku posve formalnim postupkom.

Vratimo se sada na problem formiranja tablice prelaza i izlaza koja opisuje rad sekvencijalnog sklopa na osnovu njegovog verbalnog opisa. Za tu svrhu se kao najpogodniji pokazao ***metod grafova***. Primjena ovog metoda zasniva se na činjenici da se rad sekvencijalnih sklopova jako lijepo i pregledno može opisati pomoću *grafova*. Pri tome, čvorovima grafa odgovaraju *stanja sklopa,* dok granama grafa odgovaraju *prelazi iz stanja u stanje* (ovdje ćemo pretpostaviti da je pojam grafa čitatelju odnosno čitateljici intuitivno poznat, bez ulaženja u formalnu definiciju samog pojma grafa). Kraj grana grafa upisuju se *vrijednosti ulaza* koje dovode do odgovarajućih prelaza iz stanja u stanje. Što se tiče izlaza iz sklopa, njih je nekada prirodnije pridružiti *čvorovima grafa*, a nekada *granama grafa* (prvi slučaj odgovara automatima Mooreovog tipa, a drugi slučaj automatima Mealyjevog tipa). Na primjer, rad RS flip-flopa možemo predstaviti grafom na sljedećoj slici:



U ovom primjeru, izlaz iz sklopa Q direktno je vezan za stanje sklopa (resetovan ili setovan). S druge strane, graf sa sljedeće slike opisuje rad serijskog sabirača. U ovom grafu, izlaz C je pridružen granama grafa, s obzirom da izlaz zavisi kako od zatečenog stanja, tako i od ulaza koji izazivaju prelaz iz

tekućeg u novo stanje (Mealyjev automat). Stanje "Nije bilo prenosa" označeno je sa dvostrukim rubom, što po konvenciji označava *početno stanje* (tj. stanje za koje podrazumijevamo da vrijedi u trenutku kada sklop započne sa radom).



Važnost opisivanja rada sekvencijalnih sklopova uz pomoć grafova leži u činjenici da je na osnovu verbalnog opisa rada sekvencijalnog sklopa često mnogo lakše formirati graf koji opisuje njegov rad nego tablicu prelaza i izlaza ili analitički model. Nakon što se formira graf, postoje veoma jednostavni formalni postupci za njegovo prevođenje u tablicu prelaza i izlaza (iz koje, u slučaju potrebe, možemo očitati i analitički model). Na taj način, do tražene tablice prelaza i izlaza lako dolazimo indirektnim putem. Formiranje grafa na osnovu verbalnog opisa demonstriraćemo na jednom primjeru koji je veoma ilustrativan, mada nema neposredne veze sa računarskom tehnikom.

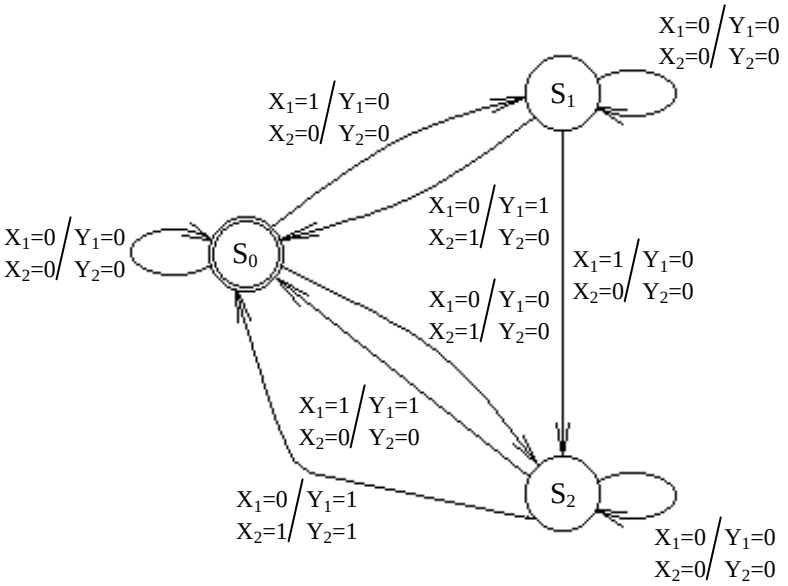
Primjer 17.3:

- U automatu za prodaju velikih boca Coca-Cole, jedna boca košta 3 KM. Automat prima kovanice od 1 KM i 2 KM. Radom automata upravlja sekvencijalni sklop sa dva ulaza X_1 i X_2. Na ulazu X_1 se pojavljuje logička jedinica (u trajanju jednog taktnog impulsa) u trenutku kad se ubaci kovanica od 1 KM, a na ulazu X_2 pojavljuje se logička jedinica (u istom trajanju) kad se ubaci kovanica od 2 KM. U svim ostalim trenucima ulazi X_1 i X_2 su na logičkoj nuli. Sekvencijalni sklop posjeduje dva izlaza Y_1 i Y_2. Logička jedinica na izlazu Y_1 pokreće motor koji izbacuje bocu Coca-Cole, dok logička jedinica na izlazu Y_2 daje nalog mašini da izbaci kusur od 1 KM. Opisati rad sekvencijalnog sklopa koji upravlja ovom mašinom pomoću grafa.

Očigledno se radi o sekvencijalnom sklopu, s obzirom da mašina mora da vodi evidenciju o tome koliko je prethodno novca ubačeno da bi znala kako da reagira na ubacivanje sljedeće kovanice. Pri tome je važno primijetiti da nije potrebno voditi evidenciju o *ukupnoj* količini novca u mašini, nego samo o količini novca koja je ubačena *nakon posljednje izbačene boce Coca-Cole*. Naime, nakon svake izbačene boce, mašina se, sa aspekta korisnika, ponaša kao da je tek započela sa radom. Ova činjenica je od presudne važnosti, jer na osnovu nje možemo zaključiti da se sa aspekta ponašanja sklop može nalaziti samo u jednom od tri različita stanja, koja možemo simbolički označiti sa S_0, S_1 i S_2, sa značenjima prikazanim u sljedećoj tabeli:

STANJE	ZNAČENJE
S_0	Nakon izbacivanja posljednje boce nije ubačena niti jedna kovanica
S_1	Nakon izbacivanja posljednje boce ubačeno je ukupno 1 KM
S_2	Nakon izbacivanja posljednje boce ubačeno je ukupno 2 KM

Do izbacivanja Coca-Cole dolazi ukoliko se u stanju S_1 ubaci kovanica od 2 KM, ili ako se u stanju S_2 ubaci bilo kovanica od 1 KM bilo kovanica od 2 KM (pri čemu se u posljednjem slučaju izbacuje i kusur). Pri tome se mašina vraća u početno stanje (S_0). Na osnovu provedenog rezonovanja, sasvim je lako nacrtati graf koji opisuje rad sklopa:



Primijetimo da se nigdje ne pojavljuje situacija $X_1 = 1$ i $X_2 = 1$, s obzirom da istovremeno ubacivanje kovanica od 1 KM i 2 KM nije moguće. Alternativno se, umjesto grafom, rad sklopa može opisati sljedećom tablicom čija je struktura jasna sama po sebi, a koju je lako sastaviti neposrednim iščitavanjem grafa:

X_1	X_2	STARO STANJE	NOVO STANJE	Y_1	Y_2
0	0	S_0	S_0	0	0
0	0	S_1	S_1	0	0
0	0	S_2	S_2	0	0
0	1	S_0	S_2	0	0
0	1	S_1	S_0	1	0
0	1	S_2	S_0	1	1
1	0	S_0	S_1	0	0
1	0	S_1	S_2	0	0
1	0	S_2	S_0	1	0

Ova tablica po svojoj formi jako podsjeća na tablice prelaza i izlaza, uz izuzetak što se u njoj nalaze *opisne oznake stanja* (S_0, S_1 itd.) umjesto *bita stanja*. Stoga se ovakva tablica naziva ***opisna*** (***deskriptivna***, ***nekodirana***) ***tablica prelaza i izlaza***, za razliku od "prave" tablice prelaza i izlaza u kojoj se javljaju biti stanja, i koja se često naziva i ***kodirana tablica prelaza i izlaza***. Međutim, od ovakve opisne tablice prelaza i izlaza do kodirane tablice prelaza i izlaza koja nam je neophodna za formalno projektiranje sklopa samo je jedan korak. Naime, opisno prikazana stanja je neophodno *kodirati binarnim brojevima*, odnosno uvesti *bite stanja*. Za tu svrhu potrebno je izabrati n bita stanja pri čemu je $2^n \geq N$ gdje je N broj stanja sklopa (tj. broj čvorova grafa). Nakon toga se svakom od N stanja dodijeli na proizvoljan način binarni broj (tzv. ***kôd stanja***) od n bita, ali tako da svakom stanju odgovara različit binarni broj. Biti tog broja biće upravo traženi biti stanja. U našem primjeru je $N = 3$, pa je dovoljno uzeti $n = 2$ bita stanja, jer je $2^2 = 4 \geq 3$. Tako ćemo stanja S_0, S_1 i S_2 kodirati sa tri različita dvobitna binarna broja (postoje 24 različita načina da se ovo uradi). Usvojimo npr. sljedeće kodiranje:

STANJE	BITI STANJA	
	Q_1	Q_2
S_0	0	0
S_1	0	1
S_2	1	1

Kada smo usvojili kodiranje stanja, posve je lako sastaviti kodiranu tablicu prelaza i izlaza. Za tu svrhu dovoljno je u deskriptivnoj tablici prelaza i izlaza svaku opisnu oznaku stanja zamijeniti odgovarajućim kodom stanja, tj. odgovarajućim bitima stanja. Međutim, kako kodirana tablica prelaza i izlaza mora imati *sve kombinacije* ulaza i bita stanja, može se desiti da neki redovi dobijene tablice ostanu prazni, bilo zbog toga što su neke kombinacije ulaza nemoguće (npr. $X_1 = X_2 = 1$ u razmatranom primjeru), bilo zbog toga što neke kombinacije bita stanja ne postoje (npr. kombinaciju $Q_1 = 1$ i $Q_2 = 0$ nismo iskoristili za kodiranje niti jednog stanja). Ovakve redove u tablici možemo popuniti oznakama "x" s obzirom da se te kombinacije ulaza i stanja svakako neće nikada pojaviti, a na taj način poboljšavamo optimizaciju pri korištenju postupaka minimizacije. Kada primijenimo sve što je upravo rečeno, dobijamo sljedeću kodiranu tablicu prelaza i izlaza za traženi sklop:

X_1	X_2	$Q_1(n)$	$Q_2(n)$	$Q_1(n+1)$	$Q_2(n+1)$	Y_1	Y_2
0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0
0	0	1	0	x	x	x	x
0	0	1	1	1	1	0	0
0	1	0	0	1	1	0	0
0	1	0	1	0	0	1	0
0	1	1	0	x	x	x	x
0	1	1	1	0	0	1	1
1	0	0	0	0	1	0	0
1	0	0	1	1	1	0	0
1	0	1	0	x	x	x	x
1	0	1	1	0	0	1	0
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x

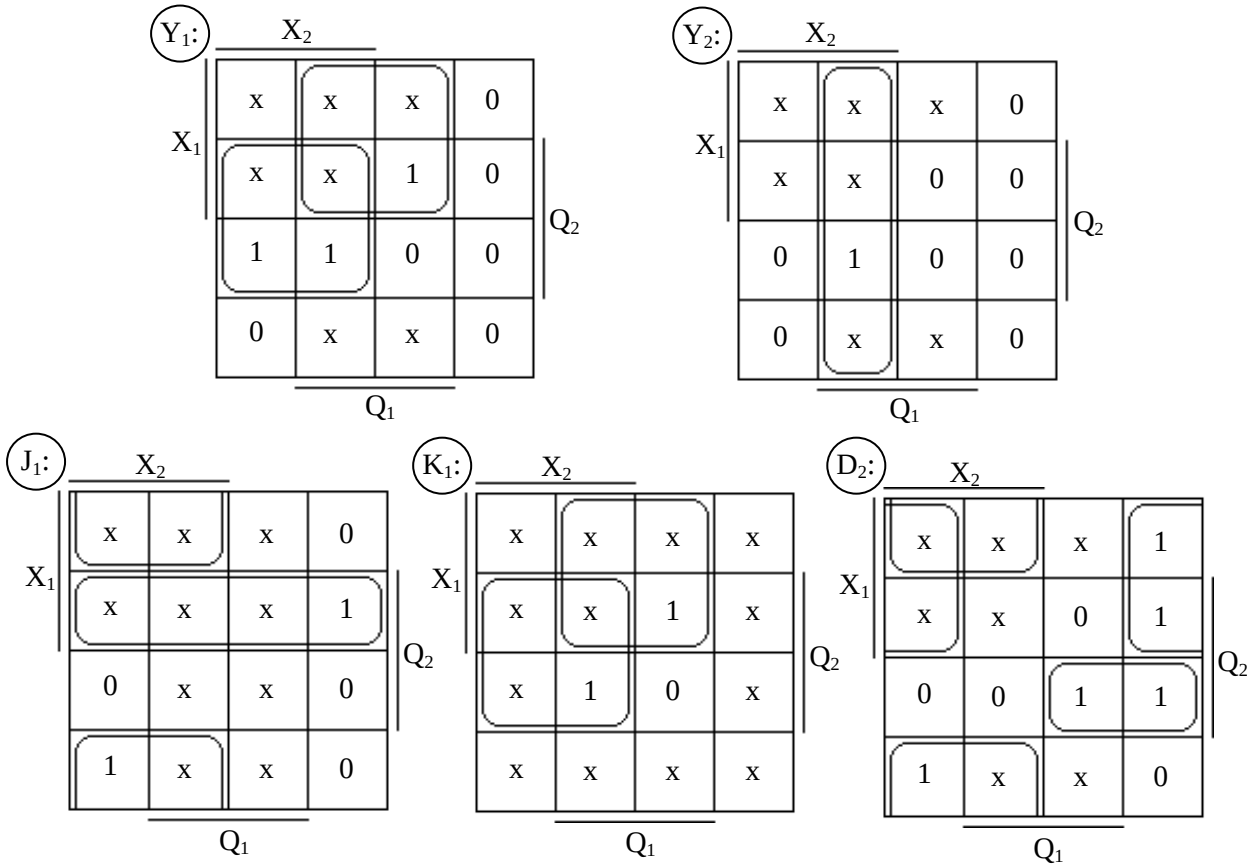
Uz malo vještine ova tablica se veoma lako očitava direktno iz grafa, bez potrebe da se prvo formira deskriptivna tablica prelaza i izlaza, čime se značajno štedi na vremenu. Za tu svrhu dovoljno je prvo upisati u tablicu sve kombinacije ulaza i bita stanja, a zatim analizirati jednu po jednu granu grafa, ustanoviti kojem redu tablice odgovara data grana (u skladu sa čvorom iz kojeg grana izlazi i ulaza koji odgovaraju grani), i zatim upisati odgovarajuće podatke u odgovarajući red na osnovu novog stanja (u skladu sa čvorom u koji grana ulazi i izlaza koji odgovaraju grani). Pri tome je radi lakšeg snalaženja preporučljivo kraj čvorova grafa ili unutar njih upisati odgovarajuće bite stanja koji su dodijeljeni stanjima prilikom kodiranja stanja. Kada se iscrpe sve grane grafa, eventualno preostale redove tablice prosto treba popuniti znacima "x". Alternativno je moguće i analizirati jedan po jedan red tablice, i utvrđivati koja grana grafa (ako takve uopće ima) odgovara posmatranom redu, i na osnovu toga izvršiti popunjavanje posmatranog reda (ukoliko odgovarajuće grane nema, red se puni znacima "x").

Ovim smo formirali kodiranu tablicu prelaza i izlaza za traženi sklop, koju bi bilo prilično teško (mada ne i nemoguće, uz dosta koncentracije) sastaviti neposredno iz verbalnog opisa rada sklopa. Drugim riječima, izvršili smo sve neophodne pripreme da bismo mogli izvršiti projektiranje sklopa formalnom metodom. Dovršimo projektiranje ovog sklopa. Nakon što smo obavili korak 1. (formiranje tablice prelaza i izlaza), prelazimo na korak 2. (izbor flip-flopova). Kako u našem primjeru imamo dva bita stanja, potrebna su nam dva flip-flopa. Kako za svaki od bita stanja možemo proizvoljno izabrati jedan od četiri flip-flopa (RS, JK, D ili T), imamo ukupno $4^2 = 16$ mogućnosti izbora (obično se biraju JK ili D flip-flopovi). Odlučimo se (posve proizvoljno) da bit stanja Q_1 realiziramo JK flip-flopom, a bit stanja Q_2 D flip-flopom (naravno, oba bita stanja mogla su se realizirati i flip-flopovima istog tipa). Nakon ovog izbora, možemo sastaviti sljedeću proširenu tablicu prelaza, izlaza i pobuda:

X_1	X_2	$Q_1(n)$	$Q_2(n)$	$Q_1(n+1)$	$Q_2(n+1)$	Y_1	Y_2	J_1	K_1	D_2
0	0	0	0	0	0	0	0	0	x	0
0	0	0	1	0	1	0	0	0	x	1
0	0	1	0	x	x	x	x	x	x	x
0	0	1	1	1	1	0	0	x	0	1
0	1	0	0	1	1	0	0	1	x	1
0	1	0	1	0	0	1	0	0	x	0
0	1	1	0	x	x	x	x	x	x	x
0	1	1	1	0	0	1	1	x	1	0
1	0	0	0	0	1	0	0	0	x	1
1	0	0	1	1	1	0	0	1	x	1
1	0	1	0	x	x	x	x	x	x	x
1	0	1	1	0	0	1	0	x	1	0
1	1	0	0	x	x	x	x	x	x	x
1	1	0	1	x	x	x	x	x	x	x
1	1	1	0	x	x	x	x	x	x	x
1	1	1	1	x	x	x	x	x	x	x

Naravno, pri praktičnom radu nećemo crtati novu tablicu, nego ćemo samo na prethodnu tablicu dopisati nove kolone. Radi lakšeg snalaženja, ulazi prvog i drugog flip-flopa su indeksirani oznakama 1 i 2. Kolone J_1 i K_1 formirane su na osnovu kolona $Q_1(n)$ i $Q_1(n+1)$ i tablice pobuda za JK flip-flop, dok je kolona D_2 formirana na osnovu kolona $Q_2(n)$ i $Q_2(n+1)$ i tablice pobuda za D flip-flop. Možemo primijetiti da su kolone D_2 i $Q_2(n+1)$ identične. Ovo je posve očekivano, jer za D flip-flop vrijedi $Q(n+1) \equiv D$. Zbog toga kolonu D_2 nismo morali ni pisati, nego smo mogli samo iznad oznake $Q_2(n+1)$ dopisati oznaku D_2. Ovaj trik uvijek treba koristiti kada se za realizaciju koriste D flip-flopovi.

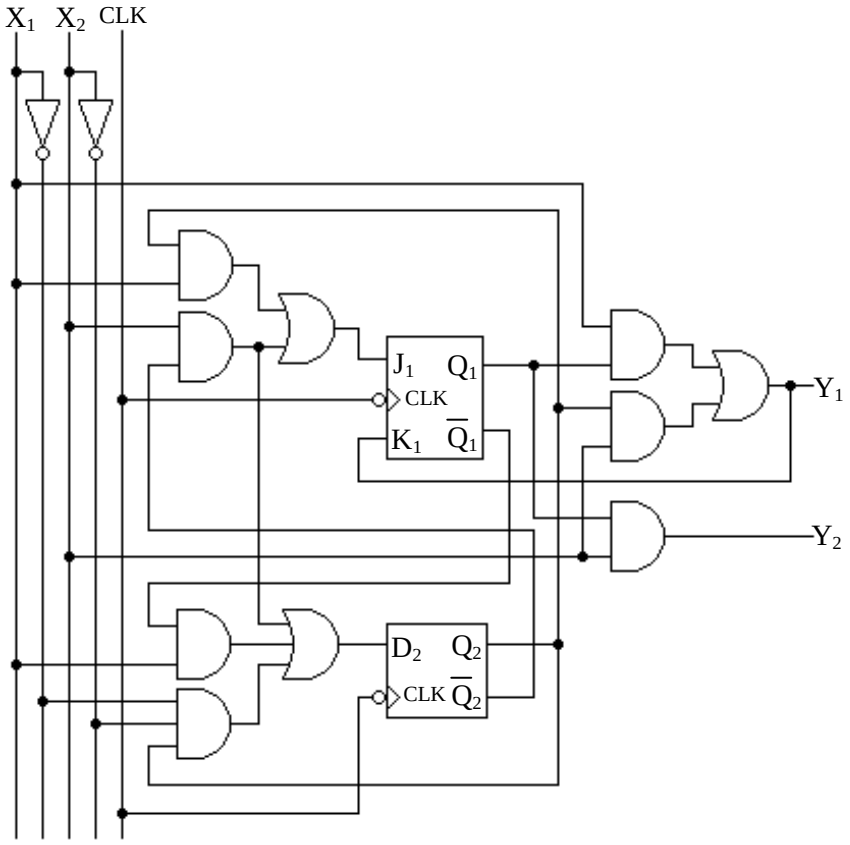
Preostaje nam još da realiziramo Y_1, Y_2, J_1, K_1 i D_2 kao funkcija od X_1, X_2, $Q_1(n)$ i $Q_2(n)$. Ovo je rutinski dio posla, koji ćemo obaviti uz pomoć nekoliko Veitchovih dijagrama:



Iz ovih dijagrama očitavamo:

$$Y_1 = X_1 Q_1 \vee X_2 Q_2$$
$$Y_2 = X_2 Q_1$$
$$J_1 = X_1 Q_2 \vee X_2 \overline{Q}_2$$
$$K_1 = X_1 Q_1 \vee X_2 Q_2$$
$$D_2 = X_1 \overline{Q}_1 \vee X_2 \overline{Q}_2 \vee \overline{X}_1 \overline{X}_2 Q_2$$

Jasno je da se za funkciju K_1 mogao izvesti i jednostavniji oblik $K_1 = X_1 \vee X_2$ (preko dvije osmice), ali na ovaj način su funkcije Y_1 i K_1 identične, što daje jednostavniju realizaciju. Preostaje još samo da nacrtamo traženi sklop, što je posve lako učiniti na osnovu gornjih jednačina (taktni impulsi se vode istovremeno na oba flip-flopa). Nažalost, sheme sekvencijalnih sklopova su gotovo uvijek veoma nepregledne, zbog prisustva brojnih povratnih sprega:



Primijetimo da smo izvršili kompletnu realizaciju sklopa a da pritom uopće nismo ni izveli jednačine koje opisuju funkcioniranje sklopa u analitičkom obliku. U slučaju potrebe, analitički model koji opisuje rad sklopa smo mogli izvesti direktno iz kodirane tablice prelaza i izlaza, izražavajući direktno $Q_1(n+1)$, $Q_2(n+1)$, Y_1 i Y_2 u funkciji od X_1, X_2, $Q_1(n)$ i $Q_2(n)$. Ovo možemo lako izvesti pomoću Veitchovih dijagrama, nakon čega se dobije sljedeći model (primijetimo da smo funkcije izlaza Y_1 i Y_2 već određivali za potrebe realizacije sklopa):

$$Q_1(n+1) = X_2 \overline{Q}_2 \vee \overline{X}_1 \overline{X}_2 Q_1 \vee X_1 \overline{Q}_1 Q_2$$
$$Q_2(n+1) = X_1 \overline{Q}_1 \vee X_2 \overline{Q}_2 \vee \overline{X}_1 \overline{X}_2 Q_2$$
$$Y_1 = X_1 Q_1 \vee X_2 Q_2$$
$$Y_2 = X_2 Q_1$$

Jasno je da analitički matematski model koji opisuje rad sklopa ne može ovisiti od izbora flip-flopova, s obzirom da su oni samo sredstvo za realizaciju sklopa, odnosno jedan od načina za memoriranje bita stanja do narednog vremenskog trenutka. S druge strane, analitički model bitno zavisi od *načina kodiranja stanja*, odnosno od načina na koji način su stanja predstavljena pomoću bita stanja, kao i uostalom od toga kako su uopće izabrana sama stanja. Kao posljedica toga, krajnja realizacija sekvencijalnog sklopa također bitno ovisi od načina kodiranja stanja (ali i od izbora flip-flopova). Na primjer, mi smo usvojili kodiranje stanja $S_0 \rightarrow 00$, $S_1 \rightarrow 01$ i $S_2 \rightarrow 11$. Bilo koje drugo kodiranje stanja, npr. $S_0 \rightarrow 11$, $S_1 \rightarrow 00$ i $S_2 \rightarrow 10$ dovelo bi do funkcionalno ekvivalentnog sklopa, ali čiji bi matematski model kao i realizacija bili drugačiji. Kao što je već rečeno, tri stanja se sa dva bita mogu kodirati na 24 načina, a malo kombinatorike pokazuje da se općenito N stanja može kodirati sa n bita na $2^n!/(2^n-N)!$ načina (uz $2^n \geq N$). Pri tome će svaki način kodiranja dati drugačiji sklop sa aspekta realizacije. Prirodno je stoga postaviti pitanje *kako izvršiti kodiranje stanja* sa ciljem da se dobije *optimalan sklop* sa aspekta realizacije (odnosno sklop sa najmanjim brojem upotrijebljenih elemenata). Na žalost, odgovor na ovo pitanje do danas nije nađen (mada postoje izvjesne smjernice koje ne vode nužno ka optimalnom rješenju), tako da je jedini postupak koji garantira pronalaženje optimalnog rješenja ispitivanje svih mogućih kodiranja stanja. Pri manjem broju stanja nije nemoguće ispitati sve moguće kombinacije, ali broj kombinacija rapidno raste sa porastom broja stanja. Na primjer, za 10 stanja (što je još uvijek sasvim mali sklop), broj mogućih kodiranja stanja sa 4 bita iznosi 16!/6! = 29059430400 kombinacija, što je nemoguće testirati u razumnom vremenu!

Opisana činjenica jasno ukazuje da su na polju projektiranja sekvencijalnih sklopova još uvijek otvorena brojna pitanja za potencijalno istraživanje. Da stvar bude još gora, sasvim je moguće stanja kodirati sa većim brojem bita nego što izosi minimalan broj bita potreban za kodiranje stanja. Na primjer, tri stanja S_0, S_1 i S_2 lijepo se mogu kodirati sa dva bita, ali ništa nas ne sprečava da izvršimo kodiranje sa *tri bita*, npr. kodiranje $S_0 \rightarrow 001$, $S_1 \rightarrow 010$ i $S_2 \rightarrow 100$ ili $S_0 \rightarrow 000$, $S_1 \rightarrow 010$ i $S_2 \rightarrow 111$. Prirodno je postaviti pitanje *ima li smisla* vršiti ovakva kodiranja, s obzirom da njihova realizacija zahtijeva više flip-flopova nego što je neophodno. Odgovor je *potvrdan*. Naime, može se desiti da neka od realizacija sa više flip-flopova zahtijeva znatno manje pratećih elemenata nego neka realizacija sa manje flip-flopova, tako da se generalno pokaže kao povoljnija (ipak, nema nikakvog smisla kodirati N stanja sa više od N bita, jer bi tada neki biti stanja uvijek imali konstantne vrijednosti 0 ili 1). To još više povećava broj potencijalnih kandidata koji bismo trebali ispitati da pronađemo optimalnu realizaciju. Na primjer, tri stanja S_0, S_1 i S_2 mogu se kodirati sa tri bita na 8!/5! = 336 načina, tako da za naš automat za Coca-Colu imamo $24 + 336 = 360$ smislenih kodiranja stanja (već za sklop sa svega 10 stanja postoji 12141181517511899528421677568000 smislenih kodiranja stanja). Broj mogućih realizacija je još veći, s obzirom da realizaciju svakog bita stanja možemo proizvoljno izabrati jedan od više različitih tipova flip-flopova, a različiti izbori vode ka različitim realizacijama (ipak, iskustvo pokazuje da upotreba JK flip-flopova u većini slučajeva vodi jednostavnijim realizacijama u odnosu na druge tipove flip-flopova). Stoga se, uz trenutno znanje kojima raspolaže teorija sekvencijalnih sklopova, *moramo pomiriti sa spoznajom* da ni za jednu realizaciju iole složenijeg sekvencijalnog sklopa *ne možemo garantirati* da ne postoji eventualno jednostavnija realizacija koja obavlja istu funkciju!

U nekim slučajevima je na osnovu verbalnog opisa rada sekvencijalnog sklopa prirodnije formirati graf kod koga su izlazi iz sklopa pridruženi *čvorovima* a ne *granama* grafa. Takvi grafovi vode ka sklopovima Mooreovog tipa, kao što ćemo vidjeti u sljedećem primjeru.

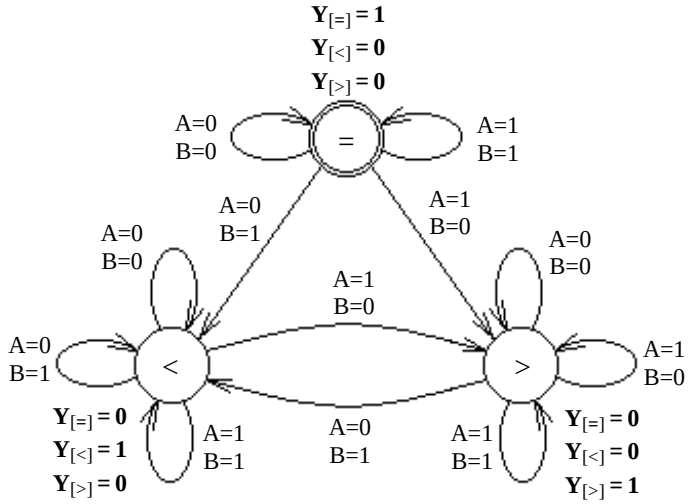
Primjer 17.4:

- Sekvencijalni sklop (nazovimo ga ***serijski komparator***) posjeduje dva ulaza A i B na koji se redom dovode biti dva broja koja se porede počev od bita *najmanje težine*. Sklop ima tri izlaza $Y_{[>]}$, $Y_{[<]}$ i $Y_{[=]}$ pri čemu logička jedinica na prvom izlazu označava da je prvi broj veći od drugog (odluka se donosi na osnovu do tada pristiglih bita), jedinica na drugom izlazu označava da je prvi broj manji od drugog dok jedinica na trećem izlazu označava da su brojevi jednaki (u bilo kojem trenutku vremena samo jedan od izlaza može biti na logičkoj jedinici). Brojevi koji se porede posmatraju se kao serija (sekvenca) bita a trenutak njihovog dolaska odreden je taktnim impulsima.

Da bismo sastavili graf koji opisuje rad ovog sklopa, analizirajmo sam postupak poređenja dva broja ukoliko analizu vršimo posmatrajući brojeve *zdesna nalijevo* (tj. od bita najmanje težine). Ako analiziramo brojeve bit po bit, sve dok su odgovarajući parovi bita jednaki, možemo pretpostavljati da bi brojevi mogli biti jednaki. Međutim, nakon što eventualno naiđemo na prvi par bita koji se razlikuje, možemo znati da brojevi ne mogu biti jednaki. Tada će odluka o tome koji je broj veći zavisiti koji je od odgovarajućih bita koji se ispituju veći. Kako se krećemo ka bitima veće težine, odluka o tome koji je broj veći može se mijenjati (u zavisnosti od trenutnog para bita koji se ispituje), tako da će konačna odluka biti donesena tek kad se ispita posljednji par bita (važno je voditi računa o tome da se parovi bita dovode na ulaz sklopa jedan po jedan, tako da sklop ne vidi bite *unaprijed*, i ne može donositi nikakvu odluku na osnovu bita koji tek treba da dođu na ulaz, već samo na osnovu onih bita koji su do tog trenutka došli na ulaz). Na osnovu ove analize možemo zaključiti da se traženi sklop u svakom trenutku vremena na osnovu onoga što je dotad poznato može nalaziti samo u jednom od tri različita stanja, koja možemo obilježiti sa "=", "<" i ">":

STANJE	ZNAČENJE
=	Na osnovu dosada pristiglih bita izgleda da su brojevi jednaki
<	Na osnovu dosada pristiglih bita izgleda da je prvi broj manji od drugog
>	Na osnovu dosada pristiglih bita izgleda da je prvi broj veći od drugog

Kada smo identificirali moguća stanja sklopa, sasvim je lako na osnovu prethodne analize formirati graf koji opisuje rad sklopa. Kako stanja sklopa u ovom slučaju jednoznačno definiraju i kakvi bi izlazi iz sklopa trebali biti (npr u stanju "=" imamo $Y_{[>]} = 0$, $Y_{[<]} = 0$ i $Y_{[=]} = 1$), mnogo je prirodnije izlaze pisati kraj čvorova grafa nego kraj grana grafa (mada bi i to bilo moguće, npr. kraj svake grane koja vodi u čvor ">" mogli bismo pisati izlaze $Y_{[>]} = 1$, $Y_{[<]} = 0$ i $Y_{[=]} = 0$). Stoga ćemo tako učiniti i u sljedećem grafu, koji opisuje rad traženog komparatora. Radi bolje preglednosti, izlazi koji su pridruženi čvorovima označeni su podebljano:



Ovakav graf vodiće do sklopa Mooreovog tipa (da smo izlaze dodijelili granama, dobili bismo sklop Mealyjevog tipa, što čitatelj odnosno čitateljka mogu uraditi kao vježbu). Sada je potrebno izvršiti kodiranje stanja sklopa. Usvojimo sljedeće kodiranje, sa dva bita stanja:

STANJE	Q_1	Q_2
=	0	0
<	0	1
>	1	0

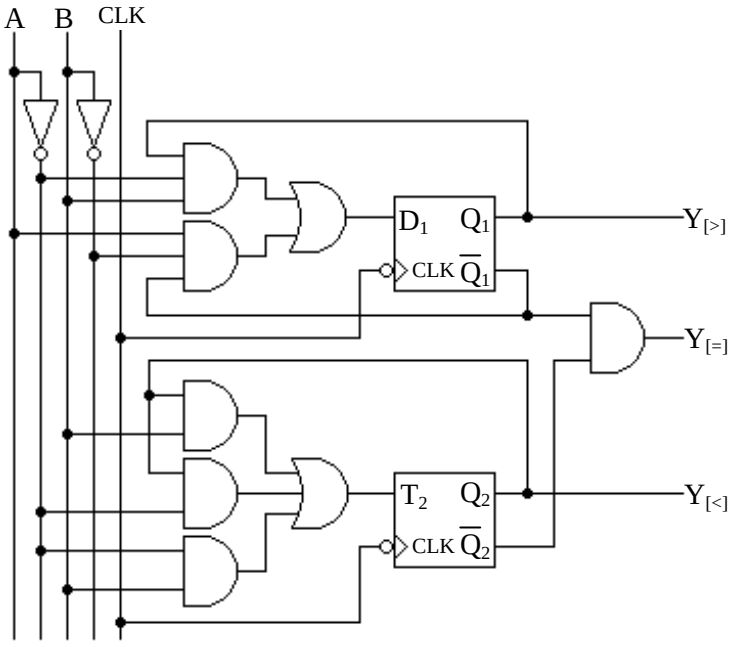
Kako kod Mooreovih modela izlazi ovise samo o trenutnim stanjima, funkcije izlaza možemo odmah odrediti na osnovu usvojenog kodiranja stanja. Na taj način, o izlazima se više ne moramo brinuti (tj. ne trebamo ih pisati u kodiranu tablicu prelaza i izlaza) čime se pojednostavljuje postupak:

Q_1	Q_2	$Y_{[>]}$	$Y_{[<]}$	$Y_{[=]}$
0	0	0	0	1
0	1	0	1	0
1	0	1	0	0
1	1	x	x	x

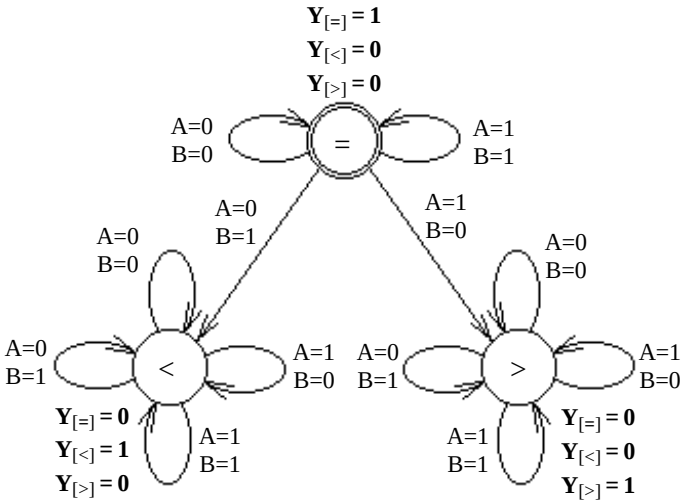
Iz ove tablice neposredno očitavamo $Y_{[>]} = Q_1$, $Y_{[<]} = Q_2$ i $Y_{[=]} = \overline{Q}_1\overline{Q}_2$. Da bismo sastavili proširenu kodiranu tablicu prelaza i pobuda, potrebno je da izaberemo flip-flopove koje ćemo koristiti. Izaberimo, na primjer, da prvi flip-flop bude T flip-flop a drugi D flip-flop. Sada, na osnovu grafa, usvojenog kodiranja stanja, i tablica pobude za T i D flip-flop neposredno slijedi tražena tablica:

A	B	$Q_1(n)$	$Q_2(n)$	$Q_1(n+1)$	$Q_2(n+1)$ $(\equiv D_2)$	T_1
0	0	0	0	0	0	0
0	0	0	1	0	1	0
0	0	1	0	1	0	0
0	0	1	1	x	x	x
0	1	0	0	0	1	0
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	x	x	x
1	0	0	0	1	0	1
1	0	0	1	1	0	1
1	0	1	0	1	0	0
1	0	1	1	x	x	x
1	1	0	0	0	0	0
1	1	0	1	0	1	0
1	1	1	0	1	0	0
1	1	1	1	x	x	x

Na osnovu ove tablice mogu se odrediti izrazi za T_1 i D_2 u funkciji od A, B, Q_1 i Q_2. Čitateljima odnosno čitateljkama se ostavlja za vježbu da sami izvedu ove izraze i provjere tačnost realizacije traženog serijskog komparatora koja je prikazana na sljedećoj slici:



Treba primijetiti da je poređenje dva binarna broja logički prirodnije ukoliko poređenje vršimo počev od bita *najveće težine* (tj. *slijeva nadesno*). Naime, u tom slučaju, čim primijetimo da se jedan par bita razlikuje, odmah možemo donijeti konačnu odluku o tome koji je broj veći, odnosno dalji biti koji dolaze ne mogu utjecati na donesenu odluku. U tom slučaju, graf koji opisuje rad komparatora mogao bi izgledati ovako:



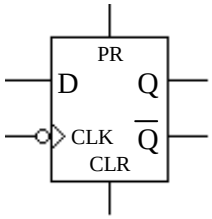
Čitateljima i čitateljkama se ostavlja za vježbu da realiziraju serijski komparator opisan ovim grafom.

Pažljivija analiza prethodnog grafa prirodno nameće izvjesna pitanja. Primijetimo da sklop opisan prethodnim grafom nakon što jednom dospije u stanja označena sa "<" ili ">" više nikada ne izlazi iz njih, bez obzira na to šta se dešava na ulazima sklopa. Stoga je logično postaviti pitanje kako uopće započeti poređenje novih brojeva, tj. novih sekvenci bita (nakon što je obavljeno poređenje jedne sekvence bita), s obzirom da sklop treba da započne rad u stanju obilježenom sa "=". Da bi se riješio ovaj problem, svi stvarni sekvencijalni sklopovi uvijek imaju jedan specijalan ulaz nazvan ***RESET*** koji služi za postavljanje sklopa u *početno stanje*. Kada se na ovom ulazu pojavi jedinica, sklop prelazi u početno stanje, neovisno od vrijednosti na ostalim ulazima. Stoga se ovaj ulaz može koristiti za inicijalizaciju sklopa. Tako, na primjer, u slučaju serijskog komparatora, kada god želimo da poredimo nove brojeve, sklop možemo dovesti u početno stanje dovođenjem signala RESET = 1.

RESET ulazi se moraju koristiti u gotovo svim sekvencijalnim sklopovima. Naime, prilikom crtanja grafa mi uvijek pretpostavljamo jedno od stanja kao početno stanje, i pretpostavljamo da će sklop započeti rad u tom stanju. Međutim, takve garancije nam ne može dati same po sebi. Na primjer, pretpostavimo da smo početno stanje sklopa kodirali kombinacijom bita stanja 00 (slična diskusija vrijedi i za sve druge kombinacije). Sklop će započeti rad u tom stanju samo ukoliko se zadesi da su oba flip-flopa koji realiziraju bite stanja resetovani. Međutim, u prethodnom poglavlju smo vidjeli da se početno stanje flip-flopova se *nikada ne može znati unaprijed* sve do prvog setovanja ili resetovanja. Prema tome, i ovdje nam je potreban RESET signal kojim možemo dovesti sklop u početno stanje bez obzira kakva su bila početna stanja flip-flopova (što zapravo određuje kakvo je zaista bilo stanje sklopa u kojem je on započeo rad).

RESET ulaz se može izvesti kao i svi drugi ulazi u sekvencijalni sklop. Za tu svrhu dovoljno je u graf koji opisuje rad sekvencijalnog sklopa dodati nove grane koje iz bilo kojeg čvora vode u početni čvor kad god je RESET = 1. Međutim, na ovaj način se projektiranje sekvencijalnih sklopova komplicira uvođenjem dodatnog ulaza. Da bi se olakšala izvedba RESET ulaza, gotovo sve praktične izvedbe flip-flopova imaju dodatni ulaz obilježen sa ***CLEAR*** (skraćeno CLR). Dovođenje jedinice na ovaj ulaz bezuvjetno resetuje flip-flop (tj. postavlja Q = 0) bez obzira šta se dešavalo na ostalim ulazima. Na taj način, ukoliko smo početno stanje sklopa kodirali sa svim bitima stanja postavljenim na 0 (što se obično radi), RESET ulaz je najlakše realizirati tako što se on dovede na CLR ulaze svih korištenih flip-flopova. Neki flip-flopovi imaju i ulaz obilježen sa ***PRESET*** (skraćeno PR), pri čemu dovođenje jedinice na ovaj ulaz bezuvjetno setuje flip-flop (tj. postavlja Q=1) beu obzira na ostale ulaze (PRESET i CLEAR se ne smiju istovremeno postaviti na 1). PRESET ulaz olakšava izvedbu RESET ulaza u slučajevima kada je

početno stanje sklopa kodirano tako da svi biti stanja nisu nule. U tom slučaju, RESET ulaz se vodi na CLEAR ulaze onih flip-flopova koji u početnom stanju trebaju biti resetovani, odnosno PRESET ulaze onih flip-flopova koji u početnom stanju trebaju da budu setovani. Sljedeća slika prikazuje tipični simbol koji prikazuje D flip-flop sa CLEAR i PRESET ulazima:



Radi jednostavnosti, u mi u nastavku nikada nećemo crtati RESET ulaze, ali se podrazumijeva da oni *postoje*. Također, vrijedi napomenuti da se CLEAR i PRESET ulazi u flip-flopove uvijek izvode *asinhrono*, tj. dovođenje jedinice na CLEAR odnosno PRESET ulaz bezuvjetno resetuje odnosno setuje flip-flop bez obzira na taktne impulse. Stoga se ovi ulazi u sinhronim sekvencijalnim sklopovima ne smiju koristiti nizašta drugo osim za ostvarivanje RESET ulaza sekvencijalnog sklopa.

Kao što se iz do sada izloženog može vidjeti, projektiranje sinhronih sekvencijalnih sklopova je manje ili više rutinski (iako često naporan i dugotrajan) posao nakon što se formira graf koji opisuje njegov rad (ili neki drugi ekvivalentni matematički model rada sklopa). Međutim, formiranje grafa na osnovu verbalnog opisa traži izvjesnu vještinu koja se stiče samo iskustvom. Sljedeći primjer ilustrira da naizgled sitni detalji mogu značajno utjecati na formiranje ispravnog modela.

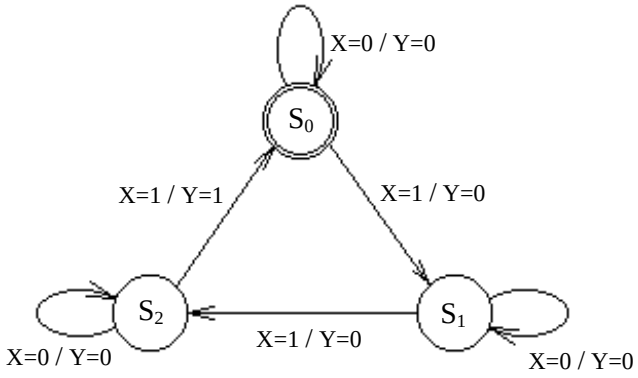
Primjer 17.5:

- Potrebno je projektirati sekvencijalni sklop sa jednim ulazom i jednim izlazom koji će na svoj ulaz propustiti *svaku treću jedinicu* koja se pojavi na ulazu. U svim ostalim slučajevima izlaz iz sklopa treba da bude nula. Predstaviti rad ovog sekvencijalnog sklopa grafom.

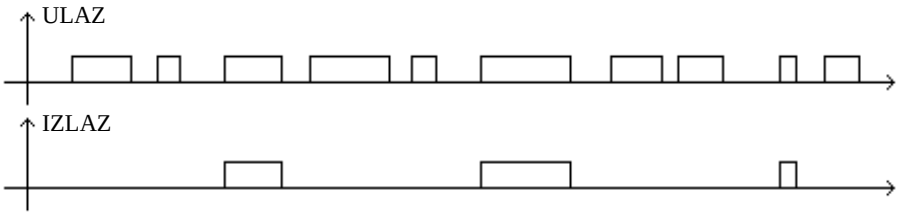
Rješenje ovog problema bitno ovisi od toga šta se podrazumijeva pod *svakom trećom jedinicom* na ulazu. Najprirodnije je pretpostaviti da se jedinice odnosno nule na ulazu odmjeravaju u trenutku nailaska taktnih impulsa (odnosno njihovih ivica ukoliko se koristi ivično okidanje), tako da pod *jedinicom* odnosno *nulom* na ulazu podrazumijevamo vrijednost ulaza u trenucima određenim taktnim impulsima. Ovakav opis je najviše u skladu sa matematičkim modelima sekvencijalnih sklopova. Dakle, uz ovakvu pretpostavku, primjer ulazne i izlazne sekvence mogao bi izgledati ovako (ulazi i izlazi su odmjeravani u trenicima određenim taktnim impulsima):

ULAZ:	0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, ...
IZLAZ:	0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, ...

