Priprave na vaje za predmet Osnove digitalnih vezij

Miha Moškon

2017

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Kataložni zapis o publikaciji (CIP) pripravili v Narodni in univerzitetni knjižnici v Ljubljani

COBISS.SI-ID=292687616

ISBN 978-961-6209-94-6 (pdf)

Elektronska izdaja knjige je na voljo na URL:
http://zalozba.fri.uni-lj.si/moskon2017.pdf

Recenzenta: doc. dr. Mira Trebar, prof. dr. Nikolaj Zimic
Založnik: Založba UL FRI, Ljubljana
Izdajatelj: UL Fakulteta za računalništvo in informatiko, Ljubljana
1. izdaja, 2017
Urednik: prof. dr. Franc Solina

Kazalo

Predgovor

Pred vami je skripta priprav na laboratorijske vaje pri predmetu Osnove digitalnih vezij, ki se izvaja na 1. stopnji univerzitetnih študijskih programov Računalništvo in informatika in Računalništvo in matematika na Fakulteti za računalništvo in informatiko Univerze v Ljubljani. Skripta služi obveznim pripravam, ki jih morajo slušatelji predmeta opraviti pred vsako laboratorijsko vajo. Vsaka priprava je sestavljena iz kratkega povzetka teoretičnih osnov in enega ali več praktičnih zgledov. Na podlagi tako pridobljenega znanja lahko študent pristopi k reševanju kviza, katerega oddaja je obvezna za pristop k naslednji laboratorijski vaji. Upam, da bo skripta služila svojemu namenu in bo študentom omogočila lažje osvajanje znanja s področja osnov digitalnih vezij, ki ga v določeni meri potrebuje vsak računalničar. Teoretične osnove lahko študenti najdejo v prosojnicah na spletni učilnici in dodatni literaturi kot je [4] (v slovenskem jeziku) in [2] (v angleškem jeziku), dodatne vaje pa so na voljo v [1, 3].

doc. dr. Miha Moškon, 2017

Zahvala

Najprej bi se zahvalil svojim predhodnikom, tj. asistentom pri predmetu Preklopne strukture in sistemi, za zasnovo vaj, ki jih je evolucija študijskih programov pripeljala v stanje, v katerem so danes. Prav tako bi se rad zahvalil prof. dr. Nikolaju Zimicu, za sodelovanje pri prenovi vaj po bolonjski reformi. Zahvaljujem se asistentoma Primožu Pečarju in Domnu Šoberlu, ki sta sodelovala pri nastajanju osnutka skripte, ki jo imate pred seboj. Zahvaljujem se tudi asistentoma Juretu Demšarju in Mattiji Petroniju za določene popravke in dopolnitve tekom izvajanja vaj v študijskih letih 2014/2015 in 2015/2016. Rad bi se zahvalil vsem sodelavcem Laboratorija za računalniške strukture in sisteme, predvsem pa Miranu Koprivcu, za pomoč pri izvajanju vaj in iskanju ter popravljanju napak. Za slednje bi se zahvalil tudi obema recenzentoma, prof. dr. Nikolaju Zimicu in doc. dr. Miri Trebar. Nenazadnje se zahvaljujem vsem študentom za njihove kritike, pohvale in predloge glede izvajanja vaj pri predmetu Osnove digitalnih vezij.

1 Priprava na 1. laboratorijske vaje

1.1 Boolova algebra in preklopne funkcije

Preklopne (tudi logične) funkcije so funkcije nad preklopnimi (tudi logičnimi) spremenljivkami (spremenljivke, ki lahko zavzamejo vrednosti iz množice $\{0, 1\}$) in nad katerimi temelji procesiranje podatkov z uporabo digitalnih vezij. Preklopne funkcije (operatorji), elementi nad katerimi operirajo (operandi) in pravila po katerih operirajo so definirani z Boolovo algebro.
Boolovo algebro lahko definiramo kot trojček $\{X, O, P\}$, kjer je X množica elementov Boolove algebre (operandov), O množica osnovnih funkcij (operatorjev), ki vsebuje disnjunkcijo ($\vee$) in konjunkcijo ($\cdot$), in P množica postulatov oziroma pravil, ki so opisana v nadaljevanju.

1.1.1 Postulati Boolove algebre

Zaprtost

$$\texttt{P1}: x, y \in X; x \vee y \in X$$
$$\texttt{P1}^*: x, y \in X; x \cdot y \in X$$

Nevtralni element

$$\texttt{P2}: x, 0 \in X; x \vee 0 = x$$
$$\texttt{P2}^*: x, 1 \in X; x \cdot 1 = x$$

Komutativnost

$$\texttt{P3}: x, y \in X; x \vee y = y \vee x$$
$$\texttt{P3}^*: x, y \in X; x \cdot y = y \cdot x$$

Distributivnost

$$\texttt{P4}: x, y, z \in X; x \vee (y \cdot z) = (x \vee y) \cdot (x \vee z)$$
$$\texttt{P4}^*: x, y, z \in X; x \cdot (y \vee z) = (x \cdot y) \vee (x \cdot z)$$

Inverzni element

$$\texttt{P5}: \forall x \in X, \exists \overline{x} \in X; x \vee \overline{x} = 1$$
$$\texttt{P5}^*: \forall x \in X, \exists \overline{x} \in X; x \cdot \overline{x} = 0$$

Število elementov

$$\texttt{P6}: \exists x, y \in X; x \neq y$$

1.1.2 Lastnosti Boolove algebre

Iz postulatov izhajajo določene lastnosti, ki jih lahko uporabimo pri poenostavljanju logičnih izrazov. V nadaljevanju je naštetih nekaj bolj uporabnih lastnosti.

Idempotenca

$$x \vee x \vee \cdots \vee x = x$$
$$x \cdot x \cdot \cdots \cdot x = x$$

Absorpcija

$$x \vee x \cdot y = x$$
$$x \cdot (x \vee y) = x$$

Asociativnost

$$(x \vee y) \vee z = x \vee (y \vee z) = x \vee y \vee z$$
$$(x \cdot y) \cdot z = x \cdot (y \cdot z) = x \cdot y \cdot z$$

DeMorganovo pravilo

$$\overline{(x_1 \vee x_2 \vee \cdots \vee x_n)} = \overline{x}_1 \cdot \overline{x}_2 \cdot \cdots \cdot \overline{x}_n$$
$$\overline{(x_1 \cdot x_2 \cdot \cdots \cdot x_n)} = \overline{x}_1 \vee \overline{x}_2 \vee \cdots \vee \overline{x}_n$$

1.2 Podajanje preklopnih funkcij

Vsako preklopno funkcijo lahko podamo na različne načine, pri čemer so osnovni načini sledeči:

- *Logični izraz* funkcijo podaja analitično, tj. kot enačbo, v kateri nastopajo logične spremenljivke (operandi), ki so med seboj povezane preko preklopnih funkcij (operatorji).
- *Logična shema* funkcijo podaja grafično, tj. s shemo njene realizacije. Vhodi v shemo opisujejo vhodne spremenljivke, izhodi iz sheme pa izhodne. V shemi uporabljamo logične simbole, ki predstavljajo osnovne logične operatorje (opozorilo: v različni literaturi boste srečali različne logične simbole).
- *Pravilnostna tabela* funkcijo podaja tabelarično, tj. podaja vse možne kombinacije vhodnih vektorjev in funkcijske vrednosti pri posamezni kombinaciji. Pri tem na levi strani tabele nastopajo vhodne (neodvisne) spremenljivke, ki določajo vhodni vektor, na desni pa izhodne (odvisne) spremenljivke. Poljubno preklopno funkcijo z n vhodnimi spremenljivkami lahko opišemo s pravilnostno tabelo, ki ima 2^n vrstic.
- *Veitchevega diagrama* zaenkrat še ne bomo podrobneje obravnavali.

1.3 Osnovne preklopne funkcije

Boolova algebra definira osnovna logična operatorja, tj. disjunkcijo (or) in konjunkcijo (and). Preko postulata *Inverzni element* vpeljemo še negacijo (not). V nadaljevanju podajamo osnovne logične operatorje, s katerimi lahko zapišemo poljubno preklopno funkcijo in so realizirani tudi znotraj družine logičnih čipov 7400.

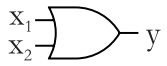
1.3.1 Disjunkcija

Disjunkcija (ALI, OR) vrne logično 1, ko je na logično 1 postavljena vsaj ena izmed njenih vhodnih spremenljivk. V logičnem izrazu jo označujemo s simbolom $\vee$. Slika 1.1 podaja logični izraz (a), logični simbol (b) in pravilnostno tabelo (c) za disjunkcijo z dvema vhodnima spremenljivkama.

1.3.2 Konjunkcija

Konjunkcija (IN, AND) vrne logično 1, ko so na logično 1 postavljene vse vhodne spremenljivke. V logičnem izrazu jo označujemo s simbolom $\wedge$ ali $\cdot$, včasih pa simbol celo izpustimo (podobno kot pri množenju). Slika 1.2 podaja logični izraz

$y = x_1 \vee x_2$



x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

(a) (b) (c)

Slika 1.1 Logični izraz (a), logični simbol (b) in pravilnostna tabela (c) za disjunkcijo z dvema vhodnima spremenljivkama.

(a), logični simbol (b) in pravilnostno tabelo (c) za konjunkcijo z dvema vhodnima spremenljivkama.

$y = x_1 \cdot x_2 = x_1 x_2$

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

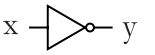
(a) (b) (c)

Slika 1.2 Logični izraz (a), logični simbol (b) in pravilnostna tabela (c) za konjunkcijo z dvema vhodnima spremenljivkama.

1.3.3 Negacija

Negacija (NE, NOT) invertira vhodno logično spremenljivko. V logičnem izrazu jo označujemo s črto nad vhodno spremenljivko. Slika 1.3 podaja logični izraz (a), logični simbol (b) in pravilnostno tabelo (c) za negacijo.

$y = \overline{x}$



x	y
0	1
1	0

(a) (b) (c)

Slika 1.3 Logični izraz (a), logični simbol (b) in pravilnostna tabela (c) za negacijo.

1.3.4 Peircov operator

Peircov operator (NE ALI, NOR) predstavlja invertirano (negirano) disjunkcijo. Operator vrne logično 1, ko so na logično 0 postavljene vse vhodne spremenljivke. V logičnem izrazu ga označujemo s simbolom $\downarrow$. Slika 1.4 podaja logični izraz (a), logični simbol (b) in pravilnostno tabelo (c) za Peircov operator z dvema vhodnima spremenljivkama.

$$y = x_1 \downarrow x_2 = \overline{(x_1 \vee x_2)}$$

(a)

(b)

x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	0

(c)

Slika 1.4 Logični izraz (a), logični simbol (b) in pravilnostna tabela (c) za Peircov operator z dvema vhodnima spremenljivkama.

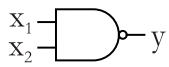
Opozorilo: asociativnost za Peircov operator ne velja: $x_1 \downarrow x_2 \downarrow x_3 \neq (x_1 \downarrow x_2) \downarrow x_3 \neq x_1 \downarrow (x_2 \downarrow x_3)$!

1.3.5 Shefferjev operator

Shefferjev operator (NE IN, NAND) predstavlja invertirano (negirano) konjunkcijo. Operator vrne logično 0, ko so na logično 1 postavljene vse vhodne spremenljivke. V logičnem izrazu ga označujemo s simbolom $\uparrow$. Slika 1.4 podaja logični izraz (a), logični simbol (b) in pravilnostno tabelo (c) za Shefferjev operator z dvema vhodnima spremenljivkama.

$$y = x_1 \uparrow x_2 = \overline{(x_1 x_2)}$$

(a)



(b)

x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	0

(c)

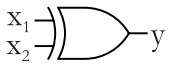
Slika 1.5 Logični izraz (a), logični simbol (b) in pravilnostna tabela (c) za Shefferjev operator z dvema vhodnima spremenljivkama.

Opozorilo: asociativnost za Shefferjev operator ne velja: $x_1 \uparrow x_2 \uparrow x_3 \neq (x_1 \uparrow x_2) \uparrow x_3 \neq x_1 \uparrow (x_2 \uparrow x_3)$!

1.3.6 Ekskluzivni ali

Ekskluzivni ali (XOR, tudi vsota po modulu 2) dveh vhodnih spremenljivk vrne logično 1, ko logično vrednost 1 ekskluzivno zavzema ena vhodna spremenljivka. Pri več vhodnih spremenljivkah funkcija vrne logično 1, ko je na logično vrednost 1 postavljenih liho število vhodnih spremenljivk, kar si lahko interpretiramo tudi kot vsota po modulu 2 (opozorilo: v Logisimu XOR privzeto ne deluje na tak način – pod lastnostmi XOR vrat moramo nastaviti polje *Multiple-Input Behavior* na vrednost *When an odd number are on*). V logičnem izrazu ekskluzivni ALI označujemo s simbolom ∇ ali $\oplus$. Slika 1.6 podaja logični izraz (a), logični simbol (b) in pravilnostno tabelo (c) za ekskluzivni ali z dvema vhodnima spremenljivkama.

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



x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

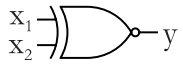
(a) (b) (c)

Slika 1.6 Logični izraz (a), logični simbol (b) in pravilnostna tabela (c) za ekskluzivni ali z dvema vhodnima spremenljivkama.

1.3.7 Ekvivalenca

Ekvivalenca (XNOR) dveh vhodnih spremenljivk vrne logično 1, ko sta vhoda enaka in je tako enaka negiranemu ekskluzivnemu ali. Enakost pa ne velja za več vhodnih spremenljivk. V logičnem izrazu ekvivalenco označujemo s simbolom $\equiv$. Slika 1.7 podaja logični izraz (a), logični simbol (b) in pravilnostno tabelo (c) za ekvivalenco z dvema vhodnima spremenljivkama.

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



x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1

(a) (b) (c)

Slika 1.7 Logični izraz (a), logični simbol (b) in pravilnostna tabela (c) za ekvivalenco z dvema vhodnima spremenljivkama.

1.3.8 Implikacija

Implikacija (IMP) dveh vhodnih spremenljivk, tj. $x_1 \rightarrow x_2$ vrne logično 0 le v primeru, da je operand na levi strani (tj. x_1) enak 1, operator na desni strani (tj. x_2) pa enak 0. Implikacija nima elektronske implementacije znotraj družine 7400. Prav tako operator nima logičnega simbola. Slika 1.8 podaja logični izraz (a), in pravilnostno tabelo (b) za implikacijo.

$$y = x_1 \rightarrow x_2 = \overline{x}_1 \vee x_2$$

(a)

x_1	x_2	y
0	0	1
0	1	1
1	0	0
1	1	1

(b)

Slika 1.8 Logični izraz (a) in pravilnostna tabela (b) za implikacijo.

Opozorilo: komutativnost za implikacijo ali ne velja: $x_1 \rightarrow x_2 \neq x_2 \rightarrow x_1$!

1.4 Analitično reševanje preklopnih funkcij

Pri analitičnemu reševanju preklopnih funkcij preko postulatov in lastnosti Boolove algebre funkcijo podano z logičnim izrazom preoblikujemo v želeno obliko. Kadar želimo določeno lastnost formalno dokazati, se moramo pri spreminjanju oblike analitičnega izraza striktno držati postulatov Boolove algebre, pri čemer uporabimo le en postulat naenkrat, pri vsakem koraku izpeljave pa pripišemo postulat, ki smo ga uporabili.

Zgled 1 *Z uporabo postulatov dokaži enakost* $x \vee x = x$*!*

Rešitev 1

$$\begin{aligned}
& x \vee x = (x \vee x) \cdot 1 && (P2^*) \\
=& (x \vee x) \cdot (x \vee \overline{x}) && (P5) \\
=& x \vee (x \cdot \overline{x}) && (P4^*) \\
=& x \vee 0 && (P5^*) \\
=& x && (P2)
\end{aligned}$$

Navadno pri poenostavljanju izrazov korake izpuščamo in nad izrazom neposredno uporabljamo lastnosti, ki sledijo iz postulatov. Če naloga od nas eksplicitno ne zahteva uporabe postulatov, jo rešujemo na tak način. Ker so postulati in lastnosti

definirani nad osnovnimi logičnimi operatorji (konjunkcija, disjunkcija in negacija), vse funkcije v izrazu najprej izrazimo s temi.

Zgled 2 *Poenostavi izraz* $x_2 \to \left(\overline{(x_1 \vee \overline{x}_3)\,(\overline{x}_1 \vee \overline{x}_3)}\right)!$

Rešitev 2

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

2 Priprava na 2. laboratorijske vaje

2.1 Zapis preklopnih funkcij

Analitični zapis podajamo z logičnim izrazom oziroma logično enačbo. Pogosto se uporabljajo tri posebne oblike analitičnega zapisa, in sicer normalna, popolna in minimalna. Preklopna funkcija je podana v normalni obliki, če je sestavljena iz največ dveh nivojev logičnih operatorjev. Oblika je popolna, če na prvem nivoju logičnih operatorjev v vseh izrazih nastopajo vse vhodne spremenljivke. Minimalna oblika je najkrajša možna oblika zapisa preklopne funkcije.

2.2 Analitični zapis in popolne normalne oblike zapisa preklopnih funkcij

Pogledali si bomo popolno disjunktivno normalno obliko (PDNO) in popolno konjunktivno normalno obliko (PKNO) zapisa preklopne funkcije.

2.2.1 Popolna disjunktivna normalna oblika (PDNO)

- Disjunktivna: operator 2. nivoja je disjunkcija ($\vee$).
- $f(x_1, x_2, ..., x_n) = \vee_{i=0}^{2^n-1} m_i f(\vec{w}_i)$
- $f(\vec{w}_i)$ ... vrednost funkcije pri i-tem vhodnem vektorju (vrstici)
- minterm je konjunktiven izraz - vse vhodne spremenljivke so povezane preko konjunkcije
- m_i ... minterm i; $m_i = x_1^{w_{1,i}} \cdot x_2^{w_{2,i}} \cdot ... \cdot x_n^{w_{n,i}}; i = 0, 1, 2, ..., 2^n - 1$
- $x^w = \begin{cases} x, w = 1 \\ \overline{x}, w = 0 \end{cases}$
- $w_{j,i}$...j-ti bit binarnega zapisa števila i
- PDNO lahko zapišemo v krajši obliki kot $f(x_1, x_2, ..., x_n) = \vee^n(i_1, i_2, ..., i_k)$, kjer $i_1, i_2, ..., i_k$ določajo indekse mintermov, ki nastopajo v PDNO.

Recept: pri določanju PDNO disjunktivno vežemo tiste minterme, pri katerih je funkcijska vrednost 1. Pri tem se i-ti minterm nanaša na funkcijsko vrednost $f(\vec{w}_i)$ (glej tabelo 2.1).

x_1	x_2	x_3	f(x_1,x_2,x_3)	minterm
0	0	0	$f(\vec{w}_0)$	m_0
0	0	1	$f(\vec{w}_1)$	m_1
0	1	0	$f(\vec{w}_2)$	m_2
0	1	1	$f(\vec{w}_3)$	m_3
1	0	0	$f(\vec{w}_4)$	m_4
1	0	1	$f(\vec{w}_5)$	m_5
1	1	0	$f(\vec{w}_6)$	m_6
1	1	1	$f(\vec{w}_7)$	m_7

Tabela 2.1 Primer pravilnostne tabele in zaporedja mintermov za preklopno funkcijo treh vhodnih spremenljivk.

Zgled 3 *Zapiši minterm 9, pri 4-ih vhodnih spremenljivkah:*

Rešitev 3 *Velja torej:*

- $n = 4$,
- $i = 9_{[10]} = 1001_{[2]}$,
- $m_9 = x_1^1 x_2^0 x_3^0 x_4^1 = x_1 \overline{x}_2 \overline{x}_3 x_4$.

2.2.2 Popolna konjunktivna normalna oblika (PKNO)

- Konjunktivna: operator 2. nivoja je konjunkcija (&).
- $f(x_1, x_2, ..., x_n) = \&_{i=0}^{2^n-1} (M_{2^n-1-i} \vee f(\vec{w}_i))$
- maksterm je disjunktiven izraz - vhodne spremenljivke so povezane preko disjunkcije
- M_{2^n-1-i} ... maksterm $2^n - 1 - i$; $M_{2^n-1-i} = x_1^{\overline{w}_{1,i}} \vee x_2^{\overline{w}_{2,i}} \vee ... \vee x_n^{\overline{w}_{n,i}}; i = 0, 1, 2, ..., 2^n - 1$
- PKNO lahko zapišemo v krajši obliki: $f(x_1, x_2, ..., x_n) = \&^n(i_m, i_{m-1}, ..., i_1)$, kjer $i_m, i_{m-1}, ..., i_1$ določajo indekse makstermov, ki nastopajo v PKNO.

