Uvod v programirljive digitalne sisteme

Andrej Trost
Univerza v Ljubljani
Fakulteta za elektrotehniko
http://lniv.fe.uni-lj.si/pds-uvod.html

Ljubljana, 2016

CIP - Kataložni zapis o publikaciji
Narodna in univerzitetna knjižnica, Ljubljana

004.312(078.5)(0.034.2)

TROST, Andrej, 1972-
Uvod v programirljive digitalne sisteme [Elektronski vir] / Andrej Trost. - 1. izd. - El. knjiga. - Ljubljana : Založba FE, 2016

Način dostopa (URL): http://lniv.fe.uni-lj.si/pds-uvod.html

ISBN 978-961-243-304-8 (pdf)

283859968

URL: http://lniv.fe.uni-lj.si/pds-uvod.html

Recenzenta: prof. dr. Andrej Žemva, doc. dr. Matej Možek
Založnik: Založba FE, Ljubljana
Izdajatelj: UL Fakuleta za elektrotehniko, Ljubljana
Urednik: prof dr. Sašo Tomažič

1. izdaja

Kazalo

Predgovor

Digitalne elektronske naprave nas obkrožajo na vsakem koraku. Kaj imajo skupnega pametna ura, digitalni fotoaparat, avtomobilski računalnik in industrijski krmilni sistem? To so računske naprave, ki imajo vgrajeno digitalno elektroniko. Njihovo delovanje lahko spreminjamo s postopkom programiranja. Digitalne naprave so utemeljene s poenostavitvijo fizikalnih zakonov v preproste inženirske modele, ki temeljijo na matematični logiki. Nekaj mejnikov v razvoju logičnih vezij so postavili:

- George Boole z matematičnimi osnovami logike (1847),
- Nikola Tesla, ki patentira prva elektronska logična vrata (1898),
- Claude E. Shanon, ki uporabi Boolovo algebro za analizo in izdelavo logičnih vezij (1937).

Digitalne računske naprave so doživele razmah z razvojem elektronike, še posebej z izumom integriranega vezja, ki je omogočil izdelavo mikroprocesorjev in miniaturizacijo računalnikov. Digitalni sistemi so danes vgrajeni v množico osebnih, gospodinjskih, transportnih in industrijskih naprav. Poznavanje osnov digitalnih elektronskih sistemov je pomembno za študenta elektrotehnike, saj je zelo verjetno, da se bo v praksi srečal s snovanjem ali programiranjem digitalnih naprav.

Pomen oznak pri posameznih poglavjih:

 svinčnik označuje teoretično razlago, ki je obvezni del snovi;

 spajkalnik nakazuje praktične primere in vaje;

integrirano vezje označuje podrobnejšo razlago zgradbe vezja.

Zahvala

Najprej bi se zahvalil sodelavcu *Andreju* za diskusije o vsebini predmeta Programirljivi digitalni sistemi in ostalim sodelavcem s Fakultete za elektrotehniko. Zahvaljujem se svoji ženi *Marini* za koristne nasvete pri prvem branju gradiva in prijatelju *Petru* za jezikovni pregled in popravke.

1 Uvod v digitalna vezja

1.1 Elektronska vezja

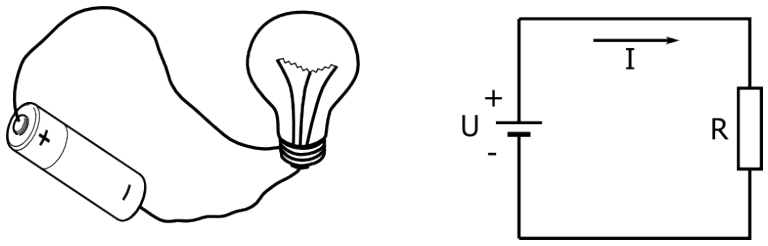
Električno vezje je povezava izvorov električne energije, vodnikov in komponent. Vezje imenujemo tudi tokokrog, saj v njem teče tok iz pozitivnega priključka napajalnega vira prek komponent proti negativnemu priključku vira. Če se ta povezava prekine, vezje ne more opravljati svoje naloge. Sodobna električna vezja vsebujejo veliko različnih vrst komponent, kot so upori, kondenzatorji, tuljave, polprevodniške diode, tranzistorji, integrirana vezja in pretvorniki, npr. mikrofon, senzorji, motorji, grelci, svetila.

Komponente so med seboj povezane z vodniki, ki prenašajo tok med posameznimi točkami v vezju. Električni vodnik, npr. bakrena žica, tvori skupaj s priključki komponent povezavo v vezju. Električna aktivnost na povezavi se prenese na vse komponente, priključene na to povezavo. Nekatere povezave so namenjene prenosu električne energije na komponente, druge pa prenašajo podatke. Povezave, ki prenašajo podatke, se imenujejo *signalne povezave*. Po njih tečejo majhni tokovi in so iz vodnikov majhnega preseka. Povezave za prenos električne energije imenujemo tudi *napajalne povezave* in so močnejše, ker morajo prenašati večje tokove.

Električna vezja izvajajo naloge z uporabo električne energije, npr. poženejo motor ali prižgejo luč. *Elektronska vezja* pa so sestavljena iz komponent, ki jih krmilijo električni signali. Večina sodobnih elektronskih vezij uporablja signale z napetostjo nekaj voltov glede na maso. Analogna vezja prenašajo podatke v obliki napetostnega nivoja signala ali velikosti toka. Primer je temperaturni senzor, ki pretvarja temperaturo v padec napetosti. Analogne podatke teoretično določimo poljubno natančno (npr. temperatura $20.213657...°C$), v praksi pa nas omejuje šum, ki je vedno prisoten. *Digitalna vezja* uporabljajo le nekaj napetostnih nivojev za prenos podatkov v številski obliki in so zaradi tega manj občutljiva na šum in motnje iz okolice.

Na podlagi proučevanja električnih pojavov so znanstveniki oblikovali fizikalne zakone. Zakone elektromagnetizma opisujejo zelo splošne in za praktično uporabo precej zapletene fizikalne enačbe. Inženirji se ukvarjamo s praktično uporabo izsledkov znanosti. Elektroinženirji uporabljamo poenostavljene *modele* in elemente, ki pod določenimi pogoji dovolj dobro opisujejo dogajanje v električnih vezjih. Poenostavljeni ali abstraktni modeli omogočajo učinkovitejšo obravnavo električnih vezij.

Analiza delovanja modela vezja je hitrejša in cenejša kot izdelava fizičnega vezja. V nekaterih primerih lahko nepravilno delovanje vezja predstavlja nevarnost za človeka ali okolico, zato je pri razvoju novega vezja bolje narediti analizo na modelu vezja.



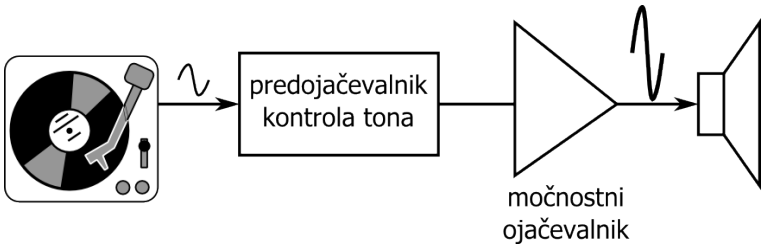
Slika 1.1: Vezje (tokokrog) in električna shema modela vezja.

Slika 1.1 prikazuje preprosto vezje z baterijo in žarnico na levi strani ter elektrotehniški model vezja na desni strani. V modelu vezja smo uporabili več poenostavitev: baterijo predstavlja idealen vir napetosti, povezovalne žice so idealni prevodniki, žarnico pa smo predstavili z uporom. Osnovni princip izdelave modelov električnih vezij je, da komponente vezja zamenjamo z diskretnimi *elementi*, ki jih opišemo z enim parametrom: baterijo z napetostjo, žarnico pa z upornostjo. S takim modelom lahko naredimo preprost izračun: ugotovimo, kakšen tok bo tekel pri določeni napetosti baterije in kakšna moč se bo trošila na žarnici.

Model vezja, ki ga predstavlja *električna shema*, je za inženirje elektronike osnovna dokumentacija za izdelavo fizičnega vezja. Shematski model vezja prikazuje elemente, signalne in napajalne povezave v vezju. Model lahko skiciramo, pregledujemo in analiziramo poljubno mnogokrat in precej hitreje in ceneje, kot bi to delali na realnem vezju. Z računalniškimi programi lahko analiziramo delovanje vezja s pomočjo simulacije do poljubne natančnosti pred fizično izdelavo. Danes se praktično vsa vezja začnejo načrtovati v obliki računalniških modelov v orodjih CAD (angl. Computer Aided Design). Model vezja s temi orodji hitro sestavimo in ga preučimo pred fizično izdelavo, kar prihrani veliko časa in stroškov. Zavedati pa se moramo, da računalniški model ni pravo vezje in da je dober toliko, kolikor točne so bile predpostavke načrtovalcev. Računalniški model vezja je lahko v grafični obliki (shema ali diagram) ali pa je narejen z opisom v nekem jeziku. Grafični zapis je lažje berljiv in primernejši za manjša vezja ali pa grobo blokovno predstavitev velikih vezij. Jezikovni opis je učinkovitejši pri kompleksnih vezjih in primeren za avtomatsko obdelavo v računalniku.

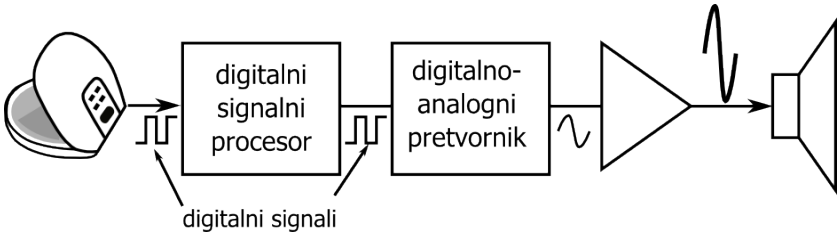
1.2 Analogna in digitalna vezja

Električni signali, ki predstavljajo fizikalne količine, so večinoma analogni in lahko zavzamejo poljubno vrednost znotraj nekega območja. Primer takšnega signala je električna napetost na mikrofonu, ki je sorazmerna pritisku zvočnih valov. Zvočni (avdio) signal prenašamo in obdelujemo v analogni ali digitalni obliki, ki ima pred analogno več prednosti.



Slika 1.2: Analogni predvajalnik glasbe: klasični gramofon.

Klasični gramofon je primer analognega predvajalnika glasbe. Avdio signal zajemamo z iglo, ki drsi po gramofonski plošči in pretvarja nihanje v majhno električno napetost. Analogni električni signal najprej potuje na predojačevalnik in enoto za kontrolo tona, nato pa na močnostno ojačevalno stopnjo, s katero dobimo dovolj močan signal za predvajanje na zvočniku. Za kvalitetno predvajanje zvoka na analognem gramofonu je zelo pomembno, da se plošča vrti enakomerno, da ni napak ob zajemu signala in da se čim manj šuma pojavi v vezju ter prevaja skozi ojačevalnik. Šum je v električnih vezjih vedno prisoten in predstavlja največjo težavo pri načrtovanju analognih elektronskih naprav.

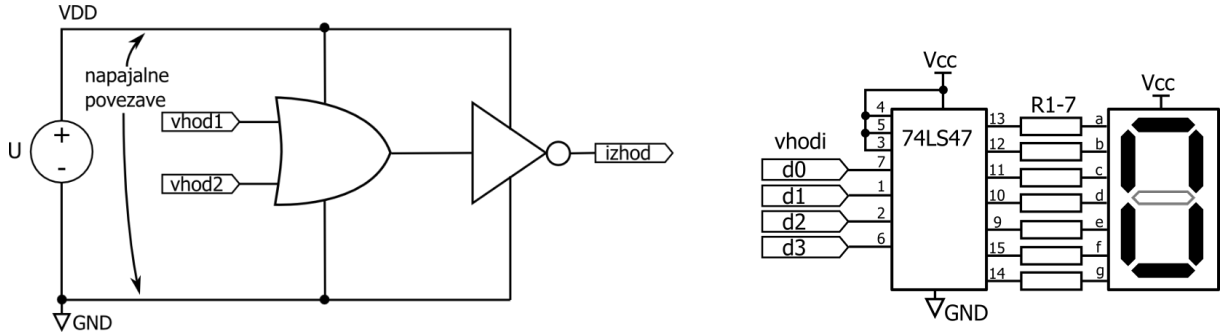


Slika 1.3: Digitalni predvajalnik glasbe.

Digitalni predvajalnik glasbe ima shranjen zvok v obliki številskih podatkov na zgoščenki ali v elektronskem pomnilniku. Podatke sprejme in obdela digitalni signalni procesor. Na izhodu procesorja dobimo digitalne vzorce zvoka, ki jih preko preko digitalno-analognega pretvornika spremenimo v zvočni signal, ojačimo in predvajamo na zvočniku, kot prikazuje slika 1.3.

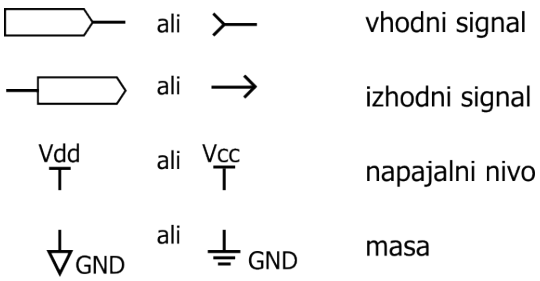
Na zgoščenki ni zapisana amplituda zvočnega signala, ampak matematična koda, ki jo *dekodira* digitalni procesor. Če je prišlo pri branju do manjših napak, bo postopek dekodiranja še vedno uspešno določil shranjeno amplitudo zvoka. Kodirani podatki so lahko shranjeni v zgoščeni obliki (npr. mp3), kar omogoča zapis daljših posnetkov na ploščo. Digitalni signal ni občutljiv na šum in ga enostavno shranjujemo, tako da tudi ob neenakomernem branju podatkov s plošče ali prenašanju prek spleta z ustrezno digitalno obdelavo poslušamo kvaliteten zvok.

Digitalna vezja so sestavljena iz logičnih elementov, blokov, napajalnih povezav in signalnih povezav. Vhodni logični elementi in bloki so po dogovoru postavljeni na levi strani simbola, izhodni priključki pa so na desni strani. Posamezne vhode logičnih elementov lahko povežemo skupaj, če prihaja signal na več vhodov iz istega vira. Spoji med povezavami so na shemi označeni s polnimi črnimi krožci. Povezav z izhodov logičnih elementov ne smemo vezati skupaj, ker bi naredili kratek stik. Vhodni in izhodni priključki celotnega vezja so označeni s posebnimi simboli, ki so prikazani na sliki 1.5.



Slika 1.4: Primer digitalnega vezja z logičnimi elementi in bloki.

Digitalno vezje potrebuje napajanje s konstantnim in stabilnim napetostnim virom za vse elemente. Pozitivni napajalni priključek ima v shemah digitalnih vezij oznako V_{dd} ali V_{cc}. Drugi pol je napajalna masa z oznako GND (angl. ground) in predstavlja referenco, proti kateri izmerimo vse napetosti v vezju. Električna shema z večjim številom elementov in posledično veliko napajalnih povezav postane nepregledna, zato prikažemo le oznake za napajanje in maso na ustreznih priključkih. Priključek mase označimo z navzdol obrnjenim trikotnikom in oznako GND, priključek pozitivnega pola pa s kratko črto, nad katero je oznaka.

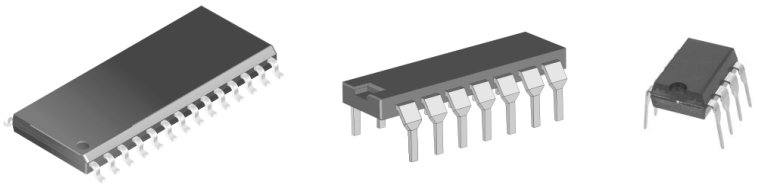


Slika 1.5: Oznake posebnih povezav na shemah digitalnih vezij.

Poenostavljene logične sheme prikazujejo le povezavo logičnih gradnikov, brez napajalnih signalov. Poenostavljene sheme so preglednejše in jih bomo uporabljali v knjigi za prikaz osnovnih vezav logičnih elementov in gradnikov.

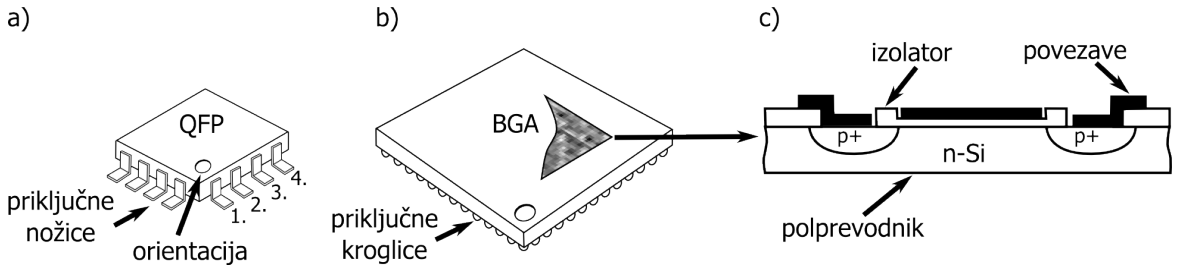
1.3 Digitalna integrirana vezja

Digitalna vezja so narejena iz komponent, ki so najpogosteje v obliki *integriranega vezja*. Integrirano vezje je miniaturno elektronsko vezje, narejeno na skupni osnovi (substratu) iz polprevodniškega materiala. Označujemo ga tudi z besedo čip, ki izhaja iz računalniškega slenga in pomeni silicijevo rezino. Silicij je polprevodniški material, na katerem z dodajanjem primesi naredimo različne elektronske komponente, npr. upore, kondenzatorje, tranzistorje in jih z naparjenimi prevodnimi plastmi povežemo med seboj. Digitalni čipi opravljajo različne logične funkcije, delimo pa jih tudi glede na obliko ohišja in tehnološko izvedbo.



Slika 1.6: Integrirana vezja v ohišju s priključki v dveh vrstah.

Integrirano vezje je precej manjše od vezja, sestavljenega iz posameznih diskretnih komponent. Vezje na substratu je vgrajeno v ohišje, iz katerega izhajajo priključne nožice (angl. pin) za zunanje signale in je primerno za montažo na tiskanem vezju. Manjša integrirana vezja imajo zunanje priključke razporejene v dveh vrstah. Takšna ohišja so označena s kratico DIL (angl. Dual In-Line) ali SO (angl. Small Outline).



Slika 1.7: Integrirano vezje v ohišju QFP (a), BGA (b) in presek plasti na silicijevi rezini (c).

Digitalna integrirana vezja imajo lahko zelo veliko zunanjih priključkov, ki so razporejeni na vseh štirih stranicah (QFP, angl. Quad Flat Pack) ali pa v obliki matrike kroglic pod integriranim vezjem (BGA, angl. Ball Grid Array).

Vezja z enako logično funkcijo in ohišjem imajo lahko različno tehnološko izvedbo, ki vpliva na lastnosti zunanjih signalov, porabo vezja in hitrost delovanja. V naslednjem poglavju bomo spoznali lastnosti signalov dveh danes najpogostejših tehnoloških izvedb digitalnih vezij: CMOS in LVCMOS.

1.4 Digitalni sistemi

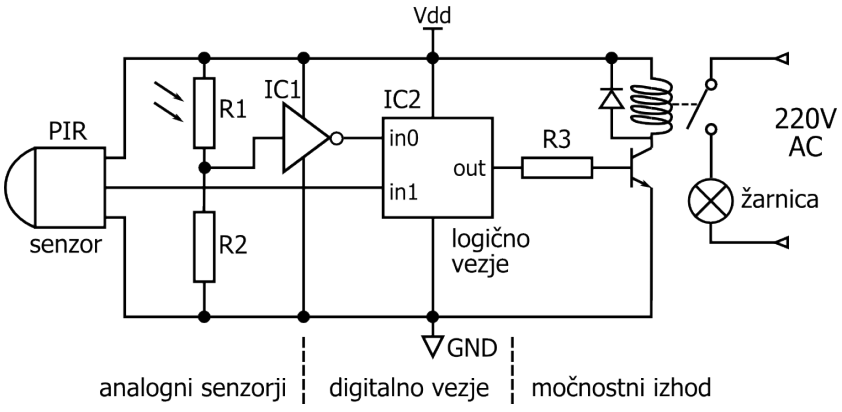
Digitalni sistemi so elektronski sistemi, ki izvajajo funkcije z uporabo digitalne logike. Na tej temeljijo današnji računalniki in mikroprocesorji, ki jih najdemo vgrajene v v prenosne telefone, avtomobilske krmilne sisteme, naprave zabavne elektronike itd. Vgrajeni sistemi (angl. embedded system) vsebujejo tudi analogne gradnike, prek katerih komunicirajo z okolico.

Slika 1.8: Vhodi in izhodi digitalnega sistema.

Vhodi v digitalni sistem prihajajo iz senzorjev, ki merijo fizikalne količine (npr. temperaturo, svetlost, jakost zvoka) in jih pretvarjajo v električni signal. Analogno digitalni pretvornik (A/D) pretvori ta signal v obliko, primerno za vhode digitalnega vezja. Tipke ali stikala pa povežemo neposredno na digitalne vhode.

Z digitalnimi izhodi krmilimo prikazovalnike in indikatorje, analogni izhod za aktuatorje pa dobimo prek digitalno analognega pretvornika (D/A). Aktuatorji pretvorijo električni signal nazaj v neko fizikalno količino (toploto, svetlobo, zvok ipd.).

Vzemimo preprost primer digitalnega sistema za avtomatsko prižiganje luči v temi, ki ga prikazuje slika 1.9. Sistem ima na vhodu senzor gibanja in senzor svetlobe, na izhod pa je vezana žarnica. Žarnica naj se prižge le v primeru, ko prvi senzor zazna gibanje, senzor svetlobe pa javlja, da je noč.



Slika 1.9: Digitalni sistem za avtomatsko prižiganje luči.

Nekateri senzorji, kot npr. senzor gibanja, že vsebujejo celotno vezje z merilno elektroniko in digitalnim izhodom. Signal iz preprostega svetlobnega senzorja, fotoupora, pa moramo pretvoriti v digitalno obliko. V vezju smo uporabili napetostni delilnik in logična vrata IC1, ki delujejo kot

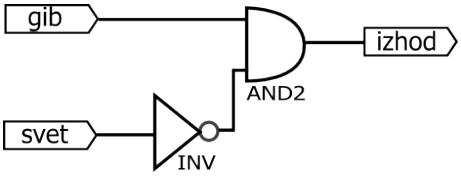
primerjalnik. Kadar je na napetost na vhodu manjša od nastavljenega praga, dobimo na izhod en logični nivo, sicer pa drugega.

Žarnica potrebuje omrežno napetost 220 V, zato jo priključimo na izhod logičnega vezja prek močnostnega pretvornika. Pretvornik lahko naredimo kot elektronsko stikalo z vezavo tranzistorja in releja.

1.5 Izvedba sistema

Pri izdelavi digitalnega vezja sistema za avtomatsko prižiganje luči potrebujemo dve logični komponenti: logični negator (IC1) in logična vrata AND (IC2). Logične komponente dobimo v obliki digitalnih integriranih vezij, ki izvajajo eno ali več osnovnih logičnih operacij. Digitalno vezje naredimo s povezavo več integriranih vezij. Digitalna integrirana vezja za osnovne logične operacije so zelo razširjena in jih dobimo v različnih tehnoloških izvedbah.

Če želimo spremeniti delovanje vezja, moramo dodati nove komponente in jih ponovno povezati, kar utegne biti precej zamudno. Tukaj nam rešitev ponujajo *programirljiva logična vezja*, ki so, podobno kot procesorji, univerzalni gradniki. Načrt vsebine programirljivega vezja pripravimo na računalniku, ga prevedemo in naložimo v vezje. Slika 1.10 prikazuje shematski načrt logičnega vezja, ki je pripravljen za prevajanje v tehnologijo programirljivih vezij.



Slika 1.10: Shema logičnega vezja za senzor gibanja.

Zgradbo in delovanje vezja lahko opišemo tudi s kodo, ki je podobna programom za mikroprocesorje. Kodo napišemo v enem izmed jezikov za opis strojne opreme (npr. VHDL ali Verilog), jo prevedemo in naložimo v programirljivo vezje. Primer opisa senzorja gibanja v jeziku VHDL:

```
entity senz is
   port ( gib   : in    std_logic;
          svet  : in    std_logic;
          izhod : out   std_logic);
end senz;

architecture opis of senz is
begin
 izhod <= gib and (not svet);
end opis;
```

Mikroprocesor

Opravila, ki jih izvaja digitalni sistem, lahko napišemo v obliki programa za mikroprocesor. Načrtovalci digitalnih sistemov radi posegajo po mikroprocesorjih, ker so to zelo razširjene komponente in za večino opravil najdemo na tržišču primeren procesor. Najenostavnejši procesorji za vgrajene naprave so v obliki integriranega vezja z digitalnimi vhodnimi in izhodnimi vmesniki in se imenujejo *mikrokrmilniki*. Vsebujejo pomnilnik, različne vrste digitalnih vmesnikov in analognih pretvornikov za enostavno povezavo s senzorji in aktuatorji.

Mikrokrmilnik je na shemi predstavljen kot gradnik IC2 z dvema digitalnima vhodoma in0 in in1 ter enim izhodom (out). Na vhod in0 je pripeljan digitalni signal iz svetlobnega senzorja, na in1 pa signal iz senzorja gibanja. Delovanje digitalnega sistema opišemo s kratkim programom v jeziku C, ki je danes najbolj razširjen način opisa delovanja procesorjev. Program prevedemo v strojno kodo in ga naložimo v mikrokrmilnik. Naredimo preprost program, ki postavi izhod na 1, ko sta oba vhoda in0 in in1 postavljena na 1, sicer pa naj bo izhodni signal 0. Program se izvaja v neskončni zanki `while(1)`, v kateri najprej preberemo vrednosti z vhodov, nato pa izračunamo in nastavimo izhod:

```
while (1) {
   gib = digitalRead(in0);
   svet = digitalRead(in1);

   if (gib & svet) digitalWrite(out, 1);
   else digitalWrite(out, 0);
}
```

Če želimo narediti spremembo v delovanju, npr. dodati ugašanje luči z zakasnitvijo, moramo dopolniti, prevesti in na novo naložiti program. Izvedbe digitalnih sistemov s procesorji so zelo prilagodljive in zaradi tega tudi zelo popularne.

Razvojni sistemi

Razvojni sistemi so vnaprej pripravljena vezja s komponentami, ki jih oblikujemo s postopkom programiranja. Mikroprocesorski razvojni sistemi vsebujejo mikroprocesor s pomnilnikom in vmesnikom za prenos programske kode iz računalnika v pomnilnik procesorja.

Programirljivi razvojni sistemi omogočajo, da vezje razvijemo na računalniku in s programiranjem oblikujemo povezave na fizičnem vezju. To je zelo učinkovit in sodoben postopek prototipne izdelave vezij, ki ga bomo spoznali pri praktičnem delu.

1.6 Tiskano vezje

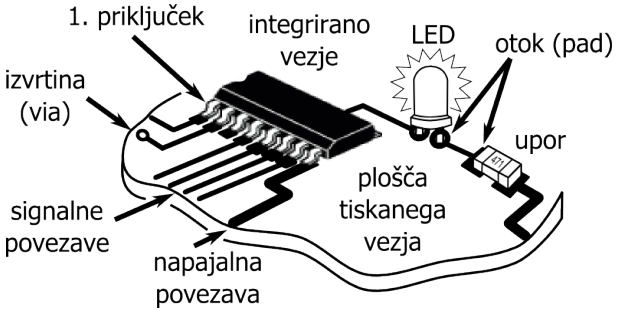
Čeprav lahko komponente elektronskega vezja med seboj povežemo s prevodnimi žicami, je to le redko praktičen in primeren način izdelave vezja. Integrirana vezja v ohišju DIL povezujemo na prototipni plošči z luknjicami. Takšen način povezovanja se uporablja predvsem za prototipno

izdelavo manjših vezij, kjer na vezju izvajamo testiranja in meritve v procesu načrtovanja. Vezja na prototipni plošči niso primerna za masovno proizvodnjo.

Običajno so komponente sestavljene in povezane na plošči z imenom *tiskano vezje*. Priprava in izdelava tiskanega vezja sicer zahteva nekaj več časa kot preprosto povezovanje komponent s prevodniki, vendar dobimo na koncu izdelek, ki je primeren za vgradnjo v napravo.

Proizvedena tiskana vezja morajo imeti zanesljive in permanentne povezave med vsemi komponentami. Poleg tega mora biti vezje poceni in primerno za masovno izdelavo. Vezja so danes večinoma narejena na neprevodni plošči (npr. vitroplast plošče iz epoksi in steklenih vlaken) z bakrenimi povezavami. Komponente so pritrjene in povezane s postopkom spajkanja.

Proizvodnja tiskanih vezij se začne z $1-2\,\mathrm{mm}$ debelimi ploščami, ki so iz obeh strani prevlečene z bakreno folijo. Na bakreni plošči natisnemo vzorec, ki preprečuje jedkanje na mestih s povezavami (od tod ime tiskano vezje, angl. *Printed Circuit Board*). Plošče nato potopimo v kislino, ki odstrani odvečni baker, tako da ostanejo le povezave in otoki (angl. *pads*) za spajkanje komponent. Nekatere komponente imajo priključke, ki gredo skozi tiskano vezje, za katere naredimo izvrtine, novejše komponente pa so pritrjene kar na površini tiskanega vezja (angl. *Surface Mount Devices*). Tiskana vezja imajo eno, dve ali več bakrenih plasti ali slojev. Večslojna tiskana vezja so narejena z lepljenjem izdelanih dvoplastnih vezij v sendviče. Preprosta elektronska vezja so dvo- ali štiriplastna, zapletena računalniška plošča pa ima tudi več kot 20 plasti.



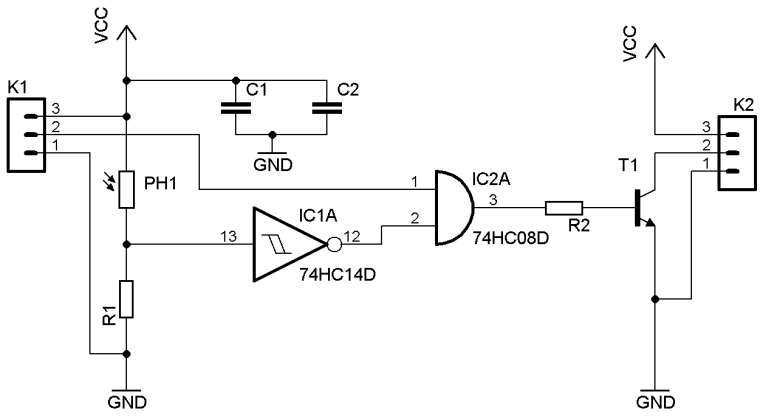
Slika 1.11: Tiskano vezje s površinsko montiranimi elektronskimi elementi.

Nekatere izvrtine v vezju so namenjene za povezave med plastmi (angl. *via*), zato jih v postopku galvanizacije prekrijemo s tankim kovinskim slojem. Zunanje bakrene plasti zaščitimo pred oksidacijo z lakom, spajkalni otoki pa morajo ostati odkriti. Tej plasti pravimo maska za spajkanje ali stop maska. Bakrene spajkalne otoke zaščitimo pred oksidiranjem s postopkom kositranja, na koncu pa natisnemo napise na tiskani plošči s sitotiskom. Napisi označujejo ime komponente, njihove vrednosti in orientacijo za lažje sestavljanje in testiranje vezja. Orientacija komponent z več priključki je določena tako, da je posebej označen prvi priključek (s tiskano oznako, številko ali kvadratno obliko otoka).

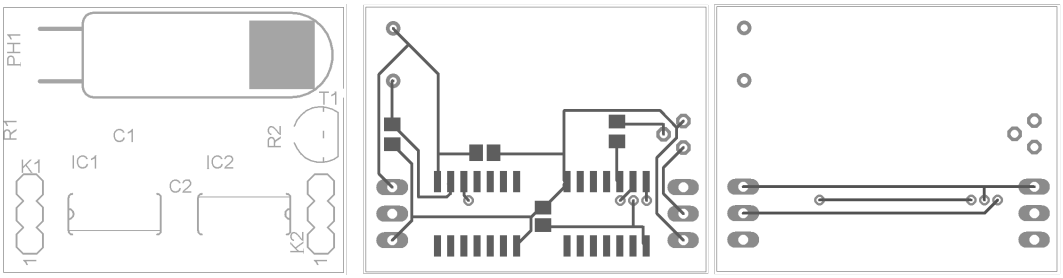
Elektronske komponente imajo izpostavljene kovinske priključke, s katerimi jih pritrdimo na tiskano vezje, tako da je komponenta mehansko pritrjena in električno povezana. To storimo v procesu spajkanja, pri katerem s staljeno kovino (spajko) oblijemo priključek na komponenti in na tiskanem vezju. Ko se spajka ohladi, se strdi in tvori trdno prevodno vez.

Naloga

1. Preglej shemo vezja za avtomatsko prižiganje luči in določi digitalne elemente, napajalne povezave in signalne povezave. Poišči na načrtu tiskanega vezja vsa integrirana vezja in pasivne komponente, kot so upori ali kondenzatorji. Ali vezje vsebuje kakšne digitalne vhode in izhode? Ali so na vezju kakšni senzorji in izhodi za aktuatorje?



Slika 1.12: Shema vezja za avtomatsko prižiganje luči v programu Eagle.



Slika 1.13: Načrt tiskanega vezja (sitotisk, zgornja in spodnja plast).