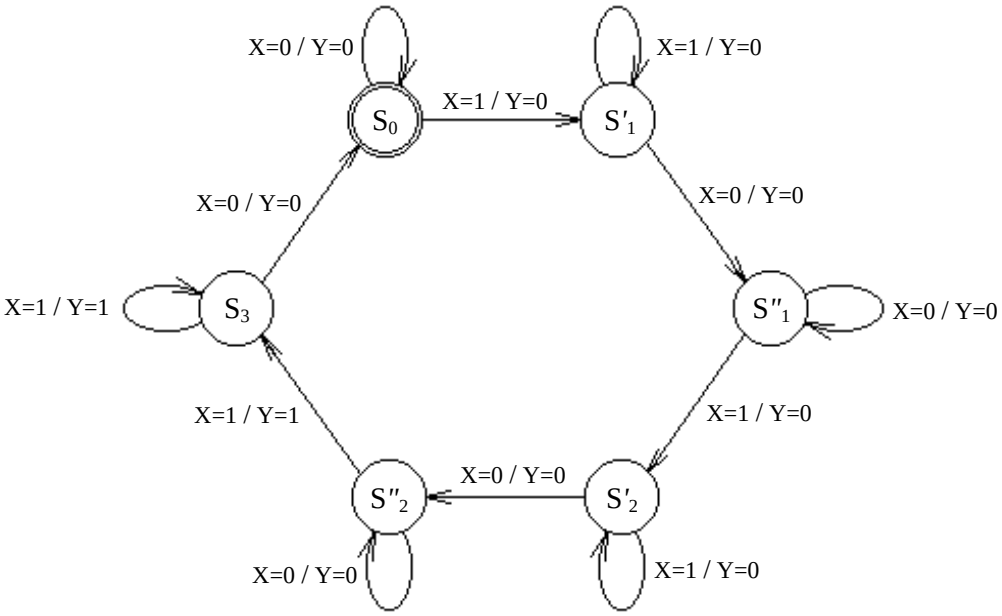
Nije teško modelirati ovakvo ponašanje grafom. Vjerovatno najjednostavnije rješenje je model Mealyjevog tipa sa tri stanja, koja vode računa koliko se jedinica pojavilo od posljednje jedinice koja je "propuštena" na izlaz:



Ovaj graf tačno opisuje ponašanje traženog sklopa, i ukoliko se eksplicitno ne kaže drugačije, mi ćemo podrazumijevati upravo ovakvu interpretaciju postavke problema. Međutim, sama postavka problema se može shvatiti i na drugačiji način. Praktične primjene mogle bi nas navesti da pod *jedinicom na ulazu* posmatramo signal čija je vrijednost 1 bez obzira na njegovo trajanje, tako da bi pod *trećom jedinicom na ulazu* zapravo smatrali *treći jedinični impuls* koji bi se pojavi na ulazu. U tom slučaju, vremensko ponašanje sklopa moglo bi se prikazati sljedećim vremenskim dijagramom, koji prikazuje ulaz i izlaz iz sklopa:



Da bismo ovakvo ponašanje realizirali sinhronim sekvencijalnim sklopom, moramo prvo pretpostaviti da taktni impulsi nailaze u mnogo kraćim razmacima nego što iznosi tipično trajanje "jedinice" na ulazu (u suprotnom se može desiti da jedinica uopće ne bude registrirana). Međutim, u tom slučaju, "jedinica" na ulazu može trajati znatno duže od jednog taktnog impulsa. Stoga, sklop mora voditi računa ne samo o tome kada se "jedinica" na ulazu *pojavila*, nego i *kada je prestala* (s obzirom da ne možemo računati da "jedinica" traje koliko i razmak između dva taktna impulsa). Ovim se broj stanja koja sklop mora posjedovati efektivno udvostručava. Jedno od mogućih rješenja prikazano je sljedećim grafom, čija analiza, nakon provedenih razmatranja, ne bi trebala da predstavlja problem (čitateljima i čitateljkama se ostavlja za vježbu da ustanove smisao svakog od 6 stanja prikazanih na grafu):



Vidimo da graf koji modelira postavljeni problem zavisi od toga kako se problem tumači. Prvo i jednostavnije rješenje bolje odgovara specifikaciji zadanoj u postavci problema. Međutim, čisto praktične situacije češće nameću potrebu za sklopovima koji se ponašaju poput drugog, složenijeg rješenja. Takvu potrebu je bitno *eksplicitno naglasiti* već pri formiranju verbalnog modela. Stoga je formiranje jasnog i preciznog *verbalnog modela* od ključne važnosti za kasnije formiranje ispravnog matematskog modela, a zatim i za realizaciju sklopa. Nejasni i neprecizni verbalni modeli mogu dovesti do sklopova koji ne rade u skladu sa očekivanjima. Ovo je naročito bitno istaknuti, jer često je upravo projektant u situaciji da mora da precizno *verbalno* opiše *kako bi se sklop trebao da ponaša* na osnovu nedovoljno preciznih i nepotpunih informacija o željenom ponašanju sklopa.

(?) Pitanja i zadaci

17.1 Objasnite šta su elementarni automati (flip-flopovi).

17.2 Objasnite ulogu povratnih sprega za potrebe realizacije sekvencijalnih sklopova.

17.3 Pokažite da se serijski spoj dva invertora zatvoren povratnom spregom može nalaziti u dva stabilna stanja.

- Add preset and clear inputs to the edge-triggered D flip-flop of Figure 6.24. Draw the logic schematic of the revised -circuit.

- Starting with the basic circuit schematic for the -master/slave J-K flip-flop, show how to add asynchronous preset and clear inputs to force the flip-flop into a 1 (preset) or 0 (clear) state. Draw a timing waveform for the preset input, clear input, clock, -master stage outputs (P, P'), and slave stage outputs (Q, Q') showing the operation of preset and clear.

- Any flip-flop type can be implemented from another type with suitable logic applied to the latter's inputs. Show how to implement a J-K flip-flop starting with a D flip-flop.
  Show how to implement a J-K flip-flop starting with a T flip-flop.
  Show how to implement a D flip-flop starting with a J-K flip-flop.
  Show how to implement a D flip-flop starting with a T flip-flop.
  Show how to implement a T flip-flop starting with a J-K flip-flop.
  Show how to implement a T flip-flop starting with a D flip-flop.

- To see this four-step process in action, let's look at another implementation of a counter. We will design a 3-bit counter that advances through the sequence 000, 010, 011, 101, 110, 000 and repeats. Not all of the possible combinations of the 3 bits represent a valid state. The unused states, 001, 100, and 111, can be used as don't-care conditions to simplify the logic.

- Primjer za sekvencijalni: Parity checker

- The odd parity checker of Section 8.1.1 generates a 1 whenever a bit stream of serial inputs contain an odd number of 1's. This is useful in a data communication subsystem for checking that transmitted data has been sent correctly. Data is transmitted as 8 data bits appended with a ninth parity bit. The 9-bit sequence must be in odd parity. That is, if the data bits have an odd number of 1's, the parity bit is 0. If the data bits have an even number of 1's, the parity bit is 1. You are to design a parity checker that asserts OK if the 9-bit sequence is correct in odd parity and ERROR otherwise.
  Is it possible to write a state diagram with a small number of states to describe the behavior of this finite state machine? Does your state diagram need to track all possible sequences of 9 bits? Consider implementing the subsystem using the parity checker FSM of Figure 8.4 in conjunction with a synchronous 4-bit counter like the TTL 74163. Draw a schematic using logic gates, a single flip-flop, and the counter. Draw a timing diagram including a bit sequence that leads to ERROR and one that leads to OK. Make sure you are using components with the same kind of timing behavior!

- Consider the following finite state machine specification: "A finite state recognizer has one input (X) and one output (Z). The output is asserted whenever the input sequence 010 has been observed, as long as the sequence 100 has never been seen."

- We are given the following description: "A 3-bit serial lock is used to allow entry to a locked room. The lock has a RESET button, an ENTER button, and a two-position switch to represent the key value being entered. When the signal UNLOCK is asserted, an electromechanical relay is released,

allowing the door to open. The unlock process begins when the operator presses RESET. He or she then sets the input switch, followed by pressing the ENTER button. This is repeated for the second and third key digits. An ERROR light should be illuminated if, after entering the three binary digits, the operator has not matched the key. The process can be retried by hitting RESET again."

- Suppose you are told that a Mealy machine is implemented with three flip-flops, two inputs, and six asynchronous outputs. Consider the complete state diagram for this machine (that is, there are no don't cares). Answer the following questions:
  What are the minimum and maximum numbers of states in the state diagram?
  What are the minimum and maximum numbers of transition arrows starting at a particular state?
  What are the minimum and maximum numbers of transition arrows that can end in a particular state?
  What are the minimum and maximum numbers of different binary patterns that can be displayed on the outputs?

- Suppose you are told that a Moore machine has five flip-flops, three inputs, and nine outputs. Answer the following questions:
  What are the minimum and maximum numbers of states in the state diagram?
  What are the minimum and maximum numbers of transition arrows starting at a particular state?
  What are the minimum and maximum numbers of transition arrows that can end in a particular state?
  What are the minimum and maximum numbers of different binary patterns that can be displayed on the outputs?

- Implement a two-input Mealy machine that produces a 1 at its single output when the values of the two inputs -differ at the time of the previous clock pulse. Show your state -diagram. Describe what each of your states is supposed to represent.

- A finite state machine has one input (X) and two outputs (Z1 and Z2). An output Z1 = 1 occurs every time the input sequence 101 is observed, provided the sequence 011 has never been seen. An output Z2 = 1 occurs every time the input 011 is observed. Note that once Z2 = 1, Z1 = 1 can never occur. Assuming the machine is to be implemented in the Mealy design style, draw the corresponding state diagram. (Hint: The minimum number of states is eight.)

- A finite state machine has one input and one output. The output becomes 1 and remains 1 thereafter when at least two 0's and at least two 1's have occurred as inputs, regardless of the order of occurrence. Assuming this is to be implemented as a Moore machine, draw a state diagram for the machine. (Hint: You can do this in nine states.)

- A Moore machine has two inputs (X1, X2) and one output (Z). Produce the state diagram or ASM chart for the machine, given the following specification. The output remains a constant value unless one of the following input sequences occurs:
  The input sequence X1 X2 = 00, 11 causes the output to become 0.
  The input sequence X1 X2 = 01, 11 causes the output to become 1.
  The input sequence X1 X2 = 10, 11 causes the output to change value.

- A sequential circuit has one input (X) and one output (Z). Draw a Mealy state diagram for each of the following cases:
  The output is Z = 1 if and only if the total number of 1's received is divisible by 3 (for example, 0, 3, 6,).
  The output is Z = 1 if and only if the total number of 1's received is divisible by 3 and the total number of 0's received is an even number greater than zero (nine states are sufficient).

- A sequential circuit has two inputs and two outputs. The inputs (X1 X2) represent a 2-bit binary number, N. If the present value of N is greater than the previous value, then Z1 = 1. If the present value of N is less than the previous value, then Z2 = 1. Otherwise, Z1 and Z2 are 0.

Derive a Mealy machine state diagram. (Hint: The machine needs only five states.)
Derive a Moore machine state diagram. (Hint: The machine needs at least 11 states.)

- A Moore machine has one input and one output. The output should be 1 if the total number of 0's received at the input is odd and the total number of 1's received is an even number greater than 0. This machine can be implemented in exactly six states. Draw a complete state diagram. Indicate what each state is meant to represent.

- You are to design a Mealy state diagram for a digital lock. Assume that two debounced push-buttons, A and B, are available to enter the combination. An electromechanical interlock guarantees that the buttons cannot be activated simultaneously. The lock should have the following features:

- The combination is A-A-B-A-B-A. If this sequence is correctly entered, an output signal is asserted that causes the lock to open.
  For any state, three B pulses in a row should guarantee to reset the control to its initial state.
  When any out-of-sequence use of the A push-button occurs, an output is asserted that rings a bell to warn that the lock is being tampered with. Once the lock is open, pressing either A or B will cause the lock to close without signaling an error. Draw a Mealy state diagram for this finite state machine. Indicate what each state represents and what input conditions cause state and output changes. Not everything may have been specified, so write down any assumptions you make.

- Design a state diagram to perform the following function. There are two data inputs A and B, a check input C, and an output D. The FSM takes as input two continuous, synchronous streams of 4-bit twos complement numbers in a bit-serial form with the most significant (sign) bit first. The least significant bit is marked by a 1 on the check line (C). During the time slot in which C is asserted, the output D should go to a 1 if the twos complement number on A is larger than the twos complement number on B.
  Complete the timing diagram in Figure Ex8.24 to make sure you fully understand the statement of the problem.

Draw a state diagram that implements this specification using as few states as possible. (Note: It is possible to implement this machine in six or fewer states.)

- Consider the combination lock finite state machine of Section 8.5.4 (bio nedavno). Add another input, CHANGE COMBINATION, similar to ENTER, that makes it possible to change the lock value once the door has been unlocked. Assume that the new combination is available on three switches. Draw a new state chart incorporating this change.

- Given the state diagram in Figure Ex9.16 and the state assignment S0 = 000, S1 = 001, S2 = 010, S3 = 011, S4 = 100, and S5 = 101, do the following:
  Write the encoded state table, and derive the minimized Boolean equations for implementing the next-state and output functions. Assume the state registers are implemented with D flip-flops.
  Repeat the above, but this time using J-K flip-flops.

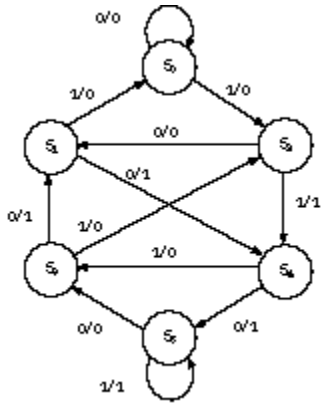


Figure Ex9.16 State diagram for Exercise 9.16.

- Given the state diagram in Figure Ex9.17 and the state assignment A = 000, B = 001, C = 011, D = 111, E = 101, implement the state machine using a minimum number of gates and J-K flip-flops. You may assume that an external RESET signal places the machine in state A (000).

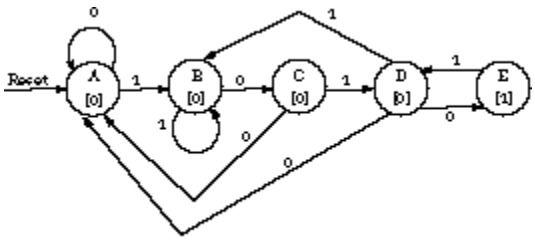


Figure Ex9.17 State diagram for Exercise 9.17.

- Implement the following finite state machine description using a minimum number of states and a good state assignment, assuming D flip-flops are used for the state registers. The machine has a single input X, a single output Z, and will assert Z = 1 for every input sequence ending in the string 0010 or 100.

- (Design Process) Your task is to implement a finite state machine with toggle flip-flops given the following state diagram. The finite state machine is actually a complex Gray code counter. The counter has two control inputs, I1 and I0, which determine the next state. The counter's functional specification is as follows. When I1 I0 = 00, it is a Gray code up-counter. When I1 I0 = 01, it is a Gray code down-counter. When I1 I0 = 10, it is a Gray code count by two. Finally, when I1 I0 = 11, the counter holds it current state. The state diagram is shown in Figure Ex9.19.
  Complete a state transition table, including the next-state bits (Q1 and Q0) and the needed inputs to the two toggle state flip-flops T1 and T0 to obtain that next state.
  Produce the four-variable K-maps for the next-state functions. Obtain the minimized two-level implementation.
  Draw an implementation schematic, using a minimum number of inverters and two-input NAND, NOR, XOR, and XNOR gates. Assume that complements are not available.

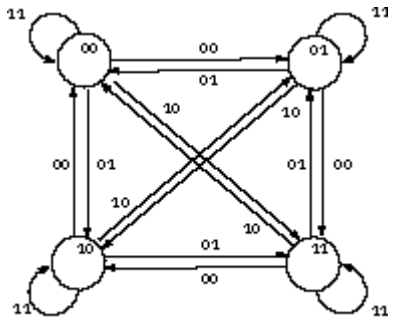


Figure Ex9.19 State diagram for Exercise 9.19.

- Design a Mealy finite state machine with input X and output Z. The output Z should be asserted for one clock cycle whenever the sequence 0111 or 1000 has been input on X. The patterns may overlap. For example, X = 0000111000 should generate the output stream Y = 0000001001.
  Complete the state diagram for the sequence detector, without concern for state minimization.
  Complete the state table for the state diagram derived in part (a).
  Use row matching or implication charts to minimize the state table derived in part (b).
  Use the state assignment guidelines to obtain a good state assignment for the reduced state machine of part (c). Justify your method in terms of the high-, medium-, and low-priority assignment guidelines.

Implement your encoded, reduced state table using D flip-flops.
Implement your encoded, reduced state table using T flip-flops.
Implement your encoded, reduced state table using J-K flip-flops.

- Primjer: BCD to EXCESS3 serijski konverter: puni dijagram stanja, reducirani

- Primjeri sekvencijalnog sa ROM-om, PLA-om

- Objasniti koje uvjete treba zadovoljavati neki sekvencijalni sklop da bi bio sinhroni sekvencijalni sklop, a zatim objasniti zbog čega su sinhroni sekvencijalni sklopovi izuzetno značajni za potrebe prakse.

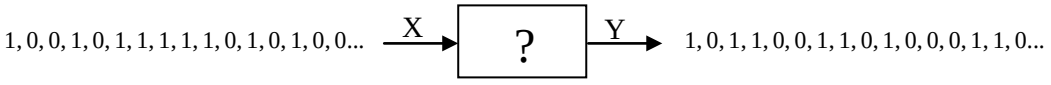
- Napraviti RS flip-flop ukoliko nam je na raspolaganju jedan T flip-flop, i proizvoljan broj osnovnih logičkih kola, tj. AND, OR i NOT logičkih kola.

- Napraviti JK flip-flop ukoliko nam je na raspolaganju jedan D flip-flop, i proizvoljan broj NOR logičkih kola.

- Nacrtati graf stanja za serijski sabirač (sumatora) i projektovati njegovu strukturu koristeći formalne metode projektovanja sekvencijalnih sklopova ukoliko nam je na raspolaganju jedan T flip-flop i proizvoljan broj AND, OR i NOT logičkih kola. Da li je ovaj sklop Mealyjev ili Mooreov automat? Obrazložiti odgovor.

- Potrebno je projektovati serijski binarni komparator sa dva ulaza x_1 i x_2, i tri izlaza y_1, y_2 i y_3. Na ulaze x_1 i x_2 dovode se respektivno biti dva binarna broja A i B koja se porede, redom počev od bita najveće težine. Na izlazima sklopa treba da se pojavi logička jedinica na samo jednom od izlaza y_1, y_2 i y_3 ovisno od toga da li je $A > B$, $A < B$ ili $A = B$ respektivno.
  a) Nacrtati graf stanja ovog sekvencijalnog sklopa.
  b) Projektovati detaljnu strukturu ovog sklopa ukoliko su nam na raspolaganju jedan JK flip-flop, jedan D flip-flop, te osnovna logička kola (AND, OR, NOT).
  c) Da li je ovaj sklop Mealyjev ili Mooreovv automat? Obrazložiti odgovor.

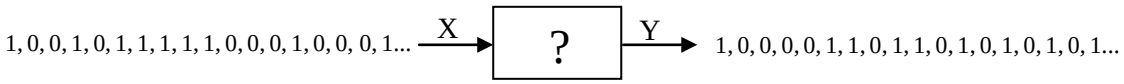
- Radi povećanja sigurnosti prenosa podataka između dva računara, potrebno je projektovati sekvencijalni sklop sa jednim ulazom i jednim izlazom, koji izvrće (invertira) svaki treći bit koji mu se dovede na ulaz, dok sve ostale bite ostavlja nepromijenjenim. Na primjer, ukoliko se na ulaz ovog sklopa dovede niz bita 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, na izlazu sklopa će se pojaviti niz bita 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0 (invertovan je svaki treći bit).



  a) Nacrtati graf Mealyjevog tipa kojim se opisuje rad ovog sekvencijalnog sklopa.
  b) Projektovati strukturu ovog sekvencijalnog sklopa koristeći formalne metode projektovanja sekvencijalnih sklopova. Na raspolaganju su jedan JK flip-flop, jedan D flip-flop i proizvoljan broj NAND logičkih kola.
  c) Pretvoriti graf Mealyjevog tipa koji opisuje rad ovog sekvencijalnog sklopa u ekvivalentni graf Mooreovog tipa.

- Potrebno je projektovati sekvencijalni sklop sa jednim ulazom i jednim izlazom, koji izvrće (invertira) svaki četvrti bit koji mu se dovede na ulaz, dok sve ostale bite ostavlja nepromijenjenim. Na primjer, ukoliko se na ulaz ovog sklopa dovede niz bita 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1

na izlazu sklopa će se pojaviti niz bita $1,0,0,0,0,1,1,0,1,1,0,1,0,1,0,1,0,1$ (invertovan je svaki četvrti bit).

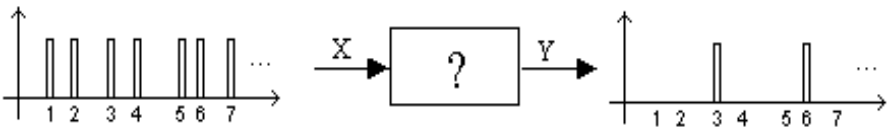


a) Nacrtati graf kojim se opisuje rad ovog sekvencijalnog sklopa.

b) Projektovati strukturu ovog sekvencijalnog sklopa koristeći formalne metode projektovanja sekvencijalnih sklopova. Na raspolaganju su jedan RS flip-flop, jedan T flip-flop i proizvoljan broj AND, OR i NOT logičkih kola.

c) Da li je ovaj sklop Mealyjev ili Mooreov automat? Obrazložiti odgovor.

- Potrebno je projektovati sekvencijalni sklop koji će na izlaz propustiti svaki treći impuls koji mu se dovede na ulaz, a ostale ignorirati, kao što je prikazano na sljedećoj slici:



a) Nacrtati graf stanja ovog sekvencijalnog sklopa.

b) Projektovati detaljnu strukturu ovog sklopa ukoliko su nam na raspolaganju jedan RS flip-flop, jedan T flip-flop, te osnovna logička kola.

c) Da li je ovaj sklop Mealyjev ili Mooreov automat? Obrazložiti odgovor.

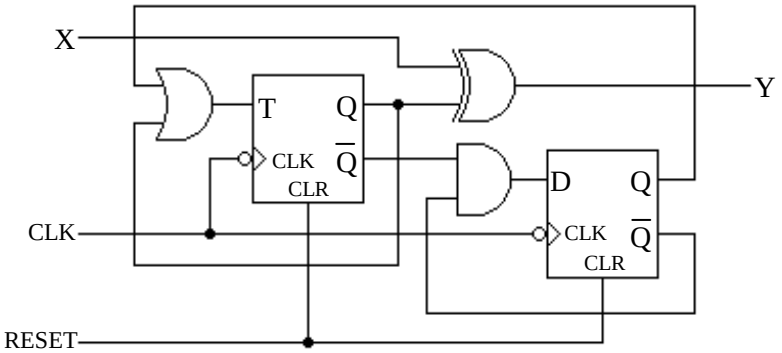
18. Analiza i transformacije sinhronih sekvencijalnih sklopova

U prethodnom poglavlju smo vidjeli da je projektiranje sinhronih sekvencijalnih sklopova u potpunosti formalizirano od trenutka kada je poznat njihov matematski model (mada sam postupak može biti veoma glomazan). Pored toga što omogućavaju relativno jednostavno projektiranje, analiza sinhronih sekvencijalnih sklopova je također sasvim jednostavna (za razliku od općenitih sekvencijalnih sklopova, čija analiza može biti praktično neizvodljiva bez pomoći računara, čak i u relativno jednostavnim slučajevima). Razlog za ovu jednostavnost leži u činjenici da svi problemi pri analizi sekvencijalnih sklopova nastaju usljed prisustva povratnih sprega, a sinhroni sekvencijalni sklopovi mogu imati samo strogo kontrolirane povratne sprege koje obavezno prolaze kroz neki od flip-flopova, čije je ponašanje veoma precizno određeno.

Prije nego što demonstriramo jednostavnost analize sinhronih sekvencijalnih sklopova, recimo šta se tačno podrazumijeva pod analizom sekvencijalnog sklopa. Mada je pravi cilj analize ustanoviti *šta radi*, odnosno po mogućnosti *čemu služi* dati sekvencijalni sklop, smatra se da je analiza sekvencijalnog sklopa izvršena onog trenutka kada je poznat njegov *matematski model*, bilo u formi tablice prelaza i izlaza, funkcija prelaza i izlaza, ili grafa. Pri tome je najpoželjnije imati model u formi grafa, jer je iz grafa najlakše zaključiti čemu bi sklop eventualno mogao da služi, odnosno kako bi mogao glasiti njegov *verbalni model*. Stoga ćemo pod analizom sekvencijalnog sklopa podrazumijevati postupak kojim se iz šeme sekvencijalnog sklopa može rekonstruisati *graf koji opisuje njegov rad* (i, eventualno, iz grafa zaključiti šta bi sklop mogao da radi). Ovaj postupak je najlakše ilustrirati na konkretnim primjerima.

Primjer 18.1:

- Analizirati sekvencijalni sklop prikazan na slici.



Na prvom mjestu, treba primijetiti da ovaj sklop zaista *jeste* sinhroni sekvencijalni sklop, odnosno da su ispunjeni svi uvjeti koje treba da zadovolji sinhroni sekvencijalni sklop. Označimo izlaz iz lijevog flip-flopa sa Q_1, a izlaz iz desnog flip-flopa sa Q_2. Sada, sa sheme lako možemo očitati sljedeće relacije, koje prikazuju zavisnosti izlaza Y i ulaza u flip-flopove od ulaza X i vrijednosti bita stanja:

$$T = Q_1 \vee Q_2$$

$$D = \overline{Q}_1 \overline{Q}_2$$

$$Y = X \oplus Q_1$$

Sada, ako znamo da se T flip-flop opisuje funkcijom prelaza $Q(n+1) = T \oplus Q(n)$ a taktovani D flip-flop funkcijom prelaza $Q(n+1) = D$, možemo pisati sljedeće relacije:

$$Q_1(n+1) = Q_1 \oplus T = Q_1 \oplus (Q_1 \vee Q_2) = \overline{Q}_1 Q_2$$

$$Q_2(n+1) = D = \overline{Q}_1 \overline{Q}_2$$

Ove relacije očigledno predstavljaju funkcije prelaza traženog sekvencijalnog sklopa. Zajedno sa relacijom $Y = X \oplus Q_1$ koja predstavlja funkciju izlaza, dobijamo potpuni matematički model traženog sklopa u *analitičkom obliku*:

$$Q_1(n+1) = \overline{Q}_1 Q_2$$

$$Q_2(n+1) = \overline{Q}_1 \overline{Q}_2$$

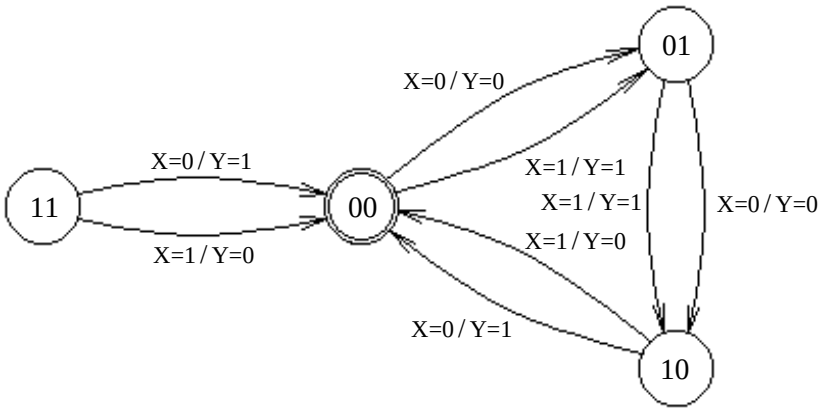
$$Y = X \oplus Q_1$$

Međutim, dobijeni analitički model je veoma nedeskriptivan, i teško je iz njega išta zaključiti o ponašanju sklopa. Nešto deskriptivniji je *tabelarni model*, u vidu tablice prelaza i izlaza, koji je sasvim lako formirati na osnovu prethodnog analitičkog modela:

X	$Q_1(n)$	$Q_2(n)$	$Q_1(n+1)$	$Q_2(n+1)$	Y
0	0	0	0	1	0
0	0	1	1	0	0
0	1	0	0	0	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	0	1
1	1	0	0	0	0
1	1	1	0	0	0

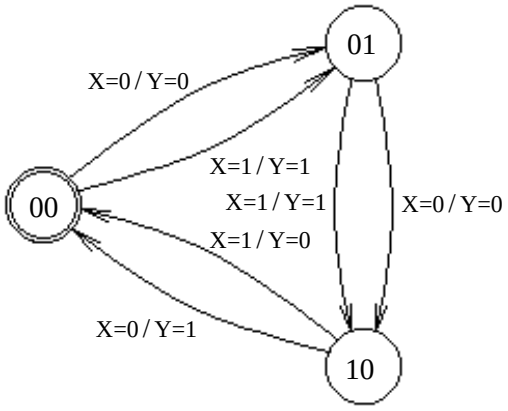
Prije nego što formiramo model u vidu grafa, i pokušamo ustanoviti šta ovaj sklop zapravo radi, razmotrimo zašto je uopće bilo potrebno da sklop koji smo analizirali bude *sinhroni* sekvencijalni sklop. Prvo, nepostojanje povratnih sprega koje ne prolaze kroz flip-flopove garantira da se ni pri kakvom raspisivanju jednačina ista promjenljiva ne može pojaviti u istom vremenskom trenutku sa obje strane znaka jednakosti. Drugo, činjenica da su svi flip-flopovi taktovani na isti način i iz istog izvora taktnih impulsa garantira da svi flip-flopovi imaju isti "osjećaj za vrijeme", tako da se ista promjenljiva *n* može koristiti za opis rada svih flip-flopova u sklopu.

Graf koji opisuje rad sklopa sasvim je lako formirati na osnovu tablice prelaza i izlaza, koristeći obrnuti postupak od postupka koji se kod projektiranja sekvencijalnih sklopova koristi za pretvaranje grafovskog modela u tabelarni model. Kako imamo dva bita stanja, ovaj sklop može imati najviše $2^2 = 4$ stanja. Označimo ova stanja upravo sa "00", "01", "10" i "11" (prema kombinaciji bita stanja), da ne uvodimo posebne oznake za njih. Tada, prateći tabelu veoma lako možemo sastaviti sljedeći graf:



Činjenicu da je stanje "00" početno stanje zaključili smo posmatrajući kako je spojen RESET signal. Kako se on dovodi na CLR ulaze oba flip-flopa, očigledno nailaskom RESET signala oba flip-flopa postaju resetovana, odnosno $Q_1 = Q_2 = 0$, odnosno stanje "00" je zaista početno stanje.

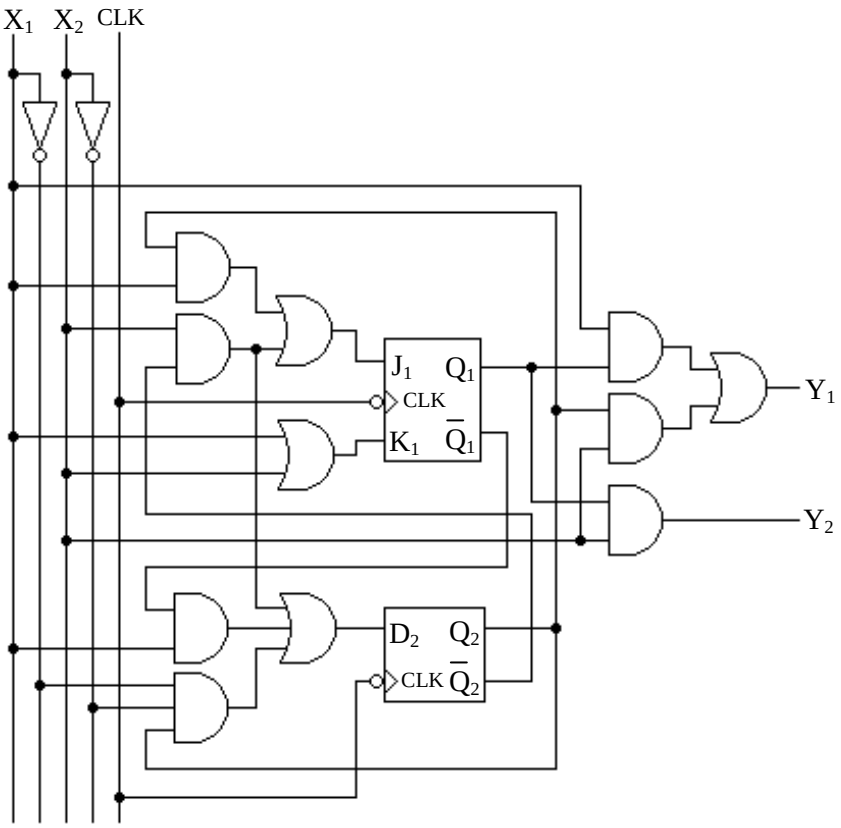
Mada smo ovim formirali graf koji opisuje sklop, dobijeni graf se može pojednostaviti ako uočimo da je stanje označeno sa "11" ***nedostižno***. Naime, u čvor "11" ne vodi *niti jedna grana*, tako da ako sklop započne rad iz stanja "00", nikakva sekvenca ulaza ne može ga dovesti u stanje "11". Stoga, stanje "11" možemo u potpunosti *eliminirati*, što nas dovodi do sljedećeg grafa:



Kada znamo graf koji opisuje rad sklopa, sasvim lako možemo odrediti kakva će biti sekvenca vrijednosti na izlazu za svaku zadanu sekvencu vrijednosti na ulazu. Na primjer, ukoliko na ulaz X dovedemo sekvencu bita 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0 na izlazu Y ćemo dobiti sekvencu bita 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0 što lako možemo utvrditi prateći graf. Nakon malo razmišljanja nije teško utvrditi ni šta ovaj sklop zapravo radi: on svaki treći bit koji mu se dovede na ulaz *invertira*, dok ostale bite prosljeđuje *neizmijenjene* na izlaz.

Primjer 18.2:

- Analizirati sekvencijalni sklop prikazan na slici.



Kao i u prethodnom primjeru, sa slike lako utvrđujemo sljedeće relacije:

$$Y_1 = X_1 Q_1 \vee X_2 Q_2$$
$$Y_2 = X_2 Q_1$$
$$J_1 = X_1 Q_2 \vee X_2 \overline{Q}_2$$
$$K_1 = X_1 \vee X_2$$
$$D_2 = X_1 \overline{Q}_1 \vee X_2 \overline{Q}_2 \vee \overline{X}_1 \overline{X}_2 Q_2$$

Prve dvije relacije očito predstavljaju *funkcije izlaza*. Da bismo dobili *funkcije prelaza*, moramo iskoristiti funkcije prelaza za JK i D flip-flop, tako da dobijamo sljedeće relacije:

$$Q_1(n+1) = J_1 \overline{Q}_1 \vee \overline{K}_1 Q_1 = (X_1 Q_2 \vee X_2 \overline{Q}_2)\overline{Q}_1 \vee \overline{X_1 \vee X_2} Q_1 = X_1 \overline{Q}_1 Q_2 \vee X_2 \overline{Q}_1 \overline{Q}_2 \vee \overline{X}_1 \overline{X}_2 Q_1$$

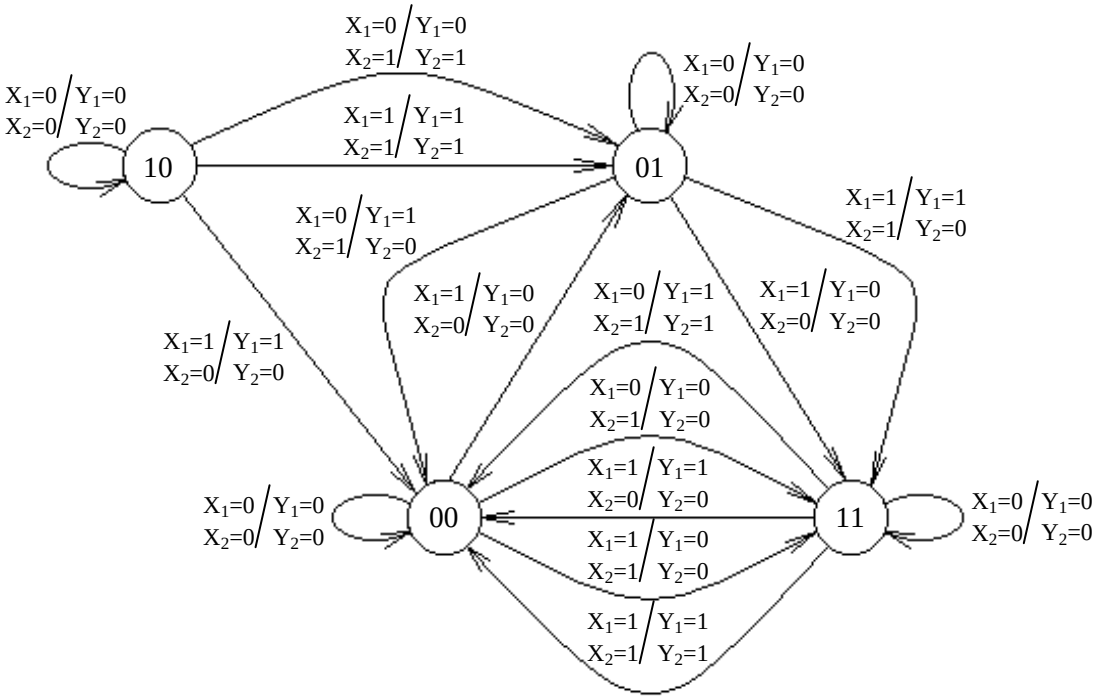
$$Q_2(n+1) = D_2 = X_1 \overline{Q}_1 \vee X_2 \overline{Q}_2 \vee \overline{X}_1 \overline{X}_2 Q_2$$

Ove dvije relacije zajedno sa funkcijama izlaza tvore traženi matematski model u analitičkom obliku. Mada ovaj model u potpunosti opisuje rad sklopa, teško je na osnovu ovog modela predvidjeti ponašanje sklopa na proizvoljne sekvence vrijednosti na ulazima.

Sljedeći korak ka razumijevanju sklopa je formiranje modela u tabelarnom obliku. Primijetimo da se tabela prelaza i izlaza može odrediti na osnovu polaznih jednačina i bez potrebe za određivanjem funkcija prelaza. Naime, na osnovu jednačina za J_1 i K_1 moguće je prvo odrediti J_1 i K_1 za svaku moguću kombinaciju vrijednosti za X_1, X_2, Q_1 i Q_2, a zatim na osnovu određenih vrijednosti za J_1 i K_1 i vrijednosti $Q_1(n)$ pomoću funkcije prelaza za JK flip-flop odrediti vrijednost $Q_1(n+1)$. Ovo je urađeno u sljedećoj tabeli, u kojoj su prikazane i pomoćne kolone u kojima su izračunate vrijednosti za J_1 i K_1.

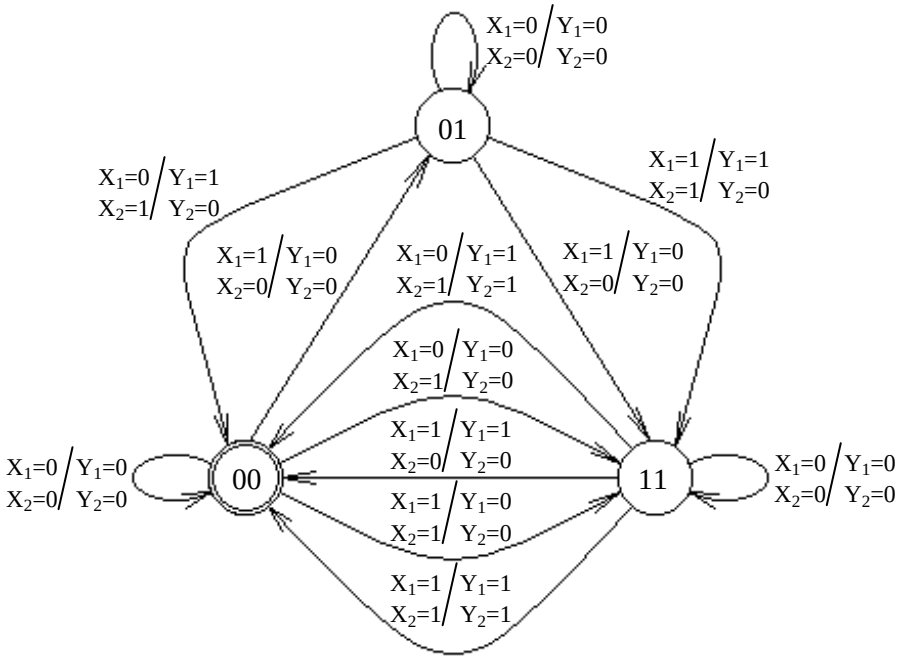
X_1	X_2	$Q_1(n)$	$Q_2(n)$	Y_1	Y_2	J_1	K_1	$Q_1(n+1)$	$Q_2(n+1)$
0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	1
0	0	1	0	0	0	0	0	1	0
0	0	1	1	0	0	0	0	1	1
0	1	0	0	0	0	1	1	1	1
0	1	0	1	1	0	0	1	0	0
0	1	1	0	0	1	1	1	0	1
0	1	1	1	1	1	0	1	0	0
1	0	0	0	0	0	0	1	0	1
1	0	0	1	0	0	1	1	1	1
1	0	1	0	1	0	0	1	0	0
1	0	1	1	1	0	1	1	0	0
1	1	0	0	0	0	1	1	1	1
1	1	0	1	1	0	1	1	1	1
1	1	1	0	1	1	1	1	0	1
1	1	1	1	1	1	1	1	0	0

Graf koji opisuje rad sklopa možemo formirati na isti način kao u prethodnom primjeru. I u ovom slučaju ćemo imati najviše $2^2 = 4$ stanja, koja možemo ponovo označiti sa "00", "01", "10", "11". Kako u ovom sklopu imamo ne jedan nego dva ulaza, iz svakog čvora moraju izlaziti po $2^2 = 4$ grane, po jedna za svaku od 4 moguće kombinacije vrijednosti ulaza X_1 i X_2. Stoga će dobijeni graf biti znatno glomazniji nego u prethodnom primjeru, mada ga je u principu lako sastaviti. Jedan od mogućih izgleda dobijenog grafa prikazan je na sljedećoj slici (razumljivo je da se isti graf može nacrtati na praktično neograničeno mnogo različitih načina, koji su u suštini identični):



Kako na shemi sklopa nije prikazan RESET signal, ne možemo znati koje je stanje početno. Međutim, primijetimo da je stanje "10" po nečemu karakteristično u odnosu na ostala stanja. Naime, u ovo stanje ne vodi nijedna grana (osim petlje iz njega samog), tako da je ono *nedostižno*, osim u slučaju da je upravo ono početno stanje. Međutim, čak i ukoliko ono jeste početno stanje, sklop se više nikada ne može u njega vratiti nakon što ga jednom napusti. Stanja koja imaju osobinu da se sklop više ne može u njih vratiti nakon što ih jednom napusti nazivaju se ***prolazna stanja***. U svakom slučaju, stanje "10" je upravo prolazno stanje.

Kad god nam je poznato početno stanje, sa grafa je potrebno ukloniti sva *nedostižna stanja*, odnosno stanja u koja je nemoguće doći iz početnog stanja prateći strelice na grafu. Ukoliko je npr. poznato da je početno stanje ovog sklopa stanje "00", stanje "10" je nedostižno, tako da ga možemo eliminirati, nakon čega graf postaje mnogo jednostavniji (isto bi vrijedilo da je početno stanje bilo koje drugo stanje osim samog stanja "10"):



Ovim je završena analiza i ovog sklopa. Time nam je funkcioniranje sklopa u potpunosti *poznato*, u smislu da za svaku unaprijed zadanu sekvencu vrijednosti na ulazima X_1 i X_2 možemo nedvosmisleno odrediti kakva će biti sekvenca vrijednosti na izlazima Y_1 i Y_2. Ipak, veoma je teško zaključiti šta sklop zapravo radi, odnosno *čemu ovaj sklop služi*. Ako bismo pokušali da sastavimo *verbalni opis* rada sklopa, u najboljem slučaju bismo mogli iskonstruisati neki tekst u kojem bi se spominjale razne kombinacije nula i jedinica.

Primijetimo da je sklop koji smo analizirali u Primjeru 18.2 zapravo sklop iz automata za Coca-Colu iz Primjera 17.3. (uz neznatno drugačiji, ali u datom kontekstu funkcionalno ekvivalentan oblik za funkciju K_2). Međutim, jasno je da postupkom analize sklopa sigurno ne možemo doći do takvog verbalnog opisa. Primijetimo također da dobijeni graf ima grane koje odgovaraju kombinacijama ulaza $X_1 = X_2 = 1$, mada je u postavci problema u Primjeru 17.3 rečeno da se ta kombinacija *ne može desiti*. To je isto činjenica koja se ne može doznati analizom sklopa, jer se prilikom analize pretpostavlja da su sve kombinacije ulaza *moguće*, a eventualne informacije o zabranjenim kombinacijama mogu jedino biti date kao *dopunske informacije*. Stoga, graf koji smo dobili kao rezultat analize pokazuje *kako bi opisani automat za Coca-Colu reagirao kada bi istovremeno ubacivanje kovanica od 1 KM i 2 KM bilo moguće*. Također, primijetimo da ukoliko sa ovog grafa uklonimo grane koje odgovaraju kombinacijama ulaza $X_1 = X_2 = 1$, dobićemo principijelno isti graf kao u Primjeru 17.3, samo nacrtan na drugi način. Ovo je, uostalom, posve očekivano, s obzirom da se radi o istom sklopu.

Do sada smo vidjeli da se matematski modeli sekvencijalnih sklopova mogu izraziti u vidu automata *Mealyjevog tipa* ili *Moorovog tipa*. Međutim, do sada razmotreni postupci za projektiranje automata Mealyjevog tipa daju sklopove koji u jednom ne baš zanemarljivom detalju odstupaju od definicije diskretnog, pa samim tim i definicije digitalnog sistema! Naime, diskretni sistemi bi trebali da reagiraju na vrijednosti ulaza samo u *fiksnim, unaprijed određenim trenucima vremena.* S druge strane, kod svih projektiranih sklopova Mealyjevog tipa uvijek postoji *direktan put* od ulaza ka izlazu, koji ne prolazi niti kroz jedan flip-flop (razlog za ovo je prilično očigledan iz samog postupka projektiranja). Na primjer, u Primjeru 18.1 postoji direktan put od ulaza X do izlaza Y, dok u Primjeru 18.2 (automat za Coca-Colu) postoje direktni putevi od ulaza X_1 do izlaza Y_1 i od ulaza X_2 do izlaza Y_1 i Y_2. Stoga, bez obzira što flip-flopovi mijenjaju stanja samo u diskretnim trenucima vremena (koja su određena trenucima nailaska taktnih impulsa), izlazi mogu promijeniti vrijednost praktično *istog trena kada se promijene i ulazi* (ako zanemarimo kašnjenja koja unose logička kola), bez obzira na taktne impulse. Tako se u Primjeru 18.2. vrijednost izlaza Y_1 može promijeniti čim se promijeni vrijednost ulaza X_1 ili X_2, dok se vrijednost izlaza Y_2 može promijeniti čim se promijeni vrijednost ulaza X_2, čak ukoliko do te promjene dođe *između dva taktna impulsa*. S druge strane, kod sklopova Mooreovog tipa ne dolazi do ovog problema, jer su njihovi izlazi direktno vezani samo na izlaze flip-flopova (s obzirom da su izlazi funkcije samo bita stanja), koji se mijenjaju samo u diskretnim trenucima. Stoga se i izlazi sklopova Mooreovog tipa mijenjaju isključivo sinhronizirano sa taktnim impulsima.

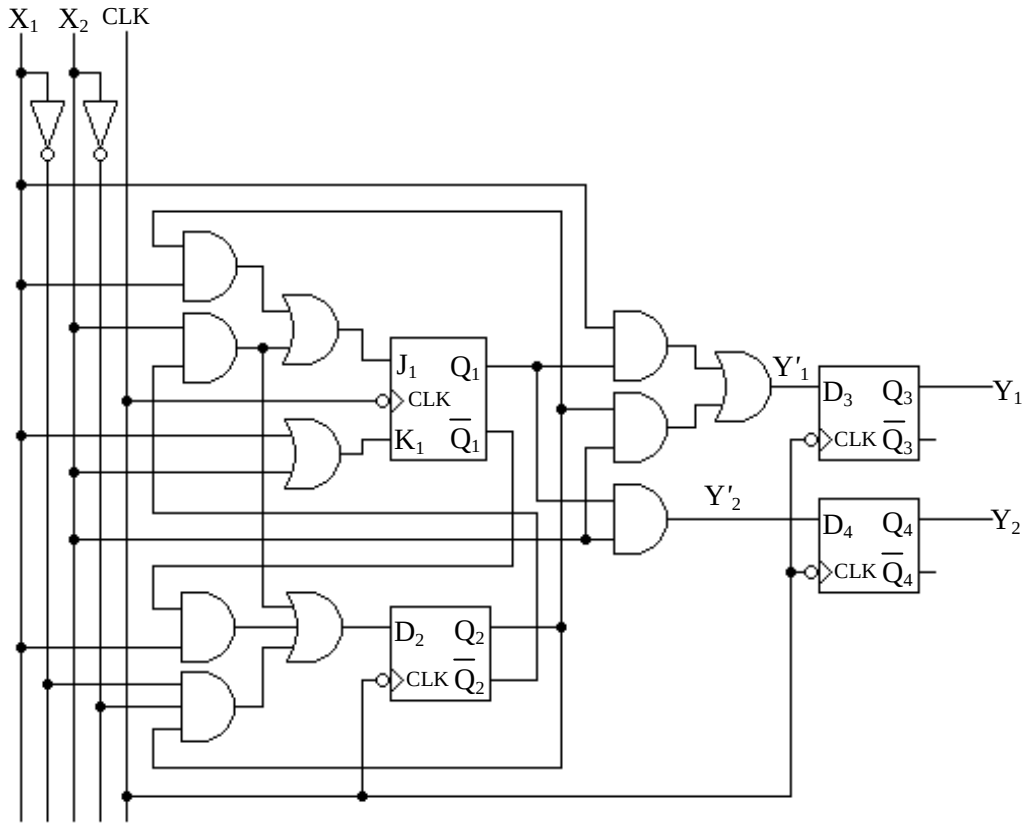
Opisani problem često se iskazuje tvrdnjom da sklopovi Mealyjevog tipa, mada mijenjaju stanja *sinhrono* (tj. u skladu sa nailaskom taktnih impulsa), reagiraju *asinhrono* na promjenu vrijednosti na ulazima. S druge strane, sklopovi Mooreovog tipa su *u potpunosti sinhroni*. Pomenuta osobina sklopova Mealyjevog tipa najčešće nije velika smetnja, ali postoje situacije u kojima može da smeta. Stoga se generalno smatra da su sklopovi Mooreovog tipa *znatno pouzdaniji*. S druge strane, sklopovi Mealyjevog tipa često su jednostavnije građe od sklopova Mooreovog tipa koji obavljaju istu funkcionalnost.

