OSNOVE LOGIČNIH VEZIJ

Mira Trebar

19. september 2005

Kazalo

Poglavje 1

Uvod

V knjigi so predstavljene osnovne tehnike načrtovanja in gradnje logičnih vezij, ki so sestavni del digitalnih sistemov. Vsak digitalni sistem ima vhode in izhode in je opredeljen s funkcijami za preslikavo vhodov v nove izhode. Opisane so tako praktične tehnike razvoja logičnih vezij, kakor tudi nekateri teoretični problemi s poudarkom na splošnih konceptih v povezavi s trenutnim razvojem tehnologije. Opisi in rešitve logičnih vezij so podani tako, da bralcu omogočajo razumeti osnovne funkcije digitalnih sistemov in mu dajejo toliko znanja, da lahko sam pristopi k razvoju in načrtovanju enostavnega digitalnega vezja.

Digitalna vezja in sistemi v vsakem trenutku zavzamejo eno od dveh diskretnih vrednosti ali stanj, to sta 0 in 1 (ang.: low, high ali ang. false, true), zato govorimo o binarni ali dvo-vrednostni predstavitvi informacije. Logična vrednost 0 ponazarja najnižjo napetost, logična vrednost 1 pa najvišjo napetost v digitalnem vezju. V nekaterih primerih se pojavljajo tudi tristanjski izhodi, ki imajo še tretje neaktivno stanje, ali stanje visoke impedance. Izhod se v tem stanju logično odklopi od linije, ki je na voljo drugim izhodom.

Drugo poglavje obravnava osnovne številske sisteme (desetiški, binarni, oktalni, ...) in njihovo pretvarjanje ter dvojiško računanje. Poglavje zaključimo s kratko predstavitvijo kod, ki se uporabljajo v digitalnih sistemih za predstavitev informacije [2,6,9].

Boolova algebra je kot osnovno matematično orodje za opisovanje logičnih funkcij predstavljena v tretjem poglavju. Njene funkcije povežemo s stikalnimi shemami.

V četrtem poglavju so razložene osnovne logične funkcije, različni zapisi in njihove predstavitve. Tu je posebno poudarjen zapis logičnih funkcij v popolnih normalnih oblikah, kjer so uporabljene osnovne logične operacije konjunkcije, disjunkcije in negacije.

Osnovnim zapisom sledi poenostavljanje zapisov logičnih funkcij v petem poglavju. Predstavljeni sta dve metodi minimizacije, ki se uporabljata za poenostavljanje logičnih vezij. Tu so vključene tudi nepopolne logične funkcije ali funkcije z redundancami, ki s svojimi nedoločenimi vhodnimi kombinacijami omogočajo enostavnejše rešitve.

Šesto poglavje vključuje razširjavo zapisov logičnih funkcij tudi z drugimi operatorji, kot so negacija konjunkcije ali Shefferjev operator, negacija disjunkcije ali Pierceov operator, seštevanje po modulu 2 in ekvivalenca. Obsežne logične funkcije v normalnih oblikah so pretvorjene v večnivojske oblike.

S sedmim poglavjem preidemo iz teorije logičnih funkcij v drugi del, ki opisuje gradnjo kompleksnejših logičnih vezij. Delimo jih v kombinacijska ali odločitvena vezja in sekvenčna vezja.

V okviru kombinacijskih vezij so v sedmem poglavju predstavljena enostavna aritmetična vezja, s katerimi je možno graditi poljubno velike enote za izvajanje aritmetičnih funkcij (seštevanje, odštevanje, množenje, ...). Pri gradnji logičnih vezij poleg logičnih vrat uporabljamo strukturalne in programabilne gradnike, ki so predstavljeni v osmem in

devetem poglavju. Ti gradniki so zelo primerni za izvedbo večjega števila logičnih funkcij.

Sekvenčna vezja v desetem poglavju se začnejo z najmanjšo enoto za shranjevanje informacije, to je pomnilno celico. V nadaljevanju so prikazani postopki gradnje vezij od enostavnih pomnilnih celic, do kompleksnih registrov in števcev.

V enajstem poglavju so predstavljena sekvenčna logična vezja kot avtomati, ki služijo kot krmilne enote v različnih digitalnih sistemih. Na kratko sta opisana Mooreov in Mealyjev avtomat, njuna ekvivalenca, minimizacija ter načrtovanje in realizacija.

Poglavje 2

Predstavitev števil in kod

Običajno želimo pridobljene podatke obdelovati, prenašati in shranjevati, zato jih moramo predstaviti v najbolj primerni obliki. Če smo jih pridobili v diskretnih korakih, potem imamo opraviti z digitalnimi podatki in jih predstavljamo z znaki, številkami ali črkami [1, 4, 5].

2.1 Številski sistemi

Za predstavitev podatkov v digitalnih sistemih se uporabljajo različni številski sistemi: desetiški, dvojiški ali binarni, oktalni in heksadecimalni. Število N je definirano s pozicijskim zapisom tako, da je cifra v sekvenci množena s potenco osnove ustreznega mesta v številu. Celi del števila ima pozitivne potence, ulomljeni del števila pa negativne potence. Število N je v splošnem zapisu podano kot

$$\begin{aligned} N &= b_{n-1} * r^{n-1} + b_{n-2} * r^{n-2} + ... + b_0 * r^0 + \\ &+ b_{-1} * r^{-1} + ... + b_{-p+1} * r^{-p+1} + b_{-p} * r^{-p}, \end{aligned} \tag{2.1}$$

kjer so: N - število, b_i - koeficienti v zapisu števila, r - osnova številskega sistema, n - število mest celega dela števila in p - število mest ulomljenega dela. Tako je število N v pozicijskem zapisu podano kot zaporedje koeficientov

$$N = b_{n-1}b_{n-2}...b_1b_0, b_{-1}b_{-2}...b_{p-1}b_{-p} \ . \tag{2.2}$$

Desetiški številski sistem

Desetiški številski sistem je določen z osnovo r=10 in koeficienti b_i, ki so predstavljeni s ciframi 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Zapis desetiškega števila N je podan kot

$$N = b_{n-1} * 10^{n-1} + ... + b_0 * 10^0 + b_{-1} * 10^{-1} + ... + b_{-p} * 10^{-p}. \tag{2.3}$$

Primer: Pozicijski zapis desetiških števil:
$3428_{10} = 3 * 10^3 + 4 * 10^2 + 2 * 10^1 + 8 * 10^0$
$1,325_{10} = 1 * 10^0 + 3 * 10^{-1} + 2 * 10^{-2} + 5 * 10^{-3} = 1 + 0,3 + 0,02 + 0,005$

Dvojiški številski sistem

V digitalni logiki je uporabljen dvojiški številski sistem z osnovo r=2 in koeficienti b_i, ki imajo dve vrednosti: 0, 1. Število N je v dvojiškem sistemu predstavljeno kot

$$N = b_{n-1} * 2^{n-1} + ... + b_1 * 2^1 + b_0 * 2^0 + b_{-1} * 2^{-1} + ... + b_{-p} * 2^{-p}. \tag{2.4}$$

Primer: Pozicijski zapis dvojiških števil in pretvorba v desetiško število:
$11101_2 = 1 * 2^4 + 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 16 + 8 + 4 + 1 = 29_{10}$
$1,11_2 = 1 * 2^0 + 1 * 2^{-1} + 1 * 2^{-2} = 1 + 0,5 + 0,25 = 1,75_{10}$

Osmiški številski sistem

Osmiški (oktalni) številski sistem ima osnovo r=8 in koeficiente b_i, ki so predstavljeni s ciframi 0, 1, 2, 3, 4, 5, 6, 7. Število N je v osmiškem sistemu podano kot

$$N = b_{n-1} * 8^{n-1} + ... + b_1 * 8^1 + b_0 * 8^0 + b_{-1} * 8^{-1} + ... + b_{-p} * 8^{-p}. \quad (2.5)$$

Primer: Pozicijski zapis osmiških števil in pretvorba v desetiško število:
$137_8 = 1 * 8^2 + 3 * 8^1 + 7 * 8^0 = 1 * 64 + 3 * 8 + 7 * 1 = 64 + 24 + 7 = 95_{10}$
$25,37_8 = 2 * 8^1 + 5 * 8^0 + 3 * 8^{-1} + 7 * 8^{-2} = 2 * 8 + 5 * 1 + 3 * 0,125 + 7 * 0,016 = 21,487_{10}$

Šestnajstiški številski sistem

Šestnajstiški (heksadecimalni) številski sistem ima osnovo r=16 in koeficiente b_i, ki so predstavljeni s ciframi 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ter črkami A=10, B=11, C=12, D=13, E=14, F=15. Število N je v šestnajstiškem sistemu podano kot

$$N = b_{n-1} * 16^{n-1} + ... + b_0 * 16^0 + b_{-1} * 16^{-1} + ... + b_{-p} * 16^{-p} \quad (2.6)$$

Primer: Pozicijski zapis šestnajstiških števil in pretvorba v desetiško število:
$3A9_{16} = 3 * 16^2 + A * 16^1 + 9 * 16^0 = 3 * 256 + 10 * 16 + 9 * 1 = 937_{10}$
$3A,C_{16} = 3 * 16^1 + A * 16^0 + C * 16^{-1} = 3 * 16 + 10 * 1 + \frac{12}{16} = 58,75_{10}$

2.2 Pretvarjanje števil

Pretvorba števila iz desetiškega v poljuben številski sistem

Pri pretvarjanju števil iz desetiškega številskega sistema v drug številski sistem uporabimo Hornerjevo metodo. Celi del števila in ulomljeni del števila pretvarjamo ločeno.

1) Ponavljamo postopek deljenja celega dela števila z novo osnovo in uporabimo sekvenco ostankov za določanje novega števila.

2) Ponavljamo postopek množenja ulomljenega dela števila z novo osnovo in uporabimo sekvenco celih delov zmnožka za zapis ulomljenega dela novega števila.

Primer: Pretvorba desetiškega števila $N = 14,25_{10}$ v dvojiško število.

Celi del števila 14_{10} pretvorimo v dvojiško obliko s ponavljanjem deljenja z 2, vse dokler je deljenec različen od nič. Ostanki deljenja so koeficienti celega dela dvojiškega števila.

Deljenje z 2	Ostanek
14:2=7	$0 = b_0$
7:2=3	$1 = b_1$
3:2=1	$1 = b_2$
1:2=0	$1 = b_3$

Pretvorimo ulomljeni del števila $0,25_{10}$ v dvojiško obliko tako, da ponavljamo množenje z

Množenje z 2	Celi del
0,25*2=0,5	$0 = b_{-1}$
0,50*2=1,0	$1 = b_{-2}$

2, dokler ni ulomljeni del zmnožka enak 0. V vsakem koraku množenja je celi del zmnožka koeficient ulomljenega dela dvojiškega števila, ki ga odštejemo od zmnožka.
Postopek množenja je končan po drugem koraku, ker je ulomljeni del produkta po izločitvi celega dela zmnožka enak 0. Desetiško število $N = 14,25_{10}$ je v dvojiškem pozicijskem zapisu predstavljeno kot $N = b_3b_2b_1b_0, b_{-1}b_{-2} = 1110,01_2$.

Primer: Pretvorba desetiškega števila $N = 19,379_{10}$ v osmiško število s štirimi mesti ulomljenega dela.

Celi del števila 19_{10} pretvorimo v osmiško obliko s ponavljanjem deljenja z 8, vse dokler je deljenec različen od nič. Ostanki deljenja so koeficienti celega dela osmiškega števila.

Deljenje z 8	Ostanek
19:8=2	$3 = b_0$
2:8=0	$2 = b_1$

Pretvorimo ulomljeni del števila $0,379_{10}$ v osmiško število s ponavljanjem množenja z 8, dokler ne izračunamo štiri osmiške koeficiente. V vsakem koraku množenja je celi del zmnožka koeficient ulomljenega dela števila in ga odštejemo od zmnožka.

Množenje z 8	Celi del
0,379*8=3,032	$3 = b_{-1}$
0,032*8=0,256	$0 = b_{-2}$
0,256*8=2,048	$2 = b_{-3}$
0,048*8=0,384	$0 = b_{-4}$

Desetiško število $N = 19,379_{10}$ je v osmiškem pozicijskem zapisu predstavljeno kot $N = b_1b_0, b_{-1}b_{-2}b_{-3}b_{-4} = 23,3020_8$.

Primer: Pretvorba desetiškega števila $N = 172,379_{10}$ v šestnajstiško število s tremi mesti ulomljenega dela.

Celi del števila 172_{10} pretvorimo v šestnajstiško obliko s ponavljanjem deljenja s 16, vse dokler je deljenec različen od nič. Ostanki deljenja so koeficienti celega dela šestnajstiškega števila.

Deljenje s 16	Ostanek
$172 : 16 = 10$	$C = b_0$
$10 : 16 = 0$	$A = b_1$

Pretvorimo ulomljeni del števila $0,379_{10}$ v šestnajstiško obliko tako, da ponavljamo množenje s 16, dokler ne izračunamo tri šestnajstiške koeficiente. V vsakem koraku množenja je celi del zmnožka koeficient ulomljenega dela števila in ga odštejemo od zmnožka.
Desetiško število $N = 19,379_{10}$ je v šestnajstiškem pozicijskem zapisu predstavljeno kot $N = b_1b_0, b_{-1}b_{-2}b_{-3} = AC,610_{16}$.

Množenje s 16	Celi del
0,379*16=6,064	$6 = b_{-1}$
0,064*16=1,024	$1 = b_{-2}$
0,024*16=0,348	$0 = b_{-3}$

Pretvorba števila iz osmiškega v dvojiški številski sistem in obratno

Za zapis osmiških števil od 0 do 7 imamo tri bite v dvojiškem zapisu ($2^3 = 8$).

Dvojiški zapis	Osmiško število
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Vsak koeficient osmiškega števila posebej pretvorimo v 3-mestni dvojiški zapis in dobimo dvojiško kodo. Če imamo po pretvorbi na levi strani eno ali več ničel, jih v dvojiškem zapisu ne upoštevamo.

Primer: Pretvorite osmiško število $N = 137_8$ v dvojiško število.

$$N = 137_8 = (001)(011)(111) = 1011111_2 \ .$$

Vsako dvojiško število pretvorimo v osmiško število tako, da ga razstavimo v 3-mestno obliko od desne proti levi in izpišemo osmiške cifre v enakem vrstnem redu.

Primer: Pretvorite dvojiško število $N = 1011010011_2$ v osmiško število.

$$N = 1011010011_2 = (001)(011)(010)(011) = 1323_8 \ .$$

Pretvorba števila iz šestnajstiškega v dvojiški sistem in obratno

Za zapis vsakega koeficienta šestnajstiškega števila potrebujemo štiri bite v dvojiškem zapisu ($2^4 = 16$).
Vsak koeficient šestnajstiškega števila posebej pretvorimo v 4-mestni dvojiški zapis in dobimo dvojiško kodo. Če imamo po pretvorbi na levi strani eno ali več ničel, jih v dvojiškem zapisu ne upoštevamo.

Primer: Pretvorite šestnajstiško število $N = 3A9_{16}$ v dvojiško število.

$$N = 3A9_{16} = (0011)(1010)(1001) = 1110101001_2$$

Vsako dvojiško število pretvorimo v šestnajstiško tako, da ga razstavimo v 4-mestni zapis od desne proti levi in izpišemo šestnajstiške koeficiente v ustreznem vrstnem redu.

Primer: Pretvorite dvojiško število $N = 10101011011_2$ v šestnajstiško število.

$$N = 10101011011_2 = (0101)(0101)(1011) = 95B_{16}$$

Dvojiški zapis	Šestnajstiško število
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	*A*
1011	*B*
1100	*C*
1101	*D*
1110	*E*
1111	*F*

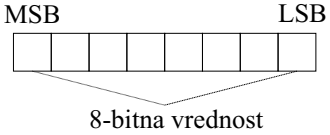
2.3 Dvojiško računanje

Pri dvojiškem računanju bomo spoznali operaciji seštevanja in odštevanja. Najmanjša enota računanja je **bit**, kjer izvedemo operacijo na enomestnem dvojiškem številu. Ostale enote računanja so mnogokratnik števila 2. Uporabljamo skupino 4 bitov - nibble, skupino 8 bitov - bajt, skupino 16 bitov - beseda, skupino 32 bitov - dvojna beseda, itd. V večmestnem zapisu števila vedno določimo:

- najmanj pomemben bit ali bit z najmanjšo utežjo (**LSB**-Least Significant Bit), ki se nahaja na desni strani zapisa števila in najmanj prispeva k vrednosti števila,
- najbolj pomemben bit ali bit z največjo utežjo (**MSB**-Most Significant Bit), ki se nahaja najbolj levo v zapisu in največ prispeva k vrednosti števila.

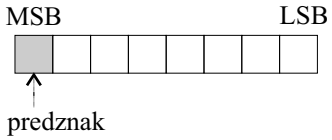
Števila so v digitalnih sistemih predstavljena kot niz bitov, ki jih obravnavamo na dva načina:

1) **Števila brez predznaka (unsigned)** so nenegativna števila. Za n-bitno število imamo vrednosti med 0 in $2^n - 1$ (8-bitno število ima vrednosti od 0 do $2^8 - 1 = 255$ (Slika 2.1), 16-bitno število ima vrednosti od 0 do $2^{16} - 1 = 65535$, itd.).



Slika 2.1: 8-bitno dvojiško število brez predznaka

2) **Števila s predznakom (signed)** so predstavljena na štiri načine: s predznakom in velikostjo, z odmikom, z eniškim komplementom in z dvojiškim komplementom (Slika 2.2). Pri n-bitnih številih s predznakom imamo vrednosti zapisane od -2^n do $2^n - 1$ (8-bitna števila so od -128 do +127, 16-bitna števila so od -32768 do +32767, itd.).



Slika 2.2: 8-bitno dvojiško število s predznakom

- **Predznak in velikost** - najbolj pomemben bit števila b_{n-1} je predznak (ang. sign), ostali biti pa predstavljajo vrednost števila. Pozitivna števila imajo predznak 0, negativna števila pa predznak 1. Pri 4-bitnem številu je maksimalno pozitivno število $N = +7_{10}$ podano kot $N = 0111_2$ in negativno $N = -7_{10}$ kot $N = 1111_2$. Število nič ima dve predstavitvi ($+0_{10} = 0000_2$, $-0_{10} = 1000_2$), kar matematično nima posebnega vpliva, lahko pa zaplete logična vezja.
- **Odmik** - k številu se najprej prišteje konstanta, s katero se zagotovi pozitivno vrednost. Število se nato predstavi kot pozitivno. Konstanti pravimo odmik in je običajno enak 2^{n-1}. Število $N = +7_{10}$ je z odmikom predstavljeno kot $N = 1111_2$, število $N = -7_{10}$ pa kot $N = 0001_2$. Število nič ima eno samo vrednost $0_{10} = 1000_2$.
- **Eniški komplement (1'K)** - pozitivna števila so predstavljena enako kot pri številih s predznakom in velikostjo, negativna pa določimo z negacijo vseh bitov, vključno s predznakom. Za število N dolžine n bitov dobimo predstavitev v eniškem komplementu po enačbi $\overline{N} = (2^n - 1) - N$. Pozitivna števila imajo predznak 0, negativna števila pa predznak 1. Število $N = +7_{10}$ je predstavljeno kot $N = 0111_2$, $N = -7_{10}$ pa kot $\overline{N} = 1000_2$. Število nič ima dve predstavitvi ($+0_{10} = 0000_2$, $-0_{10} = 1111_2$).

  1'K - eniški komplement dvojiškega števila je število, ki mu zamenjamo ničle z enicami in enice z ničlami (negiramo posamezno vrednost števila).

N	1'K(N)
00101101_2	11010010_2
01001100_2	10110011_2
10000101_2	01111010_2

- **Dvojiški komplement (2'K)** - imamo podobno shemo kot pri številih v eniškem komplementu, le da je tu samo ena predstavitev za nič ($0_{10} = 0000_2$). Za število N dolžine n bitov dobimo predstavitev v dvojiškem komplementu $N^* = 2^n - N$. Pozitivna števila imajo predznak 0, negativna števila pa predznak 1. Najmanjše negativno število je $N = -2^{n-1}$. Pozitivno število $N = +7_{10}$ je predstavljeno kot $N = 0111_2$, negativno število $N = -7_{10}$ pa kot $N^* = 1001_2$.

  Dvojiški komplement števila (2'K) dobimo tako, da ga najprej pretvorimo v eniški komplement (1'K) in mu prištejemo konstanto 1: 2'K(N)=1'K(N)+1.

N	2'K(N)	= 1'K(N)+1
00101101_2	$11010010_2 + 1_2$	$= 11010011_2$
01001100_2	$10110011_2 + 1_2$	$= 10110100_2$
00000101_2	$11111010_2 + 1_2$	$= 11111011_2$

2.3.1 Seštevanje dvojiških števil

Oglejmo si seštevanje dvojiških števil brez predznaka in s predznakom za števila predstavljena v dvojiškem komplementu. Seštevanje poteka po stolpcih od desne proti levi. Če je vsota večja ali enaka bazi, potem pride do prenosa, ki ga prištejemo naslednjemu stolpcu. Dvojiški števili X in Y, ki ju želimo sešteti, morata biti desno poravnani na najmanj pomembnih (LSB) bitih. Prenos (ang. carry) c_i je enak 1 pri seštevanju dveh ali treh enic v stolpcu. Zadnji prenos seštevanja se ne upošteva v rezultatu.

Primer: Seštevanje števil brez predznaka $X = 130_{10}$ in $Y = 23_{10}$ v dvojiškem sistemu, ki daje vsoto $Z = X + Y = 153_{10}$.

Tabela 2.1: Seštevanje števil brez predznaka

X			1	0	0	0	0	0	1	0			130
Y	+		0	0	0	1	0	1	1	1		+	23
	+	0	0	0	0	0	1	1	0		$c_i - prenos$		
Z			1	0	0	1	1	0	0	1			153

Primer: Seštevanje števil s predznakom $X = 30_{10}$ in $Y = -23_{10}$ v dvojiškem sistemu, ki daje vsoto $Z = X + Y = 30_{10} + (-23_{10}) = 7_{10}$.

Tabela 2.2: Seštevanje števil s predznakom

X			0		0	0	1	1	1	1	0		30
Y	+		1		1	1	0	1	0	0	1	+	-23
	+	1	1		1	1	1	0	0	0	$c_i - prenos$		
Z			0		0	0	0	0	1	1	1		7

2.3.2 Odštevanje dvojiških števil

Odštevanje dveh števil s predznakom X in Y lahko izvedemo kot operacijo odštevanja $X - Y$ ali kot operacijo seštevanja $X + (-Y)$. V prvem primeru podani števili samo odštejemo, v drugem pa pretvorimo število Y v dvojiški komplement ($2'K$) in ga nato prištejemo k številu X.

Desno poravnani dvojiški števili odštevamo po stolpcih od desne proti levi. Pri odštevanju 0-1, 0-1-1 ali 1-1-1 na i-tem mestu upoštevamo sposodek (ang. borrow) $b_i = 1$, ki ga odštejemo na višjem mestu.

Primer: Odštevanje števil s predznakom $X = 33_{10}$, $Y = 21_{10}$, ki daje razliko $D = X - Y = 33_{10} - 21_{10} = 12_{10}$

Tabela 2.3: Odštevanje dvojiških števil

X			0		0	1	0	0	0	0	1		33
Y	−		0		0	0	1	0	1	0	1	−	21
	−	0	0		0	1	1	1	0	0	b_i-sposodek		
D			0		0	0	0	1	1	0	0		12

Odštevanje dvojiških števil izvedeno s seštevanjem v dvojiškem komplementu

Ker je v praksi odštevanje izvedeno z isto logiko kot seštevanje, je uporabljena komplementarna aritmetika. Razlika števil $X - Y = X + (-Y)$ je izvedena s seštevanjem števila X in dvojiškega komplementa: 2'K$(Y) = -Y =$ 1'K(Y)+1. Zadnji prenos seštevanja v rezultatu ne upoštevamo.

Primer: Odštevanje števil s predznakom $X = 33_{10}$, $Y = 21_{10}$ v dvojiškem sistemu, kjer je razlika izvedena s seštevanjem $D = X + (-Y) = 12_{10}$.

Najprej zapišemo število $Y = 21_{10} = 00010101_2$ v dvojiškem komplementu in dobimo $-Y = 11101010_2 + 1_2 = 11101011_2$ ter ga seštejemo z $X = 00100001_2$.

Tabela 2.4: Odštevanje izvedeno s seštevanjem v 2'K

X			0		0	1	0	0	0	0	1			33
$-Y$	+		1		1	1	0	1	0	1	1		+	-21
	+	1	1		1	0	0	0	1	1		c_i-prenos		
D			0		0	0	0	1	1	0	0			12

Prekoračitev

Prekoračitev (ang. overflow) imamo pri seštevanju števil, če je rezultat seštevanja dveh pozitivnih števil negativno število, ali če seštevamo dve negativni števili in dobimo pozitiven rezultat. Oglejmo si primer seštevanja $(5_{10}+3_{10})$ in odštevanja $(-7_{10}-2_{10})$ za 4-bitna števila, kjer imamo v obeh primerih prekoračitev, ki jo ugotovimo s primerjavo zadnjih dveh prenosov. Če sta prenosa različna **01** ali **10** je prišlo do prekoračitve pri dvojiškem računanju.

Tabela 2.5: Prekoračitev

5				0	1	0	1		-7				1	0	0	1
3		+		0	0	1	1		-2		+		1	1	1	0
		+	**0**	**1**	1	1					+	**1**	**0**	0	0	
-8				1	0	0	0		7				0	1	1	1

2.4 Kode

Koda je pravilo, ki ga uporabljamo za prirejanje številk, črk in drugih znakov. V digitalnih sistemih najpogosteje uporabljamo BCD kodo, Grayevo kodo in ASCII kodo.

Grayeva koda

Grayeva koda je pogosto uporabljena za kodiranje osmiških ali šestnajstiških števil, ni pa primerna pri operacijah računanja. Za Grayevo kodo je značilno, da se dvema zaporednima kodama spremeni vrednost samo na enem bitu. 1-bitna Grayeva koda je enaka dvojiški kodi, ker imamo samo en bit z dvema vrednostima 0 in 1.

Dvojiška koda	1-bitna Grayeva koda
0	0
1	1

Dvo-bitna Grayeva koda je zrcalna slika eno-bitne kode na zadnjem mestu z dodano ničlo na zgornji polovici zrcalne osi in enico na spodnji polovici osi pri najbolj pomembnem bitu. Zrcalna os je na polovici tabele.

Dvojiška koda	2-bitna Grayeva koda
00	0 0
01	0 1
10	1 1
11	1 0

Tri-bitna Grayeva koda je zrcalna slika dvo-bitne kode na zadnjih dveh mestih z dodano ničlo na zgornji polovici zrcalne osi in enico na spodnji polovici osi na najbolj pomembnem bitu.

Dvojiška koda	3-bitna Grayeva koda
000	0 00
001	0 01
010	0 11
011	0 10
100	1 10
101	1 11
110	1 01
111	1 00

Štiri-bitna Grayeva koda je zrcalna slika tri-bitne kode z dodano ničlo na zgornji polovici zrcalne osi in enico na spodnji polovici osi.

Dvojiška koda	4-bitna Grayeva koda
0000	0 000
0001	0 001
0010	0 011
0011	0 010
0100	0 110
0101	0 111
0110	0 101
0111	0 100
1000	1 100
1001	1 101
1010	1 111
1011	1 110
1100	1 010
1101	1 011
1110	1 001
1111	1 000

Proces generiranja Grayeve kode v tabeli se po opisanem pravilu nadaljuje za poljubno število bitov.

Pretvorba n-bitne dvojiške kode v Grayevo kodo

Pri pretvorbi n-bitne dvojiške kode (DK) v Grayevo kodo (GK) imamo dva postopka. Najbolj pomemben bit je enak za obe kodi ($b_{n-1} = g_{n-1}$).

P 1) Beremo dvojiško kodo od leve proti desni. Sprememba dveh sosednih bitov iz 0 v 1 ali iz 1 v 0 pri dvojiški kodi generira 1 v Grayevi kodi. Enakost dveh sosednih bitov generira 0 v Grayevi kodi.

P 2) Dvojiško kodo za eno mesto zamaknjeno desno prištejemo k dani kodi. Vsoti dveh sosednjih bitov 0+1=1 in 1+0=1 generirata 1 v Grayevi kodi, vsoti 0+0=0 in 1+1=0 pa generirata 0 v Grayevi kodi. Pri seštevanju ne upoštevamo prenosa.

Primer: Pretvorba 10-bitne dvojiške kode v Grayevo kodo.

Tabela 2.6: Pretvorba dvojiške kode v Grayevo kodo

					P 1)											P 2)					
DK:	0	1	1	0	1	0	0	1	1	0			0	1	1	0	1	0	0	1	1	0
												+		0	1	1	0	1	0	0	1	1
		s	e	s	s	s	e	s	e	s												
GK:	0	1	0	1	1	1	0	1	0	1			0	1	0	1	1	1	0	1	0	1

Oznaka 's' pri pretvorbi pomeni spremembo vrednosti pri dveh zaporednih bitih, 'e' pa enako vrednost dveh zaporednih bitov.

Pretvorba n-bitne Grayeve kode v dvojiško kodo

Pri pretvorbi n-bitne Grayeve kode (GK) v dvojiško kodo (DK) imamo dva postopka.Najbolj pomemben bit je enak za obe kodi ($g_{n-1} = b_{n-1}$).

P 1) Beremo Grayevo kodo od leve proti desni. Enica v Grayevi kodi pomeni, da se mora naslednji bit dvojiške kode spremeniti glede na predhodnega (0 v 1 in 1 v 0), ničla v Grayevi kodi pa pomeni, da je naslednji dvojiški znak enak predhodnemu.

P 2) Dvojiško kodo izračunamo tako, da prepišemo MSB bit Grayeve kode na prvo mesto dvojiške kode, ga nato prištejemo k naslednjemu bitu Grayeve kode (0+1 = 1 in 1+0 = 1, sicer pa je vsota 0) in dobimo drugi bit dvojiške kode, ki ga prištejemo tretjemu bitu Grayeve kode in dobimo tretji bit dvojiške kode ter ta postopek ponavljamo do zadnjega bita Grayeve kode.

Primer: Pretvorba 10-bitne Grayeve kode v dvojiško kodo.

Tabela 2.7: Pretvorba: Grayeva koda v dvojiško kodo

					P 1)											P 2)					
GK:	1	0	0	0	1	1	0	1	0	1			1	0	0	0	1	1	0	1	0	1
												+		1	1	1	1	0	1	1	0	0
		e	e	e	s	s	e	s	e	s												
DK:	1	1	1	1	0	1	1	0	0	1			1	1	1	1	0	1	1	0	0	1

BCD koda (Binary Coded Decimal)

BCD koda predstavlja desetiške cifre zapisane v dvojiški obliki. Vsaka cifra od 0,1,...,9 je zapisana s 4-bitnim dvojiškim ekvivalentom.

BCD	Desetiško število
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9

Zapis desetiških števil v BCD kodi je pretvorba vsake desetiške cifre v 4-bitno kodo.

Desetiško število	BCD				
150	0001	0101	0000		
235,8	0010	0011	0101	,	1000

ASCII koda (American Standard Code for Information Interchange)

Vsi podatki shranjeni in procesirani v računalniku niso numerični, vendar so zaradi prednosti dvojiškega sistema shranjeni v dvojiški obliki. ASCII koda je 7-bitna in vključuje poleg števil še črke in ostale znake na tipkovnici. Celotno tabelo ASCII kode s 127 znaki najdemo v številnih priročnikih, zato si oglejmo samo nekaj primerov predstavljenih v ASCII kodi. Omenimo še to, da so cifre 0, 1,..., 9 na prvih treh mestih kodirane z 011 pred dvojiško kodo ustreznega simbola, velike črke abecede z 100, male črke z 110, itd.

Simboli	ASCII koda	Šestnajstiška koda
0	011 0000	30
1	011 0001	31
2	011 0010	32
A	100 0001	41
B	100 0010	42
C	100 0011	43
a	110 0001	61
b	110 0010	62
c	110 0011	63
(	010 1000	28
+	010 1011	2B
....		

2.5 Vaje

VAJA 2.5.1:

Pretvorite desetiško število $N = 145_{10}$ v dvojiško število.
Celi del števila pretvorimo z deljenjem števila z 2 (Tabela 2.8).

Tabela 2.8: Pretvorba števila - $N = 145_{10}$

Deljenje z 2	Ostanek
145:2=72	$1 = b_0$
72:2=36	$0 = b_1$
36:2=18	$0 = b_2$
18:2=9	$0 = b_3$
9:2=4	$1 = b_4$
4:2=2	$0 = b_5$
2:2=1	$0 = b_6$
1:2=0	$1 = b_7$

Desetiško število $N = 145_{10}$ pretvorjeno v dvojiško število je $N = 10010001_2$.

VAJA 2.5.2:

Pretvorite desetiško število $N = 7,29_{10}$ v dvojiško število z osmimi mesti ulomljenega dela.
Celi del števila pretvorimo z deljenjem števila z 2 (Tabela 2.9).

Tabela 2.9: Pretvorba celega dela števila - $N = 7_{10}$

Deljenje z 2	Ostanek
7:2=3	$1 = b_0$
3:2=1	$1 = b_1$
1:2=0	$1 = b_2$

Ulomljeni del števila pretvorimo z množenjem števila z 2 (Tabela 2.10). Postopek računanja ulomljenega dela prenehamo pri koeficientu b_{-8}.

Tabela 2.10: Pretvorba ulomljenega dela števila - $N = 0,29_{10}$

Množenje z 2	Celi del
0,29*2=0,58	$0 = b_{-1}$
0,58*2=1,16	$1 = b_{-2}$
0,16*2=0,32	$0 = b_{-3}$
0,32*2=0,64	$0 = b_{-4}$
0,64*2=1,28	$1 = b_{-5}$
0,28*2=0,56	$0 = b_{-6}$
0,56*2=1,12	$1 = b_{-7}$
0,12*2=0,24	$0 = b_{-8}$

Desetiško število $N = 7,29_{10}$ je pretvorjeno v dvojiško število $N = 111,01001010_2$.

VAJA 2.5.3:

Seštevanje števil brez predznaka $X = 66_{10}$ in $Y = 35_{10}$. Vsota je $Z = X + Y = 101_{10}$ (Tabela 2.11).

Tabela 2.11: Seštevanje števil brez predznaka

X			0	1	0	0	0	0	1	0			66
Y	+		0	0	1	0	0	0	1	1		+	35
	+	0	0	0	0	0	0	1	0		c_i-prenos		
Z			0	1	1	0	0	1	0	1			101

VAJA 2.5.4:

Seštevanje števil $X = 150_{10}$ in $Y = 112_{10}$ brez predznaka. Vsota je $Z = X + Y = 262_{10}$ (Tabela 2.12).

Tabela 2.12: Seštevanje števil brez predznaka

X			1	0	0	1	0	1	1	0			150
Y	+		0	1	1	1	0	0	0	0		+	112
		1	1	1	1	0	0	0	0		c_i-prenos		
Z			0	0	0	0	0	1	1	1			262

Zadnji prenos pri seštevanju je 1, kar pomeni, da je vsota števil večja od 255_{10} in je ni možno predstaviti z 8-bitnim številom. Rezultat seštevanja je $Z = 7_{10}$ in predstavlja samo tisti del vrednosti, ki je večja od 255_{10}. Rezultat seštevanja dveh števil brez predznaka je napačen, ker je prišlo do prenosa C=1.

VAJA 2.5.5:

Odštevanje števil s predznakom: $X = 65_{10}$, $Y = 35_{10}$, $D = X - Y = 30_{10}$ (Tabela 2.13).

Tabela 2.13: Odštevanje števil

X			0	1	0	0	0	0	0	1			65
Y	-		0	0	1	0	0	0	1	1		-	35
	-	0	1	1	1	1	1	1	0		b_i-sposodek		
D			0	0	0	1	1	1	1	0			30

VAJA 2.5.6:

Odštevanje števil s predznakom izvedemo s seštevanjem v dvojiškem komplementu:
$X = 65_{10}$, $-Y = -35_{10}$, $D = X + (-Y) = 30_{10}$.
Najprej zapišemo število $Y = 00100011_2$ v dvojiškem komplementu $-Y = 11011100_2 + 1_2 = 11011101_2$ in ga prištejemo številu X (Tabela 2.14).

Tabela 2.14: Odštevanje števil izvedeno s seštevanjem v 2'K

X			0	1	0	0	0	0	1				65
$-Y$	+		1	1	0	1	1	1	0	1		+	-35
	+	1	1	0	0	0	0	0	1		c_i-prenos		
D			0	0	0	1	1	1	1	0			30

VAJA 2.5.7:

Odštevanje števil s predznakom: $X = 33_{10}$, $Y = 65_{10}$. Razlika je $D = X - Y = 33_{10} - 65_{10} = -32_{10}$ (Tabela 2.15).

Tabela 2.15: Odštevanje števil s predznakom

X			0	0	1	0	0	0	0	1			33
Y	-		0	1	0	0	0	0	0	1		-	65
	-	1	1	0	0	0	0	0	0		b_i-sposodek		
D			1	1	1	0	0	0	0	0			-32

Razlika števil je negativna, zato je sposodek na najvišjem mestu 1 in rezultat je podan v 2'K. Če želimo videti, da je negativna vrednost razlike 32_{10}, moramo $D = 11100000_2$ pretvoriti v pozitivno število in dobimo 2'K$(D) = 00011111_2 + 1_2 = 00100000_2$, kar je desetiško število 32_{10}.

VAJA 2.5.8:

Odštevanje števil s predznakom: $X = 33_{10}$, $Y = 65_{10}$. Razlika je izvedena s seštevanjem $D = X + (-Y) = -32_{10}$.
Najprej zapišemo število $Y = 65_{10} = 01000001_2$ v dvojiškem komplementu in dobimo $-Y = 10111110_2 + 1_2 = 10111111_2$ ter ga seštejemo z $X = 00100001_2$ (Tabela 2.16). Razlika je negativno število $D = -32_{10}$.

Tabela 2.16: Odštevanje izvedeno s seštevanjem v 2'K

X			0	0	1	0	0	0	0	1			33
$-Y$	+		1	0	1	1	1	1	1	1		+	-65
	+	0	0	1	1	1	1	1	1		c_i-prenos		
D			1	1	1	0	0	0	0	0			-32

VAJA 2.5.9:

Pretvorba 13-bitne dvojiške kode (DK) v Grayevo kodo (GK) (Tabela 2.17).

Tabela 2.17: Pretvorba: DK ⇒ GK

DK:	0	1	0	0	1	0	1	1	1	0	1	0	0
		s	s	e	s	s	s	e	e	s	s	s	e
GK:	0	1	1	0	1	1	1	0	0	1	1	1	0

VAJA 2.5.10:

Pretvorba 10-bitne Grayeve kode (GK) v dvojiško kodo (DK) (Tabela 2.18).

Tabela 2.18: Pretvorba: GK ⇒ DK

GK:		1	0	1	0	1	0	0	1	1	1
	+		1	1	0	0	1	1	1	0	1
DK:		1	1	0	0	1	1	1	0	1	0

NALOGE Z REŠITVAMI:

2.1 Pretvorite desetiška števila v dvojiška števila:
a)200 b)170 c)258 d)234,567 (5 dec mest)
R: a)11001000 b)10101010 c)100000010 d) 11101010,10010

2.2 Pretvorite dvojiška števila v desetiška števila:
a)11100000 b)10011010 c)11110000,0011 d)11100,011
R: a)224 b)154 c)240,1875 d)28,375

2.3 Pretvorite naslednja šestnajstiška števila v dvojiška števila:
a)19_{16} b)$5F_{16}$ c)$E5,04_{16}$ d)$1B,78_{16}$
R: a)11001 b)1011111 c)11100101,000001 d)11011,01111

2.4 Pretvorite dvojiška števila v šestnajstiška števila:
a)11110010 b)101001100100 c)111110,000011 d)10001,11111
R: a)$F2_{16}$ b)$A64_{16}$ c)$3E,0C_{16}$ d)$11,F8_{16}$

2.5 Število v 2'K pretvorite v desetiško s predznakom:
a) 11110001 b)01110000 c)10000000 d)11111000
R: a)-15 b)+112 c)-128 d)-8

2.6 Desetiško število s predznakom pretvorite v 8-bitni dvojiški komplement (2'K):
a) -35 b)-100 c)20 d)8
R: a) 11011101 b)10011100 c)00010100 d)00001000

2.7 Seštejte dvojiški števili s predznakom in zapišite rezultat v desetiškem številskem sistemu:
a) N=00100111 , M=00110110 R: 93
b) N=01100100 , M=00010010 R: 118
c) N=11100111 , M=00110010 R: 25
d) N=00111111 , M=00011100 R: 91

2.8 Poiščite razliko števil $N - M$ in zapišite rezultat v dvojiškem številskem sistemu s predznakom:
a) N= 74 , M= 47 R: 00011011
b) N= 113 , M = 46 R: 01000011
c) N= 104 , M = 106 R: 11111110
d) N= 76 , M = 104 R: 11100100

2.9 Pretvorite dvojiško kodo v Grayevo kodo:
a) DK=1110100111 R: GK=1001110100
b) DK=1100101101101 R: GK=1010111011011

2.10 Pretvorite Grayevo kodo v dvojiško kodo:
a) GK=11011010011010 R: DK=10010011101100
b) GK=1001101100011 R: DK=1110110111101

Poglavje 3

Osnove logičnih vezij in Boolova algebra

3.1 Realna in idealna logična vezja

Logična vezja obravnavamo v idealnem svetu z matematičnim posnemanjem realnih razmer v digitalnih sistemih. Ker pa se pri določenih nalogah srečamo z realnimi logičnimi vezji, si oglejmo nekaj bistvenih značilnosti in pojmov, ki nam predstavljajo povezavo med realnim in idealnim svetom logičnih vezij.

Napetostni nivoji in logika

V logičnih vezjih uporabljamo vrednosti 0 in 1, ki nam v realnem logičnem vezju predstavljata napetostna nivoja (0V proti +5V, -12V proti +12V). Pogosto imamo v logičnih vezjih najnižjo možno napetost (ang. L-low) 0V in najvišjo napetost (ang. H-high) do 5,5V. Logična vezja z uporabljenimi napetostnimi nivoji lahko obravnavamo v pozitivni logiki, označeni s PL, oziroma negativni logiki, označeni z NL (Tabela 3.1).

Tabela 3.1: Napetosti - logika

Napetost	PL	NL
L	0	1
H	1	0

Tehnologija logičnih vezij

Tehnologija logičnih vezij pomeni izbiro materiala iz katerega so zgrajeni transistorji kot osnovni gradniki integriranih vezij. Med najbolj popularni uvrščamo bipolarno in MOS tehnologijo. Pri obravnavanju in načrtovanju logičnih vezij se bomo srečali z različnimi integriranimi elementi ali čipi v TTL tehnologiji, ki ima logična vrata izvedena s transistorji.

3.2 Boolova algebra

Osnovno matematično orodje, ki ga uporabljamo v analizi in sintezi digitalnih logičnih vezij, je Boolova algebra. Razvil in predstavil jo je v prvi polovici devetnajstega stoletja

George Boole [1]. Operacije v Boolovi algebri morajo zadoščati množici pravil, ki jih imenujemo postulati, katerih avtor je Huntington. Postulati so med seboj neodvisni, jih ne dokazujemo in jih uporabimo za dokaz izrekov [2, 4, 5]. Operacije Boolove algebre so:

1) Negacija ali logični NE (NOT): $\bar{x}$, x'.

Tabela 3.2: Negacija

Opis	x	$\bar{x}$
če $x = 0$, potem $\bar{x} = 1$	0	1
če $x = 1$, potem $\bar{x} = 0$	1	0

2) Konjunkcija ali logični IN (AND): xy, $x.y$, $x\&y$, $x \wedge y$.

Tabela 3.3: Konjunkcija

Opis	x	y	xy
če $x = 1$ in $y = 1$, potem $x.y = 1$,	0	0	0
sicer $x.y = 0$	0	1	0
	1	0	0
	1	1	1

3) Disjunkcija ali logični ALI (OR): $x \vee y$, $x + y$.

Tabela 3.4: Disjunkcija

Opis	x	y	$x \vee y$
če $x = 1$ ali $y = 1$ ali $x = y = 1$, potem $x \vee y = 1$,	0	0	0
sicer $x \vee y = 0$	0	1	1
	1	0	1
	1	1	1

Vrstni red operacij v Boolovi algebri je natančno določen tako, da se negacija vedno izvede prva, sledi ji konjunkcija in nato disjunkcija. Z uvedbo oklepajev določamo vrstni red operacij Boolove algebre. Izraz $x \vee (yz)$ je ekvivalenten izrazu $x \vee yz$, zato nam oklepajev ni potrebno pisati. V izrazu $(x \vee y)(z \vee w)$ oklepaj določa najprej operacijo disjunkcije med vhodnima spremenljivkama, ki ji sledi operacija konjunkcije.

3.2.1 Postulati Boolove algebre

Boolova algebra je definirana z množico elementov X, ki zavzamejo vrednosti (0,1), z dvema binarnima operacijama disjunkcije ($\vee$) in konjunkcije (&) ter eno unarno operacijo, to je negacijo ($^-$). Postulati Boolove algebre so:

- Množica X vsebuje vsaj dva elementa $x, y \in X$, tako da velja $x \neq y$.
- *Zaprtost:* Za vsak $x, y \in X$ velja

  p1 : $x \vee y \in X$

  p1': $x.y \in X$

- *Obstoj nevtralnih elementov:* Za vsak $x, y \in X$ velja

  p2 : za $0 \in X$, $x \vee 0 = x$

  p2': za $1 \in X$, $x.1 = x$

- *Komutativnost:* Za vsak $x, y \in X$ velja

  p3 : $x \vee y = y \vee x$

  p3': $x.y = y.x$

- *Distributivnost:* Za vsak $x, y, z \in X$ velja

  p4 : $x \vee (y.z) = x \vee y.z = (x \vee y).(x \vee z)$

  p4': $x.(y \vee z) = (x.y) \vee (x.z) = x.y \vee x.z$

- *Komplementarnost:* Za vsak $x \in X$ obstaja $\overline{x} \in X$ in velja

  p5 : $x \vee \overline{x} = 1$

  p5': $x.\overline{x} = 0$

Vsi postulati Boolove algebre se nahajajo v parih in se razlikujejo v logični operaciji in konstanti. To lastnost imenujemo dualnost. Vsak dualni izraz v Boolovi algebri je izpeljan iz originalnega tako, da zamenjamo operacijo konjunkcije z disjunkcijo in obratno ter konstanto 0 s konstanto 1 in obratno. Spremenljivke v izrazu ostanejo nespremenjene.

$$(x \vee y \vee z \vee ...0 \vee 1)^d = x.y.z...1.0$$
$$(x.y.z...0.1)^d = x \vee y \vee z \vee ... \vee 1 \vee 0$$

Splošen zapis izraza v dualni obliki je podan kot

$$f^d(x, y, z, ...) \quad = \quad \bar{f}(\bar{x}, \bar{y}, \bar{z}, ...).$$

Primer:

a) Izrazu $x \vee 0 = x$ je dualni izraz $x.1 = x$.

b) Dualen zapis izraza $f(x, y, z)$.

$$\begin{aligned} f(x, y, z) &= \bar{x}.\bar{y}.z \vee \bar{x}.y.\bar{z} \vee x.\bar{y}.z \\ f^d(x, y, z) &= \overline{\bar{\bar{x}}.\bar{\bar{y}}.\bar{z} \vee \bar{\bar{x}}.\bar{y}.\bar{\bar{z}} \vee \bar{x}.\bar{\bar{y}}.\bar{\bar{z}}} \\ &= \overline{(x.y.\bar{z})}.\overline{(x.\bar{y}.z)}.\overline{(\bar{x}.y.\bar{z})} \\ &= (\bar{x} \vee \bar{y} \vee z).(\bar{x} \vee y \vee \bar{z}).(x \vee \bar{y} \vee z) \end{aligned}$$

Rešitev, ki je izpeljana na osnovi definicije dualnosti, ustreza pravilu zamenjave operatorjev disjunkcije s konjunkcijo in obratno.

3.2.2 Izreki Boolove algebre

Izreki Boolove algebre predstavljajo množico iz postulatov izpeljanih relacij nad dvojiškimi spremenljivkami, ki jih je potrebno dokazati (Tabela 3.5).

Tabela 3.5: Izreki Boolove algebre

	Operacije s konstantama 0 in 1	
1.	$x \vee 1 = 1$	$x.0 = 0$
	Idempotenca	
2.	$x \vee x = x$	$x.x = x$
	Dvojna negacija	
3.	$\overline{\overline{x}} = x$	
	Asociativnost	
4.	$(x \vee y) \vee z = x \vee (y \vee z) = x \vee y \vee z$	$(x.y).z = x.(y.z) = x.y.z$
	Poenostavitve	
5.	$x \vee x.y = x$	$x.(x \vee y) = x$
6.	$(x \vee \bar{y}).y = x.y$	$x.\bar{y} \vee y = x \vee y$
7.	$x.y \vee x.\bar{y} = x$	$(x \vee y).(x \vee \bar{y}) = x$
8.	$(x \vee y).(\bar{x} \vee z) = x.z \vee \bar{x}.y$	$x.y \vee \bar{x}.z = (x \vee z).(\bar{x} \vee y)$
	DeMorganov izrek	
9.	$\overline{x \vee y \vee z \vee ...} = \overline{x}.\overline{y}.\overline{z}...$	$\overline{x.y.z...} = \overline{x} \vee \overline{y} \vee \overline{z} \vee ...$

Z DeMorganovim izrekom zapišemo negacijo izraza z zamenjavo vseh spremenljivk z njihovimi negacijami in obratno, vseh konstant 1 z 0 in obratno ter operacije konjunkcije z disjunkcijo in obratno. Izrek velja za $n = 2, 3, 4, ...$ spremenljivk.
Negacija disjunkcije spremenljivk ali izrazov je konjunkcija negiranih spremenljivk ali izrazov.

$$\begin{aligned} \overline{x \vee y} &= \bar{x}.\bar{y} \\ \overline{x \vee y \vee z} &= \bar{x}.\bar{y}.\bar{z} \end{aligned}$$

Negacija konjunkcje spremenljivk ali izrazov je konjunkcija negiranih spremenljivk ali izrazov.

$$\begin{aligned} \overline{x.y} &= \bar{x} \vee \bar{y} \\ \overline{x.y.z} &= \bar{x} \vee \bar{y} \vee \bar{z} \end{aligned}$$

Primer: Uporaba DeMorganovega izreka pri iskanju negiranega zapisa $\overline{f(x,y,z)}$ za izraz $f(x,y,z)$.

$$\begin{aligned} f(x,y,z) &= \bar{x}\bar{y}z \vee \bar{x}y\bar{z} \vee x\bar{y}z \vee xyz \\ \overline{f(x,y,z)} &= \overline{\bar{x}\bar{y}z \vee \bar{x}y\bar{z} \vee x\bar{y}z \vee xyz} \\ &= \overline{(\bar{x}\bar{y}z)}.\overline{(\bar{x}y\bar{z})}.\overline{(x\bar{y}z)}.\overline{(xyz)} \\ &= (x \vee y \vee \bar{z})(x \vee \bar{y} \vee z)(\bar{x} \vee y \vee \bar{z})(\bar{x} \vee \bar{y} \vee \bar{z}) \end{aligned}$$

Dokazi izrekov

Vsak od izrekov je razvit iz postulatov Boolove algebre. Dokaze izpeljemo tako, da izraz na levi strani enačbe z uporabo postulatov pretvorimo v izraz na desni strani enačbe. Oglejmo si nekaj primerov dokazov izrekov:

Primer: Dokažimo izrek $x \vee 1 = 1$.

$$\begin{aligned} x \vee 1 &= (x \vee 1).1 \\ &= (x \vee 1).(x \vee \bar{x}) \\ &= x \vee (1.\bar{x}) \\ &= x \vee \bar{x} \\ &= 1 \end{aligned}$$

Primer: Dokažimo izrek $x.x = x$.

$$\begin{aligned} x.x &= x.x \vee 0 \\ &= x.x \vee x\bar{x} \\ &= x.(x \vee \bar{x}) \\ &= x.1 \\ &= x \end{aligned}$$

Primer: Dokažimo izrek $x \vee x.y = x$.

$$\begin{aligned} x \vee x.y &= x.1 \vee x.y \\ &= x.(1 \vee y) \\ &= x.1 \\ &= x \end{aligned}$$

Uporaba postulatov in izrekov za poenostavljenje logičnih zapisov

Postulate in izreke Boolove algebre uporabljamo pri poenostavljanju ali preoblikovanju logičnih zapisov.

Primer: Oglejmo si primer poenostavljanja logičnega zapisa funkcije $f(x, y, z)$, ki je podana z disjunktivno povezanimi konjunktivnimi izrazi.

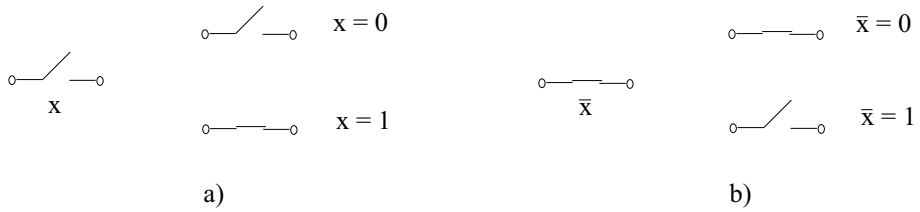
$$\begin{aligned} f(x, y, z) &= \bar{x}\bar{y}z \vee \bar{x}yz \vee x\bar{y}z \vee xyz \\ &= \bar{x}z(\bar{y} \vee y) \vee xz(y \vee \bar{y}) \\ &= \bar{x}z(1) \vee xz(1) \\ &= z(\bar{x} \vee x) \\ &= z(1) \\ &= z \end{aligned}$$

3.2.3 Boolova algebra in stikala

Stikala so najenostavnejša predstavitev digitalnega sistema, ki ponazarjajo njegova logična stanja. Nanašajo se direktno na tranzistorje kot enostavne elektronske elemente iz katerih so zgrajeni digitalni sistemi. Za operacije Boolove algebre, postulate, izreke in logične zapise lahko narišemo svojo stikalno shemo, kar nam omogoča, da si jih laže predstavljamo in zapomnimo.

Stikalo

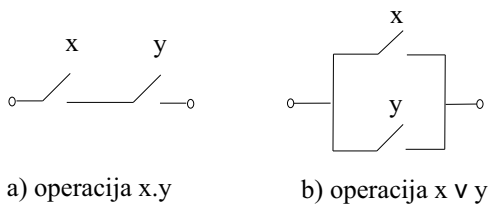
Stikalo predstavlja idealno delovanje tranzistorja. Glede na delovanje definiramo dve vrsti stikal: **normalno odprto stikalo** in **normalno zaprto stikalo** (Slika 3.1). Normalno odprto stikalo je pri logični vrednosti 0 (nizek napetostni nivo) odprto in pri logični vrednosti 1 (visok napetostni nivo) zaprto. Za normalno zaprto stikalo velja ravno nasprotno, tako da je pri logični vrednosti 0 stikalo zaprto in pri 1 odprto.



Slika 3.1: a) Normalno odprto stikalo, b) Normalno zaprto stikalo

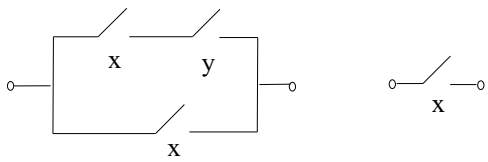
Stikalne sheme Boolovih operacij so zelo enostavne. Določene so z zapisom, ki vsebuje negacije, konjunkcije in disjunkcije. Oglejmo si nekaj stikalnih shem za normalno odprta stikala.

Primer: Za konjunkcijo $(x.y)$ imamo dve zaporedno vezani stikali, in za disjunkcijo $(x \vee y)$ dve paralelno vezani stikali (Slika 3.2).



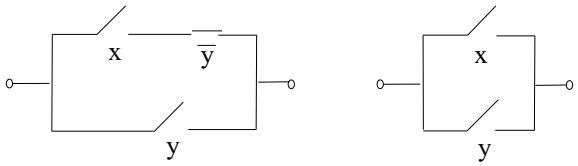
Slika 3.2: Stikalna shema Boolovih operacij AND in OR

Primer: Izrek $x \vee xy = x$ v stikalni shemi (Slika 3.3) ima redundančno vejo zaporedne vezave stikal x in y, ker je izhod določen s paralelno vejo stikala x.



Slika 3.3: Stikalna shema izreka $x \vee xy = x$

Primer: Pri izreku $x\bar{y} \vee y = x \vee y$ v stikalni shemi (Slika 3.4) imamo v paralelni veji y in $\bar{y}$, kar pomeni, da je v primeru zaprtega stikala y, stikalo $\bar{y}$ v zaporedni vezavi z x odprto in nima vpliva na izhod.



Slika 3.4: Stikalna shema izreka $x\bar{y} \vee y = x \vee y$

3.3 Vaje

VAJA 3.4.1:

S postulati dokažite izrek $(x \vee \bar{y}).y = x.y$.

$$\begin{aligned}(x \vee \bar{y}).y &= x.y \vee \bar{y}.y \\ &= x.y \vee 0 \\ &= x.y\end{aligned}$$

VAJA 3.4.2:

S postulati dokažite izrek $x.0 = 0$.

$$\begin{aligned}x.0 &= x.0 \vee 0 \\ &= x.0 \vee x\bar{x} \\ &= x.(0 \vee \bar{x}) \\ &= x.\bar{x} \\ &= 0\end{aligned}$$

VAJA 3.4.3:

S postulati dokažite izrek $x \vee x = x$.

$$\begin{aligned}x \vee x &= (x \vee x).1 \\ &= (x \vee x).(x \vee \bar{x}) \\ &= x \vee (x.\bar{x}) \\ &= x \vee 0 \\ &= x\end{aligned}$$

VAJA 3.4.4:

Poiščite negacijo logičnega zapisa z DeMorganovim izrekom.

$$\begin{aligned}f(x_1, x_2, x_3) &= \bar{x}_1\bar{x}_2x_3 \vee \bar{x}_1\bar{x}_3 \vee x_1x_2x_3 \\ \overline{f(x_1, x_2, x_3)} &= \overline{\bar{x}_1\bar{x}_2x_3 \vee \bar{x}_1\bar{x}_3 \vee x_1x_2x_3} \\ &= \overline{(\bar{x}_1\bar{x}_2x_3)}.\overline{(\bar{x}_1\bar{x}_3)}.\overline{(x_1x_2x_3)} \\ &= (x_1 \vee x_2 \vee \bar{x}_3).(x_1 \vee x_3).(\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)\end{aligned}$$

VAJA 3.4.5:

Poenostavite logični zapis z uporabo postulatov in izrekov.

$$\begin{aligned}f(x_1, x_2, x_3) &= x_1\bar{x}_2x_3 \vee x_1x_2 \vee x_1x_3 \vee \bar{x}_1x_2\bar{x}_3 \\ &= x_1x_3(\bar{x}_2 \vee 1) \vee x_2(x_1 \vee \bar{x}_1\bar{x}_3) \\ &= x_1x_3(1) \vee x_2(x_1 \vee \bar{x}_1)(x_1 \vee \bar{x}_3) \\ &= x_1x_3 \vee x_2(1)(x_1 \vee \bar{x}_3) \\ &= x_1x_3 \vee x_2(x_1 \vee \bar{x}_3)\end{aligned}$$

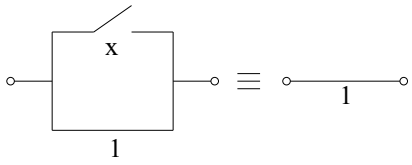
VAJA 3.4.6:

Poenostavite logični zapis z uporabo postulatov in izrekov.

$$\begin{aligned} f(x_1, x_2, x_3, x_4) &= x_1\bar{x}_2x_3x_4 \vee x_1x_4 \vee \bar{x}_1\bar{x}_2x_3x_4 \vee x_1x_2\bar{x}_4 \\ &= \bar{x}_2x_3x_4(x_1 \vee \bar{x}_1) \vee x_1(x_4 \vee x_2\bar{x}_4) \\ &= \bar{x}_2x_3x_4 \vee x_1(x_4 \vee x_2)(x_4 \vee \bar{x}_4) \\ &= \bar{x}_2x_3x_4 \vee x_1(x_2 \vee x_4) \end{aligned}$$

VAJA 3.4.7:

Za izrek $x \vee 1 = 1$ narišite stikalno shemo.

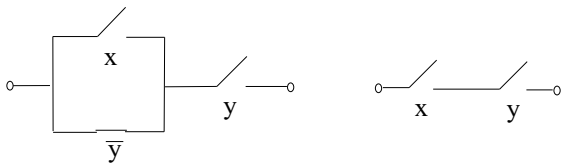


Slika 3.5: Stikalna shema izreka $x \vee 1 = 1$

Stikalo za x v zgornji paralelni veji nima nobenega vpliva na izhod, ker je ta določen z žico v spodnji veji. (Slika 3.5).

VAJA 3.4.8:

Za izrek $(x \vee \bar{y}).y = x.y$ narišite stikalno shemo.



Slika 3.6: Stikalna shema izreka $(x \vee \bar{y}).y = x.y$

Stikalo $\bar{y}$ v paralelni vezavi je redundančno, ker je izhod določen z odprtim ali zaprtim stikalom y v serijski vezavi (Slika 3.6).

Poglavje 4

Logične funkcije

Logične funkcije v kombinacijskem vezju imajo izhode odvisne samo od trenutnih vhodov. Predstavili in zapisali jih bomo z naslednjimi parametri:

- 0,1 - konstanti,
- $x_1, x_2, ..., x_n$ - vhodne spremenljivke,
- $f(x_1, x_2, ..., x_n)$ - logična funkcija,
- w_i - i-ta vhodna kombinacija ali i-ti vhodni vektor,
- $f(w_i) = f_i$ - funkcijska vrednost i-te vhodne kombinacije ali i-tega vhodnega vektorja.

Logična funkcija $f(x_1, x_2, ..., x_n)$ razdeli množico vhodnih kombinacij oz. vhodnih vektorjev v dve podmnožici. Prvo podmnožico sestavljajo vhodne kombinacije w_i, ki imajo funkcijsko vrednost $f(w_i) = 1$, drugo pa tiste, ki imajo funkcijsko vrednost $f(w_i) = 0$. Vsaka spremenljivka zavzame vrednost 0 ali 1, zato je logična funkcija z n spremenljivkami določena z 2^n vhodnimi kombinacijami, ali vhodnimi vektorji w_i, $i = 0, 1, ..., 2^n - 1$, in enakim številom funkcijskih vrednosti $f(w_i)$ [6, 2].
Logično funkcijo lahko zapišemo oziroma predstavimo na različne načine: s pravilnostno tabelo, v normalni obliki, v večnivojski obliki, grafično, z logično shemo.

Zapis logične funkcije v pravilnostni tabeli

Logično funkcijo podamo v pravilnostni tabeli, kjer na levo stran tabele vpišemo vhodne kombinacije w_i, $i = 0, 1, ..., 2^{n-1}$, pri vhodnih spremenljivkah x_1, x_2,..., x_n, desna stran tabele pa vsebuje zapis logične funkcije s funkcijskimi vrednostmi posamezne vhodne kombinacije $f(w_i)$ (Tabela 4.1).

Tabela 4.1: Pravilnostna tabela

$x_1x_2...x_n$	$f(x_1, x_2, ..., x_n)$
w_0	$f(w_0) = f_0$
w_1	$f(w_1) = f_1$
w_2	$f(w_2) = f_2$
...	...
w_{2^n-2}	$f(w_{2^n-2}) = f_{2^n-2}$
w_{2^n-1}	$f(w_{2^n-1}) = f_{2^n-1}$

Primer: Za funkcijo treh spremenljivk $f(x_1, x_2, x_3)$ zapišemo pravilnostno tabelo z $2^3 = 8$ vhodnimi kombinacijami w_i, $i = 0, 1, ..., 7$, in pripadajočimi funkcijskimi vrednostmi f_i (Tabela 4.2). Vrstica v pravilnostni tabeli vsebuje vrednost logične funkcije pri trenutnih vrednostih vhodov x_1, x_2, x_3.

Tabela 4.2: Pravilnostna tabela logične funkcije za $n = 3$

i	w_i	$x_1x_2x_3$	$f(x_1, x_2, x_3)$
0	$w_0 =$	0 0 0	$f(w_0) = f_0$
1	$w_1 =$	0 0 1	$f(w_1) = f_1$
2	$w_2 =$	0 1 0	$f(w_2) = f_2$
3	$w_3 =$	0 1 1	$f(w_3) = f_3$
4	$w_4 =$	1 0 0	$f(w_4) = f_4$
5	$w_5 =$	1 0 1	$f(w_5) = f_5$
6	$w_6 =$	1 1 0	$f(w_6) = f_6$
7	$w_7 =$	1 1 1	$f(w_7) = f_7$

Zapis logične funkcije v normalni obliki

Logične funkcije zapisujemo v kanonični ali dvonivojski obliki, ker imamo dva nivoja operatorjev (1.nivo, 2.nivo) v katere vstopajo vhodne spremenljivke. Pri tako določenem zapisu logične funkcije bomo v nadaljevanju uporabljali:

1) popolno normalno obliko,

2) normalno obliko,

3) minimalno normalno obliko.

Logično funkcijo zapisano v normalni obliki lahko vpišemo v pravilnostno tabelo ali pa poiščemo popolno normalno obliko z uporabo izrekov in postulatov Boolove algebre. Ogledali si bomo najprej logične funkcije v pravilnostni tabeli in jih zapisali v popolni normalni obliki.

Zapis logične funkcije v nenormalni ali večnivojski obliki

Večnivojska oblika logične funkcije se od normalne oblike razlikuje po številu nivojev operatorjev. Logična funkcija je v tem primeru zapisana s tremi ali več nivoji operatorjev v katere vstopajo vhodne spremenljivke. Poleg operatorjev konjunkcije, disjunkcije in negacije bomo v teh oblikah uporabili tudi druge, ki jih bomo spoznali v nadaljevanju.

