SLOVENSKÁ TECHNICKÁ UNIVERZITA
Fakulta elektrotechniky a informatiky
Ústav elektroniky a fotoniky

Logické systémy
Zbierka riešených príkladov

Autori:
Ing. Lukáš Nagy, PhD.
prof. Ing. Viera Stopjaková, PhD.

Marec, 2024
Bratislava

Obsah

Úvod

Učebný text „Zbierka riešených príkladov k predmetu Logické systémy", predstavuje ucelený súbor teoretického základu a postupu riešenia príkladov pre uvedený predmet, ktorý sa vyučuje v prvom ročníku na Fakulte elektrotechniky a informatiky, Slovenskej technickej univerzity v Bratislave.Sú v ňom obsiahnuté jednotlivé témy predmetu „Logické systémy" zoradené podľa náplne a harmonogramu samotných prednášok počas semestra. Tento učebný text *nemá byť vnímaný* ako prednáškový materiál, či snáď dokonca náhrada za prednášky alebo samotné cvičenia. Ide o zbierku príkladov spolu so vzorovým riešením pre každú skupinu úloh. Príklady majú slúžiť ako rozšírenie a obohatenie príkladov riešených počas cvičení.

V prvej tretine skrípt sa nachádzajú najnutnejšie teoretické princípy a základy vybrané z Booleovskej algebry potrebné pre nasledujúce témy v rámci semestra, ktoré môžu napomôcť k úspešnému zvládnutiu záverečnej skúšky z predmetu „Logické systémy". Ide predovšetkým o definície matematických pojmov a nástrojov pre opis a manipuláciu s binárnymi číslami (dátami).

Druhá časť dokumentu obsahuje učivo a príklady pokrývajúce teóriu, analýzu a návrh kombinačných logických obvodov pomocou základných nástrojov Booleovskej algebry. Táto časť napríklad zahŕňa aj návrh kombinačných obvodov na tranzistorovej úrovni, t. j. na úrovni hardvérových spínačov nanometrových rozmerov na kremíkovom čipe integrovaného obvodu.

Tretia a posledná časť návodu na cvičenia sa zaoberá teoretickým základom potrebným pre analýzu a návrh sekvenčných logických obvodov – konečných stavových automatov. Táto časť je zároveň zhrnutím učiva celého semestra, nakoľko prepája a dáva do súvisu všetky predtým získané poznatky, vedomosti a zručnosti.

Poznámka: Autori tohto dokumentu nemajú formálne matematické vzdelanie, preto je možné, že niektoré matematické názvy a vyjadrenia použité v tejto práci nebudú stopercentne korektné alebo v súlade s ich striktnou matematickou definíciou. Zamerali sme sa skôr na aplikáciu vybranej časti matematiky v technickom svete.

Prevod medzi číselnými sústavami

Táto úvodná kapitola opisuje teoretický základ a postup prevodu čísel medzi desiatkovou (dekadickou), dvojkovou (binárnou) a šestnástkovou (hexadecimálnou) číselnou sústavou. Všetky tieto sústavy partia medzi pozičné číselné sústavy. Uvedené postupy konverzie čísel medzi sústavami boli vybrané z viacerých možných prístupov a nemali by byť považovať za „jediné správne".
Každá pozičná číselná sústava sa matematicky zapisuje všeobecnou definíciou vyjadrenou v rovnici (1.1).

$$\sum_{i=0}^{N} a_i.z^i = \left(a_N.z^N\right) + \left(a_{N-1}.z^{N-1}\right) + \cdots + \left(a_i.z^i\right) + \cdots + \left(a_1.z^1\right) + \left(a_0.z^0\right)$$
$$a = \{0, 1, \ldots z-1\} \tag{1.1}$$
$$z \in \mathbb{N}$$

kde
a – symbol číslice (cifra)
z – základ číselnej sústavy (2, 10, 16)
z^i – váha číslice

Slovný opis rovnice (1.1) pre desiatkovú sústavu by sa dal interpretovať ako súčet a_0 jednotiek, a_1 desiatok, a_2 stoviek, a_3 tisícok atď. Pričom základ sústavy je $z = 10$, preto číslice (a) môžu nadobúdať hodnoty 0 až 9. Váhy sú $10^0 = 1$, $10^1 = 10$, $10^2 = 100$, $10^3 = 1000$ atď. Napríklad pre číslo 3025 v desiatkovej sústave môžeme povedať, že ide o 3 tisícky, 0 stoviek, 2 desiatky a 5 jednotiek.

Rovnaký postup vieme aplikovať na ktorúkoľvek číselnú sústavu, ale hodnoty parametrov rovnice (1.1) sa budú javiť menej priamočiarejšie. Opíšme si dvojkovú, resp. binárnu sústavu. Základ číselnej sústavy je $z = 2$. Číslice teda môžu nadobúdať hodnoty 0 a 1.
Váhy (alebo bitové váhy) budú mať hodnotu pre 8-bitové číslo $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$, $2^5 = 32$, $2^6 = 64$, $2^7 = 128$ atď. Hodnota nasledujúcej váhy je vždy dvojnásobok tej predchádzajúcej. Binárne číslo 8-bitovej šírky môže vyzerať napríklad takto: 10110101.
Pri šestnástkovej sústave sú parametre nasledovné. Základ sústavy je $z = 16$ a číslice môžu byť od 0 po 15. Dvojciferné číslice budeme v hexadecimálnej sústave označovať písmenami. Teda 10 = A, 11 = B, 12 = C, 13 = D, 14 = E, 15 = F. Váhy sú teda $16^0 = 1$, $16^1 = 16$, $16^2 = 256$, $16^3 = 4096$, $16^3 = 65536$ atď. Príklad čísla v hexadecimálnej sústave môže vyzerať napríklad nasledovne: C47F.

1.1 Prevod medzi dekadickou a binárnou sústavou

Pri prevode z desiatkovej číselnej sústavy do dvojkovej existuje nespočetne veľa algoritmov. Jeden z najčastejšie používaných postupov je *celočíselné* delenie základom sústavy. Teda v našom prípade delenie číslom 2. Zvyšok po delení môže v tomto prípade byť buď 0, alebo 1. Posledný zvyšok po ukončení delenia predstavuje najviac významový bit – MSB. Obr. 1.1 ilustruje opisovaný postup prevodu čísla 3025 z desiatkovej do binárnej sústavy.

$$
\begin{aligned}
3025 : 2 &= 1512 \quad \text{zvyšok } 1 \ \text{LSB} \\
1512 : 2 &= 756 \quad \text{zvyšok } 0 \\
756 : 2 &= 378 \quad \text{zvyšok } 0 \\
378 : 2 &= 189 \quad \text{zvyšok } 0 \\
189 : 2 &= 94 \quad \text{zvyšok } 1 \\
94 : 2 &= 47 \quad \text{zvyšok } 0 \\
47 : 2 &= 23 \quad \text{zvyšok } 1 \\
23 : 2 &= 11 \quad \text{zvyšok } 1 \\
11 : 2 &= 5 \quad \text{zvyšok } 1 \\
5 : 2 &= 2 \quad \text{zvyšok } 1 \\
2 : 2 &= 1 \quad \text{zvyšok } 0 \\
1 : 2 &= 0 \quad \text{zvyšok } 1 \ \text{MSB}
\end{aligned}
$$

$$3025_{\text{D}} = 101111010001_{\text{B}}$$

$$101111010001 = 1011\_1101\_0001$$

Obr. 1.1: Prevod do dvojkovej sústavy celočíselným delením

Opačný smer prevodu (z binárnej do dekadickej) si opíšeme na základe postupného súčtu bitových váh. Napríklad, ak sa snažíme previesť binárne číslo 10011101 do desiatkovej sústavy sčítame tie bitové váhy, kde má príslušný bit hodnotu logickej 1. V našom prípade ide o váhy $2^7 = 128$, $2^4 = 16$, $2^3 = 8$, $2^2 = 4$ a $2^0 = 1$, a ich sčítaním dostaneme konečné dekadické číslo 157. Spomínaný prevod je ilustrovaný na Obr. 1.2.

$$\begin{array}{cccccccc} 128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 \\ 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{array}$$

$$128+16+8+4+1 = 157$$

$$10011101_B = 157_D$$

Obr. 1.2: Prevod do desiatkovej sústavy súčtom bitových váh

1.2 Prevody medzi dekadickou a hexadecimálnou sústavou

Prevod z desiatkovej do šestnástkovej sústavy sa dá opäť dosiahnuť celočíselným delením základom sústavy, teda delením číslom 16, pričom zvyšok môže mať hodnotu od 0 po 15. Opisovaný postup však nie je praktický, pre ručné počítanie bez kalkulačky. Použijeme preto medzikrok – prevod do binarnej sústavy. Následne každé *štyri* bity (lebo $2^4 = 16$) prevedieme do šestnástkovej sústavy, pričom začíname od najmenej významového bitu. Obr. 1.3 ukazuje postup prevodu čísla 423 z dekadickej do hexadecimálnej sústavy. Najprv číslo 423 prevedieme do binárnej sústavy, v ktorej bude mať tvar 110100111. Pre prehľadnosť ho zapíšeme do nového, čitateľnejšieho formátu po štvoriciach bitov: 1 | 1010 | 0111. Posledný kvartet môžeme v prípade potreby doplniť nulami. Výsledné číslo sa tým nezmení a dostávame: 0001 | 1010 | 0111. Následne každý binárny kvartet prevedieme do hexadecimálnej sústavy zvlášť. Správny výsledok opisovaného prevodu do hexadecimálnej sústavy je 1A7 pretože 0001 = 1 | 1010 = A | 0111 = 7.

$$\begin{aligned} 423_D &= 110100111_B = \\ &= 000110100111_B = \\ &= \quad 1 \quad\ A \quad\ 7_H \end{aligned}$$

Obr. 1.3: Prevod čísla z desiatkovej do šestnástkovej sústavy

Spätný prevod z hexadecimálnej do dekadickej sústavy je doslova identický postup, ale opačným smerom. Najprv číslice z hexadecimálnej sústavy prevedieme do binárnej a nakoniec do desiatkovej sústavy. Ilustrácia postupu je zobrazená na Obr. 1.4. Príklad opisuje prevod čísla C2. Každú cifru hexadecimálneho čísla prevedieme zvlášť do štyroch bitov binárnej sústavy. C = 1100 | 2 = 0010, čo zapíšeme spolu ako 11000010. Po súčte bitových váh pre jednotkové bity dostávame výsledné dekadické číslo: 128+64+2 = 194.

$$C2_H =$$
$$= 11000010_B$$

128	64	32	16	8	4	2	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	0	0	0	1	0

$$128+64+2 = 194_D$$

Obr. 1.4: Prevod čísla z hexadecimálnej do desiatkovej sústavy

1.3 Príklady prevodov medzi sústavami

Príkladov na manuálny prevod čísiel medzi rôznymi číselnými sústavami je prakticky neobmedzené množstvo. V rámci predmetu „Logické systémy" sa však obmedzíme len na prirodzené čísla ($\mathbb{N}$) do šírky 8 bitov. Jedinou výnimkou budú práve príklady na prevod medzi číselnými sústavami. Skúšku správnosti manuálneho prevodu čísla vieme vykonať použitím kalkulačky alebo pomocou internetových zdrojov. Rovnako vieme aj vymyslieť ľubovoľné číslo vo vybranej číselnej sústave na precvičenie.

Riešené prevody z desiatkovej do binárnej sústavy:

$12_D = 1100_B$	$54_D = 110110_B$	$33_D = 100001_B$	$189_D = 10111101_B$
$147_D = 10010011_B$	$18_D = 10010_B$	$46_D = 101110_B$	$29_D = 11101_B$
$131_D = 10000011_B$	$287_D = 100011111_B$	$73_D = 1001001_B$	$111_D = 1101111_B$
$13_D = 1101_B$	$55_D = 110111_B$	$34_D = 100010_B$	$190_D = 10111110_B$
$132_D = 10000100_B$	$288_D = 100100000_B$	$74_D = 1001010_B$	$113_D = 1110001_B$
$148_D = 10010100_B$	$19_D = 10011_B$	$47_D = 101111_B$	$30_D = 11110_B$
$10_D = 1010_B$	$62_D = 111110_B$	$56_D = 111000_B$	$112_D = 1110000_B$
$210_D = 11010010_B$	$31_D = 11111_B$	$49_D = 110001_B$	$38_D = 100110_B$
$41_D = 101001_B$	$20_D = 10100_B$	$119_D = 1110111_B$	$23_D = 10111_B$
$14_D = 1110_B$	$122_D = 1111010_B$	$22_D = 10110_B$	$64_D = 1000000_B$
$65_D = 1000001_B$	$72_D = 1001000_B$	$32_D = 100000_B$	$69_D = 1000101_B$
$8_D = 1000_B$	$23_D = 10111_B$	$154_D = 10011010_B$	$169_D = 10101001_B$

Riešené prevody z binárnej do desiatkovej sústavy:

$110_B = 6_D$	$11010111_B = 215_D$	$101110_B = 46_D$	$101010_B = 42_D$
$111001_B = 57_D$	$11010_B = 26_D$	$11011_B = 27_D$	$110001_B = 49_D$
$10010001_B = 145_D$	$10010011_B = 147_D$	$101011_B = 43_D$	$10101010_B = 170_D$
$110100_B = 52_D$	$1100111_B = 103_D$	$1010111_B = 87_D$	$10101100_B = 172_D$
$1010100_B = 84_D$	$1011100_B = 92_D$	$1111100_B = 124_D$	$1001100_B = 76_D$
$1001_B = 9_D$	$10101_B = 21_D$	$1111111_B = 127_D$	$1101111_B = 111_D$
$100101_B = 37_D$	$110101_B = 53_D$	$1100000_B = 96_D$	$1100001_B = 97_D$
$11000100_B = 196_D$	$110011_B = 51_D$	$100111_B = 39_D$	$10011_B = 19_D$
$11011000_B = 216_D$	$100101100_B = 300_D$	$100001_B = 33_D$	$11101_B = 29_D$
$101001_B = 41_D$	$10001_B = 17_D$	$10011000_B = 152_D$	$11010001_B = 209_D$
$1001101_B = 77_D$	$101101101_B = 365_D$	$110100_B = 52_D$	$111110_B = 62_D$
$110110_B = 54_D$	$100001110_B = 270_D$	$1100110_B = 102_D$	$110111_B = 55_D$

Riešené prevody z desiatkovej do hexadecimálnej sústavy:

$12_D = C_H$	$54_D = 36_H$	$33_D = 21_H$	$189_D = BD_H$
$147_D = 93_H$	$18_D = 12_H$	$46_D = 2E_H$	$29_D = 1D_H$
$131_D = 83_H$	$287_D = 11F_H$	$73_D = 49_H$	$111_D = 6F_H$
$13_D = D_H$	$55_D = 37_H$	$34_D = 22_H$	$190_D = BE_H$
$132_D = 84_H$	$288_D = 120_H$	$74_D = 4A_H$	$113_D = 71_H$
$148_D = 94_H$	$19_D = 13_H$	$47_D = 2F_H$	$30_D = 1E_H$
$10_D = A_H$	$62_D = 3E_H$	$56_D = 38_H$	$112_D = 70_H$
$210_D = D2_H$	$31_D = 1F_H$	$49_D = 31_H$	$38_D = 26_H$
$41_D = 29_H$	$20_D = 14_H$	$119_D = 77_H$	$23_D = 17_H$
$14_D = E_H$	$122_D = 7A_H$	$22_D = 16_H$	$64_D = 40_H$
$65_D = 41_H$	$72_D = 48_H$	$32_D = 20_H$	$69_D = 45_H$
$8_D = 8_H$	$23_D = 17_H$	$154_D = 9A_H$	$169_D = A9_H$

Riešené prevody z hexadecimálnej do desiatkovej sústavy:

$16_H = 22_D$	$32_H = 50_D$	$25_H = 37_D$	$22_H = 34_D$
$A32_H = 2610_D$	$C3_H = 195_D$	$34_H = 52_D$	$85_H = 133_D$
$1FF_H = 511_D$	$18_H = 24_D$	$14_H = 20_D$	$19_H = 25_D$
$D22_H = 3362_D$	$A2_H = 162_D$	$38_H = 56_D$	$2A_H = 42_D$
$F55_H = 3925_D$	$CD_H = 205_D$	$AD_H = 173_D$	$100_H = 256_D$
$102_H = 258_D$	$111_H = 273_D$	$22_H = 34_D$	$4F_H = 79_D$
$FF_H = 255_D$	$B1_H = 177_D$	$10_H = 16_D$	$DA_H = 218_D$
$AA_H = 170_D$	$A6_H = 166_D$	$4_H = 4_D$	$20_H = 32_D$
$DC1_H = 3521_D$	$AF_H = 175_D$	$69_H = 105_D$	$32_H = 50_D$
$AC2_H = 2754_D$	$FB_H = 251_D$	$3D_H = 61_D$	$49_H = 73_D$
$65_H = 101_D$	$CC_H = 204_D$	$2D_H = 45_D$	$11_H = 17_D$
$B8_H = 184_D$	$1C1_H = 449_D$	$1D_H = 29_D$	$CF_H = 207_D$

Pravdivostná tabuľka

Pravdivostná tabuľka (PT) je jeden z kľúčových nástrojov zápisu či definície funckií v Booleovej algebre (B-funkcií). Ide o nástroj, ktorý budeme používať počas celého semestra, pričom významná väčšina príkladov a úloh vyžaduje zápis a prácu s pravdivostnou tabuľkou. Preto je veľmi dôležité, aby študenti ovládali a dostatočne si precvičili vytváranie a čítanie PT, ako aj prácu s ňou. Ako názov napovedá, PT obsahuje iba logické hodnoty a premenné (0 / 1, áno / nie, pravda / nepravda). Ak máme tabuľku s N vstupnými a M výstupnými premennými, bude mať tabuľka $N+M$ stĺpcov a 2^N riadkov, pričom číslo 2^N zároveň určuje počet všetkých možných kombinácií vstupných premenných. Obr. 2.1 zobrazuje pravdivostné tabuľky pre jednu, dve, tri a štyri premenné. V rámci predmetu „Logické systémy" budeme pracovať maximálne so štyrmi vstupnými premennými, nakoľko pre vyšší počet premenných sú pravdivostné tabuľky rozsiahle a neprehľadné. Modrou farbou je vyznačená vstupná časť, zelenou farbou je označená výstupná časť tabuľky. Pre prehľadnosť sme na úvod rozdelili časti tabuľky odtieňami farieb. Pri ručnom kreslení si môžeme pomôcť horizontálnou čiarou každé štyri riadky a dvomi vertikálnymi čiarami pre oddelenie vstupnej a výstupnej časti PT.

A	Y
0	
1	

A	B	Y
0	0	
0	1	
1	0	
1	1	

A	B	C	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

A	B	C	D	Y
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

Obr. 2.1: Pravdivostné tabuľky pre rôzny počet premenných

Ako bolo spomenuté, vstupná časť PT musí obsahovať *všetky* možné kombinácie vstupných premenných. Postupov, ako správne vyplniť ľavú stranu PT, je niekoľko. My si vysvetlíme dva najpoužívanejšie. Prvou možnosťou je inkrementácia N-bitového čísla v binárnej sústave, pričom začíname od nuly v prvom riadku PT a postupujeme po riadkoch tabuľky. Druhou možnosťou je vyplnenie prvej polovice prvého stĺpca (stĺpec premennej A na Obr. 2.1) logickými nulami a druhú polovicu stĺpca vyplníme logickými jednotkami. Nasledujúci stĺpec vyplníme opäť striedavo logickými nulami a jednotkami, ale tentokrát po štvrtinách rozsahu. Nasledujúci stĺpec bude vypĺňaný po osminách rozsahu atď. Stĺpec poslednej premennej musí byť vyplnený striedavo každý riadok tabuľky. Môžeme tiež povedať, že smerom od prvého po posledný stĺpec vstupnej časti PT sa zdvojnásobuje frekvencia striedania logických núl a jednotiek v riadkoch pravdivostnej tabuľky.

2.1 Booleovské funkcie

Elementárne booleovské operácie/funkcie, z ktorých sa dajú odvodiť všetky ostatné, sú tieto: logická negácia, logický súčin a logický súčet. Tieto tri logické funkcie tvoria tzv. *úplný súbor logických funkcií* (ÚSBF). Pre zjednodušenie zápisu sa často používajú kombinácie negácie a logického súčinu či súčtu, ktoré takisto tvoria ÚSBF.

1. Logická negácia (NOT): $f = \overline{A}$
2. Logický súčin (AND): $f = A.B = AB$
3. Logický súčet (OR): $f = A + B$
4. Negovaný logický súčin (NAND): $f = \overline{A.B} = \overline{AB}$
5. Negovaný logický súčet (NOR): $f = \overline{A + B}$

Tu musíme zdôrazniť, že v rámci tohto textu a aj celého predmetu Logické systémy operátory „+" a „ ." vyjadrujú vždy logický súčet a logický súčin. Vhodnou kombináciou uvedených funkcií je možné „vyskladať" ľubovoľne zložitú booleovskú funkciu, resp. ľubovoľne zložitú funkciu vieme prepísať do tvaru elementárnych booleovských funkcií. Definíciu opisovaných funkcií môžeme vidieť v nasledujúcich pravdivostných tabuľkách – Tab. 2.1 pre dve vstupné premenné a Tab. 2.2 pre tri vstupné premenné.

Tab. 2.1: Pravdivostná tabuľka základných Booleovských funkcií

A	B	$\overline{A}$	$A.B$	$A + B$	$\overline{A.B}$	$\overline{A + B}$
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	1	0
1	1	0	1	1	0	0

Tab. 2.2: Pravdivostná tabuľka základných booleovských funkcií

A	B	C	$\overline{A}$	$A.B.C$	$A+B+C$	$\overline{A.B.C}$	$\overline{A+B+C}$
0	0	0	1	0	0	1	1
0	0	1	1	0	1	1	0
0	1	0	1	0	1	1	0
0	1	1	1	0	1	1	0
1	0	0	0	0	1	1	0
1	0	1	0	0	1	1	0
1	1	0	0	0	1	1	0
1	1	1	0	1	1	0	0

Je potrebné zaviesť ešte jeden významný pojem, ktorý budeme používať počas množstva príkladov v nasledujúcich kapitolách. Ide o tzv. De Morganovo pravidlo, ktoré spája do súvislosti logický súčet a logický súčin. Matematická definícia De Morganovho pravidla pre dve a tri premenné je zapísaná vo výrazoch (2.2). Pre kompletnosť uvádzame všetky možné kombinácie interpretácie De Morganovho pravidla pre už spomínané dve či tri premenné. Analogicky by sme však vedeli odvodiť toto pravidlo pre štyri, päť a viac premenných.

$$\begin{aligned} A+B &= \overline{\overline{A}.\overline{B}} & \qquad A+B+C &= \overline{\overline{A}.\overline{B}.\overline{C}} \\ \overline{A}+\overline{B} &= \overline{A.B} & \qquad \overline{A}+\overline{B}+\overline{C} &= \overline{A.B.C} \\ \overline{A+B} &= \overline{A}.\overline{B} & \qquad \overline{A+B+C} &= \overline{A}.\overline{B}.\overline{C} \\ \overline{\overline{A}+\overline{B}} &= A.B & \qquad \overline{\overline{A}+\overline{B}+\overline{C}} &= A.B.C \end{aligned} \tag{2.2}$$

Ľudskou rečou by sa De Morganovo pravidlo dalo voľne opísať nasledovne: „Pravidlo dáva do rovnosti logický súčet a logický súčin za pomoci použitia logických negácií". Negácie je potrebné aplikovať v dvoch „úrovniach". Prvá úroveň je *vždy* naviazaná na jednotlivé premenné (resp. operandy), teda každá premenná na *jednej* strane rovnice bude negovaná. Druhá úroveň negovania sa *vždy* viaže na celú jednu stranu rovnice. Pričom nezáleží, na ktorej strane bude negácia aplikovaná.

2.2 Vzorový príklad

Úloha: Vytvorte pravdivostnú tauľku pre booleovskú funkciu $f = A.\overline{B}.C + A.\overline{C}$.
Riešenie: Funkcia f je funkciou troch premenných (A, B, C). Pravdivostná tabuľka (Tab. 2.3) bude mať teda osem riadkov (2^3). Pri riešení vzorového príkladu budeme postupovať krok po kroku. Avšak po vyriešení zopár príkladov budeme schopní výrazne zredukovať počet potrebných krokov, a tým aj potrebný čas na vyriešenie, čo je dôležité pri testoch počas semestra, ako aj pri záverečnej skúške.

1. V prvom kroku vyplníme vstupnú časť PT (definičný obor funkcie).
2. Pripravíme si potrebné negované hodnoty vstupných premenných.
3. Vypočítame prvý člen funkcie $A.\overline{B}.C$
4. Vypočítame druhý člen funkcie $A.\overline{C}$
5. Vykonáme logický súčet čiastkových výpočtov podľa definície funkcie f.

Tab. 2.3: Pravdivostná tabuľka riešeného príkladu

A	B	C	$\overline{B}$	$\overline{C}$	$A.\overline{B}.C$	$A.\overline{C}$	f
0	0	0	1	1	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	1	0	0	0
0	1	1	0	0	0	0	0
1	0	0	1	1	0	1	1
1	0	1	1	0	1	0	1
1	1	0	0	1	0	1	1
1	1	1	0	0	0	0	0

Úloha: Sú funkcie $f = A.\overline{B}.C + A.\overline{C}$ a $g = \overline{A}.B.C + \overline{A}.C$ ekvivalentné?
Riešenie: Zostrojíme pravdivostné tabuľky pre obe funkcie (Tab. 2.5) a porovnáme výstupné hodnoty pri identických hodnotách vstupných premenných.

Tab. 2.4: PT funkcie f

A	B	C	$\overline{B}$	$\overline{C}$	$A.\overline{B}.C$	$A.\overline{C}$	f
0	0	0	1	1	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	1	0	0	0
0	1	1	0	0	0	0	0
1	0	0	1	1	0	1	1
1	0	1	1	0	1	0	1
1	1	0	0	1	0	1	1
1	1	1	0	0	0	0	0

Tab. 2.5: PT funkcie g

A	B	C	$\overline{A}$	$\overline{A}.B.C$	$\overline{A}.C$	g
0	0	0	1	0	0	0
0	0	1	1	0	1	1
0	1	0	1	0	0	0
0	1	1	1	1	1	1
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	0	0	0	0
1	1	1	0	0	0	0

Keď porovnáme stĺpce funkcií f a g, je zrejmé, že funkcie nie sú ekvivalentné, nakoľko výstupné hodnoty funkcií sa rôznia.

2.3 Príklady

Vytvorte pravdivostnú tabuľku pre uvedené funkcie troch premenných. V prípade potreby si môžete vymyslieť vlastné príklady funkcií a pre overenie správnosti použiť online nástroje.

$f = A.C + A.\overline{B}.C + B.C$

A	B	C	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$f = A.\overline{B}.\overline{C} + A.\overline{B} + \overline{A}.C$

A	B	C	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

$f = \overline{A}.B.C + A.B.\overline{C}$

A	B	C	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

$f = A.(B + C) + B.C$

A	B	C	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$f = B.C + (A + B)$

A	B	C	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

$f = (A + B).(A + C)$

A	B	C	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

$f = B.(\overline{A} + C) + \overline{B}.C$

A	B	C	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$f = B.\overline{C} + (\overline{A} + B)$

A	B	C	f
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

$f = (A + B + \overline{C}).(B + C)$

A	B	C	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	1
1	1	1	1

$f = A.D + A.\overline{B}.D + A.C$

A	B	C	D	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

$f = A.B.\overline{D} + A.\overline{B}.C + \overline{A}.D$

A	B	C	D	f
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

$f = \overline{A}.B.D + A.B.\overline{C}$

A	B	C	D	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

$f = A.C + A.\overline{B}.D + C.D$

A	B	C	D	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

$f = A.B.\overline{C} + A.\overline{D} + \overline{A}.C$

A	B	C	D	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

$f = \overline{A}.B.C + A.B.\overline{C.D}$

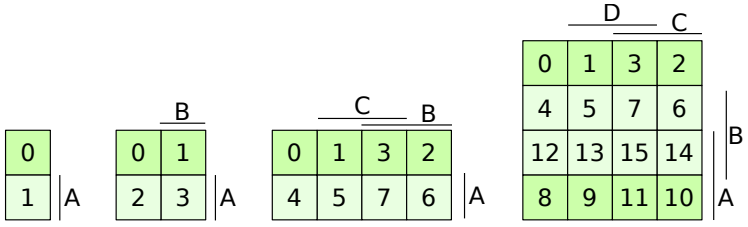
A	B	C	D	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Overte platnosť De Morganovho pravidla pre štyri premenné.
Napríklad interpretáciu pravidla $\overline{A.B.C.D} = \overline{A} + \overline{B} + \overline{C} + \overline{D}$

A	B	C	D	$\overline{A.B.C.D}$	$\overline{A} + \overline{B} + \overline{C} + \overline{D}$
0	0	0	0	1	1
0	0	0	1	1	1
0	0	1	0	1	1
0	0	1	1	1	1
0	1	0	0	1	1
0	1	0	1	1	1
0	1	1	0	1	1
0	1	1	1	1	1
1	0	0	0	1	1
1	0	0	1	1	1
1	0	1	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1
1	1	0	1	1	1
1	1	1	0	1	1
1	1	1	1	0	0

Karnaughova mapa

Karnaughova mapa (K-mapa) predstavuje ďalší významný nástroj pre prácu s funkciami v booleovskej algebre. Ide o grafickú reprezentáciu pravdivostnej tabuľky, ktorá sa zväčša používa na grafickú minimalizáciu booleovských funkcií. V tejto kapitole si ukážeme, ako sa vytvára Karnaughova mapa a ako sa následne vypĺňa *výstupnými* hodnotami funkcie/funkcií z pravdivostnej tabuľky.
Karnaughova mapa sa skladá z 2^N políčok, kde N je počet vstupných premenných booleovskej funkcie. Postupov ako vytvoriť K-mapu je niekoľko, a preto by sa nasledujúci návod nemal brať ako „jediný správny". Obr. 3.1 zobrazuje Karnaughove mapy pre jednu, dve, tri a štyri vstupné premenné booleovskej funkcie, pričom tieto sú priamo spojené s ich pravdivostnými tabuľkami z Obr. 2.1. Jednotlivé políčka sme pre účel vysvetlenia očíslovali, pričom číslo políčka reprezentuje číslo príslušného riadku PT (indexujeme vždy od 0). Karnaughova mapa obsahuje tzv. „pruhy" (alebo pásy) vstupných premenných. Tieto je možné umiestniť viacerými spôsobmi, avšak v rámci predmetu „Logické systémy" budeme používať práve zobrazenú konfiguráciu.



Obr. 3.1: Karnaughova mapa pre rôzny počet premenných

Označenie, resp. názov pruhov vstupných premenných je vždy proti „smeru hodinových ručičiek", pričom prvý pruh je nazvaný vždy podľa prvej premennej (pozri Obr. 2.1). Pruhy vstupných premenných predstavujú oblasť K-mapy, kde je daná vstupná premenná v logickej jednotke. Oblasť, kde sa pruh nenachádza, je daná vstupná premenná v logickej nule. Pri dodržaní umiestnenia a označenia pruhov K-mapy bude poradie políčok zodpovedať Obr. 3.1. Všimnime si, že poradie políčok pri troch a štyroch premenných nie je monotónne (poradie preskakuje v horizontálnom či vertikálnom smere). Samozrejme pri inom rozložení a/alebo označení pruhov bude poradie políčok iné, ale bude vyjadrovať to isté.
Nesmierne dôležitou vlastnosťou K-mapy je jej trojrozmerná interpretácia. Karnaughova mapa je v skutočnosti 3D objekt a jej vonkajšie okraje sú preto susediace. Táto vlastnosť je *kriticky* dôležitá pri minimalizácii Booleovských funkcií.

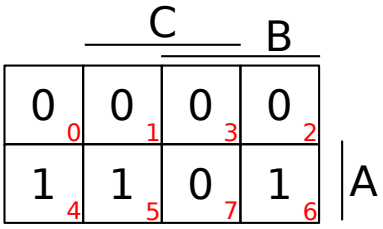
3.1 Vzorový príklad

Úloha: Vytvorte Karnaughovu mapu pre booleovskú funkciu $f = A.\overline{B}.C + A.\overline{C}$.
Riešenie: Pravdivostná tabuľka pre uvedenú funkciu je zobrazená nižšie v Tab. 3.1.