2
Digitalni signali

Digitalni signali so signali, ki lahko zavzamejo le končno število različnih stanj. Imenujemo jih tudi *diskretni signali*. Preprost primer digitalnega signala je število dvignjenih prstov na roki – število je lahko le ena izmed vrednosti iz diskretnega območja med 0 in 10. Beseda *digitalni* prihaja iz latinskega izraza digitus, ki pomeni prst. V tem poglavju bomo predstavili digitalne signale v obliki dvojiškega zapisa in napetostnih nivojev v elektronskem vezju.

2.1 Binarni signali

V digitalnih elektronskih vezjih se največkrat uporabljajo *binarni* signali, ki lahko zavzemajo le dve možni stanji, označeni kot:

- napačno (false) ali pravilno (true),
- nizko (potencial V_L) ali visoko (V_H),
- številka 0 ali 1.

Dve stanji opisujeta preproste pojave, kot so prižgana oziroma ugasnjena žarnica ali stikalo. Predstavljata lahko logično trditev, ki je pravilna ali napačna. Zapis stanj v obliki pravilno (true) ali napačno (false) je primeren za obravnavo vezij, ki izvajajo odločitvene operacije. Primer preproste logične operacije je negacija: napačno stanje spremenimo v pravilno in obratno. Element, ki izvaja logično negacijo, imenujemo logični negator ali inverter.

Najpogostejši zapis digitalnih stanj je v obliki številskih vrednosti 0 ali 1. Takšen zapis ni samo najkrajši, ampak je tudi primeren za računanje, saj digitalna vezja velikokrat izvajajo računske operacije. Vrednost 0 ali 1, ki jo zavzame enostaven signal, imenujemo binarna števka

ali *bit* (angl. binary digit). Vodila v digitalnih vezjih pa prenašajo večbitne vrednosti v dvojiškem zapisu. Za razumevanje dvojiških vrednosti si najprej poglejmo, kako so sestavljena večmestna desetiška števila. Vrednost desetiškega števila lahko zapišemo kot vsoto števk, pomnoženih s koeficienti potence 10. Enice množimo z 10^0, desetice z 10^1, stotice z 10^2 itn. Primer:

$$523_{(10)} = 5 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0$$

Popolnoma enako velja za dvojiška števila, le da pri izračunu desetiške vrednosti uporabimo potence števila 2. Posamezne števke dvojiškega števila pomnožimo s potencami števila 2 in seštejemo. Pri zapisu celega števila množimo skrajno desno dvojiško števko z 2^0, naslednjo z 2^1 in tako dalje. Poglejmo si primer izračuna desetiške vrednosti 4-bitnega števila:

$$1100_{(2)} = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 8 + 4 = 12_{(10)}$$

Pretvorbo najlažje naredimo tako, da nad vsako števko zapišemo ustrezno potenco števila 2 in seštejemo tiste potence, pod katerimi je binarna števka 1:

$$\underset{1}{\overset{8}{}}\underset{1}{\overset{4}{}}\underset{0}{\overset{2}{}}\underset{0}{\overset{1}{}}_{(2)} = 8 + 4 = 12$$

Prvi digitalni mikroprocesor Intel 4004 je računal s 4-bitnimi vrednostmi, ki v desetiškem sistemu pokrijejo območje le ene desetiške števke. Če bi želeli računati z dvomestnimi desetiškimi števili (vrednosti med 0 in 99), bi potrebovali 7-bitni dvojiški zapis z območjem med 0 in 127. Pri delu z dvojiškimi signali je koristno, če znamo potence števila 2 na pamet. Te določajo zgornjo mejo območja vrednosti, kot prikazuje tabela 2.1.

N	N-bitna števila	2^N	območje vrednosti
2	00–11	4	0–3
3	000–111	8	0–7
4	0000–1111	16	0–15
5	00000–11111	32	0–31
6	000000–111111	64	0–63
7	0000000–1111111	128	0–127
8	00000000–11111111	256	0–255
9	000000000–111111111	512	0–511
10	0000000000–1111111111	1024	0–1023

Tabela 2.1: Območja vrednosti binarnih pozitivnih števil.

Obseg vrednosti N-bitnega števila izračunamo s potenco 2^N. Do sedaj smo se ukvarjali z naravnimi števili, ki so samo pozitivna. Če želimo predstaviti cela števila, ki so pozitivna in negativna, moramo dodati še en bit za predznak. Ker je delo z velikimi dvojiškimi vrednostmi nerodno, se v modelih digitalnih vezij uporablja *šestnajstiški* zapis, ker je precej kompakten in nudi enostavno pretvorbo v dvojiškega. Vsaka šestnajstiška števka je zapisana z natanko štirimi dvojiškimi, kot prikazuje tabela 2.2. Dvojiški zapis pretvorimo v šestnajstiškega tako, da združujemo in pretvarjamo po 4 števke hkrati.

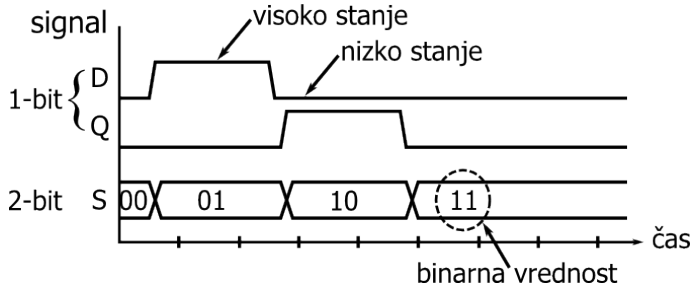
Primer pretvorbe 8-bitne vrednosti:

$$\begin{array}{cccc} 8 & 4 & 2 & 1 \\ \hline 0 & 0 & 0 & 1 \end{array} \quad \begin{array}{cccc} 8 & 4 & 2 & 1 \\ \hline 1 & 1 & 0 & 0 \end{array}_{(2)} = 1_{(10)} \quad 12_{(10)} = 1C_{(16)}$$

b_3	b_2	b_1	b_0	desetiško	šestnajstiško
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	2	2
0	0	1	1	3	3
0	1	0	0	4	4
0	1	0	1	5	5
0	1	1	0	6	6
0	1	1	1	7	7
1	0	0	0	8	8
1	0	0	1	9	9
1	0	1	0	10	A
1	0	1	1	11	B
1	1	0	0	12	C
1	1	0	1	13	D
1	1	1	0	14	E
1	1	1	1	15	F

Tabela 2.2: Celoštevilsko kodiranje 4-bitnih binarnih vrednosti.

Potek signalov, ki se spreminjajo s časom, opazujemo na časovnem diagramu (angl. waveform). Običajno opazujemo več signalov, ki jih vrišemo v en diagram z enako časovno skalo za vse signale. Za opazovanje signalov, ki v vezju zelo hitro spreminjajo stanja, uporabimo ustrezen merilni inštrument: osciloskop ali logični analizator.

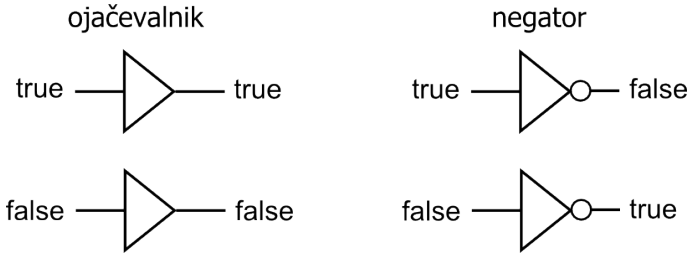


Slika 2.1: Časovni diagram digitalnih signalov.

Večbitne signale predstavimo v časovnem diagramu na traku, v katerem so zapisane trenutne vrednosti signala. Programska oprema za prikazovanje časovnega diagrama omogoča nastavitev prikaza v binarni obliki ali dekodirani decimalni, šestnajstiški, znakovni ASCII ipd.

2.2 Logični napetostni nivoji

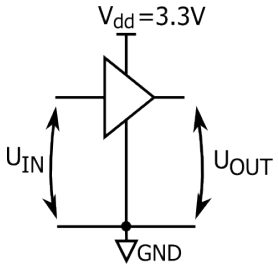
Gradniki digitalnih vezij imajo vhodne priključke, na katerih zaznavajo stanje binarnih signalov in izhodne priključke, na katere vsiljujejo binarne vrednosti. Za primer vzemimo dva najbolj enostavna gradnika, ki imata le en vhod in izhod: logični ojačevalnik in negator. Ojačevalnik ima na izhodu enako logično stanje, kot je na vhodu, pri negatorju pa je stanje na izhodu obrnjeno.



Slika 2.2: Logični ojačevalnik in logični negator.

V elektronskem vezju predstavlja logično stanje potencial na priključku oziroma napetost priključka proti masi. Stanje logične ničle je določeno z nizkim (V_L) potencialom, stanje logične enice pa z visokim (V_H) potencialom signala oziroma nizko in visoko napetostjo proti masi.

Slika 2.3 prikazuje napetosti na logičnem ojačevalniku pri nizkem in visokem stanju na vhodu. Iz priložene tabele vidimo, da predstavljajo različne napetosti enako logično stanje, zato potrebujemo dogovor o logičnih napetostnih nivojih.

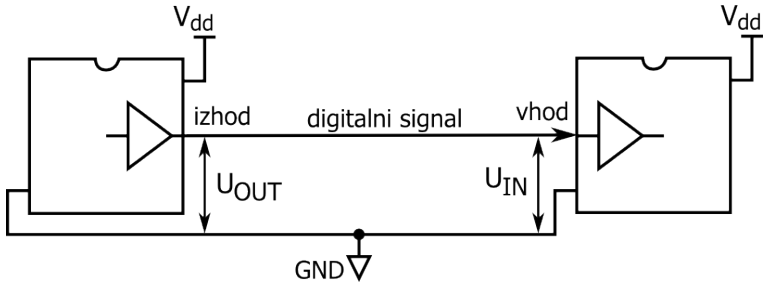


	U_{IN} [V]	U_{OUT}[V]	
nizka (V_L)	0.0	0.0	(V_L)
	0.5	0.0	
visoka (V_H)	3.0	3.3	
	3.3	3.3	(V_H)

Slika 2.3: Potenciali in napetosti na vhodu in izhodu ojačevalnika.

Osnovni gradniki digitalnih vezij se obnašajo kot preprosta elektronska stikala, ki preklapljajo med potencialom V_L in V_H. Elektronska stikala so bila včasih narejena z releji ali elektronkami, danes pa z različnimi elementi v polprevodniški tehnologiji. Z razvojem elektronike se spreminjajo tudi osnovni elementi in njihove električne lastnosti. Da bi digitalna vezja v različnih tehnologijah lahko povezali med seboj, moramo uvesti nek dogovor, ki določa potenciale za nizko in visoko stanje na vhodih in izhodih vezja. Vrednosti potencialov oz. napetosti gledamo v ustaljenem stanju, zato se dogovor imenuje statični red (angl. static discipline).

Elektronska stikala v digitalnih vezjih niso idealna, saj imajo neko upornost, ki povzroči, da visoko stanje V_H ni enako napajalnemu potencialu V_{dd} in da je nizko stanje V_L nekoliko višje od potenciala mase. Prav tako moramo upoštevati možne razlike v napajalnih napetostih

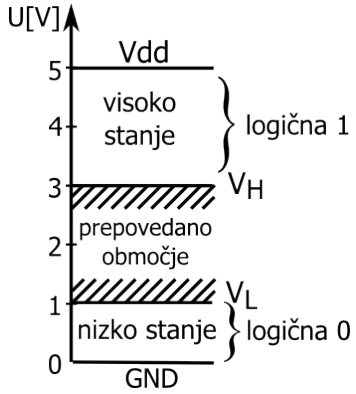


Slika 2.4: Povezava dveh digitalnih vezij.

integriranih vezij. Statični red omogoča pravilno interpretacijo signalov, ki potujejo med dvema digitalnima vezjema. Vzemimo najpreprostejši in pogost primer digitalne povezave, ko je izhod enega vezja vezan na vhod drugega, kot prikazuje slika 2.4. Enostaven dogovor bi lahko določal, da predstavljajo vse napetosti med V_{dd} in $V_{dd}/2$ visoko stanje (logično 1), napetosti med 0 in $V_{dd}/2$ pa nizko stanje (logično 0):

$$\begin{aligned} &\text{logična 0:} \quad 0\,\mathrm{V} \leq V_L \leq V_{dd}/2 \\ &\text{logična 1:} \quad V_{dd}/2 \leq V_H \leq V_{dd} \end{aligned}$$

Pri takšnem dogovoru se pojavi težava, če dobi sprejemnik na vhod napetost $V_{dd}/2$. Da bi lahko sprejemnik nedvoumno razločeval med logično 0 in 1, dodamo prepovedano območje potencialov na signalni liniji. Vpeljali bomo dva nova potenciala: V_{IH} je minimalni potencial, ki se na vhodu logičnega vezja interpretira kot visoko stanje, V_{IL} pa maksimalni, ki predstavlja nizko stanje.

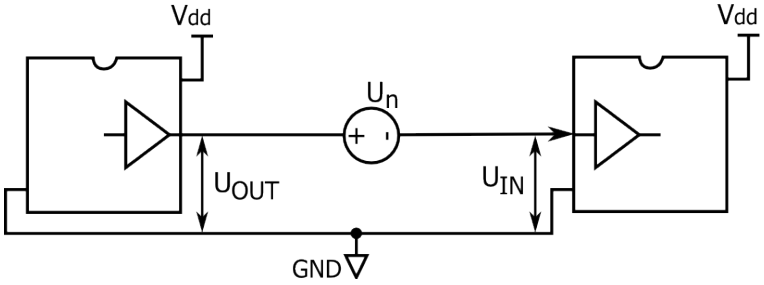


Slika 2.5: Dogovor o potencialih za nizko in visoko stanje.

Konkretne vrednosti so odvisne od tehnologije in zahtev – večje prepovedano območje poveča robustnost sistema, večje območje pravilnih stanj pa združljivost z več tehnologijami.

Na signalni povezavi se lahko inducira šum, ki ga modeliramo kot dodatno napetost U_n na signalni povezavi. Šum lahko povzroči neveljavno stanje na vhodu sprejemnika, kljub temu da je na oddajni strani stanje z veljavnim potencialom. Inducirani šum je v splošnem pozitivna ali negativna napetost. Denimo, da se na povezavi inducira $U_n = 200\,\mathrm{mV}$ šumne napetosti.

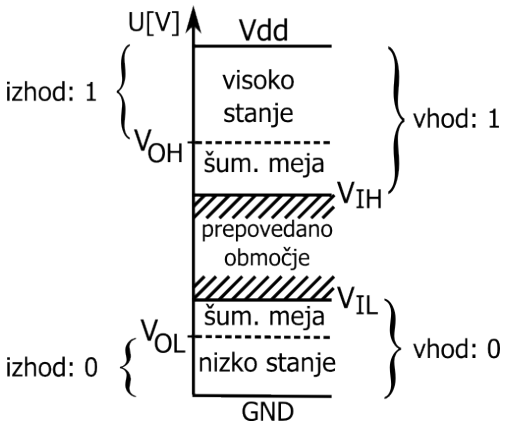
Če je na izhodu vezja v tehnologiji CMOS nizko stanje z napetostjo $0.9\,\mathrm{V}$, bo vsota napetosti $0.9\,\mathrm{V} + 0.2\,\mathrm{V} = 1.1\,\mathrm{V}$ že v prepovedanem območju.



Slika 2.6: Model povezave med dvema vezjema, ki upošteva inducirani šum.

Odpornost logičnih vezij na šum naredimo z ožjim območjem veljavnih potencialov na izhodni strani in širšim na vhodni strani vezja. Razlika v širini območja se imenuje šumna meja in zagotavlja določeno odpornost na induciran šum pri komunikaciji. Veljavno območje potencialov na oddajni in sprejemni strani je podano z enačbami:

	$oddajnik$	$sprejemnik$
logična 0:	$0\,\mathrm{V} \leq V_L \leq V_{OL}$	$0\,\mathrm{V} \leq V_L \leq V_{IL}$
logična 1:	$V_{OH} \leq V_H \leq V_{dd}$	$V_{IH} \leq V_H \leq V_{dd}$



Slika 2.7: Statični red z upoštevanjem šumne meje.

Vrednosti potencialov določajo standardi, npr. TTL, CMOS, LVCMOS. Izjava proizvajalca digitalnih integriranih vezij o skladnosti s standardom zagotavlja, da bomo brez težav povezali signale različnih vezij med seboj.

Inducirani šum lahko povzroči, da bo vrednost signala zunaj meja napajalnih napetosti: višja od V_{dd} ali nižja od GND (negativna napetost). Takšen signal se na sprejemniku sicer pravilno interpretira, lahko pa povzroči uničenje vezja, če preseže določene meje.

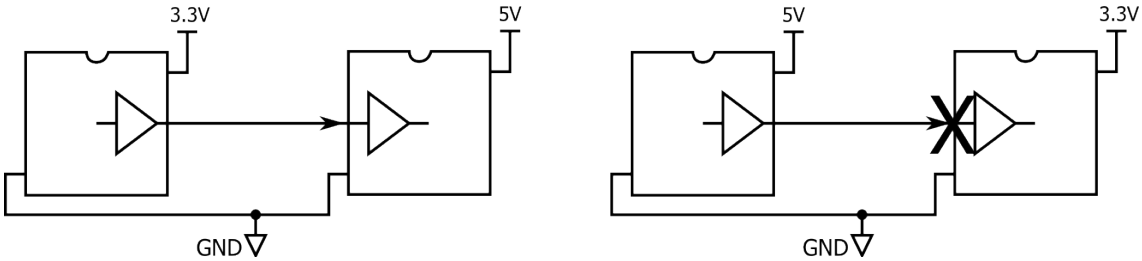
Tabela podaja primer vrednosti potencialov za vezja CMOS z napajalno napetostjo 5 V in za LVCMOS z napetostjo 3.3 V (standard JEDEC).

oznaka	pomen	CMOS [V]	LVCMOS [V]
V_{dd}	napajalna napetost	5.0	3.0–3.6
V_{IH}	vhodni visok nivo	3.0	2.0
V_{IL}	vhodni nizek nivo	1.0	0.8
V_{OH}	izhodni visok nivo	3.1	Vdd - 0.2
V_{OL}	izhodni nizek nivo	0.2	0.2

Tabela 2.3: Statični parametri 5-voltnih CMOS in 3.3-voltnih LVCMOS vezij.

2.3 Povezovanje različnih standardov

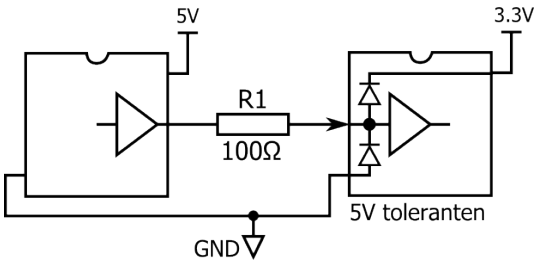
Včasih potrebujemo povezavo med integriranimi vezji, ki uporabljajo različne standarde. Poglejmo si primer povezave 5-voltnega in 3.3-voltnega vezja. Izhod 3.3-voltnega vezja brez težav krmili vhod 5-voltnega vezja. Visoko stanje vezja tehnologije LVCMOS je vsaj 3.1 V, kar je dovolj velik potencial, da se interpretira kot logična 1 v drugem vezju. Nizko stanje na izhodu je največ 0.2 V, kar ponovno zadošča za 5-voltne vhode tehnologije CMOS. Zavedati se moramo le, da smo s takšno povezavo znižali šumno mejo.



Slika 2.8: Povezovanje 5-voltnih na 3.3-voltna vezja.

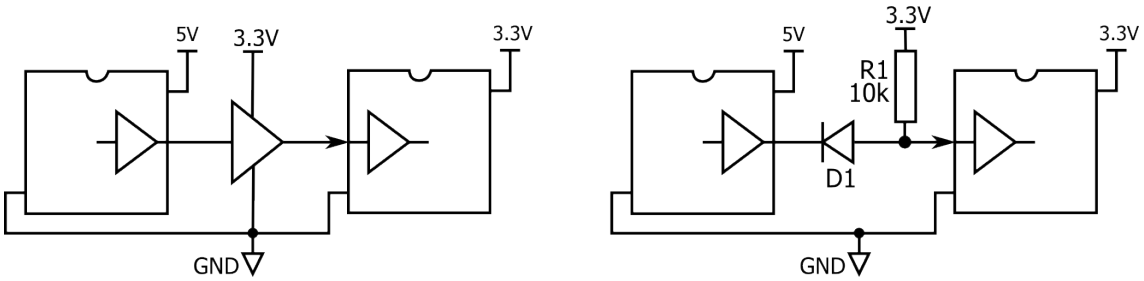
Pri obratni povezavi s 5-voltnega izhoda na vhod 3.3-voltne logike moramo biti precej previdnejši in jo brez pregleda specifikacij elementov ne smemo narediti. Visoko stanje 5-voltne logike je namreč precej nad napajalnim nivojem LVCMOS, zato bo po signalni liniji stekel velik enosmerni tok čez vhod vezja proti 3.3-voltni napajalni liniji. To je posledica delovanja zaščitnih diod na vhodih vezja, ki pri previsoki napetosti kratko sklenejo vhod proti napajalni povezavi. Tok skozi vhod omejimo z zaporedno vezanim zaščitnim upornikom.

Tipična upornost zaščitnega upornika je nekaj 100 Ω in je kompromis med visoko vrednostjo, ki povzroči počasnejši prenos signala, in nizko, pri kateri teče višji tok. Poleg omejitve toka je treba upoštevati tudi omejitve napetosti, saj sodobna vezja uniči že statična napetost na vhodu. Trend v tehnologiji integriranih vezij gre namreč proti zmanjševanju dimenzij in strukture v vezju so tako majhne, da hitro pride do preboja, ki jih trajno poškoduje. Sodobna integrirana vezja



Slika 2.9: Povezava 5-voltnega vezja na 3.3-voltno preko zaščitnega upora.

lahko poškoduje že napetost, višja od $4\,\mathrm{V}$. Če želimo takšno vezje priklopiti na izhod CMOS, ne zadošča le zaščitni upornik. Najboljša rešitev je z uporabo namenskega ojačevalnika za pretvorbo potencialnih nivojev, za manj zahtevne primere pa naredimo vezje za omejitev napetosti iz pasivnih elementov.



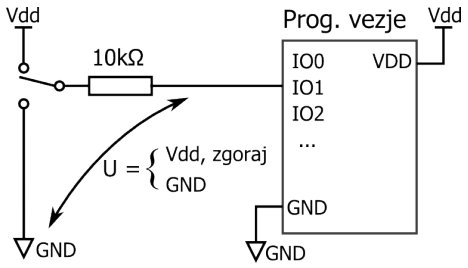
Slika 2.10: Pretvorba logičnih nivojev za občutljiva vezja LVCMOS.

Dioda D1 prevaja, ko je na izhodu oddajnega vezja nizko stanje. V tem primeru je na diodi napetost okoli $0.7\,\mathrm{V}$, ki jo zazna vezje kot logično 0 na vhodu. Kadar je na izhodu napetost $5\,\mathrm{V}$, je dioda zaprta in na vhod drugega vezja pride prek upora R1 napetost $3.3\,\mathrm{V}$. Vrednost upora R1 naj bo nekaj $k\Omega$.

2.4 Pozitivna in negativna logika

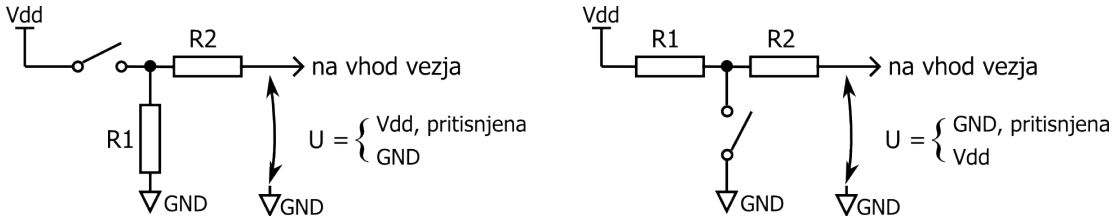
Vhodni signali v digitalna vezja prihajajo od drugih vezij ali neposredno od uporabnika. Uporabniške vhodne naprave so stikala, posamezne tipke ali tipkovnica, rotacijski kodirniki ipd. Na razvojnih sistemih bomo največkrat zasledili posamezne tipke ali preklopna stikala, ki dajo na vhod vezja napetost Vdd ali GND glede na maso.

Za vklop napajanja električnega vezja običajno uporabljamo stikalo, v digitalnih vezjih pa tudi za nastavljanje signalnih vhodov. Slika 2.11 prikazuje vezavo preklopnega dvopolnega stikala na vhod vezja. Preklopni priključek stikala je vezan na vhod vezja prek upora, ki služi za zaščito programirljivega vezja. Programirljivim vezjem namreč šele v postopku programiranja določimo, ali bo posamezni signal vhod ali izhod. Če bi pomotoma povezali izhodni signal prek stikala neposredno na napajalno napetost ali maso, bi naredili kratek stik in poškodovali vezje. Upor z dovolj veliko upornostjo v tem primeru omeji tok in do poškodb ne pride.



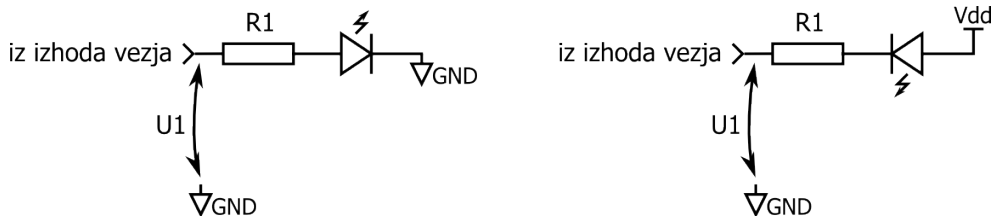
Slika 2.11: Vezava preklopnega stikala na programirljivo vezje.

Enopolna stikala ali tipke vežemo, kot prikazuje slika 2.12. Tipka je, zaporedno z uporom R1 vezana na napajalno napetost in maso, prek zaščitnega upora R2 pa je priključena na vhod vezja. Tipične vrednosti obeh uporov so nekaj $k\Omega$. Razlika med obema vezavama je, da je enkrat pri pritisnjeni tipki na vhodu vezja V_{dd}, pri spuščeni pa GND; pri drugi vezavi je ravno obratno.



Slika 2.12: Vezava tipke s pozitivno in negativno logiko.

Delovanje digitalnega vezja opazujemo na izhodih prek izhodnih naprav. Primeri izhodnih naprav so monitorji, LCD-prikazovalniki, prikazovalniki iz svetlečih diod (LED) in preprosti indikatorji z žarnico ali svetlečo diodo. Slika 2.13 prikazuje dve možni vezavi indikatorja s svetlečo diodo, ki se uporabljata na razvojnih sistemih. Svetleča dioda je zaporedno z uporom priključena med izhodom vezja in eno od napajalnih sponk (V_{dd} ali GND).

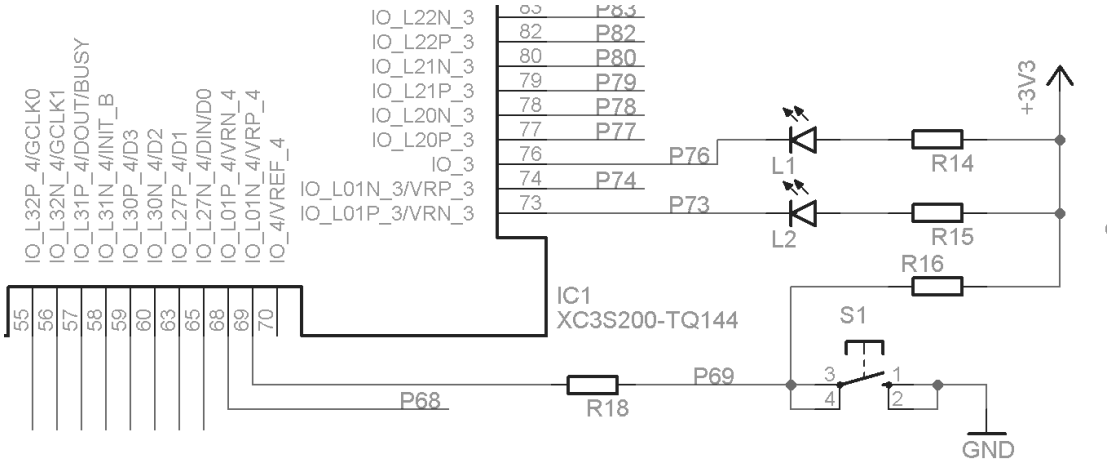


Slika 2.13: Vezava svetleče diode (LED) s pozitivno in negativno logiko.

Svetleča dioda je polprevodniški element, ki zasveti, ko je med njenima priključkoma napetost okoli 2 V (natančna vrednost je odvisna od vrste in barve) in teče tok v smeri trikotnika. Za majhne indikatorske LED zadošča tok nekaj mA. Iz teh podatkov in napajalne napetosti lahko izračunamo vrednost zaporednega upora, npr. $(3.3\,\mathrm{V} - 2\,\mathrm{V})/3.3\,\mathrm{mA} = 390\,\Omega$.

Naloge

1. Preglej izsek sheme digitalnega razvojnega sistema in ugotovi, kako so priključene vhodne tipke in izhodne LED. Kakšna je napetost na vhodu vezja IC1 ob pritisnjeni tipki? Kakšna napetost mora biti na izhodu IC1, da bo LED svetila?



2. Poišči na spletu statične parametre logičnih gradnikov v tehnologiji TLL.

$V_{IH} =$ _____

$V_{IL} =$ _____

$V_{OH} =$ _____

$V_{OL} =$ _____

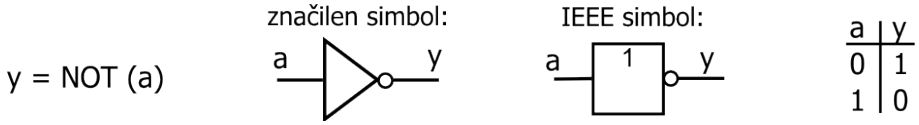
3. Ugotovi, kako bi sestavil vezavo tipke in svetleče diode, da bi ob pritisku na tipko LED ugasnila, hkrati pa bi na vhodu digitalnega vezja bila napetost GND.

3
Logična vrata

Logična vrata so osnovni gradniki digitalnih vezij. Uporabljajo se za izvedbo logičnih funkcij nad binarnimi vrednostmi, ki jih prenašajo digitalni signali.

3.1 Osnovna logična vrata

Logična vrata izhajajo iz matematične predstavitve osnovnih operacij nad binarnimi števili, ki jih obravnava Boolova algebra. Najpreprostejša operacija je negacija binarne vrednosti: Boolov izraz NOT(a) ima vrednost 1, kadar je $a = 0$, pri $a = 1$ pa ima vrednost 0. Načrtovanje kombinacijskega digitalnega vezja začnemo z opisom problema, ki ga prevedemo v Boolove izraze. Nato narišemo shemo vezja z uporabo grafičnih simbolov logičnih funkcij.

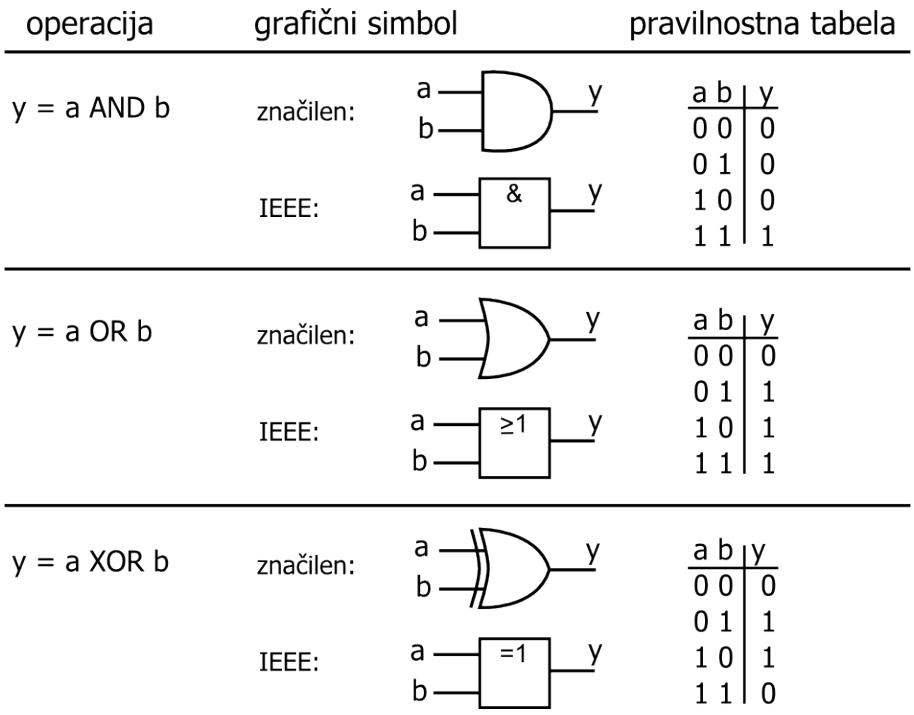


Slika 3.1: Boolova negacija, simbol negatorja in pravilnostna tabela.

V električnih shemah zasledimo dve obliki grafičnega simbola negatorja: značilen trikotni simbol in standardni simbol v obliki pravokotnika (standard IEEE std. 91–1984). Oba simbola imata majhen krožec na izhodnem priključku, ki označuje negiranje oz. inverzijo logične vrednosti. Pri risanju logičnih shem z računalniškimi orodji bomo večkrat naleteli na značilno obliko simbola, zato bomo to obliko uporabljali tudi v našem gradivu. Delovanje negatorja predstavimo s *pravilnostno tabelo*, v kateri so določene vrednosti izhodov pri vseh možnih stanjih vhoda.

