5. PREKLOPNE STRUKTURE ALI PREKLOPNI MNOGOPOLI

5.1 Matrično opisovanje preklopnih vezij in struktur

5.1.1 Osnovna simbolika

Vektor:

$\mathbf{a}_n$ – vodoravni vektor

$\mathbf{a}^m$ – navpični vektor

Matrika:

$\mathbf{A}_{1:n}^{1:m}$ - matrika reda m x n

$\mathbf{A}_{1:m}^{1:n}$ - matrika reda n x m

Opredelimo najprej tri osnovne nize matrik:

$$\mathbf{C}_{1:(m+n)}^{1:t} = \mathbf{A}_{1:m}^{1.t}\ \mathbf{B}_{1.n}^{1:t}$$ - vodoravni niz matrik **A** in **B**

$$\mathbf{C}_{1:t}^{1:(m+n)} = \begin{matrix}\mathbf{A}_{1:n}^{1:t} \\ \mathbf{B}_{1:t}^{1:n}\end{matrix}$$ - navpični niz matrik **A** in **B**

$$\mathbf{C}_{1:m}^{1:n} = \left[\mathbf{A}_{1:n}^{1:m}\right]^T;\ c_j^i = a_i^j$$ - transpozicija matrike **C**

Spremenljivke **A, B, C** so matrike z razsežnostmi: t x n ter m x t.

5.1.2 Operacije nad vektorji in matrikami

Redukcija vektorja k skalarju:

$$c = \circ \mid \mathbf{a}_n; \qquad c = \circ \mid \mathbf{a}^m,$$

kjer je:

$\circ$ - splošni znak za operator,

$c = a_1 \circ a_2 \circ a_3 \circ a_4 \circ \ldots\ldots \circ a_n$ – vodoravna redukcija vektorja k skalarju,

$c = a^{1} \circ a^{2} \circ a^{3} \circ a^{4} \circ \ldots\ldots \circ a^{m}$ – navpična redukcija vektorja k skalarju.

Med matrikami definirajmo splošno dvojno operacijo:

$$\mathbf{C}_{1:m}^{1:n} = \mathbf{A}_{1:t}^{1:n} *\circ \mathbf{B}_{1:m}^{1.t}$$

$$c_j^i = * \mid \left(a_{1:t}^i \circ b_i^{1:t} \right)$$

Elemente vrstice v matriki **A** povežemo z operacijo »$\circ$« z elementi stolpca matrike **B**.

Nato pa z vektorsko redukcijo preidemo preko operacije » *« na matriko **C**.

Negacija:

$$\mathbf{C}_n^m = \overline{\mathbf{A}}_n^m; \qquad c_j^i = \overline{a}_j^i$$

Konjunkcija:

$$C_n^m = \mathbf{A}_n^m \bullet \mathbf{B}_n^m; \qquad c_j^i = a_j^i \bullet b_j^i$$

PREKLOPNA VEZJA

Disjunkcija:

$$\mathbf{C}_n^m = \mathbf{A}_n^m + \mathbf{B}_n^m; \quad c_j^i = a_j^i + b_j^i$$

Izključna ALI operacija:

$$\mathbf{C}_n^m = \mathbf{A}_n^m \oplus \mathbf{B}_n^m; \quad c_j^i = a_j^i \oplus b_j^i$$

5.2 Popolna disjunktivna normalna oblika vektorske preklopne funkcije

Doslej obravnavane preklopne funkcije so imele obliko:

$$f(x_1\ x_2,\ldots\ldots,x_n) = \sum_{i=0}^{2^n-1} f_i\, m_i\ ;$$

$\mathbf{x} = [x_1, x_2, \ldots\ldots, x_n]$,

$\mathbf{m} = \left[m_0, m_1, \ldots\ldots m_{2^n-1} \right]$

Funkcijske vrednosti pa:

$$f = \left[f_0, f_1, \ldots\ldots f_{(2^n-1)} \right]^T$$

Skalarna preklopna funkcija je v novi simbolični izražavi:

$y = f(\mathbf{x}) = \mathbf{m}\ \sum\!\&\ \mathbf{f}$

Vektor **m** je z vektorjem **x** povsem določen.

Mintermski vektor **m** je dan z operacijo konjunkcije in ekvivalence:

$$\mathbf{m}(\mathbf{x}) = \mathbf{x} \,\&\, \equiv \mathbf{W}^T,$$

S tem so dani vsi elementi za PDNO:

$$y = f(\mathbf{x}) = [\, \mathbf{x} \,\&\, \equiv \mathbf{W}^T] \sum \& \, \mathbf{f}$$

Izjavnostna tabela dobi obliko:

x	y = f (**x**)
W	**f**

Pri posplošitvi predpostavimo vektorsko izhodno preklopno funkcijo:

f(**x**) preide v vektorsko funkcijo **y,** z njo pa preide tudi vektor **f** v matriko **D**.

Torej:

y = **f**(**x**) = [**x** & ≡ **W**^{T}] ∑ & **D** – matrična popolna disjunktivna normalna

To obliko vektorske preklopne funkcije, včasih imenujemo tudi strukturna popolna disjunktivna normalna oblika (SPDNO).

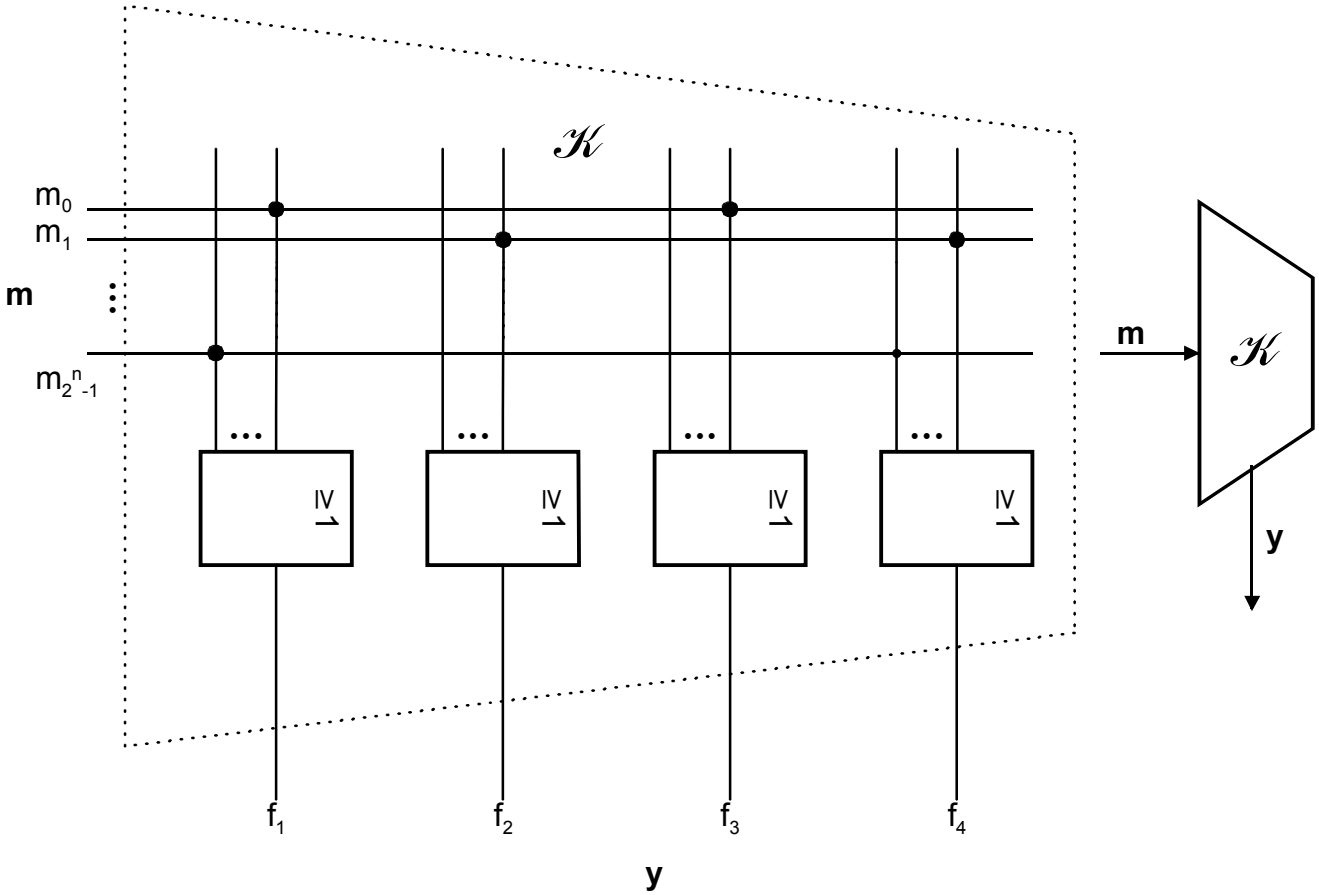
Pri vektorski preklopni funkciji dobi tudi izjavnostna tabela nove simbole:

x	**y** = f (**x**)
W	**D**

5.3 Standardna mnogopolna preklopna vezja

5.3.1 Kodirna vezja – kodirniki (encoder-ji)

$\mathbf{y} = \mathbf{m}\,\Sigma\,\&\,\mathbf{K}$; k_j^i = konstanta



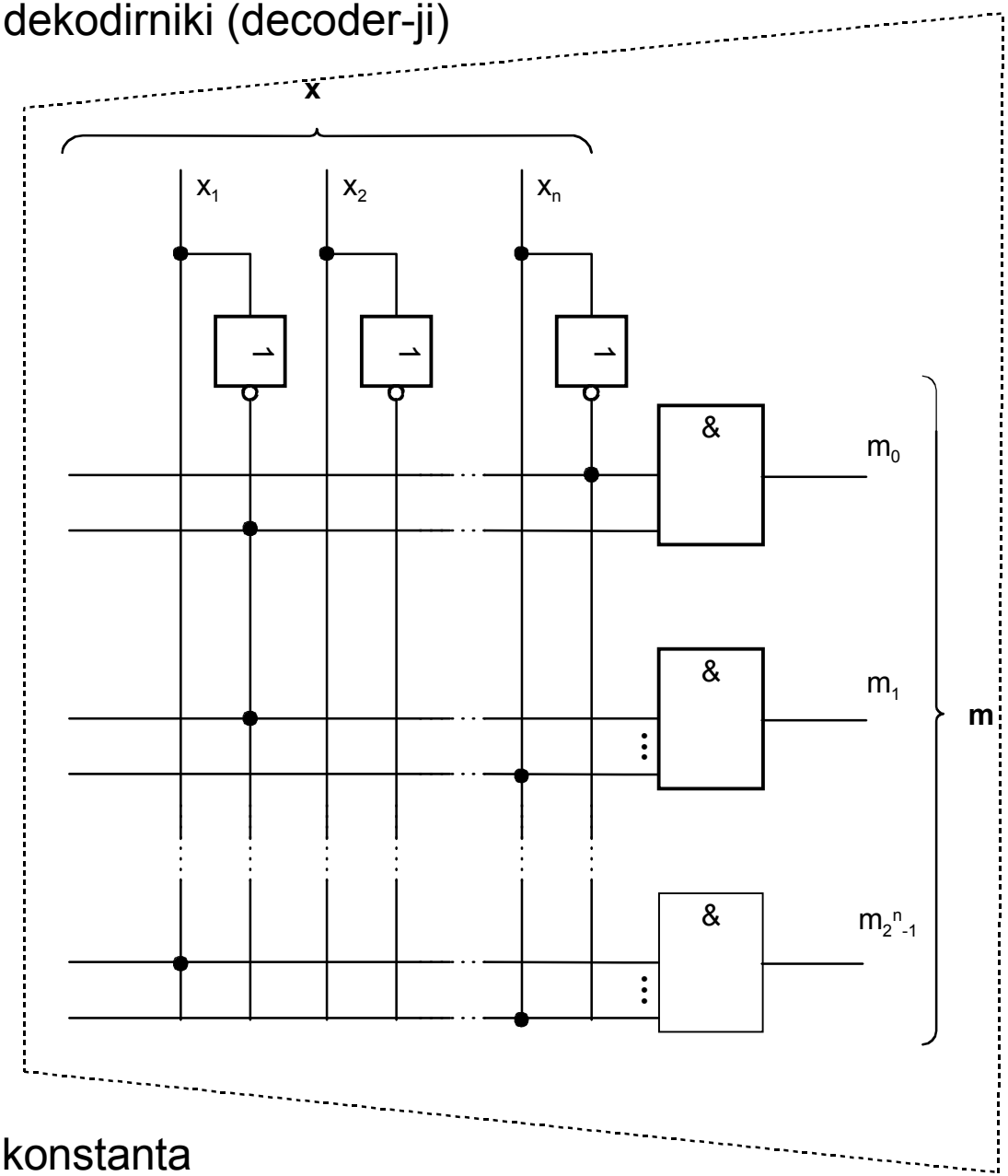
V konstantah matrike **K** je zapisano kodirno pravilo.