Tab. 3.1: Pravdivostná tabuľka prvého riešeného príkladu

A	B	C	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Keďže ide o funkciu troch premenných Karnaughova mapa bude mať osem (2^3) políčok. Ak umiestnime a nazveme pruhy vstupných premenných podľa predloženého postupu, môžeme Karnaughovu mapu naplniť pomocou uvedeného poradia (červené čísla v políčkach predstavujú čísla príslušných riadkov PT).

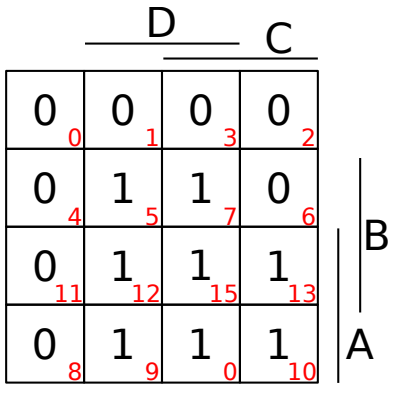


Obr. 3.2: Karnaughova mapa pre prvý riešený príklad

Úloha: Vytvorte K-mapu pre booleovskú funkciu $f = B.D + A.\overline{B}.D + A.C$
Riešenie: Pravdivostná tabuľka pre uvedenú funkciu je zobrazená v Tab. 3.2 a výsledná K-mapa na Obr. 3.3

Tab. 3.2: Pravdivostná tabuľka druhého riešeného príkladu

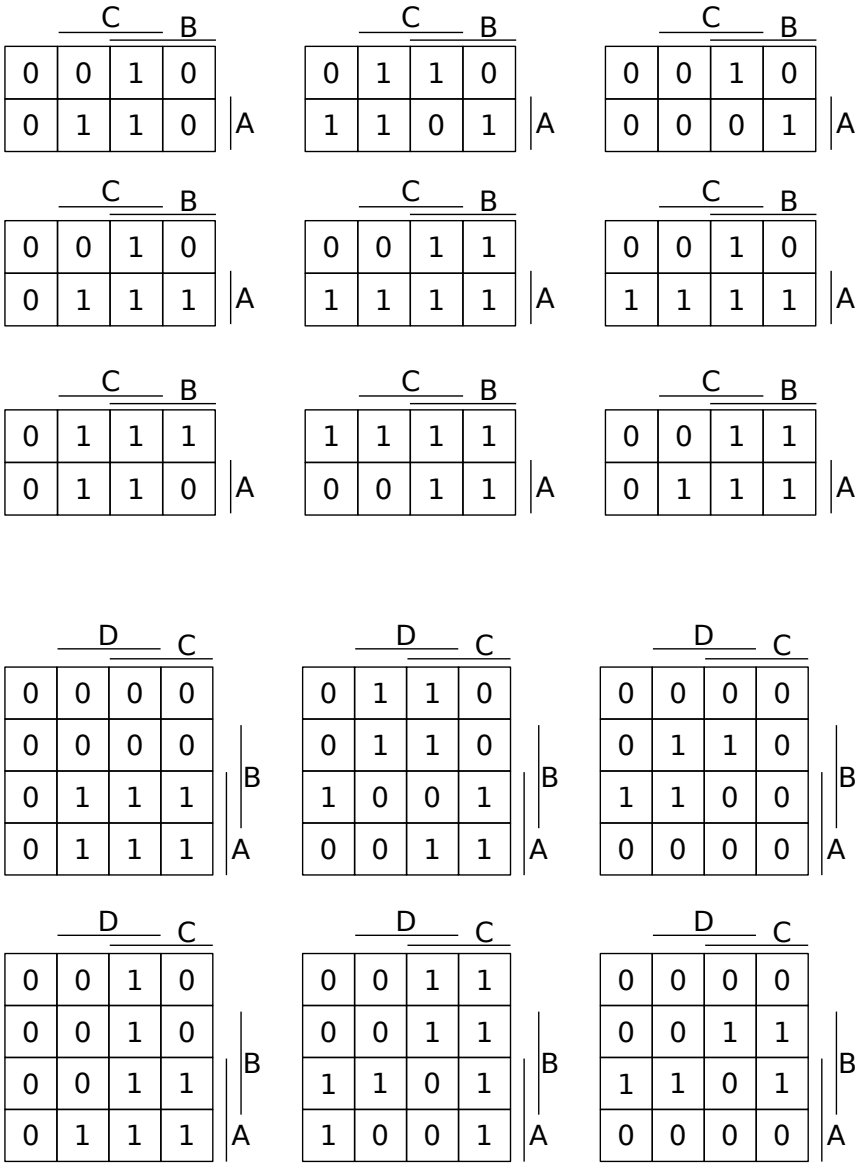
A	B	C	D	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1



Obr. 3.3: Karnaughova mapa druhého riešeného príkladu

3.2 Príklady

Vyvtorte Karnaughove mapy pre funkcie z príkladov v závere kapitoly „Pravdivostná tabuľka". Rozloženie vyriešených Karnaughových máp je identické s rozložením pravdivostných tabuliek v predchádzajúcej kapitole.



Úplná normálna forma (ÚDNF a ÚKNF)

V klasickej algebre môžeme napríklad funkciu $f = a + a$ zapísať aj ako $f = 2a$, teda analogicky môžeme pristupovať aj k zápisu booleovských funkcií. Booleova algebra pozná niekoľko foriem zápisu funkcií, ale v rámci predmetu „Logické systémy" sa obmedzíme iba na použitie tzv. *normálnej formy*. Samotných normálnych foriem je tiež niekoľko druhov. V tejto kapitole sa budeme venovať tzv. *úplnej normálnej forme* funkcie.

Potrebujeme si najskôr zadefinovať (nie striktne matematicky) pojmy ako *jednotkový / nulový bod* booleovskej funkcie a *elementárny logický súčin / súčet*. Jednotkový (nulový) bod booleovskej funkcie je bod, kde výstupná hodnota funkcie nadobúda stav logickej jednotky (nuly).

Elementárny logický súčin sa vzťahuje iba na jednotkové body booleovskej funkcie a predstavuje súčin všetkých vstupných premenných v tvare logických jednotiek. Teda, ak sa nachádza vstupná premenná v jednotkovom bode booleovskej funkcie v stave logickej nuly, pri zápise do elementárneho logického súčinu je potrebné ju negovať. Inak povedané, všetky vstupné premenné musia byť v stave logickej jednotky, aby ich logický súčin tvoriaci jednotkový bod funkcie bol v stave logickej jednotky.

Elementárny logický súčet sa viaže s nulovými bodmi booleovskej funkcie a je zložený z logického súčtu všetkých vstupných premenných v tvare logických núl. Opäť, ak je vstupná premenná v nulovom bode funkcie v stave logickej jednotky, je potrebné ju v rámci elementárneho logického súčtu zapísať v negovanom tvare. Tiež môžme povedať, že všetky vstupné premenné musia byť v stave logickej nuly, aby ich logický súčet tvoriaci nulový bod funkcie bol tiež v stave logickej nuly.

4.1 Vzorový príklad

Príklad vytvárania ÚDNF a ÚKNF si uvedieme na pravdivostnej tabuľke pre elementárny logický súčin a aj súčet. Pravdivostná tabuľka (Tab. 4.1) obsahuje opis jednotkových a nulových bodov definovanej funkcie v podobe elementárnych logických súčinov, resp. elementárnych logických súčtov.

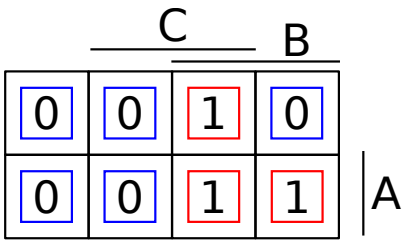
Tab. 4.1: Pravdivostná tabuľka s vytvoreným elementárnym súčinom a súčtom

A	B	C	$f = \overline{A + \overline{B}}.C + (A + C).B$	El. súčin	El. súčet
0	0	0	0		$A + B + C$
0	0	1	0		$A + B + \overline{C}$
0	1	0	0		$A + \overline{B} + C$
0	1	1	1	$\overline{A}.B.C$	
1	0	0	0		$\overline{A} + B + C$
1	0	1	0		$\overline{A} + B + \overline{C}$
1	1	0	1	$A.B.\overline{C}$	
1	1	1	1	$A.B.C$	

Ak sčítame všetky elementárne logické súčiny z pravdivostnej tabuľky, dostávame tzv. *úplnú disjunktívnu normálnu formu – ÚDNF* pôvodnej funkcie. Analogicky, ak vynásobíme všetky elementárne logické súčty z vyššie uvedenej pravdivostnej tabuľky, výsledkom bude tzv. *úplná konjuktívná normálna forma – ÚKNF* pôvodného zápisu funkcie. Matematický zápis všetkých spomínaných tvarov funkcie nájdeme v nasledujúcom zápise (4.3).

$$\begin{aligned} f &= \overline{A + \overline{B}}.C + (A + C).B \\ &= \overline{A}.B.C + A.B.\overline{C} + A.B.C \\ &= (A + B + C).(A + B + \overline{C}).(A + \overline{B} + C).(\overline{A} + B + C).(\overline{A} + B + \overline{C}) \end{aligned} \tag{4.3}$$

Je zrejmé, že ÚDNF funkcie je výhodné použiť ak má pôvodne zadaná funkcia malý počet jednotkových bodov. Analogicky môžeme tvrdiť, že je vhodnejšie použiť ÚKNF ak má pôvodná Booleovská funkcia malý počet nulových bodov. Ideálna príležitosť pre prepísanie Booleovskej funkcie do tvaru ÚDNF alebo ÚKNF nastáva v prípade, ak má funkcia iba jediný jednotkový, resp. nulový bod.
Druhý možný spôsob, ako sa dopracovať k ÚDNF alebo ÚKNF zadanej Booleovskej funkcie je pomocou použitia Karnaughovej mapy (Obr. 4.1). Týmto postupom nemusíme písať do pravdivostnej tabuľky elementárne logické súčiny či súčty. Zároveň si tiež rozšírime vedomosti a možnosti práce s Karnaughovou mapou, ktoré budú nutné v ďalších úlohách a témach v rámci učebného textu.



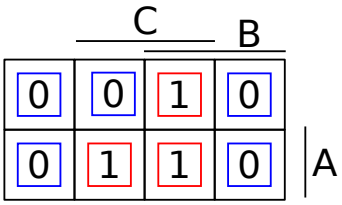
Obr. 4.1: Karnaughova mapa s vyznačenými jednotkovými a nulovými bodmi

Ako prvý výstup z K-mapy si odvodíme ÚDNF. Musíme brať ohľad na umiestnenie, označenie a význam pruhov vstupných premenných. Zameriame sa na jednotkový bod Booleovskej funkcie v hornom riadku K-mapy. Pruh vstupnej premennej A sa v danom jednotkovom bode nachádza v stave logickej nuly, pruh vstupnej premennej B sa nachádza v stave logickej jednotky a pruh vstupnej premennej C sa v danom bode nachádza v stave logickej jednotky. Z uvedeného vyplýva, že analyzovaný jednotkový bod môžeme opísať ako $\overline{A}.B.C$, pričom zároveň ide aj o elementárny logický súčin jednotkového bodu. Druhým analyzovaným jednotkovým bodom bude bod z druhého riadku K-mapy vľavo. Tu sa premenná A nachádza v stave logickej jednotky, B sa nachádza v stave logickej jednotky a C sa takisto nachádza v stave logickej jednotky. Preto bude elementárny logický súčin jednotkového bodu $A.B.C$. Posledný jednotkový bod (dole vpravo v K-mape) je opísaný elementárnym logickým súčinom na základe pruhov nasledovne.

Premenná A sa nachádza v logickej jednotke, premenná B je takisto v logickej jednotke, premenná C sa nachádza v stave logickej nuly. Elementárny logický súčin je preto $A.B.\overline{C}$. Ak sčítame elementárne logické súčiny získané opísaným postupom z K-mapy, výsledkom bude identická ÚDNF ako vo výraze (4.3).
Pri odvodení ÚKNF aplikujeme rovnaký postup, ale vstupné premenné pre elementárne logické súčty budeme zapisovať v tvare logickej nuly, rovnako ako v prípade zápisu pomocou pravdivostnej tabuľky. Opäť budeme postupovať zľava doprava a začneme v hornom riadku K-mapy. Prvý nulový bod funkcie opíšeme nasledovne. Všetky vstupné premenné sa nachádzajú v stave logickej nuly, preto budú všetky vstupné premenné v elementárnom súčte bez negácie, čiže $A+B+C$. Druhý nulový bod funkcie je opísaný premennou A bez negácie, premenná B taktiež bez negácie a premenná C bude negovaná, nakoľko tá sa v opisovanom nulovom bode nachádza v stave logickej jendotky, takže dostaneme zápis $A + B + \overline{C}$. Tretí nulový bod bude opísaný elementárnym logickým súčtom, kde bude negovaná iba premenná B, pretože ostatné premenné sú v stave logickej nuly, čiže dostávame $A+\overline{B}+C$. Nasledujúci nulový bod/elementárny súčet bude obsahovať iba negáciu premennej A, lebo iba tá sa nachádza v stave logickej jednotky: $\overline{A} + B + C$. Posledný elementárny súčet bude obsahovať negácie nad premennými A a C. Teda zápis bude nasledovný: $\overline{A} + B + \overline{C}$. Finálny zápis ÚKNF je identický ako pri postupe s použitím pravdivostnej tabuľky (výraz 4.3).
Ide o individuálnu preferenciu, ktorý postup si pri riešení príkladov zvolíme, oba totiž musia priniesť identické výsledky ako pre ÚDNF, tak aj pre ÚKNF. Postup s Karnaughovou mapou sme si ukázali z dôvodu ukázania možnosti výberu a zároveň spôsobu vykonania skúšky správnosti riešenia. Najdôležitejší aspekt je však zoznámenie sa s vytváraním a prácou s pruhmi vstupných premenných pri K-mape, ktorá ako už bolo spomenuté, bude kľúčová v nasledujúcej kapitole (ale aj ďalších).

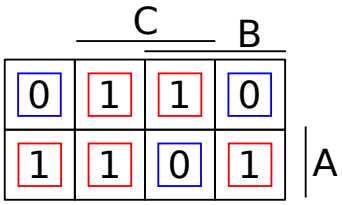
4.2 Príklady

Nájdite ÚDNF a ÚKNF pre funkcie z predchádzajúcej kapitoly. My sme tieto príklady vyriešili pomocou Karnaughovej mapy. Skúste však ako „skúšku správnosti" vyriešiť príklady pomocou zápisu elementárnych logickýcg súčinov a súčtov z pravdivostných tabuliek.



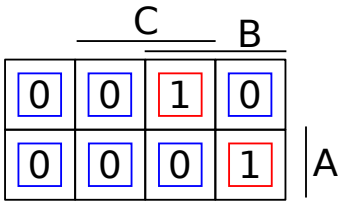
UDNF $= \overline{A}.B.C + A.\overline{B}.C + A.B.C$

UKNF $= (A+B+C).(A+B+\overline{C}).$
$.(A+\overline{B}+C).(\overline{A}+B+C).(\overline{A}+\overline{B}+C)$



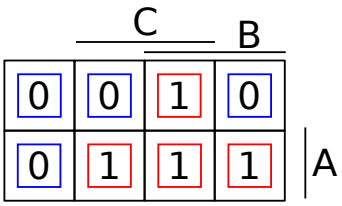
UDNF $= \overline{A}.\overline{B}.C + \overline{A}.B.C + A.\overline{B}.\overline{C} +$
$+A.\overline{B}.C + A.B.\overline{C}$

UKNF $= (A+B+C).(A+\overline{B}+C).$
$.(\overline{A}+\overline{B}+\overline{C})$



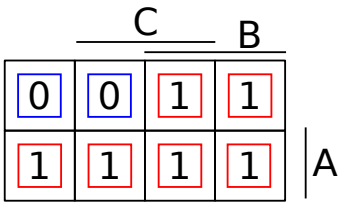
UDNF $= \overline{A}.B.C + A.B.\overline{C}$

UKNF $= (A+B+C).(A+B+\overline{C}).$
$.(A+\overline{B}+C).(\overline{A}+B+C).(\overline{A}+B+\overline{C}).$
$.(\overline{A}+\overline{B}+\overline{C})$



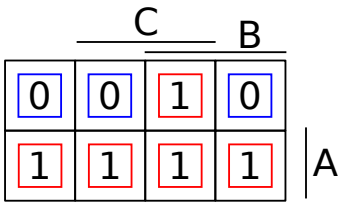
UDNF $= \overline{A}.B.C + A.\overline{B}.C + A.B.C$
$A.B.\overline{C}$

UKNF $= (A+B+C).(A+B+\overline{C}).$
$.(A+\overline{B}+C).(\overline{A}+B+C)$



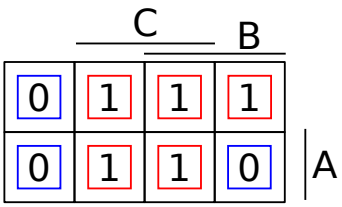
UDNF $= \overline{A}.B.C + \overline{A}.B.\overline{C} + A.\overline{B}.\overline{C} +$
$+A.\overline{B}.C + A.B.C + A.B.\overline{C}$

UKNF $= (A+B+C).(A+B+\overline{C})$



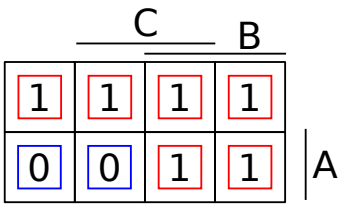
UDNF $= \overline{A}.B.C + A.\overline{B}.\overline{C} + A.\overline{B}.C$
$+A.B.C + A.B.\overline{C}$

UKNF $= (A+B+C).(A+B+\overline{C}).$
$.(A+\overline{B}+C)$



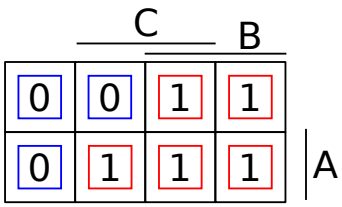
UDNF $= \overline{A}.\overline{B}.C + \overline{A}.B.C + \overline{A}.B.\overline{C} +$
$+A.\overline{B}.C + A.B.C$

UKNF $= (A+B+C).(\overline{A}+B+C).$
$.(\overline{A}+\overline{B}+C)$



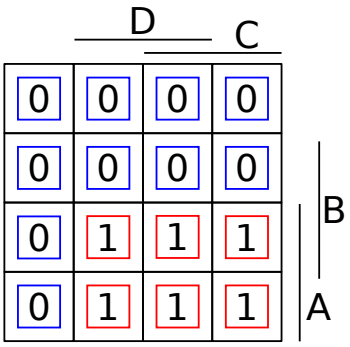
UDNF $= \overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C + \overline{A}.B.C +$
$+\overline{A}.B.\overline{C} + A.B.C + A.B.\overline{C}$

UKNF $= (\overline{A}+B+C).(\overline{A}+B+\overline{C})$



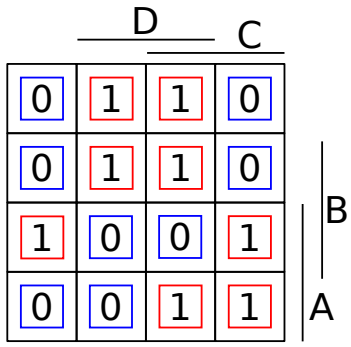
UDNF $= \overline{A}.B.C + \overline{A}.B.\overline{C} + A.\overline{B}.C +$
$+A.B.C + A.B.\overline{C}$

UKNF $= (A+B+C).(A+B+\overline{C}).$
$.(\overline{A}+B+C)$



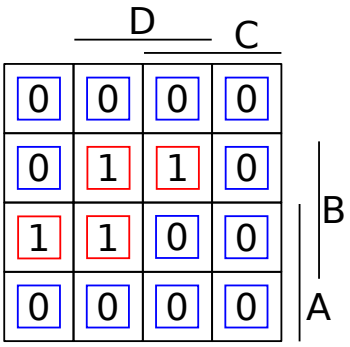
UDNF $= A.B.\overline{C}.D + A.B.C.D + A.B.C.\overline{D} + A.\overline{B}.\overline{C}.D + A.\overline{B}.C.D + A.\overline{B}.C.\overline{D}$

UKNF $= (A+B+C+D).(A+B+C+\overline{D}).(A+B+\overline{C}+\overline{D}).(A+B+\overline{C}+D).(A+\overline{B}+C+D).(A+\overline{B}+C+\overline{D}).(A+\overline{B}+\overline{C}+\overline{D}).(A+\overline{B}+\overline{C}+D).(\overline{A}+\overline{B}+C+D).(\overline{A}+B+C+D)$



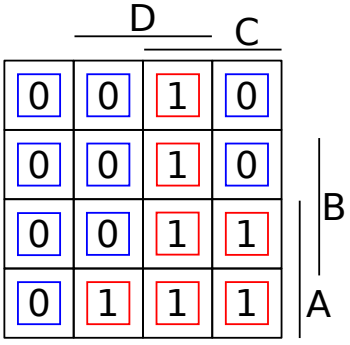
UDNF $= \overline{A}.\overline{B}.\overline{C}.D + \overline{A}.\overline{B}.C.D + \overline{A}.B.\overline{C}.D + \overline{A}.B.C.D + A.B.\overline{C}.\overline{D} + A.B.C.\overline{D} + A.\overline{B}.C.D + A.\overline{B}.C.\overline{D}$

UKNF $= (A+B+C+D).(A+B+\overline{C}+D).(A+\overline{B}+C+D).(A+\overline{B}+\overline{C}+\overline{D}).(\overline{A}+\overline{B}+C+\overline{D}).(\overline{A}+\overline{B}+\overline{C}+\overline{D}).(\overline{A}+B+C+D).(\overline{A}+B+C+\overline{D})$



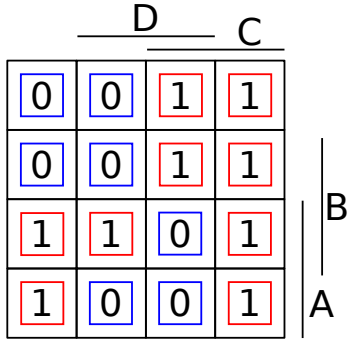
UDNF $= \overline{A}.B.\overline{C}.D + \overline{A}.B.C.D + A.B.\overline{C}.\overline{D} + A.B.\overline{C}.D$

UKNF $= (A+B+C+D).(A+B+C+\overline{D}).(A+B+\overline{C}+\overline{D}).(A+B+\overline{C}+D).(A+\overline{B}+C+D).(A+\overline{B}+\overline{C}+\overline{D}).(\overline{A}+\overline{B}+\overline{C}+\overline{D}).(\overline{A}+\overline{B}+\overline{C}+D).(\overline{A}+B+C+D).(\overline{A}+B+C+\overline{D}).(\overline{A}+B+\overline{C}+\overline{D}).(\overline{A}+B+\overline{C}+D)$



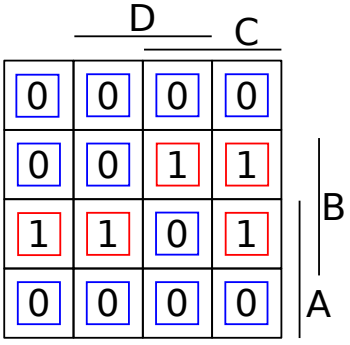
UDNF $= \overline{A}.\overline{B}.C.D + \overline{A}.B.C.D + A.B.C.D + A.B.C.\overline{D} + A.\overline{B}.\overline{C}.D + A.\overline{B}.C.D + A.\overline{B}.C.\overline{D}$

UKNF $= (A+B+C+D).(A+B+C+\overline{D}).(A+B+\overline{C}+D).(A+\overline{B}+C+D).(A+\overline{B}+\overline{C}+\overline{D}).(\overline{A}+\overline{B}+C+\overline{D}).(\overline{A}+\overline{B}+\overline{C}+D).(\overline{A}+B+C+D)$



UDNF $= \overline{A}.\overline{B}.C.D + \overline{A}.\overline{B}.C.\overline{D} + \overline{A}.B.C.D + \overline{A}.B.C.\overline{D} + A.B.\overline{C}.\overline{D} + A.B.\overline{C}.D + A.B.C.\overline{D} + A.\overline{B}.\overline{C}.\overline{D} + A.\overline{B}.C.\overline{D}$

UKNF $= (A+B+C+D).(A+B+C+\overline{D}).(A+B+\overline{C}+\overline{D}).(A+\overline{B}+C+D).(A+\overline{B}+C+\overline{D}).(\overline{A}+\overline{B}+C+D).(\overline{A}+B+C+\overline{D}).(\overline{A}+B+\overline{C}+\overline{D})$



UDNF $= \overline{A}.B.C.D + \overline{A}.B.C.\overline{D} + A.B.\overline{C}.\overline{D} + A.B.\overline{C}.D + A.B.C.\overline{D}$

UKNF $= (A+B+C+D).(A+B+C+\overline{D}).(A+B+\overline{C}+\overline{D}).(A+B+\overline{C}+D).(A+\overline{B}+C+D).(A+\overline{B}+C+\overline{D}).(\overline{A}+\overline{B}+C+D).(\overline{A}+B+C+\overline{D}).(\overline{A}+B+\overline{C}+\overline{D}).(\overline{A}+B+\overline{C}+D)$

Minimálna normálna forma (MDNF a MKNF)

Motivácia pre túto kapitolu je uvedená priamo v názve. V tejto kapitole sa naučíme, ako prepísať pôvodne zadanú Booleovskú funkciu do tzv. *minimálnej* normálnej formy, ktorá obsahuje iba najnutnejší počet Booleovských operácií a premenných. Inými slovami povedané zminimalizujeme zadanú funkciu tak, aby sme zachovali výstup pôvodnej funkcie (v pravdivostnej tabuľke), pričom na realizáciu budeme potrebovať najmenší možný počet premenných a operácií. Výhody minimálnej formy Booleovskej funckie sú zrejmé. Spracovanie minimálnej formy vyžaduje menej prostriedkov. Či už ide o ručné počítanie na papier v rámci štúdia alebo výpočet v rámci skutočného digitálneho (logického) systému realizovaného na hardvérovej úrovni. Digitálny systém pracujúci s minimálnym počtom premenných a logických operácií bude určite schopný priniesť vypočítaný výsledok rýchlejšie, čo prináša vyšší výpočtový výkon, ďalej bude na výpočet potrebovať menej energie, čo znamená dlhšiu životnosť batérie, resp. nižšie požiadavky na chladenie hardvérového systému. Ďalšou nespornou výhodou bude, že hardvér vykonávajúci výpočet bude obsahovať menej potencionálnych chýb, resp. porúch, a bude teda spoľahlivejší. V neposlednom rade bude určite takýto systém fyzicky menší a hlavne lacnejší ako v prípade pôvodnej nezminimalizovanej Booleovskej funkcie. Spomenuté výhody by sa určite dali ďalej rozširovať. Môžeme však s istotou tvrdiť, že pomocou vhodnej aplikácie Booleovskej algebry dokážeme vyvíjať lacnejší, úspornejší, výkonnejší a spoľahlivejší hardvér používaný v digitálnych (logických) systémoch.

Tak ako v predchádzajúcej kapitole, aj teraz budeme rozlišovať medzi disjunktívnou (MDNF) a konjuktívnou (MKNF) normálnou formou zápisu. Pozornému oku neunikne, že MDNF (MKNF) je v podstate len ÚDNF (ÚKNF) bez „nadbytočných" (redundantných) vstupných premenných a operácií. Disjunktívna normálna forma sa takisto môže označiť aj ako „súčet súčinov" (anglicky Sum-of-Products – SoP). Konjunktívnu normálnu formu môžeme analogicky nazvať „súčin súčtov" (anglicky Product-of-Sums – PoS). Tieto dve formy zápisu Booleovských funkcií sú kriticky dôležité. V minulosti sa podľa nich navrhovali integrované obvody pre programovateľné logické polia (PLA, PAL, CPLG). Všeobecný zápis MDNF je opísaný v rovnici (5.4). Ide o súčet členov nazývaných *minterma* – m_i, resp. čiastkový elementárny logický súčin.

$$f(a, b, c, \dots) = \sum m_i \tag{5.4}$$

Všeobecný zápis MKNF je definovaný v rovnici (5.5), ktorá predstavuje súčin členov nazývaných *maxterma*, čo predstavuje čiastkový elementárny logický súčet – M_i.

$$f(a, b, c, \dots) = \prod M_i \tag{5.5}$$

Postupov, ako sa dopracovať k minimálnej normálnej forme pôvodne zadanej Booleovskej funkcie, existuje niekoľko. My si ukážeme dva najzákladnejšie prístupy. Konkrétne algebrickú minimalizáciu aplikovaním zákonov a axióm Booleovskej algebry, a grafickú minimalizáciu pomocou Karnaughovej mapy. Oba postupy majú však spoločný cieľ – nájsť MDNF alebo MKNF. Matematicky sú tieto formy zápisu ekvivalentné. Z rôznych dôvodov, ktoré si popíšeme neskôr, sa však vo všeobecnosti preferuje zápis vo forme MDNF.

5.1 Algebrická minimalizácia

Algebrická minimalizácia, ako jej názov napovedá, sa opiera o aplikáciu základných zákonov, pravidiel a axiómov Booleovskej algebry pre zminimalizovanie pôvodnej Booleovskej funkcie. Táto metóda vyžaduje určitý nadhľad a skúsenosti, pričom nie vždy zaručuje úspešné zvládnutie úlohy. Z tohoto dôvodu sa nepoužíva v praxi veľmi často. Je však vynikajúca pre precvičenie si zručností či už z klasickej alebo Booleovskej algebry. Takisto je dôležité mať aspoň prehľad o základných postupoch, ak by sme pri riešení zložitej úlohy potrebovali upraviť výrazy do vhodnejšieho tvaru. Zoznam zákonov a axiómov Booleovskej algebry býva spomenutý na prednáškach a nie je preto cieľom tejto publikácie ich opäť spomínať. Ako príklad minimalizácie pomocou týchto zákonov si uvedieme nasledovnú minimalizáciu Booleovskej funkcie.

Úloha: Algebraicky zminimalizujte uvedenú funkciu $f = (A + \overline{B} + \overline{C}).(A + \overline{B}.C)$.
Riešenie:

$$\begin{aligned} f &= (A + \overline{B} + \overline{C}).(A + \overline{B}.C) \\ &= A.A + A.\overline{B} + A.\overline{C} + A.\overline{B}.C + \overline{B}.\overline{B}.C + B.C.\overline{C} \\ &= A + A.\overline{B} + A.\overline{C} + A.\overline{B}.C + \overline{B}.C + 0 \\ &= A\left(1 + \overline{B} + \overline{C} + \overline{B}.C\right) + \overline{B}.C \\ &= A.1 + \overline{B}.C \\ &= A + \overline{B}.C \end{aligned}$$

Popis uvedeného riešenia je nasledovný. V prvom kroku použijeme *distributívny* zákon a roznásobíme zátvorky systémom „každý s každým". Nasledujúcim krokom bude aplikovanie zákona *idempotencie* na členy $A.A$ a $\overline{B}.\overline{B}.C$ a následne použitie zákona *komplementárnosti* na člen $B.C.\overline{C}$. V treťom riadku úpravy opäť použijeme distributívny zákon a vyberieme pred zátvorku premennú A, po ktorej nám vnútri zátvorky ostáva logická jednotka a zvyšné členy. Vo štvrtom kroku aplikujeme zákon *agresivity a neutrality* na obsah zátvorky. Inak povedané, logická 1 plus čokoľvek sa bude vždy rovnať jednej. Výsledný zápis zminimalizovanej funkcie v poslednom riadku nám demonštruje rozsah minimalizácie. Namiesto šestich operandov a ôsmich operátorov (vrátane negácií) dokážeme identický výstup dosiahnuť s len tromi operandami a tromi operátormi. Pre istotu si môžeme spraviť skúšku správnosti pomocou pravdivostnej tabuľky.

