Logické obvody 1 - Úvod

Principy logických obvodů jsou vlastně velice jednoduché. Mikrovlnná trouba, pračka, otvírání garážových vrat i počítač jsou řízeny ve své podstatě logickými obvody, které vyhodnocují určitou situaci podle navržené logické funkce.

Představme si třeba výtah, u kterého je nutné hlídat, zda jsou zavřené dveře, jestli není přetížen a zda je stisknuto tlačítko volby patra atd. Výše uvedená fakta jsou pro nás tzv. "vstupní proměnné". Podle navrženého logického obvodu pak svými "výstupními funkcemi" zapíná motor, signalizuje přetížení nebo spouští alarm při nenadálém zablokování výtahu, tedy automaticky ovládá chod výtahu.

```
vstupní proměnné = čidla, snímače, senzory
výstupní funkce  = akční členy, signalizace
```

Logickou závislost „výstupů" na „vstupech" řeší vnitřní struktura. Ta je navržena podle principů výrokové logiky a může být řešena jako kombinační logický obvod nebo sekvenční logický obvod.

Skutečná fyzická realizace záleží už jen na našich možnostech. Můžeme vytvořit náš obvod pomocí diskrétních integrovaných obvodů nebo paměťovým obvodem třeba v jazyce VHDL.

```
návrh = výroková logika, graf přechodů
vnitřní struktura = kombinační obvod, sekvenční obvod
realizace = diskrétní součástky, paměťový obvod
```

Kombinační logický obvod

Realizuje takové logické funkce, u kterých je logická hodnota výstupní funkce dána pouze kombinací okamžitých hodnot vstupních proměnných.

kombinační funkce nejčastěji zapisujeme:

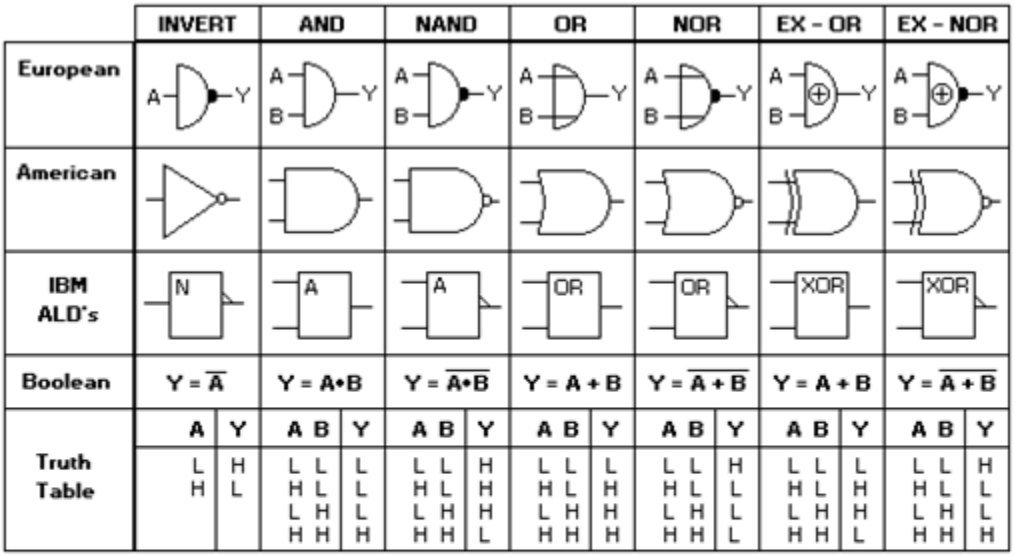
1. slovním popisem
2. pravdivostní tabulkou
3. algebraickým výrazem

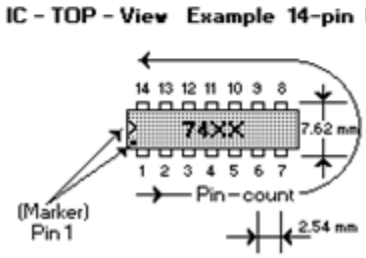
4. stavovým indexem
5. Karnaughovou mapou
6. logickým obvodem
7. jazykem VHDL

Rozdělujeme je podle stupně integrace:

