1. ZÁKLADNÍ POJMY DIGITÁLNÍ TECHNIKY

Abstrakce v digitální technice:

signály se pokládají za skokově proměnné,

v nejjednodušším případě dvě možné hodnoty

logická jednička	log. **1**	**1**
logická nula	log. **0**	**0**

Popis pomocí *dvouhodnotových veličin:*

1. logická interpretace → **1**, **0**

2. pravdivostní interpretace → výrok pravdivý (**T**), nepravdivý (**F**)

3. interpretace formou binárních číslic 1, 0 (užívá zvlášť pro vícebitové skupiny)

4. interpretace vyjadřující aktivní (**1**) a neaktivní (**0**) stav určité řídicí veličiny;

5. další možnosti: např. kontaktová reprezentace → sepnuto (**1**), rozepnuto (**0**)

Nejčastěji interpretace logická,
v programových prostředcích interpretace formou binárních číslic.

Zobrazení dvouhodnotových veličin

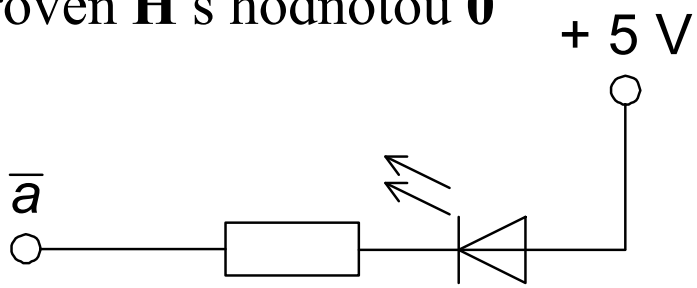
Zobrazením pomocí *úrovně* fyzikální veličiny (napětí, proudu)

⇨ úroveň **H** (vyšší hodnota),

⇨ úroveň **L** (nižší hodnota)

kladná logika pro úroveň **H** s hodnotou **1**,

záporná logika pro úroveň **H** s hodnotou **0**



Zobrazením pomocí *změny úrovně* fyzikální veličiny

vyznačení určitého okamžiku - např. pro zápis do registru, pro inkrementaci čítače apod.

aktivní hrana (vzestupná nebo sestupná)

Logické veličiny

logické konstanty (**0, 1**),

logické proměnné, které se označují pomocí identifikátorů.

Digitální systémy

systémy kombinační, u nichž hodnoty výstupních veličin závisí jen na okamžitém stavu vstupních veličin,

systémy sekvenční, kde hodnoty výstupních veličin závisí i na předchozích hodnotách vstupních veličin, obsahují paměť.

Číselné soustavy a kódy

Přirozené číslo N lze obecně vyjádřit základem B pomocí symbolů nebo číslic a_i

$$N = a_{n-1} \cdot B^{n-1} + a_{n-2} \cdot B^{n-2} + ... + a_1 \cdot B^1 + a_0 \cdot B^0$$

$$N = a_{n-1} a_{n-2} .... a_1 a_0$$

- základ 2 se symboly 0 a 1,
- základ 10 se symboly 0, 1, …, 8, 9,
- základ 8 (oktanový) se symboly 0, 1, …, 6, 7,
- základ 16 se symboly 0, 1, …, 8, 9, A, B, C, D, E, F.

Metoda postupného odečítání

Metoda se snadno použije pro přechod od základu B k základu 2.

Původní číslo se rozkládá odečítáním zmenšujících se mocnin základu, přičemž se hledá mocnina čísla 2 rovná převáděnému číslu nebo menší.

Příklad

	190	
$2^7 = 128$	$\underline{- 128}$	$\rightarrow 1$
	62	
$2^6 = 64$	příliš velké	$\rightarrow 0$
$2^5 = 32$	$\underline{- 32}$	$\rightarrow 1$
	30	
$2^4 = 16$	$\underline{- 16}$	$\rightarrow 1$
	14	

$2^3 = 8$ $- 8$ $\rightarrow 1$

6

$2^2 = 4$ $- 4$ $\rightarrow 1$

2

$2^1 = 2$ $- 2$ $\rightarrow 1$

0

$2^0 = 1$ $\rightarrow 0$

Výsledek $\mathbf{190_{10} = 10111110_2}$

Metoda postupného dělení

Celé číslo N se základem B se napíše v základu B

$$N_{B_2} = a_{n-1} \cdot B_2^{n-1} + a_{n-2} \cdot B_2^{n-2} + ... + a_1 \cdot B_2^2 + a_1 \cdot B_2^1 + a_0$$

Když vydělíme N základem B_2, dostaneme podíl Q_1 a zbytek

$$N_{B_2} = Q_1 \cdot B_2 + R_1$$

$$N_{B_2} = B_2 \cdot \left[a_{n-1} \cdot B_2^{n-2} + a_{n-2} \cdot B_2^{n-3} + ... + a_2 \cdot B_2^1 + a_1 \right] + a_0$$

Zbytek R_1 představuje koeficient a_0. Když vydělíme podíl Q_1 základem B_2, dostaneme

$$Q_1 = B_2 \cdot \left[a_{n-1} \cdot B_2^{n-3} + a_{n-2} \cdot B_2^{n-4} + ... + a_2 \right] + a_1$$

Zbytek R_2 představuje koeficient a_1. Dělit se může v libovolném základě, pokud můžeme v příslušném základě snadno počítat.

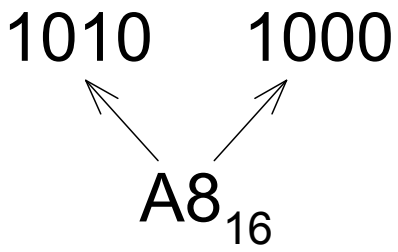
Příklad

$1358_{10} \rightarrow N_{16}$

1358	16			
078	84	16		
14	04	5	16	
		5	0	
R_1	R_2	R_3		

Odtud je číslo $1358_{10} = 54E_{16}$

Vzájemný převod z dvojkové do šestnáctkové soustavy: rozdělit dvojkové prvky do čtveřic, čtveřice váhový součet.

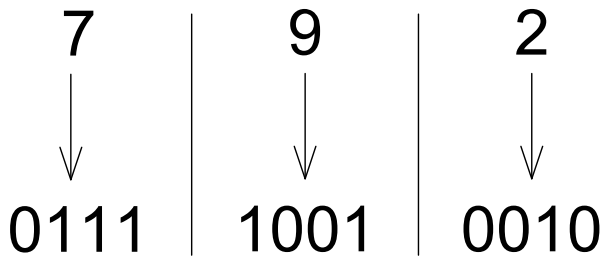


Zobrazení dvojkově kódované desítkové soustavy (BCD)

Převodu se lze vyhnout použitím dvojkově kódovaného desítkového zápisu (BCD), kdy se zapíše každý desítkový prvek pomocí svého 4-bitového dvojkového ekvivalentu

Příklad

zobrazit 792_{10} v BCD



Šest kombinací (10 až 15) nevyužito, BCD číslo zabírá větší místo než jeho dvojkový ekvivalent.

Použití: často při sčítání událostí a zobrazení počtů

čítač - čtyřbitové dekády, obsah každého čítače je přímo zobrazován – použití zobrazovacího obvodu např. 74247

Kód 1 z 10

Výhody:

dobrá čitelnost kódu,

možnost detekovat přenosové chyby díky vysoké redundanci,

menší náklady při kódování a dekódování

Nevýhody:

větší technické náklady způsobené redundancí

Použití převážně v telekomunikačním provozu

Grayův kód

Při přechodu od jedné desítkové číslice k další se kódové slovo mění jen v jediném bitu.

Využití zejména při při převodu analogových měřených veličin na digitální signály – potlačení chyb převodu

Tab. 2.1: Zobrazení jednotlivých kódů

desítkově	dvojkově	Grayův kód	1 z 10
0	0000	0000	0000000001
1	0001	0001	0000000010
2	0010	0011	0000000100
3	0011	0010	0000001000
4	0100	0110	0000010000
5	0101	0111	0000100000
6	0110	0101	0001000000
7	0111	0100	0010000000
8	1000	1100	0100000000
9	1001	1101	1000000000
10	1010	1111	
11	1011	1110	
12	1100	1010	
13	1101	1011	
14	1110	1001	
15	1111	1000	

Aritmetické operace

Aritmetické operace v soustavě o základu z se provádí stejným způsobem jako v soustavě desítkové. Je však nutné mít na paměti, že v soustavě se základem z dochází k přenosu jakmile výsledek převýší $z - 1$.

