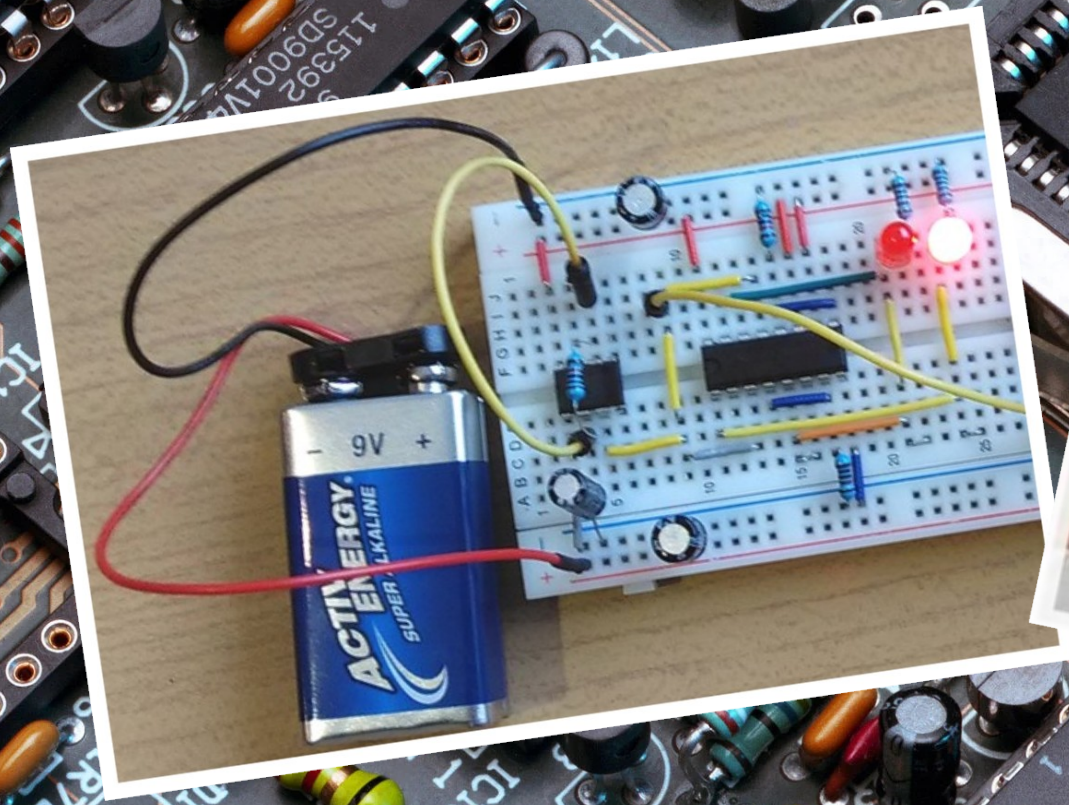
A digitális elektronika alapjai



5. Kombinációs logikai hálózatok – 2. rész

Felhasznált és ajánlott irodalom

- Gulyás Dénes: Számítógép architektúrák *(interaktív jegyzet)*
- Mike Gábor: A digitális elektronika alapjai *(jegyzet és videók)*
- Zalotay Péter: Digitális technika
- Végh János: Ismerkedés a digitális elektronikával
- Mészáros Miklós: Logikai algebra alapjai, logikai függvények I.
- Mingesz Róbert: Digitális technikai tananyagok
- F-alpha.net: Digital Electronics
- Electronics Tutorials: Logic Gates
- M. Morris Mano and Michael D. Ciletti: Digital Design
- Simon Frazer University: CMPT-150: Introduction to Computer Design

Logikai áramkör szimulátorok

- LogiSim szimulátor: www.cburch.com/logisim/
- Falstad.com: Circuit simulator
- CircuitVerse: Simulator
- University of Genoa: Deeds Simulator
- Gatecat: Breadboard Simulator v1.0
- Logic.ly: Logic.ly Simulator (online demo)

A kettes számrendszer

- Hófehérke törpéi a bányászott gyémántokat csomagolják. Akinek már két gyémánt, vagy csomag van a kezében, összecsomagolja, és továbbadja a mellette állónak. Hány gyémánt után jutunk el az alábbi ábrán látható helyzetig?

8 + 4 + 2 + 0 = $\mathbf{14_{10}}$

Kettes számrendszerben csak azt írjuk le, hogy az adott helyiértéken van-e „csomag” (1), vagy nincs (0) = $\mathbf{1110_2}$

Forrás: Ránkyné Szalay Rita, A 2-es számrendszer tanítása interaktív módon

A kettes számrendszer

Dec	Bin	Hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

- **Összeadási példa:** $8_{10} + 7_{10} = 15_{10}$
  Bináris alakban: $1000_2 + 0111_2 = 1111_2$
- **A kettes számrendszer** helyiértékes számrendszer: jobbról balra haladva minden egyes számjegy a 2 eggyel nagyobb hatványát fejezi ki (2^0=1-től kezdve). A kettes számrendszerben ábrázolt szám értékét úgy kapjuk meg, hogy összeadjuk azokat a kettő-hatványokat, amelyek helyiértékénél 1 áll. Például:

$$110011_2 = 1\cdot2^5+1\cdot2^4+0\cdot2^3+0\cdot2^2+1\cdot2^1+1\cdot2^0 = 32+16+2+1 = 51_{10}$$

- **Átváltás maradékos osztással:**

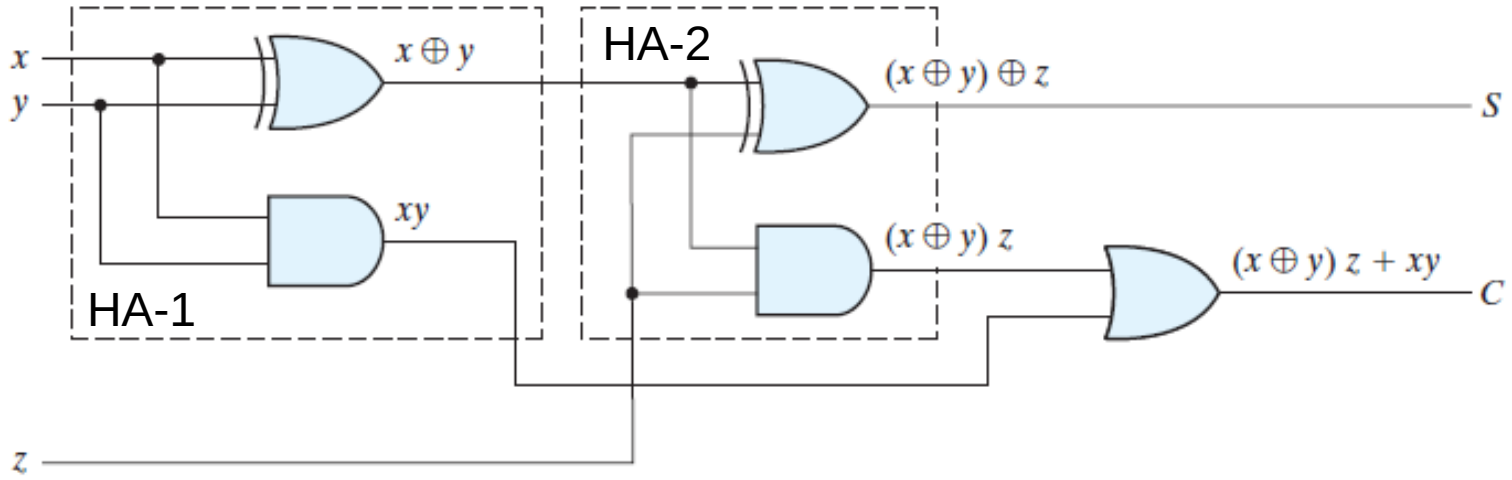
41	: 2	=	20	marad **1**
20	: 2	=	10	marad **0**
10	: 2	=	5	marad **0**
5	: 2	=	2	marad **1**
2	: 2	=	1	marad **0**
1	: 2	=	0	marad **1**