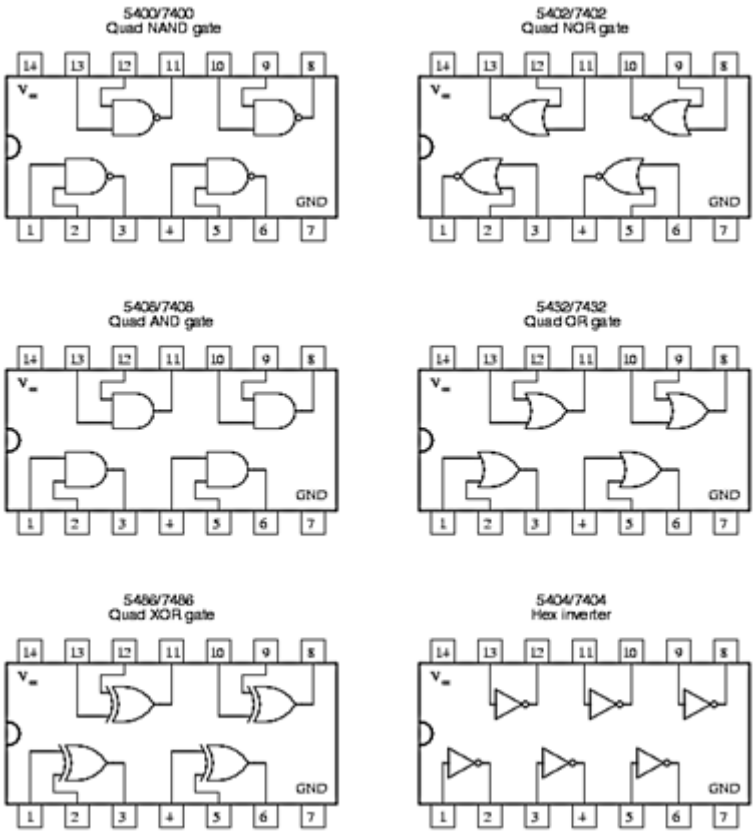
1. SSI - small scale integration,(1-10 hradel),AND,OR,NOT,NAN-D,NOR,EXOR,EXNOR
2. MSI - medium scale integration,(10-100),aritmetické obvody,dekodé-ry,multiplexory
3. LSI - large scale integration,(100-100 000),ALU,řadič

SSI - základní logické členy





SSI - vnitřní zapojení základních logických členů



Slovní zadání příkladů

Následující příklady jsou pouze ukázkou, jak by mohly být jednotlivé úlohy zadány. První tři úlohy by se mohly řešit jako jednoduché kombinační obvody pomocí pravdivostní tabulky. Pro čtvrtý, pátý a šestý příklad je již lepší využít integrované obvody střední integrace a u sedmého příkladu jde o návrh jednoduchého sekvenčního automatu. Každý návrh je pak trochu jiný.

1. př.: Navrhněte logický obvod, který vysílá signál v případě poruchy jednoho nebo obou větráků. U každého větráku jsou umístěny snímače, které vysílají trvale signál, dojde-li k poruše větráku.
2. př.: Automat na nápoje vydá vodu nebo citronovou a nebo malinovou limonádu podle výběru stisknutého tlačítka. Vodu zákazník dostane zdarma, limonády pouze po vložení příslušné finanční částky. Navrhněte ovládání ventilů zásobníků: vody, citronového sirupu a malinového sirupu a zásobníku peněz. Limonáda se míchá z vody a příslušného sirupu, tedy musí být spuštěny oba ventily najednou. Pokud zadá zákazník špatnou kombinaci, rozsvítí se signalizace "špatná volba" a peníze jsou vráceny.
3. př.: Navrhněte úplnou binární sčítačku pro dvě jednobitová čísla.
4. př.: Navrhněte obvod, který rozsvítí led diodu, pokud součet dvou čtyřbitových binárních čísel je větší než dekadické číslo 5.

5. př.: Vytvořte multiplexor a demultiplexor.
6. př.: Navrhněte asynchronní čítač, který má volitelný směr čítání (čítá vzestupně nebo sestupně).
7. př.: Navrhněte synchronní čítač, který bude simulovat světelný semafor.

Booleova algebra

Booleova algebra je binární algebra (má jen dvě konstanty 0 a 1) a využívá AND, OR a NOT jako úplný soubor funkcí (žádné jiné nemá). **Používá se k úpravě a minimalizaci logických funkcí.** Platí v ní tři zákony, podobně jako v klasické algebře:

Asociativní

Při stejném operátoru nezáleží na závorkách.

$$(A \cdot B) \cdot C = A \cdot (B \cdot C)$$
$$(A + B) + C = A + (B + C)$$

Komutativní

Nezáleží na pořadí prvků.

$$A \cdot B = B \cdot A$$
$$A + B = B + A$$

Distributivní

První rovnice je běžné roznásobení, ale pozor, druhá rovnice v běžné algebře neplatí. Když si obě rovnice pořádně prohlédnete, tak se jen zamění znaménka 😉

$$A \cdot (B + C) = A \cdot B + A \cdot C$$
$$A + (B \cdot C) = (A + B) \cdot (A + C)$$

Zákony není potřeba nějak studovat, prostě se počítá jako v matematické algebře. Pro zjednodušování jsou však mnohem důležitější následující pravidla:

Pravidla pro jednu proměnnou

$$A \cdot A = A \qquad A \cdot \overline{A} = 0 \qquad \overline{\overline{A}} = A$$
$$A + A = A \qquad A + \overline{A} = 1$$

Agresivnost nebo neutrálnost 0 a 1:

$$A \cdot 0 = 0 \qquad A \cdot 1 = A$$
$$A + 0 = A \qquad A + 1 = 1$$

Pravidla pro dvě proměnné

$$A + \overline{A}B = A + B$$
$$\overline{A} + AB = \overline{A} + B$$

DeMorganův teorém

Oba zákony platí pro dvě i pro více proměnných.

$$\overline{(A + B)} = \overline{A} \cdot \overline{B} \qquad \overline{(A + B + C + \ldots)} = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \ldots$$
$$\overline{(A \cdot B)} = \overline{A} \cdot \overline{B} \qquad \overline{(A \cdot B \cdot C \cdot \ldots)} = \overline{A} + \overline{B} + \overline{C} + \ldots$$

Je dobré umět zjednodušit jednoduché logické funkce, ale pro složitější je výhodné využít Karnaughovu mapu. V současnosti zjednodušování umí některé simulační programy, např. NI Multisim. V případě, že nerealizujeme obvod základními logickými členy, není nutné funkci zjednodušovat vůbec, neboť se do paměti zapisuje buď celá pravdivostní tabulka, nebo se to řeší přes instrukce.

Příklady

Zjednodušení logické funkce pomocí pravidel Booleovy algebry

Úprava pomocí těchto pravidel je velmi zdlouhavá a dělá se při ní poměrně dost chyb. Navíc většinou není úplně jisté, zda je výsledek již opravdu minimální. Velkou

výhodou je jen, že lze využít kdykoliv "ručně" a pro více proměnných není (mimo počítačových algoritmů) jiná metoda vhodná.

1.př. :

$f = ab(\bar{c}+d)+ab\bar{c}\bar{d}+ad(b+\bar{a}b)+\bar{a}bd$ ... roznásobím e závorky (distributivní zákon)

$f = ab\bar{c}+abd+ab\bar{c}\bar{d}+abd+aabd+\bar{a}bd$ ... $aa = a$ (pravidlo absorbce)

$f = ab\bar{c}+abd+ab\bar{c}\bar{d}+abd+abd+\bar{a}bd$ ... je tam 3x abd, tak zase platí pravidlo absorbce

$f = ab\bar{c}+abd+ab\bar{c}\bar{d}+\bar{a}bd$ ... vytkneme $ab\bar{c}$ a bd

$f = ab\bar{c}(1+d)+bd(a+\bar{a})$ ... $1+$cokoliv, je vždy 1 (agresivní 1) a $a+\bar{a}=1$ (pravidla pro jednu proměnnou)

$f = ab\bar{c}+bd = b(a\bar{c}+d)$

2.př. :

$f = (\bar{a}+b)(a+b+\bar{c})(\bar{a}+c)(b+c) = (\bar{a}a+\bar{a}b+\bar{a}\bar{c}+ab+bb+b\bar{c})(\bar{a}b+\bar{a}c+bc+cc)$ ... roznásobení, absorbce a pravidlo $\bar{a}a=0$

$f = (0+\bar{a}b+\bar{a}\bar{c}+ab+b+b\bar{c})(\bar{a}b+\bar{a}c+bc+c)$ ... v první závorce vytkneme b, ve druhé závorce vytkneme c

$f = [b(\bar{a}+a+1+\bar{c})+\bar{a}\bar{c}][c(\bar{a}+b+1)+\bar{a}b]$ ... $1+$cokoliv je 1

$f = (b+\bar{a}\bar{c})(c+\bar{a}b)$ ... roznásobím e

$f = bc+\bar{a}bb+\bar{a}cc+\bar{a}\bar{a}bc$ ... absorbce a pravidlo $\bar{a}a=0$

$f = bc+\bar{a}b+0+\bar{a}bc$ ... vytkneme $\bar{a}b$

$f = bc+\bar{a}b(1+\bar{c})$ ... $1+$cokoliv je 1

$f = bc+\bar{a}b = b(c+\bar{a})$

Zjednodušení logické funkce pomocí zákonů De Morgana

Stejně jako úprava podle Booleovy algebry i tady jde spíše o procvičení si logického myšlení a aplikace zákonů na příkladech. V současné době se většina obvodů realizuje zcela jinak a nebo lze využít již hotových počítačových programů. Nicméně na SŠ a i na VŠ se zatím oba dva typy příkladů vyskytují a zkouší a tak je dobré tyto úpravy znát a velmi dobře procvičit 😉 .