Recept: pri določanju PKNO konjunktivno vežemo tiste maksterme, pri katerih je funkcijska vrednost 0. Pri tem se $(2^n - 1 - i)$-ti maksterm nanaša na funkcijsko vrednost $f(\vec{w}_i)$ (glej tabelo 2.2).

x_1	x_2	x_3	f(x_1,x_2,x_3)	maksterm
0	0	0	$f(\vec{w}_0)$	M_7
0	0	1	$f(\vec{w}_1)$	M_6
0	1	0	$f(\vec{w}_2)$	M_5
0	1	1	$f(\vec{w}_3)$	M_4
1	0	0	$f(\vec{w}_4)$	M_3
1	0	1	$f(\vec{w}_5)$	M_2
1	1	0	$f(\vec{w}_6)$	M_1
1	1	1	$f(\vec{w}_7)$	M_0

Tabela 2.2 Primer pravilnostne tabele in zaporedja makstermov za preklopno funkcijo treh vhodnih spremenljivk.

Zgled 4 *Zapiši maksterm 9 pri 4-ih vhodnih spremenljivkah.*

Rešitev 4 *Velja torej:*

- $n = 4$,
- $2^n - 1 - i = 15 - 9 = 6$,
- $2^n - 1 - i = 6_{[10]} = 0110_{[2]}$,
- $M_9 = x_1^{\overline{0}} \vee x_2^{\overline{1}} \vee x_3^{\overline{1}} \vee x_4^{\overline{0}} = x_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_4$.

Zgled 5 *Podano funkcijo pretvori v popolno disjunktivno normalno obliko na dva načina:*

- *s pomočjo pravilnostne tabele,*
- *analitično z razširitvijo.*

Rešitev 5 *Funkcija:*

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

je že zapisana v disjunktivni normalni obliki, ki pa ni popolna.

Pretvorba s pomočjo pravilnostne tabele
Zapišimo pravilnostno tabelo, s pomočjo katere lahko zapišemo PDNO, tako da prepišemo tiste minterme, pri katerih je vrednost funkcije enaka 1.

Če izpišemo samo indekse teh mintermov, dobimo skrajšano obliko PDNO:
$f(x_1, x_2, x_3) = \vee^3(0, 4, 5, 6, 7)$.

x_1	x_2	x_3	f(x_1,x_2,x_3)	minterm	maksterm
0	0	0	1	m_0	M_7
0	0	1	0	m_1	M_6
0	1	0	0	m_2	M_5
0	1	1	0	m_3	M_4
1	0	0	1	m_4	M_3
1	0	1	1	m_5	M_2
1	1	0	1	m_6	M_1
1	1	1	1	m_7	M_0

Tabela 2.3 Pravilnostna tabela funkcije $f(x_1, x_2, x_3) = x_1 \vee x_1\overline{x}_2 \vee \overline{x}_2\overline{x}_3$.

Zapišimo PDNO še v eksplicitni obliki:
$f(x_1, x_2, x_3) = \overline{x}_1\overline{x}_2\overline{x}_3 \vee x_1\overline{x}_2\overline{x}_3 \vee x_1\overline{x}_2x_3 \vee x_1x_2\overline{x}_3 \vee x_1x_2x_3$.

Analitična pretvorba z razširitvijo
Funkcijo razširimo v popolno (na prvem nivoju morajo nastopati vse vhodne spremenljivke):

$$
\begin{aligned}
& x_1 \vee x_1\overline{x}_2 \vee \overline{x}_2\overline{x}_3 \\
=& x_1(x_2 \vee \overline{x}_2)(x_3 \vee \overline{x}_3) \vee x_1\overline{x}_2(x_3 \vee \overline{x}_3) \vee (x_1 \vee \overline{x}_1)\overline{x}_2\overline{x}_3 \\
=& x_1x_2x_3 \vee x_1x_2\overline{x}_3 \vee x_1\overline{x}_2x_3 \vee x_1\overline{x}_2\overline{x}_3 \vee \cancel{x_1\overline{x}_2x_3} \vee \cancel{x_1\overline{x}_2\overline{x}_3} \vee \cancel{x_1\overline{x}_2\overline{x}_3} \vee \overline{x}_1\overline{x}_2\overline{x}_3 \\
=& x_1x_2x_3 \vee x_1x_2\overline{x}_3 \vee x_1\overline{x}_2x_3 \vee x_1\overline{x}_2\overline{x}_3 \vee \overline{x}_1\overline{x}_2\overline{x}_3 \\
=& \vee^3 (7, 6, 5, 4, 0) \\
=& \vee^3 (0, 4, 5, 6, 7)
\end{aligned}
$$

2.2.3 Relaciji med makstermi in mintermi

Med mintermi in makstermi zaradi DeMorganovega pravila veljata sledeči relaciji:

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

Relaciji lahko uporabimo pri pretvarjanju funkcije iz PDNO v PKNO in obratno. Pri podanem zapisu najprej pogledamo kateri termi manjkajo:

- Če je funkcija podana v PDNO, izpišemo manjkajoče minterme - v pravilnostni tabeli so soležni z enicami.
- Če je funkcija podana v PKNO, izpišemo manjkajoče maksterme - v pravilnostni tabeli so soležni z ničlami.

S tem smo dobili pozicije termov, ki nastopajo v iskani obliki zapisa preklopne funkcije. Indekse iskanih termov dobimo tako, da nad izpisanimi mintermi oziroma makstermi uporabimo zgoraj navedeni relaciji.

2.2.4 Pretvarjanje med PDNO in PKNO

Dva možna načina reševanja:

1. zapišemo pravilnostno tabelo in iz nje razberemo rešitev,
2. z upoštevanjem relacij med mintermi in makstermi.

Zgled 6 *Funkcijo podano v PDNO pretvori v PKNO:*

- *s pomočjo pravilnostne tabele,*
- *z upoštevanjem relacij med mintermi in makstermi.*

Rešitev 6 *Demonstrirajmo oba načina pretvorbe.*
Pretvorba iz PDNO v PKNO: s pomočjo pravilnostne tabele
Glej tabelo v prejšnjem zgledu!
Prepišemo maksterme, pri katerih je vrednost funkcije enaka 0.
PKNO: $\&^3(6,5,4)$
Eksplicitna PKNO:
PKNO: $(x_1 \vee x_2 \vee \overline{x}_3)(x_1 \vee \overline{x}_2 \vee x_3)(x_1 \vee \overline{x}_2 \vee \overline{x}_3)$.

Pretvorba iz PDNO v PKNO: z upoštevanjem relacij med mintermi in makstermi
Pretvorimo $\vee^3(0,4,5,6,7)$ *v PKNO:*

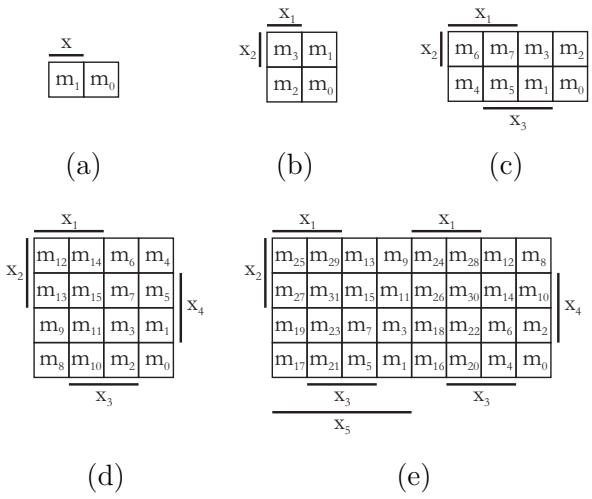
1. *zapišemo manjkajoče minterme:* m_1, m_2, m_3.
2. *izračunamo indekse makstermov:* M_6, M_5, M_4.
3. *rešitev:* $\&^3(6,5,4)$.

Na podoben način, bi lahko pretvarjali tudi iz PKNO v PDNO.

3 Priprava na 3. laboratorijske vaje

3.1 Zapis preklopnih funkcij z Veitchevim diagramom

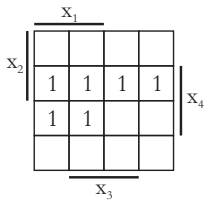
Veitchevi diagrami igrajo pomembno vlogo pri minimizaciji preklopnih funkcij. Veitchev diagram pri n vhodnih spremenljivkah je sestavljen iz 2^n polj, pri čemer posamezno polje vsebuje funkcijsko vrednost pri posameznem mintermu. Postopek zapisovanja Veitchevega diagrama je razmeroma preprost, in sicer izhajamo iz diagrama za eno spremenljivko (glej sliko 3.1(a)). Diagram za dve spremenljivki dobimo tako, da podvojimo diagram za eno spremenljivko. Označiti je potrebno tudi pokritja, in sicer tako, da posamezna spremenljivka pokriva polovico Veitchevega diagrama. Pokritja lahko izbiramo na različne načine. V splošnem dobimo diagram za n spremenljivk tako, da podvojimo diagram za $n-1$ spremenljivk in na novo dodani del pokrijemo z dodano spremenljivko.



Slika 3.1 Slika (a) prikazuje Veitchev diagram za eno vhodno spremenljivko, slika (b) za dve, slika (c) za tri, slika (d) za štiri, slika (e) pa za pet.

Zgled 7 *Funkcijo* $f(x_1, x_2, x_3, x_4) = \vee^4(5, 7, 9, 11, 13, 15)$ *zapiši v Veitchev diagram.*

Rešitev 7 *V polja, ki se nanašajo na minterme, pri katerih je funkcijska vrednost enaka 1, vpišemo enice, ostala polja pa pustimo prazna (glej sliko 3.2).*



Slika 3.2 Veitchev diagram funkcije $f(x_1, x_2, x_3, x_4) = \vee^4(5, 7, 9, 11, 13, 15)$.

3.2 Funkcijsko poln sistem

Funkcijsko poln sistem je nabor logičnih funkcij, s katerimi lahko izrazimo katerokoli logično funkcijo. Iz definicije Boolove algebre sledi, da je $\{\vee, \cdot, \neg\}$ funkcijsko poln sistem (operator $\neg$ predstavlja negacijo).
Funkcijsko polnost ugotavljamo na dva načina:

- s pretvorbo na nek znan funkcijsko poln sistem,
- s preverjanjem pripadnosti osnovnim zaprtim razredom.

Zgled 8 *S pretvorbo na negacijo, konjunkcijo in disjunkcijo pokaži, da je Shefferjev operator funkcijsko poln sistem.*

Rešitev 8 *S Shefferjevim operatorjem izrazimo negacijo.*

$$\overline{x} = \overline{xx} = x \uparrow x$$

S Shefferjevim operatorjem izrazimo konjunkcijo.

$$x_1 x_2 = \overline{\overline{x_1 x_2}} = \overline{x_1 \uparrow x_2} = (x_1 \uparrow x_2) \uparrow (x_1 \uparrow x_2)$$

S Shefferjevim operatorjem izrazimo disjunkcijo.

$$x_1 \vee x_2 = \overline{\overline{x_1 \vee x_2}} = \overline{\overline{x}_1 \overline{x}_2} = (x_1 \uparrow x_1) \uparrow (x_2 \uparrow x_2)$$

Podobno bi se dalo pokazati za Peircov operator.

4 Priprava na 4. laboratorijske vaje

4.1 Zaprti razredi in preverjanje funkcijske polnosti sistema

Funkcijo polnost nabora logičnih funkcij lahko preverjamo s pripadnostjo nabora osnovnim zaprtim razredom logičnih funkcij (Postov teorem). Osnovni zaprti razredi so sledeči:

- T_0 – razred preklopnih funkcij, ki ohranjajo ničlo ($f(0,0,\ldots,0)=0$)
- T_1 – razred preklopnih funkcij, ki ohranjajo enico ($f(1,1,\ldots,1)=1$)
- S – razred sebidualnih funkcij ($\overline{f}(\overline{x}_1,\overline{x}_2,\ldots,\overline{x}_n)=f(x_1,x_2,\ldots,x_n)$)
- L – razred linearnih funkcij ($f(x_1,x_2,...,x_n)\in L$, $f(x_1,x_2,...,x_n)=$ $=a_0\nabla a_1x_2\nabla...\nabla a_nx_n$)
- M – razred monotonih funkcij ($f(x_1,x_2,...,x_n)\in M,\forall i,j:$ $\vec{w}_i<\vec{w}_j\rightarrow f(\vec{w}_i)\leq f(\vec{w}_j)$)

Če nabor odpira vse osnovne razrede, je poln. Nabor odpira nek osnovni razred, če vsaj ena izmed funkcij v podanem naboru ne pripada izbranemu razredu.

Zgled 9 *S pomočjo zaprtih razredov preveri funkcijsko polnost nabora* $\{\vee,\rightarrow,1\}$.

Rešitev 9 *Cilj: Pri vsakem osnovnem razredu želimo najti vsaj eno funkcijo iz podanega nabora, ki temu razredu ne pripada.*

1. *Razred* T_0*:*
   - $\vee: f(0,0)=0\vee 0=0$*; pripada razredu.*
   - $1:1\neq 0$*; ne pripada razredu.*
2. *Razred* T_1*:*
   - $\vee: f(1,1)=1\vee 1=1$*; pripada razredu.*

- $\rightarrow$: $f(1,1) = 1 \rightarrow 1 = 1$; *pripada razredu.*
- $1 : 1 = 1$; *pripada razredu*

Nabor $\{\vee, \rightarrow, 1\}$ *ne odpira razreda* T_1*, torej nabor ni poln. Vseeno nadaljujmo s postopkom.*

3. *Razred* S*:*

   Pripadnost lahko dokazujemo na dva načina:

   - *analitično: ali velja* $f(x_1, x_2, ..., x_n) = \overline{f}(\overline{x}_1, \overline{x}_2, ..., \overline{x}_n)$,
   - *tabelarično: ali velja* $f(\vec{w}_i) \neq f(\vec{w}_{2^n-1-i})$.

   Postopek:

   - 1 : *(analitično)*

$$\begin{aligned} \overline{1} &= 1 \\ 0 &= 1 \quad \textit{protislovje} \end{aligned}$$

     Funkcija ne pripada razredu S*. Vseeno nadaljujmo.*

   - $\vee$: *(analitično)*

$$\begin{aligned} \overline{f}(\overline{x}_1, \overline{x}_2) &= f(x_1, x_2) \\ \overline{\overline{x}_1 \vee \overline{x}_2} &= x_1 \vee x_2 \quad \textit{(desno stran dopolnimo do PDNO)} \\ x_1 x_2 &= x_1(x_2 \vee \overline{x}_2) \vee x_2(x_1 \vee \overline{x}_1) \\ x_1 x_2 &= x_1 x_2 \vee x_1 \overline{x}_2 \vee x_1 x_2 \vee \overline{x}_1 x_2 \\ x_1 x_2 &= \overline{x}_1 x_2 \vee x_1 \overline{x}_2 \vee x_1 x_2 \\ \vee^2(3) &\neq \vee^2(1,2,3) \qquad \rightarrow \vee \notin S \end{aligned}$$

   - $\rightarrow$: *(tabelarično)*

x_1	x_2	$x_1 \rightarrow x_2$
0	0	1
0	1	1
1	0	0
1	1	1

     $f(w_0) = f(w_3)$ – *funkcija ne pripada* S*.*

4. *Razred L* :
Funkcija $f(x_1, x_2, \ldots, x_n)$ spada v razred linearnih funkcij, če jo lahko zapišemo kot

$$f(x_1, x_2, \ldots, x_n) = a_0 \nabla a_1 x_1 \nabla a_2 x_2 \nabla \cdots \nabla a_n x_n.$$

Pripadnost lahko preverjamo:

- *analitično,*
- *z Veitchevim diagramom.*

Analitično preverjanje:

- *predpostavimo, da funkcija $f(x_1, x_2, \ldots, x_n)$ spada v razred linearnih funkcij,*
- *določimo koeficiente $a_0, a_1, \cdots, a_n$,*
- *preverimo, če se dobljena funkcija $a_0 \nabla a_1 x_1 \nabla a_2 x_2 \nabla \cdots \nabla a_n x_n$ ujema s podano funkcijo $f(x_1, x_2, \ldots, x_n)$.*

Postopek:

- $\vee$:
Predpostavimo, da je disjunkcija linearna funkcija, tedaj jo lahko zapišemo kot:

$$f(x_1, x_2)_L = x_1 \vee x_2 = a_0 \nabla a_1 x_1 \nabla a_2 x_2$$

$$\begin{aligned} f(0,0)_L &= a_0 \nabla a_1 0 \nabla a_2 0 = a_0 = 0 \nabla 0 = 0 \quad (a_0 = 0) \\ f(0,1)_L &= 0 \nabla a_1 0 \nabla a_2 1 = a_2 = 0 \nabla 1 = 1 \quad (a_2 = 1) \\ f(1,0)_L &= 0 \nabla a_1 1 \nabla a_2 0 = a_1 = 1 \nabla 0 = 1 \quad (a_1 = 1) \\ f(x_1, x_2)_L &= 0 \nabla 1 x_1 \nabla 1 x_2 = x_1 \nabla x_2 \end{aligned}$$

Preverimo za vse preostale vhodne vektorje (v primeru protislovja postopek ustavimo):

$$\begin{aligned} f(1,1) &= 1 \vee 1 = 1 \\ f(1,1)_L &= x_1 \nabla x_2 = 1 \nabla 1 = 0 \end{aligned}$$

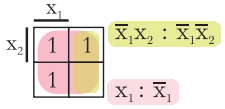
Protislovje. Funkcija ne pripada razredu.

Preverjanje z Veitchevim diagramom:

- *Preklopna funkcija spada v razred L, če pri primerjavi pokritij velja popolna enakost ali popolna različnost vrednosti funkcije.*

Postopek:

- $\vee$:
  Preverimo pokritja:



  - $\overline{x}_1 x_2 : \overline{x}_1 \overline{x}_2$ *(popolnoma različna),*
  - $x_1 : \overline{x}_1$ *(niti popolnoma različna niti popolnoma enaka).*

  Funkcija ne pripada razredu.

5. *Razred M: Funkcija pripada M, če pri vseh vhodnih vektorjih velja, da je pri manjšem vektorju vrednost funkcije manjša ali enaka vrednosti pri večjem vektorju. Relacija* je manjši *je definirana na sledeči način:*

   - *Za vhodna vektorja w_i in w_j velja $w_i < w_j$, če za vsako mesto k velja $w_{k,i} \leq w_{k,j}$. Primer:* $(1,0,0,1,0,1,0,0) < (1,1,0,1,0,1,0,1)$.

   Pri preverjanju pripadnosti je dovolj, da preverimo le sosedne vhodne vektorje. Vhodna vektorja sta sosedna, če se razlikujeta le na enem mestu. Primer: $(1,0,0,1,0)$ *in* $(1,0,1,1,0)$.

   Postopek:

   - $\rightarrow$:

x_1	x_2	$x_1 \rightarrow x_2$
0	0	1
0	1	1
1	0	0
1	1	1

   Preverjamo samo sosedne vektorje. Sosedni vhodni vektorji so (w_0, w_1), (w_0, w_2), (w_1, w_3) *in* (w_2, w_3). *Hitro najdemo protislovje, in sicer* $w_0 < w_2, f(w_0) > f(w_2)$. *Funkcija torej ne pripada M.*

Zgled 10 *Analitično preveri ali preklopna funkcija* $f(x_1, x_2, x_3, x_4) = \&^4(0, 1, 6, 8, 9, 14)$ *pripada razredu linearnih funkcij.*

x_1	x_2	x_3	x_4	$f(x_1,x_2,x_3,x_4)$	$f(x_1,x_2,x_3,x_4)_L$
0	0	0	0	1	1
0	0	0	1	0	0
0	0	1	0	1	1
0	0	1	1	1	0
0	1	0	0	1	1
0	1	0	1	1	0
0	1	1	0	0	1
0	1	1	1	0	0
1	0	0	0	1	1
1	0	0	1	0	0
1	0	1	0	1	1
1	0	1	1	1	0
1	1	0	0	1	1
1	1	0	1	1	0
1	1	1	0	0	1
1	1	1	1	0	0

Rešitev 10 *Postopek:*
Predpostavimo, da dana funkcija pripada razredu linearnih funkcij, tedaj jo lahko zapišemo kot:

$$f(x_1,x_2,x_3,x_4)_L = a_0 \nabla a_1 x_1 \nabla a_2 x_2 \nabla a_3 x_3 \nabla a_4 x_4$$

$$\begin{aligned}
f(0,0,0,0)_L &= a_0 \nabla a_1 0 \nabla a_2 0 \nabla a_3 0 \nabla a_4 0 = 1 \quad (a_0 = 1) \\
f(0,0,0,1)_L &= 1 \nabla a_1 0 \nabla a_2 0 \nabla a_3 0 \nabla a_4 1 = 1 \nabla a_4 = 0 \quad (a_4 = 1) \\
f(0,0,1,0)_L &= 1 \nabla a_1 0 \nabla a_2 0 \nabla a_3 1 \nabla a_4 0 = 1 \nabla a_3 = 1 \quad (a_3 = 0) \\
f(0,1,0,0)_L &= 1 \nabla a_1 0 \nabla a_2 1 \nabla a_3 0 \nabla a_4 0 = 1 \nabla a_2 = 1 \quad (a_2 = 0) \\
f(1,0,0,0)_L &= 1 \nabla a_1 1 \nabla a_2 0 \nabla a_3 0 \nabla a_4 0 = 1 \nabla a_1 = 1 \quad (a_1 = 0) \\
f(x_1,x_2,x_3,x_4)_L &= 1 \nabla 0 x_1 \nabla 0 x_2 \nabla 0 x_3 \nabla 1 x_4 = 1 \nabla x_4 = \overline{x}_4
\end{aligned}$$

Preverimo za vse preostale vhodne vektorje (v primeru protislovja postopek ustavimo):

$$\begin{aligned}
f(0,0,1,1) &= 1 \\
f(0,0,1,1)_L &= 1 \nabla 1 = 0
\end{aligned}$$

Protislovje. Funkcija ne pripada razredu.

