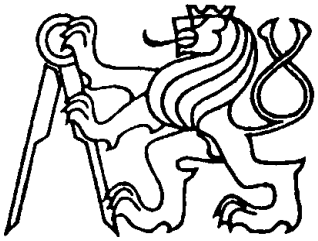
Spínací a číslicová technika

Jiří Podlešák, Petr Skalický



Praha 1994

Obsah

Úvod

Rozvoj mikroelektroniky, který začal okolo roku 1960, umožnil využití číslicové techniky nejen v oblasti výpočetní techniky, ale i v ostatních technických oborech, zvláště pak v radiotechnice, měřící a sdělovací technice. Přechod od analogového přenosu, zpracování a vyhodnocení informací k číslicovému je nejen způsoben zvýšenými nároky na parametry zařízení, ale i rozvojem vhodné technické základny pro jeho realizaci.

Základní vlastnosti analogového signálu je jen relativní přesnost, protože jeho zkreslení se projeví určitou ztrátou informace, kterou přenáší. Naproti tomu u číslicového (dvojkového) signálu, kdy přenášený signál je vyjádřen posloupností čísel, i při velkém zkreslení nedochází ke ztrátě informace, pokud se mohou spolehlivě rozlišit obě diskrétní úrovně signálu. Další výhodou při zpracování číslicových signálů je možnost využívat nejen postupy, které modelují chování reálných soustav, ale i metody ryze matematické. S rozvojem součástkové základny pro číslicovou techniku vznikla potřeba teoretických prostředků umožňující popis, analýzu a návrh logických a číslicových obvodů, z kterých se vytváří číslicové soustavy - rychlé násobičky, řadiče, číslicové filtry, specializované procesory, analyzátory spektra, atd.

Skriptum, jehož posláním je seznámit s nejdůležitějšími prostředky pro analýzu a návrh číslicových a logických obvodů, je rozděleno do šesti částí, z nichž první pojednává o číselných soustavách a převodech mezi nimi, o vyjádření racionálních čísel v pevné a pohyblivé desetinné čárce a o základních aritmetických operacích s těmito čísly. Druhá část je věnována po krátkém úvodu do Booleovy algebry a zjednodušování logických funkcí metodám návrhu kombinačních obvodů z logických členů NAND, NOR, AND-OR-INVERT a z obvodů se střední hustotou integrace jako jsou multiplexery, dekodéry, paměti a programovatelná pole PAL, GAL, atd. Po metodách návrhu logických kombinačních obvodů se v další kapitole popisují obvody realizující paralelní aritmetické operace porovnání, sčítání, odčítání, násobení a obvody kódovací pro zabezpečení přenosu informace. Další kapitoly jsou věnovány logickým sekvenčním obvodům. Po úvodním přehledu prostředků používaných k popisu sekvenčních obvodů je pozornost věnována analýze asynchronních a synchronních obvodů. Po analýze sekvenčních obvodů následuje popis jednotlivých fází návrhu asynchronního, impulsního a synchronního obvodu, na který navazuje několik ukázek návrhu obvodu. Část pojednávající o aplikacích sekvenčních obvodů přináší příklady na použití funkce pamatování, posouvání a čítání a je zakončena realizacemi kódovacích obvodů a řídicích jednotek. Poslední kapitola je věnována architekturám programovatelných logických obvodů a metodám jejich použití k realizaci logických obvodů.

1. Číselné soustavy

Pracujeme-li s čísly, používáme přirozeně symboly, jimž přiřazujeme deset různých hodnot od nuly (symbol 0) do devíti (symbol 9). Používáme to, co matematici nazývají soustavou zobrazení čísel se základem deset. První stroje na zpracování čísel byly mechanické a nebylo u nich složité zobrazovat čísla se základem deset pomocí mechanické součástky, která měla deset stabilních stavů, z nichž každý odpovídal jednomu symbolu soustavy se základem deset. Velký pokrok v oblasti číslicových výpočtů nastal zavedením elektronek, tranzistorů, integrovaných obvodů a mikroprocesorů ať již universálních nebo speciálních. Nové stroje jsou od dřívějších odlišné nejen strukturou, výkonem, spolehlivostí, ale hlavně použitím dvojkové soustavy zobrazení čísel a veličin. Právě realizace elektronického obvodu se dvěma stabilními stavy je jednoduchá a umožňuje další vývoj.

Každé celé číslo N lze obecně vyjádřit v soustavě se základem B pomocí symbolů neboli číslic a_i takto

$$N_B = a_n B^n + a_{n-1} B^{n-1} + \ldots + a_1 B^1 + a_0 B^0 \qquad (1.1)$$

V praxi se základ nezapisuje a spokojujeme se vypsáním koeficientů a vedle sebe

$$N_B = a_n a_{n-1} a_{n-2} \ldots a_1 a_0 \qquad (1.2)$$

Čísla 1302_{10} v desítkové soustavě a 101101_2 v soustavě dvojkové odpovídají těmto součtům součinů

$$1.10^3 + 3.10^2 + 0.10^1 + 2.10^0 = 1302_{10} \qquad (1.3)$$

$$1.2^5 + 0.2^4 + 1.2^3 + 1.2^2 + 0.2^1 + 1.2^0 = 45_{10} \qquad (1.4)$$

Ve dvojkové soustavě se počítá stejným způsobem jako v soustavě desítkové. Číslice se zvětšuje o 1, když všechny číslice u nižších vah jsou 1 (základ 2) nebo 9 (základ 10).V logických systémech se pro práci s čísly používá čtyř číselných základů:

a) Základ 2 (binární) se symboly 0 a 1 (někdy O a I)

b) Základ 8 (oktalový) se symboly 0,1,2,3,4,5,6,7 - nyní se již nepoužívá

c) Základ 10 (dekadický) se symboly 0,1,2,3,4,5,6,7,8,9

d) Základ 16 - (hexadecimální) se symboly 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F. Písmena A,B,C,D,E,F mají po řadě váhy 10,11,12, 13,14,15 v základu 10. Hexadecimální vyjádření se v současné době nejčastěji používá k vyjadřování obsahu pamětí mikropočítačů.

1.1. Princip změny základu

Metoda postupného odčítání

Metodu lze snadno použít k přechodu od základu B k základu B_1. Původní číslo se rozkládá odečítáním zmenšujících se mocnin základu B_1, kdy je hledána mocnina čísla B_1 rovná nebo menší než zbývající část převáděného čísla.

Příklad 1.1 Převeďte $190_{10} \rightarrow N_2$.

Mocnina	Rozdíl	Výsledek	Mocnina	Rozdíl	Výsledek
$2^7 = 128$	190 - 128 = 62	1	$2^3 = 8$	14 - 8 = 6	1
$2^6 = 64$	62 - 64 = -2	0	$2^2 = 4$	6 - 4 = 2	1
$2^5 = 32$	62 - 32 = 30	1	$2^1 = 2$	2 - 2 = 0	1
$2^4 = 16$	30 - 16 = 14	1	$2^0 = 1$	0 - 1 = -1	0

Odtud $190_{10} = 10111110_2$

Metoda postupného dělení

Metoda postupného dělení základem vychází z následujícího odvození, při kterém předpokládáme, že číslo N v základu B chceme vyjádřit v základu B_1.Vyjádříme nejprve číslo N v základu B_1 takto

$$N_{B_1} = a_n B_1^n + a_{n-1} B_1^{n-1} + \ldots + a_1 B_1^1 + a_0 B_1^0 \qquad (1.5)$$

Jestliže je nyní budeme dělit základem B_1, dostaneme podíl P_1 a zbytek Z_1, který představuje koeficient a_0 čísla N v základu B_1.

$$N_{B_1} = P_1 . B_1 + Z_1 = \left[a_n B_1^{n-1} + a_{n-1} B_1^{n-2} + \ldots + a_1 B_1^0 \right] . B_1 + a_0 \qquad (1.6)$$

Vydělíme-li podíl P_1 základem B_1 můžeme psát tento vztah

$$P_1 = P_2 . B_1 + Z_2 = \left[a_n B_1^{n-2} + a_{n-1} B_1^{n-3} + \ldots + a_2 B_1^0 \right] . B_1 + a_1 \qquad (1.7)$$

Zbytek Z_2 představuje koeficient a_1. Dalším dělením získáváme postupně koeficienty hledaného čísla v pořadí od nejnižší k nejvyšší váze. Dělit se může v libovolném základě, pokud můžeme v příslušném základě snadno počítat.

Příklad 1.2 Vyjádřete číslo 1358_{10} v základě 16.

Dílčí podíl	1358:16=84	84:16=5	5:16=0
Zbytek	14	4	5

Odtud je číslo $1358_{10} = 54E_{16.}$

Příklad 1.3 Číslo 1357_8 vyjádřete v základě 16

Osmičkové číslo můžeme nejprve převést na číslo desítkové pomocí sčítání mocnin a pak dělit v základu 10 ($1357_8 = 1.8^3 + 3.8^2 + 5.8^1 + 7 = 751_{10}$).

Dílčí podíl	751:16=46	46:16=2	2:16=0
Zbytek	15	14	2

Můžeme rovněž dělit číslo 1357_8 šestnácti, převedenými do základu 8 (dělení prováděné v osmičkové soustavě). V osmičkové soustavě platí $2_8.5_8 = 12_8$, $2_8.6_8 = 14_8$, $17_8 = 15_{10} = F_{16}$, $16_8 = 14_{10} = E_{16}$ a $2_8 = 2_{10} = 2_{16}$

Dílčí podíl	1357:20=56	56:20=2	2:20=0
Zbytek	17	16	2

Odtud $1357_8 = 2EF_{16}$.

Převod můžeme provést pomocí dvojkového vyjádření (viz dále). Osmičkové číslo převedeme do dvojkové soustavy a rozdělíme na čtveřice, které představují dvojkově vyjádřené hexadecimální znaky. Odtud $1357_8 = 2EF_{16}$.

1	3	5	7
0 0 1	0 1 1	1 0 1	1 1 1
2	E		F

Metoda postupného násobení

Každé reálné číslo $N < 1$ lze obecně vyjádřit v základu B pomocí symbolů b_i takto

$$N_B = b_{-1}B^{-1} + b_{-2}B^{-2} + \ldots + b_{-n+1}B^{-n+1} + b_nB^{-n} \qquad (1.8)$$

Číslo N pak můžeme zapsat

$$N = 0, b_{-1}b_{-2}b_{-3}\ldots b_{-n} \qquad (1.9)$$

kde b_{-n} je koeficient s nejmenší váhou. To znamená, že číslo je určeno s přesností B^{-n} ($\left|N - 0, b_{-1}b_{-2}b_{-3}\ldots b_{-n}\right| < B^{-n}$). Převod čísla $N < 1$ lze provést obdobně jako u čísel přirozených metodou postupného odečítání základu nebo metodou postupného násobení, při kterém postupným násobením čísla N a jeho zbytků základem B získáváme jednotlivé koeficienty čísla N v základu B.

$$N.B = b_{-1} + b_{-2}B^{-1} + \ldots + b_{-n+1}B^{-n+2} + b_{-n}B^{-n+1} = b_{-1} + Z_1 \qquad (1.10)$$

$$Z_1.B = b_{-2} + b_{-3}B^{-1} + \ldots + b_{-n+1}B^{-n+3} + b_{-n}B^{-n+2} = b_{-2} + Z_2 \qquad (1.11)$$

Koeficienty b_{-i} hledaného čísla N získáváme v pořadí jak jdou za sebou jako celou část součinu N.B pro i = 1 a $Z_i.B$ pro $i \geq 2$.

Příklad 1.4 Převeďte číslo $0,725_{10}$ do osmičkové a dvojkové soustavy.

0,725	x 8 =	$5,8 \rightarrow 5$	0,725	x 2 =	$1,45 \rightarrow 1$
0,8	x 8 =	$6,4 \rightarrow 6$	0,45	x 2 =	$0,90 \rightarrow 0$
0,4	x 8 =	$3,2 \rightarrow 3$	0,9	x 2 =	$1,80 \rightarrow 1$
0,2	x 8 =	$1,6 \rightarrow 1$	0,8	x 2 =	$1,60 \rightarrow 1$
			0,6	x 2 =	$1,20 \rightarrow 1$
			0,2	x 2 =	$0,40 \rightarrow 0$

Odtud $0,725_{10} = 0,5631...._8 = 0,101110011001...._2$

Zobrazení dvojkové informace

Mějme číslo vyjádřené dvojkově a hledejme jeho osmičkový ekvivalent pomocí postupného dělení osmi

$$N_2 = a_n 2^n + a_{n-1} 2^{n-1} + \ldots + a_1 2^1 + a_0 2^0 =$$
$$= 8.\left[a_n 2^{n-3} + a_{n-1} 2^{n-4} + \ldots + a_3\right] + a_2 2^2 + a_1 2^1 + a_0 \qquad (1.12)$$

kde $Z_1 = a_2 2^2 + a_1 2^1 + a_0$. Prvek s nejmenší váhou v základu 8 se rovná váhovanému součtu tří prvků s nejmenší váhou ve dvojkové soustavě. Pokud znovu dělíme získaný podíl P_1 osmi získáme druhý zbytek Z_2, který bude tvořen další váhovanou trojicí dvojkových prvků $Z_2 = a_5 2^2 + a_4 2^1 + a_3$. Z odvození vyplývá, že převod dvojkového čísla do osmičkové soustavy je velmi jednoduchý. Dvojkové číslo se rozdělí do trojic bitů, které se pomocí jednoduchého váhového součtu převedou do osmičkové soustavy např. takto

111	010	000	101	100
7	2	0	5	4

Odtud N = $111010000101100 = 72054_8$.

Při dvojkově hexadecimálním převodu se rozdělí dvojkové číslo do čtveřic a pak se čtveřice vyjádří pomocí váhovaného součtu symbolem z šestnáctkové soustavy.

1010	0100	0111		E	1	9
A	4	7		1110	0001	1001

Odtud $E19_{16} = 111000011001_2$ a $101001000111_2 = A47_{16}$.

Přestože hexadecimální vyjádření je vhodně zhuštěným záznamem dvojkové informace, potřebujeme často zobrazit vypočítanou veličinu v desítkové soustavě. Vlastní převod do desít-

kové soustavy se provádí v závislosti na způsobu realizace metodami uvedenými v předcházející části nebo pomocí dekadické korekce do tzv. BCD kódu. Každá desítková číslice je potom vyjádřena čtyřmi bity, které váhovaně určují její hodnotu. BCD kódů je velmi mnoho, nejčastěji se setkáváme s kódem BCD 8421, kde označení 8421 určuje váhy jednotlivých bitů (normální binární vyjádření čísel od 0 do 9). Protože šest kombinací 10 až 15 je v tomto vyjádření nevyužito, zabírá desítkové číslo dvojkově zapsané větší počet míst, než jeho dvojkový ekvivalent.

0110	0001	1001
6	1	9

$$619_{10} = 011000011001_{BCD8421}$$

1.2. Vyjádření kladných a záporných čísel

Doposud jsme hovořili pouze o vyjádření kladných celých i necelých číslech. Potřebujeme-li zobrazovat čísla se znaménkem je zřejmé, že počet bitů rezervovaný na rozsah čísel bude nutno zvětšit o jeden bit, který ponese informaci o znaménku. Čtyři nejpoužívanější vyjádření reálných čísel v pevné desetinné čárce nyní popíšeme.

Vyjádření ± absolutní hodnota.

Při tomto vyjádření představuje nejvyšší bit znaménko (obvykle 0 = +, 1 = -) a zbývajících n-1 bitů určuje absolutní hodnotu zobrazovaného čísla. Máme-li k dispozici n bitů, pak můžeme zobrazovat kladná i záporná čísla v intervalu <0, 2^{n-1} -1>. Ve vyjádření existují dvě nuly, z nichž použití záporné nuly 10000 ... 00 se většinou zakazuje.

Vyjádření záporných čísel jednotkovým doplňkem

U tohoto vyjádření jsou kladná čísla vyjádřena stejně jako ve tvaru ± absolutní hodnota. Záporná čísla jsou v jednotkovém doplňku, který označujeme ^1A a vypočteme z následujícího vztahu

$$^1A = 2^n - 1 - A = 2^n - 1 - \sum_{i=0}^{n-2} a_i 2^i \qquad (1.13)$$

Hodnota $2^n - 1$ představuje číslo, které má na všech n bitech samé jedničky. Odečteme-li od této hodnoty n-1 bitové číslo A, získáme číslo mající oproti číslu A všechny bity negované. Máme-li k dispozici n bitů můžeme zobrazovat čísla kladná i záporná v intervalu < 0, 2^{n-1}-1>. Nejvyšší bit značí znaménko (0 = +, 1 = -). Ve vyjádření existuje kladná (000 .. 0) i záporná (111 .. 1) nula.

Vyjádření záporných čísel dvojkovým doplňkem

U tohoto vyjádření jsou kladná čísla vyjádřena stejně jako ve tvaru ± absolutní hodnota. Záporná čísla jsou ve dvojkovém doplňku, který označujeme ^{2}A a vypočteme z následujícího vztahu

$$^{2}A = 2^{n} - A = 1 + {}^{1}A = 2^{n} - \sum_{i=0}^{n-2} a_i 2^{i} \qquad (1.14)$$

Jak vyplývá z předešlého textu získáme dvojkový doplněk tak, že všechny bity čísla A znegujeme (jednotkový doplněk) a pak k němu přičteme jedničku na nejnižší pozici. Máme-li k dispozici n bitů můžeme zobrazovat kladná čísla v intervalu $< 0,2^{n-1}-1>$ a záporná čísla v intervalu. $< -1,-2^{n-1}>$. Nejvyšší bit opět značí znaménko (0 = +, 1 = -). Ve vyjádření existuje již jen jedna nula (000 .. 0).

Vyjádření +1/2 intervalu

Číslo v tomto vyjádření získáme ze vztahu

$$^{1/2}A = 2^{n-1} + A = 2^{n-1} + \sum_{i=0}^{n-2} a_i 2^{i} \qquad (1.15)$$

kde n je počet bitů čísla. Analýzou vztahu (1.15) zjistíme, že při n platných bitech můžeme zobrazovat kladná čísla v intervalu $< 0,2^{n-1}-1>$ a záporná čísla v intervalu. $< -1,-2^{n-1}>$. Bit s nejvyšší váhou opět značí znaménko s tím, že 0 = -, 1 = +. Stejně jako v případě dvojkového doplňku má vyjádření jen jednu nulu (100 .. 0). V tabulce 1.1 jsou uvedeny příklady vyjádření některých racionálních čísel v pevné desetinné čárce

Číslo	bin.hodnota	± | A |	^{1}A	^{2}A	$^{1/2}A$
+10	+1010	01010	01010	01010	11010
+0	+0000	00000	00000	00000	10000
-0	-0000	10000	11111	-------	-------
-2	-0010	10010	11101	11110	01110
-14	-1110	11110	10001	10010	00010
-3,5	-0011,10	10011,10	11100,01	11100,10	01100,10
-0,5	-0000,10	10000,10	11111,01	11111,10	01111,10
-16	nelze vyjadřit	nelze vyjádřit	nelze vyjádřit	10000	00000

Tabulka 1.1

1.3. Vyjádření čísel s pohyblivou desetinnou čárkou

Chceme-li pracovat s čísly v pohyblivé desetinné čárce, musíme je rozšířit o číslo vyjadřující hodnotu exponentu včetně jeho znaménka. Číslo s pohyblivou čárkou pak bývá vyjádřeno ve tvaru

$$A = 0$$

$$A = m.B^{e} \quad \text{nebo} \quad A = (m.B).B^{e-1} \qquad (1.16)$$

kde m je mantisa (část za desetinnou čárkou), B je základ číselné soustavy a e je hodnota exponentu. Pro ilustraci číslo 6725430, pak bude vyjádřeno buď jako $0{,}672543.10^{7}$ nebo $6{,}725430.10^{6}$. Pro dvojkovou soustavu, kterou se budeme zabývat, můžeme ve shodě s vyjádřením (1.16) psát

$$A = 0$$

$$A = znaménko\left(\begin{matrix}1\\0\end{matrix} + mantisa\right).2^{znaménko\ exponent} \qquad (1.17)$$

Mantisa představuje hodnotu na desetinných místech a můžeme ji v souladu s předcházejícím textem vyjádřit výrazem

$$mantisa = a_{-1}B^{-1} + a_{-2}B^{-2} + \ldots + a_{-n+1}B^{-n+1} + a_{n}B^{-n} \qquad (1.18)$$

Závorka v rovnici (1.17) může nabývat hodnoty v intervalu <1,0;2,0) nebo <0,5;1,0) podle toho, je-li k mantise připočítávána hodnota 1,0 či nikoliv. Obě vyjádření se od sebe liší o jedničku v hodnotě exponentu. Přesnost čísla na desetinných místech je dána hodnotou 2^{-n}. Hodnotu exponentu vyjádříme celým číslem se znaménkem

$$znaménko\ exponent = z_e a_{m-2} a_{m-3} \ldots a_0 \qquad (1.19)$$

Příklad 1.5 Určete počet bitů nezbytný k vyjádření čísel v pohyblivé desetinné čárce s přesností tří desetinných míst a exponentem v rozsahu $10^{\pm 19}$.

Aby číslo v pohyblivé čárce mělo přesnost tří desetinných míst, musí hodnota n splňovat vztah

$$2^{-n} \le 10^{-3} \qquad n \ge 3 \big/ \log 2 \ge 9{,}92 \qquad (1.20)$$

Odtud n = 10. Nezbytný počet bitů m k vyjádření exponentu určíme ze vztahu

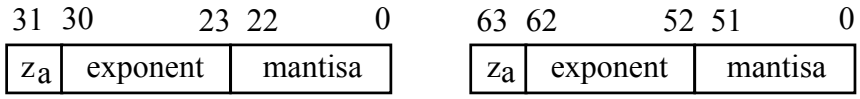
$$2^{2^{m-1}} \ge 10^{19} \qquad m \ge 1 + \frac{\log\left(19 \big/ \log 2\right)}{\log 2} \ge 6{,}97 \qquad (1.21)$$

Odtud m = 7. Celé číslo A bude vyjádřeno včetně znaménka 18 bity - symboly, které jsou zobrazeny vztahem (1.22).

z_a	$a_{-1}a_{-2}a_{-3}a_{-4}a_{-5}a_{-6}a_{-7}a_{-8}a_{-9}a_{-10}$	$z_e a_5 a_4 a_3 a_2 a_1 a_0$
znaménko	mantisa	exponent

(1.22)

Standard pro vyjádření čísel v pohyblivé desetinné čárce v jednoduché i dvojnásobné přesnosti je dána předpisem IEEE 754-1985. Čísla s jednoduchou přesností jsou vyjádřena 32 bity obr.1.1 ve tvaru $(-1)^z.(1+m).2^{e+127}$, kde bity z a m představují číslo ve vyjádření ±absolutní hodnota a exponent e ∈<-126;127> je ve vyjádření +1/2 intervalu zmenšený o hodnotu jedna. Krajní hodnoty exponentu e+127 = 0 a e+127 = 255 se využívají k vyjádření čísel, která nelze ve formátu daném rovnicí (1.17) zobrazit. Při dvojnásobné přesnosti jsou čísla vyjádřena stejně jako při přesnosti jednoduché a tím, že k exponentu je příčítána hodnota 1023 (e ∈ <-



Obr.1.1

1022;1023>), jak vyplývá z obr.1.1. Oba formáty ve svém instrukčním souboru podporuje např. signálový procesor s pohyblivou čárkou MOTOROLA DSP 96002 s tím, že je využito tzv. skrytého bitu. To znamená, že v bitovém vyjádření výrazu (1+m) nenalezneme právě tu jedničku. V aritmetických knihovnách, které využíváme někdy v programech psaných v jazyce symbolických adres, se setkáváme s vyjádřením čísel, které se více či méně blíží tomuto standardu.

Číslo		Vyjádření		
		z	mantisa	exponent
$1{,}000.10_0$	$1{,}000000.2_0$	0	0000000000	1000000
$8{,}000.10_0$	$1{,}000000.2_3$	0	0000000000	1000011
$0{,}000.10_0$	$0{,}000000.2_0$	0	0000000000	0000000
$-1{,}750.10_0$	$1{,}750000.2_0$	1	1100000000	1000000
$1{,}250.10_2$	$1{,}953125.2_6$	0	1111010000	1000110
$-1{,}250.10_2$	$-1{,}953125.2_6$	1	1111010000	1000110
$-7{,}812.10_{-2}$	$-1{,}249920.2_{-4}$	1	0011111111	0111100

Tabulka 1.2

Například v aritmetikách INTEL pro µP 8080 nebo FL48 a FL51 - TESLA IMA pro µP8048 a 8051 je využíváno vyjádření (1+m) ve tvaru ± absolutní hodnota se skrytým bitem (skrytou jedničkou) a exponentem je vyjádření +1/2 intervalu viz tab.1.2. Dojde-li při aritmetické operaci k přetečení nebo podtečení potom je situace indikována příznakem CY (8080) nebo F0

(8048,8051) a nikoliv maximální nebo minimální hodnotou exponentu. Nula je charakterizována nulovým exponentem a záporná nula je zakázána. Naproti tomu u signálových procesorů s pohyblivou desetinnou čárkou Texas Instruments TMS320C3x a TMS320C4x, jejichž aritmeticko-logické jednotky podporují operace se zápornými čísly vyjádřené dvojkovým doplňkem, mají vyjádřenu hodnotu (1+m) i exponent ve vyjádření dvojkovým doplňkem.

Příklad k samostatnému řešení

Příklad 1.3.1 Odvoďte čísla v pohyblivé čárce z tabulky 1.2 za předpokladu, že závorka je ve tvaru (0+mantisa) i exponent jsou ve vyjádření záporných čísel dvojkovým doplňkem (aritmetika FLTGPT20 - Texas Instruments pro signálový procesor TMS320C20).

1.4. Základní operace s čísly v pevné desetinné čárce

Existuje pět aritmetických operací s dvojkovými čísly v pevné desetinné čárce, které tvoří základ operací s čísly v pohyblivé desetinné čárce. Jedná se o porovnávání, sčítání, odčítání, násobení a dělení. Tyto operace lze realizovat kombinačními a sekvenčními logickými obvody včetně mikroprocesorů. U současných výkonných mikroprocesorů jsou tyto operace realizovány jedinou instrukcí, nalezneme však i mikroprocesory u nichž je zastoupeno pouze sčítání (např. MHB8048).

Sčítání

Sčítání ve dvojkové soustavě je nejdůležitější aritmetickou operací, protože tvoří základ pro zbývající aritmetické operace tj. odčítání, násobení a dělení. Sčítání ve dvojkové soustavě se provádí podle stejného algoritmu jako sčítání v desítkové soustavě. Obecně lze součet čísel A a B zapsat takto

$$S = A + B \qquad (1.23)$$

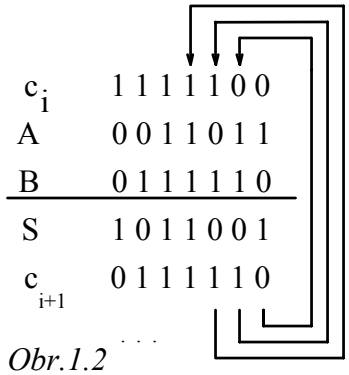
nebo

$$\sum_{i=0}^{n} s_i.2^i = \sum_{i=0}^{n-1} a_i.2^i + \sum_{i=0}^{n-1} b_i.2^i \qquad (1.24)$$

kde

$$a_i.2^i + b_i.2^i + c_i.2^i = s_i.2^i + c_{i+1}.2^{i+1} \qquad (1.25)$$

Rovnice (1.25) popisuje operaci, kterou musíme realizovat v jednom binárním řádu.



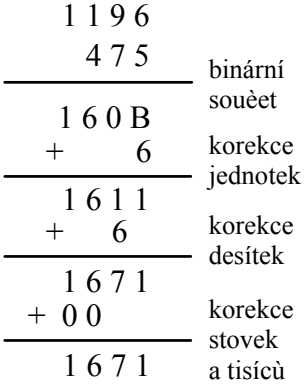
Obr.1.2

Příklad 1.6 Sečtete čísla 27_{10} a 62_{10} ve dvojkové soustavě.

Nejprve převedeme sčítaná čísla do dvojkové soustavy a zjistíme, že platí $A=27_{10} = 0011011_2$ a $B = 62_{10} = 0111110_2$. Z obr.1.2 vyplývá, že $S = 1011001_2 = 89_{10.}$ Rovnice (1.25) popisuje součet dvojkových čísel v jednom řádu včetně přenosu do vyššího řádu c_{i+1}. Logický kombinační obvod, který realizuje rovnici (1.25), se nazývá **úplná binární sčítačka**.

Sčítání čísel v BCD kódu

Existují realizace, u kterých potřebujeme naměřenou hodnotu v BCD kódu korigovat o nějakou konstantní hodnotu. Jedná-li se o součet dvou čísel v BCD kódu je zbytečné převádět čísla na binární, sečíst je a potom konvertovat na číslo desítkové. Mnohem výhodnější je provést součet obou BCD čísel pomocí binární sčítačky a potom získaný součet korigovat pomocí tzv. **dekadické korekce**, která převede výsledek opět do kódu BCD. Dekadická korekce se provádí na čtyřech bitech vyjadřujících jeden dekadický řád (jednu dekadickou číslici) a koriguje rozdíl mezi modulem těchto 4 bitů (tj. modulem 16) a požadovaným modulem 10 v desítkové soustavě. Jestliže součet dvou BCD čísel v jednom řádu je menší jak 9, korekce není třeba, je-li větší jak 9 nebo byl při součtu překročen modul 16, potom je třeba korigovat rozdíl mezi oběma moduly tj. přičíst hodnotu 6. Tuto operaci je nutné uplatnit na všechny řády počínaje nejnižším. Na obr.1.3 je zobrazen případ součtu čísel v BCD kódu 1196 a 475. Binární součet uskutečníme v 16-bitové binární sčítačce, protože každá desítková číslice je vyjádřena čtyřmi bity. Dekadickou korekci musíme postupně aplikovat na všechny dekády počínaje jednotkami. Protože cifra po součtu jednotek je větší jak 9, přičteme k jednotkám hodnotu 6. V řádu desítek došlo k přenosu do řádu stovek (došlo k překročení modulu 256) a proto přičteme hodnotu 6 do řádu desítek. V řádu stovek a tisíců nepřičítáme nic, protože nedošlo ani k přenosu do vyššího řádu ani v součtu není číslice větší než 9.



Obr.1.3

Odčítání

Analogicky jako pro sčítání je možné odvodit vztah (1.26) popisující rozdíl dvou čísel v jednom řádu, který by dával základ pro realizaci úplné binární odčítačky. Ukazuje se však, že existují jiná řešení problému, která nevyžadují realizaci nového obvodu - odčítačky.

$$a_i.2^i - b_i.2^i - c_i.2^i = r_i.2^i - c_{i+1}.2^{i+1} \qquad (1.26)$$

Odčítání s jednotkovým doplňkem

V kapitole 1.2 bylo ukázáno, že jednotkový doplněk čísla je dán vztahem ${}^1\mathrm{B} = 2^n - 1 - \mathrm{B}$. Provedením součtu čísel $\mathrm{A}+{}^1\mathrm{B} = \mathrm{A} = \mathrm{B} + 2^n - 1$ získáváme požadovaný rozdíl čísel A - B k němuž je přičtena hodnota $2^n - 1$, kde 2^n je číslo mající jedničku na bitu vlevo od znaménkového bitu. Určeme nyní jak nám tato hodnota ovlivní požadovaný rozdíl v případech A > B a A ≤ B. Pro jednotlivé případy můžeme psát

pro A > B

$A + {}^1B > 2^n - 1$

$A - B = A + {}^1B + 1 - 2^n$

K dosažení správné hodnoty rozdílu A-B potřebujeme přičíst chybějící jedničku a odečíst hodnotu 2^n, čehož dosáhneme tzv. **kruhovým přenosem**

pro A ≤ B

$A + {}^1B \le 2^n - 1$

$A - B = 2^n - 1 + A - B =$

$= 2^n - 1 - R = {}^1R$

Je-li rozdíl záporný, pak získáme přímo jeho jednotkový doplněk

Rozdíl A - B

$A_{10} = 11 \qquad A = 001011$

$B_{10} = 5 \qquad {}^1B = 111010$

A	0 0 1 0 1 1	
^{1}B	1 1 1 0 1 0	
	1 0 0 0 1 0 1	
kruhový přenos	→ 1	
	1 0 0 0 1 1 0	

Výsledek kladný

A - B = 000110 = 6_{10}

Rozdíl B - A

$A_{10} = 11 \qquad {}^1A = 110100$

$B_{10} = 5 \qquad B = 000101$

B	0 0 0 1 0 1
^{1}A	1 1 0 1 0 0
	0 1 1 1 0 0 1

Výsledek záporný

A - B = 111001 = ¹rozdílu = -6_{10}

Odčítání s dvojkovým doplňkem

Při odčítání pomocí dvojkového doplňku je situace obdobná jako u jednotkového doplňku. Rozdíl čísel A - B za pomoci dvojkového doplňku čísla B realizujeme opět pomocí součtu $A + {}^2B$. Tím získáváme požadovaný rozdíl A - B zvětšený o hodnotu 2^n. Pro jednotlivé případy můžeme psát

pro A > B

$A+{}^2B > 2^n$

$A-B = A+{}^2B-2^n$

K dosažení správné hodnoty rozdílu A-B potřebujeme odečíst hodnotu 2^n. Protože se jedná o 1 na bitu vlevo od znaménka do dalšího zpracování ji neuvažujeme.

pro A ≤ B

$A+{}^2B \le 2^n$

$A-B = 2^n + A - B =$

$= 2^n - R = {}^2R$

Je-li rozdíl záporný, pak získáme přímo jeho dvojkový doplněk

Rozdíl A - B

$A_{10} = 21 \quad A = 010101$

$B_{10} = 13 \quad {}^2B = 110011$

A	0 1 0 1 0 1
^{2}B	1 1 0 0 1 1
	1 0 0 1 0 0 0

Výsledek kladný

A - B = 001000 = 8_{10}

Rozdíl B - A

$A_{10} = 21 \quad {}^2A = 101011$

$B_{10} = 13 \quad B = 001101$

B	0 0 1 1 0 1
^{2}A	1 0 1 0 1 1
	0 1 1 1 0 0 0

Výsledek záporný

A - B = 111000 = ^{2}rozdílu = -8_{10}

Rozdíl čísel v BCD kódu

Rozdíl čísel v kódu BCD je možné provádět dvěma způsoby. Při programovém řešení je na některých procesorech (Z80) možné postupovat stejně jako při sčítání. Většina mikroprocesorů však neumožňuje aplikovat dekadickou korekci po rozdílu čísel a proto je nutné řešit problém jinak. Pomoci si zde můžeme stejně jako u čísel binárních pomocí tzv. dekadického doplňku (dvojkového) daného výrazem ${}^D\mathrm{B} = 10^n - \mathrm{B}$.

Rozdíl A-B

A	0 0 4 5
^{D}B	9 9 8 4
	9 9 C 9
dekadická korekce	+ 6
	1 0 0 2 9

výsledek kladný

Rozdíl B-A

^{D}A	9 9 5 5
B	0 0 1 6
	9 9 6 B
dekadická korekce	+ 6
	0 9 9 7 1

výsledek záporný

Obr.1.4

V případě, že A>B potom součet $A+{}^D\mathrm{B} \ge 10^n$. Odtud

$$A-B = A+{}^DB-10^n \qquad (1.27)$$

K dosažení správné hodnoty rozdílu A-B potřebujeme odečíst hodnotu 10^n. Protože se jedná o jedničku na bitu vlevo od nejvyššího bitu v dalším zpracování ji neuvažujeme. V případě, že $A \le B$ potom $A+^{D}B < 10^n$ a pro rozdíl můžeme psát

$$A - B = 10^n + A - B = 10^n - R = ^{D}R \qquad (1.28)$$

Je-li rozdíl záporný, pak získáváme přímo jeho dekadický doplněk. Jako příklad si ukážeme oba případy při rozdílu dvou BCD čísel A=45 a B=16 obr.1.4. Protože je rozdíl BCD čísel realizován jako součet čísla s dekadickým doplňkem, může být na něj uplatněna dekadická korekce. Dvojkový dekadický doplněk ^{D}B čísla získáme pomocí jednotkového dekadického doplňku, ke kterému přičteme jedničku a provedeme dekadickou korekci.

Násobení

Násobení ve dvojkové soustavě se provádí podle stejného algoritmu jako násobení v soustavě desítkové. Nejprve aritmeticky vynásobíme každým bitem čísla B celé číslo A a získáme tak jednotlivé mezivýsledky obr.1.5. Ty potom aritmeticky sečteme. Porovnáním tabulek pro aritmetický součin dvou proměnných ve dvouhodnotové logice s tabulkou pro logický součin zjistíme, že jsou shodné. Díky tomu získáme dílčí mezivýsledky $b_j * (a_n a_{n-1} .... a_1 a_0)$ (součiny) jako logické součiny bitu b_j se všemi bity a_i (bude-li $b_j = 1$, pak mezivýsledek bude $a_n a_{n-1} .... a_1 a_0$, bude-li $b_j = 0$ potom bude mezivýsledek nulový).

```
A       1 1 1 0 1
xB          1 0 1
      -------------
        1 1 1 0 1
      0 0 0 0 0
    1 1 1 0 1
  -------------------
  1 0 0 1 0 0 0 1
```

A x B = $10001001_2 = 145_{10}$

Obr.1.5

Násobení čísel se znaménkem

Násobení čísel se znaménkem se nejlépe analyzuje na případu součinu čísel < 1. To znamená, že v souladu s částí 1.1 můžeme obě čísla vyjádřit výrazem

$$\tilde{X} = (-1).\tilde{x}_z + \sum_{i=1}^{n} \tilde{x}_{-i}.2^{-i} \qquad (1.29)$$

kde $\tilde{X}$ představuje číslo kladné nebo záporné vyjádřené dvojkovým doplňkem. Pro součin AxB pak získáme tento výraz

$$\tilde{A}x\tilde{B} = \left(-\tilde{b}_z\right).\tilde{A} + \sum_{i=1}^{n} \tilde{b}_{-i}.\tilde{A}.2^{-i} \qquad (1.30)$$

z kterého vyplývá, že součin čísel se znaménkem se skládá ze součtů čísel $\tilde{A}.2^{-i}$ v případech kdy $b_{-i} = 1$, od kterého odečteme v případě $b_z = 1$ hodnotu A. Součin $\tilde{A}.2^{-i}$ je číslo posunuté o i bitů vpravo s tím, že z levé strany je doplňováno znaménkovým bitem. Ponecháme na čtenáři,

aby se sám přesvědčil, že dělení záporného čísla dvěmi je realizováno posunem doprava a doplněním jedničky na nejvyšší bit. Na obr.1.6 jsou zobrazeny tři případy násobení se znaménkem, které přechází do úvahy při součinu čísel A= ±0,625 a B= ± 0,750 (+0,625=0,101; -0,625=1,011; +0,750=0,110 a -0,750=1,010), kde bit před řádovou čárkou značí znaménko.

```
A     1,0 1 0          A     0,1 0 1          A     1,0 1 1
xB    0,1 0 1          xB    1,0 1 0          xB    1,0 1 0
-------------          -------------          -------------
  1,1 1 1 0 1 0          0,0 0 0 0 0 0          0,0 0 0 0 0 0
  0,0 0 0 0 0            0,0 0 1 0 1            1,1 1 0 1 1
  1,1 0 1 0              0,0 0 0 0              0,0 0 0 0
-------------          -------------          -------------
 11,1 0 0 0 1 0          0,0 0 1 0 1 0          1,1 1 0 1 1 0
   ↑ výsledek záporný    -0,1 0 1               -1,0 1 1
                         -------------          -------------
A x B = - 0,46875       11,1 0 0 0 1 0         10,0 1 1 1 1 0
                          ↑ výsledek záporný     ↑ výsledek kladný
```

Obr.1.6 A x B = - 0,46875 A x B = + 0,46875

Dělení

Dělení dvojkových čísel se provádí podle stejného algoritmu jako dělení v soustavě desítkové. V desítkové soustavě může podíl části dělence a dělitele dosáhnout hodnoty 0 až 9, v soustavě dvojkové pouze hodnoty 0 nebo 1. Tím se nám problém dělení redukuje na zjištění, zda dělitel je větší (do výsledku se píše 0) nebo menší (do výsledku se píše 1) než část dělence. Celý algoritmus, který je zobrazen na obr.1.7, se nejsnáze realizuje pomocí odčítaček (sčítaček s použitím dvojkového doplňku dělitele) a přepínače, který zajišťuje do dalšího výpočtu přenos rozdílu části dělence a dělitele (rozdíl je kladný) nebo původní část dělence s přidaným bitem z dalšího řádu (rozdíl je záporný).

```
A     1 0 0,0 1 1   Výsledek          1 1          Výsledek
B    -1 1 1,                        -1 1 1,
-------------                       -------------
 1 1 0 1  ..... 0                    1 1 0 0  ..... 0
   1 0 0 0 ..... ,                     1 1 1
  -1 1 1                              -1 1 1
-------------                       -------------
 0 0 0 1  ..... 1                    0 0 0 0  ..... 1      Obr.1.7
```

$A/B = 4,375_{10}/7,0 = 0,101_2 = 0,625_{10}$ Zbytek = 0.

Porovnání

Předpokládejme, že máme dvě n-bitová čísla A a B, potom můžeme napsat šest logických funkcí f_1 až f_6 porovnávající jejich hodnoty.

$$f_1 = (A = B) \qquad f_2 = (A \neq B) \qquad f_3 = (A > B) \qquad (1.31)$$
$$f_4 = (A \geq B) \qquad f_5 = (A \langle B) \qquad f_6 = (A \leq B)$$

Pro tyto funkce lze psát následující logické výrazy

$$(A = B) = (a_n = b_n) \cap (a_{n-1} = b_{n-1}) \cap ... \cap (a_0 = b_0) \qquad (1.32)$$

$$(A \neq B) = (a_n \neq b_n) \cup (a_{n-1} \neq b_{n-1}) \cup ... \cup (a_0 \neq b_0) \qquad (1.33)$$

$$(A > B) = (a_n > b_n) \cup (a_n = b_n)(a_{n-1} > b_{n-1}) \cup ... \qquad (1.34)$$
$$.. \cup (a_n = b_n) \cap ... \cap (a_1 = b_1) \cap (a_0 > b_0)$$

$$(A \geq B) = (a_n \geq b_n) \cup (a_n = b_n)(a_{n-1} \geq b_{n-1}) \cup ... \qquad (1.35)$$
$$.. \cup (a_n = b_n) \cap ... \cap (a_1 = b_1) \cap (a_0 \geq b_0)$$

kde $\cup$ představuje sjednocení (logický součet) a $\cap$ průnik (logický součin). Porovnáním funkcí f_1 až f_6 zjistíme, že platí

$$f_1 = \bar{f}_2 \qquad f_3 = \overline{f_6} \qquad f_4 = \overline{f_5} \qquad f_1 = f_4 . f_6 = \overline{f_3 . f_5} \qquad (1.36)$$

K realizaci funkcí f_1 až f_6 tedy postačí pouze funkce f_3 a f_5 nebo f_4 a f_6, které můžeme určit ze vztahů (1.34) a (1.35) nebo pomocí operace odčítání, která se v mikroprocesorech používá k porovnání dvou čísel.

Z popisu základních aritmetických operací s pevnou desetinnou čárkou vyplývá, že k jejich realizaci je nezbytná operace aritmetického součtu, negace k vytvoření jednotkového doplňku, realizaci odčítání s pomocí jednotkového doplňku a logického součinu k vytvoření mezivýsledků při násobení a funkce umožňující určit nulovost operandu nebo jeho bitu.

1.5. Základní operace s čísly v pohyblivé čárce

Nyní si ukážeme, že základní aritmetické operace s čísly s pohyblivou desetinnou čárkou se skládají ze známých operací v pevné desetinné čárce, o kterých bylo pojednáno v předcházející kapitole, rozšířené o operaci posunu směrem doprava (dělení čísla dvěma). Potřebný posun čísla směrem doleva (násobení dvěma) lze realizovat pomocí sčítání.

Při odvození vztahů pro operace v pohyblivé čárce budeme předpokládat, že oba operandy jsou ve tvaru

$$X = z_x . M_x . 2^{e_x} \qquad Y = z_y . M_y . 2^{e_y} \qquad (1.37)$$

kde e_x a e_y jsou exponenty se znaménkem, z_x, z_y jsou znaménka čísel a M_x, M_y jsou normované mantisy ($M \in$ <1.0,2.0) nebo <0.5,1.0)).

Sčítání a odčítání

Pro součet nebo rozdíl $Z = X \pm Y$ můžeme za předpokladu $e_x \geq e_y$ psát

$$Z = z_x . M_x . 2^{e_x} \pm z_y . M_y . 2^{e_y} = \left(z_x . M_x \pm z_y . M_y . 2^{\left(e_y - e_x\right)} \right) . 2^{e_x} = \qquad (1.38)$$

$$= \left(z_x . M_x \pm z_y . m_y \right) . 2^{e_x} = z_z . m_z . 2^{e_x} = z_z . M_z . 2^{e_z}$$

kde m_y je nenormovaná mantisa menšího čísla, kterou získáme posunem normované mantisy M_y doprava o $(e_y - e_x)$ bitů (srovnáme exponenty obou čísel). Jestliže je hodnota $(e_y - e_x)$ srovnatelná nebo dokonce větší než počet bitů mantisy bude se při součtu nebo rozdílu uplatňovat jenom část čísla Y nebo se neprojeví jeho hodnota vůbec. Výraz $z_x . M_x \pm z_y . m_y$ jak představuje součet nebo rozdíl dvou čísel se znaménkem v pevné desetinné čárce. Výsledkem této operace je nenormovaná hodnota mantisy výsledku m_z, kterou je třeba posunout doprava (vydělit 2) jestliže je m_z větší než největší normovaná hodnota, nebo násobit 2 (posouvat doleva) tak dlouho, dokud její hodnota nedosáhne normovaného intervalu. Za každý posun je třeba přičíst nebo odečíst jedničku z exponentu.

Násobení

Pro součin dvou operandů vyjádřených stejně jako v předešlém případě můžeme psát

$$Z = X . Y = z_x . M_x . 2^{e_x} . z_y . M_y . 2^{e_y} = \left(z_x . M_x . z_y . M_y \right) . 2^{\left(e_x + e_y\right)} = \qquad (1.39)$$

$$z_z . m_z . 2^{e_x + e_y} = z_z . M_z . 2^{e_z}$$

kde $\left(z_x . M_x\right) . \left(z_y . M_y\right)$ představuje součin dvou normovaných mantis (čísel s pevnou desetinnou čárkou) a $e_z = e_x + e_y$ je součet exponentů operandů. Násobením mantis získáváme mantisu výsledku m_z, kterou je případně třeba znormovat násobením (pro vyjádření <0.5,1.0)) nebo dělením (pro vyjádření <1.0,2.0)) podle toho, který tvar pro normovanou mantisu je v aritmetice využit. Při normování hodnoty m_z je třeba příslušně korigovat exponent.

Dělení

Pro podíl dvou operandů Y/X můžeme psát

$$Z = \frac{Y}{X} = \frac{z_y . M_y . 2^{e_y}}{z_x . M_x . 2^{e_x}} = \frac{z_y . M_y}{z_x . M_x} . 2^{(e_x - e_y)} = z_z . m_z . 2^{(e_x - e_y)} = z_z . M_z . 2^{e_z} \qquad (1.40)$$

kde $(z_y . M_y) / (z_x . M_x)$ představuje podíl dvou normovaných mantis (čísel s pevnou desetinnou čárkou) a $e_z = e_y - e_x$ je rozdíl exponentů obou operandů. Po provedení podílu je opět nutné znormovat podíl m_z a zkorigovat příslušně exponent.

Odmocnina

Pro realizaci odmocniny je nejprve nutné upravit exponent odmocňovaného čísla tak, aby byl dělitelný dvěma a proto jej rozložíme takto

$$e_y = 2 . e'_y + e''_y \qquad (1.41)$$

kde $e'_y = e / 2$ a $e''_y = (e_y)_{\mathrm{mod}\ 2}$. Za předpokladu, že znaménko je kladné můžeme psát

$$Z = \sqrt{M_y . 2^{2.(e'_y + e''_y)}} = 2^{e'_y + e''_y} . \sqrt{M_y . 2^{-e''_y}} = \sqrt{m_y} . 2^{e'_y + e''_y} = M_z . 2^{e_z} \qquad (1.42)$$

kde m_y je mantisa M_y případně posunutá o jednu pozici doprava, když hodnota e_y je lichá. Odmocninu z m_y čísla s pevnou desetinnou čárkou realizujeme algoritmy, které lze nalézt [15].

2. Logické kombinační obvody

2.1. Definice logické funkce a základní operátory Booleovy algebry

Logický obvod se nazývá kombinačním, jestliže jeho výstupy závisí pouze na vstupních kombinacích a ne na jejich předcházejících hodnotách, s výjimkou krátkého přechodného děje. Kombinační obvod nemá žádnou paměť předchozích stavů a proto každé kombinaci vstupních proměnných odpovídá jen jediná výstupní kombinace. Většina logických kombinačních obvodů realizuje určitou funkcionální transformaci, jako je dekódování, kódování, přepínání a porovnání. Těmito obvody se též realizují paralelní aritmetické operace jako je sčítání, odčítání, násobení a dělení binárních (dvojkových) čísel. Funkcionální transformaci realizovanou logickým obvodem lze popsat logickými funkcemi s n vstupními logickými proměnnými $x_1, x_2, \ldots, x_n$ a m výstupními proměnnými $y_1, y_2, \ldots, y_m$ takto

$$y_j = f_j(x_1, x_2, \ldots, x_n) \qquad (2.1)$$

kde j = 1,2, ... ,m. Logické proměnné $x_1, x_2, \ldots, x_n$ a $y_1, y_2, \ldots, y_m$ jsou obvykle dvouhodnotové tzn., že mohou nabývat pouze dvou různých hodnot označovaných 0,1 nebo L,H, kterým v elektronických logických obvodech odpovídá velikost rozdílu napětí mezi dvěma uzly obvodu nebo proudu určitou větví obvodu.

Logická funkce n dvouhodnotových proměnných přiřazuje každé z 2^n možných kombinací hodnot $x_1, x_2, \ldots, x_n$ hodnotu funkce $\boldsymbol{y_j = f_j(x_1, x_2, \ldots, x_n)}$, kde $y_j \in (0,1)$. Pokud je známa hodnota funkce f pro všechny možné kombinace vstupních proměnných, pak přiřazení je jednoznačné a logickou funkci označujeme jako **funkci určitou**. V případě, že některým kombinacím není přiřazena hodnota funkce (uvedené kombinace při správné činnosti vnějších obvodů nemohou nastat nebo nám při nich nezáleží na hodnotě funkce), pak hodnotu funkce označujeme symbolem X a logickou funkci označujeme jako **funkci neurčitou**.

Logické funkce f_j popisující transformaci realizovanou logickým obvodem lze zapsat do **kombinační tabulky** (pravdivostní tabulky), která určuje závislost mezi vstupním a výstupním stavem obvodu. Pro n vstupních dvouhodnotových proměnných má kombinační tabulka 2^n řádků. Obecný příklad kombinační tabulky je zobrazen tab.2.1.

Vstupní stav	Výstupní stav
$x_1, x_2, \ldots, x_n$	$y_1 = f_1(x_1, x_2, \ldots, x_n)$, ... , $y_m = f_m(x_1, x_2, \ldots, x_n)$
$x_1, x_2, \ldots, x_n$	$y_1, y_2, \ldots, y_m$

2^n řádků *Tabulka 2.1*

Booleova algebra reprezentuje jeden z možných matematických jazyků, jehož prostřednictvím lze jednoznačně popsat chování dvouhodnotových kombinačních obvodů i vnitřní strukturu obvodu. Boolovu algebru lze zavést souborem postulátů a teorémů, které jsou podrobně popsány v práci [2], a z kterých uvedeme jen ty nejdůležitější:

Logická proměnná

Logická proměnná x je veličina, která může nabývat pouze dvou hodnot (označovaných 0 a 1) a nemůže se spojitě měnit. Tuto definici lze vyjádřit

$$x = 1 \text{ jestliže } x \neq 0 \text{ a } x = 0 \text{ jestliže } x \neq 1$$

Funkce rovnosti

Říkáme, že dvě logické proměnné x_1 a x_2 se rovnají, když $x_1 = 1$ a $x_2 = 1$ nebo $x_1 = 0$ a $x_2 = 0$, což zapisujeme $x_1 = x_2$. Dvě veličiny $A = a_n....a_2a_1$ a $B = b_n....b_2b_1$ se sobě rovnají, když platí $a_i = b_i$ pro všechna i.

Logické operátory

Jsou definovány tři logické operátory, které mezi proměnnými provádějí tři základní operace negace, logický součin a logický součet.

Negace		Logický součet (OR)			Logický součin (AND)		
x	$\overline{x}$	x	y	x + y	x	y	x.y
0	1	0	0	0	0	0	0
1	0	0	1	1	0	1	0
		1	0	1	1	0	0
		1	1	1	1	1	1

Tabulka 2.2

Nejdůležitější postuláty

a. Univerzální vazby	x + 0 = x	x . 0 = 0
	x + 1 = 1	x . 1 = x
b. Doplňky	$x + \overline{x} = 1$	$x . \overline{x} = 0$
c. Idempotence	x + x = x	x . x = x

d. De Morganova pravidla	$\overline{x+y} = \overline{x}.\overline{y}$	$\overline{x.y} = \overline{x}+\overline{y}$
e. Absorbce	$x + x.y = x$	$x.(x + y) = x$
	$x+\overline{x}.y = x+y$	$\overline{x}.(x+y) = \overline{x}.y$
f.Absorbce konsensu	$x.y+\overline{x}.z+y.z = x.y+\overline{x}.z$	
	$(x+y).(\overline{x}+z).(y+z) = (x+y).(\overline{x}+z)$	

Zápis logické funkce

Je dokázáno, že funkce $f(x_1,x_2,\ldots,x_n)$ může být zapsána ve dvou tvarech, zvaných základní součtový tvar a základní součinový tvar.

$f_{\text{základní součtový tvar}}$ = součet základních součinů přímých nebo negovaných proměnných

$f_{\text{základní součinový tvar}}$ = součin základních součtů přímých nebo negovaných proměnných

V prvém případě nabývá každý základní součin (minterm) hodnoty 1 pro tu kombinaci proměnných, kdy funkce má mít hodnotu 1 a hodnoty 0 pro všechny ostatní kombinace. V druhém případě nabývá každý součet (maxterm) hodnoty 0 pro tu kombinaci, kdy funkce má mít hodnotu 0 a pro všechny ostatní kombinace nabývá hodnoty 1.

Příklad 2.1

Odvoďte základní součtový a základní součinový tvar logické funkce $f(x_1,x_2,x_3)$ pro kterou platí, že f = 1, když většina proměnných je rovna jedné a f = 0 v ostatních případech. Logickou funkci $f(x_1,x_2,x_3)$ nejprve vyjádříme pomocí kombinační tabulky tab.2.3.

x_1	x_2	x_3	$f(x_1,x_2,x_3)$
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

Tabulka 2.3

a) Základní součtový tvar vyjadřuje funkci jako součet případů, kdy nabývá hodnoty 1.

Případy, kdy f = 1

kombinace x_3,x_2,x_1	odpovídající součiny (mintermy)
0 1 1	$\overline{x}_3.x_2.x_1$
1 0 1	$x_3.\overline{x}_2.x_1$
1 1 0	$x_3.x_2.\overline{x}_1$
1 1 1	$x_3.x_2.x_1$

Odtud funkce $f = \overline{x}_3.x_2.x_1 + x_3.\overline{x}_2.x_1 + x_3.x_2.\overline{x}_1 + x_3.x_2.x_1$

b) Základní součinový tvar vyjadřuje funkci jako

součin případů, kdy nabývá hodnoty 0. Případy, kdy f = 0

kombinace x_3, x_2, x_1	odpovídající součty (maxtermy)	Odtud funkce
0 0 0	$x_3 + x_2 + x_1$	$f = \left(x_3 + x_2 + x_1\right).\left(\overline{x}_3 + x_2 + x_1\right).$
0 0 1	$x_3 + x_2 + \overline{x}_1$	$.\left(x_3 + \overline{x}_2 + x_1\right).\left(x_3 + x_2 + \overline{x}_1\right)$
0 1 0	$x_3 + \overline{x}_2 + x_1$	
1 0 0	$\overline{x}_3 + x_2 + x_1$	

2.2. Zjednodušování zápisu logické funkce

Logická funkce vyjádřená úplnou základní součtovou nebo součinovou formou z kombinační tabulky není jediným možným vyjádřením realizované logické funkce. Dá se většinou nalézt jednodušší algebraické vyjádření, u něhož můžeme předpokládat, že povede na jednodušší technickou realizaci obvodu. Který ze zápisů logické funkce povede na minimální složitost obvodu, závisí nejen na použitých logických členech, ale též na dalších kritériích, která jsou na vlastnosti obvodu kladena, jako je zpoždění a spotřeba obvodu, spolehlivost, potlačení hazardních stavů, atd. V této části budou popsány tři procedury, kterými lze dospět k minimálnímu algebraickému vyjádření logické funkce. První metodou je algebraická minimalizace, která je metodou intuitivní a nevhodnou pro více jak tři proměnné. Vychází při úpravách zápisu funkce ze znalosti teorémů a postulátů Boolovy algebry.

Příklad 2.2 Zjednodušte logickou funkci $f = \overline{x}_3.x_2.x_1 + x_3.\overline{x}_2.x_1 + x_3.x_2.\overline{x}_1 + x_3.x_2.x_1$ ***v základním tvaru***

K funkci f můžeme nejprve přičíst dvakrát součin $x_3x_2x_1$ na základě postulátu 4c, kdy $x_3x_2x_1 + x_3x_2x_1 = x_3x_2x_1$. Funkci můžeme upravit takto

$$f = \overline{x}_3.x_2.x_1 + x_3.\overline{x}_2.x_1 + x_3.x_2.\overline{x}_1 + x_3.x_2.x_1 + x_3.x_2.x_1 + x_3.x_2.x_1 = \qquad (2.2)$$

$$x_2x_1.\left(\overline{x}_3 + x_3\right) + x_3x_1.\left(\overline{x}_2 + x_2\right) + x_3x_2.\left(\overline{x}_1 + x_1\right) =$$

$$x_2x_1.1 + x_3x_1.1 + x_3x_2.1 = x_2x_1 + x_3x_1 + x_3x_2$$

Příklad 2.3 Dokažte, že platí $x_2.x_1 + x_1.\overline{x}_2.x_3 = x_2.x_1 + x_3.x_1$

Vezměme levou stranu rovnice, v které první výraz vynásobíme logickou jedničkou vytvořenou pomocí proměnné x_3 tzn. $\left(x_3 + \overline{x}_3\right)$

$$x_2.x_1 + x_3.\overline{x}_2.x_1 = x_2.x_1.\left(\overline{x}_3 + x_3\right) + x_3.\overline{x}_2.x_1 = \qquad (2:3)$$

$$x_2.x_1.x_3 + x_2.x_1.\overline{x}_3 + x_3.\overline{x}_2.x_1 + x_3.x_2.x_1 =$$

$$x_2x_1.1 + x_3x_1.1 = x_2x_1 + x_3x_1$$

Vhodnější než algebraická minimalizace je zjednodušování součtové (součinové) funkce iterační procedurou s využitím součtových (součinových) tvarů označovaných jako konsensy.

Definice ***Konsens součinů $x_i a$ a $\overline{x}_i b$ (součtů $(x_i + g)$ a $(\overline{x}_i + d)$ je součin $a.b$ (součet $g + d$, kde a a b jsou součiny (g a d jsou součty), které neobsahují ani x_i ani $\overline{x}_i$***

V tabulce 2.4 jsou uvedeny příklady konsensů součinů. Vlastnost konsensu součinů lze vyjádřit touto rovnicí

$$x_i.\alpha + \overline{x}_i.\beta = \alpha.\beta + x_i.\alpha + \overline{x}_i.\beta \qquad (2.4)$$

Minimalizace funkce pomocí konsensů probíhá ve dvou krocích případně iteračně opakovaných.

$x_i.\alpha$	$x_i.\beta$	konsens	x_i
$x_1.x_2.x_4$	$\overline{x}_4.x_5.x_6$	$x_1.x_2.x_5.x_6$	x_4
$x_1.x_4.x_5$	$\overline{x}_4.x_5.x_6$	$x_1.x_5.x_6$	x_2
$\overline{x}_4.x_5.x_6$	$x_4.x_5.\overline{x}_6$	0	x_4, x_6
$x_4.x_5.x_6$	$\overline{x}_6.x_1$	není definován	

Tabulka 2.4

1. Výraz ve tvaru součtu součinů nebo součinu součtů se rozšíří o některé nebo všechny konsensy.

2. Z rozšířeného výrazu (pomocí získaných konsenců) vypustíme všechny složky, které lze na základě teorémů Booleovy algebry absorbovat ostatními složkami výrazu.

3. V případě, že byly odvozeny jen některé konsensy, opakujeme body 1 a 2 až po dosažení minimálního výrazu.

Příklad 2.4 ***Minimalizujte logickou funkci $f = x_1.x_2 + \overline{x}_1.x_3 + \overline{x}_2.x_4 + x_3.x_4$***

$$f = x_1.x_2 + \overline{x}_1.x_3 + \overline{x}_2.x_4 + x_3.x_4 \qquad (2:5)$$

konsens $x_2.x_3$

konsens $x_4.x_3$

Protože poslední výraz $x_3.x_4$ funkce f je konsensem prvních tří součinů, můžeme jej z výrazu vypustit a psát tento zjednodušený výraz pro funkci f

$$f = x_1.x_2 + \overline{x}_1.x_3 + \overline{x}_2.x_4 \qquad (2.6)$$

Příklad 2.5 ***Zjednodušte funkci $f = (\overline{x}_1 + x_2).(x_1 + x_2 + x_3).(\overline{x}_1 + x_3).(x_2 + x_3)$***

$$f = (\overline{x}_1 + x_2).(x_1 + x_2 + x_3).(\overline{x}_1 + x_3).(x_2 + x_3) = \qquad (2.7)$$

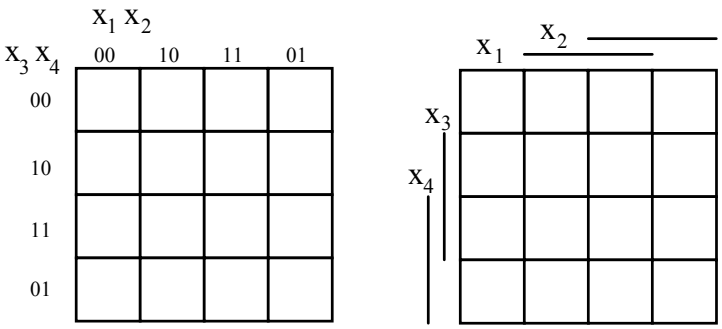
$$= (\overline{x}_1 + x_2).(\overline{x}_1 + x_3).\left[x_1.(x_2 + x_3) + (x_2 + x_3).(x_2 + x_3)\right] =$$

$$= (\overline{x}_1 + x_2).(\overline{x}_1 + x_3).\left[(x_2 + x_3).(x_1 + 1)\right] =$$

$$= (\overline{x}_1 + x_2).(\overline{x}_1 + x_3).(x_2 + x_3)$$

Karnaughova metoda minimalizace pomocí mapy

Karnaughova mapa představuje další prostředek reprezentace logické funkce. Mapa je vhodně uspořádaný zápis kombinační tabulky daný transformací jednoho řádku tabulky na jedno pole mapy. Zapíšeme-li na jednotlivá pole mapy hodnoty funkce tak, jak jsou definovány pro příslušné kombinace vstupních proměnných, bude mapa jednoznačně reprezentovat danou funkci. Přiřazení kombinací hodnot vstupních proměnných jednotlivým polím mapy se označuje jako kódování. Řádky i sloupce Karnaughovy mapy jsou kódovány Grayovým kódem.



Obr.2.1

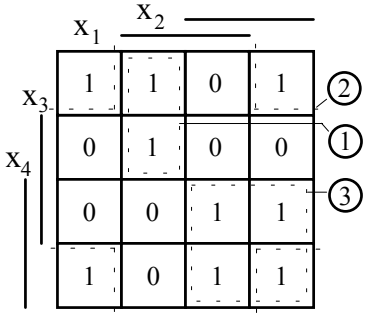
Číslo	Binární kód $x_3\,x_2\,x_1$	Grayův kód $x_3\,x_2\,x_1$
0	0 0 0	0 0 0
1	0 0 1	0 0 1
2	0 1 0	0 1 1
3	0 1 1	0 1 0
4	1 0 0	1 1 0
5	1 0 1	1 1 1
6	1 1 0	1 0 1
7	1 1 1	1 0 0

Tabulka 2:5

Základní vlastností Grayova kódu je to, že sousední slova konstantní délky se liší pouze v jedné proměnné. Tuto vlastnost splňuje i první a poslední kódové slovo (kód je uzavřen sám do sebe) tab. 2:5. Na obr.2.1 je příklad Karnaughovy mapy pro čtyři proměnné. Díky tomu, že vstupní proměnné jsou v řádcích i sloupcích mapy kódovány Grayovým kódem,

$x_4\,x_3\,x_2\,x_1$	f	$x_4\,x_3\,x_2\,x_1$	f
0 0 0 0	1	1 0 0 0	1
0 0 0 1	1	1 0 0 1	0
0 0 1 0	1	1 0 1 0	1
0 0 1 1	0	1 0 1 1	1
0 1 0 0	0	1 1 0 0	0
0 1 0 1	1	1 1 0 1	0
0 1 1 0	0	1 1 1 0	1
0 1 1 1	0	1 1 1 1	1

Tabulka 2.6



Obr.2.2

liší se výrazy (součiny nebo součty) dvou sousedních políček pouze v jedné proměnné. Spojováním výrazů sousedních políček provádíme algebraickou minimalizaci, která díky jasnému geometrickému postupu vyhýbá problematickému hledání těchto součtů nebo součinů. Jako příklad vezmeme funkci definovanou kombinační tabulkou 2.6. Odpovídající Karnaughova mapa je na obr.2:2. Základní součtový tvar této funkce je dán touto rovnicí

$$f = \overline{x}_4.\overline{x}_3.\overline{x}_2.\overline{x}_1 + \overline{x}_4.\overline{x}_3.\overline{x}_2.x_1 + \overline{x}_4.\overline{x}_3.x_2.\overline{x}_1 + \overline{x}_4.x_3.\overline{x}_2.x_1 + \qquad (2{:}8)$$

$$x_4.\overline{x}_3.\overline{x}_2.\overline{x}_1 + x_4.\overline{x}_3.x_2.\overline{x}_1 + x_4.\overline{x}_3.x_2.x_1 + x_4.x_3.x_2.\overline{x}_1 + x_4.x_3.x_2.x_1$$

V mapě obr.2.2 můžeme udělat tři smyčky, kterými spojíme sousední políčka. Smyčka ze dvou políček (označená č.1) může být vyjádřena

$$\overline{x}_4.x_3.\overline{x}_2.x_1 + x_4.\overline{x}_3.\overline{x}_2.\overline{x}_1 = \overline{x}_4.\overline{x}_2.x_1.\left(\overline{x}_3 + x_3\right) = \overline{x}_4.\overline{x}_2.x_1 \qquad (2{:}9)$$

Smyčka ze čtyř políček v rozích mapy (označená č.2)

$$\overline{x}_4.\overline{x}_3.\overline{x}_2.\overline{x}_1 + \overline{x}_4.\overline{x}_3.x_2.\overline{x}_1 + x_4.\overline{x}_3.\overline{x}_2.\overline{x}_1 + x_4.\overline{x}_3.x_2.\overline{x}_1 = \qquad (2.10)$$

$$\overline{x}_4.\overline{x}_3.\overline{x}_1.\left(\overline{x}_2 + x_2\right) + x_4.\overline{x}_3.\overline{x}_1.\left(\overline{x}_2 + x_2\right) =$$

$$\overline{x}_3.\overline{x}_1.\left(\overline{x}_4 + x_4\right) = \overline{x}_3.\overline{x}_1$$

Smyčka ze čtyř políček (označená č.3)

$$x_4.x_3.x_2.x_1 + x_4.\overline{x}_3.x_2.x_1 + x_4.x_3.x_2.\overline{x}_1 + x_4.\overline{x}_3.x_2.\overline{x}_1 = \qquad (2{:}11)$$

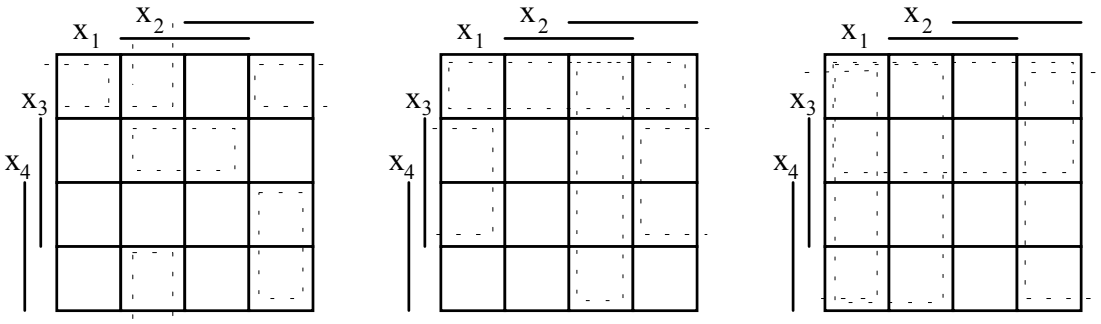
$$= x_4.x_2.x_1.\left(x_3 + \overline{x}_3\right) + x_4.x_2.\overline{x}_1.\left(x_3 + \overline{x}_3\right) = x_4.x_2.\left(x_1 + \overline{x}_1\right)$$

Odtud dostaneme výsledný vztah

$$f_{\min} = \overline{x}_4.\overline{x}_2.x_1 + \overline{x}_3.\overline{x}_1 + x_4.x_2 \qquad (2.12)$$

Při spojení čtyř políček (č.3) v mapě jsme použili jedno rohové políčko, které bylo již použito ve smyčce (č.2). To však nevadí, protože k logické funkci můžeme na základě postulátu c (idempotence) přidat tentýž součin několikrát.

Smyčkami ze dvou, čtyř, osmi, atd. (2^n, kde $n = 1,2,\ldots$) sousedních políček, jejichž možné umístění v mapě pro čtyři proměnné je zobrazeno na obr.2.3, se automaticky vyloučí proměnné, které mění svůj stav a zůstanou ty, které svůj stav nemění. Proměnné, které nemění



Obr.2.3

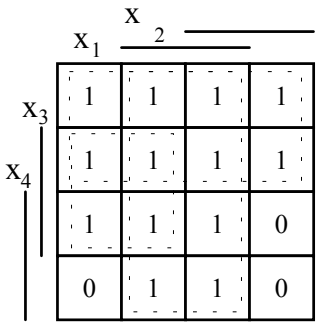
svůj stav se ve výrazu objevují:

- pro **součtový tvar** (smyčky z jednotkových políček) v přímém tvaru - pokud proměnné nabývají hodnoty 1, negované - pokud proměnné nabývají hodnoty 0
- pro **součinový tvar** (smyčky z nulových políček) v přímém tvaru - pokud proměnné nabývají hodnoty 0, negované - pokud proměnné nabývají hodnoty 1

Minimalizace logické funkce pomocí Karnaughovy mapy probíhá na základě těchto tří pravidel:

1. Musí být pokryta všechna políčka, kde funkce nabývá hodnoty 1 (pro součtovou formu) nebo hodnoty 0 (pro součinovou formu).
2. Snažíme se dosáhnou minimálního počtu smyček (tzn. součtů nebo součinů).
3. Snažíme se dosáhnout minimálního počtu proměnných v jednotlivých součinech nebo součtech.

Příklad 2.8 Minimalizujte výraz $\overline{x}_1.\overline{x}_2.x_3 + x_1.x_4 + \overline{x}_4.x_3 + \overline{x}_3.x_1 + \overline{x}_4.\overline{x}_1$ ***pomocí Karnaughovy mapy.***



Obr.2.4

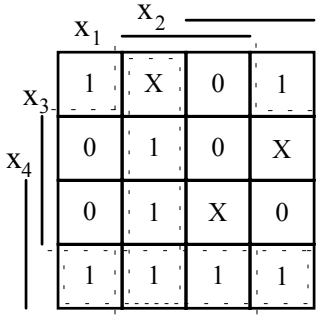
Nejprve zapíšeme logický výraz do mapy (obrácený postup vytváření výrazu z mapy). Např výrazu $\overline{x}_1.\overline{x}_2.x_3$ budou v mapě odpovídat dvě jedničky pro $x_1 = x_2 = 0$ a $x_3 = 1$ (tj. první sloupec druhý a třetí řádek). V získané mapě můžeme vytvořit tři smyčky. Pro splnění bodu 3 zásad pro minimalizaci vytvoříme druhou smyčku nikoliv ze čtyř ($x_1.x_4$), ale z osmi políček, které odpovídá výraz x_1. Třetí smyčku opět neuděláme ani z jednoho nebo dvou políček, ale z políček čtyř ($\overline{x}_2.x_3$). Tak získáme minimální výraz logické funkce f

$$f_{min} = \overline{x}_4 + x_1 + \overline{x}_2.x_3 \qquad (2.13)$$

$x_4\,x_3\,x_2\,x_1$	f	$x_4\,x_3\,x_2\,x_1$	f
0 0 0 0	1	1 0 0 0	1
0 0 0 1	X	1 0 0 1	1
0 0 1 0	1	1 0 1 0	1
0 0 1 1	0	1 0 1 1	1
0 1 0 0	0	1 1 0 0	0
0 1 0 1	1	1 1 0 1	1
0 1 1 0	X	1 1 1 0	0
0 1 1 1	0	1 1 1 1	X

Tabulka 2.7

Příklad 2.9 Odvoďte zjednodušený vztah pro neúplně zadanou logickou funkci, která je popsána kombinační tabulkou 2.7.



Obr.2.5

Karnaughova mapa této neurčité logické funkce je na obr.2.5. Při minimalizaci neurčité funkce postupujeme stejně ja-

ko pro funkce určité s tím rozdílem, že políčka s obsahem X použijeme jako jednotková nebo nulová pole podle toho, jak to usnadní tvoření smyček. V tomto případě je výhodné neurčitý stav pro kombinaci $x_1 = 1, x_2 = x_3 = x_4 = 0$ realizovat jako jednotkový a tak získat tento minimální výraz

$$f_{\min} = \overline{x}_1 . \overline{x}_3 + \overline{x}_3 . x_4 + x_1 . \overline{x}_2 \qquad (2.14)$$

Příklady k samostatnému řešení

Příklad 2.2.1 Zjednodušte pomocí teorémů Booleovy algebry funkci čtyř proměnných

$$f = x_4 + x_2 . x_3 . x_4 + x_2 . x_3 . \overline{x}_4 + x_3 . \overline{x}_4 + x_1 . x_4 + \overline{x}_1 . x_2$$

Příklad 2.2.2 Užitím de Morganova zákona určete komplementy funkcí

$$x_1 . \left(\overline{x}_2 + \overline{x}_3 . x_4 + \overline{x}_5 . x_6 \right)$$

$$x_1 . \overline{x}_2 . x_3 + \overline{x}_1 + x_2 + x_4 . \left(x_1 . x_2 . \overline{x}_4 + \overline{x}_2 \right)$$

$$x_1 + \overline{x}_2 . \overline{x}_3 . x_4 . \left(\overline{x}_1 . \overline{x}_4 + x_2 . \left(\overline{x}_3 + x_1 \right) \right)$$

Příklad 2.2.3 Dokažte, že platí vlastnost konsensu popsaná rovnicí (2.4).

Příklad 2.2.4 Odvoďte minimální součinovou formu funkce, $f = x_1 . x_2 + x_2 . x_3 + \overline{x}_2 . \overline{x}_3$ bez použití Karnaughovy mapy.

Příklad 2.2.5 Minimalizujte pomocí konsensů funkce

$$x_1 . x_3 . \overline{x}_4 + \overline{x}_1 . \overline{x}_2 . x_3 + \overline{x}_1 . x_3 . x_4 + \overline{x}_2 . x_3 . x_4$$

$$\left(x_1 + \overline{x}_2 + \overline{x}_3 \right) . \left(x_1 + x_2 + \overline{x}_3 + \overline{x}_4 \right) . \left(\overline{x}_1 + x_2 + x_3 + x_4 \right) . \left(x_2 + \overline{x}_3 + x_4 \right)$$

Příklad 2.2.6 Odvoďte užitím Karnaughovy mapy minimální součinovou a součtovou formu funkcí čtyř a pěti proměnných nabývajících jednotkové hodnoty pro tyto mintermy (mintermy se někdy vyjadřují číslem, které binárně představuje příslušnou kombinací vstupních proměnných).

$$f_1 \left(x_4, x_3, x_2, x_1 \right) = \sum \left(0,1,4,5,12,13,14 \right)$$

$$f_1 \left(x_4, x_3, x_2, x_1 \right) = \sum \left(0,4,9,13,18,19,22,23,25,29 \right)$$

2.3. Návrh logických kombinačních obvodů

Při návrhu logických kombinačních obvodů je zpravidla třeba vykonat následující body:

1. Z obecných požadavků na chování obvodu určit počet vstupních a výstupních proměnných a stanovit kombinační (pravdivostní) tabulku.
2. Z pravdivostní tabulky určit logické funkce popisující chování obvodu, provést jejich minimalizaci s ohledem na soubor logických členů pro vytvoření struktury obvodu.
3. Pokud je to nutné, je třeba optimální výrazy vyšetřit zda nevykazují hazardní stavy, popřípadě provést jejich rozšíření o nadbytečné složky, které vznik hazardních stavů potlačují.
4. Z navržených výrazů realizovat logický kombinační obvod.
5. Pro kontrolu provést analýzu navrženého obvodu a ověřit zda splňuje požadavky na něj kladené.

V předcházejících částech byl probrán v podstatě bod 2 a nyní se zaměříme na bod 4, který pojednává o realizaci logických obvodů pomocí logických členů, které se vyrábějí ve formě monolitických obvodů. Jedná se především o členy NAND se dvěma, třemi, čtyřmi, osmi a třinácti vstupy, členy NOR se dvěma, třemi, čtyřmi a osmi vstupy a členy AND-OR-INVERT. Dále si však ukážeme jak lze k realizaci logické funkce využít i obvody se střední a vysokou hustotou integrace jako jsou universální logické jednotky nebo programovatelné paměti a logická pole (PROM, EPROM, PAL, GAL, EPLD, FPGA, LCA atd.).

2.3.1. Realizace logických kombinačních obvodů se členy NAND a NOR

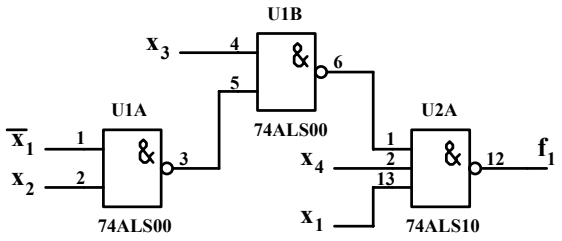
Logický člen NAND realizuje negaci logického součinu a logický člen NOR realizuje negaci logického součtu. Logický člen NAND i NOR tvoří úplný systém, protože umožňuje realizovat všechny základní operace: logický součin, logický součet a negaci. Výčet základních logických členů NAND i NOR je uveden v tabulce 2.8. Kromě těchto základních obvodů jsou k dispozici Shottkyho logické obvody s označením 74S-- nebo 74F--, které se vyznačují přibližně čtvrtinovým zpožděním signálu přes logický člen, a obvody typu 74LS-- , které se vyznačují až pětinovým odběrem ze zdroje, obvody typu 74ALS-- , které se vyznačují až desetinovým odběrem ze zdroje a přibližně polovičním zpožděním vůči standardní řadě TTL. Kromě těchto starších obvodů jsou nyní k dispozici obvody typu 74AS-- vyrobené v bipolární technologii a 74HC--, 74HCT--, 74AC--, 74ACT-- vyrobené v technologii CMOS a 74BCT-- a 74ABT-- vyrobené v kombinované technologii BiCMOS.

Realizaci navržených logických funkcí se členy NAND nebo NOR lze nejsnáze provádět na základě teorému 1.

Teorém 1. Každý kombinační obvod realizovaný logickými členy NAND (NOR), který má k stupňů (k=1,2,3,) modeluje Boolovský výraz, který má v lichých stupních operace logický součet (součin) nebo negaci a v sudých stupních logický součin (součet) nebo negaci. Vstupní proměnné lichých stupňů jsou ve výrazu v komplementu.

TYP OBVODU			Počet vstupů	Počet logických členů	Provedení	Logický člen
TTL		CMOS				
	s OK					
7400	7401	4011	2	4		NAND
	7403		2	4		NAND
74804			2	6		NAND
7437	7438		2	4	výkonové	NAND
7410	7412	4023	3	3		NAND
7420	7422	4012	4	2		NAND
7430		4068	8	1		NAND
74133			13	1		NAND
74132		4093	2	4	Smith	NAND
7402	7433	4001	2	4		NOR
74805			2	6		NOR
7427		4025	3	3		NOR
7425		4002	4	2		NOR
74260			2	5	Shottky	NOR
		4078	8	1		NOR
7404	7405	4069	1	6		NEGACE
7414		4584	1	6	Smith	NEGACE

Tabulka 2.8



Obr.2.6

Jednotlivé stupně označujeme vzestupně od poslední matematické operace, kterou musíme vykonat. Operace negace se ve výsledném výrazu neobjevuje, zajišťuje pouze posun operace logický součet nebo logický součin na požadovaný stupeň obvodu, který ji realizuje.

Příklad 2.10 Analyzujte logické kombinační obvody z obr.2.6 a obr.2.7.

Pro funkci f_1, která je realizována logickými členy NAND, můžeme psát

$$f_1 = \overline{x_1.x_4.\overline{x_3.\overline{\overline{x}_1.x_2}}} = \overline{x}_1 + \overline{x}_4 + x_3.\overline{\overline{x}_1.x_2} = \quad (2.15)$$

$$= \overline{x}_1 + \overline{x}_4 + x_3.\left(x_1 + \overline{x}_2\right)$$

$$\uparrow \quad \uparrow \quad \uparrow \quad \uparrow$$

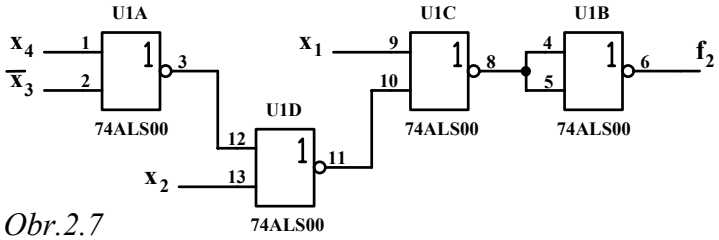
1st 1st 2st 3st

Při úpravách byl použit pouze de Morganův teorém. Pro funkci f_2, která je realizována logickými členy NOR, platí

$$f_2 = \overline{\overline{\overline{\overline{x}_3 + x_4} + x_2 + x_1}} = x_1 + \overline{x}_2.\left(x_4 + \overline{x}_3\right) \quad (2.16)$$

$$\uparrow \quad \uparrow \quad \uparrow$$

1st negace 2st 3st 4st



Obr.2.7

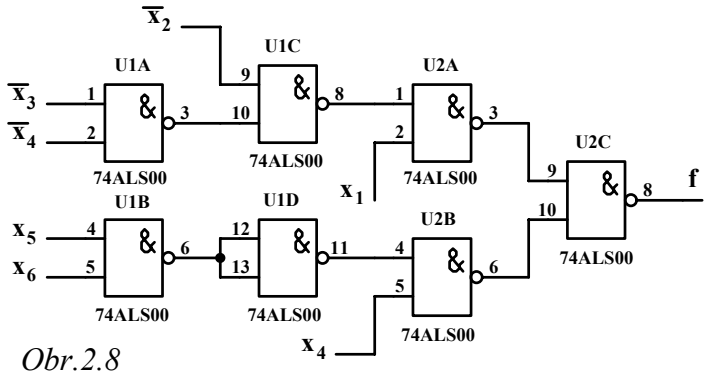
První stupeň obvodu realizující negaci, která ve výsledném vztahu nevystupuje, zajišťuje pouze posun operace logický součet na druhý stupeň.

Každý booleovský výraz lze pak realizovat obvodem s logickými členy NAND takto:

1. Určíme pořadí aritmetických operací s tím, že poslední operaci ve výrazu označíme č.1, předposlední č.2, atd.
2. Vkládáním stupňů, které realizují negaci upravíme obvod tak, aby operace logický součet připadala na stupně liché a operace logický součin na stupně sudé.
3. Vstupní proměnné v lichých stupních negujeme.

Obdobně postupujeme i při realizaci obvodu s logickými členy NOR.

Příklad 2.11 Realizujte logickou funkci $f = x_1.x_2 + x_1.\overline{x}_3.\overline{x}_4 + x_4.x_5.x_6$ ***s dvouvstupovými logickými členy NAND.***



Obr.2.8

Logickou funkci upravíme tak, aby bylo zřejmé pořadí prováděných matematických operací, které očíslujeme dle bodu 1. Nyní zjistíme, zda operace logického součtu připadá na liché stupně a operace logického součinu na stupně sudé. V tomto případě připadá logický součin proměnných $x_5.x_6$ na lichý (třetí) stupeň obvodu rov.(2.17). Protože v lichém stupni obvodu se ale realizuje logický součet, musíme do třetího stupně vložit negaci, která nám zajistí posun operace logický součin na požadovaný sudý stupeň obvodu (čtvrtý). Na obr.2.8 je pak zobrazeno výsledné zapojení obvodu k jehož realizaci je zapotřebí 1 a 3/4 obvodu MH74ALS00.

$$f = x_1.x_2 + x_1.\overline{x}_3.\overline{x}_4 + x_4.x_5.x_6 = x_1.\left(x_2 + \overline{x}_3.\overline{x}_4\right) + x_4.\left(x_5.x_6\right) \qquad (2.17)$$

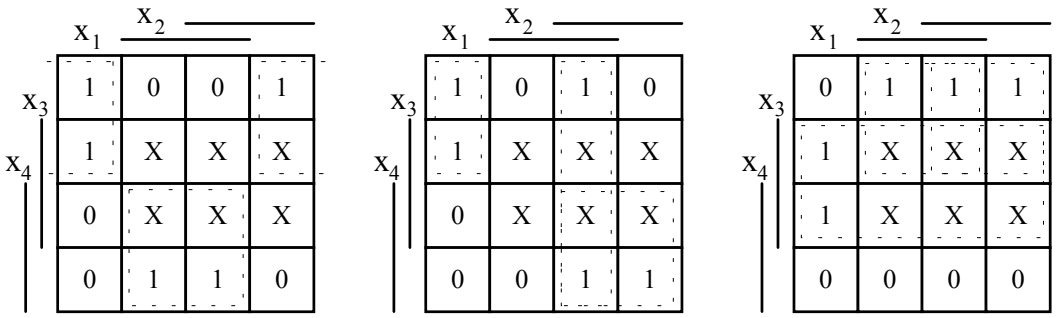
	↑	↑	↑	↑	↑	↑	
pořadí matematických operací	***2***	***3***	***4***	***1***	***2***	***3***	
stupeň logického obvodu	***2***	***3***	***4***	***1***	***2***	***4***	***3 negace***

Příklad 1.12 Navrhněte a realizujte převodník desítkového kódu BCD 5421 na kód BCD 8421+3 s logickými členy NAND.