Příklad (dvojková soustava)

$0 + 0 = 0$	$0 - 0 = 0$	$0 \cdot 0 = 0$	$0 : 0 =$ nedefinováno
$0 + 1 = 1$	$1 - 0 = 1$	$0 \cdot 1 = 0$	$1 : 0 =$ nedefinováno
$1 + 0 = 1$	$1 - 1 = 0$	$1 \cdot 0 = 0$	$0 : 0 = 0$
$1 + 1 = 10$	$10 - 1 = 1$	$1 \cdot 0 = 1$	$1 : 1 = 1$

přenos do vyššího řádu nebo výpůjčka z vyššího řádu

násobení se prakticky převádí na sčítání a posun.

2. KOMBINAČNÍ LOGICKÉ FUNKCE

Kombinační logická funkce

- pravidlo přiřazující každé kombinaci hodnot **0** a **1** přiřazených vstupním proměnným z definičního oboru funkce jedinou hodnotu výstupní proměnné.

Úplně určená kombinační logická funkce

taková funkce, jejíž definiční obor zahrnuje všechny kombinace vstupních proměnných.

Neúplně určené kombinační logická funkce

její definiční obor nezahrnuje některé tyto kombinace.

Tab. 3.: Kombinační logické funkce jedné vstupní proměnné

hodnoty vstupní proměnné x	0	1	zápis funkce:	název funkce:
odpovídající funkční hodnoty $y = f(x)$	0	0	$y = f_0(x) = \mathbf{0}$	nulová funkce
	0	1	$y = f_1(x) = x$	totožnost, opakování
	1	0	$y = f_2(x) = \bar{x}$	negace, inverze
	1	1	$y = f_3(x) = \mathbf{1}$	jednotková funkce

Určení kombinačních logických funkcí *n* proměnných

Sestavování tabulky pro *n* proměnných:

1. do *n* řádků nad sebou vypíšeme možné hodnoty vstupních proměnných tak, aby v jednotlivých sloupcích vytvořily všechny možné kombinace hodnot těchto proměnných - např. tak, že tyto sloupce budou představovat *n*-bitová binární čísla odpovídající pořadí každého sloupce, počet těchto kombinací je $m = 2^n$

2. pod těmito řádky představujícími vstupní proměnné vytvoříme řádky odpovídající funkčním hodnotám jednotlivých funkcí tak, že do těchto řádků vypíšeme všechny možné kombinace *m* funkčních hodnot, těchto řádků a tedy možných funkcí je 2^m

3. celkem je počet možných funkcí *n* proměnných 2^{2^n}

Počet úplně určených kombinačních logických funkcí dvou proměnných je tedy 16.

Nejdůležitější kombinační logické funkce dvou proměnných

$y = a \cdot b$	log. součin, konjunkce, AND
$y = \overline{a \cdot b}$	funkce NAND
$y = a + b$	log. součet, disjunkce, OR
$y = \overline{a + b}$	funkce NOR
$y = a \oplus b = a \,.\, \overline{b} + \overline{a} \,.\, b$	nonekvivalence, funkce EX-OR, exklusivní součet
$y = \overline{(a \oplus b)} = a \,.\, b + \overline{a} \,.\, \overline{b}$	ekvivalence

Tab. 2.2: Kombinační logické funkce dvou vstupních proměnných

hodnoty vstupních proměnných x_1, x_2	x_1	**0 0 1 1**	**zápis funkce výrazem**	**název funkce**
	x_2	**0 1 0 1**		
odpovídající funkční hodnoty $y = f(x_1, x_2)$:		**0 0 0 0**	$y = f_0(x_1, x_2) = \mathbf{0}$	nulová funkce
		0 0 0 1	$y = f_1(x_1, x_2) = x_1 \,.\, x_2$	logický součin
		0 0 1 0	$y = f_2(x_1, x_2) = x_1 \,.\, \overline{x_2}$	
		0 0 1 1	$y = f_3(x_1, x_2) = x_1$	opakování x_1
			atd.	

Počet logických funkcí velmi rychle roste s ***počtem vstupních proměnných***.
Logické funkce 1 proměnné - stačí inverze,
logické funkce 2 proměnných - logický součet a součin,
logické funkce většího počtu proměnných - další složitější základní logické funkce nebo použít několika elementárních logických funkcí = soubor takových funkcí se nazývá ***úplný soubor logických funkcí***.

Úplný soubor logických funkcí:

1. NAND - touto jedinou funkcí můžeme vyjádřit všechny KLF libovolného počtu proměnných;

2. NOR - platí pro ni totéž co pro funkci NAND;

3. úplnými soubory funkcí jsou i takové soubory, jimiž lze výše uvedené funkce vyjádřit, tedy například funkce **OR spolu s inverzí**, funkce **AND spolu s inverzí** a další.

Booleova algebra

V Booleově algebře se používají logické reprezentace dvouhodnotových veličin - logických proměnných. Základní zákony této algebry mají podobný tvar jako mají zákony běžné algebry.

$$a + a = a, \qquad a \,.\, a = a, \qquad a + \bar{a} = \mathbf{1}, \qquad a \,.\, \bar{a} = \mathbf{0};$$

$$a \,.\, (b + c) = a \,.\, b + a \,.\, c;$$

$$a + (b \,.\, c) = (a + b) \,.\, (a + c);$$

$$a + (\bar{a} \,.\, b) = a + b;$$

$$a \,.\, b = \overline{(\bar{a} + \bar{b})} \quad , \quad a + b = \overline{(\bar{a} \,.\, \bar{b})}$$ *de Morganova pravidla*

Hodnota logického výrazu s operátory logického součtu a logického součinu se nezmění, jestliže vzájemně tyto operátory zaměníme (tj. operátory logického součtu nahradíme operátory logického součinu a naopak), invertujeme proměnné a výsledek.

2.1 Způsoby zápisu a zobrazení kombinačních logických funkcí

Abychom mohli s kombinačními logickými funkcemi pracovat, musíme je nejprve zapsat či zobrazit.

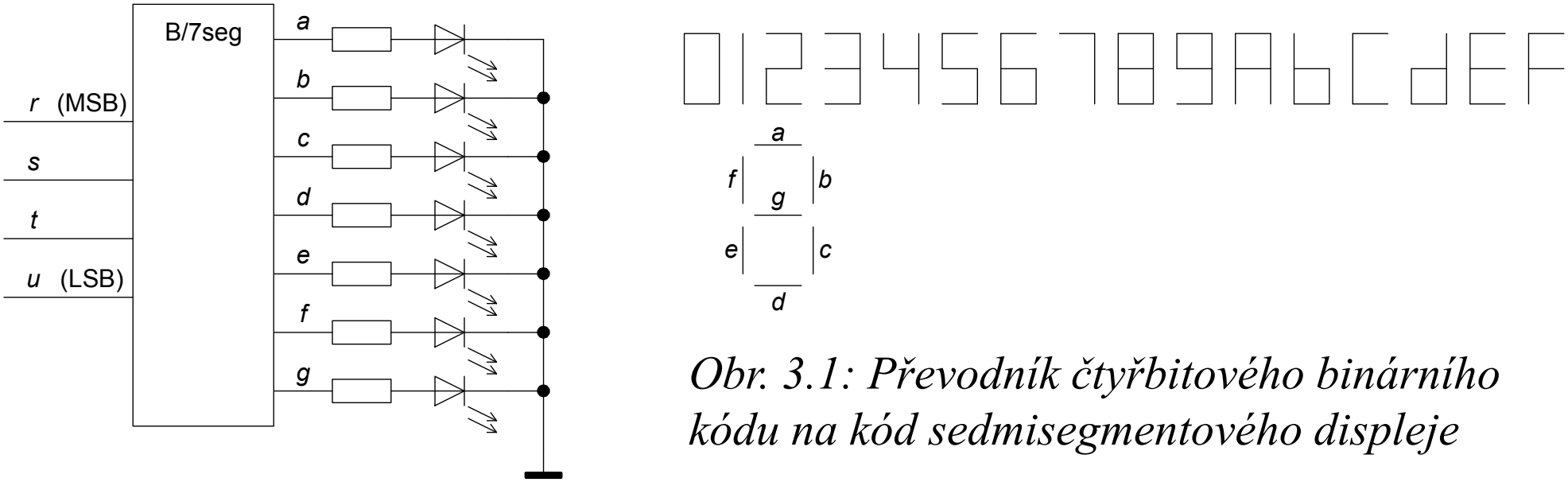
Nejčastěji se používají tyto způsoby zápisu, popř. zobrazení kombinačních logických funkcí:

- zápis pomocí pravdivostní tabulky,
- zápis logickým výrazem,
- zobrazení pomocí mapy,
- zobrazení pomocí logického schématu.