1.př. :

$f = \overline{\overline{\overline{a+b}\cdot\bar{a}\cdot\bar{b}}} = \overline{\overline{(\overline{(a+b)}\cdot\bar{a})\cdot\bar{b}}}$ ... a postupně zbavujeme "zhora" negací podle De Morganových zákonů

$f = \overline{(\overline{(a+b)}\cdot\bar{a})}+\bar{\bar{b}}$ ... dvě stejně dlouhé negace se vyruší

$f = (\overline{a+b})\cdot\bar{a}+b$

$f = (\bar{a}+\bar{b})\cdot\bar{a}+b = \bar{a}\bar{a}+\bar{a}\bar{b}+b = \bar{a}+\bar{a}\bar{b}+b = \bar{a}(1+\bar{b})+b = \bar{a}+b$ ... a upravíme podle Booleových pravidel

2.př. :

$f = \overline{a+b+c(\overline{a+b})}$ ... lze postupovat i "zdola"

$f = \overline{a+b+\overline{c\cdot\bar{a}\bar{b}}} = \overline{a+b+\bar{c}+\bar{a}\bar{b}} = \overline{a+b+\bar{c}+\bar{\bar{a}}+\bar{\bar{b}}} = \overline{a+b+\bar{c}+a+b} = \overline{a+b+\bar{c}} = \bar{a}\cdot\bar{b}\cdot\bar{\bar{c}} = \bar{a}\bar{b}c$

Logické obvody 2 - Řešení příkladů a pravdivostní tabulka

Řešení slovního zadání

Pokud máme úlohu zadánu jako slovní příklad, je zapotřebí udělat něco jako rozbor úlohy. To v podstatě znamená určit vstupní proměnné, (z minula již víme, že to budou nějaká čidla nebo snímače) a určit výstupní funkce (to jsou akční členy – takže vykonají nějakou akci ...). Prohlídneme si zadání příkladu a.:

a. *Logický obvod vysílá signál v případě poruchy jednoho nebo obou větráků. U každého větráku jsou umístěny snímače, které vysílají trvale signál, dojde-li k poruše větráku.*

Zeleně je označeno, kde se hovoří o snímačích a jejich počtu a červeně je označeno místo, které představuje požadovanou činnost navrhovaného zařízení, myslím tím činnost našeho navrhovaného obvodu. Takže budeme potřebovat dvě vstupní proměnné, protože větráky jsou dva a jednu výstupní funkci, neboť je požadováno pouze vyslání signálu v případě poruchy, je otázkou jakého, ale budeme předpokládat, že jde třeba o rozsvícení ledky.

1. **vstupní proměnné** ... 2 větráky (2 snímače) => 2 proměnné např. ***a*** a ***b***

V úvodu je uvedeno, že se logické obvody navrhují podle výrokové logiky, takže by to chtělo nějaké výroky 😎. Výrok vyjadřuje nějaké tvrzení, na které lze odpovědět buď ano, nebo ne.

proměnná ***a*** ... *"první větrák přestal pracovat"* ... pokud ano, pak ***a*** = 1; pokud ne, pak ***a*** = 0

proměnná ***b*** ... *"druhý větrák přestal pracovat"* ... pokud ano, pak ***b*** = 1; pokud ne, pak ***b*** = 0

(v praxi se výrok zapisuje tak, aby představoval stav, že se událost, kterou hlídáme, stala. Pak logickou 1 přiřadíme odpovědi ano a logickou 0 odpovědi ne. Samozřejmě může to být i naopak, ale když to budete takto dodržovat, bude to lepší)

2. **výstupní proměnné** ... 1 signál (1 led dioda) => 1 výstupní funkce ***f***

Výstupní funkce ***f*** ... *"rozsvítí se led dioda"* ... ano, je-li ***f*** = 1; ne, je-li ***f*** = 0

Vlastní logiku, tedy to, kdy se opravdu rozsvítí ledka, provádí navržený logický obvod podle zadání tj.:
"v případě, že dojde k poruše jednoho nebo obou větráků"

Logiku je nejjednodušší navrhnout pomocí pravdivostní tabulky, takže co to vlastně je pravdivostní tabulka?

Pravdivostní tabulka

Jednoznačně přiřazuje určitou konkrétní kombinaci vstupních proměnných jedné nebo několika výstupním funkcím.

Levá část obsahuje všechny možné kombinace vstupních proměnných a stavový index (což je dekadicky vyjádřená binární hodnota příslušné kombinace vstupních proměnných). Počet těchto kombinací je dán počtem proměnných. Takže máme-li dvě proměnné např. ***a*** a ***b***, které mohou každá nabývat hodnoty *0* nebo *1*, pak máme čtyři různé kombinace a to: *00, 01, 10 a 11*. Obecně je počet všech kombinací $N = 2^n$, kde ***n*** se rovná počtu vstupních proměnných.

Pravá část obsahuje hodnoty výstupních funkcí, které se stanoví na základě logiky slovního zadání.