Primer: Štirinivojski zapis logične funkcije v odvisnosti od vhodnih spremenljivk x, y, z, v.

$$g = (((\bar{x}.\bar{y}) \vee z).x) \vee y.\bar{v}$$

Grafični zapis logične funkcije

Za grafični zapis logičnih funkcij uporabljamo Veitchev ali Karnaughjev diagram v katerega vpisujemo funkcijske vrednosti. Uporabljamo ga pri ugotavljanju linearnosti logičnih funkcij in pri minimizaciji logičnih funkcij.

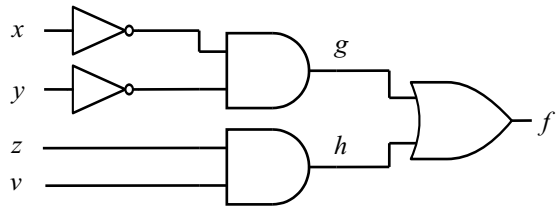
Predstavitev logične funkcije z logično shemo

Če je logična funkcija predstavljena z logičnimi vrati, njihovimi povezavami in povezavami vhodov na logična vrata, dobimo logično shemo. Taka predstavitev se uporablja pri realizaciji in testiranju digitalnih vezij. Oglejmo si predstavitev dveh logičnih funkcij v različnih logičnih vezjih.

Primer: Vzemimo Boolov izraz $f = \bar{x}.\bar{y} \vee z.v$, ga zapišimo v obliko z oklepaji glede na vrstni red izvajanja operacij in dobimo

$$f = ((\bar{x}.\bar{y}) \vee (z.v)),$$

kjer nam vsak par v oklepaju predstavlja izraz določen z logičnimi vrati. Funkcija je podana z množico vmesnih rezultatov od znotraj navzven, kar lahko zapišemo kot $g = \bar{x}.\bar{y}$, $h = z.v$ in $f = g \vee h$. Za funkcijo dobimo logično vezje (Slika 4.1) zgrajeno iz dvojih konjunktivnih vrat in enih disjunktivnih vrat.



Slika 4.1: Logično vezje funkcije f

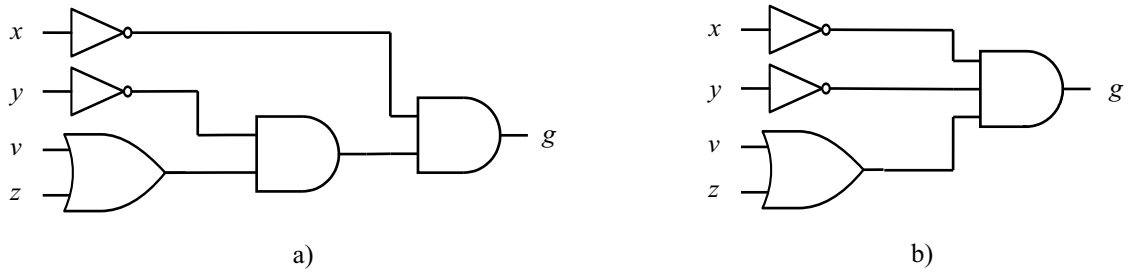
Primer: Vzemimo Boolov izraz $g = \bar{x}.\bar{y}.(v \vee z)$ in ga zapišimo v obliko z oklepaji na dva načina:

a) operacije so 2-vhodne (Slika 4.2.a)

$$g = (\bar{x}(\bar{y}(v \vee z)))$$

b) operacije so 2- in 3-vhodne (Slika 4.2.b)

$$g = \bar{x}\bar{y}(v \vee z)$$



Slika 4.2: Logično vezje funkcije g

4.1 Elementarne logične funkcije

Za n vhodnih spremenljivk je v Booleovi algebri možno tvoriti 2^{2^n} različnih funkcij. Za dve spremenljivki $n = 2$ imamo 16 različnih funkcij, ki jih definiramo kot elementarne logične funkcije (Tabela 4.3).

Tabela 4.3: Elementarne logične funkcije

x_1	1	1	0	0	Opis	Logične funkcije
x_2	1	0	1	0		
F_0	0	0	0	0	konstanta 0	0
F_1	0	0	0	1	NOR (Pierce)	$x_1 \downarrow x_2$
F_2	0	0	1	0	negacija implikacije x_2 v x_1	$\overline{x_2 \rightarrow x_1}$
F_3	0	0	1	1	negacija spremenenljivke x_1	$\bar{x}_1$
F_4	0	1	0	0	negacija implikacije x_1 v x_2	$\overline{x_1 \rightarrow x_2}$
F_5	0	1	0	1	negacija spremenljivke x_2	$\bar{x}_2$
F_6	0	1	1	0	XOR (seštevanje po mod 2)	$x_1 \nabla x_2$
F_7	0	1	1	1	NAND (Sheffer)	$x_1 \uparrow x_2$
F_8	1	0	0	0	AND (konjunkcija)	$x_1 x_2$
F_9	1	0	0	1	XNOR (ekvivalenca)	$x_1 \equiv x_2$
F_{10}	1	0	1	0	spremenljivka x_2	x_2
F_{11}	1	0	1	1	implikacija x_1 v x_2	$x_1 \rightarrow x_2$
F_{12}	1	1	0	0	spremenljivka x_1	x_1
F_{13}	1	1	0	1	implikacija x_2 v x_1	$x_2 \rightarrow x_1$
F_{14}	1	1	1	0	OR (disjunkcija)	$x_1 \vee x_2$
F_{15}	1	1	1	1	konstanta 1	1

Za najpogosteje uporabljene logične operatorje bomo uporabljali angleške oznake, kot so: negacija (NOT), konjunkcija (AND), disjunkcija (OR), negacija konjunkcije (NAND), negacija disjunkcije (NOR), seštevanje po modulu 2 (XOR), ekvivalenca (XNOR).

Zapis elementarnih logičnih funkcij

V množici elementarnih logičnih funkcij dveh spremenljivk imamo konstanti 0 in 1, spremenljivki x_1 in x_2, negaciji spremenljivk $\bar{x}_1$ in $\bar{x}_2$, logični operaciji konjunkcije in disjunkcije ter druge logične operacije. Vseh šestnajst elementarnih funkcij glede na značilnosti lahko razdelimo v štiri skupine:

1) Enostavne logične funkcije, ki jih zapišemo iz osnov Boolove algebre.

   $F_0 = 0$ - konstanta 0

   $F_3 = \bar{x}_1\bar{x}_2 \vee \bar{x}_1 x_2 = \bar{x}_1(\bar{x}_2 \vee x_2) = \bar{x}_1$ - negacija spremenljivke x_1

   $F_5 = \bar{x}_1\bar{x}_2 \vee x_1\bar{x}_2 = \bar{x}_2(\bar{x}_1 \vee x_1) = \bar{x}_2$ - negacija spremenljivke x_2

   $F_8 = x_1 x_2$ - AND (konjunkcija)

   $F_{10} = \bar{x}_1 x_2 \vee x_1 x_2 = x_2(\bar{x}_1 \vee x_1) = x_2$ - spremenljivka x_2

   $F_{12} = x_1\bar{x}_2 \vee x_1 x_2 = x_1(\bar{x}_2 \vee x_2) = x_1$ - spremenljivka x_1

   $F_{14} = x_1 \vee x_2$ - OR (disjunkcija)

   $F_{15} = 1$ - konstanta 1

2) Dve najbolj pogosti logični funkciji, ki ju dobimo z negacijo disjunkcije (NOR) in negacijo konjunkcije (NAND).

   $F_1 = \bar{x}_1\bar{x}_2 = \overline{x_1 \vee x_2} = x_1 \downarrow x_2$ - NOR (Pierce)

   $F_7 = \bar{x}_1 \vee \bar{x}_2 = \overline{x_1 x_2} = x_1 \uparrow x_2$ - NAND (Sheffer)

3) Naslednji par logičnih funkcij sestavljata XOR in XNOR

   $F_6 = \bar{x}_1 x_2 \vee x_1\bar{x}_2 = x_1 \nabla x_2$ - XOR (seštevanje po mod 2)

   $F_9 = \bar{x}_1\bar{x}_2 \vee x_1 x_2 = x_1 \equiv x_2$ - XNOR (ekvivalenca)

   Z negacijo logične funkcije f_6 dobimo funkcijo f_9 in obratno.
   negacija XOR funkcije: (NOT XOR) = XNOR: $\overline{x_1 \nabla x_2} = x_1 \equiv x_2$
   negacija XNOR funkcije: (NOT XNOR) = XOR: $\overline{x_1 \equiv x_2} = x_1 \nabla x_2$

4) Zadnja skupina logičnih funkcij ima uporabljeno logično operacijo implikacije, za katero nimamo svojih logičnih vrat in se zelo redko pojavlja v izvedbi logičnih vezij.

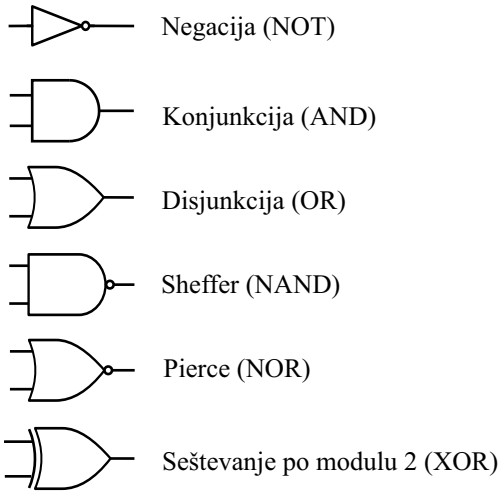
   $F_2 = \bar{x}_1 x_2 = \overline{x_2 \rightarrow x_1}$ - negacija x_2 implicira x_1

   $F_4 = x_1\bar{x}_2 = \overline{x_1 \rightarrow x_2}$ - negacija x_1 implicira x_2

   $F_{11} = \bar{x}_1 \vee x_2 = (x_1 \rightarrow x_2)$ - x_1 implicira x_2

   $F_{13} = x_1 \vee \bar{x}_2 = (x_2 \rightarrow x_1)$ - x_2 implicira x_1

Za večino logičnih operatorjev, ki jih uporabljamo v zapisih logičnih funkcij, imamo tudi logična vrata s katerimi rišemo logične sheme, ki nam ponazarjajo delovanje vezja. Slika 4.3 prikazuje simbole logičnih vrat.



Slika 4.3: Logična vrata

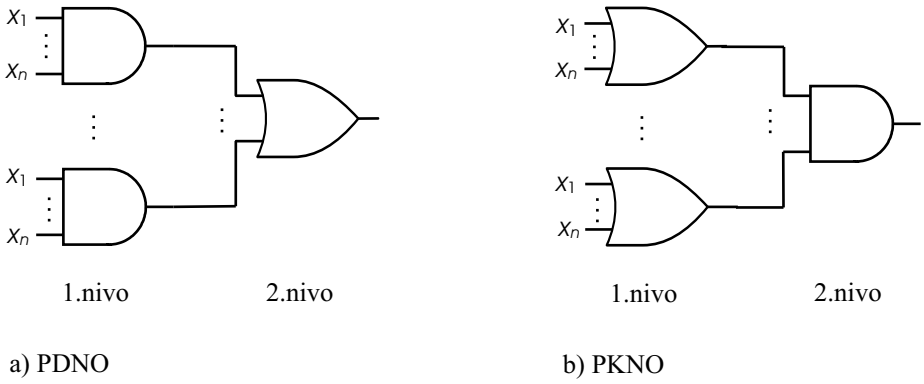
4.2 Popolna normalna oblika logične funkcije

Zapis logičnih funkcij v popolni normalni obliki pomeni, da ima funkcija dva nivoja operatorjev in da vse spremenljivke vstopajo v prvi nivo operatorjev (Slika 4.4).

Uporabljamo dva zapisa logične funkcije v popolni normalni obliki z operatorji konjunkcije in disjunkcije:

- Popolna disjunktivna normalna oblika (PDNO) - na prvem nivoju imamo operatorje konjunkcije, in na drugem nivoju operator disjunkcije (Slika 4.4.a).

- Popolna konjunktivna normalna oblika (PKNO) - na prvem nivoju imamo operatorje disjunkcije in na drugem nivoju operator konjunkcije (Slika 4.4.b).



Slika 4.4: Logična shema popolne normalne oblike logične funkcije

4.2.1 Popolna disjunktivna normalna oblika (PDNO)

Zapis logične funkcije $f(x_1, x_2, ..., x_n)$ v PDNO:

- Za vsako vhodno kombinacijo w_i, $i = 1, 2, ..., 2^n - 1$ zapišemo minterm m_i, ki je določen kot konjunktivna povezava vseh vhodnih spremenljivk. V mintermu je spremenljivka negirana ($\bar{x}_i$), če ima v vhodni kombinaciji vrednost 0 in je nenegirana (x_i), če ima v vhodni kombinaciji vrednost 1.
  V tabeli 4.4 je zapisanih $2^3 = 8$ mintermov logične funkcije treh spremenljivk.

Tabela 4.4: Mintermi - $m_i, i = 0, 1, ..., 7$

i	$x_1x_2x_3$	m_i	f_i
0	0 0 0	$m_0 = \bar{x}_1\bar{x}_2\bar{x}_3$	f_0
1	0 0 1	$m_1 = \bar{x}_1\bar{x}_2x_3$	f_1
2	0 1 0	$m_2 = \bar{x}_1x_2\bar{x}_3$	f_2
3	0 1 1	$m_3 = \bar{x}_1x_2x_3$	f_3
4	1 0 0	$m_4 = x_1\bar{x}_2\bar{x}_3$	f_4
5	1 0 1	$m_5 = x_1\bar{x}_2x_3$	f_5
6	1 1 0	$m_6 = x_1x_2\bar{x}_3$	f_6
7	1 1 1	$m_7 = x_1x_2x_3$	f_7

- Vsak minterm m_i je konjunktivno povezan s funkcijsko vrednostjo f_i v pripadajoči vrstici. Dobljeni konjunktivni izrazi so nato disjunktivno povezani v popolno disjunktivno normalno obliko (PDNO).

$$f(x_1, x_2, x_3) = m_0f_0 \vee m_1f_1 \vee m_2f_2 \vee m_3f_3 \vee m_4f_4 \vee m_5f_5 \vee m_6f_6 \vee m_7f_7$$

Splošna definicija zapisa logične funkcije v PDNO za n spremenljivk je podana z operacijo disjunkcije, ki povezuje 2^n mintermov konjunktivno povezanih s funkcijskimi vrednostmi kot

$$f(x_1, x_2, ..., x_n) = \bigvee_{i=0}^{2^n-1} m_i f_i \quad . \tag{4.1}$$

Primer: Oglejmo si zapis podane logične funkcije $f(x_1, x_2, x_3)$ v popolni disjunktivni normalni obliki (PDNO) (Tabela 4.5).

Tabela 4.5: Funkcija $f(x_1, x_2, x_3)$

i	$x_1x_2x_3$	f
0	0 0 0	1
1	0 0 1	0
2	0 1 0	1
3	0 1 1	0
4	1 0 0	1
5	1 0 1	0
6	1 1 0	0
7	1 1 1	1

Za zapis funkcije v PDNO uporabimo analitično obliko za tri vhodne spremenljivke

$$f(x_1, x_2, x_3) = m_0 f_0 \vee m_1 f_1 \vee m_2 f_2 \vee m_3 f_3 \vee m_4 f_4 \vee m_5 f_5 \vee m_6 f_6 \vee m_7 f_7$$

in vanjo vstavimo funkcijske vrednosti iz pravilnostne tabele

$$\begin{aligned} f(x_1, x_2, x_3) &= m_0.1 \vee m_1.0 \vee m_2.1 \vee m_3.0 \vee m_4.1 \vee m_5.0 \vee m_6.0 \vee m_7.1 \\ &= m_0 \vee m_2 \vee m_4 \vee m_7 \\ &= \bar{x}_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1x_2\bar{x}_3 \vee x_1\bar{x}_2\bar{x}_3 \vee x_1x_2x_3 \ . \end{aligned}$$

Numeričen zapis logične funkcije v PDNO

Logično funkcijo podano v popolni disjunktivni normalni (PDNO) lahko zapišemo krajše v numerični obliki. Pri oznaki funkcije povemo število vhodnih spremenljivk n kot f^n in v oklepaju za operatorjem disjunkcije ($\vee$) v naraščajočem vrstnem redu vpišemo desetiške indekse mintermov pri katerih imamo funkcijske vrednosti 1.

Primer: Logična funkcija $f(x_1, x_2, x_3)$ iz tabele 4.5 je v numerični obliki zapisana kot

$$f^3 = \vee(0, 2, 4, 7) \ .$$

4.2.2 Popolna konjunktivna normalna oblika (PKNO)

Zapis logične funkcije $f(x_1, x_2, ..., x_n)$ v PKNO:

- Za vsako vhodno kombinacijo w_i, $i = 1, 2, ..., 2^n - 1$ zapišemo maksterm M_j, ki je določen kot disjunktivna povezava vseh vhodnih spremenljivk. V makstermu je spremenljivka negirana ($\bar{x}_i$), če ima v vhodni kombinaciji vrednost 1 in je nenegirana (x_i), če ima v vhodni kombinaciji vrednost 0. V tabeli 4.6 je zapisanih $2^3 = 8$ makstermov funkcije treh spremenljivk.

Tabela 4.6: Makstermi - $M_j, j = 7, 6, ..., 0$

i	$j = 2^3 - 1 - i$	$x_1x_2x_3$	M_j	f_i
0	7	0 0 0	$M_7 = x_1 \vee x_2 \vee x_3$	f_0
1	6	0 0 1	$M_6 = x_1 \vee x_2 \vee \bar{x}_3$	f_1
2	5	0 1 0	$M_5 = x_1 \vee \bar{x}_2 \vee x_3$	f_2
3	4	0 1 1	$M_4 = x_1 \vee \bar{x}_2 \vee \bar{x}_3$	f_3
4	3	1 0 0	$M_3 = \bar{x}_1 \vee x_2 \vee x_3$	f_4
5	2	1 0 1	$M_2 = \bar{x}_1 \vee x_2 \vee \bar{x}_3$	f_5
6	1	1 1 0	$M_1 = \bar{x}_1 \vee \bar{x}_2 \vee x_3$	f_6
7	0	1 1 1	$M_0 = \bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3$	f_7

- Vsak maksterm M_j je disjunktivno povezan s funkcijsko vrednostjo f_i v pripadajoči vrstici. Dobljeni disjunktivni izrazi so konjunktivno povezani v popolno konjunktivno normalno obliko (PKNO).

$$\begin{aligned} f(x_1, x_2, x_3) = \quad & (M_7 \vee f_0)(M_6 \vee f_1)(M_5 \vee f_2)(M_4 \vee f_3) \\ & (M_3 \vee f_4)(M_2 \vee f_5)(M_1 \vee f_6)(M_0 \vee f_7) \end{aligned}$$

Splošna definicija zapisa logične funkcije v PKNO za n spremenljivk je podana z operacijo konjunkcije, ki povezuje 2^n makstermov disjunktivno povezanih s funkcijskimi vrednostmi.

$$f(x_1, x_2, ..., x_n) = \mathop{\&}_{i=0}^{2^n-1} (M_{j=2^n-1-i} \vee f_i) \tag{4.2}$$

Primer: Oglejmo si zapis podane logične funkcije $f(x_1, x_2, x_3)$ v popolni konjunktivni normalni obliki (PKNO) (Tabela 4.7).

Tabela 4.7: Funkcija $f(x_1, x_2, x_3)$

i	$j = 2^3 - i - 1$	$x_1x_2x_3$	f
0	7	0 0 0	1
1	6	0 0 1	0
2	5	0 1 0	1
3	4	0 1 1	0
4	3	1 0 0	1
5	2	1 0 1	0
6	1	1 1 0	0
7	0	1 1 1	1

Za zapis funkcije v PKNO uporabimo analitično obliko za tri vhodne spremenljivke

$$\begin{aligned} f(x_1, x_2, x_3) = \quad & (M_7 \vee f_0)(M_6 \vee f_1)(M_5 \vee f_2)(M_4 \vee f_3) \\ & (M_3 \vee f_4)(M_2 \vee f_5)(M_1 \vee f_6)(M_0 \vee f_7) \end{aligned}$$

in vanjo vstavimo funkcijske vrednosti

$$\begin{aligned} f(x_1, x_2, x_3) = \quad & (M_7 \vee 1)(M_6 \vee 0)(M_5 \vee 1)(M_4 \vee 0) \\ & (M_3 \vee 1)(M_2 \vee 0)(M_1 \vee 0)(M_0 \vee 1) \\ = \quad & M_6M_4M_2M_1 \\ = \quad & (x_1 \vee x_2 \vee \bar{x}_3)(x_1 \vee \bar{x}_2 \vee \bar{x}_3)(\bar{x}_1 \vee x_2 \vee \bar{x}_3)(\bar{x}_1 \vee \bar{x}_2 \vee x_3) \ . \end{aligned}$$

Numeričen zapis logične funkcije v PKNO

Logično funkcijo podano v popolni konjunktivni normalni (PKNO) lahko zapišemo krajše v numerični obliki. Pri oznaki funkcije povemo število vhodnih spremenljivk n kot f^n in v oklepaju za operatorjem konjunkcije (&) v padajočem vrstnem redu vpišemo desetiške indekse makstermov pri katerih imamo funkcijske vrednosti 0.

Primer: Logična funkcija $f(x_1, x_2, x_3)$ iz tabele 4.7 je v numerični obliki zapisana kot

$$f^3 = \&(6, 4, 2, 1) \ .$$

4.2.3 Relacije med mintermi in makstermi

Mintermi in makstermi predstavljajo zapis vhodnih kombinacij logične funkcije z vhodnimi spremenljivkami in operacijama disjunkcije in konjunkcije (Tabela 4.8).

Tabela 4.8: Mintermi in makstermi

i	$x_1x_2x_3$	m_i	$j = 2^3 - 1 - i$	M_j
0	0 0 0	$m_0 = \bar{x}_1\bar{x}_2\bar{x}_3$	7	$M_7 = x_1 \vee x_2 \vee x_3$
1	0 0 1	$m_1 = \bar{x}_1\bar{x}_2x_3$	6	$M_6 = x_1 \vee x_2 \vee \bar{x}_3$
2	0 1 0	$m_2 = \bar{x}_1x_2\bar{x}_3$	5	$M_5 = x_1 \vee \bar{x}_2 \vee x_3$
3	0 1 1	$m_3 = \bar{x}_1x_2x_3$	4	$M_4 = x_1 \vee \bar{x}_2 \vee \bar{x}_3$
4	1 0 0	$m_4 = x_1\bar{x}_2\bar{x}_3$	3	$M_3 = \bar{x}_1 \vee x_2 \vee x_3$
5	1 0 1	$m_5 = x_1\bar{x}_2x_3$	2	$M_2 = \bar{x}_1 \vee x_2 \vee \bar{x}_3$
6	1 1 0	$m_6 = x_1x_2\bar{x}_3$	1	$M_1 = \bar{x}_1 \vee \bar{x}_2 \vee x_3$
7	1 1 1	$m_7 = x_1x_2x_3$	0	$M_0 = \bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3$

Za minterme in maksterme lahko določimo naslednje relacije:

1) Negacija minterma in maksterma:

$$\overline{m}_i = M_{2^n-1-i}$$
$$\overline{M}_j = m_{2^n-1-j}$$

Primer:
$\overline{m}_5 = \overline{x_1\bar{x}_2x_3} = \bar{x}_1 \vee x_2 \vee \bar{x}_3 = M_2$
$\overline{M}_5 = \overline{x_1 \vee \bar{x}_2 \vee x_3} = \bar{x}_1x_2\bar{x}_3 = m_2$

2) Disjunkcija istoležnega minterma in maksterma:

$$m_i \vee M_{2^n-1-i} = 1$$
$$M_j \vee m_{2^n-1-j} = 1$$

Primer:
$m_5 \vee M_2 = m_5 \vee \overline{m}_5 = 1$
$M_5 \vee m_2 = M_5 \vee \overline{M}_5 = 1$

3) Konjunkcija istoležnega minterma in maksterma:

$$m_i.M_{2^n-1-i} = 0$$
$$M_j.m_{2^n-1-j} = 0$$

Primer:
$m_5.M_2 = m_5.\overline{m}_5 = 0$
$M_5.m_2 = M_5.\overline{M}_5 = 0$

4) Konjunkcija dveh mintermov in disjunkcija dveh makstermov:

$$m_i.m_j = 0, i \neq j$$
$$M_i \vee M_j = 1, i \neq j$$

Primer:
$m_5.m_7 = (x_1\bar{x}_2x_3).(x_1x_2x_3) = x_1\bar{x}_2x_2x_3 = 0$
$M_3 \vee M_6 = (\bar{x}_1 \vee x_2 \vee x_3) \vee (x_1 \vee x_2 \vee \bar{x}_3) = \bar{x}_1 \vee x_1 \vee x_2 \vee x_3 \vee \bar{x}_3 = 1$

5) Disjunkcija vseh mintermov in konjunkcija vseh makstermov:

$$\bigvee_{i=0}^{2^n-1} m_i = 1$$
$$\bigwedge_{j=2^n-1}^{0} M_j = 0$$

Disjunkcija vseh mintermov pomeni, da imamo pri vseh mintermih funkcijsko vrednost 1, kar je konstanta 1.
Konjunkcija vseh makstermov pomeni, da imamo pri vseh makstermih funkcijsko vrednost 0, kar je konstanta 0.

4.2.4 Pretvorbe logičnih funkcij

Podano logično funkcijo lahko vedno pretvorimo iz PDNO v PKNO ali obratno tako, da jo zapišemo v pravilnostno tabelo in iz nje zapišemo PKNO. Oglejmo si analitičen zapis za pretvarjanje logičnih funkcij iz ene oblike v drugo.

Pretvorba PDNO ⇒ PKNO

Za logično funkcijo podano z mintermi v PDNO

$$f = f(x_1, x_2, ..., x_n) = \bigvee_{i=0}^{2^n-1} m_i f_i \quad , \tag{4.3}$$

nas zanima, kako pridemo do zapisa z makstermi v PKNO. Zapišemo najprej negacijo funkcije $\bar{f}$ v PDNO tako, da negiramo funkcijske vrednosti f_i in dobimo

$$\bar{f} = \bigvee_{i=0}^{2^n-1} m_i \bar{f}_i \quad . \tag{4.4}$$

Izpis negirane funkcije nam da izraz

$$\bar{f} = m_0\bar{f}_0 \vee m_1\bar{f}_1 \vee ... \vee m_{2^n-1}\bar{f}_{2^n-1} \; . \tag{4.5}$$

S ponovno negacijo zgornje enačbe dobimo nazaj funkcijo f, kjer z uporabo postulatov in izrekov Boolove algebre ter relacij med mintermi in makstermi pridemo do zapisa logične funkcije v PKNO.

$$\begin{aligned} f = \overline{\bar{f}} &= \overline{m_0\bar{f}_0 \vee m_1\bar{f}_1 \vee ... \vee m_{2^n-1}\bar{f}_{2^n-1}} \\ &= \overline{(m_0\bar{f}_0)}.\overline{(m_1\bar{f}_1)}...\overline{(m_{2^n-1}\bar{f}_{2^n-1})} \\ &= (\overline{m}_0 \vee f_0)(\overline{m}_1 \vee f_1)...(\overline{m}_{2^n-1} \vee f_{2^n-1}) \\ &= (M_{2^n-1} \vee f_0)(M_{2^n-2} \vee f_1)...(M_0 \vee f_{2^n-1}) \end{aligned} \tag{4.6}$$

Iz zgornje enačbe dobimo splošen zapis PKNO kot

$$f(x_1, x_2, ..., x_n) = \overset{2^n-1}{\underset{i=0}{\&}}(M_{2^n-1-i} \vee f_i) \ . \tag{4.7}$$

Najkrajši način pretvorbe logične funkcije iz PDNO v PKNO imamo pri uporabi numeričnega zapisa. Iz PDNO določimo minterme s funkcijskimi vrednostmi $f_i = 0$, to so manjkajoči mintermi m_i. Poiščemo istoležne maksterme M_j, $j = 2^n - 1 - i$ in zapišemo PKNO.

Primer: Oglejmo si pretvorbo logične funkcije iz PDNO v PKNO.
PDNO: $f(x_1, x_2, x_3) = f^3 = \vee(3, 4, 6, 7)$

Logična funkcija ima v PDNO pri mintermih 3,4,6,7 vrednost $f_i = 1$, pri preostalih pa funkcijsko vrednost $f_i = 0$, kar pomeni, da manjkajoči mintermi določajo PKNO. Če želimo funkcijo zapisati v PKNO, moramo manjkajoče minterme pretvoriti v maksterme. Zapis mintermov in makstermov nam pove, da se na mestu minterma i nahaja maksterm j. Pretvorbo mintermov m_i v maksterme M_j zato izvedemo z enačbo $j = 2^n - 1 - i = 7 - i$:

- manjkajoči mintermi m_i za $f_i = 0$ so: 0,1,2,5
- makstermi M_j za $f_i = 0$ so: 7,6,5,2

Popolno konjunktivno normalno obliko sedaj zapišemo z makstermi v padajočem vrstnem redu in dobimo

PKNO: $f(x_1, x_2, x_3) = f^3 = \&(7, 6, 5, 2)$.

Pretvorba PKNO ⇒ PDNO

Za logično funkcijo podano z makstermi v PKNO

$$f = f(x_1, x_2, ..., x_n) = \overset{2^n-1}{\underset{i=0}{\&}}(M_{2^n-1-i} \vee f_i) \tag{4.8}$$

nas zanima, kako pridemo do zapisa z mintermi v PDNO. Zapišemo najprej negacijo funkcije $\bar{f}$ v PKNO tako, da negiramo funkcijske vrednosti f_i in dobimo

$$\bar{f} = \overset{2^n-1}{\underset{i=0}{\&}}(M_{2^n-1-i} \vee \bar{f}_i) \tag{4.9}$$

Izpis negirane funkcije nam da izraz

$$\bar{f} = (M_{2^n-1} \vee \bar{f}_0)(M_{2^n-2} \vee \bar{f}_1) \vee ... \vee (M_0 \vee \bar{f}_{2^n-1}) \ , \tag{4.10}$$

ki ga še enkrat negiramo in dobimo funkcijo f, ki bo zapisana v popolni disjunktivni normalni obliki.

$$\begin{aligned} f = \overline{\bar{f}} &= \overline{(M_{2^n-1} \vee \bar{f}_0)(M_{2^n-2} \vee \bar{f}_1) \vee ... \vee (M_0 \vee \bar{f}_{2^n-1})} \\ &= \overline{(M_{2^n-1} \vee \bar{f}_0)} \vee \overline{(M_{2^n-2} \vee \bar{f}_1)} \vee ... \vee \overline{(M_0 \vee \bar{f}_{2^n-1})} \\ &= \overline{M}_{2^n-1} f_0 \vee \overline{M}_{2^n-2} f_1 \vee ... \vee \overline{M}_0 f_{2^n-1} \\ &= m_0 f_0 \vee m_1 f_1 \vee ... \vee m_{2^n-1} f_{2^n-1} \end{aligned} \tag{4.11}$$

Iz tega dobimo splošen zapis za PDNO

$$f(x_1, x_2, ..., x_n) = \overset{2^n-1}{\underset{i=0}{\vee}} m_i f_i \ . \tag{4.12}$$

Najkrajši način pretvorbe logične funkcije iz PKNO v PDNO imamo pri uporabi numeričnega zapisa. Iz PKNO določimo maksterme s funkcijskimi vrednostmi $f_i = 1$, to so manjkajoči makstermi M_j. Poiščemo istoležne minterme m_i, $i = 2^n - 1 - j$ in zapišemo PDNO.

Primer: Oglejmo si pretvorbo logične funcije iz PKNO v PDNO.
PKNO: $f(x_1, x_2, x_3) = f^3 = \&(7, 6, 5, 2)$

Logična funkcija ima v PKNO pri makstermih 7,6,5,2 vrednost $f_i = 0$, pri preostalih pa funkcijsko vrednost $f_i = 1$, kar pomeni, da z definicijo manjkajočih makstermov določimo vhodne kombinacije, ki bodo upoštevane v PDNO. Če želimo funkcijo zapisati v PDNO, manjkajoče maksterme M_j pretvorimo v minterme m_i z $i = 2^n - 1 - j = 7 - j$:

- manjkajoči makstermi M_j za $f_i = 1$ so: 4,3,1,0
- mintermi m_i za $f_i = 1$ so: 3,4,6,7.

Popolno disjunktivno normalno obliko zapišemo v numeričnem zapisu z dobljenimi mintermi v naraščajočem vrstnem redu in dobimo

PDNO: $f(x_1, x_2, x_3) = f^3 = \vee(3, 4, 6, 7)$.

4.3 Vaje

VAJA 4.3.1:

Zapis logičnih funkcij v PDNO in PKNO:
Za podane logične funkcije v pravilnostni tabeli zapišite PDNO in PKNO v analitični in numerični obliki.

Tabela 4.9: Funkcije v pravilnostni tabeli: $F_i(x_1, x_2, x_3)$

i	j	$x_1x_2x_3$	F_1	F_2	F_3	F_4	F_5	F_6
0	7	0 0 0	0	1	0	1	1	0
1	6	0 0 1	1	0	0	1	1	1
2	5	0 1 0	1	1	1	0	0	1
3	4	0 1 1	1	1	0	0	1	0
4	3	1 0 0	0	0	1	1	0	1
5	2	1 0 1	0	1	0	0	1	0
6	1	1 1 0	1	1	1	0	0	1
7	0	1 1 1	1	0	0	1	0	1

Analitičen zapis funkcij v PDNO:

$$
\begin{aligned}
F_1 &= \bar{x}_1\bar{x}_2x_3 \vee \bar{x}_1x_2\bar{x}_3 \vee \bar{x}_1x_2x_3 \vee x_1x_2\bar{x}_3 \vee x_1x_2x_3 \\
F_2 &= \bar{x_1}\bar{x}_2\bar{x}_3 \vee \bar{x}_1x_2\bar{x}_3 \vee \bar{x}_1x_2x_3 \vee x_1\bar{x}_2x_3 \vee x_1x_2\bar{x}_3 \\
F_3 &= \bar{x_1}x_2\bar{x}_3 \vee x_1\bar{x}_2\bar{x}_3 \vee x_1x_2\bar{x}_3 \\
F_4 &= \bar{x}_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2x_3 \vee x_1\bar{x}_2\bar{x}_3 \vee x_1x_2x_3 \\
F_5 &= \bar{x}_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2x_3 \vee \bar{x}_1x_2x_3 \vee x_1\bar{x}_2x_3 \\
F_6 &= \bar{x}_1\bar{x}_2x_3 \vee \bar{x}_1x_2\bar{x}_3 \vee x_1\bar{x}_2\bar{x}_3 \vee x_1x_2\bar{x}_3 \vee x_1x_2x_3
\end{aligned}
$$

Analitičen zapis funkcij v PKNO:

$$
\begin{aligned}
F_1 &= (x_1 \vee x_2 \vee x_3)(\bar{x}_1 \vee x_2 \vee x_3)(\bar{x}_1 \vee x_2 \vee \bar{x}_3) \\
F_2 &= (x_1 \vee x_2 \vee \bar{x}_3)(\bar{x}_1 \vee x_2 \vee x_3)(\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \\
F_3 &= (x_1 \vee x_2 \vee x_3)(x_1 \vee x_2 \vee \bar{x}_3)(x_1 \vee \bar{x}_2 \vee \bar{x}_3)(\bar{x}_1 \vee x_2 \vee \bar{x}_3)(\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \\
F_4 &= (x_1 \vee \bar{x}_2 \vee x_3)(x_1 \vee \bar{x}_2 \vee \bar{x}_3)(\bar{x}_1 \vee x_2 \vee \bar{x}_3)(\bar{x}_1 \vee \bar{x}_2 \vee x_3) \\
F_5 &= (x_1 \vee \bar{x}_2 \vee x_3) \vee (\bar{x}_1 \vee x_2 \vee x_3)(\bar{x}_1 \vee \bar{x}_2 \vee x_3)(\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \\
F_6 &= (x_1 \vee x_2 \vee x_3) \vee (x_1 \vee \bar{x}_2 \vee \bar{x}_3)(\bar{x}_1 \vee x_2 \vee \bar{x}_3)
\end{aligned}
$$

Numeričen zapis funkcij v PDNO IN PKNO:

PDNO	*PKNO*
$F_1^3 = \vee(1,2,3,6,7)$	$F_1^3 = \&(7,3,2)$
$F_2^3 = \vee(0,2,3,5,6)$	$F_2^3 = \&(6,3,0)$
$F_3^3 = \vee(2,4,6)$	$F_3^3 = \&(7,6,4,2,0)$
$F_4^3 = \vee(0,1,4,7)$	$F_4^3 = \&(5,4,2,1)$
$F_5^3 = \vee(0,1,3,5)$	$F_5^3 = \&(5,3,1,0)$
$F_6^3 = \vee(1,2,4,6,7)$	$F_6^3 = \&(7,4,2)$

VAJA 4.3.2:

Za podano logično funcijo $f^4 = \vee(3, 4, 7, 9, 12, 15)$ zapišite PKNO.

Manjkajoči mintermi m_i za $f_i = 0$ so: 0,1,2,5,6,8,10,11,13,14.
Če želimo funkcijo zapisati v PKNO, moramo manjkajoče minterme pretvoriti v istoležne maksterme. Pretvorbo mintermov m_i v maksterme M_j izvedemo z $j = 2^n - 1 - i = 15 - i$.

Makstermi M_j za $f_i = 0$ so: 15,14,13,10,9,7,5,4,2,1.
Popolno konjunktivno normalno obliko zapišemo z makstermi v padajočem vrstnem redu za znakom konjunkcije.

PKNO: $f^4 = \&(15, 14, 13, 10, 9, 7, 5, 4, 2, 1)$

VAJA 4.3.3:

Za podano logično funcijo $f^4 = \&(15, 13, 11, 10, 8, 6, 5, 4, 2, 1, 0)$ zapišite PDNO.

Manjkajoči makstermi M_j za $f_i = 1$ so: 14,12,9,7,3.
Če želimo funkcijo zapisati v PDNO, moramo manjkajoče maksterme pretvoriti v istoležne minterme. Pretvorbo makstermov M_j v minterme m_i izvedemo z $i = 2^n - 1 - j = 15 - j$.

Mintermi m_i za $f_i = 1$ so: 1,3,6,8,12.
Popolno disjunktivno normalno obliko zapišemo z mintermi v naraščajočem vrstnem redu za znakom disjunkcije.

PDNO: $f^4 = \vee(1, 3, 6, 8, 12)$

VAJA 4.3.4:

Logično funkcijo, ki je podana v disjunktivni normalni obliki zapišite v PDNO in PKNO.

$$f(x_1, x_2, x_3) \quad = \quad x_1\bar{x}_2x_3 \vee x_1x_2 \vee x_1x_3 \vee \bar{x}_1x_2\bar{x}_3$$

Logično funkcijo podano v DNO zapišemo v pravilnostno tabelo in iz nje določimo PDNO in PKNO (Tabela 4.10). Za vsako konjunkcijo določimo pri vseh vhodnih kombinacijah vrednosti 0 ali 1 in nato v zadnji koloni dobimo logično funkcijo tako, da vse kolone konjunkcij disjunktivno povežemo.

Tabela 4.10: Zapis funkcije iz DNO v pravilnostno tabelo

i	$x_1x_2x_3$	$x_1\bar{x}_2x_3$	x_1x_2	x_1x_3	$\bar{x}_1x_2\bar{x}_3$	f
0	0 0 0	0	0	0	0	0
1	0 0 1	0	0	0	0	0
2	0 1 0	0	0	0	1	1
3	0 1 1	0	0	0	0	0
4	1 0 0	0	0	0	0	0
5	1 0 1	1	0	1	0	1
6	1 1 0	0	1	0	0	1
7	1 1 1	0	1	1	0	1

PDNO: $f(x_1, x_2, x_3) = \bar{x}_1x_2\bar{x}_3 \vee x_1\bar{x}_2x_3 \vee x_1x_2\bar{x}_3 \vee x_1x_2x_3$

PKNO: $f(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3)(x_1 \vee x_2 \vee \bar{x}_3)(x_1 \vee \bar{x}_2 \vee \bar{x}_3)(\bar{x}_1 \vee x_2 \vee x_3)$

Logično funkcijo podano v DNO lahko še na drug način pretvorimo v PDNO z uporabo postulatov in izrekov. Postopek je primeren pri funkcijah z majhnim številom spremenljivk in konjunkcij.

$$\begin{aligned} f(x_1, x_2, x_3) &= x_1\bar{x}_2x_3 \vee x_1x_2(x_3 \vee \bar{x}_3) \vee x_1(x_2 \vee \bar{x}_2)x_3 \vee \bar{x}_1x_2\bar{x}_3 \\ &= x_1\bar{x}_2x_3 \vee x_1x_2x_3 \vee x_1x_2\bar{x}_3 \vee x_1x_2x_3 \vee x_1\bar{x}_2x_3 \vee \bar{x}_1x_2\bar{x}_3 \\ &= x_1\bar{x}_2x_3 \vee x_1x_2x_3 \vee x_1x_2\bar{x}_3 \vee \bar{x}_1x_2\bar{x}_3 \end{aligned}$$

VAJA 4.3.5:

Podano logično funkcijo zapišite v PDNO in PKNO.

$$f(x_1, x_2, x_3, x_4) = (x_1 \vee \bar{x}_2 \vee x_3 \vee x_4)(\bar{x}_1 \vee x_2 \vee \bar{x}_3)$$

Logična funkcija ima dve konjunktivno vezani disjunkciji, ki ju enostavno z izreki in postulati prevedemo v PKNO.

$$\begin{aligned} f(x_1, x_2, x_3, x_4) &= (x_1 \vee \bar{x}_2 \vee x_3 \vee x_4)(\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee \bar{x}_4x_4) \\ &= (x_1 \vee \bar{x}_2 \vee x_3 \vee x_4)(\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee \bar{x}_4)(\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee x_4) \end{aligned}$$

Logično funkcijo sedaj zapišimo v numerični obliki PKNO in jo nato pretvorimo v PDNO.
PKNO: $f^4 = \&(11, 5, 4)$
Manjkajoče maksterme za $f_i = 1$: 15,14,13,12,10,9,8,7,6,3,2,1,0 pretvorimo v istoležne minterme in dobimo PDNO. Pretvorbo makstermov M_j v minterme m_i izvedemo z $i = 2^n - 1 - j = 15 - j$.

Mintermi m_i za $f_i = 1$ so: 0,1,2,3,5,6,7,8,9,12,13,14,15.
PDNO: $f^4 = \vee(0, 1, 2, 3, 5, 6, 7, 8, 9, 12, 13, 14, 15)$

VAJA 4.3.6:

Podano logično funkcijo zapišite v PDNO in PKNO.

$$f(x_1, x_2, x_3, x_4) = (x_1 \downarrow x_2).\bar{x}_3 \vee ((\bar{x}_2 \equiv x_4) \downarrow \bar{x}_1)$$

Logično funkcijo pretvorimo v popolno disjunktivno normalno obliko s pretvorbo logičnih operacij NOR, OR, ekvivalence v OR, AND in NOT in postulatov ter izrekov.

$$\begin{aligned} f(x_1, x_2, x_3, x_4) &= (\bar{x}_1.\bar{x}_2).\bar{x}_3 \vee \overline{(x_2 \nabla x_4)}.\overline{\bar{x}_1} \\ &= (\bar{x}_1.\bar{x}_2).\bar{x}_3 \vee (x_2 \equiv x_4).x_1 \\ &= \bar{x}_1\bar{x}_2\bar{x}_3 \vee x_1\bar{x}_2\bar{x}_4 \vee x_1x_2x_4 \\ &= \bar{x}_1\bar{x}_2\bar{x}_3(x_4 \vee \bar{x}_4) \vee x_1\bar{x}_2(x_3 \vee \bar{x}_3)\bar{x}_4 \vee x_1x_2(x_3 \vee \bar{x}_3)x_4 \\ &= \bar{x}_1\bar{x}_2\bar{x}_3x_4 \vee \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4 \vee x_1\bar{x}_2x_3\bar{x}_4 \vee x_1\bar{x}_2\bar{x}_3\bar{x}_4 \vee x_1x_2x_3x_4 \vee x_1x_2\bar{x}_3x_4 \end{aligned}$$

Zapišimo PDNO v numeričnem zapisu in iz njega določimo PKNO.
PDNO: $f^4 = \vee(0, 1, 8, 10, 13, 15)$

Manjkajoči mintermi m_i za $f_i = 0$ so: 2,3,4,5,6,7,9,11,12,14.
Pretvorba mintermov m_i v maksterme M_j, $j = 2^n - 1 - i = 15 - i$.

Makstermi M_j za $f_i = 0$ so: 13,12,11,10,9,8,6,4,3,1.
Popolno konjunktivno normalno obliko zapišemo z makstermi v padajočem vrstnem redu za znakom konjunkcije.
PKNO: $f^4 = \&(13, 12, 11, 10, 9, 8, 6, 4, 3, 1)$

NALOGE:

4.1) Pretvorba logičnih funkcij iz PDNO v PKNO:

	PDNO	*PKNO*
1	$f^3 = \vee(0, 1, 4, 6)$	$f^3 = \&(5, 4, 2, 0)$
2	$f^3 = \vee(1, 5, 7)$	$f^3 = \&(7, 5, 4, 3, 1)$
3	$f^3 = \vee(0, 2, 4, 7)$	$f^3 = \&(6, 4, 2, 1)$
4	$f^4 = \vee(0, 1, 7, 9, 12, 14, 15)$	$f^4 = \&(13, 12, 11, 10, 9, 7, 5, 4, 2)$
5	$f^4 = \vee(3, 4, 8, 11)$	$f^4 = \&(15, 14, 13, 10, 9, 8, 6, 5, 3, 2, 1, 0)$
6	$f^4 = \vee(1, 2, 3, 4, 7, 9, 10, 12, 14)$	$f^4 = \&(15, 10, 9, 7, 4, 2, 0)$

4.2) Pretvorba logičnih funkcij iz PKNO v PDNO:

	PKNO	*PDNO*
1	$f^3 = \&(7, 4, 1, 0)$	$f^3 = \vee(1, 2, 4, 5)$
2	$f^3 = \&(5, 4, 0)$	$f^3 = \vee(0, 1, 4, 5, 6)$
3	$f^3 = \&(7, 4, 3, 1, 0)$	$f^3 = \vee(1, 2, 5)$
4	$f^4 = \&(12, 11, 9, 8, 5, 4, 3, 2, 1, 0)$	$f^4 = \vee(0, 1, 2, 5, 8, 9)$
5	$f^4 = \&(13, 11, 9, 5, 1, 0)$	$f^4 = \vee(0, 1, 3, 5, 7, 8, 9, 11, 12, 13)$
6	$f^4 = \&(15, 11, 8, 1)$	$f^4 = \vee(1, 2, 3, 5, 6, 8, 9, 10, 11, 12, 13, 15)$

4.3) Pretvorba logičnih funkcij iz DNO in KNO v PKNO in PDNO:

	DNO ali KNO	PDNO	PKNO
1.	$f(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3)$	$f^3 = \vee(1, 2, 3, 4, 5, 6, 7)$	$f^3 = \&(7)$
2.	$f(x_1, x_2, x_3) = x_1\bar{x}_2 \vee x_2x_3 \vee \bar{x}_1x_3$	$f^3 = \vee(1, 3, 4, 5, 7)$	$f^3 = \&(7, 5, 1)$
3.	$f(x_1, x_2, x_3) = (x_1 \vee x_3)(x_2 \vee \bar{x}_3)$	$f^3 = \vee(3, 4, 6, 7)$	$f^3 = \&(7, 6, 5, 2)$
4.	$f(x_1, x_2, x_3) = x_1\bar{x}_2x_3 \vee \bar{x}_3$	$f^3 = \vee(0, 2, 4, 5, 6)$	$f^3 = \&(6, 4, 0)$
5.	$f(x_1, x_2, x_3) = (x_1 \vee \bar{x}_2)(\bar{x}_1 \vee \bar{x}_3)$	$f^3 = \vee(0, 1, 4, 6)$	$f^3 = \&(5, 4, 2, 0)$
6.	$f(x_1, x_2, x_3) = x_1x_3 \vee \bar{x}_2$	$f^3 = \vee(0, 1, 4, 5, 7)$	$f^3 = \&(5, 4, 1)$

Poglavje 5

Minimizacija logičnih funkcij

Logična funkcija je v zapisu v popolni disjunktivni normalni obliki (PDNO) ali popolni konjunktivni normalni obliki (PKNO) najbolj obsežna, kar pomeni, da ima največje število logičnih operatorjev in največ vhodov. Naš cilj je poenostaviti logično funkcijo tako, da dobimo čim krajšo obliko. Postopek poenostavljanja logičnih funkcij imenujemo minimizacija in nas pripelje do minimalne disjunktivne normalne oblike (MDNO) in minimalne konjunktivne normalne oblike (MKNO) [2, 6]. Spoznali in uporabljali bomo naslednji dve metodi minimizacije logičnih funkcij:

- Veitcheva metoda,
- Quineova metoda.

Minimizacija logičnih funkcij poteka na osnovi pravila **sosednosti**, ki združi dva konjunktivna izraza dolžine k v en konjunktivni izraz dolžine $k-1$. Dva izraza sta sosedna, če je ista spremenljivka v enem izrazu negirana ($\bar{x}_i$) in v drugem izrazu nenegirana (x_i), vse ostale spremenljivke pa so enake.

Primer: Konjunktivna izraza, podana v disjunktivni obliki, $x_1x_2\bar{x}_3x_4$ in $x_1x_2x_3x_4$ sta sosedna, ker so v obeh spremenljivke x_1, x_2, x_4, četrta spremenljivka pa se v njih pojavlja kot x_3 in $\bar{x}_3$. Združevanje konjunktivnih izrazov na osnovi sosednosti ponazorimo z uporabo postulatov Boolove algebre v disjunktivni obliki za

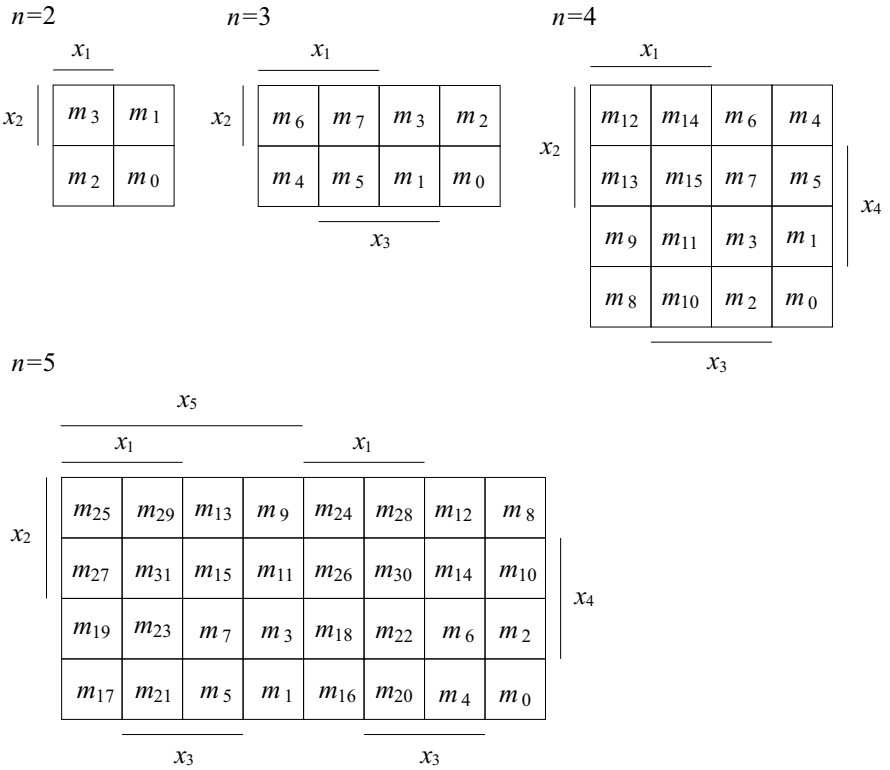
$$x_1x_2\bar{x}_3x_4 \vee x_1x_2x_3x_4 = x_1x_2x_4(\bar{x}_3 \vee x_3) = x_1x_2x_4 \ .$$

Vsebovalnik je konjuntivni izraz dolžine $k-1$, ki ga dobimo z združevanjem dveh konjunkcij dolžine k na osnovi pravila sosednosti.
Glavni vsebovalnik je tisti vsebovalnik dolžine $k-1$, ki ga na osnovi sosednosti ni mogoče združiti z nobenim drugim vsebovalnikom (konjunktivnim izrazom) iste dolžine.

5.1 Veitcheva metoda minimizacije

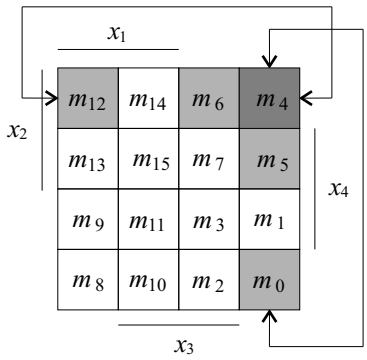
Veitcheva metoda minimizacije je grafična metoda, pri kateri funkcijo predstavimo s posebej oblikovanim diagramom. Primerna je za logične funkcije z majhnim številom spremenljivk ($n \leq 5$). Oglejmo si Veitcheve diagrame za minimizacijo logičnih funkcij z $n = 2, 3, 4, 5$ spremenljivkami (Slika 5.1). Vsaka spremenljivka v diagramu pokriva polovico diagrama, en del kot negirana spremenljivka in drug del kot nenegirana spremenljivka. Vsaka nenegirana/negirana spremenljivka pokriva tudi polovico vsake druge nenegirane

spremenljivke in polovico negirane spremenljivke. Kvadrati diagrama predstavljajo minterme s predpisano funkcijsko vrednostjo $f_i = 0$ ali $f_i = 1$ v zapisu logične funkcije. Za podano razporeditev vhodnih spremenljivk x_1,...,x_n smo v Veitchevem diagramu označili mesta mintermov m_i, $i = 0, 1, ..., 2^n - 1$, ki jih bomo uporabljali pri vpisu logičnih funkcij. Zaradi preglednosti vpisujemo v Veitchev diagram samo minterme s funkcijskimi vrednostmi 1, tiste s funkcijskimi vrednostmi 0 pa pustimo prazne. Veitchev diagram nam



Slika 5.1: Veitchevi diagrami z mintermi za n=2,3,4,5 spremenljivk

omogoča enostavno iskanje glavnih vsebovalnikov na osnovi pravila sosednosti. Sosednost konjunktivnih izrazov je direktno povezana s sosednostjo celic. Robne celice imajo sosedne celice na drugi strani diagrama, to je zgoraj ali spodaj oziroma levo ali desno. Na sliki 5.2 so za minterm m_4 označeni sosedni mintermi m_6 levo in m_{12} levo naokrog ter m_5 spodaj in m_0 spodaj naokrog.



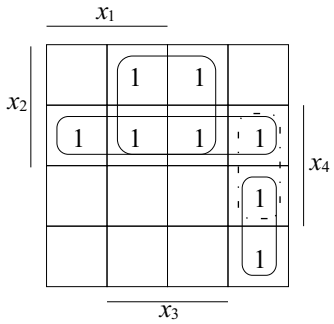
Slika 5.2: Sosednost minterma m_4

5.1.1 Minimalna disjunktivna normalna oblika (MDNO)

Postopek iskanja minimalne disjunktivne normalne oblike:

- V Veitchev diagram vpišemo enice na mesta mintermov m_i, $i = 0, 1, ..., 2^n - 1$ s funkcijskimi vrednostmi $f_i = 1$.
- Vsak minterm (enico v kvadratu diagrama) primerjamo z vsakim mintermom po sosednosti (razlikovanje i-te spremenljivke za negacijo) in določimo nove konjunktivne izraze (vsebovalnike), ki jih ponovno primerjamo po sosednosti dokler je možno. V primeru ko nimamo več sosednosti, je ta konjunktivni izraz v diagramu **glavni vsebovalnik** in ga obkrožimo. To je tak konjunktivni izraz, ki je disjunktivno vsebovan v logični funkciji tako, da za upoštevane enice v Veitchevem diagramu ne obstaja noben krajši konjunktivni izraz pri opazovanih spremenljivkah.
- Za zapis funkcije v minimalni disjunktivni normalni obliki (MDNO) poiščemo potrebne glavne vsebovalnike, ki najugodneje pokrijejo minterme v diagramu in jih disjunktivno povežemo.

Primer: Za logično funkcijo $f^4 = \vee(0, 1, 5, 6, 7, 13, 14, 15)$ poiščimo minimalno disjunktivno normalno obliko (MDNO).



V Veitchev diagram vpišemo funkcijske vrednosti 1, ki so podane z mintermi v PDNO. Z združevanjem enic na osnovi sosednosti dobimo štiri glavne vsebovalnike: x_2x_3, x_2x_4, $\bar{x}_1\bar{x}_2\bar{x}_3$, $\bar{x}_1\bar{x}_3x_4$.
Za MDNO so potrebni samo prvi trije glavni vsebovalniki, ki so obkroženi s polno črto. Glavni vsebovalnik $\bar{x}_1\bar{x}_3x_4$ ni potreben, ker sta enici že upoštevani v drugih dveh in sta označeni s prekinjeno črto.

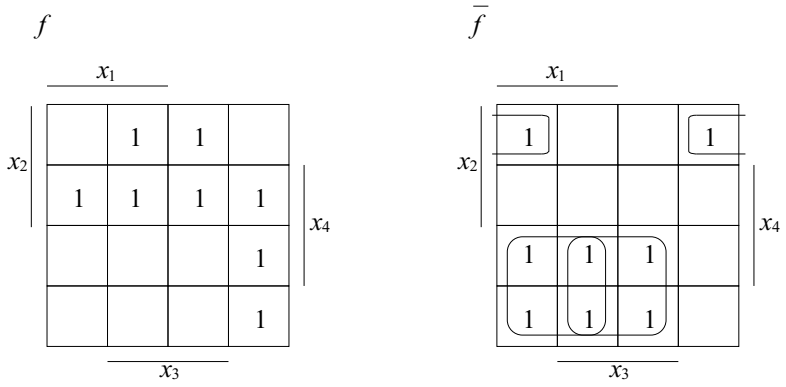
MDNO: $f(x_1, x_2, x_3, x_4) = x_2x_3 \vee x_2x_4 \vee \bar{x}_1\bar{x}_2\bar{x}_3$

5.1.2 Minimalna konjunktivna normalna oblika (MKNO)

Postopek iskanja minimalne konjunktivne normalne oblike:

- Za podano logično funkcijo zapišemo v Veitchev diagram njeno negacijo ($\bar{f}$). Funkcijske vrednosti 1 se nahajajo pri tistih mintermih m_i, ki so imeli prej $f_i = 0$.
- Negirano logično funkcijo $\bar{f}$ poenostavimo v Veitchevem diagramu in jo zapišemo v MDNO.
- MDNO negirane logične funkcije $\bar{f}$ še enkrat negiramo, razrešimo po DeMorganovem izreku in dobimo minimalno konjunktivno normalno obliko (MKNO).

Primer: Za logično funkcijo $f^4 = \vee(0, 1, 5, 6, 7, 13, 14, 15)$, ki je že zapisana v Veitchevem diagramu (f) določimo minimalno konjunktivno normalno obliko (MKNO).
Ob Veitchevem diagramu za logično funkcijo f narišemo nov Veitchev diagram negirane logične funkcije $\bar{f}$, ki ima enice tam, kjer ima funkcija f ničle.



Poiščemo MDNO negirane funkcije $\bar{f}$:
$\bar{f}$=$\bar{f}(x_1, x_2, x_3, x_4) = x_1\bar{x}_2 \vee \bar{x}_2x_3 \vee x_2\bar{x}_3\bar{x}_4$

Minimalno konjunktivno normalno obliko (MKNO) dobimo s ponovno negacijo minimalne disjunktivne normalne oblike funkcije $\bar{f}$.
$f(x_1, x_2, x_3, x_4) = \overline{\bar{f}} = \overline{x_1\bar{x}_2 \vee \bar{x}_2x_3 \vee x_2\bar{x}_3\bar{x}_4} = (\bar{x}_1 \vee x_2)(x_2 \vee \bar{x}_3)(\bar{x}_2 \vee x_3 \vee x_4)$
MKNO: $f(x_1, x_2, x_3, x_4) = (\bar{x}_1 \vee x_2)(x_2 \vee \bar{x}_3)(\bar{x}_2 \vee x_3 \vee x_4)$

5.1.3 Minimalna normalna oblika (MNO)

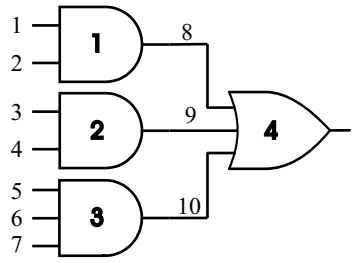
Minimalno normalno obliko (MNO) logične funkcije določimo tako, da zapišemo minimalno disjunktivno normalno obliko (MDNO) in minimalno konjunktivno normalno obliko (MKNO). Za obe obliki izračunamo potrebno število operatorjev in število vhodov.

Število operatorjev določimo tako, da preštejemo vse operatorje prvega nivoja (AND ali OR) in dodamo en operator drugega nivoja (OR ali AND).

Število vhodov določimo tako, da preštejemo vse vhodne spremenljivke, ki vstopajo v operatorje prvega nivoja in vhode v operator drugega nivoja.

Za določanje MNO pogledamo najprej število operatorjev in izberemo tisto minimalno obliko, ki ima manj operatorjev. Če je število operatorjev enako, izberemo za MNO tisto obliko, ki ima manj vhodov. V primeru enakega števila operatorjev in vhodov sta obe obliki enakovredni.

Primer: Poglejmo določanje števila operatorjev in števila vhodov za logično funkcijo v MDNO predstavljeno z logično shemo, kjer so operatorji logična vrata, vhodi pa povezave spremenljivk na vrata in medsebojne povezave vrat (Slika 5.3). Na prvem nivoju imamo tri konjunktivna vrata in na drugem nivoju ena disjunktivna vrata, kar nam da skupaj štiri logična vrata, ali štiri operatorje, če bi opazovali zapis funkcije v MDNO. Vhode preštejemo najprej na prvem nivoju, kjer vstopajo spremenljivke in nato še na drugem nivoju, kjer so izhodi konjunkcij pripeljani na vhode disjunkcije. V logični shemi imamo sedem vhodov na prvem nivoju in tri na drugem, kar nam da skupaj 10 vhodov.



Slika 5.3: MDNO - logična shema

Primer: Za logično funkcijo $f^4 = \vee(0, 1, 5, 6, 7, 13, 14, 15)$, ki je podana v MDNO in MKNO, določimo minimalno normalno obliko (MNO).

MDNO: $f(x_1, x_2, x_3, x_4) = x_2x_3 \vee x_2x_4 \vee \bar{x}_1\bar{x}_2\bar{x}_3$
MKNO:$f(x_1, x_2, x_3, x_4) = (\bar{x}_1 \vee x_2)(x_2 \vee \bar{x}_3)(\bar{x}_2 \vee x_3 \vee x_4)$

Iz zapisa MDNO in MKNO izračunamo število operatorjev in število vhodov (Tabela 5.1). V MDNO imamo tri konjunkcije na prvem in eno disjunkcijo na drugem nivoju in v MKNO tri disjunkcije na prvem in eno konjunkcijo na drugem nivoju, kar nam da za obe obliki kot rezultat štiri potrebne operatorje. V MDNO imamo sedem vhodov v prvi nivo konjunkcij in 3 vhode v drugi nivo disjunkcije, kar nam da skupaj 10 vhodov. Enako število dobimo tudi za MKNO.

Tabela 5.1: Določanje števila operatorjev in vhodov

	MDNO			MKNO		
	1.nivo	2.nivo	Vsota	1.nivo	2.nivo	Vsota
Operatorji	AND - 3	OR - 1	4	OR - 3	AND - 1	4
Vhodi	7	3	10	7	3	10

Za določitev MNO gledamo vsoto za število operatorjev in število vhodov (Tabela 5.1) pri MDNO in MKNO. Število operatorjev je za obe obliki enako (4), zato pogledamo število vhodov, ki je tudi enako (10). Obe minimalni obliki logične funkcije sta enakovredni, zato je MNO = MDNO = MKNO.

5.1.4 Minimizacija nepopolnih logičnih funkcij

O nepopolnih logičnih funkcijah ali funkcijah z redundancami govorimo takrat, kadar imamo takšne vhodne kombinacije, ki se pri normalnem delovanju nikoli ne pojavijo na vhodu vezja. Funkcijske vrednosti v takem primeru niso vnaprej določene z nobeno od binarnih vrednosti $(0, 1)$, zato jih v pravilnostni tabeli ali Veitchevem diagramu označimo z $\times$. Ker je redundančnim vhodnim kombinacijam pripisana vrednost minterma m_i=0, dobimo v popolni disjunktivni normalni obliki izraz $m_i.f_i = 0$, ki omogoča, da je funkcijska vrednost $f_i = 0$ ali $f_i = 1$.
Oglejmo si zapis nepopolnih logičnih funkcij v pravilnostni tabeli in numerični obliki ter postopek minimizacije v Veitchevem diagramu.

Primer: Logična funkcija (Tabela 5.2) ima pri dveh vhodnih kombinacijah ($i = 2, 6$) nedoločeno funkcijsko vrednost.

