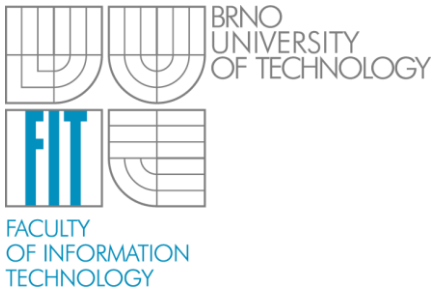
Návrh číslicových systémů (INC)

Otto Fučík

Vysoké učení technické v Brně
Fakulta informačních technologií
Božetěchova 2, 612 66 Brno



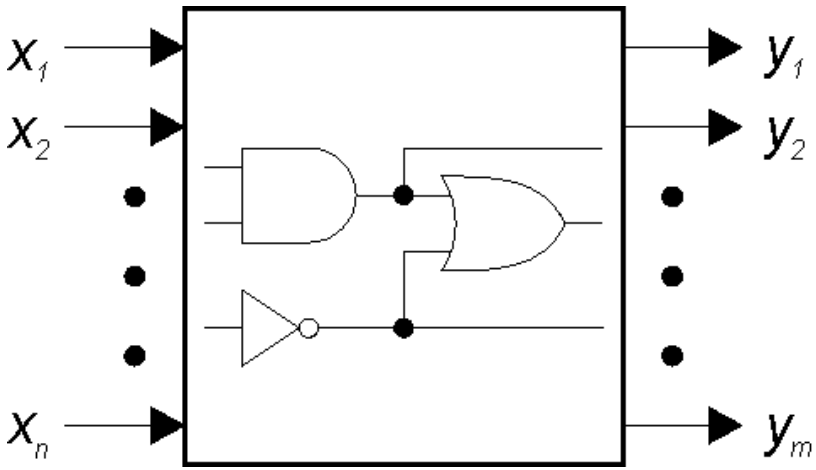
Použitá literatura

N. Frištacký, M. Kolesár, J. Kolenička a J. Hlavatý: „Logické systémy", SNTL Praha, 1986
M. Eysselt: „Logické systémy", SNTL Praha, skriptum VUT v Brně, 1985
J. F. Wakerly: „Digital Design. Principles and Practices", Prentice Hall, ISBN 0-13-769191-2, 2000
V. P. Nelson, H.T.Nagle, B.D.Carroll, J.D.Irwin: „Digital Logic Circuit Analysis & Design", ISBN 0-13-463894-8, 1995
T.L.Floyd: „Digital Fundamentals", Prentice Hall, ISBN 0-13-080850-4, 2000
D.J Smith: „HDL Chip Design", ISBN 0-9651934-3-8, 1996

Kombinační obvody

Kombinační logické obvody

- Hierarchicky uspořádaný obvod, ve kterém jednotlivé komponenty zpracovávají a mezi sebou komunikují informaci reprezentovanou v binární podobě (log. úrovně)
  - Každá komponenta má kombinační chování
  - Vstup každé komponenty je připojen pouze k jednomu výstupu předchozí komponenty nebo zdroji log. „0“ či „1“
  - Výstupy nelze spojovat
  - Pozn.: Pouze v případě tzv. montážní logiky se spojením výstupů hradel (např. s tzv. otevřeným kolektorem) realizují log. funkce - viz dále
  - Struktura neobsahuje cykly (zpětné vazby)
  - Funkční a časové chování lze odvodit z funkčního a časového chování jednotlivých komponent

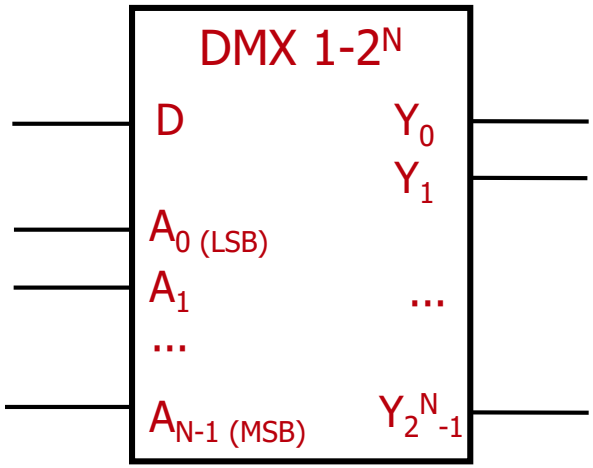


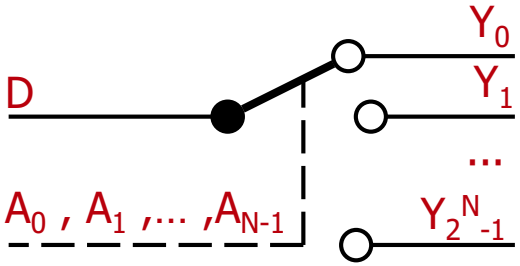
Kombinační obvody: Moduly

- Z praktického hlediska je účelné vytvářet funkční moduly, které
  - Jsou sestaveny z jednodušších komponent (log. členů, jednodušších modulů)
  - Vykonávají specifickou (často používanou) funkci
  - Slouží jako stavební bloky složitějších log. obvodů
  - Jsou ekonomické - vyrábí se ve velkých sériích
- Příklad
  - Demultiplexor
  - Dekodér
  - Multiplexor
  - Kodér
  - Sčítačka
  - Posouvač
  - Násobička, atd.

Demultiplexor (anglicky Demultiplexer)

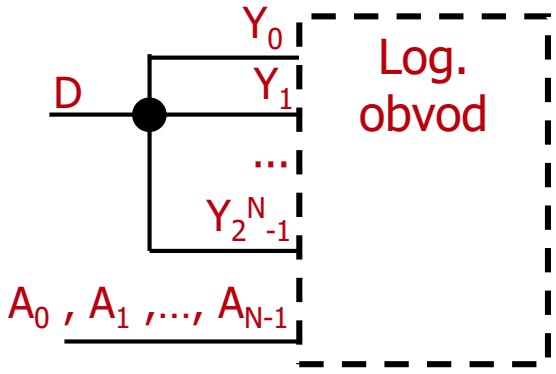
- Charakteristika
  - Demultiplexor je kombinační log. síť s jedním datovým vstupem D, N adresovými vstupy A0..A_{N-1} a 2^N výstupy Y_0..Y_{2^N-1}
  - Přenáší logickou hodnotu z datového vstupu D na jeden z 2^N výstupů, přičemž ostatní výstupy mají neaktivní log. úrovně
  - Výstup je určen binární hodnotou adresovacích vstupů
  - Funguje jako přepínač jednoho vstupu na 2^N výstupů
  - Značí se DMX 1-2^N
  - Platí $Y_i = D \cdot m_i$, kde
    - Y_i…výstup i
    - D…datový vstup
    - m_i…minterm i určený adresou A





Demultiplexor (anglicky Demultiplexer)

- Pokud se na datový vstup D převede konstanta 0 nebo 1, dostáváme tzv. dekodér s výstupy aktivními v log. 0, resp. v log. 1, viz dále
- Funkci DMX lze v některých případech nahradit propojením datového vstupu D se všemi výstupy
  - Logické obvody, které jsou připojeny na výstup demultiplexoru, nejsou „aktivovány" signály $Y_0,\ldots,Y_{2^N-1}$, ale příslušnou log. hodnotu signálu $D=Y_0,\ldots,Y_{2^N-1}$ si čtou na základě adresy $A_0,\ldots,A_{N-1}$
  - Demultiplexor tedy nepřenáší hodnotu datového vstupu na výstup, ale tato hodnota je přivedena na vstupy všech následujících obvodů, které si ji čtou samy na základě adresy



Demultiplexor 1-4

- Příklad implementace
  - Pravdivostní tabulka
  - Výraz
  - Logické značky
  - Logické schéma

Vstupy			Výstupy			
D	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	**0**
0	0	1	0	0	**0**	0
0	1	0	0	**0**	0	0
0	1	1	**0**	0	0	0
1	0	0	0	0	0	**1**
1	0	1	0	0	**1**	0
1	1	0	0	**1**	0	0
1	1	1	**1**	0	0	0

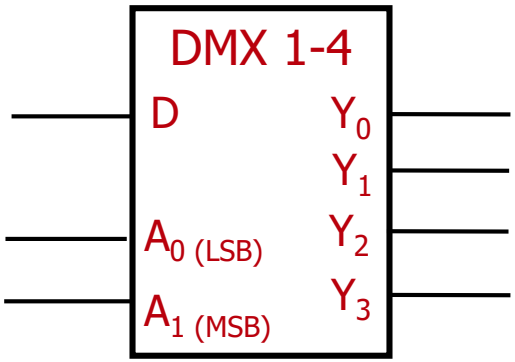
$$Yi = D \cdot m_i$$

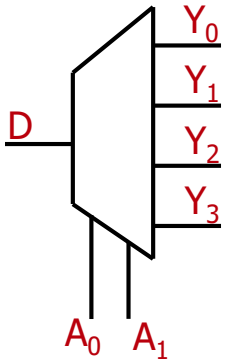
$$Y_0 = D \cdot \overline{A}_1 \cdot \overline{A}_0 = D \cdot m_0$$

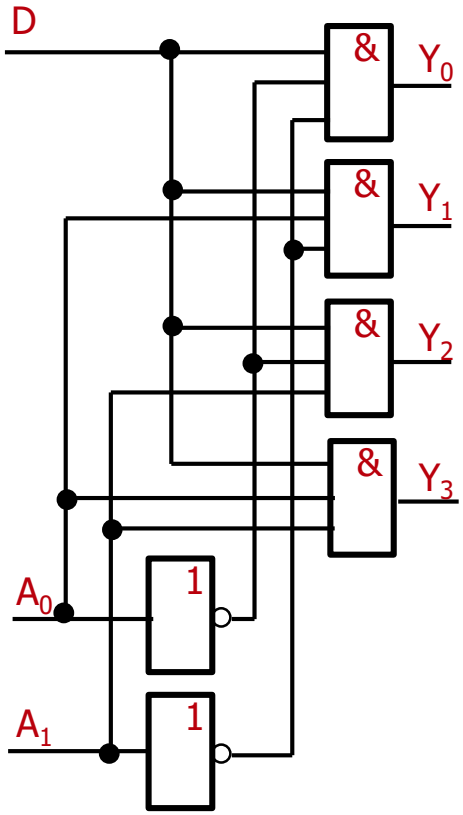
$$Y_1 = D \cdot \overline{A}_1 \cdot A_0 = D \cdot m_1$$

$$Y_2 = D \cdot A_1 \cdot \overline{A}_0 = D \cdot m_2$$

$$Y_3 = D \cdot A_1 \cdot A_0 = D \cdot m_3$$



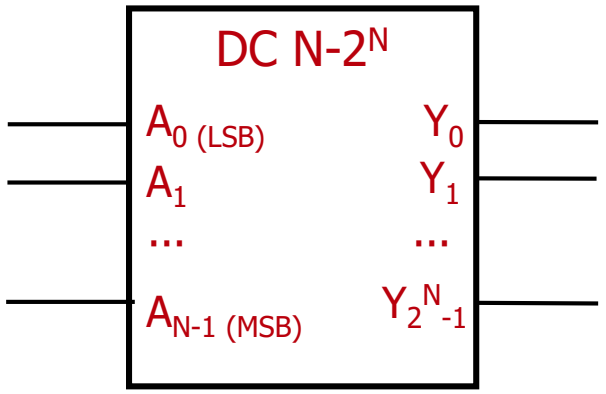




Dekodér (anglicky Decoder)

- Charakteristika
  - Pokud na datový vstup D demultiplexoru přivedeme hodnotu log. 1, dostáváme dekodér s výstupy aktivními v log. 1
  - Dekodér je kombinační log. síť s N adresovými vstupy a 2^N výstupy
  - Převádí binární kód (N bitů) na kód 1 z $M=2^N$
  - Nazýváme též N-bitový dekodér
  - Binární kód na vstupu určuje, který výstup bude platný (vždy jen jeden)
  - Některé výstupy mohou být nevyužity – např. BCD dekodér, viz dále
  - Značí se DC N-2^N (též DC 1-2^N)
- Použití
  - Adresový dekodér (paměti)
  - Generování log. funkcí
  - Dekodér pro displeje
  - atd.

$$Y_i = m_i$$



Dekodér 2-4 (1 ze 4)

- Příklad implementace pomocí demultiplexoru 1-4
  - Pozn.: m_i = minterm i

$$Y_i = m_i$$

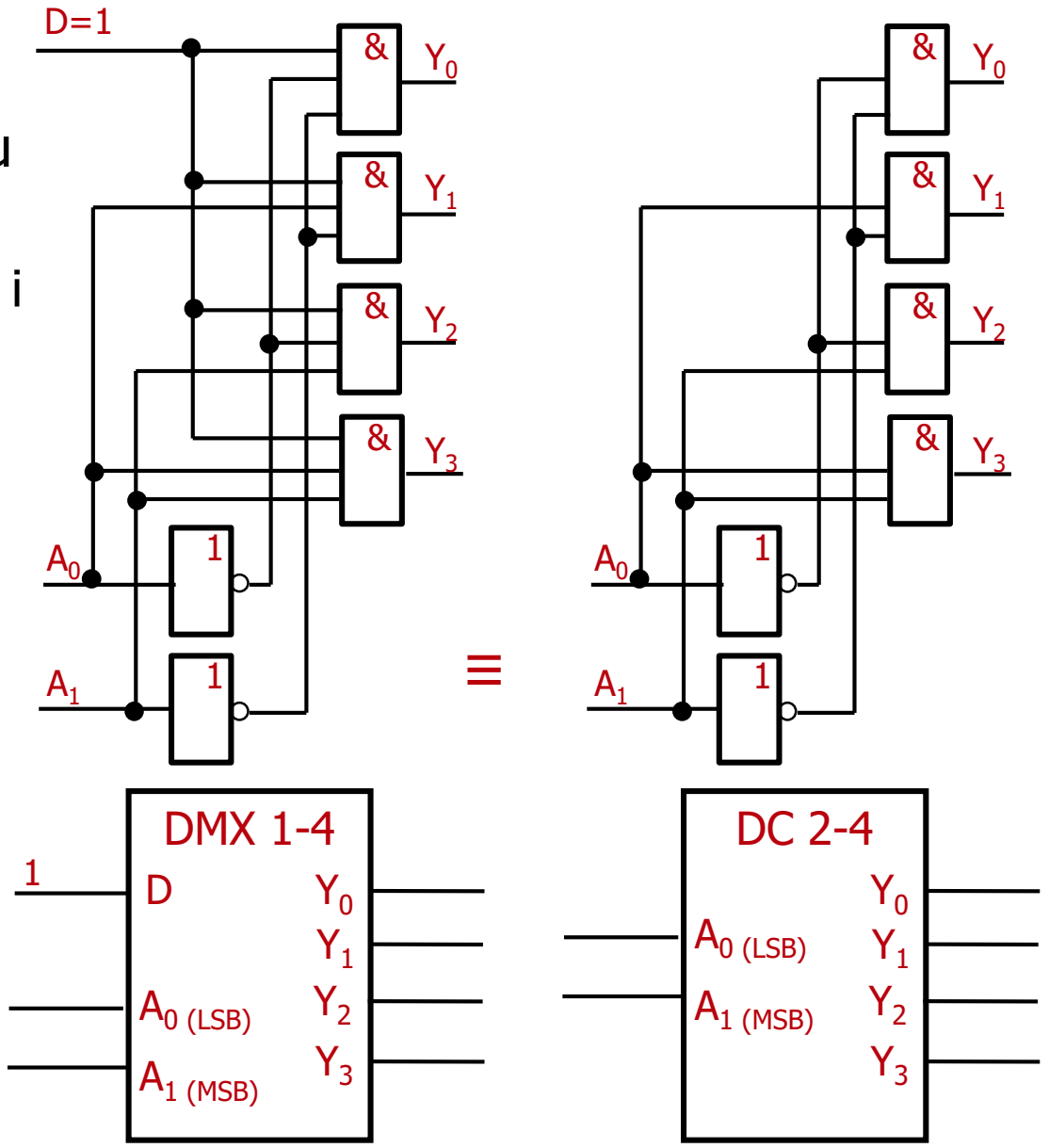
$$Y_0 = \overline{A}_1 \cdot \overline{A}_0 = m_0$$

$$Y_1 = \overline{A}_1 \cdot A_0 = m_1$$

$$Y_2 = A_1 \cdot \overline{A}_0 = m_2$$

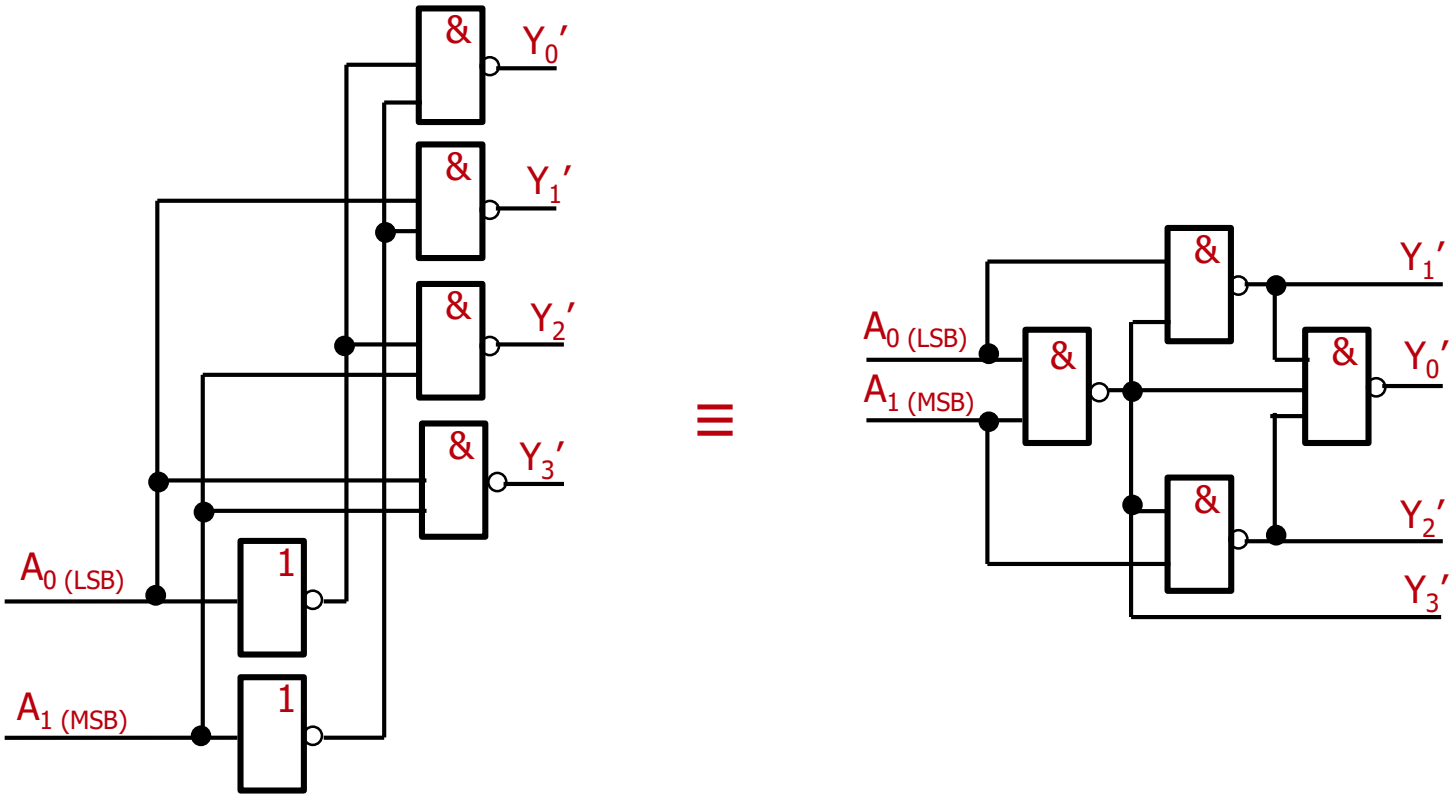
$$Y_3 = A_1 \cdot A_0 = m_3$$

Vstupy		Výstupy			
A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	**1**
0	1	0	0	**1**	0
1	0	0	**1**	0	0
1	1	**1**	0	0	0



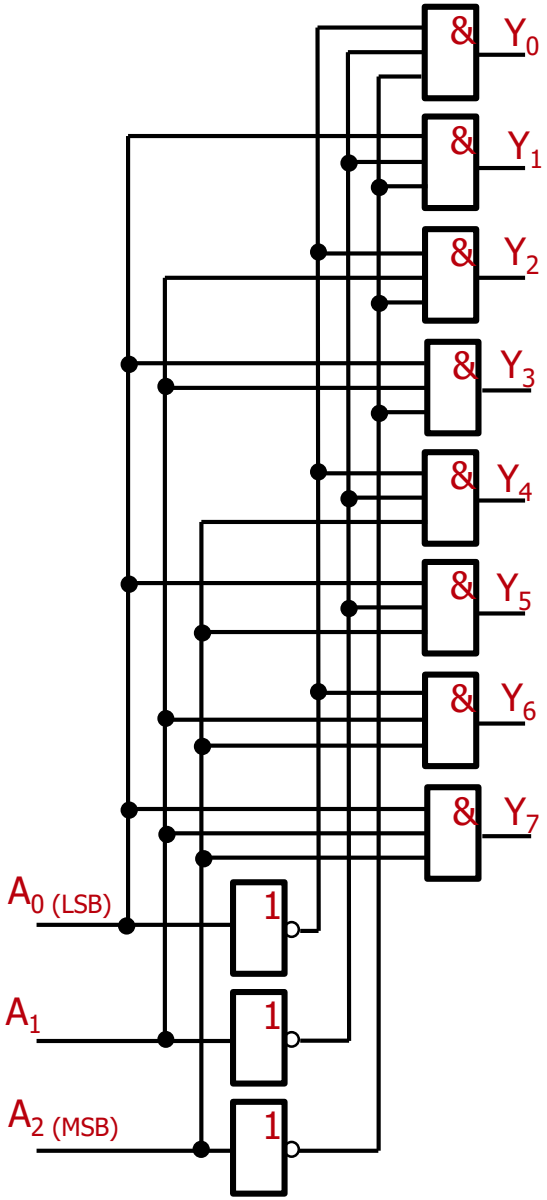
Dekodér 2-4 (1 ze 4)

- Příklady realizace
  - S výstupy aktivními v nule (použité členy INV, NAND)
  - Alternativní řešení s výstupy aktivními v nule (pouze členy NAND)



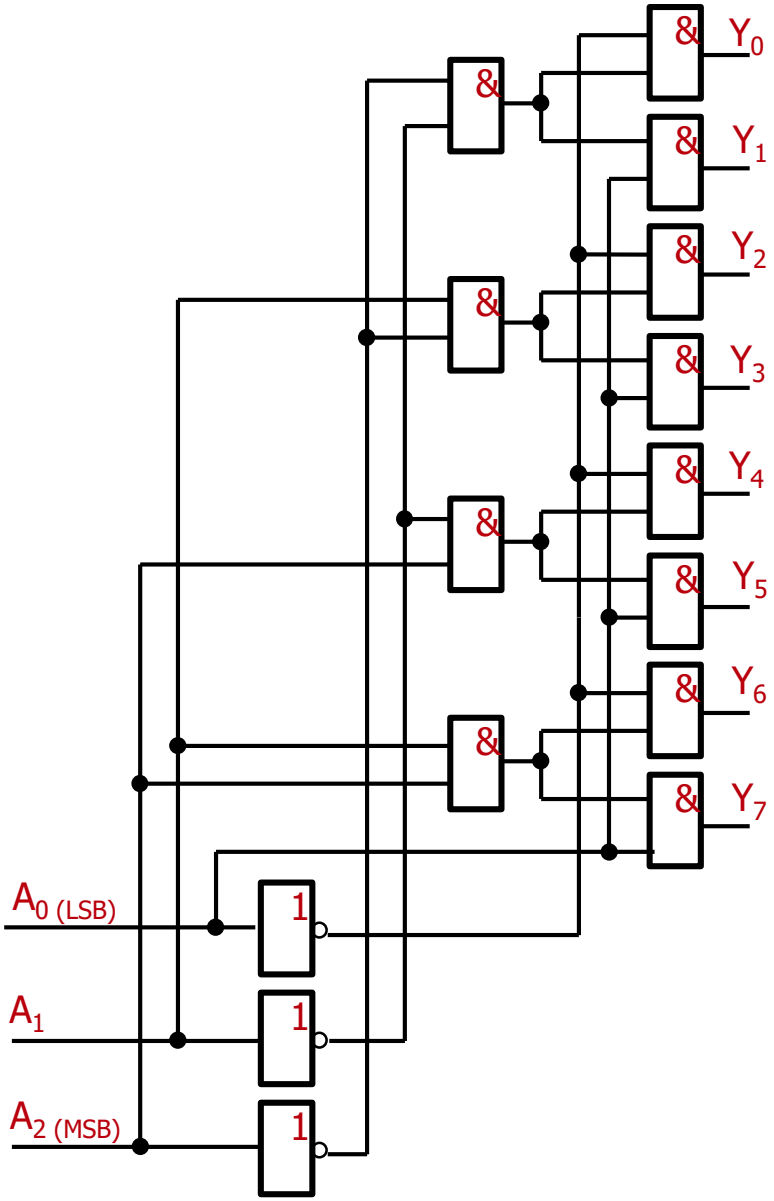
Dekodér 3-8 (1 z 8)

- Paralelní struktura
  - Pro každý výstup je třeba realizovat jeden minterm - log. člen AND s N vstupy, kde N je rovno počtu vstupů
  - V praxi jsou však počty vstupů log. členů AND limitovány technologickými možnostmi - nutno použít stromovou či maticovou strukturu