2.1.1 Zápis kombinační logické funkce pravdivostní tabulkou

Jako příklad tohoto způsobu zápisu uvedeme popis převodníku čtyřbitového binárního kódu na kód sedmisegmentového displeje s hexadecimálním zobrazením. Náčrt zapojení a zobrazované znaky (hexadecimální číslice) jsou na obrázku.

Vstupní proměnné **MSB** (Most Significant Bit) a **LSB** (Least Significant Bit) označují nejvýznamnější a nejméně významný bit.



Obr. 3.1: Převodník čtyřbitového binárního kódu na kód sedmisegmentového displeje

Převodník čtyřbitového binárního kódu na kód sedmisegmentového displeje, při hodnotě **1** proměnných ***a*** až ***g*** odpovídající segmenty svítí.

Tab. 2.3.: Pravdivostní tabulka převodníku

číslo (stavový index)	číslo (hex)	vstupy				výstupy						
		r	s	t	u	a	b	c	d	e	f	g
0	0	0	0	0	0	1	1	1	1	1	1	0
1	1	0	0	0	1	0	1	1	0	0	0	0
2	2	0	0	1	0	1	1	0	1	1	0	1
3	3	0	0	1	1	1	1	1	1	0	0	1
4	4	0	1	0	0	0	1	1	0	0	1	1
5	5	0	1	0	1	1	0	1	1	0	1	1
6	6	0	1	1	0	1	0	1	1	1	1	1
7	7	0	1	1	1	1	1	1	0	0	0	0
8	8	1	0	0	0	1	1	1	1	1	1	1
9	9	1	0	0	1	1	1	1	1	0	1	1
10	A	1	0	1	0	1	1	1	0	1	1	1
11	B	1	0	1	1	0	0	1	1	1	1	1
12	C	1	1	0	0	1	0	0	1	1	1	0
13	D	1	1	0	1	0	1	1	1	1	0	1
14	E	1	1	1	0	1	0	0	1	1	1	1
15	F	1	1	1	1	1	0	0	0	1	1	1

2.1.2 Zápis kombinační logické funkce logickým výrazem

Logický výraz - zápis skupiny identifikátorů logických proměnných vzájemně oddělených logickými operátory, přičemž se pro vyjádření pořadí provádění operací v případě potřeby používají závorky.
Nejpoužívanější operátory pro základní logické operace - logický součet, součin, inverze, fukce EX-OR, existují i další operátory pro jiné operace a s alternativními symboly operátorů pro uvedené logické funkce.

Zvláštní typy logických výrazů:

součinový term - obsahuje jen operátory logického součinu (nazývaný též implikant, konjunkce),
součtový term - obsahuje jen operátory logického součtu (inhibent, disjunkce),
minterm - součinový term obsahující všechny vstupní proměnné (které mohou být přítomny v přímém nebo v inverzním tvaru),
maxterm - součtový term obsahující podobně všechny vstupní proměnné,
úplný term - minterm nebo maxterm.

Z de Morganových pravidel plyne:
součtový term sestavený z určité kombinace vstupních proměnných je roven inverzi součinového termu sestaveného z týchž proměnných, které mají opačné znaky inverze, tj. proměnná obsažená v součtovém termu bez inverze je v odpovídajícím součinovém termu invertovaná a naopak.

Z definice vyplývá, že logická funkce představovaná **mintermem** má nulovou hodnotu pro všechny kombinace vstupních proměnných s výjimkou jediné, u níž jsou vstupní proměnné uvedené v zápisu mintermu s inverzí nulové a proměnné uvedené v tomto zápisu bez inverze jsou rovny **1**. Vzhledem k tomu, že při interpretaci zápisu hodnot vstupních proměnných formou binárních číslic představuje číslo vzniklé tímto způsobem hodnotu stavového indexu *s*, budeme značit příslušný minterm symbolem $\boldsymbol{k_s}$.

Podobně funkce představovaná **maxtermem** má hodnotu rovnou **1** pro všechny kombinace vstupních proměnných s výjimkou té, pro niž je přiřazení hodnot proměnných opačné než bylo uvedeno u mintermu. Tedy proměnná je nulová, je-li v zápisu maxtermu uvedena bez inverze, a má hodnotu **1** v opačném případě. Tento maxterm budeme značit symbolem $\boldsymbol{d_s}$.

Zápis **kombinační logické funkce** - různé způsoby a s použitím různých operátorů.
Dva základní způsoby zápisu funkce:

- zápis výrazem typu *součet součinů* (Sum of Products, SOP),
- zápis výrazem typu *součin součtů* (Product of Sums, POS).

Nejpoužívanější operátory pro základní logické operace - logický součet, součin, inverze, fukce EX-OR, existují i další operátory pro jiné operace a s alternativními symboly operátorů pro uvedené logické funkce.

***Součet součinů* (Sum of Products, SOP)**
Pro úplné termy (= mintermy) - *úplný součtový tvar* zápisu
Pro některé neúplné termy - *zkrácený (zjednodušený) součtový tvar* zápisu.

***Součin součtů* (Product of Sums, POS)**
Pro úplné termy (= maxntermy) - *úplný součinový tvar* zápisu
Pro některé neúplné termy - *zkrácený (zjednodušený) součinový tvar* zápisu.

realizace kombinační logické funkce $\Rightarrow$ *minimální tvary* zápisu

Lze snadno ukázat, že je-li počet vstupních proměnných n, je počet mintermů a maxtermů z těchto proměnných vytvořených právě $N = 2^n$.

Vyjádření kombinační logické funkce f(x_n, ..., x_1) v úplném tvaru součtu součinů:

$$\mathbf{f}(x_n, ..., x_1) = f_0 \,.\, k_0 + f_1 \,.\, k_1 + ... + f_{N-1} \,.\, k_{N-1}$$

Vyjádření funkce f(x_n, ..., x_1) v úplném tvaru součinu součtů:

$$\mathbf{f}(x_n, ..., x_1) = (f_0 + d_0) \,.\, (f_1 + d_1) \,.\, ... \,.\, (f_{N-1} + d_{N-1})$$

Použijeme-li k realizaci například číslicové integrované obvody typu **NAND** nebo **NOR**, pokládáme obvykle za minimální takový zápis typu součtu součinů nebo součinu součtů, který vyžaduje co nejmenší počet potřebných vývodů použitých obvodů, což zhruba odpovídá co nejmenšímu počtu symbolů vstupních proměnných použitých v zápisu funkce.

Zápis funkce v úplném součtovém a součinovém tvaru je *jednoznačný*.
Minimálních tvarů však může být pro určitou funkci více.

Někdy může být potřebné doplnit zkrácený tvar zápisu logické funkce na úplný tvar.

Bývá to například při realizaci funkcí pomocí **multiplexorů**. Úpravu je možno provést tak, že se členy, které neobsahují některé proměnné, doplní činiteli typu , kde *a* je proměnná chybějící v členu.