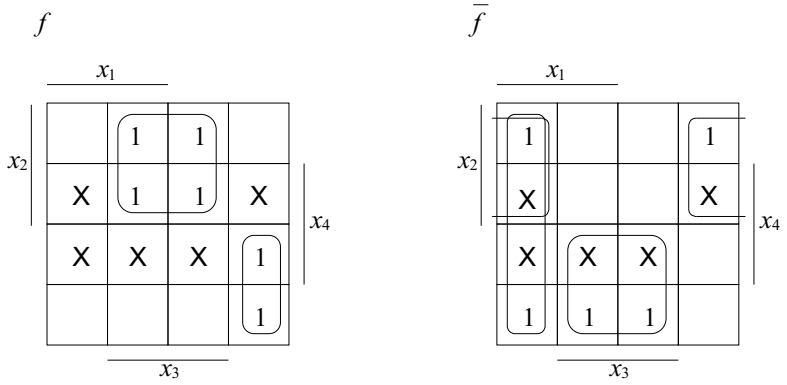
Tabela 5.2: Nepopolna logična funkcija

i	j	$x_1x_2x_3$	f
0	7	0 0 0	0
1	6	0 0 1	1
2	5	0 1 0	×
3	4	0 1 1	0
4	3	1 0 0	1
5	2	1 0 1	0
6	1	1 1 0	×
7	0	1 1 1	1

Redundančne vhodne kombinacije v numeričnem zapisu podamo tako, da operatorju konjunkcije ali disjunkcije pripišemo redundančno oznako × kot indeks k operatorju, ki določa minterme ali maksterme:
$f^3 = \vee(1,4,7); \vee_\times(2,6)$ ali $f^3 = \vee(1,4,7); \&_\times(5,1)$
$f^3 = \&(7,4,2); \&_\times(5,1)$ ali $f^3 = \&(7,4,2); \vee_\times(2,6)$

Oglejmo si minimizacijo nepopolnih logičnih funkcij. V Veitchev diagram vpišemo minterme s funkcijskimi vrednostmi 1 in redundancami ×. V postopku minimizacije redundančnim vhodnim kombinacijam določimo takšne funkcijske vrednosti, ki nam pri iskanju glavnih vsebovalnikov dajejo najkrajše konjunktivne izraze.

Primer: Za logično funkcijo z redundantnimi vhodnimi kombinacijami poiščimo MDNO, MKNO in določimo MNO.
$f^4 = \vee(0,1,6,7,14,15), \vee_\times(3,5,9,11,13)$



V Veitchev diagram za funkcijo f vpišemo funkcijske vrednosti 1 in redundance ter poiščemo MDNO. V minimizacijskem postopku redundance nismo upoštevali v nobenem glavnem vsebovalniku, zato imajo vrednost 0.
MDNO: $f(x_1,x_2,x_3,x_4) = x_2x_3 \vee \bar{x}_1\bar{x}_2\bar{x}_3$

V drug Veitchev diagram vpišemo negirano funkcijo $\bar{f}$, ki ima redundance na istih mestih kot f in funkcijske vrednosti 1, tam kjer smo prej imeli 0. Poiščemo najprej MDNO negirane funkcije, kjer so vse redundance upoštevane v glavnih vsebovalnikih. Izračunamo še MKNO tako, da še enkrat negiramo MDNO negirane funkcije in jo razrešimo po DeMorganovem izreku.

$\bar{f} = x_1\bar{x}_3 \vee x_2\bar{x}_3 \vee \bar{x}_2x_3$

$f(x_1, x_2, x_3, x_4) = \overline{\overline{f}} = \overline{x_1\bar{x}_3 \vee x_2\bar{x}_3 \vee \bar{x}_2 x_3} = (\bar{x}_1 \vee x_3)(\bar{x}_2 \vee x_3)(x_2 \vee \bar{x}_3)$

MKNO: $f(x_1, x_2, x_3, x_4) = (\bar{x}_1 \vee x_3)(\bar{x}_2 \vee x_3)(x_2 \vee \bar{x}_3)$

Tabela 5.3: Določanje števila operatorjev in vhodov

	MDNO			MKNO		
	1.nivo	2.nivo	Vsota	1.nivo	2.nivo	Vsota
Operatorji	AND - 2	OR - 1	3	OR - 3	AND - 1	4
Vhodi	5	2	7	6	3	9

Za logično funkcijo z redundancami ugotovimo še MNO iz izračunanega števila operatorjev in vhodov (Tabela 5.3). Število operatorjev za MDNO (3) je manjše kot za MKNO (4), zato je minimalna normalna oblika MNO = MDNO.

5.2 Quineova metoda minimizacije

Postopek:

- Logično funkcijo zapišemo v **tabelo glavnih vsebovalnikov**. V prvem stolpcu n so vsi mintermi m_i, ki imajo funkcijsko vrednost $f_i = 1$.
- Glavne vsebovalnike poiščemo tako, da po pravilu sosednosti primerjamo vsak minterm v stolpcu z vsemi ostalimi. Dva sosedna minterma označimo z $\surd$, izpustimo spremenljivko, ki se nahaja v x_i in $\bar{x}_i$ obliki in nov konjunktivni izraz vpišemo v stolpec dolžine $n-1$. V nadaljevanju upoštevamo za iskanje sosednosti tudi že označene minterme. Vsak minterm ima n sosedov, zato je lahko v minimizaciji večkrat uporabljen. Ko smo pregledali celoten stolpec mintermov, nadaljujemo iskanje sosednosti z izrazi dolžine $n-1$ in dobimo nove konjunktivne izraze v stolpcu izrazov $n-2$. Postopek ponavljamo vse dokler obstojajo sosedne konjunkcije oziroma do stolpca izrazov z eno spremenljivko. Vsi izrazi, ki ostanejo v tabeli neoznačeni, so glavni vsebovalniki.
- Napravimo novo **tabelo pokritij** tako, da v vrstice vpišemo glavne vsebovalnike, v stolpce pa minterme. V njej označimo pripadnost ali pokritje mintermov z dobljenimi glavnimi vsebovalniki. Vsak minterm, ki je vsebovan v glavnem vsebovalniku, označimo z znakom vsebovanja ($\surd$).
- Iz tabele določimo potrebne glavne vsebovalnike po naslednjem pravilu: stolpec, ki vsebuje en sam znak vsebovanja pove, da je pripadajoči glavni vsebovalnik potreben v MDNO, zato ga obkrožimo. Minterm v tem stolpcu označimo s +, vse druge minterme, ki imajo znak vsebovanja pri označenem potrebnem vsebovalniku oziroma v isti vrstici pa z −. To ponovimo za vse stolpce z enim samim znakom vsebovanja ($\surd$). Ko smo končali, pogledamo ali so vsi mintermi že pokriti z označenimi glavnimi vsebovalniki (vsak minterm ima znak + ali −). Če ta pogoj ni izpolnjen nadaljujemo z naslednjim korakom.
- Narišemo novo tabelo z neoznačenimi mintermi (v vsakem stolpcu imajo dva ali več znakov vsebovanja) in še neuporabljenimi glavnimi vsebovalniki. V primeru enakih stolpcev izpišemo samo enega. Med glavnimi vsebovalniki v novi tabeli izberemo tiste, ki najugodneje pokrijejo vse preostale minterme, kar pomeni, da upoštevamo

število spremenljivk v glavnih vsebovalnikih. Če je v tabeli pokritij ostal en sam nepokrit minterm z dvema znakoma vsebovanja, lahko kar tam izberemo enega od glavnih vsebovalnikov in ga obkrožimo.

Primer: Za logično funkcijo $f^4 = \vee(0, 1, 5, 6, 7, 13, 14, 15)$ poiščimo minimalno disjunktivno normalno obliko (MDNO) s Quineovo metodo minimizacije.

V stolpec n vpišemo vse minterme s funkcijsko vrednostjo 1 in poiščemo glavne vsebovalnike (Tabela 5.4).

Tabela 5.4: Glavni vsebovalniki

n		$n-1$		$n-2$
$\bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4$	$\surd$	$\bar{x}_1\bar{x}_2\bar{x}_3$		x_2x_4
$\bar{x}_1\bar{x}_2\bar{x}_3x_4$	$\surd$	$\bar{x}_1\bar{x}_3x_4$		x_2x_3
$\bar{x}_1x_2\bar{x}_3x_4$	$\surd$	$\bar{x}_1x_2x_4$	$\surd$	
$\bar{x}_1x_2x_3\bar{x}_4$	$\surd$	$x_2\bar{x}_3x_4$	$\surd$	
$\bar{x}_1x_2x_3x_4$	$\surd$	$\bar{x}_1x_2x_3$	$\surd$	
$x_1x_2\bar{x}_3x_4$	$\surd$	$x_2x_3\bar{x}_4$	$\surd$	
$x_1x_2x_3\bar{x}_4$	$\surd$	$x_2x_3x_4$	$\surd$	
$x_1x_2x_3x_4$	$\surd$	$x_1x_2x_4$	$\surd$	
		$x_1x_2x_3$	$\surd$	

Začnemo s stolpcem n, primerjamo minterme vsakega z vsakim in jih na osnovi sosednosti združujemo v konjunktivne izraze dolžine $n-1$. Pogledamo prva dva minterma $\bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4$ in $\bar{x}_1\bar{x}_2\bar{x}_3x_4$ v stolpcu n. Ugotovimo, da sta sosedna po spremenljivki x_4, ju označimo z znakom vsebovanja ($\surd$) in vpišemo v stolpec $n-1$ konjunkcijo $\bar{x}_1\bar{x}_2\bar{x}_3$. Iskanje sosednih mintermov nadaljujemo vse do zadnjih dveh mintermov v stolpcu in vsi mintermi združeni na osnovi sosednosti upoštevani v stolpcu $n-1$, so označeni z znakom vsebovanja ($\surd$). Postopek primerjave in združevanja ponovimo s konjunkcijami v stolpcu $n-1$ in določamo stolpec $n-2$, itd. V stolpcu $n-2$ smo dobili dva konjunktivna izraza, ki nista sosedna in postopek iskanja glavnih vsebovalnikov je končan. Vsi konjunktivni izrazi, ki nimajo oznake $\surd$ so glavni vsebovalniki in jih vpišemo v začetno tabelo pokritij (Tabela 5.5) skupaj z mintermi, ki imajo funkcijske vrednosti 1. Za vsak minterm logične funkcije označimo njegovo pokritje $\surd$ v enem ali več glavnih vsebovalnikih. S tabelo pokritij bomo določili potrebne glavne vsebovalnike in jih označili z oklepajem ( ).

Tabela 5.5: Začetna tabela pokritij

	0	1	5	6	7	13	14	15
$\bar{x}_1\bar{x}_2\bar{x}_3$	$\surd$	$\surd$						
$\bar{x}_1\bar{x}_3x_4$		$\surd$	$\surd$					
x_2x_4			$\surd$		$\surd$	$\surd$		$\surd$
x_2x_3				$\surd$	$\surd$		$\surd$	$\surd$

Iz začetne tabele pokritij ugotovimo, da ima stolpec pri mintermu m_0 en sam znak vsebovanja, zato je glavni vsebovalnik $\bar{x}_1\bar{x}_2\bar{x}_3$ potreben. Označimo ga z oklepajem, znak vsebovanja pri mintermu 0 s '+', ostale znake vsebovanja v tej vrstici pa z '−' (Tabela 5.6). Isto ponovimo za stolpec pri mintermu m_6 in glavnem vsebovalniku x_2x_3 (Tabela 5.7) ter mintermu m_{13} in glavnem vsebovalniku x_2x_4 (Tabela 5.8).

Tabela 5.6: Tabela pokritij, 1 korak

	0	1	5	6	7	13	14	15
$(\bar{x}_1\bar{x}_2\bar{x}_3)$	$\surd +$	$\surd -$						
$\bar{x}_1\bar{x}_3x_4$		$\surd$	$\surd$					
x_2x_4			$\surd$		$\surd$	$\surd$		$\surd$
x_2x_3				$\surd$	$\surd$		$\surd$	$\surd$

Tabela 5.7: Tabela pokritij, 2 korak

	0	1	5	6	7	13	14	15
$(\bar{x}_1\bar{x}_2\bar{x}_3)$	$\surd +$	$\surd -$						
$\bar{x}_1\bar{x}_3x_4$		$\surd$	$\surd$					
x_2x_4			$\surd$		$\surd$	$\surd$		$\surd$
(x_2x_3)				$\surd +$	$\surd -$		$\surd -$	$\surd -$

Tabela 5.8: Tabela pokritij, 3 korak

	0	1	5	6	7	13	14	15
$(\bar{x}_1\bar{x}_2\bar{x}_3)$	$\surd +$	$\surd -$						
$\bar{x}_1\bar{x}_3x_4$		$\surd$	$\surd$					
(x_2x_4)			$\surd -$		$\surd -$	$\surd +$		$\surd -$
(x_2x_3)				$\surd +$	$\surd -$		$\surd -$	$\surd -$

Sedaj nimamo nobenega stolpca več z enim samim znakom vsebovanja, zato pogledamo ali smo z označenimi glavnimi vsebovalniki pokrili vse minterme. Minterm je pokrit z enim od glavnih vsebovalnikov, če ima pri oznaki $\surd$ še oznako $+$ ali $-$. Ugotavljanje pokritja mintermov:

- glavni vsebovalnik $\bar{x}_1\bar{x}_2\bar{x}_3$ pokrije minterma m_0 in m_1
- glavni vsebovalnik x_2x_4 pokrije minterme m_5, m_7, m_{13}, m_{15}
- glavni vsebovalnik x_2x_3 pokrije minterme m_6, m_7, m_{14}, m_{15}

Iz zapisa lahko vidimo, da z zgornjimi tremi glavnimi vsebovalniki pokrijemo vse minterme (v vsakem stolpcu imamo vsaj en znak $+$ ali en znak $-$), ki jih disjunktivno povežemo in dobimo minimalno disjunktivno normalno obliko (MDNO).

MDNO: $f(x_1, x_2, x_3, x_4) = x_2x_3 \vee x_2x_4 \vee \bar{x}_1\bar{x}_2\bar{x}_3$.

Postopek iskanja MKNO s Quineovo metodo

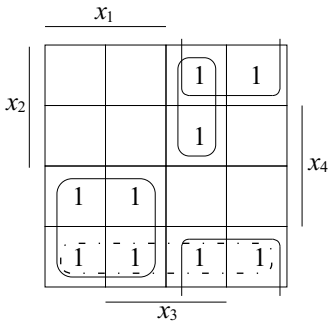
- Za logično funkcijo v PDNO poiščemo negirano funkcijo $\bar{f}$.
- Postopek minimizacije negirane funkcije izvedemo po Quineovi metodi minimizacije za MDNO.
- Minimalno disjunktivno normalno obliko negirane funkcije še enkrat negiramo, razrešimo po DeMorganovem izreku in rezultat je minimalna konjunktivna normalna oblika (MKNO).

5.3 Vaje

VAJA 5.3.1:

Zapišite logično funkcijo $f^4 = \&(14, 12, 10, 3, 2, 1, 0)$ v minimalni disjunktivni normalni obliki (MDNO).

Najprej zapišemo funkcijo v PDNO, tako da poiščemo manjkajoče maksterme M_j za $f_i = 0$: (15,13,11,9,8,7,6,5,4), jih pretvorimo v minterme m_i, $i = 15 - j$ in dobimo funkcijo v PDNO $f^4 = \vee(0, 2, 4, 6, 7, 8, 9, 10, 11)$. Dobljeno logično funkcijo zapišemo v Veitchev diagram in jo minimiziramo.



Z združevanjem enic dobimo glavne vsebovalnike: $x_1\bar{x}_2$, $\bar{x}_2\bar{x}_4$, $\bar{x}_1\bar{x}_4$, $\bar{x}_1x_2x_3$.

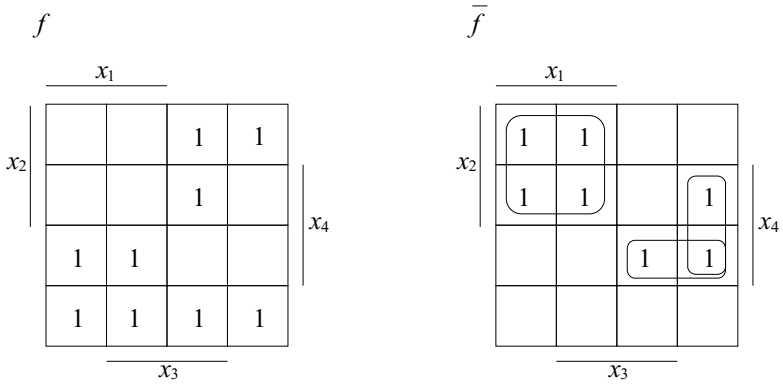
V Veitchevem diagramu imamo štiri glavne vsebovalnike. Za MDNO so potrebni samo trije glavni vsebovalniki (označeni s polno črto), ker so štiri enice iz glavnega vsebovalnika $\bar{x}_2\bar{x}_4$ (označen s prekinjeno črto) že upoštevane v drugih dveh glavnih vsebovalnikih.

MDNO: $f(x_1, x_2, x_3, x_4) = x_1\bar{x}_2 \vee \bar{x}_1\bar{x}_4 \vee \bar{x}_1x_2x_3$

VAJA 5.3.2:

Zapišite MKNO za logično funkcijo $f^4 = \vee(0, 2, 4, 6, 7, 8, 9, 10, 11)$.

Za določanje MKNO bomo zapisali funkciji f in $\bar{f}$ v Veitchev diagram.



Poiščemo najprej MDNO negirane logične funkcije $\bar{f}$, jo še enkrat negiramo in razrešimo z DeMorganovim izrekom ter dobimo MKNO:

$$\bar{f} \;=\; \bar{f}(x_1, x_2, x_3, x_4) = x_1x_2 \vee \bar{x}_1\bar{x}_3x_4 \vee \bar{x}_1\bar{x}_2x_4$$

$$\begin{aligned} f(x_1,x_2,x_3,x_4) &= \overline{\overline{f}} = \overline{x_1x_2 \vee \bar{x}_1\bar{x}_3x_4 \vee \bar{x}_1\bar{x}_2x_4} \\ &= (\bar{x}_1 \vee \bar{x}_2)(x_1 \vee x_3 \vee \bar{x}_4)(x_1 \vee x_2 \vee \bar{x}_4) \end{aligned}$$

MKNO: $f(x_1,x_2,x_3,x_4) = (\bar{x}_1 \vee \bar{x}_2)(x_1 \vee x_3 \vee \bar{x}_4)(x_1 \vee x_2 \vee \bar{x}_4)$

VAJA 5.3.3:

Za logično funkcijo $f^4 = \&(14,12,10,3,2,1,0)$ zapisano v MDNO in MKNO določite minimalno normalno obliko (MNO).
MDNO: $f(x_1,x_2,x_3,x_4) = x_1\bar{x}_2 \vee \bar{x}_1\bar{x}_4 \vee \bar{x}_1x_2x_3$
MKNO:$f(x_1,x_2,x_3,x_4) = (\bar{x}_1 \vee \bar{x}_2)(x_1 \vee x_3 \vee \bar{x}_4)(x_1 \vee x_2 \vee \bar{x}_4)$

Za MDNO in MKNO določimo število operatorjev in število vhodov (Tabela 5.9).

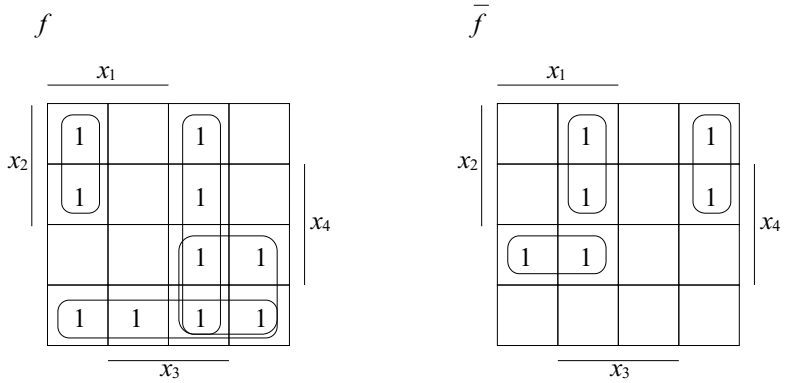
Tabela 5.9: Določanje operatorjev in vhodov

	MDNO			MKNO		
	1.nivo	2.nivo	Vsota	1.nivo	2.nivo	Vsota
Operatorji	AND - 3	OR - 1	4	OR - 3	AND - 1	4
Vhodi	7	3	10	8	3	11

Število operatorjev za MDNO (4) je enako kot za MKNO (4), število vhodov pa je pri MDNO (10) manjše kot pri MKNO (11), zato je minimalna normalna oblika MNO = MDNO.

VAJA 5.3.4:

Zapišite logično funkcijo $f^4 = \vee(0,1,2,3,6,7,8,10,12,13)$ v minimalni disjunktivni normalni obliki (MDNO), v minimalni konjunktivni normalni obliki (MKNO) in določite minimalno normalno obliko (MNO).



MDNO: $f(x_1,x_2,x_3,x_4) = \bar{x}_1x_3 \vee \bar{x}_1\bar{x}_2 \vee \bar{x}_2\bar{x}_4 \vee x_1x_2\bar{x}_3$
Poiščimo MKNO podane logične funkcije.

$$\begin{aligned} \bar{f} &= \bar{x}_1x_2\bar{x}_3 \vee x_1x_2x_3 \vee x_1\bar{x}_2x_4 \\ f(x_1,x_2,x_3,x_4) = \overline{\bar{f}} &= \overline{x_1x_2\bar{x}_3 \vee x_1x_2x_3 \vee x_1\bar{x}_2x_4} \\ &= (x_1 \vee \bar{x}_2 \vee x_3)(\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)(\bar{x}_1 \vee x_2 \vee \bar{x}_4) \end{aligned}$$

MKNO: $f(x_1,x_2,x_3,x_4) = (x_1 \vee \bar{x}_2 \vee x_3)(\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)(\bar{x}_1 \vee x_2 \vee \bar{x}_4)$

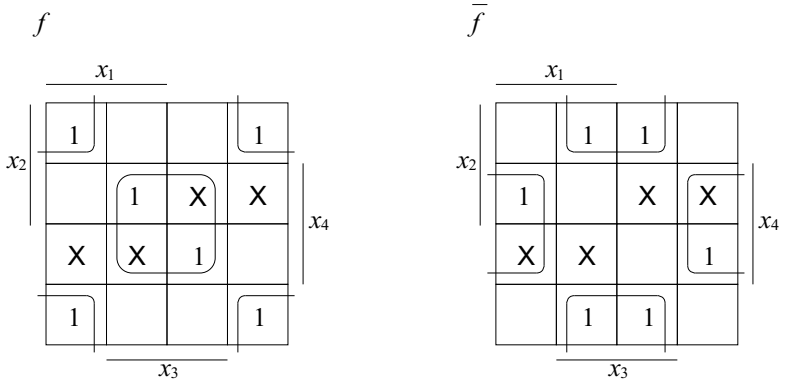
Tabela 5.10: Določanje operatorjev in vhodov

	MDNO			MKNO		
	1.nivo	2.nivo	Vsota	1.nivo	2.nivo	Vsota
Operatorji	AND - 4	OR - 1	5	OR - 3	AND - 1	4
Vhodi	9	4	13	9	3	12

Določimo število operatorjev in vhodov za MDNO in MKNO (Tabela 5.10).
Število operatorjev za MKNO (4) je manjše kot za MDNO (5), zato je minimalna normalna oblika MNO = MKNO.

VAJA 5.3.5:

Zapišite logično funkcijo z redundancami $f^4 = \vee(0,3,4,8,12,15)$, $\vee_\times(5,7,9,11)$ v minimalni disjunktivni normalni obliki (MDNO), v minimalni konjunktivni normalni obliki (MKNO) in določite minimalno normalno obliko (MNO).



MDNO: $f(x_1, x_2, x_3, x_4) = \bar{x}_3\bar{x}_4 \vee x_3x_4$

Poiščimo MKNO podane logične funkcije.

$$\begin{aligned} \bar{f} &= x_3\bar{x}_4 \vee \bar{x}_3x_4 \\ f(x_1, x_2, x_3, x_4) = \overline{\bar{f}} &= \overline{x_3\bar{x}_4 \vee \bar{x}_3x_4} \\ &= (\bar{x}_3 \vee x_4)(x_3 \vee \bar{x}_4) \end{aligned}$$

MKNO: $f(x_1, x_2, x_3, x_4) = (\bar{x}_3 \vee x_4)(x_3 \vee \bar{x}_4)$

Določimo število operatorjev in vhodov za MDNO in MKNO (Tabela 5.11).

Tabela 5.11: Določanje operatorjev in vhodov

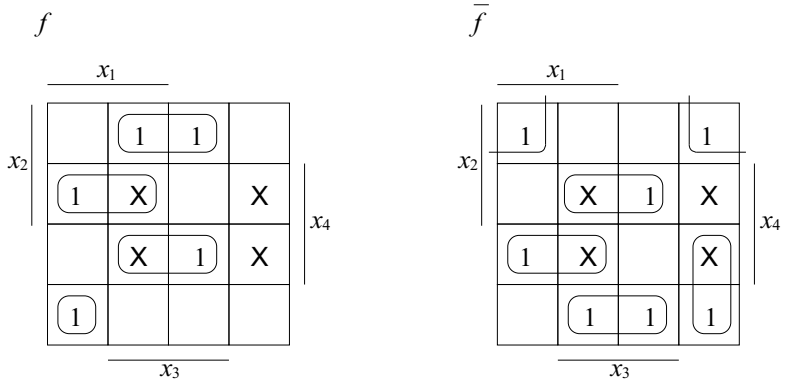
	MDNO			MKNO		
	1.nivo	2.nivo	Vsota	1.nivo	2.nivo	Vsota
Operatorji	AND - 2	OR - 1	3	OR - 2	AND - 1	3
Vhodi	4	2	6	4	2	6

Število operatorjev za MDNO (3) je enako kot za MKNO (3), število vhodov je tudi enako, zato je minimalna normalna oblika MNO = MDNO = MKNO.

VAJA 5.3.6:

Zapišite logično funkcijo z redundancami $f^4 = \&(15, 13, 11, 8, 6, 5, 3)$, $m_1 = 0$, $m_5 = 0$, $m_{11} = 0$, $m_{15} = 0$ v minimalni disjunktivni normalni obliki (MDNO), v minimalni konjunktivni normalni obliki (MKNO) in določite minimalno normalno obliko (MNO).

Najprej zapišemo funkcijo v PDNO, tako da poiščemo manjkajoče maksterme M_j za $f_i = 1$: (14,12,10,9,7,4,2,1,0), jih pretvorimo v minterme m_i, $i = 15 - j$ in dobimo funkcijo v PDNO $f^4 = \vee(1, 3, 5, 6, 8, 11, 13, 14, 15)$, če ne bi upoštevali redundanc. Pri upoštevanju redundanc moramo izločiti minterme m_i: 1,5,11,15 in PDNO logične funkcije z redundancami je $f^4 = \vee(3, 6, 8, 13, 14)$, $\vee_\times(1, 5, 11, 15)$.



MDNO: $f(x_1, x_2, x_3, x_4) = x_2x_3\bar{x}_4 \vee x_1x_2x_4 \vee \bar{x}_2x_3x_4 \vee x_1\bar{x}_2\bar{x}_3\bar{x}_4$

Poiščimo MKNO podane logične funkcije.

$$\begin{aligned} \bar{f} &= \bar{x}_1\bar{x}_2\bar{x}_3 \vee x_2\bar{x}_3\bar{x}_4 \vee x_2x_3x_4 \vee x_1\bar{x}_2x_4 \vee \bar{x}_2x_3\bar{x}_4 \\ f(x_1, x_2, x_3, x_4) = \overline{\bar{f}} &= \overline{\bar{x}_1\bar{x}_2\bar{x}_3 \vee x_2\bar{x}_3\bar{x}_4 \vee x_2x_3x_4 \vee x_1\bar{x}_2x_4 \vee \bar{x}_2x_3\bar{x}_4} \\ &= (x_1 \vee x_2 \vee x_3)(\bar{x}_2 \vee x_3 \vee x_4)(\bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4) \\ &\quad (\bar{x}_1 \vee x_2 \vee \bar{x}_4)(x_2 \vee \bar{x}_3 \vee x_4) \end{aligned}$$

MKNO: $f(x_1, x_2, x_3, x_4) = (x_1 \vee x_2 \vee x_3)(\bar{x}_2 \vee x_3 \vee x_4)(\bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4)(\bar{x}_1 \vee x_2 \vee \bar{x}_4)(x_2 \vee \bar{x}_3 \vee x_4)$

Tabela 5.12: Določanje MNO

	MDNO			MKNO		
	1.nivo	2.nivo	Vsota	1.nivo	2.nivo	Vsota
Operatorji	AND - 4	OR - 1	5	OR - 5	AND - 1	6
Vhodi	13	4	17	15	5	20

Število operatorjev za MDNO (5) je manjše kot za MKNO (6), zato je minimalna normalna oblika MNO = MDNO.

VAJA 5.3.7:

Poiščite MDNO za funkcijo $f^3 = \vee(0, 1, 3, 5, 7)$ s Quineovo metodo minimizacije.

V stolpec n vpišemo minterme s funkcijsko vrednostjo 1 in poiščemo glavne vsebovalnike (Tabela 5.13).

Tabela 5.13: Glavni vsebovalniki

n		$n-1$		n-2
$\bar{x}_1\bar{x}_2\bar{x}_3$	$\surd$	$\bar{x}_1\bar{x}_2$		x_3
$\bar{x}_1\bar{x}_2x_3$	$\surd$	$\bar{x}_1x_3$	$\surd$	
$\bar{x}_1x_2x_3$	$\surd$	$\bar{x}_2x_3$	$\surd$	
$x_1\bar{x}_2x_3$	$\surd$	x_2x_3	$\surd$	
$x_1x_2x_3$	$\surd$	x_1x_3	$\surd$	

V tabelo pokritij (Tabela 5.14) vpišemo vse glavne vsebovalnike in minterme ter označimo njihovo pokritje.

Tabela 5.14: Tabela pokritij

	0	1	3	5	7
$(\bar{x}_1\bar{x}_2)$	$\surd +$	$\surd -$			
(x_3)		$\surd -$	$\surd +$	$\surd -$	$\surd -$

Stolpec pri mintermu m_0 ima en sam znak vsebovanja, zato je glavni vsebovalnik $\bar{x}_1\bar{x}_2$ potreben. Označimo stolpec minterma m_0 s $+$ in vse znake vsebovanja v vrstici z $-$. Isto ponovimo za kolono pri mintermu m_3 in dobimo, da je drugi potrebni glavni vsebovalnik x_3. V tem primeru sta oba glavna vsebovalnika že potrebna, zato samo preverimo ali smo pokrili vse minterme:

- glavni vsebovalnik $\bar{x}_1\bar{x}_2$ pokrije minterma m_0 in m_1
- glavni vsebovalnik x_3 pokrije minterme m_1, m_3, m_5, m_7

Iz zapisa lahko vidimo, da z zgornjima glavnima vsebovalnikoma pokrijemo vse minterme, zato je minimalna disjunktivna normalna oblika:

MDNO: $f(x_1, x_2, x_3) = x_3 \vee \bar{x}_1\bar{x}_2$.

VAJA 5.3.8:

Poiščite MDNO za funkcijo $f^3 = \vee(0, 3, 4, 5, 7)$ s Quineovo metodo minimizacije.

V kolono n vpišemo minterme s funkcijsko vrednostjo 1 in poiščemo glavne vsebovalnike (Tabela 5.15).

Tabela 5.15: Glavni vsebovalniki

n		$n-1$
$\bar{x}_1\bar{x}_2\bar{x}_3$	$\surd$	$\bar{x}_2\bar{x}_3$
$\bar{x}_1x_2x_3$	$\surd$	x_2x_3
$x_1\bar{x}_2\bar{x}_3$	$\surd$	$x_1\bar{x}_2$
$x_1\bar{x}_2x_3$	$\surd$	x_1x_3
$x_1x_2x_3$	$\surd$	

V tabelo pokritij (Tabela 5.16) vpišemo vse glavne vsebovalnike in minterme ter označimo njihovo pokrivanje.

Tabela 5.16: Tabela pokritij

	0	3	4	5	7
$(\bar{x}_2\bar{x}_3)$	$\surd$ +		$\surd$ -		
(x_2x_3)		$\surd$ +			$\surd$ -
$x_1\bar{x}_2$			$\surd$	$\surd$	
x_1x_3				$\surd$	$\surd$

Kolona pri mintermu m_0 ima en sam znak vsebovanja, zato je glavni vsebovalnik $\bar{x}_2\bar{x}_3$ potreben. Označimo kolono minterma 0 s + in vse znake vsebovanja v vrstici z −. Isto ponovimo za kolono pri mintermu m_3 in dobimo, da je drugi potrebni glavni vsebovalnik x_2x_3. Ko nimamo nobene kolone več z enim samim znakom vsebovanja, pogledamo ali smo z označenimi glavnimi vsebovalniki pokrili vse minterme:

- glavni vsebovalnik $\bar{x}_2\bar{x}_3$ pokrije minterma m_0 in m_4
- glavni vsebovalnik x_2x_3 pokrije minterma m_3 in m_7

Iz zapisa lahko vidimo, da z zgornjima glavnima vsebovalnikoma ne pokrijemo minterma m_5, ki ima dva znaka vsebovanja brez oznake + ali −. Narišemo novo tabelo za minterm m_5 in glavna vsebovalnika $x_1\bar{x}_2$, x_1x_3. Ker sta oba glavna vsebovalnika v tabeli enake dolžine, dobimo dve enakovredni rešitvi za MDNO (Tabela 5.17).

Tabela 5.17: Izbira potrebnega glavnega vsebovalnika

	5
$x_1\bar{x}_2$	$\surd$
x_1x_3	$\surd$

a) MDNO: $f(x_1, x_2, x_3) = \bar{x}_2\bar{x}_3 \vee x_2x_3 \vee x_1\bar{x}_2$ in
b) MDNO: $f(x_1, x_2, x_3) = \bar{x}_2\bar{x}_3 \vee x_2x_3 \vee x_1x_3$.

NALOGE

5.1) $f^4 = \vee(2, 3, 6, 7, 8, 11, 12, 14, 15)$

MDNO: $f(x_1, x_2, x_3, x_4) = \bar{x}_1x_3 \vee x_3x_4 \vee x_2x_3 \vee x_1\bar{x}_3\bar{x}_4$

MKNO: $f(x_1, x_2, x_3, x_4) = (x_1 \vee x_3)(x_3 \vee \bar{x}_4)(\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee x_4)$

MDNO: (5,13), MKNO: (4,11) $\Rightarrow$ MNO = MKNO

5.2) $f^4 = \&(13, 9, 8, 7, 5, 4, 3, 1)$

MDNO: $f(x_1, x_2, x_3, x_4) = \bar{x}_3x_4 \vee x_1x_2x_4 \vee \bar{x}_1\bar{x}_2x_4 \vee \bar{x}_1\bar{x}_3$

MKNO:$f(x_1, x_2, x_3, x_4) = (\bar{x}_1 \vee x_4)(\bar{x}_3 \vee x_4)(\bar{x}_1 \vee x_2 \vee \bar{x}_3)(x_1 \vee \bar{x}_2 \vee \bar{x}_3)$

MDNO: (5,14), MKNO: (5,14) $\Rightarrow$ MNO = MDNO = MKNO

Poglavje 6

Dvo- in večnivojske logične funkcije

Ogledali si bomo zapise in realizacije dvo- in večnivojskih logičnih funkcij z različnimi nabori operatorjev. Za elementarni nabor logičnih operatorjev konjunkcije, disjunkcije in negacije, smo že ugotovili, da je možno z njim zapisati vsako logično funkcijo. Ker želimo uporabiti tudi Shefferjeve operatorje, Pierceove operatorje in operator seštevanja po modulu 2, moramo za želeni nabor pokazati, da je funkcijsko poln sistem. Nabor logičnih operatorjev ali funkcij je funkcijsko poln, če je možno z njim zapisati negacijo, konjunkcijo in disjunkcijo.

6.1 Funkcijsko polni sistemi

Funkcijsko poln sistem nam zagotavlja, da z njim lahko zapišemo vsako logično funkcijo z n vhodnimi spremenljivkami. Najbolj pogosto uporabljeni funkcijsko polni sistemi so:

1) **(AND, NOT)**

   Dokaz: Zapis disjunkcije s konjunkcijo in negacijo:

   $$x_1 \vee x_2 = \overline{\overline{x_1 \vee x_2}} = \overline{\bar{x}_1 \bar{x}_2}$$

2) **(OR, NOT)**

   Dokaz: Zapis konjunkcije z disjunkcijo in negacijo

   $$x_1 x_2 = \overline{\overline{x_1 x_2}} = \overline{\bar{x}_1 \vee \bar{x}_2}$$

3) **(NAND)** - $x_1 \uparrow x_2 = \overline{x_1 x_2} = \bar{x}_1 \vee \bar{x}_2$

   NAND operator je funkcijsko poln sistem, ker je dobljen iz konjunkcije in negacije, ki tvorita funkcijsko poln sistem.

   Dokaz: Zapis negacije, konjunkcije in disjunkcije z NAND:

   $$\begin{aligned} \bar{x} &= \bar{x} \vee \bar{x} = \overline{x x} = x \uparrow x \\ x_1 x_2 &= \overline{\overline{x_1 x_2}} = \overline{x_1 \uparrow x_2} = (x_1 \uparrow x_2) \uparrow (x_1 \uparrow x_2) \\ x_1 \vee x_2 &= \overline{\overline{x_1 \vee x_2}} = \overline{\bar{x}_1 \bar{x}_2} = (\bar{x}_1 \uparrow \bar{x}_2) = (x_1 \uparrow x_1) \uparrow (x_2 \uparrow x_2) \end{aligned}$$

4) **(NOR)** - $x_1 \downarrow x_2 = \overline{x_1 \vee x_2} = \bar{x}_1\bar{x}_2$

NOR operator je funkcijsko poln sistem, ker je dobljen iz disjunkcije in negacije, ki tvorita funkcijsko poln sistem.

Dokaz: Zapis negacije, konjunkcije in disjunkcije z NOR:

$$\begin{aligned} \bar{x} &= \bar{x}\bar{x} = \overline{x \vee x} = x \downarrow x \\ x_1x_2 &= \overline{\overline{x_1x_2}} = \overline{\bar{x}_1 \vee \bar{x}_2} = (\bar{x}_1 \downarrow \bar{x}_2) = (x_1 \downarrow x_1) \downarrow (x_2 \downarrow x_2) \\ x_1 \vee x_2 &= \overline{\overline{x_1 \vee x_2}} = \overline{x_1 \downarrow x_2} = (x_1 \downarrow x_2) \downarrow (x_1 \downarrow x_2) \end{aligned}$$

5) **(XOR, AND, 1)** - $x_1 \nabla x_2 = \bar{x}_1x_2 \vee x_1\bar{x}_2$

Dokaz: Zapis negacije in disjunkcije z XOR, AND, 1:

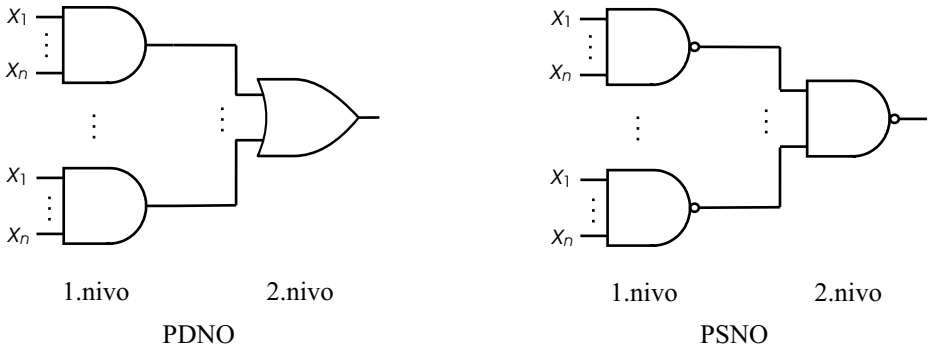
$$\begin{aligned} \bar{x} &= \bar{x}.1 \vee x.0 = x\nabla 1 \\ x_1 \vee x_2 &= \overline{\overline{x_1 \vee x_2}} = \overline{\bar{x}_1\bar{x}_2} = ((x_1\nabla 1)(x_2\nabla 1))\nabla 1 \end{aligned}$$

6.2 Predstavitev logičnih funkcij z NAND in NOR operatorji

Z elementarnim sistemom operatorjev (AND, OR, NEG) smo spoznali zapise normalnih oblik logičnih funkcij. Z uporabo NAND in NOR operatorjev, ki tvorita funkcijsko poln sistem, pridemo do novih normalnih oblik logičnih funkcij. Iz popolne disjunktivne normalne oblike dobimo z NAND operatorji popolno Shefferjevo normalno obliko (PSNO) (Slika 6.1), iz popolne konjunktivne normalne oblike pa z NOR operatorji popolno Pierceovo normalno obliko (PPNO) (Slika 6.2). Normalna oblika se ohranja pri pretvorbi disjunktivne oblike v Shefferjevo obliko in konjunktivne oblike v Pierceovo obliko. Če disjunktivno normalno obliko zapišemo s Pierceovimi operatorji ali konjunktivno s Shefferjevimi operatorji, dobimo nenormalne trinivojske oblike.

PSNO - Popolna Shefferjeva normalna oblika

Popolno Shefferjevo normalno obliko (PSNO) dobimo iz popolne disjunktivne normalne oblike (PDNO) tako, da v PDNO zamenjamo oba operatorja AND in OR z NAND.



Slika 6.1: Pretvorba PDNO v PSNO

V zapisu logične funkcije minterme (m_i) nadomestimo s Shefferjevimi mintermi (s_i).

Določimo jih tako, da v mintermu m_i, za $i = 0, 1, ..., 2^n - 1$ zamenjamo operator konjunkcije (AND) s Shefferjevim operatorjem (NAND). Zapis logične funkcije v PSNO podamo kot

$$f(x_1, x_2, ..., x_n) = \overset{2^n-1}{\underset{i=0}{\uparrow}} (s_i \uparrow f_i) \quad . \tag{6.1}$$

Primer: Pretvorba logične funkcije iz PDNO v PSNO

Logično funkcijo treh spremenljivk $f(x_1, x_2, x_3)$ podano v PDNO kot

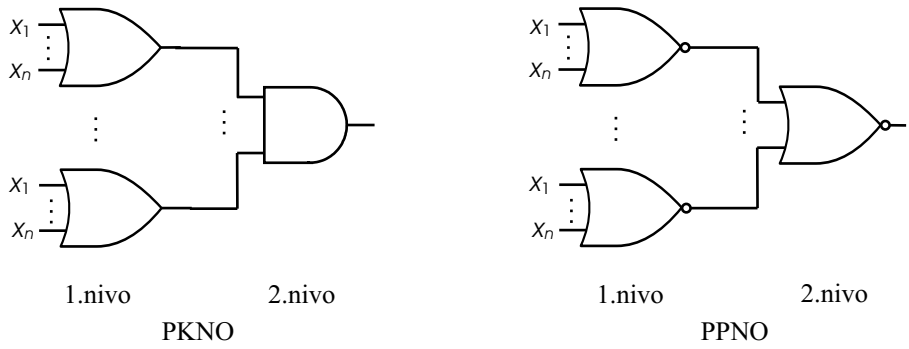
$$\begin{aligned} f(x_1, x_2, x_3) &= \vee(5, 7) \\ &= x_1\bar{x}_2x_3 \vee x_1x_2x_3 \end{aligned}$$

zapišemo v PSNO tako, da zamenjamo operatorje konjunkcije (AND) in disjunkcije (OR) z NAND operatorji in dobimo

$$\begin{aligned} f(x_1, x_2, x_3) &= \uparrow (5, 7) \\ &= (x_1 \uparrow \bar{x}_2 \uparrow x_3) \uparrow (x_1 \uparrow x_2 \uparrow x_3) \ . \end{aligned}$$

PPNO - Popolna Pierceova normalna oblika

Popolno Pierceovo normalno obliko (PPNO) dobimo iz popolne konjunktivne normalne oblike (PKNO) tako, da v PKNO zamenjamo oba operatorja OR in AND z NOR.



Slika 6.2: Pretvorba PKNO v PPNO

V zapisu logične funkcije maksterme (M_j) nadomestimo s Pierceovimi makstermi (S_j). Določimo jih tako, da v makstermu M_j, za $j = 2^n - 1, ..., 1, 0$ zamenjamo operator disjunkcije (OR) s Pierceovim operatorjem (NOR). Zapis logične funkcije v PPNO podamo kot

$$f(x_1, x_2, ...x_n) = \overset{2^n-1}{\underset{i=0}{\downarrow}} (S_{2^n-1-i} \downarrow f_i) \quad . \tag{6.2}$$

Primer: Pretvorba logične funkcije iz PKNO v PPNO

Logično funkcijo treh spremenljivk $f(x_1, x_2, x_3)$ podano v PKNO kot

$$\begin{aligned} f(x_1, x_2, x_3) &= \&(6, 3) \\ &= (x_1 \vee x_2 \vee \bar{x}_3)(\bar{x}_1 \vee x_2 \vee x_3) \end{aligned}$$

zapišemo v PPNO tako, da zamenjamo operatorje disjunkcije (OR) in konjunkcije (AND) z NOR operatorji in dobimo

$$\begin{aligned} f(x_1, x_2, x_3) &= \downarrow (6, 3) \\ &= (x_1 \downarrow x_2 \downarrow \bar{x}_3) \downarrow (\bar{x}_1 \downarrow x_2 \downarrow x_3) \ . \end{aligned}$$

Normalne oblike logičnih funkcij s Shefferjevimi in Pierceovimi operatorji

Logično funkcijo v disjunktivni ali konjunktivni normalni obliki prevedemo v zapis s Shefferjevimi (NAND) oziroma Pierceovimi (NOR) operatorji tako, da dvakrat negiramo celotno funkcijo, ali minterme, ali maksterme, ker ostane nespremenjena. Z DeMorganovim izrekom logično funkcijo prevedemo v negacije konjunkcij (NAND) ali pa v negacije disjunkcij (NOR) na prvem in drugem nivoju.

Zapis DNO z NAND in KNO z NOR Pri zapisu disjunktivne normalne oblike (DNO) logične funkcije z NAND oziroma konjunktivne normalne oblike (KNO) logične funkcije z NOR operatorji ohranimo dvonivojsko ali normalno obliko logične funkcije.

- Shefferjeva normalna oblika (SNO) - zapis DNO z NAND operatorji

  Funkcijo podano v DNO pretvorimo v zapis z NAND operatorji tako, da celotno funkcijo dvakrat negiramo. Eno negacijo razrešimo po DeMorganovem izreku in dobimo negirane konjunkcije ali spremenljivke na prvem nivoju. Negirane konjunkcije na prvem nivoju in negacijo konjunkcije na drugem nivoju nadomestimo z NAND operatorji in dobimo Shefferjevo normalno obliko (SNO).

  Primer: Logično funkcijo iz disjunktivne normalne oblike pretvorimo v Shefferjevo normalno obliko.

$$\begin{aligned} f(x_1,x_2,x_3) &= x_1x_2 \vee \bar{x}_2x_3 \vee \bar{x}_3 \\ f(x_1,x_2,x_3) &= \overline{\overline{x_1x_2 \vee \bar{x}_2x_3 \vee \bar{x}_3}} \\ &= \overline{\overline{(x_1x_2)}.\overline{(\bar{x}_2x_3)}.\overline{(\bar{x}_3)}} \\ &= (x_1 \uparrow x_2) \uparrow (\bar{x}_2 \uparrow x_3) \uparrow x_3 \end{aligned}$$

- Pierceova normalna oblika (PNO) - zapis KNO z NOR operatorji

  Funkcijo podano v KNO pretvorimo v zapis z NOR operatorji tako, da celotno funkcijo dvakrat negiramo. Eno negacijo razrešimo po DeMorganovem izreku in dobimo negirane disjunkcije ali spremenljivke na prvem nivoju. Negirane disjunkcije na prvem nivoju in negacijo disjunkcije na drugem nivoju nadomestimo z NOR operatorji in dobimo Pierceovo normalno obliko (PNO).

  Primer: Logično funkcijo iz konjunktivne normalne oblike pretvorimo v Pierceovo normalno obliko.

$$\begin{aligned} f(x_1,x_2,x_3) &= (x_1 \vee x_2).(\bar{x}_2 \vee \bar{x}_3).x_3 \\ f(x_1,x_2,x_3) &= \overline{\overline{(x_1 \vee x_2).(\bar{x}_2 \vee \bar{x}_3).x_3}} \\ &= \overline{\overline{(x_1 \vee x_2)} \vee \overline{(\bar{x}_2 \vee \bar{x}_3)} \vee \bar{x}_3} \\ &= (x_1 \downarrow x_2) \downarrow (\bar{x}_2 \downarrow \bar{x}_3) \downarrow \bar{x}_3 \end{aligned}$$

Dvonivojske oblike logičnih funkcij z operatorji AND, OR, NAND, NOR

Iz nabora operatorjev AND, OR, NAND, NOR je možno določiti 16 različnih oblik logičnih funkcij, od katerih je 8 normalnih oblik (Tabela 6.1), ostale pa imajo dodano negacijo na izhodu prvega nivoja, ali pa na izhodu drugega nivoja. Osnovni obliki logičnih funkcij sta DNO in KNO in sta uporabljeni za izpeljavo ostalih normalnih oblik.

Tabela 6.1: Normalne oblike logičnih funkcij

1)	AND/OR (DNO)	OR/AND (KNO)
2)	NAND/NAND (SNO)	NOR/NOR (PNO)
3)	NOR/OR	NAND/AND
4)	OR/NAND	AND/NOR

DNO in normalne oblike zapisane iz DNO:

1) AND/OR ali disjunktivna normalna oblika (DNO)

$$f(x_1, x_2, x_3) = x_1 \vee x_2\bar{x}_3$$

2) NAND/NAND ali Shefferjeva normalna oblika (SNO) - dvakrat negiramo logično funkcijo

$$\begin{aligned} f(x_1, x_2, x_3) &= \overline{\overline{x_1 \vee x_2\bar{x}_3}} \\ &= \overline{\bar{x}_1.\overline{(x_2\bar{x}_3)}} \\ &= \bar{x}_1 \uparrow (x_2 \uparrow \bar{x}_3) \end{aligned}$$

3) NOR/OR - dvakrat negiramo konjunkcije prvega nivoja

$$\begin{aligned} f(x_1, x_2, x_3) &= x_1 \vee \overline{\overline{x_2\bar{x}_3}} \\ &= x_1 \vee \overline{(\bar{x}_2 \vee x_3)} \\ &= x_1 \vee (\bar{x}_2 \downarrow x_3) \end{aligned}$$

4) OR/NAND - dvakrat negiramo logično funkcijo

$$\begin{aligned} f(x_1, x_2, x_3) &= \overline{\overline{x_1 \vee x_2\bar{x}_3}} \\ &= \overline{\bar{x}_1.\overline{(x_2\bar{x}_3)}} \\ &= \bar{x}_1 \uparrow (\bar{x}_2 \vee x_3) \end{aligned}$$

KNO in normalne oblike zapisane iz KNO:

1) OR/AND ali konjunktivna normalna oblika (KNO)

$$f(x_1, x_2, x_3) = x_2(\bar{x}_1 \vee \bar{x}_2 \vee x_3)$$

2) NOR/NOR ali Pierceova normalna oblika (PNO) - dvakrat negiramo logično funkcijo

$$\begin{aligned} f(x_1, x_2, x_3) &= \overline{\overline{x_2.(\bar{x}_1 \vee \bar{x}_2 \vee x_3)}} \\ &= \overline{\bar{x}_2 \vee \overline{(\bar{x}_1 \vee \bar{x}_2 \vee x_3)}} \\ &= \bar{x}_2 \downarrow (\bar{x}_1 \downarrow \bar{x}_2 \downarrow x_3) \end{aligned}$$

3) NAND/AND - dvakrat negiramo disjunkcije prvega nivoja

$$\begin{aligned} f(x_1, x_2, x_3) &= x_2.\overline{\overline{(\bar{x}_1 \vee \bar{x}_2 \vee x_3)}} \\ &= x_2.\overline{(x_1x_2\bar{x}_3)} \\ &= x_2.(x_1 \uparrow x_2 \uparrow \bar{x}_3) \end{aligned}$$

4) AND/NOR - dvakrat negiramo logično funkcijo

$$\begin{aligned} f(x_1,x_2,x_3) &= \overline{\overline{x_2.(\bar{x}_1 \vee \bar{x}_2 \vee x_3)}} \\ &= \overline{\bar{x}_2 \vee \overline{(\bar{x}_1 \vee \bar{x}_2 \vee x_3)}} \\ &= \bar{x}_2 \downarrow (x_1 x_2 \bar{x}_3) \end{aligned}$$

Nenormalne oblike logičnih funkcij s Shefferjevimi in Pierceovimi operatorji

Pri zapisu disjunktivne normalne oblike (DNO) logične funkcije z NOR oziroma konjunktivne normalne oblike (KNO) logične funkcije z NAND imamo na izhodu dodano negacijo, zato logična funkcija postane trinivojska.

- Zapis DNO logičnih funkcij z NOR operatorji

  Funkcijo podano v DNO pretvorimo v zapis z NOR operatorji tako, da celotno funkcijo in vse konjunktivne izraze prvega nivoja dvakrat negiramo. Eno negacijo pri konjunktivnih izrazih razrešimo po DeMorganovem izreku in dobimo negirane disjunkcije. Negirane disjunkcije na prvem nivoju in eno negacijo disjunkcije na drugem nivoju nadomestimo z NOR operatorji, ena negacija na izhodu pa ostane in v tem primeru dobimo trinivojski zapis logične funkcije.

  Primer: Logično funkcijo v disjunktivni normalni obliki zapišemo z NOR operatorji.

$$\begin{aligned} f(x_1,x_2,x_3) &= x_1 x_2 \vee \bar{x}_2 x_3 \vee \bar{x}_3 \\ f(x_1,x_2,x_3) &= \overline{\overline{\overline{x_1 x_2} \vee \overline{\bar{x}_2 x_3}} \vee \bar{x}_3} \\ &= \overline{\overline{\bar{x}_1 \vee \bar{x}_2} \vee \overline{x_2 \vee \bar{x}_3} \vee \bar{x}_3} \\ &= \overline{(\bar{x}_1 \downarrow \bar{x}_2) \downarrow (x_2 \downarrow \bar{x}_3) \downarrow \bar{x}_3} \end{aligned}$$

- Zapis KNO logičnih funkcij z NAND operatorji

  Funkcijo podano v KNO pretvorimo v zapis z NAND operatorji tako, da celotno funkcijo in vse disjunktivne izraze prvega nivoja dvakrat negiramo. Eno negacijo pri disjunktivnih izrazih razrešimo po DeMorganovem izreku in dobimo negirane konjunkcije, ki jih nadomestimo z NAND operatorji. Negirane konjunkcije na prvem nivoju in eno negacijo konjunkcije na drugem nivoju nadomestimo z NAND operatorji, ena negacija na izhodu pa ostane in v tem primeru dobimo trinivojski zapis logične funkcije.

  Primer: Logično funkcijo v konjunktivni normalni obliki zapišemo z NAND operatorji.

$$\begin{aligned} f(x_1,x_2,x_3) &= (x_1 \vee x_2).(\bar{x}_2 \vee \bar{x}_3).x_3 \\ f(x_1,x_2,x_3) &= \overline{\overline{\overline{(x_1 \vee x_2)}.\overline{(\bar{x}_2 \vee \bar{x}_3)}}.x_3} \\ &= \overline{\overline{(\bar{x}_1 \bar{x}_2)}.\overline{(x_2 x_3)}.x_3} \\ &= \overline{(\bar{x}_1 \uparrow \bar{x}_2) \uparrow (x_2 \uparrow x_3) \uparrow x_3} \end{aligned}$$

6.3 Zapis logičnih funkcij z XOR, AND, 1

Logična operacija seštevanja po modulu 2 (XOR) je osnovna funkcija dvojiškega računanja. V naboru logičnih funkcij z n spremenljivkami imamo manjšo skupino funkcij, ki jih je možno zapisati samo z XOR operatorji in jih imenujemo **linearne logične funkcije**. Preostale logične funkcije pa z uporabo operatorjev XOR, AND in konstante 1, ki tvorijo funkcijsko poln sistem, zapišemo v **Reed-Mullerjevi obliki**.

6.3.1 Linearne logične funkcije

Logična funkcija je linearna, če jo je mogoče zapisati kot linearni polinom

$$f(x_1, x_2, ..., x_n) = a_0 \nabla a_1 x_1 \nabla a_2 x_2 \nabla ... \nabla a_n x_n \ , \tag{6.3}$$

kjer so $a_0, a_1, ..., a_n$ koeficienti, ki zavzamejo vrednosti Boolovih konstant $0, 1$.

Ogledali si bomo analitičen in grafičen način ugotavljanja linearnosti. Pri analitičnem načinu sproti računamo koeficiente a_i, $i = 0, 1, ..., n$ in jih uporabimo za ugotavljanje linearnosti. Pri grafičnem načinu pa najprej v Veitchevem diagramu ugotovimo, ali je funkcija linearna, nato izračunamo koeficiente in jih vstavimo v linearni polinom.

Analitično ugotavljanje linearnosti in zapis linearnega polinoma

Za podano logično funkcijo predpostavimo, da je linearna, zato izpišemo v tabeli 2^n linearnih enačb za vsako vhodno kombinacijo z vstavljanjem vrednosti vhodnih spremenljivk v linearni polinom. Izberemo $n+1$ enačb in izračunamo koeficiente $a_0, a_1, a_2, ..., a_n$. Če je za izračunane koeficiente izpolnjenih še preostalih $2^n - (n+1)$ enačb, je logična funkcija linearna. V linearni polinom vstavimo izračunane koeficiente in dobimo enostaven zapis logične funkcije z operatorji seštevanja po modulu 2 (XOR).

Primer: Z analitičnim postopkom ugotovimo ali je logična funkcija v pravilnostni tabeli linearna.

Tabela 6.2: Ugotavljanje linearnosti

$x_1x_2x_3$	$f(x_1, x_2, x_3)$	$a_0 \nabla a_1 x_1 \nabla a_2 x_2 \nabla a_3 x_3 = f_i$	Poenostavljene enačbe
0 0 0	0	$a_0 \nabla 0.a_1 \nabla 0.a_2 \nabla 0.a_3 = 0$	$a_0 = 0$
0 0 1	1	$a_0 \nabla 0.a_1 \nabla 0.a_2 \nabla 1.a_3 = 1$	$a_0 \nabla a_3 = 1$
0 1 0	1	$a_0 \nabla 0.a_1 \nabla 1.a_2 \nabla 0.a_3 = 1$	$a_0 \nabla a_2 = 1$
0 1 1	1	$a_0 \nabla 0.a_1 \nabla 1.a_2 \nabla 1.a_3 = 1$	$a_0 \nabla a_2 \nabla a_3 = 1$
1 0 0	0	$a_0 \nabla 1.a_1 \nabla 0.a_2 \nabla 0.a_3 = 0$	$a_0 \nabla a_1 = 0$
1 0 1	1	$a_0 \nabla 1.a_1 \nabla 0.a_2 \nabla 1.a_3 = 1$	$a_0 \nabla a_1 \nabla a_3 = 1$
1 1 0	1	$a_0 \nabla 1.a_1 \nabla 1.a_2 \nabla 0.a_3 = 1$	$a_0 \nabla a_1 \nabla a_2 = 1$
1 1 1	0	$a_0 \nabla 1.a_1 \nabla 1.a_2 \nabla 1.a_3 = 0$	$a_0 \nabla a_1 \nabla a_2 \nabla a_3 = 0$

Izračunamo koeficiente a_0, a_1, a_2, a_3 iz naslednjih enačb:

$$\begin{array}{rrl} & a_0 = 0 & \Rightarrow a_0 = 0 \\ a_0 \nabla a_3 = 1 & 0 \nabla a_3 = 1 & \Rightarrow a_3 = 1 \\ a_0 \nabla a_2 = 1 & 0 \nabla a_2 = 1 & \Rightarrow a_2 = 1 \\ a_0 \nabla a_1 = 0 & 0 \nabla a_1 = 0 & \Rightarrow a_1 = 0 \end{array}$$

Za izračunane koeficiente preverimo še ostale enačbe:

$$a_0 \nabla a_2 \nabla a_3 = 1 \qquad 0 \nabla 1 \nabla 1 = 0 \neq 1$$

Že pri enačbi $a_0 \nabla a_2 \nabla a_3 = 1$ ugotovimo neenakost izračunanih koeficientov na levi strani enačbe s funkcijsko vrednostjo, zato logična funkcija ni linearna.

Grafično ugotavljanje linarnosti in zapis linearnega polinoma

Pri grafičnem načinu opazujemo pogoj linearnosti in zapis linearnih enačb za izračun koeficientov v Veitchevem diagramu. Za ugotavljanje linearnosti logične funkcije z n vhodnimi spremenljivkami uporabimo prepogibanje kvadratov oziroma pravokotnikov, v katerih opazujemo enakost ali negacijo funkcijskih vrednosti.

Linearnost ugotavljamo grafično z naslednjim postopkom:

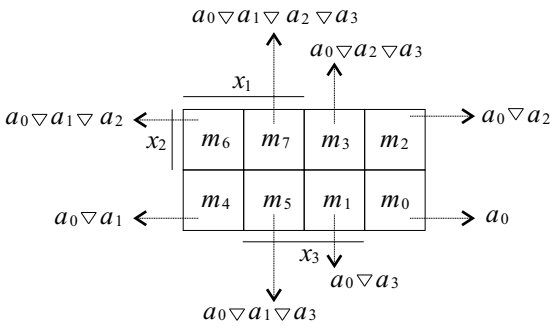
- V prvem koraku $k = 1$ kvadrat z mintermom m_0 prepognemo s kvadratom pri spremenljivki x_n, to je mintermom m_1, in pogledamo ali sta funkcijski vrednosti enaki ali različni. Pogoj enakosti ali negacije je vedno izpolnjen (če ima minterm m_0 funkcijsko vrednost 1, ima lahko minterm m_1 funkcijsko vrednost 0 ali 1 in obratno).

- V vsakem naslednjem koraku $k = 2, ..., n$ opazovana pravokotnika iz prejšnjega koraka združimo v nov pravokotnik in ga prepognemo proti pravokotniku pri spremenljivki $x_{(n+1)-k}$. Če imajo vsi kvadrati v pokritem pravokotniku enake ali negirane vrednosti kvadratov s katerimi smo jih prekrili, je pogoj linearnosti izpolnjen. V koraku $k = 2$, na primer pravokotnik z mintermoma m_0 in m_1 prepognemo proti pravokotniku z mintermoma m_2 in m_3 (če sta minterma $m_0 = 1$ in $m_1 = 0$, imamo enakost kadar sta $m_2 = 1$ in $m_3 = 0$ ali negacijo kadar sta $m_2 = 0$ in $m_3 = 1$).

- Logična funkcija je linearna, če je izpolnjen pogoj linearnosti v vseh n korakih.

Zapis linearnega polinoma

Če smo v Veitchevem diagramu ugotovili, da je logična funkcija linearna, jo lahko zapišemo v obliki linearnega polinoma. Vzamemo $n + 1$ linearnih enačb, izračunamo koeficiente $a_0, a_1, ..., a_n$ in jih vstavimo v linearni polinom. Linearne enačbe lahko izpišemo kar iz Veitchevega diagrama. Za logično funkcijo s tremi vhodnimi spremenljivkami imamo pri vsakem mintermu zapisano linearno enačbo (Slika 6.3).



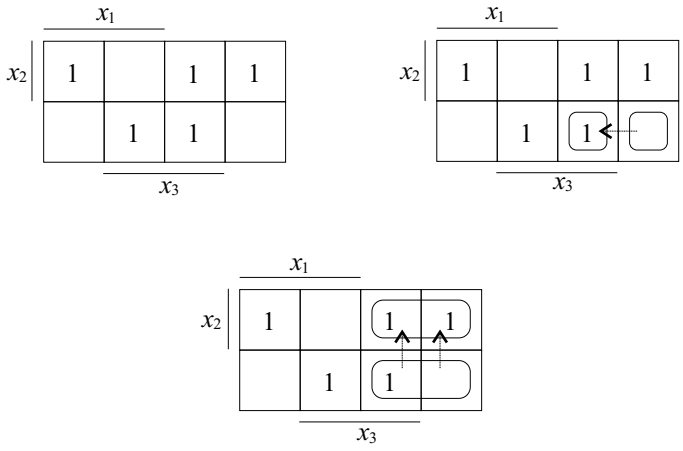
Slika 6.3: Linearne enačbe v Veitchevem diagramu

Primer: Z grafičnim postopkom v Veitchevem diagramu ugotovimo ali je logična funkcija

linearna. Če ugotovimo, da je funkcija linearna, izračunajmo koeficiente in jo zapišimo v obliki linearnega polinoma.

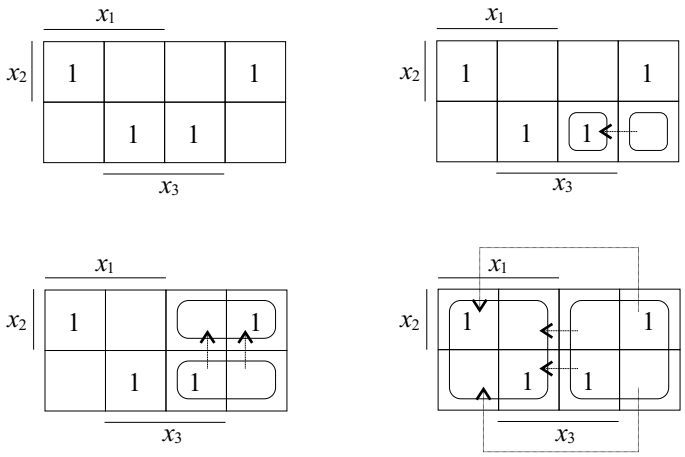
$$f(x_1, x_2, x_3) = \vee(1, 2, 3, 5, 6)$$

Prepognemo kvadrat z mintermom m_0 proti spremenljivki x_3. Prepognemo oba kvadrata skupaj proti spremenljivki x_2. Rezultat prepogiba ni niti enakost, niti negacija, zato logična funkcija ni linearna.

