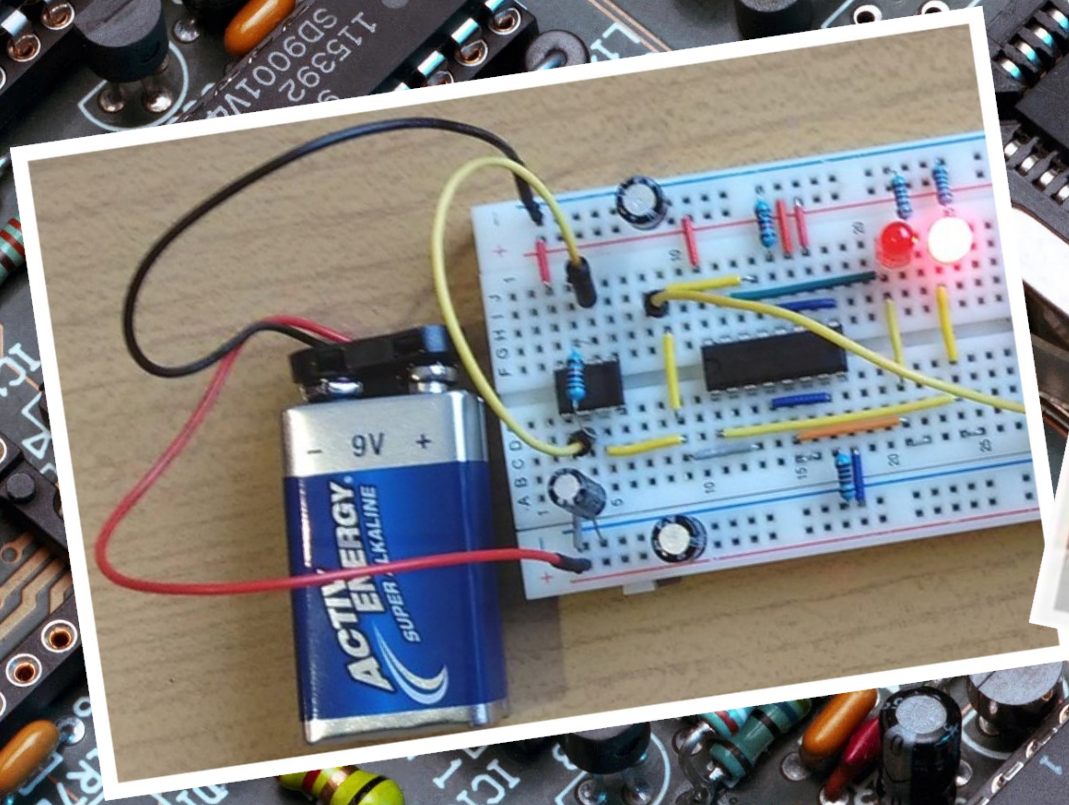
A digitális elektronika alapjai



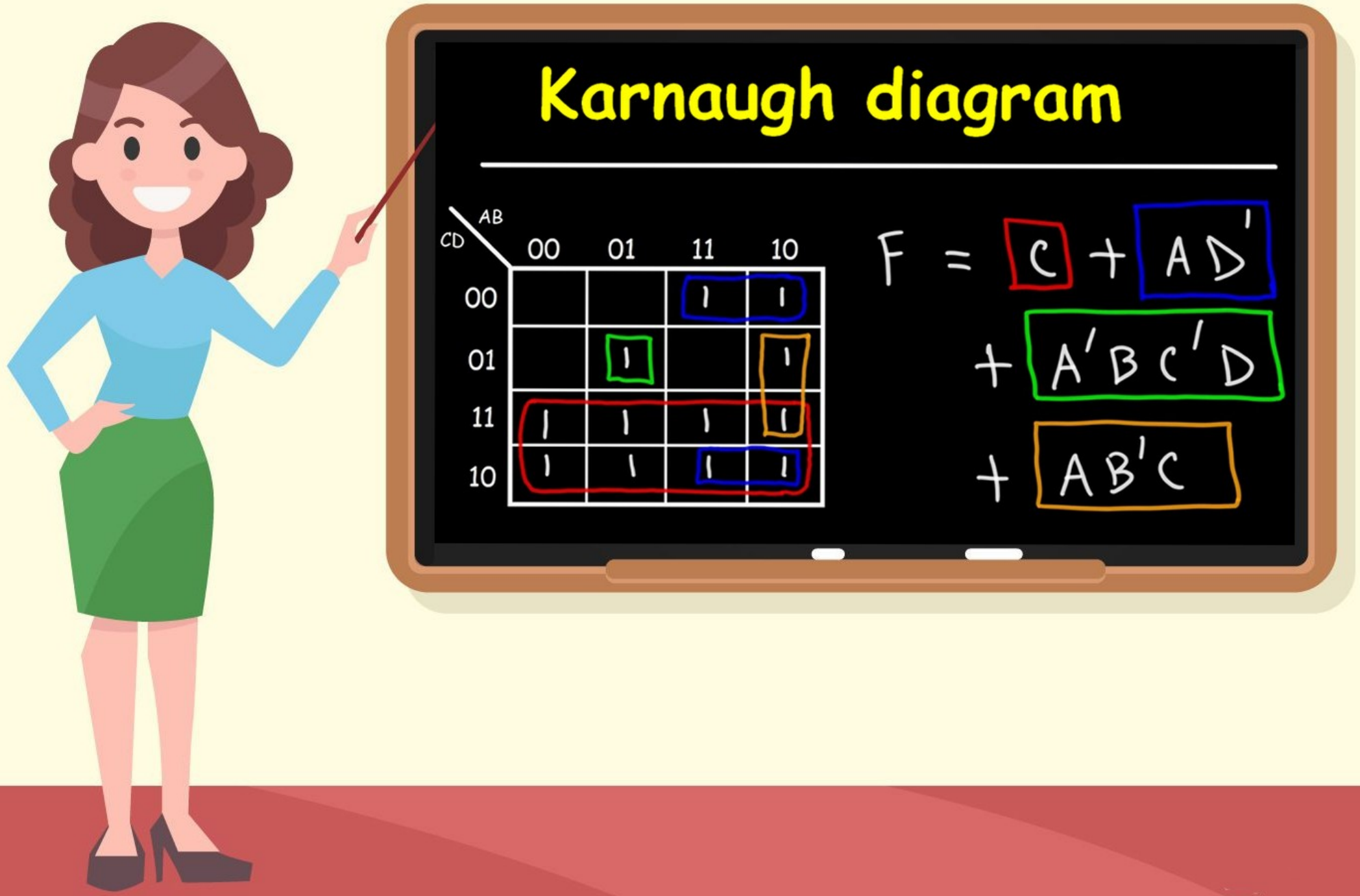
7. Kombinációs logikai hálózatok – 4. rész

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)


Karnaugh diagram
AB
CD
00
01
11
10
00
01
11
10
F = C + AD'
+ A'BC'D
+ AB'C

Veitch-, ill. Karnaugh-diagram

- Ha az igazságtáblázatot úgy írjuk át, hogy egy tengelyre egy, vagy két változót csoportosítunk, akkor könnyen felismerhetjük az egyszerűsítési lehetőségeket
- Vegyük pl. a $Q = \bar{A}B\bar{C} + A\bar{B}\bar{C} + AB\bar{C}$ függvényt:

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

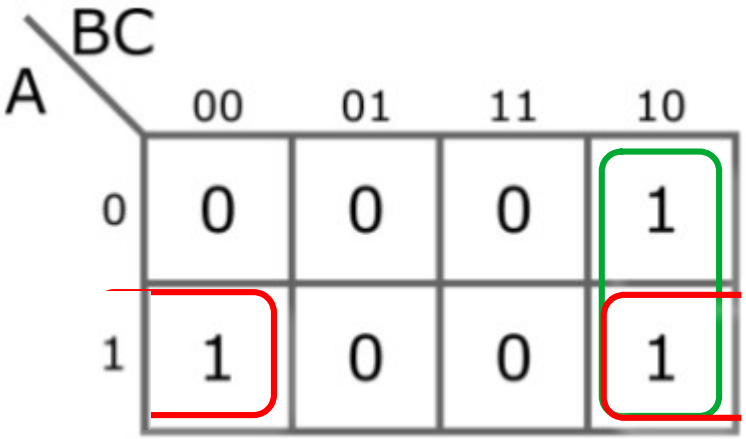
Karnaugh-diagram

A \ BC	00	01	11	10
0	0 m0	0 m1	0 m3	1 m2
1	1 m4	0 m5	0 m7	1 m6

Veitch-diagram

A \ B	0	0	0	1
	m0	m1	m3	m2
1	1	0	0	1
	m4	m5	m7	m6

C



- A keretezésnél 1, 2, 4, 8 mintermet lefedő, „nullamentes” téglalapokat keresünk

$$m_2 + m_6 = \bar{A}B\bar{C} + AB\bar{C} = (\bar{A} + A)B\bar{C} = B\bar{C}$$

$$m_4 + m_6 = A\bar{B}\bar{C} + AB\bar{C} = A(\bar{B} + B)\bar{C} = A\bar{C}$$

$$Q = A \cdot \bar{C} + B \cdot \bar{C}$$

Logisim és utódai...

- A Logisim programot dr. Carl Burch írta, a fejlesztése 2014-ben leállt,
de a nyíltforrású program továbbfejlesztésébe többen is belefogtak:
- Logisim-evolution ami kombinációs logikai feladatoknál képes „sűríteni” az igazságtábázatot és K-map ill. Veitch diagram alapján felismeri az egyszerűsítési lehetőségeket. Szimulációban logikai IC-ket (74xxx) is ismer, illetve PLA és FPGA orientált fejlesztésekre is használható
- Digital hasonló az előzőhöz, de az igazságtáblázat „sűrítésére” nem képes. FPGA fejlesztésekhez az áramkör VHDL, ill. Verilog formátumban is exportálható

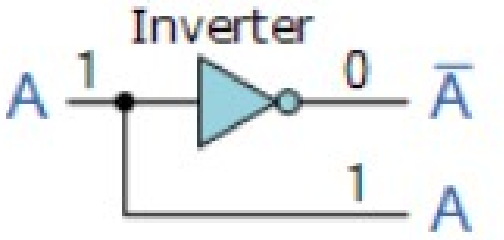
D3	D2	D1	D0	Y	X	V
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	1	0	1
0	0	1	1	1	0	1
0	1	0	0	0	1	1
0	1	0	1	0	1	1
0	1	1	0	0	1	1
0	1	1	1	0	1	1
1	0	0	0	1	1	1
1	0	0	1	1	1	1
1	0	1	0	1	1	1
1	0	1	1	1	1	1
1	1	0	0	1	1	1
1	1	0	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	1	1	1

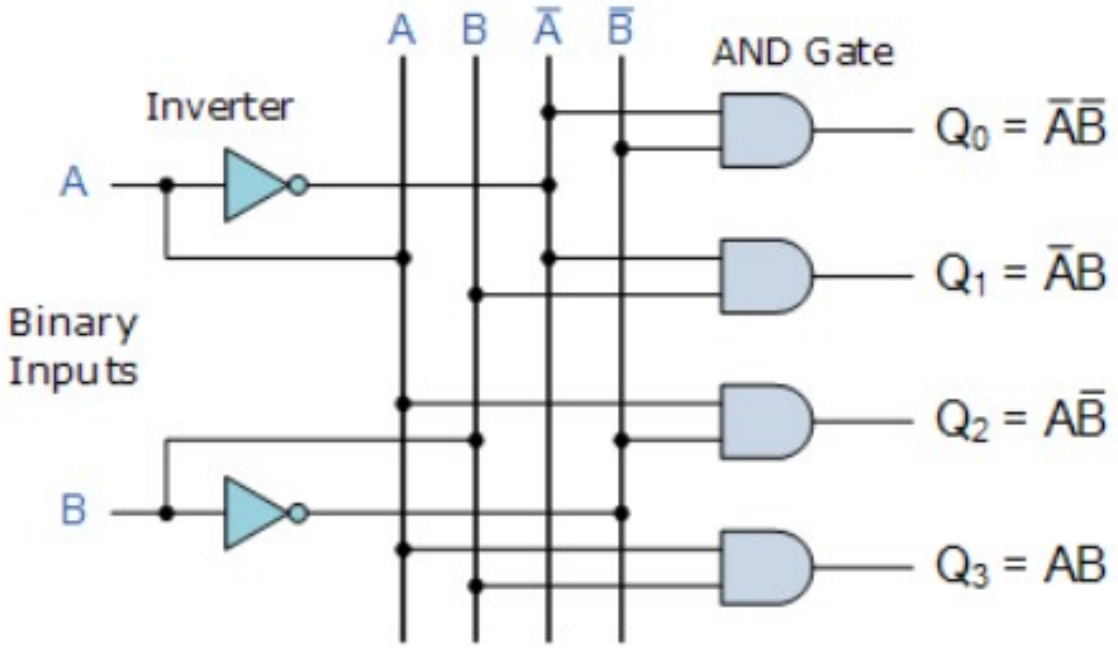
D3	D2	D1	D0	Y	X	V
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	-	1	0	1
0	1	-	-	0	1	1
1	-	-	-	1	1	1


Kódolók és dekódolók

Emlékeztető: Bináris dekódolók

- A **bináris dekódoló** olyan kombinációs logikai áramkör, ami az n db bemenetére érkező bináris információt $m \leq 2^n$ egyedi kimenő jellé alakítja. Tipikus alkalmazásai: cím- és utasításdekódolók, kijelző vezérlő dekódoló
- Az inverter (**NEM** kapu) például 1 bites bináris dekódolónak tekinthető.
- Általánosságban az n bitről m kimenetre dekódoló áramkör m darab **mintermet** valósít meg
- **Példa: 2 bitről 4 vonalra dekódoló áramkör**