Úloha: Algebricky zminimalizujte uvedenú funkciu $f = \overline{A}.\overline{C} + B.\overline{C} + \overline{A}.B.C + A.C$.
Riešenie:

$$\begin{aligned} f &= \overline{A}.\overline{C} + B.\overline{C} + \overline{A}.B.C + A.C \\ &= \overline{A}.\overline{C} + B.\overline{C} + C(\overline{A}.B + A) \\ &= \overline{A}.\overline{C} + B.\overline{C} + C.(B + A) \\ &= \overline{A}.\overline{C} + B.\overline{C} + B.C + A.C \\ &= \overline{A}.\overline{C} + B(\overline{C} + C) + A.C \\ &= \overline{A}.\overline{C} + B.1 + A.C \\ &= B + A.C + \overline{A}.\overline{C} \\ &= B + \overline{A \oplus C} \end{aligned}$$

Slovný opis postupu môžeme zhrnúť do nasledujúcich viet. V prvom kroku vyberieme pred zátvroku premennú C z posledných dvoch členov výrazu. Obsah zátvorky môžeme upraviť v treťom riadku podľa zákona absorbcie. Následne roznásobíme zátvorku podľa distributívneho zákona. V ďalšom kroku môžeme opäť vyňať pred zátvorku premennú B zo stredných členov funkcie. Na obsah zátvorky aplikujeme zákon komplementárnosti, podľa ktorého sa bude vždy rovnať logickej jednotke. Predposledný riadok je MDNF pôvodne zadanej funkcie. V poslednej úprave využijeme definíciu negovaného exkluzívneho logického súčtu (operácia NXOR, resp. XNOR). Posledný riadok však nemôžeme nazvať MDNF, nakoľko tvar zápisu nespĺňa kritérium použitia iba elementárnych logických operácií. Opäť môžeme porovnať zložitosť pôvodneho a zminimalizovaného zápisu Booleovskej funkcie. Dvanásť operácií v rámci štyroch členov výrazu oproti dvom operáciám po minimalizácii.

5.2 Grafická minimalizácia

Pri grafickej minimalizácii využívame Karnaughovu mapu. Pre efektívny proces minimalizácie je preto nutné mať spoľahlivo osvojenú tvorbu a vyplnenie K-mapy. V tejto kapitole si rozšírime súbor operácií smerujúcich k minimalizácii pôvodne zadanej Booleovskej funckie. Najdôležitejšou časťou procesu minimalizácie je vytvorenie tzv. „slučiek", ktoré budú neskôr definovať výslednú minimálnu normálnu formu. Ide v podstate o vytvorenie skupiny susedných jednotkových alebo nulových bodov funkcie. Pre tvorbu spomenutých slučiek platia nasledujúce pravidlá, ktoré nie sú zoradené podľa priority.

- Počet bodov v slučke musí byť mocninou 2^N. Teda 1, 2, 4, 8, 16 ...
- Maximalizujeme počet bodov v slučke
- Minimalizujeme počet slučiek
- Každý jednotkový/nulový bod funkcie musí byť aspoň v jednej slučke
- Slučky sa môžu prekrývať
- Slučky nesmú byť v diagonálnom smere
- K-mapa je interpretovaná ako trojrozmerná, resp. vonkajšie hrany K-mapy sú susediace.

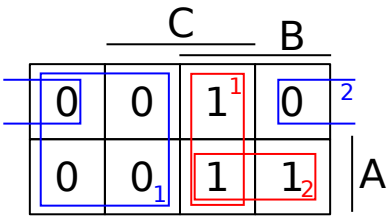
Aplikáciu uvedených pravidiel pre tvorbu slučiek v K-mape môžeme vidieť v nasledujúcich príkladoch.

5.3 Vzorový príklad

Úloha: Graficky zminimalizujte uvedenú funkciu $f = \overline{A + \overline{B}}.C + (A + C).B$. Zminimalizovanú funkciu zapíšte v tvare MDNF a MKNF.
Riešenie: Pre zadanú funkciu vytvoríme pravdivostnú tabuľku (Tab. 5.3) a vyplníme K-mapu (Obr. 5.1).

Tab. 5.1: Pravdivostná tabuľka vzorového príkladu

A	B	C	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



Obr. 5.1: Karnaughova mapa vzorového príkladu

Zamerajme sa najprv na opis slučiek pre jednotkové body funkcie, pracujeme teda na odvodení MDNF. Prvá aj druhá slučka sú dvojice v ortogonálnom smere. Podliehajú všetkým vyššie uvedeným pravidlám a ide v podstate o veľmi jednoduchý a základný príklad. Slučky pre nulové body funkcie reprezentujúce budúcu MKNF predstavujú príklad, ktorý ukazuje aplikáciu pravidla ohľadom 3D interpretácie K-mapy. Druhá slučka je dvojica *susediacich* nulových bodov, nakoľko ľavý a pravý okraj K-mapy sú v skutočnosti spojené (ako aj horný a dolný okraj).
Nasledujúcim krokom je opis slučiek pomocou vstupných premenných. Každá slučka vlastne predstavuje jednu mintermu/maxtermu výslednej funkcie, a preto sa snažíme minimalizovať celkový počet slučiek. Opis slučiek je veľmi podobný opisu jednotkových/nulových bodov funkcie v prípade ÚDNF/ÚKNF, pracujeme teda s pruhmi v K-mape. Avšak s tým rozdielom, že vstupná premenná, ktorá v práve opisovanej slučke mení svoju logickú hodnotu bude v opise vynechaná. Preto sa snažíme vytvárať čo najväčšie slučky, aby čo najviac vstupných premenných bolo vynechaných

z finálneho zápisu. MDNF funkcie z príkladu je zapísaná v (5.6). Slučka číslo 1 opisuje svoju mintermu nasledovne. Vstupná premenná A bude zo zápisu vynechaná, pretože v hornej časti slučky $A = 0$ a v dolnej časti slučky $A = 1$. Mení teda svoju logickú hodnotu a nevieme ju jednoznačne opísať. Premenná B sa nachádza v oboch častiach slučky v logickej jednotke, bude preto zapísaná v minterme priamo (bez negácie). Premenná C sa tiež nachádza v oboch častiach slučky v logickej jednotke. Preto bude takisto zapísaná v minterme bez negácie. Slučku číslo 2 zapíšeme do mintermy identickým postupom. Teda postupne prejdeme všetky vstupné premenné a kontrolujeme, či vyšetrovaná premenná mení alebo nemení svoju logickú hodnotu v rámci slučky. Premenná A je v oboch častiach v logickej jednotke, bude preto do mintermy zapísaná priamo. Premenná B sa takisto v oboch častiach slučky rovná logickej jednotke, zapíšeme ju preto priamo. Premenná C je v ľavej časti slučky v stave logickej jednotky, ale v pravej časti je v stave logickej nuly. Bude preto z opisu mintermy vynechaná. Funkciu (5.6) by sme mohli ďalej upraviť vyňatím premennej B pred zátvorku a ešte viac tak znížiť počet operácií. Tento spôsob zápisu by sme však už nemohli nazývať MDNF, nakoľko nespĺňa matematickú definíciu takéhoto zápisu.

$$MDNF(f) = \overbrace{B.C}^{1} + \overbrace{A.B}^{2} \tag{5.6}$$

Odvodenie MKNF je zapísané výrazom (5.7). Maxterma prislúchajúca slučke číslo 1 je tvorená štvoricou bodov v K-mape. Premenná A je v hornej polovici slučky v stave logickej nuly a v dolnej polovici sa rovná logickej jednotke. Nevieme ju preto jednoznačne opísať a bude zo zápisu maxtermy vynechaná. Vstupná premenná B ja v celej slučke v stave logickej nuly, a preto ju zapíšeme do maxtermy priamo. Premenná C v opisovanej slučke mení svoj stav a bude vynechaná, pretože v ľavej strane slučky je $C = 0$, ale v pravej polovici $C = 1$. Môžeme teda zhrnúť, že prvá slučka/maxterma je definovaná iba vstupnou premennou B. V druhej slučke mení svoj stav iba vstupná premenná B. V ľavej časti rozpoltenej slučky je táto premenná v stave logickej nuly, kým v pravej časti je v stave logickej jednotky. Premenné A a C sa v tejto slučke rovnajú logickej nule, a preto budú v maxterme zapísané priamo.

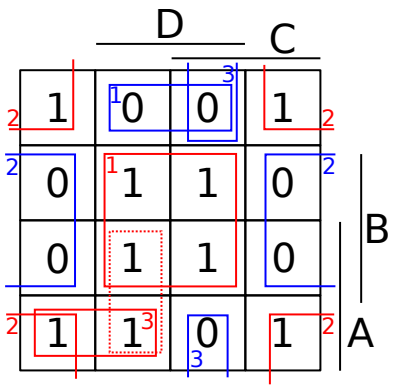
$$MKNF(f) = \overbrace{B}^{1} . \overbrace{(A + C)}^{2} \tag{5.7}$$

Matematicky sú minimálne formy funkcie ekvivalentné s pôvodnou funkciou, teda $MDNF(f) = MKNF(f) = f$.

Úloha: Graficky zminimalizujte uvedenú funkciu $f = A.\overline{C}.(D + \overline{B}.C) + B \oplus \overline{D}$.
Riešenie:

Tab. 5.2: Pravdivostná tabuľka minimalizovanej funkcie

A	B	C	D	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1



Obr. 5.2: Karnaughova mapa minimalizovanej funkcie

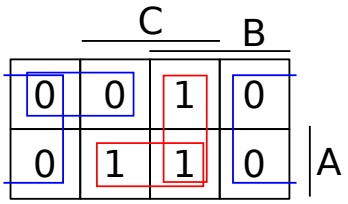
Po vyplnení Karnaughovej mapy a vytvorení slučiek pre MDNF a MKNF môžeme vidieť, že existujú dve riešenia pre MDNF. Všimnime si druhú slučku tvorenú štvoricou jednotkových bodov v rohoch K-mapy. Opäť využívame 3D interpretáciu K-mapy. Tretia slučka je dvojica jednotkových bodov a môže byť tvorená dvomi rôznymi dvojicami políčok.

$$MDNF(f) = \overbrace{\overline{B}.\overline{D}}^{1} + \overbrace{\overline{B}.\overline{D}}^{2} + \overbrace{A.\overline{B}.\overline{C}}^{3}$$
$$= \overbrace{\overline{B}.\overline{D}}^{1} + \overbrace{\overline{B}.\overline{D}}^{2} + \overbrace{A.\overline{C}.D}^{3'}$$
$$MKNF(f) = \overbrace{(A + B + \overline{D})}^{1}.\overbrace{(\overline{B} + D)}^{2}.\overbrace{(B + \overline{C} + \overline{D})}^{3}$$

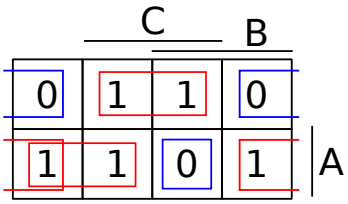
V K-mape sme vyznačili prvú možnosť plnou čiarou a druhú možnosť bodkovanou čiarou. Obe možnosti sú rovnako dobré a správne. Súčet minterm $B.D$ a $\overline{B}.\overline{D}$ je učebnicová definícia operácie NXOR (resp. XNOR) a po prepise výrazu s použitím NXOR už ide o ďalšiu *optimalizáciu* funkcie (vysvetlíme nekôr). Po tejto úprave však už nemôžeme spomínaný výraz nazývať MDNF.
Uvedený príklad zároveň ilustruje situáciu, kedy má funkcia viac ako jedno správne riešenie pre MDNF. MKNF je taktiež tvorená tromi slučkami. Za zmienku stoja rozpoltené slučky číslo 2 a 3, ktoré sa znova opierajú o trojrozmernú interpretáciu K-mapy.

5.4 Príklady

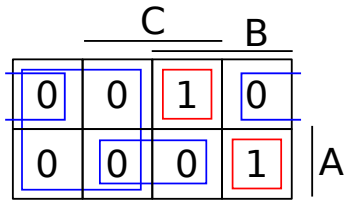
Nájdite MDNF a MKNF pre Karnaughove mapy z predchádzajúcej kapitoly. Niektoré K-mapy ponúkajú viac správnych riešení, teda porozmýšľajte nad každou K-mapou a prediskutujte možné riešenia.



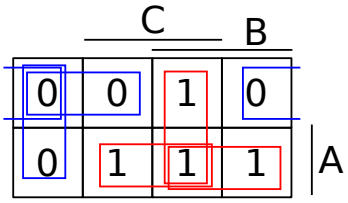
MDNF $= A.C + B.C$
MKNF $= C.(A + B)$



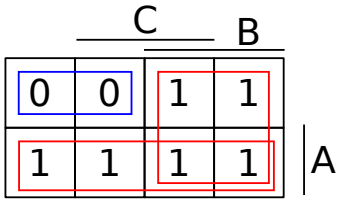
MDNF $= A.\overline{B} + A.\overline{C} + \overline{A}.C$
MKNF $= (\overline{A} + \overline{B} + \overline{C}).(A + C)$



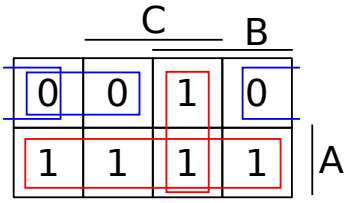
MDNF $= \overline{A}.B.C + A.B.\overline{C}$
MKNF $= B.(\overline{A} + \overline{C}).(A + C)$



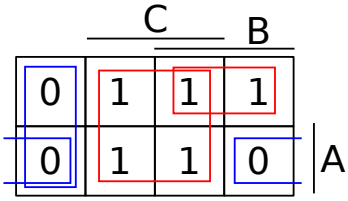
MDNF $= A.C + A.B + B.C$
MKNF $= (A + C).(A + B).$
$.(B + C)$



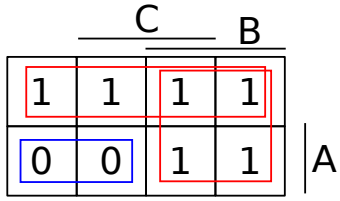
MDNF $= A + B$
MKNF $= A + B$



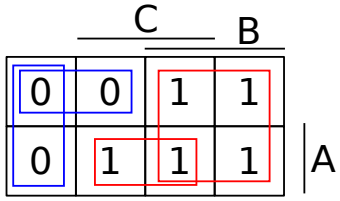
MDNF $= A + B.C$
MKNF $= (A + C).(A + B)$



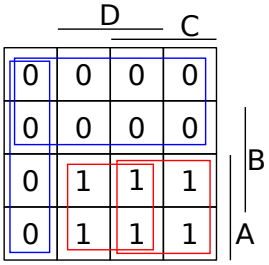
MDNF $= \overline{A}.B + C$
MKNF $= (B + C).(\overline{A} + C)$



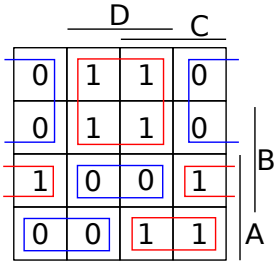
MDNF $= \overline{A} + B$
MKNF $= \overline{A} + B$



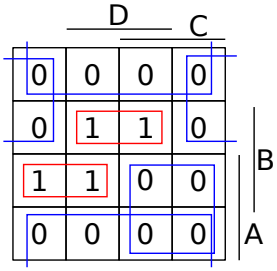
MDNF $= B + A.C$
MKNF $= (A + B).(B + C)$



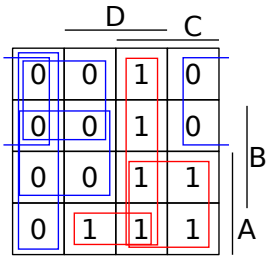
MDNF $= A.D + A.C$
MKNF $= A.(C + D)$



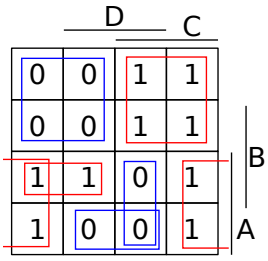
MDNF $= \overline{A}.D + A.\overline{B}.C + A.B.\overline{D}$
MKNF $= (A + D).(\overline{A} + \overline{B} + \overline{D}).$
$.(\overline{A} + B + C)$



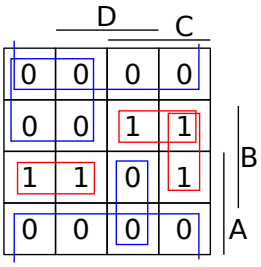
MDNF $= A.B.\overline{C} + \overline{A}.B.D$
MKNF $= B.(\overline{A} + \overline{C}).(A + D)$



MDNF $= A.C + C.D + A.\overline{B}.D$
MKNF $= (A + D).(A + C).$
$.(\overline{B} + C).(C + D)$



MDNF $= A.\overline{D} + \overline{A}.C + A.B.\overline{C}$
MKNF $= (A + C).(\overline{A} + \overline{C} + \overline{D}).$
$.(\overline{A} + B + \overline{D})$



MDNF $= A.B.\overline{C} + B.C.\overline{D} + \overline{A}.B.C$
MKNF $= B.(A + C).$
$.(\overline{A} + \overline{C} + \overline{D})$

Optimalizácia MDNF a MKNF

Ako prvé si musíme opísať rozdiel medzi minimalizáciou a optimalizáciou. Optimalizácia je proces, cieľom ktorého je ďalej upraviť už zminimalizovanú funkciu (v tvare MDNF alebo MKNF) do takého tvaru, aby vo výsledku funkcia obsahovala menej operácií a písmen, resp. operátorov a operandov. Druhým cieľom optimalizácie je zjednodušenie výsledného hardvéru, teda zníženie celkového počtu tranzistorov, ktoré budú vykonávať daný výpočet. V praxi sa však veľmi často stáva, že využívame kombináciu oboch prístupov. Postup, ako určiť potrebný počet tranzistorov pre zadanú Booleovskú funkciu, si ukážeme až v nasledujúcich kapitolách. Preto sa nateraz obmedzíme iba na použitie De Morganových (DM) pravidiel a základných axióm Booleovskej algebry. Totiž už len samotná aplikácia De Morganových zákonov na MDNF/MKNF formu Booleovskej funkcie predstavuje dostatočnú optimalizáciu a ďalšie význammné šetrenie prostriedkov (oneskorenie/čas výpočtu, spotreba energie, veľkosť/zložitosť hardvéru atď.). Zamerajme sa na odvodené MDNF a MKNF Booleovských funkcií z predchádzajúcej kapitoly a aplikujme na ne De Morganove pravidlá. Našim cieľom je teda upraviť zadané výrazy do tvaru len s jediným typom operátora, a to negovaný logický súčin alebo negovaný logický súčet. Inými slovami, upravíme výrazy do takého tvaru, aby boli realizované iba pomocou operácie NAND alebo NOR.

Hoci sú MDNF, MKNF a aj pôvodná Booleovská funkcia matematicky ekvivalentné (majú identický výstup v pravdivostnej tabuľke), ako už bolo spomenuté, hlavná motivácia je v „zjednodušovaní" zápisu výrazov čo prináša zjednodušenie elektronického hardvéru realizujúceho výpočty. Optimalizovaná MDNF či MKNF je takisto matematicky ekvivalentná s predchádzajúcimi formami zápisu.

Ľudskému oku by sa dokonca mohlo zdať, že optimalizovaný zápis sa javí niekedy zložitejší pre pridané negácie a podobne. Avšak MDNF alebo MKNF po úprave De Morganovými zákonmi skutočne vyžadujú nižší počet tranzistorov pre praktickú hardvérovú realizáciu v CMOS technológii (v súčasnosti absolútne najčastejší spôsob výroby integrovaných obvodov na čipe resp. tranzistorov na kremíku).

6.1 Vzorový príklad

Prvou výslednou funkciou v predchádzajúcej kapitole bol výstup algebrickej minimalizácie. V nasledujúcich výrazoch si ukážeme praktickú aplikáciu De Morganovych pravidiel. V prvom prípade sa snažíme zápis funkcie premeniť tak, aby sme použili iba operáciu NAND. V druhom zápise je našim cieľom prepísať ten istý výraz do tvaru, kde použijeme iba operáciu NOR.

$$NAND(MDNF) = A + \overline{B}.C = \overline{(\overline{A}).(\overline{\overline{B}.C})}$$

$$NOR(MDNF) = A + \overline{B}.C = A + \overline{B + \overline{C}} = \overline{\overline{\overline{A + \overline{B + \overline{C}}}}}$$

V prvej funkcii sme pre prehľadnosť pridali (nepotrebné) zátvorky. Aplikovali sme De Morganovo pravidlo nasledovne. Operátor logického súčtu sme prepísali na logický súčin. Následne sme pridali negácie pre oba operandy (premenná A a člen $\overline{B}.C$) a nakoniec ešte jednu negáciu nad celú pravú stranu rovnice. Ide o učebnicovú aplikáciu De Morganovho zákona, ktorá je vysvetlená v podrobnom detaile v druhej kapitole.

Druhú možnosť aplikácie De Morganových zákonov môžeme opísať nasledovne. Operátor logického súčinu prepíšeme na logický súčet a opäť pridáme negácie pre oba operandy t. j. premenná $\overline{B}$ a premenná C sa stanú B a $\overline{C}$. Ďalej pridáme druhú úroveň negácie nad celý člen. Keďže naším cieľom je realizácia funkcie iba za pomoci operácie NOR, musíme pridať nad celú pravú stranu rovnice dve negácie. Tento krok nijako nemení výstupnú hodnotu funkcie a zároveň nám zabezpečí spomínanú podmienku.

Optimalizáciu druhej funkcie z predchádzajúcej kapitoly zapíšeme nasledovne.

$$NAND(MDNF) = B + A.C + \overline{A}.\overline{C} = \overline{(\overline{B}).(\overline{A.C}).(\overline{\overline{A}.\overline{C}})}$$

$$NOR(MDNF) = B + A.C + \overline{A}.\overline{C} = B + \overline{\overline{A} + \overline{C}} + \overline{A + C} = \overline{\overline{B + \overline{\overline{A} + \overline{C}} + \overline{A + C}}}$$

Optimalizáciu MDNF tretej funkcie z predchádzajúcej kapitoly zapíšeme nasledovne. Ide o funckiu zminimalizovanú pomocou Karnaghovej mapy.

$$NAND(MDNF) = B.C + A.B = \overline{\overline{(B.C)}.\overline{(A.B)}}$$
$$NOR(MDNF) = B.C + A.B = \overline{\overline{B} + \overline{C}} + \overline{\overline{A} + \overline{B}} = \overline{\overline{\overline{B} + \overline{C}} + \overline{\overline{A} + \overline{B}}}$$

MKNF tej istej minimalizácie pomocou K-mapy zapíšeme v nasledujúcich riadkoch. V tomto prípade sme nútení pridávať dvojitú negáciu pri aplikácii De Morganovho pravidla s cieľom realizovať ju pomocou operácie NAND. Môžeme teda tvrdiť, že úprava MDNF do tvaru obsahujúceho operáciu NAND je priamočiarejšia. Rovnako ako úprava MKNF do tvaru opierajúceho sa o operáciu NOR. Z tohto dôvodu býva jedna z oboch možností celkovým najefektívnejším riešením funkcie pre hardvérovú realizáciu.

$$NAND(MKNF) = B.(A + C) = B.\overline{(\overline{A}.\overline{C})} = \overline{B.\overline{\overline{A}.\overline{C}}}$$
$$NOR(MKNF) = B.(A + C) = \overline{\overline{B} + \overline{(A + C)}}$$

6.2 Príklady

Pre MDNF a MKNF Booleovských funkcií z predchádzajúcej kapitoly odvoďte realizáciu iba pomocou operácie NAND a iba pomocou operácie NOR. Ak je to možné, vyberte najefektívnejšiu realizáciu danej Booleovskej funkcie.

$$\mathbf{MDNF} = A.C + B.C$$
$$NAND(MDNF) = \overline{\overline{A.C}.\overline{B.C}}$$
$$NOR(MDNF) = \overline{\overline{\overline{A} + \overline{C}} + \overline{\overline{B} + \overline{C}}}$$

$$\mathbf{MKNF} = C.(A + B)$$
$$NAND(MKNF) = \overline{C.\overline{\overline{A}.\overline{B}}}$$
$$NOR(MKNF) = \overline{\overline{C} + \overline{A + B}}$$

$$\mathbf{MDNF} = A.\overline{B} + A.\overline{C} + \overline{A}.C$$
$$NAND(MDNF) = \overline{\overline{A.\overline{B}}.\overline{A.\overline{C}}.\overline{\overline{A}.C}}$$
$$NOR(MDNF) = \overline{\overline{(\overline{A} + B)} + \overline{(\overline{A} + C)} + \overline{(A + \overline{C})}}$$

$$\mathbf{MKNF} = (A + D).(A + C).(\overline{B} + C).(C + D)$$
$$NAND(MKNF) = \overline{\overline{(\overline{A}.\overline{D})}.\overline{(\overline{A}.\overline{C})}.\overline{(B.\overline{C})}.\overline{(\overline{C}.\overline{D})}}$$
$$NOR(MKNF) = \overline{\overline{(A + D)} + \overline{(A + C)} + \overline{(\overline{B} + C)} + \overline{(C + D)}}$$

$$\mathbf{MDNF} = A.C + A.B + B.C$$
$$NAND(MDNF) = \overline{\overline{A.C}.\overline{A.B}.\overline{B.C}}$$
$$NOR(MDNF) = \overline{\overline{(\overline{A} + \overline{C})} + \overline{(\overline{A} + \overline{B})} + \overline{(\overline{B} + \overline{C})}}$$

$$\mathbf{MKNF} = (A + C).(A + B).(B + C)$$
$$NAND(MKNF) = \overline{\overline{(\overline{A}.\overline{C})}.\overline{(\overline{A}.\overline{B})}.\overline{(\overline{B}.\overline{C})}}$$
$$NOR(MKNF) = \overline{\overline{(A + C)} + \overline{(A + B)} + \overline{(B + C)}}$$

$$\mathbf{MDNF} = A.C + B.\overline{C} + \overline{A}.\overline{B}$$
$$NAND(MDNF) = \overline{\overline{A.C}.\overline{B.C}.\overline{A.B}}$$
$$NOR(MDNF) = \overline{\overline{(\overline{A} + \overline{C})} + \overline{(\overline{B} + C)} + \overline{(A + B)}}$$

$$\mathbf{MKNF} = (\overline{A} + B + C).(A + \overline{B} + \overline{C})$$
$$NAND(MKNF) = \overline{\overline{(A.\overline{B}.\overline{C})}.\overline{(\overline{A}.B.C)}}$$
$$NOR(MKNF) = \overline{(\overline{\overline{A} + B + C}) + (\overline{A + \overline{B} + \overline{C}})}$$

$$\mathbf{MDNF} = \mathbf{MKNF} = A + B$$
$$NAND(MDNF) = NAND(MKNF) = \overline{\overline{A}.\overline{B}}$$
$$NOR(MDNF) = NOR(MKNF) = \overline{\overline{A + B}}$$

$$\mathbf{MDNF} = A + B.C$$
$$NAND(MDNF) = \overline{\overline{A}.\overline{B.C}}$$
$$NOR(MDNF) = \overline{A + \overline{(\overline{B} + \overline{C})}}$$

$$\mathbf{MKNF} = (A + C).(A + B)$$
$$NAND(MKNF) = \overline{\overline{A.\overline{C}}.\overline{A.\overline{B}}}$$
$$NOR(MKNF) = \overline{\overline{(A + C)} + \overline{(A + B)}}$$

$$\mathbf{MDNF} = \overline{A}.B + C$$
$$NAND(MDNF) = \overline{\overline{\overline{A}.B}.\overline{C}}$$
$$NOR(MDNF) = \overline{\overline{(A + \overline{B})} + C}$$

$$\mathbf{MKNF} = (B + C).(\overline{A} + C)$$
$$NAND(MKNF) = \overline{\overline{(\overline{B}.\overline{C})}.\overline{(A.\overline{C})}}$$
$$NOR(MKNF) = \overline{\overline{(B + C)} + \overline{(\overline{A} + C)}}$$

$$\mathbf{MDNF} = \mathbf{MKNF} = \overline{A} + B$$
$$NAND(MDNF) = NAND(MKNF) = \overline{A.\overline{B}}$$
$$NOR(MDNF) = NOR(MKNF) = \overline{\overline{\overline{A} + B}}$$

$\mathbf{MDNF} = B + A.C$
$NAND(MDNF) = \overline{\overline{B}.\overline{A.C}}$
$NOR(MDNF) = \overline{\overline{B + (\overline{A} + \overline{C})}}$

$\mathbf{MKNF} = (A + B).(B + C)$
$NAND(MKNF) = \overline{\overline{\overline{A}.\overline{B}.\overline{B}.\overline{C}}}$
$NOR(MKNF) = \overline{\overline{(A + B)} + \overline{(B + C)}}$

$\mathbf{MDNF} = A.D + A.C$
$NAND(MDNF) = \overline{\overline{A.D}.\overline{A.C}}$
$NOR(MDNF) = \overline{\overline{(\overline{A} + \overline{D})} + \overline{\overline{A} + \overline{C}}}$

$\mathbf{MKNF} = A.(C + D)$
$NAND(MKNF) = \overline{\overline{A.\overline{(\overline{C}.\overline{D})}}}$
$NOR(MKNF) = \overline{\overline{A} + \overline{(C + D)}}$

$\mathbf{MDNF} = \overline{A}.D + A.\overline{B}.C + A.B.\overline{D}$
$NAND(MDNF) = \overline{\overline{\overline{A}.D}.\overline{A.\overline{B}.C}.\overline{A.B.\overline{D}}}$
$NOR(MDNF) = \overline{\overline{(A + \overline{D})} + \overline{(\overline{A} + B + \overline{C})} + \overline{(\overline{A} + \overline{B} + D)}}$

$\mathbf{MKNF} = (A + D).(\overline{A} + \overline{B} + \overline{D}).(\overline{A} + B + C)$
$NAND(MKNF) = \overline{\overline{\overline{A}.\overline{D}}.\overline{A.B.D}.\overline{A.\overline{B}.\overline{C}}}$
$NOR(MKNF) = \overline{\overline{(A + D)} + \overline{(\overline{A} + \overline{B} + \overline{D})} + \overline{(\overline{A} + B + C)}}$

$\mathbf{MDNF} = A.B.\overline{C} + \overline{A}.B.D$
$NAND(MDNF) = \overline{\overline{A.B.\overline{C}}.\overline{\overline{A}.B.D}}$
$NOR(MDNF) = \overline{\overline{(\overline{A} + \overline{B} + C)} + \overline{(A + \overline{B} + \overline{D})}}$

$\mathbf{MKNF} = B.(\overline{A} + \overline{C}).(A + D)$
$NAND(MKNF) = \overline{\overline{B.\overline{A.C}}.\overline{\overline{A}.\overline{D}}}$
$NOR(MKNF) = \overline{\overline{B} + \overline{\overline{A} + \overline{C}} + \overline{(A + D)}}$

$$\mathbf{MDNF} = A.C + C.D + A.\overline{B}.D$$
$$NAND(MDNF) = \overline{\overline{A.C}.\overline{C.D}.\overline{A.\overline{B}.D}}$$
$$NOR(MDNF) = \overline{\overline{\overline{A} + \overline{C}} + \overline{\overline{C} + \overline{D}} + \overline{\overline{A} + B + \overline{D}}}$$

$$\mathbf{MKNF} = (A + D).(A + C).(\overline{B} + C).(C + D)$$
$$NAND(MKNF) = \overline{\overline{A.\overline{D}}.\overline{A.\overline{C}}.\overline{B.\overline{C}}.\overline{C.\overline{D}}}$$
$$NOR(MKNF) = \overline{\overline{(A + D)} + \overline{(A + C)} + \overline{(\overline{B} + C)} + \overline{(C + D)}}$$

$$\mathbf{MDNF} = A.\overline{D} + \overline{A}.C + A.B.\overline{C}$$
$$NAND(MDNF) = \overline{\overline{A.\overline{D}}.\overline{\overline{A}.C}.\overline{A.B.\overline{C}}}$$
$$NOR(MDNF) = \overline{\overline{(\overline{A} + D)} + \overline{(A + \overline{C})} + \overline{(\overline{A} + \overline{B} + C)}}$$

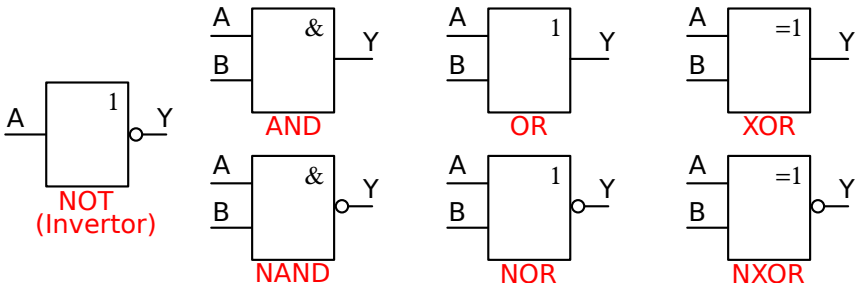
$$\mathbf{MKNF} = (A + C).(\overline{A} + \overline{C} + \overline{D}).(\overline{A} + B + \overline{D})$$
$$NAND(MKNF) = \overline{\overline{(\overline{A}.\overline{C})}.\overline{(A.C.D)}.\overline{(A.\overline{B}.D)}}$$
$$NOR(MKNF) = \overline{\overline{A + C} + \overline{(\overline{A} + \overline{C} + \overline{D})} + \overline{(\overline{A} + B + \overline{D})}}$$

$$\mathbf{MDNF} = A.B.\overline{C} + B.C.\overline{D} + \overline{A}.B.C$$
$$NAND(MDNF) = \overline{\overline{A.B.\overline{C}}.\overline{B.C.\overline{D}}.\overline{\overline{A}.B.C}}$$
$$NOR(MDNF) = \overline{\overline{(\overline{A} + \overline{B} + C)} + \overline{(\overline{B} + \overline{C} + D)} + \overline{(A + \overline{B} + \overline{C})}}$$

$$\mathbf{MKNF} = B.(A + C).(\overline{A} + \overline{C} + \overline{D})$$
$$NAND(MKNF) = \overline{B.\overline{(\overline{A}.\overline{C})}.\overline{(A.C.D)}}$$
$$NOR(MKNF) = \overline{\overline{B} + \overline{(A + C)} + \overline{(\overline{A} + \overline{C} + \overline{D})}}$$

Návrh kombinačných obvodov na hradlovej úrovni

V predchádzajúcej kapitole sme sa naučili odvodiť pomocou matematiky hardvérovo najefektívnejší spôsob zápisu Booleovskej funkcie. Najčastejšie ide o zápis MDNF pomocou operácie NAND či zápis MKNF pomocou operácie NOR. V tejto kapitole sa naučíme matematický zápis prekresliť do elektrickej schémy opisujúcej elektronický obvod, ktorý bude hardvérovo vykonávať daný výpočet. Prejdeme tak z abstraktnej matematiky do reálneho fyzikálneho sveta dizajnu digitálneho hardvéru. Matematické operácie z Booleovskej algebry v skutočnom hardvéri vykonávajú tzv. *logické hradlá* (anglicky „logic gate"). *Logické hradlo* alebo tiež *logický člen* je základná stavebná bunka každého zložitejšieho digitálneho (logického) systému. Inými slovami, každý komplexný digitálny systém vieme prekresliť do schematickej podoby tvorenej iba základnými logickými hradlami. Logické hradlo sa vnútri skladá z vhodne zapojených tranzistorov. Toto učivo je však témou až nasledujúcej kapitoly. Schematický opis digitálneho systému, ktorý sa skladá z logických hradiel, častokrát nazývame *hradlová schéma* alebo schéma na *hradlovej úrovni*. Teraz si predstavíme schematické značky jednotlivých logických hradiel, ktoré vykonávajú už známe Booleovské funkcie, resp. operácie. Tieto sú zobrazené na Obr. 7.1. Ide o dvojvstupové hradlá, teda vedia vykonať danú operáciu pre dve jednobitové premenné. Analogicky by sme však ľahko vedeli nakresliť hradlo pre N vstupných bitov.