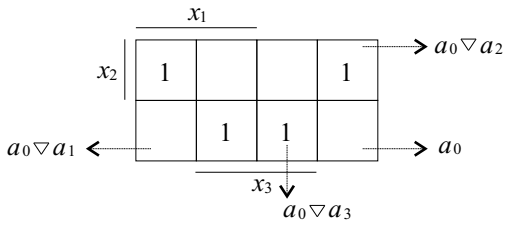


Primer: Z grafičnim postopkom v Veitchevem diagramu ugotovimo ali je logična funkcija linearna. Če ugotovimo, da je funkcija linearna, izračunajmo koeficiente in jo zapišimo v obliki linearnega polinoma.
$f(x_1, x_2, x_3) = \vee(1, 2, 5, 6)$

Prepognemo kvadrat z mintermom a_0 proti spremenljivki x_3 in nato oba skupaj proti spremenljivki x_2. Rezultat prepogiba je negacija obeh funkcijskih vrednosti, kar izpolnjuje pogoj linearnosti, zato prepognemo vse štiri kvadrate proti spremenljivki x_1. Rezultat prepogiba je enakost v vseh štirih kvadratih. Pregledali smo cel Veitchev diagram in ker smo povsod dobili izpolnjen pogoj enakosti ali negacije pomeni, da je logična funkcija linearna.



Ker je logična funkcija linearna, izračunamo koeficiente a_0, a_1, a_2, a_3 iz zapisanih linearnih enačb (Slika 6.4).



Slika 6.4: Enačbe za izračun koeficientov a_0, a_1, a_2, a_3

$$\begin{array}{lll} & & \Rightarrow a_0 = 0 \\ a_0 \nabla a_3 = 1 & 0 \nabla a_3 = 1 & \Rightarrow a_3 = 1 \\ a_0 \nabla a_2 = 1 & 0 \nabla a_2 = 1 & \Rightarrow a_2 = 1 \\ a_0 \nabla a_1 = 0 & 0 \nabla a_1 = 0 & \Rightarrow a_1 = 0 \end{array}$$

Izračunane koeficiente vstavimo v linearni polinom $f(x_1, x_2, x_3) = a_0 \nabla a_1 x_1 \nabla a_2 x_2 \nabla a_3 x_3$ in dobimo zapis z XOR operatorji.

$$f(x_1, x_2, x_3) = a_0 \nabla a_1 x_1 \nabla a_2 x_2 \nabla a_3 x_3 = 0 \nabla 1.x_1 \nabla 1.x_2 \nabla 0.x_3 = x_2 \nabla x_3$$

6.3.2 Reed-Mullerjeva oblika logičnih funkcij

Operator XOR skupaj z operatorjem AND in konstanto 1 predstavlja funkcijsko poln sistem. Uporabimo ga za realizacijo logičnih funkcij, ki niso linearne. Zapis logičnih funkcij v opisanem funkcijsko polnem sistemu predstavlja Reed-Mullerjeva oblika

$$f(x_1, x_2, ..., x_n) = a_0 \nabla a_1 x_1 \nabla ... \nabla a_n x_n \nabla a_{n+1} x_1 x_2 \nabla ... \nabla a_{n+k} x_1 x_2 x_3 ... \nabla a_{2^n-1} x_1 ... x_n \ ,$$

kjer so $a_i, i = 0, 1, 2, ...2^n - 1$ binarne vrednosti koeficientov (konstanti $0, 1$). V zapisu logične funkcije v Reed-Mullerjevi obliki imamo 2^n členov.

Prevedba logične funkcije v Reed-Mullerjevo obliko:

1) Funkcijo zapišemo v ortogonalni disjunktivni normalni obliki. Pogoj ortogonalnosti pomeni, da za vsak par konjunkcij q_i, q_j,..., q_k v disjunktivni normalni obliki velja $q_i.q_j = 0$, $q_i.q_k = 0$, $q_j.q_k = 0$, itd. Vsaka logična funkcija v popolni disjunktivni normalni obliki (PDNO) je vedno ortogonalna. Ortogonalno minimalno disjunktivno normalno obliko (OMDNO) dobimo v postopku minimizacije tako, da je vsak minterm, ki nastopa v opisu funkcije, upoštevan samo v enem glavnem vsebovalniku ali vsebovalniku.

   Primer: Logična funkcija $f(x_1, x_2, x_3) = x_1 x_2 \vee \bar{x}_1 x_3$ je ortogonalna, ker je konjunkcija obeh konjunktivnih izrazov enaka nič ($x_1 x_2 . \bar{x}_1 x_3 = 0$).

2) Operator disjunkcije ($\vee$) v ortogonalni obliki zamenjamo z XOR operatorjem (∇).

3) Nadomestimo negacijo $\bar{x} = x \nabla 1$.

4) Odpravimo oklepaje z distributivnim zakonom: $x_i(x_j \nabla x_k) = x_i x_j \nabla x_i x_k$.

5) Poenostavimo logično funkcijo: $x_i \nabla x_i = 0$ (dva enaka člena odpadeta).

Primer: Oglejmo si pretvorbo logične funkcije $f(x_1, x_2, x_3) = x_1(x_2 x_3 \vee \bar{x}_3)$ v Reed-Mullerjevo obliko.

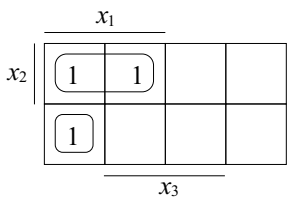
a) V prvem primeru funkcijo najprej prevedemo v popolno disjunktivno normalno obliko (PDNO).

$$\begin{aligned} f(x_1, x_2, x_3) &= x_1(x_2x_3 \vee \bar{x}_3) \\ &= x_1x_2x_3 \vee x_1\bar{x}_3 \\ &= x_1x_2x_3 \vee x_1(x_2 \vee \bar{x}_2)\bar{x}_3 \\ &= x_1x_2x_3 \vee x_1x_2\bar{x}_3 \vee x_1\bar{x}_2\bar{x}_3 \end{aligned}$$

Logično funkcijo iz PDNO pretvorimo v Reed-Mullerjevo obliko.

$$\begin{aligned} f(x_1, x_2, x_3) &= x_1x_2x_3 \vee x_1x_2\bar{x}_3 \vee x_1\bar{x}_2\bar{x}_3 \\ &= x_1x_2x_3 \nabla x_1x_2\bar{x}_3 \nabla x_1\bar{x}_2\bar{x}_3 \\ &= x_1x_2x_3 \nabla x_1x_2(x_3 \nabla 1) \nabla x_1(x_2 \nabla 1)(x_3 \nabla 1) \\ &= x_1x_2x_3 \nabla x_1x_2x_3 \nabla x_1x_2 \nabla x_1x_2x_3 \nabla x_1x_2 \nabla x_1x_3 \nabla x_1 \\ &= x_1x_2x_3 \nabla x_1x_3 \nabla x_1 \end{aligned}$$

b) Logično funkcijo zapišemo v ortogonalni minimalni disjunktivni normalni obliki (OMDNO). V Veitchevem diagramu imamo en glavni vsebovalnik x_1x_2 in en vsebovalnik $x_1\bar{x}_2\bar{x}_3$ tako, da je vsak minterm v minimizaciji upoštevan samo enkrat.



$$f(x_1, x_2, x_3) = x_1x_2 \vee x_1\bar{x}_2\bar{x}_3$$

Prevedba funkcije iz OMDNO v Reed-Mullerjevo obliko:

$$\begin{aligned} f(x_1, x_2, x_3) &= x_1x_2 \vee x_1\bar{x}_2\bar{x}_3 \\ &= x_1x_2 \nabla x_1\bar{x}_2\bar{x}_3 \\ &= x_1x_2 \nabla x_1(x_2 \nabla 1)(x_3 \nabla 1) \\ &= x_1x_2 \nabla x_1x_2x_3 \nabla x_1x_2 \nabla x_1x_3 \nabla x_1 \\ &= x_1x_2x_3 \nabla x_1x_3 \nabla x_1 \end{aligned}$$

Pri obeh pretvorbah logične funkcije smo dobili enak zapis v Reed-Mullerjevi obliki. Odločitev o izbiri začetne oblike logične funkcije za pretvorbo je odvisna od kompleksnosti logične funkcije v enem od možnih zapisov.

6.4 Večnivojske logične funkcije

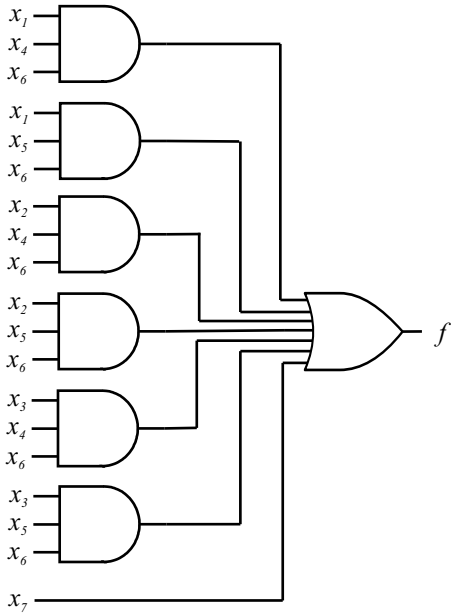
Najbolj običajen zapis logičnih funkcij je ena od normalnih oblik, kot sta MDNO oziroma MKNO. V izvedbi logičnih funkcij taki zapisi še vedno lahko zahtevajo veliko število vrat. Pogosto obstaja kakšen drug način zapisa logične funkcije, ki utegne zmanjšati število logičnih vrat. Taki zapisi logičnih funkcij, ki zmanjšajo število logičnih vrat, ali celo število vhodov, običajno povečajo število nivojev logičnega vezja. Vezja, ki jih dobimo na ta način, imenujemo večnivojska logična vezja. V tem primeru lahko pridobimo na

prostoru, a istočasno izgubimo na hitrosti delovanja vezja.

Vzemimo logično funkcijo $f = f(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$ v minimalni normalni obliki.

$$f = x_1x_4x_6 \vee x_1x_5x_6 \vee x_2x_4x_6 \vee x_2x_5x_6 \vee x_3x_4x_6 \vee x_3x_5x_6 \vee x_7$$

Za izvedbo logične funkcije potrebujemo šest 3-vhodnih konjunkcij in eno 7-vhodno disjunkcijo (Slika 6.5).



Slika 6.5: Dvonivojska izvedba logične funkcije

Podano normalno obliko logične funkcije lahko z uporabo Boolovih zakonov in izrekov preoblikujemo. V vseh konjunktivnih izrazih se nahaja spremenljivka x_6, ki jo lahko izpostavimo. Dobljen izraz ima v oklepaju konjunkcije, ki jim izpostavimo x_4 in x_5 in dobimo nov izraz, ki ga znova skrajšamo in dobimo nov zapis logične funkcije.

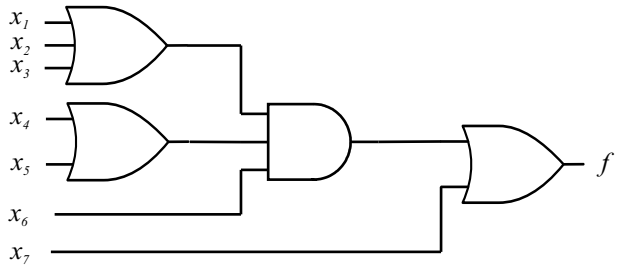
$$\begin{aligned} f &= (x_1x_4 \vee x_1x_5 \vee x_2x_4 \vee x_2x_5 \vee x_3x_4 \vee x_3x_5)x_6 \vee x_7 \\ f &= [(x_1 \vee x_2 \vee x_3)x_4 \vee (x_1 \vee x_2 \vee x_3)x_5]x_6 \vee x_7 \\ f &= (x_1 \vee x_2 \vee x_3)(x_4 \vee x_5)x_6 \vee x_7 \end{aligned}$$

Funkcijo f lahko zapišemo kot zaporedje izrazov v normalni obliki:

$$\begin{aligned} f &= ABx_6 \vee x_7 \\ A &= x_1 \vee x_2 \vee x_3 \\ B &= x_4 \vee x_5 \end{aligned}$$

Izvedba logične funkcije v novi obliki zahteva ena 3-vhodna OR vrata, dvoje 2-vhodnih OR vrat in ena 3-vhodna AND vrata, kar je v celoti štiri vrata z devetimi vhodi (Slika 6.6).

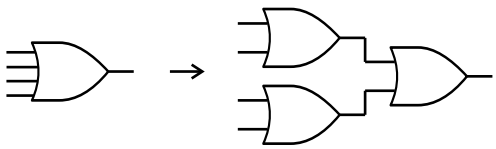
Minimalno normalno obliko logične funkcije lahko dobimo na zelo enostaven način z eno od minimizacijskih metod. Prehod signalov preko takih vezij je hiter, ker gre največ preko dveh nivojev vrat, če ne upoštevamo negacij spremenljivk na vhodu. Pri taki obliki se lahko pojavijo primeri, ko imajo posamezna vrata veliko vhodov, kar lahko zahteva več prostora. V številnih tehnologijah, vključno s TTL, so vrata z velikim številom vhodov



Slika 6.6: Trinivojska izvedba logične funkcije

nadomeščena z vrati z manjšim številom vhodov. Iz tega razloga je pomembna sinteza večnivojskih logičnih vezij.

Optimalno dvonivojsko vezje je določeno z minimalnim številom konjunkcij, disjunkcij in spremenljivk za realizacijo logične funkcije v pravilnostni tabeli. Optimalno večnivojsko vezje pa ni tako enostavno definirati, ker se postavlja vprašanje ali je bolj kritično imeti manjše število vrat, ali manjše število spremenljivk v končnem zapisu. Procesu določanja večnivojskih funkcij pravimo **sinteza** in ne optimizacija. Cilj je narediti razumno večnivojsko realizacijo, brez posebnih trditev, da je rezultat najboljši možen.
Sinteza vključuje dve fazi. V prvi fazi, ki je neodvisna od tehnologije, se zmanjša število vhodov logičnih vrat (ang. fan-in) z naraščanjem števila nivojev. Tu se ne oziramo na to, kakšna vrata bomo uporabili, ampak spreminjamo logični izraz na osnovi matematičnih pravil Boolove algebre. Druga faza preslika logične izraze v potrebno izvedbo na osnovi knjižnice vrat. V primeru, da imamo samo 2-vhodna OR vrata, v izrazu pa se pojavijo 4-vhodna vrata, jih izvedemo s tremi 2-vhodnimi vrati v dveh nivojih, kar poveča zakasnitveni čas vezja (Slika 6.7).



Slika 6.7: Izvedba 4-vhodnih vrat z 2-vhodnimi

Faktorski zapis logične funkcije

Večnivojska sinteza prevede izraze v posebno drevesno strukturo, ki jo imenujemo faktorska oblika. V njej so uporabljene operacije AND in OR v nekakšni **'disjunkciji konjunkcij od disjunkcij konjunkcij...'**, kot vidimo na spodnjem primeru.

$$F = (x_1x_2 \vee \bar{x}_2x_3)(x_3 \vee x_4(x_5 \vee x_1\bar{x}_3)) \vee (x_4 \vee x_5)(x_6x_7)$$

Faktorsko obliko lahko prepišemo kot sekvenco dvonivojskih izrazov.

$$F = F_1F_2 \vee F_3F_4$$

$$\begin{aligned} F_1 &= x_1x_2 \vee \bar{x}_2x_3 \\ F_2 &= x_3 \vee x_4F_5 \\ F_3 &= x_4 \vee x_5 \\ F_4 &= x_6x_7 \\ F_5 &= x_5 \vee x_1\bar{x}_3 \end{aligned}$$

Za razliko od minimizacije normalnih oblik pri sintezi logičnih funkcij v večnivojske oblike nimamo podane metode po kateri dobimo optimalno obliko. Osnovne operacije za preoblikovanje večnivojskih izrazov ali funkcij so:

- dekompozicija,
- ekstrakcija
- faktorizacija,
- substitucija,
- razpad.

Dekompozicija

Dekompozicija nadomesti eno logično funkcijo z množico novih izrazov. Za logično funkcijo F imamo v normalni obliki 12 spremenljivk in 9 vrat (AND, OR, NEG). Z dekompozicijo dobimo enostavnejšo trinivojsko logično funkcijo, ki ima 8 spremenljivk in 7 vrat.

$$\begin{aligned} F &= x_1x_2x_3 \vee x_1x_2x_4 \vee \bar{x}_1\bar{x}_3\bar{x}_4 \vee \bar{x}_2\bar{x}_3\bar{x}_4 \\ &= x_1x_2.(x_3 \vee x_4) \vee \bar{x}_3\bar{x}_4.(\bar{x}_1 \vee \bar{x}_2) \\ &= x_1x_2.(x_3 \vee x_4) \vee \overline{(x_1x_2)}.\overline{(x_3 \vee x_4)} \end{aligned}$$

Iz preoblikovanega zapisa logične funkcije lahko zapišemo dve enostavni logični funkciji

$$\begin{aligned} F_1 &= x_1x_2 \\ F_2 &= x_3 \vee x_4 \end{aligned}$$

in zapišemo logično funkcijo F kot

$$F = F_1F_2 \vee \bar{F}_1\bar{F}_2 \ .$$

Ekstrakcija

Ekstrakcijo uporabimo v primeru večjega števila logičnih funkcij. Iz vseh podanih logičnih funkcij določimo skupne izraze ali podfunkcije, katerih iskanje ni prav enostavno, ker morajo biti to izrazi, ki so skupni vsem funkcijam.

$$\begin{aligned} F &= (x_1 \vee x_2)x_3x_4 \vee x_5 \\ G &= (x_1 \vee x_2)\bar{x}_5 \\ H &= x_3x_4x_5 \end{aligned}$$

V logičnih funkcijah imamo skupna izraza $x_1 \vee x_2$ za F in G ter x_3x_4 za F in H. Zapišemo ju z novima funkcijama

$$\begin{aligned} X &= x_1 \vee x_2 \\ Y &= x_3x_4 \ . \end{aligned}$$

Vse tri funkcije po določitvi skupnih izrazov (ekstrakciji) zapišemo kot

$$\begin{aligned} F &= XY \vee x_5 \\ G &= X\bar{x}_5 \\ H &= Yx_5 \ . \end{aligned}$$

Logične funkcije v prvotnem zapisu imajo 11 spremenljivk in v izvedbi zahtevajo 8 vrat. Po izvedbi ekstrakcije je spremenljivk še vedno 11, število logičnih vrat pa samo 7.

Faktorizacija

Faktorizacija je preoblikovanje dvonivojske logične funkcije v večnivojsko tako, da določi potencialno skupne izraze ali podfunkcije. Uporabljena je kot predhodni korak ekstrakcije.

$$\begin{aligned} F &= x_1x_3 \vee x_1x_4 \vee x_2x_3 \vee x_2x_4 \vee x_5 \\ F &= x_1.(x_3 \vee x_4) \vee x_2.(x_3 \vee x_4) \vee x_5 \\ F &= (x_1 \vee x_2)(x_3 \vee x_4) \vee x_5 = F_1F_2 \vee x_5 \end{aligned}$$

$$\begin{aligned} F_1 &= x_1 \vee x_2 \\ F_2 &= x_3 \vee x_4 \end{aligned}$$

Logična funkcija v dvonivojski obliki ima 9 spremenljivk in 5 logičnih vrat. Po faktorizaciji je število spremenljivk zmanjšano na 7 in število vrat na 4.

Substitucija

Substitucija ali nadomestitev je uporaba nove funkcije G v funkciji F. Za podano logično funkcijo F določimo $G = x_1 \vee x_2$ in jo lahko izrazimo v odvisnosti od G kot

$$\begin{aligned} F &= x_1 \vee x_2x_3 = (x_1 \vee x_2).(x_1 \vee x_3) \\ &= G(x_1 \vee x_3) \ . \end{aligned}$$

Ko so takšni izrazi v funkciji določeni, je lahko substitucija uporabljena za ponovno preoblikovanje funkcije v faktorsko obliko.

Razpad

Razpad je obratno substituciji in je uporabljen za zmanjšanje števila nivojev logične funkcije, če se pojavijo časovne omejitve. Kot primer poglejmo razpad funkcije z upoštevanjem izraza $G = x_1 \vee x_2$ iz substitucije funkcije F.

$$\begin{aligned} F &= G(x_1 \vee x_3) \\ &= (x_1 \vee x_2)(x_1 \vee x_3) \\ &= x_1x_1 \vee x_1x_3 \vee x_1x_2 \vee x_2x_3 \\ &= x_1 \vee x_2x_3 \end{aligned}$$

Vse te operacije so lahko za obsežne logične funkcije zelo zapletene za ročno izpeljevanje, zato imamo na voljo programska orodja. V zgornjih primerih smo skušali podati nekaj značilnih pristopov za načrtovanje enostavnih logičnih vezij.

6.5 Vaje

VAJA 6.5.1: Pretvorba DNO v SNO

Zapišite DNO logičnih funkcij z NAND operatorji.

$$\begin{aligned} a) f(x_1,x_2,x_3,x_4) &= x_1x_2\bar{x}_3 \vee \bar{x}_2x_3 \vee \bar{x}_3x_4 \\ \\ f(x_1,x_2,x_3,x_4) &= \overline{\overline{x_1x_2\bar{x}_3 \vee \bar{x}_2x_3 \vee \bar{x}_3x_4}} \\ &= \overline{\overline{(x_1x_2\bar{x}_3)}.\overline{(\bar{x}_2x_3)}.\overline{(\bar{x}_3x_4)}} \\ &= (x_1 \uparrow x_2 \uparrow \bar{x}_3) \uparrow (\bar{x}_2 \uparrow x_3) \uparrow (\bar{x}_3 \uparrow x_4) \end{aligned}$$

$$\begin{aligned} b) f(x_1,x_2,x_3,x_4) &= x_1\bar{x}_3 \vee \bar{x}_2\bar{x}_4 \vee \bar{x}_4 \\ \\ f(x_1,x_2,x_3,x_4) &= \overline{\overline{x_1\bar{x}_3 \vee \bar{x}_2\bar{x}_4 \vee \bar{x}_4}} \\ &= \overline{\overline{(x_1\bar{x}_3)}.\overline{(\bar{x}_2\bar{x}_4)}.\overline{(\bar{x}_4)}} \\ &= (x_1 \uparrow \bar{x}_3) \uparrow (\bar{x}_2 \uparrow \bar{x}_4) \uparrow x_4 \end{aligned}$$

VAJA 6.5.2: Pretvorba KNO v PNO

Zapišite KNO logičnih funkcij z NOR operatorji.

$$\begin{aligned} a) f(x_1,x_2,x_3) &= (x_1 \vee x_3)(\bar{x}_1 \vee \bar{x}_2)\bar{x}_2 \\ f(x_1,x_2,x_3) &= \overline{\overline{(x_1 \vee x_3)(\bar{x}_1 \vee \bar{x}_2)\bar{x}_2}} \\ &= \overline{\overline{(x_1 \vee x_3)} \vee \overline{(\bar{x}_1 \vee \bar{x}_2)} \vee \overline{(\bar{x}_2)}} \\ &= (x_1 \downarrow x_3) \downarrow (\bar{x}_1 \downarrow \bar{x}_2) \downarrow x_2 \end{aligned}$$

$$\begin{aligned} b) f(x_1,x_2,x_3,x_4) &= (x_1 \vee x_3 \vee \bar{x}_4)(\bar{x}_1 \vee \bar{x}_2)(\bar{x}_2 \vee x_4) \\ \\ f(x_1,x_2,x_3,x_4) &= \overline{\overline{(x_1 \vee x_3 \vee \bar{x}_4)(\bar{x}_1 \vee \bar{x}_2)(\bar{x}_2 \vee x_4)}} \\ &= \overline{\overline{(x_1 \vee x_3 \vee \bar{x}_4)} \vee \overline{(\bar{x}_1 \vee \bar{x}_2)} \vee \overline{(\bar{x}_2 \vee x_4)}} \\ &= (x_1 \downarrow x_3 \downarrow \bar{x}_4) \downarrow (\bar{x}_1 \downarrow \bar{x}_2) \downarrow (\bar{x}_2 \downarrow x_4) \end{aligned}$$

VAJA 6.5.3

Zapišite DNO logične funkcije v dvonivojski obliki NOR/OR in OR/NAND.

$$f(x_1,x_2,x_3,x_4) = x_1 \vee x_2\bar{x}_3 \vee \bar{x}_1x_4$$

a) NOR/OR - negiramo konjunkcije na prvem nivoju

$$\begin{aligned} f(x_1,x_2,x_3,x_4) &= x_1 \vee \overline{\overline{x_2\bar{x}_3}} \vee \overline{\overline{\bar{x}_1x_4}} \\ &= x_1 \vee \overline{(\bar{x}_2 \vee x_3)} \vee \overline{(x_1 \vee \bar{x}_4)} \\ &= x_1 \vee (\bar{x}_2 \downarrow x_3) \vee (x_1 \downarrow \bar{x}_4) \end{aligned}$$

b) OR/NAND - negiramo celotno funkcijo

$$f(x_1,x_2,x_3,x_4) = \overline{\overline{x_1 \vee x_2\bar{x}_3 \vee \bar{x}_1x_4}}$$

$$\begin{aligned} &= \overline{\bar{x}_1.\overline{(x_2\bar{x}_3)}.\overline{(\bar{x}_1x_4)}} \\ &= \bar{x}_1 \uparrow (\bar{x}_2 \vee x_3) \uparrow (x_1 \vee \bar{x}_4) \end{aligned}$$

VAJA 6.5.4:

Zapišite KNO logičnih funkcij v dvonivojski obliki AND/NOR in NAND/AND.

$$f(x_1, x_2, x_3, x_4) = x_3(\bar{x}_1 \vee \bar{x}_2 \vee x_3)(x_1 \vee \bar{x}_2 \vee x_4)$$

a) AND/NOR - negiramo celotno funkcijo

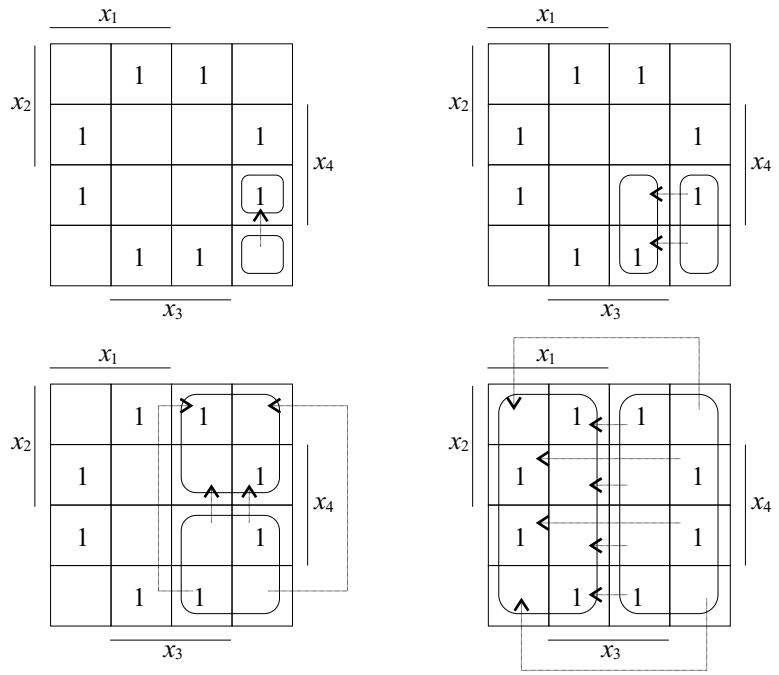
$$\begin{aligned} f(x_1, x_2, x_3, x_4) &= \overline{\overline{x_3(\bar{x}_1 \vee \bar{x}_2 \vee x_3)(x_1 \vee \bar{x}_2 \vee x_4)}} \\ &= \overline{\bar{x}_3 \vee \overline{(\bar{x}_1 \vee \bar{x}_2 \vee x_3)} \vee \overline{(x_1 \vee \bar{x}_2 \vee x_4)}} \\ &= \bar{x}_3 \downarrow (x_1x_2\bar{x}_3) \downarrow (\bar{x}_1x_2\bar{x}_4) \end{aligned}$$

b) NAND/AND - negiramo disjunkcije na prvem nivoju

$$\begin{aligned} f(x_1, x_2, x_3, x_4) &= x_3.\overline{\overline{(\bar{x}_1 \vee \bar{x}_2 \vee x_3)}}.\overline{\overline{(x_1 \vee \bar{x}_2 \vee x_4)}} \\ &= x_3.\overline{(x_1x_2\bar{x}_3)}.\overline{(\bar{x}_1x_2\bar{x}_4)} \\ &= x_3.(x_1 \uparrow x_2 \uparrow \bar{x}_3).(\bar{x}_1 \uparrow x_2 \uparrow \bar{x}_4) \end{aligned}$$

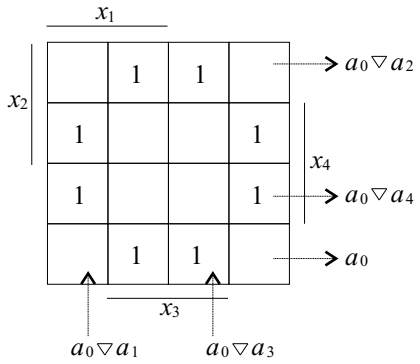
VAJA 6.5.5:

Če je logična funkcija $f(x_1, x_2, x_3, x_4)=\vee(1, 2, 5, 6, 9, 10, 13, 14)$ linearna, jo zapišite v obliki linearnega polinoma.



Slika 6.8: Prepogibanje likov za ugotavljanje linearnosti

Prepognemo kvadrat z mintermom m_0 navzgor proti spremenljivki x_4 in nato oba skupaj proti spremenljivki x_3. Rezultat prepogiba je negacija vrednosti, kar izpolnjuje pogoj linearnosti, zato nadaljujemo in prepognemo štiri kvadrate skupaj proti spremenljivki x_2. Rezultat prepogiba je enakost v vseh štirih kvadratih, zato nadaljujemo z zadnjim prepogibom osmih kvadratov proti x_1, kjer imamo znova enakost. Pregledali smo cel Veitchev diagram in ker smo povsod dobili izpolnjen pogoj enakosti ali negacije pomeni, da je logična funkcija linearna (Slika 6.8).



Slika 6.9: Linearne enačbe za a_0, a_1, a_2, a_3, a_4

Iz linearnih enačb (Slika 6.9) izračunamo koeficiente a_0, a_1, a_2, a_3, a_4, jih vstavimo v enačbo in dobimo zapis v obliki linearnega polinoma.

$$\begin{array}{lll} & & \Rightarrow a_0 = 0 \\ a_0 \nabla a_4 = 1 & 0 \nabla a_4 = 1 & \Rightarrow a_4 = 1 \\ a_0 \nabla a_3 = 1 & 0 \nabla a_3 = 1 & \Rightarrow a_3 = 1 \\ a_0 \nabla a_2 = 0 & 0 \nabla a_2 = 0 & \Rightarrow a_2 = 0 \\ a_0 \nabla a_1 = 0 & 0 \nabla a_1 = 0 & \Rightarrow a_1 = 0 \end{array}$$

$$\begin{aligned} f(x_1, x_2, x_3, x_4) &= a_0 \nabla a_1 x_1 \nabla a_2 x_2 \nabla a_3 x_3 \nabla a_4 x_4 \\ &= 0 \nabla 0.x_1 \nabla 0.x_2 \nabla 1.x_3 \nabla 1.x_4 \\ &= x_3 \nabla x_4 \end{aligned}$$

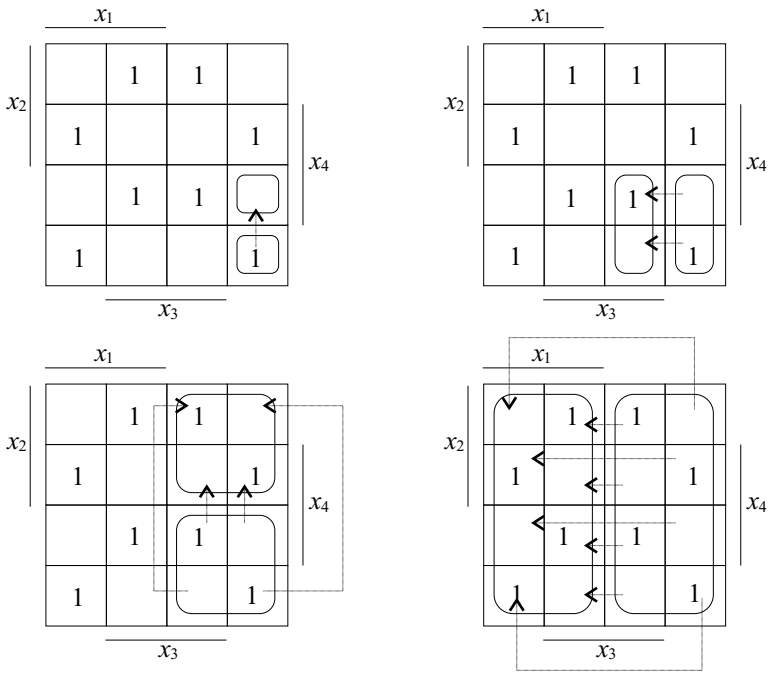
VAJA 6.5.6:

Pokažite, da je logična funkcija linearna, izračunajte koeficiente in jo zapišite v obliki linearnega polinoma.

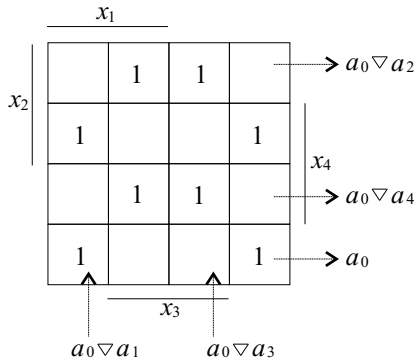
$$f(x_1, x_2, x_3, x_4) = x_2 x_3 \bar{x}_4 \vee \bar{x}_2 x_3 x_4 \vee x_2 \bar{x}_3 x_4 \vee \bar{x}_2 \bar{x}_3 \bar{x}_4$$

Vpišemo disjunktivno normalno obliko logične funkcije v Veitchev diagram, tako da za vsak konjunktiven izraz poiščemo ustrezne pravokotnike in vanje vpišemo funkcijske vrednosti 1.

Prepognemo kvadrat z mintermom m_0 proti x_4 in nato oba skupaj proti spremenljivki x_3. Rezultat prepogiba je negacija vrednosti, kar izpolnjuje pogoj linearnosti, zato prepognemo štiri kvadrate proti spremenljivki x_2. Rezultat prepogiba je negacija v vseh štirih kvadratih, zato nadaljujemo z zadnjim prepogibom osmih kvadratov proti x_1, kjer imamo enakost. Pregledali smo cel Veitchev diagram in ker smo povsod dobili izpolnjen pogoj enakosti ali negacije pomeni, da je logična funkcija linearna (Slika 6.10).



Slika 6.10: Prepogibanje likov za ugotavljanje linearnosti



Slika 6.11: Linearne enačbe za a_0, a_1, a_2, a_3, a_4

Iz enačb (Slika 6.11) izračunamo koeficiente a_0, a_1, a_2, a_3, a_4 in jih vstavimo v linearni polinom logične funkcije štirih spremenljivk.

$$\begin{array}{lll} & & \Rightarrow a_0 = 1 \\ a_0 \nabla a_4 = 0 & 1 \nabla a_4 = 0 & \Rightarrow a_4 = 1 \\ a_0 \nabla a_3 = 0 & 1 \nabla a_3 = 0 & \Rightarrow a_3 = 1 \\ a_0 \nabla a_2 = 0 & 1 \nabla a_2 = 0 & \Rightarrow a_2 = 1 \\ a_0 \nabla a_1 = 1 & 1 \nabla a_1 = 1 & \Rightarrow a_1 = 0 \end{array}$$

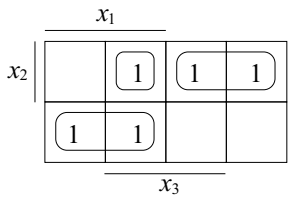
$$\begin{aligned} f(x_1, x_2, x_3, x_4) &= a_0 \nabla a_1 x_1 \nabla a_2 x_2 \nabla a_3 x_3 \nabla a_4 x_4 \\ &= 1 \nabla 0.x_1 \nabla 1.x_2 \nabla 1.x_3 \nabla 1.x_4 = 1 \nabla x_2 \nabla x_3 \nabla x_4 \end{aligned}$$

VAJA 6.5.7:

Zapišite podano logično funkcijo v Reed-Mullerjevi obliki.

$$f(x_1, x_2, x_3) = \bar{x}_1 x_2 \vee x_1 \bar{x}_2 \vee x_1 x_2 x_3$$

Logično funkcijo bomo vpisali v Veitchev diagram in poiskali ortogonalno minimalno disjunktivno normalno obliko (OMDNO) tako, da bo vsaka enica pri določanju glavnih vsebovalnikov oziroma vsebovalnikov upoštevana samo enkrat (Slika 6.12).



Slika 6.12: Veitchev diagram (OMDNO)

Prevedba funkcije iz OMDNO v Reed-Mullerjevo obliko:

$$\begin{aligned} f(x_1, x_2, x_3) &= \bar{x}_1 x_2 \vee x_1 \bar{x}_2 \vee x_1 x_2 x_3 \\ &= \bar{x}_1 x_2 \nabla x_1 \bar{x}_2 \nabla x_1 x_2 x_3 \\ &= (x_1 \nabla 1) x_2 \nabla x_1 (x_2 \nabla 1) \nabla x_1 x_2 x_3 \\ &= x_1 x_2 \nabla x_2 \nabla x_1 x_2 \nabla x_1 \nabla x_1 x_2 x_3 \\ &= x_1 \nabla x_2 \nabla x_1 x_2 x_3 \end{aligned}$$

VAJA 6.5.8:

Zapišite podano logično funkcijo v Reed-Mullerjevi obliki.

$$f(x_1, x_2, x_3, x_4) = \bar{x}_1 x_2 \bar{x}_3 x_4 \vee x_1 x_2 x_3 \bar{x}_4$$

Logična funkcija je že podana v PDNO, ki je osnova za pretvorbo v Reed-Mullerjevo obliko.

$$\begin{aligned} f(x_1, x_2, x_3, x_4) &= (x_1 \nabla 1) x_2 (x_3 \nabla 1) x_4 \nabla x_1 x_2 x_3 (x_4 \nabla 1) \\ &= (x_1 x_2 \nabla x_2)(x_3 x_4 \nabla x_4) \nabla x_1 x_2 x_3 x_4 \nabla x_1 x_2 x_3 \\ &= x_1 x_2 x_3 x_4 \nabla x_2 x_3 x_4 \nabla x_1 x_2 x_4 \nabla x_2 x_4 \nabla x_1 x_2 x_3 x_4 \nabla x_1 x_2 x_3 \\ &= x_2 x_3 x_4 \nabla x_1 x_2 x_4 \nabla x_1 x_2 x_3 \nabla x_2 x_4 \end{aligned}$$

VAJA 6.5.9:

Podano večnivojsko logično funkcijo zapišite v disjunktivni normalni obliki (DNO).

$$f(x_1, x_2, x_3, x_4) = (x_1 \nabla (x_3 \leftarrow x_2)) \vee (x_4 \downarrow (x_1 \rightarrow x_2))$$

$$\begin{aligned} &= \bar{x}_1 (x_3 \vee \bar{x}_2) \vee x_1 (\overline{x_3 \vee \bar{x}_2}) \vee (\overline{x_4 \vee (\bar{x}_1 \vee x_2)}) \\ &= \bar{x}_1 x_3 \vee \bar{x}_1 \bar{x}_2 \vee x_1 \bar{x}_3 x_2 \vee \bar{x}_4 (\overline{\bar{x}_1 \vee x_2}) \\ &= \bar{x}_1 x_3 \vee \bar{x}_1 \bar{x}_2 \vee x_1 x_2 \bar{x}_3 \vee \bar{x}_4 x_1 \bar{x}_2 \end{aligned}$$

VAJA 6.5.10:

Podano večnivojsko logično funkcijo zapišite v disjunktivni normalni obliki (DNO).

$$\begin{aligned} f(x_1,x_2,x_3) &= ((x_2 \equiv x_3) \rightarrow x_1) \uparrow (\bar{x}_1 \vee x_2) \\ &= \overline{\overline{((x_2 \equiv x_3)} \vee x_1)(\bar{x}_1 \vee x_2)} \\ &= \overline{\overline{((x_2 \equiv x_3)} \vee x_1)} \vee \overline{(\bar{x}_1 \vee x_2)} \\ &= (x_2 \equiv x_3)\bar{x}_1 \vee x_1\bar{x}_2 \\ &= \bar{x}_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1x_2x_3 \vee x_1\bar{x}_2 \end{aligned}$$

VAJA 6.5.11

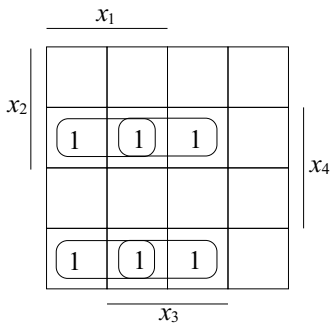
Podano večnivojsko logično funkcijo zapišite v disjunktivni normalni obliki (DNO).

$$\begin{aligned} f(x_1,x_2,x_3,x_4) &= ((x_1x_3) \downarrow \bar{x}_2)\nabla(\bar{x}_2 \vee x_4) \\ &= \overline{((x_1x_3) \downarrow \bar{x}_2)}.(\bar{x}_2 \vee x_4) \vee ((x_1x_3) \downarrow \bar{x}_2).\overline{(\bar{x}_2 \vee x_4)} \\ &= \overline{\overline{((x_1x_3) \vee \bar{x}_2)}.(\bar{x}_2 \vee x_4)} \vee \overline{((x_1x_3) \vee \bar{x}_2)}.(x_2\bar{x}_4) \\ &= (x_1x_3 \vee \bar{x}_2)(\bar{x}_2 \vee x_4) \vee ((\bar{x}_1 \vee \bar{x}_3)x_2)(x_2\bar{x}_4) \\ &= x_1\bar{x}_2x_3 \vee x_1x_3x_4 \vee \bar{x}_2 \vee \bar{x}_2x_4 \vee \bar{x}_1x_2\bar{x}_4 \vee x_2\bar{x}_3\bar{x}_4 \end{aligned}$$

VAJA 6.5.12

Poiščite dekompozicijo logične funkcije $f^4 = \vee(2,7,8,10,13,15)$ in jo zapišite v obliki večnivojske logične funkcije.

Za podano logično funkcijo najprej zapišemo MDNO (Slika 6.13), ki predstavlja dvonivojsko obliko.



Slika 6.13: Minimizacija

$$MDNO: f(x_1,x_2,x_3,x_4) = x_1x_2x_4 \vee x_1\bar{x}_2\bar{x}_4 \vee x_2x_3x_4 \vee \bar{x}_2x_3\bar{x}_4$$

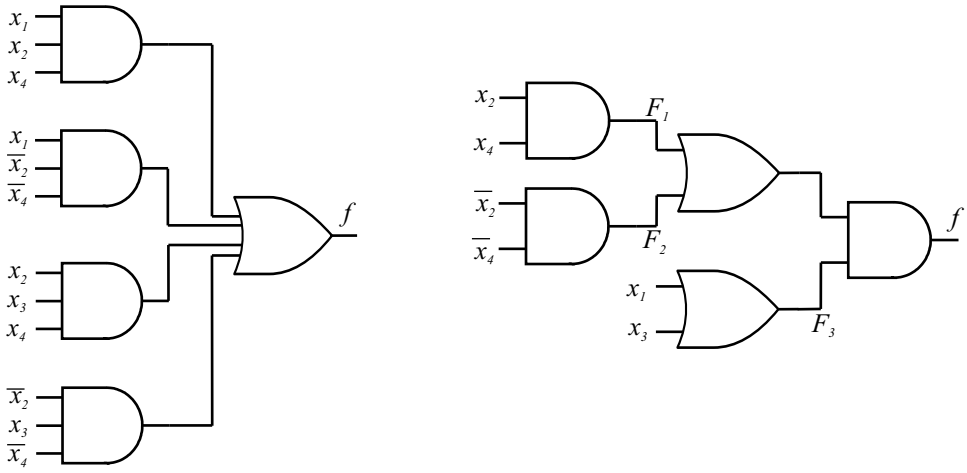
Za dekompozicijo logične funkcije vzamemo MDNO in jo z uporabo Boolovih postulatov in izrekov preoblikujemo v naslednji zapis:

$$\begin{aligned} f(x_1,x_2,x_3,x_4) &= x_1.(x_2x_4 \vee \bar{x}_2\bar{x}_4) \vee x_3.(x_2x_4 \vee \bar{x}_2\bar{x}_4) \\ &= (x_2x_4 \vee \bar{x}_2\bar{x}_4).(x_1 \vee x_3)\ . \end{aligned}$$

V zapisu imamo tri različne podfunkcije $F_1 = x_2x_4$, $F_2 = \bar{x}_2\bar{x}_4$, $F_3 = x_1 \vee x_3$, ki jih vstavimo v logično funkcijo in dobimo

$$f(x_1, x_2, x_3, x_4) \;=\; (F_1 \vee F_2).F_3 \;.$$

Logična funkcija je z uporabo dekompozicije postala trinivojska, uporabljena logična vrata so vsa dvovhodna (Slika 6.14).



Slika 6.14: Dvonivojska in trinivojska oblika logične funkcije

VAJA 6.5.13

Pri logičnih funkcijah F, G, H z uporabo ekstrakcije določite skupne podfunkcije in jih zapišite v novi obliki.

$$\begin{aligned} F &= \bar{x}_1.(x_2 \vee x_3) \vee \bar{x}_2\bar{x}_3 \vee x_1\bar{x}_3 \\ G &= \bar{x}_2x_3x_4 \vee \bar{x}_1\bar{x}_2\bar{x}_3 \vee x_4x_5 \\ H &= x_1\bar{x}_3 \vee (x_2 \vee x_3).x_4 \vee \bar{x}_3x_4x_5 \end{aligned}$$

Skupne podfunkcije v zgornjih logičnih funkcijah so $F_1 = x_2 \vee x_3$, $F_2 = x_1\bar{x}_3$, $F_3 = x_4x_5$. Zapis logičnih funkcij po ekstrakciji je enak:

$$\begin{aligned} F &= \bar{x}_1.F_1 \vee \bar{F}_1 \vee F_2 \\ G &= \bar{x}_2x_3x_4 \vee \bar{x}_1\bar{F}_1 \vee F_3 \\ H &= F_2 \vee F_1x_4 \vee \bar{x}_3F_3 \end{aligned}$$

NALOGE:

6.1 Dvonivojske oblike logičnih funkcij:

$a) f^4 = \vee(0, 1, 4, 6, 9, 10, 12, 15)$

NAND/NAND (iz PDNO):
$f^4 = \uparrow (0, 1, 4, 6, 9, 10, 12, 15)$

NOR/NOR (iz PKNO):
$f^4 = \downarrow (13, 12, 10, 8, 7, 4, 2, 1)$

AND/NOR (iz MKNO):
$f(x_1, x_2, x_3, x_4) = (x_1x_2x_3\bar{x}_4) \downarrow (x_2\bar{x}_3x_4) \downarrow (\bar{x}_1x_2x_4) \downarrow (\bar{x}_2x_3x_4) \downarrow (\bar{x}_1\bar{x}_2x_3) \downarrow (x_1\bar{x}_2\bar{x}_3\bar{x}_4)$

$b) f^4 = \&(15, 11, 10, 8, 5, 3, 2, 0)$

NAND/NAND (iz MDNO):
$f(x_1, x_2, x_3, x_4) = (\bar{x}_2 \uparrow x_4) \uparrow (x_2 \uparrow x_3 \uparrow \bar{x}_4) \uparrow (x_1 \uparrow \bar{x}_2 \uparrow \bar{x}_3) \uparrow (\bar{x}_1 \uparrow \bar{x}_2 \uparrow x_3)$

OR/NAND (iz MDNO):
$f(x_1, x_2, x_3, x_4) = (x_2 \vee \bar{x}_4) \uparrow (\bar{x}_2 \vee \bar{x}_3 \vee x_4) \uparrow (\bar{x}_1 \vee x_2 \vee x_3) \uparrow (x_1 \vee x_2 \vee \bar{x}_3)$

6.2 Za podane logične funkcije ugotovite ali so linearne in jih zapišite v obliki linearnega polinoma.

1) $f^3 = \&(7, 4, 2, 1)$
R: $f(x_1, x_2, x_3) = x_1 \nabla x_2 \nabla x_3$

2) $f^3 = \&(6, 5, 3, 2)$
R: $f(x_1, x_2, x_3)$ ni linearna.

3) $f(x_1, x_2, x_3, x_4) = x_1\bar{x}_3\bar{x}_4 \vee x_1x_3x_4 \vee \bar{x}_1x_3\bar{x}_4 \vee \bar{x}_1\bar{x}_3x_4$
R: $f(x_1, x_2, x_3, x_4) = x_1 \nabla x_3 \nabla x_4$

6.3 Zapišite podane preklopne funkcije v Reed-Mullerjevi obliki:

1) $f(x_1, x_2, x_3) = x_1x_2 \vee \bar{x}_1\bar{x}_2\bar{x}_3$
R: $f(x_1, x_2, x_3) = 1 \nabla x_1 \nabla x_2 \nabla x_3 \nabla x_1x_3 \nabla x_2x_3 \nabla x_1x_2x_3$

2) $f(x_1, x_2, x_3, x_4) = (x_1 \uparrow x_2)(x_2 \equiv x_3)x_4 \vee (x_1 \uparrow x_2)\bar{x}_3x_4$
R: $f(x_1, x_2, x_3, x_4) = x_4 \nabla x_3x_4 \nabla x_1x_2x_4 \nabla x_2x_3x_4$

3) $f(x_1, x_2, x_3, x_4) = \bar{x}_2x_3 \vee x_3\bar{x}_4 \vee x_2\bar{x}_3x_4$
R: $f(x_1, x_2, x_3, x_4) = x_3 \nabla x_2x_4$

Poglavje 7

Aritmetična vezja

Aritmetična vezja bomo obravnavali kot primere razvoja kombinacijskih vezij, zgrajenih iz logičnih vrat tako, da je izhod v vsakem trenutku določen s trenutno kombinacijo vhodov. Za različne sheme seštevanja in odštevanja števil bomo uporabili nepredznačna, pozitivna in negativna števila v dvojiškem sistemu. Aritmetična vezja so značilni primeri kompromisa med hitrostjo in kompleksnostjo vezja.

Načrtovanje aritmetičnih vezij obsega:

1) določanje vhodnih spremenljivk in izhodnih funkcij iz opisa problema,

2) zapis izhodnih funkcij v pravilnostno tabelo,

3) analitičen zapis in minimizacijo izhodnih funkcij,

4) risanje logične sheme kombinacijskega vezja (povezava logičnih vrat ali gradnikov, ki so potrebni za izvedbo izhodnih funkcij).

7.1 Polovični in polni seštevalnik

Polovični seštevalnik

Polovični seštevalnik je najbolj enostavno aritmetično vezje z dvema vhodoma, ki ju seštejemo in kot rezultat dobimo dvojiško vsoto in prenos (ang. carry).

1) Vhodni spremenljivki: x_0, y_0, izhodni funkciji: dvojiška vsota $z_0 = x_0 + y_0$ in prenos c_0.

2) Pravilnostna tabela (Tabela 7.1)

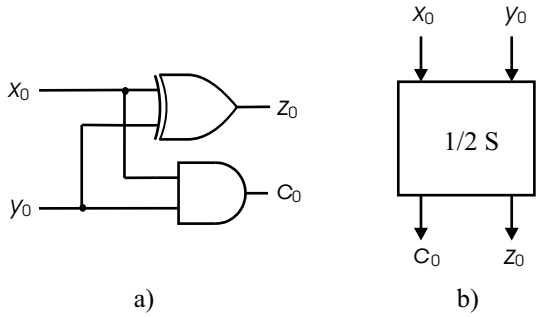
Tabela 7.1: Polovični seštevalnik

x_0	y_0	z_0	c_0
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

3) Zapis logičnih funkcij in minimizacija:

$$\begin{aligned} z_0 &= \bar{x}_0 y_0 \vee x_0 \bar{y}_0 = x_0 \nabla y_0 \\ c_0 &= x_0 y_0 \end{aligned}$$

4) Logična shema prikazuje vsoto z_0 kot izhod XOR vrat in prenos c_0 kot izhod AND vrat (Slika 7.1.a). Polovični seštevalnik bomo uporabili kot osnovno enoto 1-bitnega seštevanja za aritmetične operacije n-bitnih števil, zato ga rišemo v obliki bloka (Slika 7.1.b).



Slika 7.1: Polovični seštevalnik: a) logična shema, b) blok shema

Polni seštevalnik

Pri seštevanju večjega števila bitov nam polovični seštevalnik ne zadošča. Pri ročnem seštevanju vemo, da je prenos iz $i-1$ kolone seštevanja potrebno prišteti vsoti i-te kolone. Zato polovičnemu seštevalniku na vhodu dodamo še prenos iz prejšnjega mesta.

1) Vhodne spremenljivke so: x_0, y_0, c_{in}, izhodni funkciji sta: dvojiška vsota $z_0 = x_0 + y_0 + c_{in}$ in prenos c_0.

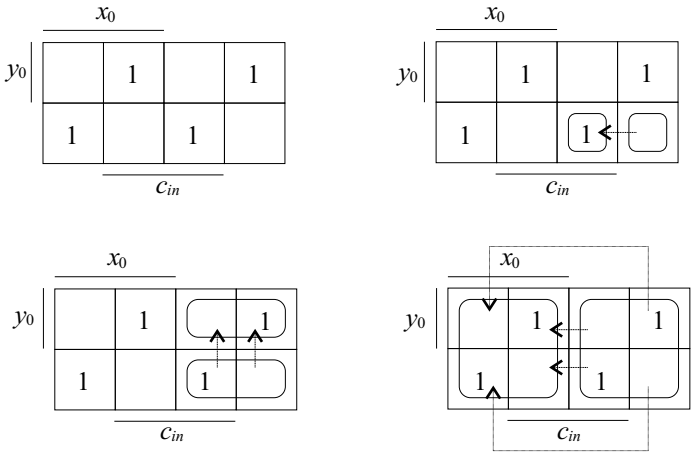
2) Pravilnostna tabela (Tabela 7.2)

Tabela 7.2: Polni seštevalnik

x_0	y_0	c_{in}	z_0	c_0
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

3) Zapis logičnih funkcij in minimizacija:

$$\begin{aligned} z_0 &= \bar{x}_0 \bar{y}_0 c_{in} \vee \bar{x}_0 y_0 \bar{c}_{in} \vee x_0 \bar{y}_0 \bar{c}_{in} \vee x_0 y_0 c_{in} \\ c_0 &= \bar{x}_0 y_0 c_{in} \vee x_0 \bar{y}_0 c_{in} \vee x_0 y_0 \bar{c}_{in} \vee x_0 y_0 c_{in} \end{aligned}$$



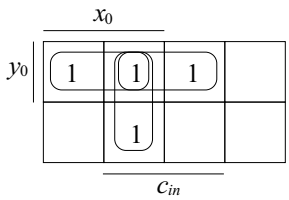
Slika 7.2: Ugotavljanje linearnosti izhoda z_0

Logična funkcija seštevanja se ne da minimizirati v smislu DNO. Ker imamo operacijo seštevanja po modulu 2, poglejmo kako je z njeno izvedbo z XOR vrati. Grafično hitro ugotovimo, da je funkcija linearna (Slika 7.2).

Izračunamo koeficiente a_0, a_1, a_2, a_3 in zapišemo linearni polinom.

$$\begin{aligned} a_0 &= 0 \\ a_0 \nabla a_1 = 1 &\rightarrow a_1 = 1 \\ a_0 \nabla a_2 = 1 &\rightarrow a_2 = 1 \\ a_0 \nabla a_3 = 1 &\rightarrow a_3 = 1 \end{aligned}$$

Izhod za prenos c_0 minimiziramo (Slika 7.3).

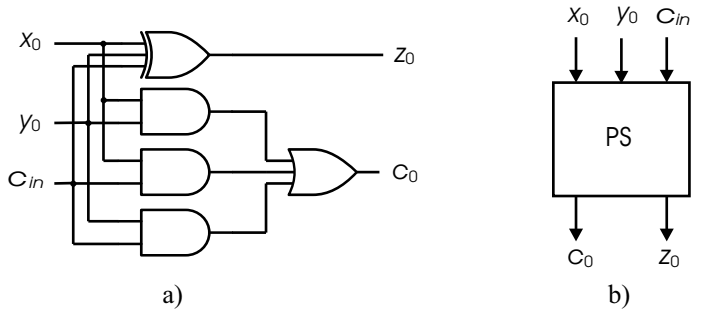


Slika 7.3: Minimizacija izhoda c_0

Izhodni funkciji polnega seštevalnika sta:

$$\begin{aligned} z_0 &= a_0 \nabla a_1 x_0 \nabla a_2 y_0 \nabla a_3 c_{in} = x_0 \nabla y_0 \nabla c_{in} \\ c_0 &= x_0 y_0 \vee x_0 c_{in} \vee y_0 c_{in} \ . \end{aligned}$$

4) V logični shemi polnega seštevalnika imamo za izhodno funkcijo z_0 uporabljena ena XOR vrata in za c_0 dvonivojsko povezavo treh AND vrat na prvem in enih OR vrat na drugem nivoju (Slika 7.4.a). Polni seštevalnik bomo uporabili kot osnovno enoto 1-bitnega seštevanja na mestih s prenosom, zato ga predstavljamo z blok shemo (Slika 7.4.b).



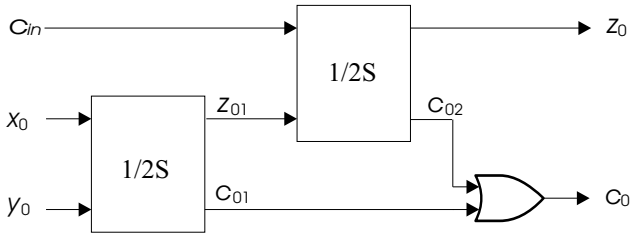
Slika 7.4: Polni seštevalnik: a) logična shema, b) blok shema

Gradnja polnega seštevalnika z dvema polovičnima seštevalnikoma

Če vzamemo polovični seštevalnik kot najmanjšo enoto seštevanja, ga lahko uporabimo pri gradnji polnega seštevalnika. Vsota na izhodu polnega seštevalnika je lahko podana kot rezultat dveh korakov seštevanja $z_0 = (x_0 + y_0) + c_{in}$ (Slika 7.5). S prvim polovičnim seštevalnikom najprej seštejemo $z_{01} = x_0 + y_0$. Rezultatu prištejemo prenos c_{in} na drugem polovičnem seštevalniku in dobimo vsoto z_0 in prenos c_{02}. Prenos c_0 na izhodu polnega seštevalnika je disjunkcija prenosov prvega polovičnega seštevalnika c_{01} in drugega polovičniga seštevalnika c_{02}.

Iz prenosa c_0 izračunajmo, da pri povezavi dveh polovičnih seštevalnikov v polni seštevalnik potrebujemo OR vrata.

$$\begin{aligned} c_0 &= \bar{x}_0 y_0 c_{in} \vee x_0 \bar{y}_0 c_{in} \vee x_0 y_0 \bar{c}_{in} \vee x_0 y_0 c_{in} \\ &= c_{in}(\bar{x}_0 y_0 \vee x_0 \bar{y}_0) \vee x_0 y_0 (\bar{c}_{in} \vee c_{in}) \\ &= c_{in}(x_0 \nabla y_0) \vee x_0 y_0 \\ &= c_{in} z_{01} \vee c_{01} \\ &= c_{02} \vee c_{01} \end{aligned}$$



Slika 7.5: Polni seštevalnik iz dveh polovičnih seštevalnikov in OR vrat

7.2 Polovični in polni odštevalnik

Polovični odštevalnik

Polovični odštevalnik je enostavno aritmetično vezje z dvema vhodoma, ki ju odštejemo in dvema izhodoma in kot rezultat dobimo razliko in sposodek (ang. borrow).

1) Vhodni spremenljivki sta: x_0, y_0, izhodni funkciji sta: razlika $d_0 = x_0 - y_0$ in sposodek b_0.

2) Pravilnostna tabela polovičnega odštevalnika (Tabela 7.3)

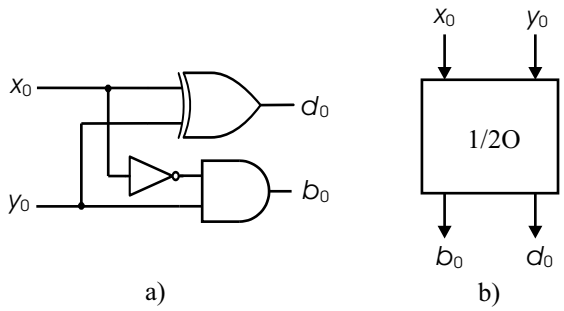
Tabela 7.3: Polovični odštevalnik

x_0	y_0	d_0	b_0
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

3) Zapis logičnih funkcij in minimizacija:

$$\begin{aligned} d_0 &= \bar{x}_0 y_0 \vee x_0 \bar{y}_0 = x_0 \nabla y_0 \\ b_0 &= \bar{x}_0 y_0 \end{aligned}$$

4) Logična shema prikazuje razliko d_0 kot izhod XOR vrat in sposodek b_0 kot izhod AND vrat (Slika 7.6.a). Polovični odštevalnik bomo uporabili kot osnovno enoto 1-bitnega odštevanja za aritmetične operacije n-bitnih števil, zato ga rišemo v obliki bloka (Slika 7.6.b).



Slika 7.6: Polovični odštevalnik: a) logična shema, b) blok shema

Polni odštevalnik

Pri odštevanju večjega števila bitov nam polovični odštevalnik ne zadošča, ker je sposodek iz $i-1$ kolone odštevanja potrebno odšteti od razlike i-te kolone. Zato polovičnemu odštevalniku na vhodu dodamo še sposodek iz prejšnjega mesta.

1) Vhodne spremenljivke so: x_0, y_0, b_{in}, izhodni funkciji sta: razlika $d_0 = x_0 - y_0 - b_{in}$ in sposodek b_0.

2) Pravilnostna tabela (Tabela 7.4)

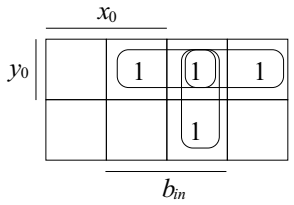
3) Zapis logičnih funkcij in minimizacija:

$$\begin{aligned} d_0 &= \bar{x}_0 \bar{y}_0 b_{in} \vee \bar{x}_0 y_0 \bar{b}_{in} \vee x_0 \bar{y}_0 \bar{b}_{in} \vee x_0 y_0 b_{in} \\ b_0 &= \bar{x}_0 \bar{y}_0 b_{in} \vee \bar{x}_0 y_0 \bar{b}_{in} \vee \bar{x}_0 y_0 b_{in} \vee x_0 y_0 b_{in} \end{aligned}$$

Logična funkcija odštevanja d_0 se ne da minimizirati v smislu DNO. V pravilnostni tabeli vidimo, da je operacija odštevanja v dvojiškem sistemu enaka operaciji seštevanja po modulu 2, zato jo izvedemo z XOR vrati.

Tabela 7.4: Polni odštevalnik

x_0	y_0	b_{in}	d_0	b_0
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

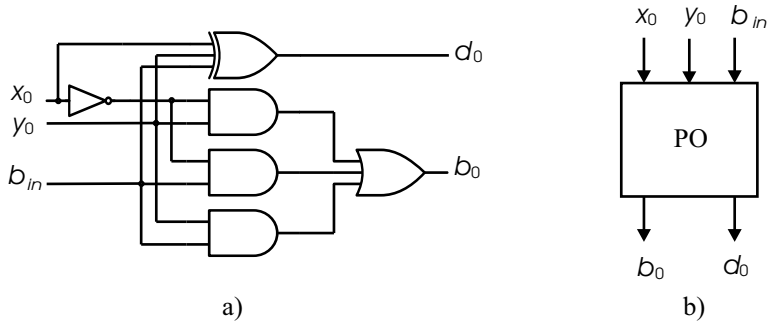


Slika 7.7: Minimizacija izhoda b_0

Sposodek b_0 minimiziramo (Slika 7.7) in zapišemo izhoda polnega odštevalnika.

$$d_0 = a_0 \nabla a_1 x_0 \nabla a_2 y_0 \nabla a_3 b_{in} = x_0 \nabla y_0 \nabla b_{in}$$
$$b_0 = \bar{x}_0 y_0 \vee \bar{x}_0 b_{in} \vee y_0 b_{in}$$

4) V logični shemi polnega odštevalnika imamo za izhodno funkcijo d_0 uporabljena ena XOR vrata in za b_0 dvonivojsko povezavo treh AND vrat z negiranim vhodom x_0 na prvem in enih OR vrat na drugem nivoju (Slika 7.8.a). Polni odštevalnik bomo uporabili kot osnovno enoto 1-bitnega odštevanja na mestih s prenosom zato, ga predstavljamo z blok shemo (Slika 7.8.b).



Slika 7.8: Polni odštevalnik: a) logična shema, b) blok shema

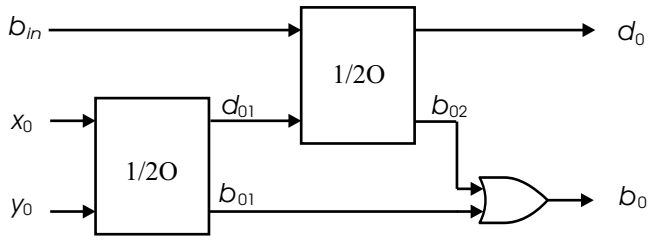
Gradnja polnega odštevalnika z dvema polovičnima odštevalnikoma

Razlika na izhodu polnega odštevalnika je lahko podana kot rezultat odštevanja v dveh korakih $d_0 = (x_0 - y_0) - b_{in}$. S prvim polovičnim odštevalnikom najprej odštejemo $d_{01} = x_0 - y_0$ (Slika 7.9). Rezultatu odštejemo sposodek b_{in} na drugem polovičnem odštevalniku in dobimo razliko d_0. Sposodek na izhodu polnega odštevalnika b_0 je disjunkcija sposodkov prvega polovičnega odštevalnika b_{01} in drugega polovičnega odštevalnika

b_{02}.

