2. Logické systémy

Logickými systémy rozumíme takové systém s diskrétním paramerem, u nichž vstupní proměnné nabývají pouze hodnot 0,1. V některých literárních pramenech se také těmto systémům říká dvouhodnotové nebo boolské.

Logické systémy charkterizuje:

1. množina logických hodnot $\{ 0,1 \}$
2. množina všech vstupních vektorů $\mathbf{X} = \{ (x_1,x_2...x_n) \mid x_i \in \{0,1\}\}$ při n vstupech
   vyjádřeno kartézským součinem $\mathbf{X} = \{0,1\} \times \{0,1\} \times ..... \times \{0,1\} = \{0,1\}^n$

$\{0,1\}^n$ 2^n - uspořádaných n-tic pro n - vstupů

Př: $A=\{0,1\}^3= \{(000),(001),(010),(011),(100),(101),(110),(111)\}$

3. množina všech výstupních vektorů $\mathbf{Z} = \{0,1\}^m$ při m - výstupech
4. množina všech vnitřních (stavových) vektorů
   $\mathbf{S} = \{0,1\}^k$ při k - vnitřních proměnných

V některých literárních pramenech se používá odlišná terminologie:

množina všech vektorů	- abeceda
vektor	- písmeno
sekvence vektorů	- slovo

<u>Čas</u>:
Logický systém je dynamický a pracuje v čase. Často u těchto systémů používáme <u>diskrétní čas</u> t.j. pouze určité významné časové okamžiky - změny. Takovýmto systémům říkáme <u>synchronní systémy.</u> Změny jsou akceptovány pouze při náběžných (resp. spádových) hranách synchronizačního signálu.

<u>Struktura systému</u>
Logický systém je tvořn kompozicí ze vzájemně propojených jednodušších systémů. Tuto kompozici nazýváme <u>strukturou systému</u>.

Je definována dvojicí **(P,V)**

kde $\mathbf{P} = \{ P_1...P_n\}$ - množina prvků struktury
$\mathbf{V} = \{ V_1...V_s\}$ - množina vazeb mezi prvky

Struktura se vyjadřuje orientovaným grafem: uzel - prvek, hrana - vazba

Oblastí našeho sledování je <u>konečný automat</u> s vlastnostmi:

1. množiny vstupních, výstupních i stavových vektorů jsou konečné
2. časová množina je zobrazitelná na množinu celých čísel (časově diskrétní systém)
3. stavově přechodová funkce a výstupní funkce jsou stacionární

Dělení logických systémů

Při dělení logických systémů nás zajímá model, u kterého se zabýváme pouze jeho vnějším dynamickým chováním. Podle chování dělíme logické systémy na:

Kombinační - jeho odezva v určitém časovém okamžiku je podmíněna výhradně okamžitými hodnotami na vstupech kombinačního systému.

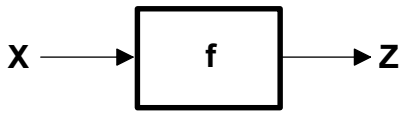
chování lze popsat funkcí : $\mathbf{f} : \mathbf{X} \rightarrow \mathbf{Z}$

kde **X** - množina vstupních vektorů
Z - množina výstupních vektorů

Tato funkce přiřazuje vektoru $\mathbf{x} \in \mathbf{X}$ určitý výstupní vektor $\mathbf{z} = \mathbf{f}(\mathbf{x})$, kde $\mathbf{z} \in \mathbf{Z}$. Tomuto přiřazení se také říká kombinační zobrazení, které přiřadí každému vektoru **x** pouze jeden vektor **z** a má vlastnosti funkce.

Zobrazení **f** můžeme rozložit na m dílčích zobrazení (funkcí), které definují hodnoty jednotlivých výstupních proměnných:

$$\begin{aligned} \mathbf{z}_1 &= \mathbf{f}_1(\mathbf{x}) \\ \mathbf{z}_2 &= \mathbf{f}_2(\mathbf{x}) \qquad \text{kde } \mathbf{x} \in \mathbf{X} \\ &\vdots \\ \mathbf{z}_m &= \mathbf{f}_m(\mathbf{x}) \end{aligned}$$



Obr.2.1. Kombinační systém

Oborem hodnot funkcí $\mathbf{f}_1 \ldots\ldots \mathbf{f}_m$ je množina $\{0,1\}$. Funkce se označují pojmem logická funkce.

Sekvenční - výstupní vektor závisí nejen na vstupním vektoru ale i na sekvenci vstupních vektorů v předcházejících bodech diskrétního času popř. na časových intervalech změn vstupních vektorů.V závislosti na předcházejících vstupních vektorech může sekvenční systém při určitém vstupním vektoru generovat různé výstupní vektory. To znamená, že systém si musí pamatovat předešlé situace a musí mít tedy **paměť** (výstup závisí na historii systému)

chování lze popsat funkcí : $\mathbf{F}_S : \mathbf{X}^* \rightarrow \mathbf{Z}^*$

kde $\mathbf{X}^*$ a $\mathbf{Z}^*$ jsou nekonečné množiny všech posloupností s konečnou délkou sestavené ze vstupních vektorů množiny **X** resp. výstupních vektorů množiny **Z** . Posloupnosti $\mathbf{X}^*$ a $\mathbf{Z}^*$ nazýváme vstupní a výstupní slovo.

Funkce $\mathbf{F}_s$ každému vstupnímu slovu $\mathbf{X}^*$ přiřazuje výstupní slovo $\mathbf{Z}^*$ se stejnou délkou. Vstupní slovo $\mathbf{X}^*$ je posloupnost vektorů **x** od časového okamžiku t = 0 při kterém se systém nachází ve stavu **s.**

Funkce $\mathbf{F}_s$ je funkce sekvenčního systému (sekvenční zobrazení). Toto zobrazení má nekonečný obor hodnot a lze je vyjádřit pomocí dvou funkcí s konečným oborem hodnot.

$$\begin{aligned} \mathbf{s}(t+1) &= \mathbf{f}_s(\mathbf{s}(t), \mathbf{x}(t)) && \text{funkce následného stavu } \mathbf{f}_s \\ \mathbf{z}(t) &= \mathbf{f}_z(\mathbf{s}(t), \mathbf{x}(t)) && \text{funkce současného výstupu } \mathbf{f}_z \end{aligned}$$

kde **x** (t), **z** (t), **s** (t) - vstupní, výstupní, stavový vektor systému
s (t+1) - následný stav

Stav systému vyjadřuje jeho paměť neboli informaci o tom, co se odehrálo v minulosti (historie).

Př.: Vstup je reprezentován jako sekvence vektorů $\mathbf{x}_1 ... \mathbf{x}_k$, kde $\mathbf{x}_1$ je první a $\mathbf{x}_k$ je k-tý vektor. Stav systému je v i-tém kroku dán vektorem $\mathbf{s}_i$ a může nabývat libovolnou hodnotu z množiny stavů **S**. Protože následný stav závisí na předešlém, lze psát

$$\mathbf{s}_{i+1} = \mathbf{f}_s (\mathbf{x}_i, \mathbf{s}_i)$$

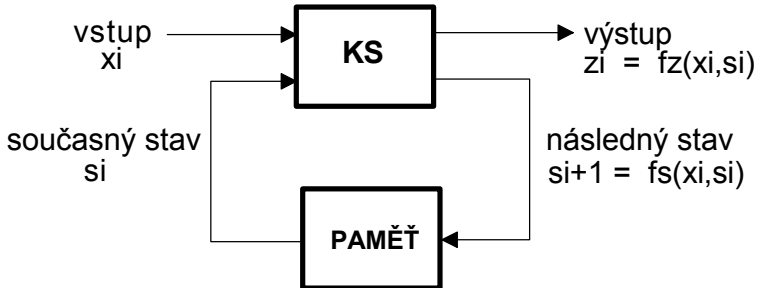
kde $\mathbf{f}_s$ je funkce následného stavu systému.

Výstup systému je reprezentován sekvencí vektorů $\mathbf{z}_i$, které jsou prvky množiny výstupních vektorů **Z** a výstup v i-tém kroku je dán

$$\mathbf{z}_i = \mathbf{f}_z (\mathbf{x}_i, \mathbf{s}_i)$$

kde $\mathbf{f}_z$ je výstupní funkce systému.

Z uvedého vyplývá blokové schema obecného logického (sekvenčního) systému



Obr.2.2. Sekvenční systém

Opět lze zobrazení $\mathbf{f}_s$ a $\mathbf{f}_z$ rozložit na **m** jednotlivých zobrazení (funkcí), které definují hodnoty jednotlivých výstupních proměnných a **n** jednotlivých zobrazení (funkcí), které určují následné stavy systému tentokráte ale pro určitý časový okamžik i.

$$\begin{array}{ll} \mathbf{z}_1^{(i)} = \mathbf{f}_{z1} (\mathbf{x}^{(i)}, \mathbf{s}^{(i)}) & \mathbf{s}_1^{(i+1)} = \mathbf{f}_{s1} (\mathbf{x}^{(i)}, \mathbf{s}^{(i)}) \\ \mathbf{z}_2^{(i)} = \mathbf{f}_{z2} (\mathbf{x}^{(i)}, \mathbf{s}^{(i)}) & \mathbf{s}_2^{(i+1)} = \mathbf{f}_{s2} (\mathbf{x}^{(i)}, \mathbf{s}^{(i)}) \\ \vdots & \vdots \\ \mathbf{z}_m^{(i)} = \mathbf{f}_{zm} (\mathbf{x}^{(i)}, \mathbf{s}^{(i)}) & \mathbf{s}_n^{(i+1)} = \mathbf{f}_{sn} (\mathbf{x}^{(i)}, \mathbf{s}^{(i)}) \end{array}$$

kde $\mathbf{x} \in \mathbf{X}$ *a* $\mathbf{s} \in \mathbf{S}$

Je samozřejmé, že pro f_s = const se ze sekvenčního automatu stává automat kombinačního typu (množina stavů je prázdná).

<u>Základní definice pojmů z oboru logických systémů</u>

<u>Logická proměnná</u> - proměnná logického systému, která nabývá pouze hodnot 0 nebo 1
<u>Logický signál</u> - druh fyzikálního signálu, který nabývá pouze hodnot patřících do vzájemně se nepřekrývajících množin
<u>Logická funkce</u> - vztah závislých a nezávislých logických proměnných.
<u>Logický člen</u> - zařízení realizující logické funkce
<u>Logický systém</u> - model logického obvodu, který matematickým popisem charakterizuje chování logického obvodu

Logický obvod - druh fyzikálního systému, u něhož každá veličina nabývá v ustáleném stavu pouze dvou hodnot z dvou vzájemně se nepřekrývajících množin

Logický soubor - soubor logických členů a pomocných zařízení pro realizaci logických obvodů
Každý ucelený soubor se skládá se z členů umožňujících informaci:

získat	snimače, převodníky, ovládací prvky
zpracovat	funkční část relizující funkční předpis
přenést	kodéry, dekodéry, přenosové kanály
využít	akční členy, výkonové prvky

Logické systémy (obvody) s pevnou funkcí - navržená struktura je pevná pro řešení určité úlohy. Funkci obvodu jednoznačně určuje jeho struktura (zapojení). Změna funkce znamená změnu struktury.

Logické systémy (obvody) s programovanou funkcí -funkce je určena programem uloženým v paměti. Změna funkce obvodu podmiňuje změnu programu. Jednotlivé operace (funkční předpis) odpovídají jednotlivým instrukcím.

2.1. Logická funkce a její vyjádření

Logická funkce je přiřazení mezi závislými logickými proměnnými (výstupy) a nezávislými logickými proměnnými (vstupy). Obecně řečeno je to funkční předpis pro kombinační zobrazení množiny vstupních vektorů na množinu výstupních vektorů.

$$\mathbf{f} : \mathbf{X} \rightarrow \mathbf{Z}$$

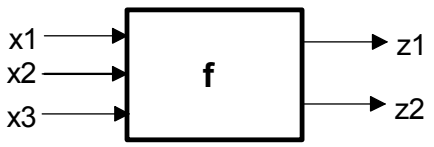
nebo $$\mathbf{z} = \mathbf{f}\ (\mathbf{x})$$

Pro logický obvod s **n** vstupy a **m** výstupy má množina vstupních vektorů **X** 2^n prvků (vektorů) a množina výstupních vektorů **Z** 2^m prvků (vektorů). Maximální možný počet zobrazení (funkcí) pro takto obecně definovaný logický obvod je

$$(2^n)^{2^m}$$

t.j. každému výstupnímu vektoru $\mathbf{z} \in \mathbf{Z}$ (je jich 2^m) je nutné přiřadit všechny vstupní vektory $\mathbf{x} \in \mathbf{X}$ (je jich 2^n).

Př.: příklad funkce:



X = {(000),(001),(010),(011),(100),(101),(111)}
Y = {(00),(01),(10),(11)}

x_3	x_2	x_1	Z_2	Z_1
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	1	1
1	0	1	1	0
1	1	0	0	1
1	1	1	0	0

Celkem tento obvod může realizovat 4096 funkcí.

Základní vyjádření logické funkce je pravdivostní tabulka. Kombinační zobrazení předešlého příkladu můžeme rozepsat na dvě boolské funkce:

$$z_1 = f^{(1)}\ (x_1 x_2 x_3)$$
$$z_2 = f^{(2)}\ (x_1 x_2 x_3)$$

Boolská funkce

Boolská funkce n - proměnných je logická funkce realizující zobrazení:

$$\mathbf{f} : \mathbf{X} \rightarrow \{\mathbf{0}, \mathbf{1}\}$$

kde **X** - je množina všech vstupních vektorů n-proměnných $\mathbf{X} = \{0,1\}^n$
f - je množina všech boolských funkcí

množinu funkcí **f** lze rozložit na jednotlivé funkce

$$\mathbf{f}_j : \mathbf{X} \rightarrow \{\mathbf{0}, \mathbf{1}\} \qquad \textit{pro} \quad j = 1,2,...m$$

Maximální možný počet těchto zobrazení je 2^{2^n} pro n-vstupních proměnných t.j množina **X** má 2^n prvků a množina **Z** pouze 2 prvky.

Př.: Zjistit, počet boolských funkcí 2 vstupních proměnných

X = {(00),(01),(10),(11)}
Z = {0,1}

Celkem obvod může realizovat 16 boolských funkcí viz následující tabulka:

x_1 x_2	0 1 0 1 0 0 1 1	Popis		Logický výraz
f_0	0 0 0 0	nulová funkce		$Z = 0$
f_1	0 0 0 1	logický součin	$x_1 \wedge x_2$	$Z = x_1 x_2$
f_2	0 0 1 0	inhibice	$x_2 \not\rightarrow x_1$	$Z = x_2 \bar{x}_1$
f_3	0 0 1 1	kopie x_2		$Z = x_2$
f_4	0 1 0 0	inhibice	$x_1 \not\rightarrow x_2$	$Z = \bar{x}_2 x_1$
f_5	0 1 0 1	kopie x_1		$Z = x_1$
f_6	0 1 1 0	nonekvivalence	$x_1 \neq x_2$	$Z = x_1\bar{x}_2 + \bar{x}_1 x_2$
f_7	0 1 1 1	logický součet	$x_1 \vee x_2$	$Z = x_1 + x_2$
f_8	1 0 0 0	Pierceova funkce	$x_1 \downarrow x_2$	$Z = \overline{x_1 + x_2}$
f_9	1 0 0 1	ekvivalence	$x_1 \equiv x_2$	$Z = x_1x_2 + \bar{x}_1\bar{x}_2$
f_{10}	1 0 1 0	negace x_1		$Z = \bar{x}_1$
f_{11}	1 0 1 1	implikace	$x_1 \rightarrow x_2$	$Z = \bar{x}_1 + x_2$
f_{12}	1 1 0 0	negace x_2		$Z = \bar{x}_2$
f_{13}	1 1 0 1	implikace	$x_2 \rightarrow x_1$	$Z = x_1 + \bar{x}_2$
f_{14}	1 1 1 0	Schefferova funkce	$x_1 \mid x_2$	$Z = \overline{x_1 x_2}$
f_{15}	1 1 1 1	jednotková funkce		$Z = 1$

Tab.2.1. Funkce dvou vstupních proměnných

Logická (boolská) funkce může být <u>úplně definovaná</u> - je jednoznačně definovaná výstupní hodnota pro každý vstupní vektor. V opačném případě mluvíme o <u>neúplně definované</u> funkci

(pro některé vstupní vektory není výstup definován). Celkový počet funkcí pro neúplně definovanou funkci n- proměnných je

$$3^{2^n} \qquad \mathbf{f} : \mathbf{X} \rightarrow \{\mathbf{0}, \mathbf{1}, \mathbf{x}\}$$

Př.: Dekodér dekadických čísel (od 10 do 15 jsou x - mimokódové kombinace)

Logický výraz

Logický výraz je analytickým vyjádření logické funkce. Její hodnoty získáme vyhodnocením logického výrazu pro jednotlivé vstupní vektory. Logický výraz může být tedy modelem chování (vyhodnocení výrazu) i modelem struktury (lze podle něj pomocí souboru realizačních prvků realizovat logický obvod se stejným chováním).