Problem asinhronog reagiranja sklopova Mealyjevog tipa na promjene vrijednosti na ulazima može se riješiti tako što na svaki od izlaza sklopa dodamo još po jedan D flip-flop, pri čemu sada izlaze iz tako modificiranog sklopa uzimamo sa izlaza dodatih flip-flopova. Na taj način, izlazi će se moći mijenjati samo pod kontrolom taktnih impulsa. Ovako modificirani sklopovi Mealyjevog tipa često se nazivaju ***sinhronizirani Mealyjevi sklopovi***, mada je ovaj naziv donekle nekonzistentan, jer je nakon ovakve modifikacije sklop zapravo postao sklop Mooreovog tipa (izlazi modificiranog sklopa jednaki su stanjima novododatih flip-flopova). Može se primijetiti da se na ovaj način ustvari vrši pretvorba Mealyjevog u Mooreov automat upravo na način koji je opisan u dokazu Teoreme 15.2 o ekvivalenciji Mealyjevih i Mooreovih automata.

Primjer 18.3:

- Formirati sinhroniziranu verziju Mealyjevog sklopa iz Primjera 18.2.

U skladu sa opisanim postupkom, potrebno je uvesti dva nova D flip-flopa, dovesti izlaze izvornog sklopa na njihove D ulaze, a njihove izlaze proglasiti za izlaze iz novog sklopa. Na taj način dobijamo sljedeću shemu:



Ovim postupkom smo zapravo uveli dva nova bita stanja Q_3 i Q_4 i definirali $Y_1 = Q_3$ i $Y_2 = Q_4$. Ako izlaze iz izvornog sklopa označimo sa Y'_1 i Y'_2, imamo $D_3 = Y'_1$ i $D_4 = Y'_2$. Kako za D flip-flop vrijedi $Q(n+1) = D$, slijedi:

$$Y_1(n+1) = Q_3(n+1) = D_3 = Y'_1(n)$$
$$Y_2(n+1) = Q_4(n+1) = D_4 = Y'_2(n)$$

Odavde imamo $Y_1(n) = Y'_1(n-1)$ i $Y_2(n) = Y'_2(n-1)$. Dakle, dobili smo sklop Mooreovog tipa (jer je $Y_1 = Q_3$ i $Y_2 = Q_4$) čiji su izlazi jednaki izlazima izvornog Mealyjevog sklopa, samo što mu izlaz kasni za jednu vremensku jedinicu u odnosu na izvorni sklop, upravo kao što predviđa Teorema 15.2.

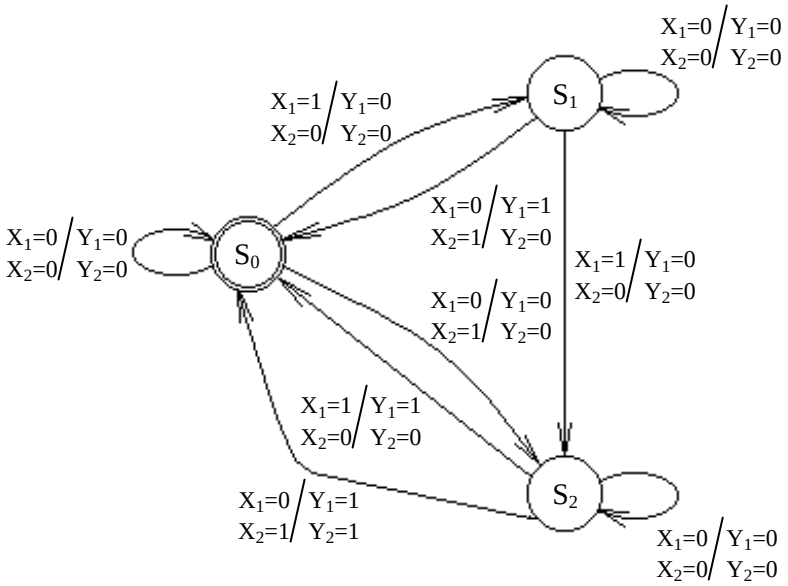
Opisani postupak pretvaranja Mealyjevog automata u ekvivalentni Mooreov automat očigledno je veoma jednostavan za realizaciju. Međutim, njegov nedostatak je što zahtijeva uvođenje m novih flip-flopova gdje je m broj izlaza, što zapravo znači uvođenje m novih bita stanja, i povećanje broja stanja 2^m puta. Na taj način graf koji opisuje novodobijeni sklop može postati neprihvatljivo glomazan. S druge strane, metod grafova omogućava i drugi način za pretvorbu Mealyjevih automata u ekvivalentne Mooreove automate koji često dovodi do mnogo jednostavnijih grafova, koji kasnije omogućavaju dizajn ekvivalentnih Mooreovih automata sa manjim brojem flip-flopova. Po ovom metodu, treba razmotriti grane početnog grafa koji ulaze u svaki od čvorova. Ukoliko svakoj grani koja ulazi u neki čvor odgovara ista kombinacija vrijednosti izlaza, tu kombinaciju vrijednosti izlaza treba pripisati posmatranom čvoru. Međutim, ukoliko različitim granama koje ulaze u neki čvor odgovaraju različite

kombinacije vrijednosti izlaza, tada taj čvor treba razbiti na onoliko drugih čvorova koliko ima različitih kombinacija vrijednosti izlaza u granama koje ulaze u taj čvor, i svaku od tih kombinacija dodijeliti odgovarajućem čvoru. Grane koje su išle u razbijeni čvor treba usmjeriti ka novoformiranim čvorovima, u skladu sa uvedenim obilježavanjem (grane koje imaju odgovarajuću kombinaciju vrijednosti izlaza treba usmjeriti u čvor kojem je ta kombinacija dodijeljena). Iz svakog od novoformiranih čvorova treba da vodi onoliko grana koliko i iz polaznog čvora, koje vode u iste čvorove kao i grane koje su išle iz početnog čvora. Postupak treba ponoviti za sve čvorove koji dopuštaju ovakvo razbijanje. Na kraju, treba ukloniti sve oznake izlaza uz grane grafa, čime je formiran graf koji odgovara automatu Mooreovog tipa. Lako je uvjeriti se da je ovako dobijen automat funkcionalno ekvivalentan polaznom Mooreovom automatu. Opisani postupak najbolje je ilustrirati na konkretnom primjeru.

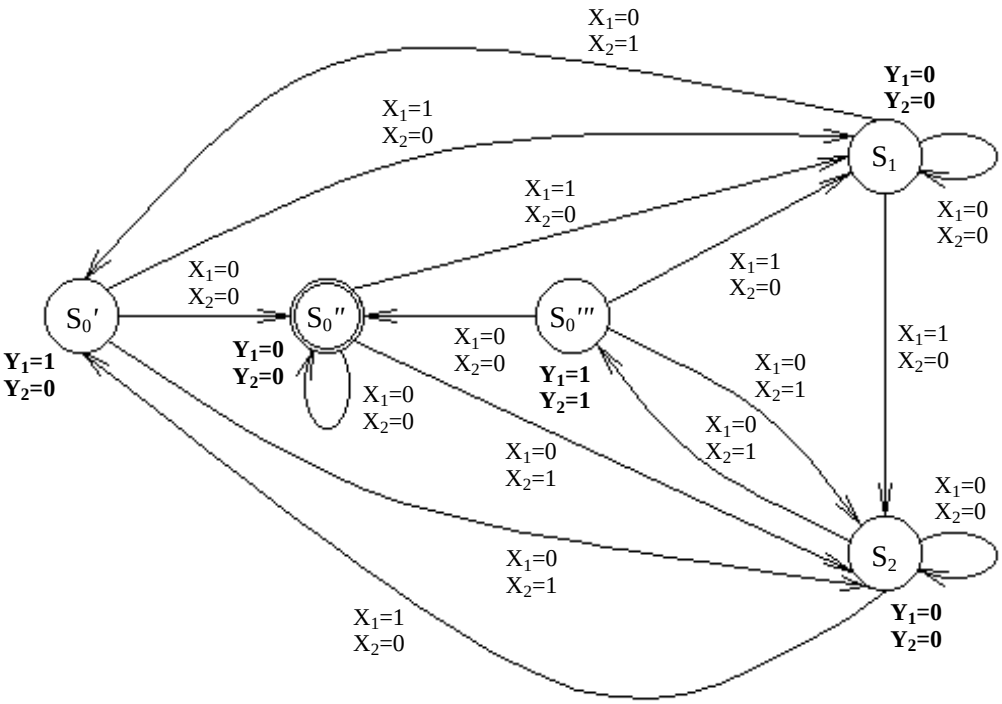
Primjer 18.4:

- Za graf Mealyjevog tipa koji odgovara automatu za Coca-Colu iz Primjera 17.3 formirati ekvivalentni graf Mooreovog tipa.

  Pođimo od grafa koji smo sastavili u Primjeru 17.3:



U ovom grafu postoje 3 čvora koja odgovaraju stanjima S_0, S_1 i S_2. Obje grane koje ulaze u čvor S_1 imaju pridružene vrijednosti izlaza $Y_1 = 0$ i $Y_2 = 0$. Stoga čvor S_1 ne treba razbijati, već mu samo treba pridružiti kombinaciju vrijednosti izlaza $Y_1 = 0$ i $Y_2 = 0$. Slično, sve tri grane koje ulaze u čvor S_2 imaju također pridružene vrijednosti izlaza $Y_1 = 0$ i $Y_2 = 0$, tako da ovu kombinaciju izlaza treba pridružiti ovom čvoru, bez njegovog razbijanja. Međutim, sa čvorom S_0 situacija je složenija. Od četiri grane koje vode u čvor S_0, dvije grane imaju pridružene vrijednosti izlaza $Y_1 = 1$ i $Y_2 = 0$, dok po jednoj grani odgovaraju kombinacije vrijednosti izlaza $Y_1 = 0$ i $Y_2 = 0$ odnosno $Y_1 = 1$ i $Y_2 = 1$. Stoga, čvor S_0 treba razbiti na tri nova čvora koja ćemo nazvati recimo S'_0, S''_0 i S'''_0. Čvoru S'_0 pridružićemo npr. kombinaciju izlaza $Y_1 = 1$ i $Y_2 = 0$, čvoru S''_0 kombinaciju $Y_1 = 0$ i $Y_2 = 0$, a čvoru S'''_0 kombinaciju $Y_1 = 1$ i $Y_2 = 1$. Dvije grane koje su vodile u čvor S_0 kojima je odgovarala kombinacija $Y_1 = 1$ i $Y_2 = 0$ preusmjerićemo u čvor S'_0. Slično, grane koje su vodile u čvor S_0 kojima su odgovarale kombinacije $Y_1 = 0$ i $Y_2 = 0$ odnosno $Y_1 = 1$ i $Y_2 = 1$ preusmjerićemo respektivno u čvorove S''_0 i S'''_0. Konačno, kako su iz čvora S_0 vodile tri grane, moramo dopisati isti broj odgovarajućih grana iz svakog od čvorova S'_0, S''_0 i S'''_0. Dakle, iz svakog od ova tri čvora mora postojati grana koja za $X_1 = 1$ i $X_2 = 0$ vodi u čvor S_1, a za $X_1 = 0$ i $X_2 = 1$ u čvor S_2. Na kraju, trebamo izbrisati sve oznake izlaza koje su bile pridružene granama, i dobijamo traženi graf. Može se postaviti pitanje *koje je početno stanje* novog grafa pošto je početno stanje S_0 prvobitnog grafa razbijeno. Lako je zaključiti da je to stanje S''_0 kojem odgovara kombinacija $Y_1 = 0$ i $Y_2 = 0$, jer upravo takva kombinacija treba da vrijedi na početku rada sklopa. Konačno, dobijamo graf sa sljedeće slike:

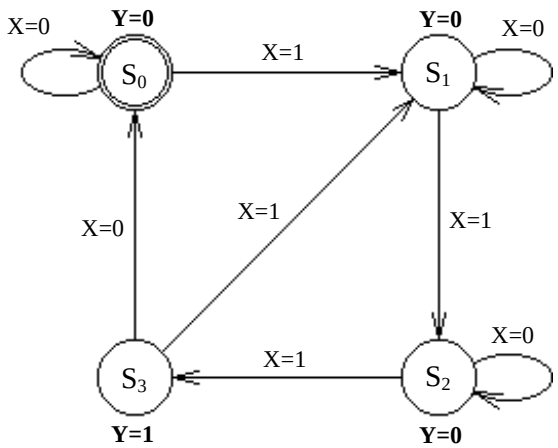


Dobijeni graf ima 5 stanja. Kako se 5 stanja mogu kodirati pomoću 3 bita stanja, na osnovu ovog grafa moguće je sastaviti Mooreov automat ekvivalentan Mealyjevom automatu iz Primjera 18.2, a koji će imati jedan flip-flop manje u odnosu na sinhronizirani Mealyjev automat realiziran u Primjeru 18.3.

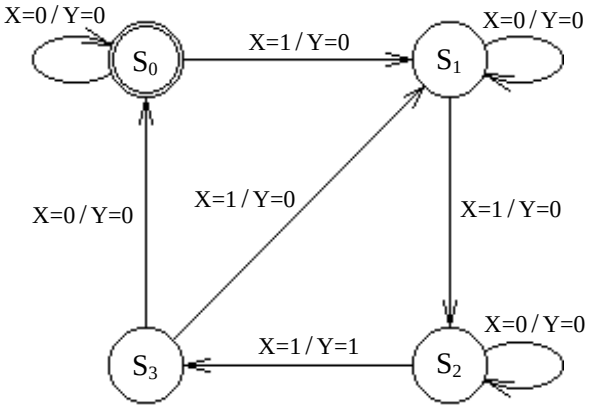
Metod grafova omogućava i veoma jednostavnu pretvorbu Mooreovih automata u ekvivalentne Mealyjeve automate. Ova pretvorba nije toliko često potrebna, s obzirom da se Mooreovi automati smatraju pouzdanijim. Međutim, veoma često je građa Mealyjevih automata jednostavnija od građe Mooreovih automata, tako da u slučaju kada je jednostavnost građe važna vrijedi razmotriti i model Mealyjevog tipa. Pretvorba Mooreovog automata u Mealyjev izvodi se pomoću grafova tako što se prosto vrijednosti izlaza koje su pridružene nekom čvoru pridruže svim granama koje ulaze u taj čvor. Na taj način dobijamo graf Mealyjevog tipa, koji ima isti broj grana i čvorova kao i polazni graf Mooreovog tipa, pa samim tim dobijeni Mooreov automat ima jednak broj stanja kao i polazni Mealyjev automat. Međutim, veoma često se nakon obavljene pretvorbe pokaže da su neka od stanja u novodobijenom grafu suvišna ili međusobno ekvivalentna, tako da se mogu eliminirati, čime se dobija graf sa manje stanja u odnosu na polazni graf, kojem može odgovarati jednostavniji sklop.

Primjer 18.5:

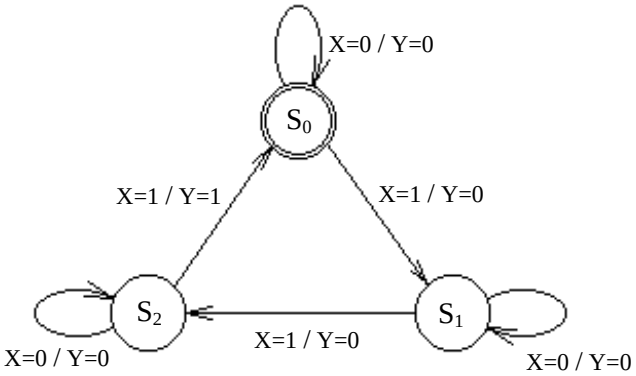
- Za graf sa slike koji odgovara automatu Mooreovog tipa naći ekvivalentni graf Mealyjevog tipa, i razmotriti eventualnu mogućnost njegovog pojednostavljenja.



Ukoliko sve vrijednosti izlaza koje su pridružene čvorovima pridružimo granama koje ulaze u pripadni čvor, neposredno dobijamo sljedeći ekvivaletni graf Mealyjevog tipa:



Međutim, pažljivijom analizom može se primijetiti da se ovaj graf može pojednostaviti. Naime, razmotrimo stanja S_0 i S_3. Za oba navedena stanja vrijedi da iz njih automat pod dejstvom ulaza $X = 0$ prelazi u stanje S_0 uz emitiranje izlaza $Y = 0$, dok pod dejstvom ulaza $X = 1$ prelazi u stanje S_1 uz emitiranje izlaza $Y = 0$. Prema tome, ponašanje automata u stanjima S_0 i S_3 je *apsolutno identično*, odnosno između ova dva stanja nema nikakve razlike. Prema tome, jedno od ova dva stanja (npr. S_3) moguće je ukloniti, i sve grane koje su išle u čvor S_3 preusmjeriti u čvor koji odgovara njemu ekvivalentnom stanju S_0. Na taj način dobijamo graf Mealyjevog tipa sa jednim stanjem manje u odnosu na polazni graf, prikazan na sljedećoj slici:



Dobijeni graf je zapravo graf koji smo razmatrali u Primjeru 17.5. Čitateljima i čitateljkama se ostavlja da pokažu da će se ranije opisanom pretvorbom grafova Mealyjevog tipa u grafove Mooreovog tipa ponovo dobiti graf koji odgovara polaznom grafu (samo uz eventualno drugačije imenovana stanja).

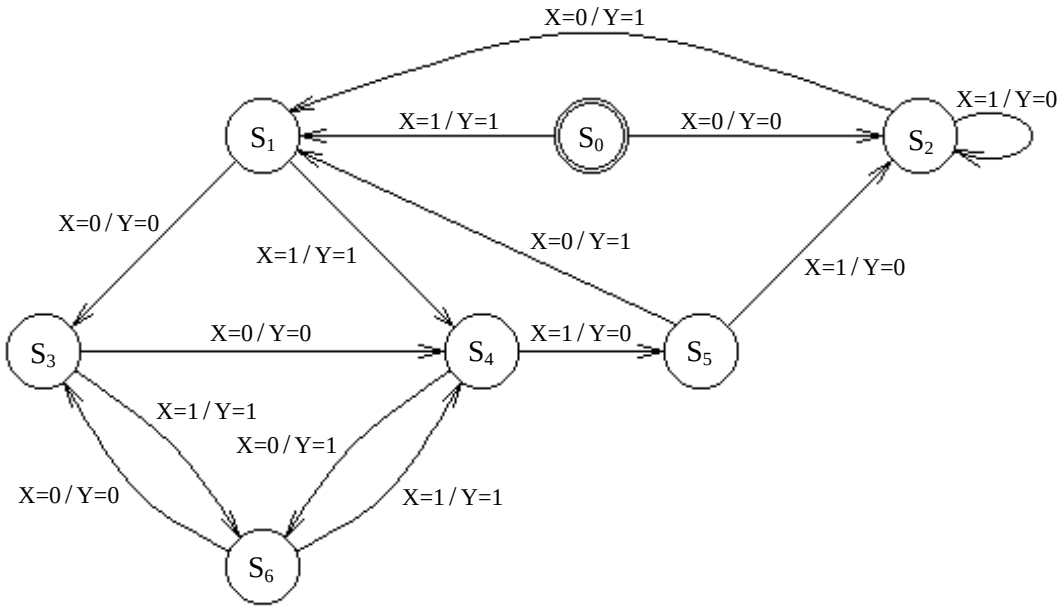
Posljednji primjer jasno pokazuje da se može desiti da postoje stanja koja su suvišna u smislu da su ekvivalentna nekom od postojećih stanja. Nekada je takva suvišna stanja posve lako uočiti, kao npr. u prethodnom primjeru, u kojem je očigledno da se automat u stanjima S_0 i S_3 ponaša posve identično. Međutim, često nije lako samo posmatranjem uočiti da su neka stanja ekvivalentna. Pretpostavimo, na primjer, da neki automat iz stanja S_2 pod dejstvom ulaza $X = 0$ prelazi u stanje S_5 uz emitiranje izlaza $Y = 1$, a pod dejstvom ulaza $X = 1$ u stanje S_0 uz emitiranje izlaza $Y = 0$. Pretpostavimo dalje da automat iz stanja S_3 pod dejstvom ulaza $X = 0$ prelazi u stanje S_1 uz emitiranje izlaza $Y = 1$, a pod dejstvom ulaza $X = 1$ u stanje S_4 uz emitiranje izlaza $Y = 0$. Iz ovoga što je rečeno, ne izgleda da su stanja S_2 i S_3 ekvivalentna. Zaista, ona to ne moraju biti. Međutim, ukoliko bi se pokazalo da su stanja S_1 i S_5 ekvivalentna, kao da su i stanja S_0 i S_4 ekvivalentna, kao posljedicu bismo očigledno imali i da su stanja S_2 i S_3 ekvivalentna! Dakle, između dva ili više stanja mogu postojati prilično složene ovisnosti koje za posljedicu mogu imati njihovu ekvivalentnost, koja ne mora biti očigledna na prvi pogled. Stoga su razvijene brojne metode za detekciju i uklanjanje suvišnih stanja u grafovima. Ove metode su značajne zbog toga što se često dešava da graf koji se dobije prilikom pretvaranja verbalnog opisa rada

sekvencijalnog sklopa u opis dat grafom sadrži više stanja nego što bi zaista trebao da ima, zbog toga što projektant previdi da su neka stanja bila nepotrebna ili ekvivalentna postojećim stanjima. Kako jednostavniji graf vodi jednostavnijem sklopu, prije nego što se krene u realizaciju sklopa korisno je razmotriti može li se dobijeni graf prethodno pojednostaviti.

Jedna od najboljih metoda za detekciju ekvivalentnih stanja sastoji se u sljedećem. Posve je jasno da stanja mogu biti ekvivalentna jedino ukoliko pod dejstvom istih vrijednosti na ulazima emitiraju iste vrijednosti na izlazima. Stoga je prvi korak razvrstavanje svih stanja u grupe, pri čemu svaku grupu čine ona stanja koja pod dejstvom istih vrijednosti na ulazima emitiraju iste vrijednosti na izlazima (stanja u istoj grupi predstavljaju potencijalne kandidate za međusobno ekvivalentna stanja). Nakon toga se svaka od grupa dalje dijeli na nove grupe (podgrupe), pri čemu svaku podgrupu sačinjavaju ona stanja koja pod dejstvom istih vrijednosti na ulazima prelaze u stanja iz istih grupa. Postupak se ponavlja sve dok je moguća dalja podjela grupa na podgrupe. Kada dođemo do situacije da dalja podjela na podgrupe nije moguća, tada možemo zaključiti da su sva stanja koja pripadaju istoj grupi ekvivalentna. U ekstremnom slučaju, svaka grupa će se sastojati od po samo jednog stanja, što bi bila indikacija da nema suvišnih stanja. Kao i sve formalne postupke, i ovaj postupak je najlakše shvatiti kroz konkretan primjer:

Primjer 18.6:

- Ispitati da li u grafu sa sljedeće slike postoje suvišna stanja, i ukoliko postoje, ukloniti ih.



U ovom grafu očigledno možemo uočiti dvije grupe stanja. Prva grupa (označimo je sa G_1) pod dejstvom ulaza $X = 0$ emitira izlaz $Y = 0$ a po dejstvom ulaza $X = 1$ izlaz $Y = 1$, i njoj pripadaju stanja S_0, S_1, S_3 i S_6. Druga grupa (označimo je sa G_2) pod dejstvom ulaza $X = 0$ emitira izlaz $Y = 1$ a po dejstvom ulaza $X = 1$ izlaz $Y = 0$, i njoj pripadaju stanja S_2, S_4 i S_5. Ovo možemo simbolički predstaviti na sljedeći način:

G_1:				G_2:		
S_0	S_1	S_3	S_6	S_2	S_4	S_5

Razmotrimo sada kako se odvijaju prelasci iz jednog stanja u drugo za svako od stanja. Iz stanja S_0 pod dejstvom ulaza $X = 0$ automat prelazi u stanje S_2, koje pripada grupi G_2, dok pod dejstvom ulaza $X = 1$ prelazi u stanje S_1 koje pripada grupi G_1. Ovu činjenicu skraćeno ćemo pisati tako što ćemo stanju S_0 pridružiti par (G_2,G_1). Iz stanja S_1 pod dejstvom ulaza $X = 0$ automat prelazi u stanje S_3, koje pripada grupi G_1, dok pod dejstvom ulaza $X = 1$ prelazi u stanje S_4 koje pripada grupi G_2. Ovu činjenicu ćemo

pisati kao par (G_1,G_2) pridružen stanju S_1. Na sličan način, stanjima S_2, S_3, S_4, S_5 i S_6 pridružujemo redom parove (G_1,G_2), (G_2,G_1), (G_1,G_2), (G_1,G_2) i (G_1,G_2). Ovo možemo simbolički prikazati na sljedeći način:

G_1:				G_2:		
S_0	S_1	S_3	S_6	S_2	S_4	S_5
(G_2,G_1)	(G_1,G_2)	(G_2,G_1)	(G_1,G_2)	(G_1,G_2)	(G_1,G_2)	(G_1,G_2)

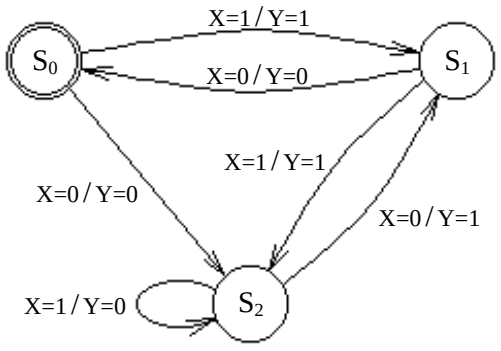
Grupu G_1 sada razbijamo na dvije podgrupe, recimo G_{11} i G_{12}. Jednu podgrupu (recimo G_{11}) čine stanja S_0 i S_3 kojima je pridružen par (G_2,G_1), dok drugu podgrupu (G_{12}) čine stanja S_1 i S_6 kojima je pridružen par (G_1,G_2). Grupu G_2 ne možemo dijeliti, s obzirom da je svim stanjima u grupi G_2 pridružen isti par (G_1,G_2). Novonastalu podjelu ćemo simbolički prikazati na sljedeći način:

G_{11}:		G_{12}:		G_2:		
S_0	S_3	S_1	S_6	S_2	S_4	S_5

Sada ponovo trebamo razmotriti prelaske iz stanja u stanje. Iz stanja S_0 pod dejstvom ulaza $X=0$ automat prelazi u stanje S_2, koje pripada grupi G_2, dok pod dejstvom ulaza $X=1$ prelazi u stanje S_1 koje pripada grupi G_{12} (prema novoj podjeli), tako da stanju S_0 pridružujemo par (G_2,G_{12}). Na sličan način, stanjima S_1, S_2, S_3, S_4, S_5 i S_6 pridružujemo redom parove (G_{12},G_2), (G_{12},G_2), (G_2,G_{12}), (G_{12},G_2), (G_{12},G_2) i (G_{12},G_2). Simbolički ovo pridruživanje možemo prikazati na sljedeći način:

G_{11}:		G_{12}:		G_2:		
S_0	S_3	S_1	S_6	S_2	S_4	S_5
(G_2,G_{12})	(G_2,G_{12})	(G_{12},G_2)	(G_{12},G_2)	(G_{12},G_2)	(G_{12},G_2)	(G_{12},G_2)

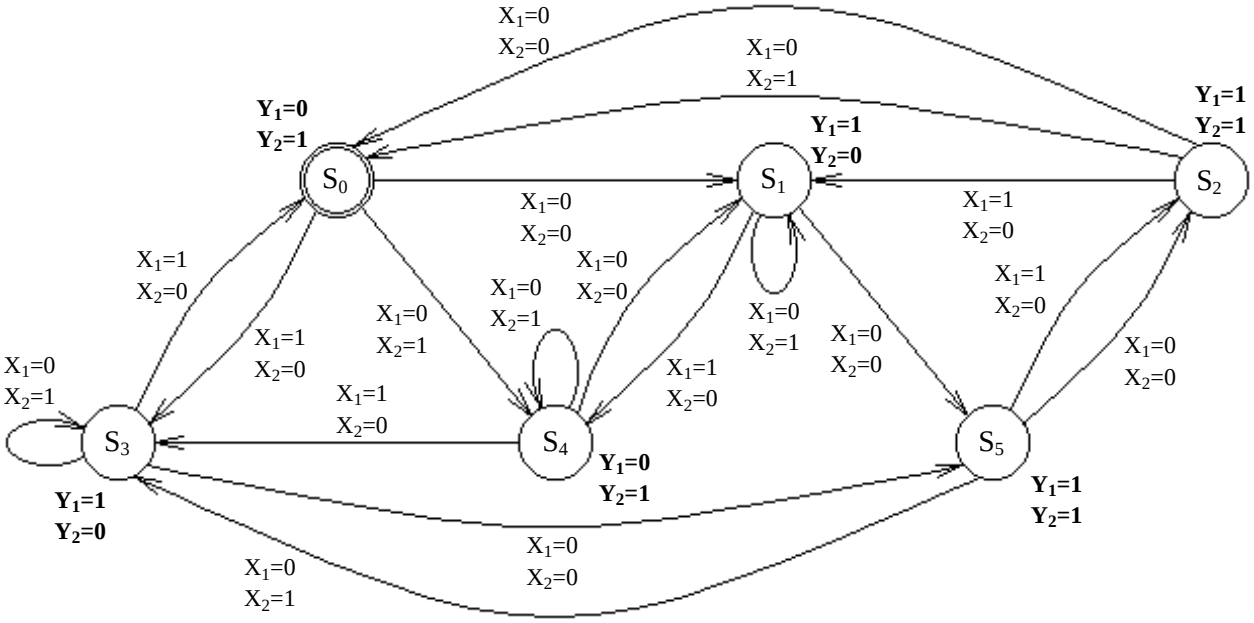
U ovom trenutku dalja podjela na podgrupe više nije moguća, jer su svim stanjima unutar svake od grupa pridruženi isti parovi. Stoga su sva stanja koja se nalaze unutar svake od grupa G_{11}, G_{12} i G_2 ekvivalentna, odnosno vrijedi $S_0 \cong S_3$, $S_1 \cong S_6$, $S_2 \cong S_4 \cong S_5$. Stoga iz svake skupine ekvivalentnih stanja možemo zadržati samo jedno, a ostala izbaciti i sve grane koje su vodile u izbačene čvorove usmjeriti u ekvivalentni čvor koji je zadržan. Na primjer, iz skupine ekvivalentnih stanja S_0, S_3 zadržaćemo stanje S_0, iz skupine S_1, S_6 stanje S_1, a iz skupine S_2, S_4 i S_5 stanje S_2. Čvorove koje odgovaraju izbačenim stanjima ćemo ukloniti (zajedno sa granama koje izlaze iz njih), a sve grane koje su vodile u uklonjene čvorove usmjerićemo u njima ekvivalentne čvorove (na primjer, grane koje su vodile u čvor S_5 preusmjerićemo u čvor S_2). Na taj način, kao rezultat obavljene transformacije dobijamo sljedeći graf ekvivalentan polaznom grafu, koji je očito znatno jednostavniji:



U ovom primjeru izvršena je optimizacija grafa koji je bio Mealyjevog tipa. Na potpuno identičan način mogu se optimizirati i grafovi Mooreovog tipa, samo što se početno razvrstavanje stanja u grupe vrši neposredno na osnovu vrijednosti izlaza pridruženih čvorovima (stanja kojima su pridruženi isti izlazi svrstavaju se u istu grupu). Nakon toga, postupak teče na isti način kao za slučaj grafova Mealyjevog tipa.

Primjer 18.7:

- Optimizirati graf stanja sa sljedeće slike, ukoliko je to moguće.



Stanja ćemo razvrstati u tri grupe, prema pridruženim vrijednostima izlaza. U prvu grupu (G_1) razvrstaćemo stanja S_0 i S_4 kojima su pridružene vrijednosti izlaza $Y_1 = 0$ i $Y_2 = 1$, u drugu grupu (G_2) razvrstaćemo stanja S_1 i S_3 kojima su pridružene vrijednosti izlaza $Y_1 = 1$ i $Y_2 = 0$, dok ćemo u treću grupu (G_3) razvrstati stanja S_2 i S_5 kojima su pridružene vrijednosti izlaza $Y_1 = 1$ i $Y_2 = 1$:

G_1:		G_2:		G_3:	
S_0	S_4	S_1	S_3	S_2	S_5

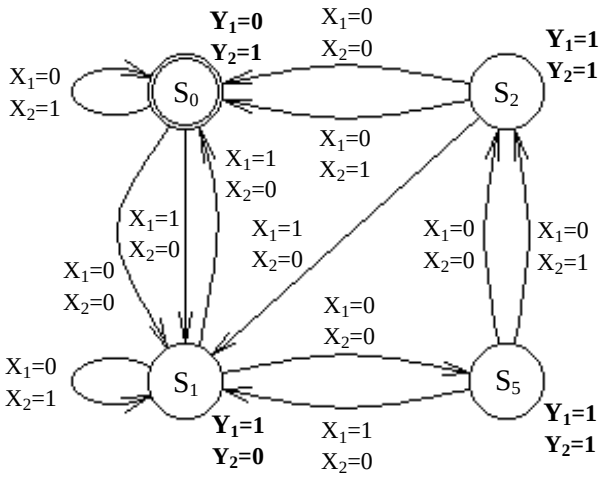
Postupak se dalje nastavlja na isti način kao u prethodnom primjeru, analizom kako se odvijaju prelasci iz jednog stanja u drugo za svako od stanja, i za svaku moguću kombinaciju vrijednosti na ulazima. Ako bolje razmotrimo graf, vidjećemo da se kombinacija vrijednosti ulaza $X_1 = X_2 = 1$ nigdje ne javlja, tako da je potrebno razmotriti tri kombinacije: $X_1 = X_2 = 0$; $X_1 = 0$ i $X_2 = 1$; $X_1 = 1$ i $X_2 = 0$. Iz stanja S_0 automat pod dejstvom ove tri kombinacije prelazi redom u stanja S_1, S_4 i S_3, koja pripadaju redom grupama G_2, G_1 i G_2. Stoga ćemo stanju S_0 pridružiti trojku (G_2,G_1,G_2). Na sličan način, stanjima S_1, S_2, S_3, S_4 i S_5 pridružujemo redom trojke (G_3,G_2,G_1), (G_1,G_1,G_2), (G_3,G_2,G_1), (G_2,G_1,G_2) i (G_3,G_2,G_3), što možemo simbolički prikazati ovako:

G_1:		G_2:		G_3:	
S_0	S_4	S_1	S_3	S_2	S_5
(G_2,G_1,G_2)	(G_2,G_1,G_2)	(G_3,G_2,G_1)	(G_3,G_2,G_1)	(G_1,G_1,G_2)	(G_3,G_2,G_3)

Nakon prvog koraka, vidimo da se grupa G_3 može razbiti na dvije podgrupe (npr. G_{31} i G_{32}). U sljedećem koraku, proces razvrstavanja teče kao što je prikazano sljedećom shemom:

G_1:		G_2:		G_{31}:	G_{32}:
S_0	S_4	S_1	S_3	S_2	S_5
(G_2,G_1,G_2)	(G_2,G_1,G_2)	(G_{32},G_2,G_1)	(G_{32},G_2,G_1)	(G_1,G_1,G_2)	(G_{31},G_2,G_{31})

Dalje razvrstavanje očigledno više nije moguće, odakle slijedi da su stanja S_0 i S_4 odnosno S_1 i S_3 ekvivalentna. Zadržimo li stanja S_0 i S_1 a odbacimo S_3 i S_4 kao suvišna, i grane koje vode u njih preusmjerimo u njima ekvivalentna stanja, dobijamo sljedeći optimizirani graf:



Primjer 18.8:

- Pokazati da se graf Mooreovog automata iz Primjera 18.5 ne može optimizirati, dok se graf koji se dobije nakon pretvorbe u ekvivalentni Mealyjev automat može optimizirati (pri tome se dobija isto rješenje koje smo već dobili primjenom intuitivne optimizacije).

U polaznom grafu u grupu G_1 možemo razvrstati stanja S_0, S_1 i S_2 kojima je pridružena vrijednost izlaza Y = 0, dok stanje S_3 kojem je pridružena vrijednost izlaza Y = 0 razvrstavamo u grupu G_2. Dalji tok razvrstavanja odvija se na sljedeći način:

G_1:			G_2:
S_0	S_1	S_2	S_3
(G_1,G_1)	(G_1,G_1)	(G_1,G_2)	(G_1,G_1)

G_{11}:		G_{12}	G_2:
S_0	S_1	S_2	S_3
(G_1,G_1)	(G_1,G_{12})	$(G_{12}.G_2)$	(G_{11},G_{11})

G_{111}:	G_{112}:	G_{12}	G_2
S_0	S_1	S_2	S_3
(G_1,G_{112})	(G_{112},G_{12})	(G_{12},G_2)	(G_{111},G_{112})

Vidimo da je na kraju svako stanje razvrstano u zasebnu grupu. Prema tome, ekvivaletnih stanja nema, i polazni graf se ne može optimizirati. Razmotrimo sada ekvivalentni graf Mealyjevog tipa. U grupu G_1 možemo razvrstati stanja S_0, S_1 i S_3, jer se u njima i za ulaz X = 0 i za ulaz X = 1 emitira izlaz Y = 0, dok stanje S_2 u kojem se za ulaz X = 0 emitira izlaz Y = 0 a za ulaz X = 1 izlaz Y = 1 razvrstavamo u grupu G_2. Dalji tok razvrstavanja odvija se na sljedeći način:

G_1:			G_2:
S_0	S_1	S_3	S_2
(G_1,G_1)	(G_1,G_2)	(G_1,G_1)	(G_2,G_1)

G_{11}:		G_{12}	G_2:
S_0	S_3	S_1	S_2
(G_1,G_{12})	(G_1,G_{12})	(G_{12},G_2)	(G_2,G_{11})

Ovim je razvrstavanje završeno, jer dalja podjela na grupe nije moguća. Nakon obavljenog razvrstavanja vidimo da su stanja S_0 i S_3 ekvivalentna, što je isti zaključak koji smo i ranije izveli prostim logičkim rasuđivanjem.

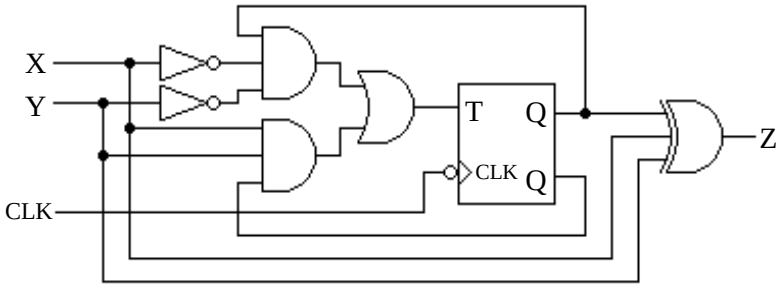
(?) Pitanja i zadaci

18.1 Objasnite šta se podrazumijeva pod analizom sekvencijalnih sklopova.

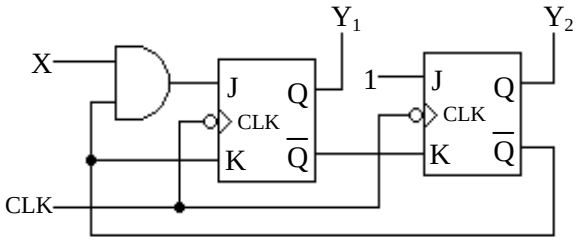
18.2 Objasnite zbog čega činjenica da je neki sekvencijalni sklop sinhronog tipa drastično pojednostavljuje postupak njegove analize.

18.3 Objasnite šta su nedostižna a šta prolazna stanja sekvencijalnog sklopa.

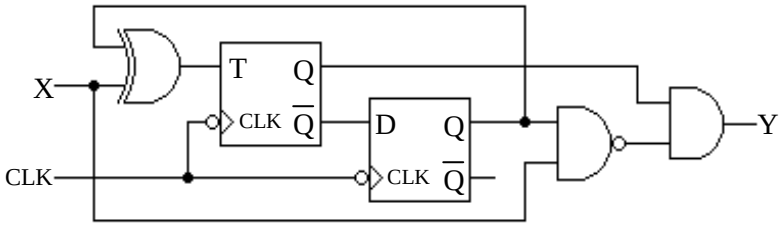
18.4 Analizirajte sekvencijalni sklop sa sljedeće slike, odnosno utvrdite graf koji opisuje njegov radi. Možete li na osnovu dobijenog grafa zaključiti čemu služi ovaj sekvencijalni sklop?



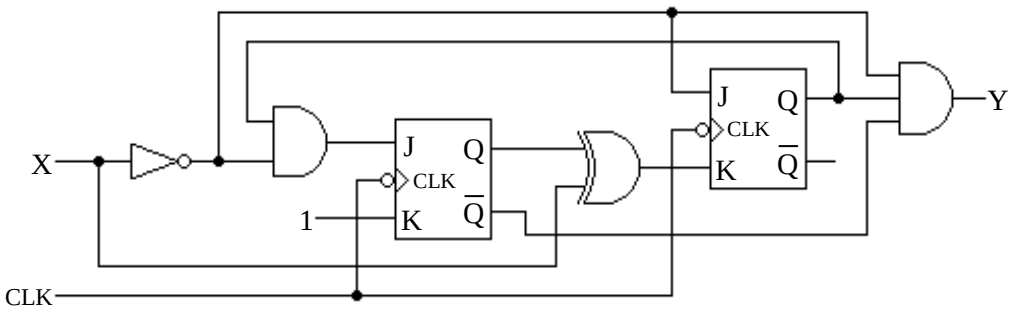
18.5 Analizirajte sekvencijalni sklop sa sljedeće slike:



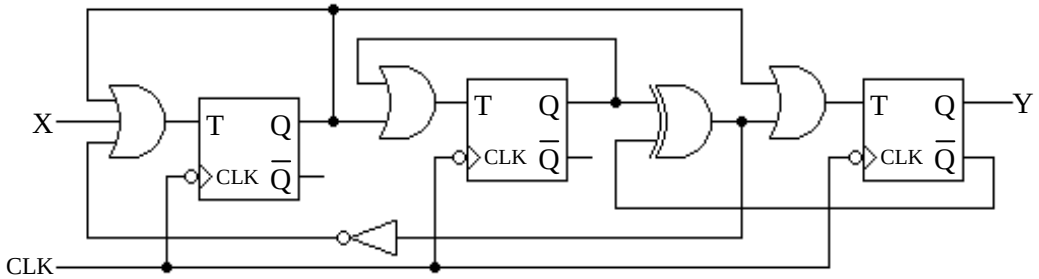
18.6 Analizirajte sekvencijalni sklop sa sljedeće slike:



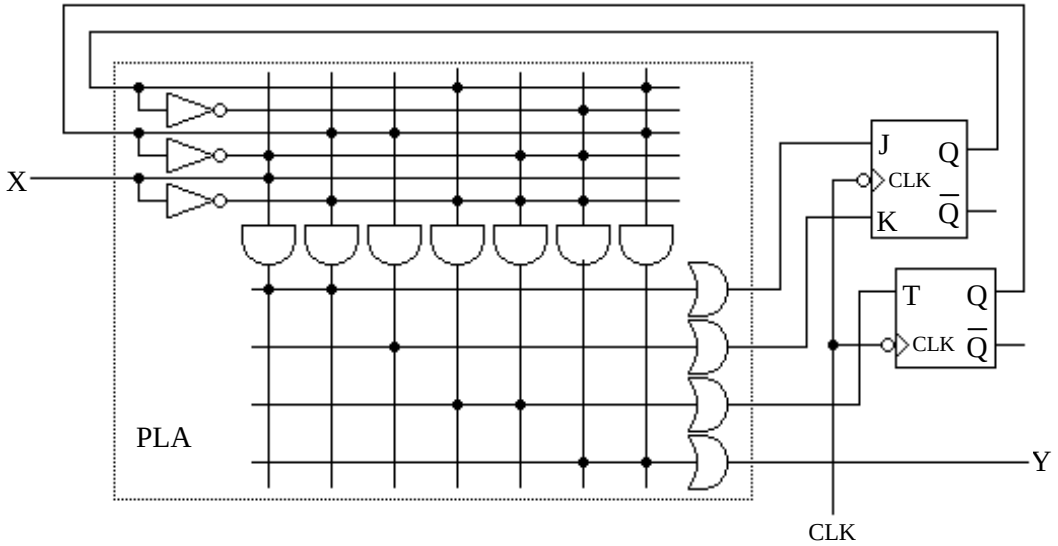
18.7 Analizirajte sekvencijalni sklop sa sljedeće slike:



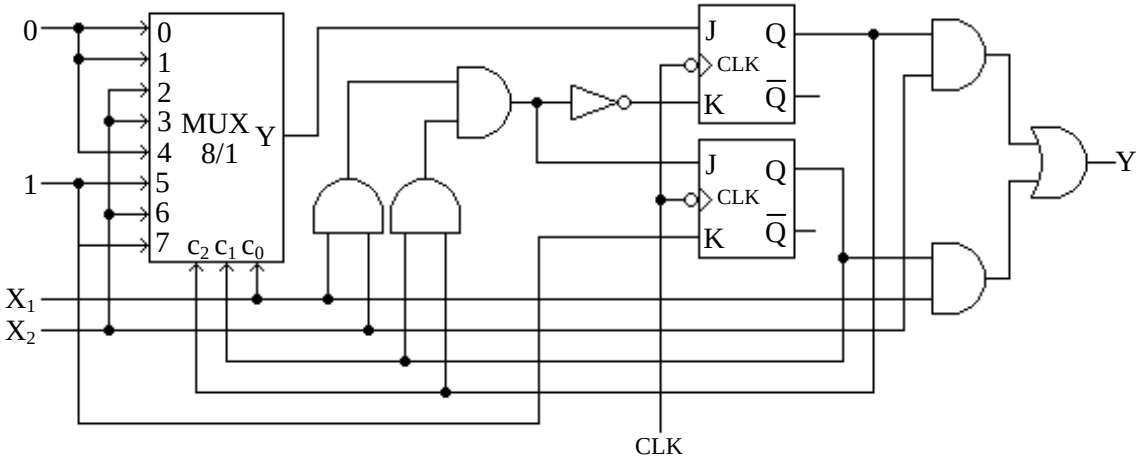
18.8 Analizirajte sekvencijalni sklop sa sljedeće slike:



18.9 Analizirajte sekvencijalni sklop sa sljedeće slike, u kojem su logičke funkcije neophodne za funkcionalnost sklopa realizirane uz pomoć PLA komponente:



18.10 Analizirajte sekvencijalni sklop sa sljedeće slike, u kojem su logičke funkcije neophodne za funkcionalnost sklopa realizirane uz pomoć multipleksera i dodatnih logičkih kola:



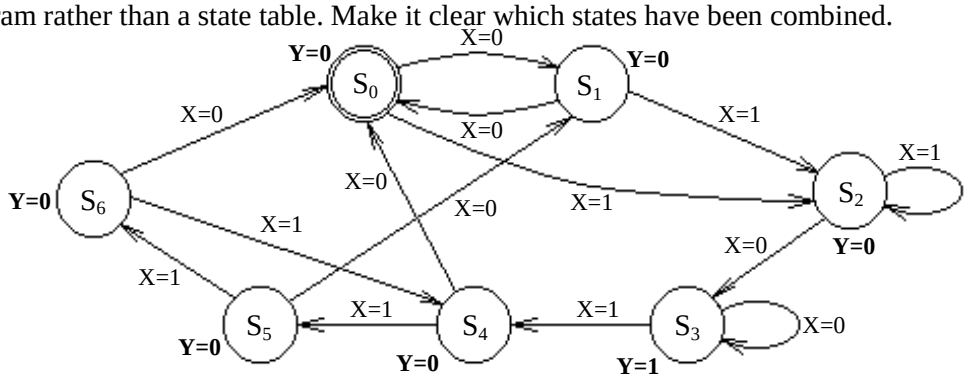
18.11 Objasnite zbog čega sekvencijalni sklopovi Mealyjevog tipa narušavaju definiciju digitalnih sklopova.

18.12 Objasnite šta su sinhronizirani Mealyjevi sklopovi i zbog čega se koriste. Da li se zaista radi o sklopovima Mealyjevog tipa?

