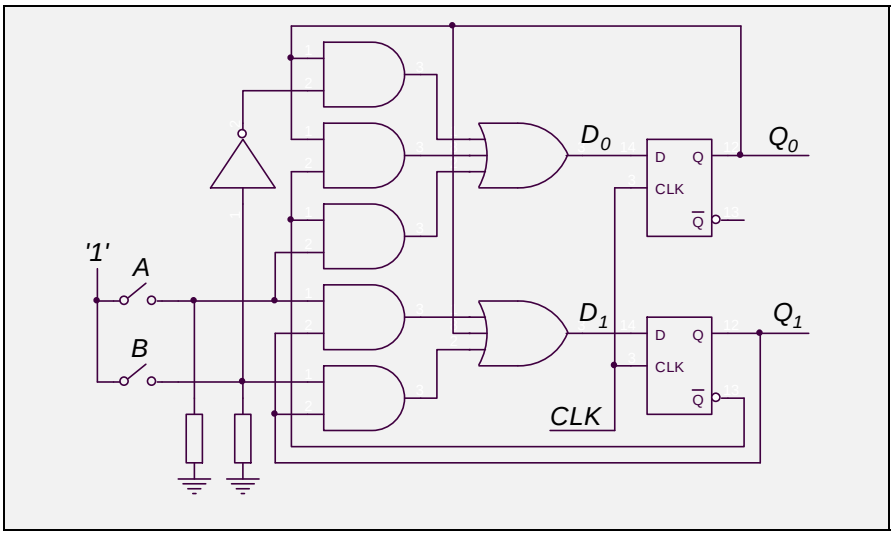
Dušan Ponikvar

# Digitalna elektronika za fizike

Maj 2009



# Kazalo

# Predgovor

Digitalna elektronska vezja uporabljamo, ker omogočajo veliko natančnost in hitrost računanja. Osnovni gradniki takih vezij so logična vrata, iz večjega števila logičnih vrat pa sestavimo kompleksna vezja, kot je na primer računalnik. V tem zapisu poskušamo utemeljiti osnove digitalne elektronike, kot jih potrebujejo študenti fizike za razumevanje delovanja in uporabo digitalnih vezij.

V prvem poglavju »Števila in signali v digitalni elektroniki« se seznanimo z zapisi števil v digitalni elektroniki in pretvarjanjem med različnimi zapisi ter predstavljanjem vrednosti števil v digitalnem vezju. Navedene so tipične napetosti in tokovi signalov v vezju, kakor tudi zgledi za prilagajanje zunanjih električnih signalov na velikosti, primerne za digitalna vezja.

V drugem poglavju »Kombinacijska logična vezja« govorimo o logičnih enačbah, poenostavljanju in implementacijah vezij z logičnimi vrati, kjer je odziv odvisen od trenutnih vrednosti vhodnih spremenljivk. Za zgled je vzeto vezje seštevalnika. Opisan je princip vodila in vezja za dostop do vodila.

V tretjem poglavju »Moduli in vodilo« opišemo nekaj značilnih digitalnih vezij, ki jih sestavimo iz logičnih vrat. Te module bomo uporabili v nadaljevanju zapisa pri števnikih, avtomatih in procesorju. Opisan je princip delovanja vodila v digitalnih sistemih ter podan zgled za prenašanje podatkov po vodilu.

V četrtem poglavju »Časovni potek signalov« se dotaknemo posledic spreminjanja vrednosti signalov. Zaradi končne hitrosti delovanja logičnih vezij je potrebno na pravilen Signale lahko zakasnimo tudi namenoma z logičnimi vrati ali RC členi ter tako formiramo različne oblike sunkov.

V petem poglavju »Spominske celice in njihova uporaba« se seznanimo z osnovnimi celicami in sestavi celic za pomnenje podatkov, kot so na primer register, pomični register ter spominska enota. V tem poglavju uvedemo diagram prehodov med stanji in tabele, ki podajajo enakovredno informacijo.

V šestem poglavju »Števniki in avtomati« uporabimo spominske celice za

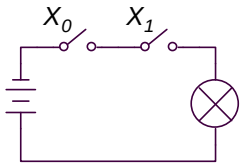
gradnjo števnikov asinhronega in sinhronega tipa in navedemo njihove lastnosti. S pomočjo števnika in kombinacijskega logičnega vezja generiramo poljubna zaporedja vrednosti signala. Idejo sinhronega števnika razdelamo v postopek za načrtovane poljubnega števnika, to pa še razširimo v idejo sinhronega avtomata ter generiranja poljubnih signalov, ki se spreminjajo po v naprej definiranem programu ter na podlagi vrednosti vhodnih spremenljivk.

V zadnjem poglavju »Zasnova procesorja« zarišemo pot za gradnjo univerzalnega digitalnega računskega vezja, ki je na podlagi programa uporabnika sposobno priti do iskanega rezultata. Tako vezje je srce vsakega sodobnega mikroprocesorja.

Avtor

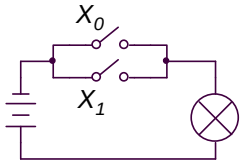
# Uvod

Vsi smo že poskusili povezati baterijo, žarnico in stikalo v električni tokokrog. Ko stikalo sklenemo, žarnica sveti. Kadar uporabimo več stikal, lahko s preklapljanjem vseh dosežemo, da žarnica sveti. Relacijo med svetenjem žarnice in potrebnimi preklopi stikal opišemo z logično funkcijo. Vzemimo na primer vezje s slike A. Žarnica sveti, ko je stikalo $X_0$ sklenjeno in stikalo $X_1$ sklenjeno. Vezje opravlja tako imenovano operacijo IN (»AND« v angleščini).



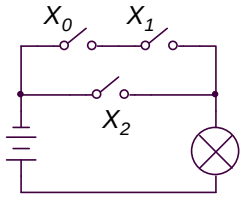

*Slika A: Funkcija IN*

Podobno vezje s slike B opravlja funkcijo ALI (»OR« v angleščini), saj žarnica sveti če je sklenjeno stikalo $X_1$ ali če je sklenjeno stikalo $X_0$. Idejo lahko razširimo na vezje s slike C, tu žarnica sveti takrat, ko je sklenjeno stikalo $X_1$ in hkrati sklenjeno stikalo $X_0$ ali pa je sklenjeno samo stikalo $X_2$.

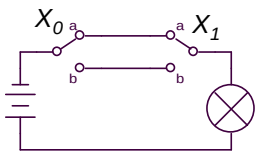



*Slika B: Funkcija ALI*

Igro nadaljujemo in vežemo stikala na primer po sliki D, kjer žarnico vklopimo na dveh različnih stikalih $X_1$ in $X_0$, pri katerih sta možna položaja stikal označena z 'a' in 'b'. Žarnica sveti takrat, ko je stikalo




*Slika C: Kombinirana stikala*




*Slika D: Z izmeničnimi stikali dosežemo več*

$X_0$ v položaju 'a' in stikalo $X_1$ v položaju 'a' ali pa takrat, ko stikalo $X_0$ ni v položaju 'a' in stikalo $X_1$ ni v položaju 'a', kar bo v nadaljevanju popisano s funkcijo ekskluzivni ALI (»XOR« v angleščini).

Stikala lahko postavljamo poljubno kompliciranо in dosežemo kompleksne logične funkcije, ki prižgejo žarnico. S stikali bi lahko tudi računali, če bi jih le uporabili dovolj in bi znali interpretirati svetenje žarnic.

Pri digitalni elektroniki bomo poskusili računati podobno, vendar z boljšimi sredstvi. Uvedli bomo osnovne gradnike logičnih vezij, se naučili pisati logične enačbe, jih poenostavljati in iz osnovnih gradnikov sestaviti ustrezno vezje, ki po enačbi računa. Iz takih vezij bomo zgradili sisteme, ki zmorejo mnoge kompleksne matematične operacije.

# Števila in signali v digitalni elektroniki

## 1. Zapisi števil v digitalni elektroniki

Ljudje smo vajeni zapisovati vrednosti spremenljivk s ciframi. Poznamo deset različnih cifer in z njimi lahko zapišemo deset različnih vrednosti. Če naj ima spremenljivka več različnih vrednosti, uporabimo več cifer drugo poleg druge, pri tem ima vsaka desno zapisana cifra desetkratno težo svoje leve sosede. Pravimo, da za zapis vrednosti števila uporabljamo decimalni sistem.

Vrednost spremenljivke v elektroniki predstavljamo s električnim signalom, na primer z napetostjo ali tokom. Signale med sklopi elektronskega vezja vodimo po žici. V digitalni elektroniki ima signal lahko le eno od dveh različnih vrednosti, ki jih običajno označimo z 0 (nič) in 1 (ena). V literaturi najdemo še oznake F (»false«) in T (»true«), LO (nizek nivo, »low«) in HI (visok nivo, »high«) in podobne. To ustreza matematičnemu zapisu vrednosti spremenljivke z eno cifro. Ker ima cifra lahko le eno od dveh možnih vrednosti, govorimo o binarnem sistemu.

Za računanje potrebujemo spremenljivke, ki imajo več različnih vrednosti. Take v binarnem sistemu zapišemo z več ciframi, od katerih ima vsaka cifra lahko le eno od dveh različnih vrednosti. Pri tem ima vsaka bolj levo zapisana cifra dvakrat tolikšno težo kot njena desna soseda. Posamezno cifro binarno zapisanega števila s tujko imenujemo bit.

Zgled: Zapišimo decimalno vrednost binarno zapisanega števila $1011_2$. Indeks ob številu označuje binarni ($_2$) ali decimalni ($_{10}$) sistem zapisa.

$$1011_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 8_{10} + 0_{10} + 2_{10} + 1_{10} = 11_{10}$$

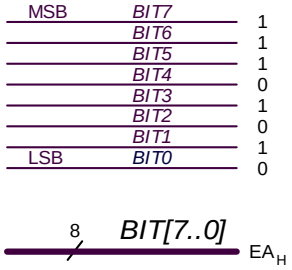
Zgled: Poiščimo binarni ekvivalent decimalnega števila $234_{10}$ s pomočjo zaporednih deljenjih z dva in ostankov, ki prepisani od spodaj navzgor predstavljajo binarno vrednost decimalnega števila.

| | | | | | | |
|---|---|---|---|---|---|---|
| 234 | ÷ 2 = | 117 | + | 0 | | $234_{10} = 11101010_2$ |
| 117 | ÷ 2 = | 58 | + | 1 | | |
| 58 | ÷ 2 = | 29 | + | 0 | | |
| 29 | ÷ 2 = | 14 | + | 1 | | |
| 14 | ÷ 2 = | 7 | + | 0 | | |
| 7 | ÷ 2 = | 3 | + | 1 | | |
| 3 | ÷ 2 = | 1 | + | 1 | | |
| 1 | ÷ 2 = | 0 | + | 1 | | |

Z M biti lahko zapišemo $2^M$ različnih vrednosti, ki se zvrstijo od 0 do $2^M$-1, če zaznamujemo le pozitivna cela števila. Ker je binarni zapis večjih števil zaradi velikega števila bitov za človeka nepregleden, uporabljamo heksadecimalni zapis. Pri tem zapisu ima cifra lahko eno od šestnajstih različnih vrednosti, ki jih zaznamujemo s ciframi od 0 do 9 ter črkami od A za 10 do F za 15. Iz binarnega zapisa prevedemo število v heksadecimalni zapis tako, da združimo po štiri bite binarnega zapisa v eno cifro heksadecimalnega zapisa. Združevati začnemo z desne. V obratno smer pretvarjamo tako, da namesto posamezne heksadecimalne cifre zapišemo grupo štirih binarnih cifer. Indeks $_H$ označuje heksadecimalni zapis.

Zgled: $11101010_2 = EA_H$

V digitalni elektroniki za predstavitev spremenljivke, ki jo zapišemo z več biti, potrebujemo več žic, signal na vsaki žici pa predstavlja en bit. Na zgornji polovici slike 1.1 je prikazan niz žic, ki vodijo posamezne bite spremenljivke zapisane s skupaj osmimi biti. Žica, ki prenaša najmanj pomemben bit, je označena z *BIT0* oziroma LSB (»least significant bit«), žica za najbolj pomemben bit pa z *BIT7* oziroma MSB (»most significant bit«). Prenašana vrednost znaša $234_{10}$ oziroma $11101010_2$. Tak skupek žic imenujemo vodilo (s tujko »bus«).




*Slika 1.1: Vodilo, prenašana vrednost znaša $234_{10}$*

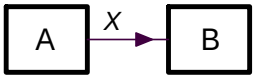
Kadar je vodilo sestavljeno iz velikega števila žic, postane slika nepregledna, zato namesto posameznih žic raje narišemo eno debelo črto ter ob njej navedemo število žic, ki jih črta predstavlja, nad njo pa imena žic v vodilu. Zgled je na sliki 1.1 spodaj. Poleg je napisana vrednost signalov na žici v heksadecimalni obliki.

Digitalni signali imajo lahko še druge vrednosti, ki jih uvedemo zato,

da poenostavimo delovanje digitalnih elektronskih vezij in ne spreminjajo rezultata računanja. Te dodatne vrednosti so X (karkoli, vrednost signala ni pomembna, ker ne vpliva na rezultat) in Z (signal trenutno ni aktiven in zato je žica, ki sicer vodi ta signal prosta in na razpolago za druge signale). Te dodatne vrednosti bomo podrobneje obravnavali kasneje.

## 2. Tehnologija in nivoji napetosti

Tehnologija izdelave osnovnih gradnikov digitalne elektronike stalno napreduje in omogoča vedno hitrejša vezja z manjšo porabo električne energije. Govorimo o generacijah integriranih vezij in družinah logičnih vrat, za vsako je med drugimi lastnostmi značilna tudi napetost, ki označuje logični vrednosti nič in ena. V času pisanja te knjige se pogosto uporabljajo vezja, izdelana v tehnologiji HCMOS (»High speed Complementary Metal Oxide Semiconductor«, integrirana vezja imajo oznako 74HCxx, pri tem xx zaznamuje funkcijo vezja). Nominalna napajalna napetost za HCMOS vezja znaša +5V, delujejo pa od +2V do +6V.

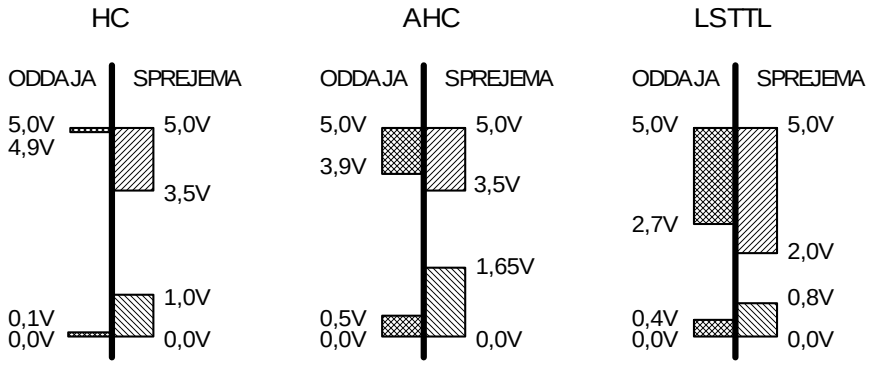



*Slika 2.1: Enoti A in B povezuje signal X*

Na sliki 2.1 signal *X* povezuje logični enoti A in B, obe sta izdelani v isti tehnologiji HCMOS. Zahtevamo, da enota B pravilno interpretira vrednost signala *X* kljub tolerancam pri izdelavi obeh enot ter motnjam, ki lahko kvarijo signal. Zato je predpisano, da mora enota A v najbolj neprimernih pogojih delovanja oddajati vrednost nič z napetostjo do +0,1V, enota B pa mora vhodno napetost, ki je manjša od +1,0V razumeti za vrednost nič. Potem velja, da lahko motnje dosežejo celo vrednost +0,9V, pa bo enota B kljub temu pravilno interpretirala ničlo z enote A.

Sorodno je predpisano, da mora enota A oddajati vrednost ena z napetostjo več kot +4,9V ter enota B vhodno napetost, ki je večja od +3,5V, razumeti za logično ena. Tokrat so lahko motnje velike do -1,4V, pa bo vrednost ena še vedno pravilno interpretirana v enoti B. Tak predpis je za družino HCMOS, ki jo napajamo z napetostjo +5V, prikazan na sliki 2.2. Za primerjavo je na isti sliki prikazanih še nekaj predpisov za druge družine (novejša AHC in starejša LSTTL). Področje napetosti med +1,0V in +3,5V se ne uporablja in je prepovedano. Če v vezju namerimo to napetost, je zagotovo nekaj narobe.

Ista vezja HCMOS lahko napajamo tudi z manjšo napetostjo, na primer 3,3V. To storimo, kadar vezja neposredno navezujemo na enote, ki uporabljajo take napajalne napetosti. Pri tem moramo biti pozorni na pravilne nivoje za ničle in enke, ki se prilagodijo novemu napajanju.




*Slika 2.2: Logični nivoji za različne družine logičnih vezij*

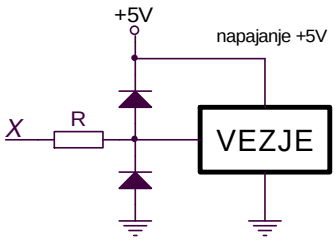
Signali, ki jih dajejo digitalna vezja, so šibki signali. To pomeni, da z izhodnim signalom vezja ne moremo, na primer, poganjati žarnice. Vezje ne daje dovolj energije, da bi žarnica svetila. Dovoljeni izhodni tok je majhen in ga ne smemo preseči s priključevanjem premajhnega bremena, sicer vezje uničimo. Tipični dovoljeni izhodni tok znaša do nekaj mA!

Logična vezja pravilno delujejo le, če so vhodni signali pravilne velikosti. V splošnem velja, da se lahko vhodni signal giblje med napajalnima napetostima vezja. Vhodni signal, ki napajalne napetosti presega, trajno poškoduje vezje.

# 3. Prilagoditev signala na vhodu v logično vezje

Pri priključevanju signalov s fizikalnih eksperimentov na digitalna elektronska vezja ali pri mešanju digitalnih vezij različnih družin in napajalnih napetosti se lahko zgodi, da napetosti za logično nič in ena ne ustrezata. Takrat lahko logična vezja napačno interpretirajo signale ali pa se celo pokvarijo.

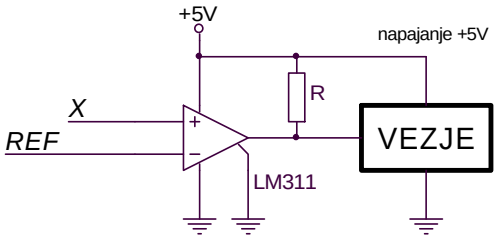
Z vezjem na sliki 3.1 omejimo vhodni signal *X* na vrednosti, ki zanesljivo ne uničijo logičnih vezij, napajanih s +5V. Potrebujemo dve diodi in upornik R. Posamezna dioda prevaja, kadar je signal *X* bodisi bolj pozitiven od pozitivne napajalne napetosti +5V logičnega vezja ali kadar je vhodni signal bolj negativen od negativne napajalne napetosti logičnega vezja (zemlja, »GND«). Upornik R izberemo glede na velikost vhodne napetosti in tehnologijo logičnih vezij tako, da je tok skozenj pri najbolj napačni




*Slika 3.1: Zaščita vezja pred prevelikimi vhodnimi napetostmi*

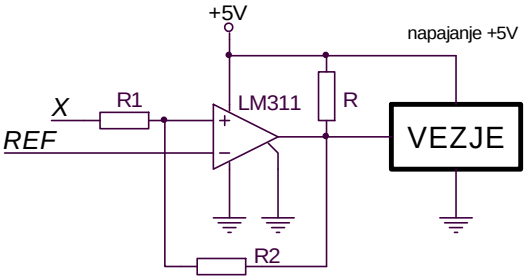
vhodni napetosti do nekaj mA. Tipična vrednost upornika R znaša nekaj kΩ.

Če je vhodni signal *X* premajhen, ga moramo najprej dovolj povečati, da logično vezje pravilno interpretira signal. Uporabimo lahko komparator po sliki 3.2, pri tem referenčno napetost *REF* izberemo tako, da komparator zanesljivo loči med nič in ena vhodnega signala *X*. Če na primer pričakujemo, da napetost 100mV zaznamuje logično nič signala *X*, napetost 200mV pa njegovo logično ena, izberemo *REF* = 150mV.



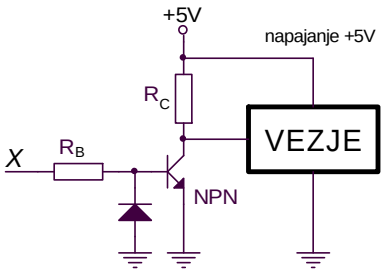

*Slika 3.2: Pravilna nivoja signala zagotovi komparator*

Kadar je vhodni signal zašumljen, uporabimo komparator s histerezo po sliki 3.3, širino histerezne zanke definira razmerje upornikov R1 in R2, ki ga prilagodimo šumu. Vrednost upornika R mora veliko biti manjša od R2, če naj bodo logični nivoji pravilni. V obeh primerih sme biti vhodni signal le od 0V do +5V, sicer lahko poškoduje komparator. Če je vhodni signal izven teh meja, pred komparator vežemo še upornik in diodi s slike 3.1.




*Slika 3.3: Komparator s histerezo*

Enostavneje, a manj zanesljivo, vhodni signal prilagodimo in vezje zaščitimo pred preveliko vhodno napetostjo s tranzistorjem po sliki 3.4, kjer upornik $R_B$ izberemo glede na velikost vhodnega signala *X* tako, da tok skozenj ne obremenjuje preveč vira signala, upornik $R_C$ pa ima vrednost nekaj kΩ. Dioda prepreči poškodbe tranzistorja takrat, ko je vhodna napetost *X* negativna.




*Slika 3.4: S tranzistorjem lahko prilagodimo signal*

# 4. Moč in logična vezja

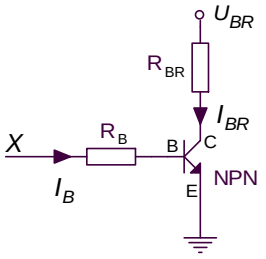
Kadar želimo z logičnim vezjem krmiliti breme s tokom ali napetostjo, ki sta večja od dovoljene vrednosti za logično vezje, moramo izhodni signal vezja najprej ojačiti. V ta namen uporabljamo tranzistorje in releje.

**Tranzistor**

Tranzistor je ojačevalnik toka. Skozi kolektorski priključek C teče tok $I_{BR}$, ki je največ za za mnogokratnik $\beta$ večji od toka $I_B$ v priključek baze B.

$$I_{BR} = \beta \cdot I_B$$

Tipična vrednost faktorja $\beta$ je za male tranzistorje 100, za tranzistorje večjih moči pa 10. Tranzistor uporabimo po sliki 4.1, Kadar je vrednost $X$ enaka logični nič, tok v bazo tranzistorja ne teče in zato tudi ni toka skozi breme $R_{BR}$. Kadar je vrednost $X$ enaka logični ena, teče v bazo tranzistorja tok $I_B$, zato teče do β-krat večji tok tudi skozi breme $R_{BR}$. Tok $I_B$ v bazo je določen z napetostjo $X$ in upornikom $R_B$ po formuli:




*Slika 4.1: Tranzistor kot ojačevalnik toka in napetosti*

$$I_B = \frac{X - 0{,}7}{R_B}$$

Pri tem smo upoštevali padec napetosti med priključkoma za bazo in emitor, ki za silicijev tranzistor znaša 0,7V. Tok $I_B$ ne sme preseči največjega dovoljenega izhodnega toka logičnega vezja, tranzistor pa mora imeti dovolj veliko ojačenje, da bo skozi kolektorski priključek tranzistorja takrat tekel tok $I_{BR}$, ki znaša:

$$I_{BR} = \frac{U_{BR}}{R_{BR}}$$

Zgled: Uporabljamo logična vrata v tehnologiji HCMOS, napajana s 5V, breme pa je priključeno na $U_{BR}$ = +12V ter ima upornost $R_{BR}$ = 10Ω. Kakšne so vrednosti upornika $R_B$ in minimalno ojačenje tranzistorja $\beta$?