Příklad:

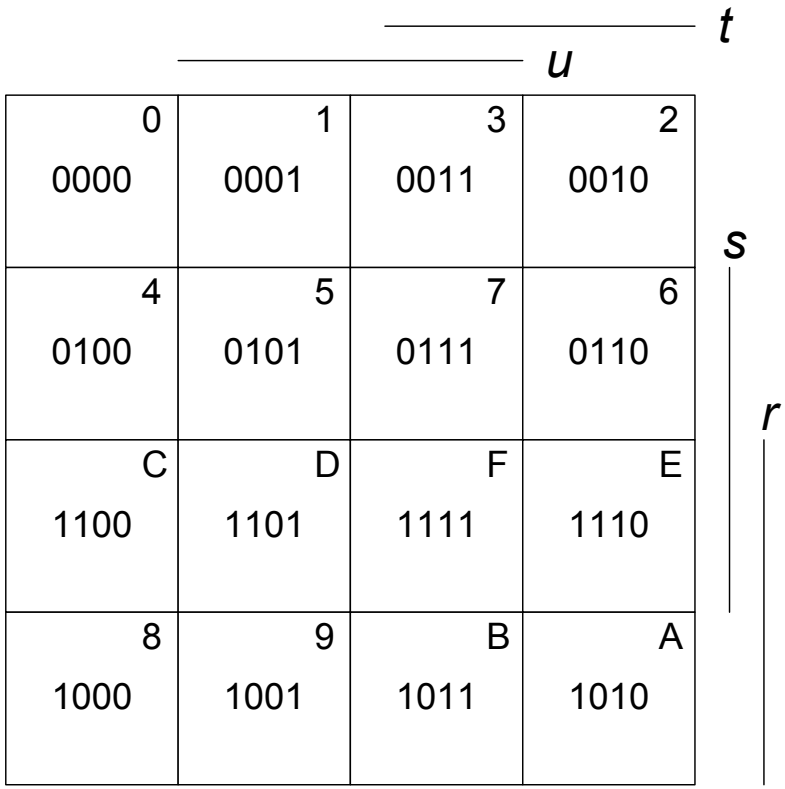
$$a \cdot b \cdot c + \overline{b} \cdot c = a \cdot b \cdot c + \left(a + \overline{a}\right) \cdot \overline{b} \cdot c = a \cdot b \cdot c + a \cdot \overline{b} \cdot c + \overline{a} \cdot \overline{b} \cdot c$$

2.1.3 Zobrazení kombinační logické funkce pomocí mapy

Karnaughova mapa - upravený způsob zápisu pravdivostní tabulky

buňky mapy = řádky tabulky

stavové indexy sousedních buněk se v binární soustavě liší vždy v hodnotě jedné vstupní proměnné



Obr. 2.2: Karnaughova mapa pro čtyři vstupní proměnné

2.1.4 Zobrazení kombinační logické funkce logickými schématy s kombinačními logickými členy

Zápis logické funkce pomocí logického výrazu můžeme snadno převést do grafického tvaru - vstupní a výstupní proměnné naznačíme ve formě vstupních a výstupních signálů logického schématu.

Operace prováděné s proměnnými znázorníme pomocí grafických značek **- logických členů**.

Většinou se používají značky představující jeden druh logické operace – např.:

NAND

NOR

EX-OR

AND-OR-INVERT apod.

2.2 Zjednodušování zápisu kombinačních logických funkcí

Realizace logických funkcí - například pomocí digitálních integrovaných obvodů řady 74 - obvykle vycházíme z minimálního tvaru zápisu funkce, který získáme z jiných tvarů *zjednodušením (minimalizací)*.

Zjednodušování
algebraické úpravy,
Karnaughovy mapy,
počítačové metody (např. Quineho a McCluskeyho - převod metody Karnaughovy mapy do algoritmického vyjádření).

Minimalizace úplně určených funkcí
Při zjednodušování pomocí algebraických úprav využíváme nejčastěji vztahu

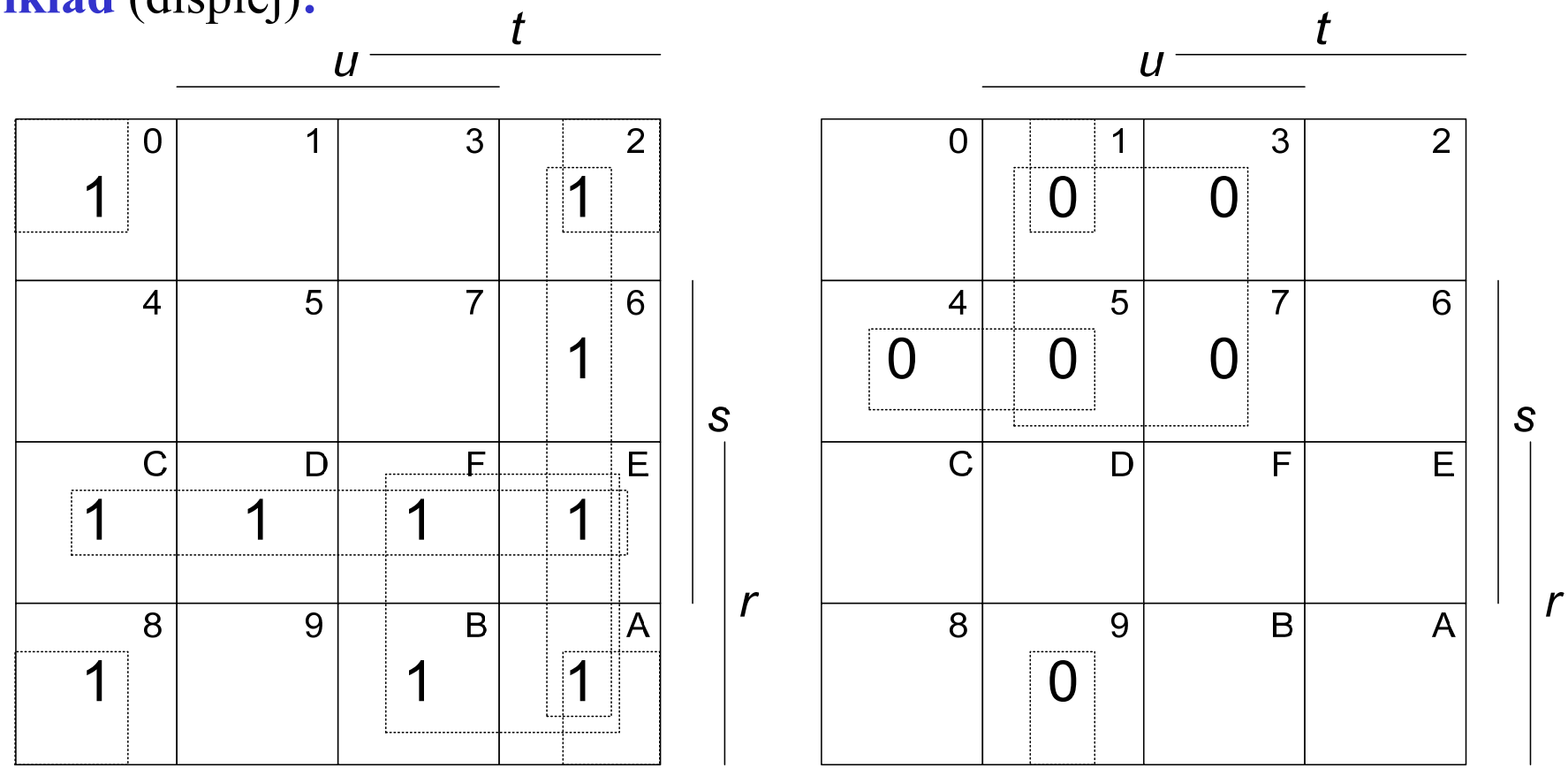
$$a + \overline{a} = 1$$

Obsahuje-li logická funkce zapsaná v součtovém tvaru dva termy, které se vzájemně liší jen v jedné proměnné, je možno zbývající proměnné z jejich součtu vytknout