Obr. 7.1: Schematické značky základných logických hradiel

Úloha: Nakreslite schému na hradlovej úrovni pre vybranú Booleovskú funkciu z predchádzajúcej kapitoly.
Riešenie: Každá ručne kreslená elektrická schéma je originál. Desať ľudí zrejme nakreslí ten istý obvod desiatimi rôznymi spôsobmi, a preto neexistuje jediný správny postup kreslenia. Existuje len súbor odporúčaní, ako by mala čitateľná a prehľadná hradlová schéma logického systému vyzerať. Na predmete „Logické systémy" učíme tvorbu elektrickej hradlovej schémy nasledovne:

1. V ľavej hornej časti papiera nakreslíme vertikálne čiary podľa počtu vstupných premenných (každá čiara bude predstavovať jednu premennú).

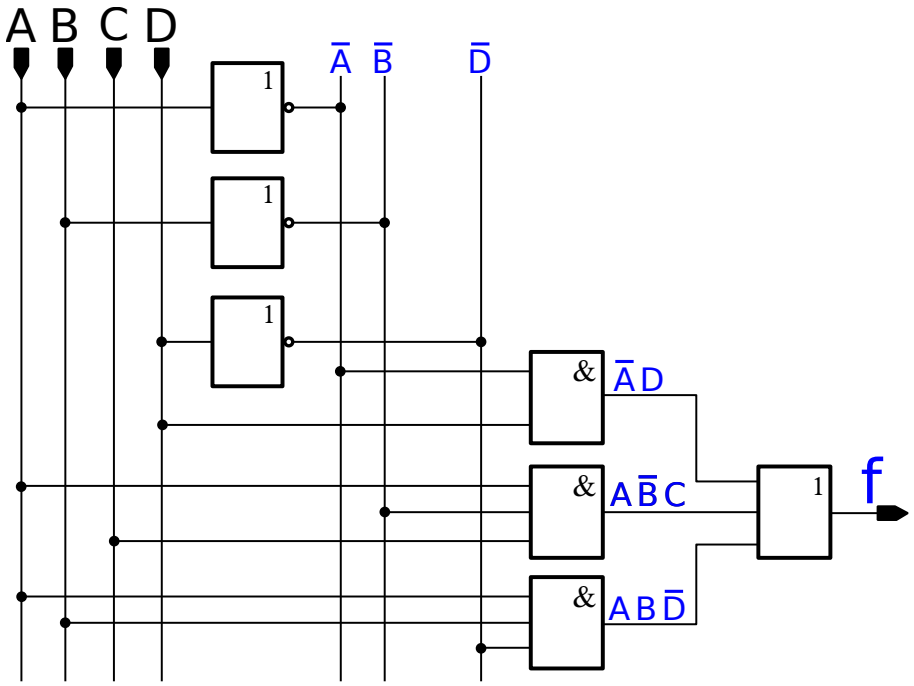
2. Pre negované vstupné premenné nakreslíme invertor a príslušnú vertikálnu čiaru s negovaným vstupným signálom.

3. Kreslíme a ortogonálne prepájame logické hradlá podľa vzorovej Booleovskej funkcie.

4. Napokon vizuálne skontrolujeme uzly, názvy vstupov/výstupov a pod.

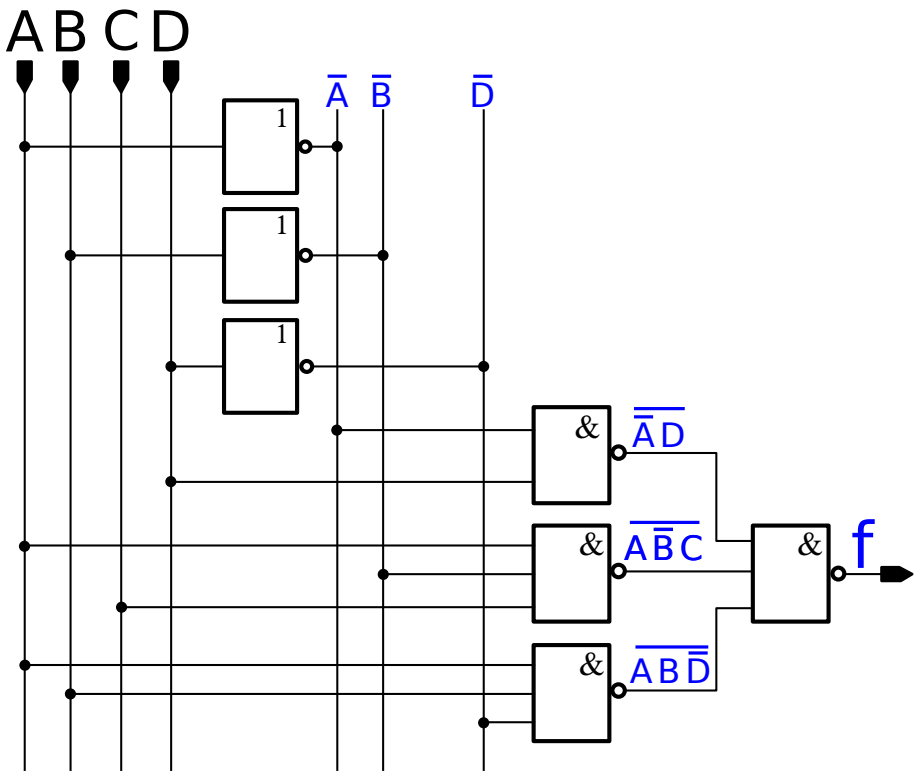
7.1 Vzorový príklad

Vyberme si Booleovskú funkciu $f = \overline{A}.D + A.\overline{B}.C + A.B.\overline{D}$. Ide o MDNF jednej z Karnuaghových máp pre štyri premenné z predchádzajúcej kapitoly. Vidíme, že funkciu tvoria tri mintermy, ktoré budeme realizovať hradlami AND s adekvátnym počtom vstupov. Na záver čiastkové výstupy z AND-ov sčítame v trojvstupovom hradle typu OR. Výslednú schému vidíme na Obr. 7.2.



Obr. 7.2: Hradlová schéma zvolenej funkcie (36 MOS tranzistorov)

Vidíme, že na realizáciu takéhoto obvodu budeme potrebovať celkovo sedem hradiel. Ak nakreslíme optimalizovanú verziu vybranej Booleovskej funkcie, taktiež budeme potrebovať sedem hradiel, avšak celkový počet tranzistorov bude o 22 % nižší. Ako vypočítať potrebný počet tranzistorov sa naučíme v nasledujúcej kapitole. Výslednú schému optimalizovanej funkcie vidíme na Obr. 7.3.



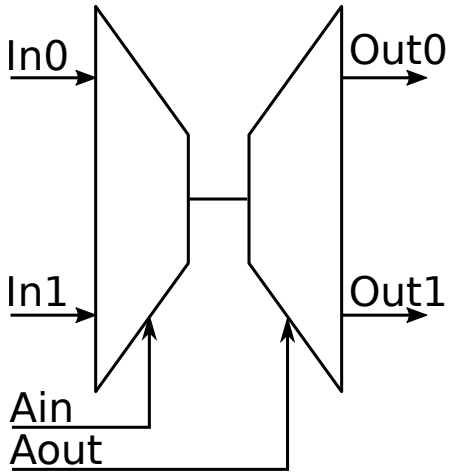
Obr. 7.3: Hradlová schéma optimalizovanej funkcie (28 MOS tranzistorov)

Úloha: Navrhnite 1-bitový selektor 2-na-2 na hradlovej úrovni. Neaktívny výstupný kanál bude v logickej jednotke.

Riešenie: Postup riešenia príkladu je v podstate zhrnutie doterajších vedomostí a postupov z predchádzajúcich kapitol. Ako prvé si nakreslíme blokovú schému (Obr. 7.4), aby sme korektne definovali a označili vstupy a výstupy navrhovaného obvodu. Vstupy do obvodu musia korešpondovať so vstupnou časťou PT. Následne na základe slovného zadania a znalosti funkčnosti obvodu selektora z prednášok vytvoríme PT (Tab. 7.1).

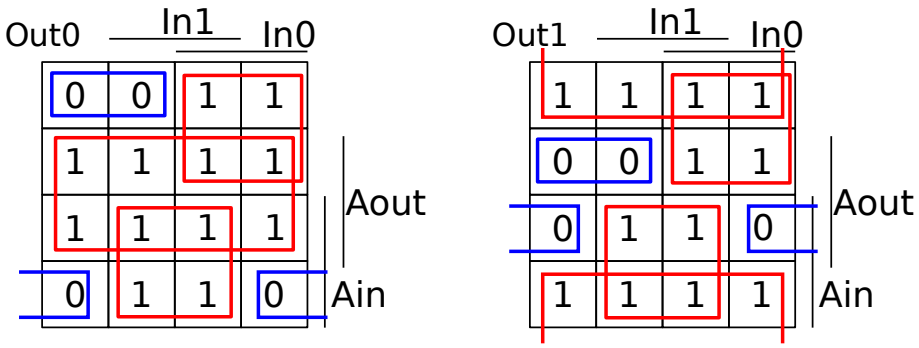
Tab. 7.1: Pravdivostná tabuľka navrhovaného selektora

Ain	Aout	In0	In1	Out0	Out1
0	0	0	0	0	1
0	0	0	1	0	1
0	0	1	0	1	1
0	0	1	1	1	1
0	1	0	0	1	0
0	1	0	1	1	0
0	1	1	0	1	1
0	1	1	1	1	1
1	0	0	0	0	1
1	0	0	1	1	1
1	0	1	0	0	1
1	0	1	1	1	1
1	1	0	0	1	0
1	1	0	1	1	1
1	1	1	0	1	0
1	1	1	1	1	1



Obr. 7.4: Bloková schéma navrhovaného selektora

Nasledujúcim krokom bude vytvorenie K-mapy pre obe výstupné funkcie, vytvorenie slučiek a nájdenie MDNF alebo MKNF. Obe K-mapy sú zobrazené nižšie na Obr. 7.5.

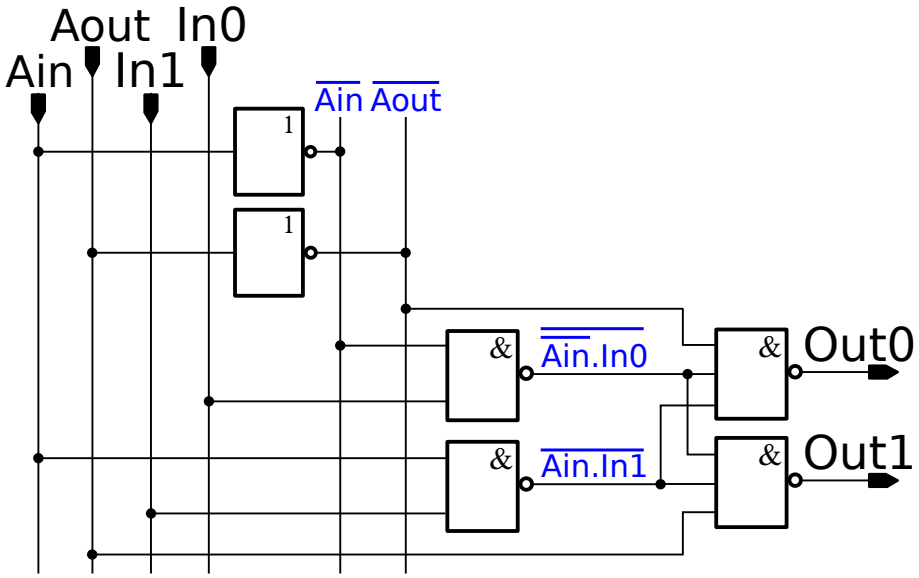


Obr. 7.5: Karnaughove mapy pre výstupné funkcie navrhovaného selektora

Výstupné MDNF a MKNF pre navrhovaný selektor sú uvedené v nasledujúcich rovniciach. Obe výstupné funkcie boli tiež optimalizované pre hardvérovú realizáciu pomocou hradla NAND, resp. NOR.

$$
\begin{aligned}
MDNF : Out0 &= Aout + \overline{Ain}.In0 + Ain.In1 \\
&= \overline{\overline{Aout}.(\overline{\overline{Ain}.In0}).(\overline{Ain.In1})} \\
Out1 &= \overline{Aout} + \overline{Ain}.In0 + Ain.In1 \\
&= \overline{\overline{Aout}.(\overline{\overline{Ain}.In0}).(\overline{Ain.In1})} \\
MKNF : Out0 &= (\overline{Ain} + Aout + In1).(Ain + Aout + In0) \\
&= \overline{\overline{(\overline{Ain} + Aout + In1)} + \overline{(Ain + Aout + In0)}} \\
Out1 &= (\overline{Ain} + \overline{Aout} + In1).(Ain + \overline{Aout} + In0) \\
&= \overline{\overline{(\overline{Ain} + \overline{Aout} + In1)} + \overline{(Ain + \overline{Aout} + In0)}}
\end{aligned}
$$

Ako si môžeme všimnúť, obe výstupné MDNF obsahujú dve identické slučky, a teda aj mintermy. Bude preto jednoznačne výhodnejšie použiť implementáciu MDNF pre finálnu schému a zdieľať čiastkové výpočty pre oba dátové výstupy.



Obr. 7.6: Hradlová schéma navrhovaného selektora 2-na-2

7.2 Príklady

Pre vybranú funkciu z predchádzajúcej kapitoly nakreslite schému na hradlovej úrovni. Majme však na pamäti, že každá schéma je originál. Najrozumnejšie bude, keď porovnáte vami vytvorené schémy s kolegami a prediskutujte správnosť riešenia, prehľadnosť a čitateľnosť. Nižšie sú uvedené slovné úlohy z praxe. Riešením týchto príkladov sú schémy na hradlovej úrovni. Keďže každá schéma je originál, porovnajte a prediskutujte vaše riešenie s kolegami alebo cvičiacimi.

Úloha 1: Navrhnite optimalizovaný odvod na hradlovej úrovni, realizujúci funkciu $f = x^2 + 1$, pričom číslo x bude 4-bitové číslo.

Úloha 2: Navrhnite optimalizovaný odvod na hradlovej úrovni, realizujúci funkciu $f = x^2 + x + 1$, pričom číslo x bude 4-bitové číslo.

Úloha 3: Navrhnite optimalizovaný 1-bitový selektor 2-na-2 na hradlovej úrovni. Neaktívny kanál bude v logickej nule.

Úloha 4: Navrhnite optimalizovaný komparátor dvoch dvojbitových čísiel. Výstup bude v logickej jednotke ak A $>$ B.

Úloha 5: Navrhnite optimalizovaný komparátor dvoch dvojbitových čísiel. Výstup bude v logickej jednotke ak A $<$ B.

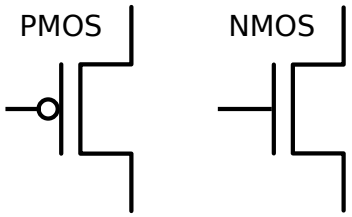
Úloha 6: Navrhnite optimalizovanú paralelnú násobičku dvoch dvojbitových čísiel.

Návrh kombinačných obvodov na tranzistorovej úrovni

V súčasnosti je významná väčšina integrovaných digitálnych (logických) systémov na čipe realizovaných pomocou CMOS technológie. Ide o čipy používané v rôznych prenosných a iných zariadeniach – smartfón, TV, počítač, tablet, smart hodinky, navigačné zariadenia a pod. V roku 2023 bol na trhu dostupný výrobca ponúkajúci 3 nm CMOS technológiu pre hromadnú výrobu integrovaných obvodov. Koncom roka 2024 je očakávané spustenie tzv. „risk production" pre 2 nm (20 Å) technológiu. Len pre ilustráciu, priemer ľudského vlasu má približne 100 μm. Teda rozmery polovodičových štruktúr na čipe vyrobenom takouto technológiou majú rozmery ako keby sme ľudský vlas rozrezali po dĺžke cca 5000-krát.

Už sme spomínali, že logické hradlá sa skladajú zo základných elektronických súčiastok na čipe – tranzistorov (typu MOS), ktoré sú „vhodne" zapojené. V tejto kapitole sa naučíme, ako zapojiť MOS tranzistory tak, aby vykonávali zadanú logickú operáciu či funkciu. Schéma zobrazujúca obvod sa v takomto prípade nazýva *tranzistorová schéma* alebo schéma hradla/obvodu na *tranzistorovej úrovni*. Ide o najefektívnejší spôsob návrhu logických operácií, ale za cenu pracného ručného návrhu s prirodzenými obmedzeniami. Ide nakoniec o akýsi kompromis medzi efektivitou návrhu a časom potrebným na návrh. Nie je mysliteľné, aby sa komplexné digitálne obvody (napr. CPU jadro 64-bitového moderného procesora) navrhovali ručne na tranzistorovej úrovni, ba ani na hradlovej úrovni. Je zrejmé, že takéto obvody sa navrhujú na vyššej úrovni abstrakcie – napr. bloková, RTL, sub-systémová či systémová úroveň, kde za pohodlnosť a spracovateľnosť veľkého dizajnu nakoniec zaplatíme jeho zníženou výslednou efektivitou.

Postup riešenia je veľmi podobný ako pri hľadaní MDNF a MKNF Booleovskej funkcie. Zmeny sú v podstate len pri samotnom kreslení schémy a pri zápise výrazov pre *pull-down* časť (časť, ktorá pripája výstup obvodu k nízkej hodnote napätia – GND). Pri písaní funkcie pre *pull-up* časť (časť, ktorá pripája výstup obvodu k vysokej hodnote napätia – V_{DD}) postupujeme ako pri hľadaní MDNF. Slučky s jednotkovými bodmi funkcie prislúchajú tranzistorom typu PMOS a pull-up časti. Analogicky, slučky s nulovými bodmi funkcie prislúchajú tranzistorom typu NMOS a pull-down časti. Pri zápise pull-down funkcie však nulové slučky opisujeme, ako keby išlo o jednotkové slučky. Ide teda o priamy protiklad pri porovnaní s hľadaním klasickej MKNF. Logický súčin v Booleovskej funkcii pre pull-up a pull-down časti je reprezentovaný tranzistormi zapojenými v sérii, logický súčet v Booleovskej funkcii je prekreslený ako tranzistory zapojené paralelne. Schematické značky MOS tranzistorov, z ktorých sa skladá výsledné logické hradlo vidíme na Obr. 8.1.



Obr. 8.1: Schematická značka MOS tranzistorov v CMOS technológii

Pred kreslením samotnej tranzistorovej schémy si upravíme výrazy do kompaktnejšieho tvaru a spravíme skúšku správnosti. Všimnime si, že pull-up aj pull-down časť vyžadujú rovnaký počet premenných, a tým aj tranzistorov. Všetky vstupné premenné pre pull-up časť sú negované a všetky premenné pre pull-down časť sú bez negácie. V praxi toto pravidlo nemusí platiť, ale v rámci predmetu „Logické systémy" sa obmedzíme na jednotný tvar premenných. Logický súčin a súčet sú použité komplementárne („do kríža"). Pri kreslení pull-up časti nekreslíme invertory pre negované vstupné premenné, pretože využijeme komplementárne správanie PMOS a NOMS tranzistorov. Postup návrhu je nasledovný:

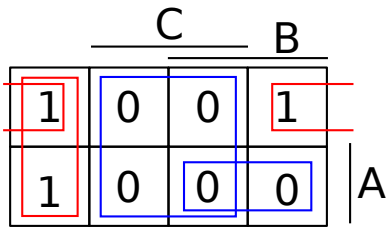
1. Vytvoríme pravdivostnú tabuľku zadanej funkcie.
2. Vytvoríme Karnaughovu mapu a v nej vytvárame slučky rovnako ako pri minimalizácii Booleovských funkcií.
3. Napíšeme výsledné funkcie pre pull-up a pull-down časť. Pozor pri pull-down časti.
4. Skúška správnosti získaných výrazov (negácie, operácie, počet písmen atď.).
5. Nakreslíme tranzistorovú schému pre získané výrazy z predchádzajúceho kroku. Negácie pre PMOS časť nekreslíme.
6. Skontrolujeme výslednú schému – napájanie, zem, vstupy, výstupy, uzly.

8.1 Vzorový príklad

Úloha: Navrhnite hradlo na tranzistorovej úrovni v CMOS technológii vykonávajúce funkciu $f = \overline{A.B + C}$.
Riešenie: Vytvoríme PT (Tab. 8.1), nakreslíme K-mapu a vytvoríme slučky (Obr. 8.2). Slučky jednotkových bodov predstavujú zápis pull-up časti hradla, slučky nulových bodov reprezentujú pull-down časť.

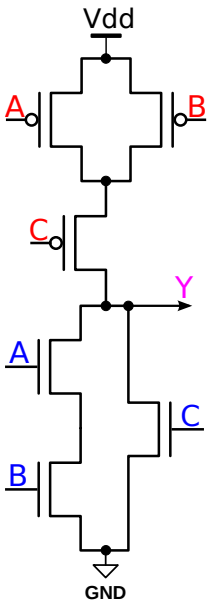
Tab. 8.1: Pravdivostná tabuľka navrhovaného hradla

A	B	C	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0



Obr. 8.2: Karnaughova mapa navrhovaného hradla

Pull-Up: $PU = MDNF = \overline{A}.\overline{C} + \overline{B}.\overline{C} = \overline{C}.(\overline{A} + \overline{B})$
Pull-Down: $PD = C + A.B$



Obr. 8.3: Jedna z možných schém zapojenia hradla funkcie $f = \overline{A.B + C}$

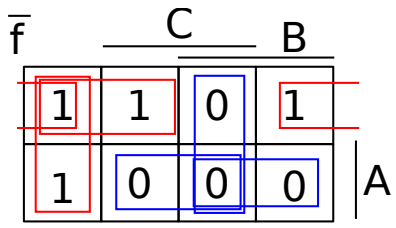
Opäť platí, že možností pre nákres výslednej tranzistorovej schémy (Obr. 8.3) môže byť niekoľko. V pull-down časti by sme mohli prehodiť tranzistory A a B (lebo platí komutatívny zákon) alebo v pull-up časti by sme mohli prehodiť tranzistor C a paralelný spoj tranzistorov $(A + B)$ medzi V_{DD} a výstupom. Napriek viacerým možnostiam vnútorného zapojenia dostávame stále navonok rovnaké správanie obvodu. Výstupný uzol je v závislosti od okamžitých hodnôt vstupných premenných „potiahnutý" buď k zemi GND (výstup bude v logickej nule), alebo k napájaniu VDD (výstup bude v logickej jednotke). Ak by sme chceli pôvodnú funkciu realizovať na hradlovej úrovni, potrebovali by sme jedno dvojvstupové hradlo AND pre realizáciu $A.B$ (6 tranzistorov) a následne ďalšie dvojvstupové hradlo NOR (4 tranzistory) pre premennú C a čiastkový výpočet hradla AND. Spolu ide teda o 10 tranzistorov: $6\mathrm{T}_{AND} + 4\mathrm{T}_{NOR}$. Keď porovnáme 6 tranzistorov potrebných pri realizácii obvodu priamo na tranzistorovej úrovni versus 10 tranzistorov na hradlovej úrovni, dostávame až 40 %-nú úsporu počtu tranzistorov!

Len pre predstavu, pri moderných dizajnoch digitálnych systémov na čipe s desiatkami miliárd tranzistorov na jedinom čipe má zmysel šetriť jednotky až desatiny percent úspory, nakoľko v absolútnych číslach ide o relevantný počet tranzistorov, ktoré môžu významne zvýšiť plochu (a cenu) čipu či spotrebu energie.

Úloha: Navrhnite na tranzistorovej úrovni v CMOS technológii hradlo vykonávajúce funkciu majority troch bitov $f = A.B + A.C + B.C$.
Riešenie: Kedže v CMOS technológii sa priamo vyjadrujú negované Booleovské funkcie, začneme realizovať funkciu $\overline{f}$. PT je zapísaná v Tab. 8.1 a prislúchajúca K-mapa je na Obr. 8.4.

Tab. 8.2: Pravdivostná tabuľka navrhovaného hradla

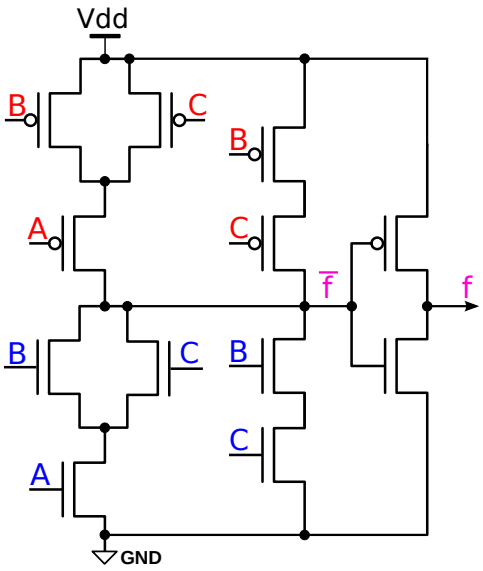
A	B	C	$\overline{f}$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0



Obr. 8.4: Karnaughova mapa navrhovaného hradla

Pull-Up: $PU = \overline{A}.\overline{B} + \overline{A}.\overline{C} + \overline{B}.\overline{C} = \overline{A}.(\overline{B} + \overline{C}) + \overline{B}.\overline{C}$
Pull-Down: $PD = A.B + B.C + A.C = A.(B + C) + B.C$

Funkcie pre pull-up a pull-down časti hradla umožňujú iba vyňatie jednej vybranej premennej pred zátvorku, čím ušetríme spolu dva tranzistory. Tranzistorová schéma je zobrazená na Obr. 8.5.



Obr. 8.5: Jedna z možných schém zapojenia hradla majorita 2-z-3

Na dokončenie schémy zapojíme ešte dva tranzistory ako invertor, aby sme dosiahli na výstupe hradla požadovaný tvar funkcie f. Keby sme danú funkciu majority troch bitov navrhovali na hradlovej úrovni, potrebovali by sme tri dvojvstupové hradlá AND (3 x 6 tranzistorov) a jedno trojvstupové hradlo OR (8 tranzistorov). Spolu to teda je 26 MOS tranzistorov, kým návrh na tranzistorovej úrovni vyžaduje iba 12 MOS tranzistorov. Ak porovnáme tieto dve čísla, dostávame takmer 54 %-nú úsporu počtu potrebných súčiastok.

8.2 Príklady

Úloha 1: Navrhnite na tranzistorovej úrovni v CMOS technológii elementárne hradlo operácie NAND a NOR pre dve, tri a štyri premenné.

Úloha 2: Odvoďte z nich hradlá AND a OR.

Úloha 3: Nakreslite tranzistorovú schému pre uvedené Booleovské funkcie a vypočítajte percentuálnu úsporu počtu tranzistorov oproti realizácii na hradlovej úrovni. Ide o príklady z bežných cvičení. Nakoľko sa pri návrhu na tranzistorovej úrovni obmedzujeme jednotným tvarom vstupných premenných, počet príkladov je tiež obmedzený. Skúste však navhnúť hradlo na tranzistorovej úrovni pre ľubovoľnú funkciu a prípadné potrebné negácie vstupných premenných môžeme vytvoriť klasickým invertorom zloženým z dvoch extra tranzistorov. Svoje riešenie porovnajte s kolegami, prípadne prediskutujte s cvičiacimi.

$$f_1 = \overline{A.B + C}$$
$$f_2 = \overline{(A + B).C}$$
$$f_3 = \overline{(\overline{A} + \overline{B}).\overline{C}}$$
$$f_4 = \overline{\overline{A}.\overline{B} + \overline{C}}$$

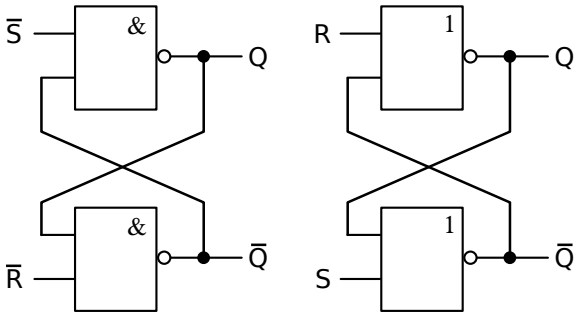
$$f_5 = \overline{(A + B).C.D}$$
$$f_6 = \overline{(A + B).(C + D)}$$
$$f_7 = \overline{(A + B + C).D}$$
$$f_8 = \overline{A.B + C + D}$$
$$f_9 = \overline{A.B + C.D}$$
$$f_{10} = \overline{A.B.C + D}$$
$$f_{11} = \overline{B}.(\overline{A} + \overline{C} + \overline{D})$$
$$f_{12} = \overline{A.B.(C + D)}$$

Pamäťové elementy

Preklápací obvod (anglicky „flip-flop") je špeciálny typ digitálneho (logického) obvodu. Slúži ako pamäťový prvok používaný v sekvenčných logických obvodoch, ktorého činnosť je riadená pomocou *hrany* hodinového signálu. S taktom hodinového signálu sa teda do tohto pamäťového obvodu periodicky zapisuje konkrétna vstupná logická hodnota, ktorá je taktiež kopírovaná na jeho výstup. Preklápacích obvodov existuje niekoľko druhov, podľa spôsobu ich operácie. My sa zoznámime s preklápacími obvodmi typu JK, T a D. Najznámejší a najpoužívanejší z nich je preklápací obvod typu D, veľmi často nazývaný aj ako *register*.