2.1.1. Boolská algebra

Boolská algebra je prostředkem pro popis chování logických systémů a transformace logických výrazů.

Def.: Boolská algebra je definována jako konečná množina prvků obsahující:

1. logické proměnné a, b, c,...
2. dvě binární operace $\vee$ (+), $\wedge$ (.)
3. unární operaci inverse (negace) ($\bar{\ }$) - přiřazuje doplněk
4. dva logické stavy 0,1 označované jako logické konstanty

Této množině říkáme svaz, který je distributivní (má jednoznačný doplněk) a komplementární (každý z prvků má jediný doplněk). Každý zákon má svoji duální formu (doplňková operace, doplňkový prvek).

Axiomy :

$1 . 1 = 1$	$0 + 0 = 0$
$1 . 0 = 0 . 1 = 0$	$0 + 1 = 1 + 0 = 1$
$0 . 0 = 0$	$1 + 1 = 1$
$\bar{1} = 0$	$\bar{0} = 1$

Zákony :

$a + a = a$	$a . a = a$	tautologie
$a + b = b + a$	$a . b = b . a$	komutativní zákon
$(a + b) + c = a + (b + c)$	$(a . b) . c = a . (b . c)$	asociativní zákon
$a + (a . b) = a$	$a . (a + b) = a$	zákon absorbce
$0 . a = 0$	$1 + a = 1$	agresivnost 0 a 1
$0 + a = a$	$1 . a = a$	neutrálnost 0 a 1
$a . (b + c) = ab + ac$	$a + (b . c) = (a + b) . (a + c)$	distributivní zákon
$a . \bar{a} = 0$	$a + \bar{a} = 1$	zákon vyloučeného třetího

Další zákony : (odvozené)

$\bar{\bar{a}} = a$		zákon dvojí negace
$a + \bar{a}b = a + b$	$a . (\bar{a} + b) = ab$	zákon absorbce negace
$\overline{a + b} = \bar{a} . \bar{b}$	$\overline{a . b} = \bar{a} + \bar{b}$	De Morganovy zákony

Zobecněné zákony

zákony o absorbci

$a . f (a, \bar{a}, b,...) = a . f (1, 0, b,...)$ $\qquad a + f (a, \bar{a}, b,...) = a + f (0, 1, b,...)$

$\bar{a} . f (a, \bar{a}, b,...) = a . f (0, 1, b,...)$ $\qquad \bar{a} + f (a, \bar{a}, b,...) = a + f (1, 0, b,...)$

De Morganovy zákony

$\overline{a . b . c .....} = \bar{a} + \bar{b} + \bar{c} + ...$ $\qquad \overline{a + b + c + ...} = \bar{a} . \bar{b} . \bar{c} .....$

Shannonův teorém o rozkladu

$$f(a, \bar{a}, b,...) = a \,.\, f(1, 0, b,...) + \bar{a} \,.\, f(0, 1, b,...)$$
$$f(a, \bar{a}, b,...) = [\, a + f(0, 1, b,...)\,] \,.\, \bar{a} + f(0, 1, b,...)\,]$$

<u>Princip duality</u>: Každé rovnosti výrazů přísluší rovnost duálních výrazů podle transformace

$0 \rightarrow 1$ $\quad + \rightarrow .$ $\quad a \rightarrow \bar{a}$

$1 \rightarrow 0$ $\quad . \rightarrow +$ $\quad \bar{a} \rightarrow a$

Př.:

$$y = \bar{a} + \bar{b} \quad \rightarrow \quad \bar{y} = \bar{a} \,.\, \bar{b}$$

2.1.2. Formy popisu logické funkce

<u>1. Pravdivostní tabulka</u>

x_3	x_2	x_1	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Pravdivostní tabulka je nejpřirozenější forma zápisu logické funkce. Obsahuje výčet všech vstupních vektorů a jim odpovídajících výstupů.
Je to tedy 2^n řádků a n sloupců pro n vstupních proměnných a m sloupců pro m výstupních proměnných.

Pravdivostní tabulka tvoří <u>model chování</u> .

<u>2. Logický výraz</u>

Logický výraz je analytickým vyjádřením logické funkce (algebraická forma). Tvoří <u>model chování</u> (vyhodnocení výrazu) i <u>model struktury</u> (možnost realizace souborem logických prvků).

Nejdříve se věnujme definicím některých používaných termínů.

Term je člen logického výrazu neboli boolský výraz. Rozeznáváme term :

- <u>součinový</u> - boolský výraz, který neobsahuje operátor součtu (+)
  př.: $x_1 \bar{x}_2 x_3$
  pro n proměnných 3^n-1 termů
- <u>součtový</u> - boolský výraz, který neobsahuje operátor součinu (.)
  př.: $x_1 + \bar{x}_2 + x_3$
  pro n proměnných 3^n-1 termů

Minterm je součinový term, který obsahuje všechny proměnné v přímé nebo negované formě
Př.: $x_1 \bar{x}_2 x_3 x_4 \bar{x}_5 .... \bar{x}_n$

Maxterm je součtový term, který obsahuje všechny proměnné v přímé nebo negované formě
Př.: $\bar{x}_1 + x_2 + \bar{x}_3 + \bar{x}_4 + x_5 + ... + x_n$

Každý minterm nebo maxterm nabývá logické hodnoty 1 nebo 0 právě pro jeden vstupní vektor.

Každou úplnou logickou funkci můžeme vyjádřit výrazem ve tvaru úplné <u>normální disjunktivní</u> nebo úplné <u>normální konjunktivní formy</u>.

disjunktivní $f(x_1, x_2, \ldots x_n) = \sum_{i=0}^{2^n-1} f(x_i).\ Mi(x_i)$

konjunktivní $f(x_1, x_2, \ldots x_n) = \prod_{i=0}^{2^n-1} [f(x_i) + Ma(x_i)]$

kde $x_1 \ldots x_n$ - vstupní proměnné
$f(x_i)$ - hodnota logické funkce f pro vstupní vektor $\mathbf{x}_i$
$Mi(x_i)$ - minterm nabývající hodnotu 1 pro vstupní vektor $\mathbf{x}_i$
$Ma(x_i)$ - maxterm nabývající hodnotu 0 pro vstupní vektor $\mathbf{x}_i$

platí $Ma(x_i) = \overline{Mi}(x_i)$

Př.: $Mi(x_i) = \bar{x}_1\, x_2\, \bar{x}_3$ $\quad Ma(x_i) = \overline{\bar{x}_1\, x_2\, \bar{x}_3} = (x_1 + \bar{x}_2 + x_3)$

disjunktivní forma $f = \bar{x}_1\, x_2\, \bar{x}_3 + x_1 x_2\, \bar{x}_3 + x_1\, \bar{x}_2 x_3$
konjuktivní forma $f = (x_1 + x_2 + x_3)(\bar{x}_1 + x_2 + x_3)(x_1 + x_2 + \bar{x}_3)(x_1 + \bar{x}_2 + \bar{x}_3)$

Z předešlého platí, že: součet všech mintermů = 1
součin všech maxtermů = 0

V praxi většinou existují formy <u>smíšené</u> (t.j. kombinace konjuktivní i disjunktivní formy).

Z předešlého vyplývá, že boolský výraz, který obsahuje operace součtu, součinu a negace může být modelem každé logické funkce.

<u>Úplný soubor logických funkcí</u>
je takový soubor, pomocí něhož můžeme realizovat libovolnou logickou funkci
př.- soubor obsahuje všechny logické funkce

<u>Minimální a úplný soubor logických funkcí</u>
je takový soubor , z něhož žádnou logickou funkci nelze vypustit nebo nahradit kombinací ostatních
př.: NAND, NOR, implikace a negace, inhibice a negace aj.

<u>3. Seznam indexů vstupních vektorů</u>

Index je definován jako dekadická hodnota binárně zakódovaného vstupního písmene

$$I = \sum_{i=0}^{n-1} 2^i x_i$$

kde 2^i - váha vstupní proměnné i
x_i - logická proměnná

<u>Seznam indexů</u> vstupních písmen představuje zjednodušený zápis pravdivostní tabulky. Uvádí seznam indexů, pro něž logická funkce:

nabývá hodnoty **1** $f(x_1 x_2 x_3) = \Sigma\,(2,3,5,7)$
nebo hodnoty **0** $f(x_1 x_2 x_3) = \Pi\,(0,1,4,6)$

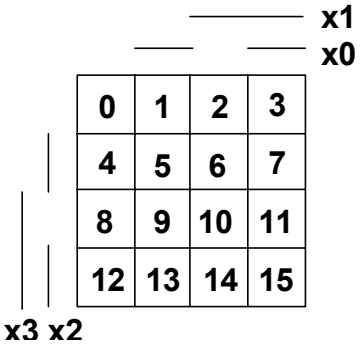
Pro neúplné funkce doplňujeme seznam o vstupní písmena, pro něž není logická funkce definována

pro disjunktivní formu $+ \Sigma_x$ (seznam)
pro konjunktivní formu $. \Pi_x$ (seznam)

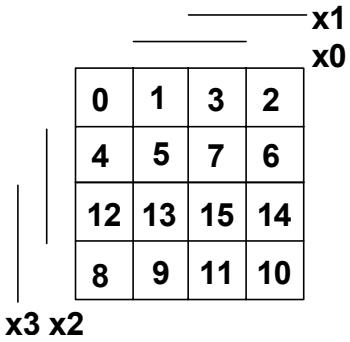
4. Mapa logické funkce

Vznik : proměnné logickéfunkce rozdělíme na dvě skupiny, přičemž každé kombinaci jedné skupiny přiřadíme sloupec a každé kombinaci druhé skupiny přiřadíme řádek. Průnik řádků a slopuců (políčko) odpovídá kombinaci všech proměnných (minterm - vektor) a odpovídá jednomu řádku v pravdivostní tabulce. Do čtverečku vpisujeme hodnotu logické funkce pro dané vstupní písmeno (vektor). Záznamy jsou 1, 0, x .

Podle kódování proměnných (označení mapy) dělíme mapy na :



Svobodova (binární) mapa

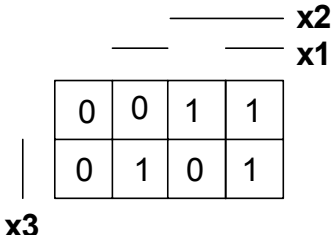


Karnaughova (gray-kód) mapa

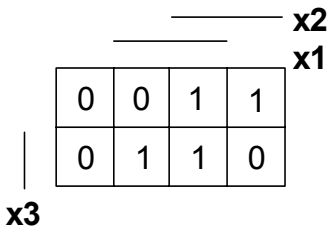
Do políček mapy jsou vepsány indexy vstupních písmen.

Vlastnosti: Karnaughova mapa zajišťuje vzájemnou "sousednost " políček - mezi sousedními políčky mapy je změna pouze v jedné proměnné (totéž platí i o vnějších hranicích mapy - mapa je jako balón). Tento způsob vyjádření logické funkce budeme nadále používat jako základní Svobodova mapa zjednodušuje porovnání jednotlivých písmen vstupního vektoru (jeden pro řádek, druhý pro sloupec) kódovaných v binárním kódu - komparace binárních čísel.

Př. Pro původní logickou funkci ze stavové tabulky jsou obě mapy následující:



Svobodova mapa

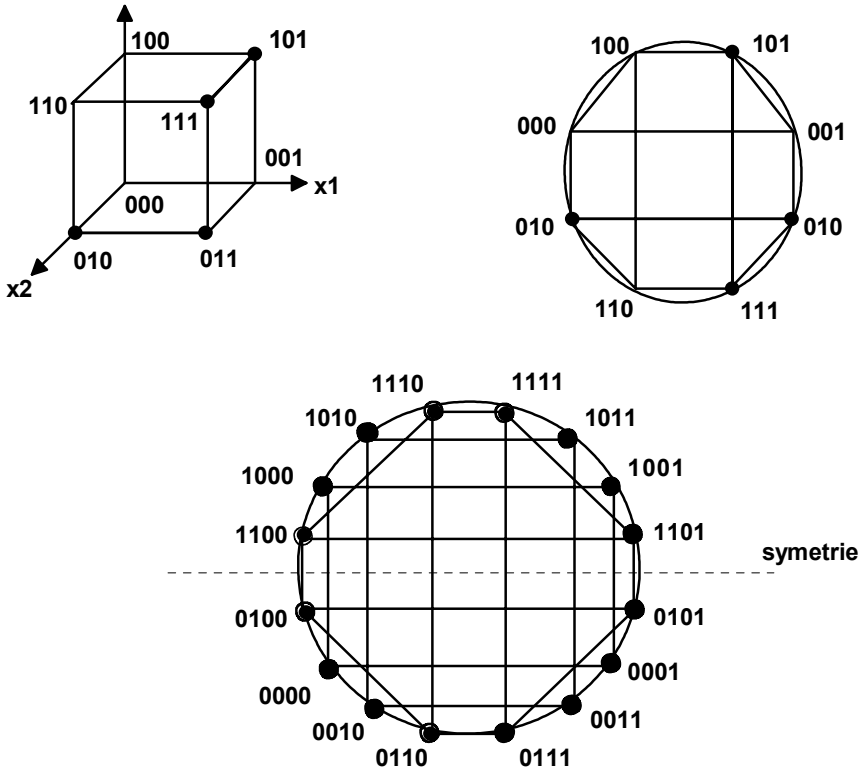


Karnaughova mapa

5. Vícerozměrná krychle

Další variantou popisu logické funkce je vícerozměrná krychle. Pro n-proměnných je to n-rozměrná krychle, která vznikne vytvořením 2^n vektorů, přičemž souřadnice jsou dány hodnotami vstupních proměnných. Koncové body těchto vektorů leží na vrcholech jednotkové n-rozměrné krychle.

přiřazení logické funkce - vyplněný vrchol - funkce je jednotková
- prázdný vrchol - funkce je nulová



Obr.2.3. Vícerozměrná krychle pro 3 a 4 proměnné v prostorové a ploché formě

Toto vyjádření je poměrně složité a používá se zejména pro demonstraci bezpečnostních kódů (kódová vzdálenost mezi obsazenými vrcholy).

2.1.3. Minimalizace logických výrazů

Účelem minimalizace je minimalizovat logický výraz jako model struktury s cílem získat co nejjednodušší výraz pro původní logickou funkci a tím i tvar logickéfunkce pro minimální realizační strukturu.

Kriteria - 1. minimální počet operací (počet logických členů)
2. minimální počet proměnných (počet vstupů logických členů)
3. kombinace obou předešlých kriterií pro určitý soubor realizačních prvků

V zásadě se používá kriterium 1. s přihlédnutím ke kriteriu 2. pro určitý soubor realizačních prvků.