Pravdivostní tabulka pro náš předchozí příklad

Bude mít tři sloupce vlevo (stavový index ***s*** a dvě vstupní proměnné ***a*** a ***b***) a celkem 4 řádky ($N=2^2$)

A jen jeden sloupec vpravo ***f***, protože máme jen jednu výstupní funkci.

s	*b*	*a*	*f*
0	0	0	
1	0	1	
2	1	0	
3	1	1	

Ze slovního zadání vyplývá, že výstupní funkce bude rovna 1 (tj. rozsvítí se led dioda): *„v případě, že dojde k poruše jednoho nebo obou větráků“*

Pravdivostní tabulka po doplnění ukazuje všechny případy, ke kterým může dojít:

s	*b*	*a*	*f*	*slovní popis*
0	0	0	0	*žádný větrák nemá poruchu*
1	0	1	1	*první větrák má poruchu*
2	1	0	1	*druhý větrák má poruchu*
3	1	1	1	*oba větráky mají poruchu*

Výhodou pravdivostní tabulky je, že zachycuje všechny kombinace, které mohou nastat a jak bude daný obvod reagovat. Nic jiného ani kombinační obvod neumí a my nemůžeme na nic zapomenout 🙂.

Neúplně zadaná funkce

Někdy se však stane, že pro určitou kombinaci na vstupu neexistuje fyzikální realizace, prostě daná vstupní kombinace není možná. Pak takové pravdivostní tabulce říkáme:

Pravdivostní tabulka s neúplně zadanou funkcí (nebo funkcemi). Důvod je ten, že nevíme, zda výstupní funkce bude nabývat pro danou kombinaci log.1 nebo log.0. Vlastně je to jedno, protože daná vstupní kombinace ani nenastane, pak píšeme ***x*** místo 1 nebo 0. Pro názornost si uvedeme jednoduchý příklad.

Příklad neúplně zadané funkce

Př.: Navrhněte hlídání hladiny v nádrži tak, aby se rozsvítila kontrolka K1, pokud klesne hladina pod minimum a kontrolka K2, pokud hladina přesáhne maximum.

1. **vstupní proměnné** ... 2 snímače (minimum a maximum) => 2 proměnné např. ***a*** a ***b***

proměnná ***a*** ... *"snímač minima pod vodou"* ... pokud ano, pak ***a*** = 1; pokud ne, pak ***a*** = 0

proměnná ***b*** ... *"snímač maxima pod vodou"* ... pokud ano, pak ***b*** = 1; pokud ne, pak ***b*** = 0

2. **výstupní proměnné** ... 2 signály (2 led diody, K1 a K2) => 2 výstupní funkce ***f1*** a ***f2***

výstupní funkce ***f1*** ... *"rozsvítí se K1"* ... ano, je-li ***f1*** = 1; ne, je-li ***f1*** = 0

výstupní funkce ***f2*** ... *"rozsvítí se K2"* ... ano, je-li ***f2*** = 1; ne, je-li ***f2*** = 0

pravdivostní tabulka

s	b	a	f_1	f_2	slovní popis
0	0	0	1	0	*ani jeden snímač není pod vodou => hladina pod minimem, svítí K1*
1	0	1	0	0	*snímač min. pod vodou, snímač max. ne => žádaný stav, nesvítí nic*
2	1	0	x	x	*snímač min. je „na suchu“, snímač max. je pod vodou =>* ***fyzikálně nemožné***
3	1	1	0	1	*oba snímače pod vodou = > hladina nad maximem, svítí K2*

Tyto *"neúplně zadané"* stavy je důležité v tabulce vyznačit (a ne je vynechat), neboť se využívají pro zjednodušení minimalizace výstupní funkce.

Zápis funkce stavovým indexem

Jak už bylo uvedeno v předchozím článku, funkci lze také zapsat pomocí stavového indexu. Jde vlastně o zkrácený zápis pravdivostní tabulky. Stavový index je dekadické číslo, jehož hodnota odpovídá binární kombinaci vstupních proměnných v určitém řádku. Samozřejmě zde záleží v jakém pořadí se vstupní proměnné uvádí. V předchozích příkladech je proměnná ***a*** vždy nejvíce vpravo a tedy jde o bit s nejnižší vahou 2^0.

Zápisy pak mají následující tvar:

1.př.: úplně zadaná funkce - porucha vrtáku

za symbolem *"suma"* je výčet jedničkových stavů výstupní funkce
za symbolem *"pí"* je výčet nulových stavů výstupní funkce

$$f(b,a) = \sum(1,2,3)$$
$$f(b,a) = \prod(0)$$

2.př.: neúplně zadaná funkce - Hlídání hladiny vody

Za symbolem *"suma"* je výčet jedničkových stavů výstupní funkce a za symbolem *"suma s x"* jsou nedefinované stavy

Za symbolem *"pí"* je výčet nulových stavů výstupní funkce a za symbolem *"pí s x"* jsou nedefinované stavy

$$f1(b,a) = \sum(0) + \sum_X(2) \qquad f2(b,a) = \sum(3) + \sum_X(2)$$
$$f1(b,a) = \prod(1,3) + \prod_X(2) \qquad f2(b,a) = \prod(0,1) + \prod_X(2)$$

Oba zápisy, ať už ze sumou nebo pí, vyjadřují stejnou funkci.