Najprv si však predstavme najjednoduchší a základný obvod, ktorý obsahuje pamäťovú funkciu. Ide o tzv. SR alebo RS obvod. V anglickej literatúre sa veľmi často označuje aj ako SR/RS latch. Obvody typu *latch* (slovenský preklad je „záchytný register") tiež disponujú pamäťovou funkciou, ale namiesto na hranu reagujú na úroveň synchronizačného hodinového signálu. V praxi sa používajú len veľmi výnimočne, prípadne v špeciálnych prípadoch. Keďže RS obvod nie je synchronizovaný hranou hodinového signálu, nemôžeme ho nazývať preklápací obvod. RS latch obvod (na trhu predávaný ako integrovaný obvod 74LS279) tvoria dve hradlá typu NAND alebo typu NOR zapojené do vzájomnej spätnej väzby (Obr. 9.1), vďaka ktorej získavame najjednoduchšiu funkciu pamäte. Detajlnejší opis návrhu, vlastností a činnosti RS obvodu je súčasťou prednášok. Na jednoduchom RS latch obvode sú vybudované ďalšie zložitejšie pamäťové latch a preklápacie obvody typu JK, T, či D. K základnému RS obvodu sú v takom prípdade pridávané ďalšie logické hradlá na rozšírenie nových funkcií a parametrov obvodu. Aj keď každý typ preklápacieho obvodu obsahuje pamäťovú funkciu, odlišujú sa však funkciou zápisu do pamäte.

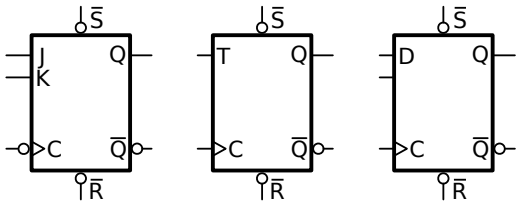
Od jednoduchého RS obvodu, ktorý vie pamäť nastaviť do logickej jednotky (funkcia set), ďalej nastaviť ju do logickej nuly (funkcia reset) alebo si zapamätať predchádzajúcu hodnotu, až po D preklápací obvod, ktorý funguje ako skutočný register pre vstupné dáta, avšak je obvodovo najzložitejší. V praxi sa najčastejšie stretneme práve s preklápacími obvodmi typu D a JK. Obľúbenosť D preklápacieho obvodu je daná jednoduchosťou návrhu N-bitového registra. Obvod JK je obľúbený z viacerých dôvodov. Ide o druhý najjednoduchší pamäťový obvod s množstvom funkcií. Ďalšou výhodou je, že aj napriek dvom riadiacim vstupným signálom býva pridružená kombinačná logika pre JK obvod zväčša jednoduchšia ako pre iné pamäťové obvody vďaka jeho tabuľke prechodov.



Obr. 9.1: Zapojenie SR/RS obvodu pomocou hradiel NAND a NOR

Funkcie *reset* a *set* (niekedy aj *clear* a *preset*) ostávajú využívané v zložitejších preklápacích obvodoch pre prepísanie pamäte logickou nulou lebo logickou jednotkou. Tieto funkcie môžu byť aktivované synchrónne (podriadené hodinovému signálu) alebo asynchrónne (nepodriadené hodinovému signálu). Opis vlastností a možností ohľadom problematiky funkcií set a reset však presahuje rozsah a účel tohto učebného textu. Každopádne reset a set sú vstupné riadiace signály, ktoré synchrónne alebo asynchrónne umožňujú prepísať pamäť preddefinovanou logickou hodnotou (pre reset je to logická nula, pre set je to logická jednotka).
Obr. 9.2 zobrazuje schematické značky najpoužívanejších preklápacích obvodov. Ide o už spomínaný preklápací obvod JK, synchronizovaný dobežnou hranou hodín spolu s asynchrónnym setom a resetom, na trhu predávaný ako obvod 74HC112. Preklápací obvod typu T dostaneme, ak JK obvodu skratujeme vstupy J a K.

Stretnúť sa s ním však môžeme aj pri návrhu integrovaných obvodov či pri návrhu programovateľných digitálnych obvodov napr. FPGA. Obvod s označením 74HC74 (alebo tiež 74HC273 či 74HC574) predstavuje klasický D preklápací obvod (jednobitový register) synchronizovaný nábežnou hranou hodinového signálu spolu s asynchrónnym setom a resetom. Skúste zistiť z dokumentácie obvodov, ako budú reagovať dané obvody, ak by sme aktivovali funkcie set a reset naraz. Prediskutujte to s kolegami.



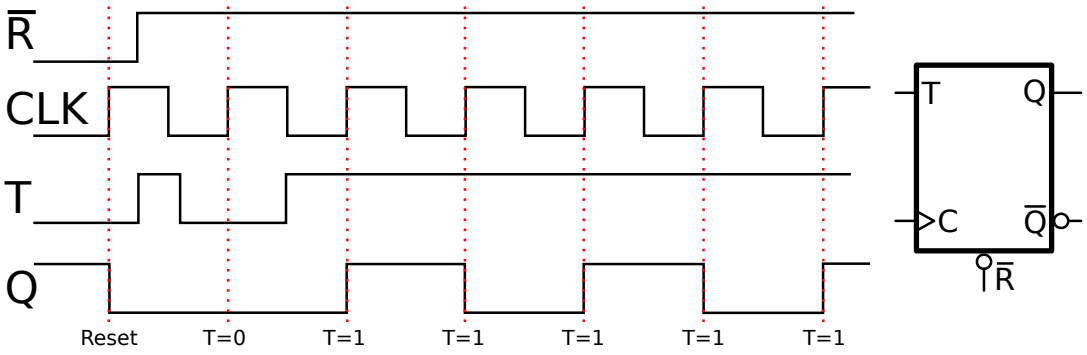
Obr. 9.2: Schematické značky najpoužívanejších preklápacích obvodov

Problematika pamäťových obvodov je príliš zložitá a rozsiahla pre túto publikáciu. Viac informácií sa môžete dozvedieť na prednáškach, prípadne nájdete informácie v literatúre a online priestore.

9.1 Vzorový príklad

Úloha: Nakreslite schematickú značku a priebeh signálov v čase pre T preklápací obvod synchronizovaný nábežnou hranou hodín, spolu so synchrónnym resetom aktivovaným logickou nulou. Demonštrujte všetky funkcie obvodu.

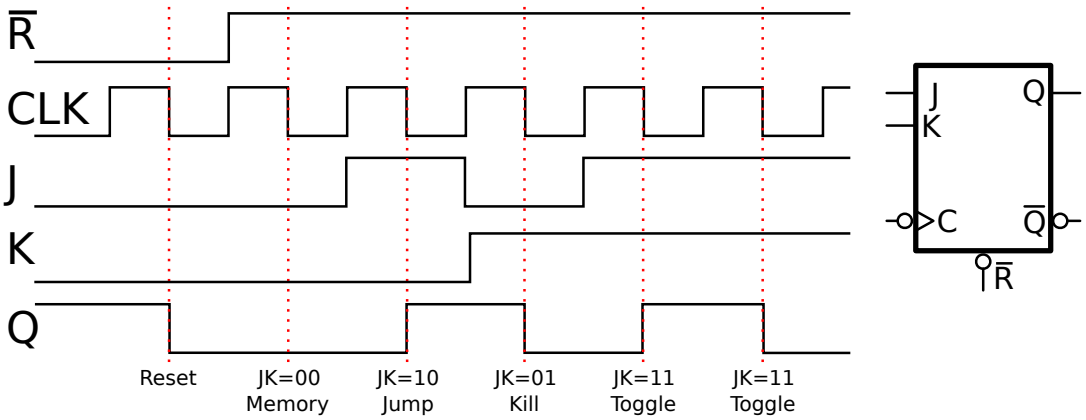
Riešenie:



Obr. 9.3: Schematická značka a časové priebehy signálov T preklápacieho obvodu

Pri prvej nábežnej hrane hodinového signálu sa preklápací obvod vyresetuje, aby sme v pamäti mali známu hodnotu bez ohľadu na to, v akom stave bola pamäť predtým (napr. pri zapnutí napájania sa do pamäte zapíše náhodná a pre nás neznáma hodnota). Z tohto dôvodu pamäť resetujeme – nastavíme výstup pamäťového obvodu do nuly. Pri druhej nábežnej hrane hodín je reset neaktívny (je v logickej jednotke). Riadiaci vstup T = 0, preto sa predchádzajúci stav neneguje a ostáva v nule. Na tretí a každý ďalší hodinový cyklus je T = 1. Predchádzajúci stav, resp. hodnota na výstupe obvodu bude preto pri každej nábežnej hrane hodinového signálu negovaná.

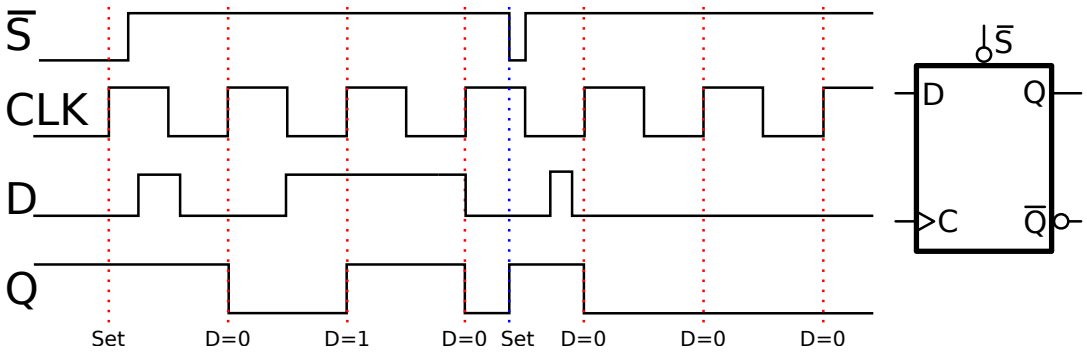
Úloha: Nakreslite schematickú značku a priebeh signálov v čase pre JK preklápací obvod synchronizovaný dobežnou hranou hodín, spolu so synchrónnou funkciou reset aktivovanou logickou nulou. Demonštrujte všetky funkcie obvodu.
Riešenie:



Obr. 9.4: Schematická značka a časové priebehy signálov JK preklápacieho obvodu

Pri prvej dobežnej hrane hodinového signálu dochádza ku synchrónnemu resetu. Pri druhej dobežnej hrane sa bude pamätať predchádzajúca hodnota v pamäti obvodu – nula z resetu. Pri tretej dobežnej hrane hodinového signálu je aktivovaná funkcia „Jump", teda výstup sa nastaví do stavu logickej jednotky. Pri štvrtej hrane hodinového signálu je aktivovaná funkcia „Kill", preto bude výstup obvodu vynulovaný. Pri piatej a šiestej hrane hodín je aktivovaná funkcia preklopenia pôvodneho stavu, teda pôvodná nula sa stane jednotkou.

Úloha: Nakreslite schematickú značku a priebeh signálov v čase pre D preklápací obvod synchronizovaný nábežnou hranou hodín, spolu s asynchrónnou funkciou set aktivovanou logickou nulou. Demonštrujte všetky funkcie obvodu.
Riešenie:



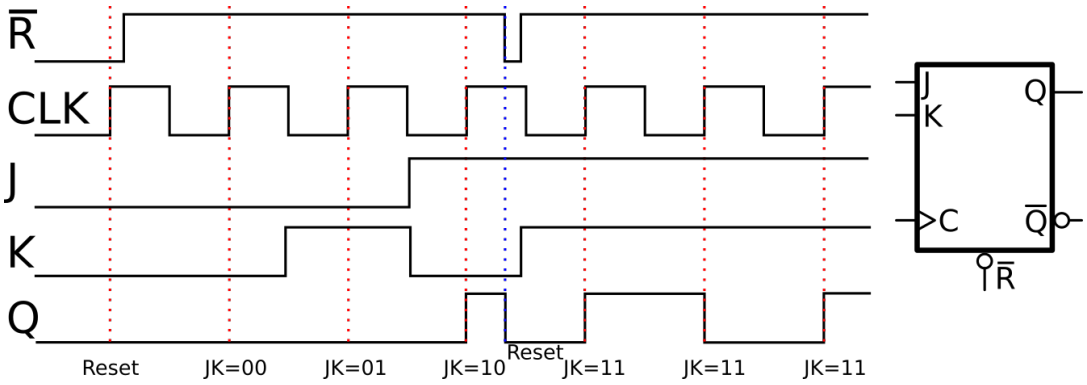
Obr. 9.5: Schematická značka a časové priebehy signálov D preklápacieho obvodu

Kedže tento obvod obsahuje asynchrónny set, musíme demonštrovať jeho aktivovanie mimo synchronizácie hodinovým signálom. Na úplnom začiatku (t = 0) je obvod v asynchrónnom sete, teda výstup pamäte bude nastavený do stavu logickej jednotky. Pri druhej nábežnej hrane hodinového signálu zapisujeme na výstup obvodu logickú nulu. Pri treťom cykle hodinového signálu zapíšeme logickú jednotku. Nasleduje zápis nuly a pred piatym hodinovým cyklom sa aktivuje asynchrónny set, ktorý (asynchrónne) nastaví výstup obvodu do logickej jednotky. Piaty, šiesty a siedmy cyklus zapíšu do pamäťe logickú nulu.

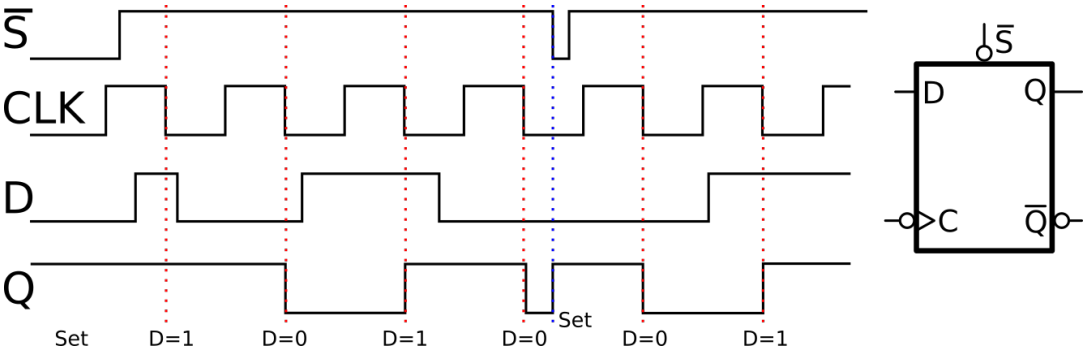
9.2 Príklady

Všetky uvedené časové priebehy sú len jednou z mnohých správnych možností. Opäť platí, že tieto úlohy majú mnoho rôznych správnych riešení. Je môžné zvoliť ľubovoľné poradie demonštrácie funkcií preklápacieho obvodu.

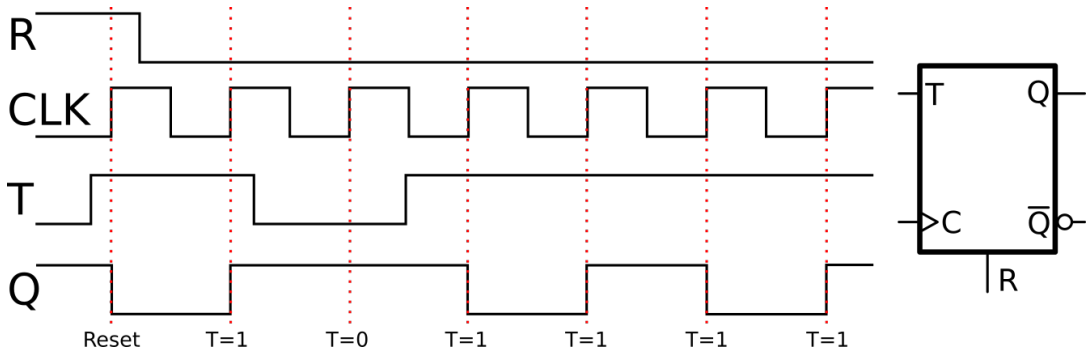
Úloha 1: Nakreslite schematickú značku a priebeh signálov v čase pre JK preklápací obvod synchronizovaný nábežnou hranou hodín, spolu s asynchrónnou funkciou reset aktivovanou logickou nulou. Demonštrujte všetky funkcie obvodu.



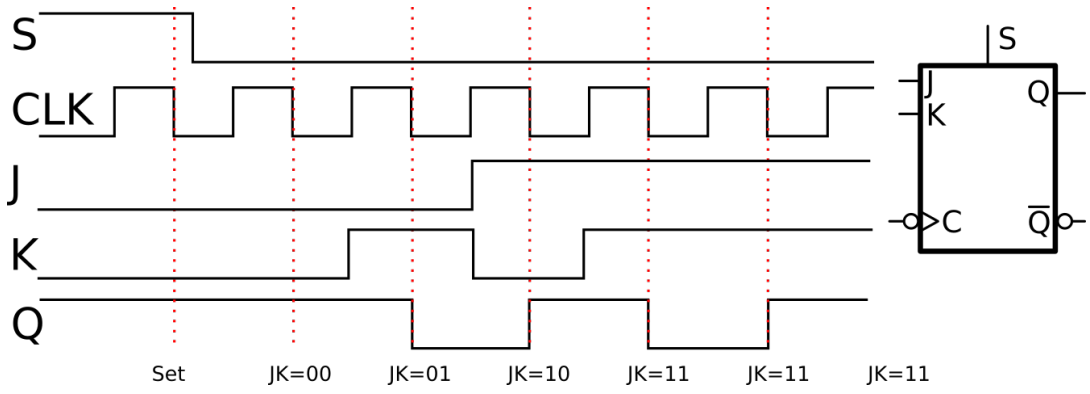
Úloha 2: Nakreslite schematickú značku a priebeh signálov v čase pre D preklápací obvod synchronizovaný dobežnou hranou hodín, spolu s asynchrónnou funkciou set aktivovanou logickou nulou. Demonštrujte všetky funkcie obvodu.



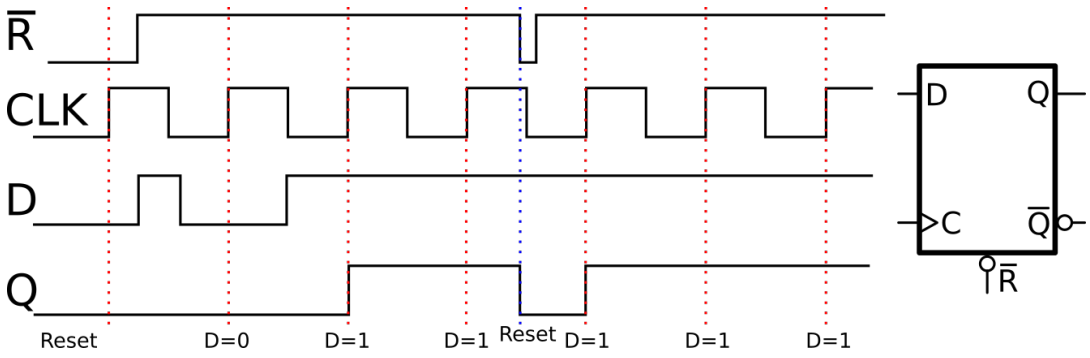
Úloha 3: Nakreslite schematickú značku a priebeh signálov v čase pre T preklápací obvod synchronizovaný nábežnou hranou hodín, spolu so synchrónnou funkciou reset aktivovanou logickou jednotkou. Demonštrujte všetky funkcie obvodu.



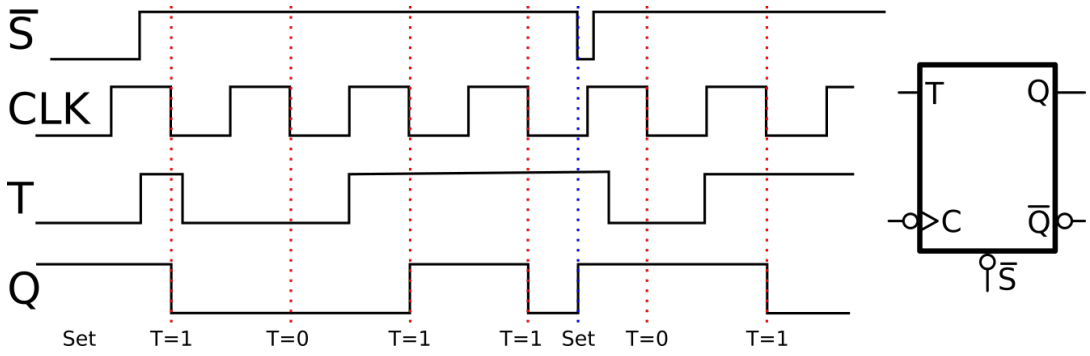
Úloha 4: Nakreslite schematickú značku a priebeh signálov v čase pre JK preklápací obvod synchronizovaný dobežnou hranou hodín, spolu so synchrónnou funkciou set aktivovanou logickou jednotkou. Demonštrujte všetky funkcie obvodu.



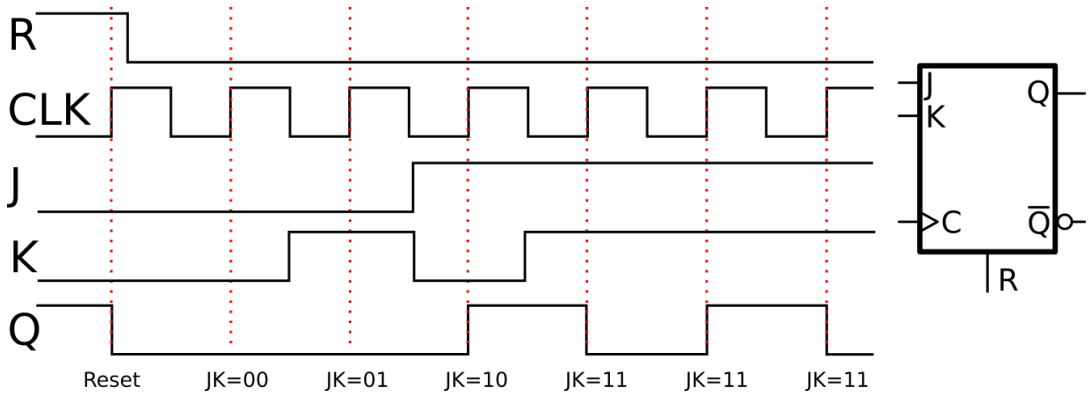
Úloha 5: Nakreslite schematickú značku a priebeh signálov v čase pre D preklápací obvod synchronizovaný nábežnou hranou hodín, spolu s asynchrónnou funkciou reset aktivovanou logickou nulou. Demonštrujte všetky funkcie obvodu.



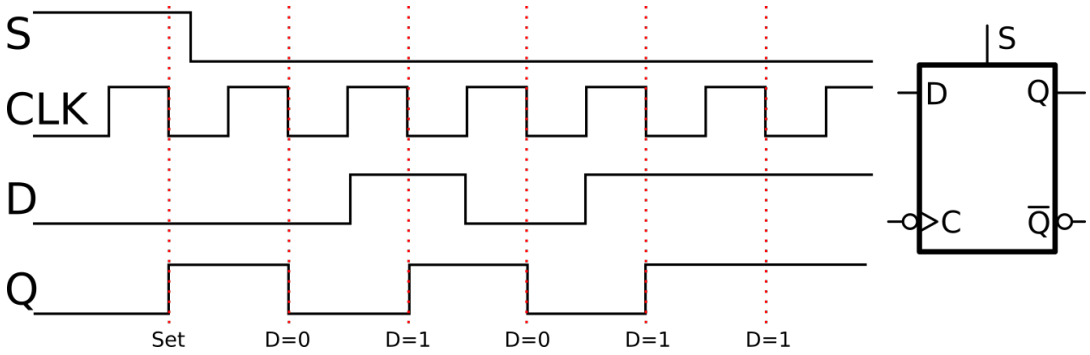
Úloha 6: Nakreslite schematickú značku a priebeh signálov v čase pre T preklápací obvod synchronizovaný dobežnou hranou hodín, spolu s asynchrónnou funkciou set aktivovanou logickou nulou. Demonštrujte všetky funkcie obvodu.



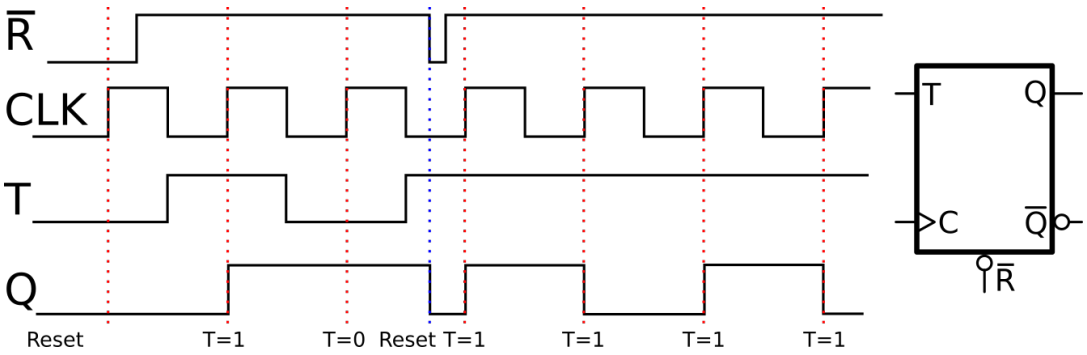
Úloha 7: Nakreslite schematickú značku a priebeh signálov v čase pre JK preklápací obvod synchronizovaný nábežnou hranou hodín, spolu so synchrónnou funkciou reset aktivovanou logickou jednotkou. Demonštrujte všetky funkcie obvodu.



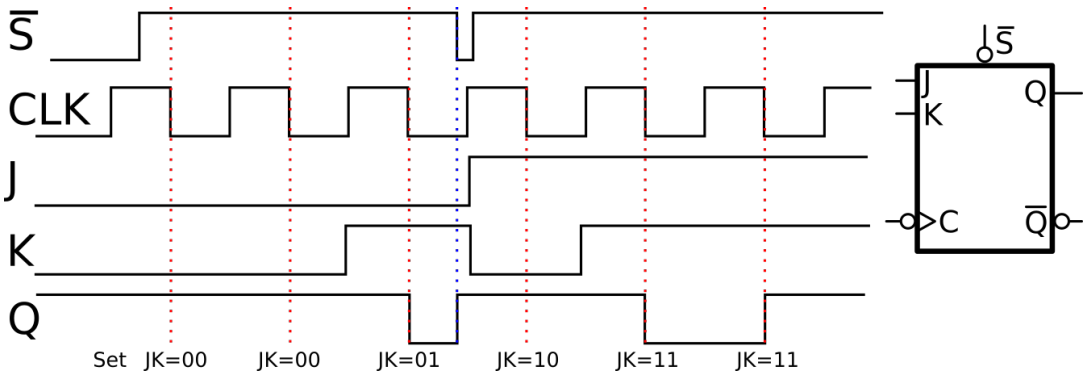
Úloha 8: Nakreslite schematickú značku a priebeh signálov v čase pre D preklápací obvod synchronizovaný dobežnou hranou hodín, spolu so synchrónnou funkciou set aktivovanou logickou jednotkou. Demonštrujte všetky funkcie obvodu.



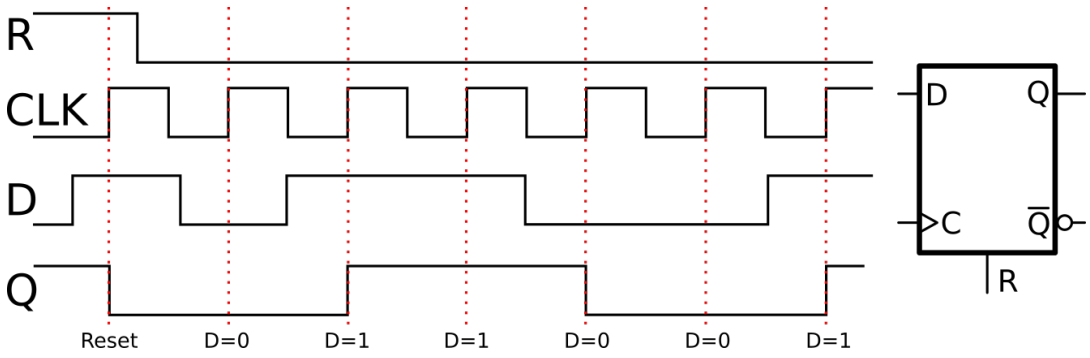
Úloha 9: Nakreslite schematickú značku a priebeh signálov v čase pre T preklápací obvod synchronizovaný nábežnou hranou hodín, spolu s asynchrónnou funkciou reset aktivovanou logickou nulou. Demonštrujte všetky funkcie obvodu.



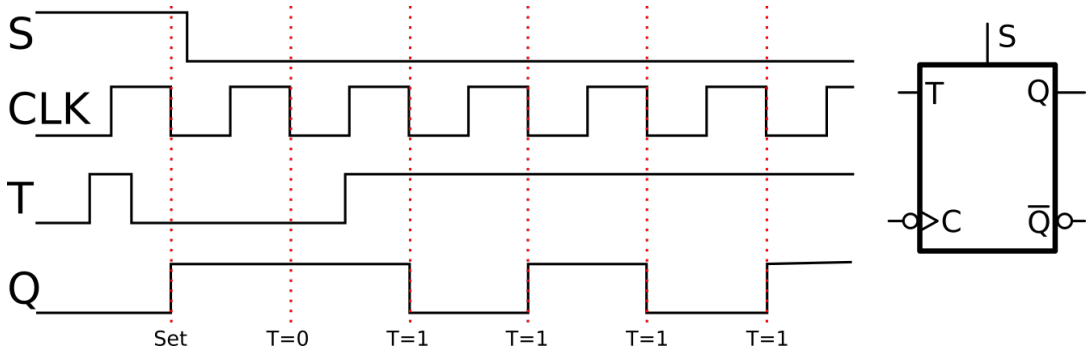
Úloha 10: Nakreslite schematickú značku a priebeh signálov v čase pre JK preklápací obvod synchronizovaný dobežnou hranou hodín, spolu s asynchrónnou funkciou set aktivovanou logickou nulou. Demonštrujte všetky funkcie obvodu.



Úloha 11: Nakreslite schematickú značku a priebeh signálov v čase pre D preklápací obvod synchronizovaný nábežnou hranou hodín, spolu so synchrónnou funkciou reset aktivovanou logickou jednotkou. Demonštrujte všetky funkcie obvodu.



Úloha 12: Nakreslite schematickú značku a priebeh signálov v čase pre T preklápací obvod synchronizovaný dobežnou hranou hodín, spolu so synchrónnou funkciou set aktivovanou logickou jednotkou. Demonštrujte všetky funkcie obvodu.



Ekvivalencia a redukcia stavov konečných stavových automatov

Tak, ako sme boli motivovaní pomocou matematiky šetriť hradlá alebo tranzistory pri návrhu kombinačných obvodov, rovnako sa snažíme pomocou matematiky minimalizovať rozsah potrebných prostriedkov pri návrhu sekvenčných logických obvodov, konkrétne v tejto kapitole pamäťového priestoru v rámci konečného stavového automatu. Každý konečný stavový automat (anglicky *Finite State Machine* – FSM) alebo len „automat" má vnútorné stavové premenné a výstupné premenné, ktoré sú najčastejšie uložené v registroch. Konečný stavový automat je častokrát implementovaný aj pri klasickom programovaní, pričom dáta sú vtedy ukladané väčšinou v operačnej pamäti typu RAM. Predstaveným postupom v tejto kapitole sa naučíme zredukovať počet stavov (a prechodov) pri zachovaní správnej a nezmenenej pôvodnej funkcie FSM. Uvedený algoritmus môžeme použiť pri redukcii stavových automatov implementovaných do hardvéru, ale aj pri softvérovej implementácii.

Pri návrhu FSM najčastejšie pracujeme so stavovým diagramom, kde si ručne zadefinujeme predstavu o počte stavov a podmienkach prechodov medzi nimi. Je všeobecne známe, že ručný návrh nie je najefektívnejší, nakoľko pri ňom môžu vzniknúť napr. redundantné stavy a s nimi aj redundantné prechody, ktoré sa počas redukcie snažíme zlúčiť. Redukcia stavov FSM často prináša zlúčenie stavov, ktoré na prvý pohľad nevyzerajú byť ekvivalentné. Ide o prirodzenú motiváciu ukladať do pamäte čo najmenší počet bitov stavových premenných.

Proces redukcie je postavený na práci s tabuľkou prechodov. Ako prvý krok si rozdelíme stavy podľa hodnoty výstupných dát. V podstate združíme do množín stavy, ktoré generujú rovnaký výstup.