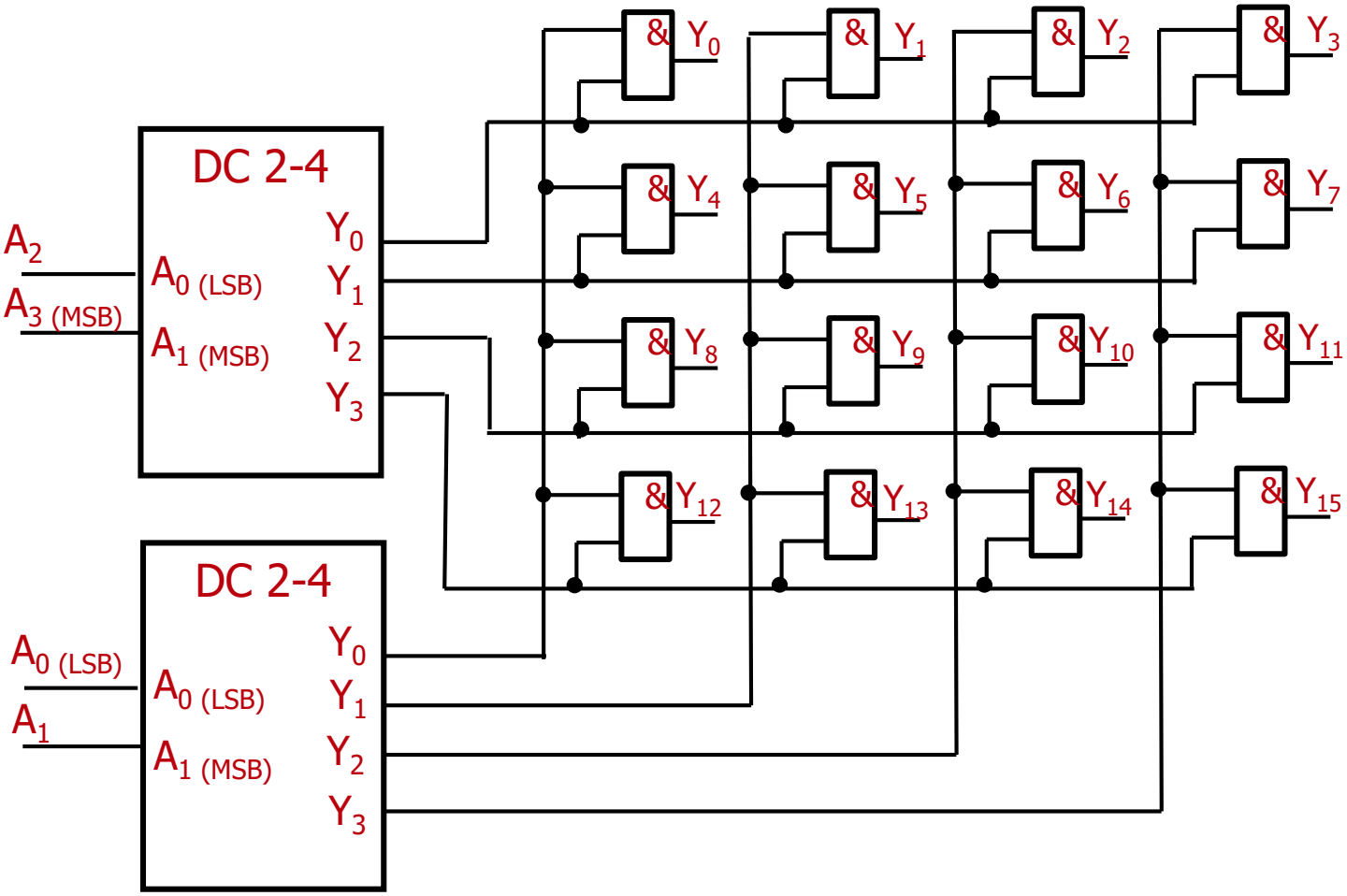
Dekodér 3-8 (1 z 8)

- Stromová struktura
  - Lze použít hradel AND s méně vstupy, než je počet vstupních proměnných
  - Větší zpoždění než paralelní struktura



Dekodér 4-16 (1 ze 16)

- Maticová struktura
  - Vhodné pro velký počtu výstupů
  - Typické použití pro paměťové dekodéry



Dekodér 3-8 (1 z 8): 74138

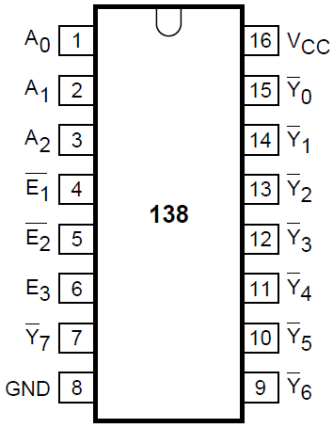
- Příklad realizace
  - Standardní IO typu 74138
  - Výstupy jsou aktivní v nule
- Povolovací vstupy umožňují
  - Využít DC jako DMX (E=D)
  - Řadit DC kaskádně – pro tvorbu rozsáhlejších DC

INPUT						OUTPUT							
$\overline{E}_1$	$\overline{E}_2$	E_3	A_0	A_1	A_2	$\overline{Y}_0$	$\overline{Y}_1$	$\overline{Y}_2$	$\overline{Y}_3$	$\overline{Y}_4$	$\overline{Y}_5$	$\overline{Y}_6$	$\overline{Y}_7$
H	X	X	X	X	X	H	H	H	H	H	H	H	H
X	H	X	X	X	X	H	H	H	H	H	H	H	H
X	X	L	X	X	X	H	H	H	H	H	H	H	H
L	L	H	L	L	L	L	H	H	H	H	H	H	H
L	L	H	H	L	L	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	L	H	H	H	H	H
L	L	H	H	H	L	H	H	H	L	H	H	H	H
L	L	H	L	L	H	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	L	H	H
L	L	H	L	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	L

Note

1. H = HIGH voltage level;
   L = LOW voltage level;
   X = don't care.

$$E = \overline{E}_1 \cdot \overline{E}_2 \cdot E_3$$



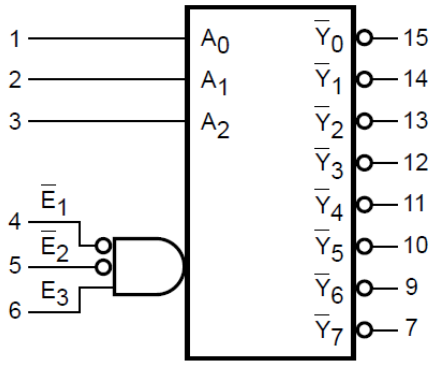
- Funkce je standardizována
  - VCC – kladný pól napájení (+5V)
  - GND – záporný pól napájení (0V)
  - Pouzdro Dual-In-Line (DIL) pouzdro (rozteč vývodů je 1/10" = 2,54 mm)

PIN	SYMBOL	DESCRIPTION
1, 2 and 3	A_0, A_1 and A_2	address inputs
4 and 5	$\overline{E}_1$ and $\overline{E}_2$	enable inputs (active LOW)
6	E_3	enable input (active HIGH)
7, 9, 10 11, 12, 13, 14 and 15	$\overline{Y}_7$ to $\overline{Y}_0$	outputs (active LOW)
8	GND	ground (0 V)
16	V_{CC}	DC supply voltage

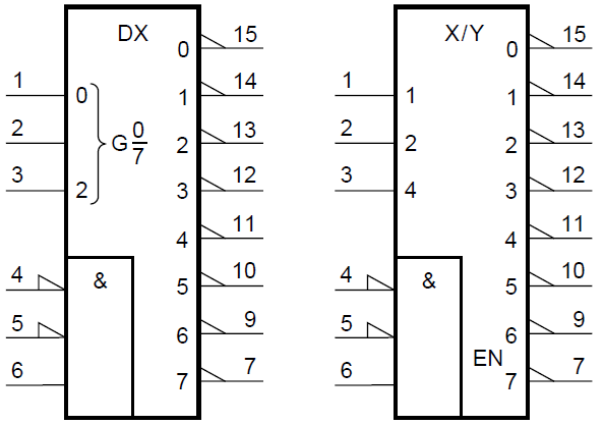
[Obrázky: Datasheet, Philips Semiconductors]

Dekodér 3-8 (1 z 8): 74138

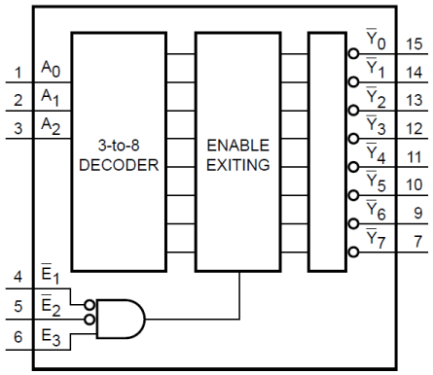
- Logický symbol



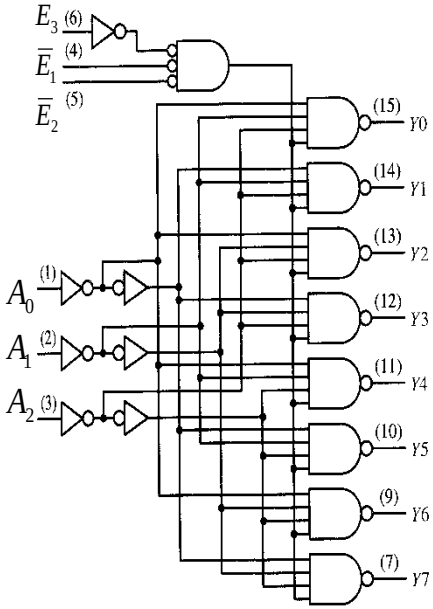
- Logický symbol dle IEC



- Funkční diagram



- Logické schéma



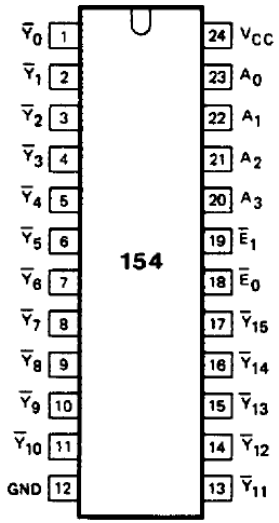
[Obrázky: Datasheet, Philips Semiconductors]

Dekodér 4-16 (1 ze 16): 74154

- Příklad realizace jako standardní IO typu 74154
  - S výstupy aktivními v nule
- Funkční tabulka
- Zapojení vývodů pouzdra
- Povolovací vstupy umožňují
  - Využít DC jako demultiplexor (E=D)
  - Řadit DC kaskádně – pro tvorbu rozsáhlejších DC
  - Podmínka Enable

$$E = \overline{E}_0 \cdot \overline{E}_1$$

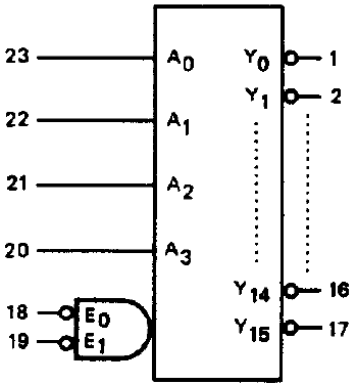
		INPUTS				OUTPUTS													
$\overline{E}_0$	$\overline{E}_1$	A_0	A_1	A_2	A_3	$\overline{Y}_0$	$\overline{Y}_1$	$\overline{Y}_2$	$\overline{Y}_3$	$\overline{Y}_4$	$\overline{Y}_5$	$\overline{Y}_6$	$\overline{Y}_7$	$\overline{Y}_8$	$\overline{Y}_9$	$\overline{Y}_{10}$	$\overline{Y}_{11}$	$\overline{Y}_{12}$	$\overline{Y}_{13}$	$\overline{Y}_{14}$	$\overline{Y}_{15}$
H	H	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	L	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	H	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	L	L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	H	L	L	L	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	H	H	L	L	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	H	L	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H
L	L	H	L	H	L	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H
L	L	L	H	H	L	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H
L	L	H	H	H	L	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H
L	L	L	L	L	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H
L	L	H	L	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H
L	L	L	H	L	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H
L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H
L	L	L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H
L	L	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H
L	L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L



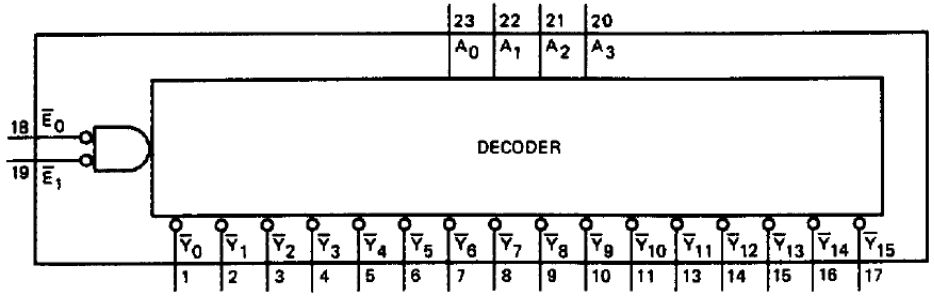
[Obrázky: Datasheet, Philips Semiconductors]

Dekodér 4-16 (1 ze 16): 74154

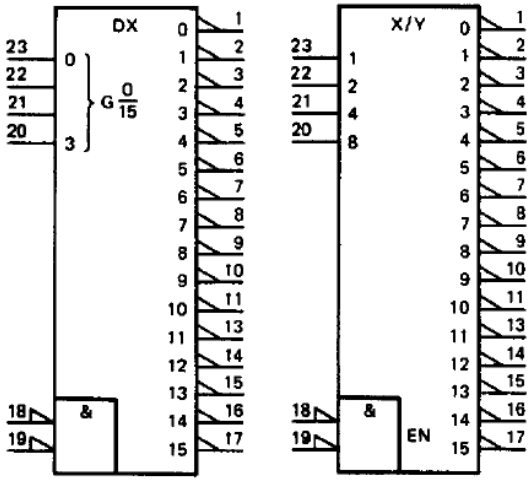
- Logický symbol



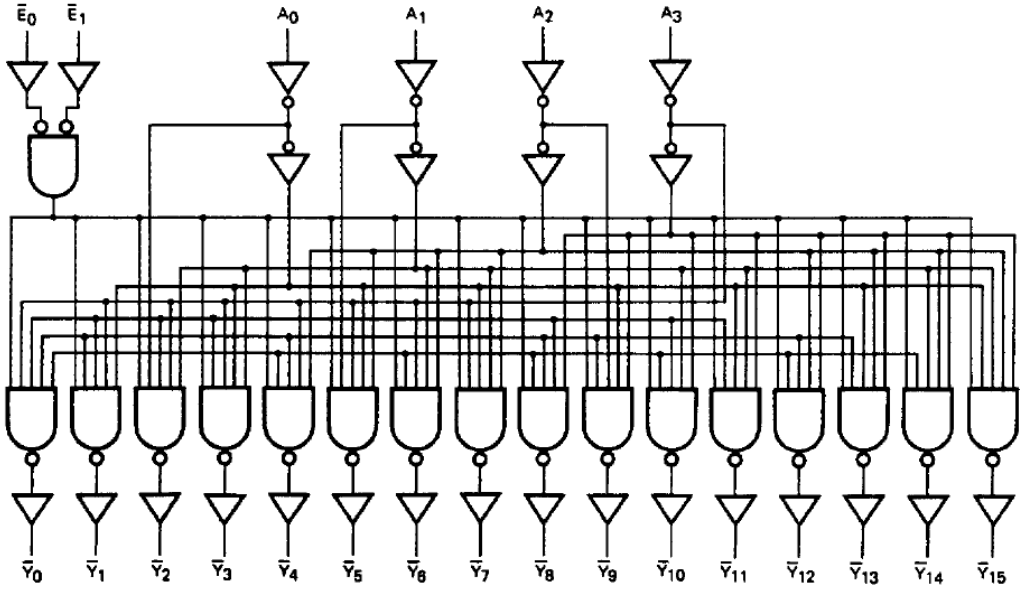
- Funkční diagram



- Logický symbol dle IEC



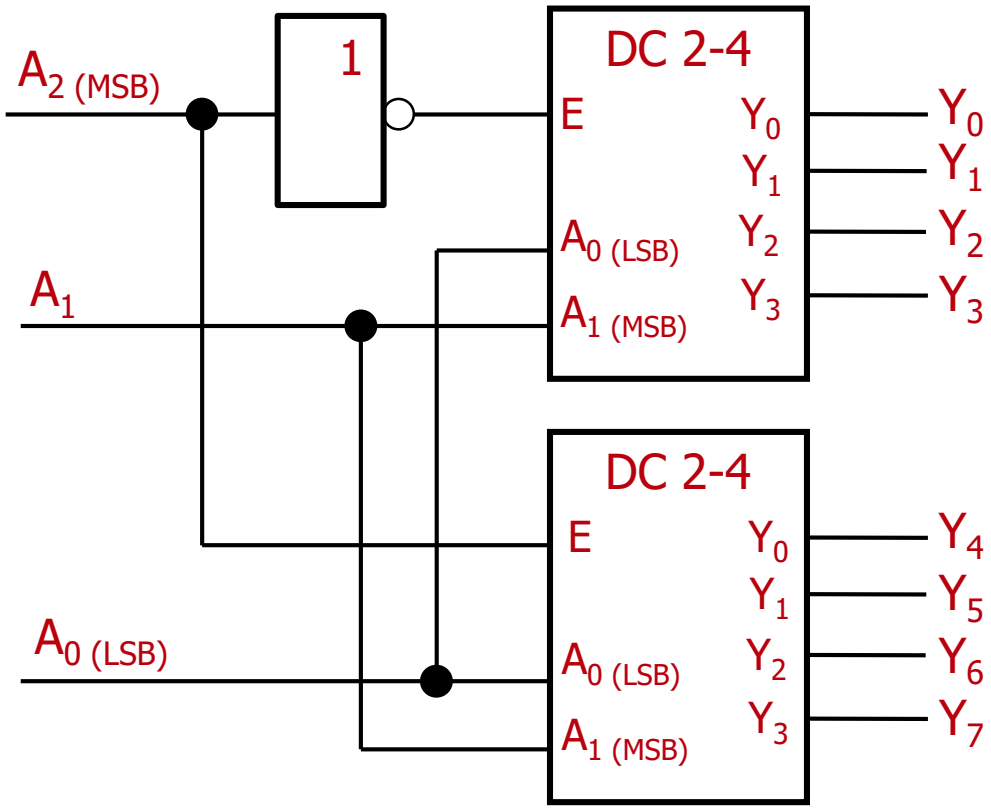
- Logické schéma



[Obrázky: Datasheet, Philips Semiconductors]

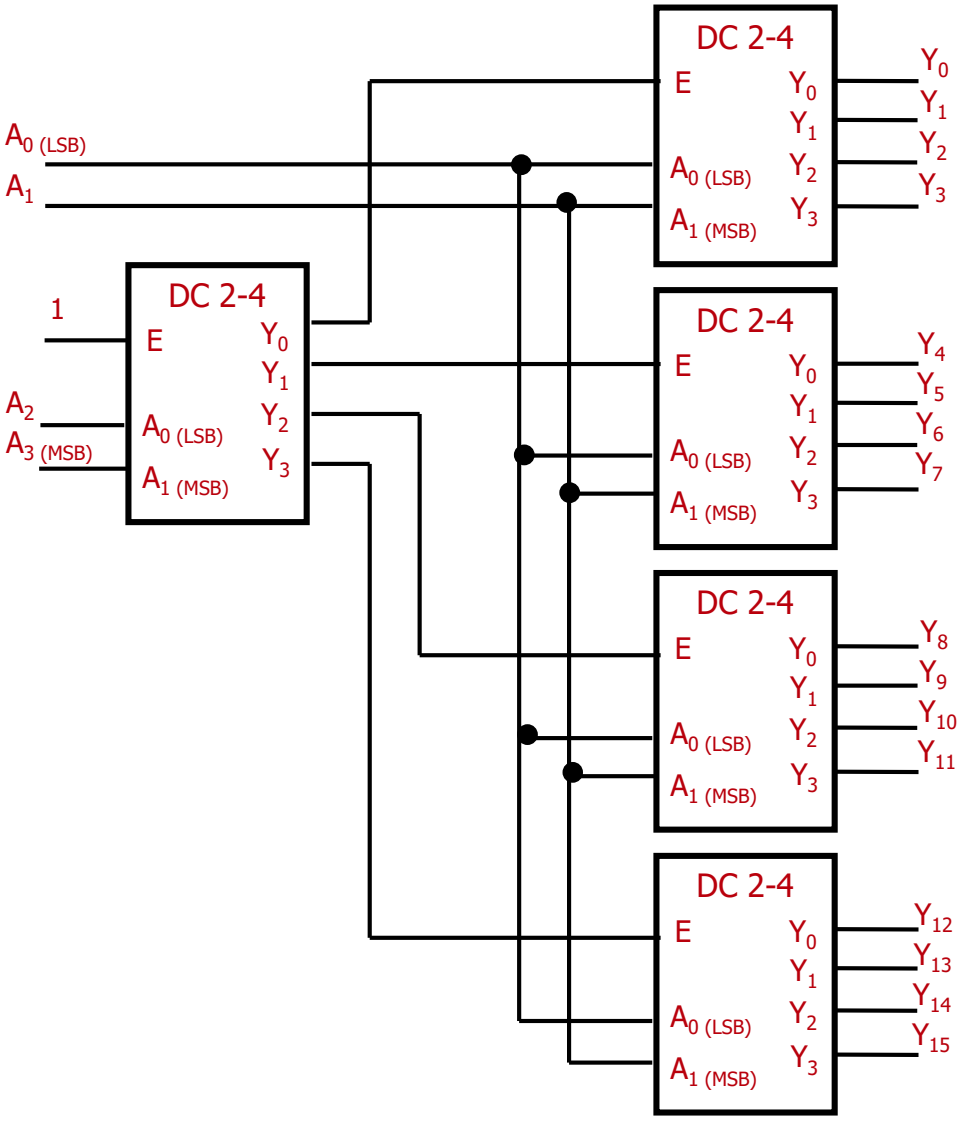
Dekodér: Více výstupů

- Dekodér s povolovacím (datovým) vstupem (= demultipexor)
  - Lze využít k budování rozsáhlejších dekodérů z jednodušších
- Ukázka postupu při návrhu zdola-nahoru
  - Z jednodušších komponent se staví složitější
- Příklad
  - Dekodér 3-8 (1 z 8) ze dvou dekodérů 2-4 (1 ze 4)



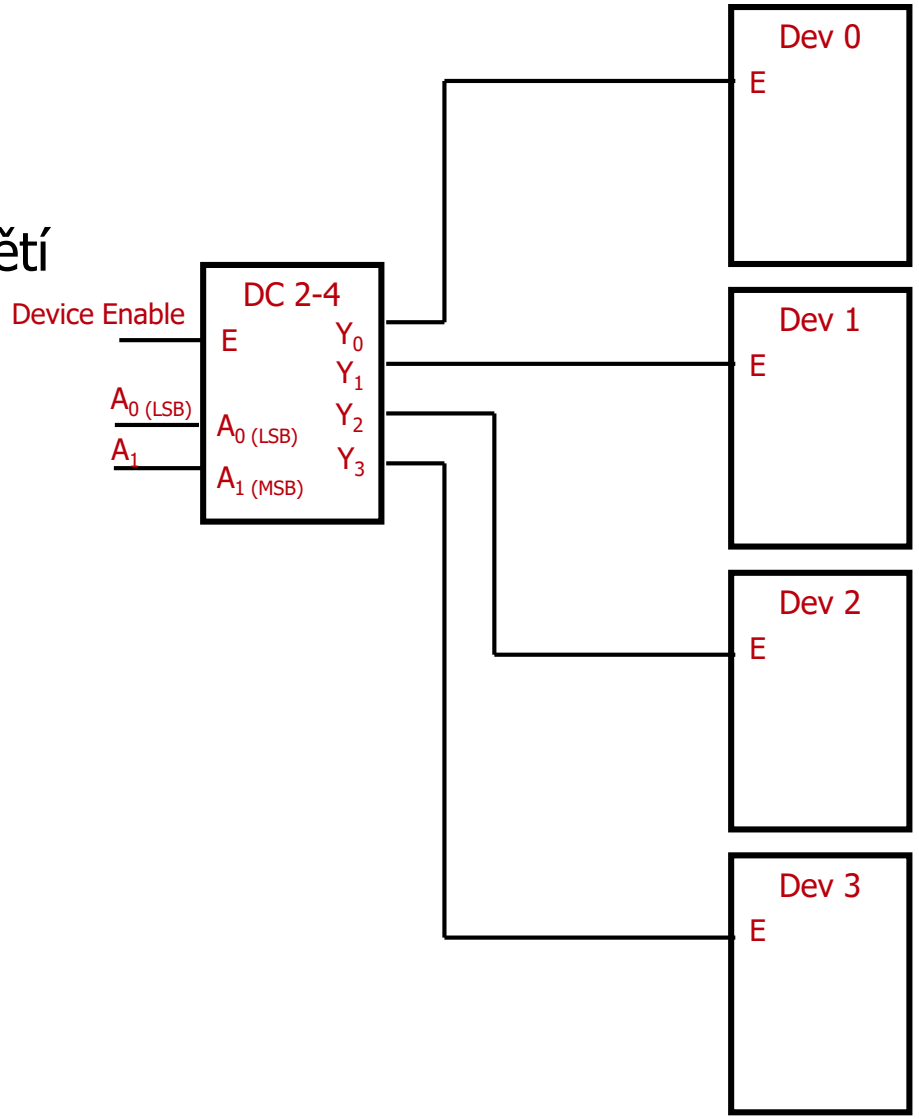
Dekodér: Více výstupů

- Příklad
  - Dekodér 4-16 (1 ze 16) ze čtyř dekodérů 2-4 (1 ze 4)