5 Priprava na 5. laboratorijske vaje

5.1 Minimalne oblike zapisa preklopnih funkcij

Minimalna oblika zapisa preklopne funkcije podaja najkrajši možen zapis te funkcije. Poznamo več minimalnih oblik zapisa. Pogledali si bomo sledeče:

- minimalna disjunktivna normalna oblika (MDNO): določa najkrašo disjunktivno normalno obliko zapisa preklopne funkcije,
- minimalna konjunktivna normalna oblika (MKNO): določa najkrašo konjunktivno normalno obliko zapisa preklopne funkcije,
- minimalna normalna oblika (MNO): določa nakrajšo normalno obliko zapisa preklopne funkcije.

Pri določanju minimalne disjunktivne normalne oblike (MDNO) izhajamo iz iskanja *glavnih vsebovalnikov*. Glavni vsebovalnik predstavlja najkrajši konjunktivni izraz, ki je skupen podmnožici mintermov, ki sestavljajo PDNO podane funkcije. Najmanjša množica glavnih vsebovalnikov, ki skupaj pokrijejo celotno množico mintermov v PDNO, predstavlja minimalno disjunktivno normalno obliko.

Pri določanju minimalne konjunktivne normalne oblike (MKNO) izhajamo iz določanja MDNO negirane funkcije, ki jo z uporabo DeMorganovega pravila pripeljemo do konjunktivne oblike.

Pri določanju minimalne normalne oblike (MNO) upoštevamo dejstvo, da MNO predstavlja krajšo izmed MDNO in MKNO.

Glavne vsebovalnike iščemo na podlagi *sosednosti* med konjunktivni izrazi. Dva konjunktivna izraza sta sosedna, če se razlikujeta po natanko eni negaciji:

- izraza $x_1^{w_1} \cdot x_2^{w_2} \cdot ... \cdot x_i^{w_i} \cdot ... \cdot x_n^{w_n}$ in $x_1^{w_1} \cdot x_2^{w_2} \cdot ... \cdot x_i^{\overline{w}_i} \cdot ... \cdot x_n^{w_n}$ sta sosedna, ker se razlikujeta le po negaciji nad vhodno spremenljivko z indeksom i.

V splošnem velja, da ima izraz, ki vsebuje n vhodnih spremenljivk, n sosednih izrazov.

Zgled 11 *Zapiši vse izraze, ki so sosedni izrazu* $x_1\overline{x}_2x_4$.

Rešitev 11 *V sosednih izrazih nastopajo enake spremenljivke kot v izhodiščnem izrazu. Od izhodiščnega izraza se sosedni razlikujejo po natanko eni negaciji. Sosedni izrazi so torej:*

- $\overline{x}_1\overline{x}_2x_4$,
- $x_1x_2x_4$ *in*
- $x_1\overline{x}_2\overline{x}_4$.

Glavni vsebovalnik dveh sosednih izrazov določimo z upoštevanjem lastnosti, da je disjunkcija dveh termov, ki se razlikujeta natanko po negaciji nad eno spremenljivko, neodvisna od vrednosti te spremenljivke. Glavni vsebovalnik dveh sosednih izrazov $x_1^{w_1} \cdot x_2^{w_2} \cdot \ldots \cdot x_{i-1}^{w_{i-1}} \cdot x_i^{w_i} \cdot x_{i+1}^{w_{i+1}} \cdot \ldots \cdot x_n^{w_n}$ in $x_1^{w_1} \cdot x_2^{w_2} \cdot \ldots \cdot x_{i-1}^{w_{i-1}} \cdot x_i^{\overline{w}_i} \cdot x_{i+1}^{w_{i+1}} \cdot \ldots \cdot x_n^{w_n}$ je torej izraz $x_1^{w_1} \cdot x_2^{w_2} \cdot \ldots \cdot x_{i-1}^{w_{i-1}} \cdot x_{i+1}^{w_{i+1}} \cdot \ldots \cdot x_n^{w_n}$.

Zgled 12 *Določi glavni vsebovalnik izrazov* $\overline{x}_1\overline{x}_2\overline{x}_3x_4$ *in* $x_1\overline{x}_2\overline{x}_3x_4$.

Rešitev 12 *Izraza se razlikujeta le po negaciji nad spremenljivko* x_1. *Disjunkcija med izrazoma je torej neodvisna od vrednosti te spremenljivke, zato je njun glavni vsebovalnik* $\overline{x}_2\overline{x}_3x_4$.

5.2 Veitchev postopek minimizacije

Veitchev postopek minimizacije izkorišča lastnosti Veitchevega diagrama. Sosednost mintermov je namreč neposredno povezana s sosednostjo celic v diagramu. Pri celicah, ki se nahajajo na robovih diagrama se sosednost prenaša na zrcalno stran (glej sliko 5.1).

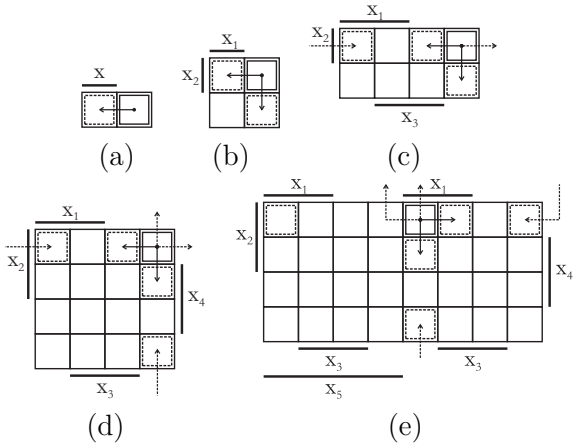
Zgled 13 *V Veitchevem diagramu za 4 spremenljivke označi izraze, ki so sosednji izrazu* $m_{14} \vee m_{15}$.

Rešitev 13 *Izraz zapišimo v razširjeni obliki:*

$$m_{14} \vee m_{15} = x_1x_2x_3\overline{x}_4 \vee x_1x_2x_3x_4 = x_1x_2x_3$$

Sosedni izrazi so torej:

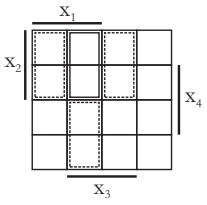
- $\overline{x}_1x_2x_3 = \overline{x}_1x_2x_3\overline{x}_4 \vee \overline{x}_1x_2x_3x_4 = m_6 \vee m_7$



Slika 5.1 Slika (a) prikazuje sosednost v Veitchevem diagramu za 1 vhodno spremenljivko, slika (b) za 2, slika (c) za 3, slika (d) za 4, slika (e) pa za 5.

- $x_1x_2\overline{x}_3 = x_1x_2\overline{x}_3\overline{x}_4 \vee x_1x_2\overline{x}_3x_4 = m_{12} \vee m_{13}$
- $x_1\overline{x}_2x_3 = x_1\overline{x}_2x_3\overline{x}_4 \vee x_1\overline{x}_2x_3x_4 = m_{10} \vee m_{11}$

Veitchev diagram z označemi sosednimi izrazi prikazuje slika 5.2.



Slika 5.2 Izrazi, ki so sosedni izrazu $m_{14} \vee m_{15}$.

Zgled 14 *V Veitchevem diagramu za 5 spremenljivk označi izraze, ki so sosednji izrazu* $m_{22} \vee m_{30}$.

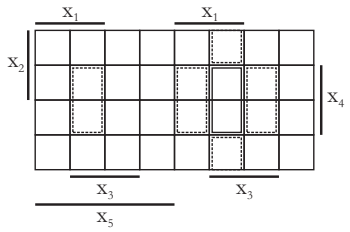
Rešitev 14 *Izraz zapišimo v razširjeni obliki:*

$$m_{22} \vee m_{30} = x_1\overline{x}_2x_3x_4\overline{x}_5 \vee x_1x_2x_3x_4\overline{x}_5 = x_1x_3x_4\overline{x}_5$$

Sosedni izrazi so torej:

- $\overline{x}_1 x_3 x_4 \overline{x}_5$
- $x_1 \overline{x}_3 x_4 \overline{x}_5$
- $x_1 x_3 \overline{x}_4 \overline{x}_5$
- $x_1 x_3 x_4 x_5$

Veitchev diagram z označemi sosednimi izrazi prikazuje slika 5.3.



Slika 5.3 Izrazi, ki so sosedni izrazu $m_{22} \vee m_{30}$.

5.2.1 Določanje minimalne normalne oblike z Veitchevim diagramom

Za minimalno normalno obliko (MNO) velja, da predstavlja krajšo izmed minimalne disjunktivne normalne oblike (MDNO) in minimalne konjunktivne normalne oblike (MKNO). Za določitev MNO moramo torej najprej določiti MDNO in MKNO.

Določanje minimalne disjunktivne normalne oblike z Veitchevim diagramom

Postopek določanja je sledeč:

1. Funkcijo predstavimo z Veitchevim diagramom.
2. Poiščemo glavne vsebovalnike, tako da med seboj združujemo čim večje število sosednih mintermov, pri katerih je funkcijska vrednost enaka 1. Združujemo lahko 1,2,4,8,16,... mintermov. Iščemo najmanjši nabor pokritij, s katerim pokrijemo vse enice. V vsakem koraku poskušamo dobiti čim večje pokritje, saj s tem izločimo večje število vhodnih spremenljivk.
3. Zapišemo MDNO na podlagi glavnih vsebovalnikov.

Določanje minimalne konjunktivne normalne oblike z Veitchevim diagramom

1. Negirano funkcijo predstavimo z Veitchevim diagramom.
2. Poiščemo MDNO negirane funkcije.
3. Negiramo MDNO negirane funkcije, s čimer dobimo izhodiščno funkcijo.
4. Z upoštevanjem DeMorganovega pravila funkcijo prevedemo v MKNO.

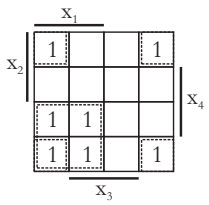
Določanje minimalne normalne oblike

1. Določimo število operatorjev (logičnih vrat) in operandov (vhodov) za MDNO in za MKNO posebej. Negacij ne štejemo.
2. Minimalna oblika je tista, ki ima manjše število operatorjev. Če je število operatorjev pri obeh enako, je minimalna tista, ki ima manjše število operandov.

Zgled 15 *Za funkcijo* $f(x_1, x_2, x_3, x_4) = \vee^4(0, 4, 8, 9, 10, 11, 12)$ *določi minimalno normalno obliko.*

Rešitev 15 *Najprej določimo MDNO:*

1. *Narišemo Veitchev diagram funkcije in označimo glavne vsebovalnike (Slika 5.4).*

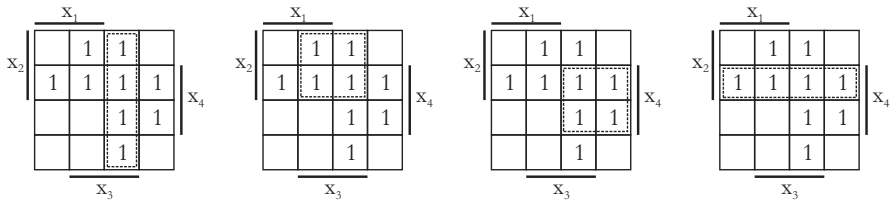


Slika 5.4 Glavni vsebovalniki funkcije $\vee^4(0, 4, 8, 9, 10, 11, 12)$.

2. *Izpišemo glavne vsebovalnike in s tem MDNO:*

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

Potem določimo MKNO:



Slika 5.5 Glavni vsebovalniki funkcije $\vee^4(1,2,3,5,6,7,13,14,15)$. Slika (a) prikazuje glavni vsebovalnik $\overline{x}_1x_3$, slika (b) x_2x_3, slika (c) $\overline{x}_1x_4$, slika (d) pa x_2x_4.

1. *Narišemo Veitchev diagram negirane funkcije* $\overline{f(x_1,x_2,x_3,x_4)}=$ $\vee^4(1,2,3,5,6,7,13,14,15)$ *in označimo glavne vsebovalnike (Slika 5.5).*

2. *Izpišemo glavne vsebovalnike, ki določajo negirano funkcijo:* $\overline{f(x_1,x_2,x_3,x_4)} = \overline{x}_1x_3 \vee x_2x_3 \vee \overline{x}_1x_4 \vee x_2x_4$. *S tem dobimo MDNO negirane funkcije.*

3. *Dobljeno MDNO negiramo, uporabimo DeMorganovo pravilo in dobimo MKNO:* $f(x_1,x_2,x_3,x_4) = \overline{\overline{x}_1x_3 \vee x_2x_3 \vee \overline{x}_1x_4 \vee x_2x_4} = (x_1 \vee \overline{x}_3)(\overline{x}_2 \vee \overline{x}_3)(x_1 \vee \overline{x}_4)(\overline{x}_2 \vee \overline{x}_4)$.

Sedaj lahko določimo MNO:

1. *Določimo število operatorjev pri MDNO (za realizacijo potrebujemo dvoja AND vrata in ena OR vrata):* $2+1=3$.

2. *Določimo število operandov pri MDNO (2 vhoda v prva AND vrata, 2 vhoda v druga AND vrata, 2 vhoda v OR vrata):* $2+2+2=6$.

3. *Zapišemo par število operatorjev/število operandov za MDNO:* $[3,6]$.

4. *Določimo število operatorjev pri MKNO (za realizacijo potrebujemo ena AND vrata in štiri OR vrata):* $1+4=5$.

5. *Določimo število operandov pri MKNO (AND vrata so 4-vhodna, vsa OR vrata pa 2-vhodna):* $4+4\cdot 2=12$.

6. *Zapišemo par število operatorjev/število operandov za MKNO:* $[5,12]$.

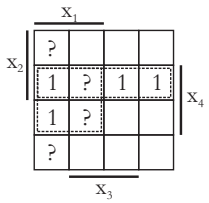
7. *Para leksikografsko primerjam med seboj. Ker ima MDNO manjše število operatorjev predstavlja MNO funkcije. MNO je torej* $f(x_1,x_2,x_3,x_4) = x_1\overline{x}_2 \vee \overline{x}_3\overline{x}_4$.

Zgled 16 *Minimiziraj nepopolno preklopno funkcijo* $f^4 = \vee^4(5,7,9,13) \vee^4_?(8,11,12,15)$.

***Opomba**: preklopna funkcija je nepopolna, če funkcijskih vrednosti nima določenih pri vseh vhodnih vektorjih.*

Rešitev 16 *Določimo MDNO:*

1. *Narišemo Veitchev diagram funkcije in označimo glavne vsebovalnike (slika 5.6) Pri tem vprašaje pokrijemo po potrebi.*

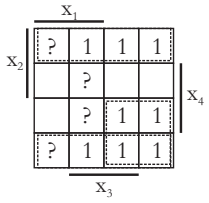


Slika 5.6 Glavni vsebovalniki funkcije $f^4 = \vee^4(5,7,9,13) \vee^4_? (8,11,12,15)$.

2. *Izpišemo MDNO:* $f^4 = x_1x_4 \vee x_2x_4$.
3. *Določimo število operatorjev in operandov:* $[3,6]$.

Določimo MKNO:

1. *Narišemo Veitchev diagram negirane funkcije (kjer so bila prej prazna polja pišemo enice, kjer so bile prej enice pustimo prazno, kjer so bili prej vprašaji, pustimo vprašaje - slika 5.7). Označimo glavne vsebovalnike.*



Slika 5.7 Glavni vsebovalniki funkcije $f^4 = \vee^4(0,1,2,3,4,6,10,14) \vee^4_? (8,11,12,15)$.

2. *Izpišemo MDNO negirane funkcije:* $\overline{f^4} = \overline{x}_4 \vee \overline{x}_1\overline{x}_2$.

3. Določimo MKNO: $f^4 = \overline{\overline{x}_4 \vee \overline{x}_1\overline{x}_2} = x_4(x_1 \vee x_2)$.

4. Določimo število operatorjev in operandov: $[2, 4]$.

Določimo MNO:

- *Ker ima MKNO manjše število operandov je MNO zapis:*$f^4 = x_4(x_1 \vee x_2)$.

6 Priprava na 6. laboratorijske vaje

6.1 Simetrične preklopne funkcije

Funkcija je popolnoma simetrična, če lahko zamenjamo poljubni dve vhodni spremenljivki in se funkcijska vrednost ne spremeni. Naj ima funkcija f izhodno vrednost 1 pri vsakem tistem vhodnem vektorju, pri katerem ima natanko a vhodnih spremenljivk vrednost 1. Število a tedaj imenujemo *simetrijsko število* funkcije f. Če za funkcijo f obstaja neprazna množica simetrijskih števil, je funkcija *popolnoma simetrična*.

Zgled 17 *Zapiši PDNO funkcije* $f_{\{0,2\}}(x_1, x_2, x_3)$.

Rešitev 17 *Pomagamo si s pravilnostno tabelo:*

x_1	x_2	x_3	$f_{\{0,2\}}(x_1, x_2, x_3)$
0	*0*	*0*	*1*
0	*0*	*1*	*0*
0	*1*	*0*	*0*
0	*1*	*1*	*1*
1	*0*	*0*	*0*
1	*0*	*1*	*1*
1	*1*	*0*	*1*
1	*1*	*1*	*0*

PDNO lahko preberemo neposredno iz tabele:

$$f(x_1, x_2, x_3) = \vee^3(0, 3, 5, 6).$$

Zgled 18 *Zapiši PDNO funkcije* $f_{\{0,2\}}(x_1, \overline{x}_2, x_3)$.

Rešitev 18 *Pomagamo si s pravilnostno tabelo:*

x_1	x_2	x_3	x_1	$\overline{x}_2$	x_3	$f_{\{0,2\}}(x_1, \overline{x}_2, x_3)$
0	*0*	*0*	*0*	*1*	*0*	*0*
0	*0*	*1*	*0*	*1*	*1*	*1*
0	*1*	*0*	*0*	*0*	*0*	*1*
0	*1*	*1*	*0*	*0*	*1*	*0*
1	*0*	*0*	*1*	*1*	*0*	*1*
1	*0*	*1*	*1*	*1*	*1*	*0*
1	*1*	*0*	*1*	*0*	*0*	*0*
1	*1*	*1*	*1*	*0*	*1*	*1*

PDNO lahko preberemo neposredno iz tabele:

$$f(x_1, x_2, x_3) = \vee^3(1, 2, 4, 7).$$

6.2 Quineova metoda minimizacije

Medtem, ko so Veitchevi diagrami zelo primerni za ročno minimizacijo funkcij z manjšim številom vhodnih spremenljivk (do vključno 5), je za večje število vhodnih spremenljivk zelo zaželena avtomatizacija iskanja minimalne oblike. Pri tem se zelo dobro obnese Quineova (tudi Quine–McCluskey) metoda, ki za minimizacijo uporablja tabelaričen postopek, ki ga je enostavno sprogramirati. Metoda temelji na iskanju potrebnih glavnih vsebovalnikov na podalgi podobnega postopka kot Veitcheva metoda, le da pri tem uporablja drugačna orodja.

6.2.1 Določanje MDNO

- Preklopno funkcijo zapišemo v popolni disjunktivni normalni obliki (PDNO).
- Poiščemo vse sosednje minterme in njihove glavne vsebovalnike:
  1. Narišemo tabelo z n stolpci, pri čemer je n število vhodnih spremenljivk. V prvega vpišemo vse minterme, ki določajo preklopno funkcijo.
  2. Izraze v stolpcu medsebojno primerjamo in ugotavljamo sosednost. Primerjamo vsakega z vsakim.
  3. Sosedna izraza prečrtamo in v naslednji stolpec vpišemo njun glavni vsebovalnik.
  4. Ponavljamo koraka 2 in 3, dokler ne gremo čez vse kombinacije. Pri tem upoštevamo tudi že prečrtane izraze.
  5. Če v predhodnem koraku nismo našli nobenega vsebovalnika ali pa smo prišli v zadnji stolpec zaključimo.
- Izpišemo samo potrebne glavne vsebovalnike:

1. Narišemo tabelo pokritij z vrsticami, ki predstavljajo glavne vsebovalnike (samo tiste, ki niso prečrtani) in stolpci, ki predstavljajo minterme.
2. Za vsak glavni vsebovalnik označimo minterme, ki jih pokriva.
3. Poiščemo najmanjšo množico glavnih vsebovalnikov, ki skupaj pokrijejo vse minterme.