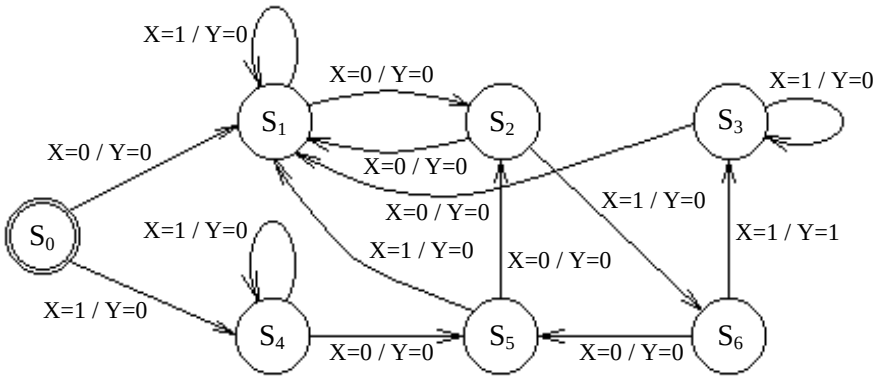
18.13 Pretvorite ser. sum.

- Use the implication chart method to reduce the 4-bit string recognizer state diagram of Figure 9.2. (The output is asserted after each 4-bit input sequence if it consists of one of the binary strings 0110 or 1010. The machine returns to the reset state after each 4-bit sequence.).

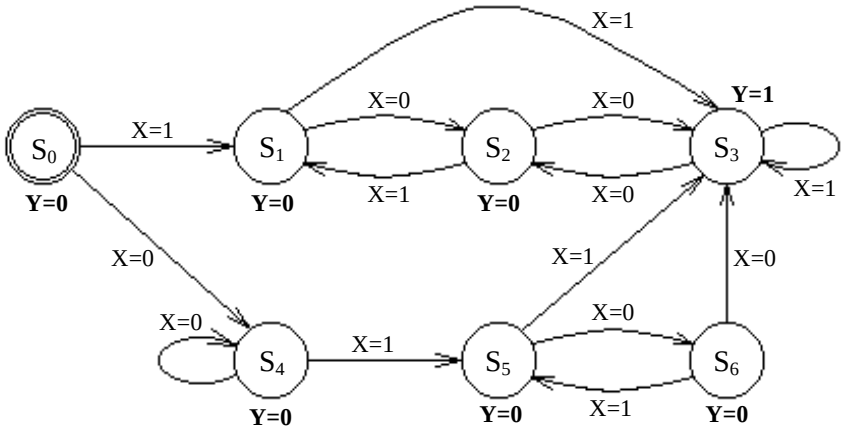
- Given the state diagram in Figure Ex9.2, obtain an equivalent reduced state diagram containing a minimum number of states.
  You may use row matching or implication charts. Put your final answer in the form of a state diagram rather than a state table. Make it clear which states have been combined.



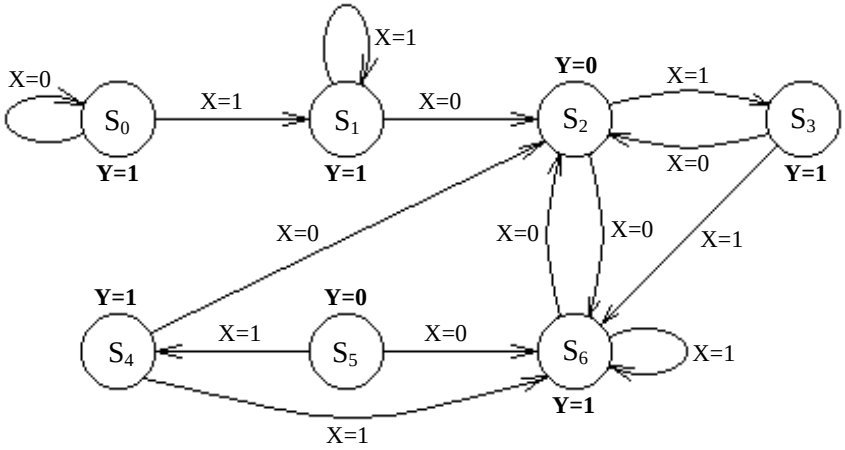
- Given the state diagram in Figure Ex9.3, determine which states should be combined to determine the reduced state diagram. You may use row matching or implication charts.



- Given the state diagram in Figure Ex9.4, draw the fully reduced state diagram. State succinctly what strings cause the recognizer to output a 1.



- (State Reduction) Starting with the state diagram of Figure Ex9.5, use the implication chart method to find the minimum state diagram.
  Which of the original states are combined?



SIGURNO:
6.11
6.19
6.24
6.31

SUMNJIVO:

6.22c
6.26
6.27
6.32
7.16 KNF
7.27
8.14ab
8.15
8.18ab
10.13

Dokazati 0'=1 i 1'=0 preko aksioma
Formule 57-60 iz Dzekijeve skripte
Vise zadataka oko Postove teoreme, formule sa NAND i NOR iz Dzekija
Izgleda da se 2 aksioma booleove algebre mogu izbaciti
U P11: Uvjeti za formiranje parova, cetvorki itd nisu toliko
  jednostavni koliko je prvobitno bilo napisano
U P11 bolje objasniti XOR realizacije
U P11 bolje objasniti sisteme funkcija
Vidjeti da li sumator na 2KK daje ispravno izlazni prenos!
Dodati pokoji zadatak tipa dekoder+prateca logika (dzeki)

15.5

15.8

15.11

$$Y(n) = \alpha\,(\beta+1)\,Y(n-1) - \alpha\,\beta\,Y(n-2) + c$$

$$Y(0) = \alpha\,(\beta+1)\,Y(-1) - \alpha\,\beta\,Y(-2) + c = c$$
$$Y(1) = \alpha\,(\beta+1)\,Y(0) - \alpha\,\beta\,Y(-1) + c = \alpha\,(\beta+1)\,c + c = (\alpha\,\beta + \alpha + 1)\,c$$
$$Y(2) = \alpha\,(\beta+1)\,Y(1) - \alpha\,\beta\,Y(0) + c = (\alpha\,\beta + \alpha)\,(\alpha\,\beta + \alpha + 1)\,c - \alpha\,\beta\,c + c$$
$$Y(3) = \alpha\,(\beta+1)\,Y(2) - \alpha\,\beta\,Y(1) + c$$
$$Y(4) = \alpha\,(\beta+1)\,Y(3) - \alpha\,\beta\,Y(2) + c$$

15.13

15.14

15.20

15.21

15.22

15.23 $y_j(n) = F_j[x_1(n), x_2(n), \ldots x_N(n), x_1(n-1), x_2(n-1), \ldots x_N(n-1), x_1(n-2), x_2(n-2), \ldots x_N(n-2), \ldots]$

$$S(n) = \left[\frac{x_1(n-1)}{2} + \frac{x_2(n-1)}{2^2} + \ldots + \frac{x_N(n-1)}{2^N}\right] + \left[\frac{x_1(n-2)}{2^{N+1}} + \frac{x_2(n-2)}{2^{N+2}} + \ldots + \frac{x_N(n-2)}{2^{2N}}\right] + \ldots =$$
$$= \sum_{i=1}^{N} \frac{x_i(n-1)}{2^i} + \sum_{i=1}^{N} \frac{x_i(n-2)}{2^{N+i}} + \sum_{i=1}^{N} \frac{x_i(n-3)}{2^{2N+i}} + \ldots = \sum_{k=1}^{\infty} \sum_{i=1}^{N} \frac{x_i(n-k)}{2^{N(k-1)+i}}$$

$$S(n+1) = \sum_{k=1}^{\infty} \sum_{i=1}^{N} \frac{x_i(n+1-k)}{2^{N(k-1)+i}} = \sum_{k=0}^{\infty} \sum_{i=1}^{N} \frac{x_i(n-k)}{2^{Nk+i}} = \sum_{i=1}^{N} \frac{x_i(n)}{2^i} + \frac{1}{2^N} \sum_{k=1}^{\infty} \sum_{i=1}^{N} \frac{x_i(n-k)}{2^{N(k-1)+i}}$$

odnosno

$$S(n+1) = \frac{1}{2^N} S(n) + \sum_{i=1}^{N} \frac{x_i(n)}{2^i}$$

Dalje,

$$x_i(n-k) = \lfloor 2^{N(k-1)+i}\, S(n) \rfloor - 2 \cdot \lfloor 2^{N(k-1)+i-1}\, S(n) \rfloor$$

$$\Phi[x_1(n), x_2(n), \ldots x_N(n), S(n)] = \frac{1}{2^N} S(n) + \sum_{i=1}^{N} \frac{x_i(n)}{2^i}$$

$$\Psi_j[x_1(n), x_2(n), \ldots x_N(n), S(n)] = F_j[x_1(n), x_2(n), \ldots x_N(n), \lfloor 2\,S(n) \rfloor, \lfloor 2^2\,S(n) \rfloor - 2 \cdot \lfloor 2\,S(n) \rfloor,$$
$$\lfloor 2^3\,S(n) \rfloor - 2 \cdot \lfloor 2^2\,S(n) \rfloor, \lfloor 2^4\,S(n) \rfloor - 2 \cdot \lfloor 2^3\,S(n) \rfloor, \ldots]$$

$$S(n+1) = \Phi[x_1(n), x_2(n), \ldots x_N(n), S(n)]$$
$$y_j = \Psi_j[x_1(n), x_2(n), \ldots x_N(n), S(n)]$$

15.26

15.27

15.28

15.29

15.30

15.31

15.32

15.33

15.34

15.35

15.36

15.37

15.38

15.40

15.41

15.43

15.48

15.49

15.50

Upute i rješenja

1.7 Barem 6800 uzoraka u sekundi.

3.4 6 bita.

3.5 $v = 1/(100000 - 10000) = 1/90000$, $i = -\log_2 v \approx 16{,}458$ bita. Slijedi da je 17 pitanja ponuđenog tipa posve dovoljno za otkrivanje tražene informacije.

3.8 5242880 bita.

3.12 Na disketu može stati oko 737 stranica, a na CD oko 332800 odnosno 358400 stranica čistog teksta, ovisno od kapaciteta diska (650 MB ili 700 MB). Slijedi da na jedan CD može stati više čistog teksta nego što čovjek može pročitati tokom čitavog života!

3.13 Oko 61 KB.

3.14 a) Oko 3,5 minute. b) Oko 24,6 megabajta.

3.15 a) Oko 150 KB. b) Oko 9,8 sekundi. c) Oko 3,57 sati.

3.16 Oko 89 puta.

4.3 $(13442)_{10} = (54122)_7$

4.4 $(413203)_5 = (13553)_{10} = (200120222)_3$

4.8 $(25)_{10} = (11001)_2 = (19)_{16}$ $(53)_{10} = (110101)_2 = (35)_{16}$
$(176)_{10} = (10110000)_2 = (B0)_{16}$ $(500)_{10} = (111110100)_2 = (1F4)_{16}$
$(1023)_{10} = (1111111111)_2 = (3FF)_{16}$ $(2412)_{10} = (100101101100)_2 = (96C)_{16}$
$(4000)_{10} = (111110100000)_2 = (FA0)_{16}$ $(33852)_{10} = (1000010000111100)_2 = (843C)_{16}$
$(179978)_{10} = (101011111100001010)_2 = (2BF0A)_{16}$

4.9 $(1000)_2 = (8)_{10} = (8)_{16}$ $(110101)_2 = (53)_{10} = (35)_{16}$
$(111001)_2 = (57)_{10} = (39)_{16}$ $(10101101101)_2 = (1389)_{10} = (56D)_{16}$
$(1111100110001100110)_2 = (511078)_{10} = (7CC66)_{16}$

4.10 $(372)_{16} = (882)_{10} = (1101110010)_2$ $(F2)_{16} = (242)_{10} = (11110010)_2$
$(ABC)_{16} = (2748)_{10} = (101010111100)_2$ $(7D4)_{16} = (2004)_{10} = (11111010100)_2$
$(8FC)_{16} = (2300)_{10} = (100011111100)_2$ $(FFA)_{16} = (4090)_{10} = (111111111010)_2$
$(1000)_{16} = (4096)_{10} = (1000000000000)_2$ $(4077)_{16} = (16503)_{10} = (100000001110111)_2$
$(AFE0)_{16} = (45024)_{10} = (1010111111100000)_2$ $(FFFF)_{16} = (65535)_{10} = (1111111111111111)_2$

4.11 $(127)_{10} = (177)_8$ $(798)_{10} = (1436)_8$
$(1000)_{10} = (1750)_8$ $(10000)_{10} = (23420)_8$

$(101)_8 = (65)_{10}$ $(757)_8 = (495)_{10}$
$(1000)_8 = (512)_{10}$ $(4077)_8 = (2111)_{10}$
$(77777)_8 = (32767)_{10}$

4.12 $(11001)_2 = (31)_8$ $(1110011)_2 = (163)_8$
$(1011011)_2 = (133)_8$ $(1001000111000101)_2 = (110705)_8$

$(101)_8 = (1000001)_2$ $(252)_8 = (10101010)_2$
$(4077)_8 = (100000111111)_2$

4.13 $(101)_8 = (41)_{16}$ $(252)_8 = (AA)_{16}$
$(4077)_8 = (83F)_{16}$

$(8FC)_{16} = (4374)_8$ $(4077)_{16} = (40167)_8$
$(AFE0)_{16} = (127740)_8$

4.14 597 = ♑♌♌♌♌ ♋♑♉♑♌ = 905
♑♑♋ = $(101011)_2$ ♋♑♑♌♉ = $(1644)_8$
♌♑♋♉♋♑♑ = $(1B3A)_{16}$ $(AB0)_{16}$ = ♑♑♑♋♉♉
$(110010011)_2$ = ♌♑♌♉♋ $(771)_8$ = ♌♋♋♑♌

4.15 $2^{24} - 1 = 16777215$

4.16 Od -2^{23} do $2^{23} - 1$, odnosno od –8388608 do 8388607.

4.17 $(1101{,}01101)_2 = (13{,}40625)_{10}$

4.18 $(17{,}8125)_{10} = (10001{,}1101)_2$

4.19 $(15{,}3)_{10} = (1111{,}01001100110011...)_2 = (1111{,}0\overline{1001})_2$

4.23 $18 = 18/32 \cdot 32 = 0{,}5625 \cdot 2^5 \quad \Rightarrow \quad e = 5; \quad m = 0{,}5625$
$0{,}1 = 0.8 \cdot 1/8 = 0{,}8 \cdot 2^{-3} \quad \Rightarrow \quad e = -3; \quad m = 0{,}8$

4.26 Oko 1509949 znakova po ASCII, odnosno oko 754974 znakova po UNICODE standardu.

4.27 Oko 298 stranica.

4.31 Najviše 866 slika.

4.32 a) 65536 boja. b) Najviše 433 slike. c) Barem oko 3.46 ili više.

4.33 Najviše 6 slika.

4.34 a) Najviše 23 slike. b) Oko 394 slike.

5.1
a) $(100110)_2 + (111)_2 = (101101)_2$ $(38 + 7 = 45)$
b) $(110111)_2 + (101)_2 = (111100)_2$ $(55 + 5 = 60)$
c) $(111110)_2 + (10111)_2 = (1010101)_2$ $(62 + 23 = 85)$
d) $(111001)_2 + (10001)_2 = (1001010)_2$ $(57 + 17 = 74)$
e) $(11011100110)_2 + (10011001)_2 = (11101111111)_2$ $(1766 + 153 = 1919)$
f) $(1111111)_2 + (10101010)_2 = (100101001)_2$ $(127 + 170 = 297)$

5.2
a) $(100110)_2 - (111)_2 = (11111)_2$ $(38 - 7 = 31)$
b) $(110111)_2 - (101)_2 = (110010)_2$ $(55 - 5 = 50)$
c) $(111110)_2 - (10111)_2 = (100111)_2$ $(62 - 23 = 39)$
d) $(111001)_2 - (10001)_2 = (101000)_2$ $(57 - 17 = 40)$
e) $(11011100110)_2 - (10011001)_2 = (11001001101)_2$ $(1766 - 153 = 1613)$
f) $(1111111)_2 - (10101010)_2 = -(101011)_2$ $(127 - 170 = -43)$

5.3
a) $(100110)_2 \cdot (111)_2 = (100001010)_2$ $(38 \cdot 7 = 266)$
b) $(110111)_2 \cdot (101)_2 = (100010011)_2$ $(55 \cdot 5 = 275)$
c) $(111110)_2 \cdot (10111)_2 = (10110010010)_2$ $(62 \cdot 23 = 1426)$
d) $(111001)_2 \cdot (10001)_2 = (1111001001)_2$ $(57 \cdot 17 = 969)$
e) $(11011100110)_2 \cdot (10011001)_2 = (10000011111011101l0110)_2$ $(1766 \cdot 153 = 270198)$
f) $(1111111)_2 \cdot (10101010)_2 = (101010001010110)_2$ $(127 \cdot 170 = 21590)$

5.4
a) $(10000101)_2 : (111)_2 = (10011)_2$ $(133 : 7 = 19)$
b) $(100110101011)_2 : (110111)_2 = (101101)_2$ $(2475 : 55 = 45)$
c) $(11110011011)_2 : (100001)_2 = (111011)_2$ $(1947 : 33 = 59)$
d) $(10101011)_2 : (110000)_2 = (11,1001)_2$ $(171 : 48 = 3{,}5625)$
e) $(100101)_2 : (1010)_2 = (11,1011001100110...)_2 = (11,\overline{10110})_2$ $(37 : 10 = 3{,}7)$
f) $(101)_2 : (11000)_2 = (0,001101010...)_2 = (0,\overline{00110})_2$ $(5 : 24 = 0{,}208333... = 0.2\overline{083})$

5.5
a) $(21022)_3 + (10221)_3 = (102020)_3$ $(197 + 106 = 303)$
b) $(13203)_5 + (4413)_5 = (23121)_5$ $(1053 + 608 = 1661)$
c) $(37127)_8 + (14736)_8 = (54065)_5$ $(15959 + 6622 = 22581)$
d) $(FC1B)_{16} + (4AD)_{16} = (100C8)_{16}$ $(64539 + 1197 = 65736)$
e) $(21022)_3 - (10221)_3 = (10101)_3$ $(197 - 106 = 91)$
f) $(13203)_5 - (4413)_5 = (3240)_5$ $(1053 - 608 = 445)$
g) $(37127)_8 - (14736)_8 = (22171)_8$ $(15959 - 6622 = 9337)$
h) $(FC1B)_{16} - (4AD)_{16} = (F76E)_{16}$ $(64539 - 1197 = 63342)$

5.6 (B = 3)

	0	1	2
0	0	0	0
1	0	1	2
2	0	2	11

(B = 8)

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	10	12	14	16
3	0	3	6	11	14	17	22	25
4	0	4	10	14	20	24	30	34
5	0	5	12	17	24	31	36	43
6	0	6	14	22	30	36	44	52
7	0	7	16	25	34	43	52	61

(B = 5)

	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	11	13
3	0	3	11	14	22
4	0	4	13	22	31

(B = 16)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E
3	0	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D
4	0	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	0	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	0	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	0	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	0	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	0	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	0	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	0	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	0	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

5.7
a) $(21022)_3 \cdot (10221)_3 = (1001122102)_3$ $(197 \cdot 106 = 20882)$
b) $(13203)_5 \cdot (4413)_5 = (130441344)_5$ $(1053 \cdot 608 = 640224)$
c) $(37127)_8 \cdot (14736)_8 = (623107162)_8$ $(15959 \cdot 6622 = 105680498)$
d) $(FC1B)_{16} \cdot (4AD)_{16} = (49ACA3F)_{16}$ $(64539 \cdot 1197 = 77253183)$

5.8 a) $(2202121)_3 : (201)_3 = (10221)_3$ (2014 : 19 = 106)
b) $(1434030)_5 : (32)_5 = (24140)_5$ (30515 : 17 = 1795)
c) $(51354)_8 : (127)_8 = (364)_8$ (21228 : 87 = 244)
d) $(36F902)_{16} : (B9)_{16} = (4C12)_{16}$ (3602690 : 185 = 19474)

5.9

	☓	♌	♓	♋
☓	☓	☓	☓	☓
♌	☓	♌	♓	♋
♓	☓	♓	♌☓	♌♓
♋	☓	♋	♌♓	♓♌

5.10 a) $♋♓♌☓ + ♓☓♋ = ♌☓☓♌♋$ (228 + 35 = 263)
b) $♓☓♋♓ - ♌♌♋☓ = ♋☓♓$ (142 – 92 = 50)
c) $♓☓♋ \cdot ♌♌☓♋ = ♓♋♌♌♓♌$ (35 · 83 = 2905)
d) $♌♌♌♌♋ : ♌♋ = ♋☓♌$ (343 : 7 = 49)

5.12 Sve što u binarnom brojnom sistemu vrijedi za množenje odnosno dijeljenje sa 2^n, u brojnom sistemu sa proizvoljnom bazom $B > 1$ vrijedi za množenje odnosno dijeljenje sa B^n.

5.14 Brojevi 16 i –39 u *2KK* zapisu na 8 bita glase redom 00010000 i 11011001. Nakon binarnog sabiranja dobijamo binarni broj 11101001. Jedinica na najvišem mjestu ukazuje da se radi o negativnom broju. Nakon oduzimanja jedinice i komplementiranja dobijamo 00010111, što je binarni zapis broja 23, odnosno rezultat je –23.

5.15 Analogno zadatku 5.14, samo što odgovarajući binarni zapisi redom glase 0000000000010000, 1111111111011001, 1111111111101001 i 0000000000010111.

5.16 Postupak oduzimanja binarnih zapisa 00010000 i 11011001 (u tom poretku) daje binarni zapis 00110111, ako ignoriramo "posudbu" koju bismo trebali izvršiti prilikom oduzimanja cifara najveće težine, a koju nemamo odakle "uzeti". Dobijeni rezultat je upravo binarni zapis broja 55. Slično, postupak oduzimanja binarnih zapisa 11011001 i 00010000 (u tom poretku) daje binarni zapis 11001001 koji ukazuje na negativan rezultat. Oduzimanje jedinice i komplementiranje daje 00110111 odnosno 55 dekadno, odakle slijedi da je rezultat –55.

5.17 Množenjem binarnih zapisa 00010000 i 11011001 dobijamo binarni zapis koji više ne može stati u 8 bita. Stoga je prirodno da ga pokušamo predstaviti kao binarni zapis u $8 + 8 = 16$ bita, s obzirom da produkt dva broja od n bita može imati najviše $2n$ bita. Na taj način dobijamo binarni zapis 0000110110010000, koji očito odgovara pozitivnom broju 3472, dok bi rezultat trebao biti negativan broj –624.

5.18 Očigledno nam trebaju petocifreni *10KK* kôdovi, koji za brojeve –372 i –23441 respektivno glase 99628 i 76559. Dalje imamo račune 99628 + 23441 = 123069, 372 + 76559 = 76931 i 99628 + 76559 = 176187. U prvom slučaju, ignoriranje cifre viška daje tačan rezultat 23441. U drugom slučaju, ukoliko znamo da je rezultat negativan, oduzimanje jedinice i odbijanje od devetke (komplementiranje u bazi 10) daje 23069, što je apsolutna vrijednost tačnog rezultata –23069. U trećem slučaju, nakon ignoriranja cifre viška, odbijanja jedinice i komplementiranja dobijamo 23813, što je apsolutna vrijednost tačnog rezultata –23813. Očigledno se javlja problem kako prepoznati kada je rezultat negativan, tj. kada je potrebno komplementiranje. Za razliku od binarnog brojnog sistema, odgovor na ovo pitanje nije jednoznačno definiran i rješava se uvođenjem različitih konvencija, u koje ovdje nećemo ulaziti.

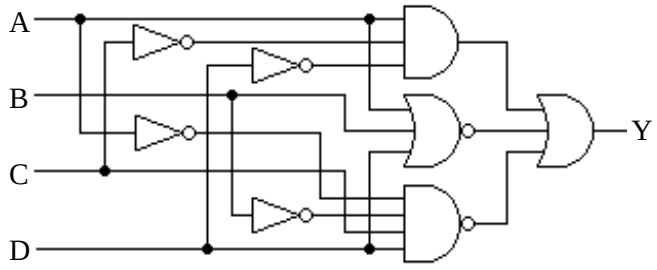
5.19 a) $(010210)_{3KK}$ b) $(10342)_{5KK}$ c) $(3506420)_{8KK}$ d) $(37DB41)_{16KK}$

6.3 $Y = (A \vee \neg C) \wedge \neg[C \vee (B \wedge E)] \vee [(C \wedge D) \vee (\neg A \wedge E)] \wedge \neg(B \vee C)$
Napomena: Okrugle zagrade unutar uglastih zagrada mogu se izostaviti ako smatramo da konjukcija ima veći prioritet u odnosu na disjunkciju.

6.4 Prikazano je samo krajnje rješenje (bez međukoraka u tabeli):

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

6.7 Jedna moguća realizacija izgleda ovako:



6.11 Uputa: Pravilo [13] slijedi iz pravila [11] ako uvrstimo Y = Z = 1 (pri tome je potrebno iskoristiti pravila [4] i [2]). Slično, pravilo [14] slijedi iz pravila [12] za Y = Z = 0, uz korištenje pravila [1] i [3]. Za dokazivanje pravila [17], treba prvo primijeniti pravilo [11], a zatim iskoristiti (prethodno dokazano) pravilo [13], čime dobijamo pravilo [18] koje smo također ranije dokazali. Za

dokazivanje pravila [21] treba uvrstiti $X = X \vee XY$ (što je pravilo [18], koje znamo dokazati), a nakon toga iskoristimo pravilo komutacije [7] i također ranije dokazano pravilo potpunog sažimanja [16].

6.12 Uputa: Postupite kao pri dokazu prve De Morganove teoreme uz zamjenu uloga konjukcije i disjunkcije, i zamjenu uloga konstanti 0 i 1.

6.13 Na osnovu aksioma logičke algebre i pretpostavki tvrdnje možemo pisati sljedeći lanac jednakosti:

$Q = Q \cdot 1 = Q \cdot (P \vee \overline{P}) = Q \cdot P \vee Q \cdot \overline{P} = P \cdot Q \vee \overline{P} \cdot Q = 0 \vee \overline{P} \cdot Q = P \cdot \overline{P} \vee \overline{P} \cdot Q =$

$= \overline{P} \cdot P \vee \overline{P} \cdot Q = \overline{P} \cdot (P \vee Q) = \overline{P} \cdot 1 = \overline{P}$

Utvrdite sami gdje je koje pravilo primijenjeno i gdje su iskorištene pretpostavke tvrdnje.

6.14 Uputa: Uzmite $P = \overline{X}$ i $Q = X$. Lako se vidi da vrijedi $PQ = 0$ i $P \vee Q = 1$ (na osnovu pravila [7] i [5], odnosno pravila [8] i [6]). Pravilo dvostruke negacije sada neposredno slijedi iz tvrdnje iz zadatka 6.13.

6.15 Uputa: Za prvu De Morganovu teoremu treba uzeti $P = XY$ i $Q = \overline{X} \vee \overline{Y}$, a za drugu De Morganovu teoremu $P = X \vee Y$ i $Q = \overline{X}\,\overline{Y}$. Zatim treba pokazati da u oba slučaja vrijedi $PQ = 0$ i $P \vee Q = 1$, što je prilično jednostavno. De Morganove teoreme tada direktno slijede iz tvrdnje iz zadatka 6.13.

6.16 Takvih funkcija ima 2^{2^n}. Naime, tablica istine ma koje logičke funkcije od *n* promjenljivih ima $N = 2^n$ redova, a u svakom od njih funkcija može uzeti vrijednost 0 ili 1, što daje ukupno 2^N mogućih različitih kombinacija. Specijalno za *n* = 2, 3, 4, 5 i 6 dobijamo 16, 256, 65536, 4294967296 i 18446744073709551616 međusobno neekvivalentnih logičkih funkcija (vidimo da njihov broj rapidno raste sa porastom *n*).

6.17 $Y = AB \vee \overline{B}C$. Uputa: Postupite slično kao u Primjeru 6.8.

6.18 a) $Y = B$ b) $Y = A \vee B \vee \overline{D}$

6.19 $Y = \overline{A}\,\overline{B} \vee BC \vee A\overline{C}$. Još je moguće eventualno izvući faktor C ispred ili iza zagrade, na primjer, $Y = \overline{A}\,\overline{B} \vee (A \vee B)C$, ali to nije neko bitno pojednostavljenje. Uputa: Nakon primjene pravila distributivnosti, eliminirajte član $\overline{A}C$ na sličan način kao u Primjeru 6.8. Postoji još jedan ekvivalentan i podjednako složen oblik funkcije, $Y = AB \vee \overline{B}\,\overline{C} \vee \overline{A}C$ (provjerite sami), ali njega nije lako izvesti iz polaznog oblika funkcije.

6.20 $Y = \overline{C}(\overline{B} \vee \overline{E})$

6.21 $Y = BC \vee AC \vee AB$. Uputa: Uvrstite $ABC = ABC \vee ABC \vee ABC$ i iskoristite pravilo potpunog sažimanja na odgovarajuće članove.

6.22 a) $Y = ABC \vee (\overline{A} \vee \overline{C})D$. Uputa: Član BD eliminirate kao u Primjeru 6.9.

b) $Y = \overline{D}$ c) $Y = A\overline{B}\,\overline{C} \vee (\overline{A}C \vee \overline{B})D$ d) $Y = \overline{B}D(A \vee \overline{C})$

6.23 a) $\overline{Y} = AB$ b) $\overline{Y} = \overline{A}\,\overline{B} \vee \overline{C}$ c) $\overline{Y} = \overline{A}(\overline{B} \vee C)$

d) $\overline{Y} = \overline{A}(B \vee C)$ e) $\overline{Y} = \overline{A} \vee \overline{B}(\overline{C} \vee \overline{D})$ f) $\overline{Y} = (\overline{A} \vee \overline{B} \vee \overline{C})(\overline{B} \vee CD)$

Negiranjem sve promjenjive i njihove negacije, kao i konjukcije i disjunkcije međusobno mijenjaju uloge. Napomenimo da se funkcija pod f) nakon primjene pravila distribucije može dodatno transformirati u jednostavniji oblik $\overline{Y} = \overline{A}CD \vee \overline{B}$, što gore nije bilo učinjeno, jer se na taj način gubi uočeno svojstvo.

6.24 a) $\overline{Y} = \overline{A}(B \vee C \vee D) \vee (A \vee D)(\overline{B} \vee \overline{A}C) = \ldots = \overline{A}(B \vee C \vee D) \vee A\overline{B}$

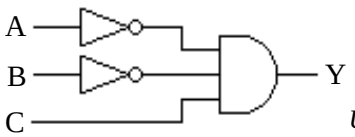
Uputa: Člana $\overline{B}D$ možete se osloboditi slično kao u Primjeru 6.8.

b) $\overline{Y} = (A \vee \overline{B} \vee \overline{C})[A\overline{B}\,\overline{D} \vee (\overline{A} \vee \overline{B} \vee D)B] = \ldots = A\overline{B}\,\overline{D} \vee B(AD \vee \overline{C})$

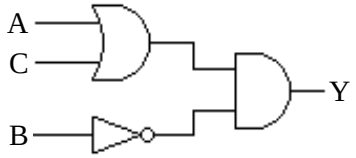
Uputa: Do oblika $\overline{Y} = A\overline{B}\,\overline{D} \vee ABD \vee \overline{A}B\overline{C} \vee B\overline{C}\,\overline{D}$ lako se dolazi. Nakon toga, izvucite faktor B ispred zagrade iz posljednja 3 člana, i razmotrite šta se može učiniti sa izrazom unutar zagrade.

6.25 $\overline{Y} = C \vee BE$

6.26 $Y = \overline{\overline{A\overline{B}} \vee \overline{(\overline{A} \vee B)C} \vee (A \vee \overline{\overline{B}C})} = \ldots = \overline{A}\,\overline{B}C$

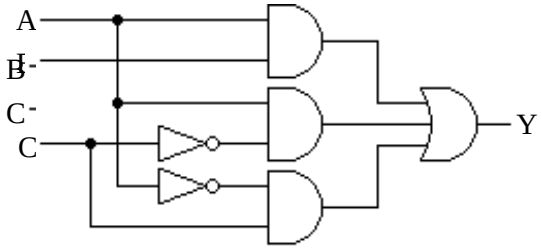


6.27 $Y = (A\overline{B} \vee \overline{\overline{A} \vee B}\,C) \vee \overline{A \vee \overline{\overline{B}C}} = \ldots = \overline{B}(A \vee C)$



6.28 $Y = ABC \vee AB \vee \overline{\overline{A} \vee B \vee C} \vee \overline{A}\,\overline{B}C \vee BC \vee \overline{A}BC = \ldots = AB \vee A\overline{C} \vee BC \vee \overline{A}C$

Dalje se, na sličan način kao u Primjeru 6.8, može eliminirati bilo član AB, bilo član BC (ali ne oba istovremeno), i dobiti jedan od sljedeća dva ekvivalentna oblika $Y = AB \vee A\overline{C} \vee \overline{A}C$ odnosno $Y = A\overline{C} \vee \overline{A}C \vee BC$. Prihvatimo li prvu varijantu, dobijamo sljedeću realizaciju:

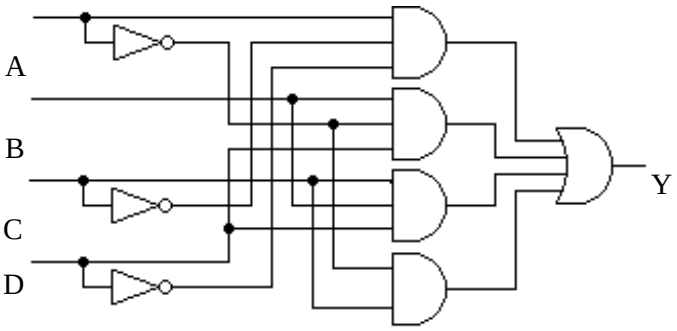


6.29 $Y = AB\overline{\overline{B}C} \vee C\overline{B}C \vee (C \vee B)(\overline{B} \vee \overline{A}) = \ldots = B \vee C$

6.30 $Y = A\overline{C}\,\overline{D} \vee A\overline{C}\,\overline{D}\,\overline{B} \vee \overline{A \vee \overline{B} \vee \overline{D}} \vee ABCD \vee \overline{A}C\,\overline{\overline{A}BCD} = \ldots =$

$= A\overline{C}\,\overline{D} \vee \overline{A}BD \vee ABCD \vee \overline{A}\,\overline{B}C \vee \overline{A}C\overline{D} = \ldots = A\overline{C}\,\overline{D} \vee \overline{A}BD \vee BCD \vee \overline{A}C$

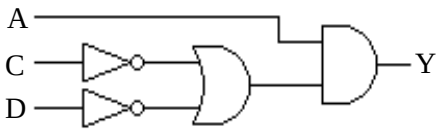
Napomena: Potrebna je prilična vještina da se samo primjenom zakona logičke algebre data funkcija svede na ovdje prikazani krajnji oblik. Dalje značajnije pojednostavljenje nije moguće. Na osnovu prikazanog oblika slijedi sljedeća realizacija:



6.31 $Y = \overline{\overline{A \vee B}\,\overline{\overline{A}B} \vee C} \vee A\overline{B \vee C} = \ldots = A$

Funkcija dobijena minimizacijom je trivijalna, tako da za njenu realizaciju nije potreban nikakav sklop (izlaz je prosto jednak promjenljivoj A).

6.32 $Y = \overline{\overline{A \vee C}\,\overline{\overline{A}\,\overline{B} \vee D}}\,A\overline{CD} = \ldots = A(\overline{C} \vee \overline{D})$



7.4 $m_{14}(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = \overline{x}_1\overline{x}_2\overline{x}_3x_4x_5x_6\overline{x}_7$

$M_{14}(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = x_1 \vee x_2 \vee x_3 \vee \overline{x}_4 \vee \overline{x}_5 \vee \overline{x}_6 \vee x_7$

Uputa: Razmotrite binarni zapis indeksa $i = 14$.

7.5 Uputa: Razmotrite binarni zapis indeksa *i* i De Morganove teoreme.

7.6 DNF: $Y = AC \vee \overline{A}\,\overline{B} \vee \overline{B}C$ (moguća su i druga rješenja)

SDNF: $Y = ABC \vee A\overline{B}C \vee \overline{A}\,\overline{B}C \vee \overline{A}\,\overline{B}\,\overline{C}$ (jedinstveno do na poredak članova)

7.7 DNF: $Y = \overline{A}C \vee D$

SDNF: $Y = ABCD \vee AB\overline{C}D \vee A\overline{B}CD \vee A\overline{B}\,\overline{C}D \vee \overline{A}BCD \vee \overline{A}B\overline{C}D \vee \overline{A}\,\overline{B}CD$
$\vee\, \overline{A}\,\overline{B}C\overline{D} \vee \overline{A}\,\overline{B}\,\overline{C}D = m_{15} \vee m_{13} \vee m_{11} \vee m_9 \vee m_7 \vee m_6 \vee m_5 \vee m_3 \vee m_2 \vee m_1$

7.8 KNF: $Y = (\overline{A} \vee C)(A \vee \overline{B})(\overline{B} \vee C)$ (jedna od mogućnosti)

SKNF: $Y = (\overline{A} \vee \overline{B} \vee C)(\overline{A} \vee B \vee C)(A \vee \overline{B} \vee \overline{C})(A \vee \overline{B} \vee C) = M_6 M_4 M_3 M_2$

7.9 SDNF: $Y = AB\overline{C} \vee A\overline{B}C \vee \overline{A}B\overline{C} \vee \overline{A}\,\overline{B}C \vee \overline{A}\,\overline{B}\,\overline{C} = m_6 \vee m_5 \vee m_2 \vee m_1 \vee m_0$

SKNF: $Y = (\overline{A} \vee \overline{B} \vee \overline{C})(\overline{A} \vee B \vee C)(A \vee \overline{B} \vee \overline{C}) = M_7 M_4 M_3$

7.10 SDNF: $Y = A\overline{B}C$

SKNF: $Y = (\overline{A} \vee \overline{B} \vee \overline{C})(\overline{A} \vee \overline{B} \vee C)(\overline{A} \vee B \vee C)(A \vee \overline{B} \vee \overline{C})(A \vee \overline{B} \vee C)(A \vee B \vee \overline{C})(A \vee B \vee C)$

7.11 a) $Y = ABCD \vee A\overline{B}C\overline{D} \vee A\overline{B}\,\overline{C}D \vee A\overline{B}\,\overline{C}\,\overline{D} \vee \overline{A}BCD \vee \overline{A}\,\overline{B}C\overline{D} \vee \overline{A}\,\overline{B}\,\overline{C}D \vee \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$

b) $Y = M_{14} M_{13} M_{12} M_{11} M_6 M_5 M_4 M_3 =$
$= (\overline{A} \vee \overline{B} \vee \overline{C} \vee D)(\overline{A} \vee \overline{B} \vee C \vee \overline{D})(\overline{A} \vee \overline{B} \vee C \vee D)(\overline{A} \vee B \vee \overline{C} \vee \overline{D})(A \vee \overline{B} \vee \overline{C} \vee D)$
$(A \vee \overline{B} \vee C \vee \overline{D})(A \vee \overline{B} \vee C \vee D)(A \vee B \vee \overline{C} \vee \overline{D})$

c) $\overline{Y} = m_{14} \vee m_{13} \vee m_{12} \vee m_{11} \vee m_6 \vee m_5 \vee m_4 \vee m_3 =$
$= ABC\overline{D} \vee AB\overline{C}D \vee AB\overline{C}\,\overline{D} \vee A\overline{B}CD \vee \overline{A}BC\overline{D} \vee \overline{A}B\overline{C}D \vee \overline{A}B\overline{C}\,\overline{D} \vee \overline{A}\,\overline{B}CD$

d) $\overline{Y} = M_{15} M_{10} M_9 M_8 M_7 M_2 M_1 M_0 =$
$= (\overline{A} \vee \overline{B} \vee \overline{C} \vee \overline{D})(\overline{A} \vee B \vee \overline{C} \vee D)(\overline{A} \vee B \vee C \vee \overline{D})(\overline{A} \vee B \vee C \vee D)(A \vee \overline{B} \vee \overline{C} \vee \overline{D})$
$(A \vee B \vee \overline{C} \vee D)(A \vee B \vee C \vee \overline{D})(A \vee B \vee C \vee D)$

7.12 a)

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

b) SDNF: $Y = ABC \vee A\overline{B}C \vee \overline{A}\,\overline{B}C$

SKNF: $Y = (\overline{A} \vee \overline{B} \vee \overline{C})(\overline{A} \vee \overline{B} \vee C)(\overline{A} \vee B \vee C)(A \vee \overline{B} \vee C)(A \vee B \vee C)$

c) Recimo, $Y = A\overline{B}C \vee \overline{A}C$, $Y = \overline{A}BC \vee \overline{B}C$ ili $Y = \overline{A}C \vee \overline{B}C$

d) Recimo, $Y = C(\overline{A} \vee \overline{B})$

7.13 $Y = AB \vee \overline{A}B \vee \overline{A}\,\overline{B}$

7.14 a) SDNF: $Y = ABC \vee AB\overline{C} \vee A\overline{B}C \vee A\overline{B}\,\overline{C}$

SKNF: $Y = (A \vee \overline{B} \vee \overline{C})(A \vee \overline{B} \vee C)(A \vee B \vee \overline{C})(A \vee B \vee C)$

b) SDNF: $Y = A\overline{B}\overline{C} \vee \overline{A}B\overline{C}$

SKNF: $Y = (\overline{A} \vee \overline{B} \vee \overline{C})(\overline{A} \vee B \vee \overline{C})(\overline{A} \vee B \vee C)(A \vee \overline{B} \vee \overline{C})(A \vee B \vee \overline{C})(A \vee B \vee C)$

c) SDNF: $Y = ABC \vee AB\overline{C} \vee A\overline{B}C \vee A\overline{B}\,\overline{C} \vee \overline{A}BC \vee \overline{A}B\overline{C} \vee \overline{A}\,\overline{B}C \vee \overline{A}\,\overline{B}\,\overline{C}$

SKNF: ne postoji

d) SDNF: ne postoji

SKNF: $Y = (\overline{A} \vee \overline{B} \vee \overline{C})(\overline{A} \vee \overline{B} \vee C)(\overline{A} \vee B \vee \overline{C})(\overline{A} \vee B \vee C)(A \vee \overline{B} \vee \overline{C})(A \vee \overline{B} \vee C)$
$(A \vee B \vee \overline{C})(A \vee B \vee C)$

Napomena: Konstanta 0 je jedina logička funkcija za koju ne postoji SDNF, dok je konstanta 1 jedina logička funkcija za koju ne postoji SKNF.

7.16 DNF: $Y = ABC \vee A\overline{C}D \vee \overline{A}B\overline{C} \vee \overline{A}CD$ (jedna od mogućnosti)

KNF: $Y = (A \vee B \vee C)(A \vee \overline{C} \vee D)(\overline{A} \vee B \vee \overline{C})(\overline{A} \vee C \vee D)$

7.17 a) Y = 1 b) Y = 0

7.18 $X \uparrow Y = [(X \downarrow X) \downarrow (Y \downarrow Y)] \downarrow [(X \downarrow X) \downarrow (Y \downarrow Y)]$

$X \downarrow Y = [(X \uparrow X) \uparrow (Y \uparrow Y)] \uparrow [(X \uparrow X) \uparrow (Y \uparrow Y)]$

7.19 Bez prethodnog pojednostavljenja:

$Y = [(A \uparrow A) \uparrow B] \uparrow \{[A \uparrow (C \uparrow C)] \uparrow [(B \uparrow B) \uparrow C]\}$
$Y = \{[A \downarrow (B \downarrow B)] \downarrow \{[(A \downarrow A) \downarrow C] \downarrow [B \downarrow (C \downarrow C)]\}\} \downarrow \{[A \downarrow (B \downarrow B)] \downarrow \{[(A \downarrow A) \downarrow C] \downarrow [B \downarrow (C \downarrow C)]\}\}$

Nakon prethodnog pojednostavljenja:

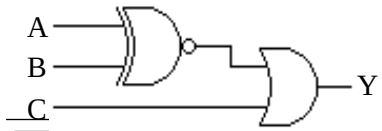
$Y = [(A \uparrow A) \uparrow C]\,[(B \uparrow B) \uparrow C]$
$Y = [(A \downarrow A) \downarrow (B \downarrow B)] \downarrow (C \downarrow C)$

Napomena: Prikazana rješenja nisu jedinstvena. U prikazanim rješenjima bez prethodnog pojednostavljenja ipak su izvršena neka trivijalna pojednostavljenja, recimo ona koja slijede iz pravila dvojne negacije.

7.20 $A \oplus B = [A \uparrow (B \uparrow B)]\,[(A \uparrow A) \uparrow B] = (A \downarrow B) \downarrow [(A \downarrow A) \downarrow (B \downarrow B)]$

Napomena: Postoje i druge mogućnosti (pogledajte npr. zadatak 7.23).

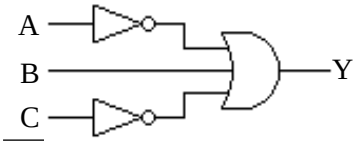
7.24 $Y = \overline{\overline{\overline{A} \oplus B}\,\overline{AB}} \vee C \vee \overline{A}\,\overline{B \vee C} = \ldots = AB \vee \overline{A}\,\overline{B} \vee C = \overline{A \oplus B} \vee C$



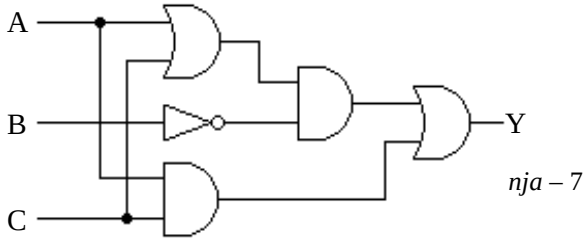
7.25 $Y = [\overline{A}B \vee (\overline{A} \oplus B)\overline{C}] \vee \overline{\overline{A}\,\overline{B}\,\overline{C}} = \ldots = 0$

Funkcija dobijena minimizacijom je trivijalna, tako da za njenu realizaciju nije potreban nikakav sklop (izlaz je identički jednak nuli).

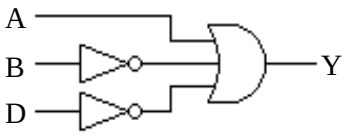
7.26 $Y = [(A \oplus \overline{B}) \vee \overline{AC}] \vee \overline{\overline{AC}\,A} \vee ABD = \ldots = \overline{A} \vee B \vee \overline{C}$



7.27 $Y = \overline{\overline{A}B} \vee \overline{\overline{A} \oplus \overline{B}\,\overline{C}} \vee \overline{\overline{A}\,\overline{B}\overline{C}} = \ldots = (A \vee C)\overline{B} \vee AC$ ili $A(\overline{B} \vee C) \vee \overline{B}C$



7.28 $Y = [A\overline{B} \vee \overline{(A \oplus B)D}] \vee A\overline{C}\,\overline{\overline{B} \vee D} = \ldots = A \vee \overline{B} \vee \overline{D}$



7.29 a) $Y = 1 \oplus A \oplus AB$ b) $Y = 1 \oplus A \oplus B \oplus AB$ c) $Y = AB \oplus AC \oplus ABC$
d) $Y = 1 \oplus A \oplus C \oplus AC \oplus BC$ e) $Y = A \oplus C \oplus AC$ f) $Y = C \oplus ABC$

7.30 Rezultati provedenog ispitivanja sumarno su prikazani u sljedećoj tabeli:

	0	1	A	$\overline{A}$	AB	$A \vee B$	$A \oplus B$	$A \uparrow B$	$A \downarrow B$	$A \Rightarrow B$
Zadrška nule	da	ne	da	ne	da	da	da	ne	ne	ne
Zadrška jedinice	ne	da	da	ne	da	da	ne	ne	ne	da
Linearnost	da	da	da	da	ne	ne	da	ne	ne	ne
Samodualnost	ne	ne	da	da	ne	ne	ne	ne	ne	ne
Monotonost	da	da	da	ne	da	da	ne	ne	ne	ne

Napomena: Iz prikazane tabele Postove teoreme, neposredno slijedi da se uz pomoć implikacije i konstante 0 također može izraziti ma koja logička funkcija. I zaista, vrijede relacije:

$\overline{X} = X \Rightarrow 0$, $XY = [X \Rightarrow (Y \Rightarrow 0)] \Rightarrow 0$, $X \vee Y = (X \Rightarrow 0) \Rightarrow Y$

7.31 Uputa: Vrijedi $f(x_1, \ldots x_i, \ldots x_n) = f(x_1, \ldots 0, \ldots x_n)\, x_i \vee f(x_1, \ldots 1, \ldots x_n)\overline{x_i}$, što je sasvim lako provjeriti. Dalje, prema osobini monotonosti vrijedi $f(x_1, \ldots 0, \ldots x_n) \le f(x_1, \ldots 1, \ldots x_n)$ odakle je ili $f(x_1, \ldots 0, \ldots x_n) = 0$, ili $f(x_1, \ldots 1, \ldots x_n) = 1$, ili $f(x_1, \ldots 0, \ldots x_n) = f(x_1, \ldots 1, \ldots x_n)$. U sva tri slučaja, iz izraza $f(x_1, \ldots x_i, \ldots x_n)$ se možemo osloboditi negacije promjenljive x_i. Slijedi da se izraz $f(x_1, \ldots x_i, \ldots x_n)$ može napisati samo preko konjukcije i disjunkcije ukoliko se u tom obliku mogu napisati izrazi $f(x_1, \ldots 0, \ldots x_n)$ i $f(x_1, \ldots 1, \ldots x_n)$, koji zavise od n–1 promjenljivih. Tvrdnja sad slijedi na osnovu indukcije po broju promjenljivih.