Dekodér: Využití pro dekódování adres

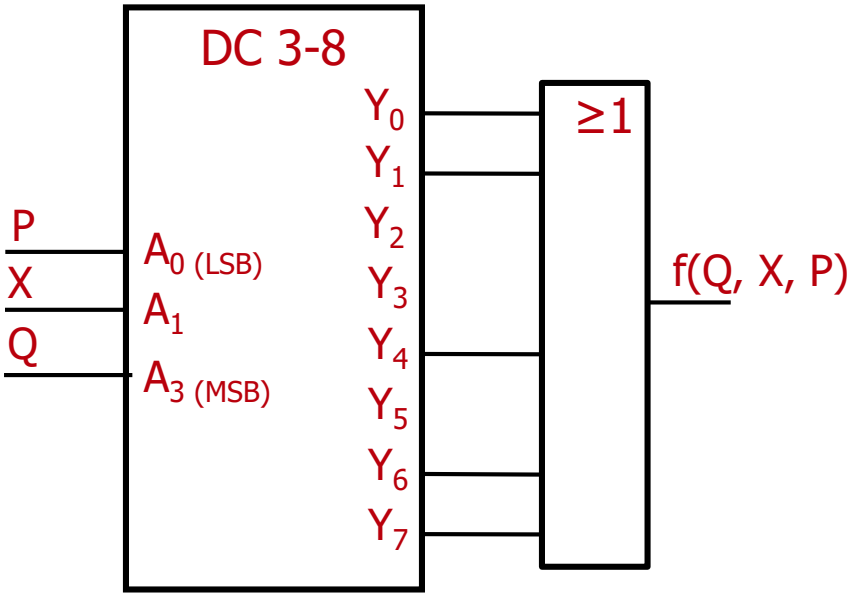
- Adresový dekodér
  - N-bitová adresa je dekódována pro výběr jednoho z 2^n vstupně-výstupních zařízení či pamětí
- Příklad
  - 4bitový dekodér umožňuje vybrat jedno ze 4 zařízení připojených např. v adresovém prostoru procesoru
  - Signál „Device Enable" povoluje výběr (povoluje) zařízení, které je určeno adresou A_1A_0

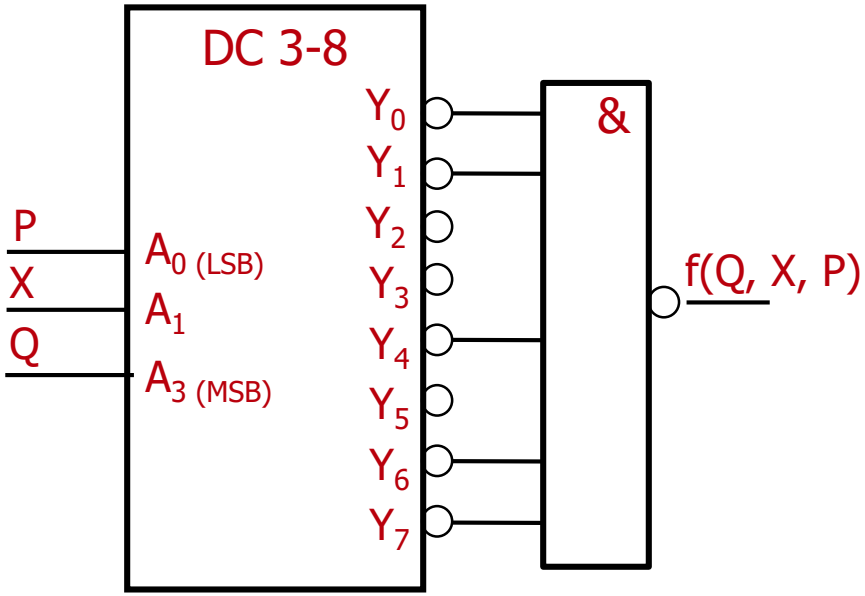


Dekodér: Použití pro generování log. funkcí

- Implementace funkce $f(Q, X, P) = \sum m(0,1,4,6,7)$
  - S výstupem aktivním v jedničce pomocí tříbitového dekodéru s výstupy aktivními v log. 1 a log. členu OR
  - S výstupem aktivním v nule pomocí tříbitového dekodéru s výstupy aktivními v nule a log. členu NAND (DeMorganův zákon)

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

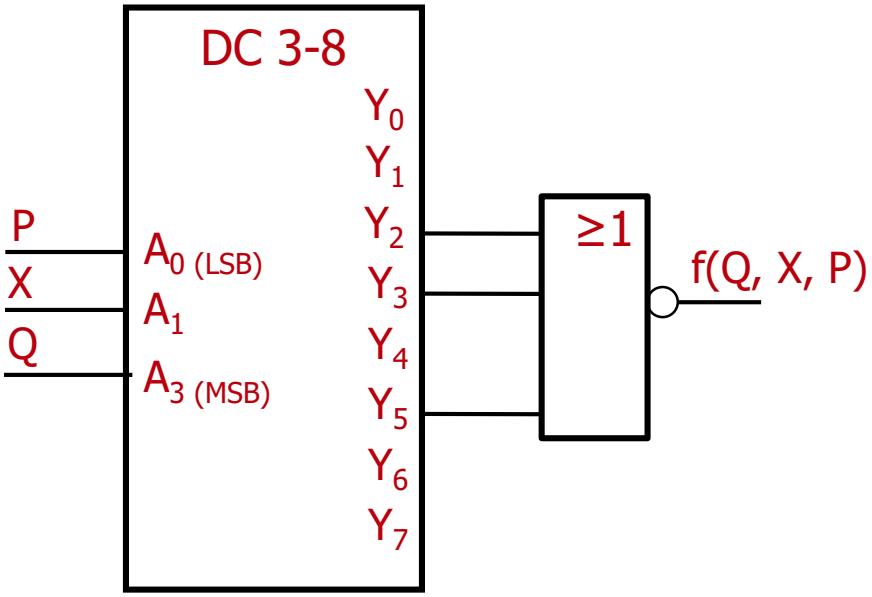


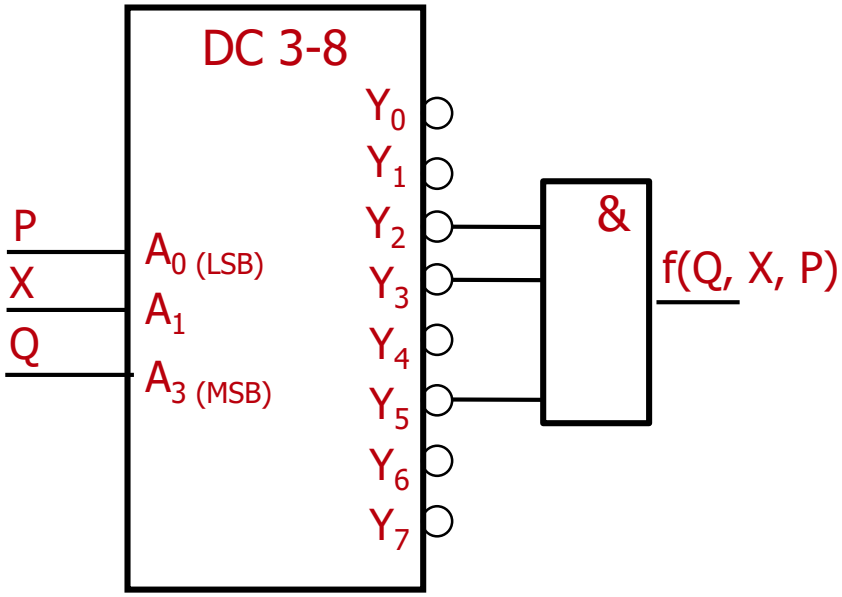


Dekodér: Použití pro generování log. funkcí

- Implementace funkce $f(Q, X, P) = \sum m(0,1,4,6,7)$
  - S výstupem aktivním v jedničce pomocí tříbitového dekodéru s výstupy aktivními v nule a log. členu NOR
  - S výstupem aktivním v nule pomocí tříbitového dekodéru s výstupy aktivními v nule a log. členu AND (DeMorganův zákon)

$$f(Q, X, P) = \sum m(0,1,4,6,7) = \prod M(2,3,5)$$

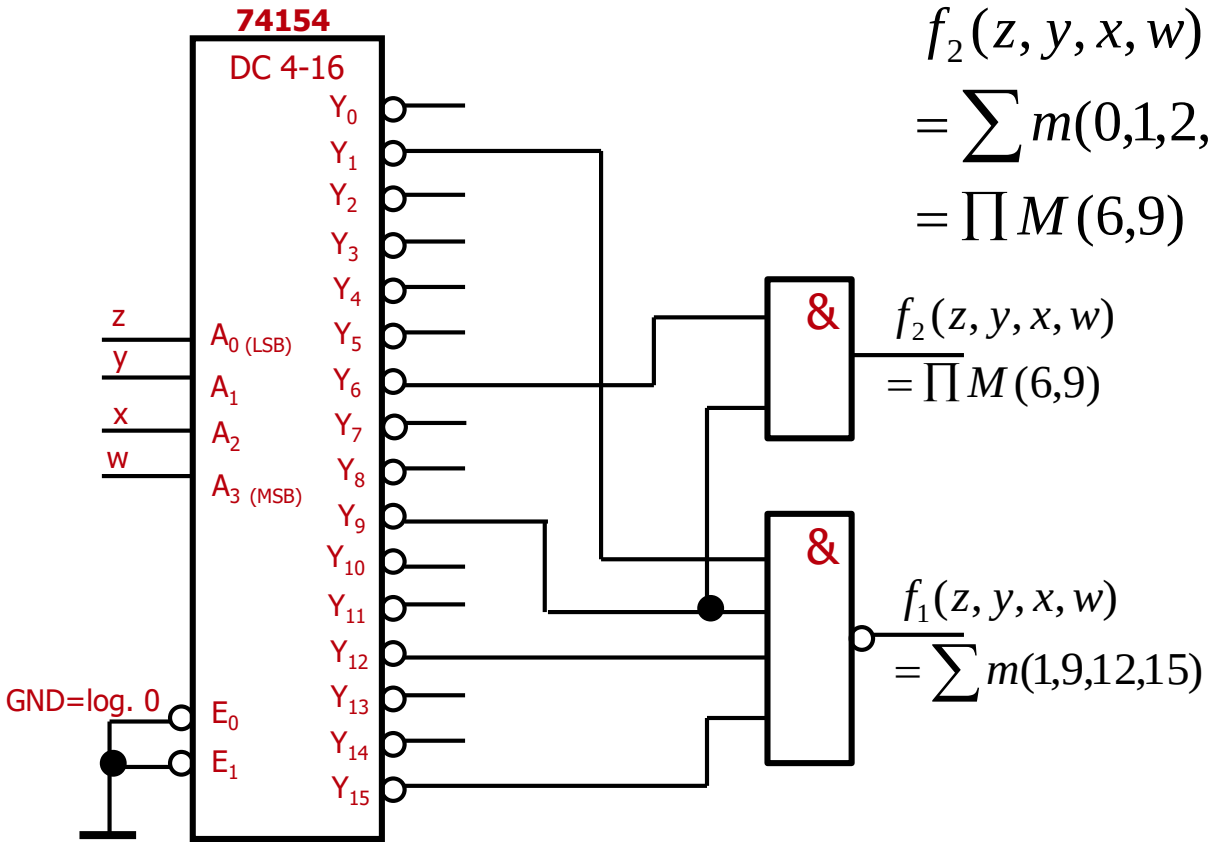




- Příklad
  - Implementace funkcí pomocí dekodéru DC 4-16 s povolovacími vstupy aktivními v nule - IO 74154

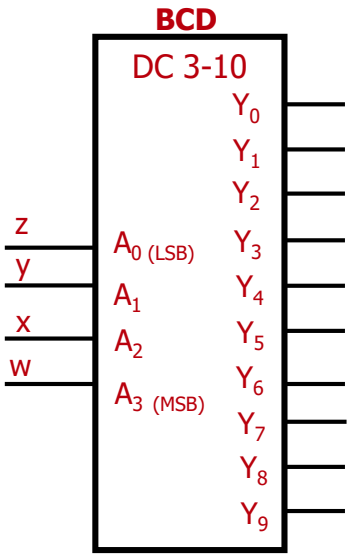
$$f_1(z,y,x,w) = \sum m(1,9,12,15)$$

$$f_2(z,y,x,w) = \sum m(0,1,2,3,4,5,7,8,10,11,13,14) = \prod M(6,9)$$



Dekodér: Dekadický (Binary Coded Decimal - BCD)

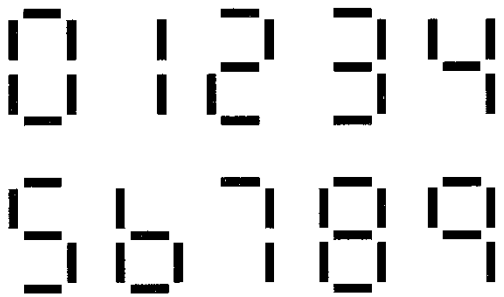
- Dekóduje kód BCD (prvních deset binárních čísel) na kód „1 z 10" (vždy pouze jeden z deseti výstupů je aktivní)

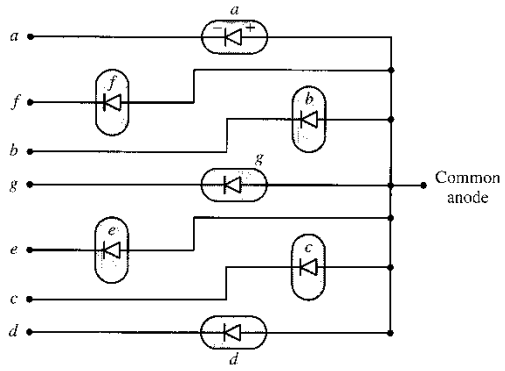


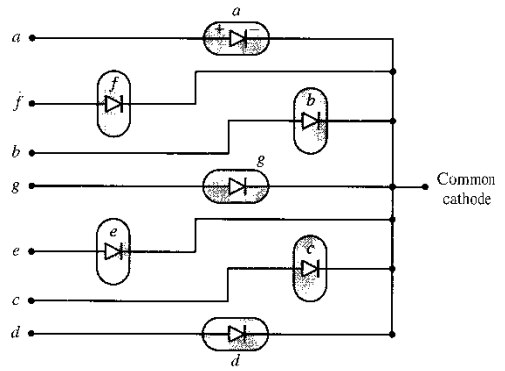
#	A_3	A_2	A_1	A_0	Y_9	Y_8	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**
1	0	0	0	1	0	0	0	0	0	0	0	0	**1**	0
2	0	0	1	0	0	0	0	0	0	0	0	**1**	0	0
3	0	0	1	1	0	0	0	0	0	0	**1**	0	0	0
4	0	1	0	0	0	0	0	0	0	**1**	0	0	0	0
5	0	1	0	1	0	0	0	0	**1**	0	0	0	0	0
6	0	1	1	0	0	0	0	**1**	0	0	0	0	0	0
7	0	1	1	1	0	0	**1**	0	0	0	0	0	0	0
8	1	0	0	0	0	**1**	0	0	0	0	0	0	0	0
9	1	0	0	1	**1**	0	0	0	0	0	0	0	0	0
10	1	0	1	0	0	0	0	0	0	0	0	0	0	0
11	1	0	1	1	0	0	0	0	0	0	0	0	0	0
12	1	1	0	0	0	0	0	0	0	0	0	0	0	0
13	1	1	0	1	0	0	0	0	0	0	0	0	0	0
14	1	1	1	0	0	0	0	0	0	0	0	0	0	0
15	1	1	1	1	0	0	0	0	0	0	0	0	0	0

Dekodér: Kód BCD na 7-segmentový displej

- Sedmi segmentový displej LED
  - Se společnou anodou (dekodér s výstupy aktivními v nule)
  - Se společnou katodou (dekodér s výstupy aktivními v jedničce)







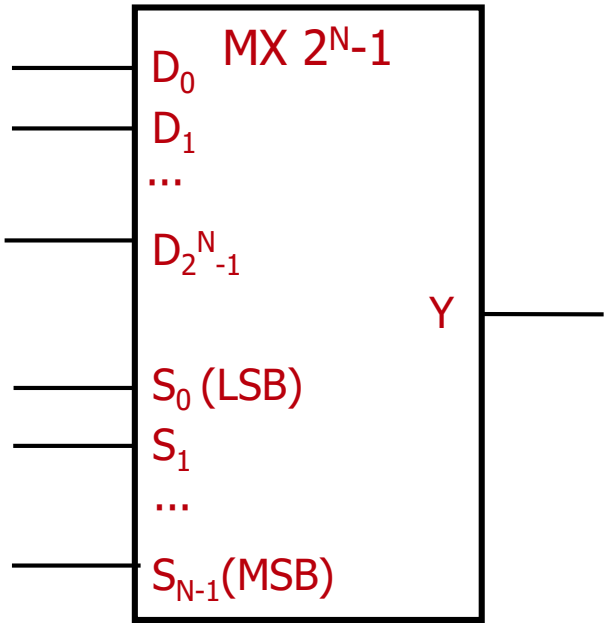
- Pravdivostní tabulka
  - Výstupy aktivní v jedničce

Číslo	BCD kód				Segmenty displeje						
#	A_3	A_2	A_1	A_0	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1

Multiplexor (anglicky Multiplexer)

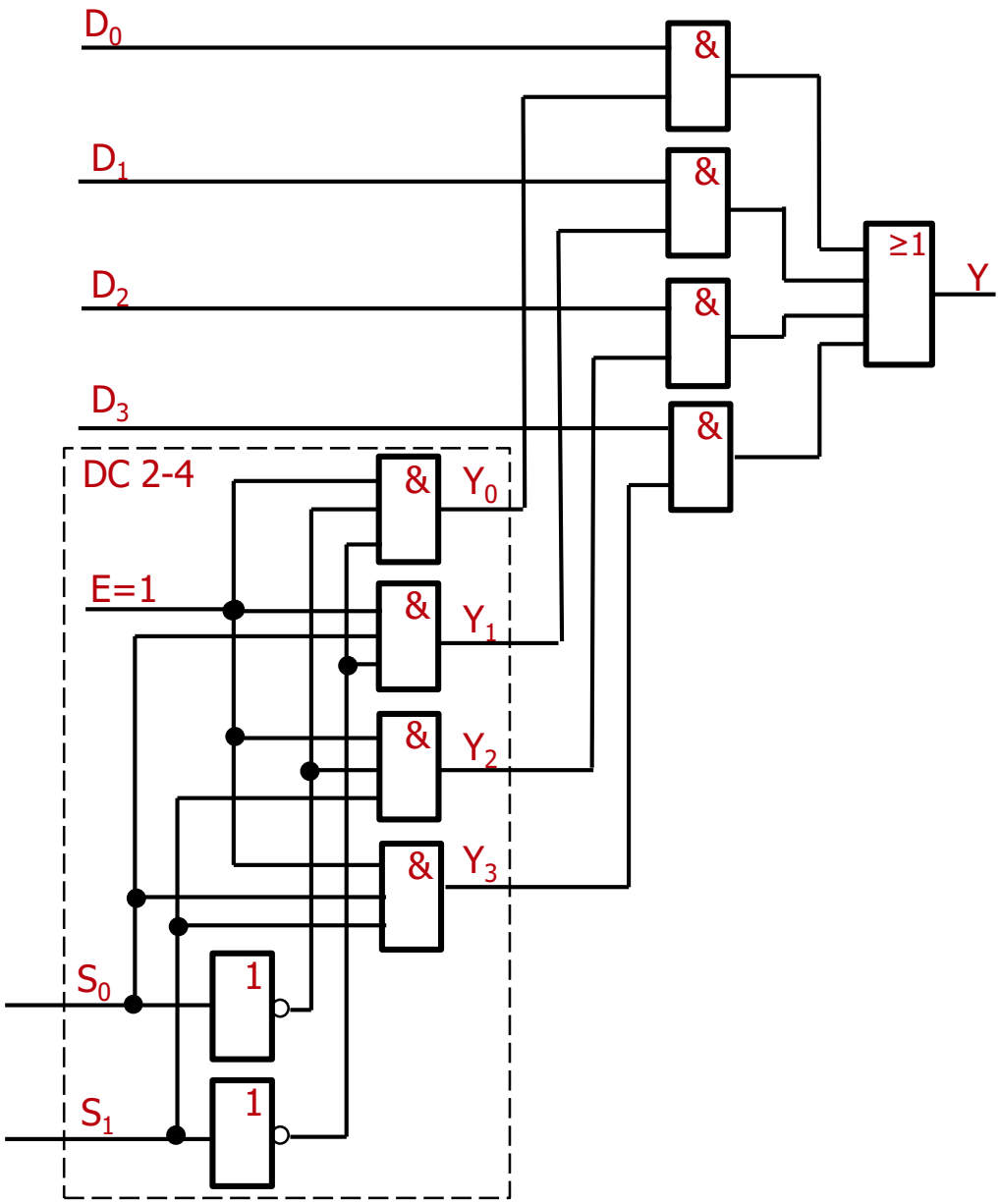
- Charakteristika
  - Kombinační log. síť, která přepíná log. úrovně z více vstupů na jeden výstup
  - Značí se MX či MUX
  - Pomocí binární kombinace na řídicím vstupu S se vybere log. úroveň z příslušného vstupu D_i a kopíruje se (generuje se shodná log. úroveň) na výstup Y
- Použití
  - Data selector
  - Generátor log. funkcí
  - atd.

$$Y = \sum_{i=0}^{j} m_i \cdot D_i$$



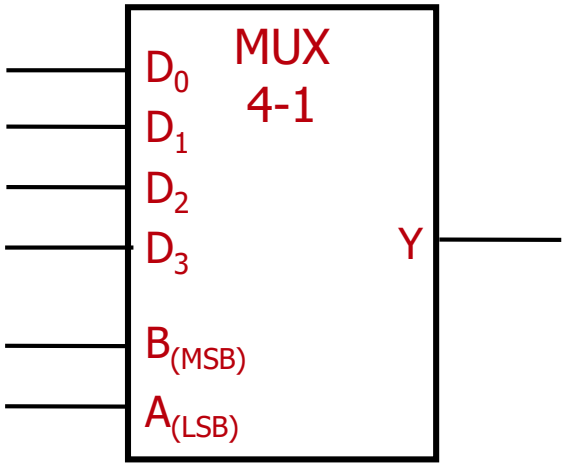
Multiplexor 4-1 (1 ze 4)

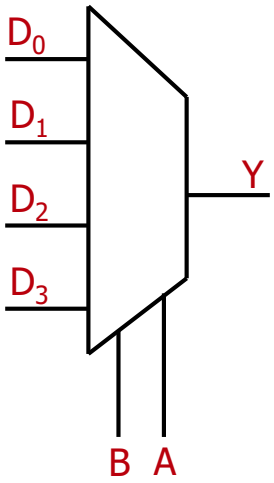
- Konstrukce pomocí dekodéru s povolovacím (datovým) vstupem E=1

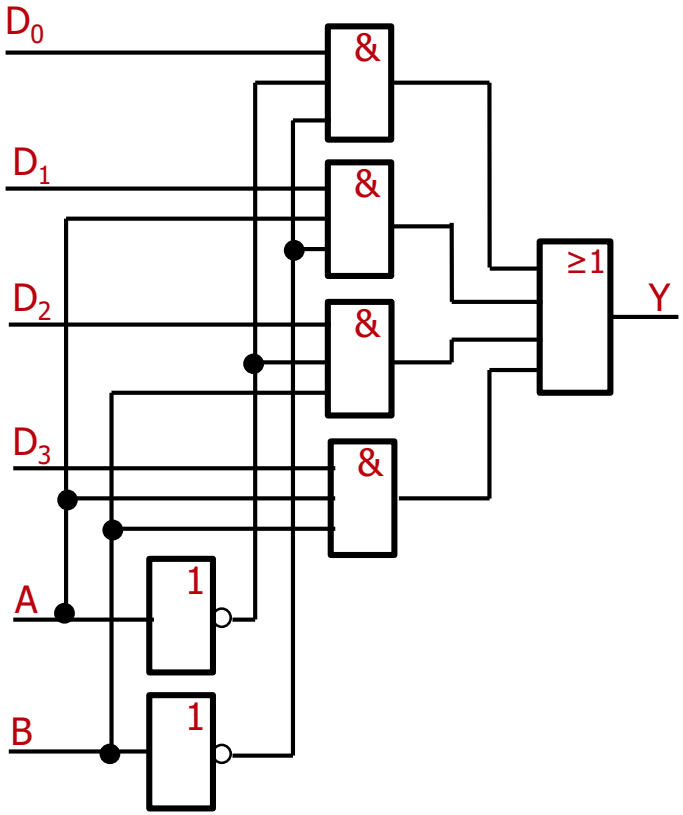


Multiplexor 4-1

- Logické značky, schéma, výraz
  - Pozor na pořadí (váhy) řídicích vstupů B (MSB), A (LSB)







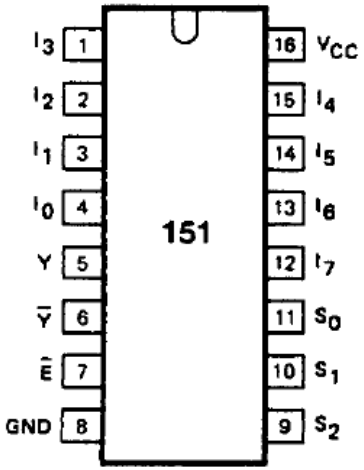
$$Y = \overline{B} \cdot \overline{A} \cdot D_0 + \overline{B} \cdot A \cdot D_1 + B \cdot \overline{A} \cdot D_2 + B \cdot A \cdot D_3$$

Multiplexor 8-1: 74151

- Příklad realizace jako standardní IO typu 74151
  - S povolovacím vstupem E’ (aktivní v nule) a komplementárními výstupy Y a Y’
- Funkční tabulka
- Zapojení vývodů pouzdra

INPUTS												OUTPUTS	
$\overline{E}$	S_2	S_1	S_0	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	$\overline{Y}$	Y
H	X	X	X	X	X	X	X	X	X	X	X	H	L
L	L	L	L	L	X	X	X	X	X	X	X	H	L
L	L	L	L	H	X	X	X	X	X	X	X	L	H
L	L	L	H	X	L	X	X	X	X	X	X	H	L
L	L	L	H	X	H	X	X	X	X	X	X	L	H
L	L	H	L	X	X	L	X	X	X	X	X	H	L
L	L	H	L	X	X	H	X	X	X	X	X	L	H
L	L	H	H	X	X	X	L	X	X	X	X	H	L
L	L	H	H	X	X	X	H	X	X	X	X	L	H
L	H	L	L	X	X	X	X	L	X	X	X	H	L
L	H	L	L	X	X	X	X	H	X	X	X	L	H
L	H	L	H	X	X	X	X	X	L	X	X	H	L
L	H	L	H	X	X	X	X	X	H	X	X	L	H
L	H	H	L	X	X	X	X	X	X	L	X	H	L
L	H	H	L	X	X	X	X	X	X	H	X	L	H
L	H	H	H	X	X	X	X	X	X	X	L	H	L
L	H	H	H	X	X	X	X	X	X	X	H	L	H