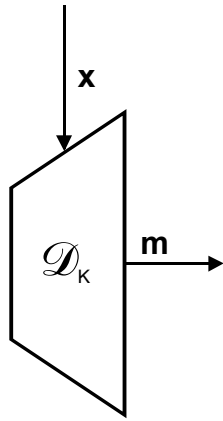
Pri kodirniku je število mintermov vedno večje od števila izhodnih funkcij – elementov vektorske izhodne funkcije.

Ta zožitev števila spremenljivk se odraža tudi na blokovnem simbolu kodirnika.

5.3.2. Dekodirna vezja – dekodirniki (decoder-ji)

Tu gre torej za »razširitev«; povečanje števila spremenljivk od vhoda proti izhodu.

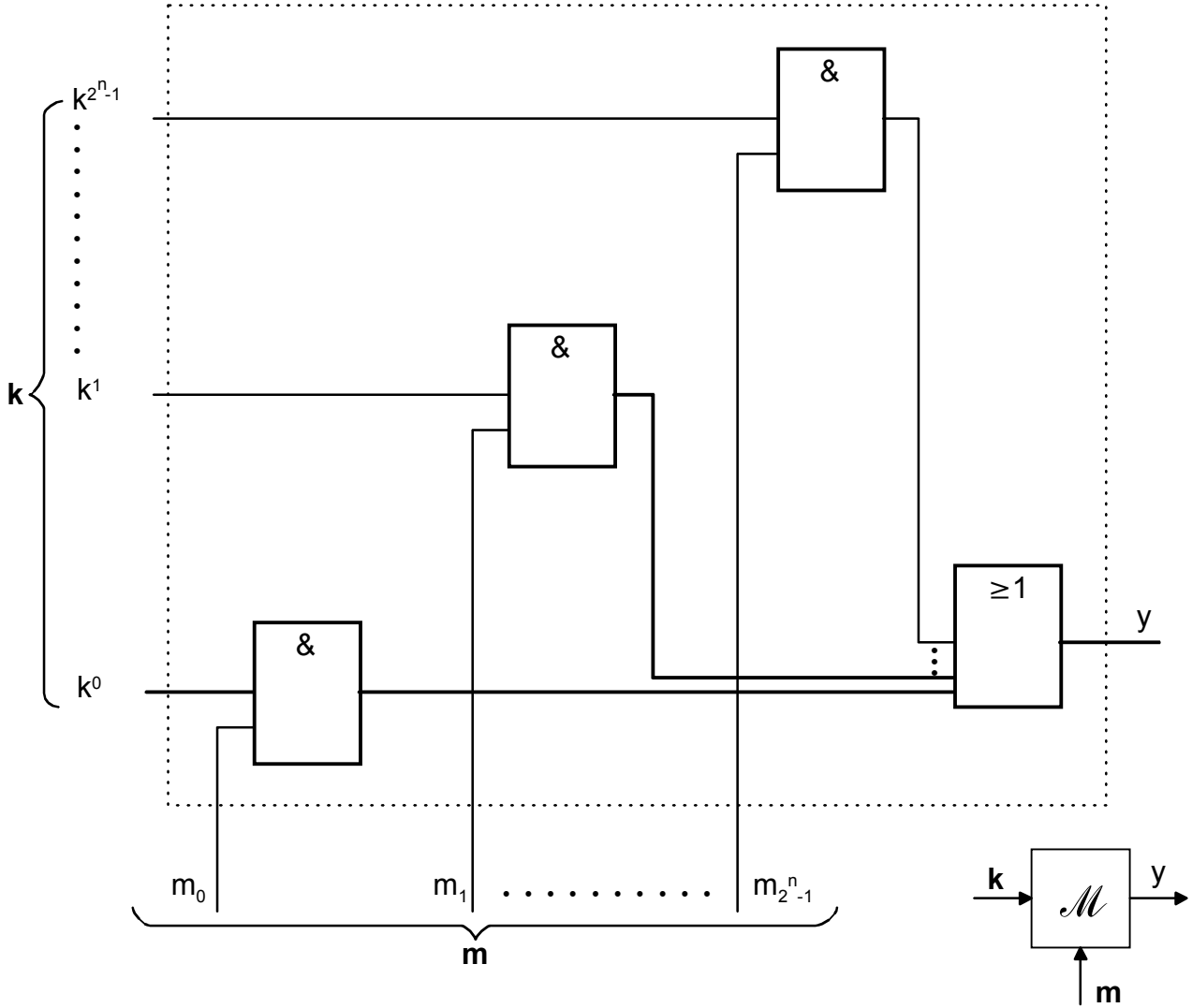




$\mathbf{m} = \mathbf{x}\ \& \equiv \mathbf{W}^{T}$; w_j^i = konstanta

5.3.3 Multipleksor (Multiplexer)

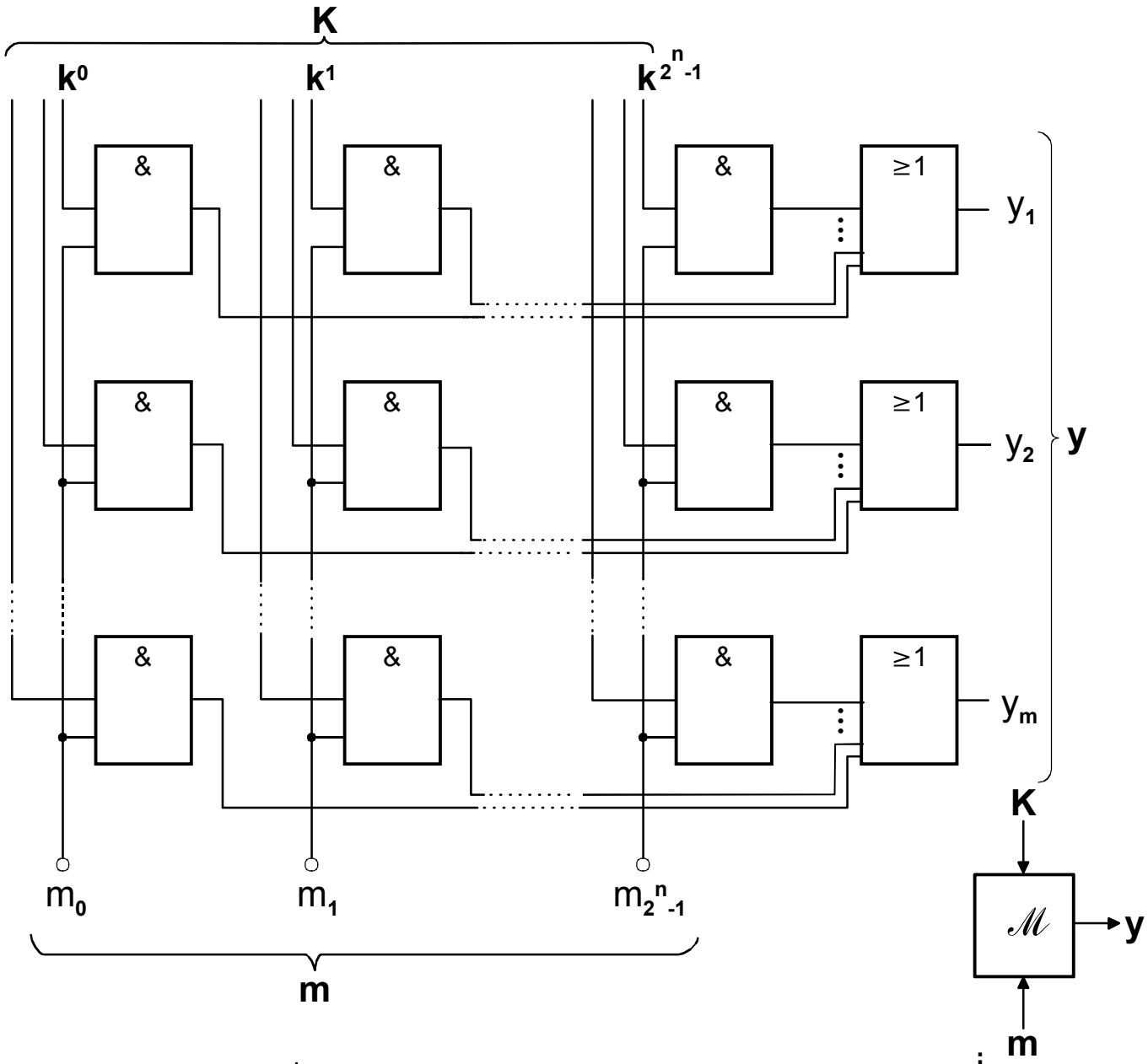
5.3.3.1. Skalarni neadresni multipleksor



Izhod je skalarna preklopna funkcija, ki jo določa enačba:

$y = \mathbf{m}\,\Sigma\,\&\,\mathbf{k}$; k^i = spremenljivka

5.3.3.2 Vektorski neadresni multipleksor

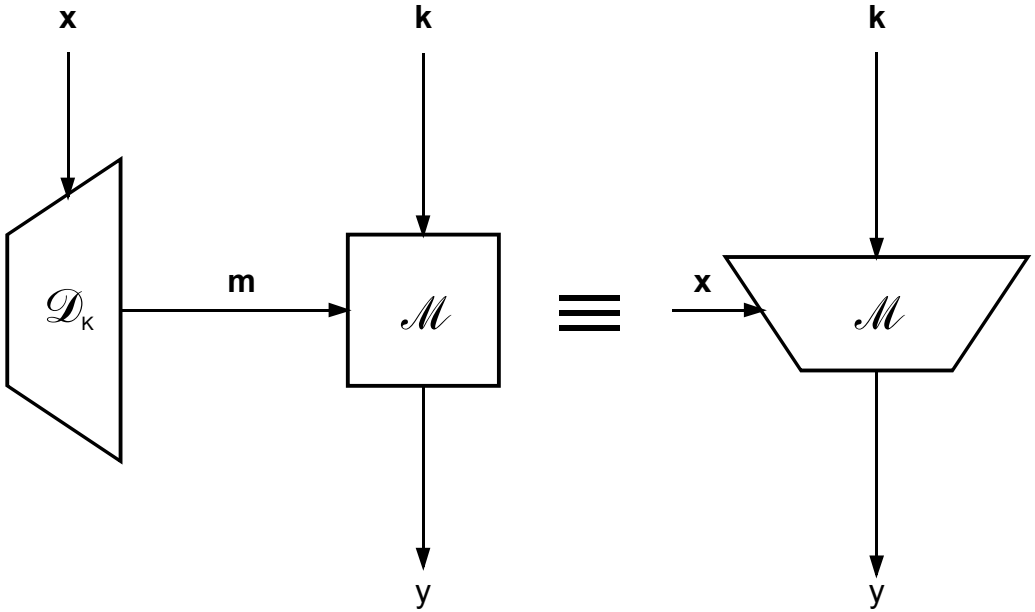


$\mathbf{y} = \mathbf{m}\,\Sigma\,\&\,\mathbf{K}$; k_j^i = spremenljivke, ki tvorijo vektorje k^i