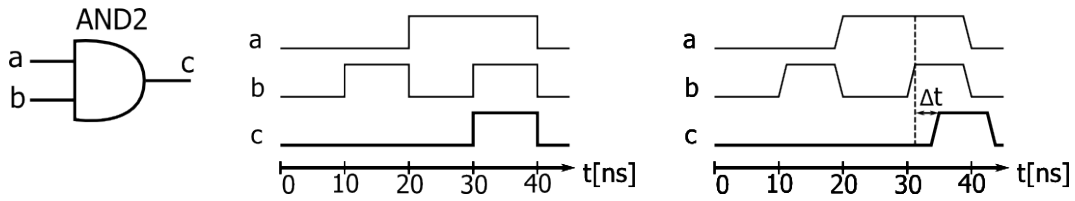
Poglejmo še nekaj Boolovih operacij nad dvema binarnima signaloma: *logična in* (a **AND** b) je 1, kadar sta prvi in drugi operand 1, sicer ima vrednost 0. Operacija *logični ali* (a **OR** b) je 1, kadar je vsaj en operand enak 1. Vrednost 0 ima le, ko sta oba operanda 0. Operacija *ekskluzivni ali* (a **XOR** b) je 1, kadar sta operanda različna, sicer pa ima vrednost 0.

Za vsako operacijo obstaja grafični simbol v značilni in standardni obliki, ki predstavlja ustrezna logična vrata. Delovanje teh operacij opisuje pravilnostna tabela s štirimi vrsticami, ker imamo štiri možne kombinacije vhodnih signalov: 0 0, 0 1, 1 0 in 1 1.

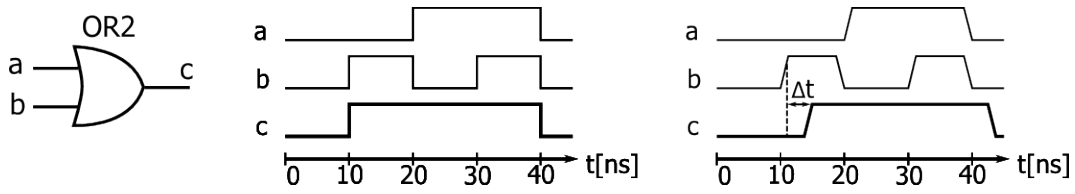


Slika 3.2: Boolove operacije, simboli logičnih vrat in pravilnostne tabele.

Delovanje logičnih vrat lahko opazujemo tudi na časovnem diagramu, kjer s časom spreminjamo stanje vhodov in opazujemo izhod. Logični simulator običajno ne upošteva zakasnitev vrat, zato se izhod spremeni takoj ob spremembi vhodov. V realnih logičnih vratih se izhod ne spremeni takoj ampak z določeno zakasnitvijo, ki je posledica preklopnih časov elektronskih elementov, iz katerih so logična vrata narejena.



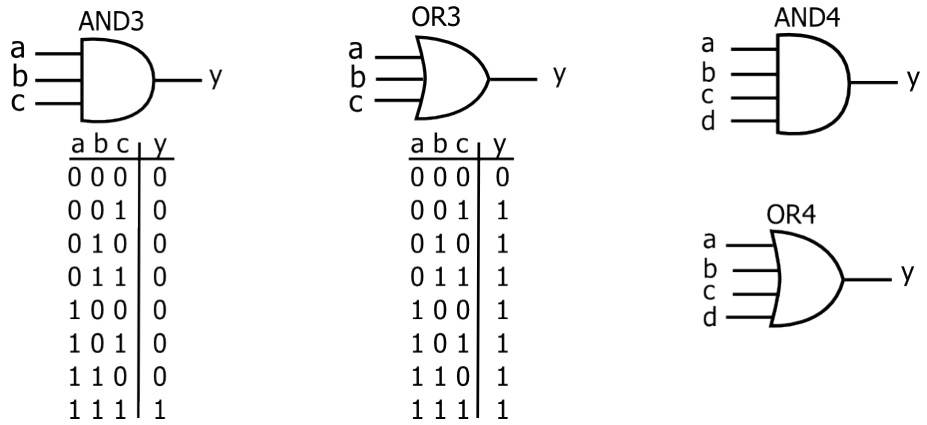
Slika 3.3: Idealni in realni časovni diagram logičnih vrat AND2.



Slika 3.4: Idealni in realni časovni diagram logičnih vrat OR2.

Na realnih časovnih diagramih smo označili zakasnitev z Δt. Zakasnitve posledično omejujejo hitrost spreminjanja signalov na vhodih in s tem hitrost delovanja vezja. Kadar so majhne v primerjavi s pričakovanimi časovnimi intervali spreminjanja signalov, jih lahko ignoriramo. Za analizo vezja pogosto zadostuje idealni časovni diagram, v katerem zakasnitve niso prikazane.

Logična vrata imajo lahko tudi več kot dva vhodna signala. Izhod večvhodnih vrat AND je 1 v primeru, ko so vsi vhodi enaki 1, sicer je izhod enak 0. Izhod vrat OR pa je 1, kadar je vsaj eden izmed vhodov enak 1, 0 pa je le primeru, ko so vsi vhodi enaki 0.



Slika 3.5: Logična vrata AND in OR z več vhodi.

S predstavljenimi operacijami lahko zapišemo enačbe v Boolovi aritmetiki za reševanje poljubne logične naloge. Izkaže se, da lahko vse logične funkcije izrazimo s kombinacijo osnovnih operacij: AND, OR in NOT. To lastnost izkoriščajo nekatera programirljiva integrirana vezja, ki jim v postopku programiranja določimo povezave med osnovnimi operacijami in tako omogočimo, da izvajajo poljubno digitalno funkcijo. Operacija XOR pa je osnovni element aritmetičnih gradnikov za seštevanje, odštevanje in primerjavo večbitnih logičnih vrednosti.

3.2 Načrtovanje logičnih vezij

Postopek načrtovanja preprostih digitalnih vezij, ki izvajajo logične operacije za rešitev podane naloge, bomo predstavili na nekaj primerih. Vsak primer se začne z opisom naloge, ki ga pretvorimo v Boolove izraze in nato v shemo logičnega vezja.

Avtomobilski alarm

Avtomobilski alarm naj se sproži, kadar je nastavljen in kadar je izpolnjen vsaj eden izmed pogojev: ali je aktiviran senzor gibanja ali pa so odprta vrata. Signal $alarm$ naj predstavlja stanje alarma, signal $vklop$ določa, ali je alarm nastavljen, signal $vrata$ naj predstavlja stanje vrat (stanje 1 pomeni odprta) in signal gib stanje senzorja. Delovanje alarma opišemo z logično enačbo:

$$alarm = vklop \text{ AND } (vrata \text{ OR } gib)$$

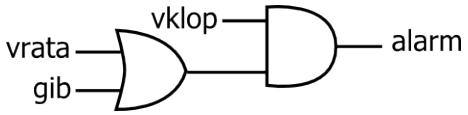
Sedaj lahko računamo vrednosti signala $alarm$ pri različnih vhodnih vrednostih:

- pri $vklop = 1, vrata = 0, gib = 0$ dobimo $alarm = 1 \text{ AND } (0 \text{ OR } 0) = 1 \text{ AND } 0 = 0$
- pri $vklop = 1, vrata = 1, gib = 0$ dobimo $alarm = 1 \text{ AND } (1 \text{ OR } 0) = 1 \text{ AND } 1 = 1$
- pri $vklop = 0, vrata = 1, gib = 0$ dobimo $alarm = 0 \text{ AND } (1 \text{ OR } 0) = 0 \text{ AND } 1 = 0$

Na podlagi logične enačbe lahko dokažemo nekatere trditve. Npr. trditev, da se alarm ne bo sprožil, če je vklop na 0. Logični izraz **AND** bo imel avtomatsko vrednost 0, če je na eni strani vrednost 0.

$$alarm = 0 \text{ AND } (vrata \text{ OR } gib) = 0$$

Logično enačbo pretvorimo v shemo vezja v nekaj korakih. Najprej narišemo signal $alarm$, ki je izhod z logičnih vrat **AND**. Vhod teh vrat je enkrat signal $vklop$, drugič pa izhod vrat **OR**, ki imajo na vhodu signala $vrata$ in gib.



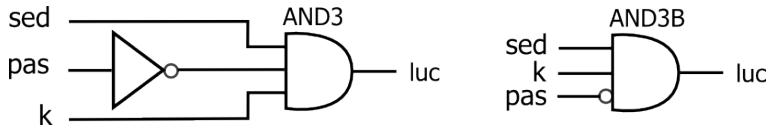
Slika 3.6: Shema vezja za avtomobilski alarm.

Indikator varnostnega pasu

Naredimo vezje, ki prižge opozorilni indikator, če nismo pripeti z varnostnim pasom. Signal s senzorja varnostnega pasu (pas) je v stanju 1, ko je pas pripet, in v stanju 0, kadar je odpet. Senzor v sedežu postavi signal sed v stanje 1, kadar nekdo sedi v avtu. Poleg tega imamo informacijo, ali je ključ (k) v položaju, ko je avto v pogonu (stanje 1). Indikatorska luč se prižge, če voznik sedi v avtu *in* je pas odpet *in* avto v pogonu, kar zapišemo z enačbo:

$$luc = sed \text{ AND NOT}(pas) \text{ AND } k$$

Logično vezje naredimo s trivhodnimi logičnimi vrati AND in negatorjem ali pa s posebnim simbolom vrat AND, kjer negiran vhod označimo s krožcem.

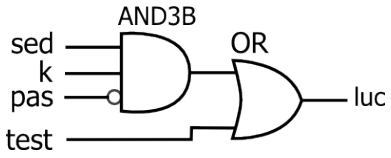


Slika 3.7: Dve obliki sheme vezja za opozorilno luč varnostnega pasu.

Opozorilni avtomobilski indikatorji se prižgejo za kratek čas tudi, kadar obrnemo ključ. S tem preverimo, ali opozorilne lučke res delujejo. Vzemimo, da imamo na voljo testni signal $test$, ki se, ko obrnemo ključ za nekaj sekund postavi na 1, potem pa gre nazaj na 0. Indikatorska luč se prižge, kadar je testni signal na 1 *ali* v primeru, če voznik ni pripet.

$$luc = test \text{ OR } sed \text{ AND NOT}(pas) \text{ AND } k$$

Shemo digitalnega vezja indikatorja s testnim signalom prikazuje slika 3.8.



Slika 3.8: Končna sheme vezja za opozorilno luč varnostnega pasu.

Ugibaj kombinacijo

Naredimo enostavno igro, pri kateri en igralec nastavi kombinacijo dveh signalov a in b, drugi pa skuša nastavljeno kombinacijo uganiti s postavljanjem svojih dveh signalov c in d. Ko sta nastavljeni kombinaciji enaki, se vklopi indikatorska dioda.

Naloge se lotimo tako, da najprej rešimo enostavnejši problem primerjave dveh signalov. Uporabili bomo lastnost ekskluzivne ali operacije, ki postavi izhod na 1, kadar sta vhodna signala različna. Ker želimo postaviti izhod na 1, kadar sta vhodna signala enaka, moramo le še negirati izhod. Zapišimo Boolove izraze za primerjavo signalov a in c ter b in d:

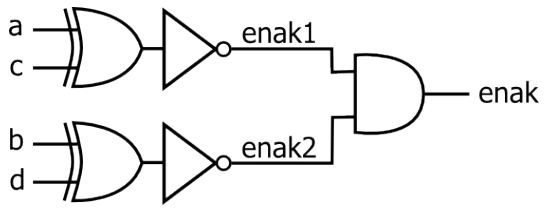
$$enak1 = \text{NOT}(a \text{ XOR } c)$$

$$enak2 = \text{NOT}(b \text{ XOR } d)$$

Sedaj ni težko rešiti prvotne naloge: kombinaciji signalov sta enaki, kadar je postavljen na 1 rezultat prve primerjave *in* druge primerjave:

$$enak = enak1 \text{ AND } enak2$$

Slika 3.9 prikazuje ustrezno logično vezje, ki se v terminologiji digitalnih vezij imenuje primerjalnik. Na podoben način bi naredili primerjavo enakosti večbitnih vrednosti.



Slika 3.9: Vezje za primerjavo dveh 2-bitnih vhodnih vrednosti.

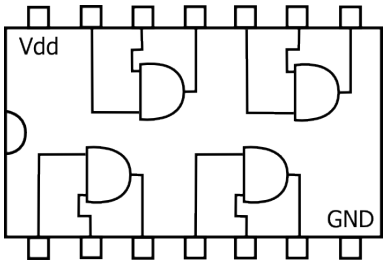
3.3 Izdelava vezij z logičnimi vrati

Logična vrata so narejena z elektronskimi stikali. Prva digitalna vezja so bila narejena iz elektromehanskih stikal – relejev, ki so jih pozneje zamenjale elektronske cevi (elektronke). To so bili veliki in okorni gradniki, ki so bili pogosto v okvari. Leta 1945 so med iskanjem napake v digitalnem računalniku Mark II našli hrošča med kontakti releja. Od takrat računalniške napake imenujejo hrošč (angl. bug) in postopek odkrivanja ter odpravljanja napak *razhroščevanje* (debugging). Releje in elektronke so nadomestili precej manjši in zanesljivejši tranzistorji. Največji skok v razvoju digitalnih sistemov pa so omogočila *integrirana vezja*, v katerih je na majhni rezini silicija narejeno celotno vezje.

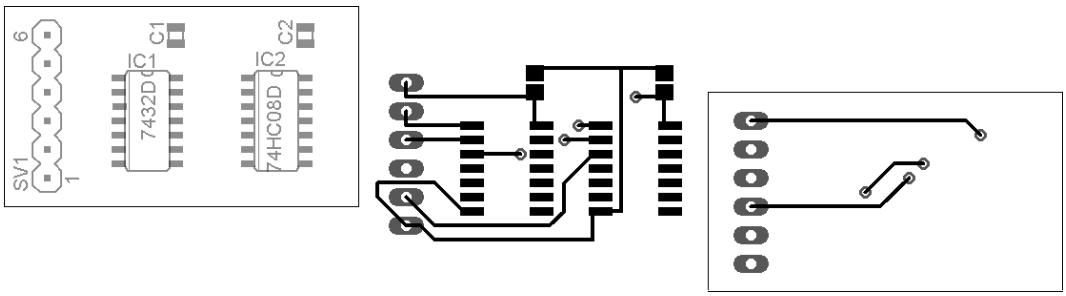
Integrirana vezja

Proizvajalci integriranih vezij so pripravili celo vrsto digitalnih integriranih vezij za potrebe razvijalcev digitalnih sistemov. Med njimi najdemo osnovna logična vrata, pomnilnike in sestavljene gradnike za najpogosteje uporabljene naloge. Digitalna integrirana vezja, ki so narejena v določenem proizvodnem procesu, z med seboj združljivimi statičnimi parametri, imenujemo *družina* logičnih vezij. Najbolj znana je družina logičnih vezij 7400, ki vsebuje predstavnike vseh pomembnejših logičnih gradnikov.

Slika 3.10 prikazuje integrirano vezje z oznako 7408, v katerem so štiri dvovhodna logična vrata AND, na sliki 3.11 pa je načrt tiskanega vezja za avtomobilski alarm, ki vsebuje dve integrirani vezji z logičnimi vrati.



Slika 3.10: Integrirano vezje 7408 s štirimi logičnimi vrati AND.



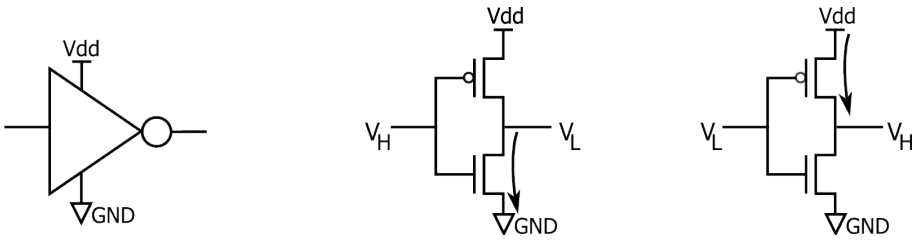
Slika 3.11: Načrt tiskanega vezja za avtomobilski alarm (sitotisk, zgornja in spodnja plast).

Elektronska stikala

Digitalna integrirana vezja vsebujejo elektronska stikala, ki so narejena s polprevodniškimi *tranzistorji*. Tranzistor deluje kot stikalo, pri katerem električno prevodnost med dvema kontaktoma spreminjamo z električnim potencialom na kontrolnem vhodu.

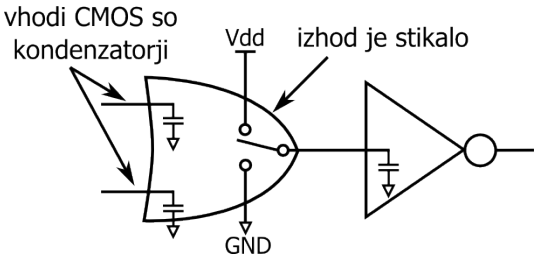
Digitalna vezja z oznako CMOS so zgrajena iz komplementarnih tranzistorjev vrste nMOS in pMOS. Slika 3.12 prikazuje zgradbo logičnega negatorja v izvedbi CMOS. Kadar je na vhodu negatorja visok potencial, bo prevajal spodnji tranzistor (nMOS) in povezal izhod na maso oz. potencial V_L. Če je na vhodu nizek potencial pa prevaja zgornji (pMOS) in takrat je na izhodu napajalna napetost oz. potencial V_H.

Zgornji in spodnji tranzistor sta odprta izmenično, zato pri konstantnem vhodu tudi skozi napajalni sponki gradnika CMOS praktično tok ne teče. Digitalni gradniki v izvedbi CMOS imajo izredno majhno porabo in so idealni za izdelavo kompleksnih vezij. Tehnologija integriranih vezij omogoča izdelavo vedno manjših tranzistorjev oz. vedno večjega števila le-teh na enaki površini. Digitalna integrirana vezja vsebujejo danes na milijone polprevodniških tranzistorjev, ki delujejo kot stikala.



Slika 3.12: Logični negator v CMOS izvedbi.

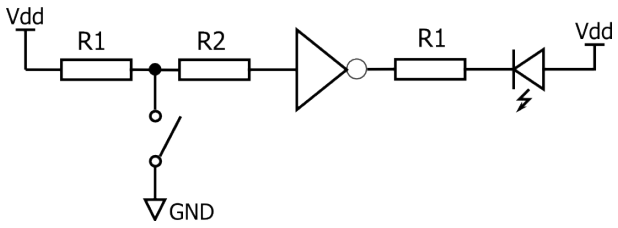
Pri obravnavi digitalnih vezij bomo uporabljali poenostavljene modele. Kontrolni vhodi tranzistorjev CMOS se obnašajo kot kondenzatorji, skozi katere enosmerni tok ne teče. Za enosmerne napetosti in tokove predstavlja takšen vhod odprte sponke, tako da enosmerni tok prek vhodnega signala ne teče. Izhod predstavimo zelo poenostavljeno s stikalom, ki preklaplja izhodni signal med napajalnima nivojema V_{dd} in GND. V resnici izhod ni idealno stikalo, ampak ima neko upornost, saj je tok skozi izhodni preklopni element omejen (običajno $10-20mA$). Slika 3.13 predstavlja električni model digitalnih vhodnih in izhodnih sponk vezja v tehnologiji CMOS.



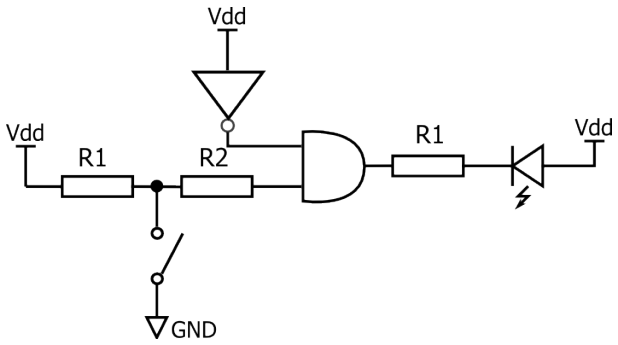
Slika 3.13: Poenostavljen model vhoda in izhoda v tehnologiji CMOS

Naloge

1. Preglej obe logični shemi in ugotovi, v katerem položaju stikala bo LED prižgana.



2. Poglej vezavo na shemi in ugotovi, kdaj bo LED prižgana.



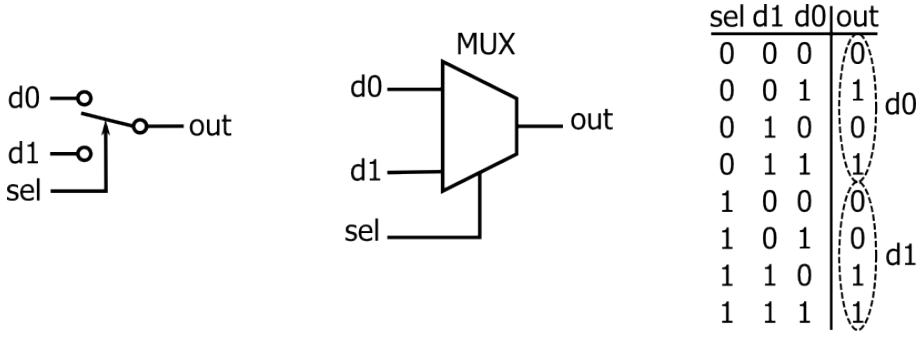
3. Utemelji, zakaj ne smemo vezati skupaj izhodov dveh logičnih vrat. Pomagaj si s poenostavljenim modelom logičnega gradnika v tehnologiji CMOS.

4
Kombinacijski gradniki

Digitalna vezja in sisteme sestavljamo s povezavo vnaprej pripravljenih gradnikov, ki izvajajo različne logične funkcije. Gradniki so logična vezja, delimo jih na kombinacijska in sekvenčna. Kombinacijska vezja imajo izhod odvisen le od trenutne kombinacije na vhodu, pri sekvenčnih vezjih pa je odvisen še od shranjenega stanja. Obravnavali bomo kombinacijske izbiralnike, dekodirnike in osnovne pomnilne elemente.

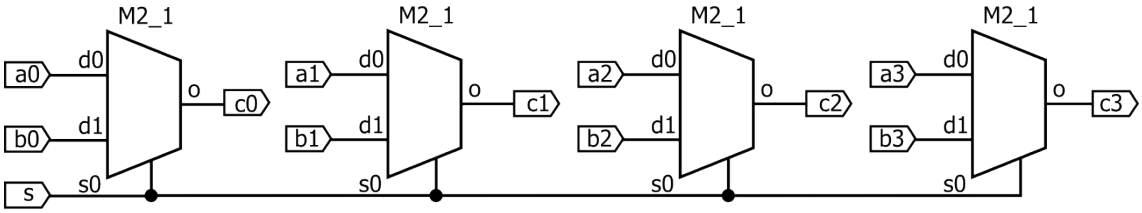
4.1 Kombinacijski izbiralnik

Kombinacijski izbiralnik ali multiplekser deluje kot preklopno stikalo, ki prenese na izhod vrednost enega izmed podatkovnih vhodov. Izbiralnik ima dva podatkovna vhoda, izbirni vhod in izhod. Ko je izbirni signal sel enak 0, dobimo na izhodu vrednosti z vhoda $d0$, sicer pa vrednosti z vhoda $d1$, kot prikazuje pravilnostna tabela.



Slika 4.1: Izbiralnik 2-1: stikalna shema, simbol in pravilnostna tabela.

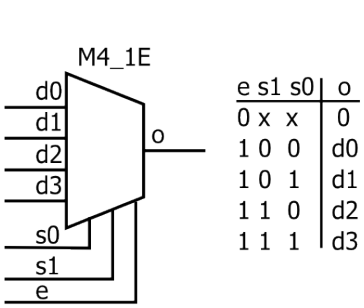
Podatkovni vhodi in izhod so lahko večbitne vrednosti. Izbiralnik z n-bitnimi podatkovnimi signali vsebuje n osnovnih izbiralnikov, pri katerih je izbirni vhod vezan skupaj. Oglejmo si vezje 4-bitnega izbiralnika 2-1, ki na izhod c, sestavljen iz signalov $c0-c3$ prenese enega izmed 4-bitnih vhodnih signalov a ali b.



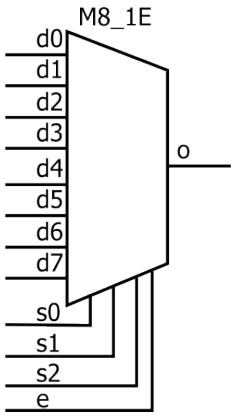
s	c3 c2 c1 c0
0	a3 a2 a1 a0
1	b3 b2 b1 b0

Slika 4.2: Vezava štirih izbralnikov v izbiralnik 4-bitnih vrednosti.

Naloga izbiralnika je prenašanje večjega števila vhodnih vrednosti prek enega izhodnega signala, kar izkoriščamo na primer v komunikacijskih vmesnikih. Nekateri izbiralniki imajo dodaten vhodni signal e (angl. enable) za omogočanje izhoda. Kadar je ta signal na logični 0, je izhod postavljen na 0, ne glede na stanje ostalih vhodov, kadar je signal e enak 1, pa deluje izbiralnik normalno.



e	s1	s0	o
0	x	x	0
1	0	0	d0
1	0	1	d1
1	1	0	d2
1	1	1	d3



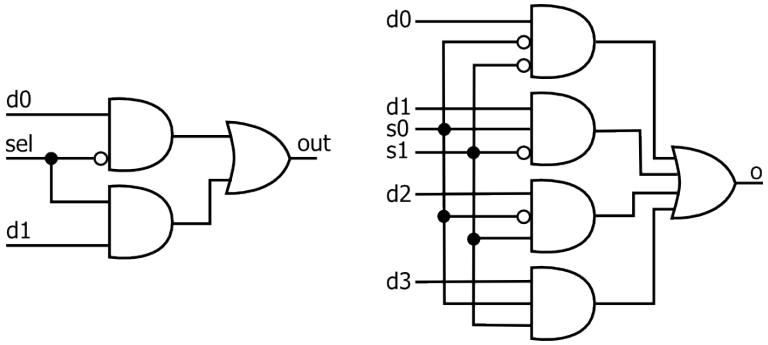
e	s2	s1	s0	o
0	x	x	x	0
1	0	0	0	d0
1	0	0	1	d1
1	0	1	0	d2
1	0	1	1	d3
1	1	0	0	d4
1	1	0	1	d5
1	1	1	0	d6
1	1	1	1	d7

Slika 4.3: Izbiralnika 4-1 in 8-1 z dodatnim vhodom za omogočanje e.

V splošnem imajo izbiralniki več podatkovnih vhodov in večbitni izbirni vhod. Izbiralnik z oznako n-1 ima n-bitni izbirni vhod in 2^n podatkovnih vhodov. Slika 4.3 prikazuje simbole izbiralnikov 4-1 in 8-1. Podrobnosti delovanja teh izbiralnikov razberemo iz pravilnostnih tabel, ki so zapisane v zgoščeni obliki. Tabele ne vsebujejo vseh kombinacij vhodnih signalov, ker bi že za manjši izbiralnik 4-1 potrebovali kar 128 vrstic. Ko je vhodni signal e postavljen na 0, je izhod definiran ne glede na stanje ostalih vhodov, zato so njihova stanja označena z x.

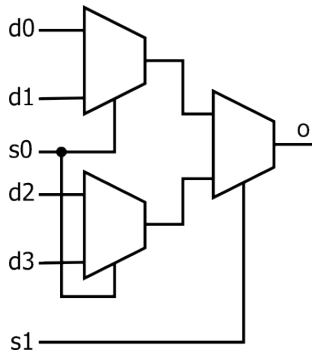
4.2 Izdelava izbiralnikov

Izbiralnik je kombinacijsko vezje, ki ga lahko naredimo iz osnovnih logičnih vrat. Vezje sestavimo iz vrat AND, ki prepuščajo izbrani podatkovni vhod, in vrat OR, s katerimi združimo vse izhode. Izbirni signali gredo neposredno ali pa preko negatorjev na vrata AND. Kadar je kombinacija teh signalov na logični 1, bo izhod vrat enak podatkovnemu vhodu, v vseh ostalih primerih pa bo izhod enak 0. S primerno vezavo izbirnega signala zagotovimo, da bodo le ena logična vrata prepuščala izbrani vhodni signal.



Slika 4.4: Izvedba izbiralnikov 2-1 in 4-1 z logičnimi vrati.

Število podatkovnih vhodov izbiralnika lahko enostavno podvojimo z zaporedno vezavo treh izbiralnikov. Na sliki 4.5 je vezje izbiralnika 4-1, ki je narejeno iz treh manjših izbiralnikov 2-1.



Slika 4.5: Izvedba izbiralnika 4-1 z uporabo izbiralnikov 2-1.

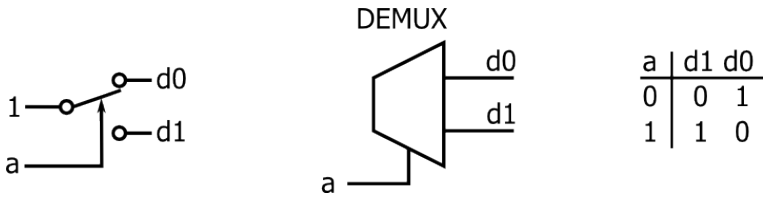
Razlika med obema prikazanima izvedbama izbiralnika 4-1 je v vrsti in številu logičnih vrat: prva zahteva 4 trivhodna AND in ena OR, druga pa 6 dvovhodnih AND in 3 vrata OR. Pomembno je tudi število nivojev logičnih vrat med vhodom in izhodom: pri prvi izvedbi sta dva nivoja, pri drugi pa so štirje. Zaradi zakasnitev na štirih zaporednih logičnih vratih je drugo vezje počasnejše v primerjavi s prvim.

4.3 Kombinacijski dekodirnik

Z izrazom dekodirnik označujemo digitalna vezja, ki pretvarjajo eno obliko binarne kode v drugo. Med kombinacijske dekodirnike spadajo razdeljevalnik, binarni dekodirnik in splošno dekodirno vezje.

Razdeljevalnik in binarni dekodirnik

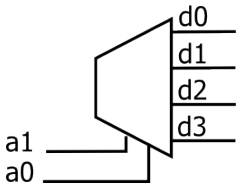
Razdeljevalnik ali demultiplekser ima nasprotno funkcijo od izbiralnika, kar nakazuje tudi zrcaljen simbol. Razdeljevalnik ima v splošnem n vhodnih signalov in 2^n izhodnih signalov. Izhodnih signalov je toliko, kolikor je dvojiških kombinacij na vhodu: enovhodni razdeljevalnik ima 2 izhoda, 2-vhodni ima 4 izhode, 3-vhodni ima 8 izhodov, 4-vhodni pa 16 izhodov. Razdeljevalnik ima le na enem izbranem izhodu logično 1, in sicer na tistem, ki ustreza trenutni vhodni kombinaciji.



a	d1	d0
0	0	1
1	1	0

Slika 4.6: Razdeljevalnik 1-2: stikalna shema, simbol in pravilnostna tabela.

Slika 4.13 prikazuje grafični simbol dvobitnega razdeljevalnika in njegovo pravilnostno tabelo. Pri vhodni kombinaciji 0 0 je logična 1 na izhodu $d0$, pri kombinaciji 0 1 je logična 1 na izhodu $d1$ itn.

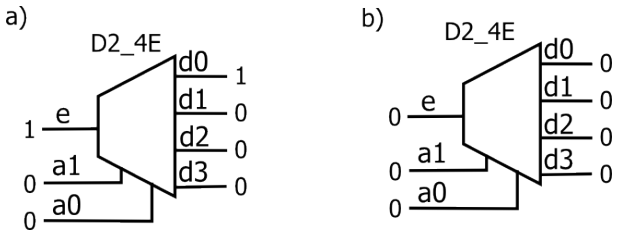


a1	a0	d3	d2	d1	d0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Slika 4.7: 2-bitni razdeljevalnik.

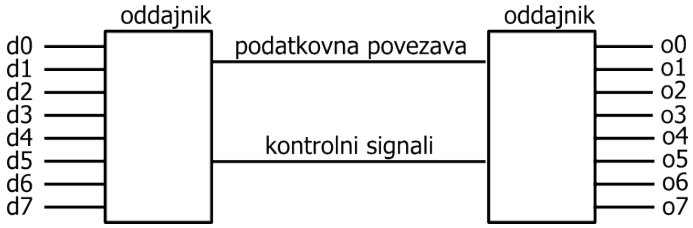
V digitalnem vezju uporabljamo razdeljevalnik tam, kjer želimo s kombinacijo na vhodih aktivirati natančno enega izmed množice gradnikov. Kombinacijo stanj na vhodih obravnavamo kot naslov (a, angl. address), ki je dodeljen izbranemu gradniku vezja. Razdeljevalnik v tem primeru služi kot dekodirnik naslovov.

Razdeljevalnik z dodatnim vhodom za omogočanje imenujemo *Binarni dekodirnik*. Signal za omogočanje obravnavamo kot podatkovni vhod, katerega vrednost se prenese na izbrani izhod. Kadar je signal e enak 1, dobimo na izbranem izhodu logično 1, kadar je e enak 0, so pa vsi izhodi na logični 0.



Slika 4.8: Binarni dekodirnik a) pri $e = 1$ deluje normalno, b) pri $e = 0$ so vsi izhodi 0.

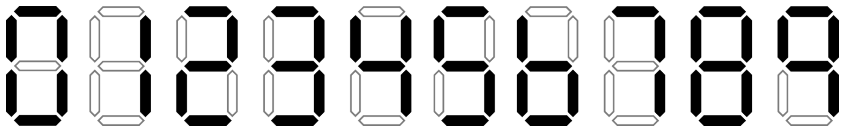
Binarni dekodirnik uporabljamo v komunikacijskih sistemih. Primer uporabe prikazuje slika 4.9, kjer z izbiralnikom združimo več vhodov in jih prenesemo prek ene podatkovne povezave, binarni dekodirnik pa opravi inverzno operacijo.