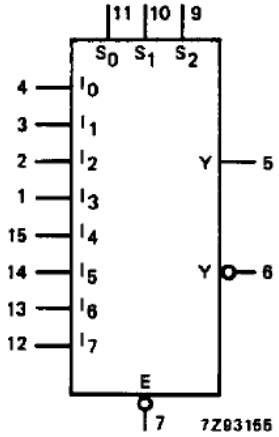
PIN NO.	SYMBOL	NAME AND FUNCTION
4, 3, 2, 1, 15, 14, 13, 12	I_0 to I_7	multiplexer inputs
5	Y	multiplexer output
6	$\overline{Y}$	complementary multiplexer output
7	$\overline{E}$	enable input (active LOW)
8	GND	ground (0 V)
11, 10, 9	S_0, S_1, S_2	select inputs
16	V_{CC}	positive supply voltage



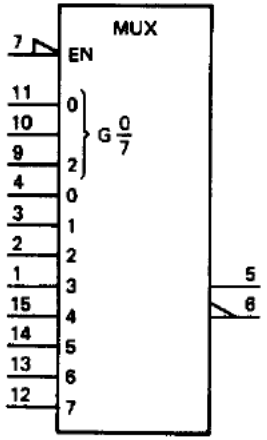
[Obrázky: Datasheet, Philips Semiconductors]

Multiplexor 8-1: 74151

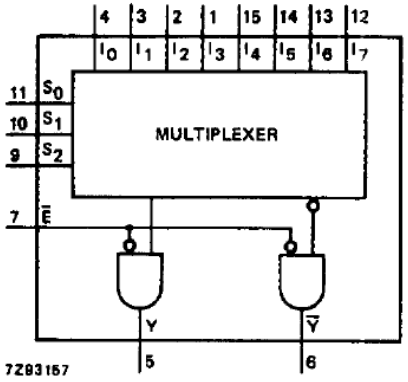
- Logický symbol



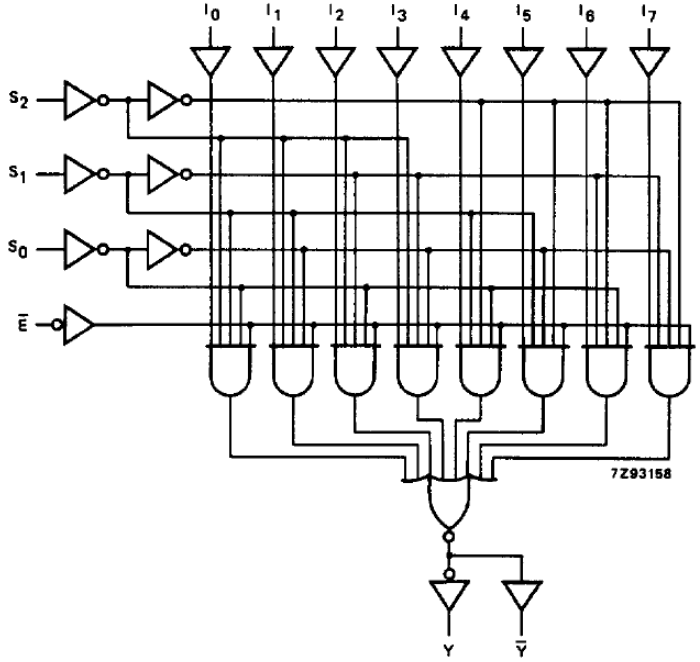
- Logický symbol dle IEC



- Funkční diagram



- Logické schéma



[Obrázky: Datasheet, Philips Semiconductors]

Multiplexor dvojnásobný (dvoubitový) 4-1 : 74153

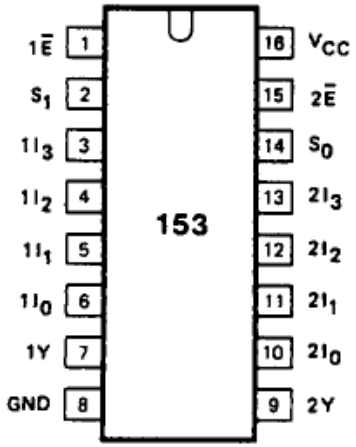
- Příklad realizace jako standardní IO typu 74153
  - S povolovacím vstupem
  - Jedná se o dva nezávislé multiplexory
- Použití
  - Přepínání dvoubitové informace

Funkční tabulka

- Zapojení vývodů pouzdra

SELECT INPUTS		DATA INPUTS				OUTPUT ENABLE	OUTPUT
S_0	S_1	nI_0	nI_1	nI_2	nI_3	$n\overline{E}$	nY
X	X	X	X	X	X	H	L
L	L	L	X	X	X	L	L
L	L	H	X	X	X	L	H
H	L	X	L	X	X	L	L
H	L	X	H	X	X	L	H
L	H	X	X	L	X	L	L
L	H	X	X	H	X	L	H
H	H	X	X	X	L	L	L
H	H	X	X	X	H	L	H

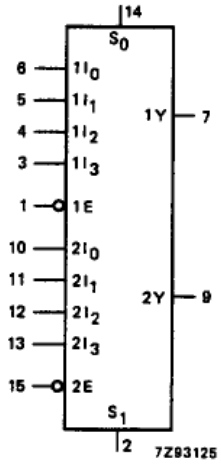
PIN NO.	SYMBOL	NAME AND FUNCTION
1, 15	$1\overline{E}$, $2\overline{E}$	output enable inputs (active LOW)
14, 2	S_0, S_1	common data select inputs
6, 5, 4, 3	$1I_0$ to $1I_3$	data inputs from source 1
7	1Y	multiplexer output from source 1
8	GND	ground (0 V)
9	2Y	multiplexer output from source 2
10, 11, 12, 13	$2I_0$ to $2I_3$	data inputs from source 2
16	V_{CC}	positive supply voltage



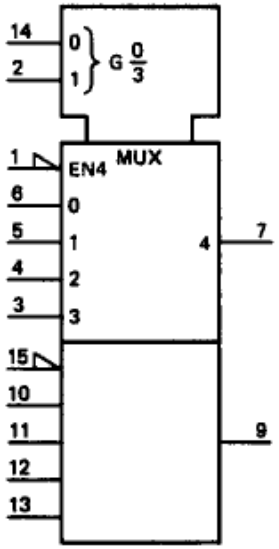
[Obrázky: Datasheet, Philips Semiconductors]

Multiplexor dvojnásobný (dvoubitový) 4-1 : 74153

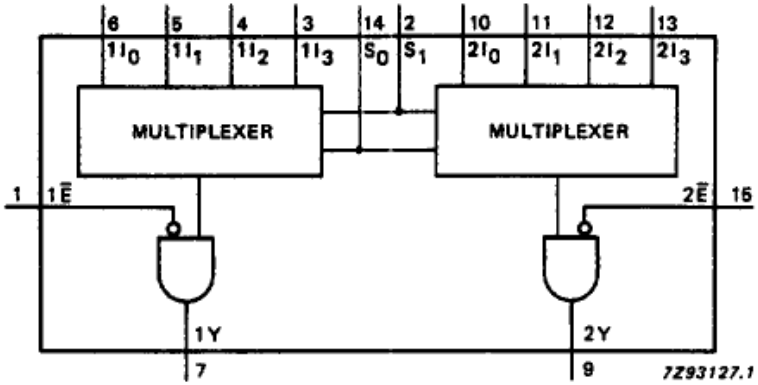
- Logický symbol



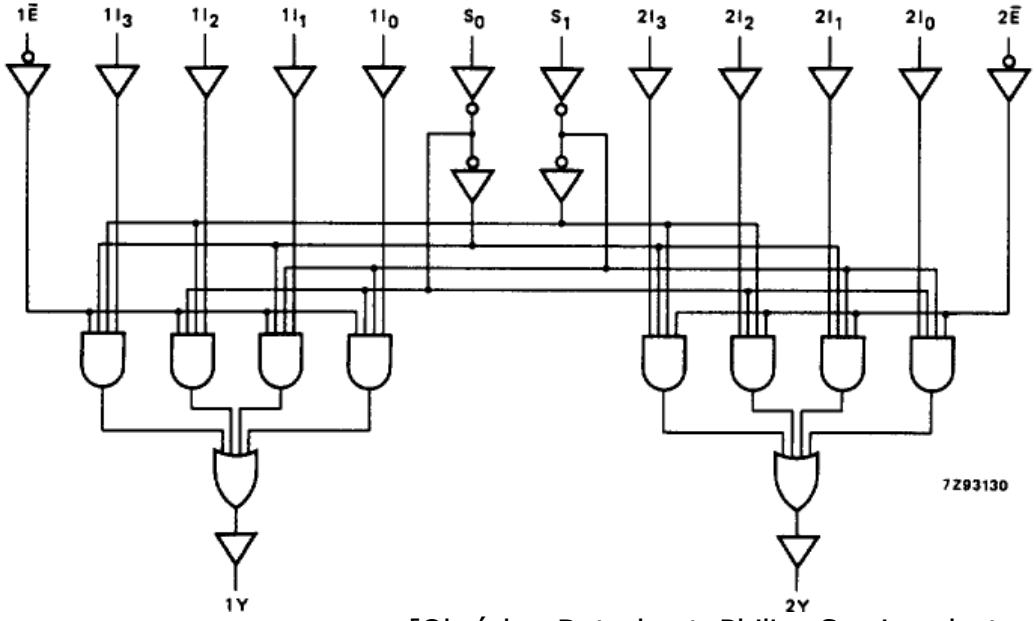
- Logický symbol dle IEC



- Funkční diagram



- Logické schéma

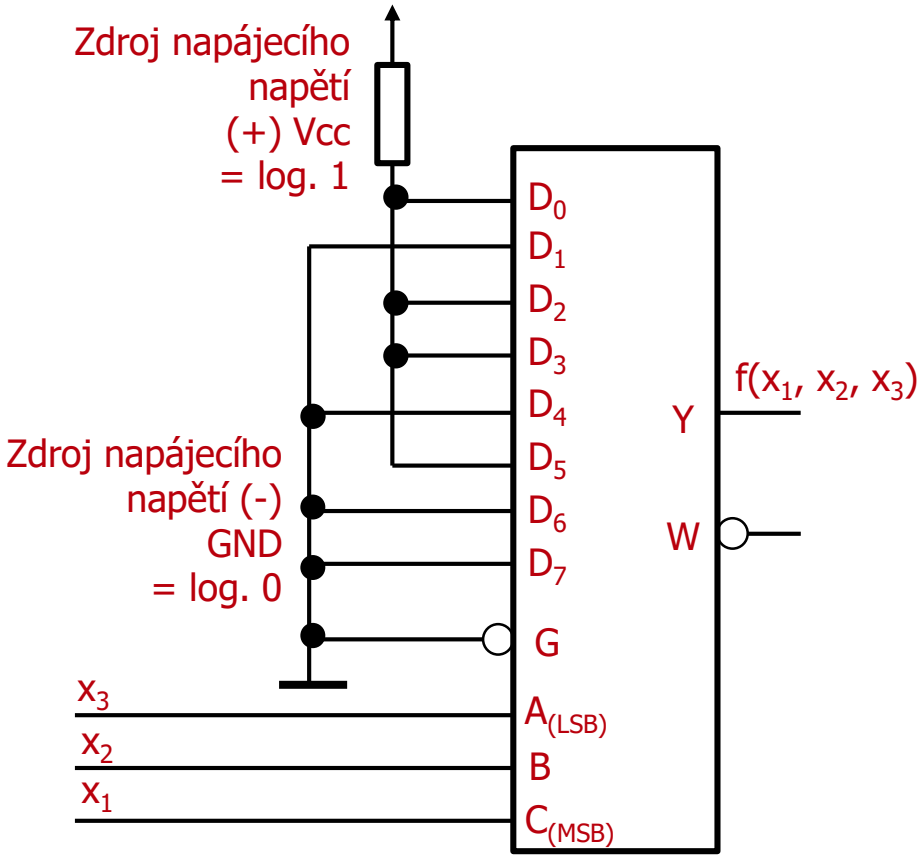


[Obrázky: Datasheet, Philips Semiconductors]

Multiplexor: Použití pro implementaci log. funkcí

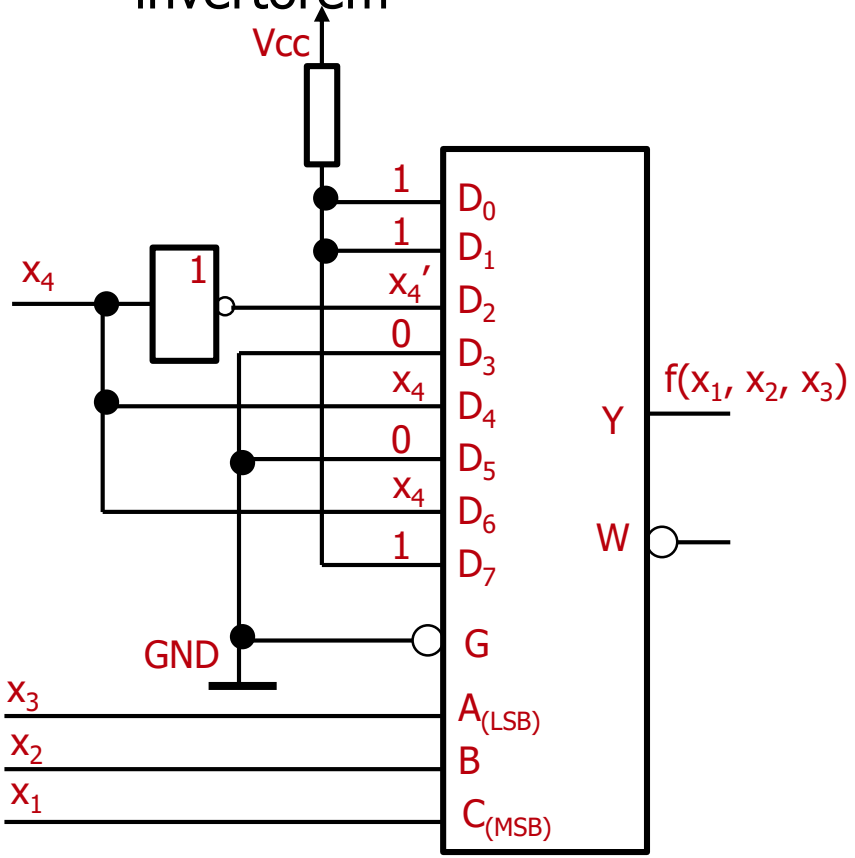
- Příklad: Implementace funkce $f(x_1, x_2, x_3) = \sum m(0,2,3,5)$
  - Pravdivostní tabulka
  - Realizace multiplexorem 8-1 (standardní IO 74151A)

Vstupy		Výstup
C, B, A	Data	Y
x_1, x_2, x_3		$f(x_1, x_2, x_3)$
000	$D_0=1$	1
001	$D_1=0$	0
010	$D_2=1$	1
011	$D_3=1$	1
100	$D_4=0$	0
101	$D_5=1$	1
110	$D_6=0$	0
111	$D_7=0$	0



Multiplexor: Použití pro implementaci log. funkcí

- Příklad
  - Implementace funkce $f(x_1, x_2, x_3, x_4) = \sum m(0,1,2,3,4,9,13,14,15)$
  - Realizace multiplexorem 8-1 (standardní IO 74151A) a invertorem



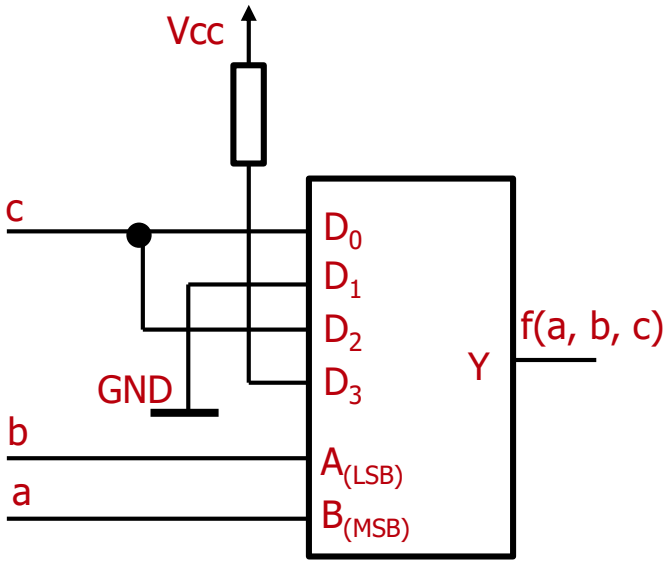
	C, B, A	x_4	f	=	Y
	x_1, x_2, x_3				$f(x_1, x_2, x_3, x_4)$
0 1	000 000	0 1	1 1	1	$D_0=1$
2 3	001 001	0 1	1 1	1	$D_1=1$
4 5	010 010	0 1	1 0	x_4'	$D_2=x_4'$
6 7	011 011	0 1	0 0	0	$D_3=0$
8 9	100 100	0 1	0 1	x_4	$D_4=x_4$
10 11	101 101	0 1	0 0	0	$D_5=0$
12 13	110 110	0 1	0 1	x_4	$D_6=x_4$
14 15	111 111	0 1	1 1	1	$D_7=1$

Multiplexor: Použití pro implementaci log. funkcí

- Příklad – verze 1
  - Implementace funkce
  - Pravdivostní tabulka
  - Realizace multiplexorem 4-1

$$f(a,b,c) = a \cdot b + \bar{b} \cdot c$$
$$= a \cdot b \cdot 1 + 1 \cdot \bar{b} \cdot c$$
$$= a \cdot b \cdot (c + \bar{c}) + (a + \bar{a}) \cdot \bar{b} \cdot c$$
$$= a \cdot b \cdot c + a \cdot b \cdot \bar{c} + a \cdot \bar{b} \cdot c + \bar{a} \cdot \bar{b} \cdot c$$

Vstupy		**Výstup**
B, A	**Data**	**Y**
a, b		**f(a, b, c)**
00	D_0=c	c
01	D_1=0	0
10	D_2=c	c
11	D_3=1	1

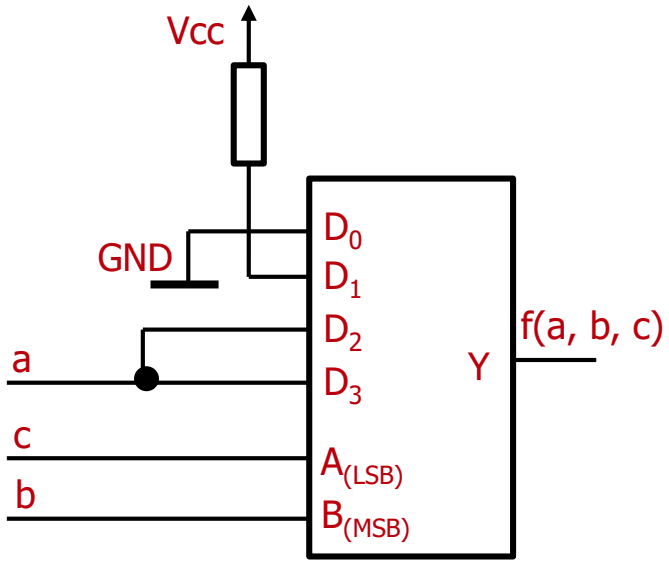


Multiplexor: Použití pro implementaci log. funkcí

- Příklad – verze 2
  - Implementace funkce
  - Pravdivostní tabulka
  - Realizace multiplexorem 4-1

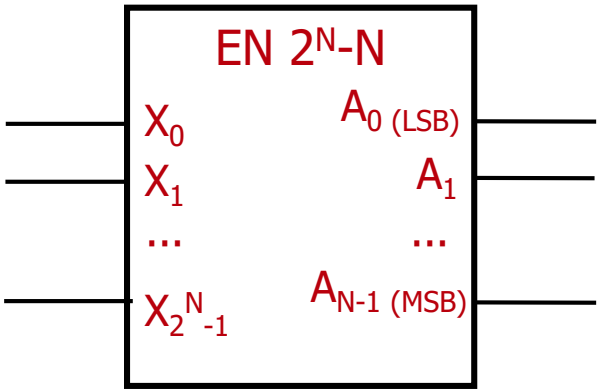
$$f(a,b,c) = a \cdot b + \bar{b} \cdot c$$
$$= a \cdot b \cdot 1 + 1 \cdot \bar{b} \cdot c$$
$$= a \cdot b \cdot (c + \bar{c}) + (a + \bar{a}) \cdot \bar{b} \cdot c$$
$$= a \cdot b \cdot c + a \cdot b \cdot \bar{c} + a \cdot \bar{b} \cdot c + \bar{a} \cdot \bar{b} \cdot c$$

Vstupy		**Výstup**
B, A	**Data**	**Y**
b, c		**f(a, b, c)**
00	D_0=0	0
01	D_1=1	1
10	D_2=a	a
11	D_3=a	a



Kodér (anglicky Encoder)

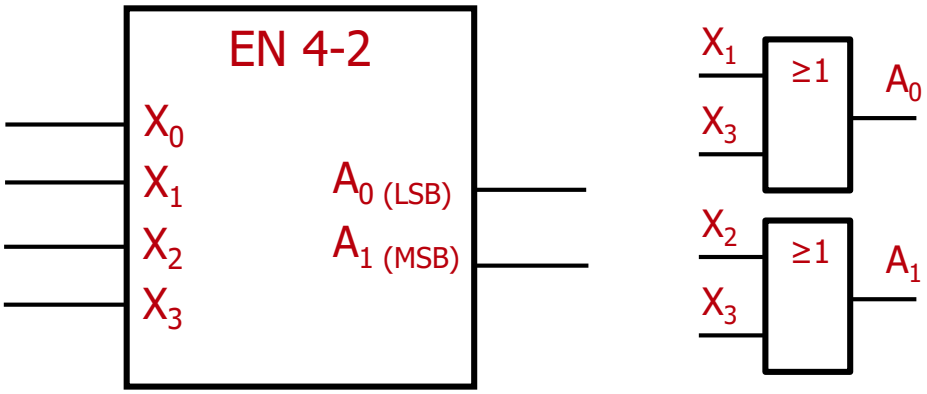
- Charakteristika
  - Kodér je kombinační log. síť, která převádí kód „1 z M”, kde $M=2^N$ na N bitový binární kód (číslo aktivního vstupu)
  - Má opačnou funkci k dekodéru
  - Značí se EN 2^N-N
  - Na výstupu je binární kód (číslo) aktivního vstupu (právě jen jeden může být aktivní v daním čase)
  - Pokud nelze zajistit, aby byl vždy jen jeden vstup aktivní, je třeba použít tzv. prioritní kodér (viz dále)
- Použití
  - Převody kódů
  - Kódování
  - Stanovení priority



Kodér 4-2

- Příklad
  - Funkce je opačná k dekodéru 2-4
  - Výstupem je binární číslo odpovídající váze (číslu) aktivního vstupu
  - X - (don't care): hodnota není dle definice kodéru platná
  - Dle definice může být platná právě jen jedna proměnná – neurčené hodnoty nemohou nastat
  - Pokud může být více vstupů aktivních, je třeba tuto situaci ošetřit (detekovat – viz dále)

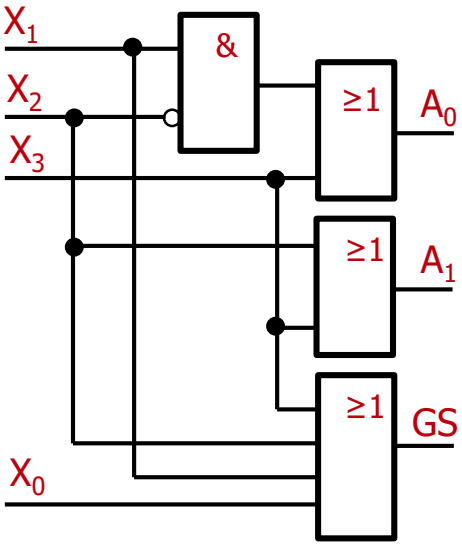
Vstupy				Výstupy	
X_3	X_2	X_1	X_0	A_1	A_0
0	0	0	0	X	X
0	0	0	1	0	0
0	0	1	0	0	1
0	0	1	1	X	X
0	1	0	0	1	0
0	1	0	1	X	X
0	1	1	0	X	X
0	1	1	1	X	X
1	0	0	0	1	1
1	0	0	1	X	X
1	0	1	0	X	X
1	0	1	1	X	X
1	1	0	0	X	X
1	1	0	1	X	X
1	1	1	0	X	X
1	1	1	1	X	X

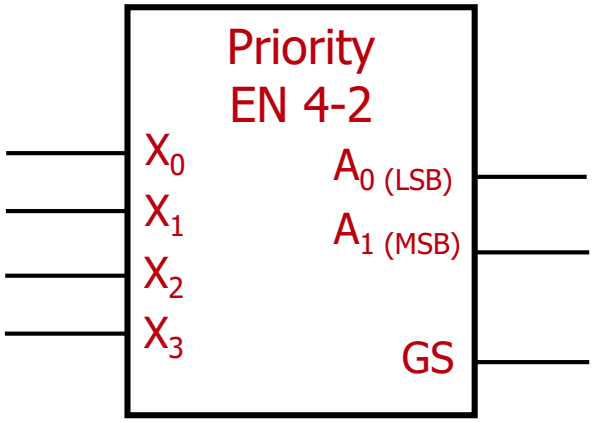


Prioritní kodér (anglicky Priority Encoder) 4-2

- Výstupní binární číslo určuje, který ze vstupních signálů s nejvyšší prioritou je aktivní
  - Pokud je více vstupů aktivních, výstupem bude binární číslo toho, který má nejvyšší prioritu
  - A1, A0 – číslo aktivního vstupu s nejvyšší prioritou
  - GS – detekce - jeden či více vstupů je aktivní