5.3.3.3 Skalarni adresni multipleksor

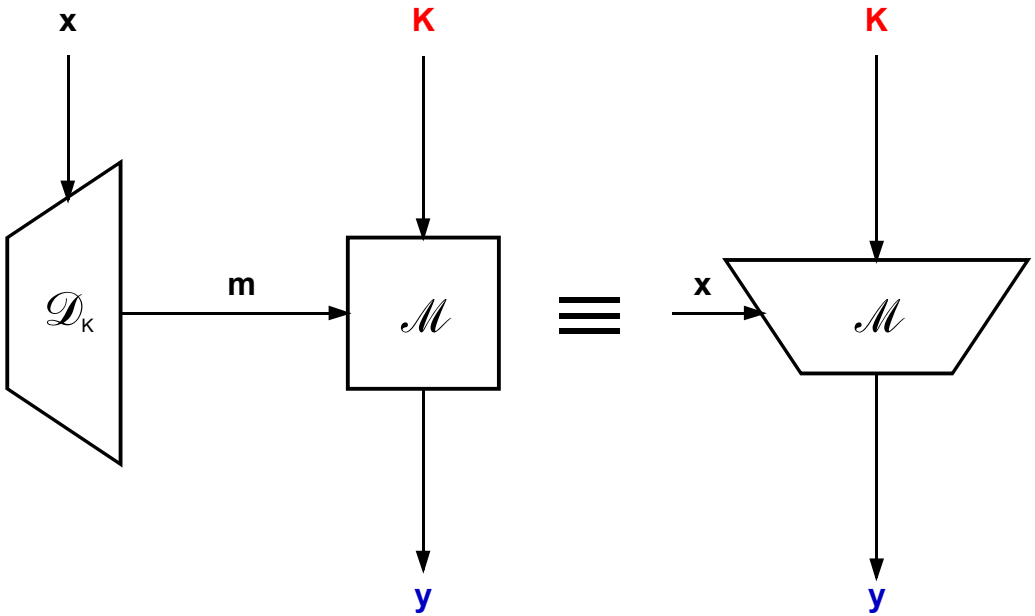
Dve elementarni operaciji: dekodiranje - multipleksiranje

Tako dobimo skalarni adresni multipleksor:



$y = (\mathbf{x} \,\&\equiv \mathbf{W}^{\mathbf{T}})\Sigma\,\&\,\mathbf{k}$; k^i = spremenljivka - skalarna

5.3.3.4 Vektorski adresni multipleksor



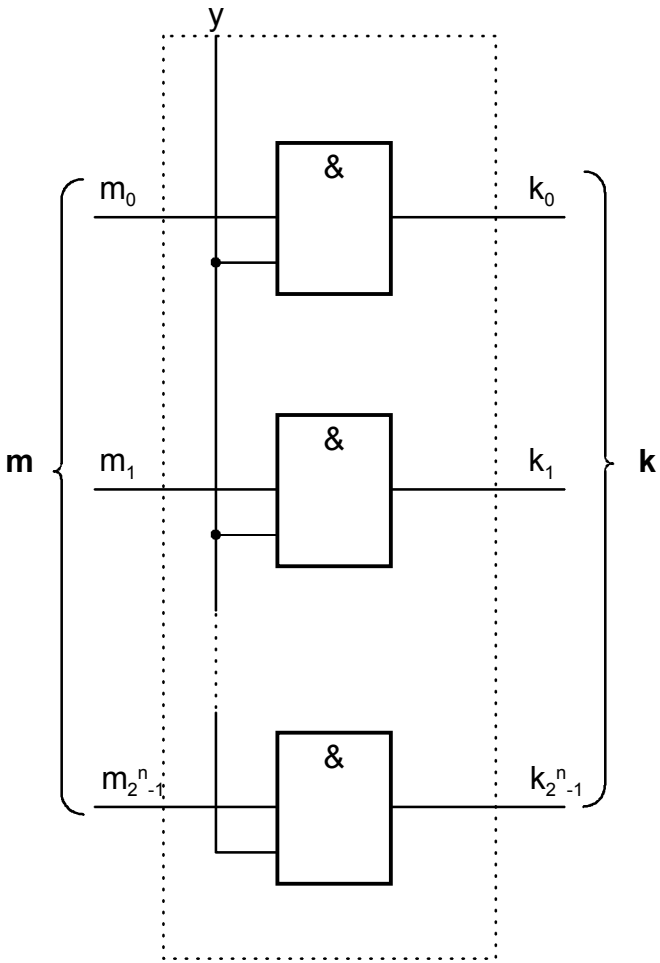
$\mathbf{y} = (\mathbf{x} \,\&\equiv\, \mathbf{W}^{\mathbf{T}})\Sigma\,\&\,\mathbf{K}$; k_j^i = spremenljivke, ki tvorijo vektorje $\mathbf{k}^{\mathbf{i}}$

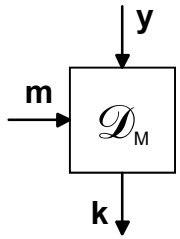
5.3.4 Demultipleksor (Demultiplexer)

5.3.5.1 Neadresni skalarni demultipleksor

Demultipleksiranje je obratna operacija od multipleksiranja, zato imamo tudi tu enake različice, kot pri multipleksorjih.

Skalarni neadresni demultipleksor



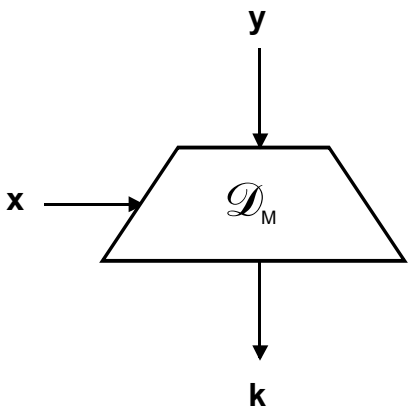


Izhodna funkcija navadnega - neadresnega skalarnega demultipleksorja je tako:

$\mathbf{k} = \mathbf{m}^{\mathbf{T}} \Sigma \,\&\, y$ - y je skalarna funkcija

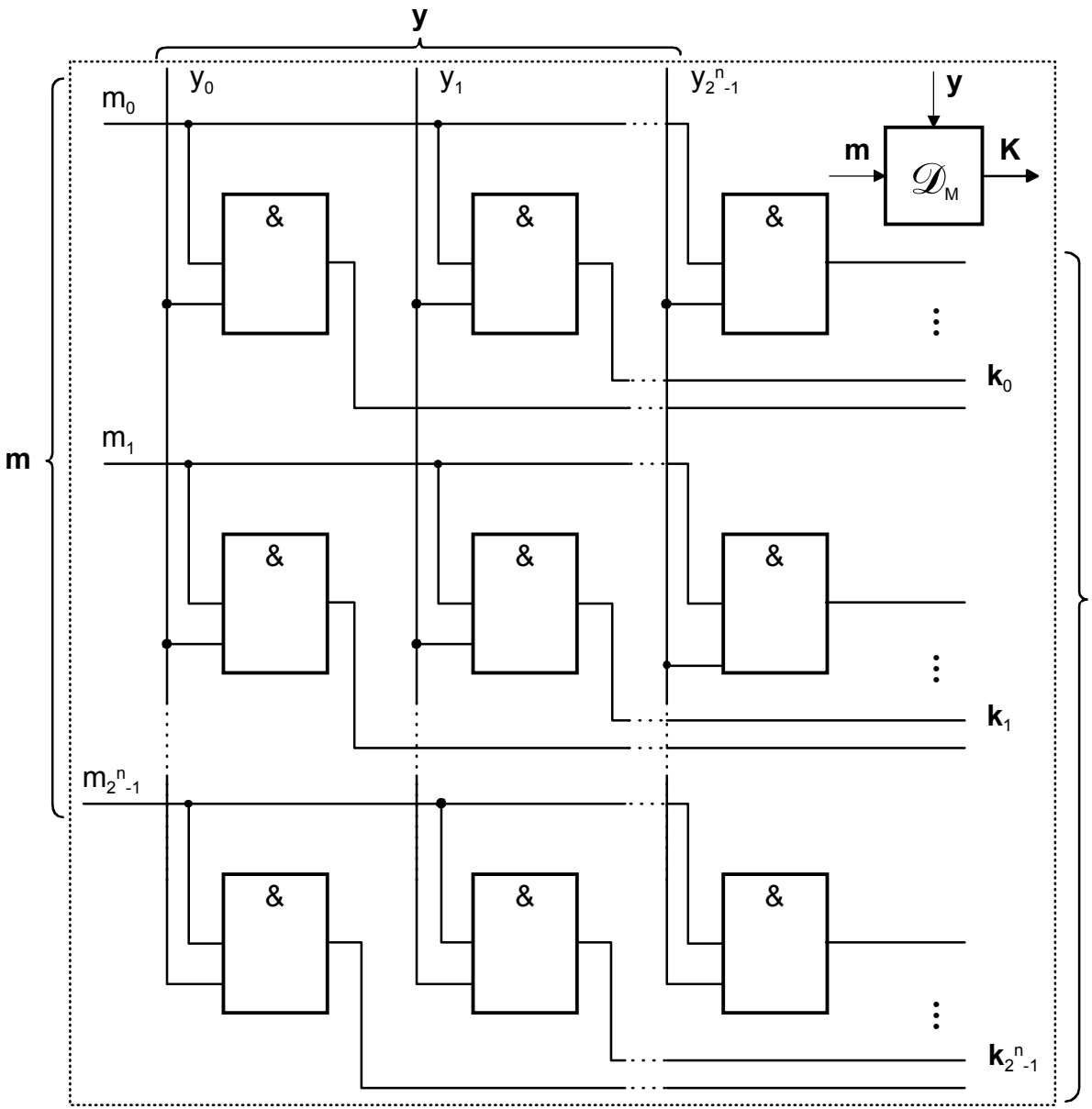
5.3.5.2 Adresni skalarni demultipleksor

Vhodna funkcija je tu enaka kot prej, naslavljanje pa je omogočeno vektorju neodvisnih spremenljivk.



$\mathbf{k} = (\mathbf{x}\& \equiv \mathbf{W^T})^T \Sigma \& \, y$ - y je še vedno skalarna funkcija

5.3.5.3 Neadresni vektorski demultipleksor

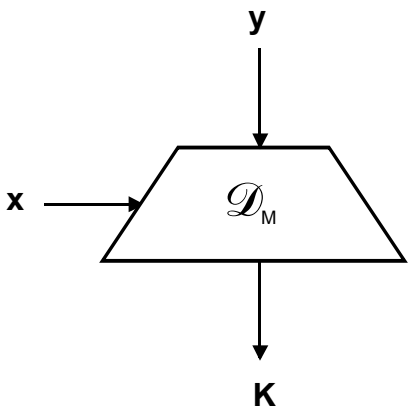


Izhod je torej matrika **K**, ki daje večbitne podatke zopet v paralelni obliki.