7.32 a) $Y = A + B - 2AB$ b) $Y = 1 - A - B + 2AB$ c) $Y = 1 - A - B + AB$
d) $Y = 1 - A + AB$ e) $Y = 1 - A - B + AB$ f) $Y = AB + AC - ABC$
g) $Y = 1 - A - C + AC + BC$ h) $Y = A + C - AC$ i) $Y = C - ABC$

8.7 Jesu.

8.8 Nije (član BD se može izbaciti).

8.11 Uputa: Potrebno je prvo pokazati da su sve elementarne konjukcije u DNF koja se dobije nakon izvršavanja svih mogućih sažimanja proste implikante. Pretpostavimo da postoji elementarna konjukcija φ koja nije prosta implikanta. Tada iz nje možemo izbaciti neku promjenljivu ili njenu negaciju (recimo X_i) i tako dobiti kraću elementarnu konjukciju φ' koja je također implikanta. Proširimo sada φ' do SDNF oblika. Tako dobijene minterme moraju biti i minterme polazne funkcije, s obzirom da je φ' njena implikanta. Zbog činjenice da φ'' ne sadrži X_i, sve tako dobijene minterme moraju činiti parove od kojih jedan član sadrži X_i, a drugi njenu negaciju (ovo je posljedica razvoja $\varphi' = \varphi' X_i \vee \varphi' \overline{X_i}$). Međutim, u postupku sažimanja polazeći od takvih mintermi, jednom ćemo doći do članova $\varphi' X_i$ i $\varphi' \overline{X_i}$, odnosno φ i $\varphi' X_i$ (s obzirom da je $\varphi' X_i = \varphi$), što protivurječi pretpostavci da je φ dobijena nakon što su iscrpljene sve mogućnosti sažimanja. Dalje je potrebno pokazati da ne postoji prosta implikanta koja se ne može dobiti opisanim postupkom. Pretpostavimo da takva postoji, i nazovimo je ψ. Njenim proširivanjem do SDNF dobićemo minterme koje su ujedno i minterme polazne funkcije. Posmatranjem tog proširivanja u obrnutom smjeru dobićemo lanac sažimanja koji upravo dovodi do ψ polazeći od mintermi koje čine SDNF polazne funkcije, što protivurječi pretpostavci da se ψ ne može dobiti kao rezultat sažimanja.

8.12 a) CD, $B\overline{C}\,\overline{D}$, $A\overline{B}\,\overline{C}$, $A\overline{C}\,\overline{D}$, $A\overline{B}D$, $AB\overline{D}$, ABC

b) $Y = CD \vee B\overline{C}\,\overline{D} \vee A\overline{B}\,\overline{C} \vee AB\overline{D}$, $Y = CD \vee B\overline{C}\,\overline{D} \vee A\overline{B}\,\overline{C} \vee ABC$,
$Y = CD \vee B\overline{C}\,\overline{D} \vee A\overline{C}\,\overline{D} \vee A\overline{B}D \vee AB\overline{D}$, $Y = CD \vee B\overline{C}\,\overline{D} \vee A\overline{C}\,\overline{D} \vee A\overline{B}D \vee ABC$

c) $Y = CD \vee B\overline{C}\,\overline{D} \vee A\overline{B}\,\overline{C} \vee AB\overline{D}$ ili $Y = CD \vee B\overline{C}\,\overline{D} \vee A\overline{B}\,\overline{C} \vee ABC$ (obje su MDNF)

8.13 $Y = (A \vee B \vee C)(A \vee \overline{C} \vee D)(B \vee \overline{C} \vee D)(\overline{B} \vee C \vee \overline{D})$

Uputa: Proste implikante negacije ove funkcije su $\overline{A}\,\overline{B}\,\overline{C}$, $\overline{A}\,\overline{B}\,\overline{D}$, $\overline{A}C\overline{D}$, $\overline{A}C\overline{D}$, $\overline{B}C\overline{D}$ i $B\overline{C}D$. Posljednje tri su esencijalne, a od prve tri dovoljno je zadržati samo prvu.

8.14 a) Proste implikante: $\overline{A}B\overline{C}D$, $\overline{A}\,\overline{B}\,\overline{D}$, $AB\overline{D}$, $\overline{B}C$, AC

NDNF: Y = $\overline{A}B\overline{C}D \vee \overline{A}\,\overline{B}\,\overline{D} \vee AB\overline{D} \vee \overline{B}C \vee AC$ (sve proste implikante su esencijalne)

NDNF je ujedno i MDNF.

b) Proste implikante: $B\overline{C}$, BD, $\overline{C}\,\overline{D}$

NDNF: Y = $BD \vee \overline{C}\,\overline{D}$ (implikanta $B\overline{C}$ može se izostaviti, a preostale dvije su esencijalne)

NDNF je ujedno i MDNF.

8.15 Y = $(B \vee \overline{D})(\overline{C} \vee D)$

Uputa: Proste implikante negacije ove funkcije su $\overline{B}C$, $\overline{B}D$ i $C\overline{D}$. Posljednje dvije su esencijalne, a prva se može potpuno izostaviti.

8.16 a) Proste implikante: $A\overline{B}$, $\overline{B}C$, $B\overline{C}$, $A\overline{C}$

NDNF: Y = $A\overline{B} \vee \overline{B}C \vee B\overline{C}$, Y = $\overline{B}C \vee B\overline{C} \vee A\overline{C}$ (implikante $\overline{B}C$ i $B\overline{C}$ su esencijalne)

Obje NDNF su ujedno i MDNF.

b) Proste implikante: $A\overline{C}$, $\overline{B}$

NDNF: Y = $A\overline{C} \vee \overline{B}$ (obje proste implikante su esencijalne)

NDNF je ujedno i MDNF.

c) Proste implikante: $A\overline{B}$, $\overline{A}B$, $B\overline{C}$, $\overline{B}C$, $A\overline{C}$, $\overline{A}C$

NDNF: Y = $A\overline{B} \vee \overline{A}B \vee A\overline{C} \vee \overline{A}C$, Y = $A\overline{B} \vee \overline{A}B \vee B\overline{C} \vee \overline{B}C$, Y = $A\overline{C} \vee \overline{A}C \vee B\overline{C} \vee \overline{B}C$,
Y = $A\overline{B} \vee \overline{A}C \vee B\overline{C}$, Y = $\overline{A}B \vee A\overline{C} \vee \overline{B}C$

MDNF: Y = $A\overline{B} \vee \overline{A}C \vee B\overline{C}$ ili Y = $\overline{A}B \vee A\overline{C} \vee \overline{B}C$

8.17 Pogledajte rješenje zadatka 8.14 pod b) (radi se o istoj funkciji, zapisanoj na drugačiji način).

8.18 a) Proste implikante: $\overline{A}BD$, $\overline{B}\,\overline{D}$

NDNF: Y = $\overline{A}BD \vee \overline{B}\,\overline{D}$ (obje proste implikante su esencijalne)

NDNF je ujedno i MDNF.

b) Proste implikante: $\overline{A}\,\overline{B}\,\overline{C}$, $AB\overline{C}$, $\overline{A}\,\overline{B}\,\overline{D}$, $\overline{A}BD$, $AB\overline{D}$, $\overline{A}\,\overline{C}\,\overline{D}$, $B\overline{C}D$

NDNF: Y = $\overline{A}\,\overline{B}\,\overline{D} \vee \overline{A}BD \vee AB\overline{D} \vee \overline{A}\,\overline{B}\,\overline{C} \vee AB\overline{C}$, Y = $\overline{A}\,\overline{B}\,\overline{D} \vee \overline{A}BD \vee AB\overline{D} \vee \overline{A}\,\overline{B}\,\overline{C} \vee B\overline{C}D$,
Y = $\overline{A}\,\overline{B}\,\overline{D} \vee \overline{A}BD \vee AB\overline{D} \vee \overline{A}\,\overline{C}\,\overline{D} \vee AB\overline{C}$, Y = $\overline{A}\,\overline{B}\,\overline{D} \vee \overline{A}BD \vee AB\overline{D} \vee \overline{A}\,\overline{C}\,\overline{D} \vee B\overline{C}D$

Sve četiri NDNF su ujedno i MDNF.

c) Proste implikante: $\overline{A}BD$, $\overline{B}\,\overline{D}$, $A\overline{D}$, C

NDNF: Y = $\overline{A}BD \vee \overline{B}\,\overline{D} \vee A\overline{D} \vee C$ (sve proste implikante su esencijalne)

NDNF je ujedno i MDNF.

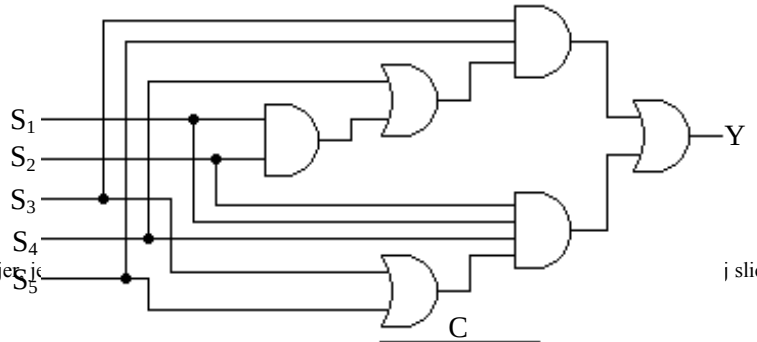
8.19 Y = $(\overline{A} \vee B \vee \overline{C})(\overline{A} \vee \overline{B} \vee D)(\overline{B} \vee C)$ ili Y = $(\overline{A} \vee B \vee \overline{C})(\overline{A} \vee \overline{C} \vee D)(\overline{B} \vee C)$

Uputa: Proste implikante negacije ove funkcije su $A\overline{B}C$, $AB\overline{D}$, $AC\overline{D}$ i $B\overline{C}$. Prva i posljednja su esencijalne, a od preostale dvije možemo ravnopravno zadržati bilo koju.

8.20 $Y = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} \vee \overline{A}\,\overline{B}DE \vee ACDE \vee B\overline{C}DE \vee A\overline{D}\,\overline{E} \vee B\overline{D}\,\overline{E}$ ili
$Y = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} \vee ABDE \vee \overline{A}\,\overline{C}DE \vee \overline{B}CDE \vee A\overline{D}\,\overline{E} \vee B\overline{D}\,\overline{E}$

Uputa: Proste implikante ove funkcije su $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$, $\overline{A}\,\overline{B}\,\overline{C}E$, $\overline{A}\,\overline{B}DE$, $ABDE$, $\overline{A}\,\overline{C}DE$, $ACDE$, $\overline{B}CDE$, $B\overline{C}DE$, $A\overline{D}\,\overline{E}$, $B\overline{D}\,\overline{E}$ i $\overline{C}\,\overline{D}\,\overline{E}$. Deveta i deseta implikanta su esencijalne, a ostale nisu. Od ovih implikanti moguće je formirati čak 15 različitih nesvodljivih disjunktivnih normalnih formi (dvije sa po 6 implikanti, osam sa po 7 implikanti, i pet sa po osam implikanti) od kojih su prikazane dvije minimalne.

8.25 Uputa: Na osnovu postavke problema, prvo sastavite tabelu istine, a nakon toga formirajte SDNF tražene funkcije. Nakon toga, primijenite Quineov algoritam da nađete MDNF tražene funkcije, koja glasi $Y = S_3S_4S_5 \vee S_1S_2S_3S_5 \vee S_1S_2S_3S_4 \vee S_1S_2S_4S_5$. Dobijena forma se može dodatno optimizirati na razne načine, npr. na oblik $Y = S_3S_5(S_4 \vee S_1S_2) \vee S_1S_2S_4(S_3 \vee S_5)$. Na osnovu predloženog oblika slijedi realizacija kao na sljedećoj slici:



9.2 Na primjer, je slici:

A \ CD	00	01	11	10
00	y_0	y_1	y_3	y_2
01	y_4	y_5	y_7	y_6
11	y_{12}	y_{13}	y_{15}	y_{14}
10	y_8	y_9	y_{11}	y_{10}

Prikazani razmještaj se zapravo isu ništa drugo nego gore prikazani dijagram, uz korištenje drugačije simbolike, koja je prika ljenih simbola je jasan sam po sebi):

9.3 Uputa: Prvo pokažite da se bina sažimati moraju razlikovati samo u jednom bitu.

9.4 Uputa: Prvo pokažite da svi bin tvorku moraju imati identične bite na svim pozicijama, osim na dvije pozicije.

9.5 Ako su indeksi mintermi koje čine osmica redom i, j, k, l, m, n, p i q uz $i<j<k<l<m<n<p<q$, tada mora vrijediti $j-i = l-k = n-m = q-p$ i $k-i = p-m$. Pored toga, razlike $j-i$, $k-i$ i $m-i$ moraju biti stepeni broja 2.

9.6 Uputa: Posmatrajte uvjete koje trebaju zadovoljavati indeksi mintermi koje čine par, i pokažite da kako god da se rasporede polja u dvije dimenzije, sa 5 ili više promjenljivih će se morati dogoditi da dva susjedna polja ne čine par (ovo se lakše vidi kada se indeksi posmatraju u binarnom brojnom sistemu).

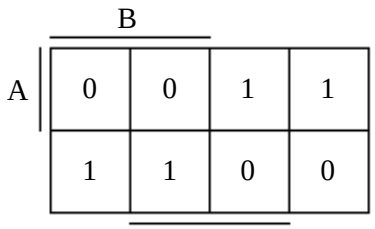
9.9 To su prekrivanja koja čine parovi (y_0, y_1), (y_1, y_3), (y_4, y_6) i (y_6, y_7), odnosno parovi (y_0, y_4), (y_1, y_3), (y_3, y_7) i (y_4, y_6).

9.10

A \ B	0	1	3	2
0	1	0	0	0
1	1	0	0	0
3	1	0	0	1
2	1	0	0	1

MDNF: $Y = \overline{A}\,\overline{C} \vee B\overline{C}$
MKNF: $Y = (\overline{A} \vee B)\,\overline{C}$

9.11



MDNF: $Y = A\overline{B} \vee \overline{A}B$

MKNF: $Y = (A \vee B)(\overline{A} \vee \overline{B})$

9.12 a) MDNF: $Y = A\,D$ C

MKNF: $Y = \overline{A}\,\overline{D}$

b) MDNF: $Y = \overline{A}\,\overline{B}C \vee A\overline{B}\,\overline{C} \vee \overline{A}\,\overline{C}D \vee ACD$

MKNF: $Y = (A \vee \overline{B} \vee \overline{C})(\overline{A} \vee \overline{B} \vee C)(A \vee C \vee D)(\overline{A} \vee \overline{C} \vee D)$

c) MDNF: $Y = AB\overline{D} \vee \overline{A}\,\overline{C}\,\overline{D} \vee ACD \vee \overline{B}\,\overline{C}D$ ili $Y = \overline{A}\,\overline{B}\,\overline{C} \vee ABC \vee A\overline{B}D \vee B\overline{C}\,\overline{D}$

MKNF: $Y = (\overline{A} \vee B \vee D)(\overline{B} \vee C \vee \overline{D})(A \vee \overline{C})$

d) MDNF: $Y = ACD \vee \overline{A}\,\overline{D} \vee \overline{B}$

MKNF: $Y = (\overline{A} \vee \overline{B} \vee C)(A \vee \overline{B} \vee \overline{D})(\overline{A} \vee \overline{B} \vee D)$ ili $Y = (A \vee \overline{B} \vee \overline{D})(\overline{A} \vee \overline{B} \vee D)(\overline{B} \vee C \vee \overline{D})$

U slučaju a) MDNF i MKNF su identični, i zahtijevaju 4 ulaza za realizaciju. U slučaju b) MDNF i MKNF su jednako povoljne za realizaciju, jer obje zahtijevaju 19 ulaza. U slučaju c) MKNF forma je povoljnija (15 ulaza za MKNF prema 20 ulaza za MDNF). U slučaju d) MDNF forma je povoljnija (11 ulaza za MDNF prema 15 ulaza za MKNF).

9.13 MDNF: $Y = B\overline{C} \vee \overline{D}$

MKNF: $Y = (B \vee \overline{D})(\overline{C} \vee \overline{D})$

9.14 a) MDNF: $Y = \overline{B}\,\overline{C} \vee \overline{B}\,\overline{D} \vee BCD$

MKNF: $Y = (\overline{B} \vee C)(\overline{B} \vee D)(B \vee \overline{C} \vee \overline{D})$

b) MDNF: $\overline{Y} = B\overline{C} \vee B\overline{D} \vee \overline{B}CD$

MKNF: $\overline{Y} = (B \vee C)(B \vee D)(\overline{B} \vee \overline{C} \vee \overline{D})$

9.15 a) MDNF: $Y = \overline{A}B \vee C$

MKNF: $Y = (\overline{A} \vee C)(B \vee C)$

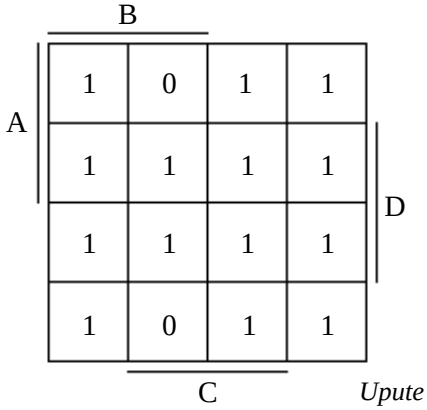
b) MDNF: $\overline{Y} = A\overline{C} \vee \overline{B}\,\overline{C}$

MKNF: $\overline{Y} = (A \vee \overline{B})\overline{C}$

9.16 MDNF: $Y = \overline{A}\,\overline{C} \vee \overline{B}$

MKNF: $Y = (\overline{A} \vee \overline{B})(\overline{B} \vee \overline{C})$

9.17



MDNF / MKNF: $Y = \overline{B} \vee \overline{C} \vee D$

9.18 MDNF: $Y = AB\overline{D}E \vee \overline{A}B\overline{D}\overline{E} \vee A\overline{B}D$

MKNF: $Y = (\overline{A} \vee B \vee D)(A \vee \overline{D})(A \vee \overline{E})(\overline{B} \vee \overline{D})(\overline{B} \vee E)$

MDNF oblik (18 ulaza) je povoljniji za realizaciju od MKNF (20 ulaza).

9.19 MDNF: $Y = A\overline{B}\overline{D} \vee \overline{A}D \vee \overline{A}E \vee BD \vee B\overline{E}$

MKNF: $Y = (A \vee B \vee D \vee E)(\overline{A} \vee \overline{B} \vee D \vee \overline{E})(\overline{A} \vee B \vee \overline{D})$

MKNF oblik (18 ulaza) je povoljniji za realizaciju od MDNF (20 ulaza).

9.20 MDNF: $Y = A\overline{B}\overline{D} \vee ABD \vee ADE \vee CE$

MKNF: $Y = (\overline{B} \vee C \vee D)(\overline{B} \vee D \vee E)(B \vee \overline{D} \vee E)(A \vee C)(A \vee E)$

9.21 MDNF: $Y = \overline{A}\overline{B}\overline{C}\overline{D} \vee ADEF \vee BD\overline{E}F \vee A\overline{B}\overline{F} \vee B\overline{C}D \vee B\overline{C}\overline{E}$

MKNF: $Y = (A \vee B \vee \overline{D} \vee \overline{E})(\overline{A} \vee D \vee \overline{E} \vee \overline{F})(B \vee \overline{C} \vee E \vee \overline{F})(A \vee \overline{C} \vee \overline{E})(A \vee \overline{C} \vee F)(\overline{B} \vee \overline{C} \vee F)(\overline{B} \vee D)$

ili $Y = (A \vee B \vee \overline{D} \vee \overline{E})(\overline{A} \vee D \vee \overline{E} \vee \overline{F})(B \vee \overline{C} \vee E \vee \overline{F})(A \vee B \vee \overline{C})(A \vee \overline{C} \vee \overline{E})(\overline{B} \vee \overline{C} \vee F)(\overline{B} \vee D)$

9.22 Postoji nekoliko različitih MDNF i MKNF. Jedna mogućnost je:

MDNF: $Y = A\overline{B}CD\overline{E} \vee \overline{A}BCD\overline{E} \vee BC\overline{D}EF \vee AB\overline{C}\overline{E} \vee AB\overline{D}\overline{E} \vee \overline{A}BEF \vee ADEF \vee \overline{B}CD\overline{F}$
$\vee \overline{B}\overline{C}EF \vee \overline{C}DEF$

MKNF: $Y = (\overline{A} \vee \overline{B} \vee C \vee D \vee \overline{E})(\overline{A} \vee \overline{C} \vee \overline{D} \vee \overline{E} \vee \overline{F})(\overline{A} \vee \overline{B} \vee \overline{C} \vee \overline{D})(A \vee \overline{B} \vee C \vee E)(A \vee B \vee E \vee \overline{F})$
$(A \vee D \vee E)(B \vee \overline{C} \vee D)(B \vee C \vee E)(B \vee C \vee F)(\overline{B} \vee \overline{E} \vee \overline{F})$

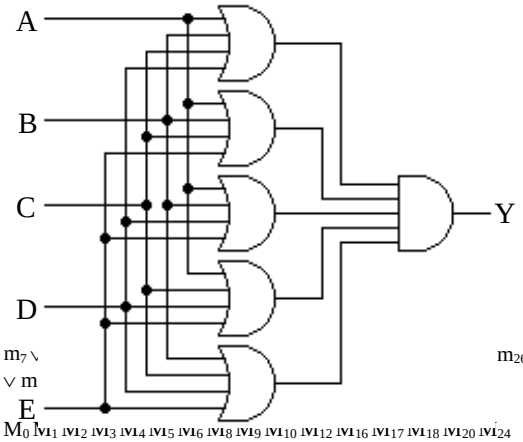
9.34 SDNF: $Y = m_3 \vee m_5 \vee m_6 \vee m_7 \vee m_9 \vee m_{10} \vee m_{11} \vee m_{12} \vee m_{13} \vee m_{14} \vee m_{15} \vee m_{17} \vee m_{18} \vee m_{19} \vee m_{20}$
$\vee m_{21} \vee m_{22} \vee m_{23} \vee m_{24} \vee m_{25} \vee m_{26} \vee m_{27} \vee m_{28} \vee m_{29} \vee m_{30} \vee m_{31}$

SKNF: $Y = M_0 M_1 M_2 M_4 M_8 M_{16}$

MDNF: $Y = AB \vee AC \vee AD \vee AE \vee BC \vee BD \vee BE \vee CD \vee CE \vee DE$

MKNF: $Y = (A \vee B \vee C \vee D)(A \vee B \vee C \vee E)(A \vee B \vee D \vee E)(A \vee C \vee D \vee E)(B \vee C \vee D \vee E)$

MKNF je povoljnija za realizaciju (25 umjesto 30 ulaza):



9.35 SDNF: $Y = m_7 \vee m_{26} \vee m_{27} \vee m_{28} \vee m_{29}$
$\vee m$

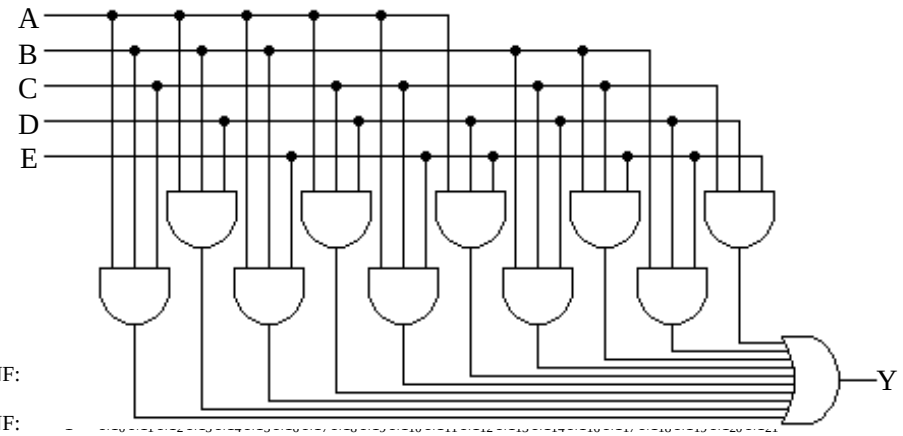
SKNF: $Y = M_0 M_{11} M_{12} M_{13} M_{14} M_{15} M_{16} M_{18} M_{19} M_{10} M_{12} M_{16} M_{17} M_{18} M_{20} M_{24}$

MDNF: $Y = ABC \vee ABD \vee ABE \vee ACD \vee ACE \vee ADE \vee BCD \vee BCE \vee BDE \vee CDE$

MKNF: $Y = (A \vee B \vee C)(A \vee B \vee D)(A \vee B \vee E)(A \vee C \vee D)(A \vee C \vee E)(A \vee D \vee E)(B \vee C \vee D)$

$(B \vee C \vee E)(B \vee D \vee E)(C \vee D \vee E)$

MDNF i MKNF su podjednako povoljne za realizaciju (40 ulaza u oba slučaja). Prikazana je realizacija na osnovu MDNF:



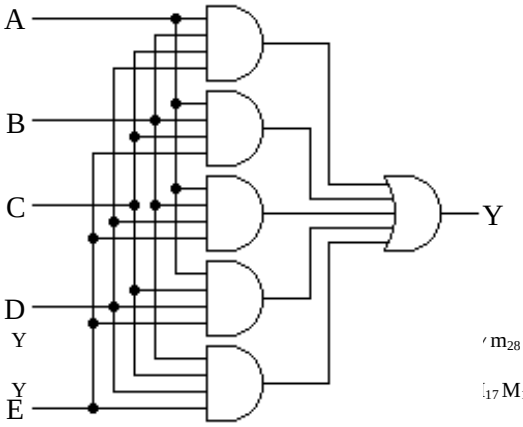
9.36 SDNF:

SKNF: $M_{22} M_{24} M_{25} M_{26} M_{28}$

MDNF: $Y = ABCD \vee ABCE \vee ABDE \vee ACDE \vee BCDE$

MKNF: $Y = (A \vee B)(A \vee C)(A \vee D)(A \vee E)(B \vee C)(B \vee D)(B \vee E)(C \vee D)(C \vee E)(D \vee E)$

MDNF je povoljnija za realizaciju (25 umjesto 30 ulaza):



9.37 SDNF: Y m_{28}

SKNF: Y $M_{17} M_{18} M_{20} M_{23} M_{24} M_{27} M_{29} M_{30} M_{31}$

MDNF: $Y = \overline{A}\,\overline{B}CDE \vee \overline{A}B\overline{C}DE \vee \overline{A}BC\overline{D}E \vee \overline{A}BCD\overline{E} \vee A\overline{B}\,\overline{C}DE \vee A\overline{B}C\overline{D}E \vee A\overline{B}CD\overline{E}$
$\vee AB\overline{C}\,\overline{D}E \vee AB\overline{C}D\overline{E} \vee ABC\overline{D}\,\overline{E}$

MKNF: $Y = (\overline{A} \vee \overline{B} \vee \overline{C} \vee \overline{D})(\overline{A} \vee \overline{B} \vee \overline{C} \vee \overline{E})(\overline{A} \vee \overline{B} \vee \overline{D} \vee \overline{E})(\overline{B} \vee \overline{C} \vee \overline{D} \vee \overline{E})(A \vee B \vee C)$
$(A \vee B \vee D)(A \vee B \vee E)(A \vee C \vee D)(A \vee C \vee E)(A \vee D \vee E)(B \vee C \vee D)(B \vee C \vee E)$

$(B \vee D \vee E)(C \vee D \vee E)$

I MDNF i MKNF zahtijevaju po 65 ulaza za realizaciju (povoljnije realizacije su moguće uz upotrebu EXOR logičkih kola).

9.38 SDNF: $Y = m_{15} \vee m_{23} \vee m_{27} \vee m_{29} \vee m_{30}$

SKNF: $Y = M_0 M_1 M_2 M_3 M_4 M_5 M_6 M_7 M_8 M_9 M_{10} M_{11} M_{12} M_{13} M_{14} M_{16} M_{17} M_{18} M_{19} M_{20} M_{21}$
$M_{22} M_{24} M_{25} M_{26} M_{28} M_{31}$

MDNF: $Y = \overline{A}BCDE \vee A\overline{B}CDE \vee AB\overline{C}DE \vee ABC\overline{D}E \vee ABCD\overline{E}$

MKNF: $Y = (\overline{A} \vee \overline{B} \vee \overline{C} \vee \overline{D} \vee \overline{E})(A \vee B)(A \vee C)(A \vee D)(A \vee E)(B \vee C)(B \vee D)(B \vee E)(C \vee D)$
$(C \vee E)(D \vee E)$

MDNF zahtijeva 35, a MKNF 41 ulaz za realizaciju.

10.4 20 ns za polaznu mrežu, 40 ns za optimiziranu mrežu.

10.5 a) $Y = ABD(C \vee E) \vee F$ (8 ulaza, 3 nivoa)

b) $Y = B(AE \vee C \vee D) \vee ACD$ (12 ulaza, 4 nivoa)
c) $Y = (A \vee B)(C \vee DE)$ (8 ulaza, 3 nivoa)
d) $Y = (A \vee B \vee C)(D \vee E) \vee AF$ (11 ulaza, 3 nivoa)
e) $Y = [(A \vee B)(C \vee D) \vee E](F \vee G)$ (12 ulaza, 4 nivoa)

10.6 $Y = (\overline{A}C \vee \overline{B})(A\overline{C} \vee D)$

Uputa: Svedite datu funkciju na MKNF, i ručno optimizirajte dobijenu MKNF.

10.7 a) $Y = \overline{A}(B \oplus D) \vee \overline{C}$ b) $Y = \overline{B \oplus D} \vee \overline{A} \vee C$

c) $Y = \overline{B \oplus D}\,\overline{C} \vee \overline{A}$ d) $Y = (B \oplus D) \vee A \vee \overline{C}$

10.8 SDNF za traženu logičku funkciju glasi $Y = m_1 \vee m_2 \vee m_4 \vee m_7 \vee m_8 \vee m_{11} \vee m_{13} \vee m_{14}$. Upis ove funkcije u Veitchov dijagram daje 8 usamljenih jedinica koje se ni sa čim ne mogu upariti, tako da je MDNF date funkcije istovjetna sa SDNF (tj. SDNF je ujedno i optimalna DNF). Isto vrijedi i za MKNF: ona je istovjetna sa SKNF, koja glasi $Y = M_0 M_3 M_5 M_6 M_9 M_{10} M_{12} M_{15}$. Stoga je SDNF (ili SKNF) ujedno i najekonomičniji oblik funkcije za realizaciju pomoću dvonivoske mreže sa AND, OR i NOT logičkim kolima, i glasi

$$Y = \overline{A}\,\overline{B}\,\overline{C}D \vee \overline{A}\,\overline{B}C\overline{D} \vee \overline{A}B\overline{C}\,\overline{D} \vee \overline{A}BCD \vee A\overline{B}\,\overline{C}\,\overline{D} \vee A\overline{B}\,CD \vee AB\overline{C}D \vee ABC\overline{D}$$

Ipak, dozvolimo li upotrebu ekskluzivne disjunkcije, funkcija se može drastično pojednostaviti i svesti na oblik

$Y = A \oplus B \oplus C \oplus D$

Do ovog oblika može se lako doći intuicijom, posmatrajući samu formulaciju problema, a nije ga teško izvesti i formalnom primjenom pravila logičke algebre:

$$Y = \overline{A}\,\overline{B}\,\overline{C}D \vee \overline{A}\,\overline{B}C\overline{D} \vee \overline{A}B\overline{C}\,\overline{D} \vee \overline{A}BCD \vee A\overline{B}\,\overline{C}\,\overline{D} \vee A\overline{B}\,CD \vee AB\overline{C}D \vee ABC\overline{D} =$$
$$= \overline{A}\,\overline{B}(\overline{C}D \vee C\overline{D}) \vee \overline{A}B(\overline{C}\,\overline{D} \vee CD) \vee A\overline{B}(\overline{C}\,\overline{D} \vee CD) \vee AB(\overline{C}D \vee C\overline{D}) =$$
$$= (\overline{A}\,\overline{B} \vee AB)(\overline{C}D \vee C\overline{D}) \vee (\overline{A}B \vee A\overline{B})(\overline{C}\,\overline{D} \vee CD) = \overline{A \oplus B}(C \oplus D) \vee (A \oplus B)\overline{C \oplus D} =$$
$$= (A \oplus B) \oplus (C \oplus D) = A \oplus B \oplus C \oplus D$$

10.9 MDNF: $Y = \overline{A}\,\overline{B}CD \vee \overline{A}BC\overline{D} \vee \overline{A}B\overline{C}D \vee A\overline{B}\,\overline{C}D \vee A\overline{B}C\overline{D} \vee AB\overline{C}\,\overline{D}$

MKNF: $Y = (A \vee B \vee C)(A \vee B \vee D)(A \vee C \vee D)(B \vee C \vee D)(\overline{A} \vee \overline{B} \vee \overline{D})(\overline{A} \vee \overline{B} \vee \overline{C})(\overline{A} \vee \overline{C} \vee \overline{D})(\overline{B} \vee \overline{C} \vee \overline{D})$

MDNF za ovaj slučaj je također jednaka SDNF. MKNF je prostoja od SKNF, ali je ipak manje povoljna za realizaciju od MDNF. Međutim, uz upotrebu ekskluzivne disjunkcije, mogu se izvesti znatno kraći oblici funkcije. Ovdje su prikazana tri međusobno ekvivalentna oblika koji su optimalni sa aspekta broja ulaza:

$Y = (A \oplus B)(C \oplus D) \vee (A \oplus D)(B \oplus C)$

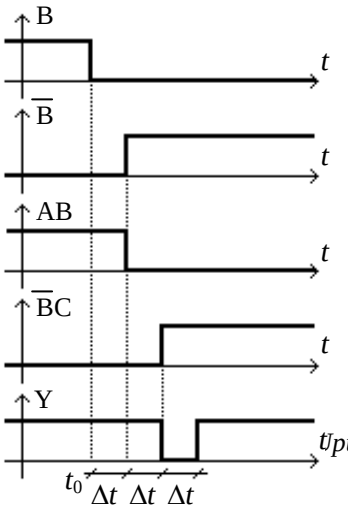
$Y = (A \oplus B)(C \oplus D) \vee (A \oplus C)(B \oplus D)$

$Y = (A \oplus C)(B \oplus D) \vee (A \oplus D)(B \oplus C)$

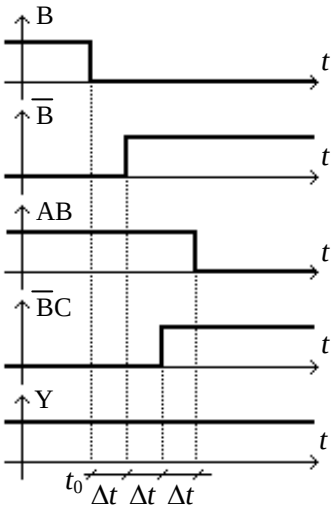
Da se izvede bilo koji od ovih oblika, treba udvojiti dva člana u MDNF obliku po pravilu $X = X \vee X$, a zatim pogodno grupirati članove.

10.12 Prikazane slike ilustriraju stanje u pojedinim tačkama mreže za oba razmatrana slučaja:

Sva kola imaju isto kašnjenje



Kolo koje realizira član AB *ima trostruko veće kašnjenje*



Lako se vidi da do lažnog impulsa neće doći ukoliko AND kolo koje realizira član AB ima kašnjenje koje je veće ili jednako zbiru kašnjenja invertora i drugog AND kola.

10.13 a) $Y = \overline{A}B \vee \overline{A}C \vee B\overline{C}$ b) $Y = A\overline{B} \vee AC \vee BC$

10.14 a) $Y = AB\overline{D} \vee \overline{A}\,\overline{C}\,\overline{D} \vee A\overline{C}D \vee BC\overline{D} \vee B\overline{C}D \vee AB$

Jedna od implikanti $A\overline{C}D$ odnosno $B\overline{C}D$ je suvišna sa aspekta optimalnosti, a dodana je radi uklanjanja rizika.

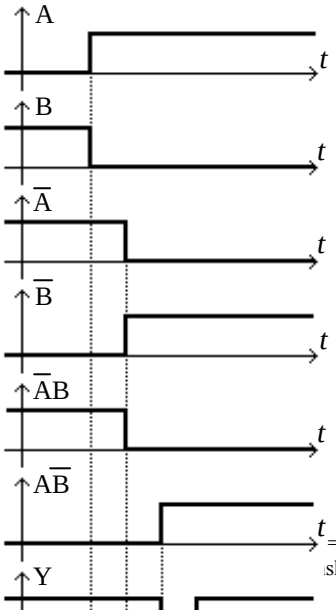
b) $Y = AB \vee \overline{A}D \vee \overline{B}\,\overline{C} \vee \overline{C}D$

Radi uklanjanja rizika, dodana je implikanta $\overline{C}D$.

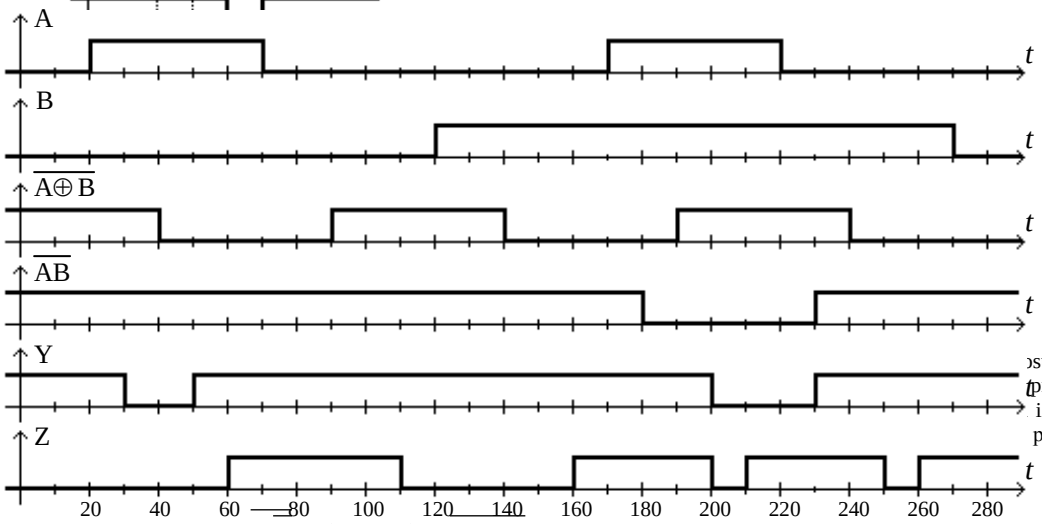
10.15 $Y = \overline{A}\,\overline{B}\,\overline{C}E \vee \overline{A}DE \vee \overline{B}\,\overline{C}\,\overline{D} \vee \overline{B}\,\overline{D}\,\overline{E} \vee C\overline{D}\,\overline{E}$

Radi uklanjanja rizika, dodane su implikante $\overline{A}\,\overline{B}\,\overline{C}E$ i $\overline{B}\,\overline{D}\,\overline{E}$.

10.16 Sa sljedećih vremenskih dijagrama se jasno vidi da se uz navedeni scenario na izlazu javlja lažna vrijednost Y = 0 u vremenskom intervalu od $t = t_0 + 2\Delta t$ do $t = t_0 + 3\Delta t$:



10.17 Mreža bi t = $\overline{AB}$ i Z = A ∨ B. Za datu mrežu, uz zadane pretpostavke i uz zadane oblike ulaznih sig ski dijagrami:



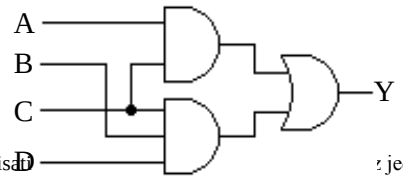
Zav su svoju
oče promjeni
vrij izlazu Z
nak pri B = 1
dol

10.18 a) Izvorna logička funkcija: $Y = AB \vee (A \oplus C) \vee A \oplus C\,A \vee ABD$

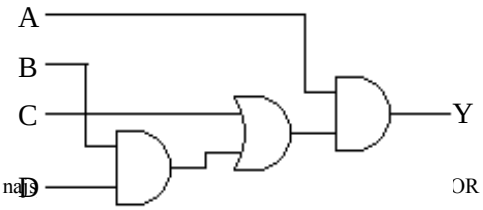
Optimalna mreža u 2 nivoa slijedi iz MDNF ili MKNF:

MDNF: $Y = AC \vee ABD$, MKNF: $Y = A(B \vee C)(C \vee D)$

Obje varijante su podjednako povoljne za realizaciju (7 ulaza). Prikazana je realizacija na osnovu MDNF:



b) Možemo pisati [illegible] jedan nivo više):



c) 50 ns (duž najsporije putanje su 3 invertora, 2 OR logička kola, i jedno AND logičko kolo)

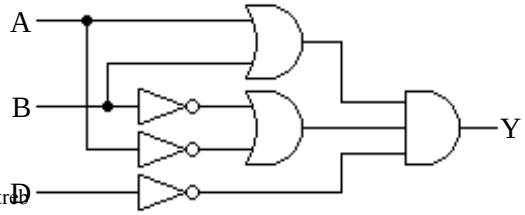
d) 20 ns za mrežu pod a), 30 ns za mrežu pod b) (koja nije optimalna sa aspekta brzine)

10.19 a) Izvorna logička funkcija: $Y = \overline{\overline{AB} \vee A \oplus \overline{B}\,\overline{D} \vee AC\overline{B \vee D}}$

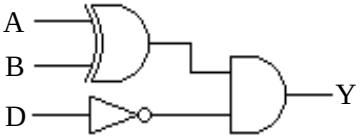
Optimalna mreža u 2 nivoa slijedi iz MDNF ili MKNF:

MDNF: $Y = \overline{A}B\overline{D} \vee A\overline{B}\,\overline{D}$, MKNF: $Y = (A \vee B)(\overline{A} \vee \overline{B})\,\overline{D}$

Realizacija na osnovu MKNF je povoljnija (1 ulaz manje):



b) Uz upotrebu

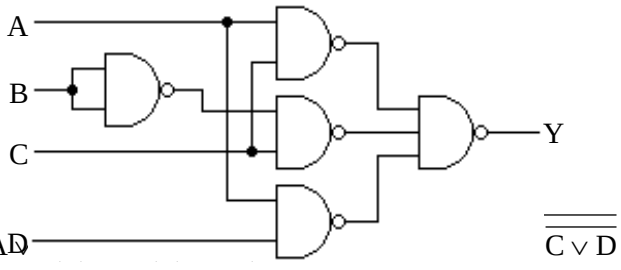


c) 60 ns (duž najsporije putanje su 3 invertora, 2 OR logička kola, i jedno AND logičko kolo)

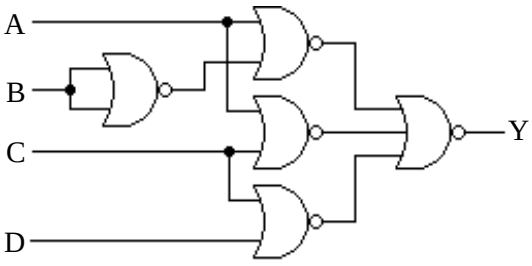
d) 65 ns (duž najsporije putanje su po jedno NOT, EXNOR, NAND, NOR i OR logičko kolo)

e) Uz uvjete date pod c), 30 ns za mrežu pod a), 20 ns za mrežu pod b), iz čega slijedi da je pod navedenim uvjetima mreža pod b) ne samo jednostavnija, nego i brža. Uz uvjete navedene pod d), 30 ns za mrežu pod a), 35 ns za mrežu pod b). Vidimo da odluka da li je brža mreža pod a) ili b) zavisi od odnosa kašnjenja koje unose pojedine vrste logičkih kola!

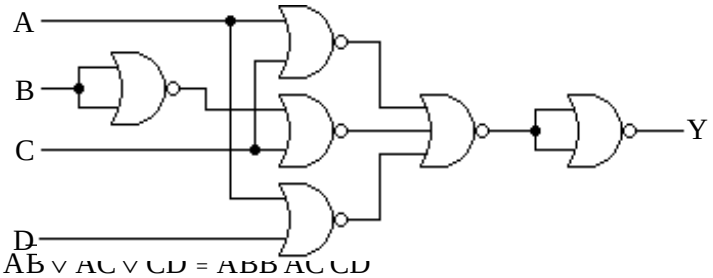
10.20 a) $Y = AC \vee AD \vee \overline{B}C = \overline{\overline{AC}\;\overline{AD}\;\overline{\overline{B}C}}$



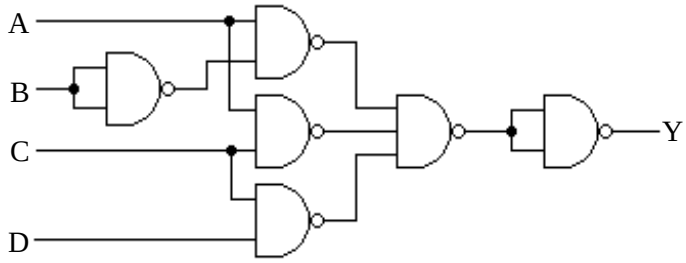
b) $Y = (AD \vee \overline{\overline{C \vee D}}$



c) $\overline{Y} = (A \vee C)(A \vee D)(\overline{B} \vee C) = \overline{\overline{A \vee C} \vee \overline{A \vee D} \vee \overline{B \vee B} \vee C}$



d) $\overline{Y} = A\overline{B} \vee AC \vee CD = \overline{A\overline{B}\,\overline{AC}\,\overline{CD}}$



10.21 a) NAND: $Y = \overline{A\overline{B}B\overline{C}C\,\overline{A}A\overline{C}\,\overline{A}A\overline{B}}$ NOR: $Y = \overline{\overline{A \vee B \vee C} \vee \overline{A \vee A \vee B} \vee \overline{B \vee A \vee A} \vee \overline{C \vee C}}$

b) NAND: $Y = \overline{A\overline{C}C\,\overline{A}A\,\overline{B}\overline{B}C}$ NOR: $Y = \overline{\overline{A \vee B \vee B} \vee \overline{A \vee A \vee C} \vee \overline{C \vee C} \vee \overline{A \vee C}}$

c) NAND: $Y = \overline{\overline{A}A\,\overline{B}B\,CD}$ NOR: $Y = \overline{\overline{A \vee B \vee C} \vee \overline{C \vee D} \vee \overline{D \vee A} \vee \overline{B \vee C} \vee \overline{C \vee D} \vee \overline{D}}$

d) NAND: $Y = \overline{A\,\overline{B}B\overline{C}C}$ NOR: $Y = \overline{\overline{A \vee A} \vee \overline{B \vee B} \vee \overline{A \vee A} \vee \overline{C \vee C}}$

e) NAND: $Y = \overline{\overline{A}AC\,A\overline{B}B}$ NOR: $Y = \overline{\overline{A \vee A} \vee \overline{B \vee B} \vee \overline{A \vee C}}$

f) NAND: $Y = \overline{A\overline{C}\,\overline{A}AB}$ NOR: $Y = \overline{\overline{A \vee B} \vee \overline{A \vee A} \vee C}$

g) NAND: $Y = \overline{\overline{A}A\,\overline{B}B\,A\overline{C}C}$ NOR: $Y = \overline{\overline{A \vee B} \vee \overline{B \vee A} \vee \overline{A \vee C} \vee \overline{C}}$

h) NAND: $Y = 0$ NOR: $Y = 0$

i) NAND: $Y = \overline{\overline{A}AB\,A\overline{C}CD}$ NOR: $Y = \overline{\overline{A \vee B} \vee \overline{A \vee A} \vee \overline{C \vee C} \vee \overline{A \vee A} \vee D}$

j) NAND: $Y = \overline{A\overline{B}B\overline{C}C\,\overline{A}A\overline{B}C}$ NOR: $Y = \overline{\overline{A \vee B} \vee \overline{B \vee B} \vee \overline{C \vee A} \vee \overline{A \vee C} \vee \overline{C}}$

10.22 a) NAND: $Y = \overline{\overline{A}AB\overline{D}D\,\overline{A}A\,\overline{B}B\overline{D}\,C}$

NOR: $Y = \overline{\overline{A \vee A} \vee \overline{C \vee C} \vee \overline{B \vee B} \vee \overline{C \vee C} \vee \overline{D \vee D} \vee \overline{B \vee C} \vee \overline{C \vee D}}$

b) NAND: $Y = \overline{\overline{B}B\overline{D}D\,\overline{B}D\,A\,\overline{C}C}$

NOR: $Y = \overline{\overline{A \vee A} \vee \overline{B \vee B} \vee \overline{C \vee D} \vee \overline{A \vee A} \vee \overline{B \vee C} \vee \overline{D \vee D}}$

c) NAND: $Y = \overline{B\overline{C}C\overline{D}\,B\overline{B}C\overline{C}\,D\overline{D}\,A}$

NOR: $Y = \overline{\overline{A \vee A} \vee \overline{C \vee C} \vee \overline{A \vee A} \vee \overline{B \vee B} \vee \overline{D} \vee \overline{A \vee A} \vee \overline{B \vee D} \vee \overline{D}}$

d) NAND: $Y = \overline{B\overline{D}D\,B\overline{B}D\,\overline{A}A\,C}$

NOR: $Y = \overline{A \vee \overline{B} \vee \overline{B} \vee \overline{C} \vee \overline{C} \vee \overline{D} \vee \overline{D} \vee \overline{A \vee \overline{B} \vee \overline{C} \vee \overline{C} \vee \overline{D}}}$

10.23 a) NAND: $Y = \overline{\overline{ABC}\;\overline{AAD}\;\overline{CCD}}$ NOR: $Y = \overline{\overline{A \vee D} \vee \overline{C \vee D} \vee \overline{A \vee A} \vee \overline{B \vee C} \vee C}$

b) NAND: $Y = \overline{\overline{DD}}$ NOR: $Y = \overline{\overline{D \vee D}}$

c) NAND: $Y = \overline{\overline{A\overline{BB}\,\overline{CC}}\;\overline{\overline{AA}CD}\;\overline{\overline{BB}D}}$ NOR: $Y = \overline{\overline{\overline{A \vee A} \vee \overline{B \vee B} \vee \overline{B \vee B} \vee C} \vee \overline{\overline{A \vee D} \vee \overline{C \vee C} \vee D}}$

d) NAND: $Y = \overline{\overline{A\overline{BB}D}\;\overline{\overline{BB}\,\overline{CC}}\;D}$ NOR: $Y = \overline{B \vee \overline{D \vee D} \vee \overline{A \vee C \vee C}}$

10.24 $Y = \overline{\overline{A \vee C \vee C} \vee \overline{B \vee B} \vee \overline{A \vee A} \vee C \vee D}$

Uputa: MKNF date funkcije se može optimizirati na oblik $Y = (\overline{A}C \vee \overline{B})(A\overline{C} \vee D)$. Zatim transformirajte tako optimizirani oblik u oblik pogodan za realizaciju pomoću NOR kola višestrukom primjenom pravila dvojne negacije i De Morganovih pravila.