Zgled 19 *Preklopno funkcijo* $f = \vee^4(1,4,6,7,8,9,10,11,15)$ *zapiši v MDNO s Quineovo metodo minimizacije.*

Rešitev 19 *S pomočjo Quineove metode zgradimo sledečo tabelo:*

4		*3*		*2*		*1*
(1)	$\overline{x}_1\overline{x}_2\overline{x}_3x_4$	*(1,6)*	$\overline{x}_2\overline{x}_3x_4$	*(5,8)*	$x_1\overline{x}_2$	
(2)	$\overline{x}_1x_2\overline{x}_3\overline{x}_4$	*(2,3)*	$\overline{x}_1x_2\overline{x}_4$	*(6,7)*		
(3)	$\overline{x}_1x_2x_3\overline{x}_4$	*(3,4)*	$\overline{x}_1x_2x_3$			
(4)	$\overline{x}_1x_2x_3x_4$	*(4,9)*	$x_2x_3x_4$			
(5)	$x_1\overline{x}_2\overline{x}_3\overline{x}_4$	*(5,6)*	$x_1\overline{x}_2\overline{x}_3$			
(6)	$x_1\overline{x}_2\overline{x}_3x_4$	*(5,7)*	$x_1\overline{x}_2\overline{x}_4$			
(7)	$x_1\overline{x}_2x_3\overline{x}_4$	*(6,8)*	$x_1\overline{x}_2x_4$			
(8)	$x_1\overline{x}_2x_3x_4$	*(7,8)*	$x_1\overline{x}_2x_3$			
(9)	$x_1x_2x_3x_4$	*(8,9)*	$x_1x_3x_4$			

Zgradimo tabelo pokritij:

		m_1	m_4	m_6	m_7	m_8	m_9	m_{10}	m_{11}	m_{15}
✓	$\overline{x}_2\overline{x}_3x_4$	✓					✓			
✓	$\overline{x}_1x_2\overline{x}_4$		✓	✓						
	$\overline{x}_1x_2x_3$			✓	✓					
✓	$x_2x_3x_4$				✓					✓
	$x_1x_3x_4$								✓	✓
✓	$x_1\overline{x}_2$					✓	✓	✓	✓	

Iz tabele določimo potrebne glavne vsebovalnike: $f_{MDNO}(x_1,x_2,x_3,x_4) = x_1\overline{x}_2 \vee x_2x_3x_4 \vee \overline{x}_1x_2\overline{x}_4 \vee \overline{x}_2\overline{x}_3x_4$.

6.2.2 Določanje MKNO

Postopek je podoben kot pri Veitchevi minimizaciji:

1. funkcijo negiramo,
2. s Quineovo metodo izračunamo MDNO negirane funkcije,
3. rezultat negiramo in z DeMorganovim pravilom pretvorimo v MKNO.

Zgled 20 *Preklopno funkcijo* $f = \vee^4(1,4,6,7,8,9,10,11,15)$ *zapiši v MKNO s Quineovo metodo minimizacije.*

Rešitev 20 *Funkcijo najprej negiramo:* $\overline{f} = \vee^4(0,2,3,5,12,13,14)$.

S Quineovo metodo zgradimo sledečo tabelo:

	4		*3*
(1)	$\overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4$	*(1,2)*	$\overline{x}_1\overline{x}_2\overline{x}_4$
(2)	$\overline{x}_1\overline{x}_2x_3\overline{x}_4$	*(2,3)*	$\overline{x}_1\overline{x}_2x_3$
(3)	$\overline{x}_1\overline{x}_2x_3x_4$	*(4,6)*	$x_2\overline{x}_3x_4$
(4)	$\overline{x}_1x_2\overline{x}_3x_4$	*(5,6)*	$x_1x_2\overline{x}_3$
(5)	$x_1x_2\overline{x}_3\overline{x}_4$	*(5,7)*	$x_1x_2\overline{x}_4$
(6)	$x_1x_2\overline{x}_3x_4$		
(7)	$x_1x_2x_3\overline{x}_4$		

Zgradimo tabelo pokritij:

		m_0	m_2	m_3	m_5	m_{12}	m_{13}	m_{14}
✓	$\overline{x}_1\overline{x}_2\overline{x}_4$	✓	✓					
✓	$\overline{x}_1\overline{x}_2x_3$		✓	✓				
✓	$x_2\overline{x}_3x_4$				✓		✓	
	$x_1x_2\overline{x}_3$					✓	✓	
✓	$x_1x_2\overline{x}_4$					✓		✓

Iz tabele določimo potrebne glavne vsebovalnike:
$\overline{f}_{MDNO}(x_1,x_2,x_3,x_4) = \overline{x}_1\overline{x}_2\overline{x}_4 \vee \overline{x}_1\overline{x}_2x_3 \vee x_2\overline{x}_3x_4 \vee x_1x_2\overline{x}_4$

Funkcijo ponovno negiramo in jo preko DeMorganovega pravila pretvorimo v konjunktivno normalno obliko:

$$\begin{aligned}\overline{\overline{f}}_{MDNO}(x_1,x_2,x_3,x_4) = f_{MKNO}(x_1,x_2,x_3,x_4) &= \overline{\overline{x}_1\overline{x}_2\overline{x}_4 \vee \overline{x}_1\overline{x}_2x_3 \vee x_2\overline{x}_3x_4 \vee x_1x_2\overline{x}_4} \\ &= (x_1 \vee x_2 \vee x_4)(x_1 \vee x_2 \vee \overline{x}_3)(\overline{x}_2 \vee x_3 \vee \overline{x}_4)(\overline{x}_1 \vee \overline{x}_2 \vee x_4)\end{aligned}$$

7 Priprava na 7. laboratorijske vaje

7.1 Ločenje preklopnih funkcij

Ločenje preklopne funkcije po spremenljivki x_i je definirano z izrazom

$$f(x_1, x_2, ..., x_{i-1}, x_i, x_{i+1}, ..., x_n) = \\ f(x_1, x_2, ..., x_{i-1}, 0, x_{i+1}, ..., x_n)\overline{x}_i \vee f(x_1, x_2, ..., x_{i-1}, 1, x_{i+1}, ..., x_n)x_i$$

za ločenje konjunktivnih izrazov in z izrazom

$$f(x_1, x_2, ..., x_{i-1}, x_i, x_{i+1}, ..., x_n) = \\ (f(x_1, x_2, ..., x_{i-1}, 0, x_{i+1}, ..., x_n) \vee x_i)(f(x_1, x_2, ..., x_{i-1}, 1, x_{i+1}, ..., x_n) \vee \overline{x}_i)$$

za ločenje disjunktivnih izrazov. Pri tem funkcijama, ki nastopata v izrazu, pravimo *funkcijska ostanka*:

$$f_0(x_1, x_2, ..., x_{i-1}, x_{i+1}, ..., x_n) = f(x_1, x_2, ..., x_{i-1}, 0, x_{i+1}, ..., x_n),$$

$$f_1(x_1, x_2, ..., x_{i-1}, x_{i+1}, ..., x_n) = f(x_1, x_2, ..., x_{i-1}, 1, x_{i+1}, ..., x_n).$$

Za realizacijo preklopnih funkcij z multiplekserji uporabljamo konjunktivno ločevanje izrazov, zato bomo v nadaljevanju obravnavali le tega.

Zgled 21 *Funkcijo* $f(x_1, x_2, x_3) = \vee^3(0, 2, 5, 6, 7)$ *loči preko spremenljivke* x_1.

Rešitev 21 *Funkcijo lahko zapišemo kot:*

$$\begin{aligned} f(x_1, x_2, x_3) = & \quad \overline{x}_1\overline{x}_2\overline{x}_3 \vee \overline{x}_1x_2\overline{x}_3 \vee x_1\overline{x}_2x_3 \vee x_1x_2\overline{x}_3 \vee x_1x_2x_3 \\ = & \quad (\overline{0}\overline{x}_2\overline{x}_3 \vee \overline{0}x_2\overline{x}_3 \vee 0\overline{x}_2x_3 \vee 0x_2\overline{x}_3 \vee 0x_2x_3)\overline{x}_1 \vee \\ & \quad \vee(\overline{1}\overline{x}_2\overline{x}_3 \vee \overline{1}x_2\overline{x}_3 \vee 1\overline{x}_2x_3 \vee 1x_2\overline{x}_3 \vee 1x_2x_3)x_1 \\ = & \quad (\overline{x}_2\overline{x}_3 \vee x_2\overline{x}_3)\overline{x}_1 \vee (\overline{x}_2x_3 \vee x_2\overline{x}_3 \vee x_2x_3)x_1 \end{aligned}$$

Pri tem dobimo funkcijska ostanka:

$$f_0(x_2, x_3) = \overline{x}_2\overline{x}_3 \vee x_2\overline{x}_3$$
$$f_1(x_2, x_3) = \overline{x}_2 x_3 \vee x_2\overline{x}_3 \vee x_2 x_3$$

Ločimo funkcijski ostanek $f_0(x_2, x_3)$ še po x_2:

$$\begin{aligned} f_0(x_2, x_3) &= (\overline{0}\overline{x}_3 \vee 0\overline{x}_3)\overline{x}_2 \vee (\overline{1}\overline{x}_3 \vee 1\overline{x}_3)x_2 \\ &= \overline{x}_3\overline{x}_2 \vee \overline{x}_3 x_2 \end{aligned}$$

Pri tem dobimo funkcijska ostanka:

$$f_{00}(x_3) = \overline{x}_3$$
$$f_{01}(x_3) = \overline{x}_3$$

Ločimo funkcijski ostanek $f_1(x_2, x_3)$ še po x_2:

$$\begin{aligned} f_1(x_2, x_3) &= (\overline{0}x_3 \vee 0\overline{x}_3 \vee 0x_3)\overline{x}_2 \vee (\overline{1}x_3 \vee 1\overline{x}_3 \vee 1x_3)x_2 \\ &= x_3\overline{x}_2 \vee (\overline{x}_3 \vee x_3)x_2 \\ &= x_3\overline{x}_2 \vee 1x_2 \end{aligned}$$

Pri tem dobimo funkcijska ostanka:

$$f_{10}(x_3) = x_3$$
$$f_{11}(x_3) = 1$$

7.2 Multiplekser

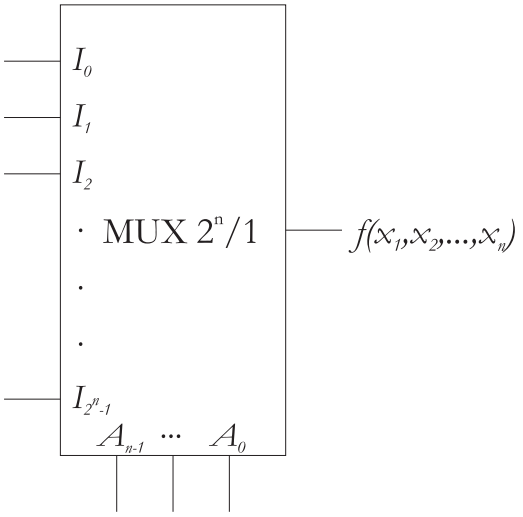
Multiplekser spada v družino strukturalnih preklopnih vezij. Multiplekser ima dva niza vhodnih spremenljivk, in sicer *naslovne spremenljivke* na naslovnih vhodih $(A_n, A_{n-1}, ..., A_0)$ in *podatkovne spremenljivke* na podatkovnih vhodih $(I_0, I_1, ..., I_{2^n-1})$. Na podlagi vrednosti naslovnih spremenljivk se izbere podatkovni vhod, ki se bo preslikal na izhod multiplekserja. Multiplekser v kombinaciji s konstanto 0 ali 1 predstavlja funkcijsko poln sistem - z njim oziroma s kombinacijo več multiplekserjev lahko realiziramo poljubno logično funkcijo.

Multiplekser v odvisnosti od vrednosti naslovnih vhodov na izhod preslika enega izmed podatkovnih vhodov. Natančneje, na izhod se preslika podatkovni vhod z indeksom, ki je enak desetiškemu zapisu vrednosti na naslovnih vhodih. Splošno delovanje multiplekserja prikazuje tabela 7.1.

Multiplekser z n naslovnimi spremenljivkami označujemo kot MUX $2^n/1$. Realizacijo funkcije z multiplekserjem praviloma podajamo z logično shemo. Splošno logično shemo za multiplekser MUX $2^n/1$ prikazuje slika 7.1.

A_{n-1}	A_{n-2}	...	A_0	f
0	0	...	0	I_0
0	0	...	1	I_1
.	.	...	.	.
.	.	...	.	.
.	.	...	.	.
1	1	...	0	I_{2^n-2}
1	1	...	1	I_{2^n-1}

Tabela 7.1 Splošno delovanje multiplekserja MUX $2^n/1$.



Slika 7.1 Splošna logična shema za multiplekser MUX $2^n/1$.

7.3 Realizacija preklopnih funkcij z multiplekserji

Z uporabo konstant 0 in 1 in multiplekserja MUX $2^n/1$ lahko realiziramo poljubno preklopno funkcijo z $n+1$ vhodnimi spremenljivkami. Realizacija funkcij z uporabo multiplekserjev z manjšim številom naslovnih vhodov poteka z drevesno oziroma kaskadno vezavo multiplekserjev.

Realizacija preklopnih funkcij z multiplekserji poteka na podlagi ločenja. Funkcijo ločimo preko vhodnih spremenljivk, ki jih vežemo na naslovne vhode, na podatkovnih vhodih pa realiziramo funkcijske ostanke.

V nadaljevanju bomo demonstrirali realizacijo funkcije $f(x_1, x_2, x_3) = \vee^3(0, 2, 5, 6, 7)$

z multiplekserji različnih velikosti.

7.3.1 Število vhodnih spremenljivk je enako številu naslovnih vhodov multiplekserja

Na razpolago imamo konstanti 0 in 1 in multiplekser, ki ima enako število naslovnih vhodov, kot je število vhodnih spremenljivk preklopne funkcije. V tem primeru na naslovne vhode vežemo funkcijske vrednosti v enakem vrstnem redu kot ti nastopajo v pravilnostni tabeli, na naslovne vhode pa vhodne spremenljivke - prav tako v enakem vrstnem redu kot nastopajo v pravilnostni tabeli. Funkcijo torej ločimo preko vseh vhodnih spremenljivk, zato lahko vse funkcijske ostanke izrazimo s konstantami 0 in 1.

Zgled 22 *Realiziraj funkcijo* $\vee^3(0,2,5,6,7)$ *z MUX* $8/1$.

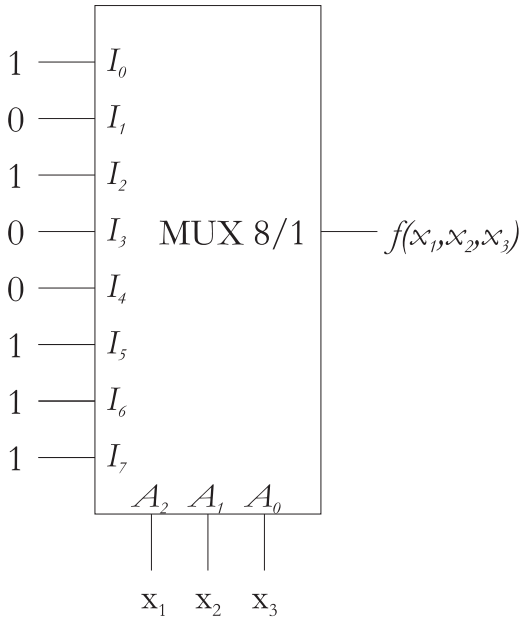
Rešitev 22 *V prvem koraku zapišemo pravilnostno tabelo funkcije.*

x_1	x_2	x_3	$f(x_1,x_2,x_3)$
0	*0*	*0*	*1*
0	*0*	*1*	*0*
0	*1*	*0*	*1*
0	*1*	*1*	*0*
1	*0*	*0*	*0*
1	*0*	*1*	*1*
1	*1*	*0*	*1*
1	*1*	*1*	*1*

V drugem koraku na podatkovne vhode vežemo funkcijske vrednosti v enakem vrstnem redu kot ti nastopajo v pravilnostni tabeli, na naslovne vhode pa vhodne spremenljivke - prav tako v enakem vrstnem redu kot nastopajo v pravilnostni tabeli (glej sliko 7.2).

7.3.2 Število vhodnih spremenljivk je za 1 večje od števila naslovnih vhodov multiplekserja

Na razpolago imamo konstanti 0 in 1 in multiplekser, ki ima število naslovnih vhodov za 1 manjše, kot je število vhodnih spremenljivk preklopne funkcije. Funkcijo ločimo po spremenljivkah glede na njihov vrstni red od leve proti desni. Iz tega sledi, da skrajno leve vhodne spremenljivke vežemo na naslovne vhode multiplekserja (najbolj levo spremenljivko na naslovni vhod z najvišjim indeksom). Na podatkovnih vhodih realiziramo funkcijski ostanek, ki ga lahko vedno izrazimo z najmanj pomembno vhodno spremenljivko, njeno negacijo, konstanto 0 in konstanto 1.



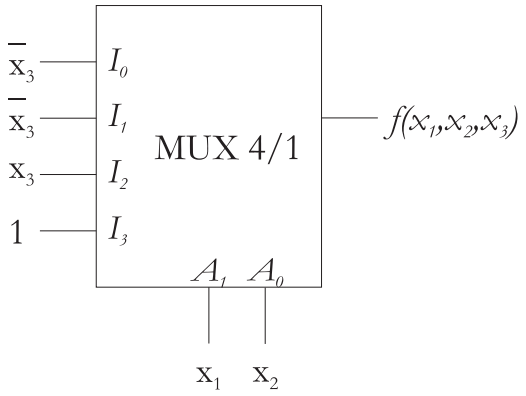
Slika 7.2 Realizacija 3-vhodne funkcije z multiplekserjem MUX 8/1.

Zgled 23 *Realiziraj funkcijo* $\vee^3(0,2,5,6,7)$ *z MUX* 4/1.

Rešitev 23 *V prvem koraku zapišemo pravilnostno tabelo funkcije.*

x_1	x_2	x_3	$f(x_1,x_2,x_3)$	
0	*0*	*0*	*1*	$I_0 = \overline{x}_3$
0	*0*	*1*	*0*	
0	*1*	*0*	*1*	$I_1 = \overline{x}_3$
0	*1*	*1*	*0*	
1	*0*	*0*	*0*	$I_2 = x_3$
1	*0*	*1*	*1*	
1	*1*	*0*	*1*	$I_3 = 1$
1	*1*	*1*	*1*	

V drugem koraku na naslovne vhode vežemo najpomembnejše spremenljivke (v našem primeru x_1 *in* x_2*). V našem primeru na vhod* A_1 *vežemo* x_1*, na vhod* A_2 *pa* x_2*. S tem funkcijo ločimo preko spremenljivk* x_1 *in* x_2*. S pomočjo spremenljivke, ki je ostala, uporabe negacij, konstante 0 in 1, realiziramo funkcijske ostanke ločenja, ki jih vežemo na podatkovne vhode (glej sliko 7.3).*



Slika 7.3 Realizacija 3-vhodne funkcije z multiplekserjem MUX 4/1.

7.3.3 Število vhodnih spremenljivk je za več kot 1 večje od števila naslovnih vhodov multiplekserja

Če je število vhodnih spremenljivk funkcije za več kot za 1 večje od števila naslovnih vhodov multiplekserja, moramo za realizacijo uporabiti večje število multiplekserjev, ki jih med seboj vežemo v drevesno (kaskadno) vezavo. Ponavadi funkcijo ločimo po najpomembnejših spremenljivkah, ni pa to pravilo. V tem primeru začnemo z vezavo najpomembnejših spremenljivk na naslovne vhode zadnjega multiplekserja v kaskadi.

Zgled 24 *Realiziraj funkcijo* $\vee^3(0,2,5,6,7)$ *z MUX* 2/1.