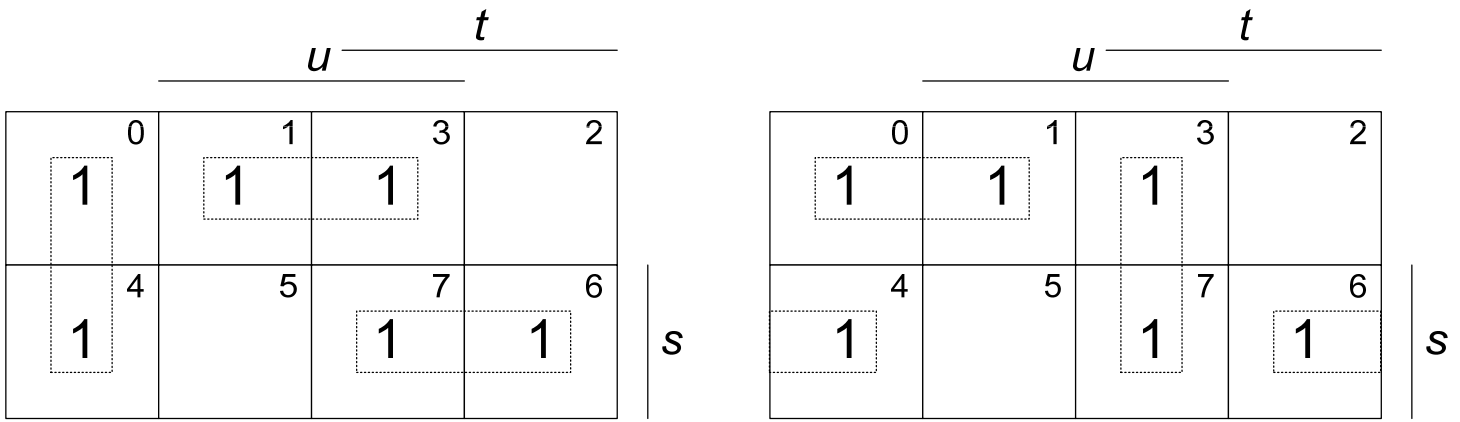
Realizace logických funkcí - například pomocí digitálních integrovaných obvodů řady 74 - obvykle vycházíme z minimálního tvaru zápisu funkce, který získáme z jiných tvarů *zjednodušením (minimalizací)*.

Příklad: $rst\overline{u} + rstu = rst.(\overline{u} + u) = rst$

Příklad (displej)**:**



Dvojí výběr:

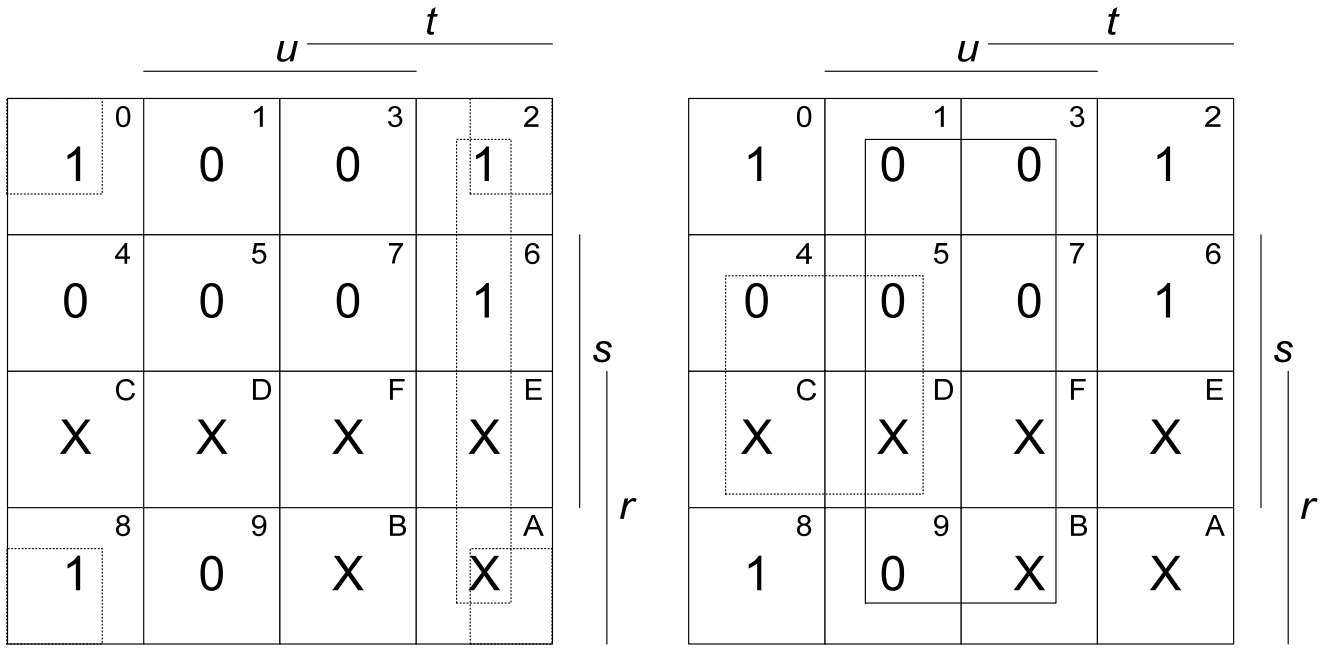


$$y = \bar{t} \cdot \bar{u} + \bar{s} \cdot u + s \cdot t \qquad y = \bar{s} \cdot \bar{t} + t \cdot u + s \cdot \bar{u}$$

Obr. 2.5: Funkce se dvěma možnými minimálními součtovými tvary

Minimalizace neúplně určených funkcí

Pravdivostní tabulka neúplně určené funkce neobsahuje všechny řádky, které má tabulka úplně určené funkce se stejným počtem proměnných. Tedy pro některé kombinace vstupních proměnných není hodnota funkce definována. Pro tyto kombinace můžeme hodnotu funkce definovat dodatečně tak, aby vyjádření funkce bylo co nejjednodušší.



$$e = \overline{s} \cdot \overline{u} + t \cdot \overline{u} \qquad e = \overline{u} \cdot \left(\overline{s} + t\right)$$

Obr. 2.6: Minimalizace funkce e s využitím neúplnosti její definice

3. REALIZACE KOMBINAČNÍCH LOGICKÝCH FUNKCÍ

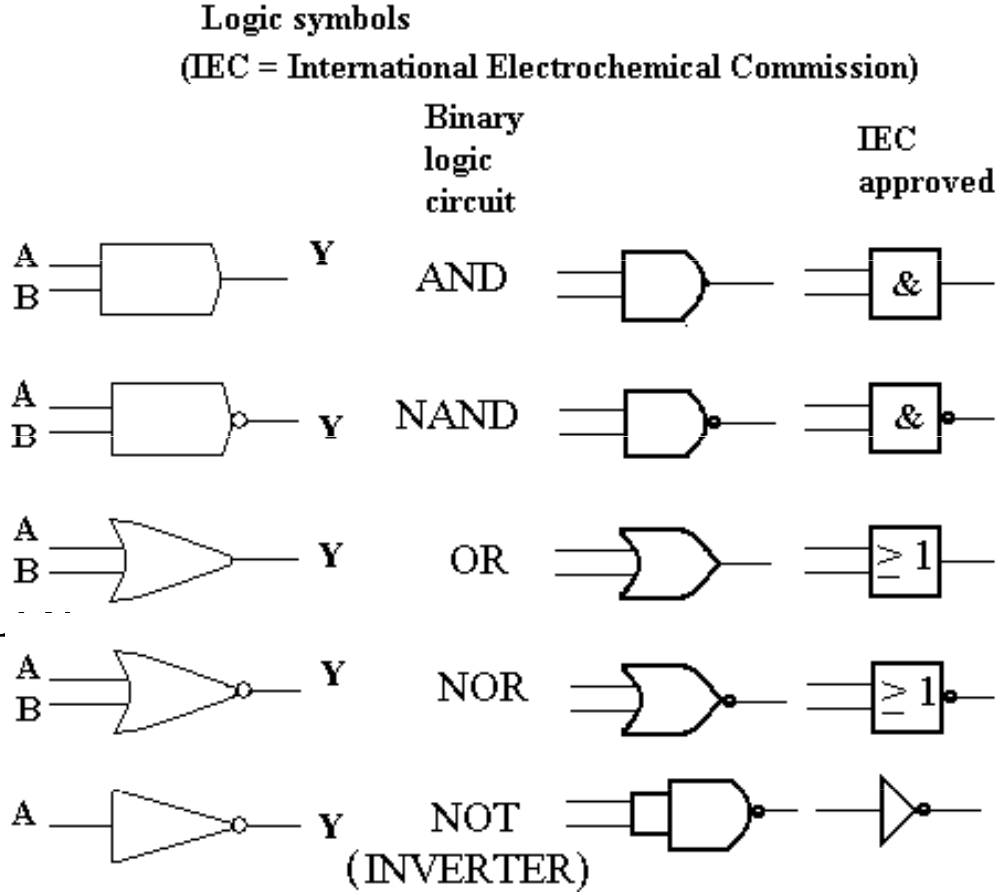
Realizace kombinační logické funkce = sestavení zapojení obvodu, který ze vstupních proměnných vytvoří výstupní proměnné v souhlasu se zadanou logickou funkcí.

použití moderních **mikroelektronických součástek** - často jediný IO (katalog nebo PROM nebo PLD)

základní způsob realizace kombinační logické funkce = pomocí kombinačních logických obvodů představujících realizaci základních logických členů v integrované podobě, kdy se vychází ze zápisu logické funkce v některém z výše uvedených tvarů součtu součinů nebo součinu součtů.