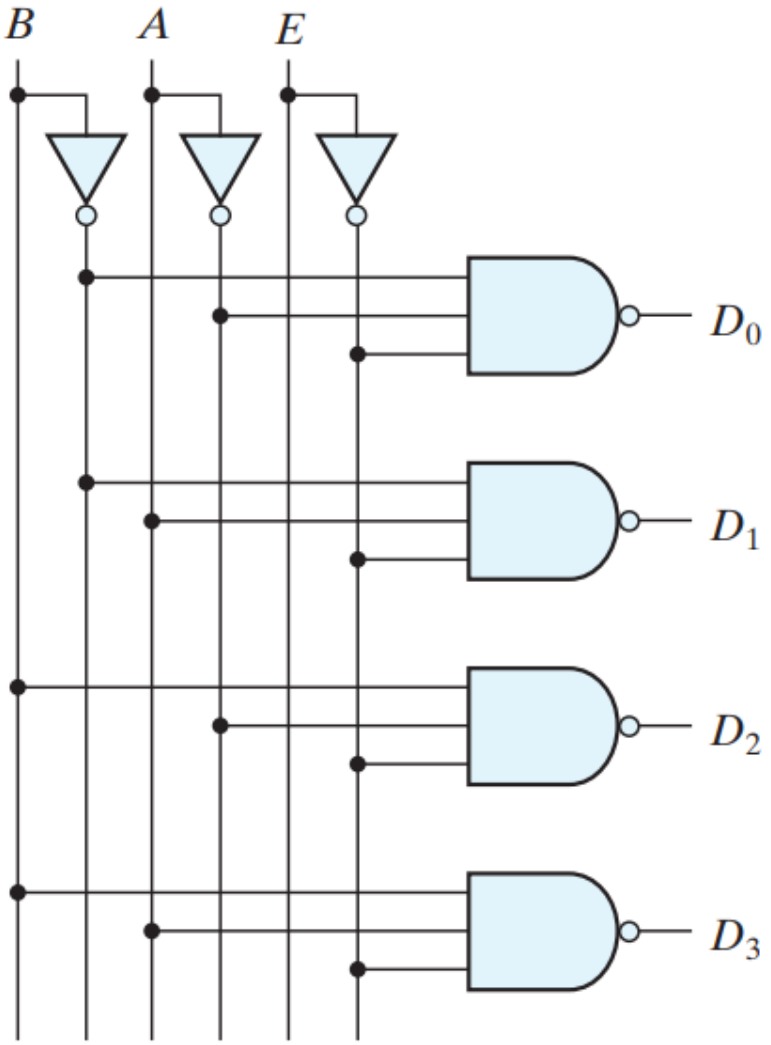


A	B	Q0	Q1	Q2	Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Negált kimenetű bináris dekódolók

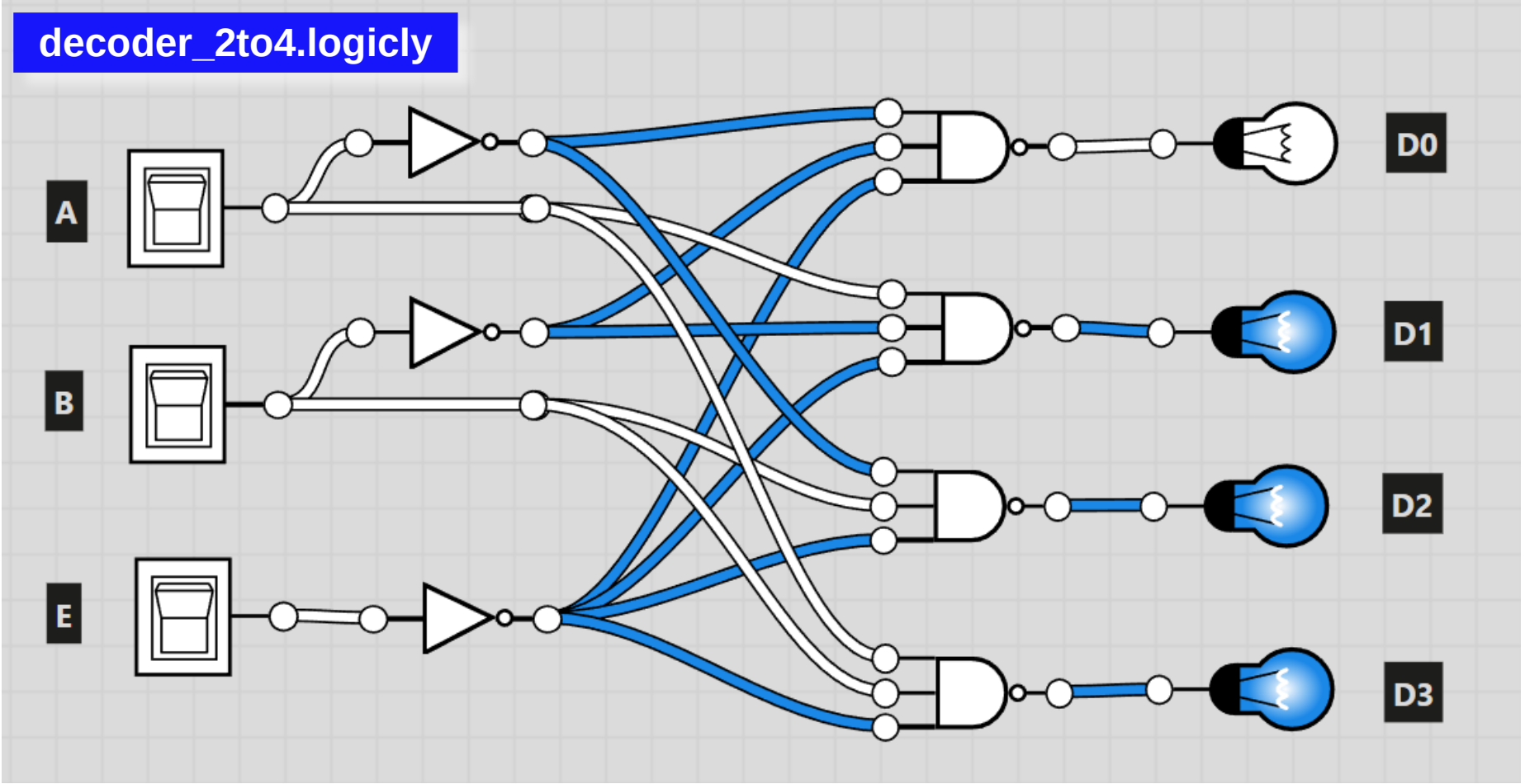
- Némelyik dekódolót **NAND** (Nem-ÉS) áramkörökkel valósítják meg, ilyenkor a kimenetek aktív állapota alacsony szint (0). Egyidejűleg legfeljebb egy kimenet lehet aktív
- Ha a kapubemenetek számát bővítjük, akkor egy engedélyező jelet is bevezethetünk. Az alábbi kapcsolásban az ***E*** engedélyező jel alacsony szintű állapota aktiválja a dekódolót

E	B	A	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0



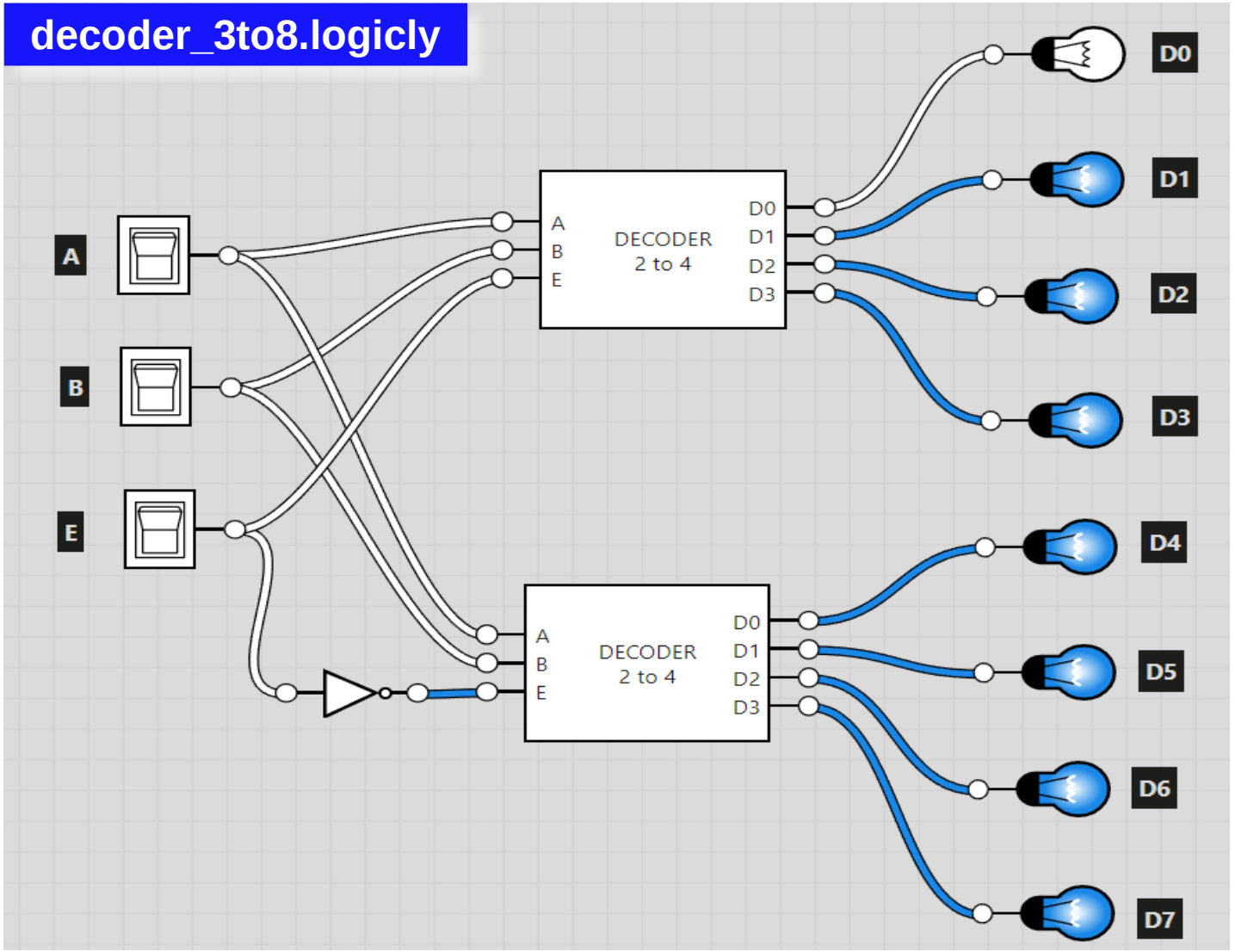
NAND dekódoló két bitről 4 vonalra

- Az előző oldalon bemutatott kapcsolást a **Logic.ly** programban is kipróbáltuk, s a kapcsolást alkatrészként is elmentettük



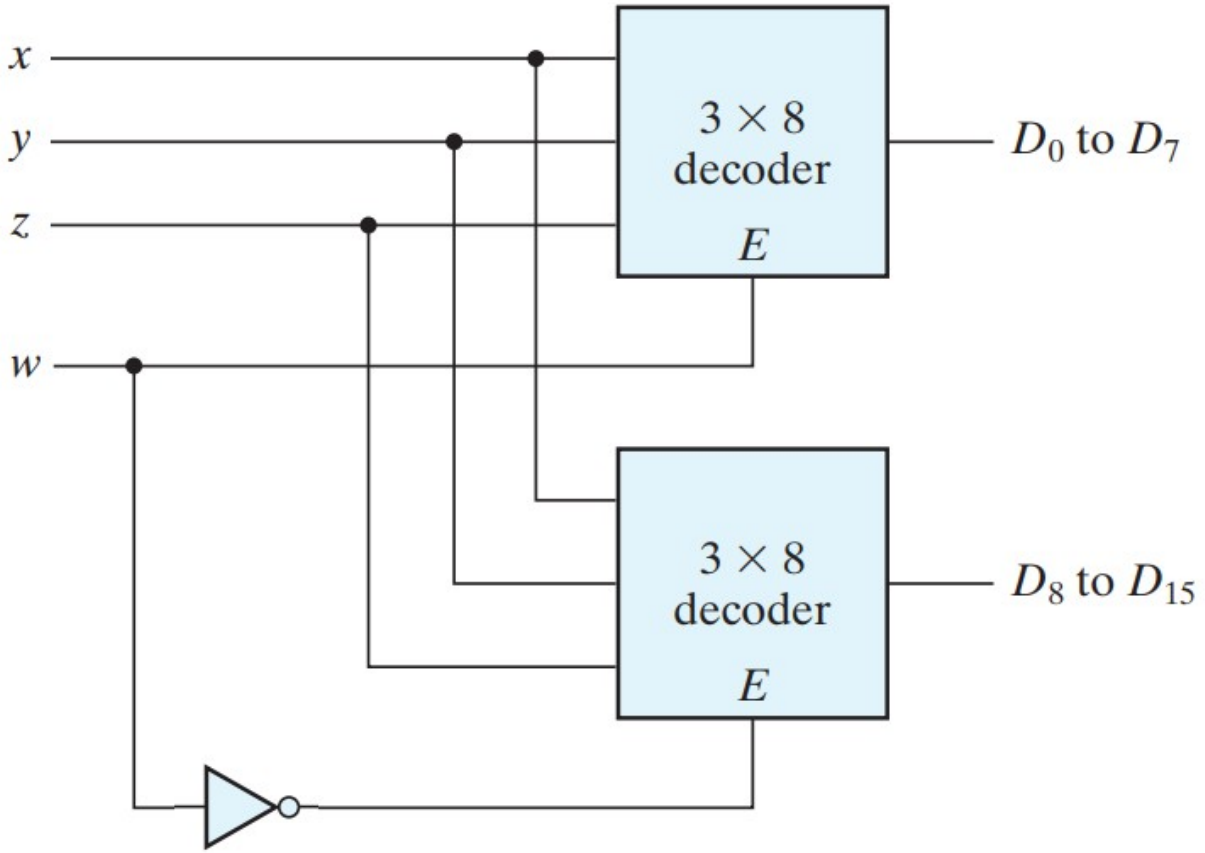
NAND dekódoló három bitről 8 vonalra

- Az ***E*** bemenő jelet adatnak tekintve könnyen kiterjeszthetjük a kapcsolást 3 bitből 8 vonalra dekódolóvá



NAND dekódoló 4 bitről 16 vonalra

- A három bitről 8 vonalra dekódolókból (ilyen pl. a **74HC138** IC is), könnyen készíthetünk 4 bitről 16 vonalra dekódolót
- Megjegyzés: ha a dekódolók nem **NAND,** hanem **AND** típusúak, akkor az inverter a másik ágba kötendő!