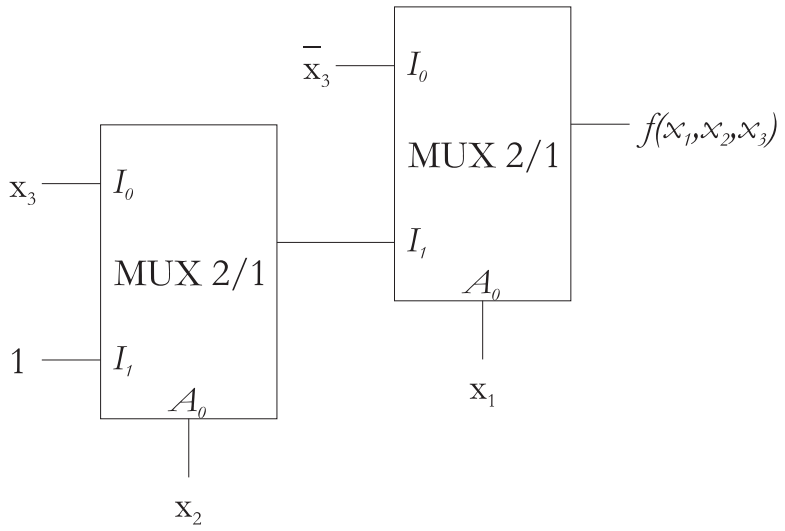
Rešitev 24 *V prvem koraku zapišemo pravilnostno tabelo in funkcijo*

- *najprej ločimo po* x_1,
- *zgornji del (funkcijski ostanek pri* $x_1 = 0$*) lahko izrazimo z* $\overline{x}_3$,
- *spodnji del (funkcijski ostanek pri* $x_1 = 1$*) ločimo še po* x_2 *(zgornji del, t.j. funkcijski ostanek pri* $x_2 = 0$*, lahko izrazimo z* x_3*, spodnji del, t.j. funkcijski ostanek pri* $x_2 = 1$*, pa s konstanto 1).*

Ostanke pri ločenju smo izračunali že v zgledu 21.

x_1	x_2	x_3	$f(x_1, x_2, x_3)$	
0	*0*	*0*	*1*	$\overline{x}_3$
0	*0*	*1*	*0*	
0	*1*	*0*	*1*	
0	*1*	*1*	*0*	
1	*0*	*0*	*0*	x_3
1	*0*	*1*	*1*	
1	*1*	*0*	*1*	1
1	*1*	*1*	*1*	

V drugem koraku narišemo shemo realizacije. Ker smo tabelo najprej ločili po x_1, x_1 *uporabimo kot naslovni vhod zadnjega multiplekserja v kaskadi. Funkcijski ostanek* f_0 *po* x_1 *lahko izrazimo z* x_3 – $\overline{x}_3$ *vežemo na podatkovni vhod* I_0 *multiplekserja. Spodnjega dela, ki predstavlja funkcijski ostanek* f_1 *po* x_1, *ne moremo izraziti na enostaven način, zato ga ločimo še po spremenljivki* x_2 *in za njegovo realizacijo uporabimo dodaten multiplekser MUX* 2/1. *Le-tega vežemo na vhod* I_1 *zadnjega multiplekserja v kaskadi. Na naslovni vhod dodatnega multiplekserja vežemo spremenljivko* x_2. *Funkcijski ostanek* f_0 *ločenja po* x_2 *lahko izrazimo z* x_3, *ostanek* f_1 *pa s konstanto* 1 - *razvidno iz tabele. Na podatkovni vhod* I_0 *dodatnega multiplekserja torej vežemo* x_3, *na vhod* I_1 *pa konstanto* 1. *Realizacijo prikazuje slika 7.4.*



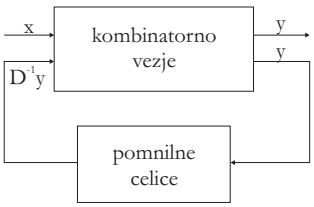
Slika 7.4 Realizacija 3-vhodne preklopne funkcije z dvema multiplekserjema MUX 2/1.

8 Priprava na 8. laboratorijske vaje

8.1 Sekvenčna vezja

Do zdaj smo obravnavali *kombinatorna vezja*, ki ne omogočajo pomnjenja. Pomnjenje je (običajno) realizirano z uporabo sinhronih pomnilnih celic (z uporabo n pomnilnih celic lahko realiziramo pomnjenje 2^n različnih vrednosti). Izhod sinhronih pomnilnih celic se spreminja ob določeni (pozitivni ali negativni) fronti *urinega signala* (angl. *clock*) – sinhronizacija z urinim signalom (glej sliko 8.1). *Sekvenčna vezja* so vezja, ki so sestavljena iz kombinatornega dela in pomnilnih celic (glej sliko 8.2).

Slika 8.1 Časovni potek vzorčnega primera urinega signala. Puščica navzgor označuje pozitivno (prehod iz 0 v 1), puščica navzdol pa negativno fronto (prehod iz 1 v 0). Slika prikazuje tri periode urinega signala.



Slika 8.2 Struktura sekvenčnih vezij.

8.2 Splošna pomnilna enačba

Vsako sekvenčno vezje lahko zapišemo s splošno pomnilno enačbo:

$$D^1 q_i = q_i g_{i,1} \vee \overline{q}_i g_{i,2},$$

pri čemer je

- q_i: ena izmed preklopnih spremenljivk, ki opisuje notranje stanje sekvenčnega vezja
- $D^k q_i$: vrednost spremenljivke q_i po k časovnih korakih
- $g_{i,1}$ in $g_{i,2}$: preklopni funkciji, odvisni od vhodnih spremenljivk $\mathbf{x} = \{x_1, x_2, ..., x_n\}$.

Novo notranje stanje sekvenčnega vezja je torej določeno s trenutnim notranjim stanjem vezja in s stanjem zunanjih vhodov. Notranje stanje se spremeni ob pozitivni ali negativni fronti urinega signala. Izbrana fronta je za celotno vezje vedno enaka, s čimer sekvenčne komponente v vezju med seboj sinhroniziramo.

8.3 Enostavne pomnilne celice

8.3.1 RS pomnilna celica (*Reset Set*)

RS pomnilna celica je najenostavnejša celica, katere delovanje lahko opišemo z enačbo

$$D^1 q = q\overline{r} \vee s,$$

pri čemer r in s predstavljata zunanja vhoda v celico. Kot nakažejo že imena vhodov, vhod r (*reset*) izhod pomnilne celice postavi na 0, vhod s (*set*) pa na 1. Če sta oba vhoda enaka 0, se izhod pomnilne celice ohranja (pomni). Za pravilno delovanje celice mora veljati pogoj $rs = 0$ (vhoda ne smeta biti istočasno 1). Delovanje RS pomnilne celice lahko ponazorimo s Tabelo 8.1.

r	s	$D^1 q$
0	0	q
0	1	1
1	0	0
1	1	x

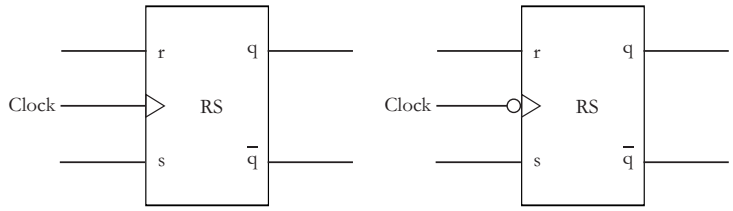
(a)

q	$D^1 q$	r	s
0	0	?	0
0	1	0	1
1	0	1	0
1	1	0	?

(b)

Tabela 8.1 Tabela (a) prikazuje izhod RS pomnilne celice v naslednjem časovnem koraku v odvisnosti od trenutnih vhodov, (b) pa stanja vhodov, s katerimi pridemo do želenega izhoda v naslednjem časovnem koraku (vzbujevalna tabela). Pri tem x prikazuje nedefiniran izhod oziroma nedovoljeno kombinacijo vhodov.

Logična simbola RS pomnilne celice sta prikazana na sliki 8.3.



(a) (b)

Slika 8.3 Logična simbola RS pomnilne celice, pri čemer je celica na sliki (a) sinhronizirana s pozitivno fronto urinega signala, celica na sliki (b) pa z negativno fronto.

8.3.2 JK pomnilna celica (*Jump Kill*)

JK pomnilna celica predstavlja razširitev RS pomnilne celice. Njeno delovanje lahko opišemo z enačbo

$$D^1q = q\overline{k} \vee \overline{q}j,$$

kjer sta j in k zunanja vhoda. Če je vsaj eden izmed vhodnih signalov j in k enak 0, je njeno delovanje enako delovanju RS pomnilne celice. Pri tem ima signal j (*jump*) enako vlogo kot signal s, signal k (*kill*) pa enako vlogo kot signal r RS pomnilne celice. Pri JK pomnilni celici je kombinacija vhodov $jk = 1$ dovoljena. Če postavimo oba vhoda na 1, se izhod pomnilne celice (q) ob urini fronti zamenja ($\overline{q}$). Delovanje JK pomnilne celice lahko ponazorimo s Tabelo 8.2.

k	j	D^1q
0	0	q
0	1	1
1	0	0
1	1	$\overline{q}$

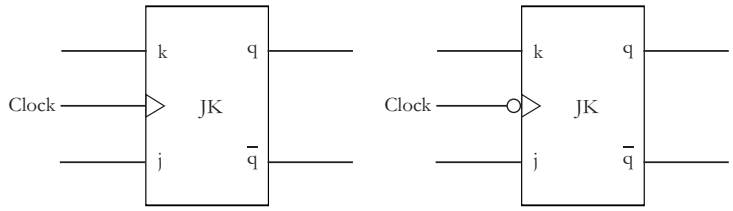
(a)

q	D^1q	k	j
0	0	?	0
0	1	?	1
1	0	1	?
1	1	0	?

(b)

Tabela 8.2 Tabela (a) prikazuje izhod JK pomnilne celice v naslednjem časovnem koraku v odvisnosti od trenutnih vhodov, (b) pa stanja vhodov, s katerimi pridemo do želenega izhoda v naslednjem časovnem koraku (vzbujevalna tabela).

Logična simbola JK pomnilne celice sta prikazana na sliki 8.4.



(a) (b)

Slika 8.4 Logična simbola JK pomnilne celice, pri čemer je celica na sliki (a) sinhronizirana s pozitivno fronto urinega signala, celica na sliki (b) pa z negativno fronto.

8.3.3 T pomnilna celica (*Trigger*)

T pomnilna celica ob fronti urinega signala spreminja izhod v primeru, da je zunanji vhod t postavljen na 1, sicer pa se vrednost izhoda ohranja. Njeno delovanje lahko opišemo z enačbo

$$D^1q = q\bar{t} \vee \bar{q}t,$$

kjer je t zunanji vhod. Delovanje T pomnilne celice lahko ponazorimo s Tabelo 8.3.

t	D^1q
0	q
1	$\bar{q}$

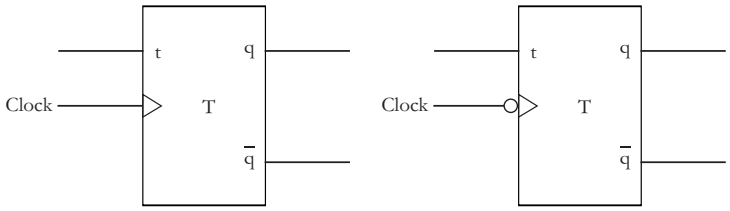
(a)

q	D^1q	t
0	0	0
0	1	1
1	0	1
1	1	0

(b)

Tabela 8.3 Tabela (a) prikazuje notranje izhod T pomnilne celice v naslednjem časovnem koraku v odvisnosti od vrednosti vhoda, (b) pa vrednost vhoda, s katero pridemo do želenega izhoda v naslednjem časovnem koraku (vzbujevalna tabela).

Logična simbola T pomnilne celice sta prikazana na sliki 8.5.



(a) (b)

Slika 8.5 Logična simbola T pomnilne celice, pri čemer je celica na sliki (a) sinhronizirana s pozitivno fronto urinega signala, celica na sliki (b) pa z negativno fronto.

8.3.4 D pomnilna celica (Delay)

D pomnilna celica ob fronti urinega signala postavi svoj izhod na vrednost vhodnega signala. Njeno delovanje si lahko torej interpretiramo kot zakasnitev vhodnega signala d. Opišemo ga lahko z enačbo

$$D^1 q = d.$$

Delovanje D pomnilne celice lahko ponazorimo s Tabelo 8.4.

d	D^1q
0	0
1	1

(a)

q	D^1q	d
0	0	0
0	1	1
1	0	0
1	1	1

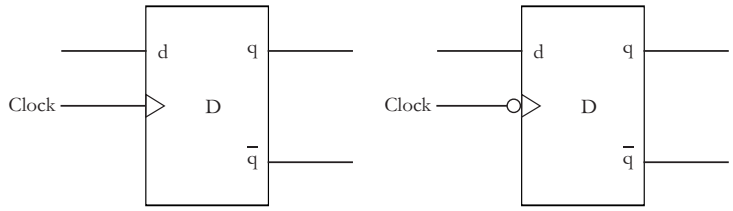
(b)

Tabela 8.4 Tabela (a) prikazuje izhod D pomnilne celice v naslednjem časovnem koraku v odvisnosti od vrednosti vhoda, (b) pa vrednost vhoda, s katero pridemo do želenega izhoda v naslednjem časovnem koraku (vzbujevalna tabela).

Logična simbola D pomnilne celice sta prikazana na sliki 8.6.

8.4 Realizacija sekvenčnih vezij s pomnilnimi celicami

Podano imamo sekvenčno vezje in tip pomnilnih celic, ki jih lahko uporabimo za njegovo realizacijo. Naš cilj je določitev kombinatornega dela vezja, ki bo vhodne spremenljivke prilagodil delovanju danih pomnilnih celic na tak način, da bo izhod



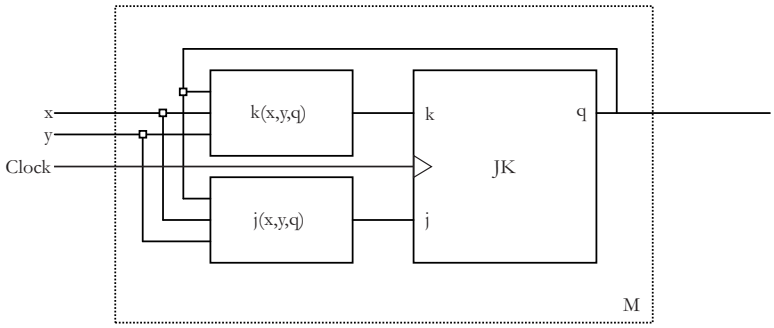
(a) (b)

Slika 8.6 Logična simbola D pomnilne celice, pri čemer je celica na sliki (a) sinhronizirana s pozitivno fronto urinega signala, celica na sliki (b) pa z negativno fronto.

pomnilnih celic odražal delovanje podanega sekvenčnega vezja. Postopek realizacije bomo demonstrirali na zgledu.

Zgled 25 *S pomočjo JK pomnilne celice realiziraj M celico, ki deluje po enačbi* $D^1q = q(x \equiv y) \vee \overline{q}\ \overline{y}$.

Rešitev 25 *Osnutek sheme realizacije prikazuje slika 8.7. Določiti je torej potrebno kombinatorni del vezja:* $k(x, y, q)$ *in* $j(x, y, q)$.



Slika 8.7 Shema realizacije celice M, pri čemer $k(x, y, q)$ določa kombinatorno vezje na vhodu k, $j(x, y, q)$ pa kombinatorno vezje na vhodu j JK pomnilne celice.

V prvem koraku v tabeli zapišemo funkcijo, ki določa delovanje sekvenčnega vezja. Pri tem so neodvisne spremenljivke x, y *in* q, *odvisna spremenljivka pa je* D^1q:

x	y	q	D^1q
0	*0*	*0*	*1*
0	*0*	*1*	*1*
0	*1*	*0*	*0*
0	*1*	*1*	*0*
1	*0*	*0*	*1*
1	*0*	*1*	*0*
1	*1*	*0*	*0*
1	*1*	*1*	*1*

Na podlagi prehodov iz q v D^1q in na podlagi vzbujevalne tabele JK pomnilne celice (glej Tabelo 8.2 (b)) lahko določimo vrednosti, ki morajo biti na vhodih k in j pri posamezni kombinaciji:

x	y	q	D^1q	k	j
0	*0*	*0*	*1*	*?*	*1*
0	*0*	*1*	*1*	*0*	*?*
0	*1*	*0*	*0*	*?*	*0*
0	*1*	*1*	*0*	*1*	*?*
1	*0*	*0*	*1*	*?*	*1*
1	*0*	*1*	*0*	*1*	*?*
1	*1*	*0*	*0*	*?*	*0*
1	*1*	*1*	*1*	*0*	*?*

Funkciji $k(x, y, q)$ in $j(x, y, q)$ izrazimo z vhodnimi spremenljivkami x, y in q. Pri tem si lahko pomagamo z Veitchevim diagramom:

y \ x	$\overline{x}$	$\overline{x}$	x	x
$\overline{y}$	?	0	1	?
y	?	1	0	?
	$\overline{q}$			

$$k(x, y, q) = \overline{x}\, y \vee x\, \overline{y}$$

y \ x	$\overline{x}$	$\overline{x}$	x	x
$\overline{y}$	0	?	?	0
y	1	?	?	1
	$\overline{q}$			

$$j(x, y, q) = \overline{y}$$

Funkciji realiziramo v shemi, ki jo prikazuje slika 8.7.

9 Priprava na 9. laboratorijske vaje

9.1 Moorov in Mealyjev končni avtomat

Končni avtomat (angl. *Finite State Machine*) je določen s peterico

$$A = \{X, S, Z, \delta, \lambda\},$$

kjer je

- X – vhodna abeceda: končna neprazna množica možnih vhodov v avtomat oziroma množica vhodnih črk,
- S – notranja abeceda: končna neprazna množica možnih stanj avtomata,
- Z – izhodna abeceda: končna neprazna množica možnih izhodov avtomata oziroma izhodnih črk,
- δ – funkcija prehajanja notranjih stanj: funkcija, ki v odvisnosti od trenutnega stanja in vhodne črke določa naslednje stanje avtomata,
- λ – izhodna funkcija: funkcija, ki določa izhodno črko avtomata v odvisnosti od trenutnega stanja avtomata (Moorov avtomat) oziroma v odvisnosti od trenutnega stanja in vhodne črke avtomata (Mealyjev avtomat).

Avtomate navadno podajamo tabelarično s *tabelo prehajanja stanj* ali grafično z *diagramom prehajanja stanj*. V tabeli prehajanja stanj podajamo naslednje stanje in izhodno črko avtomata v odvisnosti od trenutnega stanja (zgornja vrstica tabele) in vhodne črke avtomata (levi stolpec tabele) kot prikazuje spodnja tabela.

	izhodna črka (Moore)
	trenutno stanje
vhodna črka	naslednje stanje + izhodna črka (Mealy)

Diagram prehajanja stanj je podan z usmerjenim grafom, katerega vozlišča določajo stanja avtomata, povezave med vozlišči pa opisujejo prehode med stanji ob prisotnosti vhodnih črk. V primeru Moorovega avtomata stanjem pripišemo izhodno črko, ki pa je v primeru Mealyjevega avtomata vezana na prehode med stanji.

Zgled 26 *Nariši diagram prehajanja stanj za Moorov avtomat, ki je podan s sledečo tabelo prehajanja stanj*

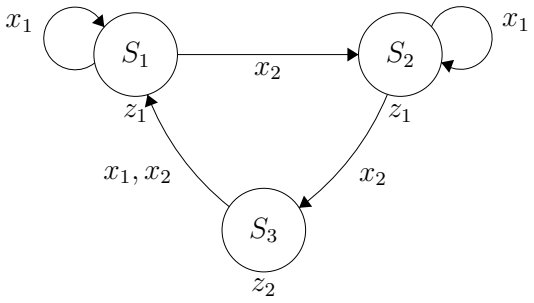
	z_1	z_1	z_2
	S_1	S_2	S_3
x_1	S_1	S_2	S_1
x_2	S_2	S_3	S_1

Kakšno je zaporedje izhodnih črk (izhodna beseda), ki jo avtomat vrne pri zaporedju vhodnih črk (vhodni besedi) $x_1x_1x_2x_2x_2x_2x_1$ *in začetnem stanju* S_1*?*

Rešitev 26
Najprej zapišimo vse tri abecede avtomata:

- $X = \{x_1, x_2\}$,
- $S = \{S_1, S_2, S_3\}$,
- $Z = \{z_1, z_2\}$.

Diagram prehajanja stanj ima torej tri vozlišča, iz vsakega vozlišča pa vodita največ dve povezavi (za vsako vhodno črko ena):



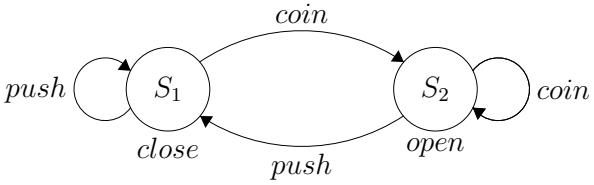
Simulirajmo še delovanje avtomata pri vhodni besedi $x_1x_1x_2x_2x_2x_2x_1$ *in začetnem stanju* S_1 *z uporabo spodnje tabele:*

vhodna črka		x_1	x_1	x_2	x_2	x_2	x_2	x_1
stanje	S_1	S_1	S_1	S_2	S_3	S_1	S_2	S_2
izhodna črka		z_1	z_1	z_1	z_2	z_1	z_1	z_1

Izhodna beseda je torej $z_1z_1z_1z_2z_1z_1z_1$.