Logické obvody 3 - Normální formy a Karnaughovy mapy

Realizace pomocí logických členů, co je potřeba

1. **zjednodušení (minimalizace) logické funkce**

   a. buď: výpis výstupní funkce v algebraickém tvaru + minimalizace pomocí pravidel Booleovy algebry
   b. nebo: zápis výstupní funkce přímo do Karnaughovy mapy, která slouží k minimalizaci výstupní funkce grafickou metodou

2. **nakreslení schéma logického obvodu**

   a. buď: přímo pomocí základních logických členů – AND, OR, NOT nebo i dalších (EXOR, ...)
   b. nebo: pouze pomocí členů NAND nebo pouze pomocí členů NOR, napřed je ale potřeba převést logickou funkci podle De Morganových zákonů

Výpis výstupní logické funkce v algebraickém tvaru

1. Jako **úplná normální disjunktní forma (ÚNDF)**

   - tu získáme z pravdivostní tabulky tak, že vytvoříme součiny vstupních proměnných v řádcích, kde má výstupní funkce hodnotu $f = 1$ tzv. mintermy. Všechny tyto mintermy pak sečteme. Každá proměnná v součinu je zapsána tak, že pokud nabývá hodnoty log 0, pak ji píšeme s negací, pokud log 1, pak píšeme bez negace.

2. jako **úplná normální konjunktní forma (ÚNKF)**

   - která se skládá ze součtů vstupních proměnných v řádcích, kde má výstupní funkce hodnotu $f = 0$ tzv. maxtermů, a všechny tyto maxtermy pak vynásobíme. Každá proměnná v součtu je zapsána tak, že pokud nabývá hodnoty log 0, pak ji píšeme bez negace, pokud log 1, pak píšeme s negací.

Příklad vytvoření ÚNDF a ÚNKF z pravdivostní tabulky:

s	b	a	f	slovní popis	minterm	maxterm
0	0	0	0	žádný větrák není rozbitý	-	a+b
1	0	1	1	první větrák se rozbil	$a\bar{b}$	-
2	1	0	1	druhý větrák se rozbil	$\bar{a}b$	-
3	1	1	1	oba větráky se rozbily	ab	-

ÚNDF: $f = a\bar{b} + \bar{a}b + ab$

ÚNKF: $f = a + b$

Samozřejmě teď bychom mohli nakreslit schéma obvodu, ale vždy je dobré výstupní funkci zkusit zjednodušit. Již dříve jsme využívali pravidla Booleovy algebry, takže jen zopakujeme 😎

Minimalizace ÚNDF podle Booleovy algebry

$$f = a\bar{b} + \bar{a}b + ab = a(\bar{b} + b) + \bar{a}b = a + \bar{a}b = a + b$$

Jak je vidět, dospěli jsme ke stejnému výsledku jako je ÚNKF. Oba tvary jsou samozřejmě tatáž funkce, takže to zas tak velké překvapení není. Zajímavé je na tom jen to, že ÚNKF byl tvar, který byl již minimální (byl to jediný řádek, resp. jediná nulová hodnota funkce, takže proto). Princip minimalizace spočívá v určitém vhodném slučování jedniček pro ÚNDF a nul pro ÚNKF, nicméně minimalizace se dělá především pro ÚNDF a pro ÚNKF se volí pouze, pokud je výrazněji méně nul než jedniček, což byl náš případ.

Schéma výstupní funkce ze základních logických členů

Tak a teď schéma ze základních logických členů. Příklad je příliš jednoduchý, takže je hned zřejmé, že postačuje použít jeden OR.

Raději uvedu ještě jeden příklad, který je zadán pravdivostní tabulkou a opět máme vypsat ÚNDF a ÚNKF:

s	c	b	a	f	minterm	maxterm
0	0	0	0	0	-	$a+b+c$
1	0	0	1	0	-	$\bar{a}+b+c$
2	0	1	0	0	-	$a+\bar{b}+c$
3	0	1	1	1	$ab\bar{c}$	
4	1	0	0	0	-	$a+b+\bar{c}$
5	1	0	1	1	$a\bar{b}c$	
6	1	1	0	1	$\bar{a}bc$	
7	1	1	1	1	abc	

ÚNDF: $f = ab\bar{c} + a\bar{b}c + \bar{a}bc + abc$

ÚNKF: $f = (a+b+c)(\bar{a}+b+c)(a+\bar{b}+c)(a+b+\bar{c})$

Minimalizace ÚNDF podle Booleovy algebry

$f = ab\bar{c} + a\bar{b}c + \bar{a}bc + abc = ab(\bar{c}+c) + a\bar{b}c + \bar{a}bc = ab + a\bar{b}c + \bar{a}bc = a(b+\bar{b}c) + \bar{a}bc = ab + ac + \bar{a}bc = ab + c(a+\bar{a}b) =$
$= ab + ac + bc$

Tak tady už byla úprava mnohem zajímavější, ale realizovatelná.