Slika 4.9: Vezje za kombiniranje (multipleksiranje) podatkovnih signalov.

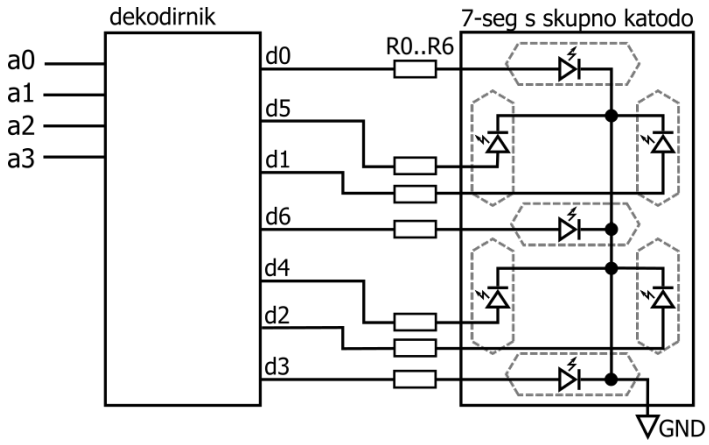
7-segmentni dekodirnik

Poleg binarnih dekodirnikov poznamo še vrsto drugih dekodirnih vezij, ki imajo vsi skupno lastnost, da kombinacijo stanj na vhodih pretvorijo v neko drugo kombinacijo na izhodih. Med bolj znanimi je dekodirnik za 7-segmentne prikazovalnike. Prikazovalnik je sestavljen iz sedmih svetlečih diod, ki so razporejene tako, da lahko prikažejo desetiške števke (slika 4.10).



Slika 4.10: Prikaz desetiških števk na 7-segmentnem prikazovalniku.

Desetiške števke predstavimo v dvojiški obliki s 4-bitno vrednostjo. Dekodirnik pretvarja 4-bitne vhode $a0$ do $a3$ v 7-bitne izhode $d0$ do $d6$, ki so vezani na posamezne segmente za prikaz



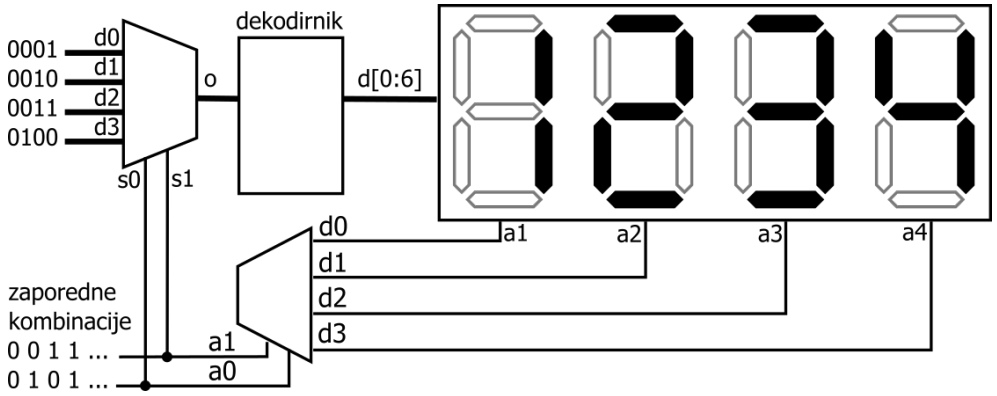
a3..a0	d6...d0
0000	0111111
0001	0000110
0010	1011011
0011	1001111
0100	1100110
0101	1101101
0110	1111101
0111	0000111
1000	1111111
1001	1101111

Slika 4.11: Vezava dekodirnika za 7-segmentni prikazovalnik in pravilnostna tabela.

desetiških števk. Slika 4.11 prikazuje priklop dekodirnika na prikazovalnik in pravilnostno tabelo za prikaz števk med 0 in 9.

Poznamo prikazovalnike LED s skupno katodo in prikazovalnike LED s skupno anodo. Nekateri prikazovalniki imajo še dodatno diodo za prikaz decimalne pike. Segmente prikazovalnikov s skupno katodo prižigamo z logično 1, katode vseh svetlečih diod pa morajo biti vezane na maso (slika 4.11).

Prikazovalniki s skupno anodo imajo skupni priključek vezan na V_{dd} in posamezne segmente prižigamo z logično 0. Ker delujejo LED z negativno logiko. je treba v pravilnostni tabeli dekodirnika negirati izhode. Takšen dekodirnik je narejen v integriranem vezju 74LS47 (shema na sliki 1.4).



Slika 4.12: Skupna vezava štirih LED prikazovalnikov s skupno anodo.

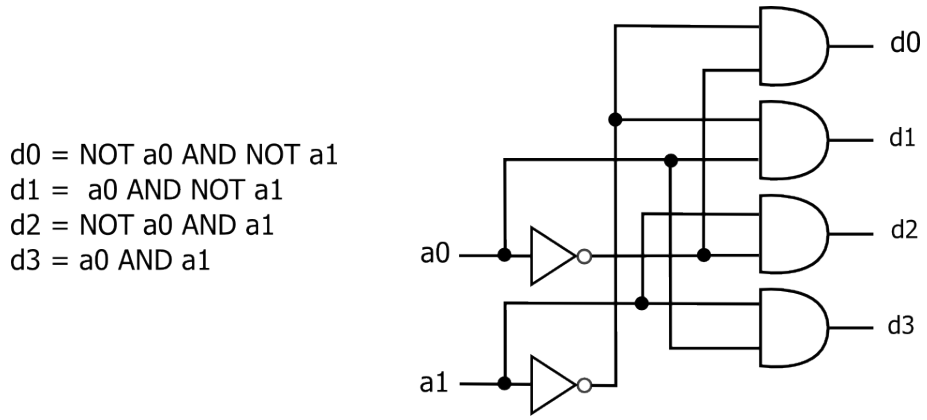
Kadar potrebujemo v vezju več prikazovalnikov, povežemo njihove podatkovne signale ($d0$ do $d6$) skupaj, s signali na skupnih anodah ali katodah pa izmenično prižigamo posamezne prikazovalnike. Če imamo štiri prikazovalnike, tako namesto $4 \times 7 = 28$ signalov potrebujemo le 7 podatkovnih in 4 izbirne signale. Z razdeljevalnikom poskrbimo, da je v vsakem trenutku prižgan le en prikazovalnik. Če izbrani prikazovalnik zamenjamo vsaj 20-krat v sekundi, naše

oko ne bo opazilo menjav in bomo videli prižgane vse prikazovalnike hkrati. Na dekodirnik moramo pošiljati zaporedne binarne kombinacije, v vezju pa je še izbiralnik 4-1, ki poskrbi, da se v ritmu menjave izbranih prikazovalnikov spreminjajo tudi dekodirani podatkovni signali.

4.4 Izdelava razdeljevalnikov

Do izvedbe razdeljevalnika z osnovnimi logičnimi vrati pridemo tako, da zapišemo logične enačbe za posamezne signale. V dvobitnem razdeljevalniku je izhod $d0$ enak 1, kadar sta oba vhoda $a0$ in $a1$ enaka 0: vhoda torej negiramo in uporabimo operacijo AND. Podobno razmišljamo za ostale izhode in zapišemo enačbe:

$$
\begin{aligned}
d0 &= \text{NOT}(a0)\ \text{AND}\ \text{NOT}(a1)\\
d1 &= a0\ \text{AND}\ \text{NOT}(a1)\\
d2 &= \text{NOT}(a0)\ \text{AND}\ a1\\
d3 &= a0\ \text{AND}\ a1
\end{aligned}
$$



Slika 4.13: Izvedba 2-bitnega razdeljevalnika z logičnimi vrati.

Razdeljevalnik je narejen iz štirih logičnih vrat AND in negatorjev. V logičnih izrazih opazimo, da se negirana signala $a0$ in $a1$ pojavljata večkrat, kar izkoristimo za optimizacijo vezja. Namesto da bi uporabili štiri negatorje, uporabimo le dva in povežemo negiran signal na več vrat AND, kot prikazuje slika 4.13.

Binarni dekodirnik ima enako strukturo kot izbiralnik. Dodaten vhodni signal e pripeljemo na vsa logična vrata AND, saj predstavlja dodaten pogoj, da je na izbranem izhodu logična 1. V shemi dvobitnega binarnega dekodirnika bi tako imeli 3-vhodna logična vrata AND.

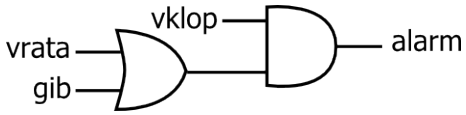
4.5 Načrtovanje vezij z izbiralniki

Kombinacijska vezja naredimo na podlagi pravilnostne tabele z osnovnimi logičnimi vrati ali pa z izbiralniki. V pravilnostni tabeli izbiralnika so na izhodu vsi podatkovni vhodi. Če povežemo na podatkovne vhode konstantne vrednosti 0 ali 1, lahko naredimo poljubno pravilnostno tabelo.

Za izvedbo kombinacijskega vezja z več izhodnimi signali uporabimo več izbiralnikov. V splošnem kombinacijskem vezju, brez optimizacije, potrebujemo toliko izbiralnikov, kot je izhodnih signalov. Velikost izbiralnikov je odvisna od števila vhodov kombinacijskega vezja.

Avtomobilski alarm z izbiralnikom

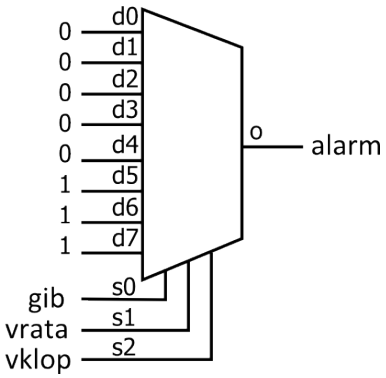
Z izbiralnikom naredimo vezje avtomobilskega alarma, ki se sproži, kadar je vklopljen in je aktiviran senzor gibanja ali pa so odprta vrata. Izvedbo alarma z logičnimi vrati smo si že ogledali:



Slika 4.14: Logično vezje za avtomobilski alarm.

Če želimo to vezje narediti z izbiralnikom, moramo najprej napisati pravilnostno tabelo za izhodni signal. Vezje ima 3 vhodne signale in v pravilnostni tabeli je $2^3 = 8$ kombinacij. Kombinacijsko vezje avtomobilskega alarma naredimo z enim izbiralnikom 8-1, ki ima na podatkovnih vhodih konstante logične vrednosti, kot so določene v tabeli.

vklop	vrata	gib	alarm
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

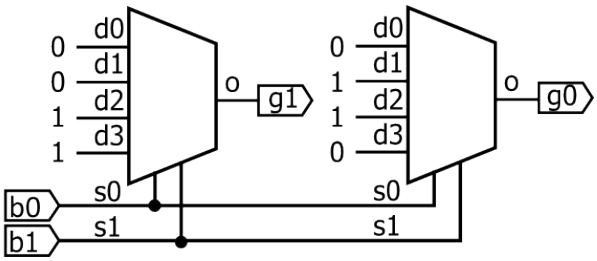


Slika 4.15: Izvedba avtomobilskega alarma z izbiralnikom.

Grayev dekodirnik

Naredimo dvobitni dekodirnik, ki pretvarja vhodno dvojiško kodo v Grayevo kodo. Značilnost Grayeve kode je, da se pri zaporednih vrednostih vedno spremeni le en bit. Slika 4.16 prikazuje logično tabelo 2-bitne pretvorbe in izvedbo vezja z izbiralniki.

b1	b0	g1	g0
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0



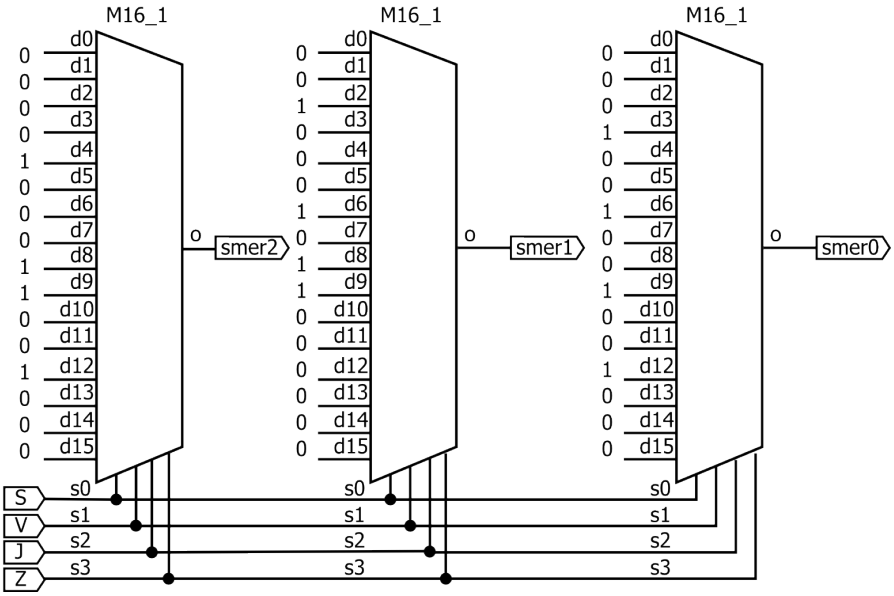
Slika 4.16: Grayev dekodirnik: pravilnostna tabela in izvedba z izbiralniki.

Dvobitni dekodirnik ima dva vhodna signala in dva izhodna signala. Za vsak izhod potrebujemo svoj izbiralnik 4-1, ki ima dva izbirna vhoda. Vezje lahko poenostavimo, če upoštevamo, da so vrednosti signala $g1$ enake signalu $b1$, zato v praksi potrebujemo le en izbiralnik za določitev signala $b0$.

Dekodirnik za smer vetra

Naredimo vezje za dekodiranje smeri vetra. Iz vetrnice dobimo 4 signale: S, J, V in Z, med katerimi sta eden ali dva postavljena na 1, ostali pa so 0. Naloga dekodirnika je pretvoriti 4-bitni vhod v 3-bitno kombinacijo, ki jo bomo določili s pravilnostno tabelo. Tabela naj definira vse kombinacije, tudi tiste, ki jih ne pričakujemo (označene z x):

veter	Z	J	V	S	smer2:0
x	0	0	0	0	0 0 0
S	0	0	0	1	0 0 0
V	0	0	1	0	0 1 0
SV	0	0	1	1	0 0 1
J	0	1	0	0	1 0 0
x	0	1	0	1	0 0 0
JV	0	1	1	0	0 1 1
x	0	1	1	1	0 0 0
Z	1	0	0	0	1 1 0
SZ	1	0	0	1	1 1 1
x	1	0	1	0	0 0 0
x	1	0	1	1	0 0 0
JZ	1	1	0	0	1 0 1
x	1	1	0	1	0 0 0
x	1	1	1	0	0 0 0
x	1	1	1	1	0 0 0



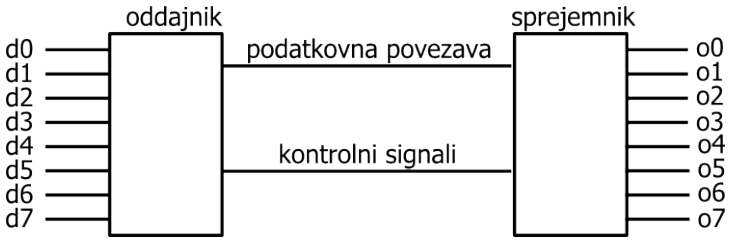
Slika 4.17: Dekodirnik smeri vetra z izbiralniki.

V vezju uporabimo tri izbiralnike 16-1, ki imajo izbirne vhode $s0$ do $s3$ vezane skupaj in priključene na vhodne signale. Na podatkovne vhode povežemo konstante iz pravilnostne tabele. Pri tem moramo paziti na vrstni red: izbirni signal $s3$ je vezan na signal na levi strani pravilnostne tabele, $s2$ na naslednji signal itn.

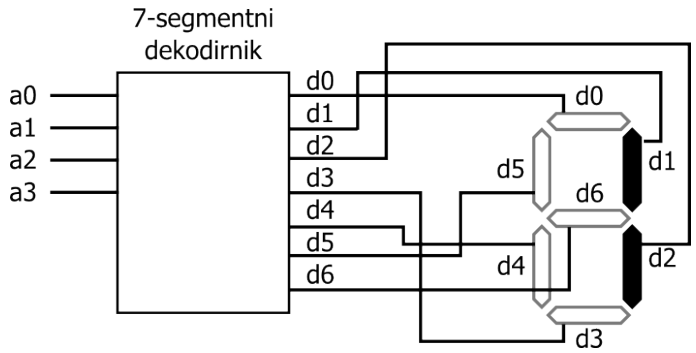
Naloge

1. Razmisli, kako bi sestavil izbiralnik 8-1, če imaš na voljo le izbiralnike 4-1 in 2-1. Nariši shemo vezave!

2. Opiši zgradbo in delovanje vezja za prenos osmih signalov preko ene podatkovne povezave. Katere kombinacijske gradnike potrebuješ na oddajni in sprejemni strani?



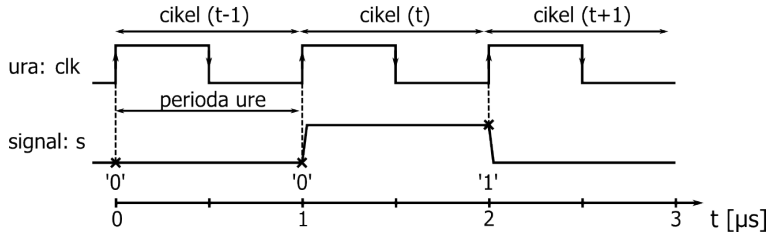
3. Ugotovi, koliko izbiralnikov in kakšne vrste izbiralnike potrebujemo za izdelavo 7-segmentnega dekodirnika.



5
Sekvenčna vezja

Sekvenčna vezja so logična vezja, pri katerih je izhod odvisen od vhodov in shranjenega stanja. Osnovni sekvenčni gradniki so pomnilni elementi, v katerih se shranjujejo logične vrednosti. Najpreprostejši gradniki shranijo le en bit (logično 0 ali 1), sestavljeni gradniki pa shranijo večbitno vrednost. *Stanje vezja* predstavljajo vse logične vrednosti, ki so v nekem trenutku shranjene v vezju.

V sekvenčnih vezjih imamo običajno poseben signal, ki določa, kdaj naj se izhodi spremenijo. Takšen signal je v obliki periodičnih impulzov in ga označujemo z imenom ura (angl. clock, clk). Uro dobimo iz posebnega vezja, ki se imenuje oscilator. V digitalnih sistemih bomo najpogosteje naleteli na kvarčne oscilatorje, s katerimi dobimo stabilno uro natančno določene frekvence.



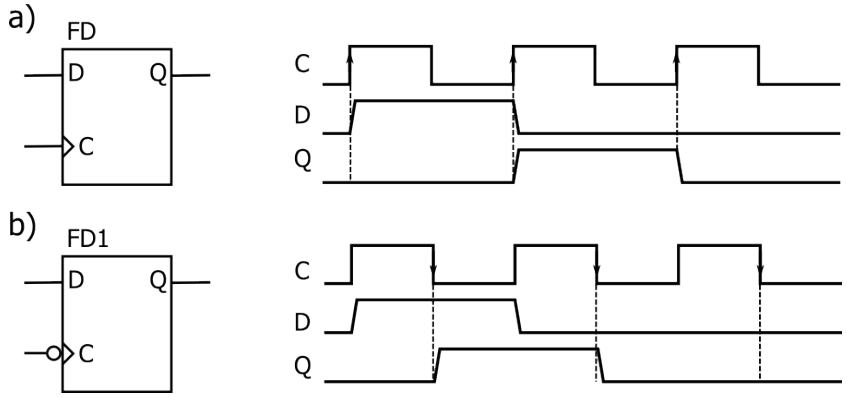
Slika 5.1: Proženje na nivo ali na fronto ure.

Slika 5.1 prikazuje primer časovnega diagrama urnega signala. Glavni parameter ure je perioda, ki predstavlja časovni interval med sosednjimi impulzi. Časovni interval ene periode, v katerem se stanje spremeni iz nizkega v visoko (ali obratno), imenujemo *urni cikel*. Včasih je pomembno tudi razmerje med trajanjem visokega in nizkega cikla, ki ga podajamo v odstotkih. Obratna vrednost periode je frekvenca, v primeru s slike velja: $f = 1/1\,\mu\mathrm{s} = 1\,\mathrm{MHz}$. Trenutek

prehoda iz nizkega v visoko stanje imenujemo prednja ali naraščajoča fronta ure. Prehod iz visokega v nizko stanje pa imenujemo zadnja ali padajoča fronta. Fronte ure lahko na časovnem diagramu posebej označimo s puščicami.

5.1 Flip-flop

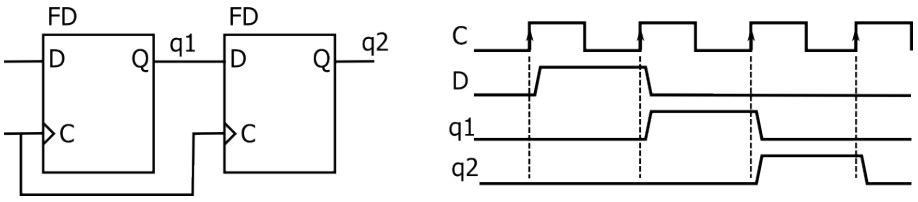
Pomnilni elementi, ki spreminjajo stanje ob taktu ure, se imenujejo *flip-flopi*. Podatkovni flip-flop (angl. Data Flip-Flop, DFF) prenaša logično stanje z vhoda D na izhod Q ob fronti ure. Vhodni podatkovni signal določa vrednost, ki bo na izhodu vezja v naslednjem urnem ciklu: $Q(t) = D(t-1)$, $Q(t+1) = D(t)$ itn.



Slika 5.2: Simbola in časovna diagrama podatkovnih flip-flopov.

Skupna lastnost flip-flopov je, da se izhodni signal spreminja le ob prednji ali zadnji fronti ure. Podatkovni flip-flop, ki je prožen na zadnjo fronto ure, je označen z negacijo (krožcem) pred vhodom za uro.

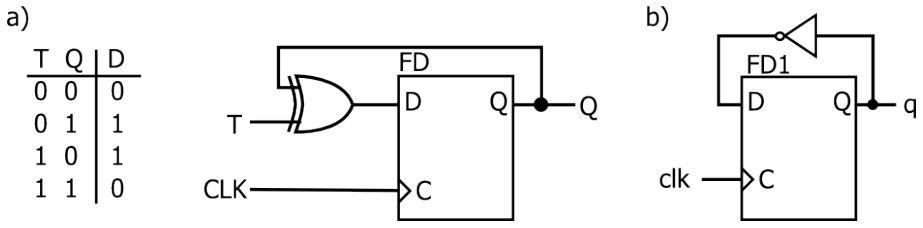
Če vežemo drug za drugim dva podatkovna flip-flopa, bomo zakasnili signal za dva cikla, kot prikazuje slika 5.3. Takšna vezava je osnova za izvedbo elementov, ki jih imenujemo pomikalni registri.



Slika 5.3: Zakasnitev signalov pri prehodu čez zaporedne flip-flope D.

Preklopni flip-flop ali flip-flop T (angl. Toggle) ima krmilni signal T, ki povzroči, da izhod ob naraščajoči fronti ure zamenja vrednost: $Q(t+1) = \text{NOT } Q(t)$. Kadar je signal T na 0, izhod ohranja vrednost. Preklopni flip-flop naredimo iz podatkovnega flip-flopa in logike na

vhodu. Podatkovni flip-flop izvaja funkcijo: $Q(t+1) = D(t)$, logika na vhodu pa določa $D(t)$ v odvisnosti od vhoda T in stanja $Q(t)$, kar opišemo s pravilnostno tabelo.

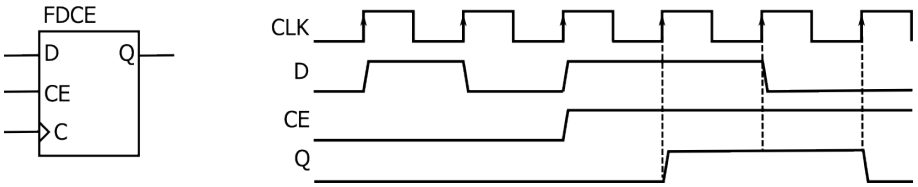


Slika 5.4: a) tabela in zgradba flip-flopa T, b) flip-flop s stalnim preklapljanjem izhoda.

Pravilnostna tabela določa logično funkcijo na vhodu flip-flopa, ki je v našem primeru XOR. Če potrebujemo flip-flop, ki stalno preklaplja izhod, uporabimo inverter med izhodom in vhodom flip-flopa (slika 5.4b).

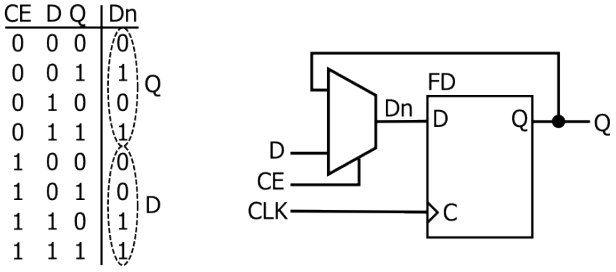
5.2 Register

Register je podatkovni pomnilni gradnik s signalom za omogočanje CE. Kadar je CE na 0, bo izhod registra ohranjal vrednost.



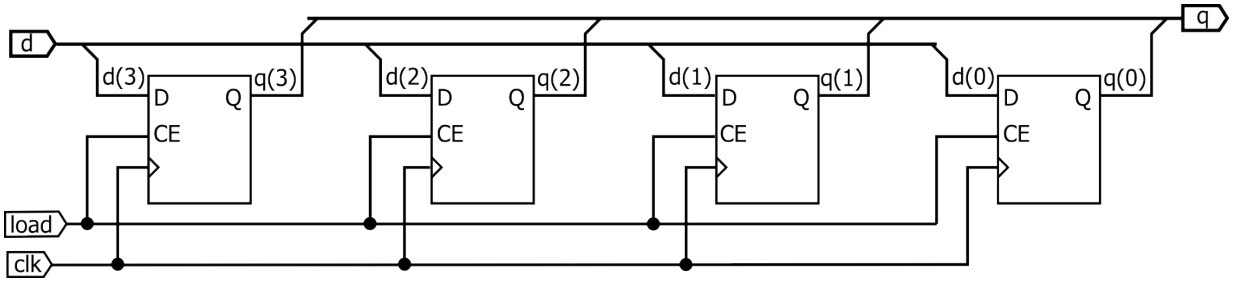
Slika 5.5: Simbol in časovni diagram flip-flopa, ki ohranja stanje izhoda.

Ohranjanje vrednosti izhoda opisuje enačba: $Q(t+1) = Q(t)$. Delovanje registra opišemo s tabelo, v kateri so na levi strani vse kombinacije vhodov CE, D in stanja Q, na desni strani pa je signal Dn, ki določa naslednje stanje $Q(t+1)$. Tabela je identična pravilnostni tabeli izbiralnika 2-1, ki predstavlja logiko na vhodu flip-flopa.



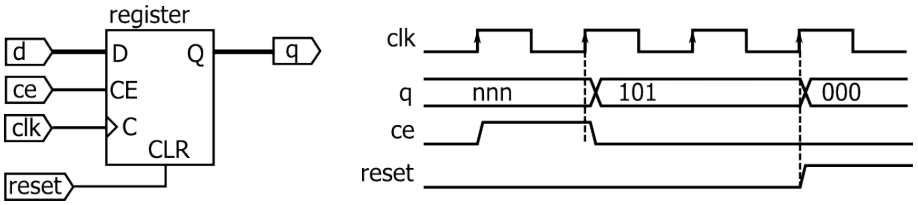
Slika 5.6: Tabela in zgradba enobitnega registra.

Zadrževanje vrednosti bi dosegli tudi z elektronskim stikalom, ki bi odklopilo uro. Takšna rešitev za sinhrona vezja ni najboljša, zato se na vhodu registra pogosteje uporablja izbiralnik. Register, ki shranjuje večbitne besede, naredimo z vzporedno vezavo flip-flopov FDCE. Slika 5.7 prikazuje logično shemo 4-bitnega registra z uro in signalom $Load$ za nalaganje nove vrednosti.



Slika 5.7: Shema 4-bitnega registra.

Flip-flopi v registru imajo pogosto tudi kontrolni signal CLR, ki deluje kot reset; ko ga postavimo na 1, gre izhod na 0, ne glede na fronto ure. Pravimo, da ima flip-flop asinhroni reset, ker se izhod spremeni asinhrono, ne glede na uro. Izhod ostane na 0, dokler je ta signal aktiven, ko gre nazaj na 0, pa začne delovati kot običajno in izhod se lahko spremeni na 1 šele ob naslednji prednji fronti ure.



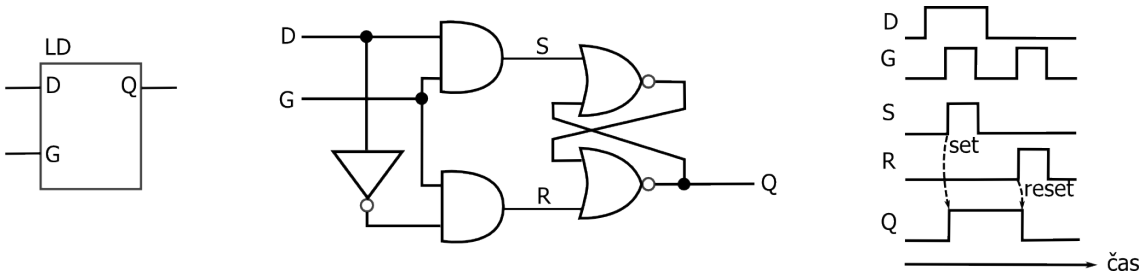
Slika 5.8: Simbol in časovni diagram registra s signalom reset.

5.3 Izvedba flip-flopov

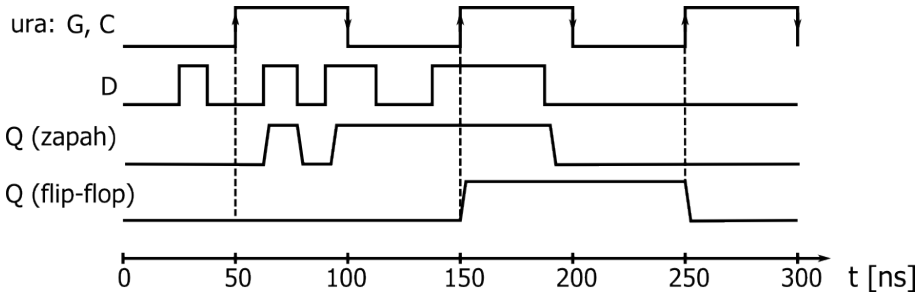
Za izvedbo so najenostavnejši asinhroni pomnilni gradniki oz. zapahi. Zapah z oznako SR ima dva kontrolna vhoda: S (angl. Set) in R (angl. Reset). Ob aktivnem signalu S se izhod Q postavi na 1, ob aktivnem R pa se postavi na 0. Kadar sta oba kontrolna signala na 0, izhod ohranja zadnjo vrednost. Primer: začetno stanje izhoda Q naj bo 0;

- S=1 in R=0 povzroči, da gre izhod na 1;
- S=0 in R=0 ohranja izhod na 1;
- S=0 in R=1 povzroči, da gre izhod na 0;
- S=0 in R=0 ohranja izhod na 0.

Kadar sta oba vhoda zapaha na 1, izhod ni stabilen in preklaplja med logično 0 in 1. Osnovni stabilen gradnik je zapah D, ki ima podatkovni vhod D, kontrolni vhod (ali vrata) za omogočanje G (angl. Gate) in izhod Q. Kadar je $G = 1$, se stanje s podatkovnega vhoda prenese na izhod. Pravimo, da je zapah transparenten in vse spremembe vhoda z majhno zakasnitvijo prenaša na izhod. Ko gre signal G na 0, se stanje izhoda zapahne in ohranja zadnjo vrednost.

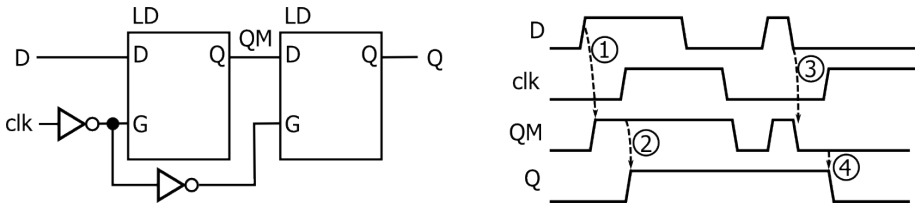


Slika 5.9: Zapah D: simbol, logična shema in časovni diagram.



Slika 5.10: Primerjava časovnega poteka izhoda pri zapahu D in flip-flopu.

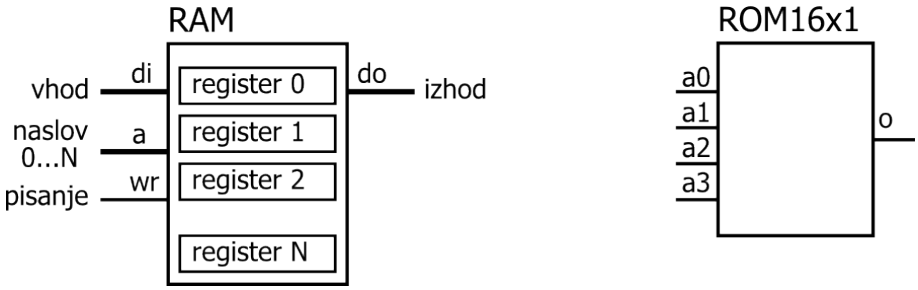
Podatkovni flip-flop lahko naredimo s povezavo dveh zapahov. Prvi zapah je transparenten, ko je signal ure (clk) na 0, ob prehodu ure na 1 pa zapahne zadnjo vrednost (1). Obenem postane transparenten drugi zapah, ki prenese vrednost na izhod (2). Če pride do spremembe na podatkovnem vhodu, ko je ura na 0, se sprememba sicer prenese na izhod prvega zapaha (3), ne pa tudi na izhod vezja. Do spremembe na izhodu vezja pride šele ob naslednji naraščajoči fronti ure (4), tako da dobi izhod vrednost s podatkovnega vhoda tik pred fronto ure.