Zgled 27 *Nariši diagram prehajanja stanj in zapiši tabelo prehajanja stanj Moorovega avtomata, ki nadzira odpiranje krožnih vrat. Mehanizem drži vrata zaprta, dokler avtomat ne zazna, da je bil vstavljen kovanec. V tem primeru avtomat mehanizmu javi, da lahko sprosti zaporo vrat. Ko avtomat zazna prehod skozi vrata, ta javi mehanizmu, naj zapre vrata.*

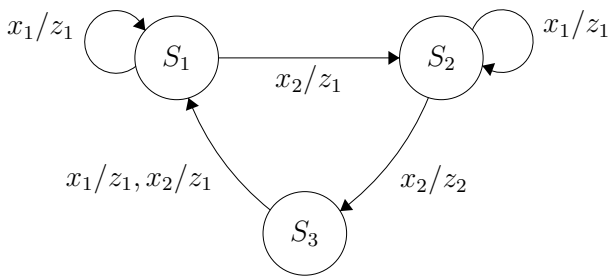
Rešitev 27 *Avtomat bo imel dve stanji, in sicer stanje, v katerem drži vrata zaprta (S_1) in ima tako izhodno črko close, in stanje, v katerem so vrata odprta (S_2) in ima tako izhodno črko open. Prehod iz S_1 v S_2 bomo izvedli, ko avtomat detektira, da je bil vstavljen kovanec (vhodna črka coin), prehod iz S_2 v S_1 pa, ko avtomat detektira prehod skozi vrata (vhodna črka push).*



Zapišimo še tabelo prehajanja stanj avtomata:

	close	*open*
	S_1	S_2
coin	S_2	S_2
push	S_1	S_1

Zgled 28 *Za Mealyjev avtomat podan z diagramom prehajanja stanj zapiši tabelo prehajanja stanj.*



Kakšno je zaporedje izhodnih črk (izhodna beseda), ki jo avtomat vrne pri zaporedju vhodnih črk (vhodni besedi) $x_1x_1x_2x_2x_2x_2x_1$ *in začetnem stanju* S_1*?*

Rešitev 28 *Abecede avtomata so enake kot pri Moorovem avtomatu iz zgleda 26. Zapišimo tabelo prehajanja stanj:*

	S_1	S_2	S_3
x_1	S_1/z_1	S_2/z_1	S_1/z_1
x_2	S_2/z_1	S_3/z_2	S_1/z_1

Simulirajmo še delovanje avtomata pri vhodni besedi $x_1x_1x_2x_2x_2x_2x_1$ *in začetnem stanju* S_1 *z uporabo spodnje tabele:*

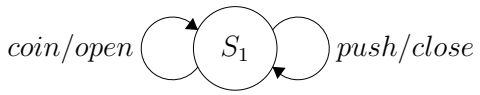
vhodna črka		x_1	x_1	x_2	x_2	x_2	x_2	x_1
stanje	S_1	S_1	S_1	S_2	S_3	S_1	S_2	S_2
izhodna črka		z_1	z_1	z_1	z_2	z_1	z_1	z_1

Izhodna beseda je torej $z_1z_1z_1z_2z_1z_1z_1$*.*

Avtomat pri enakih pogojih generira enako izhodno abecedo kot Moorov avtomat iz zgleda 26. Podrobnejša analiza bi pokazala, da sta si avtomata ekvivalentna.

Zgled 29 *Nariši še Mealyjev avtomat v skladu z navodili iz zgleda 27.*

Rešitev 29 *V primeru Mealyjevega avtomata lahko avtomat z enim samim stanjem generira obe izhodni črki (*open *in* close*). Diagram prehajanja stanj ima tako sledečo obliko:*



Zapišimo še tabelo prehajanja stanj avtomata:

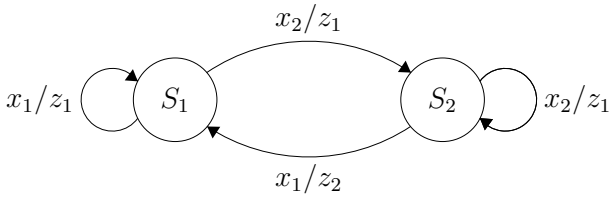
	S_1
coin	$S_1/open$
push	$S_1/close$

9.2 Pretvorbe med avtomati

Kot smo videli že v prejšnjih zgledih imata lahko dva avtomata različnega tipa (Moore in Mealy) popolnoma enako delovanje in sta si tako ekvivalentna (izjema je začetna izhodna črka, saj Mealyjev avtomat le-to zgenerira šele pri prvem prehodu, pri Moorovem avtomatu pa je izhodna črka vedno prisotna). V splošnem velja, da za vsak Moorov avtomat obstaja vsaj en njegov Melyjev ekvivalent in obratno.
Pretvorbo iz Melyjevega avtomata v Moorovega lahko izvedemo po sledečem postopku:

1. Mealyjev avtomat zapišemo tabelarično.
2. Zapišemo vse tri abecede Moorovega avtomata, pri čemer se vhodna in izhodna abecedi ohranjata, notranjo abeceda pa tvorimo s pari notranje stanje/izhodna črka, ki se pojavijo znotraj tabele Mealyjevega avtomata.
3. Zapišemo tabelo Moorovega avtomata, v kateri nastopajo vsa stanja, ki smo jih definirali v prejšnjem koraku. Izhodna črka za stanje je določena z izhodno črko, katere oznaka nastopa pri posameznem stanju. Prehodi med stanji so določeni s prehodi med ekvivalentimi stanji v Mealyjevem avtomatu,

Zgled 30 *Mealyjev avtomat, podan s spodnjim diagramom, pretvori v Moorov avtomat.*



Rešitev 30 *Podan Mealyjev avtomat zapišemo tabelarično:*

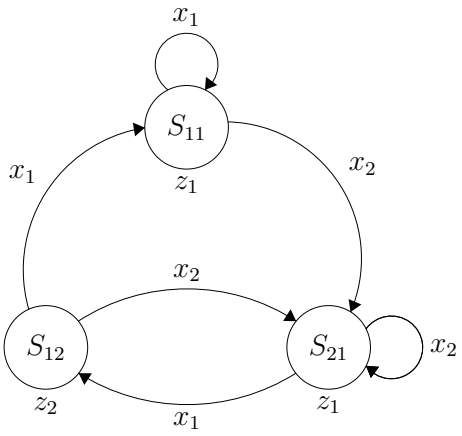
	S_1	S_2
x_1	S_1/z_1	S_1/z_2
x_2	S_2/z_1	S_2/z_1

Notranjo abecedo Moorovega avtomata tvorijo pari naslednje stanje/izhodna črka Mealyjevega avtomata, ki se pojavijo znotraj tabele prehajanja stanj. Notranja abeceda je torej: S_1/z_1, S_1/z_2, S_2/z_1 (opomba: stanje S_2/z_2 lahko izpustimo, ker se v tabeli ne pojavi – predpostavljamo, da to stanje ni začetno stanje avtomata). Zaradi lažjega zapisa stanja preimenujemo v: S_{11}, S_{12}, S_{21}. Vhodna in izhodna abeceda ostaneta enaki.

Na podlagi delovanja Melyjevega avtomata lahko tako zapišemo tabelo Moorovega avtomata:

	z_1	z_2	z_1
	S_{11}	S_{12}	S_{21}
x_1	S_{11}	S_{11}	S_{12}
x_2	S_{21}	S_{21}	S_{21}

In ga narišemo.



Po podobnem postopku lahko izvedemo pretvorbo Moorovega v Mealyjev avtomat:

1. Moorov avtomat zapišemo tabelarično.
2. Zapišemo vse tri abecede Mealyjevega avtomata, ki so kar enake abecedam Moorovega avtomata.
3. Zapišemo tabelo Mealyjevega avtomata, pri čemer je naslednje stanje enako kot pri Moorovem avtomatu, izhodna črka pa je določena z izhodno črko stanja Moorovega avtomata, v katerega bo avtomat z določeno kombinacijo stanje/vhodna črka prišel.

Zgled 31 *Moorov avtomat, ki predstavlja rešitev prejšnjega zgleda, pretvori v Mealyjev avtomat.*

Rešitev 31 *Pretvorimo dobljeni Moorov avtomat nazaj v Mealyjevega. Vse tri abecede se ohranjajo. Zapišimo tabelo prehajanja stanj:*

	S_{11}	S_{12}	S_{21}
x_1	S_{11}/z_1	S_{11}/z_1	S_{12}/z_2
x_2	S_{21}/z_1	S_{21}/z_1	S_{21}/z_1

Dobili smo avtomat s tremi notranji stanji, ki je enakovreden prvotnemu Mealyjevemu avtomatu z dvema stanjema. Stanji S_{11} *in* S_{12} *sta enaki tako po izhodnih črkah kot tudi po prehodih. Stanji lahko tako zakodiramo zgolj z enim notranjim stanjem – recimo mu stanje* S_1*. Stanje* S_{21} *preimenujmo, da bo zapis avtomata bolj pregleden – recimo mu stanje* S_2*. Nad stanji avtomata tako izvedemo sledečo preslikavo:* $S_{11} \to S_1$
$S_{12} \to S_1$
$S_{21} \to S_2$

Tako dobimo avtomat, ki je enak izhodiščnemu:

	S_1	S_2
x_1	S_1/z_1	S_1/z_2
x_2	S_2/z_1	S_2/z_1

10 Priprava na 10. laboratorijske vaje

10.1 Realizacija končnih avtomatov s pomnilnimi celicami (Moorov avtomat)

Z uporabo logičnih vrat in pomnilnih celic želimo realizirati Moorov avtomat. Postopek realizacije je sestavljen iz sledečih korakov:

1. *Zapis kodirnih tabel*: vhodna abeceda, notranja abeceda in izhodna abeceda so predstavljene z abstraktnim zapisom. Za realizacijo s preklopnimi funkcijami moramo le-te predstaviti s preklopnimi spremenljivkami. Pri tem upoštevamo dejstvo, da lahko z i vhodnimi spremenljivkami zakodiramo 2^i vhodnih črk, z j izhodnimi spremenljivkami 2^j izhodnih črk, s k pomnilnimi celicami pa 2^k notranjih stanj. S kodiranimi tabelami povežemo posamezne spremenljivke oziroma notranja stanja pomnilnih celic s posameznimi črkami oziroma stanji avtomata.

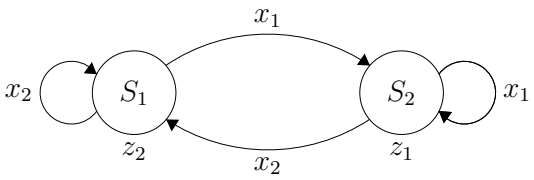
2. *Zapis pravilnostne tabele avtomata*: na podlagi diagrama prehajanja stanj oziroma tabele prehajanja stanj in kodirnih tabel, lahko zapišemo pravilnostno tabelo avtomata. Pri tem na levi strani tabele nastopajo spremenljivke, ki določajo vhodne črke in trenutna notranja stanja avtomata (neodvisne spremenljivke), na desni strani pa spremenljivke, ki določajo notranje stanje avtomata v naslednjem časovnem koraku in spremenljivke, ki določajo izhodne črke avtomata (odvisne spremenljivke).

3. *Določitev vhodov v pomnilne celice*: na podlagi prehodov med spremenljivkami, ki določajo trenutno stanje avtomata in stanje avtomata v naslednjem časovnem koraku ter vzbujevalnih tabel pomnilnih celic, ki jih imamo na razpolago, lahko določimo potrebne vhode v pomnilne celice v posamezni vrstici (tabelo dopolnimo na podoben način kot smo jo pri realizaciji sekvenčnih vezij s pomnilnimi celicami).

4. *Izpis in minimizacija izhodne funkcije in funkcije prehajanja stanj*: na podlagi pravilnostne tabele lahko s pomočjo Veitchevega diagrama izpišemo preklopne funkcije, ki določajo izhodne črke avtomata (izhodna funkcija) in preklopne

funkcije, ki nastopajo na vhodih pomnilnih celic in tako določajo prehode med stanji avtomata (funkcija prehajanja stanj)

Zgled 32 *Z uporabo T pomnilnih celic in poljubnih logičnih vrat realiziraj Moorov avtomat, ki je podan z diagramom prehajanja stanj.*



Rešitev 32 *Postopek je sledeč:*

1. *Zapišemo kodirne tabele, ki določijo kodiranje vhodne abecede, notranje abecede in izhodne abecede. Za zapis vseh črk vhodne abecede je dovolj ena vhodna spremenljivka (x); prav tako je za zapis vseh črk izhodne abecede dovolj ena izhodna spremenljivka (y). Ker imamo zgolj dve notranji stanji avtomata, je za njegovo realizacijo potrebna ena pomnilna celica T z notranjim stanjem q. Kodirne tabele so torej*

	x
x_1	0
x_2	1

	q
S_1	0
S_2	1

	y
z_1	0
z_2	1

2. *Na podlagi kodirnih tabel in podanega diagrama prehajanja stanj lahko zapišemo pravilnostno tabelo avtomata. Pri tem na levi strani tabele nastopajo spremenljivke, ki določajo trenutno notranje stanje avtomata (q) in vhodno črko (x), na desni pa spremenljivke, ki določajo notranje stanje avtomata v naslednjem časovnem koraku (D^1q) in izhodno črko (y):*

q	x	D^1q	y
0	*0*	*1*	*1*
0	*1*	*0*	*1*
1	*0*	*1*	*0*
1	*1*	*0*	*0*

3. *Na podlagi prehodov med spremenljivkami q in D^1q in vzbujevalne tabele za T pomnilno celico, lahko določimo vrednosti, ki morajo biti na vhodu t pri posamezni kombinaciji vhodnih spremenljivk:*

q	x	D^1q	y	t
0	*0*	*1*	*1*	*1*
0	*1*	*0*	*1*	*0*
1	*0*	*1*	*0*	*0*
1	*1*	*0*	*0*	*1*

4. *Na podlagi pravilnostne tabele lahko s pomočjo Veitchevega diagrama izpišemo funkcijo, ki nastopa na vhodu T pomnilne celice in izhodno funkcijo avtomata:*

x \ q	1	0
	0	1

x \ q	0	1
	0	1

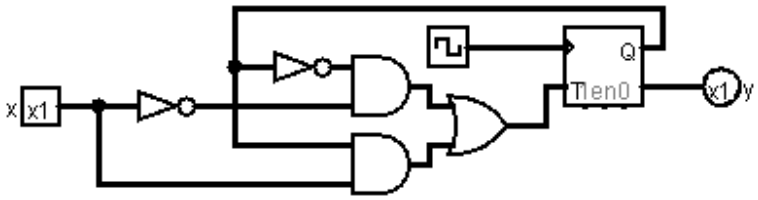
$$t = \overline{x}\ \overline{q} \vee xq = x \equiv q \qquad y = \overline{q}$$

Če funkcijo na vhodu T pomnilne celice vstavimo v enačbo T pomnilne celice ($D^1q = t\overline{q} \vee \overline{t}q$), dobimo funkcijo prehajanja stanj *avtomata:*

$$D^1q = (x \equiv q)\overline{q} \vee \overline{(x \equiv q)}q = (x \equiv q)\overline{q} \vee (x \nabla q)q$$

Po definiciji velja, da je izhodna črka Moorovega avtomata določena s trenutnim stanjem avtomata. Velja torej, da je izhodno črko pri Moorovem avtomatu vedno mogoče izraziti zgolj s spremenljivkami, ki določajo trenutno stanje avtomata (v našem primeru $y = \overline{q}$).

Realizacijo avtomata v Logisimu prikazuje slika 10.1.



Slika 10.1 Realizacija avtomata iz zgleda v Logisimu.

Zgled 33 *Z uporabo D pomnilnih celic realiziraj Moorov avtomat, podan s tabelo*

	z_1	z_2	z_3
	S_1	S_2	S_3
x_1	S_1	S_1	S_1
x_2	S_2	S_3	S_2

Rešitev 33 *Postopek je sledeč:*

1. *Za zapis dveh črk vhodne abecede je dovolj ena vhodna spremenljivka (x). Ker ima izhodna abeceda tri črke, za njen zapis potrebujemo dve spremenljivki (y_1 in y_2). Ker imamo tri notranja stanja avtomata, sta za njegovo realizacijo potrebni dve pomnilni celici D z notranjimi stanji q_1 in q_2. Kodirne tabele so torej*

	x
x_1	0
x_2	1

	q_1	q_2
S_1	0	0
S_2	0	1
S_3	1	0

	y_1	y_2
z_1	0	0
z_2	0	1
z_3	1	0

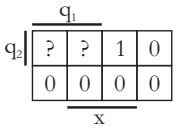
2. *Zapišemo pravilnostno tabelo avtomata (ko je $q_1 = 1$ in $q_2 = 1$, funkcijska vrednost ni določena):*

q_1	q_2	x	D^1g_1	D^1g_2	y_1	y_2
0	*0*	*0*	*0*	*0*	*0*	*0*
0	*0*	*1*	*0*	*1*	*0*	*0*
0	*1*	*0*	*0*	*0*	*0*	*1*
0	*1*	*1*	*1*	*0*	*0*	*1*
1	*0*	*0*	*0*	*0*	*1*	*0*
1	*0*	*1*	*0*	*1*	*1*	*0*
1	*1*	*0*	*?*	*?*	*?*	*?*
1	*1*	*1*	*?*	*?*	*?*	*?*

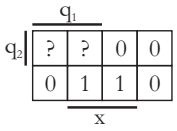
3. Na podlagi prehodov iz spremenljivke q_1 v D^1q_1 ter prehodov iz spremenljivke q_2 v D^1q_2, lahko določimo vrednosti, ki morajo biti na vhodu D pomnilnih celic (pomnilna celica z vhodom d_1 bo hranila notranje stanje q_1, celica z vhodom d_2 pa q_2):

q_1	q_2	x	D^1g_1	D^1g_2	y_1	y_2	d_1	d_2
0	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
0	*0*	*1*	*0*	*1*	*0*	*0*	*0*	*1*
0	*1*	*0*	*0*	*0*	*0*	*1*	*0*	*0*
0	*1*	*1*	*1*	*0*	*0*	*1*	*1*	*0*
1	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*
1	*0*	*1*	*0*	*1*	*1*	*0*	*0*	*1*
1	*1*	*0*	*?*	*?*	*?*	*?*	*?*	*?*
1	*1*	*1*	*?*	*?*	*?*	*?*	*?*	*?*

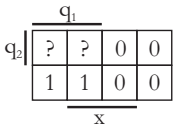
4. Določimo funkcije za realizacijo:



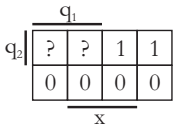
$d_1 = q_2x$



$d_2 = \overline{q}_2x$

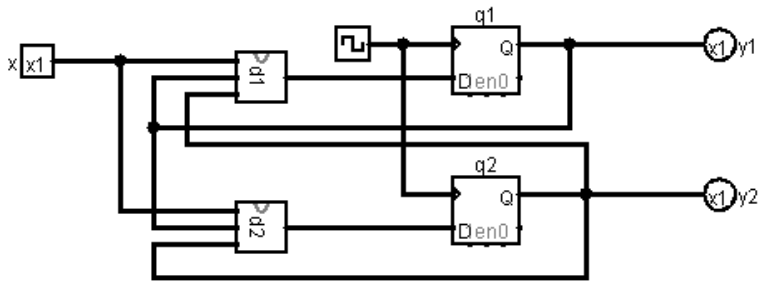


$y_1 = q_1$

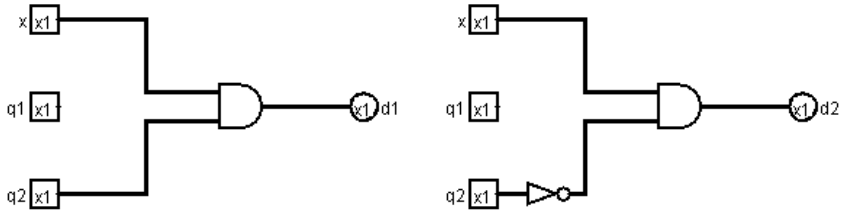


$y_2 = q_2$

Realizacijo avtomata v Logisimu prikazujejo slike 10.2, 10.3(a) in 10.3(b).



Slika 10.2 Realizacija avtomata iz zgleda v Logisimu. Zaradi splošnosti sheme sta funkciji, ki vstopata v pomnilni celici D, umeščeni v ločena modula (glej sliki 10.3(a) in 10.3(b)).



Slika 10.3 Realizacija preklopne funkcije, ki vstopa v D pomnilno celico z vhodom d_1 (a) in preklopne funkcije, ki vstopa v D pomnilno celico z vhodom d_2 (b).

11 Priprava na 11. laboratorijske vaje

11.1 Realizacija končnih avtomatov s pomnilnimi celicami (Mealyjev avtomat)

Obnašanje Moorovega avtomata je že v osnovi sinhrono. Stanje avtomata se v odvisnosti od vhodne črke spremeni le ob urini fronti. Ker je izhodna črka neposredno odvisna le od njegovega stanja, se ta sinhrono spremeni ob spremembi stanja avtomata.

Za razliko od Moorovega avtomata je pri Mealyjevem avtomatu izhodna črka neposredno odvisna od stanja avtomata in vhodnih spremenljivk. To pomeni, da se v primeru spremembe vhodne spremenljivke, izhodna črka lahko spremeni asinhrono (spremeni se v trenutku spremembe vhodne spremenljivke in ne le ob urini fronti oziroma spremembi stanja avtomata). Tako obnašanje ni dovoljeno in zahteva eksplicitno sinhronizacijo izhodne črke. To najenostavneje dosežemo tako, da izhodno vezje (realizacijo izhodne funkcije) vežemo na vhode D pomnilnih celic (pri realizaciji Mealyjevega avtomata z n izhodnimi črkami torej potrebujemo še $\lceil \log_2(n) \rceil$ dodatnih D pomnilnih celic). S tem dosežemo, da se izhodna črka spremeni ob urini fronti, torej sinhrono s spremembo stanja avtomata.
Realizacija Mealyjevega avtomata je zelo podobna realizaciji Moorovega avtomata. Ponazorili jo bomo z zgledom, ki sledi.