Dostaneme tak ideálnu úroveň redukcie, pretože vytvoríme minimálny možný počet množín – jednu množinu stavov pre každú unikátnu výstupnú hodnotu. Skupina týchto množín stavov sa nazýva *prvá trieda ekvivalencie* – E_1. V neskorších krokoch redukovania sa budú tieto množiny rozkladať na viacero menších, pričom budeme zvyšovať úroveň triedy ekvivalencie popri kontrole prechodov v triede ekvivalencie o úroveň nižšie. Kontrolujeme, či sú cieľové stavy prechodov príslušných overovaných stavov v spoločnej množine. Toto pravidlo musí byť splnené pre každý hodnotu výstupu (logická nula aj logická jednotka). Ak kontrolovaná dvojica stavov nespĺňa podmienky ekvivalencie, nebude možné ich v danej triede ekvivalnecie udržať spolu v jednej množine. Dôjde preto k rozdeleniu jednej množiny na ďalšie menšie. Pri určitej triede ekvivalencie však už nedôjde k žiadnej zmene v zápise množín stavov. V takejto situácii môžeme postup redukcie ukončiť, pretože všetky nasledujúce triedy by vyzerali rovnako ako predchádzajúce. Takto vytvorené množiny ekvivalentných (pôvodných) stavov predstavujú nové stavy a vzniká nový, zredukovaný automat. Tento zapíšeme do tabuľky prechodov a nakreslíme výsledný zredukovaný stavový diagram.

Postup by sme mohli zhrnúť nasledovne:

1. Získame tabuľku prechodov automatu.
2. Rozdelíme stavy do množín podľa hodnôt výstupnej funkcie – vzniká trieda ekvivalencie E_1.
3. Kontrolujeme celistvosť vyššej triedy ekvivalencie (E_2) pomocou kontroly *ekvivalencie prechodov* v rámci nižšej množiny (E_1).
4. Zvyšujeme triedu ekvivalencie dovtedy, kým $E_{i+1} = E_i$.
5. Každá množina ekvivaletných pôvodných stavov predstavuje jeden nový stav v redukovanom automate.
6. Prepíšeme starú tabuľku prechodov do novej, zredukovanej tabuľky.
7. Nakreslíme diagram stavov zredukovaného automatu.

Každopádne, uvedený postup je veľmi algoritmický a mechanicky ľahko vykonateľný. Pre príklady v rámci predmetu „Logické systémy" sa obmedzíme iba na jeden, maximálne dva bity vstupných dát. Ide nám o ručný zápis na papier. V praxi by sme samozrejme využili softvér postavený na opísanom alebo podobnom algoritme. Takýto softvér dokáže redukovať automaty s výrazne väčšou šírkou slova a množstvom stavov a prechodov.

10.1 Vzorový príklad

Úloha: Zredukujte stavový automat definovaný v Tab. 10.1. Nakreslite diagram zredukovaného automatu a napíšte novú tabuľku prechodov automatu.
Riešenie: Môžeme vytvoriť prvú triedu ekvivalencie – E_1. Pri pohľade na tabuľku vidíme dva vzory výstupov. Prvá skupina stavov generuje výstup {0 0} a druhá skupina stavov generuje výstup {1 1}.

Tab. 10.1: Definícia prechodov a výstupných hodnôt stavového automatu

	IN = 0	IN = 1
A	B / 0	D / 0
B	D / 1	A / 1
C	F / 0	D / 0
D	E / 0	D / 0
E	F / 0	D / 0
F	D / 1	C / 1

Teraz môžeme zapísať triedu ekvivalencie E_1. Následne budeme kontrolovať vyššiu úroveň triedy ekvivalencie pre všetky možné kombinácie stavov v rámci jednej množiny. V tomto prípade máme dve množiny stavov, kde prvá obsahuje celkom šest unikátnych kombinácií a druhá obsahuje iba jednu kombináciu.

$$E_1 = \{\{A, C, D, E\}\{B, F\}\}$$

Vytvoríme vyššiu triedu ekvivalencie E_2 tak, že vyšetríme ekvivalenciu stavov pre všetky možné kombinácie v rámci množín stavov E_1:

$$A\ E_2\ C : (B\ E_1\ F) \wedge (D\ E_1\ D) \rightarrow A\ E_2\ C$$
$$A\ E_2\ D : (B\ \not{E_1}\ E) \wedge (D\ E_1\ D) \rightarrow A\ \not{E_2}\ D$$
$$A\ E_2\ E : (B\ E_1\ F) \wedge (D\ E_1\ D) \rightarrow A\ E_2\ E$$
$$C\ E_2\ D : (F\ \not{E_1}\ E) \wedge (D\ E_1\ D) \rightarrow C\ \not{E_2}\ D$$
$$C\ E_2\ E : (F\ E_1\ F) \wedge (D\ E_1\ D) \rightarrow C\ E_2\ E$$
$$D\ E_2\ E : (E\ \not{E_1}\ F) \wedge (D\ E_1\ D) \rightarrow D\ \not{E_2}\ E$$
$$B\ E_2\ F : (D\ E_1\ D) \wedge (A\ E_1\ C) \rightarrow B\ E_2\ F$$

Zo zápisu vidíme, že stav D nie je E_2 ekvivalentný so žiadnym členom množiny, a teda príslušná množina sa bude rozdeľovať. Trieda ekvivalencie E_2 je definovaná nasledovne:

$$E_2 = \{\{A, C, E\}\{D\}\{B, F\}\}$$

Opäť vyšetríme vyššiu triedu, teda E_3.

$$A\ E_3\ C : (B\ E_2\ F) \wedge (D\ E_2\ D) \rightarrow A\ E_3\ C$$
$$A\ E_3\ E : (B\ E_2\ F) \wedge (D\ E_2\ D) \rightarrow A\ E_3\ E$$
$$C\ E_3\ E : (F\ E_2\ F) \wedge (D\ E_2\ D) \rightarrow C\ E_3\ E$$
$$B\ E_3\ F : (D\ E_2\ D) \wedge (A\ E_2\ C) \rightarrow B\ E_3\ F$$

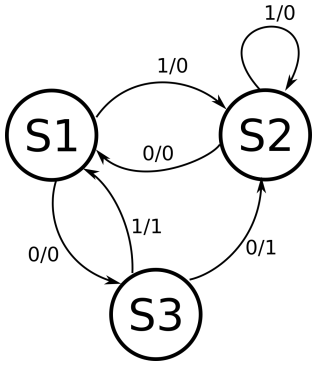
Pri kontrole tretej triedy ekvivalencie nedošlo k žiadnemu prípadu neekvivalencie a nedošlo teda k zmene zápisu množín. Je splnená podmienka ukončenia redukovania automatu, nakoľko $E_3 = E_2 = E$.

$$E_3 = \{\{A, C, E\}\{D\}\{B, F\}\} = E$$

Finálny zredukovaný automat bude vyžadovať iba tri stavy namiesto šestich pôvodných. Prvá množina stavov {A, C, E} bude v novom automate nazvaná ako stav S_1. Druhá množina stavov je v podstate iba pôvodný stav D, a teda S_2 v novom automate. Tretia množina je tvorená stavmi B a F, resp. v novom automate to bude nový stav s názvom S_3. Nová tabuľka stavov automatu je uvedená v Tab. 10.2. Postup prepisu zredukovaného automatu do tabuľky nových stavov je detailne vysvetlený na prednáškach, ale v princípe môžeme finálny zápis množín E považovať za akýsi „prekladový slovník". Môžeme teda pracovať s pôvodnou tabuľkou a použiť prekladový slovník pre zápis novej tabuľky. Pôvodný stav A sa po novom bude volať S_1. Pri vstupe v logickej nule dôjde k prechodu do stavu B s výstupom v nule. Stav B sa po novom volá S_3 a výstup môžme opísať. Pri vstupe v logickej jednotke dôjde k prechodu do stavu D s výstupom v logickej nule, čo je v novom automate stav S_2 a výstup bude opäť identický. Nasledujúci riadok v novej tabuľke je pre stav S_2, ten je v pôvodnej tabuľke označený ako stav D a prechody má E/0 a D/0. Preto po novom sa tieto prechody zapíšu ako $S_1/0$ a $S_2/0$. Posledný riadok pre stav S_3 je v pôvodnej tabuľke stav B alebo F. Je jedno, ktorý stav si zvolíme. Ak sme v postupe nespravili chybu, musia všetky možnosti vyprodukovať identický zápis v novej tabuľke. V tomto príklade sme si zvolili stav B. Ten má prechody definované ako D/1 a A/1. Prepis pomocou „slovníka" do novej tabuľky je preto $S_2/1$ a $S_1/1$. Podľa novej tabuľky stavov (Tab. 10.2) môžeme následne nakresliť stavový diagram redukovaného automatu – Obr. 10.1.

Tab. 10.2: Definícia zredukovaného stavového automatu

	IN = 0	IN = 1
S_1	S_3 / 0	S_2 / 0
S_2	S_1 / 0	S_2 / 0
S_3	S_2 / 1	S_1 / 1

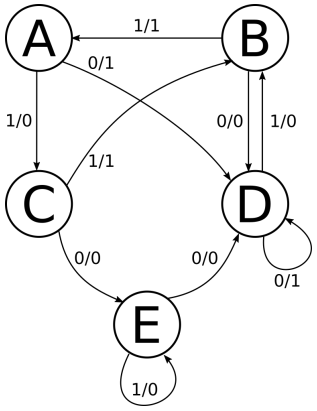


Obr. 10.1: Stavový diagram automatu po redukcii

Úloha: Zredukujte stavový automat definovaný na Obr. 10.2. Nakreslite diagram zredukovaného automatu a napíšte novú tabuľku prechodov automatu.
Riešenie: Pri redukcii potrebujeme najprv získať tabuľku stavov. Tvorba tabuľky z diagramu je intuitívny postup a ide vlastne o textový ekvivalent diagramu. Tabuľka automatu podľa obrázka je uvedená v Tab. 10.1.

Tab. 10.3: Odvodená tabuľka stavového automatu

	IN = 0	IN = 1
A	D / 1	C / 0
B	D / 0	A / 1
C	E / 0	B / 1
D	D / 1	B / 0
E	D / 0	E / 0



Obr. 10.2: Zadaný diagram stavového automatu

Teraz môžeme zapísať triedu ekvivalencie E_1 do tvaru.

$$E_1 = \{\{A, D\}\{B, C\}\{E\}\}$$

Vyšetríme ekvivalenciu stavov v rámci vyššej triedy, teda E_2.

$$A\ E_2\ D : (D\ E_1\ D) \wedge (C\ E_1\ B) \rightarrow A\ E_2\ D$$
$$B\ E_2\ C : (D\ \not{E_1}\ E) \wedge (A\ \not{E_1}\ B) \rightarrow B\ \not{E_2}\ C$$
$$E\ E_2\ E : (D\ E_1\ D) \wedge (E\ E_1\ E) \rightarrow E\ E_2\ E$$

Zo zápisu vidíme, že stavy B a C nie sú E_2 ekvivalentné, a teda množina sa bude rozkladať. Posledná kontrola je len pre úplnosť, pretože je zrejmé, že stav E bude ekvivalentný sám so sebou. Uvedieme teda E_2 v množinovom zápise.

$$E_2 = \{\{A, D\}\{B\}\{C\}\{E\}\}$$

Následne vyšetríme ekvivalentné stavy v rámci ďalšej triedy ekvivalencie, teda E_3

$$A\ E_3\ D : (D\ E_2\ D) \wedge (C\ \not{E_2}\ B) \rightarrow A\ \not{E_3}\ D$$
$$B\ E_3\ B : (D\ E_2\ E) \wedge (A\ E_2\ B) \rightarrow B\ E_3\ B$$
$$C\ E_3\ C : (D\ E_2\ D) \wedge (E\ E_2\ E) \rightarrow C\ E_3\ C$$
$$E\ E_3\ E : (D\ E_2\ D) \wedge (E\ E_2\ E) \rightarrow E\ E_3\ E$$

$$E_3 = \{\{A\}\{B\}\{C\}\{D\}\{E\}\}$$

Vidíme, že všetky množiny z pôvodnej triedy E_1 sa nám postupne rozdelili a vlastne sme sa vrátili do zápisu pôvodného automatu, kde je každý stav uvedený samostatne. Inými slovami, daný automat už nie je viac redukovateľný, a teda nedokážeme viac zlúčiť stavy a automat je už v najefektívnejšom tvare.

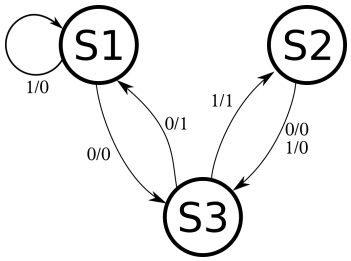
10.2 Príklady

Úloha 1: Zredukujte stavový automat definovaný v tabuľke na obrázku. Nakreslite diagram zredukovaného automatu a napíšte novú tabuľku prechodov automatu.

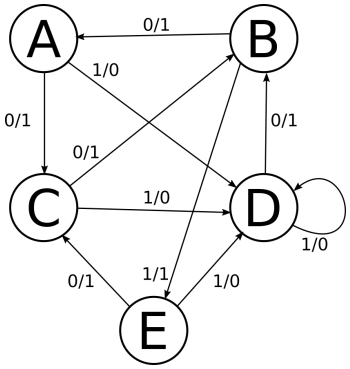
	IN = 0	IN = 1
A	B / 0	C / 0
B	C / 1	F / 1
C	E / 0	A / 0
D	B / 0	D / 0
E	A / 1	F / 1
F	E / 0	B / 0

Redukovaný automat:

	IN = 0	IN = 1
S_1	S_3 / 0	S_1 / 0
S_2	S_3 / 0	S_3 / 0
S_3	S_1 / 1	S_2 / 1

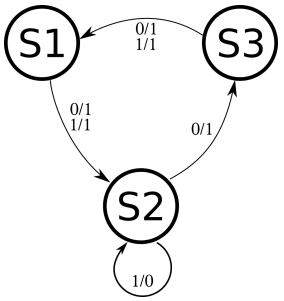


Úloha 2: Zredukujte stavový automat definovaný na obrázku. Nakreslite diagram zredukovaného automatu a napíšte novú tabuľku prechodov automatu.



Redukovaný automat:

	IN = 0	IN = 1
S_1	S_2 / 1	S_2 / 1
S_2	S_3 / 1	S_2 / 0
S_3	S_1 / 1	S_1 / 1

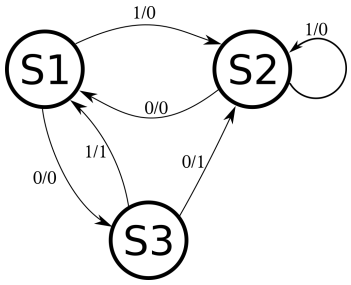


Úloha 3: Zredukujte stavový automat definovaný v tabuľke. Nakreslite diagram zredukovaného automatu a napíšte novú tabuľku prechodov automatu.

	IN = 0	IN = 1
A	B / 0	D / 0
B	D / 1	A / 1
C	F / 0	D / 0
D	E / 0	D / 0
E	F / 0	D / 0
F	D / 1	C / 1

Redukovaný automat:

	IN = 0	IN = 1
S_1	S_3 / 0	S_2 / 0
S_2	S_1 / 0	S_2 / 0
S_3	S_2 / 1	S_1 / 1



Úloha 4: Zredukujte stavový automat definovaný v tabuľke. Nakreslite diagram zredukovaného automatu a napíšte novú tabuľku prechodov automatu.

	IN = 0	IN = 1
A	B / 0	C / 0
B	C / 1	B / 0
C	D / 1	C / 0
D	F / 0	B / 0
E	F / 0	C / 0
F	A / 0	E / 1

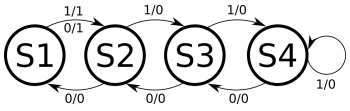
Automat nie je redukovateľný.

Úloha 5: Zredukujte stavový automat definovaný v tabuľke. Nakreslite diagram zredukovaného automatu a napíšte novú tabuľku prechodov automatu.

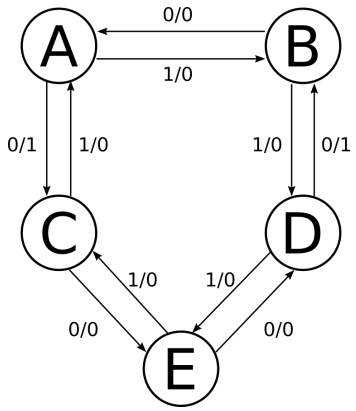
	IN = 0	IN = 1
A	B / 1	F / 1
B	A / 0	E / 0
C	F / 1	B / 1
D	B / 0	G / 0
E	F / 0	G / 0
F	C / 0	D / 0
G	E / 0	G / 0

Redukovaný automat:

	IN = 0	IN = 1
S_1	S_2 / 1	S_2 / 1
S_2	S_1 / 0	S_3 / 0
S_3	S_2 / 0	S_4 / 0
S_4	S_3 / 0	S_4 / 0



Úloha 6: Zredukujte stavový automat definovaný na obrázku. Nakreslite diagram zredukovaného automatu a napíšte novú tabuľku prechodov automatu.



Automat nie je redukovateľný.

Keďže všetky návrhy stavových automatov sú s jednobitovým vstupom, z každého stavu musí viesť presne jeden prechod pre prípad vstupu v logickej nule a druhý prechod pre prípad vstupu v logickej jednotke. Cieľový stav je v podstate ľubovoľný. Preto si môžeme vyskúšať vytvoriť ďalšie príklady povedzme so šesť alebo sedem stavovým automatom a náhodne si nakresliť prechody a názvy stavov v diagrame alebo priamo do tabuľky stavov. Pri kreslení diagramu však musíme dodržať, že z každého stavu musia viesť presne dva prechody von: jeden pre IN = 0 a druhý pre IN = 1. Výstupné hodnoty k jednotlivým prechodom si môžete opäť vymyslieť ľubovoľne, ale aj táto voľba ovplyvňuje možnosti redukcie automatu (kvôli E_1).

Návrh konečných stavových automatov

Návrh Mealyho konečného stavového automatu (FSM) v podstate pozostáva z návrhu kombinačnej časti, ktorá na základe aktuálnych vstupov a stavu v pamäti vypočíta aktuálne budiace premenné (nasledujúci stav) a aktuálny výstup. Pri návrhu Moorovho automatu navrhujeme dva bloky kombinačnej logiky. Jeden kombinačný blok bude pre výpočet budiacich premenných, druhá kombinačná časť slúži pre výpočet výstupnej funkcie, resp. výstupných hodnôt len na základe aktuálnych stavových premenných. Zameriame sa na návrh Mealyho automatu, nakoľko jeho návrh je väčšinou jednoduchší.

Berme do úvahy stavové diagramy z predchádzajúcej kapitoly. Avšak namiesto názvu stavov A, B, C atď. budeme pracovať s logickými nulami a jednotkami. Názov stavu „A" je v podstate len symbolické (pre ľudí čitateľné) označenie pre binárny kód uložený v registri vo forme stavovej premennej. Stav „B" a všetky ostatné sú tiež v skutočnosti len unikátna binárna hodnota zapísaná do pamäťovej časti. Predstavme si ľubovoľný stavový diagram s 10 stavmi. Na pokrytie 10 stavov bude musieť pamäťová časť (tvorená preklápacími obvodmi, resp. registrom) obsahovať minimálne 4 bity, ak použijeme priamy binárny kód. Pri niektorých zložitejších návrhoch sa nepoužíva systém kódovania s hustotou 2^N, ale napríklad tzv. „one-hot" kódovanie stavov, ktoré má hustotu N stavov pri N bitoch.

Postup návrhu sa opiera o prácu s mierne komplexnejšou pravdivostnou tabuľkou – tzv. *tabuľkou stavov*. V nej bude zaznamenaný požadovaný prechod medzi stavmi, výstupná funkcia a potom návrh kombinačnej časti pre výpočet budiacich premenných podľa zvoleného typu preklápacieho obvodu.

Kombinačná časť zabezpečujúca výpočet budúceho stavu (budiace premenné) musí totiž byť prispôsobená diagramu a zároveň aj funkcii pamäťového elementu.
Nutný základ je tzv. tabuľka prechodov pre jednotlivé preklápacie obvody. Tá je zapísaná v Tab. 11.1. Stav „X" značí stav, na ktorom nezáleží. Ide o stav, ktorý v Karnaughovej mape môžeme využiť raz ako logickú nulu a druhý raz ako logickú jednotku, podľa toho, ako nám to v danom momente vyhovuje. Túto možnosť využijeme pri tvorbe slučiek počas hľadania MDNF alebo MKNF hľadanej funkcie. Totiž, bunky K-mapy s označením „X" môžu zásadným spôsobom ovplyvniť minimalizáciu Booleovskej funkcie. S používaním stavu „X" sme sa mohli stretnúť na hodinách cvičení minimalizácie B-funkcií.

Tab. 11.1: Tabuľka prechodov pre všetky typy preklápacích obvodov

Q_{t-1}	Q_t	SR	JK	T	D
0	0	0X	0X	0	0
0	1	10	1X	1	1
1	0	01	X1	1	0
1	1	X0	X0	0	1

Tabuľka 11.1 by sa dala interpretovať nasledovne. V ľavej časti tabuľky prechodov máme prechody jedného bitu pamäte. Ide o prechod pamäte z „predchádzajúceho/starého" do „nasledujúceho/nového" stavu. Keďže máme dva stĺpce, celkovo to bude znamenať štyri možné kombinácie prechodov medzi stavmi. Na pravej strane tabuľky sú stĺpce pre jednotlivé preklápacie obvody. Ide o podmienky, ktoré treba na vstupe preklápacieho obvodu zabezpečiť, aby došlo k prechodu medzi stavmi z aktuálneho riadku tabuľky. Napríklad, uvažujme preklápací obvod T. V pamäti máme hodnotu „starého" stavu logická nula ($Q_{t-1} = 0$) a chceme do pamäte zapísať logickú jednotku ($Q_t = 1$). Ide teda o druhý riadok tabuľky. Preklápací obvod T potrebuje mať na vstupe T = 1, aby pri nasledujúcom takte hodinového signálu bola do pamäte zapísaná logická jednotka. K tomu dôjde, pretože T = 1 neguje predchádzajúcu hodnotu výstupu. V druhom príklade uvažujme preklápací obvod typu RS (RS obvod synchronizovaný hranou hodín), ktorý je v stave logickej nuly ($Q_{t-1} = 0$) a po nasledujúcom takte hodinového signálu chceme mať v pamäti opäť „novú" nulu ($Q_t = 0$), t. j. prvý riadok tabuľky.

Pri RS obvode máme dve možnosti, ako dosiahnuť tento prechod. Prvá možnosť je zapamätať si „starý" stav logickej nuly (S = 0, R = 0), alebo resetovať obsah pamäte bez ohľadu na to, čo v nej predtým bolo (S = 0, R = 1). Preto je v tabuľke zápis SR = 0X, lebo v oboch možnostiach je S = 0 a druhý vstup môže byť R = 0 alebo R = 1. Odporúčame neučiť sa tabuľku naspamäť. Výhodnejšie je naučiť sa jej porozumieť a odvodiť ju, pretože napríklad môže nastať prípad, že budeme mať k dispozícii preklápací obvod s negovanými vstupmi (napr. $\overline{J}$ a $\overline{K}$).
Najčastejší postup návrhu konečného stavového automatu sa začína ručným návrhom stavového diagramu. Ten potom redukujeme pomocou postupu, ktorému sme sa venovali v predchádzajúcej kapitole. Po redukcii stavov môžeme pristúpiť k samotnému návrhu FSM. Vo vstupnej (ľavej) časti tabuľky stavov budú stĺpce pre každý dátový vstup a stĺpce pre bity predchádzajúceho (starého) stavu automatu. V strednej časti tabuľky budú bity súčasného (nového) stavu a stĺpce výstupných dát (premenných). V poslednej časti tabuľky stavov bude opis kombinačnej logiky pre výpočet prechodov medzi stavmi. Musíme si zvoliť pamäťový element a potom pracovať s jeho tabuľkou prechodov. Typ preklápacieho obvodu budeme mať v príkladoch zadaný. V praxi sa najčastejšie využíva preklápací obvod typu D a typu JK. Po vyplnení celej tabuľky stavov môžeme pristúpiť k minimalizácii Booleovskej funkcie. Vytvoríme Karnaughove mapy pre každý potrebný stĺpec výstupnej funkcie a každý stĺpec kombinačnej logiky, ktorá vypočíta budiace premenné. Po minimalizácii použijeme De Morganove zákony pre optimalizáciu počtu tranzistorov v zapojení. Ako posledný krok nakreslíme elektrickú schému obvodu na hradlovej úrovni.

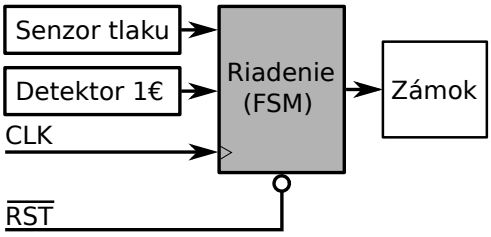
11.1 Vzorový príklad

Úloha: Navrhnite stavový automat, ktorý bude riadiť zámok turniketu. Vstupný digitálny signál predstavujú senzor vhodenej mince a senzor tlaku na zábranu, resp. začiatok rotácie zábrany. Všetky signály sú v pozitívnej logike.

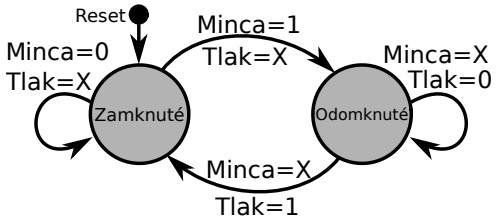
Riešenie: Ako prvé si nakreslíme blokovú schému celého návrhu – Obr. 11.1. Získame tak ilustračný obraz a prehľad o fungovaní, vstupoch a výstupoch atď. Naším cieľom je teda navrhnúť riadiaci digitálny logický obvod, ktorý odomyká a zamyká zámok na turnikete, na základe vstupu zo senzorov vhodenej mince a senzoru tlaku na zábranu. Hodinový signál (CLK) a signál resetu ($\overline{RST}$) sú systémové vstupy, ktoré sú potrebné pri návrhu akéhokoľvek stavového automatu. Reset môže byť v tomto prípade aj synchrónny, aj asynchrónny. Typ resetu nepredstavuje prioritu v tomto príklade, resp. návrhu FSM. Frekvencia hodinového signálu nemusí byť veľmi vysoká. Vstupné digitálne signály zo senzorov sú v podstate generované ľudskou činnosťou, teda pre vzorkovanie týchto signálov bude dostatočné aj pri frekvencii CLK = 1 kHz. Uvedomme si, že perióda vzorkovania vstupných signálov pri tejto frekvencii hodinového signálu je 1 ms. Čiže signály zo senzorov sú vzorkované tisíc krát za sekundu.

Stavový diagram na Obr. 11.2 by sme vedeli opísať približne takto. Po inicializačnom alebo vyžiadanom resete bude zámok na turnikete zamknutý. Ide teda o predvolený kľudový stav, v ktorom nedochádza ani k tlaku na zábranu, a ani nie je vhodená minca. Ak užívateľ zatlačí na zábranu, ale predtým nevhodil mincu, zámok ostane zamknutý. Táto funckia je zapísaná v diagrame prechodom na úplne ľavom konci obrázka. Prioritu má premenná s názvom „Minca". Pokiaľ užívateľ vhodí mincu (v našom prípade 1 €), zámok sa odomkne a ostane odomknutý až pokiaľ užívateľ neprejde zábranou. Podmienky pre prechod do odomknutého stavu vidíme vo vrchnej časti diagramu. Premenná „Minca" má opäť prioritu. Podmienky pre zotrvanie v odomknutom stave sú vidieť vpravo. V tomto prechode ma prioritu premenná „Tlak". Pokiaľ užívateľ nezatlačí na turniket, ostáva odomknuté. Ak užívateľ vhodí ďalšiu mincu, ostáva odomknuté, ale nezvýši sa počet možných rotácií turniketu. Akonáhle užívateľ zatlačí na zábranu, mechanizmus dovolí dokončiť rotáciu zábrany a riadiaca logika prejde prechodom do stavu, keď sa zámok zamkne. Toto je štandardné správanie automatu a turniketu ako celku.

Ide o prechod v spodnej časti diagramu a opäť vidíme, že prioritu má premenná „Tlak". Ak by užívateľ v odomknutom stave zatlačil na turniket a zároveň by znova vhodil mincu, zámok sa zamkne po rotácii zábrany, a teda užívateľ opäť nezíska vyšší počet rotácií. Samozrejme, nie je problém navrhnúť riadenie s opakovaním rotácie zábrany, toto riešenie je však hardvérovo zložitejšie. Preto zvolíme možnosť, že viacero vhodených mincí bude ignorovaných. Tento návrh je samozrejme ďaleko od reálneho nasadenia, ale na účely cvičenia návrhu FSM poslúži veľmi dobre aj takýto zjednodušený a samoúčelný príklad.



Obr. 11.1: Bloková schéma riadenia turniketu



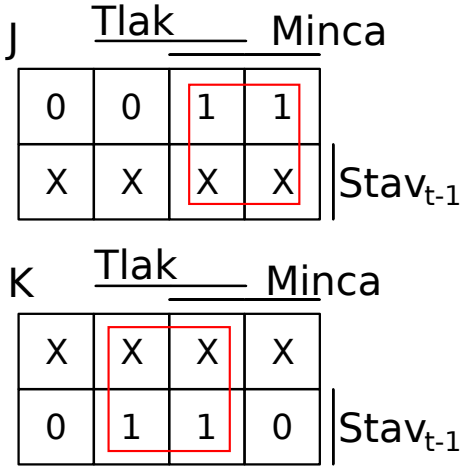
Obr. 11.2: Stavový diagram

Vidíme, že automat má len dva stavy. Budeme preto potrebovať iba jeden bit pamäte. Použijeme pre náš návrh JK preklápací obvod synchronizovaný nábežnou hranou hodinového signálu. Výstup preklápacieho obvodu je zároveň aj riadicim signálom pre zámok, teda výstup navrhovaného automatu. Opísané správanie riadenia turniketu je zapísané v tabuľke stavov – Tab. 11.2.

Tab. 11.2: Tabuľka stavov navrhovaného turniketu s preklápacím obvodom JK

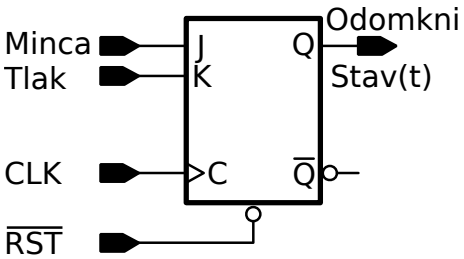
$Stav_{t-1}$	*Minca*	*Tlak*	$Stav_t$	*JK*	*Poznamka*
0	0	0	0	0X	Zamknuté, ostáva kľudový stav.
0	0	1	0	0X	Tlak na zábranu bez mince. Ostáva zamknuté.
0	1	0	1	1X	Vhodená minca, odomkne sa.
0	1	1	1	1X	Vhodená minca a tlak na zábranu súčasne.*
1	0	0	1	X0	Užívateľ ešte neprešiel turniketom, ostáva odomknuté.
1	0	1	0	X1	Užívateľ práve prechádza turniketom. Zamkne sa.
1	1	0	1	X0	Extra minca. Ostáva odomknuté.
1	1	1	0	X1	Extra minca + Tlak = Zamkne sa.

* Situáciu by sme chceli interpretovať nasledovne. Je zamknuté, užívateľ vhodí mincu a súčasne tlačí na zábranu. Ako prvé by sa mal odomknúť zámok, užívateľ prejde turniketom a na konci rotácie zábrany by sa mal zámok zamknúť. Zdanlivo jedna činnosť je v skutočnosti zložená z dvoch prechodov v rámci nášho návrhu FSM. Prvý prechod bude do „nového" odomknutého stavu, z ktorého okamžite prechádzame do ďalšieho „nového" zamknutého stavu. Tento prechod sa udeje po tlaku na zábranu a prejdením užívateľa. Ide však o separátny prechod definovaný v inom riadku tabuľky stavov automatu.
Stĺpec s budiacimi premennými J a K vypĺňame popri zameraní sa na stĺpce $Stav_{t-1}$ a $Stav_t$, pričom sledované prechody stavov treba zabezpečiť podľa tabuľky prechodov preklápacieho obvodu JK (Tabuľka 11.1). Týmto máme za sebou najväčšiu časť práce, nakoľko ďalší postup riešenia už poznáme. Vytvoríme K-mapy a vytvoríme slučky (Obr. 12.2). Keďže používame JK preklápací obvod, v K-mape sa vyskytuje veľa stavov X. Ako už bolo spomenuté, využijeme ich vo svoj prospech pri minimalizácii. Výstup minimalizácie nám vlastne určuje, že celé riadenie turniketu s vyššie uvedenými vlastnosťami dokážeme realizovať iba jediným JK preklápacím obvodom, ktorého J vstup bude budený senzorom vhodenej mince a vstup K bude budený senzorom tlaku na zábranu. Schéma zapojenia je na Obr. 12.3.