Vstupy				Výstupy		
x_3	x_2	x_1	x_0	A_1	A_0	GS
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	1
0	0	1	1	0	1	1
0	1	0	0	1	0	1
0	1	0	1	1	0	1
0	1	1	0	1	0	1
0	1	1	1	1	0	1
1	0	0	0	1	1	1
1	0	0	1	1	1	1
1	0	1	0	1	1	1
1	0	1	1	1	1	1
1	1	0	0	1	1	1
1	1	0	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	1	1	1



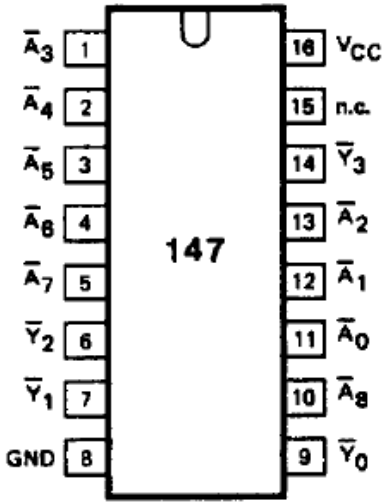


Prioritní kodér 10-4: 74147

- Příklad realizace jako standardní IO typu 74147
  - Deset vstupů aktivních v nule (BCD kód)
  - 4 výstupy – adresa aktivního vstupu s nejvyšší prioritou
- Funkční tabulka
- Zapojení vývodů pouzdra

INPUTS									OUTPUTS			
$\overline{A}_0$	$\overline{A}_1$	$\overline{A}_2$	$\overline{A}_3$	$\overline{A}_4$	$\overline{A}_5$	$\overline{A}_6$	$\overline{A}_7$	$\overline{A}_8$	$\overline{Y}_3$	$\overline{Y}_2$	$\overline{Y}_1$	$\overline{Y}_0$
H	H	H	H	H	H	H	H	H	H	H	H	H
X	X	X	X	X	X	X	X	L	L	H	H	L
X	X	X	X	X	X	X	L	H	L	H	H	H
X	X	X	X	X	X	L	H	H	H	L	L	L
X	X	X	X	X	L	H	H	H	H	L	L	H
X	X	X	X	L	H	H	H	H	H	L	H	L
X	X	X	L	H	H	H	H	H	H	L	H	H
X	X	L	H	H	H	H	H	H	H	H	L	L
X	L	H	H	H	H	H	H	H	H	H	L	H
L	H	H	H	H	H	H	H	H	H	H	H	L

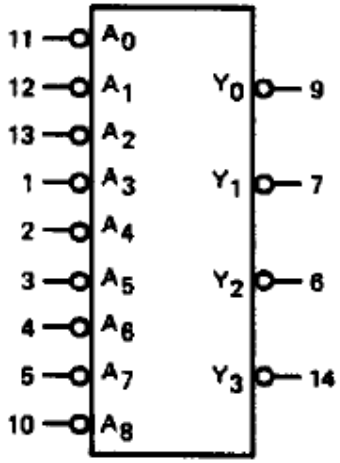
PIN NO.	SYMBOL	NAME AND FUNCTION
8	GND	ground (0 V)
9, 7, 6, 14	$\overline{Y}_0$ to $\overline{Y}_3$	BCD address outputs (active LOW)
11, 12, 13, 1, 2, 3, 4, 5, 10	$\overline{A}_0$ to $\overline{A}_8$	decimal data inputs (active LOW)
15	n.c.	not connected
16	V_{CC}	positive supply voltage



[Obrázky: Datasheet, Philips Semiconductors]

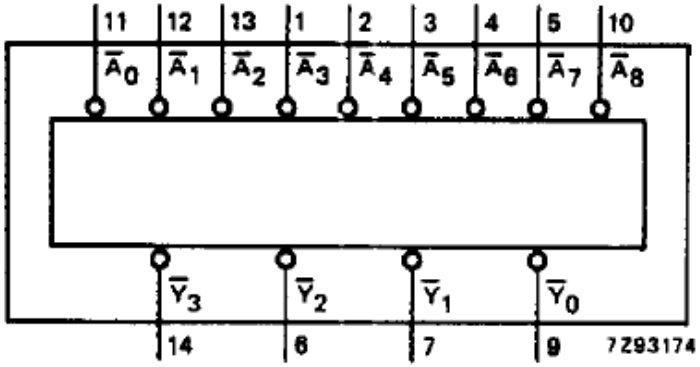
Prioritní kodér 10-4: 74147

- Logický symbol

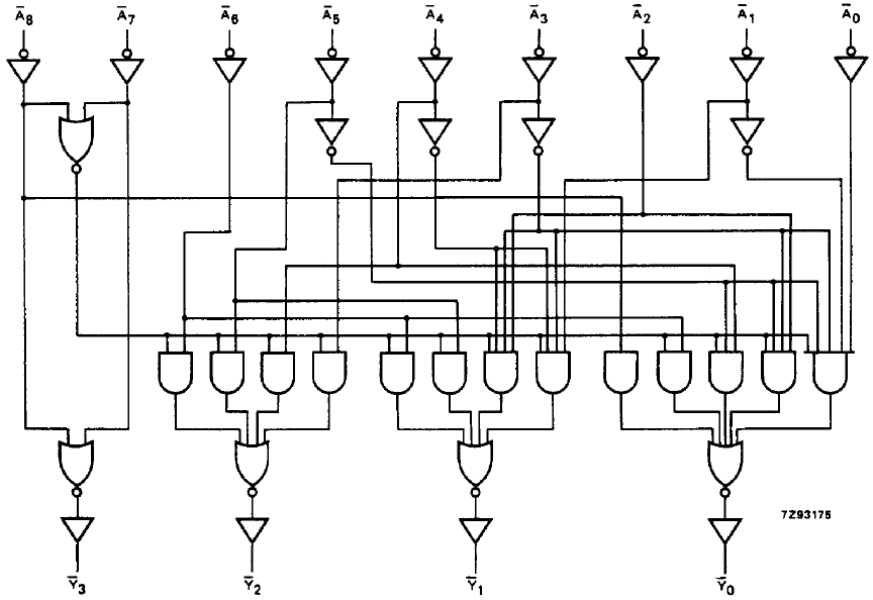


- Logický symbol dle IEC

- Funkční diagram



- Logické schéma

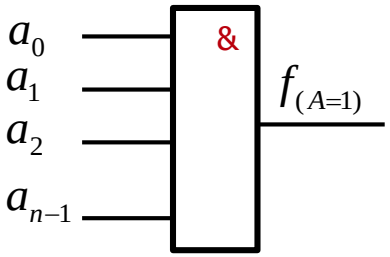


[Obrázky: Datasheet, Philips Semiconductors]

Aritmetické a logické operace

- Test na jedničkovou hodnotu
  - Pokud jsou všechny bity vstupního vektoru rovny log. 1, pak je výstup roven log. 1

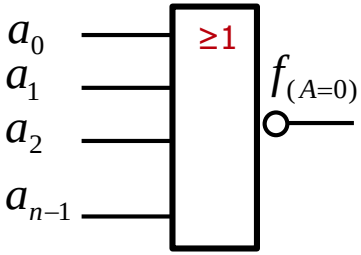
$$f_{(A=1)} = a_0 \cdot \ldots \cdot a_{n-1}$$



- Test na nulovou hodnotu
  - Pokud jsou všechny bity vstupního vektoru rovny log. 0, pak je výstup roven log. 1

$$f_{(A=0)} = \overline{a_0} \cdot \overline{a_1} \cdot \ldots \cdot \overline{a_{n-1}}$$

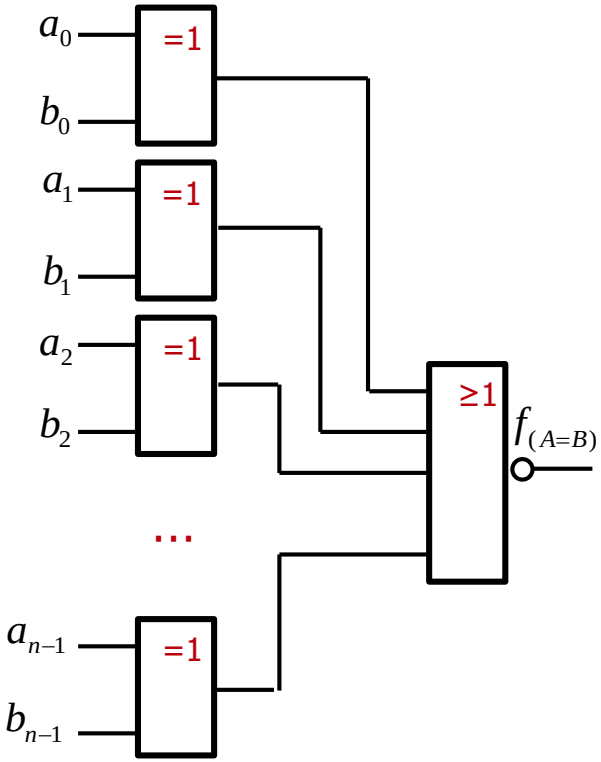
$$= \overline{a_0 + \ldots + a_{n-1}}$$



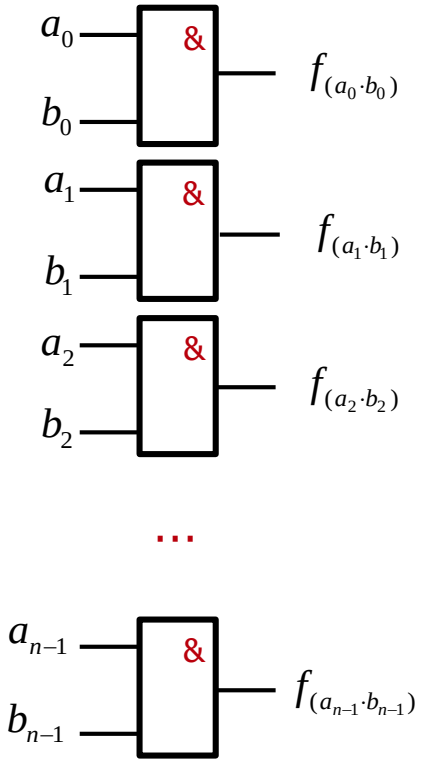
Aritmetické a logické operace

- Test na shodu
  - Porovnání dvou n-bitových vektorů

$$f_{(A=B)} = \overline{a_0 \oplus b_0 + \ldots + a_{n-1} \oplus b_{n-1}}$$



- N-bitové logické operace
  - Provádí se bit po bitu
  - Příklad N-bitový AND



Aritmetické a logické operace

- Relace
  - C – přenos (carry)
  - Z – nula (zero)
  - N – záporné (negative)
  - V – přetečení (overflow)

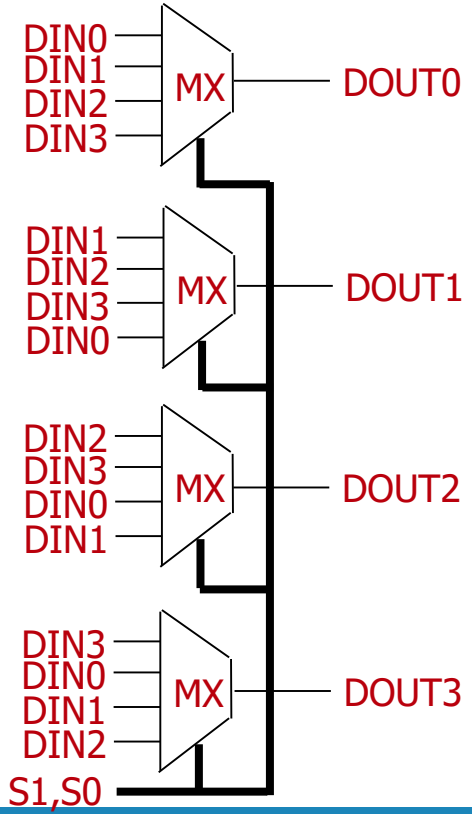
Relace	Bez znaménka	Se znaménkem
$A = B$	Z	Z
$A \neq B$	$\bar{Z}$	$\bar{Z}$
$A < B$	$\overline{C + Z}$	$\overline{(N \oplus V) + Z}$
$A > B$	$\bar{C}$	$(N \oplus V)$
$A \leq B$	C	$\overline{(N \oplus V)}$
$A \geq B$	$\bar{C} + \mathrm{Z}$	$(N \oplus V) + Z$

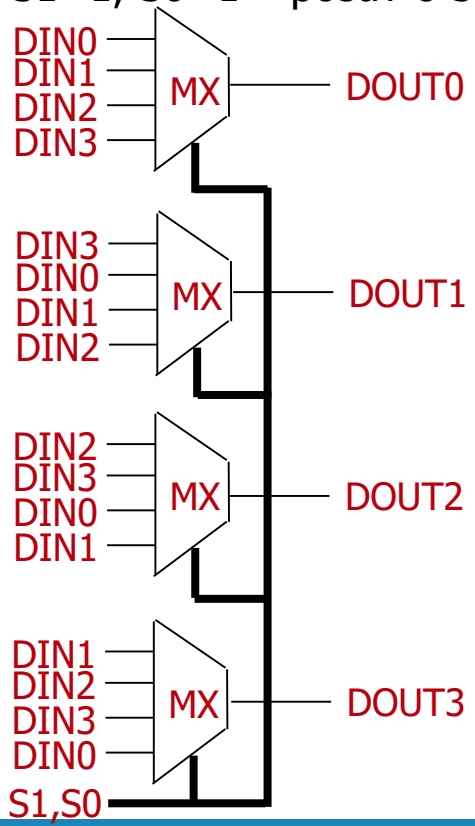
Aritmetické a logické operace

- Posuvy a rotace
- Logický posuv vlevo
  - SLL - Shift Left Logical
  - LSB←0
  - Např.: SLL(1011) = 0110
- Aritmetický posuv vlevo
  - SLA - Shift Left Arithmetic
  - LSB←0
  - Např.: SLA(1011) = 0110
- Rotace vlevo
  - ROL - Rotate Left
  - LSB ←MSB
  - Např.: ROL(1011) = 0111
- Posuv logický vpravo
  - SRL - Shift Right Logical
  - 0→MSB
  - Např.: SRL(1011) = 0101
- Aritmetický posuv vpravo
  - SRA - Shift Right Arithmetic
  - MSB→MSB (šíření znaménka)
  - Např.: SRA(1011)=1101
- Rotace vpravo
  - ROR - Rotate Right
  - LSB→MSB
  - Např.: ROR(1011) = 1101

Aritmetické a logické operace

- Válcový posouvač - n-bitů v jednom kroku (barrel shifter)
- Rotace vpravo v jednom kroku
  - S1=0, S0=0 – posuv o 0 bitů vpravo
  - S1=0, S0=1 – posuv o 1 bit vpravo
  - S1=1, S0=0 – posuv o 2 bity vpravo
  - S1=1, S0=1 – posuv o 3 bity vpravo
- Rotace vlevo v jednom kroku
  - S1=0, S0=0 – posuv o 0 bitů vlevo
  - S1=0, S0=1 – posuv o 1 bit vlevo
  - S1=1, S0=0 – posuv o 2 bity vlevo
  - S1=1, S0=1 – posuv o 3 bity vlevo

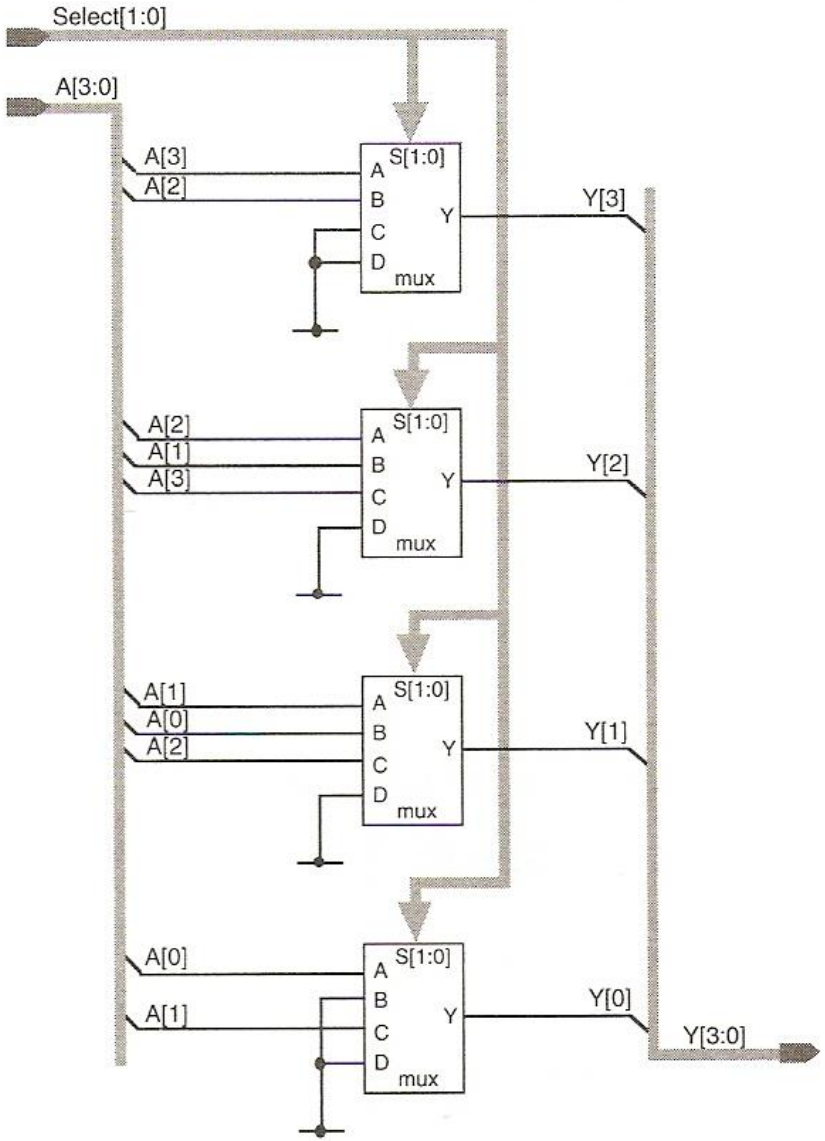




Aritmetické a logické operace

- Posouvač – 1 bit v jednom kroku
  - SLL - Shift Logical Left
  - SRL - Shift Logical Right

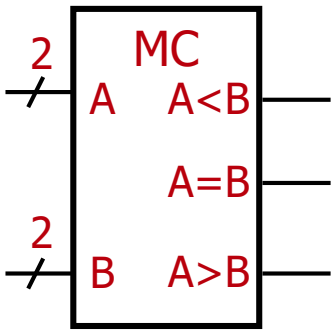
Select	Operace
00	Y←A
01	Y←SLL(A)
10	Y←SRL(A)
11	Y←0



[D.J Smith: „HDL Chip Design"]

Aritmetické a logické operace

- Komparátor (magnitude comparator)
- Příklad 2bitového komparátoru
  - Porovnání hodnot vstupních dvoubitových operandů A a B
- Poznámka:
  - Porovnávat operandy lze též např. pomocí sčítačky přičtením záporného operandu a testováním hodnoty výsledku:
  - Výsledek operace (A-B)=0 => A=B
  - Znaménko výsledku operace (A-B)=1 => A<B

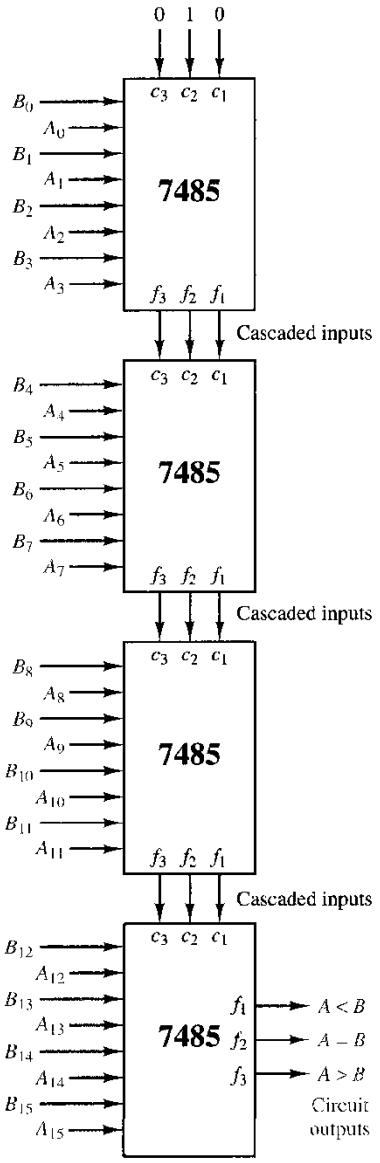


a_1	a_0	b_1	b_0	A<B	A=B	A>B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

Aritmetické a logické operace

- Komparátor: příklad - standardní modul (IO 7485)
  - Čtyřbitový komparátor
  - Lze řadit do kaskády - vícebitové komparátory

Comparing inputs				Cascading inputs			Outputs		
A3, B3	A2, B2	A1, B1	A0, B0	A > B	A < B	A = B	A > B	A < B	A = B
A3 > B3	×	×	×	×	×	×	H	L	L
A3 < B3	×	×	×	×	×	×	L	H	L
A3 = B3	A2 > B2	×	×	×	×	×	H	L	L
A3 = B3	A2 < B2	×	×	×	×	×	L	H	L
A3 = B3	A2 = B2	A1 > B1	×	×	×	×	H	L	L
A3 = B3	A2 = B2	A1 < B1	×	×	×	×	L	H	L
A3 = B3	A2 = B2	A1 = B1	A0 > B0	×	×	×	H	L	L
A3 = B3	A2 = B2	A1 = B1	A0 < B0	×	×	×	L	H	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	H	L	L	H	L	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	L	H	L	L	H	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	L	L	H	L	L	H
A3 = B3	A2 = B2	A1 = B1	A0 = B0	×	×	H	L	L	H
A3 = B3	A2 = B2	A1 = B1	A0 = B0	H	H	L	L	L	L
A3 = B3	A2 = B2	A1 = B1	A0 = B0	L	L	L	H	H	L



[Obrázky: Datasheet, Philips Semiconductors]

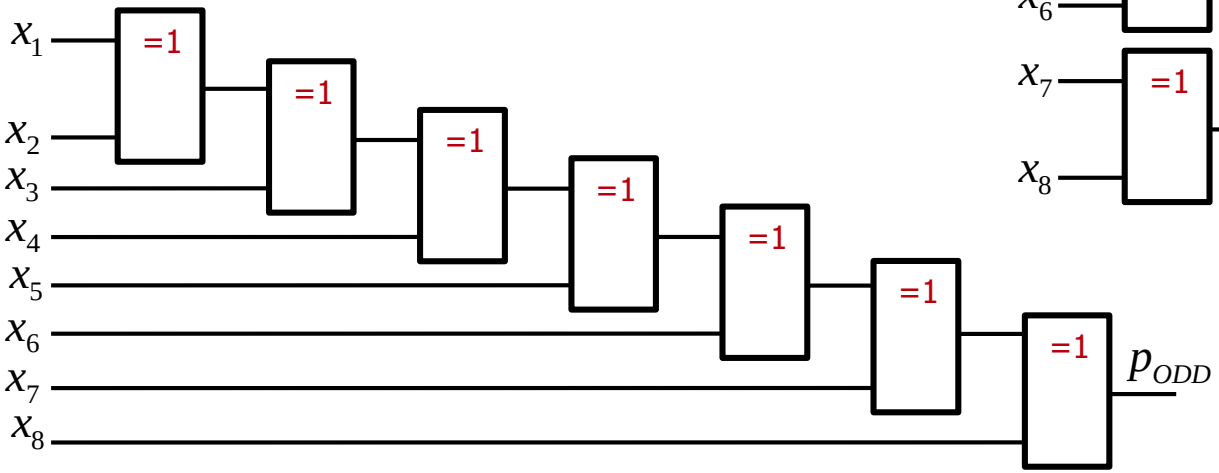
Zabezpečení informace: Parita

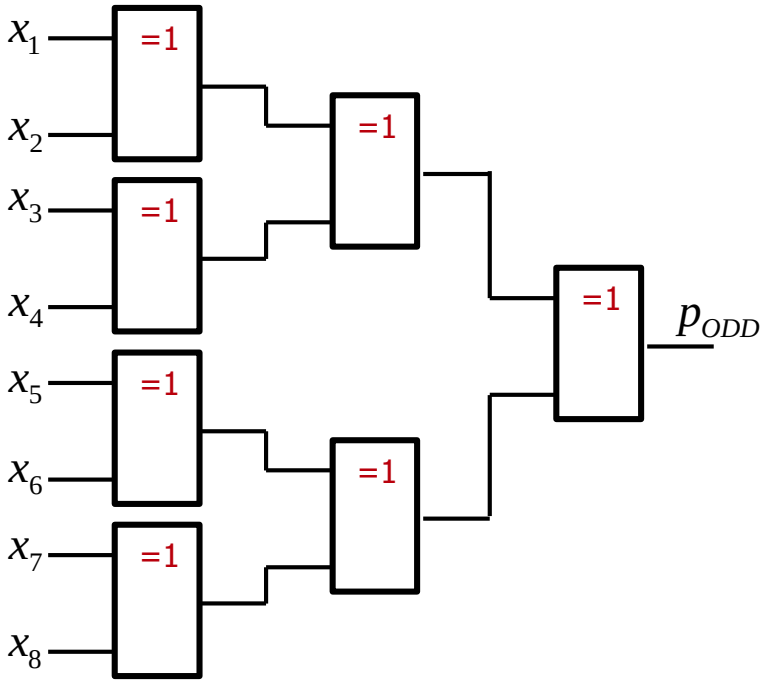
- Určuje zda sudý, resp. lichý počet vstupních proměnných nabývá hodnoty log. 1
  - Sudá - počet jedniček ve vstupním vektoru je sudé číslo
  - Lichá - počet jedniček ve vstupním vektoru je liché číslo
- Použití
  - Zabezpečení informace
- Příklad
  - Zabezpečení BCD kódu sudou paritou

#	BCD				Sudá parita
0	0	0	0	0	**1**
1	0	0	0	1	**0**
2	0	0	1	0	**0**
3	0	0	1	1	**1**
4	0	1	0	0	**0**
5	0	1	0	1	**1**
6	0	1	1	0	**1**
7	0	1	1	1	**0**
8	1	0	0	0	**0**
9	1	0	0	1	**1**