10.25 a) NAND: $Y = \overline{A\overline{BB}C\;\overline{AA}\,\overline{BB}\,\overline{CC}\;A\overline{CC}D\;\overline{AA}CD}$

NOR: $Y = \overline{\overline{A \vee A} \vee \overline{B \vee B} \vee \overline{C \vee C} \vee \overline{A \vee \overline{B \vee B} \vee C} \vee \overline{A \vee \overline{C \vee C} \vee D} \vee \overline{\overline{A \vee A} \vee C \vee D}}$

b) NAND: $Y = \overline{AC\overline{DD}\;A\overline{CC}D\;B\overline{CC}\;\overline{DD}\;B\overline{CD}}$

NOR: $Y = \overline{A \vee \overline{C \vee C} \vee D \vee \overline{A \vee C \vee \overline{D \vee D}} \vee \overline{B \vee \overline{C \vee C} \vee \overline{D \vee D}} \vee \overline{B \vee C \vee D}}$

11.1 a) $Y = \overline{A}B\overline{D} \vee ACD \vee \overline{B}CD \vee A\overline{B}$

b) $Y = \overline{A}\,\overline{B} \vee \overline{B}\,\overline{C}$

c) $Y = A\overline{B}C \vee \overline{A}\,\overline{C} \vee BD$ ili $Y = ACD \vee \overline{A}\,\overline{C} \vee BD$

d) $Y = A\overline{B}\,\overline{C} \vee BD \vee \overline{C}D$

11.2 MDNF: $Y = \overline{A}\,\overline{C}D \vee A\overline{D} \vee \overline{B}\,\overline{C} \vee C\overline{D}$ ili $Y = \overline{A}\,\overline{C}D \vee A\overline{B} \vee A\overline{D} \vee C\overline{D}$

MKNF: $Y = (\overline{A} \vee \overline{B} \vee \overline{D})(A \vee C \vee D)(\overline{C} \vee \overline{D})$

11.3 a) $Y = ABC \vee \overline{A}\,\overline{C} \vee \overline{A}D \vee BD$

b) $Y = (\overline{A} \vee B)(A \vee D)(C \vee D)$

c) $Y = ABC \vee \overline{A}\,\overline{B} \vee BD$

11.4 a) $Y = A\overline{B}D \vee A\overline{B}\,\overline{C} \vee \overline{A}B\overline{C} \vee \overline{A}BE \vee \overline{B}\,\overline{C}D$

b) $Y = \overline{A}B\overline{D}E \vee \overline{B}\,\overline{C}\,\overline{E} \vee \overline{C}\,\overline{D}$

11.5 a) MDNF: $Y = \overline{A}\,\overline{B}\,\overline{D} \vee \overline{A}BE \vee \overline{B}C\overline{E} \vee BDE \vee CDE$

MKNF: $Y = (\overline{A} \vee B \vee C)(\overline{A} \vee D \vee \overline{E})(B \vee C \vee \overline{D})(\overline{B} \vee E)$ ili

$Y = (\overline{A} \vee C \vee E)(\overline{A} \vee D \vee \overline{E})(B \vee C \vee \overline{D})(\overline{B} \vee E)$

b) MDNF: $Y = \overline{B}C \vee \overline{B}\,\overline{D} \vee BE$

MKNF: $Y = (\overline{B} \vee E)(B \vee C \vee \overline{D})$

Vidimo da korištenjem činjenice da je logička funkcija nepotpuno definirana možemo ostvariti velike uštede u realizaciji.

11.6 a) $X = \overline{A}\,\overline{B}C \vee D$ $Y = \overline{A}\,\overline{B}\,\overline{C} \vee AD$ $Z = \overline{A}\,\overline{B} \vee \overline{A}D$

b) $X = \overline{A}\,\overline{B}C \vee D$ $Y = \overline{A}\,\overline{B}\,\overline{C} \vee AD$ $Z = \overline{A}\,\overline{B}\,\overline{C} \vee \overline{A}\,\overline{B}C \vee \overline{A}D$

U slučaju pod a) potreban je 21 ulaz (invertori se mogu zajednički iskoristiti u sve tri funkcije). U slučaju pod b) potrebno je 20 ulaza. Naime, iako funkcija Z nije optimalna, u njoj se mogu iskoristiti članovi koji su već iskorišteni za realizaciju funkcija X i Y. Primijetimo također da se implikantu D koja se javlja u funkciji X ne isplati razbijati na dvije implikante koje se javljaju u funkcijama Y i Z.

Napomena: Ni varijanta pod b) nije najekonomičnija. Još ekonomičniju varijantu (19 ulaza) dobijamo ukoliko funkciju Z predstavimo u MKNF obliku.

11.7 MDNF, neovisno posmatrano:

$X = \overline{A}BD \vee BC$ $Y = \overline{A}BD \vee CD$ $Z = ABC \vee \overline{A}BD \vee CD$

MKNF, neovisno posmatrano:

$X = (\overline{A} \vee C)(C \vee D)B$ $Y = (\overline{A} \vee C)(B \vee C)D$ $Z = (\overline{A} \vee C)(A \vee D)(B \vee C)(B \vee D)$

Za realizaciju na osnovu MDNF, potrebno je 18 ulaza, s obzirom na postojanje zajedničkih članova, a za realizaciju na osnovu MKNF, potreban je 21 ulaz, također zbog postojanja zajedničkih faktora. DNF i KNF oblici koji bi omogućili veći broj zajedničkih članova odnosno faktora ne postoje, ali se jedan ulaz može uštedjeti ukoliko se primijeti da je $Z = ABC \vee Y$ (ipak, mreža time postaje tronivoska).

11.8 MDNF (uz $CD \neq 1$), neovisno posmatrano:

$X = A\overline{C}\,\overline{D} \vee \overline{A}C \vee BD$ $Y = AC \vee BD$ $Z = A\overline{D} \vee C$

MKNF (uz $CD \neq 1$), neovisno posmatrano:

$X = (A \vee C \vee D)(\overline{A} \vee \overline{C})(B \vee \overline{D})$ $Y = (A \vee D)(B \vee \overline{D})(C \vee D)$ $Z = (A \vee C)\overline{D}$

Postoji još nekoliko jednako ekonomičnih MKNF oblika za funkciji Y, ali je izabran oblik koji ima zajednički faktor kao u funkciji X, što omogućava realizaciju sa 24 ulaza. Također, zbog postojanja zajedničkog člana BD, funkcije zasnovane na MDNF mogu se realizirati sa 21 ulazom. Ekonomičnija realizacija na bazi DNF dobija se ukoliko Z napišemo u obliku

$Z = A\overline{C}\,\overline{D} \vee C$

s obzirom da tako možemo uštediti jedan ulaz, zbog postojanja zajedničkog člana kao u funkciji X (s druge strane, ne isplati se razbijati implikantu C na dva člana koji postoje u funkcijama X i Y). Iz sličnih razloga, ekonomičniju realizaciju nabazi KNF dobijamo ukoliko Z napišemo kao

$Z = (A \vee C \vee D)\,\overline{D}$

11.9 MDNF (uz $BC \neq 1$), neovisno posmatrano:

$X = AB \vee \overline{A}D$ $\quad Y = \overline{A}C \vee A\overline{D}$ $\quad Z = A\overline{C}\,\overline{D} \vee AB \vee \overline{A}C \vee \overline{A}D$

MKNF (uz $BC \neq 1$), neovisno posmatrano:

$X = (\overline{A} \vee B)(A \vee D)$ $\quad Y = (A \vee C)(\overline{A} \vee \overline{D})$ $\quad Z = (\overline{A} \vee B \vee \overline{D})(A \vee C \vee D)(\overline{A} \vee \overline{C})$

Realizacija preko MDNF zahtijeva 21 ulaz (zbog postojanja zajedničkih članova), a jedan ulaz se može prišteditи ukoliko se uoči da se čitava funkcija X sadrži u izrazu za funkciju Z (mreža tada postaje tronivoska). DNF oblike sa većim brojem zajedničkih članova nije moguće naći. Realizacija preko MKNF zahtijeva 26 ulaza, s obzirom da nema zajedničkih faktora. Dva ulaza je moguće uštediti ukoliko se X i Y napišu u sljedećim, znatno složenijim oblicima, ali u kojima se pojavljuju identični faktori koji se javljaju i u Z:

$X = (\overline{A} \vee B \vee \overline{D})(A \vee C \vee D)(B \vee D)$ $\quad Y = (\overline{A} \vee B \vee \overline{D})(A \vee C \vee D)(C \vee \overline{D})$

Još jedan ulaz možemo uštediti (po cijenu povećanja broja nivoa mreže) ukoliko prvo realiziramo pomoćnu funkciju

$W = (\overline{A} \vee B \vee \overline{D})(A \vee C \vee D)$

a zatim uzmemo

$X = (B \vee D)\,W$ $\quad Y = (C \vee \overline{D})\,W$ $\quad Z = (\overline{A} \vee \overline{C})\,W$

11.10 Uz $X = [x_2;x_1;x_0]$ i $Y = [y_2;y_1;y_0]$ redovi tablice istine koji odgovaraju Y redom glase 001, 010, 101, 011, 011, 101, 010 i 001, tako da dobijamo:

$y_2 = x_2\overline{x}_1x_0 \vee \overline{x}_2x_1\overline{x}_0$ $\quad y_1 = x_2\overline{x}_0 \vee \overline{x}_2x_0$ $\quad y_0 = \overline{x}_2x_1x_0 \vee x_2\overline{x}_1 \vee x_1x_0$

Član $\overline{x}_2x_1x_0$ nije optimalan u y_0 posmatrano samostalno, ali se javlja kao zajednički član u y_2.

11.11 Uz $X = [x_2;x_1;x_0]$ i $Y = [y_3;y_2;y_1;y_0]$ redovi tablice istine koji odgovaraju Y redom glase 0001, 0101, 0000, 0010, 1010, 1011, 0101 i 0101, tako da dobijamo:

$y_3 = x_2\overline{x}_1$ $\quad y_2 = \overline{x}_2\overline{x}_1x_0 \vee x_2x_1$ $\quad y_1 = \overline{x}_2x_1x_0 \vee x_2\overline{x}_1$

$y_0 = x_2x_1 \vee \overline{x}_2\overline{x}_1 \vee x_2x_0$ $\quad$ (ili $y_0 = x_2x_1 \vee \overline{x}_2\overline{x}_1 \vee \overline{x}_1x_0$)

11.12 Uz $X = [x_3;x_2;x_1;x_0]$ i $Y = [y_3;y_2;y_1;y_0]$, DNF oblici koji omogućavaju optimalnu realizaciju glase

$y_3 = \overline{x}_3x_2x_1x_0 \vee x_3\overline{x}_2 \vee x_3\overline{x}_1 \vee x_3\overline{x}_0$ $\quad y_2 = \overline{x}_2x_1x_0 \vee x_2\overline{x}_1 \vee x_2\overline{x}_0$

$y_1 = \overline{x}_1x_0 \vee x_1\overline{x}_0$ $\quad y_0 = \overline{x}_0$

a KNF oblici koji omogućavaju optimalnu realizaciju glase

$y_3 = (\overline{x}_3 \vee \overline{x}_2 \vee \overline{x}_1 \vee \overline{x}_0)(x_3 \vee x_2)(x_3 \vee x_1)(x_3 \vee x_0)$ $\quad y_2 = (\overline{x}_2 \vee \overline{x}_1 \vee \overline{x}_0)(x_2 \vee x_1)(x_2 \vee x_0)$

$y_1 = (\overline{x}_1 \vee \overline{x}_0)(x_1 \vee x_0)$ $\quad y_0 = \overline{x}_0$

Ipak, najjednostavnije oblike možemo dobiti ukoliko dozvolimo i upotrebu ekskluzivne disjunkcije, što se lako izvodi elementarnim transformacijama:

$y_3 = x_3 \oplus x_2x_1x_0$ $\quad y_2 = x_2 \oplus x_1x_0$ $\quad y_1 = x_1 \oplus x_0$ $\quad y_0 = \overline{x}_0$

11.13 Uz $X = [x_3;x_2;x_1;x_0]$ i $Y = [y_3;y_2;y_1;y_0]$, i uzmajući u obzir nepotpunu definiranost odgovarajućih funkcija, DNF oblici koji omogućavaju optimalnu realizaciju glase

$y_3 = x_2x_1x_0 \vee x_3\overline{x}_0$ $\quad y_2 = \overline{x}_2x_1x_0 \vee x_2\overline{x}_1 \vee x_2\overline{x}_0$

$y_1 = \overline{x}_3\overline{x}_1x_0 \vee x_1\overline{x}_0$ $\quad y_0 = \overline{x}_0$

$C = x_3x_0$

a KNF oblici (uz iste pretpostavke) koji omogućavaju optimalnu realizaciju glase

$y_3 = (x_3 \vee x_0)(x_2 \vee \overline{x}_0)(x_1 \vee \overline{x}_0)$ $\quad y_2 = (\overline{x}_2 \vee \overline{x}_1 \vee \overline{x}_0)(x_2 \vee x_1)(x_2 \vee x_0)$

$y_1 = (\overline{x}_1 \vee \overline{x}_0)(x_1 \vee x_0)\overline{x}_3$ $\quad y_0 = \overline{x}_0$

$C = x_3x_0$

Uz upotrebu ekskluzivne disjunkcije, moguće je pojednostaviti y_2 i y_1, na sljedeći način:

$y_2 = x_2 \oplus x_1x_0$ $\quad y_1 = (x_1 \oplus x_0)\overline{x}_3$

Navedeni oblik za y_1 slijedi direktno iz MKNF, ili se može dobiti ukoliko se iz Veitchovog dijagrama očita sljedeći DNF, oblik koji nije optimalan:

$y_1 = \overline{x}_3\overline{x}_1x_0 \vee \overline{x}_3x_1\overline{x}_0$

Ipak, prikazani oblici su ekvivalentni samo uz pretpostavku o nepotpunoj definiranosti funkcija.

11.14 $O_1 = \overline{\text{SHIFT}} \cdot I_1 \vee \text{SHIFT} \cdot I_0$ $\quad O_0 = \overline{\text{SHIFT}} \cdot I_0$

11.15 $O_1 = \overline{\text{SWAP}} \cdot I_1 \vee \text{SWAP} \cdot I_0$ $\quad O_0 = \text{SWAP} \cdot I_1 \vee \overline{\text{SWAP}} \cdot I_0$

11.16 Blok za šifriranje očito obavlja transformacije $000 \to 001$, $001 \to 110$, $010 \to 011$, $011 \to 000$, $100 \to 101$, $101 \to 010$, $110 \to 111$ i $111 \to 100$, tako da uz $X = [x_2;x_1;x_0]$ i $Y = [y_2;y_1;y_0]$ imamo

$y_2 = \overline{x}_2\overline{x}_1x_0 \vee x_2x_1 \vee x_2\overline{x}_0$ $\quad y_1 = x_1\overline{x}_0 \vee \overline{x}_1x_0$ $\quad y_0 = \overline{x}_0$

Blok za dešifriranje treba da obavlja inverzne transformacije $001 \to 000$, $110 \to 001$, $011 \to 010$, $000 \to 011$, $101 \to 100$, $010 \to 101$, $111 \to 110$ i $100 \to 111$, odnosno, drugim redoslijedom napisano, transformacije $000 \to 011$, $001 \to 000$, $010 \to 101$, $011 \to 010$, $100 \to 111$, $101 \to 100$, $110 \to 001$ i $111 \to 110$, što je dovoljno za opis njegove funkcionalnosti, bez potrebe da se zna analitički izraz za X izražen preko Y (usput, nije teško provjeriti da on glasi $X = (5Y + 3) \bmod 8$). Odavde, za blok za dešifriranje dobijamo:

$x_2 = \overline{y}_2y_1\overline{y}_0 \vee y_2y_0 \vee y_2\overline{y}_1$ $\quad x_1 = y_1y_0 \vee \overline{y}_1\overline{y}_0$ $\quad x_0 = \overline{y}_0$

11.17 Uputa: Neka je prvo $m > n$, i neka je $A = 2^m - 1$ i $B = 2^n - 1$. Tada je $A + B = 2^m + 2^n - 1 < 2^{m+1} - 1$, pa je m bita dovoljno. Analogno za $m < n$. Za $m = n$ imamo $A + B = 2^{m+1} - 2 > 2^m - 1$, pa m bita nije dovoljno, ali je $m + 1$ bita dovoljno.

11.18 Uz $A = [a_1;a_0]$, $B = [b_1;b_0]$ i $C = [c_2;c_1;c_0]$, redovi tablice istine koji odgovaraju C redom glase 000, 111, 110, 101, 001, 000, 111, 110, 010, 001, 000, 111, 011, 010, 001 i 000, tako da dobijamo:

$c_2 = \overline{a}_0b_1b_0 \vee \overline{a}_1\overline{a}_0b_0 \vee \overline{a}_1b_1$

$c_1 = a_1\overline{a}_0b_1b_0 \vee \overline{a}_1\overline{a}_0\overline{b}_1b_0 \vee a_1a_0\overline{b}_1 \vee \overline{a}_1a_0b_1 \vee a_1\overline{b}_1\overline{b}_0 \vee \overline{a}_1b_1\overline{b}_0$

$c_0 = a_0\overline{b}_0 \vee \overline{a}_0b_0$

Ukoliko dozvolimo upotrebu mreža sa više nivoa i EXOR logička kola, možemo dobiti:

$c_2 = \overline{a}_0b_0(a_1 \vee b_1) \vee \overline{a}_1b_1$ $\quad c_1 = \overline{a}_0b_0 \oplus a_1 \oplus b_1$ $\quad c_0 = a_0 \oplus b_0$

Uputa: Za formiranje izraza za c_1 postupite slično kao u Primjeru 11.3.

11.19 Ovaj put su dva bita za predstavljanje izlaza dovoljna, tako da uz $A = [a_1;a_0]$, $B = [b_1;b_0]$ i $C = [c_1;c_0]$ dobijamo:

$c_1 = a_1a_0\overline{b}_1 \vee a_1\overline{b}_1\overline{b}_0 = a_1\overline{b}_1(a_0 \vee \overline{b}_0)$ $\quad c_0 = a_0\overline{b}_0 \vee \overline{a}_0b_0 = a_0 \oplus b_0$

Vidimo da pretpostavka $A > B$ drastično pojednostavljuje sklop.

11.20 Ukoliko usvojimo da je sklop realiziran na osnovu funkcija kakve su prikazane u rješenju prethodnog zadatka, dobijamo da se uz pretpostavku $A < B$ sklop ponaša kao da vrijede pravila $0-1=1$, $0-2=0$, $0-3=1$, $1-2=1$, $1-3=0$ i $2-3=1$. Drugim riječima, umjesto rezultata –1 ili –3 pojavljuje se 1, a umjesto rezultata –2 pojavljuje se 0.

11.21 Slično kao Zadatku 11.19, ali ovaj put uz potpuno definirane funkcije, za optimalne DNF oblike dobijamo:

$c_1 = a_1 a_0 \overline{b}_1 \vee a_1 \overline{b}_1 \overline{b}_0$

$c_0 = a_1 \overline{a}_0 \overline{b}_1 b_0 \vee a_1 a_0 \overline{b}_0 \vee a_0 \overline{b}_1 \overline{b}_0$

a za optimalne KNF oblike:

$c_1 = a_1 \overline{b}_1 (a_0 \vee \overline{b}_0)$

$c_0 = (a_1 \vee a_0)(a_1 \vee \overline{b}_1)(a_0 \vee \overline{b}_1)(a_0 \vee b_0)(\overline{a}_0 \vee \overline{b}_0)$ (pored ovog, postoje još 3 MKNF oblika)

Najekonomičniju realizaciju dobijamo ako uzmemo MKNF oblik za c_1, a MDNF oblik za c_0 preuredimo u oblik

$c_0 = a_1 \overline{a}_0 \overline{b}_1 b_0 \vee a_0 \overline{b}_0 (a_1 \vee \overline{b}_1)$

11.22 Za formiranje izlaza C potrebna su 3 bita. Uz $A = [a_1;a_0]$, $B = [b_1;b_0]$ i $C = [c_2;c_1;c_0]$, redovi tablice istine koji odgovaraju C redom glase 101, 110, 111, 000, 110, 111, 000, 001, 111, 000, 001, 010, 000, 001, 010 i 011, tako da dobijamo:

$c_2 = \overline{a}_0 \overline{b}_1 \overline{b}_0 \vee \overline{a}_1 \overline{a}_0 \overline{b}_0 \vee \overline{a}_1 \overline{b}_1$

$c_1 = a_1 \overline{a}_0 \overline{b}_1 \overline{b}_0 \vee \overline{a}_1 \overline{a}_0 b_1 \overline{b}_0 \vee a_1 a_0 b_1 \vee \overline{a}_1 a_0 \overline{b}_1 \vee a_1 b_1 b_0 \vee \overline{a}_1 \overline{b}_1 b_0$

$c_0 = a_0 b_0 \vee \overline{a}_0 \overline{b}_0$

Ukoliko dozvolimo upotrebu mreža sa više nivoa i EXOR logička kola, možemo dobiti:

$c_2 = \overline{a}_0 \overline{b}_0 (\overline{a}_1 \vee \overline{b}_1) \vee \overline{a}_1 \overline{b}_1$ $\quad c_1 = (a_0 \vee b_0) \oplus a_1 \oplus b_1$ $\quad c_0 = \overline{a_0 \oplus b_0}$

Uputa: Za formiranje izraza za c_1 postupite slično kao u Primjeru 11.3.

11.23 Za formiranje izlaza C potrebno je 5 bita. Uz $A = [a_1;a_0]$, $B = [b_1;b_0]$ i $C = [c_4;c_3;c_2;c_1;c_0]$, redovi tablice istine koji odgovaraju C redom glase 00000, 00001, 00100, 01001, 00001, 00010, 00101, 01010, 00100, 00101, 01000, 01101, 01001, 01010, 01101 i 10010, tako da dobijamo:

$c_4 = a_1 a_0 b_1 b_0$

$c_3 = a_1 a_0 \overline{b}_1 \vee a_1 \overline{a}_0 b_1 \vee a_1 b_1 \overline{b}_0 \vee \overline{a}_1 b_1 b_0$ (postoje još 2 podjednako ekonomične varijante)

$c_2 = a_1 \overline{a}_0 \overline{b}_1 \vee a_1 \overline{a}_0 b_0 \vee \overline{a}_1 b_1 \overline{b}_0 \vee a_0 b_1 \overline{b}_0$

$c_1 = a_0 b_0$

$c_0 = a_0 \overline{b}_0 \vee \overline{a}_0 b_0$

Dozvolimo li upotrebu EXOR logičkih kola, jednostavnim transformacijama možemo izvesti značajna pojednostavljenja za c_3, c_2 i c_0:

$c_3 = a_1 (a_0 \oplus b_1) \vee b_1 (a_1 \oplus b_0)$ $\quad c_2 = a_1 \overline{a}_0 \oplus b_1 \overline{b}_0$ $\quad c_0 = a_0 \oplus b_0$

Postoji još jedan interesantan izraz za c_3 izražen preko ekskluzivne disjunkcije, neznatno povoljniji kako sa aspekta ekonomičnosti, tako i sa aspekta brzine, ali njega je teže izvesti:

$c_3 = a_1 \overline{a}_0 b_1 \overline{b}_0 \oplus a_1 a_0 \oplus b_1 b_0$

11.24 $m + n$ bita.

11.25 Ako brojeve koji se množe označimo sa A i B, a njihov produkt sa C, onda su za formiranje izlaza C potrebna 4 bita. Uz $A = [a_1;a_0]$, $B = [b_1;b_0]$ i $C = [c_3;c_2;c_1;c_0]$, redovi tablice istine koji odgovaraju C redom glase 0000, 0000, 0000, 0000, 0000, 0001, 0010, 0011, 0000, 0010, 0100, 0110, 0000, 0011, 0110 i 1001, tako da dobijamo:

$c_3 = a_1a_0b_1b_0$

$c_2 = a_1b_1\bar{b}_0 \vee a_1\bar{a}_0b_1 = a_1b_1(\bar{a}_0 \vee \bar{b}_0)$ (MKNF je ovdje povoljnija)

$c_1 = \bar{a}_1a_0b_1 \vee a_1\bar{a}_0b_0 \vee a_1\bar{b}_1b_0 \vee a_0b_1\bar{b}_0$

$c_0 = a_0b_0$

Dozvolimo li upotrebu mreža u više nivoa, izraz za c_1 možemo napisati u obliku

$c_1 = a_1b_0(\bar{a}_0 \vee \bar{b}_1) \vee a_0b_1(\bar{a}_1 \vee \bar{b}_0)$

što se, uz upotrebu ekskluzivne disjunkcije, lako može prikazati u vrlo kompaktnom obliku

$c_1 = a_1b_0 \oplus a_0b_1$

Također, izraze za c_3 i c_2 možemo napisati kao

$c_3 = a_1b_1c_0$, $c_2 = a_1b_1\bar{c}_0$

Postoje i druge interesantne mogućnosti za c_1. Na primjer, ukoliko pođemo od MKNF oblika za c_1 (koji je manje povoljan od MDNF), ručno ga lako možemo transformirati u oblik

$c_1 = (\bar{a}_1 \vee \bar{a}_0 \vee \bar{b}_1 \vee \bar{b}_0)(a_1b_0 \vee a_0b_1)$

Ovaj oblik je interesantan jer se može napisati u obliku

$c_1 = \bar{c}_3(a_1b_0 \vee a_0b_1)$

što omogućava još jednostavniju realizaciju. Zanimljivo je da se ovaj oblik može očitati direktno iz Veitchovog dijagrama. Naime, kad bi u Veitchovom dijagramu za c_1 imali i mintermu m_{15}, MDNF za c_1 bi se dobila na osnovu dvije četvorke. Množeći takvu MDNF sa makstermom M_{15} (koja će poništiti nepostojeću mintermu m_{15}), dobijamo upravo dati oblik za c_1.

11.26 Uz $A = [a_1;a_0]$, $B = [b_1;b_0]$ i $C = [c_3;c_2;c_1;c_0]$, dobijamo:

$c_3 = a_1a_0b_0 \vee a_1b_1$

$c_2 = a_1\bar{a}_0\bar{b}_1 \vee a_1a_0\bar{b}_0 \vee a_0b_1$ ili $c_2 = a_1\bar{a}_0\bar{b}_1 \vee a_1\bar{b}_1\bar{b}_0 \vee a_0b_1$

$c_1 = \bar{a}_1a_0\bar{b}_1 \vee a_1\bar{a}_0b_0 \vee a_1b_1b_0 \vee a_0\bar{b}_1\bar{b}_0$

$c_0 = a_0b_0$

Uz upotrebu višenivoskih mreža, c_3 dopušta minorno pojednostavljenje kao $c_3 = a_1(a_0b_0 \vee b_1)$. Interesantno je da od dva jednako ekonomična MNDF oblika za c_2 prvi ne dopušta dalju optimizaciju (provjerite), dok se drugi oblik može napisati kompaktnije kao

$c_2 = a_1\bar{b}_1(\bar{a}_0 \vee \bar{b}_0) \vee a_0b_1$

Što se tiče c_1, slično kao u prethodnom zadatku možemo izvesti sljedeće interesantne oblike:

$c_1 = a_1b_0(\bar{a}_0 \vee b_1) \vee a_0\bar{b}_1(\bar{a}_1 \vee \bar{b}_0) = a_1b_0 \oplus a_0\bar{b}_1$

$c_1 = (\bar{a}_1 \vee \bar{a}_0 \vee b_1 \vee \bar{b}_0)(a_1b_0 \vee a_0\bar{b}_1)$

11.27 Za izlaze Q i R dovoljna su po dva bita. Uz $A = [a_1;a_0]$, $B = [b_1;b_0]$, $Q = [q_1;q_0]$ i $R = [r_1;r_0]$, dobijamo:

$q_1 = a_1\bar{b}_1$ $q_0 = a_1a_0 \vee a_1\bar{b}_0 \vee a_0\bar{b}_1$

$r_1 = a_1\bar{a}_0b_1b_0$ $r_0 = \bar{a}_1a_0b_1 \vee a_0\bar{b}_0$

11.28 Ukoliko usvojimo da je sklop realiziran na osnovu funkcija kakve su prikazane u rješenju prethodnog zadatka, dobijamo da se za $B = 0$ sklop ponaša kao da vrijede pravila $0/0 = 1$ bez ostatka, $1/0 = 1$ sa ostatkom 1, $2/0 = 3$ bez ostatka, i $3/0 = 3$ sa ostatkom 1.

11.29 U ovom slučaju su potrebna 3 bita za Q, tako da uz $A = [a_2;a_1;a_0]$, $B = [b_1;b_0]$, $Q = [q_2;q_1;q_0]$ i $R = [r_1;r_0]$, dobijamo:

$q_2 = a_2\bar{b}_1$ $q_1 = a_1\bar{b}_1 \vee a_2a_1 \vee a_2\bar{b}_0$

$q_0 = a_2\bar{a}_1b_1b_0 \vee \bar{a}_2a_1a_0 \vee a_1\bar{b}_0 \vee a_0\bar{b}_1$

$r_1 = \bar{a}_2a_1\bar{a}_0b_1b_0 \vee a_2\bar{a}_1a_0b_1b_0$ $r_0 = a_2\bar{a}_1\bar{a}_0b_1b_0 \vee \bar{a}_2\bar{a}_1a_0b_1 \vee a_2a_1a_0b_1 \vee a_0\bar{b}_0$

11.30 Ukoliko usvojimo da je sklop realiziran na osnovu funkcija kakve su prikazane u rješenju prethodnog zadatka, dobijamo da se za $B = 0$ sklop ponaša kao da vrijede pravila $0/0 = 0$ sa ostatkom 1, $1/0 = 1$ bez ostatka, $2/0 = 3$ sa ostatkom 1, i $3/0 = 3$ bez ostatka, $4/0 = 6$ sa ostatkom 1, $5/0 = 1$ bez ostatka, $6/0 = 7$ sa ostatkom 1, i $7/0 = 7$ bez ostatka.

11.31 Na osnovu formirane tablice istine, lako se izvodi:

$Y_{[=]} = \overline{a}_1\overline{a}_0\overline{b}_1\overline{b}_0 \vee \overline{a}_1 a_0 \overline{b}_1 b_0 \vee a_1\overline{a}_0 b_1\overline{b}_0 \vee a_1 a_0 b_1 b_0$

$Y_{[<]} = \overline{a}_1\overline{a}_0 b_0 \vee \overline{a}_0 b_1 b_0 \vee \overline{a}_1 b_1$

$Y_{[>]} = a_1 a_0 \overline{b}_0 \vee a_0\overline{b}_1\overline{b}_0 \vee a_1\overline{b}_1$

Dopustimo li upotrebu višenivoskih mreža i EXNOR kola, nije teško izvesti:

$Y_{[=]} = \overline{a_1 \oplus b_1} \cdot \overline{a_0 \oplus b_0}$

$Y_{[<]} = \overline{a}_0 b_0 (\overline{a}_1 \vee b_1) \vee \overline{a}_1 b_1$

$Y_{[>]} = a_0\overline{b}_0 (a_1 \vee \overline{b}_1) \vee a_1\overline{b}_1$

Ipak, najveću uštedu možemo ostvariti ukoliko uočimo da je $Y_{[=]} = \overline{Y_{[<]} \vee Y_{[>]}} = \overline{Y_{[<]}}\,\overline{Y_{[<]}}$, što nije lako izvesti analitički, ali je intuitivno očigledno iz same formulacije problema.

11.32 Realizacija sklopa neposredno slijedi iz jednačine $Y = \overline{c}_2 \overline{A} B \vee \overline{c}_2 A \overline{B} \vee \overline{c}_1 A B \vee \overline{c}_0 \overline{A}\,\overline{B}$.

11.33 Najekonomičnija realizacija sklopa na bazi DNF dobija se iz sljedećeg sistema jednačina:

$I_1 = x_2\overline{x}_1 x_0 \vee \overline{x}_2\overline{x}_0 \vee x_3 \vee x_1$ $\qquad$ $I_2 = x_2\overline{x}_1 x_0 \vee x_2\overline{x}_0 \vee \overline{x}_1\overline{x}_0 \vee x_3$

$I_3 = \overline{x}_1\overline{x}_0 \vee x_1 x_0 \vee \overline{x}_2$ $\qquad$ $I_4 = x_2\overline{x}_1 x_0 \vee \overline{x}_2 x_1 \vee x_2\overline{x}_0 \vee x_3$

$I_5 = \overline{x}_2\overline{x}_0 \vee x_1\overline{x}_0$ $\qquad$ $I_6 = x_2 \vee \overline{x}_1 \vee x_0$

$I_7 = x_2\overline{x}_1 x_0 \vee \overline{x}_2 x_1 \vee \overline{x}_2\overline{x}_0 \vee x_1\overline{x}_0 \vee x_3$

Član $x_2\overline{x}_1 x_0$ nije optimalan u funkcijama I_1, I_2 i I_4 posmatranim same za sebe, ali ovaj oblik minimizira ukupan broj neophodnih ulaza u logička kola za potrebe realizacije sklopa.

11.34 a) Najekonomičnija realizacija na bazi DNF slijedi iz jednačina:

$I_1 = \overline{x}_3 x_2\overline{x}_1 x_0 \vee \overline{x}_3 x_1 x_0 \vee x_3 x_2\overline{x}_0 \vee x_3\overline{x}_2\overline{x}_1 \vee x_2 x_1 \vee \overline{x}_2\overline{x}_0$

$I_2 = \overline{x}_3 x_2\overline{x}_1 x_0 \vee \overline{x}_3 x_2 x_1\overline{x}_0 \vee \overline{x}_3\overline{x}_1\overline{x}_0 \vee x_3 x_2\overline{x}_0 \vee x_3\overline{x}_2 \vee x_3 x_1$

$I_3 = x_3 x_2\overline{x}_1 x_0 \vee \overline{x}_3\overline{x}_1\overline{x}_0 \vee \overline{x}_3 x_1 x_0 \vee \overline{x}_2\overline{x}_1 \vee \overline{x}_2\overline{x}_0$

$I_4 = \overline{x}_3 x_2 x_1\overline{x}_0 \vee x_3 x_2\overline{x}_1 x_0 \vee \overline{x}_3 x_2\overline{x}_1 \vee x_3\overline{x}_2 \vee x_3 x_1 \vee \overline{x}_2 x_1$

$I_5 = \overline{x}_3 x_2 x_1\overline{x}_0 \vee x_3 x_2\overline{x}_1 x_0 \vee x_3 x_2\overline{x}_0 \vee x_3 x_1 \vee \overline{x}_2\overline{x}_0$

$I_6 = \overline{x}_3 x_2 x_1\overline{x}_0 \vee x_3 x_2\overline{x}_1 x_0 \vee \overline{x}_3 x_2\overline{x}_1 \vee \overline{x}_3 x_1 x_0 \vee x_3\overline{x}_2 \vee \overline{x}_2\overline{x}_1$

$I_7 = \overline{x}_3 x_2\overline{x}_1 x_0 \vee \overline{x}_3 x_2 x_1\overline{x}_0 \vee x_3 x_2\overline{x}_1 x_0 \vee \overline{x}_3\overline{x}_2\overline{x}_0 \vee x_3 x_2\overline{x}_0 \vee x_3\overline{x}_2\overline{x}_1 \vee \overline{x}_2 x_1 x_0$

Ni jedna od ovih funkcija nije optimalna sama za sebe, već su ovi DNF oblici odabrani tako da se minimizira ukupan broj neophodnih ulaza u logička kola za potrebe realizacije sklopa (ovu globalno optimalnu realizaciju nije posve jednostavno pronaći).

b) Najekonomičnija realizacija na bazi KNF slijedi iz jednačina:

$I_1 = (\overline{x}_3 \vee \overline{x}_2 \vee x_1 \vee \overline{x}_0)(\overline{x}_3 \vee x_2 \vee \overline{x}_1 \vee \overline{x}_0)(x_3 \vee \overline{x}_2 \vee x_1 \vee x_0)(x_3 \vee x_2 \vee x_1 \vee \overline{x}_0)$

$I_2 = (\overline{x}_3 \vee \overline{x}_2 \vee x_1 \vee \overline{x}_0)(x_3 \vee x_2 \vee \overline{x}_1 \vee x_0)(x_3 \vee x_2 \vee x_1 \vee \overline{x}_0)(x_3 \vee \overline{x}_1 \vee \overline{x}_0)$

$I_3 = (\overline{x}_3 \vee \overline{x}_2 \vee x_1 \vee x_0)(\overline{x}_3 \vee x_2 \vee \overline{x}_1 \vee \overline{x}_0)(x_3 \vee \overline{x}_2 \vee x_1 \vee \overline{x}_0)(\overline{x}_3 \vee \overline{x}_2 \vee \overline{x}_1)(\overline{x}_2 \vee \overline{x}_1 \vee x_0)$

$I_4 = (\overline{x}_3 \vee \overline{x}_2 \vee x_1 \vee x_0)(x_3 \vee \overline{x}_2 \vee \overline{x}_1 \vee \overline{x}_0)(x_3 \vee x_2 \vee x_1)$

$I_5 = (x_3 \vee \overline{x}_2 \vee x_1 \vee \overline{x}_0)(x_3 \vee \overline{x}_2 \vee x_1 \vee x_0)(x_3 \vee \overline{x}_1 \vee \overline{x}_0)(x_2 \vee x_1 \vee \overline{x}_0)$

$I_6 = (\overline{x}_3 \vee \overline{x}_2 \vee x_1 \vee x_0)(x_3 \vee x_2 \vee \overline{x}_1 \vee x_0)(\overline{x}_3 \vee \overline{x}_2 \vee \overline{x}_1)$

$I_7 = (\overline{x}_3 \vee x_2 \vee \overline{x}_1 \vee x_0)(x_3 \vee \overline{x}_2 \vee x_1 \vee x_0)(x_3 \vee x_2 \vee x_1 \vee \overline{x}_0)(\overline{x}_2 \vee \overline{x}_1 \vee \overline{x}_0)$

Funkcije I_2, I_3, I_5 i I_6 nisu optimalne same za sebe, nego su njihovi KNF oblici odabrani tako da se minimizira ukupan broj neophodnih ulaza u logička kola za potrebe realizacije sklopa.

11.35 Neka je $X = [x_3;x_2;x_1;x_0]$ i $Y = [y_3;y_2;y_1;y_0]$. Za konverziju iz klasičnog binarnog zapisa u Grayev kod, nakon formiranja tablice istine, minimizacije, i primjena osobina ekskluzivne disjunkcije, lako se dobija

$y_3 = x_3 \qquad y_2 = x_3 \oplus x_2 \qquad y_1 = x_2 \oplus x_1 \qquad y_0 = x_1 \oplus x_0$

Tablicu istine za inverznu pretvorbu možemo formirati analognim postupkom kao u Zadatku 11.16, nakon čega dobijamo

$x_3 = y_3 \qquad x_2 = y_3 \oplus y_2 \qquad x_1 = y_3 \oplus y_2 \oplus y_1 \qquad x_0 = y_3 \oplus y_2 \oplus y_1 \oplus y_0$

Interesantno je da se, u oba slučaja, konverziona matrica može realizirati korištenjem samo EXOR logičkih kola.

11.36 Uz $X = [x_3;x_2;x_1;x_0]$ i $Y = [y_3;y_2;y_1;y_0]$, za konverziju iz zapisa "znak i vrijednost" u zapis u *1KK* kodu, nakon formiranja tablice istine, minimizacije, i primjena osobina ekskluzivne disjunkcije, dobijamo

$y_3 = x_3 \qquad y_2 = x_3 \oplus x_2 \qquad y_1 = x_3 \oplus x_1 \qquad y_0 = x_3 \oplus x_0$

Slično, nakon formiranja tablice istine za inverznu pretvorbu, lako se dobija

$x_3 = y_3 \qquad x_2 = y_3 \oplus y_2 \qquad x_1 = y_3 \oplus y_1 \qquad x_0 = y_3 \oplus y_0$

I u ovom slučaju se konverziona matrica može realizirati korištenjem samo EXOR logičkih kola. Međutim, posebno je interesantna činjenica da se ista konverziona matrica može koristiti i za direktnu i za inverznu konverziju, s obzirom da su funkcije koje realiziraju direktnu i inverznu konverziju identičnog oblika.

11.37 Uz iste konvencije kao u prethodnim zadacima, dobijamo:

$y_3 = x_3x_2 \vee x_3x_1 \vee x_3x_0 = x_3(x_2 \vee x_1 \vee x_0)$

$y_2 = x_3\overline{x}_2x_1 \vee x_3\overline{x}_2x_0 \vee x_2\overline{x}_1\overline{x}_0 \vee \overline{x}_3x_2 = x_3\overline{x}_1\overline{x}_0 \oplus x_3 \oplus x_2$

$y_1 = x_3\overline{x}_1x_0 \vee \overline{x}_3x_1 \vee x_1\overline{x}_0 = x_3x_0 \oplus x_1$

$y_0 = x_0$

Prikazani oblik funkcije y_2 u kojem se javlja ekskluzivna disjunkcija nije očigledan, i za njegovo izvođenje potrebne su izvjesne dosjetke. Za inverznu pretvorbu dobijamo:

$x_3 = y_3$

$x_2 = y_3y_2\overline{y}_1 \vee \overline{y}_3y_2 \vee y_3\overline{y}_2 = y_3y_2\overline{y}_1 \vee (y_3 \oplus y_2)$

$x_1 = y_3\overline{y}_1y_0 \vee \overline{y}_3y_1 \vee y_1\overline{y}_0 = y_3y_0 \oplus y_1$

$x_0 = y_0$

11.38 Uz iste konvencije kao u prethodnim zadacima, dobijamo

$y_3 = x_3\overline{x}_2 \vee x_3\overline{x}_1 \vee x_3\overline{x}_0 = x_3(\overline{x}_2 \vee \overline{x}_1 \vee \overline{x}_0) = x_3\overline{x_2x_1x_0}$

$y_2 = x_3\overline{x}_2x_1x_0 \vee \overline{x}_3x_2 \vee x_2\overline{x}_1 \vee x_2\overline{x}_0 = x_3x_1x_0 \oplus x_2$

$y_1 = x_3\overline{x}_1x_0 \vee \overline{x}_3x_1 \vee x_1\overline{x}_0 = x_3x_0 \oplus x_1$

$y_0 = \overline{x}_3x_0 \vee x_3\overline{x}_0 = x_3 \oplus x_0$

Za inverznu pretvorbu dobijamo:

$x_3 = y_3$

$x_2 = \overline{y}_3y_2 \vee y_2y_1 \vee y_2y_0 = y_2(\overline{y}_3 \vee y_1 \vee y_0)$

$x_1 = y_3\overline{y}_1\overline{y}_0 \vee \overline{y}_3y_1 \vee y_1y_0 = y_3\overline{y}_0 \oplus y_1$

$x_0 = y_3 \oplus y_0$

12.1 Za $n = 3$ smo imali jednačine

$y_0 = \overline{x}_0 \qquad y_1 = x_1 \oplus x_0 \qquad y_2 = x_2 \oplus x_1x_0 \qquad y_3 = x_3 \oplus x_2x_1x_0$

odakle se lako zaključuje da u općem slučaju vrijedi

$y_0 = \overline{x}_0, \quad y_i = x_i \oplus x_{i-1}x_{i-2}...x_1x_0 \quad \text{za} \quad i = 1..n{-}1$

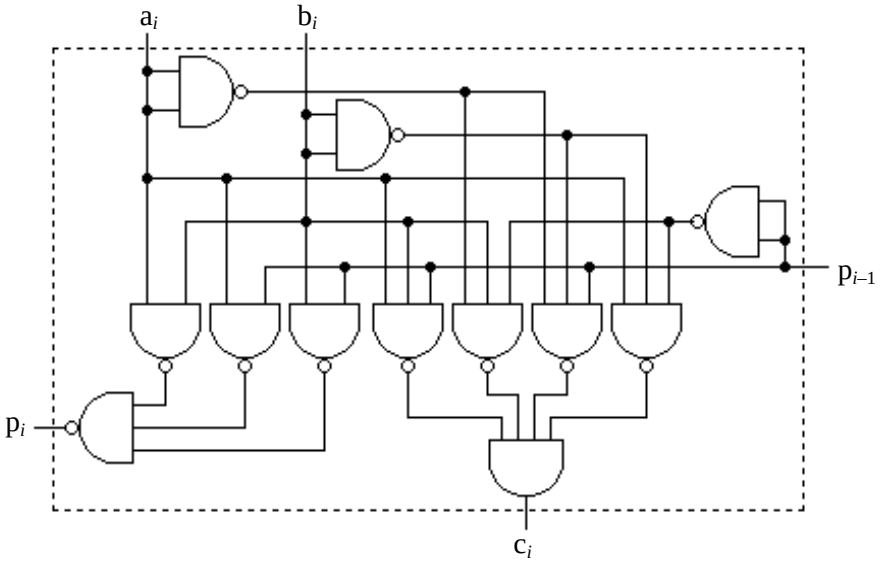
12.2 Za $n = 3$ smo imali jednačine

$y_0 = x_1 \oplus x_0 \qquad y_1 = x_2 \oplus x_1 \qquad y_2 = x_3 \oplus x_2 \qquad y_3 = x_3$

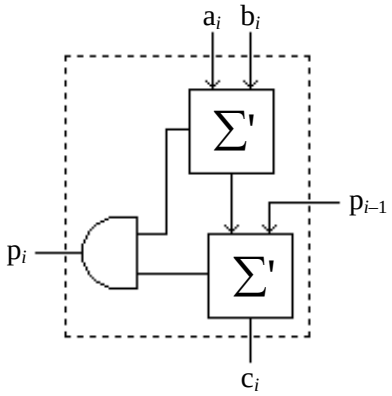
odakle se lako zaključuje da u općem slučaju vrijedi

$y_i = x_{i-1} \oplus x_i \quad \text{za} \quad i = 0 \,..\, n-2, \quad y_{n-1} = x_{n-1}$

12.3 $c_i = \overline{\overline{a_i b_i p_{i-1}}\ \overline{a_i a_i b_i}\ \overline{p_{i-1} p_{i-1}}\ \overline{a_i a_i}\ \overline{b_i b_i}\ \overline{p_{i-1}}\ \overline{a_i}\ \overline{b_i b_i}\ \overline{p_{i-1} p_{i-1}}}$, $\quad p_i = \overline{\overline{a_i b_i}\ \overline{a_i p_{i-1}}\ \overline{b_i p_{i-1}}}$



12.4 Pažljivom analizom jednačina polusumatora i punog sumatora, veoma lako se dolazi do sljedeće realizacije:



Od neposredne realizacije, ova realizacija punog sumatora razlikuje se po drugačijim vremenskim kašnjenjima izlaznih signala i signala u svim pojedinačnim tačkama sklopa u odnosu na trenutke promjene ulaznih signala, tako da se razlikuju i njihovi aspekti rizičnosti.