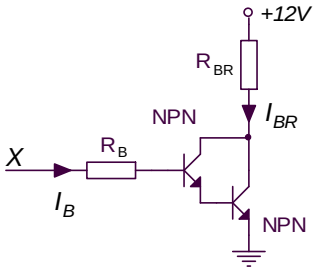
Največji dovoljeni izhodni tok za HCMOS tehnologijo znaša 4mA, zato izberemo upornik $R_B$ iz formule:

$$R_B = \frac{X - 0{,}7}{I_B} = \frac{4{,}9V - 0{,}7V}{4mA} = 1050\ \Omega \approx 1k\Omega$$

Skozi breme $R_{BR}$ lahko teče pri napajanju +12V do 1,2A toka. Tokovno ojačenje $\beta$ tranzistorja mora zato znašati vsaj:

$$\beta = \frac{I_{BR}}{I_B} = \frac{1{,}2A}{0{,}004A} = 300$$

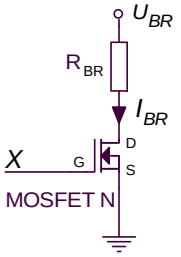
Tako velikega ojačenja en tranzistor, ki je delan za tokove nad 1A, ne zmore. Uporabimo pa lahko dva tranzistorja v takoimenovani »Darlington« vezavi po sliki 4.2. Ojačenje para je enako produktu ojačenj posameznega tranzistorja. Vrednost upornika $R_B$ prilagodimo padcu napetosti med bazo in emitorjem para tranzistorjev, ki znaša približno 2 x 0,7V = 1,4V, na vrednost 875Ω oziroma najbližjo vrednost iz lestvice standardnih vrednosti upornikov, ta znaša 820Ω. Seveda mora biti desni tranzistor tudi dovolj močan, da zmore prevajati tok vsaj 1,2A in prenesti napetost vsaj 12V.




*Slika 4.2: Darlington vezava tranzistorjev za večje ojačenje toka*

**MOSFET (»Metal Oxyde Semiconductor, Field Effect Transistor«)**

Podobno priključimo tranzistor MOSFET po sliki 4.3. Tokrat signal *X* z logičnega vezja povežemo direktno na kontrolni priključek G (»gate«) tranzistorja, saj je tok v priključek G enak nič. MOSFET prevaja tok med elektrodama D (»drain«) in S (»source«) takrat, kadar je napetost med elektrodama G in S pozitivna in dovolj velika, tipično nekaj voltov.




*Slika 4.3: Povezava na breme s pomočjo MOSFET-a tranzistorja*

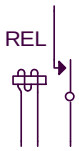
Tranzistor izberemo tako, da prenese napetost bremena $U_{BR}$ ter tok bremena $I_{BR}$.

Poleg tega moramo biti pozorni še na to, da ga izhodna napetost $X$ logičnega vezja naredi popolnoma prevodnega. Ta lastnost se skriva v vrednosti $U_{GS}$, ki je podana v specifikacijah tranzistorja in mora biti manjša od minimalne izhodne napetosti logičnega vezja, ki predstavlja logično ena.

Tranzistor MOSFET nudi le majhno upornost toku $I_{BR}$ takrat, ko tranzistor prevaja. Tipična vrednost upornosti med elektrodama D in S je mnogo manjša od 1Ω in zato se MOSFET med prevajanjem toka le malo greje.

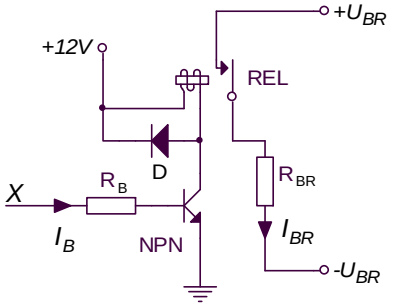
**Rele**

Rele sestavlja tuljava in niz kontaktov v njenem magnetnem polju, električni simbol za rele je na sliki 4.4. Kadar skozi tuljavo teče tok, se zaradi magnetnega polja kontakti sklenejo (pri nekaterih modelih pa razklenejo, odvisno od fizične razporeditve kontaktov v releju), to pa povzroči priklop (ali odklop) bremena.




*Slika 4.4: Simbol za rele*

Tuljava releja je zgrajena za izbrano napetost. Ko tuljavo priključimo na tako napetost, se kontakta skleneta. Potrebna napetost je navadno večja od tiste, ki jo dobimo iz logičnega vezja, poleg tega logično vezje ni dovolj močno in ne more samo pognati potrebnega toka skozi tuljavo releja, da bi ta sklenil kontakta. Zato uporabimo shemo s slike 4.5. Tranzistor NPN najprej dovolj ojači signal z logičnih vrat za tuljavo releja REL, ki je grajena za 12V in 50mA, rele pa potem priklopi breme na napetost med priključkoma $+U_{BR}$ in $-U_{BR}$. Upornik $R_B$ ima vrednost nekaj kΩ, določimo jo po prej navedeni formuli. Vezava je posebej primerna zato, ker je breme galvansko ločeno od logičnega vezja in z njim lahko na primer priklapljamo tudi bremena na omrežno napetost, ne da bi bilo logično vezje povezano z za dotik človeka nevarnim omrežjem. Dioda D prepreči poškodbe tranzistorja ob izklopu releja zaradi v tuljavi nakopičene energije.




*Slika 4.5: Priključitev bremena $R_{BR}$ preko releja*

# Kombinacijska logična vezja

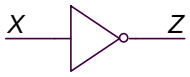
## 5. Osnovni gradniki digitalnih elektronskih vezij

Osnovni gradniki digitalnih elektronskih vezij so logična vrata. Ta izvajajo logične operacije, ki podajajo relacije med vhodi in izhodi logičnih vrat. Poznamo tri osnovne logične operacije in torej tri osnovne tipe logičnih vrat.

a) Logična vrata NOT, negator ali invertor opravljajo logično inverzijo. Negator ima en priključek za vhodni in en za izhodni signal, pri tem je vrednost signala na izhodu obratna vrednosti vhodnega signala. Na sliki 5.1 je narisan simbol za negator, poleg je tabela, ki popisuje vse možne vrednosti vhodnega signala *X* in ustrezajoče vrednosti izhodnega signala *Z*. Ker ima negator le en vhodni signal, ima tabela le dve vrstici.

   Logično funkcijo inverzije zapišemo v enačbi: $Z = \bar{X}$

| *X* | *Z* |
|---|---|
| 0 | 1 |
| 1 | 0 |

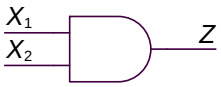



*Slika 5.1: Negacija*

b) AND: logična vrata AND (IN) opravljajo logično operacijo, ki jo matematiki imenujejo konjunkcija. Vrata AND imajo dva ali več priključkov za vhodne signale in en priključek za izhodni signal, pri tem je vrednost izhodnega signala ena samo takrat, ko imajo vsi vhodni signali vrednost ena. Na sliki 5.2 je narisan elektronski simbol za vrata AND z dvema vhodnima priključkoma, poleg je tabela, ki popisuje vse možne kombinacije vrednosti vhodnih signalov in ustrezajoče vrednosti izhodnih signalov. Ker so na sliki vrata AND z dvema vhodnima priključkoma, so možne štiri različne kombinacije vrednosti vhodnih signalov, zato ima tabela štiri vrstice. Vhodnih priključkov je lahko tudi več.

   Logično funkcijo konjunkcije zapišemo v enačbi: $Z = X_1 \cdot X_2 = X_1 X_2$.

| $X_1$ | $X_2$ | $Z$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

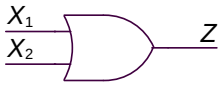



*Slika 5.2: Konjunkcija ali funkcija AND*

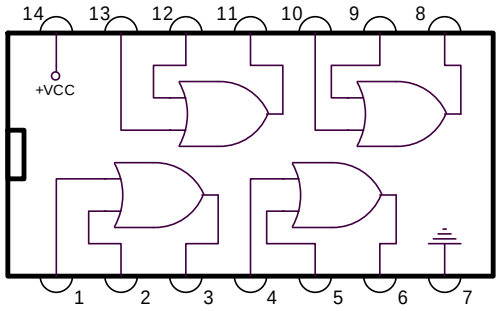
c) OR: logična vrata OR (ALI) opravljajo logično operacijo, ki jo matematiki imenujejo disjunkcija. Vrata OR imajo dva ali več priključkov za vhodne signale in en priključek za izhodni signal, pri tem ima izhodni signal vrednost ena, ko ima katerikoli od vhodnih signalov vrednost ena. Na sliki 5.3 je narisan elektronski simbol za vrata OR z dvema vhodnima priključkoma, poleg je tabela, ki popisuje vse možne kombinacije vrednosti vhodnih signalov in ustrezajoče vrednosti izhodnega signala. Ker so na sliki vrata OR z dvema vhodnima priključkoma, so možne štiri različne kombinacije vrednosti vhodnih signalov, zato ima tabela štiri vrstice. Vrata OR imajo lahko tudi več vhodnih priključkov.

   Logično funkcijo disjunkcije zapišemo v enačbi: $Z = X_1 + X_2$.

| $X_1$ | $X_2$ | $Z$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |




*Slika 5.3: Disjunkcija ali funkcija OR*




*Slika 5.4: Razpored priključkov logičnih vrat v integriranem vezju, pogled od zgoraj*

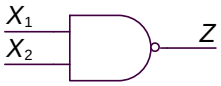
Vsa osnovna logična vrata (NOT, AND in OR) so na razpolago v obliki integriranih vezij. Navadno je v enem integriranem vezju z na primer štirinajst nožicami razporejenih več logičnih vrat iste vrste. Za primer naj služi integrirano vezje z oznako 74HC32, ki vsebuje štiri vrata OR s po dvema vhodnima priključkoma, razpored vrat in

priključkov je na sliki 5.4, pogled od zgoraj. Spodnji desni (nožica 7) in zgornji levi (nožica 14) priključek digitalnega integriranega vezja sta namenjena napajalni napetosti nič voltov ($V_{DD}$ oziroma GND) in +5V ($V_{SS}$ oziroma $V_{CC}$).

Osnovna logična vrata lahko med sabo povezujemo. Pogosto se izkaže potreba po hkratni konjunkciji in negaciji, zato so vrata NAND (NeIN), ki opravljajo obe logični operaciji hkrati, na razpolago v obliki integriranih vezij. Simbol za taka vrata in ustrezna tabela sta na sliki 5.5, enačba pa se glasi:

$$Z = \overline{X_1 \cdot X_2} = \overline{X_1 X_2}$$

| $X_1$ | $X_2$ | $Z$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

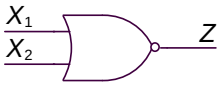



*Slika 5.5: Hkratna konjunkcija in negacija, vrata NAND*

Prav tako je pogosta potreba po hkratni disjunkciji in negaciji in tudi vrata NOR (NALI) so na razpolago v obliki integriranih vezij. Simbol za vrata NOR je na sliki 5.6, kjer je tudi ustrezna tabela, enačba pa se glasi:

$$Z = \overline{X_1 + X_2}$$

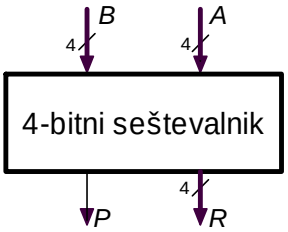
| $X_1$ | $X_2$ | $Z$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |




*Slika 5.6: Hkratna disjunkcija in negacija, vrata NOR*

# 6. Logične enačbe: seštevalnik

Ena osnovnih matematičnih operacij je seštevanje. Sestavimo digitalno vezje, ki zna sešteti dve štirimestni binarni števili. Pri tem se bomo naučili precej o logičnih vezjih in digitalni elektroniki.




*Slika 6.1: Simbol za štiri-bitni seštevalnik*

Na sliki 6.1 je narisan blok, ki ga potrebujemo in ustrezni signali. Argumenta seštevanja sta vhodna signala $A$ in $B$, ki sta narisana kot štiri-bitni vodili (biti so $A_3$ do $A_0$ ter $B_3$ do $B_0$), rezultat $R$ je prav tako štiri-bitno vodilo (biti so $R_3$ do $R_0$). Dodan je še signal $P$ za prenos; rezultat seštevanja dveh štiri-bitnih števil je lahko dolg tudi pet bitov.

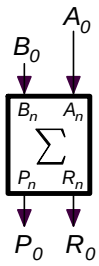
Račun zapišemo:

| | | | | |
|---|---|---|---|---|
| | $A_3$ | $A_2$ | $A_1$ | $A_0$ |
| + | $B_3$ | $B_2$ | $B_1$ | $B_0$ |
| $P$ | $R_3$ | $R_2$ | $R_1$ | $R_0$ |

Seštevati začnemo tako, da seštejemo bita $A_0$ in $B_0$ ter dobimo bit rezultata $R_0$ ter prenos do sosednjega desnega mesta, ki ga označimo s $P_0$. S seštevanjem nadaljujemo tako, da seštejemo prenos s prvega mesta $P_0$ in bita $A_1$ ter $B_1$, dobimo bit rezultata $R_1$ ter prenos do sosednjega levega bita $P_1$. Namesto hkratnega seštevanja treh bitov lahko tudi najprej seštejemo bita $A_1$ in $B_1$ ter nato k temu delnemu rezultatu prištejemo še prenos $P_0$ prejšnjega seštevanja na desnem bitu. Postopek nadaljujemo še za ostale bite proti levi, dokler ni račun končan.

# 7. Delni seštevalnik

Za seštevanje v digitalni elektroniki torej potrebujemo osnovno enoto, ki je sposobna sešteti dva bita $A_n$ in $B_n$ v rezultat $R_n$ ter prenos $P_n$. Simbol za tako enoto je na sliki 7.1. Na vhode enote na primer priključimo bita $A_0$ in $B_0$ zgoraj navedenega računa ter dobimo rezultat $R_0$ ter prenos $P_0$.




*Slika 7.1: Modul seštevalnika*

Pri seštevanju dveh bitov $A_n$ in $B_n$ lahko pride do štirih različnih kombinacij. To pokažemo s štirimi ločenimi računi po sliki 7.2.

Te kombinacije pregledneje prikažemo v tabeli s štirimi stolpci po sliki 7.3. Pri tem so v levih dveh stolpcih po vrsticah zapisane vse možne kombinacije vhodnih bitov $A_n$ in $B_n$, v desnih dveh stolpcih pa ustrezajoče vrednosti izhodnih bitov $R_n$ in $P_n$, ki vezje naredijo za

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | | | 0 | | | 1 | | | 1 | |
| + | 0 | | + | 1 | | + | 0 | | + | 1 | |
| 0 | 0 | | 0 | 1 | | 0 | 1 | | 1 | 0 | |

*Slika 7.2: Možne kombinacije vrednosti argumentov seštevanja in rezultati*

seštevalnik. Vezje ima torej dva izhoda za $P_n$ in $R_n$, zato ga sestavimo iz dveh delov. Pri tem si pomagamo z osnovnimi gradniki, ki so na razpolago v digitalni elektroniki.

Že z enostavnim primerjanjem tabel lahko ugotovimo, da sta tabeli za logično funkcija AND iz poglavja o osnovnih gradnikih logičnih vezij in prenos $P_n$ identični. Do izhodnega signala $P_n$ seštevalnika torej pridemo, če vhodna signala $A_n$ in $B_n$ navežemo na vhodna priključka vrat AND. Do identične ugotovitve pridemo tudi s sklepanjem. Prenos $P_n$ ima vrednost ena takrat, ko ima bit $A_n$ vrednost ena in bit $B_n$ vrednost ena, kar ustreza enačbi $P_n = A_n \cdot B_n$ ter definiciji logične funkcije AND.

| $A_n$ | $B_n$ | $P_n$ | $R_n$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

*Slika 7.3: Tabela za seštevanje*

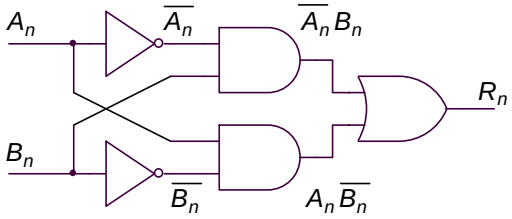
Z rezultatom $R_n$ je nekaj več dela. Ta ima vrednost ena v dveh primerih:

- ko ima bit $A_n$ vrednost nič in bit $B_n$ vrednost ena ali

- ko ima bit $A_n$ vrednost ena in bit $B_n$ vrednost nič.

Z besedami povedano takoj prevedemo v logično enačbo:

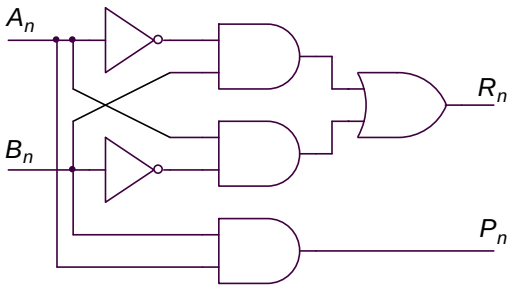
$$R_n = \overline{A_n} \cdot B_n + A_n \cdot \overline{B_n}$$




*Slika 7.4: S takim vezjem izračunamo vsoto $R_n$*

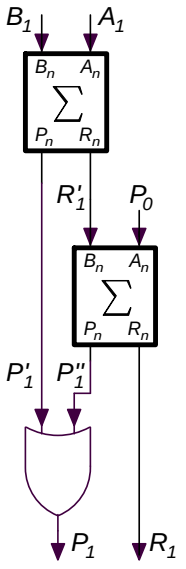
Pri tem je treba poudariti, da ima logična funkcija AND prednost pred funkcijo OR enako, kot ima v matematiki množenje prednost pred seštevanjem. Najvišjo prioriteto ima negacija, prav tako kot potenciranje v

matematiki. Prav tako, kot v matematiki, bi lahko tudi v logičnih enačbah s pomočjo oklepajev spremenili vrstni red opravljanja logičnih operacij, to bomo izkusili v naslednjih poglavjih. Shema digitalnega vezja za računanje rezultata $R_n$ po zgornji formuli je na sliki 7.4, kjer je lepo razvidna tudi prioriteta logičnih funkcij. Na sliki 7.5 je še kompletna shema seštevalnika dveh bitov, ki mu v digitalni elektroniki rečemo delni seštevalnik.




*Slika 7.5: Kompletno vezje delnega seštevalnika*

# 8. Raba delnega seštevalnika




*Slika 8.1: Dva delna seštevalnika*

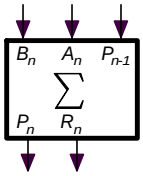
Delni seštevalnik uporabimo za računanje bita $R_0$ rezultata. Računanje ostalih bitov rezultata je bolj kompliciranо. Sešteti moramo namreč tri bite, z delnim seštevalnikom pa lahko seštejemo le dva. Zato nalogo razčlenimo na dva delna seštevalnika. V prvem seštejemo bita $A_1$ in $B_1$ ter dobimo vmesni rezultat $R_1'$ ter $P_1'$, k vmesnemu rezultatu $R_1'$ prištejemo še bit prenosa desnega seštevanja $P_0$ po sliki 8.1.

Kadar sta vrednosti bitov $A_1$ in $B_1$ enaki ena, ima prenos $P_1'$ vrednost ena, vmesni rezultat $R_1'$ pa je nič. Ko k vmesnemu rezultatu prištejemo prenos $P_0$, lahko postane rezultat $R_1$ kvečjemu ena, prenos $P_1$ pa je enak $P_1'$.

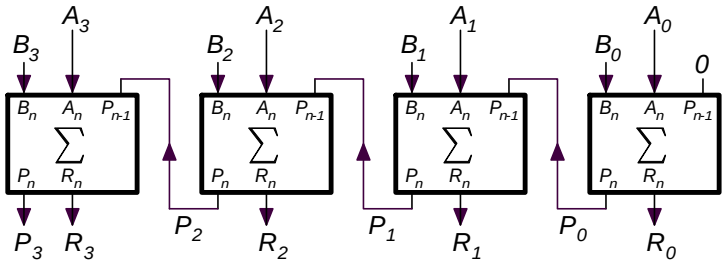
Kadar je katerikoli od bitov $A_1$ ali $B_1$ enak nič, je prenos $P_1'$ enak nič, vmesni rezultat $R_1'$ pa kvečjemu ena. Ko k vmesnemu rezultatu $R_1'$ prištejemo še prenos z desne $P_0$, lahko pride do prenosa $P_1''$. Zaradi tega mora biti tudi prenos $P_1$ iz

seštevalnika enak ena.

Skupni izhodni prenos $P_1$ je torej ena takrat, ko pride do prenosa $P_1'$ ali ko pride do prenosa $P_1''$, do obeh prenosov hkrati ne more priti. Zato povežemo oba vmesna prenosa z vrati OR v skupni izhodni prenos $P_1$ po sliki 8.1. Takemu modulu rečemo polni seštevalnik in sešteje tri bite v rezultat in prenos. Simbol za polni seštevalnik je na sliki 8.2, štiri polne seštevalnike pa sestavimo v štiri-bitni seštevalnik po sliki 8.3.




*Slika 8.2: Simbol za polni seštevalnik*




*Slika 8.2: Štirje polni seštevalniki sestavljajo štiri-bitni seštevalnik*

## 9. Polni seštevalnik

Izdelave polnega seštevalnika, ki lahko naenkrat sešteje tri bite, se lotimo še enkrat od začetka in se poučimo še o poenostavljanju logičnih funkcij. Tudi tokrat zapišemo izraz za operacijo, ki jo želimo opraviti:

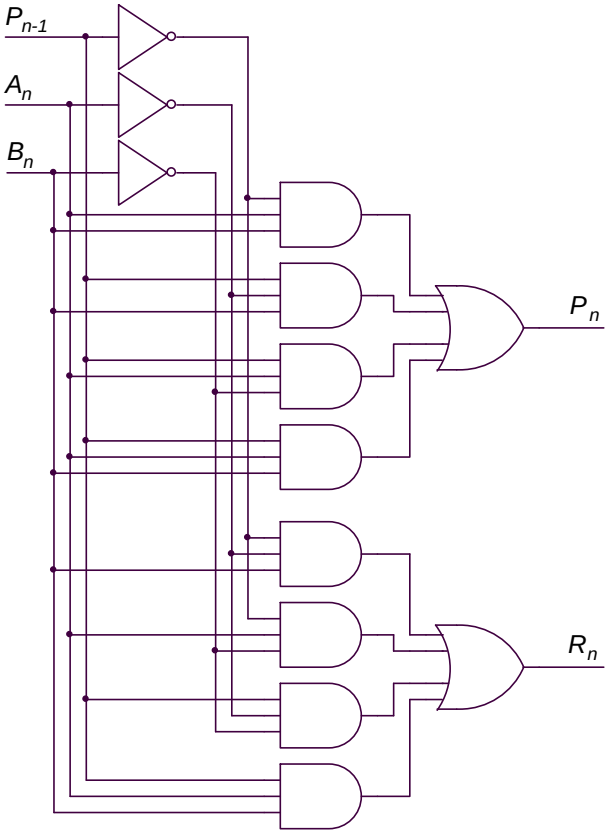
$$\begin{array}{cc} & A_n \\ + & B_n \\ + & P_{n-1} \\ \hline P_n & R_n \end{array}$$

Ta izraz prevedemo v tabelo, ki ima pet stolpcev. Trije stolpci na levi predstavljajo vhodne bite $A_n$, $B_n$ in $P_{n-1}$, desna stolpca pa prenos $P_n$ in rezultat $R_n$. Ker imamo opraviti s tremi vhodnimi biti, je možnih osem različnih kombinacij njihovih vrednosti; tabela ima torej osem vrstic. V desni del tabele so že vpisane prave vrednosti izhodnih bitov za funkcijo polnega seštevalnika.

| $P_{n-1}$ | $A_n$ | $B_n$ | $P_n$ | $R_n$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

Zgornja polovica tabele za polni seštevalnik je enaka, kot smo jo napisali v poglavju o delnem seštevalniku, saj seštevamo samo dva bita $A_n$ in $B_n$, bit $P_{n-1}$ ima v vsaki vrstici vrednost nič. Preostanek tabele s $P_{n-1}$ je ena je nov.




*Slika 9.1: Vezje za polni seštevalnik*

Iz tabele najprej z besedami izrazimo tiste vrstice, kjer ima prenos $P_n$ vrednost ena, to se zgodi v štirih primerih:

- ko ima bit $P_{n-1}$ vrednost nič in bit $A_n$ vrednost ena in bit $B_n$ vrednost ena ali
- ko ima bit $P_{n-1}$ vrednost ena in bit $A_n$ vrednost nič in bit $B_n$ vrednost ena ali
- ko ima bit $P_{n-1}$ vrednost ena in bit $A_n$ vrednost ena in bit $B_n$ vrednost nič ali
- ko ima bit $P_{n-1}$ vrednost ena in bit $A_n$ vrednost ena in bit $B_n$ vrednost ena.

Z enačbo to zapišemo:

$$P_n = \overline{P_{n-1}} \cdot A_n \cdot B_n + P_{n-1} \cdot \overline{A_n} \cdot B_n + P_{n-1} \cdot A_n \cdot \overline{B_n} + P_{n-1} \cdot A_n \cdot B_n$$

Na soroden način lahko najprej z besedami, nato pa z enačbo zapišemo izraz na rezultat $R_n$:

$$R_n = \overline{P_{n-1}} \cdot \overline{A_n} \cdot B_n + \overline{P_{n-1}} \cdot A_n \cdot \overline{B_n} + P_{n-1} \cdot \overline{A_n} \cdot \overline{B_n} + P_{n-1} \cdot A_n \cdot B_n$$

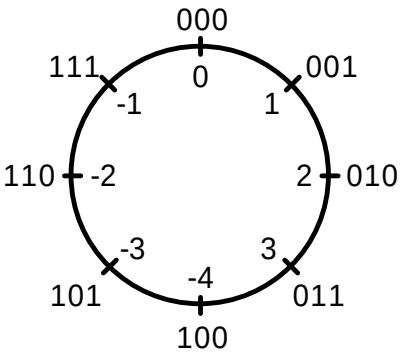
Na podlagi izpisanih enačb narišemo elektronsko shemo vezja polnega seštevalnika na sliki 9.1, ki je sestavljen iz dveh delov za rezultat $R_n$ in prenos $P_n$. Vsak kos je sestavljen iz enih vrat OR s štirimi vhodnimi priključki, štirih vrat AND s po tremi vhodnimi priključki ter negatorjev.