Nejčastěji **realizace kombinační logické funkce** pomocí digitálních integrovaných obvodů:

NAND,
NOR,
popřípadě AND, OR,
např. AND-OR-INVERT, EX-OR,
multiplexery a demultiplexery,
speciální kombinační obvody
čítačky, násobičky,
multiplexery, demultiplexery ad.,
programovatelné logické obvody (PL

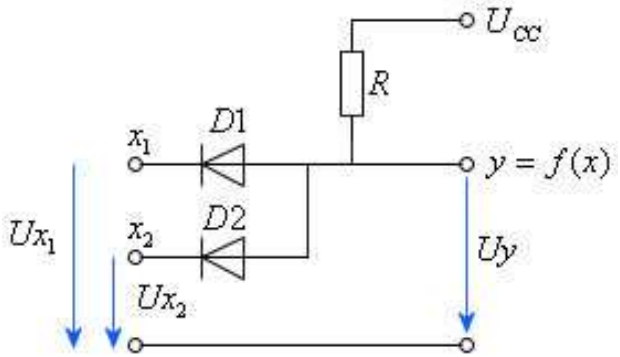


Zvláštní případy:

tranzistory,
diody apod.

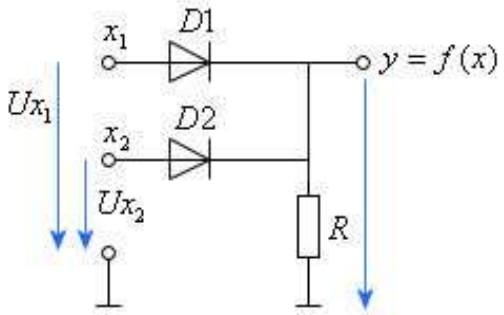
Diodová logika.
Je to vlastně funkční zapojení, které se skládá z diod a rezistorů, a umožňuje nám z něj vytvářet dvě základní hradla AND a OR a z nich potom skládat složitější zapojení. Diodová logika je poměrně rychlá – až 200 MHz.

Prvním z nich bude logický součin.
Zajímat nás budou samozřejmě i logické jedničky a nuly, ale především to tentokráte budou napětí.



Ux_2	Ux_1	Uy	f(x)
0V	0V	0,7V	0
0V	U_{cc}	0,7V	0
U_{cc}	0V	0,7V	0
U_{cc}	U_{cc}	U_{cc}	1

Na obrázku si můžete prohlédnout zapojení logického hradla AND složeného ze dvou diod a jednoho odporu.

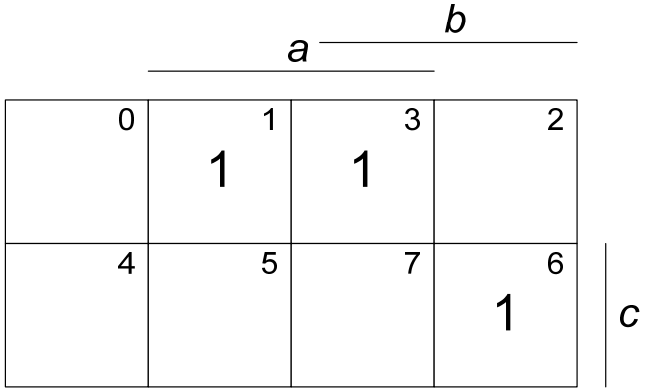


Ux_1	Ux_2	Uy	$f(x)$
0V	0V	0V	0
0V	10V	9,3V	1
10V	0V	9,3V	1
10V	10V	9,3V	1

3.1 Realizace kombinační logické funkce základními kombinačními digitálními obvody

3.1.1 Realizace kombinační logické funkce součinovými a součtovými obvody

Při realizaci - zápis funkce v součtovém nebo součinovém tvaru
výhodné použít logické členy téhož typu, tj. buď součinové nebo součtové



$$y = a \cdot \overline{c} + \overline{a} \cdot b \cdot c$$

převod s využitím de Morganových pravidel

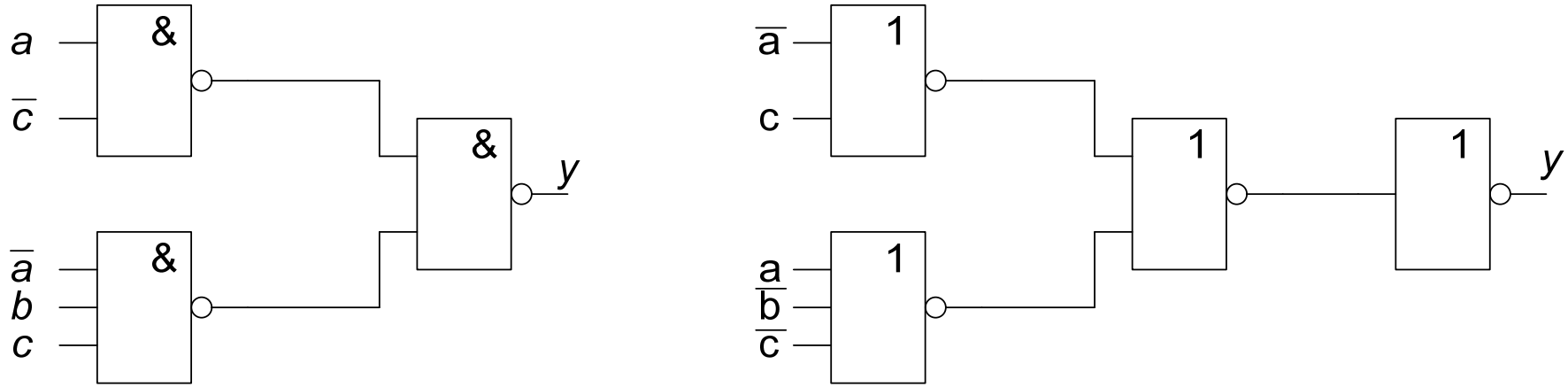
Obr. 3.1: Mapa realizované funkce y

Zápis v součtovém tvaru:

$$y = a \cdot \overline{c} + \overline{a} \cdot b \cdot c$$

$$y = \overline{\left(\overline{(a \cdot \overline{c})} \cdot \overline{(\overline{a} \cdot b \cdot c)}\right)}$$

$$y = \overline{(\overline{a} + c)} + \overline{(a + \overline{b} + \overline{c})}$$