Kódoló áramkörök (encoders)

- A kódoló áramkör feladata a fordítottja a dekódolónak, például 2^n bemenő vonal állapotának megfelelően kiad egy *n* bites bináris kódot a kimenetén
- Egy példa a 8 bemenetet hárombites bináris számmá alakító kódoló, (az igazságtáblázata alább látható). A kimenetek az alábbi formulák szerint egy-egy négybemenetű VAGY áramkörrel egyszerűen megépíthetők

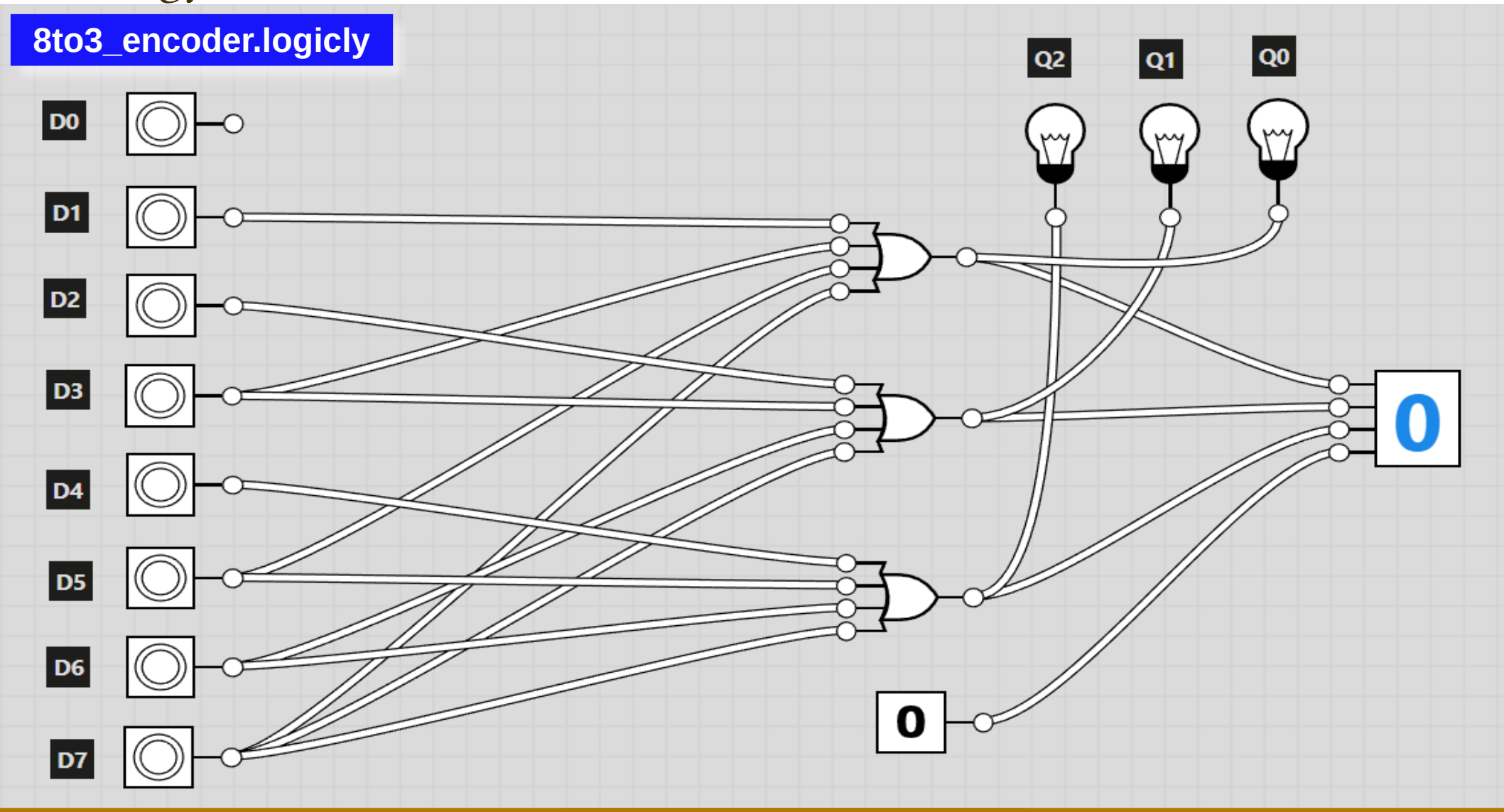
$$z = D_1 + D_3 + D_5 + D_7$$
$$y = D_2 + D_3 + D_6 + D_7$$
$$x = D_4 + D_5 + D_6 + D_7$$

Bemenetek								Kimenetek		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

- **Hátrányok:**
  - Egyszerre csak egy bemenet lehet aktív, különben hamis kódot kapunk
  - Ha egyik bemenet sem aktív, ugyanúgy 000 kódot kapunk, mint amikor a **D0** bemenet aktív *(a későbbiekben majd mutatunk megoldást ezekre)*

8 vonalról 3 bitre kódoló áramkör

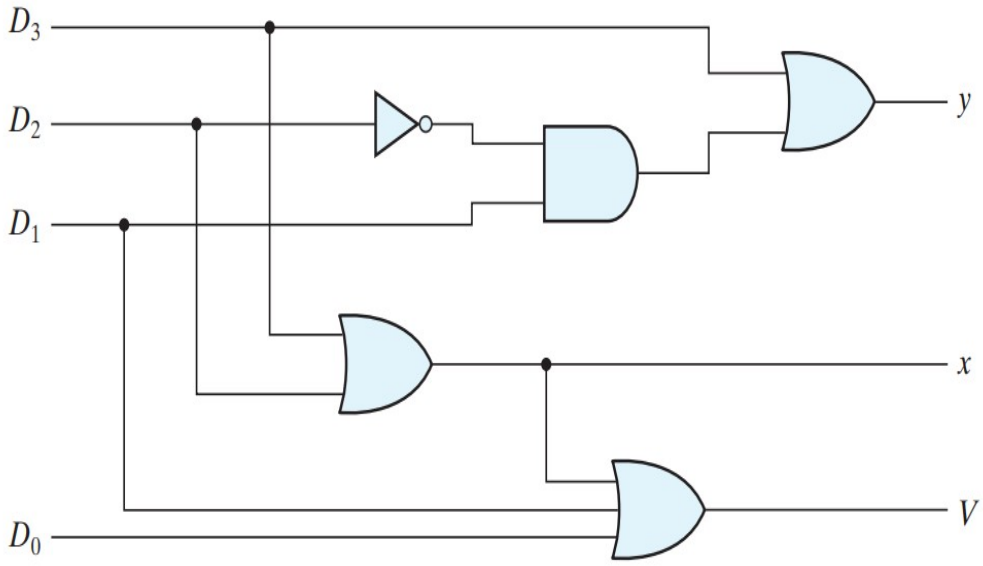
- A kapcsolást a **Logic.ly** programban is kipróbálhatjuk. Láthatjuk, hogy a nullával nem tudunk mit kezdeni



Elsőbbségi kódoló (priorty encoder)

- Az elsőbbségi kódoló több aktív bemenet esetén csak a legmagasabb prioritásút veszi figyelembe. A prioritás előre behuzalozott, a bemenetek sorszámától függ
- A tétlen állapot megkülönböztetésére (amikor egyik bemenet sem aktív) egy külön jelet is előállítunk, ami azt jelzi, hogy van-e aktív bemenet
- A mikrovezérlők, például, a beérkező megszakítási kérelmek priorizálására is használnak ilyen áramköröket. Az alábbi példában egy 4 bemenetű elsőbbségi kódolót mutatunk be.

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1



Az igazságtáblázat tömörebb felírása

- Lényeges egyszerűsítés az igazságtáblázat felírásánál, ha a számunkra érdektelen sorokat nem részletezzük, hanem összevonjuk
- A négybites elsőbbségi kódolónál így 1+3+7 =11 sorral kevesebbet kell leírni
- Pl. az XX10 sor négy mintermet jelent: 0010, 1010, 0110, 1110

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

D3	D2	D1	D0	X	Y	V
0	0	0	0	X	X	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

Négybites elsőbbségi kódoló

Az elsőbbségi kódoló igazság-táblázatából kiindulva a Veitch- vagy Karnaugh-diagram segítségével optimalizálhatjuk a kapcsolást

D3	D2	D1	D0	Y	X	V
0	0	0	0	x	x	0
0	0	0	1	0	0	1
0	0	1	0	1	0	1
0	0	1	1	1	0	1
0	1	0	0	0	1	1
0	1	0	1	0	1	1
0	1	1	0	0	1	1
0	1	1	1	0	1	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

$X = D2 \vee D3$

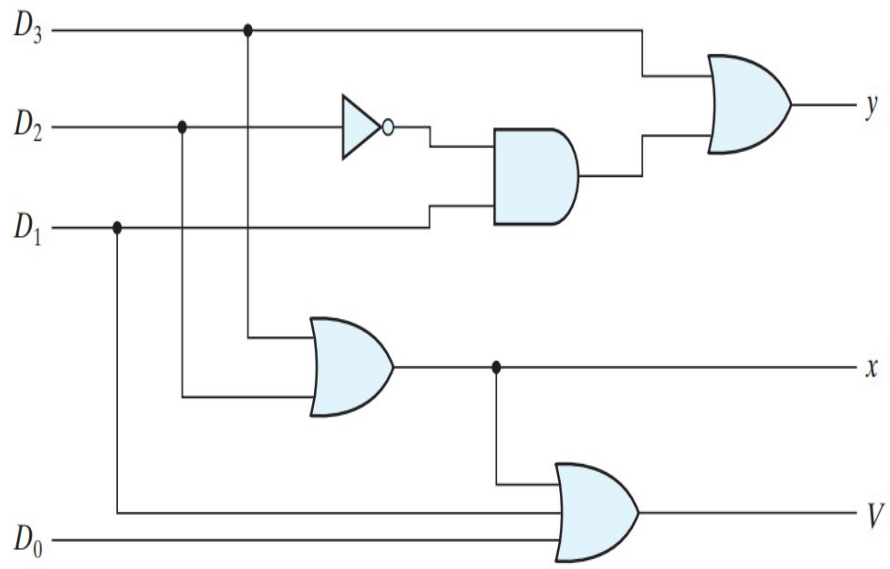
	$\overline{D1}$		D1		
$\overline{D3}$	X (0)	0 (1)	0 (3)	0 (2)	$\overline{D2}$
	1 (4)	1 (5)	1 (7)	1 (6)	D2
D3	1 (12)	1 (13)	1 (15)	1 (14)	
	1 (8)	1 (9)	1 (11)	1 (10)	$\overline{D2}$
	$\overline{D0}$		D0	$\overline{D0}$	

$Y = (D1 \wedge \overline{D2}) \vee D3$

	$\overline{D1}$		D1		
$\overline{D3}$	X (0)	0 (1)	1 (3)	1 (2)	$\overline{D2}$
	0 (4)	0 (5)	0 (7)	0 (6)	D2
D3	1 (12)	1 (13)	1 (15)	1 (14)	
	1 (8)	1 (9)	1 (11)	1 (10)	$\overline{D2}$
	$\overline{D0}$		D0	$\overline{D0}$	

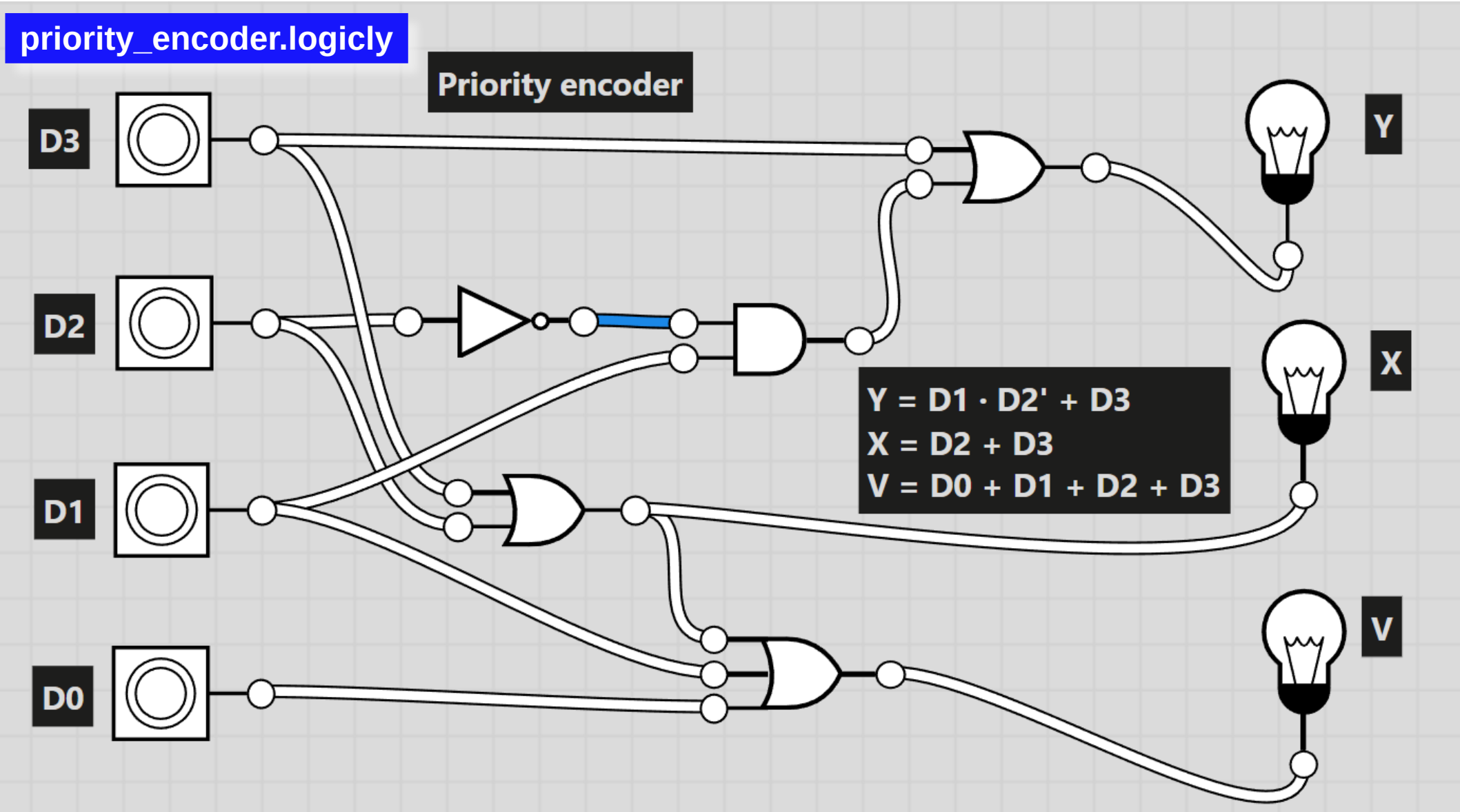
$V = D0 \vee D1 \vee D2 \vee D3$

	$\overline{D1}$		D1		
$\overline{D3}$	0 (0)	1 (1)	1 (3)	1 (2)	$\overline{D2}$
	1 (4)	1 (5)	1 (7)	1 (6)	D2
D3	1 (12)	1 (13)	1 (15)	1 (14)	
	1 (8)	1 (9)	1 (11)	1 (10)	$\overline{D2}$
	$\overline{D0}$		D0	$\overline{D0}$	