Číslo	BCD 5421				BCD 8421+3			
	x_4	x_3	x_2	x_1	y_4	y_3	y_2	y_1
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	1	0	0	0	1	0	0	0
6	1	0	0	1	1	0	0	1
7	1	0	1	0	1	0	1	0
8	1	0	1	1	1	0	1	1
9	1	1	0	0	1	1	0	0

Tabulka 2.9

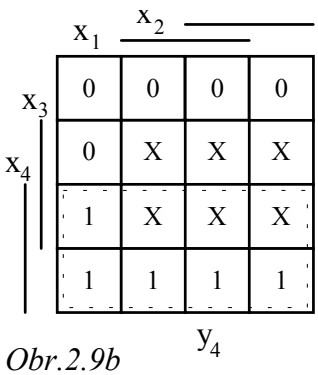
Každé dekadické číslici 0 až 9 je v jednotlivých desítkových kódech přiřazen určitý vektor ($x_n, x_{n-1}, \ldots, x_2, x_1$) konstantní délky (nyní 4), kde $x_i \in (0,1)$ pro i = 1,2, .. ,n. Počet vstupních proměnných převodníku bude roven délce vstupního vektoru, počet výstupních proměnných bude roven počtu prvků výstupního slova. Funkcionální transformace, kterou má převodník realizovat, je popsána pravdivostní tabulkou z obr.2.9. To znamená, že transformace bude realizována čtyřmi logickými funkcemi o čtyřech proměnných, z nichž každá bude představovat jeden prvek výstupního vektoru $\boldsymbol{y_i = f_i(x_4, x_3, x_2, x_1)}$ kde i=1, 2, 3, 4. Každou funkci f_i (v tomto případě neurčitou) zapíšeme do Karnaughovy mapy a provedeme její minimalizaci obr.2.9. Z obr.2.9 již snadno zjistíme tyto výrazy pro výstupní proměnné



Obr.2.9a

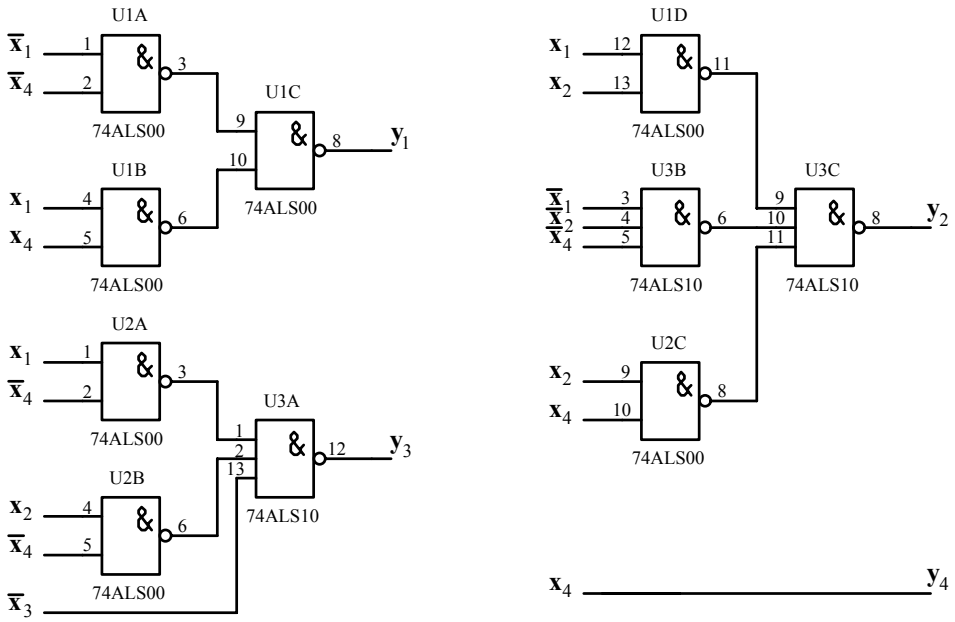
$$y_1 = \overline{x}_1.\overline{x}_4 + x_1.x_4 \qquad y_2 = \overline{x}_1.\overline{x}_2.\overline{x}_4 + x_1.x_2 + x_2.x_4 \qquad (2.18)$$

$$y_3 = x_1.\overline{x}_4 + x_2.\overline{x}_4 + x_3 \qquad y_4 = x_4$$



Obr.2.9b

Za předpokladu, že použijeme k realizaci logické členy NAND s více vstupy, bude dvoustupňová realizace dána obr.2.10. K realizaci bude třeba těchto obvodů: 74ALS10 a 1 a 3/4 74ALS00. Maximální zpoždění průchodu signálu přes převodník bude součet dob zpoždění při přechodu výstupu z úrovně 1→0 (t_{pHL}) a z úrovně 0→1 (t_{pLH}) tzn. $t_{pLH} + t_{pHL}$= (11 + 18)ns = 29ns.



Obr.2.10

2.3.2. Realizace LKO se členy AND-OR-INVERT

Logický člen AND-OR-INVERT realizuje negaci součtu logických součinů. Jedná se o člen, který realizuje na jednom čipu integrovaného obvodu dvoustupňový obvod, jehož časové parametry jsou stejné jako u obvodu NAND nebo NOR. Toho můžeme využít při realizaci časově náročnějších obvodů. Z této skupiny logických členů je v novějších řadách obvodů TTL k dispozici pět typů monolitických obvodů, které modelují funkce

$$y_1 = \overline{\sum_{i=1}^{2} x_{i1}^{o}.x_{i2}^{o}} \qquad y_2 = \overline{\sum_{i=1}^{2} x_{i1}^{o}.x_{i2}^{o}.x_{i3}^{o}} \qquad (2.19)$$

kde x_{ij}^{o} je x_{ij} nebo $\overline{x}_{ij}$. Funkci y_1 a y_2 realizují poloviny obvodu 74LS51, 74ALS51 nebo 74F51. Zde je třeba upozornit na to, že obvod 7451 z klasické řady TTL modeluje funkci y_2 shodnou s funkcí y_1.

$$y_3 = \overline{\sum_{i=1}^{2} x_{i1}^o . x_{i2}^o + \sum_{i=1}^{2} x_{i1}^o . x_{i2}^o . x_{i3}^o} \qquad (2.20)$$

Funkci y_3 realizuje jeden obvod 74LS54 nebo 74ALS54. Stejně jako u obvodu 7451, realizuje obvod 7454 z klasické řady TTL jenom čtyři součiny o dvou proměnných.

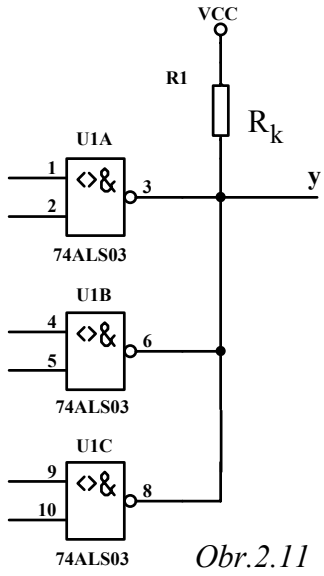
$$y_4 = \overline{\sum_{i=1}^{2} x_{i1}^o . x_{i2}^o . x_{i3}^o . x_{i4}^o} \qquad (2.21)$$

Funkci y_4 realizuje jeden obvod 74LS55.

$$y_5 = \overline{x_1^o . x_2^o + x_3^o . x_4^o + x_5^o . x_6^o . x_7^o + x_8^o . x_9^o . x_{10}^o . x_{11}^o} \qquad (2.22)$$

kde x_j je x_j nebo $\overline{x}_j$. Funkci y_5 realizuje obvod 74S64 nebo 74F64. Stejnou funkci realizuje i obvod s otevřeným kolektorem 74S65. Obvod typu AND-OR-INVERT lze realizovat i obvody s otevřeným kolektorem, kde logický součet se realizuje na společném kolektorovém odporu viz. obr.2.11. Jako příklad si uvedeme funkci realizovatelnou paralelním spojením výstupů logických členů NAND se dvěma vstupy s otevřeným kolektorem 74LS03.

$$y_6 = \overline{\sum_{i=1}^{40} x_{i1}^o . x_{i2}^o} \qquad (2.23)$$



Obr.2.11

K uvedeným funkcím y_1 až y_6 je třeba připomenout, že nejsou úplným výčtem obvodů AND-OR-INVERT. Ve standardní řadě TTL byly ještě obvody 7450 a 7453, kterými bylo možné realizovat dva nebo čtyři součiny o dvou proměnných. Tyto funkce bylo dále možné rozšířit pomocí dvou obvodů (expandérů) 7460, realizujících až čtyři součiny o čtyřech proměnných. Tato konfigurace se v nových řadách TTL již neobjevuje a proto se jí nebudeme zabývat. Bližší informace může čtenář získat v prácí [3].

Realizace logické funkce obvody AND-OR-INVERT se obvykle provádí následujícím způsobem:

1) Vyjádříme minimální součtovou formu negace zadané logické funkce. Pokud je možné pokrýt tuto formu výrazy (2.19) až (2.23), pak funkci lze realizovat přímo jednostupňovým obvodem.

2) Jestliže funkci nelze realizovat přímo, musíme provést rozklad funkce $\bar{f}$ podle jedné proměnné x_k takto

$$\bar{f} = \overline{x_k} . \overline{f_k(0)} + x_k . \overline{f_k(1)} \qquad (2.24)$$

Pokud funkce $f_k(0)$ a $f_k(1)$ lze pokrýt výrazy (2.19) až (2.23), pak funkci f lze realizovat dvoustupňovým obvodem se členy AND-OR-INVERT. Rovnici (2.24)

realizujeme v prvém stupni (bráno od výstupu) a funkce $\overline{f_k(0)}$ a $\overline{f_k(1)}$ ve stupni druhém. Funkce $f_k(0)$ a $f_k(1)$ jsou minimalizované výrazy negací funkcí $\overline{f_k(0)}$ a $\overline{f_k(1)}$.

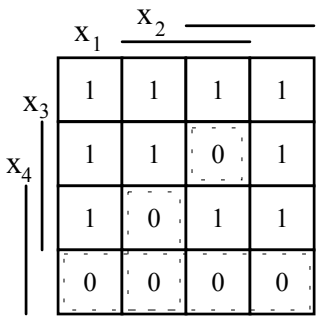
3) Nelze-li ani po rozkladu realizovat funkce $f_k(0)$ a $f_k(1)$ přímo, provedeme jejich další rozklad podle proměnné x_l $(l \neq k)$. Jsou-li v rovnici (2.24) výrazy, které neobsahují proměnnou x_l, je třeba je vynásobit logickou jedničkou $(x_l + \overline{x}_l)$.

Příklad 2.13 Realizujte logický kombinační obvod členy AND-OR-INVERT, který indikuje logickou jedničkou binárně vyjádřená čísla 0,1,2,3,4,5,6,12,14,15.

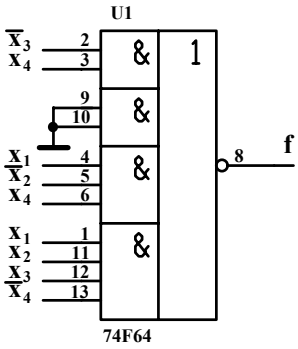
Funkci f zapíšeme do Karnaughovy mapy a vyjádříme minimální součtovou formu funkce $\overline{f}$ (v mapě budeme hledat sousední nulová políčka obr.2.12).

$$\overline{f} = \overline{x}_3.x_4 + x_1.x_2.x_3.\overline{x}_4 + x_1.\overline{x}_2.x_4 \qquad (2.25)$$

Funkce f je vyjádřena výrazem, který lze pokrýt funkcí y_5 (2.22) a funkcemi obvodů 7450 nebo 7453 s expanderem 7460. Z těchto možností se nyní rozhodneme pro realizaci rovnicí (11.1), protože vystačíme s jedním integrovaným obvodem 74S64. Na obr.2.13 je zobrazena realizace funkce f. Vstup(y) nepoužitého členu AND musíme uzemnit.



Obr.2.12



Obr.2.13

Příklad 2.14 Realizujte členy AND-OR-INVERT logický kombinační obvod indikující hodnotou log.1 čísla 1,2,4,6,7,10,11,13,15 vyjádřená binárně čtyřmi proměnnými x_4, x_3, x_2, x_1.

Funkci zapíšeme do mapy obr.2.14 a vyjádříme minimální součtovou formu funkce $\overline{f}$.

$$\overline{f} = x_1.x_2.\overline{x}_3.\overline{x}_4 + x_1.\overline{x}_2.x_3.\overline{x}_4 + \overline{x}_1.\overline{x}_2.\overline{x}_3 + \overline{x}_1.x_3.x_4 + \overline{x}_2.\overline{x}_3.x_4 \qquad (2.26)$$

Získaný výraz nelze realizovat jednostupňovým obvodem, protože funkce (2.26) není podmnožinou funkcí (2.19) až (2.23). Provedeme proto její rozklad např. podle proměnné x_3.

$$\overline{f} = x_3.\left(x_1.\overline{x}_2.\overline{x}_4 + \overline{x}_1.x_4\right) + \overline{x}_3.\left(x_1.x_2.\overline{x}_4 + \overline{x}_1.\overline{x}_2 + \overline{x}_2.x_4\right) \qquad (2.27)$$

Nalezneme minimální výrazy pro funkce $f_3(0)$ a $f_3(1)$

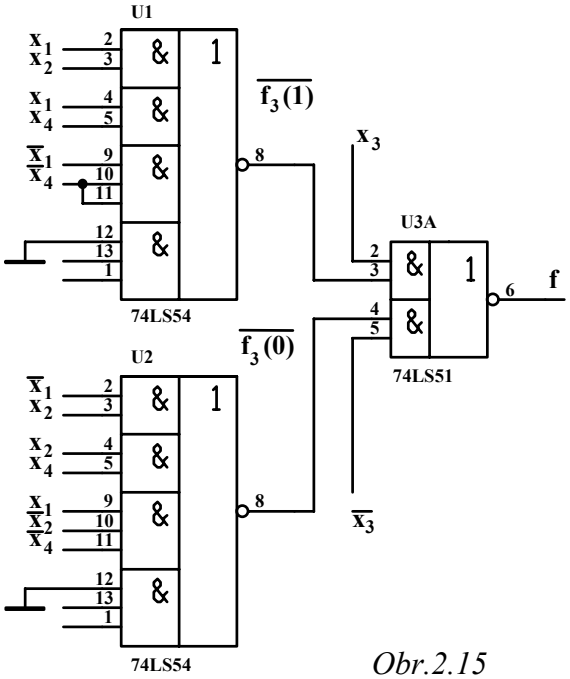
$$\overline{\overline{x}_1.\overline{x}_2 + \overline{x}_2.x_4 + x_1.x_2.\overline{x}_4} = \left(x_1 + x_2\right).\left(x_2 + \overline{x}_4\right).\left(\overline{x}_1 + \overline{x}_2 + x_4\right) = \qquad (2.28)$$

$$= \overline{x}_1.x_2.\overline{x}_4 + x_1.\overline{x}_2.\overline{x}_4 + \overline{x}_1.x_2 + x_2.x_4 + x_1.x_2.x_4 \Rightarrow$$

$$f_3(0) = x_2.x_4 + \bar{x}_1.x_2 + x_1.\bar{x}_2.\bar{x}_4 \qquad (2.29)$$

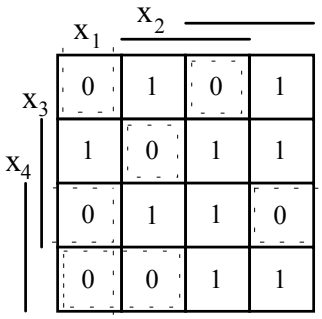
$$f_3(1) = \overline{x_1.\bar{x}_2.\bar{x}_4 + \bar{x}_1.x_4} = (x_1 + \bar{x}_4).(\bar{x}_1 + x_2 + x_4) = \qquad (2.30)$$

$$= x_1.x_2 + x_1.x_4 + \bar{x}_1.\bar{x}_4 + x_2.x_4 = x_1.x_2 + x_1.x_4 + \bar{x}_1.\bar{x}_4$$



Obr.2.15

Funkce $f_3(0)$ a $f_3(1)$ jsou podmnožinou funkce (2.20) a proto funkci f lze realizovat dvoustupňovým obvodem. Rovnici (2.27) bude realizovat první stupeň obvodu tvořený 1/2 74LS51. V druhém stupni se budou realizovat funkce $\overline{f_3(0)}$ a $\overline{f_3(1)}$ pomocí



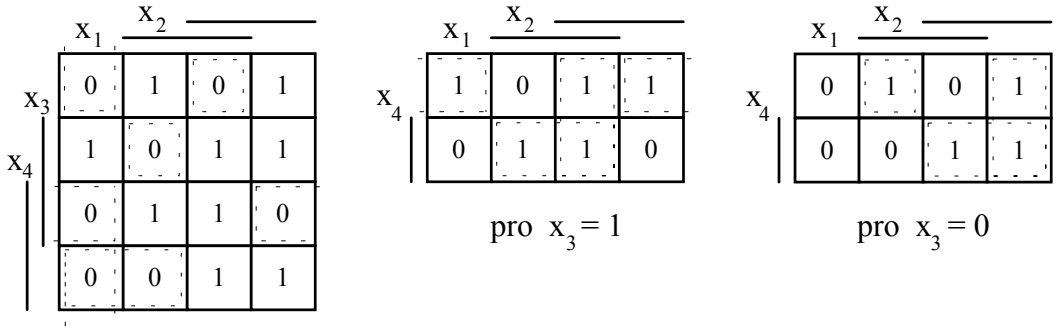
Obr.2.14

dvou obvodů 74LS54 obr.2.15.

Pro složitější funkce $\overline{f_k(0)}$ a $\overline{f_k(1)}$ je získávání minimálních výrazů jejich negací pracné. Tyto výrazy však můžeme mnohem snadněji získat přímo z Karnaughovy mapy funkce f tak, že ji rozdělíme na dvě mapy pro $x_k = 1$ a $x_k = 0$. Z map potom snadno odvodíme přímo funkce $f_k(0)$ a $f_k(1)$. Pro funkci z předešlého příkladu získáme tyto Karnaughovy mapy obr.2.16, z kterých odvodíme tyto výrazy pro $f_3(1)$ z mapy pro $x_3 = 1$ a $f_3(0)$ z mapy pro $x_3 = 0$.

$$f_3(1) = x_1.x_2 + x_1.x_4 + \bar{x}_1.\bar{x}_4 = x_2.\bar{x}_4 + x_1.x_4 + \bar{x}_1.\bar{x}_4 \qquad (2.31)$$

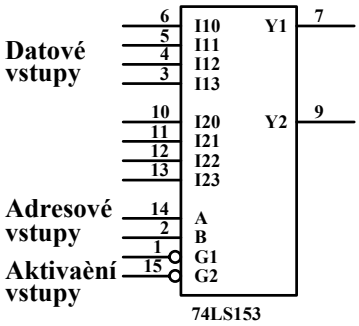
$$f_3(0) = x_2.x_4 + \bar{x}_1.x_2 + x_1.\bar{x}_2.\bar{x}_4 \qquad (2.32)$$



Obr.2.16

2.3.3. Realizace LKO s multiplexery

V předcházejících částech byly probrány způsoby realizace logických kombinačních obvodů jednoduchými logickými členy. Složitost obvodu byla posuzována z hlediska počtu použitých logických členů a počtu jejich vstupů. Tato hlediska jsou vhodná v případě realizace kombinačních obvodů z integrovaných obvodů s nízkou hustotou integrace, u nichž je velmi omezený počet logických členů v pouzdře. Při použití obvodů se střední a vysokou hustotou integrace se uvedená hlediska již tak výrazně neuplatňují a pravdivostní (kombinační) tabulka logické funkce se realizuje přímo pomocí univerzálních logických jednotek (ULJ) jako jsou multiplexery nebo pevné či programovatelné paměti.



Obr.2.17

G	B	A	Y
1	X	X	0
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3

Tabulka 2.10

Číslicový multiplexer, který z obvodového hlediska představuje jednoduchý vícepolohový přepínač, není v současnosti jako samostatná součástka již významným stavebním prvkem pro návrh logických kombinačních obvodů. Je však součástí řady základních stavebních bloků v programovatelných polích a proto se jím budeme nyní zabývat. Multiplexery jako samostatné součástky se vyrábí s 2,4,8 nebo 16 vstupy pro data, 1,2,3 nebo 4 řídícími (adresovacími) vstupy, jedním výstupem (případně komplementárním výstupem) a vstupem vybavovacím. Je-li počet adresovacích vstupů k, pak obvod má 2^k vstupů pro data a kombinace na adresovacích vstupech určuje, který signál z datových vstupů je připojen na výstup multipexeru. Výstupní proměnnou lze ve shodě se symbolickým označením obr.2.17 a pravdivostní tabulkou tab.2.10 pro multiplexer se čtyřmi datovými vstupy vyjádřit vztahem

$$Y = \overline{G}.\left(\overline{A}.\overline{B}.I_0 + A.\overline{B}.I_1 + \overline{A}.B.I_2 + A.B.I_3\right) \qquad (2.33)$$

kde A,B jsou adresovací vstupy, I_0, I_1, I_2, I_4 jsou vstupy pro data. Rovnice (2.33) představuje úplnou součtovou formu pro funkci dvou proměnných (k=2). Dá se dokázat, že multiplexerem s k adresovacími vstupy můžeme realizovat logické funkce o n = k + 1 vstupních proměnných. Vytkneme-li všechny kombinace k vstupních proměnných, které připojíme na adresovací vstupy multiplexeru, zůstanou nám u výrazů těchto kombinací závorky typu $(\overline{x}_z.f_i + x_z.f_j)$, kde f_i a f_j jsou funkční hodnoty realizované funkce a x_z je zbývající vstupní proměnná. Analýzou této závorky $(\overline{x}_z.f_i + x_z.f_j)$ snadno zjistíme, že může v závislosti na funkčních hodnotách f_i a f_j nabývat hodnot $0, 1, x_z$ nebo $\overline{x}_z$.

Příklad 2.15 Realizujte logickou funkci $f(x_1,x_2,x_3) = x_1.\bar{x}_2 + x_2.\bar{x}_3 + \bar{x}_1.x_2$ ***pomocí čtyřvstupého multiplexeru.***

K realizaci logické funkce můžeme přistoupit několika způsoby. Je-li funkce zadána výrazem můžeme provést její porovnání s logickým výrazem realizovaným multiplexerem. V našem případě se jedná o funkci tří proměnných, kterou můžeme realizovat multiplexerem se dvěma adresovacími vstupy. K adresování je možné používat všechny kombinace dvou proměnných z x_1,x_2,x_3. To znamená, že můžeme adresovat soubory (x_1,x_2), (x_2,x_3), (x_1,x_3). Zvolíme-li k adresování proměnné x_1,x_2 pak ze vztahu (2.34) pro jednotlivé vstupy odvodíme $I_0 = 0$, $I_1 = 1$, $I_2 = 1$ a $I_3 = \bar{x}_3$. Na obr.2.18 je zobrazena výsledná realizace obvodu.

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

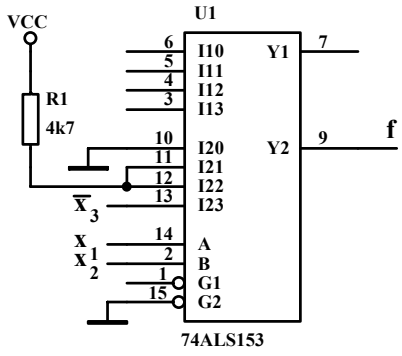
$$= x_1.\bar{x}_2 + (x_1 + \bar{x}_1).x_2.\bar{x}_3 + \bar{x}_1.x_2 =$$

$$= \bar{x}_1.\bar{x}_2.(0) + x_1.\bar{x}_2.(1) + \bar{x}_1.x_2.(\bar{x}_3 + 1) + x_1.x_2.(\bar{x}_3) \qquad (2.34)$$

x_3	x_2	x_1	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	1
1	1	1	0

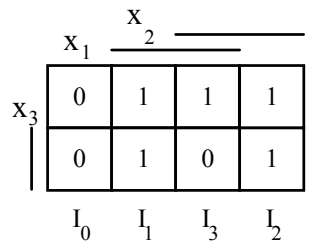
Tabulka 2.11

Máme-li logickou funkci zadanou pravdivostní tabulkou není nutné vyjadřovat logickou funkci výrazem a porovnávat jej s funkcí multiplexeru. Zvolíme n-1 proměnných k adresování multiplexeru a najdeme v pravdivostní tabulce dvojice řádků odpovídající jednotlivým kombinacím těchto n-1 proměnných, které s liší v hodnotě zbývající proměnné. Pro naši logickou funkci je pravdivostní tabulka dána tab.2.11. Zvolíme-li k adresování proměnné x_1,x_2, potom odpovídající dvojice řádků budou dány řádky (1;5),(2;6),(3;7) a (4;8) pravdivostní tabulky. Hodnota zbývající proměnné x_3 a funkční hodnota v daném řádku odpovídají jednomu výrazu závorky $(\bar{x}_z.f_i + x_z.f_j)$. Hodnotu závorky získáme nejsnadněji porovnáním hodnoty x_3 s funkční hodnotou. Pro dvojici řádků (1;5) zjistíme, že funkční hodnota nezávisí na proměnné x_3 a je rovna 0. Odtud $I_0 = 0$, protože adresující proměnné $x_1 = x_2 = 0$. Obdobně můžeme pro další dvojice řádků pravdivostní tabulky získat vztahy $I_1 = 1$, $I_2 = 1$ a $I_3 = x_3$. Výsledná realizace již byla zobrazena na obr.2.18. Další možností k odvození hodnot pro datové vstupy multiplexeru je použití předcházejícího postupu aplikovaného do Karnaughovy mapy. Mapu obr.2.19 uspo-



Obr.2.18

řádáme do dvou řádků a 2^{n-1} sloupců, kde n je počet vstupních proměnných. Řádky mapy se od sebe liší ve zbývající proměnné x_z, která nebyla použita k adresování. Sloupce mapy odpovídají kombinacím adresujících proměnných, které binárně určují hodnotu datového vstupu. Hodnotu, kterou musíme na datový vstup připojit, určuje obsah sloupce.



Obr.2.19

Jak vyplývá z tabulky 2.12, v které je výčet dostupných multiplexerů vyráběných v řadách TTL LS a ALS lze s multiplexery jednostupňovým obvodem realizovat až funkci pěti proměnných. Logickou funkci více než pěti proměnných je možné realizovat vícestupňovým obvodem nebo za pomoci multiplexerů s třístavovým výstupem. V případě vícestupňového obvodu s multiplexery vytkneme z funkce $\boldsymbol{f(x_1,x_2,\ldots,x_n)}$ některé proměnné např. $x_k, x_{k+1}, \ldots, x_n$, kterými budeme adresovat multiplexer v prvém stupni. Po vytknutí kombinací proměnných $x_k, x_{k+1}, \ldots, x_n$ získáme 2^{n-k+1} funkcí zbývajících proměnných $\boldsymbol{g_s(x_1,x_2,\ldots,x_{k-1})}$ z nichž každá přísluší jedné kombinaci rovnice (2.35).

$$f(x_1,x_2,\ldots,x_n) = \sum_{s=1}^{2^{n-k+1}} g_s(x_1,x_2,\ldots,x_{k-1}).K_s \qquad (2.35)$$

kde $K_s = x_k^o.x_{k+1}^o....x_n^o$ a $x_i^o = x_i$ nebo $\overline{x}_i$. Tím redukujeme problém návrhu funkce n proměnných na problém maximálně k-1 proměnných. Funkce $\boldsymbol{g_s(x_1,x_2,\ldots,x_{k-1})}$ potom realizujeme v druhém stupni 2^{n-k+1} multiplexery s k-2 adresovými vstupy jejichž výstupy připojíme na

TYP OBVODU			Typ	Počet MUX	Provedení	Řada TTL
TTL		CMOS				
	3 stav					
	74250		1 z 16	1		AS
	74850		1 z 16	1		AS
74151	74251	5412/3st	1 z 8	1		ALS,AS
74350	74354		1 z 8	1		LS
	74356		1 z 8	1		LS
	74357		1 z 8	1	OK	LS
74153	74253	4539	1 ze4	2		ALS,AS
74352	74353		1 ze4	2	Invert	ALS,AS
74157	74257	4519	1 ze 2	4		ALS,AS
74158	74258		1 ze 2	4	Invert	ALS,AS
	74857		1 z 6	1		ALS,AS
74398			1 ze 2	4	Reg.Y i /Y	LS
74399			1 ze 2	4	Reg. Y	LS

Tabulka 2.12

odpovídající vstupy multiplexeru v prvém stupni. Je-li k-2 velké, pokračujeme v rozkladu každé funkce $g_s(x_1, x_2, \ldots, x_{k-1})$ stejně jako v prvém stupni dokud počet proměnných rozdělených funkcí neodpovídá možnostem použitých multiplexerů. Zjednodušení vícestupňového obvodu s multiplexery spočívá v tom, že permutací proměnných prvého stupně je možné ztotožnit nebo komplementovat funkce v dalších stupních a tím zmenšit počet multiplexerů.

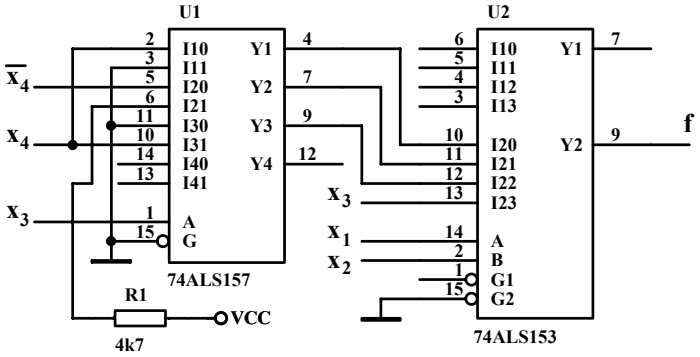
Příklad 2.16 Realizujte dvoustupňovou formu s multiplexery 74ALS153 a 74ALS157 logickou funkci, která je dána vztahem

$$f = x_1.x_2.x_3 + \bar{x}_1.x_2.x_3.x_4 + \bar{x}_1.\bar{x}_2.\bar{x}_3.x_4 + x_1.\bar{x}_2.\bar{x}_3.\bar{x}_4 + x_1.\bar{x}_2.x_3$$

Pro adresování multiplexeru v prvém stupni zvolme například proměnné x_1, x_2. Funkci f potom upravíme takto

$$f = \bar{x}_1.\bar{x}_2.(\bar{x}_3.x_4) + x_1.\bar{x}_2.(\bar{x}_3.\bar{x}_4 + x_3) + \bar{x}_1.x_2.(x_3.x_4) + x_1.x_2.(x_3)$$

$$= \bar{x}_1.\bar{x}_2.g_0(x_3, x_4) + x_1.\bar{x}_2.g_1(x_3, x_4) + \bar{x}_1.x_2.g_2(x_3, x_4) + x_1.x_2.g_3(x_3, x_4) \qquad (2.36)$$



Obr.2.20

Pro jednoduchost obvodové realizace zvolme k adresování všech multiplexerů druhého stupně proměnnou x_3. Pro jednotlivé funkce $g_s(x_3, x_4)$ můžeme psát

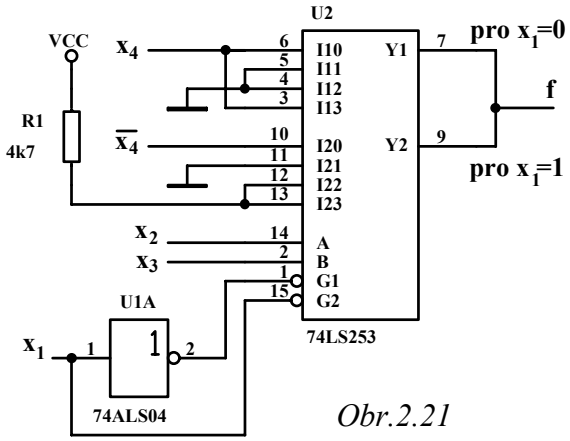
$g_0(x_3, x_4) = x_3.0 + \bar{x}_3.x_4$,
$g_1(x_3, x_4) = x_3.1 + \bar{x}_3.\bar{x}_4$,
$g_2(x_3, x_4) = x_3.x_4 + \bar{x}_3.0$ a
$g_3(x_3, x_4) = x_3.1 + \bar{x}_3.0$. Funkci $g_3(x_3, x_4)$ není nutné realizovat pomoci multiplexeru a proměnnou x_3 můžeme připojit přímo na vstup prvního stupně. Obr.2.20 zobrazuje výsledné zapojení obvodu.

Druhou již zmíněnou možností je použití multiplexerů s třístavovým výstupem. V tomto případě každý multiplexer realizuje část logické funkce pro kterou je aktivován pomocí aktivačního vstupu G, který má oproti standardním multiplexerům odlišnou funkci. Je-li G=0 obvod je aktivován a jeho výstup odpovídá stavu na adresových a datových vstupech. V opačném případě G=1 je výstup obvodu nezávisle na datových a adresových vstupech ve stavu vysoké impedance. Jestliže nyní spojíme výstupy dvou nebo více multiplexerů s třístavovým výstupem a zajistíme, že bude aktivován pro všechny kombinace vstupních proměnných jenom jeden, můžeme takto realizovat logickou funkci.

Příklad 2.17 Zrealizujte pomocí multiplexeru s třístavovým výstupem typu 74ALS253 logickou funkci čtyř proměnných z předcházejícího příkladu.

Předepsaný obvod obsahuje dva multiplexery se dvěma adresovacími vstupy, které umožňují realizovat funkce tří proměnných. Proto nejprve provedeme rozklad funkce f podle jedné proměnné, kterou budeme aktivovat jednotlivé multiplexery např. x_1.

$$f = \bar{x}_1.\left(\bar{x}_2.\bar{x}_3.x_4 + x_2.x_3.x_4\right) + x_1.\left(\bar{x}_2.\bar{x}_3.\bar{x}_4 + \bar{x}_2.x_3 + x_2.x_3\right) \qquad (2.37)$$

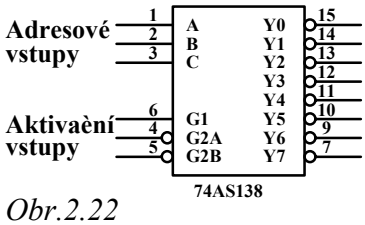


Obr.2.21

Použijeme-li nyní k adresování multiplexerů například proměnné x_2, x_3 potom snadno pro datové vstupy obou multiplexerů odvodíme $I_{10} = x_4$, $I_{11} = 0$, $I_{12} = 0$, $I_{13} = x_4$, $I_{20} = \bar{x}_4$, $I_{21} = 0$, $I_{22} = 1$ a $I_{23} = 1$. Na obr.2.21 je výsledné zapojení obvodu. K příkladu je třeba poznamenat, že řešení obvodu není zcela čisté, protože bude při změně proměnné x_1 docházet na krátký (několik ns) ke spojení obou aktivovaných výstupů.

2.3.4. Realizace LKO s dekodéry

Jednou z dalších součástek, která byla vyrobena pro jiné účely a kterou lze využít k realizaci logického kombinačního obvodu, je dekodér. Dekodér je obecně obvod, který zajišťuje převod jednoho kódu (kombinací vstupních proměnných) na kód jiný. Ve vztahu ke kódovacímu obvodu se jedná o obvod, který realizuje inverzní operaci. V integrované podobě se vyrábí několik dekodérů, které můžeme rozdělit na obvody pro řízení sedmisegmentových zobrazovacích jednotek (BIN nebo BCD → 7 segment) a dekodéry dvojkové, které realizují převod binárního čísla na kód 1 z n (BIN → 1 z n). Z hlediska realizace logických kombinačních obvodů se budeme zajímat pouze o dekodéry BIN (BCD) → kód 1 z n, které se vyrábí se 4 vstupy a 16 výstupy (74154), se 3 vstupy a 8 výstupy (74AS138) a dvakrát se 2 vstupy a 4 výstupy (74AS139). Tyto obvody se nejčastěji používají jako adresové dekodéry pro vytvoření aktivačních signálů CS (aktivace obvodu) případně OE (aktivace třístavového budiče) pro paměti nebo periferní obvody mikropočítačů.



Obr.2.22

Z hlediska kombinačních obvodů můžeme funkci dekodéru např. 74AS138 popsat vztahy (2.37) kde C,B,A jsou adresovací vstupy určující binárně hodnotu výstupu, který bude aktivní (log.0), jestliže budou splněny aktivační podmínky G1, G2A a G2B. Při aktivačních podmínkách bude každý výstup dekodéru realizovat negaci mintermu z adresovacích vstupů. Logickou

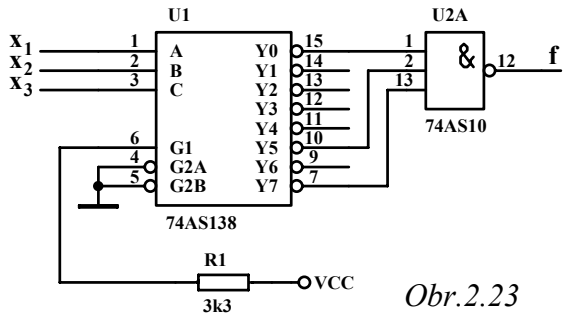
funkci ve tvaru úplné součtové formy potom můžeme realizovat jako součet potřebných

$$\begin{aligned} Y_0 &= A + B + C + \overline{G1} + G2A + G2B \\ Y_1 &= \overline{A} + B + C + \overline{G1} + G2A + G2B \\ &\dots \\ Y_7 &= \overline{A} + \overline{B} + \overline{C} + \overline{G1} + G2A + G2B \end{aligned} \qquad (2.37)$$

negovaných mintermů (výstupů dekodéru).

Příklad 2.18 Zrealizujte logickou funkci tří proměnných $x_1.\overline{x}_2.x_3 + x_1.x_2.x_3 + \overline{x}_1.\overline{x}_2.\overline{x}_3$.

Na výstupech dekodéru 74AS138 za předpokladu, že vstupní proměnné připojíme na adresovací vstupy a zaktivujeme obvod (G1=1, G2A=G2B=0), budou všechny negované mintermy vstupních proměnných (adresovacích proměnných), z kterých vždy jen jeden bude nulový. Realizace kombinačního obvodu potom spočívá v součtu případů (kombinací vstupních proměnných), pro které má být funkční hodnota rovna 1. V našem případě pro adresování $C = x_3$, $B = x_2$ a $A = x_1$ vytvoříme funkci jako součet výstupů $f = \overline{Y0} + \overline{Y5} + \overline{Y7}$ obr.2.23. Tento obvod můžeme chápat jako předstupeň k realizaci LKO pomocí pamětí PROM.



Obr.2.23

2.4. Hazardní stavy v logických kombinačních obvodech

Logické obvody, které realizují určité logické funkce, mohou vlivem zpoždění signálu v jednotlivých logických členech po přechodnou dobu vykazovat na výstupech jiné hodnoty, než odpovídá modelovaným funkcím. Takovou to situaci označujeme jako **hazard.** V logických kombinačních obvodech rozlišujeme tři typy hazardů: statický, dynamický a souběhový.

Statický hazard vykazuje ten obvod, u kterého při přechodu mezi dvěma sousedními stavy vstupních proměnných (stavy, které se liší v hodnotě jedné proměnné) se stejnou hodnotou výstupu (log.0 nebo log.1) dochází na přechodnou dobu ke změně předepsané výstupní hodnoty. Předpokládejme, že dva stavy se liší ve vstupní proměnné x_i, potom funkční hodnotu vyjádřenou součtovou nebo součinovou formou můžeme vyjádřit takto

$$y = \overline{x}_i.f_i(0) + x_i.f_i(1) \qquad (2.38)$$

$$y = (\overline{x}_i + f_i(0)).(x_i + f_i(1)) \qquad (2\,39)$$

Předpokládejme, že v rovnici (2.38) pro součtovou formu je $f_i(0) = f_i(1) = 1$**,** potom pro dva vstupní stavy, které se liší pouze v proměnné x_i musí platit

$$y = \overline{x}_i + x_i \qquad (2.40)$$

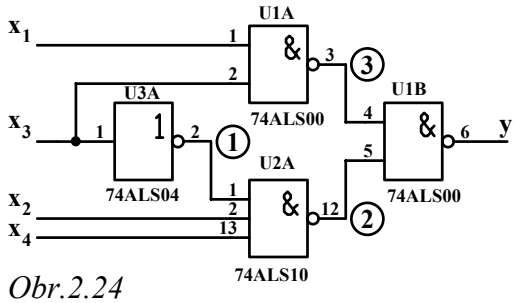
Nyní je zřejmé, že nutnou a postačující podmínkou k tomu, aby obvod vykazoval statický hazard je, aby během přechodného stavu platilo

$$y = \overline{x}_i + x_i = 0 \qquad (2.41)$$

Obdobně pro součinovou formu, za předpokladu $f_i(0) = f_i(1) = 0$, vznikne statický hazard jestliže platí

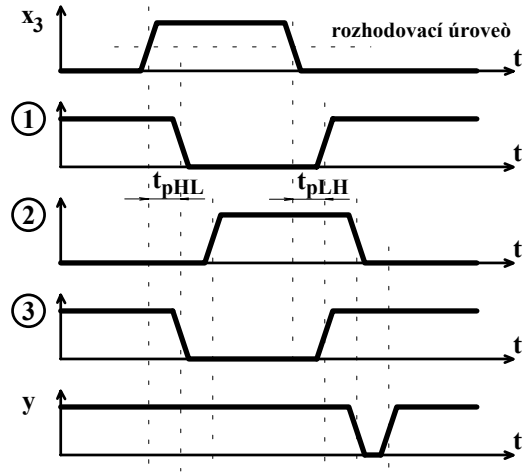
$$y = \overline{x}_i . x_i = 1 \qquad (2.42)$$

Příklad 2.19 Mějme obvod, který realizuje logickou funkci $y = x_1.x_3 + x_2.\overline{x}_3.x_4$ pomocí logických členů NAND obr.2.24, z nichž každý má zpoždění t_p.

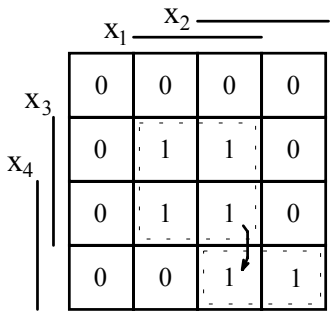


Obr.2.24

Použijeme-li již odvozené vztahy, potom ke statickému hazardu v realizovaném obvodu musí docházet při změně vstupní proměnné x_3 při vstupních proměnných $x_1 = x_2 = x_4 = 1$. Z obr.2.25 je zřejmé jak ke statickému hazardu v jedničce vlivem konečného zpoždění v logických členech dochází. Statický hazard lze stanovit nejen použitím rovnice (2.41) nebo (2.42), ale také přímo z Kar-



Obr.2.25



Obr.2.26

naughovy mapy. Jestliže zapíšeme funkci f do mapy obr.2.26 vidíme, že statický hazard může vzniknout tehdy, jestliže sousední jednotková (nebo nulová) políčka jsou pokryta různými výrazy. Minterm funkce $x_1.x_2.x_3.x_4$ je pokryt výrazem $x_1.x_3$ a minterm $x_1.x_2.\overline{x}_3.x_4$ výrazem $x_2.\overline{x}_3.x_4$. Odtud zjistíme, že ke statickému hazardu bude docházet při změně té proměnné, v které se obě sousední políčka liší tj. proměnné x_3 při $x_1 = x_2 = x_4 = 1$. Jeho maximální délku lze odhadnout z časových parametrů použitých obvodů, ale skutečná šířka se bude měnit s teplotou, napájením, zatížením, atd.

Způsob odstraňování hazardních stavů je zřejmý z předcházejícího textu. Hazard bude potlačen, jestliže kritická změna proměnné, která je příčinou hazardu bude pokryta konsensem složek $\overline{x}_i.f_i(0), x_i.f_i(1)$ nebo $(\overline{x}_i + f_i(0)), (x_i + f_i(1))$ nebo výrazy, které tyto konsensy pokrývají. Z toho plyne, že odstraňování hazardů nutně vede k neminimálním logickým funkcím a tudíž i k složitějším realizacím obvodu. Hazardy lze vylučovat přímo při odvozování minimálních forem funkce z mapy tím, že všechny přechody, které mohou způsobit hazard pokryjeme dalšími implikanty funkce, které by normálně nemusely být vybrány.

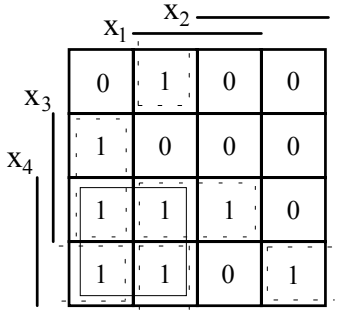
Příklad 2.20 Odvoďte minimální formu logické funkce bez statických hazardních stavů zadané Karnaughovou mapou z obr.2.27.

Pro minimální součtovou formu snadno odvodíme tento vztah

$$f_{\min} = \overline{x}_1.\overline{x}_2.x_3 + \overline{x}_1.\overline{x}_3.x_4 + x_1.\overline{x}_2.\overline{x}_3 + x_1.x_3.x_4 \qquad (2.43)$$

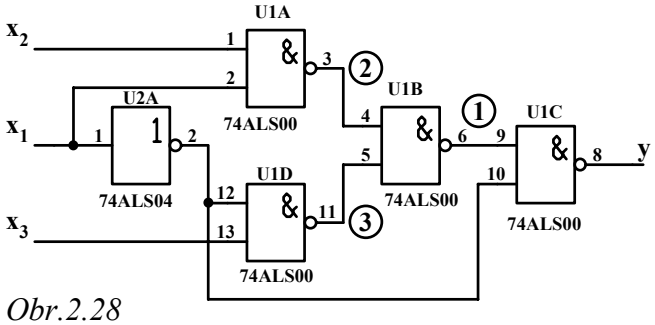
Z mapy je zřejmé, že mohou vzniknout 4 statické hazardy, které lze potlačit složkou $\overline{x}_2.x_4$. Výraz bez hazardů má pak tvar $f = f_{\min} + \overline{x}_2.x_4$. Ke stejnému výsledku se můžeme dostat i algebraickým postupem

$$\overline{x}_1.\overline{x}_2.x_4 + x_1.\overline{x}_2.x_4 + \overline{x}_2.\overline{x}_3.x_4 + \overline{x}_2.x_3.x_4 = \overline{x}_2.x_4 \qquad (2.44)$$

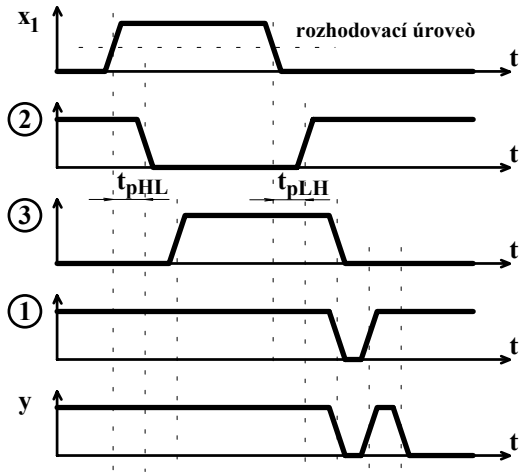


Obr.2.27

Dynamický hazard je stav obvodu kdy výstupní proměnná při přechodu z 0→1 nebo 1→0 projde posloupností stavů 0→1→0→1 nebo 1→0→1→0. Jinak řečeno místo skokové změny na výstupu obvodu se objevují zákmity dané strukturou obvodu. V obvodech s logickými členy se může vyskytnout dynamický hazard tehdy, jestliže tutéž proměnou zavádíme do obvodu přímo a v komplementu v různých stupních obvodu obr.2.28. Dynamický hazard se může vyskytnout pouze ve více něž dvoustupňových obvodech a je způsoben statickým hazardem v obvodu. Na



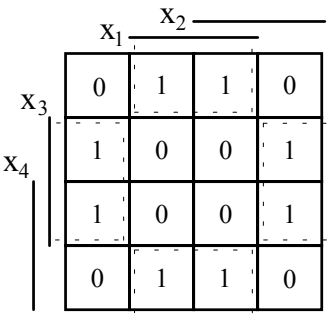
Obr.2.28



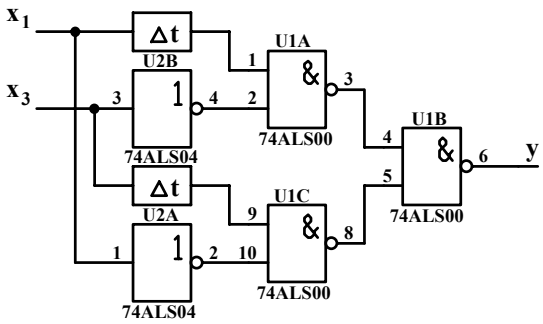
Obr.2.29

obr.2.29 jsou zobrazeny časové průběhy pro změnu vstupní proměnné x_1 při $x_2 = 1$ a $x_3 = 1$ u obvodu z obr.2.28.

Souběhový hazard je přechodný stav obvodu, který je vyvolán současnou změnou dvou nebo více vstupních proměnných, při němž výstupní signál na přechodnou dobu nabývá nesprávné hodnoty. Jedná se v podstatě o rozšíření definice statického hazardu. Prostředky potlačení těchto hazardů jsou stejné pro případy, kdy přechod mezi stavy obvodu, při němž vzniká statický hazard, lze pokrýt implikantem modelované funkce. Zvláštním případem souběhového hazardu je hazard, který nelze pokrýt konsensem obr.2.30. Takový

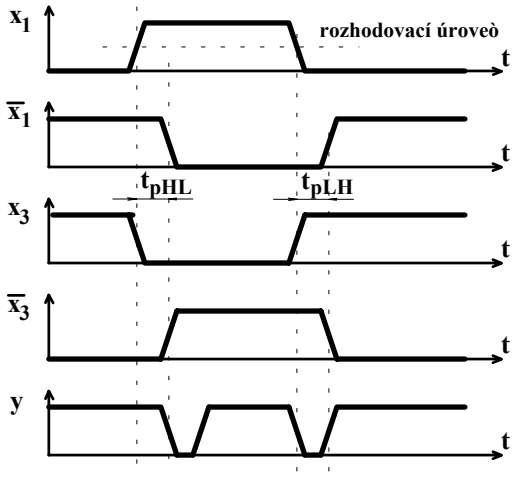


Obr.2.30



Obr.2.32

hazard je možné potlačit pouze zařazením dodatečných zpožďovacích členů přímo do kombinačního obvodu obr.2.32.



Obr.2.31

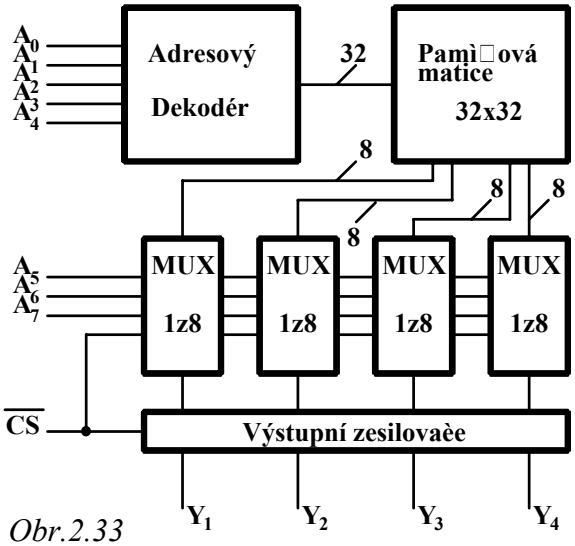
2.5. Realizace LKO programovatelnými logickými obvody

PLD (Programmable logic device) mají pro návrháře logických systémů proti obvodům SSI a MSI mnoho předností. Několik obvodů MSI/SSI může být nahrazeno jedním obvodem PLD a lze tak dosáhnout nejen požadované redukce počtu obvodů, ale i redukce jeho odběru. Programovatelnost obvodů přináší možnost menších změn obvodového zapojení desky pro změny v jejím okolí. Návrh a realizace obvodů PLD je ve srovnání s klasickým návrhem nebo návrhem hradlových polí mnohem rychlejší. Navíc u novějších obvodů existuje možnost chránit svůj obsah proti přečtení přepálením ochranné pojistky.

Paměti

Paměti PROM byly první široce používanou programovatelnou logickou skupinou. Paměť se obecně skládá ze čtyř základních částí, kterými jsou: paměťová matice, adresový dekodér pro výběr řádků paměťové matice, adresový dekodér pro výběr sloupců paměťové

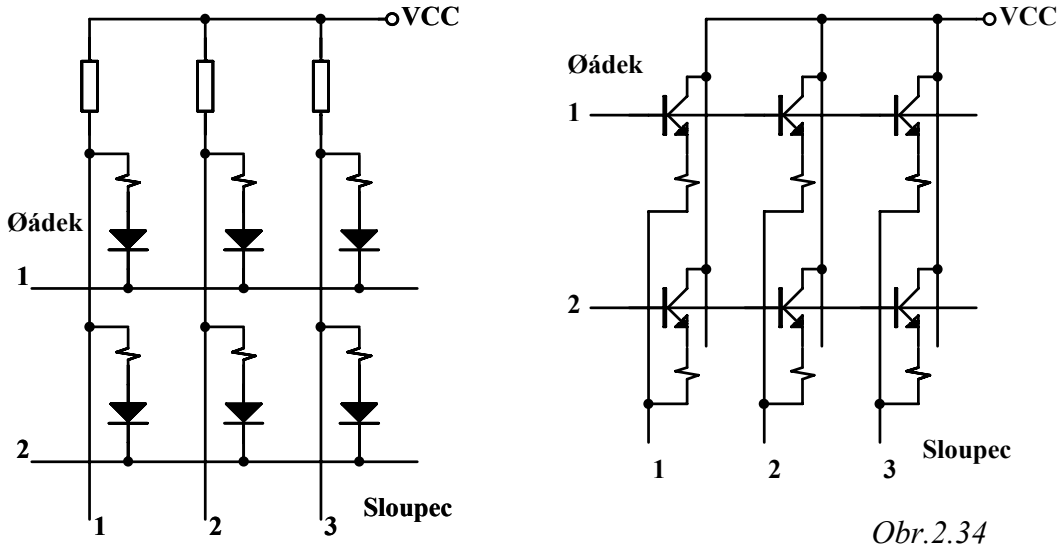
matice a bloku výstupních zesilovačů obr.2.33. Podle typu paměťové matice pak můžeme provádět další členění pamětí, nezávisle na tom jakou technologií jsou vyrobeny, na paměti ROM, PROM, EPROM a EEPROM (E^2PROM).



Obr.2.33

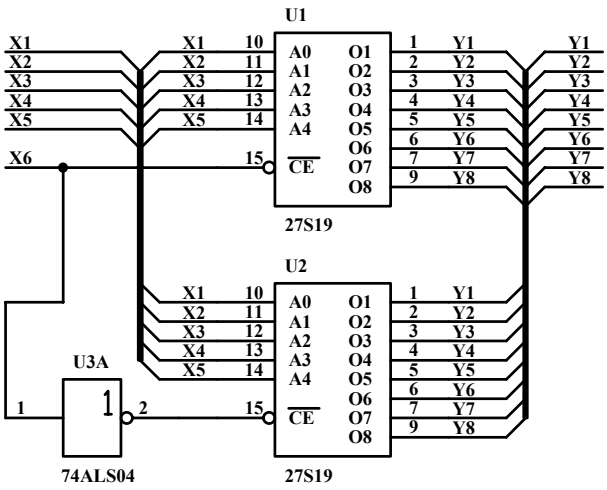
Paměti ROM (Read Only Memory), označované též jako pevné, se používají jako nositele neměnné informace. Požadovaná informace (kombinace log. 0 nebo log.1) je v nich uložena (naprogramována) ve formě přítomnosti či absence spojky mezi řádkem a sloupcem paměťové matice. Tyto paměti, které se hodí pro velkosériovou výrobu, se programují přímo u výrobce poslední maskou v procesu výroby. Nejčastěji se používají jako převodníky kódů nebo generátory znaků pro videopaměti nebo tiskárny.

Paměti PROM (Programmable ROM), které se hodí pro vývoj a malosériovou výrobu, mají od výrobce připravené všechny spojky mezi řádky a sloupci paměťové matice. Při programování jsou požadované spojky přetavovány (rozpojovány) obr.2.34. Takto je dosažen požadovaný obsah paměti, který může být změněn pouze přetavením dalších propojek. Velmi často bývaly používány k realizaci rychlých nestandardních kombinačních obvodů jako jsou dekodéry adres, převodníky kódů, atd., nyní se s nimi setkáme jako s rychlými paměťmi rozšiřující výstupy programovatelných kontrolérů kapitola 5.4.



Obr.2.34

Paměti EPROM (Erasable PROM) a EEPROM (Electrically EPROM), které se používají ve vývoji, malosériové i sériové výrobě, mají paměťovou matici vytvořenou na principu izolovaného kanálu v CMOS struktuře. Do kanálu může být uložen náboj (buňka paměťové matice je naprogramována), nebo odveden ultrafialovým zářením (EPROM) nebo elektricky (EEPROM). Tyto paměti se nejčastěji používají jako paměti programu a konstant v mikroprocesorové technice, k realizaci logických obvodů se obvykle nepoužívají.



Obr.2.35

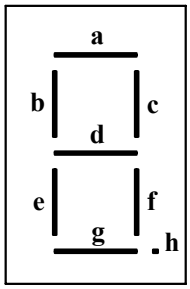
Z vnitřní struktury paměti, kterou lze znázornit obr.2.33, vyplývá, že každé kombinaci adresových vodičů jsou v paměťové matici přiděleny buňky, které lze libovolně naprogramovat. Odtud návrh logického kombinačního obvodu spočívá v přenesení pravdivostní tabulky navrhovaného obvodu do paměťové matice. Pamětí s n adresovými vstupy a m výstupy potom můžeme realizovat m logických funkcí s n vstupními proměnnými. Potřebujeme-li realizovat logickou funkci s větším počtem vstupních proměnných, než je počet adresovacích vodičů paměti, můžeme využít třístavového výstupu (nebo výstupu s otevřeným kolektorem), kterým je každá paměť s ohledem na použití v mikroprocesorové technice vybavena. Logické funkce rozložíme do více pamětí, které paralelně spojíme obr.2.35. Pro správnou činnost obvodu smí být aktivována pouze jedna paměť. Při realizacích s paměťmi musíme počítat s problémy, které vyplývají z vlastností pamětí. Každá paměť má definovánu dobu vybavení, za kterou od přivedení platné adresy nebo aktivace paměti bude na jejím výstupu platná informace. Po dobu vybavení není výrobcem garantován stav výstupů, na kterých se mohou objevit zákmity. Nevýhodou, zvláště u bipolárních pamětí PROM, je velká spotřeba ze zdroje (50 až 140mA). Obtížná je i realizace pamětí s velkým počtem adresových vodičů, protože přidáním vstupního adresového vodiče se vždy zdvojnásobí paměťová matice (propojek).

Příklad 2.31 Navrhněte logický kombinační obvod - dekodér sedmisegmentového displeje zobrazující dekadické znaky 0 až 9 a znaky A,u,r,n,t,h odpovídající hodnotám 10 až 15 vstupních proměnných.

Návrh logického kombinačního obvodu pamětí spočívá v zápisu pravdivostní tabulky navrhovaného obvodu do paměti. Pro případ dekodéru sedmisegmentového displeje, jehož segmenty si označíme podle obr.2.36, vytvoříme pravdivostní tabulku 2.13 za předpokladu, že výstup dekodéru nabývá hodnoty 0 pro svítící segment. Z katalogu zjistíme, že nám postačí paměť PROM typu MH74188 s kapacitou 32x8 bitů a s výstupy s otevřeným kolektorem (tj.

Stav	Vstupní proměnné $x_4x_3x_2x_1$	Zobrazený znak	Segment g f e d c b a
0	0 0 0 0	0	0 0 0 1 0 0 0
1	0 0 0 1	1	1 0 1 1 0 1 1
2	0 0 1 0	2	0 1 0 0 0 1 0
3	0 0 1 1	3	0 0 1 0 0 1 0
4	0 1 0 0	4	1 0 1 0 0 0 1
5	0 1 0 1	5	0 0 1 0 1 0 0
6	0 1 1 0	6	0 0 0 0 1 0 0
7	0 1 1 1	7	1 0 1 1 0 1 0
8	1 0 0 0	8	0 0 0 0 0 0 0
9	1 0 0 1	9	0 0 1 0 0 0 0
10	1 0 1 0	A	1 0 0 0 0 0 0
11	1 0 1 1	b	0 0 0 1 1 1 1
12	1 1 0 0	c	1 1 0 0 1 1 1
13	1 1 0 1	d	1 0 0 0 1 1 1
14	1 1 1 0	E	0 1 0 0 1 0 1
15	1 1 1 1	F	1 0 0 0 1 0 1

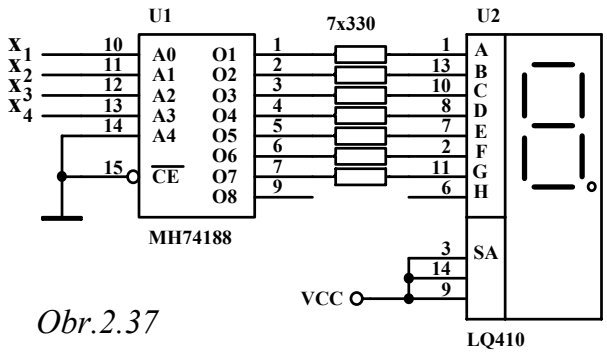
s pěti adresovacími vodiči a osmi výstupy). Protože dekodér má pouze 4 vstupní proměnné, využijeme jen polovinu kapacity paměti a nevyužijeme jeden výstup (Y_8). Výstupy aktivované paměti mají v nenaprogramovaném stavu hodnotu 0. Obsah paměti - dekodéru sedmisegmentového displeje, který do



Obr.2.36

paměti PROM naprogramujeme je dán tabulkou 2.14. Na obr.2.37 je potom zobrazeno konkrétní zapojení dekodéru se sedmisegmentovým displejem typu LQ410. Pro odpory R snadno odvodíme

$$R = \frac{U_{cc} - U_v}{\min\left(I_{F_{\max}}, I_{OL_{\max}}\right)} = \frac{5 - 1{,}6}{12.10^{-3}} = 283\Omega \qquad (2.45)$$



Obr.2.37

Z řady E12 vybereme 330Ω. Při tomto řešení je proud I jednotlivými segmenty displeje omezen maximálním proudem $I_{OL_{max}}$=12mA výstupem paměti. Pokud by bylo nutné dosáhnou většího proudu segmentem, potom je nezbytné vřadit tranzistor mezi každý výstup paměti a omezující odpor připojený na katodu segmentu.

Adresa $A_4A_3A_2A_1A_0$		Obsah $O_8O_7O_6O_5O_4O_3O_2O_1$		Adresa $A_4A_3A_2A_1A_0$		Obsah $O_8O_7O_6O_5O_4O_3O_2O_1$	
binárně	hex	binárně	hex	binárně	hex	binárně	hex
0 0 0 0 0	00	0 0 0 0 1 0 0 0	08	0 1 0 1 0	0A	0 0 0 0 0 0 0 1	01
0 0 0 0 1	01	0 1 1 0 1 1 0 1	6D	0 1 0 1 1	0B	0 1 1 1 1 0 0 0	78
0 0 0 1 0	02	0 0 1 0 0 0 1 0	22	0 1 1 0 0	0C	0 1 1 1 0 0 1 1	73
0 0 0 1 1	03	0 0 1 0 0 1 0 0	24	0 1 1 0 1	0D	0 1 1 1 0 0 0 1	71
0 0 1 0 0	04	0 1 0 1 0 1 0 1	55	0 1 1 1 0	0E	0 1 0 1 0 0 1 0	52
0 0 1 0 1	05	0 0 0 1 0 1 0 0	14	0 1 1 1 1	0F	0 1 0 1 0 0 0 1	51
0 0 1 1 0	06	0 0 0 1 0 0 0 0	10	1 0 0 0 0	10	0 0 0 0 0 0 0 0	00
0 0 1 1 1	07	0 0 1 0 1 1 0 1	2D	až			
0 1 0 0 0	08	0 0 0 0 0 0 0 0	00	1 1 1 1 1	1F	0 0 0 0 0 0 0 0	00
0 1 0 0 1	09	0 0 0 0 0 1 0 0	04				

Tabulka.2.14

Ke skutečnému dokončení realizace daného obvodu je třeba paměť PROM naprogramovat na zařízení, kterému se říká programátor. Do programátorů, které jsou v současné době řízeny počítačem, je nutné předat požadovaná data ve vhodné formě. Data lze zadávat přímo hexadecimálně na určená paměťová místa počítače řídícího programátor. Výpis obsahu paměti lze zkráceně zapsat způsobem obvyklým pro výpisy obsahů pamětí

```
0000    08,6D,22,24,55,14,10,2D,00,04,01,78,73,71,52,51
0010    00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
```

Jinou možností je přenos dat do programátoru ve formátu INTEL HEX, který produkuje většina překladačů jazyka symbolických adres (asemblerů). Ve formátu INTEL HEX, jehož struktura je popsána v dodatku A, bude obsah naší paměti zapsán takto

```
:10000000086D22245514102D0004017873715251E0
:1000100000000000000000000000000000000000E0
:00000001FF
```

Programovatelná pole

Programovatelná logická pole (PLD) nám přináší možnost výrazného snížení počtu obvodů pro danou aplikaci a jedná-li se o obvody v technologii CMOS tak i výrazné snížení odběru ze zdroje. Další výhodou při návrhu logických obvodů programovatelnými poli je vcelku dostupné a rozsáhlé programové vybavení pro počítače PC. To umožňuje návrháři vlastní návrh obvodu, minimalizaci zadaných funkcí i simulaci chování navrženého obvodu. Všechny

tyto výhody upřednostňují realizaci logických obvodů programovatelnými logickými poli. O vlastní struktuře programovatelných polí, která obsahuje řadu sekvenčních obvodů bude pojednáno souhrnně až v kapitole následující po sekvenčních obvodech. I když k realizaci kombinačního obvodů jsou použitelné skoro všechny programovatelné obvody, prakticky přicházejí do úvahy jenom některé PAL, FPGA, MACH, LCA. Přístup k návrhu logického obvodu s těmito obvody lze rozdělit do dvou cest, které dosud odpovídají základnímu dělení programovatelných obvodů. V případě PLD založených na makrobuňkách se dosud návrh realizuje z logických rovnic (i neminimalizovaných), které zapíšeme spolu s nezbytnými příkazy do vstupního souboru (ASCII editorem) pro příslušný návrhový systém. Ten provede jeho kontrolu, minimalizaci funkcí, vygenerování souboru JEDEC a umožňuje na základě simulačních příkazů kontrolu i zobrazení chování navrženého obvodu. Soubor JEDEC, který je systémem generován, a jehož struktura je popsána v dodatku B, slouží jako vstupní soubor pro programovací zařízení (programátor) těchto obvodů a přenáší v sobě informaci o buňkách, které mají být naprogramovány.

V případě druhé skupiny obvodů (FPGA a LCA), které mají výrazně přizpůsobivější architekturu a vyšší hustotu integrace, podporují jejich návrhové systémy realizace logických obvodů vycházející ze schématu nakresleného z knihoven součástek (částečně shodných s klasickou řadou TTL) v některém známém návrhovém systému jako je OrCAD, Viewlogic, atd. Tato zdánlivá výhoda příliš neusnadňuje návrháři vlastní návrh logického obvodu. Proto některé systémy podporují jako vstupní data vstupní soubory pro obvody PAL a EPLD. V současné době jsou však i u obvodů s makrobuňkami vyvíjeny systémy podporující návrh ze schématu (např. PALASM4). V této části se omezíme pouze na příklady vycházející z logických výrazů.

Příklad 2.22. Navrhněte pomocí programovatelného obvodu PAL20L8 prioritní kodér 8→3 s třístavovým výstupem ovládaným signálem OE aktivním v logické nule a dekodér 1 ze 4 s aktivačními vstupy E_1 a E_2 aktivními v log.0 a log.1..

OE	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	Pl	Y_2	Y_1	Y_0
1	X	X	X	X	X	X	X	X	z	z	z	z
0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	1	0	0	0
0	X	1	0	0	0	0	0	0	1	0	0	1
0	X	X	1	0	0	0	0	0	1	0	1	0
0	X	X	X	1	0	0	0	0	1	0	1	1
0	X	X	X	X	1	0	0	0	1	1	0	0
0	X	X	X	X	X	1	0	0	1	1	0	1
0	X	X	X	X	X	X	1	0	1	1	1	0
0	X	X	X	X	X	X	X	1	1	1	1	1

Tabulka 2.15

Prioritní kodér je obvod, který rychle vyhodnotí polohu prvního jedničkového bitu ve skupině vstupních vodičů. Polohu této jedničky indikuje binárním číslem na svých výstupech. Z hlediska

mikroprocesorové techniky bude prioritní kodér vyhodnocovat prioritu žádostí periferních obvodů o přerušení procesoru. V tabulce 2.15 je zobrazena pravdivostní tabulka navrhovaného prioritního kodéru, který je vybaven aktivačním vstupem OE pro aktivaci třístavového budiče, 8 vstupy I_0 až I_7, třemi výstupy Y_2, Y_1, Y_0, které určují binárně polohu jedničky na vstupech a výstup Pl, který úrovní log.1 určuje platnou informaci na výstupech Y_2, Y_1, Y_0. Tak je možné odlišit případ jedničky na vstupu I_0 od nepřítomnosti žádné jedničky. Z pravdivostní tabulky intuitivně odvodíme tyto vztahy

$$Y2 = I_7 + I_6 + I_5 + I_4 \qquad (2.46)$$

$$Y1 = I_7 + I_6 + (I_3 + I_2).\bar{I}_7.\bar{I}_6.\bar{I}_5.\bar{I}_4 = I_7 + I_6 + (I_3 + I_2).\bar{I}_5.\bar{I}_4 \qquad (2.47)$$

$$Y_0 = I_7 + I_5.\bar{I}_7.\bar{I}_6 + I_3.\bar{I}_7.\bar{I}_6.\bar{I}_5.\bar{I}_4 + I_1.\bar{I}_7.\bar{I}_6.\bar{I}_5.\bar{I}_4.\bar{I}_3.\bar{I}_2 = \qquad (2.48)$$
$$= I_7 + I_5.\bar{I}_6 + I_3.\bar{I}_6.\bar{I}_4 + I_1.\bar{I}_6.\bar{I}_4.\bar{I}_2$$

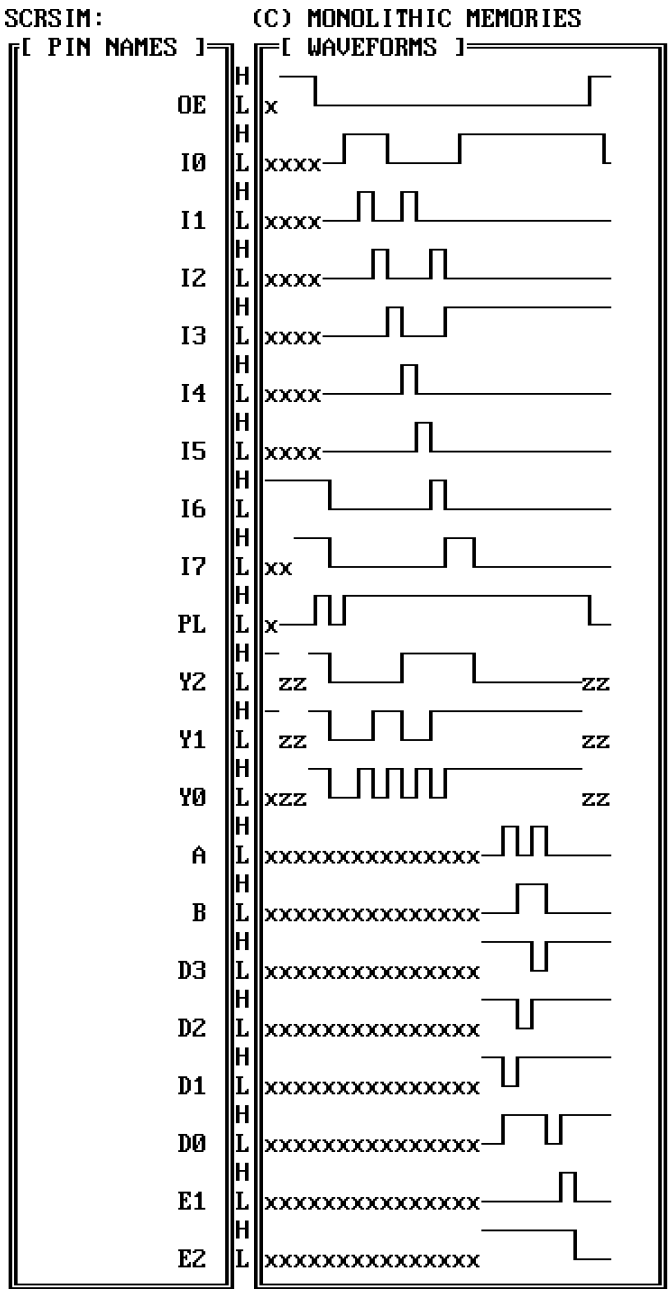
Pro výstupy dekodéru snadno odvodíme tyto vztahy

$$D0 = A + B + E1 + \overline{E2} \qquad D1 = \overline{A} + B + E1 + \overline{E2} \qquad (2.49)$$

$$D2 = A + \overline{B} + E1 + \overline{E2} \qquad D3 = \overline{A} + \overline{B} + E1 + \overline{E2} \qquad (2.50)$$

Návrhem booleovských rovnic jsme ukončili první fázi návrhu a realizace logického obvodu s programovatelným polem. Druhou fází je vlastní návrh obvodu pomocí některého návrhového systému jako je ABEL, OPAL, PLDshell, PALASM, Tango, PLPL, CUPL, TI-ALS, XACT. Pro tuto fázi musí návrhář vytvořit vstupní soubor ve formátu požadovaném návrhovým systémem. V závislosti na návrhovém systému může být obvod zadán booleovskými rovnicemi, které jsou nejrozšířenějším způsobem specifikace návrhu, nebo pravdivostní tabulkou, stavovými rovnicemi, schématem zapojení nebo průběhy signálů. V návrhovém systému dojde k syntaktické kontrole vstupního souboru, po které následuje převod vstupních údajů na společnou platformu. Tu obvykle tvoří logické rovnice, jestliže se jedná o PLD založený na makrobuňkách kap.6. Tato část tak umožňuje, aby různě zadané chování obvodu bylo jednotně specifikováno a bylo tak odděleno od ostatních částí návrhového procesu. Po získání logických rovnic dochází k jejich rozšíření, které je následováno procesem minimalizace. Minimalizace je v různých vývojových systémech realizována jiným algoritmem, nejčastěji se používají algoritmy PRESTO, ESPRESSO, ESPRESSO MV-II, atd. Poslední fází návrhu je implementace minimálních rovnic do programovatelného pole a vygenerování souboru JEDEC. JEDEC slouží jako universální prostředek popisu, jak realizovat vlastní programování obvodu PLD (zda nastavit či nenastavit programovatelné spínače propojovacích polí AND a OR. Vygenerováním souboru JEDEC můžeme ukončit druhou fázi návrhu. Většinou tomu tak není, protože potřebujeme ověřit chování navrženého obvodu. To umožňuje simulační část pro kterou si můžeme ve vstupním souboru připravit data. Teprve po důkladném ověření je vhodné přistoupit k poslední třetí fázi, kterou je vlastní programování obvodu. Ať už je obvod PLD vyroben v bipolární nebo unipolární technologii ať

je reprogramovatelný či nikoliv, informace ze souboru JEDEC musí být přenesena (naprogramována) do obvodu PLD. Tento proces je vykonáván prostřednictvím vhodného programátoru logických obvodů.



Obr.2.38

Kromě obecného popisu vlastností programovatelných obvodů, o kterých bude pojednáno až po probrání vlastností sekvenčních obvodů v kapitole 6, se zaměříme na přípravu vstupních dat pro návrhový systém PALASM2 (PALASM4). PALASM2 je vývojový systém široce rozšířený a je dostupný pro počítače PC, PS/2, VAX a některé další. O jeho základních vlastnostech a pravidlech vytvoření vstupního souboru pro vývojový systém je pojednáno v dodatku C těchto skript. V nejbližší době bude tento vývojový systém postupně nahrazen systémem PALASM4, který umožňuje návrh nových obvodů PALCE, MACH a podporuje návrh vycházející ze schématu logického obvodu. Na základě odvozených rovnic a dodatku C vytvoříme vstupní soubor PKODER.PDS. Na obr.2.38 jsou zobrazeny kontrolní průběhy získané simulačním programem.

Soubor PKODER.PDS

TITLE Prioritni koder

PATTERN 01.
REVISION 01_def
AUTHOR Skalický
COMPANY ČVUT FEL
DATE 9.4.1994

CHIP Pkoder PAL20L8

;PIN	1	2	3	4	5	6	7	8	9	10	11	12
	I0	I1	I2	I3	I4	I5	I6	I7	OE	A	B	GND
;PIN	13	14	15	16	17	18	19	20	21	22	23	24
	E1	E2	Y0	Y1	Y2	D0	D1	D2	D3	Pl	E3	VCC

EQUATIONS

```
Y2=I7+I6+I5+I4                          Y2.TRST=/OE
Y1=I7+I6+/I4*/I5*(I3+I2)                Y1.TRST=/OE
Y0=I7+I5*/I6+I3*/I4*/I6+I1*/I2*/I4*/I6  Y0.TRST=/OE
Pl=(I7+I6+I5+I4+I3+I2+I1+I0)*/OE

D0=A+B+E1+/E2                           D1=/A+B+E1+/E2
D2=A+/B+E1+/E2                          D3=/A+/B+E1+/E2
```

SIMULATION

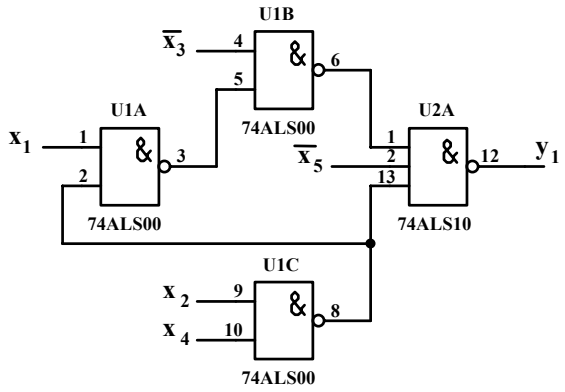
TRACE_ON OE I0 I1 I2 I3 I4 I5 I6 I7 Pl Y2 Y1 Y0 A B D3 D2 D1 D0 E1 E2

```
SETF I6                 ;Kontrola prioritního kodéru
SETF OE
SETF I7
SETF /OE
SETF /I0 /I1 /I2 /I3 /I4 /I5 /I6 /I7
SETF I0 /I1 /I2 /I3 /I4 /I5 /I6 /I7
SETF I0 I1 /I2 /I3 /I4 /I5 /I6 /I7
SETF I0 /I1 I2 /I3 /I4 /I5 /I6 /I7
SETF /I0 /I1 /I2 I3 /I4 /I5 /I6 /I7
SETF /I0 I1 /I2 /I3 I4 /I5 /I6 /I7
SETF /I0 /I1 /I2 /I3 /I4 I5 /I6 /I7
SETF /I0 /I1 I2 /I3 /I4 /I5 I6 /I7
SETF /I0 /I1 /I2 I3 /I4 /I5 /I6 I7
SETF I0
SETF /I7
SETF /A /B /E1 E2       ;Kontrola dekodéru 1 ze 4
SETF A /B /E1 E2
SETF /A B /E1 E2
SETF A B /E1 E2
SETF /A /B /E1 E2
SETF /A /B E1 E2
SETF /A /B /E1 /E2
SETF OE
SETF /I0
```

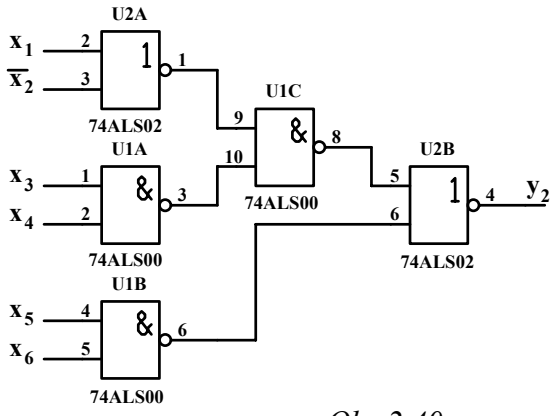
TRACE_OFF

Příklady k samostatnému řešení

Příklad 2.5.1 Odvoďte logickou funkci, kterou realizuje obvod z obr.2.39 a obr.2.40.



Obr.2.39

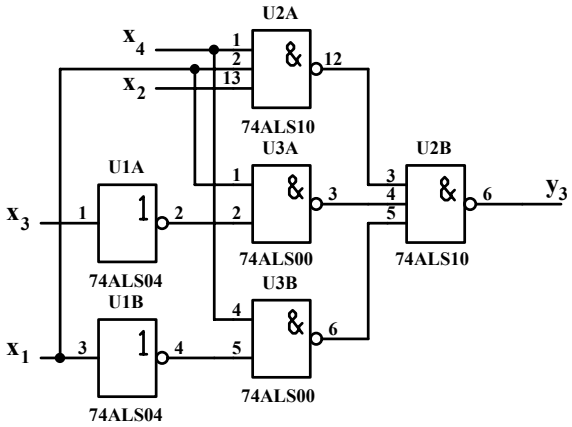


Obr.2.40

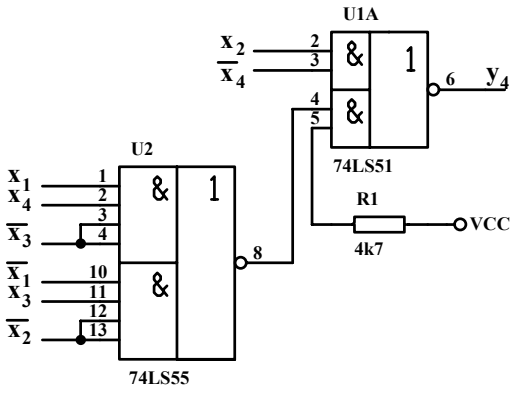
Příklad 2.5.2 Navrhněte obvod z obr.2.41 z logických členů NAND a obvod z obr.2.42 z logických členů AND-OR-INVERT tak, aby byl znemožněn vznik statického hazardu.

Příklad 2.5.3 Zjednodušte a realizujte logickými členy NAND a AND-OR-INVERT logickou funkci $y_5 = \overline{x}_1.x_2.\overline{x}_3 + x_1.\overline{x}_2.\overline{x}_3.\overline{x}_4 + \overline{x}_1.\overline{x}_2.\overline{x}_3.x_4 + x_1.\overline{x}_2.\overline{x}_3.x_4$.

Příklad 2.5.4 Logický kombinační obvod z obr.2.43 realizujte logickými členy AND-OR-INVERT typu 7451.

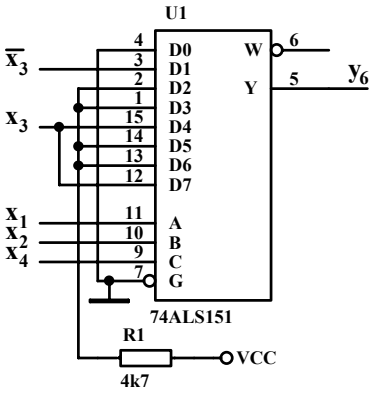


Obr.2.41

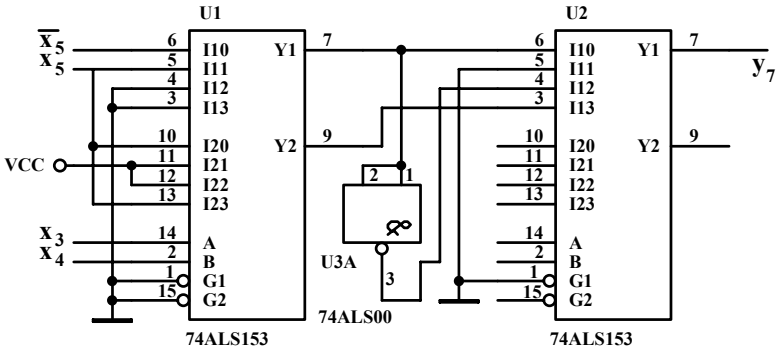


Obr.2.42

Příklad 2.5.5 Dvoustupňový kombinační obvod s multiplexery obr.2.44 realizujte dvoustupovými logickými členy NOR.



Obr.2.43



Obr.2.44

Příklad 2.5.6 Navrhněte logický kombinační obvod - dekodér 1 ze 4, který má dva adresovací vstupy A,B a čtyři výstupy O_i***, z nichž je aktivní (log.0) jen jeden za předpokladu, že vstupní proměnná CS=0. Číslo aktivního výstupu odpovídá dvojkové hodnotě přivedené na adresovací vstupy. Je-li CS=1 není aktivní žádný výstup. Obvod realizujte logickými členy NAND.***

Příklad 2.5.7 Navrhněte převodník kódu 2 z 5 typu 84210 na kód BCD8421. Převodník doplňte výstupní proměnnou C, která hodnotou 0 označuje, že na vstupu převodníku je kombinace patřící do kódu 2 z 5. Obvod realizujte pamětí PROM.

Příklad 2.5.8 Navrhněte prioritní kodér - logický kombinační obvod mající čtyři vstupy I_3, I_2, I_1 ***a*** I_0 ***a dva výstupy*** Y_1, Y_0***. Dvojková hodnota výstupů*** $Y_1 Y_0$ ***nechť udává, který vstup je roven jedné (*** $Y_1 Y_0 = 10$***, jestliže*** $I_2 = 1$***, atd.). Jestliže dva nebo více vstupů je rovno jedné, nechť hodnota výstupu je rovna nejvyšší pořadové hodnotě i vstupu*** $I_i = 1$***. Obvod realizujte logickými členy NAND.***

Příklad 2.5.9 Navrhněte a realizujte pomocí multiplexerů se dvěma adresovacími vstupy obvod indikující hodnotu 1 přítomnost číslic 0 až 9 vyjádřených v Johansonově kódu.