Nejprve je třeba se věnovat definicím některých pojmů:

Implikant boolské funkce f - je výraz ve tvaru součinového termu, který definuje funkci f_1 pro níž platí $f_1 \rightarrow f$ (implikace - je součástí, pokrývá), funkce se skládá z implikantů.

Přímý implikant boolské funkce f (primterm) - je každý implikant boolské funkce f, pro který neexistuje $f_1 \rightarrow f_1' \rightarrow f$ t.j. tento implikant se nedá zjednodušit

Př.: $f = a\,\bar{c}\;b + a\,\bar{c} + c\,\bar{b}$

implikant $(a\bar{c}\,b) \rightarrow f = 1$ není přímý implikant
protože platí $(a\bar{c}\,b) \rightarrow (a\bar{c}) \rightarrow f = 1$
t.j. implikant $(a\bar{c})$ je přímý implikant

Minimalizační metody

Standardní postup při minimalizaci je následující:

1. nalezení množiny přímých implikantů
2. výběr minimálního počtu implikantů nutného k pokrytí logické funkce f

<u>Principem</u> je vyhledání dvou implikantů, které se liší pouze v jedné proměnné.

$$\left.\begin{array}{l} a\bar{b}c \\ a\bar{b}\bar{c} \end{array}\right\rangle\; a\bar{b}$$

a nalezení přímého implikantu.

<u>1. Minimalizace výrazů s využitím Booleovy algebry</u>

Metoda vychází z normálních forem analytického vyjádření logické funkce a používá úpravu podle zákonů Booleovy algebry.

Př.:

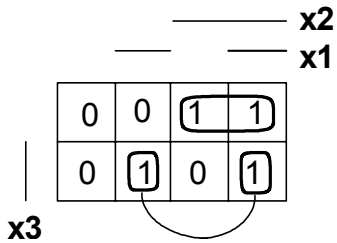
$$\begin{aligned} f(x_1x_2x_3) &= \bar{x}_1x_2\bar{x}_3 + x_1x_2\bar{x}_3 + x_1\bar{x}_2x_3 + x_1x_2x_3 \\ &= x_2x_3(\bar{x}_1 + x_1) + x_1x_3(\bar{x}_2 + x_2) \\ &= x_2\bar{x}_3 + x_1x_3 \end{aligned}$$

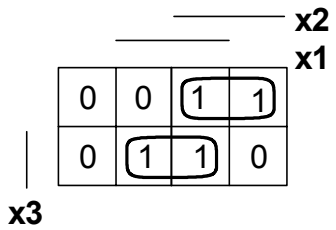
Další úpravou je <u>vytýkání</u> před závorku, kterou můžeme získat <u>absolutně minimální formu</u> logické funkce. Výsledkem je smíšený tvar obsahující závorky, který má ale nejmenší počet operací i proměnných.
Většinou pracujeme s disjunktivní formou logické funkce. Minimální konjunktivní tvar logické funkce dostaneme tak, že vytvoříme minimální disjunktivní tvar <u>negace</u> dané funkce a výsledek znegujeme.

<u>2. Minimalizace výrazů s využitím map</u>

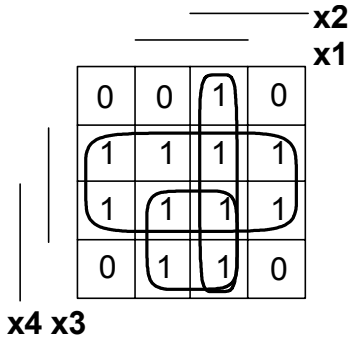
Jedno políčko v mapě představuje minterm. Vzájemná poloha jedniček v mapě nese informaci o možném výskytu sousedních implikantů. Snažíme se sestavit množinu implikantů, které představují přímý implikant. Disjunktivní tvar těchto přímých implikantů tvoří minimalizovaný výraz logické funkce.
<u>Hledáme minimalizační smyčku</u> t.j. takovou oblast mapy, která obsahuje 2^K políček, přičemž ke každému políčku přísluší v této smyčce. K sousedních políček (sousední mintermy). Snažíme se získat co nejmenší počet co největších minimalizačních smyček, které zahrnují všechny jedničky v mapě. Smyčka pak závisí na n-K proměnných. Smyčky se mohou překrývat t.j. mohou zahrnovat jedničky (mintermy) vícekrát do různých termů .

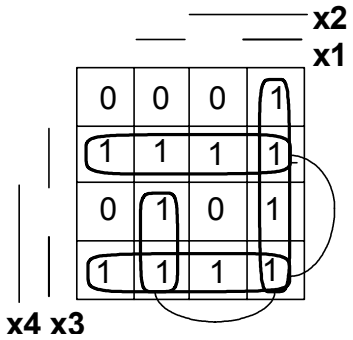
Existuje více minimálních forem podle výběru vhodných minimalizačních smyček. Z obr.2.4. je zřejmé, že minimalizační smyčky sousedních mintermů jednoho termu tvoří v Karnaughově mapě kompaktní smyčku na rozdíl od Svobodovy (binární) mapy, kde minimalizační smyčka jednoho termu může být složena z několika částí topologicky v mapě oddělených (i když jsou sousední - spojnice mezi minimalizačními smyčkami). Proto se pro minimalizaci ze Svobodovy mapy používaly <u>mřížky</u>, které zjednodušovaly výběr sousedních políček.





disjunktivní tvar funkce f $f = x_2\bar{x}_3 + x_1x_3$





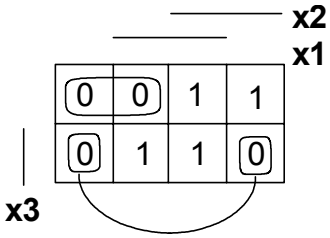
disjunktivní tvar funkce f $f = x_3 + x_1x_2 + x_1x_4$

Obr.2.4. Minimalizace logické funkce ve Svobodově a Karnaughově mapě

V následujícím proto pro jednoduchost budeme používat vyjádření logické funkce i minimalizaci z Karnaughových map.

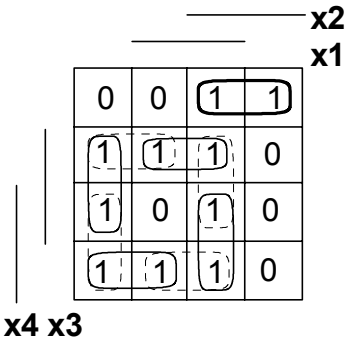
Pro získání <u>konjunktivní formy</u> logického výrazu provádíme "minimalizaci z nul" s použitím principu duality pro všechny proměnné a operace t.j popis pomocí <u>maxtermů</u> (to samé bychom získali vytvořením disjunktivní formy z nul a následnou negací funkce).

konjunktivní tvar funkce f
$f = (x_2 + x_3)(x_1 + \bar{x}_3)$



Obr.2.5. Minimalizace "z nul"

Při minimalizaci z mapy lze nalézt <u>podstatný přímý implikant</u> t.j. takový implikant, který nelze pokrýt žádným jiným logickým výrazem (termem) a nepodstatný <u>přímý implikant</u>, který lze pokrýt i jinak.



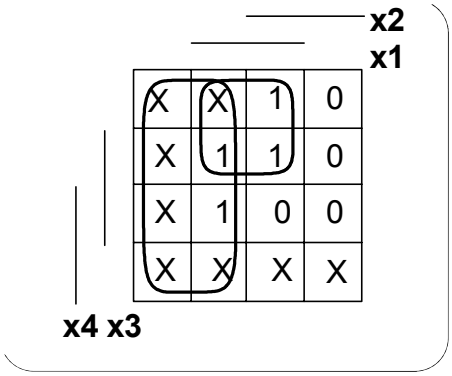
$x_2\bar{x}_3\bar{x}_4$ - podstatný implikant

ostatní - nepodstatné

Obr.2.6. Podstatný a nepodstatný implikant

Každá minimální disjunktivní forma obsahuje <u>všechny podstatné přímé implikanty</u> (jádro řešení) a výběr nepodstatných přímých implikantů

Pro <u>neúplně definovanou</u> logickou funkci můžeme symbol **x** (na hodnotě nezáleží) zahrnout pro 1 i pro 0 mininalizační smyčku. Příklad takto zadané funkce je na obr.2.7.



po minimalizaci je výraz:

$f = x_1\bar{x}_4 + \bar{x}_2$

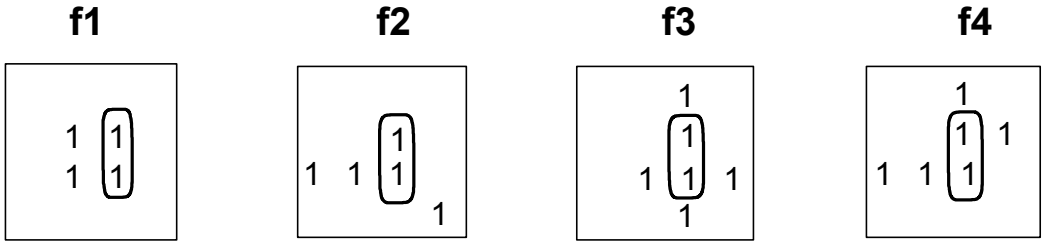
využití značení x zjednoduší realizaci

Obr.2.7. Neúplně definovaná logická funkce

<u>Minimalizace skupiny logických výrazů</u>

Kombinační zobrazení většinou reprezentuje několik logických funkcí. V mapě minimalizujeme pouze jednu logickou funkci, takže nelze exaktně provést <u>skupinovou minimalizaci</u>. Skupinovou minimalizaci lze provést pouze intuitivně z několika map pro jednotlivé logické funkce s cílem minimalizovat realizační spotřebu obvodů pro celé zobrazení (skupinu logických funkcí).

Postupujeme tak, že se snažíme najít ve všech mapách společné implikanty a ty zachovat - <u>skupinové implikanty</u>.



Obr.2.8. Skupinová minimalizace

Posledním krokem při minimalizaci je <u>vytýkání</u> před závorku čímž získáme <u>absolutně minimální formu</u> logické funkce.

<u>3. Minimalizace Quine - McCluskey</u>

Minimalizace Quine - McCluskey je systematickou metodou pro nalezení minimálního normálního tvaru boolské funkce. Vzhledem ke svému jednoznačnému algoritmu je vhodná pro počítač. Je využitelná zejména pro funkce velkého počtu proměnných, kde "ruční" minimalizace je neúnosná.

Postup :
1. Vytvoření standardní (plné) boolské funkce (rozklad funkce na mintermy)- všechny implikanty obsahují všechny proměnné v přímé nebo negované formě
2. Systematické zjednodušování $fa + f\bar{a} = f$ (každý implikant s každým implikantem) - tvoří strom s několika úrovněmi zjednodušení (t.j. výsledek prvé úrovně zjednodušení se opět zjednodušuje metodou "každý s každým" a vzniká druhá úroveň zjednodušení, která se opět zjednodušuje stejným způsobem). V každé úrovni zůstávají členy, které nelze dále zjednodušit - přímé implikanty.
3. Nalezení přímých implikantů funkce (implikanty, které se nedají dále zjednodušit z jednotlivých úrovní stromu). Sestavuje se tabulka, ve které se označuje, které přímé implikanty (řádky) pokrývají které mintermy plné boolské funkce (sloupce). Některé

mintermy boolské funkce mohou být pokryty několika přímými implikanty (nepodstatné přímé implikanty), některé mintermy boolské funkce mohou být pokryty pouze jedním přímým implikantem (podstatný přímý implikant).

4. Nalezení podstatných přímých implikantů - jádro řešení (implikanty, které se nedají nahradit jinými z množiny nalezených přímých implikantů)
5. Výběr minimálního množství nepodstatných přímých implikantů. Sestavuje se opět tabulka pokrytí mintermů plné boolské funkce tentokráte bez podstatných přímých implikantů funkce (jejich řádky se vynechávají) a bez všech mintermů, které tyto implikanty pokrývají (vynechávají se příslušné sloupce. Existují optimalizační postupy pro výběr vhodných tak, aby pokryly ostatní.

Metoda umožňuje skupinovou minimalizaci logických funkcí

Další podrobnosti o této metodě lze nalézt v lit.[2] , [4] a řadě dalších.

4. Minimalizace výrazů s využitím smíšené formy - TANT

Metoda TANT (Tree-level AND-NOT logic with True imputs) generuje tříúrovňové (maximálně 3 členy v serii) logické sítě se členy NAND. Vychází z metody minimalizace Quine - McCluskey s tím, že některé implikanty se doplňují smíšenou formou, která umožňuje jednodušší výslednou realizaci logické funkce než původní metoda Quine - McCluskey (čistě disjunktivní forma):

Př.:	$x_1\bar{x}_3$	eventuelní možnost vyloučení
	$\bar{x}_0\bar{x}_3$	většího počtu implikant jedním
	$x_0x_1x_3$	doplňujícím členem
doplnění	$\bar{x}_3(x_1+\bar{x}_0)$	

S takto doplněnou množinou implikantů se pracuje dál podobným způsobem, jako u metody Quine - McCluskey.
Výběr implikantů se provádí opět přes tabulky pokrytí. Opět existují optimalizační postupy pro výběr vhodných implikantů tak, aby pokryly ostatní.

Další podrobnosti o této metodě lze nalézt v lit [4].

5. Další metody minimalizace

Kromě uvedených metod minimalizace existuje celá řada dalších, které využívají odlišné formy popisu logické funkce. Jednou z nich je minimalizace z 1 a 0 bodů funkce, která využívá tabulkový popis logické funkce a je založena na algebraickém zápisu odlišujících charakteristik 1 bodů (funkce f=1) od všech 0 bodů (funkce f=0) s následující úpravou pomocí zákonů Booleovy algebry. Další metodou minimalizace je Svobodova metoda, která využívá kombinaci zobrazení logické funkce na vícerozměrné krychli. Metoda má jednoznačný algoritmus a vede na disjunktivní tvar logické funkce.

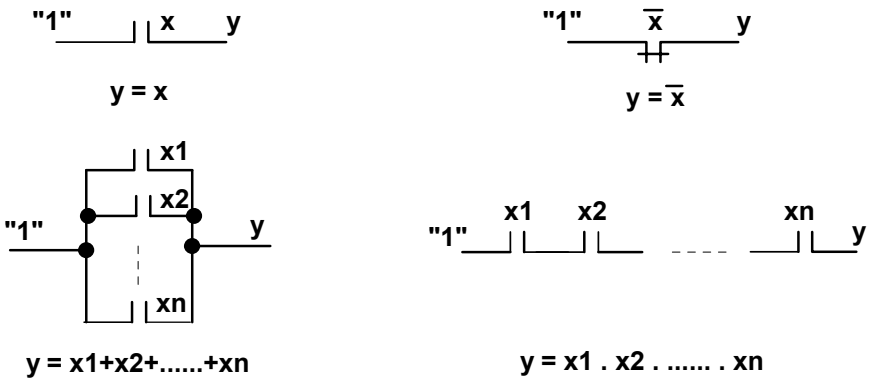
Další podrobnosti o těchto metodách lze nalézt v lit [2].

2.1.4. Realizační prvky

Pro realizaci kombinačních logických obvodů je podmínkou použití realizační soustavy, která vyhovuje požadavkům na úplný soubor logických funkcí. V praxi tyto soustavy nejsou soubory minimální, ale obsahují množství prvků realizujících další složitější funkce dvou nebo více proměnných. Takto pojaté realizační soustavy umožňují návrháři zjednodušit celé řešení logické sítě. V této části budou uvedeny základní logické prvky (kontaktní a IO), které budou používány v dalších kapitolách.