Tehát:
$41_{10} = 101\,001_2$

Forrás: Wikipédia

Ismétlés: A teljes összeadó

- Két **fél-összeadó** összekapcsolásával és egy **VAGY** kapuval is megépíthetjük a **teljes összeadót** (*full adder, FA*)



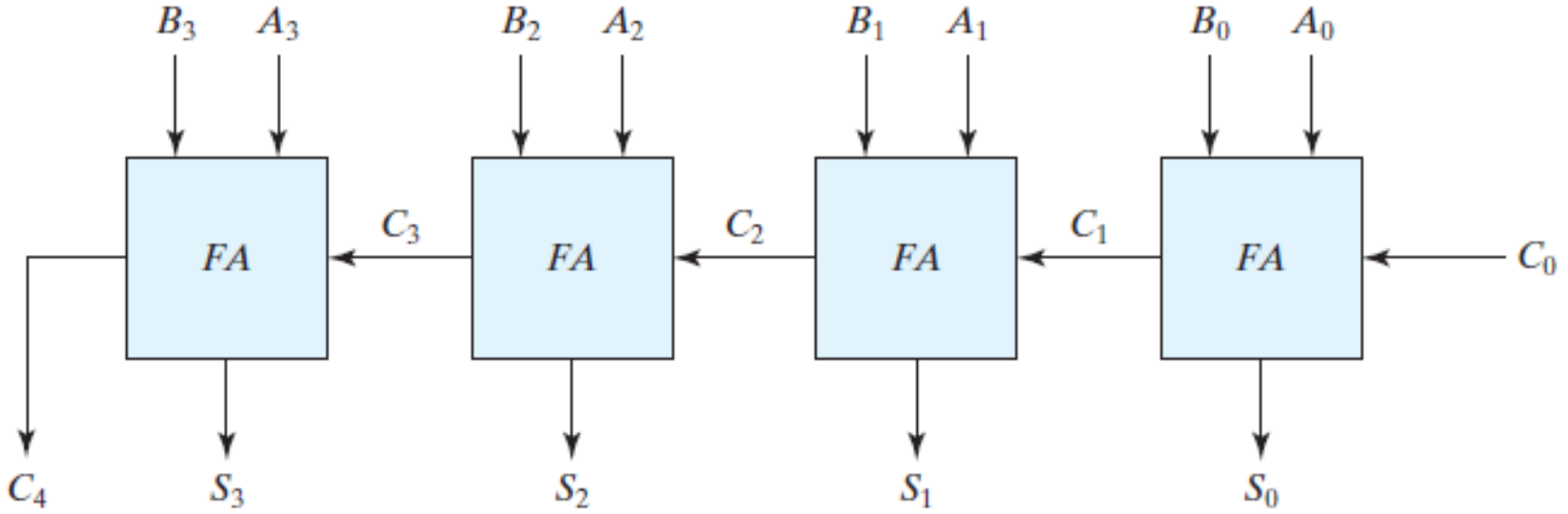
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Az ábra jelöléseivel:
x és ***y*** az összeadandó bitek
z az áthozat

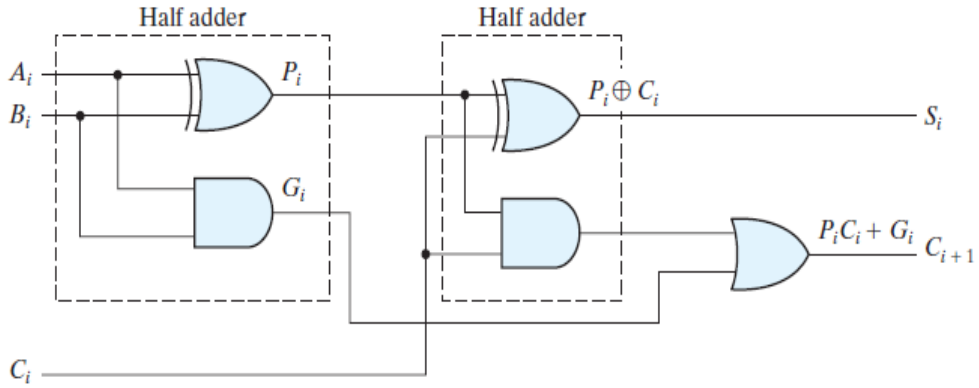
S az összeg
C az átvitel

Többjegyű összeadó

Az előzőekben bemutatott teljes összeadót (full adder) láncbafűzhetjük egy többjegyű bináris összeadó megvalósítására. Az ábrán egy négybites összeadó látható.



Ennek a kapcsolásnak a hátránya az átvitel lassú terjedése, ami a működési sebességet korlátozza. A probléma megoldására speciális kapcsolást dolgoztak ki, az átvitel előre történő kiszámításához (look ahead carry).



$$C_0 = \text{áthozat}$$

$$C_1 = G_0 + P_0C_0$$

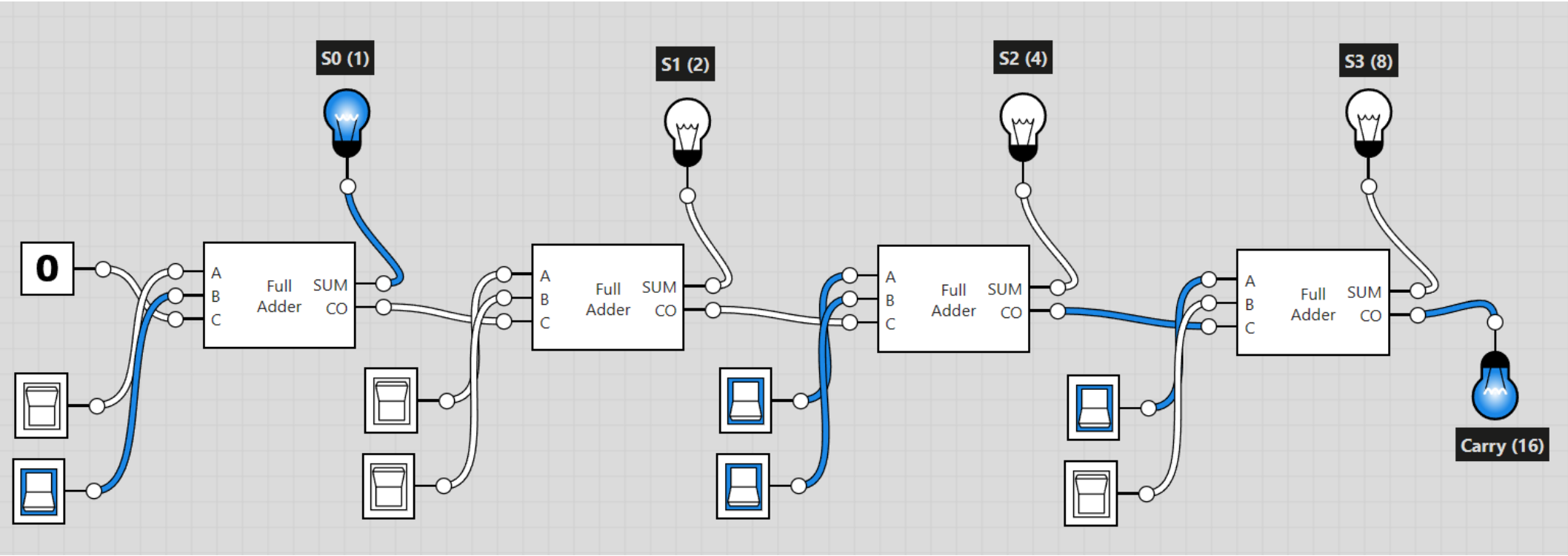
$$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0) = G_1 + P_1G_0 + P_1P_0C_0$$