Iz sposodka b_0 izračunamo, da pri povezavi dveh polovičnih odštevalnikov v polni odštevalnik potrebujemo OR vrata.

$$\begin{aligned} b_0 &= \bar{x}_0\bar{y}_0 b_{in} \vee \bar{x}_0 y_0 \bar{b}_{in} \vee \bar{x}_0 y_0 b_{in} \vee x_0 y_0 b_{in} \\ &= b_{in}(\bar{x}_0\bar{y}_0 \vee x_0 y_0) \vee \bar{x}_0 y_0(\bar{b}_{in} \vee b_{in}) \\ &= b_{in}\overline{(x_0 \bigtriangledown y_0)} \vee \bar{x}_0 y_0 \\ &= b_{in}\bar{d}_{01} \vee b_{01} \\ &= b_{02} \vee b_{01} \end{aligned}$$

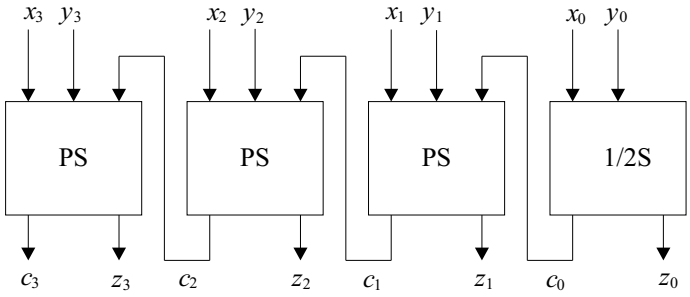


Slika 7.9: Polni odštevalnik iz dveh polovičnih odštevalnikov in OR vrat

7.3 Seštevalniki in odštevalniki

Polovični in polni seštevalnik ali odštevalnik so vezja, ki izvedejo 1-bitno aritmetično operacijo. Ker imamo v digitalnih sistemih opraviti z večbitnimi podatki (4 biti, 8 bitov, 16 bitov, 64 bitov,...), moramo večmestno seštevanje ali odštevanje izvesti s povezovanjem ustreznega števila polovičnih in polnih seštevalnikov/odštevalnikov.

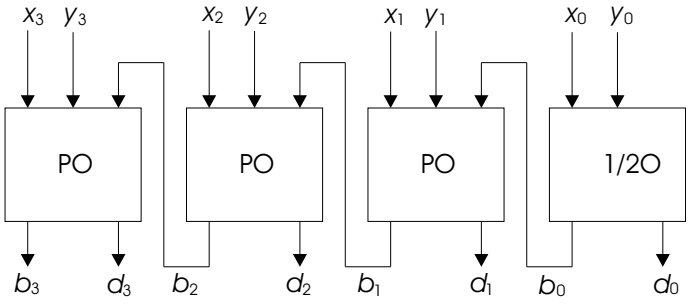
4-bitni seštevalnik je zgrajen iz enega polovičnega in treh polnih seštevalnikov. Oglejmo si seštevanje dveh 4-bitnih besed $X = (x_3, x_2, x_1, x_0)$ in $Y = (y_3, y_2, y_1, y_0)$. Vsota $Z = X + Y$ je tudi 4-bitna beseda $Z = (z_3, z_2, z_1, z_0)$ (Slika 7.10).



Slika 7.10: 4-bitni seštevalnik

Zadnji bit 4-bitnega seštevalnika z_0 izvedemo s polovičnim seštevalnikom, za ostale tri bite z_1, z_2, z_3 pa so potrebni polni seštevalniki, ker prihaja pri seštevanju do prenosa c_i, ki ga moramo upoštevati v operaciji seštevanja na višjem mestu. Zadnji prenos seštevanja c_3 lahko pomeni prekoračitev (ang. overflow) in je aktiven takrat, ko je vsota dveh pozitivnih števil tako velika, da je ne moremo zapisati s podanim številom bitov.

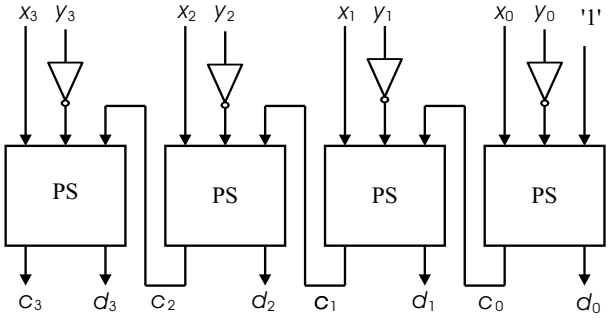
4-bitni odštevalnik je zgrajen iz enega polovičnega in treh polnih odštevalnikov. Oglejmo si odštevanje dveh 4-bitnih besed $X = (x_3, x_2, x_1, x_0)$ in $Y = (y_3, y_2, y_1, y_0)$. Razlika $D = X - Y$ je 4-bitna beseda $D = (d_3, d_2, d_1, d_0)$ (Slika 7.11).



Slika 7.11: 4-bitni odštevalnik

4-bitni odštevalnik izveden s polnimi seštevalniki
Odštevanje zelo pogosto izvedemo kar s seštevalnikom, tako da poiščemo negativno vrednost druge besede in jo prištejemo k prvi $D = X - Y = X + (-Y)$.

Zgradimo 4-bitni odštevalnik z uporabo polnih seštevalnikov. Negativno število $-Y$ dobimo tako, da izračunamo njegov dvojiški komplement, ki ga dobimo z negacijo vhodov y_i in logično konstanto 1 na prenosu iz najnižjega mesta $c_{in} = 1$ (Slika 7.12).



Slika 7.12: 4-bitni seštevalnik za odštevanje števil $D = X + (-Y)$

7.4 Uporaba vezij za izračun prenosov

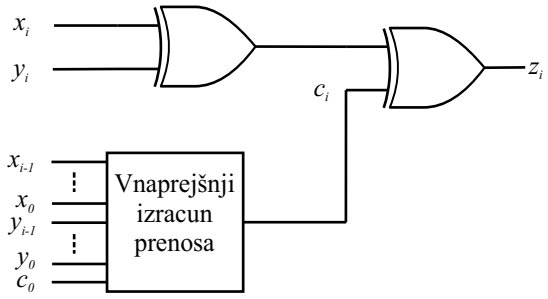
Pri uporabi aritmetičnih operacij v digitalnih sistemih želimo imeti vezja, ki zelo hitro posredujejo rezultat operacije. Upoštevanje prenosa v opisanih seštevalnikih in odštevalnikih za n-bitov zahteva $2n$ korakov, da dobimo rezultat. Ker želimo čas izvajanja operacij čim bolj skrajšati, uporabimo vezja, ki prenose izračunajo vnaprej (ang. Carry Lookahead Circuits).

Vezje za izračun prenosov

Izhodni funkciji polnega seštevalnika enostavno zapišemo kot

$$\begin{aligned} z_i &= x_i \nabla y_i \nabla c_i \\ c_{i+1} &= x_i y_i \vee x_i c_i \vee y_i c_i \ . \end{aligned}$$

V n-bitnem seštevalniku je prenos c_i na vsakem mestu funkcija prenosov z vseh nižjih mest, zato je osnovna ideja pri računanju prenosa vnaprej direktno izraziti vsak c_i z x_i, x_{i-1},...,x_0, y_i, y_{i-1},...,y_0 in c_0 (Slika 7.13). Ta funkcija je zelo obsežna, vendar se da vedno izraziti kot dvonivojska funkcija, če želimo pri izračunu izhoda imeti zakasnitve preko dveh nivojev vrat.



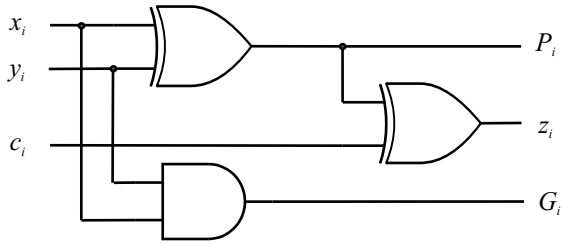
Slika 7.13: Seštevanje i-tega bita z vezjem za izračun prenosa

Zapišimo dve novi funkciji $G_i = x_i y_i$ in $P_i = x_i \nabla y_i$ za izračun prenosa. Iz prejšnje analize prenosa vemo, da je ta 1, če sta x_i in y_i oba 1, zato to funkcijo imenujemo generiranje prenosa (G_i).
Če je eden ali drugi vhod 1, pa bo prenos naprej takšen, kot je prenos prišel na vhod, oziroma kadar bo XOR funkcija med x_i in y_i enaka 1, se bo prenos samo prenesel naprej (P_i). Vsota in prenos sta kot funkcija signalov generiranja G_i in prenašanja P_i izražena z

$$z_i = x_i \nabla y_i \nabla c_i$$

$$\begin{aligned} c_{i+1} &= x_i y_i \vee x_i c_i \vee y_i c_i \\ &= x_i y_i \vee c_i(x_i \vee y_i) \\ &= x_i y_i \vee c_i(x_i \nabla y_i) \\ &= G_i \vee c_i P_i \end{aligned}$$

Slika 7.14 predstavlja izhodno vezje s signaloma G_i in P_i.

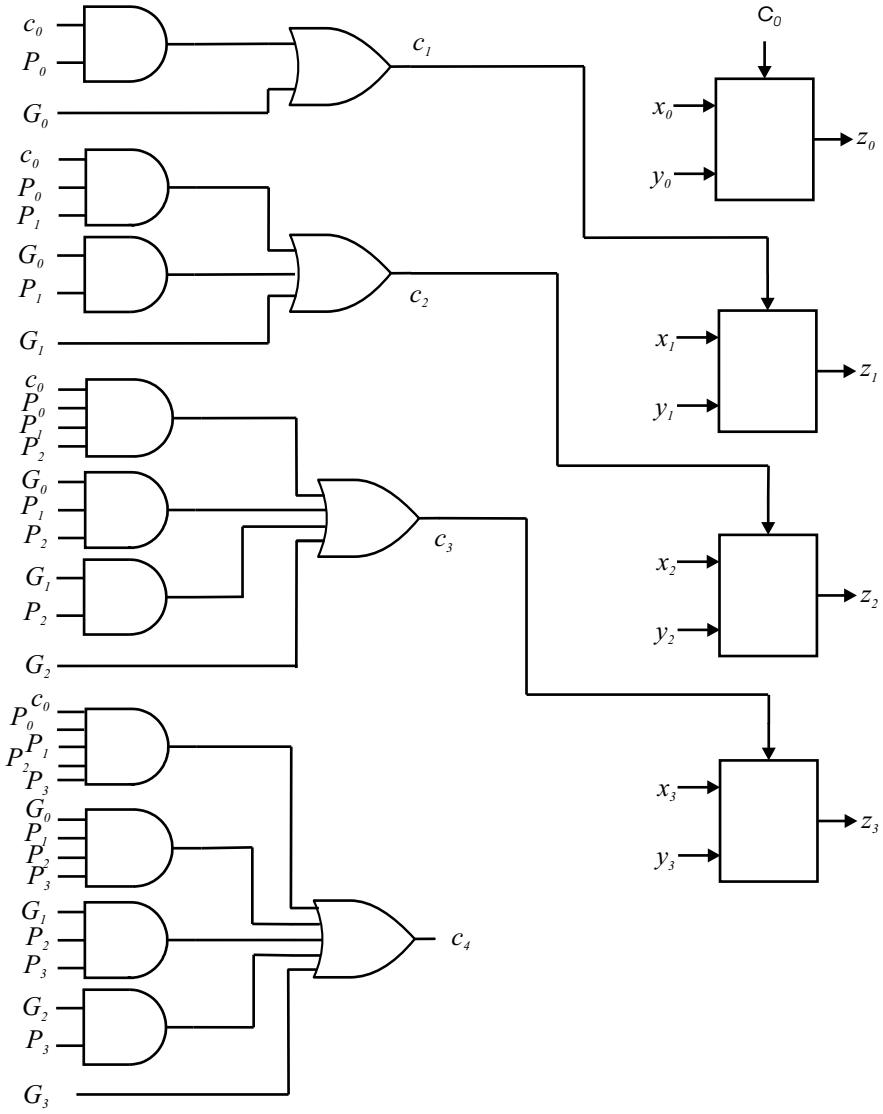


Slika 7.14: Vezje za izračun signalov G_i in P_i

Enačbe za izračun prenosa 4-bitnega seštevanja so

$$\begin{aligned} c_1 &= G_0 \vee P_0 c_0 \\ c_2 &= G_1 \vee P_1 c_1 = G_1 \vee P_1(G_0 \vee P_0 c_0) = G_1 \vee P_1 G_0 \vee P_1 P_0 c_0 \\ c_3 &= G_2 \vee P_2 c_2 = G_2 \vee P_2(G_1 \vee P_1 c_1) = G_2 \vee P_2 G_1 \vee P_2 P_1 G_0 \vee P_2 P_1 P_0 c_0 \\ c_4 &= G_3 \vee P_3 c_3 = G_2 \vee P_3(G_2 \vee P_2 c_2) \\ &= G_3 \vee P_3 G_2 \vee P_3 P_2 G_1 \vee P_3 P_2 P_1 G_0 \vee P_3 P_2 P_1 P_0 c_0 \end{aligned}$$

Slika 7.15 predstavlja izvedbo logike za izračun prenosov 4-bitnega seštevalnika:



Slika 7.15: Vezje za izračun prenosov

Aritmetična vezja v TTL tehnologiji

V TTL tehnologiji imamo na voljo naslednja standardna aritmetična vezja:

Seštevalniki
7482 - 2-bitni polni seštevalnik, ki zaporedno izvede seštevanje dveh 2-bitnih števil z vhodnim prenosom c_{in}. Na izhodu generira 2-bitno vsoto in prenos c_{out}.
7483 - 4-bitni seštevalnik z logiko za hitri izračun prenosa v čipu. Takšen 4-bitni polni seštevalnik je hitrejši od 2-bitnega.
74183 - dva ločena 1-bitna polna seštevalnika v čipu.
74283 - 4-bitni dvojiški seštevalnik z logiko za izračun prenosov.

Vezja za izračun prenosov

74182 - 4-bitno vezje za izračun prenosov ima po štiri vhode za G_i, P_i in vhod c_n, ki na izhodu daje tri vmesne prenose c_{n+z}, c_{n+y}, c_{n+x} in izhoda P, G, ki omogočata gradnjo

večnivojskih vezij za izračun prenosov.

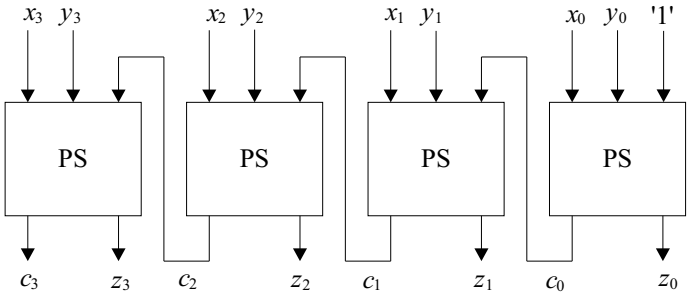
Aritmetično logična enota - ALE

Aritmetično logična enota je kombinacijsko vezje, ki izvede različne aritmetične in logične operacije dveh n- bitnih operandov. Tipične ALE enote imajo 4-bitne operande in tri do pet funkcijskih vhodov, ki omogočajo izvrševanje do 32 različnih aritmetičnih in logičnih funkcij.
74181 - 4-bitna aritmetično logična enota z dvema operandoma A in B ter vhodom c_n. Operacija je izbrana s štirimi funkcijskimi vhodi S_0, S_1, S_2, S_3. Poleg rezultatov operacij ima enota še izhode: c_{n+4} - prenos, $\bar{P}$ in $\bar{G}$ - izhoda za računanje prenosov med enotami, $A = B$ - izhod, ki pri odštevanju določa enakost operandov (rezultat odštevanja je nič).

7.5 Vaje

VAJA 7.5.1:

Realizirajte 4-bitni seštevalnik z izhodi, ki vsoti dveh 4-bitnih besed X, Y prišteje konstanto 1.
Imamo 4-bitni seštevalnik z vhodi $X = (x_3, x_2, x_1, x_0)$, $Y = (y_3, y_2, y_1, y_0)$ in izhodi $Z = (z_3, z_2, z_1, z_0)$. Na izhodih 4-bitnega seštevalnika je vsota $Z = X + Y + 1$, zato uporabimo v vezju štiri polne seštevalnike (Slika 7.16). Konstanto 1 prištejemo najmanj pomembnemu bitu, tako da jo pripeljemo na c_{in} vhod.

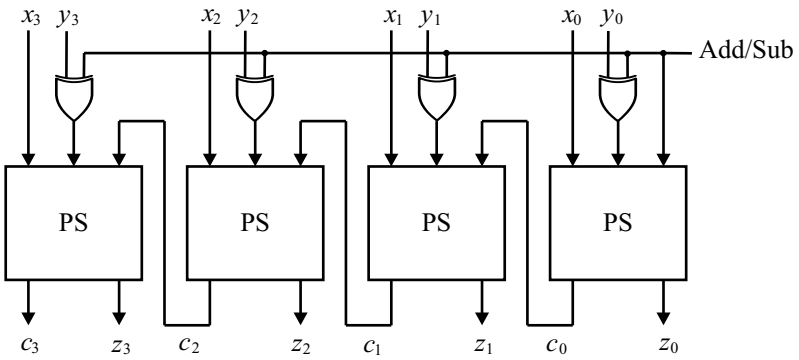


Slika 7.16: 4-bitni seštevalnik

VAJA 7.5.2:

Realizirajte logično vezje 4-bitnega seštevalnika $Z = X + Y$ in odštevalnika $Z = X - Y$ z uporabo štirih polnih seštevalnikov, tako da z zunanjim signalom Add/Sub (Seštej/Odštej) izbirate eno od aritmetičnih operacij.

Funkciji seštevanja in odštevanja izvedemo z enim samim vezjem tako, da zagotovimo na Y vhodu dvojiški komplement. Uporabimo štiri XOR logična vrata, ki izvedejo eniški komplement (Slika 7.17), če je na enem vhodu konstanta 1. Krmilni vhod Add/Sub peljemo na vhod XOR vrat in c_{in} vhod najmanj pomembnega polnega seštevalnika. Pri Add/Sub = 0 je na izhodih XOR vrat operand Y in se izvede seštevanje $Z = X + Y$, pri Add/Sub=1 se izvede negacija vhodov Y, ki skupaj s $c_{in} = 1$ zagotovi dvojiški komplement in se izvede odštevanje $Z = X + (-Y)$.



Slika 7.17: 4-bitni paralelni seštevalnik/odštevalnik

Poglavje 8

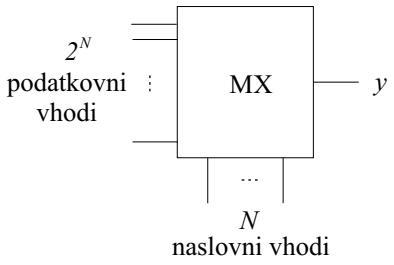
Strukturalni gradniki

Pri kompleksnih logičnih funkcijah nam število logičnih vrat (OR, AND, NAND, NOR, XOR, itd.), ki sodijo v razred operatorjev nižje stopnje integracije, v realizaciji zelo hitro narašča in vezja so zelo kompleksna. Spoznali bomo gradnike katerih funkcija je že vnaprej določena s podano strukturo, zato jih imenujemo tudi strukturalni gradniki in jih uvrščamo v razred operatorjev srednje stopnje integracije (MSI - Medium Scale Integration). Vezja, ki jih dobimo z uporabljenimi gradniki, imenujemo strukturalna logična vezja. Omogočajo nam gradnjo kompleksnejših logičnih vezij z manj čipi. Pri načrtovanju logičnih vezij uporabljamo gradnike, kot so:

- multiplekser,
- demultiplekser/dekodirnik,
- kodirnik, itd.

8.1 Multiplekser (MX)

Multiplekser je kombinacijsko vezje z N naslovnimi vhodi in 2^N podatkovnimi vhodi ter enim izhodom y (Slika 8.1). Multiplekser izbere enega od podatkovnih vhodov in ga preslika na izhod v odvisnosti od izbranega naslova. Ker gre za operacijo izbire, lahko uporabljamo tudi izraz izbiralnik ali selektor. Spoznali bomo tri osnovne multiplekserje (2/1MX, 4/1MX, 8/1MX), ki so na voljo v komercialno dostopnih čipih in jih bomo uporabljali pri gradnji logičnih vezij.



Slika 8.1: Multiplekser - MX

1-naslovni multiplekser - 2/1MX

2/1MX ima en naslovni vhod A_0 in dva podatkovna vhoda I_0, I_1. Njegovo delovanje zapišemo v pravilnostno tabelo na običajen način z dvojiškimi vrednostmi vseh vhodnih

spremenljivk A_0, I_0, I_1 (Tabela 8.1.a), ali na krajši funkcionalen način (Tabela 8.1.b).

Tabela 8.1: 1-naslovni multiplekser - 2/1MX

a)

A_0	I_0	I_1	y
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	0
1	1	1	1

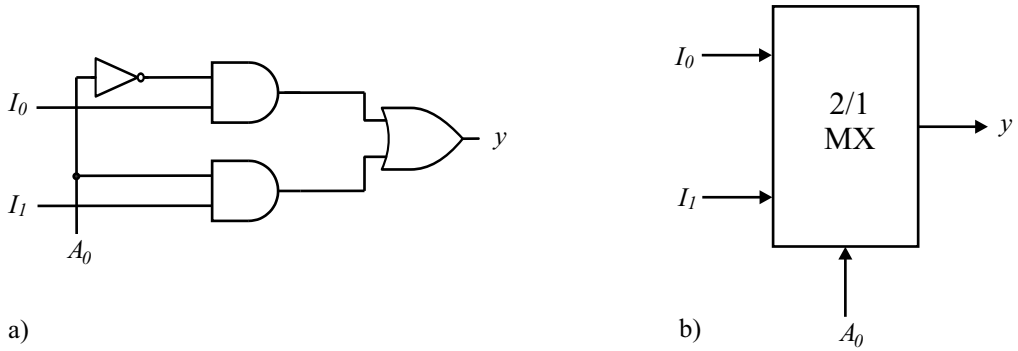
b)

A_0	y
0	I_0
1	I_1

Izhod 1-naslovnega multiplekserja (2/1MX) je podan z enačbo

$$y = \bar{A}_0 I_0 \vee A_0 I_1$$

in je pri vrednosti naslovnega vhoda $A_0 = 0$ enak podatkovnemu vhodu I_0, pri vrednosti naslovnega vhoda $A_0 = 1$ pa podatkovnemu vhodu I_1.
Multiplekser 2/1 je izveden z logičnimi vrati AND, OR, NEG (Slika 8.2.a) in ga v logičnih vezjih rišemo v obliki bloka (Slika 8.2.b).



Slika 8.2: 1-naslovni multiplekser (2/1MX): a) logična shema, b) blok shema

2-naslovni multiplekser - 4/1MX

4/1MX ima dva naslovna vhoda A_1, A_0 in štiri podatkovne vhode I_0, I_1, I_2, I_3. Njegovo delovanje zapišemo v obliki funkcijskega opisa (Tabela 8.2), kjer je podatkovni vhod izbran s pripadajočo dvojiško kodo na naslovnih vhodih.

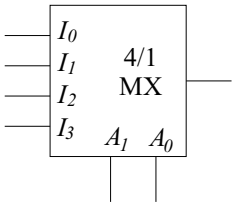
Tabela 8.2: 2-naslovni multiplekser - 4/1MX

A_1	A_0	y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Izhod 2-naslovnega multiplekserja je podan z enačbo

$$y = \bar{A}_1 \bar{A}_0 I_0 \vee \bar{A}_1 A_0 I_1 \vee A_1 \bar{A}_0 I_2 \vee A_1 A_0 I_3 \ .$$

Slika 8.3 prikazuje blok shemo 2-naslovnega multiplekserja za risanje logičnih vezij.



Slika 8.3: 2-naslovni multiplekser (4/1MX)

3-naslovni multiplekser - 8/1MX

8/1MX ima tri naslovne vhode A_2, A_1, A_0 in osem podatkovnih vhodov I_0, I_1, I_2, I_3, I_4, I_5, I_6, I_7. Njegovo delovanje zapišemo v obliki funkcijskega zapisa (Tabela 8.3), kjer je podatkovni vhod izbran s pripadajočo dvojiško kodo na naslovnih vhodih.

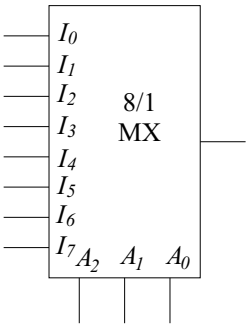
Tabela 8.3: 8/1 MX

A_2	A_1	A_0	y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

Izhod 3-naslovnega multiplekserja je podan z enačbo

$$\begin{aligned} y &= \bar{A}_2\bar{A}_1\bar{A}_0I_0 \vee \bar{A}_2\bar{A}_1A_0I_1 \vee \bar{A}_2A_1\bar{A}_0I_2 \vee \bar{A}_2A_1A_0I_3 \\ &\vee \quad A_2\bar{A}_1\bar{A}_0I_4 \vee A_2\bar{A}_1A_0I_5 \vee A_2A_1\bar{A}_0I_6 \vee A_2A_1A_0I_7 \ . \end{aligned}$$

Slika 8.4 prikazuje blok shemo 3-naslovnega multiplekserja za risanje logičnih vezij.



Slika 8.4: 3-naslovni multiplekser (8/1MX)

Realizacija logičnih funkcij z multiplekserji

Logična funkcija, ki jo želimo realizirati z multiplekserjem je podana z množico spremenljivk $X = (x_1, x_2, ..., x_n)$ in izhodom $f(X)$. Ker imamo pri multiplekserjih podatkovne

in naslovne vhode, bomo za realizacijo logičnih funkcij množico spremenljivk razdelili v dva dela: $X = X_a \bigcup X_d$. V množico naslovnih spremenljivk X_a bomo iz funkcije izbrali toliko spremenljivk, kot imamo naslovnih vhodov v izbranem multiplekserju. Vse preostale spremenljivke v logični funkciji postanejo podatkovne in jih razvrstimo v množico podatkovnih spremenljivk X_d.

Naslovne spremenljivke - X_a - število naslovnih spremenljivk X_a je določeno z izbranim multiplekserjem :

- 2/1MX ima en naslovni vhod A_0, zato bo ena spremenljivka logične funkcije naslovna,
- 4/1MX ima dva naslovna vhoda, kjer je A_1 bolj pomemben bit naslova (MSB) in A_0 je manj pomemben bit naslova (LSB), zato bosta dve spremenljivki logične funkcije naslovni,
- 8/1MX ima tri naslovne vhode kjer je A_2 najbolj pomemben bit naslova (MSB- Most Significant Bit), A_0 pa je najmanj pomemben bit naslova (LSB-Least Significant Bit), zato bodo tri spremenljivke naslovne.

Podatkovne spremenljivke - X_d - so vse tiste spremenljivke iz množice X, ki niso izbrane kot naslovne ($X_d = X - X_a$).

Realizacija logičnih funkcij z multiplekserji poteka tako, da za izbrane naslovne spremenljivke $X_a = N$, izračunamo podatkovne vhode I_0 do $2^N - 1$ z uporabo postopka razčlenjevanja logičnih funkcij.

Razčlenjevanje logičnih funkcij v normalni obliki

Razčlenitev logične funkcije po eni spremenljivki - x_i pomeni razdelitev funkcije v dva dela tako, da jo opazujemo pri vrednosti $x_i = 0$ in $x_i = 1$ v odvisnosti od preostalih $n - 1$ spremenljivk. Pri tem dobimo v zapisu logične funkcije dva funkcijska ostanka $F_0 = f(x_1, ..., 0, ..., x_n)$ in $F_1 = f(x_1, ..., 1, ..., x_n)$.

$$\begin{aligned} f(x_1, ..., x_n) &= \bar{x}_i f(x_1, ..., 0, ..., x_n) \vee x_i f(x_1, ..., 1, ..., x_n) \\ &= \bar{x}_i F_0 \vee x_i F_1 \end{aligned}$$

Primer: Logično funkcijo $f(x_1, x_2, x_3) = \bar{x}_1 \bar{x}_2 x_3 \vee x_2 x_3$ razčlenimo po eni spremenljivki. Razčlenitev logične funkcije po spremenljivki x_1:

$$\begin{aligned} f(x_1, x_2, x_3) &= \bar{x}_1 \bar{x}_2 x_3 \vee x_2 x_3 \\ &= \bar{x}_1 (1 \bar{x}_2 x_3 \vee x_2 x_3) \vee x_1 (0 \bar{x}_2 x_3 \vee x_2 x_3) \\ &= \bar{x}_1 (x_3 (\bar{x}_2 \vee x_2)) \vee x_1 (x_2 x_3) \\ &= \bar{x}_1 (x_3) \vee x_1 (x_2 x_3) \end{aligned}$$

Razčlenitev logične funkcije po spremenljivki x_2:

$$\begin{aligned} f(x_1, x_2, x_3) &= \bar{x}_1 \bar{x}_2 x_3 \vee x_2 x_3 \\ &= \bar{x}_2 (\bar{x}_1 1 x_3 \vee 0 x_3) \vee x_2 (\bar{x}_1 0 x_3 \vee 1 x_3) \\ &= \bar{x}_2 (\bar{x}_1 x_3) \vee x_2 (x_3) \end{aligned}$$

Razčlenitev logične funkcije po spremenljivki x_3:

$$\begin{aligned} f(x_1,x_2,x_3) &= \bar{x}_1\bar{x}_2x_3 \vee x_2x_3 \\ &= \bar{x}_3(\bar{x}_1\bar{x}_20 \vee x_20) \vee x_3(\bar{x}_1\bar{x}_21 \vee x_21) \\ &= \bar{x}_3(0) \vee x_3(\bar{x}_1 \vee x_2) \end{aligned}$$

Konjunktivni/disjunktivni izrazi, ali spremenljivke x_i, $\bar{x}_i$, ali konstanti (0,1) so funkcijski ostanki razčlenjene logične funkcije, ki jih bomo uporabili pri realizaciji logičnih funkcij z 1-naslovnim multiplekserjem.

Razčlenitev logične funkcije po dveh spremenljivkah - x_ix_j pomeni razdelitev funkcije v štiri dele tako, da jo opazujemo pri vrednostih $x_ix_j = 00$, $x_ix_j = 01$, $x_ix_j = 10$ in $x_ix_j = 11$ v odvisnosti od preostalih $n-2$ spremenljivk. V zapisu logične funkcije dobimo štiri funkcijske ostanke $F_0 = f(x_1,...,0,0,...,x_n)$, $F_1 = f(x_1,...,0,1,...,x_n)$, $F_2 = f(x_1,...,1,0,...,x_n)$ in $F_3 = f(x_1,...,1,1,...,x_n)$.

$$\begin{aligned} f(x_1,...,x_n) &= \bar{x}_i\bar{x}_jf(x_1,...,0,0,...,x_n) \vee \bar{x}_ix_jf(x_1,...,0,1,...,x_n) \vee \\ &\vee \quad x_i\bar{x}_jf(x_1,...,1,0,...,x_n) \vee x_ix_jf(x_1,...,1,1,...,x_n) \\ &= \bar{x}_i\bar{x}_jF_0 \vee \bar{x}_ix_jF_1 \vee x_i\bar{x}_jF_2 \vee x_ix_jF_3 \end{aligned}$$

Primer: Logično funkcijo $f(x_1,x_2,x_3) = \bar{x}_1\bar{x}_2x_3 \vee x_2x_3$ razčlenimo po dveh spremenljivkah.
Razčlenitev logične funkcije po spremenljivkah x_1x_2:

$$\begin{aligned} f(x_1,x_2,x_3) &= \bar{x}_1\bar{x}_2x_3 \vee x_2x_3 \\ &= \bar{x}_1\bar{x}_2(1.1.x_3 \vee 0x_3) \vee \bar{x}_1x_2(1.0.x_3 \vee 1.x_3) \vee \\ &\vee \quad x_1\bar{x}_2(0.1.x_3 \vee 0.x_3) \vee x_1x_2(0.0.x_3 \vee 1.x_3) \\ &= \bar{x}_1\bar{x}_2(x_3) \vee \bar{x}_1x_2(x_3) \vee x_1\bar{x}_2(0) \vee x_1x_2(x_3) \end{aligned}$$

Razčlenitev logične funkcije po spremenljivkah x_1x_3:

$$\begin{aligned} f(x_1,x_2,x_3) &= \bar{x}_1\bar{x}_2x_3 \vee x_2x_3 \\ &= \bar{x}_1\bar{x}_3(1\bar{x}_20 \vee x_20) \vee \bar{x}_1x_3(1\bar{x}_21 \vee x_21) \vee \\ &\vee \quad x_1\bar{x}_3(0\bar{x}_20 \vee x_20) \vee x_1x_3(0\bar{x}_21 \vee x_21) \\ &= \bar{x}_1\bar{x}_3(0) \vee \bar{x}_1x_3(1) \vee x_1\bar{x}_3(0) \vee x_1x_3(x_2) \end{aligned}$$

Razčlenitev logične funkcije po spremenljivkah x_2x_3:

$$\begin{aligned} f(x_1,x_2,x_3) &= \bar{x}_1\bar{x}_2x_3 \vee x_2x_3 \\ &= \bar{x}_2\bar{x}_3(\bar{x}_110 \vee 00) \vee \bar{x}_2x_3(\bar{x}_111 \vee 01) \vee \\ &\vee \quad x_2\bar{x}_3(\bar{x}_100 \vee 10) \vee x_2x_3(\bar{x}_101 \vee 11) \\ &= \bar{x}_2\bar{x}_3(0) \vee \bar{x}_2x_3(\bar{x}_1) \vee x_2\bar{x}_3(0) \vee x_2x_3(1) \end{aligned}$$

Spremenljivke, ali konstanti (0,1) so funkcijski ostanki razčlenjene logične funkcije, ki jih bomo uporabili pri realizaciji logičnih funkcij z 2-naslovnim multiplekserjem.

Razčlenitev logične funkcije po treh spremenljivkah $x_ix_jx_k$ pomeni razdelitev funkcije v osem delov tako, da jo opazujemo pri vrednostih $x_ix_jx_k = 000$, $x_ix_jx_k = 001$,..., $x_ix_jx_k = 111$ v odvisnosti od preostalih $n-3$ spremenljivk. Pri tem dobimo

v zapisu logične funkcije osem funkcijskih ostankov $F_0 = f(x_1, ..., 0, 0, 0, ..., x_n)$, $F_1 = f(x_1, ..., 0, 0, 1, ..., x_n)$, ..., $F_7 = f(x_1, ..., 1, 1, 1, ..., x_n)$.

$$\begin{aligned} f(x_1, ..., x_n) &= \bar{x}_i\bar{x}_j\bar{x}_k f(x_1, ..., 0, 0, 0, ..., x_n) \vee \bar{x}_i\bar{x}_j x_k f(x_1, ..., 0, 0, 1, ..., x_n) \vee \\ &\vee \bar{x}_i x_j\bar{x}_k f(x_1, ..., 0, 1, 0, ..., x_n) \vee \bar{x}_i x_j x_k f(x_1, ..., 0, 1, 1, ..., x_n) \vee \\ &\vee x_i\bar{x}_j\bar{x}_k f(x_1, ..., 1, 0, 0, ..., x_n) \vee x_i\bar{x}_j x_k f(x_1, ..., 1, 0, 1, ..., x_n) \vee \\ &\vee x_i x_j\bar{x}_k f(x_1, ..., 1, 1, 0, ..., x_n) \vee x_i x_j x_k f(x_1, ..., 1, 1, 1, ..., x_n) \\ &= \bar{x}_i\bar{x}_j\bar{x}_k F_0 \vee \bar{x}_i\bar{x}_j x_k F_1 \vee \bar{x}_i x_j\bar{x}_k F_2 \vee \bar{x}_i x_j x_k F_3 \vee \\ &\vee x_i\bar{x}_j\bar{x}_k F_4 \vee x_i\bar{x}_j x_k F_5 \vee x_i x_j\bar{x}_k F_6 \vee x_i x_j x_k F_7 \end{aligned}$$

Primer: Logično funkcijo $f(x_1, x_2, x_3) = \bar{x}_1\bar{x}_2 x_3 \vee x_2 x_3$ razčlenimo po treh spremenljivkah $(x_1 x_2 x_3)$. Razčlenitev logične funkcije je enaka popolni disjunktivni normalni obliki. Namesto da bi za vsako kombinacijo vstavljali v enačbo konstanti 0 in 1, poiščemo minterme pri katerih ima funkcija vrednost 1 in jih vpišimo v razčlenjeno obliko logične funkcije. Z uporabo pstulatov in izrekov zapišimo logično funkcijo v PDNO.

$$\begin{aligned} f(x_1, x_2, x_3) &= \bar{x}_1\bar{x}_2 x_3 \vee (\bar{x}_1 \vee x_1) x_2 x_3 \\ &= \bar{x}_1\bar{x}_2 x_3 \vee \bar{x}_1 x_2 x_3 \vee x_1 x_2 x_3 \end{aligned}$$

Iz zapisa logične funkcije v PDNO vidimo, da imajo mintermi m_1, m_3 in m_7 funkcijsko vrednost 1, ki jo vpišemo pripadajočim funkcijskim ostankom v razčlenjeno obliko. Ostalih pet mintermov ima funkcijske vrednosti 0.

$$\begin{aligned} f(x_1, x_2, x_3) &= \bar{x}_1\bar{x}_2\bar{x}_3(0) \vee \bar{x}_1\bar{x}_2 x_3(1) \vee \bar{x}_1 x_2\bar{x}_3(0) \vee \bar{x}_1 x_2 x_3(1) \\ &\vee x_1\bar{x}_2\bar{x}_3(0) \vee x_1\bar{x}_2 x_3(0) \vee x_1 x_2\bar{x}_3(0) \vee x_1 x_2 x_3(1) \end{aligned}$$

Razčlenjevanje logične funkcije v pravilnostni tabeli

Razčlenitev logične funkcije po eni spremenljivki - x_i pomeni razdelitev funkcije v tabeli na dva dela tako, da jo opazujemo pri vrednosti $x_i = 0$ in $x_i = 1$ v odvisnosti od preostalih $n - 1$ spremenljivk in jo zapišemo v enačbo razčlenitve funkcije.

Primer: Logično funkcijo $f(x_1, x_2, x_3) = \bar{x}_1\bar{x}_2 x_3 \vee x_2 x_3$ razčlenimo po spremenljivki x_1 (Tabela 8.4):

Tabela 8.4: Razčlenitev po x_1

$x_1 x_2 x_3$	y	Funkcijski ostanki
0 0 0	0	
0 0 1	1	
0 1 0	0	
0 1 1	1	$F_0 = x_3$
1 0 0	0	
1 0 1	0	
1 1 0	0	
1 1 1	1	$F_1 = x_2 x_3$

Pravilnostna tabela je razdeljena v dva dela in v zgornji polovici pri spremenljivki $x_1 = 0$ je funkcijski ostanek F_0 odvisen od spremenljivke x_3. Spodnja polovica tabele pri $x_1 = 1$ pa ima funkcijski ostanek F_1, ki je enak konjunkciji spremenljivk x_2 in x_3.

$$\begin{aligned} f(x_1, x_2, x_3) &= \bar{x}_1(F_0) \vee x_1(F_1) \\ &= \bar{x}_1(x_3) \vee x_1(x_2 x_3) \end{aligned}$$

Razčlenitev logične funkcije po dveh spremenljivkah - x_ix_j pomeni razdelitev funkcije v tabeli na štiri dele tako, da jo opazujemo pri vrednostih $x_ix_j = 00$, $x_ix_j = 01$, $x_ix_j = 10$ in $x_ix_j = 11$ v odvisnosti od preostalih $n-2$ spremenljivk in jo zapišemo v enačbo razčlenitve funkcije.

Primer: Logično funkcijo $f(x_1, x_2, x_3) = \bar{x}_1\bar{x}_2x_3 \vee x_2x_3$ razčlenimo po spremenljivkah x_1x_2 (Tabela 8.5).

Tabela 8.5: Razčlenitev po x_1x_2

$x_1x_2x_3$	y	Funkcijski ostanki
0 0 0	0	
0 0 1	1	$F_0 = x_3$
0 1 0	0	
0 1 1	1	$F_1 = x_3$
1 0 0	0	
1 0 1	0	$F_2 = 0$
1 1 0	0	
1 1 1	1	$F_3 = x_3$

Pravilnostna tabela je razdeljena v štiri dele, kjer imamo vpisane pripadajoče funkcijske ostanke, ki jih uporabimo za zapis logične funkcije v razčlenjeni obliki. V funkcijskih ostankih se lahko pojavi spremenljivka x_3, $\bar{x}_3$, ali konstanti 0 in 1.

$$\begin{aligned} f(x_1, x_2, x_3) &= \bar{x}_1\bar{x}_2(F_0) \vee \bar{x}_1x_2(F_1) \vee x_1\bar{x}_2(F_2) \vee x_1x_2(F_3) \\ &= \bar{x}_1\bar{x}_2(x_3) \vee \bar{x}_1x_2(x_3) \vee x_1\bar{x}_2(0) \vee x_1x_2(x_3) \end{aligned}$$

Razčlenitev logične funkcije po treh spremenljivkah $x_ix_jx_k$ pomeni razdelitev funkcije v osem funkcijskih ostankov tako, da jo opazujemo pri vrednostih $x_ix_jx_k = 000$, $x_ix_jx_k = 001$, ..., $x_ix_jx_k = 111$ v odvisnosti od preostalih $n-3$ spremenljivk.

Primer: Razčlenitev logične funkcije po treh spremenljivkah $x_1x_2x_3$ je zapis v popolni disjunktivni normalni obliki, kjer so funkcijski ostanki enaki funkcijskim vrednostim (Tabela 8.6).

Tabela 8.6: Razčlenitev po $x_1x_2x_3$

$x_1x_2x_3$	y	Funkcijski ostanki
0 0 0	0	$F_0 = f_0 = 0$
0 0 1	1	$F_1 = f_1 = 1$
0 1 0	0	$F_2 = f_2 = 0$
0 1 1	1	$F_3 = f_3 = 1$
1 0 0	0	$F_4 = f_4 = 0$
1 0 1	0	$F_5 = f_5 = 0$
1 1 0	0	$F_6 = f_6 = 0$
1 1 1	1	$F_7 = f_7 = 1$

$$\begin{aligned} f(x_1, x_2, x_3) &= \bar{x}_1\bar{x}_2\bar{x}_3(0) \vee \bar{x}_1\bar{x}_2x_3(1) \vee \bar{x}_1x_2\bar{x}_3(0) \vee \bar{x}_1x_2x_3(1) \\ \vee\ & x_1\bar{x}_2\bar{x}_3(0) \vee x_1\bar{x}_2x_3(0) \vee x_1x_2\bar{x}_3(0) \vee x_1x_2x_3(1) \end{aligned}$$

Realizacija logičnih funkcij z multiplekserji

Pri realizaciji logičnih funkcij z multiplekserji imamo več možnosti. Enostavne logične funkcije z majhnim številom vhodnih spremenljivk pogosto lahko realiziramo samo z enim multiplekserjem in v ta namen vpeljemo trivialno oziroma minimalno rešitev. Za kompleksne logične funkcije z večjim številom vhodnih spremenljivk uporabimo večnivojsko povezavo multiplekserjev, ki jo imenujemo kaskadna rešitev.

1. Trivialna rešitev

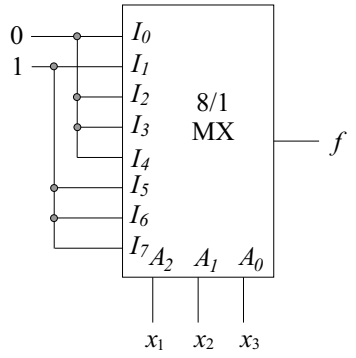
Za funkcijo z n spremenljivkami vzamemo $N = n$ naslovni multiplekser. Ker so vse spremenljivke logične funkcije pripeljane na naslovne vhode multiplekserja, imamo na podatkovnih vhodih konstanti 0 ali 1. Pri uporabi trivialne rešitve imamo naslednje možnosti realizacij ob uporabi standardnih komercialno dostopnih čipov:
- $n = 1 \rightarrow$ 1-naslovni MX (2/1MX)
- $n = 2 \rightarrow$ 2-naslovni MX (4/1MX)
- $n = 3 \rightarrow$ 3-naslovni MX (8/1MX)

Primer: Logično funkcijo $f^3 = \vee(1, 5, 6, 7)$ realiziramo s 3-naslovnim multiplekserjem.

Vhodne spremenljivke x_1, x_2, x_3 so pripeljane na naslovne vhode multiplekserja: $A_2 = x_1$, $A_1 = x_2$, $A_0 = x_3$. Če logično funkcijo razčlenimo po treh spremenljivkah dobimo funkcijske ostanke, ki so enaki funkcijskim vrednostim v pravilnostni tabeli in ustrezajo podatkovnim vhodom multiplekserja I_0, I_1, ..., I_7 (Tabela 8.7). Izvedba logične funkcije ima na podatkovnih vhodih konstanti 0 ali 1 (Slika 8.5).

Tabela 8.7: Razčlenitev funkcije f po $x_1x_2x_3$

$x_1x_2x_3$	f	Podatkovni vhodi
0 0 0	0	I_0
0 0 1	1	I_1
0 1 0	0	I_2
0 1 1	0	I_3
1 0 0	0	I_4
1 0 1	1	I_5
1 1 0	1	I_6
1 1 1	1	I_7



Slika 8.5: Trivialna rešitev - 8/1MX

2. Minimalna rešitev

Minimalna rešitev pomeni realizacijo logične funkcije z enim multiplekserjem, tako da pri n vhodnih spremenljivkah uporabimo $N = n - i$, $i = 1, 2, 3, ...$ naslovni multiplekser. Oglejmo si dve možnosti realizacij:

a) Za funkcijo z $n \leq 4$ vhodnimi spremenljivkami vzamemo $N = n - 1$ naslovni multiplekser. Ker je $n - 1$ spremenljivk logične funkcije pripeljanih na naslovne vhode multiplekserja, imamo na podatkovnih vhodih konstanti 0 ali 1 in eno vhodno spremenljivko kot x_i ali $\bar{x}_i$. Pri uporabi takšne minimalne rešitve imamo naslednje možnosti realizacij ob uporabi standardnih komercialno dostopnih čipov:
- $n = 2 \rightarrow$ 1-naslovni MX (2/1MX)
- $n = 3 \rightarrow$ 2-naslovni MX (4/1MX)
- $n = 4 \rightarrow$ 3-naslovni MX (8/1MX)

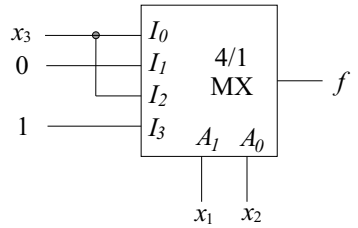
Primer: Logično funkcijo $f^3 = \vee(1, 5, 6, 7)$ realiziramo z 2-naslovnim multiplekserjem.

Naslovni spremenljivki x_1, x_2 sta pripeljani na naslovna vhoda: $A_1 = x_1, A_0 = x_2$, zato funkcijo v pravilnostni tabeli razčlenimo po dveh spremenljivkah in dobimo funkcijske ostanke, ki ustrezajo podatkovnim vhodom I_0, I_1, I_2, I_3 (Tabela 8.8).

Tabela 8.8: Razčlenitev funkcije po x_1x_2

$x_1x_2x_3$	f	Podatkovni vhodi
0 0 0	0	
0 0 1	1	$I_0 = x_3$
0 1 0	0	
0 1 1	0	$I_1 = 0$
1 0 0	0	
1 0 1	1	$I_2 = x_3$
1 1 0	1	
1 1 1	1	$I_3 = 1$

Izvedba logične funkcije $f(x_1, x_2, x_3)$ z 2-naslovnim multiplekserjem ima na naslovnih vhodih spremenljivki x_1, x_2 in na podatkovnih vhodih x_3 in konstanti 0 in 1. (Slika 8.6).



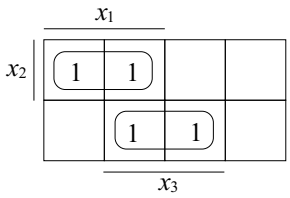
Slika 8.6: Minimalna rešitev a) - 4/1MX

b) Za logično funkcijo z n spremenljivkami smo v minimalni disjunktivni normalni obliki (MDNO) ugotovili, da je odvisna od $n - i$, $i = 2, 3, ...$ spremenljivk tako, da jo je

možno realizirati samo z enim multiplekserjem. Na podatkovnih vhodih imamo konstanti 0 ali 1 in eno spremenljivko kot x_i ali $\bar{x}_i$. Pri uporabi takšne minimalne rešitve imamo naslednje možnosti realizacij ob uporabi standardnih komercialno dostopnih čipov:
$n = 3, 4, ... \rightarrow$ 1-naslovni MX (2/1MX)
$n = 4, 5, ... \rightarrow$ 2-naslovni MX (4/1MX)
$n = 5, 6, ... \rightarrow$ 3-naslovni MX (8/1MX)

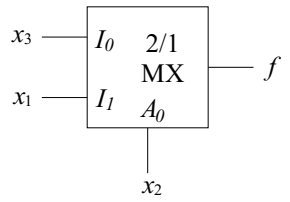
Primer: Logično funkcijo $f^3 = \vee(1, 5, 6, 7)$ realizirajmo z 1-naslovnim multiplekserjem.

Poiščimo minimalno disjunktivno normalno obliko (MDNO) in poglejmo ali jo je možno realizirati z 1-naslovnim multiplekserjem.



$$\begin{aligned} f(x_1, x_2, x_3) &= x_1x_2 \vee \bar{x}_2x_3 \\ &= \bar{x}_2(x_3) \vee x_2(x_1) \end{aligned}$$

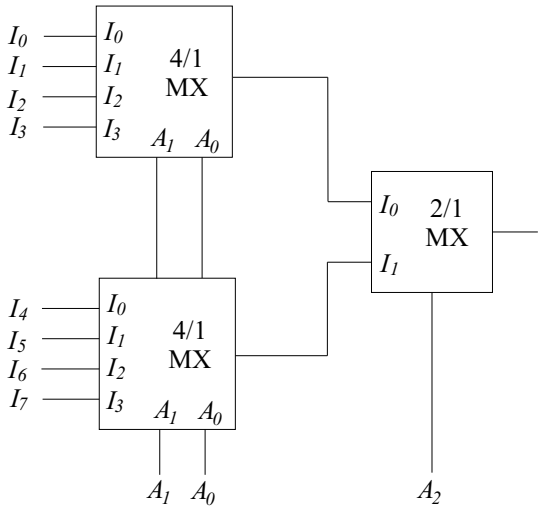
Minimalna oblika logične funkcije je enaka razčlenitvi logične funkcije po spremenljivki x_2, kjer sta oba funkcijska ostanka enaka eni sami spremenljivki, kar nam omogoča realizacijo logične funkcije z 1-naslovnim multiplekserjem. Na naslovni vhod A_0 pride spremenljivka x_2, na podatkovnem vhodu I_0 dobimo spremenljivko x_3 in na I_1 spremenljivko x_1 (Slika 8.7).



Slika 8.7: Minimalna rešitev b) - 2/1MX

3. Kaskadna rešitev

V primeru velikega števila spremenljivk ($n > N$) je potrebno za realizacijo logične funkcije več multiplekserjev razvrstiti v nivoje. Ker smo pri enem čipu omejeni z velikostjo multiplekserja, lahko vedno večnaslovni multiplekser $2^N/1$ zgradimo z več manj naslovnih kaskadno vezanih multiplekserjev. Oglejmo si 3-naslovni multiplekser zgrajen iz 1-naslovnega in dveh 2-naslovnih multiplekserjev (Slika 8.8).



Slika 8.8: 3-naslovni multiplekser - 8/1MX

Primer: Realizirajmo logično funkcijo $f(x_1, x_2, x_3, x_4, x_5, x_6) = x_1x_2x_3 \vee \bar{x}_1x_2\bar{x}_4\bar{x}_6 \vee x_1\bar{x}_2x_4x_5x_6$ z uporabo 2-naslovnih multiplekserjev.

Izberemo dve naslovni spremenljivki $A_1 = x_1, A_0 = x_2$, razčlenimo logično funkcijo in zapišemo podatkovne vhode I_0, I_1, I_2, I_3.

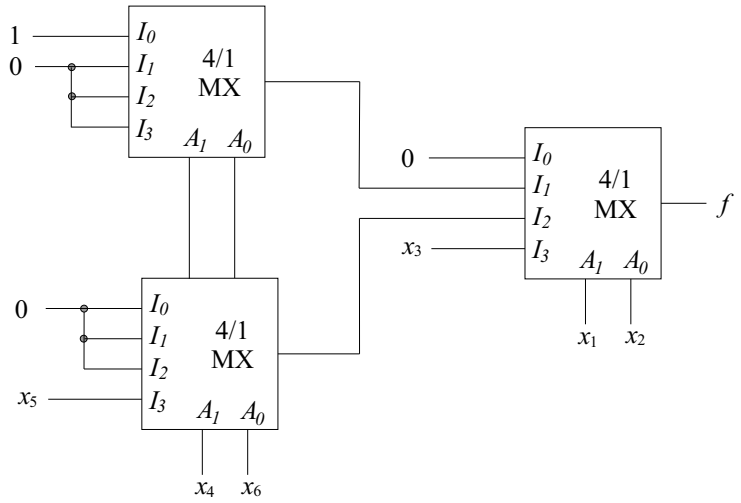
$$\begin{aligned} I_0 &= f(0,0,x_3,x_4,x_5,x_6) = 0 \\ I_1 &= f(0,1,x_3,x_4,x_5,x_6) = \bar{x}_4\bar{x}_6 = g(x_4,x_6) \\ I_2 &= f(1,0,x_3,x_4,x_5,x_6) = x_4x_5x_6 = h(x_4,x_5,x_6) \\ I_3 &= f(1,1,x_3,x_4,x_5,x_6) = x_3 \end{aligned}$$

Pri razčlenjevanju logičnih funkcij po spremenljivkah x_1 in x_2 smo dobili na podatkovnih vhodih I_1 in I_2 funkcijska ostanka $g(x_4, x_5)$ in $h(x_4, x_5, x_6)$ odvisna od dveh in treh spremenljivk. Za realizacijo teh dveh funkcij vpeljemo nov nivo multiplekserjev, katerih podatkovne vhode izračunamo z razčlenitvijo funkcij g in h po naslovnih spremenljivkah $A_1 = x_4, A_0 = x_6$ in dobimo

$$\begin{aligned} g(x_4,x_6) &= \bar{x}_4\bar{x}_6 \\ I_0 &= g(0,0) = 1 \\ I_1 &= g(0,1) = 0 \\ I_2 &= g(1,0) = 0 \\ I_3 &= g(1,1) = 0 \end{aligned}$$

$$\begin{aligned} h(x_4,x_5,x_6) &= x_4x_5x_6 \\ I_0 &= h(0,0,x_5) = 0 \\ I_1 &= h(0,1,x_5) = 0 \\ I_2 &= h(1,0,x_5) = 0 \\ I_3 &= h(1,1,x_5) = x_5 \end{aligned}$$

Po drugem koraku reševanja smo dobili na podatkovnih vhodih samo eno spremenljivko, ali pa konstanti 0 in 1, kar pomeni da imamo končno rešitev v dvonivojski izvedbi z multiplekserji (Slika 8.9).



Slika 8.9: Kaskadna rešitev - 4/1MX

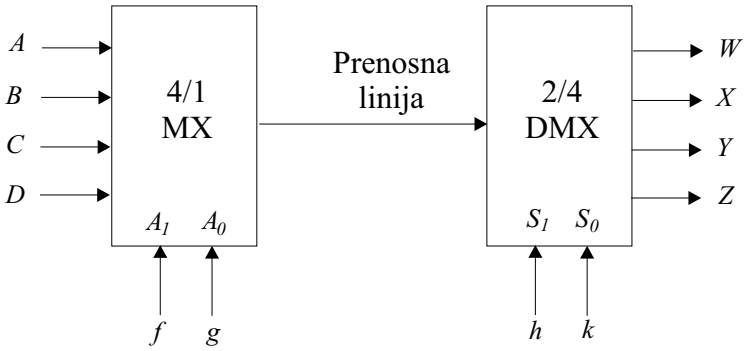
TTL multiplekserji

V TTL tehnologiji imamo naslednje multiplekserje:

74157 - (2/1MX) - vsebuje štiri 2/1 multiplekserje s podatkovnimi vhodi in istim naslovnim vhodom $A_0 = S$. V čipu je še en dodaten vhod za omogočanje izhodov označen z E. Z logično vrednostjo 0 multiplekser na izhodu prepušča izbran podatkovni vhod.
74153 - (4/1MX) - vsebuje dva 4/1 multiplekserja s podatkovnimi vhodi in istima naslovnima vhodoma $A_1 = S_1$, $A_0 = S_0$. V čipu sta še dva dodatna vhoda za omogočanje izhodov vsakega multiplekserja posebej označena z E_a in E_b. Z logično vrednostjo 0 na enable vhodih multiplekser na izhodu prepušča izbran podatkovni vhod.
74151 - (8/1MX) - vsebuje en 8/1 multiplekser s podatkvnimi vhodi in naslovnimi vhodi $A_2 = S_2$, $A_1 = S_1$, $A_0 = S_0$. V čipu je še en dodaten vhod za omogočanje izhodov označen z E. Z logično vrednostjo 0 multiplekser na izhodu prepušča izbran podatkovni vhod.

8.2 Demultiplekser/Dekodirnik

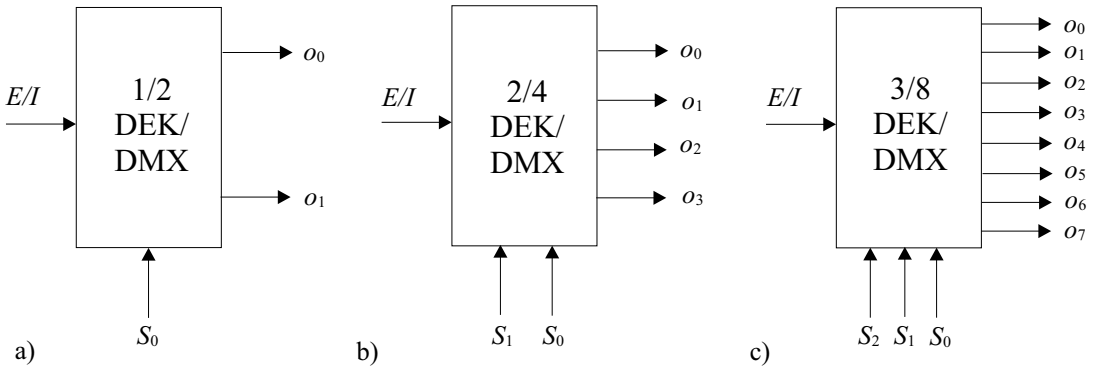
Demultiplekser (DMX) deluje ravno obratno kot multiplekser. Glede na stanje izbirnih ali krmilnih vhodov se signal na vhodu prenese na enega izmed izhodov. Uporabimo ga v enostavnem prenosnem sistemu, kjer je lahko poljuben izvor povezan s poljubnim ponorom preko ene prenosne linije v odvisnosti od izbirnih vhodov (Slika 8.10).



Slika 8.10: Prenos podatkov

Z izvornima signaloma f, g izberemo eno od linij A, B, C, D in preko prenosne linije posredujemo izvorni signal na eno od ponornih linij W, X, Y, Z v odvisnosti od ponornih signalov h, k.

Demultiplekser ima 2^N izhodov pri N krmilnih ali izbirnih vhodih ter en sam podatkovni vhod I. V komercialno dostopnih integriranih elementih imamo na voljo 1/2DMX, 2/4DMX in 3/8DMX (Slika 8.11), kjer je v oznaki podano število krmilnih vhodov/število izhodov ($N/2^N$). Po potrebi lahko iz njih zgradimo demultiplekser poljubne velikosti.



Slika 8.11: Dekodirnik/Demultiplekser

Dekodirnik je pravzaprav demultiplekser, kjer ima podatkovni vhod E (ang. E-enable) funkcijo omogočanja izhoda. Vhod $E = 1$ pomeni, da je dekodirnik aktiven (izbran izhod ima vrednost 1), $E = 0$ pa pomeni, da ni aktiven (vsi izhodi imajo vrednost 0). Dekodirnik ima 2^N izhodov pri N krmilnih ali izbirnih vhodih ter vhod enable E. V integriranih elementih imamo na voljo tri dekodirnike: 1/2 DEK, 2/4 DEK, 3/8 DEK (Slika 8.11). Uporabljajo se za pretvorbo dvojiške kode v osmiško, desetiško (BCD), šestnajstiško, redkeje pa za realizacijo logičnih funkcij. S kaskadno povezavo manjših dekodirnikov lahko zgradimo kompleksnejše dekodirnike.

Izhodne funkcije dekodirnikov/demultiplekserjev o_i, $i = 0, ..., 2^N - 1$ zapišemo v pravilnostno tabelo v odvisnosti od krmilnih vhodov s_k, $k = 0, ..., N - 1$ in vhoda enable E oziroma vhoda I ter jih podamo v obliki logičnih funkcij.

- **1/2 Dekodirnik/Demultiplekser** - ima en krmilni vhod s_0 z dvema izhodoma o_0 in o_1.

Tabela 8.9: 1/2DEK/DMX

E/I	s_0	o_0	o_1
0	0	0	0
0	1	0	0
1	0	1	0
1	1	0	1

Izhodni funkciji 1/2 dekodirnika (Tabela 8.9) sta: $o_0 = E\bar{s}_0$ in $o_1 = Es_0$,
1/2 demultiplekserja pa sta: $o_0 = I\bar{s}_0$ in $o_1 = Is_0$.

- **2/4 Dekodirnik/Demultiplekser** - ima dva krmilna vhoda s_1 in s_0 s štirimi izhodi o_0, o_1, o_2, o_3.

Tabela 8.10: 2/4DEK/DMX

E/I	s_1	s_0	o_0	o_1	o_2	o_3
0	×	×	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

Izhodne funkcije 2/4 dekodirnika (Tabela 8.10) so:
$o_0 = E\bar{s}_1\bar{s}_0$, $o_1 = E\bar{s}_1 s_0$, $o_2 = Es_1\bar{s}_0$ in $o_3 = Es_1 s_0$,
2/4 demultiplekserja pa so:
$o_0 = I\bar{s}_1\bar{s}_0$, $o_1 = I\bar{s}_1 s_0$, $o_2 = Is_1\bar{s}_0$ in $o_3 = Is_1 s_0$.

- **3/8 Dekodirnik/Demultiplekser** - ima tri krmilne vhode s_2, s_1 in s_0 z osmimi izhodi o_0, o_1, o_2, o_3, o_4, o_5, o_6, o_7.

Tabela 8.11: 3/8DEK/DMX

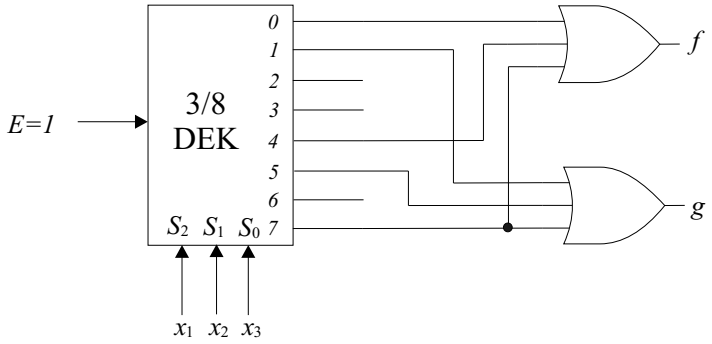
E/I	s_2	s_1	s_0	o_0	o_1	o_2	o_3	o_4	o_5	o_6	o_7
0	×	×	×	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
1	0	1	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	1

Izhodne funkcije 3/8 dekodirnika (Tabela 8.11) so:
$o_0 = E\bar{s}_2\bar{s}_1\bar{s}_0$, $o_1 = E\bar{s}_2\bar{s}_1 s_0$, $o_2 = E\bar{s}_2 s_1\bar{s}_0$, $o_3 = E\bar{s}_2 s_1 s_0$,
$o_4 = Es_2\bar{s}_1\bar{s}_0$, $o_5 = Es_2\bar{s}_1 s_0$, $o_6 = Es_2 s_1\bar{s}_0$, $o_7 = Es_2 s_1 s_0$
3/8 demultiplekserja pa so:
$o_0 = I\bar{s}_2\bar{s}_1\bar{s}_0$, $o_1 = I\bar{s}_2\bar{s}_1 s_0$, $o_2 = I\bar{s}_2 s_1\bar{s}_0$, $o_3 = I\bar{s}_2 s_1 s_0$,
$o_4 = Is_2\bar{s}_1\bar{s}_0$, $o_5 = Is_2\bar{s}_1 s_0$, $o_6 = Is_2 s_1\bar{s}_0$, $o_7 = Is_2 s_1 s_0$

Realizacija logičnih funkcij z dekodirniki

Dekodirnik ni splošno namenski kombinacijski blok, lahko pa ga tudi uporabimo za realizacijo logičnih funkcij. Obravnavamo ga kot generator mintermov (1/2DEK $\Rightarrow$ 2 minterma, 2/4DEK $\Rightarrow$ 4 mintermi, 3/8DEK $\Rightarrow$ 8 mintermov). Vsaka logična funkcija z n spremenljivkami je izražena z disjunkcijo mintermov, zato je lahko izvedena z $n/2^n$ dekodirnikom, ki mu dodamo disjunkcijo za povezavo izhodov pri katerih ima funkcija vrednosti 1.