Kontaktní logika

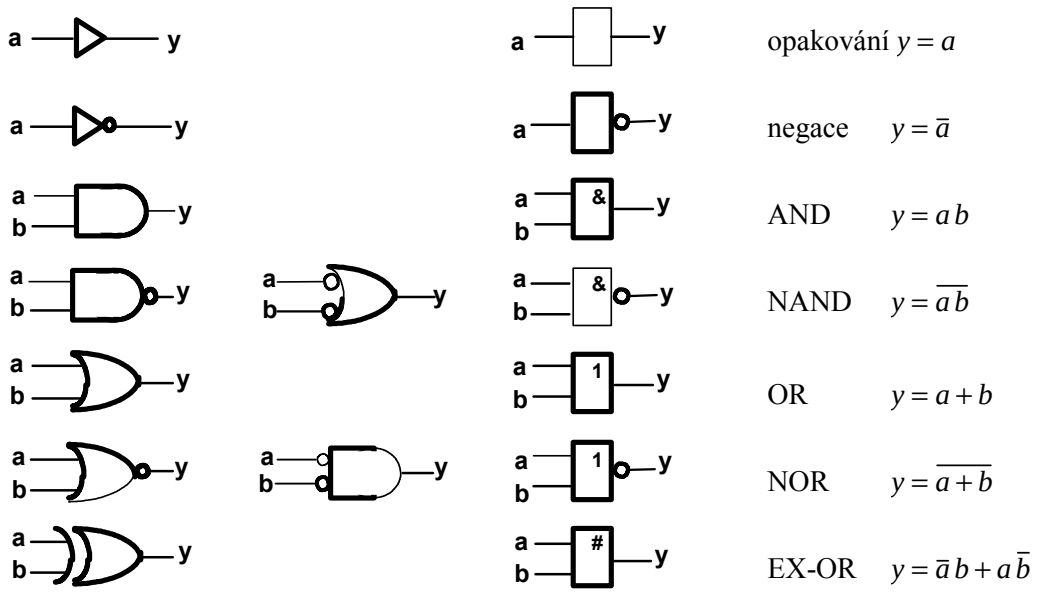
U relé jsou k dispozici spínací a rozpínací kontakty. Jednotlivé základní logické funkce je proto nutné z těchto kontaktů sestavovat - viz obr.2.9.



Obr.2.9. Kontaktní logika

Integrované obvody

Na obr.2.10. jsou uvedeny značky základních, nejčastěji používaných, logických funkcí realizačních soustav s integrovanými obvody. Na obrázku jsou uvedeny dva typy značek - klasická a "krabičková", která byla zavedena pro zjednodušení kreslení. Oba typy značek lze používat.



Obr.2.10. Integrované obvody

2.2. Kombinační logické systémy

Odezva tohoto systému je v určitém časovém okamžiku podmíněna výhradně okamžitými hodnotami proměnných na jeho vstupech.

Chování tohoto systému lze popsat funkcí: $\mathbf{f} : \mathbf{X} \rightarrow \mathbf{Z}$

kde **X** - množina všech vstupních vektorů
Z - množina všech výstupních vektorů
f - množina funkcí

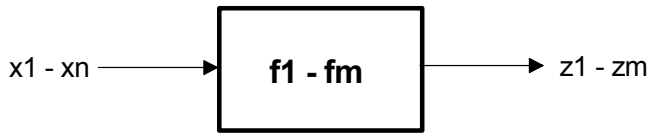
Rovnici můžeme přepsat do tvaru: $\mathbf{z} = \mathbf{f}(\mathbf{x})$

kde $\mathbf{z} \in \mathbf{Z}$ - vstupní vektor
$\mathbf{x} \in \mathbf{X}$ - výstupní vektor
f - množina funkcí

Zobrazení **f** můžeme při n - vstupech a m - výstupech rozložit na m - zobrazení (funkcí):

$$z_1 = f_1(x_1 x_2 ... x_n)$$
$$z_2 = f_2(x_1 x_2 ... x_n)$$
$$\vdots$$
$$z_m = f_m(x_1 x_2 ... x_n)$$

Blokové schema kombinačního logického systému je uvedeno na obr.2.11.



Obr.2.11. Blokové schéma kombinačního systému

Tento kombinační logický systém (množinu funkčních vztahů - algebraický model) můžeme překreslit do logické sítě (logické schema - opět abstraktní model struktury), která podrobně popisuje strukturu kombinačního logického obvodu.

Logická síť je orientovaný multigraf, který se skládá z uzlů (vstupy, výstupy, funkční prvky - logické členy), hran (vzájemné propojení uzlů) a který funkcí odpovídá výchozímu logickému systému. Funkční prvky realizují většinou základní boolské funkce.

2.2.1 Syntéza kombinačních logických obvodů

Syntéza kombinačních logických obvodů odpovídá návrhu logické sítě, která je tvořena modelováním struktury daného logického systému pomocí značek logických členů a jejich vzájemným propojením.
Postup syntézy z praktického pohledu je následující:

1. Ze zadání určíme počet vstupních a výstupních logických proměnných, počet logických funkcí.
2. Ze zadání sestavíme pravdivostní tabulku, mapu nebo jiný vhodný typ popisu logické funkce požadovaného logického obvodu.

3. Z popisu kombinační funkce získáme množinu minimalizovaných funkčních vztahů (algebraický tvar), popisujících zadaný logický systém.
4. Algebraické vyjádření funkčních vztahů převedeme na tvar, který vyhovuje zvolenému souboru logických členů a kriteriím optimality návrhu (min. počet logických členů, maximální rychlost, soubor logických členů, max. počet vstupů logických členů a j.)
5. Nakreslíme strukturu logické sítě a znovu ji optimalizujeme s ohledem na vlastnosti logických členů.
6. Následuje analyza navržené logické sítě a případná korekce návrhu. Zejména se posuzuje možnost vzniku hazardů a jejich odstranění.
7. Konečné sestavení logického obvodu a ověření správné činnosti.

Syntéza v sobě představuje řadu iterací, kdy se k jednotlivým krokům vracíme, abychom co nejvíce splnili požadavek optimálního řešení daného problému.

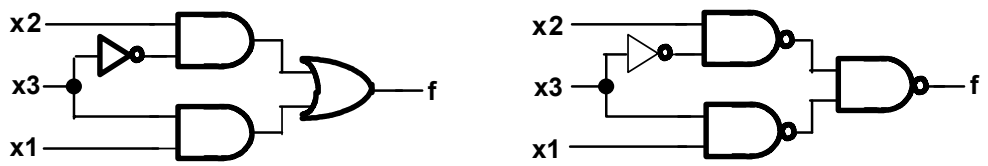
Kriteria optimality návrhu:
1. minimální počet operací (počet logických členů, cena)
2. minimální počet proměnných (počet vstupů logických členů, cena)
3. minimální počet úrovní (minimalizace zpoždění obvodu, rychlost)
4. minimální počet typů realizačních prvků (typ, počet vstupů a j.)

V řadě případů musíme kriteria optimality návrhu obvodu kombinovat. Některé minimalizační metody určitá kriteria optimality podporují.

Úroveň logické sítě je definována jako počet funkčních prvků v nejdelším řetězci logického obvodu. Pak zpoždění tohoto řetězce je při K prvcích (K úrovních logické sítě) K.T , kde T je zpoždění jednoho prvku.

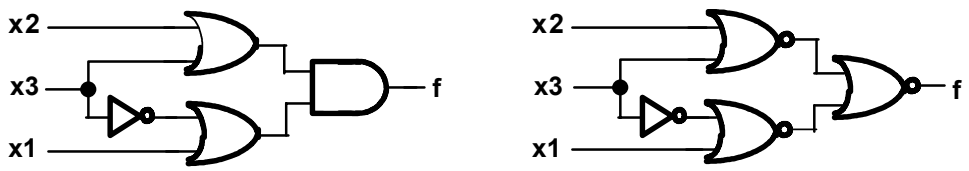
Víceúrovňová schemata vznikají : a) při realizaci logických obvodů prvky s omezeným počtem vstupů
b) při kaskádním řazení jednodušších logických sítí

Základní formou realizace logického systému z disjunktivní nebo konjunktivní formy zápisu logické funkce jsou 3 - úrovňová schemata, přičemž disjunktivní forma vede na strukturu AND/OR nebo NAND (Obr.2.12.) a konjunktivní forma vede na strukturu OR/AND nebo NOR (Obr.2.13).



$f = x_2\bar{x}_3 + x_1x_3$ $\qquad$ $f = \overline{\overline{x_2\bar{x}_3}\ .\ \overline{x_1x_3}}$

Obr.2.12. Tříúrovňová schemata z disjunktivní formy



$f = (x_2 + x_3)(x_1 + \bar{x}_3)$ $\qquad$ $f = \overline{\overline{x_2 + x_3} + \overline{x_1 + \bar{x}_3}}$

Obr.2.13. Tříúrovňová schemata z konjunktivní formy

Struktury NAND a NOR dostaneme z normální formy zápisu dvojí negací výrazu a úpravou pomocí De Morganových pravidel.

Realizace víceúrovňových sítí

Realizace víceúrovňových sítí je nutná zejména při omezeném počtu vstupů realizačních prvků. Vznikají tak víceúrovňové sítě realizované kaskádním řazením sítí jednodušších.

K prostředkům umožňujícím realizaci víceúrovňových sítí patří:

:
1. Faktorizace logické funkce (skupinová minimalizace)
2. Použití De Morganových pravidel
3. Dekompozice logické funkce

1. Faktorizace

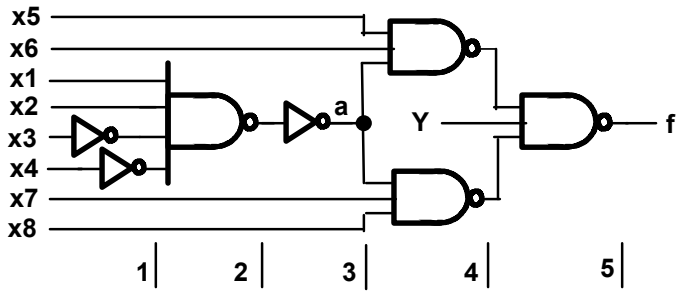
Při složitějších funkcích, funkcích s více výstupy (skupinová minimalizace) je vhodné vyhledat takové termy, které by byly společné pro realizaci implikantů zadané funkce a tím by zjednodušily výslednou realizaci. Těmto termům říkáme faktory a vyhledávání těchto faktorů faktorizace.

Př.: $f = \underbrace{x_1 x_2 \bar{x}_3 \bar{x}_4}_{a} \; x_5 x_6 \; + \; \underbrace{x_1 x_2 \bar{x}_3 \bar{x}_4}_{a} \; \bar{x}_7 x_8 + Y$

faktor

$$f = \overline{\overline{a\, x_5 x_6} \,.\, \overline{a\, \bar{x}_7 x_8} \,.\, \bar{Y}}$$

kde $a = \overline{\overline{x_1 x_2 \bar{x}_3 \bar{x}_4}}$ nebo $a = \overline{\bar{x}_1 + \bar{x}_2 + x_3 + x_4}$



Obr.2.14. Faktorizace logické funkce

2. Použití De Morganových pravidel

Jednou z klasických metod, jak získat vhodnou realizaci funkce s více vstupy je použití De Morganových pravidel. Algebraický tvar logické funkce upravujeme tak, aby odpovídala realizačním schopnostem vybrané realizační soustavy. V zásadě se používá dvojí negace původní funkce a úprava vnitřní negace pomocí De Morganových pravidel (vede na realizaci s prvky NAND a NOR).

Pro zjednodušení aplikace De Morganových pravidel a vytvoření výsledné struktury logické sítě slouží Rottovy mřížky , kde se píší pouze operátory funkce.

Př.: $F = x_3 + x_0 x_2 + \bar{x}_0 x_1 x_2$

NAND

$$\begin{array}{c} x_3 + x_0 \,.\, x_2 + \bar{x}_0 \,.\, x_1 \,.\, x_2 \\ \hline \bullet \quad + \quad \bullet \quad + \quad + \\ | \quad \bullet \quad | \quad \bullet \quad \bullet \\ \bar{x}_3 \,|\, x_0 \,|\, x_2 \,|\, \bar{x}_0 \,|\, x_1 \,|\, x_2 \end{array}$$

NOR

$$\begin{array}{c} x_3 + x_0 \,.\, x_2 + \bar{x}_0 \,.\, x_1 \,.\, x_2 \\ \hline \bullet \quad + \quad \bullet \quad + \quad + \\ + \quad \bullet \quad + \quad \bullet \quad \bullet \;;; \\ | \quad + \quad | \quad + \quad + \\ x_3 \,|\, \bar{x}_0 \,|\, x_2 \,|\, x_0 \,|\, \bar{x}_1 \,|\, \bar{x}_2 \end{array}$$

t.j. $F = \overline{\bar{x}_3 . \overline{x_0 x_2} . \overline{\bar{x}_0 x_1 x_2}}$ $\qquad F = \overline{\overline{x_3 + \overline{\bar{x}_0 + x_2} + \overline{x_0 + \bar{x}_1 + \bar{x}_2}}}$

Pravidla pro vytváření Rottovych mřížek:

1. Střídání operací je vyjádřením De Morganových pravidel
2. Sviské čáry oddělují jednotlivé použité vstupy logických členů
3. Mřížku vytváříme shora a do posledního řádku vpisujeme vstupní proměnné tak, aby počet negačních čar úpravy byl sudý
4. Lze respektovat požadavek omezeného počtu vstupů realizačních prvků
5. Umožňuje současné použití členů NAND a NOR

Příklad úpravy pro omezený počet vstupů :

Pro 2 - vstupové logické členy

NAND

$$\begin{array}{c} x_3 + x_0 \,.\, x_2 + \bar{x}_0 \,.\, x_1 \,.\, x_2 \\ \hline \bullet \quad + \quad \bullet \quad + \quad + \\ + \quad \bullet \quad | \quad \bullet \quad \bullet \\ \bullet \quad + \quad | \quad + \quad | \\ | \quad \bullet \quad | \quad \bullet \quad | \\ \bar{x}_3 \,|\, x_0 \,|\, x_2 \,|\, \bar{x}_0 \,|\, x_1 \,|\, x_2 \end{array}$$

NOR

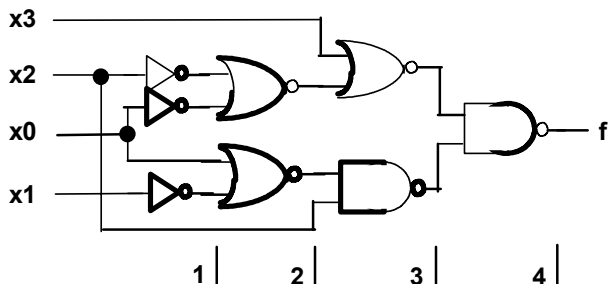
$$\begin{array}{c} x_3 + x_0 \,.\, x_2 + \bar{x}_0 \,.\, x_1 \,.\, x_2 \\ \hline \bullet \quad + \quad \bullet \quad + \quad + \\ + \quad \bullet \quad + \quad \bullet \quad \bullet \\ | \quad + \quad \bullet \quad + \quad + \\ | \quad \bullet \quad + \quad \bullet \quad \bullet \\ | \quad + \quad | \quad + \quad + \\ | \quad | \quad | \quad | \quad \bullet \\ | \quad | \quad | \quad | \quad + \\ x_3 \,|\, \bar{x}_0 \,|\, \bar{x}_2 \,|\, x_0 \,|\, \bar{x}_1 \,|\, \bar{x}_2 \end{array}$$

Takovouto úpravou lze také realizovat logické obvody se současným použitím jak členů NAND tak i členů NOR s volbou počtu vstupů obvodu.

Současné využití NAND a NOR pro 2 -vstupové logické obvody:

$$\begin{array}{c} x_3 + x_0 \,.\, x_2 + \bar{x}_0 \,.\, x_1 \,.\, x_2 \\ \hline \bullet \quad + \quad \bullet \quad + \quad + \\ + \quad \bullet \quad | \quad \bullet \quad \bullet \\ | \quad + \quad | \quad + \quad | \\ x_3 \,|\, \bar{x}_0 \,|\, \bar{x}_2 \,|\, x_0 \,|\, \bar{x}_1 \,|\, x_2 \end{array}$$

$$F = \overline{\overline{x_3 + \overline{\bar{x}_0 + \bar{x}_2}} \,.\, \overline{\overline{x_0 + \bar{x}_1} \,.\, x_2}}$$



Obr.2.15. Příklad 4 - úrovňové sítě

Systematický návrh 3 - úrovňových logických sítí s optimálním počtem prvků a minimálním zpožděním realituje metoda TANT (lit.[4]).