Slika 5.11: Izvedba flip-flopa D iz dveh zapahov.

5.4 Pomnilnik

Pomnilniki so elementi za shranjevanje podatkov in ukazov, ki jih npr. obdelujejo računalniški digitalni sistemi. Pomnilnik shrani več podatkovnih besed, vendar v posameznem ciklu zapisujemo ali beremo le eno besedo naenkrat. Elektronske polprevodniške pomnilnike označujemo s kratico RAM (angl. Random Access Memory), kar pomeni da imajo enak čas za dostop do naključno izbrane pomnilniške besede. Oznaka izhaja iz primerjave s pomnilniki, pri katerih so besede zapisane na magnetnem traku in je čas dostopa do besede odvisen od trenutnega položaja elektromagnetne glave na traku.



Slika 5.12: Blokovna shema pomnilnika RAM in ROM.

Signale na priključkih pomnilnika RAM razdelimo na:

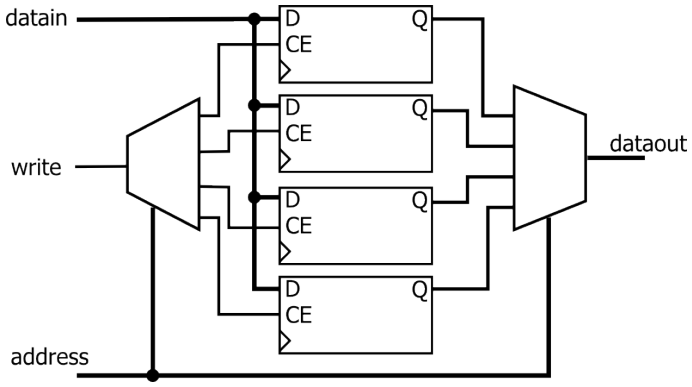
- podatkovno vodilo (ang. *data*), ki je lahko ločeno na podatkovni vhod (datain) in podatkovni izhod (dataout);
- naslovno vodilo (ang. *address*), s katerim izberemo posamezno besedo;
- krmilne signale, ki določajo operacije (branje, pisanje, mirovanje).

Pomnilne celice si lahko predstavljamo kot registre. Vsak izmed registrov ima svoj naslov v obliki binarnega števila med 0 in N. Na naslovno vodilo postavimo vrednost, s katero izberemo enega izmed registrov, s krmilnimi signali pa določimo, ali bomo v register naložili novo besedo ali pa bomo brali iz registra.

Pomnilniki z oznako ROM (angl. Read Only Memory) imajo vnaprej zapisano vsebino, ki jo preberemo tako, da nastavimo naslovne signale. Slika 5.12 prikazuje simbol pomnilnika ROM s štirimi naslovnimi signali in enim izhodom. Takšen pomnilnik, v katerega lahko shranimo 16 bitov, najdemo v celicah programirljivega vezja.

5.5 Izdelava pomnilnikov

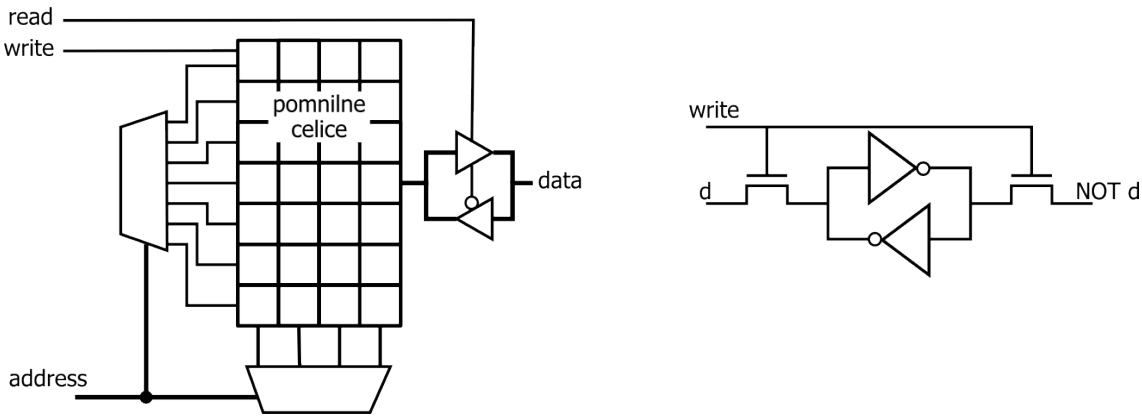
Pomnilnik je mogoče sestaviti iz elementov, ki smo jih do sedaj spoznali: registrov, binarnega dekodirnika in izbiralnika. Registri imajo podatkovne vhode vezane skupaj. Vsakemu registru dodelimo naslov *address* in prek binarnega dekodirnika omogočimo vpis besede v naslovljeni



Slika 5.13: Izvedba pomnilnika RAM iz registrov.

register, kadar je aktiven krmilni signal *write*. Branje poteka tako, da nastavimo naslov, in izbrana beseda se prek izbiralnika prenese na podatkovni izhod.

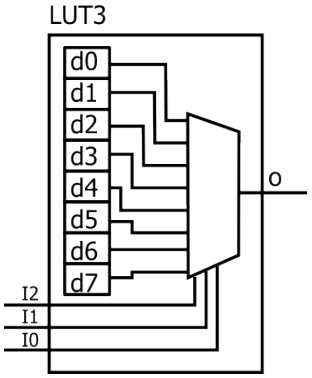
Pomnilnik iz registrov potrebuje za shranjevanje besed veliko elektronskih elementov, ker imajo registri iz flip-flopov D kompleksno zgradbo. V praksi zato uporabljamo učinkovitejše in cenejše pomnilnike, sestavljene iz matrike pomnilnih celic. Slika 5.14 prikazuje zgradbo statičnega pomnilnika RAM in posamezne pomnilne celice.



Slika 5.14: Izvedba statičnega pomnilnika in pomnilna celica.

Razdeljevalniki poskrbijo za izbiro pomnilnih celic, ki so razporejene v matriko. Vsaka celica je sestavljena iz dveh negatorjev za shranjevanje podatka in tranzistorjev za vpis nove vrednosti. Za vsak bit potrebujemo skupaj 6 tranzistorjev, kar je precej manj kot pri izvedbi s flip-flopom D. Še učinkovitejši so dinamični pomnilniki, ki hranijo zapis v enem samem tranzistorju, vendar imajo zahtevnejšo krmilno logiko.

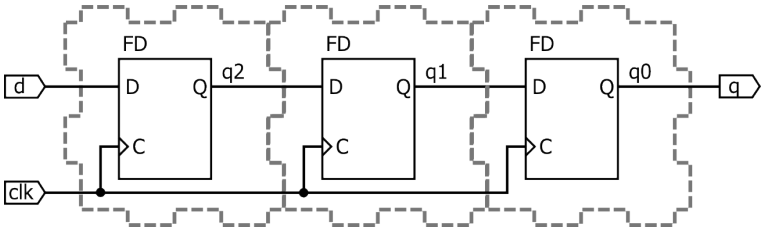
Pomnilnik ROM lahko uporabimo kot splošen gradnik za izdelavo kombinacijskih vezij, podobno kot izbiralnik. Vsebina pomnilnika mora biti enaka desni strani pravilnostne tabele kombinacijskega vezja, vhodi vezja pa so vezani na naslovne signale pomnilnika. Takšne bralne pomnilnike zasledimo v programirljivih vezjih. Imenujemo jih tudi vpogledna tabela (LUT, angl. Look-Up Table) in imajo 3 do 6 naslovnih vhodov in en izhod.



Slika 5.15: Vpogledna tabela LUT3 s tremi naslovnimi vhodi.

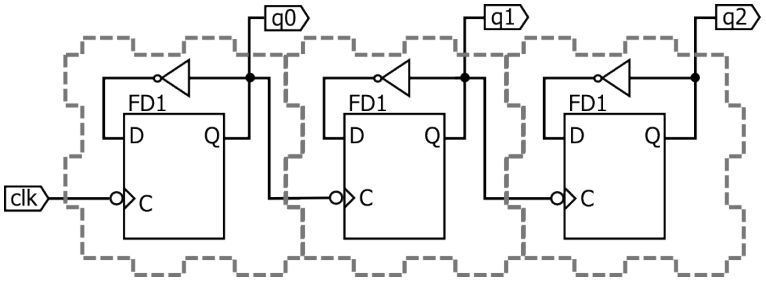
5.6 Sekvenčna vezja

Povezovanje digitalnih gradnikov lahko primerjamo z zlaganjem kock. Simboli digitalnih gradnikov imajo vhodne signale na levi in izhodne na desni strani. *Zaporedno vezavo* naredimo tako, da postavimo več gradnikov v vrsto in izhodne signale povežemo z vhodnimi.



Slika 5.16: Zaporedna vezava podatkovnih flip-flopov.

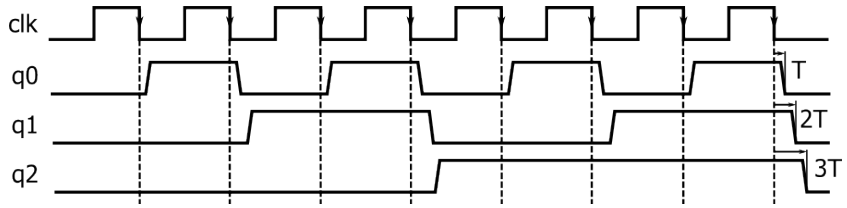
Značilnost zaporedne vezave je zakasnitev pri prenosu podatkov skozi vezje. Kontrolni signali so vezani na vse gradnike (npr. ura), podatki pa prehajajo zaporedno čez gradnike in so zakasnjeni. Pomikalni register s slike 5.16 ima zakasnitev med vhodom in izhodom 3 urne cikle.



Slika 5.17: Zgradba 3-bitnega serijskega števca.

Iz zaporedno vezanih preklopnih flip-flopov naredimo serijski števec, kot prikazuje slika 5.17. Gradniki so vezani tako, da izhod posameznega flip-flopa krmili uro naslednjega. Na

časovnem diagramu vidimo, da predstavlja zaporedje izhodnih signalov $q2$, $q1$ in $q0$ binarno zaporedje, ki se spreminja ob ciklih vhodne ure. V realnem vezju je izhod prvega flip-flopa $q0$ za čas T zakasnjen za fronto ure. Ta signal krmili naslednji flip-flop, katerega izhod ima že dvojno zakasnitev (2T) glede na vhodno uro, izhod tretjega pa trojno zakastnitev (3T).

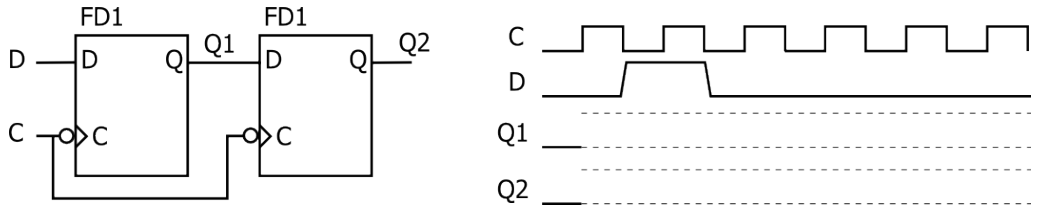


Slika 5.18: Časovni potek signalov 3-bitnega serijskega števca.

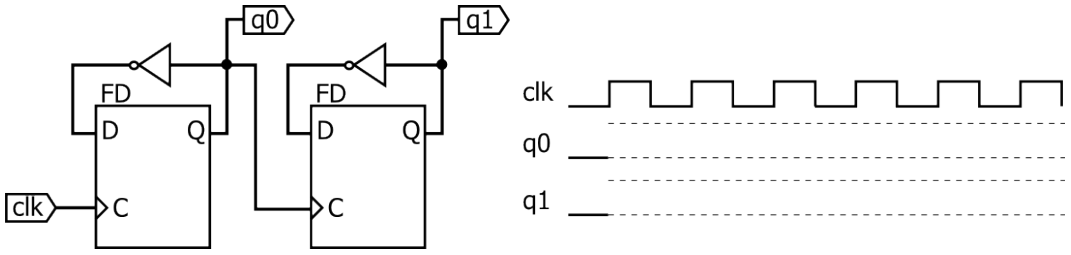
Števec pogosto uporabljamo kot generator zaporedja binarnih vrednosti. Če opazujemo vse izhode hkrati, nam različne zakasnitve preklopov povzročajo težave pri dekodiranju vrednosti in omejujejo navišjo frekvenco delovanja števca. Zaradi tega števce raje načrtujemo v obliki sinhronega vezja, kjer so vsi flip-flopi vezani na isto uro.

Naloge

1. Vriši v časovni diagram spreminjanje izhodnih vrednosti dveh zaporednih flip-flopov, ki sta prožena na zadnjo fronto ure.



2. Določi časovni potek izhodnih signalov 2-bitnega števca, ki je sestavljen iz flip-flopov na prednjo fronto ure in negatorjev.



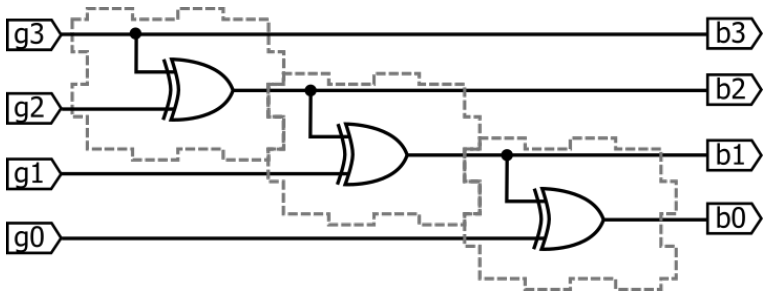
3. Ugotovi, koliko vpoglednih tabel in kakšne bi potrebovali za izdelavo 5-bitnega sinhronega števca.

6
Načrtovanje vezij

Načrtovanje vezja ali *sinteza* vezja je postopek, pri katerem pridemo iz opisa delovanja do zgradbe vezja. Digitalno vezje naredimo s povezavo osnovnih kombinacijskih in sekvenčnih gradnikov. Vezje, ki vsebuje vsaj en sekvenčni gradnik, imenujemo sekvenčno vezje. Spoznali bomo osnovne vezave logičnih gradnikov, izdelavo sinhronih števcev ter postopek načrtovanja sekvenčnih strojev z diagramom stanj.

6.1 Osnovne vezave gradnikov

Zaporedno vezavo logičnih gradnikov naredimo tako, da izhod enega gradnika vežemo na vhod drugega. Primer takšne vezave je dekodirnik 4-bitne Grayeve kode, v katerem so zaporedno vezana tri logična vrata XOR.



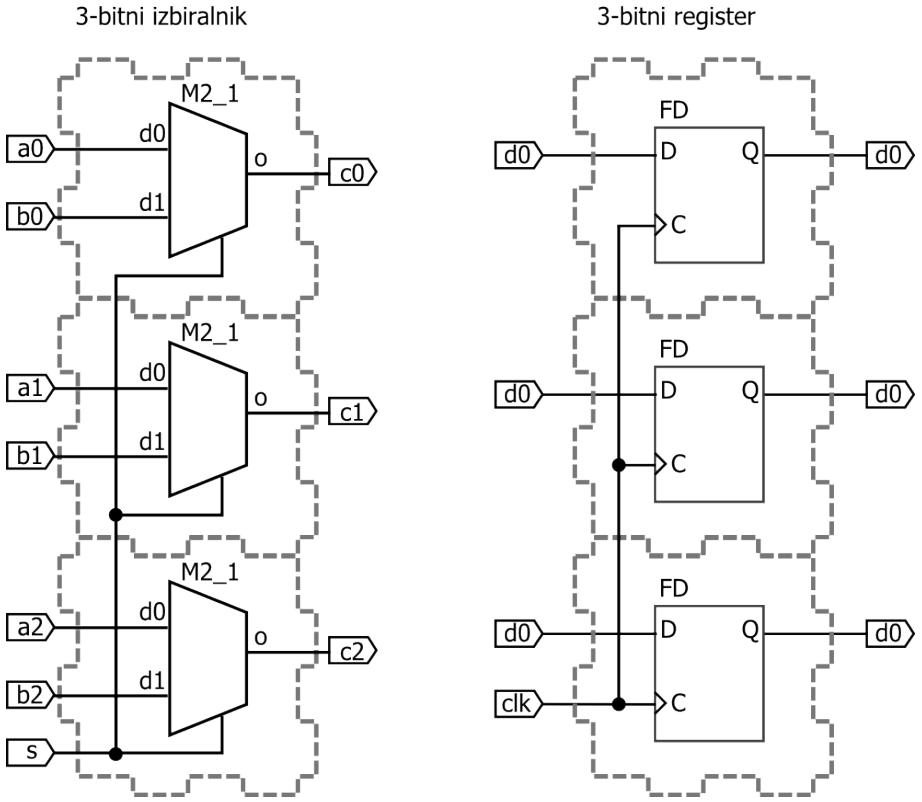
Slika 6.1: Zaporedna vezava logičnih vrat v dekodirniku Grayeve kode.

Število zaporedno vezanih kombinacijskih elementov določa število nivojev vezja (prikazani dekodirnik je 3-nivojsko vezje). Nekateri signali potujejo od vhoda čez vse elemente vezja in

imajo zato velike zakasnitve. Pri zaporedni vezavi se zakasnitve logičnih elementov seštevajo. Teoretično lahko poljubno kombinacijsko funkcijo naredimo z dvonivojskim vezjem, vendar bi marsikdaj potrebovali za izvedbo preveč logičnih elementov. Večnivojska izvedba je v nekaterih vezjih (npr. dekodirniki, seštevalniki) najboljši kompromis med velikostjo in zmogljivostjo vezja.

Z zaporedno vezavo sekvenčnih elementov vplivamo na število urnih ciklov med spremembo vhoda in spremembo na izhodu vezja. Najosnovnejši primer je zaporedna vezava flip-flopov v pomikalni register (slika 5.16). Število nivojev določa število zakasnitvenih ciklov, ne vpliva pa na najvišjo frekvenco ure. Ta je določena z zakasnitvijo znotraj flip-flopa in največjo zakasnitvijo v kombinacijskih elementih med flip-flopi.

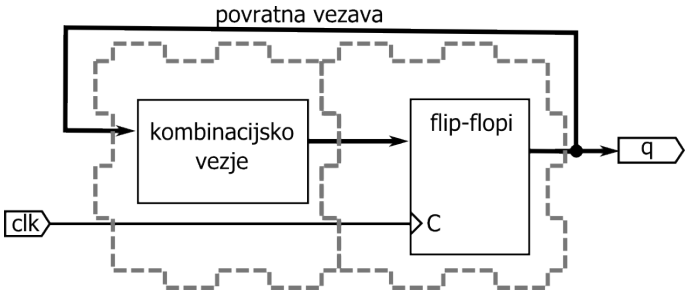
Vzporedno vezavo uporabljamo za izdelavo večbitnih struktur iz osnovnih gradnikov ali pa za hitrejša vezja. Tudi pri vzporedni vezavi so nekateri kontrolni signali gradnikov vezani skupaj, podatkovni vhodi in izhodi pa med seboj niso povezani.



Slika 6.2: Izvedba 3-bitnega izbiralnika in registra z vzporedno vezavo.

Značilnost vzporedne vezave je, da ne poveča zakasnitev signalov v primerjavi s posameznim gradnikom. V splošnem pričakujemo, da imajo večja vezja večjo zakasnitev, ker morajo signali potovati med zaporednimi gradniki ali logičnimi vrati. Vsak element prispeva nekaj zakasnitve, ki se sešteva na poti med signalnimi vhodi in izhodi vezja. Pri vzporedni vezavi pa se zakasnitve ne seštevajo, ker so podatkovni signali na posameznih gradnikih neodvisni od drugih.

Povratna vezava pripelje podatkovni izhod prek enega ali več gradnikov na vhod istega gradnika. Vezja iz logičnih vrat s povratno vezavo smo zasledili pri zgradbi osnovnih sekvenčnih gradnikov. Povratna vezava signalov povzroči ohranjanje notranjega stanja vezja, lahko pa ima tudi neželene učinke, npr. nestabilnost RS-zapaha. Kadar potrebujemo pomnilne elemente, uporabimo že narejene sekvenčne gradnike, zato v praksi nikoli ne povezujemo izključno kombinacijskih elementov v povratni vezavi.

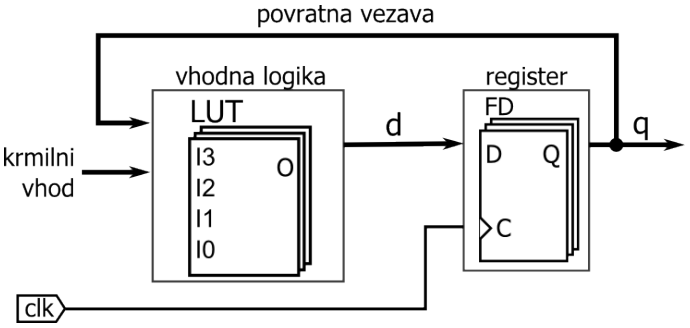


Slika 6.3: Shema sekvenčnega vezja s povratno vezavo.

Osnovno sekvenčno vezje s povratno vezavo je sestavljeno iz kombinacijske logike in flip-flopov, tako da so izhodi flip-flopov prek logike vezani na vhode. Takšna vezava se uporablja za ciklična sekvenčna vezja, kot so števci in sinhroni stroji.

6.2 Sinhrona sekvenčna vezja

Najpreprostejša sinhrona sekvenčna vezja so tista, ki ciklično spreminjajo stanje na izhodu in jih imenujemo števci. Sinhroni števci so narejeni kot digitalna vezja s povratno vezavo, na kateri mora biti vsaj en sekvenčni gradnik (register ali flip-flop). Stanje izhoda se spreminja ob fronti ure, določeno pa je z notranjim stanjem in morebitnimi krmilnimi vhodi.

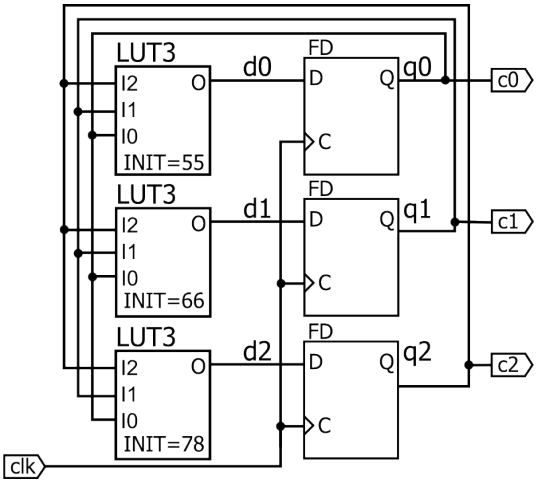


Slika 6.4: Izvedba sinhronega sekvenčnega vezja s povratno vezavo.

Sinhroni števec

Za izvedbo 3-bitnega sinhronega števca potrebujemo 3 podatkovne flip flope in 3-vhodno kombinacijsko vezje iz logičnih vrat ali vpoglednih tabel LUT. Delovanje števca opišemo s pravilnostno tabelo, ki določa vrednosti na vhodih flop-flopov ($d0$ do $d2$) glede na vrednosti trenutnih izhodov ($q0$ do $q2$). Pri binarnem števcu je vrednost v desnem stolpcu kar naslednja binarna kombinacija, zadnja kombinacija 111 pa se preslika v 000.

q2	q1	q0	d2	d1	d0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0



Slika 6.5: Tabela in zgradba 3-bitnega sinhronega števca.

Pravilnostna tabela opisuje kombinacijsko logiko na vhodih flip-flopov, ki jo najlažje naredimo s 3-vhodnimi vpoglednimi tabelami (LUT). V posamezne tabele vpišemo vrednost izhodnega stolpca pravilnostne tabele, npr. v tabelo signala $d2$ vpišemo kombinacijo 01111000.

Števec s kontrolnimi vhodi

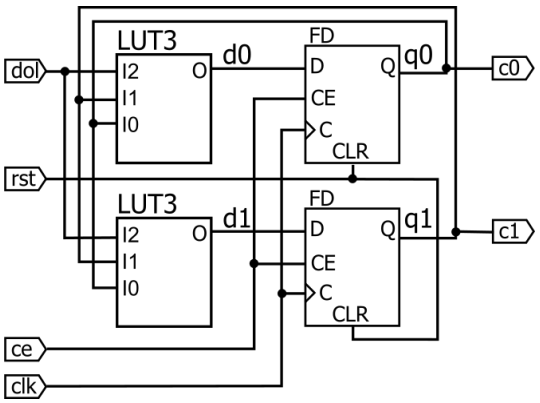
Sinhronemu števcu bomo dodali nekaj kontrolnih vhodov:

- ce – omogoči štetje, kadar je ce=1;
- dol – določa smer štetja; pri dol=1 šteje navzdol, sicer pa navzgor;
- rst – postavi izhod števca na 0.

Nekatere kontrolne signale lahko vežemo direktno na flip-flope, v našem primeru sta to signala ce in rst. Uporabiti moramo izvedbo podatkovnega flip-flopa, ki ima signal za omogočanje ure in reset signal. Signal za določanje smeri štetja pa bo vezan na kombinacijski del vezja kot dodatni vhod v vse vpogledne tabele.

Naredimo primer vezja 2-bitnega števca, ki šteje navzgor in navzdol. V pravilnostni tabeli imamo na levi strani poleg vhodov flop-flopov ($d0$ in $d1$) še signal dol. V prvih štirih vrsticah tabele je signal dol enak 0 in izhodi so enaki kot pri navadnem števcu, v drugih štirih vrsticah pa so izhodi takšni, da se smer štetja obrne.

dol	q1	q0	d1	d0
0	0	0	0	1
0	0	1	1	0
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	0	1
1	1	1	1	0

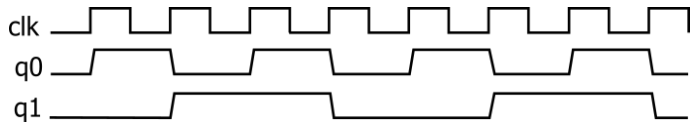


Slika 6.6: Tabela in zgradba 2-bitnega števca s kontrolnimi vhodi.

Števec po modulu in delilnik

Po opisanem postopku naredimo poljubne sinhrone števce, ki štejejo v različne smeri in z različnimi koraki štetja. Največje število bitov, ki ga potrebujemo za predstavitev izhodnega stanja (običajno je to najvišja vrednost števca), določa število izhodnih signalov, flip-flopov in vpogledne tabel.

Nekateri števci ne dajo na izhod vseh binarnih kombinacij, ker njihov cikel oz. modul štetja ni potenca števila 2, na primer števec po modulu 10 šteje od 0 do 9. Takšen števec bi uporabili za štetje enic pri digitalni uri, medtem ko bi za štetje desetic potrebovali števec po modulu 6.



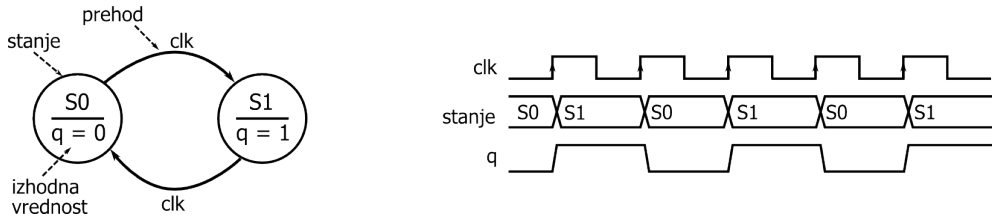
Slika 6.7: Časovni potek izhodov 2-bitnega števca.

Delilnik frekvence je primer vezja, kjer uporabimo binarni števec po določenem modulu z namenom, da zmanjšamo frekvenco ure v nekem delu vezja. Najvišji bit takšnega števca se spreminja s frekvenco, ki je enaka vhodni frekvenci deljeni z modulom štetja. Npr. dvobitni števec po modulu 4 (običajen binarni števec) spreminja izhod $q1$ s frekvenco, ki je 4-krat nižja od vhodne ure. Števec po modulu 1000 (to je 10 bitni števec) pa spreminja izhod $q9$ s 1000-krat nižjo frekvenco.

6.3 Načrtovanje z diagramom stanj

Model sinhronega sekvenčnega vezja lahko predstavimo z *diagramom stanj*. Gre za grafičen opis delovanja sistema, v katerem so prikazana stanja, izhodi in prehodi med stanji. Pri sekvenčnih vezjih je stanje kombinacija vrednosti, ki so shranjene v flip-flopih. Stanja na digramu prikazujemo s krožnicami, znotraj katerih je simbolično ime stanja, npr. $S0$, $S1$ ..., prehode med stanji pa s puščicami. Na puščice zapišemo pogoje za prehod, v katerih nastopajo vhodi v sekvenčno vezje. V vsakem stanju zapišemo vrednost izhodnih signalov pod imenom stanja.

Na sliki 6.8 je preprost diagram z dvema stanjema: $S0$ in $S1$. V stanju $S0$ je izhod q enak 0. Ob prednji fronti ure se izvrši prehod v stanje S1, v katerem se postavi q na vrednost 1. Ob naslednji prednji fronti gremo spet nazaj v S0.

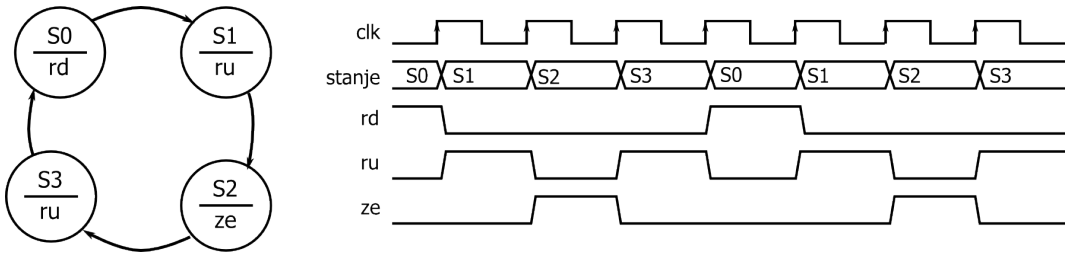


Slika 6.8: Diagram stanj in časovni potek na simulaciji.

S sledenjem prehodov v diagramu stanj razberemo časovni potek signalov v sekvenčnem vezju. Na izhodu q dobimo periodičen signal, ki je dvakrat počasnejši od vhodne ure. Prehod med stanji sinhronega vezja je vedno pogojen s fronto ure, zato pri risanju diagrama izpustimo pogoj za fronto ure pri prehodih stanj. Če je puščica brez oznake, pomeni, da se prehod v naslednje stanje izvrši ob nasledji fronti ure.

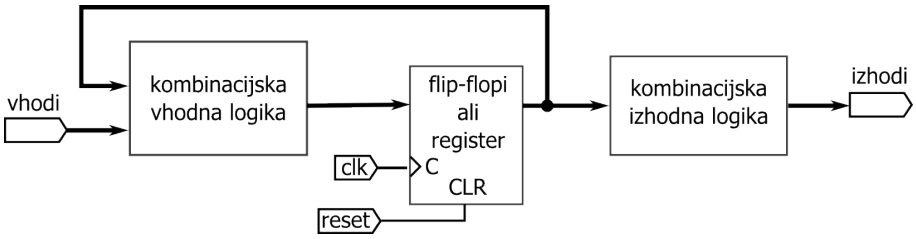
Diagram stanj pogosto uporabljamo pri načrtovanju sekvenčnih vezij, posebej pri izdelavi krmilne logike, ki generira izhodne signale v odvisnosti od vhodov in trenutnega stanja. Ogledali si bomo, kako z diagramom stanj opišemo sekvenčna vezja, ki *generirajo* izhod v danem časovnem zaporedju, in vezja, ki *zaznajo* določeno časovno zaporedje vhodnih signalov.

Narišimo diagram stanj za preprost semafor z izhodi: rd za rdečo luč, ru za rumeno in ze za zeleno. Semafor naj ima štiri stanja, ki se ciklično izmenjujejo v zaporedju: rd, ru, ze, ru.



Slika 6.9: Diagram stanj in časovni diagram za preprost semafor.

Na osnovi diagrama stanj naredimo vezje, ki ga imenujemo končni stroj stanj (angl. Finite State Machine) ali *avtomat*. Vezje je v splošnem sestavljeno iz vhodne logike, flip-flopov ali registra in izhodne logike.

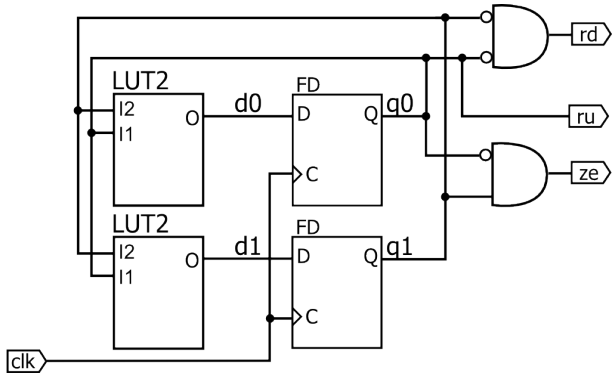


Slika 6.10: Blokovna shema končnega stroja stanj.