Pokud bychom chtěli zjednodušovat ÚNKF, bylo by nutné závorky mezi sebou roznásobit a .... to už samo o sobě je hrozná představa, takže to dělat nebudeme 😉

Schéma výstupní funkce ze základních logických členů

Ještě schéma naší funkce. Ze základních členů budeme potřebovat 3x dvouvstupový AND a 2x dvouvstupový OR.

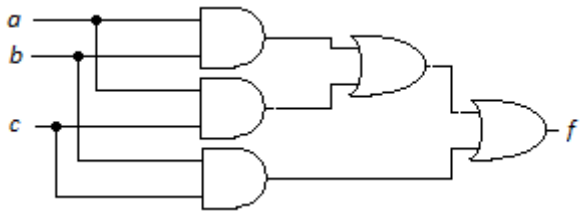


Schéma vypadá dobře a není ani moc složité, ale každý typ logického členu je obsažen v jiném integrovaném obvodu. Konkrétně dvouvstupový AND v TTL 7408, kde jsou čtyři tyto členy a dvouvstupový OR zase v TTL 7432, kde jsou také čtyři členy. Což tedy znamená celkem dva integrované obvody.

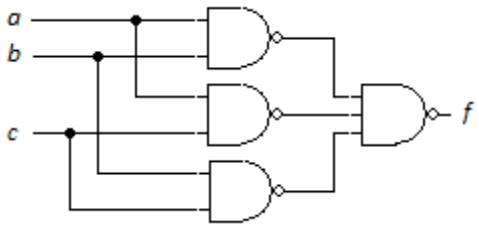
Realizace výstupní funkce pouze pomocí členů NAND

V rámci úspor (místa i financí) se minimalizace týká i omezení typů použitých členů. Takže se pak provádí převedení již minimalizované funkce na jen NAND nebo jen NOR a to pomocí De Morganových pravidel.

Já to trošku zjednoduším. Chceme-li použít jen NAND, musíme se zbavit všech součtů ve funkci a naopak chceme-li funkci realizovat jen NOR, pak je potřeba odstranit součiny. To provedeme "dvojitou negací" nad součtem (nebo součinem). Otázka je proč dvojitá negace, když podle De Morgana stačí jen jedna negace pro změnu součtu na součin (nebo součinu na součet). Tak pokud bychom naší funkci znegovali pouze jednou, tak vlastně realizujeme funkci právě opačnou než jsme chtěli. Takže ta druhá negace je tam proto, abychom nezměnili původní funkci. Pro vlastní demorganování ji nepoužijeme, jen tam zůstane.

$$f = ab + ac + bc = \overline{\overline{ab + ac + bc}} = \overline{\overline{ab} \cdot \overline{ac} \cdot \overline{bc}}$$

Schéma výstupní funkce jen z logických členů NAND



Když na to teď koukám, tak jsme si moc nepomohli, protože zase budeme potřebovat dva integrované obvody (TTL 7400 a TTL 7411). Sice je dvou i třívstupové NAND stejný typ, ale bohužel se rozlišuje i počet vstupů, ale doufám, že princip je jasný 😜.

Minimalizace výstupní funkce pomocí Karnaugovy mapy

Cílem minimalizace je nalézt co nejjednodušší vyjádření zadané logické funkce. Tato metoda je vhodná maximálně pro 4 až 5 proměnných, ale je rychlá a výsledná funkce je vždy v minimálním tvaru.

Hledáme:

MNDF, minimální normální disjunktní formu - logický součet minimálního počtu minimálních součinů (mintermů)

nebo

MNKF, minimální normální konjunktní formu - logický součin minimálního počtu minimálních součtů (maxtermů)