$$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$$

A négybites összeadó modellezése

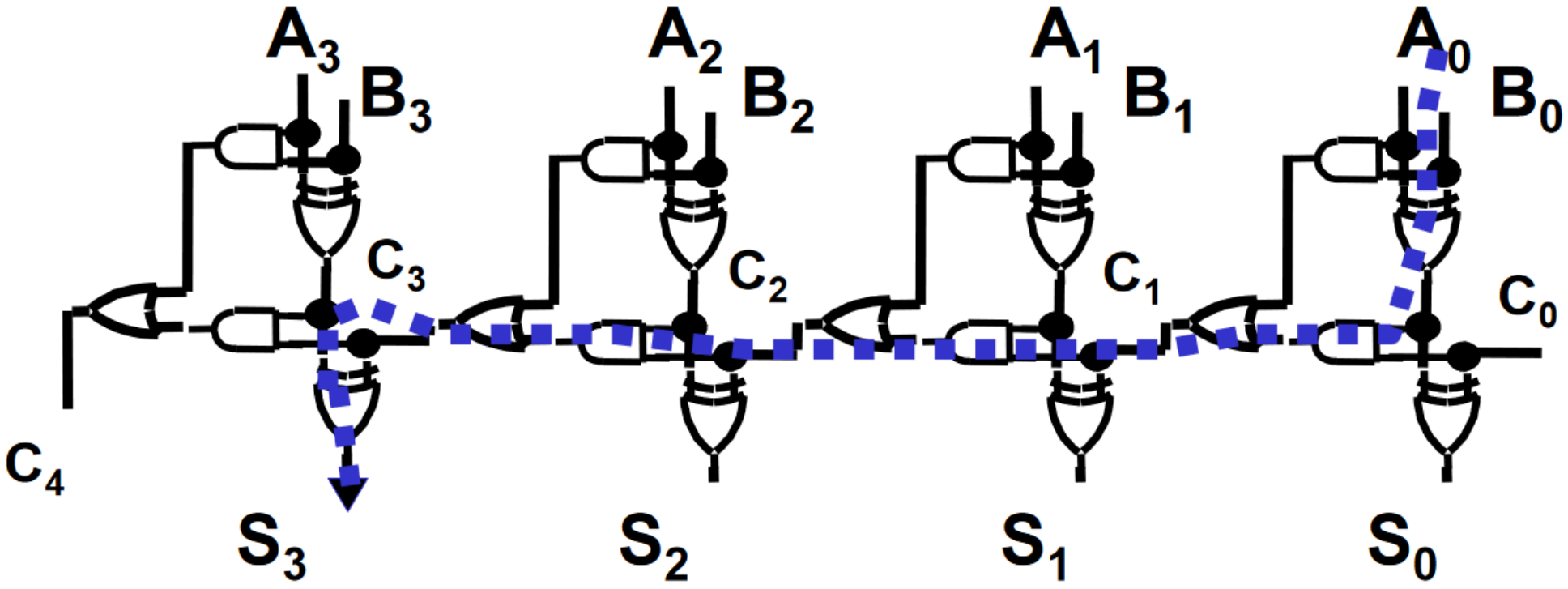
- Négy teljes összeadó felfűzésével egy négybites összeadót építünk
- A legelső áthozat nulla, a legutolsó átvitel 1 értéke pedig 16-ot jelent
- Az ábrán a 12 + 5 = 17 összeadás látható
  (A = 8 + 4; B = 4 + 1; Sum = 1; $Carry_{16}$ = 1)

4bit_adder.logicly



Négybites összeadó

- A leghosszabb terjedési idő itt az **A0** vagy **B0** valamelyikéből és a **C0**-ból származó átvitel feljutása **C4**-re (9 kapu késleltetési idő)
- Több bit esetén a terjedés még lassabb. A probléma speciális kapcsolással, az átvitel előre történő kiszámításával (look ahead carry) oldható meg



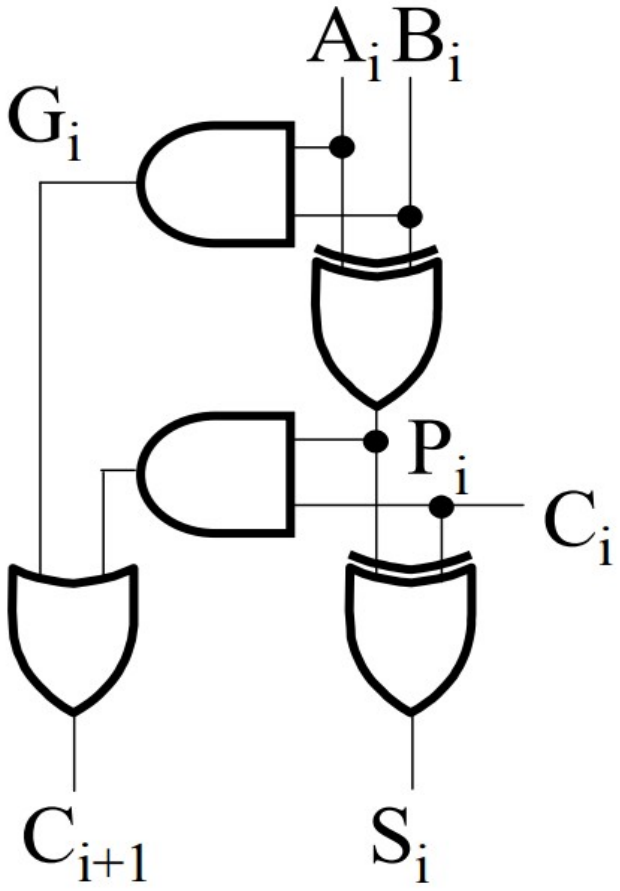
Forrás: Simon Frazer University, CMPT-150: Introduction to Computer Design

Gyors átvitelképzés

- A **gyors átvitelképzés** (Look ahead carry) kiszámításához minden teljes összeadó fokozattól két jelet kérünk: G_i és P_i
- Az i-edik fokozatban kétféle módon keletkezhet átvitel:
  - Ha A_i és B_i egyaránt ’1’, ekkor ***átvitelt generálunk*** (G_i)
  - Ha $P_i = A_i \oplus B_i =$ **’1’** , akkor a C_i áthozat propagálódik (továbbterjed), azaz $C_{i+1} = C_i$ lesz
- Ezt a két feltételt ***generáló*** (G_i) és ***propagáló*** (P_i) feltételnek nevezzük, ill. jelöljük
- A teles összeadó függvényei G_i és P_i-vel kifejezve így néznek ki:

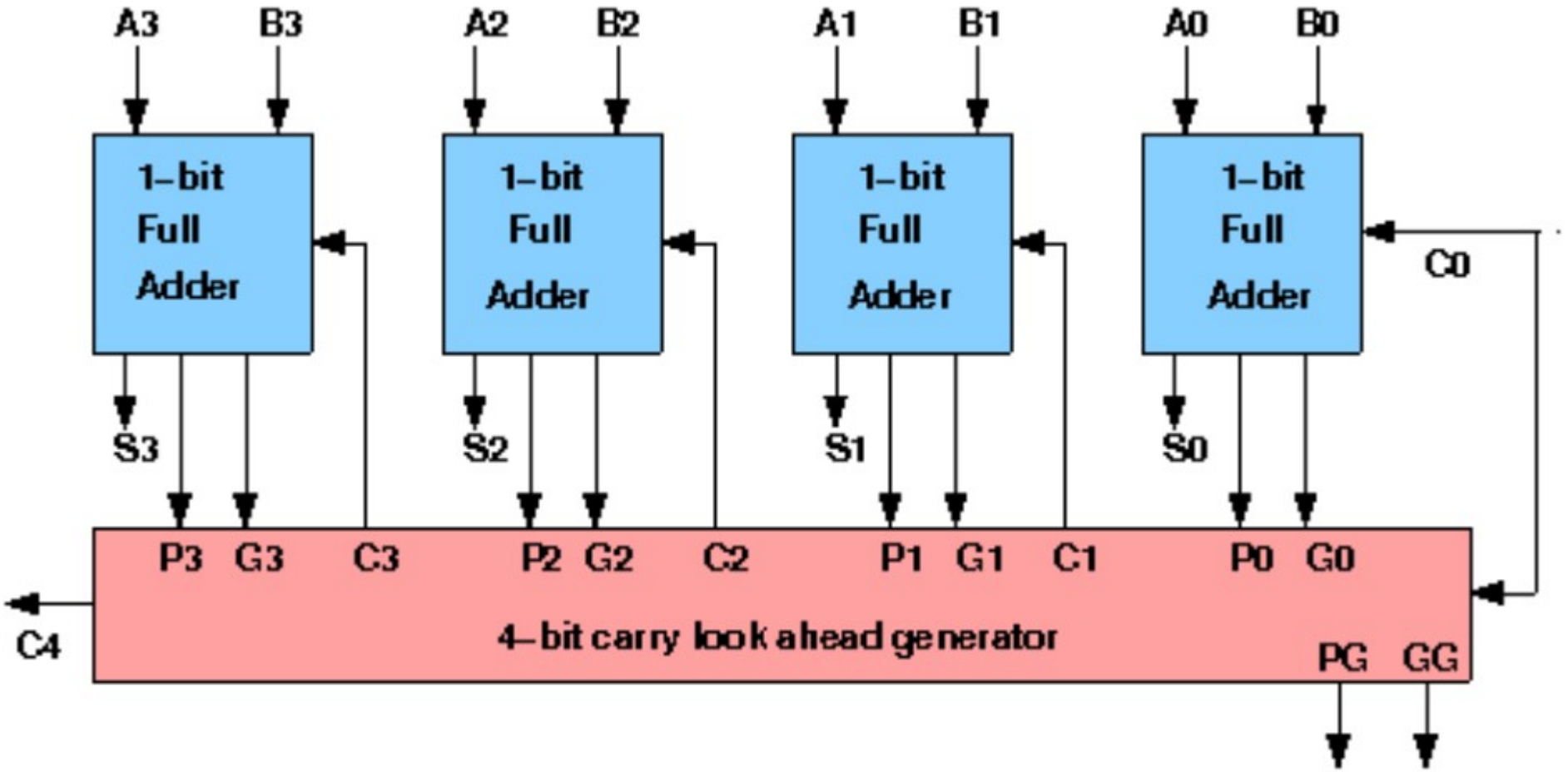
$$P_i = A_i \oplus B_i \qquad G_i = A_i B_i$$

$$S_i = P_i \oplus C_i \qquad C_{i+1} = G_i + P_i C_i$$



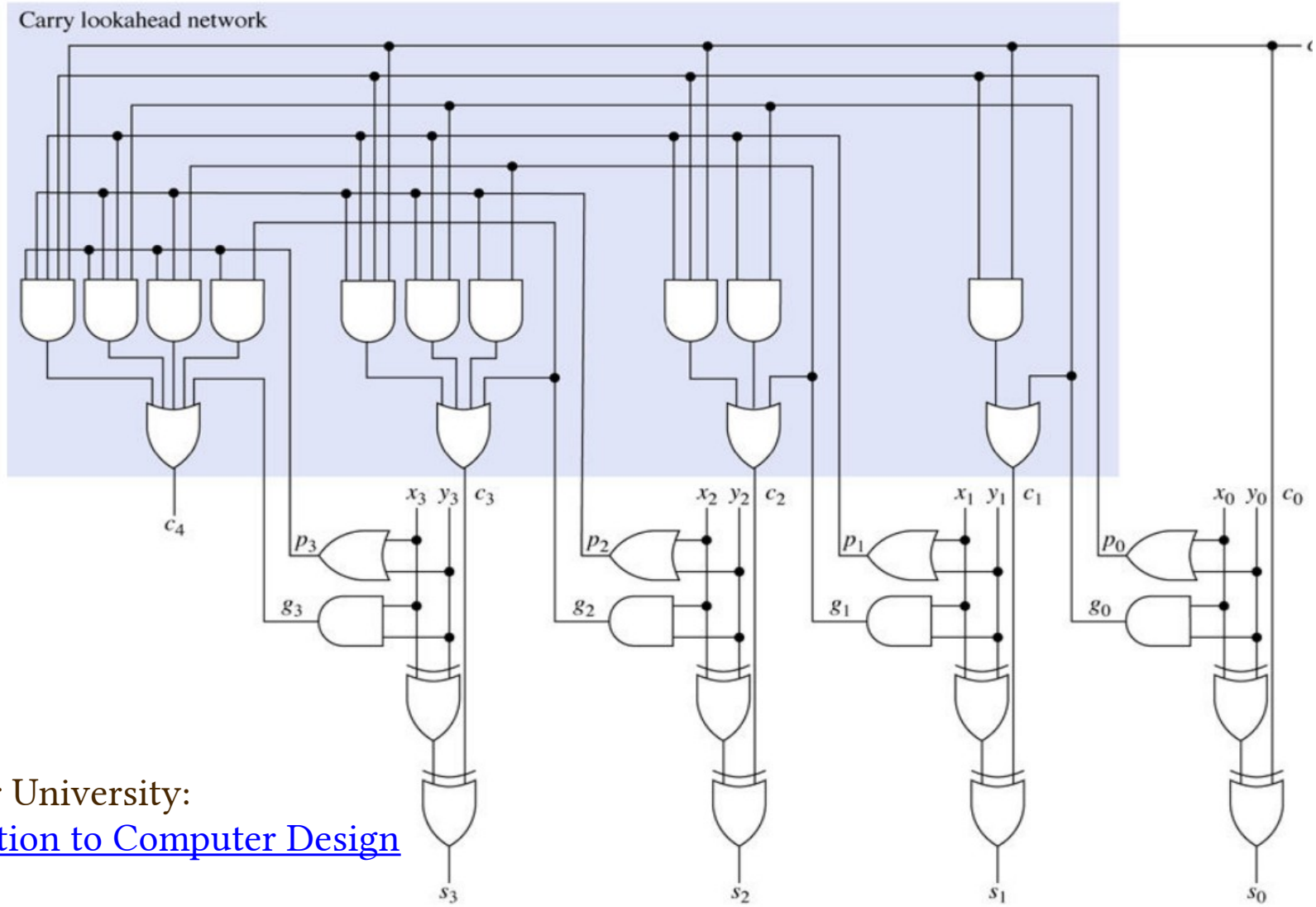
Gyors átvitelképzésű összeadó

- A **gyors átvitelképzésű összeadó** abban különbözik a *„terjedő átvitelű”* összeadótól, hogy az áthozat jel és az átvitelképzés nem lokális, hanem globális



Gyors átvitelképzésű összeadó

- A négybites **gyors átvitelképzésű összeadó** egy lehetséges megvalósítása az alábbi ábrán látható