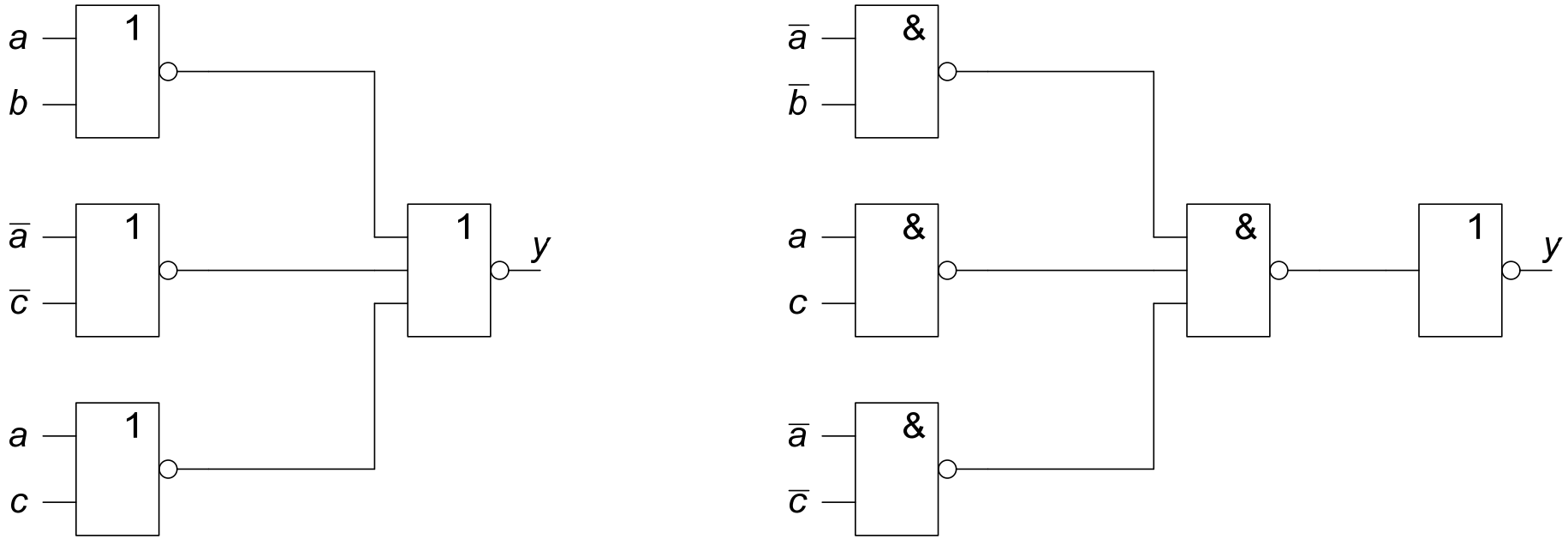


Obr. 3.2: Realizace funkce y na základě součtového tvaru zápisu

Zápis v součinovém tvaru:

$$y = (a + b) \cdot (\overline{a} + \overline{c}) \cdot (a + c)$$

$$y = \overline{\overline{(a+b)} + \overline{(\overline{a}+\overline{c})} + \overline{(a+c)}} \qquad y = \overline{(\overline{a} \cdot \overline{b})} \cdot \overline{(a \cdot c)} \cdot \overline{(\overline{a} \cdot \overline{c})}$$



Obr. 3.2: Realizace funkce y na základě součinového tvaru zápisu

V případě, že se má realizovat současně několik funkcí těchže proměnných, je někdy možné využít termy vytvořené v prvním stupni pro více funkcí, pokud tyto funkce obsahují společné termy. Mluvíme pak o skupinové minimalizaci několika funkcí těchže proměnných.

Obecnější závěr:

1.vycházíme-li při realizaci ze součtového tvaru zápisu, je funkce realizována strukturou NAND-NAND nebo NOR-OR,

2.vyjdeme-li ze součinového tvaru, dostaneme strukturu NOR-NOR nebo NAND-AND,

3.obě zapojení vycházející z téhož tvaru jsou topologicky stejná, tj. obsahují stejný počet logických členů, které mají stejný počet vstupů (de Morgan)

4.minimalizace počtu součástek,

5.složitější funkce například pomocí programovatelných logických obvodů nebo pamětí PROM, EPROM a EEPROM.

V předcházející úvaze jsme předpokládali, že máme k dispozici **přímé i invertované hodnoty vstupních proměnných**

v praxi musíme invertované vstupní signály vytvořit z přímých signálů pomocí **invertorů -** zvýšení počtu součástek

Existují metody minimalizace, jejichž pomocí lze najít minimální tvar funkce i v tomto případě:
např. **metoda minimalizace struktury TANT**
Three-stage And-Not structure with True inputs
invertory k vytvoření invertovaných vstupních proměnných struktury NAND-NAND se považují za třetí stupeň struktury a uvedenou metodou se vytvoří struktura, která v tomto stupni může obsahovat místo invertorů členy NAND, což může přinést zjednodušení

podobně **metoda minimalizace struktury TONT**
Three-stage Or-Not structure with True inputs
optimální struktura analogická struktuře NOR-NOR

3.1.2 Realizace kombinační logické funkce pomocí členů AND-OR-INVERT

vyrábějí se v několika provedeních – liší se počtem součinových sekcí a počtem vstupů v těchto sekcích

použití výhodné tam, kde má realizovaná funkce tvar odpovídající těmto počtům

např. obvod **74..51** je vhodný pro realizaci funkcí, které při vyjádření v součinovém tvaru obsahují dva součtové termy o dvou proměnných

+++ $$y = \overline{(a \cdot b + c \cdot d)} = \overline{(a \cdot b)} \cdot \overline{(c \cdot d)} = (\bar{a} + \bar{b}) \cdot (\bar{c} + \bar{d})$$

signál ze součinové části postupuje do součtové části uvnitř pouzdra, takže je zde menší zpoždění,

zjednoduší se topologie spojů na desce,

ušetřené vývody lze použít pro další vstupy

---

má-li se však realizovat více funkcí těchže proměnných, nelze užít dílčí součiny z prvního stupně pro několik funkcí.

3.2 Použití multiplexerů a demultiplexerů k realizaci kombinačních logických funkcí

Multiplexer

několik datových vstupů 2^n pro n datových vstupů,

jeden výstup (popř. dva komplementární výstupy),

adresové vstupy (A0 až A_{n-1}) pro binárně zakódovanou adresu,

často výběrový vstup S (select) – hradlování procházejících signálů (používá se např. pro sestavování větších multiplexerů z několika menších)

multiplexer s 8 datovými vstupy, 3 adresovými vstupy, 1 výstupem, 1 výběrem:

$$y = s \,.\, (k_0 \,.\, i_0 + k_1 \,.\, i_1 + \ldots + k_7 \,.\, i_7)$$

$$k_0 = \overline{a}_2.\overline{a}_1.\overline{a}_0 \qquad k_1 = \overline{a}_2.\overline{a}_1.a_0 \qquad \ldots\ldots \qquad k_7 = a_2.a_1.a_0$$

dříve byl uveden zápis kombinační logické funkce v úplném součtovém tvaru pro 3 vstupní proměnné x_3, x_2, x_1

$$f(x_3, x_2, x_1) = f_0 \,.\, k_0 + f_1 \,.\, k_1 + \ldots + f_7 \,.\, k_7,$$

kde $k_0, k_1, \ldots, k_7$ jsou mintermy složené z proměnných x_3, x_2, x_1

formálně shodné substituce:

$a_i = x_i$ (součiny těchto proměnných představují mintermy k_0 až k_7)

$i_i = f_i$

$\Rightarrow$ pomocí multiplexeru můžeme tedy realizovat jakoukoliv funkci 3 proměnných $f(x_3, x_2, x_1)$, přivedeme-li na jeho vstupy I0, I1, …, I7 signály s hodnotami $f_0, f_1, \ldots, f_7$ (vstupy uzemníme nebo připojíme přes vhodný rezistor ke zdroji napájecího napětí podle toho, je-li příslušná hodnota funkce nulová nebo jedničková)

$\Rightarrow$ z vyjádření funkce jsou tak proměnné x_3, x_2, x_1 eliminovány

podobně postup při eliminaci některých proměnných u funkcí s větším počtem proměnných

např. pro *n*>3:

1.zapíšeme funkci f(x_n, ..., x_1) v úplném součtovém tvaru (po doplnění),

2.seřadíme mintermy do skupin tak, aby v každé skupině byly proměnné x_3, x_2, x_1 ve stejném vyjádření co do přímého nebo inverzního tvaru,

3.tyto proměnné pak vytkneme,

4.pak zápis funkce je ve tvaru f(x_n, ..., x_1) = f_0 . k_0 + f_1 . k_1 + ... + f_7 . k_7,

5.přitom f_0, f_1, ..., f_7 jsou tzv. *zbytkové funkce* proměnných x_n, ..., x_4 (neobsahují již x_3, x_2, x_1)