Zabezpečení informace: Parita

- Příklad generování liché parity
  - Anglicky „odd” parity
  - Stromovou strukturou
  - Kaskádní strukturou
- Poznámka
  - Sudá (anglicky „even”) parita je negací liché



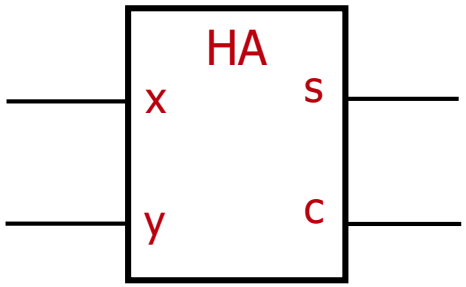


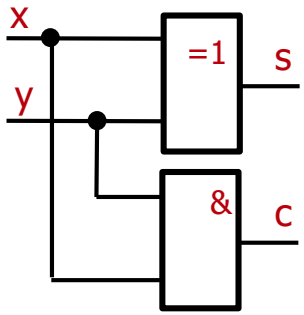
$$p_{ODD}(x_1,\ldots,x_8) = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8$$

Binární sčítačka poloviční

- Poloviční sčítačka (Half Adder - HA)
  - Nemá vstup carry (přenos z nižšího řádu)
  - Výraz pro sumu
    $$s = \overline{x} \cdot y + x \cdot \overline{y} = y \oplus x$$
  - Výraz pro přenos do vyššího řádu carry $c = x \cdot y$
  - Pravdivostní tabulka
  - Logická značka
  - Logické schéma

x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



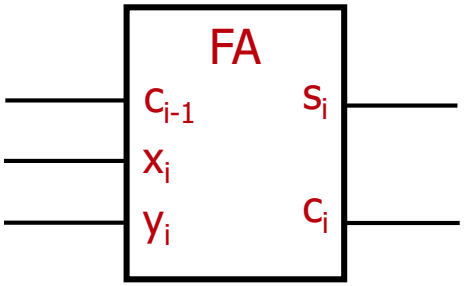


Binární sčítačka úplná

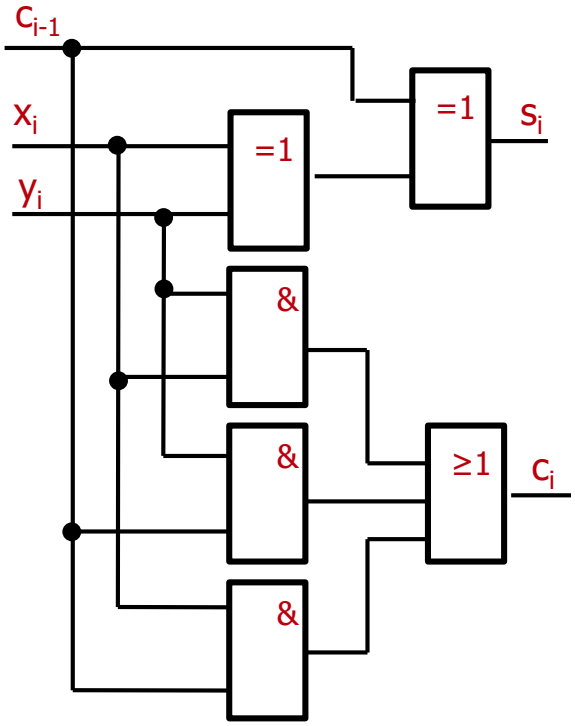
- Úplná sčítačka (Full Adder - FA)
  - Vstup c_{i-1} (carry in - přenos z nižšího řádu)
  - Výraz pro sumu
  - Výraz pro c_i (carry out - přenos do vyššího řádu)

$$s_i = \bar{y}_i \cdot \bar{x}_i \cdot c_{i-1} + \bar{y}_i \cdot x_i \cdot \bar{c}_{i-1} + y_i \cdot \bar{x}_i \cdot \bar{c}_{i-1} + y_i \cdot x_i \cdot c_{i-1}$$
$$= \bar{y}_i \cdot (\bar{x}_i \cdot c_{i-1} + x_i \cdot \bar{c}_{i-1}) + y_i \cdot (\bar{x}_i \cdot \bar{c}_{i-1} + x_i \cdot c_{i-1})$$
$$= \bar{y}_i \cdot (x_i \oplus c_{i-1}) + y_i \cdot \overline{(x_i \oplus c_{i-1})}$$
$$= y_i \oplus x_i \oplus c_{i-1}$$

$$c_i = x_i \cdot y_i + x_i \cdot c_{i-1} + y_i \cdot c_{i-1}$$

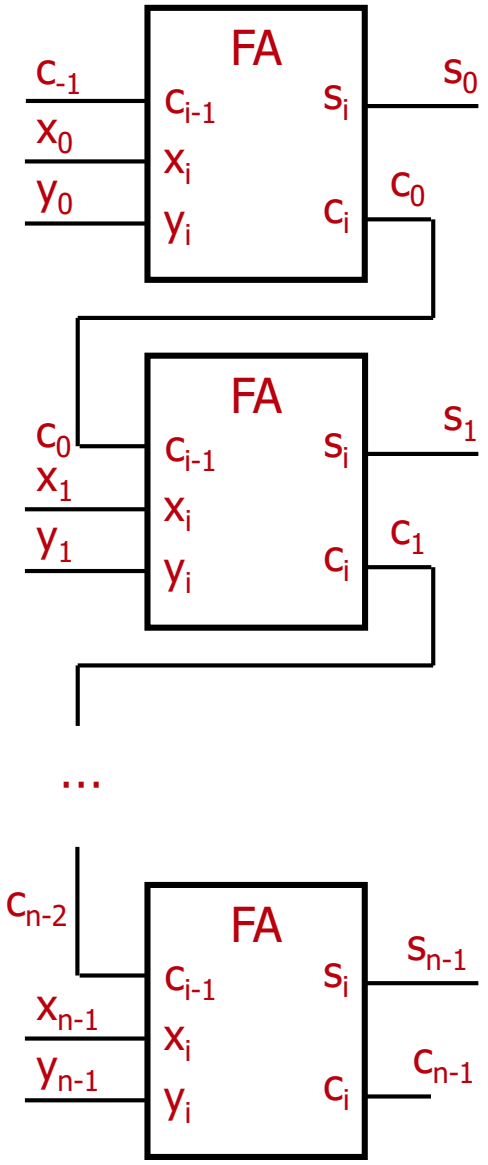


c_{i-1}	x_i	y_i	s_i	c_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



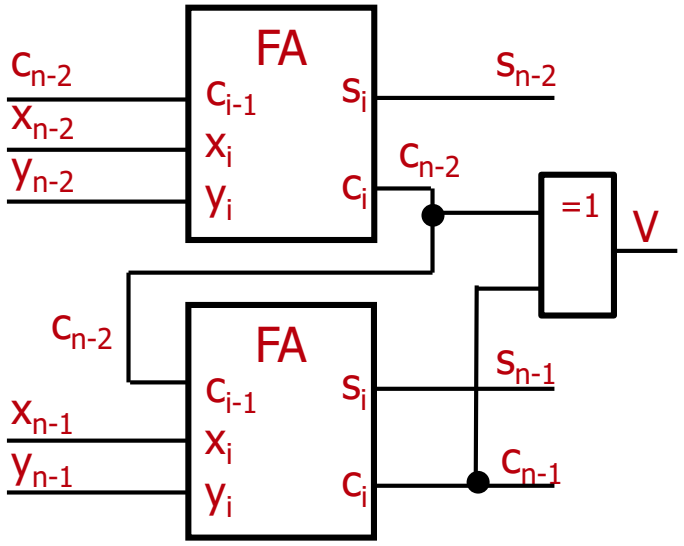
Binární sčítačka vícebitová

- Vícebitová sčítačka
  - Kaskádní zapojení jednobitových sčítaček
  - Anglicky „ripple carry” (přenos se šíří jednotlivými FA)
  - Carry musí procházet (propagate) přes všechny stupně sčítačky
  - Cenově výhodné řešení
  - Pomalé - zpoždění je úměrné počtu bitů
- Urychlení
  - Paralelní struktura
  - Se zrychleným přenosem (Carry Look Ahead), atd.



Detekce přetečení: Verze 1

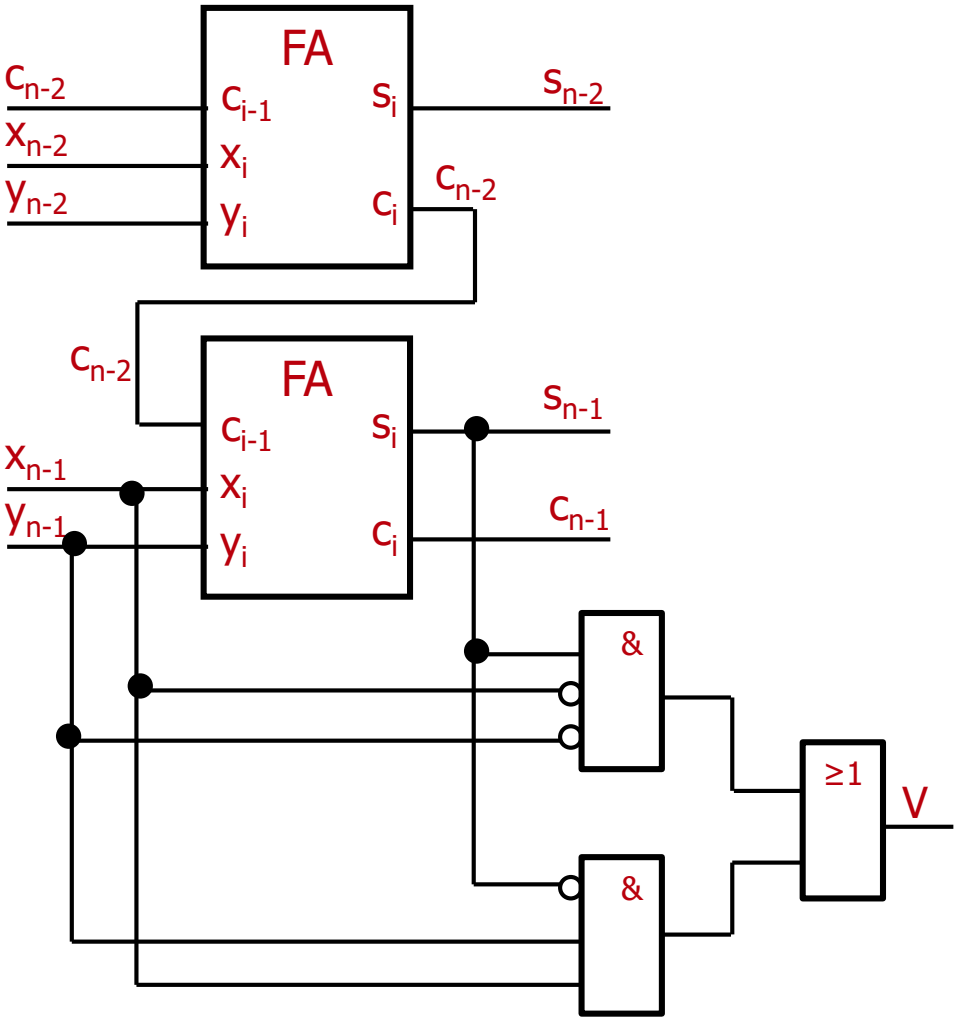
- Detekce přetečení (overflow) na základě shody přenosu do a ze znaménkového bitu
- Poznámka
  - Předpokládáme, že čísla jsou reprezentována ve dvojkovém doplňku
- Tabulka shrnující stav po provedení součtu dvou operandů



Součet	Znaménko	Přenos	Přetečení	Podmínka
$(+X)+(+Y)$	0 1	0 0	0 1	$(X+Y) \leq 2^{n-1}-1$ $(X+Y) > 2^{n-1}-1$
$(-X)+(-Y)$	1 0	1 1	0 1	$-(X+Y) \geq -2^{n-1}$ $-(X+Y) < -2^{n-1}$
$(+X)+(-Y)$ $(-X)+(+Y)$	0 1	1 0	0 0	$X \leq Y$ $X > Y$

Detekce přetečení: Verze 2

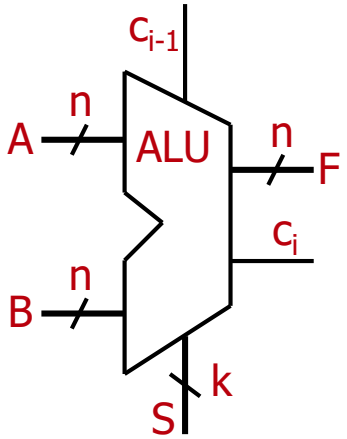
- Detekce přetečení na základě porovnání znaménkových bitů sčítanců a výsledku součtu
  - Součet kladných, resp. záporných, operandů nemůže být záporný, resp. kladný
- Poznámka
  - Předpokládáme, že čísla jsou reprezentována ve dvojkovém doplňku



Aritmetická a logická jednotka (ALU): Příklad

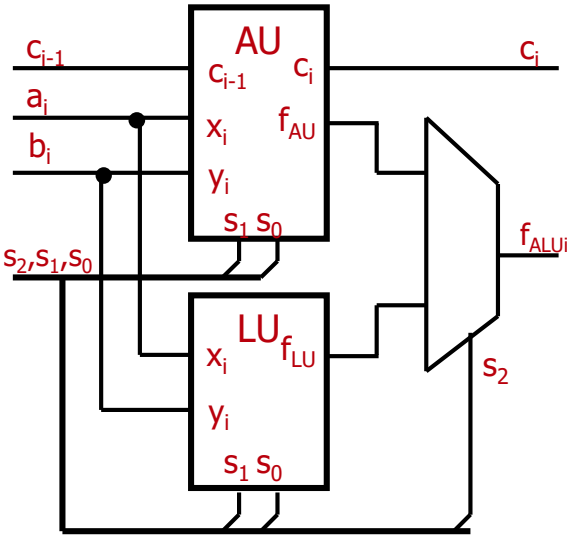
- Arithmetic Logic Unit
  - Základní stavební komponenta počítačů
- Příklad návrhu
  - Dva n-bitové operandy A a B
  - K-bitový vektor S, který vybírá operaci nad operandy A a B
- Funkční tabulka
- Logická značka

s_2	s_1	s_0	Funkce	Popis
0	0	0	F=A+B	Součet (Add)
0	0	1	F=A-B	Rozdíl (Subtract)
0	1	0	F=A+1	Přičtení jedničky (Increment)
0	1	1	F=A-1	Odečtení jedničky (Decrement)
1	0	0	F=A AND B	Logický součin
1	0	1	F=A OR B	Logický součet
1	1	0	F=NOT(A)	Negace
1	1	1	F=A XOR B	Nonekvivalence

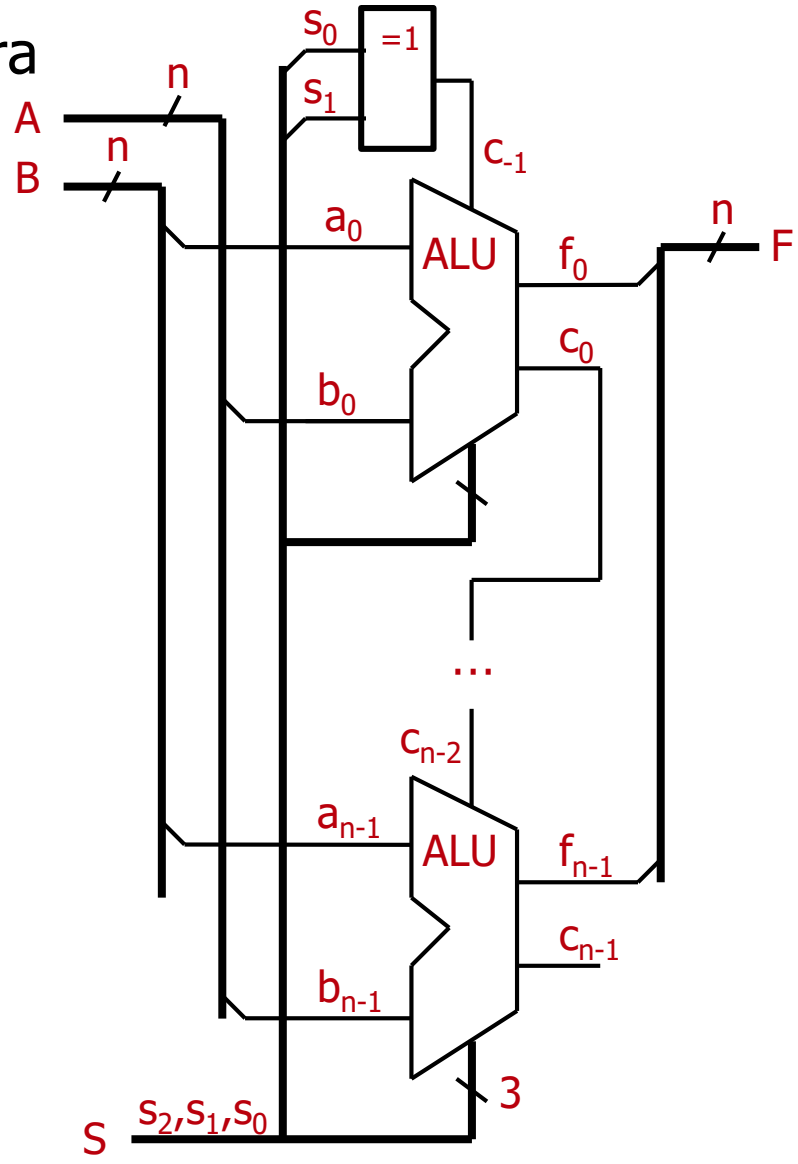


Aritmetická a logická jednotka (ALU): Příklad

- Při návrhu budeme postupovat shora dolů
  - N-bitovou ALU sestavíme z n 1bitových ALU
  - AU – aritmetická jednotka
  - AL – logická jednotka

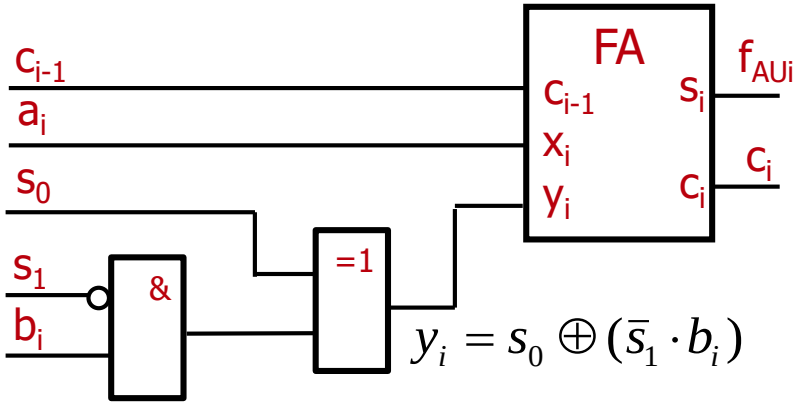


- Poznámka: Carry in do nejnižšího řádu ALU (c_{-1}) se generuje pomocí log. členu XOR, viz následující slajd



Aritmetická a logická jednotka (ALU): Příklad

- Jednobitová aritm. jednotka. (AU)
  - Odčítání = přičítání dvojkového doplňku (Subtract, Decrement)
  - Dvojkový doplněk = jedničkový doplněk (negace všech bitů) + 1
  - Bitová negace se realizuje členem XOR
  - Přičítání +1 = carry in (c_{-1}=1)



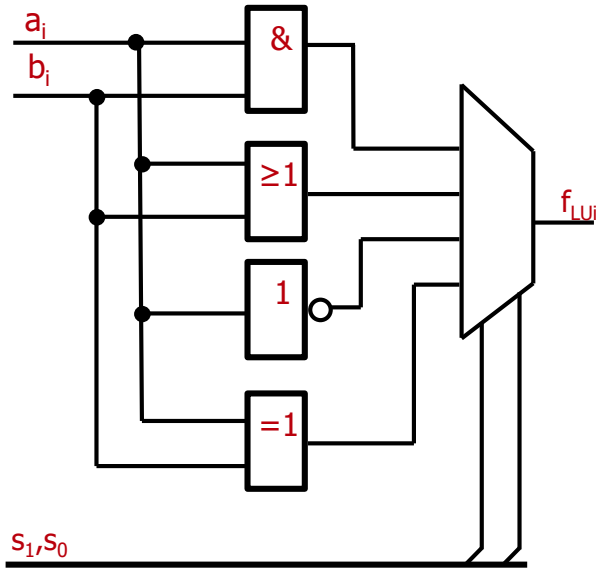
$$F = A - B = A + [B]_2$$
$$= A + (\bar{b}_{n-1} \ldots \bar{b}_0) + 1$$

$$F = A - 1 = A + (-1)$$
$$= A + [0 \ldots 1]_2 = A + (1 \ldots 1) + 0$$

s_1	s_0	b_i	y_i	c_{-1}	Funkce	Popis
0 0	0 0	0 1	$0=b_i$ $1=b_i$	0 0	F=A+B	Add (A+B+0)
0 0	1 1	0 1	$1=b'_i$ $0=b'_i$	1 1	$F=A+[B]_2$	Subtract (A-B=A + dvojkový doplněk B =A + (not B) + 1)
1 1	0 0	0 1	0 0	1 1	F=A+1	Increment (A+0+1)
1 1	1 1	0 1	1 1	0 0	$F=A+[1]_2$	Decrement (A-1=A + dvojkový doplněk (+1) =A+ (not 1 + 1) + 0)

Aritmetická a logická jednotka (ALU): Příklad

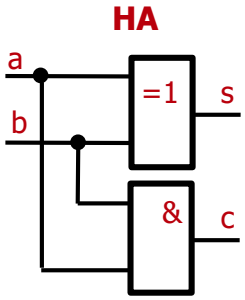
- Jednobitová logická jednotka (LU)
  - Pravdivostní tabulka
  - Logické schéma s multiplexorem

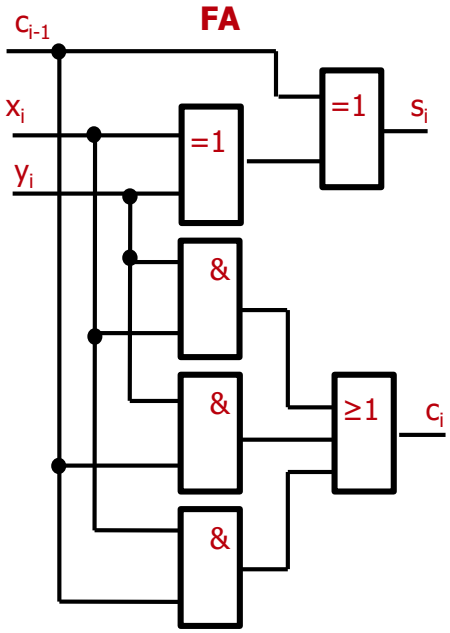


s_1	s_0	a_i	b_i	f_{LUi}	Funkce
0	0	0	0	0	F=A AND B
0	0	0	1	0	
0	0	1	0	0	
0	0	1	1	1	
0	1	0	0	0	F=A OR B
0	1	0	1	1	
0	1	1	0	1	
0	1	1	1	1	
1	0	0	0	1	F=NOT(A)
1	0	0	1	1	
1	0	1	0	0	
1	0	1	1	0	
1	1	0	0	0	F=A XOR B
1	1	0	1	1	
1	1	1	0	1	
1	1	1	1	0	

Násobička: Čísla bez znaménka

- S použitím poloviční (HA) a úplných (FA) 1bitových sčítaček
- Zpoždění?





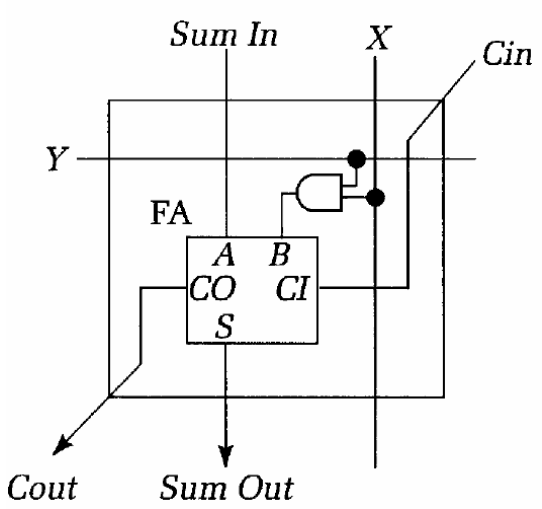
$$(1011)_2 = (11)_{10}$$
$$(1101)_2 = (13)_{10}$$

					1	0	1	1
				x	1	1	0	1
					1	0	1	1
				0	0	0	0	
			1	0	1	1		
		1	0	1	1			
$(10001111)_2$								
$= (143)_{10}$	1	0	0	0	1	1	1	1

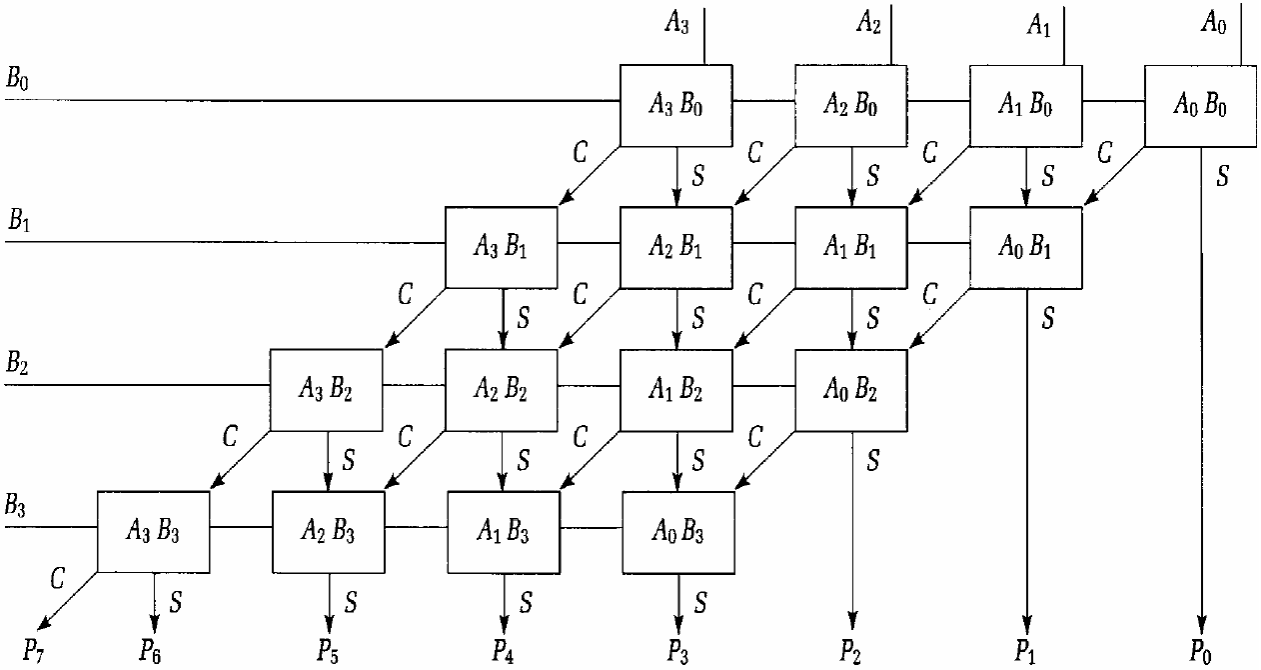
[MIT OpenCourseWare]

Násobička: Čísla bez znaménka

- S použitím jednobitové násobičky
- Zpoždění?