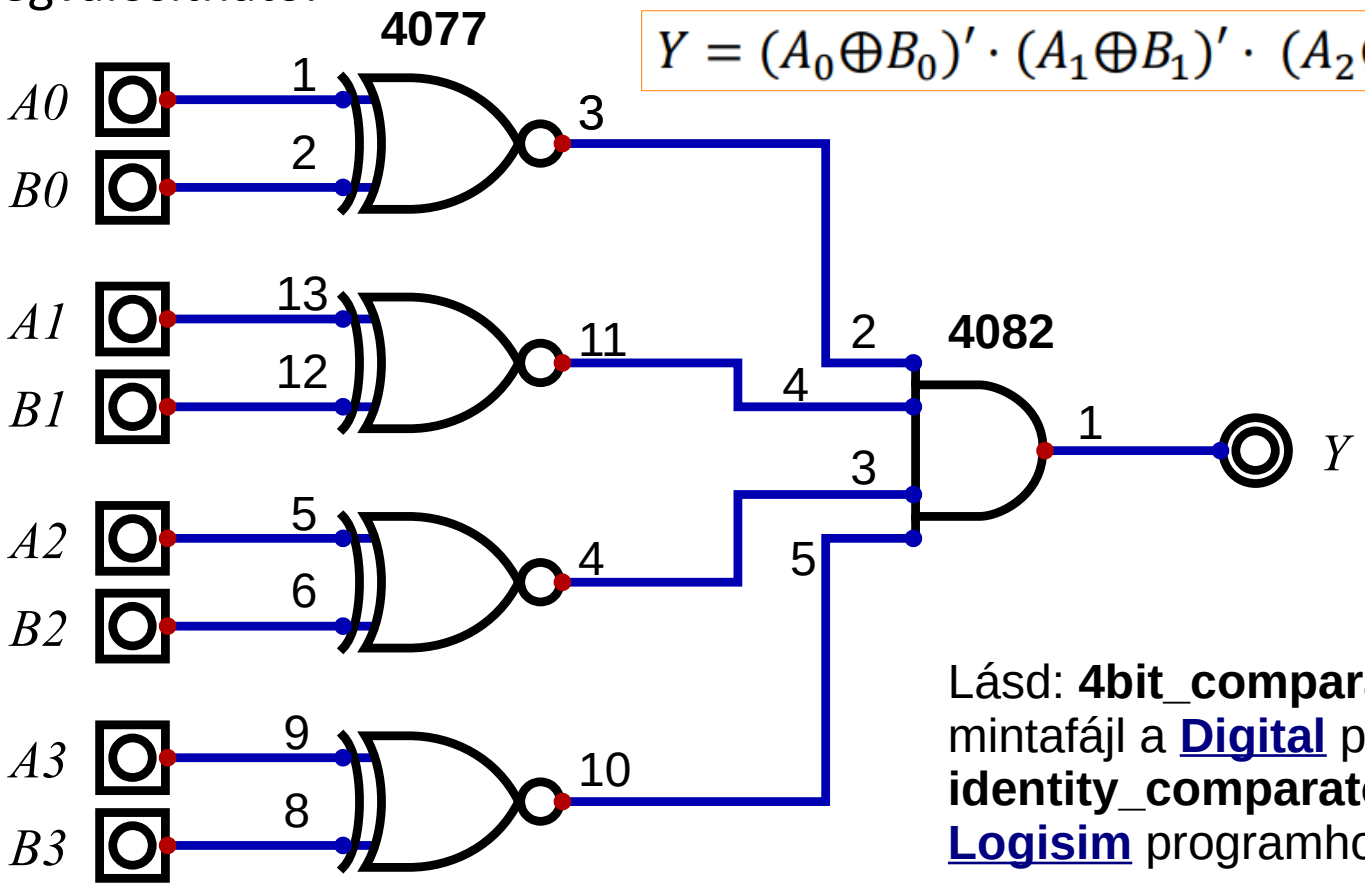


Forrás: Simon Frazer University:
[CMPT-150: Introduction to Computer Design Lecture 10]

Egyenlőség komparátor

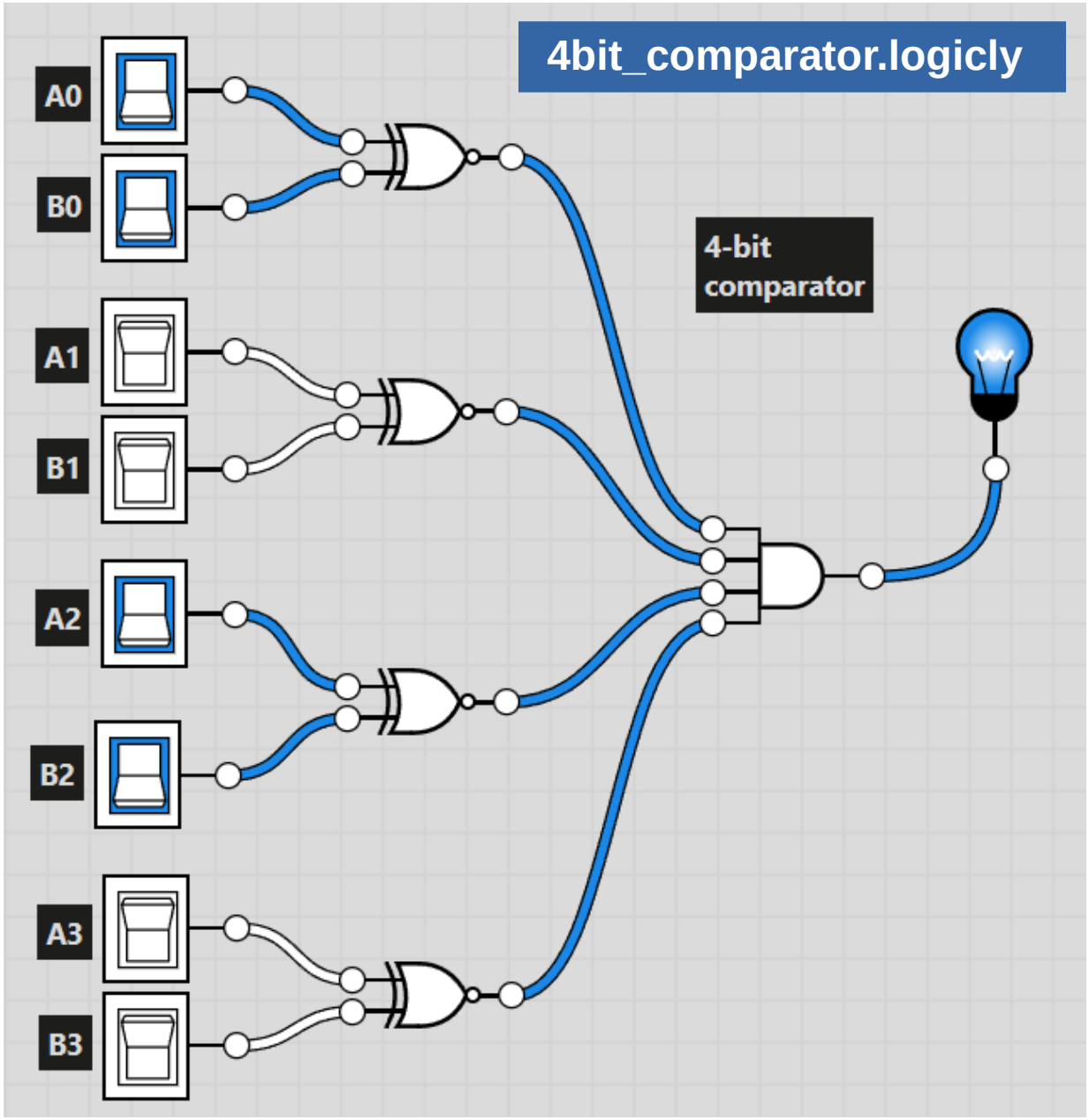
A komparátor áramkör funkciója a bemenetére adott számok összehasonlítása. Ha csak egyenlőséget jelez, akkor **egyenlőség komparátornak** *(identity comparator)* nevezzük. Ha az áramkör azt is jelzi, hogy az A szám nagyobb, vagy kisebb, mint a B szám, akkor **magnitúdó komparátornak** *(magnitude comparator)* hívjuk.
Az egyenlőség vizsgálata az egyszerűbb feladat, **XNOR** (ekvivalencia) kapukkal könnyen megvalósítható.