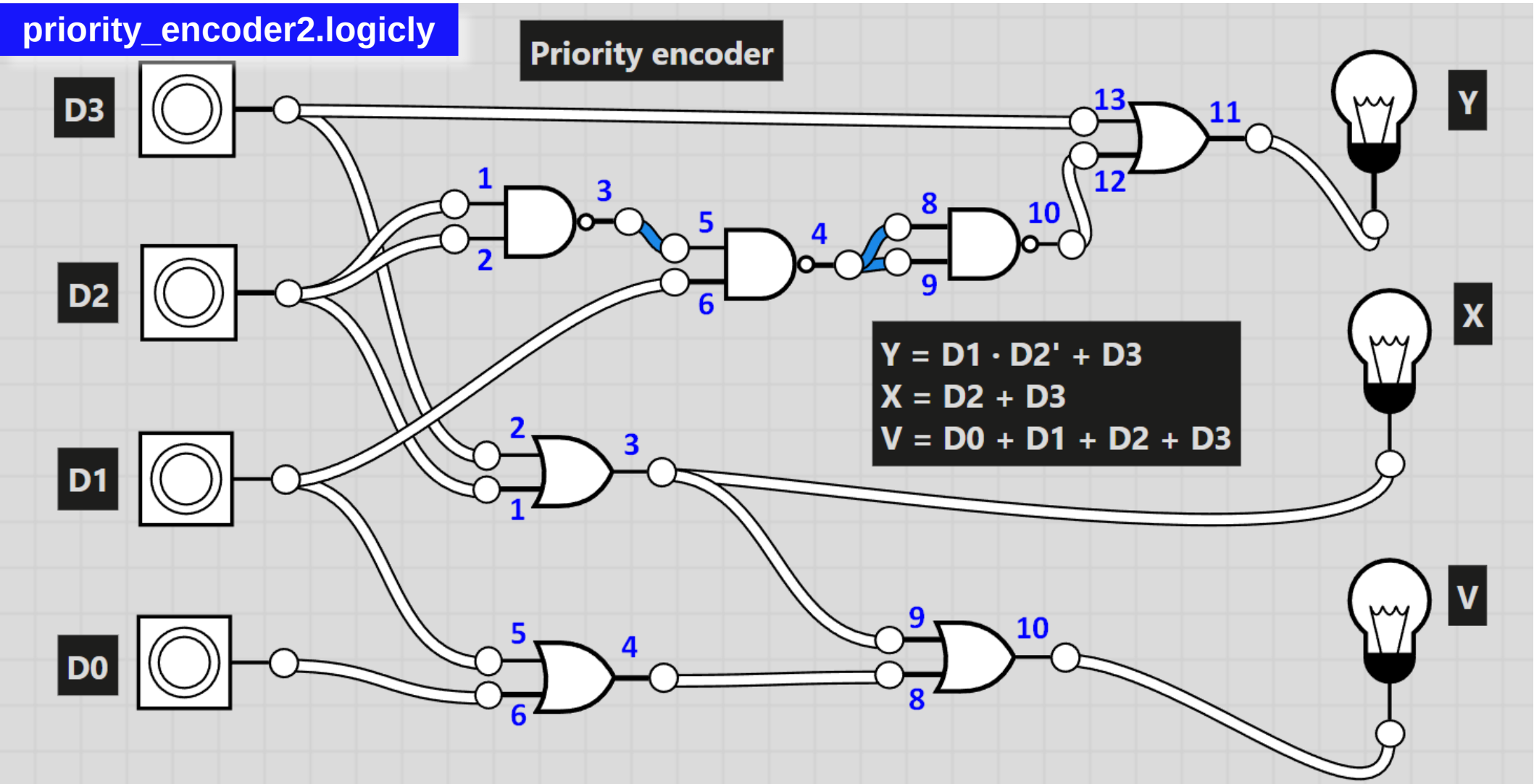
Négybites elsőbbségi kódoló

- A kapcsolást a **Logic.ly** programban is kipróbálhatjuk



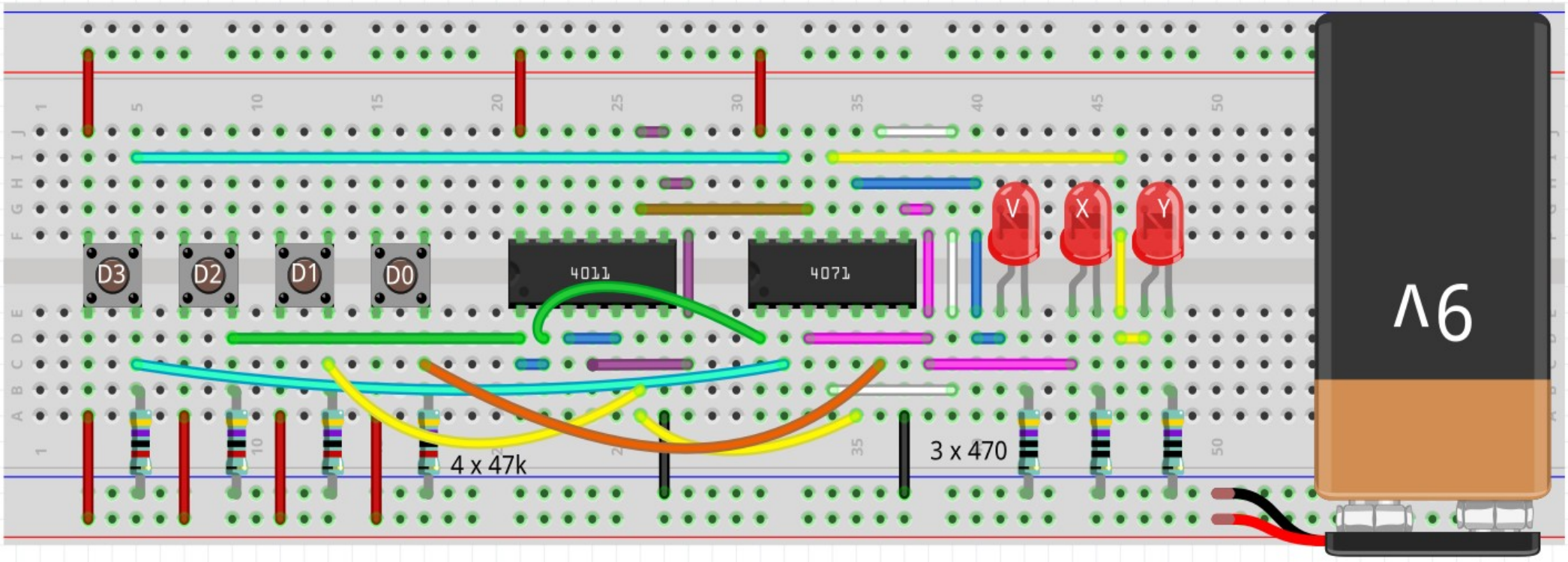
Az „optimalizált” kapcsolás

- Megépítés előtt érdemes az IC készlethez optimalizálni a kapcsolást, így két IC-ből meg tudjuk építeni (4011 NAND és 4071 OR)



A megépített elsőbbségi kódoló

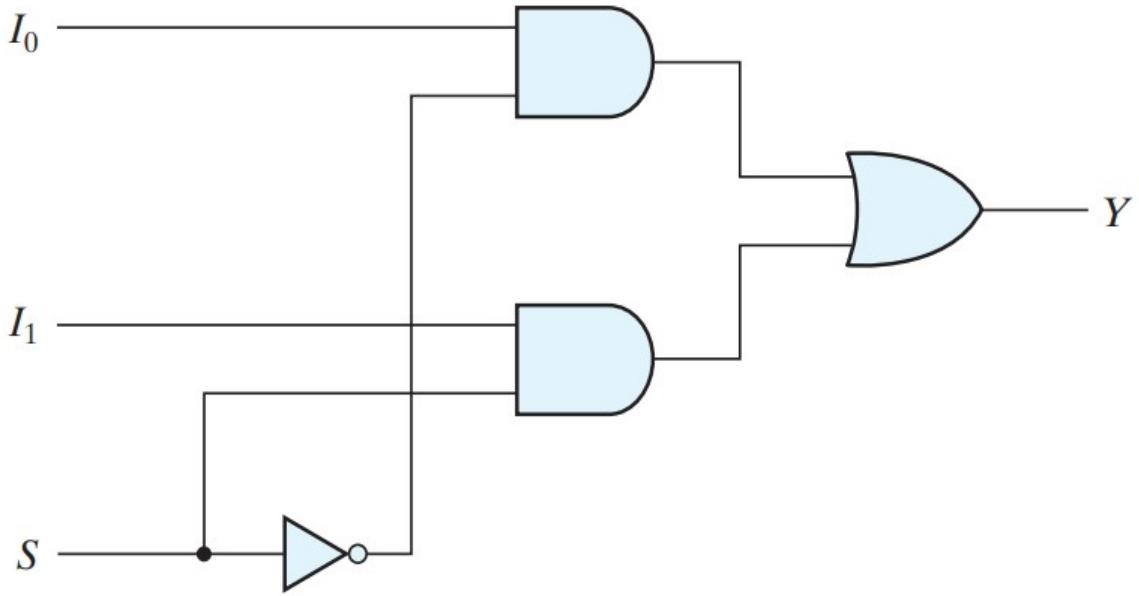
- Egy lehetséges elrendezést az alábbi ábrán mutatunk be



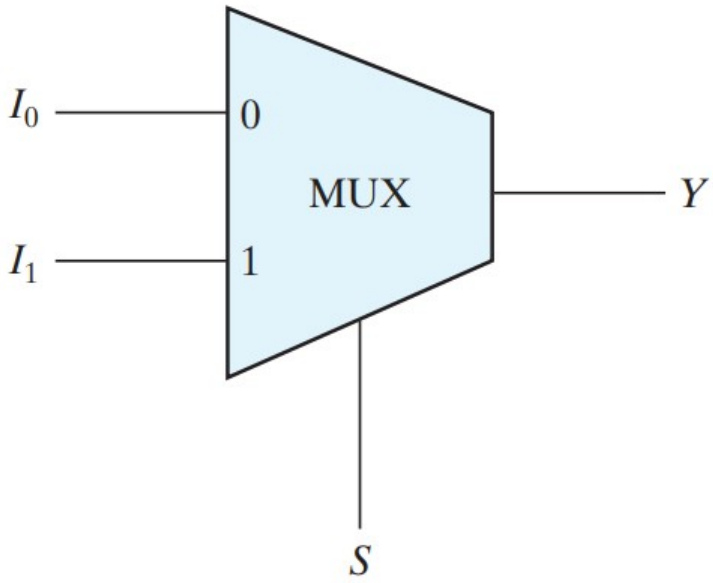
Multiplexerek és demultiplexerek

Multiplexerek

- **A multiplexer** (MUX) egy olyan kombinációs logikai hálózat, amely két vagy több bemenő jel közül a vezérlő jel alapján egyet kiválaszt és a kimenetére ad
- Az alábbi ábrán egy kétbemenetű multiplexer logikai diagramja és a blokkvázlatokban használt rajzjele látható
- Általánosságban 2^n bemenő vonal közül n vezérlő jellel választhatunk