3. Aplikace logických kombinačních obvodů

3.1. Sčítání

V kapitole 1 byl popsán princip sčítání dvou binárních čísel, který lze popsat rovnicí (1.24) s tím, že operaci prováděnou v jednom řádu popisuje vztah (1.25). Základní logický kombinační obvod, který tuto rovnici realizuje, se nazývá **úplná binární sčítačka**. Jeho pravdivostní tabulka je dána tab.3.1. Z Karnaughových map pro funkce S_i a c_{i+1} pak snadno odvodíme vztahy (3.1) a (3.2). Logické funkce pro S_i a c_{i+1} lze realizovat například obvodem z obr.3.1 s těmito parametry pro obvody z řady LS $t_{c_{max}} = 30ns$ a $t_{s_{max}} = 52ns$. Sčítačku pro N-bitová čísla můžeme vytvořit kaskádním propojením N shodných stupňů - úplných binárních sčítaček. Na obr.3.2 je příklad pro N=4, který se vyrábí v integrované podobě ve standardní řadě TTL jako obvod 7483.

a_i	b_i	c_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Tabulka 3.1

$$S_i = \overline{c}_i.\left(a_i.\overline{b}_i + \overline{a}_i.b_i\right) + c_i.\left(\overline{a}_i.\overline{b}_i + a_i.b_i\right) \qquad (3.1)$$

$$c_{i+1} = a_i.b_i + c_i.\left(a_i + b_i\right) = a_i.b_i + c_i.\left(a_i.\overline{b}_i + \overline{a}_i.b_i\right) \qquad (3.2)$$

Nevýhoda kaskádního řazení spočívá v postupném vytváření platných přenosových bitů c_{i+1} a proto doba potřebná k vytvoření součtu lineárně vzrůstá se vzrůstajícím počtem bitů sčítaných čísel podle vztahu

$$t_{o_{max}} = (N-1).t_{c_{max}} + Max\left(t_{c_{max}}, t_{s_{max}}\right) \qquad (3.3)$$

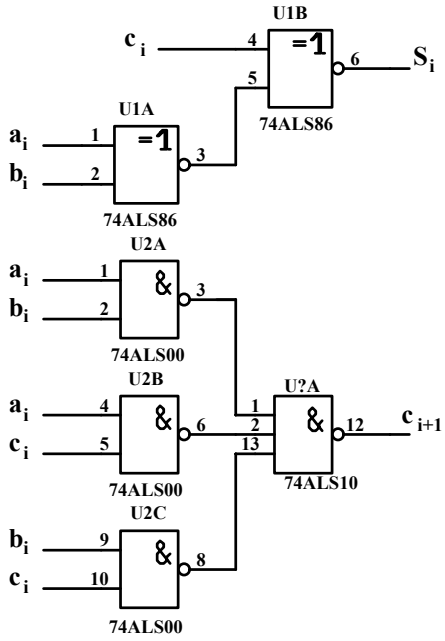
Proto se začaly vyrábět sčítačky se současným kanálem přenosu (zrychleným kanálem přenosu), který vytváří všechny potřebné přenosy najednou. Ze vztahu (3.2) snadno odvodíme tyto vztahy pro přenosy čtyřbitové sčítačky.

$$c_1 = a_0.b_0 + c_0.\left(a_0 + b_0\right) \qquad (3.4)$$

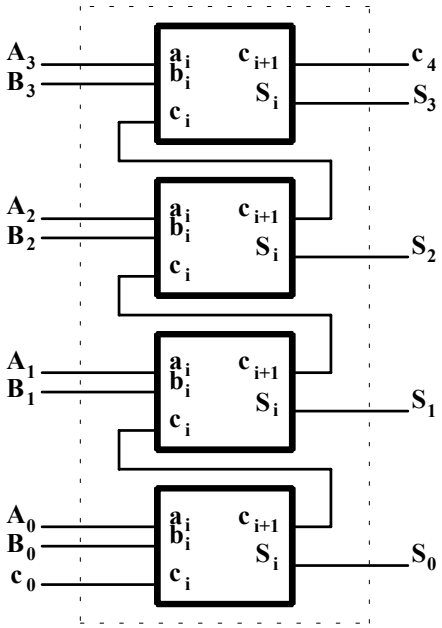
$$c_2 = a_1.b_1 + a_0.b_0.\left(a_1 + b_1\right) + c_0.\left(a_0 + b_0\right).\left(a_1 + b_1\right) \qquad (3.5)$$

$$c_3 = a_2.b_2 + a_1.b_1.\left(a_2 + b_2\right) + a_0.b_0.\left(a_1 + b_1\right).\left(a_2 + b_2\right) + c_0.\left(a_0 + b_0\right).\left(a_1 + b_1\right).\left(a_2 + b_2\right) \qquad (3.6)$$

$$c_4 = a_3.b_3 + a_2.b_2.\left(a_3 + b_3\right) + a_0.b_0.\left(a_1 + b_1\right).\left(a_2 + b_2\right).\left(a_3 + b_3\right) + a_1.b_1.\left(a_2 + b_2\right).\left(a_3 + b_3\right) + c_0.\left(a_0 + b_0\right).\left(a_1 + b_1\right).\left(a_2 + b_2\right).\left(a_3 + b_3\right) \qquad (3.7)$$

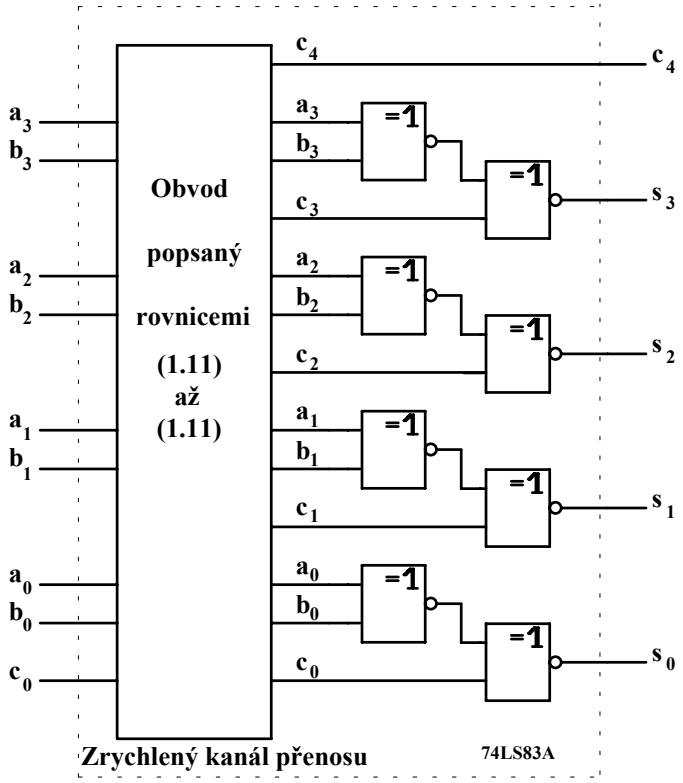


Obr.3.1



Obr.3.2

Výrazy c_0, c_1, c_2 a c_3 popisují **zrychlený kanál přenosu**, který je vhodné realizovat dvoustupňovým obvodem jako je tomu u sčítaček 74LS83A,74LS283 nebo aritmetickologické jednotky (ALU) 74181. Na obr.3.3 je symbolicky znázorněna realizace 4-bitové sčítačky se zrychleným kanálem přenosu. Jak vyplývá z rovnic (3.4) až (3.7) složitost dvoustupňové realizace zrychleného kanálu přenosu s narůstajícím počtem sčítaných bitů rychle roste. Jako mezní hodnota bývá v literatuře označována hodnota $n \leq 16$. K realizaci vícebitové sčítačky však většinou použijeme kaskádní zapojení 4-bitových sčítaček se zrychleným kanálem přenosu. V tomto zapojení je přenos mezi jednotlivými sčítačkami realizován postupně, uvnitř sčítaček je zrychlený. Pro smíšený způsob vytváření přenosu je zapojení označováno jako **hybridní sčítačka**.



Obr.3.3

Přenosy mezi čtyřbitovými sčítačkami však můžeme také realizovat dvou a více stupňovými kanály rychlého přenosu. Z rovnic (3.4) až (3.7) zjistíme, že pouze poslední člen v každé rovnici je závislý na hodnotě počátečního přenosu. Označíme-li $g_k = a_k . b_k$ a $p_k = a_k + b_k$, můžeme rovnice (3.4) až (3.7) přepsat do tvaru

$$c_1 = g_0 + c_0 . p_0 \qquad (3.8)$$

$$c_2 = g_1 + g_0 . p_1 + c_0 . p_0 . p_1 \qquad (3.9)$$

$$c_3 = g_2 + g_1 . p_2 + g_0 . p_1 . p_2 + c_0 . p_0 . p_1 . p_2 \qquad (3.10)$$

$$c_4 = g_3 + g_2 . p_3 + g_1 . p_2 . p_3 + g_0 . p_1 . p_2 . p_3 + c_0 . p_0 . p_1 . p_2 . p_3 = \qquad (3.11)$$
$$= G_0 + c_0 . P_0$$

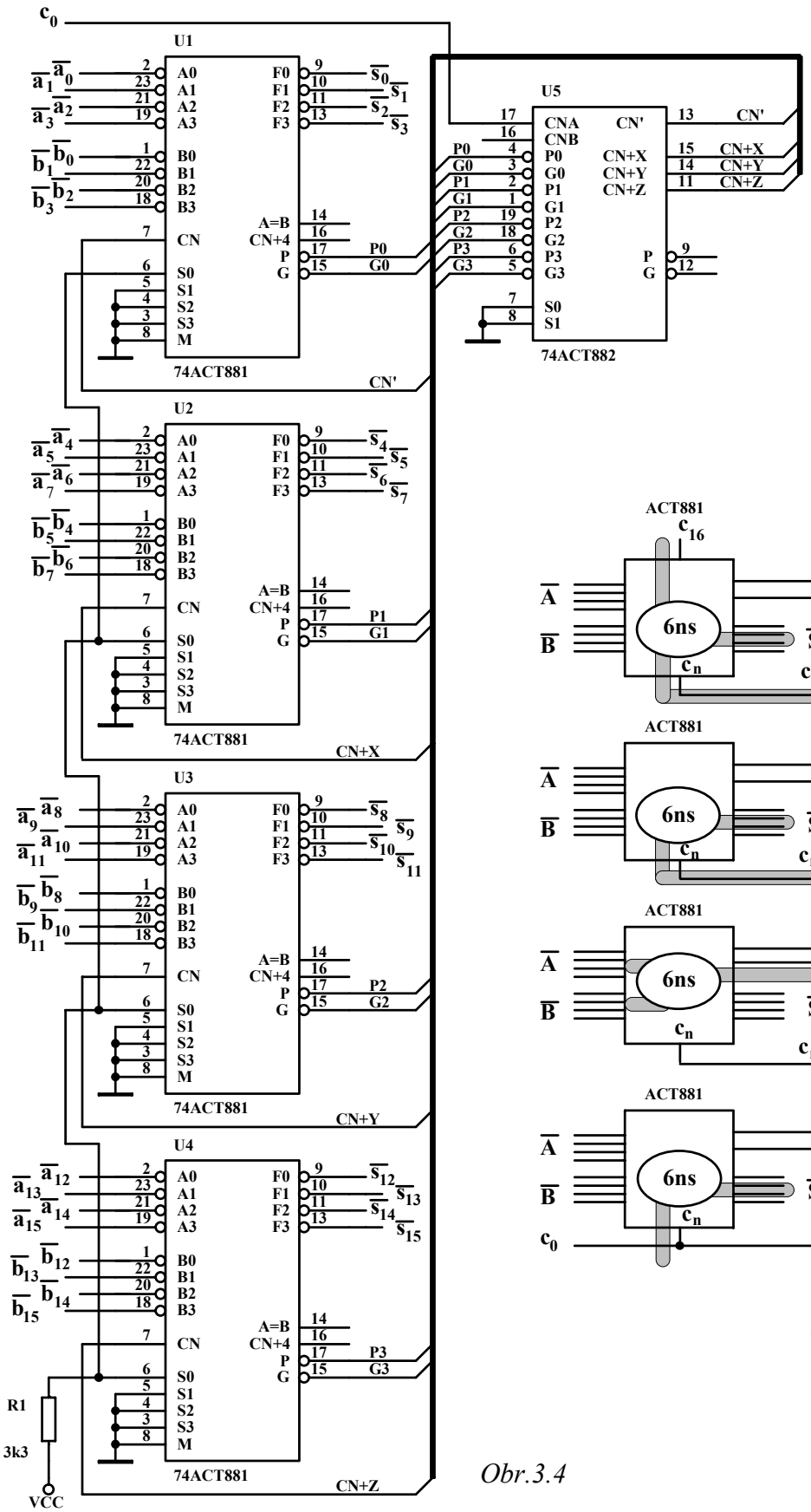
kde výrazy G_0 a P_0 jsou funkcemi pouze sčítaných bitů a_k a b_k a k=0,1,2 a 3. Obdobně můžeme pro přenos c_8 psát

$$c_8 = g_7 + g_6 . p_7 + g_5 . p_6 . p_7 + g_4 . p_5 . p_6 . p_7 + g_3 . p_4 . p_5 . p_6 . p_7 \qquad (3.12)$$
$$+ g_2 . p_3 . p_4 . p_5 . p_6 . p_7 + g_1 . p_2 . p_3 . p_4 . p_5 . p_6 . p_7 +$$
$$+ g_0 . p_1 . p_2 . p_3 . p_4 . p_5 . p_6 . p_7 + c_0 . p_0 . p_1 . p_2 . p_3 . p_4 . p_5 . p_6 . p_7 =$$
$$= G_1 + p_4 . p_5 . p_6 . p_7 . (G_0 + c_0 . P_0) = G_1 + G_0 . P_1 + c_0 . P_0 . P_1$$

kde $G_1 = g_7 + g_6 . p_7 + g_5 . p_6 . p_7 + g_4 . p_5 . p_6 . p_7$ a $P_1 = p_4 . p_5 . p_6 . p_7$. Analogicky můžeme odvodit vztahy pro c_{12} a c_{16}, z kterých zjistíme, že jsou stejné jako rovnice (3.8) až (3.11) s tím, že na-hradíme g_k hodnotou G_k a p_k hodnotou P_k. Z odvození vyplývá, že k realizaci druhého stupně zrychleného kanálu přenosu potřebujeme obvod realizující rovnice (3.8) až (3.11) a n-bitové sčítačky, které kromě obvyklých vstupů a výstupů a_i, b_i, s_i, c_0 a c_4, budou vybaveny výstupy funkcí G a P. Takovými obvody jsou 74LS181, 74LS281, 74F381, 74AS681, 74AS881 a 74AS1181. Zrychlené kanály přenosu se vyrábí jako obvody 74S182 a 74AS282, které realizují rovnice (3.8) až (3.10) a místo rovnice (3.11) realizují funkce G a P pro případné připojení třetího stupně zrychleného kanálu přerušení, a 74AS882, který realizuje přenosy $c_{n+8}, c_{n+16}, c_{n+24}$ a c_{n+32} pro 32-bitovou sčítačku. Na obr.3.4 je zobrazena 16-bitová sčítačka s dvoustupňovým kanálem zrychleného přenosu realizovaná pomocí čtyřbitových sčítaček se zrychleným kanálem přenosu 74ACT881 (první stupeň kanálu přenosu) a zrychleným kanálem přenosu 74ACT882 (druhý stupeň kanálu přenosu). Na obr.3.5 jsou zobrazeny cesty signálů nutné ke stanovení doby vytvoření potřebné k vytvoření součtu a přenosu v obvodu z obr.3.4.

3.2. Odčítání

Analogicky jako jsme postupovali při realizaci sčítání, bychom mohli odvodit pravdivostní tabulku pro úplnou binární odčítačku jako obvodu realizujícího operaci odčítání v jednom binárním řádu popsanou vztahem


c0
U1
U2
U3
U4
U5
74ACT881
74ACT882
A0
A1
A2
A3
B0
B1
B2
B3
CN
S0
S1
S2
S3
M
F0
F1
F2
F3
A=B
CN+4
P
G
CNA
CNB
CN'
CN+X
CN+Y
CN+Z
P0
G0
P1
G1
P2
G2
P3
G3
R1
3k3
VCC

Obr.3.4

ACT881
ACT882
6ns
A
B
S
cn
cn+z
cn+y
cn+x
c0
c16
P3
G3
P2
G2
P1
G1
P0
G0
P
G

Obr.3.5

$$a_i.2^i - b_i.2^i - c_i.2^i = r_i.2^i - c_{i+1}.2^{i+1} \qquad (3.13)$$

Tato cesta by vedla k vytvoření dalších integrovaných obvodů - odčítaček. Jak bylo ukázáno v kapitole 1 není nutné takový obvod vytvářet, protože odčítání můžeme realizovat jako součet dvojkového čísla s jednotkovým nebo dvojkovým doplňkem odčítaného čísla. Pro realizaci jednotkového doplňku potřebujeme invertovat všechny bity odčítaného čísla, k realizaci dvojkového doplňku musíme ještě k invertovanému číslu přičíst jedničku. Vytvoření dvojkového doplňku je možné provést kombinačním obvodem realizujícím funkcionální transformaci čísla na jeho dvojkový doplněk. Daleko častěji se vytváří z jednotkového doplňku ke kterému je přičtena jednička pomocí sčítačky. Budeme-li dvojkový doplněk využívat k operaci sčítání, můžeme přičtení jedničky k jednotkovému doplňku realizovat při vlastním sčítání tím, že zavedeme nenulový (jednotkový) počáteční přenos $c_0 = 1$.

Příklad 3.1 ***Navrhněte sčítačku a odčítačku pracující s 8-bitovými čísly X a Y (8.bit představuje znaménko) ve vyjádření dvojkovým doplňkem s pevnou desetinnou čárkou. Sčítání nebo odčítání je řízeno signálem S/O, který je roven 0 pro sčítání a 1 pro odčítání.***

V reprezentaci pomocí dvojkového doplňku zpracováváme znaménkový bit stejně jako bity významové. Tím se zjednodušuje obvodové řešení, při kterém není třeba porovnávat obě čísla pro vytvoření znaménka, což by bylo nezbytné při práci s čísly ve tvaru $\pm|A|$. Je však třeba generovat signál přeplnění pro případy, kdy výsledek přesahuje rozsah hodnot (-128,+127) zpracovávaných v systému. Při sčítání a odčítání mohou nastat tyto případy:

a) Sčítání dvou kladných čísel

Sčítačka má na vstupu operandy ve tvaru z_y, y a z_x, x, kde z_y a z_x představují znaménko (osmý bit) a x a y představují sedm významových bitů čísel X a Y. Výsledek součtu 0,x a 0,y může být

1) $0, x + 0, y = 0, s$, kde S=X+Y a $x + y \le 2^{n-1} - 1$

2) $0, x + 0, y = 1, s$, kde s je n-1 nižších řádů součtu x+y a $x + y \ge 2^{n-1}$

V druhém případě dochází k přeplnění (přetečení), protože výsledkem součtu dvou kladných čísel je číslo záporné. To je chyba, kterou je třeba indikovat.

b) Sčítání kladného a záporného čísla

Sčítačka má na vstupech operandy ve tvaru 0,x a 1,y a realizovanou operaci můžeme zapsat

$$0, x + 1, y = 0, x + \overline{\ }^1(0, y) + 1 \qquad (3.14)$$

1) pro $x > y$ tj. x=y+r (r - rozdíl) platí

$$0,x+1,y=0,y+0,r+{}^{1}(0,y)+1=0,r+2^{n}=0,r \qquad (3.15)$$

Výsledkem je kladné číslo rovné rozdílu.

2) pro $x < y$ tj. y=x+r platí

$$0,x+1,y=0,x+{}^{1}(0,x+0,r)+1=0,x+{}^{1}(0,x)+{}^{1}(0,r)+1+1= \qquad (3.16)$$
$$={}^{1}(0,r)+1$$

Výsledek odpovídá zápornému vyjádření Y-X. Ani v jednom případě nedošlo k přeplnění.

c) Sčítání dvou záporných čísel

Sčítačka zpracovává operandy ve tvaru 1,x a 1,y , pro které můžeme psát

$$1,x+1,y={}^{1}(0,x)+1+{}^{1}(0,y)+1={}^{1}(0,x+0,y)+1 \qquad (3.17)$$

Výsledek je záporná reprezentace součtu dvou kladných čísel. Pro hodnotu 0,x + 0,y přichází v úvahu tři možné případy:

1) Pro 0,x + 0,y = 0,s , kde $s \le 2^{n-1}-1$, platí

$$1,x+1,y={}^{1}(0,s)+1 \qquad (3.18)$$

Výsledek je záporná reprezentace součtu absolutních hodnot X a Y.

2) Pro 0,x + 0,y = 1,00 ... 0 platí

$$1,x+1,y={}^{1}(1,00\ldots0)+1=0,11\ldots1+1=1,00\ldots0 \qquad (3.19)$$

Limitní případ - součet X a Y se rovná hodnotě 2^{n-1}.

3) Pro 0,x + 0,y = 1,s , kde s je nižších n-1 řádů součtu X + Y, platí

$$1,x+1,y={}^{1}(1,s)+1 \qquad (3.20)$$

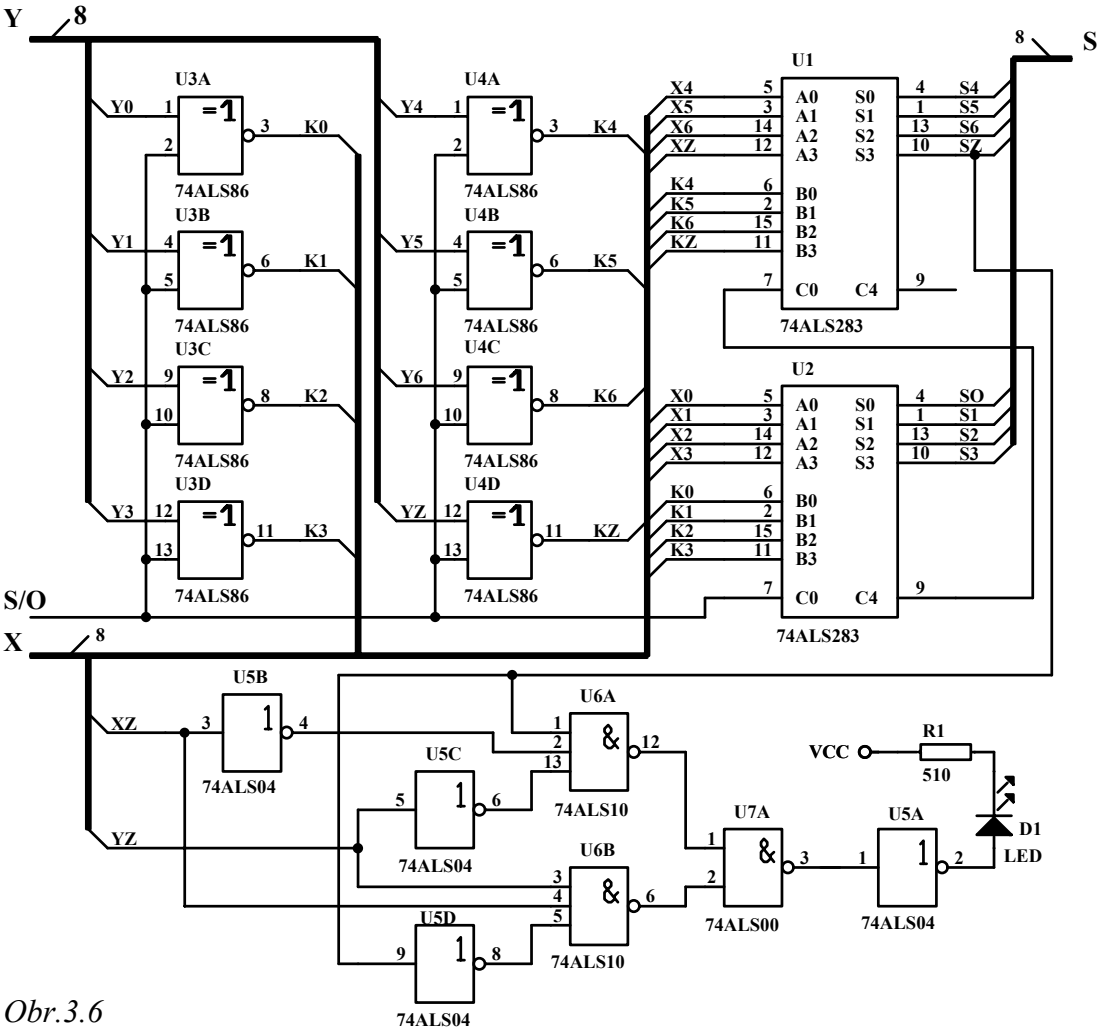
Pokud ${}^{1}(1,s)\neq 0,11\ldots1$, pak číslo 1,s patří k záporným číslům a jeho doplněk je číslo kladné. Při přeplnění (podtečení) je výsledek součtu dvou záporných čísel číslo kladné, což je chyba.

Z rozboru všech možností vyplývá, že k přetečení nebo podtečení dochází tehdy, když při součtu dvou čísel se stejným znaménkem získáme výsledek se znaménkem opačným. Pro funkci indukující chybu operace můžeme psát

$$Ch=\overline{z}_x.\overline{z}_y.z_s+z_x.z_y.\overline{z}_s \qquad (3.21)$$

kde z_x, z_y a z_s představují znaménka X,Y a součtu (nebo rozdílu). Zbývá zvolit vhodný obvod zajišťující vytvoření jednotkového doplňku v závislosti na vstupním signálu S/O. Návrh takového obvodu je velmi jednoduchý a vede na obvod EX-OR (neekvivalence) nebo EX-NOR (ekvivalence) podle polarity signálu S/O. Dvojkový doplněk vytvoříme až přímo na sčítačce zavedením nenulového počátečního přenosu pro operaci odčítání. Na obr.1.71 je výsledné

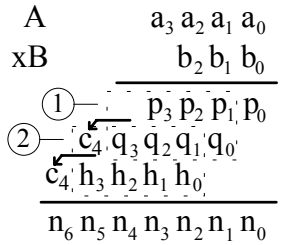
zapojení sčítačky a odčítačky pracující s čísly ve vyjádření dvojkovým doplňkem. V závislosti na signálu S/0 se na výstupech osmi obvodů EX-OR objeví číslo Y (případ sčítání) nebo jeho jednotkový doplněk (případ odčítání).



Obr.3.6

3.3. Násobení

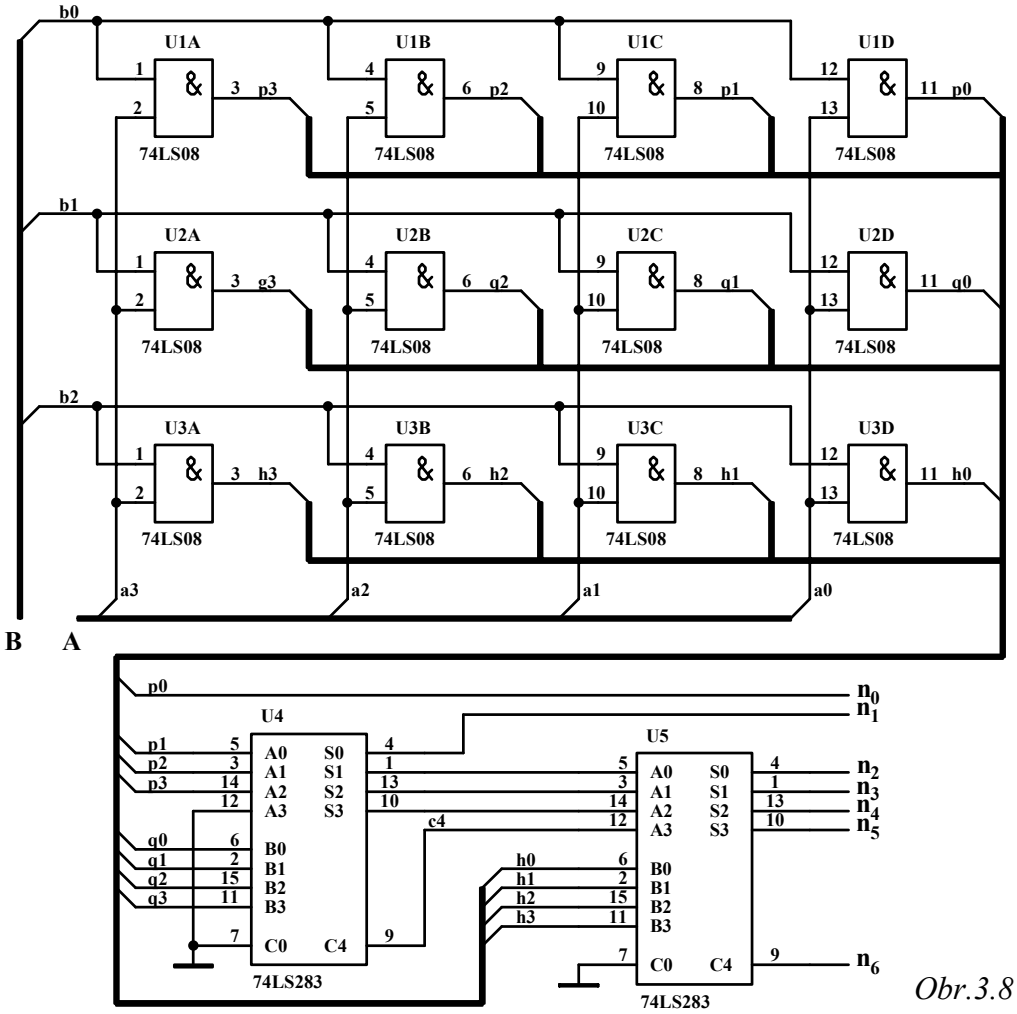
Paralelní násobičku lze realizovat kombinačním obvodem, který modeluje operaci násobení stejně jako se provádí násobení tužkou na papíře. Aritmetický součin dvou bitů je nahrazen součinem logickým, který má v tomto případě stejné vlastnosti, součty jednotlivých mezivýsledků provedeme binárními sčítačkami. Řešení takového obvodu si ukážeme na příkladu součinu čtyřbitového čísla A a tříbitového čísla B obr.3.7. Součiny p_i, q_i a h_i jsou realizovány logickými členy AND a součty ve shodě s obr.3.7 jsou realizovány sčítačkami 74LS283A. Na obr.3.8 je



Obr.3.7

zobrazena výsledná realizace násobičky. Její maximální doba potřebná k získání součinu čísel A a B je dána vztahem $t_{o_{max}} = t_{pLH_{7408}} + 2.t_{pLH(A_i \to S_i)} = 20.10^{-9} + 2.24.10^{-9} = 68\,[ns]$.

Uvedeným způsobem lze realizovat násobičku pro dvojková čísla s neomezeným počtem platných míst, složitost obvodu však rychle vzrůstá. Poněkud příznivější situace nastává při použití pamětí ROM a sčítaček. Násobená čísla A a B (např. čtyřbitová) vyjádříme ve tvaru součtu dvou nebo více čísel takto



Obr.3.8

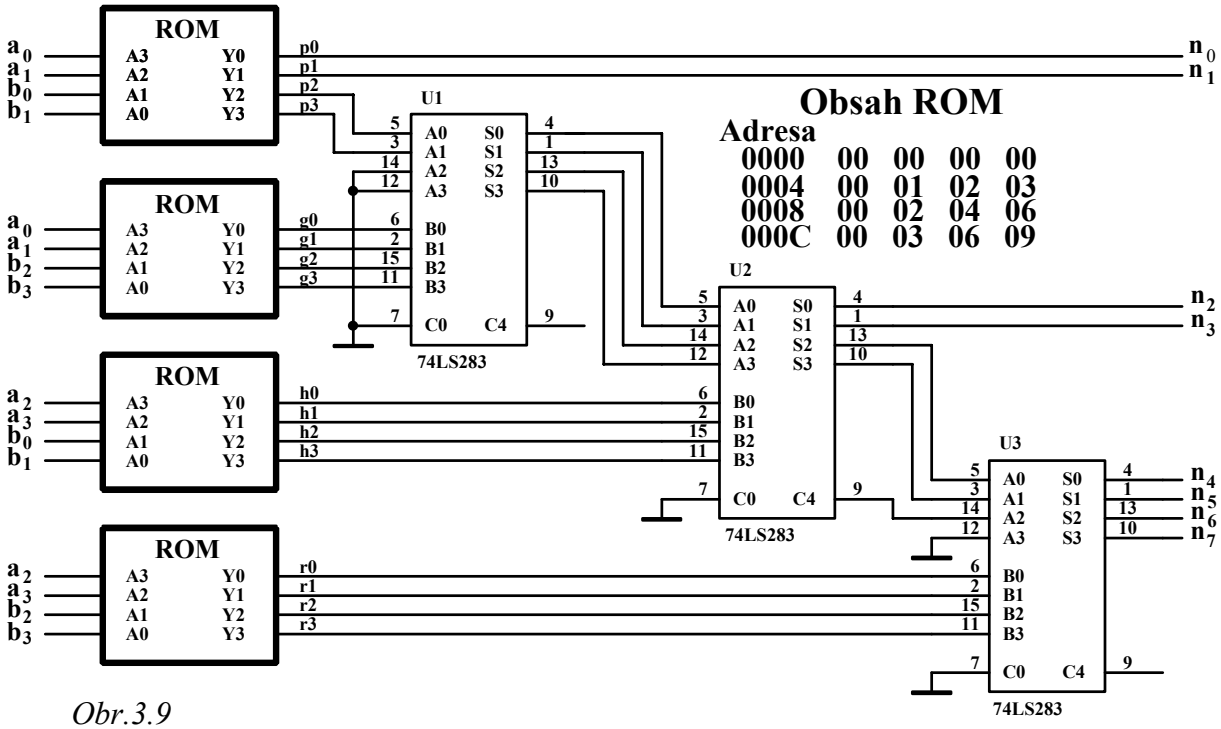
$$A = \sum_{i=0}^{3} a_i.2^i = (a_3 a_2 a_1 a_0) = (a_3 a_2 00) + (00 a_1 a_0) \qquad (3.22)$$

$$B = \sum_{i=0}^{3} b_i.2^i = (b_3 b_2 b_1 b_0) = (b_3 b_2 00) + (00 b_1 b_0) \qquad (3.23)$$

kde + nyní značí algebraický součet. Výsledný součin N čísel A a B je dán vztahem

$$N = (n_7 n_6 n_5 n_4 n_3 n_2 n_1 n_0) = (p_3 p_2 p_1 p_0) + (g_3 g_2 g_1 g_0 00) + \qquad (3.24)$$

$$+ (h_3 h_2 h_1 h_0 00) + (r_3 r_2 r_1 r_0 0000) \qquad (3.25)$$



Obr.3.9

kde $p_3p_2p_1p_0 = (a_1a_0)x(b_1b_0)$, $g_3g_2g_1g_0 = (a_1a_0)x(b_3b_2)$, $h_3h_2h_1h_0 = (a_3a_2)x(b_1b_0)$, $r_3r_2r_1r_0 = (a_3a_2)x(b_3b_2)$. Dílčí mezivýsledky (součiny) $p_3p_2p_1p_0$,, $g_3g_2g_1g_0$, $h_3h_2h_1h_0$, $r_3r_2r_1r_0$ budou uloženy v pamětech ROM, které budou adresovány vstupními proměnnými (a_1a_0) nebo (a_3a_2) a (b_1b_0) nebo (b_3b_2). Součet čtyř nebo více mezivýsledků vůči sobě vzájemně posunutých provádíme pomocí sčítaček obr.1.14. K tomuto řešení je třeba poznamenat, že stejným způsobem můžeme postupovat při programování násobení čísel v jazyce symbolických adres jejichž délka přesahuje rozsah operandů násobičky integrované v procesorové jednotce.

3.4. Porovnání čísel

Porovnání čísel je velmi častou operací v číslicové technice a tomu odpovídá i množství vyráběných integrovaných obvodů v různých variantách. Tyto obvody se používají v synchronizačních (spouštěcích) obvodech logických analyzátorů, kde zjišťují shodu mezi předem nastavenou hodnotou a vstupními signály. Často se používají místo adresových dekodérů v mikroprocesorových systémech, zvláště potřebujeme-li vytvořit jenom jeden aktivační signál, atd. Porovnávací obvody můžeme rozdělit na obvody určující rovnost nebo nerovnost vstupních proměnných (logické porovnání) nebo dvojkových čísel (aritmetické porovnání), kdy jeden bit porovnávaných čísel má funkci znaménka. Při realizaci obvodu logického porovnání můžeme vycházet přímo z pravdivostní tabulky nebo ze vztahů odvozených v kapitole 1.4.

Příklad 3.2 Navrhněte logický kombinační obvod realizující všechny funkce f_1 ***až*** f_6 ***pro porovnání dvou dvoubitových čísel pomocí logických členů NAND.***

$a_1\ a_0\ b_1\ b_0$	$f_3\ f_4\ f_5\ f_6$	$a_1\ a_0\ b_1\ b_0$	$f_3\ f_4\ f_5\ f_6$
0 0 0 0	0 1 0 1	1 0 0 0	1 1 0 0
0 0 0 1	0 0 1 1	1 0 0 1	1 1 0 0
0 0 1 0	0 0 1 1	1 0 1 0	0 1 0 1
0 0 1 1	0 0 1 1	1 0 1 1	0 0 1 1
0 1 0 0	1 1 0 0	1 1 0 0	1 1 0 0
0 1 0 1	0 1 0 1	1 1 0 1	1 1 0 0
0 1 1 0	0 0 1 1	1 1 1 0	1 1 0 0
0 1 1 1	0 0 1 1	1 1 1 1	0 1 0 1

Tabulka 3.2

Nejprve vytvoříme s ohledem na závěry z kapitoly 1.4 pravdivostní tabulku pro funkce f_3, f_4, f_5 a f_6 tab.3.2. Z Karnaughových map, které si pro funkce f_3 až f_6 napíšeme zjistíme, že pro realizaci logickými členy NAND jsou jednodušší funkce f_3 a f_5. Pro funkce f_3 a f_5.snadno odvodíme tyto výrazy

$$f_3 = a_1.\overline{b_1} + a_0.\overline{b_0}.\overline{b_1} + a_1.a_0.\overline{b_0} = a_1.\overline{b_1} + a_0.\overline{b_0}.\left(a_1 + \overline{b_1}\right) \qquad (3.25)$$

$$f_5 = \overline{a_1}.b_1 + \overline{a_0}.b_0.b_1 + \overline{a_1}.\overline{a_0}.b_0 = \overline{a_1}.b_1 + \overline{a_0}.b_0.\left(\overline{a_1} + b_1\right) \qquad (3.26)$$

Na obr.3.10 je zobrazeno konkrétní zapojení porovnávacího obvodu pro dvě dvoubitová čísla A a B.

Jak jsme již uvedli je k odvození možné použít vztahy z části 1.4. Pro funkce f_3 a f_5 můžeme psát

$$f_3 = (A > B) = \left(a_1 > b_1\right) \cup \left(a_0 = b_0\right).\left(a_0 > b_0\right) \qquad (3.27)$$

$$f_5 = (A < B) = \left(a_1 < b_1\right) \cup \left(a_0 = b_0\right).\left(a_0 < b_0\right) \qquad (3.28)$$

Logické výrazy v rovnicích (3.27) a (3.28) můžeme nyní nahradit booleovskými výrazy, které vyjadřují platnost zapsaných podmínek

$$\left(a_k > b_k\right) = a_k.\overline{b_k}, \quad \left(a_k = b_k\right) = \overline{a_k \oplus b_k} \quad \left(a_k \geq b_k\right) = a_k + \overline{b_k} \qquad (3.29)$$

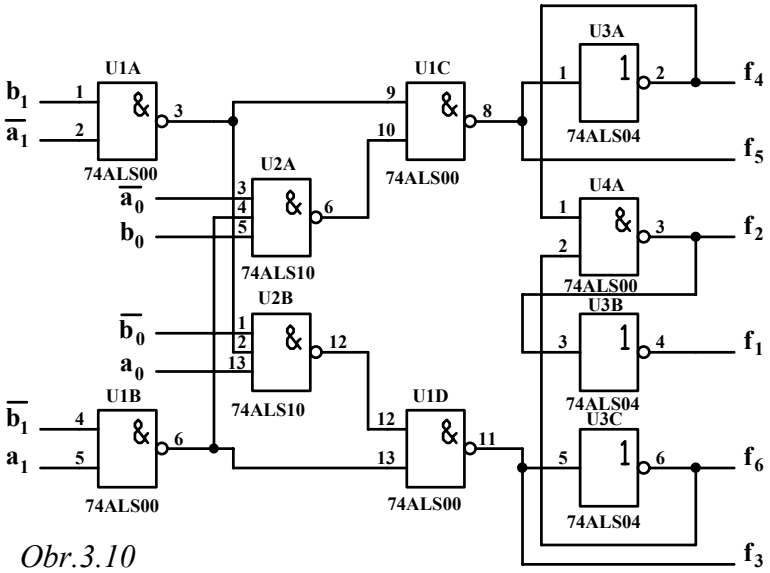
Dosadíme-li výrazy ze vztahu (3.29) do rovnic (3.27) a (3.28), potom s pomocí minimalizační metody konsensu získáme tyto závěrečné vztahy

$$f_3 = a_1.\overline{b_1} + \left(a_1.b_1 + \overline{a_1}.\overline{b_1}\right).a_0.\overline{b_0} = a_1.\overline{b_1} + a_0.\overline{b_0}.\left(a_1 + \overline{b_1}\right) \qquad (3.30)$$

$$f_5 = \overline{a_1}.b_1 + \left(a_1.b_1 + \overline{a_1}.\overline{b_1}\right).\overline{a_0}.b_0 = \overline{a_1}.b_1 + \overline{a_0}.b_0.\left(\overline{a_1} + b_1\right) \qquad (3.31)$$

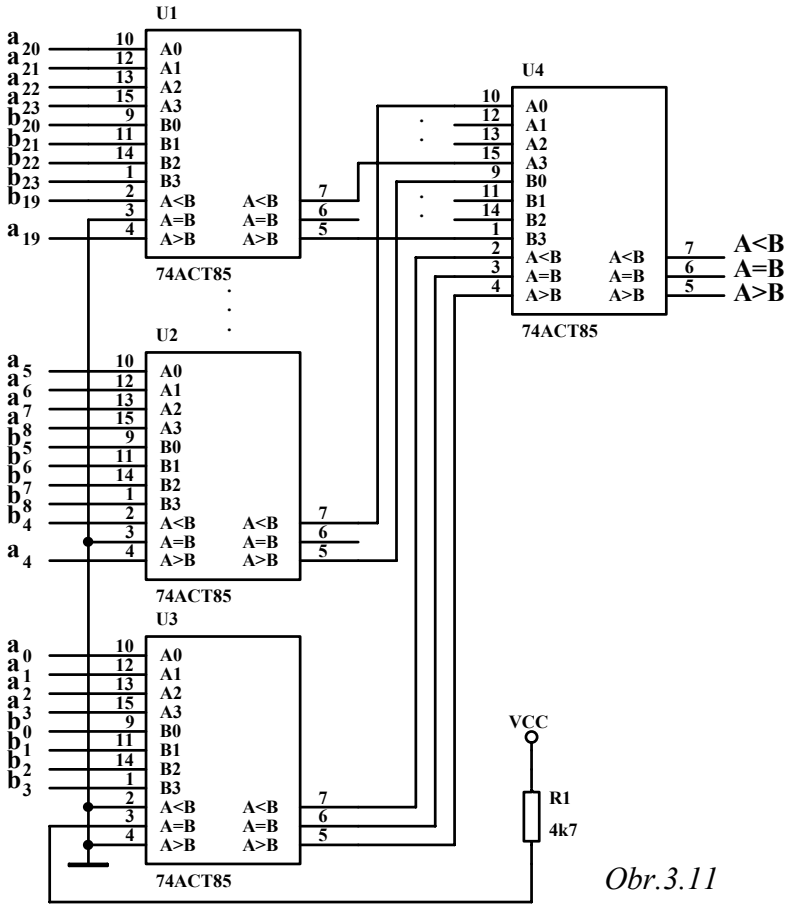
Porovnáním rovnic (3.25), (3.30) a (3.26) a (3.31) vidíme, že jsou stejné.

V současné době již při realizaci porovnávacího obvodu použijeme některý z vyráběných integrovaných obvodů, jejichž přehled je v tabulce 3.3. Některé obvody jsou vybaveny vstupy (P=Q, P<Q, P>Q) pro kaskádní řazení obvodů. Kaskádní řazení však bude vykazovat stejné nevýhody, s kterými jsme se již seznámili při sčítání čísel.



Obr.3.10

Potřebujeme-li porovnání vyhodnotit v kratší době než umožňuje kaskádní řazení, můžeme přistoupit k dvoustupňové realizaci porovnávacího obvodu. Ve druhém stupni obvodu budeme porovnávat dílčí části obou čísel např. v jednom obvodu nejnižší čtyři bity ($a_3a_2a_1a_0$) a ($b_3b_2b_1b_0$), v druhém obvodu další čtyři bity ($a_7a_6a_5a_4$) a ($b_7b_6b_5b_4$), atd. V prvém stupni obvodu potom vyhodnotíme porovnání dílčích výsledků ze stupně druhého. Na obr.3.11 je zobrazena ukázka dvou-stupňové realizace 24-bitového porovnávacího obvodu. V případě aritmetického porovnávání čísel musíme do výstupních funkcí zabudovat vliv znamének obou vstupních čísel. Jedná-li se o čísla vyjádřená v dvojkovém doplňku, můžeme použít porovnávací obvody AS885 nebo AS866, které realizují aritmetické i logické porovnání.



Obr.3.11

Obvod		Výstupy					Aktivace	Poznámka
	OK	$P = Q$	$\overline{P = Q}$	$P > Q$	$\overline{P > Q}$	$P < Q$	výstupu	
LS85		Ano	Ne	Ano	Ne	Ano	Ne	Logic
	ALS518	Ano	Ne	Ne	Ne	Ne	Ano	Logic
ALS520	ALS522	Ne	Ano	Ne	Ne	Ne	Ano	Logic
LS682	LS683	Ne	Ano	Ne	Ano	Ne	Ne	Logic
	ALS519	Ano	Ne	Ne	Ne	Ne	Ano	Logic
ALS521		Ne	Ano	Ne	Ne	Ne	Ano	Logic
LS684	LS685	Ne	Ano	Ne	Ano	Ne	Ne	Logic
LS686	LS687	Ne	Ano	Ne	Ano	Ne	Ano	Logic
ALS688	ALS689	Ne	Ano	Ne	Ne	Ne	Ano	Logic
AS885		Ne	Ne	Ano	Ne	Ano	Ano	Reg. P
AS866		Ano	Ne	Ano	Ne	Ano		Reg.P i Q

Tabulka 3.3

***Příklad 3.3** Navrhněte obvod realizující funkci* $A \geq B$ *pro devítibitová čísla se znaménkem ve formátu* $\pm|A|$.

a_8	b_8	$A \geq B$
0	0	$\|A\| \geq \|B\|$
0	1	1
1	0	0
1	1	$\|A\| \leq \|B\|$

Tabulka 3.4

Pro výslednou funkci $A \geq B$ můžeme psát pravdivostní tabulku 3.4, kde $|A|$ a $|B|$ jsou absolutní hodnoty čísel A a B vyjádřené bity 0 až 7. Funkce $|A| \geq |B|$ a $|A| \leq |B|$ můžeme vyjádřit vztahy

$$|A| \geq |B| = (|A| > |B|) \cup (|A| = |B|) \qquad (3.32)$$

$$|A| \leq |B| = (|A| < |B|) \cup (|A| = |B|) \qquad (3.33)$$

Pro výslednou funkci můžeme za pomoci rovnic (3.22) a (3.33) psát tento logický výraz

$$A \geq B = \overline{a}_8 . \overline{b}_8 . \left[(|A| > |B|) + (|A| = |B|) \right] + \qquad (3.34)$$

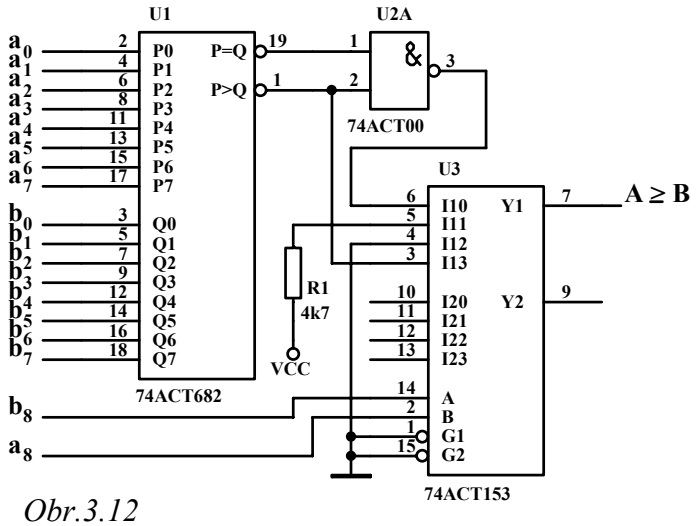
$$+ a_8 . b_8 . \left[(|A| < |B|) + (|A| = |B|) \right] + \overline{a}_8 . b_8 =$$

$$= \overline{a}_8 . \overline{b}_8 . \overline{(|A| > |B|) . (|A| = |B|)} + a_8 . b_8 . \overline{(|A| > |B|)} + \overline{a}_8 . b_8 \quad ,$$

který realizuje obvod z obr.3.12.

Mimo aplikací v kterých se obě porovnávaná čísla mohou měnit (synchronizační obvody), existují aplikace porovnávající vstupní data s pevnou neměnnou hodnotou, jako jsou komparátory nebo adresové dekodéry. V těchto případech můžeme vyjma popsaných obvodů a

již zmíněných adresových dekodérů AS138, AS139, atd. (kapitola 2.3.4) použít specializované obvody určené k tomuto účelu. Jednu skupinu tvoří programovatelné komparátory tab.3.5, u nichž je hodnota Q před použitím obvodu naprogramována na zvolenou hodnotu. Druhou skupinu tvoří adresové komparátory ALS677, ALS678 (16 adres) a ALS679, ALS680 (12 adres), jejichž vstupy P3, P2, P1 a P0 určují binárně počet nulových spodních adresovacích vodičů. Budou-li zároveň zbývající adresové vodiče v log.1 a bude platný aktivační signál, bude výstup v aktivní úrovni log.0.



Obr.3.12

Obvod	Výstup	Bitů	Vstupy	
			Programovatelné	Normální
ALS526	$\overline{P=Q}$	16	Čísla Q	Čísla P
ALS528	$\overline{P=Q}$	12	Čísla Q	Čísla P
ALS527	$\overline{P=Q}$	8	Čísla Q	Čísla P
	$P=Q$	4		Čísla P i Q

Tabulka 3.5

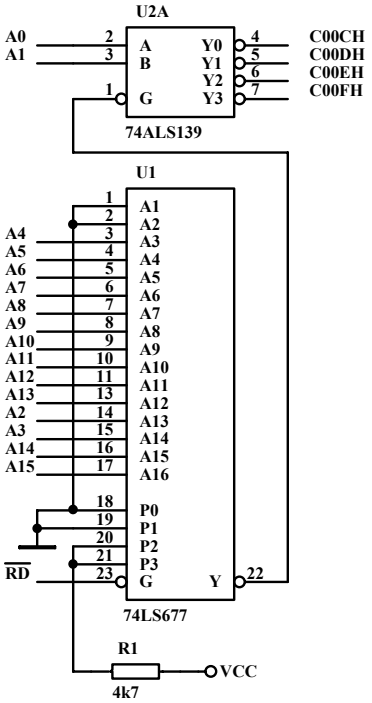
Příklad 3.4 Navrhněte adresový dekodér pro aktivaci vstupních portů s adresami C00CH, C00DH, C00EH a C00FH v adresovém prostoru datové paměti procesoru 8051.

Přímo adresovatelný prostor procesoru 8051 je 64kB a tudíž adresová sběrnice má 16 adresovacích vodičů A0 až A15. Datová paměť je ovládána řídícími signály $\overline{RD}$ (pro čtení) a $\overline{WR}$ (pro zápis). Z požadovaných adres vstupních portů vyplývá, že adresové vodiče musí nabývat hodnot

A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0
1 1 0 0 0 0 0 0 0 0 0 0 1 1 X X

kde X značí log.1 nebo log.0. Neměnné hodnoty pro všechny porty jsou na adresových vodičích A2 až A15 a proto je budeme dekódovat adresovým komparátorem. K 10 nulovým adresovým vodičům přidáme ještě 2 nevyužité vstupy adresového komparátoru a připojíme je na jeho spodní adresové vodiče. Protože předpokládáme 12 nulových vstupních hodnot, připojíme na vstupy P3,P2,P1,P0 hodnotu 1100 (12). K vytvoření 4 aktivačních signálů od

sebe odlišených adresovými vodiči A1 a A0 provedeme pomocí dekodéru 1 ze 4 typu 74ALS139 obr.3.13.



Obr.3.13

3.5. Převodníky BIN-BCD a BCD-BIN

Často používanou operací v číslicové technice je převod BCD-BIN čísla v BCD kódu na číslo binární (číslo ve dvojkové soustavě) a převod BIN-BCD čísla binárního na číslo v BCD kódu. Převod BCD-BIN, který se realizuje jednodušeji než převod BIN-BCD, je možné provádět metodami popsanými v kapitole 1. Tyto metody však nejsou vhodné pro obvodové řešení ani programové řešení na úrovni jazyka symbolických adres, lze je bez obtíží realizovat ve vyšších programovacích jazycích. Postupy, které jsou vhodnější k obvodovému i programovému řešení budeme ilustrovat na dvou příkladech.

Příklad 3.5 Navrhněte převodník čísla 0 až 99 vyjádřeného v BCD kódu na číslo binární.

Číslo v kódu BCD můžeme vyjádřit vztahem

$$N = \left(a_{13}.2^3 + a_{12}.2^2 + a_{11}.2^1 + a_{10}\right).10 + \left(a_{03}.2^3 + a_{02}.2^2 + a_{01}.2^1 + a_{00}\right) \quad (3.34)$$

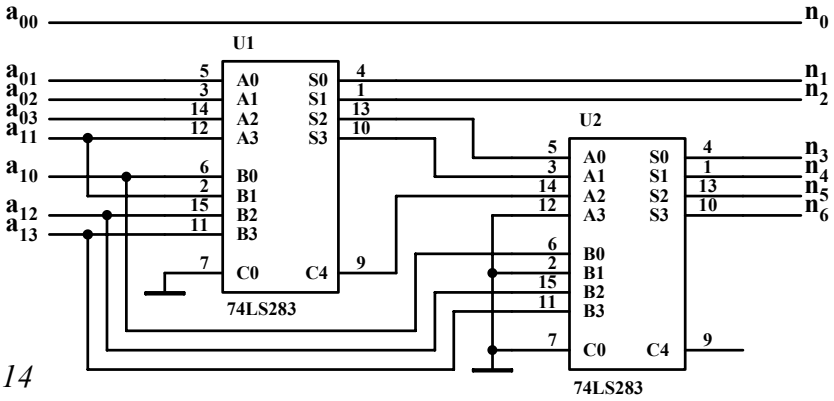
kde hodnoty $\left(a_{13}a_{12}a_{11}a_{10}\right)$ a $\left(a_{03}a_{02}a_{01}a_{00}\right)$ představují desítky a jednotky čísla v BCD kódu a znaménko + představuje aritmetický součet. Nejprve vyjádříme hodnotu 10 jako 2^3+2 a roznásobíme rovnici (3.34). Srovnáním jednotlivých výrazů podle mocnin 2^n získáme tento výraz

$$N = a_{13}.2^6 + a_{12}.2^5 + \left(a_{11} + a_{13}\right).2^4 + \left(a_{03} + a_{12} + a_{10}\right).2^3 + \quad (3.35)$$
$$+ \left(a_{11} + a_{02}\right).2^2 + \left(a_{10} + a_{01}\right).2^1 + a_{00}$$

Na obr.3.14 je nakresleno výsledné zapojení obvodu, za použití čtyřbitových sčítaček typu 74LS283.

Jednou z často používaných operací v číslicové technice je převod binárního čísla na číslo v desítkové soustavě, v které bude každá cifra vyjádřena v BCD kódu. S takovým převodem se setkáváme všude, kde naměřenou a zpracovanou hodnotu bude potřeba zobrazit na displeji přístroje. Převod binárního čísla do desítkové soustavy je možné provádět třemi metodami:

a) postupným odečítáním mocnin základu, která je vhodná pro programové řešení



Obr.3.14

b) postupného dělení základem, která je vhodná pro programové řešení, při kterém máme k dispozici operaci dělení

c) metodou využívající princip dekadické korekce

Posledně jmenovaná metoda je vhodná pro programové i obvodové řešení, které je možné realizovat jak sekvenčním tak i kombinačním obvodem. Každé binární číslo N můžeme vyjádřit vztahem

$$N = a_n.2^n + a_{n-1}.2^{n-1} + a_{n-2}.2^{n-2} + \ldots + a_1.2^1 + a_0 \qquad (3.36)$$

který snadno upravíme do tvaru

$$N = \left(\ldots\left(\left(a_n.2 + a_{n-1}\right).2 + a_{n-2}\right).2 + \ldots + a_1\right).2 + a_0 \qquad (3.37)$$

Z rovnice (3.37) vyplývá, že číslo N můžeme vytvořit postupnou sekvencí skládající se z násobení dvěma a přičtení dalšího koeficientu, který je roven 0 nebo 1. Jak vyplývá z vlastností dekadické korekce je jasné, že bude-li mezivýsledek násobený dvěma dekadické číslo, pak po jeho vynásobení a přičtení dalšího koeficientu je možné realizovat dekadickou korekci, která výsledek opět převede na dekadické číslo. Situace je analogická s realizací součtu dvou stejných BCD čísel s přenosem.

Příklad 3.6 Převeďte číslo 00110100 na číslo dekadické v BCD kódu.

							a_6	
0	0	0	0	0	0	0	**0**	Počáteční stav
							a_5	
0	0	0	0	0	0	**0**	**1**	1.krok $a_6.2 + a_5$
0	0	0	0	0	0	**0**	**1**	dekadická korekce
							a_4	
0	0	0	0	0	**0**	**1**	**1**	2.krok $(a_6.2 + a_5).2 + a_4$

0	0	0	0	0	**0**	**1**	**1**	dekadická korekce
							a_3	
0	0	0	0	**0**	**1**	**1**	**0**	3.krok $(.....).2+a_3$
0	0	0	0	**0**	**1**	**1**	**0**	dekadická korekce
							a_2	
0	0	0	**0**	**1**	**1**	**0**	**1**	4.krok $(.....).2+a_2$
0	0	0	**1**	**0**	**0**	**1**	**1**	dekadická korekce
							a_1	
0	0	**1**	**0**	**0**	**1**	**1**	**0**	5.krok $(.....).2+a_1$
0	0	**1**	**0**	**0**	**1**	**1**	**0**	dekadická korekce
							a_0	
0	**1**	**0**	**0**	**1**	**1**	**0**	**0**	6.krok $(.....).2+a_0$
0	**1**	**0**	**1**	**0**	**0**	**1**	**0**	dekadická korekce
	5				**2**			**Výsledek**

Tabulka 3.6

Algoritmus popsaný tabulkou 3.6 lze nejsnáze realizovat pomocí programu v jazyce symbolických adres pro příslušný mikroprocesor nebo pomocí kaskády sekvenčních obvodů, které modelují operace prováděné v jedné dekádě (dekadickou korekci + násobení dvěma). Obvod lze realizovat i paralelní strukturou vytvořenou z obvodů realizujících operaci v jedné dekádě a trojúhelníkově se rozšiřující v závislosti na počtu nutných kroků a délce převedeného čísla viz. tab.3.6.

3.6. Kódování

Kromě zobrazení informace a pohodlného vstupu veličin do číslicových systémů existuje mnoho důvodů, které vedly k vytvoření řady kódů. Obecně můžeme dvojkové kódy rozdělit na lineární, které se většinou využívají k zabezpečení přenášené informace proti chybám (poruchám) odposlechu atd. a na kódy lineární i nelineární, které vznikly v minulosti při řešení některých konkrétních problémů v číslicové technice.

Kódování je funkcionální transformace, která popisuje jednoznačné přiřazení vektoru $(v_1, v_2, \ldots, v_n)$ prvku (stavu) množiny A. Jestliže je v_i dvoustavová proměnná ($v_i \in \{0,1\}$) potom množina A musí mít počet prvků $\text{card}(A) \leq 2^n$. Prvky množiny A mohou být např. pole Karnaughovy mapy, stavy stavového diagramu, dekadické cifry, kombinace vstupních proměnných, atd. Z velkého množství kódů se omezíme pouze na některé desítkové a šestnáctkové kódy konstatní délky $n \geq 4$. V tabulce 3.7 jsou dekadická čísla 0 až 9 (prvky množiny A) vyjádřené některými dvojkovými kódy.

Prvek	BCD8421	BCD8421+3	Gray+3	2 z 5	Johanson
	v_4 v_3 v_2 v_1	v_4 v_3 v_2 v_1	v_4 v_3 v_2 v_1	v_5 v_4 v_3 v_2 v_1	v_5 v_4 v_3 v_2 v_1
0	0 0 0 0	0 0 1 1	0 0 1 0	1 1 0 0 0	0 0 0 0 0
1	0 0 0 1	0 1 0 0	0 1 1 0	0 0 0 1 1	0 0 0 0 1
2	0 0 1 0	0 1 0 1	0 1 1 1	0 0 1 0 1	0 0 0 1 1
3	0 0 1 1	0 1 1 0	0 1 0 1	0 0 1 1 0	0 0 1 1 1
4	0 1 0 0	0 1 1 1	0 1 0 0	0 1 0 0 1	0 1 1 1 1
5	0 1 0 1	1 0 0 0	1 1 0 0	0 1 0 1 0	1 1 1 1 1
6	0 1 1 0	1 0 0 1	1 1 0 1	0 1 1 0 0	1 1 1 1 0
7	0 1 1 1	1 0 1 0	1 1 1 1	1 0 0 0 1	1 1 1 0 0
8	1 0 0 0	1 0 1 1	1 1 1 0	1 0 0 1 0	1 1 0 0 0
9	1 0 0 1	1 1 0 0	1 0 1 0	1 0 1 0 0	1 0 0 0 0

Tabulka 3.7a

Kódovací nebo dekódovací obvod má počet vstupních a výstupních proměnných určen počtem bitů vstupující informace a délkou (počtem bitů) kódového nebo dekódovaného slova. Jedná se o realizaci většinou jednoduchého logického kombinačního obvodu jehož funkce jsou popsány tabulkou podobnou tab.3.7. Popsané kódy mají tyto význačné vlastnosti.

Grayův kód

Kód se vyznačuje tím, že sousední stavy se liší pouze v jedné proměnné. Tuto vlastnost splňuje první i poslední stav (kód je uzavřen sám do sebe). Používá se ke kódování Karnaughovy mapy, ke snímání úhlového natočení otáčejícího se hřídele a tam, kde je třeba zabezpečit (např. snímání mechanických kontaktů), aby chyba odečtu nebyla větší než nejméně významný bit (nejmenší krok). Z tabulky 3.7 snadno zjistíme, že převodní vztahy mezi binárním a Grayovým kódem jsou dány těmito vztahy (např. pro n=4).

$$g_4 = b_4 \quad g_3 = b_4 \oplus b_3 \quad g_2 = b_3 \oplus b_2 \quad g_1 = b_2 \oplus b_1 \qquad (3.38)$$

$$b_4 = g_4 \quad b_3 = g_4 \oplus g_3 \quad b_2 = g_4 \oplus g_3 \oplus g_2 \quad b_1 = g_4 \oplus g_3 \oplus g_2 \oplus g_1 \qquad (3.39)$$

Grayův kód +3 se vyznačuje stejnými vlastnostmi jako kód Grayův pro dekadické stavy 0 až 9 (stav 0 je sousední ke stavu 9). Kód **BCD8421+3 a Aikenův** byl využíván v některých počítačích pro snazší realizaci dekadických operací. U obou kódů bylo využíváno komplementárního vztahu mezi čísly 0-9, 1-8, 2-7, atd.

Dosud popsané kódy nejsou žádným způsobem zabezpečeny pro přenos. Pokud je jeden bit v přenášeném slově při přenosu poškozen chybou, nové slovo je stejně pravděpodobné jako

Prvek	Gray	Aiken
	v_4 v_3 v_2 v_1	v_4 v_3 v_2 v_1
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 1	0 0 1 0
3	0 0 1 0	0 0 1 1
4	0 1 1 0	0 1 0 0
5	0 1 1 1	1 0 1 1
6	0 1 0 1	1 1 0 0
7	0 1 0 0	1 1 0 1
8	1 1 0 0	1 1 1 0
9	1 1 0 1	1 1 1 1
10	1 1 1 1	
11	1 1 1 0	
12	1 0 1 0	
13	1 0 1 1	
14	1 0 0 1	
15	1 0 0 0	

Tabulka 3.7b

slovo původní. Jestliže použijeme ke kódování informace menší počet kombinací, existuje jistá pravděpodobnost, že nová kombinace nebude patřit do kódu a může být identifikována jako chybná.

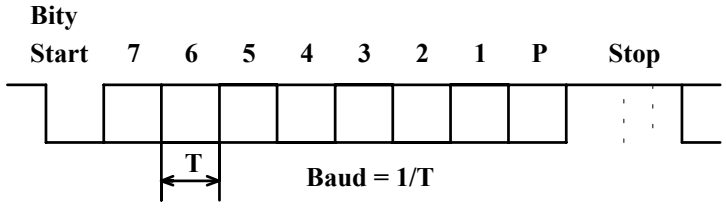
Johansonův kód má stejné vlastnosti jako kód Grayův + 3 a zároveň umožňuje identifikaci cca 60% jednoduchých chyb (tj. chyb způsobených změn jednoho bitu). **Kód 2 z 5** se využívá v telefonních ústřednách a umožňuje identifikovat všechny jednoduché chyby a cca 40% chyb dvojnásobných.

Telegrafní kódy

Od zavedení telegrafu bylo zapotřebí kódovat přenášenou informaci mezi vzdálenými místy. Přenos se proto uskutečňuje v sériovém tvaru, který potřebuje pouze dva přenosové vodiče. Standardními přenosovými kódy se staly mezi-národní telegrafní kód číslo 2 (nyní se již nepoužívá) a číslo 5 (označovaný ASCII).

U **telegrafního kódu číslo 2** je přenášená informace dána 5 bity (5 intervaly), které jsou uvedeny počátečním bitem (start bitem=log.0) a ukončeny koncovým bitem (stop bitem=log.1). Přenosová rychlost byla 50 baudů, kde baud je převrácená hodnota trvání elementárního intervalu (tj. 1/0,02 = 50). Pět dvojkových prvků poskytuje pouze 32 kombinací, které nepostačují na přenos všech písmen, číslic a řídících znaků. Počet je zdvojnásoben kódy písmenová a číslicová změna, za kterými má tatáž informace různý význam.

U **telegrafního kódu číslo 5** je přenášená informace je dána sedmi bity doplněné paritním bitem (může být i vynechán). Narozdíl od předešlého kódu je to kód zabezpečený (je-li použita parita) a v současnosti se používá pro spojení počítačů



Obr.3.15

pomocí modemů nebo ke spojení terminálů s počítačem, pokud není použit některý ze standardních sériových synchronních protokolů. Analogicky jako předcházející kód je uveden nulovým start bitem a ukončen jedním, jedna a půl nebo dvěma stop bity obr.1.12. Přenosová rychlost může nabývat jedné z následujících hodnot: 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400 baudů. V příloze F je uvedeno kódování jednotlivých znaků v kódu ASCII.

3.6.1. Lineární kódy

Kódy zabezpečující přenos informace je vhodné definovat jako kódy vytvořené v n-rozměrném lineárním prostoru T, v kterém jsou definovány operace sčítání a násobení. Omezíme-li se pouze na binární kódy, pak budeme pracovat v tělese T= {0,1}, které obsahuje jenom dva prvky [14]. Operace sčítání je pak dána vztahy

$$0+0=0, \quad 0+1=1, \quad 1+0=1 \text{ a } 1+1=0 \qquad (3.40)$$

které lze v číslicové technice realizovat pomocí funkce EX-OR a budeme ji nadále označovat symbolem $\oplus$. Operace násobení je dána vztahy

$$0.0=0, \quad 0.1=0, \quad 1.0=0 \text{ a } 1.1=1 \qquad (3.41)$$

které lze realizovat pomocí funkce AND. Binárním lineárním kódem označujeme ten kód, jehož kódové slovo je tvořeno lineární kombinací bázových prvků podprostoru s dimenzí k. Má-li pak kódové slovo délku n, mluvíme o lineárním (n,k) kódu, který má k informačních znaků a n-k kontrolních znaků.

Příklad 3.7 Kód celkové kontroly parity je popsán touto rovnicí

$$v_1 \oplus v_2 \oplus v_3 \oplus \ldots \oplus v_n = 0 \qquad (3.42)$$

Bázové prvky tohoto kódu, který je lineárním kódem (n,n-1), budou dány kódovými slovy, které obsahující jenom jeden přenášený bit. Bázové prvky jsou lineárně nezávislé a popisují jak jednotlivé informační bity ovlivňují kontrolní bit a definují jak se vytváří jakékoliv kódové slovo, které do tohoto kódu patří.

$$b_1 = 1000\ldots001$$
$$b_2 = 0100\ldots001$$
$$\vdots$$
$$b_k = 0000\ldots011$$

Definice - Generující matice lineárního (n,k) kódu, kde k ≠ 0, je určena svou bází $b_1 b_2 b_3 \ldots b_k$ zapsanou pod sebe

$$G = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix} \qquad (3.43)$$

Příklady generujících matic:

a) Binární kód celkové parity, který má 3 informační bity a jeden kontrolní (paritní), je kód (4,3) s touto generující maticí.

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

b) Opakovací kód, který opakuje přenášený bit pětkrát, je kód (5,1) (tzn., že zpráva 0110 bude vyslána ve tvaru 00000111111111100000 má generující matici ve tvaru).

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

c) Lineárním kódem, o které jsme již psali v předcházející části, je Grayův kód (4,4). Z rovnic (1.12) snadno odvodíme generující matici pro tento kód.

$$G = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Zakódování k informačních bitů $u_1 u_2 \ldots u_k$ lze definovat jako zobrazení množiny všech slov délky k na množinu kódových slov $v_1, v_2, \ldots, v_n$ předpisem

$$f(u_1, u_2, \ldots, u_k) = b_1 . u_1 \oplus b_2 . u_2 \oplus b_3 u_3 \oplus \ldots \oplus b_k . u_k \qquad (3.44)$$

nebo maticově $[u_1, u_2, \ldots, u_k].[G]$. Vztah (3.44) určuje jak vytvoříme kódové slovo $v_1, v_2, \ldots, v_n$ (zakódujeme informační bity $u_1, u_2, \ldots, u_k$). Umíme-li nyní zakódovat přenášené informační bity, musíme je na přijímací straně umět dekódovat a zjistit, zda při přenosu nedošlo k chybě.

Definice - Kontrolní matice kódu je taková matice $[H]$ z prvků tělesa T, která vyhovuje následující rovnosti za předpokladu, že $v_1, v_2, \ldots, v_n$ je kódové slovo.

$$[H] . \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad (3.45)$$

U některých kódů je určení kontrolní matice velmi jednoduché, a vyplývá přímo z principu vytvoření kódu. Například uvedený kód celkové parity (4,3) kontroluje sudý počet jedniček v přenášeném (kódové slově), a proto jeho kontrolní matice bude dána vztahem $H = [1 \quad 1 \quad 1 \quad 1]$. V případech, kdy konstrukce kontrolní matice není zřejmá, můžeme využít vlastností systematických kódů, u kterých umíme snadno vytvořit kontrolní matici.

Systematický kód je takový kód (n,k), v jehož generující matici vytváří prvních k sloupců spolu s k řádky jednotkovou matici [E] a zbývající sloupce generující matice mají libovolný obsah. To můžeme maticově zapsat $[G]=[E|B]$, kde [B] je matice s n-k sloupci a k řádky. Je třeba upozornit na to, že každý kód (n,k), který není systematický, je možné pouhou permutací řádků a sloupců generující matice předělat na kód systematický.

Věta: ***Lineární kód s generující maticí*** $[G]=[E|B]$ ***(systematický kód) má kontrolní matici*** $[H]=[-B^T|E']$, ***kde*** B^T ***je transponovaná matice*** $[B]$ ***a*** $[E']$ ***je jednotková matice.***

Otázkou zůstává jaké vlastnosti má kód (n,k) z hlediska identifikace a případného odstranění chyb v přeneseném kódovém slově. Předpokládáme-li, že využíváme všechny kombinace kódových slov, potom změna kteréhokoliv bitu u kódu nezabezpečeného ani jedním kontrolním bitem není identifikovatelná (kódová vzdálenost je rovna 1). U kódu celkové parity, kde každý bit je kontrolován jedním bitem, je kódová vzdálenost rovna 2 (kódová slova se liší ve dvou bitech). Kód umožňuje identifikovat lichý počet chybných bitů (jednonásobnou chybu) a neumožňuje identifikovat sudý počet chybných bitů (dvojnásobnou chybu). Obecně potom můžeme zjistit, že kód s n-k kontrolními bity umožňuje identifikovat chyby (n-k)-násobné (kódová délka n-k+1). Zajímáme-li se o opravu chyb v přeneseném kódovém slově, pak je zřejmé, že násobné chyby nepřevyšující polovinu kódové vzdálenosti můžeme opravit. Odtud kódem s n-k kontrolními bity můžeme opravit (n-k-1)/2 násobné chyby.

Příklad 3.8 ***Odvoďte kontrolní matici Hammingova kódu (7,4), jehož 4 informační bity*** $I_4I_3I_2I_1$ ***a 3 kontrolní bity*** $P_4P_2P_1$ ***jsou uspořádány do kódového slova*** $I_4I_3I_2P_4I_1P_2P_1$. ***Paritní bit*** P_4 ***kontroluje bity*** I_4, I_3 ***a*** I_2, ***paritní bit*** P_2 ***kontroluje bity*** I_4, I_3 ***a*** I_1, ***a paritní bit*** P_1 ***kontroluje bity*** I_4, I_2 ***a*** I_1,.

Pro generující matici, která má čtyři řádky popisující ovlivnění kontrolních bitů každým informačním bitem, snadno odvodíme

$$[G]=\begin{bmatrix}1&0&0&1&0&1&1\\0&1&0&1&0&1&0\\0&0&1&1&0&0&1\\0&0&0&0&1&1&1\end{bmatrix}\Rightarrow[G']=\begin{bmatrix}1&0&0&0&1&1&1\\0&1&0&0&1&1&0\\0&0&1&0&1&0&1\\0&0&0&1&0&1&1\end{bmatrix} \qquad (3.46)$$

Záměnou čtvrtého a pátého sloupce matice [G] získáme matici [G´] kódu, který je systematický. Pro tento kód již snadno odvodíme kontrolní matici [H´], z které po prohození 4 a 5 sloupce obdržíme kontrolní matici Hammingova kódu (7,4). Předpokládejme, že přijaté slovo bude obsahovat jen jednu jedničku (např. 0000010), potom syndrom [H].[v] bude vždy nenulový (např. 010) a bude binárně udávat pozici, na které došlo k chybě. Nechť se sám čtenář přesvědčí, že tato vlastnost vyplývající z kontrolní matice platí i pro jiná nekódová slova.

$$[G'] = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \Rightarrow [G] = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \qquad (3.47)$$

Příklad 3.9 Navrhněte logický kombinační obvod pro opravu přijatého výše uvedeného Hammingova kódu (7,4)

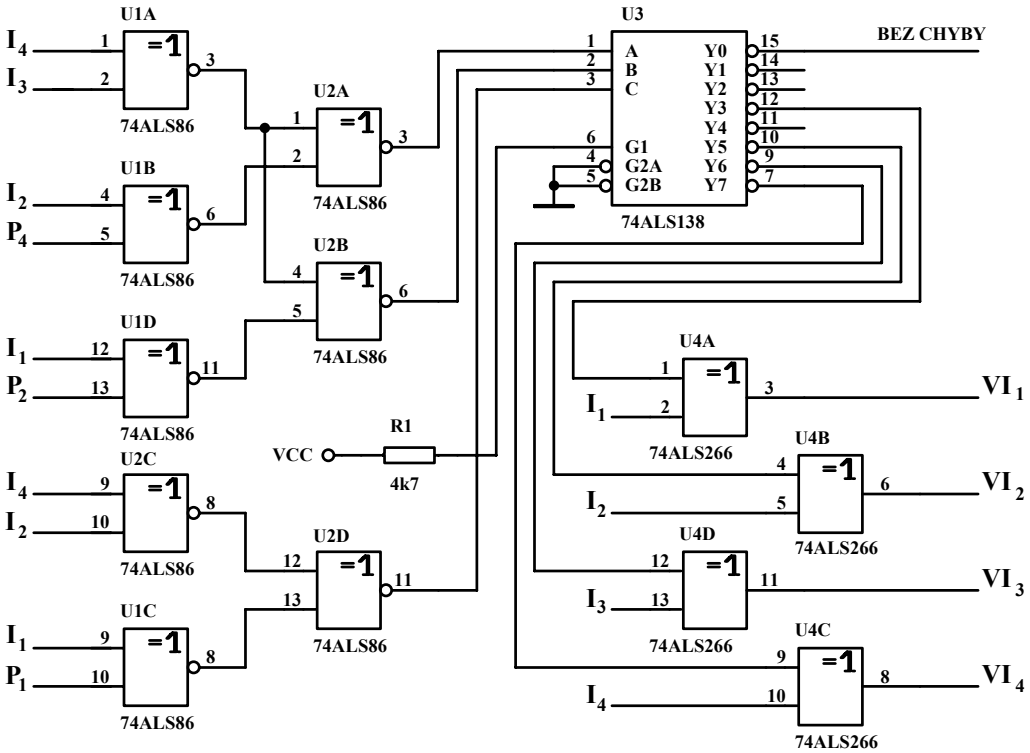
Navrhovaný obvod bude realizovat součin [H].[v], z kterého získáme tři výstupy určující hodnotu syndromu. Pro jednotlivé výstupy snadno odvodíme tyto vztahy

$$y_3 = I_4 \oplus I_3 \oplus I_2 \oplus P_4 \qquad (3.48)$$

$$y_2 = I_4 \oplus I_3 \oplus I_1 \oplus P_2 \qquad (3.49)$$

$$y_1 = I_4 \oplus I_2 \oplus I_2 \oplus P_1 \qquad (3.50)$$

Jestliže budou všechny výstupy y_3, y_2 a y_1 nulové, potom při přenosu nedošlo k chybě. Přesně řečeno mohlo dojít k trojnásobné chybě, kterou tento kód není schopen identifikovat. Bude-li alespoň jeden výstup nenulový a došlo pouze k jednoduché chybě, potom výstupy y_3, y_2, y_1 určují binárně hodnotu chybného bitu. Budeme-li výstupy y_3, y_2, y_1 dekódovat dekodérem 1 z 8 typu 74ALS138, potom můžeme příslušný chybný bit opravit pomocí obvodu 74ALS266 (EX-NOR). Při dvojnásobné chybě budou výstupy též nenulové, ale opraven bude nesprávný bit. Na obr.3.16 je výsledné zapojení obvodu pro opravu přijatého Hammingova kódu (7,4).



Obr.3.16

Tabulka 3.7 nám přináší přehled kódovacích a dekódovacích obvodů vyráběných v integrované podobě pro zabezpečení a opravu jednoduchých chyb. Jedná se vesměs o obvody vhodné k začlenění do 16 nebo 32 bitových mikroprocesorových systémů, kde zajišťují informaci ukládanou a čtenou z paměťového prostoru RAM.

Obvod		Kód	Počet bitů	Chyby	
	OK			Jednoduché	Dvojnásobné
AS280		Parita sudá/lichá	9	Indikuje	Neindikuje
AS286		Parita sudá/lichá	9	Indikuje	Neindikuje
LS636	LS637	modifik. Hammingův (13,5)	8	Indikuje Koriguje	Indikuje
ALS616	ALS617	modifik. Hammingův (22,6)	16	Indikuje Koriguje	Indikuje
LS630	LS631				
AS632A	AS633	modifik. Hammingův (29,7)	32	Indikuje Koriguje	Indikuje
AS634	AS635				

Tabulka 3.7

Příklady k samostatnému řešení:

Příklad 3.6.1 Určete úpravy obvodu sčítačky-odčítačky obr.3.6 pro případ, že vstupní čísla budou ve vyjádření jednotkovým doplňkem.

Příklad 3.6.2 Určete maximální počet bitů sčítačky s třístupňovým kanálem zrychleného přenosu tvořeného obvody 74AS282, určete potřebný počet těchto obvodů a sčítaček 74AS283, určete maximální dobu potřebnou k vytvoření součtu za předpokladu, že každý průchod obvodem bude trvat 6ns.

Příklad 3.6.3 Navrhněte logický kombinační obvod pro součet binárních čísel A a B $\in$(0,9). Výsledek vyjádřete v kódu BCD8421 (tzv. dekadická sčítačka - dekadická korekce).

4. Logické sekvenční obvody

Oproti kombinačním obvodům, kde výstupní hodnota byla dána, s výjimkou krátkého přechodného děje, okamžitou hodnotou vstupních proměnných x_1, x_2,..., x_n (vstupního vektoru $\vec{x} = [x_1, x_2, \ldots, x_n]$), je u sekvenčních obvodů výstupní hodnota závislá i na předcházejících hodnotách vstupního vektoru $\vec{x}$. Tuto závislost můžeme matematicky vyjádřit rovnicí

$$y_j^i = F\left(\vec{x}^{\,i}, \vec{x}^{\,i-1}, \vec{x}^{\,i-2}, \ldots, \vec{x}^{\,1}, \vec{z}^{\,1}\right) \qquad j = 1,2,\ldots,p \qquad (4.1)$$

Rovnici (4.1) lze upravit do následujícího tvaru tím, že v obvodu zavedeme vnitřní proměnné z_k^i (k=1,2,...,m)

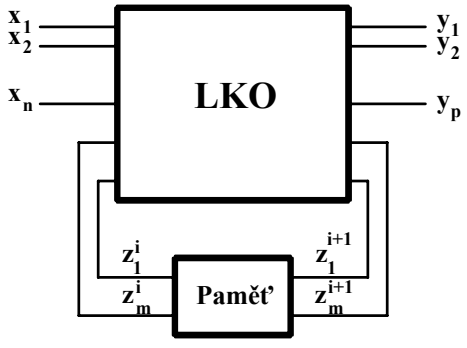
$$y_j^i = f_j\left(\vec{x}^{\,i}, \vec{z}^{\,i}\right) = f_j\left(x_1^i, x_2^i, \ldots, x_n^i, z_1^i, z_2^i, \ldots, z_m^i\right) \qquad (4.2)$$

$$z_k^{i+1} = g_k\left(\vec{x}^{\,i}, \vec{z}^{\,i}\right) = g_k\left(x_1^i, x_2^i, \ldots, x_n^i, z_1^i, z_2^i, \ldots, z_m^i\right) \qquad (4.3)$$

kde y_j^i je výstupní proměnná, x_r^i a z_k^i jsou vstupní a vnitřní proměnné v čase i. Postupným dosazováním rovnice (2.3) do rovnice (2.2) lze odvodit závislost výstupního vektoru $\vec{y}^{\,i} = [y_1^i, y_2^i, \ldots, y_p^i]$ na vstupní posloupnosti

$$\vec{y}^{\,i} = f\left(\vec{x}^{\,i}, g\left(\vec{x}^{\,i-1}, g\left(\vec{x}^{\,i-2}, g\left(\ldots g\left(\vec{x}^{\,1}, \vec{z}^{\,1}\right)\ldots\right)\right)\right)\right) \qquad (4.4)$$

kde $\vec{x}^{\,1}$ a $\vec{z}^{\,1}$ jsou počáteční vstupní a vnitřní stav obvodu. Logické funkce f_j, které se označují jako **funkce výstupů**, popisují závislost výstupů na vstupních hodnotách a vnitřních proměnných. Logické funkce g_k, které se označují jako **funkce přechodů**, popisují závislost následujících stavů vnitřních proměnných (v čase i+1) na vstupních a vnitřních proměnných. Základní zapojení LSO obr.4.1 je tvořeno pamětí a logickým kombinačním obvodem, který realizuje funkce výstupů a přechodů. Paměť sekvenčního obvodu může být realizována zpožděním v logických členech, zpožďovacími linkami nebo paměťovými obvody (klopnými obvody). Podle vlastností funkcí výstupů můžeme sekvenční obvody dělit na **Mealy**ho typ, kde výstupy jsou funkcemi vstupních i vnitřních proměnných, a na **Moore**ův typ, kde funkce výstupů jsou funkcí jenom vnitřních proměnných. Podle vlastností použitého typu paměti můžeme sekvenční obvody rozdělit na dvě skupiny:

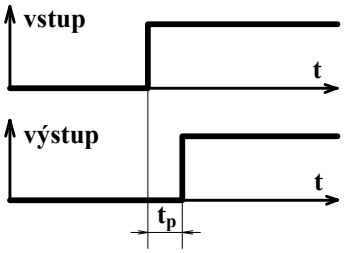


Obr.4.1

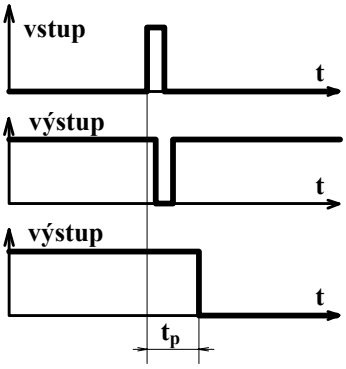
a) **synchronní obvody** - ke změnám vnitřních proměnných dochází téměř současně.

b) **asynchronní obvody** - ke změnám vnitřních proměnných dochází postupně v závislosti na tom, jak se šíří podnět obvodem.

Asynchronní sekvenční obvody můžeme pak dále dělit na obvody s hladinovými vstupy i výstupy obr.4.2a a na impulzní obvody s hladinovými nebo impulzními výstupy obr.4.2b, kde t_p je doba reakce paměťových členů.



Obr.4.2a



Obr.4.2b

V synchronních sekvenčních obvodech se používají ve zpožďovací (paměťové) funkci téměř výhradně paměťové členy (klopné obvody), které jsou řízeny zdrojem (hodinových) impulzů synchronizujících činnost všech základních obvodů systému do stejných okamžiků. Proto v synchronním systému nemohou existovat po sobě následující nestabilní vnitřní stavy obvodu. Pro spolehlivou činnost obvodu není nutné používat jeden zdroj hodinových impulzů s pravidelnými intervaly. Časový interval mezi hodinovými impulzy se volí tak, aby v obvodu odezněly přechodné jevy. Z tohoto důvodu se v synchronních obvodech neuplatní hazardní stavy, kterými může být zatížena kombinační část systému, ani hazardní stavy typické pro asynchronní sekvenční obvody. Jsou přípustné současné změny vstupních proměnných, pokud ke změně nedochází v době trvání hodinového impulzu nebo v okolí náběžné nebo sestupné hrany hodinového impulzu. Tyto okolnosti závisí na použitých paměťových členech, u kterých jsou v integrované formě podmínky pro správnou činnost a způsob synchronizace určeny výrobcem.

K realizaci asynchronního sekvenčního obvodu se mohou použít stejné základní logické členy jako v kombinačních obvodech s tím, že potřebné paměťové vlastnosti zajistíme zavedením vhodných zpětných vazeb. Paměť asynchronního sekvenčního obvodu může být realizována klopnými obvody nebo jen zpožděním v logických členech. Použití klopných obvodů pro paměťové funkce má řadu výhod. Podstatnou výhodou je relativně jednoduchý návrh při kterém se automaticky vyloučí možnost vzniku statických hazardů. U asynchronních obvodů se obvykle předpokládá, že zpoždění v logických členech LKO se vyčlení a soustředí do zpětnovazebních smyček. Ačkoliv předpokládáme ve zpětnovazebních smyčkách stejné zpoždění Δt, což nám umožňuje vyloučit rozměr času z logických výrazů a provádět řešení obvodu podobnými metodami jako u kombinačních obvodů, nelze u reálného obvodu tento předpoklad splnit. To znamená, že nelze počítat se současnou změnou dvou nebo více vnitřních proměnných. V případech, kdy by mělo dojít k současné změně vnitřních proměnných, je nutné provést analýzu všech hazardních stavů, které se v důsledku nedosažení současné změny

mohou objevit. V závislosti na době trvání vnějších signálů $\vec{x}^i$ rozeznáváme dva způsoby řízení asynchronních obvodů:

a) základní způsob řízení - po změně vektoru $\vec{x}^i$ smí jeho další změna následovat až po dosažení nového stabilního stavu obvodu.

b) impulzní - k řízení se používá impulz na jednom ze vstupů x_r. U těchto obvodů se obvykle předpokládá, že vstupní signály se nepřekrývají a k působení dalšího vstupního impulzu dochází až po ustálení odezvy na impulz předcházející. Každý impulz způsobí jen jednu změnu vnitřního stavu obvodu, a proto aktivní šířka vstupního impulzu musí být kratší než doba odezvy obvodu a současně dostatečně dlouhá na to, aby tuto odezvu vyvolala.

Každý logický sekvenční obvod je úplně specifikován šesti veličinami

$$LSO \equiv \left[\vec{x}, \vec{y}, \vec{z}, f, g, \vec{z}^1\right] \qquad (4.5)$$

kde $\vec{x}, \vec{y}$ a $\vec{z}$ jsou vektory vstupních, výstupních a vnitřních proměnných, f a g jsou funkce výstupů a přechodů a $\vec{z}^1$ je počáteční vektor vnitřních proměnných (počáteční vnitřní stav). Při jeho analýze a návrhu se setkáváme s následujícími pojmy:

Vnitřní stav obvodu může být vyjádřen kombinací hodnot vnitřních proměnných, (při analýze nebo konečné fázi návrhu obvodu) nebo obecným symbolem (při návrhu obvodu).

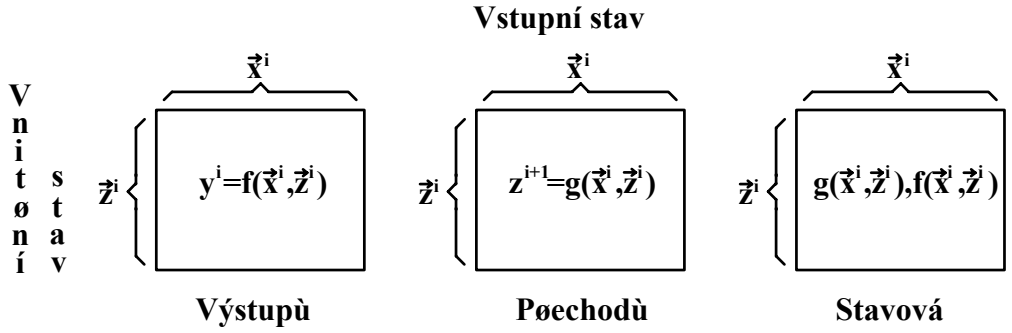
Stabilní stav asynchronního obvodu je stav, v němž obvod při konstatním vstupním vektoru $\vec{x}^i$ může setrvávat neomezenou dobu. Matematicky jej můžeme vyjádřit rovnicemi (4.6) a v tabulkách jej označujeme kroužkem.

$$\vec{x}^{i+1} = \vec{x}^i \qquad \vec{z}^{i+1} = \vec{z}^i \qquad (4.6)$$

Funkce přechodů a výstupů mohou být reprezentovány:

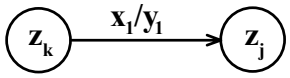
a) algebraickým výrazem - využívá se při analýze a v závěrečné fázi návrhu sekvenčního obvodu při přechodu od jeho struktury ke stavové tabulce a obráceně.

b) tabelárním vyjádřením - tabulkami z obr.4.3. Vývojová tabulka odpovídá stavové tabulce s tím, že vnitřní stavy jsou vyjádřeny obecně.