Obr. 11.3: Karnaughove mapy pre minimalizáciu funkcie pre stavovú premennú

$J = Minca$
$K = Tlak$

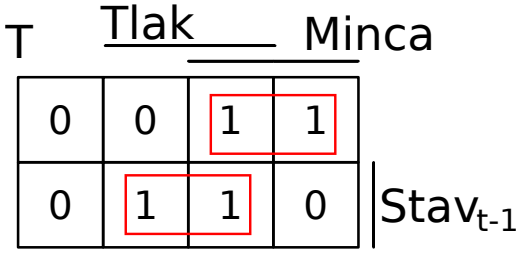


Obr. 11.4: Schéma zapojenia stavového automatu pre riadenie turniketu s JK preklápacím obvodom

Pre mierne zložitejší variant tohto istého príkladu, skúsme teraz navrhnúť ten istý stavový automat, ale tentokrát s T preklápacím obvodom – Tab. 11.3. Bloková schéma a stavový diagram sa nemení, ale tabuľka stavov bude obsahovať stĺpec pre iný preklápací obvod, tentokrát vyplnený na základe tabuľky prechodov T preklápacieho obvodu. K-mapa pre výpočet budiacej premennej pre preklápací obvod T je na Obr. 12.5.

Tab. 11.3: Tabuľka stavov navrhovaného turiniketu s preklápacím obvodom T

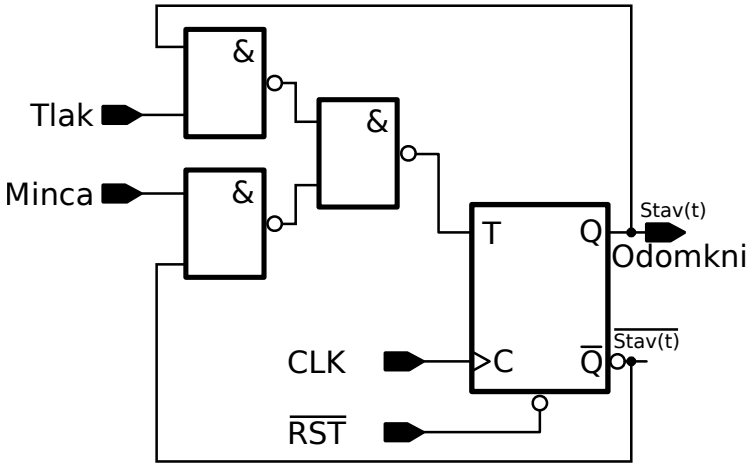
$Stav_{t-1}$	*Minca*	*Tlak*	$Stav_t$	*T*	*Poznamka*
0	0	0	0	0	Zamknuté, ostáva kľudový stav.
0	0	1	0	0	Tlak na zábranu bez mince. Ostáva zamknuté.
0	1	0	1	1	Vhodená minca, odomkne sa.
0	1	1	1	1	Vhodená minca a tlak na zábranu súčasne.*
1	0	0	1	0	Užívateľ ešte neprešiel turniketom, ostáva odomknuté.
1	0	1	0	1	Užívateľ práve prechádza turniketom. Zamkne sa.
1	1	0	1	0	Extra minca. Ostáva odomknuté.
1	1	1	0	1	Extra minca + Tlak = Zamkne sa.



$$T = Stav_{t-1}.Tlak + \overline{Stav_{t-1}}.Minca$$
$$= \overline{\overline{Stav_{t-1}.Tlak}.\overline{\overline{Stav_{t-1}}.Minca}}$$

Obr. 11.5: K-mapa pre výpočet budiacej premennej pre T preklápací obvod

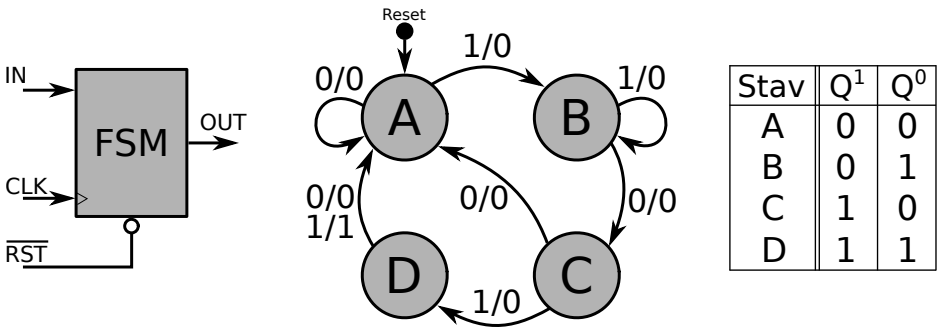
Pri pohľade na optimalizovanú MDNF pre T preklápací obvod vidíme, že budeme potrebovať tri dvojvstupové hradlá NAND. Priama, ako aj negovaná verzia premennej *Stav* je poskytovaná preklápacím obvodom ako funkcia času v závislosti od taktu hodín. Kombinačná logika spracováva vstupy zo senzorov a spätnoväzobnú informáciu o súčasnom stave. Následne vygeneruje signál pre T preklápací obvod, na základe ktorého sa zapíše „nový" stav do pamäte pri nasledujúcej nábežnej hrane hodinového signálu. Finálna schéma zapojenia je zobrazená na Obr. 12.6.
Pri porovnaní oboch schém je zrejmé, že implementácia s JK preklápacím obvodom je významne úspornejšia ohľadom počtu tranzistorov. V oboch prípadoch ide o veľmi jednoduchý príklad pre úvod do problematiky návrhu stavových automatov.



Obr. 11.6: Schéma zapojenia stavového automatu pre riadenie turniketu s T preklápacím obvodom

Skúste navrhnúť identické riadenie turniketu s D preklápacím obvodom, prípadne aj s RS preklápacím obvodom a porovnajte svoje výsledky s kolegami. Takisto skúste navrhnúť turnikety s inými vlastnosťami či prioritami pri určovaní prechodov.

Úloha: Navrhnite detektor sekvencie „1011" zo sériového jednobitového vstupného signálu ako Mealyho stavový automat. Výstup bude v logickej jednotke pri zachytení celej sekvencie. Neuvažujte prekrývanie sekvencií. Použite preklápací obvod D.
Riešenie: Začneme nakreslením blokovej schémy a odvodením stavového diagramu, ktoré sú zobrazené na Obr. 11.7.



Stav	Q^1	Q^0
A	0	0
B	0	1
C	1	0
D	1	1

Obr. 11.7: Bloková schéma, diagram stavov a kódovanie stavov detektora sekvencie „1011"

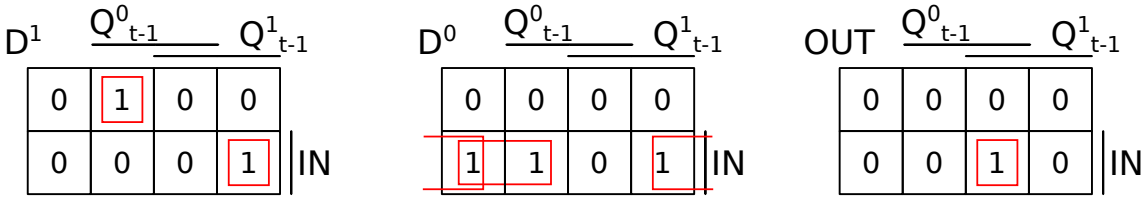
Stav „A" je počiatočný stav, z ktorého budeme začínať po inicializácii či vyžiadanom resete. V pamäti budú teda samé logické nuly. Ak na dátový vstup nášho detektora príde logická nula, nemá zmysel sa presúvať do nového stavu, keďže sme nezachytili prvú číslicu, resp. znak z našej hľadanej sekvencie. Výstup bude v podstate stále v logickej nule, okrem jediného prechodu, keď zaznamenáme nami hľadanú sekvenciu. Ak na vstup príde logická jednotka, prejdeme do stavu „B", pretože sme zachytili prvý bit z hľadanej sekvencie. Zo stavu „B" sa opäť môžeme pohnúť len dvomi smermi (pretože máme jednobitový dátový vstup). Ak príde na vstup logická jednotka, ostávame v tom istom stave. Stále totiž zaznamenávame iba prvú jednotku z celej hľadanej sekvencie. Ak bude na vstupe logická nula, prechádzame do nového stavu „C", zachytili sme druhú cifru zo sekvencie. Ak v tomto stave zachytíme na vstupe logickú nulu, vraciame sa do východzieho stavu. Pri zachytení logickej jednotky na vstupe prechádzame do stavu „D", máme zachytené tri zo štyroch bitov z hľadanej sekvencie. Zo stavu „D" sa presúvame do stavu „A" bez ohľadu na logickú hodnotu na dátovom vstupe. Rozdiel bude len vo výstupnej hodnote. Ak príde na vstup obvodu logická nula, prechod do východzieho stavu bude s logickou nulou na výstupe. Ak bude na vstupe logická jednotka, zachytili sme kompletnú sekvenciu a vraciame sa do východzieho stavu s tým, že výstup bude konečne v logickej jednotke. Označenie stavov A, B, C a D musíme previesť do binárnej číselnej sústavy. Keďže náš návrh FSM vyžaduje štyri stavy, budeme potrebovať dva bity pamäte (lebo $2^2 = 4$). Počiatočný stav „A" musí byť po resete kódovaný ako $\{Q^1\ Q^0\} = \{0\ 0\}$, ako sme už opísali vyššie. Ostatné stavy môžu byť prepísané v podstate ľubovoľne, ale najjednoduchšie bude klasické inkrementovanie hodnoty.

Na základe stavového diagramu teraz zostrojíme tabuľku stavov navrhovaného automatu – Tab. 11.4. V ľavej časti tabuľky sa nachádza dátový vstup IN a „starý" stav automatu. V strednej časti sa nachádza opis výstupu OUT a opis prechodov do „nového" stavu. V pravej časti tabuľky je prepísaný prechod medzi stavmi pre konkrétne preklápacie obvody.

Tab. 11.4: Tabuľka stavov navrhovaného detektora sekvencie „1011"

IN	Q^1_{t-1}	Q^0_{t-1}	Q^1_t	Q^0_t	OUT	D^1	D^0	*Prechod*
0	0	0	0	0	0	0	0	A → A
0	0	1	1	0	0	1	0	B → C
0	1	0	0	0	0	0	0	C → A
0	1	1	0	0	0	0	0	D → A
1	0	0	0	1	0	0	1	A → B
1	0	1	0	1	0	0	1	B → B
1	1	0	1	1	0	1	1	C → D
1	1	1	0	0	1	0	0	D → A

Po vytvorení tabuľky stavov môžeme pristúpiť priamo k tvorbe Karnaughových máp, minimalizácii a následnej optimalizácii automatu (Obr. 11.8).



Obr. 11.8: Karnaughove mapy odvodené z tabuľky stavov navrhovaného detektora

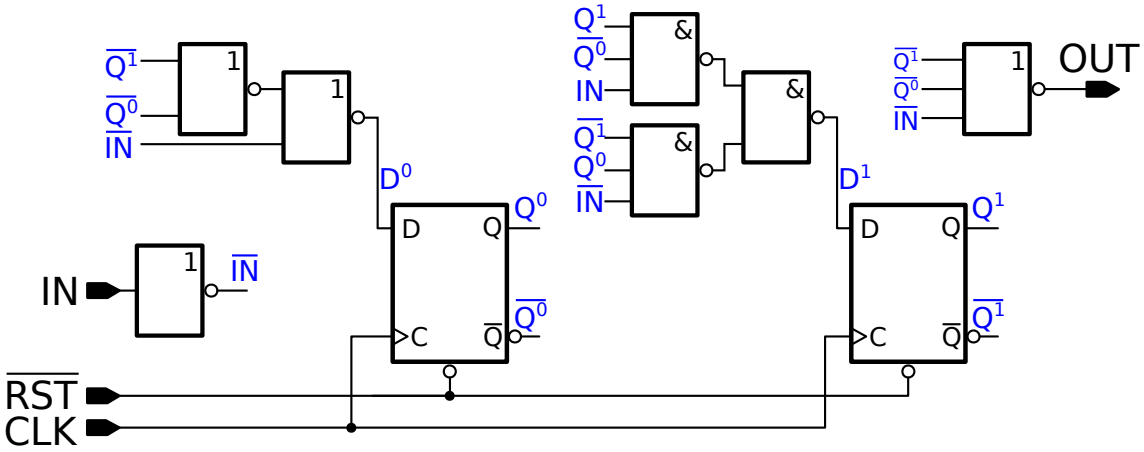
K-mapa pre výstupnú funkciu nie je v skutočnosti potrebná, pretože stĺpec má len jeden jednotkový bod. Môžeme teda napísať iba ÚDNF, ktorá bude zároveň aj MDNF. Pre úplnosť však K-mapu uvádzame.

$$\begin{aligned} OUT &= IN.Q^1_{t-1}.Q^0_{t-1} \\ &= \overline{\overline{IN} + \overline{Q^1_{t-1}} + \overline{Q^0_{t-1}}} \end{aligned}$$

Booleovské funkcie pre budiace premenné, ktoré budú ukladané v preklápacích obvodoch sú uvedené nižšie.

$$\begin{aligned} D^1 &= IN.Q^1_{t-1}.\overline{Q^0_{t-1}} + \overline{IN}.\overline{Q^1_{t-1}}.Q^0_{t-1} \\ &= \overline{\overline{IN.Q^1_{t-1}.\overline{Q^0_{t-1}}}.\overline{\overline{IN}.\overline{Q^1_{t-1}}.Q^0_{t-1}}} \end{aligned} \qquad \begin{aligned} D^0 &= IN.\overline{Q^1_{t-1}} + IN.\overline{Q^0_{t-1}} = IN.(\overline{Q^1_{t-1}} + \overline{Q^0_{t-1}}) \\ &= \overline{\overline{IN} + \overline{(\overline{Q^1_{t-1}} + \overline{Q^0_{t-1}})}} \end{aligned}$$

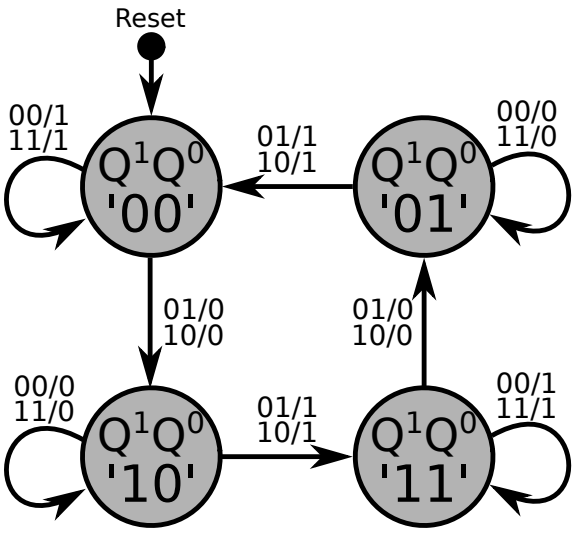
Vidíme, že pre budiacu premennú D^1 bude potrebné použiť dve trojvstupové hradlá NAND a jeden dvojvstupový NAND. Pre budiacu premennú D^0 budeme potrebovať dve dvojvstupové hradlá NOR. Pre realizáciu výstupnej premennej OUT bude potrebné iba jedno hradlo. Konkrétne trojvstupové hradlo NOR. Schéma na hradlovej úrovni sa nachádza na Obr. 11.9.



Obr. 11.9: Schéma zapojenia navrhnutého stavového automatu

Schéma bola nakreslená moderným spôsobom. Namiesto dlhých vodičov v schéme pomenujeme príslušné vstupy a výstupy hradiel, pričom vodiče s rovnakým názvom sa považujú za spojené do jedného uzla. Pri schémach obvodov s vyšším počtom hradiel a vodičov tento spôsob výrazne zvyšuje prehľadnosť a čitateľnosť a je preto preferovaný najmä pri práci s návrhovým EDA softvérom. Voľba je však na každom, ktorý spôsob kreslenia schémy použije.

Úloha: Navrhnite stavový automat podľa diagramu na Obr. 11.10. Vstupné premenné sú nazvané A a B. Pre pamäťový bit Q^1 použite preklápací obvod T, pre pamäťový bit Q^0 použite preklápací obvod D citlivý na nábežnú hranu s asynchrónnym resetom.



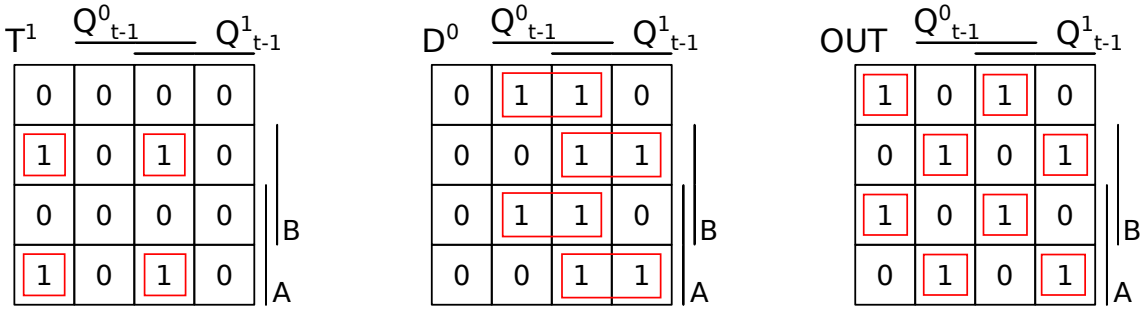
Obr. 11.10: Stavový diagram navrhovaného konečného stavového automatu

Riešenie: Keďže navrhovaný automat má dvojbitový dátový vstup, z každého stavu musia viesť presne štyri prechody. Pri tvorbe tabuľky stavov (Tab. 11.5) navrhnutého automatu postupujeme úplne identicky ako v predchádzajúcich príkladoch. Tabuľka bude len dvojnásobne väčšia.

Tab. 11.5: Tabuľka stavov navrhovaného stavového automatu

A	B	Q^1_{t-1}	Q^0_{t-1}	Q^1_t	Q^0_t	OUT	T^1	D^0	*Prechod*
0	0	0	0	0	0	1	0	0	A → A
0	0	0	1	0	1	0	0	1	B → B
0	0	1	0	1	0	0	0	0	C → C
0	0	1	1	1	1	1	0	1	D → D
0	1	0	0	1	0	0	1	0	A → C
0	1	0	1	0	0	1	0	0	B → A
0	1	1	0	1	1	1	0	1	C → D
0	1	1	1	0	1	0	1	1	D → B
1	0	0	0	1	0	0	1	0	A → C
1	0	0	1	0	0	1	0	0	B → A
1	0	1	0	1	1	1	0	1	C → D
1	0	1	1	0	1	0	1	1	D → B
1	1	0	0	0	0	1	0	0	A → A
1	1	0	1	0	1	0	0	1	B → B
1	1	1	0	1	0	0	0	0	C → C
1	1	1	1	1	1	1	0	1	D → D

Karnaughove mapy pre budiace premenné a výstupnú funckiu sú zobrazené nižšie na Obr. 11.11. Z tvaru a rozloženia slučiek (sú umiestnené „cik-cak") môžeme dedukovať, že vo výstupnej funkcii bude prítomný exkluzívny logický súčet.



Obr. 11.11: Karnaughove mapy pre budiace a výstupné premenné

Štyri slučky pre premennú T^1 nám dávajú výslednú MDNF bez akejkoľvej minimalizácie. Avšak po algebrickej úprave a použití De Morganovych pravidiel dostávame jednoduchý výraz s iba tromi operáciami.

$$\begin{aligned} T^1 &= A.\overline{B}.\overline{Q^1_{t-1}}.\overline{Q^0_{t-1}} + A.\overline{B}.Q^1_{t-1}.Q^0_{t-1} + \overline{A}.B.\overline{Q^1_{t-1}}.\overline{Q^0_{t-1}} + \overline{A}.B.Q^1_{t-1}.Q^0_{t-1} \\ &= (\overline{Q^1_{t-1} \oplus Q^0_{t-1}}).(A \oplus B) \\ &= \overline{(Q^1_{t-1} \oplus Q^0_{t-1}) + (\overline{A \oplus B})} \end{aligned}$$

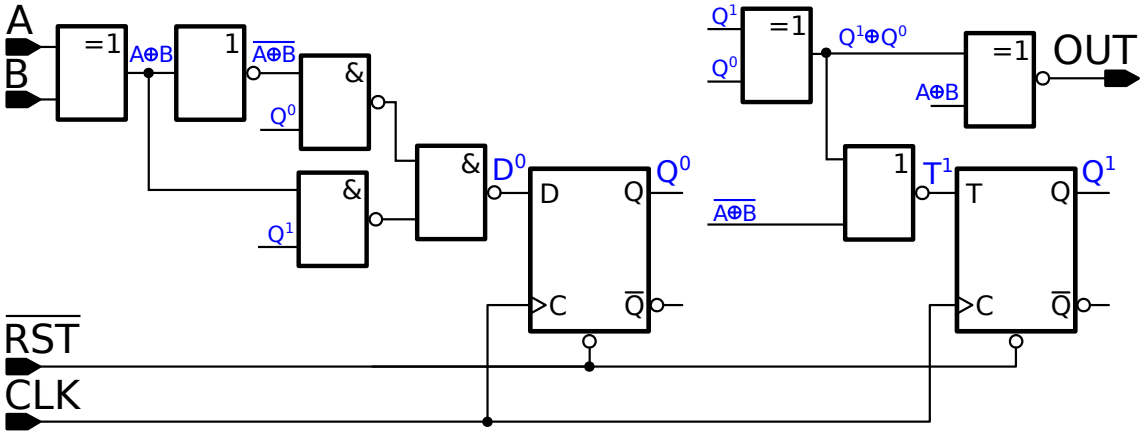
Budiaca premenná D^0 je opäť zložená zo štyroch slučiek s dvojicou jednotkových bodov. Po algebrickej úprave opäť dostávame rovnicu zloženú z troch dvojvstupových hradiel NAND a (negovaný) exkluzívny súčet medzi vstupnými premennými.

$$\begin{aligned} D^0 &= A.\overline{B}.Q^1_{t-1} + \overline{A}.B.Q^1_{t-1} + A.B.Q^0_{t-1} + \overline{A}.\overline{B}.Q^0_{t-1} \\ &= Q^1_{t-1}.(A.\overline{B} + \overline{A}.B) + Q^0_{t-1}.(A.B + \overline{A}.\overline{B}) \\ &= (A \oplus B).Q^1_{t-1} + (\overline{A \oplus B}).Q^0_{t-1} \\ &= \overline{\overline{(A \oplus B).Q^1_{t-1}}.\overline{(\overline{A \oplus B}).Q^0_{t-1}}} \end{aligned}$$

Výstupná funkcia je takisto učebnicová aplikácia operácie NXOR medzi štyrmi premennými, či už na úrovni PT, alebo K-mapy. Pre prehľadnosť uvádzame iba finálny zápis bez MDNF vyšetrovanej funkcie.

$$OUT = \overline{A \oplus B \oplus Q^1_{t-1} \oplus Q^0_{t-1}}$$

Pri kreslení schémy využijeme fakt, že v oboch definíciách pre budiace premenné, ako aj výstupná funkcia využívajú operáciu XOR, resp. NXOR medzi vstupnými a stavovými premennými. Hradlo XOR použijeme teda iba raz a výsledok výpočtu použijeme, kde to bude potrebné. Navyše, namiesto hradla NXOR pri výpočte premennej D^0 použijeme iba invertor po už vypočítanej operácii XOR. Invertor totiž vyžaduje menej tranzistorov ako hradlo NXOR, nakoľko ide o najjednoduchšie hradlo vôbec. Celková schéma zapojenia na hradlovej úrovni je zobrazená na Obr. 11.12.



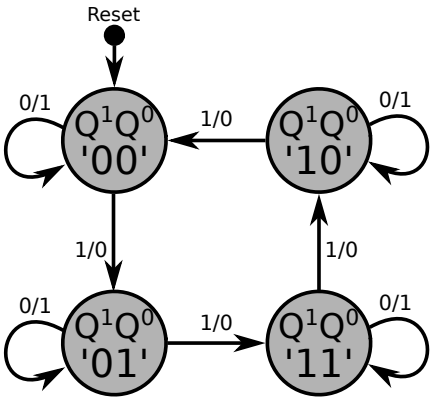
Obr. 11.12: Schéma zapojenia stavového automatu

11.2 Príklady

Úloha 1: Navrhnite stavový automat – detektor sekvencie „1101". Pre pamäťovú časť použite preklápací obvod JK citlivý na dobežnú hranu.

Úloha 2: Navrhnite stavový automat – detektor sekvencie „1X01". Pre pamäťovú časť použite preklápací obvod D citlivý na nábežnú hranu.

Úloha 3: Navrhnite stavový automat podľa diagramu na obrázku. Pre pamäťovú časť použite preklápací obvod RS/JK/T/D citlivý na nábežnú hranu.



Počítadlá a ich návrh

Počítadlo (anglicky „counter") je jednoduchý digitálny obvod s veľmi širokým uplatnením v praxi. Takmer v každom zložitejšom sekvenčnom digitálnom obvode nájdeme aspoň jedno počítadlo. Pred samotným návrhom počítadla si musíme zadefinovať, aké typy počítadiel vlastne poznáme. Základné delenie je podľa synchronizácie hodinovým signálom. Synchrónne počítadlo je v podstate klasický Mooreov stavový automat, kde budiace a stavové premenné reprezentujú čísla v binárnom kóde. Ide o prípad, keď hodinový signál budí všetky preklápacie obvody naraz (paralelne). Postup návrhu synchrónneho počítadla sa nijako neodlišuje od návrhu klasického FSM z predchádzajúcej kapitoly.
Asynchrónne počítadlo využíva zapojenie preklápacích obvodov do kaskády a hodinovým signálom je tak priamo budený iba preklápací obvod zodpovedný za ukladanie LSB výstupného čísla.
Pri návrhu synchrónneho počítadla môžeme použiť ľubovoľný preklápací obvod a kombinačnú logiku k nemu navrhnúť podľa požiadaviek zadania. Na druhej strane asynchrónne počítadlo je postavené z bloku, ktorý sa správa ako T preklápací obvod s funkciou *Toggle* v trvalej logickej jednotke (T = 1). Takéto správanie však vieme dosiahnuť viacerými spôsobmi, my sa ale zameriame na zapojenie D preklápacieho obvodu v spätnej väzbe s jeho negovaným výstupom.
Kriticky dôležitou funkciou pri návrhu obvodu s preklápacími obvodmi je ich funkcia reset, resp. set. Táto funkcia je aktivovaná vždy pre návrat do počiatočného (inicializačného) stavu. Tu si musíme dávať pozor na slovíčka, pretože aktivita signálu reset (niekedy aj clear) značí, že sa celé počítadlo prepíše logickými nulami. Funkcia set (niekedy aj preset) značí, že sa celý obsah pamäte prepíše logickými jednotkami. Navrhujeme, aby sme pri počítadlách návrat do počiatočného stavu (resp. čísla) nazývali *reštart*.

Predstavme si situáciu, že počiatočný stav počítadla je napr. číslo 64 a počítadlo bude dekrementovať obsah svojej pamäte s taktom hodinového signálu. Stav pamäťe bude po reštarte teda v binárnom kóde 1000000. Takúto operáciu nemôžeme volať reset, pretože MSB pamäte v skutočnosti využíva funkciu set.

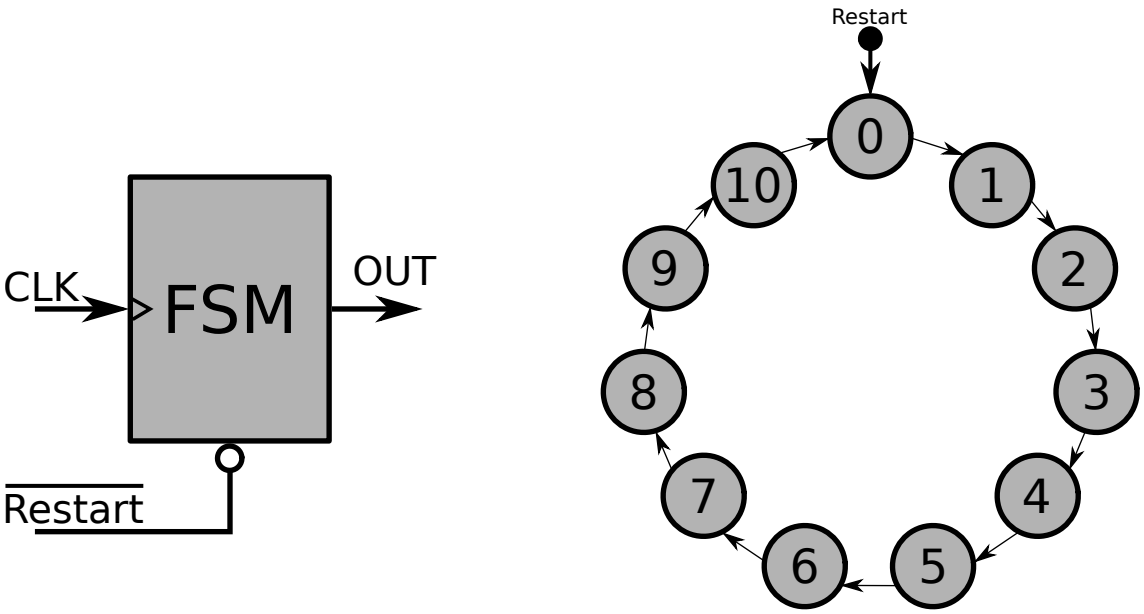
12.1 Vzorový príklad

12.1.1 Synchrónny návrh

Ako už bolo spomenuté, návrh synchrónneho počítadla sa nijako neodlišuje od návrhu klasického FSM. Jediným rozdielom je vo väčšine prípadov, že výstup počítadla predstavuje priamo obsah pamäte bez pridanej kombinačnej logiky. Nie je to však pravidlo.

Úloha: Navrhnite synchrónne počítadlo smerom vpred od 0 do 10 s krokom 1. Použite preklápací obvod T.

Riešenie: Nakreslíme si blokovú schému a diagram stavov (Obr. 12.1), pomocou ktorého opäť vyplníme tabuľku stavov. Keďže spolu máme 11 stavov, budeme teda potrebovať štyri bity pamäte.



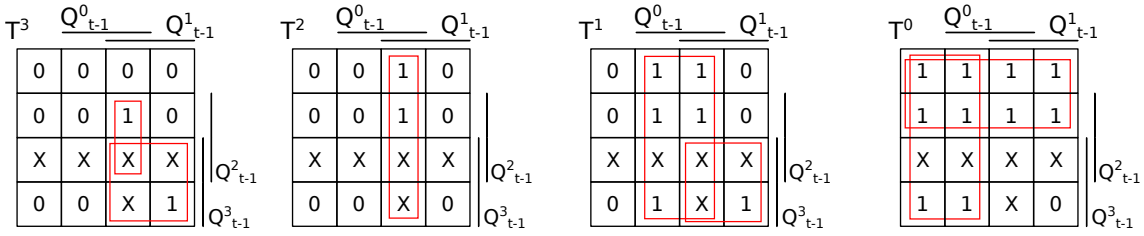
Obr. 12.1: Bloková schéma navrhovaného počítadla a jeho stavový diagram

Prvý riadok tabuľky stavov (Tab. 12.1) opisuje prechod z dekadickej nuly do dekadickej jednotky, nasledujúci riadok opisuje prechod z dekadickej jednotky do dekadickej dvojky atď. Posledný riadne definovaný riadok opisuje prechod z dekadického čísla 10 naspäť do čísla nula. Ďalšie riadky tabuľky by mali opisovať prechody z čísiel (11 a viac), ktoré pri správne navrhnutom počítadle nemôžu a nesmú nastať. Môžeme teda cieľovú hodnotu z týchto čísiel zapísať ako „stav, na ktorom nezáleží".

Tab. 12.1: Tabuľka stavov navrhovaného počítadla

Q^3_{t-1}	Q^2_{t-1}	Q^1_{t-1}	Q^0_{t-1}	Q^3_t	Q^2_t	Q^1_t	Q^0_t	T^3	T^2	T^1	T^0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	1
0	0	1	1	0	1	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	1
0	1	0	1	0	1	1	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	1
0	1	1	1	1	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	0	0	0	1
1	0	0	1	1	0	1	0	0	0	1	1
1	0	1	0	0	0	0	0	1	0	1	0
1	0	1	1	X	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X	X

Karnaughova mapa na Obr. 12.2, výstupné MDNF funkcie a ich následná úprava nevyžaduje špeciálny komentár. Ide o klasický postup, ktorý sme vykonávali už mnohokrát.