12.5 a) Na jedan od ulaza punog sumatora dovedemo A, na drugi ulaz 1, i na treći ulaz 0, nakon čega izlaz Y očitavamo sa izlaza za cifru rezultata.

b) Na dva ulaza punog sumatora dovedemo A i B, a na treći ulaz 0, nakon čega izlaz Y očitavamo sa izlaza za informaciju o prenosu.

c) Na dva ulaza punog sumatora dovedemo A i B, a na treći ulaz 1, nakon čega izlaz Y očitavamo sa izlaza za informaciju o prenosu.

d) Na dva ulaza punog sumatora dovedemo A i B, a na reći ulaz 0, nakon čega izlaz Y očitavamo sa izlaza za cifru rezultata.

e) Na dva ulaza punog sumatora dovedemo A i B, a na reći ulaz 1, nakon čega izlaz Y očitavamo sa izlaza za cifru rezultata.

Do ovih zaključaka se dolazi pažljivom analizom jednačina koje opisuju rad punog sumatora.

12.6 Tvrdnja direktno slijedi iz prethodnog zadatka, u kojem je pokazano kako se samo pomoću punog sumatora mogu realizirati osnovne logičke funkcije, na osnovu kojih se dalje može realizirati svaka logička funkcija.

12.7 Najekonomičnija realizacija dvobitnog sumatora koju smo dobili formalnim putem slijedila je iz sistema jednačina

$c_2 = a_1b_1 \vee a_0b_0\,(a_1 \vee b_1)$ $\quad c_1 = a_0b_0 \oplus a_1 \oplus b_1$ $\quad c_0 = a_0 \oplus b_1$

što za realizaciju zahtijeva tri dvoulazna AND kola, dva dvoulazna OR kola, te jedno dvoulazno i jedno troulazno EXOR kolo, što je ukupno 15 ulaza (ako EXOR kolo posmatramo ravnopravno sa ostalim). S druge strane, intuitivno projektirani dvobitni sumator zahtijeva jedan polusumator i jedan puni sumator za realizaciju. Uz najekonomičniju realizaciju punog sumatora, to ukupno zahtijeva tri dvoulazna AND kola, jedno dvoulazno OR kolo, i tri dvoulazna EXOR kola, što daje ukupno 14 ulaza. Slijedi da smo intuitivnim putem dobili ekonomičniju realizaciju! Analiza intuitivno realiziranog sklopa daje nam sljedeće jednačine:

$c_2 = a_1b_1 \vee a_0b_0\,(a_1 \oplus b_1)$ $\quad c_1 = a_0b_0 \oplus (a_1 \oplus b_1)$ $\quad c_0 = a_0 \oplus b_1$

Vidimo da se jednačina za c_2 razlikuje od one koju smo dobili formalnim putem. Naravno, ove dvije jednačine su ekvivalentne, ali je posljednja jednačina u takvom obliku da koristi član $a_1 \oplus b_1$ koji se javlja i u jednačini za c_1, odakle slijedi prikazana ušteda. Ipak, intuitivno projektirani sklop je sporiji. U izrazu za c_2 koristi se EXOR kolo, koje je tipično sporije od OR kola, dok je za formiranje c_1 potrebna prolazak kroz dva EXOR kola, umjesto kroz jedno u formalno projektiranom sklopu.

12.9 Nakon što se sklop projektira intuitivnim putem, dovoljno je sa sheme projektiranog sklopa očitati jednačine koje opisuju njegov rad, i svesti ih pomoću pravila logičke algebre na DNF ili KNF oblik, koji omogućavaju optimalnu realizaciju sa aspekta brzine. Naravno, ovim se ne garantira da ćemo dobiti najekonomičnije oblike za DNF odnosno KNF. Na žalost, minimizaciju dobijenih DNF odnosno KNF oblika obično je teško provesti ukoliko se u jednačinama javlja veliki broj promjenljivih.

12.10 Posmatrajući blok strukturu 4-bitnog paralelnog sumatora, i uzimajući u obzir jednačine polusumatora i punog sumatora, možemo pisati sljedeće jednačine:

$c_0 = a_0 \oplus b_0$
$p_0 = a_0b_0$
$c_1 = a_1 \oplus b_1 \oplus p_0 = a_1 \oplus b_1 \oplus a_0b_0$
$p_1 = a_1b_1 \vee (a_1 \vee b_1)\,p_0 = a_1b_1 \vee (a_1 \vee b_1)\,a_0b_0$
$c_2 = a_2 \oplus b_2 \oplus p_1 = a_2 \oplus b_2 \oplus [a_1b_1 \vee (a_1 \vee b_1)\,a_0b_0]$
$p_2 = a_2b_2 \vee (a_2 \vee b_2)\,p_1 = a_2b_2 \vee (a_2 \vee b_2)\,[a_1b_1 \vee (a_1 \vee b_1)\,a_0b_0]$
$c_3 = a_3 \oplus b_3 \oplus p_2 = a_3 \oplus b_3 \oplus \{a_2b_2 \vee (a_2 \vee b_2)\,[a_1b_1 \vee (a_1 \vee b_1)\,a_0b_0]\}$
$p_3 = a_3b_3 \vee (a_3 \vee b_3)\,p_2 = a_3b_3 \vee (a_3 \vee b_3)\,\{a_2b_2 \vee (a_2 \vee b_2)\,[a_1b_1 \vee (a_1 \vee b_1)\,a_0b_0]\}$
$c_4 = p_3 = a_3b_3 \vee (a_3 \vee b_3)\{a_2b_2 \vee (a_2 \vee b_2)\,[a_1b_1 \vee (a_1 \vee b_1)\,a_0b_0]\}$

Slijedi da se, nakon oslobađanja od pomoćnih promjenljivih p_0, p_1, p_2 i p_3, jednačine 4-bitnog paralelnog sumatora mogu napisati u sljedećem obliku:

$c_0 = a_0 \oplus b_0$
$c_1 = a_1 \oplus b_1 \oplus a_0b_0$
$c_2 = a_2 \oplus b_2 \oplus [a_1b_1 \vee (a_1 \vee b_1)\,a_0b_0] = a_2 \oplus b_2 \oplus (a_1b_1 \vee a_1a_0b_0 \vee a_0b_1b_0)$
$c_3 = a_3 \oplus b_3 \oplus \{a_2b_2 \vee (a_2 \vee b_2)\,[a_1b_1 \vee (a_1 \vee b_1)\,a_0b_0]\} =$
$\quad = a_3 \oplus b_3 \oplus (a_2b_2 \vee a_2a_1b_1 \vee a_1b_2b_1 \vee a_2a_1a_0b_0 \vee a_2a_0b_1b_0 \vee a_1a_0b_2b_0 \vee a_0b_2b_1b_0)$
$c_4 = a_3b_3 \vee (a_3 \vee b_3)\,(a_2b_2 \vee a_2a_1b_1 \vee a_1b_2b_1 \vee a_2a_1a_0b_0 \vee a_2a_0b_1b_0 \vee a_1a_0b_2b_0 \vee a_0b_2b_1b_0) =$
$\quad = a_3b_3 \vee a_3a_2b_2 \vee a_2b_3b_2 \vee a_3a_2a_1b_1 \vee a_3a_1b_2b_1 \vee a_2a_1b_3b_1 \vee a_1b_3b_2b_1 \vee a_3a_2a_1a_0b_0 \vee a_3a_2a_0b_1b_0$
$\qquad \vee a_3a_1a_0b_2b_0 \vee a_3a_0b_2b_1b_0 \vee a_2a_1a_0b_3b_0 \vee a_2a_0b_3b_1b_0 \vee a_1a_0b_3b_2b_0 \vee a_0b_3b_2b_1b_0$

Da bismo dobili DNF oblik, koji garantira optimalnost sa aspekta brzine, potrebno je još samo osloboditi se ekskluzivne disjunkcije (što daje prilično glomazne izraze). Eventualno, ekskluzivnu disjunkciju možemo zadržati, ukoliko dopustimo minoran gubitak na brzini.

12.11 a) Za poluoduzimač dobijamo sljedeće jednačine:

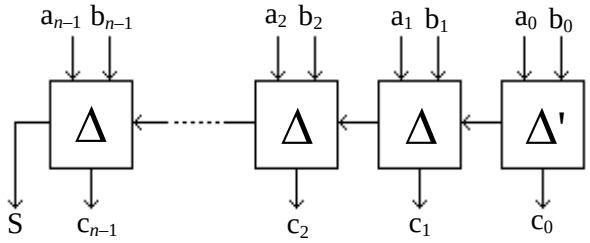
$c_0 = a_0 \oplus b_0$ $\qquad$ $p_0 = \overline{a}_0 b_0$

Za puni oduzimač dobijamo sljedeće jednačine:

$c_i = a_i \oplus (b_i \oplus p_{i-1})$ $\qquad$ $p_i = \overline{a}_i (b_i \vee p_{i-1}) \vee b_i p_{i-1} = \overline{a}_i (b_i \oplus p_{i-1}) \vee b_i p_{i-1}$

Realizacija direktno slijedi iz izvedenih jednačina. Vidimo da su izrazi za c_0 odnosno c_i identični kao kod polusumatora, odnosno punog sumatora, dok se izrazi za p_0 odnosno p_i praktično razlikuju samo što je promjenljiva a_0 negirana.

b) Realizacija direktno slijedi iz sljedeće blok strukture (pri čemu su poluoduzimač i puni oduzimač označeni analogno polusumatoru i punom sumatoru, uz korištenje simbola Δ' i Δ umjesto $\sum'$ i $\sum$):



Bitno je napomenuti da kod poluoduzimača i punog oduzimača njihovi ulazi *nisu međusobno ravnopravna* (tj. mora se znati koji je koji ulaz), za razliku od polusumatora i punog sumatora (čiji su svi ulazi posve ravnopravni), pa tu činjenicu treba nekako naznačiti na blok shemi u slučajevima kad to nije posve jasno iz konteksta.

12.12 Da.

12.13 Da, ukoliko se predvidi ulazna posudba (analogno ulaznom prenosu).

12.15 Pođimo od jednačine koja opisuje višefunkcionalnu logičku jedinicu iz Zadatka 11.32:

$Y = \overline{c}_2 \overline{A} B \vee \overline{c}_2 A \overline{B} \vee \overline{c}_1 A B \vee \overline{c}_0 \overline{A}\,\overline{B}$

Ova jednačina i dalje treba da važi, osim za $c_2 = c_1 = c_0 = 0$. Stoga, pomnožimo desnu stranu ove jednačine sa $c_2 \vee c_1 \vee c_0$, čime dobijamo funkciju Y_1 koja je identična sa Y, osim za $c_2 = c_1 = c_0 = 0$ kada vrijedi $Y_1 = 0$:

$Y_1 = (\overline{c}_2 \overline{A} B \vee \overline{c}_2 A \overline{B} \vee \overline{c}_1 A B \vee \overline{c}_0 \overline{A}\,\overline{B})(c_2 \vee c_1 \vee c_0)$

S druge strane, za $c_2 = c_1 = c_0 = 0$ treba da vrijedi $Y = A \oplus B \oplus p_u$. Pomnožimo li desnu stranu ove jednačine sa $\overline{c}_2 \overline{c}_1 \overline{c}_0$, dobijamo funkciju Y_2 koja je identična sa ovim izrazom zaY kada je $c_2 = c_1 = c_0 = 0$, a inače je $Y_2 = 0$:

$Y_2 = (A \oplus B \oplus p_u)\overline{c}_2 \overline{c}_1 \overline{c}_0$

Izrazi Y_1 i Y_2 se međusobno isključuju, tako da jednačinu traženog sklopa možemo formirato kao $Y = Y_1 \vee Y_2$, odnosno:

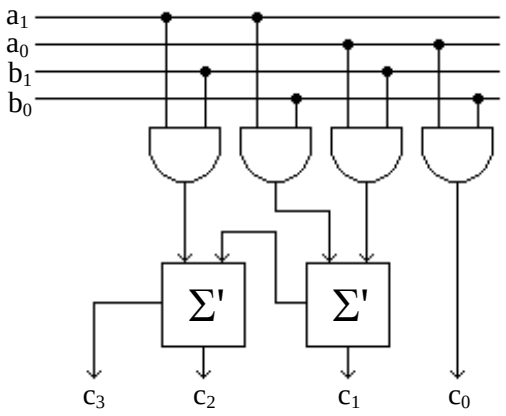
$Y = (\overline{c}_2 \overline{A} B \vee \overline{c}_2 A \overline{B} \vee \overline{c}_1 A B \vee \overline{c}_0 \overline{A}\,\overline{B})(c_2 \vee c_1 \vee c_0) \vee (A \oplus B \oplus p_u)\overline{c}_2 \overline{c}_1 \overline{c}_0$

Ovaj izraz se, sa aspekta ekonomičnosti, ne isplati svoditi na DNF. Neznatno ekonomičniju realizaciju možemo dobiti ukoliko ovaj izraz napišemo u obliku

$Y = (\overline{c}_2 \overline{A} B \vee \overline{c}_2 A \overline{B} \vee \overline{c}_1 A B \vee \overline{c}_0 \overline{A}\,\overline{B})\overline{\overline{c}_2 \overline{c}_1 \overline{c}_0} \vee (A \oplus B \oplus p_u)\overline{c}_2 \overline{c}_1 \overline{c}_0$

Što se tiče izlaza p_i, za $c_2 = c_1 = c_0 = 0$ treba da vrijedi $p_i = AB \vee (A \vee B)p_u$, ili, u ekvivalentnom obliku, $p_i = AB \vee (A \oplus B)p_u$, dok za druge vrijednosti c_2, c_1 i c_0 vrijednost izlaza p_i nije bitna. Zbog toga, možemo staviti $p_i = AB \vee (A \vee B)p_u$ neovisno od c_2, c_1 i c_0.

12.17 Tražena realizacija je prikazana na slici:



S obzirom na strukturu polusumatora, ova realizacija zahtijetva šest dvoulaznih AND kola, i dva dvoulazna EXOR kola (ukupno 16 ulaza). Ova realizacija je vrlo ekonomična, čak ekonomičnija od većine realizacija koja slijedi iz jednačina izvedenih u Zadatku 11.25, sa izuzetkom jednačina

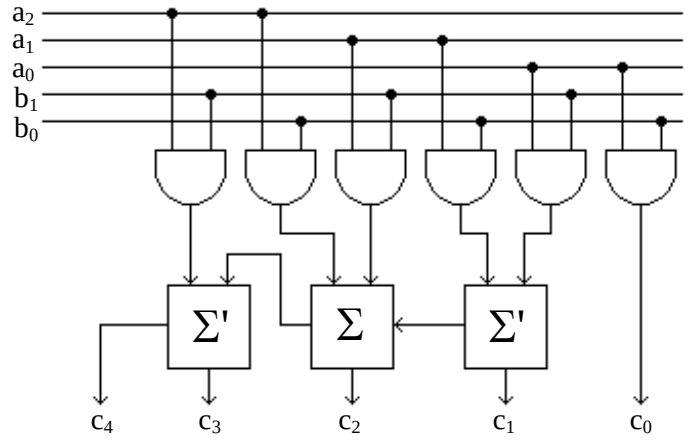
$c_0 = a_0 b_0$ $c_1 = a_1 b_0 \oplus a_0 b_1$ $c_2 = a_1 b_1 \bar{c}_0$ $c_3 = a_1 b_1 c_0$

koje omogućavaju realizaciju sa jednim ulazom manje, i uz jedno EXOR kolo manje (ali je potreban i jedan invertor).S druge strane, iz gore prikazane realizacije slijede jednačine

$c_0 = a_0 b_0$ $c_1 = a_1 b_0 \oplus a_0 b_1$ $c_2 = a_1 b_1 \oplus a_1 a_0 b_1 b_0$ $c_3 = a_1 b_1 a_0 b_0$

Dobijeni oblik za c_2 je vrlo interesantan, i nismo ga uočili prilikom formalnog projektiranja. Ako u ovim jednačinama uočimo da vrijedi $c_3 = a_1 b_1 c_0$ i $c_2 = a_1 b_1 \oplus c_3$, dolazimo do realizacije koja zahtijeva 14 ulaza (s obzirom da se faktor $a_1 b_1$ javlja kao zajednički u c_3 i c_2). Vidimo da formalni metod projektiranja može dati vrlo ekonomična rješenja. Naravno, sva ova rješenja su inferiorna sa aspekta brzine prema rješenju koje slijedi iz DNF odnosno KNF oblika.

12.18 Tražena realizacija je prikazana na slici



S obzirom na jednačine polusumatora i punog sumatora, možemo pisati:

$c_0 = a_0 b_0$
$c_1 = a_1 b_0 \oplus a_0 b_1$
$p_1 = a_1 a_0 b_1 b_0$
$c_2 = a_1 b_1 \oplus a_2 b_0 \oplus p_1 = a_1 b_1 \oplus a_2 b_0 \oplus a_1 a_0 b_1 b_0$
$p_2 = a_2 a_1 b_1 b_0 \vee (a_2 b_0 \vee a_1 b_1)\, p_1 = a_2 a_1 b_1 b_0 \vee (a_2 b_0 \vee a_1 b_1)\, a_1 a_0 b_1 b_0$
$c_3 = a_2 b_1 \oplus p_2 = a_2 b_1 \oplus [a_2 a_1 b_1 b_0 \vee (a_2 b_0 \vee a_1 b_1)\, a_1 a_0 b_1 b_0]$
$p_3 = a_2 b_1 p_2 = a_2 b_1 [a_2 a_1 b_1 b_0 \vee (a_2 b_0 \vee a_1 b_1)\, a_1 a_0 b_1 b_0]$
$c_4 = p_3 = a_2 b_1 [a_2 a_1 b_1 b_0 \vee (a_2 b_0 \vee a_1 b_1)\, a_1 a_0 b_1 b_0]$

Slijedi da se, nakon oslobađanja od pomoćnih promjenljivih p_1, p_2 i p_3, jednačine sklopa mogu napisati u sljedećem obliku:

$c_0 = a_0 b_0$

$c_1 = a_1 b_0 \oplus a_0 b_1$

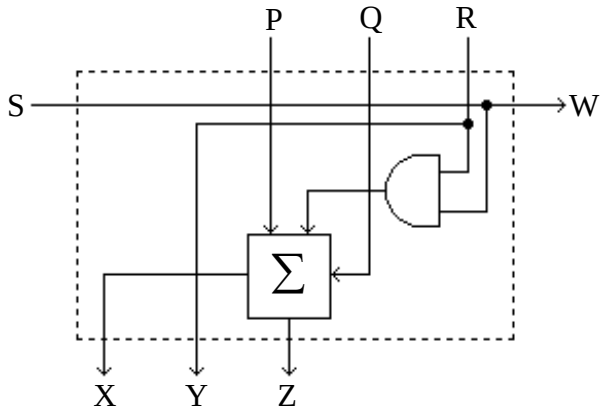
$c_2 = a_1 b_1 \oplus a_2 b_0 \oplus p_1 = a_1 b_1 \oplus a_2 b_0 \oplus a_1 a_0 b_1 b_0$

$c_3 = a_2 b_1 \oplus [a_2 a_1 b_1 b_0 \vee (a_2 b_0 \vee a_1 b_1)\, a_1 a_0 b_1 b_0] = a_2 b_1 \oplus (a_2 a_1 b_1 b_0 \vee a_1 a_0 b_1 b_0)$

$c_4 = a_2 b_1\, [a_2 a_1 b_1 b_0 \vee (a_2 b_0 \vee a_1 b_1)\, a_1 a_0 b_1 b_0] = a_2 a_1 b_1 b_0$

Da bismo dobili optimalne izraze sa aspekta brzine, potrebno je još samo osloboditi se ekskluzivne disjunkcije iz izraza za c_1, c_2 i c_3.

12.19 Posmatrajući postupak množenja binarnih brojeva, zatim blok strukturu postavljenu u zadatku, kao i blok strukturu množača iz Primjera 12.3, nije teško zaključiti da se blokovi označeni sa "*" mogu realizirati prema sljedećoj shemi:



Posmatrajmo konkretan blok koji se nalazi na *i*-toj dijagonali i *j*-tom redu blok strukture. Ulazi R i S prihvataju cifre a_i odnosno b_j koje se množe. Ove cifre se prosto prosljeđuju na izlaze Y i W respektivno, da bi svi blokovi na istoj dijagonali radili sa istim ciframa a_i, odnosno svi blokovi u istom redu radili sa istim ciframa b_j (dakle, Y = R i W = S). Ulazi P i Q prihvataju odgovarajuću cifru produkta odnosno informaciju o prenosu koja se dobija nakon što su obrađeni svi biti množioca od b_0 do b_{j-1}, dok se na izlazima X i Z javlja nova informacija o prenosu odnosno nova cifra produkta nakon što je obrađen i bit b_j. Sa slike je očigledno da jednačine koje opisuju izlaze X i Z glase $X = (P \oplus Q)\, RS \vee PQ$ odnosno $Z = P \oplus Q \oplus RS$.

12.20 Neka je $A = [a_7;a_6;a_5;a_4;a_3;a_2;a_1;a_0]$ i $B = [b_7;b_6;b_5;b_4;b_3;b_2;b_1;b_0]$. Označimo $A' = [a_7;a_6;a_5;a_4]$, $A'' = [a_3;a_2;a_1;a_0]$, $B' = [b_7;b_6;b_5;b_4]$ i $B'' = [b_3;b_2;b_1;b_0]$. Tada je $A = 2^4 \cdot A' + A''$ i $B = 2^4 \cdot B' + B''$, tako da imamo

$A \cdot B = 2^8 \cdot (A' \cdot B') + 2^4 \cdot (A' \cdot B'' + A'' \cdot B') + A'' \cdot B''$

Produkte $A' \cdot B'$, $A' \cdot B''$, $A'' \cdot B'$ i $A'' \cdot B''$, koji imaju po 8 bita, možemo izračunati pomoću četiri množača koji množe četverobitne brojeve. Dalje, zbir $A' \cdot B'' + A'' \cdot B'$ možemo izračunati pomoću 8-bitnog paralelnog sumatora. Označimo ovaj zbir sa $[p_8;p_7;p_6;p_5;p_4;p_3;p_2;p_1;p_0]$. Također, označimo produkte $A' \cdot B'$ i $A'' \cdot B''$ sa $[q_7;q_6;q_5;q_4;q_3;q_2;q_1;q_0]$ i $[r_7;r_6;r_5;r_4;r_3;r_2;r_1;r_0]$ respektivno. Uz ovakve oznake, imamo:

$2^4 \cdot (A' \cdot B'' + A'' \cdot B') = [p_8;p_7;p_6;p_5;p_4;p_3;p_2;p_1;p_0;0;0;0;0]$

$2^8 \cdot (A' \cdot B') = [q_7;q_6;q_5;q_4;q_3;q_2;q_1;q_0;0;0;0;0;0;0;0;0]$

$2^8 \cdot (A' \cdot B') + A'' \cdot B'' = [q_7;q_6;q_5;q_4;q_3;q_2;q_1;q_0;r_7;r_6;r_5;r_4;r_3;r_2;r_1;r_0]$

Konačno imamo

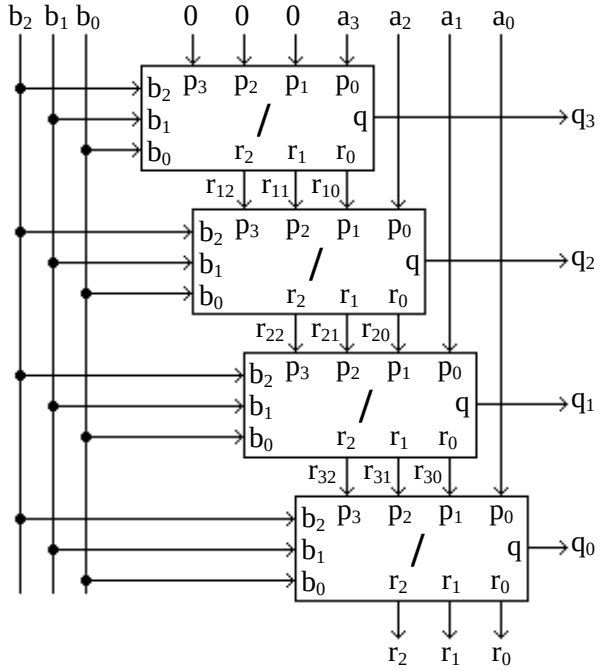
$A \cdot B = [q_7;q_6;q_5;q_4;q_3;q_2;q_1;q_0;r_7;r_6;r_5;r_4;r_3;r_2;r_1;r_0] + [0;0;0p_8;p_7;p_6;p_5;p_4;p_3;p_2;p_1;p_0;0;0;0;0]$

tako da nam je za formiranje konačnog rezultata potreban još jedan 16-bitni paralelni sumator (ili još dva 8-bitna paralelna sumatora). Na osnovu provedenih razmatranja, lako je nacrtati blok strukturu traženog sklopa (koju izostavljamo zbog glomaznosti).

12.21 Postupak dijeljenja četverobitnog broja trobitnim brojem može se prikazati sljedećom shemom:

$$
\begin{array}{rrrrrrcrrrcrrrr}
 & 0 & 0 & a_3 & a_2 & a_1 & a_0 & : & b_2 & b_1 & b_0 & = & q_3 & q_2 & q_1 & q_0 \\
\cline{1-5}
- & q_3b_2 & q_3b_1 & q_3b_0 & & & & & & & & & & & & \\
\cline{1-5}
 & r_{12} & r_{11} & r_{10} & a_2 & & & & & & & & & & & \\
- & & q_2b_2 & q_2b_1 & q_2b_0 & & & & & & & & & & & \\
\cline{1-6}
 & & r_{22} & r_{21} & r_{20} & a_1 & & & & & & & & & & \\
- & & & q_2b_2 & q_2b_1 & q_2b_0 & & & & & & & & & & \\
\cline{1-7}
 & & & r_{32} & r_{31} & r_{30} & a_0 & & & & & & & & & \\
- & & & & q_2b_2 & q_2b_1 & q_2b_0 & & & & & & & & & \\
\cline{1-8}
 & & & & r_2 & r_1 & r_0 & & & & & & & & &
\end{array}
$$

Ovdje je $[a_3;a_2;a_1;a_0]$ djeljenik, $[b_2;b_1;b_0]$ djelilac, $[q_3;q_2;q_1;q_0]$ količnik, a $[r_2;r_1;r_0]$ ostatak pri dijeljenju. Vodeće nule ispred djeljenika su dopisane zbog činjenice da i djelilac može imati vodeće nule (npr. djelilac može biti $(010)_2$). Na osnovu prikazane sheme postupka dijeljenja, slijedi da se djelitelj može projektirati prema sljedećoj blok strukturi:



Blok na slici označen sa "/" treba da generira $q = 1$ ako je $[p_3;p_2;p_1;p_0] \geq [b_2;b_1;b_0]$, a $q = 0$ u suprotnom, kao i da generira izlaze r_2, r_1 i r_0 tako da vrijedi $[r_2;r_1;r_0] = [p_3;p_2;p_1;p_0] - [qb_2;qb_1;qb_0]$. Postoji mnogo načina da se to izvede, a najekonomičniji je sljedeći. Formiramo oduzimač koji računa razliku $[p_3;p_2;p_1;p_0] - [b_2;b_1;b_0]$ bez obzira na znak rezultata. Tada je $q = 1$ ako i samo ako je rezultat nenegativan, a informacija o tome se dobija na osnovu izlazne posudbe iz oduzimača. Označimo dobijenu razliku (bez bita za znak) sa $[s_2;s_1;s_0]$ (u općem slučaju razlika može imati i bit s_3, ali zbog prirode postupka dijeljenja će uvijek biti $s_3 = 0$). Tada vrijedi $[r_2;r_1;r_0] = [s_2;s_1;s_0]$ ako je $q = 1$, a $[r_2;r_1;r_0] = [p_2;p_1;p_0]$ ako je $q = 0$, na osnovu čega slijedi

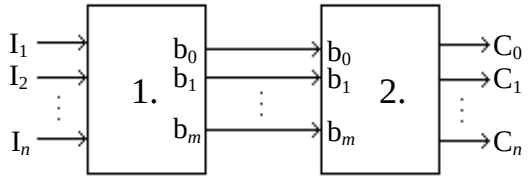
$r_2 = p_2\overline{q} \vee s_2 q$ $\qquad$ $r_1 = p_1\overline{q} \vee s_1 q$ $\qquad$ $r_0 = p_0\overline{q} \vee s_0 q$

Iz provedenog razmatranja sasvim je lako nacrtati strkuturu bloka "/".

12.22 Vidjeli smo ranije da mreže u više nivoa mogu biti mnogo ekonomičnije nego dvonivoske mreže, pogotovo ukoliko je broj nivoa veliki. Formalni postupci daju jednačine koje vode ka dvonivoskim mrežama, iz kojih je tek pomoću izvjesnih dosjetki moguće dobiti ekonomičnije jednačine koje vode ka mrežama u više nivoa (i to obično svega u 3 do 4 nivoa). S druge strane, ćelijsko-intuitivni pristup, zbog pojave kaskadnog vezivanja ćelija, izrazito forsira višenivoske mreže, ponekad čak do ekstremnih granica. Tako dobijene mreže mogu biti izrazito ekonomične. Cijena koja se zato plaća je znatno veće kašnjenje, i povećana mogućnost rizika.

12.23 Prvo rješenje zahtijeva dva puna sumatora, 5 AND kola sa ukupno 18 ulaza, i jedno petoulazno OR kolo, što s obzirom na strukturu punog sumatora daje ukupno 9 AND kola sa ukupno 26 ulaza, 3 OR kola sa ukupno 9 ulaza, i 4 EXOR kola sa ukupno 8 ulaza (43 ulaza u cjelini). Drugo rješenje zahtijeva četiri puna sumatora, i po jedno dvoulazno AND i OR kolo, što daje ukupno 9 AND kola sa ukupno 18 ulaza, 5 OR kola sa ukupno 10 ulaza, i 8 EXOR kola sa ukupno 16 ulaza (44 ulaza u cjelini). Slijedi da je prvo rješenje neznatno ekonomičnije od drugog (pogotovo što koristi manje EXOR kola, koja su složenija od ostalih). U svakom slučaju, oba rješenja su veliki napredak u odnosu na rješenje bazirano na SDNF obliku funkcije Y (koje zahtijeva 239 ulaza). Dalje, prvo rješenje je znatno brže od drugog, s obzirom da u njemu najsporiji put od nekod od ulaza do izlaza Y prolazi kroz jedno EXOR kolo i po dva AND i OR kola, dok u drugom rješenju najsporiji put prolazi kroz čak četiri EXOR kola i po tri AND i OR kola (to je, recimo, jedan od puteva od ulaza D, E ili F do izlaza koji prolazi kroz tri sumatora). Ova analiza vrijedi uz pretpostavku da EXOR logičko kolo nije više od dva puta sporije od ostalih logičkih kola (u suprotnom, postoje neki drugi putevi koji su sporiji), ali isti zaključci vrijede i bez ove pretpostavke.

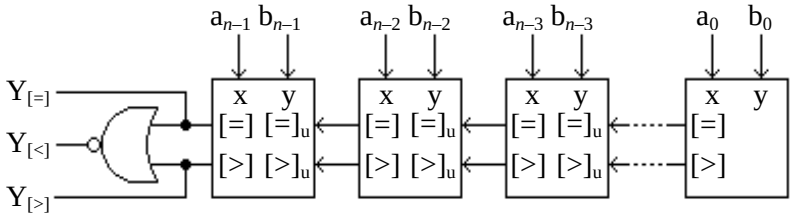
12.24 Nije teško vidjeti da se funkcije C_i, $i = 0..n$ mogu napisati kao disjunkcije svih mintermi obrazovanih od promjenljivih I_1, I_2, ... I_n u kojima se tačno i promjenljivih javlja sa znakom negacije. Međutim, lako je vidjeti da tako obrazovane funkcije C_i ujedno predstavljaju MDNF oblike (ni jedna minterma se ne može sažimati sa nijednom drugom mintermom u istoj funkciji). Slijedi da se funkcije C_i ne mogu optimizirati a da im realizacije ostanu u dva nivoa. Kada bismo ih realizirali na takav način, za sve funkcije C_i, $i = 0..n$ bilo bi nam potrebno svih 2^n mintermi, što bi na kraju za realizaciju zahtijevalo $(n+1)\cdot 2^n + n$ ulaza u logička kola. To npr. za $n = 7$ iznosi ukupno 1031 ulaz, što je zaista previše. Potražimo stoga bolje rješenje. Funkcionalnost obračunske mreže možemo razbiti na dva bloka, prema sljedećoj blok shemi:



Blok "**1.**" treba prebrojati na koliko se njegovih ulaza nalaze logičke jedinice (ili, što je isto, da izračuna zbir $I_1 + I_2 + ... + I_n$) i da rezultat prezentira na izlazu kao binarni broj $[b_m;...;b_1;b_0]$, pri čemu je $2^{m-1} \le n < 2^m$. Ovaj blok se lako može napraviti pomoću sumatora, koristeći slične ideje kao u drugom rješenju iz Primjera 12.3. Blok "**2.**" treba da na osnovu broja $[b_m;...;b_1;b_0]$ generira izlaze C_i, $i = 0..n$, što je trivijalno, jer se lako vidi da je $C_i = m_i(b_m, ... b_1, b_0)$. Na primjer, u Primjeru 12.3 smo vidjeli da se blok "**1.**" za $n = 7$ može realizirati pomoću 4 puna sumatora, što je ukupno 40 ulaza u logička kola, dok za $n = 7$ blok "**2.**" zahtijeva 3 invertora i 8 troulaznih AND kola (jer za $n = 7$ imamo $m = 2$, a potrebno nam je m+1 invertora i n+1 (m+1)-ulaznih AND kola), odnosno ukupno 27 ulaza. Slijedi da nam za obračunsku mrežu za $n = 7$ treba ukupno 67 ulaza, što je drastičan napredak u odnosu na 1031 ulaz, koji smo imali na početku!

12.26 Uz navedene pretpostavke, za brzi paralelni binarni komparator imamo maksimalno kašnjenje od 55 ns (najsporiji put prolazi kroz po jedno EXNOR, AND, OR i NOR kolo), dok za paralelni binarni komparator sa serijskom propagacijom imamo maksimalno kašnjenje od 345 ns (najsporiji put prolazi kroz 31 AND kolo, i kroz po jedno EXNOR i NOR kolo).

12.27 Traženi komparator trebao bi da ima sljedeću blok strukturu:



Krajnja desna ćelija je ista kao što je već projektirano u Primjeru 12.4, dok se ostale ćelije razlikuju od identično označenih ćelija iz Primjera 12.4. Sličnom analizom kao u Primjeru 12.4 može se izvesti da jednačine koje opisuju rad ove ćelije glase

$$[=] = \overline{x \oplus y}\,[=]_u \qquad [>] = (x \vee \overline{y})\,[>]_u \vee x\overline{y}$$

Ovakva ćelija je složenija u odnosu na slučaj kada se informacija propagira od viših bita ka nižim. Cjelokupna realizacija komparatora zasnovanog na ovakvim ćelijama zahtijeva $12n-5$ ulaza u logička kola, što iznosi 379 ulaza za $n=32$. Sa aspekta brzine, ovakvo rješenje je također lošije, jer uz realizaciju ćelija prema gornjim jednačinama, najduži put od ulaza do izlaza $Y_{[<]}$ prolazi kroz čak $2n+1$ logičko kolo (skoro dvostruko više nego pri propagaciji od viših bita ka nižim). Slijedi da je ovakva realizacija u svakom pogledu inferiorna u odnosu na realizaciju u kojoj se informacija kroz ćelije propagira od viših bita ka nižim.

12.28 Potrebno nam je jedno n-ulazno i jedno dvoulazno NOR kolo, n invertora i n punih sumatora, što uz najekonomičniju realizaciju punog sumatora zahtijeva ukupno $12n+2$ ulaza u logička kola. Najduži put od ulaza do izlaza $Y_{[<]}$ prolazi kroz čak $2n+4$ logička kola, odakle slijedi da je ovakva realizacija izuzetno spora.

12.29 a) $f_0 = f_1 = 1,\ f_2 = f_3 = 0$ b) $f_0 = f_2 = 1,\ f_1 = f_3 = 0$
c) $f_0 = f_1 = f_2 = 0,\ f_3 = 1$ d) $f_0 = 0,\ f_1 = f_2 = f_3 = 1$
e) $f_0 = f_3 = 0,\ f_1 = f_2 = 1$ f) $f_0 = f_3 = 1,\ f_1 = f_2 = 0$
g) $f_0 = f_1 = f_3 = 1,\ f_2 = 0$ b) $f_0 = f_2 = f_3 = 1,\ f_1 = 0$

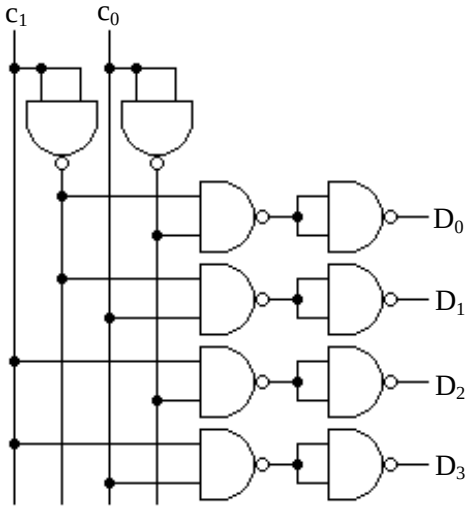
12.30 $Y = A\,\overline{c}_1\overline{c}_0 \vee B\,\overline{c}_1 c_0 \vee C\,c_1\overline{c}_0 \vee D\,c_1 c_0$ $Z = D\,\overline{c}_1\overline{c}_0 \vee C\,\overline{c}_1 c_0 \vee B\,c_1\overline{c}_0 \vee A\,c_1 c_0$

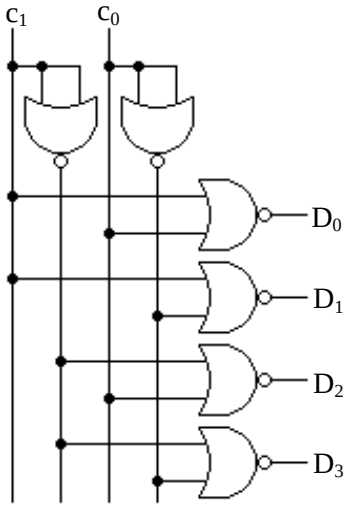
13.3 $D_0 = \overline{\overline{c_1 c_1}\ \overline{c_0 c_0}\ \overline{c_1 c_1}\ \overline{c_0 c_0}}$ $D_0 = \overline{c_1 \vee c_0}$

$D_1 = \overline{\overline{c_1 c_1}\ \overline{c_0}\ \overline{c_1 c_1}\ \overline{c_0}}$ $D_1 = \overline{c_1 \vee \overline{c_0 \vee c_0}}$

$D_2 = \overline{\overline{c_1}\ \overline{c_0 c_0}\ \overline{c_1}\ \overline{c_0 c_0}}$ $D_2 = \overline{\overline{c_1 \vee c_1} \vee c_0}$

$D_3 = \overline{\overline{c_1}\ \overline{c_0}\ \overline{c_1}\ \overline{c_0}}$ $D_3 = \overline{\overline{c_1 \vee c_1} \vee \overline{c_0 \vee c_0}}$





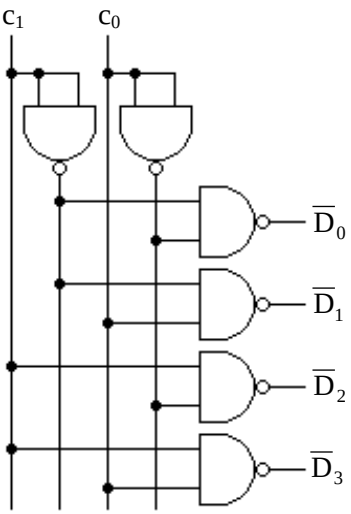
Realizacija sa NOR logičkim kolima je povoljnija.

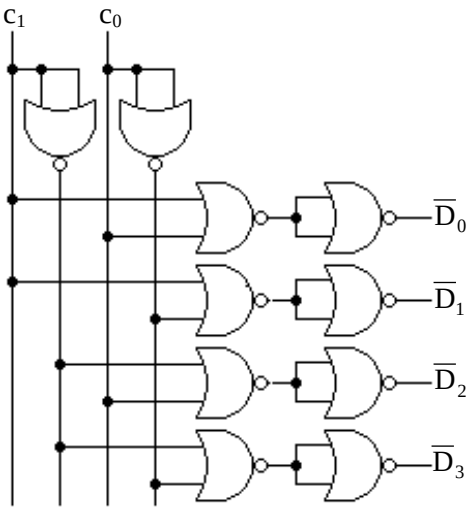
13.4 $\overline{D}_0 = \overline{\overline{c_1}c_1\,\overline{c_0}c_0}$ $\qquad \overline{D}_0 = \overline{c_1 \vee c_0 \vee c_1 \vee c_0}$

$\overline{D}_1 = \overline{\overline{c_1}c_1\,c_0}$ $\qquad \overline{D}_1 = \overline{c_1 \vee \overline{c_0} \vee c_0 \vee c_1 \vee \overline{c_0} \vee c_0}$

$\overline{D}_2 = \overline{c_1\,\overline{c_0}c_0}$ $\qquad \overline{D}_2 = \overline{\overline{c_1 \vee c_1} \vee c_0 \vee \overline{c_1 \vee c_1} \vee c_0}$

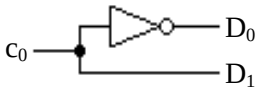
$\overline{D}_3 = \overline{c_1 c_0}$ $\qquad \overline{D}_3 = \overline{\overline{c_1 \vee c_1} \vee \overline{c_0 \vee c_0} \vee \overline{c_1 \vee c_1} \vee \overline{c_0 \vee c_0}}$



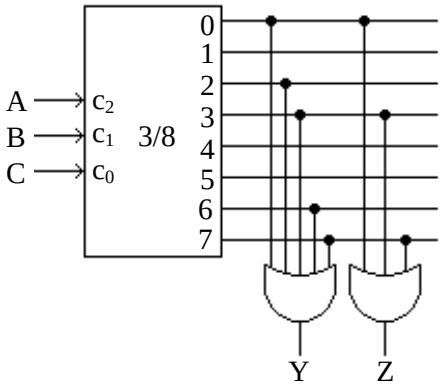


U ovom slučaju, povoljnija je realizacija sa NAND logičkim kolima.

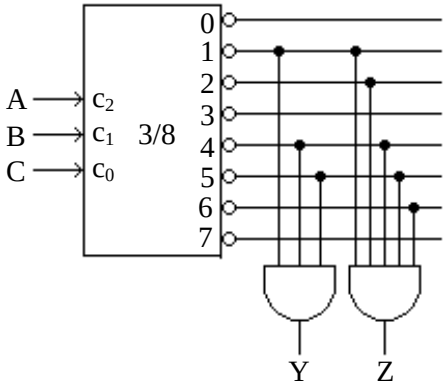
13.5 Za dekoder 1/2 imamo $D_0 = \overline{c}_0$ i $D_1 = c_0$, tako da se dekoder 1/2 sastoji samo od invertora:



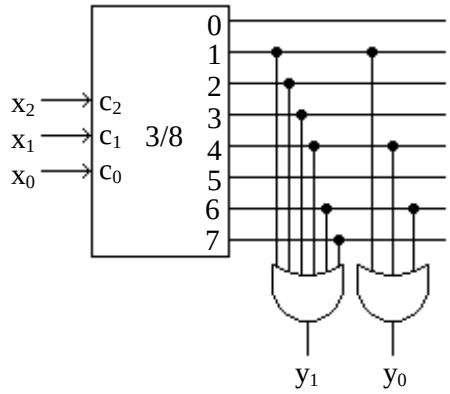
13.6 Tražena realizacija je prikazana na slici:



13.7 Tražena realizacija je prikazana na slici:

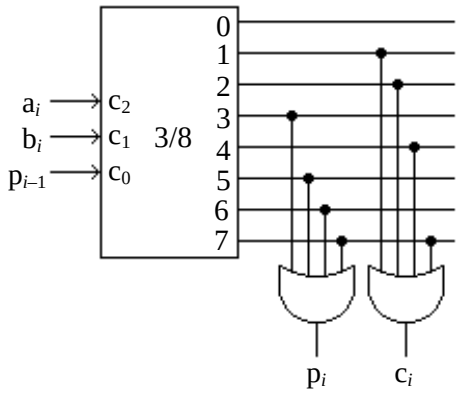


13.8 Mada ostatak dijeljenja sa 5 može imati i 3 bita (za slučaj da je ostatak 4), to se u navedenom sklopu ne dešava (svi rezultati su u opsegu od $0 \div 3$). Stoga su za reprezentaciju rezultata dovoljna dva bita, tako da su 2 OR kola dovoljna. Uz $Y = [y_1;y_0]$ imamo sljedeću realizaciju:

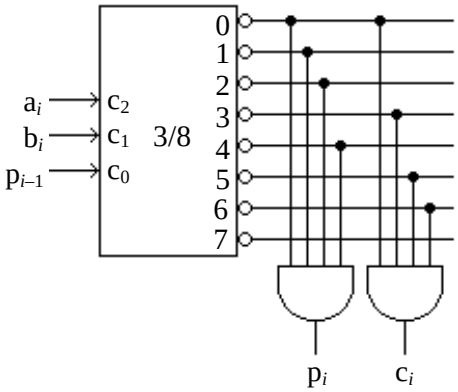


13.9 Tražene realizacije su prikazane na sljedećim slikama:

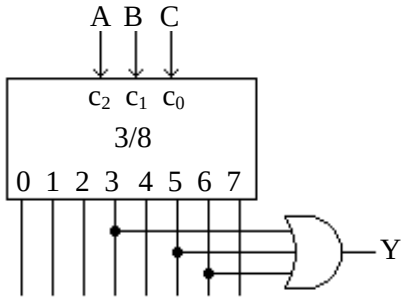
a)



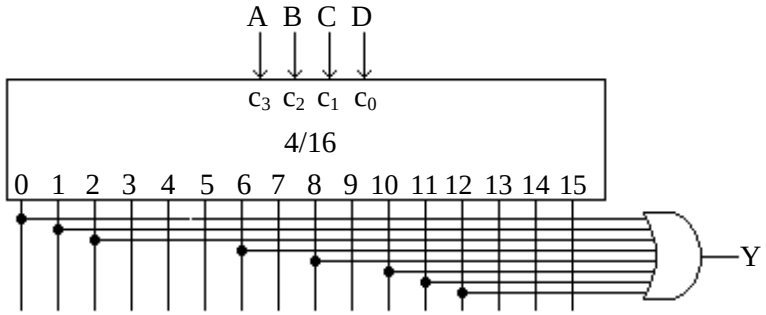
b)



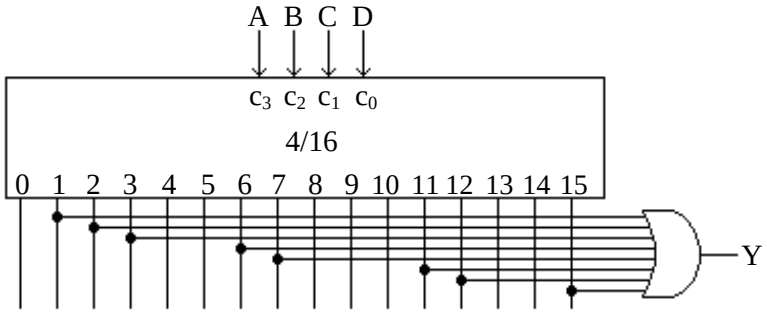
13.10 Ako traženu funkciju označimo sa Y, a ulazne promjenljive sa A, B i C, dobijamo sljedeću realizaciju:



13.11 Tražena realizacija je prikazana na slici:



13.12 Tražena realizacija je prikazana na sljedećoj slici:



13.14 a) 10240 ulaza b) 2368 ulaza

13.15 Neposredna realizacija zahtijeva 160 ulaza (ne brojeći invertore), a realizacija zasnovana na razbijanju zahtijeva 96 ulaza, što je ušteda od 64 ulaza.