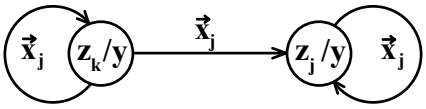


Obr.4.3

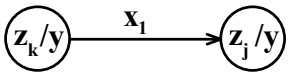
c) grafickým vyjádřením - tabulky přechodů a výstupů lze převést na orientovaný graf, kterým lze charakterizovat i typ logického sekvenčního obvodu. Na obr.4.4 jsou zobrazeny a) impulzní LSO s impulzním výstupem, b) impulzní LSO s hladinovým výstupem, c) asynchronní LS0, d) synchronní LSO - ke změně vnitřního stavu dojde pouze při synchronizačním signálu c (stejně bývá popsán i synchronizovaný asynchronní obvod).



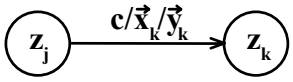
Obr.4.4a



Obr.4.4c



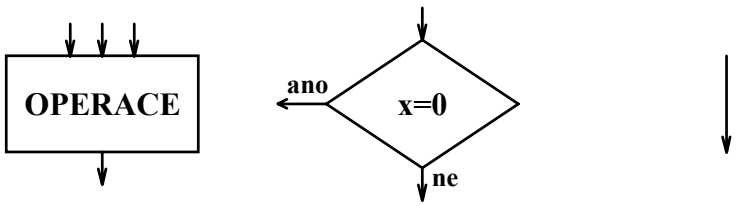
Obr.4.4b



Obr.4.4d

d) časovým diagramem vstupních a výstupních signálů obvodu.

e) vývojovým (programovým) diagramem - tento způsob zápisu se používá při popisu chování složitých sekvenčních soustav např. řadičů, procesorů počítačů, kde vystupuje velký počet proměnných a klasické prostředky reprezentace jsou nepřehledné. Vývojové diagramy obsahují tři typy prvků, které jsou uvedeny na obr.4.5.



Funkèní prvek **Rozhodovací prvek** **Spojnice**

Obr.4.5

f) Programem činnosti LSO - k popisu chování složitých sekvenčních soustav zavádí někteří autoři programovací jazyk vhodný k popisu činnosti těchto soustav (kapitola 5.4.1). V tomto jazyce vystupují instrukce typu zápis n-bitové slabiky do registru, zvětšení/zmenšení obsahu čítače o jedničku při splnění definované podmínky, atd. Chování obvodu pak popisuje program vytvořený z instrukcí tohoto jazyka.

4.1 Analýza logických sekvenčních obvodů

Předmětem analýzy sekvenčního obvodu je odvození chování známé struktury tzn. odvození funkcí výstupů f_j a přechodů g_k a jejich reprezentace stavovou tabulkou nebo stavovým diagramem.

Analýza asynchronních sekvenčních obvodů bez paměťových členů

Asynchronní logické sekvenční obvody bez paměťových členů se skládají z logických členů vytvářejících logickou síť se zpětnovazebními smyčkami. Pro dosažení modelu z obr.4.1, vyčleníme v prvním přiblížení zpoždění z jednotlivých logických členů a soustředíme jej ve formě zpožďovacích členů do zpětnovazebních smyček. Obvod je pak tvořen logickými členy bez zpoždění a zpožďovacími členy, které realizují paměťovou funkci. Analýza obvodu se pak skládá v těchto kroků:

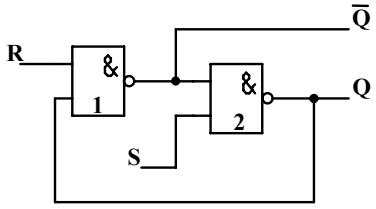
a) Určení množiny vnitřních stavů $\vec{z}$ (stanovení nemusí být jednoznačné - analýzu to však neovlivní). V obvodu nalezneme všechny zpětnovazební smyčky a vhodné výstupy logických členů, které jsou součástí zpětnovazebních smyček, označíme za vnitřní proměnné. Vnitřní proměnné volíme tak, aby jejich celkový počet byl minimální. Za výstup, kde jsme zvolili vnitřní proměnnou z_k, zařadíme zpožďovací člen.

b) Přerušení zpětnovazebních smyček. Na vstupu každého zpožďovacího členu definujeme proměnnou z_k^{i+1} a na jeho výstupu proměnnou z_k^i. Tímto způsobem je možné na jednom vodiči definovat dva různé stavy a analyzovat tak přechodné děje v obvodu.

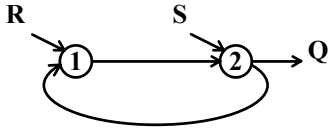
c) Rozpojením všech zpětnovazebních smyček se ze sekvenčního obvodu stal obvod pseudokombinační, u kterého snadno stanovíme algebraické výrazy pro funkce přechodů a výstupů.

d) Získané funkce přechodů a výstupů zapíšeme do stavové tabulky, určíme stabilní stavy a chování obvodu.

Příklad 4.1 Analyzujte logický sekvenční obvod z obr.4.6.



Obr.4.6



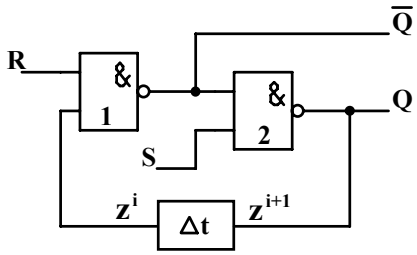
Obr.4.7

K analyzovanému zapojení nejprve nakreslíme diagram signálních toků obr.4.7, který nám může ve složitějších obvodech usnadnit vyhledávání zpětnovazebních smyček. V tomto případě je zřejmé, že v obvodu je jen jedna zpětnovazební smyčka, kterou je možné rozpojit za prvním nebo druhým logickým členem (existují dvě možnosti jak definovat jednu vnitřní proměnnou). Zvolíme-li výstupní proměnnou druhého logického členu jako vnitřní proměnnou z a vyčleníme zpoždění logických členů do zpožďovacího členu, bude situace vypadat dle obr.4.8, ze kterého již snadno určíme rovnici přechodu a výstupu

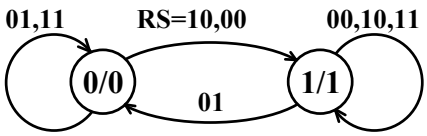
$$z^{i+1} = g\left(R^i, S^i, z^i\right) = \overline{\overline{z^i . R^i} . S^i} = \overline{S}^i + R^i . z^i \qquad (4.7)$$

$$y^i = Q^i = f\left(R^i, S^i, z^i\right) = z^{i+1} \qquad (4.8)$$

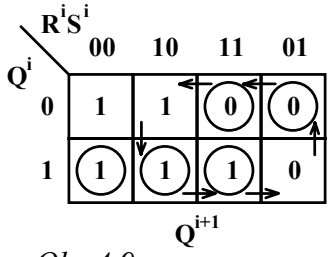
Rovnici přechodu zapíšeme do stavové tabulky, v které díky rovnici (4.8) nemusíme psát stav výstupu, a určíme stabilní stavy obvodu obr.4.9. Stabilní stavy, které označíme kroužkem, jsou v políčkách stavové tabulky, kde je splněn vztah $z^{i+1} = z^i$ tj. v horním řádku 0 a v dolním 1. Ze stavové tabulky lze odvodit odezvu obvodu na danou posloupnost vstupních stavů. Na počátku analýzy předpokládáme, že obvod se nachází ve stabilním stavu, který leží ve sloupci určeném počátečním vstupním stavem. Změnou vstupního stavu přejde obvod v daném řádku (s daným vnitřním stavem) do sloupce určeného novým vstupním stavem, kde je specifikován následující vnitřní stav. Je-li



Obr.4.8



Obr.4.10



Obr.4.9

tento stav nestabilní, přejde obvod do řádku, který přísluší novému vnitřnímu stavu. Tak obvod v daném sloupci přechází spontánně přes nestabilní stavy tak dlouho, dokud nedosáhne stabilního stavu. Takto můžeme postupně odvodit stavový diagram analyzovaného obvodu obr.4.10, který nám poskytuje lepší představu o jeho chování. Ve stavové tabulce je zachycena činnost obvodu na vstupní posloupnost RS = 10$\rightarrow$ 11 $\rightarrow$ 01 $\rightarrow$ 11 $\rightarrow$ 10.

4.1.1 Paměťové členy

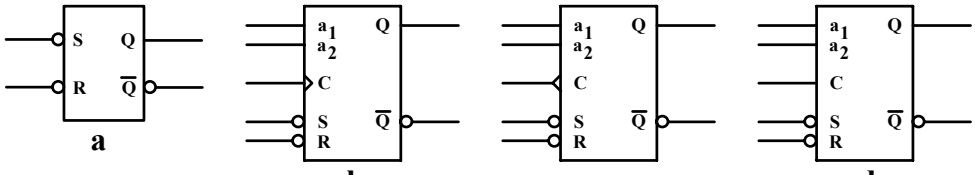
Paměťové členy, někdy označované jako klopné obvody, jsou asynchronní logické sekvenční obvody, které mají dva různé výstupní stavy, a využívají se jako paměť hodnoty logické proměnné. Paměťové členy, které se používají k realizaci čítačů, registrů, atd., můžeme podle jejich vlastností rozdělit na:

a) asynchronně řízené

b) synchronně řízené - hladinovým signálem
- náběžnou hranou synchronizačního signálu
- sestupnou hranou synchronizačního signálu

Schématické značení jednotlivých typů paměťových členů je na obr.4.11, kde a) značí asynchronní obvod s aktivní úrovní v log.0 (negace na vstupu), b) obvod synchronizovaný náběžnou hranou, c) obvod synchronizovaný sestupnou hranou (někdy se označuje hodinový

vstup negací a značkou pro náběžnou hranu) a d) obvod synchronizovaný úrovní hodinového signálu. Jak vyplývá z obr.4.11 jsou i synchronně řízené paměťové členy většinou vybaveny



Obr.4.11

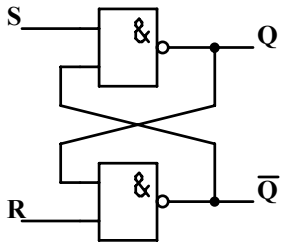
asynchronně řízenými vstupy, kterými je možné obvod nastavit (Q=1) nebo vynulovat (Q=0) bez ohledu na ostatní vstupy. Nastavovací vstup se obvykle označuje S (set) a nulovací R (reset). Paměťový člen je obvykle Mooreova typu (výstupní proměnná je současně i vnitřní proměnnou obvodu) a obvod má dva vzájemně komplementární výstupy. Chování paměťového členu vyjadřujeme operátorem (funkce přechodů a výraz omezující množinu vstupních stavů) nebo pravdivostní tabulkou. Operátor paměťového členu má tento obecný tvar

$$Q^{i+1} = f\left(a_1^i, a_2^i, Q^i\right) \qquad při \quad a\left(a_1, a_2\right) = 0 \qquad (4.9)$$

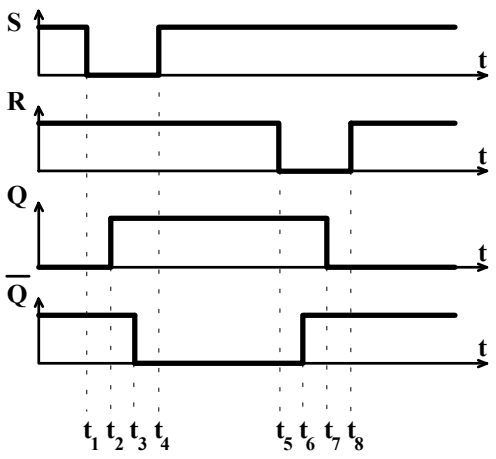
kde a_1, a_2 jsou vstupní proměnné paměťového členu.

Paměťový člen RS

Paměťový člen RS je asynchronně řízený obvod vstupními signály R (reset) a S (set). Jeho realizace se členy NAND je na obr.4.12, kde jsou též zobrazeny časové průběhy řídících signálů S a R, které jsou aktivní v log.0, a odezvy na tyto signály s respektováním zpoždění signálu v logických členech. Z obrázku je zřejmé, že minimální šířka řídícího signálu t_{smin} musí být větší než doba $\Delta t = t_3 - t_1 \approx 2.t_{pd}$, kde t_{pd} je střední zpoždění signálu jednoho logického členu NAND. Stavovou tabulku obvodu jsme již ukázali v příkladu na obr.4.9. V tab.4.1 je zobrazena pravdivostní tabulka tohoto paměťového členu i s její modifikací nezbytnou pro návrh LSO s paměťovými členy RS. Tato modifikovaná tabulka určuje hodnoty vstupních signálů, které zajišťují žádanou změnu výstupní proměnné paměťového členu.



Obr.4.12



Operátor paměťového členu RS je vyjádřen rovnicí

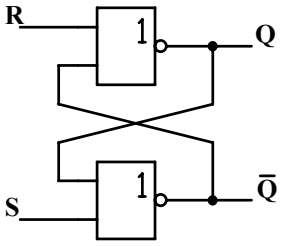
$$Q^{i+1} = \overline{S}^i + R^i . Q^i \qquad pro \quad \overline{R}^i . \overline{S}^i = 0 \qquad (4.10)$$

R^iS^i	Q^{i+1}
0 0	zak. stav
0 1	1
1 0	0
1 1	Q^i

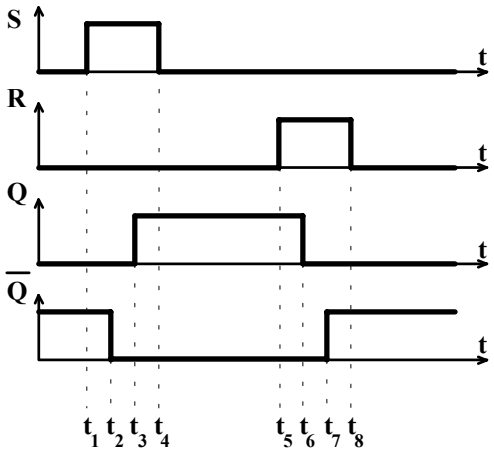
$Q^i \rightarrow Q^{i+1}$	R^iS^i
$0 \rightarrow 0$	1 X
$0 \rightarrow 1$	0 1
$1 \rightarrow 0$	1 0
$1 \rightarrow 1$	X 1

Tabulka 4.1

Stav R = S = 0 se označuje jako zakázaný (nepřípustný), protože při něm nejsou výstupy obvodu vzájemně komplementární $Q = \overline{Q} = 1$. Proto v případě, kdy je požadována u obvodu dvojice komplementárních výstupů, je třeba omezit pro tento obvod množinu vstupních stavů vyloučením stavu RS=[00]. Navíc při současné změně vstupů R a S ze stavu 0 do stavu 1 není jisté zda výstup Q



Obr.4.13



= 0 nebo Q=1.

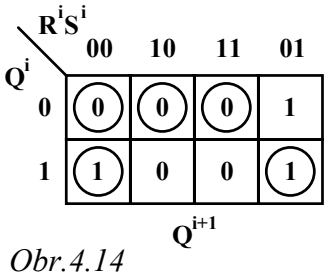
Paměťový člen RS může být realizován i s logickými členy NOR obr.4.13, kde je zobrazen s časovými průběhy řídících signálů R a S. Z obrázku je zřejmé, že aktivní úrovní signálů R a S je log.1., a proto nepřípustným stavem je stav RS=[11]. Na obr.4.14 je zobrazena stavová tabulka členu RS-NOR a v tab.4.2 je zobrazena pravdivostní a modifikovaná tabulka. Operátor paměťového členu RS-NOR je vyjádřen výrazem

$$Q^{i+1} = \overline{R}^i . \left(S^i + Q^i \right) \qquad pro \quad R^i . S^i = 0 \qquad (4.11)$$

R^iS^i	Q^{i+1}
0 0	Q^i
0 1	1
1 0	0
1 1	zak. stav

$Q^i \rightarrow Q^{i+1}$	R^iS^i
$0 \rightarrow 0$	X 0
$0 \rightarrow 1$	0 1
$1 \rightarrow 0$	1 0
$1 \rightarrow 1$	0 X

Tabulka 4.2



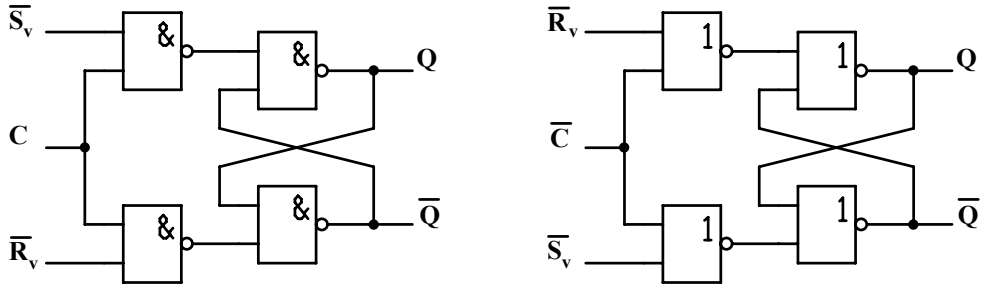
Obr.4.14

Hladinově řízené paměťové členy

Paměťové členy synchronně řízené hladinovým signálem mají výstupní stav determinovaný stavem vstupních proměnných pouze při splnění podmínky synchronizace. Pokud obvod není synchronizován jeho výstupní stav se nemění a obvod je ve stabilním stavu.

Paměťový člen RST

Na obr.4.15 jsou zobrazeny realizace paměťových členů RST logickými členy NAND a NOR. Obvody jsou synchronizovány úrovní vstupní proměnné C, kterou s ohledem na jednotné označení synchronizačního vstupu používáme. Vhodnějším označením vyplývajícím z názvu paměťového členu by bylo T. Je-li C=0 zůstává výstup obvodu Q nezávislý na



Obr.4.15

vstupních hodnotách S_v a R_v, pro C=1 se obvod chová jako paměťový člen RS. Chování obvodů můžeme popsat těmito vztahy

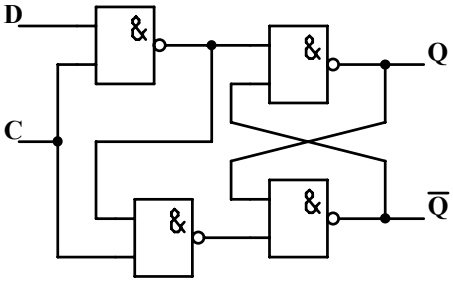
$$Q^{i+1} = C^i . \overline{S}_v^i + \left(R_v^i + \overline{C}^i\right).Q^i \qquad \text{pro} \quad \text{RST} - \text{NAND} \qquad (4.12)$$

$$Q^{i+1} = \left(\overline{C}^i + \overline{R}_v^i\right).\left(S_v^i . C^i + Q^i\right) \qquad \text{pro} \quad \text{RST} - \text{NOR} \qquad (4.13)$$

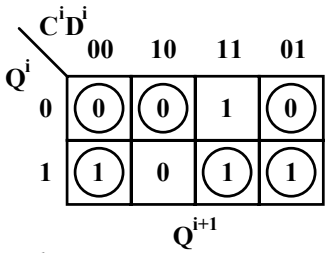
Vstupní proměnné S_v a R_v mají být v době, kdy je obvod synchronizován (T=1), konstantní. Tento požadavek se při sériovém řazení paměťových členů tohoto typu dá splnit jednofázovou impulzní synchronizací, při které musí být signál C dostatečně úzký impulz. Ten zabezpečí synchronizaci řízení výstupu Q vstupy S_v, R_v a zároveň skončí před případnou změnou na těchto vstupech. Šířka synchronizačního impulzu C je ovšem kritická a proto se častěji používá dvou a vícefázové řízení sériově řazených paměťových členů tohoto typu.

Paměťový člen D - Delay

Zapojení paměťového členu D je na obr.4.16. Jeho základem je opět paměťový člen RS, který pro nulový synchronizační signál (C=0) má na svých vstupech neaktivní úrovně (log.1) a proto jeho výstup nezávisí na vstupní proměnné D. V okamžiku, kdy hodinový signál C=1, je na vstup R přivedena vstupní proměnná D a na vstup S její negace. Výstup Q potom kopíruje stav



Obr.4.16



Obr.4.17

vstupní proměnné D, jak vyplývá i ze stavové tabulky tohoto obvodu obr.4.17. Operátor paměťového členu D je dán touto rovnicí

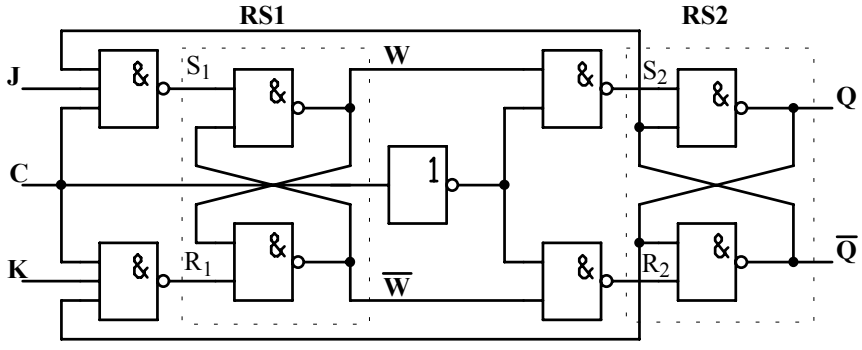
$$Q^{i+1} = \overline{C}^i . Q^i + C^i . \overline{Q}^i \qquad (4.14)$$

Paměťové členy synchronizované hranami hodinového signálu

Paměťové členy hranově synchronizované umožňují řízení svého výstupu hodnotami vstupních signálů v okamžicích změny synchronizačního signálu (vstupní signály jsou vzorkovány hranami synchronizačního signálu). Klopné obvody, jak bývají označovány hranově synchronizované paměťové členy, jsou obvykle tvořeny několika paměťovými členy mezi nimiž je pře-nos signálu řízen alespoň dvoufázově.

Paměťový člen JK

Paměťový člen JK řízený hranou hodinového signálu je příkladem realizace paměťového členu s dvoufázovým přenosem dat, který je řízen jediným hodinovým signálem. Náběžnou hranou je vstupní informace zapsána do 1. vnitřního paměťového členu a se závěrnou hranou se informace přepisuje do výstupního paměťového členu. Člen JK je označován také jako obvod master-slave (pán-sluha). Název postihuje funkci vnitřních paměťových členů obvodu JK z nichž první vystupuje jako řídící - pán a druhý jako řízený - sluha. Zjednodušená realizace obvodu JK je na obr.4.18. Struktura obvodu je tvořena dvěma RS obvody. Výstupní proměnné těchto obvodů jsou vnitřními proměnnými LSO. Obvod má vstupní proměnné J,K,C, výstupní proměnnou Q a vnitřní proměnné W a Q. Pro funkce buzení vstupů paměťových členů můžeme psát



Obr.4.18

$$R_1 = \overline{C} + \overline{K} + \overline{Q} \qquad S_1 = \overline{C} + \overline{J} + Q \qquad (4.15)$$

$$R_2 = C + W \qquad S_2 = C + \overline{W} \qquad (4.16)$$

Po dosazení do operátoru paměťového členu $Q^{i+1} = \overline{S}^i + R^i . Q^i$ dostáváme tyto funkce přechodů pro vnitřní proměnné W a Q

$$W^{i+1} = C^i . J^i . \overline{Q}^i + \left(\overline{C}^i + \overline{K}^i + \overline{Q}^i\right). W^i \qquad (4.17)$$

$$Q^{i+1} = \overline{C}^i . W^i + \left(C^i + W^i\right). Q^i \qquad (4.18)$$

W^iQ^i \ $J^iK^iC^i$	000	100	110	010	011	111	101	001
00	(00)	(00)	(00)	(00)	(00)	10	10	(00)
10	11	11	11	11	(10)	(10)	(10)	(10)
11	(11)	(11)	(11)	(11)	01	01	(11)	(11)
01	00	00	00	00	01	(01)	(01)	(01)

$W^{i+1}Q^{i+1}$

Obr.4.19

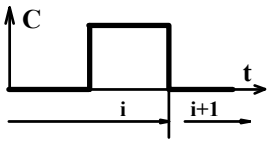
Pomocí odvozených rovnic přechodů snadno vytvoříme stavovou tabulku obvodu obr.4.19, v které zakroužkujeme stabilní stavy. Obvyklá činnost obvodu předpokládá, že s náběžnou hranou signálu C se řídí podle stavu vstupů JK paměťový člen RS1 a se závěrnou hranou C se informace z členu RS1 přepíše do paměťového členu RS2. Činnost obvodu můžeme popsat pravdivostní tabulkou tab.4.3 nebo následujícím operátorem paměťového členu, kde čas i se počítá až do sestupné hrany hodinového signálu obr.4.20.

$$Q^{i+1} = J^i . \overline{Q}^i + \overline{K}^i . Q^i \qquad (4.19)$$

J^iK^i	Q^{i+1}
0 0	Q^i
0 1	1
1 0	0
1 1	$\overline{Q}^i$

$Q^i \rightarrow Q^{i+1}$	J^iK^i
$0 \rightarrow 0$	0 X
$0 \rightarrow 1$	1 X
$1 \rightarrow 0$	X 1
$1 \rightarrow 1$	X 0

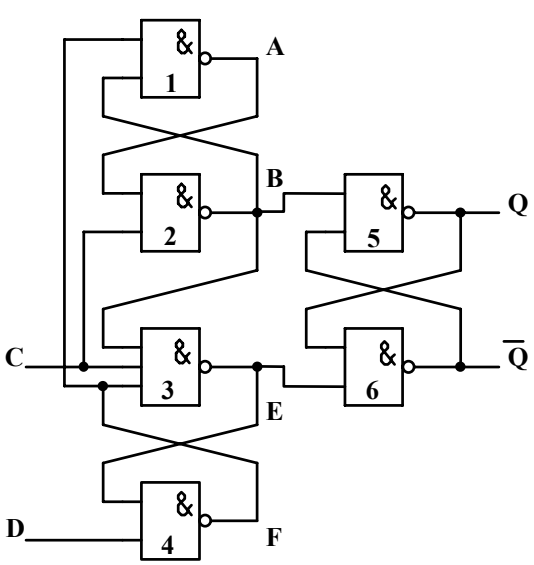
Tabulka 4.3



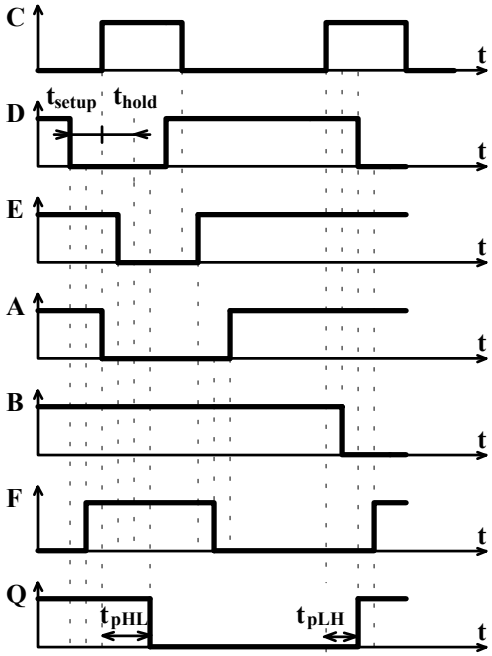
Obr.4.20

Paměťový člen D hranově řízený

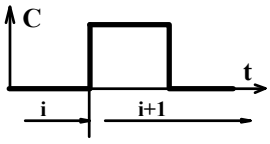
U hranově řízeného paměťového členu D se využívá zpětná vazba, která zabrání průchodu vstupního signálu D strukturou obvodu bezprostředně po náběžné hraně synchronizačního signálu C. Na obr.4.21 je zobrazeno zjednodušené zapojení paměťového členu D (bez asynchronních vstupů R a S) spolu s časovými průběhy vstupních, vnitřních a výstupních signálů. Při D=0 je další přenos případných změn signálu D blokován již v logickém členu č.4 signálem E=0 a při D=1 se blokování realizuje v logickém členu č.3 signálem B=0. Na



Obr.4.21



obr.4.21 jsou vyznačeny doby v nichž probíhá zápis hodnoty vstupní proměnné do vnitřního paměťového členu a dále přenos stavu vnitřního paměťového členu na výstup. Při malé strmosti náběžné hrany hodinového signálu dochází k prodloužení těchto dob. Ze způsobu činnosti paměťového členu vyplývá nutnost, aby v okolí aktivní hrany synchronizačního signálu byl vstupní signál D ustálený. Doby předstihu t_{setup} a přesahu t_{hold} signálu D vůči náběžné hraně synchronizačního signálu jsou znázorněny na obr.3.33 a jejich nedodržení způsobuje v sekvenčním obvodu hazardní stavy. Operátor paměťového členu D hranově řízeného je dán rovnicí (4.20), kde čas i se počítá do vzestupné hrany hodinového signálu obr.4.22.

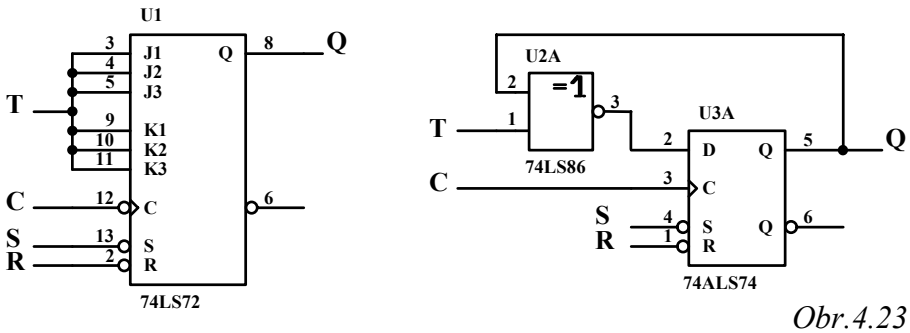


Obr.4.22

$$Q^{i+1} = D^i \qquad (4.20)$$

Paměťový člen T

Paměťový člen T (synchronní invertor) je obvod, který při vstupním signálu T=1 vždy po příchodu synchronizačního signálu změní hodnotu svého výstupu ($Q^{i+1} = \overline{Q}^i$). Pro vstupní



Obr.4.23

signál T=0 se hodnota výstupu nemění. Uvedené chování popisuje operátor paměťového členu, který je dán tímto vztahem

$$Q^{i+1} = T^i \oplus Q^i = \overline{T}^i . Q^i + T^i . \overline{Q}^i \qquad (4.21)$$

Paměťový člen T není v integrované podobě vyráběn, ale lze jej realizovat pomocí členu JK (J=K=T) nebo členu D (D=Q⊕T) obr.4.23.

Paměťový člen E

Je dalším paměťovým členem, který se v integrované podobě nevyrábí. Jeho pravdivostní tabulka tab.4.4 je kromě kombinace JK=11 shodná s pravdivostní tabulkou paměťového členu JK.

J^iK^i	Q^{i+1}
0 0	Q^i
0 1	1
1 0	0
1 1	Q^i

Tabulka 4.4

Na závěr části o paměťových členech uvedeme přehled vyráběných paměťových členů v novějších technologiích tab.4.5. Symbol a/n v tabulce značí ano pro obvod v prvém sloupci a ne pro obvod ve sloupci druhém. Symboly použité ve sloupci synchronizace mají následující význam:  - obvod synchronizovaný náběžnou hranou,  - obvod synchronizovaný sestupnou hranou,  - obvod synchronizovaný úrovní signálu,  - obvod typu master-slave se sestupnou výkonnou hranou hodinového signálu.

4.1.2. Analýza logických sekvenčních obvodů s paměťovými členy

Asynchronní i synchronní sekvenční obvody s paměťovými členy by bylo možné analyzovat stejným způsobem jako v předcházející části. Analýza se však i pro jednoduché obvody stává nepřehlednou a proto se obvody s paměťovými členy analyzují následujícím způsobem:

a) Pro dané zapojení stanovíme vnitřní proměnné, které jsou shodné s výstupy paměťových členů (počet vnitřních proměnných = počet paměťových členů) a množiny vstupních a výstupních stavů.

b) Odvodíme algebraické vyjádření funkcí pro buzení vstupů paměťových členů.

c) Dosazením funkcí pro buzení vstupů paměťových členů do příslušných operátorů získáme funkce přechodů.

d) Odvodíme funkce výstupů a stavovou tabulku, z které určíme chování obvodu.

Obvod	Obvod	Typ	Počet	Q	$\overline{Q}$	R	S	Synch.	Výstup
74LS73		JK	2	ano	ano	ano	ne		2 stav.
74LS74		D	2	ano	ano	ano	ano		2 stav.
74LS75	74LS375	D	4	ano	ano	ne	ne		2.stav.
74LS76		JK	2	ano	ano	ano	ano		2.stav.
74LS109		JK	2	ano	ano	ano	ano		2.stav.
74LS112	74LS114	JK	2	ano	ano	ano	ano		2.stav.
74LS113		JK	2	ano	ano	ne	ano		2.stav.
74LS174	74LS378	D	6	ano	ne	a/n	ne		2.stav.
74LS175	74LS379	D	4	ano	ano	a/n	ne		2.stav.
74LS273	74AS575	D	8	ano	ne	ano	ne		2.stav.
74LS279		RS	4	ano	ne	ano	ano		2.stav.
74AS373	74AS573	D	8	ano	ne	ne	ne		3.stav.
74AS374	74AS574	D	8	ano	ne	ne	ne		3.stav.
74ALS533	74ALS563	D	8	ne	ano	ne	ne		3.stav.
74AS534	74AS564	D	8	ne	ano	ne	ne		3.stav.
□74ALS576		D	8	ne	ano	ne	ne		3.stav.
74AS580		D	8	ano	ne	ne	ne		3.stav.
74AS821	74AS822	D	10	a/n	n/a	ne	ne		3.stav.
74AS823	74AS824	D	9	a/n	n/a	ano	ne		3.stav.
74AS825	74AS826	D	8	a/n	n/a	ano	ne		3.stav.
74AS841	74AS842	D	10	a/n	n/a	ne	ne		3.stav.
74AS843	74AS844	D	9	a/n	n/a	ano	ano		3.stav.
74AS845	74AS846	D	8	a/n	n/a	ano	ano		3.stav.
74AS873	74AS880	D	2x4	a/n	n/a	a/n	n/a		3.stav.
74AS874	74AS878	D	2x4	ano	ne	ano	ne		3.stav.
74AS876		D	2x4	ne	ano	ne	ano		3.stav.
74AS879		D	2x4	ne	ano	ano	ne		3.stav.

Tabulka 4.5

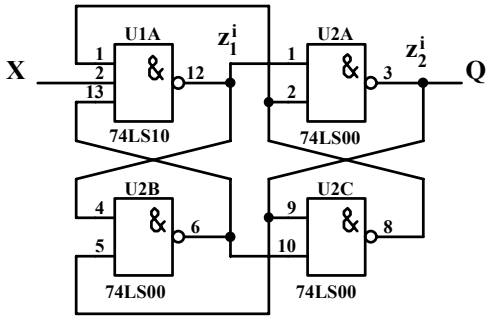
Příklad 4.2 Analyzujte logický sekvenční obvod s paměťovými členy RS z obr.4.24.

Obvod se skládá ze dvou RS paměťových členů pro jejichž vstupy S_1, R_1, S_2 a R_2 můžeme psát tyto rovnice (rovnice buzení vstupů paměťových členů)

$$S_1^i = x^i . \overline{z}_2^i \qquad R_1 = z_2^i$$
$$S_2^i = z_1^i \qquad R_2 = \overline{z}_1^i \qquad (4.22)$$

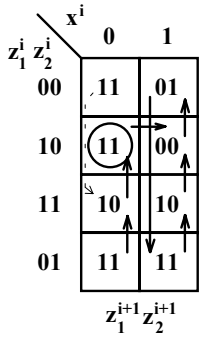
Dosadíme-li rovnice buzení vstupů (4.22) do rovnice operátoru paměťového členu RS (4.10) získáme tyto rovnice přechodů

$$z_1^{i+1} = \overline{S}_1^i + R_1^i . z_1^i = \overline{x}^i + z_2^i + z_1^i . z_2^i = \overline{x}^i + z_2^i$$
$$z_2^{i+1} = \overline{S}_2^i + R_2^i . z_2^i = \overline{z}_1^i + \overline{z}_1^i . z_2^i = \overline{z}_1^i \qquad (4.23)$$



Obr.4.24

Z rovnic (4.23) odvodíme stavovou tabulku, která je na obr.4.25 a určíme v ní stabilní stavy. Jeden stav (obvykle stabilní), který leží ve sloupci určeném kombinací vstupních proměnných, zvolíme jako počáteční ($x = 0, z_1^i z_2^i = 10$). Změnou vstupního stavu přejde obvod v daném řádku ($z_1^i z_2^i = 10$) do sloupce určeného novým vstupním stavem (x = 1), kde je specifikován následující vnitřní stav obvodu ($z_1^i z_2^i = 00$). Je-li tento stav nestabilní, přejde obvod v daném sloupci do řádku určeného novým vnitřním stavem ($z_1^i z_2^i = 00$), v kterém je definován další vnitřní stav obvodu ($z_1^i z_2^i = 01$). Tak obvod ve sloupci určeném vstupní proměnnou (x = 1) prochází nestabilními stavy. Při změně vstupního stavu $x = 1 \rightarrow 0$ přejde obvod v prvém sloupci přes nestabilní stavy 01 a 11 nebo přímo do stabilního stavu 10. Problémem zůstává přechod ze stavu 00, z kterého je předepsán přechod vyžadující současnou změnu obou vnitřních proměnných ($00 \rightarrow 11$) tzv. soubě-hový hazardní stav. Protože k současné změně obou proměnných nemusí dojít, je třeba určit jeho chování za předpokladu, že se dříve změní proměnná z_1^i ($z_1^{i+1} z_2^{i+1} = 10$) nebo z_2^i ($z_1^{i+1} z_2^{i+1} = 01$). Z tabulky však zjistíme, že v obou případech se obvod dostane do požadovaného stabilního stavu 10. Analyzovaný obvod obr.4.24 je spouštěný generátor impulzů, který pro x=1 generuje na výstupu přibližně obdélníkové napětí. Pro x=0 přejde do stabilního stavu ($z_1^i z_2^i = 10$).



Obr.4.25

Souběhy a cykly

Při analýze asynchronních sekvenčních obvodů se lze setkat se dvěma nežádoucími jevy: cykly a souběhy. **Cyklem** označujeme činnost obvodu, který při stabilním vstupním stavu, prochází uzavřenou posloupností nestabilních vnitřních stavů (např. obr.4.25 při x = 1). S výjimkou generátorů impulzů je existence cyklů nežádoucí, protože vzniká neurčitost další činnosti obvodu, který vychází z cyklu. **Souběh** nastává tehdy, když je stavovou tabulkou předepsána současná změna dvou nebo více vnitřních proměnných. Protože vlivem rozptylu parametrů součástek k současné změně více proměnných nemůže dojít, změní se nejprve jedna proměnná. Obvod může přejít do nežádoucího vnitřního stavu, který se stane novým výchozím stavem pro jeho další činnost. Pokud obvod bez ohledu na pořadí změn vnitřních proměnných přejde do původně předepsaného stavu, pak souběh označujeme jako **nekritický** (viz. obr.č.25 při x = 0, přechod ze stavu $z_1^i z_2^i = 00$). V opačném případě se jedná o souběh **kritický**.

4.1.3. Analýza asynchronních sekvenčních obvodů se synchronizovanými paměťovými členy.

Asynchronní sekvenční obvody se synchronizovanými paměťovými členy jsou obvody, u nichž jsou hodinové signály nebo asynchronní vstupy některých paměťových členů závislé na vnitřních proměnných obvodu

$$C_j = f_j\left(Q_1,\ldots,Q_{j-1},Q_{j+1},\ldots Q_r\right) \qquad (4.24)$$

kde r je počet vnitřních proměnných. Postup analýzy je v podstatě stejný jako u synchronních obvodů s tím rozdílem, že je nutné navíc odvodit také proměnné C_j (nebo R a S) paměťových členů a s tím, že u příslušného paměťového členu se může výstup měnit jedině tehdy (musí výstup nebýt odpovídající hodnot), když dojde k předepsané změně hodinového signálu (když asynchronní vstup je aktivní). Analýzu lze shrnout do těchto kroků:

a) Pro dané zapojení stanovíme vnitřní proměnné a určíme množinu vstupních a výstupních proměnných.

b) Zvolíme jeden z vnitřních stavů jako počáteční a pro všechny vstupní stavy určíme výstupní stavy a hodnoty vstupních proměnných paměťových členů.

c) Pomocí operátorů paměťových členů odvodíme následující vnitřní stav, který v dalším kroku použijeme jako stav počáteční.

d) Body b) a c) opakujeme do vyčerpání všech vnitřních stavů obvodu.

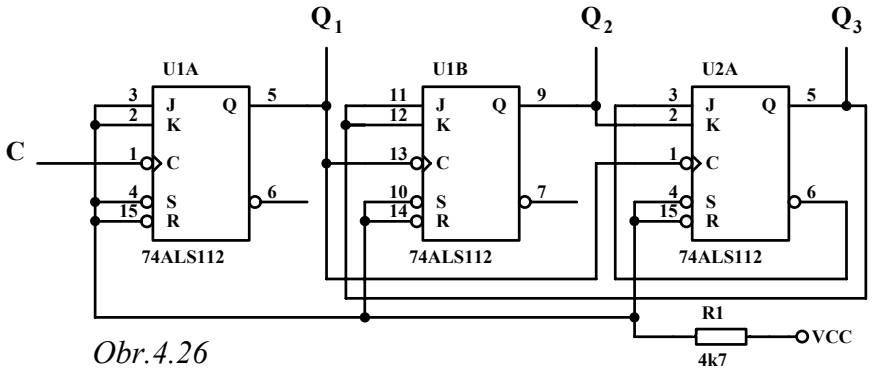
Příklad 4.3 Analyzujte zapojení z obr.4.26 s paměťovými členy JK synchronizovanými závěrnou hranou (sestupnou) hodinových impulzů.

Jedná se opět o primitivní automat s 2^3 stavy. Pro funkce buzení vstupů J,K a C lze odvodit tyto vztahy

$$J_1 = K_1 = 1 \qquad J_2 = K_2 = Q_3 \qquad (4.25)$$

$$J_3 = \overline{Q_3} \qquad K_3 = Q_2 \qquad C_2 = C_3 = Q_1 \qquad (4.26)$$

Ke změně stavu vnitřních proměnných Q_2 a Q_3 může dojít jen tehdy, když proměnná Q_1 mění svoji hodnotu z $1 \rightarrow 0$. Jako výchozí vnitřní stav zvolíme např. stav 0 ($Q_1 = Q_2 = Q_3 = 0$) a za pomoci operátoru paměťového členu JK rovnice (4.19) určíme stavy následující. Tabulka

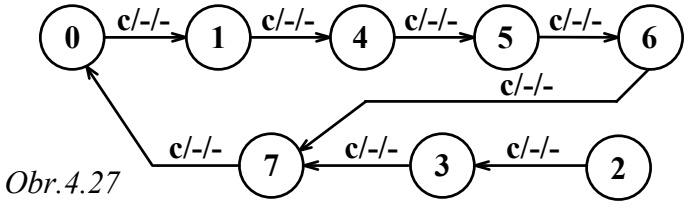


Obr.4.26

přechodů obvodu, kterou takto získáme i s vyčíslením vstupních proměnných J_j, K_j a C_j paměťových členů je v tab.4.6.

$Q_3^i\ Q_2^i\ Q_1^i$	Stav z^i	C_2^i C_3^i	$J_1^i\ K_1^i$	$J_2^i\ K_2^i$	$J_3^i\ K_3^i$	$Q_3^{i+1}\ Q_2^{i+1}\ Q_1^{i+1}$	Stav z^{i+1}
0 0 0	0	0	1 1	0 0	1 0	0 0 1	1
0 0 1	1	1	1 1	0 0	1 0	1 0 0	4
1 0 0	0	0	1 1	1 1	0 0	1 0 1	5
1 0 1	5	1	1 1	1 1	0 0	1 1 0	6
1 1 0	6	0	1 1	1 1	0 1	1 1 1	7
1 1 1	7	1	1 1	1 1	0 1	0 0 0	0
0 1 0	2	0	1 1	0 0	1 1	0 1 1	3
0 1 1	3	1	1 1	0 0	1 1	1 1 0	6

Tabulka 4.6

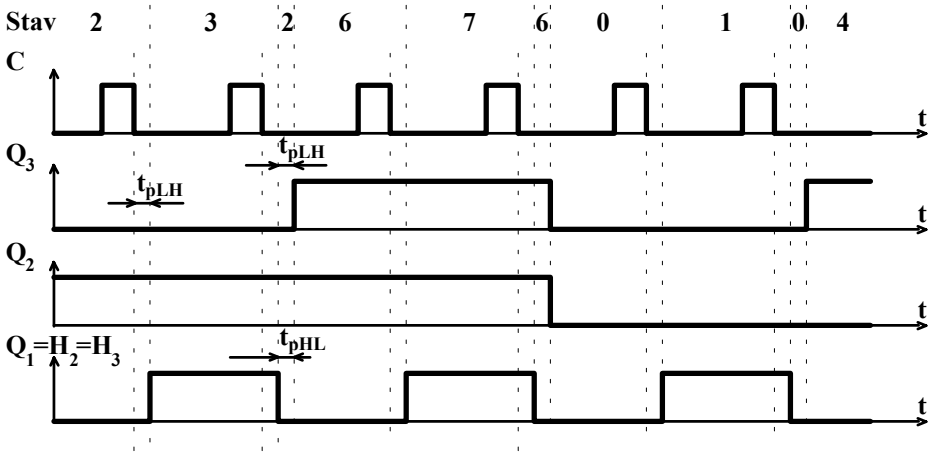


Obr.4.27

Z počátečního stavu 0 přejde obvod do stavu 1, proměnné Q_2 a Q_3 se nemohou změnit, protože C_2 a C_3 nemění stav z $1 \rightarrow 0$. Ze stavu 1 pak obvod přechází do stavu

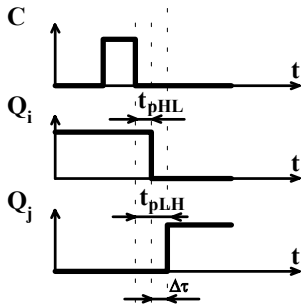
4 atd., až nakonec ze stavu 7 opět do stavu 0. Protože ve smyčce kterou vytváří stavy 0,1,4,5,6,7 nejsou stavy 2 a 3 musíme zjistit do jakých vnitřních stavů se obvod dostane z těchto dvou stavů. Na obr.4.27 je potom zobrazen výsledný diagram přechodů obvodu a na obr.4.28 jsou nakresleny časové průběhy na výstupech obvodu Q_1, Q_2 a Q_3, kde t_{pHL} a t_{pLH} jsou zpoždění výstupu Q za sestupnou hranou hodinového impulzu.

Z obr.2.28 vyplývá, že asynchronní sekvenční obvod má v okamžicích překlápění přechodné stavy, které jsou způsobeny zpožděním signálu na výstupu Q oproti signálu na hodinovém vstupu. Tento přechodový děj bude maximálně trvat dobu t_{pLH} nebo t_{pHL}. Mohlo



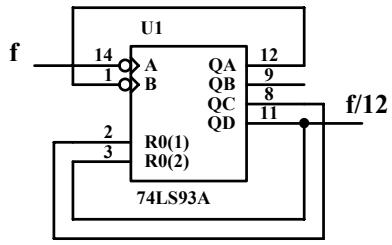
Obr.4.28

by se zdát, že synchronní čítač nemá přechodné stavy, protože klopné obvody se překlápějí současně. Ve skutečnosti to však není pravda a lze pozorovat rovněž přechodné stavy, které však budou trvat kratší dobu než u asynchronních obvodů. Přechodný stav vznikne pouze v době mezi časy t_{pLH} a t_{pHL} (tzn., že jeho doba trvání $\Delta\tau = \left|t_{pHL} - t_{pLH}\right|$ obr. 4.29. Tato doba může být rozdílná u jednotlivých stupňů, zvláště když mají různé zátěže.



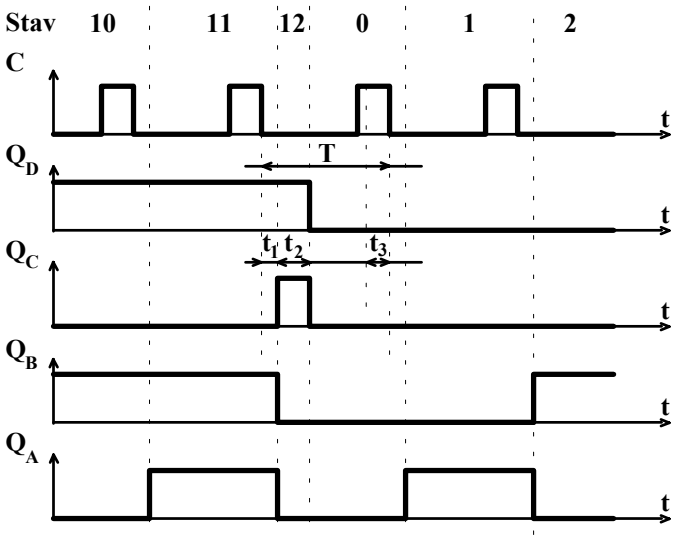
Obr.4.29

Příklad 4.4 Určete jaký nejvyšší kmitočet může dělit dělička dvanácti vytvořená pomocí čtyřbitového dvojkového čítače typu 74LS93A, který je zapojen dle obr.4.30.



Obr.4.30

Obvod 74LS93A je šestnáctkový asynchronní čítač vytvořený ze čtyř obvodů JK, z nichž 3 jsou kaskádně zapojeny za sebou (viz. vnitřní struktura obvodu). Čítač reaguje na sestupnou hranu po níž se binární číslo vytvořené z výstupů Q_4, Q_3, Q_2, Q_1 zvýší o jedničku až do hodnoty 15 a pak vynuluje. Dělení (nebo čítání) číslem N (modulo N), které je menší než modul celého čítače, se může vytvořit tak, že výstupem detektoru čísla N vynulujeme čítač pomocí asynchronního vstupu R (tzv. zkrácení cyklu kapitola 5.2). Obvod 74LS93A je vybaven dvěmi nulovacími vstupy R_{01} a R_{02}, které vynulují obvod v případě, že $R_{01}.R_{02} = 1$. Princip dělení dvanácti pomocí vstupů R_{01} a R_{02} je zřejmý z obr.4.31. Detektor čísla dvanáct je nyní realizován dvěmi propojovacími vodiči $R_{01} = Q_C$ a $R_{02} = Q_D$. Jak-mile čítač dosáhne čísla dvanáct bude součin $R_{01}.R_{02}$ mít hodnotu log.1 a čítač se vynuluje (nezávisle na hodinových impulzech). To znamená, že mezi dvěmi sestupnými hranami hodinového signálu (děleného kmitočtu) se vyskytují dva stavy obvodu tj. stav 12 a 0. Pro určení maximálního kmitočtu, který může obvod dělit, platí dvě omezující podmínky:



Obr.4.31

a) Vstupní kmitočet musí být menší než maximální přípustný kmitočet udávaný výrobcem. Jeho hodnota $f_{max} = 32\text{MHz}$.

b) Musí být zajištěny podmínky pro správnou činnost obvodu.

Situace při přechodu ze stavu 11 do stavu 12 a dále do stavu 0 je zobrazena na obr.4.31, kde t_1 je zpoždění výstupu Q_C za sestupnou hranou hodinového signálu, t_2 je zpoždění výstupů za nulovacím signálem a $t_3 = t_{setup}$ je předstih nulovacího impulzu před sestupnou hranou hodinového signálu. Z obr.4.31 je zřejmé, že pro správnou činnost obvodu musí být splněny maximální časy t_1, t_2 a t_3.Z konstrukčního katalogu [5] pak zjistíme, že

$$t_1 = t_{pLH\max} = t_{pHL(A\to Q_A)} + t_{pLH(B\to Q_C)} = 18 + 32 = 50 \quad [ns] \qquad (4.27)$$

$$t_2 = t_{pHL\max(R_0\to Q_j)} = 40 \quad [ns] \qquad t_3 = t_{setup\min} = 25 \quad [ns]$$

$$t_{pHL\,\min\left(R_0 \to Q_j\right)} > t_W\left(R_0\right)_{\min} \qquad (4.28)$$

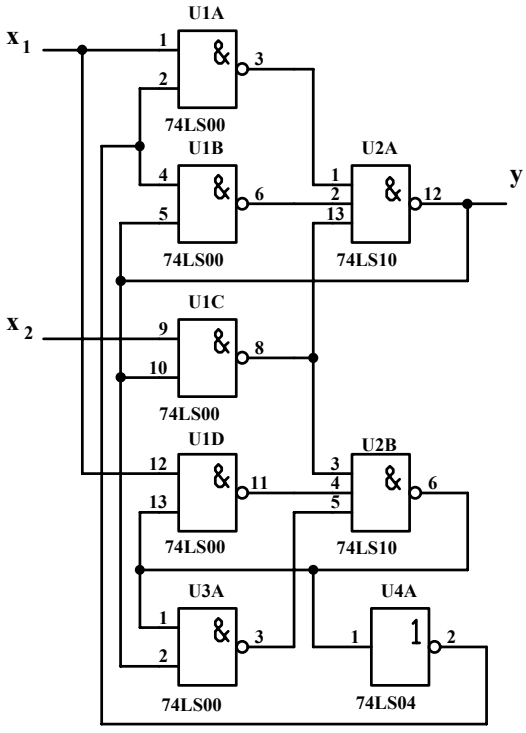
Odtud maximální kmitočet $f_{max} = 1/115.10^{-9} = 8{,}69$ MHz. To je hodnota, při které bude obvod spolehlivě pracovat pokud budou splněny podmínky tolerance zaručené výrobcem a podmínka pro správnou funkci nulování (4.28). Tato podmínka značí, že délka vzniklého nulovacího impulzu (doba t_2) musí být větší než minimální doba trvání nulovacího impulzu udaná v konstrukčním katalogu. Tato nerovnost musí být splněna i v nejnepříznivějším případě tzn.i pro nejrychleji se nulující klopný obvod. Numerické výpočty pro různé klopné obvody ukazují, ža tato podmínka je splněna jen taktak. Pro typické hodnoty t_i udávané v katalogu vypočteme $f_{max} = 12{,}5$ MHz. Každý obvod však nemusí na tomto kmitočtu pracovat spolehlivě.

Příklady k samostatnému řešení

Příklad 4.1.1. Odvoďte stavový diagram a určete chování obvodu z obr.4.32.

Příklad 4.1.2. Odvoďte chování synchronního sekvenčního obvodu z obr.4.33. V obvodu jsou použity klopné obvody typu T.

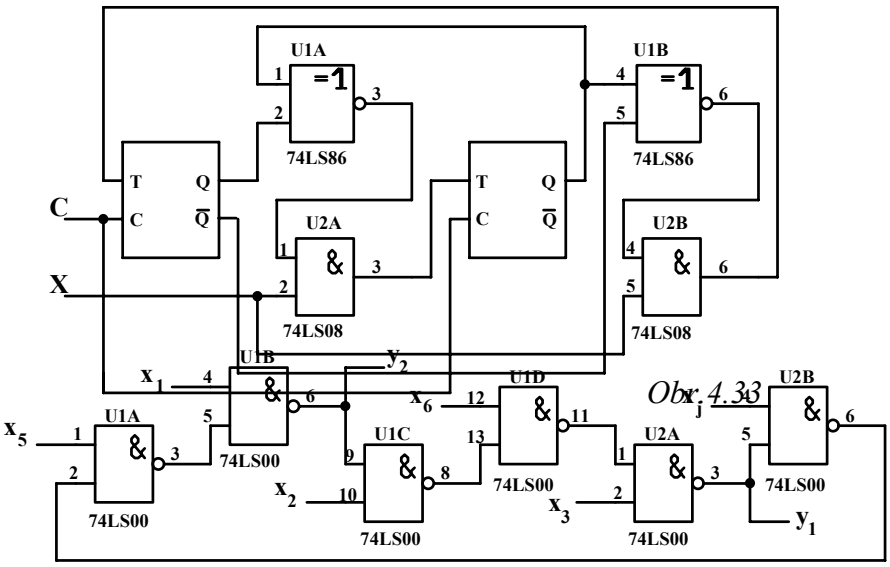
Příklad 4.1.3. Určete zda na obr.4.34 je nakreslen kombinační nebo sekvenční obvod pro $x_j = x_4$ a $x_j = \bar{x}_2$ Stanovte nutnou a postačující podmínku omezení množiny vstupních stavů zajišťující, že obvod se bude chovat jako kombinační.



Obr.4.32

Příklad 4.1.4. Jaká nejmenší doba musí být mezi dvěmi změnami proměnných x_1 a x_2 obvodu z obr.4.32, aby byla zajištěna jeho spolehlivá činnost ($t_{pHLmax} = 15[ns]$, $t_{pLHmax} = 22[ns]$).

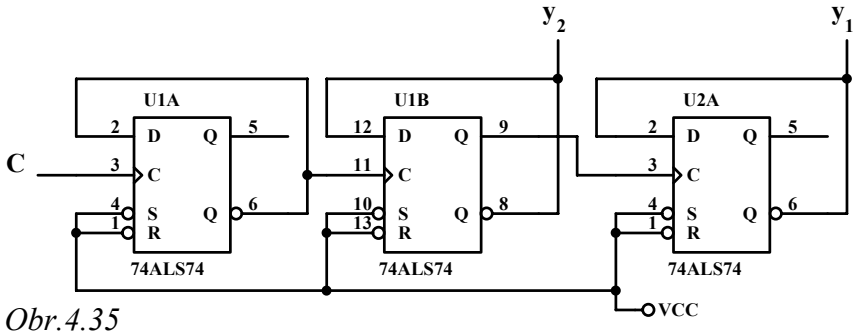
Příklad 4.1.5. Odvoďte stavový diagram obvodu z obr.4.35 a určete nejvyšší hodinový kmito-



Obr.4.33

Obr.4.34

čet zajišťující spolehlivou činnost pro paměťové členy D typy 74LS74 ($t_{pHL(C\to Q)} \le 40[ns]$, $t_{pLH(C\to Q)} \le 25[ns]$, $t_{setup} \ge 20[ns]$).



Obr.4.35

4.2. Návrh logických sekvenčních obvodů

Návrh sekvenčních obvodů je podstatně náročnější než návrh logických kombinačních obvodů, optimalizace navrhovaného obvodu se provádí ve třech fázích. Protože v jednotlivých fázích návrhu můžeme optimalizací vytvořit nepříznivé podmínky pro fáze následující, nelze jednotlivé fáze návrhu absolutně optimalizovat. Obecná metoda návrhu sekvenčních obvodů

vedoucí k nejlepší realizaci obvodu s předepsanými vlastnostmi z obvodů dané třídy nebyla vypracována. Úloha se obvykle řeší tak, že optimální obvod se vybírá z několika návrhů u kterých byla při návrhu modifikována kritéria optimálnosti v jednotlivých fázích návrhu. Postup návrhu sekvenčního obvodu lze rozdělit do následujících šesti kroků:

1) Zadané chování logického sekvenčního obvodu převedeme do formy vývojové tabulky. Při převodu dbáme na to, aby výstupní stavy byly jednoznačnou funkcí vstupních a vnitřních stavů obvodu a vnitřní stavy obvodu byly vzájemně rozlišeny pro všechny posloupnosti vstupních stavů významných z hlediska zadané úlohy.

2) První fáze minimalizace navrhovaného obvodu. V neredukované vývojové tabulce, kterou jsme získali v prvém kroku návrhu, minimalizujeme počet vnitřních stavů obvodu tím, že vyhledáváme ekvivalentní a slučitelné vnitřní stavy, sloučíme a získáme tak redukovanou formu vývojové tabulky obvodu.

3) Druhá fáze optimalizace navrhovaného obvodu je kódování vnitřních stavů. Zvolíme počet vnitřních proměnných tak, aby počet vnitřních stavů $Z = 2^n$, kde n je počet vnitřních proměnných. Jednotlivým vnitřním stavům přiřadíme kombinace vnitřních proměnných tak, aby byla zajištěna spolehlivá činnost navrhovaného obvodu tj. aby byl vyloučen vznik souběhů a cyklů u asynchronních obvodů a co nejlépe se vyhovělo optimalizačním hlediskům.

4) Zakódováním vývojové tabulky získáme tabulku stavovou, z které získáme minimální výrazy reprezentující funkce přechodů a výstupů sekvenčního obvodu. V této fázi návrhu je třeba u hladinových asynchronních obvodů provést odvození výrazů s ohledem na hazardní stavy v kombinační části obvodu. U logických sekvenčních obvodů s paměťovými členy odvodíme z funkcí přechodů za pomoci operátoru paměťového členu funkce pro buzení jejich vstupů, které vyjádříme minimálními algebraickými výrazy.

5) Výrazy pro funkce přechodů a výstupů nebo funkce pro buzení vstupů paměťových členů získané v bodě 4 obvodově realizujeme.

6) Správnost návrhu můžeme ověřit analýzou navrženého zapojení.

4.2.1. Převod zadaného chování obvodu do vývojové tabulky

V prvém kroku návrhu sekvenčního obvodu je třeba převést slovní popis chování obvodu do vývojové nebo fázové tabulky, která vyčerpávajícím způsobem popisuje chování (funkci) obvodu a má formu vhodnou pro systematickou redukci počtu vnitřních stavů. Vývojovou tabulku lze vytvořit časově méně náročným intuitivním způsobem, při kterém se můžeme dopustit chyb, nebo zdlouhavějším systematickým způsobem. Při návrhu nejprve určíme množinu vstupních $\vec{x}$ a výstupních $\vec{y}$ stavů a chování sekvenčního obvodu převedeme do tzv. normálního tvaru, kde jsou určeny vztahy mezi posloupností vstupních stavů $\vec{x}$ a odpovídající

posloupností výstupních stavů $\vec{y}$. Kombinace vstupních proměnných, pro které není definována kombinace výstupních proměnných se označují jako zakázané (nepřípustné) vstupní slovo.

Vývojovou tabulku nejsnáze vytvoříme tak, že každé přípustné dvojici vstupních $\vec{x}_i$ a výstupních $\vec{y}_j$ stavů přiřadíme jeden vnitřní stav obvodu z_k. To znamená, že všechny stavy systému ztotožníme s vnitřními stavy. Jeden z vnitřních stavů zvolíme jako počáteční a vycházejíce z něho vytváříme vývojovou nebo fázovou tabulku tak, že stanovíme všechny přechody mezi vnitřními stavy o souhlase se zadaným chováním obvodu. Při tvorbě tabulky vyloučíme nepřípustná vstupní slova a respektujeme zakázané změny vstupních slov.

Příklad 4.5 ***Odvoďte fázovou tabulku k asynchronnímu obvodu majícímu dva vstupy*** x_1 ***a*** x_2 ***a jeden výstup y. Vstupní proměnné*** x_2x_1 ***reprezentují binární čísla od 0 do 3. Pokud změna vstupního čísla*** x_2x_1 ***představuje zvětšení o jedničku, nechť y = 1. Při zmenšení o jedničku, nechť y = 0. Jakákoliv jiná změna vstupního čísla nezpůsobí změnu výstupního stavu.***

Jak vyplývá z předcházejícího textu, ztotožníme celkové stavy systému 2^{2+1}=8 s vnitřními stavy tab.4.7. Podle zadání jsou přípustné všechny vstupní i výstupní stavy a též i všechny změny vstupních stavů. Za výchozí stav obvodu zvolíme např. stav 1 a zapíšeme první řádek fázové tabulky 4.8 v souhlase se zadáním. To znamená, že např. do prvého řádku zapíšeme pro $x_2x_1 = 10$ (změna z 0→2, výstup se nemění) stav 5 , pro $x_2x_1 = 01$ (změna z 0→1, y = 1) stav 4 a pro $x_2x_1 = 11$ stav 7. V dalších řádcích můžeme vycházet ze stavů, které se v tabulce objevily nebo je brát v libovolném pořadí. Fázová tabulka, která se používá k popisu asynchronních sekvenčních obvodů s hladinovým řízením, má (v neredukovaném tvaru) v každém řádku jeden stabilní stav odpovídající příslušné fázi sekvenčního obvodu.

Vstupní proměnné	x_1	0	0	1	1	0	0	1	1
	x_2	0	0	0	0	1	1	1	1
Výstup	y	0	1	0	1	0	1	0	1
Vnitřní stav		1	2	3	4	5	6	7	8

Tabulka 4.7

J.Podlešák, P.Skalický - ***Spínací a číslicová technika,*** *Praha 1994*

Příklad 4.6 ***Odvoďte vývojový diagram a vývojovou tabulku pro sekvenční obvod se dvěma impulzními vstupy*** x_1 a x_2 ***a jedním impulzním výstupem y, Impulz y nechť je v koincidenci s druhým ze dvou po sobě jdoucích impulzů*** x_2 ***následujících po dvou impulzech*** x_1**.**

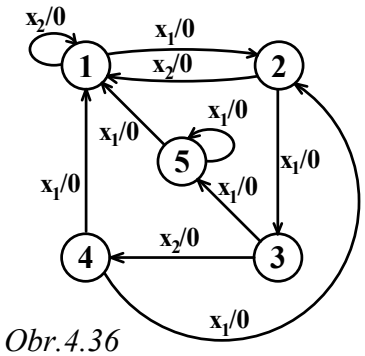
x_1x_2 0 0	0 1	1 1	1 0	y
①	5	7	4	0
②	6	8	4	1
1	6	7	③	0
1	6	8	④	1
1	5	⑦	3	0
2	5	⑧	4	1
1	⑤	8	3	0
2	⑥	8	3	1

Tabulka 4.8

Svými vnitřními stavy musí takovýto detektor posloupnosti rozlišit vstupní posloupnost podstatnou pro jeho činnost. Předpokládáme, že obvod se nachází v počátečním stavu 1 ze kterého bude procházet posloupností vnitřních stavů, které zajistí detekci požadované posloupnosti $x_1x_1x_2$. Z počátečního stavu 1 musí obvod s impulzem x_1 přejít do stavu 2, z kterého s dalším impulzem x_1 přejde do stavu 3. Ze stavu 3 na impulz x_2 přejde do stavu 4 a s druhým impulzem x_2 opět do počátečního stavu 1 při výstupu y = 1. Tento nezbytný počet vnitřních stavů k detekci požadované posloupnosti je třeba rozšířit ještě o stav 5, kterým odlišíme dva od více po sobě jdoucích impulzů x_1. Vývojový diagram je na obr.4.36 a odpovídající vývojová tabulka v tab.4.9.

z^i	x_1	x_2x_2
1	2/0	1/0
2	3/0	1/0
3	5/0	4/0
4	2/0	1/1
5	5/0	1/0

Tabulka 4.9



Obr.4.36

4.2.2. Redukce vývojové tabulky

Dalším krokem v návrhu logického sekvenčního obvodu je minimalizace počtu řádků vývojové nebo fázové tabulky získané v prvním kroku návrhu. Protože každý řádek bude později definován určitou kombinací stavů paměťových elementů, je z hlediska ekonomie návrhu důležité snížení počtu řádků (stavů obvodu). Se snížením počtu stavů se zmenší i počet vnitřních proměnných (paměťových elementů) a většinou se druhotně sníží i složitost kombinační logiky ve výsledném obvodu. Existují však výjimky, u kterých se přidáním paměťových členů redukuje kombinační logika ve výsledném obvodu. Jen zřídka je redukce taková, že vykompenzuje nebo převýší přidané (nadbytečné) paměťové elementy.

Úkolem redukce vývojové nebo fázové tabulky je získání tabulky s minimálním počtem řádek, která pro všechny vstupní posloupnosti definuje výstupní posloupnosti stejné jako

neredukovaná tabulka. Existují tři základní cesty jak tohoto záměru dosáhnout. Vývojová tabulka, do které mohou být v průběhu jejího vytváření zavedeny nadbytečné stavy, je testována na ekvivalenci stavů. Dva nebo více ekvivalentních stavů pak může být nahrazeno stavem novým, čímž se redukují řádky vývojové tabulky. U neúplně definovaných vývojových tabulek, kde některé výstupní nebo vnitřní stavy jsou neurčité, lze provést test na slučitelnost stavů. Dva nebo více slučitelných stavů pak může být nahrazeno stavem novým. U fázové tabulky, která obsahuje v každém řádku pouze jediný stabilní stav, lze slučovat řádky, které mají ve všech sloupcích slučitelné stavy. V jednom řádku pak mohou být dva nebo více stabilních stavů, které se při stejném stavu vnitřních proměnných rozlišují stavy vstupních proměnných.

Redukce vývojové tabulky slučováním ekvivalentních vnitřních stavů se uplatňuje u úplně definované tabulky. Dva vnitřní stavy jsou ekvivalentní $z_k \equiv z_j$, jestliže pro libovolně dlouhou posloupnost vstupních stavů je výstupní posloupnost stejná, ať výchozím stavem je stav z_k nebo z_j tzn., že

$$\vec{y}^{\,i} = f\left(\vec{x}^{\,i}, \vec{x}^{\,i-1}, \ldots, \vec{x}^{\,-1}, \vec{z}_k^{\,1}\right) = f\left(\vec{x}^{\,i}, \vec{x}^{\,i-1}, \ldots, \vec{x}^{\,-1}, \vec{z}_j^{\,1}\right) \qquad (4.29)$$

kde $i \geq 1$. Ze vztahu (4.29) pak vyplývá nutnost shody funkcí přechodů a výstupů, což můžeme vyjádřit vztahy

$$f\left(\vec{x}^{\,i}, \vec{z}_j^{\,i}\right) = \vec{y}^{\,i} = f\left(\vec{x}^{\,i}, \vec{z}_k^{\,i}\right) \qquad (4.30)$$

$$\vec{z}_j^{\,i+1} = g\left(\vec{x}^{\,i}, \vec{z}_j^{\,i}\right) = \vec{z}_k^{\,i+1} = g\left(\vec{x}^{\,i}, \vec{z}_k^{\,i}\right), \qquad (4.31)$$

které musí být splněny pro všechny vstupní stavy. Ze vztahu (4.30) vyplývá nutná, nikoliv postačující podmínka ekvivalentnosti stavů z_k a z_j (tj. shoda výstupních stavů). Ze vztahu (4.31) pak vyplývá, že vzájemně ekvivalentní stavy přechází pouze do stejných nebo dalších vzájemně ekvivalentních stavů. Jestliže tomu tak není (pro určité i), pak z_k a z_j nejsou ekvivalentní. V tabulce 4.10 jsou základní příklady na ekvivalenci dvou stavů 1 a 2 pro vývojové tabulky: a - impulzního obvodu s impulzním výstupem, b - obvodu s hladinovými vstupy i výstupy. V tabulce 4.11 je příklad podmíněné ekvivalence stavů 1 a 2. Stavy 1 a 2 mohou být ekvivalentní jedině tehdy, když stavy 3 a 4 budou ekvivalentní. Protože stavy 3 a 4 jsou nepodmíněně ekvivalentní, jsou i stavy 1 a 2 ekvivalentní.

	x_1	x_2
1	3/0	2/1
2	3/0	2/1

Případ a)

x_1x_2 0 0	0 1	1 1	1 0	y_1y_2
①	3	5	7	0 1
②	3	5	7	0 1

Případ b)

	x_1	x_2

x_1x_2 0 0	0 1	1 1	1 0	y_1y_2

A	3/0	2/1

Ⓐ	3	5	7	0 1

Tabulka 4.10

Pro rozsáhlé vývojové tabulky může určení ekvivalentnosti dvou stavů vytvořit řetězec zacyklených závislostí, z kterých určení ekvivalentnosti nemusí být vždy jednoduché. Proto byla vytvořena systematická metoda pro vyhledávání ekvivalentních stavů - **implikační tabulka**, která umožňuje testovat platnost rovnice (4.31) pro vstupní slova narůstající délky (i=1,2,3,..). Tato tabulka se skládá ze čtverců pro každou dvojici stavů vývojové tabulky. Pokud je dvojice stavů neekvivalentní (nemá stejné výstupní stavy pro všechny vstupní stavy - není splněn vztah (4.30)), pak čtverec jim příslušející označíme symbolem X. Jestliže dvojice stavů splňuje podmínky ekvivalence, pak do příslušného čtverce implikační tabulky zapíšeme všechny dvojice stavů, které musí splňovat podmínku ekvivalence proto, aby testovaná dvojice mohla být ekvivalentní. Tímto způsobem vyplníme všechny čtverce implikační tabulky. Vyhledávání ekvivalentních stavů v tabulce probíhá tak, že zkoumáme platnost podmínek pro ekvivalenci dvou stavů. Zjistíme-li, že dva stavy podmiňující ekvivalentnost stavů daného čtverce tabulky jsou neekvivalentní, pak ani dvojice daného čtverce není ekvivalentní a označíme ji symbolem X. Takto postupně zjistíme všechny neekvivalentní dvojice stavů a zbývající dvojice, které jsou ekvivalentní, můžeme nahradit jedním stavem.

	x_1	x_2	y
1	5	4	0
2	5	3	0
3	4	6	1
4	4	6	1

Tabulka 4.11

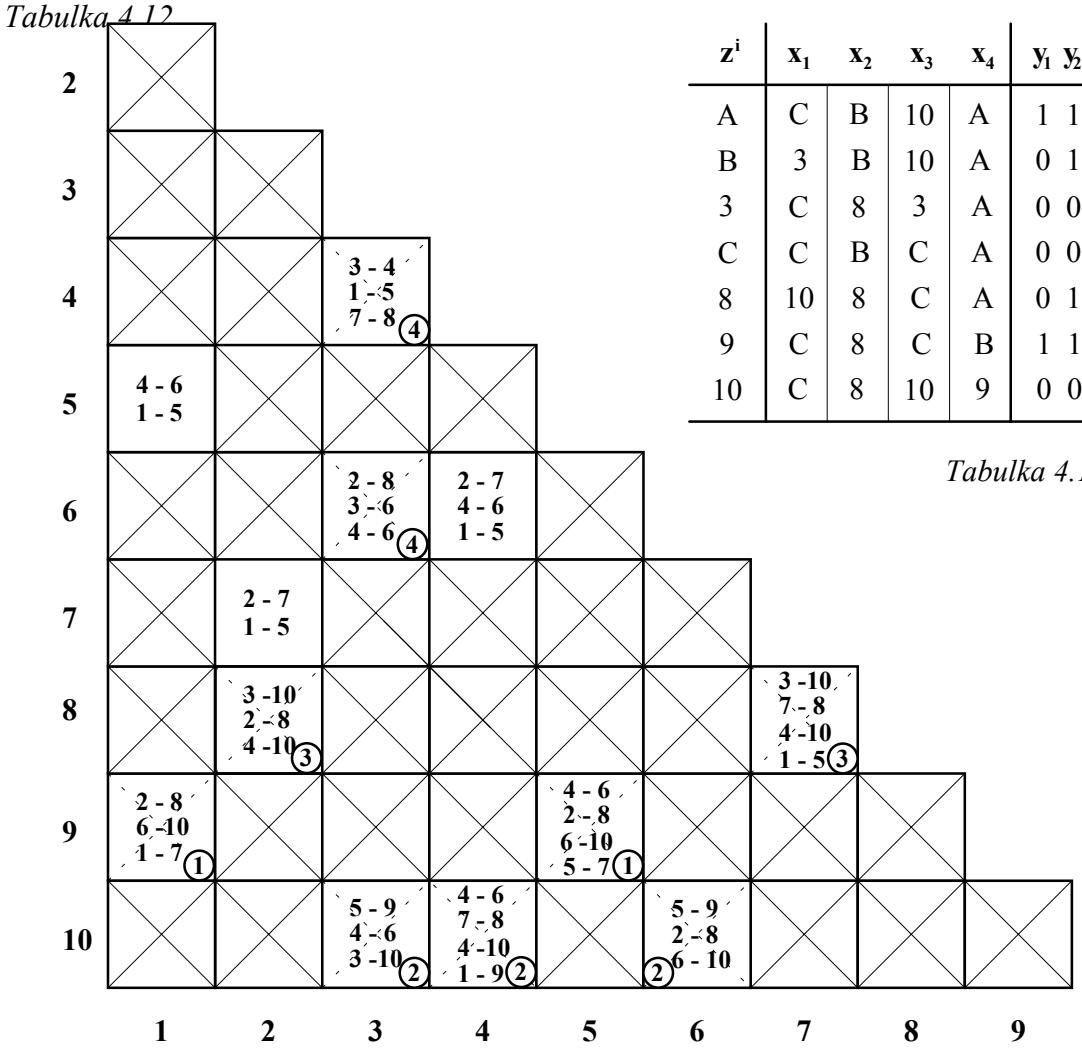
Příklad 4.7 Redukujte vývojovou tabulku impulzního obvodu s hladinovými výstupy z tab. 4.12 pomocí implikační tabulky.

Dříve než odvodíme implikační tabulku vyloučíme případné ekvivalentní stavy vyznačující se stejnými řádky vývojové tabulky. V tabulce 4.12 takové řádky nejsou a proto přistoupíme k vytvoření implikační tabulky s devíti čtverci na každé straně. Tabulku vyplníme křížky pokud stavy nemohou být ekvivalentní (mají různé výstupní stavy) nebo dvojicemi stavů podmiňujících ekvivalenci stavů daného čtverce. V získané implikační tabulce 4.13 začneme vyhledávat neekvivalentní dvojice, které podmiňují ekvivalenci jiných dvou stavů.

z^i	x_1	x_2	x_3	x_4	y_1 y_2
1	4	2	10	1	1 1
2	3	2	10	5	0 1
3	6	8	3	5	0 0
4	6	7	4	1	0 0
5	6	2	10	5	1 1
6	4	2	6	5	0 0
7	3	7	10	1	0 1
8	10	8	4	5	0 1
9	4	8	6	7	1 1
10	4	8	10	9	0 0

Tabulka 4.12

V prvém kroku zjistíme, že v tabulce jsou neekvivalentní dvojice stavů 1 - 7 a 5 - 7, které způsobují neekvivalenci dvojic stavů 1 - 9 a 5 - 9 označených ①. Neekvivalentní dvojice zjištěné v prvém kroku ① způsobují neekvivalenci stavů 3 - 10, 4 - 10 a 6 - 10 označených ②. Dvojice stavů označené ② generují neekvivalentní dvojice 2 - 8 a 7 - 8 označené ③, které generují ve čtvrtém kroku neekvivalentní dvojice 3 - 4 a 6 - 3 označené ④. Zbývající dvojice 1 - 5, 2 - 7 a 4 - 6 jsou ekvivalentní a mohou být nahrazeny jedním stavem A ≡ 1 - 5, B ≡ 2 - 7 a C ≡ 4



Tabulka 4.13

z^i	x_1	x_2	x_3	x_4	y_1 y_2
A	C	B	10	A	1 1
B	3	B	10	A	0 1
3	C	8	3	A	0 0
C	C	B	C	A	0 0
8	10	8	C	A	0 1
9	C	8	C	B	1 1
10	C	8	10	9	0 0

Tabulka 4.14

- 6. Výsledná redukovaná tabulka je dána tabulkou 4.14.

U neúplně definovaných vývojových tabulek, kde některé výstupní nebo vnitřní stavy jsou neurčité, je možné testovat **slučitelnost stavů**. Dva stavy jsou slučitelné, jestliže splňují jednu nebo obě následující podmínky:

a) Pro daný vstupní signál je přechod z jednoho stavu do následujícího stavu předepsán, kdežto z druhého stavu je přechod libovolný.

b) Výstupní stav příslušející jednomu z těchto stavů je předepsán, kdežto u druhého je odpovídající výstupní stav libovolný.

Slučitelné stavy mohou být nahrazeny jedním stavem s tím, že v novém řádku jsou libovolné přechody nebo hodnoty výstupních veličin nahrazeny předepsanými hodnotami ze sloučených stavů. Při slučování více než dvou stavů musí být podmínky slučitelnosti splněny mezi každou dvojicí ze zúčastněných stavů tzn., že stavy A,B a C jsou slučitelné tehdy, když jsou slučitelné dvojice A - B, B - C a A - C. Příklady na sloučení dvou slučitelných stavů 1 a 2, které jsou nahrazeny novým stavem A, jsou v tabulce 4.15, kde a - je příklad neurčité funkce přechodů a b,c - neurčité funkce výstupů.

z^i	x_1	x_2
1	4/0	2/1
2	-/0	2/1

Případ a)

z^i	x_1	x_2
1	4/0	2/1

z^i	x_1	x_2
1	4/-	2/1
2	4/0	2/-

Případ b)

z^i	x_1	x_2
1	4/0	2/1

z^i	x_1	x_2	y_1 y_2
1	4	2	1 0
2	0	1	- 0

Případ c)

z^i	x_1	x_2	y_1 y_2
1	4	2	1 0

Tabulka 4.15

K redukci neúplné vývojové tabulky lze použít implikační tabulku, pomocí níž zjistíme všechny s1učitelné a ekvivalentní třídy stavů. Přehled o slučitelnosti stavů získáme z tzv. **diagramu slučitelnosti**, což je neorientovaný graf, v němž uzly jsou přiřazeny jednotlivým stavům a spojnice vyjadřují slučitelnost.

Příklad 4.8 Redukujte neúplnou vývojovou tabulku impulzního sekvenčního obvodu s impulzním výstupem z tab.4.16.

Vytvoříme nejprve implikační tabulku 4.17 a určíme neekvivalentní dvojice. Přehled o sloučení stavů získáme z diagramu slučitelnosti obr.4.37, který má šest uzlů odpovídajících jednotlivým stavům. Z každého řádku nebo sloupce implikační tabulky zjistíme, zda daný stav je slučitelný či nikoliv s ostatními stavy. Např. pro stav 5 zjistíme, že není slučitelný' se stavy 2,3 a 6 a je slučitelný se stavem 1 a 4 bez jakékoliv podmínky, což vyjádříme v diagramu slučitelnosti spojnicí mezi uzly 1-5 a 4-5. Z obr.4.37 je zřejmé, že lze sloučit stavy 2,3 a 6 a stavy 1,5 nebo 4,5. Z hlediska návrhu obvodu je výhodnější ten případ, u kterého v redukované

5. Aplikace logických sekvenčních obvodů

V aplikacích sekvenčních obvodů se nejčastěji využívají obvody realizující funkci pamatování (registry), funkci čítání (čítače) a funkci posouvání (posuvné registry). Funkce pamatování se rea1izuje paměťovým registrem vytvořeným z N nezávislých paměťových členů (klopných obvodů), které uchovávají (pamatují si) hodnotu jedné dvojkové číslice nebo N logických proměnných. Registry se nejčastěji používají jako rychlé krátkodobé paměti dat, tzv. zásobníkové a vyrovnávací paměti, jako součásti aritmeticko-logických jednotek, řadičů, generátorů posloupností, převodníků sériových kódů na paralelní a naopak, atd. Podle jejich vnitřní struktury a chování je můžeme dělit na registry:

- statické - uložená informace je trvale uschována v jednotlivých paměťových buňkách, které tvoří registr.
- dynamické - uložená informace v registru cirkuluje a postupně prochází všemi paměťovými buňkami.
- posuvné - tvořené statickým registrem zapojeným tak, že umožňuje uloženou informaci posouvat v registru vpravo nebo vlevo. Posun je řízen hodinovými impulzy jedno nebo více fázovými.

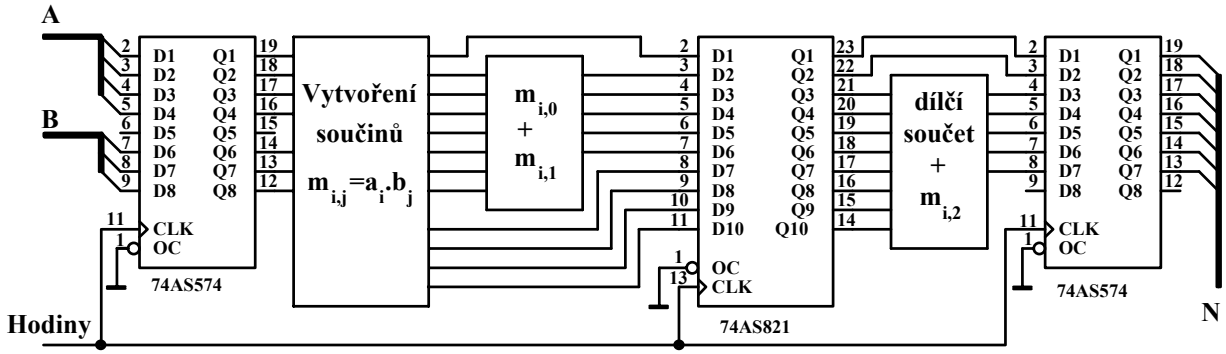
Dříve se k realizaci paměťového registru používaly paměťové členy RS, do kterých bylo možné zapisovat informaci ve dvou nebo jednom taktu. Nyní se paměťové registry realizují, jak vyplývá z tab.3.5 hlavně z klopných obvodů JK a D, které jsou synchronizovány hodinovým signálem.

Registry jako rychlé krátkodobé paměti se používají k **ošetření nestabilních stavů.** Předpokládejme, že informace přicházející po vodičích není po celou dobu jejího zpracování stabilní (platná), což může způsobit špatné vyhodnocení. Vložíme-li do informační cesty paměťový registr do kterého zapíšeme platnou informaci, máme k dispozici stabilní informaci až do začátku dalšího zápisu zpracovávané informace. Tím můžeme překrýt změny způsobené např. přepínáním vstupních signálů multiplexerem. Určitou analogií předcházejícího případu je synchronizace proměnných. Jestliže vstupní proměnné synchronního sekvenčního obvodu jsou vůči hodinovému signálu zcela asynchronní, potom je nezbytné je synchronizovat s hodinovým signálem sekvenčního obvodu. Tím zabráníme vzniku **hazardních stavů** v sekvenčním obvodu v důsledku nedodržení doby předstihu t_{setup} na paměťových členech.

Jestliže lze nějakou operaci rozložit do dvou nebo více dílčích částí, potom můžeme použitím **vyrovnávacího registru** zvýšit výkon zařízení provádějícího tuto operaci.

Příklad 5.1 Stanovte kolikrát lze zvýšit rychlost násobení čtyřbitového čísla A tříbitovým číslem B použitím vyrovnávacího registru k uchování mezivýsledku v paralelní násobičce z části 3.3.