$$Y = (A_0 \oplus B_0)' \cdot (A_1 \oplus B_1)' \cdot (A_2 \oplus B_2)' \cdot (A_3 \oplus B_3)'$$



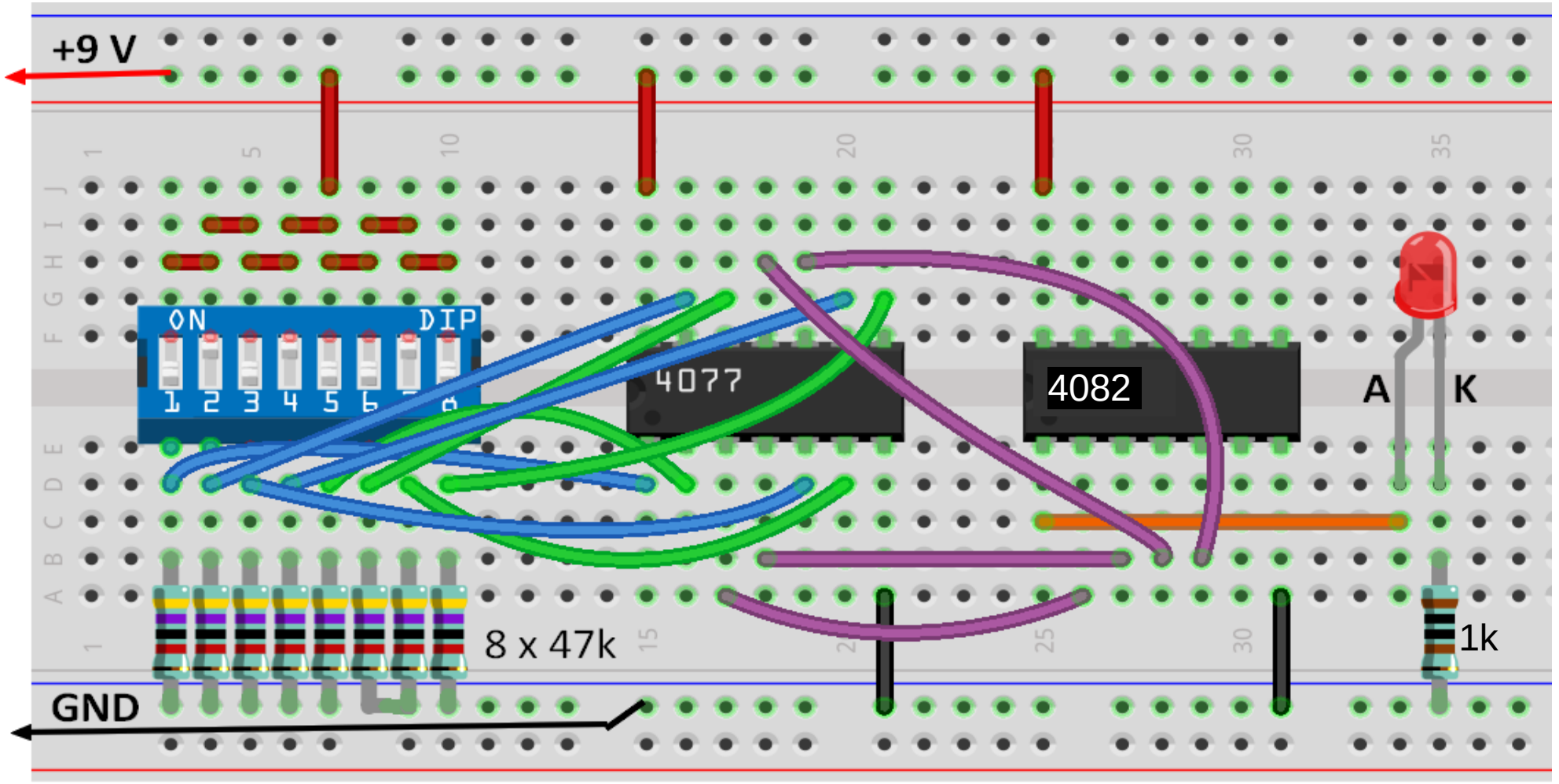
Lásd: **4bit_comparator.dig** mintafájl a **Digital** programhoz, ill. **identity_comparator.circ** a **Logisim** programhoz

Egyenlőség komparátor



Egyenlőség komparátor

- A nyolcas kapcsolósor első fele az A_i, a másik fele pedig a B_i bemeneteket állítja be
- A LED akkor világít, ha az Y kimenet ’1’ állapotban van (egyezés).



Logikai függvények kanonikus alakjai

- Korábban azt mondtuk, hogy az igazságtáblázat megadása egyértelmű, de egy igazságtáblázatnak többféle függvény, vagy áramkör is megfeleltethető
- Van azonban a logikai függvényeknek olyan felírási módja, ami egyszerű szabályokkal megadható és (az adott szabályokon belül) egyértelmű alakra vezet ez az ún. kanonikus alak
- Kanonikus alakból két típus van:
  - **Diszjunktív normálalak** (kanonikus szorzatok összege)
  - **Konjunktív normálalak** (kanonikus összegek szorzata)
- A **diszjunktív normálalakhoz** az igazságtáblázatból ki kell választani azokat a sorokat, ahol a kimenet ’1’, ezek lesznek a normálalakban szereplő ***mintermek.*** Ezekben ha a változónál egyes szerepel, akkor azt ponált alakban, ha nulla akkor pedig negált alakban kell leírni és összeszorozni. A kiválasztott ***mintermeket*** végül összeadjuk – ilyen felírást már csináltunk az előző előadásban
- A **konjunktív normálalakhoz** az igazságtáblázatból azokat a sorokat választjuk ki, ahol a kimenet ’0’, s ha a változónál ’1’ szerepel, akkor *negált* alakban, ha pedig ’0’ akkor *ponált* alakban kell összeadni. Ezeket az ún. ***maxtermeket*** végül összeszorozzuk

minterm és maxterm

- Három változó esetén a **mintermek** és **maxtermek** az alábbiak
  (M. Morris Mano and Michael D. Ciletti: Digital Design)

			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

- A **diszjunktív normálalakot** úgy kapjuk, hogy összegezzük azokat a **mintermeket,** amelyek 1-et eredményeznek az igazságtáblázatban
- A **konjunktív normálalakot** úgy kapjuk, hogy összeszorozzuk azokat a **maxtermeket,** amelyek nullát eredményeznek az igazságtáblázatban

Miért minterm és maxterm?

- A logikai függvények diszjunktív normálalakjában szereplő szorzatokat **mintermeknek** nevezzük, mert **ÉS** kapcsolatban vannak, s az **ÉS** kapcsolat igazságtáblázata a ***min(a,b)*** függvénnyel írható le

a	*b*	*a ÉS b*	*min(a,b)*
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

Jelölés: pl. $\overline{A}\cdot B\cdot\overline{C}\cdot D \Rightarrow m_5^4$ (0 1 0 1)

← Változók száma

← Minterm sorszáma

- A logikai függvények konjunktív normálalakjában szereplő összegeket **maxtermeknek** nevezzük, mert **VAGY** kapcsolatban vannak, s a **VAGY** kapcsolat igazságtáblázata a ***max(a,b)*** függvénnyel írható le

a	*b*	*a OR b*	*max(a,b)*
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1

- **Maxtermek** esetén a sorszámot a **mintermektől** eltérő módon képezzük: a ponált alakok számítanak nullának, a negált alakok pedig egynek

Jelölés: pl. $\overline{A}+B+\overline{C}+D \Rightarrow M_{10}^4$ (1 0 1 0)

Nézzünk egy példát!