Iz grafičnega opisa diagrama stanj je možno avtomatsko narediti sekvenčno vezje. Programska oprema za računalniško podprto načrtovanje najprej določi število flip-flopov na podlagi števila stanj in načina kodiranja stanj. Nato določi optimalno izvedbo kombinacijske vhodne logike in dekodirnik za izhodne signale. Vezje pa seveda lahko sintetiziramo tudi sami, in sicer z uporabo postopkov načrtovanja sekvenčnih vezij.

Načrtovanje vezja začnemo s tabelo, v kateri predstavimo prehode med stanji in kodiranje stanj. Za štiri stanja potrebujemo 2-bitni register stanj. Stanje $S0$ predstavlja kombinacija 00, $S1$ kombinacija 01 itn. Na podlagi prehajanja stanj določimo vrednosti na vhodih flop-flopov ($d0$ in $d1$) glede na vrednosti trenutnih izhodov ($q0$ in $q1$). Vrednosti iz tabele zapišemo v vpogledni tabeli LUT, ki predstavljata kombinacijsko vhodno logiko.

stanje	q1	q0	d1	d0	rd	ru	ze
S0	0	0	0	1	1	0	0
S1	0	1	1	0	0	1	0
S2	1	0	1	1	0	0	1
S3	1	1	0	0	0	1	0



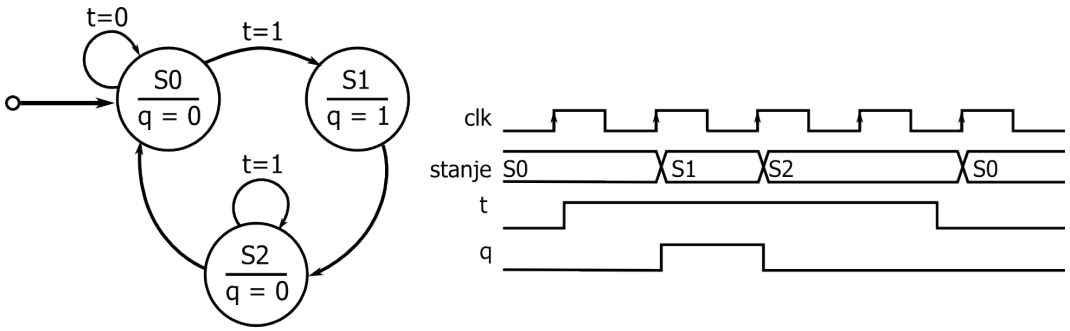
Slika 6.11: Izvedba preprostega semaforja s sekvenčnim vezjem.

Izhodni signali semaforja so odvisni od stanja. Izhodno logiko določimo kot kombinacijsko funkcijo stanja. Signal rd je ena, kadar sta $q0$ in $q1$ na 0; logika za ta signal so vrata AND z negatorji na vhodih. Iz tabele je razvidno, da se signal ru spreminja enako kot signal $q0$, zato zanj ne potrebujemo posebne logike.

6.4 Uporaba diagrama stanj

Oblikovanje impulza

Naredimo diagram vezja za oblikovanje kratkega impulza ob spremembi vhodne vrednosti na 1. Diagram vsebuje tri stanja: $S0$, $S1$ in $S2$, kot prikazuje slika 6.12. Prvo stanje $S0$ je s puščico označeno kot začetno stanje, v katerega se postavi vezje ob resetu. Če je vezje v prvem stanju in vhodni signal t na 0, ostaja v tem stanju, kar je označeno s polkrožno puščico. Ob spremembi vhoda t na 1 in fronti ure se stanje spremeni v $S1$ in izhod q gre na 1. Ob naslednji fronti ure gre v stanje $S2$, kjer je toliko časa, dokler se vhodni signal ne postavi na '0'.

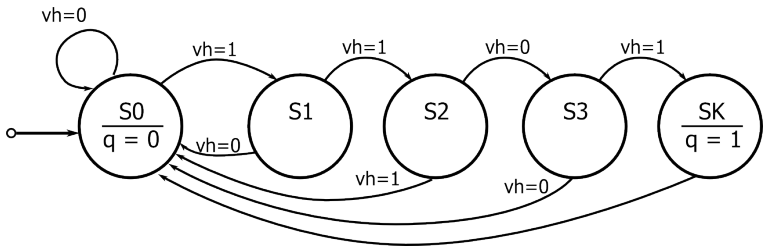


Slika 6.12: Diagram stanj in časovni diagram generatorja impulzov.

Generator kratkih impulzov je uporaben za oblikovanje signala, ki ga dobimo ob pritisku tipke. Z njim spremenimo poljubno dolg vhodni impulz v kratek izhodni impulz.

Detekcija zaporedja

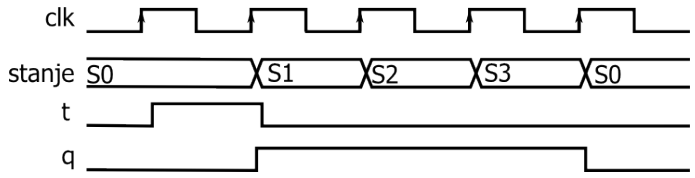
Diagram stanj uporabljamo za opis vezij, ki morajo generirati ali pa se odzivati v določenem zaporedju. Videli smo, da z diagramom lahko opišemo sekvenčno vezje, ki naredi na izhodu neko zaporedje stanj. Diagram na sliki 6.13 pa opravlja obratno nalogo: opazuje vhodni signal vh in ugotavlja, ali se spreminja po določenem zaporedju. Če se vhod ob zaporednih frontah ure postavi na vrednosti 1, 1, 0 in 1, prehaja vezje skozi vsa stanja do končnega, kjer postavi izhod q na 1. V primeru kakršne koli druge kombinacije se vrnemo v začetno stanje.



Slika 6.13: Diagram stanj za detekcijo zaporedja.

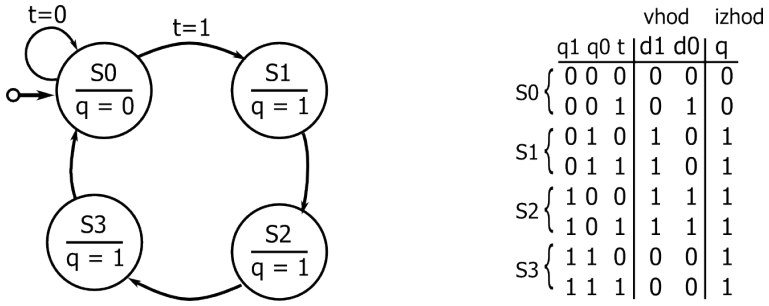
Vezje za oblikovanje daljšega impulza

Naredimo sekvenčni stroj, ki oblikuje na izhodu impulz dolžine treh urinih ciklov, kadar dobi na vhod b vrednost 1.



Slika 6.14: Časovni diagram signalov vezja za oblikovanje impulza.

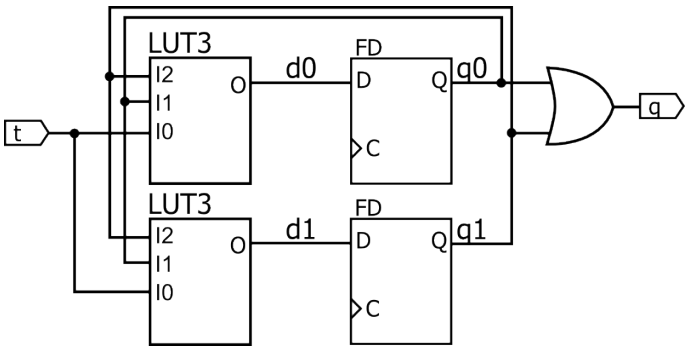
Na sliki 6.15 je diagram stanj vezja, ki vsebuje štiri stanja. Pri izdelavi tabele za vhodno in izhodno logiko moramo stanjem prirediti binarne kombinacije.



		vhod		izhod
	q1 q0 t	d1	d0	q
S0	0 0 0	0	0	0
	0 0 1	0	1	0
S1	0 1 0	1	0	1
	0 1 1	1	0	1
S2	1 0 0	1	1	1
	1 0 1	1	1	1
S3	1 1 0	0	0	1
	1 1 1	0	0	1

Slika 6.15: Diagram stanj in tabela prehajanja stanj.

Stanje $S0$ je predstavljeno s kombinacijo 00 na izhodu flip-flopov, stanje $S1$ je kombinacija 01. $S2$ je 10 in S3 je 11. Vezje ima še enobitni vhod t, ki da skupaj z dvobitnim stanjem $2^3 = 8$ kombinacij v pravilnostni tabeli. Na podlagi diagrama stanj določimo za vsako kombinacijo vrednosti na vhodu v flip-flope (naslednje stanje) in vrednosti izhoda vezja. Vhodna logika sekvenčnega vezja je narejena z dvema tabelama, izhod pa z logičnimi vrati OR.

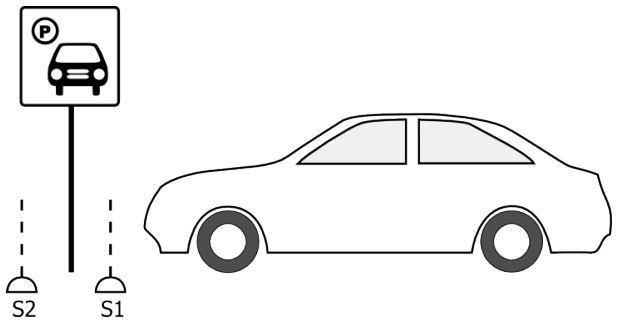


Slika 6.16: Izvedba sekvenčnega stroja za oblikovanje impulza.

Naloge

1. Koliko flip-flopov vsebuje števec s signalom za omogočanje, ki šteje od 0 do 7?

2. Nariši diagram stanj sekvenčnega vezja za zaznavanje avtomobilov, ki zapeljejo na parkirišče. Ob vhodu sta dva senzorja: S1 in S2. Sekvenčno vezje naj za en cikel postavi izhod na 1, kadar se najprej aktivira S1, nato pa S2.

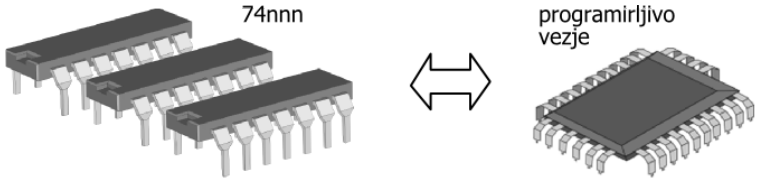


7 Programirljiva logika

Programirljiva logična vezja so elektronske komponente, v katerih s postopkom programiranja oz. konfiguracije oblikujemo digitalno vezje, da opravlja želeno funkcijo. Razlikujejo se po zgradbi, načinu programiranja in zmogljivosti. Spoznali bomo osnovne programirljive matrike PAL in PLA, programirljive naprave CPLD ter zelo zmogljive programirljive matrike FPGA.

7.1 Programirljiva integrirana vezja

Razvoj novega integriranega vezja je dolgotrajen in drag postopek, zato je veliko digitalnih naprav in sistemov narejenih z obstoječimi komponentami (angl. commercial off-the-shelf, COTS). Prve digitalne komponente v obliki integriranih vezij so bili gradniki, ki jih dobimo v družini integriranih vezij *7400*. Tiskana vezja so vsebovala veliko osnovnih integriranih vezij in inženirji so iskali možnost zamenjave množice komponent z enim integriranim vezjem.



Slika 7.1: Zamenjava osnovnih logičnih komponent z enim *programirljivim* vezjem.

Tehnologija programirljivih integriranih vezij omogoča spreminjanje delovanja že izdelanih vezij s spremembami v programski in strojni opremi. Programsko opremo lahko spreminjamo v mikroprocesorskih sistemih, strojno opremo pa v programirljivih vezjih. Novejša vezja

združujejo obe možnosti za izdelavo zelo prilagodljivih digitalnih naprav in sistemov. Prednosti programirljivih integriranih vezij:

- programirljiva vezja omogočajo hiter razvoj prototipa vezja za nov izdelek;
- stroški razvoja so nižji, ker odpade zelo draga priprava proizvodnje polprevodnikov;
- na tiskanem vezju je manj elektronskih komponent, ki so lažje nadomestljive, saj lahko načrt vezja hitro prenesemo v drugo programirljivo vezje.

Logični gradniki, ki omogočajo programiranje integriranih vezij, zasedejo velik del vezja in vezje v praksi ni nikoli 100-odstotno zasedeno. Zaradi tega imajo programirljiva vezja v primerjavi z namenskimi integriranimi vezji nekaj slabosti:

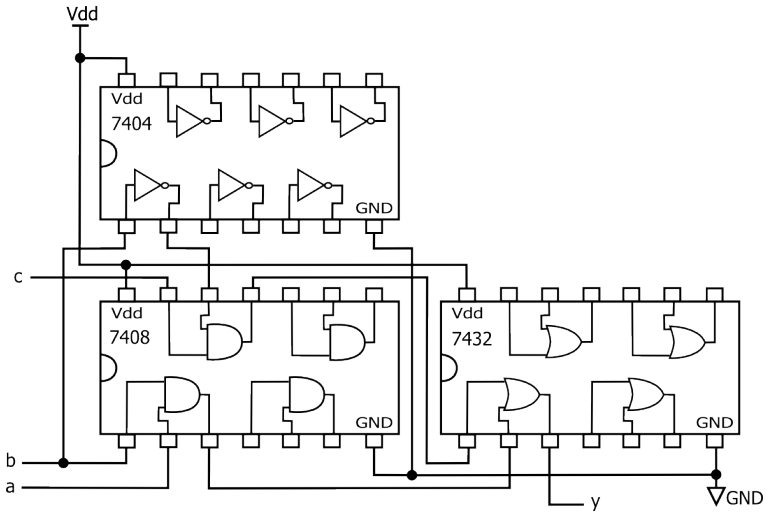
- zaradi večje površine je cena posameznega vezja višja kot cena namenskega vezja;
- programirljivi elementi vnašajo zakasnitve, zato so ta vezja nekoliko počasnejša;
- imajo večjo porabo energije.

7.2 Osnovne programirljive matrike

Poglejmo primer kombinacijskega vezja, ki izvaja logično funkcijo:

$$y = (a \text{ AND } b) \text{ OR } (c \text{ AND NOT } b)$$

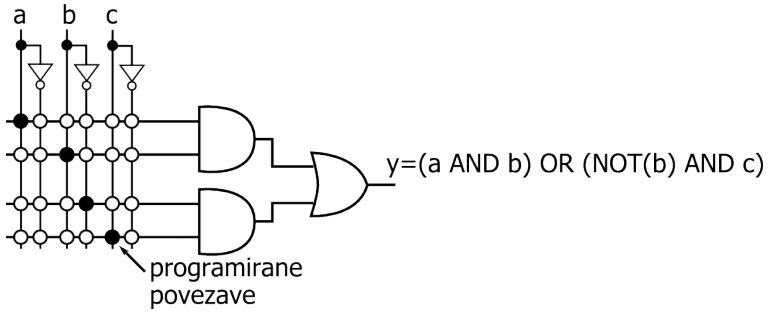
Logično vezje lahko naredimo s tremi integriranimi vezji: 7408, ki vsebuje logična vrata AND, 7432 z vrati OR in 7404, ki vsebuje negatorje.



Slika 7.2: Izvedba logičnega vezja s komponentami iz družine 7400.

Logično funkcijo lahko opišemo s tabelo, ki jo s programiranjem prenesemo v pomnilnik ROM. Vezje naredimo tako, da so vhodni signali vezani na pomnilniške naslove, izhodni pa na podatkovne izhode. Pri večjem številu vhodnih in izhodnih signalov je takšna izvedba zelo potratna glede površine integriranega vezja, sekvenčnih vezij pa s pomnilnikom ROM ne moremo narediti. Princip izdelave kombinacijskih gradnikov vezja z manjšim pomnilnikom oziroma vpogledno tabelo bomo zasledili v strukturah zmogljivih programirljivih vezij.

Kombinacijsko vezje lahko naredimo tudi s *programirljivo matriko*, v kateri programiramo povezave vhodnih signalov na logična vrat AND.



Slika 7.3: Izvedba logičnega vezja s programirljivo matriko.

Programirljiva matrika vsebuje vnaprej povezana logična vrata AND in OR, vsak vhodni signal pa je vezan direktno ali prek negatorja na vertikalne povezave. Na križiščih teh povezav s horizontalnimi povezavami proti vhodom logičnih vrat so programirljivi elementi, ki omogočajo, da povezave sklenemo ali razklenemo. Na shemi matrike so sklenjene oz. programirane povezave, narisane s polnim, razklenjene pa s praznim krogcem.

Izvedba vezja z matriko PAL

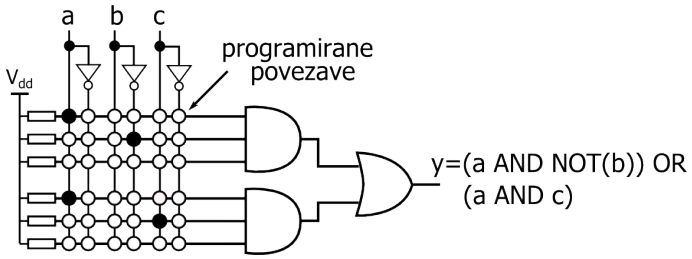
Boolova algebra pravi, da je kombinacijske funkcije vedno mogoče pretvoriti v obliko, kjer je vsak izhodni signal zapisan kot vsota (vrata OR) produktov (vrata AND) vhodov oz. negiranih vhodov. Programirljiva matrika PAL (angl. Programmable Array Logic) je sestavljena iz večvhodnih vrat AND, ki jim v postopku programiranja določimo povezave na vhodne oz. negirane vhodne signale. Pri izvedbi vezja z matriko PAL moramo kombinacijsko logiko v prvem koraku pretvoriti v primerno obliko. Tako na primer logično funkcijo za avtomobilski alarm:

$$alarm = vklop \text{ AND } (NOT(vrata) \text{ OR } gib)$$

pretvorimo v obliko:

$$alarm = (vklop \text{ AND } NOT(vrata))\text{OR}(vklop \text{ AND } gib)$$

Za izvedbo te funkcije potrebujemo dvoje dvovhodnih vrat AND, ki so vezana na vrata OR. Programirljiva matrika PAL ima v praksi logična vrata AND z nekaj 10-vhodnimi signali in večvhodna vrata OR, s katerimi naredimo uporabna kombinacijska vezja. V shemah bomo uporabljali manjše matrike zaradi nazornosti prikaza.



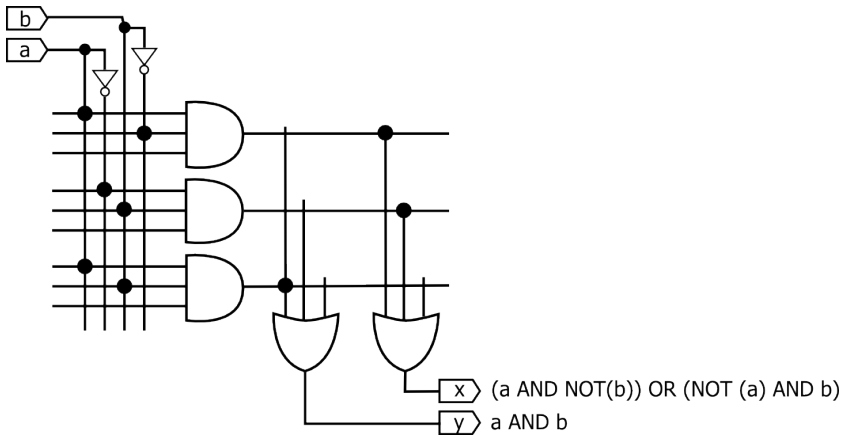
Slika 7.4: Avtomobilski alarm v programirljivi matriki PAL.

V matriki PAL, s katero bi naredili avtomobilski alarm, so prikazane programirane povezave s polnim krogcem. Na horizontalnih povezavah so tudi upori proti napajalni napetosti Vdd, ki zagotavljajo logično 1, kadar so vse povezave prekinjene. Logična 1 na posameznem vhodu vrat AND ne vpliva na funkcijo ostalih vhodov, tako npr. 3-vhodna vrata AND uporabljamo kot 2-vhodna vrata.

Programirljiva matrika PAL je najpreprostejše programirljivo vezje, ki omogoča izvedbo kombinacijskih logičnih vezij. Za izvedbo kompleksnejših funkcij potrebujemo zelo velike matrike, ki so v splošnem slabo izkoriščene. Upori proti napajalni napetosti prestavljajo stalen porabnik toka, zato so ta vezja tudi energijsko potratna. Integrirana vezja PAL se danes ne uporabljajo več, predstavili pa smo jih zato, ker na najbolj nazoren način razložijo zgradbo kompleksnih programirljivih naprav.

Izvedba vezja z matriko PLA

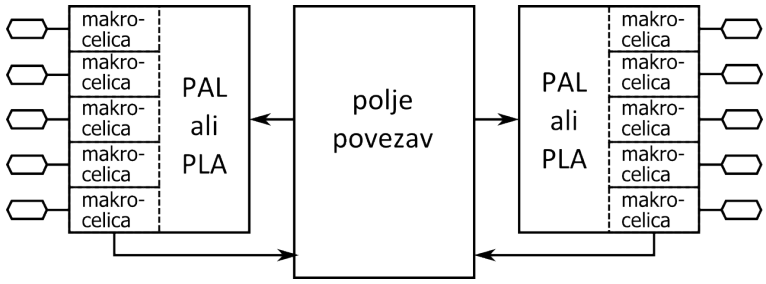
Programirljive matrike z oznako PLA (angl. Programmable Logic Array) imajo možnost programiranja povezav tako na vhodih AND kot na vhodih vrat OR. Takšna matrika doseže boljšo izkoriščenost programirljivega vezja, saj lahko nekatere produktne člene uporabimo večkrat. Primer matrike PLA, ki ima tri produktne člene ter po dva vhodna in izhodna signala, kaže slika 7.5.



Slika 7.5: Programirljiva matrika PLA.

7.3 Programirljive naprave – CPLD

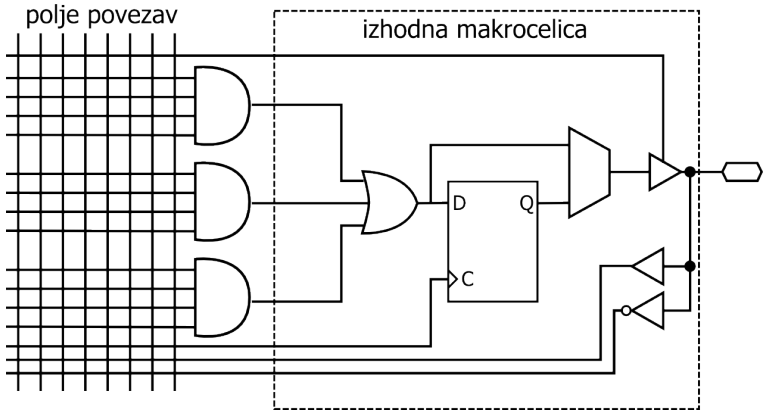
Programirljive naprave z oznako CPLD (angl. Complex Programmable Logic Device) vsebujejo več matrik PLA, ki so med seboj povezane s programirljivim povezovalnim poljem.



Slika 7.6: Blokovna shema vezja CPLD (Xilinx Coolrunner-II).

Vezja Coolrunner-II proizvajalca Xilinx vsebujejo matrike PLA, ki imajo 56 produktnih členov in 16 izhodnih signalov. Kombinacijski izhodi matrike PLA so vezani na makrocelice, ki vsebujejo pomnilne gradnike za izvedbo sekvenčnih vezij. Signali so povezani prek vhodno/izhodnih (I/O) celic na zunanje priključke ali povezovalno polje.

Kompleksne gradnike, ki jih ne moremo narediti v eni matriki PLA, razdelimo in naredimo z več matrikami. Preslikavo logične sheme v strukturo PLA opravlja programska oprema, tako da za njihovo uporabo ni potrebno podrobno poznavanje zgradbe vezij PLD.

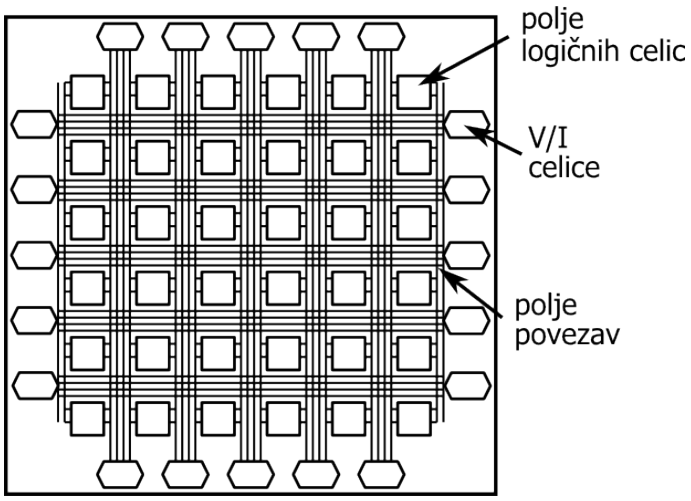


Slika 7.7: Izhodna makrocelica vezja CPLD.

Pri programiranju vezja se določijo povezave v matriki PLA ter povezave v makrocelicah in I/O-blokih ter povezovalni matriki. S takšno strukturo lahko naredimo poljubna digitalna vezja, omejeni smo le z velikostjo gradnika CPLD. Vezja CPLD omogočajo izdelavo logičnih vezij z nekaj 1000 logičnih vrat in nekaj 100 flip-flopov, ki delujejo pri frekvencah ure do okoli $200\,\mathrm{MHz}$. Imajo programski pomnilnik vrste FLASH, ki ga lahko večkrat zapišemo in ohrani vsebino ob izklopu napajanja.

7.4 Programirljiva polja vrat – FPGA

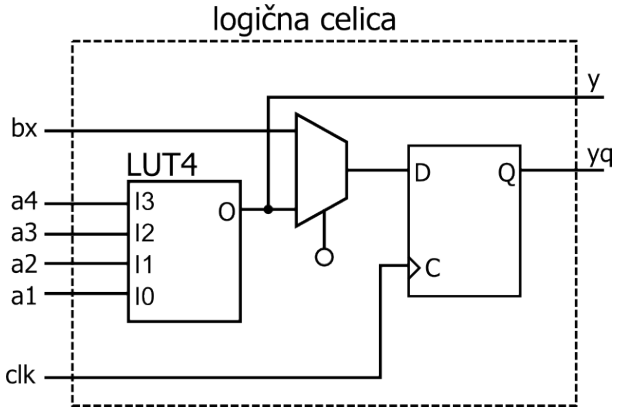
Vezja z oznako FPGA (angl. Field Programmable Gate Array) so narejena iz množice programirljivih logičnih celic in polja povezav, v katerem lahko med seboj povežemo poljubne celice. Okoli programirljive matrike so vhodno/izhodne celice, ki povezujejo signale na zunanje priključke.



Slika 7.8: Blokovna shema vezja FPGA.

Vezja FPGA vsebujejo veliko število celic in največja med njimi omogočajo izdelavo vezij z več kot 10 milijoni logičnih vrat. Sodobna vezja FPGA omogočajo izdelavo celotnih sistemov na integriranem vezju in lahko vsebujejo tudi mikroprocesorje, pomnilnike in namenske vmesnike.

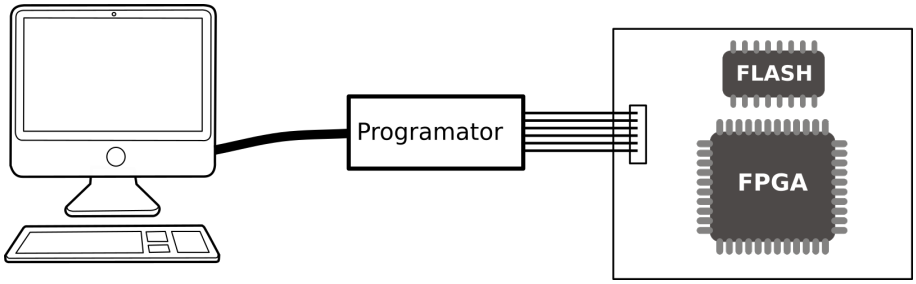
Z logičnimi celicami naredimo gradnike kombinacijskih in sekvenčnih vezij, ki jih s povezavami med celicami združujemo v digitalna vezja. Zgradba celic je odvisna od proizvajalca in družine vezij FPGA, v splošnem pa vsebujejo vpogledne tabele, programirljive izbiralnike in flip-flope.



Slika 7.9: Blokovna shema logične celice.

Struktura vezja FPGA je zelo prilagodljiva in omogoča programiranje povezav med logičnimi celicami, signalov znotraj logičnih celic in vhodno/izhodnih celic ter vsebine vpoglednih tabel. Izvedba povezav je odvisna od tehnologije programirljivega vezja.

V tehnologiji *antifuse* imajo tovarniško izdelana vezja povsod šibke povezave. Nekatere imed povezav v postopku programiranja prekinemo in tako ustvarimo želeno funkcijo. Takšna vezja lahko programiramo samo enkrat. V tehnologijah *EPROM* ali *Flash* je na mestu povezave posebno elektronsko stikalo, ki mu pri programiranju določamo stanje. Programiranje teh vezij lahko ponovimo večkrat. Največ integriranih vezij FPGA je narejenih v tehnologiji CMOS, kjer se programski podatki zapišejo v zapahe. Zapah ob izklopu napajanja izgubi shranjeno stanje, zato imamo poleg vezja FPGA na tiskanem vezju še pomnilnik Flash, iz katerega se ob zagonu naloži vsebina.

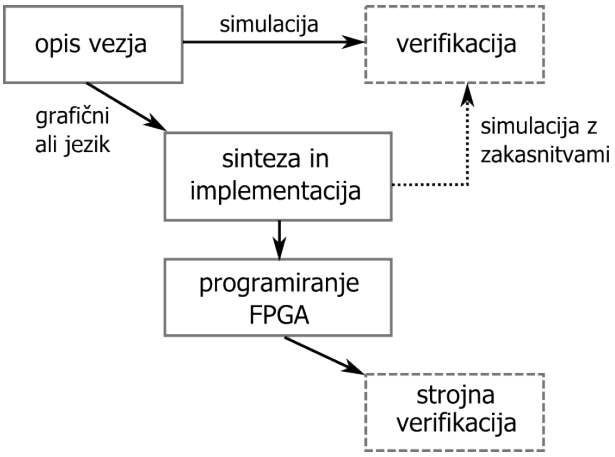


Slika 7.10: Programiranje vezja FPGA.

7.5 Računalniška orodja za programirljiva vezja

Postopek načrtovanja vezja začnemo z vnašanjem opisa vezja v računalnik. Programska oprema za računalniško načrtovanje vezij pozna različne načine vnosa vezja, ki jih v grobem razdelimo v grafični in jezikovni opis. Primer grafičnega opisa je shema vezja. Shemo narišemo v grafičnem urejevalniku s postavljanjem elementov iz knjižnice digitalnih gradnikov in risanjem povezav. Drug primer grafičnega opisa vezja je opis sekvenčnega stroja v obliki diagrama stanj. Jezikovni opis logičnega vezja predstavljajo npr. Boolove enačbe. Jezik za opis strojne opreme HDL (angl. Hardware Description Language) določa pravila takšnega opisa. Danes se uporabljata predvsem jezika VHDL in Verilog, ki omogočata opis vezja z logičnimi izrazi ali pa v obliki algoritma.

Programska oprema datoteke z opisom vezja prevede v računalniku razumljivo jezikovno obliko (HDL), ki je osnova za izvedbo simulacije in ostalih korakov prevajanja. Jezikovni opis vezja se v koraku *sinteze* pretvori v obliko, ki vsebuje vse elemente končnega vezja in povezave med njimi (angl. netlist). Ta datoteka je osnova za naslednje korake prevajanja, ki jih imenujemo tehnološka izvedba oz. *implementacija* vezja.



Slika 7.11: Osnovni koraki načrtovanja s programirljivimi vezji.

Delo razvojnega inženirja je predvsem opis vezja in preverjanje delovanja oz. verifikacija vezja. Verifikacijo najprej opravimo z računalniško simulacijo. Zapleten postopek prevajanja vezja je na srečo avtomatiziran. Programska oprema za računalniško načrtovanje vezij zahteva le nekaj nastavitev, da se implementacija izvede pod želenimi pogoji. Rezultat prevajanja lahko ponovno verificiramo s simulacijo, ki tokrat vsebuje tudi ocenjene zakasnitve gradnikov vezja. Postopek programiranja vezja je odvisen od strojne opreme in izvedbe komunikacije z računalnikom. Običajno ta korak ni zahteven, saj moramo le izbrati pravilne nastavitve in datoteko, ki naj jo programska oprema naloži v vezje.

Po programiranju preverimo delovanje vezja na strojni opremi (npr. na razvojnem sistemu). Ta postopek imenujemo strojna verifikacija. Postopek strojne verifikacije je odvisen predvsem od vrste vezja in nalog, ki jih vezje izvaja. V najpreprostejši obliki zadoščata ročno nastavljanje signalov in vizualni pregled delovanja, v zahtevnejših primerih pa potrebujemo posebno merilno opremo.

Tehnološka izvedba v programirljivem vezju

Korake tehnološke izvedbe vezja bomo predstavili na primeru majhnega digitalnega vezja, ki vsebuje kombinacijske in sekvenčne gradnike. Vezje izvaja logične operacije nad 3-bitnima vhodoma $r1$ in $r2$ glede na stanje krmilnega vhoda p. Kadar je p enak 1, se izvede operacija $r1$ AND $r2$, sicer pa $r1$ OR $r2$. Rezultat operacije se prenese na izhod vezja ($r3$) ob fronti ure.