Využití vlastností specielních logických funkcí

Každou logickou funkci více proměnných můžeme nahradit elementárními funkcemi jedné a dvou proměnných - superpozice funkcí. Postačí nám logické funkce součtu, součinu a negace (i to už není minimální soubor funkcí).
Z logických funkcí více proměnných jsou význačné a pro praxi použitelné funkce:

Logický součet	$f = a + b + c + \ .....$
Logický součin	$f = a\ .\ b\ .\ c\ .\ .....$
Schefferova funkce	$f = \overline{a\ .\ b\ .\ c\ .\ .....}$
Pierceova funkce	$f = \overline{a + b + c + \ .....}$
Ekvivalence	$f = a \equiv b \equiv c \equiv \ ....$
Nonekvivalence	$f = a \not\equiv b \not\equiv c \not\equiv \ ....$
Prahové a majoritní funkce	

2.2.2. Symetrické funkce

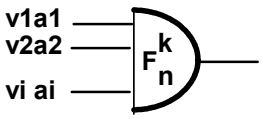
Jsou to takové logické funkce, u nichž se funkce nezmění, zaměněníme li dva libovolné vstupy mezi sebou t.j funkce je invariantní ke všem permutacím vstupních proměnných.
Mezi symetrické funkce dvou i více proměnných patří jednoduché logické funkce typu : logický součet (OR), logický součin (AND), Shefferova funkce (NAND), Pierceova funkce (NOR), ekvivalence a nonekvivalence (XOR).

Prahová logická funkce

Nejdříve několik definic použitých pojmů:

váha v_i - konstanta udávající poměrný vliv i-té nezávislé proměnné (vstupu) na výstup funkce
řád n - algebraický součet všech vah všech nezávisle proměnných
práh k - konstanta - nejmenší hodnota algebraického součtu vah nezávisle proměnných (vstupů) majících hodnotu 1 nutných k tomu, aby výstup = 1
N - počet nezávisle proměnných (vstupů)

Def.: Prahová logická funkce je rovna jedné, když algebraický součet vah nezávisle proměnných majících současně hodnotu 1 je větší nebo roven práhu této funkce.



$$F_n^k(v_i a_i) = 1 \qquad pro \qquad \sum_{i=1}^{N} v_i a_i \geq k$$

Př.: $F_6^4(a, 2b, 3c) = bc + ac + abc = c(b + a$

Symetrická prahová logická funkce

Symetrická prahová logická funkce je prahová logická funkce, jejíž nezávisle proměnné mají všechny stejné váhy ($v_i = 1$).

Def.: Symetrická prahová logická funkce je rovna jedné, když alespoň k z N vstupů jsou současně rovny jedné.

$$F_N^k(a_i) = 1 \qquad pro \qquad \sum_{i=1}^{N} a_i \geq k$$

Př.: $F_4^3(a, b, c, d) = abc + abd + acd + bc$

Majoritní logická funkce

Majoritní logická funkce je symetrická prahová logická funkce lichého řádu s práhem $k = \frac{N+1}{2}$.

Def.: Majoritní logická funkce je rovna jedné, když alespoň nadpoloviční většina vstupů je současně rovna jedné.

značení: $M(a_1, a_2, ... a_n)$

Př.: $M(a,b,c) = ab + ac + bc$

Pro práci s majoritními a prahovými logickými funkcemi a pro jejich vzájemnou náhradu platí řada pravidel a transformačních vět (viz lit[2]). Poměrně jednoduše lze s jejich pomocí realizovat složité kombinační systémy.

Široký okruh náhrady logických funkcí ukazuje následující příklad :

Logický součet $F_N^1(a_1, a_2, ..., a_N) = a_1 + a_2 + ... + a_N$
Logický součin $F_N^N(a_1, a_2, ..., a_N) = a_1 \,.\, a_2 \,.\, ... \,.\, a_N$

2.2.3. Obvody typu XOR

Velmi užitečnými obvody pro realizaci kombinačních logických funkcí jsou obvody typu XOR (nonekvivalence). Neexistuje systematický přístup k jejich použití, ale existuje řada pravidel práce s těmito obvody a způsobů přepisu boolské funkce na funkce s obvody XOR (Žegalkinova algebra). Logická funkce XOR netvoří úplný funkční systém a potřebuje jako doplněk logickou funkci AND.

Základní pravidla :

$$A\overline{B} + \overline{A}B = A\overline{B} \oplus \overline{A}B = A \oplus B \qquad A \oplus \overline{A} = 0$$
$$A + B = A \oplus B \oplus AB = A \oplus \overline{A}B = B \oplus \overline{B}A \qquad A \oplus 1 = \overline{A}$$
$$\overline{A + B} = A \oplus \overline{A}B \oplus 1 = B \oplus \overline{B}A \oplus 1 \qquad A\overline{B} = A(1 \oplus B)$$
$$AB + \overline{A}\overline{B} = AB \oplus \overline{A}\overline{B} = A \oplus \overline{B} = \overline{A} \oplus B \qquad A + \overline{B} = 1 \oplus B \oplus AB$$
$$A \oplus B \oplus C = (A \oplus B) \oplus C = A \oplus (B \oplus C) \qquad A(B \oplus C) = AB \oplus AC$$

Př.: $x_2\overline{x}_3 + x_1x_2 = x_2(\overline{x}_3 + x_1) = x_2(x_1 \oplus \overline{x}_1\overline{x}_3) = \overline{\overline{x}_2 + \overline{x}_1 \oplus \overline{x_1 + x_3}}$

2.2.4. Hradlové struktury

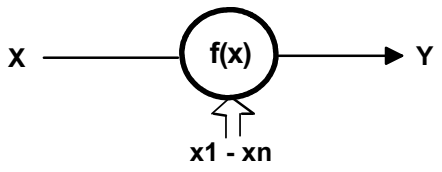
Hradlové struktury jsou specielní kombinační logické obvody, které se realizují z hradel. Vytváří se hradlové sítě, hradlové kombinační obvody.

Hradlo je logický člen, který obsahuje jediný vstup X, jediný výstup Y a množinu řídících signálů $\{ x_1, x_2 ... x_n \}$

Pro tento člen platí:

je-li $f(x_1, x_2, ... x_n) = 1$ pak $Y = X$ (resp. $Y = \overline{X}$)
je-li $f(x_1, x_2, ... x_n) = 0$ pak $Y = 0$ (resp. $Y = 1$)

Mluvíme o hradle a invertujícím hradle

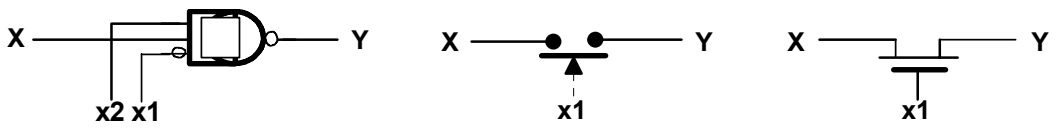


Obr.2.16. Hradlo

Průchodnost hradla - definuje boolská funkce $f(x) = f(x_1, x_{2,}....., x_n)$

pokud f (x) = 1 - hradlo je průchodné
pokud f (x) = 0 - hradlo je neprůchodné

Příklady hradel:

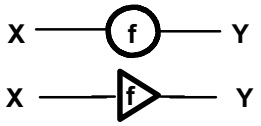


$f(x_1, x_2) = \bar{x}_1 x_2$ f (x) = x_1 f (x) = x_1

Obr.2.17. Příklady hradel

Symetrická hradla - je možný průchod oběma směry

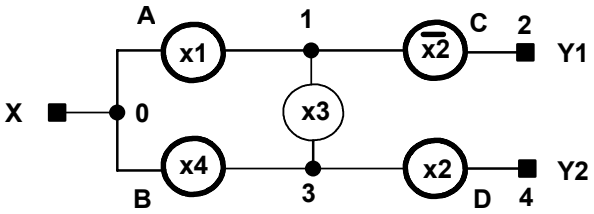
Nesymetrická hradla - průchod pouze jedním směrem



Hradlové struktury se používají velice často jako interpretace kontaktních logických sítí nebo jako prostředky analyzy hradlových polí v programovatelných logických obvodech. Velice dobře se dají také použít pro analyzu průchodu signálů resp. zpráv ve sdělovacích nebo počítačových sítích s větším počtem uzlů sítě (uzel sítě v tomto případě reprezentuje hradlo).

Hradlová síť se často nazývá (k,m) - pól , kde k je počet vstupních uzlů a m je počet výstupních uzlů sítě.

Př.:



Obr.2.18. Hradlová síť

Průchodnost signálové cesty je boolská funkce, která popisuje podmínky průchodu signálu mezi dvěma uzly hradlové sítě.

Př.: Pro hradlovou síť z obr.2.18 jsou podmínky průchodu signálu mezi jedním vstupním a dvěma výstupními uzly sítě dána čtyřmi boolskými funkcemi pro čtyři různé cesty:

$$f_{02}^1 = x_1\bar{x}_2 = g_A g_C \qquad f_{04}^1 = x_4 x_2 = g_B g_D$$
$$f_{02}^2 = x_4 x_3 \bar{x}_2 = g_B g_E g_C \qquad f_{04}^2 = x_1 x_3 x_2 = g_A g_E g_D$$

kde g jsou průchodnosti jednotlivých hradel.

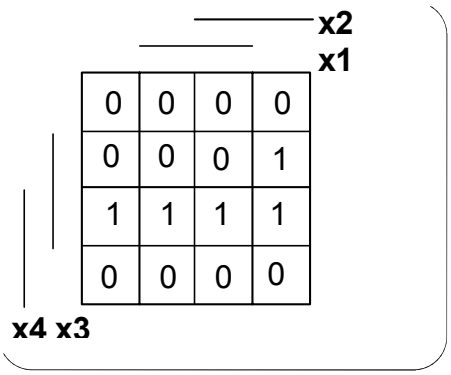
Pro výstupy hradlové sítě platí <u>výstupní funkce</u> :

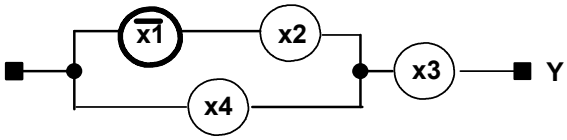
$$Y_1 = \left(f_{02}^1 + f_{02}^2\right) X = (x_1\bar{x}_2 + x_4x_3\bar{x}_2)\, X$$
$$Y_2 = \left(f_{04}^1 + f_{04}^2\right) X = (x_2x_4 + x_1x_3x_2)\, X$$

<u>Průchodností struktury</u> mezi uzly i a j nazýváme funkci vytvořenou jako logický součet všech boolských funkcí popisujících podmínky průchodu signálu všemi možnými cestami.

$$f_{ij} = f_{ij}^1 + f_{ij}^2 + ... + f_{ij}^r$$

<u>Návaznost hradlového dvojpólu a mapy</u>
Platí stejná pravidla jako pro realizaci logických sítí kontaktní logikou t.j. pro logický součet je to paralelní řazení hradel, pro logický součin je to seriové řazení hradel.

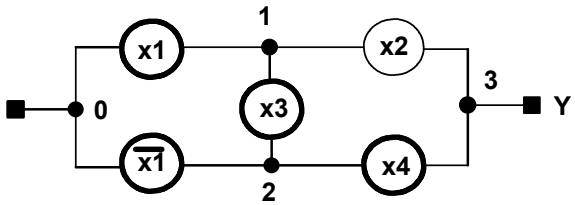




$$y = x_3(x_4 + \bar{x}_1x_2)$$

Obr.2.19. Návaznost hradlového dvojpólu a mapy

Př.: <u>můstkový dvojpól</u>



průchodnost signálových cest :

$f_{03}^1 = x_1x_2$ $\qquad$ $f_{03}^3 = x_1\bar{x}_3x_4$

$f_{03}^2 = \bar{x}_1x_4$ $\qquad$ $f_{03}^4 = \bar{x}_1\bar{x}_3x_2$

přičemž $\bar{x}_3$ je obousměrné hradlo

Obr.2.20. Můstkový dvojpól

průchodnost struktury je: $\quad f_{03} = x_1x_2 + \bar{x}_1x_4 + x_1\bar{x}_3x_4 + \bar{x}_1\bar{x}_3x_2$
$\qquad f_{03} = x_1(x_2 + \bar{x}_3x_4) + \bar{x}_1(x_4 + \bar{x}_3x_2)$

Tato rovnice představuje <u>strukturní vzorec</u> hradlové sítě.

<u>Analyza hradlových (k,m) pólů</u>

Analyzou hralových (k,m) pólů rozumíme vyhledávání výstupních funkcí hradlových sítí se zadanou strukturou.

Postup :
1. zjištění všech signálových cest mezi vstupem i a výstupem j
2. zápis funkce průchodnosti signálové cesty mezi póly i a j
3. zápis funkce strukturní průchodnosti mezi póly i a j
4. zápis výstupních funkcí hradlového (k,m) pólu

Formální popis struktury hradlových (k,m) pólů

Pro formální popis hradlových (k,m) pólů se používají boolské matice.

Pro hradlovou strukturu s p-uzly (včetně vstupů a výstupů) sestavíme čtvercovou matici nazývanou <u>matice přímých průchodností</u> :

$$G = \begin{Vmatrix} g_{11} & g_{12} & \cdots & g_{1p} \\ g_{21} & g_{22} & \cdots & g_{2p} \\ \vdots & \vdots & \cdots & \vdots \\ g_{p1} & g_{p2} & \cdots & g_{pp} \end{Vmatrix}$$

kde g_{ij} - průchodnost z uzlu i do uzlu j
$g_{ij} = 0$ - uzly nejsou sousední (nemají návaznost)
$g_{ij} = 1$ - pro uzly s i = j

Př.: Pro příklad můstkové struktury dvojpólu z obr.2.20 je matice přímých průchodností:

$$G = \begin{Vmatrix} 1 & x_1 & \bar{x}_1 & 0 \\ x_1 & 1 & \bar{x}_3 & x_2 \\ \bar{x}_1 & \bar{x}_3 & 1 & x_4 \\ 0 & x_2 & x_4 & 1 \end{Vmatrix}$$

pro symetrická hradla je matice symetrická
a naopak
prvky diagonály jsou vzdy rovny 1

<u>Matice strukturních průchodností</u> (matice výstupních funkcí)

$$F = \begin{Vmatrix} 1 & f_{12} & f_{13} & \cdots & f_{1p} \\ f_{21} & 1 & f_{23} & \cdots & f_{2p} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ f_{p1} & f_{p2} & f_{p3} & \cdots & 1 \end{Vmatrix}$$

kde f_{ij} je strukturní průchodnost mezi uzly i a j

Př.: Pro můstkovou strukturu dvojpólu z Obr.2.20 :

$$F = \begin{Vmatrix} 1 & f_{01} & f_{02} & f_{03} \\ f_{10} & 1 & f_{12} & f_{13} \\ f_{20} & f_{21} & 1 & f_{23} \\ f_{30} & f_{31} & f_{32} & 1 \end{Vmatrix}$$

kde

$$f_{01} = x_1 + \bar{x}_1\bar{x}_3 + \bar{x}_1 x_4 x_2$$
$$f_{02} = \bar{x}_1 + x_1\bar{x}_3 + x_1 x_2 x_4$$
$$atd.$$

pro symetrická hradla platí $f_{01} = f_{10}$

<u>Matici strukturních průchodností F</u> lze získat z matice G jejich násobením :

$$F = G^{p-1} = \underbrace{G \,.\, G \,.\, G \,.\, \ldots \,.\, G}_{(p-1)\ krát}$$

kde p je počet pólů

Je to obvyklé násobení matic, kde součty a součiny jsou logické.