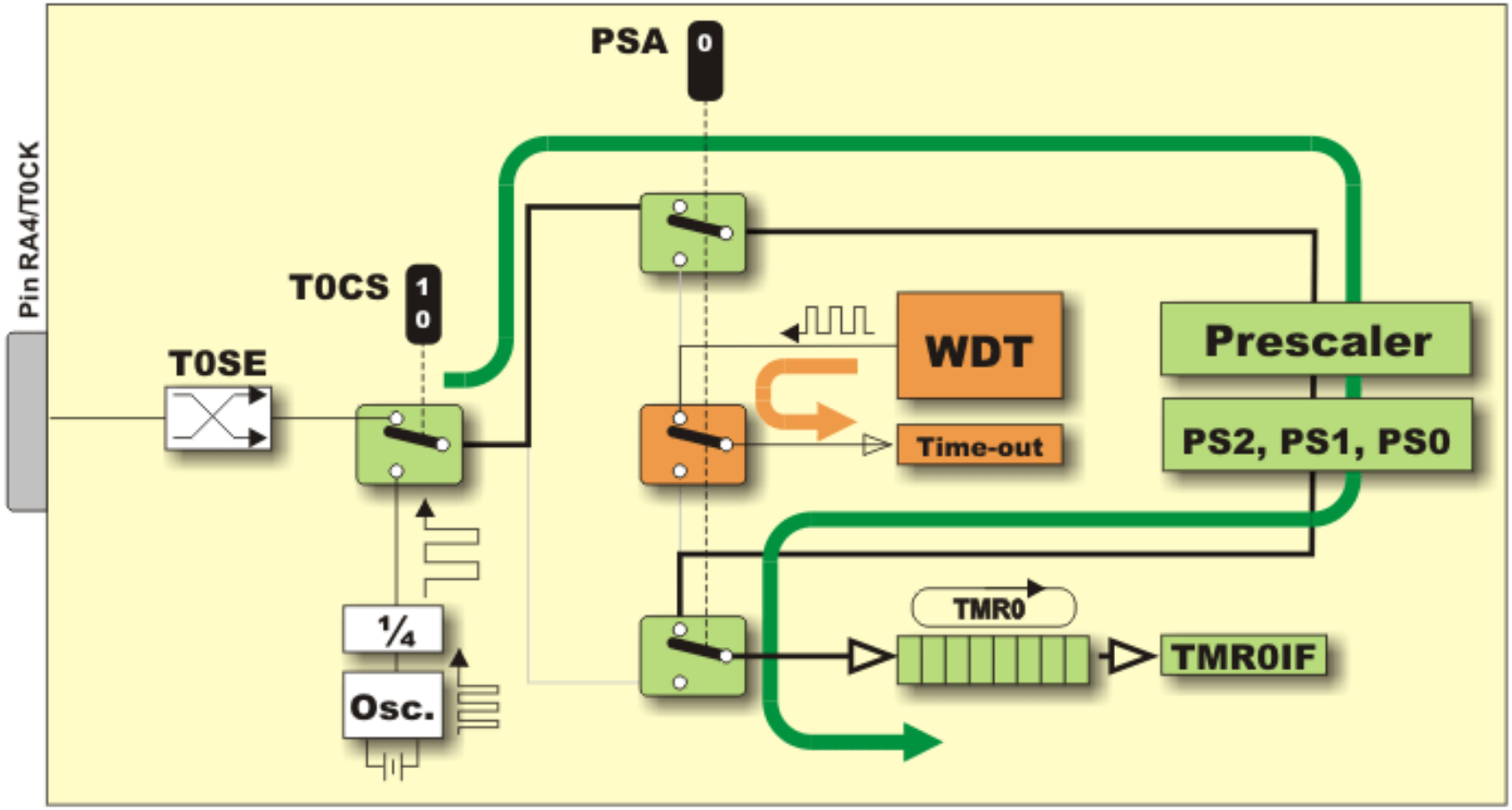
(a) Logic diagram



(b) Block diagram

Multiplexerek a gyakorlatban

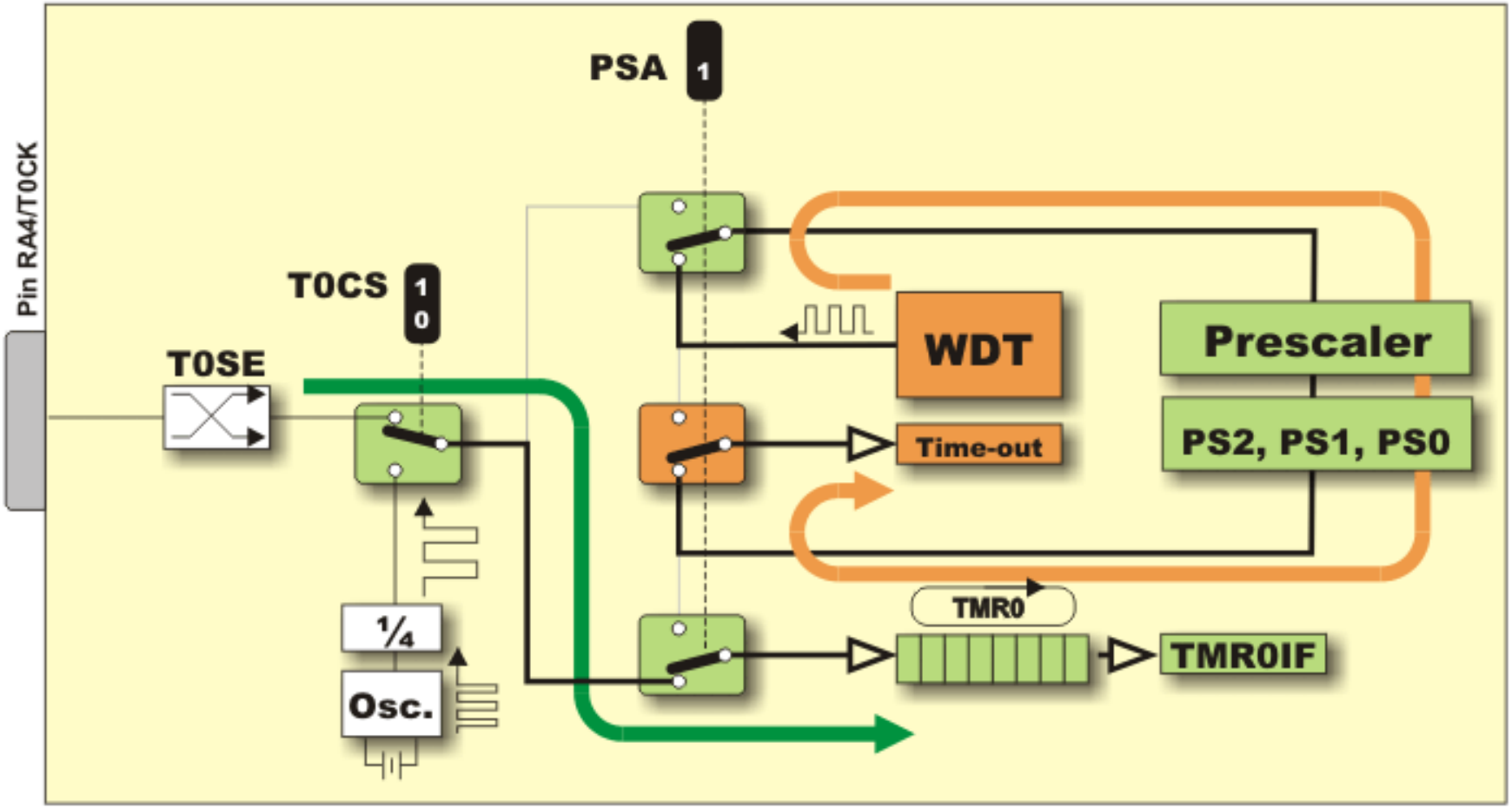
- A **PIC** mikrovezérlők **Timer0** időzítő/számláló bemenőjel választója is egy multiplexer, melynek vezérlő jele az **OPTION_REG** regiszter **T0CS** bitje
- Az órajel előosztó vagylagosan az időzítőhöz, vagy a Watchdog-hoz rendelhető, a 3 db multiplexer vezérlőjele az **OPTION_REG PSA** bitje



Forrás: www.mikroe.com

Multiplexerek a gyakorlatban

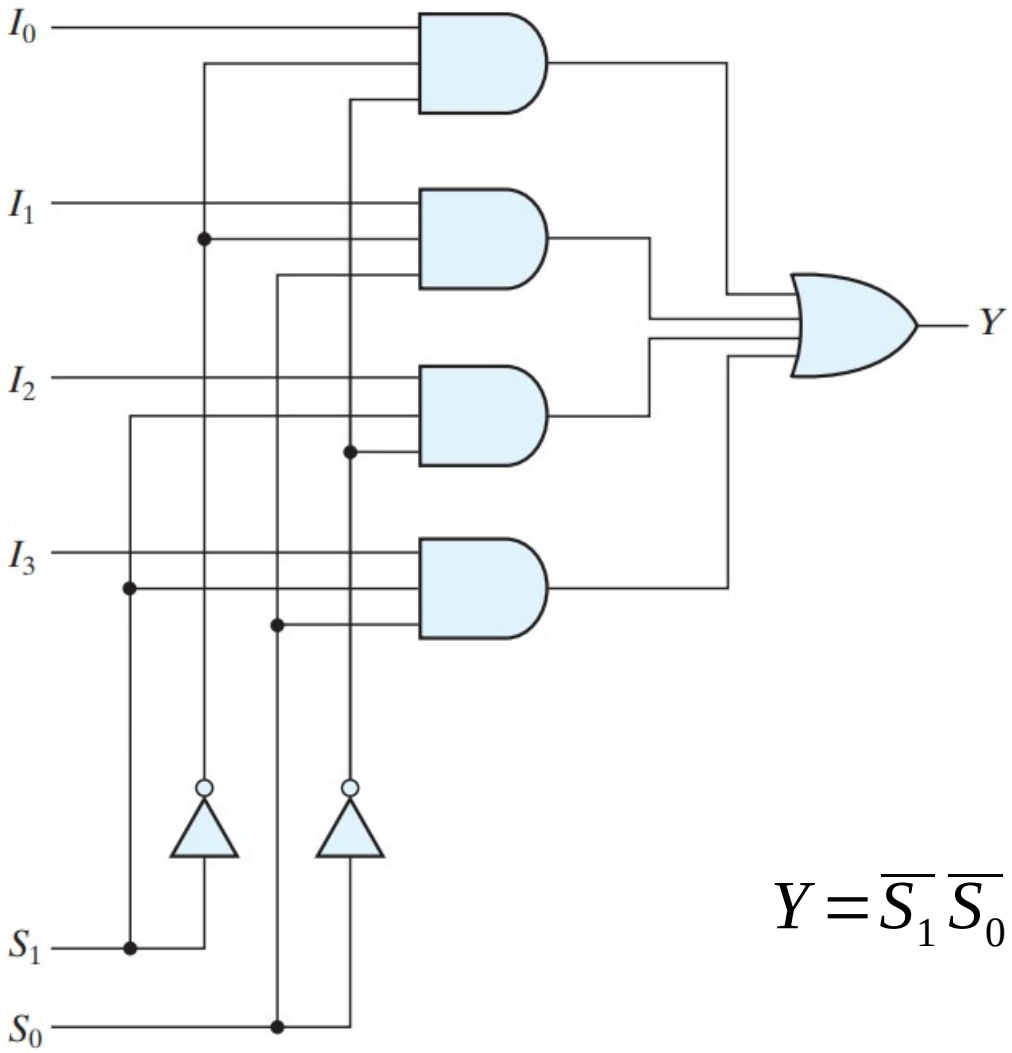
- Az **OPTION_REG** regiszter **PSA** bitjének 1-be állítása átkapcsolja a multiplexereket, s így az előosztó a Watchdog oszcillátorának jelét osztja le, **Timer0** pedig előosztás nélkül számlál



Forrás: www.mikroe.com

Négyről egy vonalra multiplexer

- Az alábbi kapcsolás négy bemenő vonal közül egyet választ ki az S_0 és S_1 kiválasztójelek segítségével és továbbít az Y kimenetre



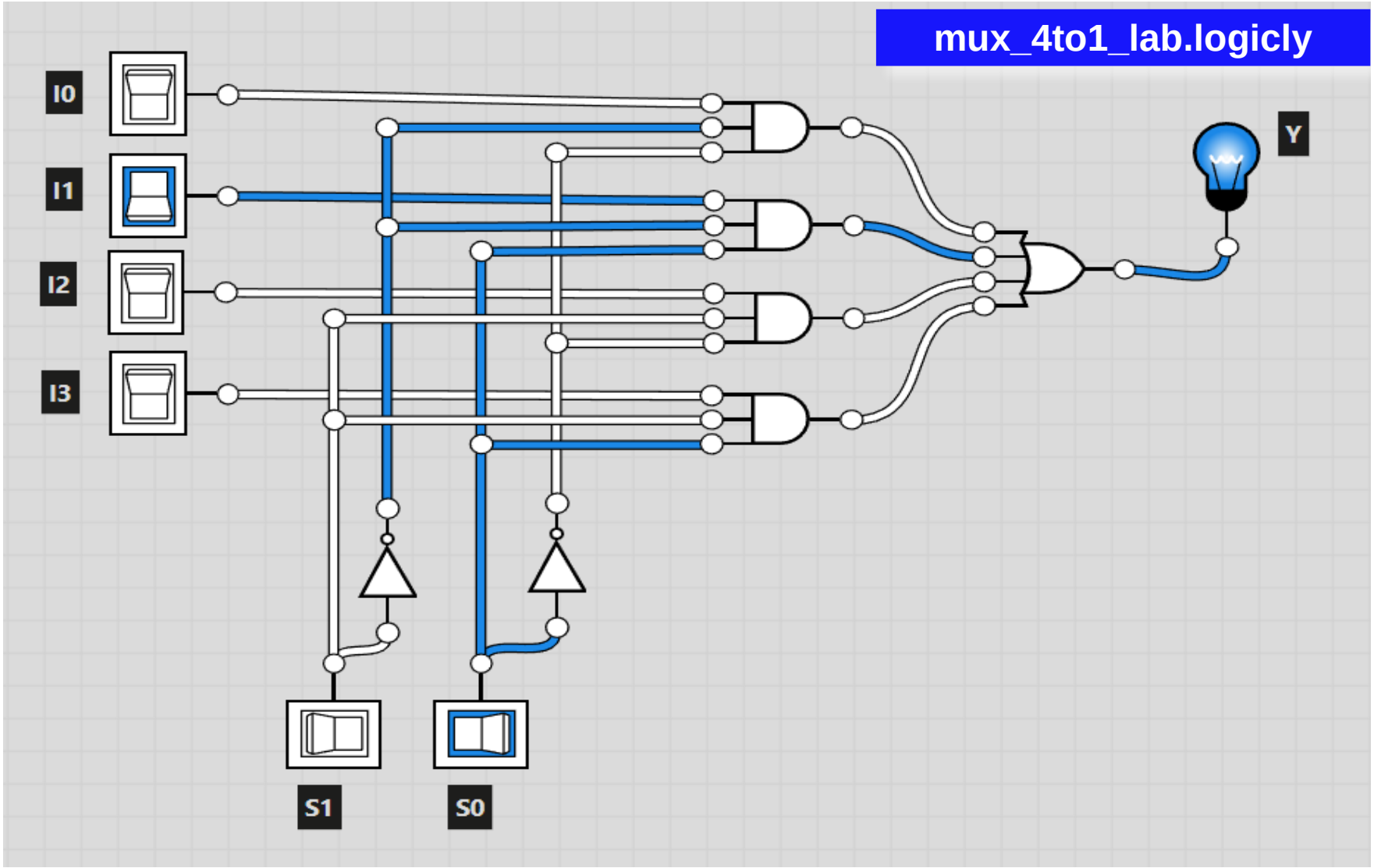
Funkciótábla:

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

$$Y = \overline{S_1}\,\overline{S_0}\,I_0 + \overline{S_1}\,S_0\,I_1 + S_1\,\overline{S_0}\,I_2 + S_1\,S_0\,I_3$$

Négyről egy vonalra multiplexer

- A kapcsolást a **Logic.ly** programban is kipróbáltuk és alkatrészként is elmentettük a **mux_4to1.logiclylib** fájlba



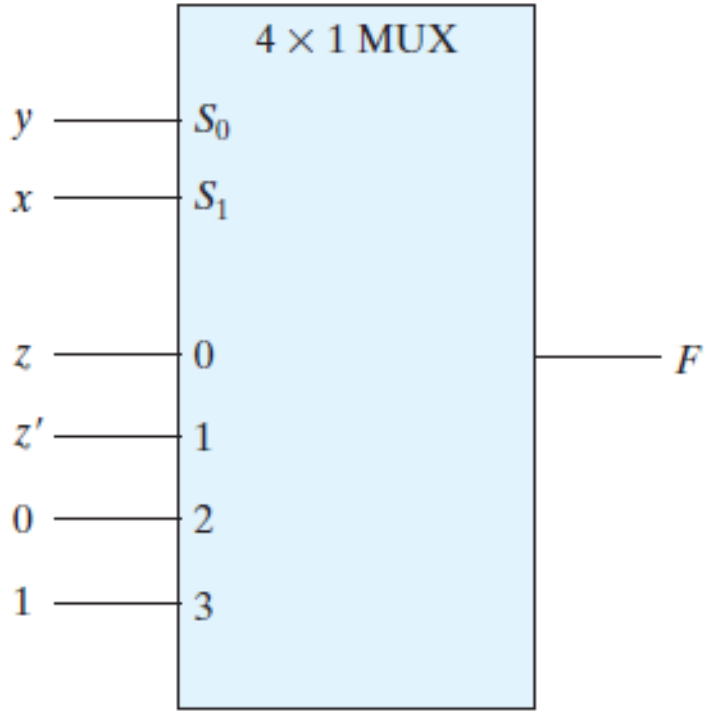
Logikai függvény megvalósítása multiplexerrel