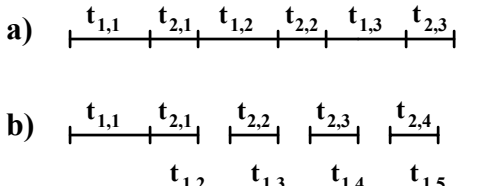
Rychlost násobení paralelní násobičky z části 3.3 je dána součtem doby potřebné k vytvoření součinů $m_{i,j} = a_i . b_j$, kde i = 0,1,2,3 a j = 0,1,2, a doby potřebné k sečtení získaných dílčích součinů. Součiny $m_{i,j}$ se realizují logickými členy AND 74ALS08 a součty



Obr.5.1

dílčích součinů pomocí obvodů 74ALS283. Maximální doba získání součinu je dána výrazem

$$t_{souè} = t_{pLH} + 2.t_{pLH(A_i \to S_i)} = (20 + 2.24).10^{-9} = 68 \ [ns] \qquad (5.1)$$



Obr.5.2

Nevýhodou zapojení je to, že obě čísla musí setrvat na vstupech násobičky po celou dobu výpočtu součinu, což se negativně projevuje na rychlosti. Tuto nevýhodu lze odstranit použitím vyrovnávacích registrů, do kterých uchováme mezivýsledky výpočtu obr.5.1. Řešení umožňuje po získání a uložení mezivýsledku začít výpočet nového mezivýsledku (součinu) a zároveň dokončit výpočet součinu z uloženého mezivýsledku. Na obr.5.2 je zobrazena časová posloupnost operací v násobičce bez vyrovnávacího registru (případ a) a s ním (případ b), kde $t_{1,i}$ je doba potřebná k získání mezivýsledku i-tého součinu a $t_{2,i}$ je doba potřebná k získání výsledku z i-tého mezivýsledku. Jak vyplývá z obr.5.2b je doba mezi dvěma zápisy do vyrovnávacího registru určena nejdéle trvající dílčí operací tj. v tomto případě dobou získání mezivýsledku. K této době musíme ještě přičíst dobu t_{setup} pro použitý registr, aby byla zajištěna jeho správná činnost. Minimální doba mezi zápisy do vyrovnávacího registru je dána vztahem

$$t_{\min} = t_{pLH} + t_{pLH(A_i \to S_i)} + t_{setup} = (20 + 24 + 3).10^{-9} = 47 \ [ns] \qquad (5.2)$$

Ačkoliv faktická doba od vstupu násobených hodnot po získání jejich součinu bude $2.t_{min}$, zvýší se rychlost násobení čísel A a B, protože za každou dobu t_{min} můžeme získat jeden výsledek součinu. Rychlost násobičky se zvýší v poměru

$$p = t_{souè} / t_{\min} = 68/47 \approx 1,44 \qquad (5.3)$$

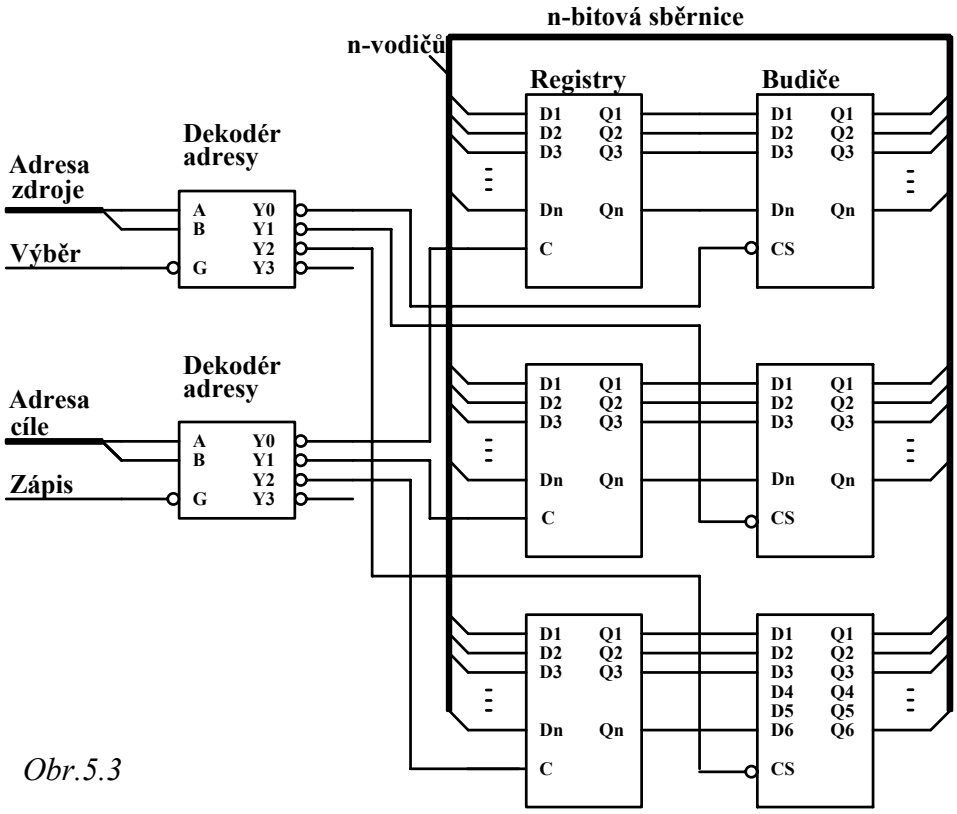
V případě použití dvou vyrovnávacích registrů, z nichž druhý by byl zařazen za obvod vytvářející součiny $m_{i,j}$, by hodnota t_{min} byla dána

$$t_{\min} = t_{pLH(A_i \to S_i)} + t_{setup} = (24+3).10^{-9} = 27 \ [ns] \qquad (5.4)$$

Poměr by se pak zvýšil na hodnotu p = 2,50.

Rozložení operací do dílčích částí, které se díky vyrovnávacím registrům překrývají, se využívá u procesorů k zvýšení jejich výkonu. U současných signálových procesorů se používá čtyř úrovňové překrývání (pipeline) jednotlivých instrukcí (čtení instrukce, její dekódování, čtení operandu a vykonání instrukce), u procesorů RISC (s redukovaným instrukčním souborem) je překrývání pětinásobné až desetinásobné. Obvody s tak velkým překrýváním jsou někdy označovány pojmem **průtoková struktura**.

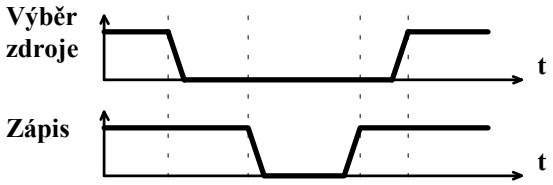
Při návrhu složitých logických ce1ků stojíme často před problémem spojení řady paměťových registrů, mezi nimiž má být přenášena informace. Spojení mezi registry lze realizovat pomocí multiplexerů nebo pomocí **sběrnice**. Přenos pomocí sběrnice obr.5.3 předpokládá, že na výstupy registru jsou zařazeny oddělovací stupně s třístavovým výstupem nebo



Obr.5.3

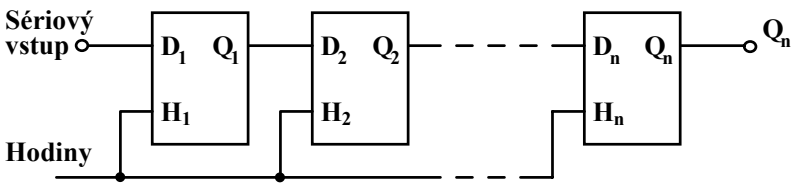
s výstupem s otevřeným kolektorem. Při prvém řešení jsou výstupy registrů přivedeny na kombinační obvody (např. multiplexery), které podle stavu vstupních proměnných (podmínek) vyberou výstupy požadovaného registru a spojí je se vstupy registrů do nichž se informace má

přenést. Podmínky, za kterých se má přenos uskutečnit vyhodnocují kombinační obvody, které ovládají zapisovací vstupy registrů. Nevýhodou řešení je značná složitost a velké množství propojovacích vodičů. Výhodou je možnost přenosu informace mezi několika dvojicemi registrů najednou. Při druhém řešení jsou výstupy registrů, které nemají třístavový výstup, připojeny přes budiče s třístavovým výstupem ke společné sběrnici, která je připojena ke vstupům všech registrů. Obvod je vybaven dvěma adreso-vými dekodéry, které v závislosti na kombi-naci vstupních proměnných (na adrese) a signálu pro výběr aktivují budič zdroje informace a vstup pro zápis do cílového registru. Tento způsob je z hlediska množství obvodů i propojovacích vodičů hospodárný, ale umožňuje v každém okamžiku přenos pouze mezi jedinou dvojicí registrů. Časový diagram na obr.5.4 ukazuje, že výběr zdroje překrývá na obou stranách zápis do cílového registru, čímž se přechodové fáze při nástupu a ukončení informace na sběrnici posu-nou mimo okamžik zápisu. Tento princip platí všeobecně. Hodnota adresy cíle i zdroje musí být po celou dobu výběru stabilní. V systémech se sběrnicemi lze do přenosové trasy začlenit obvody umožňující při přenosu informace mezi registry provedení některých aritmetických operací. Jako příklad uveďme realizaci komplementu (inverze) registru (pomocí obvodů EX-OR), posunu informace v registru (využívané v registrech aritmeticko-logických jednotek při operacích náso-bení a dělení), inkrementace a dekrementace obsahu registru, nastavení a nulování registru (prostřednictvím zvláštních vstupů) nebo zápisem požadované konstanty do příslušného registru, atd.



Obr.5.4

Funkce posouvání se realizuje posuvným registrem vytvořeným z n paměťových členů, které na každý hodinový impulz přesunou svůj obsah a_j do vedlejšího paměťového č1enu. Posuvný registr obr.5.5 se vytvoří sériovým řazením paměťových členů D (JK) tak, že se spojí výstup Q_j se vstupem D_{j+1} ($J_{j+1} = Q_j$ a $K_{j+1} = \overline{Q}_j$) a všechny hodinové vstupy se spojí. Vstup prvého č1enu se nazývá **sériový vstup**, výstup posledního stupně se nazývá **sériový výstup**. K realizaci funkce posunu, vyjma aplikací realizovaných programovatelnými obvody, se používají skoro výhradně integrované posuvné registry, které jsou shrnuty v tabulce 5.1.



Obr.5.5

U integrovaných registrů s větší délkou než 16 bitů je vyveden pouze sériový vstup, výstup a pomocné mezistupně nejsou přístupné. U registrů kratší délky se pak setkáváme s typy, které mají přístupné nejen výstupy

mezistupňů, ale i jejich vstupy, kterými lze nahrát informaci do registru paralelně. Posuvné registry se používají v generátorech binárních posloupností, v aritmetických jednotkách, převodnících paralelního kódu na kód sériový a naopak, atd.

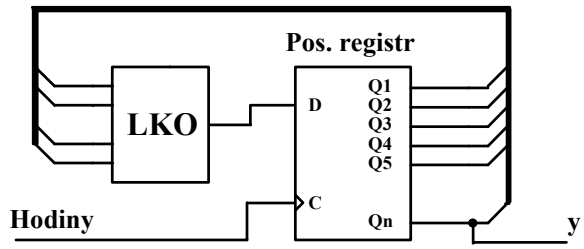
Obvod	Bitů	Vývodů	Vstupy	Výstupy	Směr	Latch	Nulov.	Výstup
74LS91	8	14	2S	2S	P	---	---	2.stav.
74AS95	4	14	P/S	P	P	---	---	2.stav.
74LS96	5	16	P/S	P	P	---	Asyn.	2.stav.
74ALS164	8	14	2S	P	P	---	Asyn.	2.stav.
74ALS165	8	16	P/S	2S	P	---	---	2.stav.
74ALS166	8	16	P/S	S	P	---	Asyn.	2.stav.
74AS194	4	16	P/S	P	P/L	---	Asyn.	2.stav.
74AS195	4	16	P/JK	P	P	---	Asyn.	2.stav.
74AS295	4	14	P/S	P	P	---	---	3.stav.
74AS299	8	20	P/S	S/P	P/L	---	Asyn.	3.stav.
74LS322	8	20	P/S	S/P	P	---	Asyn.	3.stav
74AS323	8	20	P/S	S/P	P/L	---	Synch.	3.stav.
74LS395	4	16	P/S	P/S	P/L	O	Synch.	3.stav.
74LS589	8	16	P/S	S	P	I	---	3.stav.
74LS594	8	16	S	P/S	P	O	2Asyn.	2.stav.
74LS595	8	16	S	P/S	P	O	Asyn.	3.stav.
74LS596	8	16	S	P/S	P	O	Asyn.	OK
74LS597	8	16	P/S	S	P	I	Asyn.	2.stav.
74LS598	8	20	P/2S	S/P	P	I	Asyn.	3.stav.
74LS599	8	16	S	P	P	O	Asyn.	OK
74LS671	4	20	P	P	P/L	O	Asyn.	3.stav.
74LS672	4	20	P	P	P/L	O	Synch.	3.stav
74LS673	16	24	S	P/S	P	O	Asyn.	2.stav.
74LS674	16	24	P/S	S	P	---	---	3.stav
74LS675	16	24	S	P/S	P	O	---	2.stav.
74LS676	16	24	P	S	P	---	---	2.stav.

Tabulka 5.1

kde značí P-paralelní a S-seriové vstupy nebo výstupy, směr posunu P-vpravo a L-vlevo a registrované I-vstupy a O-výstupy. Je-li ve sloupcích vstupy P/S a výstupy S/P, potom se jedná o obvod se společnými (I/O) vstupně/výstupními vodiči.

Příklad 5.2 Navrhněte generátor binární posloupnosti 1000110010110 pomocí posuvného registru.

Generátory binárních posloupností se skládají z posuvného registru se zpětnou vazbou obr.5.6. Požadovanou časovou posloupnost vytváří výstup posuvného registru synchronně s posouvacími impulzy. Zpětnou vazbu realizuje logický kombinační obvod, který dekóduje vnitřní stav posuvného registru a vytváří novou vstupní hodnotu pro sériový vstup tzn.



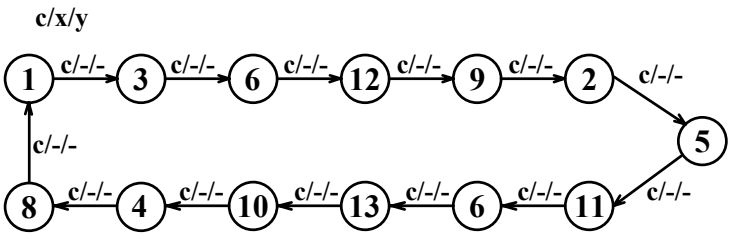
Obr.5.6

$$D_1 = f(Q_1, Q_2, \ldots, Q_N) \qquad (5.5)$$

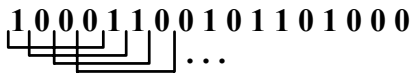
Funkci generátoru lze vyjádřit stavovým diagramem, u kterého se přechody uskutečňují do stavů $S^{i+1} = (2.S^i)_{mod2} + D^i$, kde D^i je hodnota na sériovém vstupu. Hodnota $2.S^i$ je dvojnásobek dekadického stavu S^i, který získáme posunem všech řádů o jednu pozici doleva (k vyššímu řádu). Pro hodnotu $(2.S^i)_{mod2}$ můžeme psát

$$\begin{aligned}(2.S^i)_{mod2} &= 2.S^i \quad pro \quad S^i \le 2^{n-1} - 1 \\ &= 2.S^i - 2^n \quad pro \quad S^i \ge 2^{n-1}\end{aligned} \qquad (5.6)$$

U generátoru nejprve určíme nezbytnou délku n posuvného registru, kterou stanovíme z délky generované posloupnosti d ($2^{n-1} < d \le 2^n$). Posloupnost 1000110010110 má délku 13 a proto

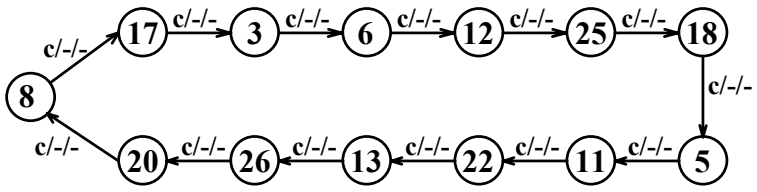


Obr.5.7

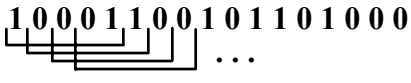


Obr.5.8

volíme n = 4. Generovanou posloupnost rozdělíme na posloupnost n-bitových (4 bitových)

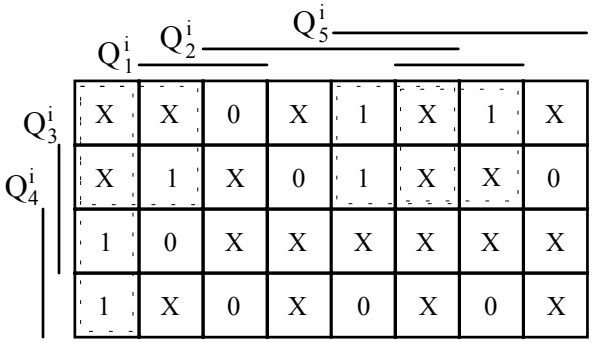


Obr.5.9



Obr.5.10

dvojkových čísel (stavů posuvného registru) a odvodíme stavový diagram obvodu obr.5.7, kde jednotlivé stavy vyjádříme dekadickým ekvivalentem dvojkového čísla (stavu registru $Q_4Q_3Q_2Q_1$).V odvozeném diagramu se nám dvakrát objevuje stav 6, z



Obr.5.11

kterého je předepsán přechod jednou do stavu 12 a po druhé do stavu 13. Tyto přechody je třeba od sebe rozlišit, což obvykle provádíme prodloužením nezbytné délky posuvného registru z n na n+1. Vytvoříme nový diagram přechodů pro n = 5 obr.5.9. Protože stavový diagram má již jednoznačné přechody mezi jednotlivými stavy, nemusíme dále prodlužovat délku posuvného registru a odvodíme funkci pro buzení jeho sériového vstupu. Funkci zapíšeme do Karnaughovy mapy obr.5.11, z které odvodíme

$$D_0 = \overline{Q_5}.\overline{Q_4}.\overline{Q_2} + \overline{Q_5}.\overline{Q_2}.\overline{Q_1} + Q_5.\overline{Q_4}.Q_2 + Q_5.\overline{Q_4}.Q_1 \qquad (5.7)$$

Za předpokladu, že generátor nebude uveden do počátečního stavu nastavovacím signálem, musíme zjistit jaké přechody realizuje odvozená funkce pro nepoužité stavy posuvného registru. Protože ze všech nevyužitých stavů se obvod dostane do použitých stavů nejpozději po sedmi hodinových impulzech, můžeme přistoupit k jeho realizaci. Vstupní soubor GENER.PDS přináší vstupní data pro realizaci generátoru pomocí obvodu PAL16R8.

Soubor GENER.PDS

```
TITLE       Generátor posloupnosti
PATTERN     01.
REVISION    01_def
AUTHOR      Skalický
COMPANY     ČVUT FEL
DATE        9.3.1994

CHIP GENER PAL16R8
;PIN  1    2    3    4    5    6    7    8    9    10
      CLK  NC   NC   NC   NC   NC   NC   NC   NC   GND

;PIN  11   12   13   14   15   16   17   18   19   20
      OE   Q1   Q2   Q3   Q4   Q5   NC   NC   NC   VCC

;Q5,Q4,Q3,Q2 a Q1 určují vnitřní stav generátoru, OE - Aktivace výstupů obvodu = log.0

EQUATIONS
Q5:=Q4      Q4:=Q3      Q3:=Q2      Q2:=Q1
Q1:=/Q5*/Q4*/Q2+/Q5*/Q2*/Q1+Q5*/Q4*Q2+Q5*/Q4*Q1

SIMULATION
TRACE_ON  CLK  Q5  Q4  Q3  Q2  Q1
```

```
SETF    /CLK  /OE
PRLDF   /Q5 /Q4 /Q3 /Q2 /Q1
FOR I:=1 TO 20 DO
BEGIN
CLOCKF  CLK
END
TRACE_OFF
```

Posuvné registry s lineární zpětnou vazbou (generátory cyklického kódu) můžeme využít ke generování posloupností, které mají podobné vlastnosti jako náhodné signály. Pravděpodobnost výskytu nul P(0) a jedniček P(1) v posloupnosti je stejná a autokorelační funkce posloupnosti v rámci své periody má jeden výrazný extrém, jako autokorelační funkce náhodného signálu. Zpětnou vazbu generátorů pseudonáhodných posloupností lze vyjádřit vztahem viz. kap.5.1

$$D_1 = C_1.Q_1 \oplus C_2.Q_2 \oplus \quad \ldots \quad \oplus C_n.Q_n \qquad (5.8)$$

kde C = 0 nebo 1 a Q_j je vnitřní proměnná posuvného registru (výstup paměťového členu). V literatuře [2] lze nalézt funkce (nedělitelné generující polynomy s periodou $2^n - 1$) zajišťující generování pseudonáhodných posloupností s posuvnými registry délky $n \leq 16$. Tyto funkce zajišťují generování posloupností s maximální délkou $2^n - 1$ (periodou $2^n - 1$) s tím, že stav 0 ($Q_n = Q_{n-1} = \ldots Q_1 = 0$) není dovolen. Jak vyplývá z rovnice (5.22) je stav 0 uzavřen sám na sebe. Generování pseudonáhodných posloupností se používá nejen v některých počítačích ke generování náhodných čísel, ale i ke kódování přenášené sériové posloupnosti. Kódování pomocí pseudonáhodné posloupnosti se provádí pomocí operace EX-OR mezi přenášeným bitem a kterýmkoliv výstupem posuvného registru (i sériovým vstupem) generujícího pseudonáhodnou posloupnost. Pro i-tý kódovaný bit můžeme psát $k_i = m_i \oplus Q_{j,i}$, kde m_i je i-tý přenášený bit a $Q_{j,i}$ je i-tý bit výstupu Q_j posuvného registru. Dekódování této posloupnosti se provádí opět generátorem pseudonáhodné posloupnosti se stejnou délkou a se stejným počátečním stavem jako na vysílací straně pomocí operace EX-OR. Pro i-tý dekódovaný bit dostaneme

$$p_i = k_i \oplus Q_{j,i} = m_i \oplus Q_{j,i} \oplus Q_{j,i} = m_i \oplus 0 = m_i \qquad (5.9)$$

K popsaným realizacím je třeba uvést, že k zajištění správné činnosti je třeba dodržet časové parametry obvodů garantované výrobcem. Dodržení těchto parametrů však nemusí zajistit ještě správnou činnost obvodu, jestliže neuvážíme zpoždění způsobená rozvodem signálů na plošném spoji, zvláště při vysokých hodinových kmitočtech. Z tohoto důvodu by měl být pro posuvné registry hodinový signál od svého zdroje rozveden tak, aby nejprve dosáhl posledního a potom postupně předcházející stupně posuvných registrů (paměťových členů).

5.1. Cyklické kódy

Cyklickým kódem nazýváme takový lineární kód, u kterého pro každé kódové slovo $v_n, v_{n-1}, \ldots, v_2, v_1$ platí, že i cyklicky posunuté slovo $v_1, v_n, v_{n-1}, \ldots, v_2$ je slovem kódovým. Definujeme-li operátor zpoždění D tak, aby platilo $D^{-1}.x(t) = x(t-1)$ a $D^{-k}.x(t) = x(t-k)$, můžeme zapisovat kódová slova $v_n, v_{n-1}, \ldots, v_2, v_1$ ve tvaru polynomů takto [16]

$$v(D) = v_n \oplus v_{n-1}.D^{-1} \oplus \ldots \oplus v_2.D^{-n+2} \oplus v_1.D^{-n+1} \qquad (5.10)$$

Z definice cyklického kódů vyplývá, že i $v(D)$, $D^{-1}.v(D)$, $D^{-2}.v(D), \ldots, D^{-k+1}.v(D)$ jsou kódová slova s tím, že $D^{-n} = 1$

$$D^{-1}.v(D) = v_1 \oplus v_n.D^{-1} \oplus v_{n-1}.D^{-2} \oplus \ldots \oplus v_2.D^{-n+1} \qquad (5.11)$$

Z uvedeného také vyplývá, že kódová slova cyklického kódu jsou tvořena lineární kombinací polynomů $v(D)$, $D^{-1}.v(D)$, $D^{-2}.v(D), \ldots, D^{-k+1}.v(D)$, které vytváří bázi tohoto kódu.

Příklad 5.3 Odvoďte generující matici cyklického kódu celkové parity (4,3).

Nejprve vyjádříme všechna kódová slova tohoto kódu a pro vysvětlení určité vlastnosti je vyjádříme i polynomy takto

Kódové slovo	Polynom	Rozklad polynomu
0 0 0 0	0	$0.(1 \oplus D^{-1})$
1 1 0 0	$1 \oplus D^{-1}$	$1.(1 \oplus D^{-1})$
1 0 1 0	$1 \oplus D^{-2}$	$(1 \oplus D^{-1}).(1 \oplus D^{-1})$
1 0 0 1	$1 \oplus D^{-3}$	$(1 \oplus D^{-1} \oplus D^{-2}).(1 \oplus D^{-1})$
0 1 1 0	$D^{-1} \oplus D^{-2}$	$D^{-1}.(1 \oplus D^{-1})$
0 1 0 1	$D^{-1} \oplus D^{-3}$	$D^{-1}.(1 \oplus D^{-1})^2$
0 0 1 1	$D^{-2} \oplus D^{-3}$	$D^{-2}.(1 \oplus D^{-1})$
1 1 1 1	$1 \oplus D^{-1} \oplus D^{-2} \oplus D^{-3}$	$(1 \oplus D^{-1})^2.(1 \oplus D^{-1})$

Tabulka 5.2

Podrobíme-li vhodné redukci závislých řádků matici vytvořenou z kódových slov, získáme tuto generující matici cyklického kódu

$$G = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \quad \text{tj.} \quad \begin{matrix} 1 \oplus D^{-1} \\ D^{-1}.(1 \oplus D^{-1}) \\ D^{-2}.(1 \oplus D^{-1}) \end{matrix} \qquad (5.12)$$

***Věta** Každý netriviální cyklický (n,k) kód obsahuje polynom g(D) stupně n-k, který má tyto vlastnosti*

a) kód se skládá ze všech násobků polynom $g(D)$

b) polynomy $g(D)$, $D^{-1}.g(D)$,..., $D^{-k+1}.g(D)$ tvoří bázi kódu

c) polynom $g(D)$ dělí polynom $1 \oplus D^{-n}$ beze zbytku

Na základě této věty bude generující matice každého cyklického kódu vypadat takto

$$G = \begin{bmatrix} g_n & g_{n-1} & g_{n-2} & . & . & g_{n-k+1} & 0 & 0 & 0 \\ 0 & g_n & g_{n-1} & g_{n-2} & . & . & g_{n-k+1} & 0 & 0 \\ & : & : & : & & & & & \\ 0 & 0 & 0 & g_n & g_{n-1} & g_{n-2} & . & . & g_{n-k+1} \end{bmatrix} \qquad (5.13)$$

Analogicky jako u lineárních kódů je možné stanovit kontrolní matici a z ní kontrolní polynom, který je dán podílem $(1 \oplus D^{-n}) / g(D)$. Kontrolní polynom $h(D) = (1 \oplus D^{-n}) / g(D)$ je zároveň generujícím polynomem pro duální cyklický kód. Kontrolní matici kódu (n,k) získáme cyklickými posuny koeficientů polynomu čteného od nejnižší mocniny

$$G = \begin{bmatrix} 0 & 0 & 0 & h_k & . & . & h_2 & h_1 & h_0 \\ 0 & 0 & h_k & . & . & h_2 & h_1 & h_0 & 0 \\ & : & : & : & & & & & \\ h_k & . & . & h_2 & h_1 & h_0 & 0 & 0 & 0 \end{bmatrix} \qquad (5.14)$$

Příklad 5.4 Nalezněte všechny binární cyklické kódy v prostoru $1 \oplus D^{-5}$.

Protože polynom $1 \oplus D^{-5}$ má kořen pro D = 1, musí být dělitelný polynomem $1 \oplus D^{-1}$.

$$1 \oplus D^{-5} = (1 \oplus D^{-1}).(1 \oplus D^{-1} \oplus D^{-2} \oplus D^{-3} \oplus D^{-4}) \qquad (5.15)$$

Polynom $(1 \oplus D^{-1} \oplus D^{-2} \oplus D^{-3} \oplus D^{-4})$ již kořen nemá a proto by mohl být dělitelný pouze nedělitelným polynomem druhého stupně $(1 \oplus D^{-1} \oplus D^{-2})$. Protože ani tento polynom nedělí polynom $(1 \oplus D^{-1} \oplus D^{-2} \oplus D^{-3} \oplus D^{-4})$ bez zbytku, existují v prostoru $(1 \oplus D^{-5})$ pouze dva cyklické kódy a to

a) kód celkové parity $g(D) = (1 \oplus D^{-1})$ $h(D) = (1 \oplus D^{-1} \oplus D^{-2} \oplus D^{-3} \oplus D^{-4})$

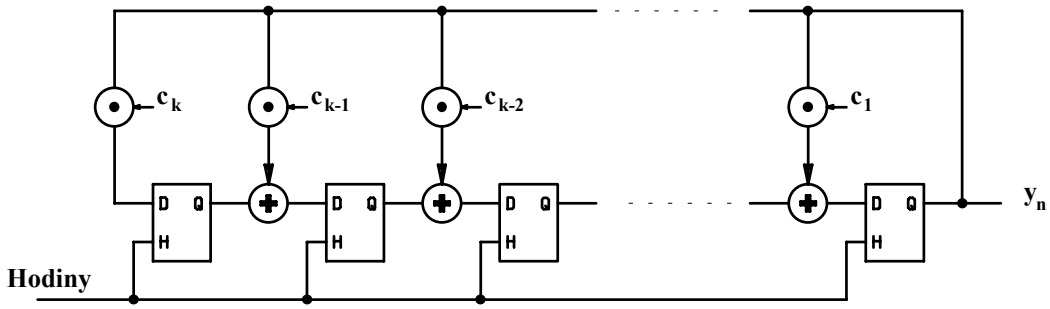
b) opakovací kód $g(D) = (1 \oplus D^{-1} \oplus D^{-2} \oplus D^{-3} \oplus D^{-4})$ $h(D) = (1 \oplus D^{-1})$

Nyní si přiblížíme problematiku kódů z hlediska obvodů, které realizují cyklické kódy. Omezíme se stále pouze na binární kódy. Operaci sčítání budeme realizovat obvody EX-OR, násobení funkcí AND a zpoždění paměťovými obvody D. Obvody lze obecně rozložit na obvody bez vstupu (oscilátory) a na obvody se vstupem, které můžeme použít jako kodéry nebo dekodéry i s možností opravy přenášené informace cyklickým kódem. V literatuře, která pojed-

nává o těchto obvodech, se zavádí tzv. transformace D, která umožňuje převod přenosových vlastností obvodů na polynomy. Transformace D je definována tímto vztahem

$$D\left[k_{n-k}\right] = D^{-k}.Y\left[D\right] \qquad (5.16)$$

Příklad 5.5 Odvoďte zpětnovazební polynom obvodu z obr 5.12.



Obr.5.12

Pro proměnnou y_n snadno ze zapojení odvodíme

$$y_n = a_1 y_{n-1} \oplus a_2 . y_{n-2} \oplus \ldots \oplus a_k . y_{n-k} \qquad (5.17)$$

Aplikujeme-li na tuto rovnici transformaci D, dostaneme pro obraz Y[D] tento vztah

$$\Phi\left[D\right].Y\left[D\right] = \left[1 \oplus a_1 . D^{-1} \oplus a_2 . D^{-2} \oplus \ldots \oplus a_n . D^{-n}\right].Y\left[D\right] \qquad (5.18)$$

kde funkce $\Phi\left[D\right]$ se nazývá zpětnovazební polynom. Jestliže $\Phi\left[D\right]$ není dělitelný hodnotou D^{-1}, pak existují polynomy (prostory) $1 \oplus D^{-i}$, které jsou touto zpětnovazební funkcí dělitelné. Nejmenší hodnota i se nazývá exponent polynomu a určuje jeho periodu. Vydělíme-li tento polynom zpětnovazební funkcí $\Phi\left[D\right]$ dostáváme generující polynom tohoto generátoru (cyklického kódu).

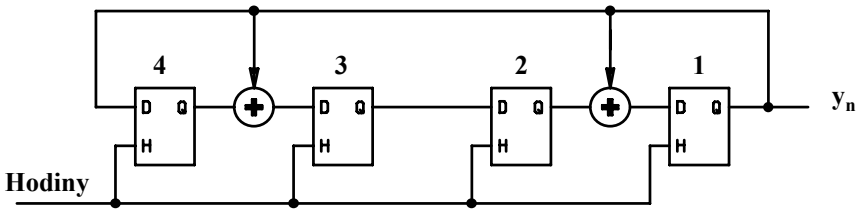
$$g(D) = \left(1 \oplus D^{-i}\right) / \Phi[D] \qquad (5.19)$$

Známe-li generující polynom cyklického kódu, potom všechny jeho generované posloupnosti musí být lineárními kombinacemi polynomů

$$g(D), D^{-1}.g(D), \ldots, D^{-k+1}.g(D) \qquad (5.20)$$

Příklad 5.6 Odvoďte všechny generované posloupnosti obvodu z obr.5.13 a porovnejte jej s klasickou analýzou obvodu se čtyřmi paměťovými obvody typu D.

Pro zpětno-vazební polynom snadno odvodíme, že $F\left[D\right] = 1 \oplus D^{-1} \oplus D^{-3} \oplus D^{-4}$. Tento polynom má periodu 6 tzn., že dělí polynom $1 \oplus D^{-6}$. Provedením podílu získáme generující polynom $g(D) = D^{-2} \oplus D^{-1} \oplus 1$. Pro jednotlivé lineární kombinace polynomů $g(D)$, $D^{-1}.g(D)$, $D^{-2}.g(D)$, $D^{-3}.g(D)$, kterých je celkem 2^4, odvodíme tyto vztahy, z kterých



Obr.5.13

určíme generované posloupnosti tab.5.3. Zbývajících 10 kombinací, které nejsou v tabulce uvedeny, vede na již uvedené posloupnosti.

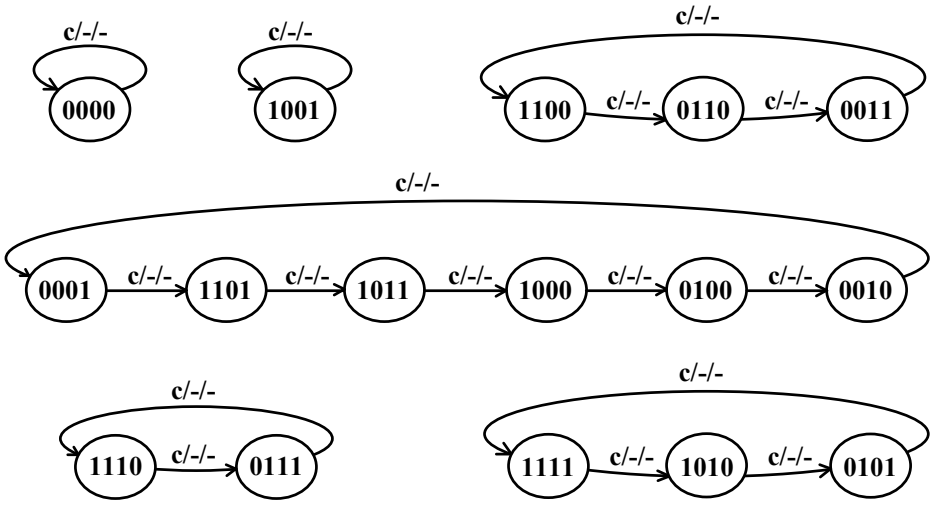
Kombinace	Polynom	Gener.posl.
$g(D).(1)$	$D^{-2} \oplus D^{-1} \oplus 1$	0 0 0 1 1 1
$g(D).(1 \oplus D^{-1})$	$D^{-3} \oplus 1$	0 0 1 0 0 1
$g(D).(1 \oplus D^{-2})$	$D^{-4} \oplus D^{-3} \oplus D^{-1} \oplus 1$	0 1 1 0 0 1
$g(D).(1 \oplus D^{-3})$	$D^{-5} \oplus D^{-4} \oplus D^{-3} \oplus D^{-2} \oplus D^{-1} \oplus 1$	1 1 1 1 1 1
$g(D).(1 \oplus D^{-1} \oplus D^{-2})$	$D^{-4} \oplus D^{-2} \oplus 1$	0 1 0 1 0 1
$g(D).(0)$	0	0 0 0 0 0 0

Tabulka 5.3

Provedeme-li klasickou analýzu tohoto sekvenčního obvodu s tím, že součty z obr.5.13 jsou realizovány funkcí EX-OR dostaneme tyto rovnice přechodů pro jednotlivé paměťové členy D

$$Q_4^{i+1} = Q_1 \quad Q_3^{i+1} = Q_4 \oplus Q_1 \quad Q_2^{i+1} = Q_3 \quad Q_1^{i+1} = Q_2 \oplus Q_1 \qquad (5.21)$$

Pomocí stavové tabulky pak snadno určíme, že obvod má stavový diagram z obr.5.14.



Obr.5.14

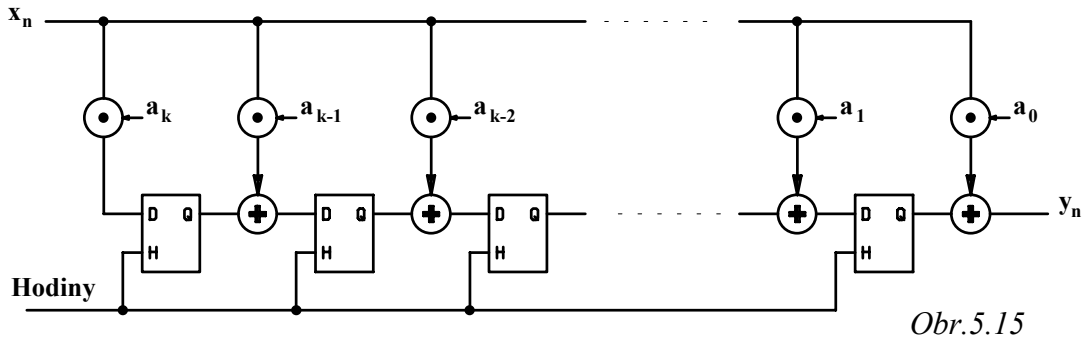
Věta** **Je-li zpětnovazební polynom dělitelný polynomem s periodou $T < T'$, která dělí T beze zbytku, pak generátor generuje i kratší periody.

U našeho případu platí, že $F[D] = 1 \oplus D^{-1} \oplus D^{-3} \oplus D^{-4} = (1 \oplus D^{-2}).(1 \oplus D^{-1} \oplus D^{-2})$, kde polynom $1 \oplus D^{-1} \oplus D^{-2}$ má periodu 3 a polynom $1 \oplus D^{-2}$ má periodu 2. Ten je však

rozložitelný na polynomy $1 \oplus D^{-1}$ s periodou 1. Proto má generátor smyčky mezi šesti stavy (perioda 6), třemi stavy, dvěma stavy i uzavřený stav sám na sebe, kterým je stav 1001. Stav 0000 je uzavřený sám na sebe vždy, když se jedná o lineární generátor.

Druhou skupinu obvodů tvoří lineární obvody, které mají vstup i výstup. U těchto obvodů zavádíme přenos jako podíl obrazu výstupu a vstupu T = Y(D)/X(D). Obvody kódující a dekódující cyklickým kódem lze dále rozdělit na obvody násobící polynomem a dělící polynomem. Na obr.5.15 je obvod realizující násobení polynomem. Výstupní hodnotu y_n lze vyjádřit tímto vztahem

$$y_n = a_0 . x_n \oplus a_1 . x_{n-1} \oplus \ldots \oplus a_k . x_{n-k} \qquad (5.22)$$



Obr.5.15

Po provedení transformace D můžeme pro přenos psát

$$T = Y(D)/X(D) = a_0 \oplus a_1 . D^{-1} \oplus a_2 . D^{-2} \oplus \ldots \oplus a_k . D^{-k} \qquad (5.23)$$

Obvod provádí násobení vstupní posloupnosti $x_n = \eta_0 \eta_1 \eta_2 \ldots \eta_t 0000$ polynomem $a_0 \oplus a_1 . D^{-1} \oplus a_2 . D^{-2} \oplus \ldots \oplus a_k . D^{-k}$ což můžeme vyjádřit výrazem

$$T = Y(D)/X(D) = \left(a_0 \oplus a_1 . D^{-1} \oplus a_2 . D^{-2} \oplus \ldots \oplus a_k . D^{-k}\right).$$
$$\left(\eta_0 \oplus \eta_1 . D^{-1} \oplus \eta_2 . D^{-2} \oplus \ldots \oplus \eta_t . D^{-t}\right) =$$
$$\gamma_0 \oplus \gamma_1 . D^{-1} \oplus \gamma_2 . D^{-2} \oplus \ldots \oplus \gamma_{t+k} . D^{-t-k} \quad \text{kde } \gamma_i = \sum_{\mu+\nu=1} a_\mu . \eta_\nu \qquad (5.24)$$

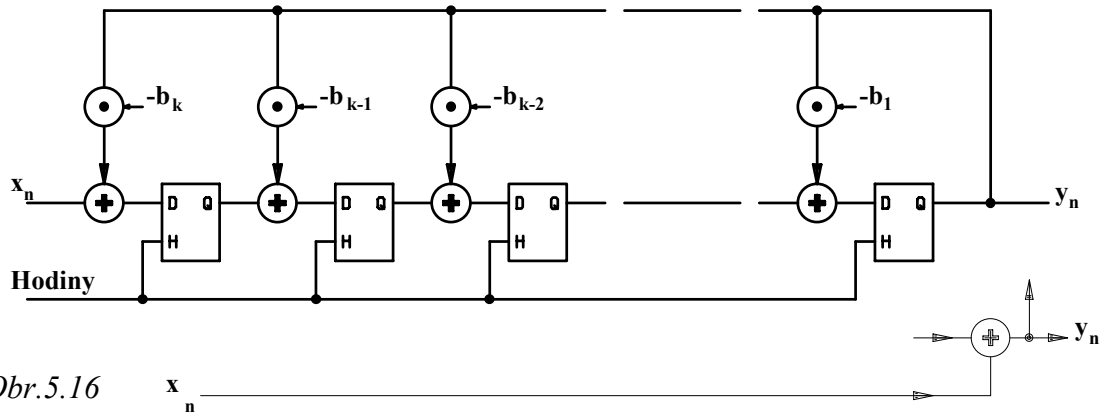
Druhou možností je dělení polynomem, které realizuje obvod z obr.5.16. Výstupní hodnotu y_n lze vyjádřit tímto vztahem

$$y_n = b_1 . y_{n-1} \oplus b_2 . x_{n-2} \oplus \ldots \oplus b_k . x_{n-k} \oplus x_{n-k} \qquad (5.25)$$

Po provedení transformace D můžeme pro přenos psát

$$T = Y(D)/X(D) = \frac{D^{-k}}{1 \oplus b_1 . D^{-1} \oplus b_2 . D^{-2} \oplus \ldots \oplus b_k . D^{-k}} \qquad (5.26)$$

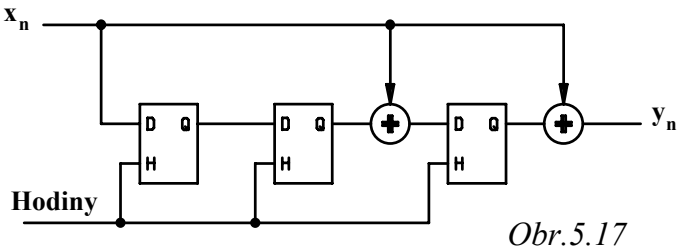
V případě, že zavedeme vstupní signál podle druhého zapojení, bude hodnota D^{-k} nahrazena hodnotou 1.



Obr.5.16

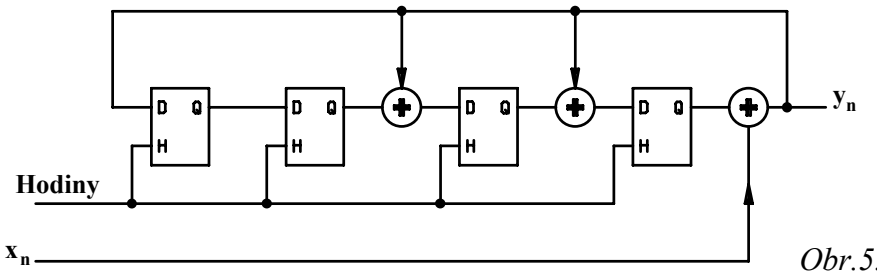
Příklad 5.7 Navrhněte kódovací obvod pro kódování Hamingovým kódem (7,4).

Cyklický Hamingův kód (7,4) má generující polynom $g(D) = 1 \oplus D^{-1} \oplus D^{-3}$ a kontrolní polynom $h(D) = (1 \oplus D^{-1}) / (1 \oplus D^{-1} \oplus D^{-3}) = 1 \oplus D^{-1} \oplus D^{-2} \oplus D^{-4}$. Jestliže zvolíme řešení využívající násobení polynomem získáme obvod z obr.5.17.



Obr.5.17

Jestliže zvolíme řešení využívající dělení polynomem, pak stejnou funkci bude realizovat i obvod z obr.5.18.



Obr.5.18

Příklady k samostatnému řešení.

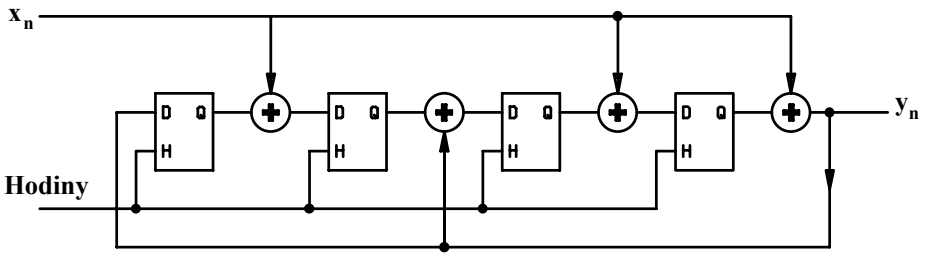
Příklad 5.1.1 Odvoďte zakódovanou posloupnost na výstupu obou obvodů pro vstupní signál 1010\|000.

Příklad 5.1.2 Nakreslete zapojení obou dekodérů pro Hammingův kód (7,4).

Příklad 5.1.3 Zrealizovaným dekodérem z příkladu 5.1.2. dekódujte získanou posloupnost z příkladu 5.1.1. Zkuste v přijímaném slově realizovat jeden chybný bit a po-

rovnejte zbývající stav obvodu po průchodu signálu se syndromem získaným pomocí kontrolní matice.

Příklad 5.1.4 Odvoďte impulzní odezvu obvodu z obr.5.19.



Obr.5.19

Impulzní odezva je dána zpětnou D-transformací přenosové funkce, což můžeme vyjádřit výrazem

$$h_i = D^{-1}[T] = \left\{ \frac{\sum_{i=0}^{M} a_i . D^{-i}}{1 \oplus \sum_{i=1}^{L} b_i . D^{-i}} \right\} \qquad (5.27)$$

Pro přenosovou funkci můžeme z obrázku odvodit výraz

$$T = \frac{1 \oplus D^{-1} \oplus D^{-3}}{1 \oplus D^{-2} \oplus D^{-4}} = 1 \oplus D^{-1} \oplus D^{-2} \oplus D^{-5} \oplus D^{-6} \oplus D^{-7} \oplus D^{-8} \oplus D^{-9} .. \qquad (5.28)$$

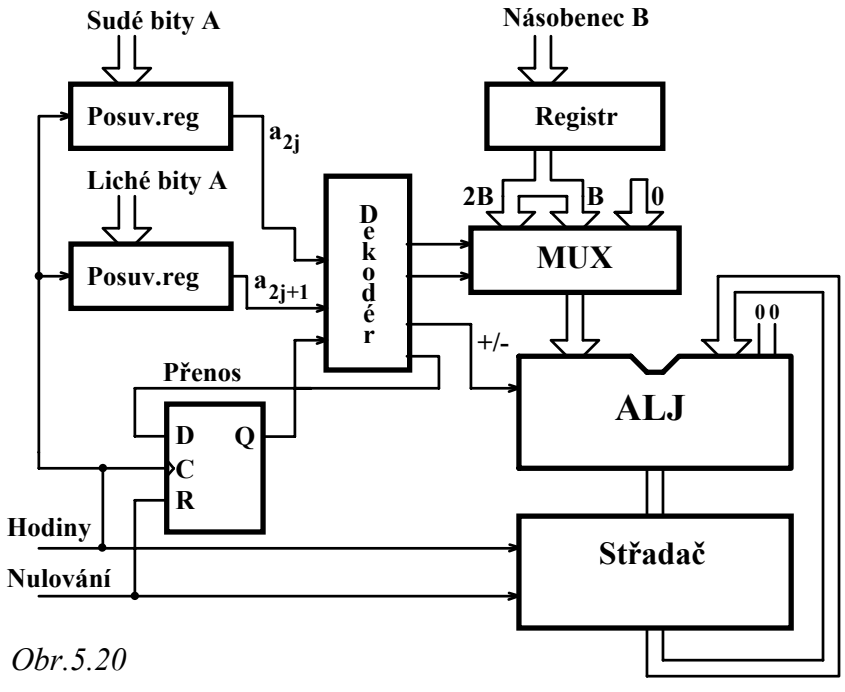
5.2. Sériové aritmetické operace

V části 3.3 bylo pojednáno o logických kombinačních obvodech, které realizují operace sčítání, odčítání, násobení a dělení. Při návrhu těchto obvodů se ukázalo, že pro vzrůstající počet zpracovávaných bitů však složitost obvodu rychle vzrůstá. V případech, kdy nejsou kladeny zvláštní nároky na rychlost zpracování, jsou sériové aritmetické operace jednou z možností, jak snížit složitost realizovaného obvodu. Při sériových operacích se úkony prováděné při paralelním zpracování v kaskádě obvodů, převedou na posloupnost po sobě následujících operací prováděných v jednom obvodu. Jeden nebo oba operandy jsou uloženy v posuvných registrech, z kterých jsou postupně vysouvány pomocí hodinového signálu. Sériové sčítání kladných dvojkových čísel lze realizovat úplnou binární sčítačkou doplněnou paměťovým členem D, který zpožďuje (v sobě uchovává) přenos do dalšího řádu. Při odečítání můžeme použít stejný obvod, pokud přivedeme menšitele ve dvojkovém doplňku, který nemusíme realizovat obvody EX-OR jako při paralelním zpracování. Sériový převod do dvojkového doplňku se provádí od nejméně platného bitu tak, že první jedničku převáděného čísla zapíšeme do výsledku a

$a_{2j+1}.2^{2j+1}$	$a_{2j}.2^{2j}$	$p_{2j}.2^{2j}$	Dílčí součin	$p_{2j+2}.2^{2j+2}$
0	0	0	0	0
0	0	1	B	0
0	1	0	2B	0
0	1	1	-B	1
1	0	0	B	0
1	0	1	2B	0
1	1	0	-B	1
1	1	1	0	1

Tabulka 5.4

zároveň do klopného obvodu, jehož výstup řídí obvod EX-OR zajišťující inverzi všech dalších převáděných bitů. Tato operace je shodná s inverzí všech bitů (jednot-kový doplněk) a přičtení jedničky. Odečítání lze realizovat i tak, že menšitele invertujeme (jednotkový doplněk) a počáteční přenos p_0 nastavíme na hodnotu log.1. Sériové násobení se řídí stejným algoritmem jako násobení paralelní násobičkou realizovanou pomocí logických členů AND a sčítačky s tím, že se dílčí součty ukládají do posuvného registru zajišťujícího řádový posun. V řadě současných jednočipových mikroprocesorů se setkáváme se sériovým násobením využívajícím tzv. **Bootova** algoritmu, při kterém oproti klasickému násobení násobence jedním bitem násobitele s následným sčítáním se provádí násobení více bity násobitele. Nejprve rozdělíme násobitele, u kterého například předpokládáme sudý počet cifer (číslic), na dvojice bitů takto



Obr.5.20

$$A = \sum_{j=0}^{n/2-1} (a_{2j+1}.2^{2j+1} + a_{2j}.2^{2j}) \qquad (5.29)$$

kde n je počet cifer násobitele A. Pro výsledný součin A a B můžeme psát

$$N = \sum_{j=0}^{n/2-1} (a_{2j+1}.2^{2j+1} + a_{2j}.2^{2j}).B \qquad (5.30)$$

Vztah (5.30) popisuje součet dílčích dvoubitových součinů, kterých je oproti klasickému algoritmu o polovinu méně (celá část n/2). Otázkou zůstává realizace dvoubitového součinu, jehož všechny případy jsou zobrazeny v tabulce 5.4. Problematické násobení 3 se vytváří generováním přenosu odpovídajícímu hodnotě 4B a odečteme hodnotu -B. Na obr.5.20 je blokově zobrazena násobička s Bootovým algoritmem využívající dvoubitového násobení. Kromě dvoubitového násobení se používají součiny tříbitové a čtyřbitové (např. násobička SAB-80166). U těchto součinů se samozřejmě komplikuje funkce dekodéru zajišťujícího připojení příslušného operandu k aritmetické jednotce ALJ.

5.3. Čítače

Čítače jsou logické sekvenční obvody, které realizují funkci čítání. Vnitřním stavem svých n paměťových členů vyjadřují počet impulzů přivedených na jejich synchronizační vstup. Impulzy se sčítají v soustavě se základem M, kde M se nazývá **modul čítače**. Obecně lze čítače dělit podle změn vnitřních proměnných na synchronní a asynchronní a podle způsobu vytváření funkcí pro buzení vstupů paměťových členů na paralelní a sériové. Další členění je možné podle:

1) směru čítání - vpřed, vzad a obousměrné (reverzibilní)
2) modulu - binární $M=2^k$, kde k je celé (obvykle 4)
   - dekadické $M=10^k$, kde k je celé (obvykle 1)
   - programovatelné s obecným základem
3) způsobu nulování a přednastavení (naplnění) - synchronní a asynchronní
4) typu výstupu - s dvoustavovým nebo třístavovým výstupem
5) aktivní hrany - náběžná hrana, sestupná hrana

K realizaci čítačů se v současné době používají skoro výhradně integrované čítače, které jsou shrnuty v tab.5.5. Pouze v případě zvláštního řízení vnějšími proměnnými je možné přistoupit k realizaci s obvody s vyšší hustotou integrace PLA,PROM a registry.

Asynchronní čítače čítající vpřed s aktivní sestupnou hranou						
Obvod	Obvod	Modul	Naplnění	Nulování	Vývodů	Výstup
74LS90	74LS290	M=10	asynch.-9	asynch.	14	2.stav.
74LS93	74LS293	M=16	-----	asynch.	14	2.stav.
74LS196		M=10	asynch.	asynch.	14	2.stav.
74LS197		M=16	asynch.	asynch.	14	2.stav.
74LS390	74LS490	2xM=10	asynch.-9	asynch.	16	2.stav.
74LS393		2xM=16	-----	asynch.	16	2.stav.

Tabulka 5.5a

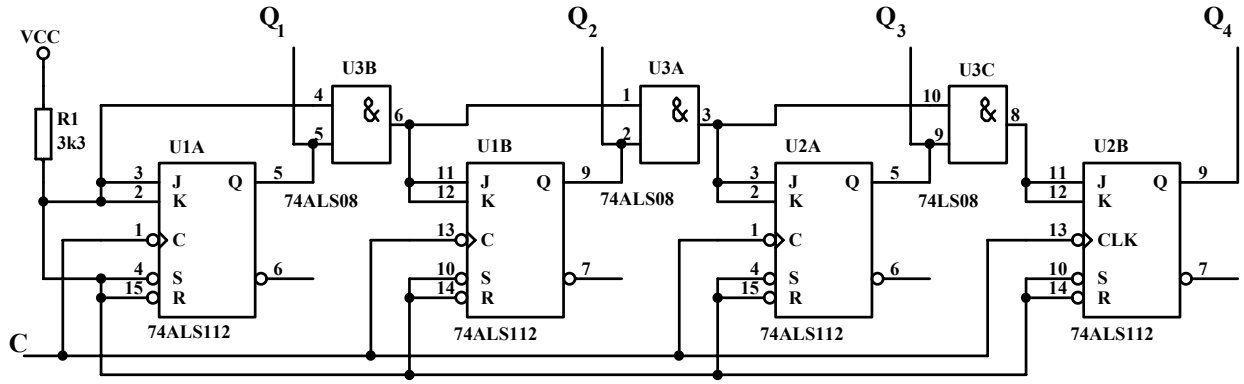
Synchronní čítače s aktivní náběžnou hranou						
Obvod	Modul	Naplnění	Nulování	Vývodů	Výstup	Směr
74AS160	M=10	synch.	asynch.	16	2.stav.	vpřed
74AS161	M=16	synch.	asynch.	16	2.stav.	vpřed
74AS162	M=10	synch.	synch.	16	2.stav.	vpřed
74AS163	M=16	synch.	synch.	16	2.stav.	vpřed
74AS168	M=10	synch.	-----	16	2.stav.	reverz
74AS169	M=16	synch.	-----	16	2.stav.	reverz
74ALS192	M=10	asynch.	asynch.	16	2.stav.	reverz
74ALS193	M=16	asynch.	asynch.	16	2.stav.	reverz
74ALS560	M=10	syn/asyn.	syn/asyn.	20	3.stav.	vpřed
74ALS561	M=16	syn/asyn.	syn/asyn.	20	3.stav.	vpřed
74ALS568	M=10	synch.	syn/asyn.	20	3.stav.	vpřed
74ALS569	M=16	synch.	syn/asyn.	20	3.stav.	vpřed
74LS668	M=10	synch.	-----	16	2.stav.	reverz
74LS669	M=16	synch.	-----	16	2.stav.	reverz
74LS690	M=10	synch.	asynch.	20	3.stav.	vpřed
74LS691	M=16	synch.	asynch.	20	3.stav.	vpřed
74LS693	M=16	synch.	synch.	20	3.stav.	vpřed
74LS696	M=10	synch.	asynch.	20	3.stav.	reverz
74LS697	M=16	synch.	asynch.	20	3.stav.	reverz
74LS699	M=16	synch.	synch.	20	3.stav.	reverz
74AS867	M=256	asynch.	asynch.	24	2.stav.	vpřed
74AS869	M=256	synch.	synch.	24	2.stav.	vpřed

Tabulka 5.5b

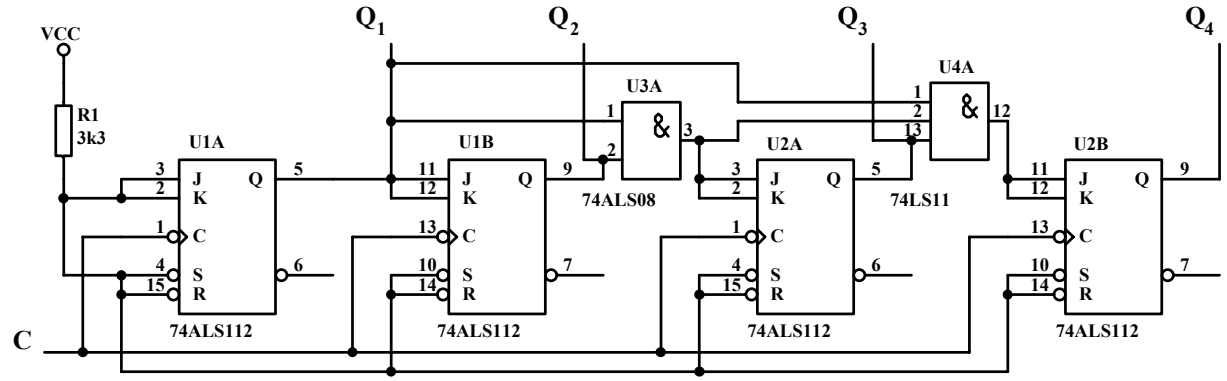
Synchronní čítače

Synchronní čítače je možné realizovat sériově tak, že funkce buzení vstupů paměťových členů tvořících čítač jsou realizovány vícestupňovou strukturou LKO. Vstupy paměťového členu jsou funkcí výstupní proměnné předcházejícího paměťového členu a funkcí vstupních proměnných, v které jsou skryty výstupní proměnné předcházejících paměťových členů. Jako

příklad takovéto struktury je na obr.5.21 pro dvojkový čítač. Tato struktura je obvodově obvykle jednodušší, ale omezuje dosažení maximálního kmitočtu v důsledku nutného čekání na průchod změny prvého stupně celou kaskádou součinových členů. Paralelní realizace stejného čítače je uvedena na obr.5.22, u níž jsou funkce buzení vstupů paměťových členů realizovány nejvýše dvoustupňovým logickým kombinačním obvodem ($J_1 = K_1 = 1$, $J_2 = K_2 = Q_1$, $J_3 = K_3 = Q_1.Q_2$, $J_4 = K_4 = Q_1.Q_2.Q_3$).



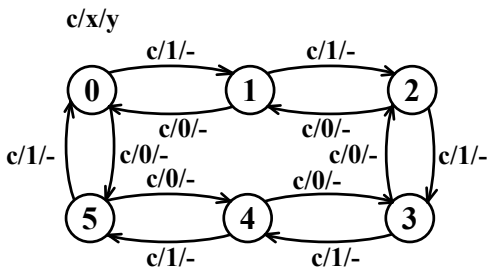
Obr.5.21



Obr.5.22

Návrh synchronního čítače se provádí stejně jako návrh synchronního sekvenčního obvodu s tím, že odpadá redukce vývojové tabulky a kódování vnitřních stavů, které je předepsáno zadáním. To vyplývá z toho, že stav čítače chceme rozumně dekódovat a proto jednotlivé stavy obvykle kódujeme binárním číslem.

Příklad 3.33 Navrhněte reverzibilní synchronní čítač mod 6 (M=6) jehož směr je řízen vstupní proměnnou x (x=1 vpřed, x=0 vzad).



Obr.5.23

Návrh čítače vychází z tabulky 5.6 charakterizující funkce přechodů nebo ze stavového diagramu obr.5.23. Čítač má šest stavů, které odlišíme pomocí tří vnitřních proměnných Q_3, Q_2, Q_1 $(6 < 2^3)$. Každému stavu přiřadíme

jednu kombinaci vnitřních proměnných a vytvoříme tabulku 5.6, z které snadno odvodíme stavovou tabulku čítače tab.5.24 . Pro funkce přechodů snadno odvodíme tyto rovnice přechodů

$$Q_1^{i+1} = \overline{Q}_1^i \qquad (5.31)$$

$$Q_2^{i+1} = \overline{x}^i . Q_1^i . Q_2^i + x^i . \overline{Q}_1^i . Q_2^i + \overline{x}^i . \overline{Q}_1^i . \overline{Q}_2^i . Q_3^i + x^i . Q_1^i . \overline{Q}_2^i . \overline{Q}_3^i$$

$$Q_3^{i+1} = \overline{x}^i . \overline{Q}_1^i . \overline{Q}_2^i . \overline{Q}_3^i + \overline{x}^i . Q_1^i . Q_2^i . \overline{Q}_3^i + x^i . \overline{Q}_1^i . \overline{Q}_2^i . Q_3^i + \overline{x}^i . Q_1^i . \overline{Q}_2^i . Q_3^i$$

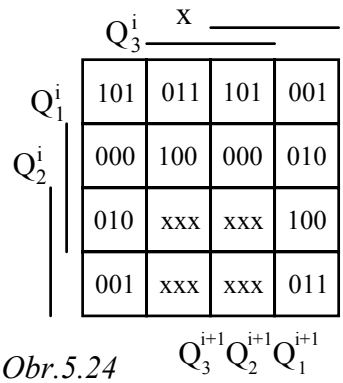
		pro x =0		pro x=1	
Stav	$Q_3^i\ Q_2^i\ Q_1^i$	Stav	$Q_3^{i+1}\ Q_2^{i+1}\ Q_1^{i+1}$	Stav	$Q_3^{i+1}\ Q_2^{i+1}\ Q_1^{i+1}$
0	0 0 0	5	1 0 1	1	0 0 1
1	0 0 1	0	0 0 0	2	0 1 0
2	0 1 0	1	0 0 1	3	0 1 1
3	0 1 1	2	0 1 0	4	1 0 0
4	1 0 0	3	0 1 1	5	1 0 1
5	1 0 1	4	1 0 0	0	0 0 0

Tabulka 5.6

Porovnáním funkcí přechodů s operátorem použitých paměťových členů dostaneme funkce buzení těchto členů. V případě paměťových členů D jsou budící funkce totožné s pravými stranami rovnic 5.31, pro paměťové členy JK snadno odvodíme tyto vztahy

$$J_1 = K_1 = 1 \qquad J_2 = \overline{x}^i . \overline{Q}_1^i . Q_3^i + x^i . Q_1^i . \overline{Q}_3^i \qquad K_2 = x^i \oplus Q_1^i \qquad (5.32)$$

$$J_3 = \overline{x}^i . \overline{Q}_1^i . \overline{Q}_2^i + x^i . Q_1^i . Q_2^i \qquad K_3 = Q_2^i + x^i . Q_1^i + \overline{x}^i . \overline{Q}_1^i$$

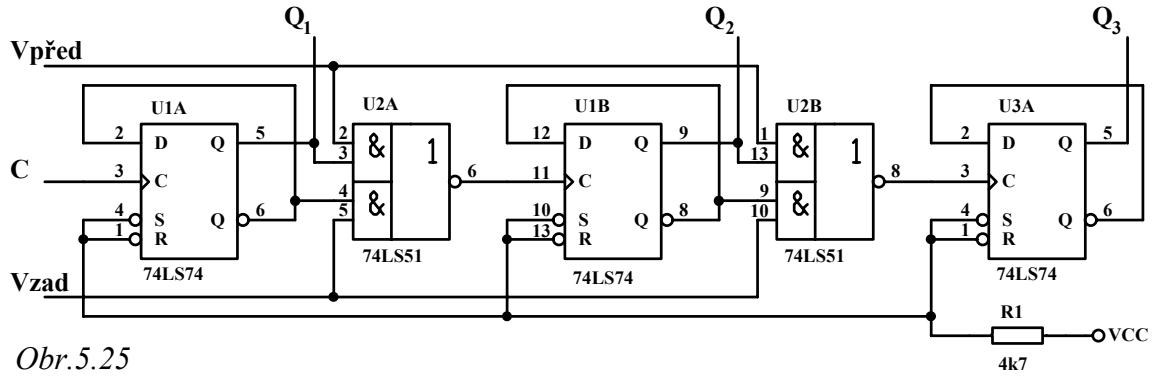


Obr.5.24

Dříve než přistoupíme k realizaci čítače (jakéhokoliv sekvenčního obvodu), musíme stanovit chování tohoto obvodu v nepoužitých (volných) stavech do kterých se obvod může dostat po připojení obvodu k napájecímu napětí. Pokud není obvod na počátku své činnosti vynulován nebo nastaven, musíme zjistit zda existuje pro všechny stavy vstupních proměnných přechod do použitých stavů stavového diagramu. Jinými slovy musíme zjistit, zda námi vytvořené funkce přechodů nezpůsobují uzavřenou smyčku jednoho nebo několika nepoužitých stavů pro nějaký vstupní stav, z které by se obvod nedostal do požadovaného stavového diagramu. V našem případě zjisíme, že funkce pro vnitřní proměnnou Q_3^{i+1} musela být upravena, aby se neuzavřel stav 6 pro x=1 sám na sebe.

Asynchronní čítače

Asynchronní čítače realizované s paměťovými členy jsou specifikované tím, že některé vstupní synchronizační proměnné C_j jsou odvozeny z výstupních proměnných klopných obvodů předcházejících stupňů. Složitost takového čítače pak obvykle vychází jednodušší než u čítače synchronního, protože se zjednoduší funkce buzení vstupů paměťových členů. Realizace asynchronního čítače z paměťových členů patří minulosti a proto se jí zde nebudeme věnovat. Případnému zájemci doporučujeme práce [2],[3]. Na obr.5.25 je zobrazen příklad asynchronního reverzibilního čítače mod 8 (M=8) realizovaného s paměťovými členy D. Výstupní proměnné Q_3, Q_2 a Q_1 se budou postupně měnit v závislosti na vzniku synchronizační podmínky pro příslušný paměťový člen viz.obr.4.28. V tomto případě by po synchronizačním signálu výstupy Q_3, Q_2 a Q_1 procházely dvěma přechodnými stavy. Na tuto okolnost musíme pamatovat



Obr.5.25

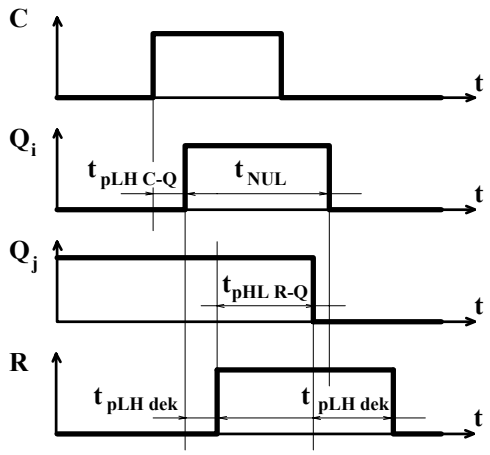
v případech, kdy výstupy čítače dekódujeme ke generování řídících signálů.

Čítače s velkým modulem

Jak již bylo v úvodu naznačeno, snažíme se co nejvíce využívat v aplikacích čítačů vyráběných v integrované podobě tab.5.5 a jen vyjímečně přistupujeme k jeho návrhu. Z tabulky však vyplývá, že integrované čítače jsou výhradně vyráběny s modulem 10, 16, 100 a 256. Zůstává otázkou jak zrealizujeme čítače s odlišným modulem M. K realizaci používáme čítač vytvořený z integrovaných obvodů (čítačů), který má modul větší než námi požadovaná hodnota. U takového čítače potom provedeme jedno z následujících nebo modifikovaných **zkrácení cyklu**.

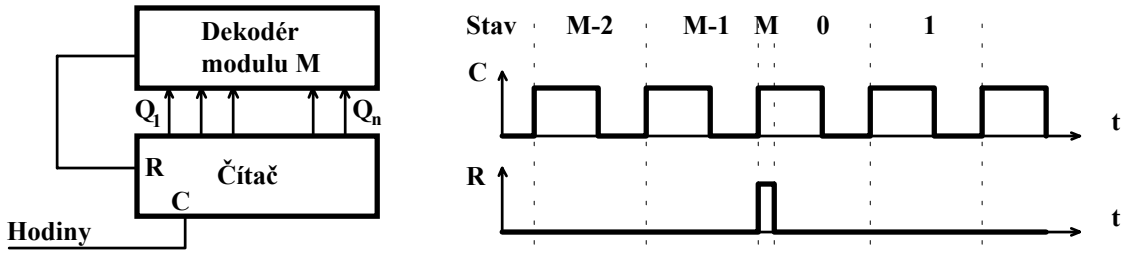
Zkrácení cyklu nulováním - asynchronní řešení

Na obr.5.26 je zobrazeno řešení, které využívá okamžiku, kdy čítač dosáhne požadovaného modulu M (čísla M) a je asynchronně vynulován výstupem logického kombinačního obvodu, který dekóduje (indikuje) číslo M. Na obr.5.27 je zobrazen časový diagram



Obr.5.27

asynchronní zpětné vazby, kde doba trvání přechodného děje je dána $t_{NUL} = t_{pLHdek} + \max(t_{pHL\,R\to Q})$ vztahem a doba trvání nulovacího impulzu $t_R = t_{pHLdek} + \min(t_{pHL\,R\to Q})$ přivedeného na vstup R. Pohledem do katalogu snadno zjistíme, že časy t_{NUL} a t_R jsou velmi krátké, řádu desítek ns. Při nízkém opakovacím kmitočtu je téměř nelze na osciloskopu sledovat, což zhoršuje údržbu a opravy. Mění se s teplotou, napětím, stárnutím a kapacitní zátěží. Špatně nastavená sonda osci-

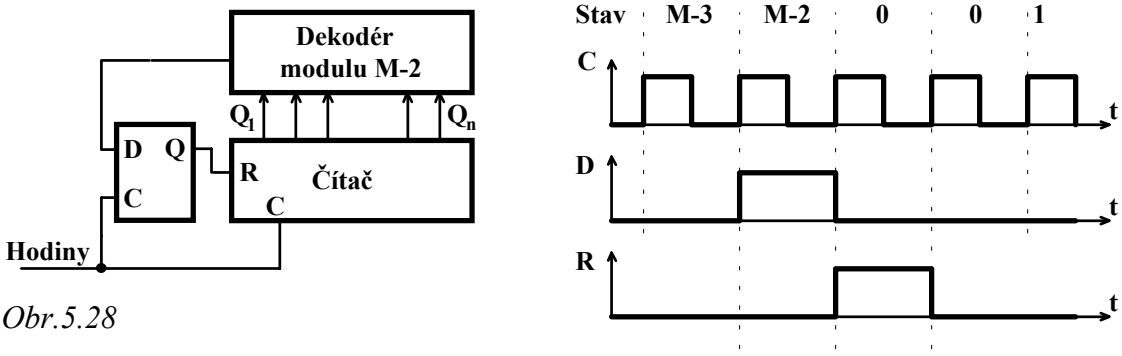


Obr.5.26

loskopu je může výrazně zkreslit. Pro správnou činnost asynchronního nulování čítače musí platit, aby $t_R > t_{Rmin}$, kde t_{Rmin} je minimální délka nulovacího impulzu stanovená výrobcem. Tato nerovnost musí být splněna i pro nejrychleji se nulující obvod. Problémem se tak může stát náhrada vadného obvodu obvodem novým, který bývá často rychlejší. Vztah $t_R > t_{Rmin}$ bývá pro různé obvody splněn jen tak tak, což často vede konstruktéry k prodloužení nulovacího impulzu pomocí obvodu RC, monostabilního obvodu atd.

Zkrácení cyklu nulováním - synchronní řešení

Při tomto řešení, které je zobrazeno na obr.5.28, zavedeme výstupy čítače do dekodéru požadovaného modulu zmenšeného o hodnotu 2. Výstup dekodéru je přiveden na vstup pamě-



Obr.5.28

ťového členu D, jehož výstup je propojen s nulovacím vstupem čítače. Jak vyplývá z časových průběhů je při dosažení hodnoty M-1 po celou periodu čítač nulován. Vzhledem k tomu, že se nachází v asynchronním režimu nereaguje na následující aktivní hranu hodinového signálu a setrvává ještě jednu periodu ve stavu 0. Uvedená dvě základní zapojení pro zkrácení cyklu lze modifikovat využitím nastavovacího vstupu a nenulové počáteční hodnoty, směrem čítání i způsobem vlastního nulování nebo přednastavení.

Mezní kmitočet čítačů závisí na typu logických členů použitých k realizaci čítačů, na způsobu jeho zapojení a případně i na jeho kódování. Jestliže označíme $t_{p\,C\to Q}$ dobu přenosu z hodinového vstupu na výstup Q při synchronním řízení klopného obvodu, t_{pd} dobu zpoždění signálu logickým členem a t_{setup} dobu předstihu ustálených hodnot na budících vstupech paměťových členů, potom pro jednotlivé způsoby zapojení čítačů můžeme psát

$$T_{\min} \geq t_{p\,C\to Q} + 2.t_{pd} + t_{setup} \qquad (5.33)$$

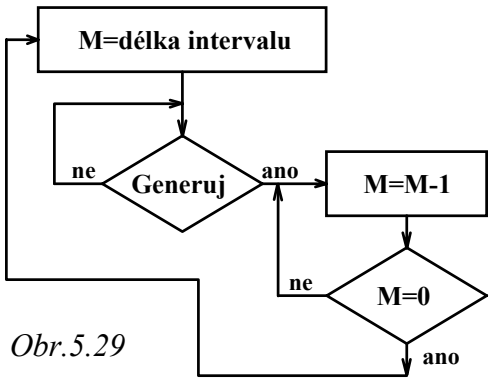
pro synchronní čítač s paralelními dvoustupňovými funkcemi buzení vstupů paměťových členů,

$$T_{\min} \geq t_{p\,C\to Q} + (n-1).t_{pd} + t_{setup}, \qquad (5.34)$$

pro synchronní čítač se sériovými funkcemi buzení vstupů paměťových členů a

$$T_{\min} \geq n.t_{p\,C\to Q} + 2.t_{pd} + t_{setup} \qquad (5.35)$$

pro asynchronní čítač s postupnou tvorbou synchronizačních signálů C_j u všech paměťových členů a s dvoustupňovými funkcemi buzení vstupů paměťových členů, kde n je počet paměťových členů. Jestliže využíváme stav čítače, bude nutné zvětšit tuto dobu o čas nutný k dekódování stavu čítače. Pokud využíváme asynchronní zkrácení cyklu, uplatňují se ještě zpoždění signálu v dekodéru modulu a zpoždění přenosu ze vstupu asynchronního řízení na výstup čítače viz. obr.5.27. Kromě popsaných časů, které ke správné činnosti čítače je nutno dodržet, je třeba pamatovat na zpoždění způsobená rozvodem signálů pomocí plošného spoje, zvláště při vysokých hodinových kmitočtech. Z tohoto důvodu by měl být hodinový signál od svého zdroje rozveden tak, aby nejprve dosáhl prvního a potom postupně následující čítače (paměťové členy).

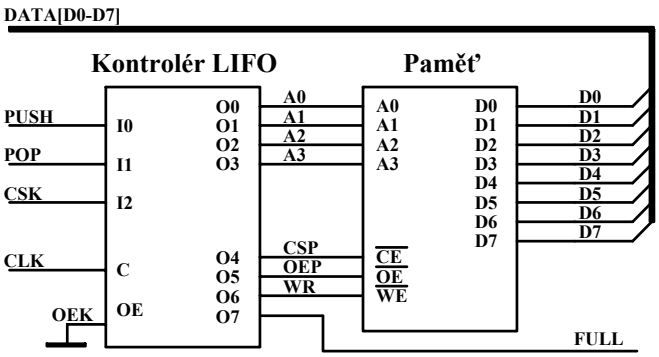


Obr.5.29

V praktických aplikacích se čítače nejčastěji používají k čítání, dělení kmitočtu, generování funkcí a jako základní součást řadičů. Z aplikací, kde využíváme funkci čítání, připomeňme měření neznámého kmitočtu po definovanou dobu a měření doby periody čítáním

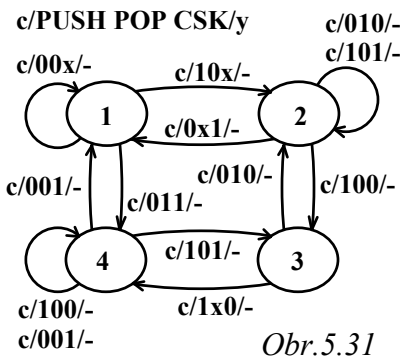
definovaného kmitočtu. Z dalších aplikací připomeňme využití čítače k adresování viz. příklad 5.9 nebo ke generování intervalu, jehož vývojový diagram je zobrazen na obr.5.29. Hlavní výhodou tohoto řešení, které se často používá v mikroprocesorové technice při programovém nebo obvodovém (časovačem) řešení, je přesnost generovaného intervalu určená stabilitou hodinového signálu. Ve funkci dělení kmitočtu se s čítači nejčastěji setkáváme ve fázových závěsech (kmitočtových syntetizátorech), kde kmitočet oscilátoru je dělen pevnou nebo programovatelnou hodnotou (přivedenou na vstupy přednastavení) a potom do fázového detektoru k porovnání s referenčním kmitočtem. Generátory číslicových funkcí mohou generovat obdélníkový průběh s různou třídou (změnou modulu čítače), trojúhelníkový a lichoběžníkový průběh (po dosažení maximální a minimální hodnoty čítače je obrácen směr čítání) a pilovitý průběh (po dosažení maximální hodnoty je čítač vynulován nebo nastaven). Ve spojení čítače s pamětí nebo ALJ (aritmeticko-logickou jednotkou) může být generován libovolný průběh. Často bývá stav čítače dekódován za účelem vytvoření řídicích signálů. Tato úloha však již spadá do problematiky použití čítače v řadičích, která bude probírána v následující kapitole.

Příklad 5.9 ***Navrhněte kontrolér paměti LIFO tvořené 16 bitovou statickou pamětí. Kontrolér má tři vstupy PUSH, POP a $\overline{CSK}$, které aktivují generování odpovídajících adres A0 až A3 a aktivačních signálů $\overline{CSP}$, $\overline{OEP}$ a $\overline{WR}$ pro příslušnou paměť. Při řešení předpokládejte, že signál $\overline{CSK}$ (aktivní v log.0) je vždy překrytý signálem PUSH nebo POP (aktivními v log.1). Kontrolér vybavte výstupem indikujícím naplnění celé paměti LIFO.***



Obr.5.30

Na obr.5.30 je zobrazeno navrhované blokové zapojení paměti LIFO (Last In First Out), která se využívá v CPU jako zásobníková paměť pro ukládání návratových adres při volání podprogramů nebo k rychlému ukládání mezivýsledků bez nutnosti specifikace adresy. Navrhovaný kontrolér se bude skládat z čítače, který bude zvětšovat o jedničku (inkrementovat) adresu před zápisem do paměti při operaci PUSH (ulož) a zmenšovat adresu (dekrementovat) po čtení z paměti při operaci POP (vyjmi). Na obr.5.32 jsou zobrazeny předpokládané časové průběhy řídících signálů. Kromě čítače bude nezbytné kontrolér vybavit řídícím

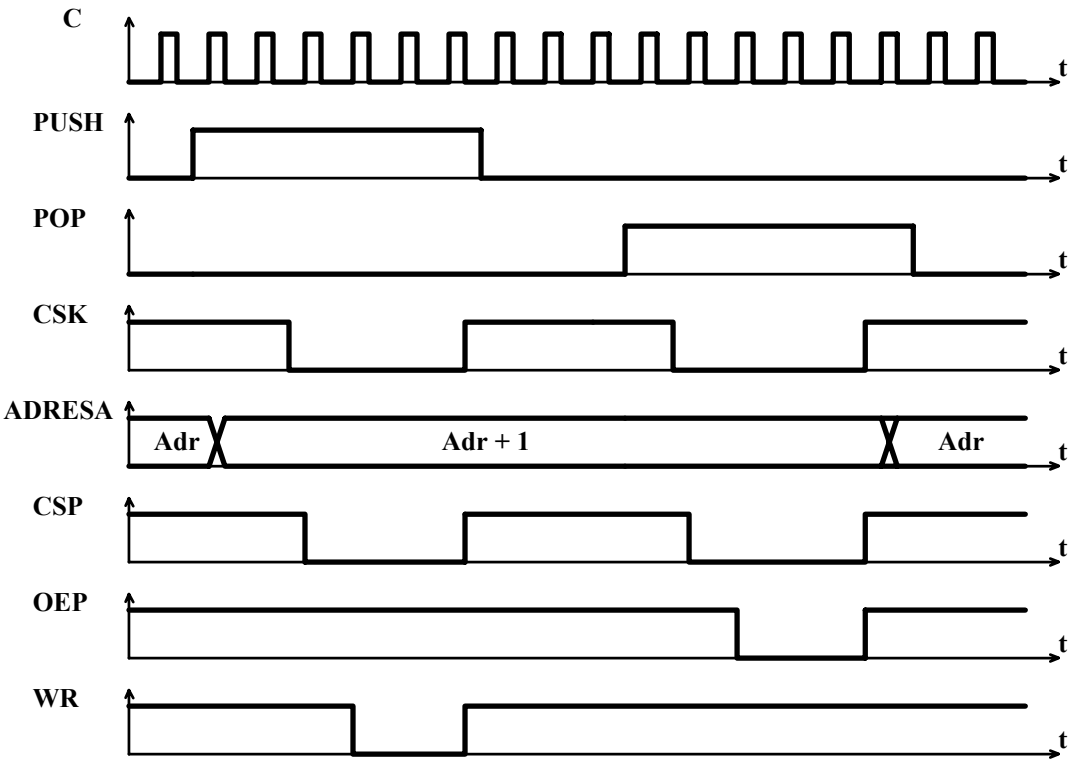


Obr.5.31

obvodem, který by rozeznal zda má k operaci s adresou dojít před generováním nebo po generování signálů $\overline{CSP}$, $\overline{OEP}$ a $\overline{WR}$ pro ovládání paměti. Vývojový diagram takového obvodu je zobrazen na obr.5.31 a k němu odpovídající vývojová tabulka 5.7. Po zakódování vnitřních stavů (1≡00, 2≡01, 3≡11 a 4≡10) snadno určíme tyto rovnice přechodů

$$Q_1^{i+1} = PUSH.\overline{Q_2} + (Q_1 + Q_2).POP.\overline{CSK} \qquad (5.36)$$

$$Q_2^{i+1} = POP.\overline{Q_1} + (Q_1 + Q_2).PUSH.\overline{CSK} \qquad (5.37)$$



Obr.5.32

	POP, PUSH, CSK							
Stav	000	001	011	010	110	111	101	100
1	---	1	2	2	---	---	4	4
2	---	1	2	3	---	---	1	2
3	---	---	---	4	---	---	---	2
4	---	1	1	4	---	---	4	3

Tabulka 5.7

Na základě obr. 5.32 a stavového diagramu potom odvodíme tyto rovnice pro řídící signály $\overline{CSP}$, $\overline{OEP}$ a $\overline{WR}$. Součiny vnitřních proměnných (Q_2,Q_1) určují stav, proměnné PUSH a POP určují typ operace a přičtení hodnoty CSK zajišťuje asynchronní ukončení signálů na vzestupnou hranu tohoto signálu.

$$CSP = \overline{(Q_1.\overline{Q}_2.POP + Q_1.Q_2 + Q_2.\overline{Q}_1.PUSH)} + CSK \qquad (5.38)$$

$$OEP = \overline{(\overline{Q}_1.\overline{Q}_2)} + \overline{POP} + CSK \qquad (5.39)$$

$$WR = \overline{(Q_2.\overline{Q}_1)} + \overline{PUSH} + CSK \qquad (5.40)$$

$$FULL = A0.A1.A2.A3 \qquad (5.41)$$

Navržený řídící obvod musíme nyní doplnit čtyřbitovým reverzibilním čítačem, který zvětší svoji hodnotu o jedničku, jestliže se řídící obvod nachází v počátečním stavu 1 (Q_2Q_1=00) a je aktivní hodnota signálu PUSH a zmenší svoji hodnotu o jedničku, jestliže se obvod nachází ve stavu 2 (Q_2Q_1=01) při neaktivním signálu CSK=0 a POP=1 nebo PUSH=0 (případ dlouhého přesahu PUSH přes signál CSK nebo impulzu PUSH bez aktivního signálu CSK). Ve výpisu vstupního souboru LIFO.PDS pro program PALASM2 jsou tyto podmínky vyjádřeny signály (řetězci) INC a DEC učenými příkazy STRING. Rovnice přechodů pro jednotlivé adresové vodiče A0, A1, A2 a A3 jsou vyjádřeny paralelními funkcemi buzení vstupů paměťových členů, které lze v případě větší paměti snadno rozšířit pro další adresové proměnné s tím, že bude nezbytné použít jiný typ programovatelného obvodu s větším počtem výstupních vodičů. Možné by bylo i použití dalšího obvodu, který by realizoval zbývající část adresového čítače a byla do něj přivedena informace o směru čítání a podmínka (přenos) určující změnu adresy. Pro počáteční nastavení je obvod vybaven nulovacím vstupem RST. Na obr.5.33 je potom zobrazeno simulované chování navrženého kontroléru.