Další podrobnosti o hradlových sítích lze nalézt v lit.[3].

2.2.5. Dekompozice kombinačních logických funkcí

Dekompozice kombinačních logických funkcí opět vychází z problému omezeného počtu vstupů realizačních logických členů. Snažíme se proto zjednodušit jak návrh tak i realizaci logického obvodu vytvořením víceúrovňové logické sítě.

Přístup k dekompozici je dvojí:
1. Algebraická formální dekompozice logické funkce.
2. Dekompozice na základě funkčního chování, která předpokládá rozklad složité funkce na několik jednodušších se vzájemným propojením.

1. Formální dekompozice

Pod pojmem dekompozice rozumíme vyjádření funkce $f(x_1,x_2...x_n)$ jako kompozici jednodušších s menším počtem vstupních proměnných.

Triviální rozklady

Triviální rozklady vychází ze Shannonova teoremu o rozkladu funkce. Logickou funkci můžeme rozkládat podle jedné nebo více proměnných.

1.Rozklad funkce podle jedné proměnné
Pro rozklad funkce podle jedné proměnné platí:

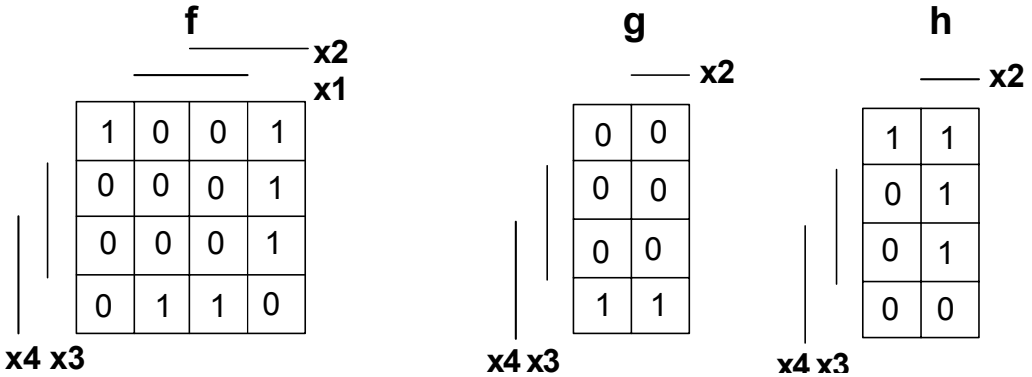
$$f(x_1,x_2,...,x_n) = x_i \,.\, f(x_1,...,x_{i-1},1,x_{i+1},...,x_n) + \bar{x}_i \,.\, f(x_1,...,x_{i-1},0,x_{i+1},....x_n)$$

Funkci $f(x_1,x_2,...,x_n)$ jsme vyjádřili jako kompozici tří funkcí:

$$F(x_i) = x_i$$
$$g(x_1,...,x_{i-1},x_{i+1},...,x_n)$$
$$h(x_1,...,x_{i-1},x_{i+1},...,x_n)$$

Zkráceně lze pak psát: $f = Fg + \bar{F}h$

Př.: Pro příklad rozkladu funkce f podle proměnné x_1 z obr.2.21. můžeme psát:



Obr.2.21. Rozklad funkce podle jedné proměnné

Platí: $F(x_i) = x_1$ $\quad g = \bar{x}_3 x_4$ $\quad h = (\bar{x}_3\bar{x}_4 + x_3x_2)$
Výsledný rozklad funkce f je: $f = x_1 \,.\, \bar{x}_3x_4 + \bar{x}_1 \,.\, (\bar{x}_3\bar{x}_4 + x_3x_2)$

2. Rozklad podle více proměnných

Totéž platí i pro rozklady, kdy funkce F představuje více proměnných. Pro rozklad funkce podle r - proměnných platí:

$$f(x_1, x_2, ..., x_n) = \sum_{i=0}^{2r-1} m_i(x_{i1}, x_{i2}, ..., x_{ir}) \,.\, g(x_{ir+1}, ..., x_{in})$$

Přičemž: $\{x_{i1}, ..., x_{ir}\} \cap \{x_{ir+1}, ..., x_{in}\} = 0$
$\{x_{i1}, ..., x_{ir}\} \cup \{x_{ir+1}, ..., x_{in}\} = \{x_1, ..., x_n\}$

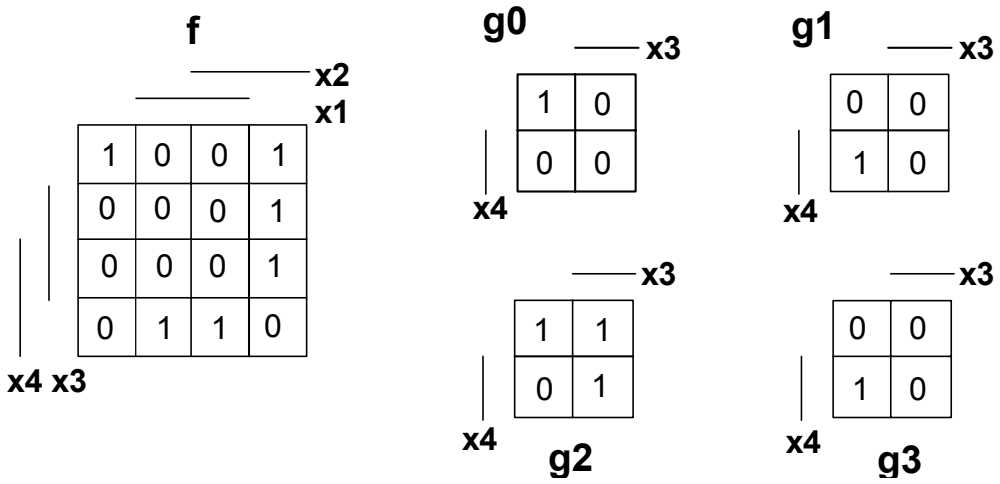
t.j. jsou podmnožiny množiny vstupních signálů $\{x_1, x_2, ..., x_n\}$ pro $1 \le r \le n-1$

Funkce m_i jsou elementární součiny proměnných $\{x_{i1}, ..., x_{ir}\}$ pro které platí:

$$\begin{aligned} m_0 &= \bar{x}_{i1}\bar{x}_{i2}...\bar{x}_{ir} \\ m_1 &= x_{i1}\bar{x}_{i2}...\bar{x}_{ir} \\ &\vdots \\ m_r &= x_{i1}x_{i2}...x_{ir} \end{aligned}$$

a $g(x_{ir+1}, ..., x_{in})$ jsou funkce, které vzniknou z funkce $f(x_1, x_2, ..., x_n)$ po dosazení konstant 0,1.

Př.: Pro rozklad funkce f podle dvou proměnných x_1 a x_2 z obr.2.21. lze psát:



Obr.2.22. Rozklad funkce podle dvou proměnných

Obecně pro rozklad podle dvou proměnných:

$$f(x_1, x_2, x_3, x_4) = m_0 g_0 + m_1 g_1 + m_2 g_2 + m_3 g_3$$

t.j.

$$f(x_1, x_2, x_3, x_4) = \bar{x}_1\bar{x}_2.f(0, 0, x_3, x_4) + x_1\bar{x}_2.f(1, 0, x_3, x_4) + \bar{x}_1 x_2.f(0, 1, x_3, x_4) + x_1 x_2.f(1, 1, x_3, x_4)$$

Vlastní rozklad funkce f z obr.2.22 je tedy:

$$f = \bar{x}_1\bar{x}_2 \,.\, \bar{x}_3\bar{x}_4 + x_1\bar{x}_2 \,.\, \bar{x}_3 x_4 + \bar{x}_1 x_2(\bar{x}_4 + \bar{x}_3) + x_1 x_2 \,.\, \bar{x}_3 x_4$$

Jednoduché disjunktivní rozklady

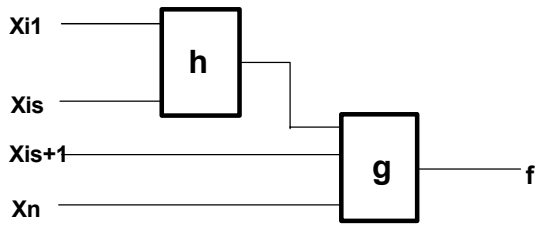
Je to rozklad funkce pro který platí:

$$f(x_1, x_2, ..., x_n) = g\,[h\,(x_{i1}, x_{i2}, ..., x_{is}), x_{is+1}, x_{is+2}, ..., x_{in}]$$

přičemž $\{x_{i1}, ..., x_{is}\} \cup \{x_{is+1}, ..., x_{in}\} = \{x_1, x_2, ..., x_n\}$

pro $1 \le s \le n-1$ Je li s = 1, pak rozklad odpovídá Shannonovu teorému o triviálním rozkladu funkce.

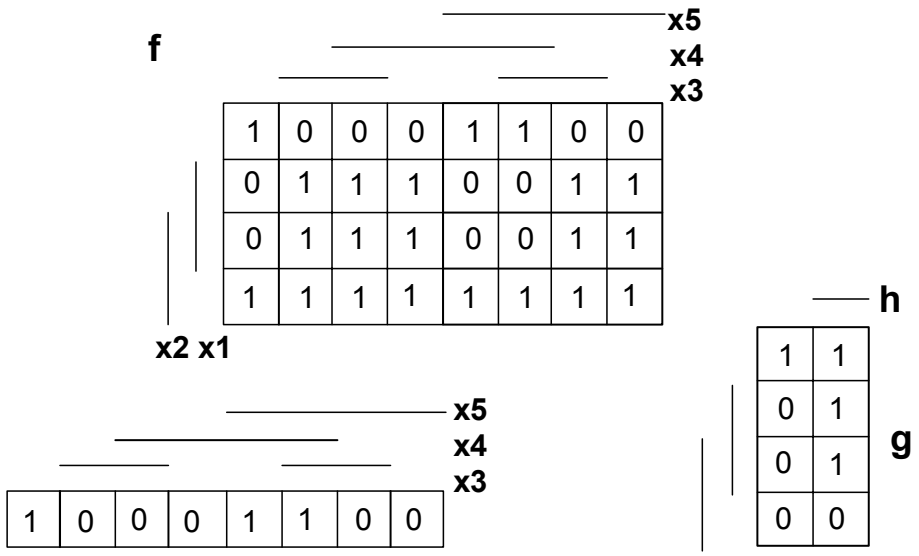
Rozklad odpovídá blokovému schematu na obr.2.23.



Nejlépe je vidět rozklad funkce se znázorněním v mapě. Mapu funkce je třeba upravit tak, aby sloupce (resp. řádky) proměnných, podle kterých rozkládáme (funkce h) byly pouze dvou typů. Potom původní mapu můžeme rozložit na dvě, které určují funkce g a h.

Obr.2.23. Jednoduchý disjunktivní rozklad

Př.: Na Obr.2.24. je funkce pěti proměnných, kterou rozkládáme podle proměnných $x_3x_4x_5$



Obr.2.24. Rozkladové mapy jednoduchého disjunktivního rozkladu

Sloupce mapy představují funkci h s přiřazením: sloupec 1001 odpovídá h=1
sloupec 0111 odpovídá h=0

Z rozkladové mapy pro funkci **h** plyne:

$$h = x_5x_4 + \bar{x}_5\bar{x}_4\bar{x}_3 = \overline{\overline{x_5x_4}\ .\ \overline{\bar{x}_5\bar{x}_4\bar{x}_3}}$$

Přiřazení sloupců proměnné **h** definuje druhá rozkladová mapa pro **g**, z níž plyne:

$$g = x_1\bar{x}_2 + \bar{h}x_2 + h\bar{x}_2 = \overline{\overline{x_1\bar{x}_2}\ .\ h \oplus x_2}$$

Původní funkce tedy dostává tvar:

$$f(x_1, x_2, x_3, x_4, x_5) = g\ [h\ (x_3, x_4, x_5), x_1, x_2]$$

Tento typ rozkladu je zejména vhodný pro neúplně určené funkce, kde neurčené funkční hodnoty (x) můžeme dodefinovat tak, aby byla splněna podmínka pouze dvou typů sloupců (resp. řádek) mapy rozkládané funkce.

Složené disjunktivní rozklady

Tyto rozklady principielně vycházejí z jednoduchých disjunktních rozkladů, ale mají více než jednu podfunkci. Existují dvě varianty:

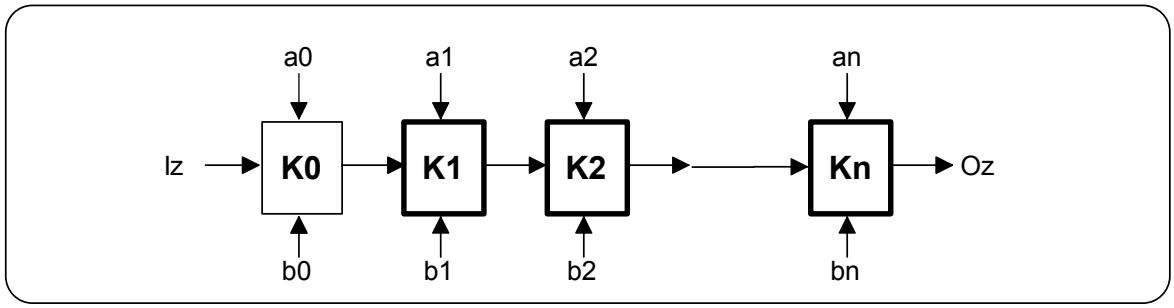
a. Iterační disjunktivní rozklad

$$f(x_1, x_2, ..., x_n) = g\ (h_m(...h_3(h_2(h_1(\mathbf{x}_1), \mathbf{x}_2), \mathbf{x}_3)...), \mathbf{x}_m)$$

kde $\mathbf{x}_1, ..., \mathbf{x}_m$ jsou podmnožiny vstupních proměnných $\{x_1, ..., x_n\}$

Z tohoto rozkladu vznikají iterační obvody. Realizují se stálým rozkladem podfunkcí. Příklad takového iteračního obvodu je uveden na obr.2.25.

Př.: Vícebitový komparátor dvou čísel A a B



Obr.2.25. Iterační obvod - vícebitový komparátor dvou čísel

b. Vícenásobný disjunktivní rozklad

$$f(x_1, x_2, ..., x_n) = g\ (h_1(\mathbf{x}_1), h_2(\mathbf{x}_2), ..., h_m(\mathbf{x}_m))$$

kde $\mathbf{x}_1, ..., \mathbf{x}_m$ jsou podmnožiny vstupních proměnných $\{x_1, ..., x_n\}$.

V tomto případě se provádí rozklad funkce jak podle sloupců tak i podle řádků.

2. Dekompozice na základě funkčního chování

Jde o intuitivní přístup k řešení složitých kombinačních obvodů s velkým počtem vstupních a výstupních proměnných. U těchto obvodů nelze postupovat klasickou cestou od pravdivostní tabulky resp. mapy přes minimalizaci k logické síti, protože pravdivostní tabulky (resp. mapy) jsou příliš rozsáhlé. Například pro 10 vstupních proměnných (8 dat a 2 řídicí signály) by pravdivostní tabulka měla 2^{10} t.j. 1024 řádků. Východiskem z této situace jsou dvě možnosti:

- použít neúplnou pravdivostní tabulku funkce
- provést funkční dekompozici logického systému

Neúplná pravdivostní tabulka
Pokud to zadání a popis funkce dovoluje, pak navrhnout neúplnou pravdivostní tabulku s označením **x** pro situace, kdy na proměnných nezáleží. Z této tabulky rovnou realizovat vztahy pro výstupní proměnné logického obvodu.

Př.: Máme navrhnout prioritní kodér z 8-mi vstupů $\{x_0, ..., x_7\}$ na 3 výstupy $\{z_0, z_1, z_2\}$ kódované v binárním tvaru. Nejvyšší priorita je 7, nejnižší je 0. Na výstupu je vždy binární číslice odpovídající aktivnímu vstupu s nejvyšší prioritou. Výstup Y indikuje, že není na vstupu žádný aktivní signál. Úplná pravdivostní tabulka by měla 256 řádek.