Az alábbi igazságtáblázatból így írhatjuk fel

A	B	C	Q
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

- **Diszjunktív alakban:**

$$Q = m_2^3 + m_4^3 + m_6^3 = \overline{A}\,B\,\overline{C} + A\,\overline{B}\,\overline{C} + A\,B\,\overline{C} = \sum^{3}(2,4,6)$$

Egyszerűsítve: $Q = A\,\overline{C} + B\,\overline{C}$

- **Konjunktív alakban:** $Q = M_0^3 \cdot M_1^3 \cdot M_3^3 \cdot M_5^3 \cdot M_7^3 =$

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

$$= \prod^{3}(0,1,3,5,7)$$

Egyszerűsítve: $Q = A\,\overline{C} + B\,\overline{C}$

(lásd: Boolean Algebra Simplifier)

- Megjegyzés: A konjunktív normálalakot akkor célszerű használni, ha az igazságtáblázatban az eredmény oszlopban kevés a ’0’

Összefüggés a kanonikus alakok között

- **A diszjunktív normálalak** így nézett ki:

$$Q = m_2^3 + m_4^3 + m_6^3 = \overline{A}\,B\,\overline{C} + A\,\overline{B}\,\overline{C} + A\,B\,\overline{C}$$

- Képezzük most a Q függvény negáltját (Q') és írjuk fel ennek a diszjunktív normálalakját! *(Ehhez azokat a mintermeket kell összegezni, amelyeknél Q = '0' állt)*

$$Q' = m_0^3 + m_1^3 + m_3^3 + m_5^3 + m_7^3$$

$$Q' = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,C + \overline{A}\,B\,C + A\,\overline{B}\,C + A\,B\,C$$

A	B	C	Q	Q'
0	0	0	0	1
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1

- Ha Q'-at ismét negáljuk és alkalmazzuk a De Morgan azonosságokat, akkor a fenti egyenletből ezt kapjuk:

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

- Ez pedig nem más, mint a **konjunktív normálalak:**

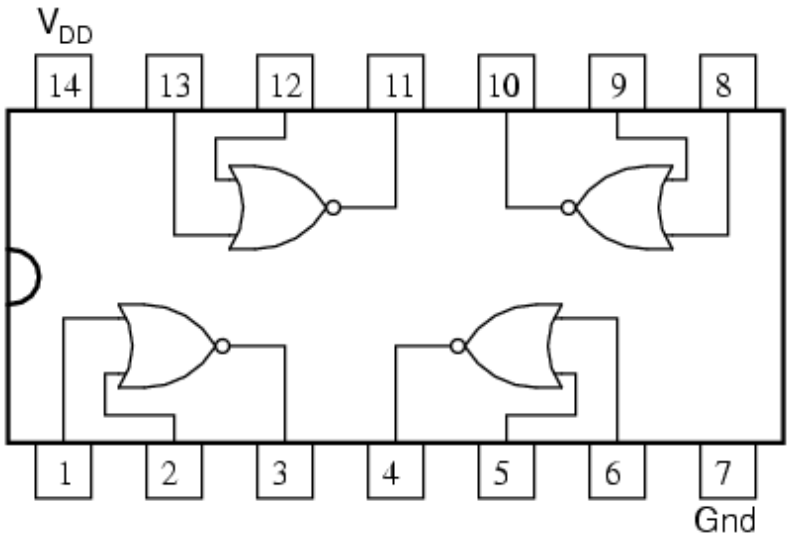
$$Q = M_0^3 \cdot M_1^3 \cdot M_3^3 \cdot M_5^3 \cdot M_7^3$$

A 4000-es sorozat tipikus tagjai

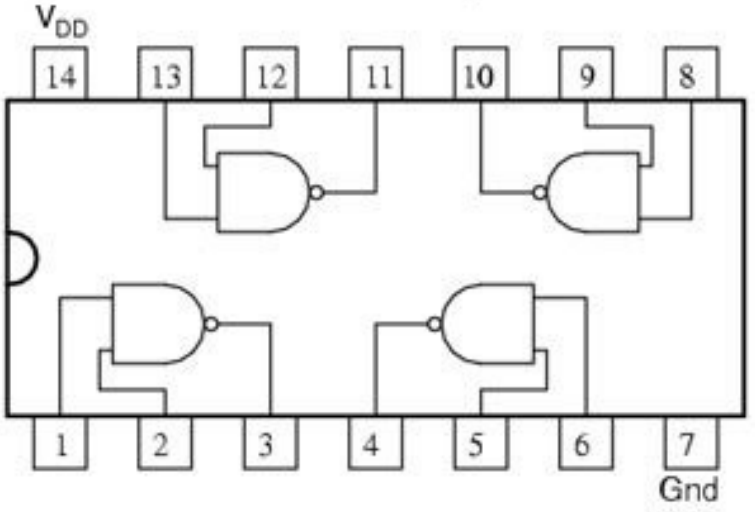
4001 CMOS Quad 2-Input NOR Gate
4011 CMOS Quad 2-Input NAND Gate
4013 CMOS Dual D-Type Flip Flop
4017 CMOS Decade Counter with 10 Decoded Outputs
4021 CMOS 8-Stage Static Shift Register
4022 CMOS Octal Counter with 8 Decoded Outputs
4023 CMOS Triple 3-Input NAND Gate
4025 CMOS Triple 3-Input NOR Gate
4026 CMOS Decade Counter/Divider with Decoded 7-Segment Display Outputs and Display Enable
4027 CMOS Dual J-K Master-Slave Flip-Flop
4028 CMOS BCD-to-Decimal or Binary-to-Octal Decoders/Drivers
4043 CMOS Quad NOR R/S Latch with 3-State Outputs
4046 CMOS Micropower Phase-Locked Loop
4049 CMOS Hex Inverting Buffer/Converter
4050 CMOS Hex Non-Inverting Buffer/Converter
4051 CMOS Single 8-Channel Analog Multiplexer/Demultiplexer with Logic-Level Conversion
4052 CMOS Differential 4-Channel Analog Multiplexer/Demultiplexer with Logic-Level Conversion
4053 CMOS Triple 2-Channel Analog Multiplexer/Demultiplexer with Logic-Level Conversion
4060 CMOS 14-Stage Ripple-Carry Binary Counter/Divider and Oscillator
4066 CMOS Quad Bilateral Switch
4069 CMOS Hex Inverter
4070 CMOS Quad Exclusive-OR Gate
4071 CMOS Quad 2-Input OR Gate
4072 CMOS Dual 4-Input OR Gate
4073 CMOS Triple 3-Input AND Gate
4075 CMOS Triple 3-Input OR Gate
4081 CMOS Quad 2-Input AND Gate
4082 CMOS Dual 4-Input AND Gate
4093 CMOS Quad 2-Input NAND Schmitt Triggers
4094 CMOS 8-Stage Shift-and-Store Bus Register