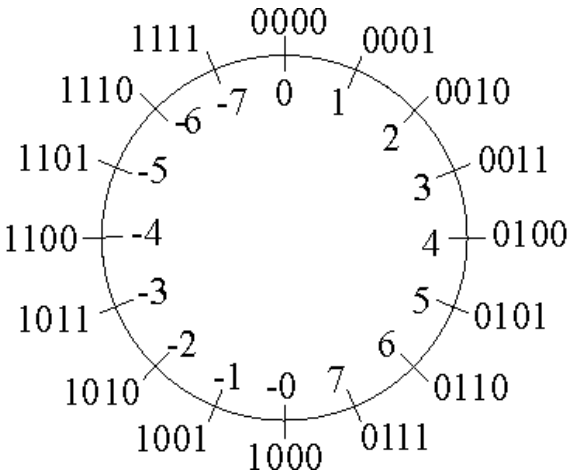


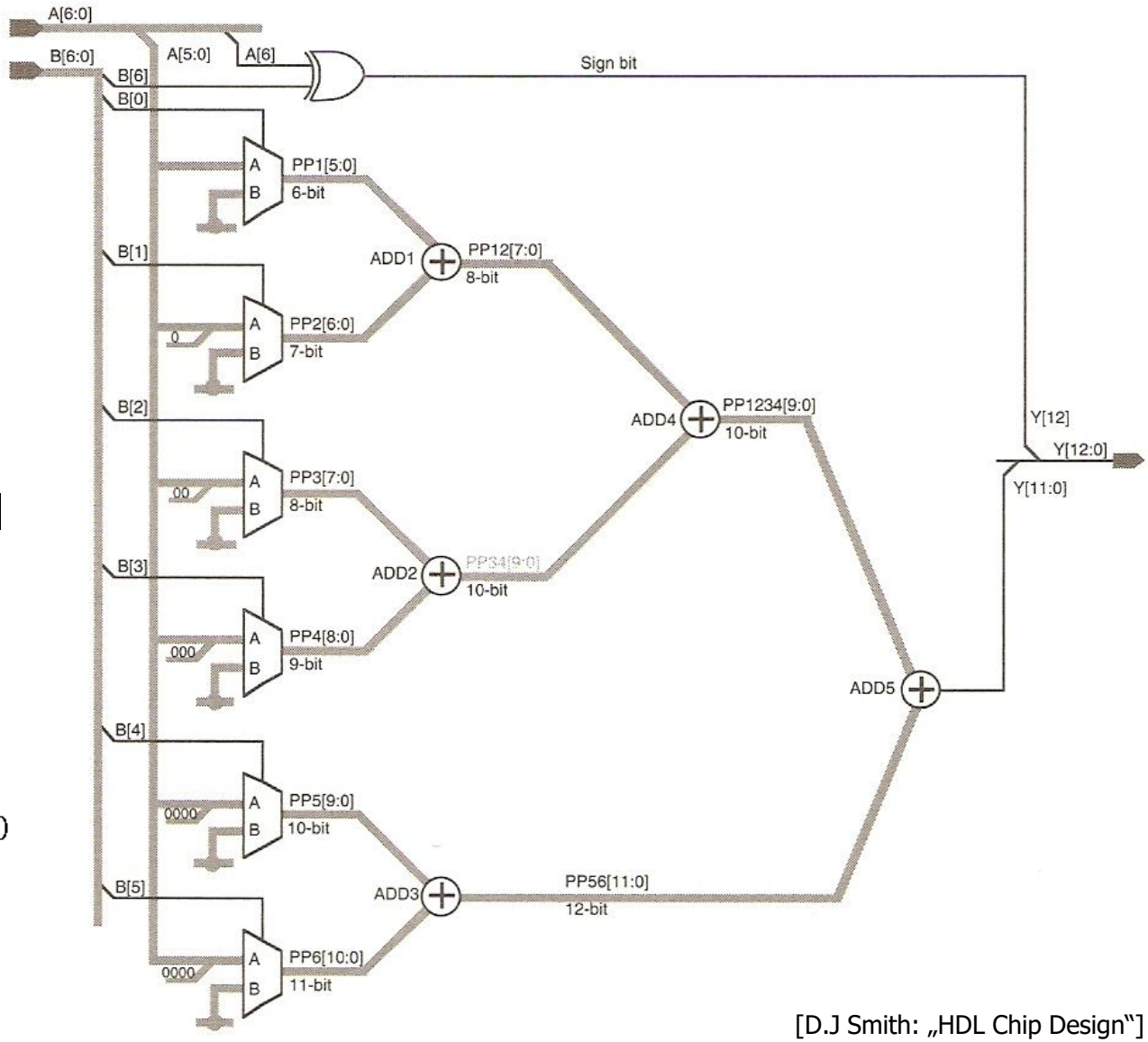
$$
\begin{array}{cccccccc}
 & & & a_3 & a_2 & a_1 & a_0 & \\
 & & & b_3 & b_2 & b_1 & b_0 & \\
 & & & a_3 \cdot b_0 & a_2 \cdot b_0 & a_1 \cdot b_0 & a_0 \cdot b_0 & \\
 & & a_3 \cdot b_1 & a_2 \cdot b_1 & a_1 \cdot b_1 & a_0 \cdot b_1 & & \\
 & a_3 \cdot b_2 & a_2 \cdot b_2 & a_1 \cdot b_2 & a_0 \cdot b_2 & & & \\
a_3 \cdot b_3 & a_2 \cdot b_3 & a_1 \cdot b_3 & a_0 \cdot b_3 & & & & \\
\hline
s_7 & s_6 & s_5 & s_4 & s_3 & s_2 & s_1 & s_0
\end{array}
$$



Násobička: Čísla v přímém kódu se znaménkem

- Výsledek
  - Součin čísel bez MSB + znaménko
- Znaménko
  - XOR znamének operandů - součin kladných i záporných čísel je kladný





[D.J Smith: „HDL Chip Design"]

Násobička: Čísla ve dvojkovém doplňku

- Kladný násobitel
  - Postup je shodný jako u čísel bez znaménka
  - Je třeba šířit znaménko – správně reprezentovat mezivýsledky jako čísla ve dvojkovém doplňku

$1101 = (-3)_{10}$, $0010 = (2)_{10}$

				1	1	0	1
				0	0	1	0
	0	0	0	0	0	0	0
	1	1	1	1	0	1	
	0	0	0	0	0		
	0	0	0	0			
0	1	1	1	1	0	1	0

$1101 = 111101$

$= (-3)_{10}$

$(1111010)_2$

$= (-6)_{10}$

- Záporný násobitel
  - Varianta 1 – obrátíme znaménka násobence i násobitele (dvojkový doplněk)

$$0010 \times 1101 = 1110 \times 0011$$
$$2 \times (-3) = (-2) \times 3$$

$1110 = (-2)_{10}$, $0011 = (3)_{10}$

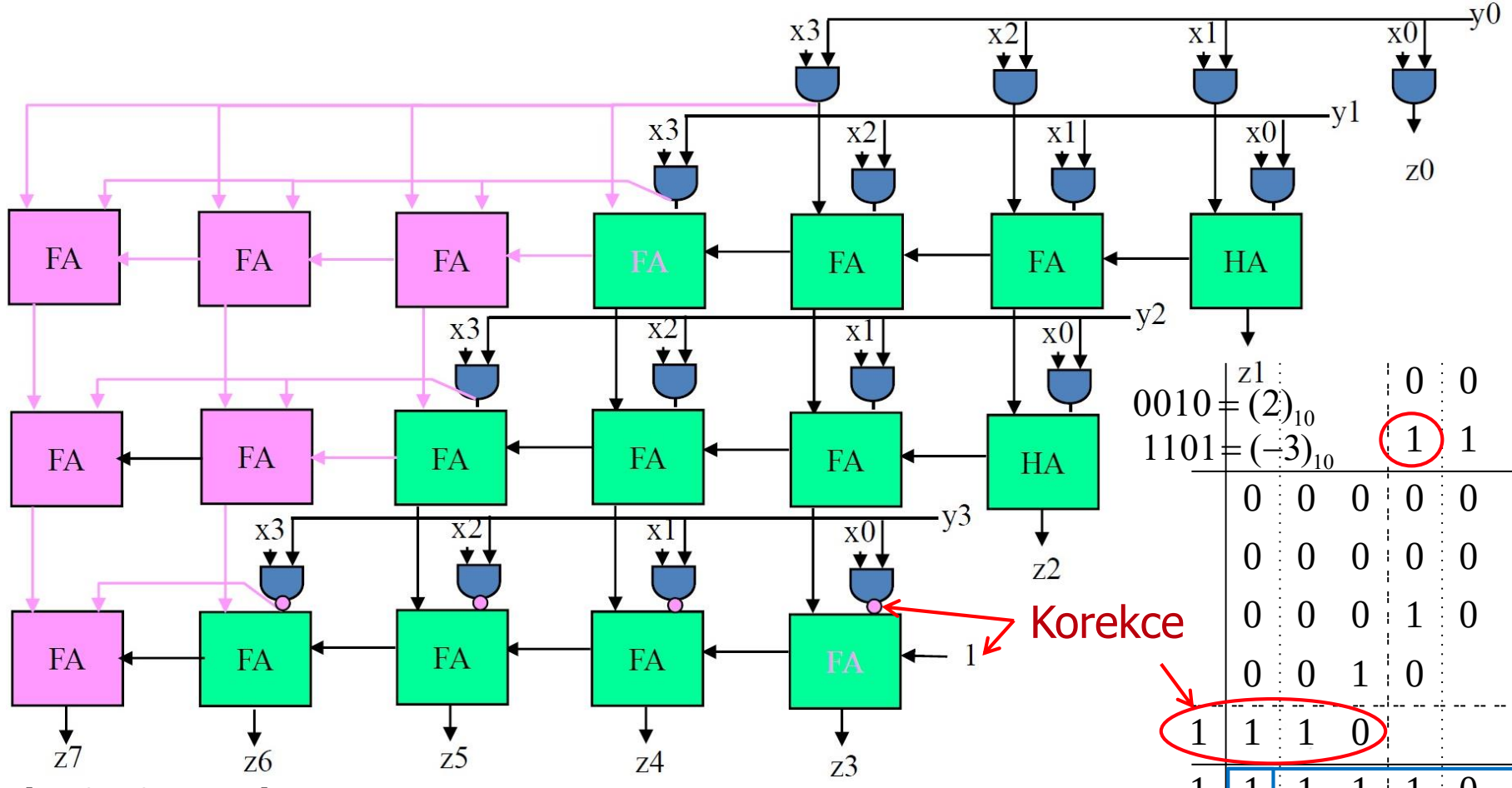
				1	1	1	0
				0	0	1	1
	1	1	1	1	1	1	0
	1	1	1	1	1	0	
	0	0	0	0	0		
	0	0	0	0			
1	1	1	1	1	0	1	0

$(1111010)_2$

$= (-6)_{10}$

Násobička: Čísla ve dvojkovém doplňku

- Záporný násobitel
  - Varianta 2 - v případě násobení znaménkovým bitem provedeme korekci výsledku přičtením dvojkového doplňku násobence



$0010 = (2)_{10}$

$1101 = (-3)_{10}$

				0	0	1	0
				1	1	0	1
0	0	0	0	0	1	0	
0	0	0	0	0	0		
0	0	0	1	0			
0	0	1	0				
1	1	1	0				
1	1	1	1	1	0	1	0

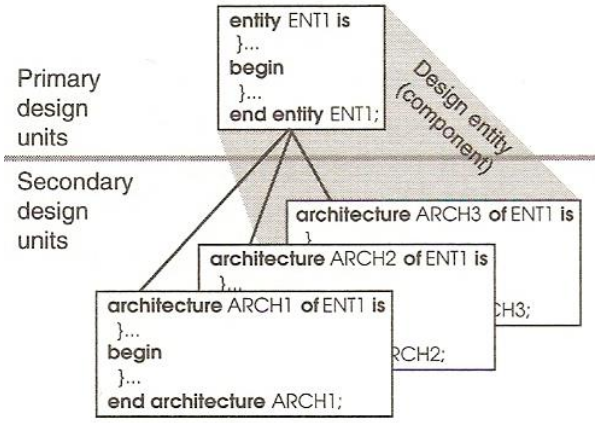
[MIT OpenCourseWare]

Jazyk VHDL: Příklad - specifikace MUX 2-1

- Deklarace entity (rozhraní)

```
library ieee;
use ieee.std_logic_1164.all;
entity mux_using_with is
    port (
    din_0    :in  std_logic;-- input 0
    din_1    :in  std_logic;-- input 1
    sel      :in  std_logic;-- Select inp
    mux_out  :out std_logic -- Mux output
    );
end entity;
```

- Entita - může mít více architektur (popisů funkce)



- Behaviorální popis
  - Příkaz „with-select”

```
architecture behavior of
    mux_using_with is
begin
    with (sel) select
    mux_out <= din_0 when '0',
               din_1 when others;
end architecture;
```

  - Příkaz „when”

```
architecture behavior of
    mux_using_when is
begin
     mux_out <= din_0 when (sel = '0')
    else        din_1;
end architecture;
```

Jazyk VHDL: Příklad - MUX 2-1 (behaviorálně)

- Příkaz „if-then-else”

```
architecture behavior of mux_using_if
    is
begin
    MUX: process (sel, din_0, din_1)
    begin
       if (sel = '0') then mux_out <=
    din_0;
       else mux_out <= din_1;
       end if;
    end process;
end architecture;
```

- Přiřazení hodnoty signálu

```
architecture logic of mux_using_assign
    is
begin
    Y <= (din_0 and not (sel)) or
          (din_1 and sel) ;
end architecture;
```

- Příkaz „case”

```
architecture behavior of mux_using_case is
begin
    MUX: process (sel, din_0, din_1) begin
         case sel is
           when '0'=> mux_out <= din_0;
           when others => mux_out <= din_1;
         end case;
     end process;
end architecture;
```

Jazyk VHDL: Příklad – dekodér (behaviorálně)

- 4bitový dekodér
  - S výstupy aktivními v nule

```
library ieee;
use ieee.std_logic_1164.all;
entity DECODER is
Port(I: in std_logic_vector(1 downto
   0);
    O: out std_logic_vector(3 downto 0)
);
end DECODER;
```

- Architektura verze 1
  - Behaviorálně („when-else")

```
architecture when_else of DECODER is
begin
   O <= "0001" when I = "00" else
        "0010" when I = "01" else
        "0100" when I = "10" else
        "1000" when I = "11" else
        "XXXX";
end when_else;
```

- Architektura verze 2
  - Behaviorálně („case")

```
architecture behv_case of DECODER is
begin
   process (I)
   begin
      case I is
         when "00" => O <= "0001";
         when "01" => O <= "0010";
         when "10" => O <= "0100";
         when "11" => O <= "1000";
         when others => O <= "XXXX";
      end case;
   end process;
end behv_case;
```

Jazyk VHDL: Příklad - prioritní kodér

- Vstup: kód 1 z 8
- Výstup: binární kód

- Knihovny

```
library ieee;
use ieee.std_logic_1164.all;
```

- Entita

```
entity priority_encoder is
   port (sel: in std_logic_vector (7 downto 0);
         code: out std_logic_vector (2 downto
   0));
end priority_encoder;
```

- Architektura
  - Behaviorálně („else-when”)

```
architecture else_when of priority_encoder is
begin
      code = "000" when sel(0) = '1'
      else   "001" when sel(1) = '1'
      else   "010" when sel(2) = '1'
      else   "011" when sel(3) = '1'
      else   "100" when sel(4) = '1'
      else   "101" when sel(5) = '1'
      else   "110" when sel(6) = '1'
      else   "111" when sel(7) = '1'
      else   "xxx";
end else_when;
```

Jazyk VHDL: Příklad - obvod s více výstupy

```
library IEEE;
use IEEE.STD_Logic_1164.all, IEEE.Numeric_STD.all;

entity COMB_1 is
    port ( A, B, C, D:  in   std_logic;
           Y1, Y2:      out std_logic);
end entity COMB_1;

architecture LOGIC of COMB_1 is
begin
    process (A, B, C, D)
        variable ABCD: unsigned(3 downto 0);
    begin
        ABCD := unsigned'(A & B & C & D);
        case ABCD is
            when "0000" => Y1 <= '1'; Y2 <= '0';
            when "0001" => Y1 <= '1'; Y2 <= '0';
            when "0010" => Y1 <= '1'; Y2 <= '0';
            when "0011" => Y1 <= '1'; Y2 <= '0';
            when "0100" => Y1 <= '1'; Y2 <= '0';
            when "0101" => Y1 <= '1'; Y2 <= '0';
            when "0110" => Y1 <= '1'; Y2 <= '0';
            when "0111" => Y1 <= '1'; Y2 <= '0';
            when "1000" => Y1 <= '1'; Y2 <= '0';
            when "1001" => Y1 <= '0'; Y2 <= '1';
            when "1010" => Y1 <= '0'; Y2 <= '1';
            when "1011" => Y1 <= '1'; Y2 <= '1';
            when "1100" => Y1 <= '1'; Y2 <= '0';
            when "1101" => Y1 <= '1'; Y2 <= '0';
            when "1110" => Y1 <= '1'; Y2 <= '0';
            when "1111" => Y1 <= '1'; Y2 <= '0';
        end case;
    end process;
end architecture LOGIC;
```

A	B	C	D	Y1	Y2
0	0	0	0	1	0
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	1	0
0	1	0	0	1	0
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	1	1	0
1	0	0	0	1	0
1	0	0	1	0	1
1	0	1	0	0	1
1	0	1	1	1	1
1	1	0	0	1	0
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	1	0

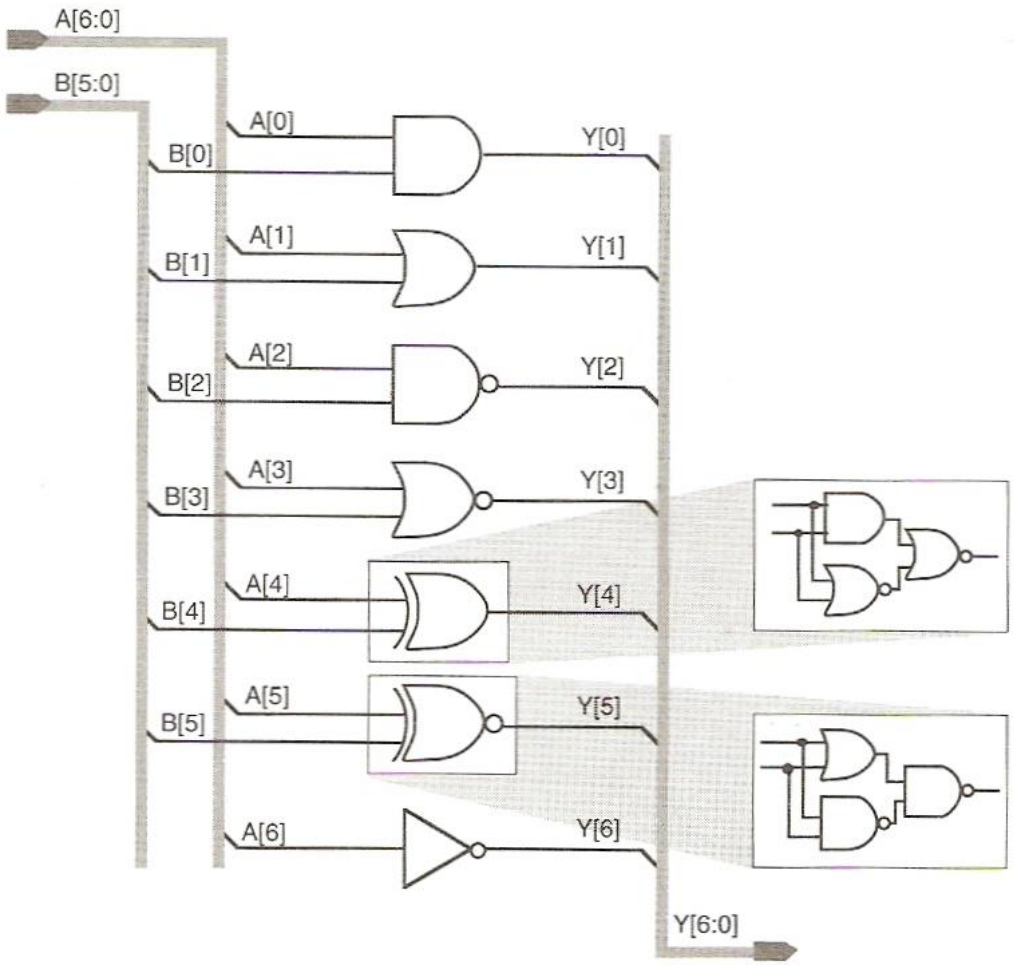
[D.J Smith: „HDL Chip Design"]

Jazyk VHDL: Příklad - logické operace

- Popis ve VHDL

```
library IEEE;
use IEEE.STD_Logic_1164.all, IEEE.Numeric_STD.all;

entity BITWISE is
    port ( A: in  unsigned(6 downto 0);
           B: in  unsigned(5 downto 0);
           Y: out unsigned(6 downto 0) );
end entity BITWISE;

architecture LOGIC of BITWISE is
begin

    process (A, B)
    begin
        Y(0) <= A(0)  and  B(0);   -- Binary AND
        Y(1) <= A(1)   or  B(1);   -- Binary OR
        Y(2) <= A(2) nand  B(2);   -- Binary NAND
        Y(3) <= A(3)  nor  B(3);   -- Binary NOR
        Y(4) <= A(4)  xor  B(4);   -- Binary XOR
        Y(5) <= A(5)  xnor B(5);   -- Binary XNOR
        Y(6) <= not A(6);          -- Unary negation
    end process;

end architecture LOGIC;
```

- Schéma syntetizovaného obvodu

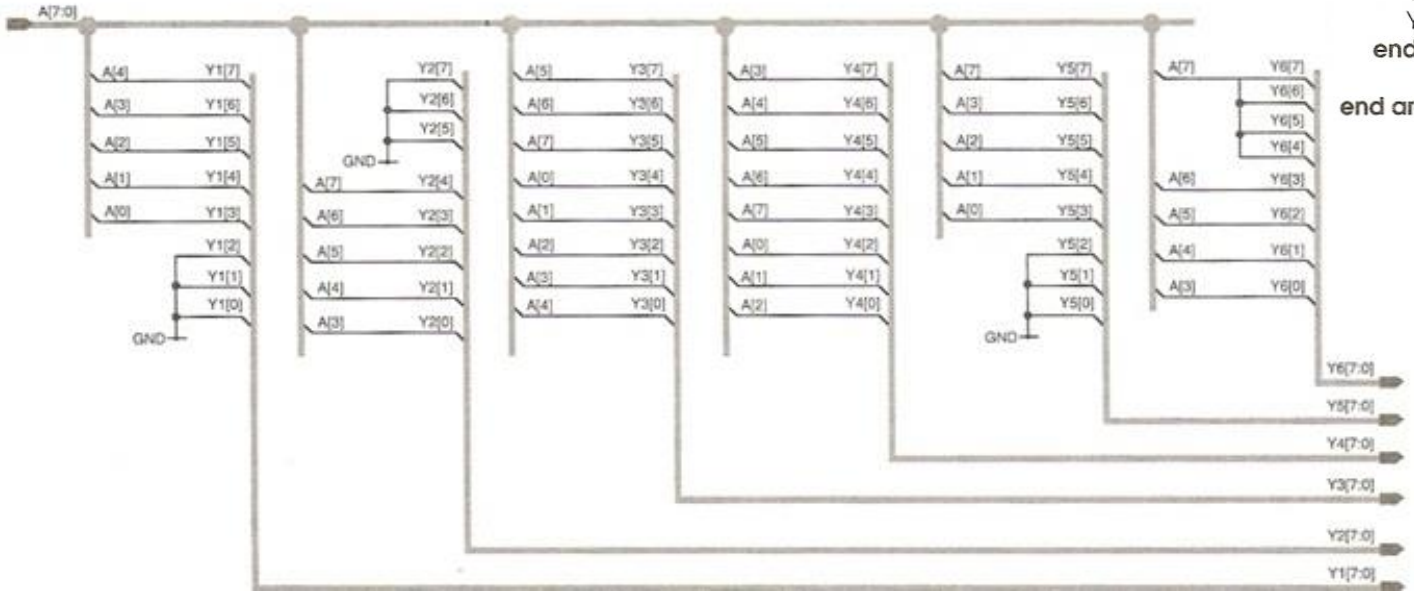


[D.J Smith: „HDL Chip Design"]