Zgled 34 *Realiziraj Mealyjev avtomat, ki je podan s tabelo prehajanja stanj. Za realizacijo notranjih stanj imaš na voljo T pomnilne celice, za generiranje izhodne črke pa D pomnilne celice.*

	S_1	S_2	S_3
x_1	S_2/z_1	S_2/z_2	S_1/z_1
x_2	S_1/z_2	S_3/z_1	S_3/z_2

Rešitev 34 *Postopek je sledeč:*

1. *Zapišemo kodirne tabele, ki določijo kodiranje vhodne abecede, notranje abecede in izhodne abecede. Za zapis vseh črk vhodne abecede je dovolj 1 vhodna*

spremenljivka (x); prav tako je za zapis vseh črk izhodne abecede dovolj 1 izhodna spremenljivka (y). Ker imamo tri notranja stanja avtomata, za njegovo realizacijo potrebujemo 2 pomnilni celici T z notranjimi stanji q_1 in q_2. Kodirne tabele so torej

	x
x_1	0
x_2	1

	q_1	q_2
S_1	0	0
S_2	0	1
S_3	1	0

	y
z_1	0
z_2	1

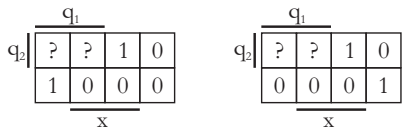
2. *Na podlagi kodirnih tabel in podanega diagrama prehajanja stanj lahko zapišemo pravilnostno tabelo avtomata. Pri tem na levi strani tabele nastopajo spremenljivke, ki določajo trenutno notranje stanje avtomata (q_1 in q_2) in vhodno črko (x), na desni pa spremenljivke, ki določajo notranje stanje (D^1q_1 in D^1q_2) in izhodno črko (D^1y) v naslednjem časovnem koraku. Vrednosti izhodne črke določamo na podlagi prehodov med stanji.*

q_1	q_2	x	D^1q_1	D^1q_2	D^1y
0	*0*	*0*	*0*	*1*	*0*
0	*0*	*1*	*0*	*0*	*1*
0	*1*	*0*	*0*	*1*	*1*
0	*1*	*1*	*1*	*0*	*0*
1	*0*	*0*	*0*	*0*	*0*
1	*0*	*1*	*1*	*0*	*1*
1	*1*	*0*	*?*	*?*	*?*
1	*1*	*1*	*?*	*?*	*?*

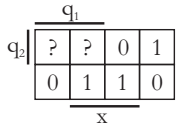
3. *Na podlagi prehodov med spremenljivkami q_1 in D^1q_1 ter q_2 in D^1q_2 in vzbujevalne tabele za T pomnilno celico, lahko določimo vrednosti, ki morajo biti na vhodih T pomnilnih celic (t_1 in t_2). Poleg tega določimo tudi vhode v D pomnilno celico, ki služi sinhronizaciji izhodne črke.*

q_1	q_2	x	D^1q_1	D^1q_2	D^1y	t_1	t_2	d
0	*0*	*0*	*0*	*1*	*0*	*0*	*1*	*0*
0	*0*	*1*	*0*	*0*	*1*	*0*	*0*	*1*
0	*1*	*0*	*0*	*1*	*1*	*0*	*0*	*1*
0	*1*	*1*	*1*	*0*	*0*	*1*	*1*	*0*
1	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*
1	*0*	*1*	*1*	*0*	*1*	*0*	*0*	*1*
1	*1*	*0*	*?*	*?*	*?*	*?*	*?*	*?*
1	*1*	*1*	*?*	*?*	*?*	*?*	*?*	*?*

4. Na podlagi pravilnostne tabele lahko s pomočjo Veitchevega diagrama izpišemo funkciji, ki nastopata na vhodih T pomnilnih celic in D pomnilne celice avtomata:



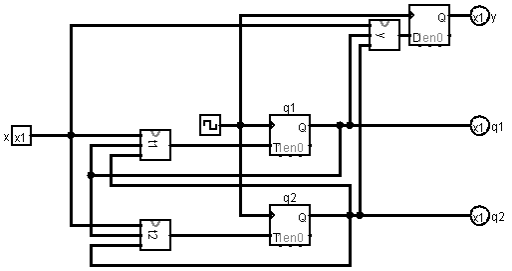
$$t_1 = q_1\overline{x} \vee q_2 x \quad t_2 = xq_2 \vee \overline{x}\ \overline{q}_1\overline{q}_2$$



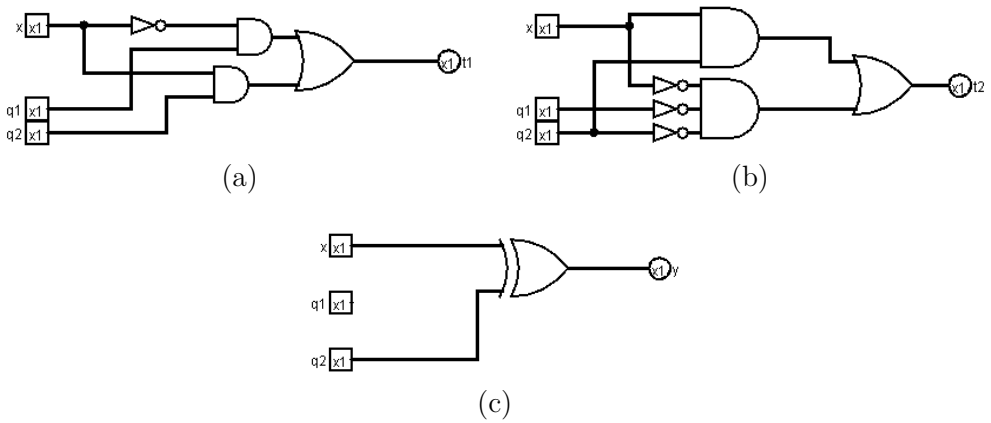
$$d = x\overline{q}_2 \vee \overline{x}q_2 = x \nabla q_2$$

Pri tem izhod iz D pomnilne celice realizira izhodno črko avtomata. Po definiciji velja, da je izhodna črka Mealyjevega avtomata določena s trenutnim stanjem in vhodno črko avtomata. Velja torej, da za realizacijo funkcije izhodne črke (funkcija, ki vstopa na vhodu D pomnilne celice) v splošnem potrebujemo tako spremenljivke, ki določajo trenutno stanje avtomata kot tudi spremenljivke, ki določajo vhodne črke avtomata.

Realizacijo avtomata v Logisimu prikazujejo slike 11.1, 11.2(a), 11.2(b) in 11.2(c).



Slika 11.1 Realizacija avtomata v Logisimu. Zaradi preglednosti sta funkciji, ki vstopata v pomnilni celici T in izhodna funkcija realizirane v ločenih modulih (glej slike 11.2(a), 11.2(b) in 11.2(c).



Slika 11.2 Realizacija preklopne funkcije, ki vstopa v T pomnilno celico z vhodom t_1 (a), preklopne funkcije, ki vstopa v T pomnilno celico z vhodom t_2 (b) in izhodne funkcije avtomata (c).

12 Dodatna vaja

12.1 Realizacija preprostega procesorja

Z gradniki, ki smo jih spoznali, se bomo lotili izdelave preprostega procesorja. Procesor je hipotetičen in izjemno poenostavljen, v osnovi pa je njegovo delovanje vseeno enako, kot delovanje kompleksnejših procesorjev.

12.2 Osnovno delovanje procesorja

Predpostavljamo, da naš procesor vsak ukaz izvrši v dveh urinih periodah. Procesor torej deluje v dveh stopnjah, in sicer:

- *prevzem ukaza* (angl. *Instruction Fetch, IF*),
- *izvedba ukaza* (angl. *Execute, EX*).

V nadaljevanju je opisana posamezna stopnja. Procesor lahko postavimo v začetno stanje (t.j. prevzem ukaza) z *Reset* vhodom.

12.2.1 Prevzem ukaza

V prvi urini periodi se iz pomnilnika, ki vsebuje sekvenco ukazov, t.i. *ukaznega pomnilnika*, prebere ukaz. Pomnilnik vsebuje več ukazov, pri čemer se vsak ukaz nahaja na svojem naslovu. Naslov iz katerega se bere trenutni ukaz je shranjen v registru, ki ga imenujemo *programski števec* (angl. *Program Counter*). Prebrani ukaz se shrani v *ukazni register* (angl. *Instruction Register*), kjer počaka na izvedbo, ki se izvrši v naslednji urini periodi - v stopnji *izvedba ukaza*.
Ko je ukaz zapisan v ukazni register, se lahko programski števec poveča. S tem dosežemo, da bo v naslednji stopnji, t.j. *prevzem ukaza*, prevzet naslednji ukaz iz ukaznega pomnilnika. Zaporedje ukazov v ukaznem pomnilniku tako sestavlja *program*, ki ga naš procesor izvaja.

12.2.2 Izvedba ukaza

Stopnja *izvedba ukaza* izvede ukaz, ki je zapisan v ukaznem registru. Pri našem procesorju je to lahko branje ali pisanje v pomnilnik ali aritmetično-logični operaciji: seštevanje in odštevanje.
Izvedba ukaza se izvrši tako, da kontrolna enota ukaz, ki je zapisan v ukaznem registru najprej *dekodira* (posamezen ukaz ima svojo kodo, ki je enolično določena s sekvenco bitov). V skladu z dekodiranim ukazom kontrolna enota določi vrednosti t.i. *kontrolnih signalov*, ki služijo kot vhodi v ostale komponente, ki opravijo dejansko izvedbo ukaza.
V našem primeru lahko kontrolno enoto predstavimo z Mealyjevim avtomatom z dvema stanjema in sedmimi različnimi prehodi med njima (glej Sliko 12.1). Avtomat v vsaki urini periodi preide iz stanja IF v stanje EX (razen v primeru aktivnosti vhoda *Reset*). Medtem, ko je iz stanja IF v stanje EX možen le en prehod (prevzem ukaza), lahko prehod iz stanja EX v stanje IF kontrolna enota izvede preko petih prehodov – izbira prehoda je odvisna od ukaza, ki ga je procesor prevzel pri prehodu kontrolne enote iz stanja IF v stanje EX.

12.3 Osnovna arhitektura procesorja

Procesor je v našem primeru sestavljen iz sledečih komponent:

- splošno namenski registri (A in B),
- programski števec (angl. *Program Counter, PC*),
- ukazni register (angl. *Instruction Register, IR*),
- kontrolna enota (angl. *Control Unit, CU*),
- aritmetično logična enota (angl. *Arithmetic Logic Unit, ALU*),
- ukazni pomnilnik (angl. *Instruction Memory, IM*),
- podatkovni pomnilnik (angl. *Data Memory, DM*),
- ostala kombinatorna logika.

Posamezna komponenta je podrobneje opisana v nadaljevanju.

12.3.1 Splošno namenski registri

Aritmetično logična enota izvaja aritmetično logične operacije (oziroma v našem primeru zgolj aritmetične operacije) nad podatki, ki so shranjeni v pomnilniku. Rezultati teh operacij se morajo slej ko prej zapisati nazaj v pomnilnik. Splošno

namenski registri služijo kot vmesnik med pomnilnikom in aritmetično logično enoto, saj aritmetično logična enota ne more neposredno dostopati do podatkov zapisanih v pomnilniku. Splošno namenski registri se torej uporabljajo za shranjevanje vmesnih rezultatov aritmetično logičnih operacij.
V našem primeru bomo imeli 2 2-bitna splošno namenska registra. Imenujemo ju register A in register B. Aritmetične operacije (seštevanje in odštevanje) lahko torej delamo nad dvema 2-bitnima podatkoma.

12.3.2 Programski števec

V stopnji *prevzem ukaza* se iz ukaznega pomnilnika prebere ukaz. Ukaz se zatem shrani v *ukazni register*. Pomnilniški naslov iz katerega se prebere ukaz, je zapisan v registru, ki mu rečemo *programski števec*.
V našem primeru je programski števec 2-bitni register. Naslavljamo lahko torej 4 pomnilniške lokacije - maksimalna dolžina programa v ukaznem pomnilniku so 4 ukazi.

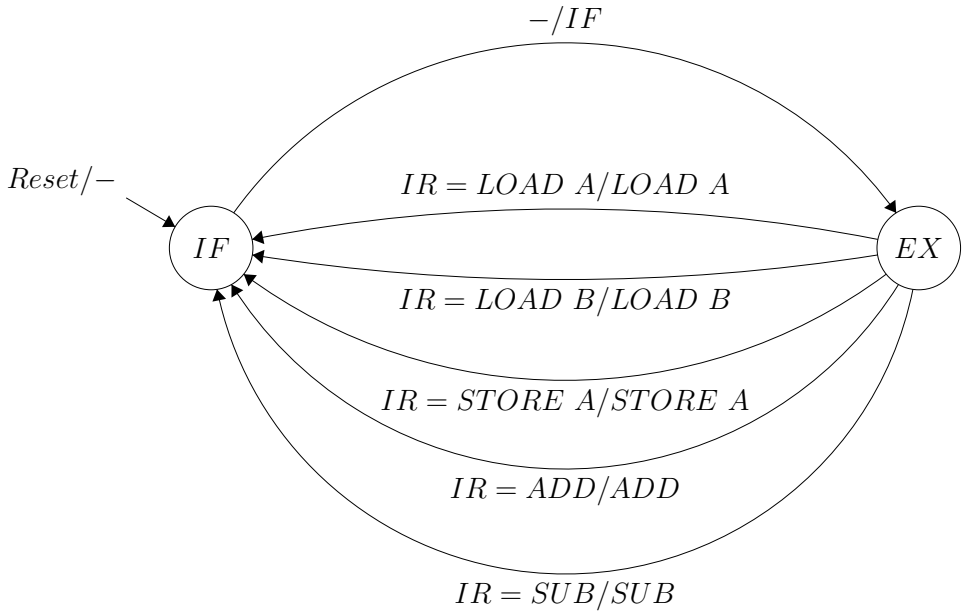
12.3.3 Ukazni register

V *ukazni register* se shrani ukaz, ki je bil prebran iz ukaznega pomnilnika v stopnjo *prevzem ukaza*. Na podlagi vsebine ukaznega registra v stopnji *izvedba ukaza* kontrolna enota določi vrednosti kontrolnih signalov. Kontrolni signali služijo kot vhodi v ostale komponente, ki opravijo dejansko izvedbo ukaza.
V našem primeru je ukazni register 4-bitni register. Naš procesor torej podpira 4-bitne ukaze.

12.3.4 Kontrolna enota

Kontrolna enota skrbi za izvedbo prevzema naslednjega ukaza in njegovo izvedbo. V skladu s trenutno stopnjo izvedbe ukaza in trenutnim ukazom določi vrednosti kontrolnih signalov, ki služijo kot vhodi v ostale komponente, ki opravijo dejanski prevzem in izvedbo ukaza. Kontrolno enoto lahko predstavimo z Mealyjevim avtomatom z dvema stanjema in sedmimi različnimi prehodi med njima (glej Sliko 12.1).
Kontrolna enota ima torej dve stanji, tj. stanje *IF*, ki je aktivno, ko je procesor v stopnji *prevzem ukaza* in stanje *EX*, ki je aktivno, ko je procesor v stopnji *izvedba ukaza*. Kot vhodne črke procesor sprejema *Reset* vhod in vsebino ukaznega registra (IR). Kontrolna enota določa vrednosti izhodnih signalov, ki so hkrati njene izhodne črke, na sledeč način:

- *IF*: izhodna črka je aktivna pri prehodu iz stanja *IF* v stanje *EX*. V tem primeru kontrolna enota sproži nalaganje naslednjega ukaza v ukazni register ($IR \leftarrow IM[PC]$) in povečanje vrednosti programskega števca ($PC \leftarrow PC+1$).



Slika 12.1 Mealyjev avtomat kontrolne enote procesorja.

- *LOAD A*: izhodna črka je aktivna pri prehodu iz stanja *EX* v stanje *IF* pri pogoju, da je v ukaznem registru ukaz *LOAD A*. V tem primeru kontrolna enota sproži nalaganje podatka iz podatkovnega pomnilnika (DM) v register *A*. Pri tem je pomnilniški naslov določen s spodnjima bitoma ukaznega pomnilnika ($A \leftarrow DM[IR[1..0]]$).

- *LOAD B*: izhodna črka je aktivna pri prehodu iz stanja *EX* v stanje *IF* pri pogoju, da je v ukaznem registru ukaz *LOAD B*. V tem primeru kontrolna enota sproži nalaganje podatka iz podatkovnega pomnilnika (DM) v register *B*. Pri tem je pomnilniški naslov določen s spodnjima bitoma ukaznega pomnilnika ($B \leftarrow DM[IR[1..0]]$).

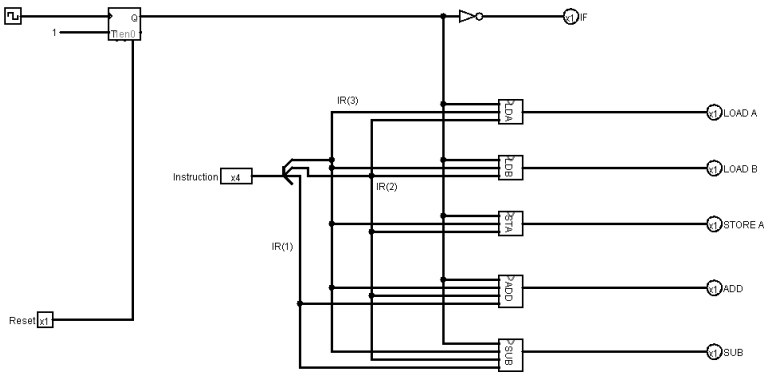
- *STORE A*: izhodna črka je aktivna pri prehodu iz stanja *EX* v stanje *IF* pri pogoju, da je v ukaznem registru ukaz *STORE A*. V tem primeru kontrolna enota sproži nalaganje vsebine registra *A* v podatkovni pomnilnik (DM), in sicer na naslov, ki je določen s spodnjima bitoma ukaznega pomnilnika ($DM[IR[1..0]] \leftarrow A$).

- *ADD*: izhodna črka je aktivna pri prehodu iz stanja *EX* v stanje *IF* pri pogoju, da je v ukaznem registru ukaz *ADD*. V tem primeru kontrolna enota

sproži seštevanje vsebine registra A in vsebine registra B, pri čemer se rezultat shrani v register A ($A \leftarrow A + B$).

- *SUB*: izhodna črka je aktivna pri prehodu iz stanja *EX* v stanje *IF* pri pogoju, da je v ukaznem registru ukaz *SUB*. V tem primeru kontrolna enota sproži odštevanje vsebine registra B od vsebine registra A, pri čemer se rezultat shrani v register A ($A \leftarrow A - B$).

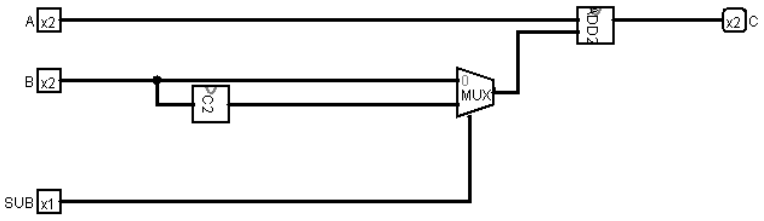
Realizacijo avtomata kontrolne enote v Logisimu prikazuje slika 12.2.



Slika 12.2 Realizacija kontrolne enote v Logisimu s T-pomnilno celico, pri čemer moduli *LDA*, *LDB*, *STA*, *ADD* in *SUB* predstavljajo kombinatorno logiko za določitev posameznega kontrolnega signala, *Instruction* pa ukaz, zapisan v ukaznem registru. Simbol, ki se v shemi nahaja pred komponento *Instruction*, predstavlja komponento, ki več enobitnih signalov združi v večbitni vektor.

12.3.5 Aritmetično logična enota

Aritmetično logična enota (angl. *Arithmetic Logic Unit, ALU*) izvaja aritmetično logične operacije nad splošno namenskimi registri kot so seštevanje, množenje, deljenje, logični pomik itd. V našem primeru ALU podpira samo dve artimetični operaciji, in sicer seštevanje in odštevanje vsebine registrov A in B. Operacija, ki se bo izvedla, je določena s kontrolnim signalom, katerega vrednost na podlagi dekodiranega signala določi kontrolna enota. Realizacija ALU v Logisimu je prikazana na sliki 12.3, pri čemer sta vhoda A in B izhoda iz splošno namenskih registrov, vhod *SUB* pa izhod iz kontrolne enote, ki ima vrednost 1, če je trenutni ukaz odštevanje. Izhod iz aritmetično logične enote določa rezultat izvedene operacije.



Slika 12.3 Realizacija aritmetično logične enote v Logisimu, pri čemer $ADD2$ predstavlja modul za izračun vsote 2-bitnih vhodov, $C2$ pa modul za izračun dvojiškega komplementa 2-bitnega vhoda (glej razdelek 12.5.4). Naslovni vhod v multiplekser (SUB) določi ali se bo izračunala vsota števil A in B ali vsota števila A in dvojiškega komplementa števila B.