Neúplně definovaná tabulka:

x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0	z_2	z_1	z_0	y
1	x	x	x	x	x	x	x	1	1	1	0
0	1	x	x	x	x	x	x	1	1	0	0
0	0	1	x	x	x	x	x	1	0	1	0
0	0	0	1	x	x	x	x	1	0	0	0
0	0	0	0	1	x	x	x	0	1	1	0
0	0	0	0	0	1	x	x	0	1	0	0
0	0	0	0	0	0	1	x	0	0	1	0
0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1

Z tabulky lze přímo psát:

$$z_2 = x_7 + x_6 + x_5 + x_4$$

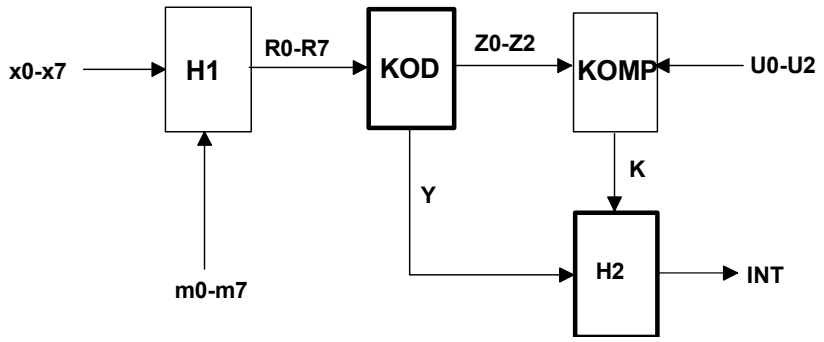
$$z_1 = x_7 + x_6 + \bar{x}_5\bar{x}_4(x_3 + x_2)$$

$$z_0 = x_7 + \bar{x}_6(x_5 + \bar{x}_4(x_3 + \bar{x}_2 x_1))$$

$$Y = \bar{x}_0\bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5\bar{x}_6\bar{x}_7$$

Funkční dekompozice
Druhý způsob je provést funkční dekompozici logického systému na řadu vzájemně propojených, ale funkčně jednoduchých bloků. Klasickou formou syntézy potom můžeme jednoduše navrhnout jak celkovou strukturu logického obvodu tak i jeho jednotlivých bloků. Do určité míry tento postup odpovídá intuitivnímu přístupu k vícenásobnému disjunktivnímu rozkladu logické funkce.

Př.: Máme realizovat obvod žádosti o přerušení, který generuje žádost o přerušení INT, pokud zakódovaný požadavek x_i z prioritního kodéru je větší než definovaná úroveň přerušení u_i. Součástí obvodu je i maska přerušení m_i, která umožňuje některý požadavek přerušení zablokovat.



Obr.2.26. Obvod žádosti o přerušení

Navržené blokové schema je uvedeno na obr.2.26:

požadavky	$x_0, ..., x_7$	
maska	$m_0, ..., m_7$	celkem 19 vstupních proměnných a
úroveň	$u_0, ..., u_2$	1 výstupní proměnná INT

Celou logickou síť jsme dekomponovali na pět jednodušších. Jsou to:

H1 - hradlová síť pro výběr maskou povolených (m=1 maskuje) žádostí o přerušení
KOD - prioritní kodér povolených požadavků o přerušení
KOMP - komparátor $z \geq u$
H2 - hradlo generace INT při $z \geq u$ a současně Y = 0

2.2.6. Hazardy v kombinačních logických obvodech

Logický výraz nebo i kombinační logická síť jsou pouze zidealizovaným strukturním modelem kombinačního logického obvodu platným pro ustálený stav. Toto vyjádření ale nevyjadřuje do důsledků reálné chování logického obvodu, který tento strukturní model realizuje, a to zejména v přechových stavech. Každý logický člen má zpoždění průchodu signálu (konečná doba reakce) a navíc existují různé délky větví realizovaného kombinačního systému, které se mezi sebou liší dobou průchodu signálu (zpožděním). Z toho vyplývají i přechodové děje na výstupech obvodu, které reprezentují přechodové chyby obvodu - hazardy a které zidealizovaný strukturní model neuvažuje.
Zjednodušeně můžeme říci, že hazard je přechodová chyba v důsledku konečné doby reakce použitých realizačních prvků.

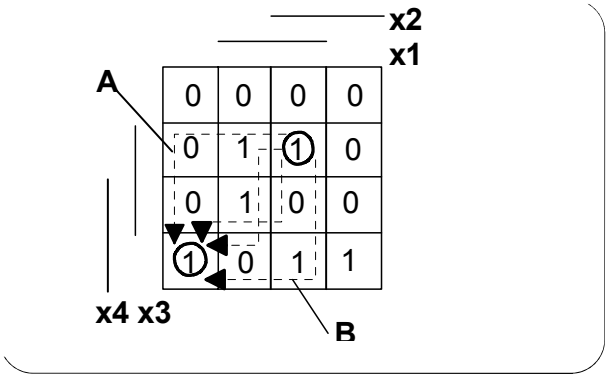
Hazardy dělíme na:
1. Funkční hazardy - hazard nevhodné změny vstupních proměnných
2. Logické hazardy - hazard realizace navržené struktury s reálnými prvky

V samotných kombinačních systémech nemají hazardy tak velký význam jako tehdy, kdy kombinační síť je součástí sekvenční sítě. Délky hazadních pulzů (t.j. změn signálů, které neodpovídají ideální realizaci strukturního modelu logické sítě - jsou nadbytečné) mohou být tak dlouhé, že sekvenční síť je akceptuje jako standardní a tím může zcela změnit svoje chování.

1. Funkční hazardy

Jsou to přechodové stavy, které vznikají na základě reálného průběhu změn vstupních proměnných. Tento případ nastává zejména při změně několika vstupních proměnných najednou.

Př.: Změna čtyř vstupních proměnných logického systému je znázorněna na obr.2.27.



změna vstupního vektoru z 0111 na 1000

Protože nelze předpokládat, že se všechny vstupní proměnné změní současně, mohou nastat v přechodovém stavu různé posloupnosti změn vstupního vektoru (4 cesty) a tím se změní i požadovaná odezva logického obvodu.

Obr.2.27. Funkční hazard

Takže např. pro přechody (cesty) z obr.2.27:

A	x:	$\underline{0111} \rightarrow 1111 \rightarrow 1011 \rightarrow 1001 \rightarrow \underline{1000}$
	z:	$\underline{1} \rightarrow 0 \rightarrow 1 \rightarrow 0 \rightarrow \underline{1}$
B	x:	$\underline{0111} \rightarrow 0101 \rightarrow 0100 \rightarrow 1100 \rightarrow \underline{1000}$
	z:	$\underline{1} \rightarrow 1 \rightarrow 0 \rightarrow 0 \rightarrow \underline{1}$

Chyby nejsou způsobeny realizací, ale vyplývají z funkce systému jako odezva na chybný vstup. Situace je tím horší, čím má logická funkce více vstupních proměnných. Každé cestě odpovídá jiná posloupnost změn výstupních proměnných v přechodovém stavu.

Zjednodušeně můžeme říci, že ***funkční hazardy jsou správné odezvy na nesprávné vstupní vektory***. Tyto hazardy nejsou odstranitelné ani jinou obvodovou realizací kombinační sítě ani pomocnými obvody.

Východiskem při odstranění tohoto typu hazardu jsou následující tři možnosti:

1. Filtr na výstupech logického obvodu, který potlačí hazardní přechody
2. Zákaz vícenásobných změn na vstupu obvodu - fundamentální režim
3. Synchronizace kombinačního obvodu t.j. zavedení dalšího signálu, který vzorkuje ustálené stavy logické sítě.

2. Logické hazardy

I za předpokladu funkce logického systému ve fundamentálním režimu (změna vstupních vektorů pouze v jedné proměnné - sousední vektory) mohou vzniknout hazardy, které závisí na realizaci logické sítě a na konečné době reakce realizačních prvků.

Rozlišujeme:

1. Statický hazard - při změně vstupního vektoru se v přechodovém ději na výstupu, který by měl zachovat svou původní úroveň ($1 \rightarrow 1, 0 \rightarrow 0$) objeví posloupnost změn např. ($1 \rightarrow 0 \rightarrow 1$ *resp.* $0 \rightarrow 1 \rightarrow 0$).
2. Dynamický hazard - při změně vstupního vektoru se při přechodovém ději na výstupu, který by se měl změnit na inverzní úroveň ($1 \rightarrow 0$ *resp.* $0 \rightarrow 1$) objeví posloupnost změn rozdílná od požadované např. ($1 \rightarrow 0 \rightarrow 1 \rightarrow 0$ *resp* $0 \rightarrow 1 \rightarrow 0 \rightarrow 1$).

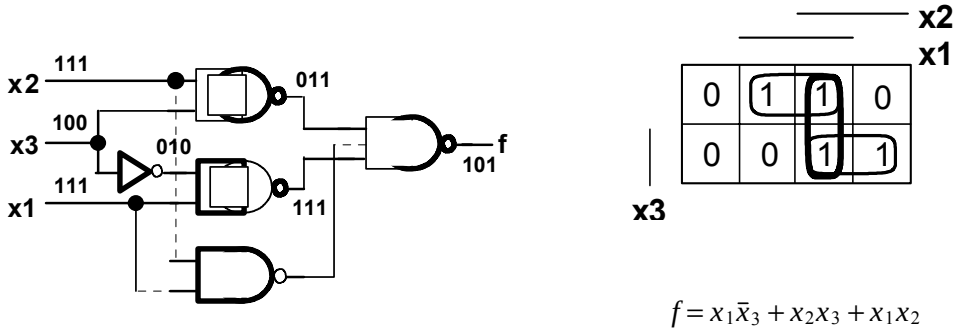
Obvod, ve kterém tyto hazardy nemohou vzniknout nazýváme bezhazardní. Takovýto obvod lze pro tyto typy hazardů z původní logické sítě navrhnout.

Statický hazard

Statický hazard může existovat v 0 nebo v 1. Jde o situaci, kdy přechod mezi sousedními vektory má za následek dočasnou změnu výstupu proti teoretickému předpokladu. Příčinou hazardů je různé zpoždění různě dlouhých větví kombinační logické sítě. Liší se podle strukturního modelu sítě na hazard v 1 (disjunktivní forma) nebo v 0 (konjunktivní forma). Příčina - různé zpoždění různě dlouhých větví sítě.

1. Kombinační obvod realizovaný z disjunktivní formy

Př.: Na obr.2.28. je příklad logického obvodu s hazardem v 1 a jeho odstranění.



Obr.2.28. Statický hazard v 1 a jeho odstranění

Hazard v 1 vzniká při přechodu z $111 \rightarrow 011$ vstupního vektoru $x_3x_2x_1$ nebo obráceně. Je to dáno tím, že reálný obvod nesplňuje základní vztah definovaný Booleovou algebrou $a \,.\, \bar{a} = 0$ *nebo* $a + \bar{a} = 1$.

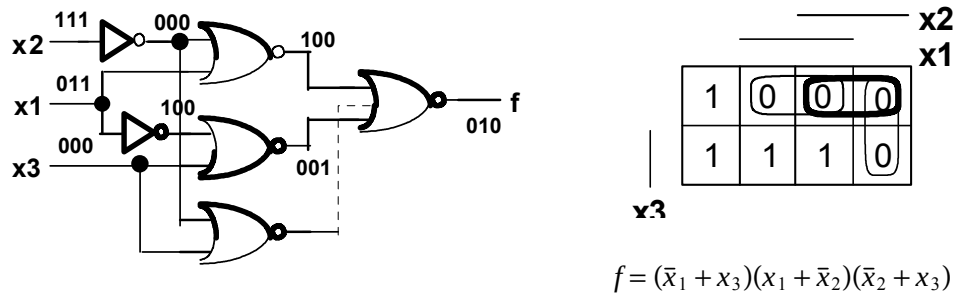
Při realizaci prvky AND, OR, negace dochází k hazardu pro $x_3 : 1 \rightarrow 0$
Při realizaci prvky NOR dochází k hazardu pro $x_3 : 0 \rightarrow 1$

U těchto obvodů vzniká pouze statický hazard v 1.

<u>Odstranění</u> : a) Kompenzace delšího zpoždění vkládáním zpožďovacích členů do rychlejších větví.
b) Vložení redundandních členů, které eliminují vliv zpoždění a překryjí hazardní přechod - <u>podmínka překrytí.</u> Tímto dostaneme bezhazardní disjunktivní formu. Redundandní člen pro příklad z br.2.28. je x_1x_2 a je v mapě označen silně.

<u>2. Kombinační obvod realizovaný z konjuktivní formy</u>

Př.: Na obr.2.29. je příklad logického obvodu s hazardem v 0 a jeho odstranění.



Obr.2.29. Statický hazard v 0 a jeho odstranění

Hazard vzniká při přechodu z $010 \rightarrow 011$ vstupního vektoru $x_3x_2x_1$ nebo obráceně.

Při realizaci prvky AND, OR, negace dochází k hazardu pro $x_1 : 0 \rightarrow 1$
Při realizaci prvky NAND dochází k hazardu pro $x_1 : 1 \rightarrow 0$

U těchto obvodů vzniká pouze statický hazard v 0.

Odstranění statického hazardu je stejné jako u disjunktivní formy. Podmínka překrytí (redundandní člen) je pro příklad z obr.2.29. $(\bar{x}_2 + x_3)$ a je označena v mapě silně.

3. Smíšená forma
U smíšené formy mohou vznikat jak hazardy v 0 tak i v 1. Způsoby jejich odstranění jsou stejné jako u předešlých obou případů.

Odstranění statických hazardů
V zásadě pro odstranění statických hazardů platí:

> Pokrýt všechny sousední 1 (resp.0) v mapě společnými minimalizačními smyčkami (smyčky musí být překryté) i za cenu redundandních členů.

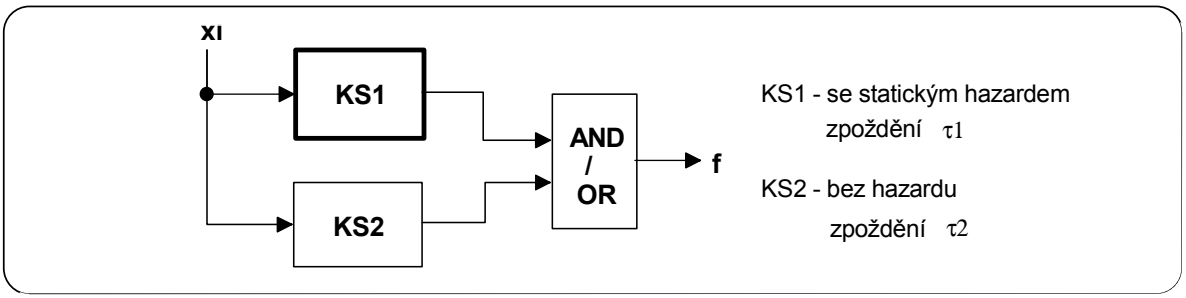
Při skupinové minimalizaci dochází k častému vzniku statických hazardů. Při jejich odstraňování pomocí redundandních členů vlastně přejdeme na výsledek jako u minimalizace individuální, což výhody skupinové minimalizace ruší.

Dynamický hazard

Při přechodu mezi sousedními vektory (fundamentální změně vstupu) může při přechodu místo jednoduché změny výstupní proměnné z $0 \rightarrow 1$ (resp. z $1 \rightarrow 0$) vzniknout posloupnost změn např. $0 \rightarrow 1 \rightarrow 0 \rightarrow 1$ (resp. $1 \rightarrow 0 \rightarrow 1 \rightarrow 0$) - vícenásobné zakmitání výstupní proměnné. Důvodem vzniku dynamického hazardu jsou v zásadě vzájemná časová zpoždění ve větvích kombinačního obvodu (s možným výskytem statického hazardu) a nebo větší počet větví realizovaného obvodu (minimálně 3 větve) s různým zpožděním průchodu signálu. Dynamický hazard vzniká zejména v obvodech realizovaných podle smíšené formy.