Egy *n* változós logikai függvényt egy *n-1* bemenetválasztós multiplexerrel is megépíthetünk. Legyen például: $F(x, y, z) = \Sigma(1, 2, 6, 7)$

(***F*** oszlopában az 1., 2., 6. és 7. sorokban áll egyes)

A bemenetválasztók *x* és *y* lesznek, a négy bemenetre pedig az igazságtáblázat alapján *z, z', 0* és *1* kötendő

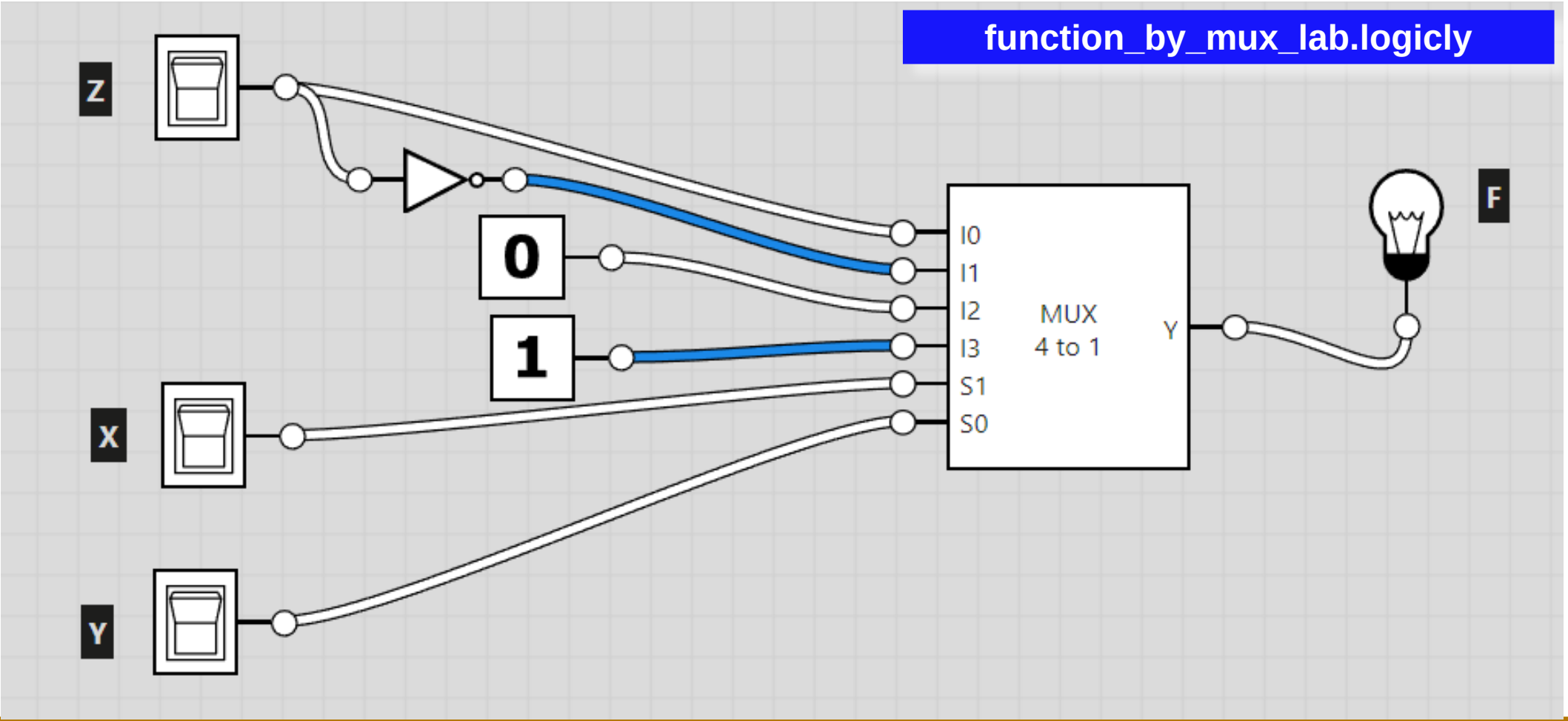
x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	



Vegyük észre, hogy az ***F*** oszlopban két-két soronként nézve a függvény értéke vagy megegyezik *z*-vel, vagy ellentétes vele, vagy csupa nulla, vagy csupa 1-es – más lehetőség nincs

Logikai függvény megvalósítása multiplexerrel

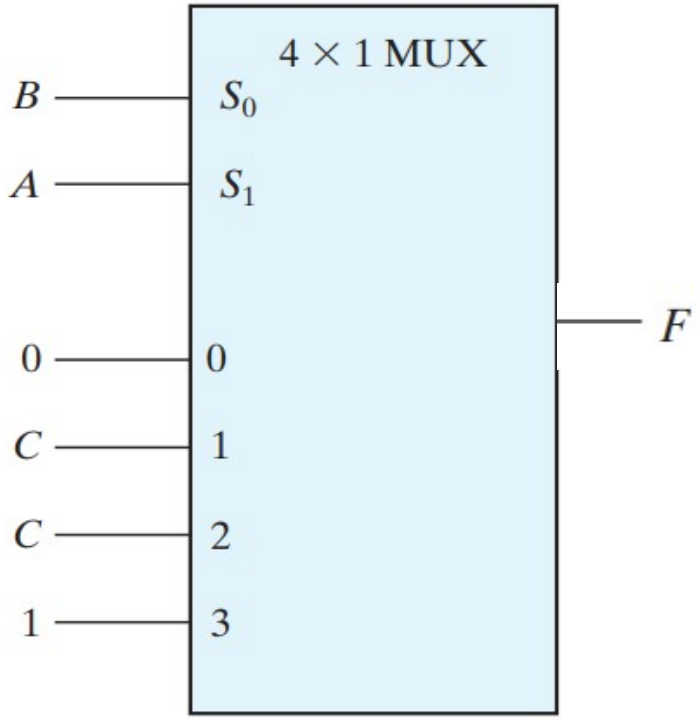
- A könyvtári modulként elmentett **mux_4to1.logiclylib** felhasználásával a a **Logic.ly** programban is megvalósítottuk a $F(x, y, z) = \Sigma(1, 2, 6, 7)$ függvényt



Logikai függvény megvalósítása multiplexerrel

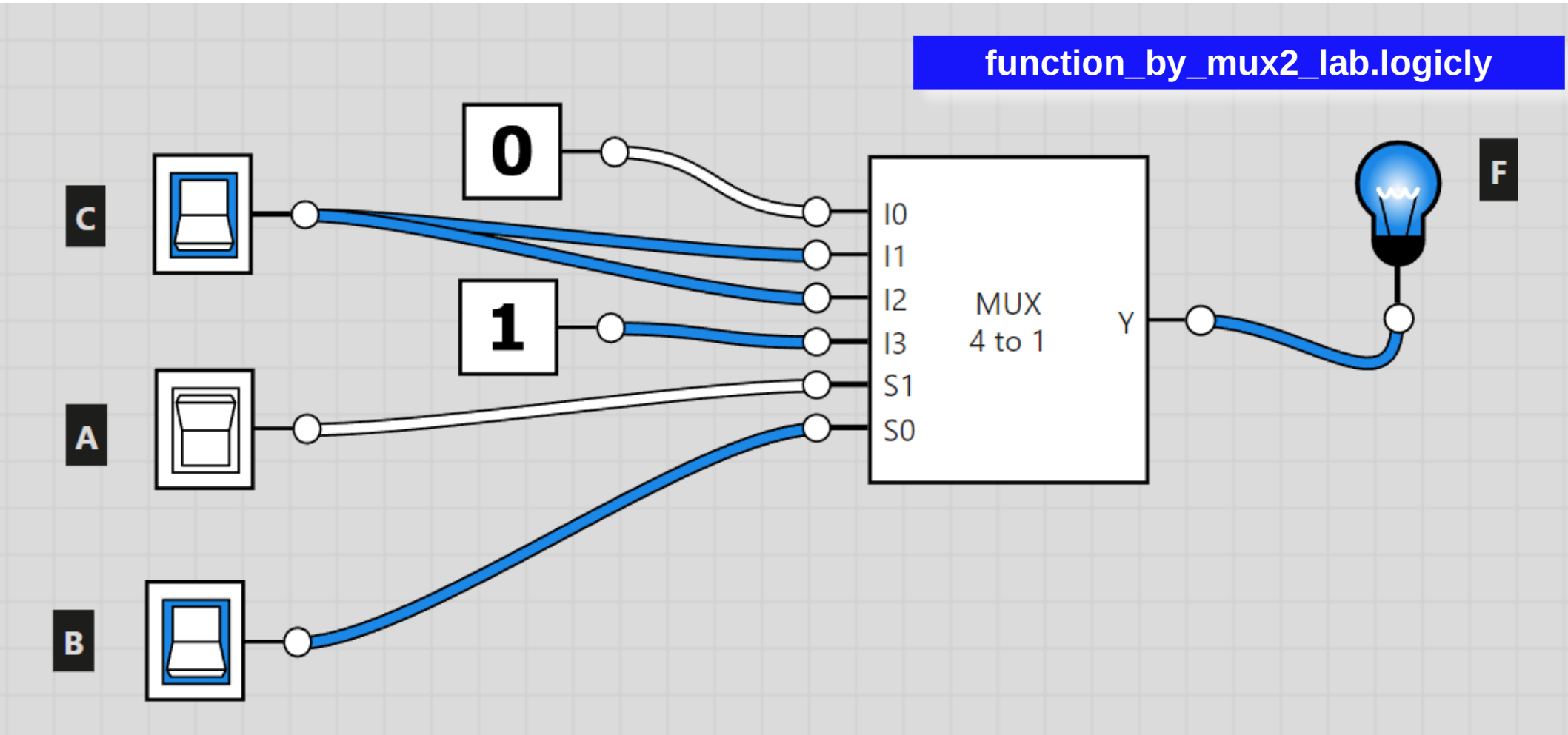
- **Nézzünk egy feladatot:** Valósítsuk meg az $F(A, B, C) = \Sigma(3, 5, 6, 7)$ függvényt egy 4 x 1 multiplexerrel!
  - Írjuk fel a függvény igazságtáblázatát!
  - Két soronként állapítsuk meg, hogy C, C', 0, vagy 1 írja le a függvényt!
  - Vegyünk egy 4x1 multiplexert és kössük be a fentiek szerint!

A	B	C	F	
0	0	0	0	F=0
0	0	1	0	
0	1	0	0	F=C
0	1	1	1	
1	0	0	0	F=C
1	0	1	1	
1	1	0	1	F=1
1	1	1	1	



Logikai függvény megvalósítása multiplexerrel

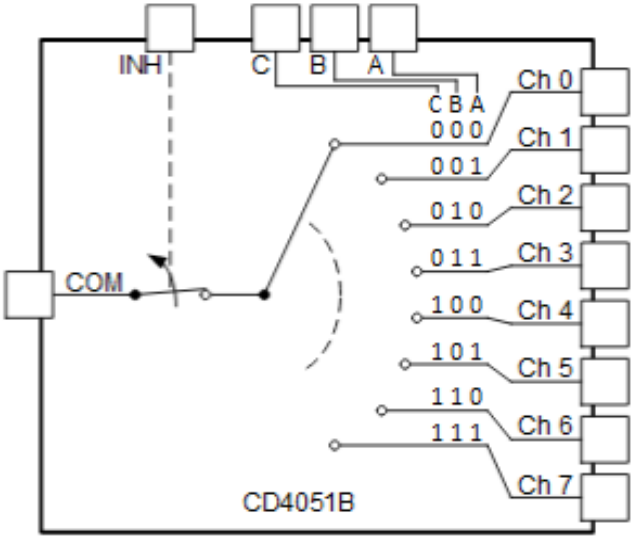
- A könyvtári modulként elmentett **mux_4to1.logiclylib** felhasználásával a a **Logic.ly** programban is megvalósítottuk a $F(A, B, C) = \Sigma(3, 5, 6, 7)$ függvényt

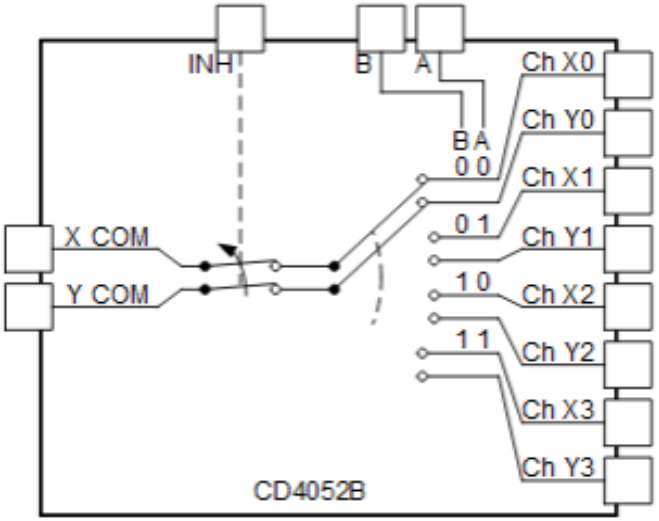


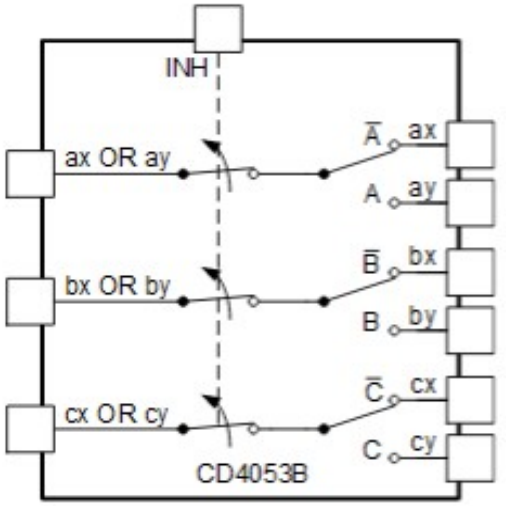
Analóg multiplexerek

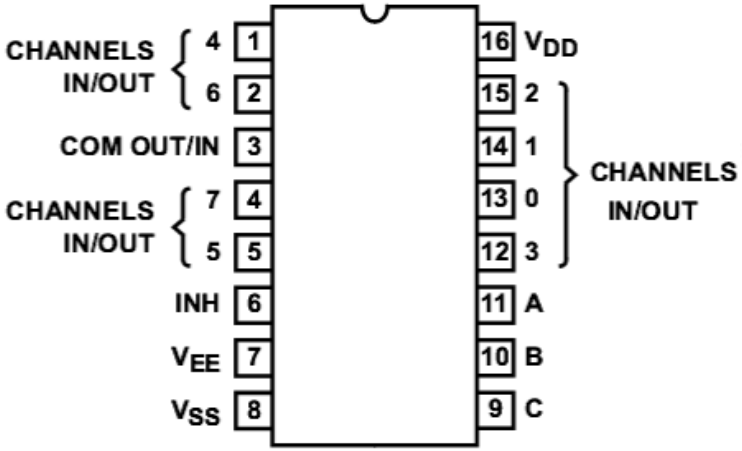
4051 CMOS Single 8-Channel Analog Multiplexer/Demultiplexer
4052 CMOS Differential 4-Channel Analog Multiplexer/Demultiplexer
4053 CMOS Triple 2-Channel Analog Multiplexer/Demultiplexer