Vsak minterm omogoči nastop ustrezne vrstice matrike **K**, ki predstavlja trenutni izhodni vektor : $\mathbf{K} = \mathbf{m}^T \Sigma \,\&\, \mathbf{y}$ - **y** je sedaj vektorska vhodna funkcija

5.3.5.4 Adresni vektorski demultipleksor

Z dodajanjem dekodirnika pridemo do kompleksne strukture vektorskega adresnega demultipleksorja, ki ga z blokovnim simbolom predstavimo takole:



$\mathbf{K} = (x\& \equiv W^T)^T \Sigma \& \mathbf{y}$ - **y** je vektorska vhodna funkcija

5.3.5 Programibilna preklopna vezja

Osnovna ideja programibilnih preklopnih vezij je prilagodljivost standardizirane preklopne strukture za opravljanje čim večjega števila funkcij odločanja in pomnjenja, ki jih lahko uporabnik sam določi; to je brez pomoči proizvajalca.

Zapišimo še enkrat popolno dijunktivno normalno obliko vektorske preklopne funkcije:

$\mathbf{y} = (\mathbf{x} \& \equiv \mathbf{W^T})\Sigma \& \mathbf{K}$; k_j^i = spremenljivka - skalarna

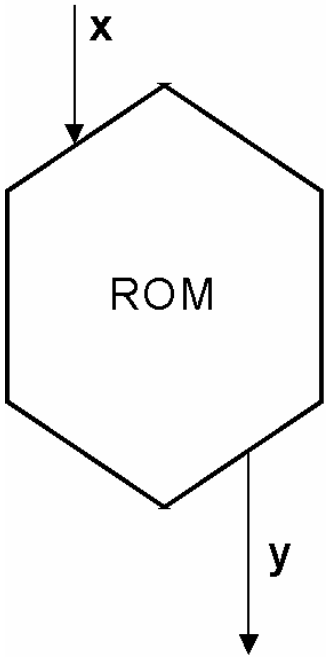
Programibilno preklopno vezje ali strukturo lahko sedaj definiramo, kot kombinacijo dekodirnika in kodirnika, pri katerem lahko uporabnik sam določa povezave v poljih AND in – ali OR.

Tip vezja	Polje - AND	Polje - OR
Programibilni bralni pomnilnik	Fiksno	Programibilno
Programibilne logične mreže - PLA	Programibilno	Programibilno
Programibilna logična polja - PAL	Programibilno	Fiksno

5. 3. 6.1 Bralni pomnilnik (Read Only Memory - ROM)

Uporaba: pomnjenje stalnih vrednosti in realizacija vektorskih preklopnih funkcij.

Blok simbol bralnega pomnilnika:



Naslavljanje pomnilnika:

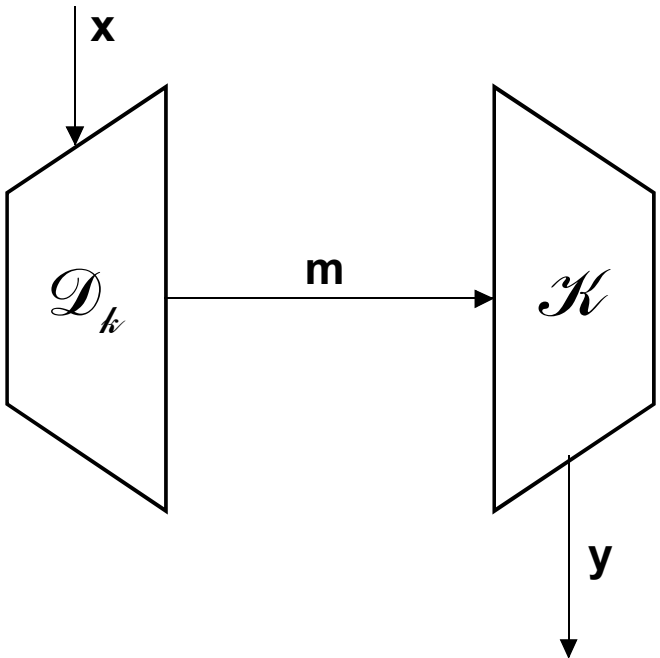
$\mathbf{m} = \mathbf{x}\,\&\equiv \mathbf{W}^{\mathbf{T}}$; w_j^i = konstanta; $\mathbf{W} \Rightarrow \mathbf{D}$; $w_j^i \Rightarrow d_j^i$

Čitanje pomnilnika

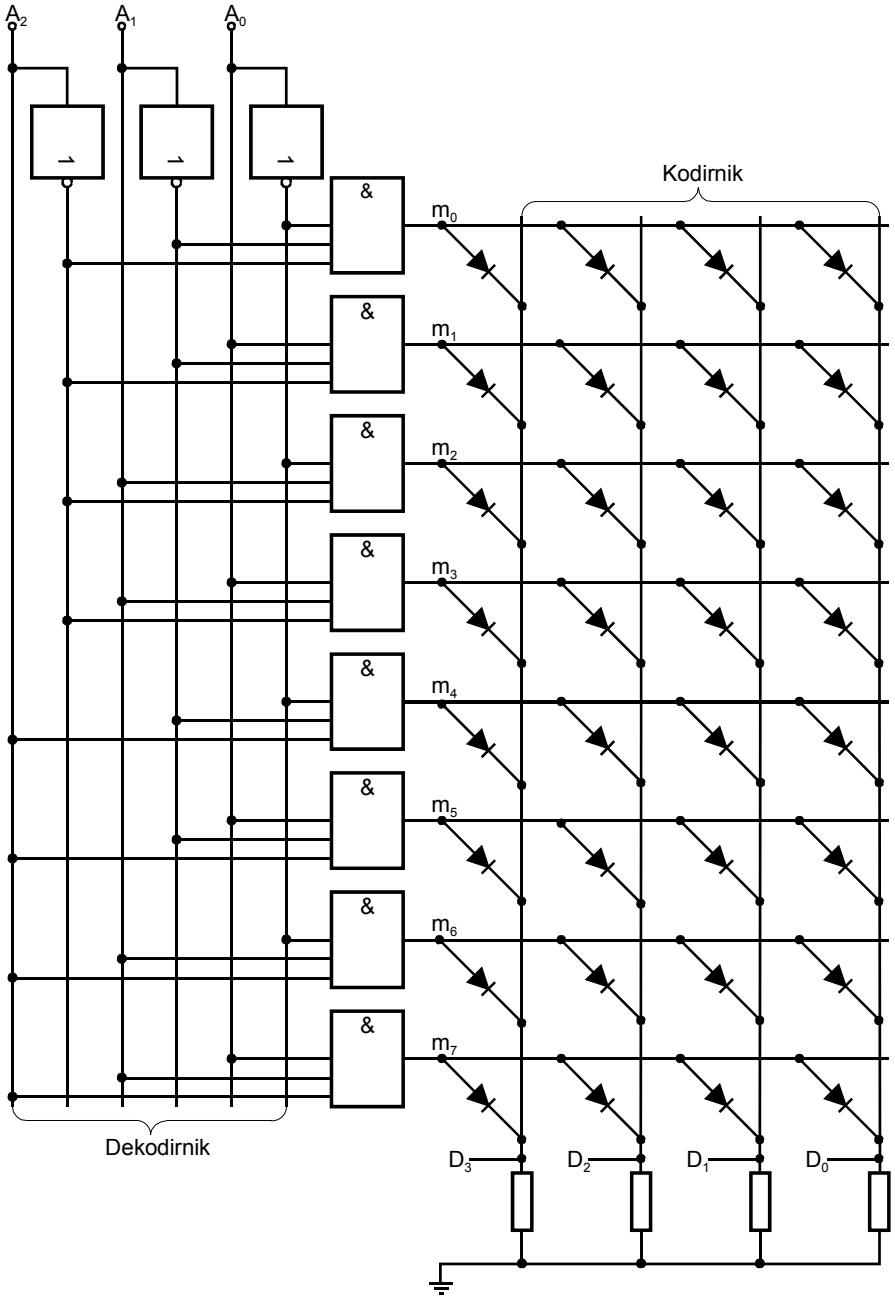
$\mathbf{y} = \mathbf{m}\,\Sigma\,\&\,\mathbf{K}$; k_j^i = konstanta

Z vstavitvijo prvega izraza v drugega dobimo: popolno disjunktivno normalno obliko vektorske preklopne funkcije – PDNOVPF:

$\mathbf{y} = (\mathbf{x}\,\&\equiv \mathbf{D}^{\mathbf{T}})\Sigma\,\&\,\mathbf{K}$; k_j^i = konstanta, ki je bila sprogramirana



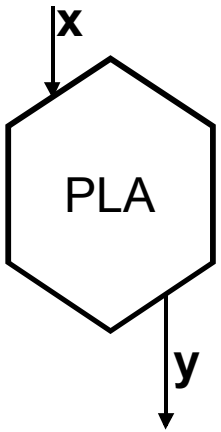
Podrobno vezje bralnega pomnilnika



5.3.6.2 PLA vezja

Ta vezja imajo, kot že vemo, programibilni obe polji – dekodirno in kodirno polje.

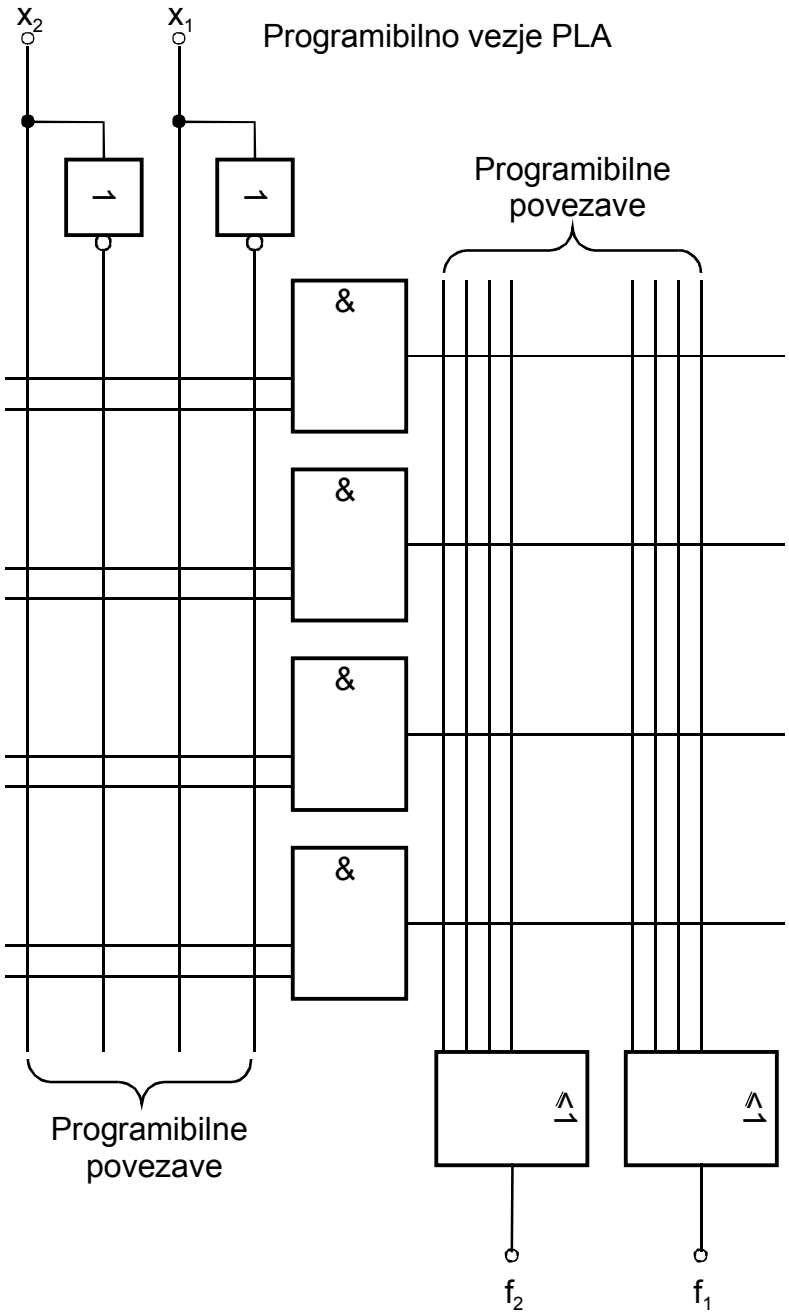
Zanja uporabljamo enak blokovni simbol, kot za ROM, saj se po osnovni notranji strukturi ne razlikujejo od njega:



Tudi izhodiščna enačba je enaka, saj gre za dijunktivno normalno obliko vektorske preklopne funkcije:

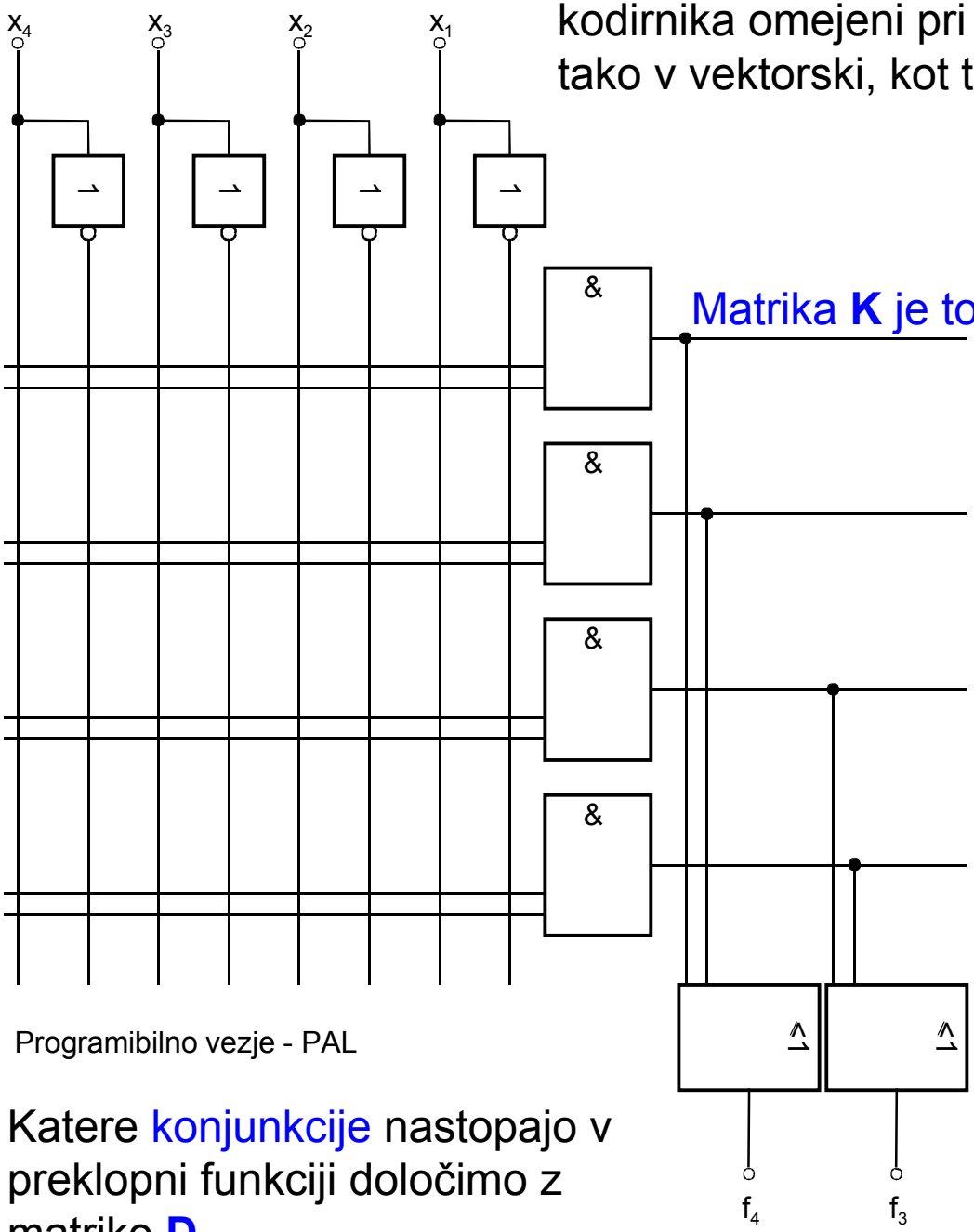
$$\mathbf{y} = (\mathbf{x} \,\&\, \equiv \mathbf{D}^{T})\Sigma \,\&\, \mathbf{K}\ ; \qquad k_j^i = \text{konstanta}$$

Podroben simbolični diagram PLA vezij



5.3.6.3 PAL vezja:

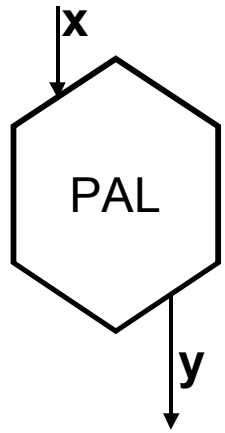
Pri uporabi PAL vezij smo zaradi vnaprej določenega kodirnika omejeni pri številu konjunkcij, ki lahko nastopajo tako v vektorski, kot tudi v skalarni preklopni funkciji.



Matrika **K** je torej določena

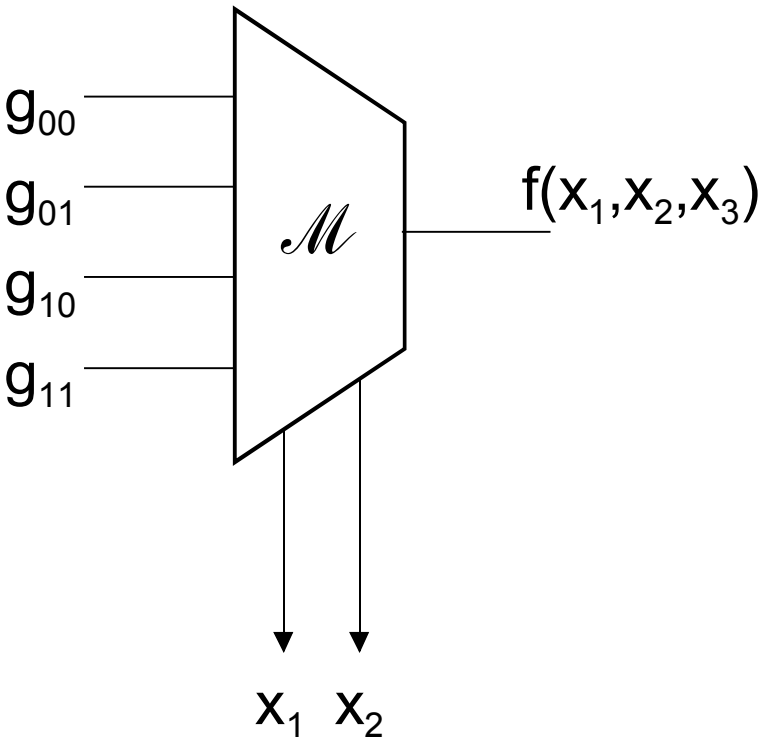
Programibilno vezje - PAL

Katere konjunkcije nastopajo v preklopni funkciji določimo z matriko **D**.



5. 4 Realizacija skalarnih in vektorskih preklopnih funkcij s preklopnimi strukturami

5. 5. 1 Multipleksor



Funkcija multipleksiranja predstavlja funkcijsko polnost, njegov izhod daje preklopno funkcijo v popolni disjunktivni normalni obliki, vsi elementi matrike **K** pa so spremenljivke.

V splošnem bo imela preklopna funkcija množico neodvisnih spremenljivk:

X = $\{x_1, x_2, \ldots\ldots\ldots, x_n\}$.

Te spremenljivke pri realizaciji funkcije z multipleksorjem razdelimo v dva dela; v podatkovne in adresne spremenljivke.

Podatkovne in adresne spremenljivke:

$\mathbf{X}_d = \{x_{d1}, x_{d2}, \ldots\ldots\ldots, x_{dd}\} \supset \mathbf{X}$,

$\mathbf{X}_a = \{x_{a1}, x_{a2}, \ldots\ldots\ldots, x_{aa}\} . \supset \mathbf{X}$,

razdelimo na takšen način, da vedno velja:

$\mathbf{X}_a \cup \mathbf{X}_d = \mathbf{X}$

$\mathbf{X}_a \cap \mathbf{X}_d = 0$

Za adresne spremenljivke najprej izberimo »a« zaporednih neodvisnih spremenljivk in naj bo:

a = 2. Torej bo:

$x_{a1} = x_1$ in $x_{aa} = x_2$.

$f(x_1, x_2, \ldots x_n) = \bar{x}_1 \bar{x}_2 g_{00} + \bar{x}_1 x_2 g_{01} + x_1 \bar{x}_2 g_{10} + x_1 x_2 g_{11}$

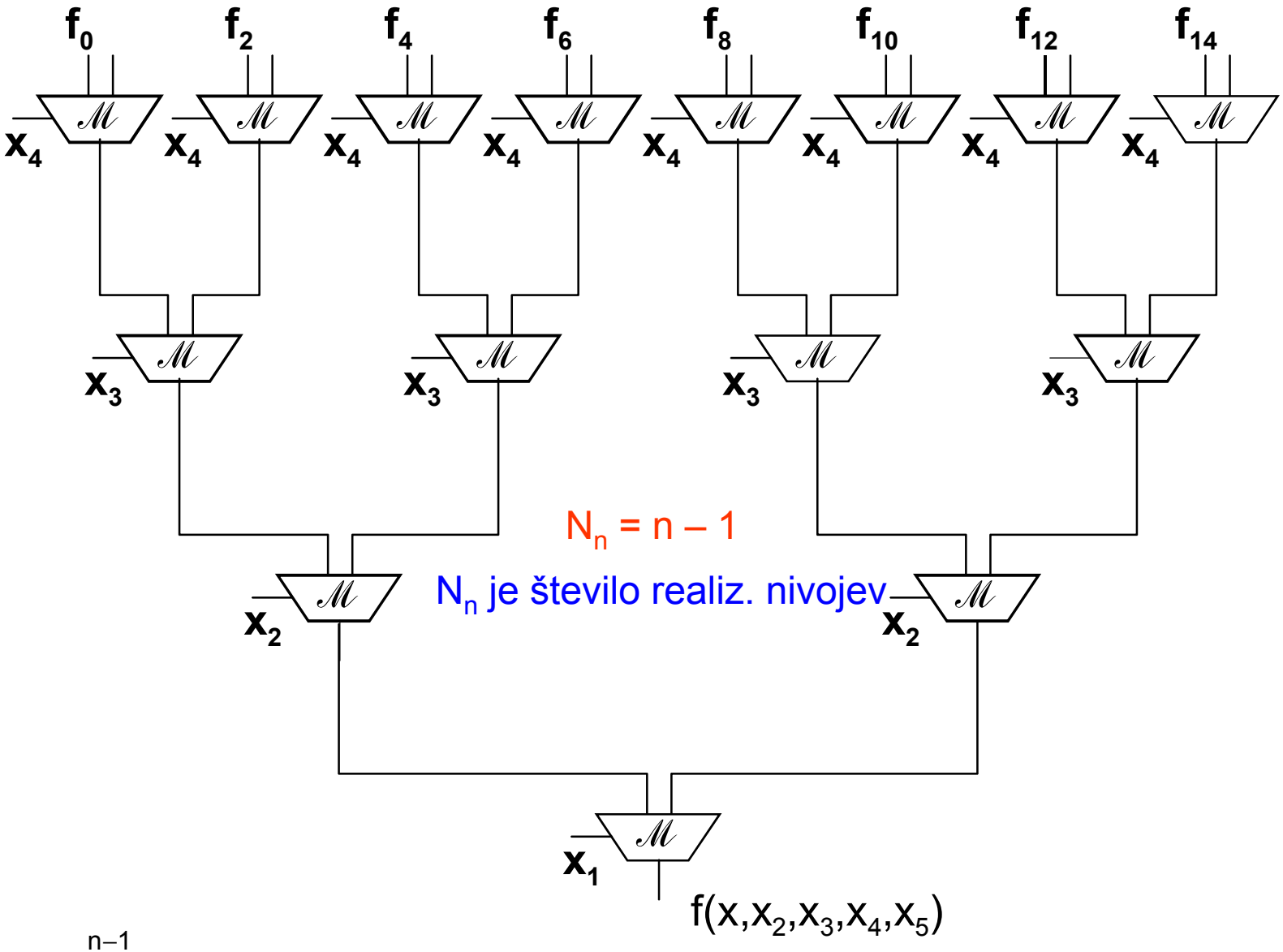
kjer je na primer:

$g_{01} = f(0, 1, x_3, \ldots\ldots x_n)$.