Jazyk VHDL: Příklad - rotace a posuvy

- Realizace pouze propojením vodičů
  - SLL - Shift Logical Left
  - SRL - Shift Logical Right
  - SLA - Shift Arithmetic Left
  - SRA - Shift Arithmetic Right
  - ROL - Rotate Logical Left
  - ROR - Rotate Logical Right

```
library IEEE;
use IEEE.STD_Logic_1164.all, IEEE.Numeric_STD.all;

entity SHIFT is
    port ( A:       in  unsigned(7 downto 0);
           Y1, Y2,
           Y3, Y4,
           Y5, Y6:  out unsigned(7 downto 0) );
end entity SHIFT;

architecture LOGIC of SHIFT is
    constant B: integer := 3;
begin

    process (A, B)
    begin
        Y1 <= A sll  B;   -- Logical shift left
        Y2 <= A srl  B;   -- Logical shift right
        Y3 <= A rol  B;   -- Logical rotate left
        Y4 <= A ror  B;   -- Logical rotate right

        Y5 <= A sla B;    -- Arithmetic shift left
        Y6 <= A sra B;    -- Arithmetic shift right
    end process;

end architecture LOGIC;
```



[D.J Smith: „HDL Chip Design"]

Jazyk VHDL: Příklad - relace

- Příklad: n-bitový komparátor
  - (A = B), (A < B), (A > B)
  - Počet bitů je definován jako parametr „generic”
- Knihovny

```
library ieee;
use ieee.std_logic_1164.all;
```

- Entita

```
entity Comparator is
generic(n: natural :=2);
Port(A: in std_logic_vector(n-1 downto
    0);
     B: in std_logic_vector(n-1 downto
    0);
        less: out std_logic;
        equal: out std_logic;
        greater: out std_logic
     );
end Comparator;
```

- Architektura
  - Behaviorálně („if-then”)

```
architecture behv_if of Comparator is
begin
     process(A,B)
     begin
        if (A<B) then
        less <= '1';
        equal <= '0';
        greater <= '0';
    elsif (A=B) then
        less <= '0';
        equal <= '1';
        greater <= '0';
    else
        less <= '0';
        equal <= '0';
        greater <= '1';
    end if;
     end process;
end behv_if;
```

Jazyk VHDL: Příklad - ALU

- 2bitová ALU má 4 operace
  - (A + B), (A – B)
  - (A AND B), (A OR B)
- Knihovny

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
```

- Entita

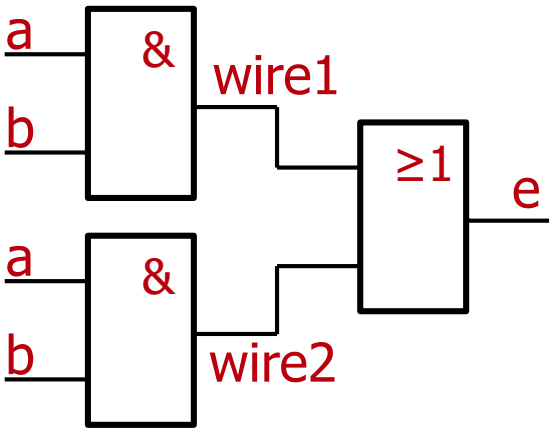
```
entity ALU is
port (A: in std_logic_vector(1 downto 0);
      B: in std_logic_vector(1 downto 0);
      Sel: in std_logic_vector(1 downto 0);
      Res: out std_logic_vector(1 downto 0)
      );
end ALU;
```

- Architektura
  - Behaviorálně („case”)

```
architecture behv_case of ALU is
begin
   process(A, B, Sel)
   begin
   case Sel is
        when "00" =>
           Res <= A + B;
        when "01" =>
           Res <= A + (not B) + 1;
        when "10" =>
           Res <= A and B;
        when "11" =>
           Res <= A or B;
        when others =>
           Res <= "XX";
        end case;
    end process;
end behv_case;
```

Jazyk VHDL: Příklad - strukturní popis

- Mějme obvod



- Příklad specifikace obvodu ve VHDL
  - Předpokládejme, že chování komponent AND a OR je již definováno v knihovně

```
library ieee;
use ieee.std_logic_1164.all;
use work.all;
entity comb_logic is
   port( a,b,c,d : in std_logic;
         e: out std_logic);
end comb_logic;
architecture behv of comb_logic is
component and is
   port (a,b : in std_logic;
   output : out std_logic);
end component;
component or is
   port (a,b : in std_logic;
   output : out std_logic);
end component;
signal wire1, wire2 : std_logic;
begin
   AND1: and port map ( a, b, wire1 );
   AND2: and port map ( c, d, wire2 );
   OR1: or port map ( wire1, wire2, e);
end behv;
```

Jazyk VHDL: Příklad - typy logických úrovní

- std_ulogic

  'U': Neinicializovaná hodnota (signál nebyl ještě buzen)

  'X': Neznámá hodnota - není možno určit, jaké hodnoty nabývá (např. zkrat mezi výstupy generujícími 0 a 1)

  '0': Logická 0 (silný zdroj, např. výstup hradla – malý výstupní odpor)

  '1': Logická 1 (silný zdroj, např. výstup hradla – malý výstupní odpor)

  'Z': Vysoká impedance (např. odpojený vodič)

  'W': Slabá (weak) hodnota, nelze říci, zda 0 či 1 (např. zkrat mezi výstupy generujícími L a H)

  'L': Slabá (weak) hodnota 0 (slabý zdroj - velký výstupní odpor)

  'H': Slabá (weak) hodnota 1 (slabý zdroj - velký výstupní odpor))

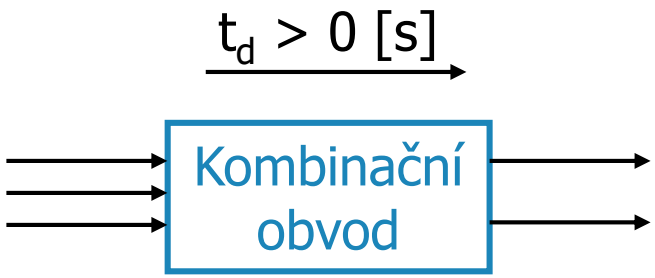
  '-': Je jedno, jaké hodnoty signál nabývá (význam jako X v pravdivostní tabulce, anglicky don't care)

- std_logic

  Pro typ std_ulogic je definována tabulka (tzv. „rezolution table”), která definuje, jakých hodnot bude nabývat signál, pokud bude generován spojenými výstupy více obvodů (více budičů, jeden signál). Implementovaný obvod bude realizovat tzv. montážní logiku, viz dále

Chování logických obvodů v čase

- Každý fyzicky realizovaný obvod má zpoždění t_d
  - Zpoždění je definováno např. jako nejhorší potřebná doba pro vygenerování platných a ustálených logických hodnot výstupních proměnných od okamžiku, kdy vstupní proměnné budou platné a stabilní logické hodnoty
- Teoreticky lze zpoždění zanedbat ($t_d = 0$)
  - Prakticky představuje zpoždění jeden z největších problémů při implementaci (rychlost výpočtů, hazardy – viz dále, vliv prostředí, atd.)

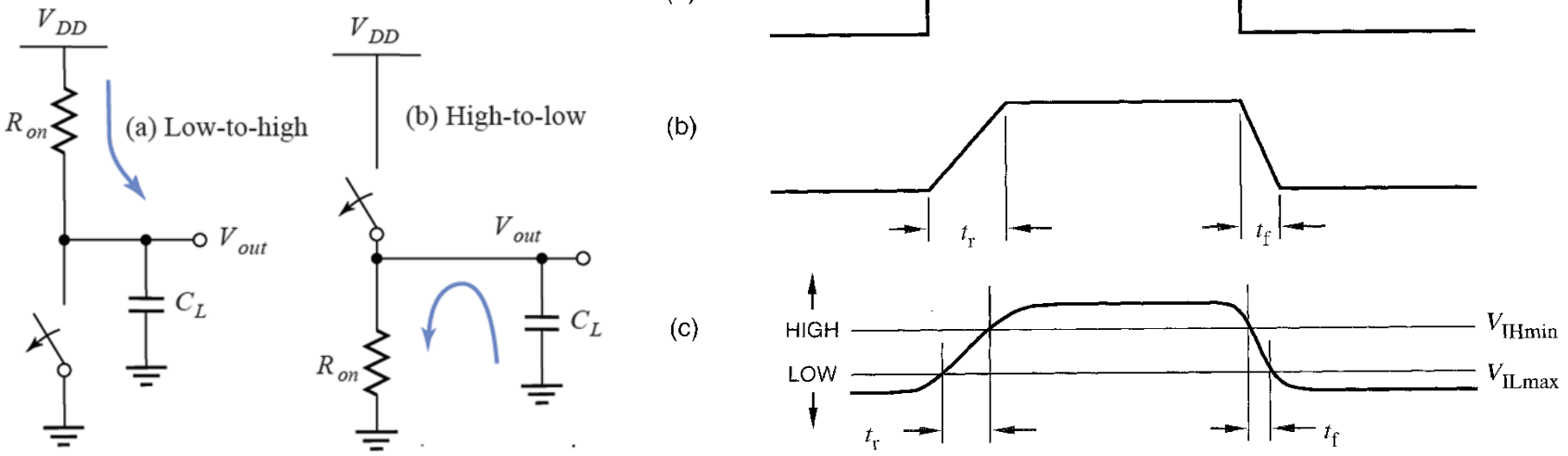


Chování logických obvodů v čase

- Časový průběh
  - Vizualizace hodnot, kterých nabývají logické proměnné (signály, hodnoty napětí) na spojích mezi logickými členy (vodiče) v průběhu času
- Zpoždění logických členů
  - Doba, za kterou se na základě změny vstupní proměnné změní výstupní proměnná
  - Definují se hodnoty: min – typické – max
  - Je třeba navrhovat pro nejhorší případ (nejvyšší povolená teplota, nejmenší napájecí napětí)
- Vzestupná doba (rise/rising time)
  - Doba přechodu z nízké (low, 0) do vysoké (high, 1) úrovně
- Sestupná doba (fall/falling time)
  - Doba přechodu z vysoké (high, 1) do nízké (low, 0) úrovně
- Šířka pulsu (pulse width)
  - Doba, po kterou zůstane hodnota konstantní mezi dvěma změnami

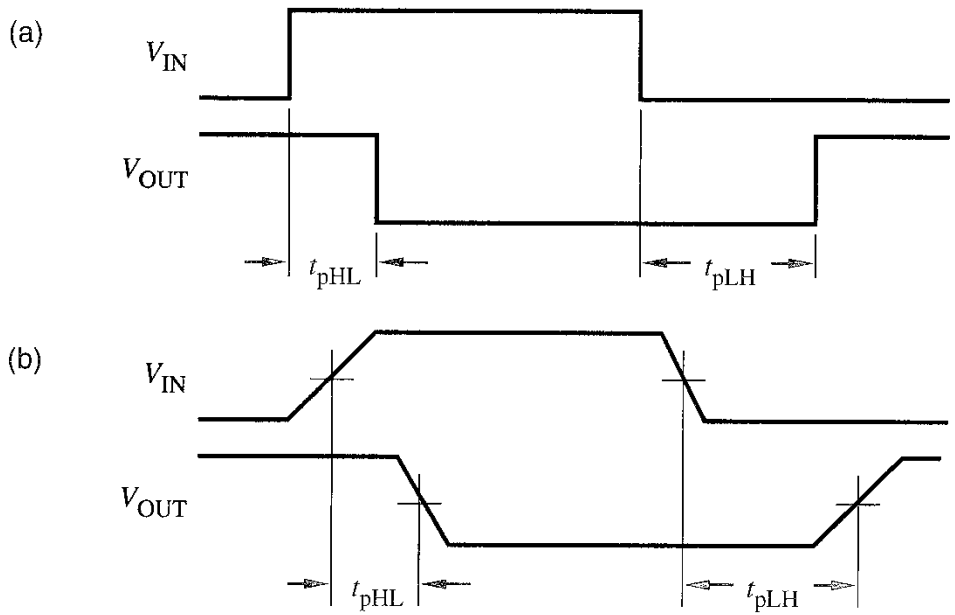
Chování logických obvodů v čase: Hrany

- Doba přechodů mezi log úrovněmi (transition time)
  - Je obecně různá pro přechody 0-1 a 1-0
  - Ideální případ - žádné zpoždění (a)
  - Aproximace průběhu pro analýzu (b)
  - Skutečný průběh napětí (c)
    - tr...doba přechodu z L do H (rise time)
    - tf...doba přechodu z H do L (fall time)
    - Závisí na kapacitě zátěže a na vlastnostech výstupních tranzistorů log. členů



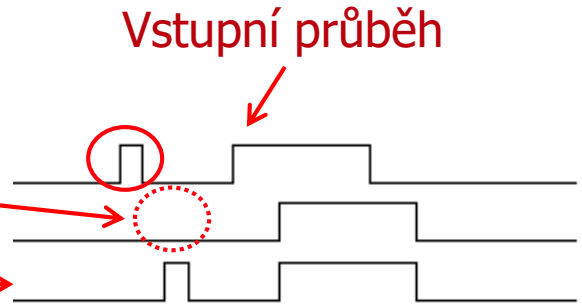
Chování logických obvodů v čase: Zpoždění

- Doba průchodu (propagation time)
  - Je obecně různá pro přechody 0-1 a 1-0
  - Ideální případ se zanedbáním doby přechodů (a)
  - Aproximace – 50 % úrovně signálů (b)
    - t_{pHL}...doba zpoždění ze vstupu na výstup při přechodu z H do L
    - t_{pLH}...doba zpoždění ze vstupu na výstup při přechodu z L do H
    - Závisí na konstrukci obvodu



Chování logických obvodů v čase: Zpoždění

- Inerční (setrvačné) zpoždění (inertial delay)
  - Dáno součtem dob průchodů 0-1 a 1-0
  - Vzniká díky setrvačnosti příslušných elektronických prvků (parazitní kapacity, rychlost tranzistorů apod.)
  - Platí, že puls kratší, než je inerční zpoždění daného prvku, tímto prvkem neprojde
  - Příklad: Puls s délkou trvání 8 ns „neprojde“ obvodem s inerčním zpožděním 10 ns
- Transportní zpoždění (transport delay)
  - Dáno rychlostí šíření signálů v daném médiu
  - V log. systémech se uplatňuje ve spojích mezi jednotlivými log. členy
  - Transportní zpoždění není, na rozdíl od inerčního, v relaci s rychlostí změn signálů (délka pulsů)



Chování logických obvodů v čase: Zpoždění

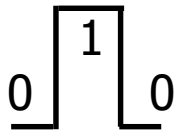
- Každý vodič či elektronický prvek (např. hradlo) má „zpoždění“, díky čemuž může pracovat jako krátkodobá paměť
  - Zpoždění lze využít pro tvorbu struktur, které jsou schopny si „pamatovat“ informaci – jedná se o elementární paměťové prvky – klopné obvody, více viz dále
- Příklad: zpožďovací linka
  - Využívá konečné rychlosti šíření signálů (akustických, elektrických apod.) ve vhodném médiu (rtuť, vodič apod.)
  - Informace vložená do linky na jejím začátku se objeví na jejím konci až za jistou dobu – linka si informaci po jistou dobu „pamatuje“
  - Využíváno např. v prvních elektronických počítačích

Chování logických obvodů v čase: Hazardy

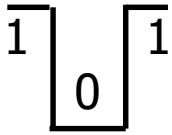
- Při minimalizaci log. výrazů
  - Může dojít k často nežádoucím situacím – tzv. hazardům
- Logická větev
  - Cesta signálu od vstupů k výstupům
  - Pro jednotlivé vstupy se může lišit její délka
- Hazardy jsou způsobeny
  - Nestejným zpožděním log. větví v log. obvodu - různé zpoždění vodičů a log. členů (rozptyly výrobního procesu, teplot, napájecího napětí)
- Druhy
  - Statický
  - Dynamický

Chování logických obvodů v čase: Statický hazard

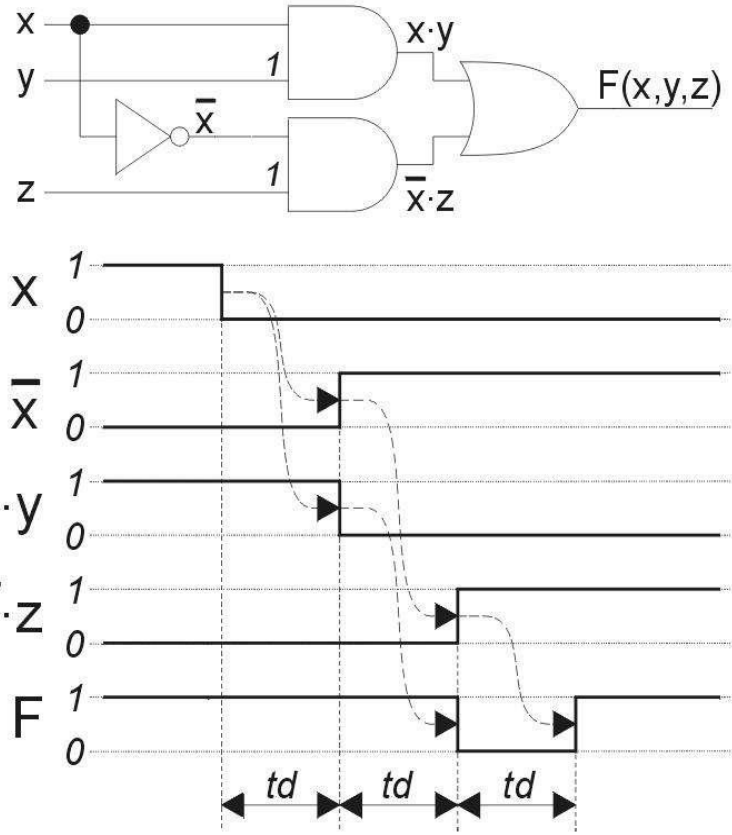
- Vyskytuje se v kombinační logické síti
  - Změna jedné vstupní proměnné způsobí přechodnou změnu výstupní proměnné, která měla být konstantní
- Na výstupu se projeví jako krátký puls
  - ÚNKF: hazard v 0 - puls z log. 0 do log. 1 a zpět



  - ÚNDF: hazard v 1 - puls z log. 1 do log. 0 a zpět



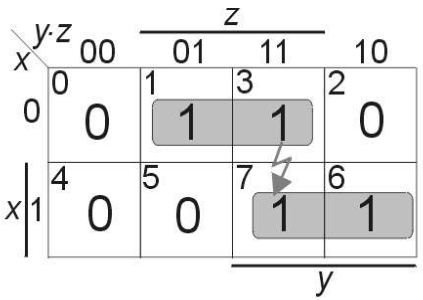
- Příklad ÚNDF - hazard v 1
  - F(x,y,z)=Σm(1,3,6,7)
    $= x.y + \bar{x}.z$

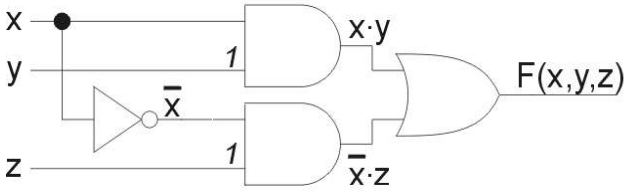


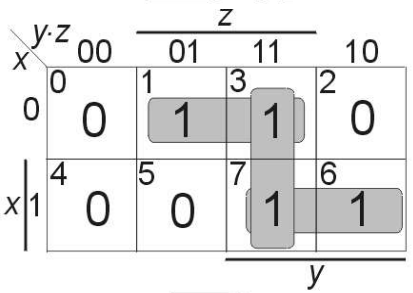
Chování logických obvodů v čase: Statický hazard

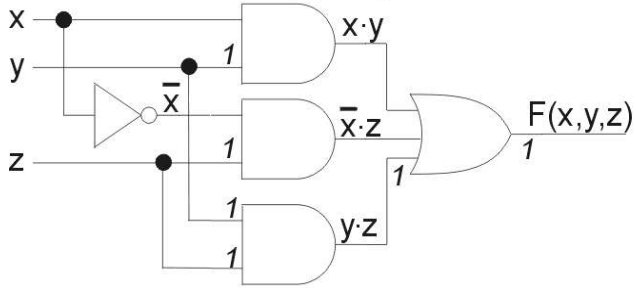
- Eliminace
  - Odstranění takových vstupních kombinací, které se mění v jedné proměnné a zároveň obě produkují stejné hodnoty výstupu log. funkce - v Karnaughově mapě jsou to buňky ležící vedle sebe
  - ÚNDF - použijeme všechny zkrácené (prosté) implikanty
  - ÚNKF - použijeme všechny zkrácené (prosté) implicenty
- Ošetřená realizace již není minimální

- Příklad F(x,y,z)=Σm(1,3,6,7)





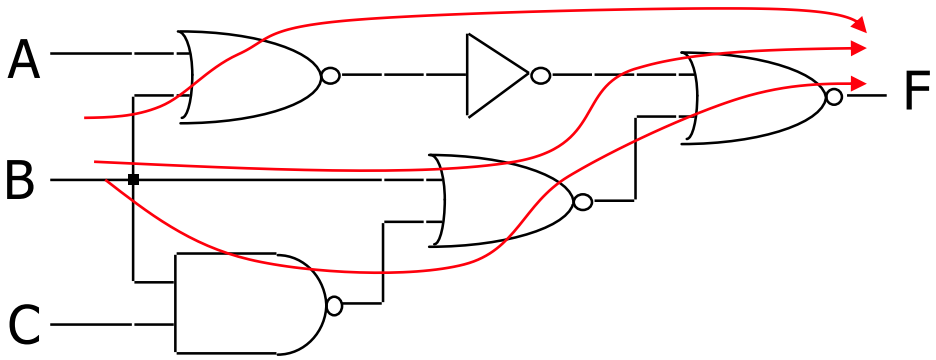




Chování log. obvodů v čase: Dynamický hazard

- Vyskytuje se v kombinační logické síti
  - Změna jedné vstupní proměnné způsobí lichý počet změn výstupní proměnné, která měla mít pouze jednu změnu
  - Vzniká jako superpozice statického hazardu a změny některé další vnitřní proměnné
- Podmínka vzniku
  - Existence nejméně 3 cest v síti mezi příslušnými vstupními a výstupními proměnnými

- Příklad



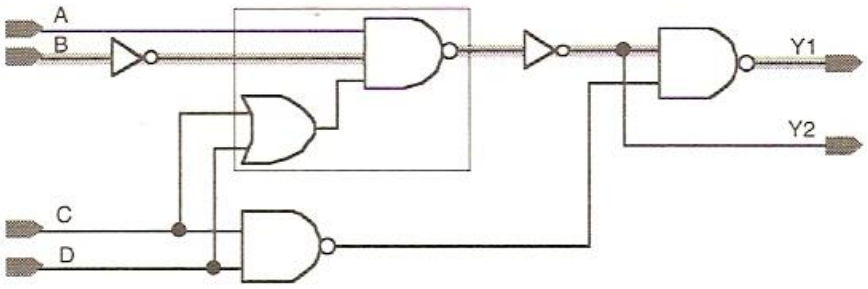
- Eliminace
  - Je třeba odstranit statické hazardy na všech úrovních obvodu - pracné a složité (kritické aplikace)
  - Asynchronní sekvenční obvody - používat pouze sítě se dvěma cestami mezi příslušnými vstupními a výstupními proměnnými
  - Synchronní sekvenční obvody - hazardy ponechat, ale výstup sítě brát jako platný (vzorkování) až po jejich odeznění (více viz dále)
- Příklad
  - Synchronní síť - klopné obvody vzorkují výstupy kombinační logické sítě (KLS) po odeznění přechodových jevů - hazardů

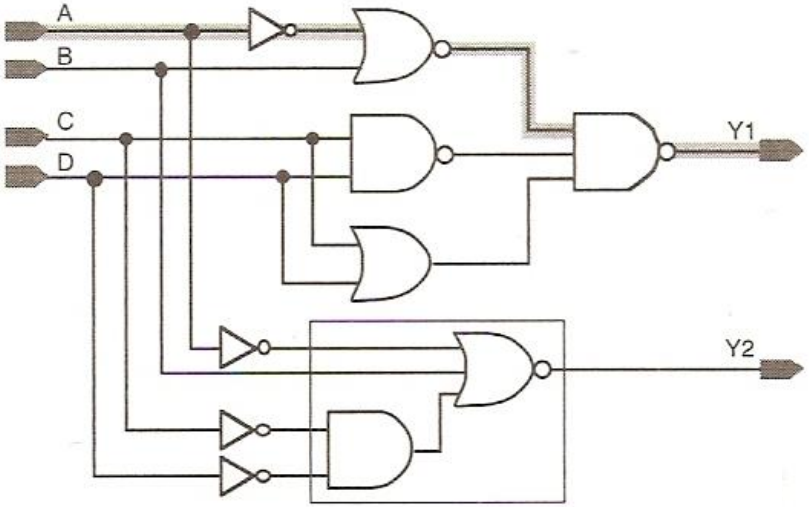
Optimalizace logických obvodů: Kritéria

- Plocha (area)
  - Část čipu, kterou obvod zabírá
  - Je ovlivněna řadou parametrů - počet log. členů (viz minimalizace), rozměry log. členů (technologie výroby, logický zisk, více viz dále), spoje
- Časování (timing)
  - Doba, za kterou se po změně vstupních proměnných ustálí výstupní proměnné obvodu
  - Je ovlivněno řadou parametrů - délka logické větve, logický zisk a zátěž log. členů, parazitní kapacity, délka spojů atd.
- Příkon (power)
  - Příkon obvodu ovlivňuje počet log. členů, pracovní frekvence, velikost napájecího napětí, parazitní kapacity, výrobní proces atd.
- Testovatelnost (test, scan)
  - Logické obvody je třeba navrhovat tak, aby bylo možno otestovat jejich správnou činnost

Optimalizace logických obvodů: Příklad

- Minimální plocha
  - Počet log. členů: 5
  - Plocha: 12 (skutečné rozměry na čipu)
  - Zpoždění: 4 t_d
- Minimální zpoždění
  - Počet log. členů: 10
  - Plocha: 28 (skutečné rozměry na čipu)
  - Zpoždění: 3 t_d





A	B	C	D	Y1	Y2
0	0	0	0	1	0
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	1	0
0	1	0	0	1	0
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	1	1	0
1	0	0	0	1	0
1	0	0	1	0	1
1	0	1	0	0	1
1	0	1	1	1	1
1	1	0	0	1	0
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	1	0