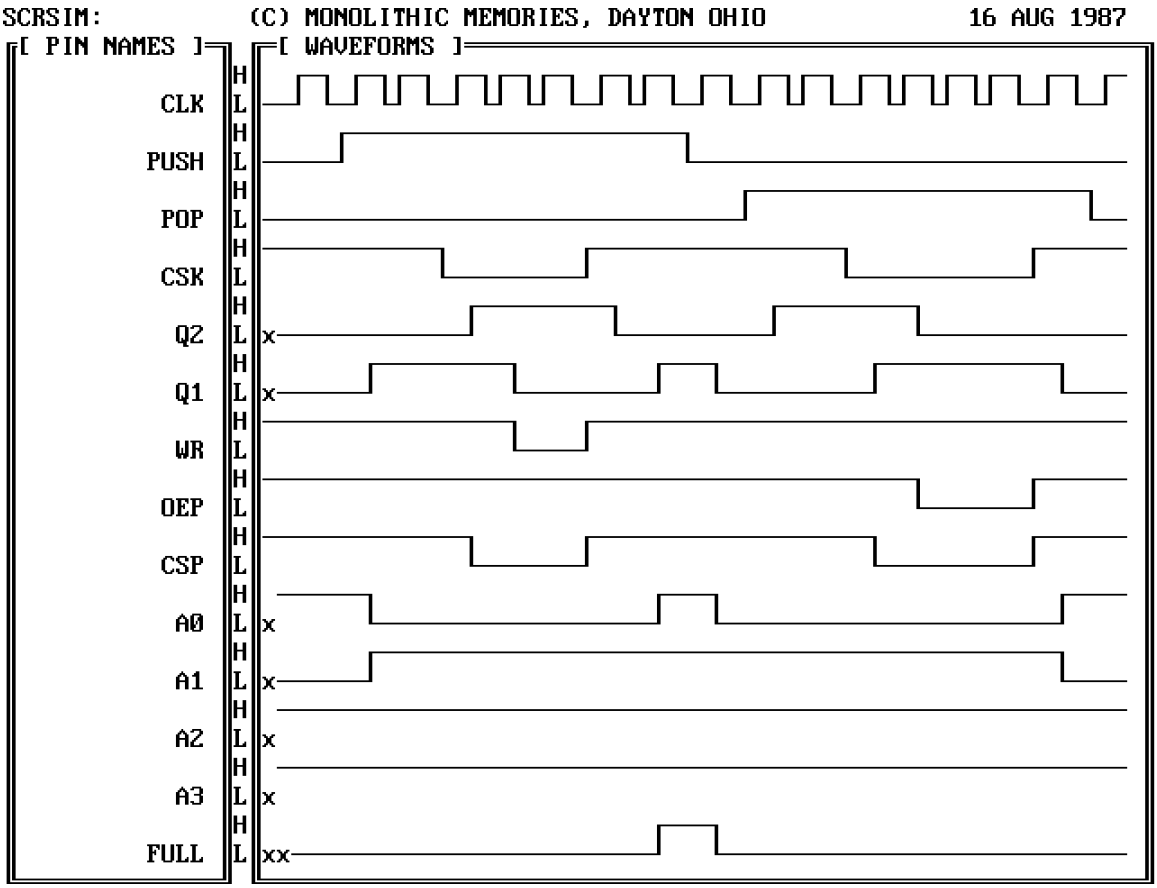
Program **LIFO.PDS**

```
TITLE       Kontrolér LIFO
PATTERN     01.
REVISION    01_def
AUTHOR      Skalický
COMPANY     ČVUT FEL
DATE        2.3.1994

CHIP LIFO PAL22V10
;PIN  1    2    3    4    5    6    7    8    9    10   11   12
      CLK  PUSH POP  CSK  RST  NC   NC   NC   NC   NC   NC   GND
;PIN  13   14   15   16   17   18   19   20   21   22   23   24
      NC   WR   OEP  A0   A1   A2   A3   Q1   Q2   CSP  FULL VCC
      GLOBAL
;Q1 a Q2 určují vnitřní stav obvodu      CSP, WR, OEP jsou ridici vystupy obvodu
;A0, A1, A2 a A3 jsou adresove vodice    FULL - Indikuje log.1 plny zasobnik
```

STRING INC 'PUSH*/Q1*/Q2'
STRING DEC 'CSK*/Q2*Q1*(POP+/PUSH)'

EQUATIONS
GLOBAL.RSTF=RST
A0:=A0:+:(INC+DEC)
A1:=A1:+:A0*INC+/A0*DEC
A2:=A2:+:A0*A1*INC+/A0*/A1*DEC
A3:=A3:+:A0*A1*A2*INC+/A0*/A1*/A2*DEC
Q1:=PUSH*/Q2+POP*/CSK*(Q2+Q1)
Q2:=POP*/Q1+PUSH*/CSK*(Q2+Q1)
WR=/(Q2*/Q1)+/PUSH+CSK
OEP=/(Q1*/Q2)+/POP+CSK
CSP=/(Q1*/Q2*POP+Q1*Q2+Q2*/Q1*PUSH)+CSK
FULL=A0*A1*A2*A3

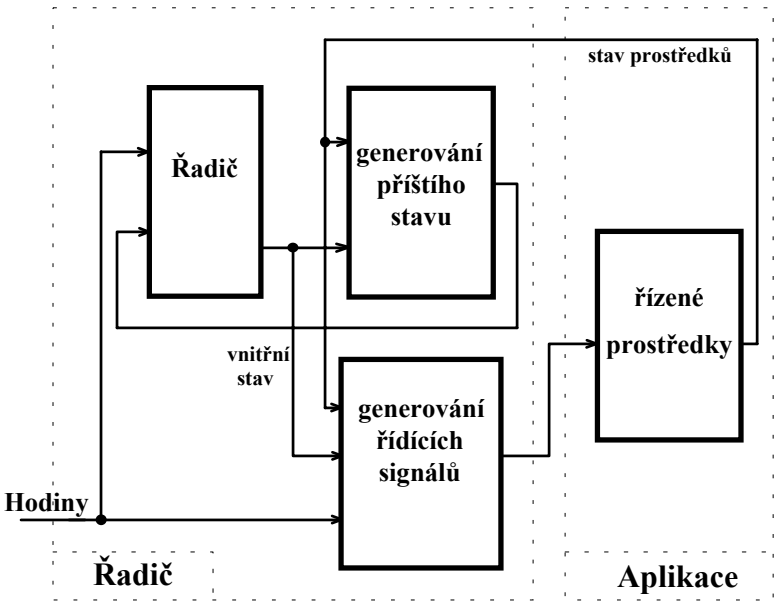
SIMULATION
TRACE_ON CLK PUSH POP CSK Q2 Q1 WR OEP CSP A0 A1 A2 A3 FULL
SETF /CLK /PUSH /POP CSK
PRLDF /Q1 /Q2 A0 /A1 A2 A3
CLOCKF CLK SETF PUSH CLOCKF CLK CLOCKF CLK SETF /CSK
CLOCKF CLK CLOCKF CLK CLOCKF CLK SETF CSK CLOCKF CLK
CLOCKF CLK SETF /PUSH CLOCKF CLK SETF POP CLOCKF CLK
CLOCKF CLK SETF /CSK CLOCKF CLK CLOCKF CLK CLOCKF CLK
CLOCKF CLK SETF CSK CLOCKF CLK SETF /POP CLOCKF CLK
TRACE_OFF



Obr.5.33

5.4. Řadiče

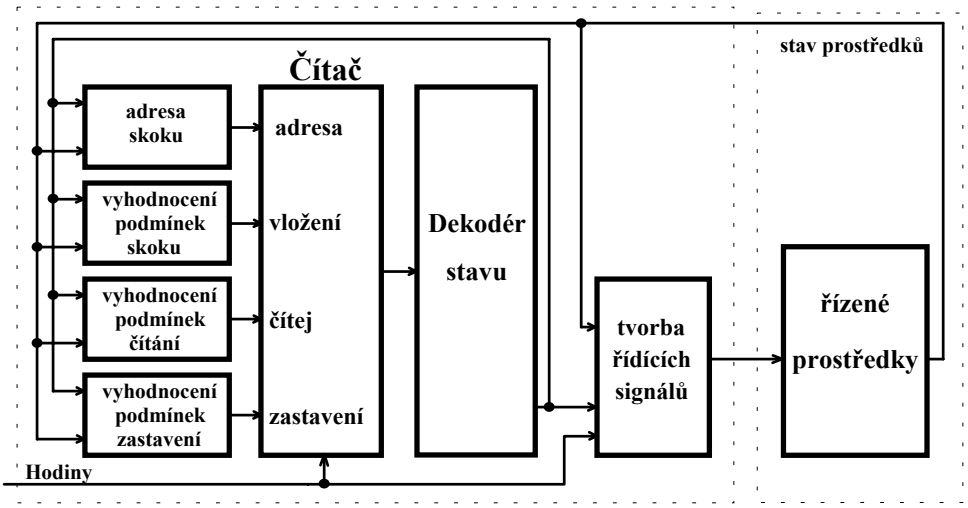
Řadičem označujeme tu součást složitého sekvenčního obvodu, která generuje řídicí signály zajišťující správnou činnost všech ostatních částí obvodu. Řadič je tedy logický sekvenční obvod, který:řídí činnost řízených prostředků i svoji vlastní (své vlastní přechody mezi stavy). Automaticky přechází do následujících stavů, jestliže neobdrží od řízených prostředků žádost o čekání na událost (vstupní podmínku) nebo žádost o provedení skoku. Na obr. 5.34 je zobrazena základní struktura řadiče, jehož činnost i činnost řízených prostředků je synchronizována



Obr.5.34

hodinovým kmitočtem. Řadiče můžeme podle vnitřní struktury rozdělit na obvodové a mikroprogramované.

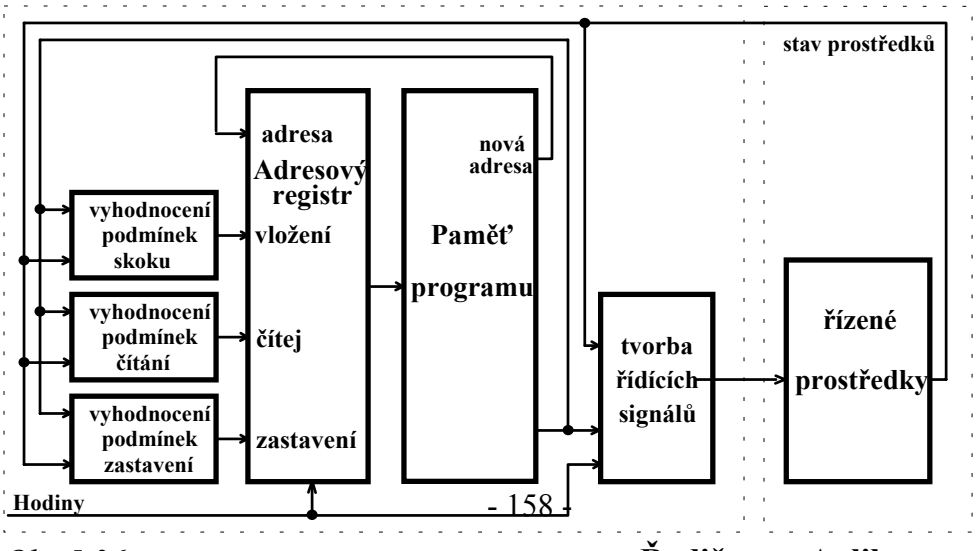
Obvodový řadič obr.5.35 je tvořen řízeným čítačem, který inkrementuje svůj obsah (přechází ze stavu S_i do S_{i+1}), umožňuje zápis nového stavu S_k k realizaci skoku a může setrvávat v definovaném stavu do splnění určité podmínky. Řízený čítač je doplněn dekodérem stavu, který generuje jednotlivé řídicí signály tím, že převádí stav řadiče (čítače) na kód 1 z N.



Obr.3.33

Obdobnou funkci jako čítač s dekodérem 1 z N realizuje kruhový čítač tvořený posuvným registrem v němž cykluje jediná jednička určující stav řadiče. Hlavní nevýhodou obvodového řadiče je to, že program jeho činnosti je pevný a tudíž jakákoliv jeho změna vede na změnu obvodového zapojení.

Mikroprogramovaný řadič je realizován adresovým registrem (čítačem), který generuje adresy pro vybavení mikroinstrukcí uložených v paměti mikroprogramu. Mikroinstrukce obsahuje řídicí signály pro řízené prostředky, adresu následující mikroinstrukce a případně i další signály. Skupinové zapojení mikroprogramovatelného řadiče je zobrazeno na obr.5.36. Hlavní výhoda tohoto řešení spočívá v možnosti i velkých změn v programu činnosti řadiče bez zásahu do jeho obvodové struktury (pokud bude vyhovovat počet vstupních a výstupních



Obr.5.36

vodičů).

Podle způsobu ovládání řízených prostředků můžeme dělit řadiče na řadiče s direktivním ovládáním a na řadiče se zpětnou vazbou. Řadič s direktivním ovládáním řízených prostředků generuje po inicializaci determinovanou posloupnost řídicích signálů a nedostává žádnou zpětnou informaci o stavu řízených prostředků. Doba reakce na řídicí signály musí být kratší než je perioda řídicích signálů. Tento typ řadiče se používá pro řízení jednoduchých obvodů jako jsou sériové aritmetické obvody, jednoduché číslicové filtry atd. Řadič se zpětnou vazbou získává zpětnou informaci o stavu řízených prostředků (tzv. stavové bity) a může podle ní upravit svoji činnost tzn. může čekat na splnění podmínky nebo provést skok na jinou řídící činnost. Podle složitosti řadiče používáme k jeho popisu stavovou tabulku (vhodnou pro jednoduché řadiče) přes popis vývojovým diagramem až po speciální jazyky přizpůsobené popisu chování složitých sekvenčních obvodů část 5.4.1.

Návrh obvodového řadiče

Při návrhu obvodového řadiče vycházíme z obvodové struktury řízené aplikace (řízených prostředků) a u synchronních systémů z vlastností použitého generátoru hodinových impulzů. Z vlastností kladených na navrhovaný řadič určíme základní průběhy řídicích signálů a zvolíme vhodnou strukturu řadiče umožňující jejich generování. Při rozboru řadiče vymezíme důležité funkční prvky a popíšeme jejich činnost časovým nebo vývojovým diagramem. Vývojový diagram musí postihovat všechny alternativy činnosti systému a je třeba jej důkladně analyzovat, protože je základem pro následující etapy návrhu. U synchronních systémů je nutné dále specifikovat vlastnosti generátoru hodinových signálů. Další návrh lze realizovat stejně jako návrh jednoduchého sekvenčního obvodu Mooreova typu (vývojová tabulka, kódování vnitřních stavů, stavová tabulka a realizace funkcí přechodů a výstupů). Postup návrhu je metodicky správný, ale obvykle nepřehledný a poskytuje mnohoznačné řešení podmíněné zkušeností návrháře. V případech návrhu řadiče i řízených prostředků, může být efektivnějším postupem apriorní vytvoření struktury řízených prostředků, generátoru hodin i typu řadiče (s čítačem nebo s registrem). Logickým návrhem určíme bloky vytvářející akce na řízených prostředcích i na řadiči samotném, které popíšeme logickými rovnicemi na základě vývojového diagramu. Při zápisu logických rovnic vycházíme z počátečního stavu řadiče, do kterého se dostane řadič po inicializaci (zapnutí). Vyjádříme logickou rovnici první akce v řízených prostředcích, která bude obsahovat okamžitý stav řadiče, logický výraz popisující akce, hodinový signál řídící akci a případně i podmínky popisující stav řízených prostředků. Stejně popíšeme akci v řadiči vedoucí k zápisu nového stavu čítače řadiče. Jestliže k takové situaci nedochází předpokládáme, že řadič přechází do následujícího stavu tak, aby docházelo k přirozenému zřetězení stavů řadiče. Takto postupujeme do vyčerpání všech stavů systému definovaných vývojovým diagramem.

Příklad 5.10 Navrhněte obvodový řadič pro sériovou sčítačku dvou desetibitových dvojkových čísel X a Y.

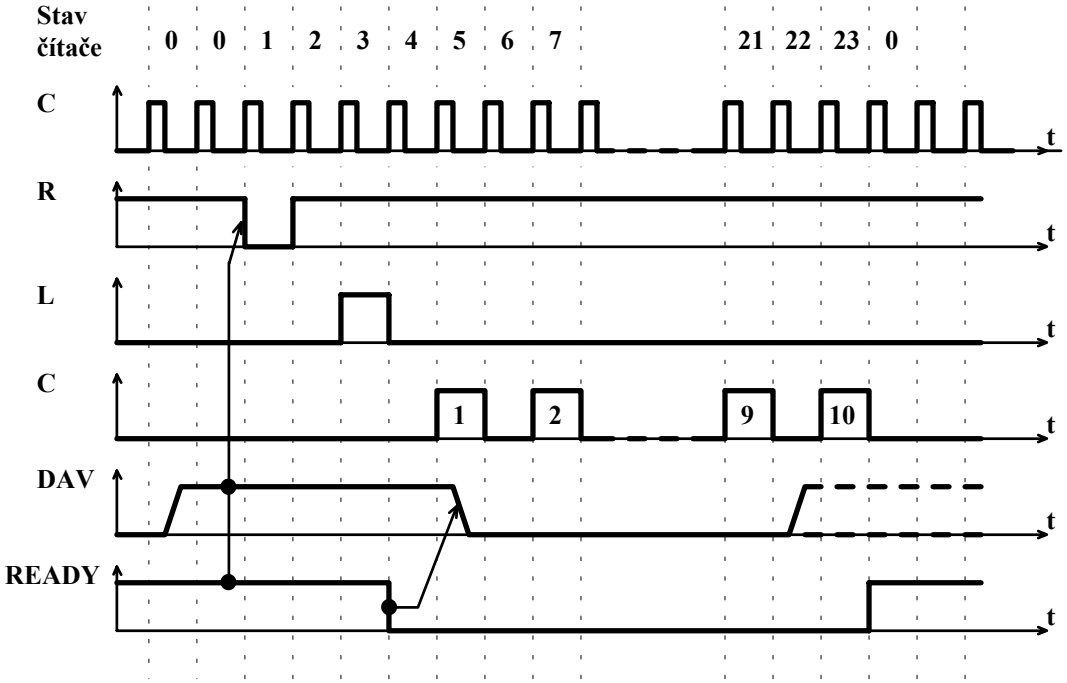
Při sériovém sčítání jsou sčítaná čísla uložena v posuvných registrech, z kterých jsou postupně vysouvány jednotlivé bity (od nejméně významného po nejvýznamnější) do úplné binární sčítačky, v které se provede jejich součet (součet v jednom řádu). Navrhovaný obvod, který bude řídit sčítání čísel, musí na počátku své činnosti nejprve vynulovat paměť (klopný obvod) přenosu p_i (nastavit nulový počáteční přenos) a provést zápis nových čísel X a Y do posuvných registrů, jestliže předcházející součet byl již dokončen a předán k dalšímu zpracování. Po těchto dvou operacích již může začít generovat deset hodinových impulzů, které vysunou sčítaná čísla a výsledek uloží opět do desetibitového posuvného registru. Navrhovaný řadič musí generovat tři řídicí signály R-nulování, L-načtení a C-hodinové impulzy a signál READY, kterým bude indikovat ukončený součet čísel. Signál READY můžeme zároveň využít k přenosu získaného součtu. O platnosti sčítaných čísel X a Y a o žádosti k jejich sečtení bude řadič informován signálem DAV. Na obr.5.37 je zobrazen časový průběh řídicích signálů zajišťujících správnou činnost obvodu, kterou musí generovat řídicí jednotka (obvodový řadič). Z časového diagramu vyplývá, že po zápisu zpracovávaných hodnot je zrušen signál READY ($1 \rightarrow 0$), z kterého periferie zjistí, že hodnoty byly převzaty. Po dokončení součtu generuje řídicí jednotka signál READY ($0 \rightarrow 1$), kterým oznamuje, že sčítačka je připravena zpracovávat další hodnoty. Jsou-li ve stejné době připraveny další hodnoty (READY = DAV = 1), pak řídicí jednotka pokračuje ve své činnosti. V opačném případě (DAV = 0) setrvává v počátečním stavu.

	Mod,DAV			
z^i	0 0	0 1	1 1	1 0
A	A	A	B	B
B	B	A	---	---

Tabulka 5.8a

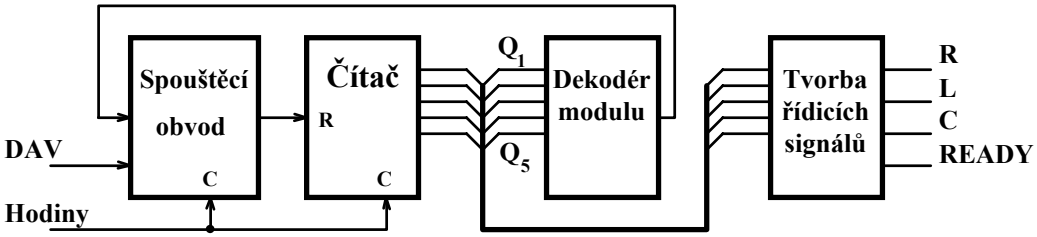
	Mod,DAV			
$z_2^i z_1^i$	0 0	0 1	1 1	1 0
0	0	0	1	1
1	1	0	---	---

Tabulka 5.8b



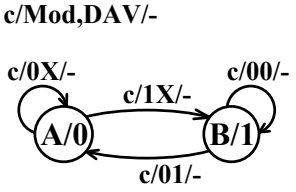
Obr.5.37

Řídicí signály (R,S,C a READY) budeme generovat dekódovacími obvody stavu



Obr.5.38

řízeného čítače obr.5.38. Oproti obecnému zapojení však použijeme jenom obvod pro vyhodnocení čítání a skoku. Protože nepředpokládáme zpětnou informaci ze sčítačky, bude se jednat o řadič s direktivním řízením. Řízený čítač, který musí mít modul 24 (obr.5.37), bude po jeho dosažení synchronně nastaven do počátečního stavu. Protože spouštěcí obvod bude synchronizován, musí dekodér pro zkrácení cyklu indikovat hodnotu (modul - 1, tzn. $23_{10} = 10111_2$). Na obr.5.39 je vývojový diagram spouštěcího a nulovacího obvodu, kde $\mathrm{Mod} = Q_5.\overline{Q}_4.Q_3.Q_2.Q_1$ je výstup dekodéru pro zkrácení cyklu a stav B (výstup y=1) odpovídá případu, kdy čítač je nulován. V tab.5.8a je zobrazena vývojová tabulka spouštěcího obvodu a v tab.5.8b je z ní odvozená stavová tabulka, vzniklá zakódováním stavů A=0, B=1. Ze stavové tabulky odvodíme tuto rovnici přechodů



Obr.5.39

$$z^{i+1} = Mod + \overline{DAV}.z^i \qquad (5.42)$$

Zbývá nám navrhnout dekodér; který ze stavů čítače vytváří řídicí signály. Signály R a L mají aktivní úroveň jen při jednom stavu čítače a proto budou dány těmito funkcemi

$$R = \overline{Q}_1 + Q_2 + Q_3 + Q_4 + Q_5 \qquad L = Q_1.Q_2.\overline{Q}_3.\overline{Q}_4.\overline{Q}_5 \qquad (5.43)$$

Pro signál READY a C odvodíme za pomoci Karnaughových map tyto vztahy

$$READY = \overline{Q}_3.\overline{Q}_4.\overline{Q}_5 \qquad C = Q_1.Q_3 + Q_1.Q_4 + Q_1.Q_5 \qquad (5.44)$$

Častou chybou je, že se používají přímo výstupy dekodéru, připojeného na synchronní čítač, a existují malé nepředvídatelné rozdíly, které mohou být rozdílnými cestami dekodérem znásobeny, a mohou způsobit vznik parazitních impulzů. Jejich odstranění je možné pomocí registru, který je aktivován v dostatečném odstupu od hodinových impulzů, kdy je stav čítače stabilní. Obvodový řadič zrealizujeme jedním obvodem PAL22V10AC, do kterého je naprogramován soubor JEDEC vytvořený návrhovým systémem ze souboru ŘADIČ.PDS. Maximální hodinový kmitočet je dán tímto výrazem

$$f_{\max} = 1/\left(t_{su} + t_{pd(C\to Q)}\right) = 1/(20+15).10^{-9} = 28{,}5 \quad [MHz] \qquad (5.45)$$

kde t_{su} je doba předstihu ze vstupu nebo zpětné vazby a $t_{pd(C\to Q)}$ je zpoždění z hodinového vstupu na výstup paměťového členy v programovatelném poli PAL22V10. Odtud bude nulovací a nastavovací impulz trvat minimálně 35[ns] a perioda generovaných hodinových impulzů C bude 70[ns]. Tyto hodnoty by nyní měly být porovnány s časovými parametry sčítačky, aby byla zajištěna její správná činnost při direktivním řízení řadičem. K návrhu řadiče lze přistoupit i z jiných pohledů, jako je hradlování hodinového kmitočtu nebo automodifikace registru (realizace logického sekvenčního obvodu pomocí registru a paměti ROM). Tato řešení lze považovat za překonaná a případnému zájemci lze doporučit [3].

Program ŘADIČ.PDS

```
TITLE       Obvodový řadič
PATTERN     01.
REVISION    01_def
AUTHOR      Skalický
COMPANY     ČVUT FEL
DATE        11.3.1994

CHIP PREPNI PAL22V10

;PIN  1   2   3   4   5   6   7   8   9   10   11   12
      CLK DAV RST NC  NC  NC  NC  NC  NC  NC   NC   GND
;PIN  13  14  15  16  17  18  19  20  21  22   23   24
      NC  Q1  Q2  Q3  Q4  Q5  R   L   C   Ready CDAV VCC
      GLOBAL
```

```
;Q5,Q4,Q3,Q2 a Q1 jsou vnitřní proměnné čítače,  R-nulovací, L-načítací a C-hodinový řídicí
;signál, CDAV synchronizovaný signál DAV
STRING NULUJ 'Q5*/Q4*Q3*Q2*Q1+/Q5*/Q4*/Q3*/Q2*/Q1*/CDAV'
;Proměnná NULUJ odpovídá rovnici pro spouštěcí obvod tj. Mod+/DAV*z^i, stav z^i=0 byl
ztotožněn s nulovým stavem čítače.

EQUATIONS
GLOBAL.RSTF=RST
Q1:=/Q1*/(NULUJ)
Q2:=(Q2:+:Q1)*/(NULUJ)
Q3:=(Q3:+:Q1*Q2)*/(NULUJ)
Q4:=(Q4:+:Q1*Q2*Q3)*/(NULUJ)
Q5:=(Q5:+:Q1*Q2*Q3*Q4)*/(NULUJ)

R:=/Q1+Q2+Q3+Q4+Q5
L:=Q1*Q2*/Q3*/Q4*/Q5
C:=Q1*Q3+Q1*Q4+Q1*Q5
READY:=/Q3*/Q4*/Q5
CDAV:=DAV           ;Synchronizace signálu DAV s hodinovým kmitočtem

SIMULATION
TRACE_ON    CLK  Q1 Q2 Q3 Q4 Q5 RST R L C DAV CDAV READY
 SETF    /CLK  RST DAV
SETF /RST
CLOCKF  CLK         CLOCKF  CLK         SETF /DAV
FOR I:=1 TO 30 DO
BEGIN
CLOCKF       CLK
END
SETF DAV            CLOCKF  CLK         CLOCKF  CLK
TRACE_OFF
```

5.4.1. Programovatelné řadiče

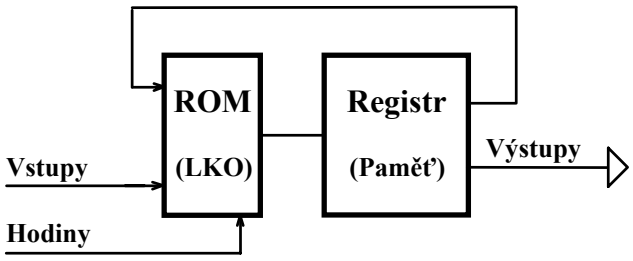
Jak bylo v předcházejících kapitolách ukázáno, návrh synchronních sekvenčních obvodů se doposud prováděl jednou ze dvou obecných metod:

- tradiční kombinační logikou s klopnými obvody
- mikroprogramováním

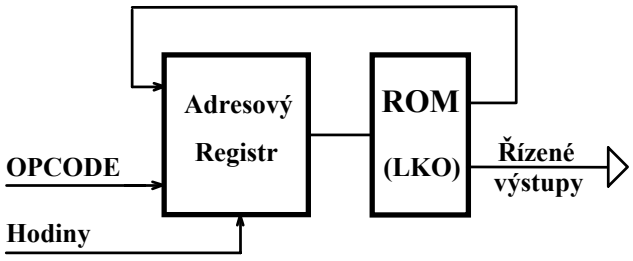
Tradiční metody se používají pro sekvenční obvody s relativně malým počtem stavů jako jsou jednoduché řídící obvody, řadiče pamětí, rozhodčí přístupu k pamětem, apod. Při návrhu se využívají stavové diagramy determinující chování obvodu s následným odvozením rovnic přechodů pro jednotlivé paměťové členy. Navržený sekvenční obvod je velmi rychlý a může být optimalizován na danou dílčí úlohu. Řešení obvodu nebývá obtížné a nyní je možné jej realizovat jedním obvodem PAL, GAL, EPLD, apod., ale pro rozsáhlé sekvenční obvody není

příliš výhodné. Oproti tomu mikroprogramování je používáno pro aplikace s velkým počtem stavů jako jsou CPU. Řídící sekvence odpovídající cyklu sekvenčního obvodu jsou uloženy v paměťovém prostoru a jsou čteny a vykonávány sekvenčně. Oblast mezi těmito hranicemi je ovládána míchanou technikou založenou na návrhu "ad hoc" využívající čítačů nebo posuvných registrů až po řadiče založené na pamětech PROM [1]. Nový trend v realizaci sekvenčních obvodů přináší obvody FPC (Field Programmable Controller) firmy AMD, které dovolují cenově efektivně využívat mikroprogramovatelnou techniku i pro obvody se středním počtem stavů a rychlostí srovnatelnou s tradičními sekvenčními obvody. Obvody Am29CPL100 vytváří skupinu jednočipových kontrolérů, které mají na jednom čipu sekvenční adresovou logiku, paměť programu i výkonný instrukční soubor podporující instrukce skoků, podmíněného větvení a volání podprogramů. Skrytý sériový registr (SSR) umožňuje návrháři určovat systémové problémy až na úrovni integrovaných obvodů.

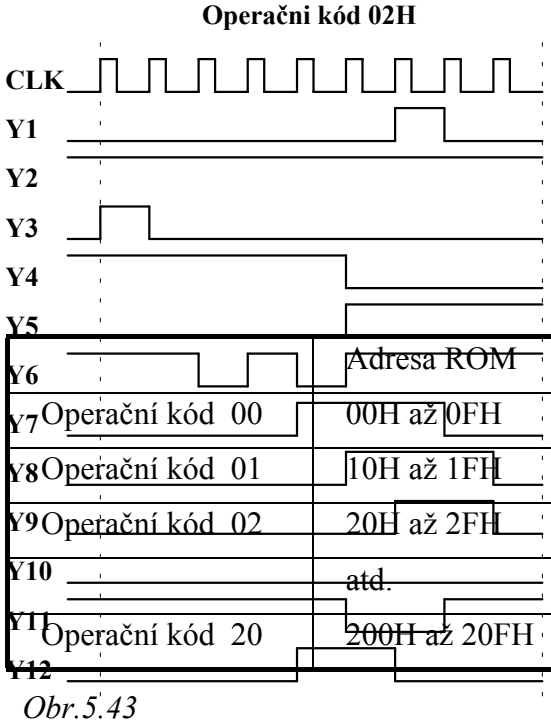
Než přistoupíme k podrobnějšímu popisu vlastností těchto obvodů, ukážeme si jakým způsobem byla vytvořena jejich vnitřní struktura a pokusíme se tak přiblížit použití těchto obvodů v návrzích sekvenčních obvodů. Historicky se obvody této skupiny vyvíjely mimo bitově orientované návrhové technologie. Jejich základ poprvé popsal v roce 1950 M.V.Wilkes jako techniku pro jednoduchý návrh počítačových řídících jednotek využívajících mikroprogramování. Mikroprogramování v podstatě reprezentuje logickou návrhovou techniku, která přináší často řadu výhod oproti technikám založeným na Boolovských rovnicích a stavových diagramech. Mikroprogramování se od programování v jazyce symbolických adres liší jen velmi málo, protože mají obdobný formát instrukčního souboru i programových struktur. Rozdíl lze vidět v tom, že mikroprogramování podporuje návrh obvodového řešení (hardware). Oproti tomu program jazyka symbolických adres je častěji spojován s manipulací s abstraktními daty. Jednou z možných realizací logického sekvenčního obvodu je tzv. automodifikace registru obr.5.40, která vlastně představuje klasický návrh sekvenčního obvodu soustředěný do dvou obvodů, kde registr realizuje paměťovou funkci obvodu a paměť ROM (PROM) realizuje kombinační část obvodu obr. 4.1. Předpokládejme nyní, že výstupy logického sekvenčního obvodu představují sériově vysílaná binární slova. Požadované výstupy (bity), které řídí logické zařízení, jsou uloženy v paměti ROM, která je periodicky čtena. U obvodu nedochází pouze k záměně pořadí registru, který se nazývá adresový, a paměti ROM obr. 5.41. Vracející se vodiče z paměti ROM, které spolu se vstupy označenými OPCODE, vytvářejí následující adresu generované sekvence, ale představují i následující adresu generované sekvence odpovídající danému operačnímu kódu OPCODE (stavu vstupních



Obr.5.40



Obr.5.41



Obr.5.43

Adresa	Obsah ROM	Adresa	Obsah ROM
20H	42E1H	25H	8F26H
21H	42A2H	26H	1F37H
22H	40A3H	27H	5B28H
23H	42A4H	28H	4320H
24H	C4A5H		

Obr.5.42

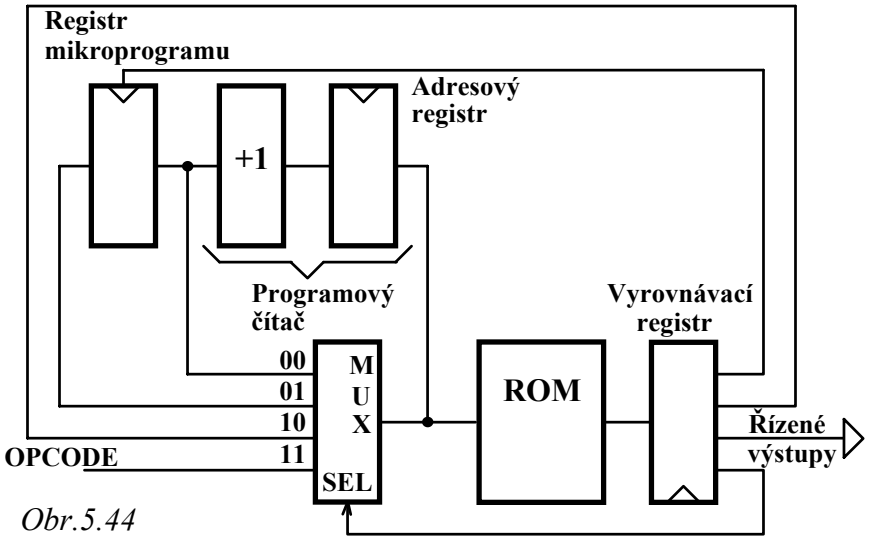
proměnných). Na obr.5.42 je uveden obsah paměti ROM a odpovídající generovaná posloupnost na řízených výstupech pro operační kód 02H. Operační kód tvoří horní část adresy, vracející se vodiče z paměti ROM tvoří spodní čtyři bity adresy. Obsah ROM představuje na nejnižších čtyřech bitech vracející se vodiče a další bity realizují řízené výstupy Y12,Y11,.. ,Y2,Y1.

Každé slovo paměti ROM se nazývá mikroinstrukce a sekvence mikroinstrukcí se nazývá mikroprogram. Tato sekvence není běžnému uživateli přístupná a může si

	Adresa ROM	Obsah paměti ROM
Operační kód 00	00H až 0FH	Mikroprogram pro instrukci DEC
Operační kód 01	10H až 1FH	Mikroprogram pro instrukci ADD
Operační kód 02	20H až 2FH	Mikroprogram pro instrukci SUB
	atd.	
Operační kód 20	200H až 20FH	Mikroprogram pro instrukci MOV

ji vyvolat pomocí OPCODE tj. pomocí (makro) programu, který je uložen v systémové (operační) paměti.

Tato konfigurace umožňuje pro daný počet instrukcí vytvořit odpovídající mikroprogramy obr.5.43. (např. o 16 mikroinstrukcích). Každý mikroprogram bude muset obsahovat i sekvenci pro načtení dalšího operačního kódu, což u počítačů s velkým počtem instrukcí by vedlo k degradaci systému. Mnohem efektivnější cestou je realizace skoku do mikropodprogramu, z kterého se vrátíme do místa odkud byl zavolán. Struktura z obr. 5.41 nám však tuto operaci neumožňuje a musíme ji proto rozšířit o tyto obvody a vlastnosti:

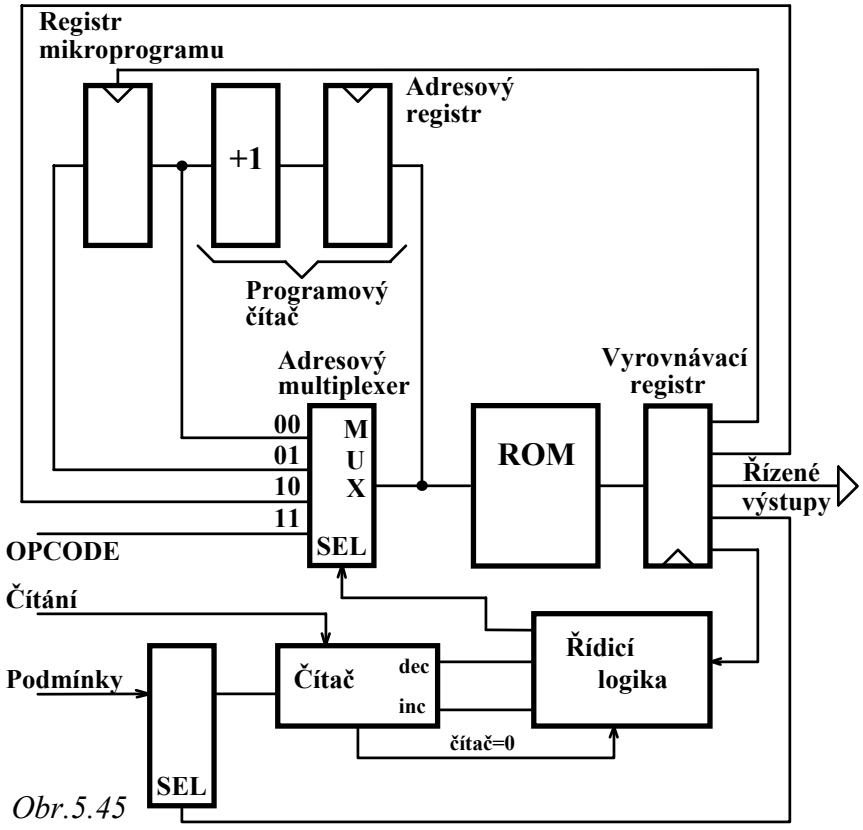


Obr.5.44

- Registr mikroprogramu uchovávající adresu následující mikroinstrukce, která bude vykonána po návratu z volaného mikropodprogramu.
- Adresový multiplexer umožňující naplnění adresového registru obsahem registru mikroprogramu, počáteční adresou mikroprogramu, adresou následující instrukce a adresou skoku.
- Skoková adresa by měla mít šířku umožňující umístění mikropodprogramu na libovolné adrese paměti ROM.
- Výstup adresového registru zvětšený o hodnotu jedna pro zajištění sekvenčního čtení instrukcí.
- Vstup počáteční adresy mikroprogramu vytvořené z operačního kódu OPCODE, jehož funkce byla již popsána u obr.5.41, doplněného na spodních bitech nulami.

Na obr.5.44 je zobrazena struktura, která splňuje tyto požadavky. Jejím základem je adresový multiplexer, který umožňuje čtení nového operačního kódu (JMP OP, SELMUX=11), skok do podprogramu (JMP, SELMUX=10, v registru mikroprogramu zůstává uschována následující adresa), návrat z podprogramu (RET, SELMUX=01) a vykonání následující mikroinstrukce (SELMUX=00), její adresa je vytvořena v čítači mikroinstrukcí. Adresový registr byl přemístěn do větve vytvářející následující adresu a byl tak vytvořen adresový (programový) čítač. Přesunem bylo navíc odstraněno zpoždění operačního kódu o jednu periodu hodin.

Dosud navržená jednotka umožňuje sekvenční čtení mikroinstrukcí, volání podprogramů a návrat z nich a skok na libovolnou adresu v paměti ROM. Jednotka nemá implementovánu instrukci podmíněného větvení, tzn. schopnost po sekvenci vykonaných mikroinstrukcí provést rozvětvení na základě stavu vnější nebo vnitřní podmínky. K realizaci větvení je třeba pozměnit ovládání multiplexeru tak, aby vnější podmínku ignoroval nebo v závislosti na její hodnotě provedl následující instrukci nebo skok na definovanou adresu. Popisovaná jednotka zatím neumožňuje realizaci programových smyček s proměnnou hodnotou jejich opakování. Proto kromě implementace podmíněného větvení, začleníme do jednotky čítač, jehož stav bude zaváděn do řídící logiky podmíněného větvení. Na obr.5.46 je zobrazena řídící jednotka, která podporuje podprogramy, smyčky, podmíněné i nepodmíněné větvení. Jednotku lze použít k realizaci řídící jednotky počítače, ale i jiného logického zařízení. U jednotky můžeme rychle měnit mikroprogram (chování řídící jednotky) bez změny její obvodové struktury. Popsaná jednotka je velmi podobná řídící jednotce v čipu Am29PL141, který byl jako první zástupce skupiny jednočipových kontrolérů Am29CLP-100 uveden na trh. Na jejím základě byl vytvořen obvod Am29PL142 s rozšířenou pamětí, registrovanými vstupy a dvouúrovňovým zásobníkem. Další generace obvodů jako je Am29-CPL142 (152) a Am-29CPL144 (154), přináší další rozšíření paměti a zásobníku (CPL144, CPL154), malý ztrátový výkon, vysokou pracovní rychlost 25 až 30 MHz a CMOS technologii plně kompatibilní s bipolárními obvody TTL.

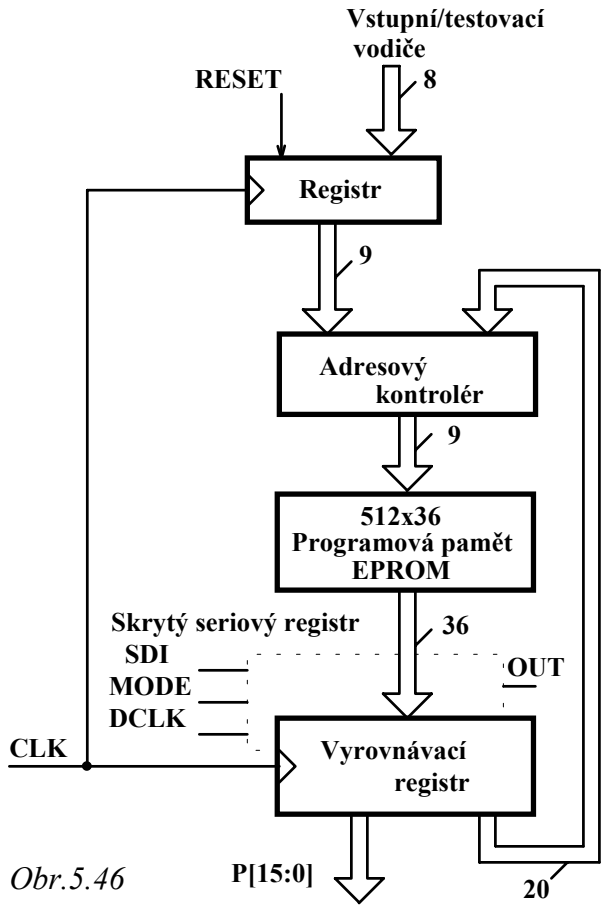


Obr.5.45

5.4.2. Popis obvodu AM29CPL154

Jednočipový programovatelný kontrolér Am29CPL154, vyrobený v CMOS technologii, umožňuje řízení logických zařízení pomocí programových sekvencí. Podmíněné i

nepodmíněné skoky, smyčky a volání podprogramů ovládané vnějšími testovacími vstupy přináší návrháři výkonné nástroje k vytvoření řídících sekvencí. Vlastnosti obvodu umožňují řízení rozmanitých logických obvodů a jednotek, jako je správa registrů, ALU,I/O, přerušovací logika a jednotky pro kontrolu sběrnice. Obvody FPC jsou mazatelné ultrafialovým zářením a programovatelné komerčními programátory. Obvody se též vyrábí i pro velké série v plastovém pouzdře DIP a PLCC bez okénka a jsou tudíž programovatelné pouze jednou.



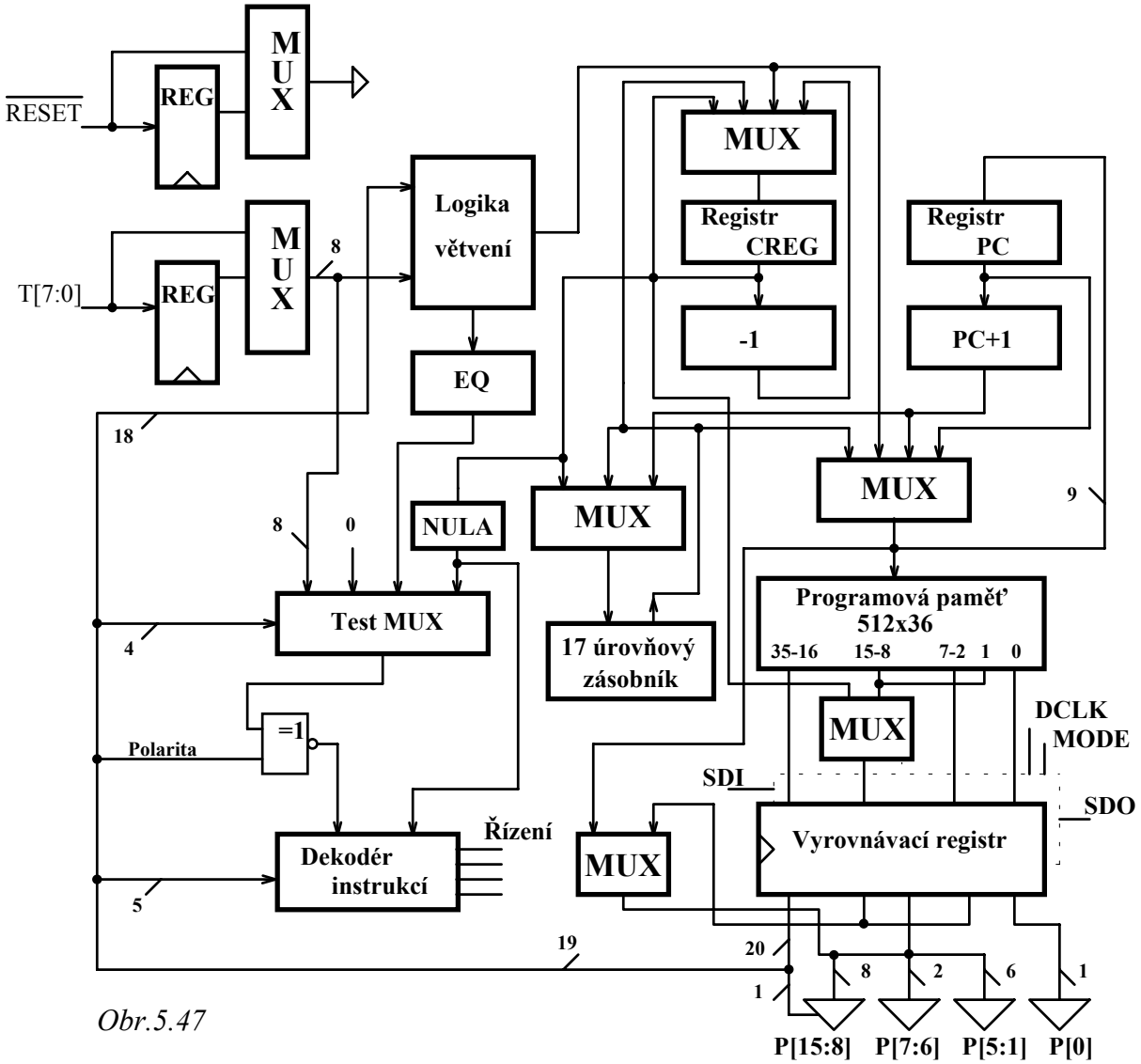
Obr.5.46

Na obr.5.46 je zobrazeno zjednodušené blokové zapojení obvodů skupiny Am29CPL100, které se skládá z adresového kontroléru, programové paměti, vyrovnávacího registru, sériového skrytého registru a registru testovacích vstupů. Vstupy adresového kontroléru, který je srdcem FPC, jsou testovací vstupy a uvnitř obvodu vracející se vodiče z vyrovnávajícího registru nesoucí masku pro testovací vstupy nebo adresu skokové instrukce. Adresový kontrolér generuje adresu následující mikroinstrukce, která po vybavení z programové paměti 64,128 nebo 512x34 (nebo 36) bitů je zapsána hodinovým signálem do vyrovnávajícího registru. Stejným hodinovým signálem jsou do registru uloženy i hodnoty testovacích vstupů a signálu RESET. Na obr.5.47 je zobrazeno blokové zapojení obvodu AM29CPL144(154). Základ tvoří adresový multiplexer typu jeden ze čtyř, který umožňuje definovat následující adresu jako:

- současnou adresu PC (adresu právě vykonávané mikroinstrukce)
- následující sekvenční adresu z programové paměti PC+1 (pro normální sekvenční vykonávání instrukcí)
- adresu uloženou v zásobníku
- adresu definovanou výstupy vyrovnávajícího registru k realizaci větvení typu GOTO.

Možnost výběru současné adresy PC jako adresy následující instrukce slouží k realizaci smyčky čekající na splnění zadané podmínky (dílčí události). Tato funkce inteligentního sekvenčního obvodu je nezbytná pro spojením s rozmanitými procesory a periferiemi.

Adresová logika kontroléru má začleněný zásobník, jehož výstup je možné přenést do čítače (o kterém bude pojednáno níže) nebo do programového čítače. Vstupní hodnotou zásobníku může být buď PC+1 (návratová adresa z podprogramu), běžný obsah vrcholu zásobníku TOS (pro smyčky na TOS) nebo obsah čítače (slouží pro bezprostřední uložení hodnoty čítače). Blok čítače, který se používá pro časování je tvořen registrem (CREG) a multiplexerem přepínajícím jeden ze čtyř zdrojů pro CREG. Vstupem může být již zmíněná hodnota ze zásobníku, nebo počáteční hodnota určená mikroinstrukcí nebo vnějšími vstupy. Zdrojem však může být i současná hodnota registru CREG nebo dekrementovaná hodnota registru CREG pro realizaci



Obr.5.47

iteračních smyček. Při každé iteraci je čítač snížen o hodnotu jedna a po dosažení nulové hodnoty jsou iterace zastaveny. Vstup RESET FPC nastavuje programový čítač na samé jedničky (na nejvyšší adresu).

Logika kontroly větvení vytváří adresy pro násobné větvení nebo pro podmíněné příkazy typu IF-THEN-ELSE. V každé instrukci mikroprogramu si může návrhář definovat testovací

podmínku pro vykonání instrukce. Tím je umožněno monitorování jak externích, tak interních událostí. U mikroinstrukcí si uživatel může nastavit polaritu testovací podmínky (testovacích vstupů) a výstupů a proto není nutné invertovat vstupní nebo výstupní signály. Programové větvení umožňuje skoky na adresu uloženou v datovém poli mikroinstrukce nebo definovanou maskovanými vnějšími vstupy. Užití vnějších vstupů jako následující adresy, podporuje mnohacestné větvení programu. Přednastavení čítače (registru CREG) hodnotou vnějších vstupů podporuje proměnné iterační smyčky. Na každé adrese programové paměti je uložena jedna mikroinstrukce (jeden stav obvodu), která se skládá ze stavu každého výstupu použitého k řízení periferií, z pole následujícího operačního kódu, bitu polarity, testovacího a datového pole a aktivačního bitu OE. Testovací podmínkou, pro podmíněné větvení může být stav vnějšího vstupu, vnitřního příznaku EQ určujícího shodu mezi vnějšími vstupy a hodnotou uloženou v mikrokódu, nulovost čítače CREG nebo nespecifikovaná podmínka (UNCOND). Výstupy programové paměti jsou přivedeny na vyrovnávací registr, který umožňuje čtení následující instrukce v době, kdy současná instrukce je vykonána.Spodních 16 výstupních bitů vyrovnávajícího registru je možné použít pro řízení definované návrhářem. Z těchto 16 bitů je spodních 8 bitů dvoustavových a horních 8 bitů třístavových, ovládaných aktivačním bitem OE uloženým v mikroinstrukci. Je-li potřeba více výstupních bitů k řízení aplikace, může návrhář použít rozšířeného módu obvodu Am29CPL100 k adresování vnějších pamětí s registrovanými výstupy (Am27S65A).

FPC pracuje ve dvou módech: normálním a diagnostickém. V normálním módu je mikroinstrukce vykonávána pro každý hodinový cyklus. Když je FPC programován pro použití k diagnostice, potom je skrytý registr SSR aktivován. Ten umožňuje jednoduché testování systému a identifikaci problému na úrovni individuálního IO. Test obvodu se provádí tak, že instrukce je posouvána sériově do SSR a potom paralelně načtena do vyrovnávacího registru. Výsledek vykonání instrukce je přenesen z vyrovnávacího registru do SSR, z kterého může být vysunut pro další analýzu.

Jak vyplývá z předešlého textu, lze každou mikroinstrukci rozdělit do několika částí (polí) jejichž uspořádání vytváří formát mikroinstrukce. U probíraných kontrolérů existují dva formáty mikroinstrukcí:

- všeobecný a porovnávací formát mikroinstrukce.

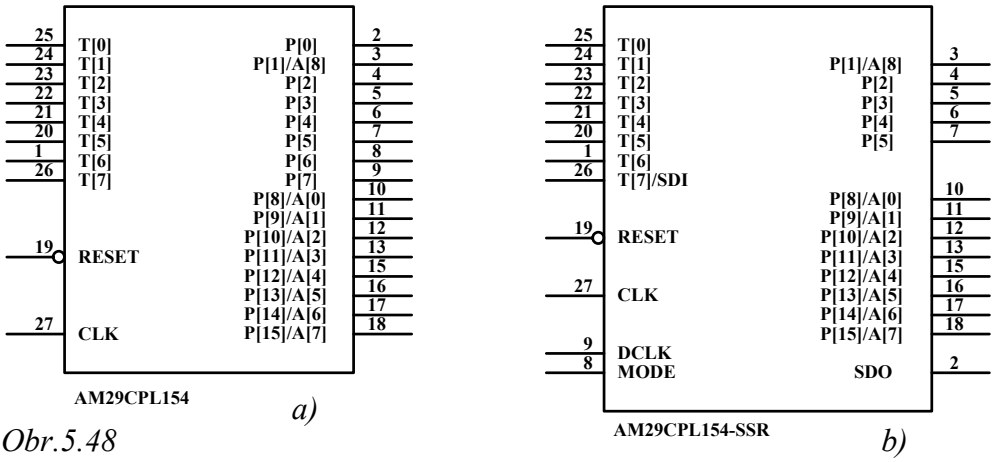
Spodních 16 bitů v každém formátu obsahuje 16 uživatelem řízených výstupních bitů, které se objevují na výstupech FPC P[15,0]. Zbývajících 20 bitů je ve všeobecném instrukčním formátu (CPL154) přiřazeno následovně:

Bity	Význam
16-24	Data (adresa podmíněného větvení, testovací vstupní maska nebo hodnota čítače)

25-28	Testovací pole (určuje, který z osmi vstupních signálů použijeme jako podmínkový kód) 0000 = T[0], 0001 = T[1], až 0111 = T[7], 1000 = EQ, 1001 = CREG a 1010 až 1111 = nepodmíněné větvení. Pravdivost nepodmíněného větvení dosáhneme výběrem UNCOND a POL=1.
29	Polarita (umožňuje uživateli určit zda testovací podmínka se vykonává na její pravdivost či nikoliv)
30-34	Operační kód (5 bitové pole operačních kódů určuje jednu z 28 mikroinstrukcí k vykonání)
35	Aktivace výstupů OE (je-li nastaven do 0, pak výstupy P[15,8] jsou v třetím stavu).

Ve formátu porovnávací mikroinstrukce, je zbývajících 20 bitů přiřazeno následovně:

Bity	Význam
16-23	Data (Maska pro vymaskování vstupů T[7,0])
24-31	Konstanta (určuje 8 bitovou konstantu pro porovnávání s hodnotou vnějších vstupů)
32 -34	Operační kód (instrukce porovnání 100)
35	Aktivace výstupů OE (je-li nastaven do 0, pak výstupy P[15,8] jsou v třetím stavu).



Obr.5.48

Na obr.5.48a je zobrazena normální konfigurace obvodu a na obr.5.48b je konfigurace diagnostická SSR. Jednotlivé vývody obvodu AM29CPL154 mají následující význam:

CLK - Hodinový vstup. Vzestupnou hranou hodin jsou ukládány hodnoty programového čítače PC, registru (CREG), zásobníku, překrývacího registru nebo i vnějších vstupů, vstupu RESET a příznaku EQ, jestliže jsou ttyto vstupy konfigurací nastaveny k interní synchronizaci.

P[15:8] - Výstupy. Horních osm všeobecně použitelných výstupů je aktivováno signálem OE z překrývacího registru. Je-li OE=1, výstupy jsou aktivovány,

v opačném případě jsou ve stavu vysoké impedance. V rozšířeném módu, po naprogramovaní bitu (EXP), jsou vývody P[1] a P[15:8] nastaveny na výstup adresy A8 a A7 až A0 z multiplexeru čítače instrukcí. Ty mohou být použity k adresování vnějších pamětí s externě registrovanými výstupy. Obsah vnitřního čítače CREG může být přenesen na řídící výstupní vývody P[1] a P[15:8] užitím instrukce OUT.

P[7:0] - Výstupy. Spodních osm všeobecně použitelných výstupů, které jsou trvale aktivované. V SSR módu se vývod P[7] stává hodinovým vstupem DCLK, P[6] se stává kontrolním vstupem MODE a P[0] se stává výstupem sériových dat SDO. SDO a SDI jsou vzorkovány DCLK podle řídícího bitu MODE. V rozšířeném módu (naprogramován bit EXP) se výstup P[1] stává výstupem nejvyššího bitu adresy A8).

RESET - Nulování. Jestliže je signál interně registrovaný, potom první vzestupná hrana hodinového impulzu zapíše stav do registru a teprve následující způsobí nastavení adresy programového čítače na samé jedničky (adresa 511). Je-li vstup naprogramován jako neregistrovaný potom po přechodu RESET 1->0 je na výstup multiplexeru PC v době inicializace přivedena adresa 511 a první hodinový impulz je zapíše do vyrovnávacího registru a vynuluje příznak EQ. Nenaprogramovaný stav odpovídá registrovanému signálu RESET.

T[7:0] - Vnější/Testovací vstupy. V podmíněných instrukcích mohou být použity testovací vstupy jako individuelní podmínky mikroinstrukcí určené 4 bitovým testovacím polem v překrývacím registru. Vstupy mohou být též použity jako adresa větvení (podpora mnohacestného větvení) nebo jako vnější vstupní hodnota pro registr CREG. Každý z těchto vstupů má v EPROM přiřazený bit, kterým lze nastavit vstup jako registrovaný. V diagnostickém režimu se stává T[7] vstupem pro sériová data SDI.

Instrukční soubor obvodu Am29CPL154

Instrukční soubor obvodu Am29CPL154 se skládá z 28 instrukcí, které lze rozdělit do 6 kategorií:

- instrukce programového větvení GOTOPL, GOTOTM, GOTOSTK, FORK
- instrukce podmíněného volání CALPL, CALTM, RET, RETPL
- instrukce se zásobníkem PSH, PSHPL, PSHTM, PSHCNTR, POP, POPCNTR
- skokové instrukce LPPL, LPTM, LPSTK, WAITPL, WAITTM
- instrukce naplnění čítače LDPL, LDTM

- ostatní instrukce DEC, DECPL, DECTM, DECGOPL, CONT, OUTPUT, CMP

Oper.kód	Mnemonika	Popis Jazyka ASM14X
19	GOTOPL	IF (cond) THEN GOTO PL(data)
		Podmíněné větvení na adresu určenou programovým návěštím PL(data).
0F	GOTOTM	IF (cond) THEN GOTO TM(data)
		Podmíněné větvení na adresu definovanou TM (vnější vstupy T[7:0] jsou maskovány bitovou maskou uloženou v datovém poli).
03	GOTOSTK	IF (cond) THEN GOTO (STACK)
		Podmíněné větvení na adresu uloženou ve vrcholu zásobníku (TOS).
18	FORK	IF (cond) THEN GOTO PL(data) ELSE GOTO (STACK)
		Podmíněné větvení na programové návěští PL(data) nebo na adresu určenou TOS.
1C	CALPL	IF (cond) THEN CALL PL(data)
		Podmíněný skok do podprogramu s adresou PL(data). Návratová adresa PC+1 je uložena do TOS.
1E	CALTM	IF (cond) THEN CALL TM(data)
		Podmíněný skok do podprogramu s adresou určenou TM (viz.GOTOTM). Návratová adresa PC+1 je uložena do TOS.
02	RET	IF (cond) THEN RET
		Podmíněný návrat z podprogramu. TOS vydává návratovou adresu.
0	RETPL	IF (cond) THEN RET, LOAD PL(data)
		Podmíněný návrat z podprogramu a načtení registru CREG hodnotou z PL(data). TOS představuje návratovou adresu z podprogramu.
04	LDPL	IF (cond) THEN LOAD PL(data)
		Podmíněné načtení CREG z PL(data)
06	LDTM	IF (cond) THEN LOAD TM(data)
		Podmíněné načtení CREG z TM(data)
15	PSH	IF (cond) THEN PUSH
		Podmíněné uložení PC+1 do vrcholu zásobníku TOS.
14	PSHPL	IF (cond) THEN PUSH, LOAD PL(data)

		Podmíněné uložení PC+1 do vrcholu zásobníku TOS a naplnění CREG hodnotou PL(data).
16	PSHTM	IF (cond) THEN PUSH, LOAD TM(data)
		Podmíněné uložení PC+1 do vrcholu zásobníku TOS a naplnění CREG hodnotu TM.
07	POP	IF (cond) THEN POP
		Podmíněný návrat hodnoty z vrcholu zásobníku
05	PSHCNTR	IF (cond) THEN PUSH TO (CREG)
		Podmíněné uložení hodnoty CREG do vrcholu zásobníku TOS.
17	POPCNTR	IF (cond) THEN POP TO (CREG)
		Podmíněný návrat hodnoty z TOS do registru CREG.
08	DEC	IF (cond) THEN DEC
		Podmíněné zmenšení registru CREG o jedničku.
OC	DECPL	WHILE (CREG<>0) WAIT ELSE LOAD PL(data)
		Podmíněné stání dokud čítač není nulový. Pak je načtena do CREG hodnota z PL(data). Tato instrukce se používá pro časování signálů. Jestliže CREG není nulový, je načítána stejná instrukce a CREG je dekrementován. Časování je úplně dokončeno až po vynulování CREG, kdy je načtena další instrukce a načtena nová hodnota do CREG.
OE	DECTM	WHILE (CREG<>0) WAIT ELSE LOAD TM(data)
		Podmíněné stání dokud čítač není nulový. Pak je načtena hodnota do CREG z TM(data) (viz.GOTOTM). Jestliže CREG není nulový, je načítána stejná instrukce a CREG je dekrementován. Časování je úplně dokončeno až po vynulování CREG, kdy je načtena další instrukce a načtena nová hodnota do CREG z TM.
1B	DECGOPL	IF (cond) THEN GOTO PL(data)ELSE WHILE (CREG<>0) WAIT
		Podmíněné čekání nebo skok. Současná instrukce bude opakovaně vykonávána dokud se podmínka nestane pravdivou nebo čítač nedosáhne nuly. Jestliže podmínka je pravdivá dojde ke skoku na adresu určenou PL(data). Jestliže čítač dosáhne nuly, aniž by bylo dosaženo pravdivosti podmínky, pokračuje další instrukcí.
08	LPPL	WHILE (CREG<>0) LOOP TO PL(data)

		Podmíněná smyčka na návěští PL(data). Tato instrukce je umístěna na konec smyčky. Jestliže CREG není nulový, je dekrementován a je proveden skok na adresu PL(data). Je-li CREG=0 je instrukce kompletní a pokračuje další instrukcí. Tato instrukce nezávisí na platnosti/neplatnosti podmínky.
0A	LPTM	WHILE (CREG<>0) LOOP TO TM(data)
		Podmíněná smyčka na adresy TM. Tato instrukce musí být umístěna na konci smyčky. Jestliže CREG není nulový, je dekrementován a je vykonáno větvení na adresu určenou TM(data). Je-li CREG=0 instrukce je dokončena a jsou vykonávány následující instrukce.
0F	LPSTK	WHILE (CREG<>0) LOOP TO (STACK)
		Podmíněná smyčka na adresu ve vrcholu zásobníku TOS. Je-li CREG$0, CREG je dekrementován a větvení je vykonáno na adresu v TOS. Je-li CREG=0 Instrukce je dokončena a jsou vykonávány následující instrukce.
1A	WAITPL	IF (cond) THEN GOTO PL(data) ELSE WAIT
		Podmíněné čekání. Současná instrukce je znovu načítána a vykonávána, dokud se nestane podmínka pravdivou. Potom je vykonáno větvení na adresu v PL(data).
1B	WAITTM	IF (cond) THEN GOTO TM(data) ELSE WAIT
		Podmíněné čekání. Současná instrukce je znovu načítána a vykonávána, dokud se nestane podmínka pravdivou. Potom je vykonáno větvení na adresu v TM(data).
0D	CONT	CONTINUE.
		Další načtená instrukce je vykonána nepodmíněně. Instrukce může být použita k vynulování příznaku EQ, použitím příznaku EQ jako testovací podmínky.
01	OUTPUT	IF (cond) THEN OUTPUT
		Registr CREG bude vyslán na vývody P[1] a P[15:8] během následujícího hodinového cyklu. Pozornost musí být věnována zajištění aktivovanému výstupu OE=1 v následující instrukci.
10-13	CMP	CMP TM(mask) TO PL(constant)
		Tato instrukce vykoná bitově EX-OR mezi TM (viz.GOTOTM) a KONSTANTOU (P[31:24]). Jestliže bude TM stejné jako KONSTANTA, příznak EQ bude nastaven do log.1 a může být podle něj provedeno větvení v další instrukci. Nejsou-li stejné EQ zůstává nezměněn. To umožňuje sekvenceru porovnání na způsob suma produktů, následované jedinou větvící podmínkou.

Poznámka: Příznak EQ je nastaven do nuly při reset nebo je-li EQ vybrána jako testovací podmínka pro větvení. Podmíněný návrat na EQ jej ponechá v nezměněném stavu.

Návrh a ověření činnosti sekvenčního obvodu s FPC vyžaduje dobrou programovou podporu. Rodina Am29CPL100 je podporována asemblerem a simulátorem. Asembler PL14X konvertuje program napsaný v jazyce vytvořeném z výrazů typu IF-THEN-ELSE a WHILE, do souboru JEDEC, který užívají ostatní moduly jako je simulátor a komerčně vyráběné programátory. Asembler umožňuje definovat data jako byty nebo slova předcházená referenčními návěštími, dovoluje přiřazení hodnot bitům v binárním, oktalovém, dekadickém nebo hexadecimálním formátu. Simulátor slouží k ověření správné činnosti vytvořeného programu. Simulátor PL14X užívá soubor JEDEC (generovaný asemblerem PL14X) a soubor testovacích vektorů určující vstupní posloupnosti specifikované návrhářem. Vypočtené výstupní signály mohou být porovnány s očekávanými výstupními hodnotami specifikovanými v souboru testovacích vektorů. Dříve než přistoupíme k příkladům použití obvodů FPC včetně jejich programového vybavení, zmíníme se krátce o struktuře programů v asembleru ASM14X. Program se skládá ze dvou povinných a šesti nepovinných částí označených (*), které musí následovat v tomto pořadí.

1. **DEVICE** - za kterým následuje jeden z obvodů skupiny Am29CPL100 uvedený v závorce
2. **DEFAULT** (*) - část určující programování nebo ponechání nepoužitých paměťových míst.
3. **DEFINE** (*) - část definující přiřazení konkrétních hodnot symbolickým jménům.
4. **DEFAULT_OUTPUT** (*) - definuje výstupy P[15:0] pro případy, kdy nejsou uvedeny u instrukcí.
5. **TEST_CONDITION** (*) - určuje testovací podmínku pro instrukce, u kterých není uvedena.
6. **OUT_POLARITY** (*) - určuje polaritu výstupů.
7. **ARCHITECTURE** (*) - část definující režim obvodu tj. diagnostický (SSR), rozšířený (EXP) a neregistrované vstupy REGTx (REGRST).
8. Hlavní program

BEGIN

Návěští: výstupy, vlastní instrukce jazyka ASM14X; " Komentáře "

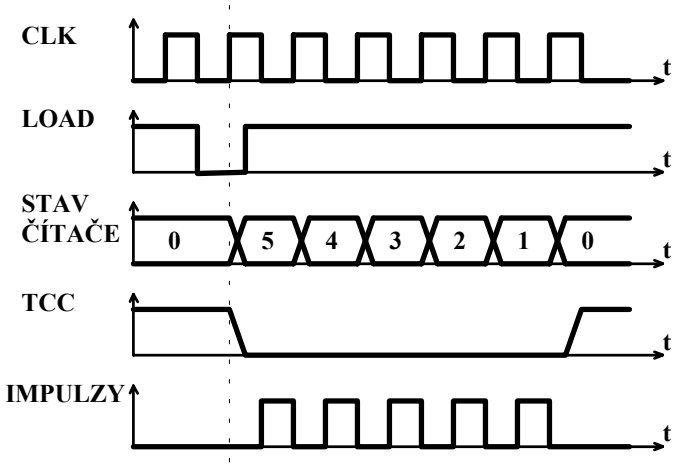
END.

Příklady použití FPC

Nyní si ukážeme dva jednoduché příklady na nahrazení obvodové realizace popsaným obvodem s příslušným programem. Jedná se pouze o ukázky realizací, u nichž by cenové porovnání vyšlo jednoznačně příznivě ve prospěch obvodového řešení. Bude-li pro nás

omezujícím faktorem prostor, potom realizace FPC se stává výhodnější i pro jednoduché řídící obvody.

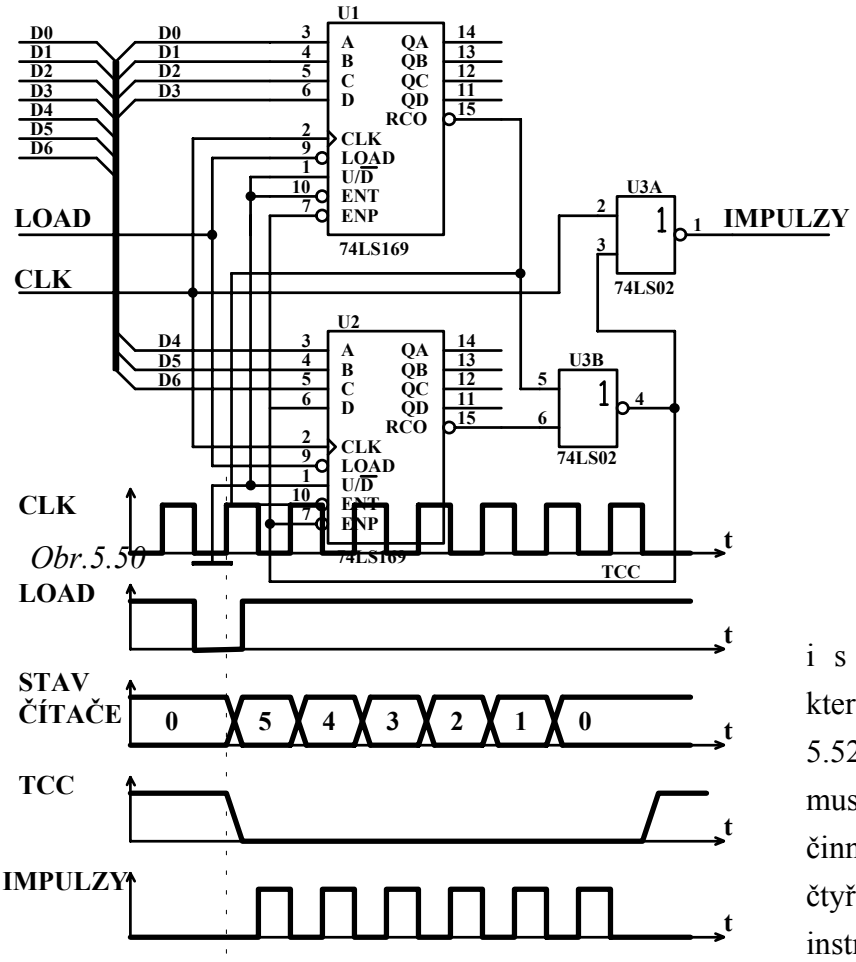
Příklad 5.11 Navrhněte řízený generátor hodinových impulzů jejichž počet je určen 7-bitovou hodnotou D6 až D0 a synchronizován signálem LOAD.



Obr.5.49

Činnost generátoru, jehož časové průběhy jsou zobrazeny na obr.5.49, lze rozložit do těchto kroků:

- čekání na signál LOAD
- načtení vstupní hodnoty do čítače a přechod výstupního signálu TCC do log.0.
- dekrementace čítače, generování hodinových impulzů a čekání na dosažení nulové hodnoty čítače
- nastavení výstupního signálu TCC do log. 1 a přechod na první krok

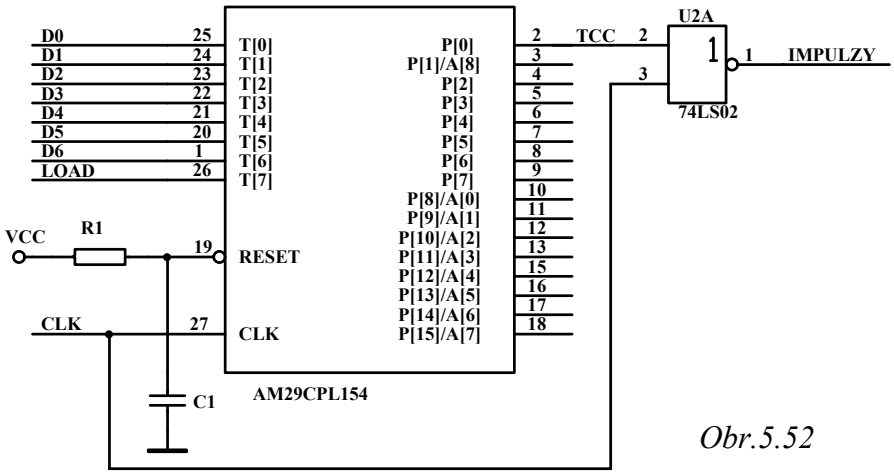


Obr.5.50

Obr.5.51

Všechny tyto kroky jsou realizovány sekvenčním obvodem z obr.5.50 s maximálním hodinovým kmitočtem $CLK = 1/(t_{pc\to RCO} + 2.t_{p02})$ =15,3 MHz zajišťující nenulovou šířku generovaných impulzů. Uvedený obvod můžeme realizovat i s popsaným obvodem FPC, který bude zapojen podle obr. 5.52. Program, který bude muset realizovat popsanou činnost se bude skládat ze čtyřech instrukcí. První instrukce bude zajišťovat

čekání na nulovost signálu LOAD (WAITPL), druhá instrukce načte hodnotu D6 až D0 (LDTM), výstup však nebude vynulován viz. časový diagram obr.5.51. Třetí instrukce bude nulovat signál TCC, dekrementovat obsah čítače CREG (LPPL) a čekat na jeho nulovost. Bude-li hodnota čítače nulová,vykoná se poslední instrukce, která ukončí generování hodinových impulzů TCC=1 a zajistí přechod do vyčkávacího režimu. Z časového diagramu obr.5.51 pro Prog.1. vyplývá, že obvod generuje o jeden hodinový impulz navíc než je zadaná hodnota. To lze odstranit zmenšením zadávané hodnoty při zachování časového diagramu nebo zmenšením přečtené hodnoty v programu, který by generoval hodinové impulzy o jednu periodu hodin později, než na obr.6. K příkladu je třeba poznamenat, že zapojení z obr.5.52 je vhodné pro kmitočty do 25MHz. Pro kmitočet generovaných impulzů do 12MHz je možné ušetřit ještě vnější člen NOR úpravou programu Prog.2 s tím, že generované hodinové impulzy budou mít poloviční kmitočet než signál CLK a budou generovány přímo výstupem P[0].



Obr.5.52

Program 1

```
DEVICE (CPL154)      " Řízený generátor signálu TCC "
OUT_POLARITY = 1111111111111111#b;
ARCHITECTURE
      REGT0=1 REGT1=1 REGT2=1 REGT3=1 REGT4=1 REGT5=1 REGT6=1
      REGT7=1; " Vstupy T[0] až T[7] jsou neregistrované "
          BEGIN
START:    0001#H, IF (T7=0) THEN GOTO PL(GENERUJ) ELSE WAIT;
            .ORG 20#H
GENERUJ:  0001#H, IF (PASS_DFLT) THEN LOAD TM(7F#H);
CEKEJ:    0000#H, WHILE (CREG<>0) LOOP TO PL(CEKEJ);
          0001#H, IF (CREG=0) THEN GOTO PL(START);
            .ORG 1FF#H
          0001#H, CONTINUE;
          END
```

```
PROM Contents: Prog.1.
hex <dec>     OE  OPCODE POL  TEST   DATA          OUTPUT
000 <  0>   [ 1 | 11010 | 1 | 0111 | 000100000 | 0000000000000001 ]
020 < 32>   [ 1 | 00110 | 1 | 1111 | 001111111 | 0000000000000001 ]
```

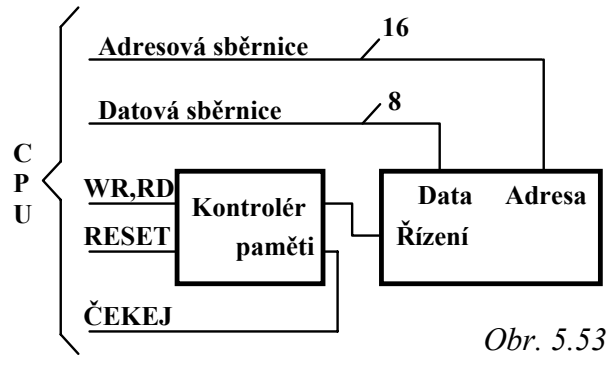
```
021 < 33>   [ 1 | 01000 | 1 | 0111 | 000100001 | 0000000000000000 ]
022 < 34>   [ 1 | 11001 | 0 | 1001 | 000000000 | 0000000000000001 ]
1FF <511>   [ 1 | 01101 | 1 | 1111 | 111111111 | 0000000000000001 ]
```

Program 2

```
DEVICE (CPL154)    " Řízený generátor hodinových signálů "
OUT_POLARITY = 1111111111111111#b;
ARCHITECTURE
      REGT0=1 REGT1=1 REGT2=1 REGT3=1 REGT4=1 REGT5=1 REGT6=1
      REGT7=1; " Vstupy T[0] až T[7] jsou neregistrované "
            BEGIN
START:      0000#H, IF (T7=0) THEN GOTO PL(GENERUJ) ELSE WAIT;
GENERUJ:    0000#H, IF (PASS_DFLT) THEN LOAD TM(7F#H);
CEKEJ:      0000#H, IF (CREG=0) THEN GOTO PL(START);
            0001#H, WHILE (CREG<>0) LOOP TO PL(CEKEJ);
            0000#H, IF (CREG=0) THEN GOTO PL(START);
              .ORG 1FF#H
            0000#H, CONTINUE;
            END.
```

```
PROM Contents: Prog.2.
hex <dec>  OE  OPCODE  POL  TEST      DATA          OUTPUT
000 <  0> [ 1 | 11010 | 1 | 0111 | 000000001 | 0000000000000000 ]
001 <  1> [ 1 | 00110 | 1 | 1111 | 001111111 | 0000000000000000 ]
002 <  2> [ 1 | 11001 | 0 | 1001 | 000000000 | 0000000000000000 ]
003 <  3> [ 1 | 01000 | 1 | 0111 | 000000010 | 0000000000000001 ]
004 <  4> [ 1 | 11001 | 0 | 1001 | 000000000 | 0000000000000000 ]
1FF <511> [ 1 | 01101 | 1 | 1111 | 111111111 | 0000000000000000 ]
```

Příklad 5.12 Navrhněte kontrolér paměťového systému 64kBx8bitů tvořeného dynamickými pamětmi, který zajistí generování občerstvovacích cyklů pamětí a synchronizaci procesorového systému při cyklu čtení a zápisu.

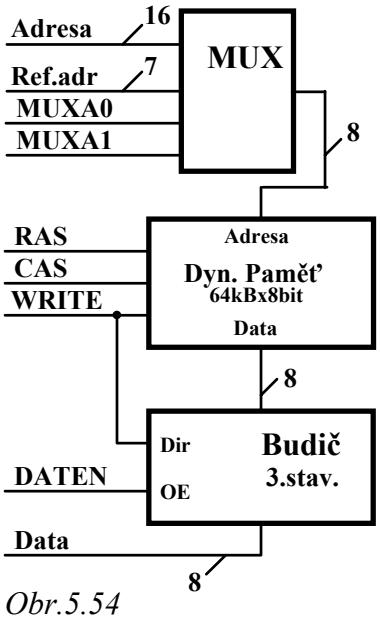


Obr. 5.53

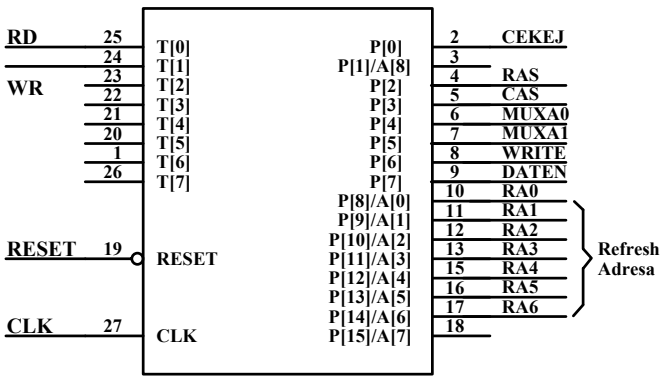
Na obr.5.53 je zobrazeno připojení paměťového systému s kontrolérem paměti k procesorovému systému. Paměťový systém obr.5.54 se skládá z dynamických pamětí 64kBx1bit, které mají 8 adresovacích vodičů, 8 datových vodičů a řídící signály CAS,RAS a WE. Potřebná 16 bitová adresa k adresování 64kB z CPU je rozdělena multiplexerem na dvě 8-bitové části tzv. řádkovou a sloupcovou adresu, jejichž přepínání zajistí signál MUXA0. Oddělení a posílení datových vodičů zajistí datový budič, jehož směr přenosu bude

řízen signálem WRITE a aktivován bude signálem DATEN. V této aplikaci budeme od řadiče požadovat tyto funkce:

- generování vlastních časovacích signálů pro řízení dynamické paměti
- Synchronizaci časování paměťového systému s procesorem
- Vytváření občerstvovacích cyklů pamětí v době mimo čtení a zápis



Obr.5.54



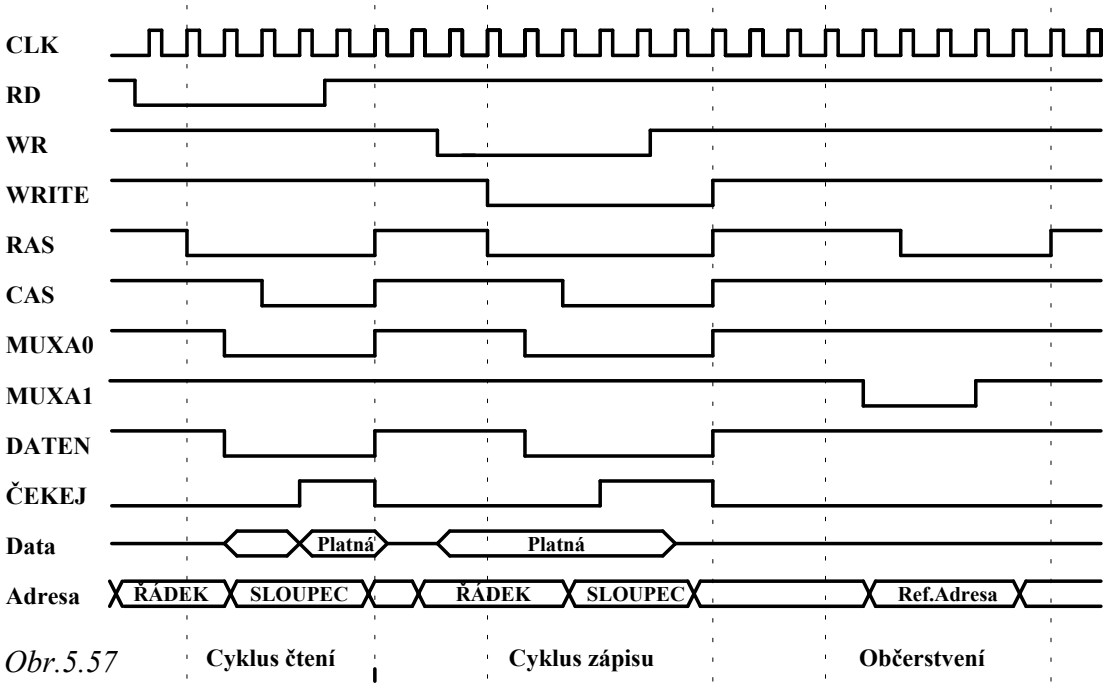
Obr. 5.55

PARAMETR	Min	Max
1. Doba předstihu adresy před RAS	0ns	
2. Doba přesahu adresy po RAS	25ns	
3. Doba předstihu adresy před CAS	0ns	
4. Doba přesahu adresy po CAS	35ns	
5. Trvání pulzu RAS	150ns	10000ns
6. Trvání pulzu CAS	85ns	10000ns
7. Doba předstihu READ před CAS	0ns	
8. Doba přesahu READ po náb.hran.CAS	0ns	
9. Doba předstihu WR před CAS	0ns	
10.Doba přesahu WR po CAS	30ns	
11.Doba předstihu dat před CAS	0ns	
12.Doba přesahu dat po CAS	35ns	
13.Doba zpoždění CAS po RAS	35ns	
14.Doba aktivace po CAS	85ns	
15.Doba aktivace po RAS	150ns	
16.Doba CAS v log.1	25ns	
17.Doba RAS v log.1	120ns	

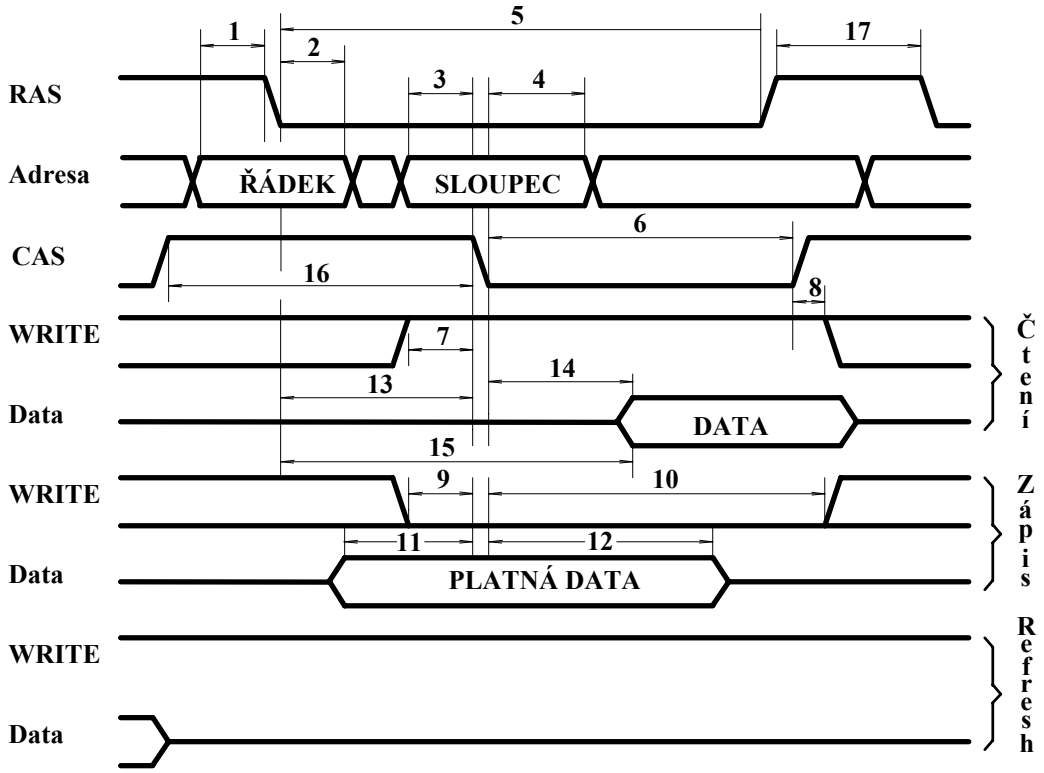
Tabulka 5.8

Na obr.5.55 je zobrazeno zapojení obvodu Am29CPL154 jako jednoduchého paměťového kontroléru generujícího signál ČEKEJ sloužícího k vytvoření signálu WAIT pro procesorovou jednotku k zajištění synchronizace cyklu paměti s CPU, MUXA0 pro přepínání řádkové

a sloupcové adresy, MUXA1 pro připínání občerstvovací adresy, RAS pro zápis řádkové adresy do dynamických pamětí, CAS pro zápis sloupcové adresy a aktivaci paměti pro zápis nebo čtení dat a signály WRITE a DATEN. Na obr.5.56 je zobrazeno typické časování dynamických pamětí pro cyklus čtení, zápisu a občerstvení pouze cyklem RAS. V tab.5.8 jsou uvedeny parametry např. pro paměť Intel 2164-15



Obr.5.57



Obr.5.56

Za předpokladu, že hodiny kontroléru CLK=25MHz (perioda 40ns) a multiplexer bude v technologii Fast ($t_{D\rightarrow Q_{max}}$ = 10,5ns), můžeme z časových údajů zjistit, že cyklus čtení, zápisu bude muset trvat minimálně 5 period kontroléru následovaných alespoň 3 cykly s RAS=log.1 a 6 period pro cyklus občerstvení. Na obr.5.57 je zobrazena předpokládaná činnost kontroléru pro čtení, zápis a občerstvení (refresh), kterou realizuje program Prog.3.