Funkcije g_{00}, g_{01}, g_{10}, g_{11} so funkcijski ostanki, ki jih kot vemo, lahko obravnavamo na enak način kot izhodiščno funkcijo, dokler ne začnejo nastopati konstante ali posamične neodvisne spremenljivke.

Če ima preklopna funkcija »n« neodvisnih spremenljivk potrebujemo za takšno realizacijo multipleksor z »n - 1« adresnimi vhodi.

V drugem primeru, ko ima multipleksor samo en adresni vhod se število multipleksorjev širi po nivojih z 2^i, kot to vidimo na spodnji sliki:



$N_M = \sum_{i=0}^{n-1} 2^i$; kjer je N_M - število vseh potrebnih multipleksorjev

Pri vseh vmesnih kombinacijah, ko je $1 < a < n - 1$ bo število vseh multipleksorjev odvisno od »a« pri čemer morata biti obe števili celi števili.

Po tem premisleku zlahka ugotovimo, da je :

$$(n - 1)/a \leq N_n \leq (n - 1)/a + 1$$

in

$$N_M = \sum_{i=1}^{N_n} 2^{ai}$$

Da gornja enačba velja tudi za obe skrajnosti, se lahko prepričamo takole:

V prvem primeru je bil $a = n - 1$, celo število iz neenačbe je 1, vsota pa ima tudi samo en člen, to je $2^0 = 1$.

V drugem primeru, ko je $a = 1$ je celo število iz neenačbe $n - 1$ in število vseh multipleksorjev je:

$$N_M = \sum_{i=0}^{N_n} 2^i = \sum_{i=0}^{n-1} 2^i = \sum_{i=0}^{3} 2^i = 2^0 + 2^1 + 2^2 + 2^3 = 1 + 2 + 4 + 8 = 15$$

Kadar pa je $1 < a < n - 1$ pa ni več vseeno kako razporedimo adresne spremenljivke, ker lahko prinese poenostavljanje znaten prihranek vezja.

Število vseh možnih minimizacijskih postopkov je enako številu kombinacij n spremnljivk s po a elementov.

$N_k = \frac{n!}{a!(n-a)!}$; kjer je N_k število vseh možnih kombinacij.

Število vseh Veitch-evih diagramov za vsak minimizacijski postopek je potem 2^a.

Vsak posamezen diagram pa ima n-a neodvisnih spremenljivk

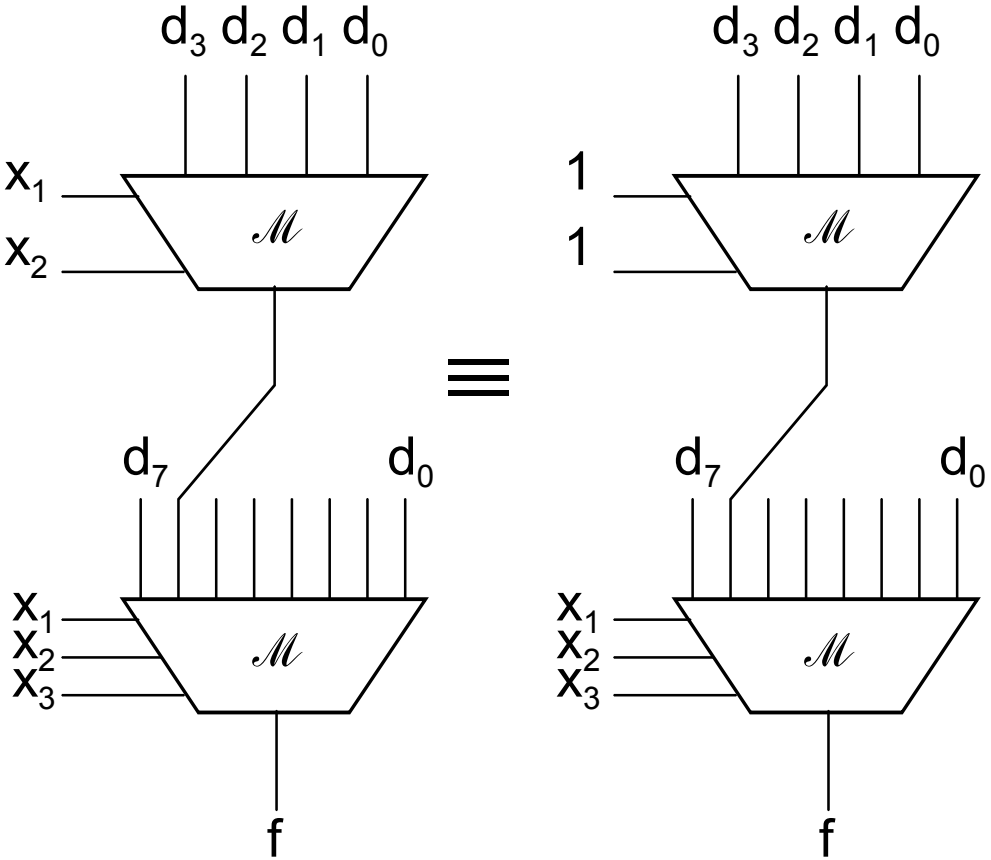
Redundanca pri adresnih spremenljivkah

Kadar število n-1 ni deljivo z a se število adresnih spremenljivk ne ujema s številom adresnih vhodov.

To pa pomeni, da se določene adresne spremnljivke ponovijo na različnih nivojih drevesne strukture.

Takšno adresno spremenljivko lahko nadomestimo kar s konstanto.

Primer redundance pri adresnih spremenljivkah:



$$f = \ldots + x_1 x_2 \bar{x}_3 d_6 + \ldots$$

$$f = \ldots + x_1 x_2 \bar{x}_3 (\bar{x}_1 \bar{x}_2 d_0 + \bar{x}_1 x_2 d_1 + x_1 \bar{x}_2 d_2 + x_1 x_2 d_3) + \ldots$$

$$f = \ldots + x_1 x_2 \bar{x}_3 (00 d_0 + 01 d_1 + 10 d_2 + 11 d_3) + \ldots$$

$$f = \ldots + x_1 x_2 \bar{x}_3 (d_3) + \ldots$$

5. 5. 2 Uporaba bralnih pomnilnikov

Kot že vemo je izhodna funkcija bralnega pomnilnika vektorska popolna disjunktivna normalna oblika:

$$\mathbf{y} = (\mathbf{x}\,\&\equiv \mathbf{D}^{\mathbf{T}})\Sigma\,\&\,\mathbf{K}\;; \qquad k^i_j = \text{konstanta}$$

Vzemimo za primer naslednjo vektorsko preklopno funkcijo:

x_1	x_2	x_3	f_1	f_2	f_3	f_4
0	0	0	0	1	1	1
0	0	1	1	0	0	0
0	1	0	1	0	1	1
0	1	1	1	1	0	0
1	0	0	0	1	1	0
1	0	1	1	0	0	1
1	1	0	1	0	1	1
1	1	1	1	1	1	0

Vsako od komponent vektorske preklopne funkcije lahko zapišemo v njeni popolni disjunktivni normalni obliki.

Tako je na primer:

$$f_1 = \bar{x}_1\bar{x}_2x_3 + \bar{x}_1x_2\bar{x}_3 + \bar{x}_1x_2x_3 + x_1\bar{x}_2x_3 + x_1x_2\bar{x}_3 + x_1x_2x_3$$

Njena minimalna disjunktivna normalna oblika pa je:

$$f_1 = x_2 + x_3$$

Pri uporabi bralnega pomnilnika za realizacijo preklopnih funkcij pa moramo opozoriti še na eno slabost.

Ker v njem lahko realiziramo le popolno disjunktivno normalno obliko ne moremo izkoristiti minimizacije in s tem prihranka vezja.

Postavlja se vprašanje ali se temu nebi dalo izogniti?

Vzemimo obširnejši primer funkcije s šestimi neodvisnimi spremenljivkami, ki zavzame vrednost 1 pri mintermih z indeksi: 4, 5, 15, 20, 29, 41, 42, 45, 47, 53, 58, 61, 63.

$$f = \sum^{6}(4,5,15,20,29,41,42,45,47,53,58,61,63)$$

Neposredno realizacijo te funkcije bi lahko izvedli na primer 64 x 4 ROM vezjem, oziroma štirimi 16 x 4 ROM vezji.

Naredimo si tabelo nastopajočih mintermov:

Mintermski indeks	Spremenljivki		Spremenljivke			
	x_1	x_2	x_3	x_4	x_5	x_6
4	0	0	0	1	0	0
5	0	0	0	1	0	1
15	0	0	1	1	1	1
20	0	1	0	1	0	0
29	0	1	1	1	0	1
41	1	0	1	0	0	1
42	1	0	1	0	1	0
45	1	0	1	1	0	1
47	1	0	1	1	1	1
53	1	1	0	1	0	1
58	1	1	1	0	1	0
61	1	1	1	1	0	1
63	1	1	1	1	1	1

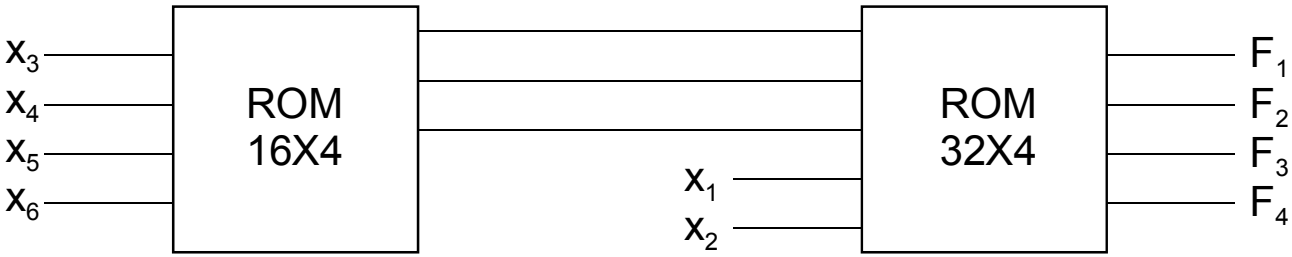
Poiščemo vse enake konjunkcije dolžine: $x_3x_4x_5x_6$.

Z njimi naredimo novo tabelo:

spremenljivke				kodirane spremenljivke		
x_3	x_4	x_5	x_6	z_1	z_2	z_3
0	1	0	0	0	0	0
0	1	0	1	0	0	1
1	0	0	1	0	1	0
1	0	1	0	0	1	1
1	1	0	1	1	0	0
1	1	1	1	1	0	1

Tem izhodnim spremenljivkam moramo dodati še spremenljivki x_1 in x_2, ki v našem primeru zavzemata vse možne vrednosti, da bomo lahko realizirali podano funkcijo.

Na drugem nivoju potrebujemo ROM velikosti 32 x 1, kar praktično pomeni 32 x 4.