Štiri take polne seštevalnike združimo v štiri-bitni seštevalnik po shemi 8.2.

# 10. Nabor števil in negativna števila

Iz matematike smo vajeni neskončne množice števil, ki jih nazorneje prikažemo na številski premici. Ko se po številski premici premikamo v desno, vrednost števil narašča. Ko se premikamo v levo, se vrednost števil najprej zmanjšuje proti nič, nato pa narašča v negativnem smislu. Premik v desno ustreza seštevanju, premik v levo pa odštevanju.

Logična vezja, ki jih uporabljamo za opravljanje matematičnih operacij, so prirejena za delo s končnim številom bitov podobno, kot da bi števila v matematiki pisali z omejenim številom cifer. Zato množica števil ni neskončna, ampak je omejena. Tako omejen nabor števil lahko nanesemo na številsko daljico. Premik v desno po daljici spet ustreza seštevanju, premik v levo pa odštevanju. Težava se pojavi, ko dosežemo na primer desno krajišče




*Slika 10.1: Številski krog z naborom števil za tri bitni zapis*

daljice. Nadaljnje povečevanje števila povzroči preskok v levo krajišče, potem pa se zaradi povečevanja števila enakomerno premikanje v desno nadaljuje. S končnim številom bitov zapisana števila so ciklična, zato jih raje nanašamo na številski krog po sliki 10.1, kjer je zgled za s tremi biti zapisana števila. Binarni zapis je na zunanji strani kroga.

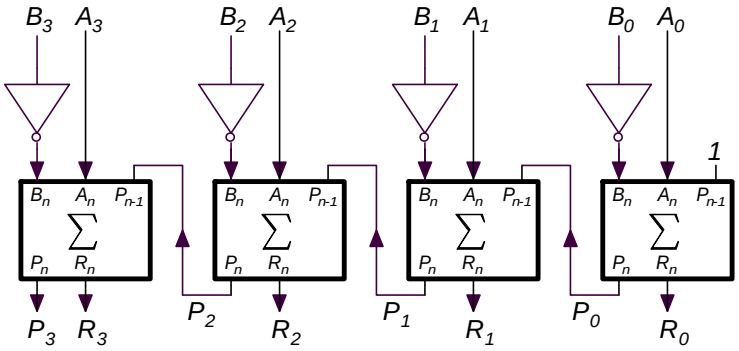
Za računanje navadno uporabljamo cela števila, torej naravna števila, število nič in negativna cela števila. V digitalni elektroniki se odločimo, da za negativna števila rezerviramo tiste kombinacije vrednosti bitov, ki se začno z ena na najbolj pomembnem mestu. Tako lahko z N biti zapišemo $2^N$ različnih vrednosti, ki obsegajo vrednosti med $-2^{N-1}$ in $2^{N-1}-1$. Tak nabor števil spet prikažemo na številskem krogu, na sliki 10.1 je na primer številski krog za število, zapisano s tremi biti. Ekvivalentne decimalne vrednosti so zapisane v notranjosti kroga.

S poskušanjem lahko pokažemo, da iz pozitivnega števila dobimo njegov negativni ekvivalent tako, da komplementiramo vrednosti vseh bitov v številu in rezultatu prištejemo ena. Obratni vrednosti števila pravimo komplement, ko zraven prištejemo ena pa dobimo dvojiški komplement. Poskusimo za število $2_{10} = 010_2$

| začetno število | invertiraj | komplement | prištej ena | dvojiški komplement |
|---|---|---|---|---|
| $2_{10} = 010_2$ | → | $101_2$ | → | $110_2 = -2_{10}$ |

# 11. Odštevanje

Ker znamo zapisati negativni ekvivalent danega števila, odštevanje ne dela več težav. Namesto, da gradimo novo vezje za odštevanje, lahko uporabimo seštevalnik in prištejemo negativni ekvivalent danega števila. Vezje, ki opravi odštevanje $R = A - B$, je na sliki 11.1. Od vezja za seštevanje s slike 8.2 se razlikuje v dveh podrobnostih: vsak od bitov števila *B* je komplementiran in v vhod za prenos najbolj desnega polnega seštevalnika je priključena enka. Tako pridemo do dvojiškega komplementa števila *B*, ki ga štiri bitni seštevalnik prišteje številu *A*.



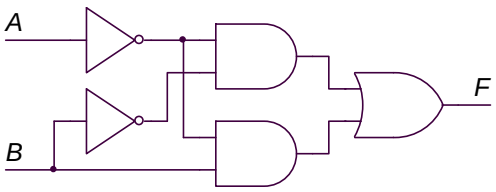

*Slika 11.1: Odštevanje $R = A - B$*

# 12. Poenostavljanje logičnih enačb

Za vezje polnega seštevalnika s slike 9.1 smo potrebovali veliko logičnih vrat. Ali je mogoče do enakega rezultata priti tudi z manj vrati? Za začetek si izmislimo logično funkcijo $F$ v tabeli in vezju na sliki 12.1 in iz nje izpišimo logično funkcijo. Ta se glasi:

$$F = \bar{A} \cdot \bar{B} + \bar{A} \cdot B$$

| $A$ | $B$ | $F$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

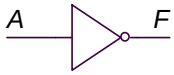



*Slika 12.1: Zgled logične funkcije*

Funkcija $F$ ima torej vrednost 1 takrat, ko ima bit $A$ vrednost nič in bit $B$ vrednost ena ali takrat, ko ima bit $A$ vrednost nič in bit $B$ vrednost nič. Sklepamo, da ima funkcija vrednost ena takrat, ko ima bit $A$ vrednost nič, od vrednosti bita $B$ pa vrednost funkcije $F$ ni odvisna. Funkcijo zato zapišemo enostavneje:

$$F = \bar{A}$$

Namesto enih vrat OR, dvojih vrat AND in dveh negatorjev uporabimo en sam negator, pa dobimo enak rezultat na sliki 12.2! Do takih skrajnih




*Slika 12.2: Poenostavljena verzija funkcije s slike 12.1*

poenostavitev pride le izjemoma, vedno pa lahko pričakujemo vezje, ki ima po poenostavljanju manj logičnih vrat od tiste verzije vezja, ki jo izpišemo in sestavimo naravnost iz tabele. Za poenostavljanje so na razpolago pravila (postulati) logične algebre, ki so navedena v nadaljevanju.

*Združljivost*: logične operacije enake vrste lahko opravljamo v poljubnem vrstnem redu.

$$A \cdot B \cdot C = (A \cdot B) \cdot C = A \cdot (B \cdot C)$$

$$A + B + C = (A + B) + C = A + (B + C)$$

*Zamenljivost*: rezultat logične operacije ni odvisen od vrstnega reda spremenljivk, vhode v logična vrata lahko med sabo poljubno zamenjamo.

$$A \cdot B = B \cdot A$$

$$A + B = B + A$$

*Izpostavljanje*: logično operacijo, ki se ponovi v več delih funkcije, lahko opravimo posebej, enkrat za vso funkcijo.

$$A \cdot B + A \cdot C = A \cdot (B + C)$$

$$(A + C) \cdot (B + C) = A \cdot B + C$$

*Pravila za funkcijo AND:* vsako od teh pravil dokažemo s tabelo.

$$A \cdot A = A$$

$$A \cdot 1 = A$$

$$A \cdot 0 = 0$$

$$A \cdot \bar{A} = 0$$

*Pravila za funkcijo OR:* vsako od teh pravil dokažemo s tabelo.

$$A + A = A$$

$$A + 1 = 1$$

$$A + 0 = A$$

$$A + \bar{A} = 1$$

*Pravila za funkcijo komplement*: pravilo dokažemo s tabelo.

$$\bar{\bar{A}} = A$$

*DeMorganov teorem*: spet ga dokažemo s tabelo, kar prepuščamo bralcu.

$$\overline{A + B} = \bar{A} \cdot \bar{B}$$

$$\overline{A \cdot B} = \bar{A} + \bar{B}$$

S pomočjo navedenih pravil lahko poenostavimo logični funkciji za polni seštevalnik. Najprej prenos:

$$P_n = \overline{P_{n-1}} \cdot A_n \cdot B_n + P_{n-1} \cdot \overline{A_n} \cdot B_n + P_{n-1} \cdot A_n \cdot \overline{B_n} + P_{n-1} \cdot A_n \cdot B_n$$

Najprej izpostavljanje:

$$P_n = (\overline{P_{n-1}} \cdot A_n + P_{n-1} \cdot \overline{A_n}) \cdot B_n + P_{n-1} \cdot A_n \cdot (\overline{B_n} + B_n)$$

Zadnji del izraza lahko poenostavimo z upoštevanjem pravil za funkcijo OR in nato še pravil za funkcijo AND v:

$$P_n = (\overline{P_{n-1}} \cdot A_n + P_{n-1} \cdot \overline{A_n}) \cdot B_n + P_{n-1} \cdot A_n$$

Če to funkcijo spremenimo v vezje, potrebujemo dvoja vrata OR s po dvema vhodoma, štiri vrata AND s po dvema vhodoma in dva negatorja. To se ne zdi posebej enostavneje od prejšnjega vezja za računanje prenosa s slike 9.1. Število vrat je enako, vendar imajo tokrat vsaka vrata manj vhodnih priključkov. To pomeni, da potrebujemo manj nožic na integriranih vezjih, zato je pričakovati da potrebujemo manjše število integriranih vezij.

Nadaljujmo s poenostavljanjem. Tisti del zgoraj zapisane funkcije, ki je v oklepaju, ima značilno obliko:

$$Y = \overline{X_1} \cdot X_2 + X_1 \cdot \overline{X_2}$$

Ta izraz se pojavi v mnogih logičnih funkcijah, zato je dobil svoje ime ekskluzivni OR ali XOR. Tako funkcijo enostavneje zapišemo kot:

$$Y = \overline{X_1} \cdot X_2 + X_1 \cdot \overline{X_2} = X_1 \oplus X_2$$

| $X_1$ | $X_2$ | $Z$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |


*Slika 12.3: Funkcija XOR*

Tabela za logično funkcijo XOR je na sliki 12.3, kjer je tudi simbol za vrata. Vrata XOR so podobno kot vrata AND, OR in NOT izdelana v obliki integriranega vezja. Imajo dva vhodna priključka in en izhodni priključek.

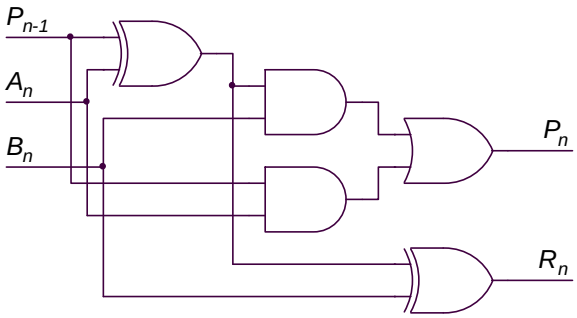
Z vrati XOR funkcijo za prenos še poenostavimo.

$$P_n = (P_{n-1} \oplus A_n) \cdot B_n + P_{n-1} \cdot A_n$$

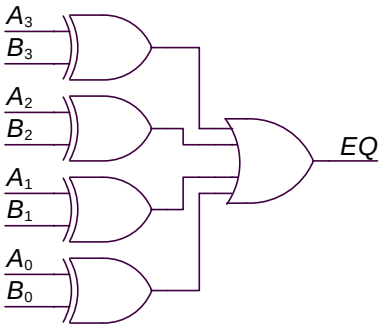
Na soroden način poenostavimo izraz za rezultat $R_n$, ki postane

$$R_n = P_{n-1} \oplus A_n \oplus B_n$$

Kompletno vezje poenostavljene verzije polnega seštevalnika je na sliki 12.4. Pri tem velja poudariti, da izraz $P_{n-1} \oplus A_n$ določi že del vezja za računanje prenosa $P_n$, zato lahko ta del vezja uporabimo še enkrat za računanje vsote $R_n$ ter tako prihranimo ena vrata XOR. Narisano vezje s slike 12.4 opravlja enako funkcijo, kot tisto s slike 9.1, vendar je bistveno manjše.




*Slika 12.4: Poenostavljeno vezje polnega seštevalnika*




*Slika 12.5: Vezje za primerjanje vrednosti dveh štiri-bitnih števil*

Vrata XOR so uporabna tudi za primerjanje dveh števil *A* in *B*, ki sta zapisani z več biti. Števili *A* in *B* sta enaki, kadar so biti zapisa obeh števil paroma enaki. Bit $A_0$ mora biti na primer enak bitu $B_0$, bit $A_1$ mora biti enak bitu $B_1$ in tako naprej za vse bite. Z vezjem s slike 12.5 primerjamo pare bitov. Kadar je par enak, je izhodni signal vrat XOR nič. Kadar so vsi biti paroma enaki, je nič tudi skupni rezultat *EQ*.

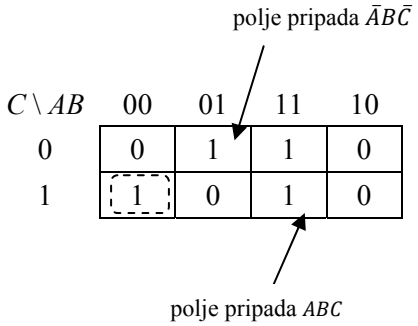
# 13. Grafična metoda za poenostavljanje logičnih funkcij

Ker je potrebnih nekaj izkušenj pri poenostavljanju za to, da izberemo optimalno pot za združevanje obstoječih in vrivanje novih členov, si pogosto pomagamo z grafičnimi metodami, pri kompleksnih logičnih enačbah pa z računalniškimi programi za poenostavljanje. Med grafične metode za poenostavljanje štejemo Karnaugh-ove diagrame, uporaba računalniških programov za poenostavljanje logičnih funkcij pa presega obseg tega gradiva.

Karnaugh-ovi diagrami so tabele s toliko polji, kolikor je možnih kombinacij vhodnih spremenljivk. Vsaki kombinacij vrednosti vhodnih spremenljivk pripada eno polje. V polja vpišemo vrednost funkcije za posamezno kombinacijo vhodnih spremenljivk. Polja so razporejena v stolpce in vrstice tako, da se vrednost posamezne spremenljivke, ki ji polje pripada, v istem stolpcu ali isti vrstici ne spreminja. Še posebej je pomembno, da se med sosednjima stolpcema ali sosednjima vrsticama spremeni vrednost le eni vhodni spremenljivki.

Za zgled je na sliki 13.1 desno narisan Karnaughov diagram za tri vhodne spremenljivke *A*, *B* in *C*. V polja so že vpisane vrednosti funkcije, za katero je Karnaughov diagram sestavljen, funkcija je podana v tabeli levo. Ob stolpcih in vrsticah Karnaugh-ovega diagrama so navedene vrednosti vhodnih spremenljivk. Tako ima na primer v obeh poljih drugega stolpca spremenljivka *A* vrednost nič, spremenljivka *B* pa vrednost ena. Sorodno ima v vsej zgornji vrstici tega diagrama spremenljivka *C* vrednost nič.

| *A* | *B* | *C* | *F* |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |




*Slika 13.1: Tabela za logično funkcijo F in ustrezen Karnaugh-ov diagram*

Iz diagrama izpisujemo logično funkcijo. Če je polje z enko obdano s samimi polji, kjer je vrednost funkcije nič, izpišemo samo kombinacijo spremenljivk za to polje. Tako je na primer označeno polje spodaj levo v

diagramu na sliki 13.1, ki ga popišemo kot:

$$F = \bar{A}\bar{B}C$$

Če je vrednost funkcije na dveh sosednjih poljih enaka, vrednost funkcije očitno ni odvisna od tiste spremenljivke, ki med polji spremeni vrednost. Zato lahko ti dve polji popišemo z enim samim izrazom, ki ne vsebuje omenjene spremenljivke. Takšni sta na primer označeni srednji polji v zgornji vrsti diagrama na sliki 13.2, kjer vrednost funkcije ni odvisna od spremenljivke *A*. Ti polji popišemo v nadaljevanju izraza *F*:

| C \ AB | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |

*Slika 13.2: Isti diagram, nadaljujemo z izpisovanjem*

$$F = \bar{A}\bar{B}C + B\bar{C}$$

Postopek ponavljamo, dokler ne popišemo vseh enic v diagramu. V zgledu je ostalo še eno tako polje spodaj, ki ga združimo s poljem tik nad njim v označeni del in dobimo končni izraz za funkcijo *F*:

| C \ AB | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |

*Slika 13.3: Še enkrat*

$$F = \bar{A}\bar{B}C + B\bar{C} + AB$$

Kadar imamo opravka s funkcijo, za katero je vrednost enaka na štirih sosednjih poljih, ki imajo skupaj obliko pravokotnika, vrednost funkcije ni odvisna od dveh spremenljivk in zato lahko sosednja štiri polja združimo ter popišemo z enim izrazom, ki teh dveh spremenljivk ne vsebuje. Za zgled zapišimo funkcijo za diagram s slike 13.4. Tokrat imamo opravka s funkcijo štirih spremenljivk. Združimo ves drugi stolpec, nato pa še osrednja štiri polja in dobimo:

$$F = \bar{A}B + BD = B(\bar{A} + D)$$

Enako velja za večje število sosednjih polj pravokotne oblike, kjer se vrednost funkcije ne spremeni. Pri izpisovanju lahko polje uporabimo večkrat v več različnih kombinacijah, pomembno je le, da vsakokrat združujemo polja z enako vrednostjo funkcije. Združujemo lahko tudi preko robov diagrama, saj se na primer tudi med prvim in zadnjim stolpcem diagrama s slike 13.4 spremeni vrednost le ene

| CD\AB | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 1 | 0 | 0 |
| 01 | 0 | 1 | 1 | 0 |
| 11 | 0 | 1 | 1 | 0 |
| 10 | 0 | 1 | 0 | 0 |

*Slika 13.4: Zgled za funkcijo štirih spremenljivk*

spremenljivke. iIz diagrama lahko izpisujemo tudi obratno vrednost funkcije $\bar{F}$, za zgled izpišemo iz istega diagrama še to:

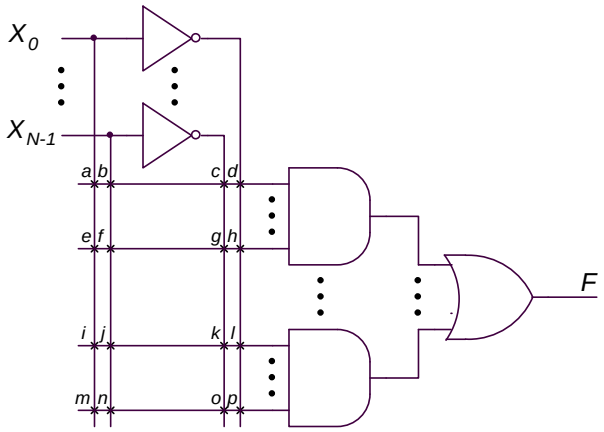
$$\bar{F} = \bar{B} + A\bar{D} \;\Rightarrow\; F = \overline{\bar{B} + A\bar{D}} = \bar{\bar{B}} \cdot \overline{A\bar{D}} = B(\bar{A} + D)$$

Za nekatera vezja velja, da do določene kombinacije vrednosti vhodnih signalov ne more priti. Takrat ni pomembno, katero vrednost ima logična funkcija za te, nemogoče kombinacije vrednosti spremenljivk, zato v diagram na mesta nemogočih kombinacij vhodnih spremenljivk pišemo vrednost *X*, v procesu poenostavljanja pa *X* zamenjamo z 1 ali 0, kar pač olajša poenostavljanje.

# Moduli in vodilo

## 14. Univerzalna struktura za realizacijo logičnih enačb

Vsako logično enačbo lahko realiziramo z logičnimi vrati v njeni osnovni obliki pred poenostavitvijo, če imamo le na razpolago dovolj logičnih vezij. Govorimo o realizaciji logične funkcije v obliki »vsote produktov«, kakršna je na primer tista za polni seštevalnik pred poenostavitvijo. V vezju vedno na desni nastopajo vrata OR, v sredini je niz vrat AND, na levi pa negatorji. Če imamo na razpolago univerzalno vezje, ki je zgrajeno na zgoraj navedeni način, z njim lahko realiziramo poljubno logično funkcijo. Na sliki 14.1 je zgled za tako vezje. Za realizacijo logične funkcije N spremenljivk potrebujemo največ N negatorjev in niz vrat AND s po 2N vhodi ter vrata OR s toliko vhodi, kolikor vrat AND je v vezju. S križci so označena mesta, kjer po potrebi povežemo žice. Za realizacijo funkcije XOR moramo na primer povezati križce z oznakami a, g, l in n.



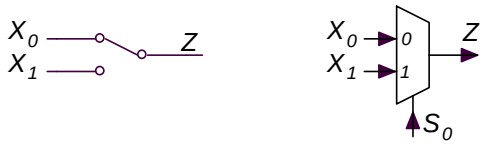

*Slika 14.1: Univerzalna struktura vezja za realizacijo logične funkcije*

Take strukture realizirajo v sodobnih logičnih vezjih tipa PAL oziroma GAL in CPLD. Vezjem s programiranjem definiramo spojena mesta in s tem logično funkcijo. V takem vezju je navadno več na sliki 14.1 narisanih struktur, ki lahko vsebujejo še dodatne elemente, ki poenostavljajo realizacijo logičnih funkcij. Z vezjem GAL16V8 lahko na primer realiziramo 8 logičnih funkcij $F_0$ do $F_7$, vsaka od njih lahko vsebuje do osem »produktov« in je lahko odvisna od do šestnajst vhodnih signalov $X_0$ do $X_{15}$. Torej je v vezju osem vrat OR s po osmimi vhodi, 8x8=16 vrat AND s po 32 vhodi ter 16 negatorjev. Z vezjem MACH4 64/64 lahko na primer realiziramo 64 logičnih funkcij, vsaka je lahko posledica 64 vhodnih spremenljivk. Na razpolago pa so tudi kompleksnejša vezja za večje število logičnih funkcij.

# 15. Nekaj kompleksnejših gradnikov logičnih vezij

**Multiplekser**

V vezjih pogosto potrebujemo stikalo, s katerim je mogoče na izhod posredovati enega od vhodnih signalov. V običajni elektroniki to funkcijo opravlja več položajno stikalo po sliki 15.1 levo. V digitalni elektroniki si pomagamo logičnim vezjem, ki opravlja enako funkcijo, slika 15.1 desno.



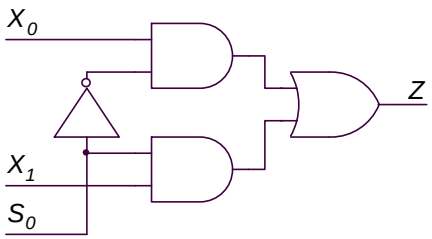

*Slika 15.1: Dvo-položajno stikalo in simbol za multiplekser 2v1*

Če naj ima multiplekser dva vhoda in en izhod, zanj velja tabela s slike 15.2. Tolmačimo jo takole: kadar ima signal $S_0$ vrednost nič, je izhodni signal $Z$ enak vhodnemu signalu $X_0$. Kadar ima signal $S_0$ vrednost ena, je izhodni signal $Z$ enak vhodnemu signalu $X_1$. To zapišemo z enačbo:

$$Z = \overline{S_0} X_0 + S_0 X_1$$

| $S_0$ | $Z$ |
|---|---|
| 0 | $X_0$ |
| 1 | $X_1$ |

*Slika 15.2: Tabela za multiplekser 2v1*




*Slika 15.3: Vezje multiplekserja 2v1*

Multiplekser sestavimo iz logičnih vrat po zgornji enačbi, vezje je na sliki 15.3. Na podoben način sestavimo tudi multiplekserje z več vhodi. Tabela na sliki 15.4 velja za multiplekser s štirimi vhodi, logična funkcija zanj je:

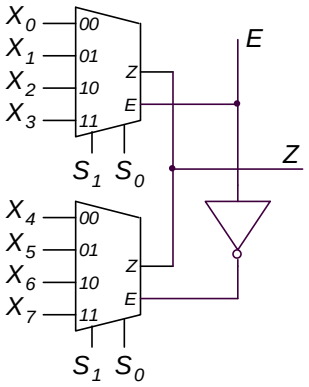
$$Z = \overline{S_1}\,\overline{S_0}X_0 + \overline{S_1}S_0X_1 + S_1\overline{S_0}X_2 + S_1S_0X_3$$

Vezje za tak multiplekser naj si bralec nariše sam. Tokrat za izbiro enega od štirih virov signala potrebujemo dva kontrolna signala $S_1$ in $S_0$.

| $S_1$ | $S_0$ | $Z$ |
|---|---|---|
| 0 | 0 | $X_0$ |
| 0 | 1 | $X_1$ |
| 1 | 0 | $X_2$ |
| 1 | 1 | $X_3$ |

*Slika 15.4: Tabela za multiplekser 4v1*

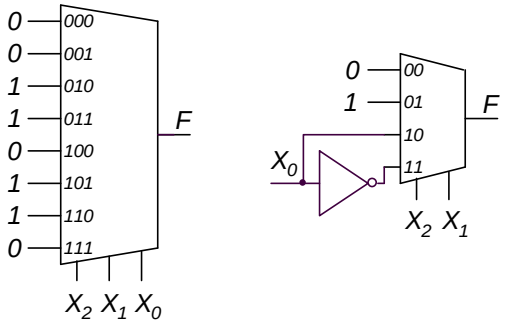
Multiplekserje dobimo tudi v obliki integriranih vezij. Taki so običajno opremljeni z dodatnim priključkom za vhodni signal $E$. Kadar ima signal $E$ vrednost nič, velja zgoraj navedeno. V nasprotnem primeru ni izbran noben od vhodov, izhodni signal $Z$ pa je nedoločen. Več takih multiplekserjev lahko vežemo v večje enote po sliki 15.5, kjer signal $E$ služi za izbiri enega od obeh multiplekserjev tako, da oba skupaj sestavljata multiplekser 8v1.