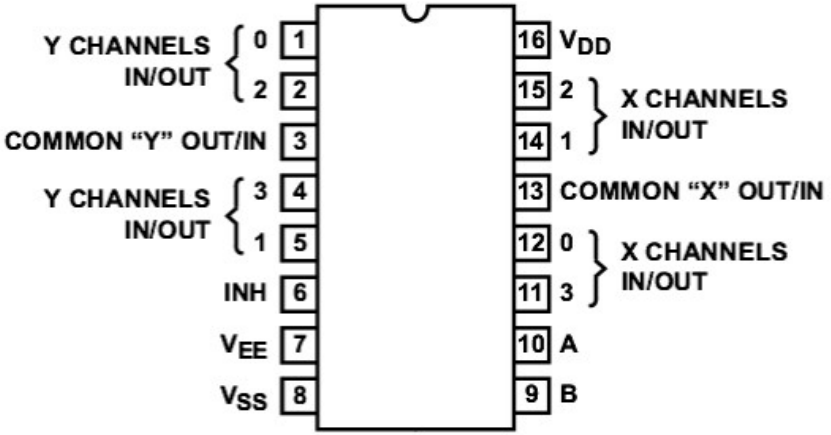
- A kiválasztás digitális, a csatorna viszont analóg jeleket visz át

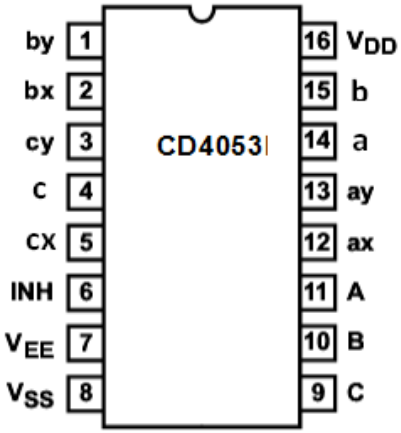






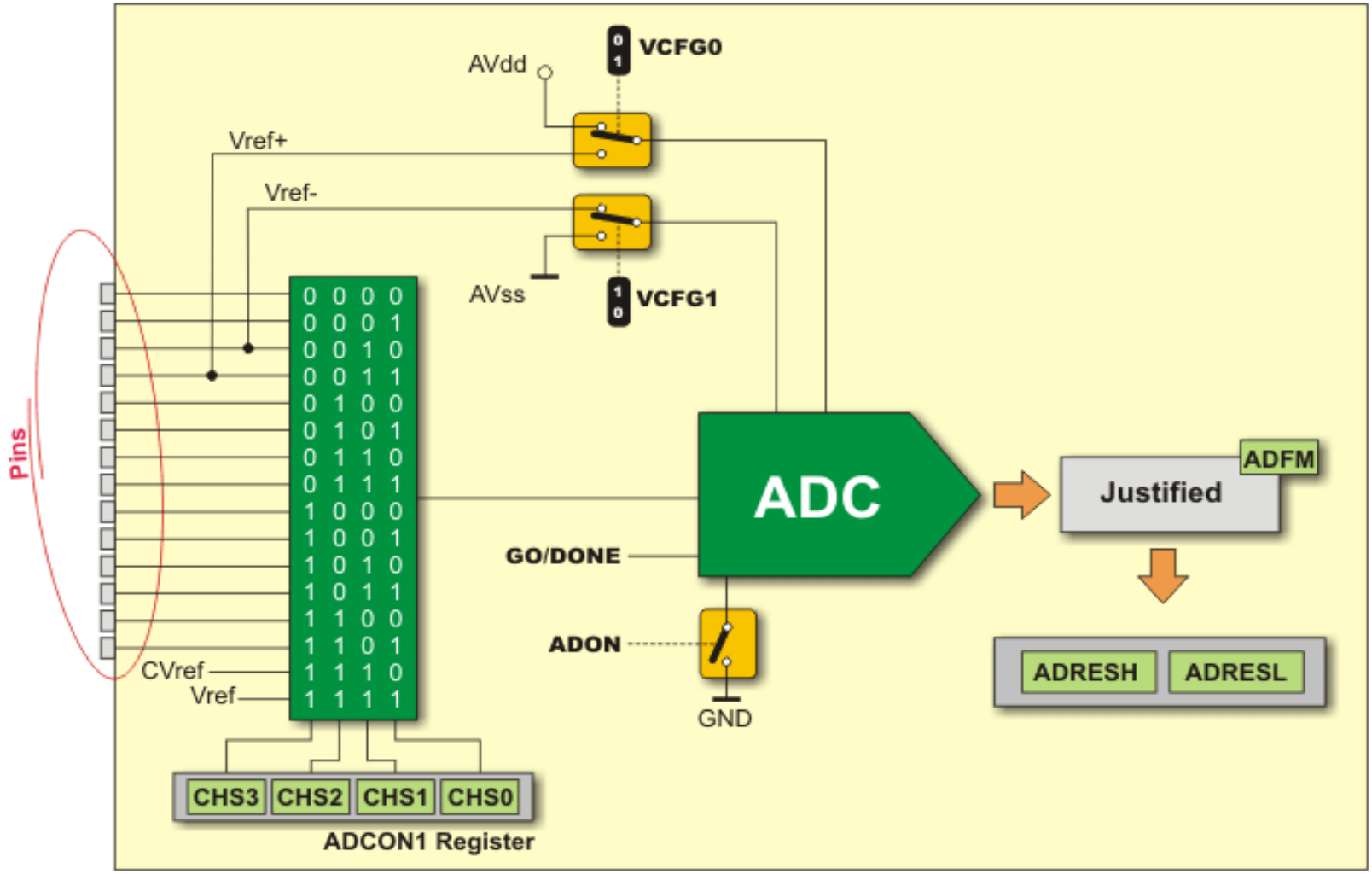






Analóg multiplexer a gyakorlatban

- A mikrovezérlők Analóg-Digitális átalakítója (ADC) több analóg bemenet közül választja ki azt az egyet, amelyen mérést indít

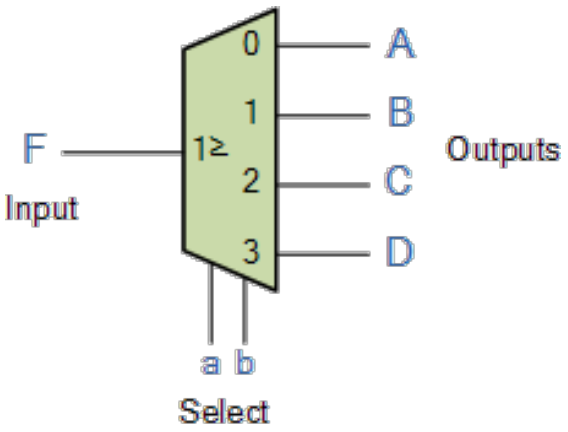


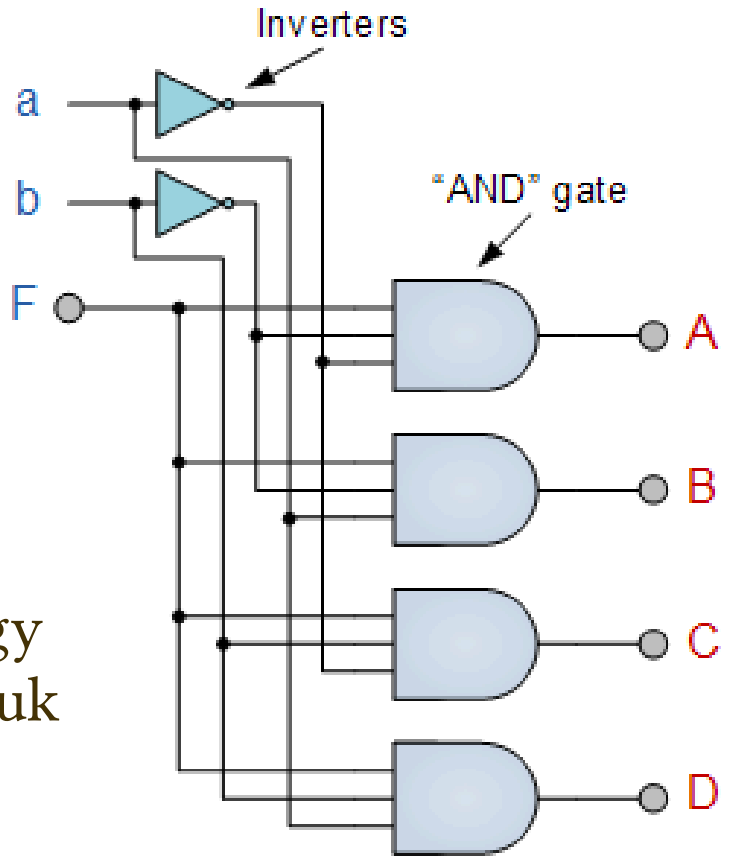
Forrás: www.mikroe.com

Demultiplexerek

- **A demultiplexer** (DMUX) egy olyan kombinációs logikai hálózat, amely egy adatbemenetet jelét a vezérlő bemenetek segítségével kiválasztott kimenetre irányítja

Blokkvázlat rajzjel



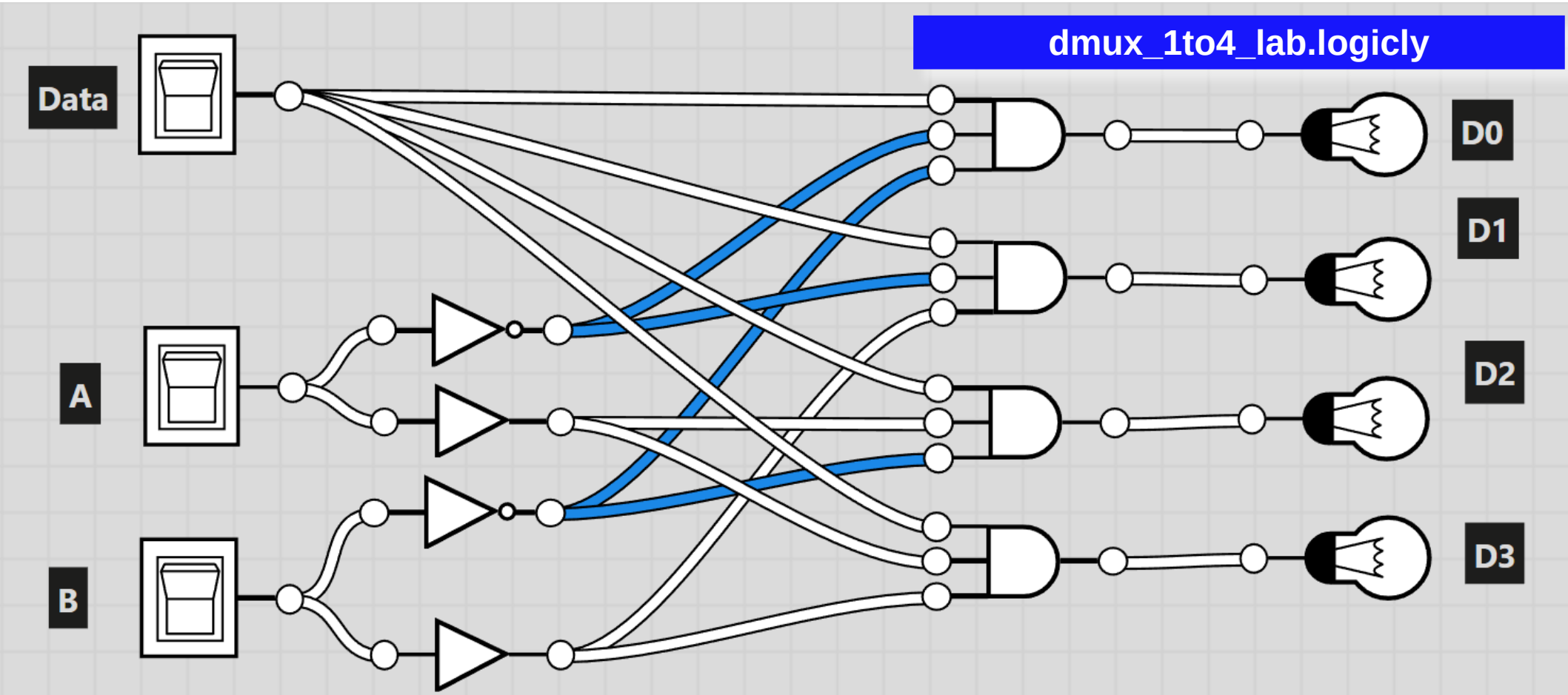


A jobboldali ábrán egy bemenőjelet a négy kimeneti csatorna valamelyikébe irányítjuk az ***a*** és ***b*** kiválasztó jelek beállításával