V tem drugem pomnilniku bo na trinajstih mestih izbranega stolpca zapisana enica, na preostalih devetnajstih pa ničla.

Mesta, kjer mora ostati zapisana enica, so podana v naslednji tabeli; pri tem smo za našo funkcijo rezervirali prvi stolpec, ostali trije pa so še prosti in jih lahko uporabimo za nek drug namen.

Vhodi		Spremenljivke			Izhodi			
x_1	x_2	Z_1	Z_2	Z_3	f_1	f_2	f_3	f_4
0	0	0	0	0	1			
0	0	0	0	1	1			
0	0	1	0	1	1			
0	1	0	0	0	1			
0	1	1	0	0	1			
1	0	0	1	0	1			
1	0	0	1	1	1			
1	0	1	0	0	1			
1	0	1	0	1	1			
1	1	0	0	1	1			
1	1	0	1	1	1			
1	1	1	0	0	1			
1	1	1	0	1	1			

Opisani postopek lahko uporabimo tudi v primeru, ko moramo realizirati več različnih funkcij, kjer je prihranek prostora v pomnilniku lahko še večji. Pri vsem tem pa ne smemo pozabiti, da smo s kaskadno vezavo zmanjšali hitrost odziva vezja.

Na vektorsko preklopno funkcijo pa lahko gledamo še iz drugega zornega kota.

Vzemimo vektorsko funkcijo iz prvega primera

Posamezne funkcijske vrednosti te funkcije:

f_{01}, f_{02}, f_{03}, f_{04}; naj bodo biti besede:

d_{01}, d_{02}, d_{03}, d_{04}, ki jo naslavlja mintrem m_0;

vhodne neodvisne spremenljivke pa preimenujmo v adresne spremenljivke :

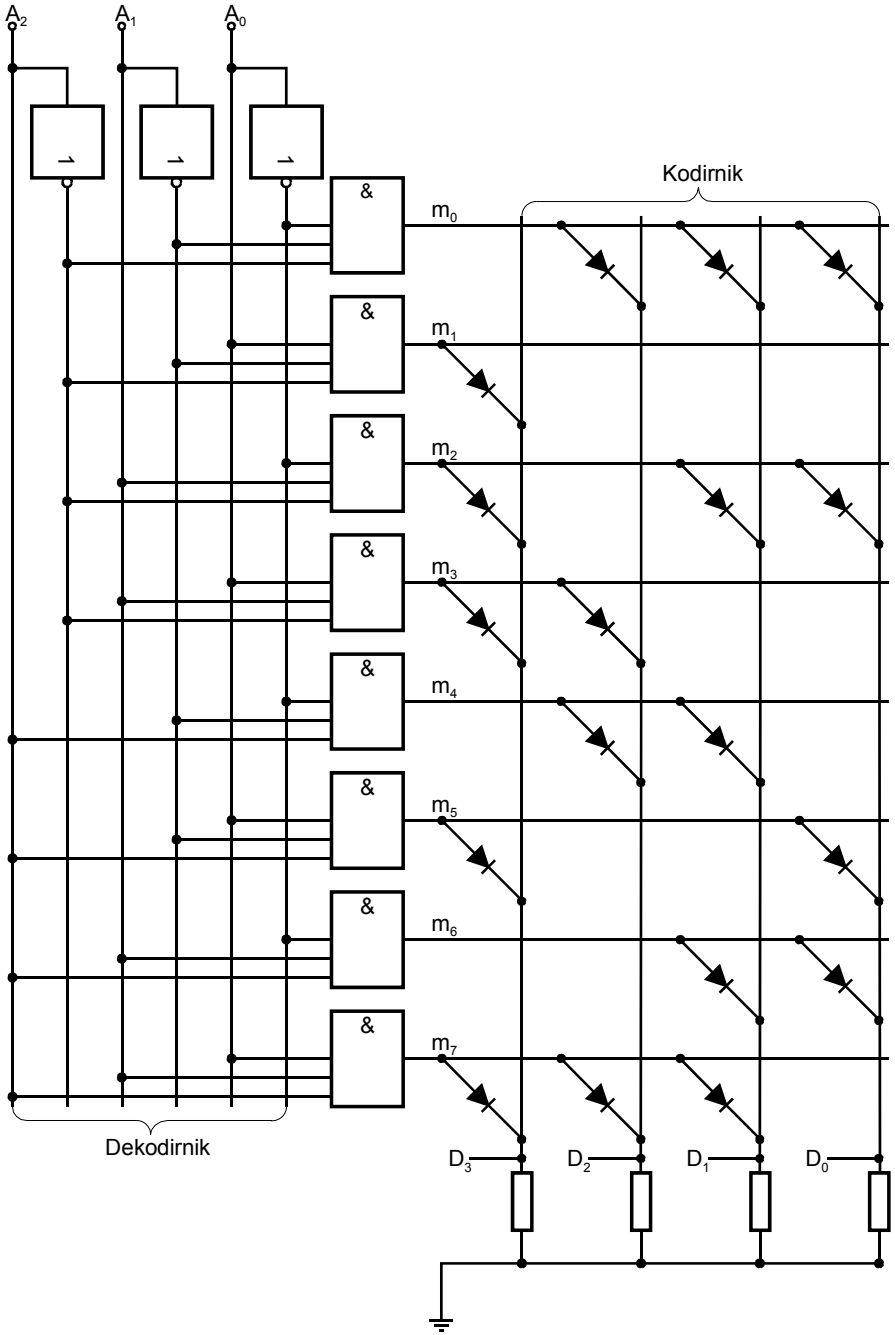
A_0, A_1, A_2.

Logično enaka izjavnostna tabela je z novimi oznakami naslednja:

A_2	A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	1	1	1
0	0	1	1	0	0	0
0	1	0	1	0	1	1
0	1	1	1	1	0	0
1	0	0	0	1	1	0
1	0	1	1	0	0	1
1	1	0	1	0	1	1
1	1	1	1	1	1	0

Nespremenjeno vezje je prevzelo vlogo bralnega pomnilnika, po katerem nosi tudi ime.

Sprogramirano ROM vezje za obravnavano vektorsko preklopno funkcijo



Realizacija vektorskih preklopnih funkcij s PLA in PAL strukturami

Njihova obsežnost je poleg števila vhodov odvisna tudi od števila konjunkcij, ki jih je možno programirati ter od števila izhodov

Obsežnost za enako število vhodov je bistveno manjša od ROM - vezij

Primerjava: FPLA v TTL izvedbi – 16 vhodov, 48 konjunkcij in 8 izhodov

Kompleksnost se odraža pri realizaciji funkcij, ker je na razpolago le omejeno število konjunkcij

Upoštevati moramo tudi to, da izolirana spremenljivka še vedno zahteva svojo konjunkcijo

Velikost vezja vedno določa najdaljša konjunkcija ne glede na položaj v vektorski funkciji

Potrebna je torej minimizacija

Najprej izvedemo klasično minimizacijo

Od tu dalje pa postopek zavisi od izbranega tipa vezja PLA oziroma PAL

Pri PLA nas omejuje število konjunkcij in število spremenljivk v konjunkcijah

Pri PAL pa še dodatno število konjunkcij v posamični komponenti vektorske preklopne funkcije

Primer realizacije s PLA

$$y_1 = x_1 + \overline{x}_2\overline{x}_3$$
$$y_2 = x_1\overline{x}_3 + x_1x_2$$
$$y_3 = \overline{x}_2\overline{x}_3 + x_1x_2$$
$$y_4 = \overline{x}_2x_3 + x_1$$

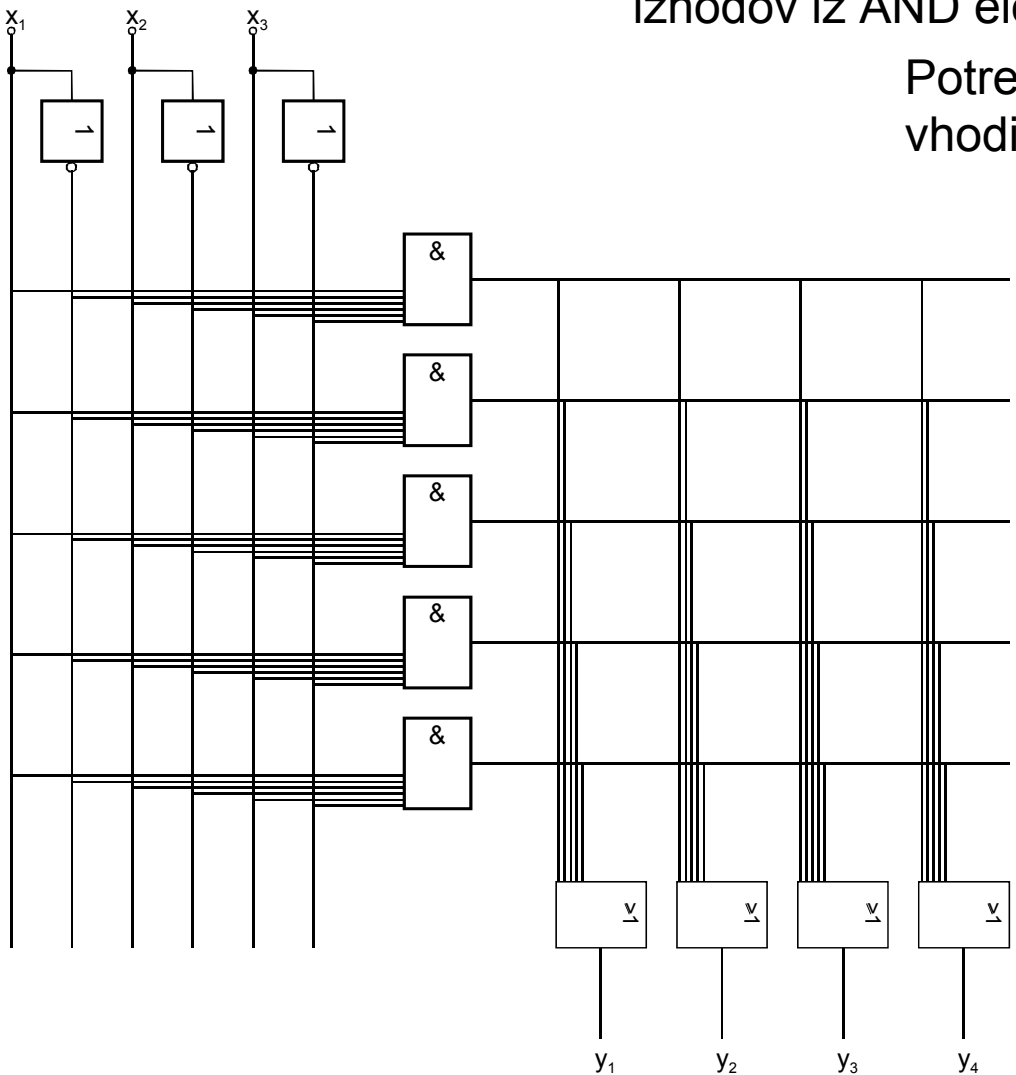
Vemo, da je potrebno število spremenljivk 3: x_1, x_2, x_3

število različnih konjunkcij $(x_1, \overline{x}_2\overline{x}_3, x_1\overline{x}_3, x_1x_2, \overline{x}_2x_3)$

število izhodnih funkcij – elementov vektorske funkcije 4

Kar ustreza številu vhodov v AND elemente, številu izhodov iz AND elementov in številu OR elementov

Potrebujemo torej vezje z najmanj tremi vhodi, petimi konjunkcijami in štirimi izhodi



Za določitev povezav vhodnih spremenljivk in povezav med AND in OR poljem uporabimo povezovalno tabelo – prirejeno pravilnostno tabelo.