*Slika 15.5: Sestavljanje multiplekserjev*

Multiplekserje uporabljamo tudi za realizacijo logičnih funkcij. Na sliki 15.6 je logična tabela treh spremenljivk in realizacija v tabeli podane logične funkcije z multiplekserjem. Za funkcijo $N$ spremenljivk potrebujemo multiplekser z $N$ kontrolnimi vhodi in $2^N$ vhodi za signale. Posamezni vhod vežemo na nič ali ena, kot zahteva tabela. Vsako logično funkcijo se da realizirati tudi z manjšim multiplekserjem, ki ima $N$-1 kontrolnih vhodov in $2^{N-1}$ vhodov za signale, pri tem je $N$ spet število vhodnih spremenljivk logične funkcije. Tokrat moramo izpuščeno spremenljivko uporabiti kot vhodno vrednost v multiplekser po sliki 15.6 desno, potrebujemo še največ en negator.

| $X_2$ | $X_1$ | $X_0$ | $F$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |



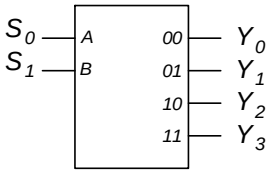

*Slika 15.6: Realizacija logičnih funkcij z multiplekserjem*

**Dekoder – selektor**

Selektor je vezje, ki ima N vhodnih in $2^N$ izhodnih priključkov. S kombinacijo vrednosti na vhodnih priključkih povemo, kateri od izhodnih priključkov naj bo od ostalih različen, navadno nič. Simbol za selektor »1 od 4« je na sliki 15.7, kjer je podana tudi logična tabela zanj. Ta selektor ima štiri priključke za izhodne signale $Y_0$ do $Y_3$ in dva priključka za vhodne signale $S_0$ do $S_1$. Za narisani selektor imajo izhodni signali $Y_0$ do $Y_3$ vrednost ena, le izbrani ima vrednost nič. Za izhodne signale tega selektorja napišemo logične enačbe:

$$\overline{Y_0} = \overline{S_1}\,\overline{S_0} \qquad \overline{Y_1} = \overline{S_1}\,S_0$$

$$\overline{Y_2} = S_1\overline{S_0} \qquad \overline{Y_3} = S_1 S_0$$



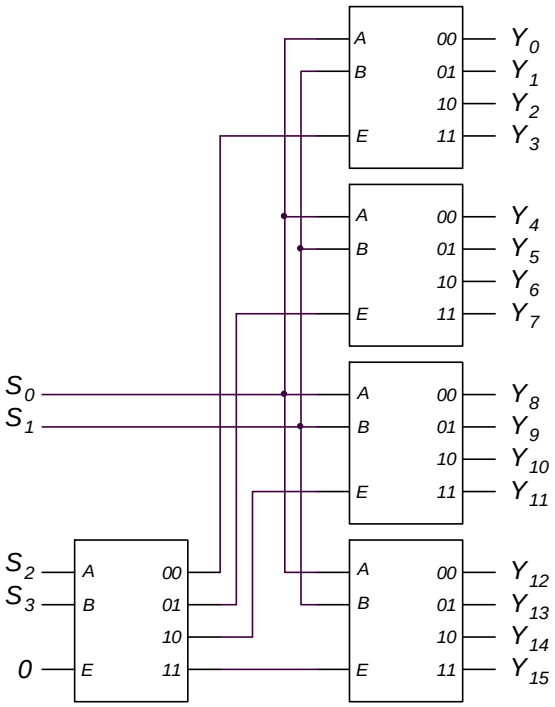

| S1 | S0 | Y3 | Y2 | Y1 | Y0 |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 |

*Slika 15.7: Selektor »1 od 4« in tabela zanj*

Selektorji imajo navadno še dodatni vhodni priključek $E$. Kadar ima signal $E$ vrednost nič, se selektor obnaša tako, kot je bilo navedeno zgoraj. V nasprotnem primeru imajo vsi izhodni priključki enako vrednost. Dopolnjena formula za izhodni signal $Y_0$ je zato:

$$\overline{Y_0} = \overline{S_1}\,\overline{S_0}\,\overline{E}$$

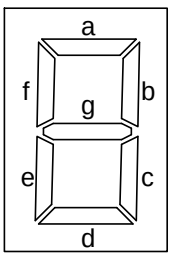
Za ostale izhodne signale naj bralec dopolni izraze sam. Signal $E$ uporabljamo na primer takrat, ko želimo več majhnih selektorjev povezati v en velik selektor, slika 15.8. Selektorje uporabljamo tudi takrat, kadar želimo aktivirati eno od množice enot. Pri tem je pomembno, da je hkrati lahko aktivna le ena od enot, ki jo identificira kombinacija vrednosti signalov $S_i$.




*Slika 15.8: Več majhnih selektorjev sestavlja večji selektor*

**Prekoder – dekoder**

Števila, ki jih uporabljamo v logičnih vezjih, so kodirana. Najpogosteje uporabljamo binarno kodiranje, včasih pa pride prav tudi druga koda. Vzemimo na primer prikaz cifre uporabniku. Za enostavnejši prikaz se uporablja zaslonček s sedmimi segmenti po sliki 15.9, kjer s prižiganjem prave kombinacije segmentov prikažemo cifro. Segmenti so označeni z zaporednimi črkami od a do g, za cifro 3 je treba na primer vklopiti segmente a, b, c, d in g. potrebujemo torej vezje, ki zna binarno kodiran zapis števila prevesti v ustrezno kodo za



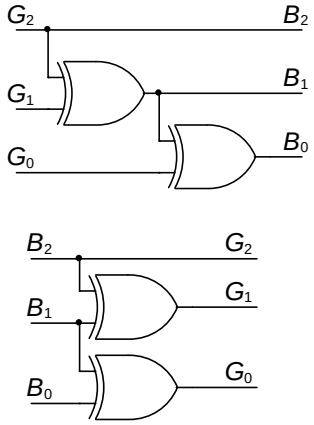

*Slika 15.9: Zaslon s sedmimi segmenti*

segmente tako, da bodo za vsako število svetili pravi. Sestavljena je iz sedmih logičnih vezij, vsako od vezij realizira logično funkcijo po enem od desnih stolpcev v spodnji tabeli; enka pomeni, da segment sveti.

| N | $X_3$ | $X_2$ | $X_1$ | $X_0$ | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 5 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 6 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 7 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 8 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 9 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |

Za fizika je včasih binarni zapis števil neprimeren. Če se zmotimo pri odčitavanju enega samega bita binarno zapisanega števila, je napaka lahko zelo velika. Bolj primeren je zapis, ki si ga je izmislil Gray. Pri tem zapisu se med sosednjima številoma spremeni le en bit, torej je napaka zaradi pomote pri branju enega bita bistveno manjša. V tabeli na sliki 15.10 je podana Grayeva koda ($G_2$ do $G_0$) za tri bitni zapis in ustrezni binarni zapis ($B_2$ do $B_0$). Iz tabele lahko izluščimo prekoder iz Grayevega zapisa v binarni zapis in obratno na isti sliki desno.

| $B_2$ | $B_1$ | $B_0$ | $G_2$ | $G_1$ | $G_0$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 |

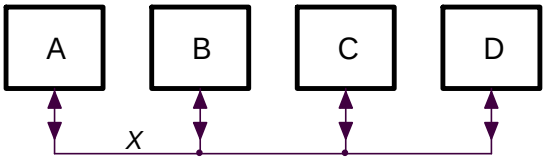



*Slika 15.10: Prekodiranje binarnega zapisa v Grayevega in nazaj*

Poleg navedenih dveh obstaja še vrsta drugih dekoderjev in prekoderjev, ki pa presegajo namen tega zapisa.

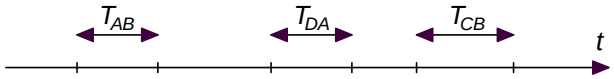
## 16. Koncept vodila in dostop do vodila

V digitalni elektroniki posamezne enote povezujemo med sabo z žicami; te vodijo električne signale med njimi. Kadar je enot veliko, število signalov in s tem žic naraste preko razumnih mej. Takrat se zatečemo k konceptu vodila (angleško »bus«), ki ga bomo tukaj za začetek opisali za eno samo žico *X*, ki povezuje štiri različne enote A do D po sliki 16.1.



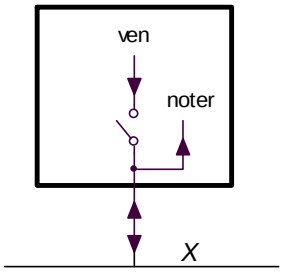

*Slika 16.1: Štiri enote povežemo s skupno žico X, ki služi izmenjavi podatkov; puščice označujejo možne smeri pretoka podatkov*

Izhodnih priključkov več enot ne smemo povezati skupaj. Privzemimo, da enote izmenjujejo podatke le v rednih časovnih intervalih. Enota A pošilja podatke enoti B le v časovnem intervalu $T_{AB}$, potem v časovnem intervalu $T_{DA}$ enota D pošilja podatke enoti A in kasneje v časovnem intervalu $T_{CB}$ enota C pošilja podatke enoti B, kot je to prikazano na časovnem diagramu na sliki 16.2.




*Slika 16.2: Enote si izmenjujejo podatke v definiranih časovnih intervalih*

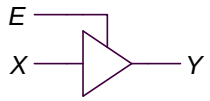
Če so enote opremljene s stikali po sliki 16.3, lahko vse enote povežemo skupaj. Signal, ki naj ga posamezna enota pošilja na skupno žico, je vezan preko stikala v enoti, stikala v vseh enotah pa so normalno razklenjena. Sklenjeno je le v tisti enoti, ki trenutno pošilja podatke na skupno žico. V času $T_{AB}$ je torej sklenjeno stikalo v enoti A, v času $T_{DA}$ stikalo v enoti D in v času $T_{CB}$ stikalo v enoti C. Vse enote so pripravljene ves čas prevzeti signal s skupne žice, saj je ta speljan v enoto mimo




*Slika 16.3: S stikalom v vsaki enoti omogočimo vezavo s slike 16.1*

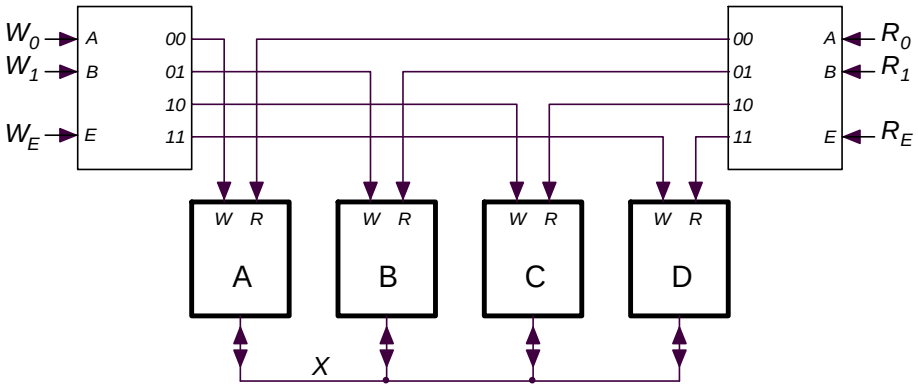
stikala.

V digitalni elektroniki ne uporabljamo mehanskih stikal, njihovo funkcijo opravljajo vrata s tako imenovanim tri-stanjskim (»three-state«) izhodom. Vrednost signala na izhodu teh vrat je lahko ena, nič ali pa Z, kar predstavlja razklenjeno stikalo. Simbol za taka vrata je na sliki 16.4. Kadar ima signal *E* vrednost nič, vrata signal *X* posredujejo na izhodni priključek *Y*. Kadar ima signal *E* vrednost ena, je izhodni signal *Y* v stanju *Z*, kot da bi bilo stikalo razklenjeno.




*Slika 16.4: Vrata s »three-state« izhodom*

Vsaka od enot A do D potrebuje dva kontrolna signala. Preko signala *W* enoti povemo, da je njej namenjen podatek na žici in naj ga zato pobere s skupne žice in uporabi (»write«, shrani za nadaljnjo uporabo). O celicah za shranjevanje bomo govorili v naslednjem poglavju. Preko signala *R* povemo enoti, da naj na skupno žico pošlje signal, ki je namenjen drugim enotam (»read«, beri), torej sklene svoje stikalo. Enote krmilimo preko selektorjev; ti zagotavljajo, da je hkrati aktivna le ena enota, slika 16.5.

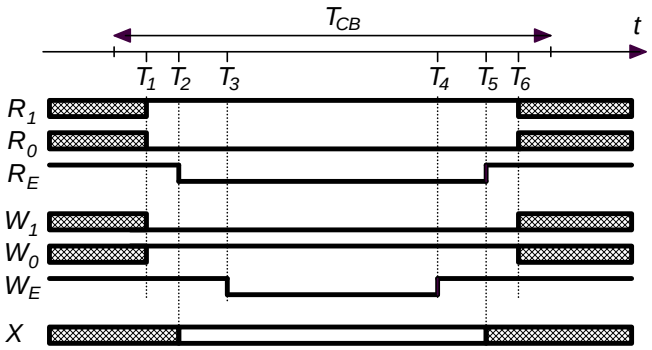



*Slika 16.5: Blokovna shema vezja z vodilom in krmiljem*

S signaloma $R_1$ in $R_0$ izberemo tisto enoto, ki naj podatke pošilja na skupno žico *X*, s signaloma $W_1$ in $W_0$ pa tisto enoto, ki naj bere s skupne žice *X*. S signaloma $W_E$ in $R_E$ izberemo trenutek vpisovanja oziroma branja. Na sliki 16.6 je narisan potek signalov, ki zagotavlja prenos podatkov od enote C do enote B v časovnem intervalu $T_{CB}$. Šrafirani del pomeni, da takrat vrednost signala ni pomembna.

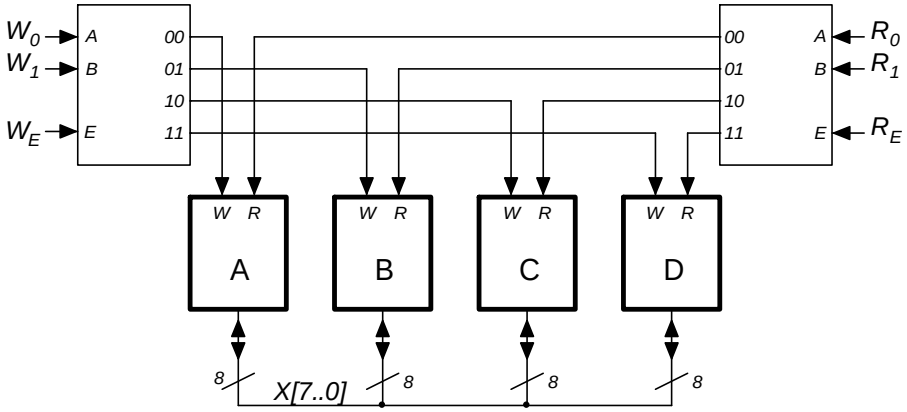
Ob času $T_1$ dobijo signali $R_1$, $R_0$, $W_1$ in $W_0$ vrednosti, ki zagotavljajo, da bo za branje izbrana enota C in za pisanje izbrana enota B. Ob času $T_2$ signal $R_E$ dovoli selektorju izbiro enote C za branje, takoj za tem se podatek iz enote C

znajde na skupni žici $X$. Ob času $T_3$ signal $W_E$ omogoči pisanje, ki traja do časa $T_4$, ko signal $W_E$ spet onemogoči pisanje. Ob času $T_5$ signal $R_E$ onemogoči še branje iz enote C, zaradi česar na vodilu $X$ signal ni več definiran, nato se lahko poljubno spremenijo še signali $R_1$, $R_0$, $W_1$ ter $W_0$.




*Slika 16.6: Potek kontrolnih signalov za prenos podatkov od enote C do enote B*

V opisanem primeru je vodilo sestavljala ena sama žica, torej so si enote lahko podajale le omejen nabod informacij. Vodilo navadno sestavlja osem, šestnajst ali dvaintrideset žic, preko njih je mogoče med enotami pošiljati v enem koraku mnogo več informacij. Kadar uporabljamo večbitno vodilo, to še vedno naredimo tako, kot je narisano na sliki 16.5, le eno žico vodila nadomestimo z množico žic, slika 16.7. To zahteva več elementov v notranjosti posamezne enote. Za osem-bitno vodilo potrebujemo v vsaki enoti osem vrat s tri-stanjskim izhodom, za vsako žico vodila po ena vrata. Prav tako potrebujemo v enoto več celic, kamor bomo vpisovali vrednosti z osmih žic vodila. Po takem vodilu lahko med enotami v enem koraku potujejo števila z vrednostjo med nič in 255 ($2^8$-1).

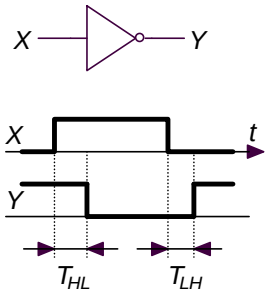



*Slika 16.7: Prenos podatkov med enotami preko 8-bitnega vodila*

# Časovni potek signalov

## 17. Zamujanje v logičnih vezjih

Vrednosti signalov se v digitalnih vezjih s časom spreminjajo. Logična vrata na spremembo ne odgovorijo takoj, ampak za računanje nove vrednosti potrebujejo nekaj časa. Ta čas imenujemo zamujanje vrat, slika 17.1. Časa zamujanja za spremembo izhodnega signala iz ena v nič in iz nič v ena sta lahko različna, zato sta označena kot $T_{HL}$ in $T_{LH}$. Za različne družine logičnih vrat se zamujanje razlikuje, tipična vrednost pa znaša nekaj nanosekund. Na prvi pogled je ta čas videti kratek in zato zanemarljiv, vendar temu ni tako. Pomislimo na hitrost delovanja sodobnih računalnikov, ki delujejo s frekvenco ure v področju GHz, torej traja posamezen cikel ure manj kot nanosekundo. Zamujanje je v primerjavi z navedenim nedopustno dolgo.

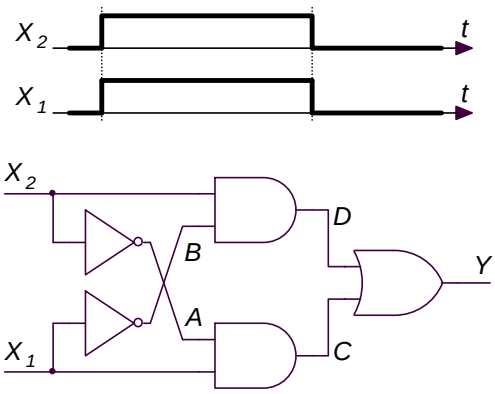



*Slika 17.1: Definicija zamujanja*

Zaradi zamujanja je treba na rezultat malo počakati, lahko pa tudi močno vpliva na računanje logične funkcije. Za zgled poglejmo računanje logične funkcije XOR. Logična enačba, po kateri je tako vezje sestavljeno, je lahko:

$$Y = \overline{X_1} X_2 + X_1 \overline{X_2}$$

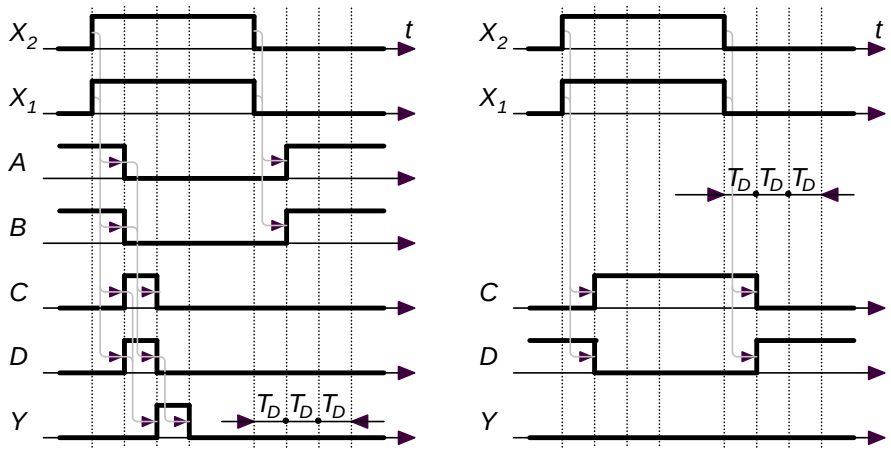
Vezje je na sliki 17.2. Ko na vhoda vrat XOR priključimo signala $X_1$ in $X_2$ na isti sliki zgoraj pričakujemo, da bo izhodni signal $Y$ vrat ves čas enak nič, saj sta vhodna signala ves čas enaka. Žal zaradi zamujanja temu ni nujno tako. Preverimo delovanje tako sestavljenih vrat XOR v idealiziranem primeru, ko je $T_{HL} = T_{LH} = T_D$ in daljši od trajanja preskoka signala vrat iz ene vrednosti v




*Slika 17.2: Vhodna signala in shema za vrata XOR*

drugo. Signali v posameznih točkah vezja so označeni s črkami.

Na levi polovici slike 17.3 je narisan potek signalov v vezju. Zaradi zamujanja je vsak preskok vrednosti izhodnega signala vrat zamujen za preskokom signala na vhodu istih vrat, kar je označeno s puščico. Posledica zamujanja je nepravilen signal v obliki napetostnega sunka na izhodu vrat XOR. Do napake na izhodu je prišlo, ker se vhodni signal razširja proti izhodu po različnih poteh, zamujanje po teh poteh pa zaradi različnega števila logičnih vrat na poti ni enako.




*Slika 17.3: Časovni potek signalov v dveh verzijah vrat XOR*

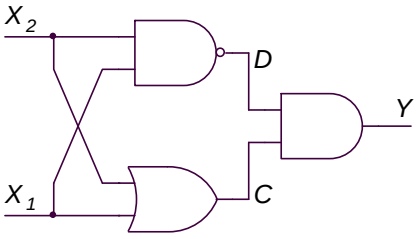
Vrata XOR lahko sestavimo tudi drugače. Najprej preoblikujemo logično enačbo:

$$Y = \overline{X_1}X_2 + X_1\overline{X_2} = (X_1 + X_2) \cdot \overline{X_1 X_2}$$

Na podlagi enačbe sestavimo vezje na sliki 17.4. Če tokrat spet predpostavimo enako zamujanje v posamičnih vratih, do sunka na izhodu ne pride, slika 17.3, desna polovica. Žal ne moremo vsakega vezja preoblikovati tako, da tovrstnih napak ni. Pri nekaterih logičnih vezjih te napake celo ne

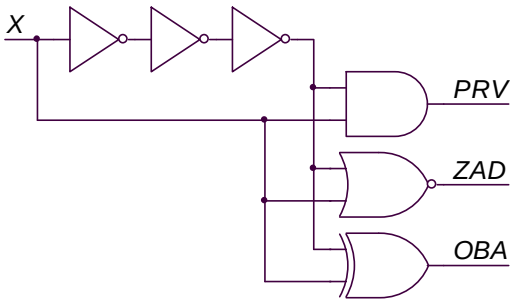
motijo. Kjer bi te napake lahko motile, je potrebno ubrati druge poti, ki so navedene v nadaljevanju tega zapisa.