Primer: Vzemimo logični funkciji $f^3 = \vee(0, 4, 7)$ in $g^3 = \vee(1, 5, 7)$ ter ju realizirajmo z enim 3/8 dekodirnikom in OR vrati (Slika 8.12).

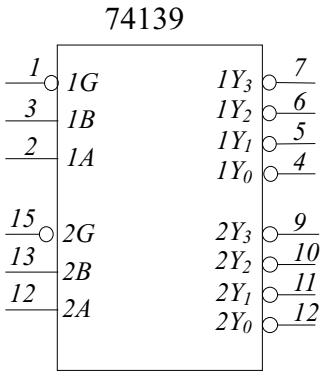


Slika 8.12: Realizacija logičnih funkcij f, g s 3/8 dekoderjem in OR vrati

TTL dekodirnik/demultiplekser

Za dekodirnike in demultiplekserje imamo enake izhodne funkcije, ki so odvisne samo od definicije podatkovnega vhoda E/I, zato jih izvedemo z enim samim logičnim vezjem. V TTL tehnologiji imamo naslednje demultiplekserje/dekodirnike:

74139 - (2/4 DEK/DMX) vsebuje dva dekodirnika/demultiplekserja (Slika 8.13) z dvema krmilnima signaloma $S_1 = B$ in $S_0 = A$. Enable signal $\bar{G}$ in vsi izhodi so aktivni ob nizki vrednosti signala (0). Tabela 8.12 prikazuje izhodne funkcije čipa. Pri vhodu $\bar{G} = 1$ je na vse izhode vsiljena logična vrednost 1, sicer pa je ob $\bar{G} = 0$ samo na en izhod vsiljena 0, ostali pa imajo 1.



Slika 8.13: Dekodirnik/demultiplekser

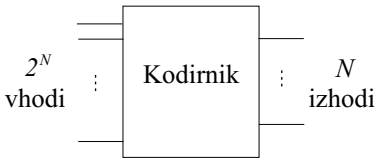
Tabela 8.12: TTL 74139

$\bar{G}$	B	A	$\bar{y}_0$	$\bar{y}_1$	$\bar{y}_2$	$\bar{y}_3$
1	×	×	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

74138 - (3/8 DEK/DMX) - vsebuje en dekodirnik/demultiplekser s tremi krmilnimi signali $S_2 = C$, $S_1 = B$ in $S_0 = A$. Enable signal $\bar{G}$ in vsi izhodi so aktivni ob nizki vrednosti signala (0).

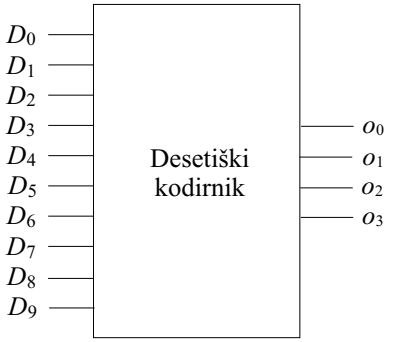
8.3 Kodirnik

Kodirnik je kombinacijsko vezje z 2^N ali manj vhodi in N izhodi, ki predstavljajo dvojiško kodo signala na vhodu (Slika 8.14). Uporabljamo ga za pretvorbo različnih kod, kot so desetiška, osmiška, šestnajstiška, idr. v dvojiško kodo. Pogosto so uporabljeni v mikroprocesorskih vhodno/izhodnih podsistemih, ki delujejo na osnovi prekinitvenih zahtev. V takem primeru bo dvojiški kodirnik opazoval vhode in določil katera enota je zahtevala obdelavo. Pri tem mora biti zagotovljeno, da je vedno samo en vhod aktiven.



Slika 8.14: Kodirnik

Primer: Oglejmo si kodirnik za pretvorbo desetiške kode v dvojiško, ki ima 10 vhodov za predstavitev desetiških števil od 0 do 9 v obliki enega aktivnega signala in daje na izhodu 4-bitno dvojiško kodo.



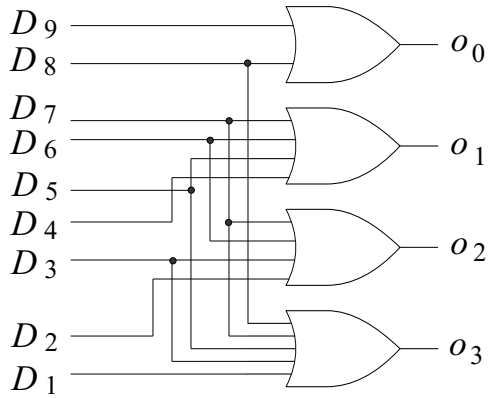
Slika 8.15: Desetiški kodirnik

Tabela 8.13: Desetiški kodirnik

D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	o_0	o_1	o_2	o_3
1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	1
0	0	0	0	0	0	1	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	1

$$
\begin{aligned}
o_0 &= D_8 \vee D_9 \\
o_1 &= D_4 \vee D_5 \vee D_6 \vee D_7 \\
o_2 &= D_2 \vee D_3 \vee D_6 \vee D_7 \\
o_3 &= D_1 \vee D_3 \vee D_5 \vee D_7 \vee D_9
\end{aligned}
$$

Logična shema desetiškega kodirnika (Slika 8.16), kjer vidimo da so kodirniki enonivojska vezja sestavljena iz disjunktivnih logičnih vrat.



Slika 8.16: Desetiški kodirnik - logična shema

8.4 Vaje

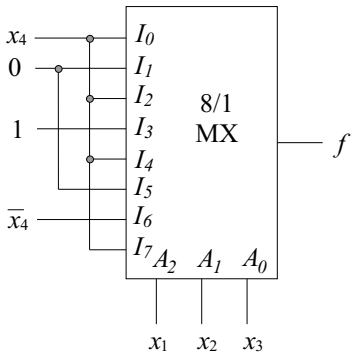
VAJA 8.4.1:

Realizirajte logično funkcijo $f(x_1, x_2, x_3, x_4) = \vee(1, 5, 6, 7, 9, 12, 15)$ z enim 3-naslovnim multiplekserjem.

Naslovne spremenljivke x_1, x_2, x_3 so pripeljane na naslovne vhode: $A_2 = x_1$, $A_1 = x_2$, $A_0 = x_3$. Če logično funkcijo razčlenimo po treh spremenljivkah dobimo funkcijske ostanke, ki so odvisni od spremenljivke x_4, ali pa so konstanta 0 ali 1. Funkcijo bomo zapisali v pravilnostni tabeli in določili podatkovne vhode multiplekserja: I_0, I_1, ..., I_7 (Tabela 8.14).

Tabela 8.14: Razčlenitev funkcije f po $x_1x_2x_3$

$x_1x_2x_3x_4$	f	
0 0 0 0	0	
0 0 0 1	1	$I_0 = x_4$
0 0 1 0	0	
0 0 1 1	0	$I_1 = 0$
0 1 0 0	0	
0 1 0 1	1	$I_2 = x_4$
0 1 1 0	1	
0 1 1 1	1	$I_3 = 1$
1 0 0 0	0	
1 0 0 1	1	$I_4 = x_4$
1 0 1 0	0	
1 0 1 1	0	$I_5 = 0$
1 1 0 0	1	
1 1 0 1	0	$I_6 = \bar{x}_4$
1 1 1 0	0	
1 1 1 1	1	$I_7 = x_4$



Slika 8.17: Realizacija funkcije $f(x_1, x_2, x_3, x_4) = \vee(1, 5, 6, 7, 9, 12, 15)$

VAJA 8.4.2:

Realizirajte podano logično funkcijo z 2-naslovnimi multiplekserji.

$$f(x_1, x_2, x_3, x_4) \quad = \quad (x_1 \nabla x_3) \equiv (x_2 \nabla x_4)$$

Logična funkcija je podana v zapisu z XOR in XNOR operatorji s štirimi spremenljivkami, zato bomo uporabili kaskadno realizacijo. Najprej logično funkcijo razčlenimo po dveh spremenljivkah, kjer imamo šest možnosti: x_1x_2; x_1x_3; x_1x_4; x_2x_3; x_2x_4; x_3x_4.

Oglejmo si rešitev naloge za razčlenitev logične funkcije po spremenljivkah x_1x_2, ki bosta postali naslovni spremenljivki $A_1 = x_1$ in $A_0 = x_2$:

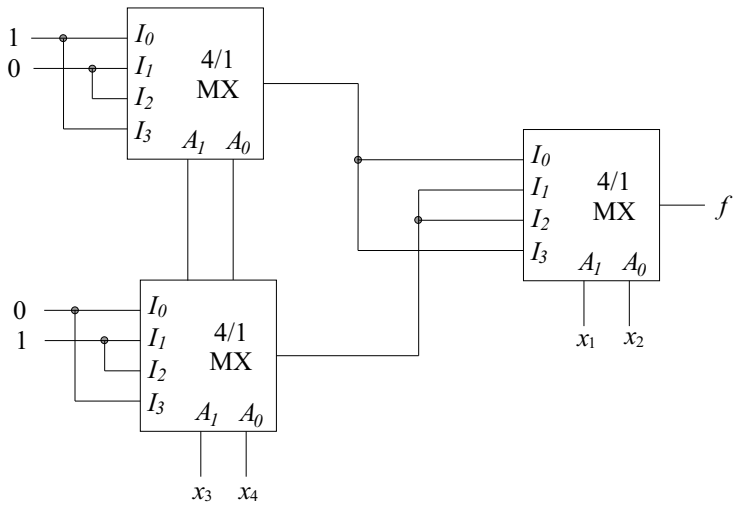
$$\begin{aligned} f(x_1,x_2,x_3,x_4) &= \bar{x}_1\bar{x}_2((0\nabla x_3) \equiv (0\nabla x_4)) \vee \bar{x}_1x_2((0\nabla x_3) \equiv (1\nabla x_4)) \\ &\vee \; x_1\bar{x}_2((1\nabla x_3) \equiv (0\nabla x_4)) \vee x_1x_2((1\nabla x_3) \equiv (1\nabla x_4)) \\ &= \bar{x}_1\bar{x}_2(x_3 \equiv x_4) \vee \bar{x}_1x_2(x_3 \equiv \bar{x}_4) \vee x_1\bar{x}_2(\bar{x}_3 \equiv x_4) \vee x_1x_2(\bar{x}_3 \equiv \bar{x}_4) \\ &= \bar{x}_1\bar{x}_2(x_3 \equiv x_4) \vee \bar{x}_1x_2(x_3\nabla x_4) \vee x_1\bar{x}_2(x_3\nabla x_4) \vee x_1x_2(x_3 \equiv x_4) \end{aligned}$$

V oklepajih smo dobili funkcijske ostanke $F_0 = F_3 = x_3 \equiv x_4$ in $F_1 = F_2 = x_3\nabla x_4$. Dobljeni funkciji bomo znova realizirali z dvema 2-naslovnima multiplekserjema, zato ju razčlenimo po spremenljivkah x_3, x_4 (Tabela 8.15):

Tabela 8.15: Razčlenitev logičnih funkcij $F_0 = F_3$ in $F_1 = F_2$

x_3	x_4	$F_0 = F_3 = x_3 \equiv x_4$	x_3	x_4	$F_1 = F_2 = x_3\nabla x_4$
0	0	1=f_0	0	0	0=f_0
0	1	0=f_1	0	1	1=f_1
1	0	0=f_2	1	0	1=f_2
1	1	1=f_3	1	1	0=f_3

V tabeli zapisane funkcijske vrednosti f_0, f_1, f_2, f_3 ustrezajo podatkovnim vhodom 2-naslovnih multiplekserjev I_0, I_1, I_2, I_3. Iz razčlenitev logičnih funkcij za realizacijo logične funkcije z 2-naslovnimi multiplekserji narišemo logično shemo, kjer sta v prvem nivoju dva multiplekserja z naslovnima spremenljivkama x_3 in x_4, katerih izhoda sta vezana na multiplekser drugega nivoja, ki daje izhodno funkcijo (Slika 8.18).



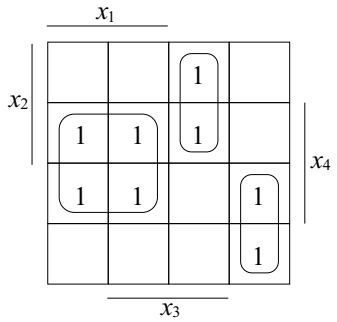
Slika 8.18: Kaskadna realizacija: $f(x_1, x_2, x_3, x_4) = (x_1\nabla x_3) \equiv (x_2\nabla x_4)$

VAJA 8.4.3:

Realizirajte podano logično funkcijo z minimalnim številom 1-naslovnih multiplekserjev.

$$f(x_1, x_2, x_3, x_4) = \&(13, 12, 11, 10, 7, 5, 3, 1)$$

Logično funkcijo bomo najprej minimizirali in iz dobljene oblike poiskali najbolj enostavno razčlenitev po eni spremenljivki za uporabo 1-naslovnih multiplekserjev.
Logično funkcija iz PKNO pretvorimo v PDNO, jo vpišemo v Veitchev diagram in minimiziramo (Slika 8.19).
$f(x_1, x_2, x_3, x_4) = \&(13, 12, 11, 10, 7, 5, 3, 1)$
manjkajoči maksermi M_j za $f_i = 1$: 15,14,9,8,6,4,2,0
$f(x_1, x_2, x_3, x_4) = \vee(0, 1, 6, 7, 9, 11, 13, 15)$



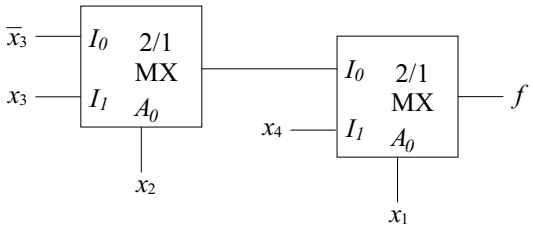
Slika 8.19: Minimizacija: $f(x_1, x_2, x_3, x_4) = \&(13, 12, 11, 10, 7, 5, 3, 1)$

$$f(x_1, x_2, x_3, x_4) = x_1x_4 \vee \bar{x}_1x_2x_3 \vee \bar{x}_1\bar{x}_2\bar{x}_3$$

Minimalna disjunktivna normalna oblika funkcije ima v eni konjunkciji x_1, v dveh pa $\bar{x}_1$ in s preoblikovanjem zapisa dobimo razčlenitev funkcije po eni spremenljivki, kar ustreza 1-naslovnemu multiplekserju. Pri $\bar{x}_1$ imamo funkcijski ostanek odvisen od spremenljivk x_2 in x_3 tudi razčlenjen po eni ali drugi spremenljivki, kar omogoča uporabo še enega 1-naslovnega multiplekserja.

$$f(x_1, x_2, x_3, x_4) = \bar{x}_1(x_2x_3 \vee \bar{x}_2\bar{x}_3) \vee x_1(x_4)$$
$$f(x_1, x_2, x_3, x_4) = \bar{x}_1[\bar{x}_2(\bar{x}_3) \vee x_2(x_3)] \vee x_1(x_4)$$

Iz končnega zapisa funkcije dobimo realizacijo logične funkcije z dvema 1-naslovnima multiplekserjema v kaskadni izvedbi (Slika 8.20).



Slika 8.20: Kaskadna realizacija: $f(x_1, x_2, x_3, x_4) = \&(13, 12, 11, 10, 7, 5, 3, 1)$

VAJA 8.4.4

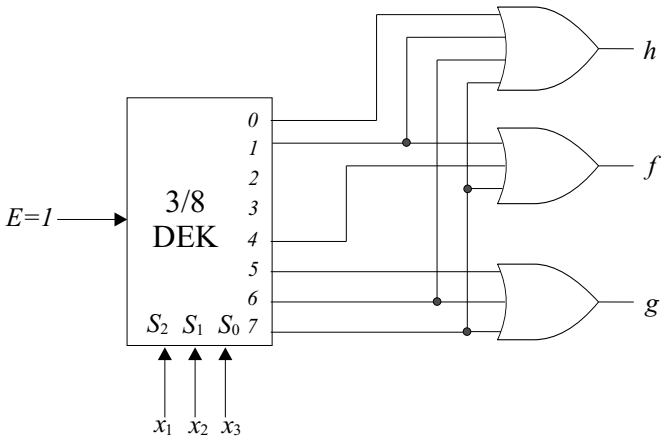
Realizirajte podane logične funkcije z enim 3/8 dekodirnikom in OR vrati.
$f^3 = \vee(1, 4, 7)$
$g(x_1, x_2, x_3) = x_1x_3 \vee x_1x_2$
$h(x_1, x_2) = \bar{x}_1\bar{x}_2 \vee x_1x_2$

Za realizacijo logičnih funkcij z dekoderjem potrebujemo popolne disjunktivne normalne oblike, zato logične funkcije zapišemo v pravilnostno tabelo (Tabela 8.16).

Tabela 8.16: Logične funkcije f, g, h

$x_1x_2x_3$	f	g	h
0 0 0	0	0	1
0 0 1	1	0	1
0 1 0	0	0	0
0 1 1	0	0	0
1 0 0	1	0	0
1 0 1	0	1	0
1 1 0	0	1	1
1 1 1	1	1	1

Za realizacijo treh logičnih funkcij potrebujemo 3/8DEK, dvoje 3-vhodnih in ena 4-vhodna OR vrata (Slika 8.21). Čeprav ima logična funkcija h samo dve vhodni spremenljivki, jo lahko realiziramo z istim dekoderjem, ker smo jo v pravilnostni tabeli določili tako, da je odvisna samo od x_1 in x_2, x_3 pa ima lahko vrednost 0 ali 1.



Slika 8.21: Realizacija logičnih funkcij f, g, h s 3/8 dekoderjem in OR vrati

Poglavje 9

Programabilni logični gradniki

Doslej smo obravnavali različne načine realizacije logičnih funkcij z dvo- ali večnivojskimi vezji in strukturalnimi gradniki, kjer na različne načine skušamo zmanjšati obsežnost logičnih vezij tako glede števila logičnih elementov (logična vrata, multiplekserji,...), kakor tudi števila povezav. Strukturalni gradniki so za realizacijo logičnih vezij zelo primerni, imajo pa vnaprej določeno izhodno funkcijo, kar lahko pomeni določene omejitve.

Proizvajalci čipov so zato razvili logične gradnike, ki so zadosti splošni za uporabo v različnih aplikacijah. Možno jih je zgraditi z ureditvijo konjunktivnih in disjunktivnih vrat v splošni strukturi polja, katerega povezave so določene ali programirane. Taki gradniki so na voljo v različnih izvedbah in omogočajo uporabniku, da jim sam definira funkcijo za določeno aplikacijo.

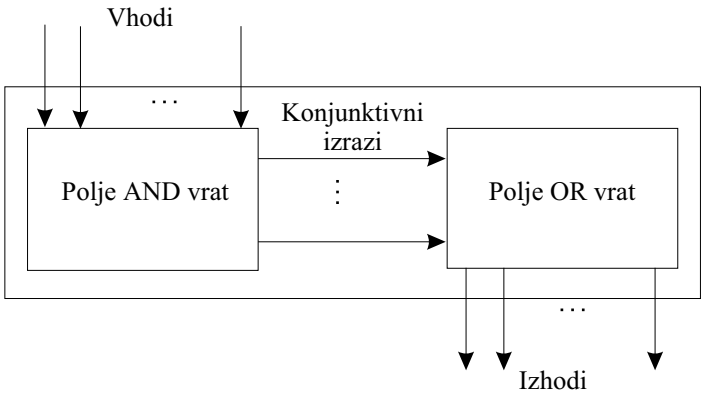
Programabilni gradniki se razlikujejo med seboj po načinu programiranja, sposobnostih brisanja in času, ki je potreben za to [4]. Oglejmo si tri različne tehnologije programiranja:

- Pri gradnikih z varovalkami, ki imajo taljive povezave, je logika z vsemi podanimi notranjimi povezavami zasnovana pri proizvajalcu. Programabilni gradnik ima izhodne funkcije določene s prekinitvijo varovalk pri zahtevanih povezavah. Varovalko prekinemo tako, da nanjo pripeljemo visok tok. Takšen element je uporaben samo za enkratno programiranje.

- Gradniki brisljivi z ultravijolično svetlobo (UV) imajo vse povezave, ki jih lahko določa uporabnik, nepovezane in se vzpostavijo med procesom programiranja. Na čipu je okno, ki omogoča brisanje vseh obstoječih povezav z UV svetlobo. Brisanje traja približno 10 minut in nato je čip znova pripravljen za programiranje.

- Električno brisljivi gradniki so namenjeni za aplikacije z minimalnim časom brisanja, ki je izvedeno kot inverzen proces programiranja. V nekaterih je možno izvesti samo delno, v drugih pa kompletno brisanje čipa in čas brisanja je v tem primeru zanemarljiv. S strani uporabnika je proces programiranja enak kot pri gradnikih z varovalkami.

Slika 9.1 prikazuje splošen blok diagram polja logičnih komponent z več vhodi in več izhodi sestavljenih iz velikega števila konjunkcij in disjunkcij organiziranih v AND in OR polju. AND polje preslika vhode v konjunktivne izraze v odvisnosti od programiranih povezav, OR polje pa sprejme te izraze in jih disjunktivno poveže v končno obliko.



Slika 9.1: Programabilni logični gradniki

Programabilni logični gradniki se razlikujejo po tem, kateri del gradnika je programabilen. Za programiranje uporabimo napravo imenovano programator. Glede na možnosti programiranja bomo obravnavali tri najenostavnejše tipe programabilnih gradnikov: PLA, PAL in ROM (Tabela 9.1). Oznaka '×' pove katero polje programiramo, oznaka '-' pa označuje fiksno polje.

Tabela 9.1: Programabilni gradniki

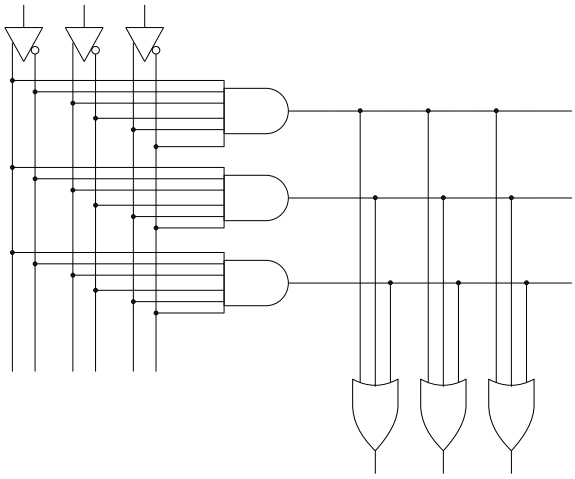
	AND polje	OR polje
PLA	×	×
PAL	×	-
ROM	-	×

9.1 PLA (Programmable Logic Array)

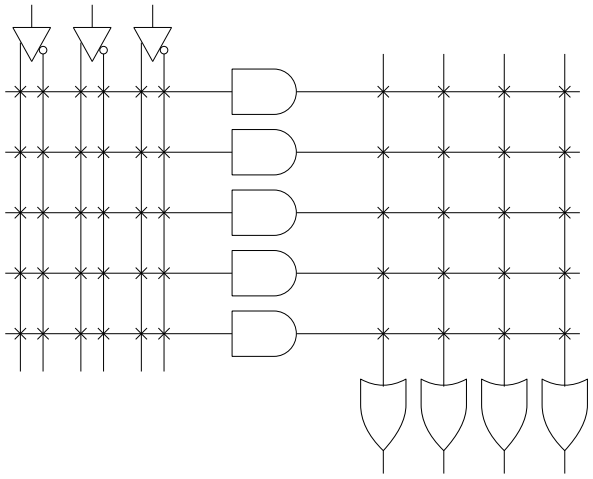
V PLA sta programabilni AND in OR polje gradnika. S PLA gradniki lahko realiziramo manjše število obsežnih logičnih funkcij. Obsežnost je določena s številom vhodnih spremenljivk, številom konjunktivnih izrazov in številom izhodov. Za primer poglejmo tipično TTL programabilno logično polje (FPLA-Field programmable logic array), ki ima 16 vhodov, 48 konjunkcij in 8 izhodov v čipu s 24 podatkovnimi pini. Velikost tega PLA gradnika pomeni isto kot oseminštirideset 16-vhodnih AND vrat in osem 48-vhodnih OR vrat. Če to primerjamo s čipom nižje stopnje integracije z 12 podatkovnimi pini, imamo v njem štiri 2-vhodna vrata.

Zgradba PLA programabilnega gradnika pred programiranjem ima vse povezave vhodov v konjunktivna vrata in izhode konjunktivnih vrat na vhode disjunktivnih vrat. Vhodna spremenljivka vstopa v vsaka konjunktivna vrata kot x_i in $\bar{x}_i$. V podani strukturi imamo pred programiranjem za tri vhodne spremenljivke vse povezave na tri 6-vhodna AND vrata, katerih izhodi so vezani na tri 3-vhodna OR vrata (Slika 9.2).

Slika 9.3 prikazuje poenostavljeno strukturo PLA pred programiranjem za tri vhodne spremenljivke in štiri izhodne funkcije, tako da rišemo samo eno črto pri logičnih vratih. Oznake × določajo število vhodov v konjunktivna vrata v AND polju in število vhodov v disjunktivna vrata v OR polju gradnika.



Slika 9.2: PLA pred programiranjem - dejanska izvedba z vrati



Slika 9.3: PLA pred programiranjem - poenostavljena strukturna shema

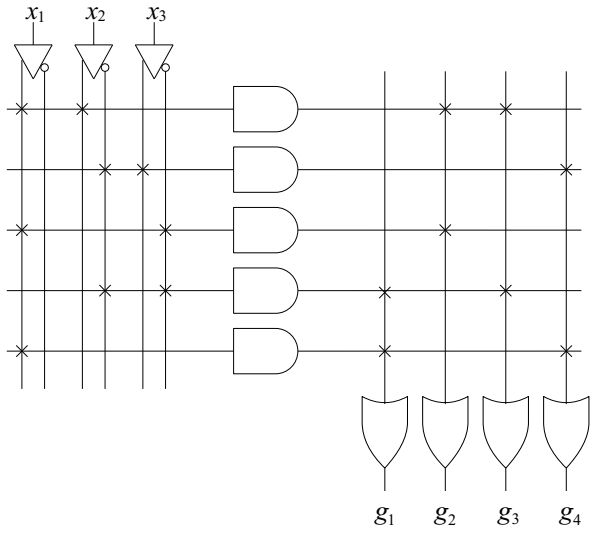
Primer: Oglejmo si realizacijo logičnih funkcij g_1, g_2, g_3, g_4 s PLA, če so vse odvisne od vhodnih spremenljivk x_1, x_2, x_3.

$$\begin{aligned} g_1 &= x_1 \vee \bar{x}_2\bar{x}_3 \\ g_2 &= x_1\bar{x}_3 \vee x_1x_2 \\ g_3 &= \bar{x}_2\bar{x}_3 \vee x_1x_2 \\ g_4 &= \bar{x}_2x_3 \vee x_1 \end{aligned}$$

Za opis logičnih funkcij g_1, g_2, g_3, g_4 bomo uporabili programabilno matriko, ki je poenostavljena oblika pravilnostne tabele (Tabela 9.2) in opisuje povezave spremenljivk v AND polju in konjunkcij v OR polju. Oznake pri spremenljivkah so: '1' - spremenljivka x_i je v konjunkciji, '0' - negirana spremenljivka $\bar{x}_i$ je v konjunkciji, '-' - spremenljivka ni v konjunkciji. Oznaka 1 pri funkcijah pomeni povezavo konjunkcije v OR vrata, oznaka 0 pa pove, da konjunkcija ne vstopa v OR vrata izhodne funkcije (Tabela 9.2).
Za programiranje štirih logičnih funkcij potrebujemo tri vhodne spremenljivke in pet konjunkcij v AND polju ter štiri izhodne funkcije. Po programiranju ostanejo v AND polju povezave, ki so zapisane v konjunktivnih izrazih logičnih funkcij. V OR polju pa ostanejo tiste povezave, ki označujejo disjunktivne izraze v logičnih funkcijah (Slika 9.4).

Tabela 9.2: PLA programabilna matrika

Konjunkcija	$x_1x_2x_3$	$g_1g_2g_3g_4$
x_1x_2	1 1 -	0 1 1 0
$\bar{x}_2x_3$	- 0 1	0 0 0 1
$x_1\bar{x}_3$	1 - 0	0 1 0 0
$\bar{x}_2\bar{x}_3$	- 0 0	1 0 1 0
x_1	1 - -	1 0 0 1



Slika 9.4: PLA po programiranju

9.2 PAL (Programmable Array Logic)

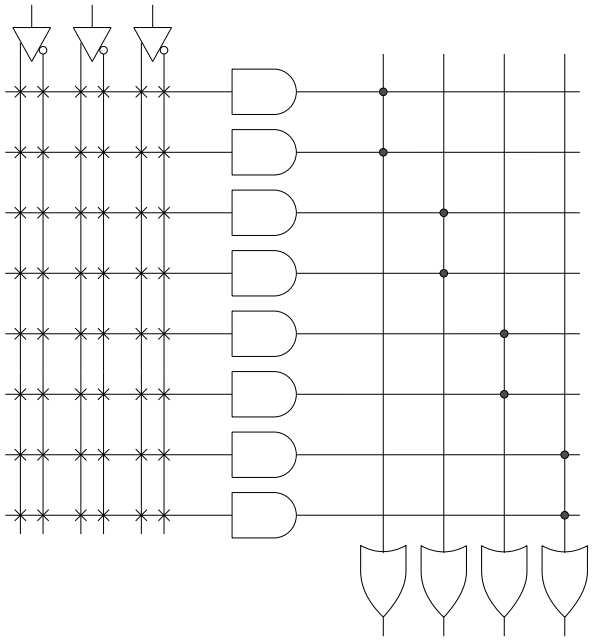
V PAL vezju je programabilno AND polje gradnika, medtem ko je OR polje fiksno. Število vhodov v OR delu je običajno omejeno na 2, 4, 8 ali 16. Za PAL je pomembna povezava med kompleksnostjo funkcij (število konjunkcij na OR vrata) in številom neodvisnih funkcij, ki jih lahko imamo v vezju. Več kot je vhodov v OR vrata, manj imamo funkcijskih izhodov v PALu. Vzemimo, da imamo PAL družino s 16 vhodi in 16 konjunkcijami, ki se razlikujejo v organizaciji OR polja: a) štiri OR vrata s 4 vhodi, b) dvoje OR vrat z 8 vhodi, c) ena OR vrata s 16 vhodi. AND polje je pri vseh popolnoma programabilno. Za PAL je v primerjavi s PLA značilno, da enakih konjunkcij ne moremo večkrat uporabiti.

Slika 9.5 prikazuje neprogramiran PAL s tremi vhodnimi spremenljivkami, osmimi 6-vhodnimi konjunkcijami ($\times$) in štirimi 2-vhodnimi disjunkcijami ($\bullet$).

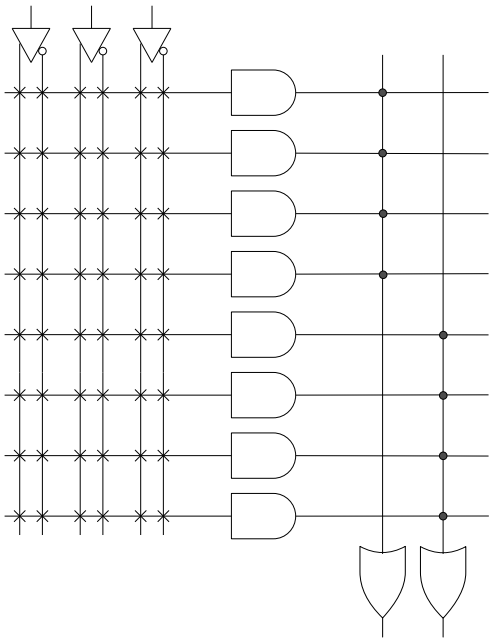
Slika 9.6 prikazuje neprogramiran PAL s tremi vhodnimi spremenljivkami, osmimi 6-vhodnimi konjunkcijami ($\times$) in dvema 4-vhodnima disjunkcijama ($\bullet$).

Primer: Realizirajmo logične funkcije g_1, g_2, g_3, g_4 s PAL gradnikom.

$$
\begin{aligned}
g_1 &= x_1 \vee \bar{x}_2\bar{x}_3 \\
g_2 &= x_1\bar{x}_3 \vee x_1x_2 \\
g_3 &= \bar{x}_2\bar{x}_3 \vee x_1x_2 \\
g_4 &= \bar{x}_2x_3 \vee x_1
\end{aligned}
$$

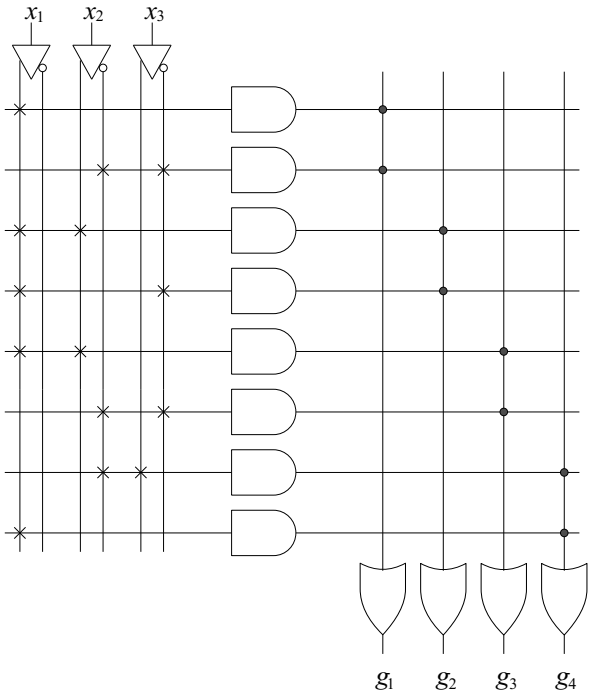


Slika 9.5: PAL pred programiranjem - 2-vhodne disjunkcije



Slika 9.6: PAL pred programiranjem - 4-vhodne disjunkcije

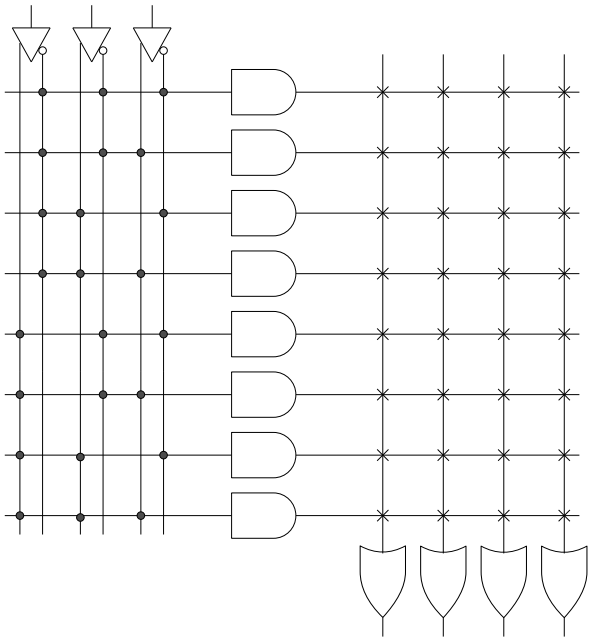
Vse štiri logične funkcije imajo največ dva vhoda v OR vrata, zato bomo uporabili PAL z dvema fiksnima vhodoma v OR polju gradnika. Po programiranju logičnih funkcij g_1, g_2, g_3, g_4 so v AND polju pri vsaki izhodni funkciji ohranjene povezave $\times$ pri spremenljivkah, ki vstopajo v konjunktivne izraze logične funkcije ali spremenljivke, ki vstopajo same v disjunkcijo (Slika 9.7). V PALu imamo programiranih 8 konjunkcij za štiri izhodne funkcije.



Slika 9.7: PAL po programiranju

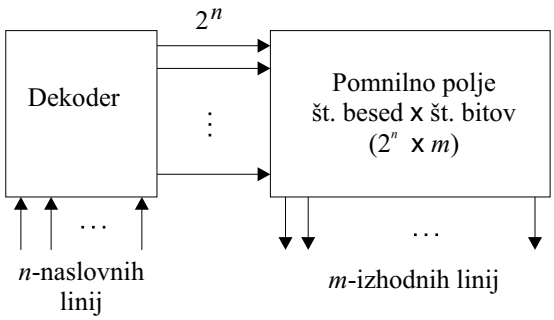
9.3 ROM (Read-Only Memories)

ROM ima programabilno OR polje gradnika, AND polje pa je fiksno. Slika 9.8 prikazuje neprogramiran ROM za tri vhodne spremenljivke in štiri izhodne funkcije, če ga opazujemo v smislu povezav v AND in OR polju. V AND polju imamo fiksno povezanih osem mintermov, kar je pravzaprav dekoder z n vhodi in 2^n izhodi. Izhodi dekoderja so pripeljani na programabilno OR polje. ROM opišemo kot programabilni element z notranjim



Slika 9.8: ROM pred programiranjem

n vhodnim dekoderjem za preslikavo n naslovnih linij v izbrano besedo, ki je poslana na izhodne linije. Vsaka beseda v ROMu je predstavljena z m biti (Slika 9.9). Velikost ROMa podamo kot (število besed× število bitov).



Slika 9.9: ROM element

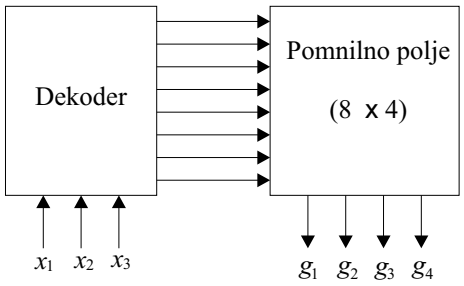
Primer: Uporabimo ROM za realizacijo logičnih funkcij g_1, g_2, g_3, g_4.

$$\begin{aligned} g_1 &= x_1 \vee \bar{x}_2\bar{x}_3 \\ g_2 &= x_1\bar{x}_3 \vee x_1x_2 \\ g_3 &= \bar{x}_2\bar{x}_3 \vee x_1x_2 \\ g_4 &= \bar{x}_2x_3 \vee x_1 \end{aligned}$$

Logične funkcije zapišemo v pravilnostno tabelo, ker funkcijske vrednosti določajo vsebino OR polja. Izvedba logičnih funkcij g_1, g_2, g_3, g_4 s tremi vhodnimi spremenljivkami x_1, x_2, x_3 (Tabela 9.3) zahteva ROM vezje z 8 besedami dolžine 4-bitov.

Tabela 9.3: Pravilnostna tabela funkcij g_1, g_2, g_3, g_4

$x_1x_2x_3$	$g_1g_2g_3g_4$
0 0 0	1 0 1 0
0 0 1	0 0 0 1
0 1 0	0 0 0 0
0 1 1	0 0 0 0
1 0 0	1 1 1 1
1 0 1	1 0 0 1
1 1 0	1 1 1 1
1 1 1	1 1 1 1



Slika 9.10: Realizacija funkcij g_1, g_2, g_3, g_4 z 8x4 ROM elementom

Število besed v ROMu je določeno s številom vhodnih spremenljivk (2^n), število izhodnih

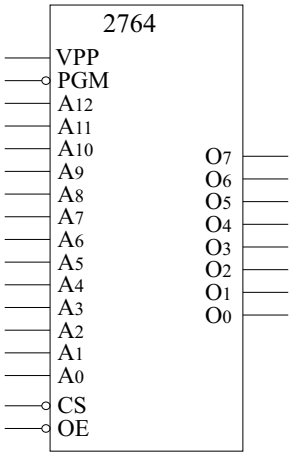
funkcij pa določa število bitov (Slika 9.10). Če je funkcija po vrednosti 1, bo v ROMu vpisana enica na ustreznem bitu besede, sicer pa bo vpisana 0.

Glede na možnosti programiranja ločimo ROM vezja z različnimi predponami:

- ROM - tovarniško programiran s funkcijo branja,
- PROM - enkratno programiranje,
- EPROM - brisljiv z UV svetlobo in večkratna možnost programiranja,
- EEPROM - električno brisljiv in programirljiv.

ROM komponente

ROM je na voljo v različnih velikostih in dolžinah besede. 27512 EPROM ima 2^{16} 8-bitnih besed, ali pol milijona bitov v 24 pinskem čipu. Oglejmo si malo bolj podrobno ROM z oznako 2764, ki ima $2^{13} = 8192$ 8-bitnih besed (Slika 9.11). Čip ima 13 naslovnih linij, ki določajo 8192 besed in 8 tri-state izhodnih linij za vsak bit besede. Dodane imamo še štiri krmilne vhode. Tri-state izhodi so omogočeni z nizkim aktivnim signalom $\overline{OE}$. Čip ima nizek aktivni signal izbire $\overline{CS}$ za povezovanje manjših/ožjih ROMov v večje/širše pomnilnike. Ker je ROM električno programiran, ima vhod $\overline{PGM}$ za določanje programirnega načina namesto bralnega načina delovanja. Vhod VPP pa zagotavlja potreben visokonapetostni izvor med programiranjem.



Slika 9.11: ROM $2^{13} \times 8$

9.4 Izbira programabilnega logičnega vezja

Spoznali smo tri različne tipe programabilnih gradnikov, zato se nam pri njihovi uporabi postavlja vprašanje, kako izbrati primerno programabilno logično vezje? Odgovor je odvisen od strukture funkcij:

- Kakšno je število konjunktivnih izrazov za izvedbo logičnih funkcij (majhno število konjunkcij pri velikem številu vhodnih spremenljivk, veliko število spremenljivk pri majhnem številu konjunkcij ali poljubno).
- Do katere stopnje se konjunktivni izrazi ponavljajo v različnih funkcijah (ena konjunkcija se pojavlja v več funkcijah, ali samo v eni).

- Pomembno je število konjunkcij, ki so disjunktivno povezane (funkcije z majhnim številom konjunkcij, z velikim številom konjunkcij, ali poljubno).

PLA so primerni za manjše število različnih konjunkcij, ki se pojavljajo na številnih izhodnih funkcijah. PAL je tudi omejen z majhnim številom konjunkcij na izhodih. Konjunktivnih izrazov, ki se ponavljajo, ni možno večkrat uporabiti, ampak so programirani vsakič posebej. PLA/PAL vezja zahtevajo več časa, ker moramo pri programiranju vedno izhajati iz minimalnih disjunktivnih normalnih oblik, kar pomeni da je pri razvoju potreben še čas za minimizacijo logičnih funkcij. ROM je primeren za veliko število konjunkcij. Pri programiranju izhajamo iz pravilnostne tabele, zato je krajši razvojni čas. Velikost ROMa je določena samo s številom vhodov in izhodov in se podvoji z dodanim vhodom.

9.5 Vaje

VAJA 9.5.1:

S programabilnimi gradniki PLA, PAL in ROM predstavite realizacijo polnega seštevalnika in polnega odštevalnika, tako da dobimo vsoto in razliko istih števil. Na vhodu imamo x_0, y_0 in c_{in} ali b_{in}, ki dajejo vsoto $z_0 = x_0 + y_0 + c_{in}$ in prenos c_0 ter razliko $d_0 = x_0 - y_0 - b_{in}$ in sposodek b_0.

Polni seštevalnik in odštevalnik bomo zapisali v pravilnostni tabeli (Tabela 9.4).

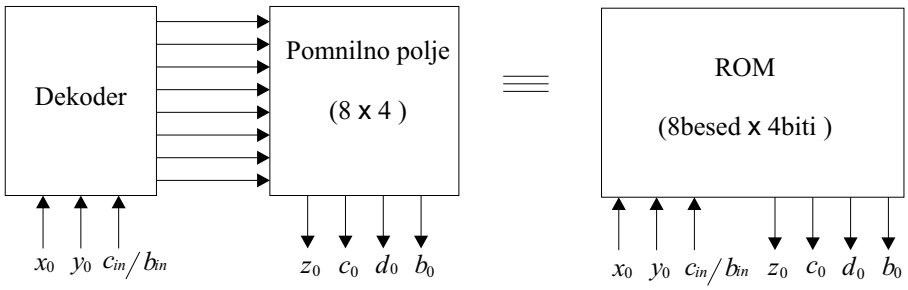
Tabela 9.4: Polni seštevalnik in odštevalnik

x_0	y_0	c_{in}/b_{in}	z_0	c_0	d_0	b_0
0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	0	1	0	1	1
0	1	1	0	1	0	1
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	1	0	0	1	0	0
1	1	1	1	1	1	1

Logični funkciji za izhoda z_0 in d_0 sta enaki in ju ni možno minimizirati, zato bomo uporabili kar PDNO. Za izhoda c_0 in b_0 pa uporabimo MDNO, ki smo ju izračunali že v poglavju z aritmetičnimi vezji.

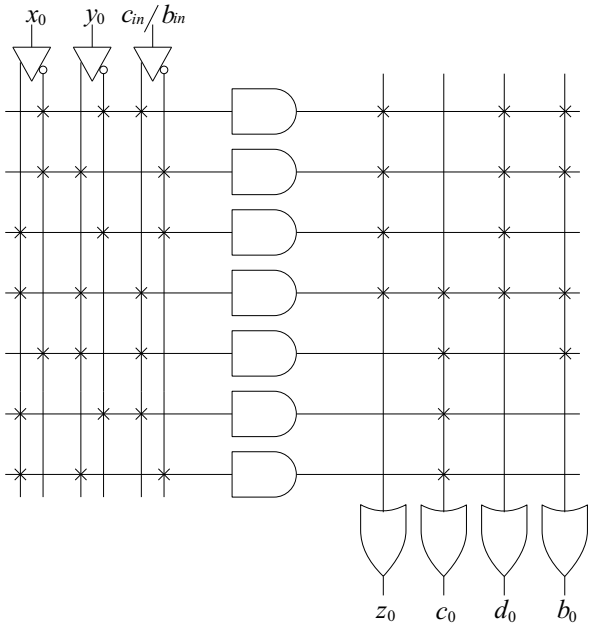
$$
\begin{aligned}
z_0 &= \bar{x}_0\bar{y}_0c_{in} \vee \bar{x}_0y_0\bar{c}_{in} \vee x_0\bar{y}_0\bar{c}_{in} \vee x_0y_0c_{in} \\
c_0 &= x_0y_0 \vee x_0c_{in} \vee y_0c_{in} \\
b_0 &= \bar{x}_0y_0 \vee \bar{x}_0b_{in} \vee y_0b_{in}
\end{aligned}
$$

V ROMu definiramo štiri izhodne funkcije, čeprav bi tudi tu lahko uporabili isti izhod za seštevanje in odštevanje. Za izvedbo polnega seštevalnika in odštevalnika potrebujemo 8×4 ROM (Slika 9.12).

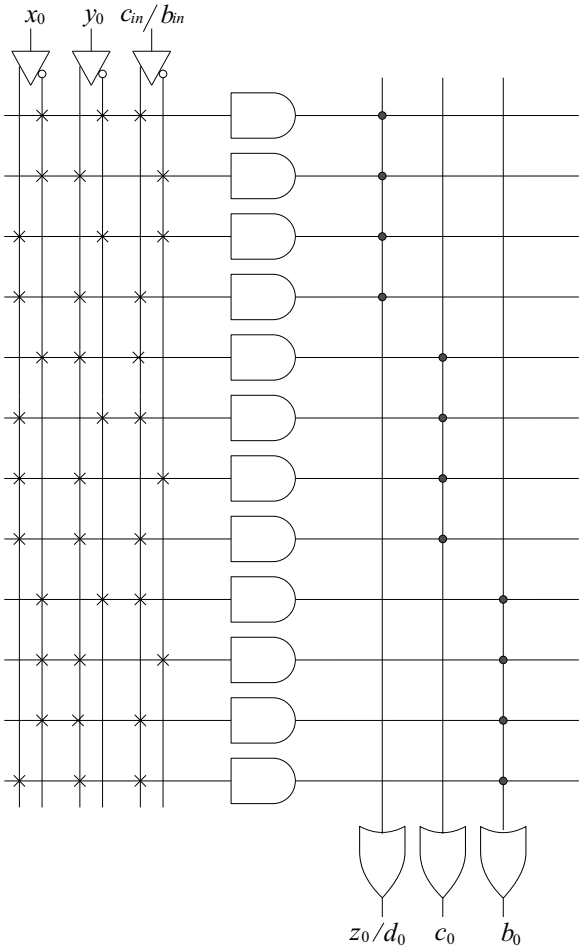


Slika 9.12: ROM - polni seštevalnik in odštevalnik

V realizaciji vezja s PLA uporabimo tri vhodne spremenljivke in štiri izhodne funkcije. Za vse štiri izhodne funkcije smo pri programiranju uporabili popolno disjunktivno normalno obliko, ker se določeni mintermi ponavljajo in ne potrebujemo dodatnih konjunktivnih operatorjev (Slika 9.13). V AND polju je potrebnih 7 konjunkcij in je enako pri programiranju PDNO ali MDNO za c_0 in b_0. V PAL gradniku smo zaradi prihranka prostora za vsoto z_0 in razliko d_0 uporabili isti izhod, ker sta funkcija seštevanja in odštevanja enaki (Slika 9.14).



Slika 9.13: PLA - polni seštevalnik in odštevalnik



Slika 9.14: PAL - polni seštevalnik in odštevalnik

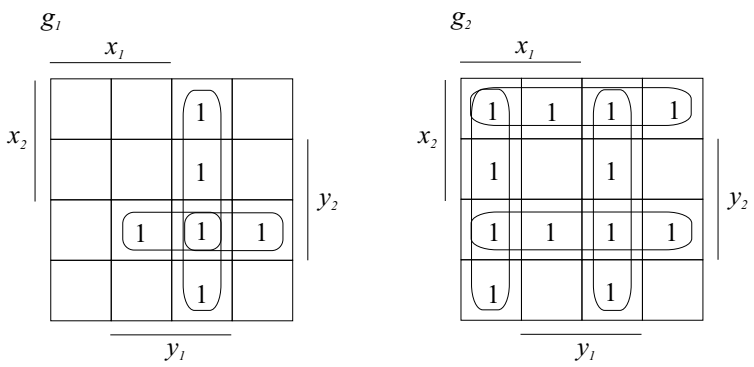
VAJA 9.5.2:

S programabilnim gradnikom PAL realizirajte vezje za primerjavo dvo-bitnih števil $X = (x_1, x_2)$ in $Y = (y_1, y_2)$, ki daje na izhodu $g_1 = g_2 = 0$, če sta števili enaki; $g_1 = g_2 = 1$, če je število X manjše od Y in $g_1 = 0, g_2 = 1$, če je število X večje od Y.

Zapišimo pravilnostno tabelo za delovanje primerjalnika (Tabela 9.5):

x_1	x_2	y_1	y_2	g_1	g_2
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	1	1
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	1	0	0
0	1	1	0	1	1
0	1	1	1	1	1
1	0	0	0	0	1
1	0	0	1	0	1
1	0	1	0	0	0
1	0	1	1	1	1
1	1	0	0	0	1
1	1	0	1	0	1
1	1	1	0	0	1
1	1	1	1	0	0

Poiščimo MDNO normalno obliko izhodnih funkcij g_1 in g_2 za realizacijo s PAL vezjem (Slika 9.15).

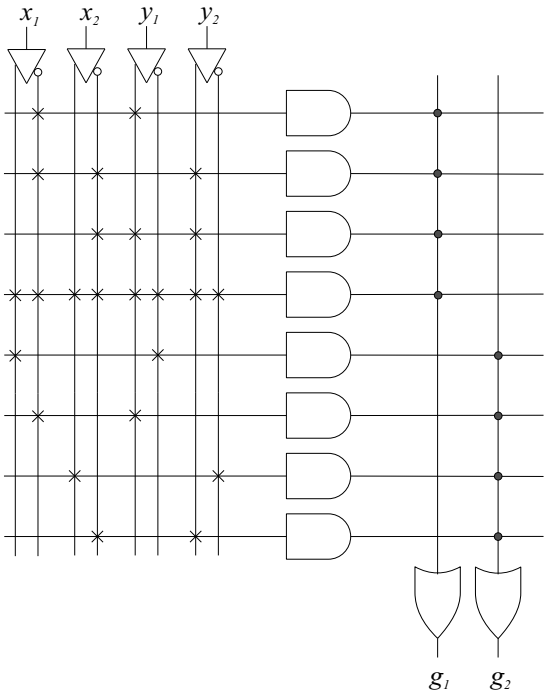


Slika 9.15: Minimizacija izhodnih funkcij primerjalnika: g_1, g_2

$$g_1 = \bar{x}_1 y_1 \vee \bar{x}_1 \bar{x}_2 y_2 \vee \bar{x}_2 y_1 y_2$$
$$g_2 = x_1 \bar{y}_1 \vee \bar{x}_1 y_1 \vee x_2 \bar{y}_2 \vee \bar{x}_2 y_2$$

V MDNO imamo za izhodni funkciji primerjalnika največ štiri konjunktivne izraze disjunktivno povezane, zato vzamemo PAL s štirimi fiksnimi vhodi v disjunkcijo (Slika 9.16). Pri izhodu g_1 je četrta konjunkcija neprogramirana, kar je enako konjunktivnemu izrazu

$x_i\bar{x}_i y_i \bar{y}_i$, $i = 1, 2$.



Slika 9.16: PAL - primerjalnik

Poglavje 10

Sekvenčna vezja

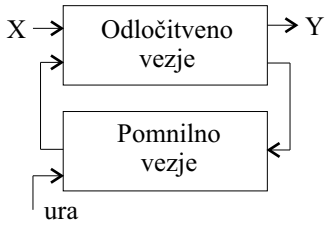
Sekvenčna logična vezja imenujemo tudi vezja s pomnilnikom ali vezja z dodano funkcijo pomnjenja. V takem vezju so izhodi odvisni od trenutnih vhodov in zgodovine vseh prejšnjih vhodov. Praktično lahko rečemo, da poljuben vhod vodi sekvenčno vezje preko majhnega števila edinstvenih konfiguracij, imenovanih stanja [2, 6].

Sekvenčno vezje je funkcija trenutne konfiguracije (stanja) vezja in vhodov ter jih preslika v naslednje stanje z novimi izhodi. Po določeni zakasnitvi postane novo stanje trenutno stanje in proces računanja naslednjih stanj in izhodov se ponovi. Sekvenčno logično vezje sestavljata:

- Odločitveno logično vezje, ki določa naslednja stanja in izhode.
- Pomnilno vezje sestavljeno iz ene ali več pomnilnih celic, kjer so shranjena stanja.

Sekvenčna logična vezja glede na delovanje delimo na:

- Sinhronska, kjer se stanja spreminjajo ob istem času. Ta čas je določen s periodičnim signalom imenovanim ura u (Slika 10.1).



Slika 10.1: Sinhronsko sekvenčno vezje

- Asinhronska, kjer se stanja spreminjajo s prihajajočimi vhodi v poljubnih časih.

V številnih napravah, ki jih uporabljamo pri svojem delu, se nahajajo sekvenčna vezja. Omenimo nekaj naprav, kot so: digitalna ura z alarmom, registri in števci v računalniku, krmilnik avtomata za tople napitke, krmilnik semaforja v cestnem križišču, in druge.

10.1 Pomnilne celice

Pomnilna celica je element, ki shranjuje en bit informacije. Vsaka celica ima izhod $Q(t)$ in negiran izhod $\bar{Q}(t)$, tako da imamo vedno prisotni obe vrednosti shranjene informacije. Delovanje pomnilne celice je določeno s krmilnimi vhodi in trenutnim izhodom ali

trenutnim stanjem $Q(t)$. V času $t+1$ dobimo zakasnjen izhod $Q(t+1)$, ki ga imenujemo naslednje stanje celice.

10.1.1 Tipi pomnilnih celic

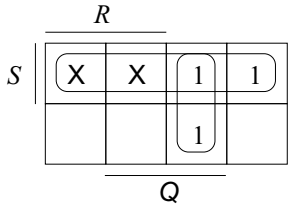
Izhodna funkcija pomnilne celice $Q(t+1)$ je odvisna od krmilnih vhodov, trenutnega izhoda in urinega signala, kadar je sinhronska. Za pomnilno celico podamo njeno delovanje v karakteristični tabeli oziroma s pomnilno enačbo, ki predstavlja logični zapis sekvenčnega izhoda $Q(t+1)$.

Glede na krmilne vhode poznamo štiri standardne tipe sinhronskih pomnilnih celic:

1) RS pomnilna celica

Tabela 10.1: Karakteristična tabela RS pomnilne celice

R	S	$Q(t+1)$	Pogoj
0	0	$Q(t)$	
0	1	1	
1	0	0	
1	1	$\times$	$RS=0$



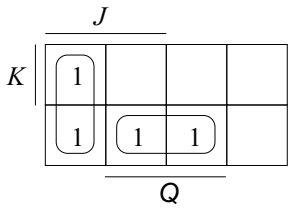
Pomnilna enačba RS pomnilne celice je

$$Q(t+1) = \bar{R}Q(t) \vee S, pogoj : R.S = 0 \ . \tag{10.1}$$

2) JK pomnilna celica

Tabela 10.2: Karakteristična tabela JK pomnilne celice

J	K	$Q(t+1)$
0	0	$Q(t)$
0	1	0
1	0	1
1	1	$\bar{Q}(t)$



Pomnilna enačba JK pomnilne celice je

$$Q(t+1) = \bar{K}Q(t) \vee J\bar{Q}(t) \ . \tag{10.2}$$

3) D pomnilna celica

Tabela 10.3: Karakteristična tabela D pomnilne celice

D	$Q(t+1)$
0	0
1	1

Pomnilna enačba D pomnilne celice je

$$Q(t+1) = D \ . \tag{10.3}$$

4) T pomnilna celica

Tabela 10.4: Karakteristična tabela T pomnilne celice

T	$Q(t+1)$
0	$Q(t)$
1	$\bar{Q}(t)$

Pomnilna enačba T pomnilne celice je

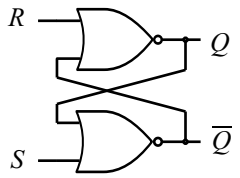
$$Q(t+1) = \bar{T}Q(t) \vee T\bar{Q}(t) \ . \tag{10.4}$$

10.1.2 RS pomnilna celica

Jedro vsake pomnilne celice (ang. flip-flop) predstavljata dvoja povratno vezana vrata NAND ali NOR. Povratno vezana vrata določajo RS pomnilno celico, z dvema krmilnima vhodoma (R, S). Krmilni vhod $R = 1$ briše pomnilno celico ($Q(t+1) = 0$), krmilni vhod $S = 1$ pa postavi celico ($Q(t+1) = 1$).

Realizacija RS pomnilne celice z NOR operatorji

Povratna vezava dveh NOR vrat določa RS pomnilno celico (Slika 10.2). Za NOR vrata je značilno, da že en vhod po vrednosti 1 daje izhod 0, zato imamo pri izhodu $Q(t)$ vhod R, pri izhodu $\bar{Q}(t)$ pa vhod S.



Slika 10.2: RS pomnilna celica - NOR operatorji

Izhod $Q(t+1)$, ki predstavlja novo naslednje stanje pomnilne celice, je podan v odvisnosti od vseh možnih vhodnih kombinacij vhodov R, S in trenutnega stanja pomnilne celice $Q(t)$ (Tabela 10.5). Krmilna kombinacija, ki se na vhodu pomnilne celice nikoli ne sme pojaviti, je označena s $\times$ in pomeni prepovedano kombinacijo.
Vpliv trenutnega stanja celice $Q(t)$ na naslednje stanje $Q(t+1)$ zapišemo krajše v karakteristični tabeli (Tabela 10.6). Če sta vhoda pomnilne celice $R = S = 1$, je delovanje celice neopredeljeno, ker imamo na obeh izhodih naslednje stanje 0, kar je v nasprotju z definicijo izhodov. Ta vhodna kombinacija je pri RS pomnilni celici prepovedana in jo označimo z $\times$ ter pripišemo pogoj, da mora veljati $RS = 0$.

Tabela 10.5: RS pomnilna celica

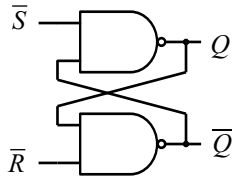
R	S	$Q(t)$	$Q(t+1)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	×
1	1	1	×

Tabela 10.6: Karakteristična tabela RS pomnilne celice

R	S	$Q(t+1)$	Pogoj
0	0	$Q(t)$	
0	1	1	
1	0	0	
1	1	×	$RS = 0$

Realizacija RS pomnilne celice z NAND operatorji

Vhoda v povratni vezavi dveh NAND operatorjev (Slika 10.3) sta ravno negirana v primerjavi s funkcijo pri vezavi NOR operatorjev, zato dobimo $\bar{R}\bar{S}$ pomnilno celico.



Slika 10.3: RS pomnilna celica - NAND operatorji

Tabela 10.7: Negirana RS pomnilna celica

$\bar{R}$	$\bar{S}$	$Q(t)$	$Q(t+1)$
0	0	0	×
0	0	1	×
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Za NAND vrata velja, da en vhod po vrednosti 0 daje izhod 1, zato imamo pri izhodu $Q(t)$ vhod $\bar{S}$, ki celico postavi in pri izhodu $\bar{Q}(t)$ vhod $\bar{R}$.

Vpliv trenutnega stanja celice $Q(t)$ na naslednje stanje $Q(t+1)$ zapišemo krajše v karakteristični tabeli (Tabela 10.8). Če sta vhoda pomnilne celice $\bar{R} = \bar{S} = 0$, je delovanje celice neopredeljeno, ker imamo na obeh izhodih naslednje stanje 1, kar je v nasprotju z definicijo izhodov. Ta vhodna kombinacija je pri RS pomnilni celici prepovedana in jo označimo z $\times$ ter pripišemo pogoj, da mora veljati $\bar{R} \vee \bar{S} = 1$.

Tabela 10.8: Karakteristična tabela negirane RS pomnilne celice

$\bar{R}$	$\bar{S}$	$Q(t+1)$	Pogoj
0	0	$\times$	$\overline{R} \vee \overline{S} = 1$
0	1	0	
1	0	1	
1	1	$Q(t)$	

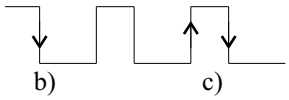
10.1.3 Sinhronske pomnilne celice

V praksi so pomnilne celice povezane z urinim signalom (u), zato jih imenujemo sinhronske. Svoje stanje (izhod) spremenijo samo takrat, kadar je na vhodu prisotna ustrezna fronta urinega signala. Urin signal je periodičen signal pravokotne oblike, ki ima definirani dve fronti. Prednja fronta je določena ob prehodu signala iz 0 v 1 in zadnja fronta ob prehodu signala iz 1 v 0. Tipi sinhronizacije v pomnilnih celicah so (Slika 10.4):

a) prednja fronta

b) zadnja fronta

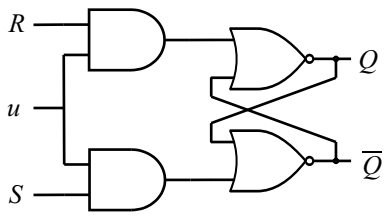
c) Master-Slave (impulz)



Slika 10.4: Tipi sinhronizacije

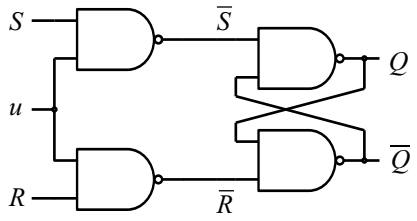
Iz osnovne strukture pomnilnih celic dobimo sinhronsko pomnilno celico tako, da dodamo na vhode povratno vezanih NAND ali NOR vrat dodaten nivo operatorjev, kamor je pripeljan urin signal u.

Primer: Pri povratni vezavi NOR vrat dodamo na vhode vrata AND, tako da je en vhod v konjunkciji urin signal u, ki s prednjo fronto sinhronizacije spreminja stanje celice, drugi vhod vrat pa je krmilni signal R oziroma krmilni signal S (Slika 10.5).



Slika 10.5: Sinhronska RS pomnilna celica z NOR povratno vezavo

Primer: Pri povratni vezavi NAND vrat dodamo na vhode vrata NAND tako, da je en vhod vrat urin signal u, ki s prednjo fronto sinhronizacije spreminja stanje celice, na drugem pa sta krmilni vhod R oziroma krmilni vhod S. Z uporabo negirane konjunkcije (NAND) smo spremenili negaciji vhodov povratne vezave ($\bar{R}$ $\bar{S}$) v RS sinhronsko pomnilno celico. (Slika 10.6).



Slika 10.6: Sinhronska RS pomnilna celica z NAND povratno vezavo

10.1.4 Vhodne funkcije pomnilnih celic

Pri načrtovanju sekvenčnih logičnih vezij običajno poznamo trenutno stanje in naslednje stanje pomnilne celice. Na osnovi prehodov med stanji nas zanimajo vhodne funkcije pomnilne celice, ki te prehode zagotavljajo. Zapisali jih bomo v **vzbujevalni tabeli**, ki nam pove, kako krmiliti ali vzbujevati vhode pomnilnih celic.

Tabela 10.9: Vzbujevalna tabela RS, JK, D, T pomnilnih celic

$Q(t)$	$Q(t+1)$	R	S	J	K	D	T
0	0	×	0	0	×	0	0
0	1	0	1	1	×	1	1
1	0	1	0	×	1	0	1
1	1	0	×	×	0	1	0

Vzbujevalne tabele pomnilnih celic dobimo tako, da na levi strani tabele vpišemo poznan trenutni izhod $Q(t)$ in željeni naslednji izhod $Q(t+1)$. Na desni strani pa s pomočjo karakteristične tabele pomnilne celice določimo krmilne vhode, ki zagotavljajo podane prehode. Zapisane imamo vzbujevalne tabele za RS, JK, D in T pomnilne celice (Tabela 10.9).