Povezovalna tabela za PLA:

Kon-junk-cije	VHODI			IZHODI			
	x_1	x_2	x_3	f_1	f_2	f_3	f_4
$x_1\ x_2$	1	1	-	0	1	1	0
$\bar{x}_2\ x_3$	-	0	1	0	0	0	1
$x_1\ \bar{x}_3$	1	-	0	0	1	0	0
$\bar{x}_2\ \bar{x}_3$	-	0	0	1	0	1	0
x_1	1	-	-	1	0	0	1

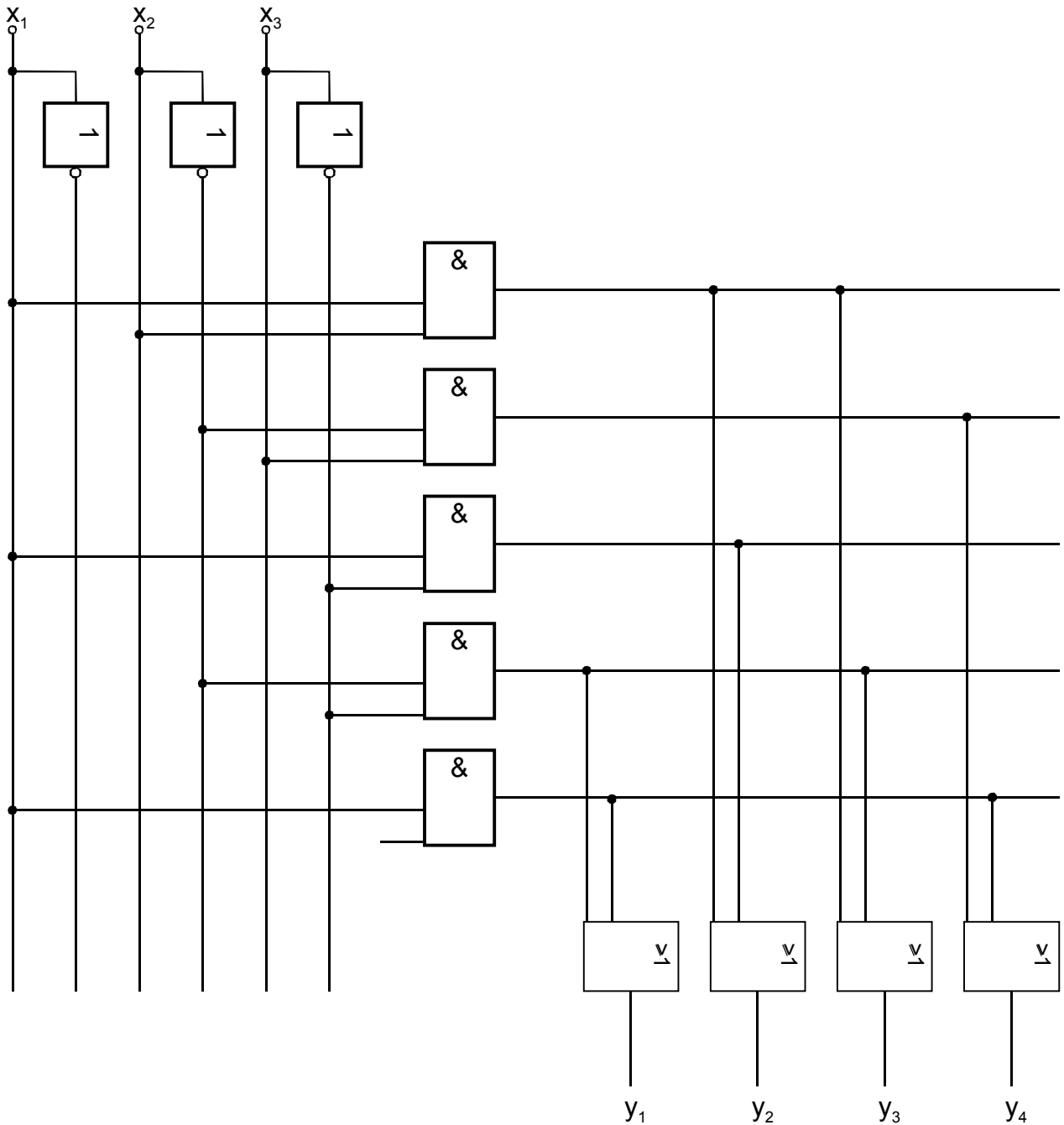
1 - originalna spremenljivka

0 - komplementirana spremenljivka

- spremenljivka ne nastopa v konjunkciji

1 - konjunkcijo si deli več komponent

Sprogramirana mreža



Kadar nam elementarna minimizacija ne zadošča, lahko uporabimo kodiranje

Z njim občutno zmanjšamo potrebno število AND elementov PLA vezja

Pri tem pristopu v AND polju povezujemo konjunkcije namesto posamičnih spremenljivk

Neodvisne spremenljivke pri tem razdelimo v skupine npr. po dve

Komponente funkcije je potrebno faktorizirati, da lahko določimo skupine neodvisnih spremenljivk

$$y_1 = x_1x_3 + x_1x_2x_4 + x_2x_3x_4$$
$$y_2 = \overline{x}_1\overline{x}_2x_3 + x_1\overline{x}_2\overline{x}_3 + \overline{x}_1x_2\overline{x}_3 + x_1\overline{x}_3\overline{x}_4 + \overline{x}_1x_2\overline{x}_3x_4 + x_1x_2x_3x_4$$
$$y_3 = x_2\overline{x}_4 + \overline{x}_2x_4$$

Faktorizacija prinese:

$$y_1 = x_1x_3 + (x_1 + x_3)x_2x_4$$
$$y_2 = (x_1\overline{x}_3 + \overline{x}_1x_3)(\overline{x}_2 + \overline{x}_4) + (\overline{x}_1\overline{x}_3 + x_1x_3)x_2x_4 =$$
$$= (x_1 + \overline{x}_3)(x_1 + x_3)(\overline{x}_2 + \overline{x}_4) + (\overline{x}_1 + x_3)(x_1 + \overline{x}_3)x_2x_4$$
$$y_3 = (x_2 + x_4) + (\overline{x}_2 + \overline{x}_4)$$

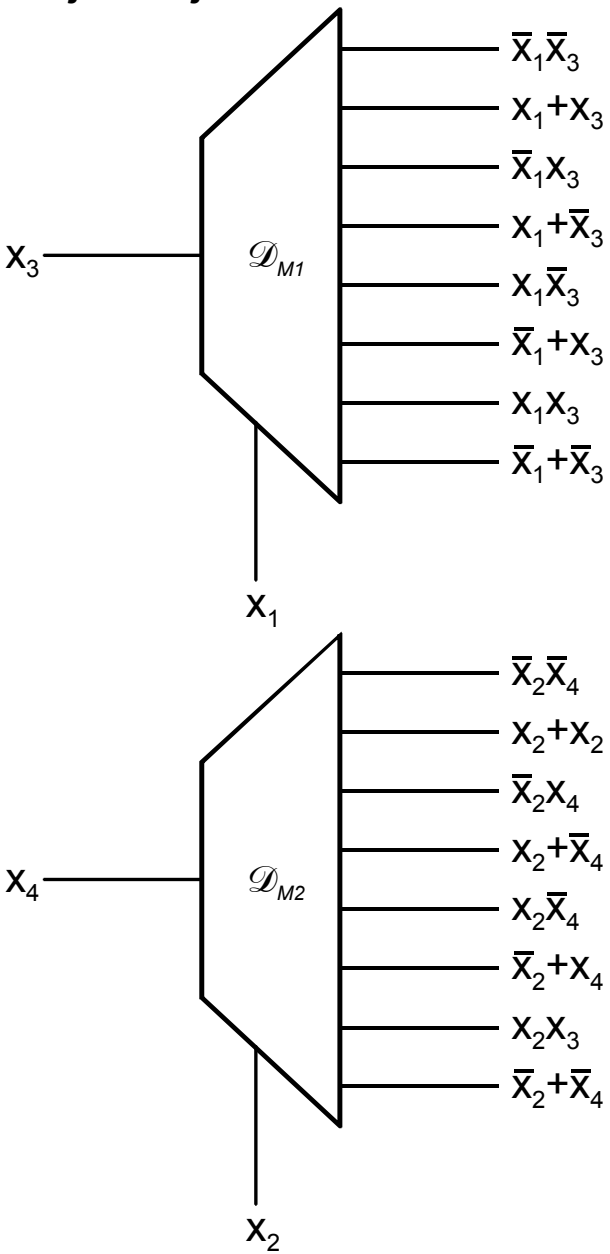
Zlahka uvidimo, da je potrebna delitev v skupini x_1x_3 in x_2x_4

Namesto enajst AND elementov, kolikor bi jih potrebovali pri elementarni minimizaciji, jih sedaj potrebujemo samo še pet

Za dekodiranje lahko uporabimo standarden demultipleksor, ki ima poleg originalnih tudi negirane izhode

Takšno izvedbo pogosto srečamo tako pri demultipleksorju, kot pri multipleksorju, ker se s tem izdatno poveča funkcijska moč teh mnogopolov

Za realizacijo te funkcije potrebujemo PLA vezje z najmanj tremi vhodi, petimi konjunkcijami in tremi izhodi



Metodo je mogoče še razširiti, če uporabimo dekodirnike, ki tvorijo samo maksterme

V neki skalarni funkciji $f(x_1,x_2,x_3,x_4)$ lahko nastopa na primer skupina spremenljivk x_1,x_2

Makstermi teh dveh spremenljivk so:

$$(x_1+x_2),(\bar{x}_1+x_2),(x_1+\bar{x}_2),(\bar{x}_1+\bar{x}_2)$$

Če so dekodirniki že vgrajeni v PLA vezje, kot je to primer pri VLSI izvedbah imenujemo takšno mrežo OR-AND-OR mreža, ki v splošnem prinese zmanjšanje celotnega PLA vezja

Pri realizaciji s PAL vezji nas omejuje še število konjunkcij v posamezni komponeti funkcije

To število je praviloma manjše od števila AND elementov.

Posamezne izvedbe teh vezij se po kompleksnosti lahko zelo razlikujejo med seboj

Tako imajo na primer PAL vezja s šestnajstimi vhodi in šestnajstimi konjunkcijaami štiri OR elemente s po štirimi vhodi ali dva OR elementa s po osmimi vhodi

Od tu izvira tudi glavna razlika glede realizacije funkcij s PAL vezji v primerjavi s PLA

Medtem, ko je pri PLA mogoče izkoristiti skupne konjunkcije, tu tega ni mogoče

Tipičen primer uporabe PAL vezja je na primer primerjalnik in naj bo le 2-biten

Posamezne komponente njegove vektorske preklopne funkcije so:

$$A = B : y_1 = \overline{A}_1\overline{A}_0\overline{B}_1\overline{B}_0 + \overline{A}_1A_0\overline{B}_1B_0 + A_1\overline{A}_0B_1\overline{B}_0 + A_1A_0B_1B_0$$
$$A \neq B : y_2 = A_1\overline{B}_1 + \overline{A}_1B_1 + \overline{A}_0B_0 + A_0\overline{B}_0$$
$$A < B : y_3 = \overline{A}_1B_1 + \overline{A}_1\overline{A}_0B_0 + \overline{A}_0B_1B_0$$
$$A > B : y_4 = A_1\overline{B}_1 + A_1A_0\overline{B}_0 + A_0\overline{B}_1\overline{B}_0$$

Te komponente vektorske funkcije so vse v minimalni disjunktivni normalni obliki

V njih sicer nastopajo skupne konjunkcije ($A_1\ \overline{B}_1$ in $\overline{A}_1B_1$), vendar tega ne moremo izkoristiti

Primerno PAL vezje bo torej edino:

4 x 16 x 4

Ustrezno sprogramirano vezje vidimo na naslednji sliki