12.3.6 Ukazni pomnilnik

V ukaznem pomnilniku so shranjeni ukazi za izvedbo. Na podlagi podanega naslova se iz ukaznega pomnilnika prebere ukaz za izvedbo. Na sekvenco ukazov v ukaznem pomnilniku lahko tako gledamo kot na program, ki ga bo naš procesor izvajal.
V našem primeru je ukazni pomnilnik realiziran z ROM pomnilnikom, v katerega zapišemo program pred zagonom procesorja. Na voljo imamo 4 lokacije (program je lahko dolg 4 ukaze). Pomnilnik torej naslavljamo z 2-bitnim naslovom. Ukazi v pomnilniku so dolgi 4 bite.

12.3.7 Podatkovni pomnilnik

Podatkovni pomnilnik vsebuje podatke nad katerimi procesor izvaja aritmetično logične operacije. Branje iz podatkovnega pomnilnika poteka preko registrov A in B. V pomnilnik lahko zapisujemo zgolj preko registra A. Prav tako kot pri ukaznem pomnilniku, tudi pri podatkovnem pomnilniku vedno podajamo naslov na katerega zapisujemo podatek oziroma s katerega beremo. V podatkovni pomnilnik lahko shranimo 4 podatke velikosti 2 bita. Naslavljamo ga torej z 2-bitnim naslovom.

12.3.8 Ostala kombinatorna logika

To so multiplekserji in ostala kombinatorna logična vrata.

12.4 Nabor ukazov

Procesor podpira izvedbo 5 različnih ukazov (glej Tabelo 12.1). Posamezen ukaz je določen s 4-bitno kodo, ki je shranjena v ukaznem pomnilniku oziroma ukaznem

registru. Pri nekaterih ukazih se del te kode uporabi kot naslov za podatkovni pomnilnik.

$IR(3)$	$IR(2)$	$IR(1)$	$IR(0)$	ukaz	pomen
0	0	x	y	*Load A* $DM[(x,y)]$	$A \leftarrow DM[(x,y)]$
0	1	x	y	*Load B* $DM[(x,y)]$	$B \leftarrow DM[(x,y)]$
1	0	x	y	*Store A* $DM[(x,y)]$	$DM[(x,y)] \leftarrow A$
1	1	0	?	*Add*	$A \leftarrow A + B$
1	1	1	?	*Sub*	$A \leftarrow A - B$

Tabela 12.1 Seznam in pomen ukazov, ki jih podpira naš procesor. Pri tem $DM[(x,y)]$ predstavlja lokacijo v podatkovnem pomnilniku na 2-bitnem naslovu (x,y).

V nadaljevanju so ukazi podrobneje opisani.

12.4.1 Load A

Ukaz *Load A* v register A naloži vrednost iz pomnilniškega naslova, ki je določen z bitoma $IR(1)$ in $IR(0)$ ukaznega registra.

12.4.2 Load B

Ukaz *Load B* v register B naloži vrednost iz pomnilniškega naslova, ki je določen z bitoma $IR(1)$ in $IR(0)$ ukaznega registra.

12.4.3 Store A

Ukaz *Store A* naloži vsebino registra A na pomnilniški naslov, ki je določen z bitoma $IR(1)$ in $IR(0)$ ukaznega registra.

12.4.4 Add

Ukaz *Add* sešteje vsebino registrov A in B in jo shrani v register A.

12.4.5 Sub

Ukaz *Sub* odšteje vsebino registra B od vsebine registra A in jo shrani v register A.

12.5 Razlaga uporabljenih gradnikov

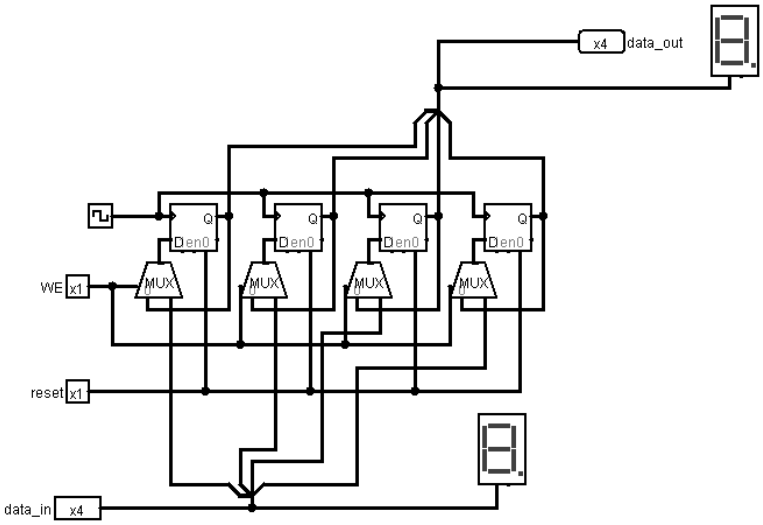
12.5.1 Register

Register je sinhroni pomnilni element, kar pomeni da omogoča pomnjenje (v n-bitni register lahko shranimo n-bitni podatek), njegova vsebina pa se spreminja sinhrono

– nov podatek lahko v register zapisujemo samo ob določeni fronti ure. Registri imajo v našem primeru sledeče vhode

- *Reset*: ob aktivnosti vhoda se vsebina registra postavi na vrednost 0.
- *Data_in*: podatkovni vhod v register.
- *WE* (*Write Enable*): če je vhod aktiven se podatek na vhodu *Data_in* ob urini fronti zapiše v register.

Vsebina registra je na izhodu *Data_out*. Realizacija 4-bitnega registra v Logisimu je prikazana na sliki 12.4.



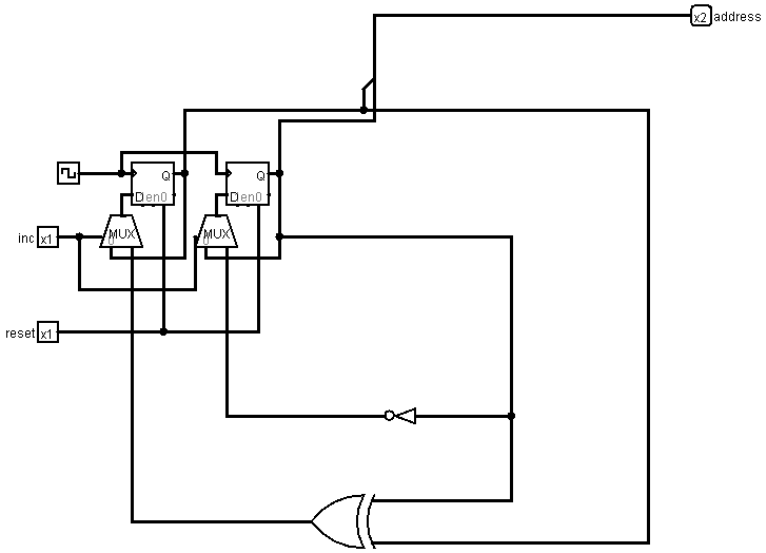
Slika 12.4 Realizacija 4-bitnega registra v Logisimu z D-pomnilnimi celicami. Če so naslovni vhodi v multiplekserje aktivni (z drugimi besedami, če je $WE = 1$), se stanje registra v naslednji urini periodi zamenja z vhodnim signalom *data_in*. V nasprotnem primeru se stanje registra ohranja.

12.5.2 Programski števec

Programski števec je v našem primeru posebna oblika registra s sledečima vhodoma:

- *Reset*: ob aktivnosti vhoda se programski števec postavi na 0.
- *Inc* (Increment): če je vhod aktiven se vsebina programskega števca ob urini fronti poveča za 1 na način 0,1,2,3,0,1...

Vsebino registra lahko beremo preko izhoda *Address*, ki služi kot naslov za branje ukaza iz ukaznega pomnilnika. Realizacija programskega števca v Logisimu je prikazana na sliki 12.5.

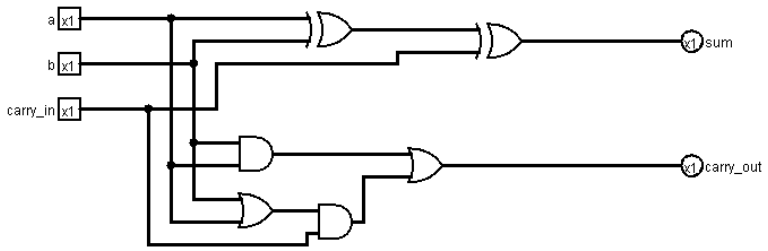


Slika 12.5 Realizacija 2-bitnega programskega števca v Logisimu. Del kombinatorne logike (XOR vrata in negator) predstavlja realizacijo povečanja vsebine programskega števca za 1. Če sta naslovna vhoda v multiplekserja (*inc*) aktivna, se stanje števca v naslednji urini periodi poveča. Če nista aktivna, se stanje ohranja.

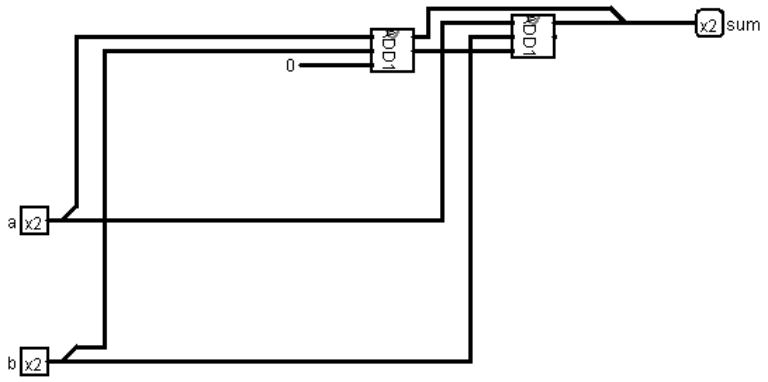
12.5.3 Seštevalnik

Seštevalnik omogoča seštevanje dveh števil in je del aritmetično logične enote. V našem primeru imamo seštevalnik, ki sešteje dve 2-bitni števili. 2-bitni seštevalnik realiziramo z vezavo dveh 1-bitnih polnih seštevalnikov. Realizacija 1-bitnega polnega seštevalnika v Logisimu je prikazana na sliki 12.6. Rezultat seštevanja dveh 1-bitnih števil a in b in prenosa *carry_in* gre na izhod *sum*. Izhod *carry_out* predstavlja prenos pri seštevanju.

2-bitni seštevalnik lahko realiziramo z vezavo dveh 1-bitnih polnih seštevalnikov kot prikazuje slika 12.7. Prenos pri seštevanju prvega seštevalnik (*carry_out*) uporabimo kot *carry_in* vhod v drugi seštevalnik.



Slika 12.6 Realizacija 1-bitnega polnega seštevalnika v Logisimu.



Slika 12.7 Realizacija 2-bitnega seštevalnika v Logisimu. Modula $ADD1$ predstavljata 1-bitna polna seštevalnika.

12.5.4 Odštevalnik

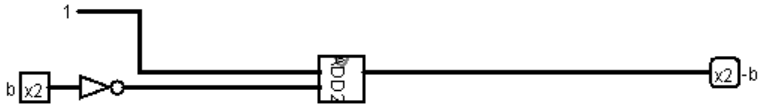
Ker velja, da je odštevanje enako prištevanju negativnega števila $(a - b = a + (-b))$, lahko pri realizaciji odštevalnika uporabimo seštevalnik, pri čemer seštevamo števili a in $-b$. V ta namen, moramo realizirati vezje, ki izračuna negativno vrednost števila b. Negativno vrednost dobimo s t.i. *dvojiškim komplementom*, ki ga lahko izračunamo z uporabo sledeče enačbe

$$-b = \overline{b} + 1$$

Vezje za izračun dvojiškega komplementa prikazuje slika 12.8.

12.5.5 Read Only Memory (ROM)

ROM pomnilnik je pomnilnik iz katerega lahko programsko zgolj beremo. V našem primeru vanj pred začetkom izvajanja ročno zapišemo program.



Slika 12.8 Realizacija vezja za izračun dvojiškega komplementa v Logisimu. Modul $ADD2$ predstavlja 2-bitni seštevalnik. Dvojiški komplement izračunamo tako, da negirani vrednosti vhodnega signala b prištejemo 1.

Vsebino ROM pomnilnika beremo tako, da na vhod A (*Address*) pripeljemo naslovni signal. Na izhodu D (*Data*) dobimo podatek, ki je zapisan na naslovu A. Z n-bitnim naslovom lahko naslavljamo 2^n lokacij.
V Logisimu mora biti pomnilnik za uporabo omogočen. Omogočimo ga tako, da na *sel* (*Select*) vhod pripeljemo konstanto 1.

12.5.6 Random Access Memory (RAM)

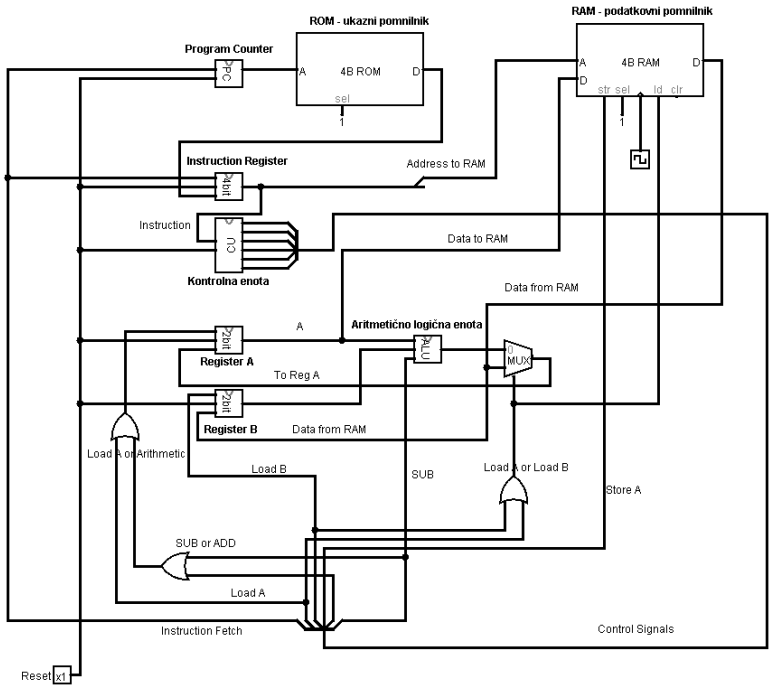
RAM pomnilnik je pomnilnik v katerega lahko tako pišemo kot tudi beremo. Pri pisanju v pomnilnik podamo podatek, ki ga želimo zapisati in naslov, na katerega želimo pisati. Pri branju iz pomnilnika podamo naslov, iz katerega beremo.
Pri uporabi RAM pomnilnika v Logisimu moramo upoštevati sledeče vhode:

- A (*Address*): naslov iz katerega beremo oziroma na katerega pišemo. Z n-bitnim naslovom lahko naslavljamo 2^n lokacij.
- D (*Data in*): podatek, ki ga zapisujemo v pomnilnik.
- *str* (*Store*): podatek na vhodu D se shrani v pomnilnik ob fronti ure, če je aktiven vhod *str*.
- *sel* (*Select*): pomnilnik omogočimo tako, da na *sel* vhod pripeljemo konstanto 1.
- *ld* (*Load*): podatek na naslovu A gre na izhod pomnilnika D (*Data out*) ob fronti ure samo, če je aktiven vhod *ld*.

Iz pomnilnika beremo preko izhoda D (*Data out*), tako da aktiviramo vhodni signal *ld*. Podatek se pojavi na izhodu ob fronti ure.

12.6 Realizacija opisanega procesorja

Z ustrezno vezavo opisanih komponent lahko v Logisimu realiziramo preprost procesor.
Realizacijo opisanega procesorja v Logisimu prikazuje slika 12.9.



Slika 12.9 Realizacija preprostega procesorja v Logisimu.

12.7 Zgled delovanja procesorja

Zgled 35 *V ukazni pomnilnik procesorja v Logisimu bomo zapisali program, ki sešteje števili, ki sta zapisani na pomnilniških naslovih 0 in 1 podatkovnega pomnilnika in rezultat zapiše na pomnilniški naslov 2.*
Najprej bomo program zapisali s t.i. mnemoniki:

```
Load A 0
Load B 1
Add
Store A 2
```

S pomočjo tabele 12.1 lahko ukaze zapišemo v dvojiški obliki:

```
0000
0101
1100
1010
```

Ukaze vnesemo v ukazni pomnilnik (ROM) v šestnajstiški obliki:

```
0
5
C
A
```

A Slovensko-angleški slovar osnovnih pojmov

Osnove

- preklopna funkcija: logic function
- pravilnostna tabela: truth table
- funkcijska vrednost: value of the function
- vhodni vektor: input vector
- logična shema: logic scheme
- logična vrata: logic gates
- spremenljivka: variable
- vhod: input
- izhod: output
- in: and
- ali: or
- množica: set
- seštevanje: addition

Standardne oblike zapisa

- PDNO (popolna disjunktivna normalna oblika): full disjunctive normal form (FDNF)
- DNO (disjunktivna normalna oblika): disjunctive normal form (DNF), also sum of products (SOP)
- PKNO (popolna konjunktivna normalna oblika): full conjunctive normal form (FCNF)

- KNO (konjunktivna normalna oblika): conjunctive normal form (DNF), also product of sums (POS)
- skrajšana oblika zapisa: compact notation
- eksplicitna oblika zapisa: explicit notation
- Veitchev diagram: Veitch diagram, its variation is known as Karnaugh map (K-map)

Funkcijska polnost

- funkcijsko poln sistem/nabor: functional complete set
- funkcijska polnost: functional completeness
- prevedba na znan funkcijsko poln sistem: proving the functional completeness of a set of functions by showing they can express all functions in already known functional complete set
- osnovni zaprti razredi: important closed classes of functions (see also Post's Functional Completeness Theorem)
- ohranjanje ničle (T_0): functions that preserve zero
- ohranjanje enice (T_1): functions that preserve one
- linearna funkcija: linear function
- monotona funkcija: monotone function
- sebidualna funkcija: self-dual function
- sosednja vektorja: adjacent vectors

Minimizacija

- MDNO (minimalna disjunktivna normalna oblika): minimal disjunctive normal form (MDNF), also minimum sum-of-products expression
- MKNO (minimalna konjunktivna normalna oblika): minimal conjunctive normal form (MDNF), also minimum product-of-sums expression
- MNO (minimalna normalna oblika): minimal normal form (MNF)
- sosednja minterma: adjacent minterms

- sosednji polji: adjacent cells
- vsebovalnik: implicant
- glavni vsebovalnik: prime implicant
- potrebni glavni vsebovalnik: essential prime implicant
- pokritje: coverage, also loop on the diagram
- nepopolna preklopna funkcija: incompletely specified function (includes don't care values/symbols, i.e. ?)
- Quineova metoda: Quine (also Quine-McCluskey) method

Simterične funkcije

- simetrična funkcija: symmetric function
- popolnoma simetrična funkcija: totally symmetric function

Multiplekserji

- multiplekser: multiplexer
- Shannonov teorem: Shannon expansion or Shannon decomposition (also Boole's expansion theorem)
- funkcijski ostanek: cofactor
- funkcijski ostanek funkcije po x: cofactor of the function with respect to x
- ločenje: decomposition
- podatkovni vhodi: data inputs
- naslovni vhodi: select inputs or address inputs

Sekvenčna vezja

- pomnilnik: memory
- sinhrona sekvenčna logika: synchronous sequential logic
- urin signal: clock signal

- pomnilna celica: flip-flop
- stanje: state
- karakteristična tabela: characteristic table
- vzbujevalna tabela: excitation table
- trava: glitches
- zdrs ure: clock skew

Avtomati

- (končni deterministični) avtomat: finite state machine (also finite state automaton)
- stanje: state
- črka: character
- beseda: sequence of characters
- vhodna abeceda: input alphabet (finite non-empty set of symbols)
- notranja abeceda: internal alphabet (finite, non-empty set of states)
- izhodna abeceda: output alphabet (finite, non-empty set of symbols)
- funkcija prehajanja (notranjih) stanj: state-transition function
- izhodna funkcija: output function
- Moorov avtomat: Moore machine
- Mealyjev avtomat: Mealy machine
- diagram prehajanja stanj: state diagram
- tabela prehajanja stanj: state transition table
- kodirna tabela: coding table

Literatura

[1] Iztok Lebar Bajec. *Preklopne strukture in sistemi: zbirka rešenih primerov in nalog z rešitvami.* Založba FE in FRI, Ljubljana, 2002.

[2] Ronald J. Tocci, Neal Widmer, and Greg Moss. *Digital Systems: Principles and Applications (11th Edition).* Pearson, 2010.

[3] Mira Trebar. *Preklopna vezja in strukture: priročnik za vaje (2. izdaja).* Založba FE in FRI, Ljubljana, 1992.

[4] Jernej Virant. *Logične osnove odločanja in pomnjenja v računalniških sistemih.* Založba FE in FRI, Ljubljana, 1996.