A 4000-es sorozat tipikus tagjai

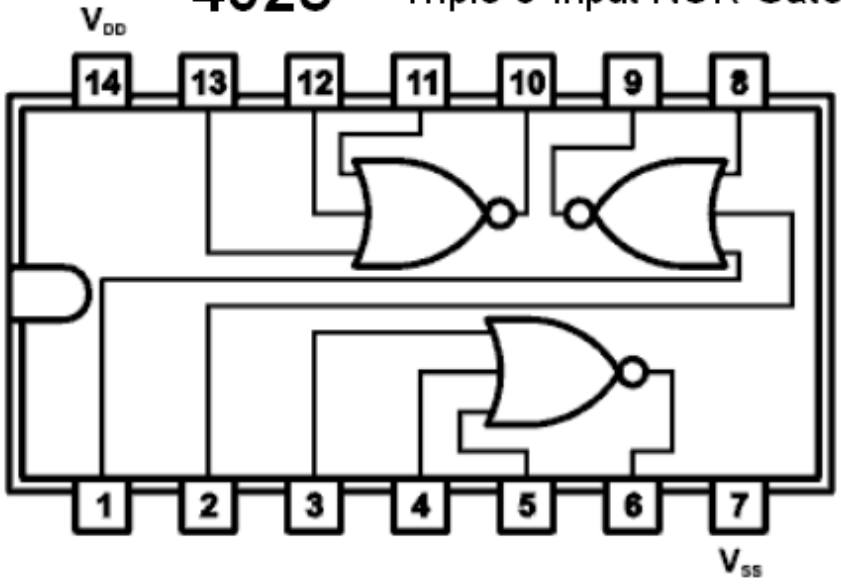
4001 Quad 2-Input NOR Gate



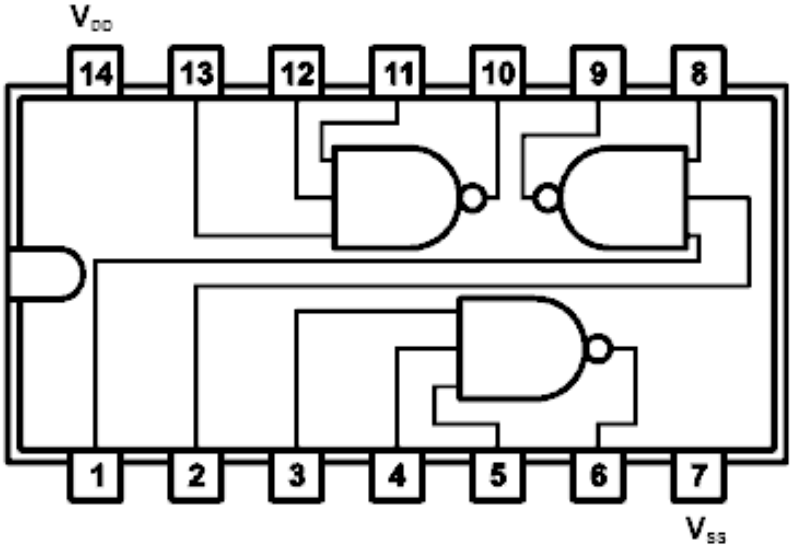
4011 Quad 2-input NAND



4025 Triple 3-Input NOR Gate

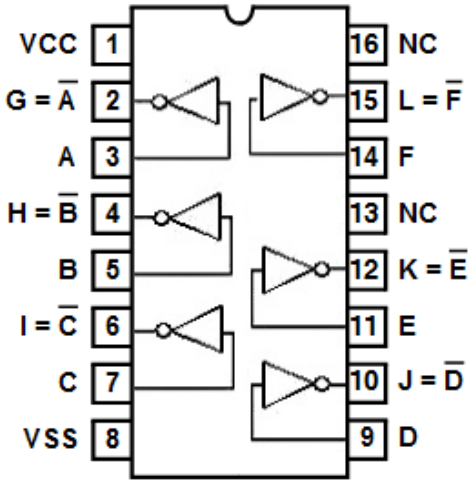


4023 Triple 3-Input NAND Gate

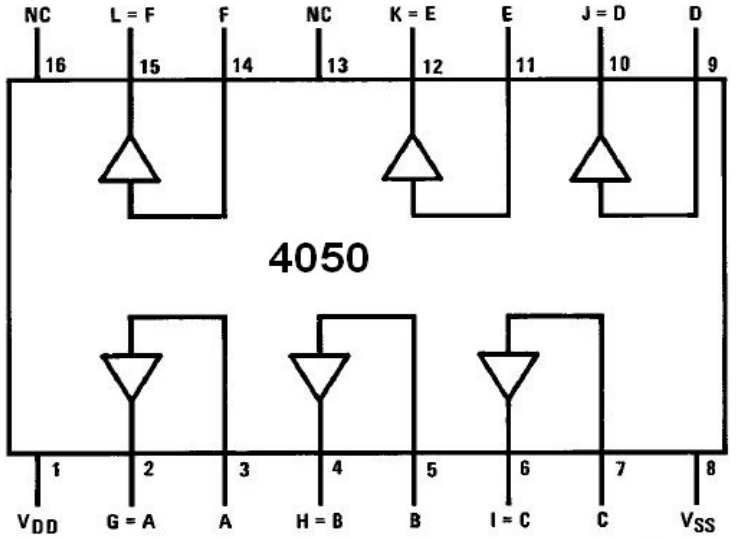


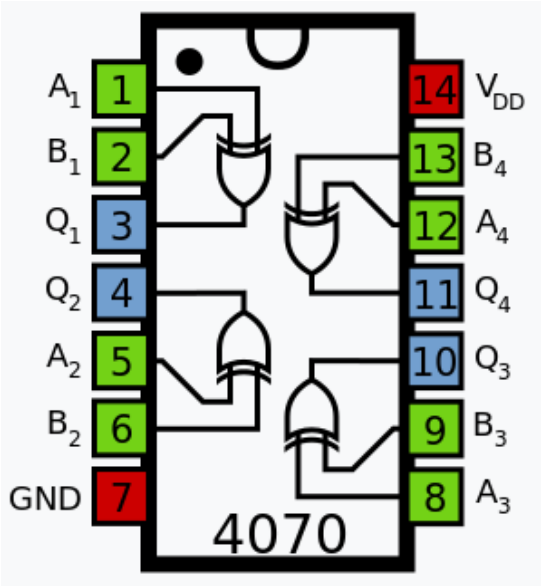
A 4000-es sorozat tipikus tagjai

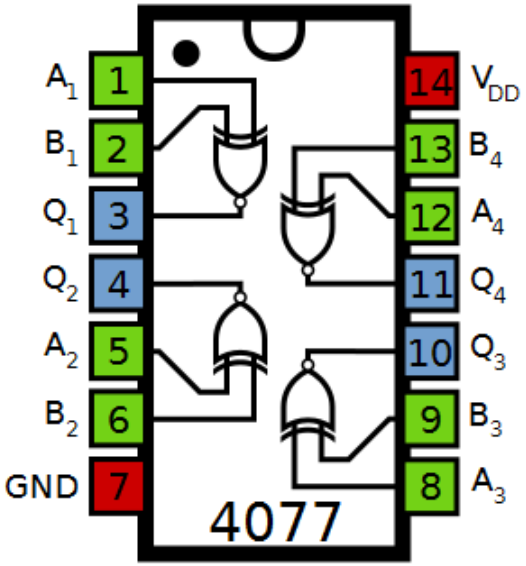
4049
Hex Inverting Buffer/Converter

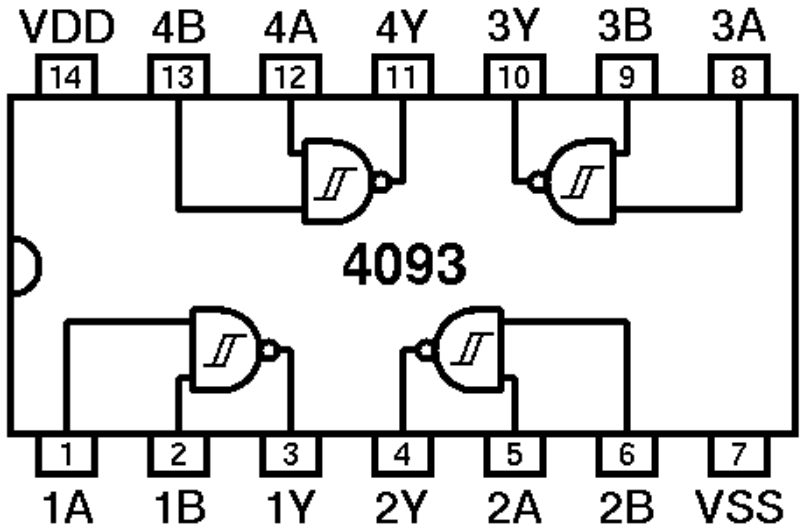


4050 Hex Non-Inverting Buffer/Converter









A 4000-es sorozat tipikus tagjai