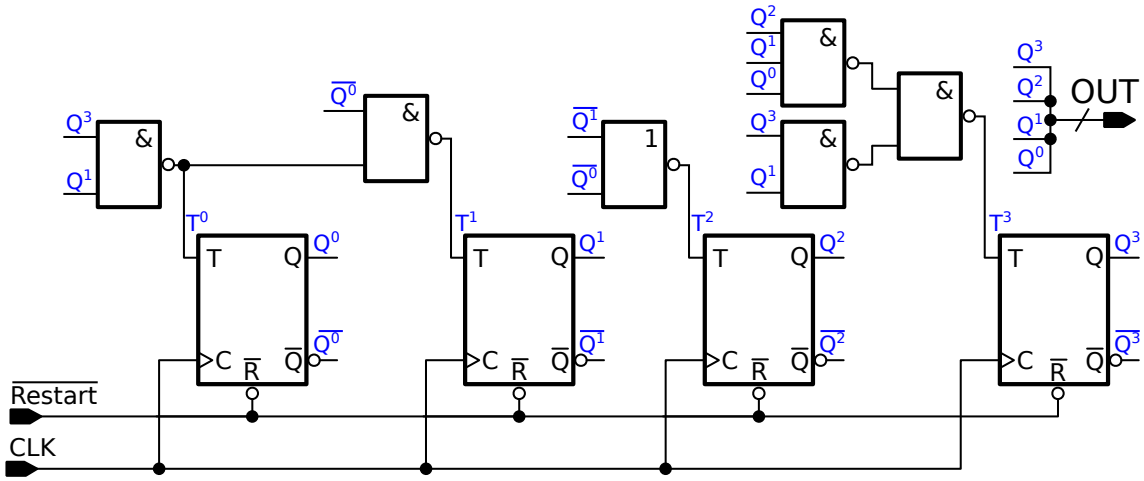


Obr. 12.2: K-mapy pre budiace premenné navrhovaného počítadla

$$T^3 = Q^3_{t-1}.Q^1_{t-1} + Q^2_{t-1}.Q^1_{t-1}.Q^0_{t-1} \qquad T^2 = Q^1_{t-1}.Q^0_{t-1}$$
$$= \overline{\overline{Q^3_{t-1}.Q^1_{t-1}}.\overline{Q^2_{t-1}.Q^1_{t-1}.Q^0_{t-1}}} \qquad = \overline{\overline{Q^1_{t-1}} + \overline{Q^0_{t-1}}}$$

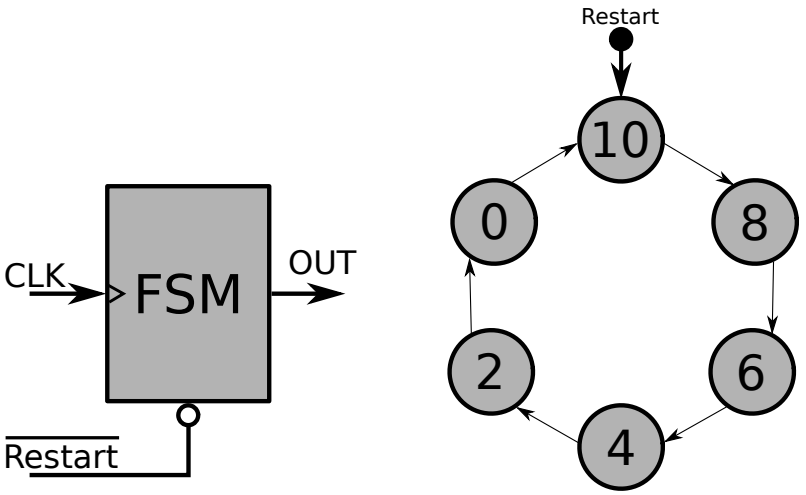
$$T^1 = Q^0_{t-1} + Q^3_{t-1}.Q^1_{t-1} \qquad T^0 = \overline{Q^3_{t-1}} + \overline{Q^1_{t-1}}$$
$$= \overline{\overline{Q^0_{t-1}}.\overline{Q^3_{t-1}.Q^1_{t-1}}} \qquad = \overline{Q^3_{t-1}.Q^1_{t-1}}$$

Schému počítadla nakreslíme prehľadným spôsobom – Obr. 12.3. Ako už bolo spomenuté vyššie, základné počítadlá väčšinou nemajú dátové vstupy a ani pridanú kombinačnú logiku pre výstupnú funkciu. Výstupom nášho počítadla je samotné číslo uložené v pamäti. V tomto konkrétnom prípade ide o bity Q^0 až Q^3 privedené priamo na štvorbitový paralelný výstup označený ako OUT.



Obr. 12.3: Schéma zapojenia navrhnutého počítadla

Úloha: Navrhnite synchrónne počítadlo smerom vzad od 10 do 0 krokom 2. Použite preklápací obvod D.
Riešenie: Ako v predchádzajúcom prípade, nakreslíme si blokovú schému a stavový diagram – Obr. 12.4 navrhovaného počítadla.



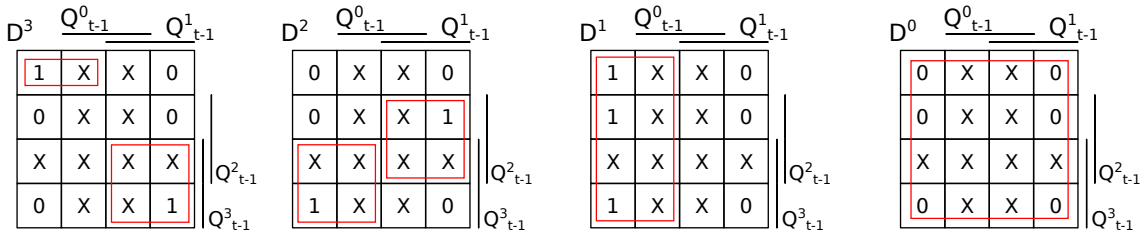
Obr. 12.4: Bloková schéma a stavový diagram počítadla smerom vzad

Po reštarte pristávame v stave, ktorý opisuje dekadické číslo 10 v binárnom kóde. To znamená, že Q^3 a Q^1 využijú funkciu set a Q^2 a Q^0 využijú funkciu reset, aby sme docielili číslo $1010_B = 10_D$. Tabuľka stavov zobrazená v Tab. 12.2 pre navrhované počítadlo je vyplnená klasicky, riadok po riadku. Nedefinované riadky opäť vyplníme hodnotou „X", teda stavom, na ktorom nezáleží.

Tab. 12.2: Tabuľka stavov navrhovaného stavového automatu

Q^3_{t-1}	Q^2_{t-1}	Q^1_{t-1}	Q^0_{t-1}	Q^3_t	Q^2_t	Q^1_t	Q^0_t	D^3	D^2	D^1	D^0
0	0	0	0	1	0	1	0	1	0	1	0
0	0	0	1	X	X	X	X	X	X	X	X
0	0	1	0	0	0	0	0	0	0	0	0
0	0	1	1	X	X	X	X	X	X	X	X
0	1	0	0	0	0	1	0	0	0	1	0
0	1	0	1	X	X	X	X	X	X	X	X
0	1	1	0	0	1	0	0	0	1	0	0
0	1	1	1	X	X	X	X	X	X	X	X
1	0	0	0	0	1	1	0	0	1	1	0
1	0	0	1	X	X	X	X	X	X	X	X
1	0	1	0	1	0	0	0	1	0	0	0
1	0	1	1	X	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X	X

K-mapy pre budiace premenné sú zobrazené na Obr. 12.5.



Obr. 12.5: Stavový diagram konečného stavového automatu

Jediná K-mapa, ktorú je vhodné komentovať, je pre budiacu premennú D^0. Vidíme jednu veľkú slučku so šestnástimi nulovými bodmi. Keďže v takejto slučke všetky vstupné premenné menia svoj stav, nemôže teda ani jedna z nich vystupovať vo výstupnej funkcii. Celý obsah slučky sa rovná nule, preto výstup budiacej premennej D^0 a stavovej premennej Q^0 sa bude rovnať nule bez ohľadu na kombináciu vstupných hodnôt a obsahu pamäte.

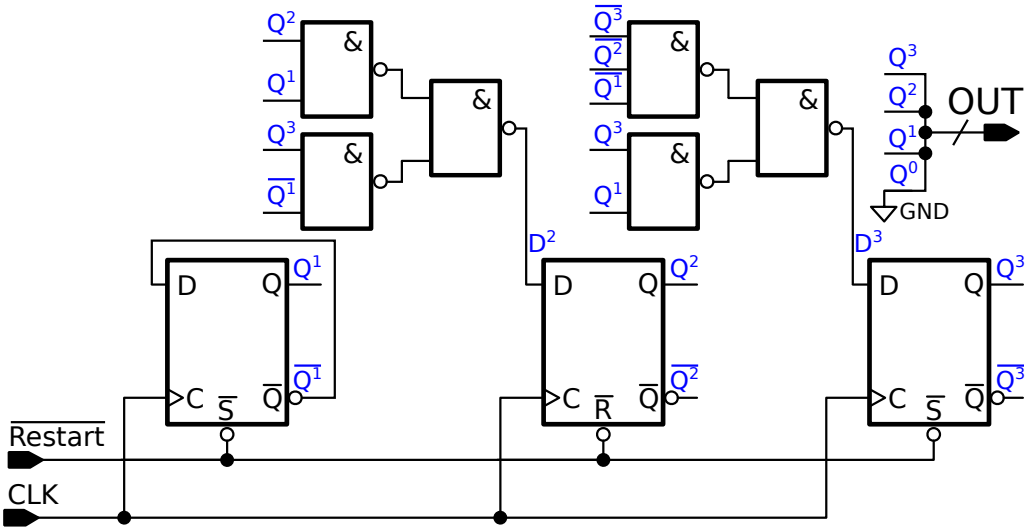
$$D^3 = Q^3_{t-1}.Q^1_{t-1} + \overline{Q^3_{t-1}}.\overline{Q^2_{t-1}}.\overline{Q^1_{t-1}} = \overline{\overline{Q^3_{t-1}.Q^1_{t-1}}.\overline{\overline{Q^3_{t-1}}.\overline{Q^2_{t-1}}.\overline{Q^1_{t-1}}}}$$

$$D^2 = Q^3_{t-1}.\overline{Q^1_{t-1}} + Q^2_{t-1}.Q^1_{t-1} = \overline{\overline{Q^3_{t-1}.\overline{Q^1_{t-1}}}.\overline{Q^2_{t-1}.Q^1_{t-1}}}$$

$$D^1 = \overline{Q^1_{t-1}}$$

$$D^0 = 0$$

Pri schéme na Obr. 12.6 opäť stojí za zmienku bit Q^0. Keďže bez ohľadu na kombináciu dát bude bit D^0 resp. Q^0 natrvalo v logickej nule, nie je potrebné použiť pamäťový obvod, ale stačí príslušný výstupný bit Q^0 pripojiť „natrvalo" na hodnotu logickej nuly. Zakreslenie „trvalej" nuly môžeme vykonať viacerými spôsobmi, pripojenie na zem (GND) je len jeden z nich.

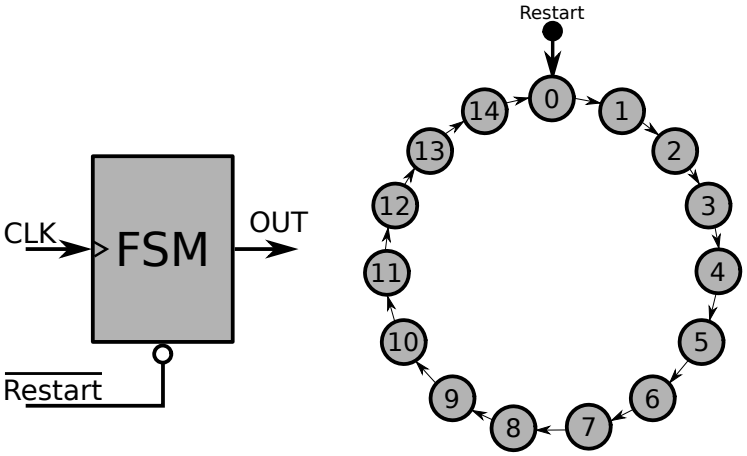


Obr. 12.6: Hradlová schéma synchrónneho počítadla smerom vzad

12.1.2 Asynchrónny návrh

Na rozdiel od synchrónneho návrhu, pri asynchrónnom počítadle sme obmedzení iba na možnosť počítania s krokom 1. Tento fakt je priamo spojený s vnútorným zapojením asynchrónneho počítadla a v princípe ide o daň za jednoduchosť návrhu takéhoto obvodu. Asynchrónne počítadlá môžu počítať smerom vpred, ako aj smerom vzad. Skrátenie cyklu pre tzv. *neúplné počítadlá* je dosiahnuté pomocou asynchrónneho reštartu, teda použité preklápacie obvody musia obsahovať *asynchrónnu* funkciu set/reset. Na účely príkladov návrhu asynchrónneho počítadla prejdeme priamo k návrhu neúplných počítadiel, nakoľko návrh asynchrónneho počítadla s počtom stavov 2^N je v podstate len zapojenie N preklápacích obvodov do kaskády.

Úloha: Navrhnite asynchrónne počítadlo smerom vpred od 0 do 14 s krokom 1.
Riešenie: Je evidentné, že na realizáciu počítadla budeme potrebovať štyri bity – Q^3 až Q^0. Keďže počítadlo bude inkrementovať obsah svojej pamäte, použijeme kaskádu preklápacích obvodov s budením z invertovaného výstupu a navrhneme asynchrónne skrátenie cyklu (interný reštart). Potom zlúčime vnútorné skrátenie cyklu s hlavným vonkajším/užívateľským reštartom. Bloková schéma na Obr. 12.7 sa nijako nelíši od predchádzajúcich príkladov a spôsob nákresu diagramu stavov takisto ostáva nezmenený.



Obr. 12.7: Bloková schéma a diagram stavov asynchrónneho počítadla

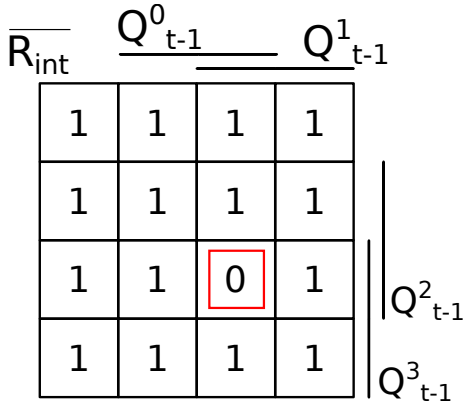
Tabuľka stavov počítadla je uvedená v Tab. 12.3. Opäť sú prítomné stavy „X" pri hodnotách, ktoré pri správnom návrhu a fungovaní nemôžu nastať. V našom prípade ide o posledný riadok tabuľky. Je potrebné uviesť, že prechody medzi stavmi v pamäti zabezpečuje samotná topológia počítadla a nie je potrebná žiadna prídavná kombinačná logika. Čo je však kriticky dôležité, je stĺpec opisujúci vnútorné skrátenie cyklu – $\overline{R_{int}}$. Vo významnej väčšine prípadov v praxi je funkcia set/reset aktivovaná logickou nulou, a preto použijeme tento prístup aj v našom príklade. Interné skrátenie cyklu alebo interný reštart bude teda aktivovaný až pri prechode z dekadického čísla 14.

Tab. 12.3: Tabuľka stavov navrhovaného stavového automatu

Q^3_{t-1}	Q^2_{t-1}	Q^1_{t-1}	Q^0_{t-1}	Q^3_t	Q^2_t	Q^1_t	Q^0_t	$\overline{R_{int}}$
0	0	0	0	0	0	0	1	1
0	0	0	1	0	0	1	0	1
0	0	1	0	0	0	1	1	1
0	0	1	1	0	1	0	0	1
0	1	0	0	0	1	0	1	1
0	1	0	1	0	1	1	0	1
0	1	1	0	0	1	1	1	1
0	1	1	1	1	0	0	0	1
1	0	0	0	1	0	0	1	1
1	0	0	1	1	0	1	0	1
1	0	1	0	1	0	1	1	1
1	0	1	1	1	1	0	0	1
1	1	0	0	1	1	0	1	1
1	1	0	1	1	1	1	0	1
1	1	1	0	0	0	0	0	1
1	1	1	1	X	X	X	X	0

Do momentu pokiaľ nepríde k synchronizačnej nábežnej hrane hodín, nie je dôvod na aktivovanie interného reštartu, pretože stále zotrvávame v čísle 14. Samotná topológia sa totiž pri synchronizačnej udalosti (hrana hodinového signálu) bude snažiť prejsť z čísla 14 do čísla 15, ale nami navrhované skrátenie cyklu počítadlo nakoniec nastaví do čísla nula.

Inými slovami, počítadlo sa pri hrane hodín snaží prejsť z čísla 14 do čísla 15, nami navrhnutý obvod však tento pokus o prechod zachytí a vyvolá interný reštart aktivovaný logickou nulou. Preto sa výstupná hodnota bude javiť ako prechod z čísla 14 priamo do čísla 0, aj keď v skutočnosti bude počítadlo na „zanedbateľne krátky" okamih v hazarde, nakoľko sa bude neúspešne snažiť prejsť do čísla 15. Ešte inými slovami môžeme povedať, že v podstate sme vyvinuli detektor čísla 15, ktorý bude aktivovať interný reštart.
Nasledujúcim krokom je zostrojenie Karnaughovej mapy pre stĺpec $\overline{R_{int}}$. Táto je na Obr. 12.8. Už z Tab. 12.3 vyššie je zrejmé, že funkcia má len jeden nulový bod. Teda najefektívnejšie bude priamo zapísať ÚKNF, ale pre úplnosť uvádzame K-mapu aj so slučkou pre MKNF = ÚKNF.



$$\overline{R_{int}} = \overline{Q^3_{t-1}} + \overline{Q^2_{t-1}} + \overline{Q^1_{t-1}} + \overline{Q^0_{t-1}}$$
$$= \overline{Q^3_{t-1}.Q^2_{t-1}.Q^1_{t-1}.Q^0_{t-1}}$$

Obr. 12.8: K-mapa skrátenia cyklu/interného reštartu asynchrónneho počítadla

Posledným krokom bude zavedenie vonkajšieho užívateľského reštartu a jeho zlúčenie s interným reštartom. Tento krok pozostáva z jednoduchej pravdivostnej tabuľky o dvoch vstupných premenných – $\overline{Restart}$ a $\overline{R_{int}}$ a jeden výstupný zlúčený reštart, ktorý nazveme $\overline{R}$. Spomínaná PT je uvedená v Tab. 12.4. Prvý riadok interpretujeme ako prípad, keď je počítadlo práve reštartované interne a zároveň aj zvonku užívateľom. Nie je preto pochýb, že zlúčený reštart $\overline{R}$ bude taktiež v logickej nule. Druhý riadok opisuje situáciu, keď je počítadlo s obsahom pamäte medzi stanovenými číselnými hranicami intervalu (v tomto príklade medzi 0 a 14), ale užívateľ aktivoval vonkajší reset. Spoločný signál bude preto opäť v logickej nule, aby došlo k požadovanému reštartu.

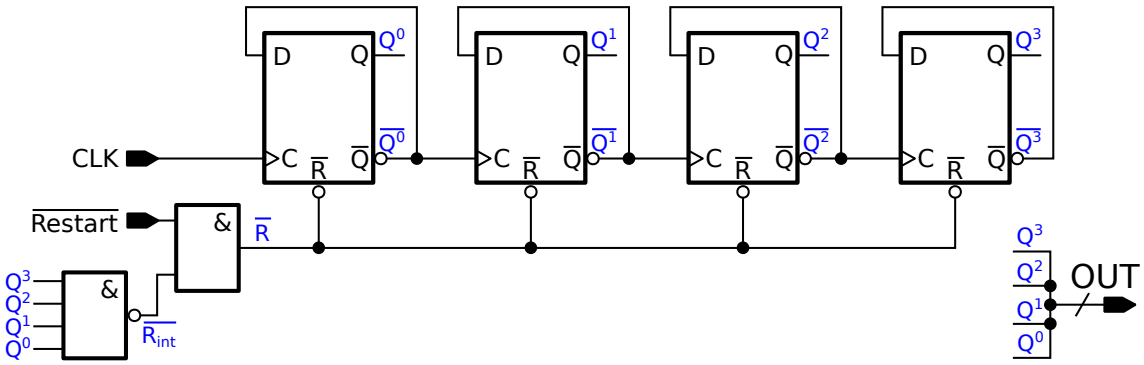
Tretí riadok opisuje situáciu, keď užívateľ neaktivuje vonkajší reštart, ale počítadlo je práve interne reštartované. Výstup bude preto opäť v logickej nule, aby došlo ku skráteniu cyklu. Posledný riadok opisuje stav, keď užívateľ neaktivuje reštart a počítadlo je opäť medzi stanovenými číselnými hranicami intervalu. Je evidentné, že tabuľka je učebnicový príklad logickej funkcie AND medzi dvomi vstupnými premennými. Nebudeme preto ani vytvárať Karnaughovu mapu a pristúpime priamo k zápisu Booleovskej funkcie pre zlúčený reštart $\overline{R}$.

Tab. 12.4: Zlúčenie interného a externého užívateľského reštartu

$\overline{Restart}$	$\overline{R_{int}}$	$\overline{R}$
0	0	0
0	1	0
1	0	0
1	1	1

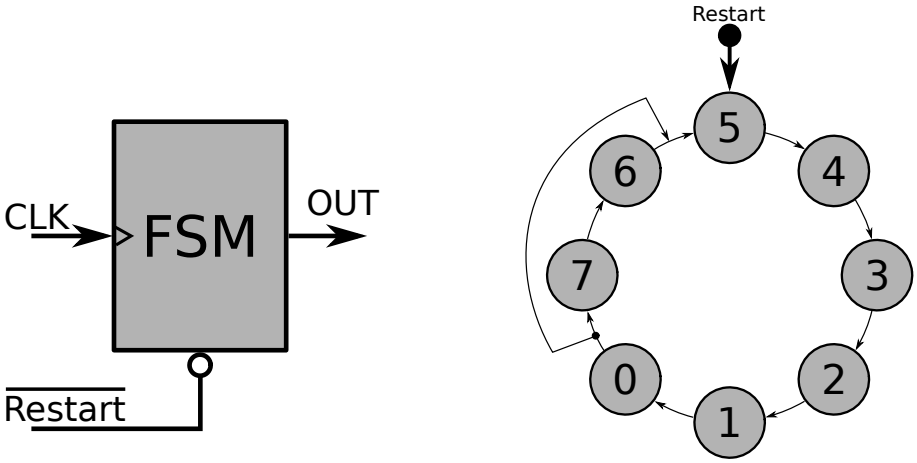
$$\begin{aligned}\overline{R} &= \overline{Restart}.\overline{R_{int}} \\ &= \overline{Restart}.\overline{(Q^3_{t-1}.Q^2_{t-1}.Q^1_{t-1}.Q^0_{t-1})}\end{aligned}$$

V tomto prípade nemá veľmi zmysel aplikovať De Morganovo pravidlo, nakoľko by sa síce z dvojvstupového hradla AND stalo hradlo NAND, čo by ušetrilo dva tranzistory, ale museli by sme pridať invertor pre signál $\overline{Restart}$ (+ 2T) a zo štvorvstupového hradla NAND by sa naopak stalo štvorvstupové hradlo AND. Na konci by sme teda paradoxne potrebovali o dva tranzistory viac ako pri pôvodnej funkcii. Schéma na Obr. 12.9 má označené uzly na základe nami opísaného postupu, všetky preklápacie obvody obsahujú asynchrónny reset, teda počiatočný stav bude číslo nula.



Obr. 12.9: Asynchrónne počítadlo od 0 do 14 smerom vpred

Úloha: Navrhnite asynchrónne počítadlo smerom vzad od 5 do 0 s krokom 1, pričom počiatočný stav bude dekadické čislo 5.
Riešenie: Je evidentné, že budeme potrebovať tri bity pamäte – Q^2 až Q^0. Pre počiatočné číslo $5_D = 101_B$ bude potrebné pre bity Q^2 a Q^0 použiť funkciu set a pre bit Q^1 použiť funckiu reset. Pre lepšiu orientáciu uvádzame v diagrame stavov na Obr. 12.10 všetky stavy asynchrónneho počítadla aj so zamýšlaným skrátením cyklu.



Obr. 12.10: Bloková schéma a stavový diagram asynchrónneho počítadla

Nasleduje tabuľka stavov a pravdivostná tabuľka pre zlúčenie reštartov – Tab. 12.5. Pri vypĺňaní tabuľky stavov smerom vzad je potrebné zvýšiť opatrnosť, oveľa ľahšie dôjde k nesprávnemu kroku. Pri dekrementovaní obsahu pamäte prídeme do stavu nula, po ktorom nasleduje skrátenie cyklu a návrat do východzej hodnoty. Ako je zrejmé z diagramu stavov, v tomto prípade musíme aktivovať interný reštart detekciou čísla sedem. Tabuľku stavov je pri takomto type príkladov lepšie čítať smerom zdola nahor, pričom začíname v riadku s číslom počiatočného stavu.

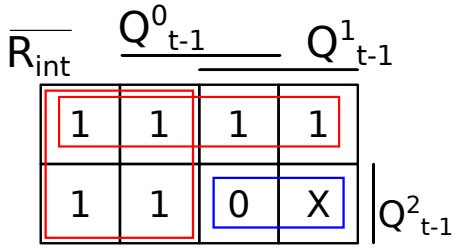
Tab. 12.5: Tabuľka stavov asynchrónneho počítadla smerom vzad

Q^2_{t-1}	Q^1_{t-1}	Q^0_{t-1}	Q^2_t	Q^1_t	Q^0_t	$\overline{R_{int}}$
0	0	0	1	0	1	1
0	0	1	0	0	0	1
0	1	0	0	0	1	1
0	1	1	0	1	0	1
1	0	0	0	1	1	1
1	0	1	1	0	0	1
1	1	0	X	X	X	X
1	1	1	X	X	X	0

Tab. 12.6: PT pre zlúčenie interného a užívateľského externého reštartu

$\overline{Restart}$	$\overline{R_{int}}$	$\overline{R}$
0	0	0
0	1	0
1	0	0
1	1	1

K-mapa na Obr. 12.11 je opäť jednoduchá a slučky v nej definujú MDNF = MKNF. Po aplikovaní De Morganovych pravidiel dostávame veľmi jednoduchú implementáciu interného reštartu pomocou dvojvstupového hradla NAND. Zlúčenie interného a vonkajšieho reštartu je opäť dosiahnuté vďaka hradlu AND. Uvedená aplikácia zlúčenia bude vždy realizovaná pomocou operácie AND, pokiaľ bude mať externý reštart vyššiu prioritu ako interný.



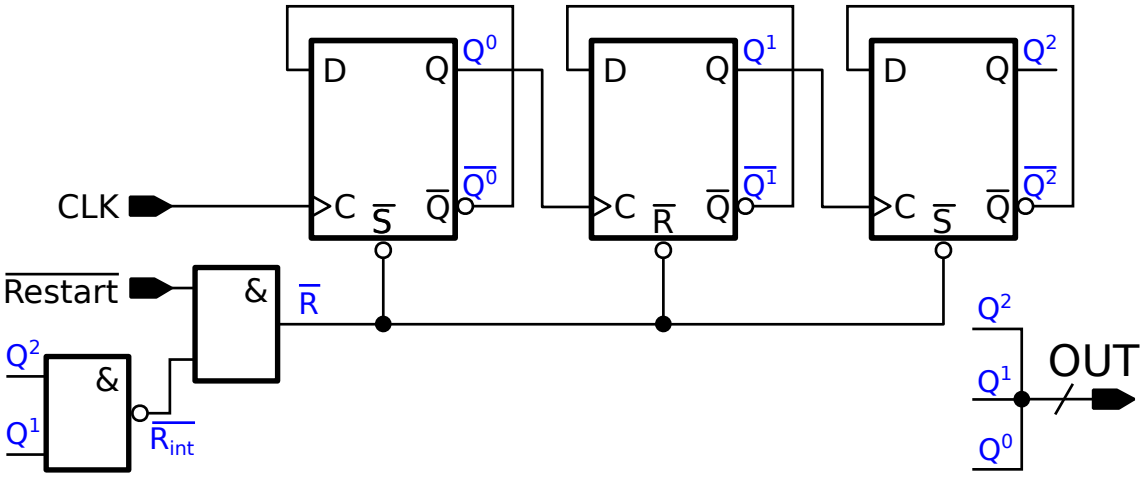
Obr. 12.11: K-mapa odvodenia interného reštartu pre skrátenie cyklu

$$\begin{aligned}\overline{R_{int}} &= \overline{Q^2_{t-1}} + \overline{Q^1_{t-1}} \\ &= \overline{Q^2_{t-1}.Q^1_{t-1}}\end{aligned}$$

$$\begin{aligned}\overline{R} &= \overline{Restart}.\overline{R_{int}} \\ &= \overline{Restart}.\overline{Q^2_{t-1}.Q^1_{t-1}}\end{aligned}$$

Schéma zapojenia na Obr. 12.12 je typovo identická s predchádzajúcou schémou s tým rozdielom, že budenie hodinového vstupu v kaskáde je tentokrát z neinvertujúceho výstupu predchádzajúceho preklápacieho obvodu. Takto dosiahneme počítanie vzad. Keď počítadlo prejde do stavu 0 a príde nasledujúca hrana hodín, počítadlo sa bude interne snažiť prejsť do stavu $7_D = 111_B$.

Interný reštart zložený z negovaného súčinu bitov Q_2 a Q_1 vygeneruje logickú nulu, ktorá nastaví počítadlo do počiatočného stavu $5_D = 101_B$. Čiže ako pri asynchrónnom počítadle smerom vpred, na zanedbateľný moment bude počítadlo v hazarde, ale nakoniec pristane v stabilnom počiatočnom stave. Všimnime si pre kontrolu, že prvý a posledný bit pamäte skutočne využívajú funkciu set pre požadovaný počiatočný stav.



Obr. 12.12: Schéma zapojenia navrhovaného asynchrónneho počítadla vzad

12.2 Príklady

Úloha 1: Navrhnite asynchrónne počítadlo smerom vpred od 0 do 11 s krokom 1. Zlúčte interný reštart s externým reštartom.

Úloha 2: Navrhnite asynchrónne počítadlo smerom vzad od 0 do 7 s krokom 1. Zlúčte interný reštart s externým reštartom.

Úloha 3: Navrhnite asynchrónne počítadlo smerom vpred od 2 do 14 s krokom 1. Zlúčte interný reštart s externým reštartom.

Úloha 4: Navrhnite asynchrónne počítadlo smerom vzad od 14 do 2 s krokom 1. Zlúčte interný reštart s externým reštartom.

Úloha 5: Navrhnite synchrónne počítadlo smerom vpred od 1 do 11 s krokom 1. Pre všetky bity pamäte použite preklápací obvod RS / JK / T / D.

Úloha 6: Navrhnite synchrónne počítadlo smerom vzad od 12 do 0 s krokom 1. Pre všetky bity pamäte použite preklápací obvod RS / JK / T / D.

Úloha 7: Navrhnite synchrónne počítadlo smerom vpred od 1 do 15 s krokom 2. Pre všetky bity pamäte použite preklápací obvod RS / JK / T / D.

Úloha 8: Navrhnite synchrónne počítadlo smerom vzad od 14 do 2 s krokom 2. Pre všetky bity pamäte použite preklápací obvod RS / JK / T / D.

Záver

Veríme, že po prečítaní týchto skrípt a prepočítaní uvedených príkladov by študenti predmetu „Logické systémy" nemali mať najmenšie problémy úspešne absolvovať písomky počas semestra a zároveň aj záverečnú skúšku. Obsah prednášok, ako aj príklady v tejto zbierke boli koncipované na základe skúseností vyučujúcich z vedecko-výskumnej praxe, ako aj z pedagogického procesu. Preto taktiež veríme, že sa študenti Fakulty elektrotechniky a informatiky, Slovenskej technickej univerzity v Bratislave stretnú s problematikou podobnou alebo priamo nadväzujúcou na obsah tohto predmetu, bez ohľadu na zameranie ich budúceho štúdia.

V prípade nejasností, návrhov na zlepšenie, alebo ak si myslíte, že ste našli v riešeniach príkladov chybu, určite to neváhajte prediskutovať so svojim cvičiacim alebo s prednášajúcim pedagógom.