Omenjenim sunkom, ki so posledica zamujanja, s tujko pravimo »glitch«. So tipično kratki, včasih manj kot znaša zamujanje in lahko povzročijo resne težave, saj jih je zaradi kratkega trajanja težko odkriti in odstraniti.




*Slika 17.4: Drugačna verzija vrat XOR*

Včasih zamujanje s pridom izkoristimo. Vezje s slike 17.5 generira sunke, ko se vrednost vhodnega signala spremeni. Bralcu za vajo prepuščamo ugotavljanje, kdaj pride do posameznega sunka na izhodu.

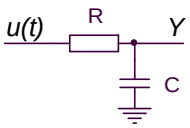



*Slika 17.5: Zamujanje je lahko koristno*

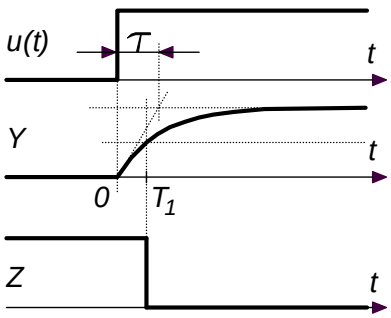
## 18. Formiranje signalov v času – enostavna verzija

Iz analogne elektronike poznamo obnašanje RC člena, slika 18.1. Ko na vhod takega člena priključimo napetostno stopnico *u(t)*, signal na izhodu vezja eksponentno narašča po sliki 18.2. Tako obnašanje izkoristimo s pomočjo logičnih vrat za generiranje zakasnjenih signalov. Vrata zvezno se spreminjajočo napetost na izhodu RC člena interpretirajo le kot nič ali ena.

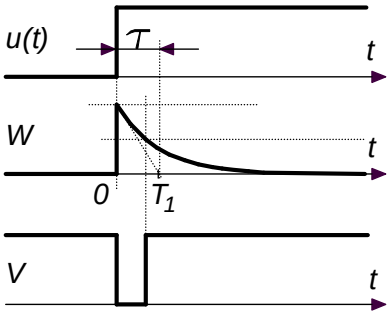
Vzemimo, da imamo opravka z negatorjem, za katerega je meja med logično nič in ena na vhodu enaka ½ napajalne napetosti. Tak negator na primer ves čas do $T_1$ signal *Y* na svojem vhodu interpretira za nič in odgovori na izhodu z ena, po času $T_1$ pa interpretira signal *Y* za logično ena in odgovori z nič. Na izhodu logičnih vrat dobimo zato




*Slika 18.1: RC člen*




*Slika 18.2: Potek signalov v RC členu in negatorju*




*Slika 18.3: Potek signalov po zamenjavi R in C*

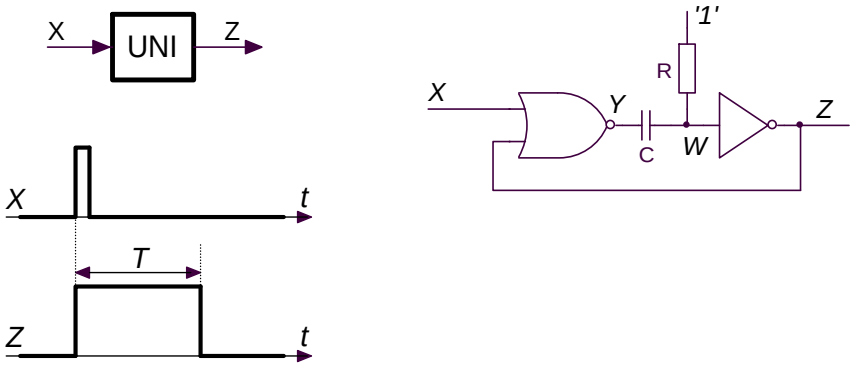
zakasnjeno in negirano verzijo napetostne stopnice *u(t)*, kasnitev $T_1$ pa je odvisna od časovne konstante $\tau$ RC člena in za dan primer znaša 0,69RC.

Če v RC členu zamenjajmo mesti upornika in kondenzatorja, je izhodna napetost RC člena drugačna, slika 18.3. Ob poskoku *u(t)* poskoči za enako vrednost tudi napetosti v točki *W* na izhodu iz RC člena, zatem pa eksponentno usiha proti nič. Tudi tokrat lahko sunek *W* povežemo na vhodni priključek negatorja in dobimo signal *V*. Sunek na izhodu negatorja pove, da je na vhodu v RC člen prišlo do poskoka napetosti, trajanje sunka na izhodu vrat pa je odvisno od časovne konstante RC člena.

Pri logičnih vratih je meja med področjema, ki sta interpretirana kot logična nič oziroma ena, le slabo določena. Še pri isti tehnologiji izdelave vrat se spreminja v dokaj širokem območju, zato je kasnitev oziroma trajanje sunka po zgornjih zgledih nezanesljivo in ga ni mogoče v naprej natančno izračunati. Tako formiranje sunkov oziroma kasnenje uporabimo zato samo takrat, ko točen čas ni najpomembnejši.

# 19. Univibrator

Kadar potrebujemo napetostni sunek konstantnega trajanja in velikosti, uporabimo univibrator, blok je na sliki 19.1 levo zgoraj. Za en sunek napetosti *X* na vhodu v univibrator UNI dobimo na izhodu univibratorja en sunek napetosti *Z*, le da je ta neodvisen od trajanja ali velikosti sunka *X*, ista slika spodaj levo. Shema vezja je na sliki 19.1 desno. Trajanje sunka napetosti na izhodu je odvisno od časovne konstante RC člena v vezju. Tudi tu velja, da je trajanje odvisno od električnih lastnosti logičnih vrat, zato trajanja izhodnega sunka ne moremo v naprej točno določiti.



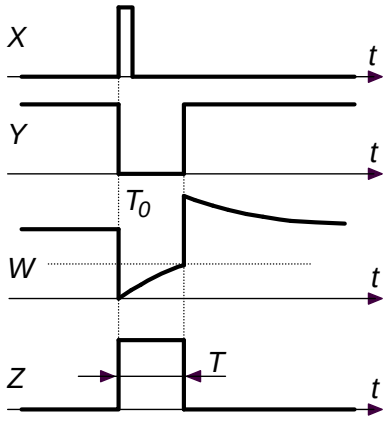

*Slika 19.1: Blok, funkcija in shema univibratorja*

Časovni potek signalov v vezju je na sliki 19.2. Če je vhodni signal *X* dolgo časa nič, se kondenzator C nabije tako, da tok skozi upornik R ne teče več. Napetost v točki *W* je takrat enaka '1', kar negator interpretira kot logično ena in se odzove z vrednostjo nič na izhodu *Z*. Vrednost nič na obeh vhodih vrat NOR povzroči enko na izhodu *Y* istih vrat.

Ko signal *X* na vhodu univibratorja ob času $T_0$ poskoči na ena, se zaradi tega signal *Y* spremeni v nič. Ker se napetost na kondenzatorju C ne more hipoma spremeniti, se takoj spremeni na nič tudi vrednost signala *W*. Na izhodu *Z* negatorja se zato pojavi vrednost ena. Ker je signal *Z* povezan tudi na vhodni priključek vrat NOR, ostane izhodni signal *Y* vrat NOR enak nič tudi, če/ko se vrednost signala *X* na vhodu univibratorja vrne na nič.

Ker je napetost na uporniku R sedaj od nič različna, teče skozenj tok. Ta tok polni kondenzator C, zato napetost v točki *W* eksponentno narašča. Ko se napetost v točki *W* poveča do vrednosti, ki jo negator interpretira kot logično ena, se izhodni signal negatorja vrne na nič. Do tega trenutka je minilo *T* časa.

Za vsak sunek na vhodu dobimo na izhodu univibratorja en sunek trajanja *T*, ki je odvisen od časovne konstante RC v vezju, deloma pa tudi od električnih lastnosti uporabljenih logičnih vrat. Zato trajanje izhodnega

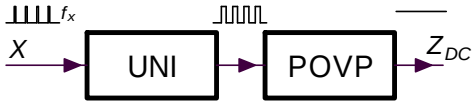



*Slika 19.2: Potek signalov v vezju s slike 19.1*

sunka ni posebej natančno določeno.

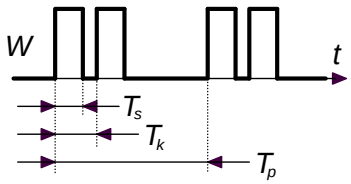
Univibrator uporabljamo v vezjih, kjer želimo prepoznati spremembo vrednosti vhodnega signala in to določen čas posredovati drugim delom digitalnega vezja. Enostavni zgledi vključujejo na primer analogno merjenje frekvence, kasnitev in kombiniranje zakasnjenih sunkov ter generiranje sunkov znanega trajanja.

Blokovna shema sistema za merjenje frekvence je podana na sliki 19.3. Vhodne sunke $X$ s frekvenco $f_x$ najprej s pomočjo univibratorja UNI izenačimo po trajanju in velikosti, nato pa povprečimo v enoti POVP. Za povprečenje lahko uporabimo RC člen s časovno konstanta, ki je mnogo večja od intervala med dvema zaporednima sunkoma na vhodu merilnika. Izhodna napetost $Z_{DC}$ je v tem primeru sorazmerna frekvenci vhodnega signala in jo lahko merimo z navadnim voltmetrom za enosmerne napetosti.



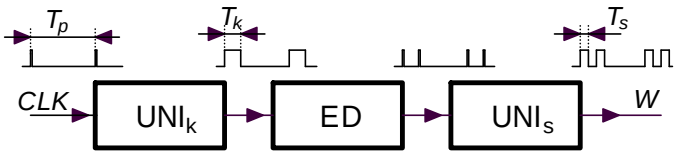

*Slika 19.3: Enostaven merilnik frekvence z univibratorjem in povprečevalnikom*

Z univibratorji lahko na primer sestavimo vezje, ki generira signal $W$ s slike 19.4, pri tem je trajajanje sunkov $T_s$ nastavljivo, prav tako je nastavljiv časovni interval $T_k$ med parom sunkov. Periodo $T_p$ ponavljanja parov sunkov definira frekvenca signala CLK.




*Slika 19.4: Tak niz sunkov lahko generiramo tudi z univibratorji*

Prvi univibrator $UNI_k$ za vsak vhodni sunek ure *CLK* generira en sunek trajanja $T_k$. Te sunke vodimo na detektor prehodov ED (podrobna shema za tak detektor je na sliki 17.5). Na izhodu OBA detektorja prehodov dobimo kratek sunek ob začetku in ob koncu sunka $T_k$, s tema sunkoma prožimo univibrator $UNI_s$, ki generira želeni signal $W$.

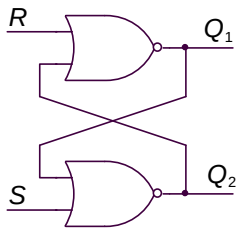



*Slika 19.5: Vezje za generiranje signala s slike 19.4*

# Spominske celice in njihova raba

## 20. Spominska celica RS

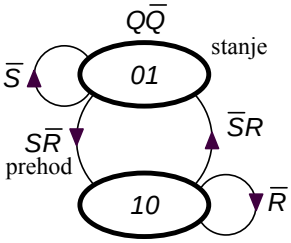
Osnovna spominska celica je dvojček ali flip-flop. Sestavimo ga iz dveh vrat tipa NOR po sliki 20.1. Pri reševanju tega vezja logične enačbe zaradi prepletenosti povezav ne pomagajo, zato poskusimo s kombiniranjem in sledenjem signalov. Za začetek naj imata oba vhoda $R$ in $S$ vrednost nič. Predpostavimo, da ima izhod $Q_1$ vrednost ena, torej je enka tudi na enem vhodu spodnjih vrat, zato je izhodna vrednost teh vrat $Q_2$ enaka nič. Ker je vezje simetrično prav tako lahko začnemo s predpostavko, da ima izhod $Q_1$ vrednost nič in ugotovimo, da ima v tem primeru izhod $Q_2$ vrednost ena. V obeh primerih se izhodni vrednosti ohranjata dokler imata vhodna signala $R$ in $S$ vrednost nič.




*Slika 20.1: Dvojček ali flip-flop z vrati NOR*

Pravimo, da se flip-flop nahaja v enem od dveh možnih stanj. Za stanje je značilna kombinacija vrednosti izhodnih signalov $Q_1$ in $Q_2$, ki za narisano vezje znaša '10' ali '01'.

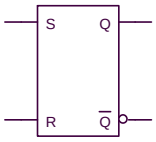
Flip-flop zamenja stanje zaradi vhodnih signalov. Naj bo flip-flop v stanju '10', torej je $Q_1$ enak ena. Vhod $S$ naj bo ves čas enak nič. Ko vhod $R$ postane ena, se izhod $Q_1$ spremeni v nič, dve ničli na vhodih spodnjih vrat pa postavita izhod $Q_2$ na ena. Novo stanje je torej '01'. Čeprav vhod $R$ vrnemo na nič, se stanje '01' ohranja. Ko potem pri stalni vrednosti nič na vhodu $R$ spremenimo vrednost vhoda $S$ na ena, se stanje spet zamenja v '10'. Z vhodnim signalom $S$ torej postavimo izhod




*Slika 20.2: Diagram prehodov za RS flip-flop*

$Q_1$ na ena, z enko na vhodu *R* pa isti izhodni signal vrnemo na nič.

Za obe stanji je značilno, da imata izhoda komplementarno vrednost, zato ju označimo s $Q$ in $\bar{Q}$ namesto $Q_1$ in $Q_2$. Stanja flip-flopa in prehode med njimi pokažemo v diagramu prehodov na sliki 20.2, pri tem smo stanje označili z vrednostjo obeh izhodov. V diagram vrišemo prehode med stanji in pogoje za prehode. Stanje '01' se ohranja, če ima signal *S* vrednost nič. Stanje se zamenja iz '01' v '10' če ima vhod *S* vrednost ena in vhod *R* vrednost nič. Podobno interpretiramo desno polovico diagrama, ki popisuje prehode iz stanja '10'.




*Slika 20.3: Simbol za RS flip-flop*

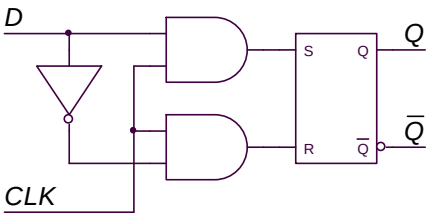
Tak flip-flop imenujemo RS flip-flop po njegovih vhodih. V digitalni elektroniki uporabljamo RS flip-flope, ki jih kupimo v obliki integriranih vezij, simbol zanj je na sliki 20.3.

Če hkrati na oba vhoda *R* in *S* priključimo enko, dobita oba izhoda $Q$ in $\bar{Q}$ vrednost nič. Tega stanja ne izrabljamo in zanj pravimo, da je prepovedano, saj je od hitrosti vrat odvisno delovanje flip-flopa takrat, ko sočasno oba vhodna signala *R* in *S* vrnemo na nič. V diagramu prehodov na sliki 20.2 ta možnost ni vrisana.

RS flip-flop je osnovna spominska celica, ki se lahko nahaja v enem od dveh možnih stanj, pri tem sta izhodna signala komplementarna. Iz enega v drugo stanje flip-flop preide zaradi kombinacije vrednosti vhodnih signalov, do prehoda pride takoj po spremembi vrednosti vhodnih signalov.

# 21. Spominska celica D

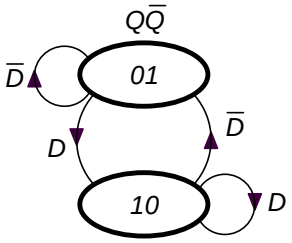
Pogosto želimo vpisovati novo vrednost v flip-flop samo v natančno določenem trenutku. Takrat postavimo pred RS flip-flop še dvoja vrata AND in negator po sliki 21.1. Negator onemogoča hkratno kombinacijo obeh vhodov *R* in *S* je ena; le eden od vhodov v RS flip-flop ima sedaj lahko vrednost ena, novi vhod imenujemo *D*, kompletno vezje pa D flip-flop. Dvoja vrata AND dopuščajo prehajanje flip-flopa iz stanja v stanje le takrat, ko ima signal *CLK* vrednost ena.




*Slika 21.1: Shema spominske celice D*

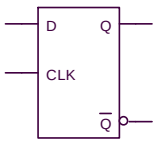
Diagram prehodov za D flip-flop je na sliki 21.2. Zanj je značilno, da do prehodov pride le takrat, ko to dopušča enica na vhodu *CLK*. Kadar ima signal CLK vrednost nič se stanje flip-flopa ohranja ne glede na vrednost vhoda D! Pravimo, da do

prehodov pride sinhrono z uro *CLK*.




*Slika 21.2: Diagram prehodov za D flip-flop*

Tudi D flip-flope kupimo v integriranih vezjih, njihov simbol v elektronskih shemah je na sliki 21.3. Na tržišču so na razpolago tudi popolneje opremljeni D flip-flopi, ki imajo na primer dodatna vhoda *R* in *S* ali pa dopuščajo prehod med stanjema ob komplementarni vrednosti ure *CLK*. Spet drugim za prehod med stanji zadostuje že preskok ure *CLK* iz ene v drugo vrednost, na primer menjava iz nič v ena. Takim pravimo, da so občutljivi za prehod ure in jih bomo v nadaljevanju največ uporabljali. Narejeni so tako, da je v linijo *CLK* do vhoda v oboja vrata AND vstavljeno še vezje za prepoznavanje roba sunka s slike 17.5, uporabljan pa je izhodni signal *PRVI*.

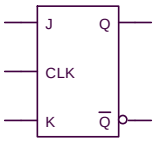



*Slika 21.3: Simbol za D flip-flop*

D flip-flop je torej spominska celica, v katero se vpiše vrednost vhoda *D* samo takrat, ko to dopusti signal vhodu *CLK*, sicer se stanje flip-flopa ohranja. Pravimo, da pride do vpisa sinhrono z uro *CLK*.

## 22. Spominska celica JK

Pri gradnji kompleksnejših števnikov potrebujemo flip-flope. Zaporedje stanj števnikov definiramo z logičnimi funkcijami, torej vrati, ki jih vežemo pred vhode flip-flopov. O tem se bomo poučili v naslednjih poglavjih. Ker za kompleksne števnike potrebujemo kompleksne logične funkcije in veliko vrat, je smiselno del logične funkcije že vključiti v flip-flop ter tako reducirati potrebno število logičnih vrat. To omogoča JK flip-flop, ki ima dva vhoda *J* in *K* ter vhod za uro *CLK* ter dva izhoda $Q$ in $\bar{Q}$. Električni simbol za JK flip-flop je na sliki 22.1.




*Slika 22.1: Simbol za JK flip-flop*

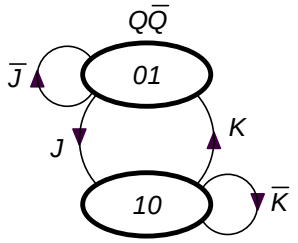
Za JK flip-flop lahko napišemo tabelo prehodov na sliki 22.2. Do vpisa v flip-flop pride le ob za izbrano izvedbo flip-flopa značilni vrednosti ure *CLK* (na primer ob prehodu ure *CLK* iz nič v ena) tako, kot je to veljalo za D flip-flop. V tabeli stolpec $Q^+$ predstavlja vrednost izhoda $Q$ po prehodu ure *CLK*. Iz tabele preberemo, da flip-flop ohranja stanje takrat, ko sta oba vhoda *J* in *K* hkrati nič ter da izhod flip-flopa dobi komplementarno vrednost takrat, ko sta oba vhoda *J* in *K* enaka ena. Za komplementarne kombinacije vhodov *J* in *K* se v flip-flop

vpiše vrednost vhoda *J*.

Diagram prehodov za JK flip-flop izpišemo iz tabele prehodov, slika 22.3. Tudi za ta diagram velja, da do prehoda pride sinhrono z signalom ure *CLK*, sicer se stanje flip-flopa ohranja.

| J | K | $Q^+$ |
|---|---|---|
| 0 | 0 | $Q$ |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | $\bar{Q}$ |

*Slika 22.2: Tabela prehodov za JK flip-flop*

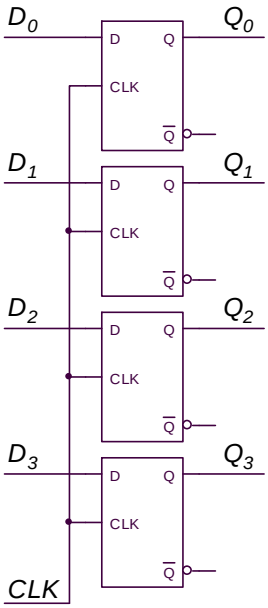



*Slika 22.3: Tabela prehodov za JK flip-flop*

JK flip-flop je torej spominska celica, pri kateri prehajanje iz stanja v stanje sledi kombinaciji dveh vhodnih signalov *J* in *K*. Vgrajena logična vezja znatno olajšajo implementacijo kompleksnih števcev in sekvenčnih vezij.

# 23. Register

Flip-flope uporabljamo za shranjevanje vrednosti spremenljivk. Ker so vrednosti običajno zapisane z več biti, potrebujemo za vsak shranjeni bit svoj flip-flop. Niz flip-flopov imenujemo register. Na sliki 23.1 je shema štiri-bitnega registra, sestavljenega iz D flip-flopov. Vhodni signali v register so označeni z $D_0$ za najmanj pomemben bit do $D_3$ za najbolj pomemben bit, izhodi pa so $Q_0$ do $Q_3$. Ure *CLK* vseh flip-flopov so vezane skupaj, zato se na D vhode priključene vrednosti sočasno zapišejo v vse flip-flope registra. Omeniti velja še praktični napotek: vrednost na vhodih registra mora biti konstantna nekaj nanosekund pred vpisovanjem in še nekaj nanosekund po vpisovanju, drugače je vpisana vrednost nezanesljiva.

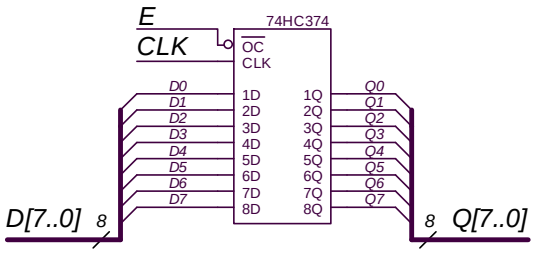



*Slika 23.1: Štiri-bitni register z D flip-flopi*

V računalniških sistemih se običajno uporabljajo osem- ali šestnajst-bitni registri, elektronski simbol za osem-bitni register je na sliki 23.2. Pri tem smo s poševno črtico in

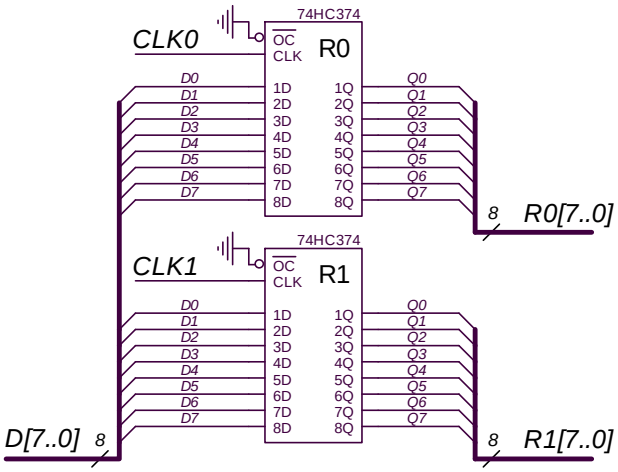
številko ob njej preko linij, ki označujeta vhodni in izhodni vodili nakazali, da gre za vodilo z osmimi žicami. Signal E je dodatni signal za omogočanje izhoda, kot je bil omenjen v poglavju 16 in je narisan na sliki 16.4. Kadar ima signal *E* vrednost nič, register pošilja svojo vsebino na vodilo *Q[7..0]*. Kadar ima signal *E* vrednost ena, je vrednost izhodov iz registra enaka *Z*.

Več osembitnih registrov lahko povežemo na isto osembitno vodilo, po katerem izmenoma prihajajo podatki za posamezen register, s signalom za vpis



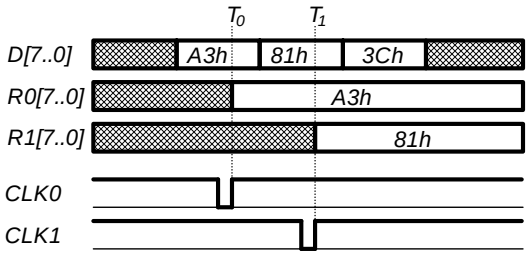

*Slika 23.2: Osem-bitni register, priključen na dve vodili*

*CLK0* do *CLK1* pa ob pravem trenutku zaklenemo vsebino vodila v izbrani register, taka vezava je na sliki 23.3. Ob času $T_0$ se vsebina vodila zapiše v register R0 in ob času $T_1$ v register R1, kot je to narisano na sliki 23.4.