Pri RS in JK pomnilnih celicah imamo redundančne kombinacije × pri nekaterih krmilnih vhodih. Tu dve krmilni kombinaciji zagotovita prehod iz trenutnega v željeno stanje, zato nam zadostuje en krmilni vhod z opredeljeno vrednostjo 0 ali 1, drugi pa je redundančen. Vhodne funkcije uporabljamo pri načrtovanju različnih sekvenčnih vezij, kot so: sinhronske pomnilne celice z različnimi operatorji, registri, števci in avtomati.

Primer: Realizirajmo sinhronsko D pomnilno celico z NAND operatorji.

Za osnovo vzemimo povratno vezavo NAND operatorjev, kjer imamo vhoda $\bar{R}$ in $\bar{S}$. V binarno aplikacijsko tabelo vpišemo delovanje D pomnilne celice ($Q(t+1) = D$), ki jo želimo realizirati. Iz vzbujevalne tabele RS pomnilne celice za poznane prehode iz $Q(t)$ v $Q(t+1)$ določimo vhodni funkciji $\bar{R}$, $\bar{S}$ (Tabela 10.10) in ju zapišemo v minimalni obliki z operatorji NAND.

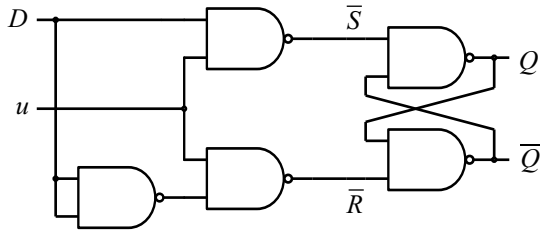
Tabela 10.10: Aplikacijska tabela D pomnilne za $\bar{R}$, $\bar{S}$

D	$Q(t)$	$Q(t+1)$	$\bar{R}$	$\bar{S}$
0	0	0	×	1
0	1	0	0	1
1	0	1	1	0
1	1	1	1	×

Redundanca × naj zavzame vrednost 0 in vhodni funkciji $\bar{R}$ in $\bar{S}$ sta enaki vhodu D ali $\bar{D}$. Ker želimo zgraditi sinhronsko D pomnilno celico, ji dodamo v konjunktivni povezavi urin signal u in zapišemo vhodni funkciji R in S z 2-vhodnimi NAND operatorji.

$$\bar{R} = D = D \vee \bar{u} = \bar{D} \uparrow u$$
$$\bar{S} = \bar{D} = \bar{D} \vee \bar{u} = D \uparrow u$$

Na vhoda $\bar{R}$ in $\bar{S}$ povratni vezavi dodamo en nivo NAND vrat s krmilnim vhodom D oziroma $\bar{D}$ pomnilne celice in z urinim signalom u, ki s prednjo fronto sinhronizacije spreminja stanje celice (Slika 10.7).



Slika 10.7: D pomnilna celica z NAND operatorji

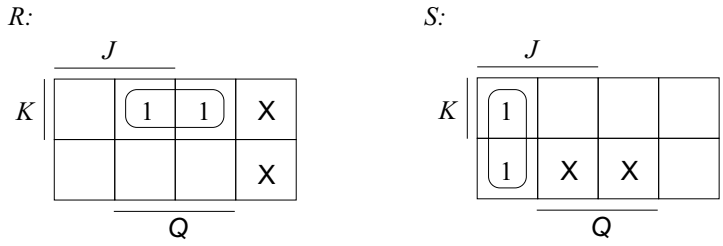
Primer: Oglejmo si izvedbo sinhronske JK pomnilne celice z NOR operatorji.

Za osnovo vzamemo povratno vezavo NOR operatorjev z vhodoma R in S. Pomnilno enačbo za JK pomnilno celico $Q(t+1) = \bar{K}Q(t) \vee J\bar{Q}(t)$ vpišemo v binarno aplikacijsko tabelo in določimo vhodni funkciji za R, S (Tabela 10.11).

Tabela 10.11: Aplikacijska tabela JK pomnilne celice za R, S

J	K	$Q(t)$	$Q(t+1)$	R	S
0	0	0	0	×	0
0	0	1	1	0	×
0	1	0	0	×	0
0	1	1	0	1	0
1	0	0	1	0	1
1	0	1	1	0	×
1	1	0	1	0	1
1	1	1	0	1	0

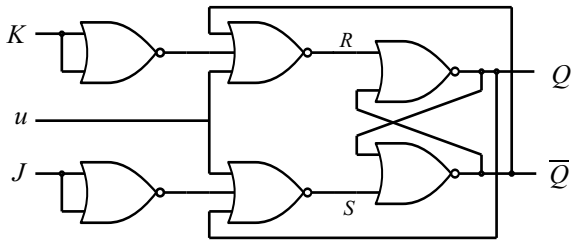
Vhodni funkciji R in S minimiziramo (Slika 10.8) in zapišemo z NOR operatorji.



Slika 10.8: Minimizacija R, S vhodnih funkcij

V minimizacijskem postopku dobimo rešitev za 2-vhodna NOR vrata. Pomnilni celici dodamo še sinhronski vhod u tako, da imamo potem na predhodnem nivoju 3-vhodna NOR vrata (Slika 10.9) z negiranima krmilnima vhodoma, ki ju še enkrat negiramo. Za sinhronizacijo pomnilne celice je uporabljena zadnja fronta.

$$\begin{aligned} R &= KQ(t) = \overline{\bar{K} \vee \bar{Q}(t)} = \bar{K} \downarrow \bar{Q}(t) \\ S &= J\bar{Q}(t) = \overline{\bar{J} \vee Q(t)} = \bar{J} \downarrow Q(t) \end{aligned}$$



Slika 10.9: JK pomnilna celica z NOR operatorji

Načrtovanje sinhronskih pomnilnih celic

Vsako pomnilno celico lahko izvedemo kot kombinacijsko vezje v povezavi s pomnilno celico drugega tipa. Kot primer navedimo možnost, da lahko D pomnilno celico realiziramo z JK pomnilno celico ali obratno z JK pomnilno celico realiziramo D pomnilno celico, itd. Sinhronska JK pomnilna celica deluje kot T pomnilna celica, če oba vhoda povežemo skupaj in ju uporabimo kot en krmilni vhod.

Primer: Realizirajte sinhronsko T pomnilno celico z uporabo sinhronske RS pomnilne celice in konjunktivnih logičnih vrat.

Zapišemo binarno aplikacijsko tabelo za T pomnilno celico z vhodnima spremenljivkama T in $Q(t)$. Določimo vhodni funkciji R in S za pomnilno celico (Tabela 10.12).

Tabela 10.12: Aplikacijska tabela T pomnilne celice

T	$Q(t)$	$Q(t+1)$	R	S
0	0	0	×	0
0	1	1	0	×
1	0	1	0	1
1	1	0	1	0

Redundanci × lahko zavzameta različni vrednosti:

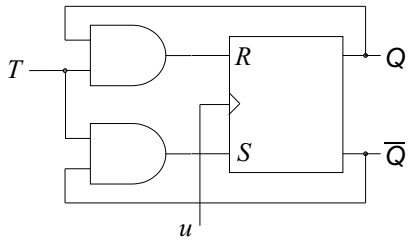
a) če je vrednost ×=0, potem je minimalna oblika vhodnih funkcij R in S

$$\begin{aligned} R &= TQ(t) \\ S &= T\bar{Q}(t) \ , \end{aligned}$$

b) če je ×=1, potem je

$$\begin{aligned} R &= T \equiv Q(t) = \overline{T} \nabla Q(t)) \\ S &= T \nabla Q(t) \ . \end{aligned}$$

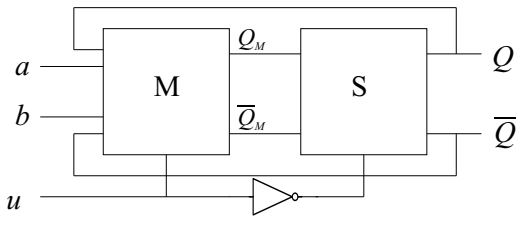
V rešitvi za realizacijo T pomnilne celice smo uporabili rešitev s sinhronsko RS pomnilno celico in logičnimi vrati AND (Slika 10.10).



Slika 10.10: Izvedba T pomnilne celice z RS in AND vrati

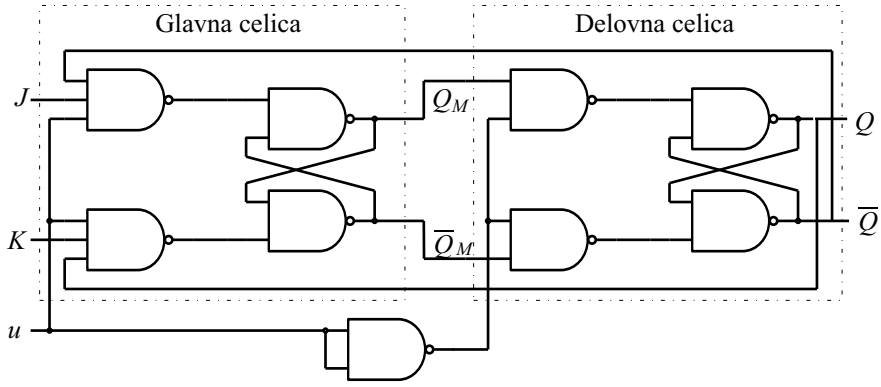
10.1.5 Pomnilne celice s predpomnjenjem

Pri sinhronizaciji pomnilnih celic s prednjo ali zadnjo fronto urinega signala u se v realnih razmerah lahko pojavijo problemi z zakasnitvijo ali prehitevanjem krmilnih signalov in z motnjami urinega signala (zelo kratka nihanja v napetosti). Za zagotavljanje stabilnih prehodov stanj v sekvenčnih vezjih so zgradili celice s predpomnjenjem (ang. Master-Slave-MS). Pomnilna celica s predpomnjenjem je zgrajena iz dveh pomnilnih celic: glavne celice (ang. Master-M) in delovne celice (ang. Slave-S) in ima delovanje pomnilnega elementa definirano v dveh fazah (Slika 10.12). V prvi fazi glavna celica sprejme nove vhode in v povezavi s trenutnim izhodom (stanjem) definira izhoda Q_M in $\bar{Q}_M$ ob prvi fronti ure in jih odda delovni celici. V drugi fazi delovna celica sprejme izhoda Q_M in $\bar{Q}_M$ kot vhoda in ob zadnji fronti ure spremeni izhoda Q in $\bar{Q}$.



Slika 10.11: Celica s predpomnjenjem

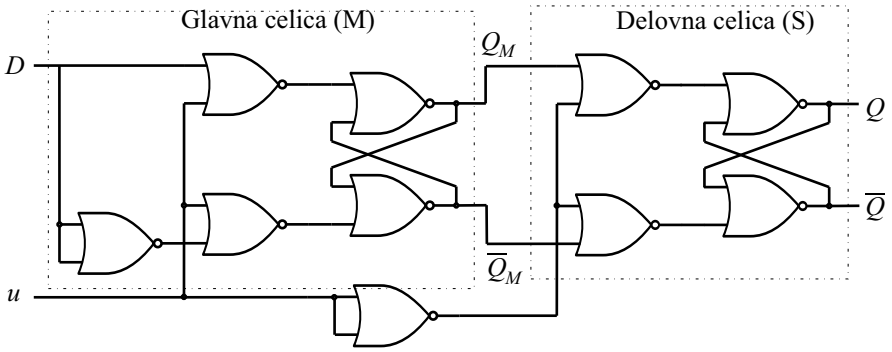
Primer: Zgradimo sinhronsko JK pomnilno celico s predpomnjenjem z uporabo NAND vrat. Uporabimo dve sinhronski JK pomnilni celici zgrajeni iz NAND logičnih vrat in ju povežemo med seboj. Glavna celica sprejme krmilna vhoda J in K, urin signal u in izhoda Q in $\bar{Q}$ iz delovne celice. Izhoda Q_M in $\bar{Q}_M$ glavne celice gresta na vhoda delovne celice, ki sprejme tudi negiran urin signal $\bar{u}$. Slika 10.12 prikazuje logično vezje JK pomnilne



Slika 10.12: JK pomnilna celica s predpomnjenjem

celice s predpomnjenjem z logičnimi vrati NAND.

Primer: Za D pomnilno celico s predpomnjenjem uporabimo NOR operatorje. Pri načrtovanju najprej zgradimo sinhronsko pomnilno D z zahtevanimi logičnimi vrati (NOR). Dobljeno logično shemo uporabimo za glavno in delovno celico, ju zaporedno med seboj povežemo in pripeljemo urin signal u na glavno celico in negiran urin signal $\bar{u}$ na delovno celico (Slika 10.13).



Slika 10.13: D pomnilna celica s predpomnjenjem

TTL pomnilne celice

V TTL tehnologiji imamo JK in D pomnilne celice:
7473 - vsebuje dve JK pomnilni celici z asinhronskim vhodom za brisanje celice. Signal $\overline{CLR}$ je aktiven z niskim nivojem in postavi stanje celice $Q(t) = 0$ neodvisno od trenutnih vrednosti vhodov in urinega signala. Standardna JK pomnilna celica se v čipu pojavlja v izvedbi celice s predpomnjenjem. Na voljo pa imamo verzijo "A"pomnilne celice s sinhronizacijo ob zadnji fronti urinega signala.
7474 - vsebuje dve D pomnilni celici z asinhronskima vhodoma za brisanje in postavljanje celice. Signal $\overline{PR}$ je aktiven z nizkim nivojem in postavi stanje celice $Q(t) = 1$, signal $\overline{CLR}$ je aktiven z nizkim nivojem in postavi stanje celice $Q(t) = 1$. V celici je uporabljena sinhronizacija s prednjo fronto urinega signala.

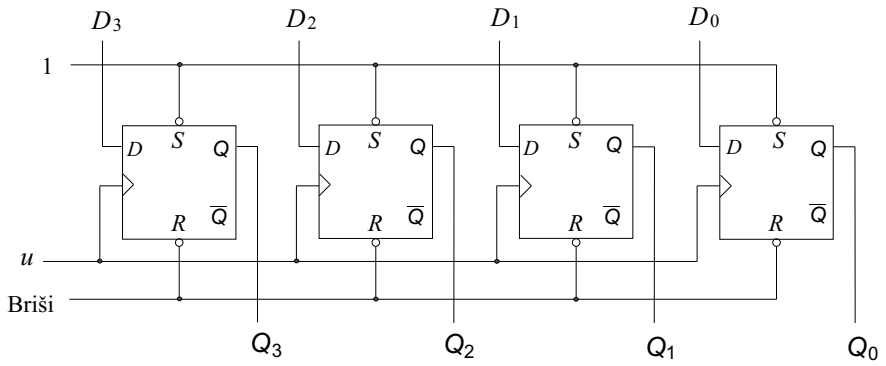
10.2 Registri

Sekvenčno vezje, ki je sestavljeno iz skupine pomnilnih celic in služi za shranjevanje podatkov in izvajanje določenih operacij nad podatki, imenujemo register. Glede na način delovanja registre delimo na:

- shranjevalne registre,
- pomikalne registre,
- splošne ali univerzalne registre.

Shranjevalni register

Shranjevalni register ima funkcijo shranjevanja na vhodu sprejetih podatkov. Zgradimo ga tako, da združimo toliko D pomnilnih celic kot je število zahtevanih bitov v registru. Slika 10.14 prikazuje 4-bitni register zgrajen iz D pomnilnih celic, ki imajo poleg sinhronskega D vhoda še asinhronska vhoda za brisanje (R) in postavljanje celice (S).



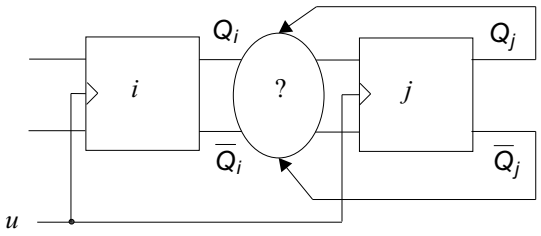
Slika 10.14: 4-bitni shranjevalni register

Vhoda postavita novo stanje celice neodvisno od ure z nizkim aktivnim signalom, to je logična 0, kar je razvidno iz negiranega vhoda na pomnilni celici. Vsaka pomnilna celica na D vhodu sprejme en bit informacije, ki se ob pojavu urine fronte shrani in je prisotna na izhodu toliko časa, dokler se vanjo ne vpiše nova vsebina. Predstavljeni shranjevalni register je na voljo v TTL čipu z oznako 74171.

Pomikalni register

Pomikalni register izvede pomik prejšnje vsebine registra in jo ponovno shrani. V registrih poznamo pomik desno ali levo in ciklični pomik desno ali levo. Pomik vsebine iz ene celice v drugo izvedemo tako, da krmilimo vpis v celico z izhodom tiste celice, ki predaja vsebino. V ta namen za vsako pomnilno celico izračunamo vhodne funkcije na osnovi opisanih prehodov vsebine. Za D pomnilno celico je vhodna funkcija določena kar z izhodom celice iz katere sprejema vsebino. Drugače pa je pri RS, JK in T pomnilnih celicah.

Oglejmo si izvedbo pomika desno (Slika 10.15) iz celice Q_i v Q_j in izračunajmo vhodne funkcije za D, RS, JK in T pomnilno celico (Tabela 10.13).



Slika 10.15: Pomik desno iz celice Q_i v Q_j

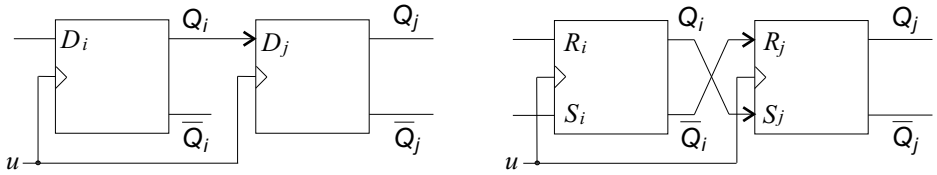
Tabela 10.13: Vhodne funkcije D, RS, JK in T celice za pomik desno

$Q_i(t)$	$Q_j(t)$	$Q_j(t+1)$	D	R_j	S_j	J_j	K_j	T_j
0	0	0	0	×	0	0	×	0
0	1	0	0	1	0	×	1	1
1	0	1	1	0	1	1	×	1
1	1	1	1	0	×	×	0	0

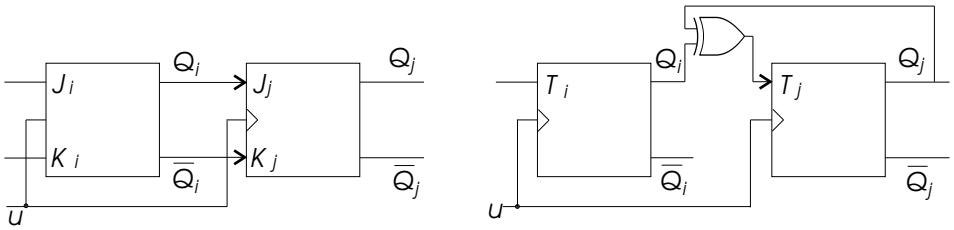
Redundančne vrednosti × opredelimo z 0 ali 1, tako da dobimo čim bolj enostavne vhodne funkcije, ki so za pomik desno iz celice i v celico j naslednje:

$$\begin{aligned} D_j &= Q_i(t) \\ R_j &= \bar{Q}_i(t), S_j = Q_i(t) \\ J_j &= Q_i(t), K_j = \bar{Q}_i(t) \\ T_j &= Q_i(t) \nabla Q_j(t) \ . \end{aligned}$$

Za D, RS in JK pomnilne celice je potrebno samo pravilno povezati izhode i-te celice na vhode j-te celice (Slika 10.16 in Slika 10.17.c). Pri uporabi T pomnilne celice za pomik potrebujemo XOR logična vrata (Slika 10.17.d).



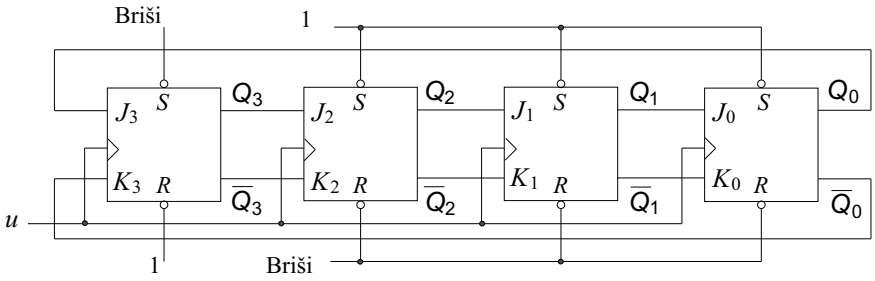
Slika 10.16: Pomik desno iz celice Q_i v Q_j: a)D celica, b)RS celica



Slika 10.17: Pomik desno iz celice Q_i v Q_j: c)JK celica, d)T celica

Za pomik levo obrnemo mesti celic i in j, dobimo enake vhodne funkcije in pomnilne celice povežemo v drugi smeri.

Oglejmo si vezje za izvedbo desnega pomika v 4-bitnem cikličnem pomikalnem registru realiziranem z JK pomnilnimi celicami. JK pomnilne celice imajo asinhronska vhoda R in S za nastavitev začetnih vrednosti registra. Vsebina registra naj se nastavi z vhodnim signalom *Briši=0*, ki postavi celico z izhodom Q_3 v stanje 1, ostale celice pa v stanje 0. Ko postavimo signal *Briši=1* se enica iz v registru v odvisnosti od urinega signala ciklično pomika desno (Slika 10.18).



Slika 10.18: 4-bitni ciklični pomikalni register desno

Univerzalni register

Univerzalni register ima lahko več načinov delovanja, kot so shranjevanje, pomik, ohranjanje vsebine, idr. Funkcija delovanja je izbrana z zunanjimi krmilnimi vhodi, ki jih pri načrtovanju registra kodiramo v krmilne spremenljivke.

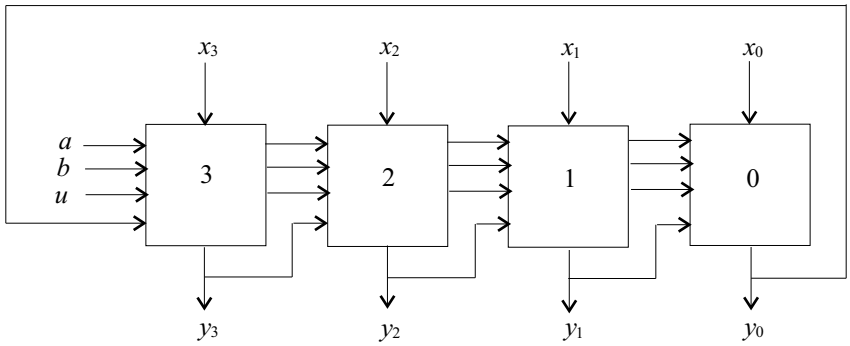
Primer: Oglejmo si, kako bi zgradili 4-bitni splošni register $Y = (y_3, y_2, y_1, y_0)$ z naslednjimi funkcijami, če ne uporabimo asinhronskih vhodov za postavljanje in brisanje pomnilnih celic:

- vpis nove vsebine z vhodov: $Y(t+1) = X$, če je $X = (x_3, x_2, x_1, x_0)$
- postavljanje registra: $Y(t+1) = 1$
- brisanje registra: $Y(t+1) = 0$
- ciklični pomik desno: $Y(t+1) = (y_0(t), y_3(t), y_2(t), y_1(t))$

Register mora imeti štiri različne funkcije, ki jih kodiramo z dvema krmilnima vhodnima spremenljivkama a in b:

a	b	*Operacija*
0	0	VPIS
0	1	SET
1	0	RESET
1	1	CPD

Za zapis binarne aplikacijske tabele delovanja registra moramo določiti vse vhodne spremenljivke, ki vplivajo na izhode (Slika 10.19). Celoten register ima naslednje vhodne spremenljivke: krmilni spremenljivki a in b, štiri zunanje vhode x_3, x_2, x_1, x_0 in štiri trenutne izhode y_3, y_2, y_1, y_0. Z opisanimi desetimi vhodnimi spremenljivkami postane zapis splošnega 4-bitnega registra zelo obsežen, zato skušamo problem poenostaviti. Vsi biti v registru imajo enako število funkcij in enako število vhodov, razlikujejo se samo njihovi indeksi, zato pristopimo k reševanju naloge tako, da opazujemo samo en bit registra in



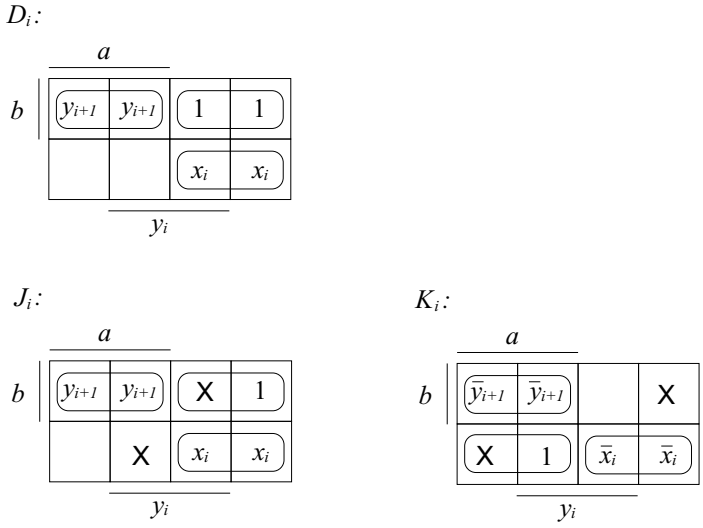
Slika 10.19: Blok shema 4-bitnega splošnega registra

izračunane funkcije posplošimo na preostale.
V binarni aplikacijski tabeli upoštevamo vhodni spremenljivki a in b, vhod x_i in izhod y_i kot časovno spremenljivko s trenutnim izhodom $y_i(t)$ in naslednjim izhodom $y_i(t+1)$ (Tabela 10.14). Iz opisanega delovanja i-tega bita registra zapišemo binarno aplikacijsko tabelo in vhodne funkcije za D in JK pomnilno celico.

Tabela 10.14: Binarna aplikacijska tabela pomnilne celice i

a	b	$y_i(t)$	$y_i(t+1)$	D_i	J_i	K_i
0	0	0	x_i	x_i	x_i	$\bar{x}_i$
0	0	1	x_i	x_i	x_i	$\bar{x}_i$
0	1	0	1	1	1	$\times$
0	1	1	1	1	$\times$	0
1	0	0	0	0	0	$\times$
1	0	1	0	0	$\times$	1
1	1	0	y_{i+1}	y_{i+1}	y_{i+1}	$\bar{y}_{i+1}$
1	1	1	y_{i+1}	y_{i+1}	y_{i+1}	$\bar{y}_{i+1}$

Minimiziramo vhodne funkcije za D in JK pomnilni celici pri i-tem bitu (Slika 10.20).



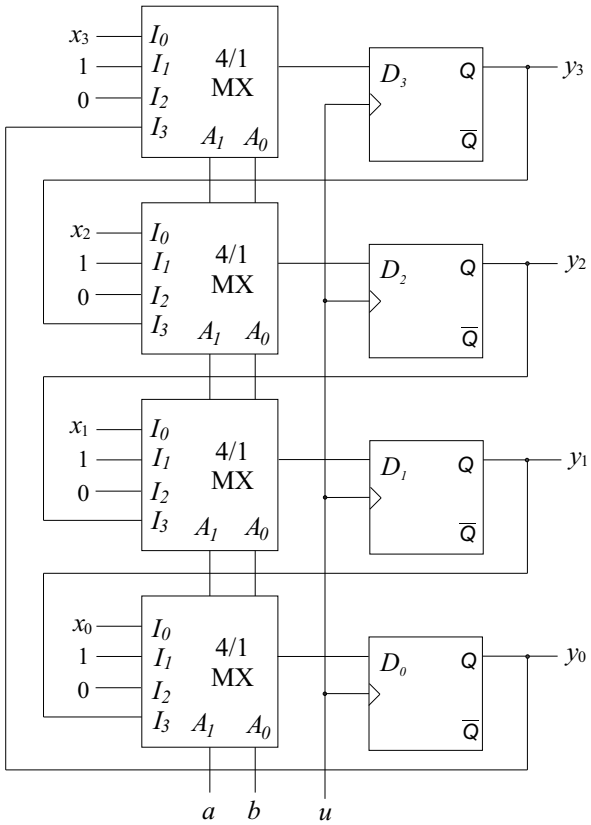
Slika 10.20: Minimizacija vhodnih funkcij D_i, J_i, K_i

$$D_i = \bar{a}\bar{b}x_i \vee \bar{a}b \vee aby_{i+1}$$
$$J_i = \bar{a}\bar{b}x_i \vee \bar{a}b \vee aby_{i+1}$$
$$K_i = \bar{a}\bar{b}\bar{x}_i \vee a\bar{b} \vee ab\bar{y}_{i+1}$$

Minimalne oblike vhodnih funkcij uporabimo za zapis vseh štirih bitov registra, tako da za indeks $i = 3, 2, 1, 0$ zapišemo štiri funkcije in zgradimo logično vezje. Za D pomnilno celico zapišimo vse štiri vhodne funkcije:

$$D_3 = \bar{a}\bar{b}x_3 \vee \bar{a}b \vee aby_0$$
$$D_2 = \bar{a}\bar{b}x_2 \vee \bar{a}b \vee aby_3$$
$$D_1 = \bar{a}\bar{b}x_1 \vee \bar{a}b \vee aby_2$$
$$D_0 = \bar{a}\bar{b}x_0 \vee \bar{a}b \vee aby_1$$

Za realizacijo univerzalnega registra je najbolje uporabiti 2-naslovne multiplekserje, ker imamo vhodne funkcije že razčlenjene po krmilnih spremenljivkah, ki jih pripeljemo na naslovna vhoda. Na podatkovnih vhodih so zunanji vhodi x_i, trenutna stanja izhodov y_i in konstanti 0 in 1 (Slika 10.21).



Slika 10.21: Logična shema 4-bitnega univerzalnega registra

10.3 Števci

Števci so sekvenčna vezja, ki izvedejo določeno sekvenco stanj v povezavi z urinim signalom.

Delovanje števca delimo glede na:

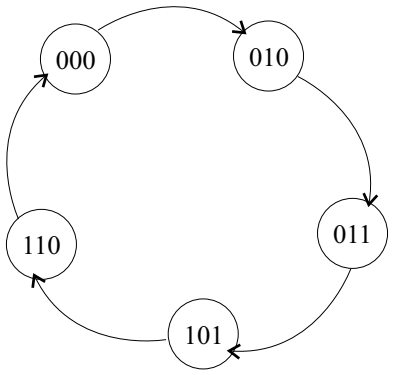
- način štetja
  - povečevanje (inkrement) vrednosti
  - zmanjševanje (dekrement) vrednosti,
- velikost števca
  - podamo število bitov (2-bitni števec, 3-bitni števec, ...)
  - modul štetja M (M=8 - števec po modulu osem). Modul štetja pomeni, da ima števec na izhodu vsa stanja, ki predstavljajo cifre od 0 do 7.

Števci so zelo pomembni v digitalnih vezjih. Uporabljamo jih za štetje dogodkov, kot je število urinih impulzov v določenem času (merjenje frekvence), za shranjevanje podatkov v digitalni uri, za sekvenčno naslavljanje pomnilnika in v aritmetičnih vezjih.

Postopek gradnje števca

- Iz opisa števca narišemo diagram prehajanja stanj (DPS), ki prikazuje število stanj in njihovo sekvenco ter določimo število bitov za zapis stanj (n-bitni števec).
- Zapišemo binarno aplikacijsko tabelo delovanja števca tako, da opišemo spremembe iz trenutnega stanja v naslednje stanje z n biti. Vsak bit števca je predstavljen s časovno spremenljivko $Q_i(t)$ in $Q_i(t+1)$.
- Izberemo pomnilne celice za realizacijo n-bitnega števca in z vzbujevalno tabelo izbrane pomnilne celice zapišemo vhodne funkcije.
- Izbira kombinacijske logike za izvedbo vhodnih funkcij, povezava s pomnilnimi celicami in risanje logičnega vezja.

Na sliki 10.22 je prikazan diagram prehajanja stanj 3-bitnega števca. Oglejmo si njegovo realizacijo s T pomnilnimi celicami.



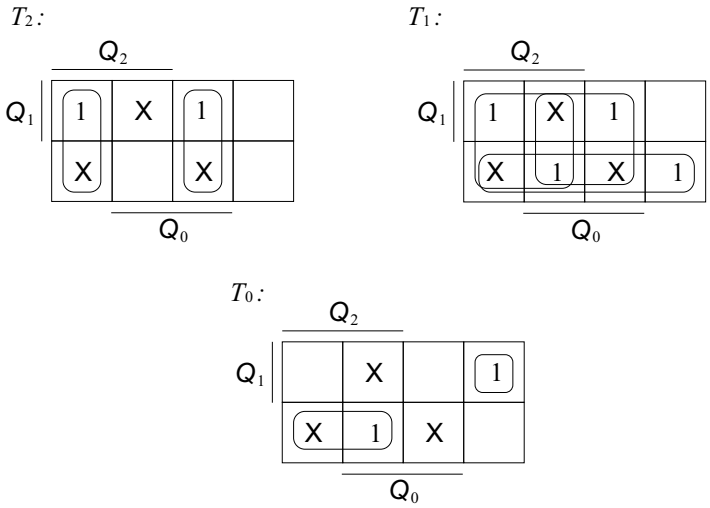
Slika 10.22: DPS za 3-bitni števec

Podani števec ima določenih samo pet stanj in prehodov med stanji, zato bomo preostala tri obravnavali kot redundančna stanja, ker se pri normalnem delovanju števca nikoli ne bodo pojavila. Zapišemo binarno aplikacijsko tabelo s tremi časovnimi spremenljivkami Q_2, Q_1, Q_0 v času t in določimo prehode v čas $t+1$ iz podanega diagrama prehajanja stanj. Za vpisano delovanje števca nato na desni strani zapišemo vhodne funkcije za T pomnilne celice (Tabela 10.15).

Tabela 10.15: Aplikacijska tabela 3-bitnega števca

$Q_2(t)$	$Q_1(t)$	$Q_0(t)$	$Q_2(t+1)$	$Q_1(t+1)$	$Q_0(t+1)$	T_2	T_1	T_0
0	0	0	0	1	0	0	1	0
0	0	1	×	×	×	×	×	×
0	1	0	0	1	1	0	0	1
0	1	1	1	0	1	1	1	0
1	0	0	×	×	×	×	×	×
1	0	1	1	1	0	0	1	1
1	1	0	0	0	0	1	1	0
1	1	1	×	×	×	×	×	×

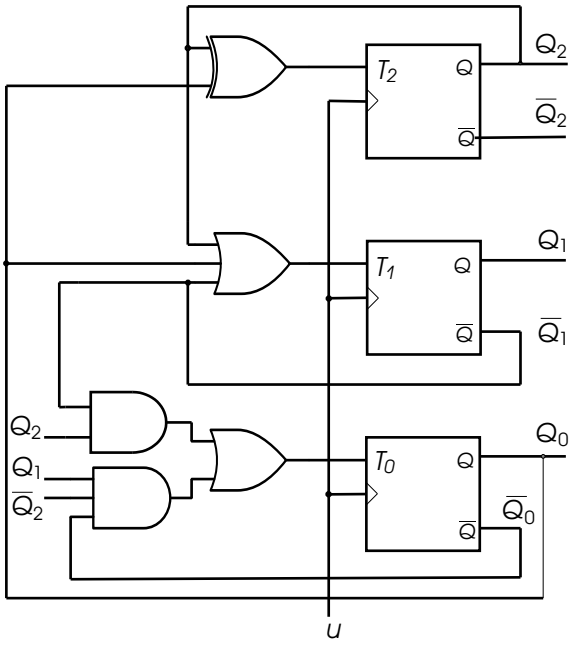
Vhodne funkcije, ki smo jih zapisali za T pomnilne celice minimiziramo (Slika 10.23).



Slika 10.23: Minimizacija vhodnih funkcij T_2, T_1, T_0

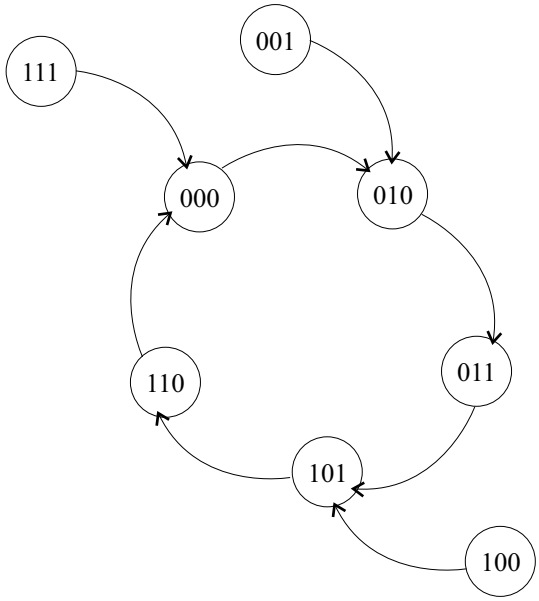
$$
\begin{aligned}
T_2 &= Q_2\bar{Q}_0 \vee \bar{Q}_2Q_0 = Q_2 \bigtriangledown Q_0 \\
T_1 &= Q_2 \vee \bar{Q}_1 \vee Q_0 \\
T_0 &= \bar{Q}_1Q_2 \vee \bar{Q}_2Q_1\bar{Q}_0
\end{aligned}
$$

Z minimizacijo dobimo zapis vhodnih funkcij T_2, T_1, T_0 in jih uporabimo za realizacijo 3-bitnega števca z logičnimi vrati (Slika 10.24).
Delovanje opisanega števca je podano enoumno, če je začetno stanje 000. Kadar pa začetno stanje ni vnaprej določeno, se ob vključitvi števca postavi poljubno stanje, kar v našem primeru pomeni, da lahko pridemo tudi v nedoločeno stanje in sekvenca štetja ni več



Slika 10.24: Logična shema 3-bitnega števca

enoumna. Ta problem lahko razrešimo tako, da nedoločena stanja v času t prehajajo v prva naslednja stanja v času $t+1$ (Slika 10.25), ali pa nedoločena stanja prehajajo v stanje 000.



Slika 10.25: 3-bitni števec s popolnimi stanji

TTL registri in števci

V TTL tehnologiji imamo več različnih tipov registrov s podatkovnimi vhodi in izhodi ter krmilnimi in asinhronskimi vhodi, ki omogočajo različne načine delovanja:
74171 - vsebuje 4-bitni shranjevalni register z asinhronskim vhodom za brisanje celice ($\overline{CLR}$), ki je aktiven z nizkim nivojem in postavi stanje celice $Q(t) = 0$ neodvisno od tre-

nutnih vrednosti vhodov in urinega signala. Ob prednji fronti urinega signala se vsebina na D_i vhodih vpiše v register z izhodi Q_i in $\bar{Q}_i$.
74377 - vsebuje 8-bitni shranjevalni register s tristanjskimi izhodi. Enable vhod za omogočanje izhodov ($\overline{EN}$) je aktiven z nizkim nivojem in postavi stanje celice v visokoimpedančno stanje, ki odklopi izhode od vodila s katerim je register povezan.
74194 - je univerzalni 4-bitni pomikalni register z asinhronskim vhodom za brisanje celice ($\overline{CLR}$), krmilnima vhodoma S_1 in S_0 za izbiro funkcije. Na voljo imamo naslednje štiri funkcije: ohranjanje vrednosti, paralelen vpis vsebine, pomik desno in pomik levo. Za pomik desno imamo na voljo vhod RSI za določitev vrednosti najbolj pomembnega bita in za pomik levo vhod LSI za določitev vrednosti najmanj pomembnega bita.

Števec v TTL tehnologiji:
74163 - sinhronski 4-bitni števec s štirimi podatkovnimi vhodi in štirimi izhodi. Sinhronska krmilna signala sta $\overline{LOAD}$, ki vpiše podatke na vhodu v pomnilne celice in $\overline{CLR}$, ki briše števec. Oba signala morata biti prisotna pred prednjo fronto urinega signala. Za povečevanje števca morata biti aktivna signala $P = T = 1$. V primeru, da gre eden od obeh signalov v logično 0, se števec ustavi in ostane v trenutnem stanju. Števec generira prenos RCO, ki se postavi na 1 v trenutku, ko števec doseže največjo vrednost 1111_2.

10.4 Vaje

VAJA 10.4.1:

Realizirajte sinhronsko T pomnilno celico z uporabo NAND operatorjev.

Najprej definiramo karakteristično tabelo T pomnilne celice (Tabela 10.16), ki jo bomo zapisali v binarno aplikacijsko tabelo za določitev vhodnih funkcij.

Tabela 10.16: Karakteristična tabela T pomnilne celice

T	$Q(t+1)$
0	$Q(t)$
1	$\bar{Q}(t)$

Za osnovo vzamemo sinhronsko RS pomnilno celico z NAND operatorji, kjer imamo vhoda $\bar{R}$ in $\bar{S}$. V binarno aplikacijsko tabelo vpišemo delovanje T pomnilne celice in iz vzbujevalne tabele RS pomnilne celice za poznane prehode iz $Q(t)$ v $Q(t+1)$ določimo vhodni funkciji $\bar{R}$ in $\bar{S}$ (Tabela 10.17).

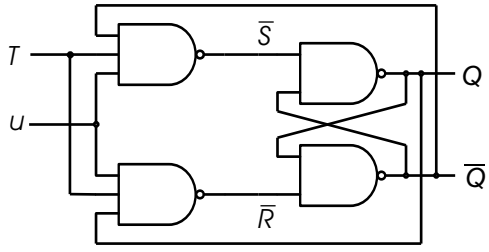
Tabela 10.17: Aplikacijska tabela T pomnilne celice za $\bar{R}$ in $\bar{S}$

T	$Q(t)$	$Q(t+1)$	$\bar{R}$	$\bar{S}$
0	0	0	$\times$	1
0	1	1	1	$\times$
1	0	1	1	0
1	1	0	0	1

Če redundanca $\times$ zavzame vrednost 1, je vhodna funkcija $\bar{R}$ enaka disjunkciji med $\bar{T}$ in $\bar{Q}(t)$. Vhodna funkcija za $\bar{S}$ pa je enaka disjunkciji med $\bar{T}$ in $Q(t)$. Obe funkciji zapišemo z NAND operatorji.

$$\bar{R} = \bar{T} \vee \bar{Q}(t) = \overline{TQ(t)} = T \uparrow Q(t)$$
$$\bar{S} = \bar{T} \vee Q(t) = \overline{T\bar{Q}(t)} = T \uparrow \bar{Q}(t)$$

Na vhoda $\bar{R}$ in $\bar{S}$ smo dodali dodaten nivo dveh 3-vhodnih NAND operatorjev. Krmilnemu vhodu T in stanju $Q(t)$ ali $\bar{Q}(t)$ je na tretjem vhodu logičnih vrat urin signal u (Slika 10.26).



Slika 10.26: T pomnilna celica z NAND operatorji

VAJA 10.4.2:

Realizirajte sinhronsko D pomnilno celico z uporabo NOR operatorjev.

Za osnovo vzamemo sinhronsko RS pomnilno celico z NOR operatorji, kjer imamo vhoda R in S. V binarno aplikacijsko tabelo vpišemo delovanje D pomnilne celice in iz vzbujevalne tabele RS pomnilne celice za poznane prehode iz $Q(t)$ v $Q(t+1)$ določimo vhodni funkciji R in S (Tabela 10.18).

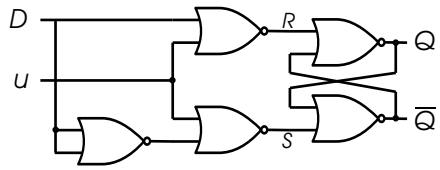
Tabela 10.18: Aplikacijska tabela D pomnilne celice za R in S povratno vezavo

D	$Q(t)$	$Q(t+1)$	R	S
0	0	0	×	0
0	1	0	1	0
1	0	1	0	1
1	1	1	0	×

Če redundanca × zavzame vrednost 1 sta vhodni funkciji

$$\begin{aligned} R &= \bar{D} \\ S &= D \end{aligned}$$

Ker želimo imeti sinhronsko D pomnilno celico z NOR operatorji bomo na vhoda R in S dodali dodaten nivo dveh 2-vhodnih NOR operatorjev in urin signal u. Vhodni funkciji za NOR povratno vezavo sta potem $R = D \downarrow u$ in $S = \bar{D} \downarrow u$ (Slika 10.27).



Slika 10.27: D pomnilna celica z NOR operatorji

VAJA 10.4.3:

Realizirajmo sinhronsko JK pomnilno celico z uporabo sinhronske RS pomnilne celice in 1- naslovnih multiplekserjev (2/1MX).

Zapišemo binarno aplikacijsko tabelo JK pomnilne celice z vhodnimi spremenljivkami J, K in trenutnim stanjem $Q(t)$. Izračunati moramo vhodni funkciji R in S za pomnilno celico, ki jo uporabimo v realizaciji (Tabela 10.19).

Tabela 10.19: Aplikacijska tabela JK pomnilne celice

J	K	$Q(t)$	$Q(t+1)$	R	S
0	0	0	0	×	0
0	0	1	1	0	×
0	1	0	0	×	0
0	1	1	0	1	0
1	0	0	1	0	1
1	0	1	1	0	×
1	1	0	1	0	1
1	1	1	0	1	0

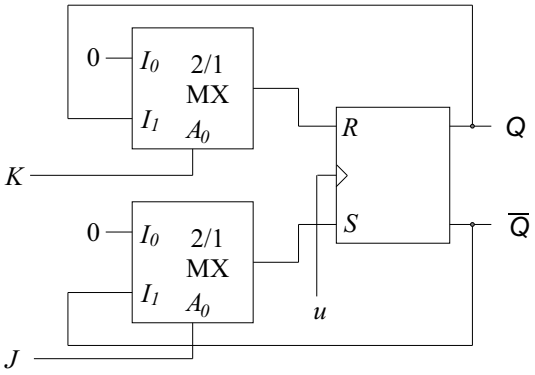
Poiščemo minimalno disjunktivno normalno obliko vhodnih funkcij R in S (Slika 10.8).

$$\begin{aligned} R &= KQ(t) \\ S &= J\bar{Q}(t)\ . \end{aligned}$$

Za realizacijo z 1-naslovnimi multiplekserji razčlenimo vhodno funkcijo R po spremenljivki K, in vhodno funkcijo S po spremenljivki J in dobimo:

$$\begin{aligned} R &= \bar{K}(0) \vee K(Q(t)) \\ S &= \bar{J}(0) \vee J(\bar{Q}(t)) \end{aligned}$$

Realizacija JK pomnilne celice z RS pomnilno celico in 1-naslovnimi multiplekserji je prikazana na sliki 10.28.



Slika 10.28: Izvedba JK pomnilne celice z RS in 1-naslovnimi MX-ji

VAJA 10.4.4:

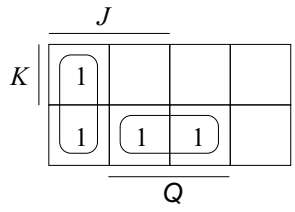
Realizirajte sinhronsko JK pomnilno celico z uporabo sinhronske D pomnilne celice in 1-naslovnega multiplekserja.

Zapišemo binarno aplikacijsko tabelo za JK pomnilno celico z vhodnimi spremenljivkami J, K in $Q(t)$. Določimo vhodno funkcijo za D pomnilno celico (Tabela 10.20).

Tabela 10.20: Aplikacijska tabela JK pomnilne celice

J	K	$Q(t)$	$Q(t+1) = D$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

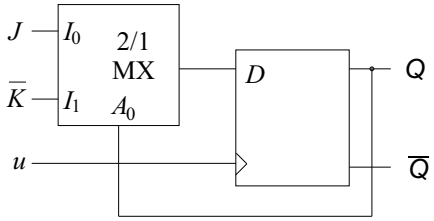
Minimiziramo vhodno funkcijo D, ker želimo dobiti minimalno rešitev z 1-naslovnim multiplekserjem.



Vhodna funkcija D v minimalni obliki

$$D = \bar{K}Q(t) \vee J\bar{Q}(t)$$

je že kar razčlenitev po časovni spremenljivki $Q(t)$, ker imamo pri $Q(t) = 0$ vhod J preslikan na izhod in pri $Q(t) = 1$ vhod K preslikan na izhod. Za naslovno spremenljivko vzamemo $Q(t)$ in dobimo realizacijo JK pomnilne celice z uporabo sinhronske D pomnilne celice in 1-naslovnega multiplekserja (Slika 10.29).



Slika 10.29: Sinhronska JK pomnilna celica - D pomnilna celica in 2/1MX

VAJA 10.4.5:

Realizirajte 2- bitni register $Y = (y_1, y_2)$ z dvema sinhronskima D pomnilnima celicama in 2-naslovnimi multiplekserji za naslednje funkcije:

- Reset - brisanje registra: $Y(t+1) = 0$
- Load - vpis 2-bitnega podatka: $Y(t+1) = X$, če je $X = (x_1, x_2)$
- Hold - ohranjanje stare vrednosti registra: $Y(t+1) = Y(t)$

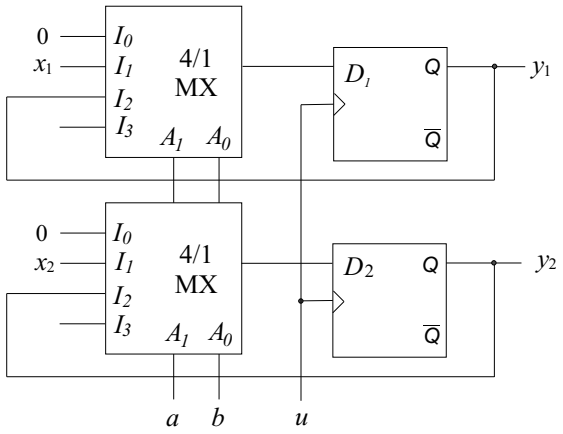
Register ima tri krmilne funkcije (Reset, Load, Hold), ki ju bomo kodirali v dve vhodni spremenljivki a in b. Zapišemo binarno aplikacijsko tabelo z vhodnimi spremenljivkami a, b, y_1, y_2 v času t in $t+1$. Podatkovni spremenljivki x_1, x_2 bomo vključili v naslednjih stanjih registra (Tabela 10.21).

a	b	*Operacija*
0	0	Reset
0	1	Load
1	0	Hold
1	1	×

Vhodni funkciji D_1 in D_2 sta enaki izhodoma y_1 in y_2 v času $t+1$. Realizacijo vhodnih funkcij z multiplekserji že poznamo, zato ne bomo posebej pisali vseh funkcijskih ostankov za razčlenitev po spremenljivkah a in b, ampak jih iz tabele direktno upoštevamo v realizaciji. Podatkovni vhod I_3 pri multiplekserjih nima definirane nobene vrednosti, ker lahko zavzame karkoli (×). V realizaciji z logično shemo predstavimo povezavo dveh D pomnilnih celic in 2-naslovnih multiplekserjev z izračunanimi podatkovnimi vhodi iz tabele (Slika 10.30).

Tabela 10.21: Vhodni funkciji D_1 in D_2 pomnilnih celic

a	b	$y_1(t)$	$y_2(t)$	$y_1(t+1)$	$y_2(t+1)$	D_1	D_2	
0	0	0	0	0	0	0	0	
0	0	0	1	0	0	0	0	
0	0	1	0	0	0	0	0	
0	0	1	1	0	0	0	0	I_0
0	1	0	0	x_1	x_2	x_1	x_2	
0	1	0	1	x_1	x_2	x_1	x_2	
0	1	1	0	x_1	x_2	x_1	x_2	
0	1	1	1	x_1	x_2	x_1	x_2	I_1
1	0	0	0	0	0	0	0	
1	0	0	1	0	1	0	1	
1	0	1	0	1	0	1	0	
1	0	1	1	1	1	1	1	I_2
1	1	0	0	×	×	×	×	
1	1	0	1	×	×	×	×	
1	1	1	0	×	×	×	×	
1	1	1	1	×	×	×	×	I_3



Slika 10.30: 2-bitni univerzalni register

VAJA 10.4.6:

Realizirajte 2- bitni register $Y = (y_1, y_2)$ z dvema sinhronskima JK pomnilnima celicama, 2- naslovnimi multiplekserji in XOR vrati za naslednje funkcije:

- Vpis - vpis 2-bitnega podatka: $Y(t+1) = X$, če je $X = (x_1, x_2)$
- Set - postavljanje registra: $Y(t+1) = 1$
- CPL - ciklični pomik levo: $y_1(t+1) = y_2(t), y_2(t+1) = y_1(t)$
- Reset - brisanje registra: $Y(t+1) = 0$

Register ima štiri krmilne funkcije (Vpis, Set, CPL, Reset), ki jih bomo kodirali z dvema vhodnima spremenljivkama a in b .
Zapišemo binarno aplikacijsko tabelo z vhodnimi spremenljivkami a, b, x_1, x_2 in izhoda vezja y_1, y_2, ki sta časovni spremenljivki (Tabela 10.22). Vhodni funkciji za JK pomnilni celici bomo poenostavili tako, da izenačimo vhoda $J = K$ in imamo samo dve vhodni funkciji, ki ju obravnavamo tako kot T pomnilno celico.

a	b	*Operacija*
0	0	Vpis
0	1	Set
1	0	CPL
1	1	Reset

Tabela 10.22: Vhodni funkciji $J_1 = K_1$ in $J_2 = K_2$ pomnilnih celic

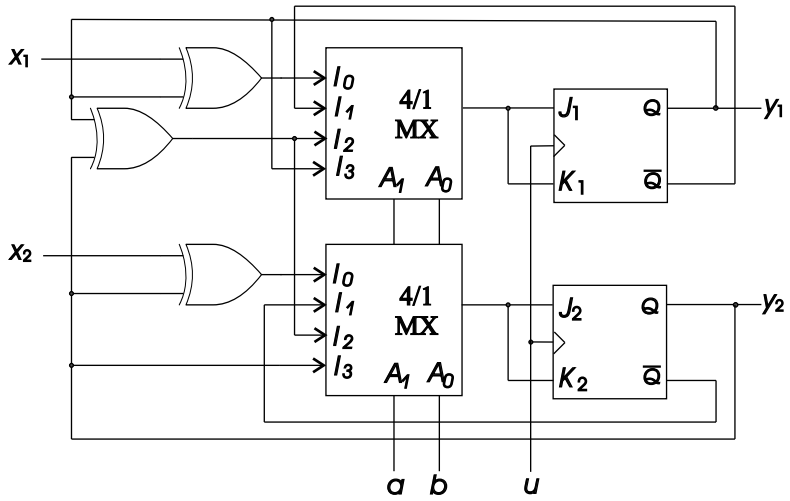
a	b	$y_1(t)$	$y_2(t)$	$y_1(t+1)$	$y_2(t+1)$		$J_1 = K_1$	$J_2 = K_2$	
0	0	0	0	x_1	x_2	$\|\|$	$x_1 \nabla y_1$	$x_2 \nabla y_2$	
0	0	0	1	x_1	x_2	$\|\|$	$x_1 \nabla y_1$	$x_2 \nabla y_2$	
0	0	1	0	x_1	x_2	$\|\|$	$x_1 \nabla y_1$	$x_2 \nabla y_2$	
0	0	1	1	x_1	x_2	$\|\|$	$x_1 \nabla y_1$	$x_2 \nabla y_2$	I_0
0	1	0	0	1	1	$\|\|$	1	1	
0	1	0	1	1	1	$\|\|$	1	0	
0	1	1	0	1	1	$\|\|$	0	1	
0	1	1	1	1	1	$\|\|$	0	0	I_1
1	0	0	0	0	0	$\|\|$	0	0	
1	0	0	1	1	0	$\|\|$	1	1	
1	0	1	0	0	1	$\|\|$	1	1	
1	0	1	1	1	1	$\|\|$	0	0	I_2
1	1	0	0	0	0	$\|\|$	0	0	
1	1	0	1	0	0	$\|\|$	0	1	
1	1	1	0	0	0	$\|\|$	1	0	
1	1	1	1	0	0	$\|\|$	1	1	I_3

Pri vpisu vhodnih spremenljivk x_1 in x_2 dobimo operacijo XOR zaradi dveh vrednosti, ki ju lahko zavzame. Pri $x_i = 0$ in $y_i = 0$ ali $x_i = 1$ in $y_i = 1$, mora celica ohraniti prejšnje stanje in na vhoda $J_i = K_i$ pripeljemo $x_i \nabla y_i = 0$. Pri $x_i = 1$ in $y_i = 0$ ali $x_i = 0$ in $y_i = 1$, mora celica zamenjati prejšnje stanje in na vhoda $J_i = K_i$ pripeljemo $x_i \nabla y_i = 1$. Vhodni funkciji v takem primeru izračunamo z dodatno tabelo, v kateri upoštevamo samo x_i, y_i v času t in $t + 1$ za določitev vhodne funkcije $J_i = K_i$ (Tabela 10.23).

Tabela 10.23: Vhodna funkcija $J_i = K_i$ za vpis vhoda x_i

x_i	$y_i(t)$	$y_i(t+1)$	$J_i = K_i$	
0	0	0	0	
0	1	0	1	
1	0	1	1	
1	1	1	0	$J_i = K_i = x_i \nabla y_i$

Realizacijo vhodnih funkcij z multiplekserji že poznamo, zato ne bomo posebej pisali vseh funkcijskih ostankov za razčlenitev po spremenljivkah a in b, ampak jih iz tabele direktno upoštevamo v realizaciji. V realizaciji z logično shemo predstavimo povezavo dveh JK pomnilnih celic, 2-naslovnih multiplekserjev in XOR vrat z izračunanimi podatkovnimi vhodi iz tabele (Slika 10.31).



Slika 10.31: 2-bitni univerzalni register z JK pomnilnimi celicami

VAJA 10.4.7:

Realizirajte števec po modulu M=8 z zmanjševanjem za 1. V izvedbi uporabite JK pomnilne celice in minimalno število logičnih vrat.

Števec po modulu M=8 se zmanjšuje in ima osem vrednosti (0, 1, 2,..., 7). Za zapis največje vrednosti 7 potrebujemo tri binarna mesta, zato zapišemo binarno aplikacijsko tabelo s tremi časovnimi spremenljivkami Q_2, Q_1, Q_0 v času t in določimo prehode v čas $t+1$ iz podanega pravila štetja (Tabela 10.24). Trenutno stanje števca na levi strani tabele zmanjšamo za 1 in ga vpišemo na desno stran tabele kot naslednje stanje. Za dobljeno delovanje števca v tabelo vpišemo vhodne funkcije JK pomnilnih celic.

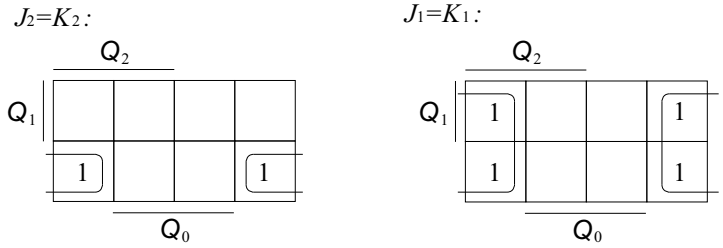
Tabela 10.24: Vhodne funkcije 3-bitnega števca

$Q_2(t)$	$Q_1(t)$	$Q_0(t)$	$Q_2(t+1)$	$Q_1(t+1)$	$Q_0(t+1)$	J_2K_2	J_1K_0	J_0K_0
0	0	0	1	1	1	1 ×	1 ×	1 ×
0	0	1	0	0	0	0 ×	0 ×	× 1
0	1	0	0	0	1	0 ×	× 1	1 ×
0	1	1	0	1	0	0 ×	× 0	× 1
1	0	0	0	1	1	× 1	1 ×	1 ×
1	0	1	1	0	0	× 0	0 ×	× 1
1	1	0	1	0	1	× 0	× 1	1 ×
1	1	1	1	1	0	× 0	× 0	× 1

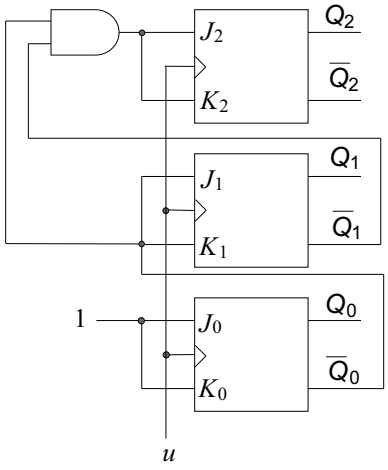
JK pomnilna celica ima dve vhodni funkciji, ki ju lahko združimo v $J = K$, ker je ob točno določeni vrednosti ene druga vedno nedoločena (×). Sedaj imamo samo tri minimalne oblike vhodnih funkcij (Slika 10.32) brez redundančnih kombinacij.

$$\begin{aligned} J_2 = K_2 &= \bar{Q}_1\bar{Q}_0 \\ J_1 = K_1 &= \bar{Q}_0 \\ J_0 = K_0 &= 1 \end{aligned}$$

Z minimizacijo vhodnih funkcij dobimo zapis vhodnih funkcij $J_2 = K_2$, $J_1 = K_1$, $J_0 = K_0$ in jih uporabimo za realizacijo števca za zmanjševanje po modulu M=8 z elementarnimi logičnimi vrati (Slika 10.33).



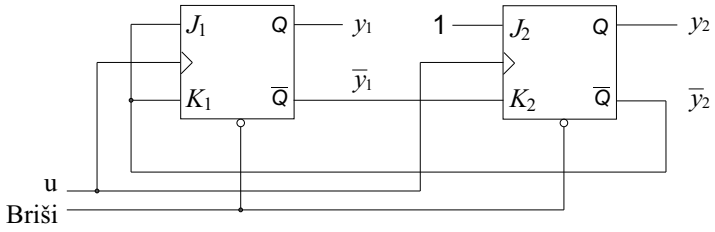
Slika 10.32: Minimizacija vhodnih funkcij $J_2 = K_2, J_1 = K_1, J_0 = K_0$



Slika 10.33: Logična shema števca po modulu M=8

VAJA 10.4.8:

Za sekvenčno vezje v logični shemi zapišite binarno aplikacijsko tabelo (Slika 10.34).



Slika 10.34: Logična shema sekvenčnega vezja

Iz podane sheme logičnega vezja najprej zapišimo vhodne funkcije pomnilnih celic za določanje prehodov stanj pri izbranih vhodnih kombinacijah.

$$\begin{aligned} J_1 = K_1 &= \bar{y}_2 \\ J_2 &= 1 \\ K_2 &= \bar{y}_1 \end{aligned}$$

Vzamemo splošno pomnilno enačbo za JK pomnilno celico $Q(t+1) = J\bar{Q}(t) \vee \bar{K}Q(t)$ in zapišemo izhode

$$\begin{aligned} y_1(t+1) &= J_1\bar{y}_1(t) \vee \bar{K}_1 y_1(t) \\ y_2(t+1) &= J_2\bar{y}_2(t) \vee \bar{K}_2 y_2(t) \end{aligned}$$

ter izračunamo naslednja stanja $y_1(t+1)$, $y_2(t+1)$ z vstavljanjem vhodnih funkcij v pomnilno enačbo in dobimo

$$\begin{aligned} y_1(t+1) &= \bar{y}_2\bar{y}_1 \vee y_2 y_1 \\ y_2(t+1) &= 1\bar{y}_2 \vee y_1 y_2 \end{aligned}$$

Iz izračunanih pomnilnih enačb zapišimo binarno aplikacijsko tabelo vezja (Tabela 10.25).

Tabela 10.25: Binarna aplikacijska tabela

$y_1(t)$	$y_2(t)$	$y_1(t+1)$	$y_2(t+1)$
0	0	1	1
0	1	0	0
1	0	0	1
1	1	1	1

Poglavje 11

Končni stroj stanj - avtomat

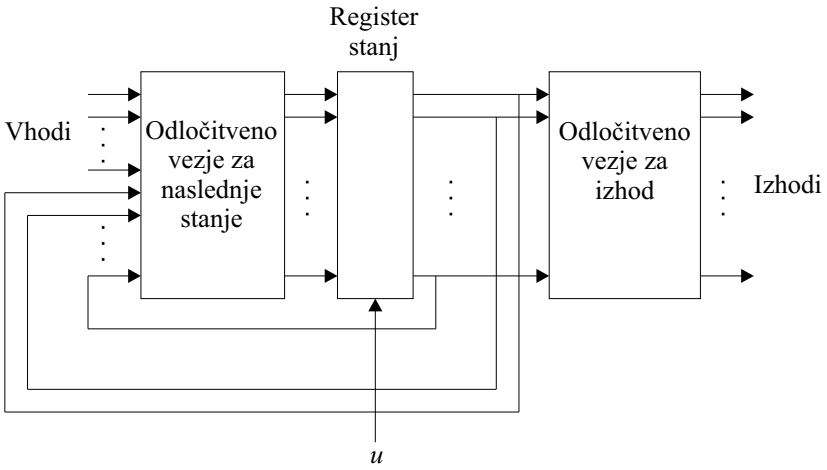
Sekvenčna vezja so najpogosteje obravnavana kot končni stroj stanj ali končni avtomat. To je sekvenčno vezje s končnim številom stanj. Števci obravnavani v prejšnjem poglavju so enostavni končni avtomati v katerih je natančno določena sekvenca posameznih stanj in je ni možno spreminjati. Bolj splošno velja, da so izhodi in naslednja stanja avtomata kombinacijske logične funkcije vhodov in trenutnega stanja. Izbira naslednjega stanja je lahko odvisna od vrednosti vhoda, kar vodi k bolj kompleksnemu delovanju kot ga poznamo pri števcih. Z avtomati realiziramo razne krmilne in odločitvene funkcije v digitalnih sistemih. Poznamo asinhronske in sinhronske avtomate. Za asinhronske avtomate je značilno, da menjajo stanje ob spremembi vhoda, njihovo delovanje pa ni časovno odvisno od drugih dogodkov. Bolj pogosti so sinhronski avtomati, ki spreminjajo stanje v odvisnosti od vhodov periodično, ob prihodu urinega signala. V nadaljevanju bomo obravnavali sinhronske končne avtomate in jih krajše imenovali kar avtomati.