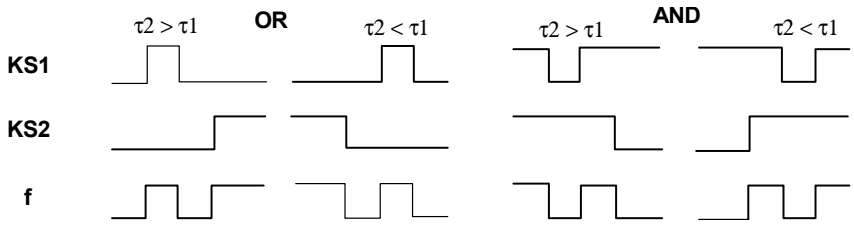
Dynamický hazard vzniká:

a) V kombinační síti se dvěma větvemi, kde jedna větev obsahuje statický hazard, který je časově posunut vůči řádné změně výstupní proměnné větve druhé. V zásadě platí, že ***Existuje-li statický hazard v části obvodu pak existuje i hazard dynamický***.
Na obr.2.30. je znázorněn příklad vzniku tohoto typu dynamického hazardu



Obr.2.30. Vznik dynamického hazardu při existenci statického

Záleží na typu výstupního členu, zpoždění jednotlivých větví a charakteru statického hazardu (v 1 nebo v 0) jak a jaký dynamický hazard vzniká.
Varianty vzniku dynamického hazardu pro příklad z obr.2.30 jsou uvedeny na obr 2.31.

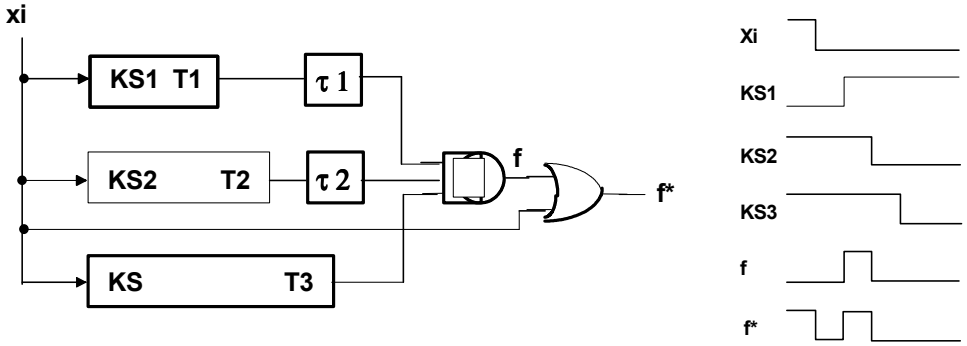


Obr.2.31. Varianty vzniku dynamického hazardu z obr.2.30.

Odstranění tohoto typu dynamického hazardu předpokládá odstranit veškeré statické hazardy kombinační logické sítě.

b) V kombinační síti bez statických hazardů s větším počtem paralelních větví (více než 3 paralelní větve), kde jednotlivé větve mají různá zpoždění průchodu signálu. Tento typ hazardu vzniká zejména u obvodů realizovaných ze smíšené formy. Příklad tohoto typu hazardu je uveden na Obr.2.32.

Zjištění dynamického hazardu a jeho odstranění:
Zjištění dynamického hazardu není jednoduché (nelze jej rozpoznat z mapy funkce). Jednou z možností je upravit smíšený tvar funkce na základní disjunktivní nebo konjunktivní tvar a nevynechávat výrazy typu $a \,.\, \bar{a}$ *nebo* $a + \bar{a}$. Pokud tyto výrazy existují, pak je možný vznik dynamického resp. statického hazardu. V zásadě při diskusi realizace logického systému musíme dbát na odstranění všech statických hazardů a všímat si délky zpoždění jednotlivých paralelních větví realizace a rozdíly v těchto zpožděních eliminovat.
Další možností je doplnění kombinačního systému o synchronizační signál, který vzorkuje ustálený stav obvodu do vyrovnávací paměti a tím odstraní jak statické tak i dynamické hazardy. Nevýhodou tohoto řešení je zpomalení odezvy takto realizovaného obvodu na asynchronní změnu vstupních vektorů, protože obvod nereaguje na změnu okamžitě, ale pouze v době existence synchronizačního signálu. Takovéto řešení je vhodné pro "čistě" synchronní systémy (včetně synchronní sekvenční části), což zjednoduší výslednou realizaci. Toto řešení však není vhodné pro realizaci asynchronních systémů.



Obr.2.32. Vznik hazardu při různém zpoždění větví (bez hazardu)

Závěrem:

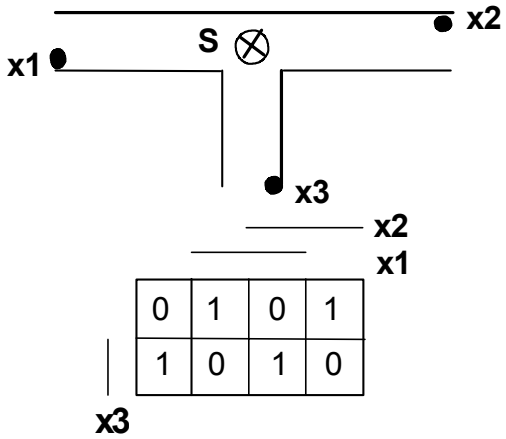
Při diskusi možnosti vzniku hazardů a jejich odstranění musíme vycházet z účelu, ke kterému kombinační logický obvod navrhujeme. Pokud logický obvod má sloužit zcela samostatně jako indikační, kontrolní nebo řídicí obvod, pak eventuelní vznik hazardů nemusí mít nežádoucí vliv na funkci (zobrazovací prvky se setrvačností oka obsluhy, pomalá reakce akčních členů a pod.). Hazardy však musíme odstranit v případě, kdy navrhovaná kombinační síť je součástí nebo navazuje na sekvenční síť, pro kterou jakékoli hazardy jsou interpretovány jako reálný vstup. V tomto případě neodstranění hazardu znamená změnu chování celého obvodu i při správném logickém návrhu.

2.2.7. Příklady

V následujícím budou uvedeny příklady návrhu typických kombinačních obvodů.

Př.1.
Navrhněte kombinační obvod pro zapínání a vypínání světla ze třech míst (tři vypinače). Světlo má svítit při sepnutí lichého počtu vypinačů.

x3	x2	x1	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



Obr.2.33. Rozsvícení světla ze třech míst

Z tabulky nebo z mapy (nelze minimalizovat) docházíme ke stejnému algebraickému vyjádření logické funkce:

$$S = x_1\bar{x}_2\bar{x}_3 + \bar{x}_1x_2\bar{x}_3 + \bar{x}_1\bar{x}_2x_3 + x_1x_2x_3$$

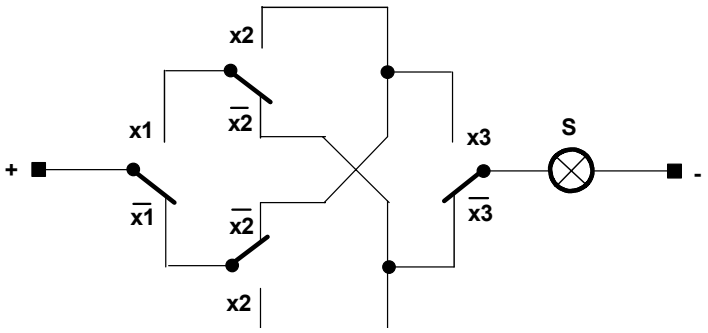
Po úpravě absolutně minimální forma

$$S = x_1(\bar{x}_2\bar{x}_3 + x_2x_3) + \bar{x}_1(x_2\bar{x}_3 + \bar{x}_2x_3)$$

nebo

$$S = x_1 .\ \overline{x_2 \oplus x_3} + \bar{x}_1 .\ x_2 \oplus x_3$$

Realizace kontaktními prvky je uvedena na Obr.2.34.

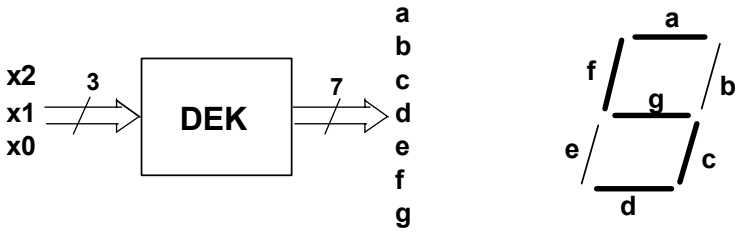


Obr.2.34. Zapínání a vypínání světla ze třech míst

K realizaci byly použity přepínací kontaktní členy a výsledné zapojení vzniklo dalším zjednodušením, které je dáno vlastností kontaktních členů (jsou průchozí oběma směry) a které není možné postihnout při úpravě výrazu. V tomto případě kromě vytýkání před závorku (přepínací člen x_1) můžeme některé kontakty vynechat jako přebytečné a získáme tak jakési "vytýkání před závorku" ale z druhé strany realizovaného obvodu (přepínací člen x_3).

Př.2.
Navrhněte dekodér ze tříbitového binárního kódu (x_2,x_1,x_0) na 7 výstupů (a až g) pro zobrazování číslic na sedmisegmentovém displeji.

Blokové schema a přiřazení výstupních proměnných jednotlivým segmentům displeje znázorňuje obr.2.35.

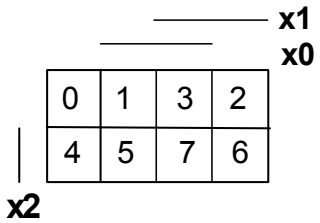


Obr.2.35. Zobrazení číslic

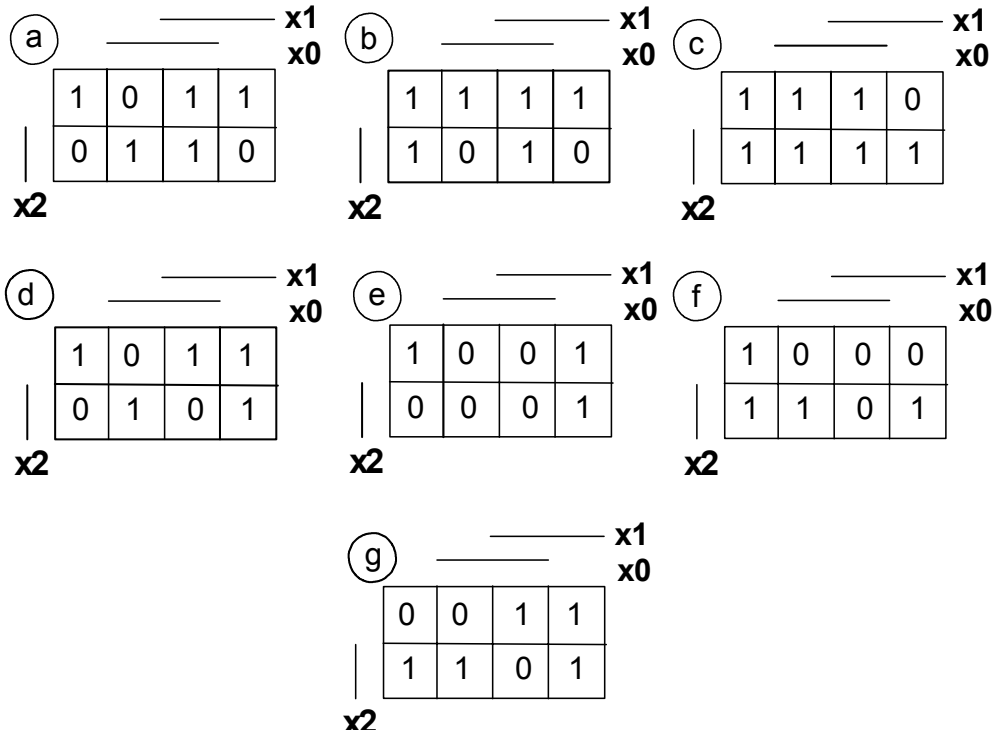
Ze zadání je zřejmé, že se jedná o obvod, který má tři vstupy (x_2,x_1,x_0) s vahami 4,2,1 a 7 výstupů (a,b,c,d,e,f,g) pro každý segment displeje.

Pravdivostní tabulka a mapy pro jednotlivé výstupní proměnné jsou uvedeny na obr.2.36.

x2	x1	x0	N	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	1	1	0	1	1	0	0	0	0
0	1	0	2	1	1	0	1	1	0	1
0	1	1	3	1	1	1	1	0	0	1
1	0	0	4	0	1	1	0	0	1	1
1	0	1	5	1	0	1	1	0	1	1
1	1	0	6	0	0	1	1	1	1	1
1	1	1	7	1	1	1	0	0	0	0



přiřazení číslic kódu



Obr.2.36. Dekodér 7-mi segmentového displeje

Z map na obr.2.36. minimalizací dostaneme výrazy pro výstupní proměnné:

$$a = \overline{\overline{x_1\bar{x}_2} \,.\, x_0 \oplus x_2} \qquad b = \overline{x_2 \,.\, x_0 \oplus x_1}$$

$$c = \overline{\bar{x}_2 \,.\, x_1 \,.\, \bar{x}_0} \qquad d = \overline{\overline{\bar{x}_2 x_1} \,.\, \overline{x_1\bar{x}_0} \,.\, \overline{\bar{x}_2\bar{x}_0} \,.\, \overline{x_0\bar{x}_1 x_2}}$$

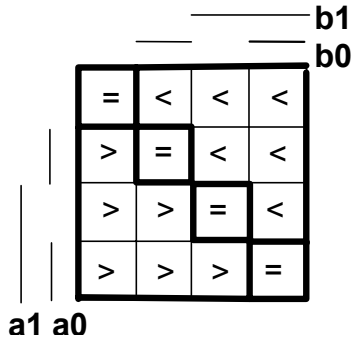
$$e = \overline{\overline{\bar{x}_0 x_1} \,.\, \overline{\bar{x}_0\bar{x}_2}} \qquad f = \overline{\overline{\bar{x}_1 x_2} \,.\, \overline{\bar{x}_0 x_2} \,.\, \overline{\bar{x}_0\bar{x}_1}}$$

$$g = \overline{\overline{\bar{x}_0 x_2} \,.\, \overline{x_1 \oplus x_2}}$$

Realizace těchto výrazů je evidentní. V tomto případě opět nemusíme odstraňovat hazardy, protože jejich existence nevadí funkci displeje ani uživateli.

Př.3.
Navrhněte aritmetický komparátor dvou dvoubitových číslic A (a_0,a_1) a B (b_0,b_1) kódovaných v binárním kódu.

V tomto případě vyjimečně použijeme binární mapu, v které oblasti aritmetické komparace budou jednoduše určitelné. Zadání příkladu můžeme rovnou vyznačit do mapy aniž použijeme pravdivostní tabulku viz obr.2.37.



Obr.2.37. Komparátor čísel A a B

Z mapy můžeme číst následující vztahy:

Pro A>B

$$[A > B] = a_1\bar{b}_1 + a_0\bar{b}_1\bar{b}_0 + a_1 a_0\bar{b}_0$$

Pro A<B

$$[A < B] = \bar{a}_1 b_1 + \bar{a}_0\bar{a}_1 b_0 + \bar{a}_0 b_0 b_1$$

Pro A=B můžeme použít doplněk do "1".

$$[A = B] = \overline{[A > B] + [A < B]}$$

Úprava na použité realizační prvky a vlastní realizace je evidentní. U tohoto obvodu je, s ohledem na jeho použití, potřeba odstranit hazardy. Statické hazardy jsou odstraněny provázáním minimalizačních smyček. Vzhledem k tomu, že se jedná o binární mapu je nezbytné při minimalizaci dát pozor na sousední (slučitelné) termy, které nejsou v mapě topologicky sousední.