Forrás: electronics-tutorials.ws/combination/comb_3.html

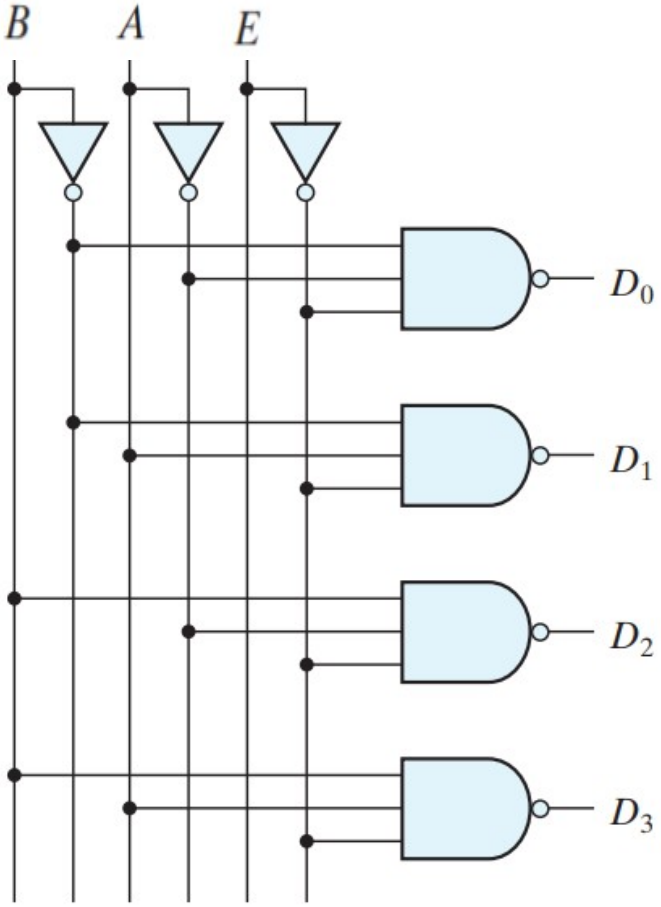
1 x 4 demultipexer

- Az 1 x 4 demultiplexert a **Logic.ly** programban is kipróbáltuk (a neminvertáló buffer kapuk elhagyhatók, itt csak az átláthatóbb huzalvezetést szolgálták egyszerű átvezetésként)



NAND dekóderből demultiplexer

- A hasonló felépítésnek köszönhetően a korábban bemutatott, engedélyező jellel rendelkező **NAND** dekódolóból is készíthetünk (negatív logikájú) 1 x 4 demultiplexert: az engedélyező jel bemenet lesz az adatbemenet, s arra a kimenetre jut el a ’0’, amelyiket az *A*, *B* bemenetek kiválasztják



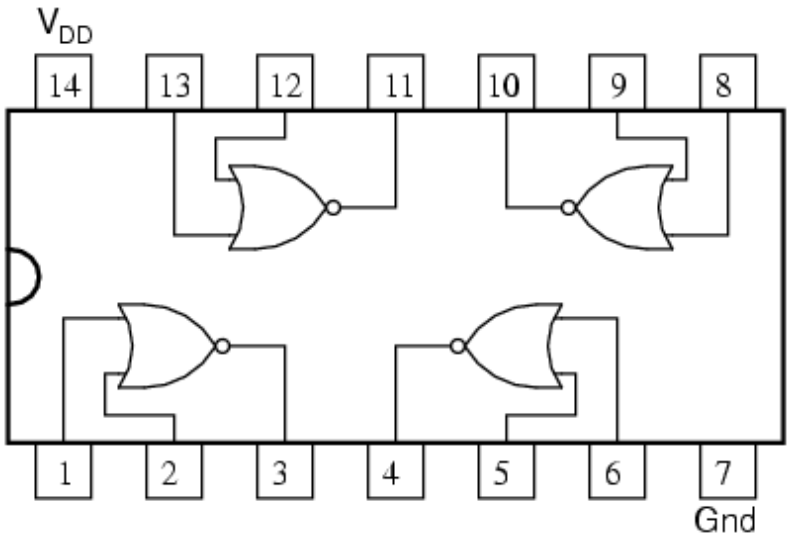
E	B	A	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

A 4000-es sorozat tipikus tagjai

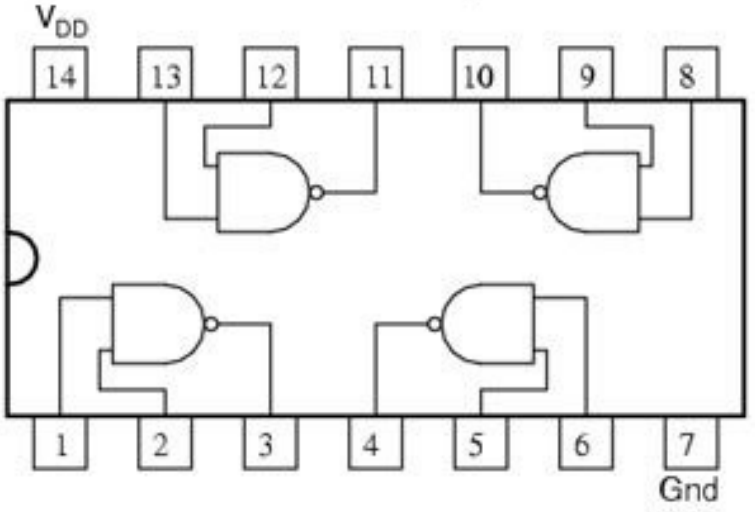
4001 CMOS Quad 2-Input NOR Gate
4011 CMOS Quad 2-Input NAND Gate
4013 CMOS Dual D-Type Flip Flop
4017 CMOS Decade Counter with 10 Decoded Outputs
4021 CMOS 8-Stage Static Shift Register
4022 CMOS Octal Counter with 8 Decoded Outputs
4023 CMOS Triple 3-Input NAND Gate
4025 CMOS Triple 3-Input NOR Gate
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

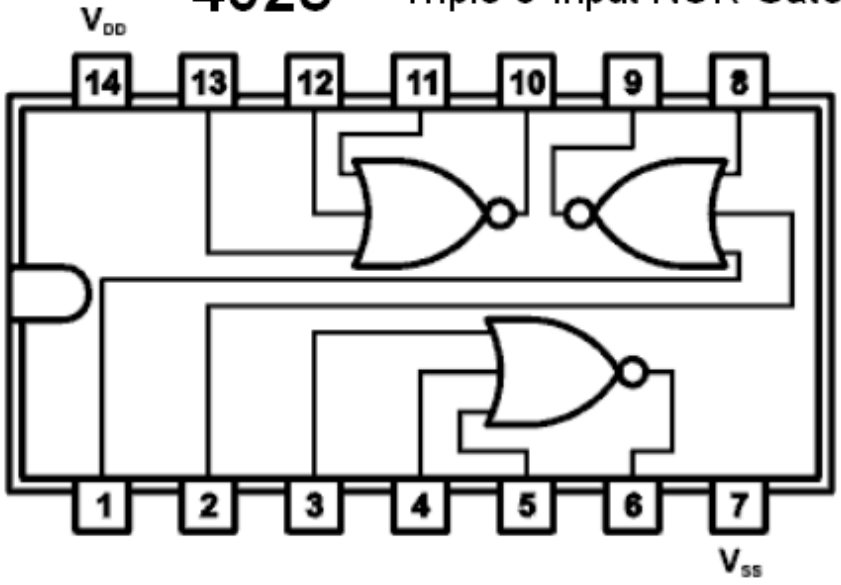
4001 Quad 2-Input NOR Gate



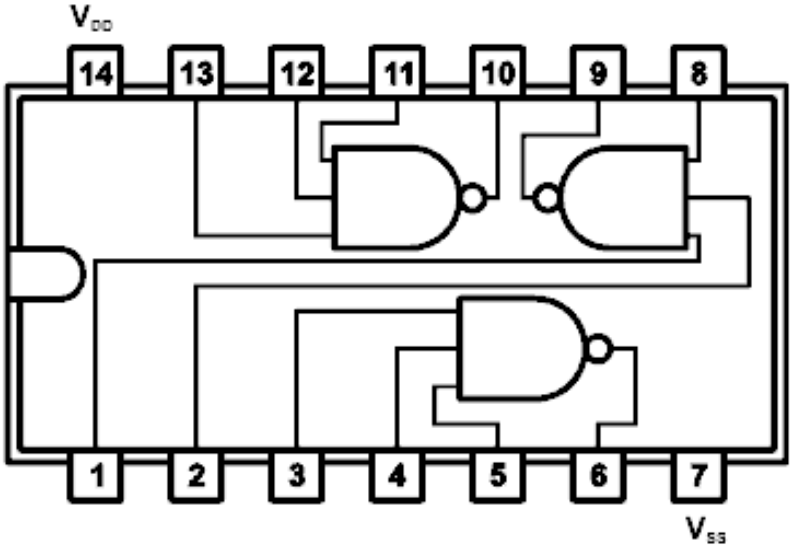
4011 Quad 2-input NAND



4025 Triple 3-Input NOR Gate

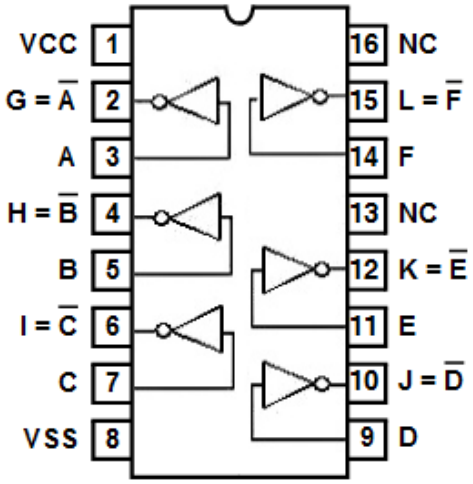


4023 Triple 3-Input NAND Gate

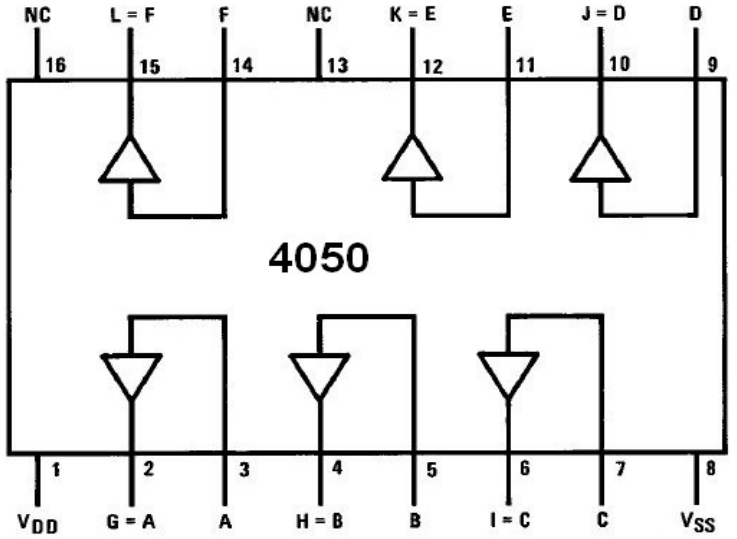


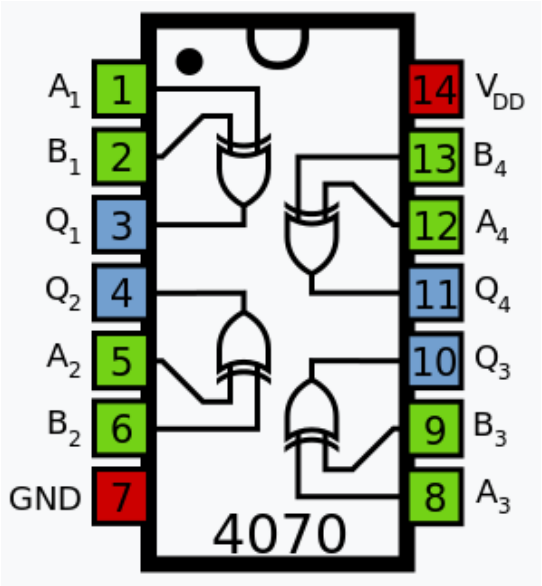
A 4000-es sorozat tipikus tagjai

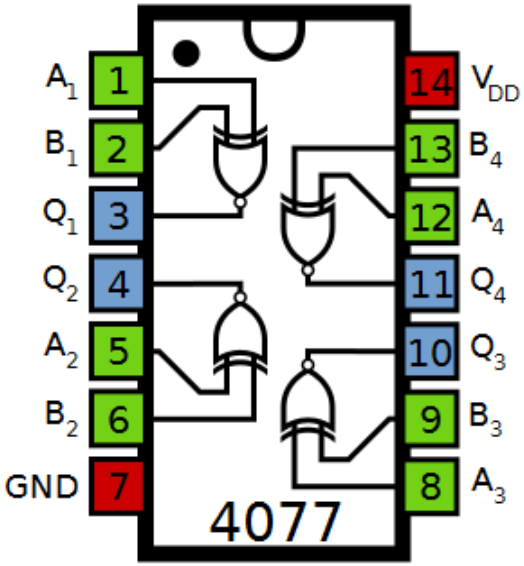
4049
Hex Inverting Buffer/Converter

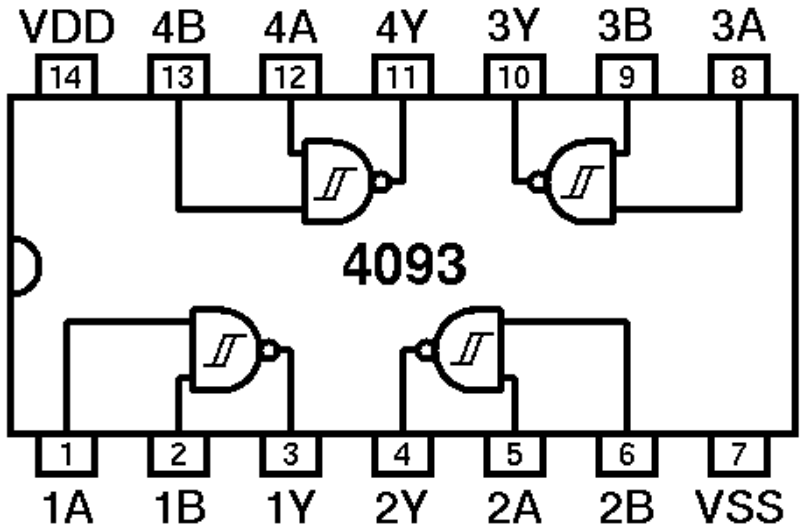


4050 Hex Non-Inverting Buffer/Converter









A 4000-es sorozat tipikus tagjai