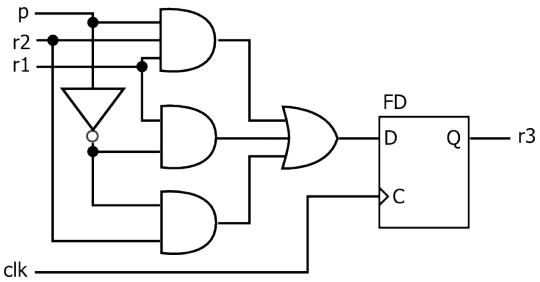
Vezje bo sestavljeno iz logičnih vrat in 3-bitnega registra. Najprej naredimo enostavnejše vezje, ki opravlja operacijo nad enobitnimi signali in shranjuje rezultat v flip-flop D. To vezje bo predstavljalo eno izmed treh celic končnega vezja in ga imenujemo *registrska celica*. Delovanje logike opišemo z enačbo:

$$r3 = (r1 \text{ AND } r2 \text{ AND } p) \text{ OR } ((r1 \text{ OR } r2) \text{ AND NOT}(p))$$

Kombinacijski izraz je sestavljen iz dveh delov. Kadar je krmilni signal p enak 0, bo vrednost prvega oklepaja 0, drugega pa $r1$ OR $r2$. Podoben razmislek naredimo pri vrednosti krmilnega

vhoda 1. Enačbo še preuredimo, da bo primerna za izvedbo s produktnimi členi programirljive matrike:

$$r3 = (r1 \text{ AND } r2 \text{ AND } p) \text{ OR } (r1 \text{ AND NOT}(p)) \text{ OR } (r2 \text{ AND NOT}(p))$$



Slika 7.12: Shema registrske celice logične računske enote.

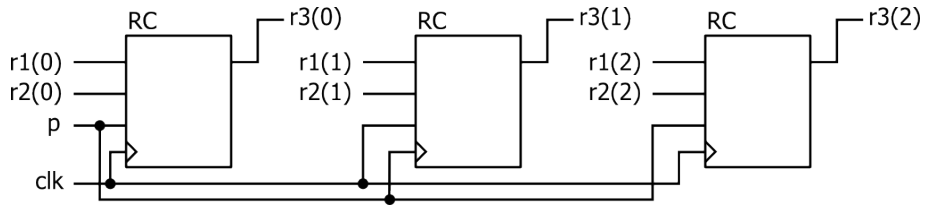
Z *vzporedno* vezavo treh registrskih celic dobimo končno blokovno shemo vezja, kot prikazuje slika 7.13. Delovanje vezja lahko opišemo tudi v jeziku VHDL. Sestavljen je iz deklaracije priključkov (*port*) in arhitekturnega dela, v katerem je obnašanje vezja opisano s pogojnimi stavki (*if*). Programska oprema iz tega opisa sintetizira logično shemo vezja.

Listing 7.1: Opis logične računske enote v jeziku VHDL.

```
entity vezje is
    Port ( clk, p : in  std_logic;
           r1, r2 : in  std_logic_vector(2 downto 0);
           r3     : out  std_logic_vector(2 downto 0));
end vezje;
architecture opis of vezje is
begin
 p: process(clk)
 begin
   if rising_edge(clk) then
     if p='1' then
       r3 <= r1 and r2;
     else
       r3 <= r1 or r2;
     end if;
   end if;
 end process;
end opis;
```

Preslikava v CPLD

Kombinacijski del vezja naredimo s programirljivo matriko. Če primerjamo shemo registrske celice s strukturo programirljive naprave CPLD, ugotovimo, da opisano celico izvedemo z eno



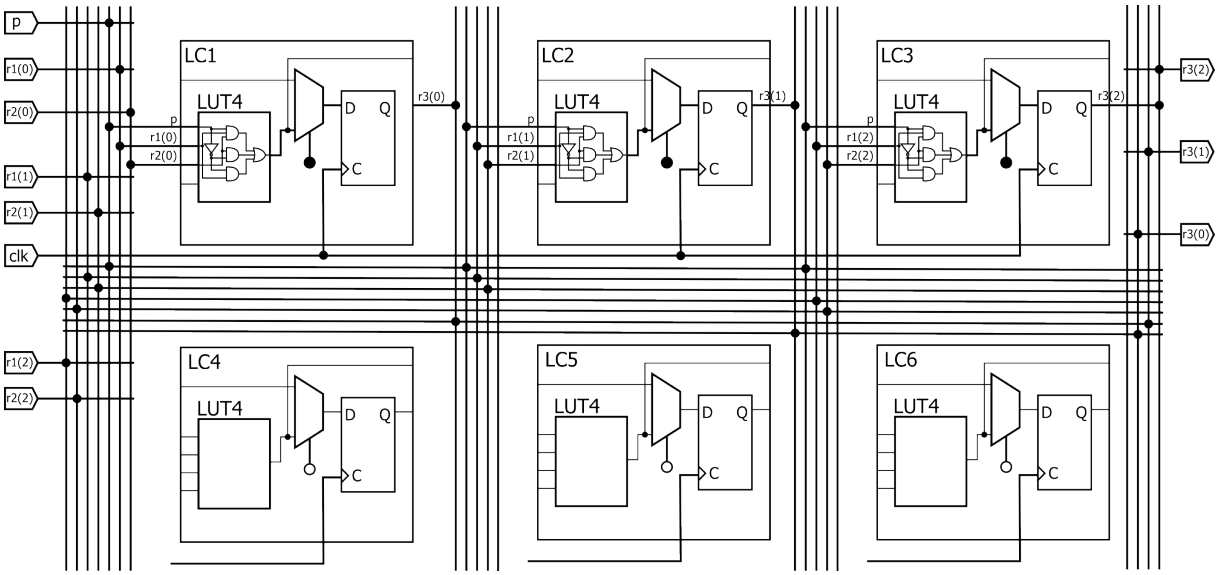
Slika 7.13: Shematski opis 3-bitne logične računske enote.

makrocelico. Celotno vezje naredimo s tremi makrocelicami vezja CPLD, v katerih morajo biti ustrezno nastavljene matrike PLA in podatkovne poti prek flip-flopov. Vzporedno vezavo vhodnih in krmilnih signalov izvedemo v polju povezav.

Naloga programske opreme za tehnološko preslikavo vezja v strukturo CPLD je pripraviti ustrezne oblike (AND–OR) zapisa logičnih izrazov ter dodeliti makrocelice in povezave v polju. Če bi imeli kompleksnejše kombinacijske izraze, bi jih morala programska oprema razdeliti na manjše dele, ki jih lahko preslika v posamezno makrocelico. Sekvenčnim gradnikom dodeli toliko makrocelic, kolikor je vseh flip-flopov v vezju.

Preslikava v FPGA

Za izvedbo s programirljivim vezjem FPGA je treba shemo vezja preslikati v logične celice. Kombinacijsko logiko posamezne registrske celice preslikamo v vpogledno tabelo (LUT), pri kateri uporabimo tri vhode, izhod pa vežemo na flip-flop v logični celici. Vidimo, da bo tudi izvedba celotnega vezja v FPGA zasedla tri logične celice v matriki. Povezave vhodnih in izhodnih signalov pa naredimo s poljem povezav, kot prikazuje slika 7.14.

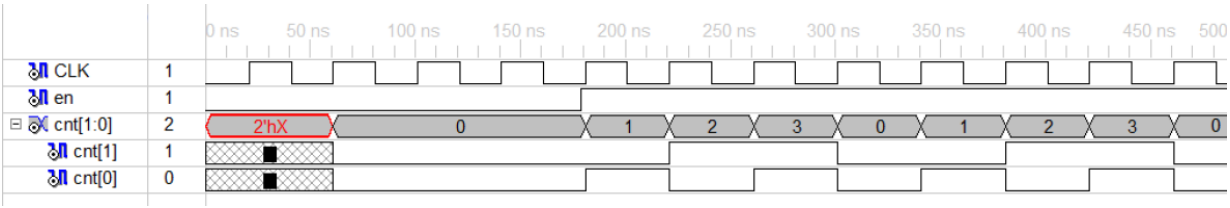


Slika 7.14: Izvedba vezja v FPGA.

Verifikacija

Računalniška simulacija modela vezja je osnovni postopek preverjanja oz. verifikacije vezja. Simulacijo pripravimo tako, da določimo časovni potek spreminjanja signalov na vhodu vezja. Časovni potek vhodnih signalov določimo v testni strukturi (angl. test bench) in je zapisan v grafični obliki, v jeziku HDL ali pa v obliki simulacijskih makrojev. Ko je testna struktura pripravljena, poženemo simulator in pregledamo rezultate na časovnem diagramu signalov (angl. waveform).

Slika 7.15 prikazuje časovni diagram simulacije 2-bitnega števca. V testni strukturi smo nastavili uro in signal en, opazujemo pa vrednost izhoda cnt. Večbitne signale, kot je $cnt(1:0)$, lahko opazujemo v obliki dvojiških, desetiških ali šestnajstiških številskih vrednosti, možen pa je tudi prikaz posameznih bitov $cnt(1)$ in $cnt(0)$.



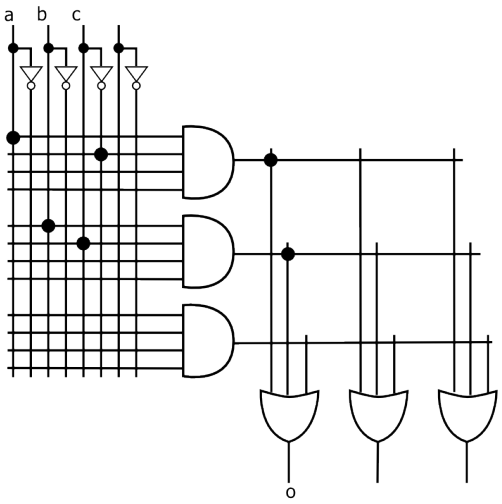
Slika 7.15: Rezultat simulacije 2-bitnega števca.

S pregledom časovnega diagrama ugotovimo, ali se vezje obnaša tako, kot smo pričakovali. V testni strukturi lahko tudi vnaprej predpišemo pričakovane vrednosti izhodov ali pa določimo pravila spreminjanja signalov in tako avtomatiziramo postopek verifikacije. Če želimo z verifikacijo res preveriti delovanje vezja, moramo dobro poznati signale, ki jih lahko pričakujemo na vhodu realnega vezja, in pripraviti testne strukture za veliko različnih primerov.

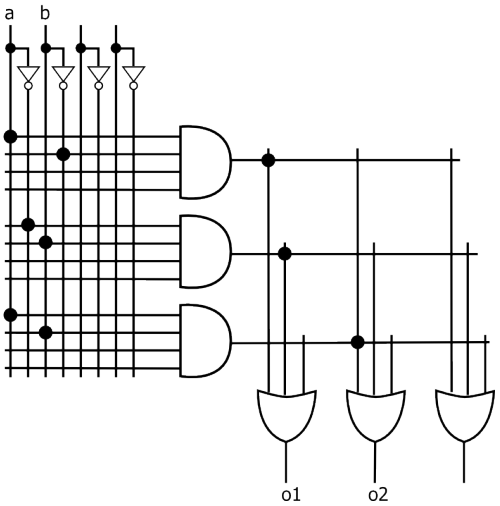
Naloga

1. Zapiši funkcijo logičnih vezij, ki sta narejeni s programirljivo matriko PLA.

a)



b)



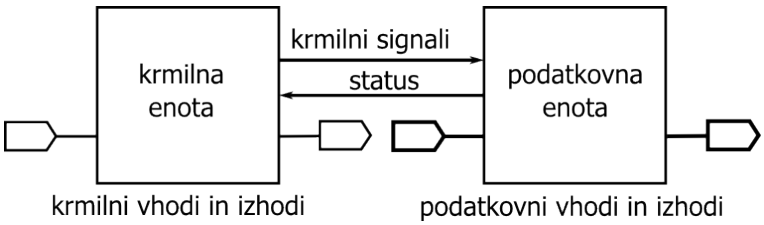
a) o = _____

b) o1 = _____

o2 = _____

8
Digitalni sistemi

Digitalni sistem je kompleksno digitalno elektronsko vezje, narejeno za izvedbo ene ali več nalog. Sestavljen je iz podsistemov ali enot za prenašanje, obdelovanje podatkov in nadzor oz. krmiljenje. Slika 8.1 prikazuje delitev digitalnega sistema na podatkovno in krmilno enoto. Krmilna enota sprejema krmilne (kontrolne) vhode in oddaja izhode za interakcijo z drugimi deli sistema. Krmilne signale pošilja v podatkovno enoto in iz nje dobiva informacijo o stanju (status). Tako aktivira različne operacije obdelave podatkov. Delovanje krmilne enote določa diagram stanj ali pa program v primeru *programirane* krmilne enote.



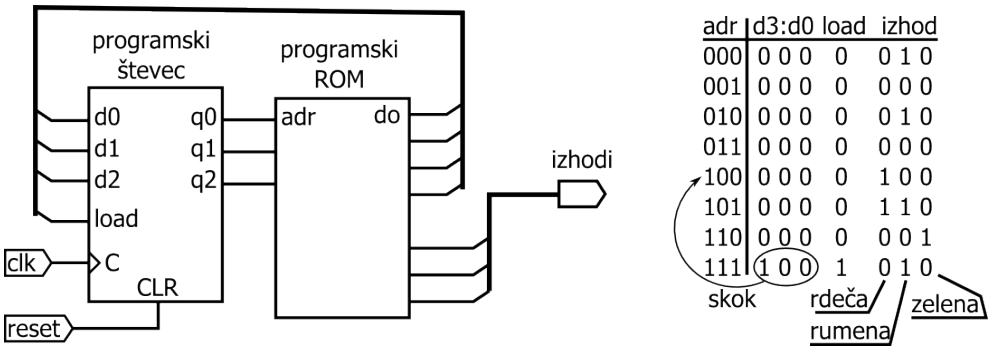
Slika 8.1: Delitev digitalnega sistema na podsisteme.

Podatkovna enota prenaša in obdeluje podatke z vhodov in jih pošilja na izhode. Delovanje enote določajo registri in operacije nad podatki, ki jih hranijo registri. Obravnavali bomo registrske *mikrooperacije*: prenos, seštevanje, odštevanje, logične operacije in pomikanje podatka. Predstavili bomo mikroprocesorske sisteme in delovanje centralne procesne enote.

8.1 Krmilna enota

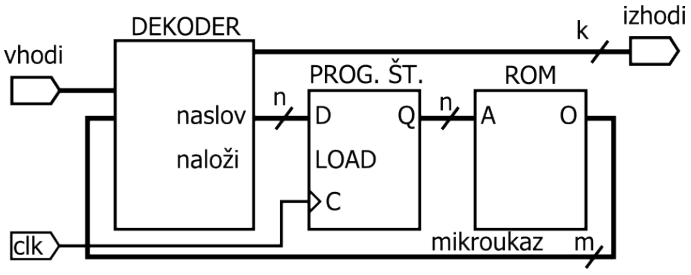
Krmilna enota je sekvenčno vezje, ki nadzira delovanje digitalnega sistema. Enoto z vnaprej definiranimi krmilnimi operacijami naredimo v obliki sekvenčnega stroja. Kadar potrebujemo večjo prilagodljivost in možnost spreminjanja krmilnega postopa, naredimo *mikroprogramirano* krmilno enoto. *Mikroprogramirana* krmilna enota ali mikrosekvenčnik ima krmilni postopek zapisan v programskem pomnilniku. Najpreprostejša izvedba vsebuje pomnilnik vrste ROM in programski števec v povratni vezavi.

Za ilustracijo si bomo ogledali delovanje enostavnega mikrosekvenčnika za krmiljenje semaforskih luči, ki je sestavljen iz 3-bitnega programskega števca in pomnilnika. V pomnilniku so shranjeni *mikroukazi*, predstavljeni v tabeli slike 8.2. Programski števec naslavlja pomnilnik in s tem poskrbi, da se ob uri izvaja zaporedje mikroukazov: ko je signal load na 0, se izvajajo mikroukazi po vrsti, pri 1 pa se izvede skok na poljuben naslov.



Slika 8.2: Mikrosekvenčnik s 3-bitnim izhodom in tabela za izvedbo semaforja.

Ob resetu se začnejo po vrsti izvajati mikroukazi, ki najprej povzročijo utripanje rumene luči, potem pa vklapljanje luči od rdeče do zelene. Zadnji mikroukaz povzroči, da se izvajanje nadaljuje na ukazu z binarnim naslovom 100, kar sproži ponoven cikel prižiganja luči.



Slika 8.3: Mikroprogramirana krmilna enota z ukaznim dekodirnikom.

Kompleksnejše enote vsebujejo še kombinacijski dekodirnik. Dekodirnik krmili programski števec glede na vrednosti trenutnega mikroukaza in krmilnih vhodov. Mikroukazi so v tem primeru zapisani bolj kompaktno, z manj biti, saj ni treba da bi posamezen bit neposredno krmilil programski števec ali izhode. Tako so narejene tudi krmilne enote mikroprocesorjev.

8.2 Registrske mikrooperacije

Prenos podatkov med registri in obdelavo podatkov imenujemo *registrski prenos* ali z angleško kratico RT (Register Transfer). Gradniki RT-vezij so registri, ki izvajajo eno ali več elementarnih operacij. Elementarne operacije ali *mikrooperacije* so npr. nalaganje podatka v register ali prištevanje k vrednosti v registru. Mikrooperacije se izvedejo v enem ciklu na vseh podatkovnih bitih hkrati. Rezultat operacije se lahko shrani nazaj v register ali pa prenese v naslednji register. Spoznali bomo štiri vrste mikrooperacij: prenos, aritmetične, logične in mikrooperacije pomika.

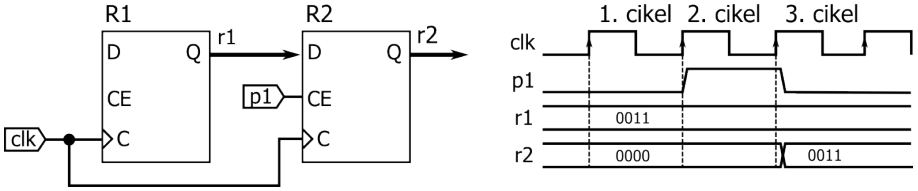
Registrski prenos

Vzemimo dva registra $R1$ in $R2$, ki imata izhodna signala $r1$ in $r2$ z enakim številom podatkovnih bitov. Prenos podatka iz enega v drug register zapišemo z izrazom:

$$r2 \leftarrow r1$$

Register $R1$ predstavlja izvor podatkov, register $R2$ pa ponor ali ciljni register. Vsebina registra, ki je izvor podatkov, se pri prenosu ne spremeni, spremeni se le podatek v ciljnem registru. Podatki se prenesejo ob fronti ure, na katero sta vezana oba registra. Če ne želimo, da se prenos izvede ob vsakem urnem ciklu, uporabimo pogojni registrski prenos. Pogoj je logični izraz, ki je vezan na kontrolni vhod CE drugega registra. Slika 8.4 prikazuje vezje iz dveh registrov, ki prenese podatek v drug register, kadar je vrednost kontrolnega signala $p1$ enaka 1.

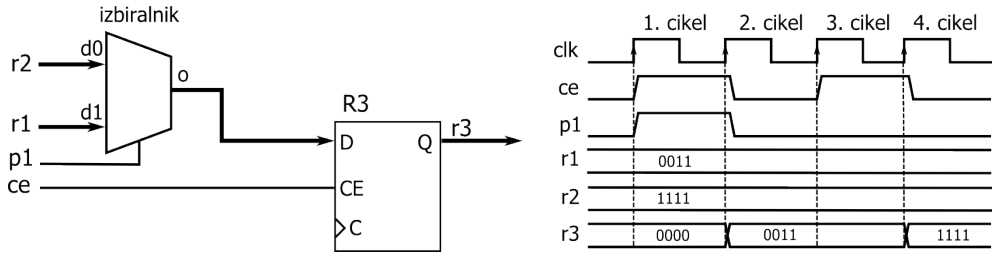
$$r2 \leftarrow r1 \mid p1 = 1$$



Slika 8.4: Vezje za pogojni prenos podatkov med dvema registroma.

V vezju so običajno vsi signali sinhronizirani z uro, zato tudi pričakujemo, da se pogoj $p1$ aktivira ob naraščajoči fronti ure. V časovnem diagramu na sliki 8.4 se $p1$ postavi na 1 v drugem urnem ciklu. Prenos podatka pa se naredi ob naraščajoči fronti tretjega cikla. V sinhronih vezjih vedno velja, da se registrski prenos izvrši en cikel kasneje, kot je izpolnjen pogoj. Razlog za takšno delovanje so zakasnitve, ki jih ima vsako realno vezje. Če se pogoj aktivira ob nekem urnem ciklu, bo zaradi zakasnitev vrednost kontrolnega signala postavljena na 1 za fronto ure, drug register pa bo sprejel novo vrednost šele ob naslednji naraščajoči fronti.

Nadgradnja osnovnega prenosa je izbirni prenos, kjer imamo več izvorov, iz katerih se podatek shrani v register. Osnovno vezje je sestavljeno iz registra in izbiralnika.



Slika 8.5: Vezje za registrski prenos z izbirnim signalom in časovni diagram.

V ciljni register $R3$ se ob aktivnem signalu $ce = 1$ shrani podatek iz vodila $r1$, kadar je $p1 = 1$, sicer pa iz vodila $r2$. Na časovnem diagramu vidimo, da se dejanski prenos podatka izvede en cikel za tem, ko sta postavljena kontrolna signala ce in $p1$. Mikrooperacijo za izbirni prenos s pogojem $p1$ zapišemo:

$$r3 \leftarrow r1 \mid p1 = 1 \text{ AND } ce = 1$$

$$r3 \leftarrow r2 \mid p1 = 0 \text{ AND } ce = 1$$

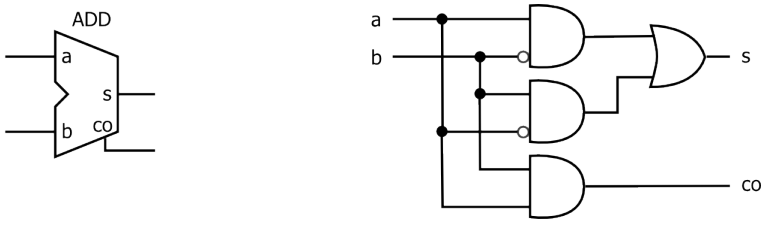
Aritmetične mikrooperacije

Osnovna aritmetična operacija je seštevanje. Naredimo izračun vsote dveh enobitnih dvojiških vrednosti pri vseh možnih kombinacijah:

$$\begin{array}{r} 0 \\ +\,0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 0 \\ +\,1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\,0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\,1 \\ \hline 10 \end{array}$$

prenos

Pri zadnji kombinaciji smo dobili prenos, ki ga upoštevamo kot dodatno števko. Logično vezje za izračun vsote se imenuje seštevalnik. Slika 8.6 prikazuje simbol in logično shemo enobitnega seštevalnika. Na vhodu sta operanda $a0$ in $b0$, na izhodu pa je vsota $s0$ (angl. sum) in prenos $c0$ (angl. carry).

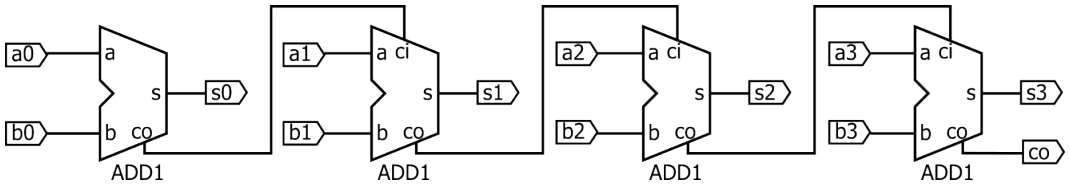


Slika 8.6: Simbol in logična shema enobitnega seštevalnika.

Postopek seštevanja večmestnih števil poznamo že iz ročnega seštevanja v desetiškem sistemu: števili poravnamo in seštevamo števke od desne proti levi. Če pride pri vsoti števk do prenosa, ga upoštevamo pri naslednji vsoti:

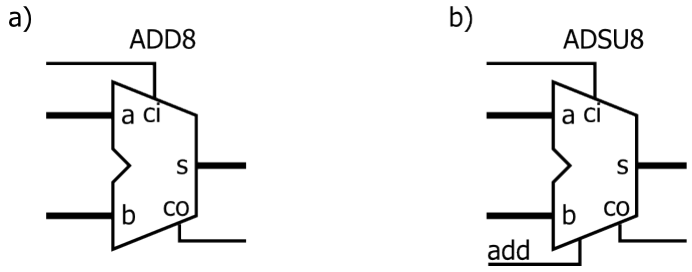
$$\begin{array}{r} 0\,0\,1\,1 \\ +\,0\,0\,0\,1 \\ \hline 0 \end{array} \quad \begin{array}{r} 0\,0\,1\,1 \\ +\,0\,0\,0\,1 \\ \hline 0\,0 \end{array} \quad \begin{array}{r} 0\,0\,1\,1 \\ +\,0\,0\,0\,1 \\ \hline 1\,0\,0 \end{array} \quad \begin{array}{r} 0\,0\,1\,1 \\ +\,0\,0\,0\,1 \\ \hline 0\,1\,0\,0 \end{array}$$

Vezje večbitnega seštevalnika naredimo s povezavo enobitnih seštevalnikov z vhodnim in izhodnim prenosom. Seštevalniki ADD1 na sliki 8.7 računajo vsoto operandov a, b in vhodnega prenosa ci. Vsak izmed seštevalnikov izračuna en bit vsote. Po seštevanju nižjih bitov se izhodni prenos upošteva kot vhod v naslednji seštevalnik. Vsoto sestavljajo biti od $s0$ do $s3$ in izhodni prenos co.



Slika 8.7: Shema 4-bitnega seštevalnika.

Seštevalniki so pogosto uporabljeni elementi, zato dobimo v knjižnicah osnovnih gradnikov že pripravljene večbitne seštevalnike. Slika 8.8 a) prikazuje simbol seštevalnika ADD8 z 8-bitnimi vhodi a in b ter izhodom s. Seštevalnik ima tudi vhod in izhod za prenos, tako da lahko z zaporedno vezavo zgradimo še večje seštevalnike.

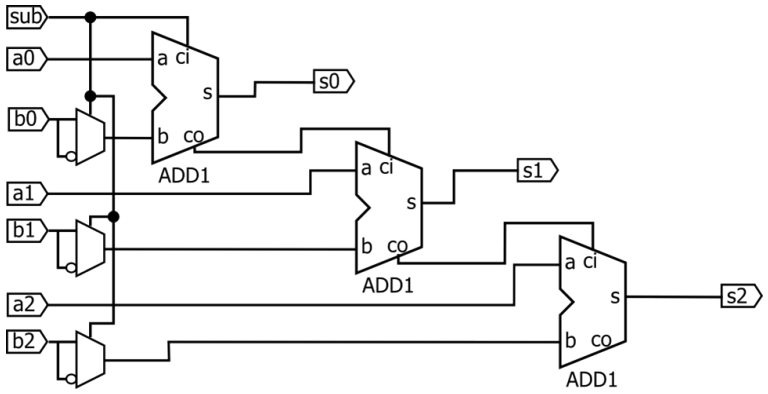


Slika 8.8: Simboli: a) 8-bitni seštevalnik in b) 8-bitni seštevalnik/odštevalnik.

Enak simbol se uporablja tudi za splošno aritmetično enoto, ki izvaja različne računske operacije nad večbitnimi signali. Slika 8.8 b) prikazuje simbol 8-bitne seštevalno/odštevalne enote ADSU8. Dodatni vhod add določa, ali enota izvaja operacijo seštevanja ($add = 1$) ali odštevanja ($add = 0$). Odštevanje binarnih števil naredimo s pomočjo dvojiškega komplementa, tako da odštevanec negiramo in naredimo vsoto z vhodnim prenosom:

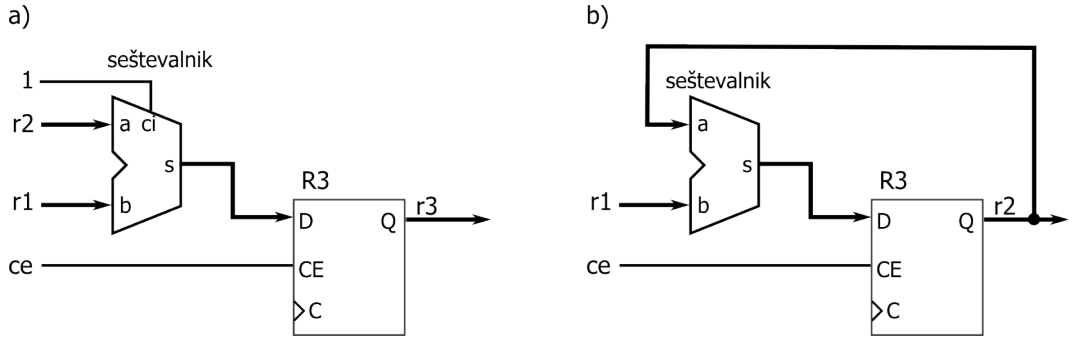
$$\begin{array}{r} 0\,0\,1\,1 \\ -\,0\,0\,0\,1 \\ \hline 0 \end{array} \quad \begin{array}{r} 0\,0\,1\,1 \\ +\,1\,1\,1\,0 \\ \hline 1 \\ 0 \end{array} \begin{array}{l} \leftarrow \text{negacija} \\ \leftarrow \text{prenos} \end{array} \quad \begin{array}{r} 0\,0\,1\,1 \\ +\,1\,1\,1\,0 \\ \hline 1\,1\,1\,1 \\ 0\,0\,1\,0 \end{array}$$

Slika 8.9 prikazuje izvedbo 3-bitnega gradnika za izračun vsote ali razlike. Kadar je signal sub na 0, bo vezje izračunalo vsoto vhodov a in b; kadar je sub enak 1, pa bodo prišle na vhod seštevalnikov negirane vrednosti b, na vhod prvega pa še prenos in rezultat bo razlika vrednosti.



Slika 8.9: Izvedba 3-bitnega seštevalno-odštevalnega gradnika.

Če povežemo na izhod večbitnega seštevalnika register, dobimo aritmetično mikrooperacijo. Seštejemo lahko izhodne vrednosti dveh registrov in vsoto shranimo v tretjega ($r3 \leftarrow r1 + r2$) ali pa enemu registru prištejemo vsoto drugega ($r2 \leftarrow r2 + r1$). Vezje se v drugem primeru imenuje akumulator in vsebuje povratno zanko.



Slika 8.10: Vezje za mikrooperacijo seštevanja: a) vsota $r1 + r2 + 1$ in b) akumulator.

Akumulator v praktični izvedbi potrebuje začetno vrednost v registru, ki jo naložimo prek dodatnega vhodnega izbiralnika ali pa s signalom reset:

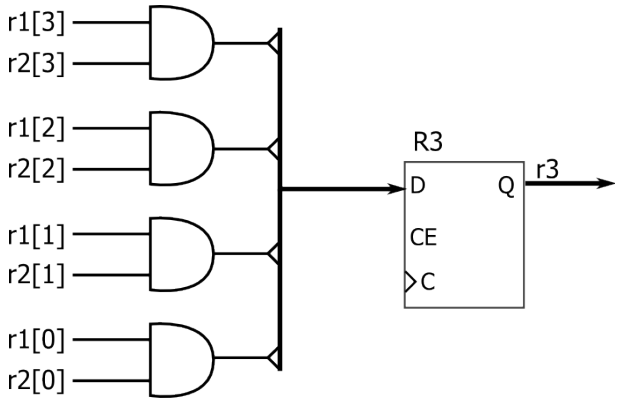
$$r2 \leftarrow r2 + r1 \mid reset = 0$$
$$r2 \leftarrow 0000 \mid reset = 1$$

Odštevanje binarnih vrednosti naredimo z uporabo dvojiškega komplementa. Vrednost, ki jo želimo odšteti, invertiramo in ji prištejemo 1 (vhodni prenos ci), nato pa seštejemo z drugo vrednostjo: $r3 \leftarrow (\text{ NOT } r1 + 1) + r2$.

Povečevanje (angl. increment) in zmanjševanje (angl. decrement) sta aritmetični mikrooperaciji, pri katerih je eden izmed vhodov konstantna vrednost, najpogosteje kar 1. Kadar prištevamo ali odštevamo vrednost k istemu signalu, dobimo dvojiški števec. Množenje in deljenje ne spadata med osnovne mikrooperacije, ker jih lahko naredimo z zaporedjem osnovnih operacij; množenje je narejeno kot zaporedno seštevanje in pomikanje vrednosti. Kadar potrebujemo v vezju hiter množilnik ali delilnik, ga naredimo v obliki paralelnega kombinacijskega vezja. Ko se rezultat pojavi na izhodu kombinacijskega množilnika ali delilnika, ga ob fronti ure shranimo v register in v tem primeru vezje predstavlja mikrooperacijo.

Logične mikrooperacije

Logične mikrooperacije so uporabne za manipulacijo s posameznimi biti v registru, saj se logična operacija izvaja nad vsakim bitom posebej. Osnovne so invertiranje ($r2 \leftarrow$ NOT $r1$), logična *in* ($r3 \leftarrow r1$ AND $r2$) ter logična *ali* ($r3 \leftarrow r1$ OR $r2$).



Slika 8.11: Izvedba logične mikrooperacije.

Vezje logičnih mikrooperacij vsebuje paralelno vezana logična vrata in register. Invertiranje je npr. uporabno pri izvedbi mikrooperacije odštevanja z uporabo dvojiškega komplementa. Drugi dve logični mikrooperaciji pa sta uporabni za maskiranje bitov. Če imamo npr. 8-bitno vrednost $r1$ in bi želeli dobiti rezultat, pri katerem so zgornji 4 biti postavljeni na 0, spodnji pa enaki spodnjim bitom signala $r1$, naredimo operacijo:

$$r3 \leftarrow r1 \text{ AND } 00001111$$

Operacija se izvrši nad istoležnimi biti:

$r1$ (podatek)	10100101
$r2$ (maska)	00001111
$r3$ (rezultat)	00000101

Kadar izvajamo maskiranje z operacijo *in*, se pobrišejo vsi biti, ki imajo v maski vrednost 0. Obratno je pri maskiranju z operacijo *ali*, kjer se vsi biti, ki so v maski postavljeni na 1, tudi na rezultatu postavijo na 1.