*Slika 23.3: Vezava dveh registrov na skupno vodilo in vpisovanje podatkov z vodila v posamezni register*

Register je torej niz običajno D flip-flopov, ki ga uporabljamo za shranjevanje večbitnih števil. Signal ure je skupen vsem flip-flopom, vhodno in izhodno vodilo pa sta z vsakim bitom vezana na vhod oziroma izhod enega od flip-flopov. Signal z vodila se vpiše v register ob značilnem prehodu signala ure.

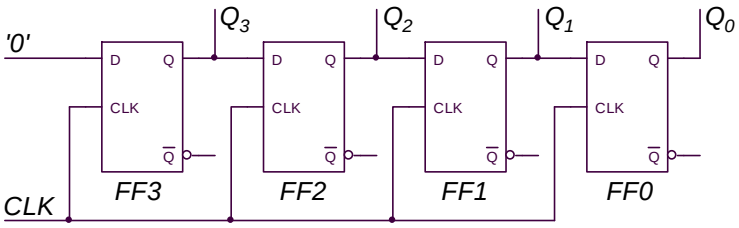



*Slika 23.4: Potek signalov med vpisovanjem vsebine vodila D v registra* R0 *in* R1. *Ob času T0 se zaradi preskoka vrednosti signala CLK0 iz nič v ena vsebina vodila D shrani v register* R0*, zaradi tega dobi vodilo R0[7..0] vrednost* $A3_H$*, kot jo je imelo takrat vodilo D. Podobno se vpiše ob poskoku signala CLK1 tudi vrednost* $81_H$ *v register* R1. *Vrednost* $C3_H$ *se ne vpiše v noben register, saj ni ustreznega poskoka ure CLK. Šrafirani del predstavlja čas, ko vrednost signalov ni znana. Na sliki ni upoštevana kasnitev v registrih.*

# 24. Pomični register

Niz D flip-flopov lahko povežemo tudi tako, da je izhod Q prvega povezan z vhodom D drugega in tako naprej do zadnjega. Shema take povezave štirih flip-flopov je na sliki 24.1, rečemo ji pomični register. Ure CLK vseh flip-flopov so povezane skupaj.

Ob značilnem dogodku na signalu ure se zato trenutna vsebina flip-flopa *FF1* zapiše v flip-flop *FF0*, vsebina flip-flopa *FF2* v flip-flop *FF1* in tako naprej. Dejansko se vsebina takega niza flip-flopov pomakne za eno mesto v




*Slika 24.1: Pomični register, pomika v desno*

desno. Če je bila vsebina pomičnega registra pred značilnim dogodkom na uri *CLK* na primer $1100_2$ in je vhod v levi flip-flop *FF3* enak nič, bo po dogodku nova vsebina pomičnega registra $0110_2$. Naredili smo vezje, ki zna število podeliti z dva.

Povezave med flip-flopi bi lahko speljali tudi drugače. Izhod drugega flip-flopa bi povezali z vhodom v prvi flip-flop, izhod tretjega z vhodom v drugi in tako do konca. V tem primeru bi se ob značilnem dogodku na uri *CLK* vsebina pomaknila za eno mesto v levo, imeli bi torej vezje, ki zna množiti z dva. Vezje bi lahko še dopolnili tako, da bi z multiplekserji glede na vrednost dodatnega vhodnega signala *LD* izbirali eno od zgoraj navedenih povezav in bi torej s signalom *LD* izbrali smer pomika levo ali desno, množenje ali deljenje z dva.

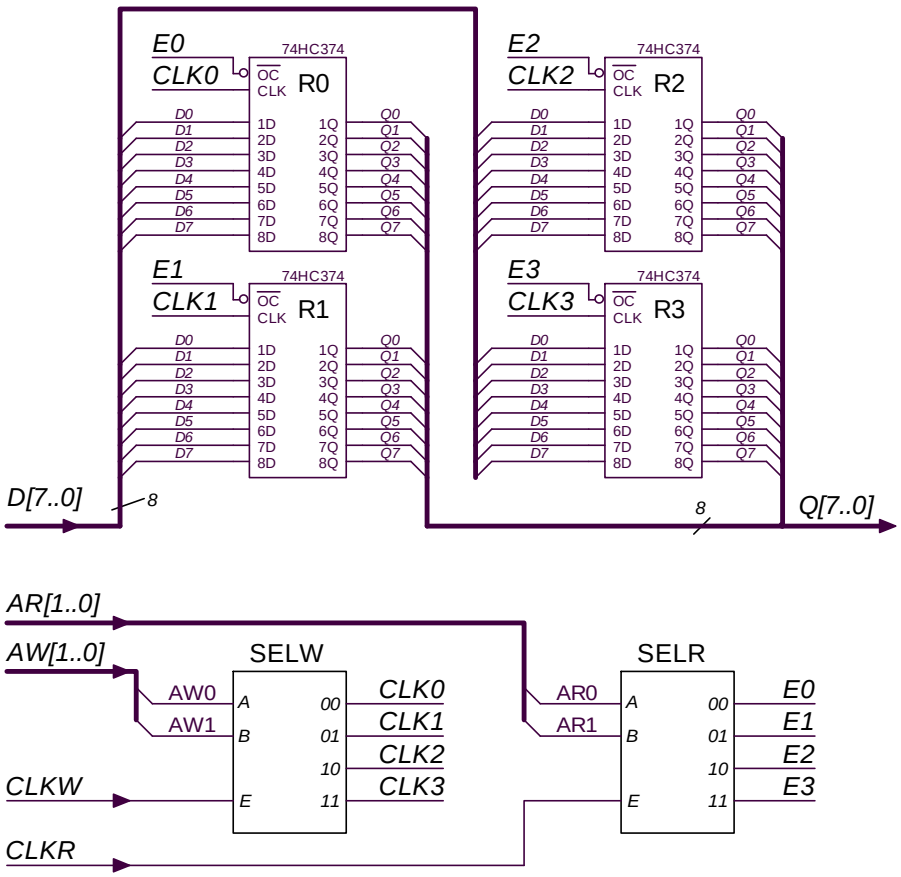
Nekateri proizvajalci dopolnijo navedeno vezje z dodatnimi vhodi, s katerimi je mogoče vzporedno spreminjati vrednost vseh flip-flopov v pomičnem registru hkrati. Vzporedno lahko izvedejo tudi vse izhode Q flip-flopov, tako dobimo pomični register v levo ali desno, ki ga je mogoče vzporedno ali serijsko vpisovati in brati. Taki so koristni pri pretvorbi vzporedno zapisanih podatkov v serijsko obliko in obratno in jih uporabljamo na primer pri osebnih računalnikih za generiranje signalov RS232.

# 25. Spominska enota z več registri ali RAM

Kadar želimo pomniti večjo količino podatkov, potrebujemo več registrov. Vzemimo, da sestavljamo vezje za pomnenje štirih števil, vsako od njih je zapisano s osmimi biti. Potrebujemo torej štiri osem-bitne registre, ki skupaj vsebujejo 32 flip-flopov. Shema vezja je na sliki 25.1.

Vhodno vodilo *D* je povezano na vhodne priključke vseh štirih registrov R0 do R3, izhodno vodilo *Q* pa na izhodne priključke istih registrov. V vezju sta še dva selektorja SELW in SELR za izbiro enega od štirih registrov, signala za izbiranje sta podana na vodilih *AW* in *AR*. Obe vodili sta dvo-bitni, saj imamo opravka z le štirimi registri ($2^2 = 4$). Postopek za vpisovanje je sledeč: najprej se mora na vodilu *D* pojaviti vrednost, ki jo želimo zapisati, na vodilu *AW* pa naslov registra, kamor želimo to vrednost vpisati. Nato je na vrsti signal *CLKW*, ki se mora spremeniti iz vrednosti ena v nič in nazaj v ena, kar povzroči sunek na z vodilom *AW* izbranem izhodu selektorja SELW in posledično sunek signala ure *CLKx* za izbrani register, kamor se zato vpišejo podatki z vodila *D*.

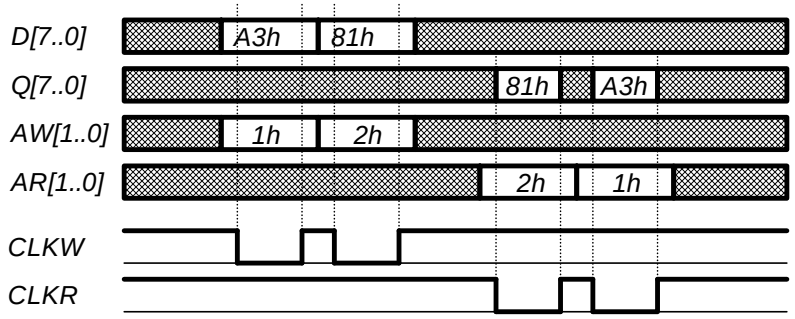
Iz spominske enote želimo tudi brati. Beremo lahko le vsebino enega izbranega registra, ne vseh hkrati. Zato so v vezju uporabljani registri s tristanjskimi izhodi; kadar je kontrolni signal *Ex* za posamični register ena, so njegovi izhodi v stanju *Z*, le kadar je kontrolni signal *Ex* za posamični register




*Slika 25.1: Zgled za spominsko enoto 4x8 (štiri števila po osem bitov), povezave so simbolne; vsi CLK0 so povezani skupaj, vsi CLK1 so povezani skupaj…*

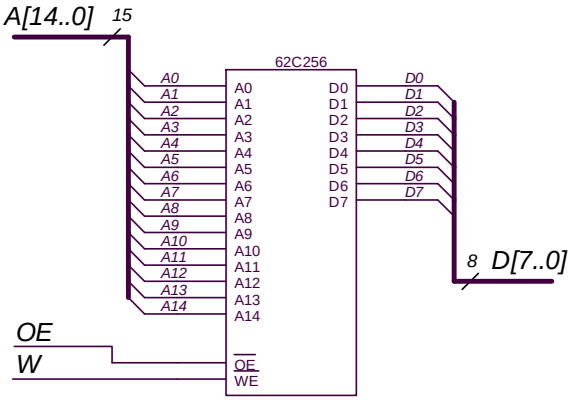
nič, register posreduje vsebino na vodilo *Q*. Le en od kontrolnih signalov *Ex* je lahko hkrati enak nič, ker jih generira selektor SELR. Z vodilom *AR* izberemo enega od registrov za branje, s signalom *CLKR*, ki se spremeni iz ena v nič in nazaj v ena med branjem pa pravi signal *Ex* aktivira izbrani register za branje.

Na sliki 25.2 je zgled za potek signalov med vpisovanjem v spominsko enoto in branje iz nje.

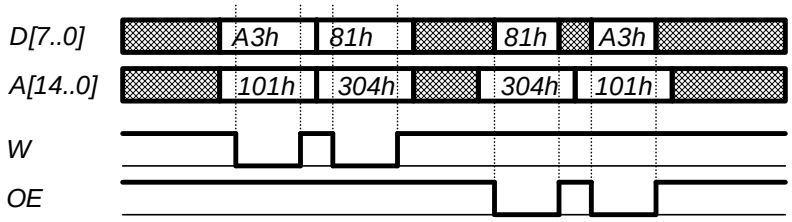



*Slika 25.2: Potek signalov pri pisanju v spominsko enoto in branju iz nje. Najprej v register* R1 *vpišemo vrednost $A3_H$, nato vpišemo vrednost $81_H$ v register* R2. *Kasneje iz registra* R2 *preberemo vrednost $81_H$ na vodilo Q, nato še vrednost $A3_H$ iz registra* R1. *Kasnitev v registrih na sliki ni upoštevana.*

Spominske enote ne gradimo na tak osnoven način, ampak kupimo že narejeno v obliki integriranega vezja. Če se odločimo, da ne bomo hkrati vpisovali v in brali iz spominske enote, lahko shajamo z manj nožicami na integriranem vezju tako, da združimo vodili *D* in *Q* v skupno vodilo *D*. Takrat tudi ne potrebujemo ločenih vodil *AW* in *AR* za izbiranje registra za pisanje in branje, ampak ju združimo v skupno vodilo




*Slika 25.3: Simbol za spominsko enoto 32768 x 8, integrirano vezje 32k x 8 RAM, oznaka 62C256*




*Slika 25.4: Potek signalov v vezju s slike 25.3 za pisanji v registra $101_H$ in $304_H$ ter branji iz teh registrov.*

za izbiro registra *A*. Pri integriranih vezjih sta signala *CLKW* in *CLKR* navadno imenovana *W* in *OE*, simbol za tako enoto, ki vsebuje 32768 registrov, vsaj ima osem flip-flopov, je na sliki 25.3. Za izbiro enega od registrov potrebujemo 15-bitno vodilo *A* (naslovno vodilo), za vpisovanje in branje pa osem-bitno vodilo *D*.

Na sliki 25.4 je potek signalov med pisanjem v tako spominsko enoto in branjem iz nje. Vzemimo, da želimo v spominsko enoto na naslov $101_H$ vpisati vrednost $A3_H$, na naslov $304_H$ pa vrednost $B1_H$. Kasneje želimo najprej prebrati vsebino z naslova $304_H$, nato pa še vsebino z naslova $101_H$.

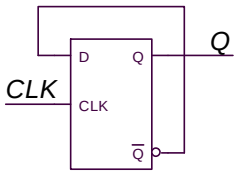
Za vpisovanje najprej spremenimo vrednosti podatkovnega in naslovnega vodila na $A3_H$ in $101_H$, nato za kratek čas spremenimo vrednost signala *W* iz ena na nič in spet nazaj na ena. S tem vpišemo v spominsko celico z naslovom $101_H$ vrednost s podatkovnega vodila. Postopek ponovimo za vpisovanje vrednosti $81_H$ na naslov $304_H$. Pri večini spominskih enot se vrednost s podatkovnega vodila *D* zapiše v izbrani register ob prehodu signala *W* iz nič v ena.

Kasneje želimo vrednosti prebrati iz spominske enote. Najprej izberemo naslov registra v enoti, ki znaša $304_H$, to vrednost postavimo na naslovno vodilo. Zatem signal *OE* spremenimo iz ena v nič, zato se na podatkovnem vodilu pojavi vsebina registra z naslovom $304_H$, ki znaša $81_H$, saj smo to vrednost predhodno tja vpisali. Ko signal *OE* vrnemo na ena, postane podatkovno vodilo nedefinirano, takrat umaknemo tudi naslov registra z naslovnega vodila. Postopek ponovimo za branje registra z naslovom $101_H$.

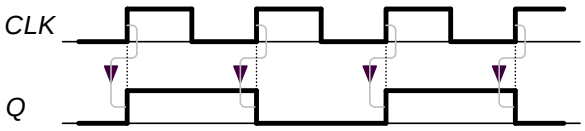
# Števniki in avtomati

## 26. Števnik – asinhrona verzija

Ko povežemo D flip-flop po sliki 26.1, dobimo T (»Toggle«) flip-flop. Vzemimo, da se signal z vhoda D vpiše v flip-flop ob prehodu ure *CLK* iz logične nič v ena. Potem se v T flip-flop ob vsakem pozitivnem prehodu ure vpiše vrednost, ki je nasprotna trenutni vrednosti; stanje se zamenja ob vsakem pozitivnem prehodu ure, kot je to narisano na sliki 26.2.



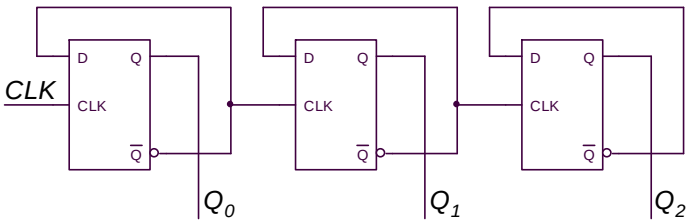

*Slika 26.1: T flip-flop*



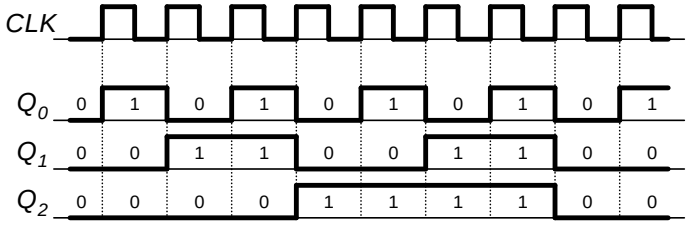

*Slika 26.2: Signali za T flip-flop*

Več T flip-flopov lahko vežemo zaporedno tako, da je vhod CLK v naslednji flip-flop povezan na izhod $\overline{Q}$ predhodnega po sliki 26.3. Izhode tako vezanih flip-flopov označimo s $Q_0$ do $Q_{N-1}$, pri tem je N število flip-flopov, na spodnji skici je N enako 3. Izhodi $Q_2$ do $Q_0$ lahko zavzamejo osem različnih stanj: $000_2$, $001_2$, $010_2$, $011_2$, $100_2$, $101_2$, $110_2$ in $111_2$. Števnik šteje od nič do sedem, potem se štetje ponavlja od nič naprej. Na sliki 26.4 je potek signalov v vezju. V splošnem lahko z N flip-flopi štejemo od 0 do $2^{N-1}$.

Pri asinhronem števniku se vrednost posameznega izhoda spremeni zaradi spremembe predhodnega izhoda. Ker v vsakem flip-flopu signal malo kasni, pri nekaterih prehodih, na primer iz stanje 011 v 100 traja tri kasnitve skozi posamezen flip-flop, da je števnik v novem, pravilnem stanju. Če med spreminjanjem poskušamo brati vrednost števnika, preberemo napačno




*Slika 26.3: S tremi T flip-flopi sestavimo tri-bitni asinhroni števnik*




*Slika 26.4: Potek signalov v tri-bitnem števniku*

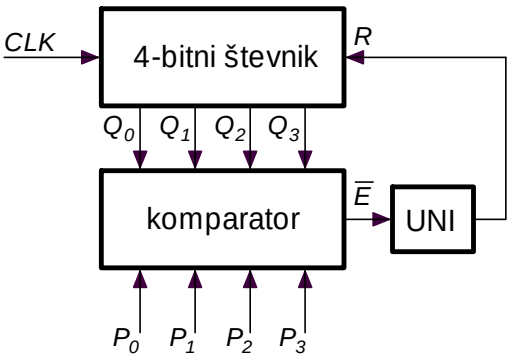
vrednost; nekateri flip-flopi so že v novem stanju, drugi še ne. Efekt je podoben podiranju postavljenih domin. Na roko podremo prvo, vendar traja nekaj časa, preden se podre tudi zadnja. Branje je zato smiselno šele takrat, ko se stanje števnika ustali.

Asinhroni števniki so hitri. Na vhod prvega flip-flopa lahko pripeljemo signal, katerega polperioda je enaka vrednosti kasnitve v flip-flopu in števnik bo signal s tako frekvenco uspešno štel. Tipična vrednost kasnitve v flip-flopu je nekaj nanosekund.

Števnike kupimo v obliki integriranih vezij. Navadno imajo še dodatne vhode, na primer za postavljanje na izhodiščno vrednost, ki ga imenujemo R (»Reset«). Nekateri števniki so taki, da jih lahko postavimo na katerokoli začetno vrednost, taki imajo vhod L (»Load«) in vhode za začetno vrednost ($A_0$ do $A_{N-1}$). Spet drugi lahko štejejo naprej ali nazaj, taki potrebujejo še vhod za definiranje smeri štetja U/D (»Up / Down«).

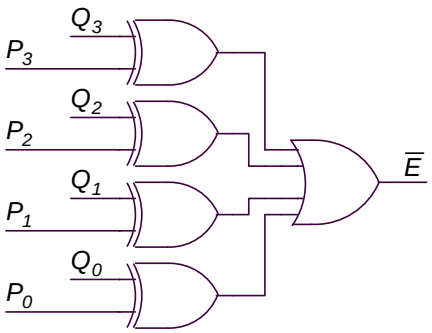
Včasih potrebujemo asinhroni števnik, ki šteje do števila K, to je drugačno od $2^{M-1}$. Tipičen zgled je števnik, ki šteje na dekadni način od 0 do 9. Takrat uporabimo shemo na sliki 26.5. Komparator primerja stanje števnika $Q_3$ do $Q_0$ z referenčno vrednostjo $P_3$ *do* $P_0$. Ko je stanje števnika enako referenčni vrednosti, komparator preko univibratorja UNI postavi števnik na izhodiščno vrednost. Navedeni števnik šteje do več kot *K*, s komparatorjem pa nabor njegovih možnih stanj le omejimo. Komparator lahko sestavimo s pomočjo niza vrat XOR po sliki 26.6, na vhode vrat paroma priključimo bite *P* in izhode števca *Q*. Ko so vsi izhodi števca paroma enaki z biti referenčnih

vrednosti, so izhodi vseh vrat XOR nič in vrata OR posredujejo na vhod *R* števnika logično nič, ki ga resetira. Tak števnik je lahko slab, če s kratkim sunkom *R* ne uspemo hkrati postaviti vseh njegovih flip-flopov na izhodiščno vrednost. Zato je smiselno podaljšati sunek *R* na čas vsaj 10ns, ki zagotovo postavi vse flip-flope na izhodiščno vrednost. Temu lahko služi univibrator UNI.



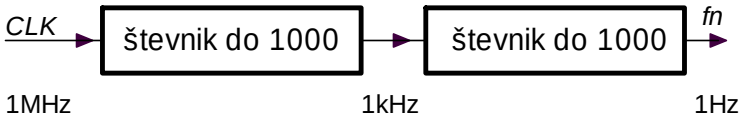

*Slika 26.5: Števnik, ki ne šteje do $2^{M-1}$*

Enakovredni števnik tistemu s slike 26.5 je mogoče sestaviti tudi s flip-flopi in logičnimi vezji, ki jih dodamo na vhode CLK in D posameznih flip-flopov, vendar postopek načrtovanja presega namen tega zapisa.




*Slika 26.6: Komparator za štiri bite*

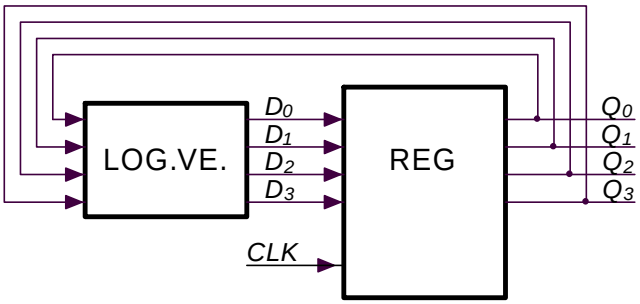
Števnik lahko uporabljamo tudi za spreminjanje frekvence digitalnega signala. Z vezjem s slike 26.7 generiramo signal *fn* s frekvenco 1Hz, čeprav ima vhodni signal *CLK* frekvenco 1MHz. Z drugačnimi števniki lahko generiramo katerokoli frekvenco, ki je dana s celoštevilčnim deljenjem osnovne frekvence ure *CLK*.




*Slika 26.7: Delilnik frekvence*

## 27. Števnik – sinhrona verzija

Štetja se lahko lotimo tudi z registrom REG. Vanj ob vsakem pozitivnem prehodu ure *CLK* vpišemo novo zaporedno število, za primer $0000_2$, $0001_2$, $0010_2$, $0011_2$, $0100_2$... . Tako dobimo števnik, ki šteje sunke ure *CLK*. Novo



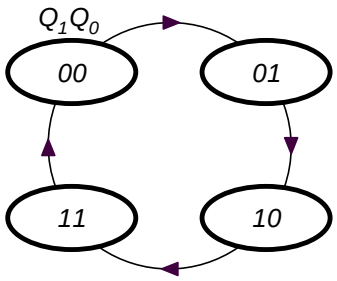

*Slika 27.1: Bločna shema štiri-bitnega sinhronega števnika*

število izračuna logično vezje LOG.VE. iz števila, ki je zapisano v registru. Ustrezna bločna shema za štiri-bitni števnik je na sliki 27.1. Če register na primer sestavimo iz štirih flip-flopov, lahko tak števnik šteje od 0 do 15.

Pri sinhronem števniku se za razliko od asinhrone verzije vsi izhodi spremenijo istočasno, saj je ura *CLK* vezana na vse flip-flope v registru. Največja frekvenca vhodnega signala je v primerjavi z asinhrono verzijo malo manjša, saj mora logično vezje naslednje stanje števnika izračunati iz trenutnega, to pa zahteva nekaj časa.

Načrtovanje sinhronega števnika je opravljeno v treh korakih: najprej zapišemo diagram prehodov med stanji števnika, iz tega izpišemo tabelo prehodov, iz tabele pa izluščimo logične enačbe, ki jih mora logično vezje LOG.VE. pred registrom računati.

Za zgled poskusimo s sinhronim števnikom, ki šteje $00_2$, $01_2$, $10_2$, $11_2$, $00_2$... Potrebujemo register z dvema flip-flopoma tipa D in logično vezje. Izhoda Q flip-flopov sta hkrati izhoda $Q_1$ in $Q_0$ števnika.