Program 3.

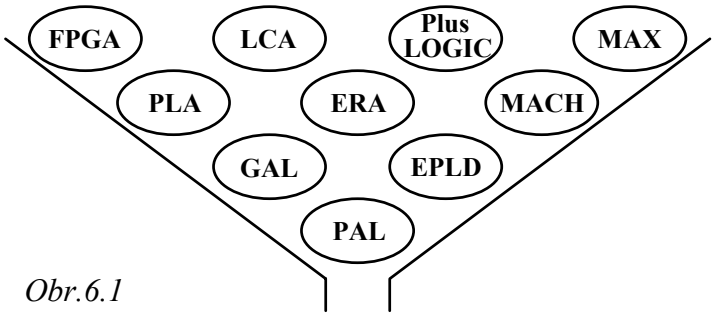
```
DEVICE (CPL154)    " Kontrolér dynamické paměti "
DEFINE
                    "Výstupy"       CEKEJ= 0001#H
                                    RAS  = 0004#H
                                    CAS  = 0008#H
                                    MUXA0= 0010#H
                                    MUXA1= 0020#H
                                    WRITE= 0040#H
                                    DATEN= 0080#H
                    "Vstupy"        RD= T0        WR= T1;

DEFAULT_OUTPUT = RAS+CAS+MUXA0+MUXA1+WRITE+DATEN;
TEST_CONDITION=PASS_DFLT;
OUT_POLARITY = 1111111111111111#B;
ARCHITECTURE
REGT0=1 REGT1=1 ;        " Neregistrované vstupy RD a WR "
```

```
          BEGIN
START:    ,LOAD PL(7F#H);   "POČÁTEČNÍ NAPLNĚNÍ REGISTRU CREG"
           "nebo  ,IF (PASS_DFLT) THEN LOAD PL(7F#H);
TESTUJ:   ,IF (RD=0) THEN CALL PL(CTENI);
          ,IF (WR=0) THEN CALL PL(ZAPIS);
REFRESH:  ,OUTPUT;                          "Registr CREG na P[1] A P[8]-P[15]"
          RAS+CAS+MUXA0+WRITE+DATEN,OUTPUT;      "MUXA1=0"
          CAS+MUXA0+WRITE+DATEN,OUTPUT;          "RAS=0"
          CAS+MUXA0+WRITE+DATEN,OUTPUT;
          CAS+MUXA0+MUXA1+WRITE+DATEN,CONTINUE;  "MUXA1=1"
          CAS+MUXA0+MUXA1+WRITE+DATEN, DEC;
                                            "CREG=CREG-1,MUXA1=1"
          ,IF (CREG=0) THEN LOAD PL(7F#H);      "RAS=1"
          ,GOTO PL(TESTUJ);                 "Navrat na test"
CTENI:    CAS+MUXA0+MUXA1+DATEN+WRITE,CONTINUE;  "RAS=0"
          CAS+MUXA1+WRITE,CONTINUE;         "MUXA0=DATEN=0"
          MUXA1+WRITE,CONTINUE;             "CAS=0"
          MUXA1+CEKEJ+WRITE,IF (RD=1) GOTO PL(KON_RD) ELSE WAIT;
                                            "Uvolni CEKEJ"
KON_RD:   ,CONTINUE;
          ,RET;
ZAPIS:    CAS+MUXA0+MUXA1+DATEN,CONTINUE;
                                            "RAS=WRITE=0"
          CAS+MUXA1,CONTINUE;               "MUXA0=DATEN=0"
          MUXA1,CONTINUE;                   "CAS=0"
          MUXA1+CEKEJ,IF (WR=1) GOTO PL(KON_WR) ELSE WAIT;
                                            "Uvolni CEKEJ"
KON_WR:   ,CONTINUE;
          ,CONTINUE;
          , RET;
             .ORG 1FF#H
          , CONTINUE;                       "Instrukce po RESET"
          END.
```

Závěrem lze říci, že obvody AM29CPL100 přináší nový trend v návrhu logických sekvenčních obvodů, kdy jedním nebo dvěma obvody lze snadno realizovat středně složité řídící sekvenční obvody, které by musely být realizovány 10,20 i více obvody MSI nebo obvodem FPGA. Porovnání FPC s obvody FPGA není asi účelné, protože FPGA jsou zcela universální obvody pro realizace kombinačních i sekvenčních logických obvodů. Naproti tomu FPC jsou specializované obvody pro realizaci rychlých řídících obvodů.

6. Programovatelné logické obvody

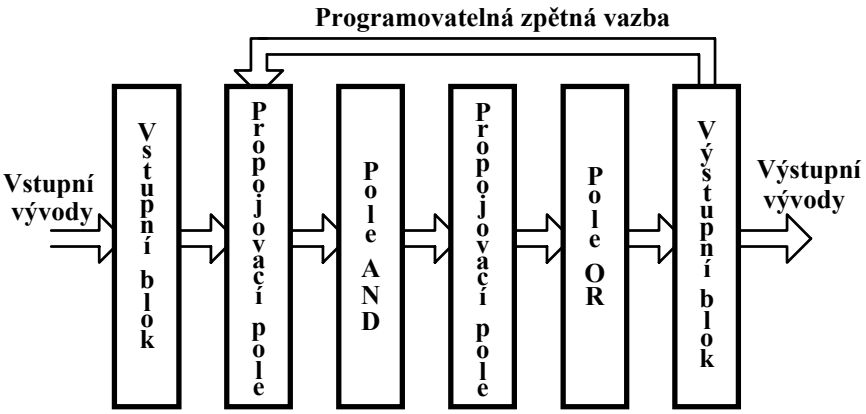
Programovatelné obvody byly vyvinuty s cílem poskytnout návrháři číslicových obvodů a systémů prostředky pro realizaci středně náročných obvodů. Byly tak vytvořeny mezistupně mezi možnostmi návrhu a realizace systémů z obvodů SSI a MSI na straně jedné a zákaznickými obvody na straně druhé. Každá z těchto krajních mezí má své výhody a nevýhody, pro každou existuje oblast ideálního použití. Obecně můžeme říci, že zákaznické obvody jsou vhodné pro velké série, kde se neuplatní vysoká cena jeho návrhu i dlouhá doba vývoje. Na straně druhé získáme výrazné zmenšení rozměrů, nižší cenu systému, vyšší výkon a spolehlivost. Oproti tomu standardní obvody SSI a MSI najdou uplatnění u jednodušších realizací a realizací s obvody LSI jako universální a pomocné obvody s nízkou cenou, snadnou dostupností a krátkou dobou implementace. Programovatelné obvody přináší nižší náklady na vývoj zařízení než u zákaznických obvodů, vyšší hustotu integrace proti obvodům SSI a MSI, ale hlavně krátkou dobu vývoje, snadnou reprodukovatelnost a dobrou přizpůsobivost k případným změnám návrhu. Právě snadné změny v návrhu je výrazně staví do protikladu k oběma krajním řešením. Možná největší výhodou obvodů PLD je velké množství výkonných a dostupných návrhových systémů.



Obr.6.1

V současné době můžeme rozdělit programovatelné obvody do zhruba deseti skupin zobrazených na obr.6.1, který představuje rozšiřující se škálu těchto obvodů různého stupně složitosti i vnitřní struktury. Uvedené obvody lze rozdělit podle vnitřní struktury na obvody založené na makrobuňkách (macrocell) a obvody tvořené programovatelnými logickými bloky, které jsou propojovány mezi sebou programovatelným polem spojek (Programmable Gate Array). V každé z těchto skupin můžeme provést rozdělení do dalších třech podskupin, charakterizujících blíže jejich vnitřní strukturu.

Architektura programovatelných polí patřících do skupiny obvodů založených na makrobuňkách vychází z faktu obr.6.2, že jakoukoliv logickou funkci lze vyjádřit součtovou formou (disjunktní formou - součtem součinů), kterou lze implementovat vhodným spojením hradel AND a OR. Programovatelnost propojení vstupů a výstupů k hradlům AND a OR umožňuje konstruovat součiny, které se následně sečtou. Počet vstupů, výstupů a programovatelnost propojovacího pole mezi nimi a poli hradel AND a OR potom rozlišují jednotlivé typy programovatelných logických polí patřících do výše zmíněné skupiny. Vstupní blok může obsahovat



Obr.6.2

vyrovnávací paměť, vstupní budiče realizující i inverze vstupních proměnných, logiku pro programovatelné povolení vstupu atd. Propojení mezi vstupy, výstupy a jednotlivými hradly je zajišťováno programovatelnými spínači realizovanými na bázi různých technologií. Spínačem může být pevné vodivé propojení (FUSE) programovatelné jeho přepálením (PROM) nebo spínač typu EPROM nebo EEPROM. Vlastní propojovací pole je buď programovatelné nebo je realizováno pevným spojením. Výstupní blok může obsahovat invertor, třístavový budič, registr (paměťový člen) nebo i poměrně složitou a dále programovatelnou výstupní logiku tzv. výstupní makrobuňku (Output macrocell). Výstup z tohoto bloku bývá přiveden zpětnou vazbou do propojovacího pole. Obvody této skupiny můžeme podle vlastností propojovacího pole dále dělit do těchto skupin:

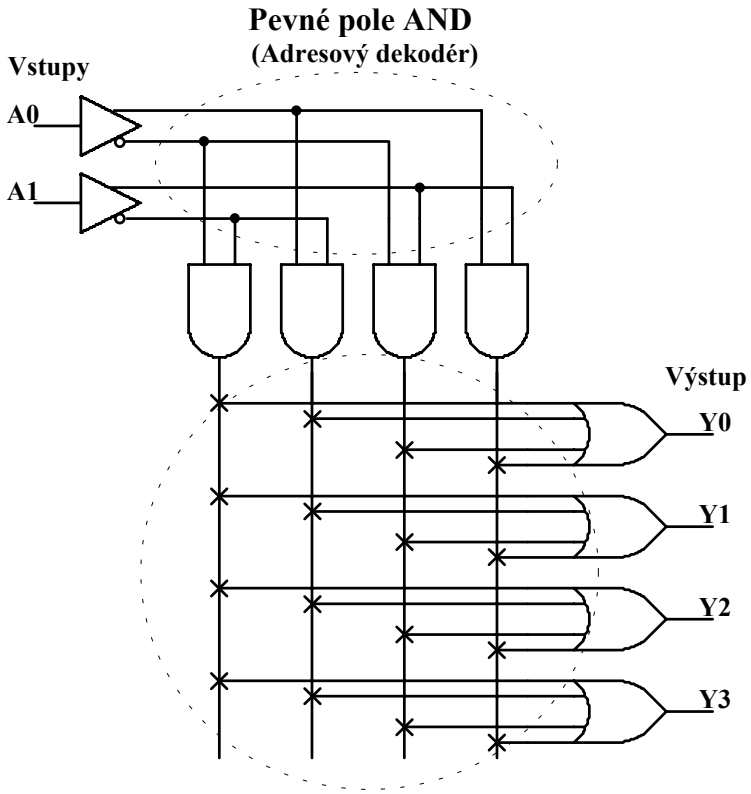
PAL,GAL,EPLD - mají programovatelnou matici AND a pevnou propojovací matici OR, v které se obvykle provádí součet 8 součinů. Architektura obvodů je pevná s malou možností změn.

PLA,PLS/PSG - mají programovatelnou matici AND i OR a proto je možné provádět velký počet součtu součinů. U obvodů PLS a PSG je navíc řada registrů s vnitřní zpětnou vazbou, která není uživateli přístupná. Obvody se používají pro realizaci řadičů, binárních generátorů (programmable sequencer) atd.

MAX,MACH, Plus LOGIC - jedná se obvody s mnoha úrovňovou programovatelnou propojovací maticí, rozšiřující polem hradel a polem programovatelných makrobuněk. Obvody se vyznačují schopností emulovat obvody PAL a PLA.

6.1. Obvody PROM

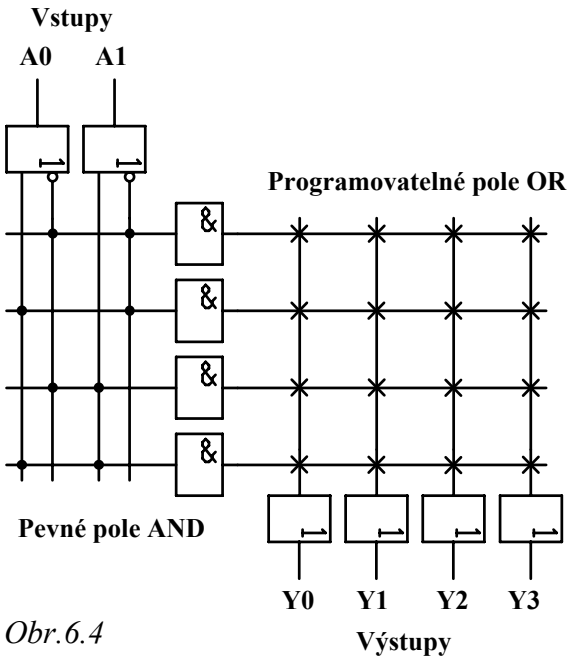
Řadu let nebyly obvody PROM (Programmable Read Only Memory) zařazovány do skupiny programovatelných logických polí, ačkoliv řada nejmenších pamětí PROM bývala často používaná jako adresové dekodéry, převodníky kódů apod. Větší paměti již bývaly používány při návrhu bipolárních mikroprocesorů pro uložení mikroinstrukcí. Z pohledu architektury obvodů PLD můžeme PROM chápat jako obvody s pevným propojovacím polem AND



Obr.6.3 **Programovatelné pole OR**

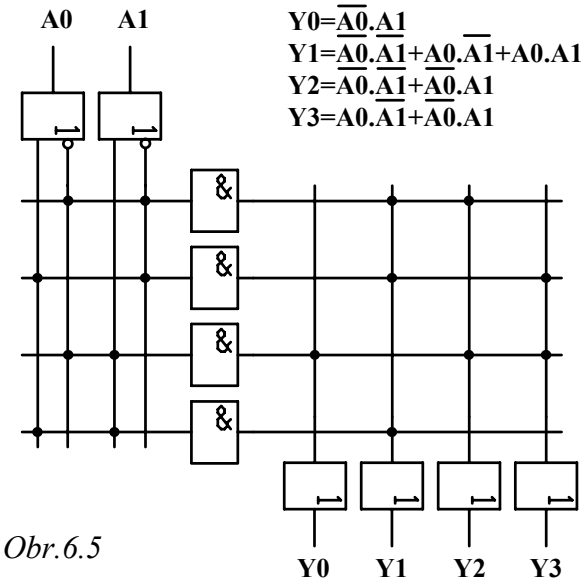
a programovatelným polem OR obr.6.3. Tato architektura, která realizuje úplnou součtovou formu logické funkce, umožňuje, aby libovolná výstupní kombinace hodnot byla programovatelně přiřazena libovolné kombinaci vstupních hodnot.

Na obr.6.4 je uveden příklad obvodu PROM se dvěma vstupy a čtyřmi výstupy v symbolice obvyklé pro obvody PLD. Každé ze čtyřech čtyřbitových slov můžeme individuálně naprogramovat a proto se obvody PROM používají k realizaci převodníků kódů, generátorů znaků a logických kombinačních obvodů v sekvenčních obvodech. Hlavní nevýhoda těchto obvodů spočívá v tom, že přidáním další vstupní proměnné se vždy zdvojnásobí velikost programovací matice, což vede k omezení počtu vstupních proměnných. Na



Obr.6.4

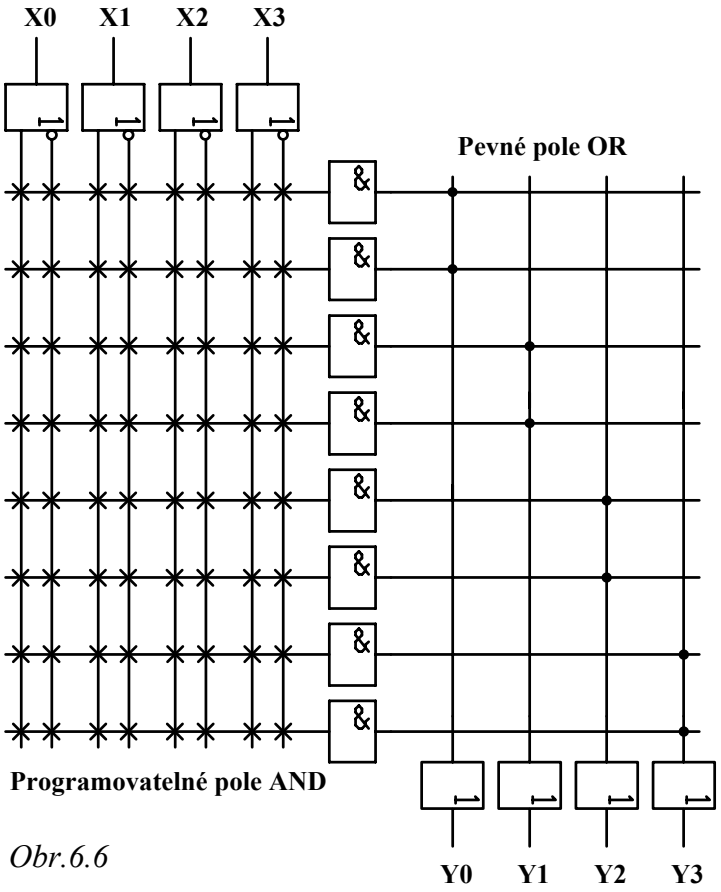
obr.6.5 je zobrazena realizace jednoduchého obvodu o čtyřech logických funkcích v architektuře PROM z pohledu PLD.



Obr.6.5

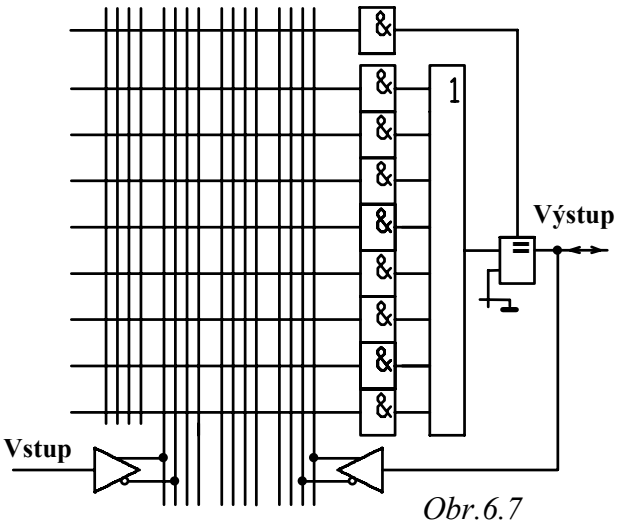
6.2. Obvody PAL

Obvody PAL (Programmable Array Logic) vychází z myšlenky, že úplná součtová forma není nejvhodnější formou k realizaci logických funkcí a lze ji redukovat některou minimalizační metodou. Získanou redukovanou formu (např. v součtovém tvaru) můžeme realizovat ve struk-tuře AND a OR tak, že příslušné proměnné nebo jejich negace přivedeme na vstupy hradel AND a potom je následně sečteme pomocí hradel OR. Tak se dostáváme ke struktuře obvodů PAL obr.6.6, u kterých je programovatelné pole hradel AND a pevné pole hradel OR. K jednomu hradlu OR je připojen pouze omezený počet součinových členů s tím, že jeden součin nelze připojit k několika hradlům OR najednou. Výhodou proti obvodům PROM je malé zvětšení programovatelného pole s přidáním další vstupní proměnné.



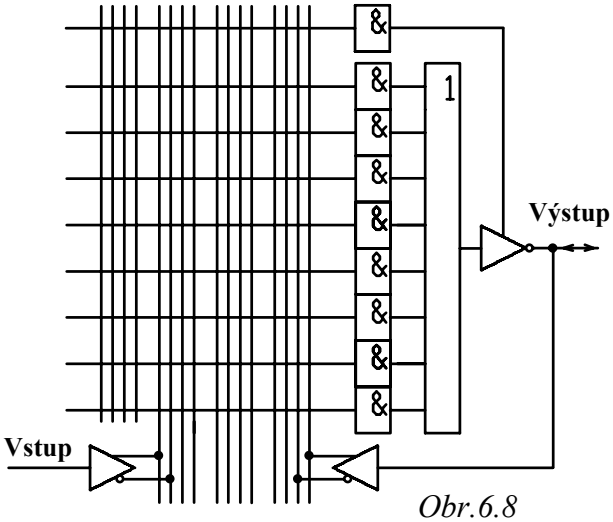
Obr.6.6

Obvody PAL se od sebe odlišují nejen počtem vstupů a výstupů, ale i konfigurací (strukturou) svého výstupu. Výstup obvodu může být pevný, invertovaný nebo s třístavovým budičem. Některé obvody mají na svém výstupu registr nebo klopný obvod. Tyto výstupy jsou pak prostřednictvím zpětné vazby přivedeny zpět do pole AND a umožňují tak realizaci logických sekvenčních obvodů. V historii obvodů PAL existuje řada konfigurací (struktur) výstupního obvodu z nichž o některých nyní pojednáme. Na obr.6.7 je zobrazena struktura výstupního obvodu u programovatelných



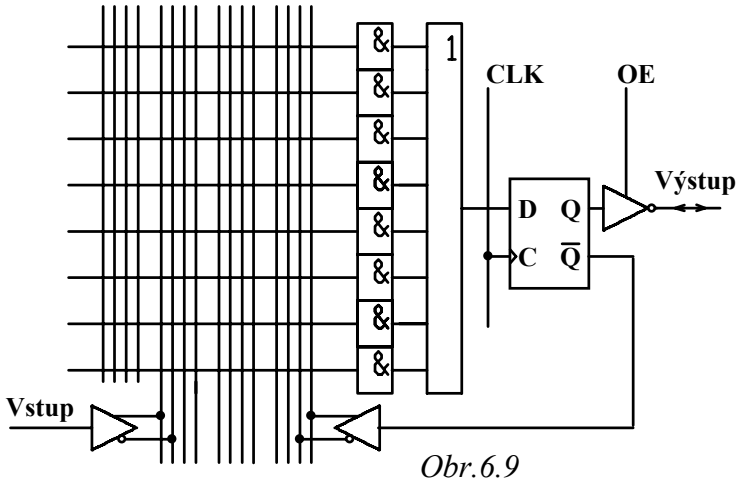
Obr.6.7

polí typu PAL x P y (např. PAL18P8), kde x je maximální počet vstupních proměnných, y je maximální počet výstupních proměnných a x+1 je celkový počet vstupních a výstupních vývodů. Jedná se o obvod s kombinačním výstupem, u něhož lze pomocí obvodu EX-OR naprogramovat polaritu výstupu (propojka na jednom vstupu EX-OR) nebo ho uvést do stavu vysoké impedance v případech, kdy vývod chceme používat jako vstupní.



Obr.6.8

Na obr.6.8 je zobrazena struktura výstupní obvodu základního obvodu PAL x L y (např. PAL16L8, PAL20L8), která se skládá pouze z programovatelného třístavového budiče. Tato struktura umožňuje definovat kombinační výstup jako dvoustavový nebo třístavový, realizovat zpětnou vazbu a používat vývod jako vstupní proměnnou. Zpětná vazba stejně jako v předcházejícím případě umožňuje realizovat asynchronní sekvenční obvody (kapitola 4). Oproti předcházejícímu případu realizuje pouze jednu polaritu výstupu (aktivní úroveň = log.0), odpovídající negované součtové formě.

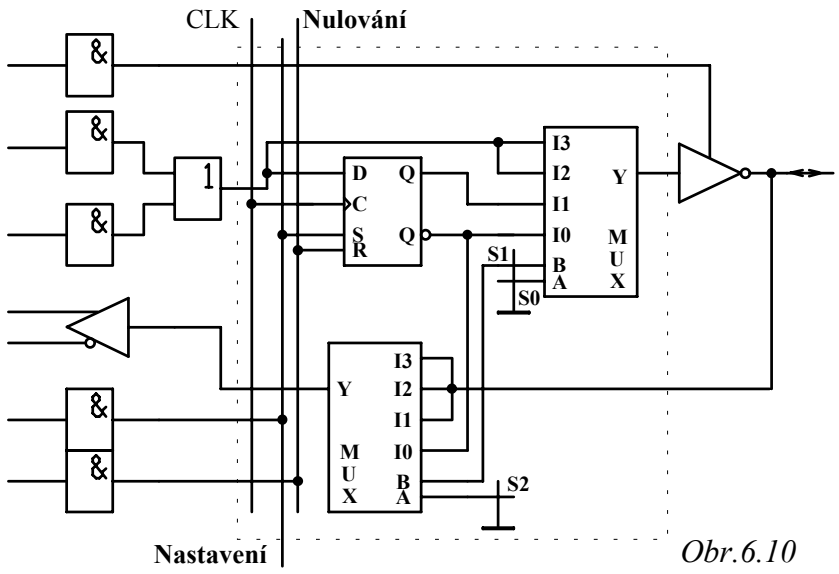


Obr.6.9

Na obr.6.9 je zobrazena struktura klasického výstupního obvodu s registrovým výstupem a zpětnou vazbou k realizaci synchronních sekvenčních obvodů. Obvody s tímto výstupem jsou označovány PAL x R y, kde y určuje maximální počet registrových výstupů. Z čísel x a y však nevyplývá zbývající počet kombinačních výstupů, jestliže je nějakými obvod vybaven. Za výstup hradla OR je zařazen paměťový člen D synchronizovaný hodinovým signálem C, který bývá společný pro všechny paměťové členy a je vyveden jako samostatný vývod, stejně jako aktivační signál pro třístavové budiče, přes které procházejí výstupy z paměťových členů.

Na obr.6.10 je zobrazena struktura výstupního obvodu programovatelného pole PAL22VP10. Jedná se o programovatelný výstupní obvod, který může být pomocí třech programovatelných spojek S2,S1 a S0 naprogramován do 6 různých režimů obsahujících všechny dosud popsané výstupní obvody. Pro stav propojek S0=S1=S2=0 je výstupní obvod konfiguro-

ván na registrový výstup se zpětnou vazbou a výstupem aktivním v log.1. Přepálením propojky S0=1 je možné změnit polaritu výstupu na aktivní v log.0. Pro stav propojek S2=1, S1=0 se bude jednat o registrový výstup se vstupně/výstupní zpětnou vazbou. Propojka S0 opět určuje polaritu výstupu (0 aktivní v log.1, 1 aktivní v log.0). Nakonec stav propojek S2=X a S1=1 konfiguruje výstupní obvod na kombinační s vstupně/ výstupní zpětnou vazbou. Propojka S0 určuje polaritu stejně jako v předcházejících případech. Hodinový signál C je pro všechny paměťové členy ve výstupních obvodech společný a je vyveden jako samostatný vývod z obvodu. Oproti klasickým obvodů PAL jsou paměťové členy vybaveny vstupem pro asynchronní nulování a vstupem pro synchronní nastavení. Tyto vstupy jsou propojeny a přivedeny na výstupy hradel AND. Tak je umožněno definovat jedním součinovým výrazem podmínku pro vynulování a jiným pro nastavení všech paměťových členů najednou. Lze tak jednodušším způsobem realizovat počáteční nulování/nastavení sekvenčního obvodu například na začátku jeho činnosti. K uvedenému obvodu je třeba poznamenat, že je jedním z obvodů, který realizuje větší počet součtů součinů (2x8, 2x10, 2x12, 2x14, 2x16) než klasické obvody PAL (pouze 8x8 součinů).



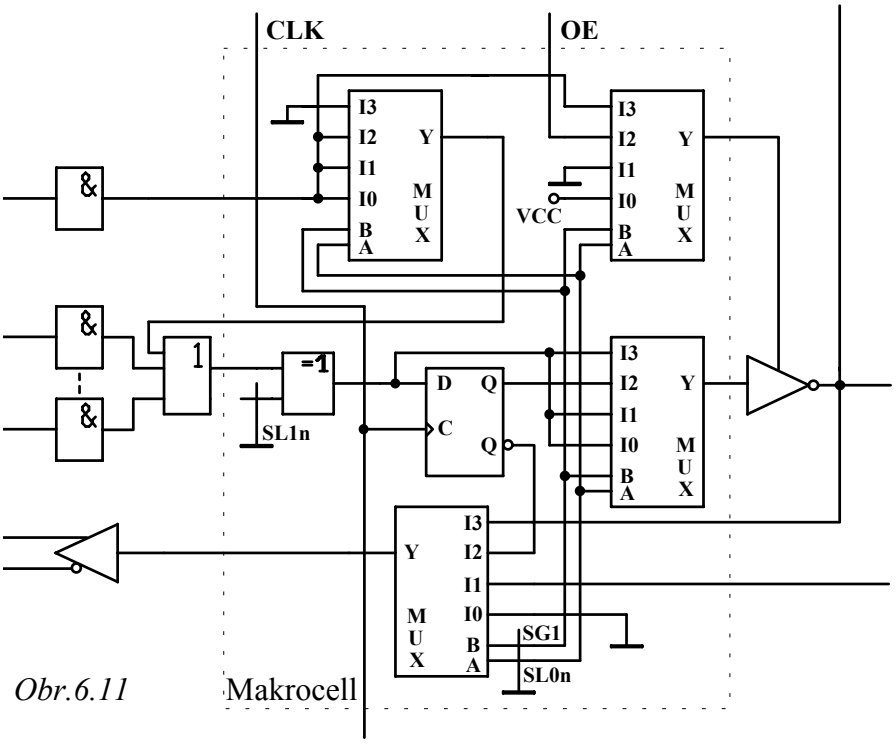
Obr.6.10

Hlavními výhodami obvodů PAL je vcelku jednoduchá architektura, která umožnila zkrátit dobu přenosu signálu přes obvod (typicky vstup/kombinační výstup 7[ns], vstup/ registrový výstup 20[ns]) a možnost programování klasickými programátory obvodů PROM. Nevýhodami obvodů PAL je jejich pevná neměnná architektura, která způsobuje, že průměrně bývá využito pouze 30% integrovaného obvodu a musí být přepáleno cca 70% propojek.

6.3. Obvody GAL

Obvody typu GAL (Genetic Array Logic) patří do skupiny reprogramovatelných obvodů PLD (EEPLD - Electrical Erasable Programmable Logic Device). Ve struktuře obvodů GAL nedochází oproti obvodům PAL k výraznému posunu, jedná se stále o obvod z programovatelným polem AND a pevnou maticí OR. Významná odlišnost spočívá v možnosti elektrického reprogramování a zavedení pojmu makrobuňka na výstupu obvodu. Programovatelná konfigurace výstupní makrobuňky (Output Logic Macrocell) přináší větší flexibilitu obvodů GAL,

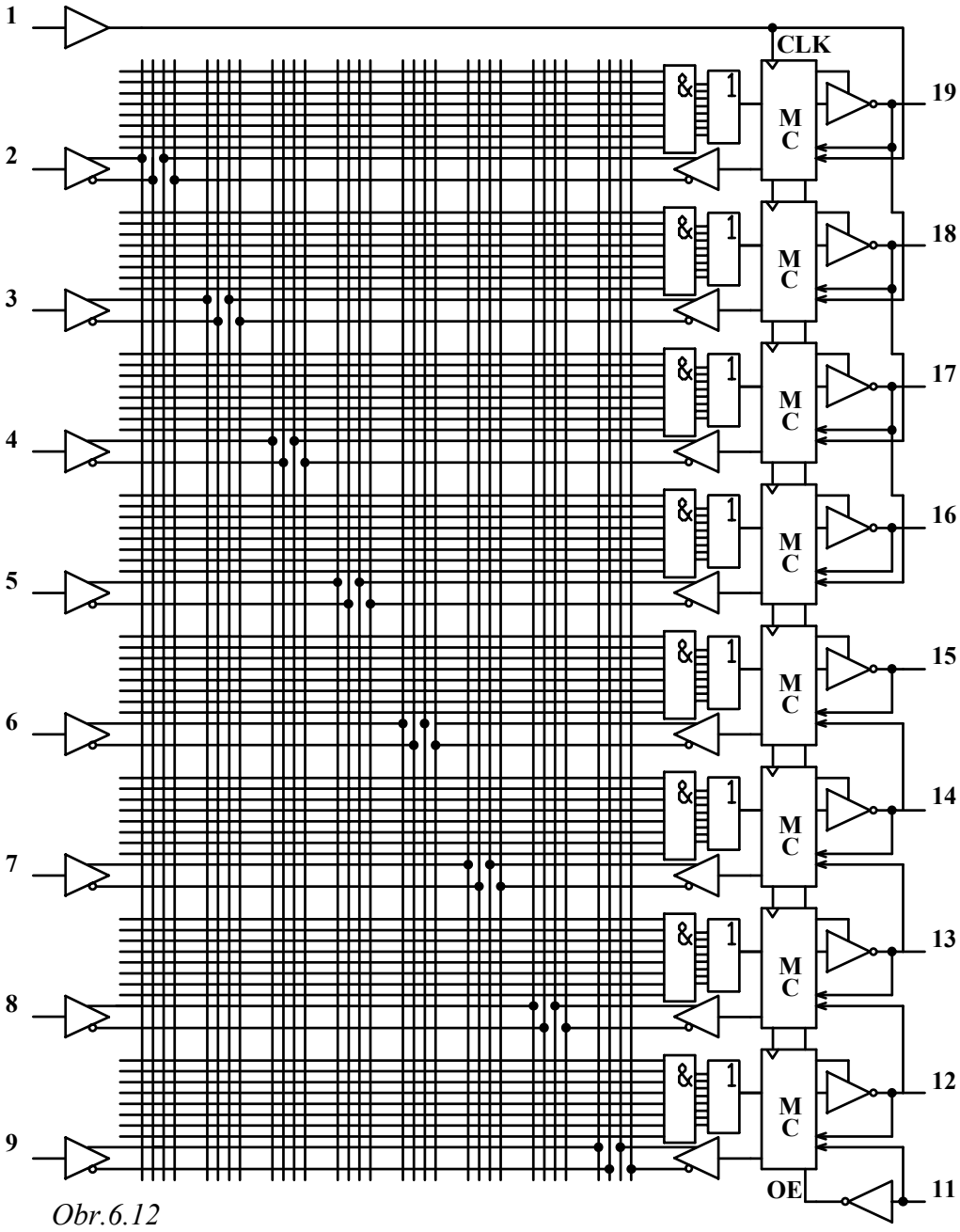
jejím výsledkem je možnost realizovat funkce (emulovat) celé řady klasických obvodů PAL. Připočteme-li možnost elektrického vymazání propojovacího pole s jeho následným reprogramováním a zhruba poloviční až čtvrtinovou spotřebu proti obvodům PAL, jsme u základů jejich úspěchu na trhu. Nejznámějšími obvody z této skupiny jsou GAL16V8, GAL20V8, GAL18V10 a GAL22V10, kde písmeno V označuje konfigurovatelnou makrobuňku a maximální počty vstupních a výstupních vodičů mají stejný význam jako u obvodů PAL. Díky konfigurovatelnosti výstupní makrobuňky může např. GAL16V8 realizovat funkci obvodů PAL 16L8, 16H8, 16R8, 16R6, 16R4, 16P8, 16RP8, 16RP6, 16RP4, 10L8, 12L6, 14L4, 16L2, 10H8, 12H6, 14H4, 16H2, 10P2, 10P8, 12P6, 14P4 a 16P2. Jako určitou nevýhodu můžeme považovat u obvodů GAL16V8 a GAL20V8 to, že aktivační signál třístavových budičů OE a hodinový signál pro jednotlivé paměťové členy makrobuňky je společný. Z dalších obvodů patřících do této řady uveďme GAL18V10 a GAL22V10, které mají stejnou makrobuňku jako již probíraný typ PAL22VP10, a mají tudíž společný hodinový signál pro všechny paměťové členy v makrobuňkách, společný, ale programovatelný vstup asynchronního nulování a synchronního nastavení a individuálně ovládaný třístavový budič na výstupu každého výstupního obvodu..



Obr.6.11

K obvodům GAL je nutné přiřadit nové obvody PALCE, které vyrábí v technologii EECMOS firma AMD. Vlastnosti jednodušších obvodů z této řady (PALCE 16V8, PALCE20V8) jsou v podstatě shodné s odpovídajícími obvody GAL. Na obr.6.12 je zobrazeno funkční schéma obvodu PALCE16V8 a na obr.6.11 je zobrazeno schéma jeho výstupní makrobuňky, kterou lze naprogramovat do sedmi režimů. Naprogramováním hodnoty SL1n lze měnit polaritu výstupního signálu, hodnotami SG1, SG0 a SL0n se nastavuje konfigurace výstupního obvodu. Proměnná $\overline{SG0}$ nahrazuje proměnnou SG1 u zpětno-vazebního multiplexeru makrobuňky 0 a 7 (vývod 12 a 19). Pro konfiguraci SG0=0, SG1=1 a SL0n=0 je makrobuňka konfigurována na registrový výstup s třístavovým budičem ovládaným aktivačním

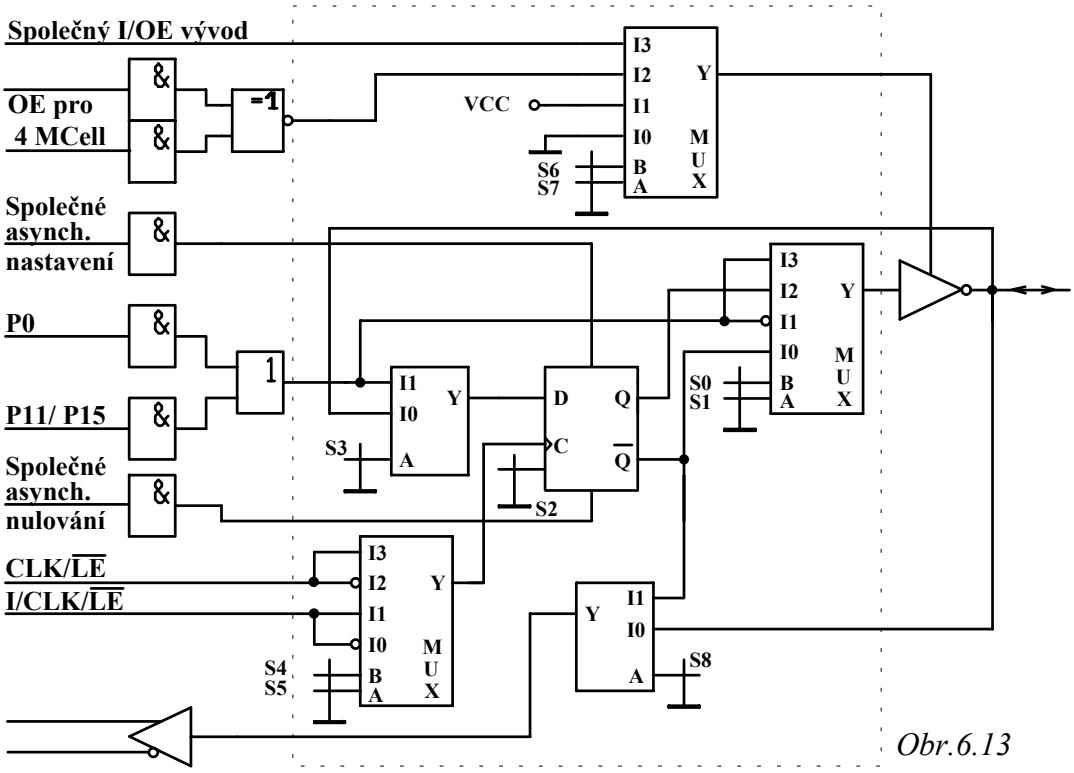
signálem OE (vývod 11). Všech 8 součinů se sčítá v hradle OR pro realizaci budící funkce paměťového členu D. Pro konfiguraci SG1=1 a SL0n=1 je makrobuňka konfigurována jako kombinační výstup s třístavovým budičem řízeným jedním součinovým členem. Zbývajících 7 se sčítá v hradle OR a vytváří hodnotu výstupu. V tomto režimu by mělo být SG0=1, aby jinak nevyužívané vstupy CLK a OE se daly používat jako běžné vstupy. Pro kombinaci SG1=0, SL0n=0 je výstupní buňka konfigurována jako dvoustavový kombinační výstup a pro kombinaci SG1=0 a SL0n=1 je třístavový budič deaktivován a příslušný vývod lze využívat jako další vstup.



Obr.6.12

Obvody jsou vybaveny bezpečnostním bitem (viz. EPLD) a elektronickým podpisem tvořeným 64 bitovým slovem v programovací paměti obvodu specifikovaným výrobcem.

Podpis je čitelný nezávisle na hodnotě bezpečnostního bitu. Po připojení k napájecímu napětí se všechny registry obvodu se automaticky vynulují. Výraznější změnu lze zaznamenat u obvodu PALCE 29M16, který je vybaven universální makrobuňkou firmy AMD zajišťující vysokou přizpůsobivost obvodu. Obvod je vybaven 16 makrobuňkami a propojovacím polem AND, které umožňuje 188 součinů o 29 proměnných nebo jejich negací. Osm makrobuněk s jednoduchou zpětnou vazbou obr.6.13 je vybaveno 9 spínacími elektricky přeprogramovatelnými (EE) buňkami, zbývajících 8 makrobuněk je vybaveno 8 EE buňkami a dvojitou zpětnou vazbou, která vznikne vynecháním zpětnovazebního multiplexeru. Buňka S0 umožňuje měnit výstupní polaritu (1= aktivní log.0, 0= aktivní log.1). Spojka S1 určuje zda výstup obvodu bude kombinační nebo registrový a spojka S2 určuje zda paměťový člen je řízen hranově nebo úrovňově (registr/ latch). Spínač S3 umožňuje použít makrobuňku k registrování vstupní proměnné nebo výstupu z pole AND-OR. Buňky S4 a S5 zajišťují výběr jednoho ze čtyřech synchronizačních signálů paměťových členů (aktivní náběžná nebo sestupná hrana, aktivní úroveň log.0 nebo log.1). Spínače S6 a S7 ovládají řízení třístavového budiče a umožňují trvalou aktivaci budiče, trvalé přivedení do stavu vysoké impedance a aktivní řízení buď z pole AND-OR nebo vývodu integrovaného obvodu. Nakonec spínač S8 u



Obr.6.13

makrobuňek se jednoduchou zpětnou vazbou určuje výběr zpětnovazebního signálu. Z popisu vyplývá, že každá makrobuňka z obr.6.13 může být individuálně naprogramována do jednoho z následujících stavů:

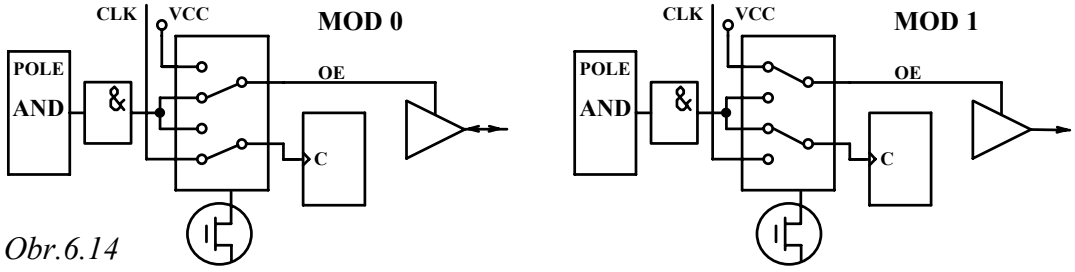
- kombinační výstup s programovatelnou polaritou

- registrový výstup s programovatelnou polaritou
- kombinační výstup s programovatelnou polaritou se vstupně/výstupní zpětnou vazbou
- registrový výstup s programovatelnou polaritou se vstupně/výstupní zpětnou vazbou
- kombinační výstup s registrovou zpětnou vazbou a aktívní úrovní log.0 na výstupu
- registrový výstup s registrovou zpětnou vazbou a aktívní úrovní log.0 na výstupu
- registrový vstup do pole AND-OR.

Globálně lze řídit aktivaci třístavových budičů vstupním vodičem nebo vytvořeným výrazem stejně jako hodinový signál pro paměťové členy. Každý paměťový člen lze individuálně asynchronně nulovat i nastavit vytvořeným výrazem. K rozsáhlým možnostem použití v aplikacích přispívá i rozšíření součtu součinů u čtyřech makrobuněk na 12 a dalších čtyřech makrobuněk na 16. Po přivedení napájecího napětí je uskutečněn automatický RESET (vynulování) obvodu.

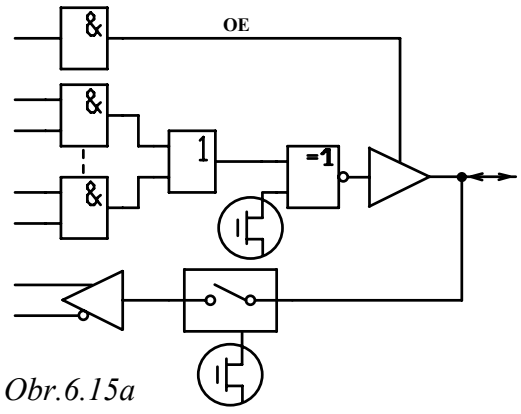
6.4. Obvody EPLD

Obvody EPLD (Erasable Programmable Logic Device) jsou obvody s programovatelnou AND a OR strukturou, kterou je možné vymazat ultrafialovými paprsky a přeprogramovat podobně jako je tomu u pamětí typu EPROM. Technologie EPROM umožnila dosáhnout oproti obvodům PAL v bipolární technologii s přepálitelnými spojkami vyšší hustoty integrace na čipu a tak bylo možné do obvodů integrovat nové funkce. Základní struktura však zůstává principiálně stejná. Obvody se vyznačují střední dobou zpoždění t_{pd} signálu pohybující se od



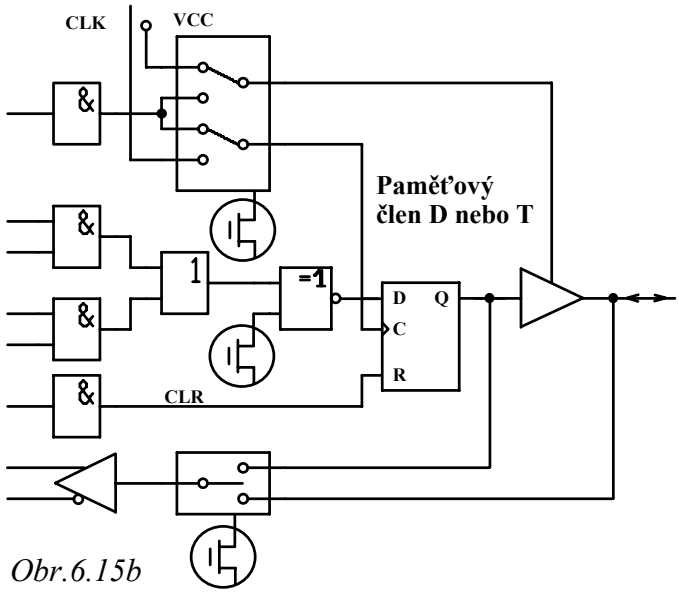
Obr.6.14

20 do 40 ns a odběrem ze zdroje cca 45mA. Hlavními představiteli této skupiny obvodů jsou EP330, EP610, EP630, EP910 a EP1810. K výraznější změně architektury oproti dosud popsaným obvodům dochází u obvodů EP610, EP630 a EP910, které se od sebe liší počtem vstupních vodičů a makrobuněk. Obvod EP610 je vybaven čtyřmi vstupy a 16 I/O (vstupně/ výstupními) vodiči a dvěma hodinovými vstupy společnými vždy pro osm makrobuněk. Do každé makrobuňky vstupuje 10 součinů obvodu AND se 40 vstupy (20 proměnných). Z těchto součinů se 8



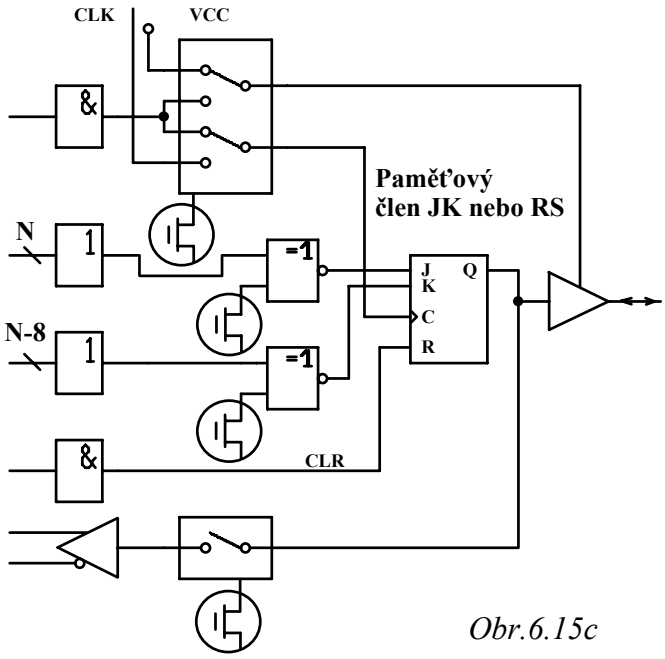
Obr.6.15a

využívá k vytvoření výstupní funkce nebo budících funkcí paměťových členů. Jeden součin se využívá k nulování registru makrobuňky a zbývající se využívá k realizaci aktivačního signálu OE nebo hodinového signálu pro paměťové členy. Použitá makrobuňka může být konfigurována na kombinační nebo registrový výstup s programovatelnou polaritou. Registr v každé makrobuňce může být konfigurován jako paměťový člen D, JK, T a nebo RS. Zpětná vazba může být programována jako registrová nebo bez registru.



Obr.6.15b

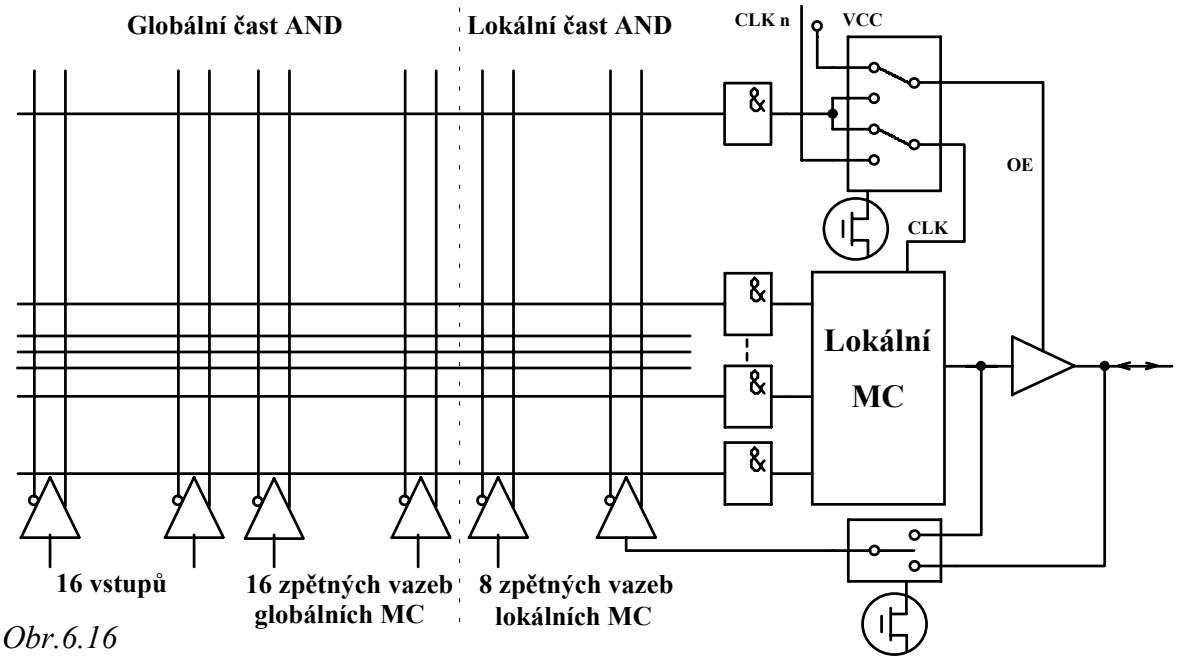
Na obr.6.14 jsou zobrazeny dva módy multiplexeru OE/CLK, který lze naprogramovat individuálně pro každou makrobuňku. V módu 0 (po vymazání pole) je výstup makrobuňky třístavový kontrolovaný jedním součinem z pole AND. Klopné obvody s tímto módem jsou synchronizovány hodinovým signálem vyvedeným pro příslušných 8 makrobuňek z IO. V módu 1 je třístavový budič aktivován trvale (dvoustavový výstup) a hodinový signál je vytvářen součinem v matici AND a umožňuje individuální řízení jednotlivých klopných obvodů v makrobuňce. Každý klopný obvod může být nastaven na aktivní náběžnou nebo sestupnou hranu hodinového signálu. Na obr. 6.15 a,b,c jsou zobrazeny základní konfigurace EP610, do kterých může být individuálně každá makrobuňka konfigurována. Při použití paměťového členu D a T se 8 součinů sčítá v hradle OR a vytváří budící funkci. Pro členy JK a RS se 8 součinů rozdělí na vytvoření obou budících funkcí pro vstupy JK a RS.



Obr.6.15c

K výraznější změně architektury dochází u obvodu EP1810, u kterého je zavedena globální a lokální makrobuňka a matice AND. Obvod je rozdělen do čtyřech ekvivalentních kvadrantů, z nichž každý obsahuje 8 lokálních a 4 globální makrobuňky, u kterých nedochází

ke změně architektury vůči předcházejícím typům. Na obr.3.33 je zobrazeno připojení lokální makrobuňky k lokální matici AND. Zapojení globální makrobuňky se od lokální liší v



Obr.6.16

neexistenci zpětnovazebního multiplexeru a rozšířeném multiplexeru OE umožňujícího uvedení výstupního budiče do stavu vysoké impedance (vývod bude používán jako vstup). Výstup třístavového budiče je přiveden do globální části matice AND (zpětné vazby globálních MC) a výstup z makrobuňky je přiveden do lokální části AND (zpětné vazby lokálních MC). Zavedení globální a lokální matice, které snižuje rozsah matice AND, vychází z předpokladu, že uživatel nepotřebuje zpětné vazby mezi všemi (v tomto případě 48) makrobuňkami, ale jenom mezi skupinami některých makrobuněk. Tyto makrobuňky budou soustředěny do příslušného kvadrantu a propojeny lokální maticí AND. Nezbytná vzájemná spojení a přístup společných vstupních proměnných zajišťuje globální matice AND. Rozsáhlost obvodu je již tak velká, že klasický návrh vycházející z rovnic a stavových diagramů není nejvýhodnějším řešením. Z těchto důvodů již výrobci obvodů vytvářejí knihovny makrofunkcí - standardních obvodů TTL jako jsou čítače, komparátory, multiplexery, dekodéry, posuvné registry atd. Zpracování podkladů pro naprogramování obvodu potom vytváří návrhové systémy zpracovávající schéma navrženého obvodu v některém známém programu jako je Orcad, Viewlogic, atd.

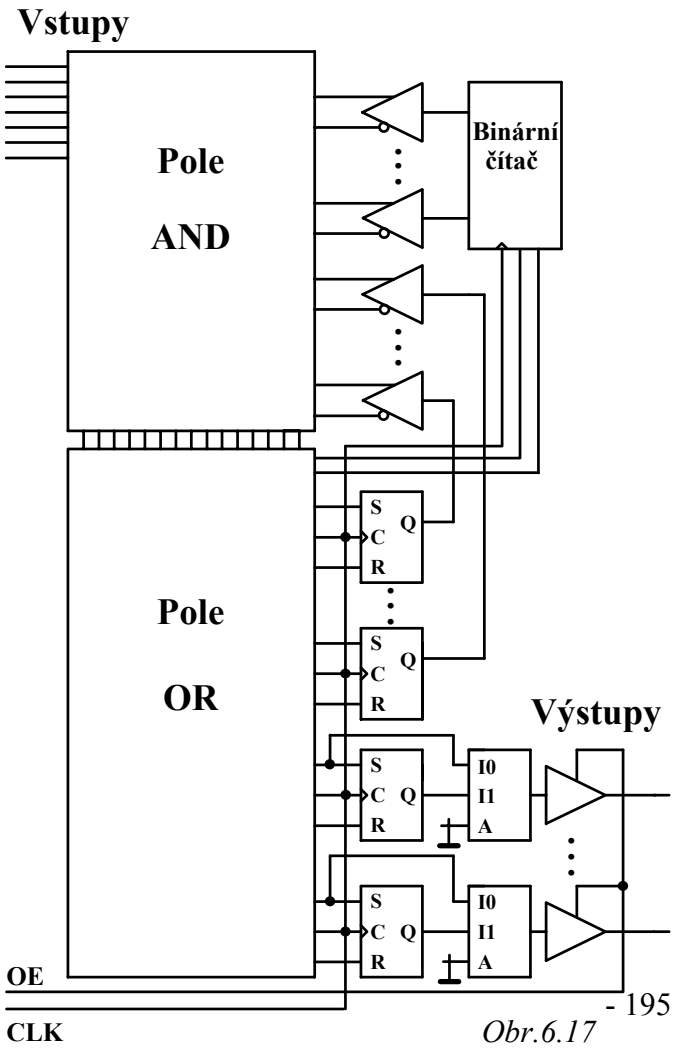
Novým prvkem, který byl u těchto obvodů použit a rozšířil se do dalších obvodů, je **bezpečnostní bit** po jehož naprogramování již nelze obsah obvodu (bitovou mapu) přečíst a je tak zabezpečen proti snadnému kopírování. Chování a případně i obsah uložený v obvodu je možné analyzovat pouze na základě stavu vstupních a výstupních proměnných, jedná-li se o obvod sekvenční tak i na jejich předcházejících kombinacích (kapitola 4).

6.5. Obvody PLA

Obvody PLA (Programmable Logic Array) jsou obvody, které mají programovatelnou matici AND i matici OR. Typickými představiteli jsou obvody TIFPLA839 a TIFPLA840, které mají matici AND 14x32 (14 proměnných x 32 součinů) a matici OR 32x6 (32 součinů x 6 výstupů) s programovatelnou polaritou výstupů. I když struktura umožňuje realizovat funkce s větším počtem součinů než 8, obvody PLA se již nedoporučují pro realizace nových zařízení.

6.6. Obvody PLS a PSG

Obvody PLS (Programmable Logic Sequencer) a PSG (Programmable Sequencer Generator), které mají programovatelnou matici AND i OR, mají skoro stejnou vnitřní strukturu obr.6.17. Rozdíl je v tom, že u obvodů PSG je ve struktuře navíc integrován binární čítač. Z většiny paměťových členů je výstup přiveden zpět do matice AND, kde se kombinují se vstupními proměnnými. Výstupy obvodu jsou tvořeny multiplexery, kterými může být konfigurován výstup jako kombinační nebo registrový. Na výstupech jsou umístěny třístavové budiče ovládané společným signálem OE, který je vyveden z IO. Všechny paměťové registry mají společný hodinový signál, který je přes obvod EX-OR (není v obrázku nakreslen) vyveden z IO. Tím je umožněno změnit výkonnou hranu hodinového signálu z náběžné na sestupnou.



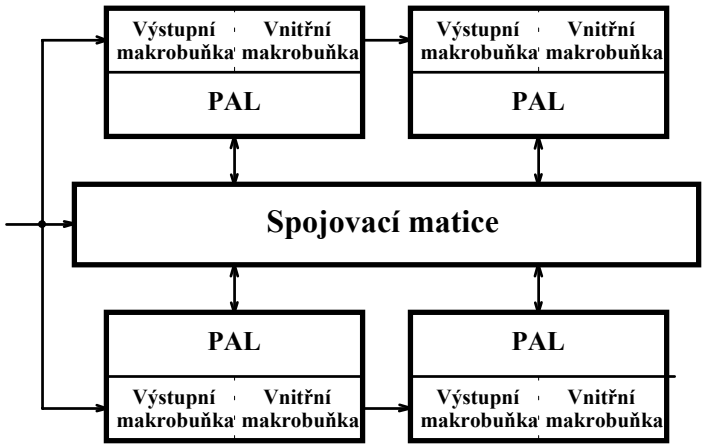
Obr.6.17

Typickými představiteli obvodů PLS jsou PLS506 (13x97x8/16), 82S105 (16x48x8/6) a 82S167 (14x48x6/6), kde čísla v závorkách v pořadí jak jdou za sebou znamenají počet vstupních proměnných, počet součinů z matice AND, počet výstupů lomeno počtem vnitřních skrytých registrů. Představitelem skupiny PSG, který i přes svoji velkou přizpůsobivost aplikaci bývá návrháři opomíjen, je obvod PSG507 (13x80x8/8). Obvod je oproti obvodům PLS vybaven 6 bitovým binárním čítačem synchronně řízeným vstupem nulování a čítání/stání (count/hold). Vzhledem k zabudovanému čítači umožňuje obvod snadno

realizovat kompletní časování řídících obvodů a zjednodušuje logické výrazy odvozené ze stavového diagramu. Použití obvodu se soustřeďuje na realizace generátorů slov, řídících obvodů, děličů kmitočtu, časovačů, atd.

6.7. Obvody MACH

Obvody MACH (Macro Array CMOS High-density) od firmy AMD představují novou cestu v realizaci návrhu rozsáhlých logických systémů. Obvody MACH jsou vyráběny technologií EECMOS a dovolují v současnosti ekvivalentní hustoty 900 a až 3600 hradel na pouzdro. Obvod MACH se skládá z několika bloků PAL vzájemně propojených spojovací maticí obr.6.18. Architektura MACH opět vychází z předpokladu, že složitý logický obvod lze vytvořit spojením několika funkčních modulů. Jednotlivé moduly budou realizovány bloky PAL a jejich vzájemné propojení zajistí spojovací matice. Obvody bez modulární struktury mohou být též realizovány obvody MACH pokud spojovací matice bude schopna zajistit požadovaná spojení mezi jednotlivými bloky PAL. Propojovací pole však nabízí dostatečnou míru flexibility i pro složité návrhy. Uspořádání obvodu bez modulární struktury pak zajistí návrhový systém a návrhář se nemusí věnovat detailní logické implementaci.



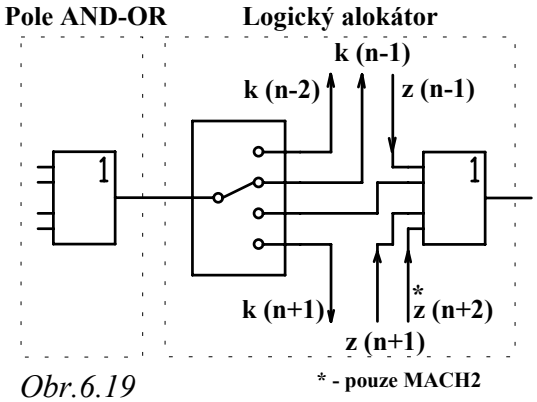
Obr.6.18

Obvod	Počet vývodů	Počet makrobuněk	Počet ekviv. hradel	Počet vstupů	Počet výstupů	Počet klopných obvodů
Řada obvodů MACH 1						
MACH110	44	32	900	38	32	32
MACH120	68	48	1200	56	48	48
MACH130	84	64	1800	70	64	64
Řada obvodů MACH 2						
MACH210	44	65	1800	38	32	64
MACH220	68	96	2400	56	48	96
MACH230	84	128	3600	70	64	128

Asynchronní obvod MACH						
MACH215	44	64	1500	38	32	64

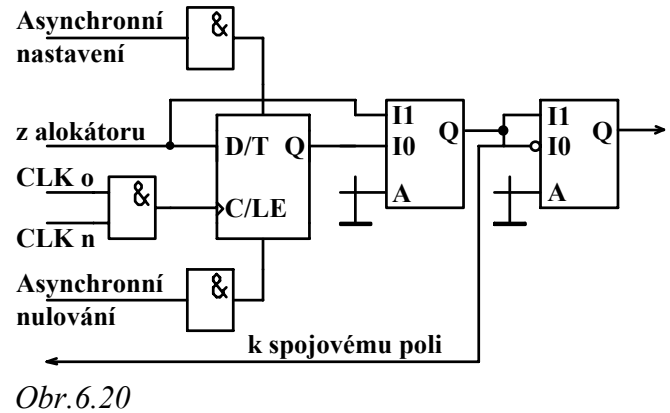
Tabulka 6.1

Obvody MACH se skládají ze synchronních obvodů MACH 1 a MACH 2 a asynchronního obvodu MACH215. Obvody z řady MACH 1 a 2 jsou ideální pro realizaci synchronních logických subsystémů jako jsou paměťové řadiče a řadiče periferií výpočetních systémů. V tabulce 6.1 jsou uvedeny některé technické parametry zástupců obvodů MACH, které se od sebe liší počtem vývodů, makrobuněk, množstvím spojení a počtem hodinových signálů. Řada obvodů MACH1 obsahuje pouze výstupní makrobuňky, řada MACH2 má výstupní i vnitřní makrobuňky.

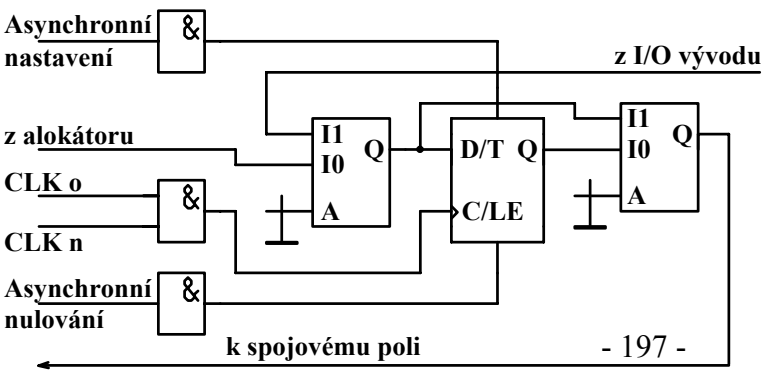


Obr.6.19

Jak vyplývá z obr.6.18 zajišťuje spojovací matice komunikaci mezi bloky PAL a rozvádí po obvodu vstupní proměnné. Většina vývodů obvodu je typu I/O a mohou být naprogramovány pro vstupní, výstupní nebo obousměrný přenos. Existuje však několik vodičů, které jsou pouze vstupní. Z každé makrobuňky bloku PAL je vyvedena vnitřní zpětná vazba do spojovacího pole a to i v případě, že se vrací do stejného bloku PAL. Tím je zajištěna vzájemná komunikace bloků PAL, velká přizpůsobivost architektury, ale zároveň se tak zajišťují i stejná zpoždění mezi jednotlivými makrobuňkami nezávisle na spojovací cestě. Každý blok PAL obsahuje programovatelné pole pro generování součinových výrazů a umožňuje vlastní realizaci Booleovských rovnic. Vstupy do pole AND přicházejí v přímém nebo invertovaném tvaru z propojovacího pole a jejich počet je 22 (MACH110, 210) nebo 26 (MACH 120, 220, 130, 230). Ačkoliv matice AND-OR realizuje součet čtyř součinů, nemusí být počet sčítaných součinů konstantní (pevný). Výsledný počet sčítaných součinů je ovlivňován obvodem mimo matici AND-OR tzv. logickým



Obr.6.20



Obr.6.21

alokátorem. Alokátor zajistí připojení jednoho ze čtyřech součtových výrazů na vstup makrobuňky obr.6.19. Jeho programování provádí automaticky na základě realizovaných logických rovnic vývojový systém. Počet součinů umožňuje dostatečnou přizpůsobivost při návrhu, protože je možné realizovat složité funkce popsané velkým počtem součinů tak, že nevyužité součiny od jiných makrobuněk využijeme k rozšíření součinové formy.

V obvodech MACH jsou použity makrobuňky výstupní obr.6.20 a interní (vnitřní) obr.6.21, které jsou součástí pouze obvodů MACH2 a zdvojnásobují celkový počet makrobuněk aniž by byl zvyšován počet vývodů integrovaného obvodu. Obě buňky lze konfigurovat jako kombinační nebo registrové s programovatelnou polaritou, u MACH2 lze využít úrovňové řízení paměťového členu (latch). Vlastní paměťový člen může být konfigurován jako člen T nebo D. Vnitřní makrobuňky nemají výstup z integrovaného obvodu a jsou vybaveny jen vnitřní zpětnou vazbu. Jednou z jejich výhod je to, že umožňují realizovat záchytný registr pro vstupní proměnnou s tím, že součinový výraz pro budící funkci paměťového členu zůstává nevyužitý a může být proto použit k rozšíření jiných výrazů. Vlastní výstup z makrobuňky prochází přes třístavový budič, který může být trvale aktivován, uveden do stavu vysoké impedance nebo aktivně řízen jedním ze dvou součinových výrazů. Každý blok PAL obsahuje součinové výrazy umožňující společné asynchronní nulování nebo přednastavení do libovolného stavu. Obvod je vybaven bezpečnostním bitem a po připojení k napájecímu napětí je automaticky vynulován.

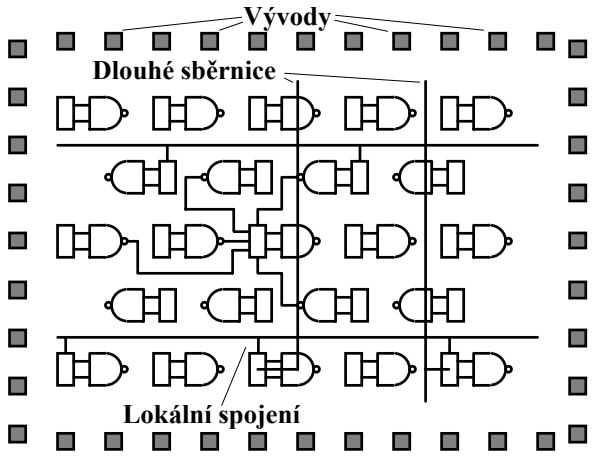
Architektura druhé skupiny programovatelných obvodů vychází z myšlenky realizovat požadovanou logickou funkci pomocí základních logických bloků, které budou mezi sebou vhodně propojeny. Tyto obvody se svoji strukturou daleko více blíží k zákaznickým obvodům a přitom kombinují výhody PLD obvodů, jako je jednoduchý návrh, vysoká přizpůsobivost, krátká doba realizace, atd. s výhodami vysoké integrace zákaznických VLSI obvodů. Obvody mohou být produkovány ve velkých sériích a teprve naprogramováním u uživatele lze dosáhnou konkrétní aplikace. Ekvivalentní hustota obvodů se v současnosti pohybuje v rozmezí 1000 až 20000 hradel a proto lze jedním obvodem realizovat i složité logické funkce, které dříve bylo nutné realizovat několika obvody SSI/MSI, PLD nebo EPLD. Návrhář při tom není vystavován rizikům spojeným s realizací pomocí zákaznických obvodů. Obvody z této skupiny se skládají ze tří stavebních prvků, kterými jsou základní logické bloky, programovatelné vstupně/výstupní bloky a programovatelné spojovací cesty. Základní bloky mohou být pevné nebo programovatelné. Pomocí programovatelných spojovacích tras mohou být jednotlivé bloky mezi sebou vzájemně propojovány a mohou tak být vytvářeny poměrně složité funkce. Funkce vstupně/výstupních bloků spočívá v zajištění přenosu proměnných mezi jednotlivými vývody a obvodem samotným.

Oproti obvodům založeným na makrobuňkách nemají tyto obvody principiální omezení v realizaci velkého logického systému, v počtu logických stupňů nebo součinových členů. Obvody mohou být aditivně zvětšovány a s nimi i složitost realizovaných funkcí. Oproti rela-

tivně pevné struktuře obvodů s makrobuňkami mohou být základní logické bloky konfigurovány na různé logické či paměťové funkce. K výrazné odlišnosti dochází u programovatelných spínačů, které jsou tvořeny tzv. antispojkami nebo statickými pamětmi RAM. Použití statické paměti ve funkci spínačů přináší možnost neomezeného reprogramování, testování a experimentování přímo na definitivní realizaci. Takováto konfigurace umožňuje reprogramování obvodu za provozu a jeho přizpůsobení změněným podmínkám.

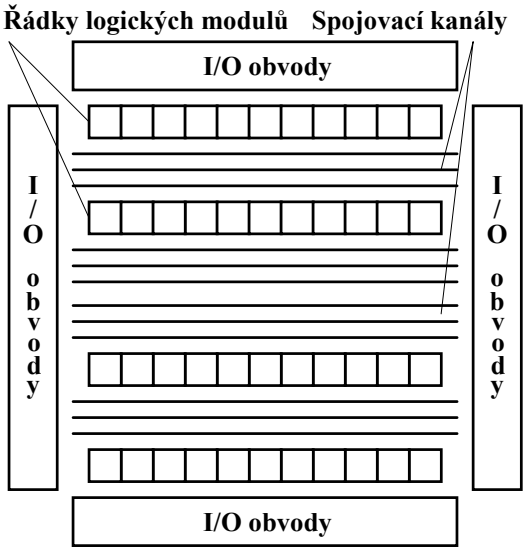
6.8. Obvody ERA

Obvody ERA jsou programovatelné obvody vycházející z myšlenky realizace logického obvodu pomocí základních logických členů (hradel). Na obr.6.22 je zobrazena základní struktura obvodu ERA tvořeného polem hradel, mezi kterými jsou umístěny lokální propojovací vodiče a tzv. dlouhé sběrnice. Po obvodu pole hradel jsou umístěny vstupy a výstupy. Obvody se vyrábějí s velkým počtem hradel na pouzdro, ale jejich využitelnost pro danou aplikaci je nízká a pohybuje se okolo 30 až 40% stejně jako u obvodů PAL. V architektuře je obtížné rozmístění hradel pro navrhovaný obvod a hlavně jejich propojení, které je velmi omezené a způsobuje tak nízkou využitelnost obvodu. U dalších zástupců této skupiny obvodů byly některé nevýhody potlačeny a obvody byly doplněny o nové prvky, které jim zajistily daleko větší úspěch.



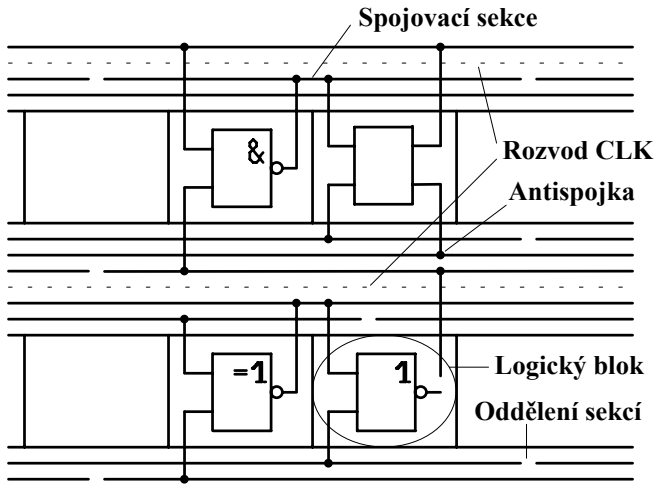
Obr.6.22

6.9. Obvody FPGA



Obr.6.23

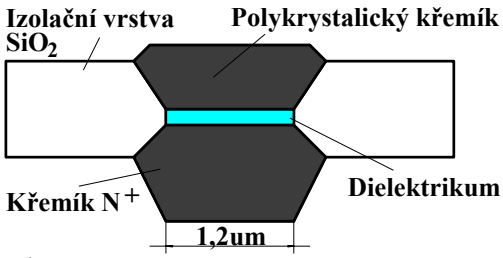
Obvody FPGA (Field Programmable Gate Array) jsou dalším zástupcem obvodů založených na principu propojování logických bloků. Tyto obvody se svými několika charakteristickými vlastnostmi, jako jsou antispojky, logické moduly, rozvedení hodinového signálu, spojovací kanály a diagnostické obvody, odlišují od ostatních představitelů programovatelných obvodů. Ve srovnání s klasickým hradlovým polem je sice obvod FPGA dražší, ale zato poskytují uživateli ekvivalentní hustotu, která je 10 až 100-krát větší než umožňují obvody PAL.



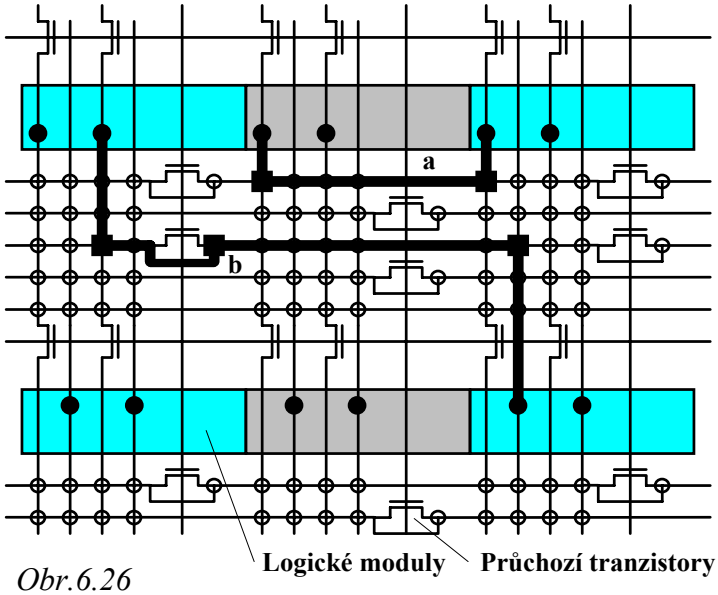
Obr.6.24

Na obr.6.23 je zobrazena celková a na obr.6.24 detailní architektura obvodů FPGA firmy Texas Instruments, které se skládají z kanálového uspořádání logických modulů (kombinačních nebo sekvenčních) jako stavebních bloků, mezi kterými jsou uloženy propojovací kanály zajišťující spojení mezi jednotlivými bloky. Po obvodu jsou potom umístěny vstupně/výstupní obvody, které zajišťují styk s okolím.

Spínacím prvkem je u těchto obvodů antispojka (antifuse) obr.6.25. Antispojka je normálně rozpojená (přerušená) součástka, která se stává vodivou po napěťovém pulsu (21V) na ni přivedenou po dobu 5ms. Antispojka se skládá ze dvou vodivých vrstev od sebe oddělených tenkou dielektrickou vrstvou, která v nepropustném stavu má odpor větší jak 100MΩ. Po naprogramování (proražení) je její odpor přibližně 500Ω. Mezi hlavní výhody antispojek patří malé rozměry, které jsou 20x menší než u spínače tvořeného statickou pamětí RAM a 10x menší než u spínače typu EPROM. Protože pracuje na opačném principu než přepalitelné spojky v obvodech PAL, kterých muselo být přepáleno 70 až 80%, je při programování programováno pouze 20 až 30% všech spínačů. Ke kanálovým spojením, která jsou rozvedena horizontálně jsou vertikálně připojeny vstupy a výstupy jednotlivých logických modulů obr. 6.24. Počet spojení na spojovací kanál závisí na počtu spojovacích cest a na jejich segmentaci, která odpovídá počtu logických modulů i realizaci jednotlivých spojení mezi moduly. Na obr.6.26 jsou zobrazena propojení mezi dvěma bloky. Propojení označené **a** využívá jeden segment a dvě antispojky, propojení označené **b** využívá dva segmenty a tři antispojky. Propojení obou segmentů zajistíme naprogramováním antispojky, která zajistí zkratování průchozího tranzistoru. Horizontální a vertikální průchozí tranzistory napomáhají při realizaci cesty k naprogramování kterékoliv antispojky. Spolu s dekódujícími obvody založenými na posuvných registrech umožňují adresovat kterýkoliv vnitřní uzel programovatelného obvodu. Pro synchronní řízení obvodu je v každém spojovacím kanálu rozveden posílený hodinový signál. Vstupně/ výstupní obvody mohou být naprogramovány jako vstupní, výstupní, třístavové nebo obousměrné. Hlavní výhoda architektury obvodů FPGA spočívá ve vysoké využitelnosti 85 až 90% logických bloků na čipu, rychlém a plně automatickém rozložení a propojení logických celků návrhovým systémem pro navržené zapojení logického obvodu.



Obr.6.25

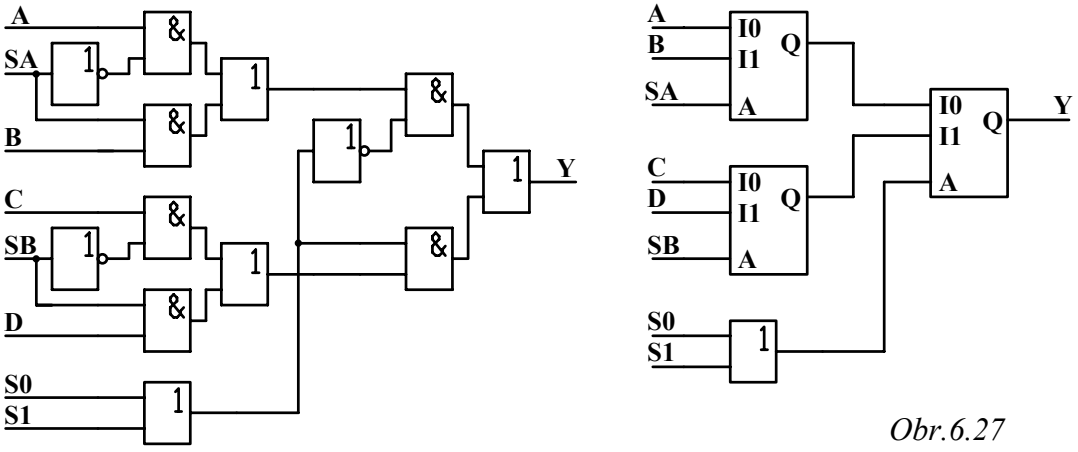


Obr.6.26

Obvody FPGA od firmy Texas Instruments se vyrábějí s logickými bloky tvořenými pouze kombinačním obvodem řada TPC10 (TPC1010A,TPC1020A) a kombinačním i sekvenčním obvodem řada TPC12 (TPC1225, TPC1240, TPC1280), jejichž základní vlastnosti jsou shrnuty v tabulce 3.33. Na obr.6.27 je zobrazena vnitřní struktura logického bloku řady TPC10, která se skládá ze tří multiplexerů jeden ze dvou, které při přepínání nevykazují

Obvod	Počet ekviv. hradel	Počet logických modulů	Počet klopných obvodů	Počet antispojek	Počet cest na kanál	Pouzdro
TPC1010A	1200	295	147	112k	22	68PLCC
TPC1020A	2000	547	273	186k	22	84PLCC
TPC1225	2500	451	341	250k	36	84PLCC
TPC1240	4000	684	565	400k	36	132CPGA
TPC1280	8000	1232	998	750k	36	176CPGA

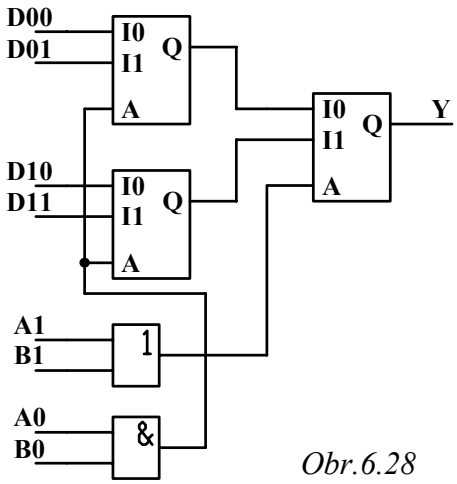
Tabulka 6.2



Obr.6.27

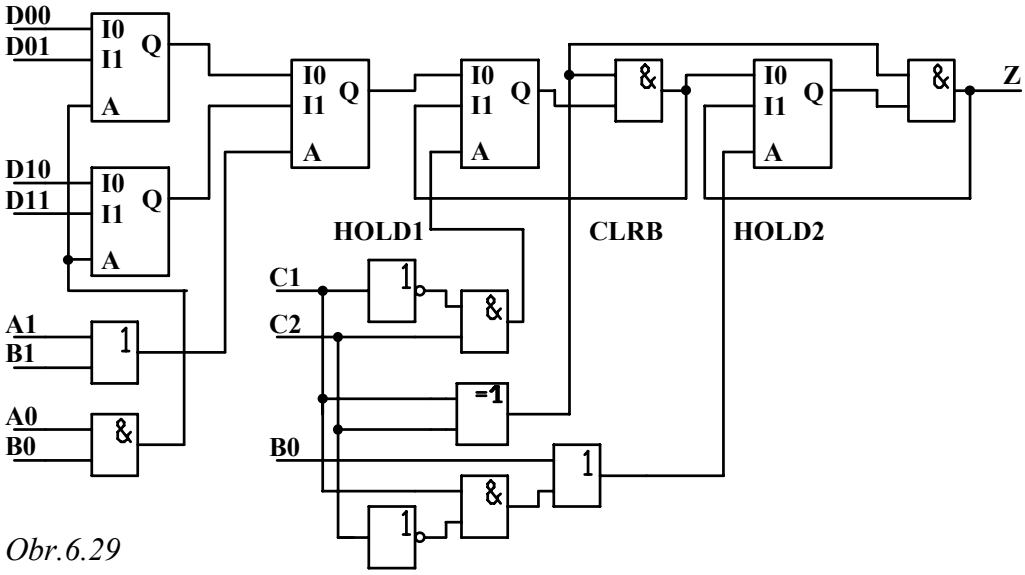
statické hazardy, a jednoho dvouvstupého obvodu OR. Snadno odvodíme, že obvod realizuje logickou funkci danou tímto výrazem

$$Y = (A.SA + B.\overline{SA}).(S0 + S1) + (C.SB + D.\overline{SB}).S0.S1, \qquad (6.1)$$



Obr.6.28

který dává široké možnosti k realizaci různých logických funkcí. Použitý multiplexer, máme-li k dispozici i negace vstupních proměnných, umožňuje realizovat libovolnou funkci dvou proměnných. Připojením některých vstupů logického bloku na napájení nebo zem, můžeme provádět programování jeho funkcí (vytvářet tzv. pevná makra). Například uzemníme-li jeden ze vstupů A nebo B multiplexeru snadno zjistíme, že multiplexer realizuje funkci logického součinu $B.\overline{SA}$ nebo A.SA. Funkci logického součtu získáme obdobně, pouze vstup A $(SA + B)$ nebo B $(A + \overline{SA})$ připojujeme na logickou jedničku (napájení). Takovýmto způsobem jsou vytvořeny knihovny základních logických funkcí (tzv. hard macros) NAND, NOR, AND, OR, XOR, MUX, ale i sčítaček nebo paměťových členů typu Lanch, Flip-Flops atd. Kromě pevných knihoven, u kterých je pro zajištění časových zpoždění nutné trvat na přesné konfiguraci použitých logických modulů, existují i knihovny volné (soft marcos). Návrh logického obvodu a jeho realizace s obvodem FPGA začíná nakreslením schématu logického obvodu např. programem Orcad vytvořeného z



Obr.6.29

knihoven obvodů FPGA. Obvody ve schématu jsou pomocí knihoven vývojovým systémem nahrazeny obvody skládající se pouze z logických modulů. Další fáze návrhu spočívá v nalezení propojovacích cest mezi jednotlivými moduly, aby bylo dosaženo obvodu získaného v předcházející fázi. Závěrečnou fází návrhu je získání programovacího předpisu určujícího, které propojky mají být propáleny.

Na obr.6.28 a obr.6.29 je zobrazena vnitřní struktura kombinačního a sekvenčního modulu obvodů řady TPC12, které se v kanále logických modulů po sobě střídají. Kombinační modul je skoro shodný s modulem použitým v řadě TPC10 a realizuje tuto funkci

$$Y = (D00.(\overline{A0} + \overline{B0}) + D01.A0.B0).\overline{(A1 + B1)} + \\ (D10.(\overline{A0} + \overline{B0}) + D11.A0.B0).(A1 + B1) \qquad (6.2)$$

Sekvenční logický modul se skládá z kombinačního modulu, za kterým je zapojen dvoustupňový sekvenční obvod, který v závislosti na hodnotách C1,C0 a B0 umožňuje realizovat tyto funkce:

- kombinační obvod realizovaný předřazeným kombinačním modulem pro C1=1, C0=0
- kombinační obvod následovaný hladinově řízeným obvodem D pro C2=0, B0=0 a C1=CLK
- kombinační obvod následovaný hranově řízeným obvodem D a asynchronním nulováním pro C1=0, C2=CLK a B0=CLR.