Mikrooperacije pomika

Z mikrooperacijami pomika spreminjamo mesta posameznih bitov v registru. Osnovni mikrooperaciji sta pomik vseh bitov za eno mesto v desno in pomik vseh bitov za eno mesto v levo. Pomik za eno mesto v desno predstavlja deljenje številske vrednosti z 2, pomik v levo pa množenje vrednosti z 2.

Poglejmo primer pomika 4-bitne vrednosti $r1$ za eno mesto v desno:

$r1$ (podatek)	1010
$r2$ (pomaknjena vrednost)	0101

Pomaknjena vrednost ima najvišji bit vedno postavljen na 0, nato pa sledijo zgornji trije biti vhodne vrednosti, najnižji bit pa pri pomikanju izpade. Enačba mikrooperacije za pomik 4-bitne vrednosti r1 za eno mesto v desno je:

$$r2 \leftarrow \{0, r1[3..1]\}$$

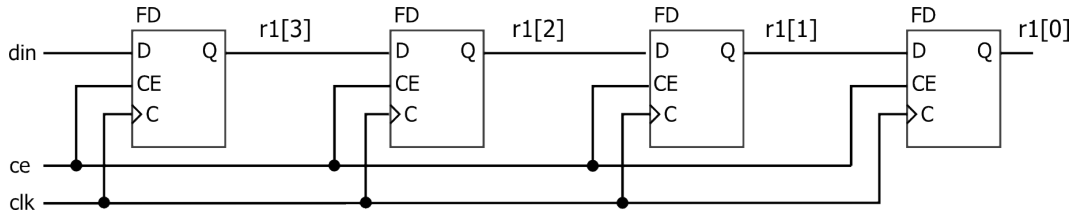
Rezultat je sestavljen iz dveh delov, ki smo ju zapisali v zavitem oklepaju. Prvi del je konstanta 0, drugi del pa sestavljajo biti vektorja $r1$ z indeksi od 3 do 1. Podobno velja pri pomiku za eno mesto v levo, le da tokrat izpade najvišji bit in dodajamo ničlo na desni strani:

$$r2 \leftarrow \{r1[2..0], 0\}$$

Vezje za osnovni mikrooperaciji pomika je zelo preprosto, saj vsebuje le register, ki ima na vhodu ustrezno povezane bite in konstanto 0. Vezje za pomikanje, pri katerem sta izvor in cilj v istem registru, se imenuje pomikalni register. Pri pomikalnem registru moramo poskrbeti, da nanj naložimo podatek. Glede na način nalaganja ločimo paralelne in serijske pomikalne registre.

Slika 8.12 prikazuje 4-bitni serijski pomikalni register, ki izvaja naslednjo operacijo:

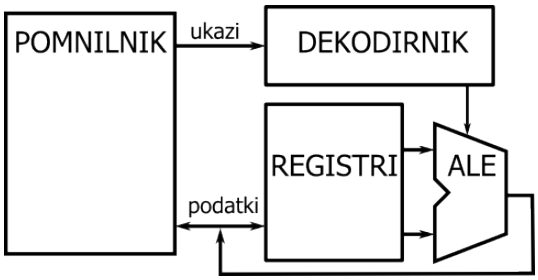
$$r1 \leftarrow \{din, r1[3..1]\} \mid ce = 1$$



Slika 8.12: Serijski pomikalni register.

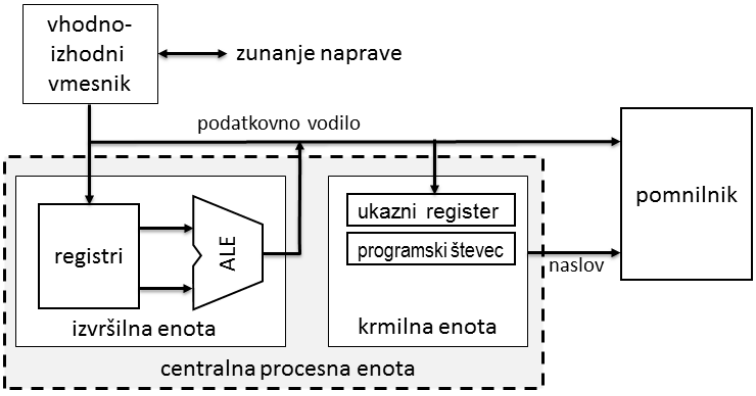
8.3 Mikroprocesorski sistemi

Mikroprocesorski sistemi so večnamenski digitalni sistemi, ki izračunajo in obdelajo podatke po ukazih, shranjenih v pomnilniku. Izraz *mikroprocesor* označuje integrirano vezje, v katerem je računalniška procesna enota. Mikroprocesor prenaša iz pomnilnika podatke in ukaze, ki jih dekodira, in izvede ustrezne mikrooperacije z registri in računskimi enotami. Rezultate začasno shrani v registre ali prenese v pomnilnik.



Slika 8.13: Prenos ukazov in podatkov v mikroprocesorju.

Jedro mikroprocesorja je *centralna procesna enota* (CPE), ki je sestavljena iz krmilne enote in podatkovne poti. Krmilna enota vsebuje ukazni dekodirnik in sekvenčno vezje s programskim števcem, ki določa korake izvajanja ukazov. Izvršilna enota se nahaja na podatkovni poti procesorja in vsebuje registre ter aritmetično-logično enoto (ALE) za obdelavo podatkov. Podatki se prenašajo prek vhodnih in izhodnih vmesnikov.



Slika 8.14: Zgradba preprostega mikroprocesorskega sistema.

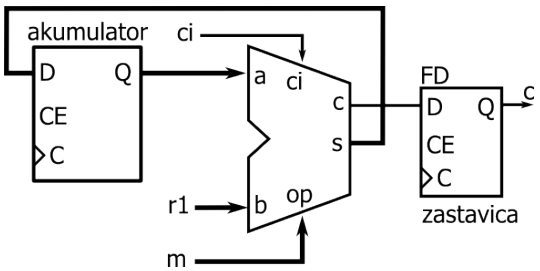
Mikroprocesor izvaja računalniški *program* v obliki zaporedja strojnih ukazov. Procesorji vrste RISC (angl. Reduced Instruction Set Computer) imajo majhen nabor strojnih ukazov z enostavnimi operacijami, ki se izvedejo v enem ali dveh korakih. Procesorji CISC (angl. Complex Instruction Set Computer) pa poznajo veliko strojnih ukazov, med katerimi so kompleksni ukazi, ki potrebujejo več korakov za izvajanje. Zmogljivi mikroprocesorji imajo lahko več jeder in računskih enot, tako da zmorejo izvajati več ukazov hkrati.

Aritmetično-logična enota

Aritmetično-logična enota (ALE) je srce mikroprocesorja, v katerem se odvija obdelava podatkov. Osembitni mikroprocesor ima ALE z 8-bitnim podatkovnim vhodom in izhodom, 32-bitni mikroprocesor pa ima 32-bitno ALE. Tipična ALE ima dva podatkovna vhoda a in b, podatkovni izhod s, kontrolni vhod za izbiro operacij op, vhodni ci in izhodni prenos c. Enota izvaja različne mikrooperacije, kot so na primer:

$s \leftarrow a$	prenos podatka
$s \leftarrow a + b$	seštevanje
$s \leftarrow a - b$	odštevanje
$s \leftarrow$ NOT a	logična negacija
$s \leftarrow a$ AND b	logični *in*
$s \leftarrow a$ OR b	logični *ali*
$s \leftarrow \{0, a[7..1]\}$	pomik v desno
$s \leftarrow \{a[6..0], 0\}$	pomik v levo

Rezultat operacije ALE shranimo v registru, izhodni prenos pa v flip-flopu, imenovanem prenosna zastavica. V praksi ima ALE še druge zastavice: ničelna zastavica se postavi na 1, kadar je rezultat operacije enak 0, negativnostna zastavica pa označuje, da je rezultat negativno število. Izhodni prenos uporabimo pri kombiniranju operacij, npr. naredimo seštevanje s prenosom iz prejšnje vsote. Prenos dobimo tudi pri operacijah pomikanja, pri katerih shranimo v prenos tisti bit, ki pri pomikanju vrednosti izpade.

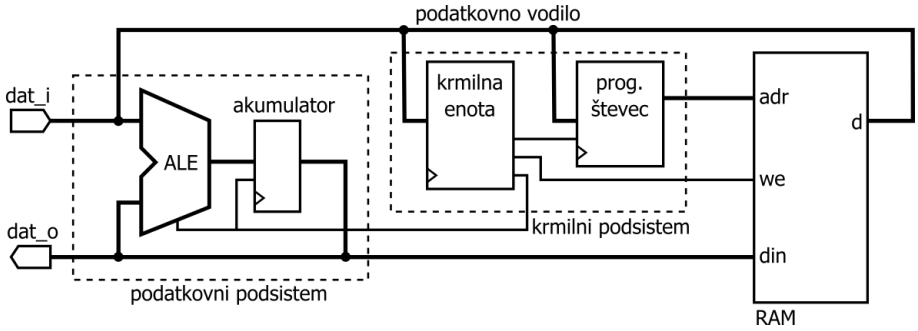


Slika 8.15: Aritmetično-logična enota z akumulatorjem.

Aritmetično-logična enota je lahko v vezavi s povratno zanko, pri kateri se rezultat vedno shrani v register z imenom akumulator, kot prikazuje slika 8.15. Vezava ALE z akumulatorjem je najpreprostejša, ni pa najbolj optimalna, ker mora mikroprocesor ob izvajanju algoritma pogosto prenašati podatke med akumulatorjem in pomnilnikom. Sodobni mikroprocesorji imajo v podatkovnem podsistemu večje število registrov, ki začasno shranjujejo podatke med izvajanjem operacij z ALE.

8.4 Centralna procesna enota

Delovanje centralne procesne enote si bomo podrobneje ogledali na primeru enostavne učne enote CPE4. Ukazi učne enote so sestavljeni iz 4-bitne mikrooperacije in naslova operanda, ki ga prenaša med pomnilnikom in podatkovnim podsistemom. Podatkovni del CPE4 sestavlja aritmetično-logična enota z akumulatorjem in n-bitno (n=8,12,16) podatkovno vodilo. V pomnilniku vrste RAM so shranjeni ukazi in podatki. Delovanje CPE4 časovno usklajuje krmilna enota, programski števec pa določa naslov trenutnega ukaza.



Slika 8.16: Blokovna shema centralne procesne enote CPE4.

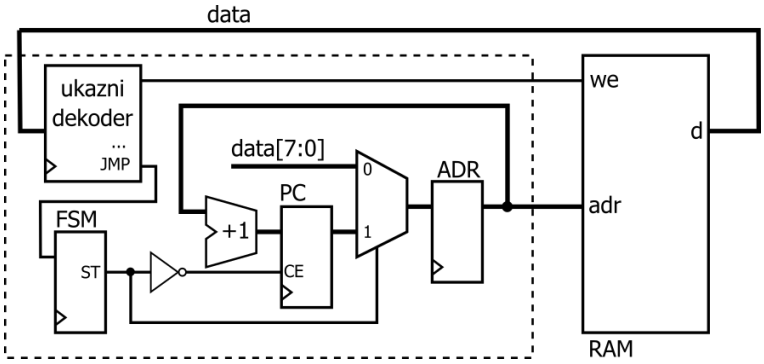
V centralni procesni enoti se izvajajo binarni strojni ukazi, katerih dolžina je odvisna od širine podatkovnega vodila. Ker je strojna koda težko berljiva, jo zapišemo raje v obliki programa v *zbirniku* (angl. assembler), ki ga prevajalnik prevede v strojno kodo.

Ukazi za izvedbo računskih operacij dobijo en operand iz akumulatorja, drugega pa prenesejo iz pomnilnika na določenem naslovu *m(Adr)*. Rezultat operacij se shrani v akumulator. Poglejmo nekaj osnovnih ukazov v zbirniku:

mikrooperacija	**zbirnik**	**pomen**
$a \leftarrow m(Adr)$	LDA Adr	naloži vrednost v akumulator
$m(Adr) \leftarrow a$	STA Adr	shrani akumulator v pomnilnik
$a \leftarrow a + m(Adr)$	ADD Adr	prištej vrednost k akumulatorju
$a \leftarrow a - m(Adr)$	SBT Adr	odštej vrednost od akumulatorja
$a \leftarrow$ NOT a	NOTA	naredi negacijo
$a \leftarrow a$ OR $m(Adr)$	ORA Adr	naredi logično *ali* operacijo
$a \leftarrow a$ AND $m(Adr)$	ANDA Adr	naredi logično *in* operacijo
$a \leftarrow \{a[n-2..0], 0\}$	SHL	pomik akumulatorja v levo
$a \leftarrow \{0, a[n-1..1]\}$	SHR	pomik akumulatorja v desno

V programih potrebujemo poleg računskih tudi skočne ukaze, s katerimi naredimo vejitve programske kode. Procesor CPE4 pozna brezpogojni skok (JMP Adr) in dva pogojna skočna ukaza. Ukaz JZE Adr izvede skok, kadar je v akumulatorju vrednost 0, ukaz JCS Adr pa, kadar je postavljena zastavica prenosa.

Krmilni podsistem procesorja CPE4 vsebuje ukazni dekoder, sekvenčni stroj (FSM), programski števec (PC) in pomnilniški naslovni register (ADR). Programski števec se po vsakem ukazu poveča za 1, razen pri skočnem ukazu, ko iz podatkovnega vodila naloži nov naslov.



Slika 8.17: Shema krmilnega podsistema procesorja CPE12.

Sekvenčni stroj skrbi za časovne cikle oz. korake izvajanja ukazov. V stanju zajemi se iz pomnilnika preneseta in dekodirata ukaz in naslov, na katerem se nahaja podatek. V stanju izvedi pa se iz pomnilnika prenese podatek in izvede ustrezna operacija. Slika 8.18 prikazuje diagram prehajanja stanj in primer izvajanja kratke kode, ki ima tri ukaze:

```
        LDA M1
zanka:  ADD M2
        JMP zanka
```

Slika 8.18: Diagram stanj krmilne enote in časovni potek izvajanja ukazov.

Za komunikacijo z okolico ima procesor vhodne in izhodne enote. Vhodni vmesnik procesorja prenaša podatke s priključka dat_i v akumulator, izhodni vmesnik pa iz akumulatorja na priključek dat_o z ukazi:

mikrooperacija	**zbirnik**	**pomen**
$a \leftarrow dat\_i$	INP ID	prenos iz vhodne enote ID
$dat\_o \leftarrow a$	OUTP ID	prenos iz registra na izhodno enoto ID

Izhodne enote krmilijo vmesnike, npr. analogno-digitalni pretvornik, serijski vmesnik za prenos podatkov po eni povezavi, pulzno-širinski modulator za pulzno krmiljene enote. Do različnih vmesnikov dostopa CPE z izbiro naslova (ID), ki je del strojnega ukaza INP ali OUTP.

8.5 Obdelava podatkov

Digitalni sistemi izvajajo obdelavo podatkov v binarni obliki: obdelavo signalov, digitaliziranega zvoka, slike ipd. Mikroprocesorski sistemi izvajajo obdelavo podatkov z zaporednimi ukazi v centralni procesni enoti.

Centralna obdelava podatkov

Poglejmo primer algoritma, ki iz vhodnega vmesnika prebere štiri vrednosti, izračuna njihovo povprečje in ga pošlje na izhod. Program prebere prva dva podatka, naredi njuno vsoto, nato pa prebere in prišteje še tretji in četrti podatek. Povprečje dobimo tako, da skupno vsoto delimo s 4, kar storimo z dvema zaporednima pomikoma vrednosti za eno mesto v desno.

Listing 8.1: Program v zbirniku za izračun povprečja štirih vrednosti.

```
start:  INP VHD ; beri 1. podatek
        STA SUM ;   shrani
        INP VHD ; beri 2. podatek
        ADD SUM ; dodaj k trenutni vsoti
        STA SUM ;   shrani vsoto
        INP VHD ; beri 3. podatek
        ADD SUM ; dodaj k trenutni vsoti
        STA SUM ;   shrani vsoto
        INP VHD ; beri 4. podatek
        ADD SUM ; dodaj k trenutni vsoti
        SHR     ; deli z 2
        SHR     ; deli z 2
        STA SUM ;   shrani rezultat
        OUTP IZH ; na izhod
```

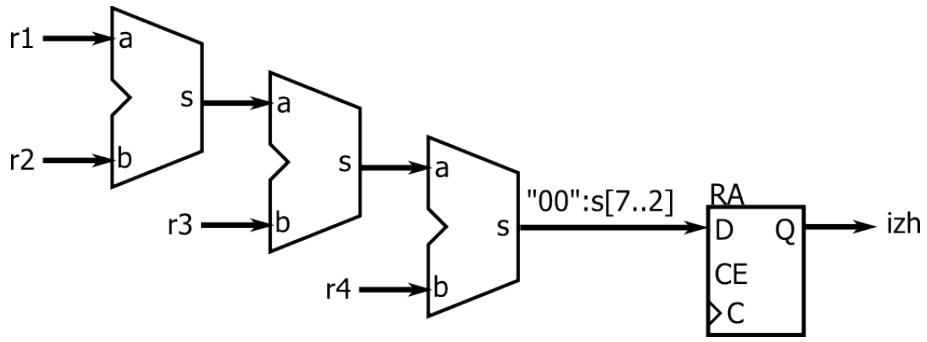
Centralna procesna enota iz pomnilnika nalaga ukaze, jih dekodira in pripravi kontrolne signale, ki izvršijo mikrooperacijo v podatkovni podatkovni poti ter shrani rezultat v izbrani register. Izvajanje ukazov poteka v več korakih, ki jih lahko opazujemo na simulatorju procesorja. Celoten program se izvede v osemindvajsetih urnih ciklih.

Mikroprocesorski sistem s centralno obdelavo podatkov je zelo fleksibilen, saj lahko njegovo delovanje hitro spremenimo s spremembo programa, ki ga naložimo v pomnilnik. Takšni sistemi se zato vgrajujejo v veliko različnih naprav in predstavljajo pomemben del digitalnih elektronskih vezij. Narejeni so v obliki mikroprocesorjev na integriranih vezjih, ki jih programiramo tako, da naložimo nov program v pomnilnik.

Glavna slabost sistemov s centralno obdelavo podatkov je v tem, da potrebujejo veliko urnih ciklov za izvajanje algoritma, ki ga razstavimo na osnovne mikrooperacije. Frekvence ure vgrajenih mikroprocesorjev so nekaj 10 ali nekaj 100 MHz, kar povsem zadostuje za veliko aplikacij. Novejša tehnologija integriranih vezij omogoča doseganje višjih frekvenc delovanja, vendar pa z višjo frekvenco narašča tudi poraba energije. V prenosnih baterijsko napajanih napravah zaradi tega ne moremo uporabljati najhitrejših in najzmogljivejših procesorjev.

Porazdeljena obdelava signalov

Kadar imamo v vezju zelo hitre signale, kot sta npr. hitri podatkovni tok in video signal, moramo uporabiti vezja s porazdeljeno oz. vzporedno obdelavo signalov, saj zaporedna obdelava na CPE ni dovolj hitra. Vzemimo primer izračuna povprečja iz prejšnjega poglavja, ki ga lahko najdemo v aplikaciji za digitalno obdelavo video slike. Vezje naredimo iz treh seštevalnikov in registra, kot prikazuje slika 8.19.



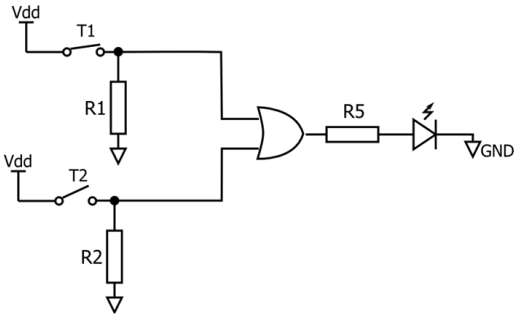
Slika 8.19: Vezje za vzporedni izračun povprečja štirih vrednosti.

Če imamo hkrati na voljo vse štiri vrednosti vhodnih podatkov (signali $r1$, $r2$, $r3$ in $r4$), dobimo rezultat v enem urnem ciklu. Tudi če potrebujemo več ciklov, da na vhode pripeljemo vse vrednosti, je vezje hitrejše od mikroprocesorja, saj ne potrebujemo ciklov za shranjevanje vmesnih rezultatov, pa tudi pomikanje je narejeno le z ustreznimi povezavami. Takšno vezje lahko pri manjši frekvenci ure obdela hiter tok video podatkov v realnem času.

Slabost porazdeljene obdelave signalov je, da potrebujemo več računskih enot in s tem večje vezje, ter da moramo za vsako spremembo algoritma razviti novo vezje. Kadar uporabljamo programirljive digitalne sisteme z vezji CPLD ali FPGA, je tudi takšna rešitev dovolj prilagodljiva, ker omogoča poljubno spreminjanje in nalaganje novega vezja. Glavna slabost je višja cena vezja v primerjavi z mikroprocesorji in omejitev velikosti porazdeljenega vezja, ki pa jo nekoliko kompenzira hiter razvoj tehnologije integriranih vezij.

Vaje

1. Dve tipki sta povezani na vhod vrat OR, ki imajo na izhodu LED. Tipka T1 je sklenjena, T2 pa razklenjena, kot prikazuje shema. Kaj se zgodi, ko sklenemo tudi na tipko T2 ?



   (a) LED zasveti

   (b) LED se ugasne

   (c) stanje LED se ne spremeni

   (d) LED ostane ugasnjena

2. Na vhod logičnega gradnika s statičnimi parametri: $V_{dd} = 3.3\,\mathrm{V}$, $V_{IH} = 2\,\mathrm{V}$, $V_{IL} = 0.8\,\mathrm{V}$ pripeljemo signal z napetostjo $U = 2.5\,\mathrm{V}$ proti masi. Kako bo gradnik ta signal intepretiral:

   (a) kot visoko stanje (logična 1)

   (b) kot nizko stanje (logična 0)

   (c) kot nedovoljeno stanje

   (d) kot kratek stik

3. Ali lahko povežemo izhod logičnega gradnika s parametri: $V_{dd} = 3.3V$, $V_{OH} = 3V$, $V_{OL} = 1V$ z vhodom gradnika, ki ima parametre: $V_{dd} = 5V$, $V_{IH} = 3V$, $V_{IL} = 2V$ in pričakujemo, da se bodo logični nivoji pravilno prenašali?

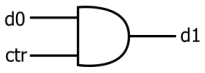
   (a) da

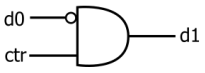
   (b) ne, ker smo v prepovedanem območju

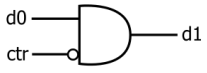
   (c) ne, ker sta napajalni napetosti različni

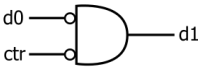
   (d) ne, ker so vse napetosti različne

4. Katera izmed vezav logičnih vrat AND deluje kot stikalo, ki ob pogoju ctr=0 prenese vhodni signal d0 na izhod, ob ctr=1 pa da na izhod logično 0 ?

a.


b.


c.


d.


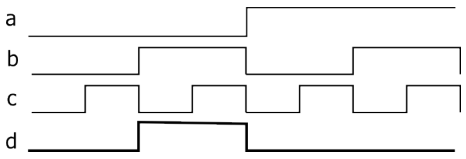
5. Ugotovi, katero logično funkcijo predstavlja časovni diagram.

   (a) $c = a$ AND b AND c

   (b) $c = a$ AND NOT(b)

   (c) $c =$ NOT(a) AND b

   (d) $c = a$ AND NOT(b) AND NOT(c)



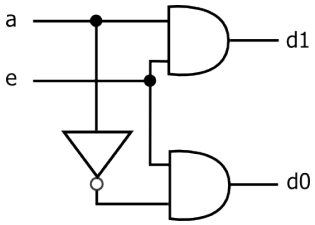
6. Kakšen bo izhod vezja, ko je na vhodu $a = 1, e = 1$?

   (a) $d0 = 0, d1 = 0$

   (b) $d0 = 1, d1 = 0$

   (c) $d0 = 0, d1 = 1$

   (d) $d0 = 1, d1 = 1$



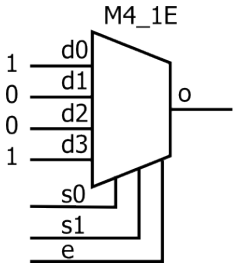
7. Na podatkovnih vhodih izbiralnika so konstantne vrednosti, kot prikazuje slika. Pri kateri kombinaciji kontrolnih vhodov bo izhod enak 1?

   (a) $s0 = 0, s1 = 0, e = 0$

   (b) $s0 = 0, s1 = 0, e = 1$

   (c) $s0 = 0, s1 = 1, e = 1$

   (d) $s0 = 1, s1 = 0, e = 1$



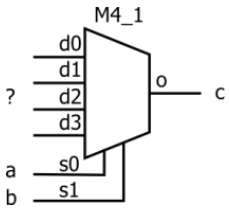
8. Katera kombinacija mora biti na vhodih d0 do d3, da se bo izbiralnik obnašal enako kot logična vrata OR2 ?

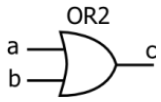
   (a) $d0 = 0, d1 = 1, d2 = 1, d3 = 1$

   (b) $d0 = 0, d1 = 0, d2 = 0, d3 = 1$

   (c) $d0 = 1, d1 = 0, d2 = 0, d3 = 0$

   (d) $d0 = 1, d1 = 1, d2 = 0, d3 = 0$





9. Koliko izbiralnikov potrebujemo za izdelavo dekodirnika, ki ima 3-biten vhod in 3-biten izhod?

   (a) 3 izbiralnike 4-1

   (b) 3 izbiralnike 8-1

   (c) 8 izbiralnikov 4-1

   (d) 8 izbiralnikov 8-1

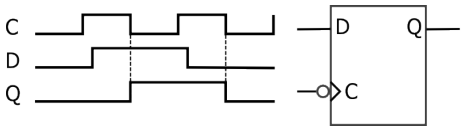
10. Kateri pomnilni gradnik predstavlja časovni diagram?

    (a) zapah D

    (b) zapah SR

    (c) flip-flop D prožen na prednjo fronto

    (d) flip-flop D prožen na zadnjo fronto



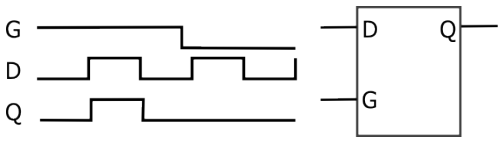
11. Kateri pomnilni gradnik predstavlja časovni diagram?

    (a) zapah D

    (b) zapah SR

    (c) flip-flop D prožen na prednjo fronto

    (d) flip-flop D prožen na zadnjo fronto



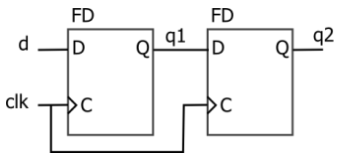
12. Poglej vezje in časovni diagram ter ugotovi, katerim signalom pripadajo oznake 1., 2. in 3.

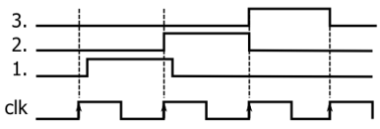
    (a) 1 = q1, 2 = q2, 3 = d

    (b) 1 = q1, 2 = d, 3 = q2

    (c) 1 = d, 2 = q1, 3 = q2

    (d) 1 = d, 2 = q2, 3 = q1





13. Najmanj koliko stanj potrebujemo v diagramu, ki bi na izhodu generiral zaporedje: 010 010 ...

    (a) Eno stanje

    (b) Dve stanji, eno za log. 0 in eno za log. 1

    (c) Tri stanja

    (d) Neskončno stanj, saj se zaporedje nikoli ne zaključi

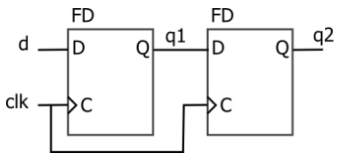
14. Poglej vezje in časovni diagram ter ugotovi katerim signalom pripadajo oznake 1., 2. in 3.?

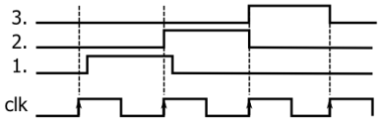
    (a) 1 = q1, 2 = q2, 3 = d

    (b) 1 = q1, 2 = d, 3 = q2

    (c) 1 = d, 2 = q1, 3 = q2

    (d) 1 = d, 2 = q2, 3 = q1





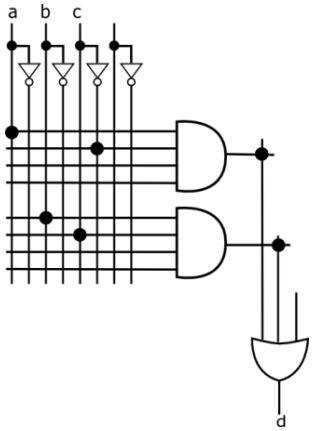
15. Ugotovi logično funkcijo, ki jo določa matrika PLA.

    (a) $c = (a$ AND $b)$ OR $(c$ AND NOT$(c))$

    (b) $c = (a$ AND $c)$ OR $(b$ AND $c)$

    (c) $c = (a$ AND NOT$(c))$ OR $(b$ AND $c)$

    (d) $c = (a$ OR $c)$ AND $(b$ OR $c)$



16. Kateri logični gradnik **ne moremo** narediti s programirljivo matriko PLA?

    (a) dekodirnik

    (b) izbiralnik

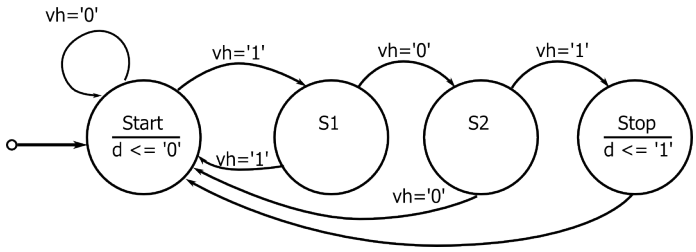
    (c) seštevalnik

    (d) register

17. S kakšno vezavo naredimo večbitni seštevalnik iz enobitnih gradnikov?

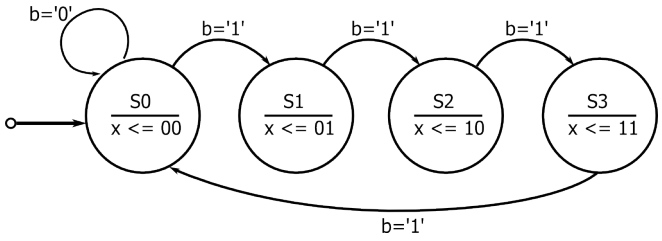
    (a) vzporedno
    (b) zaporedno
    (c) povratno
    (d) jih ne povežemo

18. Ugotovi, kakšno funkcijo opravlja vezje predstavljeno z diagramom stanj.



    (a) vezje ob $d = 0$ generira zaporedje 101 na izhodu
    (b) vezje ob $d = 0$ generira zaporedje 0101 na izhodu
    (c) vezje detektira zaporedje 101 na vhodu in postavi $d = 1$
    (d) vezje detektira zaporedje 11001 na vhodu in postavi $d = 1$

19. Ugotovi, kakšno funkcijo opravlja vezje predstavljeno z diagramom stanj.



    (a) vezje je 2-bitni števec, pri katerem je b signal za omogočanje štetja
    (b) vezje je 4-bitni števec, pri katerem je b signal za omogočanje štetja
    (c) vezje detektira zaporedje 00, 01, 10, 11 in postavi $b = 1$
    (d) vezje je 2-bitni števec, ki ga s signalom b postavimo na 00

20. Katera izmed mikrooperacij je narejena s povratno vezavo?

    (a) $r3 \leftarrow r2 - r1$

    (b) $r2 \leftarrow r2 + r1$

    (c) $r2 \leftarrow r1 \mid p1 = 1$

    (d) $r2 \leftarrow r1$ AND 0011

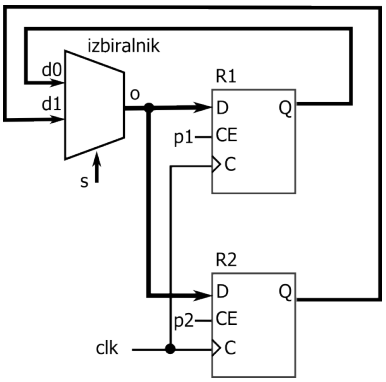
21. Kateri registrski prenos se izvede v primeru, ko je $s = 0, p1 = 0$ in $p2 = 1$?

    (a) $r1 \leftarrow r1$

    (b) $r1 \leftarrow r2$

    (c) $r2 \leftarrow r1$

    (d) $r2 \leftarrow r2$



22. Kateri gradnik se **ne nahaja** v podatkovni poti procesorja?

    (a) aritmetično-logična enota

    (b) registri

    (c) vodilo

    (d) programski števec

23. Kateri gradnik se **ne nahaja** v krmilnem delu centralno procesne enote?

    (a) ALE

    (b) pomnilnik

    (c) vodilo

    (d) dekodirnik

Literatura

1. F. Vahid, Digital design, John Wiley & Sons, 2006(2011)
2. M. M. Mano, Logic and Computer Design Fundamentals, Prentice Hall, 2007
3. K. Parnell, N. Mehta, Programmable Logic Design Quick Start Handbook, Xilinx, 2004
4. J. F. Wakerly, Digital Design Principles and Practices, Prentice Hall, 2000
5. M. Ercegovac, T. Lang, J. H. Moreno, Introduction do Digital Systems, John Wiley & Sons, 1999
6. A. Moore, FPGA for Dummies, Altera Special Eddition, John Wiley & Sons, 2014