*Slika 27.2: Diagram prehodov za dvo-bitni števnik*

Na sliki 27.2 je diagram prehodov. Med stanji števnik prehaja v smeri puščic, po en prehod se zgodi ob preskoku signala ure *CLK* iz nič v ena.

Iz diagrama izpišemo tabelo prehodov na sliki 27.3. Leva stolpca popisujeta vsa možna štiri stanja števnika, izhodna signala sta spet $Q_1$ in $Q_0$. Naslednja dva stolpca popisujeta ustrezajoča naslednja stanja števnika, označena so s spremenljivkama $Q_1^+$ in $Q_0^+$. Pri tem znak $^+$ poudarja, da gre za stanje števnika po pozitivnem prehodu

ure *CLK*. Ker smo uporabili flip-flope tipa D, je na vhode teh flip-flopov potrebno postaviti natanko tisto vrednost, ki naj bi jo izhodi dobili po prehodu ure *CLK*. Signal na vhodu $D_1$ mora biti torej enak vrednosti signala na izhodu $Q_1$ po prehodu ure, enako velja za flip-flop $Q_0$. To označujeta zadnja dva stolpca v tabeli na sliki 27.2. Če bi uporabili drugačne flip-flope, bi zadnja dva stolpca morali napisati drugače.

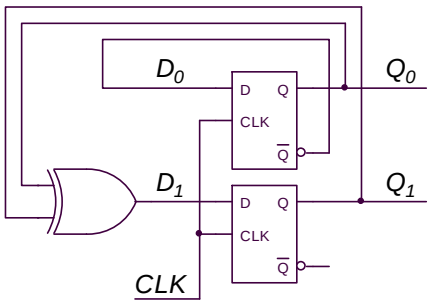
| $Q_1$ | $Q_0$ | | $Q_1^+$ | $Q_0^+$ | $D_1$ | $D_0$ |
|---|---|---|---|---|---|---|
| 0 | 0 | → | 0 | 1 | 0 | 1 |
| 0 | 1 | → | 1 | 0 | 1 | 0 |
| 1 | 0 | → | 1 | 1 | 1 | 1 |
| 1 | 1 | → | 0 | 0 | 0 | 0 |

*Slika 27.3: Tabela prehodov za diagram s slike 27.2*

Iz tabele izpišemo logični enačbi, ki popisujeta odvisnost vhodnih signalov $D_1$ in $D_0$ od trenutnega stanja flip-flopov $Q_1$ in $Q_0$, enačbi se glasita:

$$D_0 = \overline{Q_0} \quad \text{ter} \quad D_1 = Q_1 \oplus Q_0$$

Na podlagi teh dveh enačb lahko narišemo shemo števnika na sliki 27.4.




*Slika 27.4: Shema vezja dvo bitnega števnika*

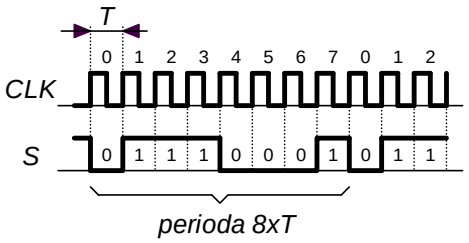
Način načrtovanja sinhronega vezja ima veliko dodatnih možnosti. Za zdaj omenimo samo to, da lahko naredimo števnik, ki šteje v drugačnem zaporedju od zapisanega. Morda nam bolj ustreza zaporedje $00_2$, $01_2$, $11_2$, $10_2$, $00_2$, ... . Za tak števnik je treba le zamenjati vrstni red stanj v diagramu prehodov s slike 27.2, postopek načrtovanja in izdelave vezja pa ostaja enak. Prav tako ni nujno, da števnik uporabi vsa možna stanja, ki jih je $2^N$, pri tem je N število flip-flopov v registru. Če potrebujemo števnik, ki šteje na primer do deset, uporabimo register s štirimi flip-flopi in v diagramu prehodov navedemo samo deset različnih stanj, ki jih povežemo s puščicami za prehode. Iz diagrama spet izpišemo tabelo prehodov in naprej enačbe ter sestavimo vezje.

Za sinhroni števnik je torej značilno, da izhodi spremenijo vrednost sočasno. Vsebuje register in logično vezje, ki iz trenutnega stanja sproti računa naslednje stanje. Postopek načrtovanja števnika je premočrten in sledi

navedenemu receptu, naredimo pa lahko števnika z različnimi zaporedji štetja in številom stanj. Števnike lahko kupimo v integriranih vezjih. Taki imajo navadno še dodatne vhode za smer štetja, postavljanje na izhodiščno vrednost, postavljanje na katerokoli željeno vrednost in podobno.

## 28. Generiranje signalov s števnikom

V digitalni elektroniki pogosto potrebujemo signale, ki po določenem zaporedju menjajo vrednosti med nič in ena. Za primer naj služi signal *S* s slike 28.1, ki ima periodo osem časovnih enot *T*. Zaporedne vrednosti signala v posameznih časovnih intervalih so podane v tabeli desno. Tak signal lahko




| $N_{(10)}$ | $N_{(2)}$ | $S$ |
|---|---|---|
| 0 | 000 | 0 |
| 1 | 001 | 1 |
| 2 | 010 | 1 |
| 3 | 011 | 1 |
| 4 | 100 | 0 |
| 5 | 101 | 0 |
| 6 | 110 | 0 |
| 7 | 111 | 1 |

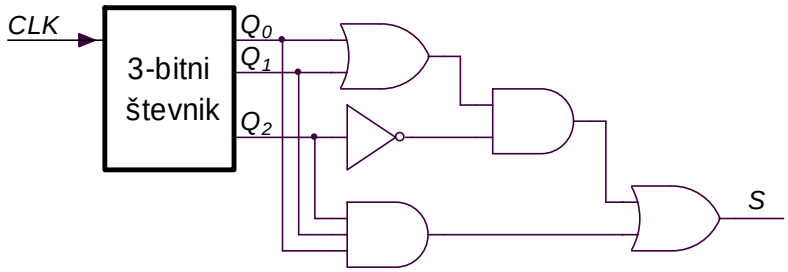
*Slika 28.1: Tole zaporedje potrebujemo, na levi je časovni potek, na desni ustrezna tabela vrednosti signala S v odvisnosti od zaporednega sunka N ure CLK*

generiramo s pomočjo števnika, ki ima osem različnih stanj in logičnega vezja, ki iz trenutnega stanja števnika izračuna vrednost signala *S*. Perioda signala ure *CLK* ustreza časovni enoti *T*. Logično enačbo za vezje ob števniku izpišemo iz tabele in se glasi:

$$S = \overline{N_2} \cdot (N_1 + N_0) + N_2 \cdot N_1 \cdot N_0$$

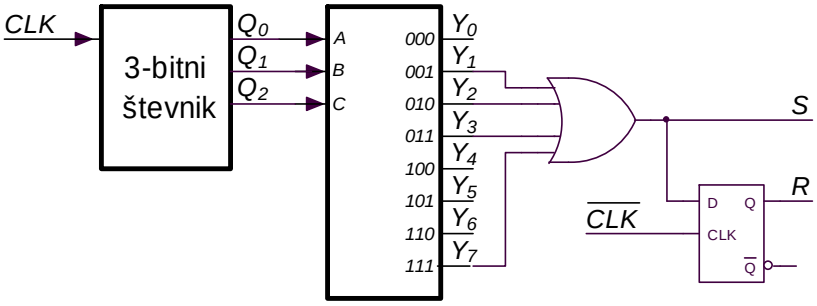
Shema vezja je na sliki 28.2.

Taka realizacija lahko vodi do kompleksnega logičnega vezja za realizacijo signala *S*, kadar je stanj več in signal *S* bolj razgiban. Če se želimo izogniti večjemu številu logičnih vrat dodamo multiplekser 4 v 1 in z njim realiziramo logično funkcijo kot smo to videli na vezju s slike 15.6. Obe verziji vezja povzročata težave, če robovi signalov na izhodih iz števca niso popolnoma poravnani v času. Težavam se ognemo tako, da signal *S* vodimo na D flip-flop, ki ga poganja ura $\overline{CLK}$, zato se vanj vpiše vrednost signala *S* takrat, ko so vsi



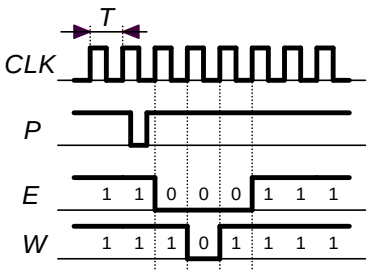

*Slika 28.2: Vezje, ki generira željeni signal S*

prehodni pojavi zaradi neporavnanosti že mimo. Novi signal imenujemo *R*, vezje zanj je narisano na sliki 28.3.




*Slika 28.3: Vezje za generiranje signala S in R*

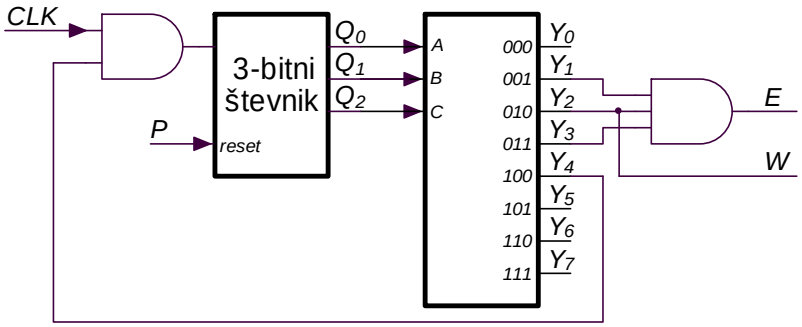
Včasih v digitalni elektroniki potrebujemo signale, ki po določenem zaporedju menjajo vrednosti, vendar njihov potek sproži zunanji prožilni signal, signal pa se po določenem številu izmenjav vrednosti nič in ena ustali na začetni vrednosti, zgled za tak signal je na sliki 28.4. Ob prihodu prožilnega sunka *P* se začne zaporedje vrednosti dveh signalov *E* in *W*, signala sta namenjena prepisovanju vrednosti iz enote v enoto kot je bilo to navedeno v poglavju 16. Ob prihodu signala *P* se najprej spremeni signal *E* in omogoči pošiljanje vrednosti iz enote A na vodilo *X*, nato se spremeni signal *W* in zapiše vrednost z vodila *X* v enoto B, po vpisovanju pa se vrne na izhodiščno vrednost še signal *E* in prekine dejavnost




*Slika 28.4: Zahtevana signala E in W*

enote A.

Vezje, ki generira zahtevano zaporedje je na sliki 28.5. Gre za kombinacijo števnika in selektorja ter logičnih vrat. Logična vrata ne prepuščajo signala ure na vhod števnika kadar je izhodna vrednost $Y_4$ selektorja enaka nič, torej takrat, ko je števnik v stanju štiri. Ker signal ure takrat ne more do števnika, se števnik ustavi. Iz mirovanja ga spravimo s signalom *P*, ki je priključen na vhod »reset« števnika. Števnik zato ob prihodu sunka *P* preide v stanje nič, signal ure lahko pride do števnika in števnik po vrsti menja stanja do takrat, ko spet pride v stanje štiri, kjer se ustavi. Izhodne signale selektorja tokrat kombiniramo z vrati AND in sestavimo zahtevani signal E, signal W pa je kar enak signalu $Y_2$ na izhodu selektorja. Če potrebujemo drugo število stanj v zaporedju, uporabimo namesto signala $Y_4$ drug izhodni signal selektorja. Tudi tu bi lahko uporabili D flip-flop zato, da bi se izognili težavam zaradi slabega prekrivanja robov izhodnih signalov selektorja.



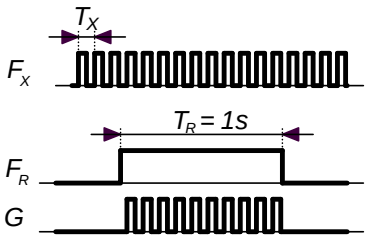

*Slika 28.5: Vezje, ki generira signala E in W s slike 28.4*

V teh vezjih se zdi smiselno uporabiti sinhrone števnike, pri asinhronih bi lahko ne-sočasno preklapljanje izhodov števnika povzročilo zelo kratke impulze v generiranem signalu *S*.

Periodična zaporedja vrednosti signala ali več signalov lahko generiramo s števnikom in dekodiranjem njegovih izhodnih stanj. Pri tem je perioda signala ure za števnik enaka najmanjšemu časovnemu intervalu v iskanem signalu. Vezje je enostavnejše, če del dekodiranja izvedemo s selektorjem ali multiplekserjem. Za generiranje enkratnih zaporedij vrednosti, ki jih sprožimo z zunanjim prožilnim signalom, lahko uporabimo števnik z vhodom za »reset« in logična vrata, ki preprečijo signalu ure dostop do števnika.

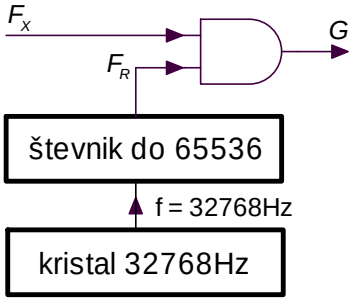
# 29. Zgled – merilnik frekvence

Sestavimo enostaven inštrument za merjenje frekvence signala $F_X$. Potrebujemo števnik SFX, ki eno sekundo šteje prehode signala $F_X$. Sekundo definiramo s oscilatorjem, ki uporablja kremenov kristal, zato je njegova frekvenca znana in stabilna. Ker je frekvenca nihanja kristalnega oscilatorja velika, jo je treba z drugim števnikom zmanjšati. Če uporabimo kristal z resonančno frekvenco 32768Hz (taki so na primer v električnih zapestnih urah), lahko s števnikom z 16 biti (šestnajst-bitni števnik ima 65536 različnih stanj) pridemo do signala $F_R$ s frekvenco 0.5Hz. Polperioda $T_R$ tega signala znaša natanko eno sekundo, kar uporabimo skupaj z vrati AND za definiranje sekundnega izseka $G$ signala $F_X$, kot je to narisano na sliki 29.1. Bločna shema takega sklopa je na sliki 29.2. Tak izsek vodimo na števnik SFX.




*Slika 29.1: Tako izsejemo iz signala $F_X$ eno sekundo trajajoč del*

Na sliki 29.3 je bločna shema predlaganega vezja za merjenje frekvence.

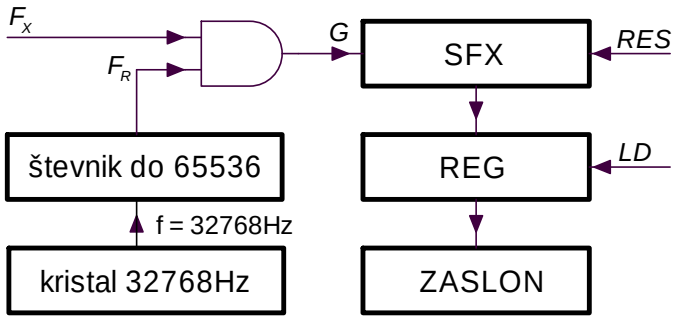



*Slika 29.2: Vezje po sliki 29.1*

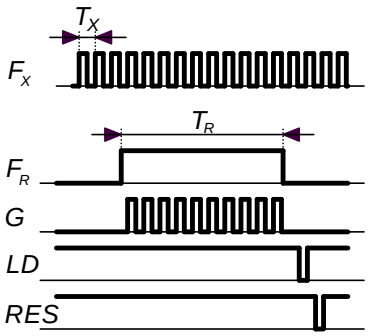
Zdi se smiselno, da po štetju rezultat prepišemo v register REG in ga od tam posredujemo na zaslon. To je primerno zaradi tega, da uporabnik ves čas vidi mirujoči rezultat in ne napredujoče vsebine števnika. Rezultat prepišemo iz števnika v register kadarkoli po meritvi, trenutek prepisa definira kratek sunek na vhodu *LD* registra.

Pred naslednjo meritvijo je potrebno števnik postaviti na izhodiščno vrednost. Pravimo, da števnik »resetiramo«. To se zgodi takrat, ko privedemo signal *RES* števnika na vrednost ena.

Za merilnik frekvence potrebujemo torej dva kontrolna signala *RES* in *LD*. Po meritvi se mora najprej prožiti signal *LD*, za njim pa še *RES*. Na sliki 29.4 je predlagano zaporedje signalov. Signala *LD* in *RES* sta lahko kratka, generira ju na primer vezje s slike 25.8 takrat, ko signal se signal $F_R$ vrne na nič in je štetja konec. Potrebujemo torej še tretji števnik KOS, ki ga držimo v izhodiščnem stanju s signalom $F_R$ med meritvijo, takoj po štetju $F_X$ pa sprostimo. Števnik KOS zato steče skozi stanja do sedem, ko se samodejno

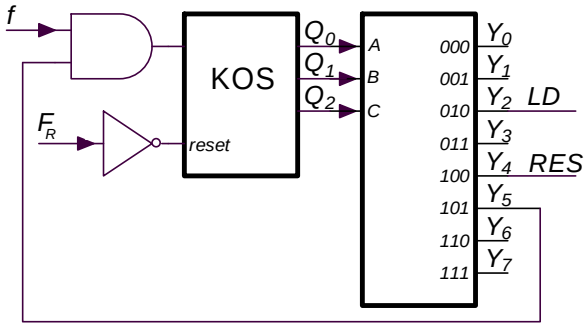



*Slika 29.3: Bločna shema merilnika frekvence z definiranimi krmilnimi signali*




*Slika 29.4: Zaporedje signalov za kontrolo merilnika frekvence*

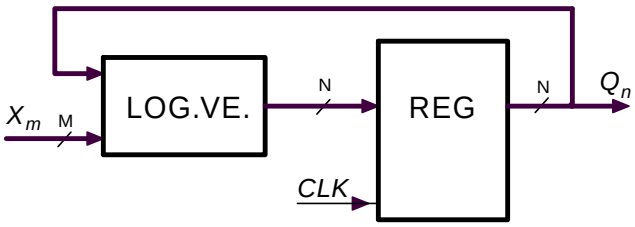
ustavi zaradi povratne vezave med sedmim izhodom selektorja in vrati AND na vhodu v tretji števnik. Izhoda tri in pet selektorja uporabimo za vir signalov za prepis *LD* in *RES*. Shema kontrolnega dela merilnika frekvence je na sliki 29.5.




*Slika 29.5: Shema kontrolnega dela merilnika frekvence; signal f prihaja s kristalnega oscilatorja*

# 30. Sinhroni avtomat

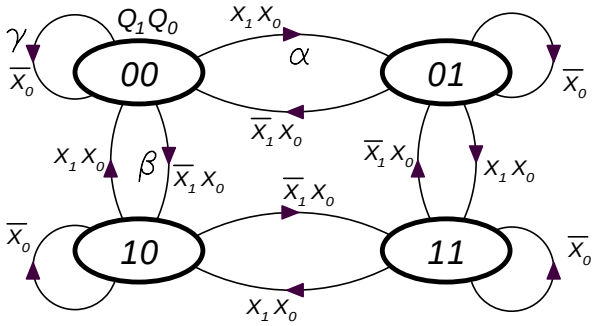
Sinhroni avtomat je nadgradnja sinhronega števnika. Prav tako ga sestavlja register REG in logično vezje LOG.VE., kombinacija vrednosti izhodnih signalov $Q_n$ pa zaznamuje stanje sinhronega avtomata. Do menjave stanja lahko pride le ob prehodu ure *CLK*, njegova blokovna shema je na sliki 30.1. Od sinhronega števnika ga ločijo priključki za vhodne signale $X_m$, ki določajo zaporedje menjave stanj.




*Slika 30.1: Blokovna shema sinhronega avtomata*

Postopek načrtovanja je enak postopku načrtovanja sinhronega števnika, le da moramo tokrat upoštevati še vhodne signale. Najprej narišemo diagram prehodov, iz njega izpišemo tabelo prehodov in iz tabele izluščimo logične funkcije za posamezni vhodni signal v register REG.

Postopek je najlaže razložiti ob zgledu. Vzemimo, da potrebujemo sinhroni avtomat z dvema izhodoma $Q_0$ in $Q_1$. Stanja naj se menjajo tako, kot je prikazano na diagramu prehodov na sliki 30.2. Menjavanje stanj je omogočeno takrat, ko ima vhodna spremenljivka $X_0$ vrednost ena, vrednost vhodne spremenljivke $X_1$ pa naj določa smer menjavanja. Stanju '00' torej sledi stanje '01' v primeru, da ima vhodna spremenljivka $X_0$ vrednost ena in vhodna spremenljivka $X_1$ prav tako vrednost ena, kar označuje puščica α. Istemu stanju naj sledi stanje '10' takrat, kadar ima vhodna spremenljivka $X_0$ vrednost




*Slika 30.2: Diagram prehodov za iskani sinhroni avtomat*

ena in vhodna spremenljivka $X_1$ vrednost nič (puščica β), stanje pa se lahko tudi ohranja, če ima le vhodna spremenljivka $X_0$ vrednost nič ne glede na vrednost vhodne spremenljivke $X_1$ (puščica γ). Prehode iz ostalih stanj bi lahko podobno popisali z besedami, vendar to nalogo prepuščamo bralcu. Diagram prehodov pravzaprav prikazuje posebni števnik, ki lahko napreduje po stanjih v naprej ali nazaj odvisno od vrednosti vhodne spremenljivke $X_1$, v trenutnem stanju pa ga ohranja vrednost spremenljivke $X_0$.

Pri risanju moramo biti pozorni, da se ob puščicah izpisani pogoji ne prekrivajo ali manjkajo. Avtomat ob enaki vrednosti vhodnih spremenljivk ne sme iz trenutnega stanja hkrati prehajati v dve različni stanji. Navesti moramo prehode iz danega stanja v novo stanje za vse možne kombinacije vhodnih spremenljivk.

Iz narisanega diagrama prehodov izpišemo tabelo prehodov na sliki 30.3. Vsak prehod iz diagramu prenesemo v svojo vrstico v tabeli, pri tem vsakokrat navedemo tudi vrednost vhodnih spremenljivk.

| $X_1$ | $X_0$ | $Q_1$ | $Q_0$ | $Q_1^+$ | $Q_0^+$ |
|---|---|---|---|---|---|
| X | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |
| X | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| X | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 |
| X | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 |

*Slika 30.3: Tabela prehodov za diagram prehodov s slike 30.2*

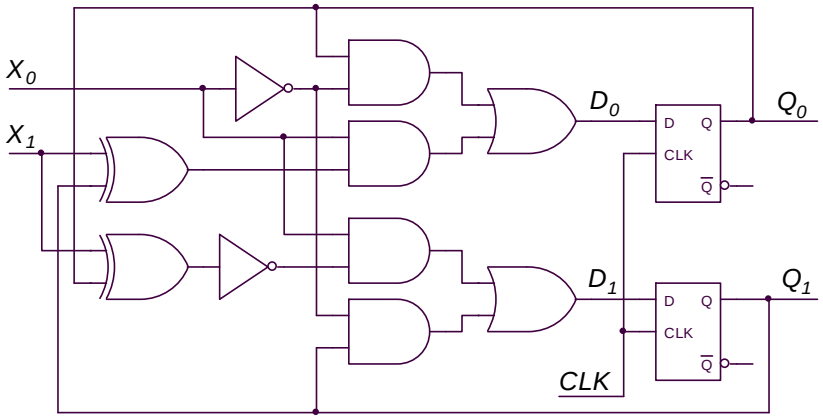
Iz tabele prehodov izpišemo logični enačbi, uporabimo D flip-flopa:

$$Q_0^+ = D_0 = Q_0 \cdot \overline{X_0} + X_0 \cdot (Q_1 \oplus X_1)$$

$$Q_1^+ = D_1 = Q_1 \cdot \overline{X_0} + X_0 \cdot (\overline{Q_0 \oplus X_1})$$

Na podlagi izpisanih enačb sestavimo logično vezje, ki krmili vhode v register, shema vezja je na sliki 30.4.

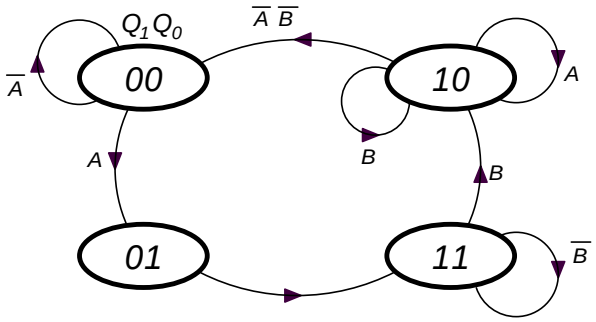
Pri opisanem postopku načrtovanja moramo biti pazljivi, da popišemo vsa možna stanja avtomata, poleg tega morajo biti vsa stanja med sabo različna. Za sinhroni avtomat z N flip-flopi v registru je možnih $2^N$ različnih stanj. Prav tako moramo biti pazljivi pri popisovanju prehodov med stanji, pogoji morajo biti enolično določeni. Če je uporabljenih stanj manj kot $2^N$, je najbolje prehode iz vseh neuporabljenih stanj speljati v eno od uporabljenih stanj, tega imenujemo začetno stanje. Tako preprečimo, da bi se ob vklopu sinhroni avtomat znašel v neuporabljenem stanju in tam obtičal.