11.1 Mooreov avtomat in Mealyjev avtomat

Avtomat opišemo z množico vhodnih črk $X = (x_1, x_2, ..., x_n)$, množico stanj $S = (s_1, s_2, ..., s_p)$ in množico izhodov $Y = (y_1, y_2, ..., y_m)$. Stanje v času t imenujemo trenutno stanje, stanje v času $t + 1$ naslednje stanje. Delovanje avtomata je opisano s prehajanjem stanj $S(t + 1) = f(X, S(t))$ in izhodi v odvisnosti od trenutnega stanja in vhodov.

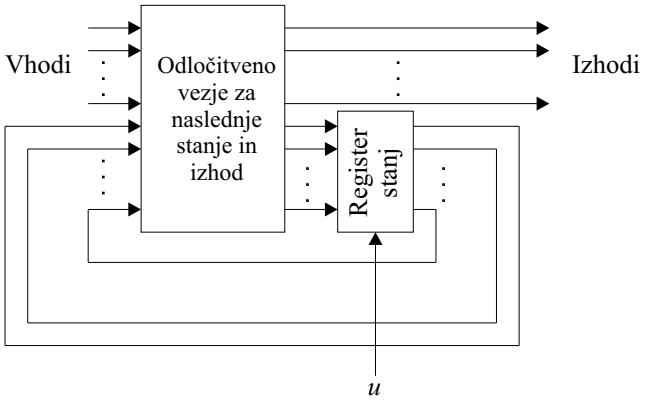
Poznamo dva osnovna načina izvedbe avtomatov:

- Mooreov avtomat - izhodi $Y = g(S(t))$ so odvisni samo od trenutnega stanja avtomata (Slika 11.1). Kombinacijska logika na osnovi vhodov in trenutnega stanja določa vhodne funkcije pomnilnih celic za vpis naslednjega stanja. Izhodi so izračunani s kombinacijsko logiko na osnovi trenutnega stanja, na izhodih pomnilnih celic in se spreminjajo z izbrano fronto urinega signala u. Izhodna logika je pri načrtovanju avtomatov lahko poenostavljena tako, da trenutna stanja predstavljajo izhode. Vsi števci, ki smo jih spoznali, so Moreovi avtomati, kjer trenutno stanje predstavlja vrednost na izhodu vezja.
- Mealyjev avtomat - izhodi $Y = g(X, S(t))$ so odvisni od vhoda in od trenutnega stanja avtomata (Slika 11.2). Izhod se spremeni takoj po spremembi vhoda neodvisno od ure. Mealyjev avtomat v tej obliki je asinhronski. Kombinacijska logika na osnovi trenutnega stanja in vhodov določa vhodne funkcije pomnilnih celic za naslednje stanje, kakor tudi izhode. Pri gradnji asinhronskega Mealyjevega avtomata v TTL tehnologiji imamo zahteven razvoj in potrebna je zelo natančna analiza



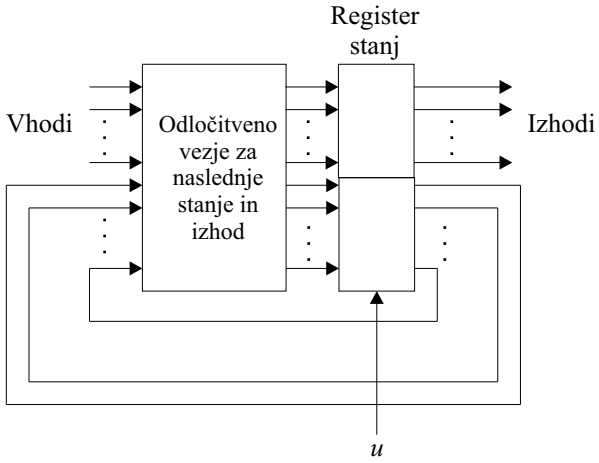
Slika 11.1: Mooreov avtomat

vhodno/izhodnega časovnega delovanja.



Slika 11.2: Mealyjev avtomat

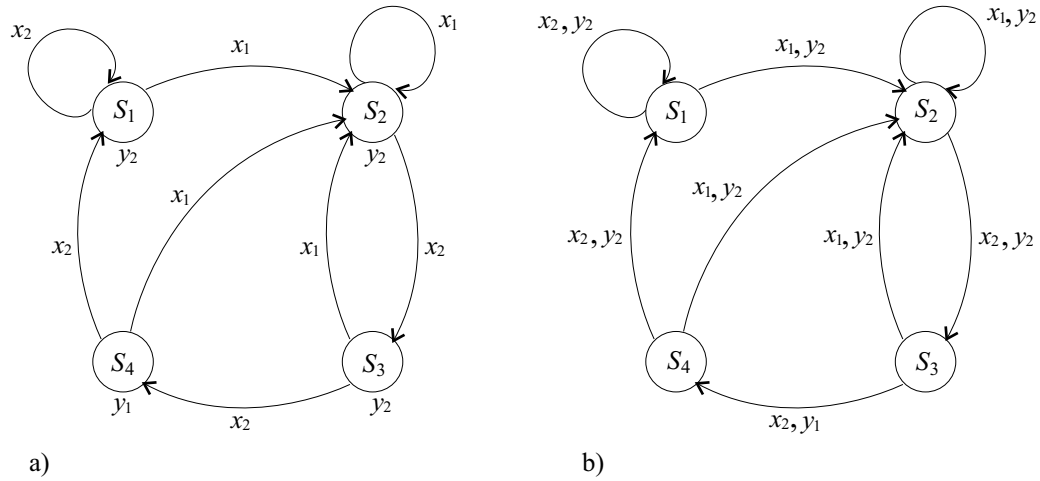
Definiramo lahko tudi sinhronski Mealyjev avtomat (Slika 11.3). Izhodi so sinhronizirani z uro tako, da so na izhod kombinacijske logike dodane pomnilne celice, ki so krmiljene z isto uro kot stanje avtomata.



Slika 11.3: Sinhronski Mealyjev avtomat

11.1.1 Opisovanje avtomatov

Delovanje avtomata predstavimo z grafom, ki ga imenujemo diagram prehajanja stanj. Vozlišča v grafu predstavljajo stanja avtomata, medtem ko se povezave med vozlišči nanašajo na prehode stanj. Ob prehodu je podan vhod, ki povzroči spremembo stanja in določi ustrezen izhod. Za Mooreov avtomat (Slika 11.4.a) označujemo izhode ob vozlišču, ker so samo funkcija trenutnega stanja. Pri Mealyjevem avtomatu (Slika 11.4.b) je izhod odvisen tudi od vhoda, zato ga pripišemo k vhodni črki na usmerjenih povezavah prehodov stanj.



Slika 11.4: Diagram prehajanja stanj - a) Moore, b) Mealy

Drug način opisa avtomatov, ki ga uporabljamo, je tabela prehajanja stanj. Vrstice predstavljajo trenutna in naslednja stanja avtomata, stolpci pa se nanašajo na vhodne črke. Za Mooreov avtomat je v vrstici trenutnega stanja $s_i(t)$ pri stolpcu vhoda x_j vpisano naslednje stanje $s_k(t+1)$, izhod y_v pa je vpisan v zadnjem stolpcu (Tabela 11.1). Za Mealyjev avtomat sta v vrstici trenutnega stanja $s_i(t)$ pri stolpcu vhoda x_j vpisana naslednje stanje $s_k(t+1)$ in izhod y_v (Tabela 11.2).

Tabela 11.1: Tabela prehajanja stanj za Mooreov avtomat

	x_1	x_2	
s_1	s_2	s_1	y_2
s_2	s_2	s_3	y_2
s_3	s_2	s_4	y_2
s_4	s_2	s_1	y_1

Tabela 11.2: Tabela prehajanja stanj za Mealyjev avtomat

	x_1	x_2
s_1	s_2, y_2	s_1, y_2
s_2	s_2, y_2	s_3, y_2
s_3	s_2, y_2	s_4, y_1
s_4	s_2, y_2	s_1, y_2

Tabela ali diagram prehajanja stanj nam omogočata določanje izhodne sekvence, ki je generirana s sekvenco vhodov in začetnim stanjem. Za vhodno sekvenco $x_1x_2x_2x_2x_1x_1x_2x_2$ določimo izhodno sekvenco, ki je pripeljana na začetno stanje s_1. Izhodna sekvenca, ki jo dobimo iz zgornje tabele prehajanja stanj je $y_2y_2y_1y_2y_2y_2y_2y_1$. Poglejmo si, kako smo prišli do dobljene izhodne sekvence. Če je avtomat v začetnem stanju s_1 in imamo vhod x_1 iz podane sekvence, potem iz tabele prehajanja stanj vidimo, da bo naslednje stanje s_2 z izhodom y_2. Iz tega stanja gre ob vhodu x_2 avtomat v stanje s_3 in ima izhod y_2. Vhod x_2 in stanje s_3 nam dasta prehod v stanje s_4 in izhod y_1, itd.

11.1.2 Ekvivalenca avtomatov

Dva avtomata sta ekvivalentna, če se pri enakem vhodnem nizu odzivata z enakim izhodnim nizom. Vsak končni avtomat tipa Moore lahko pretvorimo v ekvivalenten Mealyjev avtomat in obratno. Pri opazovanju izhodnega niza ekvivalentnih avtomatov prva črka na izhodu Mooreovega avtomata ne sodi v opazovano izhodno sekvenco, ker še ni vezana na nobeno vhodno črko.

Pretvorba Mooreovega avtomata v Mealyjev avtomat

Pri pretvorbi iz Mooreovega avtomata v Mealyjev avtomat ostane število stanj nespremenjeno. Izhodne črke Mealyjevega avtomata določimo tako, da prehodom stanj v času $t+1$ dodamo izhodno črko stanja Mooreovega avtomata.

Primer: Za Mooreov avtomat v tabeli prehajanja stanj poiščimo ekvivalenten Mealyjev avtomat.

Tabela 11.3: Tabela prehajanja stanj za Mooreov avtomat

	x_1	x_2	
s_1	s_5	s_3	y_0
s_2	s_5	s_3	y_1
s_3	s_1	s_5	y_0
s_4	s_2	s_4	y_1
s_5	s_3	s_3	y_0

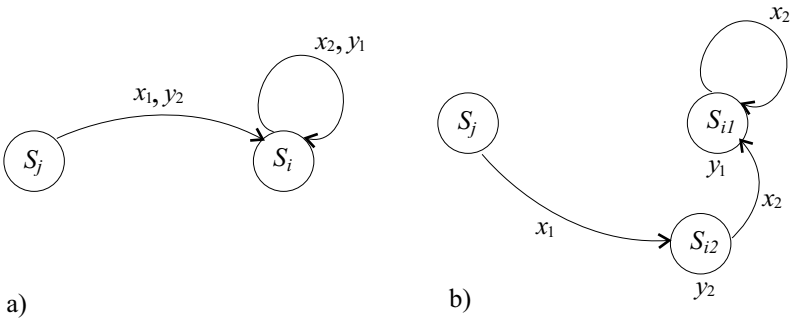
Število stanj Mealyjevega avtomata in prehodi stanj so enaki kot pri Mooreovem avtomatu. Izhod pri naslednjem stanju $s_i(t+1)$ Mealyjevega avtomata je enak izhodu tega stanja Moorovega avtomata (Tabela 11.4).

Tabela 11.4: Ekvivalenten Mealyjev avtomat

	x_1	x_2
s_1	s_5, y_0	s_3, y_0
s_2	s_5, y_0	s_3, y_0
s_3	s_1, y_0	s_5, y_0
s_4	s_2, y_1	s_4, y_1
s_5	s_3, y_0	s_3, y_0

Pretvorba Mealyjevega avtomata v Mooreov avtomat

Razpredelnica stanj za ekvivalenten Mooreov avtomat je določena na osnovi povezave stanj z vhodno črko. Število stanj je odvisno od tega koliko različnih izhodnih črk definira prehod v naslednje stanje Mealyjevega avtomata (Slika 11.5). Pri Mealyjevem avtomatu imamo dva prehoda v stanje s_i, to sta x_1, y_2 iz stanja s_j in x_2, y_1 iz stanja s_i. Če želimo zapisati ekvivalenten Mooreov avtomat moramo definirati dve stanji, kjer ima eno izhod y_1 in ga označimo z s_{i1} in drugo izhod y_2 in oznako s_{i2}. Za nova stanja Mooreovega avtomata v diagramu nato določimo še prehode stanj.



Slika 11.5: Diagram prehajanja stanj - a) Moore, b) Mealy

V tabeli Mealyjevega avtomata pregledamo prehode stanj s pripadajočimi izhodi. Vsako stanje s_i Mealyjevega avtomata ima lahko izhodno črko y_j, kar pomeni da imamo eno ali več stanj Mooreovega avtomata. Za vseh p stanj pogledamo pri prehodih stanj izhodne črke in določimo stanja Mooreovega avtomata. Indeksi stanj Mooreovega avtomata so dvojni (s_{ij}), kjer je i indeks stanja in j indeks izhoda Mealyjevega avtomata. Iste oznake se nato uporabljajo za prehode stanj pri vhodnih črkah. Če pri prehodih stanj ne najdemo vseh, je takšno stanje potrebno dodati na koncu tabele kot stanje z nedoločenim izhodom (s_{i-}).
Maksimalno število stanj, ki jih lahko dobimo je 'število stanj * število izhodov'.

Primer: Za Mealyjev avtomat v tabeli prehajanja stanj poiščimo ekvivalenten Mooreov avtomat.

Tabela 11.5: Mealyjev avtomat

	x_1	x_2
s_1	s_2, y_1	s_3, y_0
s_2	s_5, y_1	s_3, y_1
s_3	s_1, y_0	s_5, y_0
s_4	s_2, y_1	s_4, y_1
s_5	s_3, y_0	s_2, y_1

Tabela 11.5 ima pri prehodih stanj naslednje kombinacije stanj in izhodov: s_1, y_0; s_2, y_1; s_3, y_0; s_3, y_1; s_4, y_1; s_5, y_0; s_5, y_1, kar nam določa 7 stanj Mooreovega avtomata, ki jih označimo z dvojnimi indeksi: s_{10}, s_{21}, s_{30}, s_{31}, s_{41}, s_{50}, s_{51}. V zapisu vidimo, da je upoštevanih vseh pet stanj podanega Mealyjevega avtomata, zato zapišemo tabelo prehajanja za Mooreov avtomat z novimi oznakami stanj in njihovimi izhodi. Prehode stanj prepišemo iz tabele Mealyjevega avtomata z novimi oznakami, kjer se vrstice pri stanjih z več različnimi izhodi ponovijo. Izhodi Mooreovega avtomata ustrezajo izhodom, ki smo

jih upoštevali pri določanju stanj Mealyjevega avtomata.

Tabela 11.6: Zapis ekvivalentnega Mooreovega avtomata z indeksi stanj in izhodov

	x_1	x_2	
s_{10}	s_{21}	s_{30}	y_0
s_{21}	s_{51}	s_{31}	y_1
s_{30}	s_{10}	s_{50}	y_0
s_{31}	s_{10}	s_{50}	y_1
s_{41}	s_{21}	s_{41}	y_1
s_{50}	s_{30}	s_{21}	y_0
s_{51}	s_{30}	s_{21}	y_1

Dobljeni Mooreov avtomat (Tabela 11.6) preoblikujemo s preimenovanjem stanj: $s_1 = s_{10}$, $s_2 = s_{21}$, $s_3 = s_{30}$, $s_4 = s_{31}$, $s_5 = s_{41}$, $s_6 = s_{50}$, $s_7 = s_{51}$ in ga zapišemo v novo tabelo prehajanja stanj. (Tabela 11.7).

Tabela 11.7: Ekvivalenten Mooreov avtomat

	x_1	x_2	
s_1	s_2	s_3	y_0
s_2	s_7	s_4	y_1
s_3	s_1	s_6	y_0
s_4	s_1	s_6	y_1
s_5	s_2	s_5	y_1
s_6	s_3	s_2	y_0
s_7	s_3	s_2	y_1

11.2 Minimizacija avtomata

Vsak avtomat, ki ga želimo realizirati, naj ima čim manj stanj. V tem primeru imamo manj logičnih vrat in manj pomnilnih celic v implementaciji. Za avtomat z n stanji potrebujemo najmanj k pomnilnih celic, če velja $2^{k-1} < n \leq 2^k$. Z zmanjšanjem števila stanj na 2^{k-1} ali manj prihranimo eno pomnilno celico. Tudi če zmanjšanje števila stanj ne omogoča prihranka pomnilne celice, lahko v tem primeru dobimo več redundantnih kombinacij, kar zagotavlja enostavnejšo kombinacijsko logiko.

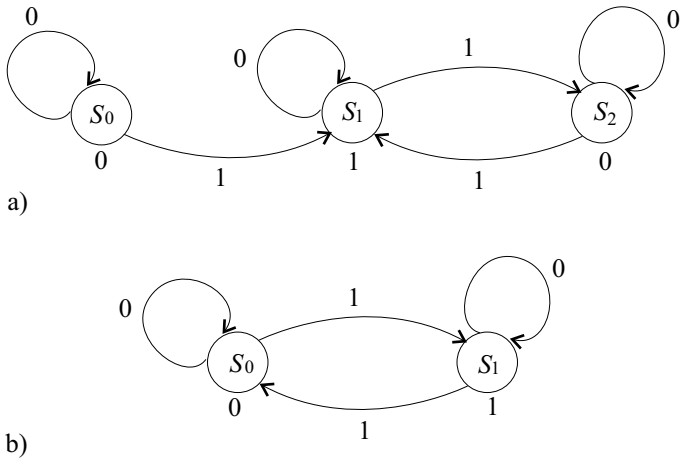
Postopek minimizacije stanj avtomata lahko izvedemo na več načinov:

- z združevanjem stanj v tabeli prehajanja stanj,
- z metodo ujemanja vrstic,
- z implikacijskim diagramom.

Prva dva postopka sta primerna za minimizacijo na papirju, tretji pa je bolj sistematičen in je primeren za računalniško izvedbo. Za naše potrebe pri načrtovanju avtomatov, si bomo podrobneje ogledali postopek združevanja stanj v tabeli prehajanja stanj.

Minimizacija avtomata združuje stanja z ekvivalentnim obnašanjem. Dve stanji sta ekvivalentni, če imata enake izhode za vse vhodne kombinacije in prehajata v isto ali ekvivalentno stanje. Oglejmo si primer vezja lihega preverjanja paritete, ki je predstavljeno z

dvema različnima diagramoma prehajanja stanj.



Slika 11.6: DPS - liho preverjanje paritete

V diagramu prehajanja stanj (Slika 11.6.a) sta stanji s_0 in s_2 ekvivalentni, ker imata obe izhod 0, obe prehajata v stanje s_1 pri vhodu 1 in obe ostajata v istem stanju ob vhodu 0. Stanji s_0 in s_2 združimo v eno stanje s_0 (Slika 11.6.b) in v tem avtomatu dobimo za vsak vhodni niz enak izhodni niz v obeh diagramih prehajanja stanj.

Postopek minimizacije z združevanjem stanj:

1) Stanja avtomata združujemo v grupe glede na enake izhodne črke stanj (Mooreov avtomat) in glede na enake izhode prehajanj stanj (Mealyjev avtomat). Grupe označimo z $G_{i,j}$, kjer je i - indeks grupe in j - korak minimizacije. Vsakemu stanju $s_k(t+1)$ v tabeli pripišemo indeks grupe - $s_k(t+1)/i$.

2) Pogledamo prehode stanj pri vsaki grupi in če gredo v isto ali v drugo grupo za vsako vhodno črko, pomeni takšna grupa novo stanje avtomata. Če ta pogoj za grupo ne velja, se grupo razdeli in ta delitev se ponavlja, dokler niso v vsaki grupi stanja, ki prehajajo v isto ali drugo grupo.

3) Vsaka grupa stanj predstavlja novo stanje minimalnega avtomata, ki ga zapišemo v novo tabelo prehajanja stanj.

Primer: Oglejmo si minimizacijo Mooreovega avtomata v podani tabeli prehajanja stanj.

Tabela 11.8: Tabela prehajanja stanj za Mooreov avtomat

	x_1	x_2	
s_1	s_5	s_3	y_0
s_2	s_5	s_3	y_1
s_3	s_1	s_5	y_0
s_4	s_2	s_4	y_1
s_5	s_3	s_3	y_0

Pri združevanju stanj Mooreovega avtomata imamo stanja s_1, s_3 in s_5 z izhodom y_0, ki jih združimo v eno grupo in stanji s_2 in s_4 z izhodom y_1, ki ju združimo v drugo grupo

Tabela 11.9: Združevanje stanj z enakim izhodom

		x_1	x_2	
	s_1	$s_5/1$	$s_3/1$	y_0
	s_3	$s_1/1$	$s_5/1$	y_0
$G_{1,1}$	s_5	$s_3/1$	$s_3/1$	y_0
	s_2	$s_5/1$	$s_3/1$	y_1
$G_{2,1}$	s_4	$s_2/2$	$s_4/2$	y_1

(Tabela 11.9) in zapišemo novo tabelo prehajanja stanj z oznakami grup. V grupi $G_{2,1}$ stanji s_2 in s_4 ne prehajata v isto grupo pri nobenem od vhodov, zato ju moramo v naslednjem koraku razdeliti v dve grupi (Tabela 11.10).

Tabela 11.10: Delitev grup

		x_1	x_2	
	s_1	$s_5/1$	$s_3/1$	y_0
	s_3	$s_1/1$	$s_5/1$	y_0
$G_{1,2}$	s_5	$s_3/1$	$s_3/1$	y_0
$G_{2,2}$	s_2	$s_5/1$	$s_3/1$	y_1
$G_{3,2}$	s_4	$s_2/2$	$s_4/3$	y_1

V minimizacijskem postopku smo dobili tri grupe $G_{1,2}$, $G_{2,2}$, $G_{3,2}$, ki jih preimenujemo v stanja $s_1 = G_{1,2}$, $s_2 = G_{2,2}$, $s_3 = G_{1,2}$ in zapišemo minimalno obliko Mooreovega avtomata. (Tabela 11.11).

Tabela 11.11: Minimalen Mooreov avtomat

	x_1	x_2	
s_1	s_1	s_1	y_0
s_2	s_1	s_1	y_1
s_3	s_2	s_3	y_1

11.3 Primeri končnih avtomatov

Za načrtovanje in implementacijo avtomata je potrebno definirati bolj splošen postopek:

- Razumevanje problema - za delovanje avtomata imamo običajno podan tekstovni opis, ki ga moramo interpretirati na nedvoumen način. Na osnovi opisa preizkusimo nekaj vhodnih sekvenc tako, da smo prepričani, da razumemo pogoje pod katerimi so generirani različni izhodi.
- Abstrakten opis delovanja avtomata - ko enkrat razumemo delovanje avtomata, ga zapišemo v tako obliko, ki je najbolj primerna za njegovo implementacijo. Ena od možnosti je diagram prehajanja stanj (DPS).
- Minimizacija stanj - zapis avtomata v diagramu stanj ima pogosto preveč stanj. Določena stanja so lahko izločena, zato ker je vhodno/izhodno obnašanje podvojeno z drugimi ekvivalentnimi potmi. Ta korak ni potreben v enostavnih avtomatih, kot so števci.

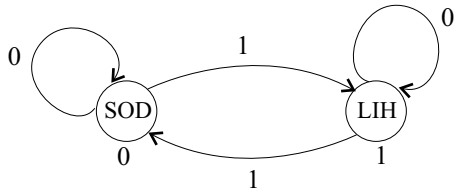
- Kodiranje stanj - v števcih smo imeli stanje in izhod enaka, zato ni bilo pomembno kako kodirati stanja. V splošnih avtomatih so izhodi funkcija bitov shranjenih v pomnilnih celicah (stanj) in vhodov, zato lahko dobro kodiranje vodi v enostavnejšo izvedbo.
- Izbira pomnilnih celic - D celice poenostavijo postopek izvedbe, ker dobimo vhodne funkcije pomnilnih celic kar z naslednjim stanjem avtomata, JK pomnilne celice zmanjšajo število potrebnih vrat.
- Izvedba avtomata - izračun kombinacijskih logičnih funkcij naslednjega stanja in izhoda za izbrane pomnilne celice in dvonivojska ali večnivojska realizacija teh funkcij.

Primer: Avtomat za preverjanje lihe paritete vhodne sekvence

Vzemimo, da želimo razviti vezje, ki šteje število enic v bitni serijski vhodni sekvenci. Če vezje potrjuje svoj izhod z 1, ko je na vhodu liho število enic, ga imenujemo liho preverjanje paritete. Vezje, ki ga želimo zgraditi, je čisto sekvenčno, ker je trenutni izhod odvisen od celotne zgodovine vhodov.

Diagram prehajanja stanj
Najprej bomo razvili diagram prehajanja stanj, ki opisuje delovanje vezja za liho preverjanje paritete. Vezje je lahko v enem od dveh stanj: v sekvenci je bilo do tega trenutka liho ali sodo število enic. Kadar je na vhodu 1, je potrebno preklopiti v drugo stanje. Na primer, če je bilo do tega trenutka prisotnih liho število enic in je trenutni vhod 1, potem bomo imeli sedaj sodo število enic. Če pa bo na vhodu 0, ostane v istem stanju.

Diagram prehajanja stanj (Slika 11.7) ima dve stanji, ki ju imenujemo LIH in SOD. Izhodi so povezani kar s stanji in jih zapišemo pod vozliščem stanja. Če je bilo na vhodu vidnih liho število enic, bo izhod 1, sicer pa bo 0. Vhodne vrednosti pa povzročajo spreminjanje stanj, ki so označene z usmerjenimi povezavami.



Slika 11.7: Diagram prehajanja stanj

Tabela prehajanja stanj
Druga predstavitev za opis delovanja vezja je tabela prehajanja stanj (Tabela 11.12), ki pogosto vsebuje simbolična imena za vhode, izhode, trenutna in naslednja stanja. Ker iz takega zapisa delovanja avtomata še vedno ne moremo definirati vezja, moramo vpeljati binarno kodiranje in zapisati binarno aplikacijsko tabelo prehajanja stanj (Tabela 11.13). V binarni tabeli smo kodirali avtomat tako, da je stanje SOD enako 0 in stanje LIH enako 1. Nova tabela sedaj deluje kot pravilnostna tabela iz katere bomo definirali logično vezje avtomata.

V binarni aplikacijski tabeli bomo vhod označili z x, trenutno stanje s $Q(t)$, naslednje stanje s $Q(t+1)$ in izhod z y.

Tabela 11.12: Simbolična tabela prehajanja stanj

Vhod	Trenutno st.	Naslednje st.	Izhod
0	SOD	SOD	0
0	LIH	LIH	1
1	SOD	LIH	0
1	LIH	SOD	1

Tabela 11.13: Binarna aplikacijska tabela prehajanja stanj

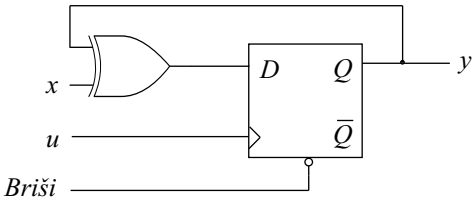
x	$Q(t)$	$Q(t+1)$	y	D	T
0	0	0	0	0	0
0	1	1	1	1	0
1	0	1	0	1	1
1	1	0	1	0	1

Naslednje stanje in izhod
Iz pravilnostne tabele zapišimo logični funkciji za naslednje stanje $Q(t+1)$ in izhod y.

$$\begin{aligned} Q(t+1) &= x \bigtriangledown Q(t) \\ y &= Q(t) \end{aligned}$$

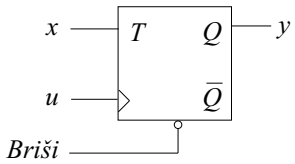
Izvedba avtomata
Zgradimo vezje avtomata, kjer bo stanje shranjeno v pomnilni celici. V našem primeru imamo dve stanji, ki ju lahko shranimo v eno pomnilno celico. Vhod v to pomnilno celico bo določal naslednje stanje.

Vzemimo najprej D pomnilno celico za realizacijo avtomata. Ker je vhod D pomnilne celice podan z $D = Q(t+1)$, imamo na vhodu XOR vrata, ki računajo vhodno funkcijo D kot funkcijo trenutnega stanja $Q(t)$ in vhoda x (Slika 11.8).



Slika 11.8: Avtomat za preverjanje lihe paritete z D pomnilno celico

Za T pomnilno celico imamo enostavnejšo rešitev, kjer z vhodom x določamo delovanje avtomata (Tabela 11.13). Opisana rešitev ne zahteva dodatnih logičnih vrat (Slika 11.9).



Slika 11.9: Avtomat za preverjanje lihe paritete s T pomnilno celico

Primer: Na primeru INC/DEC števca po modulu M=4, ki je predstavljen kot Mooreov avtomat v tabeli prehajanja stanj, si oglejmo realizacijo avtomata. Funkcija INC pomeni povečevanje števca in se izvaja ob vhodni črki x_0, DEC pa zmanjševanje števca ob vhodni črki x_1. Za števec po modulu M=4 imamo štiri stanja in štiri izhode, kjer izhodi ustrezajo stanjem števca.

Tabela 11.14: 2-bitni INC/DEC števec

	x_0	x_1	
s_0	s_1	s_3	0
s_1	s_2	s_0	1
s_2	s_3	s_1	2
s_3	s_0	s_2	3

Kodiranje avtomata
Kodiranje avtomata pomeni zapis vhodov, stanj in izhodov z binarno kodo. Za vsako abecedo določimo število spremenljivk in kodirne tabele (Tabela 11.15).

Tabela 11.15: Kodirne tabele

Vhod	a	Stanje	Q_1	Q_2	Izhod	o_1	o_2
x_0	0	s_0	0	0	0	0	0
x_1	1	s_1	0	1	1	0	1
		s_2	1	0	2	1	0
		s_3	1	1	3	1	1

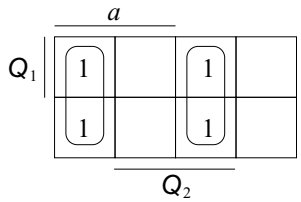
Binarna aplikacijska tabela
Iz tabele prehajanja stanj zapišemo delovanje števca v binarno aplikacijsko tabelo z vhodno spremenljivko a, dvema časovnima spremenljivkama Q_1 in Q_2 in izhodoma o_1 in o_2 (Tabela 11.16).

Tabela 11.16: Binarna aplikacijska tabela INC/DEC števca

a	$Q_1(t)$	$Q_2(t)$	$Q_1(t+1)$	$Q_2(t+1)$	o_1	o_2	T_1	T_2
0	0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	1	1
0	1	0	1	1	1	0	0	1
0	1	1	0	0	1	1	1	1
1	0	0	1	1	0	0	1	1
1	0	1	0	0	0	1	0	1
1	1	0	0	1	1	0	1	1
1	1	1	1	0	1	1	0	1

Izbira pomnilnih celic in izvedba kombinacijske logike
Na izvedbo avtomata vpliva izbira pomnilnih celic. Uporabimo T pomnilne celice, za katere izračunamo vhodni funkciji (Tabela 11.16). Zapišemo minimalni obliki za funkciji T_1 in T_2 (Slika 11.10) in izhoda o_1 in o_2 ter izberemo najprimernejše gradnike za izvedbo kombinacijskega vezja.

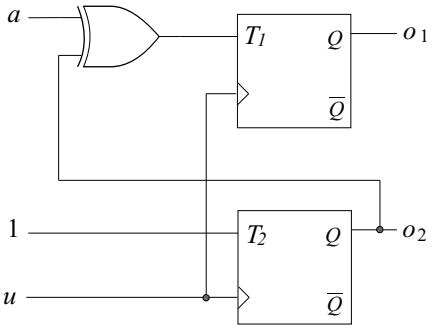
$$o_1 = Q_1(t)$$
$$o_2 = Q_2(t)$$



Slika 11.10: Minimizacija vhoda T_1 INC/DEC števca po modulu 4

$$T_1 = \bar{a}Q_2(t) \vee a\bar{Q}_2(t) = a \nabla Q_2(t)$$
$$T_2 = 1$$

Za izvedbo funkcije T_1 smo uporabili XOR operator (Slika 11.11).



Slika 11.11: Logična shema INC/DEC števca po modulu 4

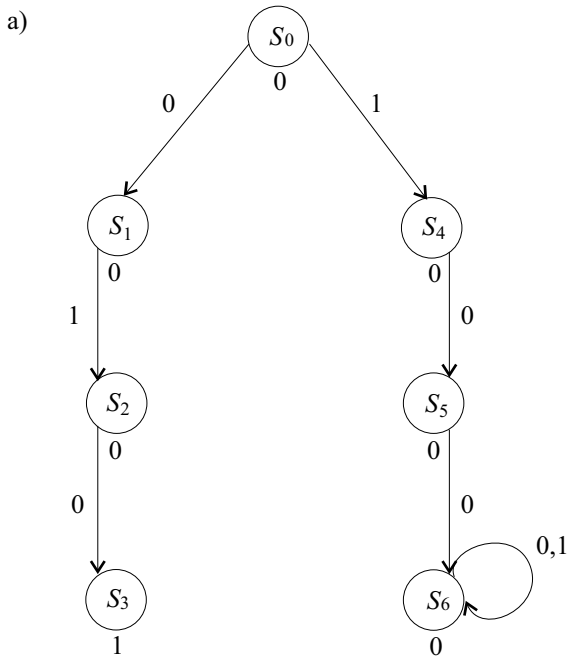
Primer: Zapisali bomo diagram prehajanja stanj avtomata za razpoznavanje končnega niza z vhodom x in izhodom y. Izhod je aktiven, kadar je na vhodu razpoznan končni niz ...010..., vse dokler se ne pojavi niz ...100.

Za razumevanje problema si zapišimo dva primera vhodno/izhodnih kombinacij za razpoznavanje končnega niza (Tabela 11.17).

Tabela 11.17: Določanje vhodno/izhodnega niza

x	:	0	0	1	0	1	0	1	0	0	1	0	...
y	:	-	0	0	0	1	0	1	0	1	0	0	0 ...
x	:	1	1	0	1	1	0	1	0	0	1	0	...
y	:	-	0	0	0	0	0	0	0	1	0	0	0 ...

Formalni zapis:
Delovanje avtomata sedaj predstavimo z diagramom prehajanja stanj za Mooreov avtomat. Začnemo tako, da narišemo najprej tiste dele diagrama, ki pomenijo razpoznavanje zahtevanih nizov: 010 in 100. Začetno stanje, ki ga dosežemo s Start vhodom označimo z s_0. Iz staja s_0 vodi ena pot z nizom 010 v stanje z izhodom 1, druga pot pa konča z zanko za niz 100 in izhodom 0 (Slika 11.12.a). Stanje s_6 pomeni da ne bomo nikoli več razpoznavali niza, zato avtomat ostane v tem stanju ob vhodu 0 ali 1.



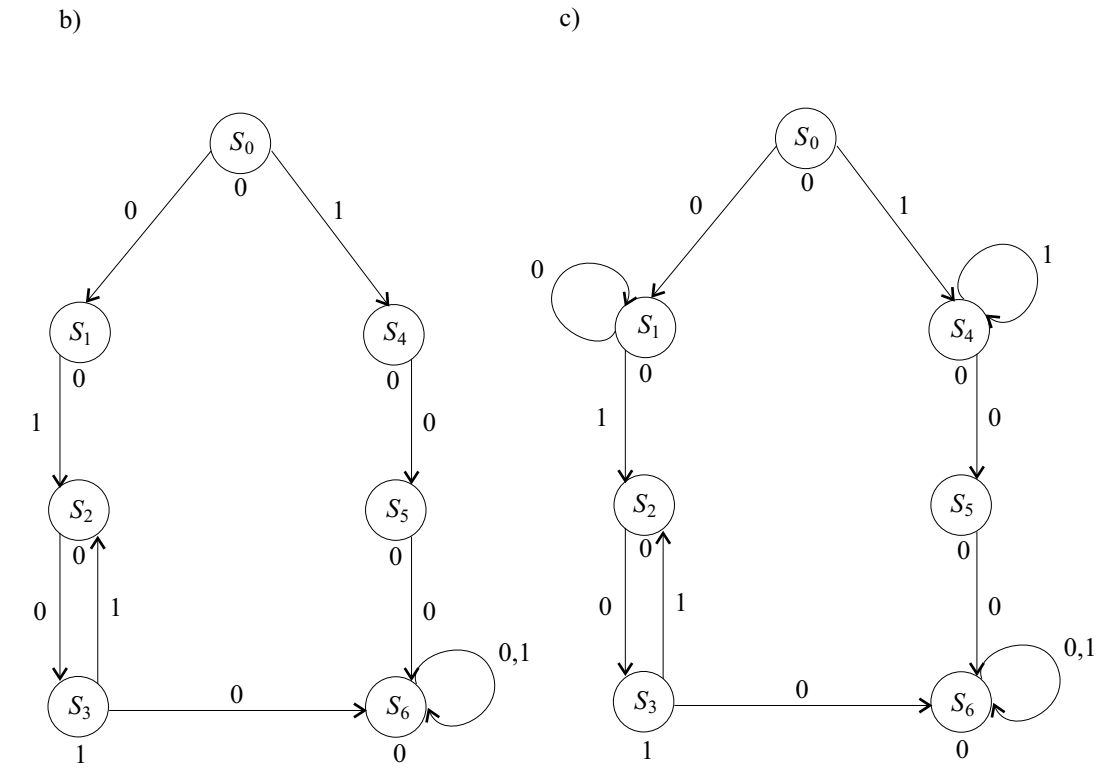
Slika 11.12: DPS - razpoznavanje nizov 010 in 100

Nadaljujemo z dodajanjem prehodov stanj še za druge vrednosti vhodov. Vsako stanje lahko prejme vhod 0 ali 1, zato poglejmo najprej kako je s prehodi za stanje s_3. Če je na vhodu 0 bomo dobili končni niz ...0100, zato povežemo s_3 s stanjem s_6 pri vhodu 0. Če je na vhodu 1, potem bo avtomat imel niz ...0101, ki nas privede v s_2, ker če bo zopet na vhodu 0 dobimo ...01010, kar nas znova privede v stanje s_3, ki daje izhod 1 (Slika 11.13.b).

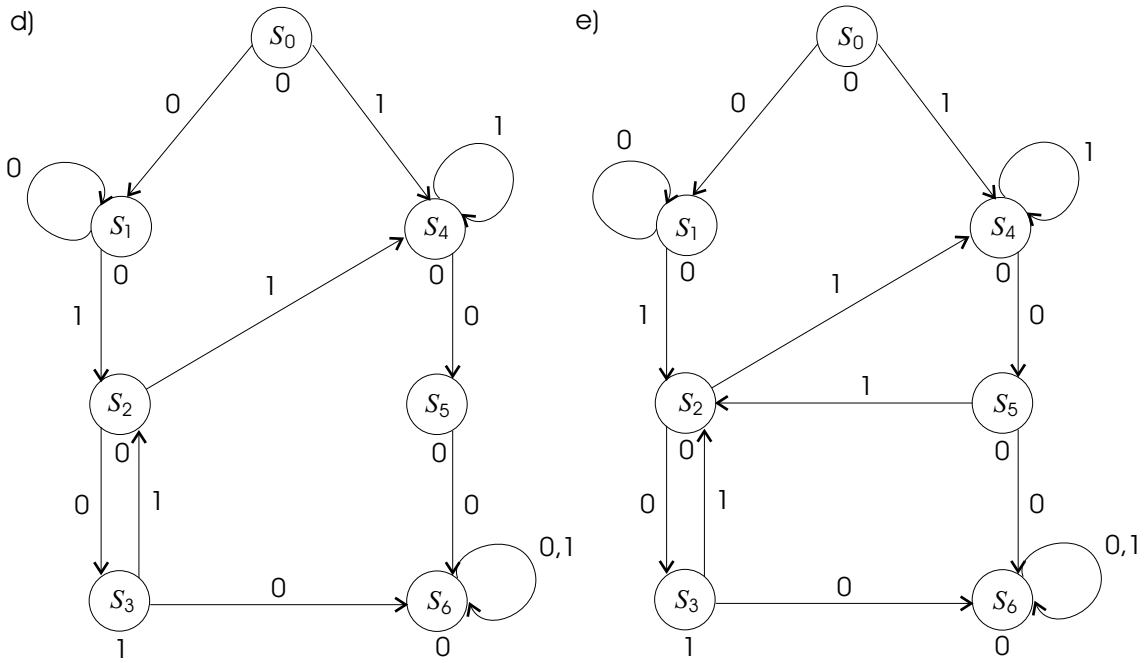
Stanje s_1 se pri vrednosti 0 na vhodu ohranja, ker je 0 vedno primerna za vsako kombinacije pred ...010. Podobno velja za stanje s_4, kjer nam pred končnim nizom ...100 lahko nastopa poljubno število 1. Avtomat v tem stanju ostane toliko časa, dokler je na vhodu niz 1 (Slika 11.13.c).

Stanje s_2 predstavlja niz ...01, ki je predpona za ...010, če bi bil vhod 0. Če je vhod 1, potem je lahko ta niz ...011 predpona za končno sekvenco ...100. Ob vhodu 1 gre zato avtomat iz stanja s_1 v stanje s_4, ki vodi do končnega niza (Slika 11.14.d).

Stanje s_5 predstavlja niz, kjer vhodu 1 sledi vhod 0 in če je naslednji vhod 1 dobimo niz ...101, ki je predpona za končno stanje ...010. Ker s_2 predstavlja niz oblike ...01, gre avtomat iz stanja s_5 v stanje s_2 (Slika 11.14.e).



Slika 11.13: DPS - prehodi pri stanjih s_3, s_1, s_4



Slika 11.14: DPS - prehodi pri stanjih s_2, s_5

11.4 Vaje

VAJA 11.7.1:

Za Mooreov avtomat v tabeli prehajanja stanj poiščite ekvivalenten Mealyjev avtomat.

Tabela 11.18: Tabela prehajanja stanj za Mooreov avtomat

	x_1	x_2	
s_1	s_1	s_3	y_0
s_2	s_3	s_1	y_1
s_3	s_1	s_5	y_1
s_4	s_2	s_2	y_1
s_5	s_5	s_3	y_0

Vhodne črke in število stanj ter prehodi stanj Mealyjevega avtomata so ekvivalentni Mooreovemu, zato jih prepišemo v novo tabelo. Prehodom stanj $s_i(t+1)$ Mealyjevega avtomata pripišemo izhodno črko Mooreovega stanja $s_i(t)$ (Tabela 11.19).

Tabela 11.19: Ekvivalenten Mealyjev avtomat

	x_1	x_2
s_1	s_1, y_0	s_3, y_1
s_2	s_3, y_1	s_1, y_0
s_3	s_1, y_0	s_5, y_0
s_4	s_2, y_1	s_2, y_1
s_5	s_5, y_0	s_3, y_1

VAJA 11.7.2:

Za Mealyjev avtomat v tabeli prehajanja stanj poiščite ekvivalenten Mooreov avtomat.

Tabela 11.20: Mealyjev avtomat

	x_1	x_2
s_1	s_2, y_1	s_4, y_0
s_2	s_2, y_1	s_4, y_1
s_3	s_1, y_0	s_2, y_0
s_4	s_2, y_1	s_1, y_0

Število stanj Moorovega avtomata bomo določili iz prehodov stanj v povezavi z izhodno črko. Tabela 11.20 ima pri prehodih stanj naslednje kombinacije stanj in izhodov: s_1, y_0; s_2, y_0; s_2, y_1; s_4, y_0; s_4, y_1, kar nam določa 5 stanj Mooreovega avtomata. Pri prehodih vidimo, da ni upoštevano stanje s_3, zato ga podamo kot s_{3-} in ima nedoločeno izhodno črko. Zapišemo tabelo prehajanja za Mooreov avtomat z novimi oznakami stanj s_{ij} in njihovimi izhodi. Prehode stanj prepišemo iz tabele Mealyjevega avtomata, kjer se vrstice pri stanjih z različnimi izhodi ponovijo. Izhodi Mooreovega avtomata ustrezajo indeksom izhoda pri določanju stanj.
Dobljeni Mooreov avtomat (Tabela 11.21) z dvojnimi indeksi preoblikujemo s preimenovanjem stanj: $s_1 = s_{10}$, $s_2 = s_{20}$, $s_3 = s_{21}$, $s_4 = s_{3-}$, $s_5 = s_{40}$, $s_6 = s_{41}$ (Tabela 11.22).

Tabela 11.21: Zapis ekvivalentnega Mooreovega avtomata

	x_1	x_2	
s_{10}	s_{21}	s_{40}	y_0
s_{20}	s_{21}	s_{41}	y_0
s_{21}	s_{21}	s_{41}	y_1
s_{3-}	s_{10}	s_{20}	$-$
s_{40}	s_{21}	s_{10}	y_0
s_{41}	s_{21}	s_{10}	y_1

Tabela 11.22: Ekvivalenten Mooreov avtomat

	x_1	x_2	
s_1	s_3	s_5	y_0
s_2	s_3	s_6	y_0
s_3	s_3	s_6	y_1
s_4	s_1	s_2	$-$
s_5	s_3	s_1	y_0
s_6	s_3	s_1	y_1

VAJA 11.7.3:

Minimizirajte Mealyjev avtomat v podani tabeli prehajanja stanj.

Tabela 11.23: Tabela prehajanja stanj za Mealyjev avtomat

	x_1	x_2	
s_1	s_5, y_1	s_2, y_1	
s_2	s_1, y_0	s_4, y_1	
s_3	s_5, y_1	s_6, y_1	
s_4	s_3, y_0	s_2, y_1	
s_5	s_4, y_1	s_3, y_0	
s_6	s_1, y_0	s_6, y_1	

Prehode stanj z enakimi izhodnimi črkami združimo v tri grupe. V prvi sta stanji s_1 in s_3 z izhodoma y_1, y_1, v drugi so stanja s_2, s_4 in s_6 z izhodoma y_0, y_1 in v tretji je stanje s_5 z izhodoma y_1, y_0 (Tabela 11.24).

Tabela 11.24: Grupiranje prehodov stanj z enakimi izhodi

		x_1	x_2	
	s_1	$s_5/3$	$s_2/2$	
$G_{1,1}$	s_3	$s_5/3$	$s_6/2$	
	s_2	$s_1/1$	$s_4/2$	
	s_4	$s_3/1$	$s_2/2$	
$G_{2,1}$	s_6	$s_1/1$	$s_6/2$	
$G_{3,1}$	s_5	$s_4/2$	$s_3/1$	

V vseh grupah imamo stanja, ki prehajajo v isto grupo, kar pomeni da smo v minimizacijskem postopku dobili tri grupe $G_{1,1}$, $G_{2,1}$, $G_{3,1}$, ki jih preimenujemo v stanja $s_1 = G_{1,1}$, $s_2 = G_{2,1}$, $s_3 = G_{1,1}$ in zapišemo minimalno obliko Mealyjevega avtomata, kjer dodamo stanjem v grupi tudi izhode (Tabela 11.25).

Tabela 11.25: Minimalen Mealyjev avtomat

	x_1	x_2
s_1	s_3, y_1	s_2, y_1
s_2	s_1, y_0	s_2, y_1
s_3	s_2, y_1	s_1, y_0

VAJA 11.7.4:

Minimizirajte Mooreov avtomat v podani tabeli prehajanja stanj.

Tabela 11.26: Tabela prehajanja stanj za Mooreov avtomat

	x_1	x_2	x_3	
s_1	s_5	s_2	s_3	y_0
s_2	s_1	s_4	s_6	y_1
s_3	s_5	s_6	s_1	y_0
s_4	s_3	s_2	s_4	y_1
s_5	s_4	s_3	s_6	y_0
s_6	s_1	s_6	s_2	y_1

Pri grupiranju stanj Mooreovega avtomata imamo stanja s_1, s_3 in s_5 z izhodom y_0, ki jih združimo v eno grupo in stanja s_2 in s_4 in s_6 z izhodom y_1, ki ju združimo v drugo grupo (Tabela 11.27).

Tabela 11.27: Grupiranje stanj z enakim izhodom

		x_1	x_2	x_3	
	s_1	$s_5/1$	$s_2/2$	$s_3/1$	y_0
	s_3	$s_5/1$	$s_6/2$	$s_1/1$	y_0
$G_{1,1}$	s_5	$s_4/2$	$s_3/1$	$s_6/2$	y_0
	s_2	$s_1/1$	$s_4/2$	$s_6/2$	y_1
	s_4	$s_3/1$	$s_2/2$	$s_4/2$	y_1
$G_{2,1}$	s_6	$s_1/1$	$s_6/2$	$s_2/2$	y_1

V grupi $G_{1,1}$ vsa tri stanja ne prehajajo v isto grupo pri nobenem od vhodov, zato jih moramo v naslednjem koraku razdeliti v dve grupi. V eni grupi ostaneta stanji s_1 in s_3, ker imata obe prehode v isto grupo pri vseh vhodnih črkah, stanje s_5 bo v drugi grupi (Tabela 11.22).

Tabela 11.28: Delitev grup

		x_1	x_2	x_3	
	s_1	$s_5/2$	$s_2/3$	$s_3/1$	y_0
$G_{1,2}$	s_3	$s_5/2$	$s_6/3$	$s_1/1$	y_0
$G_{2,2}$	s_5	$s_4/3$	$s_3/1$	$s_6/3$	y_0
	s_2	$s_1/1$	$s_4/3$	$s_6/3$	y_1
	s_4	$s_3/1$	$s_2/3$	$s_4/3$	y_1
$G_{3,2}$	s_6	$s_1/1$	$s_6/3$	$s_2/3$	y_1

V minimizacijskem postopku smo dobili tri grupe $G_{1,2}$, $G_{2,2}$, $G_{3,2}$, ki jih preimenujemo

v stanja $s_1 = G_{1,2}$, $s_2 = G_{2,2}$, $s_3 = G_{1,2}$ in zapišemo minimalno obliko Mooreovega avtomata. (Tabela 11.29).

Tabela 11.29: Minimalen Mooreov avtomat

	x_1	x_2	x_3	
s_1	s_2	s_3	s_1	y_0
s_2	s_3	s_1	s_3	y_0
s_3	s_1	s_3	s_3	y_1

VAJA 11.7.5:

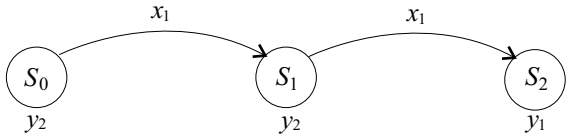
Imamo avtomat z vhodnima črkama x_1, x_2 in izhodnima črkama y_1, y_2. Z diagramom prehajanja stanj predstavite delovanje Mooreovega avtomata, ki razpozna vsak končni niz dveh zaporednih črk x_1x_1 z izhodom y_1.

Za razumevanje problema razpoznavanja si zapišimo primer vhodno/izhodnega niza Mooreovega avtomata (Tabela 11.30):

Tabela 11.30: Določanje vhodno/izhodnega niza

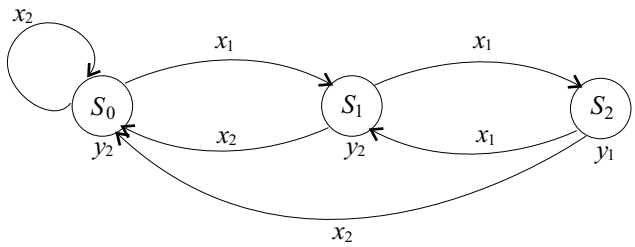
x	:	x_1	x_2	x_1	x_1	x_2	x_1	x_1	x_1	x_1	...
y	:	-	y_2	y_2	y_2	y_1	y_2	y_2	y_1	y_2	y_1	...

Iz opisa problema in sekvence razpoznavanja niza vhodnih črk, si oglejmo gradnjo diagrama prehajanja stanj Mooreovega avtomata.
Začnemo z risanjem tistega dela diagrama, ki razpozna zahtevani niz: x_1x_1. Začetno stanje označimo z s_0. Iz stanja s_0 pridemo z vhodnima črkama x_1x_1 preko stanja s_1 v stanje s_2 (Slika 11.15). Stanje s_2 je za razpoznavanje niza končno z izhodom y_1, stanji s_0 in s_1 pa imata izhod y_2.



Slika 11.15: DPS - razpoznavanje niza x_1x_1

Nadaljujemo z dodajanjem prehodov stanj še za druge možnosti vhodov. Vsako stanje lahko prejme vhod x_1 ali x_2, zato poglejmo najprej kako je s prehodi za stanje s_0. Če je na vhodu x_2, bo avtomat ostal v stanju s_0 z izhodom y_2 vse dokler ne pride x_1, ki lahko vodi v niz x_1x_1. Iz stanja s_1 se ob vhodni črki x_2 vrnemo v stanje s_0 z izhodom y_2 in znova čakamo prihod črke x_1. V stanju s_2 imamo ob vhodni črki x_1 prehod v stanje s_1 z izhodom y_2, ker se bo izhod y_1 pojavil šele pri drugem x_1. Ob vhodni črki x_2 se iz stanja s_2 vrnemo na začetek v stanje s_0 (Slika 11.16).



Slika 11.16: DPS - razpoznavanje niza x_1x_1

VAJA 11.7.6:

Imamo avtomat z vhodnima črkama x_1, x_2 in izhodnima črkama y_1, y_2. Z diagramom prehajanja stanj predstavite delovanje Mealyjevega avtomata tako, da je na izhodu y_1 vsakič, ko se na vhodu pojavi x_2 za dvema zaporednima x_1x_1.

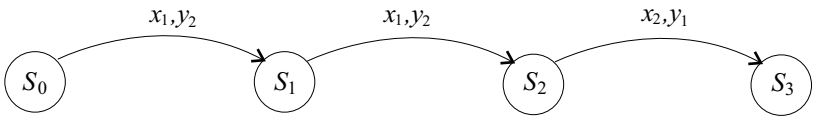
Za razumevanje problema razpoznavanja si zapišimo primer vhodno/izhodnega niza Mealyjevega avtomata (Tabela 11.31):

Tabela 11.31: Določanje vhodno/izhodnega niza

x	:	x_2	x_2	x_1	x_1	x_2	x_1	x_2	x_1	x_1	x_1	x_2	...
y	:	y_2	y_2	y_2	y_2	y_1	y_2	y_2	y_2	y_2	y_2	y_1	...

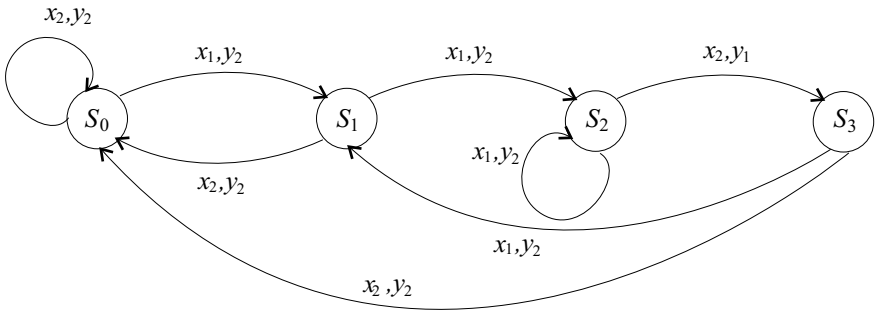
Iz opisa problema in sekvence razpoznavanja niza vhodnih črk, si oglejmo gradnjo diagrama prehajanja stanj Mealyjevega avtomata.

Začnemo z risanjem tistega dela diagrama, ki razpozna zahtevani niz: $x_1x_1x_2$. Začetno stanje označimo z s_0. Iz stanja s_0 pridemo z vhodnima črkama x_1x_1 preko stanja s_1 v stanje s_2 (Slika 11.17)in z x_2 v stanje s_3. Prehod v stanje s_3 je za razpoznavanje niza ob vhodni črki x_2 določen z izhodom y_1, prehoda v stanji s_1 in s_2 pa imata izhod y_2.



Slika 11.17: DPS - razpoznavanje niza $x_1x_1x_2$

Nadaljujemo z dodajanjem prehodov stanj še za druge možnosti vhodov. Vsako stanje lahko prejme vhod x_1 ali x_2, zato poglejmo najprej kako je s prehodi za stanje s_0. Če je na vhodu x_2, bo avtomat ostal v stanju s_0 z izhodom y_2 vse dokler ne pride x_1, ki lahko vodi v niz x_1x_1. Iz stanja s_1 se ob vhodni črki x_2 vrnemo v stanje s_0 z izhodom y_2 in znova čakamo prihod črke x_1. V stanju s_2 imamo ob vhodni črki x_1 ohranjanje stanja s_2 z izhodom y_2 in čakamo na vhodno črko x_2. Iz stanja s_3 se vrnemo v začetno stanje s_0 ob vhodni črki x_2, ker znova čakamo na sekvenco, ki jo želimo razpoznati (Slika 11.18) in v stanje s_1 ob vhodni črki x_1 .



Slika 11.18: DPS - razpoznavanje niza $x_1x_1x_2$

VAJA 11.7.7:

Realizirajte Mealyjev avtomat za razpoznavanje niza črk $x_1x_1x_2$, ki smo ga definirali z diagramom prehajanja stanj v prejšnji vaji (VAJA 11.7.6), z uporabo D pomnilnih celic in 1-naslovnih multiplekserjev.

Zapišimo najprej delovanje avtomata v tabelo prehajanja stanj (Tabela 11.32).

Tabela 11.32: Mealyjev avtomat

	x_1	x_2
s_0	s_1, y_2	s_0, y_2
s_1	s_2, y_2	s_0, y_2
s_2	s_2, y_2	s_3, y_1
s_3	s_1, y_2	s_0, y_2

Kodiranje avtomata
Kodiranje avtomata pomeni zapis vhodov, stanj in izhodov z binarno kodo. Za vsako abecedo določimo število spremenljivk in kodirne tabele (Tabela 11.33).

Tabela 11.33: Kodirne tabele

Vhod	a	Stanje	Q_1	Q_2	Izhod	o
x_1	0	s_0	0	0	y_1	0
x_2	1	s_1	0	1	y_2	1
		s_2	1	0		
		s_3	1	1		

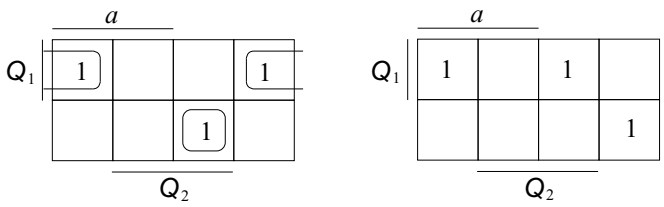
Binarna aplikacijska tabela
Iz tabele prehajanja stanj zapišemo delovanje Mealyjevega avtomata v binarno aplikacijsko tabelo z vhodno spremenljivko a, dvema časovnima spremenljivkama Q_1 in Q_2 in izhodom o (Tabela 11.34). Izhod o je odvisen od vhodne črke, zato imamo v binarni aplikacijski tabeli pri vhodu $a = 0$ vrednosti 1101, kar ustreza izhodom ob vhodni črki x_1 v tabeli prehajanja stanj in pri vhodu $a = 1$ vrednosti 1111, kar ustreza izhodom ob vhodni črki x_2 v tabeli prehajanja stanj (Tabela 11.32).

Uporaba D pomnilnih celic in izvedba kombinacijske logike z 1-naslovnimi multiplekserji
Za realizacijo Mealyjevega avtomata bomo uporabili D pomnilne celice, zato izračunamo

Tabela 11.34: Binarna aplikacijska tabela avtomata

a	$Q_1(t)$	$Q_2(t)$	$Q_1(t+1)$	$Q_2(t+1)$	o	D_1	D_2
0	0	0	0	1	1	0	1
0	0	1	1	0	1	1	0
0	1	0	1	0	1	1	0
0	1	1	0	1	1	0	1
1	0	0	0	0	1	0	0
1	0	1	0	0	1	0	0
1	1	0	1	1	0	1	1
1	1	1	0	0	1	0	0

minimalne disjunktivne normalne oblike (MDNO) vhodnih funkcij za $D_1 = Q_1$ in $D_2 = Q_2$ (Slika 11.19) ter izhoda o in jih uporabimo za realizacijo z 1-naslovnimi multiplekserji (Slika 11.20). Minimalna oblika izhoda o je enaka popolni konjunktivni normalni obliki (PKNO), ker imamo v binarni aplikacijski tabeli samo eno funcijsko vrednost nič ($f_2 = 0$).

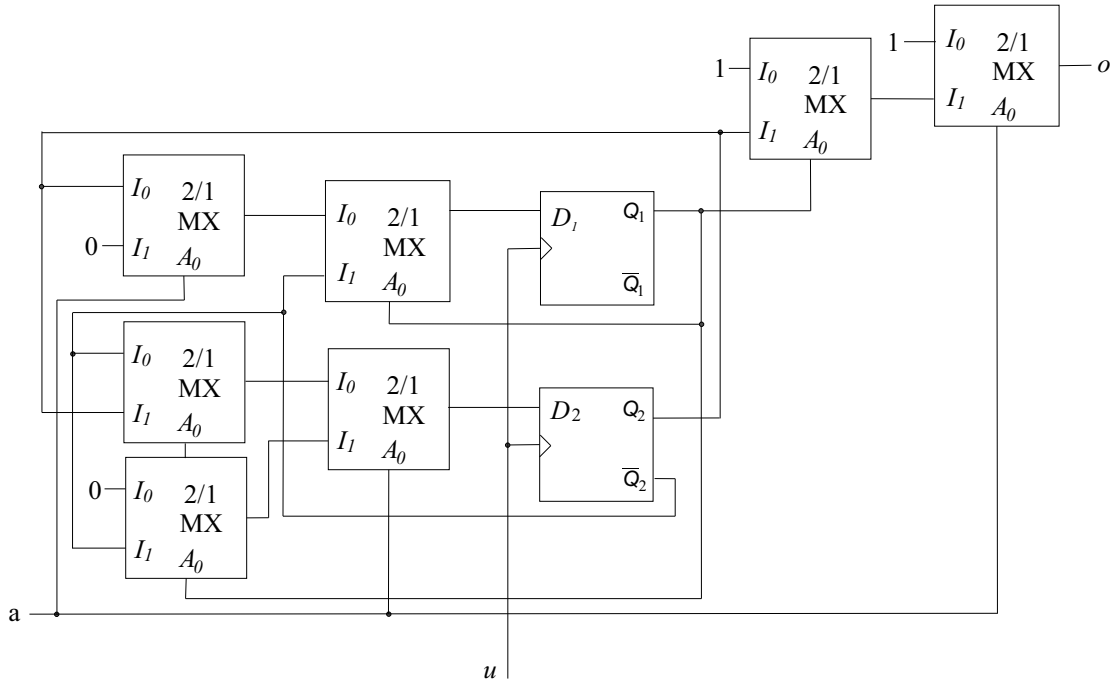


Slika 11.19: Mealyjev avtomat - minimizacija vhodnih funkcij D_1, D_2

$$
\begin{aligned}
o &= \bar{a} \vee \bar{Q}_1(t) \vee Q_2(t) \\
D_1 &= \bar{a}\bar{Q}_1(t)Q_2(t) \vee Q_1(t)\bar{Q}_2(t) \\
D_2 &= \bar{a}\bar{Q}_1(t)\bar{Q}_2(t) \vee \bar{a}Q_1(t)Q_2(t) \vee aQ_1(t)\bar{Q}_2(t)
\end{aligned}
$$

Razčlenitev vhodnih funkcij in izhoda za realizacijo z 1-naslovnimi multiplekserji je

$$
\begin{aligned}
o &= \bar{a} \vee \bar{Q}_1(t) \vee Q_2(t) \\
&= \bar{a}.(1) \vee a.(\bar{Q}_1(t) \vee Q_2(t)) \\
&= \bar{a}.(1) \vee a.[\bar{Q}_1(t).(1) \vee Q_1(t).(Q_2(t))] \\
D_1 &= \bar{a}\bar{Q}_1(t)Q_2(t) \vee Q_1(t)\bar{Q}_2(t) \\
&= \bar{Q}_1(t).(\bar{a}Q_2(t)) \vee Q_1(t).(\bar{Q}_2(t)) \\
&= \bar{Q}_1(t).[\bar{a}.(Q_2(t)) \vee a.(0))] \vee Q_1(t).(\bar{Q}_2(t)) \\
D_2 &= \bar{a}\bar{Q}_1(t)\bar{Q}_2(t) \vee \bar{a}Q_1(t)Q_2(t) \vee aQ_1(t)\bar{Q}_2(t) \\
&= \bar{a}.(\bar{Q}_1(t)\bar{Q}_2(t) \vee Q_1(t)Q_2(t)) \vee a.(Q_1(t)\bar{Q}_2(t)) \\
&= \bar{a}.[\bar{Q}_1(t)(\bar{Q}_2(t)) \vee Q_1(t)(Q_2(t))] \vee a.[\bar{Q}_1(t)(0) \vee Q_1(t)(\bar{Q}_2(t))]
\end{aligned}
\tag{11.1}
$$



Slika 11.20: Logična shema Mealyjevega avtomata

Literatura

[1] A.D.Friedman, Fundamentals of Logic Design and Switching Theory, Computer Science Press, Inc., USA 1986

[2] R.H.Katz, Contemporary Logic Design, The Benjamin/Cummings Publishing Company, Inc., 1994

[3] S.Muroga, Logic Design and Switching Theory, John & Sons, Inc.,USA 1986

[4] J.E.Palmer, D.E.Perlman, Theory and Problems of Introduction to Digital Systems, Schaum's Outline Series McGraw-Hill, Inc., 1993

[5] R.L.Tokheim, Theory and Problems of Digital Principles, Schaum's Outline Series McGraw-Hill, Inc., 1994

[6] J.Virant, Logične osnove odločanja in pomnjenja v računalniških sistemih, Zal. FER v Ljubljani, 1990