Postup minimalizace pomocí Karnaughovy mapy

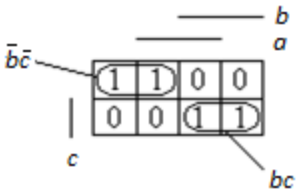
1. Zapíšeme výstupní funkci do mapy.
2. Vytvoříme smyčky. Pro hledání **MNDF** vytváříme smyčky přibližně ve tvaru čtverce nebo obdélníku, které obsahují, co největší počet „sousedních jedničkových stavů“. Počet těchto stavů musí být vždy mocninou čísla 2 (tj. 1, 2, 4, 8, 16, … atd.). Sousední jedničkové stavy jsou „jedničky“, které spolu sousedí hranou, a to i přes okraje mapy. Smyčky se mohou navzájem překrývat, neděláme však smyčky nadbytečné. Všechny jedničky musí být popsány buď v rámci některé smyčky, nebo samostatně.
3. Jednotlivé smyčky se popisují mintermy, složenými pouze z těch proměnných, které se během celé smyčky nemění (proměnná a bude v mintermu obsažena, pokud pro všechny „1“ ve smyčce nabývá stejné hodnoty, např. a = 1). Termy dvou sousedních polí se od sebe liší jen ve stavu jedné proměnné a o tuto proměnnou lze funkci, při sloučení těchto polí do smyčky, zjednodušit. Čím více sousedních „1“ je ve smyčce, tím méně proměnných bude v příslušném mintermu.
4. Výsledná **MNDF** je součtem takto vytvořených minimálních mintermů.

Pro **MNKF** platí totéž, s tím, že smyčky vytváříme kolem „sousedních nulových stavů“ a smyčky se popisují pomocí maxtermů. Výsledná **MNKF** je pak součinem těchto minimálních maxtermů.

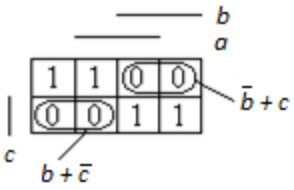
Př.: Napište pravdivostní tabulku, Karnaughovu mapu, MNDF a MNKF funkce, zadané pomocí stavových indexů.

$f(c,b,a) = \sum(0,1,6,7)$

s	c	b	a	f
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1



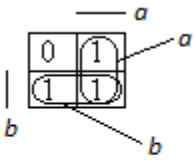
MNDF: $f=\bar{b}\bar{c} + bc$



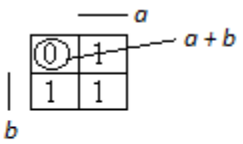
MNKF: $f=(\bar{b}+c)(b+\bar{c})$

Př.: Vytvořte pravdivostní tabulku, Karnaughovu mapu, MNDF a MNKF logického členu OR. Uvažujte dvě vstupní proměnné.

s	b	a	f
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1



MNDF: $f = a + b$



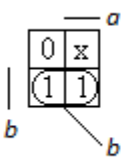
MNKF: $f = a + b$

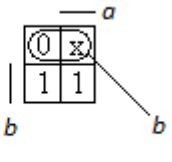
Minimalizace neúplně zadaných funkcí

Při minimalizaci neúplně zadaných funkcí postupujeme shodně jako při minimalizaci funkcí úplně zadaných s tím, že neurčené stavy **x** zahrnujeme buď do smyček s „jedničkami“ nebo do smyček s „nulami“ nebo je nemusíme zahrnout do žádné smyčky. Vždy hledíme na výhodnost jejich polohy pro tu kterou minimalizaci. Stručně řečeno, pokud je výhodné zahrnout neurčitý stav a získat tak větší smyčku (tedy menší počet proměnných) učiníme tak, jinak ne.

Př.: pravdivostní tabulka a Karnaughova mapa funkce neúplně zadané

s	b	a	f
0	0	0	0
1	0	1	x
2	1	0	1
3	1	1	1



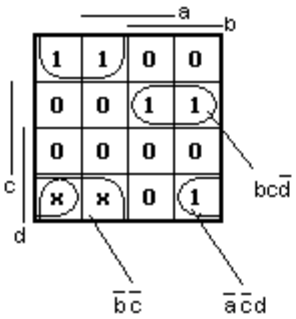


Pro **MNDF** není výhodné neurčený stav zahrnovat, vedlo by to pouze k vytvoření další smyčky a tím dalších proměnných.

Pro **MNKF** je však smyčka s **x** výhodnější neboť je popsána jen proměnnou ***b***, oproti popisu samotné „0“, jež by vedlo k popisu ***(a + b)***.

Př.: Minimalizujte neúplně zadanou funkci pomocí Karnaughovy mapy a zapište ve tvaru MNDF. Funkce je zadána stavovým indexem.

$$f(d,c,b,a) = \sum(0,1,6,7,10) + \sum_x(8,9)$$



$$MNDF: f = \bar{b}\cdot\bar{c} + bc\bar{d} + \bar{a}\cdot\bar{c}d$$

Tak myslím, že v tomto díle toho už bylo poměrně dost 😎, tak abychom se úplně mentálně nevyčerpali je nutné říci, že většina lepších simulačních programů umí provést minimalizaci ze zadané pravdivostní tabulky nebo funkce v algebraickém tvaru jedním kliknutím.