6.použijeme multiplexer - na jeho vstupy I0, I1, …, I7 přivedeme signály s hodnotami zbytkových funkcí f_0, f_1, ..., f_7 a na jeho adresové vstupy přivedeme proměnné x_3, x_2, x_1

při použití multiplexeru se 3 adresovými vstupy je tento způsob výhodný zejména pro realizaci funkcí 4 proměnných (obecně u multiplexeru s *n* adresovými vstupy pro realizaci funkcí *n* + 1 proměnných), protože pak bude na každém ze vstupů I0, I1, …, I7 multiplexeru některá z těchto hodnot:
zbývající proměnná, její inverze, 1, 0

Příklad:

realizace funkce *e* pro odpovídající segment převodníku kódu BCD na kód sedmisegmentového displeje – použijeme dříve odvozený z minimální tvar

$$e = r \cdot s + t \cdot \overline{u} + \overline{s} \cdot \overline{u}$$

⇒ eliminujme odtud pomocí adresových vstupů multiplexeru např. proměnné *s*, *t* a *u*

⇒ výraz pro funkci *e* doplníme tak, aby každý sčítanec obsahoval všechny eliminované proměnné

$$e = r \cdot s \cdot (t + \overline{t}) \cdot (u + \overline{u}) + r \cdot (s + \overline{s}) \cdot t \cdot (u + \overline{u}) + (s + \overline{s}) \cdot t \cdot \overline{u} + \overline{s} \cdot (t + \overline{t}) \cdot \overline{u}$$

⇒po roznásobení dostaneme výraz, v němž sčítance seskupíme tak, aby bylo možno vytknout součiny eliminovaných proměnných

⇒ toto seskupení nejsnáze formou *tabulky zbytkových funkcí*

Tab. 3.1:Tabulka zbytkových funkcí

$\bar{s}\cdot\bar{t}\cdot\bar{u}$	$\bar{s}\cdot\bar{t}\cdot u$	$\bar{s}\cdot t\cdot\bar{u}$	$\bar{s}\cdot t\cdot u$	$s\cdot\bar{t}\cdot\bar{u}$	$s\cdot\bar{t}\cdot u$	$s\cdot t\cdot\bar{u}$	$s\cdot t\cdot u$
1	**0**	$r+1+1=1$	r	r	r	$r+r+1=1$	$r+r=r$

$$e=\bar{s}\cdot\bar{t}\cdot\bar{u}\cdot 1+\bar{s}\cdot\bar{t}\cdot u\cdot 0+\bar{s}\cdot t\cdot\bar{u}\cdot 1+\bar{s}\cdot t\cdot u\cdot r+s\cdot\bar{t}\cdot\bar{u}\cdot r+s\cdot\bar{t}\cdot u\cdot r+s\cdot t\cdot\bar{u}\cdot 1+s\cdot t\cdot u\cdot r$$

Demultiplexer

obvod s opačnou operací než multiplexer

stejnou funkci vykonává ***dekodér***, u něhož však výstupy chápeme jako signál kódovaný v kódu 1 z *n*

3.3 Další způsoby realizace kombinační logické funkce

speciální digitální integrované obvody v řadách 74 a 40/45

⇒ velké série, levné, dokonale propracované

⇒ radla, multiplexery, demultiplexery, dekodéry, EX-OR, enkodéry s funkcí opačnou k funkci dekodérů, generátory parity, sčítačky, aritmeticko-logické jednotky atd.,

⇒některé z nich jsou vybaveny speciálními vstupními a výstupními obvody, např. výstupy s větším přípustným proudovým nebo napěťovým zatížením, s otevřeným kolektorovým výstupem, s třístavovým výstupním zesilovačem, se vstupní hysterezí (vybavené Schmittovým obvodem na vstupu) atd.

obvodů s otevřeným kolektorovým výstupem

paměti ROM, PROM a jejich různé varianty (EPROM, EEPROM atd.)

programátory

nevhodný sortiment pro realizaci kombinační logické funkce

běžné paměti PROM jsou také zhruba o řád pomalejší než jiné kombinační IO

EPROM nebo paměti OTP (One Time Programmable) se stejným čipem

EEPROM

programovatelné logické obvody PLD - Programmable Logic Devices
zejména obvody typu GAL, ale i složitější typy zvané CPLD - Complex PLD
velkou předností obvodů PLD je, že v nich lze realizovat současně bloky kombinačního i sekvenčního charakteru, což přispívá ke zmenšení potřebného počtu pouzder
rychlost (zpoždění) se blíží parametrům základních kombinačních obvodů

PŘEHLED
základní kombinační obvody NAND, NOR a jejich neinvertované verze
vhodné pro jednoduché funkce, pro jejichž realizaci vystačíme s jedním či dvěma pouzdry
snadné odstranění hazardů
jednoduchost, nízká cena, malé zpoždění signálu
nevýhody - omezený rozsah funkcí a nutnost změny zapojení včetně spoje při změně funkce
podobné výhody a nevýhody i při použití AND-OR-INVERT.

multiplexery

vhodné pro jednu funkci (nebo malý počet funkcí) s nevelkým počtem vstupních proměnných (čtyři až pět, kde lze vystačit s jedním pouzdrem multiplexeru)

jednoduchost návrhu, nízká cena, malé zpoždění

dekodéry

realizace kombinační logické funkce především v integrovaných obvodech - jsou základním stavebním prvkem pamětí PROM a programovatelných logických obvodů

efektivní při realizaci více funkcí těchže proměnných, pokud jejich tvar je takový, že nevyžaduje příliš mnoho dalších pouzder IO

speciální obvody

obvykle nejvýhodnější řešení, pokud jde právě o funkce, pro něž jsou navrženy

je-li však nutno doplňovat je dalšími kombinačními obvody, bývá často výhodnější použít programovatelné logické obvody

Paměti PROM jsou nepostradatelné tam, kde se jedná o realizaci funkcí mnoha vstupních proměnných a je žádána možnost dodatečné změny těchto funkcí zcela libovolným způsobem, pokud není na závadu jejich větší zpoždění. To je zejména případ pamětí programu pro mikropočítače, ale i dalších aplikací, kde se vyskytují velmi složité kombinační logické funkce.

Programovatelné logické obvody

Volba způsobu realizace

⇒ nejprve se přesvědčíme, zda se pro uvažovanou aplikaci nevyrábí speciální integrovaný obvod, popř. zda se nějaký takový obvod jednoduchým přizpůsobením nestane vhodným řešením,

⇒ při realizaci malého počtu jednoduchých funkcí, kde vystačíme s jedním nebo se dvěma pouzdry, použijeme základní kombinační členy (NAND, NOR a další),

⇒ při větší počtu - multiplexer nebo dekodér nebo častěji PLD (kombinační i sekvenční bloky v jednom pouzdru),

⇒pro složité funkce mnoha proměnných, na které PLD nestačí nebo nezbytná úplná univerzálnost (libovolné dodatečné změny) – PROM.

Nejčastěji **realizace kombinační logické funkce** pomocí digitálních integrovaných obvodů:

NAND,
NOR,
popřípadě AND, OR,
např. AND-OR-INVERT, EX-OR,
multiplexery a demultiplexery,
speciální kombinační integrované obvody
převodníky kódu, generátory parity, sčítačky, násobičky, multiplexery, demultiplexery ad.,
paměti PROM a EPROM,
programovatelné logické obvody (PLD).

Zvláštní případy:
tranzistory,
diody apod.

Table 3. Worst-Case Values of Primary Interfacing Parameters

PARAMETER	74HCMOS	AHC	74TTL	74LS	74AS	74ALS
V_{IH}min	3.5 V	3.85 V	2 V	2 V	2 V	2 V
V_{IL}max	1 V	1.65 V	0.8 V	0.8 V	0.8 V	0.8 V
V_{OH}min	4.9 V	3.8 V	2.4 V	2.7 V	2.7 V	2.7 V
V_{OL}max	0.1 V	0.44 V	0.4 V	0.4 V	0.4 V	0.4 V
I_{IH}max	1 μA	1 μA	40 μA	20 μA	200 μA	20 μA
I_{IL}max	–1 μA	–1 μA	–1.6 mA	–400 μA	–2 mA	–100 μA
I_{OH}max	–4 mA	–8 mA	–400 μA	–400 μA	–2 mA	–400 μA
I_{OL}max	4 mA	8 mA	16 mA	8 mA	20 mA	4 mA