13.16

p	2	3	4	5	6
q	6	5	4	3	2
Broj ulaza	904	696	640	696	904

Može se primijetiti da se ubjedljivo najpovoljnija realizacija dobija za $p = 4$, odnosno pri razbijanju dekodera na dva jednaka dijela.

13.17 Uputa: Neka je dekoder $n/2^n$ realiziran razbijanjem na dva dekodera $p/2^p$ i $q/2^q$, gdje je $p + q = n$. Neophodan broj ulaza u AND logička kola pri takvoj realizaciji je $N_u = p \cdot 2^p + q \cdot 2^q + 2 \cdot 2^p \cdot 2^q$, odnosno $N_u = p \cdot 2^p + (n-p) \cdot 2^{n-p} + 2^{n+1}$. Sad je samo potrebno pokazati da funkcija $N_u = N_u(p)$ dostiže maksimum za $p = n/2$. To možemo uraditi na razne načine, recimo pomoću diferencijalnog računa. Jednačinu $dN_u(p)/dp = 0$ nije lako riješiti, ali je sasvim lako provjeriti čistim uvrštavanjem da je $p = n/2$ njeno rješenje.

13.18 a) $N_{u0} = n \cdot 2^n$ b) $N_{u1} = n \cdot 2^{n/2} + 2^{n+1}$ c) $N_{u2} = n \cdot 2^{n/4} + 2^{n/2+2} + 2^{n+1}$

Za konkretne vrijednosti $n = 4$, $n = 8$, $n = 12$, $n = 16$, $n = 20$ i $n = 24$ dobijaju se sljedeći rezultati:

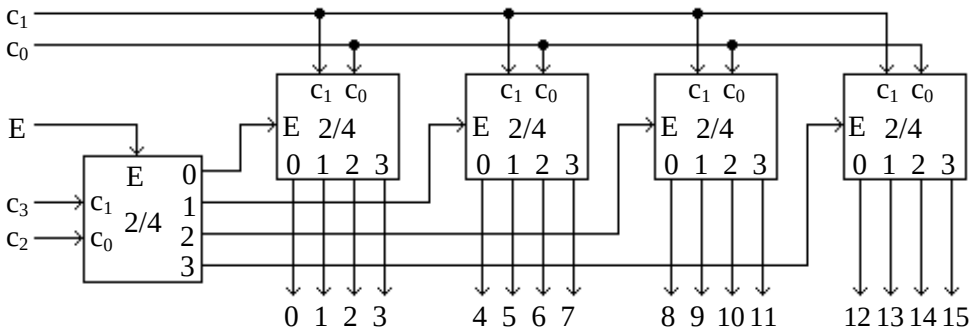
n	4	8	12	16	20	24
N_{u0}	64	2048	49152	1048576	20971520	402653184
N_{u1}	48	640	8960	135168	2117632	33652736
N_{u2}	–	608	8544	132352	2101888	33572352

Može se primijetiti da se odnos N_{u1}/N_{u0} a pogotovo N_{u2}/N_{u0} približava vrijednosti $n/2$ kako n raste. Ovo se može egzaktno pokazati preko graničnih vrijednosti. Također, može se dokazati da se broj ulaza u logička kola neophodan za realizaciju dekodera $n/2^n$ nikakvom strategijom ne može smanjiti više od $n/2$ puta u odnosu na broj ulaza pri neposrednoj realizaciji.

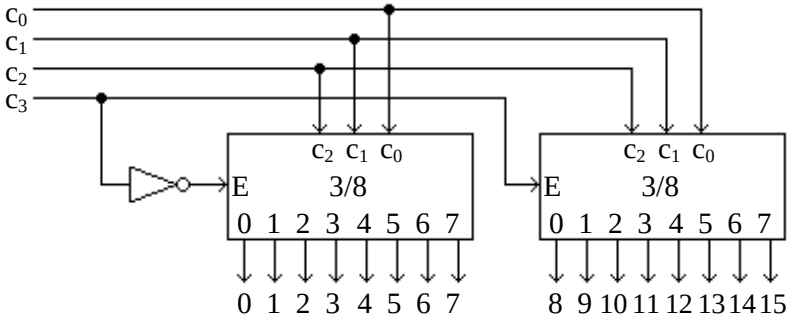
13.20 Tražena struktura se neposredno može nacrtati na osnovu jednačina

$D_0 = E\,\overline{c}_1\overline{c}_0$ $D_1 = E\,\overline{c}_1 c_0$ $D_2 = E\,c_1\overline{c}_0$ $D_3 = E\,c_1 c_0$

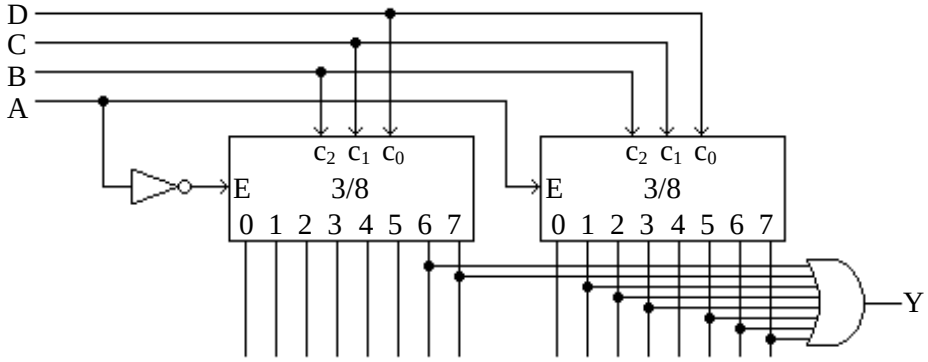
13.21 Tražena realizacija je prikazana na slici, pri čemu realizirani dekoder također ima omogućavajući ulaz:



13.22 Tražena realizacija neposredno slijedi iz činjenice da se invertor može posmatrati kao dekoder 1/2 (ali bez omogućavajućeg ulaza). Realizirani dekoder ne posjeduje omogućavajući ulaz.



13.23 Prvo je potrebno na osnovu dva dekodera 3/8 i invertora realizirati dekoder 4/16, a zatim dovršiti projektiranje kao da imamo dekoder 4/16. Tako dolazimo do sljedeće realizacije

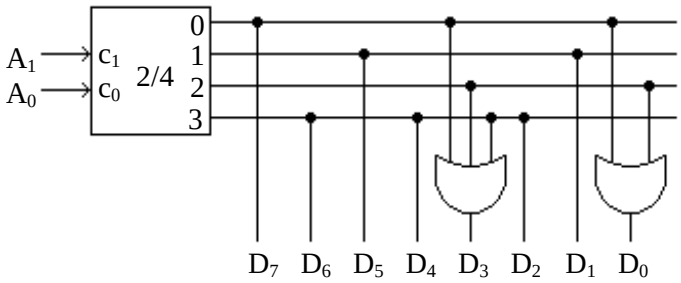


13.24 Za neposrednu realizaciju dekodera formata $(m+n)/2^{m+n}$ sa omogućavajućim ulazom neophodno je $N_u' = (m+n+1)\cdot 2^{m+n}$ ulaza u AND logička kola. Ukoliko realizaciju zasnujemo na spajanju jednog dekodera formata $m/2^m$ sa 2^m dekodera formata $n/2^n$ (sa omogućavajućim ulazima), ukupno je potrebno $N_u'' = (m+1)\cdot 2^m + (n+1)\cdot 2^{m+n}$ ulaza u AND logička kola. Kako je $N_u'' < N_u'$, to ovakva kaskadna realizacija uvijek donosi uštede u odnosu na neposrednu realizaciju. Ušteda u broju ulaza iznosi $N_u'' - N_u' = 2^m(m\cdot 2^n - m - 1)$. Uz fiksirano $m+n$, postoje vrijednosti za m i n pri kojima je ušteda najveća, ali njih nije moguće jednostavno iskazati formulom.

13.25 a) 192 ulaza b) 164 ulaza c) 140 ulaza
d) 128 ulaza e) 144 ulaza

Najekonomičnija realizacija dobija se u slučaju d), odnosno koristeći jedan dekoder 3/8 i osam dekodera 2/4 (ipak, razbijanje na jedan dekoder 2/4 i jedan dekoder 3/8 i kombiniranje izlazapomoću strategije "podijeli i osvoji" daje još veću uštedu, jer je tako potrebno svega 108 ulaza u AND logička kola).

13.27 Tražena struktura je prikazana na slici:

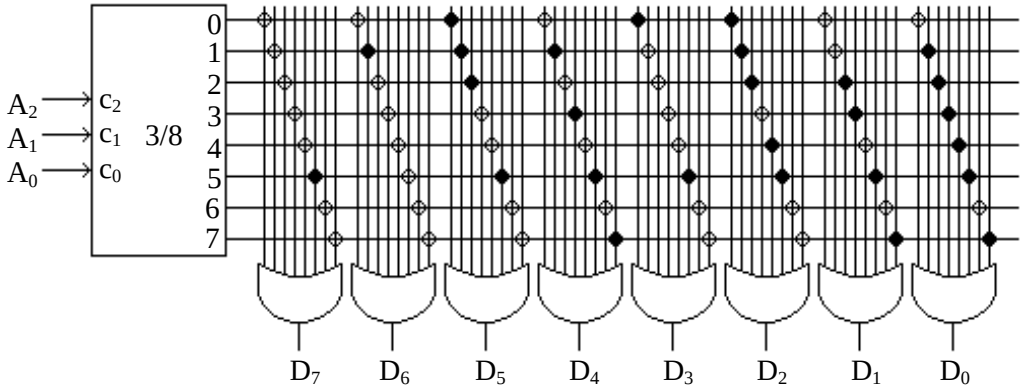


13.28 8 kilobajta.

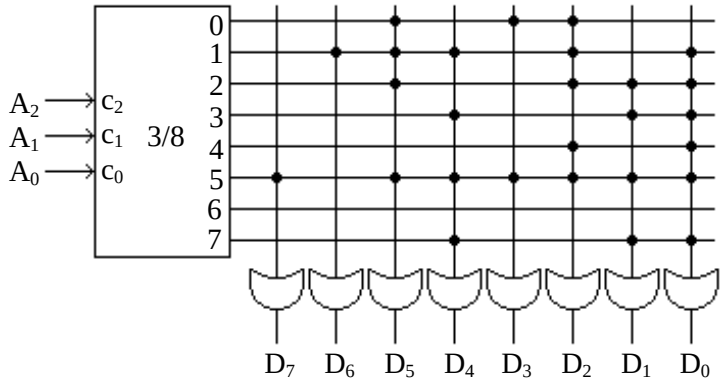
13.29 13 adresnih ulaza.

13.30 Ako ulaze a_1, a_0, b_1 i b_0 posmatramo kao adresne ulaze A_3, A_2, A_1 i A_0, a izlaze c_2, c_1 i c_0 kao izlaze podataka D_2, D_1 i D_0 respektivno, tada ovaj sklop možemo posmatrati kao ROM memoriju koja pamti 16 3-bitnih podataka, pri čemu se na adresama $0 \div 15$ nalaze respektivno binarni podaci 000, 001, 010, 011, 001, 010, 011, 100, 010, 011, 100, 101, 011, 100, 101 i 110, odnosno u dekadnom brojnom sistemu 0, 1, 2, 3, 1, 2, 3, 4, 2, 3, 4, 5, 3, 4, 5 i 6.

13.31 Tražena struktura je prikazana na slici (puni kružići označavaju čvorove koje treba zadržati, odnosno učiniti vodljivim, dok prazni kružići označavaju čvorove koje treba spaliti, odnosno ostaviti nevodljivim):



Radi bolje preglednosti, sheme poput prethodne se obično crtaju na sljedeći način, pri čemu se podrazumijeva da linije koje ulaze u OR kola predstavljaju skupinu paralelnih linija, dok čvorovi samo označavaju pozicije na kojima se nalaze aktivni čvorovi (ista konvencija se koristi prilikom shematskog prikaza PLA komponenti):



Upis podatka 19 (binarno 00010011) na adresu 3 (binarno 010) vrši se tako što prvo aktiviramo izlaz "3" iz dekodera dovođenjem $A_2 = 0$, $A_1 = A_0 = 0$. U slučaju PROM memorije, na izlaze D_7, D_6, D_5, D_3 i D_2 dovedemo jači napon koji će izazvati protok jače struje kroz čvorove koji spajaju

ove izlaze sa izlazom "3" iz dekodera, i tako ih spaliti (oni ne treba da budu prisutni). U slučaju EPROM ili EEPROM memorije, jači napon treba dovesti na izlaze D_4, D_1 i D_0, da protok jače struje učini odgovarajuće čvorove vodljivim.

13.32 a) 7 ulaza, 16 izlaza, 2048 pregorljivih čvorova
b) 9 ulaza, 8 izlaza, 4096 pregorljivih čvorova

13.33 Data memorija ima 4 izlaza ($D_0 \div D_3$), a za realizaciju punog sumatora su nam potrebna dva izlaza. Stoga će dva izlaza ostati neiskorištena (u slučaju potrebe, možemo ih iskoristiti za realizaciju nečeg drugog). Odlučimo li se da kao ulaze a_i, b_i i p_{i-1} punog sumatora iskoristimo respektivno ulaze A_2, A_1 i A_0 ROM memorije, a kao izlaze c_i i p_i respektivno izlaze D_3 i D_2 ROM memorije (ovaj izbor je proizvoljan), tada iz tablice istine punog sumatora neposredno slijedi da u memoriju na adrese $0 \div 7$ treba upisati binarne brojeve 00xx, 10xx, 10xx, 01xx, 10xx, 01xx, 01xx i 11xx respektivno. Biti označeni sa "x" mogu imati proizvoljnu vrijednost, jer se izlazi D_2 i D_1 ne koriste (naravno, odlučimo li se da i ovim izlazima damo neku funkciju, vrijednosti svih bita postaju definirane). U slučaju PROM-a, najlakše je uzeti da su svi biti "x" jedinice, jer su na početku svi čvorovi prisutni (tako ćemo najmanje čvorova morati spaliti). Drugim riječima, u memoriju treba redom upisati (dekadne) brojeve 3, 11, 11, 7, 11, 7, 7 i 15. U slučaju EPROM-a ili EEPROM-a, bolje je uzeti da su svi biti "x" nule, jer su na početku svi čvorovi nevodljivi. Tako, u memoriju treba redom upisati brojeve 0, 8, 8, 4, 8, 4, 4 i 12.

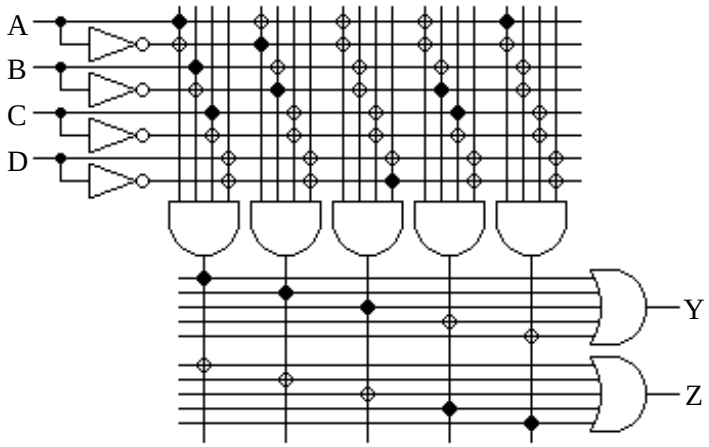
13.34 Za razliku od prošlog zadatka, ovdje imamo 6 suvišnih izlaza, i jedan suvišan ulaz (ova memorija ima 4 ulaza). Odlučimo se, na primjer, da kao ulaze a_i, b_i i p_{i-1} punog sumatora iskoristimo respektivno ulaze A_2, A_1 i A_0 ROM memorije, a kao izlaze c_i i p_i respektivno izlaze D_7 i D_6 ROM memorije. Ulaz A_3 ostaje neiskorišten. Ukoliko A_3 držimo stalno na nuli, slijedi da u memoriju na adrese $0 \div 7$ treba respektivno upisati binarne brojeve 00xxxxxx, 10xxxxxx, 10xxxxxx, 01xxxxxx, 10xxxxxx, 01xxxxxx, 01xxxxxx i 11xxxxxx. Za tretman bita "x" vrijedi ista diskusija kao u prethodnom zadatku. Na adrese $8 \div 15$ možemo upisati bilo šta. Ukoliko A_3 držimo stalno na jedinici, tada iste brojeve treba upisati na adrese $8 \div 15$, dok na adrese $0 \div 7$ možemo dovesti bilo šta. Ukoliko vrijednosti na ulazu A_3 mogu varirati od slučaja do slučaja (to npr. može biti slučaj ukoliko neiskorištene izlaze iz memorije iskoristimo za realizaciju nekih funkcija koje zavise od 4 promjenljive, tako da nam je i ulaz A_3 potreban za njihovu realizaciju), tada istu sekvencu od 8 brojeva treba upisati kako na adrese $0 \div 7$, tako i na adrese $8 \div 15$, da vrijednosti na izlazima D_7 i D_6 ne ovise od stanja ulaza A_3 (bolje rečeno, vrijednosti prva dva bita brojeva na adresama $0 \div 7$ i brojeva na adresama $8 \div 15$ trebaju biti iste).

13.35 Odlučimo li se da kao ulaze a_1, a_0, b_1 i b_0 traženog sklopa iskoristimo respektivno ulaze A_3, A_2, A_1 i A_0 ROM memorije, a kao izlaze c_2, c_1 i c_0 respektivno izlaze D_2, D_1 i D_0 ROM memorije (izlaz D_3 ostavljamo neiskorišten), tada iz tablice istine slijedi da u memoriju na adrese $0 \div 15$ treba upisati binarne brojeve x000, x001, x010, x011, x001, x010, x011, x100, x010, x011, x100, x101, x011, x100, x101 i x110 respektivno. Za tretman bita "x" vrijedi ista diskusija kao u prethodnim zadacima.

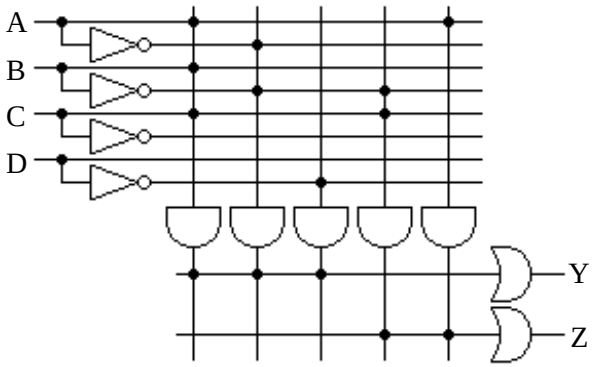
13.36 Odlučimo li se da kao ulaze x_3, x_2, x_1 i x_0 traženog sklopa iskoristimo respektivno ulaze A_3, A_2, A_1 i A_0 ROM memorije, a kao izlaze $I_1 \div I_7$ respektivno izlaze $D_1 \div D_7$ ROM memorije (izlaz D_0 ostavljamo neiskorišten), tada iz tablice istine slijedi da u memoriju na adrese $0 \div 9$ treba upisati binarne brojeve 1110111x, 0100100x, 1011101x, 1101101x, 0101110x, 1101011x, 1111011x, 0100101x, 1111111x i 1101111x respektivno. Na adrese $10 \div 15$ možemo upisati bilo šta. Za tretman bita "x" vrijedi ista diskusija kao u prethodnim zadacima.

13.37 Odlučimo li se da kao ulaze x_3, x_2, x_1 i x_0 traženog sklopa iskoristimo respektivno ulaze A_3, A_2, A_1 i A_0 ROM memorije, a kao izlaze y_3, y_2, y_1 i y_0 respektivno izlaze D_3, D_2, D_1 i D_0 ROM memorije (izlaze $D_4 \div D_7$ ostavljamo neiskorišten), tada iz tablice istine slijedi da u memoriju na adrese $0 \div 15$ treba upisati binarne brojeve xxxx0000, xxxx0001, xxxx0011, xxxx0010, xxxx0110, xxxx0111, xxxx0101, xxxx0100, xxxx1100, xxxx1101, xxxx1111, xxxx1110, xxxx1010, xxxx1011, xxxx1001 i xxxx1000. Za tretman bita "x" vrijedi ista diskusija kao u prethodnim zadacima.

13.39 Tražena struktura je prikazana na slici (puni kružići označavaju čvorove koje treba zadržati, dok prazni kružići označavaju čvorove koje treba spaliti):



Slično kao i kod prikaza programabilnih ROM memorija, sheme realizacija sa PLA komponentama se također prikazuju na pojednostavljeni način, u kojem se podrazumijeva da linije koje ulaze u AND odnosno OR kola predstavljaju skupine paralelnih linija, dok čvorovi označavaju pozicije na kojima se nalaze aktivni čvorovi. Uz ovakvu konvenciju, prethodna shema se pojednostavljeno prikazuje ovako:



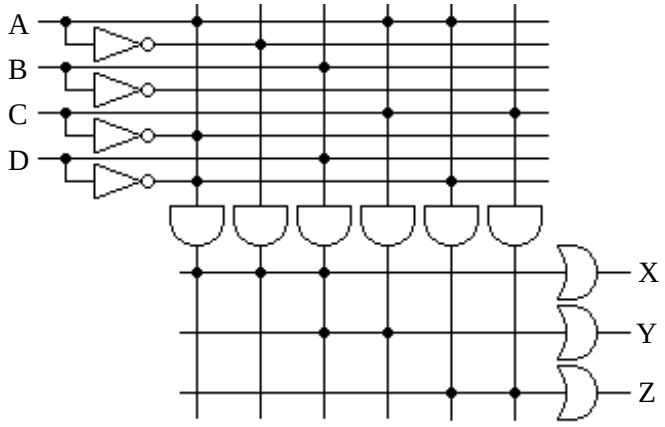
Traženu funkcionalnost nije moguće realizirati sa PLA komponentom koja sadrži manje od 5 AND kola, jer ne postoji način da se funkcije Y i Z napišu u oblicima u kojima bi se javilo manje od 5 različitih implikanti.

Napomena: Primijetimo da su AND kola u PLA realizaciji neophodna čak i za trivijalne implikante kao što je $\overline{\mathrm{D}}$ u funkciji Y odnosno A u funkciji Z, s obzirom da ne postoji način da se zaobiđe prolazak kroz AND kola (koja su ugrađena u samu PLA komponentu). Po ovoj činjenici se dizajn logičkih funkcija pomoću PLA komponenti bitno razlikuje od dizajna u kojem se koriste individualna logička kola.

13.40 $(2n + m) \cdot p$ pregorljivih čvorova.

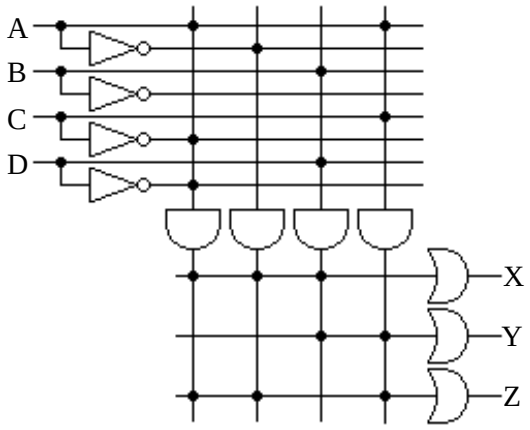
13.41 Sa PLA komponentom čija AND kola sadrže $n' < n$ ulaza nije moguće realizirati niti jednu logičku funkciju čija MDNF sadrži članove sa više od n' promjenljivih. Sa PLA komponentom čija OR logička kola sadrže $p' < p$ ulaza nije moguće realizirati niti jednu logičku funkciju čija MDNF sadrži više od p' članova.

13.42 a) Bez primjene ikakvih transformacija dobijamo realizaciju kao na sljedećoj slici (uz korištenje pojednostavljene simbolike prikazane u rješenju Zadatka 13.39), koja zahtijeva PLA komponentu sa barem 6 AND logičkih kola:



b) Uz pomoć Veitchovih dijagrama lako možemo pronaći oblike za funkcije X, Y i Z u kojima se ukupno javljaju svega 4 različite implikante, iz kojih neposredno slijedi tražena realizacija:

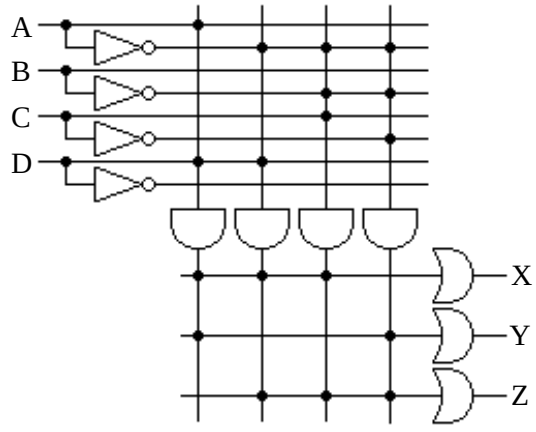
$X = A\overline{C}\,\overline{D} \vee \overline{A}C \vee BD \qquad Y = AC \vee BD \qquad Z = AC \vee \overline{A}C \vee A\overline{C}\,\overline{D}$



Napomena: Zbog potrebe za štednjom AND kola unutar PLA komponenti, tehnike objašnjene u Primjeru 11.4 za traženje zajedničkih implikanti u sistemu logičkih funkcija svoju glavnu primjenu dobijaju prilikom projektiranja pomoću PLA komponenti, i to u još drastičnijem stepenu nego pri korištenju individualnih logičkih kola. Na primjer, pri korištenju individualnih logičkih kola više bi se isplatilo uvesti novu implikantu C u funkciju Z nego koristiti postojeće implikante AC i $\overline{A}C$ iz funkcija Y odnosno X.

13.43 Tražena realizacija se dobija ukoliko funkcije X, Y i Z napišemo u sljedećem obliku (uporedite ovo rješenje sa rješenjem Zadatka 11.6, u kojem se ne koriste PLA komponente):

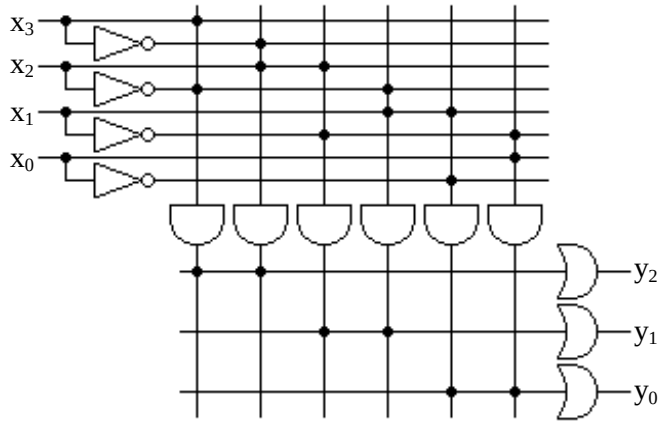
$X = AD \vee \overline{A}D \vee \overline{A}\,\overline{B}C \qquad Y = AD \vee \overline{A}\,\overline{B}\,\overline{C} \qquad Z = \overline{A}D \vee \overline{A}\,\overline{B}C \vee \overline{A}\,\overline{B}\,\overline{C}$



13.44 Tražena konverziona matrica opisana je sistemom jednačina

$y_0 = \overline{x}_1 x_0 \vee x_1 \overline{x}_0$ $\quad y_1 = \overline{x}_2 x_1 \vee x_2 \overline{x}_1$ $\quad y_2 = \overline{x}_3 x_2 \vee x_3 \overline{x}_2$ $\quad y_3 = x_3$

S obzirom da trivijalnu funkciju $y_3 = x_3$ ne treba realizirati kroz PLA komponentu, dovoljna su 4 ulaza, 3 izlaza i 6 AND logičkih kola (s obzirom da se broj implikanti ne može smanjiti):



13.45 Rješenje ponuđeno u Zadatku 11.33, koje je optimalno za realizaciju pomoću individualnih komponenti, zahtijeva 12 različitih implikanti. Da bismo udovoljili zahtjevima ovog zadatka, moramo pronaći rješenje koje ne zahtijeva više od 10 različitih implikanti, što je, uz dosta koncentracije, moguće izvesti pomoću Veitchovih dijagrama. Jedno takvo rješenje, koje zahtijeva svega 9 različitih implikanti, je sljedeće:

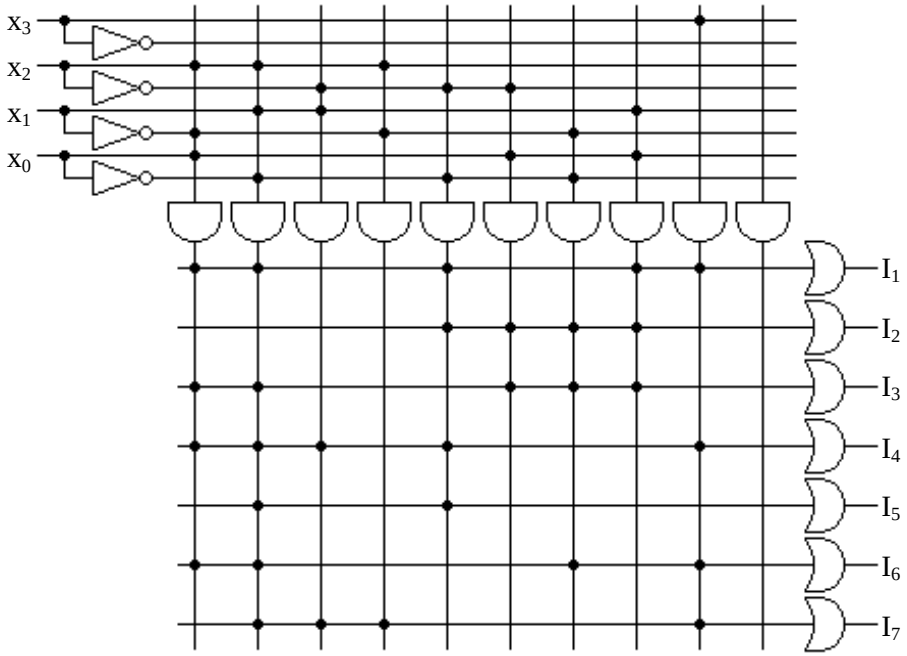
$I_1 = x_2 \overline{x}_1 x_0 \vee x_2 x_1 \overline{x}_0 \vee \overline{x}_2 \overline{x}_0 \vee x_1 x_0 \vee x_3$ $\quad I_2 = \overline{x}_2 \overline{x}_0 \vee \overline{x}_2 x_0 \vee \overline{x}_1 \overline{x}_0 \vee x_1 x_0$

$I_3 = x_2 \overline{x}_1 x_0 \vee x_2 x_1 \overline{x}_0 \vee \overline{x}_2 x_0 \vee \overline{x}_1 \overline{x}_0 \vee x_1 x_0$ $\quad I_4 = x_2 \overline{x}_1 x_0 \vee x_2 x_1 \overline{x}_0 \vee \overline{x}_2 x_1 \vee \overline{x}_2 \overline{x}_0 \vee x_3$

$I_5 = x_2 x_1 \overline{x}_0 \vee \overline{x}_2 \overline{x}_0$ $\quad I_6 = x_2 \overline{x}_1 x_0 \vee x_2 x_1 \overline{x}_0 \vee \overline{x}_1 \overline{x}_0 \vee x_3$

$I_7 = x_2 x_1 \overline{x}_0 \vee \overline{x}_2 x_1 \vee x_2 \overline{x}_1 \vee x_3$

Na osnovu ovog rješenja, sljedi prikazana realizacija (u kojoj jedno AND kolo ostaje neiskorišteno):



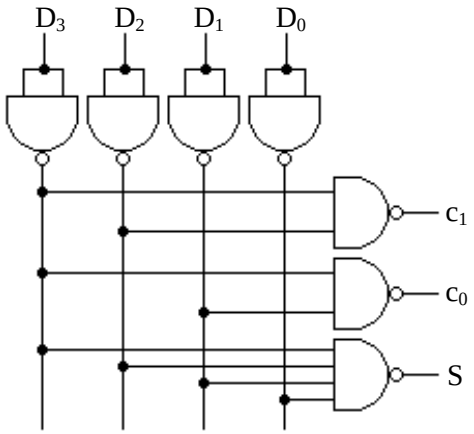
13.47 5 izlaza (računajući i izlaz S).

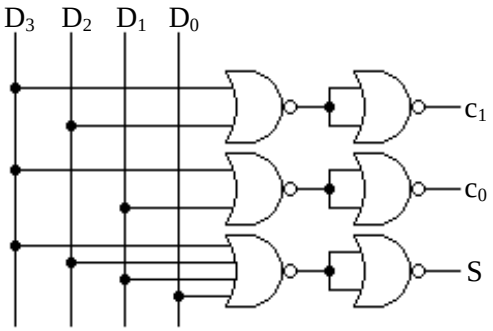
13.48 Najmanje 9, a najviše 16 ulaza.

13.49 $c_0 = \overline{\overline{D_3 D_3}\ \overline{D_1 D_1}}$ $\qquad$ $c_0 = \overline{\overline{D_3 \vee D_1} \vee \overline{D_3 \vee D_1}}$

$c_1 = \overline{\overline{D_3 D_3}\ \overline{D_2 D_2}}$ $\qquad$ $c_1 = \overline{\overline{D_3 \vee D_2} \vee \overline{D_3 \vee D_2}}$

$S = \overline{\overline{D_3 D_3}\ \overline{D_2 D_2}\ \overline{D_1 D_1}\ \overline{D_0 D_0}}$ $\qquad$ $S = \overline{\overline{D_3 \vee D_2 \vee D_1 \vee D_0} \vee \overline{D_3 \vee D_2 \vee D_1 \vee D_0}}$





Realizacija sa NOR logičkim kolima je ekonomičnija.

13.50 Rezultati analize sumirani su u sljedećoj tabeli:

Aktivni ulazi	c_1	c_0	Interpretacija
D_3, D_2	1	1	D_3
D_3, D_1	1	1	D_3
D_3, D_0	1	1	D_3
D_2, D_1	1	1	D_3
D_2, D_0	1	0	D_2
D_1, D_0	0	1	D_1
D_3, D_2, D_1	1	1	D_3
D_3, D_2, D_0	1	1	D_3
D_3, D_1, D_0	1	1	D_3
D_2, D_1, D_0	1	1	D_3
D_3, D_2, D_1, D_0	1	1	D_3

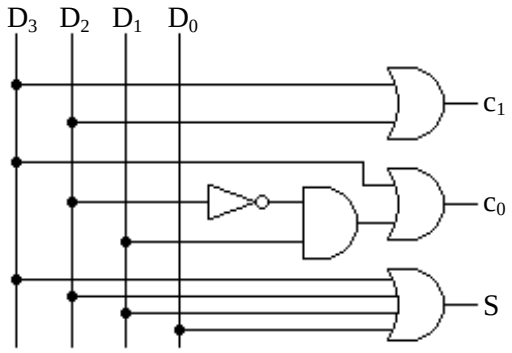
Kolona "Aktivni ulazi" označava ulaze koji su na logičkoj jedinici. Oznaka ulaza u koloni "Interpretacija" govori da se za datu kombinaciju ulaza koder ponaša kao da je od svih ulaza aktivan samo taj ulaz. Vidimo da se obični koder 4/2 ponaša skoro kao koder sa prioritetom, osim što se pri kombinaciji $D_2 = D_1 = 1$, $D_3 = D_0 = 0$ ponaša kao da je ulazu aktivan samo ulaz D_3, a ne D_2 kao što bi trebalo za koder sa prioritetom.

13.51

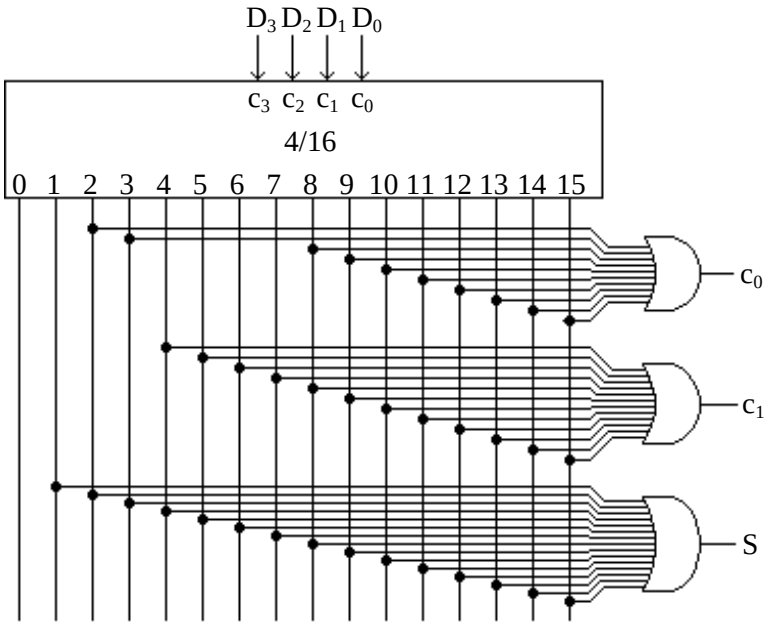
13.52 Na osnovu tablice istine, lako dolazimo do jednačina

$c_1 = D_3 \vee D_2$ $\quad c_0 = D_3 \vee \overline{D}_2 D_1$ $\quad S = D_3 \vee D_2 \vee D_1 \vee D_0$

na osnovu kojih neposredno slijedi prikazana realizacija:

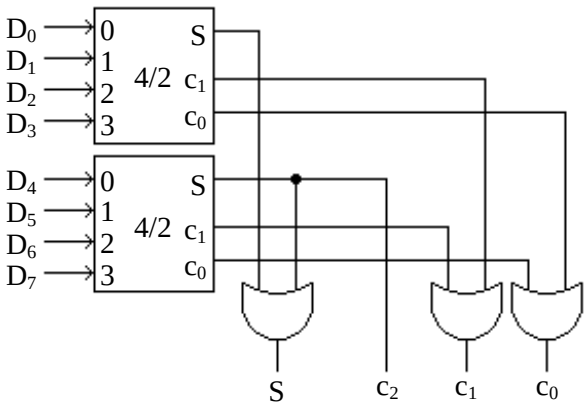


13.53 Tražena realizacija je prikazana na slici:

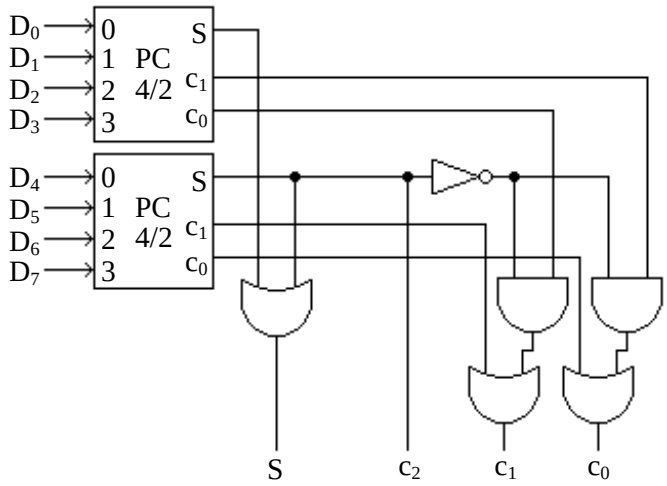


moze drugacije!

13.54



13.55



$c_2 = D_7 \vee D_6 \vee D_5 \vee D_4$

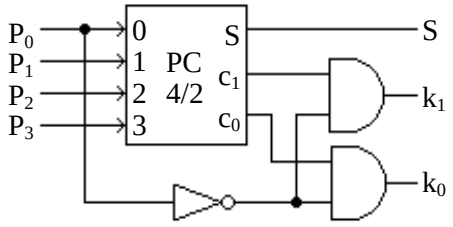
$c_1 = D_7 \vee D_6 \vee \overline{D_7 \vee D_6 \vee D_5 \vee D_4}\,(D_3 \vee D_2)$

$c_0 = D_7 \vee \overline{D}_6 D_5 \vee \overline{D_7 \vee D_6 \vee D_5 \vee D_4}\,(D_3 \vee \overline{D}_2 D_1)$

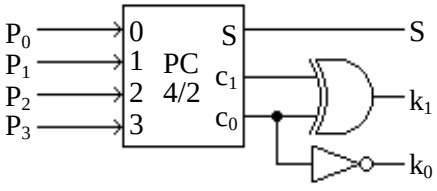
$S = D_7 \vee D_6 \vee D_5 \vee D_4 \vee D_3 \vee D_2 \vee D_1 \vee D_0$

13.56 koder s prioritetom

13.57 $k_1 = P_3\overline{P}_0 \vee P_2\overline{P}_0$, $k_0 = P_3\overline{P}_0 \vee \overline{P}_2\overline{P}_1\overline{P}_0$ ili $k_0 = P_3\overline{P}_0 \vee \overline{P}_2\overline{P}_0$ uz don't care uz pr.k.



alternativa $k_0 = \overline{c}_0$, $k_1 = c_1 \oplus c_0$



ZADACI ZA 19

- (Shift Register Design) Design the basic cell of a universal shift -register to the following specifications. The internal storage elements will be positive edge-triggered D flip-flops. Besides the clock, the shifter stage has two external control inputs, S0 and S1, and three external data inputs, SR, SL, and DI. SR is input data being shifted into the cell from the right, SL is data being shifted from the left, and DI is parallel load data. The current value of the flip-flop will be replaced according to the following settings of the control signals: S0 = S1 = 0: replace D with DI; S0 = 0, S1 = 1: replace D with SL; S0 = 1, S1 = 0: replace D with SR; S0 = S1 = 1: hold the current state. Draw a schematic for this basic shifter cell.

- Shifters are normally used to shift data in a circular pattern (the data that shifts out at one end of the shifter is shifted back into the other end), or as a logic shift (fill the shifted positions with 0's) or an arithmetic shift (propagate the high-order sign bit to the right or shift in 0's to the left). For example, if a 4-bit register contains the data 1110, the effects of the six kinds of shifts are the following:
  Circular shift right: 1110 becomes 0111.
  Circular shift left: 1110 becomes 1101.
  Logical shift right: 1110 becomes 0111.
  Logical shift left: 1110 becomes 1100.
  Arithmetic shift right: 1110 becomes 1111.
  Arithmetic shift left: 1110 becomes 1100.
  Show how to wire up a 4-bit universal shift register (TTL component 74194) to perform the following kinds of shifts:
  Circular shift right; Circular shift left; Logic shift right; Logic shift left ;
  Arithmetic shift right; Arithmetic shift left

- (Shift Register Design) Your task is to design a shift register -subsystem based on the TTL 74194 component that can implement the six kinds of shifts described in Exercise 7.2 (prošli zadatak). The -subsystem has three control inputs, S2, S1, S0 that are interpreted as follows: S2, S1, S0 = 000 is hold; 001 is circular shift right; 010 is circular shift left; 011 is logic shift right; 100 is logic shift left; 101 is arithmetic shift right; 110 is arithmetic shift left; 111 is parallel load. Show the data path for the shifter subsystem. You may use multiplexers at the shift inputs to the 74194.
  Show the combinational logic (equations or schematics) to decode the global S2, S1, S0 control inputs into the appropriate detailed control signals for the 74194 shifter and the external data path logic for handling the serial shift inputs.

- Nešto sa primjenom shiftreg za komunikaciju

- Design a 2-bit counter that behaves according to the two control inputs I0 and I1 as follows: I0, I1 = 0, 0: stop counting; I0, I1 = 0, 1: count up by one; I0, I1 = 1, 0: count down by one; I0, I1 = 1, 1: count by two. Draw the state diagram and state transition table.
  Implement the counter using T flip-flops, D flip-flops, and J-K flip-flops.
  Which choice of flip-flops leads to the minimum gate count? Assume that only two-input NAND, NOR, XOR, and XNOR gates are available. Draw the schematic for your minimum gate count implementation.
  Which choice of flip-flops leads to the minimum wire count? The same kinds of gates are available as in part (c). Draw the schematic for your minimum wire count implementation. Indicate how you have counted the interconnections.

- Design a three flip-flop counter that counts in the following sequence: 000, 010, 111, 100, 110, 011, 001, and repeat. Design the counter using toggle flip-flops. Verify that your implementation is self-starting.

- Consider the design of a 4-bit BCD counter that counts in the following sequence: 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, and then back to 0000, 0001, etc.
  Draw the state diagram and next-state table.
  Implement the counter using D flip-flops, toggle flip-flops, S-R flip-flops, and J-K flip-flops.
  Implement the counter making it self-starting just for the D flip-flop case.

- Consider the design of a 4-bit Gray code counter (that is, only one of the state bits changes for each -transition) that counts in the following sequence: 0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100, 1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000, and then back to 0000, 0001, 0011, etc.
  Draw a state diagram and next-state table.
  Implement the counter using D flip-flops, toggle flip-flops, R-S flip-flops, and J-K flip-flops.
  Do you have to worry about self-starting? Why or why not?

- The 4-bit Johnson counter advances through the sequence 0000, 1000, 1100, 1110, 1111, 0111, 0011, 0001, and repeat. Using the standard counter design process, show how to implement this count sequence using (a) D flip-flops and (b) T flip-flops. How does your solution compare with the J-K implementation of Figure 7.10 (standardni Johnson), in terms of gates and wiring -complexity?

- Analyze your solutions to Exercise 7.12(a) and (b) (pretprosli zadatak) to check whether or not they are self-starting. Draw complete state diagrams and show all states and transitions implied by your implementation.

- Figure 7.37 shows a 3-bit ripple up-counter (standardni). What simple change can you make to this circuit to convert it to a down-counter? Show your logic circuit and associated timing diagram. Explain why your circuit operates correctly.

- Malo primjera za testiranje samostartanja.

- Consider the design of a bit-serial adder. This circuit uses a single full adder to add two binary numbers presented in serial fashion, 1 bit at a time. Draw the schematic for a 4-bit version of this circuit. Two 4-bit shift registers are loaded with the data to be added in parallel. These are shifted out a bit at a time, starting with the lowest-order bit, into the A and B inputs of the full adder. The partial sum is shifted into a third register. How should the carry out be handled between subsequent bits? Define your control signals for the bit-serial adder subsystem. Draw a timing diagram that illustrates the sequencing of these signals to implement the 4-bit addition. What happens on reset?

- Flip-flops, registers, and counters can be used to implement a variety of useful clocking functions.
  Design a system that will generate a single clock pulse one period long each time a push-button is pressed (you may assume that an external reference clock is available).
  Design a system using an MSI counter that will assert a signal for exactly 13 clock pulses each time a push-button is pressed (once again, an external reference clock is available).
  In many applications, it is desirable to single step the clock as well as to halt a free-running clock signal and later restart it. Design a circuit that will generate a single clock pulse each time a STEP push-button is pressed. Provide a separate -circuit, independent of STEP, that will cleanly turn the clock on when a RUN switch is in the ON position and off when RUN is in its off position. No marginal/partial clock outputs are allowed.

- Design a 2-bit binary up-counter to the following specification. The counter has five inputs (not including the clock) and three outputs. The inputs are CLR, LOAD, COUNT, LB, and LA. CLR takes precedence over LOAD, which in turn takes precedence over COUNT. The outputs are B, A, and RCO. When CLR is asserted, the flip-flops of the counter in Exercise 7.24 are reset to 0. If LOAD is asserted (when CLR is not asserted), the flip-flops' contents are replaced by the LB and LA inputs. If COUNT is asserted (when CLR and LOAD are not asserted), the flip-flops' contents are

incremented: 00 becomes 01, 01 becomes 10, 10 becomes 11, and 11 becomes 00. When the counter is in state 11, RCO is asserted.

- Implement the counter in Exercise 7.24 using D flip-flops and as few logic components as possible. Hint: Use 4-to-1 multiplexers to implement your next-state function. You may also use XOR gates, as well as standard AND, OR, and NOT gates.

- A microprocessor with an 8-bit-wide data bus uses RAM chips of 4096 by 1-bit capacity. How many chips are needed and how should their address lines be connected to provide a memory capacity of 16 Kbytes (1 byte = 8 bits).

- The task is to create a complex counter that can count in binary or in Gray code, depending on the value of a mode input: "A synchronous 3-bit counter has a mode control input m. When m = 0, the counter steps through the binary sequence 000, 001, 010, 011, 100, 101, 110, 111, and repeat. When m = 1, the counter advances through the Gray code sequence 000, 001, 011, 010, 110, 111, 101, 100, and repeat."

- Dodati primjer: serijsko-paralelni množač sa shift registrom.

ZADACI ZA 20

- Objasniti interapte

- Varijacije na temu dizajna procesora(vidi CLC, exercises iz pogl. 11)