Příklad 6.1 Navrhněte zapojení úplné binární sčítačky z logických modulů obvodu TPC1020.

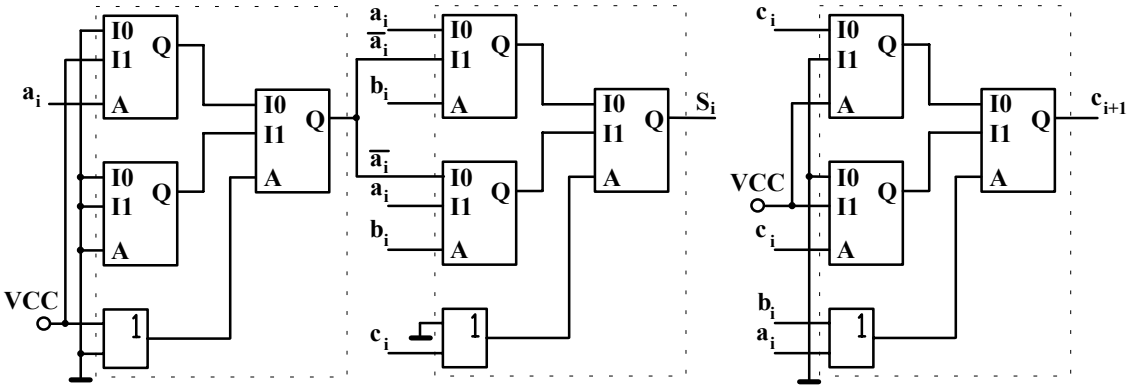
Jak bylo odvozeno v kap.3.3, můžeme pro součet S_i a přenos do dalšího řádu c_{i+1} psát

$$S_i = (a_i.\overline{b_i} + \overline{a_i}.b_i).\overline{c_i} + (\overline{a_i}.\overline{b_i} + a_i.b_i).c_i \qquad (6.3)$$

$$c_{i+1} = a_i.b_i.\overline{c_i} + c_i.(a_i + b_i) \qquad (6.4)$$

Rovnice pro součet S_i je velmi podobná rovnici charakterizující vlastnosti logického modulu (6.1) a dosáhneme ji dosazením těchto hodnot a proměnných:

$$S1 = 0, S0 = c_i, A = a_i, B = \overline{a_i}, SA = b_i, C = \overline{a_i}, D = a_i, SB = b_i \qquad (6.5)$$



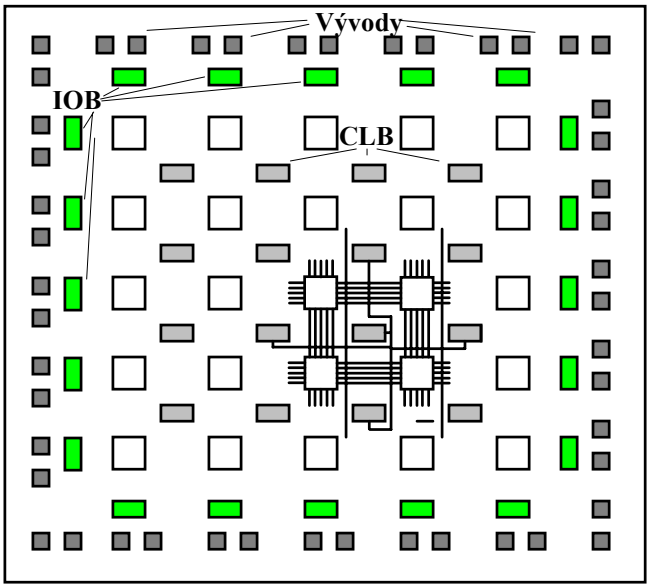
Obr.6.30

Analogicky získáme výraz pro přenos do dalšího řádu c_{i+1} přivedení těchto hodnot na logický modul

$$S1 = a_i,\ S0 = b_i,\ A = c_i,\ B = 0,\ SA = 1,\ C = 0,\ D = 1,\ SB = c_i \qquad (6.6)$$

Výsledné zapojení sčítačky bude tvořeno třemi logickými moduly obr.6.30.

6.10. Obvody LCA



Obr.6.31

Obvody LCA (Logic Cell Array) jsou další skupinou obvodů založených na principu propojování logických bloků označované v literatuře zkratkou FPGA. Firma Xiling, která produkuje většinu těchto obvodů, však začala svoje obvody označovat zkratkou LCA. I když princip zůstává s obvody předcházející skupiny stejný, svými charakteristickými vlastnostmi se LCA od ní výrazně odlišují. Na obr.6.31 je znázorněna architektura obvodů LCA, která se skládá opět ze tří stavebních prvků:

- programovatelných stavebních bloků CLB
- programovatelných vstupních a výstupních bloků IOB
- programovatelných spojovacích tras.

Na rozdíl od obvodů FPGA-TI jsou CLB umístěny uvnitř propojovací sítě, v které jsou spínače realizovány statickou pamětí RAM. Naprogramování spínačů a tedy i dané aplikace se provádí po připojení obvodu k napájení z hostitelského počítače nebo sériové paměti PROM k tomu určené. Obvody LCA firmy Xiling prošly dosud třemi vývojovými generacemi označovanými souborně XC2000 (1985), XC3000 (1987) a XC4000 (1990). Základní vlastnosti některých jejich představitelů jsou shrnuty do tabulky 6.3.

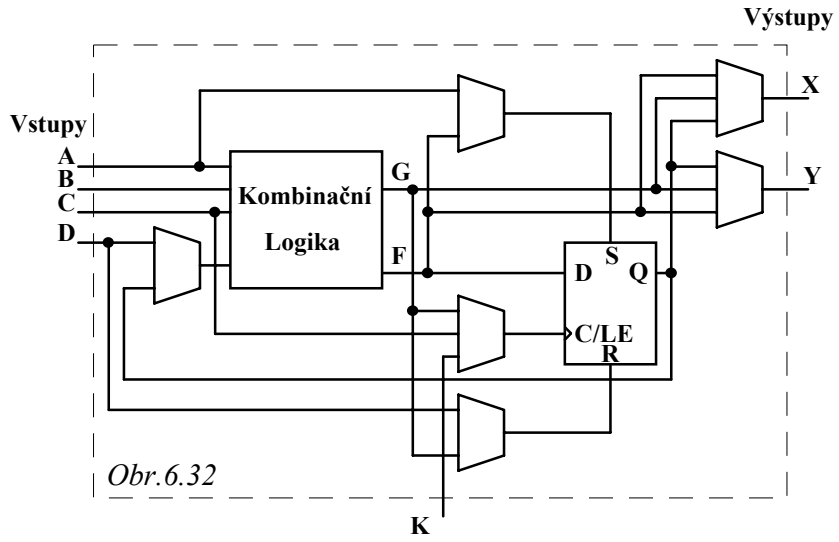
Základem obvodu LCA je matice identických konfigurovatelných logických bloků CLB, z nichž každý obsahuje programovatelnou logiku a paměťové registry. Kombinační část bloku realizuje Booleovské funkce, registry mohou být buzeny z výstupu kombinačního části nebo přímo ze svých vstupů CLB. Výstupy registrů mohou vstupovat do kombinační části přímo interní zpětnou vazbou. Vstupy a výstupy zprostředkovávají programovatelné IOB. Každý blok může být naprogramován nezávisle jako vstupní, výstupní s třístavovým výstupem a nebo obousměrný. Vstupy mohou být naprogramovány pro prahové úrovně log.0 a log.1 TTL nebo CMOS. Každý IOB také obsahuje klopný obvod, který může být použit jako vstupní nebo výstupní registr. Spojovací vedení je v LCA položeno horizontálně i vertikálně ve sloupcích a řádcích CLB obr.6.31. Programovatelné spínače, které leží v místech překřížení spojovacích

vodičů, zajišťují spojení mezi vstupy a výstupy IOB a CLB. Mimo to jsou položeny dlouhé cesty, které slouží k rozvedení kritických signálů s minimálním zpožděním. Ve srovnání s obvody z předcházející části je nesmírnou výhodou obvodů LCA možnost neomezeného přeprogramování obvodu, které ovšem vede minimálně na dvouobvodové řešení problému. V obvodech třetí generace XC4000 je část paměti RAM uživatelsky přístupná a může být využívána jako vyrovnávací nebo zásobníková paměť příslušné alikace. Obvody LCA mají výrazně vyšší pracovní hodinový kmitočet, který dosahuje hodnoty až 255MHz. I když se zdá, že propojovací síť bude umožňovat lepší variabilitu ve spojení jednotlivých bloků, nebývá v průměru využíváno více než 60% konfigurovatelných bloků. Při snaze o dobré využití počtu konfigurovatelných bloků stává se problematické vyhledávání spojovacích cest na konci propojovacího algoritmu.

Obvod	Ekvivalent počet hradel	Počet CLB	Počet kombinač. funkcí	Počet klopných obvodů	Počet vstupů/ výstupů	Počet bitů RAM
Řada XC2000						
XC2064	1 200	64	128	122	58	---
XC2018	1 800	100	200	174	74	---
Řada XC3000						
XC3020	2 000	64	128	256	64	---
XC3030	3 000	100	200	360	80	---
XC3064	6 400	224	448	688	120	---
XC3090	9 000	320	640	928	144	---
Řada XC4000						
XC4002	2 000	64	128	256	64	2 048
XC4004	4 000	144	288	480	96	4 608
XC4006	6 000	256	512	768	128	8 196
XC4010	10 000	400	800	1 120	160	12 800
XC4016	16 000	676	1 352	1 768	208	21 632
CX4020	20 000	900	1 800	2 280	240	28 800

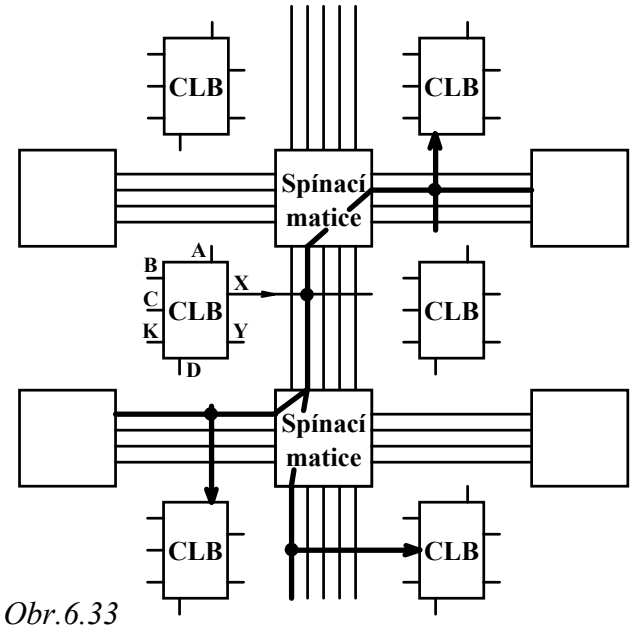
Tabulka 6.3

Obvody LCA od firmy Xiling se vyrábějí ve třech generacích z nichž každá má svoji vnitřní strukturu bloku CLB a I/O. Protože rozdíl mezi druhou a třetí generací není již tak výrazný, popíšeme si vnitřní strukturu prvé a třetí generace. Na obr. 6.32 je zobrazena vnitřní struktura CLB obvodů z řady XC2000, které jsou umístěny v matici o 8 řádcích a 8 sloupcích (XC2064) nebo 10 řádcích a 10 sloupcích (XC2018), kde lichoběžníky představují programově



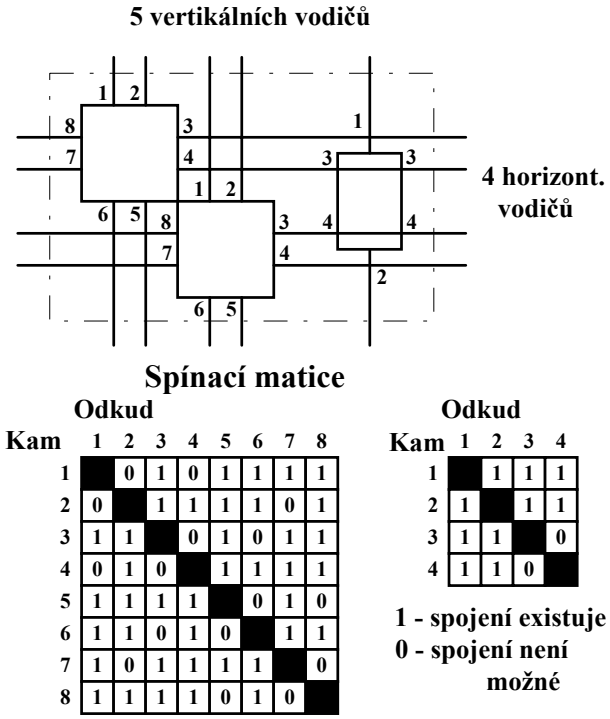
Obr.6.32

řízené multiplexery. Každý blok má kombinační část, která má čtyři vstupy A, B, C, D hodinový vstup CLK pro případné řízení paměťového členu a dva výstupy X a Y. Kombinační obvod může být nakonfigurován na libovolnou funkci 4 proměnných nebo na dvě funkce o třech proměnných se zpožděním nezávislém na realizované funkci. Pro funkci 4 proměnných jsou potom vnitřní výstupy F a G shodné. Používáme-li dvě funkce o třech proměnných, potom výstupy F a G můžeme používat nezávisle nebo vybírat jeden z nich pomocí multiplexeru řízeného vstupní proměnnou B. Kombinační funkce spolu s výstupem Q paměťového členu mohou představovat budící funkci vnitřního paměťového členu. Jeho použití v každé CLB může být naprogramováno individuálně na hranově nebo úrovňově řízený paměťový člen. Hodinový nebo aktivační signál paměťového členu může být realizován zvláštním vstupem K, všeobecně použitelným vstupem C nebo výstupem funkce G. V CLB je možné hodinový signál invertovat a proto není nutné rozvádět dva hodinové signály v programovatelném poli. Vstupem paměťového členu je logická funkce F, asynchronní nulování a nastavení může být ovládáno funkcemi nebo vstupními proměnnými.



Obr.6.33

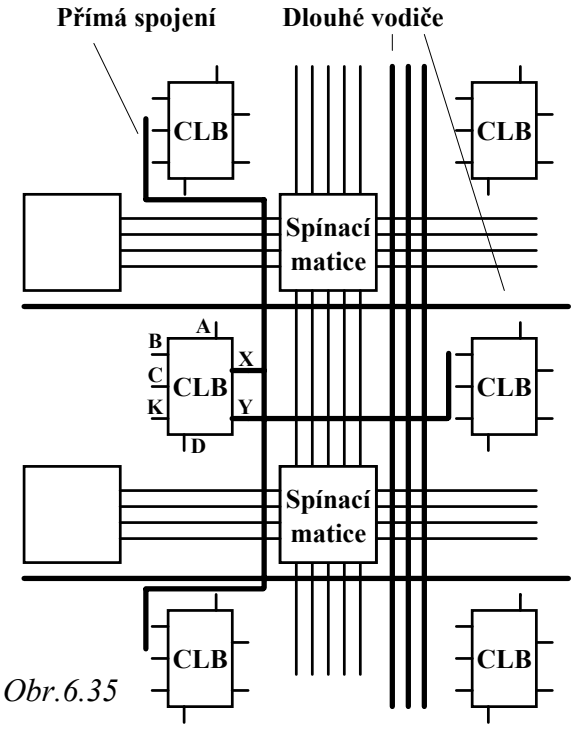
Propojení jednotlivých CLB se provádí pomocí nalezení spojovací cesty ve spínacích maticích a propojovací síti. V propojovací síti existují tři různé typy propojovacích vodičů: všeobecně použitelné vodiče, dlouhá spojení a přímá spojení. Všeobecně použitelná spojení zobrazená na obr.6.33 jsou tvořena 4 horizontálními a 5 vertikálními vodiči, které přichází do spínací matice obr. 6.34. Pravdivostní tabulky vyjadřující realizovatelné spojení ve spínací matici jsou též v obrázku zobrazeny. Každé spojení se provádí zvláštním spínacím



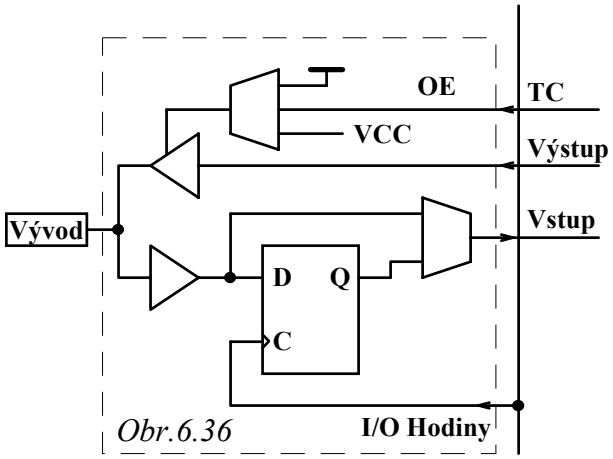
Obr.6.34

tranzistorem, který je ovládán konfiguračním bitem. Výstupy CLB jsou připojeny na vodiče této propojovací sítě a pomocí spínacích matic může zajistit propojení s ostatními CLB. Na obr.6.33 je zobrazeno takovéto spojený výstupu CLB se třemi vstupy dalších konfigurovatelných bloků. Kontrolu spojení i jejich automatické umístění v LCA struktuře provádí vývojový systém XACT, který umožňuje výpočet a zobrazení zpoždění v zvolené spojovací cestě. Na obr.6.35 jsou zobrazeny horizontální a vertikální dlouhé spojovací vodiče. V každém sloupec mezi CLB obsahuje dva dlouhé vodiče, každý řádek má pouze jeden vodič procházející přes šířku celého pole. Dlouhé vodiče procházejí spínacími maticemi jsou určeny pro privilegované signály, které musí procházet do velkých vzdáleností nebo musí mít malé zpoždění průchodu signálu.

Přímá spojení jsou zobrazena též na obr.3.35 a jsou nejvýhodnější realizací spojení mezi konfigurovatelnými a I/O bloky. Spojení realizované prostředky přímého propojení mezi CLB vykazují minimální zpoždění i prostředky k jeho realizaci. U každé CLB může být výstup X spojen přímo nebo vstupy C nebo D bloku, který se nachází nad ním, a vstupy A nebo B bloku, který se nachází pod ním. Výstup Y může být přímo propojen na vstup B následujícího bloku v řádku napravo. Jestliže CLB sousedí s I/O blokem, potom výstup bloku vlevo je propojen na vstup CLB, vstupy I/O bloku vpravo od CLB mají vstupy připojené na výstupy CLB. Bloky I/O umístěné nahoře nebo dole jsou propojeny jak na vstupy, tak na výstupy CLB. Na obr. 6.36 je zobrazena vnitřní struktura bloku I/O v řadě XC 2000. Každý blok zajišťuje spojení vývodu IO s vnitřní logikou a umožňuje jej využívat jako vstup



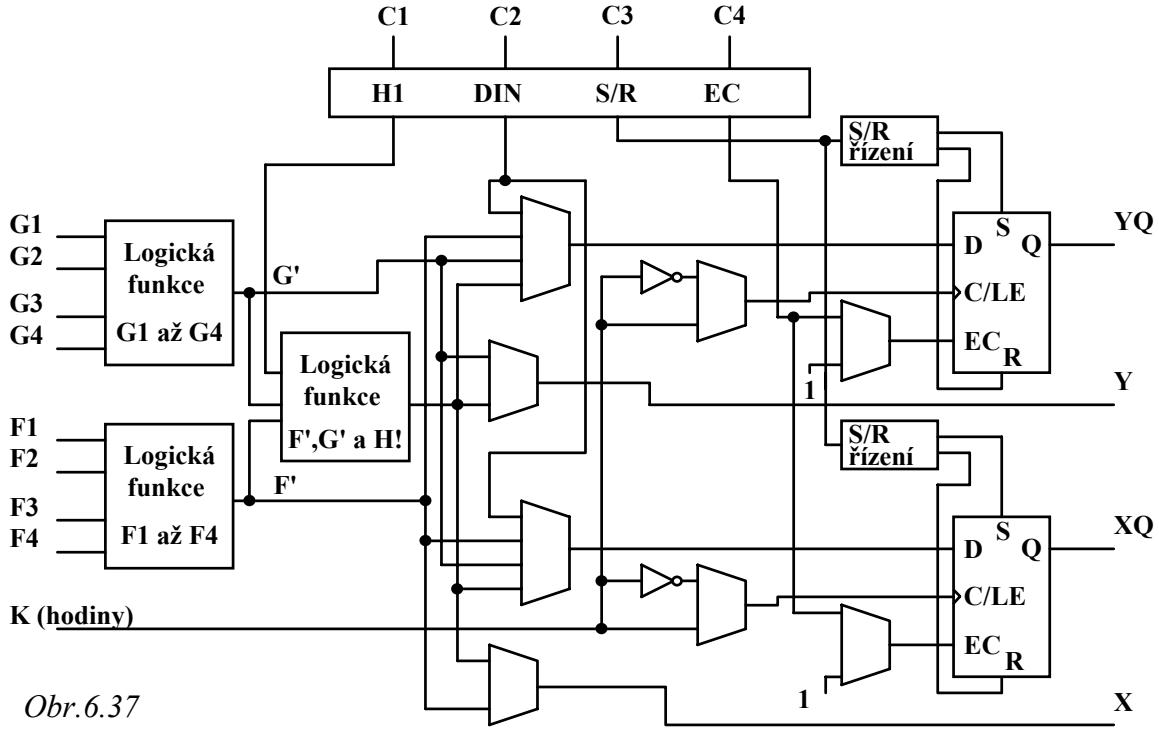
Obr.6.35



Obr.6.36

nebo výstup. I/O blok obsahuje na vývodu ochranné diody zajišťující ochranu obvodu proti zničení elektrostatickým polem. Vstupní obvod může být naprogramován na logické úrovně TTL (1,4V) nebo CMOS (2,2V) (VCC=5V). Uživatel si může vybrat vstup jako přímý nebo přes hranově řízený paměťový člen v závislosti na nastavení konfigurační buňky multiplexeru. Paměťové členy I/O bloků jsou hranově řízené společným hodinovým signálem. Během konfigurace jsou paměťové členy vynulovány aktivním signálem na nulovacím vstupu obvodu $\overline{\text{RESET}}$. Zatížitelnost výstupních budičů je 4mA a každý může být pomocí dvou konfiguračních buněk naprogramován na trvale aktivní, neaktivní nebo s řízeným třístavovým budičem signálem TS. Budič je aktivován úrovní log.0.

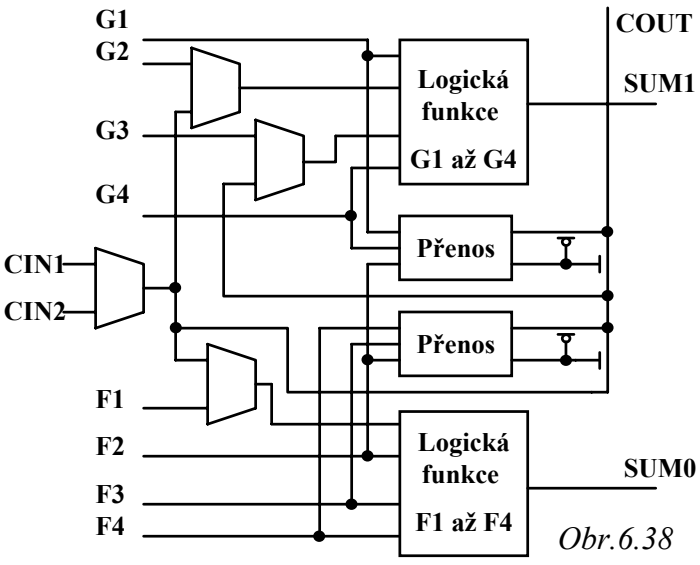
Obvody třetí generace z řady XC 4000 jsou ve srovnání se svými předchůdci výkonnější a umožňují realizovat paměť RAM přímo na čipu. Jsou mnohem přizpůsobivější dané aplikaci, umožňují zrychlení cyklu návrhu v důsledku zvýšení spojovacích prostředků a více propraco-



Obr.6.37

vaným návrhovým systémem. Na obr. 6.37 je zobrazena vnitřní struktura CLB, která zvýšila výkonnost a přizpůsobivost programovatelného pole. Každý CLB je nyní vybaven dvěma pa-

měťovými členy a dvěma čtyř vstupovými funkčními generátory logických funkcí, které poskytují návrháři dostatečnou míru flexibility. Vývojový systém může využít oba funkční generátory zcela nezávisle. Konfigurační logický blok má 13 vstupů a 4 výstupy, které jsou programově připojitelné ke spojovací síti. Každý funkční generátor, který umožňuje realizovat libovolnou booleovskou funkci čtyř vstupních proměnných (F1 až F4 nebo G1 až G4), má jeden výstup (F′ a G′). Funkční generátory jsou realizovány prostřednictvím tabulky uložené v paměti generátoru a díky tomu je velikost zpoždění nezávislá na realizované funkci. Třetí funkční generátor, jehož výstup je označen H′, může realizovat libovolnou booleovskou funkci se vstupy G′, F′ a H1, kde H1 je další vstupní signál. Výstupy funkčních generátorů jsou přivedeny na dva výstupy bloku CLB. Na výstup F může být připojen signál F′ nebo H′ a na výstup G může být připojen signál G′ nebo H′. Pomocí funkčních generátorů je možné realizovat dvě nezávislé funkce čtyř proměnných nebo jednu funkci pěti proměnných nebo jakoukoliv funkci čtyř proměnných spolu s některými funkcemi pěti proměnných nebo mohou být realizovány některé funkce až do devíti proměnných. Realizace funkcí s mnoha vstupními proměnnými v jednom bloku redukuje počet požadovaných bloků k realizaci obvodu a snižuje tak zpoždění v signálové cestě. V CLB jsou umístěny dva hranově řízené paměťové členy D se společným hodinovým signálem (K) a aktivačním hodinovým vstupem (EC). Třetím společným vstupem je společný vstup (S/R), který může být naprogramován pro každý paměťový člen nezávisle na asynchronní nastavení nebo nulování, případně nemusí být použit vůbec. V poli je dále umístěn globální signál Set/Reset, který není v obr. 6.37 nakreslen, a slouží k nastavení nebo nulování každého paměťového členu během připojení napájení, rekonfigurace obvodu nebo aktivace sítě RESET. Způsob vytváření této sítě je odlišný od vytváření spojení mezi bloky na čipu a může být připojen na jakýkoliv vývod obvodu jako globální vstup nulovaní. Díky řízenému multiplexeru a negaci může být paměťový člen řízen náběžnou nebo sestupnou hranou hodinového signálu. Budící funkcí paměťového členu může být signál G′, F′, H′ nebo vstupní signál DIN. Výstupy z klopných obvodů jsou vyvedeny jako zbývající dva výstupy Q1 a Q2.

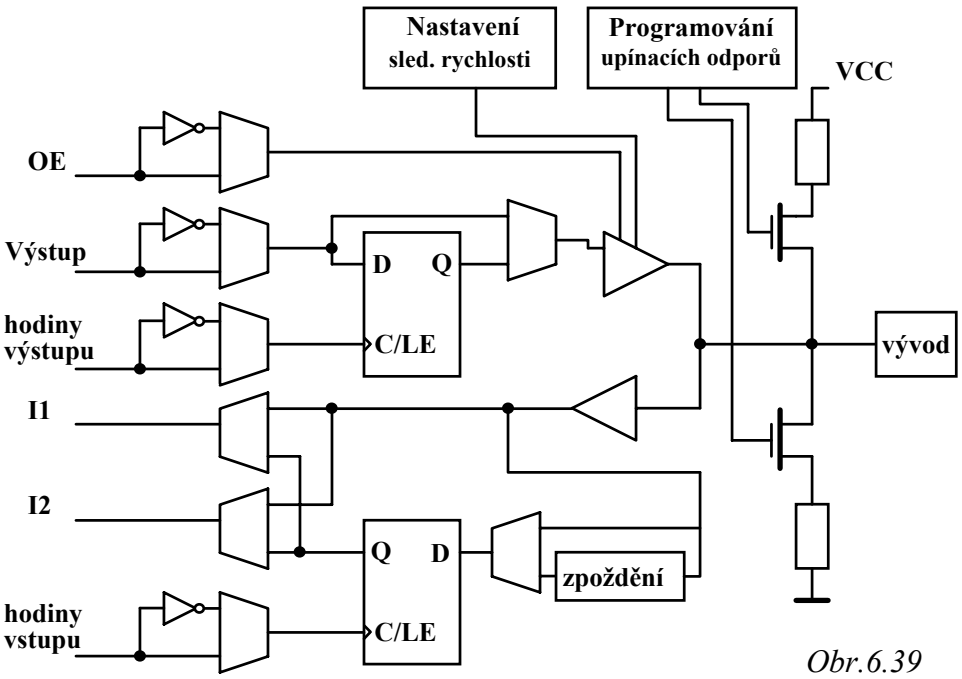


Obr.6.38

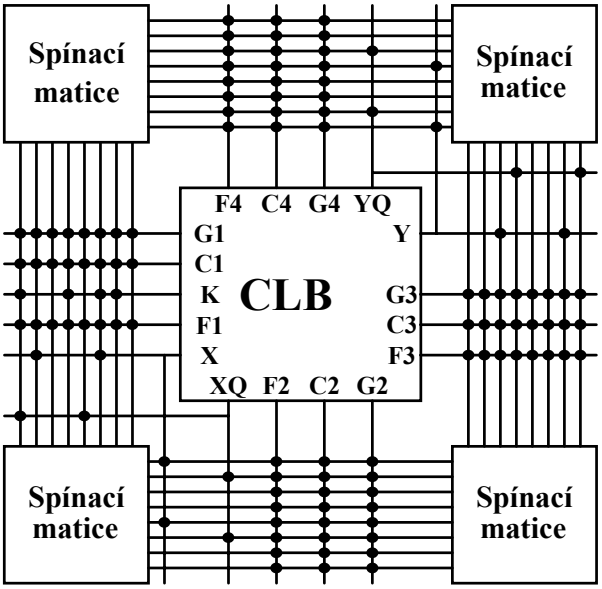
Výraznou odlišností od předcházejících řad je zařazení obvodů rychlého přenosu v bloku CLB, které umožňují generování aritmetického přenosu (Carry) nebo záporného přenosu (Borrow), čímž mohou být efektivněji realizovány sčítačky, odčítačky, akumulátory, komparátory a čítače. Dva 4 vstupové funkční generátory obr.6.38 mohou být

konfigurovány jako dvoubitové sčítačky i s vytvořením přenosu tak, aby mohly být rozšiřovány na libovolnou délku sčítaných bitů. Obvod přenosu je velmi rychlý a umožňuje realizaci například 16 bitové sčítačky s postupným přenosem z 9 bloků CLB za 20,5 ns. Pro srovnání v řadě XC3000 bylo potřeba 30 CLB na 50 ns nebo 41 CLB na 30 ns. Obvody rychlého přenosy tak umožňují realizovat s obvody LCA nové aplikace obsahující aritmetické operace jako jsou obvody rychlého adresování, nebo rychlého sčítání v číslicovém zpracování signálů, které s obvody předcházející generace nebyly příliš efektivní.

Kvantitativní změnou je též zařazení uživatelské paměti RAM do programovatelného pole. Realizace logických funkcí pomocí tabulky uložené v paměti RAM nám umožňuje konfigurovat blok CLB též jako paměťové pole s kapacitou 16x2 nebo 32x1 bitů RWM (RAM). Vstupy F1 až F4 a G1 až G4 pak představují adresové vodiče, vstupní vodiče H1, DIN a S/R se konfigurují na nový význam. V konfiguraci 16x2 bity se konfigurují na datové vstupy D0 a D1 a signál pro zápis. V konfiguraci 32x1 se vstup D1 stává zbývajícím adresovým vodičem. Konfigurace funkčních generátorů na RWM v bloku CLB neovlivňuje funkci zbývajících obvodů jako jsou paměťové členy a funkční generátor H′. Jenom počet vstupních proměnných pro tyto bloky je omezen. Paměť RWM je velmi rychlá a cyklus čtení u ní trvá 5,5 ns a zápisu 8 ns. To jsou hodnoty skoro dvakrát rychlejší než u řešení mimo čip IO. Začlenění paměti RAM nám umožňuje realizovat řadu nových aplikací jako jsou řadiče DMA (přímého přístupu do paměti), zásobníky FIFO a LIFO, snazší realizaci posuvných, indexových a stavových registrů i samotných akumulátorů. Na obr.6.39 je potom zobrazena vnitřní struktura bloku I/O. Každému vývodu obvodu je určen právě jeden blok I/O, který řídí přenos signálu daným vývodem konfigurovaným na vstupní, výstupní nebo obousměrný přenos. Vstup je přiveden na hranově nebo úrovňově řízený klopný obvod D a může být programově zpožděn o několik nanosekund s cílem kompenzovat zpoždění taktovacího (hodinového) signálu, které vznikne průchodem globálním posilujícím bufrem. Získané signály označené I1 a I2 mohou

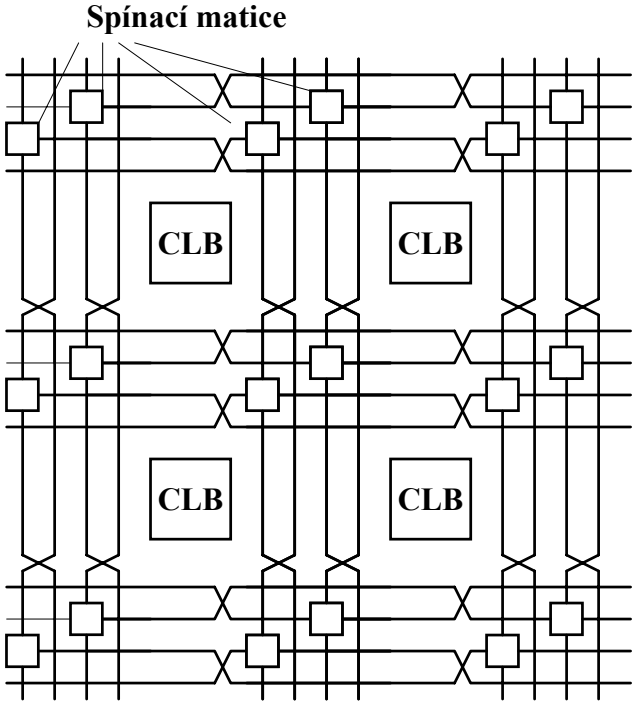


Obr.6.39



Obr.6.40

mít charakter přímých nebo registrovaných signálů. Výstupní signál může procházet blokem I/O buď přímo nebo může být uložen do paměťového členu. Výstup opět prochází třístavovým budičem řízeným aktivačním signálem OE, který umožňuje výstup trvale aktivovat, deaktivovat nebo aktivovat v případě obousměrného přenosu. Zvýšeno bylo zatížení jednotlivých výstupů v log.0 na 12 mA. Navíc je každý blok vybaven odpory, které umožňují vnitřně vývod připojit k napájení nebo na zem. Součástí bloku I/O je vestavěná testovací struktura kompatibilní se standardem IEEE 1149.1 pro testování samotného čipu nebo osazeného na desce pomocí externích testerů. Na rozdíl od předcházejících generací LCA, které měly vyvedeny vstupní signály CLB nahoře, dole a nalevo a výstupní signály napravo od CLB a podporovaly tudíž průchod signálu zleva doprava, jsou u třetí generace vstupy i výstupy vyvedeny na všechny strany obr.6.40. Architektura je symetrická, umožňuje větší přizpůsobivost, snazší rozmístění aplikace v obvodu i propojení jednotlivých bloků CLB.



Obr.6.41

Ve struktuře existují tři základní typy propojení, které se odlišují délkou, spoje s jednotkovou délkou, s dvojnásobnou délkou a dlouhé vodiče. Jednotkové spoje jsou spoje, které prochází propojovacími maticemi a jsou určeny pro vedení signálů v rámci malé oblasti čipu. Dvojnásobné spoje procházejí přes propojovací matice každého druhého bloku CLB a jsou pro snadné střídání vytvořeny vždy dvěma vodiči obr.6.41. Tyto spoje jsou nejefektivnějším pro realizaci středně dlouhých propojení. Zvláštní skupinu vertikálních dlouhých vodičů posílených vyrovnávacími registry tvoří vodiče určené k šíření hodinových synchronizačních signálů.

Dodatek A

Formát INTEL HEX

Formát INTEL HEX byl vytvořen k přenosu datového pole 8-bitových a 4-bitových hodnot do vývojových systémů a programovacích zařízení pamětí PROM/ROM a EPROM. Každý byte v tomto formátu je vyjádřen dvěma hexadecimálními znaky, které vyjadřují 8-bitovou nebo 4-bitovou hodnotu. Popisovaný formát, který se v současnosti používá ke generování obsahů pamětí překladači vyšších jazyků nebo jazyka symbolických adres, lze rozdělit do 6 částí, které postupně popíšeme.

1. **Počáteční značka pole** - je reprezentována dvojtečkou (ASCII - 3AH), která je prvním znakem přenášeného řetězce.
2. **Délka přenášeného datového pole** - je vyjádřena dvěma ASCII hexadecimálními znaky. Vyšší řád je na druhé pozici a nižší řád na pozici třetí v přenášeném záznamu. Maximální počet datových bytů je 255 (FF hexadecimálně). U posledního záznamu v přenášeném souboru je délka záznamu nulová (2 ASCII znaky jsou nuly).
3. **Počáteční adresa pole** - je vyjádřena 4 ASCII hexadecimálními znaky na 4 až 7 pozici v přenášeném řetězci, které určují adresu, na kterou má být uložen první datový údaj. Cifra s největší váhou je umístěna na pozici je umístěna na pozici 4 a s nejmenší váhou na pozici 7. První datový údaj je uložen na tuto adresu a další na následující adresová místa. Toto pole je pro poslední záznam v souboru nulové nebo obsahuje počáteční (startovací) adresu programu.
4. **Typ záznamu** - je vyjádřen dvěma ASCII hexadecimálními znaky, které specifikují typ záznamu. Vyšší řád je na pozici 8. Všechny datové záznamy jsou typu nula (00), konec záznamu je typu jedna (01). Další možné hodnoty byly rezervovány pro možný rozvoj formátu, ke kterému nedošlo.
5. **Datové pole** - je dáno pozicemi 10 až 10+2*(délka záznamu)-1. Datové byty jsou reprezentovány dvěma znaky obsahující ASCII znaky 0 až 9 a A až F, které reprezentují hexadecimální hodnotu 0 až FF (0 až 255 dekadicky). Vyšší cifra je na prvé pozici v každém páru. Na konci záznamu nejsou žádná data.
6. **Kontrolní součet** - se nachází na posledních dvou pozicích (10+2*(délka záznamu) a 10+2*(délka záznamu)+1. Kontrolní součet vyjadřuje dvěma ASCII hexadecimálními znaky dvojkový doplněk 8-bitového součtu všech bytů,

převedených do binárního tvaru z ASCII vyjádření, počínaje délkou přenášeného pole a konče posledním datovým údajem. Proto spočet celého záznamu vyjma počáteční značky musí být při bezchybném přenosu nulový.

Jeden záznam můžeme symbolicky vyjádřit tímto vztahem

Typ pole	1	2	3	4	5	6
Záznam	**:**	**pp**	**aaaa**	**uu**	**bybybyby...by**	**ks**

k formátu je třeba dodat, že mezi byty nejsou mezery, které autor použil pro zpřehlednění formátu. Na závěr uvedeme ukázku souboru v INTEL HEX formátu, který byl vygenerován překladačem jazyka symbolických adres.

```
:1000B800EA00A805EA00BC00CC00CC00CC00CC00CC00CB
:0E00C800CC00CC00B748B7B7EA002A059B6809
:04020000FA01452397
:10021000A55AA5A5CFE28FE2DFE276E30031B54A29
:10022000B5B5E65E0A00FFDC1FDCE75C1000E0000D
:1002300064F0F201F6F0E801F6F1EA01F6F2EC0101
:10024000F601EE01F603F001E6B93500E6BA360034
:10025000E6BB37004FDCEA0080029AB9FE707AB93B
:10026000B977F35C04FC9ABAFE707ABABA77CB001D
:10027000ECF2F145CA005A02FCF2CA005A02CB0065
:1005B00000FCCA00E204E000F6F0F4017ABABA776F
:0805C000F75DE001EA008C0286
:00000001FF
```

Dodatek B

JEDEC Standard č.3

V této příloze je uvedena implementace formátu JEDEC (Join Electron Device Engineering Council), který byl vytvořen pro bitově orientované logické součástky jako je PLA, PAL, EPLD apod., v listopadu 1983. Formát JEDEC se stal standardním datovým formátem pro přenos mezi systémem připravujícím data a logickým programovacím zařízením. Uvedený standard není u mnoha vývojových systémů přesně dodržován, obsahuje řadu modifikací a úprav, a proto je nutné s touto okolností počítat. Informace posílané programovacímu zařízení lze rozdělit do několika skupin (polí), které postupně probereme. Soubor JEDEC je textový soubor, který smí přenášet pouze tyto povolené znaky:

STX	02H	začátek textu	ETX	03H	konec textu
LF	0AH	nový řádek	CR	0DH	návrat vozíku

Všechny znaky od 20H až 7EH (znaky které lze tisknout na tiskárně).

Přenášený soubor generovaný vývojovým systémem nebo asemblerem začíná volitelným polem, které **identifikuje obsah návrhu** v JEDEC souboru. Může obsahovat jména autorů, název firmy, název projektu atd. Dalším volitelným polem je **přiřazení názvů vývodů** u navrhovaného PLD obvodu. Každé přiřazení je v samotném řádku, který má následující formát: N@ symbolický_název_vývodu (max. 5 znaků) @číslo_vývodu* (např. N@ CLK @1 * = přiřazení názvu CLK vývodu číslo 1). Po těchto volitelných polích následuje povinný znak STX (ASCII znak 02H), který označuje **začátek souboru**. V následujícím poli, které je polem povinným, se definuje **typ programovaného obvodu**. Pole se skládá z klíčového slova následovaného typem obvodu (např. PAL16R8) a je zakončeno hvězdičkou. Po identifikaci typu PLD mohou následovat volitelná pole bezpečnostního bitu, velikosti programované paměti a počtu vývodů. Pole **bezpečnostního bitu** se skládá z klíčového slova **G** následovaného 0 nebo 1 podle toho, je-li bezpečnostní bit nastaven nebo ne. Položka je zakončena hvězdičkou. Pole **velikosti programovatelné paměti** se stává z klíčového slova **QF** následovaného počtem programovatelných bitů daného obvodu PLD. Počet je vyjádřen dekadicky a je zakončen hvězdičkou. Pole **počtu vývodů** se skládá z klíčového slova **QP** následovaného dekadicky vyjádřeným počtem pinů daného obvodu. Položka je zakončena hvězdičkou.

Po těchto informačních a nezbytných polích následuje vlastní pole hodnot programované paměti. Každá propojka nebo paměťové místo programovaného obvodu má přiřazeno deka-

dické číslo (adresu) a může nabývat pouze dvou stavů. Nula-určuje malý odpor propojky (nerozpojené) a jednička-určuje vysokou impedanci propojky (rozpojené). Informace jsou přenášeny ve třech polích **F**,**L** a **C**.

Pole **F** - specifikuje stav nedefinovaných propojek daného obvodu. Tato informace odpovídá příkazům DEFAULT v sekci zdrojového programu. Implicitní nastavení tohoto pole je F0 - všechny nespecifikované propojky se neprogramují.

Pole **L** - Každý řádek začíná písmenem L, za kterým následuje čtyřmístné dekadické číslo určující adresu první propojky v následujícím řeťězci dat. Řetězec dat , který začíná mezerou, může být jakkoliv dlouhý a je zakončen znakem *. Většinou mívá řetězec délku 32 znaků. Pole **L** je ukončeno nepovinným řádkem s kontrolní sumou, která je definována polem **C.**

Pole **C** - Kontrolní součet pro řádkovou informaci je vypočten jako 16-bitový součet (čtyři hexadecimální znaky) 8-bitových slov vytvořených z reverzního řádku přenášených dat. Předpokládejme, že máme takovýto soubor JEDEC

```
<STX> F1
L0000 01001011 11111111 11111111 11111000*
C02EF
<ETX> 0A94
```

potom výpočet kontrolního součtu se provádí následujícím způsobem

8-bitová slova	hex
1101 0010	D2H
1111 1111	FFH
1111 1111	FFH
0001 1111	1FH

	02EFH

Jestliže číslo propojky není násobkem 8, potom poslední slovo bude vytvořeno vynulováním všech významnějších bitů.

Přenášený textový soubor JEDEC je ukončen znakem ETX (ASCII znak 03H), který signalizuje **ukončující pole,** následovaný 16-bitovou hexadecimálně vyjádřenou kontrolní sumou mezi znakem STX a ETX. Kontrolní suma je vytvořena 16-bitovým součtem všech ASCII znaků v přenášeném souboru mezi znaky STX a ETX. Paritní bit je z tohoto výpočtu vyřazen.

Příklad na soubor JEDEC vytvořený návrhovým systémem PALASM2 je uveden dále pro navržený obvod z příkladu 2.22.

```
PALASM XPLOT, V2.23D - MARKET RELEASE (2-14-89)
(C) - COPYRIGHT ADVANCED MICRO DEVICES INC., 1989

Title   : Prioritni koder        Author  : Skalicky
Pattern : 01.                   Company : CVUT FEL
```

Revision : 01_def Date : 9.4.1994

☺
PAL20L8
PKODER*
QP24*
QF2560*
G0*F0*
L0000 11111111111111111111111111111111111111*
L0040 10101011101110111011101110111111111111*
L0080 11111111111111111111111111111011111111*
L0120 00000000000000000000000000000000000000*
L0160 00000000000000000000000000000000000000*
L0200 00000000000000000000000000000000000000*
L0240 00000000000000000000000000000000000000*
L0280 00000000000000000000000000000000000000*
L0320 11111111111111111111111111111111111111*
L0360 11111111111111111111111111111101010110*
L0400 00000000000000000000000000000000000000*
L0440 00000000000000000000000000000000000000*
L0480 00000000000000000000000000000000000000*
L0520 00000000000000000000000000000000000000*
L0560 00000000000000000000000000000000000000*
L0600 00000000000000000000000000000000000000*
L0640 11111111111111111111111111111111111111*
L0680 11111111111111111111111111111110010110*
L0720 00000000000000000000000000000000000000*
L0760 00000000000000000000000000000000000000*
L0800 00000000000000000000000000000000000000*
L0840 00000000000000000000000000000000000000*
L0880 00000000000000000000000000000000000000*
L0920 00000000000000000000000000000000000000*
L0960 11111111111111111111111111111111111111*
L1000 11111111111111111111111111111101011010*
L1040 00000000000000000000000000000000000000*
L1080 00000000000000000000000000000000000000*
L1120 00000000000000000000000000000000000000*
L1160 00000000000000000000000000000000000000*
L1200 00000000000000000000000000000000000000*
L1240 00000000000000000000000000000000000000*
L1280 11111111111111111111111111111111111111*
L1320 11111111111111111111111111111110011010*
L1360 00000000000000000000000000000000000000*
L1400 00000000000000000000000000000000000000*
L1440 00000000000000000000000000000000000000*
L1480 00000000000000000000000000000000000000*
L1520 00000000000000000000000000000000000000*
L1560 00000000000000000000000000000000000000*
L1600 11111111111111111111111111101111111111*

L1640 11111111111110111011101110111111111111111*
L1680 00000000000000000000000000000000000000000*
L1720 00000000000000000000000000000000000000000*
L1760 00000000000000000000000000000000000000000*
L1800 00000000000000000000000000000000000000000*
L1840 00000000000000000000000000000000000000000*
L1880 00000000000000000000000000000000000000000*
L1920 11111111111111111111111111111011111111111*
L1960 11111111111101111111101110111111111111111*
L2000 11111011101111111111101110111111111111111*
L2040 11111111111111110111011101111111111111111*
L2080 00000000000000000000000000000000000000000*
L2120 00000000000000000000000000000000000000000*
L2160 00000000000000000000000000000000000000000*
L2200 00000000000000000000000000000000000000000*
L2240 11111111111111111111111111111011111111111*
L2280 11111111111101111011111110111111111111111*
L2320 10111111101111111011111110111111111111111*
L2360 11110111101111111011111110111111111111111*
L2400 11111111111111111111011101111111111111111*
L2440 00000000000000000000000000000000000000000*
L2480 00000000000000000000000000000000000000000*
L2520 00000000000000000000000000000000000000000*
C691B*
♥4B4E

Dodatek C

PALASM se používá k návrhu PLD (programovatelných logických obvodů) tak, že vytvořený vstupní soubor konvertuje do souboru JEDEC. Návrh vytvoří předpis určující, které propojky obvodu mají být naprogramovány. PALASM2 připouští návrh z booleovských nebo stavových rovnic. Návrh může obsahovat simulační část, která umožňuje simulovat vlastnosti návrhu bez nutnosti naprogramování obvodu.

Jestliže pracujeme s booleovskými rovnicemi, potom vstupní soubor s příponou .PDS se skládá ze dvou částí. První částí je deklarace, která charakterizuje návrh, typ obvodu, přiřazení proměnných vývodům obvodu a substituční řetězce. Sekce obsahující booleovské rovnice je uvozena klíčovým slovem EQUATIONS. Zápis vlastních rovnic vychází z běžných konvencí, které se pro zápis rovnic používají. Operátor OR je vyjádřen symbolem ′+′, operátor AND symbolem ′∗′ a operátor neekvivalence (EX-OR) symbolem ′:+.′. Znak ′/′, který předchází logickou proměnnou nebo výraz v závorce je symbolem logické negace nebo signálu s aktivní úrovní log.0. Jako u většiny PLD jazyků i jazyk PALASM definuje dva operátory přiřazení ′=′ a ′:=′. První se používá při kombinatorickém přiřazení, druhý k registrovému výstupu řízeného hodinovým signálem. Při psaní vstupního souboru není možné používat znaky ~ !@#$^&[]{}?< ani žádné z klíčových slov rezervovaných systémem pro identifikaci segmentu, příkazu, funkce nebo přiřazení vývodu. PALASM2 má rezervovaná následující slova:

AUTOR	DO	NEXT_STATE	SIMULATION
BEGIN	ELSE	OR	STATE
CHECK	END	OUTPUT_ENABLE	STRING
CHIP	EQUATIONS	OUTPUT_HOLD	THEN
CLK	FOR	PATTERN	TITLE
CLOCKF	GND	POWER_UP	TRACE_ON
CMBF	HOLD_STATE	PRLDF	TRACE_OFF
COMPANY	IF	R	TRST
CONDITION	MASTER_RESET	REVISION	VCC
DATE	MEALY_MACHINE	RSTF	WHILE
DEFAULT_BRANCH	MOORE_MACHINE	S	
DEFAULT_OUTPUT	NC	SETF	

Deklarační část vstupního souboru začíná tímto záhlavím:

TITLE	název realizace
PATTERN	název vzoru [souboru]
REVISION	označení verze souboru
AUTHOR	jméno autora
COMPANY	název instituce
DATE	datum vytvoření

CHIP označení realizace typ integrovaného obvodu

v kterém jsou uvedeny všeobecné informace včetně cílového obvodu PLD. Kterákoliv část úvodní deklarace může být vynechána kromě klíčového slova CHIP. Při překladu budou hlášena varování, ale překlad bude probíhat dál. Po tomto záhlaví se přiřazují jednotlivé signály vývodům integrovaného obvodu počínaje vývodem 1 a konče posledním vývodem. Nevyužité vývody se označují NC, zemní vývod GND a napájení VCC. Signály jsou od sebe odděleny alespoň jednou mezerou, tabelátorem nebo čárkou.

Příklad:

;Vývody	1	2	3	4	5	6	7	8	9	10
	CLK	K	S	RST	nc	nc	nc	nc	nc	GND
;Vývody	11	12	13	14	15	16	17	18	19	20
	/OE	Q1	Q2	Q3	Q4	nc	nc	nc	nc	VCC

Je-li v přiřazení vývodů obvodu uvedeno jméno signálu, předpokládá se, že jeho aktivní úrovní je log.1. Je-li před ním umístěn znak /, pak jeho aktivní úrovní je log.0.

V některých realizacích se v rovnicích popisujících chování obvodu opakují části logických funkcí, které lze před segmentem rovnic nahradit jednou proměnnou STRING. V popisu rovnic pak uvádíme jen jméno proměnné.

STRING proměnná 'logická funkce'

Logický výraz je rozumné uzavřít do závorek, aby při případném násobení proměnné s jiným výrazem nedošlo k chybné interpretaci logické funkce.

Segment rovnic, který je uveden klíčovým slovem EQUATIONS, obsahuje logické rovnice, které mají být naprogramovány do obvodu, a které definují jeho výstupy jako funkce vstupních proměnných a zpětnovazebních smyček. Za tímto názvem jsou uvedeny logické (booleovské) rovnice pro kombinační nebo sekvenční (rovnice přechodů a výstupů) obvody. **Kombinační rovnice**, které určují výstup jako okamžitou hodnotu vstupních proměnných, se vyjadřují takto

Výstupní_proměnná = Signál * Signál*...+ .... + Signál * Signál

Výstupní signál na pravé straně rovnice je definován součtovou nebo součinovou formou signálů na levé straně rovnice. Výstup může být aktivní v log.0 (/výstup) nebo v log.1 (výstup). Na obvodech s programovatelnou polaritou je propojka polarity programována nebo zůstává bez změny podle toho, jak je výstupní proměnná určena na levé straně rovnice a ve výpisu vývodů u příkazu CHIP. Když jsou obě polarity stejné, propojka je naprogramována a výstup zůstává aktivní v log.1. Jestliže jsou polarity rozdílné, potom propojka se neprogramuje a výstup zůstává aktivní v log.0.

Sekvenční rovnice, které určují budoucí výstupní hodnoty paměťových členů jako logickou funkci okamžitých hodnot vstupních proměnných a současného stavu paměťových členů se vyjadřuje takto

Výstupní_proměnná := Signál * Signál *....+..... ....+ Signál * Signál

Výstup paměťového členu může být aktivní v log.0 (/výstup) nebo v log.1 (výstup) stejně jako u kombinačních rovnic. U většiny programovatelných obvodů je synchronizace paměťových členů hodinovým signálem společná a vyvedená na jeden vývod integrovaného obvodu. U několika málo obvodů (např. PAL20RA10) se generují hodiny zvláštní funkcí.

U programovatelných obvodů (kapitola 6) je většinou možné individuální ovládání třístavového budiče na výstupech obvodu a globální nebo individuální nastavení nebo nulování obvodu. Pro tyto účely je jazyk PALASM vybaven příkazy, které zajišťují nastavení požadovaných funkcí. Je-li nulování nebo nastavení globálně programovatelné, potom musí být v přiřazení vývodů jednotlivým proměnným přidán za poslední skutečný vývod (většinou VCC) fiktivní vývod s definovanou proměnnou.

Příklad: ;vývody 23 24 25

OUT7 VCC GLOBAL

Individuální nebo globální ovládání výstupů paměťových členů se provádí příkazy:

25_vývod.SETF = Výraz pro nastavení

25_vývod.RSTF = Výraz pro nulování

Výstupní_proměnná.TRSF = Výraz pro řízení třístavového budiče

Příklad:

```
GLOBAL.SETF = A * /B
Q14.TRSF = /A * B
```

kde výstupní proměnná je globální (GLOBAL - pro všechny paměťové členy) a nebo lokální (Q14 - totožná se jménem přiřazeným některému výstupu). Podmínka může být vyjádřena GND (trvale neaktivní - interní nastavení), VCC (trvale platná) nebo logickým výrazem, který úrovní logická jedna zajišťuje příslušnou činnost.

Návrh obvodu ze stavového diagramu se provádí pro dva základní modely sekvenčních obvodů Mealyho a Moorův. Stavový diagram popisuje výstupní hodnoty obvodu pro jednotlivé stavy a popisuje přechody mezi nimi v závislosti na vstupních proměnných stavy. Návrh obvodu ze stavového diagramu potom obsahuje tři části deklarační, která obsahuje identifikaci návrhu, typ obvodu, přiřazení proměnných vývodům obvodu a substituční řetězce stejně jako u návrhu z booleovských rovnic, přiřazení vnitřních proměnných obvodu jednotlivým stavům a podmínek podmiňujících jednotlivé přechody. Příkaz STATE, který informuje systém o návrhu

ze stavového diagramu, může být následován všeobecnými nastaveními tj. příkazy určující model obvodu a definující jeho výstupní hodnoty a přechody v případech, kdy nebudou určeny stavovými a výstupními rovnicemi. Příkazy pro modely obvodu jsou **MEALY_MACHINE**, který je implicitně nastavený, a **MOORE_MACHINE**. Příkaz definující výstupní hodnoty má tento tvar: **DEFAULT_OUTPUT výstupní vývody** - výpis výstupních vývodů, které mají být nastaveny do dané hodnoty, jestliže jejich hodnoty nebudou určeny návrhem. Hodnoty jsou určeny umístěním následujících symbolů před jméno vývodu: log.1 = žádný, log.0 = /, nezáleží = % . Příkaz definující přechod je dán výrazem **DEFAULT_BRANCH stavové jméno** - určuje následující stav obvodu pro případy, kdy stav není determinován návrhem. Příkaz **DEFAULT_BRANCH HOLD_STATE** - ponechá obvod v současném stavu, jestliže není determinován návrhem.

Po všeobecných nastaveních následují **přiřazení vnitřních proměnných** stavům obvodu, které se provádí těmito výrazy:

Stavové_slovo = vývod1*....*....*vývodN

kde samotný název vývodu značí log.1 a znak pro negaci před před názvem značí log.0. Po přiřazení vnitřních proměnných následují vlastní rovnice přechodů, které mají tvar

Stavové_jméno := podmínka_1 -> následující stav

....

+ podmínka_n -> následující stav

kde podmínka_1 je určena hodnotou vstupní proměnné nebo proměnnou, které je v sekci podmínek přiřazen logický výraz.

Příklad: N0 := COND1 -> N1 + COND2 -> N1 + COND3 -> N2 + -> N0
N2 := VCC -> N3

Podmínkový segment, který je označen klíčovým slovem **CONDITIONS**, obsahuje rovnice, které přiřazují podmínkovým jménům logické výrazy (podmínky) tvořené vstupními proměnnými obvodu takto:

Podmínkové_jméno = Vstup 1 * Vstup x + ... + Vstup y * Vstup n

Jméno podmínky musí být jedinečné a smí mít maximálně 14 znaků, je-li podmínka tvořena pouze jednou proměnnou podmínková rovnice není nutná. Nepovolené nebo konfliktní podmínky vzniknou, jestliže dvě nebo více podmínek je pravdivých ve stejném čase a jsou užity ve stejné rovnici přechodů.

Poslední část souboru popisující navrhovaný obvod, může obsahovat simulační část, kterou můžeme optimalizovat a ověřit správné chování obvodu před jeho naprogramováním. Simulace vytváří prostředky k nastavení vstupních hodnot pro kontrolovaný návrh a umožňuje

kontrolu událostí podporující všechny typy obvodů PAL realizující kombinační, synchronní i asynchronní sekvenční obvody. Program reálně simuluje události generované asynchronní nebo synchronní zpětnou vazbou obvodu na vstupní signály. Simulátor detekuje zprávy o oscilačních podmínkách a konflikty v očekávaném a skutečném stavu obvodu. Klíčová slova simulační části jsou tvořena ekvivalentními anglickými slovy, která umožňují snadné pochopení. Ta nabízejí interaktivní smyčky, podmíněnná větvení, nastavení signálů, ověření signálů a hodnot a volitelný výběr sledovaných signálů. Simulační část začíná klíčovým slovem **SIMULATION**, po kterém je možné volit pomocí příkazu TRACE_ON vstupní a výstupní proměnné, jejichž průběhy budou uloženy do zvláštního souboru. Příkaz má tvar

TRACE_ON seznam vstupních a výstupních proměnných

Po definici sledovaných proměnných následují příkazy umožňující nastavení jednotlivých proměnných nebo generování hodinových signálů.

PRLDF výstup1 výstup3 - slouží k inicializace výstupních registrů u obvodů s přednastavením. U obvodů bez přednastavení představuje PRLDF vhodný příkaz k nastavení výstupů do požadovaných hodnot místo dlouhé simulace vedoucí do tohoto stavu.

SETF proměnná1 /proměnná2 - specifikuje nové vstupní hodnoty (nastaví proměnná1=log.1 a proměnná2 = log.0) a je obvykle prvním příkazem po PRLDF, když se simulují výstupy registrů obvodu. Na začátku simulace nejsou všechny signály definované a jsou označeny písmenem **x** v simulačním výpisu. Signál může být nastaven, jestliže chcete změnit jeho předešlou hodnotu. Signály v příkazu SETF jsou nastaveny do log.1, jestliže nejsou předcházeny negací (/), negované jsou nastaveny do hodnoty log.0. Ostatní signály zůstanou nedefinované x nebo v předcházející hodnotě. Když příkaz SETF ovlivňuje výstupy, potom je vygenerován vektor a vypočteny všechny výstupní stavy odpovídající novému nastavení. Vnitřně jsou události detekovány a vyhodnocovány. U asynchronních obvodů může být jedním příkazem SETF generováno několik vektorů. Ačkoliv nebudou zobrazeny, simulátor generuje vektory tak dlouho, dokud nedosáhne stabilního stavu. Je-li systém chybný, je stabilizován po deseti interních krocích a je předpokládáno, že je zacyklen (osciluje). Chyba je hlášena a simulace se zastavuje.

CLOCKF hodinový signál - vygeneruje na hodinovém signálu jeden hodinový impuls $0 \rightarrow 1 \rightarrow 0$. V příkazu CLOCKF mohou být použity jenom hodinové signály, jakýkoliv jiný vstup je v tomto příkazu zakázaný. Užití SETF inicializuje hodinový signál do log.0, a teprve potom může být použit příkaz CLOCKF.

Pro několikanásobné opakování je možné využít operace cyklu

```
FOR i:= 0 TO x    DO
        BEGIN
        CLOCKF CLK
        END
```

nebo cyklu vytvořeného příkazem WHILE

WHILE ukončující podmínka (např. Q14=1) **DO**

```
        BEGIN
        tělo cyklu
        END
```

Rozšíření simulačních možností umožňuje podmíněné vykonání některých operací

```
IF podmínka   THEN BEGIN
                   operace pro podmínka=1
                   END
      ELSE         BEGIN
                   operace pro podmínka=0
                   END
```

Pro kontrolu očekávaných hodnot výstupů můžeme využít příkaz

CHECK očekávané stavy proměnných (např. CHECK Q6 /Q7 - předpokládá, že Q6=1 a Q7=0. V případě, že nedojde ke shodě výstupů s těmito očekávanými hodnotami, bude neshoda indikována znakem ?).

Celá simulační sekvence je zakončena klíčovým slovem **TRACE_OFF**.

PALASM2 umožňuje 5 základních funkcí v pořadí, jak jsou uvedeny:

- Kontrola syntaxe vstupního souboru
- Rozšíření vstupních rovnic
- Minimalizace vstupních rovnic
- Překlad do souboru a generování výstupu JEDEC
- Simulace návrhu

Výstupy překladače jsou soubory: Propojek - JMENO.XPT a JEDEC - JMENO.JED

Výstupy simulátoru jsou soubory: Vývojový soubor - JMENO.HST, trasovací soubor - JMENO.TRF, JEDEC test data - JMENO.JDC

Po spuštění simulace program uloží výsledky do dvou souborů: jméno.HST a jméno.TRF, kde soubor s příponou .HST zobrazuje průběhy všech signálů integrovaného obvodu, soubor s příponou .TRF jen těch, které byly specifikovány v příkazu TRACE_ON. Přednastavení registrů po inicializaci hodin a vstupů je lepší provádět příkazem SETF. Se strukturou souboru JEDEC jsme se seznámili v předcházejícím dodatku, zde si ukážeme ještě strukturu souboru s příponou .XPT, který dává daleko lepší představu o programovaných

propojkách ve vnitřní struktuře pole než soubor JEDEC. V tomto souboru jsou písmenem ′X′ označeny neprogramované propojky a znakem ′-′ programované propojky.

```
PALASM XPLOT, V2.23D - MARKET RELEASE (2-14-89)
(C) - COPYRIGHT ADVANCED MICRO DEVICES INC., 1989

Title    : Prioritni koder          Author  : Skalicky
Pattern  : 01.                     Company : CVUT FEL
Revision : 01_def                  Date    : 9.4.1994

PAL20L8
PKODER

                11 1111 1111 2222 2222 2233 3333 3333
    0123 4567 8901 2345 6789 0123 4567 8901 2345 6789

  0 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
  1 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
  2 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
  3 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
  4 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
  5 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
  6 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
  7 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

  8 ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
  9 -X-X -X-- -X-- -X-- -X-- -X-- -X-- ---- ---- ----
 10 ---- ---- ---- ---- ---- ---- ---- X--- ---- ----
 11 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 12 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 13 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 14 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 15 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

 16 ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
 17 ---- ---- ---- ---- ---- ---- ---- ---- X-X- X--X
 18 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 19 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 20 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 21 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 22 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 23 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

 24 ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
 25 ---- ---- ---- ---- ---- ---- ---- ---- -XX- X--X
 26 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 27 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 28 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 29 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 30 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
```

```
31 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

32 ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
33 ---- ---- ---- ---- ---- ---- ---- ---- X-X- -X-X
34 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
35 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
36 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
37 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
38 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
39 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

40 ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
41 ---- ---- ---- ---- ---- ---- ---- ---- -XX- -X-X
42 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
43 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
44 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
45 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
46 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
47 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

48 ---- ---- ---- ---- ---- ---- ---- -X-- ---- ----
49 ---- ---- ---- -X-- -X-- -X-- -X-- ---- ---- ----
50 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
51 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
52 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
53 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
54 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
55 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

56 ---- ---- ---- ---- ---- ---- ---- -X-- ---- ----
57 ---- ---- ---- X--- ---- -X-- -X-- ---- ---- ----
58 ---- -X-- -X-- ---- ---- -X-- -X-- ---- ---- ----
59 ---- ---- ---- ---- X--- -X-- -X-- ---- ---- ----
60 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
61 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
62 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
63 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

64 ---- ---- ---- ---- ---- ---- ---- -X-- ---- ----
65 ---- ---- ---- X--- -X-- ---- -X-- ---- ---- ----
66 -X-- ---- -X-- ---- -X-- ---- -X-- ---- ---- ----
67 ---- X--- -X-- ---- -X-- ---- -X-- ---- ---- ----
68 ---- ---- ---- ---- ---- X--- -X-- ---- ---- ----
69 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
70 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
71 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

72 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
73 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
74 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
75 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
76 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
77 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
```

```
 78 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
 79 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

TOTAL FUSES BLOWN:   825
```

Literatura

[1] J.P.Perrin, M.Denouette - Logické systémy I,II, SNTL 1972

[2] J.Podlešák - Elektronické obvody III, Skriptum ČVUT Praha 1988

[3] P.Skalický - Elektronické obvody III - cvičení, Skriptum ČVUT Praha 1987

[4] Texas Instrument - Aplications Handbook, 1993

[5] MOTOROLA Fast and LS TTL Data Book, 1991

[6] XILINX The Programmable Logic Data Book, 1993

[7] Richard.L.Rudell - Multiple-Valued Minimalization for PLA Optimization

[8] Art Goldstein - Field Programmable Controlers Am29CPL100, AMD 1988

[9] Advanced Micro Devices Am29CPL154, Data Sheet, 1989

[10] Advanced Micro Devices Manual PALASM2, 1987

[11] Texas Instruments ALS and AS TTL Data Book, 1989

[12] Texas Instruments Advanced BiCMOS Technology, 1991

[13] V.Lojík - Číslicová a impulzová technika II, Skriptum ČVUT, Praha 1984

[14] Jiří Adámek - Kódování, SNTL 1989

[15] B.Lampe, G.Jorke, N.Wengel - Algorithmen der mikrorechentechnik, VEB Verlag Technik, Berlín, 1983

[16] A.D.Friedman, P.R.Melon - Teorie a návrh logických obvodů, SNTL 1983

Výsledky

Kapitola 1.3

Příklad 1.3.1

Číslo		Vyjádření		
		z	mantisa	exponent
$1{,}000.10^{0}$	$0{,}500000.2^{1}$	0	1000000000	0000001
$8{,}000.10^{0}$	$0{,}500000.2^{4}$	0	1000000000	0000100
$0{,}000.10^{0}$	$0{,}000000.2^{0}$	0	0000000000	0000000
$-1{,}750.10^{0}$	$-0{,}875000.2^{1}$	1	0010000000	0000001
$1{,}250.10^{2}$	$0{,}976562.2^{7}$	0	1111101000	0000111
$-1{,}250.10^{2}$	$-0{,}976562.2^{7}$	1	0000011000	0000111
$-7{,}812.10^{-2}$	$-0.624960.2^{-5}$	1	0110000001	1111011

Kapitola 2.2

Příklad 2.2.1

$$x_4.\left(1+x_3.x_2+x_1\right)+\overline{x}_4.x_3.\left(1+x_2\right)+x_2.\overline{x}_1=x_4+\overline{x}_4.x_3+x_2.\overline{x}_1=$$

$$=x_4.\left(x_3+\overline{x}_3\right)+\overline{x}_4.x_3+x_2.\overline{x}_1=x_4+x_3.\left(x_4+\overline{x}_4\right)+x_2.\overline{x}_1=x_4+x_3+x_2.\overline{x}_1$$

Příklad 2.2.2

a) $$\overline{x}_1+x_2.\left(x_3+\overline{x}_4\right).\left(x_5+\overline{x}_6\right)$$

b) $$\left(\overline{x}_1+x_2+\overline{x}_3\right).x_1.\overline{x}_2.\left(\overline{x}_4+x_2.\left(\overline{x}_1+\overline{x}_2+x_4\right)\right)=x_1.\overline{x}_2.\overline{x}_3.\overline{x}_4$$

c) $$\overline{x}_1.\left(x_2+x_3+\overline{x}_4+\left(x_1+x_4\right).\left(\overline{x}_2+x_3.\overline{x}_1\right)\right)=\overline{x}_1$$

Příklad 2.2.3

$$\alpha.\beta.(x+\overline{x})+\alpha.\overline{x}.\left(\beta+\overline{\beta}\right)+\beta.x.(\alpha+\overline{\alpha})=\alpha.\beta.x+\alpha.\beta.\overline{x}+\alpha.\overline{x}.\beta+\alpha.\overline{x}.\overline{\beta}+\beta.x.\alpha+\beta.x.\overline{\alpha}=$$

$$= \alpha.\bar{x}.\left(\beta + \bar{\beta}\right) + \beta.x.(\alpha + \bar{\alpha}) = \alpha.\bar{x} + \beta.x$$

Příklad 2.2.4

$$\overline{\overline{x_1.x_2 + x_2.x_3 + \bar{x}_2.\bar{x}_3}} = \overline{\left(\bar{x}_1 + \bar{x}_2\right).\left(\bar{x}_2 + \bar{x}_3\right).\left(x_2 + x_3\right)} = \overline{\bar{x}_1.\bar{x}_3.x_2 + \bar{x}_2.x_3} =$$

$$= \left(x_1 + \bar{x}_2 + x_3\right).\left(x_2 + \bar{x}_3\right)$$

Příklad 2.2.5

a)

$$x_1.x_3.\bar{x}_4 + \bar{x}_1.\bar{x}_2.x_3 + \bar{x}_1.x_3.x_4 + \bar{x}_2.x_3.x_4 = x_1.x_3.\bar{x}_4 + \bar{x}_2.x_3.\left(\bar{x}_1 + x_4 + 1\right) + \bar{x}_1.x_3.x_4 =$$

$$= \bar{x}_2.x_3 + x_1.x_3.\bar{x}_4 + \bar{x}_1.x_3.x_4$$

b)

$$\left(x_1 + \bar{x}_2 + \bar{x}_3\right).\left(x_1 + x_2 + \bar{x}_3 + \bar{x}_4\right).\left(\bar{x}_1 + x_2 + x_3 + x_4\right).\left(x_2 + \bar{x}_3 + x_4\right) =$$

$$= \left(x_1 + \bar{x}_3\right).\left(\bar{x}_1 + x_2 + x_3 + x_4\right).\left(x_2 + \bar{x}_3 + x_4\right) = \left(x_1 + \bar{x}_3\right).\left(\bar{x}_1 + x_2 + x_4\right)$$

Příklad 2.2.6

$$f_1 = \bar{x}_2.\bar{x}_4 + \bar{x}_2.x_3 + \bar{x}_1.x_3.x_4 = \left(\bar{x}_2 + x_4\right).\left(x_3 + \bar{x}_4\right).\left(\bar{x}_1 + \bar{x}_2\right)$$

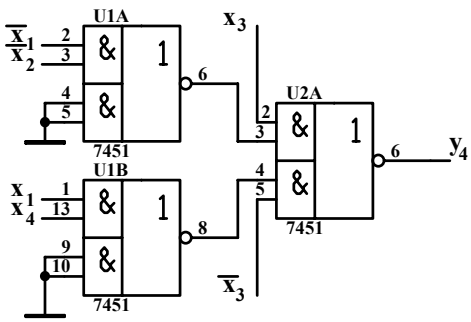
$$f_2 = x_2.x_4.x_5 + x_1.x_2.x_4 + \bar{x}_1.\bar{x}_2.\bar{x}_4.\bar{x}_5 =$$

$$= \left(\bar{x}_2 + \bar{x}_4\right).\left(x_1 + \bar{x}_4\right).\left(\bar{x}_2 + x_5\right).\left(\bar{x}_1 + x_4 + x_5\right).\left(x_2 + x_4 + \bar{x}_5\right)$$

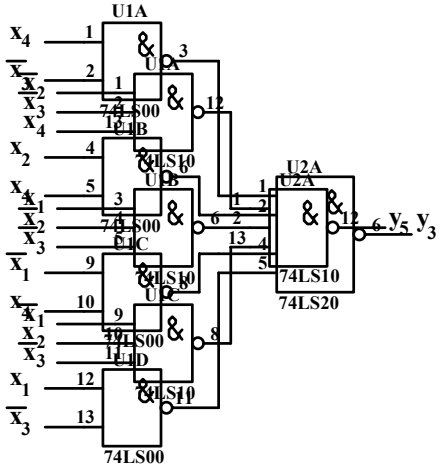
Kapitola 2.5

Příklad 2.5.1

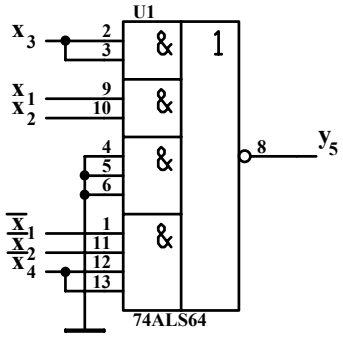
$$y_1 = x_5 + x_2.x_4 + \bar{x}_1.\bar{x}_3 \qquad y_2 = \bar{x}_1.x_2.x_5.x_6.\left(\bar{x}_3 + \bar{x}_4\right)$$



Příklad 2.5.2

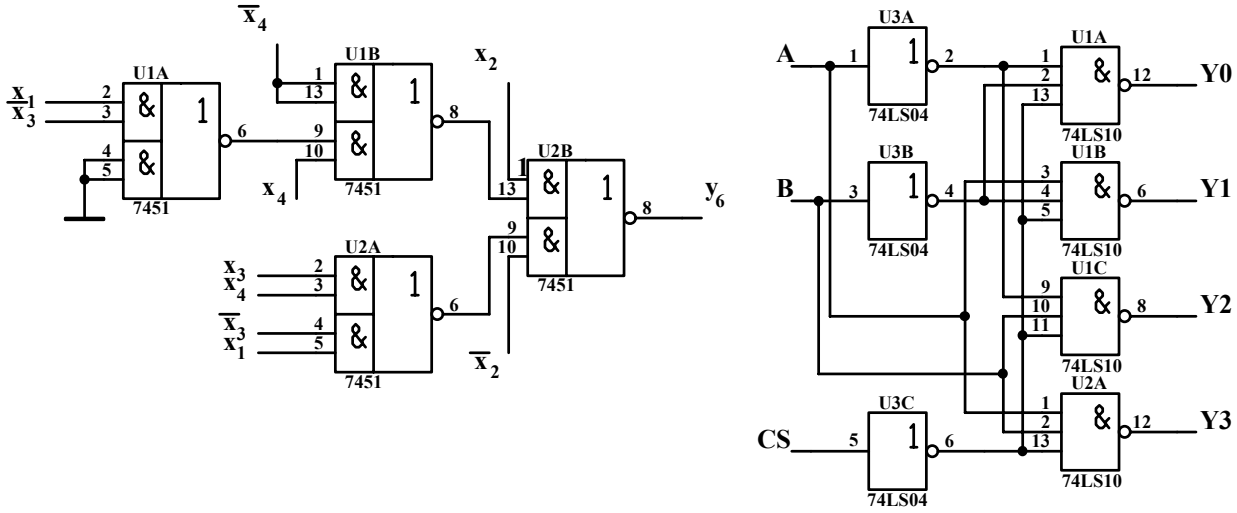


Příklad 2.5.3



Příklad 2.5.4

Příklad 2.5.6

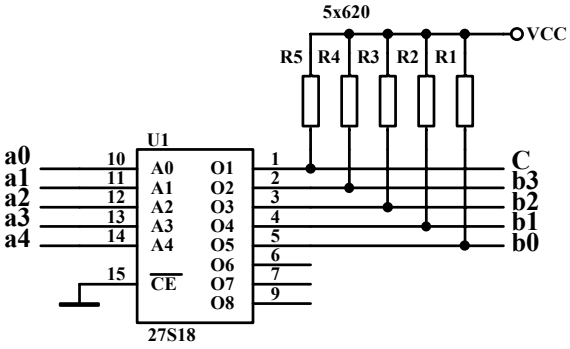


Příklad 2.5.5

$$y_7 = x_1.x_2.x_5 + \overline{x}_1.x_2.x_4 + x_2.x_4.\overline{x}_3 + x_2.\overline{x}_3.x_5 + x_2.x_3.\overline{x}_4.\overline{x}_5 + \overline{x}_1.\overline{x}_2.\overline{x}_3.\overline{x}_4.\overline{x}_5 + \overline{x}_1.\overline{x}_2.x_3.\overline{x}_4.x_5$$

Příklad 2.5.7

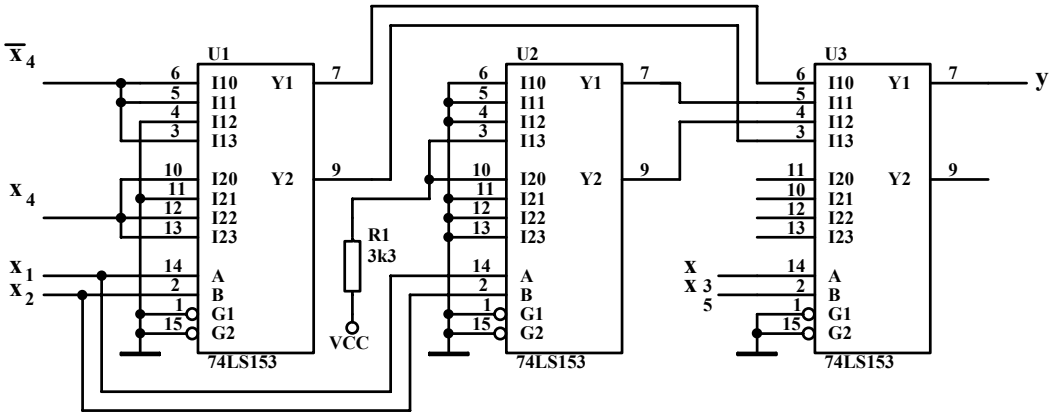
Adresa	00	01	02	03	04	05	06	07
00	10	10	10	01	10	02	03	10
08	10	04	05	10	06	10	10	10
10	10	08	09	10	00	10	10	10
18	07	10	10	10	10	10	10	10



Příklad 2.5.8

$$Y_0 = I_3 + I_1 . \bar{I}_2 \qquad\qquad Y_1 = I_2 + I_3$$

Příklad 2.5.9



Kapitola 3.6

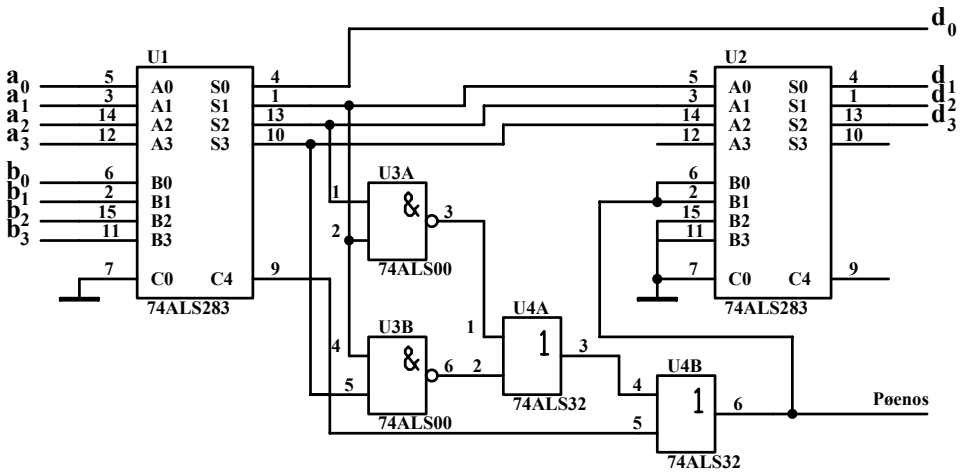
Příklad 3.6.1.

Zrušte spojení S/O se vstupem c_0 obvodu U1, vstup c_0 obvodu U1 propojte s výstupem c_4 obvodu U2.

Příklad 3.6.2.

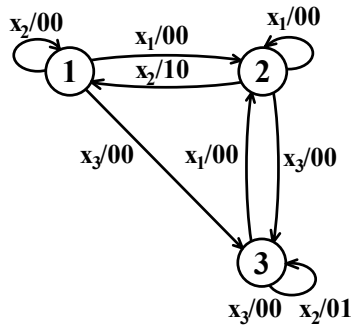
64 bitů, 5xAS282, 16xAS283, 24ns

Příklad 3.6..3



Kapitola 4.2

Příklad 4.2.1



z^i	x_1	x_2	x_3
1	2/00	1/00	3/00
2	2/00	1/10	3/00
3	2/00	3/01	3/00

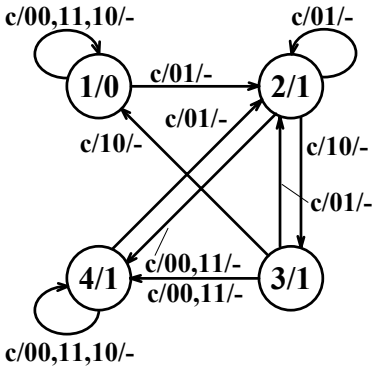
z^{i+1} / y_1y_2

Příklad 4.2.2

z^i	x_1	x_2
1	2/0	1/0
2	3/0	1/0
3	4/0	1/0
4	7/0	5/0
5	2/0	6/0
6	2/0	6/1
7	7/0	1/0

z^{i+1} / y

Příklad 4.2.3



	x_1x_2				
z^i	00	01	11	10	y
1	1	2	1	1	0
2	4	2	4	3	1
3	4	2	4	1	1
4	4	2	4	4	1

Příklad 4.2.4

x_2x_1 0 0	0 1	1 1	1 0	y
①	6	-	3	0
②	6	-	3	1
1	-	8	③	0
2	-	8	④	1
1	⑤	7	-	0
2	⑥	7	-	1
1	5	⑦	3	0
2	6	⑧	4	1

Příklad 4.2.5

a) 1 - 12 ≡ A
2 - 7 ≡ B
4 - 6 ≡ C
5 - 10 ≡ D

b) 1 - 12 ≡ A
3 - 8 ≡ B
6 - 11 ≡ C
7 - 9 ≡ D

Příklad 4.2.6

a) a=00, b=11 , c=10 , d=01

z^i \ x_1x_2	0 0	0 1	1 1	1 0
a	**1**	2	5	**8**
b	**3**	4	**5**	7
c	1	**2**	**6**	7
d	1	**4**	5	**7**

z_1z_2 \ x_1x_2	0 0	0 1	1 1	1 0
00	**00**	10	01	**00**
01	00	**01**	11	**01**
11	**11**	01	**11**	01
10	11	**10**	**10**	11

$z_1^{i+1}z_2^{i+1}$

$a = [000,111]$,$b = [100,011]$, $c = [110,001]$

$d = [010,101]$

b) a=00, b=11 , c=10 , d=01

$z_3^iz_2^iz_1^i$ \ x_1x_2	0 0	0 1	1 1	1 0
0 0 0	**000**	001	100	**000**
0 0 1	000	**001**	**001**	101
0 1 1	**011**	010	**011**	010
0 1 0	000	**010**	011	**010**
1 1 0	100	**110**	**110**	010
1 1 1	**111**	110	011	**111**
1 0 1	111	**101**	100	**101**
1 0 0	**100**	101	**100**	101

$z_3^{i+1}z_2^{i+1}z_1^{i+1}$

z_1z_2 \ x_1x_2	0 0	0 1	1 1	1 0
00	**00**	**00**	10	10
01	00	00	11	**01**
11	10	**11**	**11**	01
10	**10**	11	**10**	**10**

$z_1^{i+1}z_2^{i+1}$

Kód Jednoznačný i jednorázový

Příklad 4.2.7

a) $z_1^{i+1} = z_1.\overline{x} + z_2.x + z_1.\overline{z}_2 \qquad z_2^{i+1} = z_1.\overline{x} + z_2.x + z_1.z_2 \qquad y = z_2$

b) $R_1 = \overline{z}_2 + \overline{x} \qquad S_1 = z_2 + \overline{x} \qquad R_2 = z_1 + x \qquad S_2 = \overline{z}_1 + x$

Příklad 4.2.8

Kódování 1 = 00 , 2 = 01 , 3 = 11 , 4 = 10

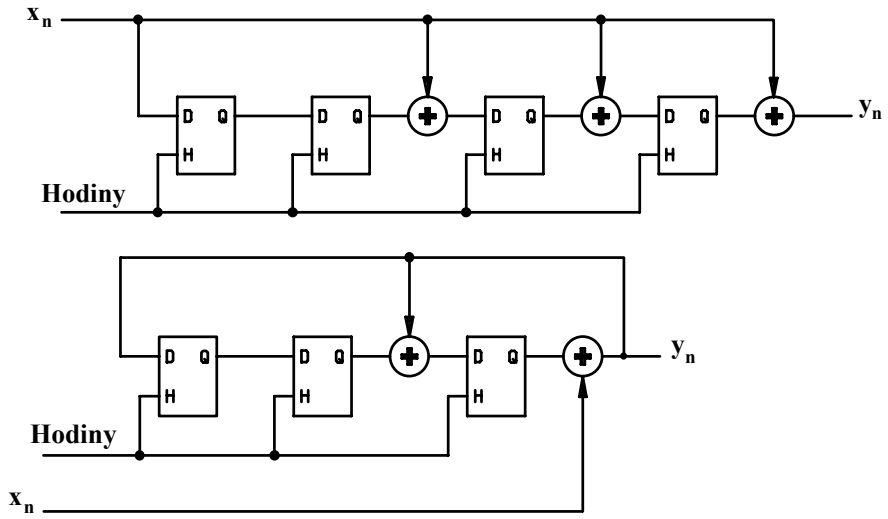
$T_1 = z_1.x_1.x_2 + z_1.\overline{x}_1.\overline{x}_2 + z_1.z_2.x_2 + \overline{z}_1.x_1.\overline{x}_2 \qquad T_2 = z_1.\overline{z}_2.\overline{x}_1 + z_1.\overline{z}_2.x_2 + z_1.\overline{x}_1.x_2 + z_2.x_1.\overline{x}_2$

Kapitola 5.1

Příklad 5.1.1

Výstupní posloupnost na signál 1010 000 je 1110010.

Příklad 5.1.2



Příklad 5.1.3

Dekódovaná posloupnost je 1010000 a obsah registrů je nulový.

Příklad 5.1.4

Impulsní odezva = 1110011 | 1110011 ....atd.