*Slika 30.4: Shema vezja sinhronega avtomata po diagramu s slike 30.2*

Sinhroni avtomat torej sestavlja register z N flip-flopi in logično vezje. Sinhroni avtomat je lahko v enem od $2^N$ stanj, prehode med stanji pa določa logično vezje in vrednost vhodnih signalov. Do prehoda pride ob prehodu ure *CLK*. Sinhroni avtomati se največ uporabljajo za generiranje nizov krmilnih signalov, pri tem je zaporedje vrednosti v nizu lahko odvisno od vhodnih spremenljivk.

S sinhronih avtomatom lahko sestavimo vezja, ki v izbranem stanju vztrajajo dokler ni izpolnjen določen pogoj. Sinhroni avtomat, za katerega je diagram prehodov na sliki 30.5, vztraja v stanju '00' dokler ne stisnemo stikala A, takrat preide v preko stanja '01' v stanje '11'. V tem stanju obstane dokler ne pritisnemo stikala B, ko preide v stanje '10' in ostane v tem stanju dokler ne sprostimo obeh stikal. Šele, ko so stikala sproščena, preide nazaj v stanje '00' in igra se lahko ponovi. Seveda do prehodov med stanji prihaja le ob prehodu ure *CLK*. Takšno vezje bi bilo uporabno za preprečitev ponovljenih prehodov




*Slika 30.5: Diagram prehodov za vezje s stikali*

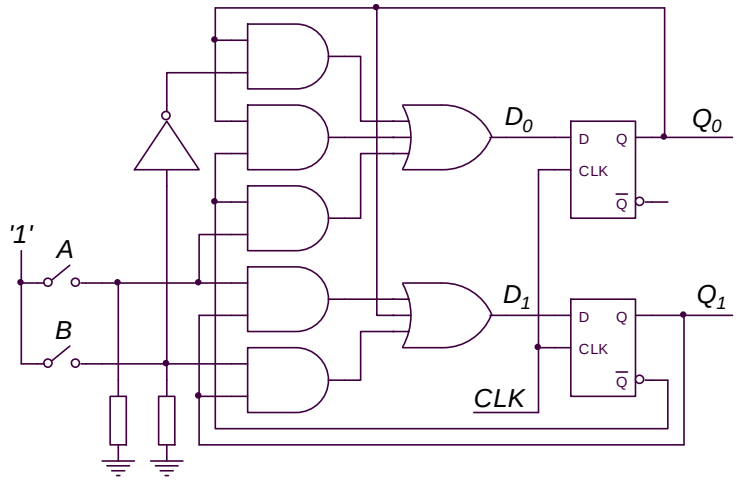
med nič in ena, do katerega pride ob preklapljanju mehanskega stikala.

Iz diagrama prehodov spet izpišemo tabelo prehodov in iz nje izluščimo dve logični enačbi spodaj:

$$Q_0^+ = D_0 = \overline{Q_1} \cdot Q_0 + Q_0 \cdot \bar{B} + \overline{Q_1} \cdot A$$

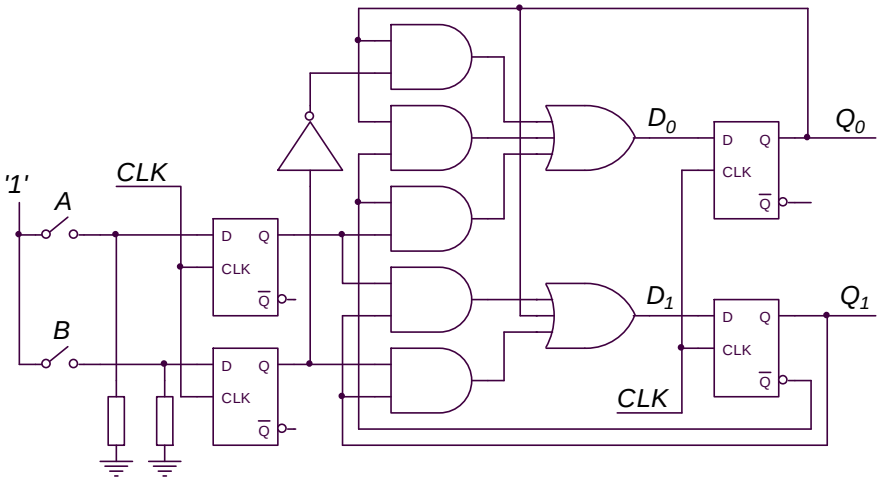
$$Q_1^+ = D_1 = Q_0 + Q_1 \cdot B + Q_1 \cdot A$$

Ustrezno vezje z D flip-flopi je na sliki 30.6.




*Slika 30.6: Vezje za diagram prehodov s slike 30.5*

Že pri registrih smo omenili, da se vrednost signala na vhodu v flip-flop ne sme spreminjati tik pred in tik po prehodu ure CLK, sicer se v register lahko vpiše napačna vrednost. To velja tudi za sinhroni avtomat; vhodni signali morajo biti konstantni dovolj dolgo pred prehodom ure *CLK* tako, da logično vezje pred flip-flopi uspe izračunati pravo vrednost. Ker v splošnem ne moremo zagotoviti, da nekdo pritiska na stikala v ravno pravem trenutku, dodamo pred sinhroni avtomat še en register, ki zaklene vrednost vhodnih spremenljivk ob prehodu ure *CLK* ter tako onemogoči spreminjanje ob neprimernem času, slika 30.7. Prav tako bi morali vezati dodatni register tudi na shemi 30.4.

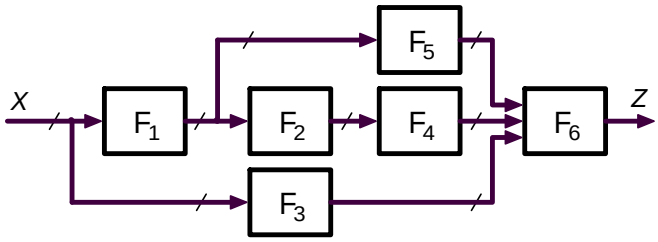



*Slika 30.7: Pred sinhroni avtomat dodamo register in preprečimo spreminjanje vhodnih vrednosti ob nepravem času*

# Zasnova procesorja

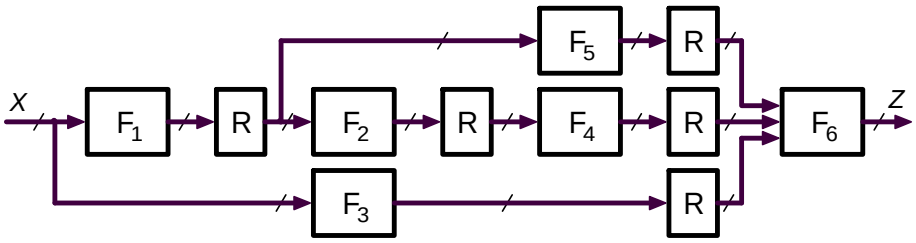
## 31. Računanje z zaporedjem matematičnih operacij

Do rešitve matematičnega problema navadno pridemo po več zaporednih matematičnih operacijah. Logično vezje, ki je namenjeno reševanju danega problema, je zato sestavljeno iz več zaporedno vezanih modulov, zgled je na sliki 31.1. Vsak od modulov opravlja eno matematično operacijo $F_1$ do $F_6$, rezultat *Z* pa je posledica vseh operacij in vhodnih spremenljivk *X*. Taka rešitev zahteva veliko elektronskih elementov in module, ki so prilagojeni za izvajanje posamične operacije, pa še kasnitve v posameznih modulih nam povzročajo preglavice. Rezultat računanja z modulom $F_2$ je odvisen od rezultata računanja modula $F_1$, zato je smiselno začeti z računanjem v modulu $F_2$ šele takrat, ko modul $F_1$ konča računanje. Podobno velja za ostale module. Do končnega rezultata *Z* pridemo šele po vsoti vseh časov, potrebnih za računanje v vsakem modulu. Signali se po različnih poteh med moduli razširjajo različno hitro, vhodna vrednost *X* pa se ne sme spremeniti dokler ni končni rezultat *Z* izračunan, zato utegne biti računanje počasno.




*Slika 31.1: Možna povezava modulov za računanje kompleksne matematične operacije, X so vhodne spremenljivke, Z rezultati. Vse povezave med moduli so vodila*

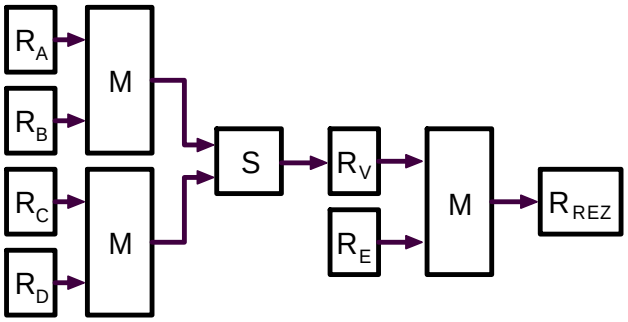
Hitrost računanja povečamo, če med posamezne module vstavimo registre R po sliki 31.2. Takoj, ko je na razpolago rezultat iz posameznega modula, ga




*Slika 31.2: Tako vstavimo registre R in uskladimo čase računanja, vsi registri uporabljajo isti signal ure CLK.*

shranimo v register na njegovem izhodu in vhodni podatki za ta modul se lahko spremenijo, modul pa računa novi vmesni rezultat medtem, ko je naslednji enoti na razpolago rezultat iz registra. Tako tehniko v računalniški terminologiji imenujejo »pipeline« in omogoča največje hitrosti računanja s podatki *X*, ki prihajajo v rednih časovnih intervalih.

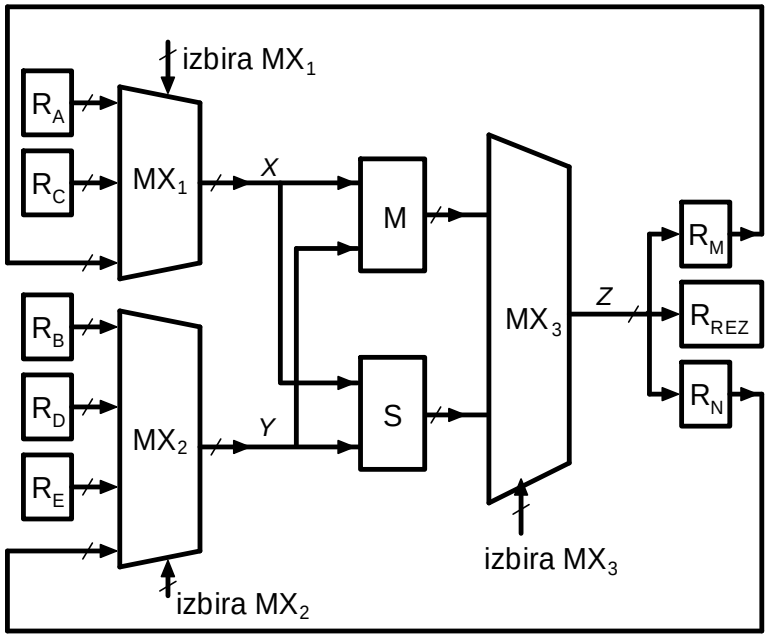
Kadar hitrost računanja ni najbolj pomembna in nam je bolj do univerzalnosti vezja, uberemo drugo pot. Vzemimo, da moramo opraviti matematični postopek: $REZ = ((A \times B) + (C \times D)) \times E$. Po ideji s slike 31.2 potrebujemo vezje s slike 31.3, sestavljajo ga trije množilniki M in seštevalnik S ter register $R_V$ za pomnenje vmesnega rezultata. Viri signalov so registri $R_A$ do $R_E$, rezultat pa se zapiše v register $R_{REZ}$. Vsak od registrov potrebuje pravočasen signal ure *CLK* zato, da se vanj vpiše vrednost v pravem trenutku. Vezje za generiranje signalov *CLK* bi lahko sestavili kot sinhroni avtomat.




*Slika 31.3: Vezje, ki računa uteženo vsoto produktov*

Zdi se potratno sestavljati vezje iz treh množilnikov, če si lahko privoščimo počasnejše računanje. Uporabimo le en množilnik, s katerim najprej zmnožimo števili *A* in *B* ter vmesni rezultat shranimo v register, potem z istim množilnikom zmnožimo še števili *C* in *D* ter produkt shranimo v drug register. V naslednjem koraku vsebini obeh registrov seštejemo, potem pa z

množilnikom vsoto pomnožimo še s številom *E*. Tokrat do rezultata vodi več korakov, vendar potrebujemo manj enot za računanje! Računanje bo torej počasnejše, vezje pa manjše.




*Slika 31.4: Tudi tako lahko računamo, tokrat potrebujemo le po en računski modul, multipleksorje in registre*

Vezje, ki računa enako operacijo, sestavimo iz množilnika M in seštevalnika S in dveh dodatnih vmesnih registrov $R_M$ in $R_N$ ter treh multipleksorjev MX. Shema vezja je na sliki 31.4. Računanje poteka po korakih.

- Najprej z multipleksorjema $MX_1$ in $MX_2$ izberemo argumenta iz registrov $R_A$ in $R_B$, ki se pojavita na vodilih *X* in *Y*. Množilnik M argumenta zmnoži, seštevalnik S pa sešteje, vendar z multipleksorjem $MX_3$ izberemo produkt, ki se pojavi na vodilu *Z* in ga zapišemo v register $R_M$.
- Potem z multipleksorjema $MX_1$ in $MX_2$ izberemo argumeta za množenje iz registrov $R_C$ in $R_D$, ki se pojavita na vodilih *X* in *Y*. Množilnik M argumenta zmnoži, seštevalnik S pa sešteje, vendar z multipleksorjem $MX_3$ izberemo produkt, ki se pojavi na vodilu *Z* in ga zapišemo v register $R_N$.
- V naslednjem koraku z multipleksorjema $MX_1$ in $MX_2$ izberemo argumenta iz registrov $R_M$ in $R_N$, ki se spet pojavita na vodilih *X* in *Y*.

Množilnik M ju zmnoži, seštevalnik S pa izračuna njuno vsoto, vendar z multipleksorjem $MX_3$ izberemo vsoto, ki se pojavi na vodilu *Z* in to zapišemo v register $R_M$.

- V zadnjem koraku z multipleksorjema $MX_1$ in $MX_2$ izberemo argumenta $R_M$ in $R_E$, ki se spet pojavita na vodilih *X* in *Y*. Množilnik M izračuna njun produkt, seštevalnik S pa vsoto, vendar z multipleksorjem $MX_3$ izberemo produkt, ki se pojavi na vodilu Z in tega zapišemo v register $R_{REZ}$.

Vsak korak na poti do rezultata je narejen na podoben način: najprej s signaloma »izbira $MX_1$« in »izbira $MX_2$« izberemo registra z argumentoma, počakamo da se operacije izvršijo, nato s signalom »izbira $MX_3$« izberemo za nas zanimiv rezultat in ga zapišemo v pravi register.

Če bi potrebovali še druge matematične ali logične operacije med argumentoma, bi poleg množilnika M in seštevalnika S dodali še enote, ki opravljajo željene operacije ter razširili multipleksor $MX_3$ tako, da bi dodatne rezultate posredovali na vodilo *Z*.

Če bi bilo zaporedje matematičnih operacij do končnega rezultata drugačno, bi ohranili strukturo vezja s slike 31.4, le zaporedje izbir $MX_1$, $MX_2$ ter $MX_3$ se spremeni.

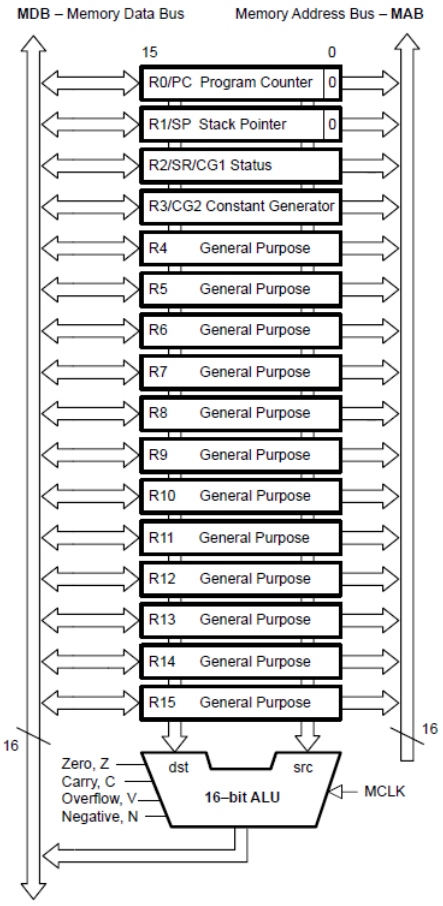
Vezje s slike 31.4 zato predstavlja univerzalno vezje za računanje, pri katerem lahko s programom, ki definira signale $MX_1$ do $MX_3$, določimo zaporedje računskih operacij. Takšno vezje je del procesorja (mikroprocesor, centralna procesna enota, ...) in je osrednji del vsakega digitalnega računalnika, in ga imenujemo izvršilno vezje, saj opravlja operacije med argumenti. Enote, ki opravljajo matematične operacije skupaj z multipleksorjem MX3 imenujemo tudi aritmetično-logična enota (ALU).

Posamezna matematična operacija množenja ali seštevanja je izvršena zelo hitro. V najboljšem primeru je potrebno počakati le nekaj kasnitev skozi logična vrata, ki sestavljajo posamezni blok in rezultat je izračunan, to pa pri sodobnih logičnih vezjih pomeni do nekaj nanosekund takrat, ko uporabimo posamična integrirana vezja z logičnimi vrati, v notranjosti kompleksnega integriranega vezja CPLD ali FPGA pa gre še hitreje. Takoj po izvršitvi trenutne operacije lahko sistem nadaljuje z izvrševanjem naslednje.

Za delovanje procesorja je nujen program, ki ga navadno zapišemo v spomin. Zdi se smiselno, da ostane program zapisan ob procesorju tudi takrat, ko zmanjka napajalne napetosti, zato spomin tipa RAM ni primeren. V procesorskih sistemih za program, ki definira osnovno delovanje procesorja, uporabljamo spomin tipa ROM (»read only memory«), v katerega vsebino zapišemo na poseben način, vsebina pa se ohranja tudi po izklopu. Za osebne

računalnike je lahko programov več in se jih da sproti menjati, spreminjati ali dopisovati. Zapisani so na trdem disku računalnika ali v delovnem spominu (RAM, »random access memory«).

Program je zaporedje navodil, kako preklapljati multipleksorje ter naslovov registrov, kjer so argumenti in kamor naj se rezultati zapišejo. Program v spominu je zapisan v kompaktni obliki, zato je potrebno elektronsko vezje, ki program najprej dekodira v signale za multipleksorje in definira pravilno časovno sosledje krmilnih signalov. Temu vezju pravimo nadzorni del procesorja, saj skrbi za pravilno delovanje izvršilnega dela. Nadzorni del procesorja je izdelan kot sinhroni avtomat, katerega vhodni parametri so koraki programa, izhodni pa signali za krmiljenje multipleksorjev in registrov. Pravimo, da ta avtomat izvaja »mikrokodo«, saj krmili vsak posamezen sklop v izvršilnem delu procesorja. Ura za sinhroni avtomat definira hitrost izvajanja operacij in mora biti usklajena s hitrostjo računanja izvršilnega dela procesorja.




*Slika 31.5: Tipična zgradba mikroprocesorja z ALUin spominsko enoto, MSP430 mikroprocesor*

Za večino mikroprocesorjev lahko program (»software«) poljubno spreminjamo, če le imamo na razpolago orodja za programiranje, strojna oprema mikroprocesorja (»hardware«) pa ostaja ves čas ista.

Na sliki 31.5 je za zgled podana notranja struktura mikroprocesorja MSP430 firme »Texas Instruments«. V procesorju je šestnajst šestnajst-bitnih registrov R0 do R15. Nekaterim registrom je dodeljena specialna naloga, drugi so splošni in namenjeni vmesnim rezultatom. Vsebino vsakega registra lahko privedemo na vhoda aritmetično-logične enote ALU, ki računa s šestnajstimi biti hkrati, tej enoti programsko določimo eno od 35 matematičnih in logičnih operacij.

Rezultat iz ALU lahko vpišemo preko vodila MDB (»Memory Data Bus«) nazaj v katerikoli register ali pa ga posredujemo dodatnim modulom, ki so povezani na vodilo MDB. Naslove teh modulov lahko določa vsebina kateregakoli registra R0 do R15 preko vodila MAB (»Memory Address Bus«). Zaporedje izvajanja operacij in argumente določa program, ki ga zapiše uporabnik. Posamezna ukaz programa se izvede v manj kot mikrosekundi, procesor pa odlikuje majhna poraba električne energije.

Na tržišču je mnogo različnih procesorjev, ki pa vsi vsebujejo zgoraj opisani izvršilni in nadzorni del. V grobem se razlikujejo v hitrosti delovanja, številu registrov in matematičnih operacij ter širini podatkovnih vodil. Posamezni procesorji so optimirani glede na ciljno uporabo. Nekateri porabijo malo električne energije in so primerni za baterijske naprave, drugi so hitri in primerni za osebne računalnike. Nekateri so opremljeni s števniki, analogno-digitalnimi in digitalno-analognimi pretvorniki in so primerni za nadziranje fizikalnih procesov, jedro z ALU in podatkovnim ter programskim spominom ter vodili pa ostaja.

# Zaključek

Smo ob koncu tega zapisa. Elektronska vezja postajajo vedno bolj kompleksna implementacija matematičnih operacij, digitalna elektronika in njen razvoj na področju merjenj in telekomunikacij ter računalništva to jasno kaže.

V pričujočem zapisu smo začeli pri osnovah digitalne elektronike. Spoznali smo osnovne lastnosti gradnikov, sestavljali smo enote za računanje, poznamo pojem vodila in postopke za pošiljanje in prejemanje podatkov po vodilih. Vemo o spominskih enotah in števnikih ter avtomatih, spoznali smo časovne poteke signalov v vezjih. Iz osnovnih gradnikov smo sestavili kompleksnejša vezja za računanje, ki preraščajo v mikroprocesorje. Pri delu nas je vodilo razumevanje delovanja in ideje, ki botrujejo določeni zasnovi in vezju. Kljub temu znanju smo pri digitalni elektroniki le začetniki.

Današnja digitalna elektronika temelji na intenzivni rabi velikih logičnih vezij, kot so na primer mikroprocesorji in FPGA vezja, ki so pred-pripravljeni kompleksni gradniki. Uporabnik s programiranjem določi njihove lastnosti in delovanje, programi pa v obliki enačb definirajo povezave med logičnimi vrati in spominskimi celicami. Žal je v enem semestru premalo časa za take gradnike. Nekaj priložnosti za seznanjanje z njimi bo še pri predmetih »Uporaba mikroprocesorjev« in »Meritve« pa morda še kje.

Za vse tiste, ki bi se želeli z digitalno elektroniko še naprej ukvarjati, priporočam v branje naslednji, že malo starejši knjigi:

- P Horrowitz, W Hill, The Art of Electronics, Cambridge Univerity Press, 1989 in kasnejše izdaje ter
- T C Hayes, P Horrowitz, Student Manual for The Art of Electronics, Cambridge Univerity Press, 1989 in kasnejše izdaje.

Na razpolago je še mnogo drugih knjig s sorodno tematiko. Največ

informacij o sodobnih vezjih je mogoče najti pri proizvajalcih, ki sproti prenavljajo in dopolnjujejo ter ponujajo kompleksna vezja in orodja ter navodila za njihovo uporabo.

Na internetu je mogoče najti podatke in nasvete v zvezi z elektroniko. Naj tukaj navedemo le nekatere, ki so v času pisanja tega zapisa aktualni.

- http://en.wikipedia.org/wiki/Wiki
- http://www.epanorama.net/index.php, članki in lista člankov o digitalni elektroniki
- http://www.nxp.com/#/homepage, proizvajalec elektronskih vezij, prej Philips Semiconductor
- http://www.ti.com/, proizvajalec elektronskih vezij Texas Instruments
- http://www.st.com/stonline/, proizvajalec SGS - Thompson
- http://www.analog.com/en/index.html, proizvajalec Analog Devices
- http://www.microchip.com, proizvajalec mikroprocesorjev Microchip

Pri odkrivanju zakonitosti digitalnega in analognega sveta koristi tudi simulacijski program, ki na osebnem računalniku pokaže odzive vezja. Generično ime za tak program je SPICE, ki je bil osnova mnogim različnim komercialnim verzijam. Ena takih je TINA firme Texas Instruments, osnovna verzija je na razpolago zastonj na naslovu:

- http://focus.ti.com/docs/toolsw/folders/print/tina-ti.html

Mnogo veselja pri delu vam želi

Avtor